資料2

第5世代のコンピュータ

データフローマシン／データベースマシンの調査研究

昭和57年2月

財団法人 日本情報処理開発協会
社団法人 日本電子工業振興協会

この報告書は、日本自転車振興会から競輪収益の一部である機械工業振興資金の補助を受けて昭和56年度に実施した「第５世代電子計算機に関する内外技術動向調査」の成果をとりまとめたものであります。

序　　文

この報告書は，財団法人　日本情報処理開発協会からの委託により，社団法人　日本電子工業振興協会が昭和５６年度事業として実施した「第５世代コンピュータ基幹技術に関する調査研究」のうち「データフローマシン／データベースマシンの調査研究」の結果をとりまとめたものである。

この調査は，１９９０年代に実用化を目標とした第５世代コンピュータの基幹技術であるデータフローマシン／データベースマシンについて，各パイロットシステムの基本設計に必要な基礎データの収集分析ならびに開発のための基本システムをとりまとめたものであり，「技術調査委員会」（委員長　東京大学工学部電気工学科助教授　田中英彦氏）を設置して実施した。

調査の実施にあたり，ご指導，ご協力いただいた関係官庁，関係各位ならびに直接労を賜わった委員各位に深く感謝の意を表わすとともに，この報告書が有益に活用いただければ幸いである。

昭和５７年２月

＜本調査で作成した報告書・資料＞

- 第５世代のコンピュータ研究開発計画
- 〃　　〃　　データフローマシン／データベースマシン
- 〃　　〃　　ロジックプログラミング
- 〃　　〃　　波及効果
- 〃　　〃　　関連技術動向調査
- 〃　　〃　　研究開発計画・付属資料

データフローマシン/データベースマシン技術調査委員会

（敬称略　順不同）

委員長	田中英彦	東京大学	工学部電気工学科　助教授
委員	雨宮真人	電電公社武蔵野電気通信研究所	基礎研究部第一研究室　調査役
〃	田中　譲	北海道大学	工学部電気工学科　講師
〃	山本昌弘	日本電気(株)	C&Cシステム研究所 コンピュータシステム研究所
〃	門脇吉彦	(株)日立製作所	神奈川工場CPU設計部主任技師
〃	滝沢　誠	(財)日本情報処理開発協会	開発部　係長
〃	相馬行雄	(株)富士通研究所	システム研究所　主任研究員
〃	伊藤徳義	沖電気工業(株)	総合システム研究所通信処理研究部 コンピュータアーキテクチャ研究室
〃	竹内彰一	三菱電機(株)	中央研究所システム第2グループ
〃	島田俊夫	電子技術総合研究所	電子計算機部計算機方式研究室
〃	喜連川　優	東京大学	工学部電気工学科
〃	後藤厚宏	東京大学	工学部電気工学科
〃	宮地泰造	三菱電機(株)	情報電子研究所情報処理開発部 ソフトウェアグループ
〃	来住晶介	沖電気工業(株)	総合システム研究所通信処理研究所 コンピュータアーキテクチャ研究室
協力者	長谷川隆三	電電公社武蔵野電気通信研究所	基礎研究部第一研究室調査員
事務局	菊池　瑛	(社)日本電子工業振興協会	開発部　部長代理
〃	清　紹英	(社)日本電子工業振興協会	電子計算機担当　課長代理
〃	小島謙二	(社)日本電子工業振興協会	電子計算機担当

はじめに

本報告書は，昭和５６年７月，(財)日本情報処理開発協会から(社)日本電子工業振興会へ委託された「データフローマシン／データベースマシンに関する調査研究」の報告書である。

この委託調査は，昭和５７年度から始められる第５世代コンピュータプロジェクトに反映するための研究開発計画の基礎調査として進められたもので，第５世代コンピュータシステムの基幹技術に関する先行研究の１つとして位置付けられる。

第５世代コンピュータシステムの目的は，知識情報処理システムを実現することである。人間が，過去に蓄積した膨大な知識を基にして新しい局面に対応し，またその経験を新しい知識として蓄えてゆくように，コンピュータにも同様の能力を持たせようというのである。これによって，従来のコンピュータで問題とされて来た多くの難点を克服することが期待されている。

このようなコンピュータは，膨大な知識を能率良く蓄え必要に応じて取り出す機能と，取り出した知識を活用して推論を行う機能とからなると考えられる。前者は知識ベースマシン，後者は推論マシンと呼ばれているがこれらのマシンを支える基本的なハードウェア機構は何であろうか。それは現代のコンピュータの延長線上にあるものであろうか，それとも全く異なった新しいものであろうか，これらの問いに対し，今完全な答えが用意されている訳ではない。それは今後の研究が進むにつれてより明確になってゆくべき性格のものである。しかし，この基本機構として今最も有望視されているもの，若しくは将来の基本機構の礎となる可能性が高いものを抽出することはできる。それは，関係データベースマシンと，データフローマシンである。関係データベースマシンは知識ベースマシンの基礎であり，データフローマシンは並列度の高い推論処理の基本機構であると考えられる。

本調査研究は，このようなデータフローマシンとデータベースマシンを対象とし，それらの検討を行うことを目的とする。具体的には，われわれはこの調査研究の目的を次のように設定し作業を進めることとした。

1. これら両マシン研究の現状分析
2. 両マシンの諸技術の検討と整理
3. マシン実現上の問題点の抽出
4. 実現諸技術の提案
5. プロトタイプマシンのイメージ作成
6. 実験機第１版の仕様作成
7. 前期研究開発作業計画の立案

8. 研究開発用人員の推定，支援環境の明確化

調査研究に携わった者とその役目分担は以下に示す通りである。

田中英彦	主査
雨宮真人	DFMグループリーダ
田中　譲	DBMグループリーダ
山本昌弘	DBMグループ
門脇吉彦	DBMグループ
滝沢　誠	DBMグループ
相馬行雄	DFMグループ
伊藤徳義	DFMグループ
竹内彰一	DFMグループ
島田俊夫	DFMグループ
喜連川優	DBMグループ
後藤厚弘	DFMグループ
宮地泰造	DBMグループ書記
来住晶介	DFMグループ書記
長谷川隆三	DFMグループ・協力者
清　紹英	事務局

以上の他，電総研の植村俊亮氏，日本電気の日吉茂樹氏，富士通の牧之内顕文の各氏には，この調査研究の初期段階でのヒアリングに御協力いただき貴重な御意見をたまわった。

この調査研究委員会は，昭和56年7月末より，昭和57年2月まで，ほぼ6カ月足らずの間，月2回程のペースで作業を進めた。始めてから10月の始めまでは，この作業の目的をしぼり，両マシンの研究現状についての認識を深めて検討を必要とする項目をしぼり込むことを行った。この間は，全員集まって作業を進めた。11月から2月の始めまでは，両マシンの技術検討，プロトタイプのイメージ作り，研究開発計画の立案等を行ったが，作業の時間を多くする意味から，データフローマシン，データベースマシンのグループに分かれ並行して作業を行った。何回かの会合は時間を多く取り，まる1日を費している。これらの並行作業では各グループリーダがまとめ役となり検討を進めた。主査は両グループに参加した。毎回の検討会では常に全員が作業結果を持ち寄って相互に検討し合い，次回までの宿題を持ち帰るというパターンで進めたため，実際に要した全時間はかなりになるであろう。

このようにして進めた検討の成果がこの報告書である。なにぶんにも短期間で作業を行ったため，細かな諸点に検討洩れの箇所や，検討不十分な点が見受けられることは否めない。しかし，この検討に携わった者はそれぞれの立場における忙しさにもかかわらず皆非常に熱心に作業を行った。この報告書が将来のコンピュータ技術の研究に役立てば幸いである。

目　　次

第Ⅰ部　データフローマシンの研究

1. 概　　要

知識情報処理システムにおける重要な機能の1つは推論機能である。これは，膨大な知識の中から関連のありそうな知識を見出すためのパターンマッチ機能としての統合化機能と，推論を進めて行く上で立てた仮説の失敗にもとづく自動的な後戻り機能を特徴とする。従って最終結論に歩り着くまでに，多くの試行錯誤を繰返しており，複雑な推論を実行するには一般に非常に多くの処理を必要とする。また，その処理は非決定的であり前似って定め難い。このような処理を従来のコンピュータの上で実行するとその逐次性の故に多くの時間を必要とし，また制御の記述が複雑になる。しかし，処理自体は本質的に多くの並列性を含んでおり，それらをハードウェアの都合上直列に実行することは，そのための制御を複雑にし処理速度を遅くこそすれ，何ら本質的なことではない。このような処理に向いた計算機構，それがデータフローマシンである。

データフローマシンは，データの依存関係による自然な操作の流れを基としており，関数型言語がその基本と考えられるので，プログラム作成上プログラマに与える負担が軽減され，ソフトウェア生産性を向上できる可能性もある。

本検討グループは，このようなデータフローマシンを実現するための基礎検討を行うことを目的とする。すなわち，データフローマシンの諸技術を整理して問題点を明らかにすること，それらの問題点を検討し幾つかの解決策若しくはそれを見出すための手順を示して研究計画を立てること，出来れば目指すべきデータフローマシンのプロトタイプイメージを示し，それに至るための実験機第1版の仕様を固めること，またそのための研究開発用人員，支援環境を推定すること等である。

このデータフローマシンの検討報告書は7章からなる。第2章は，データフローマシン研究の背景について記述しており，知識情報処理システムからの要請や現代のコンピュータシステムの問題点について述べている。また，計算モデルについての一般論やデータフローマシンの一般説明等が為されている。計算モデルの研究から見れば，制御フロー，データフロー，リダクションという分類が考えられ，リダクションマシンも検討すべき1つのアーキテクチャではあるが，その研究は未だ多くなく能力として不明な所も多いので今後の進展に待つことにし，以下，データフローを中心に検討を行っている。

第3章はデータフローマシンの現状についてのサーベイである。まず，言語についてのサーベイで，関数型言語，単一代入言語等データフローマシンに関係の深い言語の他，データフローマシン用として提案されているＶＡＬ，Ｉｄ，ＶＡＬＩＤ 等の言語の説明がある。次に各所で行われているデータフローマシンとリダクションマシンの研究開発の現状についてのサーベイがあり，これら両マシンの関連についても触れている。

第4章はデータフローマシンの諸技術を検討したものである。まず，データフローマシン用高級言語に関しては，従来の言語の問題点が如何に解決されるか，またデータフローマシン用言語一般の問題点とその対策について述べられている。次に，データフローマシンの制御方式と基本操作の検討では，発火したアクティビティの検出機構，手続き，ループ，条件式等の制御，基本構成要素等が述べられる。基本構成要素としては，アクティビティメモリ，処理実行ユニット，構造メモリ，入出力制御装置と，これらを結合するネットワークを考えており，これらそれぞれについてその構成法，問題点，等に関して述べられている。知識情報処理向きのデータフローマシンでは，特に構造データに対する取扱い能力が優れている必要があり，構造を蓄えそれに対する操作を受持つ構造メモリは1つのキー要素である。また，データフローマシンはその性質上，多くの並列動作要素を持っているため，それらの間の能率良い通信機能は全体の能力を左右する重要な要素である。

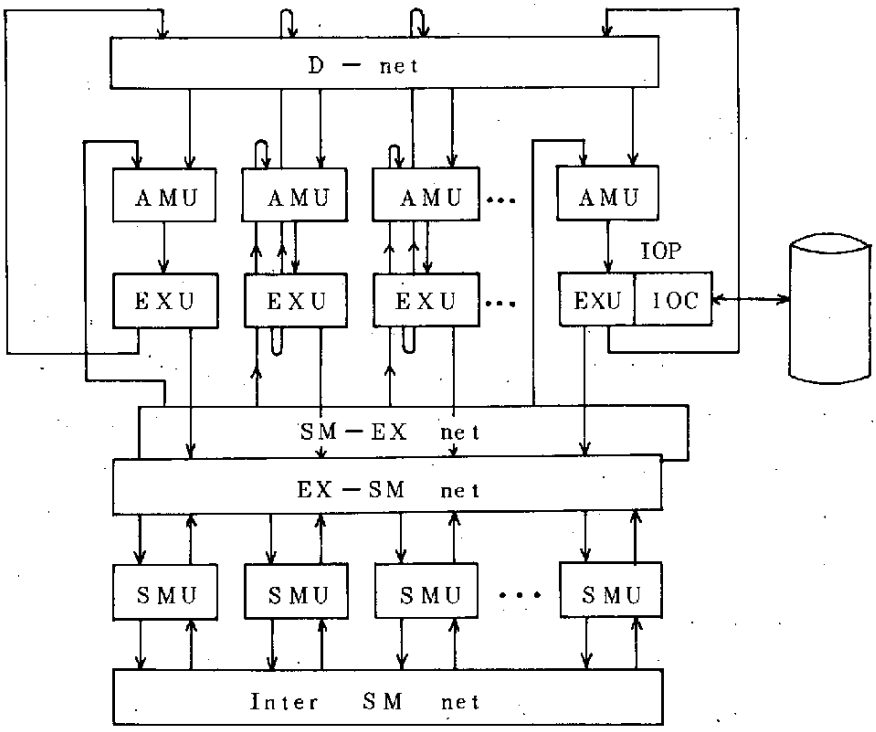
従来のコンピュータにおけるオペレーティングシステムに相当する機能に関しては未だ殆んど研究が為されていない状況であるが，やはり重要な技術である。特に，データフローマシンにおける入出力の扱い，発火したアクティビティを多くの処理実行ユニットのどこで処理するかを決定するアクティビティ割り付け方式，マシンの信頼性を維持するための誤り検出，再試行，システム再編成方式等はその中でもまず検討されねばならない項目であり，これらについての検討が述べられている。

次に，データフローマシンを実装する場合当然問題となるVLSIによる実装方式の検討，多くの並列度を含むデータフロープログラムや，データフローマシンを如何に能率良くデバッグするかの検討も為されている。特に後者は，データフローマシンの可能性を左右する重要なキーポイントとなる可能性があり，今後十分検討する必要がある。

第5章は，以上の検討結果を踏まえて，データフローマシンのプロトタイプとして提案するもののイメージを記述したものである。記述は実験機第1版と第2版という形で分けておこなわれているが，データフローマシンの基本構造は両者ともに不変である。すなわち，実験機第1版の目的はデータフローマシンの基本構造を試作実験を通して示すことにあり，データフローマシンとして重要な要素は殆んどその中に含まれている。高級言語としてはまず，汎用のデータフロー言語を設定し，基本命令も汎用の命令セットになっている。ハードウェアとしては，アクティビティメモリ，処理実行ユニット，構造メモリがそれぞれ数台ずつ設置され，それらが結合ネットワークを通して接続された形をとっている。マシン内を流れるトークンの形式としては，ジョブ，カラー等の識別用のタグフィールドが長い形になっており，制御の様々な方式実験が行いやすいようになっている。また，アクティビティ制御にはハッシングを用い，各機能要素の制御にはマイクロプログラムを用いて制御の融通性を確保している。実装は，現在入手できる程度のLSIやTTL，ICを用いるものとしており，特に高速性をねらう訳ではない。1つのアクティビティ制御に数μs程度要し，総合性能4MIPS程度が目標で

ある。実装すべきオペレーティングシステムは，実験のために必要最小限なものに限られている。

以上のような実験機の構成図を図1.1に示す。マシンの規模としては，図中のAMU，EXU，SMU等の台数がそれぞれ８～１６台程度のものを考えており，そのメモリ容量はAMU合計4MB，SMU合計8MB程度のものである。



AMU ： Activity Memory Unit

EXU ： Execution Unit

SMU ： Structure Memory Unit

IOC ： Input／Output Controller

D-net : Distribution Network

SM-EX net, EX-SM net

Inter SM net ：結合ネットワーク

図1.1　DFM実験機の構成

この実験機第１版での成果を基に作られる実験機第２版は，様々な項目の拡張・発展を目的としたものである。従って，基本構造は変わらず，速度の向上，並列度の向上，記憶容量の向上等の数量的な拡張やＲＡＳの向上等実用化技術の開発が外目には中心となる。ＶＬＳＩ実装を前提としたマシンとし，第１版における制御方式の検討結果を踏まえ，各要素の制御部はハードウェア化される。さらに，知識情報処理の核言語の開発に合わせてそれに向いた命令セットの実装，例えば統合化(unification)向き命令セット等が設けられ，推論マシンとしての機能を備えることになる。

第６章は，以上のようなデータフローマシンを開発してゆくための研究計画を記述したものである。まず，研究課題を整理し，基礎研究，高級言語，応用，アーキテクチャ，制御ソフトウェア，実装法の６つに分類し，それぞれについて，何が課題となっているかそのポイントを詳述している。次に，それらの課題を明らかにしてゆくために設置すべきサブプロジェクトの一覧が示され，課題との関係が述べられている。その後には各サブプロジェクト毎に，その目標，研究ポイント，研究内容，研究計画等が詳しく述べられている。研究計画では，第５世代コンピュータプロジェクトの前期である１９８２～１９８４年までの３年間を対象とし，どういう順序で作業を進めるべきかが線表の形で述べられている。また，それら各サブプロジェクトの遂行に必要な研究人員，所要道具等も示されている。

サブプロジェクトとして取り上げられているものは，データフローマシンの理論研究，応用研究，実験機第１版の試作，性能評価とシミュレーション，それに幾つかの個別研究である。個別研究には，高級言語，アクティビティ制御方式，アクティビティ割当方式，演算ユニット，構造メモリ，結合ネットワーク，入出力制御方式等の研究が含まれている。

第７章は，研究開発計画のまとめであって，第６章の各サブジェクト毎に述べられている各作業計画がまとめられており，合わせて各作業間の関連が示されている。また，所要人員等についてもまとめて一覧表になっている。

ここに示した研究開発計画は，最小限の人員，最短長の研究期間である。現在のコンピュータの原型が最初に作られてから今のような形になるまでに４０年程が経っている。それとは全く新しい原理にもとづくシステムの現実的可能性を示すには，３年間は如何にも短かい。従って，前期３年間はこの基本的な原理の有効性・将来性を判断するための材料を提供するための基礎研究期間であり，ここで示した作業計画以外にも種々の形で並行して研究を進める必要があると思われる。

2. データフローマシン研究の背景と意義

2.1 知識情報処理からの要請

第5世代コンピュータシステムは知識情報処理を目的として開発される。知識情報処理では従来の情報処理システムとは異なり，推論・問題解決，知識ベース管理など高度の機能が要求されることになる。これらの高度化された機能を効率的に実現するためには従来の数値計算主体の逐次制御によるノイマン方式のアーキテクチャでは無理があり，並列処理を基本とする新しいアーキテクチャを開発することが必要となる。

従来の処理方式では予め定められた手順に沿って処理を進めて行くことで事足りたが，推論を主体とする処理方式では前もって処理手順が定められているわけではなく，コンピュータは求められた目標に向かって，自ら手順を模索，設定しながら処理を進めていくことが要求される。推論処理では一意的に目標に達することはまれであり，多くの試行錯誤を経て目標に到達することになる。即ち非決定性処理が必然となり，この非決定性処理を効率的にサポート出来るマシンアーキテクチャが要求される。

非決定性処理の方式としては，depth first strategy，とbreadth first strategyの2方式が考えられる。depth first strategyでは複数個ある解法（処理手順）のうちどれか1つを選んで，目標に到る（success）か，処理が進まなくなる（failure）まで処理を先へ進めてゆく。もしfailureの場合には処理を後戻りさせ，他の解法を選び直して同様の処理を進める。一方 breadth first strategy では複数個の解法を並列的に試行してゆく。 depth first strategyは逐次的実行に適しており，現在多くの人工知能プログラムはこの方式を用いている。

しかし知識情報の規模が大きくなり，高度の知識情報処理を行う場合には推論の深度及び適用すべき推論規則の数も多くなり，それだけ試行錯誤の回数も増大することになる。このように複雑，高度化した処理を逐次制御をベースとするdepth first strategyによっていたのでは有効な時間内で問題を処理することは不可能である。したがって多数の解法を（多数のプロセッサを用いて）同時並列的に試行させ，必要な解を早く見つけるbreadth first strategy を実現することが必要になる。

知識ベース管理においても知識ベース探索では大量の知識データの中から必要なデータを高速に探索して引き出してくることが要求される。従来のノイマン方式では実行制御部とメモリ部とのネックのために，知識ベース探索の高速化は困難である。知識ベース機能実現には関係データベース

機能などメモリシステム側に処理機能を持たせた高機能化メモリ方式が要求されることになるが，このような知識ベース管理ハードウェアと推論制御用ハードウェアを一体化し，高性能システムとして機能させるためには個々の機能を独立実行させる分散制御方式の確立が必要である。

知識情報処理で扱かうデータは個々の数値ではなく，記号とそれらの関係構造を表わすデータとなる。これは一般にパターンといわれる，推論処理にしろ，知識ベース管理にしろ，パターンの探索，照合，パターン構造の解析等が処理の基本操作となる。これらの操作をサポートする演算装置はノイマン型マシンにおけるＡＬＵ等に比べはるかに高度である。特にパターンの照合，探索を効率的に行うためにはこれらの演算装置とパターン格納用メモリとが一体化される必要があり，このメモリ管理と推論実行制御を融和させ高速処理を実現するには集中制御を基本とするノイマン方式は不都合である。

最近ＶＬＳＩ技術の進展が著しく，今後さらに進展をとげるものと思われる。上記に述べた並列処理，分散制御を実現するための素子技術がVLSI技術によって確立されつつある。多数個のＶＬＳＩ素子を並列的に同時動作させ，Breadth first strategyによる推論実行や多数バンクで構成されるメモリシステム内での知識ベース並列探索などを実現するためのアーキテクチャの検討も非現実的なものではないと思われる。

このように多数個のＶＬＳＩ素子を用いて高度の並列処理を実現し，推論処理，知識ベース管理を可能とするためには高度並列処理の計算モデルとそのモデルにもとづくアーキテクチャの具体化が不可欠である。

現在，並列処理，分散制御を実現するための計算モデルおよびアーキテクチャとしてデータフローマシンがその実現化の可能性という観点から有効であると考えられる。

データフローマシンでは"各演算はその必要なオペランドが揃ったときにいつでも実行可能となる"という実行原理により処理を進める。この実行原理によりデータフローマシンは従来のノイマン型マシンに比べて高度並列処理，関数的処理，分散制御，記憶セル概念の排除，等を徹底させたアーキテクチャ方式およびプログラミング方式を実用的なものとする可能性をもっている。

またこれらの特性から，データフローマシンは関数型処理方式をさらに発展させた述語論理型処理方式すなわち先に述べた推論処理，知識ベース管理のためのハードウェア基盤を与える可能性を持っているといえる。

この可能性を追究し，実用的なマシンアーキテクチャを確立していくことが本プロジェクトの目的である。

2.2 データフローマシンの目的と意義

本プロジェクトは第5世代コンピュータシステムのハードウェアアーキテクチャの基盤を確立することを目的とするものである。第5世代コンピュータは知識情報処理を主たる目的として開発されるが，高度の知識情報処理をサポートするマシンの構造としては従来のノイマン方式による逐次型マシンでは処理能力の点で不都合である。第5世代コンピュータのソフトウェア方式は従来方式とは質的に異なり述語論理の概念を基本とし，その処理メカニズムには非決定性処理，パターン照合処理，関数的処理等の機能が要求される。これらの機能を実現しかつ処理を実用的時間内で行なうためには，多数個のＶＬＳＩ素子を同時実行させる高度並列処理の方式を確立することが，第5世代コンピュータのハードウェアアーキテクチャ開発課題となる。このような高度並列処理マシンのアーキテクチャとしてはデータフローの原理にもとづく方式が有効であると考えられる。VLSI素子技術の進展は著しく，今後さらに高集積，高機能化素子が開発されていくものと思われるが，これらのＶＬＳＩ素子を多数個用いる情報処理システムのイメージは定かでなく，コンピュータアーキテクチャに課せられた研究課題となっている。この多数個ＶＬＳＩを用いるシステムの制御方式としてはデータフロー概念にもとづいて分散制御を実現するのが現在の所最も有効なように思われる。この意味でもデータフローマシンの方式確立はコンピュータアーキテクチャにとって重要な研究項目であるといえる。

データフローマシンの開発は世界的にみてもまだ基礎研究の段階であり，実用マシンの開発を目指した研究は未だ行われていない。

また世界の研究の大半は超高速科学技術計算を応用領域と考えて研究を進めており，第5世代コンピュータプロジェクトのように知識情報処理，即ち記号処理を応用領域と考えたデータフローマシンの開発研究は日本独特のものである。この意味で本プロジェクトの構想は世界先導的であり，プロジェクトの与えるインパクトは大きいといえる。

第5世代コンピュータの開発では，そのサブシステムとして推論サブシステムおよび知識ベースサブシステムを開発し，これを融合させていくという方策をとる。この両サブシステムは中期段階で並列処理方式にもとづいて開発が進められる。このためのハードウェアメカニズムは高度並列処理をサポートするものでなければならず，その基盤は本プロジェクトで確立されたデータフローマシンアーキテクチャにより与えられることになる。即ち中期に開発される両サブシステムのハードウェアシステムとしては並列型推論マシンサブシステムおよび並列型知識ベースマシンサブシステムが予定されているが，この両マシンサブシステムの開発にはデータフロー制御方式，データストリーム処理方式，構造メモリ構成方式，ネットワーク構成方式などの要素技術の確立が必要になる。

本プロジェクトの推進によって確立されるデータフローマシン方式に関わる種々のハードウェア構成方式，実現方式，実装方式等がこれらの要素技術確立のための基盤を与えることになる。

2.3 計算モデルとデータフローマシン

最近のＶＬＳＩ技術，素子技術の進展には目覚ましいものがあり，高速・高機能・低コスト化の傾向を強めている。一方，ソフトウェアの低生産性の問題が顕在化しており，知能処理への要求が高まるにつれて，今後益々問題となってくるものと思われる。これを背景に，３０数年以上続いたノイマン原理にもとづく計算機の構造を変革し，ノイマン原理に依らない言語および計算機アーキテクチャを構築しようとする動きが活発化してきている。ノイマン原理のコンピュータの特徴は，ＣＰＵとメモリを結ぶ一本の通信路からなる構造，通信路を経由して one at a time に進められる計算，プログラムカウンタを用いた逐次的・集中的制御，線形アドレスメモリ概念にある。かってノイマン型コンピュータの利点であったこれらの特徴が，今日の計算機にさまざまな弊害をもたらしていることが明らかになってきた。

例えば，プログラムカウンタによる逐次制御（single control flow stream）は，並列処理構造を実現する上で大きな支障となり，ＶＬＳＩ技術への適合が困難である。また，プログラムの面では，一本の計算の道筋を与えてしまい並列実行可能性を失う他に，goto 文にみられるように jump 概念が入りプログラム構造を不透明にする。一方，メモリセル構造は，計算に本質的でないプログラム変数なる概念をもたらし，副作用などわずらわしい問題を引き起こす元凶になっている。

このような反省を基に，ノイマン原理から脱脚した新たな計算機構（非ノイマン型コンピュータ）が注目を集めるようになった。図2.3.1は非ノイマン型コンピュータの研究動向をまとめたものである。これらはデータフロー型マシンの研究とリダクション型マシンの研究に分かれるが，いずれも関数型マシンとして位置付けられる。

A．データフロー型マシン

データフローマシンのアイデアは最初 J. B. Dennis によって明確に提示された。

データフローマシンでは，図2.3.2に示すように，プリミティブなレベルのプログラムは演算間のデータの流れを記述するデータフローグラフで表現される。データフローグラフは，入力トークンが入れられると実行を行い，出力トークンを出す，関数を定義しているとみることもできる。

関数あるいは命令の実行契機の与え方の違いによって，データフローマシンはデータ駆動型と要求駆動型に分けることができる。

データ駆動の原理では，命令は必要なデータ（トークン）が全て揃った（到着した）ときに何時でも実行可能となる。実行原理は，命令の実行順序が制御移行（control flow）の形で予め

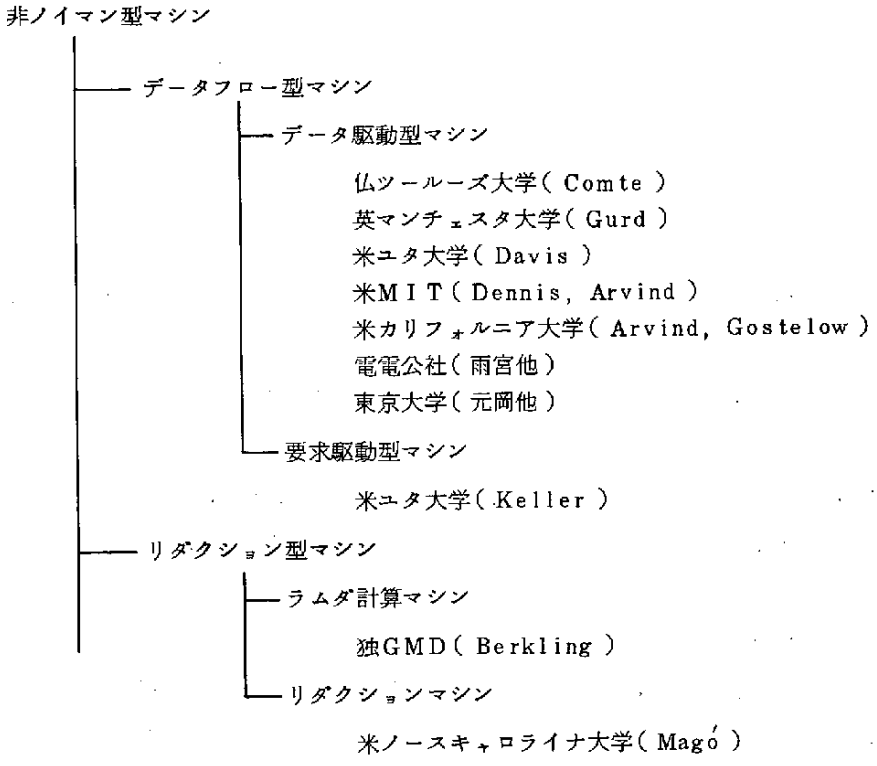


図 2.3.1　非ノイマン型マシンの研究動向

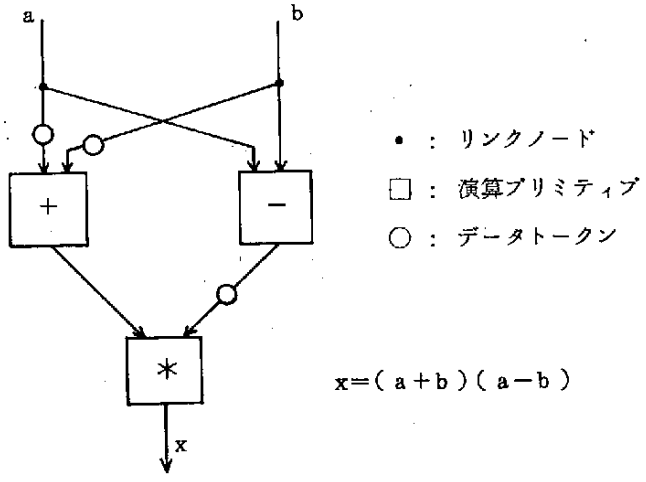


図 2.3.2　データフローグラフ

定められている従来のマシンとは大きく異なっている。関数の実行順序はプログラムカウンタの指示やデータフローグラフの位置によらず，データの依存関係のみで決まる。プログラム中に並列実行の指定を陽に行わなくとも，問題に内在する並列性がそのままの形で引出される。

データフローマシンは，このような実行原理を具体化するマシンとして提案されているものであるが，プログラムカウンタやメモリセルの概念は持っていない。ノイマン型マシンの元凶は少なくとも原理において取り除かれている。このことおよび自然な形で並列処理を達成しうる構造であることがデータフローマシンの大きな魅力である。

一方要求駆動の原理では，命令はその結果が必要とされるときに駆動され，命令の実行に必要なデータが到着してなければその発生元の命令に要求を出す。要求駆動は，無駄な計算を行わずに済み資源の有効利用が図れる点や，計算の能力の点（例えば，必要になるまで実行を引き延ばす遅延評価が行いやすい）でデータ駆動よりすぐれた面を持つが，実現性，効率の観点では問題がある。現在研究・開発されているデータフローマシンの殆んどはデータ駆動型である。一般にデータフローマシンという時にはデータ駆動型をさす場合が多い。

B．リダクション型マシン

リダクションの機構では，ひとつの式を実行できる部分から評価してその部分式を評価結果のデータで置き換えていくことによって計算が進められる。literal substitution（文字列置換え）による還元（reduction）が計算の基本となる。リダクション型マシンはこの機構をハードウェアで実現しようとするものである。

Berkling はこの考えにもとづき，スタックを用いたラムダ計算マシンを開発した。ラムダ計算においてλ式の引数への適用（application）を，式中の変数の文字列置換えという方法で還元していくものである。call by name，call by value，static binding などを実現するため機械語として数種類の application primitive を用意している。

最近では，J. Backus の提唱した関数型言語（FFP）を実行する Mago のリダクションマシンが注目されている。引数評価が最内側から行われることに特徴があり，全引数が予め評価されるまで関数の計算は始められない。計算が最内側から行われる点ではデータ駆動と差はない。

リダクションマシンは，ノイマン原理からの脱却を図った関数型マシンのあり方として示唆に富んだものであるが，実現性の面で不明な点が多い。

C．計算モデルとの関連

ノイマン原理に依らないコンピュータとして，データフロー型マシンおよびリダクション型マシンの研究が行われているが，これらのマシンの計算基盤となる計算モデルとプログラミング言語との関係について少しふれておく。計算の構造について考える場合，言語の syntax（表現の

世界）とsemantics（意味の世界）を記述したformal systemとその computing system（計算の世界）とに分けてとらえるとみ通しがよくなる。

計算モデルは computing systemを抽象的に表現したものであり，言語の semanticsをマシンの上で実現する機構を規定するものである。これまでに存在するあるいは提案された計算モデルを整理すると，control flowモデル，data flowモデル，reductionモデルに分類することができる。これらのモデルの特徴については4.2.7計算モデルで詳述する。

control flowモデルは従来のノイマン型マシンをモデル化したもので，プログラムカウンタによって逐次的に進められる計算を，制御の流れ（control flow）としてとらえる。メモリ概念が存在し，データは参照によって（by reference）処理される。制御の流れは，単一（single control flow stream）であり，従って計算の道筋は一意に定まる。

data flowモデルは，データフロー原理（by value）にもとづく計算をモデル化したものである。実行に必要なオペランド（データトークン）が到着するといつでも実行が始められる。データに依存関係がない計算は独立に（並列に）実行することができ，計算の道筋は複数存在する。data flow モデルを具現化したマシンがデータフローマシンであり，パケット通信を基本に構成される。

reductionモデルは適用型言語の semantics を忠実に反映した計算モデルである。application の過程を reduction としてとらえ，文字列の置換えにより計算が進められる。置換の対象（reducible subexpression)が複数個存在する場合，それらは任意の順序で置換が行われてもかまわない。したがって計算の道筋は複数存在することになる。reductionの方法には，直接文字列そのものを置換する string reduction と，ポインタを利用して式（グラフ）の変更を行う graph reduction の2つがある。reduction マシンの構想例はいくつかあるが実際に作られた例は，Berklingによるスタック型のマシンのみである。

ここで各計算モデルと言語及び実マシンとの関係を少し考察しておこう。

control flowモデルはノイマン原理そのものであり，従来型言語およびノイマン型マシンとの結びつきがはっきりしている。一方，data flowモデルは，言語の semanticsが先にあってそれを実現するモデルとして提案されたというより，むしろデータ駆動制御という計算の進め方から生まれたモデルである。data flowモデルと対応関係にある言語として，単一代入言語をあげることができるが，単一代入規則を遵守すれば data flow 計算に適合するという歴史的過程から生じた関係にすぎない。したがって data flow モデルの上にどのような言語を乗せるかが問題となる。data flow モデルは関数的性質を保存しており，駆動による実行という観点からも関数型（適用型）言語や論理型言語との関係が深いが，それらの言語と計算モデルとして

どういう関係にあるのか，それ程明らかになってはいない。これに対して，reductionモデルはラムダ計算における semantics を反映したもので適用型言語と密接な関連がある。しかし実際の reductionマシンとして具体的に提示された例はなく（Berklingらの試みはあるが），実現性については不明である。最近注目されだした論理型言語に目を移すと，論理型言語の semantics を反映した計算モデルも実マシンも今の所存在しない。

データフローマシンが関数型あるいは論理型言語の実行マシンとなるかを探究する糸口として，関数型言語と論理型言語の違いについてもう少しみてみよう。言語の syntax（スタイル）の面では，関数型は applicative であり，論理型は declarative である。計算モデルと関連の深い semantics の面からは次のことがいえる。関数型ではオブジェクトの変換（オブジェクトからオブジェクトへの写像）と内側計算を基本とし，どのような変換過程を辿っても一意に解に到達する（もし存在すれば）normal form property を持つ。

一方論理型では，resolution が計算の基本となる。この場合，何通りもの解が存在し，また一意に解に到達するとは限らないので normal form property は成立しない。

データフロー型マシンはかなり研究されてきており，リダクション型マシンに比べると実現性が高いと思われるが，これを関数型あるいは論理型言語の実行マシンにするためには data flow モデルとそれぞれの言語の計算モデルとの関連を明確化しておく必要がある。また data flow モデルと reductionモデルの計算機構としての得失を明らかにしていくことも重要である。

3. データフローマシン研究の現状

3.1 データフローマシン用言語の研究開発の現状

データフローマシン用言語としては単一代入型言語と関数型言語がある。単一代入型言語は1つの手続き内では同じ変数に2度以上値を代入してはいけない，いいかえれば代入文の左辺に同じ変数名を2回以上使用してはならないという制限を持っている。この制限によりデータの間の副作用がなくなり並列計算が可能となる。この型の言語は従来の手続き型言語であるFortranやAlgol等とみかけ上あまり大きな違いがないため多くのデータフローマシンに採用されている。単一代入型言語は，通常はデータフローグラフに変換され，データ駆動方式により実行される。

一方関数型言語は数学の関数の概念をそのままプログラミング言語に取り入れたもので，f(g(h(x)))のようにプログラムを記述する。関数型言語は通常ラムダ計算にもとづき，リダクション方式により処理されるものが多い。

この節では単一代入型の言語の代表としてVal，Id，Valid，CAJOLE を取りあげた。その理由は前3者はコンパイラを作成中であり活発な研究が続けられているため，またCAJOLEはその理論的な面に興味があるためである。

3.1.1 Id

A．場　所　　カリフォルニア大学アーバイン校

B．人　　Arvind，R. Thomas 等

C．特　徴

ブロック構造を持つ単一代入型の言語で構文的にはAlgol にもとづいている。言語の意味はデータフローグラフで表わされ，演算はノード，変数はグラフの線，変数の値は線上にあるトークンに対応する。線上のトークンは2個以上同時に存在してもよい。

Id の基本的機能を以下に述べる。

ブロックはノードを線で結んだもので，通常の演算を表現したものである（図3.1.1）

条件式は図3.1.2で表わされる。bが真であると各スイッチはTの方の線にトークンを出し，偽であればFの方の線にトークンを出す。

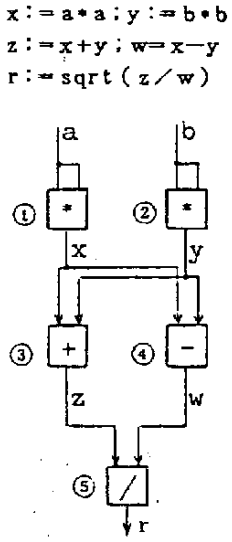


図 3.1.1

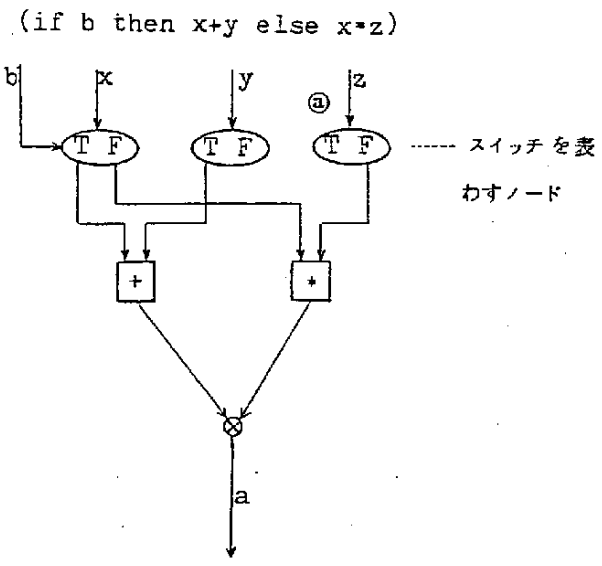


図 3.1.2

関数の呼出しは apply 演算子を用いる。図3.1.3で①の箱はⓐの線へトークンが来ると2乗根を求める関数 Sqrt の定義を出力として②の箱へ送る。②の箱はⓐの引数を表わすトークンに①から来た関数を表わすトークンを適用してその結果をトークンとして出力する。

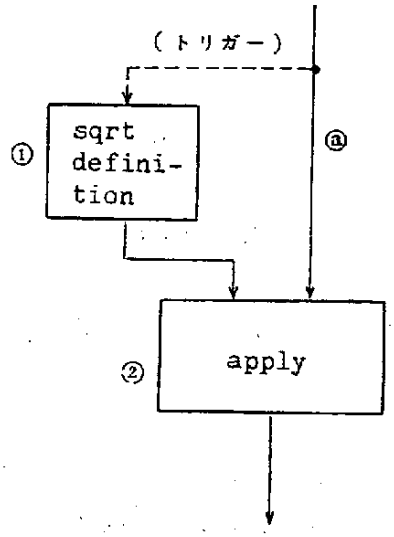


図 3.1.3

ループは図3.1.4のように表現される。

```
( initial    x ← a
  while      p(x)  do
        new    x ← f(x)
  return     x  )
```

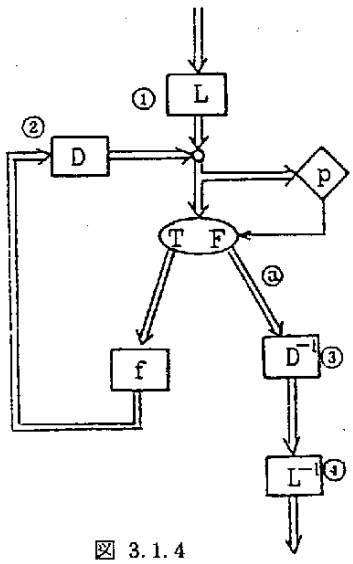


図 3.1.4

演算Lはこのループに入るトークンに色をつける働きをする。Idでは同じ関数を並列に実行する場合プログラムを共有することが可能になっている。そのためユーザ定義の関数単位でトークンに色を付け，どの関数に属するトークンであるかを識別している。ループも1個の関数とみなすのでLが必要となる。L^{-1} はループから出て行くトークンの色を取り去り，このループを呼んだ関数の色を与える。Pは繰り返しの判定式で，この値が真なら繰り返しを行い，偽ならループを抜ける。Idではループの中も並列に実行する。従ってn－1回目のループのトークンとn回目のループのトークンが入り混じるため各ループごとに色を付ける。それがDで，D^{-1} はその色を取り去る。

データ構造はトークンの列を一まとめにするストリームと構造メモリ（I－Structure）がある。この構造メモリの特徴は書き込みも読み出しも1回しか許さないことである。こうすることにより副作用を防止している。

Val との相異は思想的にはVal が静的（すなわちコンパイル時）に処理を行うのに対し，Idは動的（実行時）に処理を行う。文法的には Val は型宣言があり，Id にはない。意味的にはValはグラフの線上に1個のトークンしか許さないが，Idは複数個を許す。

D．問題点

処理を動的に行うのでオーバヘッドが大きいといわれている。

E．現　状

IdをLisp に変換する版とデータフローグラフに変換する版のコンパイラが開発されている。

―参考文献―

1）Arvind "データフローアーキテクチャの研究開発"　電子協昭和54年度特別セミナー議事録

2）Arvind 他 "An Asynchronous Programming Language and Computing Machine"　California Irvine TR114A, 1978.

3.1.2 VALID

VALID(Value Identification Language)は電電公社武蔵野通研で設計・開発が進められているデータフローマシン用高級言である。[1)]

武蔵野通研ではデータフローマシンの研究を，数値処理用と記号処理用の両面から行っているが，両マシンでのプログラミング環境を構築するために，汎用の高級言語として設計されたものである。

VALIDは関数概念を徹底させた適用型の高級言語で，シンタクスはAlgol 風を基調としながらセマンティクスはあくまで関数型を貫くという設計思想にもとづいており，以下の特徴を有している。

(1) プログラム変数の概念はない。プログラムはValue 定義，Function定義およびMacro 定義のみからなる。特にValue 定義式では，

〔 x_1 , x_2 ・・・, x_n 〕 = ＜expression＞

により複数 value を一度に定義できる。

(2) 式の集合であるBlock 概念を持つ。Block 自身も式であり，clause ・・・ end で定義される。Block 中ではscope ルールに従うlocal value の定義が許される。Block中の式は；で区切られるが，評価順序はデータ依存関係のみに依る。Blockの生成する値は return 式により示される。

(3) 繰返しは全て再帰概念にもとづいて記述される。従来型言語でのループ表現による繰り返しは，次のような再帰式で記述される。

```
for (＜iteration value names＞) init (＜initial values＞)
  do ＜block including recur expression＞
```

再帰式は無名の再帰関数を定義するものと解釈する。

（例1）

```
for (r, q : integer) init (y, o)
  do if  r＜x  then return (r, q)
              else recur (r−x, q+1)
```

(4) 複数Block の並列実行は fork−join 概念にもとづく並列実行式[2)]

```
for each ＜value name＞ in 〔＜expr1＞・・・＜exprn＞〕
  do ＜parllel body＞
  return ＜construction expression＞
```

により陽に表わすことができる。

（例2）

```
for each i in 〔1・・・n〕 do
        z = a〔i〕* b〔i〕
    return vector (z)
```

(5) その他，階層的なFunction，Macroの定義機能，データの構造化機能（vectorなど）の他，高階関数の定義機能を有する。[3)]

（例3）

```
fullrev : function (x : list) return (list)
  = if atom (x) then x
    else append (fullrev (cdr (x)),
                 list (fullrev (car (x))))
```

（例4）

```
reduce : function (f : function, x : list, a : integer)
         return (integer)
  = if null (x) then a
      else f (car (x), reduce (f, cdr (x), a));

add : function (x,y:integer) return (integer) = x + y ;
mul : function (x,y:integer) return (integer) = x + y ;
```

高階関数 reduce を使うと，整数系列の和・積は次のように定義することができる。

```
sum (x) = reduce (add, x, o)
product(x) = reduce (mul, x, 1)
```

現在，Valid コンパイラ作成が終了し（Maclispで書かれている），リンケージローダ等プログラミング環境の整備が行われている。Valid コンパイラはソースプログラムをS式表現された中間言語（データフローグラフ）に変換するものである。この中間言語から，DECシステム2020上で開発されたデータフローマシンシミュレータおよび4×4PE構成のデータフロープロセッサアレイ実験機EDDYのコードに変換するアセンブラが完成している。

VALIDではI／Oのサポートが未だ行われてなく，stream 処理，history sensitivity 処理のための機能拡張を検討中である。

—参考文献—

1) 雨宮，"データフローマシン用高級言語 Valid の設計思想"，昭和56年信学会全国大会 №1486.

2) 尾内，雨宮，"データフローマシン用言語 Valid における並列実行式の処理"，情処学会第22回（昭和56年前期）全国大会，5B－4.

3) Amamiya M., Hasegawa R., and Onai R., "A Functional Programming Language VALID and Data Flow Machine as Its Computing Apparatus", To be published shortly.

3.1.4 VAL

VAL（Value oriented Algorithmic Language）はMITのW.B.Ackerman, J.B.Dennis らによって1979年頃開発されたデータフローマシン用高級言語である。

A. 特 徴

VALは数学的意味においての関数型言語であり，並列性は implicit に表現される。つまりプログラマやコンパイラは特別な記法を用いることなく大部分の並列性を認識することができる。以下VALの特徴について簡単に述べる。

- 関数型構造

通常の言語が文（statement）と変数によってプログラムを記述するのに対し，VALでは式（expression）とValueによって記述する。VALにおいてactiveなものは全て式であり，何らかの結果を生成する。これは if文，case文，loop文においても同じである。表記上VALにおいてValue は identifierにバインドされるが，通常の変数とは異なる。この意味でVALはLISPと類似性を持つが，代数的表記であること，strongly-typed 言語であること，等が異なる。

VAL	LISP
A + B * C	(PLUS A (TIMES B C))
if A = B then C else D endif	(COND ((EQ A B) C) (T D))

図 3.1.5

- Implicit concurrency

  VALでは副作用がないため，プログラム内のほとんどの並列性は implicit に表現される。

- Special parallel features

  VALではいくつかの特別な並列性表現が用意されている。図3.1.6は配列計算における forall の例である。

```
forall I in [1,10]
  Mean, SD : real
           : = Stats(A[I], B[I], C[I]);
  Plus 2   : real
           : Mean + 2 * SD
  construct
endall
```

図 3.1.6

- Explicit limits to concurrency

VALにおいて並列性を制限する言語構造は if 文, tag case文, for文の3つである。for文の例を図3.1.7に示す。これはFibonacci(n)を求めるプログラムである。先頭の部分はループパラメータの初期化である。

```
for
    Count, Fib, Pre−Fib : integer    % init. Values for
                        : = 0, 0, 1  % the loop parameters
do
    if count < = N
        then  iter
                Pre−Fib, Fib : =
                 Fib, Fib + Pre−Fib ;
                 Count : = count + 1
              enditer
        else
              Fib                    % Expr. result
        endif
endfor
```

図 3.1.7

B. 現 状

言語仕様[1]も確定しており各種の応用プログラムによって評価が行われているが,[3] I/O関係に不充分な点もあるため, 言語仕様の練り直しも行われている。

−参考文献−

1) W.B.Ackerman, J.B.Dennis : VAL−A Value−Oriented Algorithmic Language Preliminary Reference Mannual, CSG, TR−218, MIT, June, 1979.

2) W.B.Ackerman : Data Flow Language, AFIPS, Vol. 48, June, 1979.

3) J.R.MacGraw : The VAL Language Description and Analysis, Lawrence Livermore Laboratory, Dec, 1980.

3.1.5 CAJOLE

CAJOLE〔1〕は1978年，ロンドン大学のウェストフィードカレッジ，計算機学科のC.L. Hankin，P.E. Osmon，J.A. Sharpによって発表された高水準データフロー言語である。

CAJOLEの特徴は以下の通りである。

(1) 非手続き的である。

プログラムは順序づけられない定義（文）の集合である。その実行順序はデータの到着状態にのみ依存する。

(2) 単一代入規則に従う。

このため，ループ構造を許さず再帰呼出しの形へ変換している。

(3) 関数型言語である。

プログラム内の全ての定義は関数定義と考えられる。関数定義の記述法は以下の通りである。

```
name expression = 〔 parameter list 〕
                        value expression
```

ここで，〔parameter list〕は省略することができる。

(4) 非決定的処理機構を導入している。

Dijkstra の guarded command〔2〕を取り入れている。その形式は以下の通りである。

```
guard : expression
```

guarded command の解釈法（特に複数の guard の値が真になるとき）については特に規定していない。

(5) 名前の局所化機能を備えている。

with ･･･ wend 節を用いて特定の式の名前を結合している。

(6) ブートストラッピング機能を導入している。

この機能によってユーザ自身が構文を定義できる。ブートストラップ定義の一般形は以下の通りである。

```
〔parameter list〕 construct name
                    = defining expression
```

construct name はパラメータを区切るシンボルの列である。シンボルには予約語（＋，－等）を除く1文字，ないし下線を付した文字列である。

(7) オペレータ

言語では標準的な比較，論理および算術オペレータを用意している。そのオペランドは単純オ

ペランド（例えば整数）でもよいし，集合オペランド（例えばベタタ）でもよい。

(8) システム標準構文

条件式やくり返し式等のための構文はシステムで定義されている。

以下に階乗を求めるプログラム例を示す。

```
dummy , factorial n
  = factorial (n,1)
    with factorial
            = [n,fact] if n>0
              then factorial (n-1, fact * n )
              else n,fact
              fi
    wend
```

このプログラム例では関数（factorial）の再帰呼出しが発生する度に関数本体が次々とコピーされて実行される。

問題点としてはプログラミング上の制約条件が大きすぎる（例えばループ構文を許さない）ことがあげられる。

論文[1]を発表した時点ではCAJOLEのコンパイルによって生成されるベース言語の設計途中にあり，これをもとに従来マシン上でのシミュレーション／エミュレーションを行う予定である。

一参考文献一

1) C. L. Hankin, P. E. Osman, J. A. Sharp, "A Data Flow Model of Computation". Department of Computer Science, Westfield College (Univ. of London), March 1978.

2) E. Dijkstra, "Guarded Command, Non- determinancy and a Calculus for the Derivation of Programs". Springer Verlag Lecture Notes in Computer Science No 46, 1976.

3.2 データフローマシンの研究開発の現状

データフローマシンは現在，欧米・日本の各所で研究が進められている。データフローマシンの研究の発端は，1945 年にペンシルバニア大学の Van Neuman が提言して以来今日まで発展し続けてきたノイマン型アーキテクチャに対する物理的限界と，そのアーキテクチャがもたらすソフトウェア危機の認識にある。特に1970年代における並列計算機の研究においてその楽観論が後退したことにより，データフローマシンへの期待が一層高まってきたと言える。

データフローマシンの基盤となる言語やプログラムのデータフローに関する研究は，前節で述べたように1960年代後半からあった。これに対しデータフローマシンの構造についての研究が本格的に行われるようになったのは1970年に入ってからである。先駆的研究を進めてきたのは MIT のDennisらのグループであり，当初は小規模な専用高速プロセッサから始まっている。それ以後米国のみならず欧州，日本など世界中に研究が広まってきている。

現在までに多数のマシンが提案・試作されてきており，データフローマシンの性格も，高速汎用機を目指したもの，数値計算向きのもの，記号処理向きのもの等，豊富になってきている。すでにツールーズ大学，マンチェスタ大学、東京大学などにおいては実験機が稼動している。

これまでに行われてきたデータフローマシンの研究開発は，大学や研究所が中心であったが，現在は計算機メーカにおいても本格的に取り組んでおり，データフローマシンの研究が実用段階に入っていることがわかる。

以下，海外および国内で進められているデータフローマシンの主な研究について述べる。

3.2.1 Dennis (MIT) マシン

A. 開発時期 ： 1970年前半

B. 開 発 者 ： J.B.Dennis (MIT)

C. 特　　徴 ： データフローマシンとして最初に研究されたマシン。原始データフロープロセッサ，前基本データフロープロセッサ，基本データフロープロセッサへと段階的に研究が進められている。Dennisのマシンの思想は原始データフロープロセッサ〔 DENNIS79 〕に適切に表わされているので，そのシステム構成を図 3.2.1 に，また，その中のセルブロックの詳細を図 3.2.2 に示す。

o Function Unit を共有化

実行可能な命令を検出する部分と，実際にその命令を実行する部分とを分離し，実行ユニットを共有化している。従って，命令から命令への通信時間は，命令がどのセルに位置している

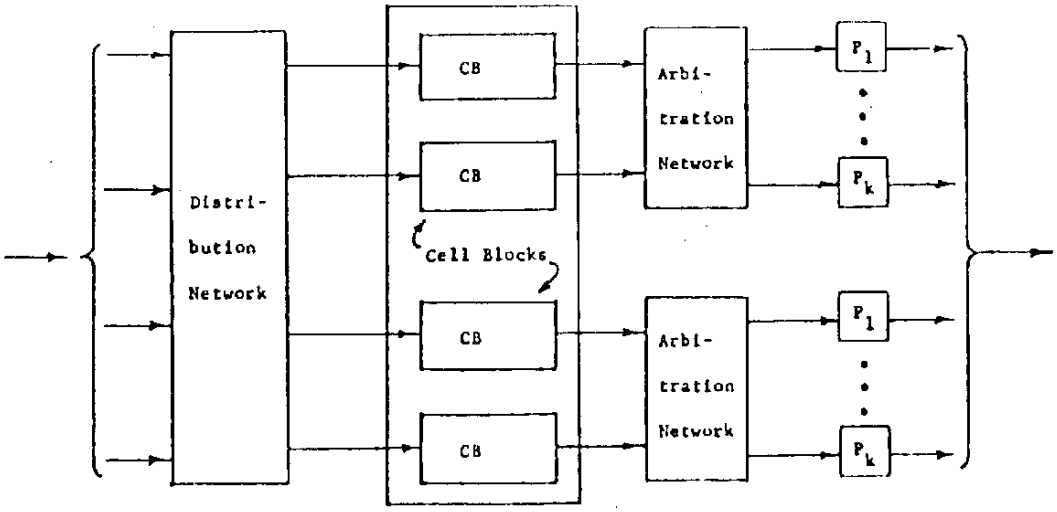


図3.2.1　Dennisマシンのシステム構成

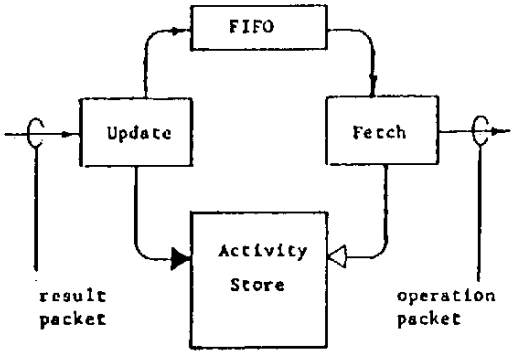


図3.2.2　セルブロックの構成

かに無関係である。

- Ack信号の使用

命令の発火規則として，オペランドデータが揃う事以外に，その命令の出力アーク上に結果

が残っていない（即ち，その次の命令によって受け取られている）事をも条件として追加されている。それ故，アーク上には1個のトークンしか存在することが許されないが，パイプライン的にデータフローグラフを使えることになる。

- セルブロックがキャッシュ的

入り切らない命令は，Instruction Memory に置いておく。例えば，手続きが呼び出された時には，手続全体のコピーを作らずに，その手続の最初の命令セルしか作らず，実行が進むにつれて順次コピーが増えて行くので必要な命令だけがセルブロックに存在する。

D. 問題点 ： 特徴でもある点が逆に問題点にもなっている。

- 命令間の通信時間が大

実行ユニットを共有にして，実行可能な命令を検出する部分と分離した構成になっていることで，ローカルなフィードバックが出来ず，命令間の通信時間が常に遅いと考えられる。

- 並列性が小，通信量が大

Ack信号を使用しているため，アーク上には1個のトークンしか存在出来ないので，識別子をつけたトークン方式に比べて並列性が出ない。

また，Ack信号を使うために，通信量がほぼ2倍になり，ネットワークに負担がかかる。

E. 現　　状 ： プロトタイプシステムの作成を計画している〔DENNIS80〕。その構成を図3.2.3に示す。

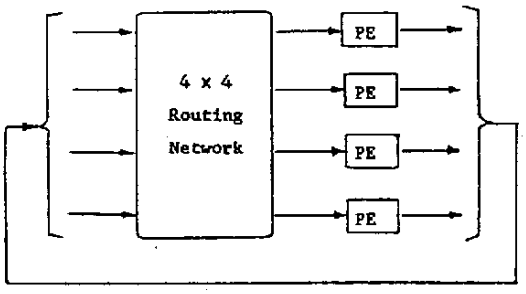


図3.2.3　Dennisのプロトタイプマシン

当初の構成図と異なり，一見，Arvind型と同一に思われる。ただしPEがマイクロプロセッサを使ったユニバーサルなモジュールで考えているので，論理的には以前と同様なのかも知れない。プロセッサ3台分が出来上がっているが，マイクロの未実装で1台でしか動作していない。UNIXシステムに接続されていて簡単なプログラムを実行出来る。

— 参考文献 —

〔DENNIS79〕 Dennis, J.B., "The Varieties of Data Flow Computers" Proc. 1st Intl Conf.on Dist. Comp., Huntsville, Alabama Aug. 1979.

〔DENNIS80〕 Dennis, J.B., Boughton, G.A., Leung, Clement K.C., "Building Blocks for Data Flow Prototypes" Proc.of the 7th Annual Symposium on Computer Architecture, vol.8 №3 May 1980

3.2.2 Arvind (MIT)マシン

A. 場　所 : マサチューセッツ工科大学

B. 人　　: Arvind

C. 特　徴 : アーキテクチャの特徴は

(1) 高度の並列性を得るためプロセッシングエレメント(PE)を高度に分散すること。

(2) 非同期動作

(3) VLSI向き

(4) 遅延があってもスループットの大きい構成。

(5) 何台でも接続可能で台数に応じ性能が向上

等である。このマシンはIdのインタプリタをハードウェア化したものである。全体の構成を図3.2.4に示す。PEとネットワークはパケットを用いて通信し，パイプライン式に働く。PEの構成は図3.2.5である。Waiting Matching Sectionはトークンの待ち合わせを行う。Instruction Fetch Sectionはトークンに対応する命令をプログラムメモリから取り出す。Service Sectionは命令を実行し結果をトークンの形式にまとめる。I−read Sectionは構造メモリの命令を処理する。その中にはまだデータが書き込まれていないメモリを読み出しに来た時はその命令を待たせる機能がある。

PEの特徴は各Sectionが非同期のパイプラインで動くこと，トークンの待ち合わせ方式，I−structureのサポート等にある。

ネットワークは8×8のルータを必要な数だけ並べてPEを接続する。PEは幾つかのグループに分けられており，プロシージャや構造メモリはこのグループを単位として割り付けられる。これらの管理はスケジューラが行う。

D. 問題点

構造メモリを各PEに分散しているのでデータアクセスのオーバヘッドが大きくなるかもしれない。手続きの割り付けやロード，構造メモリの割り付け等が動的に行われるためオーバヘッドが大きい。分散の程度が高いためスケジューリングが難しい。

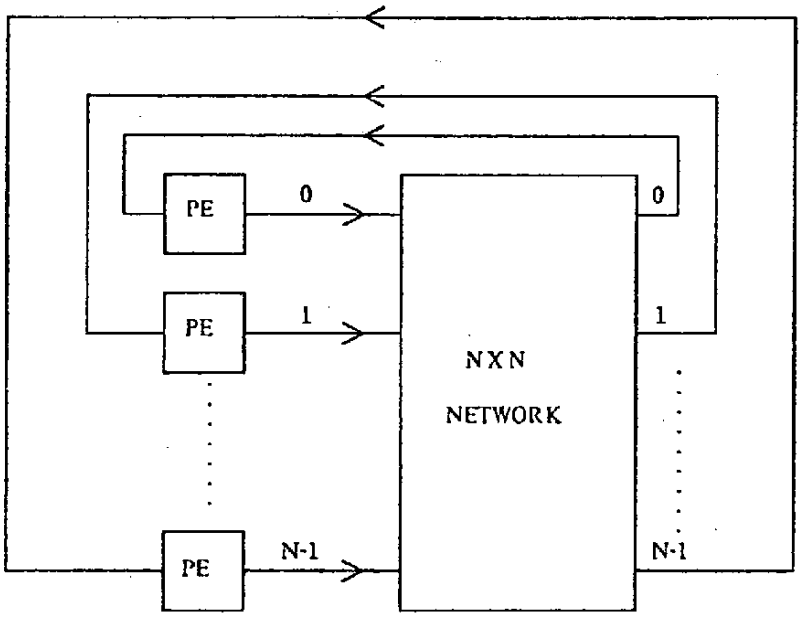


図3.3.4　全　体　構　成

E. 現　　状

このシステムの最終的インプリメントはＰＥと８×８ルータをnMos VLSIで作る予定である。すでにWaiting Matching SectionとルータはＶＬＳＩ化に着手している。またハードウェアシミュレータとしてM68000を４～８台接続したシステムを作成中である。このシステムの目的はＩｄで書かれた大きなプログラムを実行し，種々のデータを取ることである。ソフトウェアシミュレータは完成しておりスケジューラや構造メモリの研究，アーキテクチャのチューニングに使用されている。

—参考文献—

1) Arvind 他 " A Data Flow Architecture with Tagged Tokens " MIT LCS Memo TM174, 1980

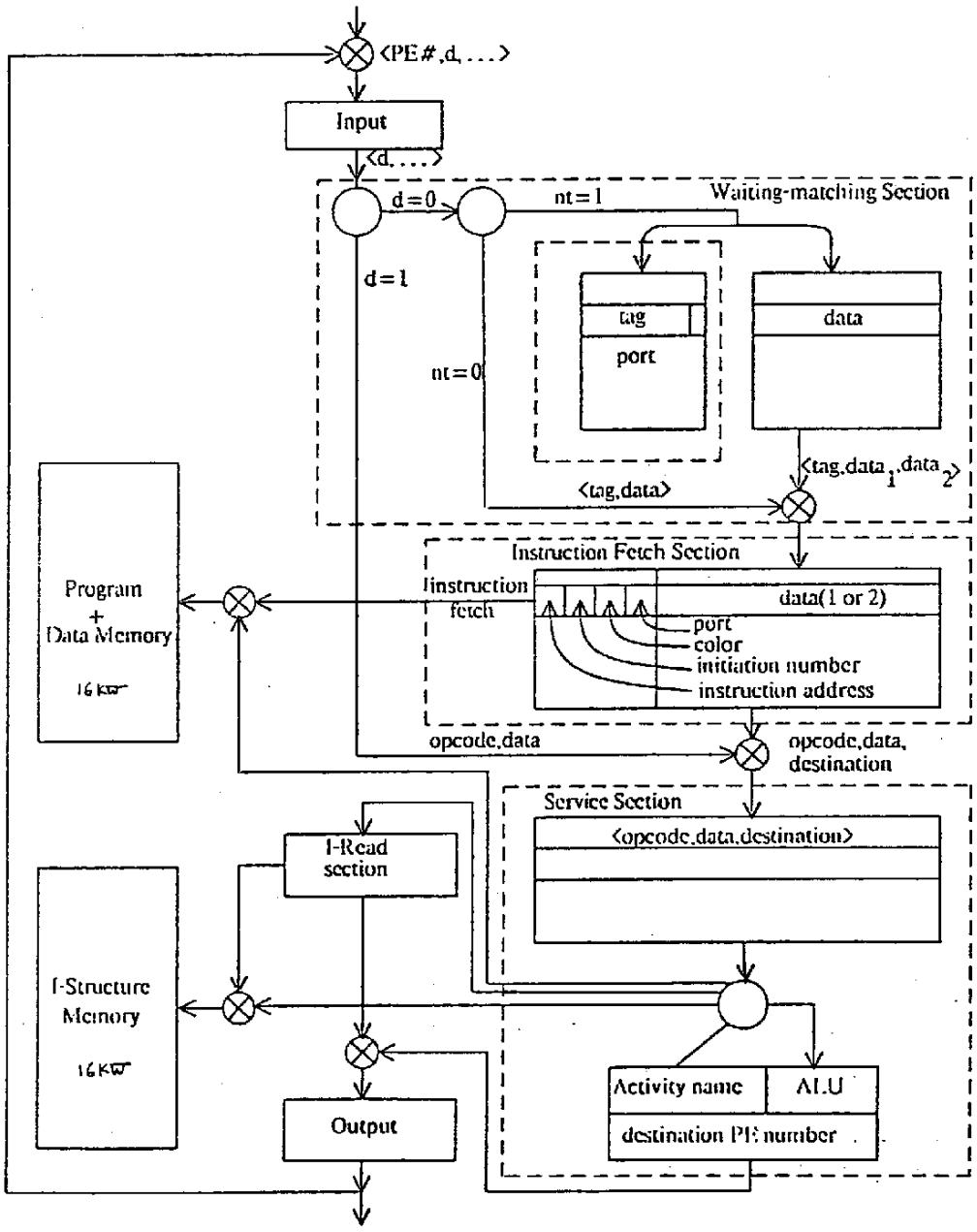

<PE#,d,...>
Input
<d,...>
d=0
nt=1
Waiting-matching Section
d=1
nt=0
tag
port
data
<tag,data1,data2>
<tag,data>
Program
+
Data Memory
16KW
Instruction Fetch Section
Instruction
fetch
data(1 or 2)
port
color
initiation number
instruction address
opcode,data
opcode,data,
destination
Service Section
<opcode,data,destination>
I-Read
section
I-Structure
Memory
16KW
Activity name
ALU
destination PE number
Output

図3.3.5　P E の 構 成

3.2.3 Rumbaugh マシン

A. 研 究 者 ： J.Rumbaugh , MIT(GE)

B. 特　　徴

複数のPU(アクティビティメモリユニットとエグゼキューションユニットを中に含み，閉じた処理ができる)とスケジューリングを行う専用プロセッサ，１つの構造体メモリと複数の構造体メモリコントローラ，プログラムメモリとスワップメモリよりなる。

１つのPU に割り付けられるのは，常に１つの手続きであり，この割り付けは完全に動的であって，call があったときに，idle PU がスケジューラにより割り付けられる。手続きの処理が終了したときは，スケジューラを通じて，caller に値が渡され，ＰＵの解放もスケジューラが行う。従って，ＰU間の直接通信はなく，各ＰＵはスケジューラとだけ通信する。

ＰＵの有効利用のため，手続きはそれが終了していなくても，call した手続きの終了を待っているような場合には，スワップメモリにスワップアウトされ，他の割り付けを待っている手続きがこのＰＵに割り付けられる。スワップアウトされた手続きは，終わるのを待っていた手続きが終了したとき idle PUが存在する場合には，そこへスワップインされる。従って，手続きは同一ＰＵでずっと処理されるわけではない。以上の動的割り付けを行うために，各ＰＵは内部の処理状態を示すアクティビティカウンタを持ち，スケジューラは各ＰＵのそれを絶えず監視している。

アクティビティの制御方式としては，コードの共有を行わないコピー生成方式であるので，カラーは用いない。また，手続きの起動やループの制御方式は，同期式である。

構造体データは一括して構造体メモリに蓄えられており，参照カウントにより管理されている。各ＰＵはそれぞれ構造体メモリコントローラを通じて構造体メモリにアクセスする。read，write 等の構造体操作命令は，構造体メモリコントローラで行われ，参照カウントの更新もこれにより行われる。

C. 問 題 点

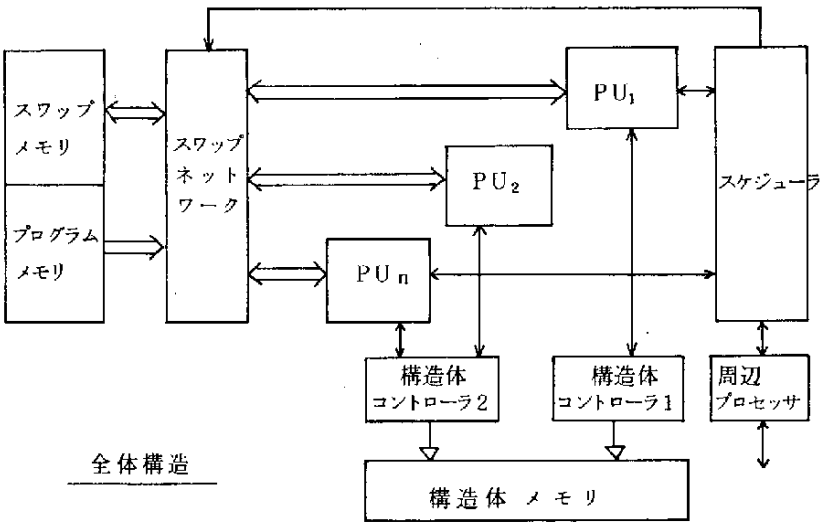
スケジューラが全資源の管理を集中一括して動的に行うため，スケジューリングプロセッサが常に高負荷であり，その処理速度が大規模システム構成にしたときに，ボトルネックになる。

D. 現　　状

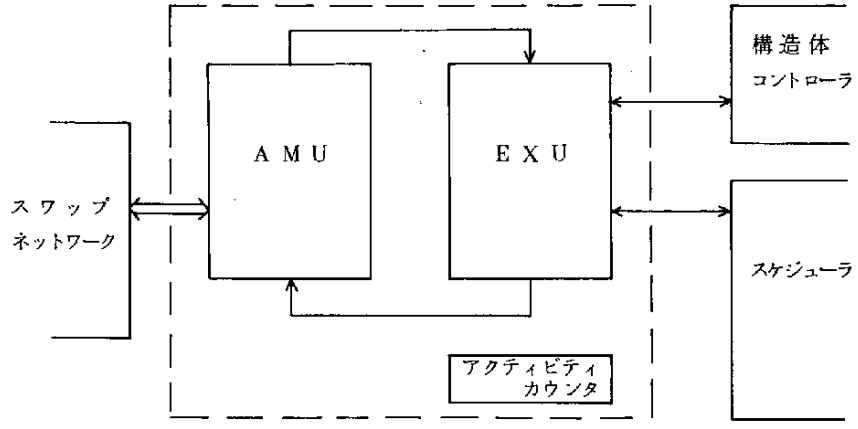
概念設計だけに終っている。

－参考文献－

J.Rumbaugh , "A Data Flow Multiprocessor," IEEE Trans. Computers Vol.C－26, №2, 1977.



全体構造



PU の構造

AMU：アクティビティ・メモリ・ユニット
EXU：エグゼキューション・ユニット

図3.2.6　Rumbaugh 機

3.2.4 LAUシステム

A. 研究者 : D.Comte, N.Hifdi, J.C.Syre. G.Durrieu 他, トゥールーズ大学

B. 特　徴

単一代入規則にもとづくLAU言語を実行する専用マシンである。LAU言語では，すべての文は代入文と見なされ，プログラムは代入文の集合であるデータ生成集合（DPS:Data Production Set）を単位として構成される。DPSは入れ子にすることができ，DPS内部の文は単一代入規則にもとづくからデータ駆動で実行される。

実行制御については，カラーはなく，DPSごとに同期制御を行っている。すなわちDPSが終了するのは，その中のすべての文が終了したときとしている。手続きはコンパイル時にユーザが指定した数だけコピーを生成しておき，Callを実行したときに，freeなコピーとリンクをとり，freeなものがないときは待ち行列に並ぶようにしている。

システムは，制御，メモリ，実行の3つのサブシステムよりなっている。命令及びデータはともにメモリサブシステムにあり，制御サブシステムは，データの有効性を示す1語1ビットのデータ制御メモリ（DCM：Data Conrol Memory）と命令ごとのオペランドの到着状態を示す命令制御メモリ（ICM：Instruction Control Memory)をもち，命令の発火制御を行う。命令実行は以下のようにして行われる。ICMより必要なオペランドすべてが到着していることが検出された命令は，そのアドレスがメモリサブシステムへ転送され，そこで，命令コードが読み出され，実行サブシステムに転送される。実行サブシステムはプロセッサアレイよりなっており，オペランドをメモリから読み出し，命令を実行し，終るとその結果をメモリサブシステムに格納し，同時に，そのデータの有効性を示すためDCM中の対応するビットをセットするよう制御サブシステムに伝える。制御サブシステムは，そのビットをセットし，その後，それをオペランドとする命令のICMを更新する。以上のようにして命令は処理される。

C. 問題点

メモリサブシステムは命令，オペランド，データリンク等，すべての情報を集中して管理しており，その結果，制御サブシステム，実行サブシステムから頻繁なアクセス要求を受けるため，ボトルネックになっている。

D. 現　状

実行サブシステムが，32個のプロセッサをもつ試作機を完成し，現在評価中である。

—参考文献—

1) A.Plas et al., "LAU System Architecture : A Parallel Data-Driven Processor Based on Single Assignment", Proc.of International Conference on Pa-

rallel Processing, 1976.

2) O.Gelly et al.,"LAU System Software : A High Level Data Driven Language for Parallel Programming,"Proc.of International Conference on Parallel Processing.1976.

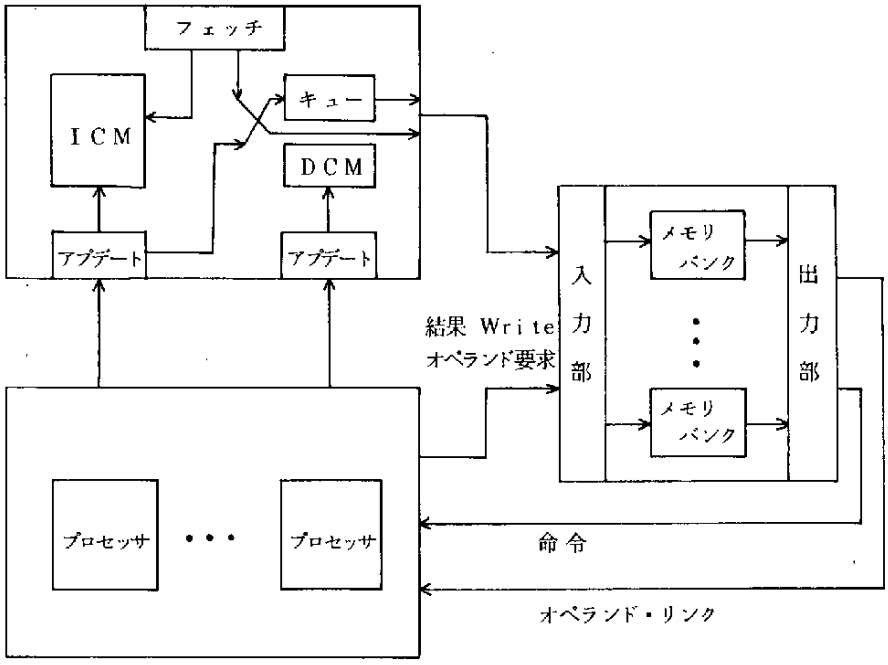


図3.2.7 LAU

3.2.5 DDM1 (Utah大学)

Davisの手によって開発されたUtah大学のDDM1は，高度にmodular化され，再帰的に構成されたデータフロー計算機である。DDM1はデータ駆動の原則にもとづいたプログラムを処理でき，パイプライン的処理および高度の並列処理を実現できる。DDM1は充分に分散された非同期多重処理を行い，規模を拡大する際にも，何ら物理的制約を課さない。その上，タスクの物理的な割り付けを自動的にかつ動的に行うため，システム拡充の際のソフト的な配慮も不要である。

ハードウェアは大きく分けて，互いに非同期に演算実行を行えるProcessor-Module とそれ

らをつなぐ通信moduleとから成っている。基本構成unitはPSE（Processor Store Element）と呼ばれ，processor module（P）と局所storage module(S)とからなり，どのPSE単体もDDN(Data Driven Net：DDM 用のデータフロープログラム）で書かれた機械語を実行できる。全体の構成はこのPSEを用いて再帰的に定義されBNFでは次のように示すことができ

＜PSEn＞：：＝＜Pn＞＜Sn＞

＜Sn＞：：＝＜ASUn＞

＜Pn＞：：＝＜APn＞1＜APn＞＜PSEGROUPn＋1＞

＜PSEGROUPn+1＞：：＝＜PSEn+1＞1＜PSEn+1＞＜PSEGROUPn+1＞

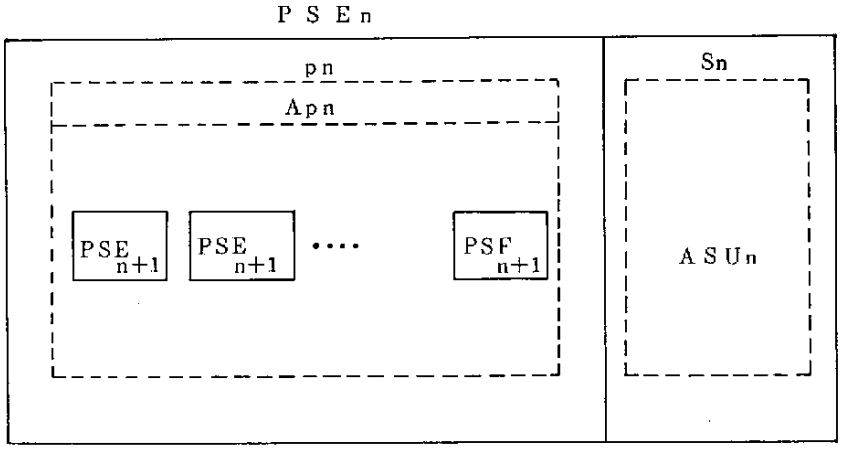


図3.2.8　PSE（第nレベル）の再帰的定義

この構成は，階層化した分散記憶構成に適したものであり，Sn, ASUnにおける記憶の機能は，任意の媒体を用いて，実現できる。PSEの上位レベル（添字が小）のものは，より多くのsub-structureを有するため，並列機能が高い。それゆえ下位のものに比べてaccess時間が長く容量の大きい媒体を用いるようにすれば，効率が良く，matchingがとれた，よいシステム 設計といえる。

下位PSEから上位PSEへのmessageはスイッチによって任意に送られる。それに対して上位PSEから下位PSEへ送る場合は，headerを付加してゆき先を指定し，スイッチ部分で選択を行う必要がある。

各PSE間にはキューが存在し，パイプライン制御を援助し，各モジュールの独立性を高める。キューの容量をover しそうになると送出側はmessageの伝達を一時停止し，空きが生じるまで

待たなければならない。

上位のPSEから"仕事"をもらったPSEは，その"仕事"の中に並列性を見出すことができ，しかも下位のPSEがidleであることがわかれば，"仕事"を分解して，それぞれ下位のPSEにわたす。下位のPSEにわたさない場合は，当該PSEで実行し，その結果を必要とするPSEに必要なだけcopyして転送する。これらの動作を再帰的に繰り返し行うことにより，DDNを実行してゆく。

DDM1の長所としては，

(1) 並列処理，pipeline処理による性能の向上

(2) 分散制御によるoverheadの分散化，信頼性の向上

(3) あらゆる構成レベルにおいて同一の構造をとるため，modular化，LSI化が容易で，機能の拡張・追加が，特別なsoft，hardの手直しを加えることなく容易にできる。

(4) Memory構成がflexibleで，任意のASUに，任意の媒体を用いることができる。

(5) Processorの冗長構成により障害の庶蔽効果がある。

があげられる。逆に短所としては，

(1) hard的には，固定された木構造であるため他のPSEが下位レベルのPSEを必要とする場合でも，不要のPSEを供出することができない。

(2) 効率を充分発揮させるためには，かなり冗長な（というよりむしろ無駄なくらいの）量のhardware を必要とする。

—参考文献—

1) Davis , A.L . "System Aspects of Data − Driven Nets (Data-Driven Computation Part II)" Burouughs IRC Report , Feb. 1975 .

2) Davis , A.L. "Structured Data(Data − Driven Computation Part III) " Burroughs IRC Report , Mar. 1975 .

3) Davis , A.L . "Data − Driven Net Queueing Phenomenon (Data − Driven Computation Part VII) " Burroughs IRC Report , Tan . 1975 .

4) Barton , R . S . , Davis, A.L., et al "System and Method for Concurrent and Pipeline Processing Employing A Data Driven Network " U.S. Patent , no.3, 978, 452 , Aug.1976 .

5) Davie , A.L . "An Overview of Data − Driver Machine #1 " Technical Report , Burroughs ASDO , July. 1976

6) Davis , A.L . "The Architecture of DDMI : A Recursive Structured Data

Driven Machine " UUCS−77−113, Oct. 1977.

7) Davis, A.L. "The Architecture of DDMI : A Recursively Structured Data Driven Machine " Proc.of the 5th Annual Symposium on Computer Architecture, pp. 210−215 , 1978.

8) Davis, A.L. " Data Driven Nets : A Maximally Concurrent , Procedural , Parallel Process Representation for Distributed Control Systems " UUCS − 78 − 108, Jnly. 1978.

9) Davis, A . L . " A Data Flow Evaluation System Based on the Concept of Recursive Locality " Proc . of 1979 National Computer Conference , vol.48 , PP. 1079−1086, 1979.

10) Glushkov , V.M., et al " Recursive Machines and Computing Technology "Proc . of IFIP Congress 1974 , pp.65−70, 1974.

3.2.6 Manchester大学のデータフローマシン

このマシンはManchester大学のJ.Gurd, I. Watson等によって開発されている[1)2)3)]

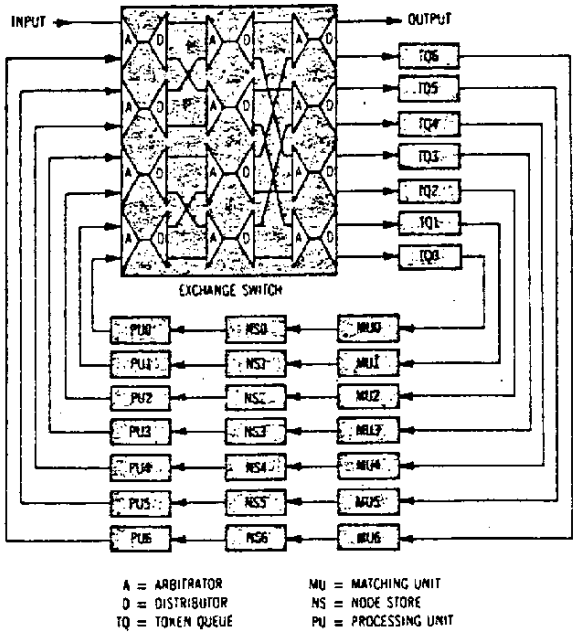


図3.2.9 Manchester大学のデータフローマシン

マシンは図3.2.9のようにサーキュラパイプラインを多層化した構造をなしている。各パイプラインのステージは，トークンのバッファリングを行うtoken queue，ノードの発火制御を行うmatching unit，プログラムを格納するnode store，命令を実行するprocessing unit，および，層間や入出力装置への通信手段を提供するswitchから構成される。

このマシンは主として数値計算の並列実行に主眼をおいており，その主要な特徴を以下にあげる。

(1) 非同期通信

各unit間の通信は非同期なハンドシェイク法で行っている。従って各unitは独立な内部クックで動作できる。

(2) 並列ハッシュによる発火検出

命令の発火検出を行うために，matching unit内に8バンクから構成される並列ハッシュ表を用意している。各バンクのメモリ容量は2K words × 96bitsである。さらに，ハッシュ表のオーバフローが発生したときのためにoverflow unitを用意している。トークンあたりのmatching unit内での処理時間は200nsと想定している。

(3) コードのセグメント化

命令アドレスをセグメント番号（6bits）とセグメント内の変位（12bits）で表わしコードの再配置を可能にしている。node store内には64エントリから成るセグメント表があり，実際のコードのアドレスはこのセグメント表から求められるベースアドレスとセグメント内の変位を加算して得られる。

(4) マイクロプログラム制御の処理装置

柔軟性を考慮して処理装置としてはマイクロプログラム制御のビットスライスマイクロプロセッサAMD2900シリーズを使用している。処理要素はこのプロセッサ6個（24ビットデータ幅）から成る演算装置とし，その平均命令実行時間を4.5μSと想定している。従って，matching storeの処理速度（命令あたり300ns）とのバランスを考慮して，processing unit内にはこの処理要素が15台並列に配置されている。

(5) 均質な層間通信

各層の間や層と入出力装置の間の通信は，バイナリルータを組み合わせた構成となっているswitchが行う（図3.2.9を参照）。従って，各層間の通信遅延は均一になっている。

(6) 色つきトークン

トークン長は96ビットであり，そのうち36ビットを色（識別子）フィールドとして割り当てている。このフィールドはさらに，関数識別子，繰り返し番号，およびインデックス識別子に分けられている。前2者は，例えばArvindのマシン4)と同様であるが，インデックス識別子は

配列要素の並列アクセス等のために使用される。

(7) 単一代入言語

前述のマシン開発と並行して，LAPSEと呼ばれる単一代入の高水準言語を設計し，そのコンパイラを開発した5)。この言語はPASCALをベースに単一代入規則を導入したもので，条件式，繰返し，（再帰）関数呼出し等の機能を備えている。また，データタイプとしては整数や論理値の外に，配列やレコード型も用意されている。

(8) 非決定的処理

記憶ノードの概念を導入することにより，Dijkstraのguarded command 6)やHoareのCommunicating Process 7)のような非決定的な制御方式についても研究している8)9)。記憶ノードは色の制御命令とmatching store内のマッチング機能の追加によって実現している。

このマシンの問題点としてはリストのような動的データ構造の機構が用意されていないということがあげられる。従って，知識情報処理に適用するためには構造メモリを付加する必要がある。

現在，1つのサーキュラパイプラインから成る実験機の試作・評価を行っている。また，データフロー言語コンパイラも稼動中である。今後，さらに，高機能な言語や多層構造マシンの開発も予定している。

—参考文献—

1) J.R.Gurd, I.Watson, "Data Driven System for High Speed Parallel Computing," Computer Design, July 1980.

2) J.R.Gurd, J.R.W.Glauert, "A Multilayeved Data Flow Computer Architecture," Dept.of Computer Science, University of Manchester, July 1978.

3) I.Watson, J.R.Gurd, "A Prototype Data Flow Compuputer With Token Labelling," Proc.of AFIPS NCC VOL.48, June 1979.

4) Arvind, V.Kathail, "A Multiple Processor Data Flow Machine that Support Generalized Procedures," Proc.of 8th Symposium on Computer Architecture, May 1981.

5) J.R.W.Glanert, "A Single — Assignment Language for Data Flow Computing," MSc. Dissertation, Dept of Computer Science, University of Manchester, Jan. 1981.

6) E.W.Diikstra," Guarded Command, Nondeterminacy and Formal Derivation of Programs," Comm.of ACM, Vol.18, №8, Aug.1975.

7) C.A.R.Hoare , "Communicating Sequential Processes", Comm.of ACM, Vol. 21, №8 , Aug.1978 .

8) A.J.Catto,J.R.Gurd, "Nondeterministic Data Flow Graphs", Proc.of IFIP Congress 80, North-Holland Publishing Company , Oct. 1980 .

9) A.J.Catto, J.R Gurd, "Resource Management in Data Flow ", Proc.of Functinal Programming Languages and Computer Architecture, Oct.1981 .

3.2.7 Kellerマシン

Utah 大学のR . M . Keller, G. Lindstrom , S.Patil らによって提案された要求駆動型のデータフローマシンである。[1)2)] 要求駆動型の発火機構では，命令はその実行結果が必要とされる場合（他から要求がある場合）にのみ発火するという特徴がある。

このマシンは図3.2.10に示すように，木構造を形成するノード群からなっている。台形のノードは通信用プロセッサで，調整／分配（ arbitration and distribution ）の働きを行う他に，配下のプロセッサの実行状態に応じて負荷の均衡を行う機能を持つ。円で示される終端のノードは汎用の処理ユニットで，それぞれがローカルメモリを持っている。これらのローカルメモリは，全体で1つのアドレス空間を提供する線形アドレスメモリとして機能することができる。場合によっては処理ユニットとしてI／Oプロセッサや特殊用途のプロセッサが接続される。これらの処理ユニットは要求駆動の原理にもとづいて動作し，要求元のユニットに必要な情報を送出する場合は，木の調整／分配ノードを経由して目的のユニットに送られることになる。

要求駆動型のプログラムはEGL（Flow Graph Language ）と呼ばれるLisp方言で書かれる。プログラムは code block および data block からなっている。 code block にはFGLのコンパイル結果（データフローグラフの線形表現）が保持される。ここでは， instruction コードは実行すると消費されデータに変化するという設計方針がとられているので，関数の application を行う際，新たに data block が割り当てられ，そこへ source code がコピーされる。図3.2.11 は car （cons （ cdr(u) , v) ）に対する data black内の初期コードを示したものである。 data block内の各エントリは literal value か instruction かのいずれかである。

instructionは，a. operation code b. demand （オペレータの引数のアドレス）c . notifier （この命令の値を待っている命令のアドレス）d.global . address（block間のリンケージのために使用される）等から構成される。プログラムの各行には参照用番号が付されている。例えば，このプログラムの第2行

```
cons d3 , d5 n1
```

において，d3，d5は「3および5の番号のついた命令にオペランドを要求せよ」ということを，またn1は「1の番号のついた命令に（consの）実行結果を通知せよ」ということを意味している。

要求駆動型の特長は，引数の値が必要になった時点で要求を出すので，不要な計算を省くことができること，要求駆動と遅延評価（cons(x，y）の評価の際，引数x，yの評価を行わず先にセルだけを返す）の機能を組み合わせて無限リストの操作がうまく扱えることにある。しかし実行効率という観点からみると，要求が次々と最内側まで（値が得られる所まで）伝搬されねばならないので，このオーバヘッドが問題となる。

Kellerマシンでは，(1)木構造を利用して最短経路の通信プロセッサを用いて通信を行うことができる。(2)関数対応に参照があることを前提に（従来のLispインプリメンテーションを念頭に置いているためであろう），ローカルメモリ構成としている。(3)動的アロケーションを行う，などの特徴を有しているが，いまだ理論的提案の段階にあり不明な点が多い。現在の研究はマシンよりも適用性型言語ＦＧＬの理論的側面に向けられている。3)

一参考文献一

1) Keller.R.M.，Lindstrom G.and Patil S.，"An Architecture for a Loosely－coupled Paralled Processor,."UUCS－78－105 Univ.of Utah，1978.

2) Keller R.M.，et al，"A Loosely-coupled Applicative Multi-Processing System", Proc.Nat.Comp.Conf.，1978.PP.861—870

3) Keller R.M.，"Semantics and Applications of Function Graphs"，Technical Report (MCS—77—09369), Univ.of Utah，March 6，1980.

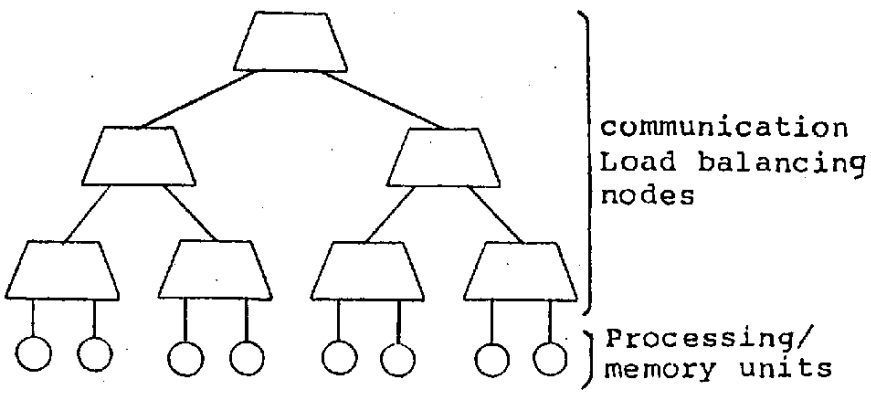


図3.2.10　Kellerマシン

car(cons(cdr(u),v))

1	car	d_2	n_0
2	cons	d_3, d_5	n_1
3	cdr	d_4	n_2
4	u		n_3
5	v		n_2

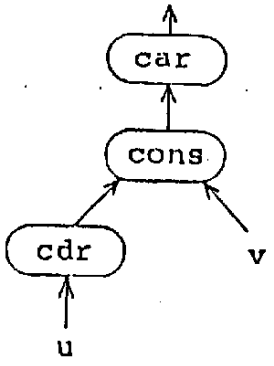


図3.2.11　グラフ表現とdata block内の初期コード

3.2.8　TOPSTAR（東京大学）

1977年頃鈴木等によって開発が始められ，1978年に第1号機であるTOPSTAR−Iが，1980年にTOPSTAR−Ⅱが栗原等によって試作された。TOPSTARはマルチマイクロプロセッサ システムを，プロシジャレベルのデータフローによって分散制御することをめざしたものである。

その構成は，処理要素であるPM（Processing Module）と通信および制御ノードであるCM（Communication／Control Module）の2種類のモジュールが図3.2.12のように結合したものである。データフロー処理単位（これをアルゴリズムと呼ぶ）ごとにPMが実行し，これらの処理単位間のデータの流れやタスク（＝アルゴリズム＋データ）の生成などをCMが制御している。

アーキテクチャの特徴は

(1) CMとPMの2種類のモジュール多数より成り，それぞれがペトリネットの"place"と"transition"の役割りを果たす。（図3.2.13）

(2) CMとPMの結合は大規模システムの実現を可能にするため部分結合とする。この際データフローグラフとの対応を柔軟に行うために，この結合をオーバラップさせ可変構造を可能にしている。

(3) プロシージャレベルのデータフロー処理においては比較的多量のデータを扱うので，DMAによってメモリ対メモリの高速転送を行う。

各PMは，結合している範囲の1つのCMに割り込みをかけ，DEQ（DEQ ueue）コマンドを送ることにより"仕事"を要求する。各CMには，それぞれいくつかのノードが割り当てられており，DEQコマンドを受けたCMは制御テーブルを調べて，実行可能な"仕事"があれ

ばそのプロシージャ，データ，結果の送り先の情報をＰＭに送る。実行可能な〝仕事〟がない場合には，その旨を知らせる。〝仕事〟をもらったＰＭはその〝仕事〟を実行し，結果の送り先のノードが割り当てられているＣＭに割り込みをかけ，ＥＮＱ（ＥＮＱ ueue）マンドを送り，ひき続いて結果のデータを送る。アイドル状態になったＰＭは再び〝仕事〟を求めてＣＭに割り込みをかける。

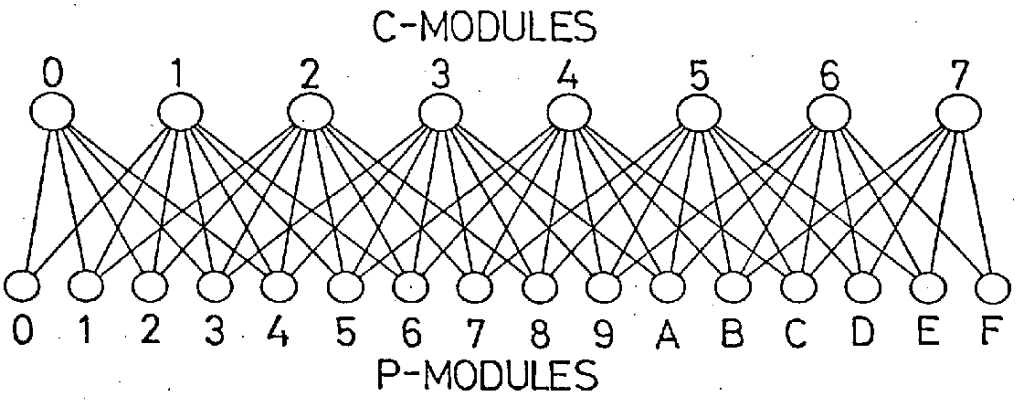


図3.2.12　ＴＯＰＳＴＡＲ－ⅡのＣＭ－ＰＭ間結合

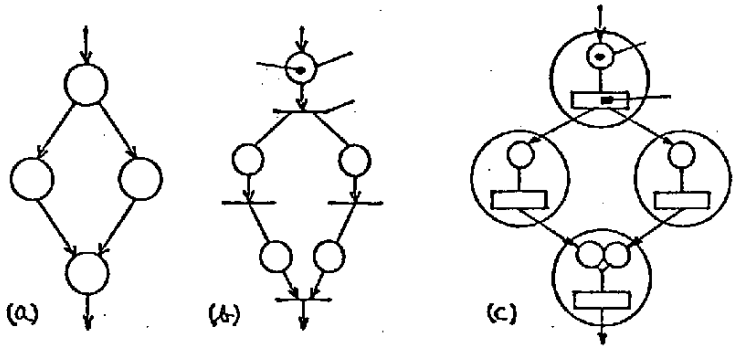


図3.2.13　データフローグラフ，ペトリネットおよびTOPSTARにおける処理の関係

以上のように，各ＰＭが"仕事"を求めて（部分結合の範囲で）自由競争を行うことにより，"忙しい"すなわち，実行可能な"仕事"が多く存在するＣＭに，自然に多くのＰＭが集まり，適切な負荷分散が行われる。

マルチプロセッサシステムにおける並列処理性には，パイプラインによる縦の並列性と独立した演算が並行して実行できる横の並列性が存在する。一般のパイプラインでは，ＦＩＦＯによる制御が行われているが，データによって処理時間が大きく異なるような場合には，追い越しを許すことにより，並列処理性の向上が期待できる。しかし，無制限な追い越しを許すと，FORK−JOINの際にデッドロックを生じる可能性があるため，セマフォによるフローコントロールを行う必要があり，TOPSTARではこれを実装している（図3.2.14）。

すでに，応用プログラムとして，論理回路シミュレーション，並行故障シミュレーション，並列ＰＲＯＬＯＧ等が実装され，評価が行われた。

問題点としては

① データフローグラフの各ノードを，どのＣＭに割り付ければ効率があがるか→（準）最適割り付けの問題。

② メモリ容量の不足と結合範囲の狭さ。

③ （データフロー一般の問題でもある）大きな構造データを扱う効率上の問題，歴史依存性，ファイル，データベースの扱い等のいわゆる「記憶」の問題。

④ 障害の検出あるいは回復の問題。

等があげられる。

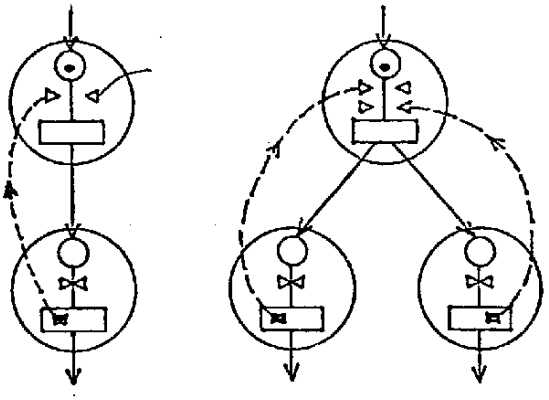


図3.2.14　セマフォによるフローコントロール

―参考文献―

情報処理学会全国大会

1) 元岡，喜連川，鈴木「A High Level Data Flow Machine － Hardware －」1979

2) 元岡，鈴木，喜連川「A High Level Data Flow Machine －Software －」1979

3) 鈴木，田中，元岡「データフローマシンＴＯＰＳＴＡＲの制御方式とその評価」1980

4) 栗原，鈴木，田中，元岡「データフローマシンＴＯＰＳＴＡＲのデータ駆動型制御方式」

5) 深沢，栗原，鈴木，田中，元岡「データフローマシンＴＯＰＳＴＡＲによる論理回路シミュレーション」1980

6) 駒田，鈴木，田中，元岡「データフローマシンＴＯＰＳＴＡＲによる並列ＬＩＳＰシステム」1980

7) 栗原，深沢，秋元，鈴木，田中，元岡「データフローマシンＴＯＰＳＴＡＲ－Ⅱのハードウェア構成と基本ユーテイリテイ」1981

8) 秋元，深沢，栗原，鈴木，田中，元岡「データフローマシンＴＯＰＳＴＡＲ－Ⅱによる論理シミュレーション」1981

9) 深沢，栗原，秋元，鈴木，田中，元岡「データフローマシンＴＯＰＳＴＡＲ－Ⅱの性能測定と評価」1981

10) 鈴木，田中，元岡「データフローマシンＴＯＰＳＴＡＲ－Ⅱにおける並列ＬＩＳＰシステム」1981

11) 相田，松方，鈴木，田中，元岡「データフローマシン向きＰＲＯＬＯＧ型言語に関する一考察」1981

12) 中田，田中，元岡「データフローマシン〝ＴＯＰＳＴＡＲ－Ⅱ〟によるコンカレント故障シミュレーション」1981

13) 相田，田中，元岡「データフローマシン〝ＴＯＰＳＴＡＲ－Ⅱ〟を用いたＰＲＯＬＯＧ並列処理試作システムの検討」1981

14) 相田，田中，元岡「並列ＰＲＯＬＯＧシステム〝Ｐａｒａｌｏｇ〟の性能測定」1982
（寄稿中）

15) 萩野，田中，元岡「データフローマシンＴＯＰＳＴＡＲ－Ⅱの動作解析と評価」1982
（寄稿中）

電気学会全国大会シンポジュウム

16) 元岡，鈴木「ＳＡＭＤ（Single Algorithm-stream Multiple Data-stream)」1979

研究会

17) 鈴木，田中，元岡「パイプライン式ＳＡＭＤ計算機とその漢字認識への応用」信学技報ＥＣ７８－２，１９７８年４月

18) 元岡，鈴木，喜連川，新岡「ＳＡＭＤ計算機～A High Level Data Flow Machine～」情処，計算機アーキテクチャー研，１９７９年５月

19) 栗原，鈴木，元岡「High Level Data Flow Machine（ＴＯＰＳＴＡＲ）のシステムプログラム」信学技報ＥＣ７９－５６，１９８０年１月

20) 鈴木，元岡「データフロー計算機の制御と評価」情処，計算機システムの解析と制御研，１９８０年２月

21) 深沢，田中，元岡「Multi Processor Simulator」電子装置設計技術研究連絡会　夏期シンポジウム研究発表資料

22) 深沢，栗原，鈴木，田中，元岡「データフロー計算機による論理シミュレータ」電子装置設計技術６－３　１９８０年１０月２１日

3.2.9 D^3C（東京大学）

D^3Cは１９７９年後半から開発されたデータフローマシンで，浅田，クエン等によって試作された。アーキクチャ上の特徴は，「分散形・ループネットワーク，非同期演算実行システム」ということができる。分散型とは「演算の実行の際に，ハードウェアユニットの選択が何らかの形である程度制御可能」なもので，概念図を示すと図３.２.１５のようになる。すなわち各構成モジュールは，メ

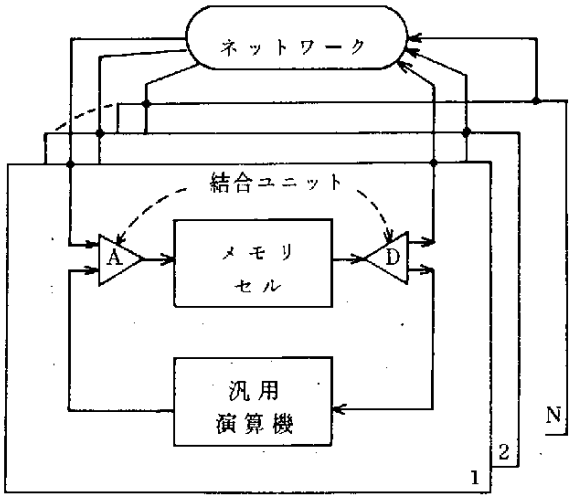


図３.２.１５　分散形モデル

モリセル・演算・結合の各ユニットを有し，それぞれ単体でデータフロー計算機をなしており，その構成モジュールをネットワークによって結合して，より高性能のシステムを目ざしている。ネットワークとしてはループネットワークを採用しているが，システム拡張性を考慮にいれた結果であり，実用化にあたってはモジュール数を数十以上接続することを想定している。ループネットワークの欠点としてトータルシステムの信頼性の低さが考えられるが，①各ノードに開閉容易なバイパススイッチを設ける　②ノード間結合はフォトカプラとする，③1ノードの停電時には，電磁リレーで強制的にバイパスされる　等の手法を実験システムにも実装し，信頼性の向上をはかっている。

「分散型」にした1つのねらいはこの「信頼性向上」にある。集中型の場合には，演算ユニットが完全な自由競争の原理で実行可能な演算をとりこんで実行するため，どの命令がどの演算装置で行われたかを知ることが容易ではなく，故障個所の検出は極めて困難である。それに対して各アクタに対して，どの構成モジュールで実行すべきかをある程度制御（監視）できれば，故障の検出，さらには故障モジュールの切断が可能であり，アクタを再配置することにより，実行の再開も可能である。

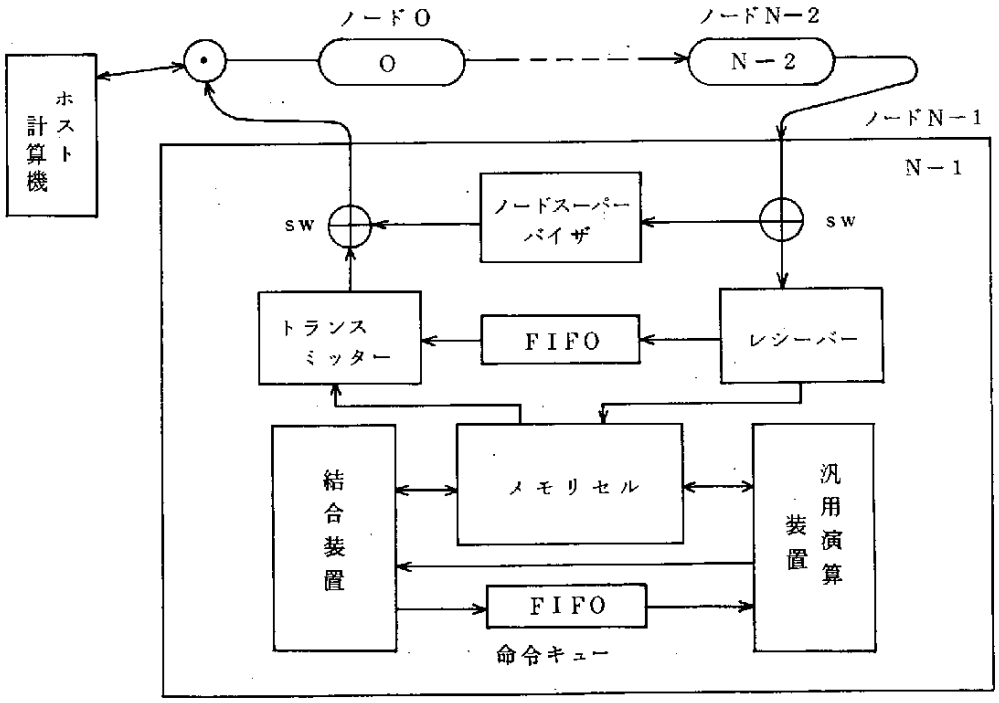


図3.2.16　D^3Cの概念構成図

ユーザのプログラムから並列性をとり出してスループットをあげるという一般的なデータフローマシンの目的の他に，ユーザプログラムのcopyを異なる構成モジュールで走らせ（原ユーザプログラムと並列に走る），結果を照合することにより，スループットをさほど下げることなく，信頼性の向上をはかることも目指している。

問題点としては，発火時の自由競争にある程度の制約を加えることにより，上記の目的を達成しているため，アクタの各構成モジュールへの割り付け方がスループットを大きく左右すること，ネットワーク構成（ループ結合）に内在する課題などがあげられる。

現在，グエン，大山等によるバス結合分散形データフロー計算機「ＥＤＡＣ」が開発中であり，シミュレーションを通して，各種設定パラメータが検討されつつある。

—参考文献—

1）浅田，グエン・ニユット，堀，斉藤，猪瀬，“データフロー計算機における故障検出の一方式”Ｓ５４情報処理分散処理システム研究会１－４

2）堀，浅田，斉藤，猪瀬“データフロー計算機における故障検出の一方式”Ｓ５４信学全大情報システム部門３６０

3）グエン・ニユット，浅田，斉藤，猪瀬“分散形データフローコンピュータの構成”Ｓ５４信学全大情報システム部門３５９

4）浅田，グエン・ニユット，堀．斉藤，猪瀬“分散形データフロー計算機の故障検出方式の評価”Ｓ５４信学全大情報システム部門３５８

5）浅田“計算機複合体における故障診断修復方式に関する研究”東京大学博士論文Ｓ５５

6）池田,“分散型データフロー計算機におけるアクタの割り付け”東京大学卒業論文Ｓ５５

7）グエン・ニユット，大山，斉藤，猪瀬，“高性能分散型データフロー計算機の制御機能”Ｓ５６信学全大情報システム部門５１０

8）大山，グエン・ニユット，斉藤，猪瀬，“高性能分散形データフロー計算機の通信機能”Ｓ５６信学全大情報システム部門５１１

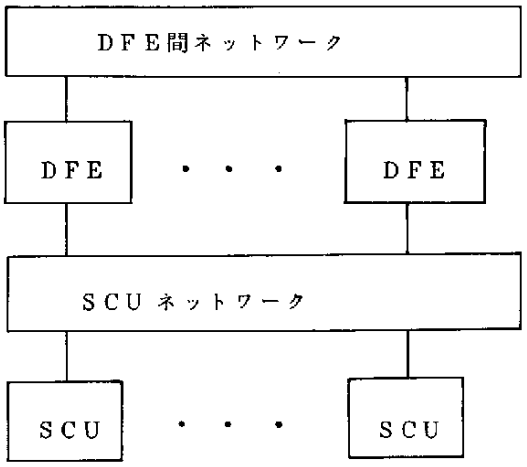
9）大山，斉藤，猪瀬，“分散型データフロー計算機のプログラムモデルを用いた性能評価”信学技報ＥＣ８０－１４　１９８０年６月

10）大山，斉藤，猪瀬“分散形データフロー計算機（ＥＤＡＣ）の待行列モデルによる理論解析”Ｓ５７信学全大情報システム部門寄稿中

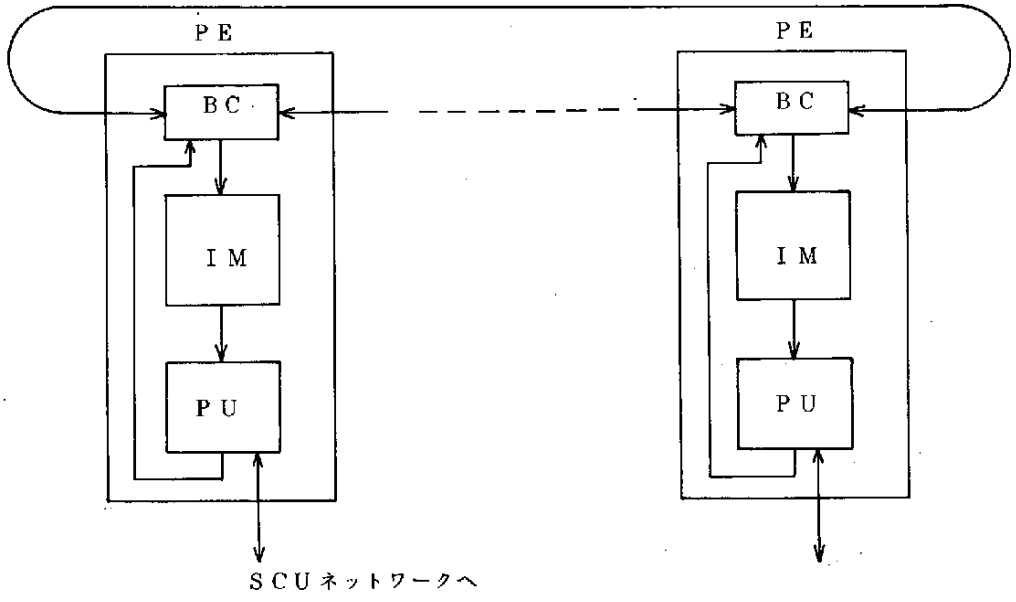
3.2.10　D^3P（沖電気工業）

沖電気の伊藤，来住，安原によって開発されているデータフローマシンである[1) 2) 3)]

マシンは図3.2.17のように，関数クラスタ（関数又は関数の集合）を実行するDFE群，構造メモリの格納，管理，操作を行うSCU群，およびそれらの装置間の結合ネットワークから成る。



（a）　D^3Pの構成



（b）DFEの構成

図3.2.17　D^3Pの構成

ＤＦＥ内はさらに，サーキュラパイプライン構造のＰＥ群，色（アクティベーション番号）の割り当て管理を行うＡＣＵ，およびＰＥ間の結合ネットワークを形成するリングバスから構成される。各ＰＥ内のパイプラインの各ステージは，コードを格納し，命令の発火制御を行うＩＭ，命令を実行するＰＵ，リングバスのトークンの入出力を制御するＢＣから成る（同図(b)）。

本マシンの特徴は以下の通りである。

A. 関数クラスタの並列実行

各ＤＦＥはそれぞれ独立に関数クラスタを並列実行すると共に，関数クラスタを適当なアルゴリズムによりＤＦＥ内のＰＥ群にマッピングすることで，クラスタの並列性も生かす。このマッピングアルゴリズムとしては，ノードＡの出力が２つ以上のノードＢ，Ｃ，Ｄ，……に渡されるとき，うち１つのノード（例えばＢ）をノードＡと同一のＰＥに，それ以外のノード（Ｃ，Ｄ，……）を順次ノードＡの隣接ＰＥに群に割り当てる方式を採用した。

関数やループの起動はトリガートークンを用いた先行制御方式とした。

B. 統合化された色

関数識別子とループ（繰り返し）識別子を区別せず統合化された色を使用している。また使い終えた色を解放するプリミティブも用意されている。これによって色空間の効率的な使用が可能となる（色フィールド長が比較的小さくてよい）。

C. ハッシュとタグによる発火検出

命令のオペランドが揃ったか否かを検出（発火検出）するために，ハッシュ表およびオペランドタグを組み合わせたハードウェア構成にしている。即ち，色フィールドをも含めたアドレス空間（各ＰＥ当たり 2^{21} 語）を１６語単位のブロックに分割し，このブロック単位で物理アドレス（各ＰＥ当たり 2^{12} 語）に変換するためにハッシュ表を使用している。実際の発火検出はこの物理アドレスをもとにオペランドタグを読み出すことによって行われる。

D. 構造データの非同期アクセス

構造データの非同期アクセス機能を実現するために，ＳＣＵメモリ内の各セル毎にＷタグとＲタグと呼ばれる２つのフラグが存在する。Ｗタグは該セルへの書き込みが行われたか否か（即ち，セル内のデータが有効か否か）を示し，Ｒタグは該セルに対して読み出し要求があったか否かを示す（要求があったときは，ＳＣＵ内の待ち行列にチェインされる）。この機構によってlenient consやstreamのような操作をサポートできる。

E. 可変語長データ

オペランド語長は１６ビット（基本語長）から６４ビット（４倍語長）まで可変である。このため浮動小数点や短いストリン等も基本データとして処理できる。

問題点としてはＳＣＵ内のガーベッジコレクションの実現方法（現在はヒープ管理法）の検討，ＤＦＥ間やＳＣＵへのネットワークの設計・評価および高水準言語の設計等が残されている。

現在，４台のＰＥから成るＤＦＥの実験システムを試作し，調整中である。ＩＭおよびＰＵはマイクロプログラム制御であり，ＰＵのＡＬＵとしてはビットスライスマイクロプロセッサ2900シリーズを用いている（ＳＣＵおよびＡＣＵの機能は各ＰＵ内でエミュレートする）。

－参考文献－

1) 来住，伊藤，安原，河村，「データフロー実験機の構成」，昭和５７年信学会全国大会，１９８２年３月。

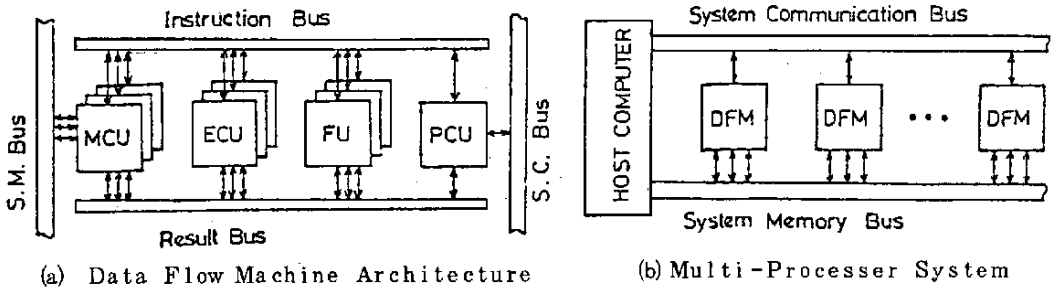
2) 伊藤，来住，安原，河村，「データフロー計算機D^3Pのアーキテクチャ」，情報処理学会，計算機アーキテクチャ研究会，№４２，１９８１年７月

3) 安原，伊藤，来住，「データフロー計算機D^3Pのアーキテクチャ」，沖電気研究開発，Ｖｏｌ，４７，№２，１９８０年１２月。

3.2.11　東北大学のデータフローマシン

このマシンは東北大学・工学部の伊藤，斉藤，星子等によって研究・開発されている。[1),2)]

図3.2.18にその構成を示す。マシンは，トークンの同期制御や命令フェッチを行うＥＣＵ群，命令の実行制御を行うＦＵ群，構造データの記憶，管理，操作を行うＭＣＵ群，プロセス（関数呼出し）のスケジューリングを行うＰＣＵ，およびこれらの装置間を結ぶ２つのバスから構成される。さらに，このマシンを複数結合したシステムも検討している（同図(b)）。



(a) Data Flow Machine Architecture　　(b) Multi-Processer System

図3.2.18　東北大学のデータフローマシン

本マシンの特徴は以下の通りである。

A. 関数性の徹底

データフローマシンにおける意味付けのあいまいなループ制御やTrueゲート，Falseゲート等のオペレータを排除し，命令レベルにおいても関数性を徹底させている。

B. 縮約過程にもとづく計算

コピー規則にもとづいてプログラムの実行過程を縮約（reduction）の過程として実現している。即ち，ＥＣＵには連想機能を有する命令レジスタ群が存在し，先着オペランドが現われた時点で（即ち，命令レジスタ群を連想検索して対応する命令がないことが確認されたとき），その命令がコードメモリよりフェッチされ，レジスタ内の空きスロットに格納（コピー）される。このレジスタの内容はその命令の全オペランドが揃った時点で読み出されて無効にされる。

C. 多入力オペランド命令

命令レジスタの各エントリにはＥＣ（Enabling Count）と呼ぶカウンタが存在し，オペランド到着毎にデクリメントされる。この値は先着オペランド到着時に初期設定され，値がゼロになると全オペランドが揃ったことになり，命令は実行可能となる。多入力オペランド命令は主として多入力引数を有する関係呼び出し用に使用される。

D. 柔軟なデータ構造

ＭＣＵはリスト構造や配列等の柔軟なデータ構造をサポートする。このため参照カウント方式を採用し，メモリの動的管理はＭＣＵ側で行う。さらに，データにタグを付与し，実行時の正当性チェックも行っている。

本マシンの問題点としては関数の起動をその全引数が揃うまで待つために並列度が低下する恐れのあること，各装置間の通信を共通バスによって行っているので接続される装置数に制限があること，およびハードウエア構造が複雑になると予想されることがあげられる。

現在，マシンのエミュレートを行うためにビットスライスマイクロプロセッサから成る４台の汎用ユニットを作成し，そのファームウエアおよびソフトウエアの開発を行っている。

—参考文献—

1) 伊藤，斉藤，星子，「関数型データフロー計算機システム」，信学技報ＥＣ－３１，１９８１年１０月，

2) 伊藤，斉藤，星子，「関数型データフローマシンのシステム構成」，情報処理学会第２２回全国大会，１９８１年３月，ＰＰ．５５～５６．

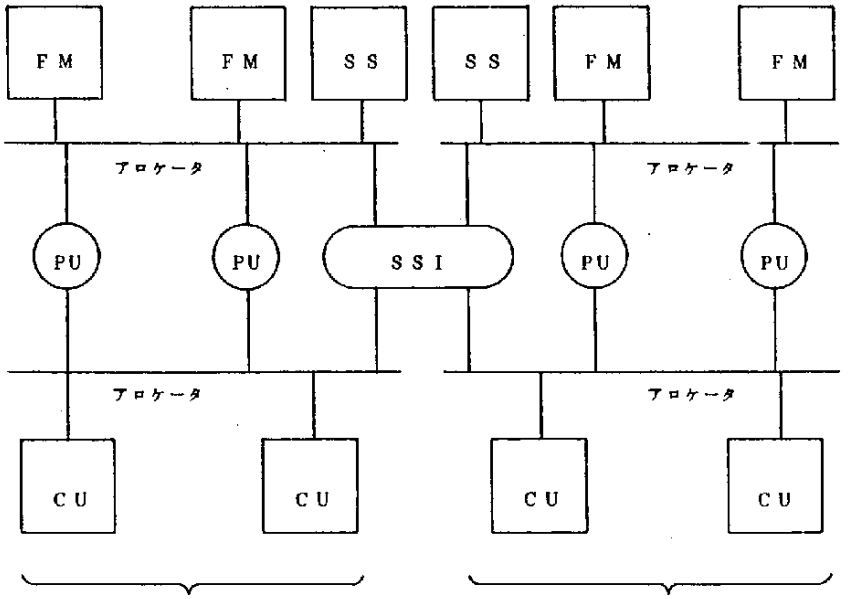
3.2.12 Passive Memoryless Architecture

A. 研 究 者 ： 寺田浩詔，浅田勝彦，大阪大学

B. 特　　徴

受動的でメモリをもたないタスクレベルでの処理機能をもつ多数の処理ユニット（ＰＵ：Processing Unit）と能動的でデータフローグラフをもちデータ駆動制御を行う多数の制御ユニット（ＣＵ：Control Unit）とそれらを選択的に結合するアロケータからなる。

アロケータはモジュールの追加・除去が容易であり，適当なインタフェースを介することにより，階層化，多重化等の種々の構成をとることができる。２つの非同期２線式自己診断リングアービタと共通バスで構成されており，２つのアービタはそれぞれ能動モジュール，受動モジュールでのアービトレーションを行い，バスは結合の確立されたユニット間でのデータ転送に使われる。アロケータによる選択的結合は常に能動的モジュールであるＣＵの要求にもとづき，ＣＵの出すセレクションコードにより行われる。セレクションコードは，ＣＵよりＰＵへのオペレーションの転送のときは，ファンクション名，ＰＵよりＣＵへのトークン転送のときは，目的ノードとカラーである。



ＳＳＩ：システムシステムインターフェース
ＳＳ　：入出力サブシステム
ＦＭ　：機能メモリ

図3.2.19 PASSIVE MEMORYLESS ARCHITECTURE

ＣＵにおけるタスクの発火制御は次のようになっている。ＣＵ中のあるタスクノードに最初にトークンが到着すると，ＣＵは発火に必要な残りのすべてのトークンをＰＵに要求する。そして，すべてそろった後，オペレーションパケットを作り，ｆｒｅｅでかつそのオペレーションを処理できるＰＵへアロケータを通じて転送する。ＰＵは結果を生成するとそれを要求しているＣＵがあるかどうかを見，あればそれを転送する

C. 問題点

アロケータに要求を出せるのがＣＵだけであるため，ノードに最初のトークンが届いてから最後のトークンが届き，オペレーションパケットを転送するまで，１つのＣＵが完全に占有されてしまい，ＣＵの利用効率が悪い。

D. 現　状

試作機を作り，評価中である。

参考文献

1) 寺田浩詔，浅田勝彦，“Passive Memoryless Architecture,”情報処理学会アーキテクチャ研究会31-2，1978

2) 浅田勝彦，小谷武史，寺田浩詔，“Passive Memoryless Architecture の一実現法，”信学技報　ＥＣ７９－７６，1980

3) 浅田勝彦，寺田浩詔，他，“Passive Memoryless Architecture 上でのデータフロー形実行制御，”信学会技報ＥＣ８０－４９，1981

3.2.1 3　富士通のデータフローマシン

A. 開発時期 ： １９８１年

B. 開 発 者 ： 富士通研究所

C. 特　　徴

データフローの原理を，実マシンの試作を通じて把握することを主目的としたマシンであり，命令レベルでの並列処理を可能としている。アーキテクチャとしては，Dennis型のセルブロックが１個でファンクションユニットが複数の場合とやや似た構成と考えられる。システム構成を図3.2.20に示す。〔ＭＡＳＵＺＡＷＡ８１〕

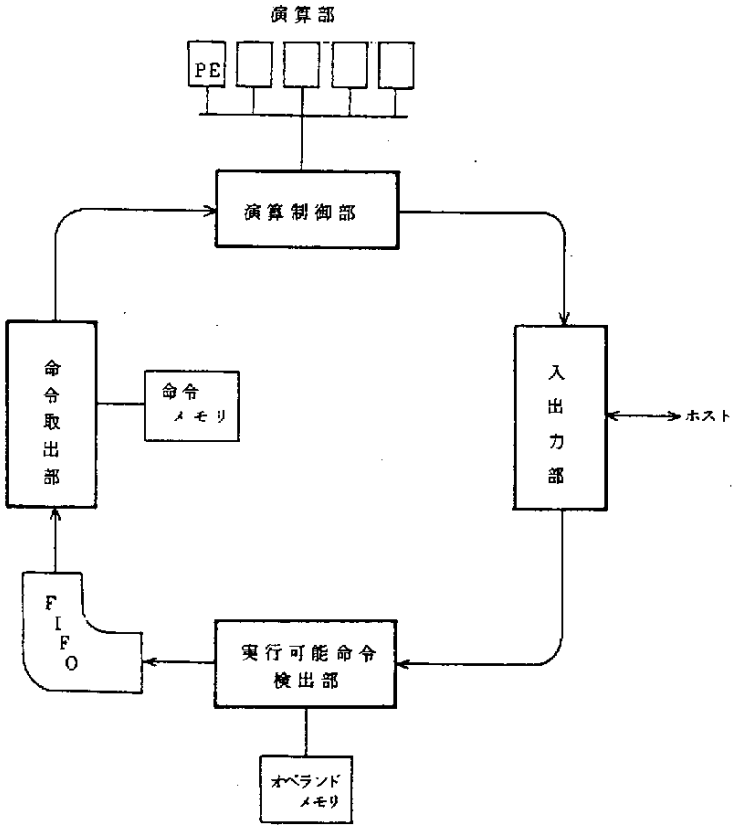


図3.2.20　富士通マシンシステム構成

- 命令メモリとオペランドメモリを別構成してメモリ競合を避けた構成。
- 演算部だけが複数個用意されており，コモンバスで接続されている。
- 高級言語を設定し，コンパイラをサポートしている〔ＩＴＡＳＨＩＫＩ８２〕。

D. 問 題 点

小規模な試作マシンであり，機能的に小さいのであまり言及出来ないが，

- 機能的に低い

  構造データ，手続，ループ等に関する機能が無い。

- 演算ユニットへのバスがネック

  演算ユニットは複数用意されているが，コモンバスによって接続されているため，バスがネックとなる。

- 条件分岐の扱い方

  条件文に対して，真偽の結果より前に式の評価を先行しており，見かけ上の並列度は上るがトータルとして見ると実行時間に不利となっている。

E. 現　　状 ： 試作機を１９８１年に作成。

演算部に汎用演算チップ４個と特殊演算ハード（ゲート，コピー等）を使い，残りの部分はハードワイヤドロジック。座標変換用の数値計算プログラムを実行させ，多重度２前後で動作する。

―参考文献―

〔ＭＡＳＵＺＡＷＡ８１〕増沢，板敷，相馬，"データ駆動型プロセッサー基礎実験機－"
昭和５６年度　信学会全国大会，№1445

〔ＩＴＡＳＨＩＫＩ８２〕板敷，佐藤，増沢，相馬，"データフローマシン用高級言語コンパイラの試作"昭和５７年　情報処理学会第２４回全国大会，№７Ｄ－３

3.2.14　電総研のデータフローマシン

A. 場　　所 ： 電子技術総合研究所

B. 人　　 弓場，島田，山口

C. 特　　徴

数値演算用とＬＩＳＰ用の２つのデータフローマシンの研究が行われている。数値演算用はArvindマシンに関してスケジューラの構成法やスケジューリングアルゴリズムについて研究し，その改良を目指している。

ＬＩＳＰ用データフローマシンＥＭ－３はLazy評価機構を並列化することにより，ＬＩＳＰ言語の実行においてより多くの並列性を引き出そうとしている。このような試みは，ＣＯＮＳについてはユタ大学や武蔵野通研で行われているが，電総研では一般的な関数評価の機構にまでこの概念を拡張している。

この関数評価機構の特徴は，関数の呼び出し機構に特別の処理形態をもうけることによってデータ駆動制御における高度な並列性の実現とLazy関数評価機構を自然な形で融合している点に

ある。関数の呼び出しが行われた場合，ここではデータ駆動制御にもとづき，この関数がある特別のオペレーションを実行したものとみなす。そしてこの関数呼び出しによって得られる関数の値を擬結果（pseudo result）と名付ける。（図3.2.21参照）。

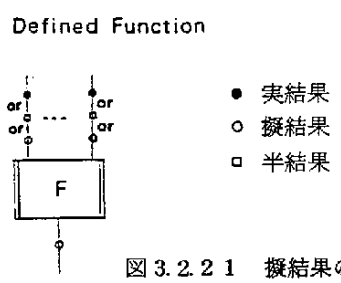


図3.2.21　擬結果の生成

関数本体の実行は擬結果の出力と同時に行われ，擬結果はその関数本体の処理が終了した時点で実結果（actual result）に置き換えられる。

擬結果はデータ駆動制御において，通常の値を持った実結果と同様に演算や関数を発火させる働きを持つ。このためデータ駆動方式を用いた関数の評価機構に対して先廻り制御やパイプライン動作が導入され，その並列性を増大する効果をもたらす。特にリスト構造を生成するオペレーションの値として半結果（semi result）というデータ型を導入し，これを本制御機構上で動作させることによりLenient Consが自然に導入される。実結果が必要とされるオペレーション（たとえば数値演算）に擬結果が渡された場合には，実結果が得られるまでそのオペレーションの実行は待たされる。

複数台のプロセッサエレメント（ＰＥ）で上述の評価機構を実行するためのＰＥの機能モデルが提案されている。これを図3.2.22に示す。上述の擬結果を管理するために各ＰＥ内には結果記憶（result store）と呼ばれる擬結果の管理領域が必要とされ，ここにおかれた擬結果表や遅延バッファによって，各ＰＥが独立して関数評価の制御を行うようになっている。

D. 現　状

高級言語としてデータフロー用ＬＩＳＰ言語（ＥＭ－ＬＩＳＰ）とデータフローグラフレベルの中間言語（ＥＭＩＬ）の仕様を定め，そのコンパイラが作成されている。上記のメカニズムの妥当性とデータ収集のためのソフトウエアシミュレータを作成中である。またＬＩＳＰで書かれたプログラムの動的実行形態を並列処理の立場から分析するための統計データを収集中である。

E. 問題点

構造データが分散配置されるため，関数評価におけるスケジューリングアルゴリズムをネットワークのハードウエア構成に従って適切に選択する必要がある。また擬結果のガーベジコレクシ

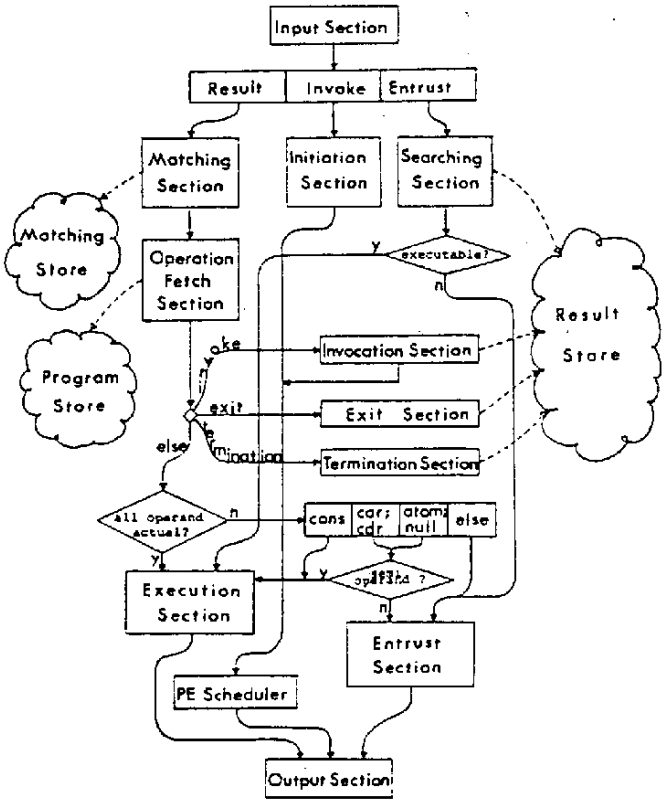


図3.2.22　PEの機能ブロック図

ョンなどが必要とされ，オーバーヘッドが増大する危険もある。

―参考文献―

1）山口，弓場，島田：「データ駆動計算機ＥＭ－３の関数評価機構」，信学技報，ＥＣ８１－５７（昭５６.１２.）

2）島田：「タグ付トークンを使用するデータ駆動計算機シミュレータ」，信学技報，EC81－58（昭５６.１２.）

3）島田：「タグ付トークンを使用するデータ駆動計算機のシミュレーション」，情報処理学会第２４回全国大会（昭５７.３）

4）山口，弓場：「データ駆動計算機ＥＭ－３のソフトウエアシミュレータ」，同上

5）弓場，山口，島田：「データ駆動計算機用Ｌｉｓｐとその中間言語」同上

6）山口，弓場，山田：「データ駆動モデルからみたＬｉｓｐプログラムの動特性」同上

3.2.15　通研のデータフローマシン

電電公社武蔵野通研で開発が進められているデータフローマシンである。超高速科学技術計算を指向した数値処理用（ＤＦＭ－Ｕ）と知能処理などの非数値計算を指向した記号処理用（ＤＦＭ－Ｌ）の２種類がある。

ＤＦＭ－Ｎは，自律的に動作する多数のデータフロープロセッサ（ＰＥと呼ぶ）をアレイ状に結合したシステムである。[1) 2)]（図3.2.23）

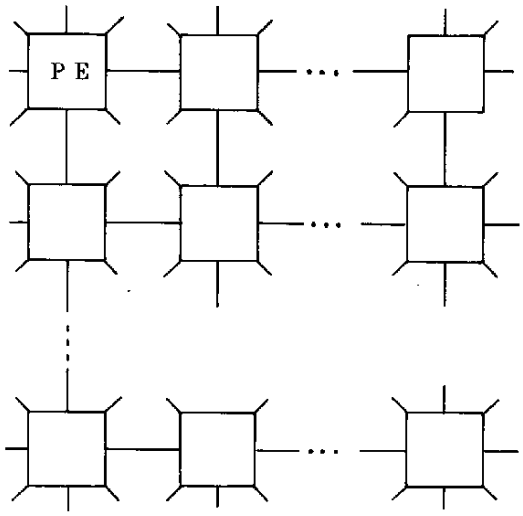


図3.2.23　ＤＦＭ－Ｎの構成

- ○　プロセッサ内およびプロセッサ間の制御を全てデータフロー概念で統一することにより，自律的で並列度の高いシステムを実現すること。
- ○　偏微分方程式の解析等の格子問題では近傍計算が主なので，各格子点をアレイ要素のＰＥに割り当てることにより，通信コストおよび遅延時間の問題を解決することを主な狙いとする。

色つきトークン方式を採用しており，静的（コンパイル時）および動的（実行時）名前づけを併用したアレイデータ処理に特徴がある。アレイデータの各要素は各ＰＥに写像され分散処理される。要素とＰＥとの対応関係は，アレイ型データ対応に写像方式を定めるディスクリプタ（Ｄ）と，要素対応にＰＥの割り当てを定めるインデクス（Ｉ）を用いて，予め静的に与えられている。処理中Ｉの変更があると，インデクス操作のマクロ命令が実行され，結果のプロセッサ間転送が引き起こされる。ループで実行されるデータはループカウントフィールド長に制限があるので，オーバフローした時点で同期をとる命令が設けられている。

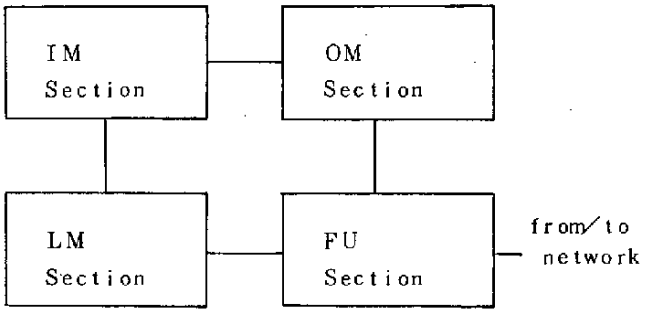


図3.2.2.4　ＰＥの構成

ＰＥは図3.2.2.4のように，命令メモリ（ＩＭ），オペランドメモリ（ＯＭ），演算ユニット（ＦＵ），リンクメモリ（ＬＭ）からなる循環パイプライン構成である。ＩＭにはプログラム（本体）が格納され，ＯＭは到着オペランドを保持するバッファとして使用される。ＬＭには結果の分配先命令名が格納されている。ＬＭではＦＵの実行結果パケットに付された結果名で分配先命令名を求め，それを基にオペランドパケットを作る。オペランドパケットがＩＭに到着すると対応する命令が読み出され，オペランドパケットと共にＯＭへ送られる。命令が2オペランド形式であれば，オペランドパケット中の＜命令名，instance名＞をKeyとしてＯＭを検索する。ＯＭのKey検索には，8重並列ハッシュ方式を採用している。一致すれば該（対となる）データを読み出し，命令パケットを構成してＦＵへ送る。一致しなければ，オペランドパケットをＯＭに保持しておく。命令が1オペランド形式ならば直ちに命令パケットと構成しＦＵへ送る。命令パケットには実行後の結果名が含まれている。ＦＵは複数の演算器からなり，浮動小数点演算，通信処理，Ｉ／Ｏ処理ユニット等が含まれる。

ＤＦＭ－Ｌは，実行制御部（ＣＭ），ＣＭ間をつなぐ通信ネットワーク，複数のバンクから構成される構造体メモリ（ＳＭ）およびＣＭ－ＳＭ間を結合する調整ネットワーク（Ａ－uet），分配ネットワーク（Ｄ－net）からなる。[3]（図3.2.2.5）.

ＤＦＭ－ＮとＤＦＭ－Ｌの実行制御機構は共通である。ＣＭはＰＥとほぼ同様の構成をとり，浮動小数点演算ユニットの代わりにＡ－net およびＤ－netへのインタフェース回路が置かれる。リスト処理の演算ユニットはＳＭに組込まれている。ＳＭの各バンクはさらに独立動作可能なcar，cdr，attribute，reference count のフィールドに分解され，それぞれに専用の操作ユニットが設けられる。ＣＭから送出された構造体操作命令パケットはパケット中のオペランドアドレスのデコードにより，対応するＳＭバンクへ転送される。Ｄ－netはＳＭから送出された結果パケットを受け取ると，結果パケット中のＣＭ番号を基に要求元ＣＭへその結果パケットを送る。

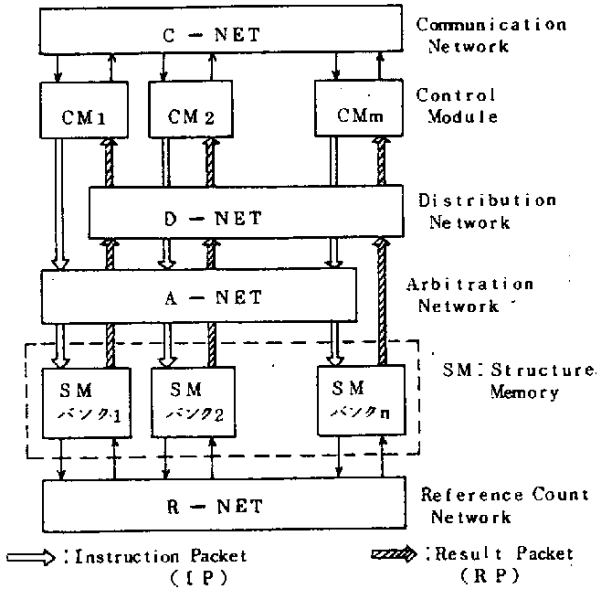


図3.2.25　DFM－Lの構成

car,cdrフィールドにはready（値が未到着）を示すタグがあり，これによりLenient cons，Lazy evaluationの機能が実現されている。[4)5)]

現在，数値処理用および非数値処理用データフローマシンの共通の基盤となる循環パイプライン型ＰＥ１台と，ＳＭ２バンクからなる実験機の設計を行っている。この実験機はＰＥ１台であるがデータフローマシンとして動き，データ駆動原理の確認の他，データフロー用高級言語Validの開発ツールとしても使用される。また，複数ＰＥでの性能を調べるため，先に開発された４×４ＰＥ（ソフトウェアでデータ駆動制御を模擬）のデータフロープロセッサアレイ実験機ＥＤＤＹを使って評価が進められている。

―参考文献―

1) 高橋，雨宮，“超高速科学技術計算を指向したデータフロープロセッサアレイ計算機の提案，”信学技報　ＥＣ８０－２４，１９８０．

2) 高橋，吉田，成瀬，雨宮，“超高速科学技術計算向きデータフロープロセッサアレイ計算機の構成と制御，”昭５７電子通信科学会全国大会

3) 雨宮，長谷川，三上，“リスト処理向きデータフローアシンアーキテクチャとそのソフトウェアシミュレータ，”　信学技報　ＥＣ８０－６９，Ｆeb.１８，１９８１．

4) 中村，長谷川，雨宮，“リスト処理向きデータフローマシン用構造体メモリの設計と評価，”　信学技報　ＥＣ８１－３２，Ｏct.２９．１９８１．

5）長谷川，雨宮，"データフローマシン上でのLazy evaluationの実現について，"
情処学会第２４回（昭５７前期）全国大会，５Ｄ－８．

3.3 リダクションマシン研究開発の現状

本章では，式の還元（reduction）という形で計算が行われるリダクション型マシンの研究開発の現状について調査を行う。リダクション型マシンでは，計算はすべて書換え規則の適用とみなされる。データフロー型マシンでの計算の考え方は，値に対してオペレーションを実行することである。即ちデータフロー型マシンではグラフのノードにデータトークンが到着すると実行可能となり，実行結果がアーク上に送出される。この過程を置換という観点からみると，ストリングまたはグラフの書き換え規則の適用としてとらえることもできる。リダクションとデータフローとの関連については，今後検討していく必要がある。

リダクション型マシンの実現の仕方には，文字列そのものを置換するstring reduction と，ポインタを利用してグラフ（リスト）を操作するgraph reductionの2通りの方法がある。string reductionでは，還元の過程でstringが伸縮する際にコピーやメモリ再配置の問題が生じてくる。graph reductionではグラフの共有が行われるのでコピーオーバヘッドの問題はないが，共有グラフ操作のための代価は支払わねばならない。

リダクション型マシンの提案は比較的少ない。国外では，Berklingが提案したラムダ計算マシン（スタックマシン）と，Backusの提唱した関数型言語FPを実行する木構造のMagó マシン，シフトレジスタを使ったTreleavenのマシンの例と，国内では日電小長谷によるCombinatorを使ったデータ駆動型リダクションマシンの例がある。以下，各マシンの特徴について述べる。

3.3.1 Berkling マシン

A．場　　所　Gesellschaft fur Mathematik und Daten　（西独）

B．人　　Berkling

C．特　　徴

ラムダ計算にもとづき，文字列置き換えにより計算を行う。マシンには関数型言語を2分木で表現したものが与えられる。計算は図3.3.1でapplyの右下の引数をλの左下の変数の値とし，λの右下の式の中の変数に代入する形で進められる。この手順を繰り返し，式の中のすべての変数が定数に置き換わった時，計算が終了する。例として図3.3.2にf（x，y）＝（6＋x）×（x－y）の式でf（5，2）を計算したものを示す。

Berklingマシンではこの2分木をたどり置き換える過程をスタックを用いて実現している。アーキテクチャの概念図を図3.3.3に示す。入力はsourceスタックに置かれる。プロセッサはsourceスタックの上部から要素を取り出す。もしその要素が2分木のterminal ノードで

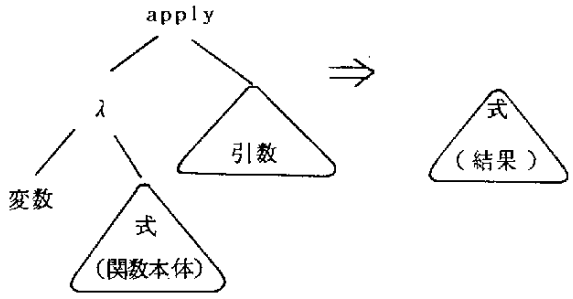


図3.3.1　　2分木表現とラムダ計算

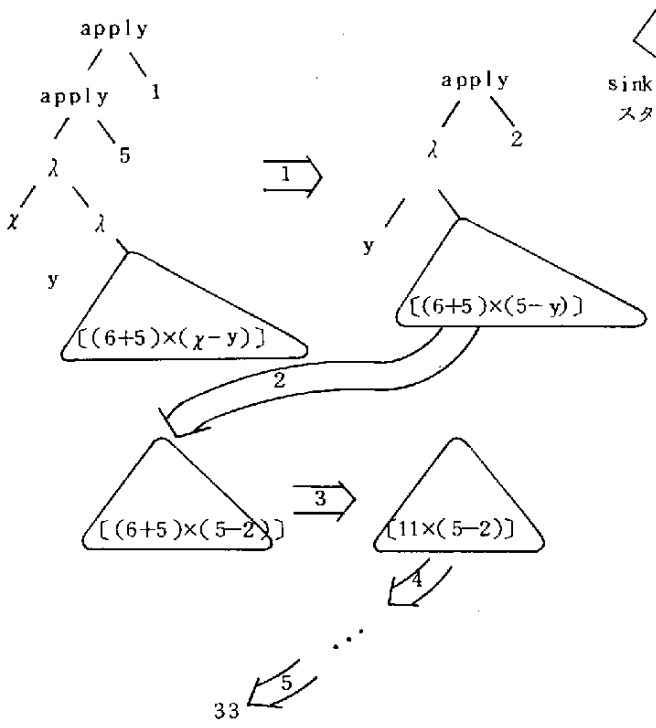


図3.3.2　計　算　の　例

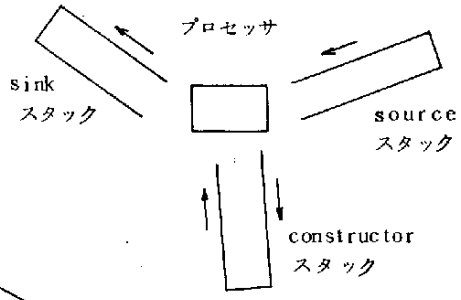


図3.3.3

Berklingマシンの概念図

あればconstructorスタックにpushする。プロセッサはこうして2分木をたどる間いつも3つのスタックを監視し，もし置き換えが可能であれば置き換えを行う。1回も置き換えを行わずに木を始めから終りまでたどった時，計算が終了する。

←AB（AをBに適用せよ）の計算例を図3.3.4に示す。まず←ABをsourceスタックに置き，終了符号（@）をconstructorスタックに置く(a)，次にsourceスタックの先頭の←を

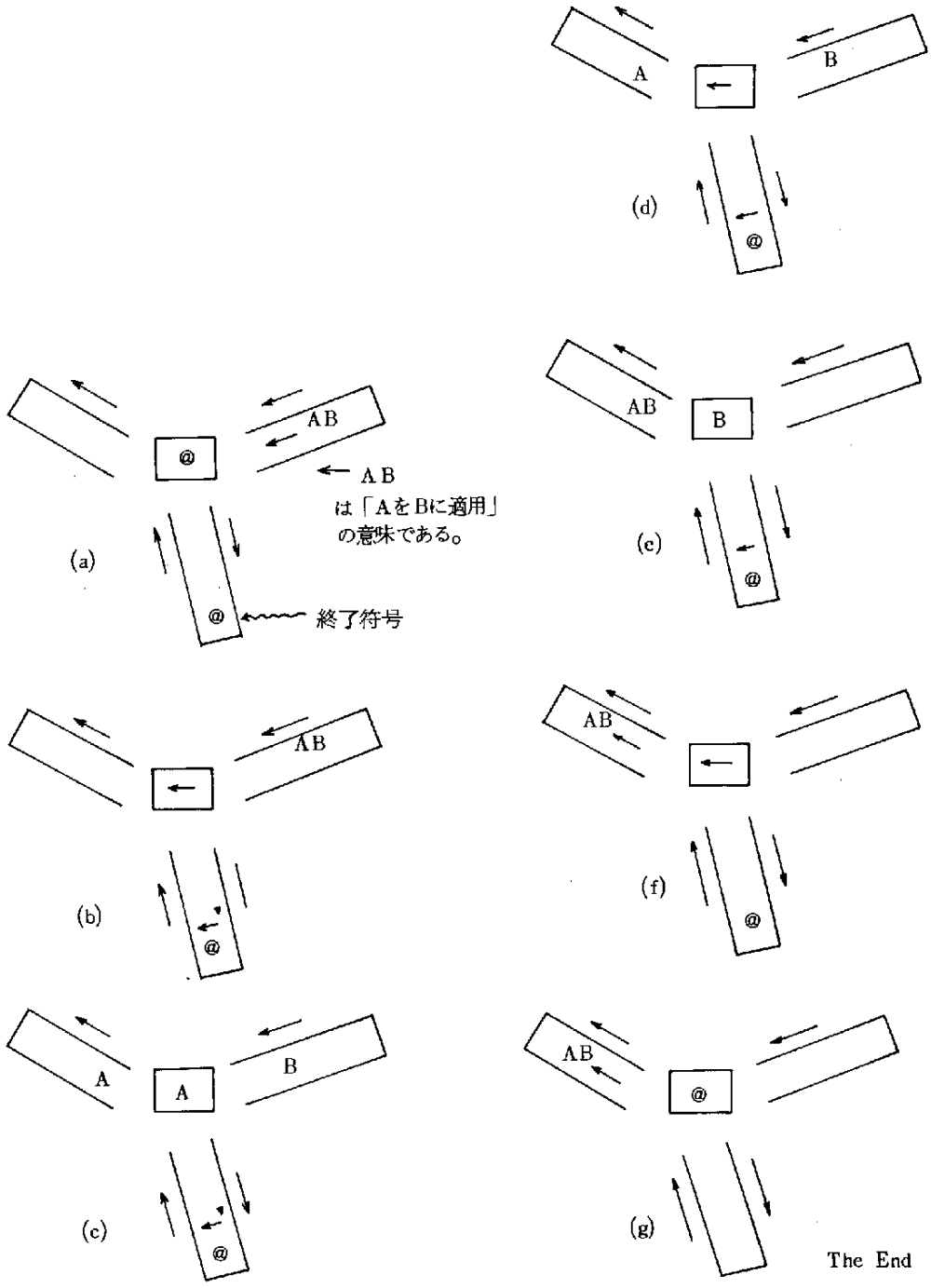


図3.3.4　　Berklingマシンの動作例

プロセッサが取り出す。←は作用素なのでconstructorスタックにpushし▼印を付けておく(b)。次にsourceスタックからAを取り出す。これはterminalノードなのでsinkスタックへpushする(c)。sinkスタックに1回pushするたびにconstructorスタックから1個取り出し，置き換えが可能かチエックするがこれは←であり▼印が付いているので▼印を消去し，←をconstructorスタックに戻す(d)。次にsourceスタックからBを取り出すとこれはterminalノードなのでsinkスタックにpushし(e)，constructorから1個取り出す。これは←なのでsinkのA，Bを使って←ABを実行する(f)。再びconstructorから1個取り出すと終了記号（@）なので計算を終了する(g)。

すべてスタックだけで操作でき，非常にシンプルなのが特徴である。

D．問　題　点

文字列置き換えの方法が効率の点で問題である。並列計算へどう拡張していくかが不明である。

E．現　　状

TTLを使って1号機が作成されている。各スタックの容量は8キロバイトである。

—参考文献—

1） Berkling，K.J.：Reduction Language for Reduction Machines，Proc 2nd Symp. on Computer Architecture，P．P．133－138，1975．

2） Organic，E.I.：コンピュータシステムアーキテクチャの技術動向，電子協 昭和53年度特別セミナー講演録54－C－371。

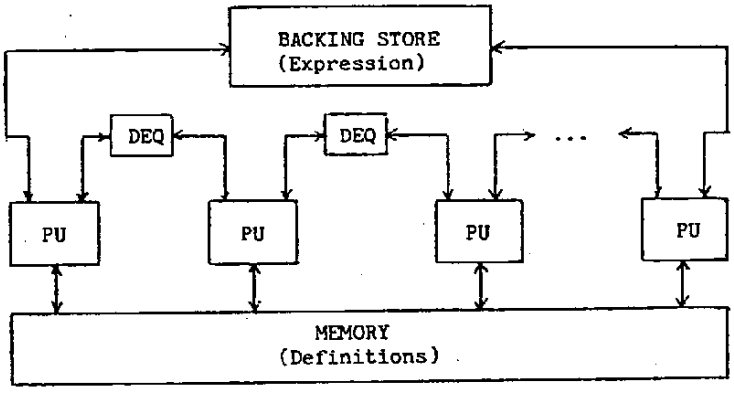
3.3.2　Treleavenのリダクションマシン

Newcastle大のP.C. Treleavenらは図3.3.5に示すリング構造リダクションマシンを1980年に提案している。

A．概　　要

マシンの全体構成は図3.3.5に示すように複数台の処理ユニット（PU），DEQ，Backingメモリ，定義メモリからなる。

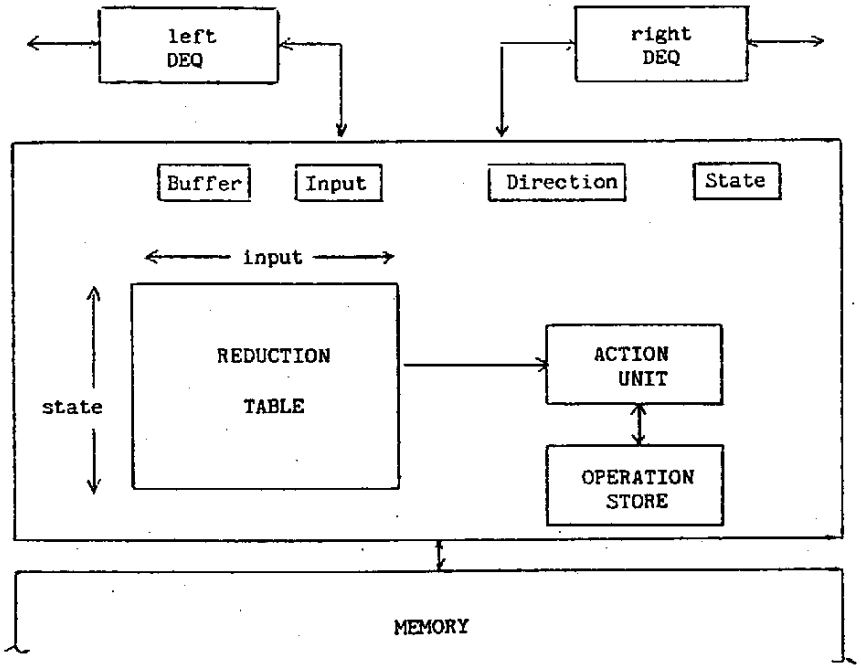
定義メモリにはユーザの定義式を入れ，BackingメモリとDEQには計算を行う式を入れる。PUは図3.3.6に示すように，バッファレジスタ，状態レジスタ，リダクション表，演算ユニット等から成る。リダクション表はPUの状態遷移を決めるもので，PUの現在の状態とDEQから読み込んだ式に対して次に何を実行するかが記述されている。



DEQ = double ended queue

PU = processing unit

図 3.3.5 Multi-processor Reduction Machine Architecture.



Processing Unit.

図 3.3.6

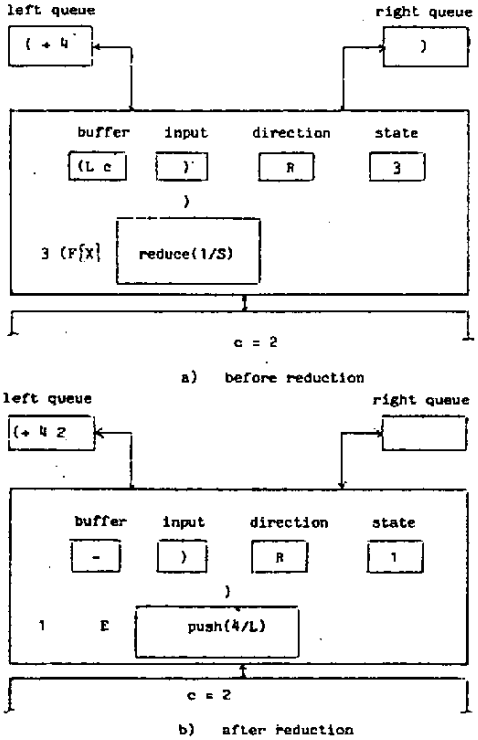
次に簡単な例に従って実行の様子を示す。図3.3.7はリダクション言語で書かれた簡単なプログラムである。ここでLはLoad, Sは Store のオペレータである。このプログラムは図3.3.8のように実行される。マシン上では図3.3.8の式が図3.3.9のように各PUとDEQ上に置かれリダクション表（図3.3.10）に従って計算が実行されていく。

```
t1 = (+ (L b) (L c))
t2 = (- (L b) (L c))
b  =      4
c  =      2
a  = (* (L t1) (L t2))
```

図3.3.7

stage	state of evaluation
1	(S a (L a))
2	(S a (* (L t1) (L t2))
3	(S a (* (+ (L b) (L c)) (- (L b) (L c))))
4	(S a (* (+ 4 2) (- 4 2)))
5	(S a (* 6 2))
6	(S a 12)
7	-

図3.3.8



Operation of a Processing Unit.

図3.3.9

B. 問 題 点

本マシンでは，各PU間のデッドロック等が起こらないようなリダクション表をつくることがポイントとなる。Treleauen らはcompiler-compiler に類するリダクション表ジェネレータを提案しているが，現在検討中の段階である。

—参考文献—

P.C. Terleaven, G.F.Mole : A Multi-Processor Reduction Machine for User-defined Reduction Languages, Proc.the 1980 Symposium on Computer Architecture, May 1980.

Direction	State	Input: empty E	operand X	operator F	brackets (	brackets)
	1 E	wait(1/S)	pass(1/S)	pass(1/S)	push(2/R)	push(4/L)
right	2 (	wait(2/S)	error(1/C)	push(3/S)	pop(2/S)	error(1/C)
right	3 (F{X}	wait(3/S)	push(3/S)	error(1/C)	pop(2/S)	reduce(1/S)
left	4 {X})	give(1/C)	push(4/S)	push(5/S)	error(1/C)	pop(4/S)
left	5 F{X})	give(1/C)	error(1/C)	error(1/C)	reduce(1/S)	error(1/C)

Format of table entries: action (new state new direction)

1 - 5

- S same direction
- C change direction
- R right
- L left

図3.3.10 Example Reduction Table

3.3.3 Magó

North Carolina大学のGyula A. Magó によって提案されたリダクション型マシンであり，図3.3.11に示すような2進木状の構成をとっている。このマシンの特徴は次の通りである。

(1) VLSI化を念頭に，数種のチップを多数規則正しく並べ木構造セルを形成する。(cellular

tree machine) (2) Backus のFPクラスの言語を直接実行する。(3) FPプログラムに内在する並列性はそのまま引き出される。 (4) 機械語そのものがFP言語である。

このマシンではFPの内側計算規則を反映し，還元可能な最内側の部分式(reducible appli-cation)を並列に評価して，結果の式に置換するためにstring reduction 方式を採用している。プログラムはFPの記法にそって書かれ，application と sequence のネスト構造である式で構成される。長さnのsequence は(a_1 , a_2 , …, an) で，application は＜operator, operand ＞で表現される。例えばベクトルの内積をとるプログラムは＜(+,(AA,＊), TR), ((1, 2, 3, 4), (11, 12, 13, 14))＞と書かれることになる。但し，各シンボルはAA(apply to all), TR(transpose), +(addition), ＊(multi-plication) を意味する。

FPの機械語プログラムは，ベクトルの形をしたLeaf セルに写像される。ここでは各式はシンボルの線形ストリングで表現され，1つのLセルに1シンボルが置かれる。式を分離するシンボルはLセル内の機械語表現では省略される。applicationとsequenceを囲むシンボルのうち＞及び）が除去され，その代わりネストの深さを表わす数字がLセルに書込まれる。

マシンは図3.3.11で示したように，Lセル及びTセル(non-leaf cell)の2種類のセルで構成されている。Lセルはメモリセルとして機能する。reduction の結果ストリングが伸縮するので，隣接するセルへのシンボルの移動などストレージアロケーションを行うことができる。Tセルは，プログラム，中間結果，制御情報などの転送の他，オペレーションの実行機能を持っている。Tセルの内部は，配下の部分木および親ノードへのrouting および中間結果の保持のため，6個のレジスタと高々4個のPEで構成される。reducible application の実行のため，必要なTセルの部分木が割り当てられるが，最短経路で情報転送が可能となるような割り付けが行われる。ルートセルは，外界との通信を行うI/Oポートを有し，プログラムはここからTセルを経由してLセルへと送られる。各セルは同期的に動作する。動作モードは3種あり，Tセル内に保持された中間結果を別のTセルへ転送するモード，Lセル内で生じた状態変化を上位のセルへ伝達し，次に取るべき動作を指示する制御情報を下位のセルへと伝達するモード，およびLセル内のストレージアロケーションを行うモードからなる。このように，上位→下位への制御情報の転送および下位から上位への状態推移情報の通知で1サイクルが終了する。 innermost application の reduction を行うため，Lセルでとられる動作はマイクロプログラムで制御される。マイクロプログラムは必要に応じて外界からLセルへロードされてくる。例えばLセルの1つがある部分式に含まれる他のLセルへ情報の広報転送を行う際はSENDマイクロ命令を実行する。これにより情報はその部分式のルートに送られそこから各Lセルへの転送が行われる。

Magó マシンでは，各reducible applicationの計算が並列にかつ局所的に行われ，glo-bal な同期制御をとりながら個々のセルはみかけ上非同期的に動作することになる。しかし文字列の置き換えのための制御はかなり複雑であり，還元が行われる時の再配置のオーバヘッドや，最適配置の工夫が必要なことなど，実現上の問題が多々残されている。

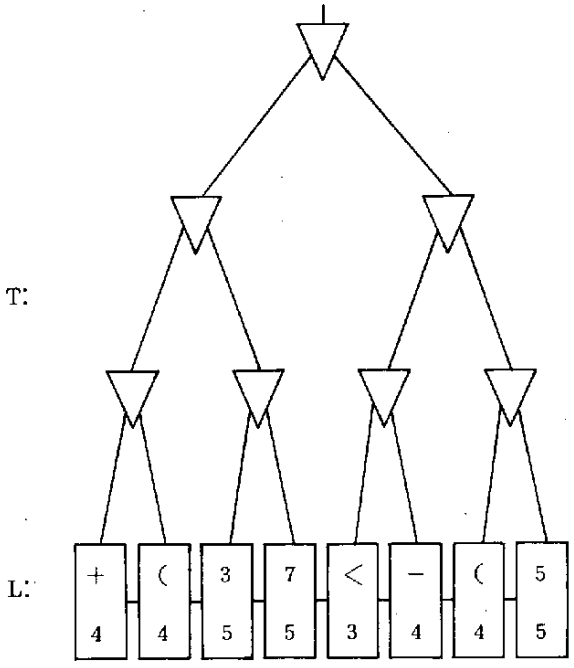


図 3.3.11　　Magó の木構造マシン

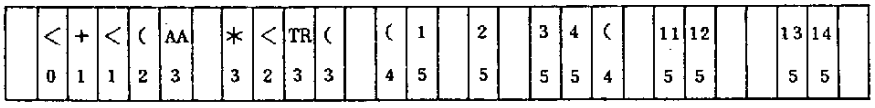


図 3.3.12　　機械語プログラムの内部表現

3.3.4　日本電気のリダクション

本マシンはNECの小長谷，山本により提案されたラムダ計算系のリダクションマシンである。[1)]

計算はリダクショングラフと呼ばれるラムダ式と等価なグラフを並列リダクションすることにより遂行される。ただし，データフローマシンが副作用を排除して並列性を極限的に追求するのに対し，本マシンでは並列性とともに，

(1)　メモリアクセスネックの解消，

(2) 実行制御メカニズムの強化,

に重点が置かれている。主な特徴は次の通りである。

① リデュースオペレータによる変数結合

仮引数と実引数との変数結合の実現方式として，結合子論理にもとづくリデュースオペレータ"S"，"K"，"I"等が使用されている。例えば図3.3.13のリダクショングラフにおいて，apノードが起動されると実引数3がリデュースオペレータの働きにより上位ノードから下位ノードに向けてブロードキャストされる。このとき，節ノードにおいて"S"は実引数をコピーするために使用され，端ノードにおいて"K"は実引数を消滅するために，"I"は仮引数を実引数と置き換えるために使用される。

② データ駆動による並列実行

図3.3.14に示すように，リデュースオペレータが全て使用され変数結合が完了すると各ノードは活性化され，データ駆動の原理により下位ノードから上位ノードに向けて命令が起動される。データフローマシンの起動メカニズムとの違いはノードにリデュースオペレータが付加されているとオペランドが揃っても命令が起動されない点にある。これにより，式の実行順序の指定や遅延評価が可能となり，条件式や汎関数の実現に利用されている。

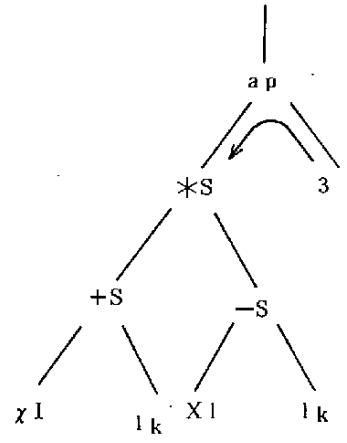


図3.3.13 変数結合

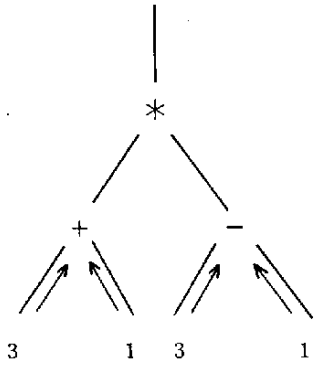


図3.3.14 命令実行

③ VLSIアーキテクチャ

本マシンの基本要素となる処理要素PEの概念図を図3.3.15に示す。PEは実行コントローラEXCと演算機能を持つ多数のセルユニットから構成される。セルユニットにはオリジナル・セルユニットOCUと実行セルユニットECUの2種があり，OCUにはリダクショングラフを分割したサブグラフの1つが格納される。

関数適用毎に対応するサブグラフがＯＣＵからＥＣＵにコピーされた各ＥＣＵに実引数が同時に転送される。変数結合は各ＥＣＵで独立に実行され，変数結合が完了するとサブグラフの命令が起動されて結果が上位のＥＣＵに転送される。

このようなマシンでは命令実行が局所的に行われるためメモリアクセスネックの少ないシステムが構築可能である。また，規則的な構造を成すためＶＬＳＩ向きのアーキテクチャといえよう。

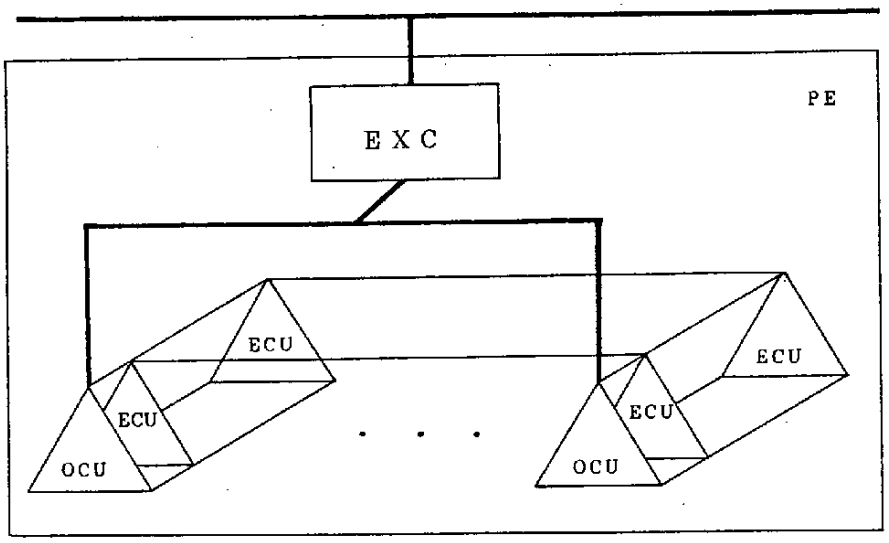


図3.3.15　処　理　要　素

－参考文献－

1）小長谷，山本，「リダクションマシンの構想について」，電子通信学会技報ＥＣ－８１－３３，１９８１年，pp．２１－３０

3.3.5　ＡＬＩＣＥ

ＡＬＩＣＥ（Applicative Language Idealized Computing Engine）はインペリアルカレッジ(英)のJ．Darlingtonらによって開発が進められているマルチプロセッサリダクションマシンであり，１９８１年にその概要が発表された。

A．概　　要

ＡＬＩＣＥは適用型言語の並列リダクションを実行するマシンであり，次にその実行の様子を

示す。図3.3.16にFactorial(n)を求める例題を示す。

```
Factorial : integer → integer
    Factorial(n)⇐ FactB (0, n)
FactB : integer × integer → integer
    FactB (i, i) ⇐ i
    FactB (i, i+1) ⇐ i+1
    FactB (i, j) ⇐ FactB (i,mid) * FactB (mid.j)
        Where mid = Integerdevide (i+j, 2)
```

図3.3.16

図3.3.16のプログラムの実行(図3.3.17)を図3.3.18のパケットの集合として表わす。(図3.3.19)

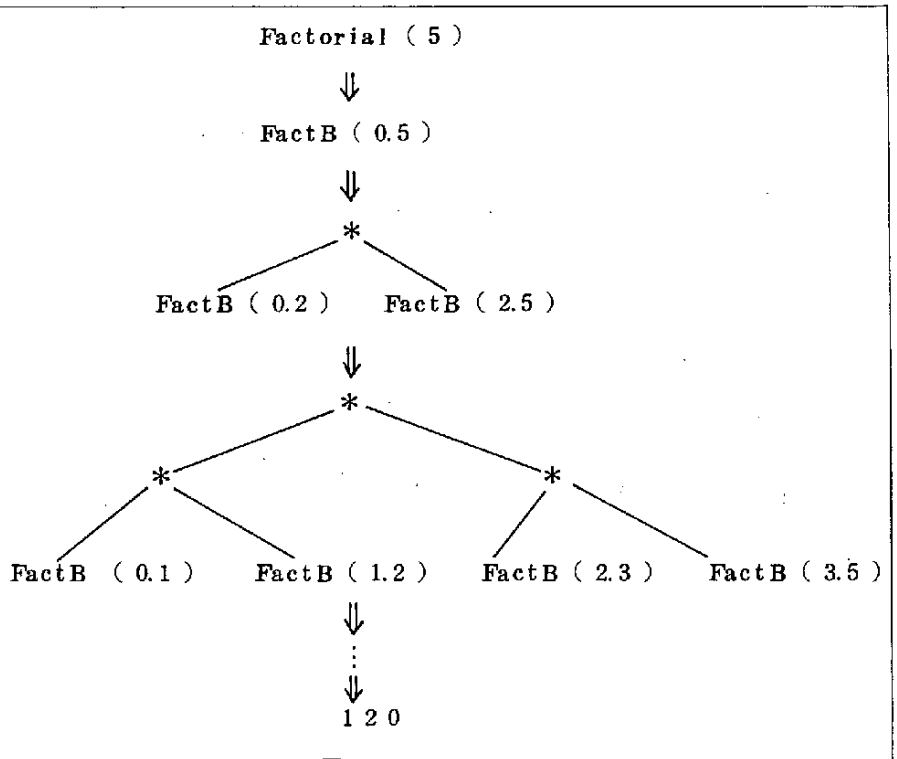


図3.3.17

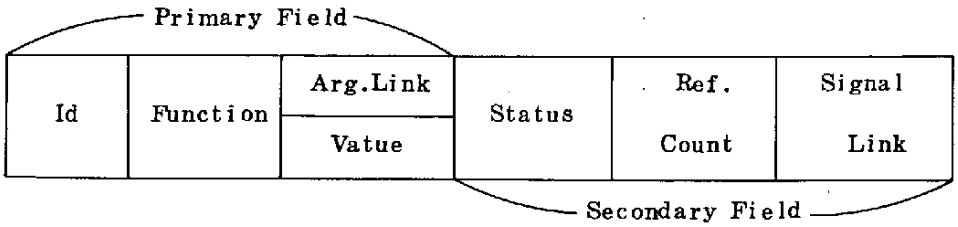


図3.3.18

パケットの前半分(Primary Field)は実行グラフのノード表現に用いられ，後半分(Secondary Field)は引数の待合せ，Lazy Eval 等の実行制御に用いる。

マシンアーキテクチャの概念図を図3.3.20に示す。マシンはパケットプールと多数の処理要素(Agent)からなる。Agent はパケットプールの中から実行可能となっているパケットを

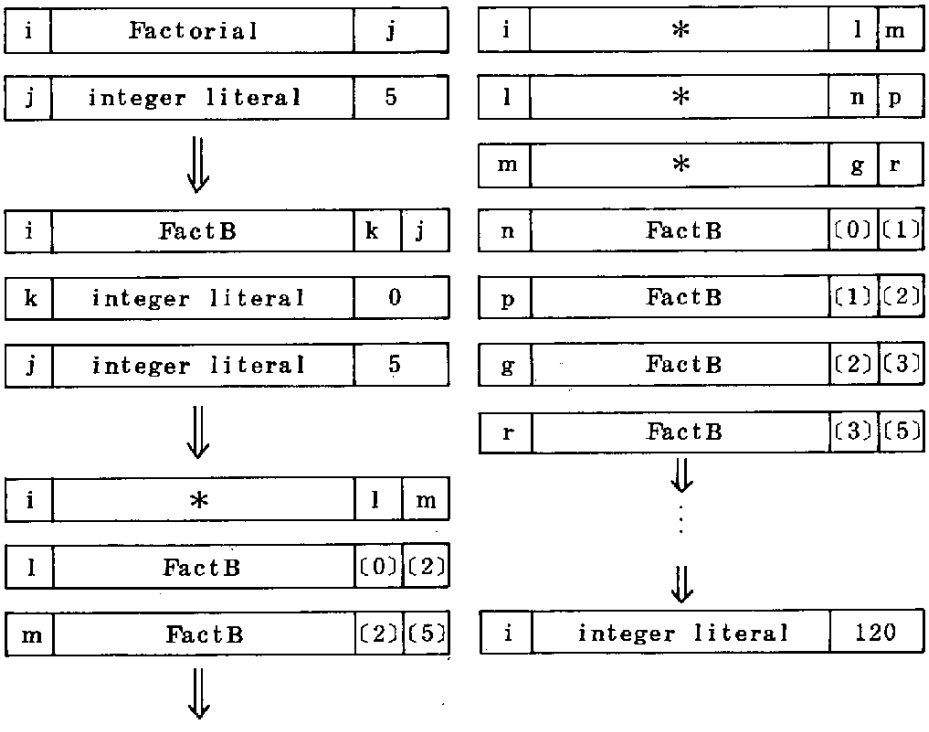


図3.3.19

取り出し，書き換え規則に従って新たなパケットに置き換える。

サポートする言語としては，Turner の Combinators，BackusのFPのようなVariable free Language や，高階の関数型言語，論理プログラミング言語等を考えている。

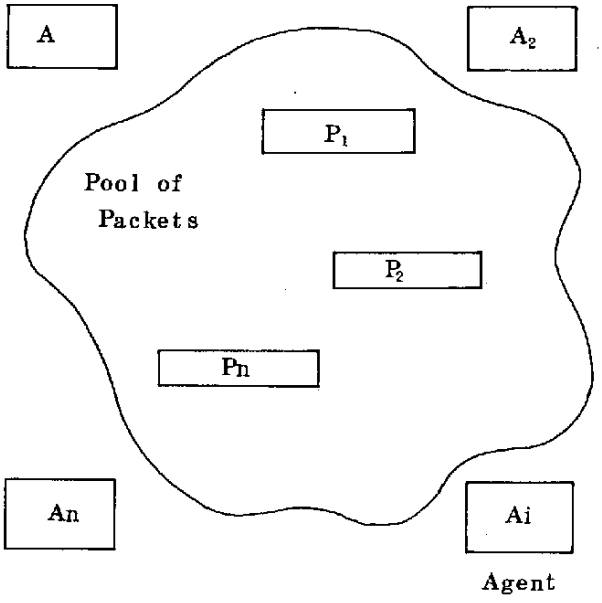


図 3.3.2 0

B．現　　状

ＡＬＩＣＥは現在のところ概念設計を進めている段階でありマシンアーキテクチャは確立していない。しかし，シミュレータ上でＨＯＰＥ，ＬＩＳＰ，ＰＲＯＬＯＧ 等を動かすことによって計算機構は明確となってきている。近いうちに，マシンアーキテクチャの詳細を盛り込んだシミュレータが開発される予定である。

—参考文献—

J. Darlington, M. Reeve ： ALICE Multi-Processor Reduction Machine For the Parallel Evaluation of Applicative Languages, Proc, of the 1981 Conf. on Functional Programming Language and Machine Architecture.

4. ＤＦＭ実現技術の検討

4.1 ＤＦＭの検討項目

データフローの概念は１９６０年代の後半より並列プログラムの表現手段として研究されてきたが，言語としての骨組みが明確にされたのは，ＭＩＴのＪ.Ｂ.Ｄｅｎｎｉｓによるデータフロー言語以来である。従ってマシンの実現はそれより遅れ１９７０年代の中頃になる。その実現例も余り多くなく，かなりの多重度を持ったシステムの検討はＬＳＩ技術が発展し始めたここ数年であり，データフローマシン実現に対する諸技術は，まだ殆んど確立されていないといえよう。この章ではＤＦＭを実現するために必要な種々の技術について，現在までに提案されている諸方式を概観して評価を加えるとともに，未検討ではあるが重要と思われる項目についてはそのポイントと解決策の提案または研究の方向の示唆を行う。

識別した検討項目は次の９項目である。

(1) ＤＦＭ用高級言語とその評価

(2) ＤＦＭの制御方式と機能要素

(3) 構造メモリ

(4) 結合ネットワークとアクティビティ割り付け

(5) 入出力方式

(6) 実装法

(7) 制御ソフトウェア

(8) 信頼性処理

(9) デバッグ機能

(1)は言語の検討で，幾つかの提案がためされているがその評価より実際に用いるべき言語の設計に関係する。(2)はアクティビティの発火検出やトークンの識別子等データフローマシン心臓部の構成法および全体構造についての項目である。(3)はデータフローマシンの１つのキーポイントである構造メモリの方式検討，(4)はマルチプロセッサ構成でネックとなり易い結合網の構成検討で，特にデータフローマシンでは重要な１つの要素である。また，アクティビティ割り付けはその多くのプロセッサ資源を有効に利用するための方式検討でスケジューラやコンパイラ，ローダ等の設計に関連する。(5)～(9)については，未だ殆んど研究が行われていない項目であるが，重要と思われるものである。(5)はデータフロー言語が関数型であることによりその扱いについては工夫を要する。(6)は今後不可欠となるＶＬＳＩ化への検討，(7)は従来の計算機で大きな位置を占めてきたオペレーティン

グシステム等の構成に関するもので，全く新しい考えにもとづくデータフローマシンに対して基本から考え直す必要がある。(8)は元々今後のシステムで重要な項目ではあるが，従来のマシンに比してゲート数が著るしく増す可能性のあるデータフローマシンに対してはより重要である他，ランダムな併行処理に対する信頼性処理という難かしい面も内在する。(9)は，作成したマシンのデバッグやプログラムのデバッグに関するもので，併行処理故の困難さが予想される。

4.2 データフローマシン用高級言語の問題点

4.2.1 従来型言語の問題点

これまでに,Fortran, PL／1, Algol, Pascal など数多くの言語が従来のノイマン型マシンの上で開発されてきた。これら従来型言語はいずれもノイマン型マシンの本質的構造ープログラムカウンタと記憶セルを反映しており，以下に示すような共通した特徴を持っている。

(1) プログラムカウンタによる逐次制御

(2) go to 文にみられるようなJump 概念

(3) 代入（assignment即ち副作用）による計算

これらの特徴から生じる問題を明らかにするため，次の計算を考察してみよう。

```
r：＝y；    g：＝0；
while    r＞＝x    do
begin    r：＝r－x；  g：＝g＋1    end.
```

これはPascal で書かれたプログラムであるが，x，yが入力されるとy／xの商がgに，余りがrに格納される。このように従来型言語では，プログラムは基本的には代入文を主体とする実行文の系列によって記述される。g，r，x，y等の変数は値を保持する記憶セル名を表わし，:＝は右辺の計算結果を左辺の記憶セルへ格納することを示している。実行文はそれぞれセミコロン（;）で区切られるが，文の並びに意味があり，ソーステキスト上に書かれた順に（上から下へまた左から右へ）実行されることを暗々裡に示している。

(1)の問題はプログラムカウンタの弊害である。本来，順序を規定する必要のないものまで無理やり実行順序が定められるので，並列処理性が損われる。例えば上記のプログラムの場合では，r：＝r－xとg：＝g＋1の計算は本来どの順序で（したがって並列に）実行してもかまわないが，書かれた順序に実行され並列実行の可能性を失っている。

また，(2)に掲げたように，go to 文によって任意に制御移行がなされることも，プログラムの構造を不明瞭にし，正しいプログラムの保証を困難にする要因になっている。Dijkstra によっ

て「構造的プログラミング」が提唱されて以来，go to 文を含めたプログラミングスタイルの反省が行われるようになってきたが，構造的プログラミング自身は従来型言語にある種の秩序をもたらす程度で，根本的な解決にはなっていない。

(3)の問題は記憶セル概念にもとづいた代入の弊害にある。従来型マシンでは代入により，記憶セルの内容を時々刻々変更することによって計算が進められるので，結果が他の演算に影響される，所謂副作用の問題が生じる。またプログラムの意味は状態，即ち記憶セルの値により定められるため，極めて不透明なものになる。（状態に依存したプログラムセマンティクス）例えば，x：＝y，y：＝x＋yの実行では，どちらの文が先に実行されるかによって，xとyのとるべき値が異なる。この結果，プログラムの正当性の検証，つまり意味の同一性の確認が困難となり，ソフトウェアの生産性の低下にもつながってくる。

4.2.2 関数型プログラミングとデータフローマシン

Backus によって従来型言語の欠陥－フォンノイマンボトルネックが指摘され，関数型プログラミング言語FPが提唱されて以来，関数型言語の重要性が強く意識されるようになった。pure Lisp に代表される関数型言語はLambda calculus を計算の基礎においた言語であり，オブジェクト（値）に関数を作用させてオブジェクト（値）を得る適用型（applicative）スタイルをとる。この点，関数型言語は，必要なオペランド（値）が到着すると演算を行いそして結果（値）を出すデータフローの計算方式と自然な結びつきを持っている。関数型言語の持つ特長として次の点があげられる。

(1) プログラムは関数としてとらえられる。従来型言語のような代入の概念はなく，ここで使われる変数はあくまで数学的変数を意味する。従来型言語のcall by name のように，引数として記憶セルの名前を受取り，そこへ値を書き込むこと（副作用）によって値を返すメカニズムは存在しない。

(2) 関数の値は引数の値のみによって定まる。代入（副作用）がないので，関数を計算する際に，他の計算によって気付かないうちに影響を受けるなどということはない。したがって関数計算は互いに独立となり，並列処理の可能性が生じる。

(3) プログラムは lamda abstraction を使って形成される。関数の合成や結合が容易であり，引数として関数を受け取り実行結果として関数を返す，higher order function を定義することができる。これらの機能を用いることにより，従来型言語に比べるとはるかに簡潔で明瞭なプログラムを書くことができる。

(4) リストのような構造体データ全体が1つの値として扱われる。構造体データの値を引数として

受け取り，また関数計算の結果としてその値を返すことができる。ただし，一度作られた構造体データは恒久不変のものであり，これを変更する場合には論理的にコピーが作られる。したがって関数型言語の実現には，コピーをいかに効率良く行うかが大きな課題となる。この際，関数性を損わないような配慮が必要である。

(5) 変数と式あるいは部分式との対応（同値）関係は常に不変である。同一の部分式は同じ値を示すので，同値の変数名と置き換えてもプログラムの意味は変わらない。このように，置き換えによる意味の不変性も関数型言語の1つの特長となっている。

ここに掲げた特長は，プログラムの記述性の面のみならず，プログラムの「正当性」の検証，要求仕様を満たすプログラムの合成などの面からも優れた性質を示しており，プログラムの生産性の向上が期待されうる。しかしながら，それにもかかわらず関数型言語が従来型言語ほどの普及をみなかった理由は，第一にフォンノイマン型マシンで実現する場合の効率の悪さにあった。

以下では，特に並列処理性という観点から，関数型言語とデータフローマシンの親和性について考察し，データフローマシン上で効率上の工夫がどのように行われているかをみてみることにする。データフローマシンは演算の実行契機の与え方の違いにより，データ駆動型と要求駆動型に分類できるが，要求駆動型は並列処理というよりむしろ遅延評価などの理論的側面と関わりが深いので，ここではデータ駆動型をとりあげることにする。

データ駆動の実行原理では，演算は必要なオペランドが全て揃った時点で何時でも実行可能となる。データ駆動の実行原理は従来のフォンノイマン型マシンに比べて，次のような利点を持っている。

(1) オペランドが揃った演算は他の演算の状態とは無関係に実行できるので，同時に複数個の演算を実行することが可能である。（並列処理）

(2) 演算の実行は局所的であり，演算間のデータの授受はトークン（値）として明示的に行われるので，他の演算から影響を受けることはない。（関数的処理）

(3) データによって演算が駆動されるので実行順序を陽に制御する必要がなく，したがって集中制御の必要性がない。（分散制御）

(4) 記憶セルという概念はない。演算を行う際に，データを保持するための記憶場所を意識する必要がなく，アドレス計算等の記憶管理のための，非本質的な計算を行わなくてよい。（記憶セル概念の排除）

(1)〜(4)にあげた利点が，関数型言語の処理にどのように生かされうるかについて，基本的な技術をいくつか述べておく。

A．関数引数の並列評価

データ駆動の原理に依ると，関数の引数に値が到着してから，即ち引数の評価が完全に終った後に関数本体が実行されることになるので，引数の評価は最内側から外側へと進行する。（ innermost evaluation ）このとき複数個の引数があれば，並列に評価が行われる。再帰関数の計算においては，ソーステキスト上で引数のネストに制限があっても，実行の段階で再帰的にネストが深まっていくので，みかけよりも高い並列処理性を得ることができる。

B．関数本体の部分的実行

データ駆動の実行原理をそのまま関数の起動メカニズムにあてはめると，全ての引数値が求まらない限り関数本体は実行されない。しかしながら，引数の値が求まったものから先に関数本体へ引渡し，その値で実行できる所まで先に計算を進めておけば，関数起動での待ち時間を省略することができ，実行効率を上げることができる。関数型言語を効率良く実行するためには関数リンケージの高速化が要求されるが，並列実行によってリンケージオーバヘッドを隠せるようにする他に，部分的実行のような機構を含めてハードウェア化の検討を行う必要がある。

C．Lenient cons によるパイプライン処理

リスト等の構造体データを動的に操作していく場合，リストがcons または append によって生成されている間，それを使用する関数の実行は待たされることになるので，部分的実行の効果は薄れてしまう。しかしこの問題はあらかじめセルを確保しておき，セルへの書き込みは値が求まった時点で行うLenient cons の概念を実現することにより解決される。これにより，non strict なリストをトークンとして流せるようになるので，リスト等の構造体データをパイプライン的に処理することが可能となり，著しい速度向上が得られる。またリストが全て作られるまで待つ必要はないので，それを使う関数の活性化時間も短縮化され，資源の有効利用を図ることができる。

D．コピーの効率化

先に述べた通り，関数性を遵守した処理では論理的に必ずコピーが作られる。しかし，append 等の操作が施されるリストの多くは，1回限りで繰返し参照されることはほとんどなく，また他から参照される可能性も少ないと思われるので，実際に全てコピーしてしまうことは経済的でない。この問題は参照カウントおよびrplacd 操作を利用することによって解決される。即ち，append されるリストが他から参照されていないとき（参照カウントが1の場合）は，最後尾のセルの値をrplacd を使って直接書き換え，append されるリストの途中に他からの参照があれば，そこから先についてはコピーを作るようにする。ただし，rplacd は副作用を起こすのでソフトウェア（ユーザ）には解放せず，基本関数としてappend が定義されているようにする必

要があろう。

E．変数の活用

式（部分式）を変数に置き換えることは，記述が簡略化されることの他に，プログラムを書いたり読んだりする思考の中で，途中結果を一担メモしておくという効果を持っている。この変数を利用して実行効率を上げることが可能である。同一の部分式が数箇所に現われる場合，その部分式を変数で置き変えてしまうと，一度だけその部分式を計算し残りの箇所にはそのコピーを送ることができるので，何度も同じ計算を繰り返さなくて済む。

以上考察してきた通り，関数型言語とデータフローマシンは論理的には極めて密な関係にあり親和性が高いといえよう。この関数型言語マシンとしてのデータフローマシンの可能性を実際のマシンアーキテクチャとして結実させるためには，関数型言語によるプログラミング経験の積重ねを行うと共に，データフローマシン上での実験データの蓄積が必要である。

4.2.3 論理型プログラミングとデータフローマシン

論理型プログラミングにおいては，プログラムは（一階）述語論理式として表現され，$Q \rightarrow P_1 \wedge P_2 \wedge \cdots \wedge Pn$（Q，$P_1$，$\cdots$ Pn は述語，Qが成立するためにはP_1 $\cdots$ Pn がすべて成立たねばならないと読む）のように，定理あるいは公理の形をした節（clause）と呼ばれる式の系列から成る。Kowalski が提唱した論理型言語PROLOGは，この節を一階述語論理の部分集合であるホーン節に限定したものである。例えば，論理型プログラミングでは階乗は次のように定義される。（ここではProlog 風に記述されている）

```
factorial （0，1）⊢ .                                    〔1〕
factorial （n，x）⊢ factorial （m，y）
                    ∧m＝n－1 ∧ x＝n＊y                   〔2〕
```

〔1〕式は「0の階乗は1である」こと，〔2〕式は「任意の自然数m，n，x，yに対して，mの階乗がyであり，m＝n－1かつx＝n＊yならば，nの階乗はxである」ことを表わしている。各式は節と呼び，節の左辺をhead，右辺をbody と呼ぶ。head は高々1個のgoal（述語のこと）から，そしてbody は複数個のgoal から成る。body のみからなる節を質問（question）と呼ぶ。

論理型プログラムの特徴は，記述が従来型言語のように手続き的（procedural でなく宣言的-declarative）であることである。ユーザはどのように実行させるかというアルゴリズムを書く必要はなく，何を行いたいかを公理の形で書けばよい。節のbody について順序の制約はなく，またプログラム中の節の間についても何ら順序関係は存在しない。ここに並列実行の可能性が生じ

てくる。（従来型マシンの上にインプリメントされたPROLOGでは，節のbodyは左から右へ，そして各節は上から下へという順序制約が付けられているが）論理式の中に現われる変数は，関数型プログラミングで用いられる変数同様，値を格納するメモリセルを表わすものではない。したがって論理型プログラムは関数的性質を持っている。

論理型プログラムにおける実行は，定理または公理の証明という形をとる。そこで使われる推論規則はresolution 規則のみである。resolution の過程でunification が行われ，変数の値が決定される。この過程は一種の書き換え規則の適用とみなすことができ，その意味では関数（適用）型言語と共通の計算の基盤を持っており，データフロー制御下の実行になじみやすいものである。

ここで論理型プログラミングと関数型プログラミングの違いを少し考察しておこう。関数型プログラミングでは階乗は次のように記述される。

```
factorial : function(n)  return(integer)
  =if   n=0  then  1
             else  n* factorial(n-1)          〔3〕
```

関数型プログラムでは仮引数と実引数の区別があり，関数への入力（引数）と出力（関数値）が予め定められている。例えば上式の場合，入力はnであり出力はfactorial（n）であることを示している。

一方，論理型プログラムでは各引数の入出力関係が予め定められておらず，それは節が実行される時に決められ，それに応じてbody の実行が異なった意味を持ってくる。例えば，〔2〕式においてnを入力，xを出力に指定するとnの階乗を計算するプログラムとなる。ところがnを出力，xを入力に指定すると，階乗がxとなるようなnを求めるプログラムとなる。またn，x共に入力として指定すると，xがnの階乗であることの真偽を判定するプログラムとなる。

もう1つの相違点は，関数起動と節起動の機構の違いである。関数型プログラムでは同名の関数を異なった目的のために複数個定義することは許されないので，〔3〕式の関数factorialは一意に起動される。一方，論理型プログラムでは節の起動の際にパターン照合が行われ，照合のとれた節が（複数個）起動される。このときhead の各引数への値の引渡しが行われ，同時に出力として指定される引数も決定される。例えば，〔1〕，〔2〕式において，nが0であればfactorial（0，1）が，そうでなければ，factorial（n，x）が起動される。したがって⊢factorial（3，x）という質問に対しては，〔2〕と照合がとれ，入力として3がnに与えられ，xは出力指定となる。

データフロー実行制御という観点から，関数型プログラムと論理型プログラムを眺めると，前者

ではデータの流れる方向が予め定められている（単方向）のに対し，後者ではデータの流れる方向は何も指定されていない（双方向）という違いが存在する。論理型プログラムでは，データの流れる方向が決定されるのは，パターン照合によって起動される節が選ばれ，その各引数に入力か出力かの指定が与えられたときである。即ち，節が活性化された時点で初めて，そのbody内のデータの流れの方向が定まることになる。

パターン照合により選ばれた節は複数個あれば並列に実行され，また各節のbodyはデータ駆動の原理に従って実行される。したがって入力データが揃ったものから実行が開始されるので，節のbodyの処理も並列的になる。

データフロー制御の下での並列実行は，処理速度の向上効果の他にも大きな意義を持っている。例えば，

child (Ares, Zeus) ⊢ . 〔4〕

child (Harmonia, Ares) ⊢ . 〔5〕

descendant (x, y) ⊢ child (x, y) . 〔6〕

descendant (x, z) ⊢ descendant (x, y) ∧ child (y, z) 〔7〕

において，⊢ descendant (Ares, Taro)という質問を実行した場合，節およびbodyに順序関係がある逐次型実行では，〔7〕式でdescendant (Ares, y)が無限に生成され，絶対に止まらない。しかしこれを並列実行で行うと，descendant (Ares, y) と child (y, Taro) が同時に起動され，child (y, Taro) は直ちに偽と判定されるのですぐに結果が得られる。

これまでみてきたように，論理型プログラミングとデータフロー制御とは自然な対応関係があり，データフロー制御ではいくつもの可能性がある非決定的計算を並列実行させることができる。しかしながら，論理型プログラムをデータフローマシン上で実際に走行させるためには，解が得られた時点でalternativeとして進められている計算をどのように止めるか，また数の爆発を抑えながら有限資源の範囲内で最大限の並列性をいかに引出すか，このとき後戻り制御の機能をどこまで取り入れるか等々，幾つかの基本的な問題を解決していかねばならない。

4.2.4 履歴依存性

関数型言語に共通していえることであるが，call by value メカニズムにもとづくデータフローの原理には，履歴（storage)という概念は存在しない。しかし現実の問題の中には，データベースの更新のように履歴そのものを扱わなければならないものが多数あり，履歴依存性を持った処理をデータフローマシン上でどのように実現するかが重要な問題になっている。

前の状態によって処理の内容を変更する，state 概念を持った計算はown value（1つ前の計算結果）を用いて計算するループ機構により実現できる。ループは再帰概念（即ちtail recur-sion ）で扱うことができるので，内部を純関数で構成し，かつ外部には履歴依存性を持った関数としてみせることが可能である。

次にデータベースを多数のユーザが共有して使用する場合の資源管理の問題について考えてみよう。データベースDの中のitem i をvに変更する計算D′ ＝ replace（D，i，v） を純関数的に取扱おうとすると，概念的には変更される度にデータベースのコピーが新たに作られることになる。しかしこれを忠実に行うことは非現実的である。実際には資源（データベース）を共有しながら使用せざるをえない。その場合，データベースが並列アクセスされる時の排他制御をどう実現するか，また関数性を保存しながら矛盾なくデータベースの中身を変更していくにはどうしたらよいか，という問題が生じてくる。

排他制御を行うには並列アクセスを識別し順序づけをするmanager 機能が必要であり，Streamの概念をこれに導入することが有効であると思われる。

D′＝replace （ D i，v ）という計算でitemそのものを変更する場合，データベースの無矛盾性を保証するには，変更を行う以前にそのitem が他の演算で使用されていないことを確かめる必要がある。リスト処理の場合とは異なり，これを効率良く行えるうまい解は今の所みつかっていない。今後の問題として残されている。

4.2.5 既存データフローマシン用高級言語の総括

データフローの概念そのものは古く，データの流れに着目した研究は１９６０年代後半からあった。プログラムの最適化や非同期並列処理の研究分野では，多くの研究がデータの依存関係を解析の基礎に置いており，１９６９年にはTesler らにより並列処理記述言語として単一代入言語（ Single Assignment Language ）が提唱された。データの流れを有向グラフで表わしたデータフロー言語はStanford 大学の Adams，IBM のKosinski （DFPL－Data FIow Programming Language，ＯＳ記述のため考案されたデータフロー言語でループ，非決定性ノードなどの表現に工夫がみられる），ＭＩＴのDennis （ ペトリネットの並列表記法の研究に端を発しており，データフローマシン言語としてPreliminary data flow machine lang-uage を発表），らにより進められてきたが，データフローマシンアーキテクチャの提案はDennis が最初である。（１９７４）

その後，ＭＩＴを始めとして英，仏，日においてデータフローマシンの研究が行われ，機械語に相当するグラフ表現のデータフロー言語に変換されることを前提とした，データフローマシン用高

級言語も幾つか提案されている。代表的なものとしてLAU, CAJOLE, Id, VAL, VALID等があるが，実際にデータフローマシン上で走行している言語は今の所LAUのみである。これらの多くは，非手続型言語，検証用の言語から影響を受けており，今後のソフトウェア工学の方向を示唆するものと思われる。以下に各言語の特徴について簡単に述べておく。

A. LAU（ Language a' assignation unique ）

ONERA-CERT（仏），単一代入規則を遵守したFortran 風の言語で，（並列）代入文の他に，EXPAND, CASE, LOOP, CALL, RETURN文を導入して制御命令の強化を図っている。コンパイルされたコードをデータ駆動制御のLAU system／1上で実行させるため，命令の発火制御，データタグ操作用の制御プリミティブを用意しているが，既存言語の改良の域を出ていない。

B. CAJOLE

London 大学（英），正当性の検証用言語として発表されたLucid に影響を受けた言語である。単一代入規則，非手順型（文の並び通りに実行する必要はない），関数性を特徴とし，名前の局所化機能（ with … wend ）を持つ。条件判定にはDijkstra の guarded command が用いられ，条件式はこれらを；でつないで記述される。繰返し機能が再帰的に定義され，それがコピールールに従って実行される点，およびユーザ自身が自由に構文を定義することができる機能を有している点も特色である。

C. Id（ Irvine data flow Language ）

Irvine 大学（米），適用型（関数型）の言語でAlgol 風ブロック構造を持つ。再帰概念があり，プログラムは代入を基本とする文ではなく，値を表わす式から構成される。データ構造は integer, real, Boolean, character の基本データ構造の他に，＜selector：value＞の組で構成される structure が用意されている。ただし，Lisp と同様に，型宣言は行われない。繰返しは1つずつ分解されて（ unfolding ），非同期的に実行されることを想定しており，値に新たな名前を付与する機能を持ったnew construct を用いてループで実現される。

Id では後述するVALとは異なり，ネストした繰返し構造からの自由な抜け出しは認めていない。履歴依存型の計算のためStream の概念（リスト概念から全ての値が揃わねばならないという制御を除いたもの－Leniency を持ったリスト）を導入している点も特点である。履歴依存型計算は繰返し構造（ for each … return all ）として実現され，Stream 概念の利用による並列処理性の向上が追求されている。実際にはStream は hole 概念を持ったI-stnuture として実現され，これを使ってarray などのデータ構造がサポートされる。

D. VAL（A Value-oriented Algorithmic Language ）

ＭＩＴ（米），対象指向型言語ＣＬＵを母体として設計された言語で，数値計算の分野における並列処理記述に主眼が置かれている。その名の通り，関数型（副作用のない）言語であるが，再帰概念は持たず，繰返しを実行の基本とする。

ＶＡＬの繰返し（ iter ）の実行は，Ｉｄと異なりunfold されず，同期的に行われる。その代わり，並列ボディを多数作り出してそれらを同時に実行させる，forall 機能が用意されている。ＶＡＬではiter の他にもimpliclt な同期概念が存在する。forall の中で使われるeval ，construct clause では，多数のデータの待ち合わせが必要となる。これを効率良く行うため，演算を一列にならべるlinear evaluation に代わり，これを２進木状にならべるtree evaluation を採用している。

データ構造はinteger, real, Boolean, character の基本スカラー型の他に，record, array の構造型がある。Ｉｄと違ってデータの型宣言がなされ，コンパイル時および実行時に型チエックが行われる。例外処理に対する考慮が払われ，undef （ array の添字範囲外），miss element （要素値がない）等のシグナルを返してデバッグの便を図るなど，実用性が重視されている。

E. Valid （ Value Identification Language ）

武蔵野通研（日），数値処理向きおよび非数値処理向きデータフローマシンの両方をサポートする目的で設計された汎用の関数型言語である。シンタクスはPascal に似ているが関数概念を徹底している。プログラム変数の概念はなく，プログラムはValue 定義，Function 定義，Macro 定義のみからなる。式の集合である Block 概念を持ち，Block の中ではscope ルールに従う local value の名前を定義できる。Block 自身も式であり，返す値はreturn 式によって示される。繰返しの記述は全て再帰概念にもとづいて行われる。従来型言語でのループ表現はparallel 構造およびiterative 構造に大別されるが，Valid は parallel構造をfork join 概念にもとづくfor each 式を用いて陽に記述する。iterative 構造は再帰性の観点からさらにtail recursive 構造とnon-tail recursive 構造に分類される。無名の再帰関数を定義するrecur を用いてこれらのiterative 構造を記述することができる。non-tail recursive 構造は再帰関数で記述してもよい。データ型としては，基本スカラー型の他に，record, array, list, set 等の構造型があり，コンパイル時に型チエックが行われる。その他，データの構造化機能，階層的なFunction, Macro 定義機能およびstream処理機能を有し，higher order function の定義も可能である。

以上データフロー高級言語の概観を行ったが，データーフロー高級言語の研究は未だその緒につ

いたばかりであり，理論的提案の段階のものが多い。これを実用に供していくには，

(1) 関数型の言語で履歴依存性をどう扱うか

(2) (1)に関連するがI／Oをどのようにサポートするか

(3) 非決定性制御をどのように提供するか

などの原理的な問題を解決していくと共に，並列処理の記述手法の開発や抽象データ型の導入など，プログラム検証手法との関連づけを行っていく必要がある。

4.2.6 非決定性処理

非決定性問題とは，処理の順序が実行前には決定することができない問題のことである。例えば，2つの並列プロセスが同じ入力装置を使う場合どちらが先に要求を出すかは実行前には決まらないことがある。このような状態はオペレーティングシステム，リアルタイムプログラム，データベース等を取り扱う時に問題となる。

人工知能の分野では解のトリーを探索する時どの技を探索すればよいかを実行前には決定できない問題が知られている。

この問題が言語にどのように反映されるかを以下に述べる。

A. Guarded Command

この命令は条件式を評価し，その値が真の場合にその条件式に付属した文を実行する。従って条件式が互に排他的でない場合は2つ以上の条件式が真になることが起こり，どの文を実行するかに関して非決定性が生じる。

データフローマシンでGuarded Command を実現するとどうなるだろうか。まず条件式は並列に実行することができる。これは関数型であれば互に副作用がないので問題はない。しかし2つ以上の条件が真となる場合には，この Command の制御をどう定義するかが問題になる。各条件式の値を一カ所に集め，その中から真となるものを1つ選んで実行するという方法がある。また真となった条件式に付属する文はすべて並列に実行してしまう方法もある。前者の場合はこのCommand の value が1つに定まるので問題はないが，後者の場合には1つの value を定める必要がある。実行されたすべての文の値を集めてリストにする方法はどこか一カ所でも収束しないとこの Command の値が定まらないのでよくない。一斉に実行したもののうち，一番早く実行を終えたものがこのCommand の値となり，他の値は吸収されてしまうような機構が望ましいように思える。一斉に実行された文が更に多数の関数を呼ぶことにより数の爆発を起こす恐れもある。これを自動的に suppress する方法として lazy evaluation がある。この場合も lazy evaluation により生じた suspension を並列に実行すると数の爆発の防止には役立

たないし，並列実行可能な環境下で suspension を止めてしまうのはもったいないかもしれない。手続き単位で互に通信を行い，お互の実行を制御するような（例えばHoareのCommunica-ting Sequential process）を与えてユーザが数の爆発の制御をするのも1つの方法であろう。

B．リソースマネジャー

2つ以上のプロセス（関数）が同じリソースをアクセスする場合にはどちらが先に要求を出すか実行前に定めることができないという非決定性のため，各要求がどのプロセスから来たかを区別する必要がある。この問題に対して Arvind, Gurd 等はマネジャーという機構を提案している。その仕組は，各プロセスは必要なパラメータと return link を持ったトークンをマネジャーに送る。マネジャーは入口でそのトークンのプロセスを識別する印を付ける（Arvindは entry ポイントを変えることによって，Gurd は index を使うことによってこれを行っている）。こうしておいてから必要なパラメータを用いてリソースを使用し，終るとその結果と待たせておいたリターンリンクとを前述の識別子を用いてマッチさせ要求があったプロセスに送り返す。この機構を実現するためには entry, exit命令（または index 処理命令）の実現が必要である。またリソースの数が実行前に定められない時はこのマネジャーを動的に create する機構が必要となる。

C．Communicating proccss

各プロセスが通信し合うことにより実行を制御する機構は，ユーザが非決定性のある問題を制御するために必要となる。逐次処理で解のトリーを探索する時の典型的な方法は縦型探索を行い fail すればバックトラッキングでもとの状態に戻して別の枝を試みるというものである。この他に横型探索や両者をmix した探索も行われている。データフローの場合並列実行が可能なので横型またはmix 型が比較的簡単に実現できるのではないかと思われる。この場合幾つかのプロセスが互に通信し，その実行を制御する機構が必要である。このような機構の1つとして Hoare の Communicating seguential process を考えてみると，変数Vを単一代入の規則を満たすように書き換えること，alternative command を用いて，communicating process を記述することによって実現できそうであり，有力な候補である。

以上非決定性問題とそれに関連した機構について述べた。第5世代計算機の高級言語にはこのような機構が導入される必要があるだろう。

4.2.7 計算モデル

計算モデルは言語のセマンティクスを表現したものであり，従って言語のスタイルにも大きな影

響をおよぼしている。また計算モデルは計算の進め方を規定したものであるからマシンの構成についても当然影響する。これまでに提案されたモデルを整理してみると図4.2.1のようになる。*注1 またこれらのモデルをデータ処理方式と制御方式で分類すると図4.2.2となる。以下に各モデルの特徴を述べる。

言　語	ノイマン型	単一代入型	関　数　型
計算モデル	control flow	data flow	reduction
マシン構成	centralized	packet communication	expression manipulation

図4.2.1　計算モデルの分類（Ⅰ）

		データ処理方式	
		by value	by reference
制御方式	by availability	data flow	control flow
	by need	string reduction	graph reduction

図4.2.2　計算モデルの分類（Ⅱ）

＊注1

この図は望ましい組み合わせを考慮して整理したもので他の組み合わせ，例えば関数型言語をdata flowモデルで表現することも可能である。なお，demand-dvivenはreductionに含めてある。

A. control flow

データ処理方式はby reference,制御方式はby availability により動作する。計算の例を図4.2.3に示す。この計算は1個のcontrol token が最初の計算にトリガーをかける。この時データbとcの値は既に指定された記憶場所になければならない(by availability)。

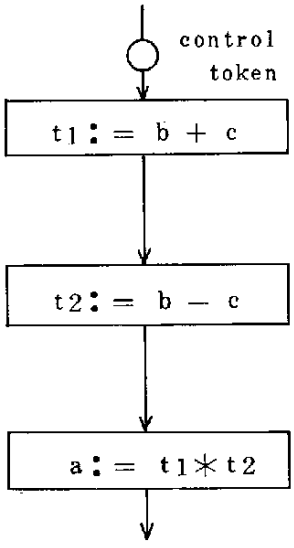


図4.2.3 control flow の例

指定された場所からbとcの値を取り出し(by reference),加算を行いt1の場所へ納める。次にcontrol token を次のパスへ出しt2の計算に進む。これをノイマン型のマシンにあてはめればcontrol token はプログラムカウンタであり,値を納める場所はメモリとなる。

この計算はt1とt2を並列に行うことができる。この場合はcontrol tokenを2つ用意すればよい。(図4.2.4)この場合大切なことはt1とt2の計算が互に干渉しないことを保障しなければならないことである。このモデルにもとづくいわゆる手続き型言語ではこの保障を記述することが大変難しく,従ってcontrol flowは並列計算にあまり適さないモデルといえよう。

B. data flow

このモデルは,データ処理方式はby value,制御方式はby availability である。計算の例を図4.2.5に示す。この計算はtoken がbとcの値を持って行く(by value)。tokenは分岐点でコピーされて2つに分かれ加算と減算の箱に到着する。それぞれの箱では2つの

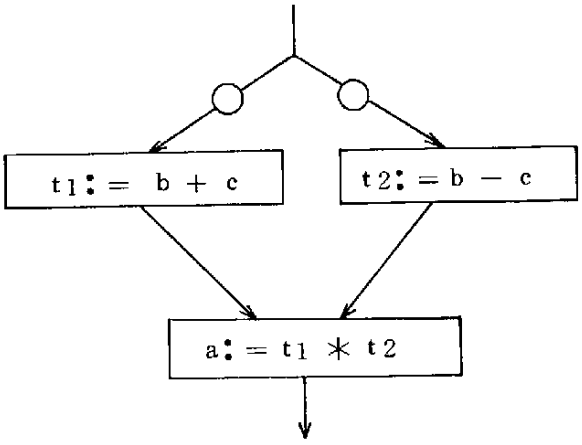


図4.2.4　control flow のマルチ化

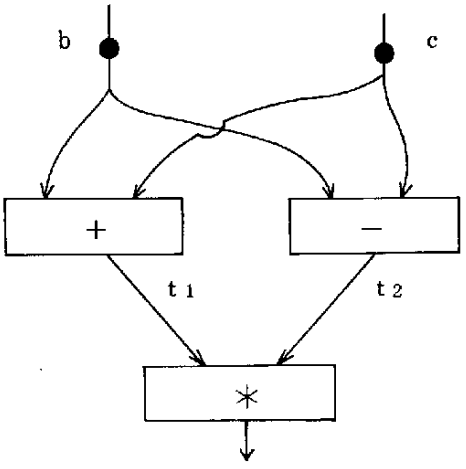


図4.2.5　data flow の例

data token を受け取った時点で by availability は保障されている。何故なら token 自身が値を運んでくるからである。この機構であれば2カ所以上で使うデータはコピーするため互の干渉はなくなる。従って言語レベルで必要なことは，必要ならばデータのコピーを保障することである。これは単一代入型または関数型の言語を用いることにより容易に達成できる。またこのモデルではdata token のみが演算の実行に必要なものであるからデータさえ到着すればグラフ内のあちこちの演算を並列に行ってよいということになる。従って並列計算に適したモデ

ルといえる。

C. reduction

これは制御方式はby needであるが，データ処理方式は式そのものを変更していくstring reductionとポインタを利用してみかけ上式の変更を行うgraph reductionがある。図4.2.6にstring reductionの例を示す。この計算はaの値を要求する所から始まる(a)。この要求によりaはその定義に置き換えられる(b)。次に図(b)内のうち定義と置き換え可能な部分が置き換えられる。この場合bとcが値4と2に置き換えられる。次に加算と減算が行われ，その値に置き換えられる(c)。この手続きを繰り返し解にいたる。graph reductionはこの過程をポインタを用いて行う。置き換えの順序についてはoutermost innermostなど色々な方式が提案されている。reductionはdata flowとほぼ同等の性質を持っているが，データ構造の取り扱いと実現効率の点に疑問が持たれている。しかし現在の所あまり研究されていないので更に検討が必要である。

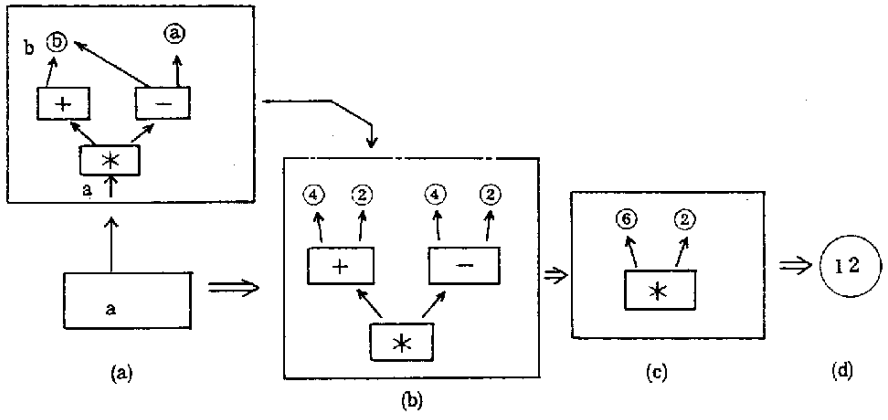


図4.2.6　string reductionの例

以上3つのモデルについて述べた。第5世代計算機システムにおいて，どのモデルが適当であるかは言語，効率，理論的な良さ等を総合的に検討して決定する必要があるが，現状ではデータフローがアーキテクチャを考え易いため多く取りあげられている。

4.3 データフローマシンの制御方式と機能要素

本節では，データフローマシンの制御方式，必要と思われる命令セット，およびデータフローマシンを構築するにあたって必要な構成要素に関する技術的動向について述べる。

4.3.1 計算機構

一般に計算の制御機構を大別すると，コントロールフロー（control flow），データフロー（data flow），およびリダクション（reduction）の3つに分類される。これらは，1つの命令が他の命令（または命令群）の実行をどのように制御していくかという制御機構にもとづいて分類したものである。

コントロールフローは，制御を逐次的に行い，並列実行時はその旨明示的に命令（例えば，FORK，JOIN命令等）により制御するものを指し，従来の計算機システムの殆んどがこれに属する。命令間のデータの受け渡しは記憶セル（レジスタやメモリ）を介して行われる。

これに対して，データフローは，各命令の実行がそれに必要なオペランドの準備状況によってのみ制御されるもので，必要なデータが揃えば各命令は実行可能となる。

一方，リダクションは，文字列の処理によって計算を進めるものであり，プログラムの各命令（関数呼出し）にその関数の定義を当てはめて置き換えしながら最終結果を得る。即ち，各命令の実行に必要なデータに対する要求が，そのデータを生成する命令の実行を引き起こすという要求駆動方式とみなすことができる。

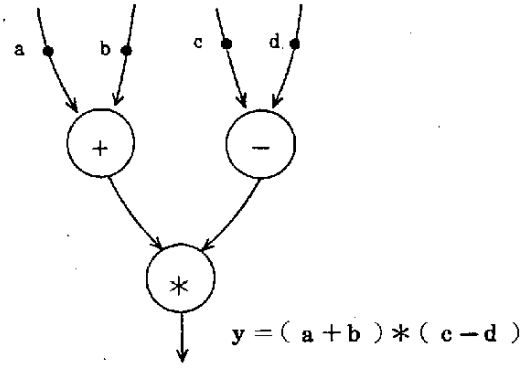
コントロールフロー方式はマシン動作が明快であり，ベクトル演算のような単純な並列処理には向いているが，一般の複雑な並列動作を制御するのには向いていない。従って知識情報処理のように複雑な並列処理を伴う応用に対しては限界がある。一方，データフロー方式はアルゴリズムに内在する並列性を自然な形でマシン上に実現できることが期待される。データフロー方式は関数型の特性を有する単一代入言語を基礎としており，処理に副作用がなく自然な形で並列処理が記述でき，意味的に理解し易いという特徴がある。リダクション方式は，この関数性をさらに徹底したものであり，機械語レベルの実行においても関数性を保持しており，いわば関数型言語の直接実行マシンと考えることができる。

以下，データフロー方式およびリダクション方式における制御方式，および，両者の間の特徴比較について述べる。

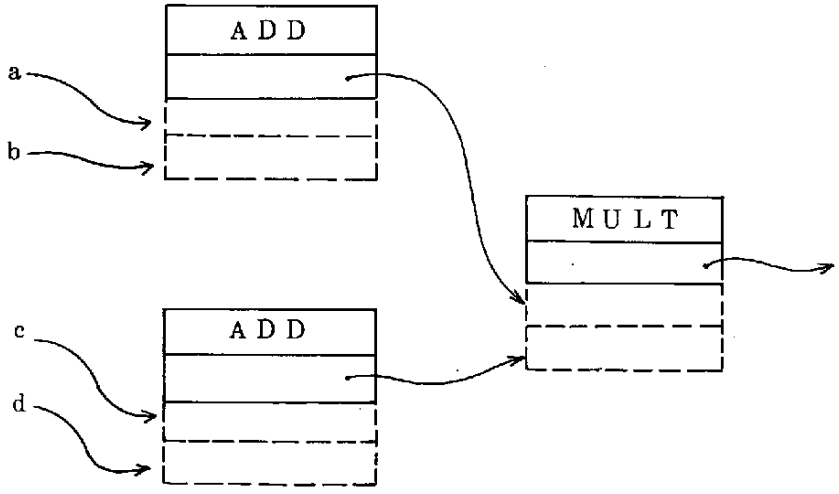
4.3.2　データフロー―マシンの制御方式

データフローマシンの機械語レベルにおけるプログラムは有向グラフで表現される。グラフ内のノードはオペレータに相当し，アークはデータ経路を表わす。グラフ内の各ノードはその入力アークからのデータの到着を待っており，データが全て揃うと実行可能となり，ノードの実行制御が行われる。ノードの実行が終るとその結果データは出力アークで結ばれた次のノードへ送られ，次のノードの実行可能判定が行われる。

図4.3.1(a)および(b)にそれぞれ，データフローグラフ例，および，そのグラフをマシン上で表現したときのプログラムテンプレートの例を示す。同図において，入力データa，b，c，およびd



（a）データフローグラフ例



（b）プログラムテンプレート例

図4.3.1　データフログラフおよび対応するプログラムテンプレート例

がいずれも到着したとき2つの加算命令は同時に実行できる。それらの命令の実行結果は次の乗算命令に渡されて，乗算命令の実行を起動する。このように，データフローマシンは，各ノードの実行制御がその入力データの到着状態にのみ依存して行われるというデータ駆動型の特性を有しているため，並列処理を容易に実現できる。

実用的なデータフローマシンを考える場合，ノードの種類としては，基本演算の他に条件分岐，ループ制御，関数呼出し等の機能が必要となるが，それらを考える時，次のような点が設計上の選択ポイントとなる。

A．同期式／非同期式の選択

同期式または非同期式の選択は関数やループの起動方法と関連している。即ち，関数やループの起動に際し，全入力引数が揃うのを待つ方法が同期式であり，入力引数のいずれかの到着により先行起動する方法が非同期式である。前者の場合，関数起動ノードを通常のノードと全く同様に取扱うことができ，インプリメントが容易であり，また関数やループの起動制御の様子がわかり易いためデバッグが比較的容易になるという特徴がある。

これに対して，後者の方法はその引数の到着に先行して，またはその引数のいずれかの到着によって，関数起動制御を行うのでインプリメントはやや複雑になるが，並列性を制限してしまう恐れがなくなる。

B．トークン識別子の付与

トークン識別子を付与するか否かは，プログラムコードを共有するか否かと密接に関係している。以下，両者の場合についてその得失を述べる。

a．識別子なし法

トークン識別子を用いない場合，関数やループ本体のコードの共有をサポートするため以下のような方法が提案されている。

(1) ready-ack 信号法

ＭＩＴのＪ．Ｂ．Dennis[1] の提案した方法であり，各ノード間のアーク（データ経路）ごとに，そのアーク上にデータがなくなったことを保証するために，データの流れと逆方向にack 信号を送る方法である。この方法では，トークンの流れがack 信号を使用しない場合に比して約2倍となり，システム内のネットワークのスループットが低下する恐れがある外，並列度も制限を受ける恐れがある。しかし，この方法は，トークンの受信側で受取ったデータに誤りがあることを検出した場合，例えばack 信号の代わりにnack 信号を送信側に送ることにより，再送機構を容易に実現できるため，高信頼度ハードウェアの実現という副次効果を生み出す可能性がある。

(2) コード使用のシリアライズ化

この方法は，関数やループ本体のコードをシリアライズして使用するため並列性が大きく制限される可能性がある。(この例としては，例えば文献[2]を参照)。また，関数の再帰呼出しの実現が困難なことや，各コードごとにモニタのような監視機構が必要となるという欠点がある。

(3) コードの複数コピーの生成

コンパイル時にあらかじめプログラムの動的特性がわかっているような場合，関数やループ本体のコードの必要分だけを用意しておけばよい。また，この方法は並列度を高めるのに非常に有効と思われる。しかし，上記の方法は一般的な応用には適用できないので，実行時に動的にコピーを生成する等の手段が必要となろう。この場合，コードのコピーのためのオーバヘッドが問題となる。

b．識別子を付与する方法

これに対して，関数やループの起動ごとにトークンに異なる識別子を割り当て，コードを各起動要求間で共有する方法が提案されている(例えば，Arvind の文献[3]を参照のこと)。この方法は，1つのコードメモリ空間に対して識別子のビット数で表わされるだけのオペランドメモリ空間を用意し，各々の識別子毎にコードメモリを共有する。これによって，各起動要求毎にコードをコピーする必要がなくなるが，オペランドメモリ(アクティビティ記憶域)の空間が非常に大きくなる。従って，実用的規模のマシンを考える場合，物理メモリへのアドレス変換機構が必要となるが，この問題はアクティビティ記憶に連想機能を備えることで解決される。即ち，命令の入力オペランド数がたかだか2つまでのとき，その命令のオペランドが揃ったか否かは，識別子と命令アドレスをキーとしてアクティビティ記憶を連想検索することによって検出できる。アクティビティ記憶の連想機能は連想メモリや並列ハッシュ等の手段によって実現されている。

この識別子を付与する方法では識別子の割り当て管理のためのオーバヘッドの問題が生ずるが，これは専用ハードウェアを用意することによってさほど大きな問題とならないと推定される。

識別子を付与する方法においては，前記のコードのコピーを生成する方法における利点を主として負荷分散の立場から実現できるため，本プロジェクトで開発するデータフローマシンは識別子を付与する方法を採用することが有望であると推定される。これらの方式の選択は，最終的にはシミュレーションによる性能の評価等をもとにして行われるべきである。

4.3.3 リダクションマシンの制御方式

リダクションマシンは関数適用を関数の定義と置換することにより計算を進め，その機械語レベルの実行においても関数型の特性を保持しているものであり，データフローマシンにおけるスイッチノードやマージノード，ループ制御等のような特殊なオペレータを排除している。従って，リダクションマシンは関数型言語の直接実行マシンともいえる。

リダクションマシンにおいて注目されるのは，それがKellerのAMPSマシン[4]で見られるようにデマンド駆動モデルを比較的容易に実現できることにある。デマンド駆動モデルにおいては，プログラム内のouter-most ノード（最終結果を得るノード）から関数適用を仰り，順次内部ノードへ向かってその実行の要求（デマンド）が転送される。要求がinnermostノード（オペランドが揃い実行可能となったノード）へ達すると，その実行制御が行われ，結果は要求と逆方向に転送され，オペランド待ちとなっているノードの実行を制御する。この後者のフェーズはデータ駆動型であり，いわゆるデータフローマシンの制御方式と同一である。

このデマンド駆動モデルにおける特徴の1つは，要求の転送を制御することにより，必要になったときに始めて計算を行うという遅延評価機能を容易に実現でき，不要な計算の進行を避けることができることである。

4.3.4 データフローマシンとリダクションマシンの特徴比較

データフローマシンとリダクションマシンの特徴を要約すると以下のようになる。

A．コンパイラ

データフローマシンにおいては単一代入言語をデータフローグラフ，またはそれと等価なプログラムテンプレートにコンパイルする必要があるのに対して，リダクションマシンは関数型言語の実接実行マシンであるために，コンパイルが不要，またはコンパイラが単純である。

一方，言語上の特徴としてリダクションマシンはループのような関数性をなくす構造を排除している。従って，プログラマまたはコンパイラはループ構造をtail recursionのような形式に変換する必要がある（従来言語に馴染んだプログラマにとっては，ループ構造を許さないプログラムの記述はかなりの負担になると思われる。従って，コンパイラによる自動変換が望ましい。）

B．遅延評価機能

この機能を実現することにより，ある種のプログラムでは不要な計算の進行を避けることができ，プログラムの停止性を保証することができる。リダクションマシンにおいては比較的容易にこの機能を実現できると考えられるが，データフローマシンにおいては例えばlagy consのようなノードを導入する等の工夫を要する。[5]

C．セマンティックス

リダクションマシンにおいてはプログラム中の関数適用を呼出された関数の定義と置換しながら計算が進められる。即ち，プログラム自体を変形しながら処理が進行していく。この過程は例えば紙面上で中間結果を書きながら代数式の解を求めていく操作に似ている。従って，セマンティック上わかり易い。また中間結果を順次トレースすることにより，計算の過程を理解し易いため，デバッグ面においても有用である。

D．ハードウェア構成と処理速度

リダクションマシンにおいては，その機械語が比較的高レベルに設定されているためハードウェア構成上の負担はかなり大きい。即ち，命令のオペランドとして任意のデータ構造が許され，しかも，そのオペランドをコピーするため，命令実行毎に柔軟なメモリ管理機構と発火制御機構が必要となる。また，基本的にコピー規則を用いているので，ネットワーク上を大量のデータが移動する恐れがあり，ネットワークの負荷がかなり大きくなるであろう。このため，ハードウェア規模や処理のオーバヘッドがデータフローマシンに比してかなり大きくなると推定される。

現在，リダクションマシンのハードウェア構成はデータフローマシンのそれに比して提案レベルのものが多く，その詳細が不明であり，今後の研究成果を待たなければならない。

E．入出力処理

データフローマシンにおいては履歴依存性を有する入出力処理方式があきらかになりつつあるのに比して，リダクションマシンにおいてはその制御方式の詳細はまだ不明である。

以上の考察より，実験機第1版としはデータ駆動型にもとづいたデータフローマシンの開発を行う。リダクション方式については，長期的視野に立った研究が必要であり，基礎研究の進展に伴ってその利点をマシン上に取り込む形で研究を進めていくのがよいと思われる（前期は上記マシン上でシミュレートまたはエミュレートしてその評価を行う）。

4.3.5 非決定的処理の制御

探索木を辿りながら次々とコルーチンを生成するような非決定的処理を実現しようとするとき，データフローマシンではやや問題が生ずる。即ち，探索木の1つの枝で解が見つかったとき，他の実行中のコルーチン群を如何に停止させるかという問題である。従来のバックトラック法にもとづいた探索では処理が本質的にシーケンシャルであり，この問題は生じない。

1つの解決法としては，各探索処理にユニークな識別子を割り当て，この識別子を全てのトークンに付与し，解が見つかった際にその探索処理に割り当てられた識別子を有する全てのトークンやアクティビティを消滅させる機構を用意することがあげられる。このとき，構造データを消去する

ことも必要である。従って，構造データも識別子毎に管理する機構が必要となる。

一方，オペレーティングシステムの資源管理プログラムにみられるように，その中に状態を記憶しておき，複数の不特定利用者からランダムに（非決定的に）使用されるものがある。このような非決定的処理の例としてはC. Hewitt のメッセージ駆動の概念,[6] E.W. Dijkstra の guarded command,[7] C. A. R. Hoare の communicating seguential proess[8] 等があげられる。

これらをデータフロー方式で実現する例として，California 大学（現MIT）のArvind等の研究[3]やManchester 大学のA. J. Catto 等の研究,[9] があげられる。いずれも不特定利用者からの要求を非決定的に受け付けるためのオペレータや処理結果を元の利用者に正確に戻すためのオペレータ等を用意することによって実現している。

このような処理は実用的なシステムには不可欠と考えられるので，十分な検討を必要とする。

4.3.6 命令セット

A. データーフローマシンの命令セット

一般的には，以下のような命令が基本となる。

a. 基本演算命令

基本データタイプに対する演算機能を有するもので，以下のようなものがあげられよう。基本データタイプのデータ幅は，実用的なものとするためには３２～６４ビット程度は必要と思われる。

(1) 加減乗除命令

入力データタイプとしては，整数，浮動小数点，文字，およびポインタ等があげられる。なお入力オペランドの正当性チェック（例えばポインタ同士の加算や乗算は誤りとする）や，データタイプの自動変換機能（例えば，整数と浮動小数点の加算はいずれかのタイプに変換してから行う）も備えていた方が良い。

なお，結果のデータとして数値そのものの他にキャリー等も出力する方法や，加減乗除を組み合わせた多項式計算のような高機能命令も考えられよう。

(2) 比較命令

入力データタイプとしては整数，浮動小数点，文字，およびポインタ等が考えられる。結果は論理値となる。

(3) 文字または文字列処理命令

文字または比較的短かい文字列の比較，加減算，および連結等の機能を有する。インプリ

メントによって異なるが，長大な文字列は構造データとして構造データ処理装置内で処理されることになろう。

(4) 論理演算

論理値の否定，論理値間の和，積および排他的論理和の機能を有する。

(5) ビット操作命令

入力データの特定のビット位置のセット，リセットおよびテスト，入力データ間のビット毎の和，積等の機能を有する。

(6) シフト命令

入力データのシフトや回転の操作を行う。

b．構造データ操作命令

配列，レコード，あるいはリストのような複雑な構造を有するデータの格納，参照，および操作を行う命令である。

構造データの取り扱い方法としては，それが参照される度にコピーする方法と，そのポインタを用いて何らかのアクティビティ間の共有を実現する方法がある。

前者の方法は単純明快でデバッグも容易になると思われるが，構造データを参照するアクティビティがシステム内で分散している場合は命令実行毎に構造データをネットワークを介してコピーする必要があるためネットワークの負荷が重くなる（従ってシステムスループットが低下する）恐れがあることや，アクティビティ格納域に可変長データを格納するための柔軟なメモリ管理機構が必要となること等の問題がある。

これに対して，従者の場合命令間で渡すオペランドとしては固定長のポインタで実現できるためネットワークの負荷は軽くなるが，実際の構造データへアクセスするためにはポインタをもとに命令を構造メモリへ渡すための手段が必要となる。また，ハードウェアの構成法によってはこの転送遅延時間が大きくなり，性能低下をもたらす恐れがある。この方法のもう1つの問題はガーベッジコレクションである。即ち，各アクティビティ間で構造データを共有するため構造データ領域をどの時点でゴミとして再生するかの制御が困難であることである。一般には，このために参照カウント法が使用される。即ち，各構造データ（の要素）毎に，その構造データを共有しているアクティビティの数を管理しておき，各アクティビティが終了する毎にそのカウントをデクリメントする。従って，この値がゼロになればそのデータは他の構造データ格納のために再使用できる。また，共有されているデータの一部が特定のアクティビティによって変更された場合，他のアクティビティに影響しないように，データの一部または全部を自動的にコピーする機能も必要とされる。この方法では，この参照カウントの制御の方式やそ

のオーバヘッドの評価が必要である。

c．条件分岐用命令

条件分岐を制御するための命令であり以下のものがある。スイッチ命令，T（true）ゲート，F（false）ゲートはデータ入力と論理値入力の2つのオペランドを受け取る。スイッチ命令はその出力ポートとしてT（true）ポートとF（false）ポートを有しており，論理値入力がT（true）のときTポートに，F（false）のときFポートに，データ入力からの値を出力する。TゲートおよびFゲートは，それぞれ，スイッチ命令のTポートおよびFポートのみを有する命令である。マージ命令は条件式の終りで，スイッチ命令で分岐したトークンを集める機能を有する。プログラムに誤りがなければ（well-formed であれば）マージ命令は明示的に設ける必要はない。

d．ループ制御命令

トークンにループ識別子を付与する方法では，ループ識別子の値を更新して新たな識別子を割り当て，ループ本体に渡される全ての引数トークンの識別子を新しい値に更新する命令が必要である。またループからの出口で，結果トークンの識別子をループに入る直前の値に戻すための命令を要する。また使用されてしまった識別子を他のループ要求で再使用する恐れのある場合は，それまで使用していた識別子を有する残留トークンを全て抹消するような識別子の解放機構を要する。一般にこの方法では，並列性を向上させるため，ループ引数のいずれかが到着したときに先行してループを起動するような方法が提案されている。

これに対して識別子を使用しない方法では上述の識別子の制御機能は不要である。しかし，後のループのトークンが先のループのトークンと衝突するのを避けるために，一般には次のループに入る毎にループ本体をコピーする方法や，ループ本体の使用をシリアライズ化するためにループの入口で全引数の同期をとる方法が提案されている。前者の方法ではコードをコピーするためのオーバヘッドが問題となるし，後者の方法では並列性が制限される恐れがある。

ループ本体の各instance が各々独立に実行できるような特殊な応用（例えばベクトル計算）においてはコンパイル時にあらかじめ複数のループ本体を用意しておき，それぞれを同時並列処理させる方法も提案されている。これに対して動的アーキテクチャにおいては特定の処理要素にのみ負荷が集中するのを避けるため，ループ本体を異なる処理要素に割り当てるためのスケジューリングが必要となろう。

e．関数呼出し制御命令

上記のループ制御命令とほとんど同様である。但し，関数呼出しの場合はその関数が様々のプログラム間で共有されるため，その戻り先を実行時に決定する必要がある。一般には，関数

の戻り先のノード位置を示す定数（ポインタ）を引数として呼出された関数に渡してやり，呼出された関数はその結果の戻り先を実行時に設定するような命令を用意する必要がある。また，識別子を使用しない場合，関数の再帰呼出しが発生するとコードをシリアライズして使用することが不可能となる。従って，この場合はコードをコピーする方法しか許されない。

f．AND／ORゲート

通常の命令は全てANDゲートの機能を有する。即ち，その全入力オペランドが揃ったときに起動される。これに対してORゲートは入力データのうちいずれかが到着すれば直ちに実行可能となり，その実行制御が行われる。通常，先着オペランド以外のデータはそれ以降そのノード内で吸収される。このORゲートは一種の記憶ノードと考えられ，関数の部分実行（関数の全引数が揃うまで待たずにいずれかの引数が到着すれば直ちに関数を起動してその一部の実行を開始する）を実現するため等に使用される。

g．記憶ノード

入力オペランドを記憶する機能を有するノードであり，非決定的処理における状態記憶やループにおける定数の記憶等のために使用される。この記憶機能は独立の命令としてインプリメントすることもできるが，通常の命令の一部の制御機能としても実現できよう。

h．入出力命令

入出力装置の特性に応じたread ／write 機能を有する。例えば，端末のようなシリアルインタフェース装置の場合は1文字read ／write する機能があればよいし，ディスクのようなブロックインタフェース装置の場合は装置と構造メモリ間のブロック毎のread ／write 機能を実現することになろう。このための低レベルの命令として入出力装置状態のセンス，入出力装置の起動および停止等が必要となろう。また入出力装置の終了等をプログラムに知らせるための割り込み制御手段も必要となろう。

これに対して，共有不可能ファイルの排他的使用，デバイスに依存しない統一的な入出力制御，及びリスト単位の転送のような高レベル入出力インタフェースも必要である。これは主として制御ソフトウェア等で実現することになろう。

i．OS等制御命令

データフローマシンにおいても従来のオペレーティングシステムに見られるようにジョブやプロセスの開始や終了，及びジョブ間の保護機構やその優先順位づけ等の機能は必要と思われる。このような機能は知識情報処理のように非決定的な制御を必要とする処理には不可欠と思われる。

データフローマシンではアクティビティがシステム内に分散されるため，例えばジョブの停

止機能を実現する場合，関連するアクティビティのみを終了させるのは困難である。

(1) ジョブやプロセスの終了

この機能を実現する手段の1つとしては，各ジョブやプロセス毎にユニークな識別子を割り当て，システム内のアクティビティにこの識別子を付与し，この識別子を手がかりにしてジョブまたはプロセスのアクティビティを消去させる方法が考えられる。

もう1つの手段としては従来のオペレーティングシステムに見られるように，各ジョブやプロセスに対して時間スライスを割り当てる方法がある。即ち，ある時間には唯一つのジョブまたはプロセスに属するアクティビティのみが動作するようにする。これによって，ジョブやプロセスの管理は容易になる。しかし，ここではアクティビティのスワッピング機構や特定アクティビティの選択的起動のような機構が要求され，ハードウェア構造や時間スライスの割り当てて制御がかなり複雑になる，システムスループットが低下するという欠点がある。

(2) ジョブやプロセスの一時実行停止

また，システムのデバッギング等の道具としてジョブやプロセスの実行を一時停止し，必要に応じて再開できる機構が必要である。特に，データフローマシンでは様々のアクティビティが並列に動作するため，プログラム実行には再現性がないと考えられるのでこのような機構は必須となろう。

このためのソフトウェア的な制御手段としてはプログラムの適当な位置にゲート命令を挿入しておき，その制御入力を制御する方法がある。図4.3.2にその概念的なデータフローグラフを示す。同図において，制御入力がなければ，トークンは処理Aを終えた時点で停止する。

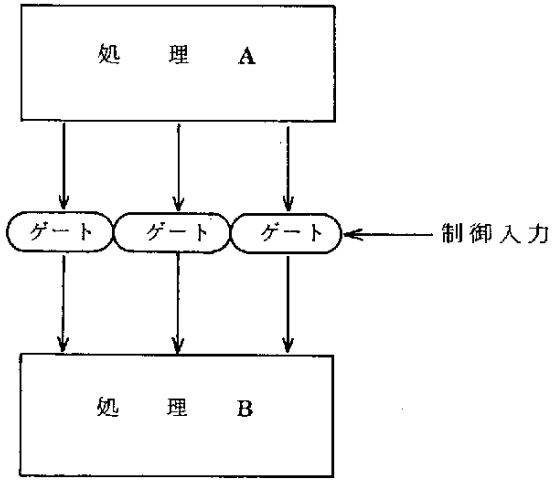


図4.3.2　処理の一時停止制御法

制御入力がゲートに加えられるとゲート命令が起動されて処理Bの実行が始まる。このとき，ゲートとしてT（ true ）ゲートを用いれば，ゲート入力に偽（ false ）の値を入力することによって以降の処理を中止させることもできる。

もう1つの制御手段としては関数やループの起動を制御するスケジューラに特定のジョブやプロセスに対してその起動をペンディングさせるように知らせる方法がある。

ハードウェア的な実現手段としては通信路の一部で特定のジョブやプロセスに属するトークンのみを貯えておき，先に進まないようにする方法がある。このためには，トークンにジョブまたはプロセス識別子を付与し，特定のジョブまたはプロセスに属するトークンのみを取り出してキューに貯えるような機構が必要となる。

これに対して，ＬＡＵシステム 2）のように各アクティビティ毎に環境制御タグビットを付加する方法も考えられる。ＬＡＵシステムではこのタグをＤＰＳ（ Data Production set）と呼ばれる関数（代入文の集合）の実行を制御するのに用いている。即ち，命令毎にそのオペランドの到着状態を示すオペランドタグを設けると共に，ループ本体のような順次処理を必要とするＤＰＳの制御を行うために環境制御タグビットを用意している（このシステムではループ識別子を使用していないためループの実行を順次制御する必要がある）。

このような概念を拡張すれば，ジョブやプロセスの一時実行停止機能を実現できる。しかし，この方式は特定のジョブやプロセスに属する全アクティビティの環境制御タグを操作するためのオーバヘッドが問題となる恐れがある。

(3) 優先順位の制御

データフローマシンにおいては，様々のジョブやプロセスに属するアクティビティが並行して実行される。従って，従来の計算機システムのようにプロセスを明示的に切り換えるディスパッチング機能は不要である。しかし，入出力装置の制御やリアルタイム処理は他プロセスに対して優先的に実行する必要があろう。即ち，プロセス毎に優先順位づけを行う必要が生ずる。

この場合，優先順位に応じてトークンの転送を制御するような機構が必要である。1つの実現手段としては，トークンに優先順位を付与し，転送路の1部に複数のＦＩＦＯ（ first-in first-out ）キューを用意し，キューの入口で優先順位に応じて各キューにトークンを分配する手段と，キューの出口で優先順位に応じてトークンの取り出し順序を制御する手段を設ける方法がある。図 4.3.3 にその構成図を示す。

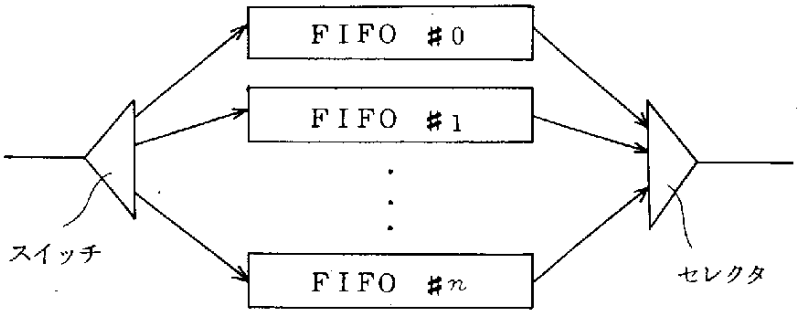


図 4.3.3　優先順位制御回路例

B．リダクションマシンの命令セット

リダクションマシンは関数型言語を直接実行するマシンである。従って，命令セットは高級言語，またはそれを変換した中間言語となる。この中間言語はその命令レベルで関数型の特性が失われない程度に設定されよう。それより低レベルのメモリ管理機能や関数呼出しにおける引数の置換や関数の発火制御等は大部分がハードウェア（マイクログラム等）に組み込まれることになろう。

従って，命令セットを明示的に規定するのは困難であるが，以下のような機能を有するプリミティブは最低限必要となろう。

a．基本演算関数

上記データフローマシンと同様な機能を有する基本演算関数をシステム組み込み関数として用意する。

b．条件分岐関数

ＬＩＳＰのＣＯＮＤ式やif-then-else構文のような機能を有する。条件式を評価することによって指定された式の選択を行う。

c．構造データ操作関数

リストのような構造データを操作したり参照したりする機能を有する。ＬＩＳＰのＣＯＮＳ，ＣＡＲ，ＣＤＲ等に相当する関数群である。

d．入出力関数

入出力装置とのデータの入出力を司どる関数である。この関数は一般に履歴依存性を必要とし，データフローマシンと同様な工夫が必要である。

e．関数呼出し制御

関数呼出しがあった場合，その関数の定義と置換しその引数を代入する機能が必要である。従って，強力で柔軟なメモリ管理機構をマシン内に内蔵していなければならない。この外，遅延評価機能を実現するためには，その要求があるまで関数適用を遅らせるような工夫が必要である。

4.3.7 機能要素のハードウェア構成

A．データフローマシン

図4.3.4にデータフローマシンの一般的な構成図を示す。同図においてAMU（Activity Memory Unit ）はトークンを入力し，命令（アクティビティ）の実行可能検出を行い，実行可能となった命令を合成して出力する機能を有する。EXU（EXecution Unit ）は実行可能命令のうち，演算命令やゲート制御命令のような共有リソースをアクセスしない命令を実行し，その結果から次の目的地へ渡すためのトークンを作成して出力する。IOU（Input／Output control Unit ）は入出力装置を制御する命令を実行する。SMU（Structure Memory Unit ）は配列やリストのような構造データの格納，管理，および操作の機能を有する。D－net，AM－SM net，および Inter SM net はいずれもトークンや命令を転送するネットワークであり，このネットワークの構成に応じていくつかの変形が考えられる。

これらの機能要素のうちIOU，SMU，およびネットワーク構成に関する詳細は4.4節以降で議論する。以下，AMUおよびEXUの詳細構成について述べる。

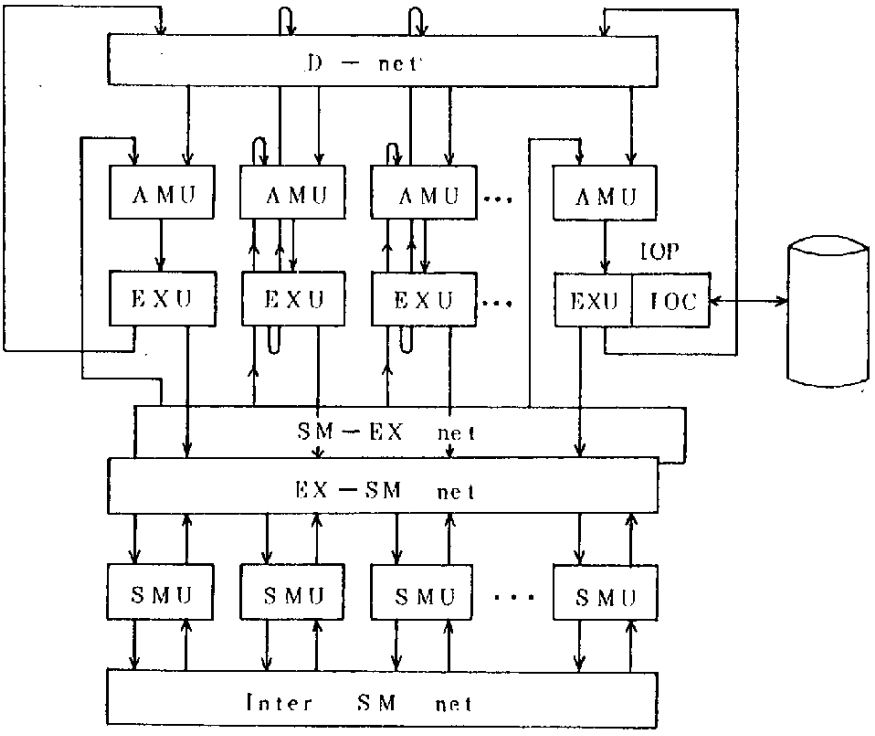
a．AMU（Activity Memory Unit ）

プログラム格納用記憶およびオペランドの一時記憶機能を持ち，命令が実行可能であるか否かを検出し実行可能となった命令を出力する機能を有する。

AMUのハードウェア構成に影響をおよぼすポイントとしては，構造を有するオペランドの表現，命令のオペランド数，定数オペランドの表現，および，結果の宛先命令の指定法がある。以下，これらの面から構成法を論ずる。

(1) 構造を有するオペランドの表現

命令のオペランドとして配列やリストのような構造データを使用するとき，どのように表現するかという問題がある。構造そのものをAMU内にオペランドとして格納する方法は，命令実行毎に構造データをネットワーク内で転送する必要があるためネットワーク負荷が重くなり（従ってネットワークのスループットが低下する恐れがある），AMU内に構造データを動的に格納・管理しなければならないためAMUのハードウェア構造が複雑となりその



AMU : Activity Memory Unit

EXU : Execution Unit

SMU : Structure Memory Unit

IOC : Input／Output Controller

D-net : Distribution Network

SM-EX net, EX-SM net

Inter SM net : 結合ネットワーク

図4.3.4 ＤＦＭ実験機の構成

処理のためのオーバヘッドが大きくなるという問題が生ずる。従って，現状のデータフローマシンにおいては構造を示すポインタを用いる方法が一般的である。この方法では，構造データそのものはSMUのような記憶装置内に格納しておき，そのポインタがAMUにトークンとして渡される。実際の構造データへのアクセスは構造データ操作命令をSMUへ転送してその処理を依頼するという形で行われる。しかし，この方法ではAMUとSMU間の命令や転送の遅延時間が問題になる恐れがあり，ネットワーク構成を含めた性能の評価をシミュレーションによって行う必要がある。

(2) 命令のオペランド数

命令の入力オペランドをいくつまでに設定するかに応じてハードウェア構造は異なる。多入力オペランド命令を許す場合，一般には各命令（アクティビティ）毎にそのオペランドの到着状態を示すオペランドカウンタや複数ビットから成るオペランドタグを設けて，その到着状態を監視する機構を設けている。しかし，この方法はハードウェア構造がやや複雑になるという欠点がある。

これに対して，命令の入力オペランドが1つまたは2つに限定されている場合は，命令が実行可能となったか否かを検出するのは，たかだか到着したオペランドの相手側オペランドがすでにオペランド一時記憶内に到着済みであるか否かを検出するだけでよい。この場合，多入力オペランド命令は2入力オペランド命令の組み合せで実現できる（ただし，オーバヘッドは大きくなると推定される）。

トークンに関数識別子やループ識別子等を付与するとオペランドのアドレス空間が非常に大きくなり，一般にはアドレス変換機構が必要となる（4.3.2を参照）。このアドレス変換に連想機能を有する連想メモリや並列ハッシュ表を用いる方法が考えられるが，命令のオペランド数が2つまでに限定されている場合，この連想検索機能を用いることにより命令の実行可能検出を行うことが可能である。即ち，多入力オペランド命令におけるオペランドカウンタやオペランドタグを用いる方法では連想検索してアドレス変換した後，カウンタやタグの内容を調べる操作が必要であるのに対して2オペランド命令の場合は連想検索の結果がそのまま命令の実行可能判定結果となり，高速化できる（連想検索の結果，目的とする相手オペランドがなければ実行可能でないし，そうでなければ実行可能である）。

従って2オペランド命令を実現する方法が有望である。このとき，連想機能を実現する手段として連想メモリ方式は検索時間の均質性が得られるという点で理想的であるが，ハードウェアコストや実現性の上で問題が残されている。このため実験機第1版では並列ハッシュ方式を用いるのが妥当と思われる。

(3) 定数オペランドの表現

コンパイル時に決定される定数を表現する方法としては，定数生成のための特殊な命令を設ける方法，一般の命令コード中に定数を埋め込む方法，定数を格納するための局所記憶をAMU内に設ける方法，および定数を構造メモリに格納する方法等が考えられる。前2者は命令コード中に直接埋め込むのに対して後2者は命令コード中に定数格納域へのポインタを用意しておくことになろう。いずれにせよ命令のフォーマットをどのようにするかを決定する要因となる。構造を有する定数の場合は，後2者のいずれかが選定されることになろう。

(4) 結果の宛先の指定法

データフロープログラムにおいては命令の実行結果を送るべき結果の宛先は一般に，命令コード内で指定される。このとき，問題となるのはその結果を複数の宛先に転送したい場合である。一般には，1つの命令で2～3個の宛先を収容できるフィールドを用意しておき，宛先数がこれを超える場合は複数命令を組み合わせて指定するようにしている。この場合，目的とする宛先へ結果を送るために余分な命令を実行する必要があり処理性能に影響をおよぼす。

これに対して，命令で任意個の宛先を指定できるような手段を設ける方法がある。1つの方法は命令長を可変にして任意個の宛先フィールドを指定できるようにする方法である。しかし，この方法はAMUの制御が難しくなるという難点がある。2番目の方法は，多数の宛先指定を局所記憶や構造メモリ内に収容し，命令コード内にそのポインタを保持する方法である。この場合も制御が難しくなることの外，宛先を読み出すためのオーバヘッドが問題となる。3番目の方法は宛先の命令の配置に規則性を与え，1つの宛先指定で複数の目的地アドレスを生成できるようにすることである（例えば，宛先の命令をi番地から連続してn個分置いておき，宛先先頭アドレスiとその個数nが与えられると，自動的にi，i＋1，…，i＋n－1番地の宛先へトークンを送るようにする）。しかし，この方法は命令の配置に制約を与えるという問題が生ずる。最後に，命令コードの格納用に連想メモリを用いる方法がある。即ち，データフローグラフ内のアーク（データ経路）毎にユニークな宛先識別子を割り当て，この識別子で命令コード記憶を連想検索する方法である。この方法の難点は連想メモリが高価であるためハードウェアコストが高くなることである。

b．EXU（EXecution Unit）

AMUから出力された実行可能命令群のうち，入出力制御や構造データアクセス等のようなアクティビティ間で共有されるリソースを必要とする命令以外の命令を実行し，その結果を命令中で指定されている目的地へ送るためにトークンを生成し，出力する。構造データ操作や入

出力制御等のための処理装置は4.4節以降で述べる。ＥＸＵ内で実装される演算装置の数は，各演算装置の処理速度とＡＭＵでの命令あたりの処理速度のバランスを考慮して決定されよう。ＥＸＵ内に複数の演算装置を収容するときは，各演算装置に命令を分配するためのスイッチや，各演算装置からの結果を集めてＥＸＵからトークンを出力するためのアービタ（Arbitor）が必要となる。

演算装置の実現法としては，汎用マイクロプロセッサを用いる方法，ビットスライスマイクロプロセッサを用いる方法，専用演算器と専用回路を設ける方法，およびカスタムＶＬＳＩを使用する方法等が考えられる。柔軟性，実現性，および処理能力等を考慮して実験機第１版の演算装置としては，書き込み可能制御記憶により制御されるビットスライスマイクロプロセッサを使用するのが適当と思われる。実験機第２版以降はこの評価をもとにカスタムＶＬＳＩ化する方向となろう。

システム内の全演算装置を汎用化して負荷分散形態とするか，演算装置の一部または全部を専用機能ユニット化して機能分散形態とするか，またはＡＭＵ－ＥＸＵ間の結合に何らかの局所性を導入するかは，システムスループットやハードウェアコスト等を総合評価して決定する必要がある。

B．リダクションマシン

リダクションマシンは実行可能な関数適用（reducible application）をその関数の定義（function definition）によって置換していくことによって計算を行う。このリダクションの機構には以下の２つの基本タイプがある。

- ストリングリダクション
- グラフリダクション

ストリングリダクションの機構は，実行可能な関数適用がプログラム内に存在するときその適用自体を関数の定義によって置換することを基本にしている。従って，関数適用があるとプログラム自身が，それと等価な形に変形される。この機構を用いているマシンの例としては，K. J. Berkling のスタックマシン，[10] G. A. Magó の木構造マシン，[11] P. C. Treleaven 等のマシン，[12] および小長谷等のマシン，[13] が提案されている。

これに対して，グラフリダクションの機構は関数の適用と関数の定義間で引数を共有することを基礎にしている。即ち，実行可能な関数適用が存在したとき，その引数はポインタを介して関数定義側に渡される。この機構を用いた例としてはR. M. Keller等のＡＭＰＳマシン 14）がある。

リダクションマシンとしては上記のようにいくつかの提案がなされているが，そのハードウェ

ア構成はそれぞれによってかなり異なり，データフローマシンのように統一的に議論するのは困難である。従って，ここでは上記具体例についてその特徴を述べるにとどめることにする。

Berkling のマシンはスタックマシンであるので並列処理を実現するのは本質的に困難であるが，単純なハードウェアによってラムダ計算にもとづくリダクションマシンを実現した点で注目されている。これ以外のリダクションマシンはセルラー構造またはマルチプロセッサ構造をなしており，並列処理を実現している。

Mogo′ のマシンは記憶機能を有するL（ Leaf ）セルと通信および計算機能を有するT(Tree)セルから構成される木構造セルラーマシンである。各Lセルはたかだか1つのシンボルしか記憶しないためその構造は単純であり，２０個程度のレジスタ，ＣＰＵ，および書き込み可能なマイクロプログラム記憶からなる。各Tセルは，数個のレジスタ，4つのプロセッサ，およびこれらと外部とを結合する通信チャネルからなる。各Lセルは独立に動作するが，マシン全体としてはそれらが協調してシンボルを木構造の上下間で渡しながら置換の処理を行う。実用的なマシンを構成するためには 10^5 個程度以上のセルを必要とするため，System on wafer のような高度のＶＬＳＩ技術を必要としよう。

Treleaven のマシンは双方向キューとＰＵ（ Processing Unit ）を交互に連結したリング構造マシンである。プログラムはキュー内に分割してロードされ，実行可能な関数適用の検出は，ＰＵがキュー内のシンボルを順に取り出すことによって行われる。ＰＵは，いくつかのレジスタやバッファ，リダクションテーブル，アクションユニット，およびオペレーションメモリからなる。リダクションテーブルはＰＵの動作を規定するための状態遷移テーブルであり，このテーブルの指定によってアクションユニットが起動される。アクションユニットは実行可能な関数適用を探索しＰＵ内のバッファに格納するための動作や実際の置換を行ったりする。関数定義は専用のメモリへ置かれ，ＰＵ間で共有される。このマシンの問題点としては，実行可能な関数適用の探索がシーケンシャルに行われるためそのオーバヘッドが大きいことや，デッドロックを避けるためにＰＵ間で相手の処理を中断させる手段が必要なこと，このＰＵ間の干渉のためＰＵ数を増加させることが必ずしも性能の向上に結びつかない恐れがあること，および関数定義メモリへのアクセス競合やそのコピーのオーバヘッドがあることがある。

小長谷等のマシンはリダクショングラフと呼ばれる木構造のプログラムを直接実行する。このリダクショングラフはラムダ式を結合子論理（ combinatory logic ）にもとづいて変形したものである。このマシンでは，リデュスオペレータと呼ばれるラムダ式における変数の結合を制御するオペレータを導入することにより遅延評価機能を実現している。マシンの構成はメモリとＣＰＵを一体化させたリダクションセルを階層的に結合したセルラーアーキテクチャであるが，そ

の詳細は不明である。

Keller 等のマシンはＰＵ（ processing unit ）を葉ノードとし，通信ノードをそれ以外のノードとする木構造をなしている。Mago' のマシンと異なり各ＰＵはセグメント単位（例えば64 KW ）のローカル記憶を持ち，システム全体で単一アドレス空間を形成している。他セグメントのメモリをアクセスしたいＰＵは通信ノードを介して通信を行う。通信ノードはまた，その配下にあるＰＵ負荷の平衡化機能を持つ。即ち，左の部分木と右の部分木のメモリ使用率を一定の値の範囲内に保つようにタスクを分割し，必要に応じて左右の部分木間でタスクの転送を制御する。このマシンの特徴はデマンド駆動モデルにもとづいていることである。従って，遅延評価機能を実現することができる。もう１つの特徴はグラフリダクションの機構を採用していることである。従って，関数呼出し時の引数の受け渡しは固定長のポインタによって実現できる。この意味ではデータフローマシンの一般的構成と似ているといえよう。

－参考文献－

1) J. B Dennis, "The Varieties of Dataflow Computers", Proc. of 1st Intl Conf. on Dist. Comp., Huntaville, Alabama, Aug.1979.

2) D.Comte.N.Hifdi, J.C.Syre, "The Data Driven LAU Multiprocessor System : Results and Perspectives". Proc. of IFIP80, North-Holland publish -ing Company, Oct.1980.

3) Arvind, K.Gostelow, W.Plouffe, "An Asynchronous programming Langage and Computing Maching". Information and Computer Science, University of California, Irvine, TR114a, Dec.1978.

4) R.M. Keller, G.Lindstrom, S.Patil, "A Loosely-coupled Applicative Multi-processing System", Proc.of Nat. Comp. Conf., 1978, pp.861-870

5) 雨宮，長谷川，「データフローマシンと関数型プログラミング」電々公社　武蔵野電気通信研究所，１９８２。

6) C.Hewitt, "A Universal Modular Actor Formalism for Artificial Intelligence", Palo Alto, CA., 1973.

7) E.W.Dijkstra, "Guarded Command, Nondeterminacy and Formal Derivation of Programs", Comm. of ACM, Vol.18, NO.8, Aug.1975

8) C.A.R Hoare, "Communicating Sequential processes", Comm. of ACM,

Vol.21, NO.8, Aug.1978

9) A.J.Catto, J.R.Gurd, "Nondeteministic Dataflow Graphs", Proc. of IFIP Congress 80 North-Holland Publishing Company, Oct.1980

10) K.J.Berkling, "Reduction Languages for Reduction Machines". Proc. 2nd Intl Symp. on Computer Architectuer, April 1975, PP.133-140.

11) G.A.Mago, "A Network of Micro-processors to Execute Reduction Reduction Languages". Int. Journ. of Computer and Information Sciences, Vol.8, NO.6, 1979.

12) P.C.Treleaven, G.F.Mole, "A Multi-processor Reduction Machine for User-defined Reduction Languages". Proc. 7th Intl Symp.on Computer Architecture, 1980, PP.121-130.

13) 小長谷, 山本, 「リダクションマシンの構想について」, 信学技報EC81-33, 1981年10月。

14) R.M.Keller, G.Lindstrom, S.Patil, "A Loosely-coupled Applicative Multi-processing System". AFIPS Conf.Proc. Vol.48, 1979, PP.861-870.

4.4 構造メモリ

構造メモリは，データフローマシンにおける構造データ操作の効率化をはかることをめざす要素技術として重要であり，その位置付け，機能，内部構造を明確にする必要がある。

これまで，海外における研究としてはＭＩＴのDennis のグループ[2]，同じくArvindのグループ[3),4),5)]また国内における研究としては武蔵野通研[6),7),8)] 沖電気[9)]などがあり，それぞれのデータフローマシンにおける機能モジュールとして構造メモリの実現方式を提案している。また，上記以外のデータフローマシン研究グループにおいても，データフローマシンにおける構造データ操作方式についての研究が進められている。[10),11),12)] 現状としては，構造メモリの実現方式について（構造メモリを必要としないデータフローマシンの実現方式を含めて）様々な議論が行われている段階である。

本節では，以下4.4.1～4.4.2において，データフローマシンにおける構造メモリの目的，持つべき機能について述べる。続いて4.4.3では，これまでに提案されている構造メモリの実現方式について述べ，4.4.4において現状技術の分析と残された問題点について明らかにしていく。

4.4.1 データフローマシンにおけるデータ構造操作と構造メモリの目的

従来のノイマン型マシンでは，物理的メモリセルをそのままプログラムにおける記憶場所として用いており，プログラムが記憶セルの管理を行う。これがノイマン型マシンの副作用の原因になっており，逐次実行の強制に結びついている。

一方，データフローマシンではデータフローグラフによって表現されたプログラムを実行し，原理的にメモリセルの概念はない。

C ← A + B

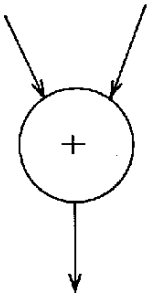


図4.4.1

つまり，図4.4.1において変数A，B，Cは記憶セルの名前ではなくデータトークンが流れるパ

スの名前である。

このようにデータフローマシンでは論理的には（プログラムからは）記憶場所は見えないが，物理的には大量のデータを計算機内部に蓄える機構が必要である。つまり，データフローマシンではデータの蓄え場所として，「関数性を保つ（副作用を持たない）」ようにデータの実体を保持する機構を持つ必要がある。

- データフローマシンの基本モデル

  データフローマシンの基本モデルを図4.4.2に示す。データフローマシンは，アクティビティメモリ（AM），処理装置（P），および両者をむすぶネットワーク（N）からなると考えられる。この場合，データはAM，P，Nに散在するパケット内に保持される。

- データフローマシンにおける構造データの扱い

  データフローマシンでは，各データ構造の選択子，構成子をデータフローグラフのノードとして表現し，構造データをトークンとしてデータフローグラフ中を流す。

  データフローマシンが，プログラムのネットワーク（データフローグラフ）をハードウェアのネットワークに射影するものとすれば，構造データはトークンパケットとしてハードウェアネットワーク上を流れることになる。

  しかしこれではデータ転送量が膨大となる可能性がある。

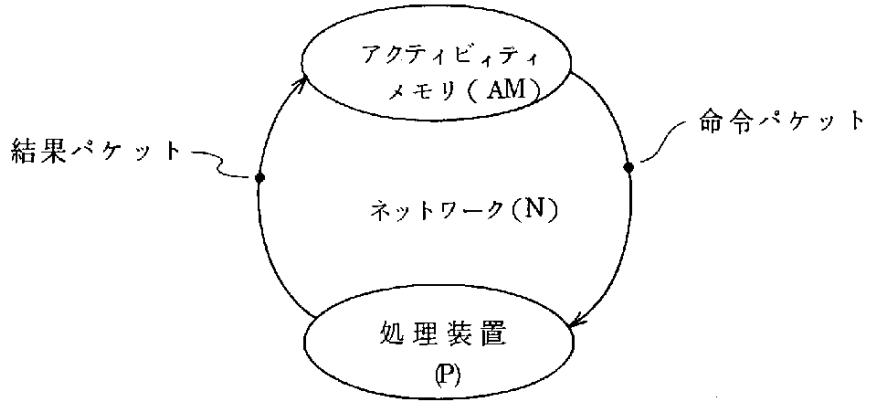


図4.4.2

そこで，より現実的なモデルとして仮の構造データとデータの実体の保持機構（メモリ）を設ける。仮の構造データをオペランドとして持つ命令パケットをメモリに与えることによって構造

データ操作を実行する機構が考えられる。（図4.4.3）

データフローマシンでは，上述のように構造データの実体を保持（メモリ）し，それに対する操作を効率よく実行（制御）する機構が必要であり，これを構造メモリと呼ぶ。

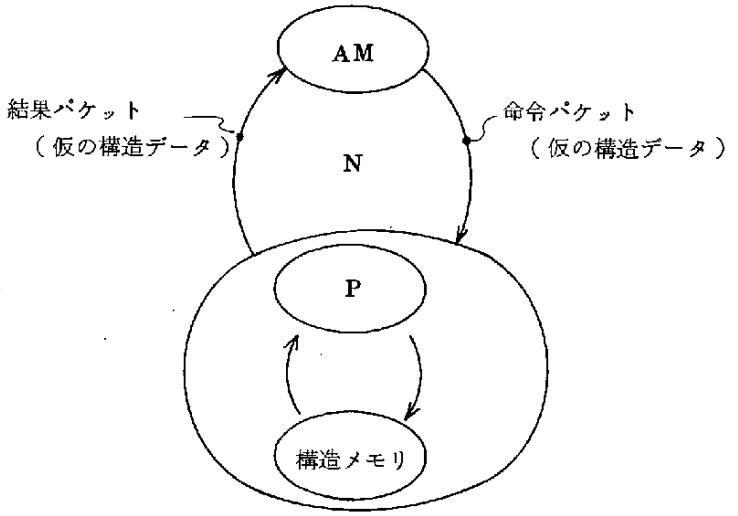


図4.4.3

4.4.2 構造メモリの機能

データフローマシンにおいて構造メモリを設ける目的は，データフローマシンにおける構造データの保持とそれに対する操作の効率化である。

本項ではこのような構造メモリが持つべき機能について考える。

A．構造データ操作の機能

a．データ型と操作

構造メモリでは，プログラムを構成する各種のデータ型，つまり基本データ型（文字，整数，実数，論理値）とそれを要素としてつくられる配列，リスト，レコード等のデータ構造を実現する機構，およびそれに対する操作を高速に実行する機構を提供しなければならない。

(1) 基本データ型

基本データ型としては，文字，整数，実数，論理値さらには文字列等が考えられ，その物理表現はそれぞれ異なる。基本データ型の扱いはそのまま構造メモリのセル構造に影響を与えるため実装時には充分に考慮すべきポイントである。また言語処理系との関連では型チェ

ックの方式が重要である。型チェックを実行時に行うとすれば構造メモリにおける型チェックのハードウェアサポートを検討することになる。

(2) データ構造

従来から用いられてきたデータ構造はプログラム構造つまり応用との結びつきが強い。たとえば記号処理においては動的なリスト構造，木構造が不可欠であり，また科学技術計算の分野では大きな配列構造を扱う必要がある。データフローマシンにおける履歴依存性のある処理（入出力等）では Stream[14]の扱いも必要である。しかし，これまでの専用計算機（Lisp マシン等）を見てわかるようにその効率化の手法は相異なる。

b．第5世代計算機からの要請

現在のところ，知識情報処理における知識データがどのような構造を持つものなのかは明確になっていない。しかし，これまでの人工知能の研究などから知識データにはリスト構造のような柔軟性のある動的データ構造が不可欠であり，その操作の効率化が重要であることがわかっている。

ではリスト構造の効率化のみでよいのであろうか？

リスト構造は万能性を持ち，従来の Lisp システムの多くはリスト構造のみを用いている。しかし，その上で実行される記号処理プログラムでは記号表のような論理的配列構造が頻出しており，これが cdr-coding 等の効率化手法の根拠のひとつにもなっている。

さらにデータフローマシンが推論エンジンへと発展した場合，関係データベースの発展形である知識ベースシステムとの融合が考えられる。この場合，構造メモリが関係データベースマシンとの接点となると考えるのが自然である。

関係データベースマシンではタプル形式のデータの集合に対する関係演算がおこなわれる。このため，構造メモリではタプル形式，つまりレコード構造に対する操作や異種の物理データ構造間での変換操作を効率よく実行する機構が必要である。

B．構造データ操作における関数性の保持

データフローマシンではデータ操作において関数性が保たれる，つまり操作のたびに入力データ（入力トークン）は消費され新たなデータが結果トークンとして生成される。これはデータが構造を持つ場合も同じである。構造の一部を書き換える操作では，変化しない部分も含めて論理的にはすべて新たな構造が結果として生成される。（図 4.4.4 ）

データフローマシンにおけるこのような関数性の要求に応えるには，高速アクセスが可能な無限の容量を持つメモリを使い捨てメモリとして用いる必要がある。しかし現実的にはメモリは有限であり，メモリ上のコピー操作は処理時間，メモリ域の消費の両面で大きなオーバヘッドとなる。

物理的メモリの有効利用は従来の計算機においても重要な課題であったが，データフローマシンではその並列性，関数性によりさらに重要な意味を持つ。

そこで構造メモリを考える場合，メモリの有効利用およびデータ操作の効率化のために有効な

- ガーベッジコレクション機能（Garbage Collection）
- コピーオーバヘッドの解消策

を検討する必要がある。

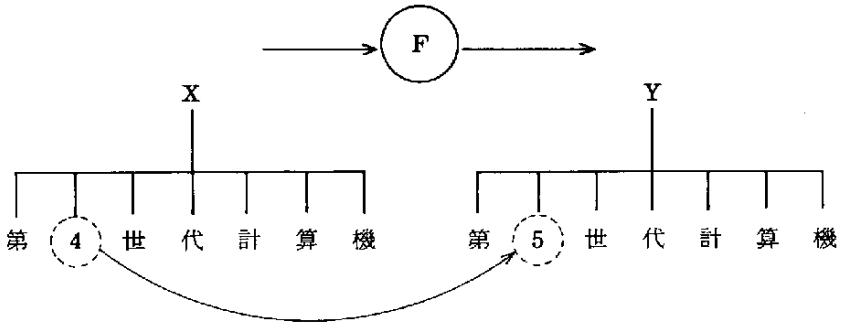


図 4.4.4

C．構造データに対する非同期アクセス機能

a．関数とデータ構造

構造データが2つの関数間で受け渡される場合を考える。（図 4.4.5）

関数f　　B ← f（A）

関数g　　C ← g（B）

A，B，C　は構造データ

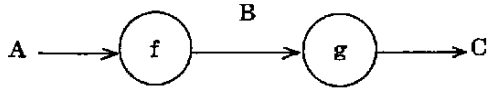


図 4.4.5

データフローマシンにおいて関数gは関数fの出力Bができ上った時に実行可能となる。ただし次のような場合も考えられる。

① 関数gが構造データBの部分構造のみを必要とする。

② 関数gが構造データBの一部を用いて少しずつ実行可能である。

このような場合，関数fが終了する前に関数gを起動することによってパイプライン的並列性を抽出することが可能となる。

b．データ構造における non-strict な操作

MITのR. E. Thomas らの文献[3]を引用してデータ構造における strict と non-strict について説明する。

最初に strict なリストと structure[注)] を次のように定義する。

リストは（）または（value, list ）ペアであり，次の3つの操作がリストに対して定義される。

first (s) : List → Value
if s=( ) then ⊥ else v such that s=(v, s')
where ⊥ represents the undefined value

rest (s) : List → List
if s=( ) then ⊥ else s' such that s=(v, s')

cons (v, s) : Value × List → List
if v≠⊥ or s=⊥ then ⊥ else s' such that *first* (s')=v and *rest* (s')=s

ここで

if V = ⊥ then …

は，「もしVの計算が終了しなければ then …」を意味する。

また structure は<>または（ selector, value ）ペアであり，次の2つの操作が定義される。ここで S−index とは structure S におけるインデックスの集合 {i | (i, v)∈S} である。

注) MITのDennisらは，配列，レコード構造の一般化として次のような構造を特に structure と呼んでいる。

structure とは<空>または（ selector, value ）ペアであり，selector は互いに相違なる。また valueは全ての基本データ型および structureである。

select (s, i) : Structure × Integer → Value
if i∉ s index then ⊥ else s_i
where s_i represents the value v such that (i, v) ∈ s

append (s, i, v) : Structure × Integer × Value → Structure
if s=⊥ or i=⊥ or v=⊥ then ⊥ else s' such that $s'_j = s_j$ j≠i
$s'_j = v$ j=i

ここで strictな関数とは引数が全てそろった時に初めて実行される関数をいう。上記のリストと structureでは，cons と append が strict な関数である。

これに対し streamは次のように non-strict な cons を用いて定義される。first と rest は strict なリストと同じである。

cons (v, s) : Value × Stream → Stream
s' such that *first* (s')=v and *rest* (s')=s

つまり streamと（ strict な ）リストでは，consの定義が異なり，streamのcons は実行時に引数の全てがそろうことを強制しない。

同様に structureのappend についても non-strict な append（ a-append ）を考えることができる。（非同期 structureと呼ぶ。）

a_append (s, i, v) : Structure × Integer × Value → Structure
if s=⊥ or i=⊥ then ⊥ else s' such that $s'_j = s_j$ j≠i
$s'_j = v$ j=i

stream，非同期structureのようなnon-strictな構成子（ non-strict cons，a-append ）を持つデータ構造では，その選択子（ first，rest，select ）はテキスト上 strictなデータ構造とかわりはないが，その振る舞いは大きく異なる。たとえば，first（ list ）は全てのlistの要素がそろうまで実行されないが，first(stream) は streamの最初の要素が確定した時すぐさま実行可能となる。

non-strict なデータ構造は，並列プログラムング，lazy elavuation[注)] において効果的であることがわかっており，データフローマシンにおいても取り込まれるべき機構である。

この機構は関数の実行制御において導入されなければ実現できないことは当然であるが，データ構造を保持する構造メモリ側においてもそのサポートが必要であり，実装方式の良し悪しによって得られる効果も大きく異なる。より具体的に表現すれば，構造データの要素または部分構造に対する非同期アクセス方式の実現である。

4.4.3 構造メモリの現状

A. 概　　要

現在までに提案または試作されたデータフローマシンにおいて構造メモリの構成方式を明確に打ち出しているものは少ないが，大別して次の2つのものになると思われる。

ひとつはAckerman の提案に属するもので，

- 木構造によるデータ構造の表現
- 参照カウントによるデータの部分構造の共有とガーベジコレクション

を基本とするものである。この方式には武蔵野通研のリスト処理向きデータフローマシンが含まれる。

もうひとつは，Arvind の I-structure memory のタイプであり，構造の要素に対する非同期的アクセス機構を中心にしたものである。これには沖電気のD^3Pが含まれる。

本節ではそれぞれのマシンにおける構造メモリの実現方式について述べる。

B. Ackerman の構造メモリ[1),2)]

a. 考 え 方

MITのB. Ackerman が提案した構造メモリについて簡単に説明する。

Ackerman が想定している構造データは

< セレクタ : 値 >

という組が集まったものである。セレクタの値は互いに相異なるものであれば何でもよいが，ここでは整数に限っておく。

セレクタの下にくる値は，整数，実数等の基本的な値，または別の構造データである。

構造データに対しては2つの演算子を作用することができる。1つは選択演算子(select

注）　non-strict なデータ構造を用いた関数の実行を Hasty Eval,[12)] Eager Eval と呼ぶ場合もある。

operator ）であり，もうひとつはアペンド演算子（ append operator ）である。

- 2分木による構造データの表現
- 参照カウントによる共有とガーベジコレクション

にある。

構造データのセレクタは2進数によって表現し，構造データ操作はその2進表現を見ながら各節で分岐する際に次の桁が1なら右へ，0なら左へと進んで2分木を探索していくことによっておこなう。図4.4.6は

B ← append （ A， 1001， 5 ）

C ← append （ B， 1 ，nil ）

を実行する様子を示す。

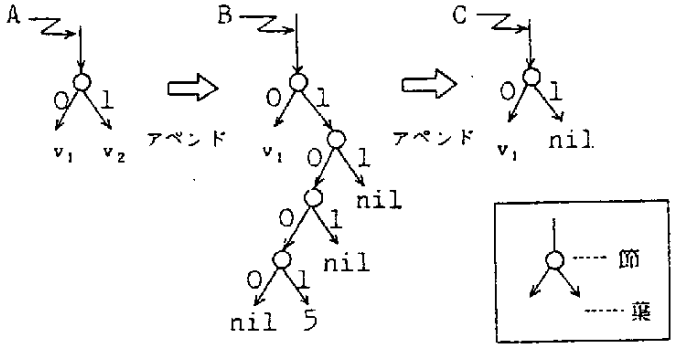


図4.4.6

構造データの各節には，その節を参照しているポインタの数を示す参照カウントが付いている。この参照カウントを用いて，構造データの部分構造の共有とガーベジコレクションがおこなわれる。図4.4.7と図4.4.8にappend 演算における例を示す。

図4.4.7において，ノードⓑから下は，構造データAと構造データBによって共有されている。（ノードⓑの参照カウントは2になっている。）ここで

C ← append （ A， 000， 6 ）

る実行すると，ノードⓑとノードⓒの部分についてコピーが行われ図4.4.8のようになる。つまり，append 演算においては，各ノードの参照カウントをチェックし，他と共有している部分についてコピー操作を行い参照カウントを更新する必要がある。

select 演算，append 演算によって参照カウントがゼロになったノードについてはガーベジコレクションが行われる。2分木構造のガーベジコレクションは基本的に再帰的な操作とな

るが，この再帰性を取り除くためにAckerman[注)]は，一方の枝についてのみガーベジコレクションを行い，その領域を使う時点で残りの枝のガーベジコレクションを行うことを提案している。

ここで述べた参照カウントを用いた方式については4.4.4において再び議論する。

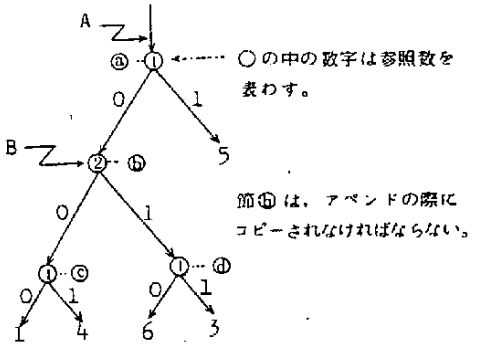


図4.4.7

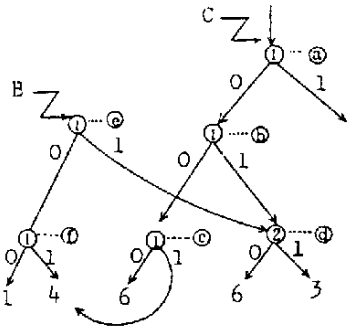


図4.4.8

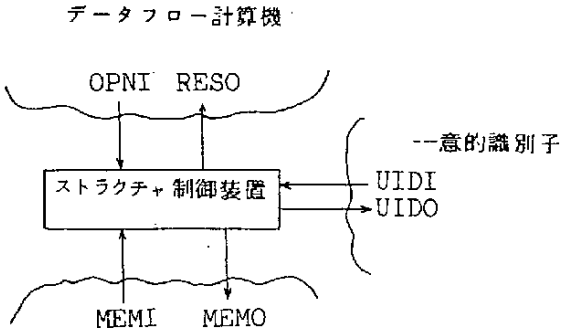
b．ハードウェア構造

Ackermanが考えていた構造メモリのハードウェア構造を図4.4.9，図4.4.10に示す。

ここでストラクチャ制御装置は3つの入出力の組を持つ。ひとつはデータフロー計算機からselect，appendの命令を受け，またその結果を戻すもの。ふたつめは相互連絡網を介して記憶装置（通常のRAMを想定している）を操作するもの。残りのひとつはフリーセルリストをかかえている一意的識別子にアクセスするものである。

一意的識別子は，先に述べたガーベジコレクションによって得られたフリーリストをかかえており，ここにアクセスすることによってフリーセルを得ることができる。

注）Ackermanが提案した構造メモリの詳細は内部レポート 15）16）にある。



OPNI— Select（ストラクチャ，セレクタ，行き先）
Append（ストラクチャ，セレクタ，行き先，アペンドすべきデータ）

RESO—（データ，行き先）

MEMO— Fetch （アドレス，タグ）
Fetch+（アドレス，タグ）
Fetch−（アドレス，タグ）

Update（アドレス，参照数，データ）

MEMI （データ，参照数，タグ）

UIDI—（アドレス，データ）
UIDO—（アドレス，データ）

図 4.4.9

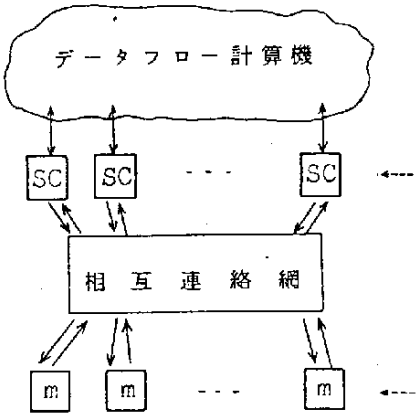


図 4.4.10

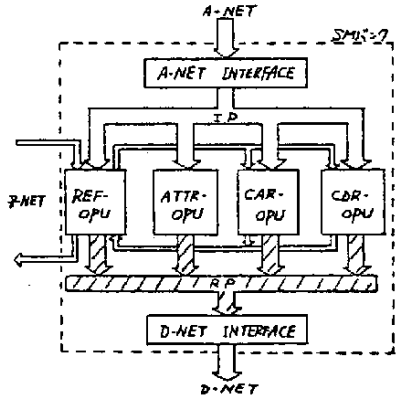


図 4.4.11

C．武蔵野通研のリスト処理向きデータフローマシンの構造メモリ[6),7),8)]

武蔵野通研では，構造メモリとアクティビティメモリとのインタフェースを純Lispの基本関数（car，cdr，cons）レベルに設定したリスト処理向きデータフローマシンを提案しており現在各種の評価が試みられている。（3.2.1 5参照）

武蔵野通研のデータフローマシンの構造メモリの特徴は以下の通りである。

① 構造メモリに対する操作は純Lispの基本関数を対象にしており，さらに，新しいメモリセルを作成する関数consを，セルの確保と値の書き込みの操作に分解するLenient consによってパイプライン的並列性を高めている。

② ガーベジコレクション法として参照カウント法を用いている。しかし参照命令実行後の更新はデータフロープログラムのブロック単位に行ない，ブロック内の個々の命令毎には行わない。[8)]

③ 構造メモリは多バンク構成になっており，さらにひとつひとつのバンクがメモリセルのフィールド対応の処理ユニットにわかれている。

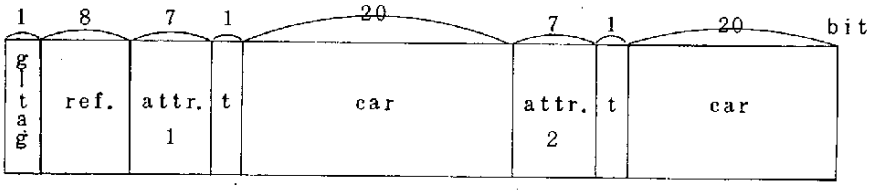
(1) 構造メモリバンクの構成

構造メモリバンクの構成を図4.4.1 1に，またメモリセルの構成と構造メモリで実行する基本関数をそれぞれ図4.4.1 2，図4.4.1 3に示す。

基本関数は処理対象とするメモリフィールドによって4種に分類でき，それぞれの処理ユニットで実行される。

(2) データの配置

cons時のセル獲得の仕方としては，各メモリバンクに対してランダムに行うことによってメモリ参照を分散させる方式をとっている。これはリスト処理における参照の局所性は一般的に少ないという立場をとっているためである。データの配置の問題は，並列性・ネットワークの負荷等に関して重要なポイントである。



ref.：reference count，attr.：attribute，t：ready tag

図4.4.12

( ):Closed in SM

OPU	関　　数	主　な　機　能
REF	REF-UP	参照カウントをカウントアップする
	REF-DOWN	参照カウントをカウントダウンする
	GET-CELL	空きセルを確保する
ATTR	ATOM	ＡＴＯＭセルか否かを識別する
	EQ	同一リストか否かを識別する
	NULL	ＮＩＬか否かを識別する
CAR	WRITE-CAR	car 部にデータを書き込む
	CAR	car 部からデータを読み出す
	(CAR-G)	car 部が指すセルに対しREF-DOWNを発行する
CDR	WRITE-CDR	cdr 部にデータを書き込む
	CDR	cdr 部からデータを読み出す
	(CDR-G)	cdr 部が指すセルに対しREF-DOWNを発行する

図4.4.13　基　本　関　数

c．ガーベジコレクション

各セルの参照カウントは get-cell 命令の実行時に設定され，そのカウントダウンはRef-down 命令によって陽に行われる。カウントダウンの結果，参照カウントがゼロになったメモリセル（ガーベジセル）のアドレスは，Ref-opu（図4.4.11）内のアドレスバッファに空がある場合はそのバッファ内に格納され，バッファが一杯の時は各セルに付いているガーベジタグをセットすることによってガーベジセルであることが明示される。

d．関数の部分実行とLenient Cons

武蔵野通研のデータフローマシンでは，Lenient Cons と呼ぶ構造データに対する非同期アクセス機構を導入することによって，関数の部分実行を実現している。ここではLenient Cons の考え方について述べる。

まず関数の部分的実行を行う理由は次の２点にあるという。

① 関数の実行においては，関数の管理，マシンへの割り付け，等のオーバヘッドが存在する。部分的実行により，引数がすべてそろう間を使ってオーバヘッド処理を進めておくことができる。

② 関数によっては引数の一部を用いて処理を進めておくことが可能であるものがある。

関数が複数の引数を有する時は，関数の起動側において部分的実行が可能である。しかし，ある関数が動的にリストデータを生成しもう一方の関数がそれを入力引数として用いる場合は，前者がリストを生成し終わるまで後者を起動することはできない。これを解決するために導入したのがLenient Consである。

Lenient Consは4.4.2Cで述べたnon-strictなconsに相当するものであり次のように実現される。

Cons（ x, y ）は図4.4.14のように4つの基本ノード（ get-cell, write car, write cdr, gate ）に分解され，またひとつのデータセルは図4.4.15に示す3フィールドの他に3個のタグ（ garbage tag, car ready tag ）が設けられている。

Lenient Consの実行の様子をcons結果に対してcar操作を施こす場合を例にとって説明する。

get cellノードにオペランドsignal（ consを含む関数あるいはブロックが起動された信号）が到着すると未使用のデータセルを確保し

$$\left.\begin{array}{l}\text{car ready tag}\\ \text{cdr ready tag}\\ \text{garbage tag}\end{array}\right\} \leftarrow 0$$

としてそのアドレスzをconsを待つノード（ carノード）およびwrite car, write cdrノードに伝える。write carノードは2つのオペランド（ アドレスzとx ）がそろうと実行され，car フィールドにxを書き込むと共に

car ready tag ← 1

とする。

consの結果として先に伝えられたアドレスzはその先のノード（ この場合はcar ）によって参照される。car命令はcar ready tagによって抑止されるが，write carがおこなわれるとcdr部にデータが書き込まれることを待つことなく実行される。

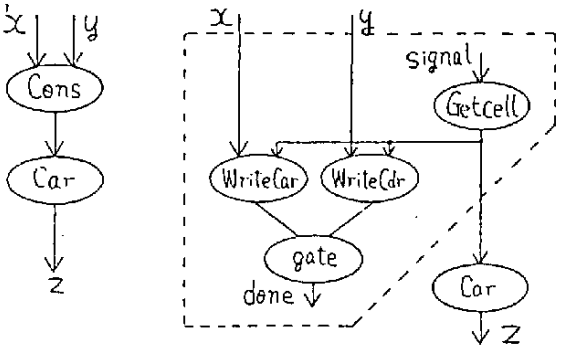


図 4.4.14

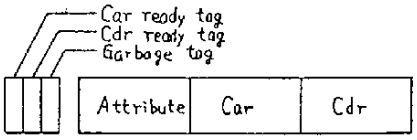


図 4.4.15

D. Arvind らの I-structure メモリ[3),4),5)]

ＭＩＴのArvind らは科学技術計算用スーパーコンピュータを目指したデータフローマシンを開発中である（ 3.2.2 参照）

Arvind らはそのマシンにおいてｉ-structure メモリを採用している。I-structure メモリとは単調に生成，消費されるＩ-structure という配列状のデータ構造のための簡易型構造メモリであり，構造データの要素に対する非同期アクセスに特徴を持つ。

a． I-structure の考え方

Arvind らは，「データフローマシンにおける構造メモリの構成を考える際に，構造データ操作における過度な汎用性は必要なく，ある制限された構造データ操作によって充分に置き換えることが可能である」という考えにもとづいてＩ-structure というデータ構造を提案している。I-structure はその生成に制限を付けた非同期structure（ 4.4.2 C b 参照）である。I-structure を定義する前に，structure における partial order を次のように定義しておく。

For structures x and y, $x \cdot\geq y$ if and only if

$$(y\_index \subseteq x\_index) \text{ and}$$
$$[i \in y\_index \Rightarrow (x_i = y_i \text{ and } x_i \text{ and } y_i \text{ are not structures})$$
$$\text{or } (x_i \cdot\geq y_i \text{ and } x_i \text{ and } y_i \text{ are structures})]$$

Similarly $x \cdot> y$ if and only if

$$(x \cdot\geq y) \text{ and } [(y\_index \subset x\_index) \text{ or}$$
$$(x\_index = y\_index \text{ and there exists an } i$$
$$\text{such that } i \in x\_index \text{ and } x_i \cdot> y_i)]$$

I-structure とは partial order ($\cdot>$) について単調に生成される structure である。

b. I-structure producer

次のような言語構造を I-structure producer と呼んでいる。非同期 structure X を n ステップで生成する時，X^i (i 番めのステップまでで生成された structure) に関して $X^i \cdot> X^{i-1}$ が $1 \leq i \leq n$ において成り立つものである。つまり，I-structure のセレクタにある要素を append した場合，2度と同じ場所に別の値を append しないことを意味する。ただし，X の i 番めの要素自体が I-structure (s) である場合，新たな I-structure v について $v \cdot> s$ であれば s を v に置き換えることはできる。

以上のように I-structure producer を定義することによって，I-structure は順序に関係なく並列に要素を生成し，非同期的にその要素をセレクトすることができるようになる。

c. Structure の単調な消費

Structure の要素に対して1度だけ参照する言語構造を structure の単調 consumer と呼ぶ。この場合，select 操作は読み捨ての操作であり，もし2度以上読む場合は別のコピーを用意することになる。

Arvind らのデータフロー言語 Id は単一代入言語であるため，右辺に複数回あらわれる変数名はその数だけのコピーを用意することになる。(ただし実装時に参照カウントを設けることも考えられる。→d)

structureの生成，消費において，I-structure として生成されることによってプログラムの並列性は増加するが，単調 consumer 自体は並列性には寄与しない。しかし，値を読み終えるとすぐさまメモリを開放できるためメモリ域の節約には大きく貢献する。

d．I-structure の宣言と検出

I-structure を言語に採用する場合，次の2つの方式が考えられる。ひとつはプログラマに宣言させる方法である。この方法の欠点は，たとえ実行時チェックを行っても I-structure に関するすべての演算が終了するまで誤使用が検出できない点にある。

もうひとつの方式は，コンパイラが自動的に I-structure を検出するものである。この方式の難しさは，言語において

X (exp) ← …

（ここで exp は任意の整数表現）

が許されている場合，exp が繰り返しのない selector を表わすという保証をしにくいことにある。

しかしArvind らの考えているマシンは主として科学技術計算を志向しており，structure のインデクス表現は単純であると仮定しているため，コンパイラによる I-structure producer のチェックはそれほど問題にならない。ただし，プログラマが I-structure とそのコンパイラに関する充分な知識がないと I-structure の利点が生かしきれないという欠点は残る。

e．I-structure の実装（ I-structure memory ）

次に，大きさがわかっている I-structure を実行時につくる場合について，その実装法を示す。

I-structure memory の各メモリセルは presence bit と呼ぶ1 bit の tag を持つ。presence bit が1である時はセルにデータがあることを示し，読み出し要求が受け付けられ，presence bit が0である時は書き込み要求が受け付けられる。

presence bit が0（セルが空）の時に読み出し要求が来た場合，当セルからはじまる deffered read request list と呼ばれるデータ到着待ちリストをつくってデータの書き込みを待つ。

I-structure は1セル分の簡単なdescriptor X°によって表わす。I-structure producer が呼ばれるとその大きさのブロックが割り当てられ，descriptor X° が I-structure の生成に関するすべてのオペレータに渡される。

メモリ管理の方法としては次の2つの方法がある。

ひとつは読み出し操作時にpresence bit に手を加えない方式である。この場合，descriptor X°に参照カウント（ 4.4.4参照）を設けることによって物理的コピーを避けることができる。ただし，あるstructure に対するセレクト操作をすべて終了した時に参照カウントを減ずる命令をコンパイラが生成しておく必要があり，またstructure が占めるメモリ領域は参照がすべて終了した時にはじめて開放される。

もうひとつの方式は，読み出し操作時にpresence bit をゼロクリアするものである。この方式は前者に比してハードウェアが複雑になるが，I-structure 全体が消費される前にメモリ領域を再使用することが可能となる。ただし，presence bit が1である時に書き込み要求が生じる場合があるため，読み出し要求の先行の場合と同様にその要求を待たせる機構が必要である。この方式は2つの難点がある。ひとつはすべてのセルに参照カウントを設けないと物理的コピーが避けられないこと。もうひとつは，複数の書き込み要求が衝突する可能性があることである。これは非同期システムにおいて充分あり得ることであり，結果が正しいものでなくなる可能性がある。

E．沖電気D^3Pの構造メモリ[9)]

沖電気のD^3P（ Distributed Data Driven Processor ）では，構造データの非同期アクセス機構を中心においた構造メモリ（ SDM ）を採用している。

基本的な考え方はArvind らのI-structure メモリと同様であるため，ここではそのハードウェア構成を中心に述べる。

D^3Pの構造メモリ（ SDM ）は構造データの要素への非同期アクセスを実現するために図4.4.16のような構成になっている。各メモリセルには，その内容が有効であるかどうか示すW（ Write ）ビット，およびデータの読み出し要求待ちがあるかどうかを示すR（ Read ）ビットが用意されている。Wビットがオンの場合は対応する語に値が格納されていることを示し，Rビットがオンの場合は対応する語にWQT（ Wait Queue Table ） のエントリアドレスが格納されていることを示す。WQTの各エントリには，要素を転送すべき目的地および次のWQTエントリへのポインタが格納される。

構造データへの要素の書き込みの場合，対応する語のRビットが調べられる。このときRビットがオンであれば，WQTのポインタをたどり，待ちとなっているエントリ中の目的地からトークンを作成する。Rビットがリセットされ，実際の書き込み動作が行われてWビットがオンにされる。

構造データからの読み出しの場合，対応する語のWビットが調べられる。これがオンのときは語の内容によりトークンを作成する。オンでないときは，WQTチェインに待ちエントリを追加する。

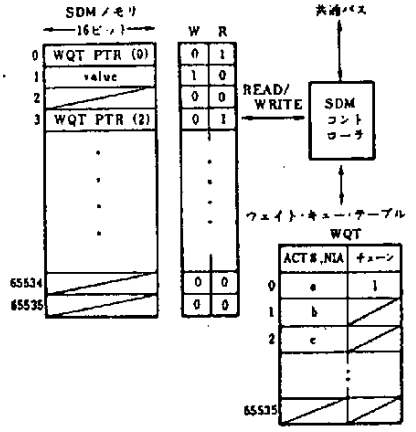


図 4.4.16

4.4.4 構造メモリの実現方式の検討

構造メモリはデータフローマシンにおける構造データ操作を効率化するためのものであり，その機能は

- 構造データ操作の機能
- 関数性を保つ機能
- 非同期アクセスの機能

にまとめられる。

本節では，これらの機能をハードウェア機構として実現する方式について検討する。

前節で述べたように，現在いくつかの実現方式が提案されてきたが，構造メモリについての課題は依然として多数残されている。本節では以下のポイントに重点を置いて議論する。

- 木構造・リスト構造と参照カウント
- non-strict なデータ構造
- データフローマシンにおける構造メモリの位置付け
- 構造メモリの高並列化
- 構造メモリの階層化

A．木構造・リスト構造と参照カウント

木構造またはリスト構造の効率のよい実現にはガーベッジコレクション機能を必要とするため，

従来のノイマン型マシンにおいても大きな問題であった。データフローマシンの環境下では，構造操作における関数性の保持と並列アクセスにより問題はさらに難しくなる。

木構造操作において逐一新たなコピーを生成するには recursive な木探索が必要であり，ワークエリアとしてのスタックも必要となる。このため従来の Lisp では rplaca のような直接的なメモリ操作を導入して効率化をはかってきた。

a．参照カウント

参照カウント（Reference Count，RC ）の考え方は比較的以前からあった。[13] しかし当時のマシンは逐次型であり

① 各メモリセルに余分なフィールドが必要である

② カウント数の更新のオーバヘッドが大きい

③ 巡環リストが実現できない

などの理由から一般的には用いられることは少なかった。しかしデータフローマシンの環境下では

① メモリ価格の低下

② 更新専用のコントローラを置くことが可能

③ 関数性のある操作では循環リストをつくる危険性はない

等の理由により有望な方式となる。

木構造，リスト構造における参照カウント方式では，各セルに参照カウンタ（ RC ）が設けられそのセルを参照しているポインタ数を値（ RC値 ）として持つ。ミクロレベルの操作としては，セルの Read／Write 操作のほかに以下のような参照カウンタの更新操作が必要である。

- RC increment
- RC decrement
- RC zero test

木構造操作において構造の各セルの参照カウントは基本的にそのセルに対するポインタが生成または消滅する度に更新される。Ackerman の例（ 図 4.4.7 ）においては，ノート ⓑ の RC値が decrement され，ノート ⓓ の RC値が increment されている。このため参照カウント方式を有効に行うには，参照カウントの更新操作用のコントローラが必要となる。たとえば武蔵野通研のデータフローマシンでは構造メモリバンク毎に専用のコントローラを設け，さらにデータの Read／Write 操作と独立に更新を実行可能とすることによって効率化を計っている。[6]

参照カウント方式では，

- 構造データの一部のみを別の値に置き換える操作において，変化する部分のみを新たに生成することによりコピーのオーバヘッドが削減できる。
- 繰り返して用いる構造データを逐一コピーする必要がない。

などの利点があり，関数性の保持とメモリ域の有効利用という２つの要求を満足させることができる。

しかし，実際には参照カウンタの更新操作が大きなオーバヘッドとなることも指摘されている。[8)]

b．参照カウントとガーベジコレクション

参照カウント方式において問題となるのは，その操作（特にRCの decrement ）によってカウント数がゼロ，つまりそのセルがガーベッジになった時，セルが指す下位のセルに対して参照カウント数の decrement 操作が伝搬することにある。

図4.4.17の例を考えてみる。Aという構造データがあり，そのcdr操作を行つた結果をBとする。まずAが参照されるとノードaのRC値は decrement され，RC値がゼロになる。これにより，ノードbとノードcのRC値をdecrement する操作の要求が生じる。もしノードb以下のノードのRC値がすべて1であったとすると，各ノードのRC値についてdecrement 操作が次々と伝搬されていくことになる。一方，参照カウントの更新操作とデータの参照操作が非同期に実行される状況では，図4.4.17の中央のようにノード のRC値が一度 increment されてから，同図右のようにdecrement （ノードaのRC値がゼロになったことによる）されることもあり得る。後者の場合は完全なオーバヘッドであり，構造が複数パクに分散しているような場合はネットワークの負担を重くする原因となる。

各セルの参照カウント数の平均が1より大きく，更新操作の下位への伝搬がそれほど頻繁に起こらない場合はよい。しかし，従来のノイマン型マシン上の Lisp システムにおいてセルに対する参照回数が多くの場合１回きりであったというデータもある。[17),18)]

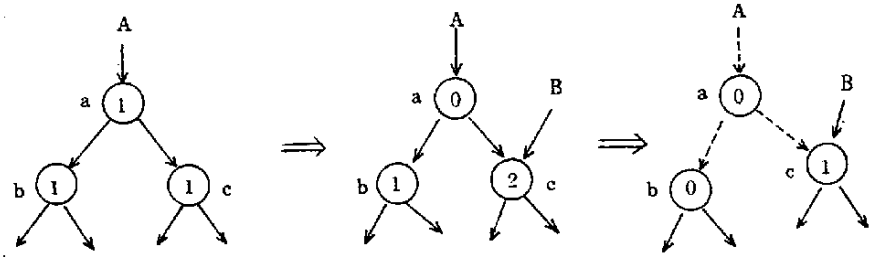


図4.4.17

c．参照カウント方式の効率化

参照カウント方式は，並列処理環境下における木構造・リスト構造のデータにおける部分構造の共有・ガーベジコレクション方式として有望であるが，構造メモリ中に実現するにあたって参照カウント方式自体の効率化を検討する必要もある。以下に参照カウント方式の効率化のアプローチについて述べる。

(1) コンパイラによる更新操作のスケジュール

武蔵野通研のデータフローマシンの初期モデルではメモリオペレーション毎に参照カウントの更新を行うアーキテクチャを採用した。しかし，シミュレーションの結果，参照カウントの更新が

- 基本関数の実行
- T／Fスイッチ
- 関数間のリンケージ

等で頻繁に起こり大きなオーバヘッドとなることが確認された。8）このため武蔵野通研では更新のオーバヘッドの削減のために高級言語 Valid （ 3.1.2 ）の特徴

- ブロック構造と Valid 名の局所性
- 単一代入規則

を利用して，コンパイル時にブロック毎の参照カウント更新操作を explicit に埋め込む方式に変更している。

武蔵野通研の評価結果[8]によると，更新操作は２０％～５０％に減少するという報告が出ている。

このような外部からの更新操作のスケジュール方式においては，次の点に注意すべきである。

- RCの更新操作のスケジュールがプログラム言語（武蔵野通研の場合はValid）の言語構造に依存している。
- プログラム言語・プログラムによっては，コンパイル時のスケジュールが難しくなるか，またはコンパイラの負担が大きくなる可能性がある。
- 構造メモリに与えられる命令としてRC更新命令を用意する必要があるため，必然的に構造データ操作命令は低レベルなものとなる。

(2) 構造メモリに対する命令の高レベル化

構造メモリに対する命令のレベルを上げることによってRCの更新操作を減らすことができる。これは図4.4.17のようなオーバヘッドが出ないように構造メモリ側でスケジュール

が可能であることによる。

ただし，一般的には命令を低レベルにした方が並列性の抽出，負荷分散に有利であるため，並列性とそれを抽出するためのオーバヘッドの間のトレードオフということになろう。

(3) そ の 他

RC操作における細かいレベルの効率化としては次のようなものがある。たとえば，リスト構造におけるappend 操作では，関数性を保つために構造データのコピーを作るのが基本である。しかし，リストを辿っていく際に途中のノードの参照数がすべて1（つまり他の場所から参照されていない）の時は，通常の Lisp における rplaca，rplacd のような直接的なメモリ操作を行っても関数性は保たれる。

このような操作を可能にするには，リストを辿る際のRC値の確認とRCの値によって新セルを確保するかどうかといった判断が必要になる。

d．参照カウント法における今後の課題

メモリセルに対し参照カウントを設ける方式は，構造データの（部分構造の）共有とガーベジコレクションという2つの意味を持つ。ただし，現時点では，〝応用プログラムにおける構造データの共有がどの程度生じるか〟について明確な指針がないため，前者の意味については更に検討する必要がある。もし共有を行わず，ガーベジコレクションのみを考える場合は，有効ビット（1ビット）のセット／リセットで充分となる。今後，参照カウント方式については，

- 構造データの共有の度合
- 効率的なハードウェア機構

等の検討が必要であると同時に，コピー操作自体の効率化も考える必要がある。

B．non-strict なデータ構造

構造データの生成に非同期性を導入した non-strict なデータ構造はパイプライン的並列性を持たらすことができる。この機構を構造メモリに取り入れる動きは比較的活発である。

4.4.3で述べた I-structure memoryは，構造データの生成に〝単調増加〟という制約を加えることによって，non-strict なデータ構造（ I-structure ） をシンプルな構造メモリ上に実装している。I-structure memory では，

- I-structure をプログラム中から抽出する方法
- メモリ領域の管理方法

等に問題があるが，Arvind らのデータフローマシンでは科学技術計算を志向しているため，問題の解決が容易になっている。Arvind らが考えている偏微分方程式のプログラムでは，構造データ操作自体は比較的単純であり，また構造データの配置もコンパイル時にスケジュールするこ

とが可能である。しかし，記号処理等の応用では構造データ操作自体が複雑であり，構造データの性質をコンパイル時に把握することは難しい。

このため，推論マシンを目指すデータフローマシンにおいてArvind らのI-structure memory をそのまま導入すること難しいと思われる。

一方，武蔵野通研の提案したリスト処理における Lenient Cons (non-strict cons) は，始めからリストという動的データ構造を対象として非同期性を導入したという点で注目できる。武蔵野通研が行ったシミュレーション評価 7)では，Lenient Cons に関して次のような結果が報告されている。

① Lenient consの導入により約３０％の実行速度の向上

② 走行関数の数の大幅減少

③ 実行命令数の均一化

②は，cons 結果を返す関数を再帰的に用いた場合，Lenient Cons によって見かけ上の結果が返されることにより，内側の関数が実行中であっても外側の関数が先に終了して畳まれるためであると考えられている。Lenient cons による上述のような効果は大変好ましいものであるが，Lenient Cons にも次のような問題がある。ひとつは構造メモリのハードウェア構造の複雑さであり，もうひとつは命令数の増加である。特に後者は，データフローマシンが大規模になり構造メモリバンクが高並列化された際，ネットワークに対する大きな負荷となる危険性がある。このため，Lenient cons のような機構を導入する場合は通信の問題を充分検討する必要がある。また，実用的な応用プログラムにおいて Lenient cons によるパイプライン効果によりどの程度処理速度が向上するかについて，さらに評価を行う必要があると思われる。

C．データフローマシンにおける構造メモリの位置付け

データフローマシンの基本モデル（ 4.4.2 参照）ではその要素として activity memory, 処理装置，ネットワークを考えた。構造メモリが，これらの主要素とどのような関係にあるべきかが実装におけるポイントのひとつである。これには次のような場合が考えられる。

(1) 処理装置に付随した高機能メモリとしての構造メモリ

アクティビティメモリから命令パケットを受けとった処理装置がその命令に従い構造メモリを操作しながら処理を進める形式である。この場合，構造メモリは物理的に処理装置の近くに置かれ，構造メモリと処理装置間のインタフェースは比較的低レベルに設定される。構造メモリは処理装置に対して受動的立場である。構造メモリのハードウェア構造は比較的簡単であり，アクティビティメモリとの間で交換される命令パケット，結果パケットに対して直接には操作をを行わない。

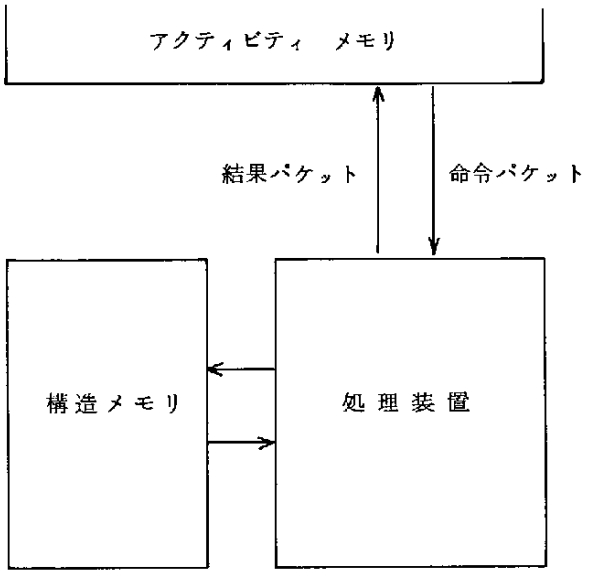


図4.4.18

(2)　構造データ操作用の処理装置としての構造メモリ

これはアクティビティメモリから発せられる命令パケットを受ける処理装置を，構造データ操作用の処理装置（構造メモリ）とそれ以外の処理装置とに機能分割した形式である。

構造メモリは命令パケットのうち構造データ操作命令を受けとり結果パケットを送り出す機構を持つ。このためそのハードウェア構造はデータフローマシンの相当な部分を占める複雑なものとなろう。しかし，アクティビティメモリや他の処理装置との関係は比較的柔軟であり，命令のレベルによって様々な形に変形できると思われる。

上述のような構造メモリの位置付けは，当然のことであるが構造メモリの機能に深く依存する。たとえばメモリ域の有効利用のために必要なガーベジコレクションとデータの共有機能を，動的に実行するか否かによって次のようになる。扱うデータ構造がリスト構造のように動的構造の場合は，構造メモリが自立的にガーベジコレクションを実行する方式が必要である。なぜならば，リスト構造では，セルがメモリ中に散在するためガーベジコレクションに必要な操作数が多く，構造メモリの外部から操作を行うとするとネットワークの負担が増えるためである。このような場合は(2)の位置付けになりやすい。

一方，実行前に構造データの動的性質が明らかになり易い配列構造などの場合は，コンパイル時にデータのallocation操作を命令としてプログラム中に埋め込むことが効果的となる

可能性がある。このため構造メモリに対する操作は read／write が主になり，(1)のようなシンプルな構造メモリになり得る。Arvind らの I-structure memory はこのタイプのものである。

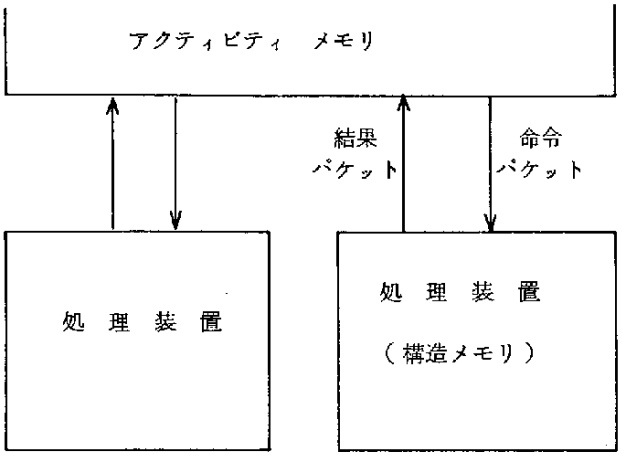


図 4.4.19

D．構造メモリの高並列化

データフローマシンを開発する上でのポイントのひとつに，ノイマン型マシンのボトルネックの解消がある。データフローマシンでは，アクティビティメモリにおいて並列性が抽出され，処理装置さらには構造メモリにおいてその並列が充分に消化されねばならない。このため構造メモリは充分に並列動作が可能な多バンク構成であることが必要である。

構造メモリを多バンク（高並列化）にすることによって生じる問題は次のとおりである。

① 構造データの構造メモリバンクへどのように割り付けるか？

② 構造メモリのバンク間を結合するのに必要なネットワークは何か？

③ また，そのネットワークに対する要求は何か？

武蔵野通研のデータフローマシンにおける構造メモリは多バンク構成になっており，A-net，D-net を介して Control Module（CM）と接続されている。（3.2.1 5参照）D-net，A-net にはオメガネットワークを用いており，各CMと構造メモリバンク間は等距離である。また，cons 時のデータ配置は A-net によって各メモリバンクに対し均等に行われ，データフローマシンにおいて構造メモリはグローバルメモリになっている。これは先に述べたようにリスト処理における参照の局所性は少ないという考えにもとづくためである。

一般に，記号処理の応用では構造データ操作が中心であり，またマルチプロセッサシステムに

おける最大のネックがネットワークにあることを考えると，処理すべき構造データを処理装置の空間的近傍に置くことが望ましい。これには並列性を活かしたまま構造データの生成や参照に局所性を持たせることが必要となる。

参照の局所性については，電総研の山口らによってLazy Eval との関係について検討されている。[10] Lazy Eval では関数を評価した値として，データの実体でなく，関数の実体と環境からなる Recipe が生成される。Recipe で表わされた構造データは必要となった時に値が求められる。これによって以下の利点があるといわれている。

① 不必要な評価が行われないため，無駄なデータが生成されない

② Recipe による構造データの表現がコンパクトな場合は，データの受け渡しによる通信オーバヘッドを減少でき，データの局所性が増大する。

参照の局所性の有無については，今後応用プログラムを充分検討して明らかにしていかなければならない問題である。

E．構造メモリの階層化

a．階層化の必要性

従来のマシンでは，メモリの階層化，仮想化によってメモリアクセスの高速化，メモリ容量の大容量化の両方の要求を満たすことに成功した。

ではデータフローマシンの環境下で構造メモリの階層化をどのようにとらえるべきであろうか？

まず構造メモリと２次記憶（磁気ディスク等）の関係について考えてみる。構造メモリにおける２次記憶の意味としては次の２つがある。

① 構造メモリにおけるメモリ空間の拡大（従来の仮想記憶）

② 構造メモリのファイル機能

b．構造メモリの階層化

従来のマシンにおけるメモリの階層化は，本質的にメモリデバイスの物理的性質つまりアクセスタイムや記憶密度に起因するものであった。

一方，データフローマシンのような高並列マシンでは実装上の前提としてＶＬＳＩ技術を仮定しており，実装に用いる半導体デバイスの種類は少ないと思われる。たとえば処理要素も構造メモリもＭＯＳデバイスによってつくられる形態である。また，構造メモリ自体が何百何千という数の並列メモリバンクによって実装される状況では全体のメモリは相当の容量におよぶであろう。このような環境下において構造メモリの階層化が必要であるかどうかが問題である。

必要となる可能性として次のものが考えられる。ひとつは処理要素との関係である。仮に処

理要素と構造メモリモジュールの結合がゆるやかな場合，処理要素の近傍に構造メモリの cache が必要となる可能性である。

もうひとつの可能性としては構造メモリの容量不足がある場合である。構造メモリは並列化されるが処理自体も並列に実行されるためメモリの容量的不足があり得るという見方である。

構造メモリの階層化の必要性は，構造メモリが，メモリ域の有効利用をどこまで実現するか，また処理プログラムが本質的に必要とする領域の大きさはどの位かに依存するため，すぐさま結論づけることはむずかしい。

もし必要であるとすると次の点が問題となる。データフローマシンはその並列性によりデータがマシン全体に散在し，かつ非同期的に処理が進行するため，従来の階層化，仮想化技術がそのままでは利用できないことである。また，ガーベジコレクションについては moving collector[13] の手法を導入する必要が生じる可能性もある。moving collector 方式を考える場合は，並列アクセスにおけるクリティカルセクションの問題もあり，今後十分に検討する必要がある。

c．構造メモリとファイル

ファイル機能のような本質的な記憶はデータフローマシンにおいても重要な要素である。ファイルはデータフローマシンの入出力として考えられるが，そこでマシンのネックとならないように入出力動作が高パーフォマンスであることが必要である。今後実用的なデータフローマシンを考える場合は重要な問題である。

―参考文献―

1) データフローアーキテクチャの研究開発；昭和５５年3月，社団法人日本電子工業振興協会

2) W.B. Ackerman : A Structure Processing Facility for Data Flow Compu-ters, ICPP, 1978

3) R.E. Thomas : I-structure An Efficient Data Structure for Functional Languages, MIT/LCS/TM－178, Sept, 1981

4) Arvind : A Data Flow Architecture with Tagged Tokens, MIT/LCS/TM－174, Sept, 1980

5) R.A. Iannucci : Instruction Set Definition for a Tagged-Token Data Flow Machine, CSG Memo 212, MIT, Dec., 1981

6) 中村，長谷川，雨宮 : リスト処理向きデータフローマシン用構造メモリの設計と評価，信学技報 EC 81－32，1981

7）三上，長谷川，雨宮　：　リスト処理向きデータフローマシンのシミュレーションによる評価，信学技報　EC　81－69，1982

8）雨宮　：　データフローマシンでのリスト処理におけるゴミ集め法，第23回情処全大　4H－4，1981

9）伊藤，来住，安原，河村　：　データフロー計算機D^3Pの実験システム，情報処理　計算機アーキテクチャ，42－1，1981

10）山口　：　記号処理データフローマシンにおけるデータ構造操作の検討，第22回　情処全大　1981

11）山口，弓場　：　記号処理用データ駆動計算機の構成，第23回　情処全大，1981

12）横井　：　記号処理データフローマシン基本計算機構の紹介，第21回　情処全大，1980

13）J.Cohen : Garbage Collection of Linked Data Structures, ACM Computing Surveys, Vol.13, No.3, 1981

14）J.Dennis et.al. : An Abstract Implementation for Concurrent Computer with Streams, Proc. of 1971 International Conference on Parallel Processing

15）W.B.Ackerman : A Structure Memory for Data Flow Computers, TR－186 MIT Aug, 1977

16）W.B. Ackerman : A Sturcture Controller for Data Flow Computers, CSG Memo 156, MIT, Jan. 1978

17）D.W.Clark et.al : An empirical study of list structure in Lisp, Commun. ACM 20, 2, Feb. 1977

18）D.W. Clark et.al. : A note on shared list structure in Lisp, Inf. Process.Lett 7, Oct.1978

4.5 ネットワーク構造とアクティビティ割り付け

本節では，データフローマシンの個々の機能モジュール間をつなぐネットワーク，並びに，複数のモジュールを有効利用して処理を進めるためのアクティビティ割り付け法について述べる。4.5.1項で，データフローマシン内で考えられるネットワークを列挙し，それぞれに対する要求仕様を述べ，4.5.2項ではネットワーク技術の現状について述べ，4.5.1項で列挙したネットワークとの適合性について検討する。4.5.3項では，アクティビティ割り付けの概略について述べ，4.5.4項では具体的な方式について検討する。

4.5.1 データフローマシンにおけるネットワークとその役割り

データフローマシン内に考えられるネットワークは，その転送するデータの性質により，トークン転送用，命令転送用，構造体転送用，プログラム転送用に分けることができる。

A．トークン転送用ネットワーク

演算の結果を所定の命令セルを含んだアクティビティメモリユニットへ転送するネットワークであり，エグゼキューションユニットからアクティビティメモリユニットへの転送路であるD-net（Distribution net）や構造体メモリからアクティビティメモリへの転送路であるSM−AM net がこれに含まれる。

このネットワークは，一定サイズ（60〜100ビット程度）のランダムにかつ高頻度に発生するトークンを非同期に処理する必要があり，このネットワークの性能がマシンの性能に大きな影響を与える。

B．命令転送用ネットワーク

アクティビティメモリに格納されている命令で実行可能になったものをオペランドとともに処理装置に転送するネットワークであり，エグゼキューションユニットに転送するA-net（Arbitration net）や構造体操作命令を構造体メモリに送るAM−SM netがこれに含まれる。

このネットワークも，トークン転送用ネットワークと同様にランダムかつ高頻度に発生する要求を処理しなければならない。ただし，個々の転送データはその中に命令コード，オペランド（複数個のこともある），目的地アドレス（複数個のこともある）を含むため，一般にトークンのサイズの平均して2倍位になると考えられる。また，1つの命令が実行可能になるためには，平均して2個弱位のトークンが必要と考えられるので，このネットワークに対する転送要求頻度はトークン転送用ネットワークの約半分位であろうと考えられる。

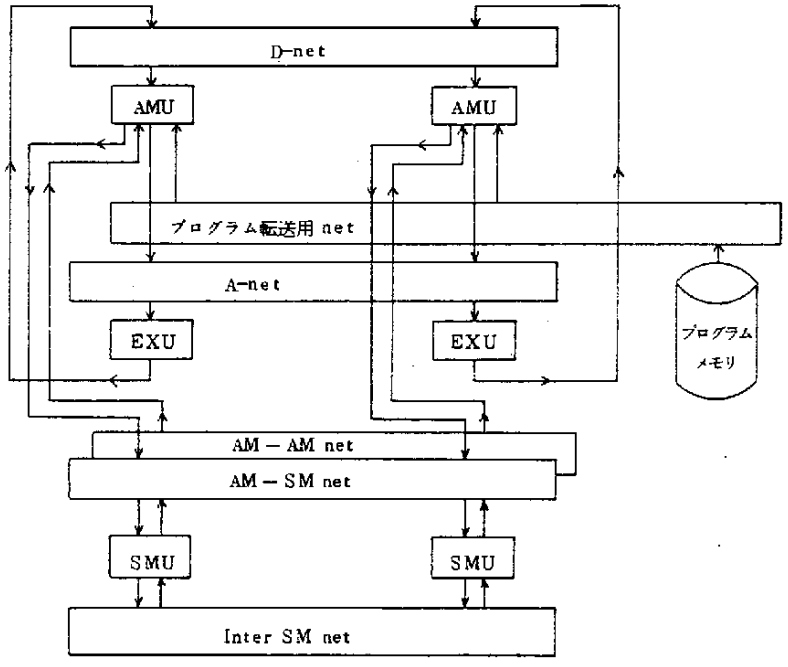
C．構造体転送用ネットワーク

構造体メモリが複数バンクより構成される場合に，バンク間で構造体やその他の情報を交換するためのネットワークであり，Inter-SM net と呼ばれる。

このネットワークに対する転送要求は命令転送用ネットワークのそれほどはないと考えられるが，構造体そのものを転送するため，一度に転送すべきデータ量は大きい。

D．プログラム転送用ネットワーク

専用のプログラムメモリ（例えば，ディスク）より，要求のあったプログラムをアクティビティメモリにロードするためのネットワークであり，プログラムの初期ロードや実行時の要求に応じてのダイナミックロードの際に使われる。従って転送要求頻度は，他のネットワークより低い。しかし，一度に転送すべきデータ量は大きい。



AMU：アクティビティメモリユニット

EXU：エグゼキューションユニット

SMU：構造体メモリユニット

図 4. 5. 1

図4.5.1に，それぞれのネットワークのシステム中で占める位置を示す。それぞれのネットワークは，転送すべきデータの性質に従ってそれぞれの要求仕様をもっているが実装の際には，性質が似ているもの同士を一本化してしまうこともできる。また，各機能モジュール間での機能分担により，ここで述べた形とは異なる形でネットワークが実装されることもあり得る。

4.5.2 ネットワーク技術の現状

A．ネットワークアーキテクチャ[1),2),3)]

ネットワークアーキテクチャを決定する際には，次の4つの選択要因があると考えられる。

a．動作モード

同期，非同期，混合

b．制御方式

分散，集中

c．交換方式

回線，パケット，混合

d．ネットワークトポロジー

共有バス

専用バス

クロスバー

多段スイッチングネットワーク

動作モードについては，データフローマシンでは，処理が非同期に行われるから非同期モードであることが必要である。制御方式については，両方式が考えられるが高頻度の要求に対応するには，集中制御の場合，制御装置に容量によりボトルネックが形成されることが多いため，分散制御の方が望しい。しかし，小規模構成の場合には集中制御でも十分対応できる。交換方式については，転送データサイズが大きく，要求頻度が低い場合には回線交換，逆に，データサイズが小さく，頻度が高い場合にはパケット交換が適しているといえよう。パケット交換の場合には各交換スイッチにおけるバッファサイズが伝達遅延との関係で重要なキーとなる。

一般にネットワークの性能を決定するのは遅延とスループットである。遅延はネットワークを構成するスイッチの動作速度，並びに，ネットワークトポロジーの影響を受ける。また，スループットは転送要求頻度の高い場合に特に重要であり，ネットワークのバンド幅（同時に処理できる要求数）等の影響を受ける。

B．ネットワークトポロジー

次にトポロジーについて検討する。

a．共有パス

一本のパスを複数のノード間通信に共用するタイプのもので，これにはバスやループが含まれる。構造が簡単で拡張が容易なのが利点であるが，集中制御であるため制御装置の容量が対応できる要求頻度に対してボトルネックとなる欠点がある。従って小規模システムに適したものといえる。しかし，階層化により要求を分散させることができればCm＊のように百台程度の構成にも対応できる。データフローマシンの中には，構造体およびプログラム転送用ネットワークとして有望である。小規模マシンであれば，トークンや命令転送用としても使えるであろう。

b．専用パス

それぞれのノード間で専用の通信パスをもつもので完全結合方式と不完全結合方式がある。

① 完全結合

あらゆるノード間に専用の通信パスをもつもので，競合がなく，直接目的地と通信できるための遅延が少ないという利点がある反面，ノード当たりのポート数が，全ノード数－1，となり，ノードの追加や，大規模構成が困難であるという欠点をもつ。拡張性に問題はあるが小規模構成システム用には適している。

② 不完全結合

1つのノードはその近傍のノードとの間にだけ専用のパスをもつものである。ノード間に直接通信するためのパスがない場合は，いくつかのノードが通信を中継する必要がある。従って，各ノードにはそのためのルーティング機能が必要となる。この種のネットワークにはその結合により局所性が導入され，この局所性のためにノード当たりのポート数は一定でネットワークの拡張が局所的な変更にのみとどまるという利点がある。これに属するトポロジーには多くのものが存在する。以下に代表的2例について述べる。

(i) 木構造

プログラム中のタスク・サブタスク等の論理的な階層関係をネットワーク上の物理的階層関係に自然にマッピングできるという利点がある。問題点としては，横方向の連絡が弱く，フレキシビィリティに欠けることや負荷がルートノード近辺に偏ることなどがある。データフローマシンの中では，D-net の候補であるが，その可能性はアクティビティの局所性・階層関係を利用した割り付け法の可能性と不可分である。

(ii) 2次元メッシュ構造[4)]

2次元格子パターンの各交点がノードに対応し，各辺が通信パスに対応したものである。プログラムに内在する局所性とネットワークの局所性を対応させることができれば，効率良い処理が可能になる。問題点としては，ネットワーク内の平均伝達遅延が全ノード数をNとしたとき，$\sqrt{N}$ のオーダーであることであり，Nの大きい大規模システムでは非局所的通信の遅延増加がプログラムの割り付け法によって問題となるであろう。データフローマシンの中では，D-netの候補であるが，その可能性は木構造ネットワークと同様にプログラムの割り付け法に依存する。

③ クロスバー[5)6)]

バンド幅が大きく遅延も小さいという利点があるが，全端末数をNとしたときに，ネットワークを構成するスイッチ素子の数がNの2乗のオーダーになり，コストおよび実装上の問題点がある。しかし，ネットワークを多段化することにより，素子数を減らすことができ，また，モジュール化することによりＬＳＩ化でき，コスト的にも実装上にも改善できる見通しがある。データフローマシン内では，トークン転送用に適している。

④ 多段スイッチングネットワーク [7)8)9)10)]

2入力2出力素子で構成する場合，全端末数をNとすると $\log_2 N$ 段で，各段にN/2 個の素子が必要である。従って，全素子数は，$\frac{N}{2}\log_2 N$ である。バンド幅が大きく，遅延は $\log_2 N$ 回のスイッチ分だけある。少量の拡張は不可能で，また，Nが大きいときには，段間の結線パターンが複雑になるという欠点がある。動作モード，交換方式については，いくつかの組み合わせが可能である。非同期式回線交換方式はプロセッサメモリ間結合として使われることが多く，同期式パケット交換はＳＩＭＤ，特に規則的なデータ交換をする処理に向いており，非同期式パケット交換はＭＩＭＤ向きといえよう。データフローマシンの中では，トークン転送用および命令転送用として，分散制御・非同期式パケット交換方式のものが有力候補である。

C．要　点

マルチ構成のシステムでは，往々にして，ネットワークがシステムの性能のボトルネックになることがあるので注意が必要である。データフローマシンで使用するネットワークの決定は用途・構成規模に応じて適切なものを選ばなければならない。小規模構成システムの場合には共有バス方式のネットワークが実装上の利点より多く使われる可能性がある。中・大規模構成になると，転送要求頻度の高いトークン転送用や命令転送用のネットワークとしては，スループットを上げるために分散制御パケット交換方式で，バンド幅の大きなものが必要になる。

一方，要求頻度の比較的低い構造体転送やプログラム転送は，多重化あるいは階層化したバス等でバーストモードで行われると思われる。大規模構成ネットワークの場合には，トポロジーが重要な問題になる。各ノードを等距離で結ぶネットワークは，一般に規模が大きくなるにつれて，遅延時間が増加したり，配線の複雑度が増加する。そのため，専用バスネットワークのような形で，局所性を導入し，平均的な遅延時間を低く抑えたり，配線の複雑度を一定にしたりする工夫が必要となるであろう。しかし，このためには，この局所性と適切に合致するようなアクティビティ割り付け法の開発が不可欠である。

―参考文献―

1） Tse-yun Feng，"A Surey of Interconnection Networks，" Computer，Vol. 14，No. 12，Dec. 1981

2） L. D. Wittie，"Communication Structures for Large Networks of Microcomputers，" IEEE Trons. Computers，Vol. C−30，No. 4，Apr. 1981

3） 田中，"並列処理システムの性能を左右する相互結合ネットワーク，"日経エレクトロニクス 12−21，1981

4） 成瀬，雨宮，"プロセッサ・アレイにおける転送系の性能評価，" 情報処理学会第２３回全国大会講演論文集，5E−6，1981

5） B. Quatember，"Modular crossbar switch for largescale multiprocessor systems − structure and implementation，" Proc. of NCC' 81，1981

6） 菅原，田中，元岡，"超多重プロセッサ間接続機構の検討，" 情報処理学会第２３回全国大会論文集，5E−5，1981

7） Chuan-Lin Wu，Tse-yun Feng，"On a class of Multistage Interconnection Networks，" IEEE Trans. Computers，Vol. C−29，No. 8，1980

8） J. B. Dennis，G. A. Boughton，C. K. C. Leung，"Building Blocks for Data Flow Prototypes，" Proc. of 7th Annu. Symp. on Computer Architecture. Vol. 8，No. 3，1980

9） H. J. Siegel，R. J. Mcmillen，"The Multistage Cube:A Versatile Interconnection Network，" Computer，Vol. 14，No. 12，Dec. 1981

10） D. M. Dias，J. R. Jump，"Packet Switching Interconnection Networks for Modular Systems，" Computer，Vol. 14，No. 12，Dec. 1981

4.5.3 アクティビティ割り付け

A．概　略

アクティビティの割り付けとは，プログラムをいくつかのブロックに分割し，複数のアクティビティメモリユニットに分散配置することを意味し，これにより，プログラム中の並列実行可能なアクティビティを複数処理装置上で並列実行することを図ると同時に，負荷の分散を行い，資源の有効利用を目指すことを目的とする。

B．プログラムの分割

一般にデータフロー言語で書かれたプログラムは，副作用がなく，また，同期制御がないので，分割が容易である。ただし，ネットワークトポロジーが局所性をもつ場合には，ブロック間での通信量を少なくするように分割しなければならない。一般には，このブロックとして，手続きやループボディ等が選ばれている。

C．静的・動的割り付け

データフロープログラムは，専用のプログラム・メモリに論理的なグラフ（データフローグラフ）の形で格納されていると考えられる。このプログラム・メモリからプログラムをアクティビティメモリに移すときには，次の２つの場合がある。

a．静的割り付け

プログラム全体のアクティビティメモリへの初期ロードであり，一般に複数のアクティビティメモリユニットに分割してロードする。ローダというシステムプログラムが行い，割り付けアクティビティメモリユニットの決定，並びに，論理的なグラフを物理的に割り付けられたアクティビティメモリユニットのアドレスにもとづき，物理アドレスでリンクしなおす必要がある。

b．動的割り付け

プログラムの実行時に必要になった手続き等を要求にもとづき，アクティビティメモリにロードすることであり，スケジューラというシステム・プログラムが行う。

ローダと同様に，割り付けアクティビティメモリユニットの決定，並びに，論理グラフの物理アドレスによるリンク等を行う他，呼び出し側と呼び出された側との間の引数の引き渡し，結果の返換のためのリンクも用意しなければならない。

いずれの場合も，並列実行可能なアクティビティの展開や資源の負荷状況を考慮して最適な割り付けを行う必要がある。

D．ネットワークトポロジーとの関係[1)]

アクティビティ割り付けは，ネットワークトポロジーと密接な関係がある。ネットワークトポロジーが局所性をもたない，すなわち，任意の端末間の通信遅延が一定の場合には，割り付けの際の各ブロックの幾何学的配置は処理時間に対して何の影響も与えないが，局所性がある場合には，割り付け方によりプログラム処理時間に含まれる全通信遅延時間に変化が生じるため，処理時間に差が出る。従ってこの場合の割り付けは，積極的にプログラムに含まれる局所性をネットワークの局所性に対応させ，遅延の小さい局所通信の割り合いを多くし，プログラム処理時間に含まれる全通信遅延時間を軽減することにより，より高速な処理を目指すことができる。しかし，この局所性の利用による高速処理と割り付けソフトウェアの複雑さとの間には，トレード・オフの関係があり，ネットワークに局所性がない場合はソフトウェアは比較的単純であり，逆に局所性がある場合には，より高速処理ができる可能性があるが，プログラムはより複雑になる。

プログラムに含まれる論理的な局所性にはアクティビティ間通信とデータ参照の2種類が考えられる。前者はDnet，後者はSMnet(SM−AM net およびAM−SM net 双方を意味するものとする）とそれぞれ関係がある。局所性をもったアクティビティの集合としては，現状では，手続きやループボディを考えるものが多く，一般的になっているが，それに対応したネットワークトポロジーの局所性については，共通の考え方はなく，今後の課題として残されている。一方データ参照の局所性については，応用に依存して様々であるとするのが一般的見方であり，記号処理ではデータ参照の局所性はないとする考え方もある。[2)]このような考え方に立てば，SM net には局所性を導入しない方がよく，従って，アクティビティの割り付けについても，これを考慮する必要がなくなる。

4.5.4 アクティビティ割り付け制御方式

A．ブロックとアクティビティメモリとの対応関係

一般にプログラムをアクティビティメモリに割り付ける際には，プログラム全体を1つのアクティビティメモリユニットに割り付けることはしないで，手続きやループボディ等のブロック単位に割り付けを行う。このような割り付け方式には，次の4通りの方式がある。

(1) 1ブロックを複数アクティビティメモリユニットに割り付ける。

(2) 1ブロックを1つのアクティビティメモリユニットに割り付ける。[3),4)]

(3) 複数ブロックを1つのアクティビティメモリユニットに割り付ける。

(4) 複数ブロックを複数アクティビティメモリユニットに割り付ける。[5),6)]

(1)の方式は，1つの手続き（あるいは，ループボディ）が占有するアクティビティメモリユニ

ット数が多く，また，スループットから考えて，ユニットの遊びが多いと予想されるのであまりとられない。(2)，(3)では，１つの手続きが完全に１つのアクティビティメモリユニットに含まれるので，アクティビティメモリユニット間の通信は，手続きレベルの通信になり，動的にコールリターン関係がアクティビティメモリユニット間に形成される。手続きやループボディがその中に高い並列性をもつ場合には，(4)の方式がとられ，それらを複数アクティビティメモリユニット上に横断的に展開することにより，１つの手続きあるいはループボディの中に含まれる並列な部分を同時に処理することが目指すことができる。

(4)の方式をとる場合の手続きやループボディの複数アクティビティメモリユニットへの展開の仕方としては次のようなものがある。[5]

(i) 文番号による展開

(ii) 繰返し番号による展開（ループ）

(iii) (i)と(ii)の混合方式

これらの方法は静的あるいは動的ロード時に行われるものであるから，できるだけ簡単で，高速処理可能なものでなければならない。要点は，実行可能アクティビティが，複数アクティビティメモリに分散して発生するように，あらかじめアクティビティを展開しておくことにあるのだが文番号や繰り返し番号といったプログラムの静的構造のみを反映したものからだけでは，十分な展開ができないことが問題といえよう。

B．割り付け先の決定基準

プログラムを分割して得られた各ブロックなどのアクティビティメモリユニットに割り付けるかを決定する際には次の２つのことを考慮する必要がある。

(1) ブロックのサイズとアクティビティメモリユニットの負荷状況

(2) ブロック間の相互関係と予想されるネットワーク通信量

(1)は主に負荷分散に関係したことであり，アクティビティメモリの発火制御機構の応答時間等にも影響する。アクティビティメモリユニットの負荷は例えば，メモリ使用状況やカラー（識別子の一部であり，コード共有のために使われる）の使用状況等が指標となる。(2)は処理速度に直接関係することであるが，ネットワーク（特にD-net）が局所性を持っていない場合には，通信コストは任意のアクティビティメモリ間で一定であるので考慮する必要はない。ネットワークが局所性をもつ場合には，各ブロックの割り付けは，(1)と(2)の両方を考慮して決めなければならない。幾何学的には，(1)は割り付け先が分散されるよう要請し，(2)はそれらが集中あるいは近接することを要請する。従って，場合によっては双方の要請が相反することもあり得る。

このような場合には，負荷分散による応答時間の改善と割り付け先の集中による通信コストの軽減とがトレードオフになっていると考えることができる。このような場合の処理については，ソフトウェアシミュレーションで研究する必要がある。

実行時に必要となる手続きやループボディの新たなインスタンスを生成する方式には次のものがある。

(1) コピー

新たに割り付けられたアクティビティメモリユニット（群）に，プログラムメモリからロードする。あるいは，別のアクティビティメモリ（群）に存在している同一ブロックをコピーする。

(2) 共　有

別のアクティビティメモリユニット（群）に存在しているブロックを共有して使う。

（この方式はカラーの使用を前提としている。）

(3) 状況に応じて，スケジューラがどちらかの方式を選択する。

カラーによるアクティビティ制御を持たない場合には必然的に(1)方式，そうでない場合には，(2)，(3)両方式がとられ得る。(1)，(2)，(3)いずれの方式をとるにしても，新たに生成されたブロックと，それを呼んだブロック間の動的なリンクはスケジューラがやらねばならない。(3)方式をとる場合にスケジューラが，コピーあるいは共有のどちらかを選ぶかは，前述の基準(1)，(2)と同様であるが，基準(2)については，生成されたインスタンスとその呼び出し側との間の通信量に限定される。

C．要　点

アクティビティ割り付け方式の設計に際しては，次の２つの重要な選択ポイントがある。第１はネットワークトポロジーにおける局所性であり，これを導入すれば，処理速度の向上は期待できるが，割り付けは通信コストを常に考慮するトポロジーに依存した複雑なものになる。第２は手続きやループボディ等のブロックの割り付け方式であり，複数アクティビティメモリにまたがって割り付ける方式と単一のアクティビティメモリユニットに割り付ける方式がある。前者の場合は，ブロック内の並列アクティビティを同時処理できるという利点があり，後者の場合は，コール・リターン等の高レベルの関係がハードウェアモジュール間に形成されるので，デバッグやエラー処理等が容易になるという利点がある。

以上，２つはハードウェア構成と密接に関係しており，ハードウェア規模によって選択も異なると考えられる。アクティビティ割り付け固有の問題としては，割り付け先アクティビティメモリユニットを決定するアルゴリズムや動的ロードの際のインスタンスの生成方式がある。これら

は，今後，シミュレーションにより詳細な検討が必要であると考えられる。

—参考文献—

1） J. B. Dennis，"The Varieties of Data Flow Computers，"Proc. 1st International Conference on Distributed Computing Systems，Ang.1979

2） 雨宮，長谷川，三上，"リスト処理向きデータフローマシンの検討，" 情報処理学会記号処理研究会資料 13−3，1980

3） 栗原，鈴木，元岡，"High Level Data Flow Machine（Topstar）のシステムプログラム，"信学技報 EC79−56

4） J. Rumbaugh，"A Data Flow Multiprocessor，"IEEE Trans. Computers Vol. C−26，No. 2，1977

5） Arvind，V. Kathail，K. A. Pingali，"A Data Flow Architecture with tagged - tokens，"MIT LCS TM−174，Sep. 1980

6） 伊藤，来住，安原，河村，"データフロー計算機 D^3P の実験システム，"情報処理学会，計算機アーキテクチャ研究会資料 42−1，1981

4.6 入出力

現在考えられているデータフローマシンは入出力については明確にされておらず，入出力オペレーションを如何に扱うかは未検討のテーマである。データフローマシンの基本的な思想は認められているが，現実方式に対しては未だ有効性が実証されておらず，その点に注力されているために，入出力に関しては未着手の状態のようである。

従って，データフローマシンをホストにつながった専用マシン的なもので考えて，有効性が明らかになった時点で，入出力を考えるというアプローチもあるが，ここで，入出力の問題点を考えてみることも必要であろう。

データフローマシンは，関数型言語を前提として考えているので，基本的にはメモリや変数の概念は存在しないことになる。しかし，例えば，端末からデータを入力したり，処理結果をプリンタに出力したいときや，処理結果を保存したり，以前に保存しておいたデータを取り出すような時には，メモリや変数の概念が必要になって来ると思われる。これは，データフローマシンと別の世界とのインタフェースと考えられ，この両者の間がどのように結びつくかが明らかになっていない。

また，入出力操作とは，一般的に逐次的なものであると考えられるが，そこに如何に並列性を抽出するかということと，その場合の問題点も重要である。

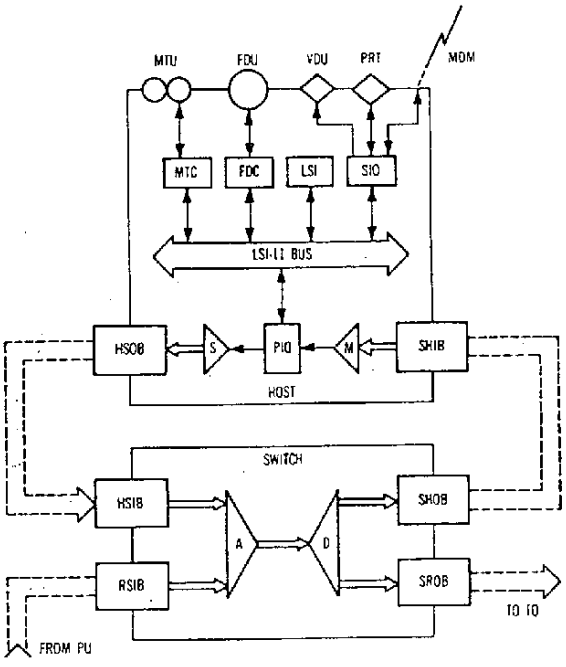
さらに，入出力に関して従来までの概念であるチャネル，割込等が，データフローマシンにおいてはどのように考えるべきなのかも明確になっていない。

4.6.1 入出力処理形態

現在，世の中で研究されているデータフローマシンにおける入出力の処理形態は，次の２つに分類出来る。

A．ホスト依存型

データフローマシンは，データ駆動による計算だけを専門に担当し，入出力に関しては，総てホストに依存する方式で，現在までに考えられているデータフローマシンは，総てこのホスト依存型である。例えば，図4.6.1に示すマンチェスタ大学のマシン〔GURD80〕は，この方式である。



DOUBLE DOTTED LINES REPRESENT ASYNCHRONOUS COMMUNICATIONS LINKS

MTU = MAGNETIC TAPE UNIT
MTC = MAGNETIC TAPE CONTROLLER
FDU = FLOPPY DISC UNIT
FDC = FLOPPY DISC CONTROLLER
VDU = CONTROL TERMINAL
PRT = 30-CHAR/s DECWRITER
MDM = 300-BAUD MODEM
SIO = 4-CHANNEL SERIAL INTERFACE
LSI = LSI-11/2 + 28-k WORD MEMORY
PIO = 16-BIT PARALLEL INTERFACE
M = 96-BIT to 16-BIT MULTIPLEXER
S = 16-BIT TO 96-BIT DEMULTIPLEXER
A = 2-WAY SYNCHRONOUS ARBITRATOR
D = 2-WAY DISTRIBUTOR
SHIB = SWITCH TO HOST INPUT BUFFER
HSOB = HOST TO SWITCH OUTPUT BUFFER
HSIB = HOST TO SWITCH INPUT BUFFER
RSIB = RING TO SWITCH INPUT BUFFER
SROB = SWITCH TO RING OUTPUT BUFFER
SRHB = SWITCH TO HOST OUTPUT BUFFER
PU = PROCESSING UNIT
TQ = TOKEN QUEUE

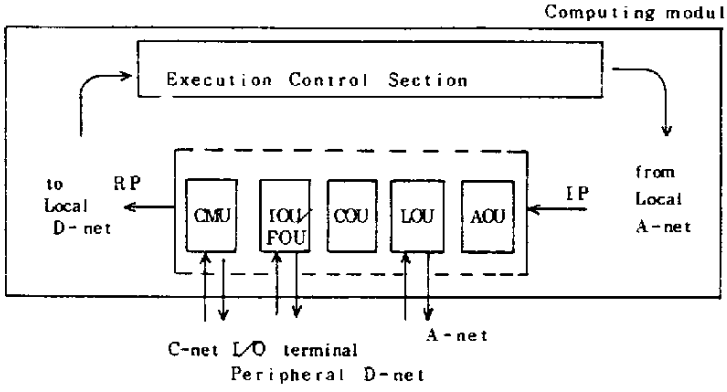
図 4.6.1　ホスト依存型の入出力処理形態例

（マンチェスタマシン）

データフローマシンを常に専用マシン的にホストに結合させておく形態で実用化まで進めることも考えられないことではない。ホストとの間のデータのやりとりが少なくて，データフローマシン内でのデータ処理に大きな負担があるような応用の場合は充分実用になるものと思われる。したがって，逆に，入出力のトラヒックが大きいような応用に対しては，ここがネックとなる可能性があるので，入出力まで含めたマシン（自立型）にしなければならないだろう。

B．自立型

データフローマシンをスタンドアローンで考えて，ホストを想定しない方式である。この場合は，入出力の処理もデータフローマシンに取り込んでしまうことになる。本方式を具体的に検討している所はない。一例としては，通研マシンの構想がある。図4.6.2にそのＰＵ部のブロック図を示す。



Arithmetic operation unit (AOU)
List operation unit (LOU)
Control operation unit (COU)
I/O operation unit (IOU)
Communication unit (CMU)
File operation unit (FOU)

図4.6.2　自立型の入出力処理形態例（通研マシン）

このマシンでは，ＰＵにIOU(I/O Operation Unit)およびFOU(File Operation Unit)を設け，それぞれ，端末Ｉ／Ｏ制御とファイルＩ／Ｏ制御を行うように考えられている。Ｉ／Ｏデータは総て，IOU/FOU 経由でパケットの形に組み立てられ，マシン内では他の演算データと同じように取り扱われる。

他の例では，Rumbaugh マシン〔RUMBAUGH 77〕がある。周辺装置プロセッサを設けて外部周辺装置とのインタフェースを司り，内部モジュールからは１つの手続実行プロセッサのように見せている。これも一種の自立型と看做すことにする。

4.6.2　入出力装置接続形態

入出力装置が，システム内のどこにつながるかによって分類すると次の２種になる。ホスト依存型は自明であるので，ここでは，自立型だけを考える。

A．タイプ1

ＰＵの一部または全部が入出力装置の接続機能を有しており，ＰＵ内でI／O 処理をするもの。従って，入出力装置は１個または複数個のＰＵに接続される。この方式では，入出力を完全にデータフローマシン内に取り込んで考えることになる。ＰＵのハード量が大きくなる欠点があるがシステムとしては，最も理想的と考えられる。図4.6.3参照。

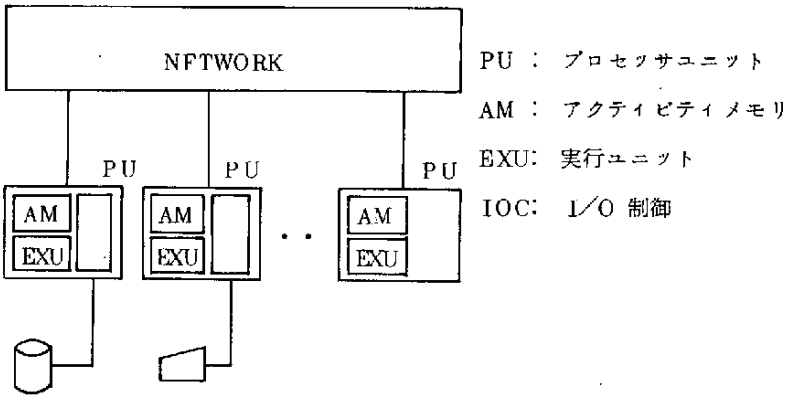


図4.6.3　入出力装置接続形態タイプ1

B．タイプ2

１個あるいは複数個の専用 I／O プロセッサをＰＵと並列に設置する形であり，入出力装置はこのプロセッサに接続させる。I／O プロセッサとして，従来型の汎用マシンを代用することも出来るので，実現上の融通性が出て来る。データフローマシンへの最初のアプローチとしては，考え易い形と思われる。図4.6.4参照。

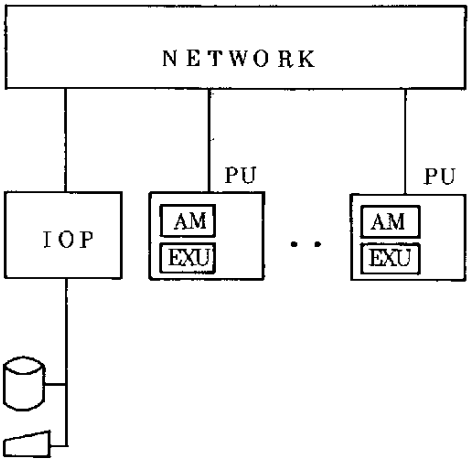


図4.6.4　入出力装置接続形態タイプ2

4.6.3　チャネルと割り込み

A．チャネル

チャネルの概念は，データフローマシンにおいてはどのように考えたら良いのだろうか。チャネルの目的に立ちかえって考えてみると，それは，次の3つを可能あるいは容易にすることであった。

・ＣＰＵでのデータ処理と入出力オペレーションを同時に実行

・複数個の入出力オペレーションを同時に実行

・種々の入出力装置を標準的手順で制御

これを見ると，最初の2つに関しては，従来のマシンのように，プログラムカウンタが1個だけであって逐次的にしか命令を実行出来ない時には妥当な考えであろう。しかし，データフローマシンの場合には，同時に多数の命令を実行することができるので本来の処理と入出力処理とを同じマシン上で同時に実行させることが考えられる。その場合，ＡＭの中に本来の処理用と入出力処理用（チャネルプログラム）の命令が混在する形と，入出力処理用には専用のＰＵを割り当てて，その中のＡＭには入出力処理用の命令しか入れない形とがあると思われる。いずれの場合も，チャネルプログラムをデータフローマシン上で動かすことになると，チャネルという概念は無いように思う。一方，3番目の標準化という意味では，チャネルの考え方は，データフローマシンにおいても必要と考える。従って，チャネルの概念は残すが，実装法は，かなり従来と異なったものになり得るのではないかと思われる。

B．割　込

従来のマシンでは，割込は次のような意味を持っていた。即ち，「システム内外の状態変化をＣＰＵに通知し，実行中の処理を一時中断して緊急度の高い処理を優先的に行う」。

ここで，実行中の処理を一時中断する理由は，勿論，従来マシンではプログラムカウンタが一個だけなので，一時には，一個の処理しか実行出来ないからである。それ故，データフローマシンのように同時に多数の処理を実行出来るマシンでは，今までのような意味での割込はなくすることが出来る可能性はある。そこで，割込の原因をみると，①入出力割込　②プログラム割込　③外部割込　④マシンチェック割込　⑤スーパーバイザコール割込がある。このうち，①～④は不測の事態が発生したことにもとづくものであり，何時生じるかを絶えずチェックしている訳には行かない。従って，ハード的なトリガ機能によって，そのための処理を起動させる必要がある。但し，ここでデータフローマシンにおいて，従来のマシンの場合と本質的に異なると思われるのは，そのトリガが発生した時に実行中であった処理を一時中断する必要はないことであろう。即ちそれまでに実行されていた処理プログラムと，新に起動された監視プログラムとが並列に実行出来ることである。このことは，概念的には，問題プログラム状態と制御プログラム状態とが混在していることになり，その点で問題がないかを検討しなくてはならない。例えば，新たに起動されたプログラムの実行の結果，他の処理プログラムを停止させる必要が生じるとか，同時に実行中の処理プログラムから別の割込原因が発生した場合の処理の仕方等を考えなくてはならない。

⑤の原因については，プログラムで意識的に監視プログラムを呼ぶのであるから，ハードによるトリガ機能は必要ないと考えられ，単にプログラム的な起動によって監視プログラムが動き出し，処理プログラムと同時に実行されると考えられる。

入出力割込を含む割込全般について，従来のように実行中の処理を中断するという思想でなく新に別のプログラムも起動されて同時に実行されるという見方が出来そうである。

4.6.4　入出力と並列性

入出力の機能を考えた場合，本質的に逐次的なものなのであろうか。従来までのマシンが一時には一個の命令しか実行出来ず，入出力プログラムが逐一に実行される故，プログラマも，本質的ではないが，逐次的に考えているように思われる。

例えば，繰返しのボディの中でREAD関数が呼ばれているとする。その場合，１番目のデータ２番目のデータ，…の順序が入れかわっては困るように思われるが，本来は，先に２番目のデータがREADされて，そのあと１番目のデータがREADされても構わない筈である。必要なのは，READされたデータがどのREAD呼出と対応しているかを明確にすることである。この対応は，

トークンに付属しているタグを利用することによってうまく付けられるだろう。ただし，入力するデータも値だけでなく，それに関連する情報（変数識別名などのタグ）を附加したものにしなくてはならないだろう。これら，具体的な実現方式を検討する必要がある。

出力の例でも同様に，並列に処理しても良い事が多い。例えば，ある図形を出力（表示）するような時は，その順番には無関係であり，最終的な形が意味を持つので，並列に処理して順番も前後して構わないだろう。しかし，プリンタに出力するような場合，一般にフォーマットが重要であり，機器のメカニカルな動きの制限で，逐次的にせざるを得ない状態にあると考えられる。

このように，入出力においても，データフローの性質を生かして並列処理を取り入れることは可能であるが，本質的に逐次性が必要なものを良く見極めて対処しなければならない。

4.6.5 入出力動作と基本命令

種々の周辺，端末装置を，データフローマシンの各要素とどのように関連させて扱うかについては未だ明らかにされていない。入出力命令の設定および意味づけも確立されていない。

ここでは，最小機能として，キーボード端末とファイルに限って考えてみる。

A．端　末

キーボードのような端末を考えると，これは純粋にプログラムから，データフローマシン外の世界とインタラクトすることを指示することと考えられる。従って，explicitにREAD，PRINTのような命令が必要である。

この場合の動作は図4.6.5のようになると思われる。（入力動作の例）

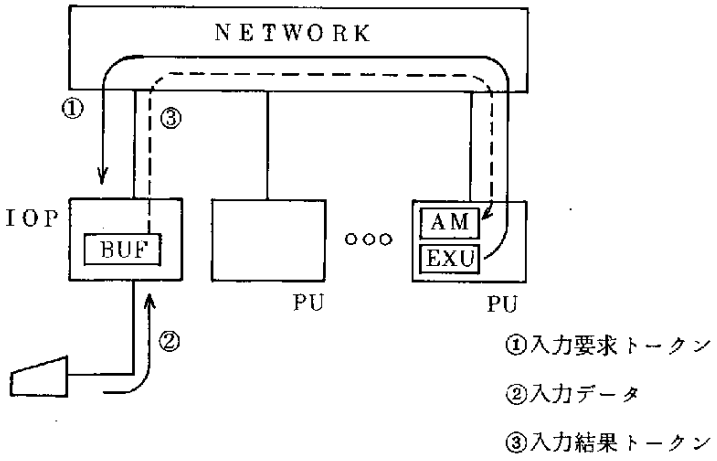


図4.6.5　端末からの入力動作

ＰＵで検出された入力命令は，ＩＯＰへ入力要求トークンとして送られ，ＩＯＰでは，この命令から通常のＩ／Ｏ制御を開始し，端末からのデータがバッファにたまると，トークンの形に変換して入力命令を出したＰＵへ送ることになる。このトークンは，ＡＭ内の入力値を待っているオペランドレジスターに渡される。この場合に考えなければならないことは，入力命令がいつ何によって起動されるのかということと，入力命令トークンにどんな情報を保持させるかということさらに，ＩＯＰ内に必要な情報は何か等である。

B．ファイル

データフローマシンは，関数型を基本とするので，原理的には処理結果は総て記憶から消えてしまうことになる。現実には，データやプログラム等の情報を，どこかに保存しておき，後に，予め保存されていた情報を取り出すということが必要である。そのために，ファイルの概念が要る。

従って，ファイルを意識した操作命令を設定することが必要である。これには，システムレベルのものとユーザレベルのものがある。ユーザレベルのものでは，プログラムの中で意識して情報の保存，取出しを行えるように保存用にMEMO，取り出し用にRECALLという命令が考えられる。その際，保存する値には識別名をつけて，ユーザレベルでは，後に参照する時にその識別名を使って取り出せるようにすることが重要と思われる。

一方，システムでは，これらの要求に対しての操作と，システム自体が主体となって行う操作とがある。それらは，プログラムのロード，ストラクチャメモリ等データのロード／アンロードであり，これら各々についての現実方式を具体的に研究することが必要である。

ファイルのロード／アンロードの場合について，動作の流れを考えてみると，図4.6.6のようにするのも一方法であろう。

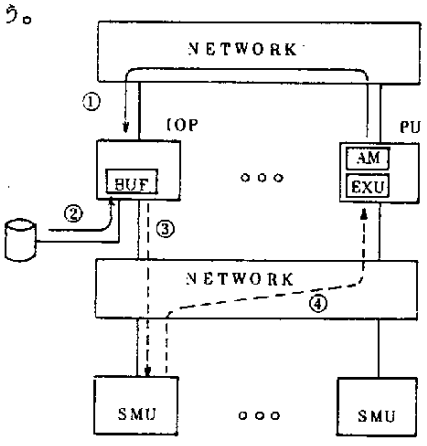


図4.6.6　ファイルからのロード動作例

この図で，入力要求が①ＩＯＰに伝えられると，ＩＯＰからファイルのロード起動が行われ，②読み出されたデータは，ＩＯＰ内のバッファに蓄えられ，③一定量になるとＩＯＰから順次ＳＭＵへ，トークンの形で送られる。ロードが完了した後，④ＰＵが，そのデータに対して操作を行う。

―参考文献―

1）〔GURD80〕Gurd，J. and Watson，I.，"Data Driven System for High Speed Parallel Computing － Part 2 ： Hardware Design"Computer Design July 1980

2）〔RUMBAUGH77〕Rumbangh，J.，"A Data Flow Multiprocessor，"IEEE Trans. on Comp. vol. C－26, No. 2，1977

4.7 データフローマシンの実装法

本節では，データフローマシンの実装に関連した事項について述べる。4.7.1項では，データフローマシンの全体構成について種々の可能性について述べ，4.7.2項では，機能モジュール間のインタフェースについて述べ，4.7.3項では，マシンの保守に関連するサービスプロセッサの実装法について述べる。4.7.4項では，データフローマシンのスループットと問題の並列度との関係について述べ，4.7.5項ではＶＬＳＩ技術の導入，4.7.6項では，データフローマシンのコンポーネントのＶＬＳＩ化について述べる。4.7.7項では，ネットワークについて実装上の観点より述べる。

4.7.1 全体構成

データフローマシンの基本構成要素としては，

① アクティビティを格納し，それらの発火検出を行うアクティビティメモリユニット（AMU：Activity Memory Unit ）

② 発火したアクティビティの処理を行うエグゼキューションユニット（EXU：Execution Unit ）

③ 構造体データを格納し，構造体操作命令を実行する構造体メモリユニット（SMU：Structue Memory Unit ）

④ 入出力処理を司さどる入出力ユニット（IOU：I/O Unit ）

等の機能モジュールがあり，それらの間をつなぐネットワークとして

⑤ AMUからEXUへ発火アクティビティを転送するA-net(Arbitration net)

⑥ EXUからAMUへ結果を転送するD-net(Distribution net)

⑦ AMUからSMUへ構造体を操作する発火アクティビティを転送するAM−SM net

⑧ SMUにおける演算の結果をAMUへ転送するSM−AM net

⑨ SMU間の構造体転送用ネットワークであるInter-SM net

⑩ プログラムをAMUへ転送するためのプログラム転送用ネットワーク

等がある。一般に，マルチ構成のデータフローマシンではAMU, EXU, SMU, IOU 等がそれぞれ複数個存在し，それらが互いにネットワークを介して通信をしながら独立に処理を進める。

全体構成を決定する際には，次のような選択要素がある。

(1) ネットワークトポロジー

(2) SMUの機能

(3) ネットワーク通信の論理レベル

(4) ＶＬＳＩ化への適合性

(5) ＩＯＵの結合位置

ネットワークトポロジーとしては，任意の端末間での均一な通信コストを保証するものと，そうでない局所性をもったものが考えられる。ネットワークが前者のようなトポロジーをもつ場合，A-netでは，１つのＡＭＵからは，すべてのＥＸＵが等距離に見える（同一の通信コストでデータ転送できる）ので，負荷の低いＥＸＵを選んで，そこへ命令を送ることができＥＸＵ全体として負荷の均等化を行うことができる。また，D-netの場合は，１つのＥＸＵからは，すべてのＡＭＵが等距離に見えるので，結果を送るべきアクティビティがどのＡＭＵに存在しても良く，従ってアクティビティのＡＭＵへの割り付けが簡単なものになる。同様のことがSM-AM netについてもいえる。また，AM-SM netについては，１つのＡＭＵからすべてのＳＭＵが等距離に見えるので，あるアクティビティが参照する構造体データは，基本的にどのＳＭＵにあっても良く，構造体データのＳＭＵへの割り付けが比較的簡単になる。このように，局所性のないトポロジーをもつネットワークを用いれば，アクティビティや構造体データの割り付けが簡単化され，また，ＥＸＵの負荷分散が容易になされるという利点がある。一方，ネットワークが局所性のあるトポロジーをもつ場合，手続き内部に見られるような局所的なアクティビティ間通信やデータ参照をトポロジーの局所性に対応させることにより，ネットワーク通信における遅延を大幅に軽減させて処理速度を上げることが可能になる。一般に，手続きやループボディのような局所性をもったアクティビティの集合をＡＭＵに割り付けると，１つのアクティビティの発火した結果生じた値を送る先のアクティビティが多くの場合，同一ＡＭＵにあることが予想される。従って，このような状況では，ＡＭＵとＥＸＵを１対１に対応づけて１つの処理ユニット（PU：Processing Unit）とし，ＡＭＵからＥＸＵＥＸＵからＡＭＵへのそれぞれのデータ転送用にネットワークを介さない専用パスを設けることにより，ネットワーク伝達遅延を排除し処理の高速化を図ることができる。

この場合，A-netは，個々のＡＭＵと対応するＥＸＵだけを結ぶ専用パスとなり，D-netは，ＥＸＵから同一ＰＵ内のＡＭＵへの専用パスと別のＰＵ内のＡＭＵへの転送用のネットワークからなり，個々のＰＵ内に転送先に応じて転送路を両者のどちらかに振り分ける交換スイッチが必要になる。また，ＡＭＵに割り付けられているアクティビティの参照する構造体データが局所的に限られている場合には，これらを含むＳＭＵを個々のＰＵ内に置くことにより，アクセス時間の短縮をはかることができる。このときは，AM-SM netが，ＰＵ内アクセスの専用パスとＰＵ外からのアクセス用のネットワークからなり，SM-AM netもＰＵ内パスとＰＵ外への結果転送用ネットワークからなると考えられる。実装上は，このAM-SM net，SM-AM net両者で使われる

ＰＵ間通信用ネットワークは，D-netで使われるＰＵ間ネットワークと共用されることも考えられる。

ＳＭＵの機能は，ＥＸＵとの間で，構造体操作命令や，メモリ管理機能をどのように分担するかに依存し，それにより，AM−SM net，SM−AM net のシステム内での結合位置が異なる。ＳＭＵが構造体操作命令を直接ＡＭＵから受けとり，結果を直接ＡＭＵに返し，かつ，参照カウント等のメモリ管理機能もＳＭＵ内で行う場合には，SM−AM net，AM−SM netともにＡＭＵとＳＭＵを結合するものになり，構造体操作命令とそうでない命令は，それぞれ，AM−SM net，A-netに分けてＡＭＵより転送される。これに対し，構造体操作命令についても部分的にＥＸＵが関与するような場合には，SM−AM net，AM−SM netは実際には，ＳＭＵからEXU，ＥＸＵからＳＭＵへの転送にそれぞれ使われることになる。このような場合には，ＳＭＵは単なるメモリに近いものになり，read，write命令のみを処理し，これ以外の高次の構造体操作演算や参照カウントの管理，結果のＡＭＵへの転送等は，ＥＸＵで他の命令の処理と同様になされることになる。

ネットワーク通信の論理レベルについては，アクティビティの割り付け方式と関連している。手続き等を複数のＡＭＵに展開して割り付ける場合は，一般にD-netの通信はアクティビティ間通信であり，ＥＸＵより発し，ＡＭＵに終る。これに対し，手続きのような内部に局所性をもったものを中心に処理する場合には，これをＡＭＵとＥＸＵが対になった１つのＰＵに割り付けるとD-netを介したＰＵ間通信は，コール・リターン等の手続き間通信だけになる。このような場合には手続き呼び出しをコール・リターンまとめて１つの演算と考えＥＸＵで一括処理することにし，D-net通信用インタフェースをＥＸＵにCall port，Return portという形で設けることが考えられる。

ＶＬＳＩへの適合性は，実装上非常に重要な問題である。システムのLogic／pin ratio が高くなるような分割が問題となる。これについては，4.7.6項で述べる。

ＩＯＵの結合位置については，ＥＸＵと同じレベルとするのとＡＭＵと同じレベルとするのと２通りの考え方がある。ＩＯＵは複数タスク間で共有される資源であるため，その結合位置により，アクティビティの割り付け方にも影響がある。

4.7.2　モジュール間インタフェース

データフローマシンでは，個々のアクティビティの発火検出，命令実行，結果転送等を機能モジュール間でパイプライン処理することにより，スループットを上げ，並列処理能力を高めることができる。このときのモジュール間インタフェースの制御方式としては同期と非同期の２種の方式が考えられるが次の３点で非同期式の方が望ましい。

① モジュールごとに別々の製造工程を経ることからくる物理誤差に対する許容性

② ステージ内遅延が不均一でも良いこと

③ モジュールを結合して，テストする際のタイミング調整が不要なこと

しかし，個々のモジュール内部の処理は非同期でも，モジュール内クロックにもとづいた同期式処理でも良い。

モジュール間の非同期インタフェースはバッファリング，およびモジュール内部が同期処理のときは，非同期入力のモジュール内クロックへの再同期化を行わねばならない。

モジュール間データパスの幅は，VLSI化を考えない限り，必要なだけの幅をとることができる。しかし，一般に，モジュール間で転送されるデータ単位は，構造体やプログラムを除けば，１００ビット前後であり，また，各モジュールは入出力用に最低２ポート必要であり，さらに，VLSI チップでとれるピン数が１００～１２０ピンであることを考えるとVLSI 化の際には，データ幅が制限され，データを数回に分けて転送することが必要となるであろう。

4.7.3 サービスプロセッサの実装

基本機能モジュール以外に，システムの保守やハードウェアの故障診断等の支援のためにサービスプロセッサを実装することが考えられる。初期の実験機では，システムの保守・診断のために不可欠であると思われるが，将来のデータフローマシンにこれが残るかどうかについて不明である。それは，マルチ構成であるが故に，データフローマシンには冗長性等を利用した自己診断修復機能等が可能となるかもしれないからである。

サービスプロセッサの実装は，その性質からして，システムハードウェアから独立しているのが望ましく，独自のバス（例えばバス）で個々の機能モジュールと結合されるという形が考えられる。サービスプロセッサは，このバスを介して，メモリにアクセスしたり，メンテナンステストを行ったり，HALT 命令を実行したりすることが考えられる。

4.7.4 スループットと問題の並列度

データフローマシンのスループットは通常独立動作する個々のＥＸＵのスループットの和として示される。しかし，与えられた問題がシリアルな性質をもつ場合には，このスループットだけの性能は当然出ない。このような場合には，多くのハードウェア資源が遊んでしまうからである。データフローマシンのスループットは与えられた問題が十分な並列性をもったときに初めてその真価を発揮するものである。従って，この項では，スループットとそれに見合うだけの並列度との関係についての簡単な考察について述べる。

今，EXUのスループットをα命令/秒とするとEXUはα秒ごとに1つの命令を消化して結果を生成する。AMU，EXU間のローカルループだけを考え，それを一周するのに要する時間をT秒としたとき，EXUを常にビージーにするためには，σdT個の発火したアクティビティやトークンがループ中の各パイプライン・ステージに少なくとも存在しなければならない。ただし，$\sigma \fallingdotseq 1 \sim 2$であり，アクティビティが発火するのに必要なトークン数や，1つの命令が生成するトークン数に依存して決まる。システム全体で，N台のPUがあり，各PUに1つのEXUがあるとすると全EXUをフル稼動するためには，少なくともN$\sigma\alpha$T個以上の発火したアクティビティやトークンが処理中である必要がある。1つのアクティビティを発火させるのに複数個のトークンが必要であることを考えると，約NαTが並列に発火しているアクティビティ数と考えられ，従って，NαT以上のプログラムの並列度が必要となる。さらに，ネットワークがパケット交換でパイプライン的に処理されているとするならば，このネットワーク中にあるパケット数を考慮に入れて，必要な並列度はさらに大きくなる。例として$1/\alpha$を20μ sec，Tを400μ sec，Nを8台とすると必要な並列度NαTは160になる。

—参考文献—

1） Arvind，V. Kathail，K. A. Pingali，"A Data Flow Architecture with Tagged Token，"MIT. LCS TM-174, 1980

2） J. Gurd，I. Watson，"Data Driven System for High Speed Parallel Computing — Part 2： hardware design，"Computer Desigh，jul. 1980

3） I. Watson，J. Gurd，"A Prototype Data Flow Computer with Token Labelling，" Proc. of NCC' 79 1979

4.7.5 VLSI技術の導入

VLSI することによる一般的利点としては，

① コスト減：ただし，大量に使用するものでないと引き合わない

② 容積減

③ 高信頼性：システム全体の部品数が減るため

④ 電力消費量減

⑤ メンテナンス・コスト減：チップ交換で済む

等があり，結果として，コンパクトでメンテナンスの容易なシステムが低価格で得られるようにな

る。

一方，設計にインパクトを与えることが予想されるVLSI 技術の特性としては

① 増大するLogic ／ pin ratio：ピン数に対する制限はそのままで，集積度が増し，チップあたりの詰め込める機能が増加したため

② チップ内信号伝播速度とチップ外へ出るときの信号速度の比が１００対１：これは必要なキャパシティロードの差からくる。

がある。チップ内の制御方式としては，大量の資源を管理する必要があり，集中制御だと ボトルネックを形成する可能性があるため，分散制御が良く，また，同期式の大規模回路では，場所により受けとったクロックにずれが生じるため，局所的には同期式であっても全体としては非同期制御が良い。

以上のVLSI技術の特性を生かした実装法としては，

① 多くの同一種チップを用い

② チップの機能はチップ内でほぼ閉じていて，チップ外への通信は低頻度かつ少ないデータ幅で済む

ようなものといえよう。

―参考文献―

1） A. L. Davis，"A Data Flow Evaluation System Based on the Concept of Recursive Locality，" Proc. of NCC' 79 1979

2） D. A. Patterson，E. S. Fehr，C. H. Séquin，"Design Considerations for VLSI Processor of X-TREE，" Proc. of Annual Symposium on Computer Architecture 1979

4.7.6 データフローマシンのＶＬＳＩ化

VLSI チップでデータフローマシンを実装する場合には，何を１チップに納めるかが問題であり，基本的に２つの方式がある。１つは機能モジュールごとに１チップ化する方式であり，他は，ＰＵ単位に１チップ化する方式である。

機能モジュール単位にVLSI化する場合は，ＰＵ単位にVLSI 化するのに比べ，各モジュールを高機能化でき，メモリ容量も大きくとることができる。また，種々のモジュール間結合をとるこ

とにより，異なる構成のシステムを作ることができる。反面，チップ内での閉じた処理はなく，外部とのデータ交換を頻繁に行うため，ピン数制限によるデータ幅の制限がボトルネックを形成する恐れがある。

ＰＵ単位にVLSI化する場合は，その中の各機能モジュールは前者と比べて小規模のものになる。しかし，ＰＵ内のＡＭＵとＥＸＵの間にローカルバスを設けることにより，チップ内処理が可能であり，チップ外との通信も低頻度に抑えることができる。

4.7.7 ネットワークの実装

この項では，ネットワークについて実装上の観点から述べる。ネットワーク実装上の問題点としてては次のものがある。

(1) 複雑さ：ネットワークを構成する素子数および線長

(2) ポート数：ネットワークノードあたりのポート数

(3) データ幅：１ラインあたりのデータ幅

(2)と(3)でノードあたりの入出力ピン数が定まる。

(4) 拡張性：連続的に種々のスケールの構成がとられること。

以上の点について，代表的ネットワークについて述べる。

A．バス，ループ

構造が簡単なため，実装は容易であり，拡張性も良い。

B．完全結合

ポート数ネックのため，大規模構成は難しい。拡張のたびに，すべてのノードについて，ポートを拡張しなければならない。

C．木構造，２次元メッシュ構造

基本構成要素数，線長，ポート数ともに少なく，拡張性も良い。しかし，各ノードスイッチのロジックは複雑であり，ルーティング，バッファリング等の通信中継機能が必要である。ノードスイッチをＬＳＩ化すれば実装は容易になる。

D．クロスバー

一般に，素子数が端末数の２乗のオーダであり，複雑である。しかし，多段化，ＬＳＩモジュールの利用により実装は簡単になる。ビットスライス化すれば，データ幅も十分とれる。

E．多段スイッチングネットワーク

端末数をＮとしたとき，素子数は$O(N\log_2 N)$であり，線長は$O(N^2)$である。拡張性は悪い。ＬＳＩ化については，２×２素子の場合，データ幅８ビットで，ピン数はほぼ５０本になる。

4×4素子でＬＳＩする場合は，ピン数制限により，データ幅は8ビット位が限度となる。ビットスライス化することにより，データ幅を拡張することは可能である。

4.8 制御ソフトウェア

本節では，データフローマシンの制御ソフトウェアについて述べる。4.8.1項では，プログラムの起動法について述べ，ローダスケジューラの概略，ならびに，システムのブートの手順についても述べる。4.8.2項では，データフローマシンのＯＳの必要とされる機能，および，各種管理表，および，ＯＳの構成について述べる。

4.8.1 プログラムの起動

はじめに，ユーザプログラムの起動とシステムの起動について概略を述べる。ユーザプログラムは一例として，次のようにして起動される。

① ＯＳにプログラム名を告げ起動要求を出す。

② ＯＳはローダにロード要求を出す。

③ ロード完了とともに，ＯＳはローダよりプログラムのルートノードのアドレスを受けとり，そこへスタートトークンを送る。

これにより，ユーザプログラムはルートノードより次々に発火していき実行される。

システムの起動は，あるＡＭＵ中のＲＯＭ内に置かれたブートルーチンを通じて行われる。その手順は次の通りである。

① 電源投入直後に，ブートルーチンへスタートトークンが自動的に送られる。

② ブートルーチンは，ＩＯＵを通じて，プログラムメモリよりＯＳの核をロードし，それにスタートトークンを送る。

次に，ローダ並びにスケジューラの動作について説明する。ローダの動作の概略を次に一例として示す。

① ＯＳより，プログラム名とともにロード命令を受けとる。

② プログラムメモリを調べて，必要なコードブロック（手続きやループボディ等）のサイズ，並びに，相互関係を調べる。

③ システム資源の状況にもとづき，あらかじめ決められた割り付け戦略により，コードブロック単位にＡＭＵへの割り付けを決定する。１つのコードブロックを複数ＡＭＵへ展開する場合にはその方式も決定する。この時点で，コードブロック単位で，物理的ベースアドレスが決定し，AMU群へのコードブロックの展開の仕方も定まっているので，各アクティビティの物理アドレスも決定される。

④ プログラムメモリからプログラムを読み出し，各アクティビティのdestinationフィールド

を上で定まった物理アドレスにもとづいて書き換え，AMUに格納する。

⑤ カラーを用いる場合には，③の時点でそれを決定し，カラー管理表に登録する。また，AMUの使用についても同時にシステムの管理表に登録する。

⑥ ロードしたプログラムのルートノードの物理アドレスを要求元に返す。

以上の中，④については，プログラムメモリとの入出力操作を必要とするので，ＩＯＵの制御のもとで行う。

スケジューラは，プログラムの実行時に必要となった手続きやループボディの生成を行う。その動作はローダのそれと似ているので，異なる点だけを示す。

① 必要なコードブロック名を呼び出し側のアドレスとともに受けとる。

② ローダの②と同じ。

③ AMUの管理表を調べて，すでにAMUにロードされているコードブロックの共用を考慮する。共用しない場合は④と同様にしてロードする。共用する場合は，④は不要で，⑤で直接，新しいカラーを割り付けてもらう。

④ ローダの④と同じ。

⑤ ローダの⑤と同じ。

⑥ 呼び出し側のアドレスおよび生成されたコードブロックのアドレスより，結果返換のためのリンクを行い，引数を渡して生成されたコードブロックに起動をかける。

以上は，データフローマシンがプログラムメモリとの入出力機構をもち，ＯＳも備えている場合であるが，実験機データフローマシンがこれらのものを持たない場合には，ホスト計算機により，ロードおよびスタートトークンの発生等を代行してもらう必要がある。この場合には，ホスト計算機とデータフローマシンの各AMUとの間に専用パスがプログラムロード用に必要であり，また，ホスト計算機からDnet へスタートトークンを転送するためのパスも必要となる。

—参考文献—

1） Arvind , V. Kathail , "A Multiple Processor Dataflow Machine that Supports generalized procedures , "MIT, LCS, CSG memo 205, 1981

2） T. Shimada, "The Structure and Instruction Set of a Simulated Tagged − Token Data Flow Machine, "MIT, LCS, CSG memo 1981

4.8.2 データフローマシンのOS

データフローマシンのＯＳの目的は，

① 複数資源の有効利用

② 入出力等のサービス

③ ファイルシステム，ＴＳＳ，バッチ，リアルタイム等のプログラミングおよび実行環境の提供

④ システム管理・維持

等があり，そのための基本機能として次のようなものが必要とされる。

① 資源管理

AMU，SMU，IOU等のハードウェア資源の管理，およびプログラム，データ，各種管理表等のソフトウェア資源の管理

② 資源割り付けとそのスケジューリング

ＴＳＳ，，バッチ，リアルタイム等，それぞれの性質に応じたプライオリティを与えて制御をする必要がある。

③ 割り込み制御

割り込みとしては，Ｉ／Ｏ割り込み，障害の警告等がある。

④ マルチユーザ環境の管理

⑤ 自己診断

⑥ 統計情報の収集

⑦ 故障からの回復

データフローマシンは，マルチ構成であるため，従来のＯＳで困難であったもので，容易に対処できるようになった問題がある。これらは，

① 負荷の変動に対する追従性が良く，一定したパフォーマンスがあること。

② 重要な処理を実行するのに最小限必要な機能を維持できる。例えば，リアルタイムジョブなど

③ データ駆動原理にもとづくため，同期処理のためのオーバヘッドが不要

また，逆にデータフロー故に困難になった問題もある。それらは，

① 障害時に実行中だったプログラムの回復処理

② マルチタスク環境下での，指定したタスクのみの瞬間時な停止

③ プログラムのトレース

等が考えられる。

ＯＳは，システムのハードウェア，ソフトウェア資源を管理するために，いくつかの管理表をもつ。それらの中の代表的なものを以下に述べる。

(1) 構成管理表

使用可能なユニットやその構成を記録している。この表により種々の構成のもとでシステムを稼動させることができる。

(2) コードブロック管理表

AMUにロードされている手続きやループボディを管理しており，スケジューラが用いる。名前，それが属するプログラム名，ユーザ名，アドレス等を記録している。

(3) カラー管理表

各カラーの使用中，あるいは，未使用等の記録

(4) ユーザ管理表

現在，システムのサービスを受けているユーザ名とそれぞれに割り付けられている資源の記録管理表は，システム管理の上で重要なものであるので，2重化するなどして，障害に耐える必要がある。また，複数のOSが分担して資源を管理する場合には，カラー管理表やコードブロック管理表は分割して管理される。

OSの構成方法は，従来のマルチプロセッサ用OSの構成方法と共通する問題が多いと思われる。以下に代表的な構成法を示す。

(1) マスタ・スレーブ方式

1つのシステム全体の制御を司るマスタPUと複数のスレーブPUよりシステムが構成され，スレーブPUでのみユーザジョブが処理される。マスタPUに指定されたPUは固定される。マスタPUのみが管理表にアクセスするため管理表に対するアクセス競合がない。ソフトウェアは簡単になるが，柔軟性に欠け，また，マスタPUの処理速度により，そこがボトルネックになる可能性がある。

(2) 複数OS方式

PUごとにほぼ完全なOSをもつ方式である。PU内で生じたOSへの要求は，その中のOSで処理する。各OSが管理表をもつため，管理表へのアクセス競合は少なく，共通にもつべき管理表も少ない。論理的にPUが閉じているため，プログラムの割り付け方に柔軟性がなく，負荷が不均一になり易い。また，OSが占有するメモリ量も多い。

(3) (1)と(2)の中間方式

システムを複数のPU群に分割し，各PU群ごとに，その群を管理するほぼ完全なOSを持ちそれを1つのPUに割り付けマスタPUとする。(1)，(2)の長所・短所をそれぞれ引きつぐ。

(4) 分散・共有型OS

OSとPUの対応は可変であり，OSはダイナミックにPU間を移動できる。システム管理表

をもち，スケジューリングを司るＯＳの核をもつＰＵは管理表へのアクセス競合を避けるため一時に一台しか作らない。タスク実行に密接に関連したＯＳ機能などは，そのコピーをそれぞれのＰＵが持ち，自分で処理する。ＯＳがダイナミックに移動できること，および一部機能のコピーが生成できることにより，負荷の平等化が図れ，資源利用効率が向上する。逆に，同一機能モジュールの複数コピーが存在することから管理表へのアクセスの際にロック制御が必要となり，また，デッドロックの危険がありその回避のためのオーバヘッドが必要となる。

(5) オブジェクト指向ＯＳ

ＯＳを共通部分のない独立な機能単位に分割し，ＰＵに分散配置する。この配置はダイナミックに変更できる。すなわち，各機能単位はＰＵ間を移動できる。メモリアクセス競合をさけるため機能単位コード共有，コピーの生成は行わず，個々の単位をオブジェクトとして扱い，要求をメッセージとしてこれに伝え，処理，実行はこのオブジェクトが一括して行う。ユーザプログラムとＯＳ機能単位，およびＯＳ機能単位間でのメッセージ交換がある。オブジェクト化により，ユーザプログラムが直接管理表にアクセスすることがなくなり，ロック制御も不要となる。データフローマシンではないが，Ｃｍ＊のＯＳ，MEDUSA がこの方式である。データフローマシンの場合，メッセージ交換機能はデータ駆動の考え方の中に備わっていると考えられ，また，オブジェクトに関しては，ＩｄのEntry，Exit 等の命令で作られたマネージャ等はオブジェクトと考えることができる。従って，この方式のＯＳはデータフローマシンでも可能である。

―参考文献―

1） Arvind, K. P. Gostelow, W. Plouffe, "Indeterminacy, Monitors and Dataflow", Proc. of Sixth ACM Symposium on Operating Systems Principles, 1977

2） A. J. Catto, J. R. Gurd, "Resource Management in Dataflow, "Proc. of 1981 Conferennce on Functional Programming and Computer Architecture

3） P. H. Enslow. Jr.. "マルチプロセッサと並列処理，"近代科学社

4.9 信頼性処理

データフローマシンの信頼性に関する検討は，まだあまりためされていない。データフローマシンが，従来型のマシンより本質的に有効なものであるか否かの諸検討の影になってしまい，そこまで検討が進んでいない現状である。しかし，データフローマシンを，従来のマシンに置き代わるものとして，その実現化を考えると，当然のことながら信頼性の点で問題点が生じて来る。そこで，ここでは，データフローマシンの信頼性について考える。

データフローマシンの信頼性を考える上で，ポイントとなることは，データフローマシンの構成上の特徴である。即ち，単体での性能はあまり高くない素子あるいはユニットを数種類設定し，それらを多数使用して並列に動作させることによって高性能なシステムを作り上げることである。この特徴は，信頼性を考える時のメリットにもなり，デメリットにもなり得る。このシステム構成上の特徴を生かした信頼化手法が必要である。

これらの特徴を信頼化ということと結びつけて考えてみると，大略，次のことがいえる。

・故障検出のためには，データフローマシン内に存在する多数の構成要素を有効に使ったチェック方式が考えられる。

・リカバリについては，データフローマシンが並列動作を基本としていることから，相当難しいと思われる。

・再構成は，同一機能の代替要素が多数あるので比較的融通性のある方式が考えられる。

以下，各々について現状を見てみよう。

4.9.1 冗長化方式

信頼化のための冗長化設計方式には，大きく分けて次の2通りがある。

A．スタティック方式

故障が生じても，それをマスクするようにハードウェアを構成しておく方式で，そのためには非常に多くのハードウェアが必要になる。

B．ダイナミック方式

基本的には，故障検出機能だけを用意しておき，故障を検出した場合，診断，修復および部分的な再実行によって結果的に故障をマスクする方式。

具体的に検討されている冗長化方式としては次のものがある。

a．Misunas 方式〔MISUNAS76〕

スタティックアプローチである。MIT Dennis のエレメンタリーデータフローマシンにつ

いての冗長設計方式であり，ストラクチャメモリは考慮していない。

計算の複数のコピーを同時に実行して，その結果を多数決処理する。結果の比較（即ち多数決）はプロセッサの命令セルで行われる。これは命令レベルのＴＭＲ（Triple Modular Redundancy ）に相当するので，各セルはそのオペランドレジスタに向けられた3つの結果値を受信出来るように構成する。これら3つの結果値は，その前の命令の3つのコピーの各々から送られるものである。

b．猪瀬方式〔ASADA80〕

ダイナミックアプローチである。ソフト指向の考え方であり，normal なプログラムの他に，それの shadowプログラムを実行させ，両者を比較して故障検出を行う方式。バックアップ系，障害モニタ系，診断系から成り，部分プログラムの入力に対して診断復旧のためにバックアップをとっておく。

c．Leung方式〔LEUNG80〕

ダイナミックアプローチである。Dennis のデータフローマシンに対する冗長設計方式であり，ノード間でack信号を使うことを前提としている。故障に出会ったパケットは，行く先ＰＥに送られる時にその旨の印が付加される。ＰＥは，印の付いたパケットを受信するとホストに通知する。ホストは，そこで，他のＰＥに連絡してネットワークにパケットを送ることを止めさせる。その結果，全体が止まり，そこでホストが修理し，再開させる。エラーパケットを受信したＰＥは，ack信号の代りに再送要求を送り主に返信する。復旧のための計算の途中結果はack信号受信まで保持しておく。

以下，信頼化の要素技術である故障検出，復旧，再構成について述べる。

4.9.2　故障検出

故障検出のために，ハードウェア的多重化と，ソフトウェア的多重化が考えられる。しかし，元々，データフローマシンが多数の同一機能要素を使ったシステムである事から考えて，その特徴を生かした検出方法が得策と思われる。即ち，ソフトウェア的多重化方式である。これは，プログラムの複数のコピーを作り，それらを並列に実行させながらチェックして行く方式である。データフローマシン内には，多数の同一機能要素があり，かつ，それらが必ずしも１００％稼動状態にある訳ではないと予想されるので，リソースの有効利用という点からも望ましい。

上記プログラム冗長方式のための冗長コード生成はコンパイラが作り出すものと考える。

A．検出方式

故障検出方式として，次の２種類が考えられる。

・ノード機能拡張方式

・特殊ノード設定方式

また，多重化の数は，スタティックかダイナミックかの選択と関連する。なお，ここでは，検出を命令レベルで行うものと考える。

a．ノード機能拡張方式

ノードの機能として，多重のオペランドを受信し，チェックする機能を持たせる。

(1) プログラム3重化

Misunas 方式で採用しているもので，図4.9.1に，その一例を示す。

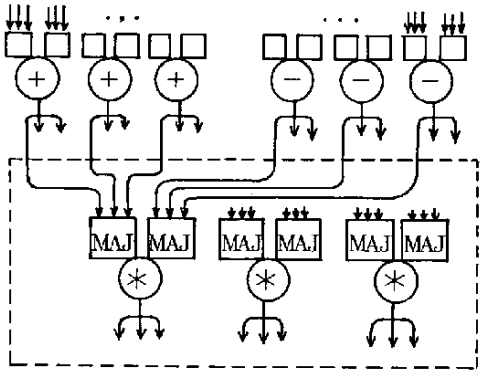


図4.9.1　プログラム3重化

（ノード機能拡張方式）

各ノードでは，実行前に入口で多数決を取る必要があるので，オペランド当たり3個のレジスタが必要となる。スタティック向。

(2) プログラム2重化

Misunas 方式から類推されるもので，図4.9.2に，その一例を示す。

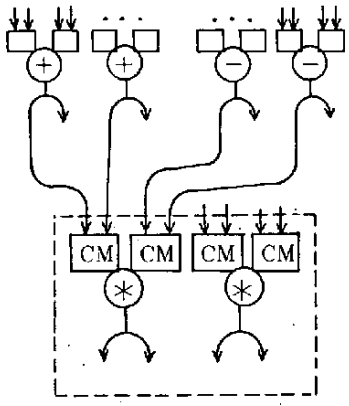


図4.9.2　プログラム2重化
（ノード機能拡張方式）

各ノードでは，実行前に一致検査をする。ダイナミック向。

b．特殊ノード設定方式

一般のノードの機能を拡張せずに，チェックノードとして多数決あるいは一致検査を行う特殊なノードを設ける。この場合も，3重化と2重化が考えられる。

(1)　プログラム3重化

猪瀬方式の変形として考えられるもので，オペランド・レジスタの削減が計れる。一例を図4.9.3に示す。スタティック向。

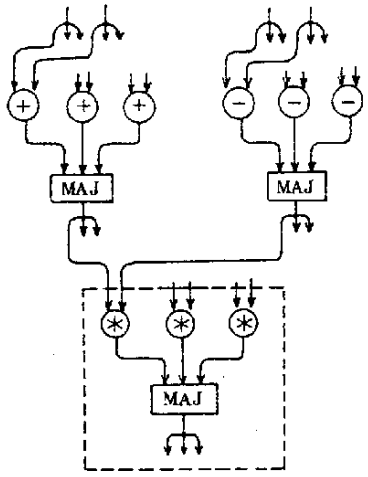


図4.9.3　プログラム3重化（特殊ノード設定方式）

ＭＡＪノードにおいて多数決がとられる。

(2) プログラム２重化

猪瀬方式で採用されているやり方であり，一例を図4.9.4に示す。

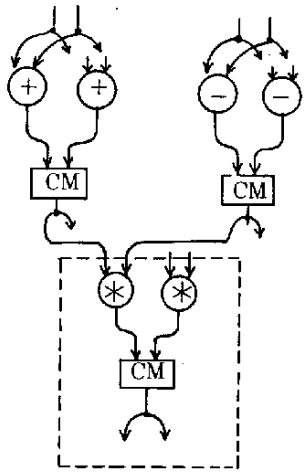


図4.9.4　プログラム２重化

（特殊ノード設定方式）

ＣＭノードにおいて一致検査を行う。最も単純であり，ハードウェア量，通信量の点で好ましい。

これらの方式は，純粋のハードウェアによる多重化方式に比べて一般に次のような融通性がある所が利点と思われる。

・完全同期処理をする必要がない

コピープログラムの実行を時間的に同時にする必要がない。例えば，図4.9.4で２つの⊕オペレーションを時間的にずれて行なっても一向に構わない。

・多重化の相手が固定的でない

コピープログラムの実行をするハードウェアは，どの実行ユニットによって実行されるかを固定にする必要がない。プログラムの割り付けによって自由に変えられる。

これらの冗長設計方式での通信量を考えてみると，冗長設計をしない方式に比べた場合の各方式でのそれは表4.9.6のようになる。

表 4.9.6　各方式での通信量

（冗長設計なしと比較した値）

ノード機能拡張方式		特殊ノード設定方式	
3 重化	2 重化	3 重化	2 重化
9　倍	4　倍	6　倍	4　倍

データフローマシンの問題点の 1 つが処理要素間での通信オーバーヘッドであることを考えると，通信量が大幅に増加するような検出方式は望ましくない。プログラム 2 重化程度が適当ではないかと考える。

B．検出レベル

故障検出のレベルは，冗長プログラム作成のやり方に対応して，次のような段階が考えられる。

・プログラム全体を単位とする

・サブプログラムを単位とする

・命令を単位とする

検出レベルは，故障場所の位置決め精度と，オーバヘッド（ハード量，実行時間）とのトレードオフとなる。例えば命令レベルで検出を行えるようにすると，故障が生じた場合に直ぐに検出可能であり，その命令位置も判明するが，命令毎にチェックノードが必要になる訳だから，実行時間や通信量，所要メモリ量がその分増加することになり，冗長化しない場合に比べて多大のオーバヘッドを課することになる。逆にプログラム全体を単位とするような冗長レベルにすると，チェックノードは少なくて済むだけ故障検出時間の遅れが生じ，また故障箇所の位置決めにも難しさが生じて来る。

これら両者の中庸をとり，サブプログラム単位にするのが現実的と思われる。サブプログラムの大きさをどのように選択するかも検討を要する項目であり，再試行のことと考え合わせてみる必要がある。サブプログラムの例を図 4.9.5 に示す。

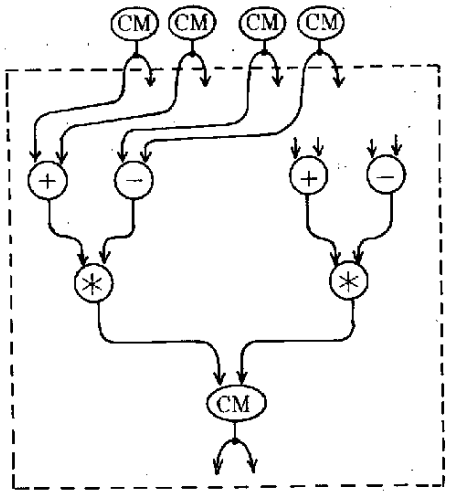


図 4.9.5　サブプログラムを単位とする故障検出

C．検出制御

2重化方式の場合には，検出の時機と，データの流れとの関連で，検証または非検証の選択が考えられる。

検証方式とは，比較するべき対象が双方とも到着し，一致したという結果が判明してから次の命令へデータを通すやり方である。原理的に確実であり，エラーが生じたら直ちにそのための処置を開始することが出来る。その代償として，正しい処理をしている場合でも常にチェックノードの判定待ちで全体の流れが遅れてしまう。3重化方式は，必然的に，この検証方式に属することになる。

一方，非検証方式は，チェックノードの判定は本来のデータの流れとは無関係に実行させておき，エラーが生じた時に初めてエラー処理を始めるものである。実行速度の向上が図れるが，生じたエラーの解析や復旧が難しくなると思われる。例えば，エラーが生じているにも拘らず，誤ったデータを使って先の処理まで進んでしまっている状態が起こり得るからである。

また，別の問題として，チェックに関する信頼性も考えなければならない。

特殊ノード設定方式では，例えば図4.9.3を見ると，⊕および⊖等のノードは3重化されているが，それらの多数決をとるＭＡＪは一個だけである。従ってＭＡＪのエラーについては何の効果もあらわさない。

これに対して，ノード機能拡張方式では，図4.9.1を見て分かるように，各ノードに多数決機

能が盛り込まれているので，そこでのエラーは，次の段のノードで検出されることになる。

以上のように，検出の信頼性に関しては，ハードウェア量を豊富に使うノード機能拡張方式が優れているといえそうであるが，ハードウェア量が増加する分だけ故障率も多くなるので単純には結論出来ない。図4.9.4のCMを2重化する案も提案されており，特殊ノード設定方式をもう一歩進めた形が適当ではないかと思われる。

D．プログラム割付

先に述べたように，冗長化プログラムはコンパイラによって生成されるが，その冗長化プログラムをハードウェアにマッピングする際，エラーを適確に検出出来るようにハードウェアの構成に応じてソフトウェア上の考慮をしておかなければならない。

即ち，2重化あるいは3重化にしても，コピーされたプログラムの対応する命令が物理的に同一のハードウェアで実行されることがあれば多重化の意味が薄れてくる。何故なら，もしも故障が生じたとしても，多重化されたそれぞれの命令出力に同じ形のエラーが生じるので結果としてエラーが認知されない事態になる。従って，コピーされたそれぞれの命令は，実行されるハードも，ネットワークを通る場合のパスも出来るだけ異なるようにソフトウェア上で考慮して割り付ける必要がある。そのための割り付けアルゴリズムをどのようにしたらよいかを検討する必要がある。現在までに若干の検討が為されており〔ASADA80〕〔IKEDA80〕その際の問題点として，実行時間のばらつき，複数プログラムの扱い方等が指摘されている。

4.9.3 リカバリ

エラーが検出された後，診断，修復，再試行をして正常動作に復旧する必要があるが，この点に関しては，現在までほとんど研究がなされていない。診断，修復は，現在までの汎用計算機で培った技術が使えると考えられるが，特に，再試行については問題が多いと思われるのでその点を中心に考える。

概念的には，データフローマシンが関数型言語を実行するマシンであるから，従来のマシンでのリカバリよりも考え易い。それは，メモリに対して副作用をすることによって処理を進めるのではなく，独立した命令同志が値を授受しつつ処理を進めることから予想されることである。従って，個々の命令レベルで考えれば，命令への入力値をセーブしておき，自命令および次の命令で正常に動作したことが確認されればセーブ情報を破棄し，異常であれば，セーブ情報を使って再試行が可能であり，実行の過去を大幅に遡らなくても済みそうである。しかし，データフローマシン内では非常に多くの命令が同時に実行されていること，関数型言語実行マシンといえども，履歴依存操作もあり得ること，遅延評価との関連等で非常に難しい問題である。

A．再試行のレベル

再試行のレベルは，エラー検出のレベルと同様に3段階考えられる。エラー検出のレベルと必ずしも対応させて考える必要はない。

このレベルは，ハードウェア量に直接影響する。データフローマシンでは並列に多数の命令が実行されているので，もし命令レベルでの再試行をしようとすれば，並列度に見合うだけの情報をセーブしておかねばならない。ただし，手法としては最もやり易いと思われる。MIT Dennis の ack 信号を再試行のための手段に利用すれば考え方としてはスマートである。ただし，ack 信号を利用した場合の欠点については，4.3 で述べられている通りである。

実現の可能性から見ると，サブプログラム単位あるいはプログラム単位で再試行するのが妥当と思われる。サブプログラム単位にする時には，サブプログラムの選び方が問題である。論理的な分割および物理的な分割を考え合わせるべきである。例えば，論理的には関数単位にするとか命令数で一定にするとか，並列度でスレッショールドを設けて分けるとかが考えられる。一方，物理的には1つのＰＵ内に納まる単位に限定するような工夫が必要と思われる。これは，セーブすべき情報をどこに蓄えるかを考えると必要な制限であろう。サブプログラム単位の例には，Rumbaugh マシン，プログラム単位の例にはＴＩマシンがあるが詳細は不明。

B．再試行方式

再試行を可能とするためには，先ず第1に処理途中の結果をセーブする必要があることである。セーブ情報に関しては何時セーブするかが問題である。これは予めコンパイラが，セーブの必要があると判断した命令（あるいはオペランド）の中にその旨のフラグをつけておき，ハードウェア側ではそのフラグを検出した時にセーブする方法が考えられる。セーブ情報をどこに蓄えるかも検討を要する。個々のプロセッサ毎にセーブ用メモリを用意し，自プロセッサ内の命令に関する情報を蓄えるのが適当であろうか。さらにセーブしておく情報には何が必要かも考えなければならない。再試行のレベルによっても異なるかも知れないが，命令，オペランド値，その他環境情報が必要であろう。構造体に関する操作命令の場合のセーブ情報は特に遅延評価機構を導入している時には，よく検討する必要があろう。

エラーを検出した際に，何に伝えるかを考えると，エラーハンドラー（またはホスト）が想定される。エラーハンドラーは，すでにセーブされているセーブ情報を取り出せるようになっていて，その情報を命令メモリやオペランドメモリにリストア出来る機構になっている必要がある。

また，これらの一連の処理をする場合に，実行中の命令を一時停止させる必要があると考えられその手法についても検討することが必要である。

4.9.4 再構成

データフローマシンは，そのハードウェア構成から考えて，再構成については従来のマシンよりも融通性が出て来ると思われる。

処理ユニット部は，ＰＵのレベルで見てもＰＵ内のＥＸＵのレベルで見ても，代替品が数多くあるので，故障ユニットを切り離しても性能の劣化は微小で済ませることが可能であろう。また，エラー検出の能力も，前述のプログラム冗長化法を採用するならば，再構成によって変わることはほとんどないと思って良いだろう。問題は，命令アドレスに関することであろう。再構成されることによって，プロセッサが一個切り離されたとすると，それまでそのプロセッサをアドレスしていた他のプロセッサ内の命令の行先を変更しなければならないことになる。これは，関数のＣＡＬＬ／ＲＥＴＵＲＮ だけでプロセッサ間のインタフェースが行われていれば比較的考え易いが，１つの関数が多数のプロセッサ間にわたって配置されている場合には，再配置が簡単には行かないであろう。

一方，ＰＵ間あるいはＳＭＵ間等を結ぶネットワークの再構成についてはほとんど研究がなされていない。ネットワークの高信頼化の手法は従来技術を延長して考えられると思うが，データフローマシンの心臓部でもあり，ここがうまく機能しなければ多くのＰＵが無駄になる可能性もあるので，今後の検討を要する。

－参考文献－

1）〔MISUNAS76〕Misunas，D. P.，"Error Detection and Recovery in a Data－Flow Computer" Proc. of Int'l Conf. on Parallel Processing Aug. 1976

2）〔ASADA80〕浅田邦博"計算機複合体における故障診断修復方式に関する研究"東京大学工学部電子工学科　博士論文

3）〔LEUNG80〕Leung，C. K. C.，Dennis，J.，"Desigh of a Fault-tolerant Packet Communication Computer Architecture "FTC-10，Oct. 1980

4）〔IKEDA80〕池田正幸"分散型データフロー計算機におけるアクタの割り付け"東京大学部電子工学科　卒業論文

4.10　デバッグ機能

データフローマシンにおいて，デバッグというテーマは最も難しいものになりそうである。

難しさの原因は，もともと，データフローマシンの特徴そのものに帰着するようである。即ち，同時に並列に沢山の命令が実行されること，およびその実行の順序が決まっていないことに由来する。これ故に，デバッグする場合に，従来マシンのそれに比較して難しくなり，従来のデバッグツールが使えないのではないかといわれている。ソフトウェアのデバッグは勿論，ハードウェア開発中のデバックにおいても相当の工夫が必要と思われる。このテーマも未検討の領域であり，今後の研究が期待される。

4.10.1　ソフトウェアデバッグ用の機能

データフローマシンにおいてソフトウェア（プログラム）のデバッグをする時の問題点を必要と思われる機能と関連させて考える。

データフローマシンは，並列実行が基本となるので，デバッグツールにも並列性を表現し易い2次元の図形を駆使するアイデアが有用になるのではないかと考える。

A．トレース

プログラムをトレースする場合に，一般には，プログラムの最初からどこまでが，思い通りに実行されたかを調べるであろう。所が，データフローマシンの場合には，プログラマが頭の中で予想する実行順序と，マシン上での実際の実行順序（あるいは表示順序も）が異なると考えられる。更に，やっかいと思われるのは，実行順序に再現性がないと考えられるからである。単一のプログラムが走行している場合は，再現性も期待出来るが，複数のプログラムが実行されている場合は到底，実行順序の再現性は期待出来ない。

従って，トレースを少なくとも従来マシン上で試みるのと同程度にするには，実行順序の指定が，ある程度ユーザから制御出来るような機能が必要であろう。もし，それが実現出来ない時は少なくとも再現性のある順序で実行出来るモードが欲しい。

関数型言語用のマシンであるから，トレースをLISPのように関数単位で行うことになるかも知れないが，当然，命令単位でも行える機構が欲しい。

B．ブレーク

指定した命令を実行した直後に，そのプログラムの実行を停止させることが必要であるが，そのためには，その命令を含むプログラム内の総ての命令を停めなければならない。しかし，注目する命令の実行と並列して実行されている命令もあるのでこれは不可能であろう。見方を変える

と，ブレークする時は，その命令に関係する前後の命令が関心の的になると思われ，注目する命令とデータ依存の関係にない命令については直ぐに停める必要がないともいえる。そのようなゆっくりとした停止でも良いと考えれば，停める方法はあるだろう。例えば，タグ付トークンを使用したシステムならば，このタグに含まれているであろうユーザ識別番号，プロセス番号をキーにして，それと合致する命令を発火させない機構にすることは可能であろう。

また，ブレークして，その時に何を見るかも問題である。多分，その命令を実行する前のオペランド値と実行後の結果値を出力することは必要と思われる。

C．ダンプ

デバッグのための手法としては非常に原始的であるが，最後の手段として用意しておく必要があるだろう。ダンプする内容も問題であり，まず考えられるのはアクティビティの内容（即ち，命令種類，オペランド値，行先番地等）である。また，マシン内のあちこちに分散している可能性のあるトークンもダンプ出来る必要がある。ただし，これは，マシンの停め方にも依存するので，トークンのダンプは不要となるかも知れない。

D．オペランド値の変更

ブレークした状態で，特定の命令のオペランド値を変更する機能も必要であろう。そのためには，すでにトークンを受信して確定しているオペランド値をクリアしたり，そこへ新しく値を入れたりするための所謂特権トークン，擬似トークンが要るかも知れない。

E．シングルステップ制御

ステップ的に実行を進める方法も必要である。データフローマシンの場合には，次に実行すべき命令は複数個存在するのが普通なので，従来のシングルステップ制御より変形が考えられる。例えば，従来と同様に，一時には一個の命令しか起動しないで逐次に進めて行く方法。 あるいは，並列に実行出来るものは同時に，それぞれのパス当たり一個のアクティビティを起動させる方法も考えられる。後者の方法はデバッグ時間の短縮に寄与する可能性もあり，２次元の表示手段が効果を発揮すると思われる。

F．例外処理

演算結果のオーバフロー，アンダフロー等をデバッグがし易いように考慮したシステムもある。ＭＩＴのマシンではＯＶＦやＵＤＦが生じてもプログラムエラーとせずに，特殊な値と看做して実行を進めて行く〔ACKERMAN79〕。エラーが生じた時は，停めるのが難しいので，この方式のように最後まで実行させるのも１つの方法であろう。最後に出力される結果に，OVF やUDF を最初に生じた命令のアドレス等が含まれていれば，デバッグの一手段として有効であろう。ただし，一般に，そのまま実行を続けても無意味なことが多いと思われるし，他のプログラムの実

行のさまたげにもなるので，早期にストップさせる方法が好ましいであろう。

4.10.2 ハードウェアデバッグ用の機能

ハードのデバッグにおいては，一般に，並列に動作する部分は非常に難しくなる。データフローマシンでは，並列に動作する要素の数は従来マシンに比べて桁違いであるから，ハードのデバッグにも是非新しい手法が望まれる。地味な作業ではあるが，マシンを早期に実現する上で是非検討すべき課題である。

A．ステップ制御

命令単位で実行を逐次進めて行ける機構が必要である。更に，それをプロセッサ毎に制御出来る機構や，環状のパイプラインをユニットの出入口で塞き止める等の機能も必要であろう。

B．リソース制限

リソースを少なくすることで，バグの発見を容易にすることも考えられる。勿論，その場合にプログラムの変更はシステム側でサポートすることが望ましい。

C．ヒストリ

ハードデバッグのためにヒストリは大きな効果がある。従来マシンに比べて，並列度の違いの分に相当する位の多くのヒストリをとっておく必要があるのではないか。ユニット毎に出入口に設けること，あるいは，特定の命令以降のヒストリをとること等が考えられる。

—参考文献—

1） 〔ACKERMAN79〕 Ackerman, W. B. , Dennis, J. B., "VAL--A Value-Oriented Algorithmic Language ： Preliminary Reference Manual". MIT/LCS /TR-218 June 1979

5. ＤＦＭ実験機のイメージ

5.1 実験機第１版のイメージ

5.1.1 実験機第１版の位置付け

データフローマシンは多数個の資源（演算ユニット，実行制御ユニット，構造メモリユニット等）を結合し，これらを並行動作させることによって高性能を得ることが方式上の狙いである。そのための研究開発の進め方としては，各ユニットの実現方式を明確にすると同時に現在あるいは将来の素子技術をパラメタ条件として各ユニットの性能をどこまで引き出せるかを明確にする要素ユニットの研究開発と，これら多数個のユニットの実行をどのようにスケジュールし，制御するかを明確にするシステム研究の２方向が必要である。データフローマシンが情報処理システムとして十分機能するためには，入出力機能を持ち，高級言語によるプログラミングをサポートすることが出来なければならない。

理想的には，高性能要素ユニットの構成法の研究と，資源割付，スケジュール等のシステム研究を同時並行させることが好ましいが，人員，予算等の現実面からの制約もあり，研究，開発の重点をどちらかに絞ることが必要である。

本プロジェクトで開発するデータフローマシンは推論，パターンマッチ等の機能を実現するものでなければならない。しかしこれらの機能仕様は現時点では必ずしも明確ではない。前期３年間の研究開発で先ず明らかにすべきことは，各ユニットのハードウェアが実現可能か，データフローマシンの方式で必然的となる各ユニット間のパケット転送は実装上どのような影響をもたらすか，各ユニットへのプログラムやデータの割り付けはどのようにスケジュールすれば良いか，デッドロック問題はどうあるべきなのか，などであり，データフロー原理にもとづく計算機方式が，従来のノイマン方式にとって代れるだけの実用化可能性を本当に秘めているのか，ということである。

この目的のために実験機第１版では，各要素ユニットについては性能よりも実現の可能性を明確にする程度に留め，開発・実験の重点をシステム上の妥当性確認，資源スケジュール等のＯＳおよび言語などソフトウェア方式を確立することにおく。

実験機の設計・試作は以下の方針で進めていく。

(1) 実験機はある程度のソフトウェアベンチマークテストが可能となるようなソフトウェアプログラミング環境を提供するものとする。プログラミングは関数型セマンティクスにもとづく高級言語を用いて行えるものとする。このため，関数型高級言語ならびにその処理系を開発する。ベンチマークテストとして例えばProlog インタプリタの作成・評価などを行うものとする。

(2) 実験機のハードウェアとしては，実行制御ユニット，演算ユニット，構造メモリユニットを各々８～１６台程度用い，これらをパケット転送用ネットワークで結合した構成をとる。各モジュールの処理速度（スループット）は２μsec 程度とする。

(3) 本実験機は記号処理専用のアーキテクチャとする。このため，特に構能メモリユニットを設計・試作し，リスト等の構造データの処理を高速化するような構成とする。

(4) Ｉ／Ｏ，ファイル処理のためのディスプレイ端末，プリンタ，および２次記憶装置の制御は全てホストコンピュータを通して行うものとする。

(5) 各要素モジュールは，ビットスライス型のマイクロプログラム可能な素子を用いて構成し，その機能実現はファームウェアによるものとする。このファームウェア機能を用いることにより，Unification等の推論機構用演算モジュールを実験的に構成できるものとする。

5.1.2 データフローマシン用高級言語

データフローマシン実験機第１版の高級言語仕様を検討するに先立ち，次のことを考慮しておく必要がある。データフローマシン実験機作成の目的は，単にデータフローマシンとしての動作確認や性能評価を行うためではなく，知識情報処理における推論マシンとしてデータフローマシンが適しているか否か，また推論マシンとするにはどのような機能や制御機構が必要とされるかを解明することにある。実験機第１版は推論マシンへの橋渡しとしてデータフローマシンの原理的問題を解明することにあり，数値処理よりも記号処理への応用に比重を置いている。ここでは，浮動小数点演算や配列データ処理などの数値計算を高速化するような特別な配慮は行わない。

実験機第１版のデータフロー高級言語が知識情報処理の応用分野で使用される言語へと発展していく，あるいはつながりを持つことを念頭に，とりあえず汎用なものを想定しそれに記号処理用オペレーションやサポート機能を付加していく方針をとる。

高級言語の設定の際には，4.2節におけるデータフローマシン用高級言語の考察を踏まえて，書き易さ，読み易さ，記述，検証能力，並列処理の記述性などの種々の観点から検討を行う必要がある。主な検討項目としては，

- 基本データ型や構造型などのデータ形式
- オペレーションの種別
- 制御構造，プログラム構造
- プログラムスタイル
- シンタクス
- セマンティクス

・計算モデル

・抽象データ型

・並列処理記述能力

・言語処理系

などがあげられる。以下，主な項目について指針を述べることにする。

A．プログラムスタイル

データフローマシン用高級言語のスタイルを関数型とするか論理型とするかという大きな選択が存在する。関数型プログラミングについては，Pure Lispという先達が存在し，計算モデルの研究もかなり行われている。またデータフローマシンとの親和性の解明もある程度なされてきた。これに対し，論理的，プログラミングは知識情報処理の観点から有用と思われるが，プログラミングの経験も浅く十分な解明がなされていない状況にある。今後，このスタイルでのプログラム経験を積むと共に理論面の検討を行っていく必要がある。以上のことを勘案し，第1版の高級言語では関数型スタイルを提供する。この際，Lisp そのものをデータフロー高級言語とする考え方も成立つが，Lisp のシンタクスにこだわらなくとも関数型言語を提供できる，実験用として各種データ型を取り入れたい，データフロー特有の feature を検討する必要があることを考慮し，シンタクスは従来型（ただし，ブロック構造を持つAlgo 風のもの），セマンティクスは関数型という方向で検討を進める。

B．データ形式

データフローマシン実験機が記号処理を目指したものであるから，取り扱うデータもリスト型が主体となる。この場合，Backus のＦＰやLisp のように扱うデータ型は唯一つ（オブジェクトまたはリスト）とすることも考えられる。これについては今後検討を行っていく必要があろう。ここでは，一応汎用のものにしておくため，各種のデータ構造を提供する。データ型として integer，real，Boolean，character，signal などの基本カラー型，vector，array，list，string，set などの構造型を候補とする。

C．オペレーション

基本オペレーションとしてどのようなものを提供するかはマシンの性格を大きく左右する。記号処理を主眼にしているので，数値計算については加減乗除程度の基本的命令を用意するにとどめ，ベクトルや配列処理用の高機能化命令セットは特に提供しない。

詳しくは「基本命令セット」の検討を待つが以下のオペレーションをここではとりあげる。

・基本数値オペレーション

加減乗除等の四則演算，最大・最小等の組み込み関数

・基本論理・関係オペレーション

・基本構造オペレーション

－基本リスト演算，Append 等の拡張演算

－基本集合演算（集合の和・差等）

－基本ストリング演算

・入出力オペレーション

リスト，ベクトル，配列，スカラー値の入出力

この他，実行時のエラー処理についても，今後検討を行っていく必要がある。データフロー制御のための基本オペレーション（ゲート，スイッチ，関数リンケージ，ガーベジコレクション等）は基本命令セット（機械語）には存在するが，高級言語のレベルでは，if then else construct や function definition などのprogram structure が提供される。

D．型チェック

検証の立場からは型チェックを行うことが好ましい。型チェックのやり方には，Pascal のように厳格な型宣言にもとづいて行うか，あるいはFortran のように処理系の方が適宜判断して行うか，またこれを静的（コンパイル時）に行うか，動的（実行時）に行うか，などの選択がある。関数型言語として有名なLisp やＦＰには型宣言はない。関数型言語と型宣言をどう融合させていくか（記述のわずらわしさを排除する方向と検証を容易にする方向との調和）を検討していく必要がある。第一版の高級言語ではインプリメンテーション時のデバックを容易にするという観点から，厳格な型宣言を行い，静的および動的にチェックを行う方針をとる。ＶＡＬではoneof（…）というunion type があり，（…）内のいずれかの型をとることを許すので，実行時でないと型チェックができない。このため，実行時に型チェックを行うためのtype check オペレーションが用意されているが，どのようなオペレーションを設けるかエラー処理のやり方も含めて検討する必要がある。

E．抽象データ型

プログラムを容易にする１つの方法として，人間の持つ抽象化能力を活用するという考え方がある。この考えは手続きの抽象化，制御構造の抽象化さらにデータ構造の抽象化へと広がってきている。検証（デバッグ）の容易さという観点からも今後導入を検討していく必要がある。関数型言語ＦＰの中にも抽象データ型を導入しようとする研究が最近行われており，日本では東工大のFLADT の例がある。しかし抽象データ型をどのように取り込むべきかについては，未だ不明な点も多く，第１版高級言語への導入は見合わせることとする。

F．並列処理記述

データフローマシンの特徴は，implicit な並列実行（即ちユーザが意識しなくともプログラムに内在する並列性が引き出される）という点にある。しかし，implicitな並列実行にまかせるだけでは十分な並列性は得られない。ループ表現における parallel構造（ループの各ステージにデータ依存関係がないもの）は，explicit に並列記述を行わせ，処理系の方で並列実行本体を生成するようにする（例えば，ＶＡＬの forall や VALIDでの for each）方が高い並列性が期待される。これは有限資源下での並列性抽出など実用的見地からも有効である。このようなexplicit な並列記述は，ＦＰではapply to all や construction に対応すると考えられるが，並列性をうまく引き出しうるような並列処理記述法の検討をさらに行う必要がある。

G．言語処理系

a．コンパイラ／アセンブラ

・ コンパイラは高級言語をデータフローグラフへ変換する働きをする。この場合，直接データフロー実験機の機械語へ落す方法と，一旦中間言語（記号によるデータフローグラフの線形表現）に翻訳し，それからアセンブラで機械語に落す方法がある。ここでは，各種応用のサポート，移植性などを考慮し，中間言語を設定する方法を採る。応用対応に中間言語の変更を行う場合でも，基本的なデータフロー制御命令は共通である。

・ データフローマシン実験機第一版では，クロスコンパイルを行わざるをえない。データフローマシンが完成したときは，その上でセルフコンパイルする方が望ましいが，その場合処理系の構造はどうあるべきか再検討する必要がある。このためにも，第一版高級言語の仕様が固まった時点で，それを使って処理系自身を記述してみることが必要である。

・ 大量のプログラムを作成する際や，ユーザ定義の関数やマクロをライブラリ化する際には分離コンパイル機能と関数間のリンクバリュー名参照の解決を行うリンケージローダが必要となる。ここでは関数定義毎の分離コンパイルが行えることとする。

b．最適化

従来型言語の処理系で開発されてきた最適化技術（コンパイル時に実行可能なものは先に実行してしまうfolding，冗長なコードの除去，ループ定数の取り出し等）の幾つかはデータフローグラフの最適化にも使用することができる。データフローマシンに特有の問題としては，参照カウント更新，関数のリンケージ・後処理などの最適化を行う必要がある。また，ＦＰの代数法則などを利用してデータフローグラフ自身を簡略化するような最適化規則の検討が今後必要である。

c．入出力処理

履歴を扱う入力および出力処理は関数型言語の難関の1つである。履歴依存性のある計算にはstream概念が有効と思われるが，その長所，短所を十分把握しておく必要がある。実験機第一版の最初の利用形態は，必要なデータを計算する前に読み込んでおき，計算終了後，結果を出力するといった単純な形態となろう。この場合にはファイル概念を必要としない。しかしマルチユーザのサポートなど現在行われているような入出力処理を提供しようとすると，ファイル概念の導入が必要となってくると思われる。streamは non strict なリスト概念であり一種のsequentialファイルと考えられるが，全てこれで解決がつくかどうか不明である。特に配列のような構造体データを扱うにはrandom access 可能なファイル概念を持込む必要があるかもしれない。sequential ファイルへの各アクセス命令はアクティビティ名で識別すると共に，critical section として排他制御を行うこと，randomファイルへのアクセスは順序指定の不要な個々の要素へのアクセス命令に分解すること，など各種方式の検討が必要である。

5.1.3 アーキテクチャ

A．データフローマシンの全体構造と考え方

a．基本方針

データフローマシン実験機第1版を考える上でのポイントは2つある。

1つは，実験機の応用として記号処理を念頭においていることである。これは，データフローマシンが将来の知識情報処理における推論マシンへと発展すると想定する限り自然な前提である。記号処理向きデータフローマシンを考えるとした場合に次のような点を考慮に入れるべきである。

- プログラム内の関数の動的性質を実行前につかむことが難しい
- 命令は，関数呼び出し等の制御命令と構造データ操作に関するものが中心である。
- 動的なデータ構造の効率よい操作が必要である。

もう一つのポイントは実験機として必要な構造上の柔軟性である。データフローマシンの研究は，現在その基本構造が少しずつ明らかになって来た段階であり，実用的な性能を得るには多くの問題を明らかにしていく必要がある。実験機第1版は，残された問題に対する幾つかの有望方式を実装して評価することができるような構造を持つ必要がある。このため，そのハードウェア構造は多少の冗長性を持つこともあり得る。

b．全体構造

データフローマシン実験機第1版の全体構造を図5.1.1に示す。

マシンは大別して3種の機能モジュールと4個のネットワークからなる。

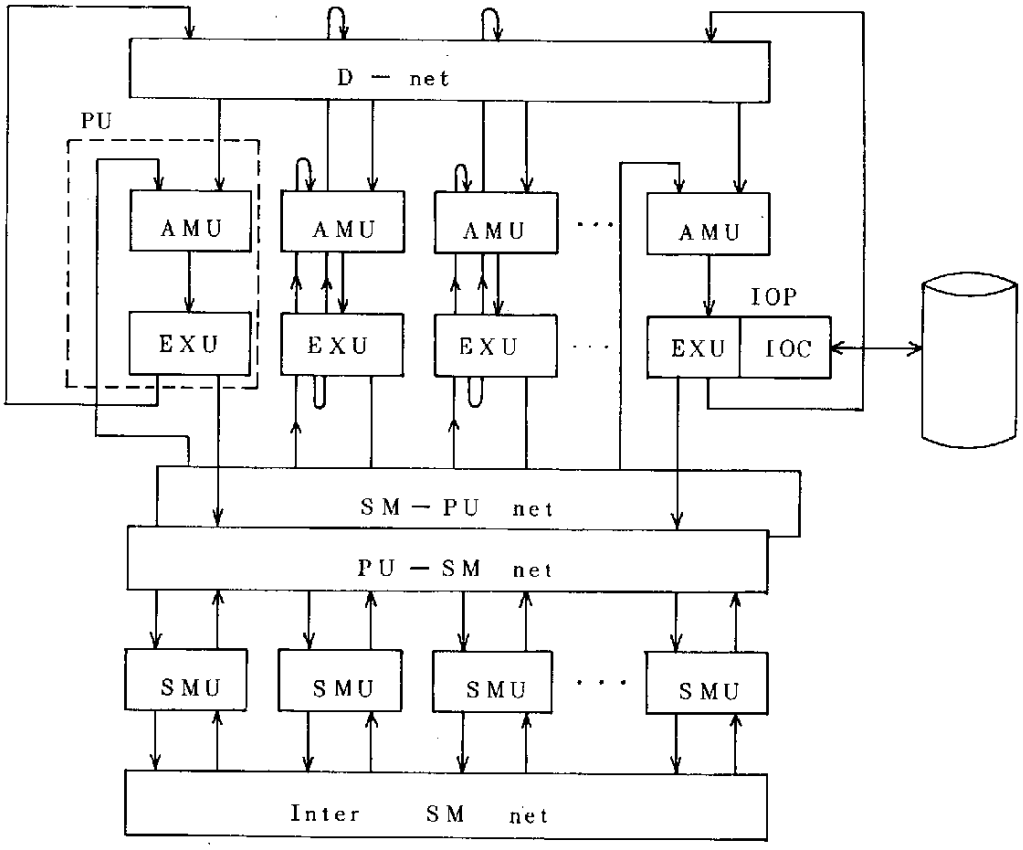


図 5.1.1

機能モジュールは，Activity Memory Unit(AMU)，Execution Unit （EXU）とStructure Memory Unit （SMU)であり，AMUとEXUのペアをProcessing Unit（PU）と呼ぶことにする。ネットワークは，EXUからAMUへトークンを渡すDistribution Network(D-Net)，PUとSMU間を結ぶPU-SM Net，SM-PU Net，およびSMU間を結ぶInter-SM Netである。

c．全体構造についての考え方

一般にデータフローマシンは，並列に置かれた数種の機能モジュールとそれらを結ぶネットワークから成ると考える。機能モジュールとしてはアクティビティメモリユニット（AMU：Activity Memory Unit），演算ユニット（EXU：Execution Unit），構造メモリユ

ット（SMU：Structure Memory Unit）がある。また結合ネットワークとしてAMUからEXUに対してトークンを送るA－net（Arbitration Network），逆にEXUからの結果トークンをAMUに送るD－net（Distribution Network），AMU（またはEXU）とSMU間を結ぶAM－SM net等が考えられる。

データフローマシンは，以上のような要素の性格を決定していくことによりその全体構造が決定されることになる。

ネットワーク構造を決定する要因としては次のものが考えられる。

・プログラムの局所性→D－ net

・演算装置の負荷分散→A－ net

・構造データの局所性→AM－SM net

以下それぞれの場合について述べる。

(1) D－net

D－ net はプログラムの局所性およびプログラムの割り付け方式に依存する。割り付け方式としては，大きく分けて，プログラムの手続き毎にAMUへ割り付ける方式（縦形割り付け）と各手続きを分散してAMUに割り付ける方式（横形割り付け）があると考えられる。横形割り付けは並列性の抽出の点において有利であるが，命令あたりの実行遅延が大きくなる。逆に縦形割り付けではそのオーバヘットが小さくできるが，並列性の抽出，負荷分散などにおいて十分な考慮が必要である。

実験機では各種の割り付け方式を検討することができるような自由度の大きい構造が望ましい。そこでD－net としては局所性の少ないものを考え，さらにEXUとAMUの組に対してローカルなパスを付け加えた構成が好ましいと思われる。

(2) A－net

A－netはAMUから発する命令トークンを複数のEXUに対して分散させるためのものである。しかし本マシンでは次の理由により省くことができると思われる。実験機が対象としている記号処理の応用におけるEXUの役割りは主としてスイッチ，ゲートなどAMUに比べて処理負荷が少ないものである。また各EXUは機能分散の必要がなく同一の構造にできると考えられる。さらにスイッチ，ゲート等の操作は高速に実行されるべきものであり，A－net を取り除くことによってオーバヘッドが小さくできると思われる。

(3) SM－AM net

記号処理の応用ではリスト構造のような動的データ構造の扱いが中心となる。各構造メモリユニット（SMU）に対する構造データの配置は構造メモリ自体の機能に依存するが，リ

スト状の構造データは一般に分散配置されることが多い。また現在のところプログラムの動的特性，構造データに対する参照の局所性について明確な指針がないため，各ＡＭＵとＳＭＵ間には比較的等距離のアクセスパスを設けておく必要があると思われる。

ＡＭＵとＳＭＵ間のネットワークは，A−net，D−net に含ませて考えることができるが，ネットワークのトラヒックを考えた場合独立なネットワークを設けたほうがよいであろう。

B．アーキテクチャの概要

データフローマシン実験機第１版の全体構造は前述（図5.1.1）の通りである。以下そのアーキテクチャの概要について述べる。

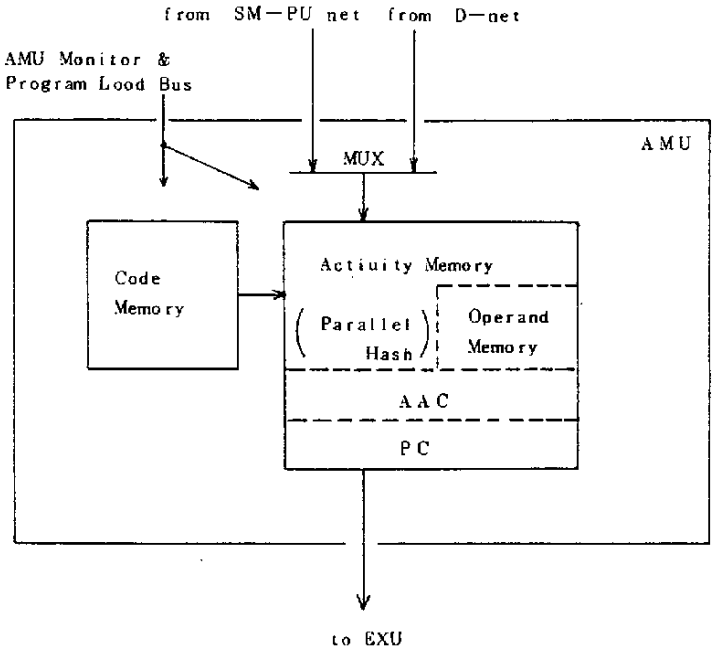
a．AMU（Activity Memory Unit）

ＡＭＵは，トークンの待ち合わせ・命令の発火制御を行うアクティビティメモリと命令メモリ（コードメモリ）が主な構成要素である。（図5.1.2）

まず，アクティビティメモリについて述べる。アクティビティメモリにおけるトークンの待ち合わせ機構は連想メモリを用いるのが理想であるが，実験機第１版では並列ハッシュ機構を用いる。アクティビティメモリでは，ハッシュ表を用いて到着トークンとそれを待っているアクティビティの照合が行われる。トークン待ちのオペランドについては，オペランドメモリを別に設ける方式も考えられる。発火可能となったアクティビティは，命令とオペランドが１つの命令パケットにまとめられＥＸＵへ送られる。この時，アクティビティ割り当て制御（ＡＡＣ），および優先順位制御（ＰＣ）によって発火アクティビティの流れがコントロールされる。実験機第１版では，アクティビティメモリにおける上記の機構を主にマイクロプログラム制御で実現するが，ハッシュを採用することにより，ハッシュの不完全性についての対処，ハッシュのオーバフロー対策等も必要である。

命令メモリは通常のＲＡＭを考え，階層化（ページング，スワッピング）による大容量化については実験機第２版の課題とする。

制御方式（5.1.4に詳細）としては色付きトークン方式を主に採用するが，コピー方式も実装可能としておく。



AAC ： Activity Allocation Controller

PC ： Priority Controller

図5.1.2　AMUの構成

各AMUについての入力パスとしては，D－net，SM−PU net からの入力の他にホスト計算機からのAMUのモニタおよびプログラムロード用のパスを設ける。（9)参照）

b．EXU（Execution Unit ）

EXUは，データフローグラフにおけるスイッチ，ゲート等の制御ノードと基本算術演算を行う処理装置である。（図5.1.3）

EXU内には複数のAE（Arithmetic Element ）とCE（Control Element ）がある。AEは基本算術演算を実行し，CEはデータフローグラフにおけるスイッチ，ゲート等の制御ノードに相当する命令を実行する。

AMUから発せられた命令パケットは，A－net を介さず，各々のAMUにローカルに接続されたEXUに渡される。命令パケットは構造データ操作命令と制御命令・基本算術演算命令に振り分けられる。構造データ操作命令はPU−SM net を介して目的のSMUへ送られる。

ＥＸＵではAE，CE によって制御命令，基本算術演算命令が実行され，結果パケットが生成される。結果パケットはＤ－net を介して目的のＡＭＵへ送られる。

ＡＭＵとＥＸＵ間のＡ－ｎｅｔが省かれている理由は，ＡＭＵとＥＸＵの負荷を比べた場合，ＡＭＵの方が大きくＥＸＵに対する負荷分散の必要がないこと。また，PU−SM net が設けられているため，ＡＭＵに対するプログラムの割り付けとＳＭＵに対するデータの分散配置の間の自由度が保たれていると考えるからである。

ＥＸＵ内の演算装置（ＡＥ，ＣＥ）の構成としては，性質の異なる物にする方式と同一構成にする方式が考えられる。本実験機の場合は浮動少数点演算器などを特に考える必要はないため同一構成のＥＸＵを用いる。

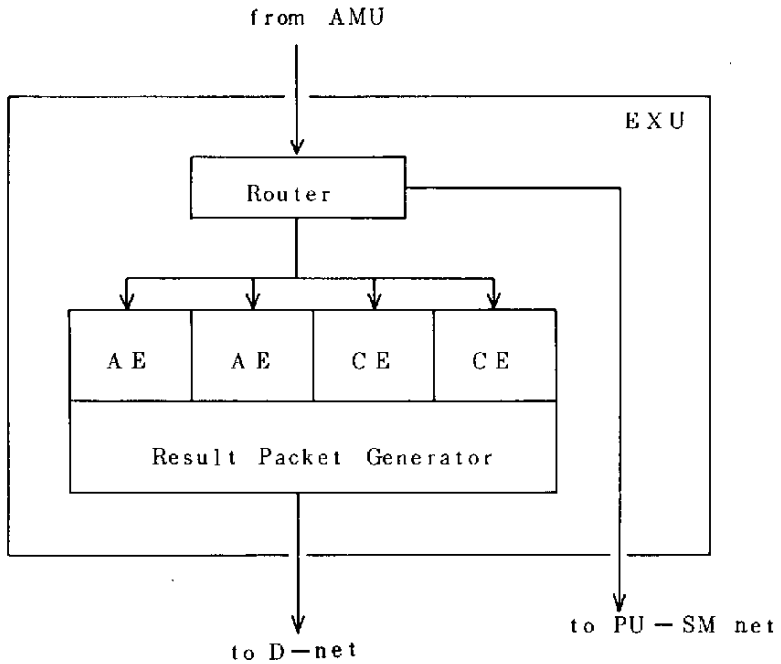


図 5.1.3　ＥＸＵの構成

c．Ｄ－net

Ｄ－ｎｅｔはＥＸＵからの結果パケットをＡＭＵに分配するネットワークであり，マシン全体において最もトラヒックが多いネットワークである。実験機第１版の規模は比較的小規模（5.1.3 Ｃ）であるため，完全結合ネットワーク等も考えられるが，将来の高並列化を想定した柔軟なものが望ましい。また，ＡＭＵとＥＸＵのペア（ＰＵ）内のローカリティを考え，図5.1.4

のようなショートカットパスを持つものにする。ネットワークとしては，扱うパケットが固定長であること，通信が非同期かつ高頻度であること等から，多段クロスバ，多段スイッチングネットワーク等が適すると考える。

またD－netは動的アクティビティ割り付けと関係が深いため，トークンの流れを監視する機構も考える必要がある。

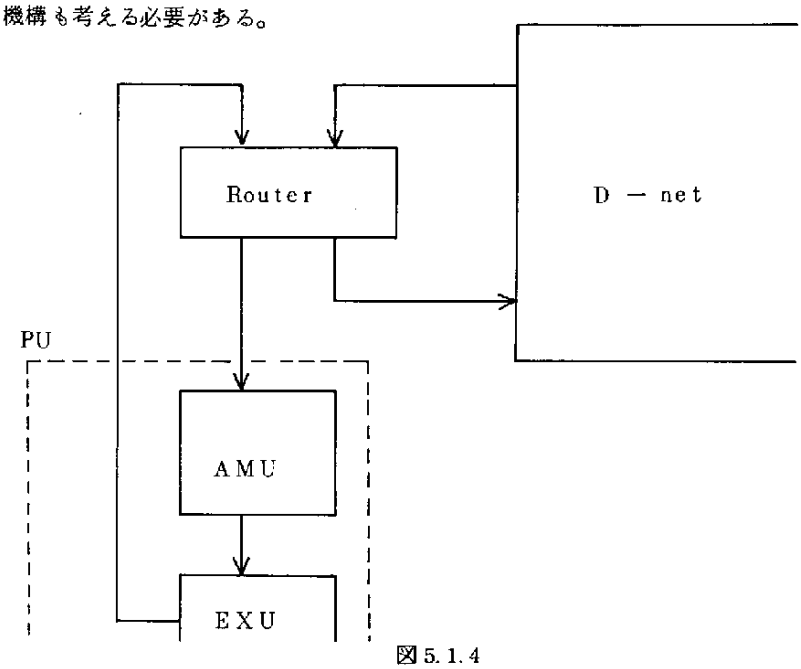


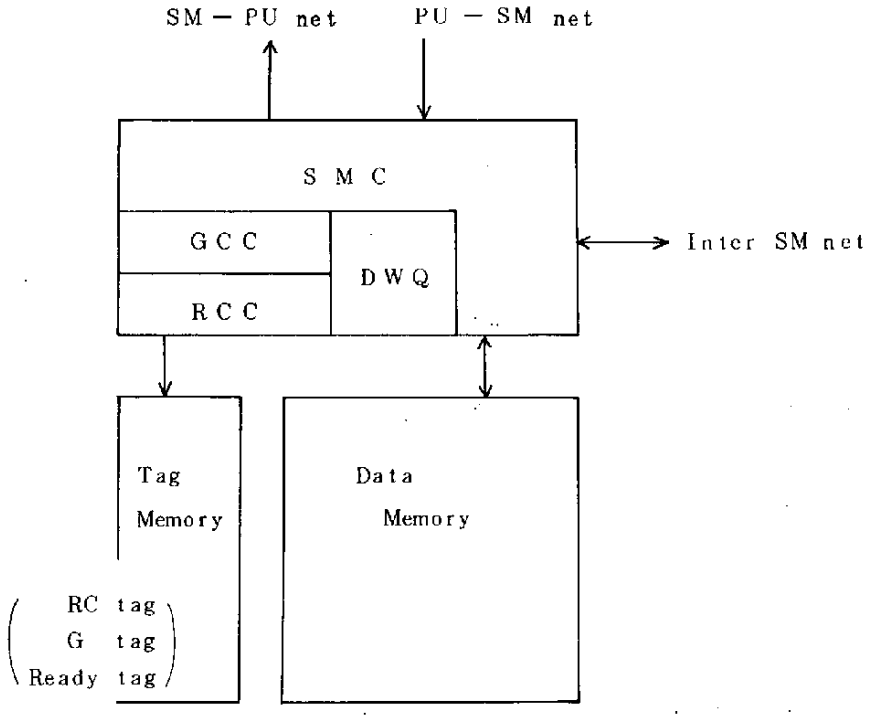
図5.1.4

d．SMU（Structure Memory Unit）

構造メモリユニット（SMU）は柔軟性のあるリスト構造のサポートを基本とする。（図5.1.4）

AMUから発せられる構造データ操作命令は，EXU，PU−SM netを介してSMU内のSMCに渡される。SMCは命令の実行後，結果パケットをSM−PU netを介して目的のAMUに送る。

SMCはハードウェア機構として，参照カウントを用いたガーベジコレクション機能(GCC，RCC）と非同期アクセス機能（Ready tag と DWQ）を持つ。



SMC ： Structure Memory Controller

GCC ： Garbage Collect Controller

RCC ： Reference Count Controller

DWQ ： Data Waiting Queue

図5.1.5　SMUの構成

e．PU−SM net と SM−PU net

この2つのネットワークはSMUとPU（AMUとEXU）を結合するネットワークである。AMUから発した命令パケットは，一度ＥＸＵに渡されるが，実際上は命令パケットの形で，ＳＭＵへ送られるためＤ－netとほぼ同様の性質を持つネットワークである。

f．Inter－SM net

動的データ構造の扱いにおいては構造データが各ＳＭＵに分散配置される状況を想定する必要がある。また後述する入出力処理におけるＳＭＵへのデータの分散化などのために，ＳＭＵ間にInter－SM netを設ける。

Inter－SM Net は各ＳＭＵに対するデータの配置法によって性格が異なると思われる。構造データが分散配置される場合は，データの実体とともに，参照カウントの伝搬も多いため多段スイッチングネットワークが適すると思われる。また，各ＳＭＵ毎に構造データをまとめ，ＳＭＵ間ではデータのコピーを積極的に行うような場合はバーストモード転送が容易な階層化共有バス等も Inter-SM net の候補と考えられる。

g．入出力

実験機では，特定のEXUにIOC（I／O Controller）が接続されたもの（これをＩＯＰ：I／O Processorと呼ぶ）を用意して入出力を行う。

ＩＯＰに接続されているＡＭＵは"入出力用のＡＭＵ"と考えられ，入出力における履歴依存性のある処理のためのmanager 機能を持つ。

ＩＯＰ内のＥＸＵは，入出力処理に必要な構造変換機能が主な役割りとなり，それに必要なワーク用のローカルメモリを持つ。

入出力の処理形態は自立型であり，実験機側から見た場合，ホスト計算機に依存せず独立して入出力動作を行うものにする。

h．ＡＭＵのモニタとプログラムロード

実験機では各ＡＭＵの監視とプログラムロードのために，図5.1.6のような専用バスを設ける。

ホストプロセッサはこれを用いて各ＡＭＵの状態のモニタと起動時のプログラムロードを行う。ただしプログラムロードは，各ＡＭＵ内にＩＰＬ用の機構を設けＩＯＰからロードする方式も考えられる。

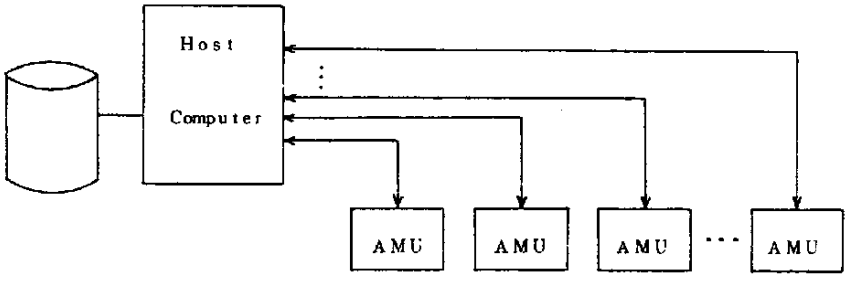


図5.1.6

C．データフローマシンの規模

前述の実験機の予想される規模について簡単に述べる。

a．全体の規模

マシンの規模はＡＭＵの台数で８～１６台程度のものを考える。ＥＸＵおよびＳＭＵについては，ＡＭＵとの処理能力のバランスにおいて決定されるが，おおよそ以下に示す規模と思われる。

ＡＭＵ　　８～１６台

ＥＸＵ　　８～１６台

（ただし，１つのＥＸＵ内のＡＥ，ＣＥの数は４台程度）

ＳＭＵ　　８～１６台

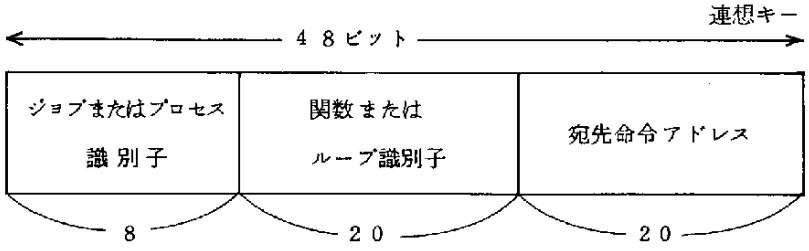
b．各ユニット内の主要メモリの容量

表 5.1.1

主要メモリ	ワード長	１ユニット当たり	全容量
アクティビティメモリ			
（ハッシュ用）	４８ bits	４KW×８バンク (*)	1.6～3.2MB
（オペランド用）	６４ bits	３２KW	約２～４MB
（*）　並列ハッシュ			
コードメモリ	６４ bits	３２～６４KW	２～８MB
構造メモリ（データ部）	７２ bits	１２８KW	８～１６MB

実験機第１版の主要メモリ容量を表 5.1.1 に示す。また各メモリのワードの構成の一例を図 5.1.7 に示す。

○　アクティビティメモリ



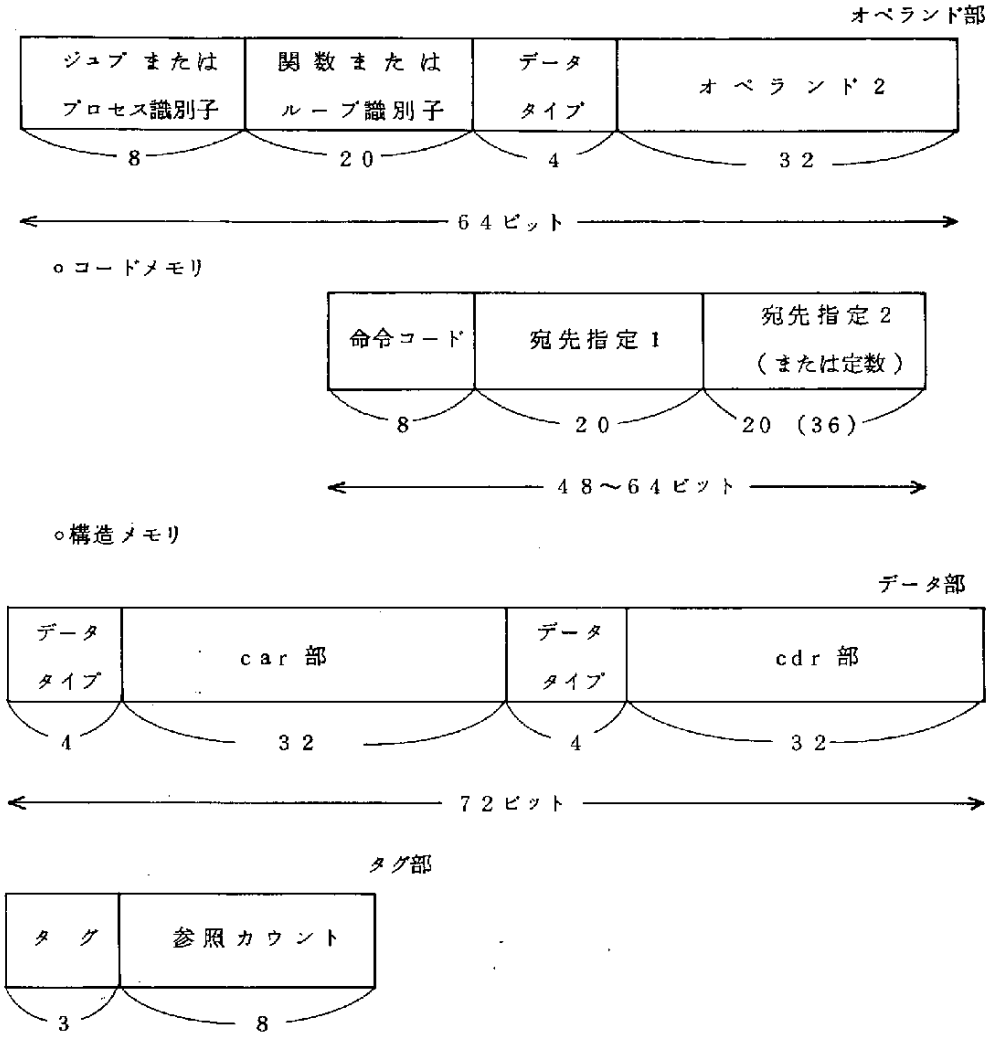


図 5. 1. 7

5. 1. 4 制御方式と命令セット

A 制御方式

4.3節で述べたようにデータフロー方式に比してリダクション方式は，そのハードウェア構成の詳細が不明でありその検討に長期的な視野に立った研究を必要とすること，性能面でやや劣ると推定されること等の問題がある。従って，実験機第1版はデータフロー方式にもとづいたデータ駆動型データフローマシンとする。リダクションの機構はこのデータフローマシン上でシミュレートまたはエミュレートし，その評価を行う。

またアクティビティ制御におけるいくつかの有望方式を比較・検討するために，トークンには少なくとも以下の目的のための識別子を収容するフィールドを用意する。

・ジョブまたはプロセス識別子

非決定的処理やジョブまたはプロセスの制御方式を解明する。

・関数またはループ識別子

関数やループの起動（instance）の間でコードを共有できる。

・プログラム状態タグ

スーパバイザモードとユーザモードを区別する。入出力制御等のシステム共有リソースに関する操作は特権命令としてスーパバイザモードでのみ実行できるようにする。

・トレースタグ

命令のトレース制御等に使用する。

実際にこれらの識別子を使用した場合とそうでない場合の制御方式を比較するために，各構成要素（特に演算ユニット）はマイクロプログラム制御のような柔軟な構造にしておく必要がある。

B．命令セット

命令は基本的に2つまでのオペランド入力を有し，また定数オペランドの指定が可能とする。命令の結果の宛先は1命令で2つあるいはそれ以上指定可能とする。

取り扱う基本データタイプとしては少なくとも，論理値（1ビットまたは整数と同一長），整数（16〜32ビット程度），浮動小数点（32ビットまたはそれ以上），文字または文字列（1〜4文字またはそれ以上），およびポインタ（20ビット程度）を含むものとし，これにデータタイプを示す4ビット長程度のタグを付与する。

また構造データタイプはポインタで表現するものとし，その実際のデータは構造メモリに格納される。構造データのタイプは配列，レコード，およびリスト等が含まれる。

a．基本演算命令

論理値，整数，浮動小数点，文字のような基本データタイプの値に対して，以下の機能を有する命令を用意する。

(1) 加減乗除および比較演算

入力オペランドは整数，浮動小数点，および文字（または文字列）である。

(2) 論理演算

入力オペランドは論理値である。

(3) ビット操作

特定のビット位置のセット／リセット，テスト，およびbitwise OR, bitwise AND

等の機能を含む。

(4) タイプ変換

整数と浮動小数点の間等のデータタイプの変換を行う。データにタグを付与し，各演算命令に自動的にタイプ変換するような機能を持たせれば，このタイプ変換命令は不要である。

基本データタイプのデータ幅は３２ビット程度とする。浮動小数点演算に関しては，必要度に応じて，命令処理装置内でエミュレートする方法，専用の浮動小数点演算ハードウェアを付加する方法，あるいはソフトウェアでシミュレートする方法等の選択の余地がある。

b．構造データ操作命令

ストリング，配列，レコード，およびリスト等の構造データに対する処理命令である。この外，抽象データタイプのサポートや，ハッシュ表のような連想検索に必要な構造データタイプのサポート等も考えられる。構造メモリを参照カウント方式で実現する場合には，参照カウントの制御機能も必要である。

知識情報処理における推論の基本機能となる統合化（unification）の高速な実行を考える場合，上記機能をも含めたリスト置換，リスト照合，およびデータベース操作のような高機能の構造データ操作命令も検討する必要があろう。

c．条件分岐用命令

条件入力によってトークンの行き先を切換えるようなスイッチ命令，T（true）ゲート，F（false）ゲート等の命令である。

d．関数／ループの制御

関数やループ本体の入口で新しい識別子を割り当てる命令，関数やループ本体に渡される引数トークンの識別子を変更する命令，および関数やループの終了時に得られた結果トークンの識別子を関数やループに入る直前の値に戻す命令を用意する。

この外，識別子の再使用をサポートするためには使用中の識別子を解放するための命令も必要とする（ただし，この場合残留している phantom token の消去を行うためのハードウェア機構が必要となる）。

関数やループ識別子を割り当てるとき，一般的にはスケジューラが介入し，新たな関数やループに対して処理装置および識別子の割り当てを行う。このとき，割り当てられた処理装置に目的とするプログラムが存在しなければ，そのローディング制御も行うことになろう。

e．AND／ORゲート

通常の命令は全てＡＮＤゲートの機能を有する。即ち，その入力オペランドが全て揃ったときに起動される。これに対してＯＲゲートは入力オペランドのうちいずれかが到着すれば，直

ちに実行可能となる。通常，これ以降に到着したオペランドは吸収される。このＯＲゲートは記憶ノードの１種と考えられ，関数の部分実行を実現するためのトリガーを生成する時等に使用される。

f．記憶ノード

入力オペランドを記憶する機能を有するノードであり，非決定的処理における状態記憶やループにおける定数記憶等のために使用される。この記憶ノードは様々の識別子間で共有されることが多いため，一般には識別子の変更機能と組み合わせてインプリメントされよう。

g．入出力命令

ファイルのOpen／Close 機能，シリアルインタフェースファイルのread／write 機能，ブロックインタフェースファイルと構造メモリ間のread／write機能，およびデバイスステータスをテストする命令が考えられる。この外，構造メモリと入出力装置間のリスト単位の転送命令やデータベース操作命令のような高機能命令の検討も必要である。

h．ジョブ／プロセス制御命令

ジョブやプロセスの生成，終了，一時実行停止，および優先順位付け等を行うための命令である。以下に各命令の実現手段例について述べるが，アクティビティが各部に分散しているデータフローマシンにおいてはその実現は困難であり，またその実現方式によってはアーキテクチャに大きな変化をもたらす恐れがあるため事前の十分な検討が必要である。

(1) ジョブ（プロセス）の生成

各ジョブにユニークなジョブ（プロセス）番号を割り当てるため，システム内にジョブ（プロセス）マネジャが必要である。

(2) ジョブ（プロセス）終了

ジョブ（プロセス）を終了させるためにはジョブで生成したトークンを消去すると共に，アクティビティ記憶や構造データ領域のようなリソースを解放する機構を要する。また，特にジョブ終了時には使用していたファイルのクローズ等の後処理を要する。トークン消去については，データ転送路にジョブ（プロセス）番号の判定回路およびゲート回路を付加することにより比較的容易に実現できよう。アクティビティ記憶内のゴミ掃除はジョブ（プロセス）番号をキーとしてアクティビティ記憶を連想検索する機能を設けるか，またはアクティビティ記憶の割り当てをジョブ（プロセス）毎に管理する等の手段が必要となろう。構造メモリの解放は，構造メモリ割り当て時にジョブ（プロセス）毎に管理する方式が有望であろう。

(3) ジョブの一時停止

ソフトウェア的な制御手法としては，例えば処理途中の適当な位置にゲート制御命令を挿入しておき，その制御入力を制御する方法や，スケジューラに知らせて指定ジョブ（プロセス）に属する関数やループの起動を一時的にサスペンドする方法がある。ハードウェア的には，例えばデータ転送路に，特定のジョブ（プロセス）に属するトークンを一時的にキューしておく方法が考えられる。（この場合，キューがオーバフローすることによってデッドロックが発生するのを避けるような手段が要求されよう。）

(4) 優先順位制御

データ転送路において，優先順位の高いジョブ（プロセス）に属するトークンが優先順位の低いジョブ（プロセス）に属するトークンを追い越すような機構が必要である。1つの実現方法としては，転送路の1部に複数のFIFO（first-in first-out）キューを用意しておき，FIFOキューの入口で優先順位に応じてトークンを各FIFOキューに分配する手段と，FIFOキューの出口で優先順位に応じてトークンの取り出し順序の重みづけを行う手段を設ける方法がある。用意するFIFOキューの数は要求される優先順位によって決定されよう。

(5) モード制御命令

通常の計算機と同様に，プログラムの実行状態をスーパバイザモードとユーザモードに区別させることは，プログラムの信頼性を向上させる上で意義があると思われる。一部の命令はスーパバイザモードでのみ実行できる特権命令としてインプリメントされよう。両モードを区別する方法として，トークンにモード状態タグを設ける方法や，特別の識別子をスーパバイザモードとして用いる等の方法が考えられる。モード制御命令はこのモード状態を変更する命令として実現されよう。モード状態に応じて優先順位付けが必要な場合は，前述の優先順位制御の機構が使用されよう（例えば，緊急度を要する入出力処理要求等はスーパバイザモードで実行し，その実行優先順位を高くすることによって，従来の割り込みのような概念を実現できよう）。

入出力命令や関数起動命令等は，入出力要求や関数呼出しのスケジューリング等のシステムのサービス機能を必要とするため，このモード制御命令の一部と考えることもできよう。

i．再構成用命令

システムの特定のハードウェアリソースに障害が発生したときや，システム規模と性能の関係を評価するときは，システム再構成の手段が必要となる。ソフトウェア的解決法としては，スケジューラやメモリ管理内のシステム構成テーブルを更新する方法がある。ハードウェア的には，ネットワークに特定の構成要素をバイパスしてデータを転送する経路を設ける等の手段

が考えられる。

j．トレース／ダンプ制御命令

トークンにトレースタグを用意し，このタグがオンであれば，命令終了時に入出力を起動するような機構が必要である。また，特定の命令コード（例えば関数起動命令）や，特定の識別子，或いは特定の命令アドレスの命令のみをトレースするような機構も必要となろう。

C．トークン形式と命令形式

トークンには前述の制御方式で述べたような識別子を付与するものとする。このうち，ジョブまたはプロセス識別子は8ビット程度，関数またはループ識別子は20ビット程度のフィールドを用意するものとする。トークンを送るべき宛先命令アドレスの指定は，ローカルアドレスの概念を導入すれば14ビット（2^{14} =16KW）程度でよいと思われるが，システム共有リソース（入出力やスケジューラ等）をアクセスするためにはグローバルアドレスの指定が必要と思われるため20ビット程度のフィールドを用意するものとする。データの長さは32ビット程度とし，必要に応じてデータタイプを表わすタグ（4ビット程度）を付加する。図5.1.8にこれらのフィールドを収容したトークン形式の例を示す。このうち，アクティビティメモリで命令の実行可能検出を行う際の連想検索キーとなるのは，ジョブまたはプロセス識別子，関数またはループ識別子，および宛先命令アドレスの各フィールドを連結した長さとなる。

命令形式はインプリメントに大きく依存するが，収容されるフィールドとしては命令コード，宛先指定フィールド，および定数フィールドが必要である。命令コードは実行すべき命令のタイプを示すもので8ビット程度あれば十分であろう。宛先指定フィールドは命令の実行結果を転送すべき目的地を示すもので一般に，次の命令アドレス，そのポート番号（命令の何番目のオペランドとなるかを示す），次の命令が実行可能となるのに必要なオペランド数，および場合によっては次の命令が収容されているPU番号を求めるための情報が収容される。この宛先は複数個指定できるようにする方が望ましい。定数フィールドは命令で定数オペランドを使用するときのみ必要であり，その長さは定数の表現法に依存する（一般には32ビット程度に制限することになろう）。

プログラムの各命令毎にこれらのフィールドに対する要求は異なるが，命令を可変長にするとアクティビティメモリの制御が複雑となるので，できれば固定長にするのが望ましい。この場合全命令に対して上記フィールドを用意すると命令語のビット長が大きくなり，フィールドに無駄が生ずる。従って，語長を48～64ビット程度に固定し，定数の有無や宛先指定の個数等に応じていくつかの命令形式を用意した方がよいと思われる。図5.1.9にその例を示す。

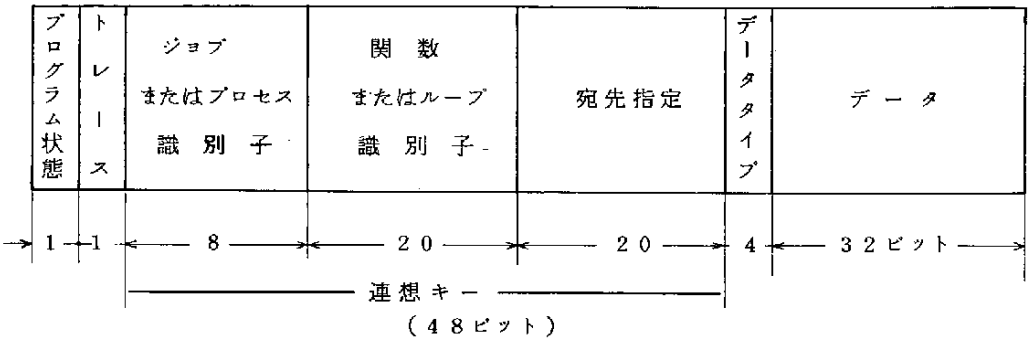
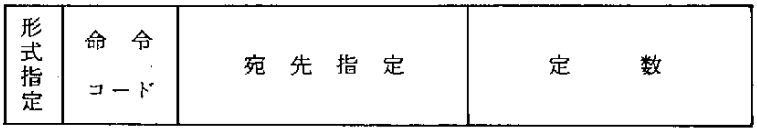
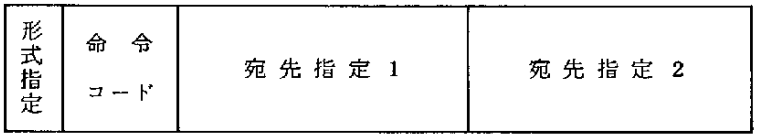


図 5.1.8　トークン形式の例



(a)　定数を含む場合



(b)　宛先指定2個までの場合

図 5.1.9　命令形式の例

5.1.5　オペレーティングシステム

実験機第1版のオペレーティングシステムとしては，マシンを動作させるのに最低限必要な機能を備えることを基本とし，更に，データフローマシンであるが故に問題となる機能に関しても，実験機第2版への準備のために試験的に取り込んでみることを考える。

A．制御プログラム

a．ハードウェア管理機能

(1)　割込

割込の概念が，従来とやや異なると思われるが，入出力割込，プログラム割込，マシンチェック割込，ＳＶＣ割込の制御を行う。

(2)　入出力制御

ＰＵと対等に置いたＩＯＰによって，入出力処理を制御する。

(3) 自動回復

故障検出後，サブプログラム単位で再試行可能にする。構造データの更新等にからんだ再試行は，第2版への課題としておく。

(4) 構成制御

デバグの容易化，故障したハードウェアの切離し，リソース制限による諸性能への影響調査を目的とした，システム構成の制御を可能とする。オペレータコマンドによってのみ行うものとする。ＰＵ，ＳＭＵ単位での切離しを可能とする。

b．タスク管理機能

(1) スケジューラ

実行時に必要となった手続き等をＰＵのＡＭにロードする。ＰＵへの割り付けは，各ＰＵの負荷状況を考慮して行う。この割り付けアルゴリズムは，可変としておき，種々の試みを可能とする。

処理優先権については，若干考慮する程度とする。

(2) メモリ管理

AM，SMUの領域管理を行う。

(3) チェックポイント

中間結果のダンプと，その情報を使っての再開始処理も出来る限り取り入れてみる。

c．データ管理機能

(1) ファイル

標準的なファイル一種類をサポートする制御プログラムを備える。

(2) 端　末

キーボード／ディスプレイをサポートすることを基本とする。

d．ジョブ管理機能

(1) ＩＰＬ

ブートプログラムをＰＵ内のＲＯＭに置き，ＩＰＬ開始スイッチにより，ブートプログラムを起動し，ＯＳをディスクからロードする。

(2) コマンド処理

端末からのコマンド入力に対しての処理を行う。

B．処理プログラム

a．言語プロセッサ

(1) コンパイラ

データフローマシン用の高級言語をサポートする。クロス方式とする。

(2) アセンブラ

アセンブラは，データフローマシンに適すると思われる図形式を採用することを考える。クロス方式。

b．サービスプログラム

(1) リンケージエディタ

(2) デバグ用サポート

・トレース

実行順序の再現性のあるモードを設定する。

・ブレーク

ユーザＩＤ等を利用して，徐々に停止出来る機能を設ける。

・シングルステップ

並列実行可能な複数パスに亘って1命令／パスのステップ制御を可能とする。

・オペランド値変更

擬似トークン発生機能を備え，ブレーク時等でのオペランド値の変更を可能とする。

デバグ用サポートツールにおいては，並列動作を分かり易く表現出来る2次元図形の駆使を念頭に置く。

5.1.6 実装法と目標性能

A．モジュール分割

アクティビティメモリユニット，エグゼキューションユニット，構造体メモリユニット，入出力ユニットという機能分割をそのまま実装上の基本モジュールとする。

B．モジュール間インタフェース

モジュールごとに別々の製造工程を経ることからくる物理的誤差に対する許容性，モジュール結合の際のタイミング調整の不要なこと等を考えて，非同期インタフェースを用いる。

C．モジュール内制御方式

モジュール内制御方式としては，同期式のパイプライン制御とし，エグゼキューションユニットについては，スループットに合わせて処理装置を並列配置する。

D．データ幅

モジュール間データパスのデータ幅については，将来のVLSI化を考慮してピン数制限をとり

入れた幅を設定する。

E. 目標性能

エグゼキューションユニットは，2 μsec に1つの命令を処理する能力をもつものとする。すなわち，0.5 MIPSを1台での目標性能とする。また，他の機能モジュールについては，この2 μsec のパイプラインステージ時間に合うように内部をパイプライン化する。

F. 素　子

素子としては，通常のTTL IC や LSIを用い，マイクロプログラム制御を中心とする。

G. サービスプロセッサの実装

システムの保守およびデバッグのため，サービスプロセッサを実装する。実装形態としては，各AMUとバスで結合し，Dnet とのインタフェースも設ける。中心となる役割りは，AMUのメモリへの read ／ write アクセス，および発火検出機構の制御であり，HALT命令やシングルステップ制御等を行う。ネットワークに対しては，トークンの挿入，除去等の機能をもつものとし，システムを構成する各ユニットのメンテナンステストをネットワークを利用して行えるようにする。

5.2　実験機第２版のイメージ

実験機第２版は第１版の経験と前期の研究成果をもとにデータフローマシンに推論機能を導入するとともにデータフローマシンの更に高度なアーキテクチャをも追求したマシンと位置づけられる。

5.2.1　高級言語

第２版の高級言語は知識情報処理の核言語の開発をもとに設計される。その言語は論理プログラミング型でunification　が主要な機能となるであろう。この言語の計算モデルがデータフローかリダクションかは前期の研究成果をもとに決定される。更にこの言語は仕様記述やデータ抽象との関連を考慮しつつ仕様が定められる必要がある。

5.2.2　ＯＳ

第１版のＯＳはかなりの部分をサービスプロセッサが担当していたが，より多数のリソースを管理する第２版ではこのような集中管理方式は効率を落とすおそれがある。そのためＯＳの分散化をはかる。

スケジューラは知識情報処理の応用に適したプログラムの動的割付法を考慮せねばならない。

また割り込みがデータフローマシンでどのような意味を持つかを明らかにし，必要ならその機能を取り入れる。

リソースの管理については非決定性問題や履歴依存性問題のポイントを明らかにし，これを解決する機構を取り入れる必要がある。またマルチジョブを可能にする機構も必要であろう。

更にＯＳ自身をデータフロー言語で記述する等してＯＳ機能のデータフロー化をはかる。

5.2.3　ハードウェア

第２版のハードウェアは推論機能をサポートするためunification　の機能をハードウェア化する。この場合，探索を並列に実行する機能，探索プロセスが互に通信する機能，並列プロセスを収束させる機能，数の爆発を制御する機能等が必要となろう。

また第２版では各ユニットをより専用化し効率を高める。例えば，実行ユニットは第１版ではマイクロプログラムであったが，第２版ではその多くを専用ロジック化する。アクティビティメモリも第１版の並列ハッシュ方式を連想メモリ方式としロジックの単純化と高速化を達成する。

構造メモリは知識情報処理に必要な機能を第１版より効率良くサポートできるようにする。またコードメモリ，構造メモリをバーチャル化し，より大規模な応用の実行を可能にする。この場合ス

ワップやファイル機構との接続方式等について検討が必要である。

5.2.4 信頼性技術

マルチプロセッサシステムの信頼性の問題は以前から重要な問題として指摘されている。第1版は信頼性に対しては最低限必要な処置を施す程度であるが，第2版はデータフロー方式における信頼化技術を十分に検討することとする。

まず故障検出方式としては，ソフトウェアやハードウェアの多重化方式がある。これらはそれぞれかなりのオーバヘッドを生じると考えられる。第2版の応用が知識情報処理ということを考えると多重化はソフトウェアを中心とすることになろう。データフローマシンは同一のプロセッサが多数あるので，同じプログラムを異なるプロセッサ群に割り付ける形のハードウェアの多重化も考慮されよう。

故障検出のレベルについては，リカバリーの方法等を考慮しつつ決定する必要がある。

リカバリーについては，データフローマシンも密結合マルチプロセッサに共通の困難さをもっている。リカバリーのためには必要な情報をセーブしておかねばならないが，これを行うためにはすべてのプロセッサの実行を一時中止しなければならない可能性があり，これは大きな負担となるため，このセーブの手法について検討が必要である。

マシンの再構成については故障ユニットを切り離す方式で実現が可能であろう。この場合故障ユニットの所属するユニット群を切り離す方式等も考えられる。

5.2.5 ネットワーク

ネットワークはより大規模のシステムを実現するため階層化と拡張性の導入が必要であろう。この場合でも遅延を少なくしスループットを高めるためには問題の持つ局所性を十分に利用する必要があろう。第2版の規模ではバス方式は高い性能が望めないと考えられ，スイッチングネットワークが有力な候補と考えられるが，そのトポロジーについては更に検討が必要である。

5.2.6 実 装 法

第2版ではnMOSまたはCMOSのVLSI化をはかる。MOS方式の利点は集積度が高いため1個以上のプロセッシングユニットを1チップ上に納めることができ，ピン数の減少がはかれるとともに信頼性の向上が期待できる。MOS方式は速度が遅いが，データフロー方式は多数のプロセッサを高いスループットで動作させ高性能を達成できるため1つのユニットの遅延時間はあまり問題とならないだろう。

この時の実装密度は数十万ゲート／チップ，クロックは５０ナノ秒程度が想定される。

このユニットの能力は１MIPS程度と考えられるので，このユニットを１００台程度接続した第２版の平均性能は２０～３０ＭＩＰＳ程度が期待される。

6. 研究開発内容

6.1 研究課題

6.1.1 基礎研究

第5世代プロジェクトではデータフローマシンを，知識情報処理における推論マシンへ発展していくものとして位置付けている。データフローマシンの基礎研究ではこれに向けて，理論面の整備および要素技術の確立を図っていくことにする。基礎研究における主要な検討課題は以下の通りである。

A. 計算モデル

述語論理型計算の基本となっているresolution method, natural reduction method, とdata flowあるいはreductionによる計算モデルとの対応性を明らかにする。関数型の計算とdata flow計算モデルとの親和性は次第に明確になりつつある。またソフトウェア基礎理論の研究により，述語論理の計算は関数型計算を包含するものであり，両者は相矛盾する関係でないことが明らかになってきている。関数型計算と述語論理型計算の橋渡しをするものは何であるか，これがdata flow計算モデルの枠組の中でどう捉えられるかを明らかにしていくことが研究課題となる。例えば，Kowalskiの提唱するConnection Graph Proof Procedureがdata flowモデルでどう捉えられるか，あるいは別の観点からGraph methodを案出することが研究テーマの1つとなる。また推論の基本メカニズムであるunificationのアルゴリズムをdata flow計算モデルでどう確立するかということも重要なテーマである。少なくともdata flow計算モデルでは変数bindingの概念はない。unificationにおけるbinding概念をdata flowモデル上でどのように捉えるかを明らかにしていく必要がある。

B. 基本メカニズムの実現法

データフロー計算機構によって述語論理型計算を行う場合の基本メカニズムを検討する。データフローによる発火制御という観点から関数型計算と述語論理型計算を比較すると，関数型ではby valueメカニズムにもとづいて関数起動が行われるのに対し，述語論理型では，節の起動はパターン照合にもとづく。このパターン照合をデータフローで行う方式およびハードウェア機構の検討が課題となる。また，関数起動の場合，引数の受渡しの方向と起動される関数が一意に定まるのに対し，節起動では，引数の受渡しの方向もまた起動される節も予め定まってはいない。これをデータフローの計算機構の中で統一的に扱う手法を検討していく。述語論理型計算プログ

ラムをデータフローグラフとして表現する方法には，アセンブル方式，コンパイル方式などプログラムを変換する方式とインタプリタ方式が考えられるが，これらの方式の得失を明確化する必要がある。

C．後戻り制御

逐次型推論の場合，goal を subgoal に分割し，その subgoal を新たな goal として推論を進めていく，この場合，goal の達成に失敗した場合のバックトラック（後戻り）が不可決となる。

データフロー制御で推論を行っていく場合には，いくつもの可能性がある非決定的な計算（subgoal）を並列に実行させることが可能である。しかしこの場合，推論が深まるにつれて数の爆発が起こる危険性がある。有限資源下でBreadth first method をとるには，何らかのバックトラック機能が必要とされる。これをデータフロー制御でどのように行うかを明確にする必要がある。また，数の爆発を抑え且つ並列性を最大限引出すような制御機構の導入が必要であり，要求駆動的要素を取り入れたデータフロー制御方式についても検討する必要がある。

D．非決定性処理と履歴依存性

Ｉ／Ｏ処理やファイル処理などで，履歴そのものをデータフローの枠組みの中でどのように扱っていくかが大きな課題となっている。これはdata flow モデルに代表されるような履歴依存性を排除した関数型計算モデルに共通の問題でもある。関数的性質と履歴依存的性質を統一的に扱うことのできる計算モデルも合わせて検討しておく必要があろう。

推論におけるAnd Or 処理のような非決定性処理では，C．で述べた数の爆発を防ぐことの他に，解が得られた時点で，alternative として進められている計算をどう止めるかという問題が生じる。同時進行している計算（プロセス）に対し一 に止める手法等の検討をデータフロー実行制御機構と合わせて進めておく必要がある。

E．抽象データ型の導入

抽象データ型・オブジェクト指向などの概念と，data flow計算モデルを含む関数型の計算モデルとの関係を明らかにしておく必要がある。抽象データ型をハードウェア機構としてどのようにサポートしていくか，データフローの枠組みへどのように取り込んでいくかについて検討することが大きな課題となる。

F．推論向きデータフローマシンＯＳの研究

データフローマシンの研究は未だ要素技術の開発の段階であり，オペレーティングシステムについては手がつけられていない。先ず，資源の割り付け方式，負荷分散制御等の基本技術の確立を行う必要がある。これを推論向きオペレーティングシステムとしていくには，有限資源下で数

の爆発を抑えながら並列性を最大限引出すための実行制御方式，負荷管理方式，プロセス間の通信制御方式の検討が必要である。

6.1.2 高級言語

データフローマシンにおける高級言語は問題の並列性を十分に引き出すために副作用のない言語が必要である。しかし実際には効率等の点で幾つかの問題があり更に研究が必要である。それを以下に述べる。

A．言語自体の研究

a．文　法

言語のスタイルとして単一代入型と関数型の言語が考えられる。両者は多くの点で共通の性質を示すがループ，構造データ，入出力等については必ずしも対応が明らかでない。これらをどう扱うかが研究課題となる。その際には言語のセマンティクスにあたる計算モデルの検討も必要である。

命令に関しては制御構造に関するものが言語の記述力に大きく影響するので検討が必要であろう。一例としてValのfor-allやwhileをあげておく。構造データに関してはストリームの記述をどうするかが課題となる。

b．data abstraction

data abstractionは近年の言語研究の成果の1つであるが，これをデータフロー言語にどのように取り込むか検討が必要である。この点については積極的に取り込もうとするValと否定的なFPの両極がありその方向は定まっていない。

c．非決定性

非決定性の問題はリソース管理やデータベース処理の他，推論の過程でも生じてくる。これらを記述する命令としてguarded commandやセマフォア等の検討が必要である。また順序がバラバラな要求を整理するための機構としてserializerとかマルチエントリ等の検討も必要であろう。

d．履歴依存性

関数型言語にはもともと履歴の概念はないが，データベースや入出力装置のように状態を保存するものを取り扱うために履歴を処理する機能が必要である。ストリームの概念が有力な解決法のように思われるが更に検討が必要である。

e．推論機構

推論機構を一般的にサポートするものとしてunificationの取り込みを検討する。それに

伴いパターンマッチやプロセス通信の命令が高級言語でも必要となるかもしれない。また論理プログラミング言語とデータフロー言語の関係についても研究の必要がある。

B．言語処理系の研究

a．コンパイラ

関数呼出し，制御構造，データ構造等のより優れた実現方式を考え，中間言語と機械語の命令セットを検討する。特にオーバヘッドを減少するための最適化が重要な課題である。

b．ローダ

ローダについては，アクティビティ割り付け方式と関係があるので6.1.4と協同で研究を行う。

c．デバッグ機能

言語の段階で導入できるデバッグ機能について検討する。トレース機能やポーズ命令，suspend と retry 等がデータフロー方式の中でどのように実現できるかを検討する必要がある。

6.1.3 応用研究

データフローマシンにおける応用研究の課題は以下の通りである。

A．応用分野の調査と並列度の測定

データフローマシンは主として数値計算を対象に発展してきたため，その応用として知識情報処理を取りあげた場合，データフローマシンがどの応用で，どの程度有効であるかが明らかになっていない。このため課題として次の２つがある。

(1) 知識情報処理のどの分野がデータフローマシンに適しているか

(2) その応用の並列度はどの程度あるのか

B．新アルゴリズム

従来のアルゴリズムは並列処理を前提として考案されていないものが多いためデータフローマシンで有効に処理できない場合がある。そのため

(1) より並列性の高い新アルゴリズムの開発

(2) 記号処理により適した計算モデルの開発が課題となる。

C．推論機能とデータベースの取り込み

推論機能をデータフローマシンで実現する場合，どのような形で推論機能を取り入れるかのイメージを明らかにする必要がある。またデータベースマシンとの結合形態のイメージも明らかにする必要があろう。従って課題として

(1) 推論機能とのインタフェースを明らかにすること

(2) データベースマシンとのインタフェースを明らかにすること

の2点をあげる。

6.1.4 アーキテクチャ

A. アクティビティ制御方式

計算の並列制御機構におけるデータフロー方式，リダクション方式，および両者を融合した方式の研究を行い，それぞれの場合のハードウエア構成や得られる性能上の得失に対する比較・検討を行う。

さらに，コピー方式とするか識別子付きトークンとするかに伴う構成上の検討，および性能の評価が必要である。また，構造を有するオペランドの表現（ポインタを用いるかコピーするか）も構成や性能に影響する。

知識情報処理や制御プログラムの記述に必要不可欠な非決定的処理の制御に必要な機構，および命令セットの検討も必要である。

B. AMU（Activity Memory Unit）

AMUにおいて，そのハードウェア構成や性能に大きく影響する要因は以下の通りである。

a. 関数やループ識別子の有無

関数やグループの起動に際してその識別子を使用しないでコードをコピーするか，トークンに識別子を付与してコードを共有するかである。前者は識別子制御のための命令セットや機能は不要であるが，関数やループ起動毎にコードをコピーしたり，その引数の同期をとるためのオーバヘッドが問題となる。これに対して後者の場合は，識別子制御のための命令や機構を必要とする。いずれが良いかは，アプリケーションやハードウェア構成に依存するためシミュレーションによる評価が必要である。

b. 構造データの表現

複雑な構造を有するオペランドを命令実行毎にコピーするか，そのデータを構造メモリに貯えておき命令にそのポインタを渡すかは設計上の大きなポイントとなる。前者はネットワーク内を構造データが流れるためその負荷が大きくなること，AMU内に動的メモリ管理機能を要すること等の問題がある。一方，後者はトークンを固定長のポインタとして扱える利点はあるが，実際のデータをアクセスするためにはさらに構造メモリへ命令へ渡すためのネットワークを必要とすることや構造メモリが共有されるため参照カウント法のようなガーベッジコレクションの機能が必要等の問題がある。

c. 命令のオペランド数

多入力オペランド命令を実現しようとすればアクティビティ（命令）毎にその到着したオペランド数を管理する機能が必要となり，ハードウエア構造はやや複雑となる。これに対して2入力オペランド命令の場合は連想機能を有するメモリを設けることで比較的容易に命令の実行

可能検出を行うことができる。しかし，この場合多入力オペランド機能は2入力オペランド命令の組み合わせで実現することになりオーバヘッドがやや大きくなると推定される。

d．結果の宛先指定

結果を複数の宛先へ送る必要のある場合，一般には命令コードに複数の宛先フィールドを収容する方法，または2～3の宛先指定を有する命令を組み合わせる方法が採用されているが，命令が可変長となりその制御が複雑になるか，または余分の命令を実行するオーバヘッドが生じる等の問題がある。これに対して宛先毎にユニークな識別子をコンパイル時に割当て，プログラム格納用のコードメモリに連想機能を付与し，宛先をキーとしてコードメモリを連想検索する方法がある。この方式の難点はコードメモリが高価になるということである。

これらの要因とそのアーキテクチャや性能に与える影響について十分な検討・評価が必要である。

C．構造メモリ

データフローマシンにおいて複雑なデータ構造を能率よくプログラムに提供するためには，データ構造の実現とそれに対する操作をハードウエアによってサポートする機構が必要である。

この目的のために，これまで幾つかの構造メモリが提案されてきたが，構造メモリに対するすべての要求を満たすものはあらわれていない。

高パーフォーマンスなデータフローマシンを構築することを目指すには，構造メモリの研究開発をさらに進めていく必要がある。

以下，構造メモリを実現する上での研究課題について述べる。

(1) データ構造と操作に関する課題

第5世代計算機システムにおいては，動的木構造，高速サーチが可能なシンボル表，可変長レコード等の複雑なデータ構造が必要となる。

そこで，研究課題としては次の2つがあげられる。

(i) 第5世代計算機システムの応用，つまり知識情報処理において必要なデータの基本構造は何か，また，それをどのようにしてハードウエアで実現するか

(ii) 構造メモリに対するデータ操作命令には何が適切か

また高度な並列処理を実現するには，

(iii) 構造データ操作における各種の性質（参照の局所性の有無，データ構造の規模，寿命等）の把握

をおこない，構造メモリのハードウエア構造に活かさねばならない。

これらの問題は，たとえばデータフローマシンのソフトウエアシミュレータの上で各種の応

用プログラムを実行させること等によって次第に明らかになっていく問題であり，プロジェクト全体において多種多数のデータを収集し議論する必要がある。

(2) メモリの有効利用と操作の効率化に関する課題

データフローマシンでは，メモリ域を意識しない（関数性のある）データ操作がおこなわれる。このため変数名も多くなり，従来のノイマン型マシンに比べてメモリ域の消費量が多い。またコピー操作も頻出するため，操作自体の効率化が重要となる。

木構造型のデータについては，各ノードに参照回数を示すタグを付すことによって複数の（論理的）構造データが部分構造を共有し，操作の効率化とガーベジコレクションをおこなう方法が考えられ，すでにハードウエア化の提案もある。しかし，1回のデータ操作に付随する操作（参照回数の更新操作の伝搬）が比較的多く，新たなオーバヘッドとなることがわかっている。このオーバヘッドの解消策としては，

構造データに対する操作毎の更新はおこなわず，更新操作をプログラム構造に従ってコンパイラが命令として埋め込む。

② 操作命令のレベルを上げる。

③ プログラムの手続き単位にメモリ領域を割り当て，手続きの終了時に領域ごと回収する。

等が考えられるが，明確な評価はできていない。そこで，

(i) 効率のよいデータの共有，およびガーベジコレクション方式の検討

が重要な課題となる。

また，高並列な環境下における，

(ii) データセルの配置方法と物理的コピー操作との関連

(iii) コピー操作自体の効率化

等についても検討する必要がある。

(3) 構造メモリの階層化・高機能化に関する課題

従来のマシンでは，メモリの階層化・仮想化によってメモリアクセスの高速化，メモリ容量の大容量化の両者の要求を満たすことに成功した。データフローマシンの環境下においてもメモリアクセスの高速化，大容量化の要求は続くと考えられるため，構造メモリの階層化を検討する必要がある。しかし，その本質はメモリデバイスの性質，たとえば物理的アクセスタイムや記憶密度に起因するものでなく，構造メモリがサポートするデータ構造のレベルの違いにあると考えるべきである。

構造メモリは単なる線型メモリではなく，メモリ機構側にデータ操作の機能や並列ガベジコレクション機能を持つ一種のLogic-in-Memory である。一方，これが扱うデータは要素

データ型からリスト構造，配列構造等多種におよぶ。更に，構造メモリをデータフローマシンとデータベースマシンとのインタフェースと考える場合，何らかの集合の形でデータを取り扱う必要がある。このような多様の機能を持つメモリ機構を，データフローマシンの高いパーフォマンスに十分な効率と容量を持つように実現することは，たとえＶＬＳＩの技術を用いたとしても容易なことではない。つまりデータ構造の取り扱いのレベルによって，効率的に操作が可能な物理構造の階層化が必要と思われる。

たとえば，各メモリセル単位の取り扱いをサポートするレベル，プログラム内の各手続きが取り扱う構造データの集合のレベル，またプログラム単位に扱うファイルのレベル等が考えられる。

このような階層化を考える上での問題は，データフローマシンという高並列かつ非同期的な環境において，いかにして柔軟性のあるハードウエア構造をつくり上げるかにある。

また，ファイルとの関連においては，ファイルを入出力として捕える方法の他に，one-level-store の考え方もあり，構造メモリとファイル機構との論理的，物理的インタフェースを再度検討する必要もあると思われる。

そこで課題としては，

(i) データフローマシン全体のパーフォーマンス，構造メモリのバンクの容量・バンク数，コストなどの関連においてスワップメモリをどのように位置付けるか

(ii) 高度な並列処理の環境下においてデータの集合をどのようにしてとらえるか

(iii) 構造メモリとスワップ用２次記憶をどのように結合するか

(iv) ファイル機構と構造メモリをどう考えるか

(v) ファイルとのデータの入出力の際に必要な物理構造の変換をどこでサポートするか

等である。

(4) 構造メモリのハードウエア構造に関する課題

構造メモリのハードウエア上の複雑さは，従来のメモリ機構とは異なり，通常の専用プロセッサ並であることが予想される。そこで実装技術を考慮した構造メモリのハードウエアアーキテクチャをひとつにまとめあげることが最終的な課題となる。

D．結合ネットワーク

(1) 用途別方式検討

- A-net，D-net，SM－AMnet，AM－SMnet； 高スループットで遅延の少ない高性能アーキテクチャの検討。トポロジーへの局所性導入の検討。
- Inter-SM net：中・大規模構成向きトポロジーの検討。
- プログラム転送用ネットワーク；中・大規模構成向きトポロジーの検討。

(2) ネットワークアーキテクチャの検討

- 高スループットで遅延の少ない高性能アーキテクチャの研究
- 局所性をもったネットワークのノードスイッチの通信制御機能の研究

(3) 実装法の研究

- ＬＳＩ化
- 高信頼化

E．入出力とファイル

入出力，ファイルに関しては，4.6で述べた様に，ほとんど研究が為されていない現状なので，先に述べた項目はすべて明らかにしなければならない。研究課題として，次のものを掲げる。

(1) 入出力動作と基本命令

端末を主とした入出力機器に対して，どの様な基本命令を設定するか。その命令はアクティビティメモリやトークンとどのような関係になるかを検討する。ＩＯＰの分担として，どんな機能まで持たせるかも明らかにする。

(2) ファイル

ファイルは，意識して使うものか，意識しないで使う方法はないかを検討する。これは1レベルストアの考え方と関数型との考えがうまく適合するのかどうかの問題である。さらに，ファイルのロード／アンロードの場合の動作について，起動，アクティビティメモリ，トークン，ストラクチャメモリ間の関係を明確にする。

(3) 割込，チャネル

チャネルや割込の概念がデータフローマシンの場合にはどう考えたらよいかを明らかにする。プログラムカウンタが一個しかなかった時の概念であるので，データフローマシンにおいても本質的に必要か否かを検討する。その際，処理プログラムと制御プログラムが同時に走行する事とか，それぞれが複数個走行する事による問題点を明らかにする。

(4) 入出力制御処理のデータフローマシンへの取込

(3)の検討を元に，入出力処理を本格的にデータフローマシンに取り込むことを検討する。そのために新しく必要な命令，アクティビティメモリの共有方法，デバイスの速度を考えた制御等の方式検討を行う。

(5) データベースとの関連

ファイルの考えをより論理的にとらえ，データフローマシンでのデータベースの構築の仕方を明らかにする。さらに，データベースマシンとデータフローマシンとの関連を明らかにする。どの様な形でインタフェースをとり，機能分担をいかにして，どの様な命令を設定するか。

(6) ストラクチャメモリの階層化

データフローマシンでは，ユーザには直接見えないが，実際のメモリの消費量が多いものと予想される。特にストラクチャメモリはコピーを取ったりすると相当量が必要となり，現実には，階層化によりスピードと容量を適度に保つ技術が必要であろう。この為の階層化の方式をファイルとの関連で明らかにする。

(7) ストリーム概念の導入

入出力に並列性を出すように，ストリームの考えを導入するためには，どんな機能が必要になるかを明らかにする。

F. アクティビティ割り付け

(1) 方式検討

- コードブロックの複数ＡＭＵへの展開方式
- 複数コードブロックの配置方式
- インスタンス生成方式
- アクティビティの局所性の利用

(2) ローダ・スケジューラの開発

- 基本アルゴリズムの検討
- 第工版用ローダ・スケジューラの試作評価

(3) データフローマシン用ＯＳの研究

- 大量資源の管理方式
- 実装方式
- スケジューリングにおけるプライオリティ制御

G．EXU（EXecution Unit）

AMUとの処理速度のバランスから見て，一般にEXUは複数の演算装置から構成されると推定される。この場合，各演算装置を専用機能化して機能分散型とするか，それとも汎用化して負荷分散型とするかの選択ポイントがある。また，演算装置をマイクロプログラム制御のような柔軟な構造にするか，高速化をねらって専用ロジックで構成するかという問題もある。これらの選択は，処理速度，ハードウェア規模，および柔軟性・拡張性等を総合評価して決定する必要がある。

6.1.5 制御ソフトウエア

データフローマシンの制御ソフトウエアに関する研究課題として以下のものがある。

A．スケジューラ

従来マシンに比して桁違いにリソースの数が多く，また，機能のあまり高くないそれらのリソースを協調させてシステムとしての性能を上げることが最大の目標であるデータフローマシンにとってスケジューリングが大きな課題である。具体的には，プログラムの動的な割付である。手続の呼出に応じて，どのような割付方法を取るのが有効かを調べる。新しい手続に対しては，どのPUのAMに割付るか，既にある手続に対しては，共有すべきかコピーをとるべきか等を明らかにする。

B．分散OS

多数のリソースのある環境であっても，それらの制御が集中しているのでは，そこがネックとなる。処理プログラムの分散と共に，制御プログラムをも分散させて全体の効率を上げることが必要であろう。これを実現するために，分散させる機能単位と独立性，その間の通信の性質等，分散の可能性とオーバヘッドについて明らかにする。

C．割り込み

割り込みがデータフローマシンでは，どのような意味を持つか，割り込みに相当するものを如何に扱うかを明らかにする。

D．復旧，再構成

基本的な信頼性向上技術として，ハードウエアエラーの検出から復旧するための方式や暴走ジョブの一掃法を検討する。具体的には，エラー検出の方法，非同期に実行中の処理を停止させる方法，復旧のためのチェックポイントの取り方等を明らかにする。さらに，再構成のレベルとハードウエアの制御方法も明らかにする。

E．マルチジョブ，マルチプロセス

マルチジョブ，マルチプロセスを行えるようにするためのリソース管理及びプロセス間通信の方式を明らかにする。

F．DFL　OS

オペレーティングシステム自体をデータフローマシン言語で記述することを検討する。OSの移殖性，OS開発コスト，言語の記述力等，データフローマシンがソフトウエアの面で発揮出来る効果を明らかにする。

6.1.6．実装法

A．ハードウエアの分割法

(1) SMUとEXUの機能分担の明確化

(2) IOUの機能の明確化，並びに，その実装形態の検討

B．VLSI実装

(1) VLSI向きアーキテクチャの検討

(2) 1チップに納める機能の検討

(3) ピン数制限からくるデータ幅の制約の性能への影響の評価

C．デバッグ

(1) ソフトウエアのデバッグ

- トレースをとる場合のレベルと再現性の問題についての検討
- ブレーク命令を実現する際のアクティビティの選択的停止のさせ方の検討
- オペランドメモリの内容操作をするための特権命令の実現方法の検討
- プログラムのステップ的実行をするための制御方法の検討
- デバッグを容易にするためのグラフィックを駆使するツールの開発

(2) ハードウエアのデバッグ

- ハードウエアのステップ制御の実現方式の検討
- 故障部位発見や障害装置の切り離しのためのリソース制限下でのシステム稼動方式の検討
- 統計情報収集のための機構の検討

6.2 研究プロジェクトの構成

前節では今後の検討課題について述べた。これらを検討し問題点を解決していくために，設定すべき作業としての研究プロジェクトの構成概要を述べる。

ここで設定する研究プロジェクトは，第5世代コンピュータ開発の前期3年において実行することを考えているので，目的はデータフローマシンの基礎研究及び可能性の検討が主眼となる。従って，プロジェクトを設ける場合の方針を次のようにした。

- 高級言語としては，データフロー向きの汎用言語を第1版の実験機用とし，核言語（第5世代コンピュータの）をベースとするものは実験機第2版で考える。
- 方式検討が中心となるので，様々なソフトウエアシミュレーションを数多く実施するが，それだけでは不十分であり，基本的な方式をもり込んだ実験機を試作する。実験機の実装は通常のＬＳＩやＩＣを用いる。
- 従来の計算機におけるオペレーティングシステム相当の機能は，未だ未開拓の分野が多く，前期で研究すべきテーマとしてはその内，アクティビティ割当て，入出力制御等，データフローマシン固有の機能のみにしぼる。実験機への実装は従って最小限の機能でよい。

設定した研究プロジェクトは次の通りである。

(1) 理論研究

(2) 高級言語の研究

(3) アーキテクチャの研究

　a. 演算ユニット

　b. アクティビティメモリ

　c. 構造メモリ

　d. 結合ネットワーク

　e. 入出力

(4) 制御方式の研究

(5) 実験機の設計と試作（開発）

(6) 性能評価とシミュレーション

　a. シミュレータ作成

　b. 性能評価

　c. 応用研究

これらのプロジェクトは，ほぼ6.1節で示した検討課題と対応するが，その他に次のような関係が

ある。

- 課題中の基礎研究は，上の(1)，(6)に主として関係する。
- 応用と諸方式検討は，(3)，(6)にて行われる。
- 制御ソフトウエアや実装法の検討は，(4)，(5)で行う。
- アクティビティ制御の検討は，(3)－ｂで行う。
- アーキテクチャの検討は(3)で行い，(5)，(6)も利用する。

6.3 研究プロジェクトの内容

6.3.1 データフローマシンの理論研究

A．目 的

知識情報処理を目的とする第５世代コンピュータのアーキテクチャ基盤を与えるものとしてデータフローマシンが十分機能し得るか否かを明確にするために，データフロー計算モデルなど非ノイマン型計算モデルの理論的特性を明確にし，推論メカニズム等知識情報処理の基本メカニズムをデータフロー計算モデルにもとづいて確立する。データフローモデルによる知識情報処理基本メカニズムを明確化する研究を通して，データフロー計算モデルと述語論理型の計算モデルとの親和性を明らかにする。

B．研究内容

a．計算モデルの研究

述語論理型の計算モデルの基本となっている resolution method，natural deduction method をデータフロー計算モデルで捉え，データフロー計算モデルの枠組で述語論理型言語の計算モデルを構築する。この計算モデルは中期に開発される並列型推論マシンサブシステムの理論的ベースを与えるものである。

b．推論基本メカニズムのデータフロー実現方式の研究

推論処理の基本メカニズムであるUnification アルゴリズム，Binding 機構のデータフローモデルによる解釈とその実現機構，パターンマッチング演算機構の明確化を行う。また並列推論におけるバックトラック制御法の確立とそのデータフロー制御による実現法を探る。（本来並列推論では Breadth first method によってバックトラックは不要であるが，現実的には有限資源であるため完全な Breadth first method は不可能である。したがって Breadth first と Depth first を混合した Bounded Breadth first method を構築しなければならない。）

c．非決定性処理と履歴依存性の実現

bのバックトラック制御を実現するためには非決定性処理のための制御命令プリミティブを整備する必要がある。またこのプリミティブの組み合わせによるバックトラック制御実現方式を確立する。これらの制御命令プリミティブは履歴依存性を有することが必要である。履歴依存性プリミティブを許したデータフロー計算モデルを体系化していく。

なお履歴依存性概念は入出力操作，ファイル操作のための計算モデル設定にも必要である。

d．抽象データ型の導入

データ構造化機構，構造データへのアクセス機構としてのデータタイプチェック操作を計算モデルとして明確にするとともに，これらのプリミティブを実現するためのメモリ機構を明らかにし，構造メモリのアーキテクチャ確立のための基礎を作る。特にタグ付メモリのアーキテクチャモデル等を研究する。

また，これらの研究の成果をデータ抽象化機能を盛り込んだ，モジュール化指向の高級言語仕様設計の基礎資料として活用する。

e．推論向データフローマシンのＯＳの研究

並列推論における並列実行枝の爆発を押え，有限資源環境内で適度の数の実行枝を選び起動するための実行制御ソフトウエア方式の研究を行う。この制御ソフトウエア方式の研究の中には，Bounded Breadth first Method の実現法，各資源への実行ボディの割付け制御，負荷分散化制御のアルゴズムを明確にする研究が含まれる。

f．スケジューリング方式とデッドロック問題

分散システムにおいて本質的に生ずるデッドロック問題の理論的研究を行う，デッドロックに関する研究テーマは，デッドロックの生起する確率の理論およびシミュレーションによる解析，デッドロック予防策の確立，デッドロック検出アルゴリズムの確立，デッドロック解消アルゴリズムの確立などである。これらのデッドロック問題を考慮し，かつ各資源の負荷が分散するようなスケジューリン方式を確立する。

C．研究ステップ

前期3年間の間にはこれらの各テーマは机上理論研究，およびシミュレーションによるモデルの検証実験をくりかえすことになる。

推論モデルの研究は５９～６０年に開発・試作される第１版実験機上で実験をくりかえすことになる。

デッドロックの研究は推論用ＯＳに限る問題ではないのでスケジューリング，資源割り付けの問題として最初の段階から始めておく。

		57	58	59
推論モデル		← 計算モデルの構築 →		シミュレーション評価 →
非決定性／履歴依存性		← 計算モデルの構築 →		シミュレーション評価 →
抽象データ型サポート機構		← モデルの構築 →	言語仕様拡張 →	評価 →
スケジューリングデッドロック		← 方式検討 →	アルゴリズム設定 →	評価実験 →
要員	研	2	2	2
	技	0	0	2

研：研究者，技：技術者 （人）　図6.3.1

6.3.2 高級言語の研究

A．目　的

データフロー用高級言語の必要な機能を明らかにし，推論機能の取り込みをはかる。

B．作業内容

a．実験機第1版用高級言語の設計

まず既存のデータフロー言語について調査し，記号処理向データフロー言語の必要な機能を明らかにする。

主な検討項目として，

(1) 文　法

(2) data abstraction

(3) 非決定性

(4) 履歴依存性

等がある。これらの項目を検討し，必要な機能を定める。次にこの機能をデータフローマシン上でどのように実現するかを具体的に検討する。これらは例えば関数呼出し，ループ処理，リカーシブ処理，データ構造処理等であり，この検討で静的にサポートする部分と動的にサポートする部分を決定する。この過程で中間言語の仕様も定める。

次にコンパイラを作成する。第1版はコンパイラ・コンパイラを用いて作業時間の短縮をはかる必要があろう。この作業は文法の記述よりもセマンティクスの記述が中心となる。この作業と並行して簡単なシミュレータを作成し，高級言語の記述テストや論理的チェックを行う。また性能評価用シミュレータ（ 6.3.6 参照 ）を用いて最適化を行う。

b．実験機第2版用高級言語の研究

第2版用高級言語では推論機能の取り込みが中心課題である。まず unification をデータフロー言語上で実現するために必要な命令やデータ構造を検討する。次にそれらをデータフローの計算モデル上でいかに表現されるかを明らかにし，中間言語を検討する。これらの作業は第5世代計算機の核言語設計グループと協力して行う。

C．作業ステップ

(1) データフロー言語の調査

(2) 実験機第1版用高級言語の設計

(3) 中間言語設計

(4) コンパイラ作成

(5) シミュレータ作成

(6) コンパイラ評価，改良

(7) 実験機第2版用高級言語の検討

	57	58	59
計画	(1)言語調査 (2)第1版用高級言語設計 (4)コンパイラ作成 (3)中間言語設計 (5)シミュレータ作成		(6)コンパイラ評価，改良 第2版用高級言語の検討
人員	研 2	研 2	研 2
	技 2	技 3	技 1
計算機	TSS 1000h	TSS 1500h	TSS 1000h
	CPU 50h	CPU 75h	CPU 50h

図6.3.2

6.3.3 アーキテクチャの研究

データフローマシンの各機能モジュールに対応して以下の研究プロジェクトが必要である。

A. EXU（EXecution Unit）

a. 目 的

アクティビティ記憶装置から渡された実行可能命令を入力し，その命令コードに応じて演算を行い，結果をトークンとして出力する機能を有するEXUのハードウエア構成，および命令セットや命令形式の検討を行う。

b. 作業内容

実験機第1版に対しては，命令セット，命令形式，および出力するトークン形式の検討を行い，そのハードウエア構成の方式設計を行い，ソフトウエアシミュレーションによって評価する。この結果は実験機第1版の詳細設計に使用される。

実験機第2版に対しては，上記第1版の評価結果をもとにさらにその高機能化，高性能化の検討を行い，ソフトウエアシミュレーションによって評価し，VLSI化の検討を行う。

c. 作業ステップ

図6.3.3，および表6.3.1にそれぞれ，線表および所要人員と所要計算機時間を示す。

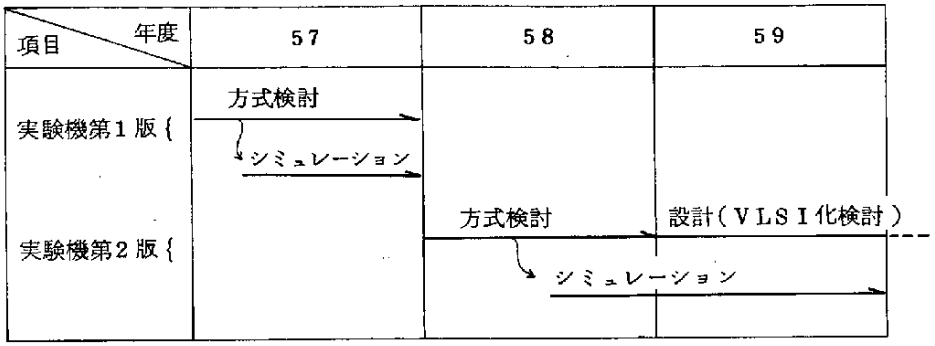


図 6.3.3　線　表

表 6.3.1　所要人員および所要計算機時間

項目＼年度		57	58	59
所要人員	研究者	2人	2人	2人
	技術者	1人	3人	2人
所要計算機時間	端末時間	400h	800h	600h
	CPU時間	20h	40h	30h

B．アクティビティメモリ

a．目　的

アクティビティの制御方式の研究，トークンやアクティビティの表現形式の検討，およびアクティビティ記憶装置のハードウエア構成の検討を行う。

b．作業内容

実験機第1版に対しては，識別子付きトークンを用いる場合の制御方式や必要な命令セットおよび並列ハッシュ法によるアクティビティ記憶装置の方式設計を行い，ソフトウエアシミュレーションによって評価を行う。この結果は，実験第1版の詳細設計に使用される。

実験機第2版に対しては，上記第1版に対する評価結果をもとに，さらにその高性能化のための方式設計とシミュレーションを行うと共に，VLSI化のための検討を行う（必要があれば，例えば連想メモリのようなチップを試作する）。

c．作業ステップ

図 6.3.4 及び表 6.3.2 にそれぞれ，その線表，および所要人員とシミュレーションに必要な

計算機時間を示す。

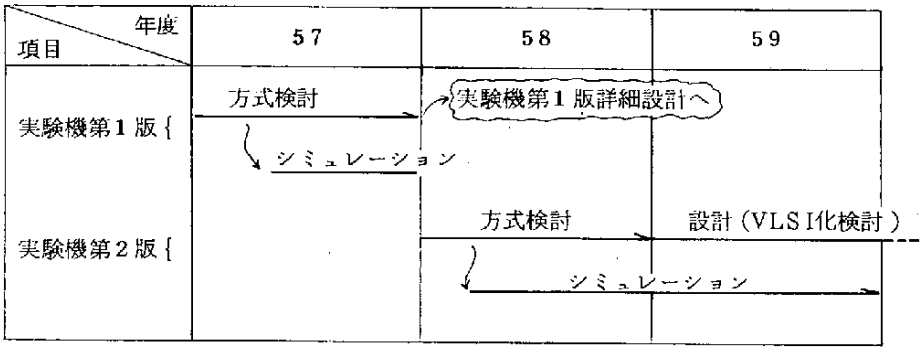


図6.3.4　線　表

表6.3.2　所要人員および所要計算機時間

項目 ＼ 年度		57	58	59
所要人員	研究者	2人	2人	2人
	技術者	1人	3人	3人
所要計算機時間	端末時間	500h	1000h	800h
	CPU時間	30h	50h	40h

C．構造メモリプロジェクト

(1) 目　的

構造メモリの内部構造およびデータフローマシンの全体構造における構造メモリの立場を明確にする。

(2) 作業内容

構造メモリはデータフローマシンを構成する上で重要な機能モジュールである。

本プロジェクトは，実験システムの開発に先行し，また並行して構造メモリの研究および試作・評価を行う。本プロジェクトの主な作業は，前期実験システムの構造メモリ（第1版）の設計と，中期以降に開発を予定している本格的推論マシンの要素技術となる構造メモリ（第2版）の検討である。

①　前期実験機用構造メモリの設計

前期実験機用構造メモリの設計はデータフローマシンの全体構造の基本方式設計から出発

する。これは前期実験機プロジェクトおよび他の個別検討プロジェクトと合同で行われる。構造メモリに関しての主な検討事項は次のとおりである。

(i) 構造メモリが実現する基本データ構造

(ii) 構造データ操作命令

（構造メモリと処理装置とのインタフェース）

(iii) 要求性能

（スループット，遅延，メモリ容量等）

これらの項目については，これまでの検討結果を基にその実現方式をとりあえず決定する。

構造メモリプロジェクトではこの構造メモリの基本方式をさらに検討するために2つの作業を行う。1つは構造メモリのバンク（構造メモリユニット）の内部構造の検討であり，もう1つはその並列化の検討である。前者の目的は，主に前述(i)(ii)の確認であり，後者は(iii)に対応する。

構造メモリバンクの検討の主な作業は，シミュレータの作成とそれを用いた評価である。シミュレータ上には，1つの構造メモリバンクが持つべき機能（操作，共有，ガーベジコレクション等）を細部にわたって実現する。シミュレータ自体はソフトウェアシミュレータが中心となるが，必要に応じて既存のマイクロプロセッサ等を用いたハードウェアシミュレータの作成もあり得る。

並列化の検討はソフトウェアシミュレータを作成して行う。ここでは，操作命令のスループット，ネットワークのトラヒック，メモリ容量等について評価を行う。

前期実験機用の構造メモリは，上記2つの作業によって得られた評価結果にもとづいて詳細設計を行い，実験機プロジェクトに引き渡される。

② 構造メモリの第2版の検討

構造メモリの第2版の設計では，性能面（バンク数，メモリ容量）での第1版の拡張とともに，構造メモリの階層化等の機能面での拡張が重要となる。このため，第1版の設計で得られた評価結果だけでなく，他プロジェクトからの評価結果を十分に取り入れ，構造メモリのイメージ設計から再度出発する。

性能面における拡張の検討では実装法に関する考慮が重要となる。中期以降に確立されると思われるＶＬＳＩ技術を予測した上で，構造メモリバンク，バンク間のネットワークなどのハードウェア構造を再検討する。

機能面における拡張の検討では，構造メモリ，スワップ用2次記憶，ファイル機構について，データフローマシンの全体構造における位置付けを考える必要がある。

イメージ設計の後は詳細にわたる実現方式の検討のためのソフトウエアシミュレータの作成を行う。本プロジェクトの最終的設計は，シミュレーション結果を用いて細部を練り直した上で，ＣＡＤシステムを用いて行われる。

(3) 作業ステップ

① 前期実験システムの全体構造の検討

実験機プロジェクト，他の個別研究プロジェクトと合同

② 構造メモリの基本設計

③ 構造メモリバンクの設計と評価

- 構造メモリバンクの設計
- ハードウエア／ソフトウエアシミュレータの作成
- シミュレータを用いた評価

④ 構造メモリの並列化の検討

- 並列型ソフトウエアシミュレータの作成
- 要求性能に関する評価

（操作命令のスループット，ネットワークのトラヒック，遅延，メモリ容量）

⑤ ③④の評価結果にもとづいた前期実験機用の構造メモリの詳細設計

⑥ 構造メモリ第2版のイメージ設計

- 第1版の拡張
- 実装法の考慮

⑦ 第2版用構造メモリのソフトウエアシミュレータの作成と評価

⑧ 第2版構造メモリの設計

(4) 線　表

構造メモリプロジェクトの作業ステップ，所要人員，予想される計算機使用時間等を図6.3.3に示す。

D．結合ネットワーク

a．目　的

用途別および規模別ネットワークアーキテクチャの研究試作，およびネットワークの実装法の研究。

b．作業内容

ネットワークアーキテクチャの研究は，規模の点で，第1版実験機用の小規模システム（最大＋数台）のものと，第2版実験機用の中・大規模システム（数百台）のものとに分かれ，本

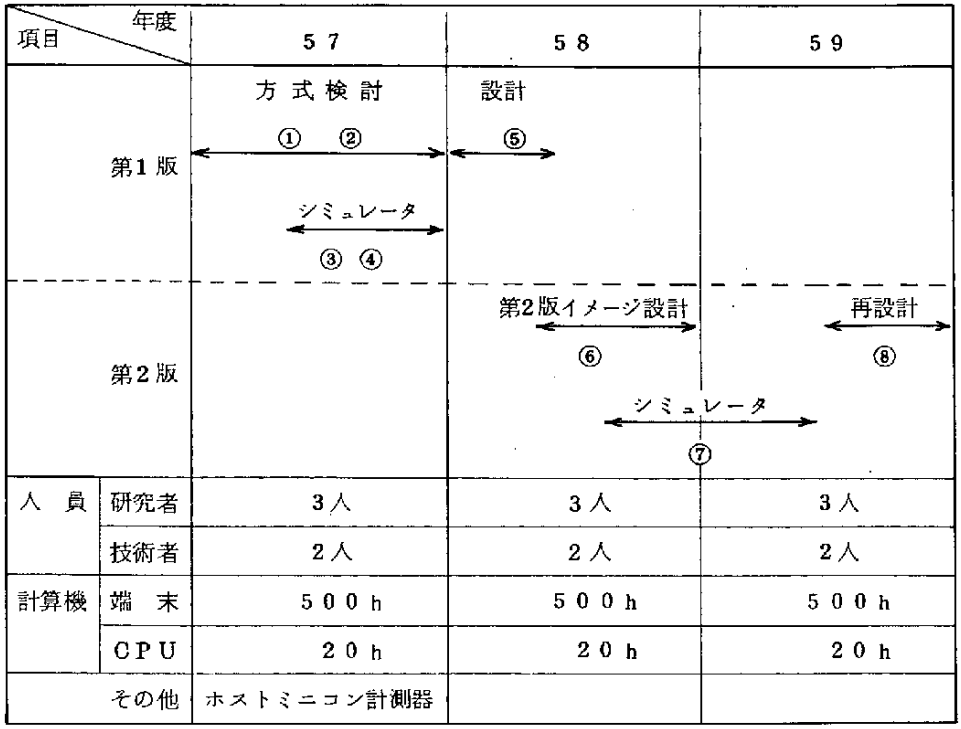


図 6.3.5

格的な研究は，スケジュールの関係より中・大規模システムを目標に行う。

用途別の研究内容について述べる。Anet，Dnet はシステムでおそらく一番負荷が高く性能に直接影響する部分であるので，高スループットをもち，遅延の小さいネットワークアーキテクチャの研究が重要である。また，Dnet に局所性を導入することはネットワーク全体の複雑度を規模にかかわらず，一定に抑えることができる他に，アクティビティの局所性と対応させることにより，処理を高速化することができるので，この局所性をもつトポロジーについては，応用にもとづいて十分に研究する必要がある。

ＳＭ－AMnet，AM－ＳMnet は，ＳMUの研究と合同で研究する必要がある。記号処理においては，ＳMUへのアクセスが高頻度であることが予想されるので，ＳＭ－ＡＭnet，ＡＭ－ＳＭnet についてもスループット，遅延ともに高性能のものが必要である。

Inter SMnet については，扱うデータが大きなものであり，要求頻度もそれほど大きくないと考えられるので，回線交換のネットワークが向いていると考えられる。数百台の構成に対し，どのようなトポロジーが適しているかについては，多くの研究を要する。

プログラム転送用ネットワークは，扱うデータのサイズが大きく，転送頻度は低い。Inter

ＳＭnet と同様に回線交換が適していると考えられるが，数百台のＡＭＵと１台（あるいは数台）のプログラムメモリを結合するネットワークのトポロジーには，何らかの工夫が必要と考えられる。

ネットワークアーキテクチャの研究としては，高スループットで遅延の少ない高性能ネットワークの研究と，局所性をもったネットワークにおけるネットワークノードスイッチの通信制御機能の研究が重要である。高性能ネットワークの研究は，クロスバーや多段スイッチングネットワークを基本モデルとして制御方式や交換方式の検討を行い，いくつかの設計について，ソフトウエアシミュレーションにより性能評価をし，設計を再検討し最終的には百台前後のプロトタイプを作成することを目標とする。また，局所性をもったネットワークについては，各ノードスイッチが通信の中継，複雑なルーティング等を行う高機能のものになるので，このノードスイッチの通信制御機能の検討がポイントになる。基本的には，このノードスイッチの結合の仕方と適当なルーティングアルゴリズムにより，種々のトポロジーを組むことができるので，このノードスイッチの汎用性は高く，アクティビティの局所性に合致したネットワークトポロジーの局所性を研究する際に有用である。

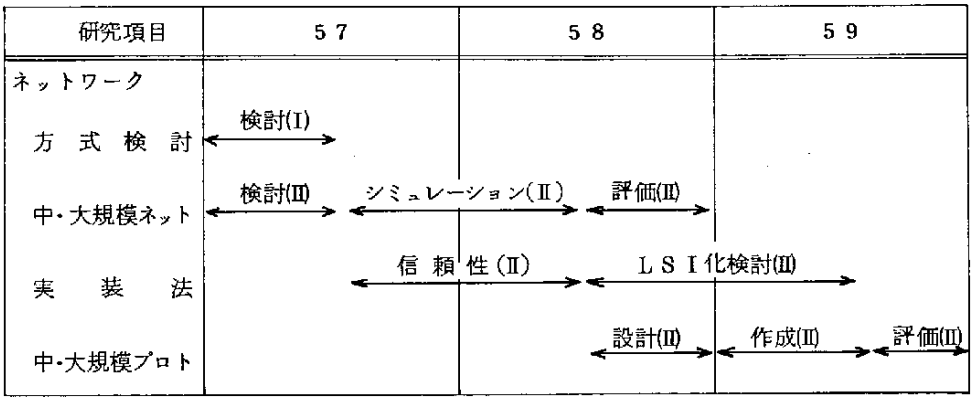
実装法に関連する研究としては，ＬＳＩ化に関するものと高信頼化に関するものがある。ＬＳＩ化は設計の段階で考慮すべきものであるが，これと関連するものとしてビットスライス化の研究がある。高信頼化については，故障の検出方法と故障からの自己回復が問題となる。対策としては，パリティチェックとデータ再送等が考えられるが，再送不能となるような故障の場合には，障害ユニットを回避するルーティング法等の検討が必要である。

c．作業ステップ

作業ステップについて，図6.3.6にもとづいて述べる。第１版実験機用小規模ネットワークについては，５７年度の前半に検討し，後半には，その結果を実験機作成プロジェクトへ渡す。第２版実験機用中・大規模ネットワークについては，５７年前半に用途別に検討を行い，いくつかの有望なネットワークアーキテクチャについて，後半に性能評価のためのネットワーク単体のソフトウエアシミュレーションを行う。５８年前半には，他の機能モジュールと結合して，全体シミュレーションを行い，トポロジーなど，他のモジュールの機能に依存する部分についての検討を行い，後半には，百台前後の規模のプロトタイプについて，実際にハードウエアの設計を開始する。５９年度前半より作成を始め，後半には，ハードウエア性能の評価を行う。また，ネットワーク実装法の検討については，５７年度後半より信頼性の研究を開始し，前述の百台前後のネットワーク設計の際に成果を反映させることを目ざす。５８年度後半からはネットワークのＬＳＩ化について検討し，プロトタイプネットワークについて，ＬＳＩ版の設計

を行う。

必要な人員，並びに，研究開発環境として必要なものを図6.3.7に示す。



(注) (I)は第1版用，(II)は第2版用であることを示す。

図 6.3.6

年度	57	58	59
人員 研究者	1人	1人	1人
人員 技術者	1人	2人	3人
端末使用時間	800h	1600h	400h
CPU タイム	50h	200h	20h
計測器	—	—	1セット
CAD使用時間	—	—	400h

図 6.3.7

E．入出力

a．目 的

データフローマシンにおける入出力の概念，動作を明らかにし，基本的な入出力処理方式を決定すること，および，知識情報処理システムを考慮した入出力機能の高度化を検討する。

b．作業内容

(1) 入出力の基本方式の検討

実験機第1版に向けて，入出力の基本機能の実装のために，次の項目に関して基本方式検

討を行う。

- 割込とＯＳ：入出力割込を含む割込全般に関して，データフローマシンにおける考え方を明らかにする。これには，制御ソフトウエアの研究プロジェクトと協力してＯＳの立場から見る必要がある。
- 入出力命令：少なくとも，端末，ファイルに関して，操作命令を明らかにする。
- 入出力動作：入出力の一連の動作を，システム内の各要素（ＡＭ，ＳＭＵ，ＩＯＰ）およびトークンと関連づけて詳細化する。

これらの方式検討の成果は，実験機第１版に反映され，検討が引きつがれる。また，本検討にもとづいて，入出力部分のシミュレーションを行う。シミュレーションには，ミニコンピュータ等をＩＯＰかわりに使用し，ソフトウエアシミュレータと結合して行うことが望ましい。本シミュレーションを通じて，入出力の基本動作の確認，方式の評価を行い，実験機第２版へ反映させる。

さらに前期の後半では，シミュレータの機能拡張，改良を行い再評価する。

(2) 入出力の高機能化の検討

基本動作の確認を終えた後，次の段階として，入出力処理をさらに高いレベルでとらえ，データフローマシンと知識情報処理との関連，および，より完全な自立型マシンにするために，次の項目について，方式検討を開始する。

- ストラクチャメモリの階層化：実用的なマシンにするために，高速化と大容量化を考え，ファイルとのインタフェース方式を明らかにする。
- データベースとの関連：ファイルの高機能化としてデータベースマシンを考えた時の機能分担，インタフェース方式を明らかにする。
- 入出力機能の見直し：入出力処理を徹底してデータフローマシンに取り込む方式を検討する。

c. 作業ステップ等

作業の線表，所要人員，所要計算機時間を図 6.3.8 に示す。

	57	58	59
計画	方式検討 （基本動作）	実験機第1版へ反映 ハード／ソフト シミュレータによる評価 方式検討 （SMの階層化）	機能拡張，改良 方式検討 （データベースとの関連） （入出力機能見直し）
人員	研 2人 技 1人	研 2人 技 4人	研 2人 技 2人
計算機	•TSS（CPU Time） 50h	•TSS 200h •ミニコン	•TSS 200h •ミニコン

図6.3.8　入出力プロジェクトの線表，所要人員等

6.3.4　制御方式の研究

A．目　的

アクティビティ割り付けの方式検討，ローダ・スケジューラの開発，およびデータフローマシン用OSの研究。

B．作業内容

アクティビティ割り付けの研究は，第1版実験機のためのアクティビティ割り付け方式の検討およびローダスケジューラの開発と，第2版実験機のためのアクティビティ割り付け方式の検討に分れる。

アクティビティ割り付けの研究項目としては，手続きやループボディ等のコードブロックの複数AMUへの展開方式，複数コードブロックの配置，およびインスタンスの生成方式があり，すべてDnetのトポロジーと関係している。トポロジーに局所性がない場合は，割り付けの際に幾何学的配置については考慮する必要がなく，並列アクティビティを複数処理装置上に展開することだけが重要となる。これに対し，トポロジーに局所性がある場合には，問題の性質に合致した幾何学的配置により，処理の高速化が図れる。問題に含まれる局所性としては，コードブロック内の局所性とコードブロック間の関係における局所性がある。コードブロック内の局所性および並列性をともに効果的に利用できる小規模AMU群のトポロジーと展開方式について研究する必

要がある。また，コードブロック間関係の局所性と対応させることが可能な上述の小規模ＡＭＵ群間のトポロジーとコードブロックの配置についても研究する必要がある。これらのトポロジーと割り付けの問題は応用に深く依存するものであるから，ソフトウエアシミュレータにより，その応用分野への適合性を評価するとともに詳しく検討する必要がある。

ローダ・スケジューラについては，第1版実験機への実装を目的として研究開発する。データフローグラフの物理アドレスへの変換，ダイナミックリンクの方式，システム資源管理および割り付けアルゴリズムの研究が中心となる。ローダスケジューラについても，ソフトウエアシミュレーションにより，その動作の評価検討をする必要があり，これらがソフト的ボトルネックになって応用プログラムの並列・非同期な性質を無効にしないよう特に注意が必要である。

データフローマシンのＯＳの研究は，資源管理，実装方式，スケジューリング，割り込み，再構成手法等の課題をもっている。資源管理，割り付けスケジューリングについては，データフローマシンのＯＳの重要な基本機能であるので，より一般的立場から研究する必要がある。また，ＯＳの実装方式についても，大量資源管理に向った分散実装について研究する必要があり，Id のマネージャ関数のようなオブジェクト化についても研究を要する。割り付けスケジューリングについては，ＴＳＳ，バッチ，リアルタイム等の性質の異なるジョブや特権ジョブなどのプライオリティに応じたスケジューリングについて研究する必要がある。割り込み処理については，従来行われているものについて，プロセスの中断を必要とするものと，単にそれを処理するプロセスを新たに生成するだけで良いものとに分類・整理し，大量の資源を利用したプライオリティスケジューリングとの関連での対処を検討する。再構成については，動的なエラー検出と復旧が重要であり，独立・非同期に動作している多数のユニット，およびそれらの間に分散しているアクティビティの障害時における管理方法を研究する必要がある。

C．作業ステップ

作業ステップについて，図6.3.9にもとづき説明する。アクティビティ割り付けの方式検討は５７年より５８年まで行い，初年度は第1版実験機のための基本アルゴリズムの検討を行い，この成果をローダ・スケジューラの設計に利用する。５８年度は，中・大規模の第2版実験機を想定した方式検討を行い，局所性を利用した割り付け法等について，全体シミュレータを利用し，検討評価を行う。データフローマシンのＯＳの研究は，この結果を引き継ぐ形で，５８年度より開始し，より一般的な資源管理方式，割り付けスケジューリング等について研究する。そして，59年度には，実際に設計を行うことを目標とする。ローダ・スケジューラの開発は，５７年度より始め，前半では，アドレス変換方式，リンク方式，資源管理法等のアルゴリズムについて検討し，後半より設計を開始する。５８年度はこのローダ・スケジューラを第1版実験機のソフトウエアシ

ミュレータに実装することにより，種々の評価を行い，再設計を行う。５９年度は，第１版実験機に実装し，後半より評価を行う。

必要な人員，ならびに研究開発環境として必要なものを図6.3.10に示す。

研究項目	５７	５８	５９～
アクティビティ割り付け方式検討	基本アルゴリズム(I) ←→	シミュレーション(II) ←→	
ローダ，スケジューラ	検討(I) ←→ 設計(I) ←→	シミュレータに実装(I) ←→	実験機実装(I) ←→ 評価(I) ←→
ＤＦＭ用ＯＳ		方式検討(II) ←→	方式検討(II) ←→

註 (I)は第1版用，(II)は第2版用であることを示す。

図6.3.9

年　度	５７	５８	５９
人員 研究者	２人	２人	２人
人員 技術者	２人	３人	２人
端末使用時間	１０００ｈ	２０００ｈ	８００ｈ
ＣＰＵタイム	１００ｈ	３００ｈ	８０ｈ
計測器	—	—	—
ＣＡＤ使用時間	—	—	—

図6.3.10

6.3.5 実験機の開発

A．目　的

データフロー方式が第５世代コンピュータを支えるハードウェアアーキテクチャとして有効なものかを明らかにし，多数モジュールを結合したシステムでの資源割り付け，スケジューリング等の分散制御方式を確立するために，実験機の設計・試作を行う。また試作された本実験機をツールとして用い，関数型プログラミング方式，述語論理型プログラミング方式，等のソフトウエア方式ならびに高度並列処理アルゴリズムの実験，開発を行う。

さらに並列推論機構の実現方式，unification用演算モジュールの構成法等を明らかにしてゆくための実験環境を与えることも本実験機開発の目的である。

B．作業内容

a．ソフトウエア・言語方式の仕様設定

5.1.2で示した言語仕様を更に具体化し，そのコンパイラ，アセンブラを開発支援用ホストコンピュータシステム上で開発する。また，ホストコンピュータから実験機上へのプログラムローダを開発する。

b．記号処理を適用領域として，aの高級言語によるプログラムの実行に適合したマシンアーキテクチャを設計する。具体的にはハードウエア命令セットの仕様設計，各命令の実行方式，トクンの識別制御・管理方式，スイッチゲート等の実行制御命令の仕様設計を行う，演算命令はリスト演算命令等構造データ処理用の命令セットとシグナルデータ，数値データ等単一データの演算命令セット，またファイル操作，入出力操作，さらには非決定性実行制御のための命令セットを設計する。

c．実験機の各要素ユニットの設計・試作を行う。設計・試作する要素ユニットはアクティビティメモリユニット（ＡＭＵ），演算ユニット（ＥＸＵ），構造メモリユニット（ＳＭＵ）であり，これらは各々８～１６個分を設計・試作する。またこれらの要素ユニットを結合するためのネットワークを設計・試作する。ネットワークは多段結合ネットワーク或いはクロスバー方式ネットワークとして，それぞれのスイッチエレメントを試作する。

d．データ構造の設定，構造データへのアクセスアルゴリズム，ガーベジコレクションアルゴリズムを定め，構造メモリユニットの設計・試作を行う。入出力，ファイル処理についてはホストコンピュータで実行させるので，演算ユニットとホストコンピュータのインタフェースを設計・作成する。

e．実験機ハードウエアの各ユニットを並列に動作させ，各ユニットの負荷を分散させて均一化するための資源割り付けアルゴリズムを明確にし，スケジューラ，ローダを含むデータフローマシン用制御ソフトウエア（ＯＳ）を設計・試作する。

f．各要素ユニットの設計段階において，シミュレーションにより設計の検証を行うとともに実験機の性能予測を行って設計にフィードバックさせる。

g．実験機上でのプログラミングをサポートするためのソフトウエア支援システムを開発する。特にデータフローマシン上でのデバッグを支援するためのデバッグユーティリティの開発を行う。

h．設計・試作された実験機上でベンチマークテストを行い方式の有効性を検証する。特に実験

機上に Prolog 処理系をインプリメントし推論処理用のハードウエアとしての有効性を明らかにする。また実験機のハードウエア量を算出し，性能・価格面からの評価を行うとともに，中期実験機（推論マシンサブシステム）試作におけるＶＬＳＩモジュール設計のための基礎データを抽出する。

C. 作業ステップ

a．方式設計

- 高級言語仕様設定
- 命令セット設計
- データ構造設計
- 制御ソフトウエア方式設計

b．詳細設計

- AMU，EXU，SMU，Network 各2ユットの個別設計
- コンパイラ設計・作成
- 対ホストインタフエース設計

c．製造，調整

- ネットワーク組み立て

d．評　価

- ベンチマークプログラム作成
- データ収集・解析

項目＼年度		57	58	59
ハード		方式設計 ← →	詳細設計 ← →	製造・調整 ← → 評価 ← →
ソフト		言語仕様設定 ← → コンパイラ製作 ← →		ベンチマークプログラム作成 ← →
		制御ソフトウエア方式設計 ← →		評価 ← →
人員	ハード 研	3人	3人	2人
	ハード 技	2人	5人	5人
要員	ソフト 研	2人	2人	3人
	ソフト 技	2人	3人	3人

図 6.3.1 1

6.3.6 性能評価とシミュレータ

A．目　的

シミュレータ作成の目的は，

(1) 言語の開発のサポート

(2) アーキテクチャの性能評価と改良

(3) 論理チェックとデバッグ

に大別される。これらはそれぞれ目的が異なるため同じシミュレータを用いると，それぞれの目的に不必要な部分のシミュレートも行うため非常に効率を悪くする。シミュレータの実行時間は実際のマシンに較べて1,000倍以上遅いのが普通であり，実用的なシミュレータを作るためには効率の問題が重要となる。従ってそれぞれの目的に応じたシミュレータをそれぞれの目的にあった言語で記述する必要がある。

B．シミュレータの作成と性能評価

a．言語開発用シミュレータ

このシミュレータは設計したデータフロー言語により色々なプログラムを作り，実行し，取り入れた機構の論理的妥当性をチェックし，設計にフィードバックするためのものである。たとえばストリームや構造メモリ用の命令を設計した時，それらがどのように働くかをチエックするのはこのシミュレータの役割である。言語の設計が終った時点ではデータフローマシンのハードウエアはまだ完成していないことが予想されるし，できあがったばかりのマシン上で新しい言語を実行するとエラーが起きた時ハードウエアとソフトウエアのどちらが悪いのか判定が難しいためこのシミュレータが必要である。

このシミュレータには2つの方式が考えられる。第1はデータフロー言語をデータフローグラフに変換し，そのグラフを解釈実行するものである。この方式は低いレベルでの論理チェックができるが実行速度は次に述べる第2の方式より遅い。第2の方式はデータフロー言語を既存の高級言語（PASCAL，LISP……etc）に変換する方式で高いレベルの論理チェックができ，実行速度が早いという特徴がある。どちらを採用するかはその時点の facility 等を考慮して決定すべきであろう。

記述言語としては対話型であること，データ構造のサポートが容易であること等が要求される。候補として Lisp をあげておく。

b．性能評価用シミュレータ

このシミュレータは全体の構造をシミュレートするものと個別の機能をシミュレートするものにわかれる。個別機能としては各ユニット，ネットワーク，構造メモリ等がある。

(1) 全体構成用シミュレータ

アクティビティメモリユニット，実行ユニット，構造メモリ，ネットワーク等の大きな要素の入出力関係と処理時間を記述する。シミュレーションの目的によってはユニット内のサブユニットについて同様の記述をする必要があるかもしれない。このシミュレータでは次の項目を測定する。

1) トークンの発生数と局所性

コンパイラや命令セットの改良，ユニットや通信系のボトルネックの発見と改良等を検討する。

2) 実行命令数

ノイマン型との比較を行い，データフローのオーバヘッドを明らかにし，コンパイラの最適化，命令セットの改良，機能のハードウエア化等を検討する。

3) 並列処理度

理論値と実測値を比較し，その差の原因を明らかにし，改良を行う。これはスケジューリング，通信系，実行ユニット数，メモリ構造等アーキテクチャ全体と関係がある。

4) メモリアクセス数と局所性

記号処理では構造メモリへのアクセスが非常に重要な要素と考えられる。この測定によりメモリ構成（分散か集中か等）やロジックインメモリ等について検討する。

5) ユニット数と性能

記号処理に必要なユニット数を検討する。これは3)の並列処理度と関係が深い。

6) ユニットの速度と性能

各ユニットの中でボトルネックとなるものを明らかにし，そのユニットの改良や高速化を検討する。

以上の測定の他にこのシミュレータではスケジューリングアルゴリズムの検討を行う。これはプロセスやデータ構造の割り付け，トークンの行先決定メカニズム，入出力の方法等について様々な方法を実験し，スケジューラの構成とアルゴリズムを決定する。

(2) 個別機能用シミュレータ

これは個別ユニットの構成や制御方式を研究するために必要なシミュレータである。記述レベルはレジスタトランスファレベルが適当であろう。たとえばアクティビティメモリと処理ユニットについて考えると，入力トークンの処理過程をレジスタやバッファに関する操作，フィールドの処理，連想過程，命令のフェッチと実行，出力トークンの作成等を記述することになる。このシミュレータ上でトークンのカラーの管理法，待ち合わせ機構，プログラム

の共有法，データのコピー，ガーベッジコレクション，関数引数の渡し方等のアルゴリズムを検討する。

個別機能用シミュレータは全体構成用シミュレータとインタフェースを合わせておき，その中にはめ込んで使用できるようにしておく。このシミュレータでの測定項目は以下の通りである。

1）稼動率

各ユニットがどの程度使用されているかを調べ，各ユニットの数と各ユニットに要求される速度等を検討する。

2）待ち行列の長さ

各バッファの使用状況とトークンの待ち時間を調べ，バッファサイズの決定やボトルネックの発見と解消に役立てる。

3）マッチング回数

アクティビティメモリで待ち合わせを行うトークンの数や待ちの状況を調べ，アクティビティメモリの容量や処理速度の設計に役立てる。

4）通信系

データフローマシンで通信系がボトルネックとなる可能性は十分にある。米国California 大Irvine 校では階層バス構造のデータフロー計算機のシミュレーションを行い，数十台程度までは性能が改善されることを示しているが，その場合のボトルネックはネットワークであった。ネットワークについてはクロスバー，パケットスイッチ，トリー，バス等色々な提案が行われており，どれが適当であるかをアプリケーションやマシンの規模を考慮しつつシミュレーションで決定する必要があろう。ここで考慮すべきことは，

1）遅延時間

2）スループット

3）転送レート

等であろう。

シミュレータの記述言語に要求される点は第1に並列性の記述が容易なこと，第2は各ユニットの統計データが取り易いこと，第3にデバッグ機能が豊富なこと等であろう。マルチプロセッサのシミュレータではデバッグに費す時間がかなり大きいと予想される。以上の点を考えると候補として SIMULA，Concurrent PASCL等が考えられよう。またデータフローハードウエア記述用言語の開発も考えられる。

c．作業ステップ

(1) 評価基準プログラム

シミュレーション評価の基準となるプログラムを作成する。これは応用研究グループと協力して行う。

(2) 評価項目決定

(3) 言語開発用シミュレータ作成とデータ収集

(4) 実験機第1版方式検討用シミュレータ作成とデータ収集

(5) 実験機第2版用個別機能シミュレータの作成とデータ収集

この作業を個別研究グループで行うか，このグループで行うかの調整が必要である。

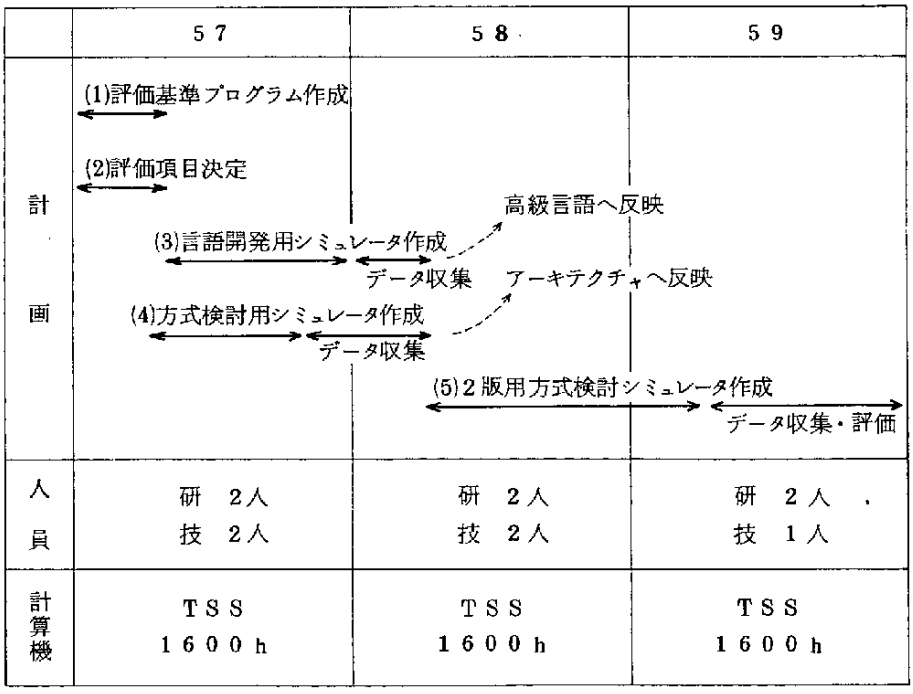


図6.3.12

C．データフローマシンの応用研究プロジェクト

a．目 的

データフローマシンの応用としてどのようなものが適当であるかを明らかにし，さらによりデータフローに適したアルゴリズムの開発を行う。また推論機能やデータベースとのインタフェースを明らかにする。

b．作業内容

(1) 応用分野と並列度の調査はまずデータフロー言語により実用規模の応用を大量に記述する

ことから始める。これは他グループが応用プログラムを使ってテストする際のテストプログラムともなる。

次にこれをシミュレータで実行し，並列度の測定を行う。同時に応用を分析し理論的に並列度の限界を調べておき両者を対照し，取りあげた応用がデータフローに適しているかどうか検討する。

(2) 新アルゴリズムは(1)の検討をもとにさらに並列度の高いアルゴリズムを基礎理論に戻って研究する。また新たな計算モデルについても検討する。

(3) 推論機能の取り込みはまず unification をデータフローマシン上でソフトウエアにより実現することから始める。次に推論研究の成果を順次データベースマシン上でソフトウエアにより実現する。この過程から推論機能に必要なインタフェースをアーキテクチャの観点から検討する。データベースとのインタフェースはデータベースマシングループの研究成果を取り入れる形で検討する。次にソフトウエアシミュレータを作成してインタフェースを具体的に検討する。

c．作業ステップ

(1) 応用分野と並列度の調査

(2) 新アルゴリズム研究

(3) 推論機能，データベースのインタフェース検討

(4) シミュレーション

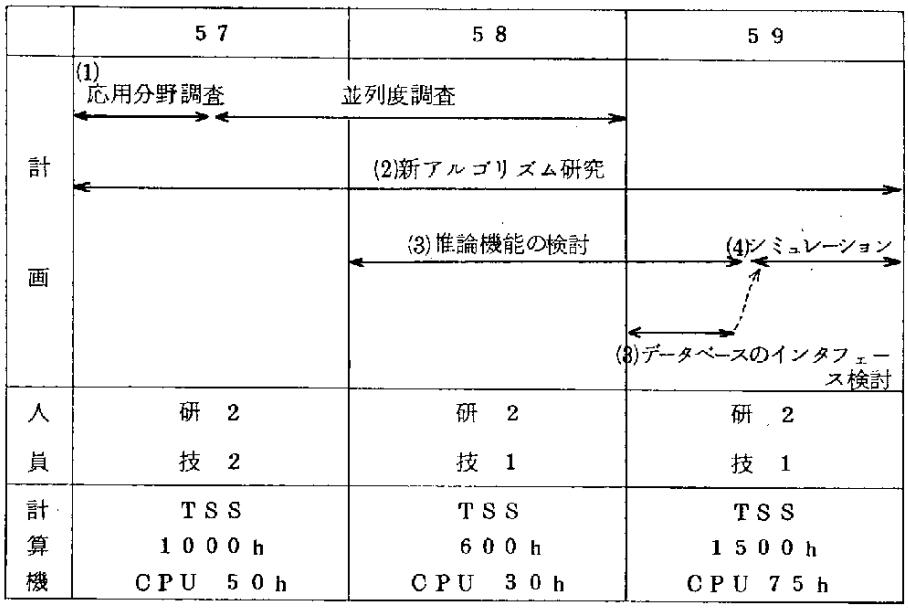


図6.3.13

7. 研究開発計画と体制

本章では，データフローマシンの各研究プロジェクトのスケジュール，所要人員，および必要とされる支援環境について概観する。

7.1 研究開発のタイムスケジュール

データフロー実験機システム第1版の研究開発プロジェクト，ならびに理論研究，高級言語，アーキテクチャ，制御方式　シミュレーション，応用研究のタイムスケジュール一覧を表7.1.1に示す。

7.2 研究開発体制

各研究プロジェクトを効率良くすすめるために必要な開発支援環境について，ソフトウエア，ハードウエア開発の観点から述べる。表7.2.1には，所要人員，計算機使用時間，および器財一覧を示す。ここで，計算機使用とは，計算能力5ＭＩＰＳ程度の大型計算機をＴＳＳで利用することを前提とする。

7.2.1 ソフトウェア開発支援環境

使用する計算機については，大型機をＴＳＳで用いるか，あるいはスーパーミニコンクラスをＴＳＳ，または個人用として用いる等議論のわかれるところであろうが，いずれにせよプログラム開発，保守の点から使いやすさを第一の基準として機種の選択を行うべきである。ここで，使いやすさとは具体的には，

- 高級言語をサポートしていること
- 良いエディタを有していること
- デバッグ機能が十分なこと
- ファイル，入出力操作が容易なこと

等があげられる。

特に，シミュレータ，高級言語コンパイラの開発においては，限られた人的資源を有効に用いるためにも高級言語と良いエディタがサポートされていることは必須であり，高級言語としては次のものが利用できることが望まれる。

- LISP
- PASCAL
- C
- SIMULA

7.2.2 ハードウェア開発環境

データフロー計算機は，従来にないアーキテクチャを採用しているので，実際のハードウエア製作に入る前に，十分に詳細な検討を行っておく必要がある。また，マルチプロセッサ構成であるために，論理的には少ない誤りでも実際の改修作業にはかなりの手間がかかることも予想される。

従って，ハードウエア設計段階においてなるべく誤りを検出・修正しておくことが必要であり，このためにはデザインオートメーション（DA）技術が不可欠である。ここで求められているDA技術には，

- 論理シミュレーション
- 実装設計の自動化
- 診断およびテスタ機能

があげられる。このなかでも論理シミュレーション技術は特に重要であり，単なるハードウエアのデバッグばかりでなく，マイクロプログラムを含めたファームウエアレベルでのデバッグをも行えることが必要である。

ここで，DAシステムと効果的に対話を行うためにはグラフィック端末が必要であり，またソフトウエアの開発やデバッグ等においても適当なプログラム情報を2次元的にグラフィック端末上に表示しつつ処理を進めることは有効であろう。特に，データフローマシンは並列処理を主体とするものであるため，処理の流れを2次元的に表現する等によるユーザ援助が望ましい。

表7.1.1　研究開発スケジュール（その1）

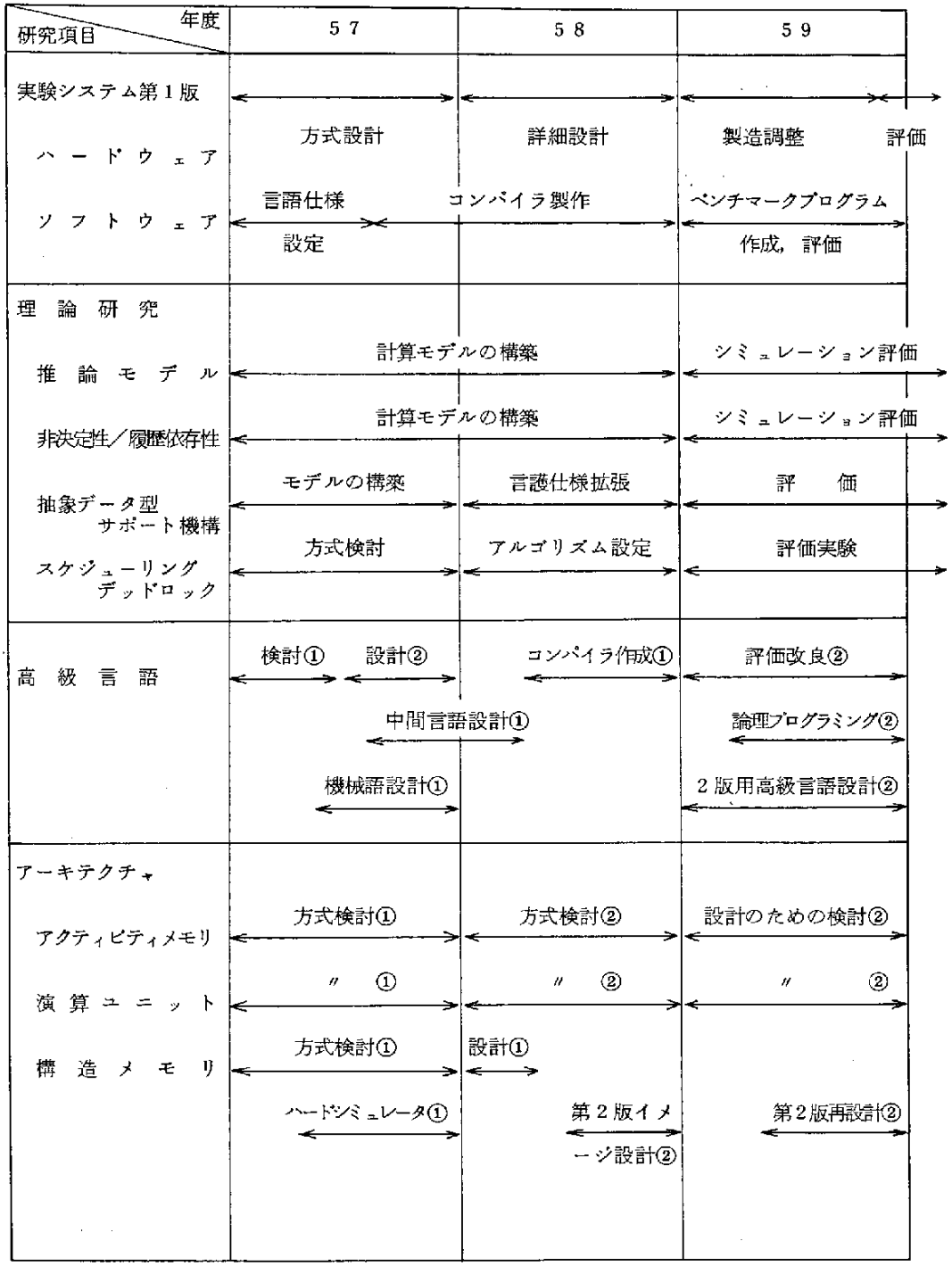
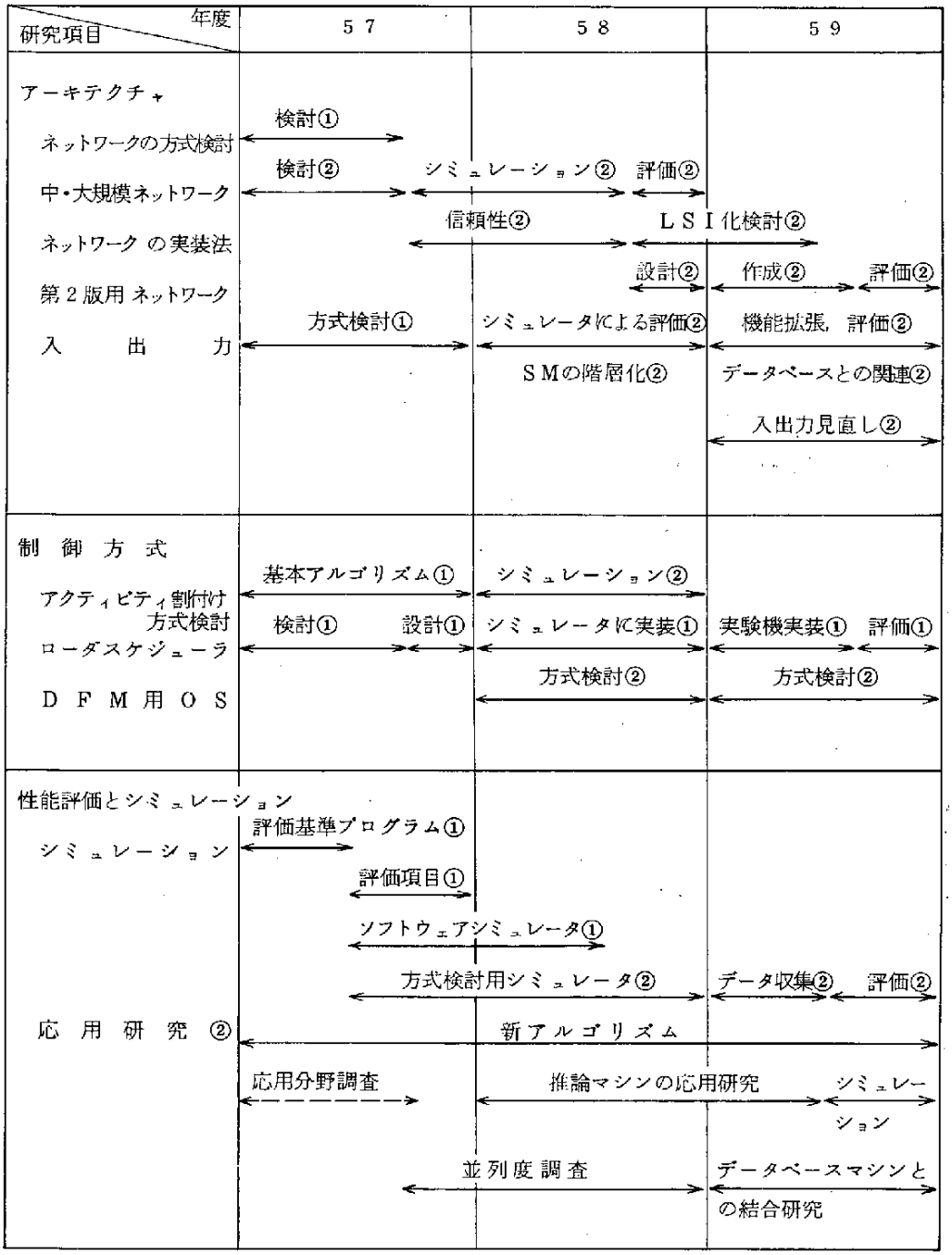


表7.1.1　研究開発スケジュール（その２）



(注)　表中で，①　実験機第１版用を対象とするもの
②　〃　第２版　〃

表7.2.1　人員，計算機使用時間一覧

研究項目	年度	研究者	技術者	計算機使用時間 端末	計算機使用時間 CPU	備考（その他器財一覧）
実験システム第1版						• DA, CADシステム
	57	5	4	1000	70	• 計測器
	58	5	8	1500	100	• ホストーミニコン
	59	5	8	800	50	• グラフィック端末
理論研究	57	2	0	500	20	• エディタ
	58	2	0	500	20	• 高級言語
	59	2	2	1000	50	• コンパイラジェネレータ
高級言語	57	2	3	1000	50	
	58	2	3	1500	75	
	59	2	1	1000	50	
アーキテクチャ						
アクティビティメモリ	57	2	1	500	30	
	58	2	3	1000	50	
	59	2	3	800	40	
演算ユニット	57	2	1	400	20	
	58	2	3	800	40	
	59	2	2	600	30	
構造メモリ	57	3	2	500	20	
	58	3	2	500	20	
	59	3	2	500	20	
ネットワーク	57	1	1	800	50	
	58	1	2	1600	200	
	59	1	3	400	20	
入出力	57	2	1	1000	50	
	58	2	4	4000	200	
	59	2	2	4000	200	
制御方式	57	2	2	1000	100	
	58	2	3	2000	300	
	59	2	2	800	80	
シミュレーション	57	2	2	1000	80	
	58	2	2	1000	80	
	59	2	1	1000	80	
応用研究	57	2	2	1000	50	
	58	2	1	600	30	
	59	2	1	1500	75	
小計	57	25	19	9300	540	
	58	25	31	15600	1095	
	59	25	27	13000	695	
総計		75	77	37900	2350	

第Ⅱ部　データベースマシン研究

1. 概　　要

1.1 背景と意義

1.1.1 データベース応用の拡大

データベースの応用は益々増加し，多くの分野に適用されており，データベースが使用されていない分野がないといっても過言でない程に広まっている。現在稼動中の大型計算機の84%でDBMSが導入されているといわれている。

データベースの規模も大きくなる傾向にあり，トランザクションの処理能力も，これに伴ってより高い性能のものが要求されつつある。

データベースの応用分野も，これまでのビジネス・ユース中心から，各種専門分野に広く使用されつつある。一例を挙げると，医用データベース，情報検索データベース，音声情報データベース，地理データベース，地質データベース，社会システム・データベース，法令データベース，統計データベース，画像データベース，CAD/CAMデータベース,等々がある。これらの分野では，処理の高速化，データベースの大容量化，処理の複雑化等が要求されている。

1.1.2 知識情報処理におけるデータベース

知識情報処理分野においても，データベース，およびその機能を拡大した知識ベースの導入は不可欠となろう。

専門家システムでは，現在のところ，データベースの規模は数10MBとされているが，将来は，10GB～1000GBのデータベースが必要になるとの予想がされている。

言語翻訳システムでは辞書データベースを必要とするが，現存のシステムでも数10MB～数GBのデータベースが必要である。

知識情報処理分野でのデータベースは，上述のように大量の知識を扱うだけでなく，これを用いて推論，演えきを行うため，検索が高速に処理されることが必要であり，時々刻々，新しい知識の追加や，知識の削除が行われるため，柔軟性や拡張性を備えることが必要とされる。

1.1.3 データベースマシン開発の必要性

大容量のデータベースを扱え，高性能な処理能力を持つデータベース・システムは，従来のように，汎用大型計算機上でDBMSを動かすことによっては実現できなくなると予想される。

このような背景より，ユーザの要求を満たす高性能なデータベース・システムを実現するための

専用プロセッサとして，データベース・マシンの開発が必要とされ，各種の実験機や一部の商用機が出現している。今後の市場性も極めて高いとされており，米国の調査会社の市場予測では，データベースマシンの売り上げは，1985年には10億ドル以上になるとされている。知識情報処理の応用が本格化するとすれば，さらに大きくなるものと考えられる。

このような時に，国家プロジェクトとして，関係データベースマシンの開発を行うことは，知識情報処理を支える知識ベースマシンを開発するための基礎を形成するのみにあらず，極めて市場性の高いデータベース・マシンの開発において，わが国が指導的立場をとるためにも，極めて有効であると考えられる。

1.2 FGCSプロジェクトにおける位置づけ

第5世代計算機が処理対象とする知識情報処理においては，種々の専門分野での高度で複雑な情報が知識という形で知識ベースに蓄積され，それに対して高度なアクセスが行われ，処理がなされることになる。

知識ベースシステムは，多数の複雑な構造をした知識を柔軟に蓄積することができ，推論や演えきを含む高度な検索を高速に処理する機能を持つことが要求される。このような知識ベースシステムの実現法には，未だ定かでない点も多いが，FGCS（the Fifth Generation Computer System）プロジェクトでは，知識ベースシステム機能を，推論機能とデータベース機能の2つに分け，これを推論マシンと関係データベースマシンの2つのサブシステムから構成されるシステムとして実現することを前期開発目標とし，中期，後期には，これら2つの機能を徐々に融合し，中期には，高並列度の知識ベースマシンと高並列度の推論マシンを，後期には知識情報処理システムを開発することを目指している。

関係データベースマシンの研究は，FGCSプロジェクトの知識ベースマシンの開発を支える基礎研究を行うと共に，中期に開発する知識ベースマシンのハードウェア実験を行うことのできる関係データベースマシンの実験機を開発することを目標とする。この実験機は，関係データベースマシンとしても充分な性能を持ち，前期に同じように開発される推論マシンと結合して，知識情報処理システムの研究開発が行えるものでもなければならない。さらに，汎用の関係データベースマシンとしても充分な性能を発揮することを目指す。

1.3 機能と性能に対する要求

1.3.1 現状のDBMSの問題点

日本におけるデータベースシステムの評価についてのアンケート調査では，4点満点で，総合評価が2.3～2.9であるのに対し，スループット／処理効率の評価は1.8～2.9と低く，処理速度の向上が望まれている。また，高水準言語，標準化，拡張性等の機能の拡充の他，オンライン・パーフォーマンスの向上を望む声が大きい。

知識情報処理への応用に関しては，拡張性や柔軟性のある蓄積構造が必要となり，高速な検索，推論，演えき，あいまいな処理等の高速かつ複雑な処理機能が必要である。これらを現状のデータベースシステムを用いて実現しようとすると，実現できなかったり，たとえできても，性能面から実用的でないものになる。

1.3.2 機能に対する要求

データベースシステムは以下のような機能を要求されるとされている。

A．ユーザ機能

- 検　索：selection, projection, join, etc.
- 更　新：addition, deletion, modification etc.
- 修　飾：position etc.

B．システム機能

- 制　御：priority, backup & recovery, security etc.
- データ手続：schema-subschema conversion, data compression, integrity etc.
- データベース・インタフェース：file linkage, data management, system interface etc.

C．アドミニストレーション機能

- 定　義
- 構　築
- 再構成

これらの中でも，高性能なデータベースシステムの実現と，知識ベースへの応用等には，オンライン・パーフォーマンスの向上が特に重要になると考えられ，これを支える機能としては，以下のようなものが重要となる。

- 検索と更新処理
- ロギングとシステム・リカバリ
- データ保護とファイル回復
- ファイル共用
- 端末管理
- 多重インクアイアリ処理

さらに，これらの機能が以下に示すように，Coddが関係データベースを提案する際に利点として指摘した基本的考え方にもとづいて実現されることが必要である。

- プログラムとデータの独立性
- 専門家から非専門家まで，共通して利用できる簡単なデータの見方の提供
- データベース応用プログラムを手続き的言語水準から集合を対象とする非手続き的言語水準へ上げる。
- 事実検索のような人工知能分野とファイル管理とを結合して将来の応用に備える。

1.3.3 性能に対する要求

データベースの応用の拡大，知識ベースへの応用に備えるためには

- 大量のデータの蓄積
- 高速の処理

の2点が必要となる。

知識ベースの将来予測によれば，数10～数100GB程度のデータベースの蓄積と検索能力が必要と考えられる。

高速処理に関しては，現状システムの数10倍～数1000倍の処理速度が必要と考えられている。

1.4 開発目標とマシンのイメージ

1.4.1 開 発 目 標

知識ベースマシン開発に先行して，高並列度のプロトタイプデータベースマシンの基礎研究を行うと共に，これらの開発に必要な基礎技術を構築することを目的として，関係データベースマシンの実験機を開発する。

A. プロトタイプデータベースマシンの基礎研究

3つの基本構想

- データストリーム処理
- データフロー制御
- オブジェクト指向

にもとづいた高並列度の関係データベースマシンを実現するためのアーキテクチャ，ハードウェア技術，ソフトウェア技術の基礎研究を行う。

セグメント化された関係データの集合演算を，このセグメントの転送と重畳して実行することをデータストリーム処理と呼んでいる。セグメントの管理は，データ記述子を用いたオブジェクト指向の考え方で行い，最下層メモリには，可動ヘッドディスク装置を用いる。ディスクの先行アクセス，バッファへのプリステージング，各種処理モジュール間のデータ転送は，セグメント化された関係データの集合を単位として行い，データフロー制御方式で制御する。

B. 関係データベースマシン実験機の開発

プロトタイプマシンの開発に必要な基礎技術の構築と，プロトタイプマシンの各種アーキテクチャのハードウェアシミュレーションを行うために，それ自体，関係データベースマシンとして充分な機能と性能を有し，ハードウェアシミュレータとしても充分に柔軟性と拡張性を持つ実験機を開発する。

処理速度は汎用大型機の数倍～十数倍程度を目標とし，数GB～十数GB 程度の容量を目標とする。

マシンを構成する機能モジュールの効率のよいアーキテクチャを明確にし，試作し，実験機に組み込むと共に，これを基に，さらに高性能な機能モジュールの開発，VLSI化を目指す。

1.4.2 マシンのイメージ

プロトタイプデータベースマシンの抽象アーキテクチャとして，現時点での委員会の考え方を図1.4.1に示す。全体は4つの基本構成要素

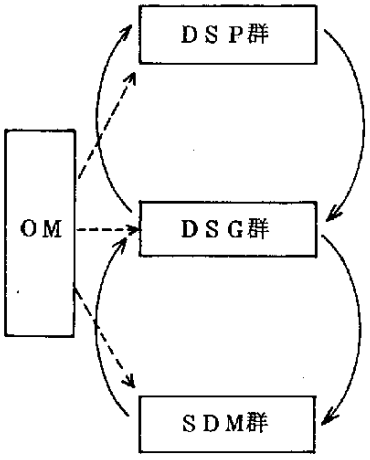
・ＤＳＰ：Date Stream Processor

・ＤＳＧ：Data Stream Generator

・ＳＤＭ：Secondary Data Manager

・ＯＭ　：Object Manager

からなる。



⟶　　データの流れ

⟶　　制御の流れ

ＯＭ　：Object Manager

ＤＳＰ　：Data Stream Processor

ＤＳＧ　：Data Stream Generator

ＳＤＭ　：Secondary Data Manager

図1.4.1　プロトタイプデータベースマシンの抽象アーキテクチャ

関係は，連想的なアクセスの可能な２次記憶システムのSDMに格納されており，処理に必要な部分だけがDSGにステージングされる。SDM群，DSG群，DSP群は，各々多数の処理モジュールからなる多重処理系である。DSGにステージングされたデータは，DSM群の多数のDSMに対して，関係データのデータストリームを生成し，各DSMは，その内の１つのデータストリームを処理し，再びDSG群の各DSGへと分散格納する。これらの操作を繰り返すことによって結果を得ることができる。

関係はセグメント化されて格納されており，処理の内容に応じて，SDM群からDSG群への転送の際に，DSP群による次の処理に適するように，関係の分割のし方が変更される。これは，SDM

群よりDSG群への転送の時，SDM群の各要素がDSG群の要素へデータを分配しながら転送することによって可能になる。これを動的クラスタリングと呼ぶ。

DSPにおける処理，および動的クラスタリングは，データの転送と重畳して実行されるデータストリーム処理方式を採用する。

SDMは可動ヘッドディスクと大容量バッファを持ち，データベース処理に必要なセグメントの先行アクセス，バッファへのプリ・ステージングが行われる。これらの処理と，SDM群からDSG群への転送，DSG群からDSP群を介して再びDSG群へ至る転送の制御は，データフロー制御方式で制御される。

処理に必要なセグメントは，OMの管理するDD/Dによって求められる。DD/Dの管理と処理は，オブジェクト指向アーキテクチャを持つOMによって管理処理される。

このように，プロトタイプデータベースマシンでは，

- データストリーム処理
- データフロー制御
- オブジェクト指向

の3つの考え方を基本にし，高並列度のマシンを目指す。

前期に開発する実験機では，以上の考えを基にし，他のアーキテクチャのハードウェア，シミュレーションの可能性も残し，図1.4.2のようなアーキテクチャを採用する。

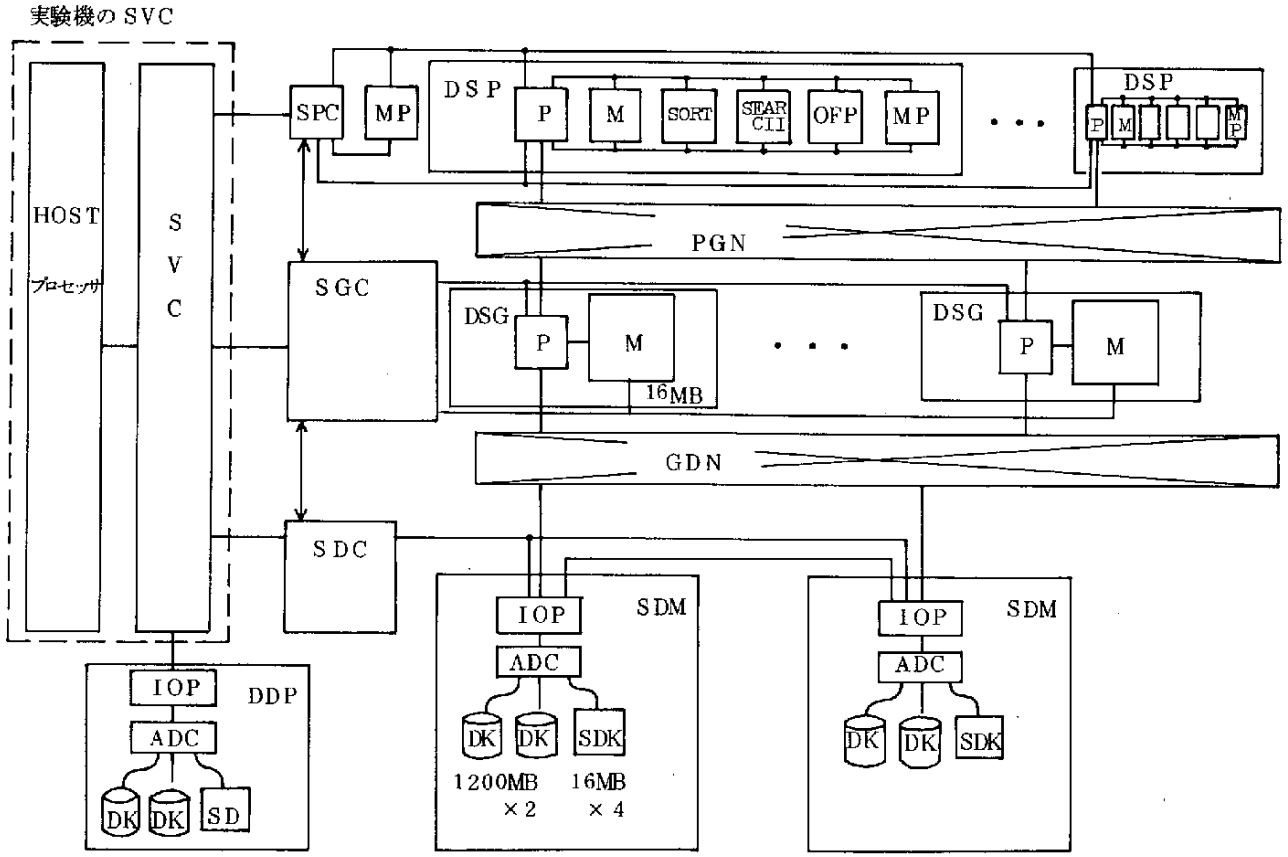

実験機の SVC
HOST
プロセッサ
S
V
C
SPC
MP
DSP
P
M
SORT
STAR CII
OFP
MP
DSP
PGN
SGC
DSG
P
M
16MB
DSG
P
M
GDN
SDC
IOP
ADC
DK
DK
SD
DDP
IOP
SDM
ADC
DK
DK
SDK
1200MB
×2
16MB
×4
IOP
SDM
ADC
DK
DK
SDK

図 1.4.2 （その 1 ）

SVC	:	Supervisor Controller	
SPC	:	Stream Processing Controller	制御部
SGC	:	〃 Generation Controller	
SDC	:	Secondary Data Controller	
DSP	:	Data Stream Processor	
P	:	Processor	
M	:	Memory	
SORT	:	SORT module	
SEARCH	:	SEARCH module	
MP	:	Math Processor	
DSG	:	Data Stream Generator	
SDM	:	Secondary Data Manager	
IOP	:	Input Output Processor	
ADC	:	Associative Disk Controller	
SDK	:	Semiconductor Disk	
DK	:	Disk	
PGN	:	Processor Generator Network	コミュニケーション・ネットワーク
GDN	:	Generator Disk Network	
DDP	:	Directory Dictionary Processor	

図1.4.2 (その2) 関係データベースマシン実験機のアーキテクチャ

2. 背景と意義

本章においては，データベースマシン開発プロジェクトを進める基礎となっている背景と，プロジェクトの意義について述べる。

このために，先ず，データベースの応用が益々拡大し，また，従来と異なった新しい応用分野として，第5世代プロジェクトが目標としている知識情報処理を対象とした応用が増大しつつあることを示す。

次に，現状でのデータベースシステムについての問題点を明確にするとともに，データベースシステムに要求される機能について言及する。そして，これらの機能を満たし，高速に処理することを可能とするデータベースマシンの必要性と需要予測について述べる。

さらに，将来の重要かつ新しい応用分野である知識情報処理で要求されるデータベースの特性について分析する。

最後に，以上の調査・分析結果にもとづいてデータベースマシン開発の社会的要請について述べる。

2.1 データベース応用の拡大

2.1.1 データベース応用の現状

データベースの応用はますます増加し，多くの分野に適用されており，データベースが使用されていない分野がないといっても過言でない程に広まっている。

ここでは，データベースシステム（DBMS）の利用状況について，国外（米国）と国内での調査結果を紹介する。

A．国外（米国）での状況

1980年3月 Computer Dicisions 誌が実施した調査結果[1]によると，計算機ユーザの54％が何らかの形でＤＢＭを使用していると報告されている。

表2.1.1は適用分野を業務別に見たものであり，一般管理，財務分野では，高い利用率を示している。

B．国内での状況

1980年3月 コンピュートピア誌の調査結果[2]によれば，日本においては，37.6％に近い導入率になっていることが示されている（図2.1.1）。表2.1.1は，業種別の導入状況について調査したものであり，殆んどの業種で適用されていることが示されている。

さらに，表2.1.2は，業務別の利用状況について詳細に調査したものであり，業種と同じように，すべての業務に適用されていることがわかる。

以上のデータは，主に，アンケート結果にもとづいて分析したものであり，全体的な傾向を知ることができる。

適用分野	割合（％）
一般管理	44.37
財務	43.73
製造	22.51
販売	12.86
購売	13.86
人事	23.47
研究・開発	14.15
その他	17.36

図2.1.1 考慮している適用分野

C．機種別の利用状況

次に，さらに詳細に利用状況をしらべるために，計算機の機種別での利用状況について示す。

図2.1.2は，使用計算機を大型機，中型機，小型機に分けて，1980年3月と1981年3月時点での各機種の利用状況を示したものである。

大型機分野では，１９８１年で８４％（１９８１年では７６％）になっており，この分野では飽和領域にきているといえる。

中型機分野では，２６％（同１５％）で，１９８０年度に比べて倍増しており，２～３年後には

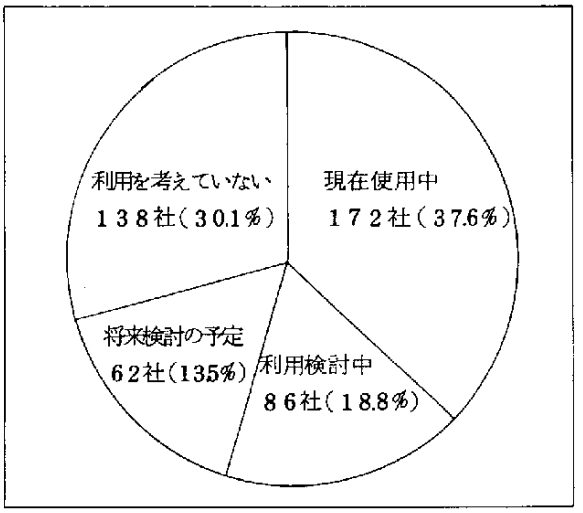


図 2.1.2　商用ＤＢＭＳの利用状況

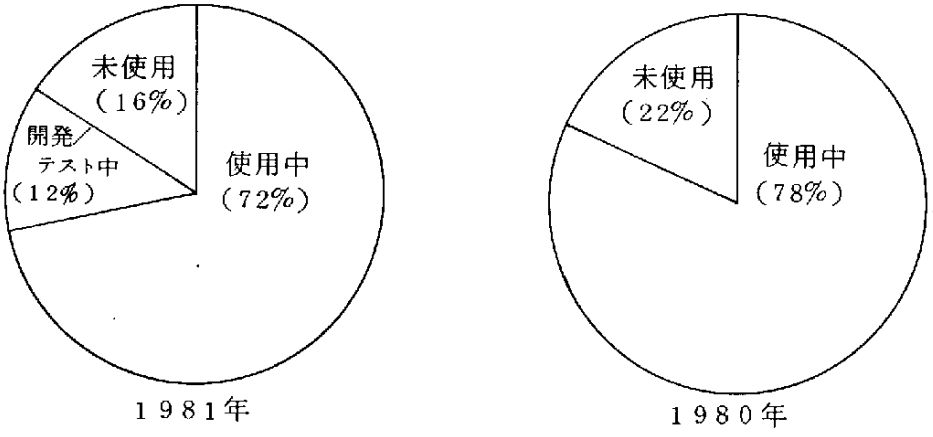
表 2.1.1　業種別商用ＤＢＭＳの利用状況

業種	アンケート回答企業数	DBMS利用企業数	利用率(%)
1 農林水産業・鉱業・建設業	36	12	(33.3)
2 食料品	21	4	(19.0)
3 繊維・パルプ・紙	36	6	(16.7)
4 化学	48	19	(39.6)
5 石油・石炭製品・ゴム	11	6	(34.3)
6 ガラス・セメント	10	1	(10.0)
7 鉄鋼	19	9	(47.4)
8 非鉄金属・金属製品	18	6	(33.3)
9 機械	48	20	(41.6)
10 電気機器	47	24	(51.1)
11 輸送機器	29	20	(69.0)
12 精密機器その他製造	25	10	(40.0)
13 商業	44	10	(22.7)
14 金融・損保・生保	32	17	(53.1)
15 不動産・運輸・倉庫	27	7	(25.9)
16 通信・電力・ガス・サービス	7	1	(14.2)
計	458	172	(37.6)

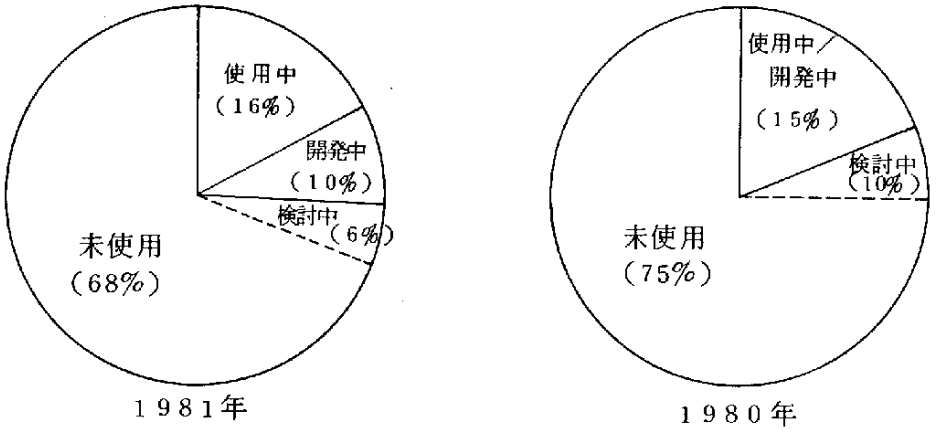
表 2.1.2　DBMS 利用企業の運用業務

（172社）

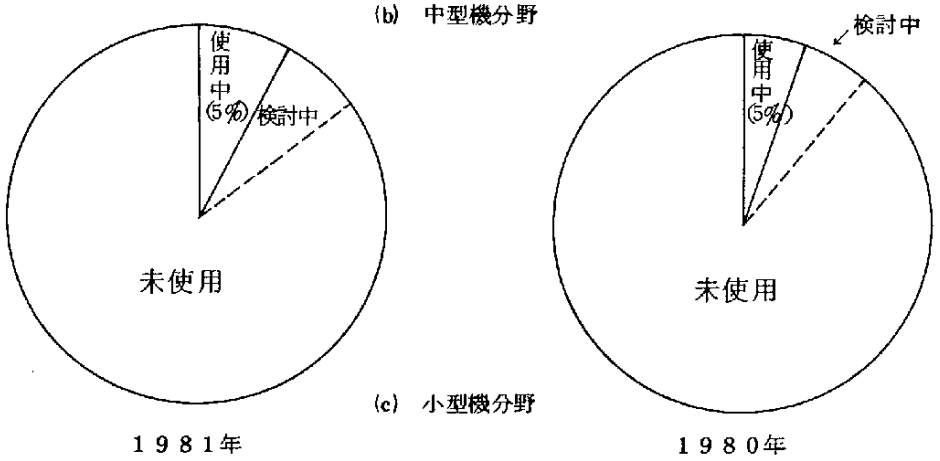
		1 人事管理	2 給与計算	3 財務会社	4 生産工程管理	5 在庫・資材・部品管理	6 原価管理	7 発注・仕入先管理	8 販売管理	9 予約管理	10 顧客・与信管理	11 積算・見積	12 輸送・運搬管理	13 データ・メッセージ交換	14 技術・情報管理	15 経営計画	16 諸学・授業計算
コンピュータ適用業務数Ⓐ		130	160	143	97	142	117	121	144	20	55	28	35	48	65	58	87
DBMS利用業務数Ⓑ		25	15	14	33	47	18	40	41	4	18	3	8	9	14	6	2
DBMS利用率ⒷⒶ(%)		19.2	9.4	9.8	34.0	33.1	15.4	33.1	28.5	20.0	32.7	10.7	22.9	18.8	21.5	10.3	2.3
業種別DBMS利用率（Ⓑ／Ⓐが48%以上の業務）	1.農林水産業・鉱業・建設業															○	
	2.食料品																
	3.繊維・パルプ・紙								○	○			○				
	4.化学									○							
	5.石油・石油製品ゴム					○	○										
	6.ガラス・セメント																
	7.鉄鋼				○						○			○			
	8.非鉄金属・金属製品	○			○						○						
	9.機械					○		○									
	10.電気機器					○		○							○		
	11.輸送機器												○				
	12.精密機器その他製造				○	○		○	○			○		○			
	13.商業	○						○	○		○						
	14.金融・損保・生保										○						
	15.不動産・運輸・倉庫			○				○	○		○						
	16.通信・電力・ガス・サービス																



(a) 大型機分野



(b) 中型機分野



(c) 小型機分野

図 2.1.2 ＤＢＭＳの導入状況

70 %近くになることが予想される。

小型機分野では，5%であるが，本分野での普及率は今後，著しく増加するものと予想される。

2.1.2 データベース規模

データベースシステムの開発において重要な要素の1つはどれだけのデータベース量のものをサポートできるようにするかである。従って，逆に，どれだけのデータベース量が要求されるかを認識することが必要になる。

すでに見たように，各種の応用分野においてデータベースシステムが利用されているが，そこで用いられているデータベースの規模がどの程度のものかは余り報告されていないのが現状と思われる。

図2.1.3は，国内において運用されている代表的データベースシステムについて，それを実行している計算機のマシン性能とデータベース規模との関係を示したものであり，CODASYL型などの従来より広く用いられているDBMSの場合である。

この図からみると，一般的傾向として，計算機の性能が倍になると，データベースは4～8倍増加することが要求されている。

図2.1.4は，INQ，RDB／V1，ADABAS，等の国内で稼動している関係データベースシステムについて，同様に示したものである。サンプル数が少いために，明日な傾向をつかむことは困難であるが，計算機の性能のアップに対して，データベース規模が増加する倍率はCODASYLなどよりも大きいと予想される。又，データベースの細かい特徴について，調査結果にもとづいて持性を述べるとすれば，次のように要約できる。

- リレーション数　　数個　～数十個
- リレーションサイズ　数十万件～数百万件
- タ　プ　ル　長　　数百B～数万B

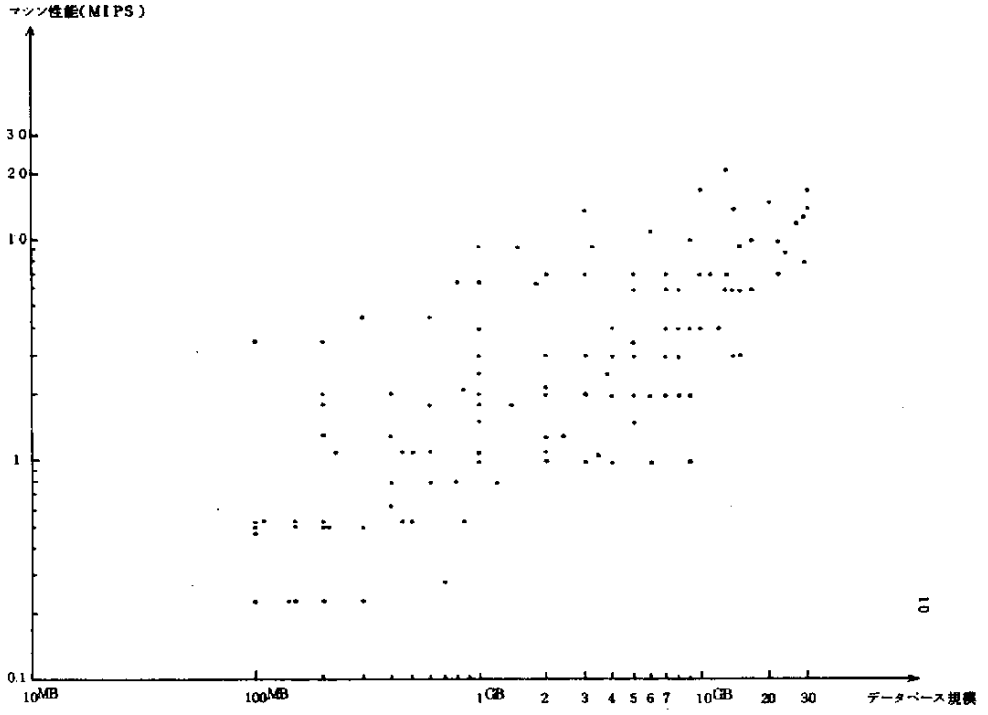


図 2.1.3　マシン性能とデータベース規模

2.1.3 処 理 速 度

応用分野の拡大，オンライン化などの利用形態の高度化，データベース量の増大，等により，データベース処理に対する高性能化が要求されている。また処理速度が向上することにより，さらに応用分野が拡大するものと予想されている。

処理速度はデータベース処理の内容に大きく依存し，それに対する要求について，必ずしも明確な定量的数値は示されていないようである。むしろ，処理速度は大きい程良いといった考え方が底流にあるといえる。

ここでは，処理能力（トランザクション数／秒）について論じられている例を示すことにより，概略の数値を得ることにする。

図 2.1.5 は，Britton Lee 社 Epstein が想定したもの[3]で，システム規模別に，データベース量と処理能力との関係を示したものである。

また，表 2.1.3 は，日電の関野が現状でのオンラインデータベースシステムにおけるトランザクション性能をデータベース量との関係を示したもの[4]である。

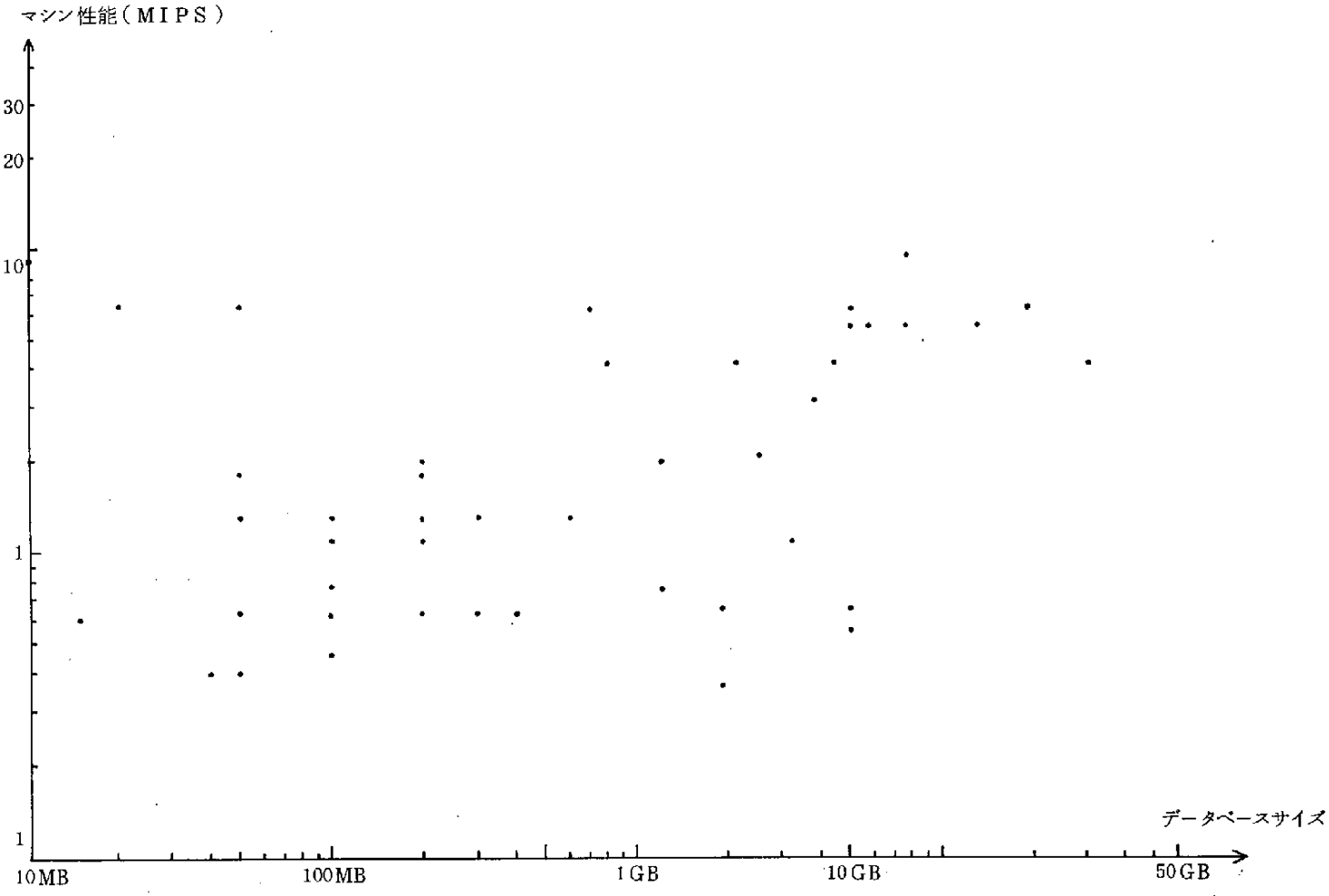


図 2.1.4　マシン性能とデータベース規模（関係データベースシステムの場合）

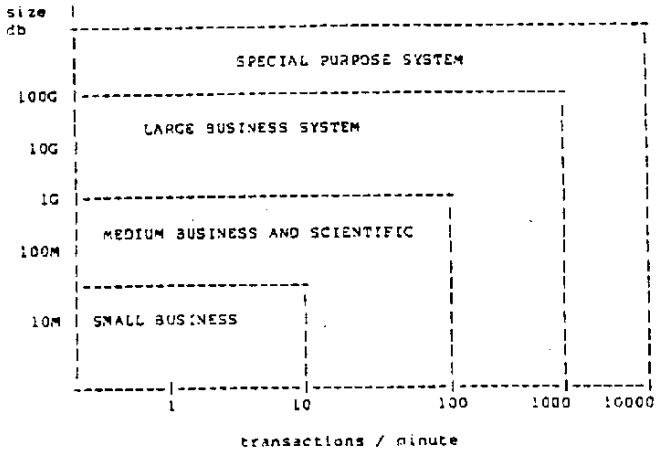


図 2.1.5　データベース量と処理速度との関係

表 2.1.3　データベース量と処理速度との関係

システム	マシン性能（MIPS）	トランザクション（件／sec）	DB 量（GB）
小規模システム	0.3－1	2－ 6	01－ 1
中規模システム	1 －3	6－ 20	1－ 10
大規模システム	3－10	20－ 60	10－100
超大規模 システム	10 以上	60－200	100－ 数百

ただし，1件のトランザクション当たりの所要命令数をすべて，150 kステップとする。

2.1.4　データベース応用分野の拡大

これまでは主として，ビジネスユースを中心にしたデータベースシステムについて述べてきた。しかし，データベースの利用はビジネスユース以外に，各種の専門分野に広く使用されつつあり，これらのあるものは，実験レベルから実用レベルになりつつある。

そして，これらの専門分野を対象としたデータベースの応用は多岐に亘っており，また，処理の複雑化，等が要求されている。

以下では，本分野の代表的なものについてリストアップし，これらの詳細については，2.3で述べることにする。

- 医用データベース
- 情報検索データベース
- 音声情報データベース
- 地理データベース
- 社会システムデータベース
- 法令データベース
- 統計データベース
- 画像データベース
- CAD/CAMデータベース
- 知識データベース
-
-
-

これらの応用は以上の他にさらに数多く挙げられ，第5世代プロジェクトが対処しようとしている知識情報処理分野の応用の1つと考えられる。

2.2 データベースシステムの現状と将来

2.2.1 問題点

すでに各種のデータベースシステムが開発され，使用されているが，必ずしも充分な評価は行われていない。また，たとえ評価されていたとしても，評価結果について明らかにされていないのが現状といえる。

ここでは，現状でのデータベースシステムの問題点が将来の要求事項を明らかにすることを目的に，一般分野と知識情報処理分野からみた現状のデータベースシステムに対する評価，問題点について述べる。

A. 一般分野からみた評価・問題点

表2.2.1は日本におけるデータベースシステムの評価についてのアンケート結果を示す。全体的には，4点満点で2.3－2.9であり，スループット／処理効率の面では1.8－2.9であり，処理速度の向上が要求されていると考えられる。

また，表2.2.2は米国における評価についてのアンケート結果を示しており，全体的には，2.8－3.4である。

さらに，図2.2.1は，使用者のアンケートによるDBMSに対する意見，不満をまとめたものであり，機能面と性能面からの問題点が指摘されているといえる。

B. 知識情報処理分野からみた問題点

知識情報処理分野からみた現状のデータベースシステムに対する問題点を厳密に論じるためには，先ず，知識情報処理の各応用分野での要求事項を明確にすることが必要になるが，それらは2.3において詳細に述べるとして，ここでは，知識情報処理の全般からみて本質となる点に焦点をしぼり言及することとする。

知識情報処理においては，各専門分野での高度で複雑な情報が知識という形でデータベースに蓄積され，それに対して高度なアクセスが行われて処理されることになる。

従って，知識情報処理で用いられるデータベースに対する要求として，多数の複雑な知識を効率良く蓄積する機構と，その中から要求を満たすものを高速に検索する機構とが最も重要である。

知識を表現し，蓄積する方式として，各種考えられているが，現状でのデータベースシステムを用いると手続き形式で実現することになり，拡張性，柔軟性の面から問題であると考えられる。このために，非手続き的な形式が有効であるとされており，セマンティックネットや述語形式，等が望まれている。

一方，知識を貯えているデータベースに対する処理として，高速な検索，推論，演えき，あいま

いな処理，等が必要であるとされており，これらが効率よく実現できる構造を備えていることが要求されている。これらを，現状のデータベースシステムを用いて構成しようとすると，実現できなかったり，または．たとえできても性能面から実用的でないものになると想像される。

表2.2.1　商用ＤＢＭの日本における評価

	回答企業数	評価項目							
		全体的な満足度	スループット／処理効率	導入のしやすさ	使いやすさ	ドキュメンテーション	技術サポート	トレーニングサービス	提供業者の信頼度と安定性
ＩＭＳ（ＩＢＭ）	27	2.6	1.9	2.5	2.5	2.7	2.7	2.7	3.2
ＰＤＭ（日立）	21	2.4	2.3	2.6	2.3	2.2	2.2	2.2	3.0
ＤＬ／１（ＩＢＭ）	18	2.5	1.8	2.3	2.5	2.4	2.3	2.1	3.3
ＡＩＭ（富士通）	17	2.3	2.2	2.2	2.2	2.1	2.2	1.9	2.4
ＩＤＳ（日電・東芝）	12	2.9	2.8	2.8	2.8	2.5	2.4	2.3	2.8
ＩＮＩＳ（富士通）	8	2.5	2.1	2.5	2.5	2.1	2.3	2.4	2.9
ＴＯＴＡＬ（シンコムシステムズジャパン）	8	2.8	2.9	3.0	3.1	2.1	2.1	2.3	2.1
ＡＤＭ（日立）	6	2.3	2.0	2.6	2.2	2.5	2.5	2.2	3.0
ＤＭＳ－１１００（ユニバック）	5	2.8	2.8	3.0	3.0	2.4	2.8	2.2	3.4
ＡＤＢＳ（日電・東芝）	5	2.8	2.2	3.0	2.8	2.4	2.6	2.4	2.8
ＤＭＳ－Ⅱ（バロース）	5	2.4	2.4	2.6	2.7	2.4	1.8	1.8	2.6
全体平均	163	2.5	2.2	2.6	2.5	2.3	2.4	2.3	2.9

表2.2.　米国における評価（ＤＡＴＡＰＲＯ社）

ＩＭＳ（ＩＢＭ）	34	2.9	2.4	2.2	2.5	2.8	2.8	2.6	—
ＤＬ／１（ＩＢＭ）	36	2.8	2.5	2.5	2.5	2.5	2.7	2.8	—
ＴＯＴＡＬ（シンコムシステム）	108	3.2	3.1	3.2	3.2	2.7	2.7	2.7	—
ＤＭＳ－Ⅱ（バロース）	30	3.4	3.4	3.4	3.4	2.5	2.6	2.5	—

(注) 評価方法としては，各カテゴリーについてその優劣を，①非常に優れている（４点），②優れている（３点），③まあまあである（２点），④劣っている（１点），の４段階で採点してもらい，その結果を各商用ＤＢＭＳごとに集計し，平均したものを，その評価値とした。すなわち特定の商用ＤＢＭＳについて，各カテゴリー別の得点の和を求め，それを回答企業数で割ったものを，その商用ＤＢＭＳのカテゴリー別の評価値としている。

なお，商用ＤＢＭＳの評価にあたっては，回答企業が５社以上のものを対象とした。

（コンピュートピア　１９８０.３）

- わかりやすさ，使いやすさ，柔軟性を備えたもの，特にユーザー言語に工夫がなされたものが欲しい。
- どのような適用業務に威力を発揮するか，プレゼンテーションを明確にすべき。
- 異なる機種やＯＳ間で互換がとれるよう，標準化を進めて欲しい。
- 小規模なマシンでも運用可能な，手軽で安価なシステムの供給が待たれる。
- 適用業務の拡大に応じ，小規模からスタートし大規模なコンフィギュレーションへと段階的に成長できるような拡張性を備えるものが望ましい。
- インストレーションや，再構成が簡便に済み，メンテナンスが容易であること。
- オンライン向けにＤＢ／ＤＣを一体的に備えていること。
- オンライン・パフォーマンスが専用システムに比べ，あまり劣ることのないような配慮，たとえば一部のモジュールのファームウエア化などが望まれる。
- メンテナンス保証水準の向上，マニュアルの改善など，ベンダー・サービスの強化など。

（コンピュートピア　１９８０.３）

図 2.2.1　　ユーザの意見・不満

2.2.2　データベースシステムの機能

2.2.1 において，ユーザからみたデータベース管理システムに対する意見，問題点が指摘されたが，本節においては，これを技術面から具体化することを目的に，データベース管理システムの機能について分析する。

現状でのデータベース管理システムが備えている機能は各種の分類がなされているが，ここでは，O.H.Bray & H.A.Freeman (Sperry Univac)の分類を紹介する[5]。

A. ユーザ機能

ユーザが直接要求する機能であり，これには，主なものとして，以下のものがあげられる。

- インクアイアリ

関係データベースシステムでは，Selection，Projection，Join，等の機能である。

- 更新

これには，Addition，Deletion，Modification，Replacement，等が該当する。

・修飾

これには，Position,等の処理があげられる。

B. システム機能

ユーザには直接見えないが，データベースシステムを運用して行くためにシステムとして備えることが必要な機能である。

・制御

Priority，Backup & Recovery，Security，等の制御がこれに当たる。

・データ手続

Schema - Subschema Conversion，Data Compression，Integrity，等がこれに属する。

・データベースインタフェース

これには，File Linkage，Data Management System Interface，等があげられる。

C. アドミニストレーション機能

データベースを運用，管理して行くために必要な機能で，例えば，データアドミニストレーション（Definition，Generation，Reorganization，等）がこれに属する。

2.2.1 で述べたユーザの要求を満し，高性能なデータベース管理システムを実現するためには，前記3つの機能を高性能に実現することが必要になるが，知識情報処理，等にみられるようなオンライン・データベースシステムの場合には，その中でも，特に次の機能が重要と考えられる。

・インクアイアと更新処理

・ロギングとシステムリカバリ

・データ保護とファイル回復

・ファイル共用

・端末管理

・多重インクアイアリ処理

このような各機能に対しては，現状でのデータベース管理システムでは充分な機能，性能が達成されていないと考えられる故，これらに対する考慮が必要と思われる。

2.2.3 データベースマシンの必要性と市場予測

前節までで，ユーザの要望，意見にもとづいたデータベースシステムに対する要求機能，性能について明らかにした。

このような背景より，ユーザの要求を満たす高性能なデータベースシステムを実現するための専

用プロセッサとして，データベースマシンの開発が必要とされ，各種の実験機や一部の商用機が出現している。

現在研究開発されているデータベースマシンの大部分は関係データベースマシンであることから，本節では，関係データベースマシンの必要性と市場予測について述べることにする。

A. 関係データベース管理システムと関係データベースマシン

関係データベースシステムは，他の方式に比べて各種の利点を備えているために，多くの計算機で実現され，応用分野も広まりつつあり，重要視されている。

関係データベースシステムの特徴，利点は提案者Codd により，以下のように指摘されている。

- プログラムとデータの独立性
- 専門家から非専門家まで，共通して利用できる簡単なデータの見方（データモデル）の提供
- データ管理者の負担の軽減
- データベース応用プログラムを手続き的言語水準から集合を対象とする非手続き的言語水準へ上げる。
- 事実検索のような人工知能分野とファイル管理とを結合して将来の応用に備える。

以上の特徴から明らかなように，関係データベースシステムは一般的な応用分野は勿論のこと，将来重要と考えられている知識情報処理分野においても非常に有効であると思われる。

しかしながら，関係データベースシステムは上で述べた利点を備えている一方で，従来の方式にはない多くの新しい機能を実現することが必要である。従って，このような機能を持つ関係データベースシステムを従来型のフォンノイマン型の汎用計算機上に実現することは性能をはじめとして多くの困難があり，また，巨大なデータベースに対する要求と広範囲の応用分野に対しても適応できるシステムの要請，等の理由から，従来より高性能な関係データベースマシンの出現が期待されていると考えられる。

B. 市場予測

以上の要請を受けて，各種のデータベースマシンが開発されつつあるが，現状において大部分は実験機であり，商用機はごく一部に限られている。そして，現時点でのデータベースマシンに対するマーケットは始まった段階といえるが，米国の調査会社※の市場予測をみると，将来大きなマーケットとなると予想されている。

図2.2.2は上記予測結果にもとづいて，今後5年間の出荷台数と金額を示したもので，IBM社が参画した場合としなかった場合に分けて示している。

※ Creative Strategies International

従って，１９８５年には＄１　billion 以上になると予想されるが，知識情報処理の応用が本格化するとすれば，さらに大きくなるものと考えられる。

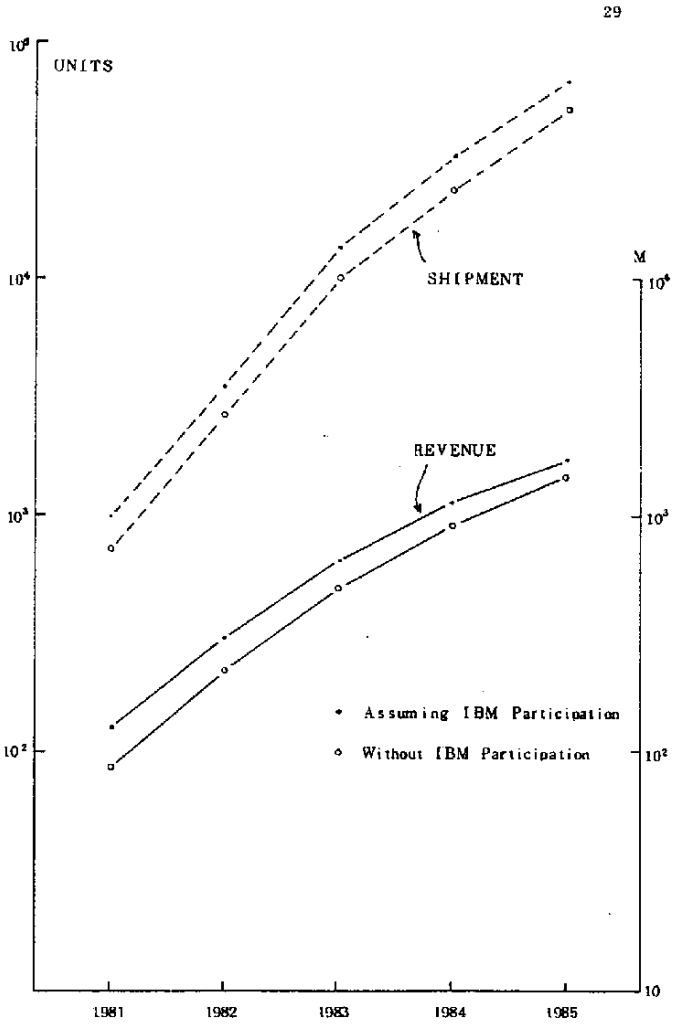


図 2.2.2　データベースマシンの市場予測

2.3 知識情報処理におけるデータベース

これまで，一般分野を対象とするデータベースシステムに対する機能，要求を中心に述べてきたが，本節では，知識情報処理分野において要求される処理概要，特徴を明らかにするとともに，そこで必要とされるデータベースについて述べる。

すでに，2.1.4において，知識情報処理に関する応用分野について列挙したが，ここでは先ず，専門家システムと言語翻訳システムについて述べ，続いて，他の分野を含めた応用として，知識工学全般について触れる。

2.3.1 専門家システムとデータベース

A. 専門家システムの分類

現在，各種の専門家システム（Expert system）が開発され，その中には，実験的に使用されているものもある。専門家システムについては，すでに詳細に報告されているので，ここでは要約について述べることにする。

専門家システムは広い分野において実現されているが，表2.3.1に代表的システムの具体例を示す。[6)]

B. 処理の概要

専門家システムで行われる処理は，一般的にとらえると，ルールベースを基にした推論処理であるといえる。そして，ある推論を行うために，backward 制御方式がとられている。

図2.3.1は一例を示しており，7個のルールから成り，各アルファベットは事実（facts）を示している。

この例において，Xが成り立つかを見つける場合には，右辺にXが現われるものを捜し，それが導出されるか否かをしらべる。従って，この例では，先ず，Rule 5 と Rule 7 のいずれかが満されるかをしらべる。このために，先ず，Rule 5についてみると，事実FとHが共に真実であるかをしらべる。この時，Fが成り立つにはRule 1が，Hが成り立つにはRule 3 が成り立つかをしらべる。このように，順にさか上がること（Recursion と呼ばれる）により事実が成立するかをしらべて行く。図2.3.2にその過程を示す。

C. ルール表現の例

図2.3.1ではルールを一般形で表現したが，具体的イメージをつかむために，代表的な専門家システムであるＭＹＣＩＮの例を図2.3.3に示す。

D. 専門家システムの特徴

専門家システムにおいて行われる処理の特徴は以下のように要約できる。

- 知識ベースと呼ばれるルールから成る巨大データベースを備える。
- データベースはパターンマッチング処理により参照される
- ルールから成るデータベースは修正，削除，追加，累積が行われる
- 実時間処理を原則とする
- 専門家を対象とするために，高度な知識表現が必要になる
- 学習機能を備えることが望まれる

E. データベースの規模

専門家システムにおいて要求されるデータベースの規模を明らかにするために，実在システムの例と，推定によるもの，について紹介する。

a. 実在システム

表2.3.2は代表的な実在のシステムのルールの量を示している。これらを構成するデータベースはルールとファクトが混在した形として実現されており，全体を総称してルールと呼んでいる。従って，ここでのデータベースとしては，全体を含めたルールにより構成されるものとして考察する。

１つのルールが５００～１Ｋバイトで表現できると考えると，表2.3.2のGrand master Chess の場合には，３０K～５０Kルールであることから，３０－５０Mバイト必要であると想定される。しかし，今後，本格的なシステムやルールが拡大することを予想すると，さらにルールの必要量が増加すると考えられ，Michieは10^{12}ビット必要であると予測している[6]。

b. 推定によるデータベース量

内科学に関する専門家システムを実現しようとした場合に必要とするファクト（facts)について，Pauker は次のように推定している。

内科学の教科書	2,０３５頁
１頁当たりのfacts	１００個（仮定）
従って，全体で	２００K facts
学際的事項を入れるために２倍する	４００K facts
保険，経済的知識の導入	５００K facts
安全係数として２倍	1,０００K facts
内科領域の専門知識を入れるため２倍する	2,０００K facts

1つの fact が500－1Kバイトで　　1G～2Gバイト

表現されるとする

表2.3.1　専門家システムの具体例

Medicine	• MYCIN for identification of bacteria in blood and urine samples, and prescription of antibiotic regime	Shortliffe, at Stanford Medical School, USA
	• INTERNIST for diagnosis in internal medicine	Myers and Pople at Pittsburgh University, USA
	• Intensive care ('iron-lung')	VM (Fagan and others)
	• Interpretation of lung tests	PUFF (Kunz)
Chemistry	• DENDRAL for identification of organic compounds	Feigenbaum, Lederberg, Djerassi, Buchanan, Carhart and others. Stanford University, USA
	• SECS system for designing organic syntheses	Wipke, University of California at Santa Cruz, USA
	• Molecular genetics	MOLGEN (Lederberg, Martin, Friedland, King, Stefik)
Other	• Consultancy for structural engineers	SACON (Bennett)
	• Consultancy for mineral prospecting	PROSPECTOR (Hart, Duda, Einaudi)

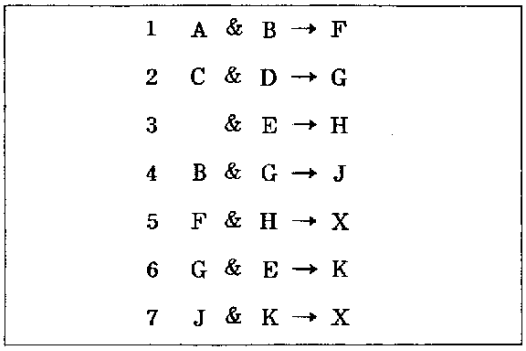


図 2.3.1　ルール表現の具体例

```
   X may be deduced from Rule 5 or Rule 7
1  Starting with Rule 5, are F & H true?
2     F can be shown to be true if A & B are both true (Rule 1)
3        A is not known to be true, so this attempt fails
4  Continuing with Rule 7, are J & K true?
5     J can be shown to be true if B & G are both true (Rule 4)
6        B is known to be true a priori
7        G can be shown to be true if C & D are both true (Rule 2)
8           C is known to be true a priori
9           D is known to be true a priori
10       therefore G is true
11    therefore J is true
12    K can be shown to be true if G & E are both true (Rule 6)
13       G is already known to be true (step 10)
14       E is known to be true a priori
15    therefore X is true
16 therefore X is true.
```

図 2.3.2　Recursion による推論過程

ULE 58

If : 1) The site of the culture is one of : those sites that are normally nonsterile, and

2) This organism and at least one of the likely pathogens associated with the of the culture agree with respect to the following properties : grammorph air

Then : There is strongly suggestive evidence(.9) that each of these pathogens is the identity of the organism.

図2.3.3　ルールの表現例

表2.3.2　専門家システムのルールの量

Skill	Nature of implementation	No of pattern-based rules in implemented system
Seeing a scene	Incremental catalogue of visual patterns. Simple scenes of shadowed polyhedra. Waltz, early 1970s (023)	10
Balancing a pole	Incremental catalogue of pattern-based rules. Michie and Chambers, mid-1960s	225
Identifying organic compounds from mass spectra	Incremental catalogue of pattern-based rules. 'DENDRAL' program of Lederberg, Feigenbaum and Buchanan	c 400
Identifying bacteria from lab tests on blood and urine	Incremental catalogue of pattern-based rules. MYCIN program of Buchanan and Shortliffe	c 400 ～900
Calculating-prodigy arithmetic	Alexander Aitken, studied by Hunter, 1962 (024), used pattern-based rules	?
Grandmaster chess	Chess-masters, studied by Binet (025), de Groot (026), Chase and Simon (027), Nievergelt (028), use pattern-based rules	30 000-50 000

2.3.2　言語翻訳システム

A. 処理の概要

言語翻訳システムの処理の流れの概要を図2.3.4に示す。入力テキストは解析フェーズを経て中間言語に変換され，中間言語レベルで入力言語と目的言語の変換（トランスファ）が行われ，

これをもとに合成を行った後，目的テキストが得られる。

解析フェーズでは，語い，構文，意味の解析が行われるが，この時，各種の解析辞書を用いて処理される。トランスファフェーズではソースと目的言語間の変換が行われるもので，この時，対訳用のトランスファ辞書が用いられる。

合成フェーズでは，文脈，意味，構文，語いの合成が行われるが，この時には，合成用辞書が用いられる。

B. データベースの規模

処理概要において述べたように，言語翻訳システムにおいて，各種の辞書を貯えるために巨大なデータベースが必要であることを示した。以下では，具体的システムについてのデータベース規模を紹介する。

a. ＳＵＳＹシステム

多言語間翻訳を目的とするザールブルッケン大学のＳＵＳＹ（ザールブルッケン自動翻訳システム）では，次に示すような辞書を備えている。

- シンタックス解析辞書

ドイツ語	135,000 エントリ
ロシア語	15,000 エントリ
英　語	5,200 エントリ

- セマンテイックス解析辞書

ドイツ語	3,500 エントリ
ロシア語	1,000 エントリ
英　語	2,700 エントリ

- トランスファ辞書

ロシア語―ドイツ語	9,000 エントリ
英　語―ドイツ語	4,100 エントリ

- シンタックス合成辞書

ドイツ語	9,000 エントリ
英　語	1,600 エントリ
フランス語	1,000 エントリ

従って，１エントリに例えば２５６バイト必要であるとすれば，約５０Ｍバイト程度要求されることになる。

b. 専門用語辞書

言語翻訳の分野においては，専門分野ごとに，各種の専門用語辞書が開発され，使用されている。

ここでは，代表的な専門用語集の辞書の大きさを参考までに示しておく。

表2.3.3は辞書の項目数を示しており，１項目が５００～１０００バイトで構成されるとすれば，ＬＥＸＩＳの場合で，1,５００～3,０００Ｍバイト必要であると考えられる。

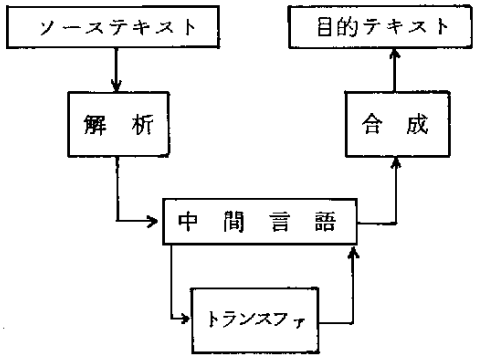


図2.3.4　言語翻訳システムの処理過程

表2.3.3　専門用語辞書の項目数

システム名	項目数	備　　考
TERMIUM	２０万	カナダ政府，モントリオール大学
LEXIS	３００万	西ドイツ連邦政府翻訳局
TEAM	７０万	シーメンス
EURODICAUTOM	２０万	ECルクセンブルグ
EWF	３９万	ドレスデン技術大学

2.3.3　知識工学が必要とするデータベース

これまで，知識工学の応用分野の中から，専門家システムと言語翻訳システムについて述べてきたが，本節においては，これらを含め，知識工学全般に対するデータベースについて考察する。

A. 知識工学の応用分野

特定分野の専門家の知識と判断機構をシステムに組み込むことによって，その専門家と同様な判断能力を有する知的システムを実現するアプローチとしてとらえられる知識工学は，医学，工

表2.3.4　知識工学の応用例

コンサルテーション・システム	有機化合物同定	DENDRAL (Stanford 大)	化合物構造の推論システム
		META-DENDRAL (ditto)	質量分析ルールの推論システム
		SU/P (CRYSALIS) (ditto)	蛋白質のX線結晶データ解析システム
		MOLGEN (ditto)	分子遺伝学用アドバイス・システム
	技術用アドバイス・システム	SACON (Stanford 大)	構造体解析システム
		SU/X (ditto)	信号解析・理解システム
		KNOBS (Mitre 社)	航空機識別シミュレータ
	診断・治療システム	MYCIN (Stanford 大)	血液伝染病・脳膜炎の診断・治療システム
		TEIRESIAS (ditto)	大規模知識ベースの構成・保守・利用のためのメタレベル知識獲用システム
		PUFF (ditto)	肺機能診断システム
		MECS-AI (東大病院)	心不全診断・治療システム
		CASNET (Rutger 大)	緑内障の診断・治療システム
		EXPERT (Rutger 大)	甲状腺疾患・リウマチ等の診断・治療システム
		PIP (MIT)	腎疾患の診断システム
言語情報処理システム	質問応答システム	SHRDLU (MIT), BUILD (MIT)	積木の世界におけるロボットの対話
		ELIZA (ditto)	精神分析用 QA システム
		GUS (Xerox 社)	旅行計画用 QA システム
		SCHOLAR (MIT)	地理学習用 CAI
		STUDENT (Bobrow, 当時 MIT)	算数の問題解決システム
		Newton (MIT)	物理の問題解決システム
		Isaac (Texas 大)	物理の問題解決システム
		京大 (長尾, 辻井)	化学の問題解決システム
		ARIMAS (阪大)	算数, 幾何の問題解決システム
	文章解析理解システム	SAM (Yale 大), TOPLE (MIT)	談話理解システム
		MARGIE (Stanford 大)	英文解析システム
		LINGOL (MIT)	英文解析システム
		PLATON (京大)	日本文解析システム
		EXPLUS (ETL)	日本文解析システム
		MILISYJ (ditto)	日本語理解システム
		VISUALISER (ditto)	同上, 但グラフィックディスプレイ応答
		MSSS-78 (通研)	日本語理解システム
	音声理解システム	HEARSAY-II (CMU), LUNAR (Harvard 大)	
	情報検索システム	PLANES (Illinois 大), REQUEST (IBM) RADDER (SRI)	関係データモデルを用いた (形式化された) データに対する QA システム
		やちまた (日本 IBM), RENDEZVOUS (IBM)	
	翻訳システム	METEO (Montreal 大)	天気予報 (英語→仏語)
		XONICS MT System (Xonics 社)	露語, チェコ語, セルビア語→英語

〔情報処理 Vol. 21. №12 (1980)〕

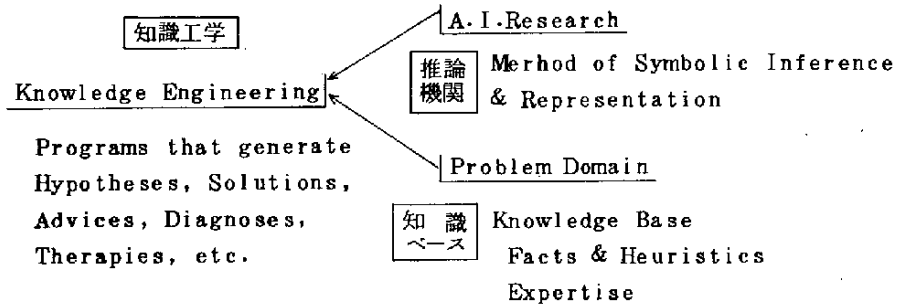


図2.3.5　知識工学の基本構成
(阪大におけるFeigenbaum, E.A. 教授の講演－1979 年10月25日－より)
〔情報処理, Vol21. №12 (1980)〕

学，科学，等の分野で適用され，数多くの実際に役立つシステムが実現されつつある。

表2.3.4は知識工学の応用例をまとめたもの[7]で，実験的なものから，一部実用されるものまで多岐に亘っている。

B．知識工学と推論マシン，関係データベースマシン

知識工学は，図2.3.5に示されるように，知識ベースと推論機関から構成されるといわれる。そして，知識工学において重要な基本メカニズムは，知識の表現，推論，獲得であると考えられている。

推論機関は，推論マシン等で実現され，探査（Traversing），作用（Application），呼び出し（Call）などの技術を用いて知識の推論を行う。

一方，知識ベースは，事実（Facts）と専門知識（Expertise）から構成され，木（Tree），網（Net），プロダクションルール，論理式（Form），関数（Function），手段（Procedure），等の形で蓄積される。そして，知識ベースは関係データベースマシン，等の知識ベースマシンによって，高性能に実現されることが期待されている。

C．データベースと関係データベースマシンに対して要求される機能

知識工学でのデータベースは知識ベースと考えられ，事実（Facts）と専門知識（ExpertiseまたはRule）とから構成されると見做される。一方，関係データベースマシンはそれらを蓄積すると共に，要求に合った情報を高速に取り出す，等の機構が組み込まれたエンジンと考えられる。

本節では，知識工学で必要とされるデータベース，関係データベースマシンの要求事項について列挙する。

a．多量の知識の蓄積

実現されるシステムの知能の高さは，その中に蓄積されている知識の量に依存するといわれる。従って，多量の知識を蓄積できるという要請は，本質的なものであり，第一に実現されなければならない。知識を貯えるデータベースの概略の規模として，数十～数百Gバイト程度が要求されると考えられる。

b．高速検索

知識工学において要求される処理は，推論,演えき，等であり，このためには，多数の知識の中から，特定の意味のものを検索し，速やかに取り出すことが基本メカニズムといえる。実際の応用プログラムにおいては，これらの基本メカニズムが複数個組み合わされて複雑になったものと想定され，それを高速に処理することが要求される。

c．拡張性／柔軟性

知識工学においては，時々刻々，新しい知識の追加や知識の削除が必要となり，これに迅速に対処できる構造が要求される。このために，手続き形式による表現よりも，セマンテイックネットや述語形式，等の非手続き形式の表現をとることにより，柔軟性，拡張性を備えることが必要である。

d. マルチユーザ支援

知識工学の応用においては，１つのシステムを多数のユーザが利用する形式が一般と考えられるため，マルチユーザ支援機能を備えることが望まれる。これには，複数ユーザからの同時検索と，同時更新処理を満足することが必要である。

e. ＲＡＳＩＳ機能の拡充

マルチユーザ環境を高度にサポートするデータベースシステムにおいてはＲＡＳＩＳ（Reliability Availability Serviceability Integrity and Security）機能が充実されていることが非常に重要になる。

この中でも，知識工学の応用分野においては，特に，データベースのintegrity，recovery 管理，security 管理が重要になるので，これらに対する拡充が望まれる。

f. 易用性の向上

システムの良否は，機能，性能とともに，使用，運用が容易かどうかに依存する。

データベースシステムでの易用性を向上させるポイントとして，ビュー，複数データモデルのサポート，等が重要であり，これらは，関係データベースマシン上のソフトウエアにより解決されることが要請される。

g. 高水準言語インタフェース

システムの利用を容易にするため備えなければならない機能として，データベースシステムにアクセスするための高水準言語インタフェースを設定することが必要である。

これまでに，高水準言語として，ＱＵＥＬ，ＳＱＬ，等が開発されているが，知識情報処理に合った新しいものを開発することが望まれる。

D. 知識工学のためのコンピュータシステム

すでに述べたごとく，知識工学を実現するためのコンピュータシステムとして，高速の推論マシンと，大容量，高性能の関係データベースマシンが要求されている。

知識工学に要求されるコンピュータシステムのイメージを明らかにするために，エジンバラ大学のMichie のシステムを例としてあげておく。[6)]

図2.3.6は１９８０年代の知識工学のためのコンピュータシステムのシナリオを提示したもので，そこでは，知識ベースとして，10^{12} ビットのものが必要であることを示している。

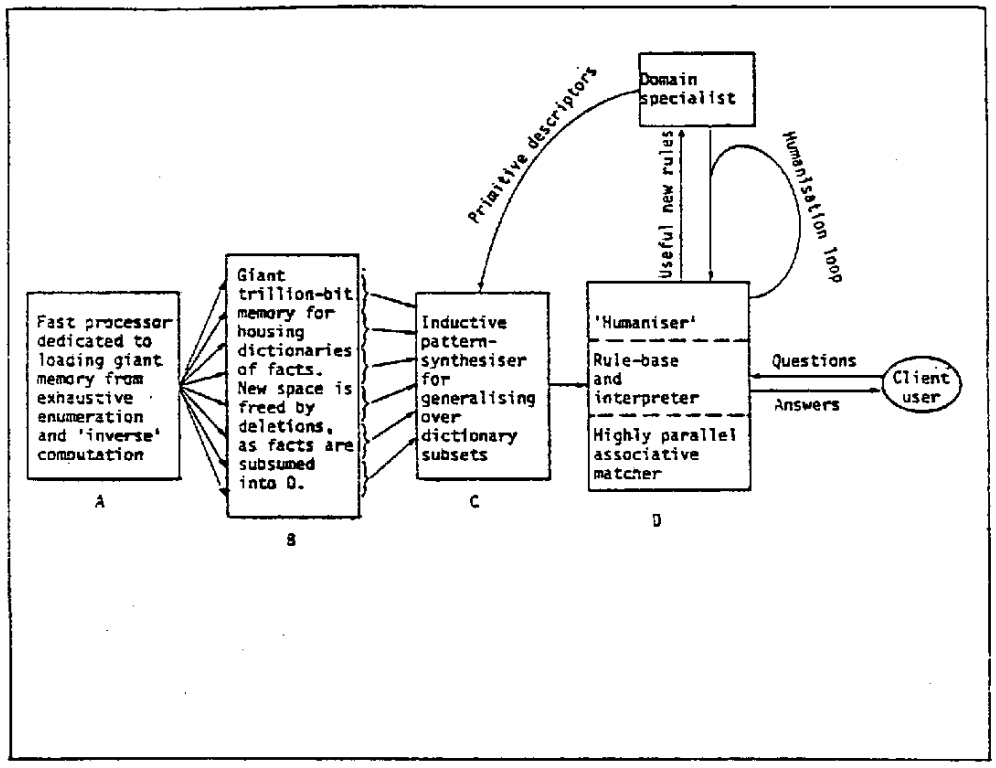


図2.3.6　1980s scenario for knowledge-based computing

2.4 データベースマシン開発の社会的要請

コンピュータシステムの高度化，多様化への要請は，ソフトウエア問題の顕在化，半導体集積回路技術の飛躍的進歩および応用分野の拡大，等から生じていると考えられる。

コンピュータシステムの応用は，従来の分野での拡大とともに，オフィス，ホーム，等の新しい分野にも浸透しつつあり，さらに，2.1および2.3で述べたような知識情報処理という高度で且つ新しい分野にも広がりつつある。

多くの応用分野の中でも知識情報処理は，１９８０年代後半に本格的に実用化されるものの１つと考えられており，人間の知的作業を支援する高度なシステムとしてとらえられている。

知識情報処理システムは，すでに述べたように，従来のシステムと異なり，演えきや推論，等の高度な機能を備えており，これが実現されると，広い分野に亘って，非常に大きなインパクトを及ぼすことが予想される。

このような背景下において，第５世代プロジェクトは，知識情報処理を実現するために必要な基礎技術を確立することを目的に，関連する巾広い分野の技術を開発することを考慮して進められるものと考えられる。

知識情報処理システムは，すでに述べたごとく，推論マシンと知識ベースマシンを核にして構成される。そして，その中で，知識ベースマシンは，多量の知識を貯えるもので，人間の脳に対応し，知識情報処理システムを実現する時には最も重要と考えられる。

知識ベースマシンを実現する方式として各種あげられるが，2.3，2.4で述べたごとく，要求される機能が効率良く実現されると考えられる関係データベースマシンが最も適当であると思われる。

以上，知識情報処理からみた関係データベースマシンの必要性について述べた。

一方，データベースの利用は，これについてもすでに述べたように，データ処理一般分野において益々拡大し，より高性能なデータベースシステムの出現が求められている。このような高性能なデータベースシステムを実現するためにも，本格的なデータベースマシンの開発が要請されていると考えられる。

以上，知識情報処理，一般の情報処理の観点から，データベースマシンの開発が要請されている背景について述べた。

—参考文献—

1) Judd Q. Bartling : Strong trend to DBMS Confirmed, Computer Decisions, July, 1980

2) 山田, 石塚：日本の商用ＤＢＭＳ利用形態とユーザ評価, コンピュートピア, 1980, 3

3) R.Epstein, et.al.: Design decisions for the intelligent database machine, ＡＦＩＰＳ ＮＣＣ 1980,

4) 関野陽 ：データベースマシンの研究開発と実用化, 電子通信学会技術研究報告 ＡＬ80-54, 1980,12

5) O.H.Bray, et. al. : Data Base Computers, Lexington Books

6) Infotech Limited : The Fifth Generation, ＩＮＦＯＴＥＣＨ ＳＴＡＴＥ ＯＦ ＴＨＥ ＡＲＴ ＲＥＰＯＲＴ, Series 9, Number 1, 1981

7) 田中幸吉：知識ベースとその応用, 情報処理, Vol. 21, No 12, 1980

3. データベースマシン研究開発の現状

3.1 データベースマシンの分類

データベースマシンは1970年代初頭からその研究が始まり，現在までに数多くのマシンアーキテクチャが提案されるに至っている。これらは，データ操作機能の一部のハードウェア化をねらったもの，新しいデバイス技術の応用に中心を置いたものから，データベース諸機能の多くを統合化し実現しようとするものまでそのレベルにも大きな差がある。ここでは，データベースマシンとしてのアーキテクチャについて分類することとし，各種処理技法については第4章で検討する。

O. H. Bray らはデータベースマシンを2つの観点から分類している。[1] 即ち，データベースの検索，更新を2次記憶媒体上で直接行うか否か，複数台のプロセッサを用いて並列処理をするか否かという2点から，以下の5つのクラスに分けている。

SPIS …… Single Processor Indirect Search

SPDS …… Single Processor Direct Search

MPDS …… Multiple Processor Direct Search

MPIS …… Multiple Processor Indirect Search

MPCS …… Multiple Processor Combined Search

データベースの検索処理等では，十分高い並列度が存しており，一台のプロセッサを用いるか（SP），あるいは多くのプロセッサにより並列処理するか（MP）は，そのパフォーマンスに大きな影響を及ぼす。近年，ハードウエアコストの低下はめざましく，データベースにおける単純な処理は比較的簡単な構成のプロセッサで十分であり，並列化は現実的である。実際，これまでの多くのマシン RAP,[4)~10)] CASSM,[11)~13)] DIRECT[14)~16)] etc）はMP型の構成を採り，並列処理による性能の向上を図っている。

CPUと主記憶の間のフォンノイマンボトルネックは主記憶，二次記憶間にも存在し，従って，この間の無駄なデータ転送を出来る限り省き，データはそれが格納されている場所に近い所で処理することが望ましいと考えられる。DS型のマシンではディスク上で直接データ操作を行いこの問題の解決を図っている。しかし，このためにはディスクからのデータストリームに完全に追従した処理がなされなくてはならないため，比較的簡単な検索しか実現できず，複雑な条件に対しては何回転も必要となってしまう。一方，IS型のマシンでは，一端データをバッファに取り込み，バッファ上で処理を施す。バッファの記憶媒体は，RAMや固体ディスクが用いられ，より柔軟なアクセス機構を提供している為，処理が効率的に行える。ISとDSは併用することも可能であり，C

SではDSでデータをある程度絞り込んでバッファに転送し，その後バッファ上で残った処理を行う。

3.1.1 SPIS型マシン

汎用プロセッサをホストコンピュータのバックエンドとし，データベース処理だけを専門に行うようにしたデータベースマシンである。プロセッサは一台のためSP型，2次記憶上のデータベースはインデックス等を用いて一端主記憶に移した後処理を施す為IS型と見做せる。このタイプは，いわゆるソフトウェアバックエンドマシンと呼ばれるものであり，ベル研のXDMS[17]に始まり，CCA社のData Computer[18]，Software AG社のADABASマシン，Britton Lee社のIDM500[19)~20)]，日本電気のGDS[21)~23)]などが載げられる。ソフトウェア後置により，バックエンド側のソフトウェアは汎用化された機能が不要となるため簡素化出来，またホストの負荷を大きく削減することが出来ると考えられ，コストパフォーマンスの良いシステムが期待できる。しかし，システムを2つに分離するため，ホストとバックエンドマシンとの通信が必要となり，この間のインタフェースのレベル，データ転送レートが問題となる。レコード単位の処理による低次インタフェースを設定するとそのオーバヘッドは無視できず非効率的であるが，より高位レベルのインタフェースを設けることによってパフォーマンスの向上を図ることが出来る。また，ホストとバックエンドの分離により，データ共用が可能となり，セキュリティの維持が容易になるなど機能的な面でもメリットが多い。

アーキテクチャ的には従来の単一プロセッサ上でのデータベースシステムが，バックエンドマシンに移されただけであり，本質的な改善がなされているわけではなく，それ程大きな性能の向上がもたらされるものではない。従って，バックエンドマシン上ではファームウエアによる機能の強化を図る場合が多い。

3.1.2 SPDS型マシン

2次記憶側でのフォンノイマンボトルネックを解消すべく，この型のマシンでは専用化されたハードウエアにより2次記憶上のデータに対して直接サーチが行われ，条件を満足する必要なレコードの必要なフィールドだけがホストに送られる。このデータ量の削減によりホストマシンでの単純な繰り返し処理をなくすことができる。一般にディスク上のデータをホストマシンの主記憶に取り込むソフトウエアオーバヘッドは条件処理のそれに比べるときわめて大きく，この回数の減少はパフォーマンスの向上に大いに有効である。SPDS型のアプローチでは図3.1.1に示される如く，専用化ハードウエアがホストI/Oチャネルの中間に位置する。この手法では，2次記憶上で直接（Directに）データ操作を行っているわけではないが，中間的な記憶媒体を用いておらず，デー

タストリームに追従した処理がなされるため，ＤＳ型と考えられる。バッファを持っていないため，限られた機能しか実現することは出来ず，完全なデータベース機能をサポートするものではない。これらはホストで行う必要がある。このタイプのマシンには，ＩＣＬ社のＣＡＦＳ[24)25)] Braunschweig大学のサーチプロセッサが属する[26)]。ＣＡＦＳでは１６個のキー比較レジスタを有し，数トラックを同時にサーチ可能である。一般に必要とするレコードは全体の１％以下であることが多く，このようなフィルター処理により大きな性能の向上が期待される。

3.1.3 ＭＰＤＳ型マシン

Logic per track 方式のマシンがこのタイプに相当する。即ち，ディスクのトラック対応にプロセッサを設け，トラック内のデータを直接サーチする方式である（図3.1.2）。記憶媒体が回転するにつれてデータストリームがプロセッサ内に取り込まれ，一定時間後，トラックに再び書き戻される。当該データがプロセッサ内に滞留している間に種々の処理を施されることになる。このような 'on the fly' の処理方式では，一回の回転内に処理出来る量は限られており，もし一回転で処理出来ない場合には印をつけておいて次の回転で残された処理を行う。多数のプロセッサを用いて並列に処理するため（ＭＰ），サーチ時間はシステムに入り切るデータベースに対してはその大きさに依存せず大変高速になるが，コストは容量と共に線形に増加するため，大容量データベースに対して適用しようとする場合には問題がある。しかし，将来，磁気バブルの如き不揮発性の固体ディスクに対してロジックを付加することが容易になり，大容量化が可能になれば，この方式も現実性を帯びてくるものと考えられる。

このタイプのマシンにはフロリダ大学のＣＡＳＳＭ，トロント大学ＲＡＰのディスク版ＲＡＰ1，電総研のＥＤＣ[27)]などが属する。ＳＰＤＳ型のマシンと同様，２次記憶からホストへのデータ転送を大きく減少することが出来，さらに多数のプロセッサを用いて並列サーチを行うことにより高速性が実現される。

3.1.4 ＭＰＩＳ型マシン

ＭＰＩＳ型のマシンは先のＭＰＤＳ型のマシンと同様，多数のプロセッサにより並列にサーチを行うが，サーチは２次記憶に対して直接行われるのではなく，一旦データベースの必要な部分を中間記憶媒体にステージングし，その上でなされる。図3.1.3にその構成を示す。ＭＰＤＳではデータベース容量に制限があり，大容量化を試みると，まずＲＡＭや固体ディスクに転送してから処理を施すことが必要になり，ＭＰＩＳ型のマシンに帰着される。この方式では，２次記憶媒体から中間記憶への高速ステージング機構が重要な役割を果す。すなわち，中間記憶上では多くのプロセッ

サにより並列処理が行われるため，２次記憶から中間記憶へのデータ転送レートは十分高くなくてはならない。このためにはトラック並列読み出しのディスク等を使う必要がある。また，中間記憶へのデータ転送中にフィルタリングされることはないので，インデックス等の高次のデータ量低減化技法も必要となる。この型のマシンでは，操作対象データが中間記憶に入り切る場合には高速に処理されるが，そうでない場合にはディスクとのデータの入換が多くなり，性能の低下を起こす場合がある。トロント大学のＲＡＰ２，ＲＡＰ３，シラキュス大学のＳＴＡＲＡＮを中心としたＲＥＬＡＣＳ，[28)29)]ウイスコンシン大学のＤＩＲＥＣＴ，カーネギーメロン大学のシストリックアレイを用いたマシン[30)]などがこのタイプに属する。中間記憶媒体としてはバブル，ＣＣＤから大容量ＲＡＭへと変わりつつある。性能は，ＳＰＩＳに比べるとプロセッサ台数を増したことによってかなり高速化されるが，単なるサーチ，即ちセレクションが中心である場合にはＭＰＤＳで十分であり，ＭＰＩＳはジョインやプロジェクションなどより複雑な演算を実現する場合に，高い性能を示すと考えられる。

3.1.5　MPCS型マシン

ＭＰＣＳ型のマシンはダイレクトサーチとインダイレクトサーチの両手法を融合し，より高いパフォーマンスを目指したものである。即ち，２次記憶から中間記憶へのデータ転送に際してある程度のサーチロジックによりフィルタリングを行い転送量を減少させ，より複雑な処理は中間記憶の上で行われる。中間記憶は複数バンクから構成され，ＭＰＩＳ同様複数台のプロセッサにより並列に処理を行う。これまでの多くのＭＰＩＳ型のマシンでは２次記憶インタフェースに関して十分な考慮が払われていなかった場合が多く，中間記憶を単なる作業記憶とみなす場合にはこのタイプになると考えられる。即ち，記憶階層間にデータストリームに追従したサーチ機構を設けるダイレクトサーチ自体はデータベースマシンにおける必須技術と見做せ，サーチの複雑度により機能分割したＭＰＣＳ型のアプローチは自然な考え方といえる。

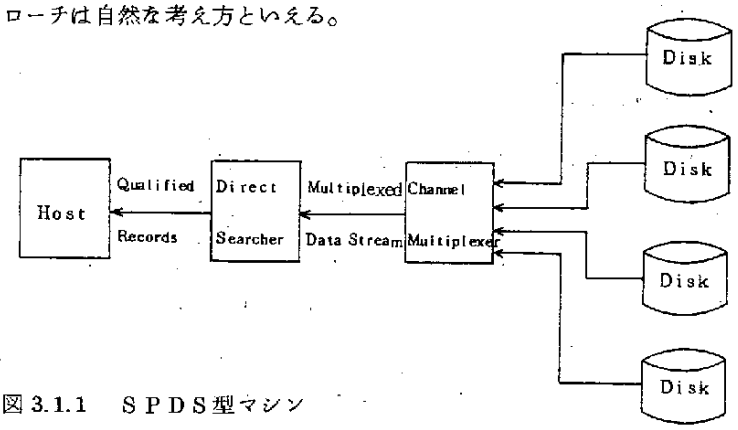
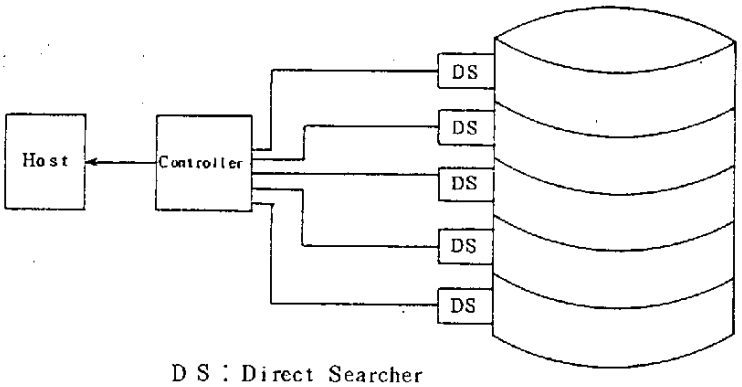
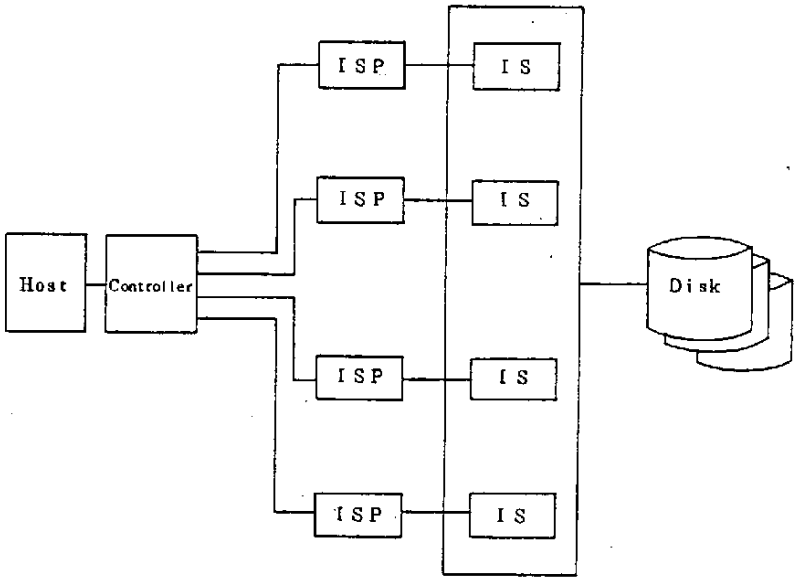


図 3.1.1　ＳＰＤＳ型マシン



D S : Direct Searcher

図 3.1.2　MPDS型マシン



I S P : Indirect Search Processor
I S　: Intermediate Storage

図 3.1.3　MPIS型マシン

3.2 データベースマシンのアプリケーション依存性

現在までに数多くのデータベースマシンが開発されてきた。これらはそれぞれ特長を有しており，限られたアプリケーションに対しては優れた性能を示すと考えられる。７０年代の乱立期を経て，８０年代に入ると，データベースマシンの研究はその反省期を迎えた感があり，これまでの種々のマシンをふり返ってみて「はたして汎用データベースマシンは存在するのか」という問題が認識されるに至っている。すなわち，一口にデータベース処理といっても，データベースは，その規模，複雑さ，問い合わせの性質などにおいて各種様々なものが存在し，これら全てに対して理想的なパフォーマンスを与えるマシンが存在するのかという疑問である。すでに，５つのタイプのデータベースマシンをアーキテクチャ上から分類したが，これらの内あるタイプが他のタイプに対して常に優れているというわけではなく，最適なアーキテクチャはそのアプリケーションの性質に依存すると考えられる。例えば，ＭＰＣＳ型のマシン上で単純な定型業務だけからなるデータベース処理を行ったとしても，コスト／パフォーマンスが良いとは考えられない。データベースマシンを評価する場合には，まずそのマシンが使われる環境を明確にする必要がある。この問題に対して，Ｐ．Hawthornは以下の３つの代表的アプリケーションを設定し，各タイプのデータベースマシンの評価を試みている。[2) 3)]

(1) Bibliographic Search

(2) Business Application

(3) Statistical Analysis Application

3.2.1 Bibliographic Search

このタイプのアプリケーションではデータのサーチに多大な時間が要され，その結果出力されるデータは比較的少ない点が特長として載げられる。データは問い合わせに対して構造化されておらず，アクセスパスも存在しない場合が多い。リレーショナルシステムでは，リストリクションとプロジェクション（重複除去は行わない）が主体となる分野といえる。このようなサーチ中心の応用では，問い合わせ変換等のシステムオーバヘッドの割合は少なく，９０％以上の時間がデータサーチに費されると考えられる。従って，一台のプロセッサよりも多数のプロセッサを用いて並列に処理を行うことにより大きなパフォーマンスの向上が期待できる。またこの種のサーチはディスクからのデータストリームに対して直接行うことが可能である。中間記憶に移した後，サーチすることも出来るが，バッファの管理等余分な仕事が増加するため，ＭＰＤＳ型のマシンが最も適していると考えられる。また，規模が小さい時にはＳＰＤＳ型のマシンで十分であろう。ＭＰＣＳ型のマシ

ンを用いたとしても活性化されるのはディスク側のプロセッサが中心で，中間記憶側のプロセッサはアイドルとなることが多くこの分野に適しているとはいえない。

3.2.2 Business Application

このタイプのアプリケーションではデータが構造化され，アクセスパスが完備されている点に特長がある。このような環境では実際にデータを操作する時間の占める割合は少なくオーバヘッドは少なくともデータ操作と同じ程度に存すると考えられる。従って，単なるプロセッサの多重化はそれ程効果をもたらさない。アクセスパスが存在するため，アクセスする領域が絞られ，多数のプロセッサを用いても無意味な場合が多い。このようなアプリケーションではＳＰＩＳ型マシンで十分ということになる。ＭＰＩＳやＭＰＣＳ型のマシンでは余分なオーバヘッドが生ずるとともにリソースの使用効率が低く，コスト的に問題がある。

3.2.3 Statistical Analysis Application

このタイプのアプリケーションでは多くのリレーションにまたがる処理が主体となる点に特長がある。例えば，計量経済学データベース，医療データベース，公害データベースなどがこのタイプに属する。これらのデータベースを用いて種々の解析を行おうとする場合には，当初データを作成する時点では関連がないと思われていたものに対しても種々の処理が施されることになり，多くのジョインオペレーションが必要となる。このようなアプリケーションではデータの処理負荷はきわめて高く，ＩＮＧＲＥＳで実現した場合には全体の９９％にも達するとされている。これはジョイン操作の負荷が重いことに依っている。従って，並列度を十分に抽出出来る環境であり，複数台のプロセッサを用いることが望ましい。データの処理負荷が重いという点はBibliographic Search と同じ環境ではあるが，ジョインが多いという点が特長といえる。そして，ジョイン，特にイクスプリシットジョインに対してはＭＰＤＳ型のマシンでは高い性能が得られないことが知られている。ジョイン処理のアルゴリズムにも依存するが，単純な処理方式ではくり返しタプルを参照することが必要となり，ＭＰＩＳ型のマシンが適していると考えられる。ハッシュビットアレイを用いるＤＳ型のマシンも存するが，これはインプリシットジョインにしか適用出来ず，一般的な解決策とはいえない。ＭＰＣＳではディスク側で一端，リストリクション，プロジェクションを施しデータ量を少なくした後，中間記憶上でジョインを並列処理することが出来，複雑な処理要求を高速に実現することが可能となる。ＭＰＣＳ型のマシンは多くの構成要素からなるため，それだけシステムオーバヘッドが増加するが，そもそもデータ操作負荷自体が重いため，これは打ち消されると考えられる。以上の如く，このタイプの応用にはＭＰＣＳ型のマシンが最も適していると考えら

れる。

3.2.4 結　論

以上の如く Bibliographic Search にはMPDS／SPDS型のマシン，Business Application にはSPIS型のマシン，Statistical Analysis Application にはMPCS型のマシンがそれぞれ適しているということがわかった。アプリケーションの分類は完全なものではなく，むしろ代表的な特性を列挙しただけにすぎず，実際には中間的なアプリケーションが中心となり必ずしも上の結論を直接適用できるとは考えられない。しかし，汎用計算機に対して多くのマシンが並存するように，データベースマシンにも，アプリケーションの種類，規模に応じて幾種類かのマシンが存在すると考える方が自然であろう。全てのアプリケーションに対して最適な1つのデータベースマシンを指向するのではなく，必要に応じて適切な技術を取り込み統合化することによって，そのアプリケーションに最も適したマシンを構成出来ることが望ましいと考えられる。

3.3 代表的なマシンの定量的評価

数多くのデータベースマシンが現在までに提案されてきているが，定量的な評価を行っているマシンはほとんど存在しない。また種々の問い合わせがどのようにマシン上で処理されていくかという点に関しても，詳細な点になると明らかでない場合が多い。これは，主に多くのマシンがページマシンであること，および，そのマシンにとって処理し易い問い合わせについては考察されてはいるものの，そうでない場合については未検討であることによる。また，マシンのパラメタ設定も前節で述べた如く対象とするデータベースの性質に依存するため，各マシンに対して平等な環境を生成し，統一的にデータベースマシンを評価することはきわめて困難である。しかしながら，いくつかの標準的な問い合わせに対する各マシンの推定される性能を比較することにより，大まかな傾向をつかむことは出来ると考えられる。以下の評価は P. Hawthorn の検討にもとづいている。[2),3)]

評価対象のデータベースマシンとしては，以下の6つを取り上げることとする。

- 連想ディスク
- RAP
- CASSM
- DBC
- DIRECT
- CAFS

これらのマシンを含め各種マシンの概容については次節で述べることとし，ここでは性能評価にとって重要な特長に関して簡単にまとめておくことにする。

3.3.1 各マシンの特長

連想ディスク……DS型のマシンであり，ここでのプロセッサは単純な構成のサーチ機構から成るものと仮定する。ホストには，条件を満足するレコードの必要なアトリビュートのみが送られる。このマシンのコストは以下のより複雑なマシンに比べて最も安価になると考えられる。

CASSM[11)~13)]……データエンコード手法を適用している数少ないマシンである。キャラクタストリング値はテーブルに貯えられ，レコードにはポインタのみが格納される。アーキテクチャ的にはMPDS型のマシンに属し，並列にサーチが行われる。各プロセッサは簡素なものを用いており，1回の回転では1つのアトリビュートに関する検索，或は，出力しか行えないが，更新や追加，ガーベジコレクション，さらに算術演算機能などが備えられている。ジョイン操作はCAFSと同様の手法により実現され，Explicit Join の場合にはホストでアトリビュートの結合処理を

行わねばならない。データには各々マークビットが付加されており，これにより複雑な処理を回転をくり返しながら行う。さらにリレーショナルモデルに対する評価には無関係ではあるが，ＣＡＳＳＭではポインタをたどる機能が付いており，階層モデルのサポートを可能にしている。

ＲＡＰ……リレーショナルモデルのサポートのみを指向したＲＡＰは，アーキテクチャ的にはＣＡＳＳＭと似たシステムであるが，ＣＡＳＳＭがＭＰＤＳ型マシンであるのに対し，ＲＡＰ，より正確には ＲＡＰ２，３では，記憶媒体をＣＣＤやＲＡＭとしており，ＭＰＩＳ型マシンと見なすことができる。ＲＡＰはＣＡＳＳＭに比べると，複数アトリビュートの比較が同時に出来る点でより機能強化されている。またタプル対応に付加されたマークビットを用い，ＣＲＯＳＳ　ＭＡＲＫオペレーションによりセミジョインを実現している。

ＤＢＣ……検索アトリビュートがクラスタリングキーワードである場合，ＫＸＵ，ＳＭ，ＳＭＩＰ，ＩＸＵなどから成るストラクチャループにより，サーチ空間を大幅に減少させることが出来，他のマシンにない大きな特長を有している。サーチ自体はＴＩＰによりＭＰＤＳ型の処理が施され，他のマシンと同様の能力をもつ。

ＤＩＲＥＣＴ……他のマシンと異なり，中間記憶の各ブロックとプロセッサがクロスバによって結合されており，任意の結合関係を生成できる点に，セルラーロジックにはない特長が存在する。これにより，問い合わせ内の並列処理，問い合わせ間の並列処理が効果的に実現される。またＣＡＳＳＭやＲＡＰがマークビットによって中間結果を表現していたのに対し，ＤＩＲＥＣＴでは中間リレーションを直接生成する方式をとっている。更にプロセッサにはＬＳＩ／１１ を用いており，処理はデータストリームに追従した形でなされるのではなく，一ページ分のデータのロードが完了した後，処理を始めるという処理形態となっている。アーチテクチャ的にはＲＡＰ同様ＭＰＩＳ型マシンに属する。

ＣＡＦＳ……機能は連想ディスクとほとんど変わらないが，ジョイン演算に関してHashed Bit Array を用いた効率の良い処理方式を採用している点に特長がある。

3.3.2　各種パラメータの設定

各マシンを評価するに当たり，その比較のため，ＩＮＧＲＥＳシステムでの実行時間も加えることにする。評価は３つの代表的な問い合わせ (1)1 リレーションに対する単純な条件検索 (2)2 リレーションにまたがるジョインを含む問い合わせ (3)Aggregate Operation を含む問い合わせに対して行われた。以下，各マシンに対する諸パラメタの設定に関して簡単にまとめておく。

A. 記憶媒体

ＤＩＲＥＣＴとＲＡＰを除く他のシステムでは，媒体として可動ヘッドディスクを採用する事

とし，ＲＡＰとＤＩＲＥＣＴに関してはその中間記憶をＣＣＤで構成するものとする，ディスクに関するパラメータを以下に示す。

DISK PARAMETERS AND VALUES

Meaning	Value
block size	512 bytes
disk rotation time	.0167 seconds
average access time	.0300 seconds
data rate to host	.0012 / block
cell read time	.0008 / block
# 512-byte blocks/cylinder	418 blocks

ＲＡＰとＤＩＲＥＣＴではページサイズが異なるが，ここでは評価を統一的に行うため，ＤＩＲＥＣＴの１６Ｋbyte を採用する。

CCD PARAMETERS

Meaning	Value
size of CCD page	16K bytes
page scan time	.012 seconds

B. プロセッサの構成

プロセッサは，ディスクの回転に追従出来る処理速度を有するものとする。先のデータ転送レートでは全てのアトリビュートに関し全タプルを処理すると仮定すると，単純な計算によりディスクの場合には２ＭＩＰＳ，ＣＣＤの場合には３ＭＩＰＳ程度のプロセッサが必要になる。しかし，実際には一タプルの一部しか処理をしない場合が多く，また条件が満たされる割合も少ないと考えられるため，これより低い処理能力で十分と考えられ，限られた機能から成る比較的簡単な構成のプロセッサを用いることで，データストリームにそった処理は実現可能であると考えられる。

C. 記憶セル数

ディスクは１シリンダ当たり１９トラックとする。従って，ＣＡＳＳＭ，ＣＡＦＳ，ＤＢＣ，連想ディスクでの並列処理数は１９となる。またＤＩＲＥＣＴは８台のプロセッサ，１６台のメモリから，ＲＡＰは１６台のプロセッサメモリペアから成るものとする。

CELL PARAMETERS AND VALUES

Parameter	Meaning	Value
NDCELL	Number of cells for CASSM, CAFS, DBC	19
NDRP	Number of cell processors in DIRECT	8
NDPAG	Number of data cells in DIRECT	16
NRCELL	Number of cells in RAP	16

D. ホストにおけるオーバヘッド

ホストマシンにはＰＤＰ11／70を仮定し，問い合わせ変換および正当性チェックなどのオーバヘッドおよびデータベースマシンとの通信時間に関しては詳細は略すが，INGRES システムでの値を採用することとする。これらのオーバヘッドはデータベースマシンを用いても必要な時間である。

HOST PARAMETERS AND VALUES

Meaning	Value best case -- worst case
host overhead I/O time	.0300 seconds
host overhead CPU time	(.006--.1600) sec.
host CPU time to communicate with backend	(.006--.0300) sec.
host time to format results for printing, perform math functions, etc.	query and database machine dependent

3.3.3 各種問い合わせに対するマシンの性能評価

代表的な 3つの問い合わせを用意し，各々のマシン上での処理時間を求めることによって比較を行う。実行環境としては単一ユーザモードを仮定する。

A. 単純な問い合わせの場合

query *Q*1:

retrieve (QTRCOURSE.day, QTRCOURSE.hour)
where QTRCOURSE.instructor = "despain, a.m."

ＱＴＲＣＯＵＲＳＥリレーションは1110タプル，274ページ（heap）から成り，1タプルは24ケのアトリビート，127バイトで構成される。day，hour アトリビュートは各々7，14バイトのキャラクタである。結果は3タプルとする。

このような単一リレーションに対する単純なＳＥＬＥＣＴＩＯＮからなる問い合わせについて各マシンの性能を検討すると，図3.3.1が得られる。

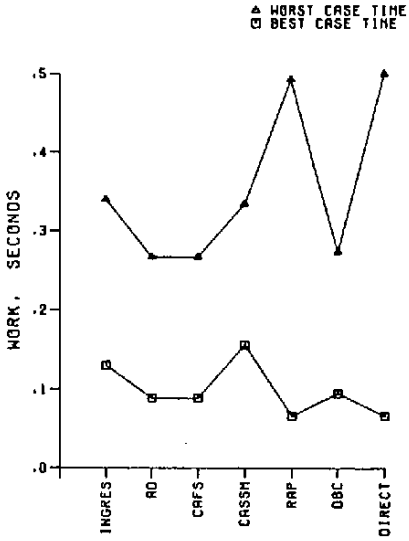


図3.3.1　Query Q1.

種々のパラメータの最小値，最大値に応じて，所要時間は最善と最悪の場合が計算でき，これら2つについて示されている。

対象リレーションは小さく1シリンダに収まるため連想ディスク（ＡＤ），ＣＡＦＳでは1回のディスク回転で処理が終了する。これに対し，ＣＡＳＳＭでは4キャラクタ以上の長いフィールドはエンコードしており，しかもプロセッサの機能が1回転に1アトリビュートの処理に限られているため，データが1シリンダに入り切った場合でも，ＱＴＲＣＯＵＲＳＥリレーションに属するタプルのマークに1回転，"despain，a.m"のストリングを見つけるのに1回転，その値をもつ全てのタプルにマークするのに1回転，dayアトリビュートを出力するのに1回転，hourアトリビュートを出力するのに1回転と，多数の回転が必要になることにより性能は低下する。ＤＢＣではＴＩＰを用いてサーチを行い，速度は連想ディスクとほとんど変わらない。ＤＩＲＥＣＴでは，ＣＣＤの中間記憶にリレーションがすでに格納されている場合には即座にサーチを開始出来るのに対し，そうでない場合にはディスクからリレーション全体を転送せねばならず，大きなオーバヘッドを生ずる。ＲＡＰの場合にも同様の事実が成り立つが，リレーションの大きさは9セル分であり，ＤＩＲＥＣＴではプロセッサが8台のため，1セル分の処理は2回目に残ることになるのに対し，ＲＡＰでは１６台のため，一回のスキャンで全てのタプルにマークを施すことが出来，ＤＩＲＥＣＴに比べてわずかによい性能が得られている。

以上の如く，一般には複雑な構成をとるマシンほどパフォーマンスがよくなると考えがちであるが，実際には前節で述べた如く，単純なセレクションやプロジェクションが主体であるような

場合には ＳＰＤＳ／ＭＰＤＳ型のマシンの方がすぐれていることがわかる。またソフトウェアによるアクセスパスだけを用いる場合でもかなり良い性能が得られることもわかる。

B．複数のリレーションにまたがる問い合わせの場合

```
retrieve ( ROOMS.building, ROOMS.roomnum,
                ROOMS.capacity,
                COURSE.day, COURSE.hour)
        where ROOMS.roomnum = COURSE.roomnum and
        ROOMS.building = COURSE.building and
                ROOMS.type = "lab"
```

ＣＯＵＲＳＥリレーションは11436タプル，2858ページ（90ＣＣＤページ），130トラック（７シリンダ）から成り，instructor name と course number に関するＩＳＡＭ構成とし，ＲＯＯＭリレーションは282タプル，29ページ，２トラック（１シリンダ）から成り room number に関するちらし編成とする。

このようなジョインを含む処理負荷の重い問い合わせについて各マシンの性能を検討すると，図3.3.2が得られる。

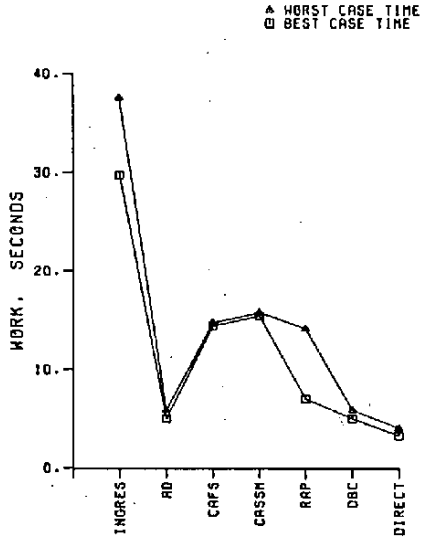


図3.3.2　Query Q2.

ＩＮＧＲＥＳで実行する場合，小さい方のＲＯＯＭリレーションに関して条件を満足するタプルをソートし，ＣＯＵＲＳＥリレーションの各タプルと比較を行うアルゴリズムを採用している。この問い合わせでは，ＣＯＵＲＳＥリレーションは７シリンダにもまたがる大きなリレーションであるため，その比較時間およびディスクからのデータ読み込み時間が大きな割合を占めること

になり，単一プロセッサではかなり負荷が重くなる。これらの単純な比較操作は並列に処理することが可能であり，MP型のマシンにより高速化することができると考えられる。MPDS型のマシンとしての連想ディスク，CAFS，CASSM，DBCについて比べてみると，先の2マシンが，後の2マシンに対してかなり優れていることがわかる。連想ディスク，およびDBCでの処理は，まずROOMについて restricted projection を施し，その結果に対し各々セレクションを発行することになる。これらのマシンでは単一リレーションの操作しか出来ず，このような展開型をとらざるを得ない。なお，DBCではクラスタリングインデックスが使える場合には検索空間を小さくすることが出来，大きく性能を向上出来ると考えられるが，インデックスを施すことの出来るアトリビュートは少なく，ここではジョインアトリビュートに関してクラスタリングされていないと仮定している。一方，CAFS，CASSMではHashed Bit Array を用いた処理方式によりジョインを実行するが，この問い合わせのジョインは，インプリシットジョインではなく，イクスプリシットジョインであるため，ホストでのタプル結合が必要となり，このためのオーバヘッドがこれらのマシンの速度を低下させている。従って，CAFSに対しても先と同じ処理手法を適用すれば同等の性能が得られると思われる。インプリシットジョインの場合にはCAFSの特長を活かすことが出来，かなり高速になると思われる。Bit Array を用いることにより11436タプルを422タプルに減らすことが出来たが，それでも47ページあり，これをソートするのにPPP11／70では14秒かかるとされ，ホストにおける処理の高速化が必要であろう。また，この場合には，ROOMの中で条件を満たすタプルは22個となり，片方のリレーションが極端に小さく，このため連想ディスクで採用しているアルゴリズムが効果をもつが，ソースリレーションもターゲットリレーションも同程度に大きい，比較的大規模なジョインを実行する場合にはディスクアクセスが頻繁になり大きく性能が低下すると考えられる。

RAP，DIRECTなどのMPIS型のマシンに関しては特にDIRECTが最も良い性能を示しており，負荷の重い演算に対しては比較的有効であることがわかる。ソースリレーションは22タプルであるため，8台のプロセッサ全てがこのページを取り込むことが出来，ターゲットリレーションの⅛のページ数を各プロセッサが処理すればよい。このO(m×n)のジョインアルゴリズムでは，容易に並列度を取り出すことが出来，各プロセッサ当たり12回のCCDページスキャンですむことになるがDIRECTでは16コのCCDバッファを有しており，プロセッサによる処理とI／Oを重畳させることが可能で，一層高速化されている。一方，RAPではマシン上でイクスプリシットジョインをサポート出来ないため処理方式は連想ディスクでの技法と同様となる。このようにMPIS型のマシンをMPDS型と同じように使うと，RAPの場合，

マーキングとタプル出力は分けなくてはならないため1つのセレクションに2回転必要であること，およびディスクからCCDに転送してからでないと処理を始められないことなどから判断して，MPDS型のマシンに劣るのは当然といえる。DIRECTではプロセッサに汎用μPを用い，結合操作自体をホストではなくバックエンド側で並列処理することが出来る唯一のマシンであり，このことからも高い性能が得られると思われる。DIRECTがMPDS型のマシンに比べてそれ程大きく性能が改善されていないのは主として一方のリレーションが小さすぎることによると考えられる。

連想ディスクではこの数が他のリレーションのディスクスキャン数に比例するのに比べ，DIRECTではリースリレーションをプロセッサ内にページレベルでバッファリングすることが出来るため，比較的大きくなった場合にもディスクI/Oは少なくてすむと考えられ，MPIS型のマシンの特長が明らかになると思われる。以上の如く，この問い合わせに対しては，MPDS，MPISマシンの優劣を評価することは困難であるが，並列処理により大きく性能を改善できることが示された。

C．アグリゲート関数を含む問い合わせの場合

*Q*3: retrieve (GMASTER.acct, GMASTER.fund,
encumb = sum (GMASTER.encumb by
GMASTER.acct, G MASTER.fund)).

GMASTERリレーションは194タプルからなり，1ページ当たり2タプルとする。従ってリレーションは1シリンダに入り，（acct，fund）の種別数は17とする。

SQLではgroup byに相当するこの演算は（acct，fund）に関してリレーションをソートし，各グループ毎に集計することにより結果が得られる。このような比較的複雑な単一リレーションに対する問い合わせについて各マシンの性能を検討すると図3.3.3が得られる。

DIRECTでは各プロセッサは汎用プロセッサであるため，集計を行うことが可能であり，各々割り当てられたサブリレーションに対しこの演算を行い中間結果をホストマシンに集めて全体としての結果を生成する。従ってプロセッサ台数分の並列度が得られ，最も処理時間が短かくてすむ。連想ディスク，RAP，CAFSでは集計機能がないため，ジョインと同じように一旦（acct，fund）に対するプロジェクションを行い，ホストでソートすることにより17タプルを生成した後，セレクションにより各イメージ毎の集計をとることになる。何回もディスクをスキャンする必要があるため，それ程性能の改善は期待出来ない。CASSMのプロセッサだけは集計機能を有しており，これによりホストの介在なしで結果を得ることが出来る。すなわち，（acct，fund）に対し1つずつ取り出してはそれと同じ値をもつタプルにマークを施し集計

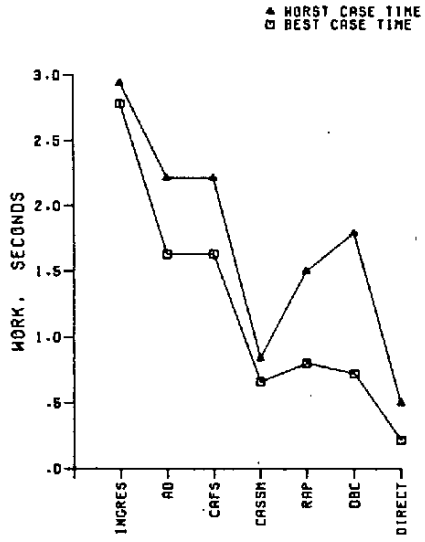


図3.3.3　　Query　Q3.

していけばよい。ホストの負荷がほとんどなく，ＤＩＲＥＣＴに次いでよい性能を示している。ＤＢＣではソート機能が付加されており，これを用いて（ acct，fund ）の重複を取り除くことが出来る。従って，連想ディスクと異なり，ホストとＤＢＭとのデータ転送量は大きく減少し，その分だけ性能が向上すると考えられる。以上の如くアグリゲート関数を含む問い合わせに対しては，そもそもその機能を有しているかどうかという点が最も大きく影響することがわかる。ＤＩＲＥＣＴでは汎用 μP を用いることによりこの問題を解決している。また，データ処理の負荷自体は比較的重く，従ってＭＰ型のマシンにより性能を改善出来ることは明らかである。

3.3.4 結　論

以上３つの問い合わせに関する各マシンの評価により，処理負荷の重い問い合わせに対してはデータベースマシンは十分大きな効果を示すことが示された。ＤＩＲＥＣＴではＲＡＰの如く単純なプロセッサではなく汎用 μP を用いることにより比較的多様な処理に対処することが出来，またクロスバを用いてプロセッサとメモリに任意の結合関係を生成することにより柔軟な処理形態がとれるため，比較的高い性能が得られているが，マシンのコスト面からの評価も必要であろう。一方，単純な問い合わせに対してはＭＰＩＳ型のマシンはそれ程有用ではなく，ＭＰＤＳ型の連想ディスクで十分であることがわかった。従って，データ処理負荷の重い問い合わせと単純なサーチの両方を効率よく実現するには，ＭＰＣＳ型のマシンにより，ＭＰＤＳ型とＭＰＩＳ型のマシンの長所を持たせればよいことになる。この考え方は自然であり，８０年代のマシンには

このタイプのマシンが存在する。[51)]

もちろん，3種類の問い合わせに対する評価だけでは十分とはいえず，大規模なリレーション間でのジョイン，プロジェクション，多段のジョイン，クラスタリンの効果，マルチユーザ環境での評価等，今後より詳細な検討が必要である。

3.4 各種データベースマシンとその特徴

現在までに，数多くのデータベースマシンが各所で開発されてきた。ここではそれらの概容を説明する。必ずしも全てのマシンが網羅されているわけではない。ソートやパターンマッチに関するハードウェアアルゴリズムはデータベースマシンを構築する上で重要な技術と考えられるが，ここでは省略する。データベースマシンとしてのアーキテクチャを提案しているものを中心に解説する。

ここで紹介するマシンとその中心となる提案者名を載げると以下のようになる。

- RAP — E. A. Ozkarahan S. A. Shuster
- RAP.3 — E. A. Ozkarahan
- CASSM — S. Y. W. Su　G. J. Lipovski
- EDC — S. Uemura
- CAFS — ICL
- RARES — C. S. Lin
- サーチプロセッサ — H. O. Leilich
- IDM500 — Britton Lee
- Data Computer — CCA
- シストリックアレイDBM — H. T. Kung
- DBCジョインプロセッサ — M. J. Menon　D. K. Hsiao
- SRM — B. Arden
- セルラアレイDBM — M. Muraszkiewicz
- DIRECT — D. J. DeWitt
- Data Flow DIRECT — D. J. DeWitt
- Tree Machine — S. W. Song
- MICRONET — S. Y. W. Su
- DIALOG — B. Yao
- WCRC — S. K. Arora
- RELACS — E. J. Oliver
- DBC — D. K. Hsiao
- バブルDBM — H. Chang
- XDMS — R. H. Canady
- データフローデータベースコンピュータ — Y. Tanaka

- GRACE　　M. Kitsuregawa

その他国内では，横浜国立大学，慶応大学，広島大学，電々公社等でもデータベースマシンの研究が活発になされている。

3.4.1 RAP 4)～9)

本マシンは1975年，Tronto大学でOzkarahanを中心に研究が開始された。リレーショナルデータベースマシンの草分け的存在といえる。

A．アーキテクチャ

RAPの全体的な構成を図3.4.1に示す。

各セルは図3.4.2に示される如く，回転型メモリの1トラックとそれに付随したサーチプロセッサ（ISMU）からなる。RAPはSIMDマシンであり，コントローラからの指令により各セルがそのトラック内のデータに対して同一の処理を施す。各セルでは幾つかの比較器により並列に処理することが可能になっている。

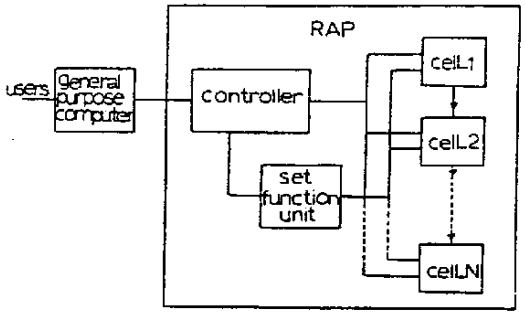


図3.4.1　Overview of RAP architecture

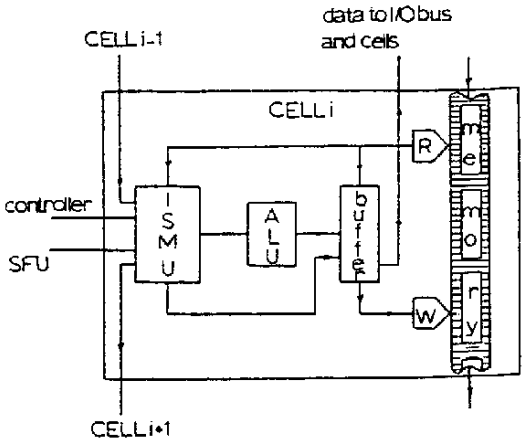


図3.4.2　Overview of cell architecture

B．データ構造

RAPではタプルに対して固定長表現を用いている。すなわち，1リレーション内では全タプルに関して長さは一定である。また1セル内にはただ1つのリレーションしか含まれない。トラック書式は図3.4.3に示される。1つのタプルが1つのブロックに格納される。各タプルには4つのマークビットが設けられており，これにより中間リレーションを表現する。また消去フラグは，タプルの削除に使用され，各セルはコントローラからの指令なしに，独自に有効データをト

ラックの前へつめる機能を有している。

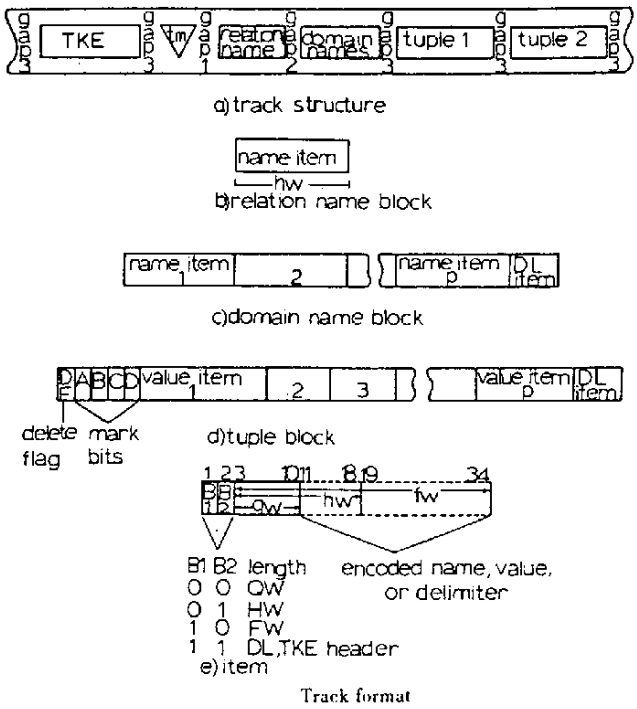


図 3. 4. 3

C．処理方式

ＲＡＰには図3.4.4に示されるごとき種類の命分が用意されている。以下，いくつかの関係代数演算についてその手法を示す。

I—Instruction Timing

TYPE OF OPERATION	INSTRUCTION	EXECUTION TIME IN NUMBER OF REVOLUTIONS UNLESS OTHERWISE SPECIFIED
Retrieval commands	MARK	1
	RESET	1
	READ	Minimum=1/2 on average. Maximum=Number of qualified tracks occupied by the relation
	READTREG	Negligible
	CROSS_MARK	Depends on data base and k (Reference 10)
	CRS_COND_MARK	Same as CROSS-_MARK +1
	GETTFIRST_MARK	1 +Fraction
	GETTFIRST	1
	SAVE	1
Update commands	ADD	1
	SUB	1
	MUL	1
	DIV	1
	REPLACE	1
Set function commands	COUNT	1 +Fraction
	MAX	1 +Fraction
	MIN	1 +Fraction
	SUM	1 +Fraction
	AVERAGE	2 +Fraction
Insertion and deletion commands	DELETE	1
	INSERT	Minimum = 1. Maximum = 2
	DROPTDOMAIN	Maximum number of relation tuples per track
Data base creation and destruction commands	DESTROY	Fraction
	CREATE	1
Decision and transfer commands	TEST	Negligible
	BC	Negligible
	EOQ	Negligible

図3.4.4

a．セレクション

各セルでそのトラック内のデータを1つずつ読みながら，流れに追従して条件判定がなされる。複数ケの比較器が設けられており，これに入いるだけの評価を一回転に行うことができる。条件式が複雑な場合は，中間結果をマークビットを用いて残しておき，次の回転で残りの処理を施す。選択されたタプルはコントローラを通じてホストに転送される。多数のセルからの出力要求が同時に生じた場合，出力できるのは1つのセルだけであり，出力待ちタプルを保持する際にもマークビットが用いられる。

b．ジョイン

ＲＡＰではインプリシットジョインしかサポートすることはできないが，このために専用命令（CROSS MARK, CRS COND MARK, GET FIRST MARK）およびセル間の通信機構が設けられている。まずソースリレーションのタプルのジョインアトリビュートをとり出し，これをターゲットリレーションを構成する各セルの比較器へ転送する。その後，ターゲットリレーションを１回転して一斉にジョインをとる。結果はマークビットによって記憶される。ターゲットリレーションを納めるだけのセルがあるとすれば，ソースリレーションのカーディナリティを各セルの比較器の数で割っただけの回転によってジョインを行うことができる。従って，一般には多数回のディスク回転を必要とし，それ程性能は改善されない。

c．プロジェクション

GET FIRST MARK 命令によってプロジェクション処理を行う。すなわち，いくつかタプルを比較器に取り込んでは，それと重複するタプルを順次除去してゆく手法をとっており，この場合にも多数のディスク回転が必要となる。

D．そ　の　他

ＲＡＰのごときセルラーロジックタイプのデータベースマシンでは，一般に大容量のデータベースをサポートしようとすると，容量に比例したコストが見込まれ，実現的ではない。これに対して，ＲＡＰでは，データベースの仮想記憶化を考えている。図3.4.5に示されるごとく１セルに対し２つのメモリを用意し，ＲＡＰへのデータベースローディングと処理とをパイプライン化する方式を提案している。すなわち，必要な部分をまず，ＲＡＰへスージングして，その後処理が施されることになる。

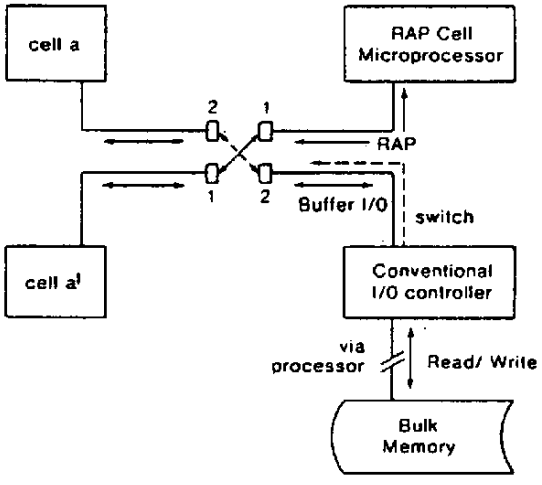


図 3.4.5　A track pair in the virtual memory configuration

3.4.2 R A P. 3 10)

RAP3はTronto大学における一連のRAPマシンの開発の後を受け，Middle East Technical大学で現在開発中のデータベースマシンである。

A. アーキテクチャ

RAP.3はRAP.1，RAP.2と同様，論理セル型のデータベースマシンであるが，各セルはさらにサブセルと呼ばれる多数のプロセッサからなる（図3.4.6）。記憶媒体としては磁気バブル，または大容量のRAM等，電子的なスタート／ストップ機能を持った回転メモリが仮定されている。回転メモリとサブセルとのタプル転送にはDMAが用いられる。

タプルは回転メモリ内に図3.4.7に示すように配置され，ワードシリアルに読み出される。タプルの構造を図3.4.8に示す。マークビットは2バイト，タプル長は1024バイト以下とされている。

B. セル内における処理方式

サブセルは，マイクロプロセッサと小容量のRAMからなる。回転メモリは個々のサブセルに対し，1タプルずつを順に送出する。各サブセルは1タプルを受け取ると同時にこのタプルに対する処理を行い，処理の終了と共に当該タプルを回転メモリに送り返す。処理は図3.4.9に示すようにパイプライン化して行われる。

パイプラインを乱さないためには，1タプルの処理時間が（k－2）×T_{LS}（k：サブセル数，T_{LS}：1タプルをロード／ストアする時間）に等しいことが必要であるが，一般にこのような状況は期待できない。RAP.3では回転メモリにスタート／ストップ機能を要請し，処理に必要な間回転を停止させることによってこの問題を解決している（図3.4.10）。

C. 評　　価

RAP.3は，問い合わせ処理をサブセル内のマイクロプロセッサとRAMを用いて行うため，ドメイン値の比較処理等が効率良く行えると同時に以前のRAPマシンにおいて，レジスタ容量不足から派生していた性能低下が改善されている。シミュレーションの結果によれば，RAP.3は以前のRAPマシンに比べて3～6倍の性能が得られるとされている。

D. 実　　装

現在，RAMを回転メモリとし，4つのサブセルを持つセルが実装されている。各サブセルはインテル社の8086，および2kバイトのRAM，1Kバイトの制御用ROMからなる。既存のマイクロプロセッサを用いたため，実装時のチップ数は大幅に減少し，1セル当たり160個程度とされている。

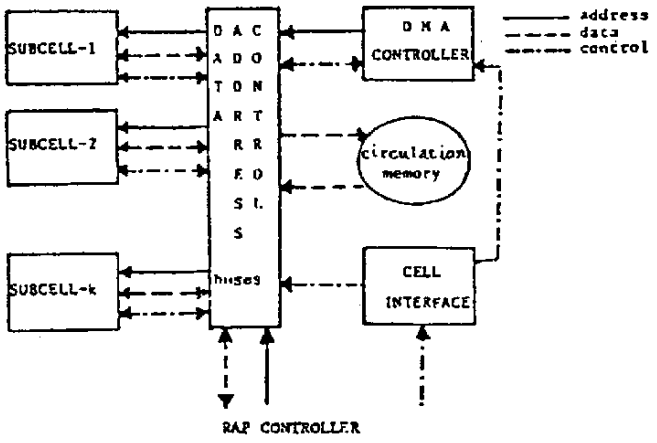


図 3. 4. 6

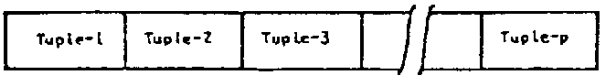


図 3. 4. 7

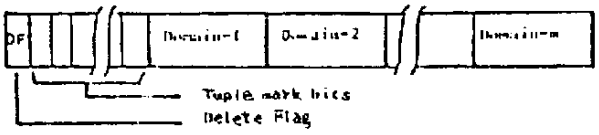


図 3. 4. 8

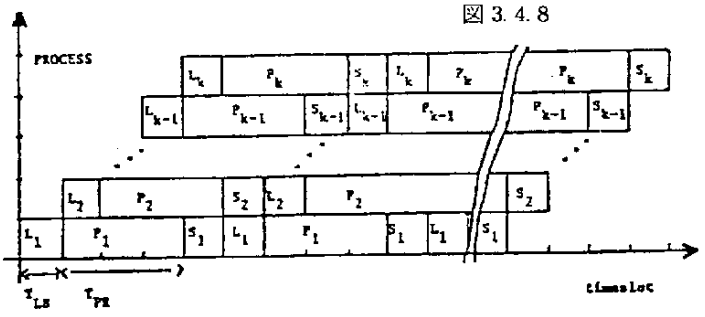


図 3. 4. 9

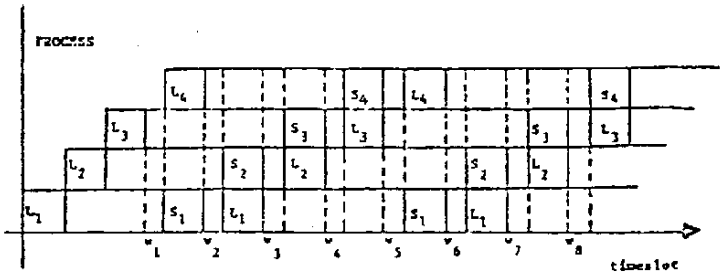


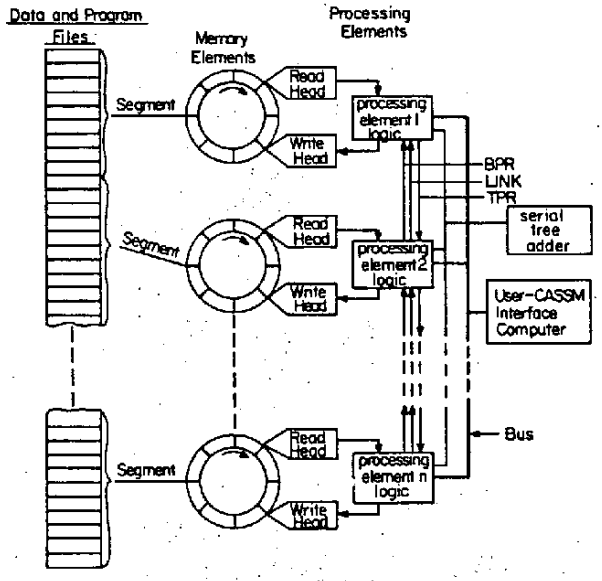
図 3. 4. 1 0

3.4.3 CASSM 11)～13)

本マシンはFlorida大，S.SUらによって開発された最初のセルラーロジックタイプデータベースマシンである。CASSMでは階層モデル，ネットワークモデル，リレーショナルモデルの3つが共にサポート可能である。

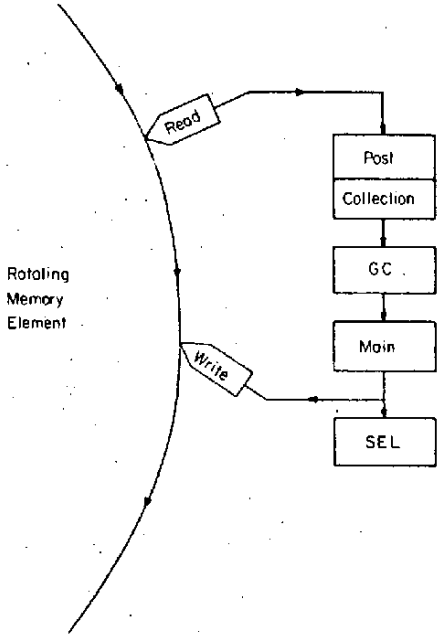
A．アーキテクチャ

CASSMは，図3.4.11に示される構成をとる。各セルの1トラックに，論理的には連続したファイルの1セグメントが格納される。各セルには，読み出しヘッドと書き込みヘッドをもった処理要素が付加されており，データ流に追従した処理がなされる。



The overall architecture of CASSM.

図3.4.11

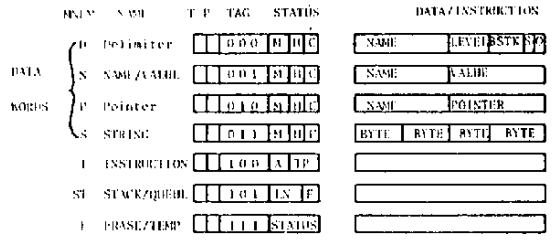


General view of a CASSM cell.

図3.4.12

各セルは図3.4.12に示されるごとく4つのサブシステム（POST，GC，Main，SEL）から構成され，これらの間でパイプライン処理がなされる。CASSMでは記憶媒体上にデータだけではなく命令も書き込まれる。SELには読み込まれた命令が一旦貯えられる。MainがSELより命令をとり出し，入力データ流に対して処理を行う。Postは終了処理を，GCはガーベージコレクションを行う。

表3.4.1



Identifier	Meaning	Identifier	Meaning
A	Marks an active instruction	STATUS	Qualifies the ERASE/TEMP words: 1 0 0 End Of File 1 1 1 GARBAGE word all other values are TEMP words
BSTK	6 bits bit-stack for search results		
BYTE	ASCII character		
C	Collection bit		
F	Flag bit, marks end of list	TP	Qualifies INSTRUCTION words: 0 0 to 1 0 instruction word 1 1 immediate operand
H	Hold bit		
LEVEL	Binary number		
LN	Stack or queue number	VALUE	Binary number
M	Match bit		
NAME	Code word		
POINTER	Record number		
Q	Q bit		
S	S bit		

CASSM word types.

B．データ構造

回転型記憶上のファイルは，４０ビットの語より構成される。一語は，３２ビットのデータ部（デリミタ，属性名と属性値のペア，文字列など）と８ビットのタグおよびステータスビット（マークビット）からなる。表3.4.1に示されるような各種のデータタイプが存在する。

C．ジョイン

ＣＡＳＳＭはＣＡＦＳと同様，Value Matchingと呼ぶビットアレイを用いたアルゴリズムを採用している。すなわち，各セルはＲＡＭビットアレイを有しており，これらは論理的にはとなり合うセル同士連結した一本の長いビットアレイを構成している。まず始めにソースリレーションの全てのタプルについてそのジョインアトリビュートの値をもとにビットアレイに１を立てる。このアドレスは長いビットマップに対して生成され，当該タプルに対するビット位置が別のセルに存在することもあるため，セル間の相互作用が必要となる。この後，ターゲットリレーションのタプルに関して同様の手続きを行いビットアレイ上１であればジョインがとれることになる。図3.4.13に処理の様子を示す。

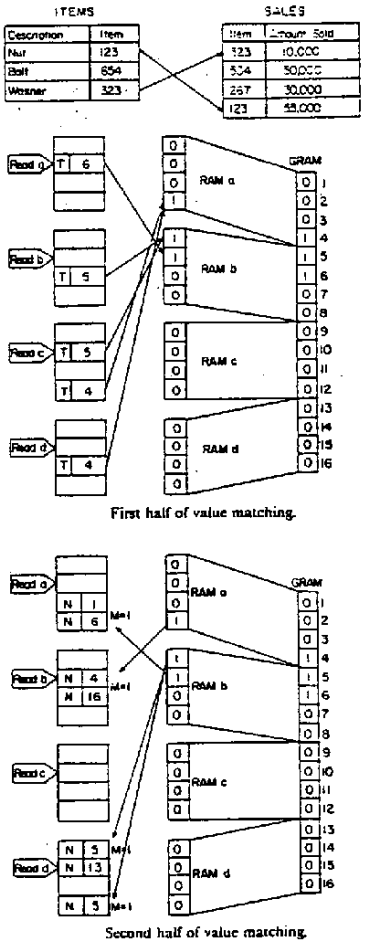


図 3.4.1 3

3.4.4 E D C 27)

EDC（Electronic-disk oriented Database Complex or ETL Database Computer）は電子技術総合研究所で開発中のデータベースマシンである。EDCは2次記憶として磁気バブルを本格的に採用した最初のマシンである。

A. アーキテクチャ

EDCの基本構成を図3.4.1 4に示す。EDCはマイクロプロセッサ，マイクロプログラムメモリ，主記憶，磁気バブルメモリからなるデータモジュールを複数台，バス結合した構成を採る。データモジュールの内の1台は制御モジュールと呼ばれ，マシン全体の動作を制御する。データ（ファイル，リレーション）は複数個のデータモジュールに渡って分散配置され，バンドと呼ばれる（図3.4.1 5）。制御モジュールから発せられたコマンドは各データモジュールにブロードキ

ャストされ，モジュールのバッファに一旦貯えられる。各モジュールは独自のスケジューリングによってこれを実行する。

B．通信機構

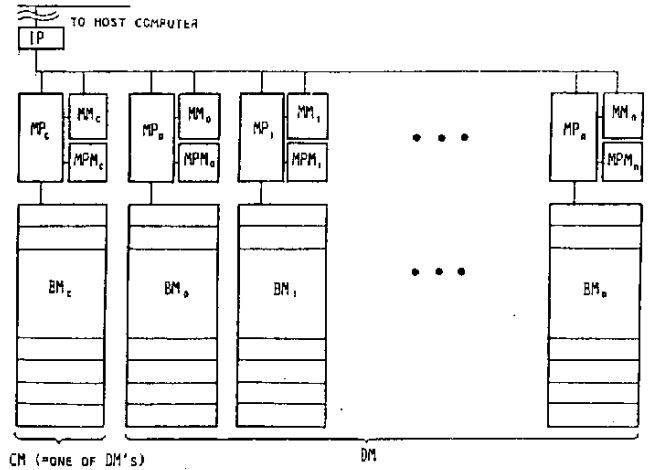
モジュール間の通信は，制御モジュールが他のデータモジュールの主記憶を共有することにより行われる。また，n対1通信，すなわち駆動されたデータモジュールが制御モジュールに実行終了を伝えるような場合に対しては特殊なカウンタを用いた制御が用いられ，制御モジュールの割り込み処理の負荷を軽減している。

C．抽象マシンのインタフェース

ＥＤＣは図3.4.16に示すように6つの階層に分けられ，これら階層間のインタフェースが定められている。ＥおよびＤインタフェースはデータモデルに依存した問い合せをモデル独立なＣインタフェースに変換する。Ｃインタフェースはバンドを対象とした命令からなるが，Ｂインタフェースはデータモジュールを対象とした命令からなり，これが各モジュールにブロードキャストされる。Ａインタフェースは内容呼出しを含む磁気バブル入出力操作を行う命令水準である。

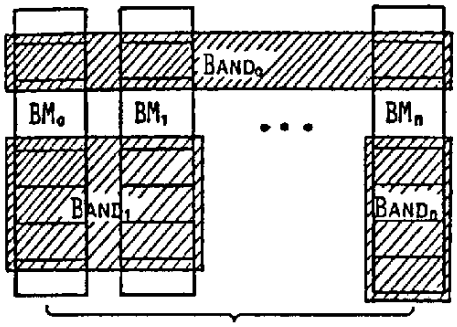
D．評　　価

ＥＤＣは将来の磁気バブルの容量増大を念頭に設計されたマシンであるが，一方で磁気バブルの大容化に伴って，内容呼出しの時間も増大し，このままのアーキテクチャでは実用規模のデータベースを対象とした処理ではディスクを用いた逐次処理と大差ないことになる。磁気バブルはその記憶構造上，アクセスタイムは改善されつつあり，単純な全数サーチではなく，その特徴を活かした応用を考える必要があろう。



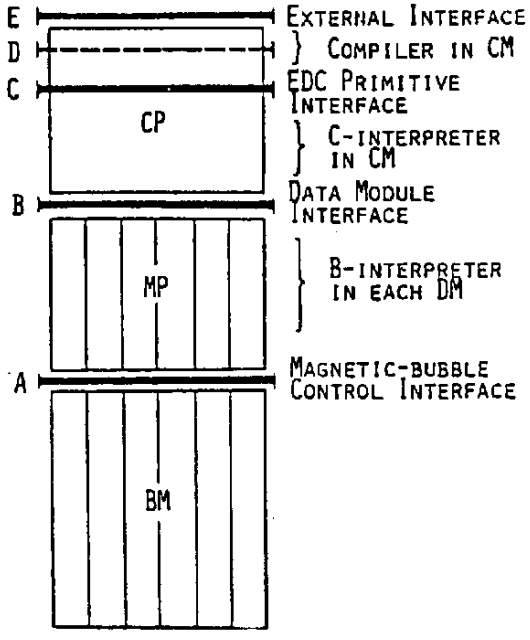
System Architecture of EDC.

図3.4.14



Data Storage Scheme.

図3.4.15



Levels of Abstract Machines.

図3.4.16

3.4.5 CAFS 24) 25)

本マシンは，ICL社によって開発されたリレーショナルデータベース用のデータベースマシンである。CAFSはセレクション，プロジェクション，ジョインなどの関係代数演算を効率よくサポートすることができる。

A．アーキテクチャ

CAFSのアーキテクチャを図3.4.17に示す。セルラロジックタイプのマシンとは異なり，記憶セルとロジックの対応は固定的ではなく最大12ディスクチャネルまでのデータストリームをマルチプレクスして処理することが可能である。CAFSでは，可変長レコード，および可変長フィールドを許しており，これはレコードの各フィールド毎に長さのコードをもたせていることによる。また16個のキーレジスタが条件検索用に使用できる。

B．ハッシュビットアレイ

CAFSでは，64Kbitのハッシュビットアレイを複数個用い，ジョイン処理における候補のふるい落とし，プロジェクション処理における重複タプルの除去に利用している。

ジョイン処理では，アトリビュート値毎に異なる値をとるカップリングインデックスを用意する方式とインデックスは持たずに直接ジョインアトリビュートにハッシュを施す方式の２種類が考えられる。後者ではハッシュ関数は完全でなくシノニムが生ずる可能性があるため，ホストマシンで正確なジョイン処理を行う必要がある。前者の方式ではシノニムはなく，必ずジョインがとれるが，タプルの結合操作自体はやはりホストマシンで行わねばならない。セミジョインの場合には問題ない。

プロジェクション処理では，ハッシュ関数を用いた場合，シノニムが生ずると異なるタプルを重複タプルとみなす可能性がある。これに関しては，ＣＡＦＳではハッシュ関数をいくつか用意することによって，それらが同時にシノニムを発生する確率をきわめて低くしている。図 3.4.18 にハッシュビットアレイを有するＣＡＦＳの構成を示す。

C．そ　の　他

ＣＡＦＳでは上記のごとく強力な検索能力を有しているが，オンラインによる更新は禁止している。

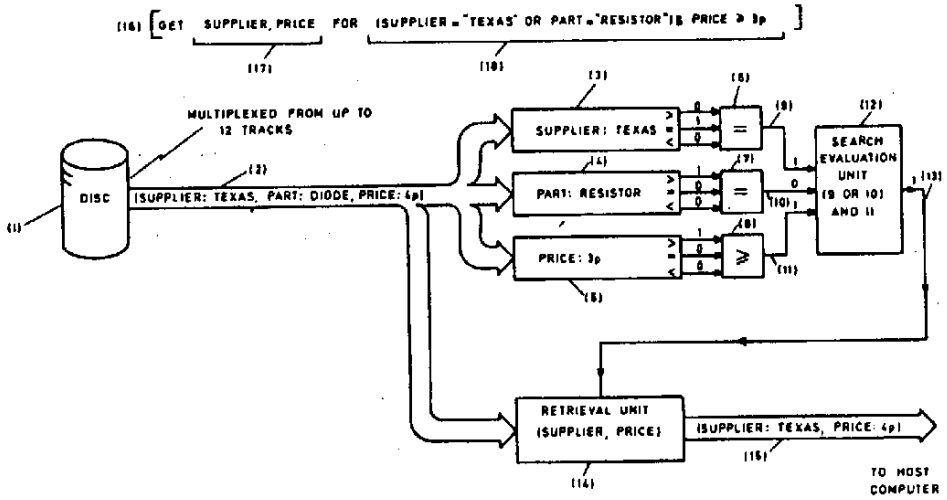


図 3.4.17　Basic content addressable file store (CAFS) architecture

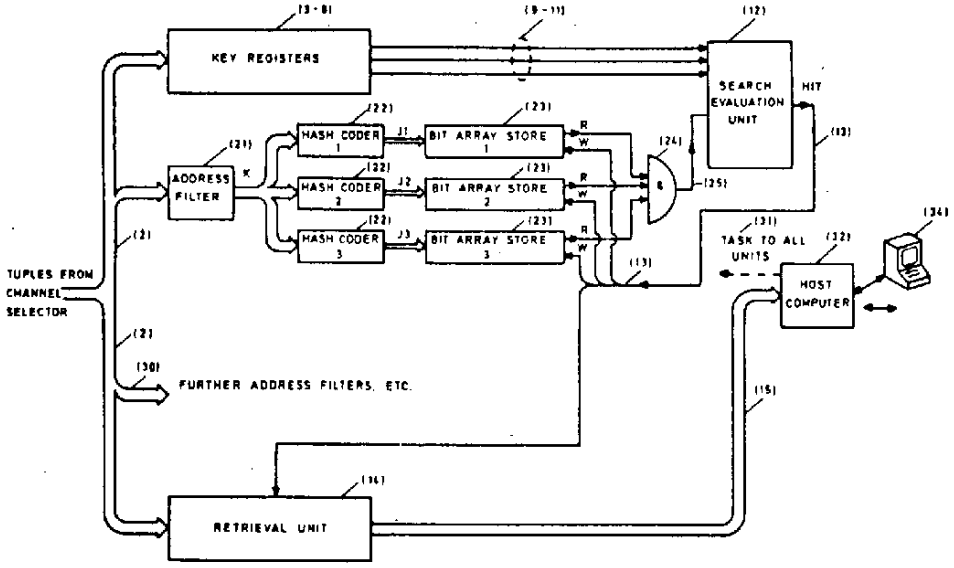


図3.4.18　CAFS architecture with the hashed address bit array store

図3.4.18

3.4.6 RARES 39)

本マシンはユタ大学のC.S.LINによって1976年に提案された。

RARESでは多重トラック同時処理に伴う出力問題に着目し，高い出力レートを得るために，トラックを横切る形でタプルを配置する方式を採っている。一般に2つ以上のタプルが1つのセクターのトラック群を横切って記憶されることが可能であり，いくつかのタプルが尚並列に処理される。従って，RARESもRAP同様出力調整機構を要するが，コンフリクトはかなり減少する。

A. 記憶構造

図3.4.19～3.4.21に示されるごとく，バンドという単位でディスクを分割する。1つのバンドは最高64トラックからなり，リレーションは，1つまたはそれ以上のバンドに分配される。バンドの幅は，できる限り無駄なスペースが少なくなるように選ぶべきであるが，あまり狭くしすぎるとこの配置の効果がなくなる。

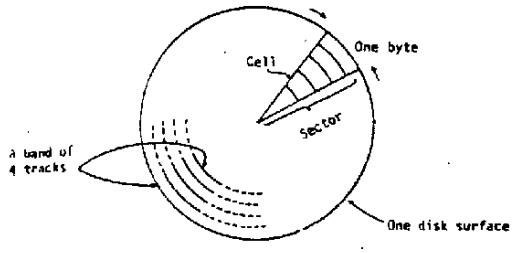


図3.4.19　Cells, sectors, and bands on a disk surface

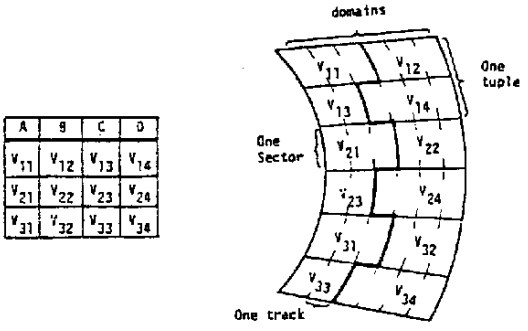


図3.4.20　A relation and its mapping to part of a band

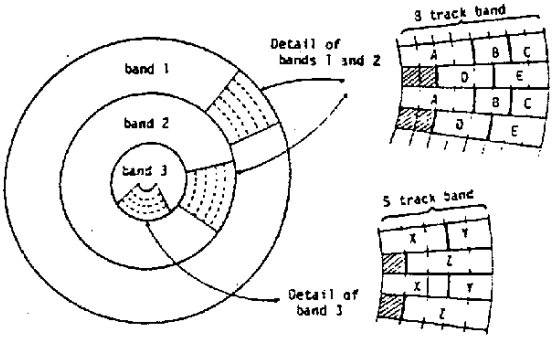


図3.4.21　A surface with three bands

B．サーチメカニズム

ＲＡＲＥＳの検索モジュールには隣接したトラックの集合が割り当てられる（２５６トラック毎）。その範囲内でバンドを処理するが，各モジュールに割り当てるトラックは互いに重複していてもよい。（図3.4.2 2 ）

各サーチモジュールの構成は図3.4.2 3に示される。ＲＡＰと異なり中間結果をマークビットとして２次記憶媒体上に残すのではなくレスポンスストアと呼ぶモジュール内のＲＡＭに記憶する。出力バッファは，各モジュールに付随する６４本のデータバスに直結している。

ＲＡＲＥＳでは，〝オーソゴナル　レイアウト〃という記憶形式を採用している点，サーチロジックが記憶セルに固定されておらずある範囲内で異なるバンドに移動できる点に特長がある。

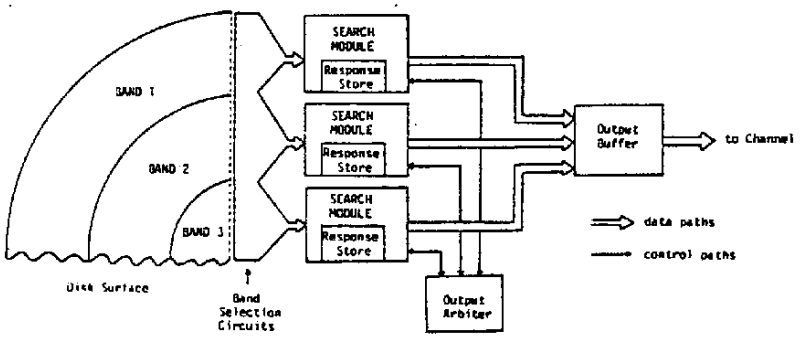


図3.4.2 2

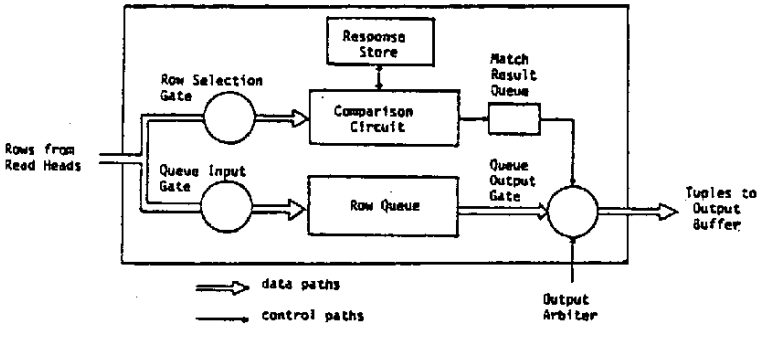


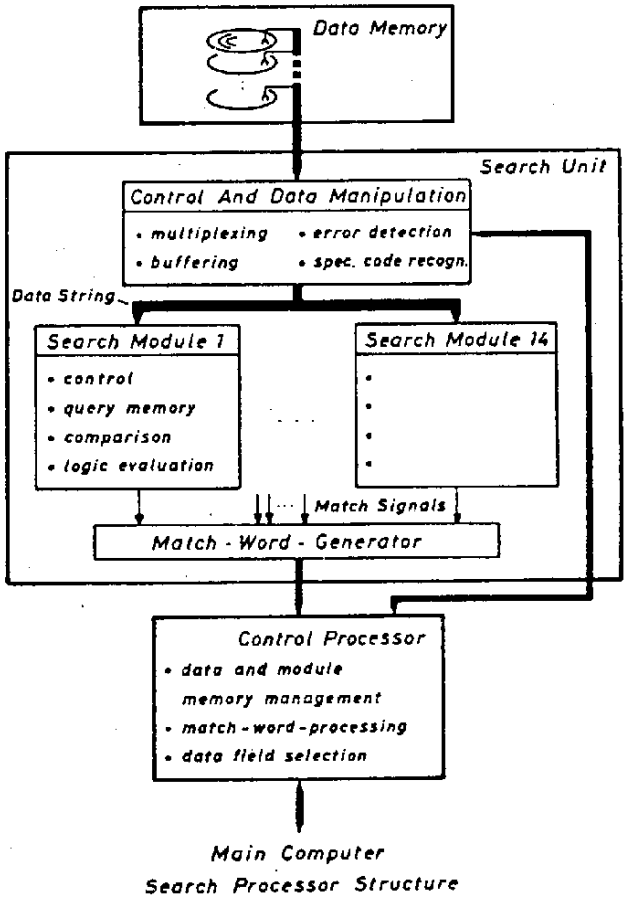
図3.4.2 3

3.4.7　サーチプロセッサ　26)

本マシンは1974年Braunschweig大のH.O.Leilichらによって研究が開始されたサーチ専用のマシンである。

A．システム構成

サーチプロセッサの構成は図3.4.24に示されるごとく3つの大きな構成要素，データメモリ，サーチユニット，コントロールプロセッサからなる。データメモリは，可動ヘッドディスクで構成され，9ケのトラックから並列にデータ転送がなされる。またコントロールプロセッサはミニコンで構成され，ホストとサーチプロセッサ間の通信をする。サーチユニットには14コのサーチモジュールが設けられ，ディスクの1回転内に1シリンダのデータに対し14コの独立した問い合せを処理することが可能である。サーチモジュールにはコントロールプロセッサによって問い合わせプログラムがロードされる。



Search Processor Structure

図3.4.24

B．サーチモジュール

サーチモジュールは図3.4.2 5に示されるごとく，問合せプログラムを格納するQuery Memory（24bit×1K語）および命令カウンタ（IC），命令レジスタ（IR），オペランドカウンタ（OC）などのレジスタ，比較器などからなる。レコードの評価は，ICによって命令を取り出した後，OCを用いてQuery Memory内のオペランドと入力データ流を順次比較することによって行われる。この際，制御網により比較器と論理評価機構が制御され，最終的な判定が求まる。図3.4.2 6に示されるように，サーチプロセッサの命令の中には，種々の比較命令や，リストリクションのためのLV命令などがふくまれる。

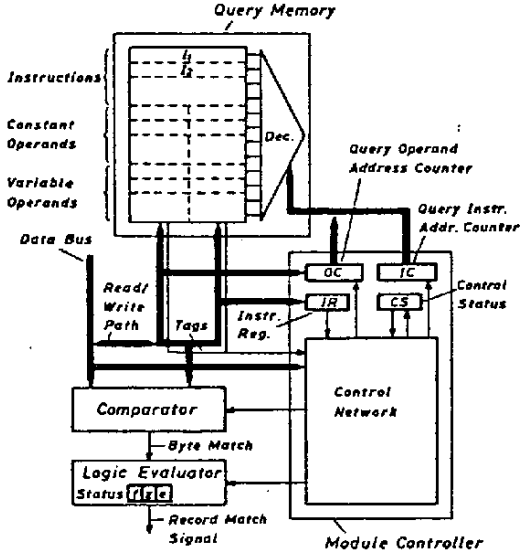


図3.4.2 5

General Structure of the Search Module

Query-Instruction	Instruction Code 4	Modificator Bits 5	Operand Address 11

Instruction Code:

ID (M)*	compare	identity
GD (M)*	compare	greater than
KD (M)*	compare	less than
ME (M)*	compare	masked bits are all "1's"
MN (M)*	compare	masked bits are all "0's"
NL		no operation
KR		skip rest of category
KN		end of category
SR		skip rest of record
SN		end of record
LV		load variable

*) instruction for constant/ variable operands

Modificator Bits:

FI	invert item's match result
BO	boolean operation (and/or)
TE	apply to second status level
WK	repeat instruction within category
WB	boolean operation when repeating

Extension of Instruction Code

Using modificator FI, an item's original match result is inverted before boolean processing. Virtually 5 more comparison operations exist, all of which can use either constant or variable operands:

ID, GD, KD, ME, MN

List of Instructions and Modificator Bits

図 3.4.2 6

C．実　　装

9トラック分のデータを1本のストリームとし同時に処理するためには，7.5MHz（137 nsec／バイト）というかなり高い処理速度が要求される。ECL等を用い，デバイスを高速化することにより対処可能であるが，ここでは処理のパイプライン化により解決を図っている。すなわち，全体をメモリアクセス，比較，論理評価の3つに分け，これらをパイプライン的に処理することにより各サブプロセス自体を137 nsec時間に延ばすことができる。（図3.4.27） なお，一般にパイプライン処理では分岐命令の取り扱いが難しいが，ここでは9トラック分の処理をマルチプレクスしているため，パイプの中に同一トラックの命令が2つふくまれることはなく，この問題は回避されている。

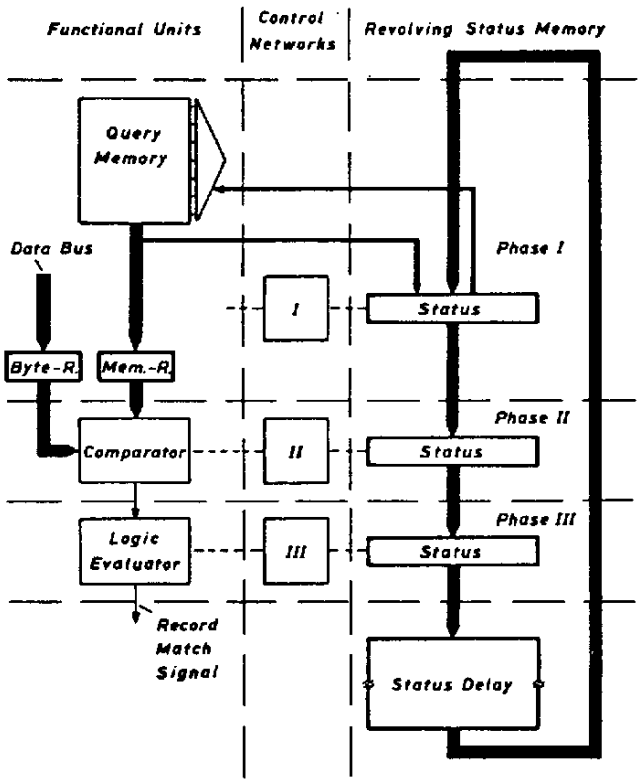


図3.4.27　*The Search Module as a Pipeline Processor*

D．性　　能

単純なセレクションでは，アクセスパスが張られている場合，サーチプロセッサはそれほど有効であるとは考えられないが，補助的記憶構造が存在しない場合，条件式が複雑な場合，リスト

リクションの場合には，汎用マシン上でのソフトウェアによる処理と比べ，大きな性能の向上が期待される。なお，ジョインやプロジェクションに関する検討はほとんどなされていない。

3.4.8 IDM500 19) 20)

IDMは，米Britton Lee 社で開発された商用データベースマシンであり，関係モデルデータベースをサポートする。

A．適用データベース規模

Britton Lee 社はデータベースユーザを図3.4.28に示されるごとく，いくつかのクラスに分類し，そのターゲットを明らかにしている。すなわち，非常に高い性能の要求されるデータベースシステムでは，多重プロセッサ構成をとるDIRECTなどのマシンが適用されると考えられるが，現実には小中規模の多くのシステムが存在し，これらのユーザ層では性能よりもむしろコストが問題となり，コスト／パフォーマンスの高いマシンが求められる。IDMは，中規模データベースのサポートを目的としている。

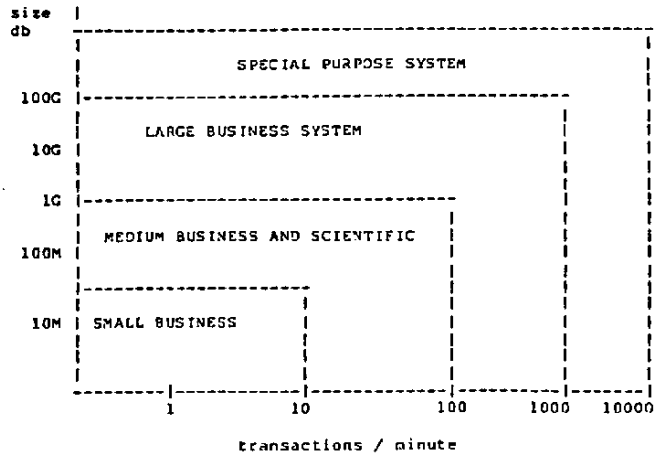
B．システム構成

中規模ビジネス応用のデータベースではアクセスパターンが明らかである場合が多く，したがってIDMでは，Logic per Track による全数サーチは行わず，ソフトウェアによるアクセスパスのサポートを中心としている。しかし，ショットキーTTLを用いた，パイプラインアーキテクチャの〝データベースアクセラレータ〟を付加することもでき，これにより，より高い性能を得ることが可能となっている。

また，記憶媒体としては可動ヘッドディスクを用いているが，これにディスクキャッシュを付加することもできる。IDMでは，最大容量32Gバイト，最大処理速度2,000トランザクション／分が可能とされている。

C．ホストインタフェース

IDMでは，ホストに対する独立性を保ち，またその間の通信負荷を減らすため，インタフェースは関係モデルに対する問い合わせレベルとされている。また本マシンが目指す程度のデータベース使用環境下では，交通機関の座席予約等，定形的な処理も多く，IDM自身に問い合わせプログラムを格納し，ホストインタフェースをプログラムコールレベルまで上げることも可能である。これによって一層，通信オーバヘッドを軽減できる。さらに，ホストI／Oプロセッサを付加することにより，ホストとの通信，およびバッファ管理を切り離し，性能向上を図れるとしている。



User classification.

図3.4.28

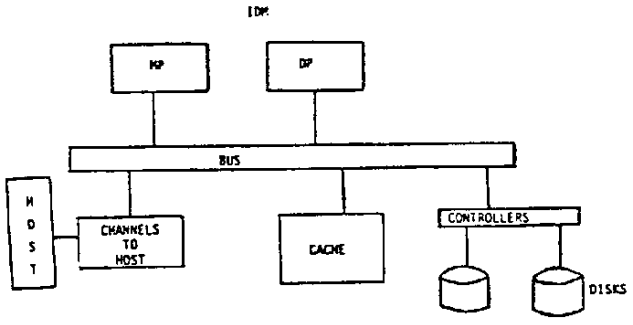


図3.4.29

3.4.9 Data Computer 18)

データコンピュータはＡＲＰＡネットワーク上の１つのユーティリティとしてＣＣＡ(Computer Corporation of America)社によって開発されたシステムであり，大容量データの蓄積および管理を司るマシンである(図3.4.30)。データコンピュータは，(1)ＡＲＰＡネットワーク上の異種計算機間におけるデータの共有 (2)大容量，低価格なオンラインファイルの提供 (3)データベース管理機能の提供の３点を目的としている。そして(1)のためにデータフォーマットの変換機能を備え，また(2)のためにはアンペックス社のテラビットメモリを用い，大記憶容量を提供している。さらに(3)のためにData Language と称する高級言語を用意し，またデータ検索の効率化，可変長データ更新の効率化などを考慮し，inverted file，chapterd file，CAT(Container Address Table)などの機能が備えられている。

A．アーキテクチャ

図3.4.31にデータコンピュータのハードウェア構成を示す。メモリは1次（主記憶），2次（ディスク），3次（Tera Bit Memory）と3階層構成を採用している。テラビットメモリはビデオテープの技術を用いた大容量記憶装置であり，媒体としては2インチ幅のビデオテープを用い，データはブロック化されて記録される。各ブロックには一意なアドレスが付けられており，ランダムアクセスが可能である。またアドレスは1,000inch／secで極めて高速にサーチされる。ブロック平均アクセス時間は15秒である。データチャネルは6×10^6bit/secの転送速度を有し，別々のトランスポートに対し同時にread，writが行なえる。RAPやCASSMなどと異なり，特殊なハードウェアは用いていない。ARPAネットワークとのインタフェースのために，IMP（Interface Message Processor）が設けられている。

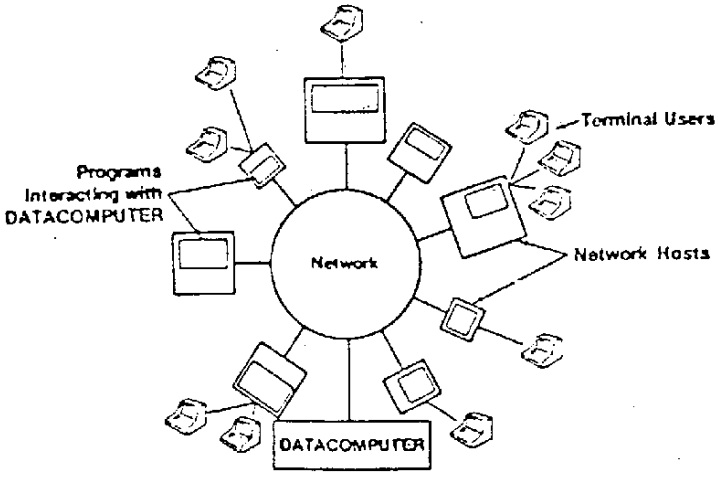


図3.4.30

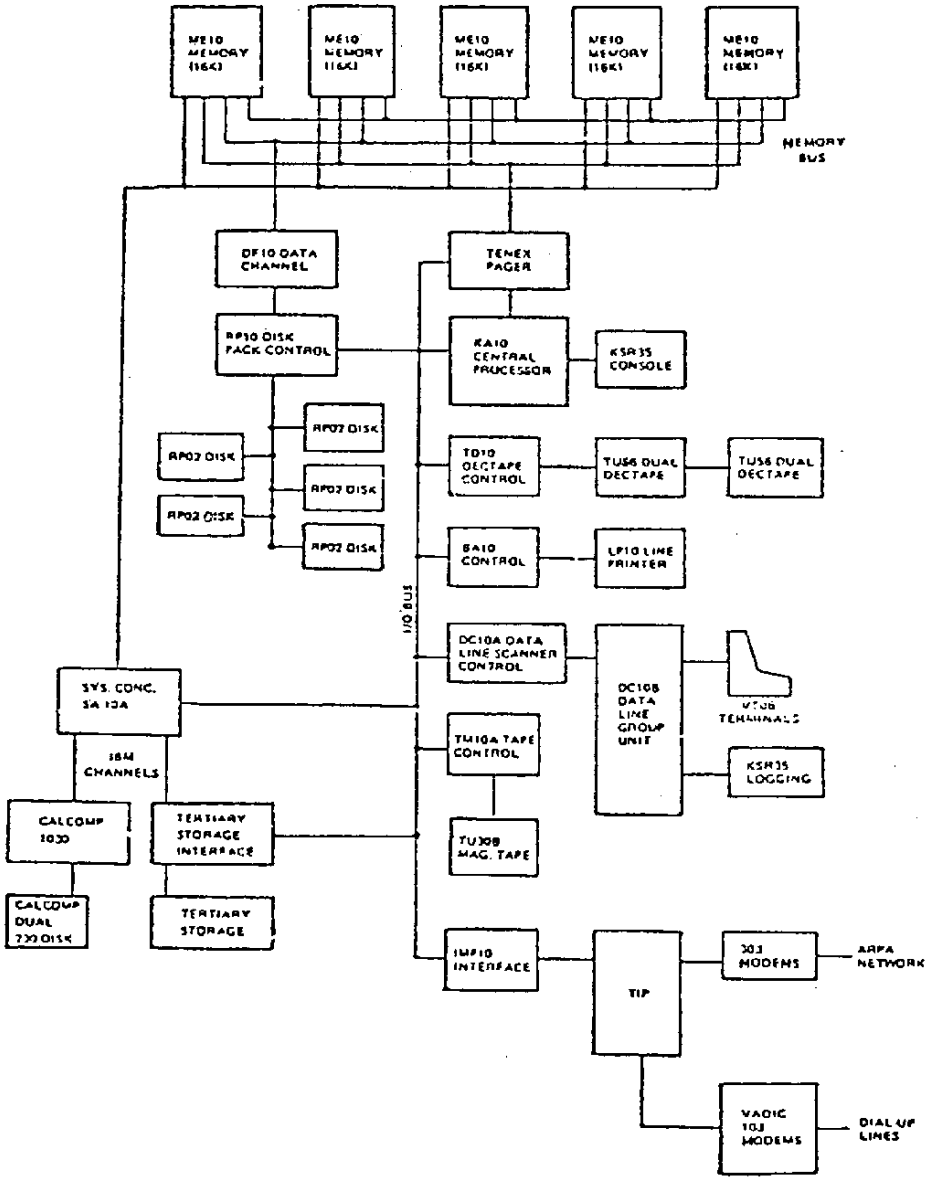

ME10 MEMORY (16K)
ME10 MEMORY (16K)
ME10 MEMORY (16K)
ME10 MEMORY (16K)
ME10 MEMORY (16K)
MEMORY BUS
DF10 DATA CHANNEL
TENEX PAGER
RP10 DISK PACK CONTROL
KA10 CENTRAL PROCESSOR
KSR35 CONSOLE
RP02 DISK
RP02 DISK
RP02 DISK
RP02 DISK
RP02 DISK
TD10 DECTAPE CONTROL
TU56 DUAL DECTAPE
TU56 DUAL DECTAPE
MA10 CONTROL
LP10 LINE PRINTER
I/O BUS
DC10A DATA LINE SCANNER CONTROL
DC10B DATA LINE GROUP UNIT
VT05B TERMINALS
SYS. CONC. SA 10A
TU10A TAPE CONTROL
KSR35 LOGGING
IBM CHANNELS
CALCOMP 1030
TERTIARY STORAGE INTERFACE
TU30B MAG. TAPE
CALCOMP DUAL 230 DISK
TERTIARY STORAGE
IMP10 INTERFACE
TIP
303 MODEMS
ARPA NETWORK
VADIC 103 MODEMS
DIAL UP LINES

図 3.4.3 1

3.4.10 シストリックアレイDBM　30）

本マシンはカーネギーメロン大学のH.T.Kungらによって提案されているシストリックアレイを用いた関係データベースマシンであり，図3.4.32に示されるシステム構成をとる。

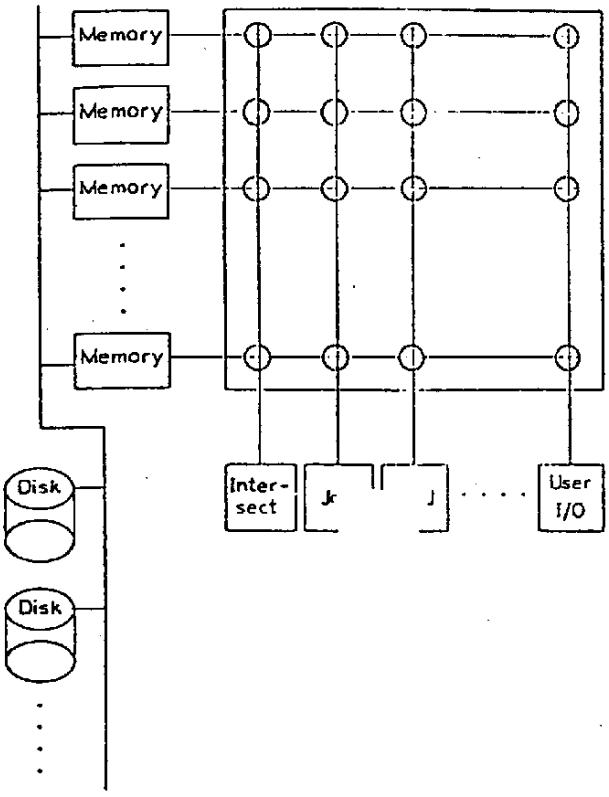


図3.4.32

A．タプル比較用アレイ

図3.4.33のごときセルを一次元状に配列し，図3.4.34のごとく各タプルの要素をずらせて入力することによって，パイプライン的に比較することができる。ここでリレーションAのi番目のタプルをa_iとし $a_i = <a_{i1}, \cdots\cdots, a_{im}>$ と表わすことにする。

さらに図3.4.35，36のごとく2次元アレイにすることにより，多くのタプル同志の比較をパイプライン的に実行することが可能となる。

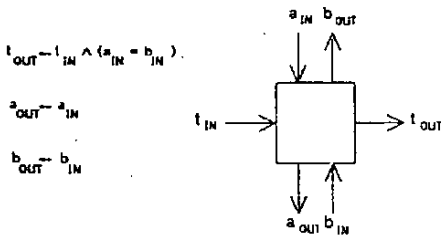


図3.4.3 3 Individual comparison processor.

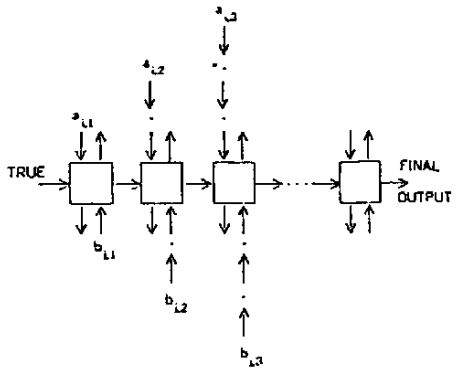


図3.4.3 4 Tuple comparison array.

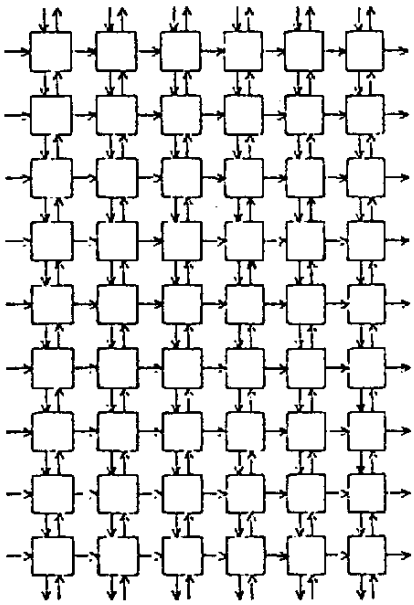
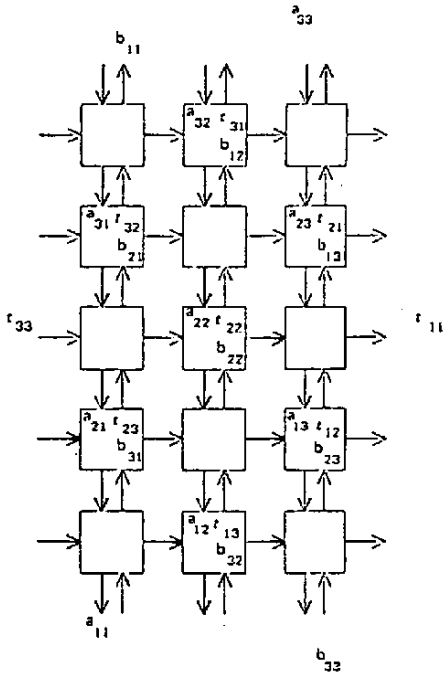


図3.4.3 5
Two-dimensional (orthogonal) comparison array.



$$t_{ij} = \begin{cases} TRUE & \text{if } t_{ij}^{initial} = TRUE, \text{ and } a_{ik} = b_{jk} \text{ for all } 1 \le k \le m, \\ FALSE & \text{otherwise.} \end{cases}$$

図3.4.3 6
Data moving through the comparison array.

B．集合積アレイ

先の２次元比較アレイの右端に１次元の蓄積アレイを設けることにより２つのリレーションの集合積をとることが可能となる。（図3.4.37） 蓄積アレイでは一番上からの入力を常に０とおき，左からの t_{ij} と上からの t_j とのORをとって下へ送ることにすると，一番下では $t_i = OR_{1\leq j\leq n}\ t_{ij}$ となり，リレーションAのｉ番目のタプルがリレーションBに含まれているかどうかが判ることになる。

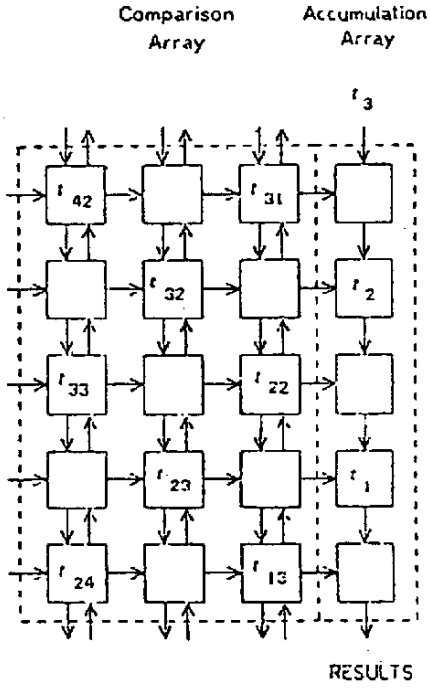


図3.4.37 Intersection array, consisting of two modules:
(2-dim) comparison array on the left, and
(1-dim) accumulation array on the right

C．重複除去アレイ

集合積アレイにおいて，同一のリレーションを上下から流すことによって実現される。リレーションの各タプル同志の比較結果はマトリックス t_{ij} と表わされるが，この下三角（対角成分を除く）に対して，各行のORをとり，もし１になればそのタプルを重複タプルとみなすことができる。従って，左端からの t_{ij} 入力を下三角に対してのみ1として集合積アレイを用いることによって，同一タプルのうち最も番号の小さいタプルを残し，他を取り除くことができる。

D. ジョイン用アレイ

ジョインでは，２リレーションの全てのタプル間で，ジョインアトリビュートに関して比較を行い，マッチしたものに対して結合操作を施す。この場合には，比較結果のマトリックスが生成されればよく，図3.4.38のごとく単なる比較アレイで充分である。

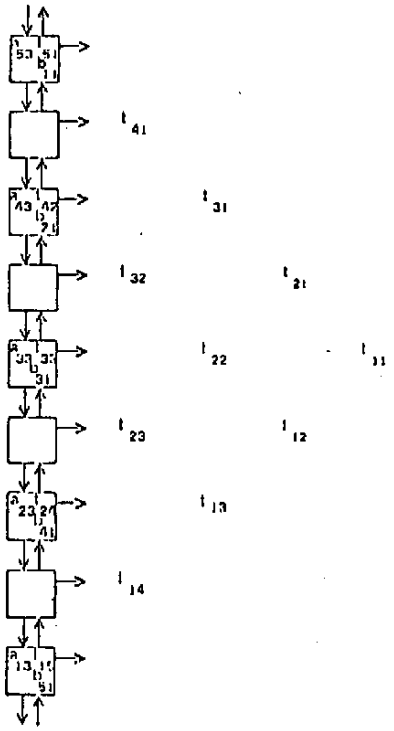


図3.4.38

E. 除算アレイ

図3.4.39に示されるごとく，除算アレイは除アレイと被除アレイから成る。除アレイには，各行毎に除リレーション全体を格納し，また被除アレイの左列には被除リレーションの商アトリビュートを重複を除いて貯えておく。この後，被除リレーションを流し，商アトリビュートの比較を行う。マッチすると右へ信号を送り，除リレーション内でさらにマッチをとる。除アレイの各行において，全てマッチすると当該タプルは商とみなされる。

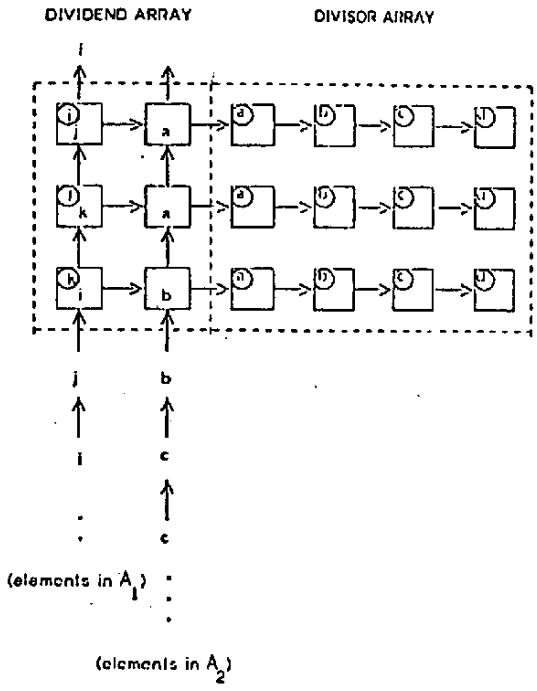


図3.4.39

F．実装と諸問題

1ビットの比較器の専有面積を240μm×150μmとし，チップを6㎜×6㎜とすると,1000ビットの比較器を1チップに集積することが可能となり，これを1000チップ用いると百万ビットの比較を並列に行えることになる。今，1タブルを1,500ビット，1リレーションを10^4タブルとし，さらに1ビットの比較に350nsecかかると仮定すると，2つのリレーションの集合積は，約50msecで行うことができ，実用規模のマシンは比較的容易に実現可能としている。（なお，1本のピンには10bitをマルチプレクスすると仮定）

また，データベースマシンへの適用に関しては，アレイに適した大きさに問題を分割することが課題として載げられている。

3.4.1 1　DBC　ジョインプロセッサ　40), 41)

本プロセッサはＤＢＣにおいてジョインの実現のみを目的とし，ＶＬＳＩ化を指向した専用プロセッサである。

A．アーキテクチャ

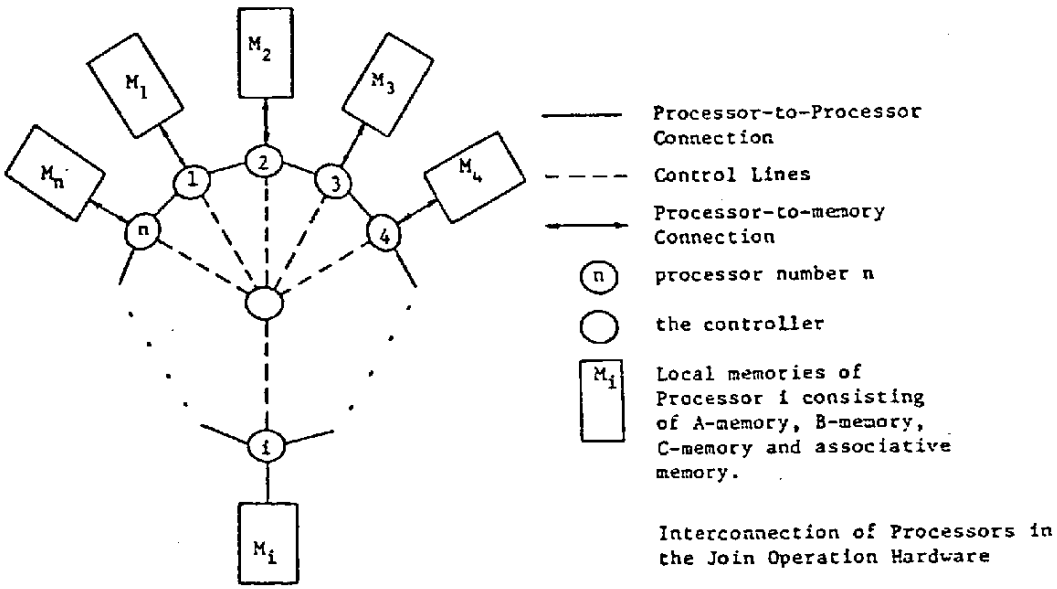


図3.4.4 0

図3.4.4 0に示すごとく，同一のプロセッサとそのローカルメモリを多数リング状に結合したアーキテクチャとなっており，各プロセッサは前段のプロセッサとの結合に加え，一台のコントローラに接続されている。ローカルメモリは４つのメモリ，Ａメモリ，Ｂメモリ，Ｃメモリ，および連想メモリからできており，Ａメモリはソースリレーション用，Ｂメモリはデータベース外部記憶と処理速度の差を緩衝することを目的としたバッファメモリ，Ｃメモリはジョインの結果を格納しておくためのメモリ，連想メモリはジョイン操作での比較演算を高速化するためのものである。

B．処理方式

ジョインはまずソースリレーションを各プロセッサのローカルメモリに並列にローディングし，その後，ターゲットリレーションを読み込みながらマッチングをとるという手法で行われる。ソースリレーションは，ジョインプロセッサのローカルメモリに入り切るという仮定をしており，基本的にはフルスキャン方式の並列化である。より詳しく述べると，ジョイン処理は以下の一連の段階から構成される。

a．ソース入力フェイズ

ソースリレーションのレコードがｎ台の全プロセッサによって並列にＢメモリに読み込まれる。全レコードをＣｓとすると各プロセッサはＣｓ／ｎレコードを入力することになる。

(1) Ａブロック計算フェイズ

Ｂメモリにソースタプルが読み込まれ始めると本フェイズが始まる。各プロセッサはＢメモリからタプルを取り出してはジョインアトリビュートを切り出し，連想メモリにもしそのエントリがなければ格納する。

(2) ソース記憶フェイズ

その後，ジョインアトリビュートに対しハッシュを施しＡメモリの適当なブロックに当該レコードを記憶する。オーバフローが生じた場合にはオーバフローエリアに入れられポインタでリンクされてゆく。

b．ターゲット入力フェイズ

上述のフェイズが終了した時点で，ソースリレーションは全プロセッサのＡメモリ内における適当なブロックに格納されていることになる。この後，本フェイズでターゲットリレーション処理が開始される。

(1) ジョインフェイズ

各プロセッサはターゲットタプルが読み込まれ始めるとＢメモリから１レコードずつ取り込み，連想メモリをチェックする。もしマッチすると対応するＡメモリのブロックを読みジョインを行う。

(2) ターゲット伝搬フェイズ

ターゲットレコードは当該プロセッサで処理を終えると他の全てのプロセッサへ順に送られる。プロセッサは前段のプロセッサから送られてきたレコードに対しても同様のジョイン処理を行う。全てのプロセッサによって処理されたターゲットレコードは捨てられる。

C．処理時間

表3.4.2に示されるパラメータを用いると各フェイズでの処理時間は次のように求まる。

ソース入力およびＡブロック計算フェイズ

$$(Zbs+Zam+Za)Cs/n$$

ターゲット入力およびターゲット伝搬フェイズ

$$Ct(Zbs+Zam)$$

ジョインフェイズ

$$ZaCr/(ne)+(Cr/n)u$$

従って全時間はこれらの和となり

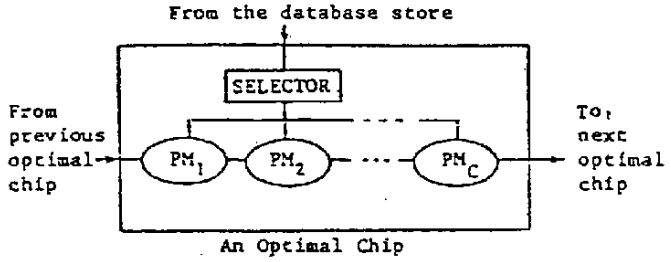
O (Cs／n＋Ct＋Cr／n) を得る。

表3.4.2

C_s	: Cardinality of the source relation.
C_t	: Cardinality of the target relation.
C_r	: Cardinality of the result relation.
Z_a	: Average time to access and read (write) a record from (to) the A-memory.
Z_{bs}	: Time taken to read (write) next record from (into) the B-memory.
Z_{am}	: Time taken to probe an am.
N	: Number of blocks in the A-memory.
e	: Efficiency of the hashing function [e.g., e=1, if the hashing scheme is perfect and there are no collisions].
n	: Number of processors in the join operation hardware.
a	: Fraction of target records that participate in the join operation.
u	: Average time to output a result record.

D. ＶＬＳＩによる実装

ジョインプロセッサの実装に当ってはＡメモリは比較的低速なＣＣＤメモリを，またＢメモリは高速ＲＡＭを仮定している。１レコード５００バイト，ソースリレーションタプル数１０万という環境を設定し，さらに種々のパラメータを仮定することにより，待ち行列解析を用いてＡおよびＢメモリの適切な容量を算出できる。さらに，この結果にもとづき，現在および十年後でのＬＳＩ化を考え，最適チップ構成を求められる。（図3.4.41） 現在の集積度では，Ａメモリは１００台のプロセッサを用いたとしても一台当たり４００Kbit程度必要となり，チップ内には入れることはできない。そこで，プロセッサとＢメモリをＬＳＩ化することとし，チップ当たり70Ｋトランジスタ，プロセッサ１台を５Ｋトランジスタと仮定すると７台のプロセッサと35KbitのＢメモリを１チップ化することが最適となる。また十年後には，11×10^6トランジスタ程度の集積度が得られるとすると，Ａメモリもチップ上に載せることができる。この時点では４２ＭbitのＡメモリ，３５０KbitのＢメモリ，そして３０台のプロセッサを１チップ化することが最適である。



PM: Processor-memory pair
C: Number of PM pairs in an optimal chip
OC: Optimal Chip

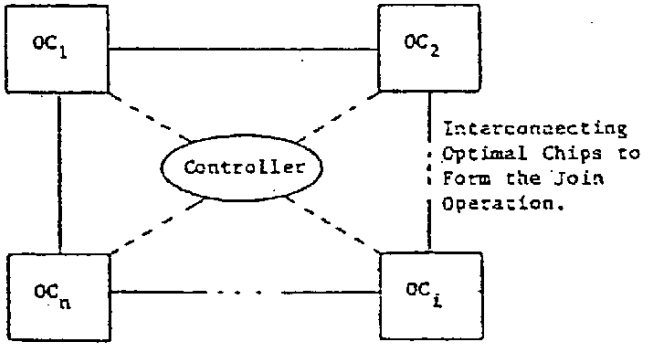


図3.4.4 1 Inter- and Intra-Connections of Optimal Chips

3.4.1 2 S R M 42)

ＳＲＭ(Single Relation Module)はプリンストン大学で提案されたデータベースマシンの基本構成要素である。マシンの全体構成は図3.4.4 2に示すように，多数のＳＲＭとジョインプロセッサをネットワーク結合したものが想定されている。

A. 内部構成

ＳＲＭは将来のＶＬＳＩ技術の向上を前提として１チップ化が仮定されている。その内部構成を図3.4.4 3に示す。１つのＳＲＭにつき１リレーションが割り合てられ，１リレーションのみをオペランドとする関係代数演算は，全て１台のＳＲＭで処理する。タプルは横方向に格納され，各ＶＰ(Value Processor)につき１アトリビュートが割り合てられる。リレーションがＳＲＭに収まりきらない場合には，ＳＲＭを横方向，縦方向に連結して用いる。

B. 処理方式

ＳＲＭを用いた関係代数演算の処理の例を以下に示す。

セレクションは，その条件中，連言形のものが並行に処理され，選言の数だけこれが繰り返される。

連言形の処理は，ＲＰ（Relation Processor）が先づＨＴＢ（Header Tuple Bus）を通じて条件アトリビュートに対応するＣＰ（Column Processor）に論理式を評価するコマンドを，またＨＣＢ（Header Column Bus）を通じて出力アトリビュートをＴＰ（Tuple Processor）に各々ブロードキャストする。続いて，ＣＰは受け取ったコマンドを同一カラムに属するＶＰにＣＢ（Column Bus）を通じてブロードキャストする。ＶＰはこのコマンドを実行し，結果をＴＰに返す。ＴＰが結果の論理積をとることによって連言形のセレクション処理が完了する。結果はＲＰを通して外部に読み出される。

ソーティングは，3.4.13におけるイクスチェンジソートを隣接したＶＰ間を結ぶバスを用いて行う。表3.4.3に，いくつかの演算について，シングルプロセッサの性能値との比較を示す。

C．評　　価

提案されたＳＲＭを１チップで実現するには，例えばアトリビュート数10，タプル数100のリレーションを仮定すると，1,000万トランジスタ／チップの集積度が必要とされている。これは，少なくとも現時点では実現不可能であり，既存のマイクロプロセッサ等を用いて負荷の重いＲＰを分離したり，図3.4.44に示すような妥協案を採用するのが現実的である。

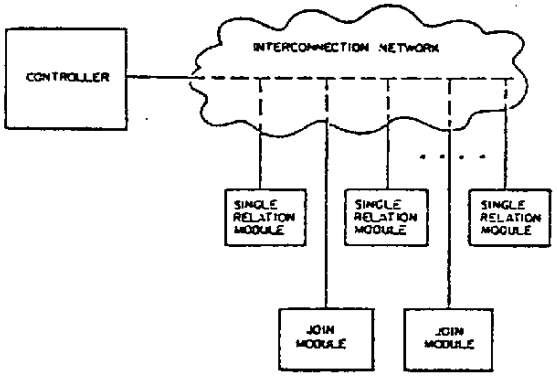


図3.4.42

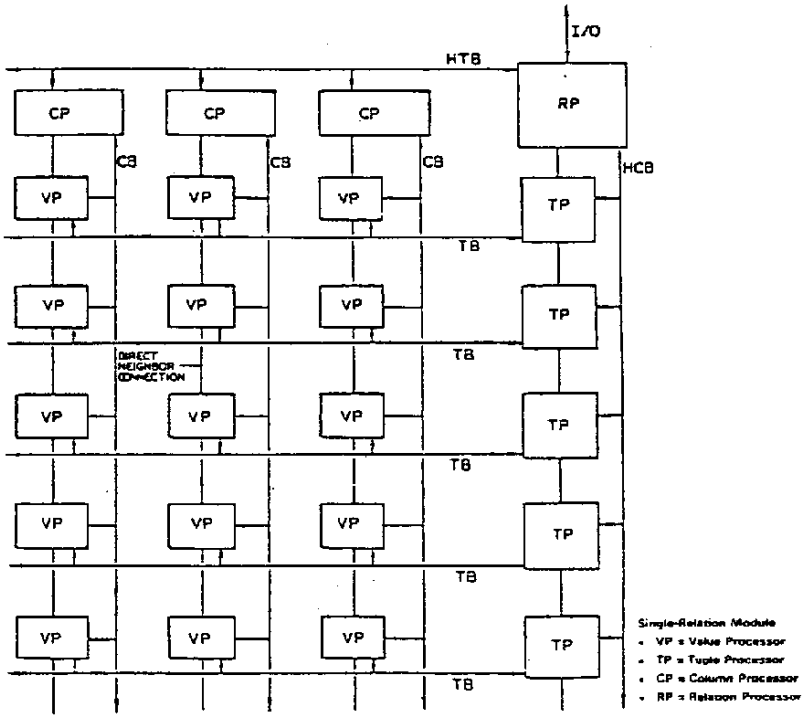


図 3. 4. 4 3

表 3. 4. 3

Operation	Sequential Computer	SRM
selection	n	1
projection	n log n	1
read out	n	n
sort	n log n	n
insert/delete	implementation-dependent	1
update	〃	1

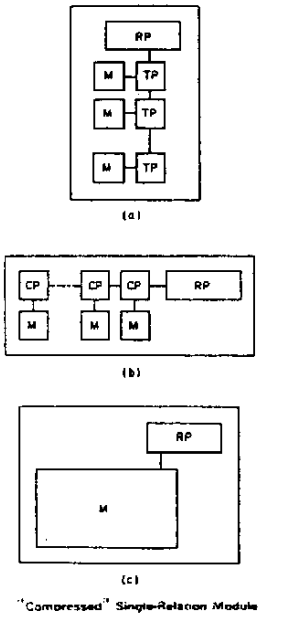


図 3. 4. 4 4

3.4.13 セルラアレイDBM 43)

ポーランドのM.Muraszkiewiczはイクスチェンジ・ソートを基本としたオーダnのソート，プロジェクションを行うセルラアレイを提案した。

A. ソートアルゴリズム

ソートのアルゴリズムは並列化されたイクスチェンジ・ソートを基本としている。ソートすべきレコード列 r_1, r_2 ……, r_n に対し，先ず r_{2i-1}, r_{2i} （ i＝1, 2, ……, 〔n／2〕）を一勢に比較し，$r_{2i-1} > r_{2i}$なら値を交換する（EPH：Even PHase）。次に r_{2i}, r_{2i+1} （ i＝1, 2, ……, 〔n／2〕）に対して同様な操作を行う（OPH：Odd PHase）。以下，この2つの動作を総計n回繰り返せばソートが完了する（図3.4.45）。

TOM	JIM	BIL	LUC	
	1		0	EPH
JIM	TOM	BIL	LUC	
		1		OPH
JIM	BIL	TOM	LUC	
	1		1	EPH
BIL	JIM	LUC	TOM	
		0		OPH
BIL	JIM	LUC	TOM	result

Example of sorting.

図3.4.45

B. ソーティングアレイ

上記のアルゴリズムを実現するセルラアレイの構成を図3.4.46に示す。各レコードはアレイ

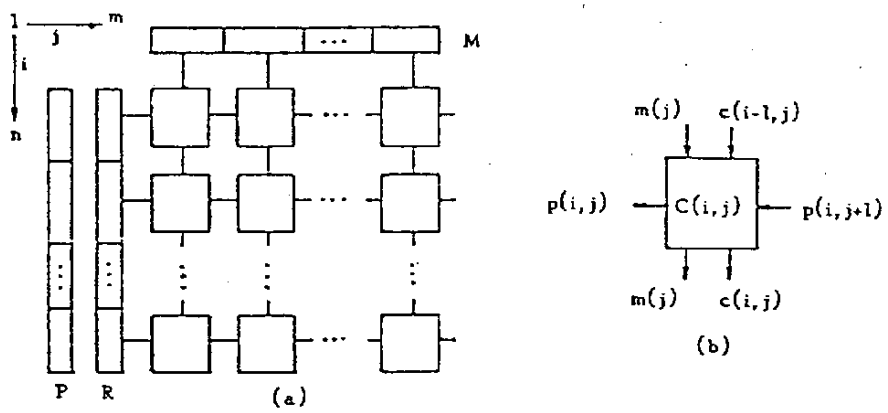


図3.4.46

の行方向に格納される。ソートに関与しないフィールドに対応するセルはマスクレジスタMによってマスクされる。フェイズレジスタPは，比較を行う隣り合った上下2つの行の下の方の行に対し1を，上の方の行に対して0を保持し，このパターンの変更によってEPH，およびOPH 2つのフェイズを指示する。アレイ中のセルを図に示す行，列方向に関しC（ i， j ）と表現し，その内容をc（ i， j ）と書く。図中，m（j）はMのマスク値，p（ i， j ）はセルの比較結果の出力信号で

$$p(i,j)=\begin{cases}1, & \text{if } m(j)=1 \wedge c(i-1,j) > c(i,j)\\ 0, & \text{if } m(j)=1 \wedge c(i-1,j) < c(i,j)\\ p(i,j+1), & \text{if } m(j)=0 \vee c(i-1,j)=c(i,j)\end{cases}$$

$$p(i,m+1)=0.$$

と定められる。ソートは，先ず各行毎に右から左へとセルを走査し，各々1つ上にあるセルとの大小比較を行う。比較結果は上記のp（ i， j ）を用いて左隣のセルに渡される。レコードの大小比較の結果はp（ i， 1 ）の値として得られ，これが結果レジスタRに格納される。従ってR（ i ）・P（ i ）＝1なるiに対するペア r_{i-1} ， r_i を交換すればよい。以上の操作をPを用いてフェイズの操作を行いつつ，n回繰り返せばソートが終了する。

C．プロジェクション

重複除去を伴うプロジェクションは，ソートと同様処理負荷の重い演算であるが，上記のソーティングアレイを用いて容易に実行することができる。すなわち，先ずレコードを上記の操作によってプロジェクション属性以外の属性にMを用いてマスクをかけ，一度ソートする。次にアレイに以下の処理および信号q（ i， j ）を仮定する。

$$q(i,j)=\begin{cases}0, & \text{if } q(i,j+1)=0 \wedge c(i-1,j)=c(i,j) \wedge m(j)=1\\ 1, & \text{if } q(i,j+1)=1 \vee c(i-1,j) \neq c(i,j) \wedge m(j)=1\\ q(i,j+1), & \text{if } m(j)=0\end{cases}$$

$$q(i,m+1)=0$$

ソートに続いてこの一致操作を施すことにより，q（ i， 1 ）の値として r_i と r_{i-1} の一致が示される。ソートと同様，この値はRに残され，あとはR（ i ）＝1のレコードのみを読み出せばプロジェクション処理が完了する。図3.4.47にプロジェクション処理の一例を示す。

S#	SN	SL	SA
1	JIM	WAW	Y
2	TOM	PAR	Y
3	IAN	NYO	N
4	JIM	LON	Y

(a)

M | 0 1 1 1 0 0 0 1 |

R

1	3 I A N N Y O N
1	1 J I M W A W Y
0	4 J I M L O N Y
1	2 T O M P A R Y

(b)

Example of projection operation in the SCA unit. (a) Relation B= (S#, SN, SL, SA). (b) State of the array after execution of SORT operation, and the state of R register after execution of MATCH operation.

図3.4.47

3.4.14 DIRECT 14)～16)

本マシンはウイスコンシン大のD.J.DE WITTによって提案されているＭＩＭＤ型のマシンであり，図3.4.48に示されるシステム構成をとる。

A．アーキテクチャ

ＤＩＲＥＣＴは以下の構成要素から成る。

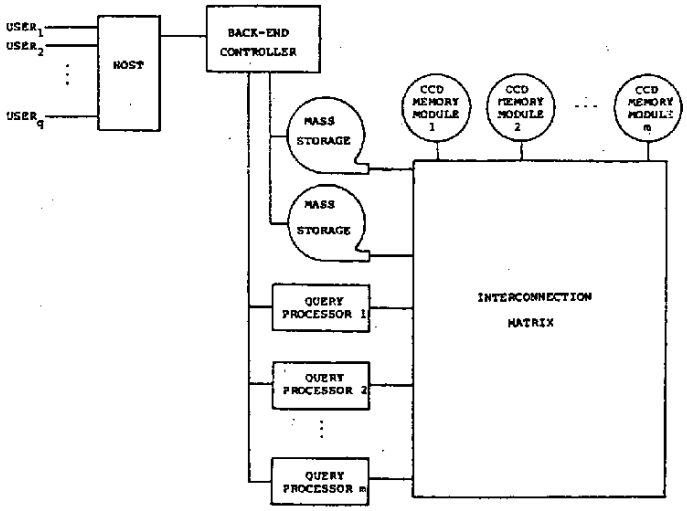
a．バックエンドコントローラ……ＰＤＰ11／40を用い，プロセッサ群の制御，およびメモリ管理を司る。ホストとはＤＭＡによって結合されている。

b．問い合わせプロセッサ……ＰＤＰ11／03を用い，コントローラによって割り当てられたページに対して，関係代数演算処理を行う。

c．メモリモジュール……16KBのＣＣＤメモリからなり，ディスクからのステージングバッファとして利用される。

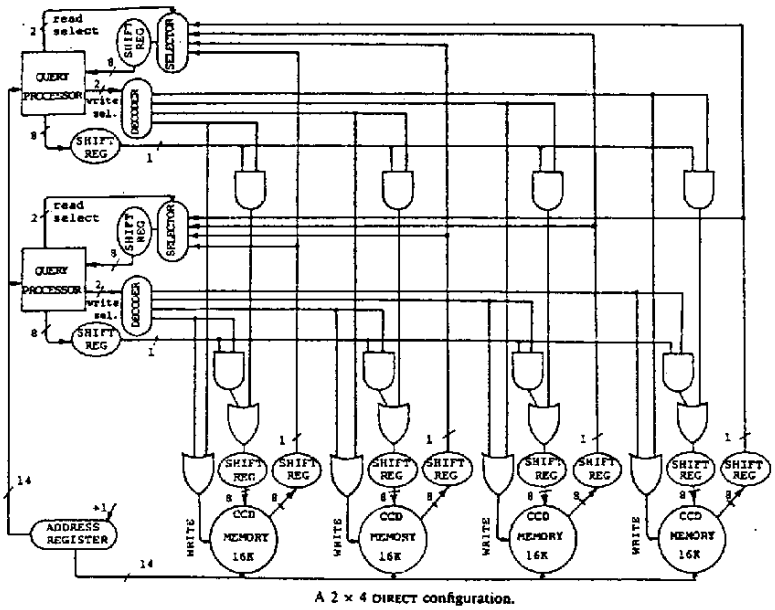
d．相互結合マトリクス……問い合わせプロセッサとメモリモジュール間をマトリックススイッチで結合する。結合ラインは１ビット幅である。

問い合わせプロセッサ2台，メモリモジュール4台のシステムを図3.4.4 9に示す。



DIRECT system architecture.

図3.4.4 8

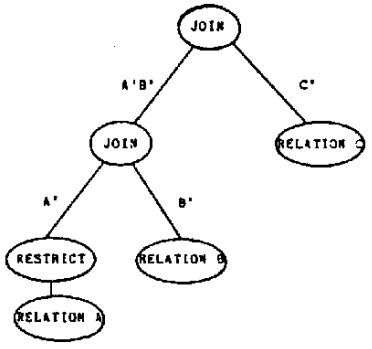


A 2 × 4 DIRECT configuration.

図3.4.4 9

B．処理方式

タプルは固定長とし，リレーションは固定長ページに分割して管理される。処理もページ単位で管理され，プロセッサには処理すべきページが動的に割り当てられる。例えば図3.4.50に示される関係代数木は，図3.4.51のごとき問い合わせパケットとしてプロセッサに渡される。プロセッサがバックエンドコントローラに対し処理すべきページを要求すると，コントローラは各プロセッサに異なるページを割り付け，並列に処理させる。このようなＭＩＭＤ制御により，高い性能が期待される。中間結果は，ＲＡＰやＣＡＳＳＭのごとくマークビットで表わすことはせず，テンポラリリレーションをＣＣＤメモリ上に動的に生成する。これによりベースリレーションをロックする必要はなく，問い合わせ間での並列度を向上させることができる。



Example query tree.

図 3.4.50

```
CREATE C
DO FOREVER
BEGIN
      - ASK BACK-END CONTROLLER (BEC) FOR THE NEXT_PAGE OF RELATION A
      - WAIT FOR BEC TO RETURN A PAGE FRAME NUMBER (PF#)
      - IF BEC RETURNS "END OF RELATION" (EOR) PROCEED TO NEXT
      OPERATION IN QUERY PACKET
        OTHERWISE /*JOIN THIS PAGE OF A WITH EVERY PAGE IN B */
            - READ NEXT PAGE OF A FROM PF#
            - SET I EQUAL TO 1
            - SET END_OF_B TO FALSE
            WHILE (END_OF_B = FALSE)
            BEGIN
                  - GET_PAGE I OF RELATION B FROM BEC
                  - WAIT FOR BEC TO RETURN  PF#
                  - IF BEC RETURNS "EOR" SET END_OF_B = TRUE
                    OTHERWISE
                        - READ PAGE I OF B FROM PF#
                        - JOIN CURRENT PAGE OF A WITH PAGE I OF B
                        - WRITE RESULTING TUPLES INTO A BUFFER
                        - WHEN THE BUFFER IS FULL
                              - ASK BEC FOR NEXT_PAGE OF C
                              - WAIT FOR PF#  FROM BEC
                              - WRITE OUTPUT BUFFER ONTO PF#
                        - INCREMENT I
                  END
END
```

図 3.4.51

C．ジョイン

DIRECTでは図3.4.5 2に示されるごとく，ジョイン処理にはソートマージよりネストループアルゴリズムの方が適していることがわかる。ページレベルのネストループ方式では，大きい方のリレーションの各ページをプロセッサに割り当て他方のリレーションをフルスキャンすることによってジョイン処理がなされる。

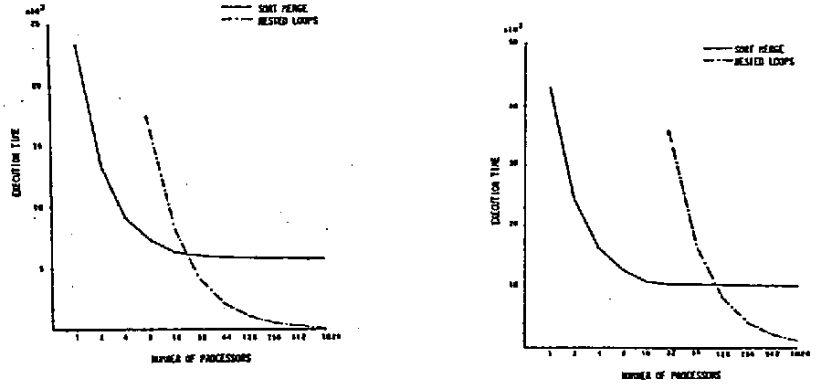


図3.4.5 2

D．コンプレスオペレーション

ページ単位の記憶管理を行っているとはいえ，複数台のプロセッサによって処理されるため，一般に結果リレーションは断片化されることになる。このため，コンプレスというオペレータが用意されている。ジョインの処理負荷はページ数の積で効くため，コンプレスする方が好ましいが，一方コンプレスの度合を良くするためには少ないプロセッサによって処理せねばならず多大の時間が要される。例えば図3.4.5 3の問い合わせ木に対して，コンプレスするプロセッサ台数を横軸に，全処理時間を縦軸にとると図3.4.5 4のごとく表わされ，コンプレスに用いるべき最適プロセッサ数が求まる。

E．性　　能

ダイレクトでは図3.4.5 5に示されるごとき種々の問い合わせクラスに対して4つのプロセッサ制御手法を用いてマシンの性能評価を行っている。最も負荷の重いクラスⅣに対するパフォーマンスを図3.4.5 6に示す。

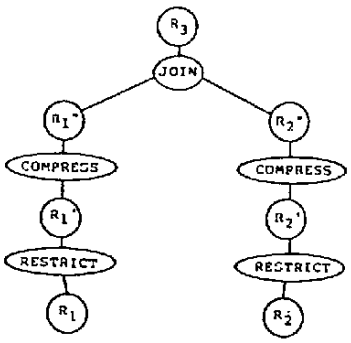


図 3. 4. 5 3

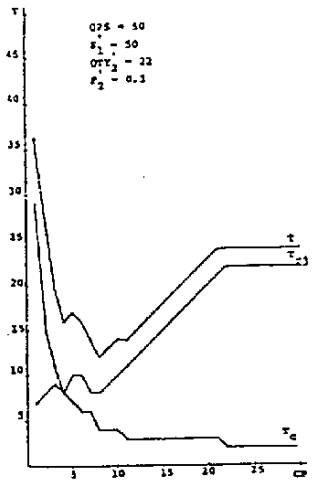


図 3. 4. 5 4

Test name	Number of queries	Description	Number of source pages read by test
Class I	5	Each with 1 restrict only	183
Class II	5	Each with 1 join and 2 restricts	250
Class III	5	2 Queries: 2 joins and 3 restricts 3 Queries: 3 joins and 4 restricts	393
Class IV	5	2 Queries: 4 joins and 5 restricts 1 Query: 5 joins and 6 restricts 1 Query: 6 joins and 7 restricts 1 Query: 7 joins and 8 restricts	529
Mix I	10	2 Queries from Class I 3 Queries from Class II 3 Queries from Class III 2 Queries from Class IV	624
Mix II	10	1 Query from Class I 4 Queries from Class II 4 Queries from Class III 1 Query from Class IV	544

図 3. 4. 5 5

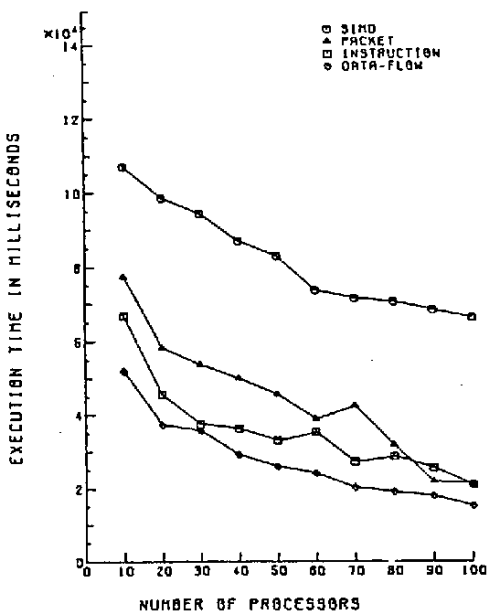


図 3. 4. 5 6

F. 問 題 点

Eでの評価では，バックエンドコントローラにおけるオーバヘッドは含まれていない。しかし，多数のプロセッサを動的に制御するためにはコントローラの負荷はきわめて高くなり，実行時間よりもオーバヘッドの方が大きくなることもある。この点に関しては，今後の課題とされている。

3.4.15 Data Flow DIRECT 44)

本マシンはプロセッサの割り付けをQuery Packet LevelからAlgebra Operator Levelに落とし，プロセッサの利用効率を上げるとともに，問い合わせ木における縦のパイプライン効果によるページトラヒックの減少を目的としたＤＩＲＥＣＴの改良版であり，データフロー制御による問い合わせ処理を行う。

A. アーキテクチャ

図3.4.57に示されるごとくデータフローダイレクトは以下の構成要素からなる。

a. ＭＣ（マスタコントローラ） ホストインタフェースを提供するとともにＩＣへのオペレータ割付を行う。

b. ＩＣ（インストラクションコントローラ） 関係代数演算の実行管理を行う。

c. Inner Ring ＭＣとＩＣの通信用バス

d. Mass Storage System ２次記憶系としてのディスクとマルチポートディスクキャッシュからなる。

e. ＩＰ（インストラクションプロセッサ） ＩＣによって割り付けられた命令パケットを実行する。

f. Outer Ring ＩＰとＩＣの通信用バス

Inner RingはＭＣ，ＩＣ間で制御メッセージのやりとりに用いられ，１Mbit／sec程度のバンド幅で充分であるのに対し，Outer Ringでは処理対象のリレーションあるいは結果リレーションがＩＣ，ＩＰ間で授受されることになり高いバンド幅が要求される（図3.4.58）。

B. 処理方式

ＭＣはＩＣに問い合わせ木の一部を割り付け，制御を複数台のＩＣに分散させる。ＩＣはＭＣによりオペレータが割り当てられると実行管理を開始する。オペランドがベースリレーションの時は発火可能であり，ディスクからローカルメモリに必要なページをコピーして命令パケットを作る。図3.4.59に示されるこのパケットはOuter Ringを通じてＩＰに割り付けられる。一方，オペランドが中間レリーションの場合にはＩＣは結果パケットがＩＰにより送られてくるのを待ち，到着するとコンプレスして当該オペレーションに関する命令パケットを生成する。ペー

ジレベルのデータフロー制御を行い，１ページでも発火可能となるとＩＰに割り付けられる。ＩＰは当該パケットの処理が完了するとＩＣに図3.4.60の制御パケットを用いて通知するとともに，図3.4.61に示される結果パケットを生成して他のＩＣへ送出する。ＩＣはＭＣに与えられた処理を完了するとＩＰを解放する。以上のごとく，ページレベルのデータフロー制御が複数のＩＣによって実現されていることがわかる。

また，ジョイン処理ではOuter Ringのトラヒックを低下させるため，パケットのブロードキャストを行っている。

C．問　題　点

ここでは同時実行制御はＭＣで行われているが，この制御もＩＣに移し，分散型のデータベースマシンを構築することも可能であり，今後の課題としている。なお，アーキテクチャの詳細については充分に検討されておらず，不明な点が多い。

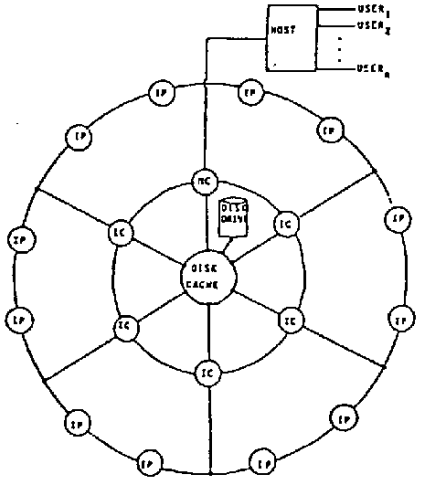


図3.4.57　A Data-flow Database Machine Configuration

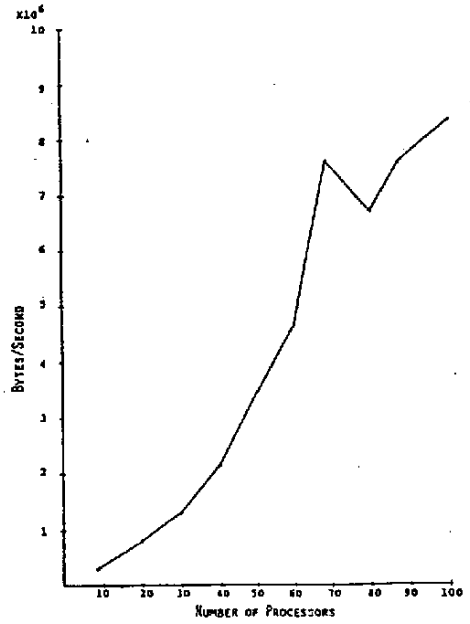


図3.4.5 8　Bandwidth Requirements of DIRECT

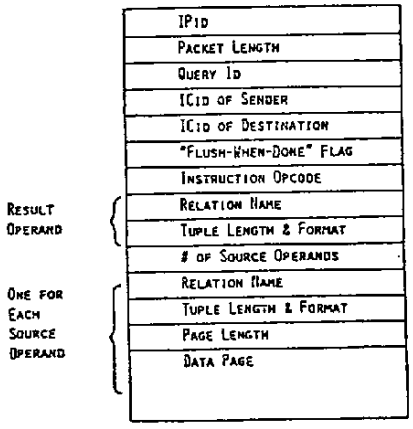


図3.4.5 9　Instruction Packet Format

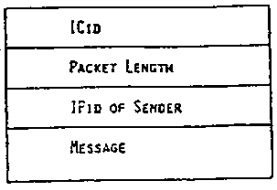


図3.4.6 0　Control Packet Format

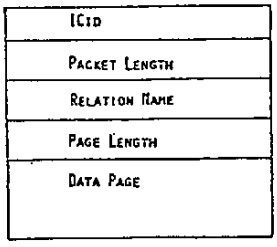


図3.4.6 1　Result Packet Format

3.4.1 6　Tree Machine　45)

本マシンはカーネギーメロン大学のS.W.Songによって提案されているＶＬＳＩ指向の関係データベースマシンであり，図3.4.6 2に示されるごとく，ハードウェアバックエンドとしてとらえられる。

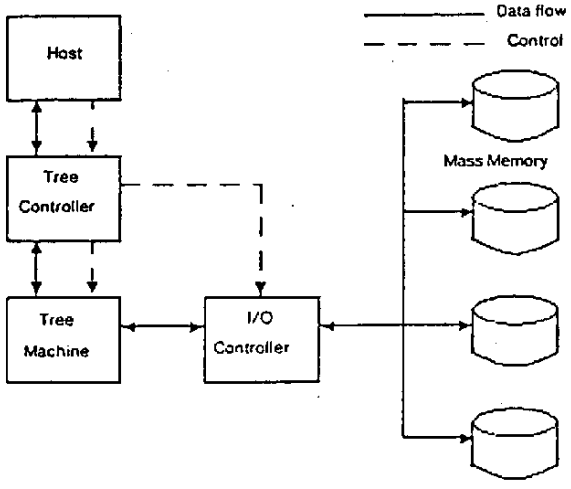


図3.4.6 2　A system configuration.

A．アーキテクチャ

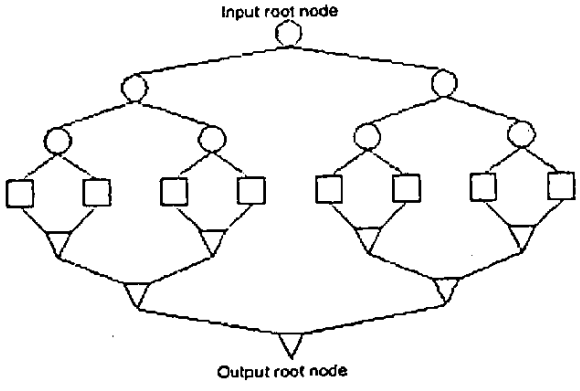


図3.4.6 3　The tree machine.

図3.4.6 3に示されるごとく，3つのノードからなる木型のアーキテクチャとなっている。○ノードは親からの命令を子へブロードキャストし，また□ノードはタプルデータを貯えるととも

に，限られた演算能力を有する。△ノードは□ノードからの演算結果データを出力するためのノードであり，データは上から下へ流れることになる。

B．データ格納方式

1つのタプルが1つの□ノードに貯えられdelete，insertがパイプライン的に実現できる記憶管理方式を採っている。

□node＝(node.freeposition, node.content)と表わされ，当該ノードが空の時node.freepositionには0～N－1までの整数(N：□ノードの数)が，空でない時にはΛが入いると同時にnode.contentにタプルデータが格納される。根ノードは空ノードのリストを$n_{First\ Free}$, $n_{First\ Free+1}$ ……, n_{N-1} の形で管理している。Insert，Deleteに関して根ノードおよび□ノードでの処理は以下の通りである。

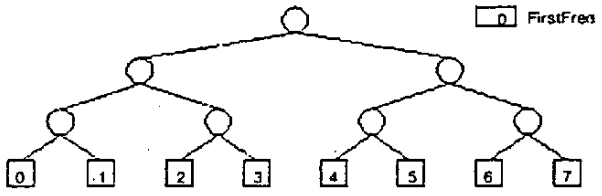
```
instruction.freeposition : = FirstFree;
instruction.content : = X;
FirstFree : = FirstFree + 1


if node.freeposition = instruction.freeposition
 then node.content : = instruction.content;
      node.freeposition : = Λ


FirstFree : = FirstFree - 1;
instruction.freeposition : = FirstFree;
instruction.content : = X


if node.content = instruction.content
 then node.freeposition : = instruction.freeposition;
      node.content : = Λ
```

図3.4.64は初期状態および6レコード挿入後2レコード削除した状態を示す。

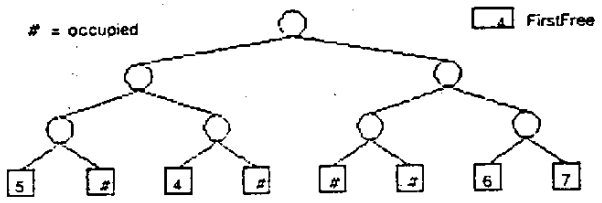
insert，deleteでは○nodeは単に命令を子にブロードキャストするだけでありパイプライン処理が可能であるが，deleteでは削除タプルが1つに確定することを仮定しており，そうでない場合にはキーによって指定しなおす必要がある。

C．出力制御問題

△nodeでは多くの出力が競合することがあり，上から下への伝搬は下のnodeが空いたことを確認した後なされるため，1レベルずつ交互にデータが流れることになる。またある□nodeの出力が遅れると，○nodeがつまりボトルネックになることがある。(図3.4.65)



An empty tree.



After 2 deletions.

図3.4.6 4

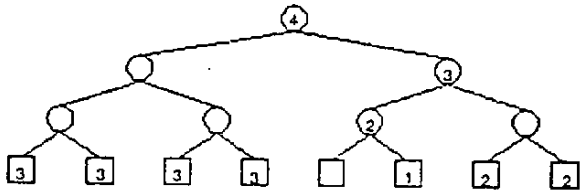


Figure 7: Blocking of flow.

図3.4.6 5

D. ジョイン

Kケずつのタプルからなる2つのリレーションA，Bのexplicit joinは本マシンでは，log N＋2（K＋結果タプル数）時間で実行できる。なお，Tree Machineを制御するTree Controller上にはlog Nケのエントリーをもつ連想メモリを用意し，タプルidをもとに実タプルを取り出して連結できるものとする。

図3.4.6 6に初期状態を示す。各ノードはジョインを終えるまで指定された動作をくり返し行う。○nodeは親から送られてきたものを子へブロードキャストする。また，□nodeはAのタプルを貯えている場合にはAのタプルとそのタプルidを△nodeに送り出し，Bのタプルを貯えている場合には○nodeから送られてくるデータとジョインアトリビュート関して比較を行いマッチするとBのタプルと送られてきたAのタプルidを△nodeへ送出する。△nodeはBり

レーションをもつ□nodeからのデータを優先的に出力する。

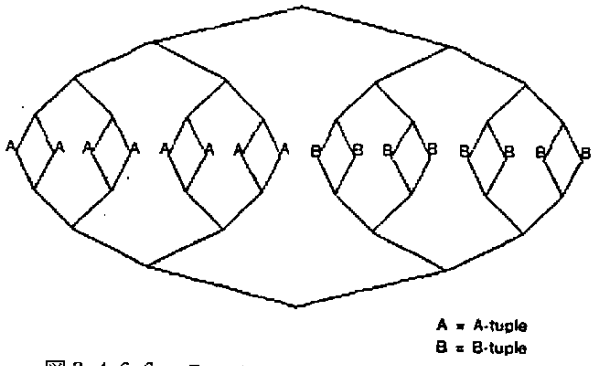


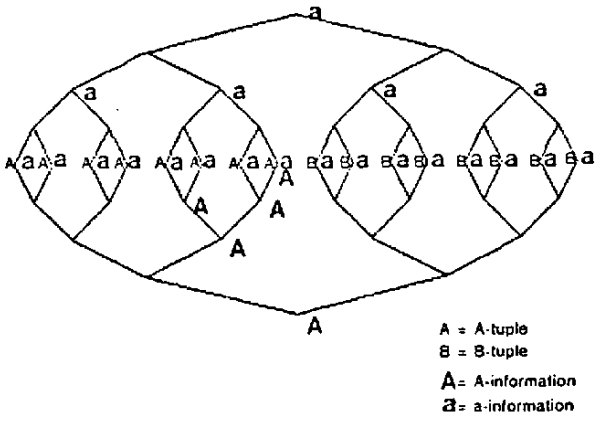
図3.4.6 6　Two relations residing in the tree.

ジョイン処理は以下のごとく進められる。Aをもつ□nodeが出力したタプルがlog N時間後Treeから出力され始めるとControllerは連想メモリに入れると同時に再びTreeに入力する。log N時間後，Bをもつ□nodeにそれが到着するとジョインが行われ結果が△nodeに出力される。△nodeはBデータを優先的に出力し，log N時間後，Treeから出始めると，Controllerは連想メモリを用いて結合処理を行う。以上の動作をくり返すことによってジョインを実現できる。（図3.4.6 7）

Controllerに用意する連想メモリがlog N分だけでよいのは，Aの□nodeから出たタプルが連想メモリに貯えられた後，再び入力されBの□nodeで処理され，Treeから出てくるまでに△nodeでのB優先処理により2×log N時間必要であり，その間データは1つおきに流れるからである。

E．実装および諸問題

1タプル64バイトとすると1シリンダ（500KB）のデータを格納するには現在では約1,000チップ必要であるが，4年後には60チップ程度になるとしている。残された問題として，1.Tree Machineへの高速ローディング　2.Tree Machineへの問題の分割の2つを載げている。この2つの問題は，多くバックエンドマシンのかかえている本質的な課題であり，Treeマシンはこの点に関しては何ら解決を与えるものではない。


A = A-tuple
B = B-tuple
A= A-information
a= a-information

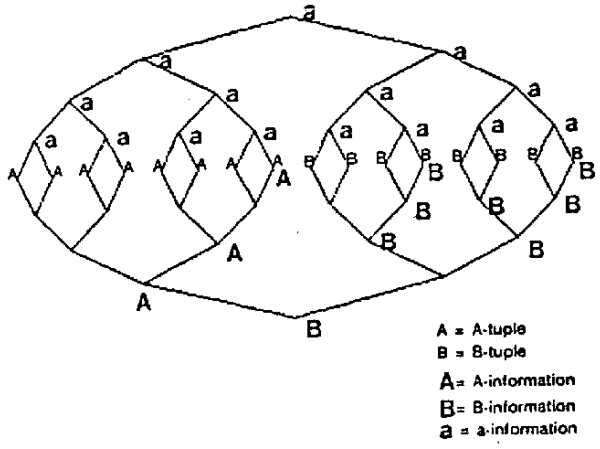

A = A-tuple
B = B-tuple
A= A-information
B= B-information
a = a-information

図 3. 4. 6 7

3.4.1 7　MICRONET　46)

MICRONETは，システム全体の低価格化，高信頼性，拡張性，高性能化を目標とするFlorida大学で開発されたマルチマイクロプロセッサ・リレーショナルデータベースシステムである。システムはコントロールモジュール，ネットワーク・コミニュケーションコントローラ（NCC）各々1台，および多数のデータプロセッシングモジュールをバス結合した構成を採る（図3.4.6 8）。各モジュールには全て同一のマイクロコンピュータシステムを用い，さらにデータプロセッシングモジュール中の各種ソフトウェアも全て同一にし，これらハード，ソフト両面での統一化によって，システムの低価格化，高信頼性を得ている。さらにプロセッサ間の交信は論理的なリレーション名を指定して行われるため，ネットワークへのプロセッサの付加，除外が極めて容易となる。

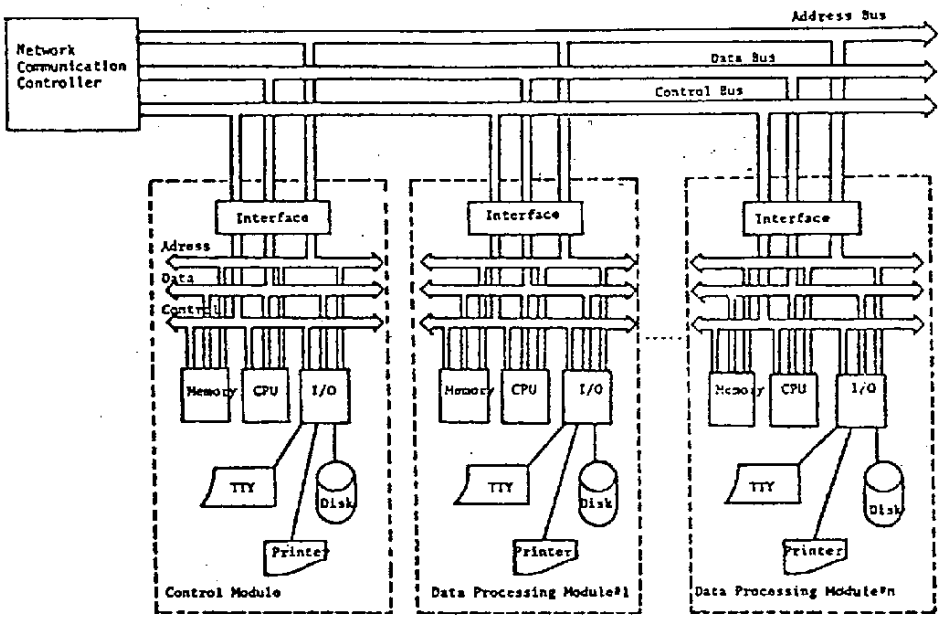


図3.4.6 8　Block diagram of a microcomputer network.

A．アーキテクチャ

ユーザが発行した問い合わせはコントロールモジュールで解析され，データプロセッシングモジュールへのマクロコマンド列に変換される。これらマクロコマンドは，NCCを通じて全てのデータプロセッシングモジュールにブロードキャストされ，同時に実行される。一方，リレーションはセグメントと呼ばれる単位に分割され，データプロセッシングモジュールに分散して配置される。方式上，MICRONETはSIMD型のデータベースマシンととらえられる。ジョイン処理等，データプロセッシングモジュールでの交信が必要な場合には，NCCを通じてメッ

セージのブロードキャストを行う。1つのマクロコマンドの実行が終了すると，コントロールモジュールに割り込みがかけられ，コントロールモジュールは次のマクロコマンドの実行に移る。

B．ファイル構造

コントロールモジュールはＢＲＤ(Base Relation Directory)と呼ばれるディレクトリ，および各リレーションに対するドメイン名，キードメイン名等を保持するインフォメーションブロックを持ち，システム内の全てのベースリレーションを管理する。一方，各データプロセッシングモジュールは，ＢＲＳＤ(Base Relation Segment Directory)を持ち，プロセッサ内のベースリレーションのセグメントを管理する。実際のタプルはその各ドメインに関して転置された形でデータブロックに保持される。連想処理の効率をあげるため，転置リスト中のドメイン値の位置はorder-preserving（順序性保持）ハッシュを用いて定められる。ハッシュが順序を保持するため，大小比較等の論理操作も高速に処理できることになる。またセグメントそのものを保持する代わりにＣＭＴ(Condensed Master Table)と呼ばれるタップルＩＤから転置リストのエントリへの変換表を持つことで記憶の節約を図っている。

ジョイン等の演算によって一時的に生成された中間リレーションに対してもＤＲＤ(Derived Relation Directory)，ＤＲＳＤ(Derived Relation Segment Directory)なるディレクトリが生成され，各々コントロールモジュール，データプロセッシングモジュールによって管理される。

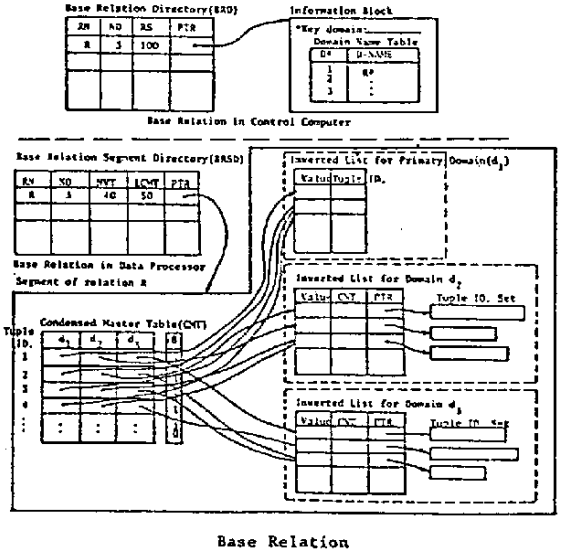


図 3.4.69

C．ジョイン処理方式

ジョイン処理にはデータプロセッシングモジュール間での通信が必要である。この場合，各データプロセッシングモジュールがＮＣＣ，従ってバスの制御権を奪い合い，権利を得たデータプロセッシングモジュールが自分を含めた全てのデータプロセッシングモジュールに１タプルをブロードキャストする。この時送出されるタプルはジョインを行う２つのリレーションの内，小さい方のリレーションから選ばれる。各データプロセッシングモジュールは，タプルを受け取るとこのタプルに対するジョイン処理を行う。この処理の後，再び各データプロセッシングモジュールはＮＣＣを奪い合い，以下ソースリレーションのタプルがつきるまで同様な処理が続けられる。

D．評　　価

10^2～10^8タプルのリレーションに関して５つの演算操作に対するシングルプロセッサ，および１００台のプロセッサを上記のように構成した場合の予測性能値を図3.4.７０に示す。いずれの場合においてもパラレルプロセッサ環境においてはかなりの性能改善が期待できることがわかる。

次にプロセッサ台数の増加に対する性能改善効果（プロセッサを１台増やした時に減少する処理時間の割合）に関する予測値を３つの演算について求めた結果を図3.4.７１，７２，７３に示す。リレーションサイズが増大するにつれ，プロセッサ台数の増加による性能改善は大きいが，一方である程度以上のプロセッサ台数の増加は，ほとんど改善をもたらさないことがわかる。

Execution Times in Seconds

	Database (power of ten)	Standard (no formed inverted)	Standard (pre-formed inverted)	Parallel implicit	Parallel explicit
equi-join	2	3.95	3.84	.35	.35
	3	4.95	3.85	.35	.35
	4	16.04	5.10	.40	.45
	5	231.88	124.44	.78	6.75
	6	13104.31	12009.84	5.65	605.45
	7	1211008.48	1200063.98	56.48	60054.68
	8	120110061.95	120000621.49	564.80	6000547.93
divide	2	3.06	2.95	.36	
	3	164.00	162.91	.38	n
	4	15137.37	15126.42	.63	o
	5	1665886.23	1665736.97	3.15	n
	6	332548122.34	332046968.80	29.41	e
	7	199629791586.73	199689791586.73	294.09	
	8	18637540251016B.2	18637540251016B.	2940.80	
restrict R(A=B)	2	.06		.36	
	3	.62	s	.37	n
	4	6.28	a	.41	o
	5	62.78	m	.86	n
	6	627.80	e	6.38	.n
	7	6276.00		63.69	
	8	62759.99		636.81	
restrict R(C<C1 or C<C2)	2	.13	.07	.35	
	3	.62	.07	.35	n
	4	5.55	.07	.35	o
	5	54.81	.08	.35	n
	6	547.43	.19	.38	e
	7	5473.63	1.27	.84	
	8	54735.87	12.07	3.27	
intersection	2	8.27	8.16	.35	
	3	9.29	8.20	.35	n
	4	21.62	10.68	.39	o
	5	358.80	249.36	.75	n
	6	25114.63	24020.16	5.41	e
	7	2411072.97	2400128.67	54.08	
	8	240110684.19	240001242.98	540.80	

図3.4.７０

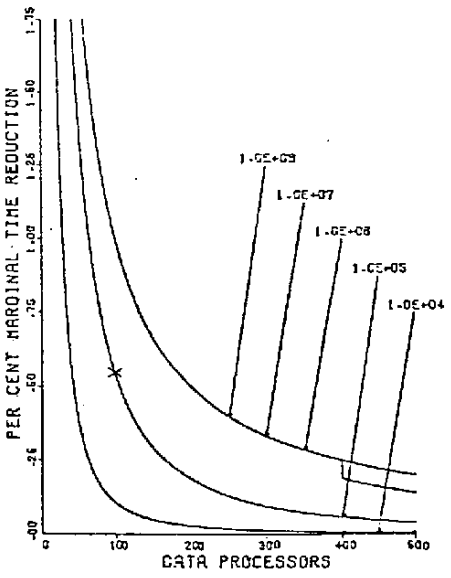


図3.4.7 1 "IMPLICIT" EQUI-JOIN

EFFECT ON EXECUTION TIMES OF INCREASING NUMBERS OF DATA PROCESSORS. DATABASE SIZES ARE FROM 10**4 TO 10**8. SIMILAR FOR INTERSECTION & RESTRICT A(A=3).

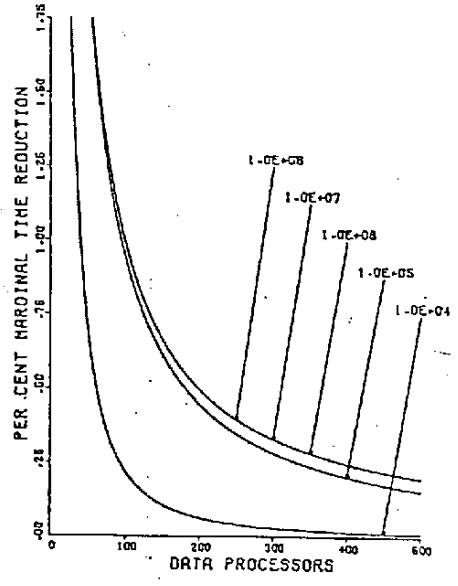


図3.4.7 2 "EXPLICIT" EQUI-JOIN

EFFECT OF EXECUTION TIMES OF INCREASING NUMBERS OF DATA PROCESSORS. DATABASE SIZES ARE FROM 10**4 TO 10**8. NOTE LITTLE CHANGE IN SLOPES AFTER 10**6.

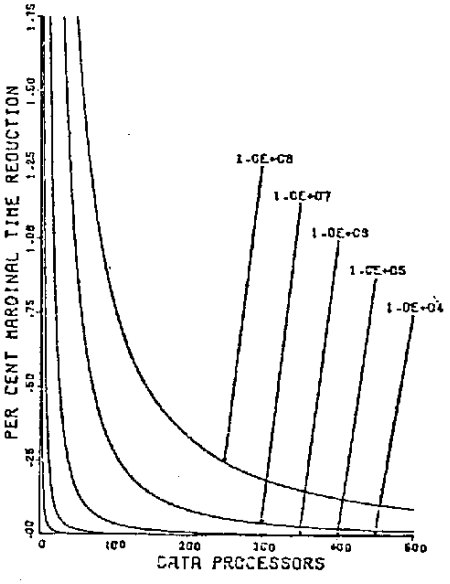


図3.4.7 3 R[C=C1 OR C=C2]

EFFECT ON EXECUTION TIMES OF INCREASING NUMBERS OF DATA PROCESSORS. DATABASE SIZES ARE FROM 10**4 TO 10**8. THIS IS THE SECOND EXAMPLE OF RESTRICTION.

3.4.18 DIALOG 47)

DIALOG(DIstributed Associative LOGic database machine)はPurdue大学で開発中のマルチプロセッサ・データベースマシンである。関係モデルに限らず他のデータモデルもサポートするとされている。マシンの基本構成はクラスタと呼ばれ，図3.4.74に示されるように1台のバックエンドコントローラとそれに制御される多数のデータモジュール，およびデータモジュールを結合する通信ネットワークからなる。データモジュールは，2次記憶，連想プロセッサ等で構成される。また，データモジュール数の増加によるネットワークコストの増大を押えるため，図3.4.75に示すような複数クラスタをコントローラを用いて接続する構成が可能である。この場合，異なるクラスタに属するデータモジュール間でもコントローラを経由することにより，相互に通信を行うことができる。

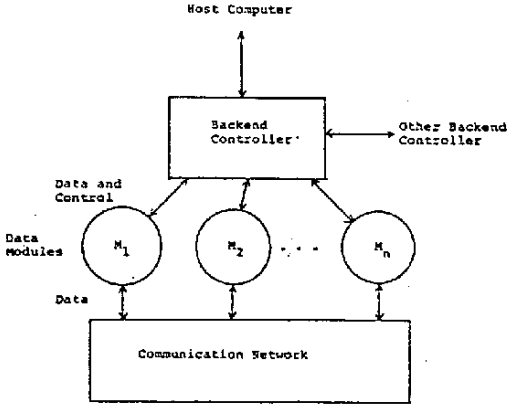


図3.4.74 System architecture of a cluster in DIALOG.

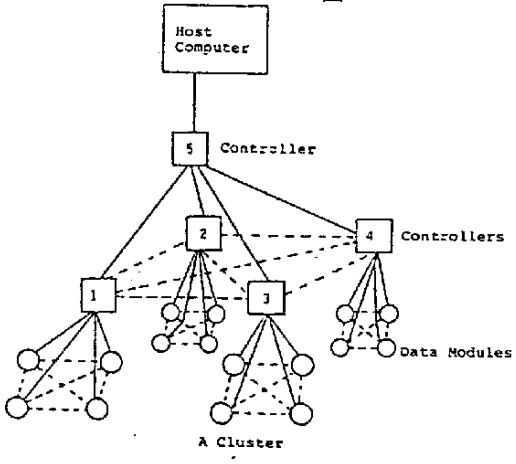
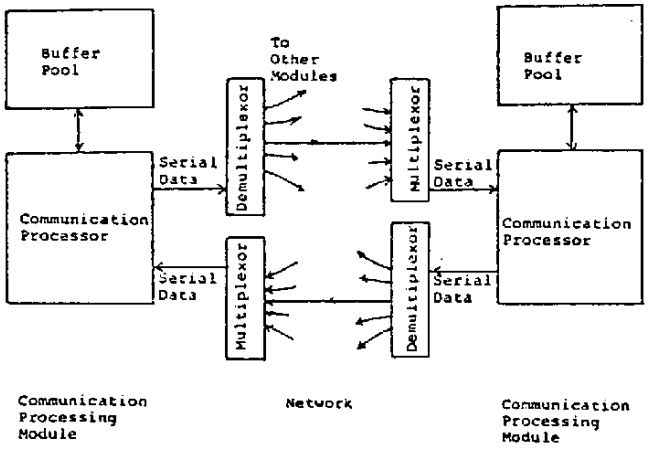


図3.4.75 An example of the hierarchical network in DIALOG.

A．ネットワーク

データモジュール間を結合するネットワークには図3.4.7 6に示されるようなマルチプレクサ／デマルチプレクサを用いたネットワークが用いられる。このネットワークでは，1対1の通信以外，ブロードキャストモードの動作も可能である。通常のジョイン処理は一方のリレーションをもう一方のリレーションを保持するデータモジュールに送出し，このデータモジュール内で処理を完了するが，ブロードキャスト機能を用いることにより，一方のリレーションを複数のデータモジュールに分散配置し，その後もう一方のリレーションをこれらモジュールにブロードキャストすることにより，複数台のデータモジュールによる並行処理が可能となる。



communication link between two data modules.

図3.4.7 6

B．データモジュールとジョイン処理

データモジュールは図3.4.7 7に示すように大きく5つのモジュールに分かれる。ジョイン処理モジュールはジョインプロセッサ，およびB_1，B_2，B_3の3つのバッファを持つ。B_1には通信処理モジュールから送られてきたオペランドリレーションが格納され，B_2には連想処理モジュールから出力されてきたもう一方のオペランドリレーションが格納される。これらのタプル毎のジョインはビットアレイB_3を参照することにより行われる。一方，物理記憶装置（2次記憶）にはビットシリアルワードパラレルのマルチループデバイスを仮定する。ここから読み出されたタプルはセレクトプロセッサ内のバッファを用いてシリアライズされ，さらにセレクション，プロジェクション処理を経て連想処理モジュールに送出される。連想処理モジュール内の連想シークエンシャルメモリにはB_1中の各タプルのジョインキーが格納されており，これらとセレクトプロセッサから送られてくるタプルとの比較が行われ，条件を満足したタプルのみがB_2に送出される。

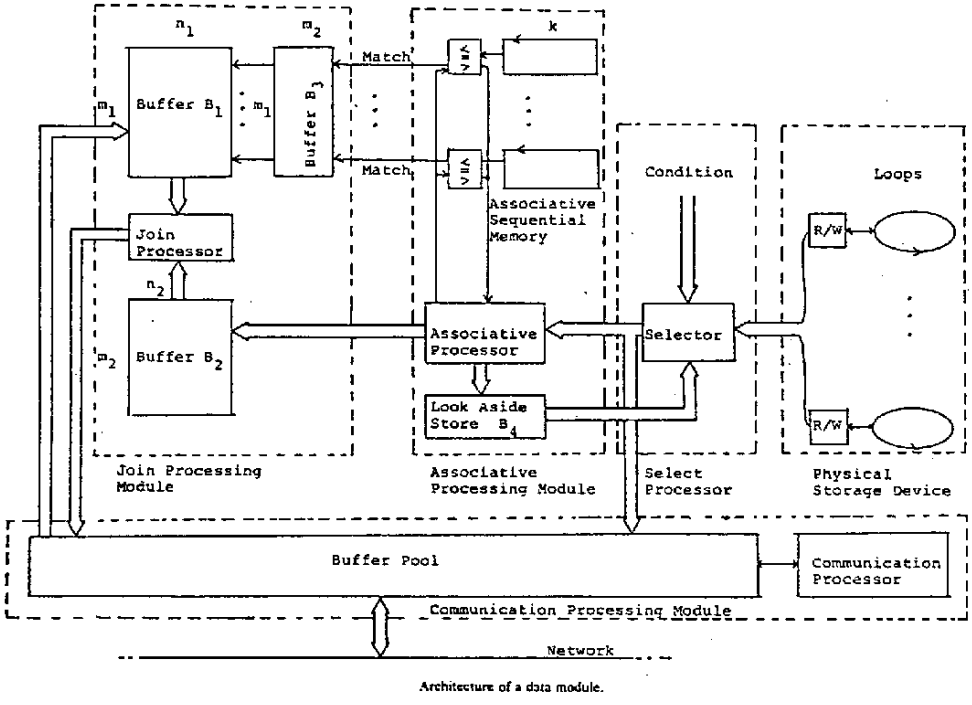


図3.4.77

図3.4.78にジョイン処理の例を，図3.4.79，80に連想処理用のハードウェア構成を示す。B_2がすでに一杯の場合には，タプルはルックアサイドストアB_4に残される。

C．バッファサイズの評価

データモジュールではバッファB_1およびB_2の大きさ（タプル数）が問題となる。すなわち，B_1に対しては，容量が大きすぎるとリレーションロード時のオーバーヘッドが増加し，一方で容量が小さすぎる場合にはジョインのヒットがほとんどなく，ジョインプロセッサが大部分アイドルとなってしまう。また，B_2の容量が小さすぎると連想処理モジュールの出力が絶えず待ちとなってしまう。B_1の容量S_1とジョインのヒット率Pをパラメータとして，必要なB_2の容量S_2をシミュレーションにより求めた結果を図3.4.81に示す。

D. そ の 他

DIALOGは，ブロードキャスト機能を用いた並列処理においても，リレーション送出時にタプルのふるい落しが行われており，DIRECTにおける並列処理環境よりも負荷の軽減が図られている。また中間リレーションを２次記憶に格納せず，そのまま次のデータモジュールに送出することによってリレーション転送レベルでのパイプライン化が実現され，一層の高速化が可能となっている。

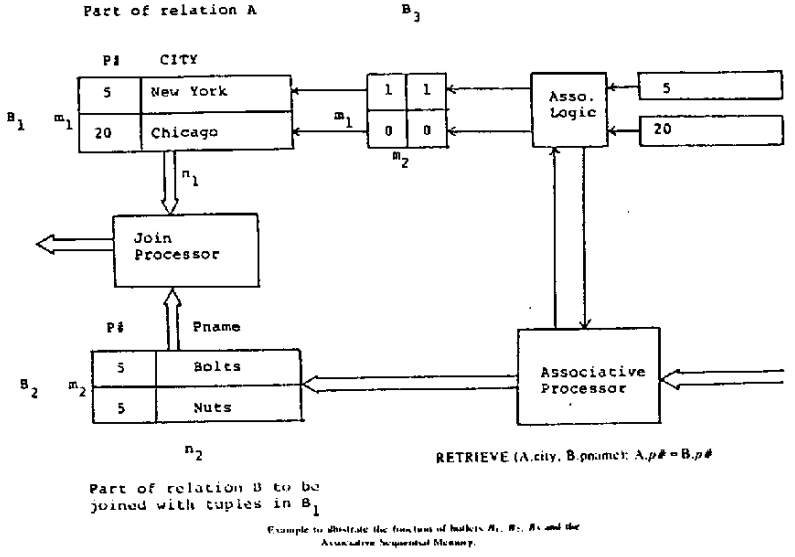
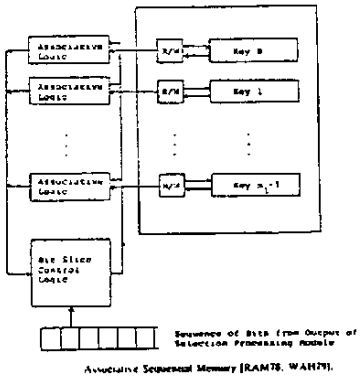
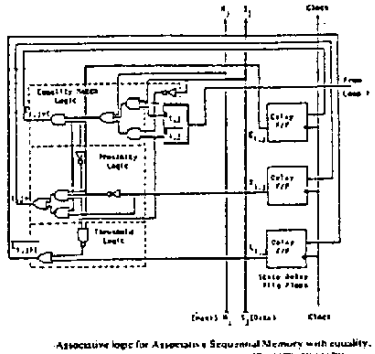


図3.4.78



Associative Sequential Memory [RAM78, WAH79].

図3.4.79



Associative logic for Associative Sequential Memory with equality, greater than, less than and proximity searches [RAM78, WAH79].

図3.4.80

		Simulation			
s_1	p	mean s_2	std. dev s_2	Pr(queue size<s_2) ≦ 0.95	Server Utilization
5	0.01	0.003	0.055	0 (0.9970)	0.099
	0.05	0.152	0.445	0 (0.8737)	0.477
	0.10	10.538	7.784	24 (0.9440)	0.976
	0.20	"	"	" (1.0)	1.0
10	0.01	0.015	0.122	0 (0.9850)	0.191
	0.02	0.068	0.266	0 (0.9352)	0.377
	0.03	0.264	0.620	1 (0.9480)	0.570
	0.04	0.925	1.313	3 (0.9407)	0.773
	0.05	9.572	6.495	20 (0.9460)	0.978
	0.07	"	"	" (1.0)	1.0
15	0.01	0.032	0.183	0 (0.9698)	0.280
	0.02	0.252	0.550	1 (0.9570)	0.588
	0.03	3.066	3.484	10 (0.9588)	0.880
	0.04	"	"	" (1.0)	1.0
25	0.01	0.177	0.523	0 (0.8641)	0.486
	0.02	29.993	12.346	46 (0.9534)	0.996
	0.03	"	"	" (1.0)	1.0
40	0.01	0.890	1.265	3 (0.9500)	0.770
	0.02	"	"	" (1.0)	1.0
50	0.01	22.332	10.911	36 (0.9500)	0.994
	0.02	"	"	" (1.0)	1.0
75	0.001	0.007	0.082	0 (0.9933)	0.141
100	0.001	0.013	0.115	0 (0.9865)	0.192
120	0.001	0.019	0.136	0 (0.9816)	0.229
140	0.001	0.028	0.169	0 (0.9729)	0.278
160	0.001	0.046	0.225	0 (0.9578)	0.317

図3.4.81

3.4.19 WCRC 48)

WCRC（Well−Connected Relation Computer）は，ANSI／SPARCが提唱する3層スキーマをマシンのアーキテクチャレベルで支援し，複数のデータモデルをサポートすることを目指すデータベースマシンである。

A．アーキテクチャ

マシンの構成を図3.4.82に示す。外部レベルでは，データモデル毎に異なるDMLで書かれた問い合わせを，概念レベルのDML WCRLに変換する。概念レベルではWCRLに変換された問い合わせを概念スキーマを参照しながら，内部レベルの言語WCRMLに変換する。概念スキーマの表現には，様々なデータモデルとの融合性の良さからEntity−Relationshipモデルが採用されている。内部レベルでは1つの問い合わせに対し，1台のQP（Query Processor）が割り合てられ，QPはWCRML で書かれたコマンドを複数台のCP（Cell Processor）を用いて実行する。QPは互いに並行に動作でき，全体としてMIMDの処理環境が形成される。CP間で直接通信し合うことはできず，QPを経由することが要求される。CPは

logic per-track式の連想プロセッサで，回転型の２次記憶を持つとされている。

B．記憶構造

概念レベルのEntity-Relationshipモデルによる表現は，多対多の２項関係に分解され，ＰＣＰ（Pseudo Canonical Partition）と呼ばれる方式によって回転記憶上に写像される。図3.4.8 3にその例を，図3.4.8 4にトラック上でのフォーマットを示す。ＰＣＰを用いたファイルでは，一定数のタプルを格納するのに必要なファイル容量は，タプルを単に並べただけの場合に必要とされるそれに比べて小さいことが示される。

C．処理方式

問い合わせ処理は，主に回転記憶への内容呼出しとマーキングを用いて行われる。例として図3.4.8 5で示されるＥ－Ｒダイアグラムに対し，

Find all department numbers and names of employees who work on the second floor of the SUNTOWER building

なる問い合わせの処理過程を以下に示す。図3.4.8 5のダイアグラムは図3.4.8 6に示されるようにＰＣＰに分解される。これらＰＣＰに対してＷＣＲＣは次のような動作を行う。

a．ＰＣＰ６（ＤＮＯ：ＦＬＯＯＲ）中，ＦＬＯＯＲ＝２のタプルを選びマークする。同時に

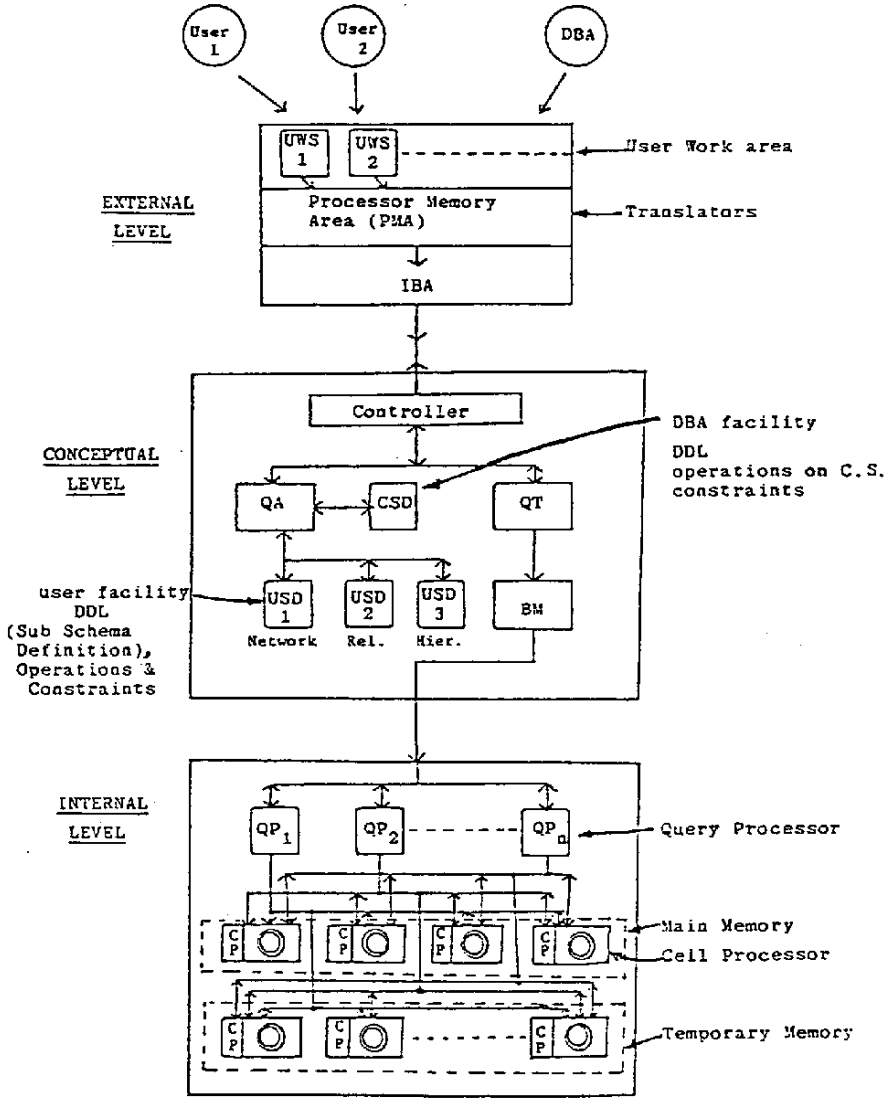
b．ＰＣＰ５（ＤＮＯ：ＡＤＤＲ）中，ＡＤＤＲ＝ＳＵＮＴＯＷＥＲのタプルを選びマークする。

c．ＰＣＰ５でマークされたタプル中，同じＤＮＯ値を持つＰＣＰ６のタプルがマークされているものだけを再びマークする。

d．ＰＣＰ８（ＤＮＯ：ＥＮＯ）のタプル中，同じＤＮＯ値を持つＰＣＰ５のタプルがマークされているものにマークする。

e．ＰＣＰ１（ＥＮＯ：ＮＡＭＥ）のタプル中，同じＥＮＯ値を持つＰＣＰ８中のタプルがマークされているものにマークする。

Ｅ－Ｒダイアグラムを表形式で管理し，同様な問い合わせ処理を実行した場合に比べると，ＷＣＲＣの方が高速であることが示される。



Overall Architecture of WCRL

図 3. 4. 8 2

a_1	b_1
a_1	b_2
a_1	b_3
a_2	b_2
a_2	b_1
a_3	b_1
a_3	b_2

(a) Binary Relation

(1) (, , ,) (a_1) (1, 2, 3)

(2) (, , ,) (a_2) (1, 2, ϕ)

(3) (, , ,) (a_3) (1, 2, ϕ)

(b) One half of track

(1) (, , ,) (b_1) (1, 2, 3)

(2) (, , ,) (b_2) (1, 2, 3)

(3) (, , ,) (b_3) (1, ϕ, ϕ)

(c) Other half of track

PCPs on a Track

図 3.4.8 3

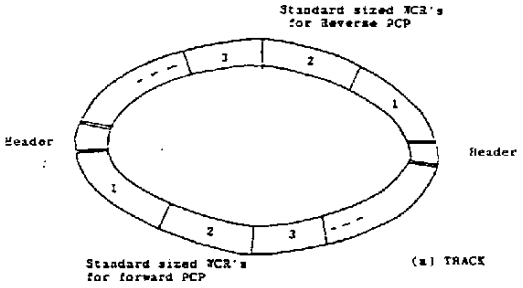


図 3.4.8 4

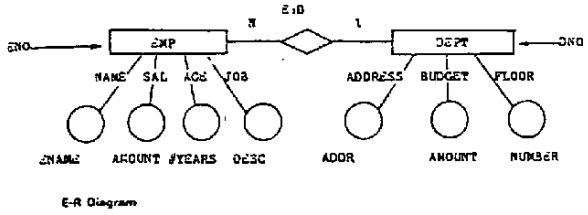


図 3.4.8 5

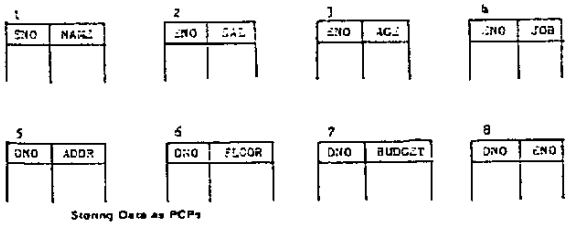
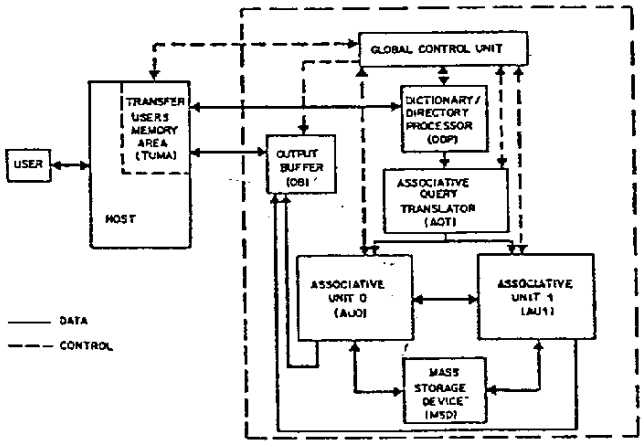


図 3.4.8 6

3.4.20 RELACS 28) 29)

RELACS(RELational Associatine Computer System)は，将来の連想メモリ（ＣＡＭ）の価格性能比向上の仮定の下で，問い合わせ処理の大部分をＣＡＭを用いて実現することを試みたバックエンドリレーショナルデータベースマシンである。システムは大きく５つのユニットに分かれ（図3.4.87），各々ＤＤＰ（Dictionary/Directory Processor），ＡＱＴ（Associatine Query Translator），ＡＵ（Assosiatine Unit），ＭＳＤ（Mass Storage Device），ＯＢ（Output Buffer）と呼ばれる。これらユニットは各々独立に動作でき，従ってこれらユニットをパイプライン化することによって問い合わせ間の並列処理が可能となる。また，問い合わせ処理の中心となるＤＤＰ，ＡＱＴ，ＡＵ各々にＣＡＭを使用することによって問い合わせ毎の処理速度も向上し，全体としてかなりの高速性能が期待できる。



RELACS, RELATIONAL ASSOCIATIVE COMPUTER SYSTEM, ORGANIZATION

図3.4.87

A. ＤＤＰ

ユーザの発行した問い合わせは，ホストを通じてまずＤＤＰに渡される。ＤＤＰは与えられた問い合わせの正当性やセキュリティチェックを行うと共に，さらに問い合わせ中の各リレーション，および属性に対してデスクリプション・データと呼ばれる内部コードを生成する。これらの機能は５つのＣＡＭとそれらを結合する４つのプログラマブルアレイを用いて実現される。（図3.4.88）

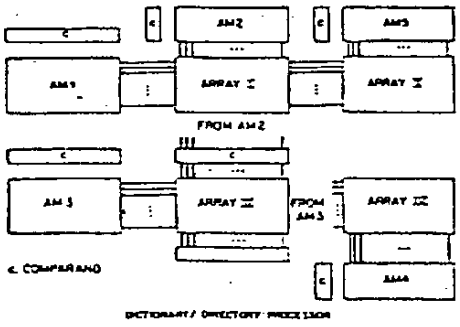


図3.4.88

B. A Q T

AQTはDDPからの入力をAUが実行可能な形式に変換する機能を持つ(図3.4.89)。DDPからの入力はCU(Control Unit)を経てQDM(Query/Data Memories)に渡され，QDMによってAUへの実行形式に変換され，結果はJQB(Job Queue Buffer)にFIFOの形で格納される。QDMは図3.4.90に示すように，さらに4つのユニットに分かれる。CUから送られてきたデータはCSR(Comparand/Stack Register)にロードされ，CSRの底から順にRCAM(Relation CAM)およびBCAM(Boolean CAM)を用いた連想処理が施される。RCAMにはリレーションのキー属性，アドレス等の情報が保持されており，さらにRCAMはRDMと呼ばれるバックアップ用のRAMメモリによって仮想化されている。一方，BCAMには属性名と属性値との大小関係や，AND，OR等の論理関係の処理に必要なデータが保持されており，BCAMはこれらの処理を行うAUのマイクロルーチンのアドレスを出力する。

C. A U

RELACSにはAUが2ユニット備えられており，各AUは図3.4.91に示すように4つのユニットからなる。Associative Systemはさらに6つの部分に分かれる(図3.4.92)。CA(Comparand Array)からIM(Input Mask register)によってマスタされたデータをAAM(Associatine Array Memory)を用いて連想サーチを行い，結果をRA(Response Array)に，選択データをOM(Output Mask register)を通じてAUOB(Associatine Unit Output Buffer)に出力する。AAMの入力を単一のレジスタでなくアレイとしたことにより，多対多のサーチ処理が可能となり，ジョイン処理の高速化が可能であるとされている。プロジェクション処理はOMを用いて行われる。

D. 評　　価

10^5 タプルのリレーションに対する性能予測値を図3.4.93に示す。ここにAAMは1K×1Kビットのモジュールを用いるものとし，従ってタプル長も1Kビット以下であると仮定する。問い合わせ中で参照しているリレーションおよび属性の数をパラメータとして評価しており，これに対する変動分が表中の各項に示されている。図から明らかなように，AMMの容量増加につれて性能は向上する。

相対的な評価としてRAPのシミュレーション性能値との比較を以下に示す。上と同様に 10^5 タプルのリレーションを対象とし，AMMの容量を8K～64Kビットまで変化させた場合，複数リレーションを参照する検索，除去，変更，追加に対し，RELACSは各々最良の場合で808～3,546倍，176～1,355倍，165～895倍，23～40倍程度 RAPの性能値を上回るとされている。一般に，問い合わせが複雑である程，RELACSの性能向上が著しい。

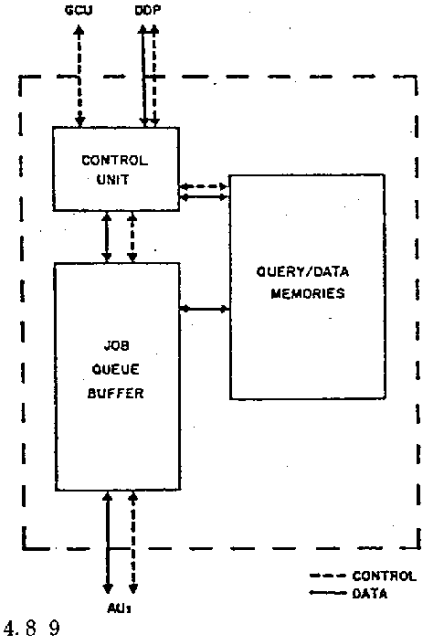


図3.4.89
AN OVERVIEW OF THE ASSOCIATIVE QUERY TRANSLATOR

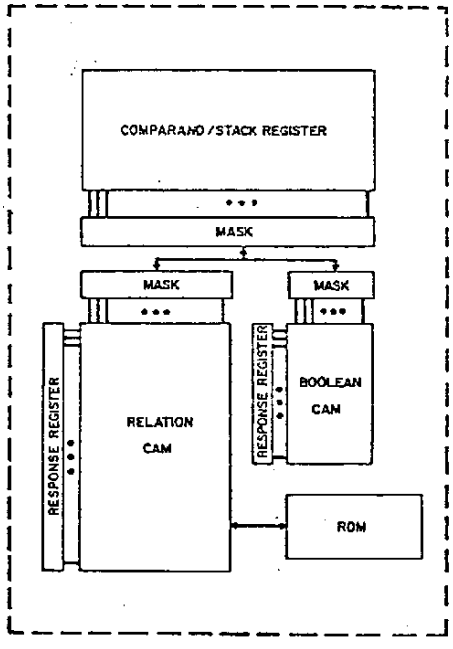


図3.4.90
QUERY/DATA MEMORIES

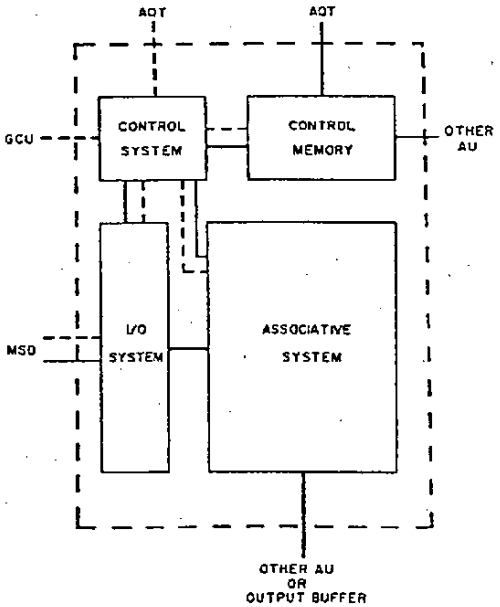


図3.4.9 1 THE ASSOCIATIVE UNITS

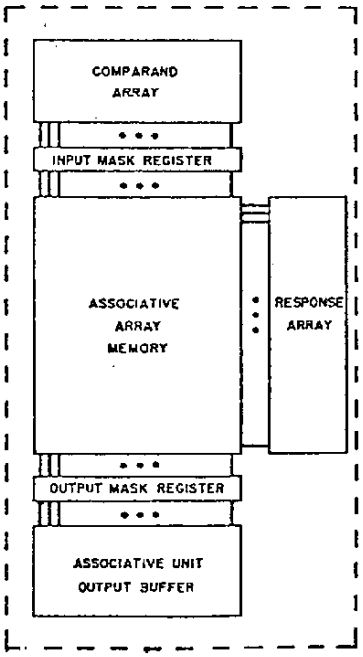


図3.4.9 2 THE ASSOCIATIVE SYSTEM

表3.4.9 3

AAM SIZE		8K	16K	32K	64K
RETRIEVAL	SIMPLE	9-22	5-14	4-11	3-9
	COMPLEX	21-184	12-101	7-59	4-32
DELETION	SIMPLE	12-28	7-15	4-9	2-5
	COMPLEX	29-324	15-177	8-98	4-45
MODIFICATION	SIMPLE	13-29	7-16	5-10	3-6
	COMPLEX	31-365	17-194	10-115	6-47

TIMES IN MILLISECONDS

図 3.4.9 3 SUMMARY OF RESULTANT TIMES WHEN RELATION CARDINALITY IS 10^3

3.4.2 1　ＤＢＣ　31)～38)

ＤＢＣは，オハイオ州立大学のD. K. Hsiaoらによって開発中のデータベースマシンである。セルラーロジックタイプデータベースマシンは大規模データベースのサポートに適しているとは考えられず，ＤＢＣでは論理要素をより少なくし，10^{10}バイト程度までの容量を経済的に達成することを目的としている。

A．特　　長

a．ＰＣＡＭ

ＰＣＡＭとはメモリ全体を多数のブロックに分け，ブロック内では，コンテントアドレッシング可能であるようなメモリである。このようなメモリに対して，検索を行う場合にはブロックのアドレスを指定するだけでよい。ブロック内のアドレスは不要となる。ＰＣＡＭの採用によりアドレッシングの単位が大きくなり（ＭＡＵと呼ぶ），複雑なName Mapping アルゴリズムが必要でなくなる。またブロック内のデータ移動に関してもディレクトリ情報の更新は不要になる。さらに，非分割型のコンテントアドレッシングメモリに比べて大きなコストの低減を図ることができる。ただし，データが多数のＭＡＵに分散している場合には当然多くのメモリをサーチせねばならないが，これについてはクラスタリング技法を適用し，性能の改善を試みている。

b．構造メモリ

データベース本体（10^{10}バイト）とディレクトリー情報（10^{7}～10^{9}バイト）を分離し，それぞれアクセスタイムの異なるＰＣＡＭに格納している。従来の多くのマシンでは一切補助データ構造をもたない場合が多く，データベース全体のサーチが必要となるが，ＤＢＣでは，構造メモリを用いて検索空間を狭める工夫をしている。

c．パイプライン構成

ＤＢＣは機能分散型のマシンであり，ディレクトリサーチ，ディレクトリ処理，マスメモリサーチの３つの処理が各機能モジュール間でパイプライン的に処理される。

d．クラスタリング

ユーザまたはＤＢＡはＰＣＡＭ中の２つ以上のＭＡＵを占める大きさのファイルを作る際，問い合わせでよく参照されると思われるキーワードをクラスタリングキーワードとして指定することができる。レコードはクラスタリングキーワードによりクラスタに分類され，同一クラスタに属するレコードは同じＭＡＵに格納される。

B．アーキテクチャ

図3.4.9.4に示されるごとくＤＢＣのアーキテクチャは２つのループ，すなわち，ストラクチ

ャループとデータループからなり，図中の各ユニットは，現在，または近い未来に利用可能な部品を用いて構成される。

ストラクチャループの各ユニットは検索内容の存在すると思われるデータベース内のＭＡＵアドレスを求めることがその主な役割である。一方，データループの各ユニットはストラクチャループで得られたＭＡＵアドレスをもとにデータベースに対するアクセスを行い，必要ならばそれらのデータにソートやセキュリティフィルターを施す。以下，各ユニットの概略を述べる。

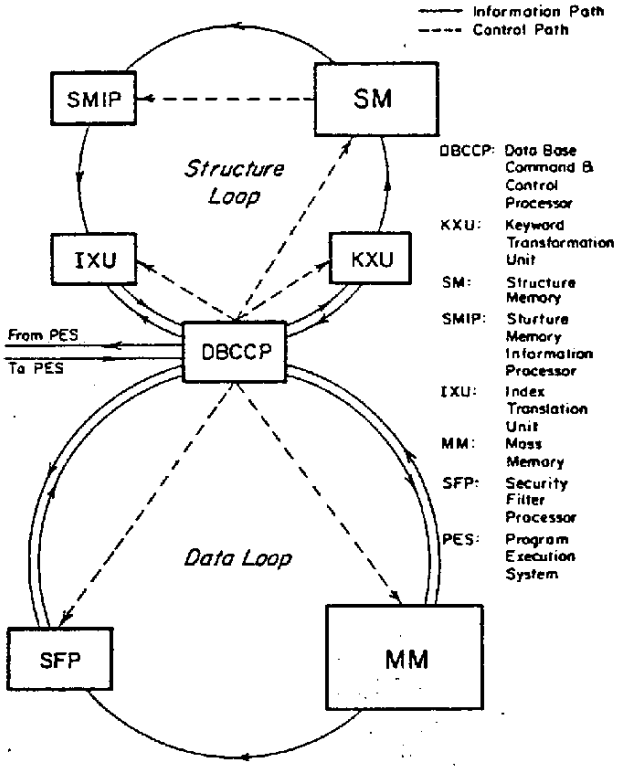


図 3.4.9 4　Architecture of the DBC.

(1) ＤＢＣＣＰ

ストラクチャループとデータループの動作を制御し，ホストとのインタフェースを提供する。１コマンド当たりの処理をディスク１～２回転内に行う必要がありマイクロプログラマブルなミニコンにより実現される。

(2) ＫＸＵ，ＩＸＵ

これら２つのユニットはストラクチャループ内での処理効率を上げるため，ＤＢＣＣＰから出力されるキーワードをエンコードしたりＳＭＩＰから出力されるインデックスをデコードする。

(3) S　　M

データベースのディレクトリ情報の管理，および処理を司る。データベースにアクセスする場合にはまず必要なディレクトリ情報をこのSMから読み出し，MAUアドレスの候補を求める。10^{10}バイトのデータベースのディレクトリ容量は10^{8}～10^{9}バイトに達すると考えられ，バブル，CCD，EBAMなどが記憶媒体の候補にあげられている。

図3.4.95に示されるような構成をしている。KXUによって探索を指示されるバケット，（属性識別子と値範囲のペアによって指定される）はモジュール間にわたって割り当てられており，各プロセッサが並列に処理することになる。またSMの性能を向上させるためにLook－Aside－Bufferが設けられており，ここに更新情報が一旦貯えられる。更新処理の他の検索に対する影響をできるだけ小さくしている。

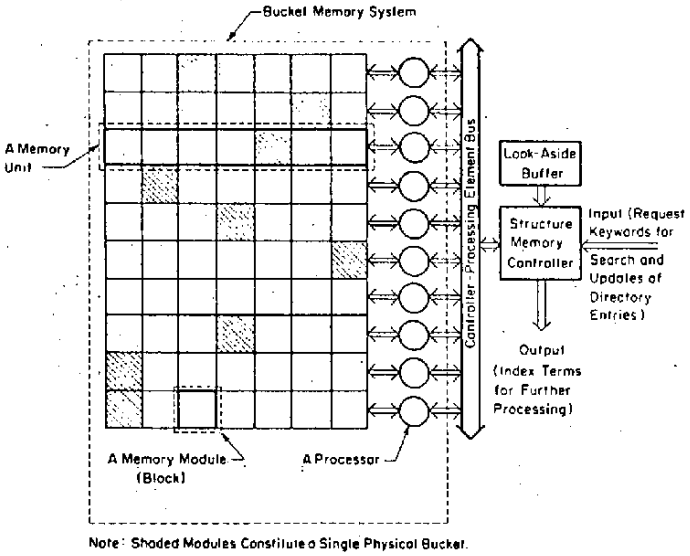


図3.4.95　Organization of the structure memory.

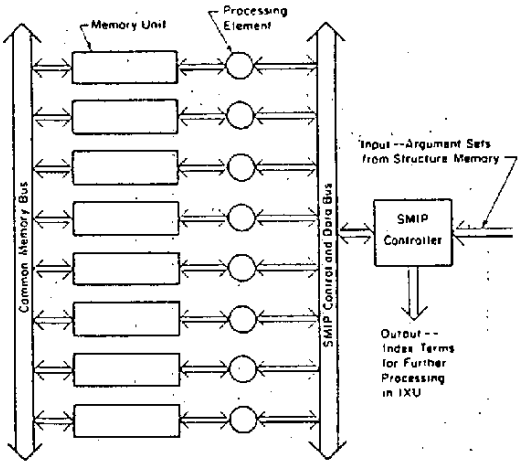
(4) SMIP

SM出力の後処理を行う部分である。すなわち問い合わせ

$$Q = P_1 \wedge P_2 \wedge \cdots\cdots \wedge P_n$$　　に対し

各述語P_iから得られるインデックスタームの積をとる。これを高速化するために図3.4.96に示されるごとく複数のメモリユニットMUとプロセッサエレメントPEから構成されている。送られてきたP_1についてのインデックスタームはコントローラによってハッシュされ，1つのPEを選択し，さらにもう一度ハッシュされてMU内のアドレスが決定される。以降の述語

P_i（ i＞1 ）によるインデックスタームは同様にハッシュされ，すでにそのインデックスタームが存在するかのチェックが全てのＰＥで並行して探索される。最終的に述語をみたすインデックスタームだけが出力される。



Organization of the structure memory information processor (SMIP).

図 3.4.9 6

(5) Ｍ　Ｍ

マスメモリではディスクの１シリンダをＭＡＵとしており，ストラクチャループによって生成されたアドレスをもつシリングに対して連想処理がなされる。ディスクはトラック並列に読み出すことが可能であり，各ヘッド毎にＴＩＰ（トラックインフォメーションプロセッサ）が付加されている。これによって１シリンダ内のデータを１回転で処理することができる。ＴＩＰは多くのディスクスピンドルによって共有化されている。（図 3.4.9 7）

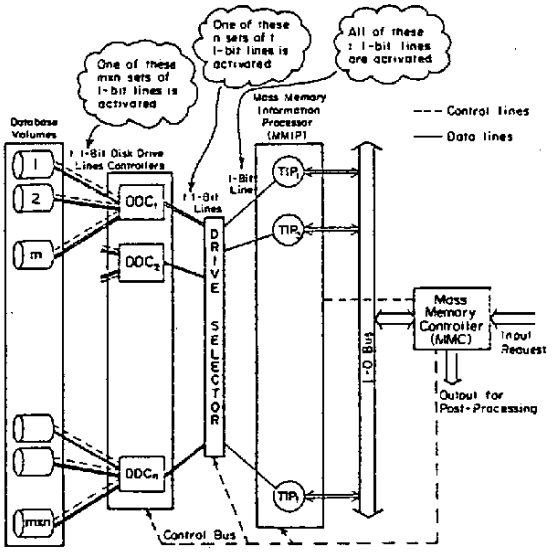
(6) Ｓ　Ｆ　Ｐ

キーワード以外のフィールドのセキュリティチェックはＭＭからレコードを検索した後，ＳＦＰで行われる。図 3.4.9 8のような構成をしており，比較演算が主体となる。その他，フィールドの選択や，ソート・集合演算なども行われる。

Ｃ．評　　価

ＤＢＣの性能に関する粗い予想では，従来のソフトウェア処理の８０～１６０倍の向上が期されるとしている。このうち２０～４０倍はディスクの並列読み出し機構によるもので，他はパイプライン処理に帰因している。

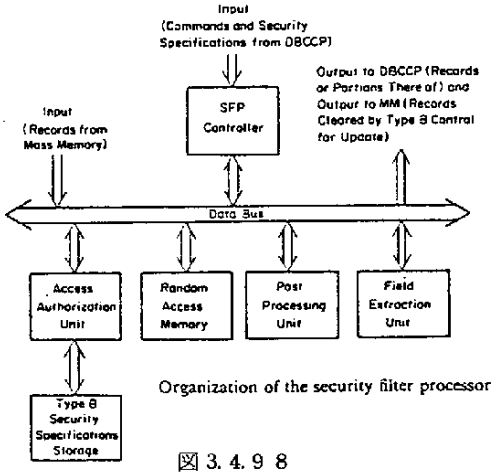
ＤＢＣの構成は上述のごとくかなり複雑なものとなっており，各機能ユニットを統括的に制御するＤＢＣＣＰに負荷が集中する恐れがある。



DDC: Disk Drive Controller
TIP: Track Information Processor
t = # of tracks per cylinder
m = # of disk drives per disk drive controller
n = # of disk drive controllers for the entire database

The mass memory organization.

図 3. 4. 9 7



Organization of the security filter processor (SFP).

図 3. 4. 9 8

3.4.22 バブルDBM 49)

本マシンは，ＩＢＭ H. Changによって提案された改良型磁気バルブを媒体とするデータベースマシンである。

A. バブルチップ

本マシンでは，リレーショナルデータベースに適した改良型磁気バブルチップを用いている。（図3.4.99） 従来のメイジャライン，マイナループ方式の磁気バブルに比べ，トランスファーラインがセグメント化されている点，およびバッファループが設けられている点が異なり，特に前者によりアトリビュートの選択読み出しが可能となっている。また，チップ外にはバブルのクロックと同期したマーカーループが設けられており，これがＲＡＰのマークビットに対応する。

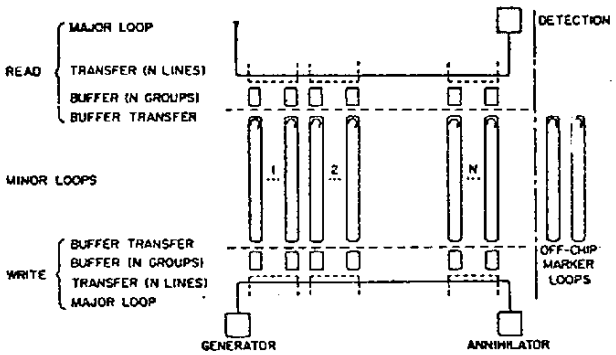


図3.4.99 An example of new chip architecture for bubbles.

B. マシン構成

本データベースマシンは，図3.4.100に示されるごとく，磁気バブルを媒体とするセルラロジック構成となっている。リレーション名やアトリビュート名，および対応するチップ番号に関する情報は，ディレクトリーに格納されている。各セルは，ＲＡＰ３と同様にバブルからのデータ流に対してサーチを行うが，このマシンの場合には，１ビット比較器だけで充分であるとしている。条件の満たされたタプルは，マーカーループに印が付けられる。

２次元のリレーションと磁気バブルの対応は明白であり，タプルが長い場合にはバブルを行方向に連結し，またタプル数が多い場合には，カラム方向に多数設ければよい。また，アトリビュート長は，セグメントの制御によって調整可能としている。

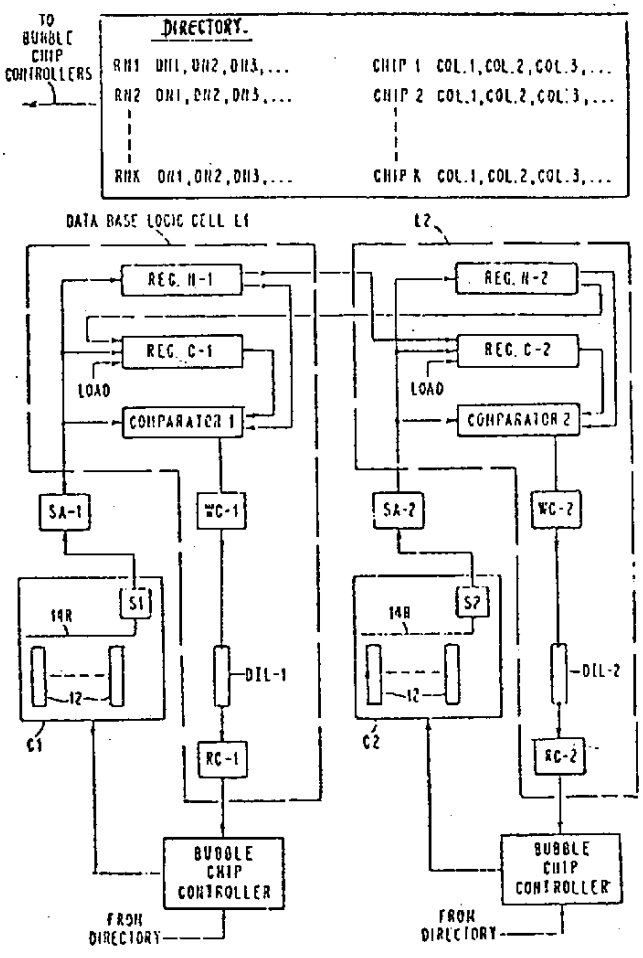
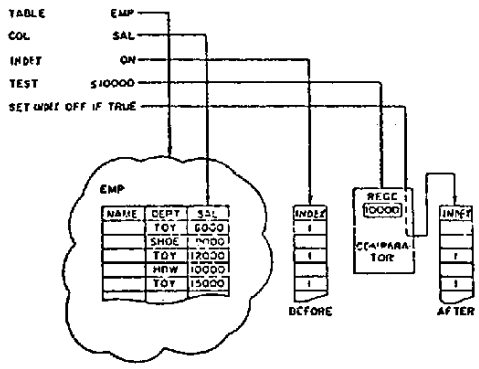


図3.4.100 A system sketch incorporating relational data-base bubble chips.

C. 処理方式

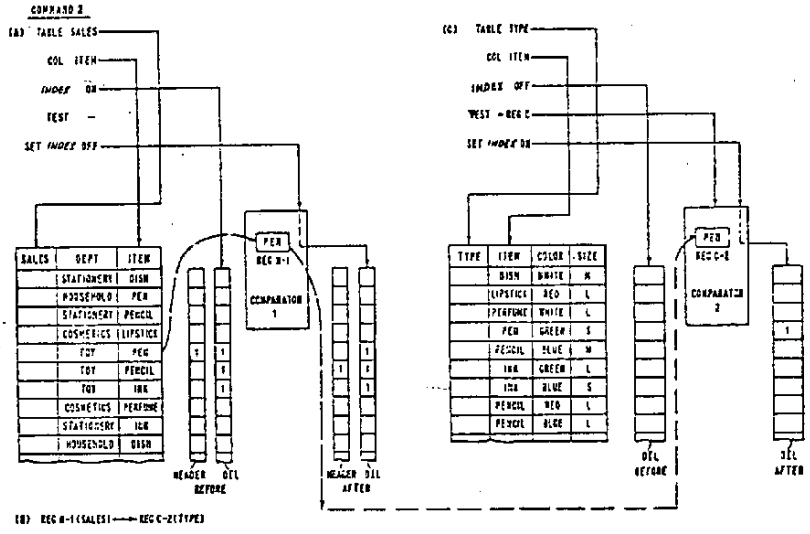
セレクションは，カラム毎に検索を行う。複数のアトリビュートを含む場合には条件を満たすタプルが毎回減少するため，効率のよい処理が可能となる。（図3.4.101）

ジョインは，複数のセル間で交信が必要となる。基本的にはＲＡＰと同じように，一方のリレーションのジョインアトリビュートを他方のリレーションを含むバブルモジュールに転送し，ジョイン操作を施す。（図3.4.102）



Data manipulation in relational data base corresponds closely to hardware operation in the modified bubble chip. The relaton name and domain name identify the chip and column. The index loop assists to restrict the retrieval and search on previously qualified tuples only. The test specifies the search criterion. The results are used to re-index the index loop(s).

図 3. 4. 101



REGC ← REGN is an instruction useful to link two relations. When used in combination with DO-loop and SELECT-NEXT, JOIN can be performed.

図 3. 4. 102

3.4.2 3 XDMS 17)（Experimental Data Management System）

本マシンは，ベル研究所のR. H. Canadyらによって開発された最初のソフトウェアバックエンドデータベースマシンである。

A．バックエンドコンセプト

データベース管理機能をホストマシンから分離することによって

(1) 経 済 性

専用化によるコストパフォーマンスの向上。ソフト的には汎用ＯＳ機能は不要であり，またルード的には浮動小数点命令，高速乗除算回路は不要であり専用化が図れる。

(2) データ共用の容易性 （図3.4.103）

(3) データベースの保護の容易性

ホストとバックエンド間での相互故障診断機構により高信頼化され，バックエンドとして切り離すことにより機密保護が容易になる。

(4) ＤＢＭ機能の付加の容易性

バックエンドシステムを一端作りあげれば，あとはホストインタフェースを修正するだけで容易にＤＢＭ機能を付加できる。

の4つの長所が生ずると考えられる。一方

(5) バックエンドマシンコスト

(6) 負荷の不均衡問題

ホストとバックエンドの間で負荷が不均衡になり，一方だけが過負荷になる恐れがある。

(7) 応答遅れ

ホストとバックエンド間の通信オーバヘッドによる応答の遅れが予想される。

上記3点の短所も生じうる。

これらの問題を明確化するために，実験システムが構成された。

B．実験システム

図3.4.104に示される実験システムは，UNIVAC1108をホストとし，マイクロプログラマブルなミニコンＭＥＴＡ－４をバックエンドプロセッサとしている。CODASYLタイプのデータベースシステムを製作し，実験がなされたが，システム作製には約6人年が費された。

C．評 価

ＸＤＭＳでは特に，(7)の問題が顕著にあらわれた。バックエンド方式をとることによって，バックエンドへのコマンド転送，バックエンドからホストへの結果の転送，およびこれらに伴う待時間，さらにホスト・バックエンド間における語長，データフォーマットの違いによる変換オー

バヘッド，という４つのオーバヘッドが追加されることになる。実験システムでは，２Ｋボーの低速回線を利用しており，２５０バイト程度の転送に２秒近く必要となった。高い転送レートをもつ回線を利用することによってある程度改善されるが，むしろホスト・バックエンド間のインタフェースレベルの向上が認識された。

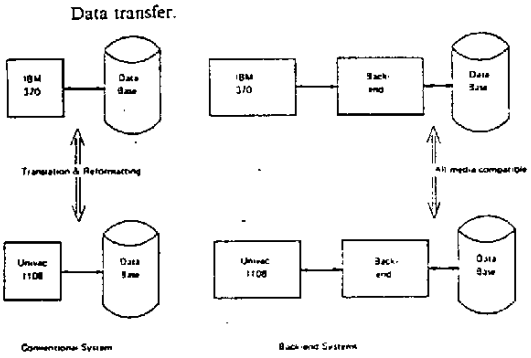


図 3. 4. 103

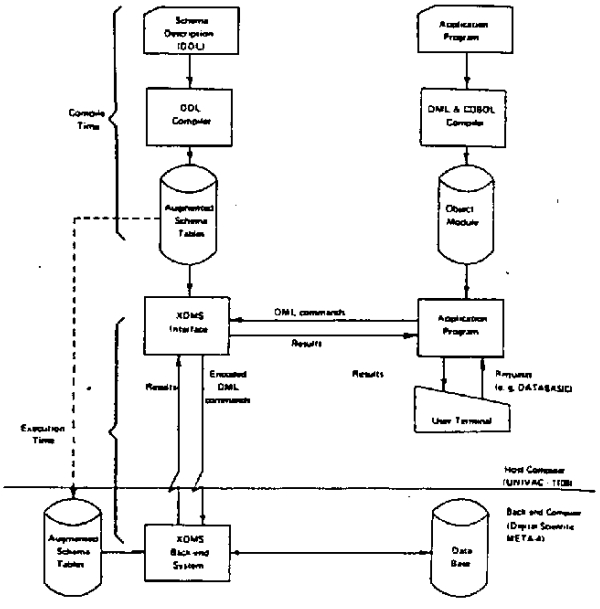


図3.4.1 0 4　XDMS data management system components.

3.4.2 4　データフローデータベースコンピュータ　50）

北海道大学の田中らによって開発中のデータベースマシンである。本マシンは，ＳＥＥ，ＳＯＥと呼ばれるサーチエンジン，ソートエンジンを基本構成要素として構築される。

A．サーチエンジン

図3.4.105に示されるサーチエンジンはランダムに与えられる入力キーに対して，それと同じキー値をもつデータを探索するマシンである。サーチエンジンの容量をＮとすると１つのキーをサーチするのにlog Ｎステップ要するが，これをパイプライン的に処理することができる。サーチエンジンの論理構造は左充填２進木であり，これはソートされたデータを入力として構成される。アーキテクチャは図3.4.106に示されるごとく，木構造をしており，各レベルの記憶容量は２のベキ乗で増加する。図3.4.107にサーチエンジンに対するローディング動作を図3.4.108に探索動作を示す。

ＳＥＥはその中をたどる際，左モードと右モードをとることができ，インターバルサーチも可能となっている。また，サーチキーの長さが長い時は，ビットスライス的な構成により拡張することができる。

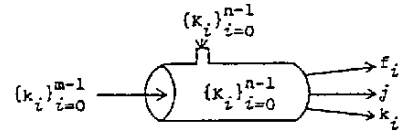


図3.4.105　Hardware symbol of an SEE.

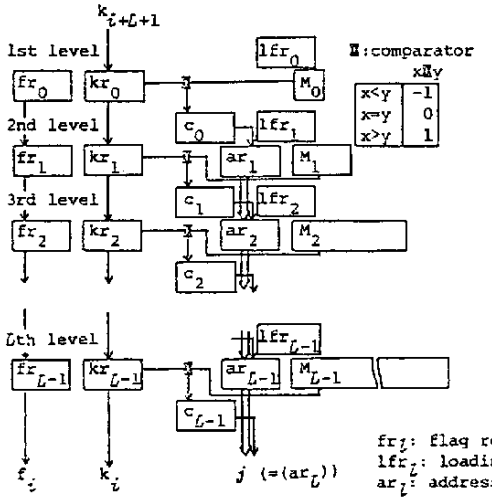


図3.4.106　Architecture of an SEE.

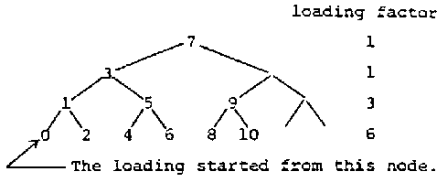


図3.4.107　A half loaded left-sided binary tree.

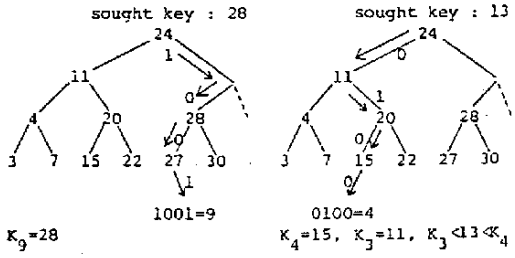


図3.4.108　Trajectory of the tree traversing and its binary number interpretation.

B．ソートエンジン

図3.4.109に示されるソートエンジンおよびソートエンジンシミュレータは，ランダムに入力されるデータをその流れに遅れることなく，パイプライン的にソートすることができる。ＳＯＥがキーのソートを行い，ＳＯＥＳがその結果にもとづき非キー部を取り扱う。アルゴリズムは，ヒープソートをパイプライン化したものであり，ヒープ生成フェイズおよびヒープ出力フェイズからなる。アーキテクチャは図3.4.110に示されるごとく，ＳＥＥ同様木構造をしている。図3.4.111にヒープ生成時の動作を図3.4.112にヒープからソート出力を得る時の動作を示す。またソータの容量を増加したい場合には，トーテンポール結合が可能である。

C．マシンアーキテクチャ

データベース処理は基本的にはソートとサーチで実現可能であるとし，上述のＳＥＥ，ＳＯＥをパケットスイッチング網で結合した図3.4.113に示されるごときマシンを提案している。10^{10}～10^{14}バイトのデータベースをサポートし，１メガタプルのジョインを５秒で行うことができるとしている。

例えば，ＲとＳのジョインは，次のように行われる。（図3.4.114）　まず，ＳＯＥを用いリレーションＲをジョインアトリビュートに関してソートする。次に，このソート出力をＳＥＥに

入れ，テーブルを作る。そしてリレーションＳをＳＥＥに入力し，ジョインをとる。この結果ジョインのとれるＲとＳのタプルＩｄペアが生成されるが，これらはリレーション本体を有するファイルプロセッサに転送され，そこで最終的に必要なアトリビュートが結合される。リレーションはカラム対応に分割されている。

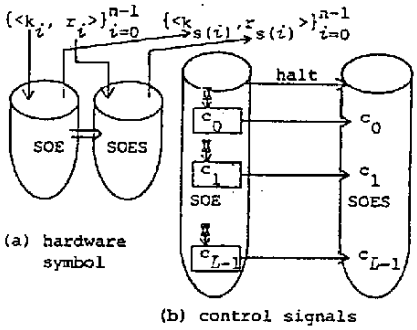
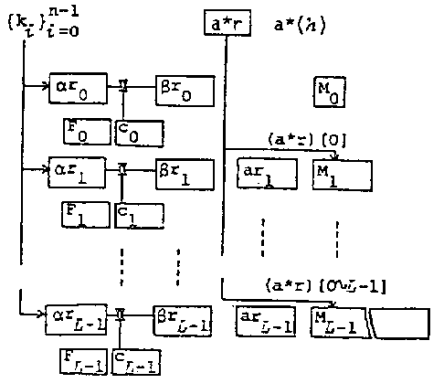


図3.4.1 0 9　Sorting with an SOE and SOES.



αr_l, βr_l: registers, F_l:even/odd flag, M_l: memory, ar_l: address register, a*r: address generator. (a*r)[0∿l] : the leftmost l+1 bits of the register a*r.

(a) architecture of an SOE.

a*(0)	0 0 0 0 0 0 ... 0
a*(1)	0 0 0 0 0 0 ... 0
a*(2)	1 0 0 0 0 0 ... 0
a*(3)	0 0 0 0 0 0 ... 0
a*(4)	0 1 0 0 0 0 ... 0
a*(5)	1 0 0 0 0 0 ... 0
a*(6)	1 1 0 0 0 0 ... 0
a*(7)	0 0 0 0 0 0 ... 0
a*(8)	0 0 1 0 0 0 ... 0
⋮	
a*(14)	1 1 1 0 0 0 ... 0
a*(15)	0 0 0 0 0 0 ... 0
a*(16)	0 0 0 1 0 0 ... 0
⋮	
a*(2^L-1)	0 0 0 0 0 0 ... 0

(b) a* sequence determining the next path.

図3.4.1 1 0　Architecture of an SOE.

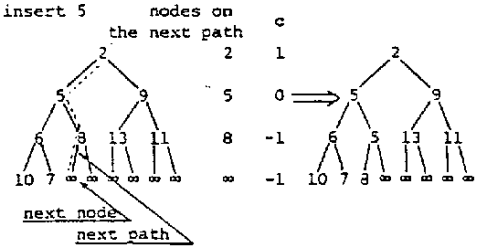
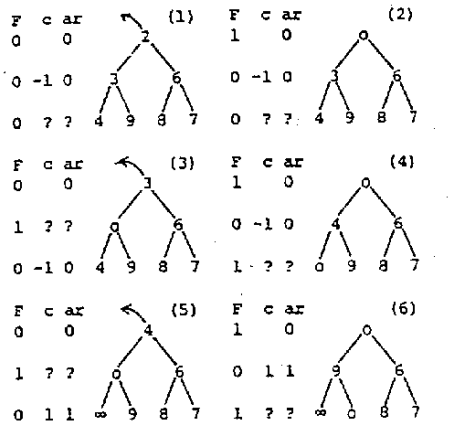
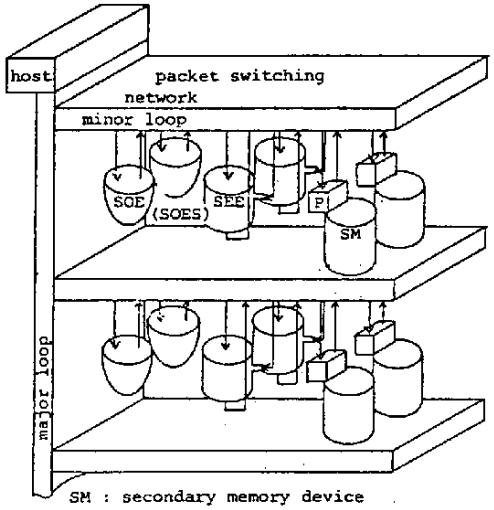


図3.4.1 1 1 Insertion of the next key into the next path.



An example of the pipelined heap output processes.



SM : secondary memory device
(e.g. magnetic bubble device)
P : microprocessor (file processor)

図3.4.1 1 3 An outline of a data flow database computer architecture.

R	A	B
1	3	a
2	2	c
3	5	b
4	1	c
5	4	a

(a)

S	C	D
1	c	z
2	a	x
3	b	z
4	a	y
5	c	x

(b)

	B	R^{-B}
1	a	1
2	a	5
3	b	3
4	c	2
5	c	4

(c)

	C	min	max
1	c	4	5
2	a	1	2
3	b	3	3
4	a	1	2
5	c	4	5

(d)

図3.4.1 1 4 Examples of relations and the processing of their join.

3.4.2 5 GRACE 51)

GRACEは，東京大学の喜連川らによって開発されているHashとSortにもとづく関係代数マシンであり，ジョインやプロジェクションなどを$O(m)$（m：メモリページサイズ）時間で処理することができる。処理負荷の重い関係代数演算が多用される環境においても高いパフォーマンスを得ることを目的として研究されている。

A. 処理方式

GRACEではHashとSortによる高速関係代数アルゴリズムを採用している。ジョイン処理の場合，ジョインアトリビュートにHashを施し，両リレーションを互いに独立なバケットに分割する。これにより$O(N\times M)$の処理負荷を$O(\sum_{i=1}^{S} n_i \times m_i)$(N, M：リレーションカーディナリティ，S バケット数，$N=\sum_{i=1}^{S} n_i$　$M=\sum_{i=1}^{S} m_i$）にすることができる。（図3.4.115）次に，こうして生成されたバケットを多数のプロセッサによって並列処理する。各プロセッサは内部に$O(n)$ソータを備えておりバケットをその大きさに比例した時間で処理することができ，データ流に追従した処理がなされる。以上のごとく，Hashを用いリレーションをクラスタリングし，バケットレベルで$O(S)$処理とし，さらに生成されたバケットをソータによって$O(n+m)$時間で処理することにより，全体として$O(N+M)$時間で処理することができる。さらに，バンクパラレリズムを反映させることが可能であり，k台のメモリモジュールを用いて，リレーションを並列に処理することにより，$O(m)$（$m=\frac{N+M}{K}$）の処理を実現している。すなわちリレーションの大きさによらず，メモリページサイズに比例した一定時間内に処理することが可能となっている。

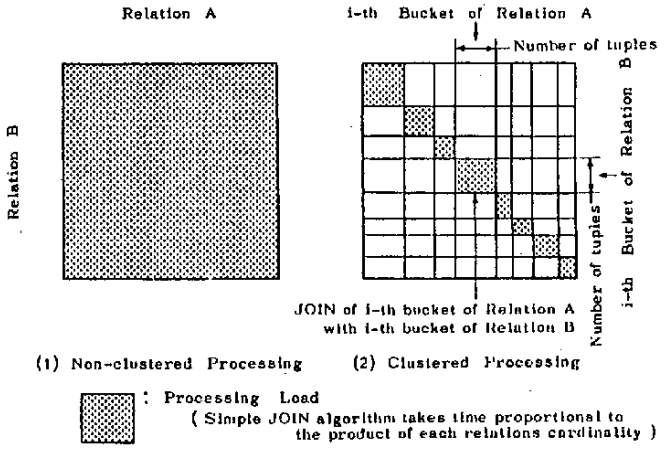


図3.4.115 Clustering Effects By Hashing In JOIN Operation

B．アーキテクチャ

図3.4.116にGRACE抽象アーキテクチャを図3.4.117に実際のマシンアーキテクチャを示す。また各モジュールを図3.4.118に示す。SDM（ディスクモジュール）は2次記憶としてディスクを用いており，ここにリレーションが貯えられている。処理時には，セレクション，プロジェクション，および，クラスタリングを施しながらDSGに必要なデータのみをステージングする。DSG（メモリモジュール）は，DSPに対してパケットシリアルなデータ流を発生することがその役割であり，媒体としては改良された磁気バブルメモリ，RAMなどが考えられている。DSP（プロセッシングモジュール）は，Stream Driven Sorter を内蔵しており，これによって割り当てられたパケットをO（n）時間で処理する。また処理結果に対して，次オペレーションに関するクラスタリング処理も行う。これらのモジュールを結合するネットワークとしては，現在，リングバスが検討されている。

C．オペレータレベルパイプライン

GRACEでは複数のデータ流に対する並列処理に加え，種々のレベルのパイプライン処理が行われている。すなわち，ソータ内 log Nプロセッサによるマージソートのパイプライン処理，パケットのパイプライン処理，そしてオペレータレベルパイプラインである。特に最後の関係代数演算子レベルのパイプライン処理により，クラスタリング処理と，ジョイン等の真処理とを重畳させることが可能となり効率の良い関係代数木処理が実現される。

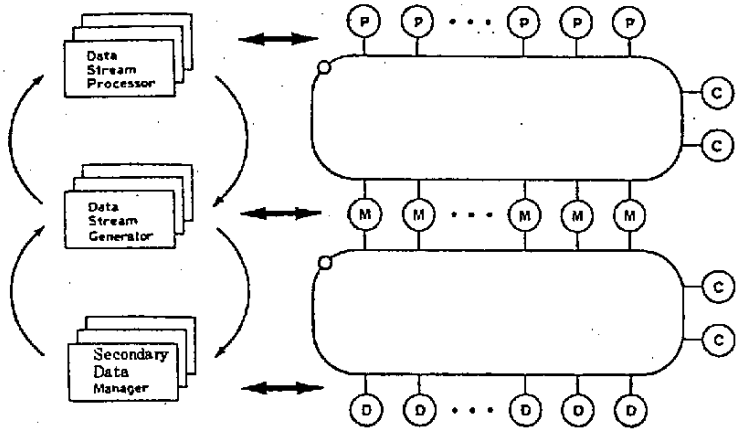
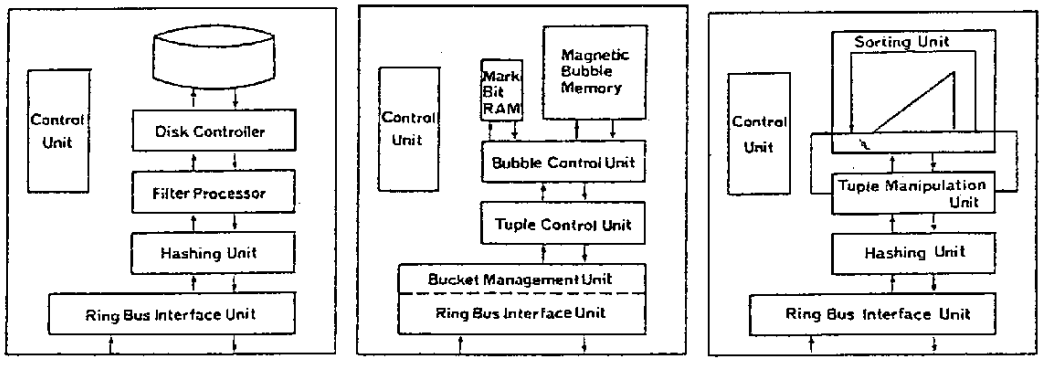


図3.4.116 Abstract Architecture Of GRACE

図3.4.117 Global Architecture Of Data Manipulation Subsystem In GRACE



Disk Module Organization

Memory Module Organization

Processing Module Organization

図3.4.118

—参考文献—

1) O.H.Bray, H.A.Freeman
「Data Base Computers」
Lexington Books, 1979

2) P.B.Hawthoron, et al,
「Performing Analysis of Alternative Database Machine Architectures」
IEEE Trans. on S.E. №1, 1982 p61~76

3) P.B.Hawthorn
「Evaluation and enhancement of the performance of relational database management systems」
Electron. Res.Lab., Univ. of California, Berkeley, Memo. M 79-70

4) E. A. Ozkarakan, S. A. Schuster, K. C. Smith: A Data Base Processor, Technical Report CSRG-43. Computer Systems Research Group, Univ. of Toronto, 1974.

5) S. A. Schuster, E. A. Ozkarahan, K. C. Smith: RAP-An Associative Processor for Data Base Management, Proc. AFIPS NCC, Vol. 44, pp. 379-387, 1975.

6) S. A. Schuster, E. A. Ozkarahan, K. C. Smith: A Virtual Memory System for a Relational Associative Processor, Proc. AFIPS NCC, Vol. 45, pp. 855-862, 1976.

7) E. A. Ozharahan, S. A. Schuster, K. C. Sevcik: Performance Evaluation of a Relational Associative Processor, ACM Trans. Database Syst., Vol. 2, No. 2, pp. 175-195, 1977.

8) E. A. Ozkarahan, K. C. Sevcik: Analysis of Architechtural Features for Enhancing the Performance of a Database Machine, ACM Trans. Database Syst., Vol. 2, No. 4, pp. 297-316, 1977.

9) S. A. Schuster, H. B. Nguyen, E. A. Ozkarahan, K. C. Smith: RAP. 2—An Associative Processor for Databases and Its Applications, IEEE Trans. Comput., Vol. C-28, No. 6, pp. 446-458, 1979.

10) K.Oflazer, E. A. Ozkarahan
「RAP.3- A multi-microprocessor cell architecture for the RAP database machine」
Proc. of the Inter. Workshop on High Lenel Language Computer Architeclure, 1980 pp108-119

11) G. P. Copeland, G. J. Lipovski, S. Y. W. Su: The Architecture of CASSM: a Cellular System for Non-numeric Processing, Proc. 1st Annu. Symp. Computer Architecture, pp. 121—128, 1973.

12) S. Y. W. Su, G. J. Lipovski: CASSM: a Cellular System for Very Large Databases, Proc. Int. Conf. on Very Large Data Bases, pp. 456-472, 1975.

13) S. Y. W. Su, L. H. Nguyen, A. Emam and G. J. Lipovski: The Architectural Features and Implementation Techniques of the Multicell CASSM, IEEE Trans. Comput., Vol. C-28, No. 6, pp. 430-445, June 1979.

14) D. J. DeWitt: DIRECT—A Multiprocessor Organization for Supporting Relational Database Management Systems, IEEE Trans. Comput., Vol. C-28, No. 6, pp. 395—406, 1979.

15) D. J. DeWitt: Query Execution in DIRECT, Proc. ACM-SIGMOD 1979, pp. 13-22, 1979.

16) H. Boral, D. J. DeWitt
「Processor Allocation Slrategies for Multiprocessor Database Machines」
ACM TODS, Vol 6, №2, June 1981, pp227-254

17) R. H. Canady, et al.: A Back-end Computer for Data Base Management, Comm. ACM, Vol. 17, No. 10, pp. 575-582, 1974.

18) T. Maril and D. Stern: The Datacomputer-A Network Data Utility, Proc. AFIPS NCC, Vol. 44, pp. 389-395, 1975.

19) R. Epstein et al
「Design decisiors for the intelligent database machine」
NCC, 1980 pp237-241

20) R. Epstein, et al
「The IDM 500-Communication Issues with Backend Processors」
COMPCON, 1981, pp 112-114

21) K. Hakozaki, et al.: A Conceptual Design of a Generalized Database Subsystem, Proc. 3rd Int. Conf. on VLDB, pp. 246-253, 1977.

22) 牧野, 箱崎, 水摩, 梅村, 日吉, 渡辺: データベースマシン実験システム, 信学技報 EC 79-68(計算機アーキテクチャ研究会 37-18), 1980.

23) 水摩, 日吉, 箱崎: データベースマシンによる CODASYL 型 DBMS の実現と評価, 信学技報 79-69(計算機アーキテクチャ研究会 37-19), 1980.

24) G. F. Coulouris, J. M. Evans and R. W. Mitchell: Towards Content-addressing in Data-bases, Computer Journal, Vol. 15, No. 2, pp. 95-98, 1972.

25) E. Babb: Implementing a Relational Database by Means of Specialized Hardware, ACM Trans. Database Syst., Vol. 4, No. 1, pp. 1-29, March 1979.

26) H. O. Leilich, et al,
「A Search Processor for Data Base Management」
VLDB, 1978, pp 280～287

27) S. Uemura, et al
「The design and implementation of a magnetic-bubble database machine」
IFIP 80, 1980, pp433～438

28) E. J. Oliver
「Utilizing associatine array deruies in a data base computer design」
COMPCON 1981, pp108～111

29) E. J. Oliver et al
「RELACS : A Relational Associative Computer System」
Proc. of the 5th Workshop on Computer Architeclure for Non-Numeric Processing, pp108-114

30) H. T. Kung, P. L. Lehman
「Systolic (VLSI) Arrays for Relational Database Operations」
Proc. of ACM-SIGMOD, Inter, Conf of Management of Data, pp105～116, 1980

31) R. I. Baum and D. K. Hsiao: Database Computers—A Step Toward Data Utilities, IEEE Trans. Comput., Vol. C-25, No. 12, pp. 1254-1259, Dec. 1976.

32) J. Banerjee, R. I. Baum and D. K. Hsiao: Concepts and Capabilities of a Database Computer, ACM Trans. Database Syst., Vol. 3, No. 4, pp. 347-384, Dec. 1978.

33) J. Banerjee, D. K. Hsiao and K. Kannan: DBC—A Database Computer for Very Large Databases, IEEE Trans. Comput., Vol. C-28, No. 6, pp. 414-429, June 1979.

34) D. S. Kerr: Data Base Machines with Large Content-addressable Blocks and Structural Information Processors, IEEE COMPUTER, Vol. 12, No. 3, pp. 64-79, March 1979.

35) D. K. Hsiao, K. Kannan and D. S. Kerr: Structure Memory Designs for a Database Computer, Proc. ACM 77 Annu. Conf., pp. 343-350, 1977.

36) K. Kannan: The Design of a Mass Memory for a Database Computer, Proc. 5th Annu. Symp. on Computer Architecture. pp. 44-51, 1978.

37) J. Banerjee and D. K. Hsiao: The Use of a Database Machine for Supporting Relational Databases, The Papers of the 4th Workshop on Computer Architecture for Non-Numeric Processing, pp. 91-98, 1978. (ACM SIGMOD Vol. X, No. 1).

38) J. Banerjee and D.K. Hsiao, 「Performance Study of a Database Machine in Supporting Relational Databases」 Proc. 4th Int. Conf. on VLDB,

39) C. S. Lin, et al
「The Design of a Rotating Assoiative Memory for Relational Database Applications」
ACMTODS, Vol, №1 pp53-65, 1976

40) Hsiao, D.K, and Memon, M.J., 「The Post Processing Functionsof a Database Computers」 Technical Report, OSU-CISRC-TR-79-6 Ohio State University, July 1979

41) M. J. Menon, D. K. Hsiao
「Design and Analysis of a Relational Join Operation for VLSI」
Int, Conf on VLDB, pp 44～55, 1981

42) B. Arden, et al
「A single - relation module for a data base machine」
COMPCON, 1981, pp 227～237

43) M. Muraszkiewicz
「Concepts of sorting and projection in a cellar array」
VLDB, 1981, pp 76-80

44) H. Boral, D. J. DeWitt
「Design Consideration for Data - flow Database Machines」.
Proc. of ACM-SIGMOD 1980
Int Conf on Management of Data, pp 94-104

45) S. W. Song
「A Highly Concurrent Tree Machine for Database Applications」
Proc of the 1980 Int. Conf. on Parallel Processing,
pp259-268, 1980

46) S. Su et al
「MICRONET : A micromputer network system for managing distributed relational databases」
VLDB, 1978, pp288-298

47) B. W. WAH et al
「DIALOG — A distributed processor organization for data base machine」
NCC, 1980, pp243-253

48) S. K. ARORA, et al
「WCRC : An ANSI SPARC machine architecture for data base management」
Computer Archterture 81, pp373-387

49) H. Chang 「On Bubble Memories And Relational Data Base」
VLDB 1978 pp 207-229

50) Y. Tanaka et al 〃Pipeline Searching and Sating Modules as Components of A Data Flow Data Base Computer〃
IFIP80, 1980, pp427～432

51) 喜連川他 「HashとSortによる関係代数マシン」
信学技報 EC81－35 1981

4. 実　現　技　術

4.1 ハードウエア

4.1.1 素子技術

データベースマシンは１９８０年代に入っていくつかの商用マシンが発表されたが，今後も高速処理，記憶の大容量化を指向した開発が続けられるものと思われる。

この処理の高速化，記憶の大容量化の技術は，新しいコンピュータアーキテクチャの開発・研究とともに，関連する素子技術の進歩に負うところが大きい。

データベースマシンの進歩に比較的関連が深い素子技術のなかで，１９８０年代に実現するであろうと予想されている素子技術を概観してみるに，半導体素子系技術のうち，従来からのシリコン素子技術による１０K～４０Kゲート以上のＶＬＳＩ技術は，多数のプロセッサに処理を分散する機能分散化指向のデータベースマシンにとっては大きなウェートを占めるであろうし，２５６K～４MビットのＲＡＭ，磁気バブルなどの記憶素子の進歩も，大容量化傾向の強いデータベースシステムにとって不可欠な技術要素であり，これらの記憶素子を使用することで，大容量固体ディスクなどの新しいアプリケーションの実現も可能となるであろう。

１９８０年代後半には実用化が可能であろうとの予測もあるジョセフソン素子なども，新しい考えにもとづくアーキテクチャの実現に道を拓くことになるかもしれない。

一方，現状では主流を占めている磁気ディスクなどの磁性メモリも，１９８０年代には現在の数倍の記憶容量が可能となるであろうが，それにもまして，磁性メモリの５～１０倍の記録密度を持つといわれる光磁気ディスクも期待される新しい記憶素子の１つで，１００Gバイト以上もの記憶容量を必要とする大規模データベースにとって有効な手段となろう。

本項では，以上述べてきたような，半導体素子，磁性メモリ，光関連素子などデータベースマシンシステムに比較的関連が強いと思われる素子技術について，１９８０年代を通した技術動向を中心に述べる。

A. 半導体素子

半導体素子関連の技術は，データベースマシンの今後の動向を左右する最も重要な技術要素であるといえる。

専用プロセッサによる機能の分散化，複数の同一プロセッサで構成する負荷分散化,指向の強いデータベースマシンにとって，低価格ＶＬＳＩプロセッサ機能の実現は重要なポイントになるであろう。ＲＡＭ，ＣＣＤ，磁気バブルなどの記憶素子の大集積化も，半導体ディスクなども，メモリ階層化技術の研究開発と実現には不可欠の技術である。

図4.1.1に半導体メモリ素子の集積度予測[1)]を示す。この図からも分かるように，ＲＡＭ，ＣＣＤ，磁気バブルの各素子ともその集積度は２～３年で２倍程度の進展は，期待できるものと思われる。

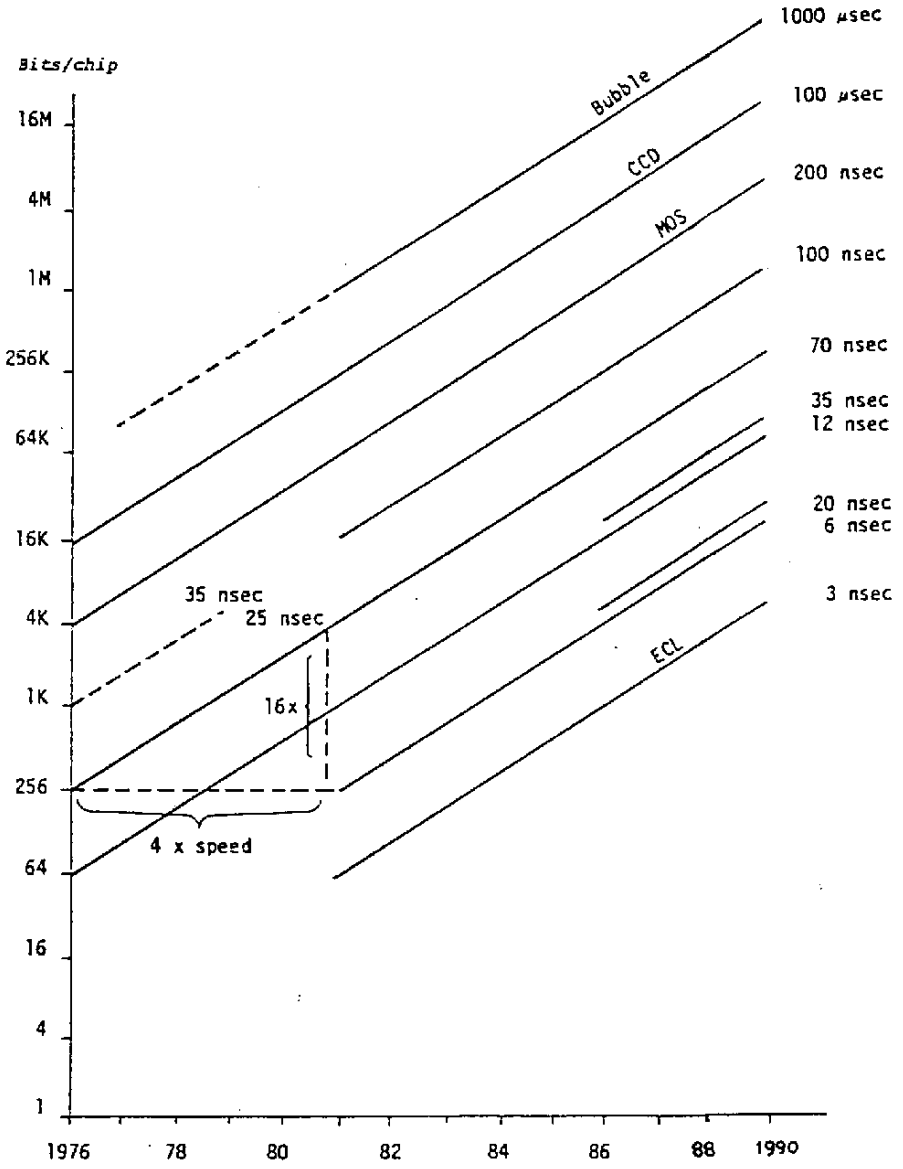


図4.1.1　半導体メモリの集積度推移[1)]

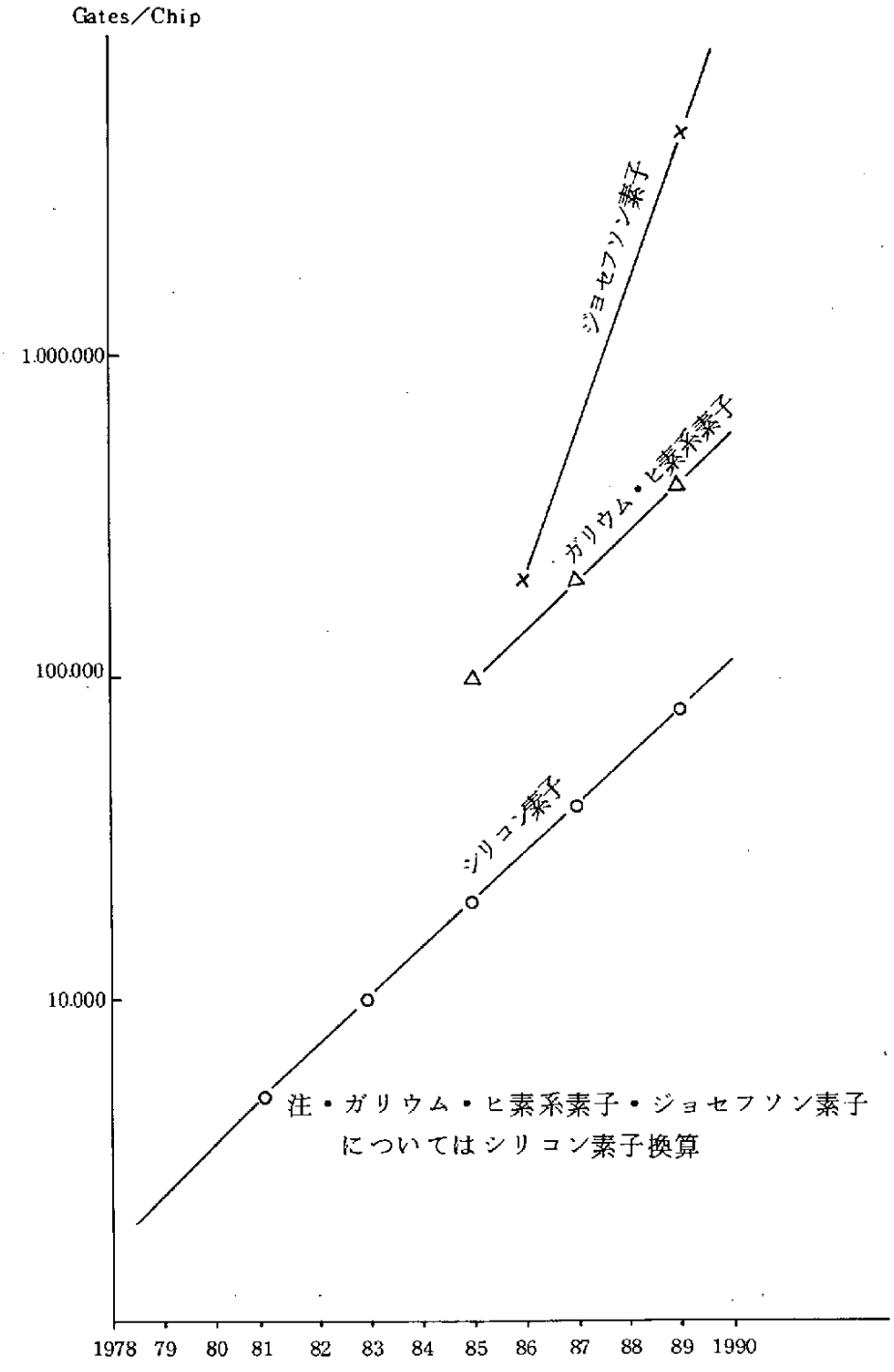


図 4.1.2　論理素子ゲート数の推移

図4.1.2にチップ当たりの論理素子ゲート数の推移を示す。この場合も，メモリ素子と同じく，2～3年で2倍程度の伸びが予想され，さらに1980年代後半には，ガリウムひ素系素子，ジョセフソン素子の実用化を予測するむきもある。

このように時々刻々激しい進歩を示す各半導体素子技術を，実験機，プロトタイプマシンに，どの時点で，どのような形で，どの様な位置付けで適用するかは，更に詳細に検討した上で決められるべきであり，今後の大きな研究課題であるといえる。

データベースオリエンテッドな新たな機能，例えば半導体ディスクを構成するメモリ素子に連想機能をも付加した形のロジックインメモリ素子，あるいはBORAM（Block Oriented Random Access Memory）などのデータベースマシン専用の記憶素子の新たな開発も検討されるべきであろう。

B．磁気記録メモリ

磁気記録メモリの中心的存在である磁気ディスクは，今計画の実験機，プロトタイプマシンの両システムにおいても，その機械的動作が伴なうことに起因するアクセス性能の低さを補うための，半導体ディスクと共に使用される可能性が大きく，大容量，低価格ゆえに両システムにおいても，記録システムの中核をになうことになるであろう。

図4.1.3に磁気ディスク装置における記録密度の推移を示す。この図からもわかるように，1970年代以降，記録密度の伸びは，2～3年で約2倍のペースを堅持しており，1980年代を通してこの傾向は続くものと予想されている。

図4.1.4に磁気ディスクのデータ転送レートの推移を示す。

磁気ディスクのアクセス性能については，機械的動作が伴なうため，著しい改善は期待できないものと予想されている。

一般的にいって，磁気ディスクは，薄膜あるいは垂直磁化技術によって記録密度の向上が期待され，大容量性，コストの面から，1980年代においても，記憶装置の中心的位置を占めていよう。

データベースマシンにとってみると，ディスクキャッシュあるいは更に大容量なMSSとの記憶階層の中における位置付け，データベースマシンオリエンテッドな機能の付加等に研究の課題があるといえよう。

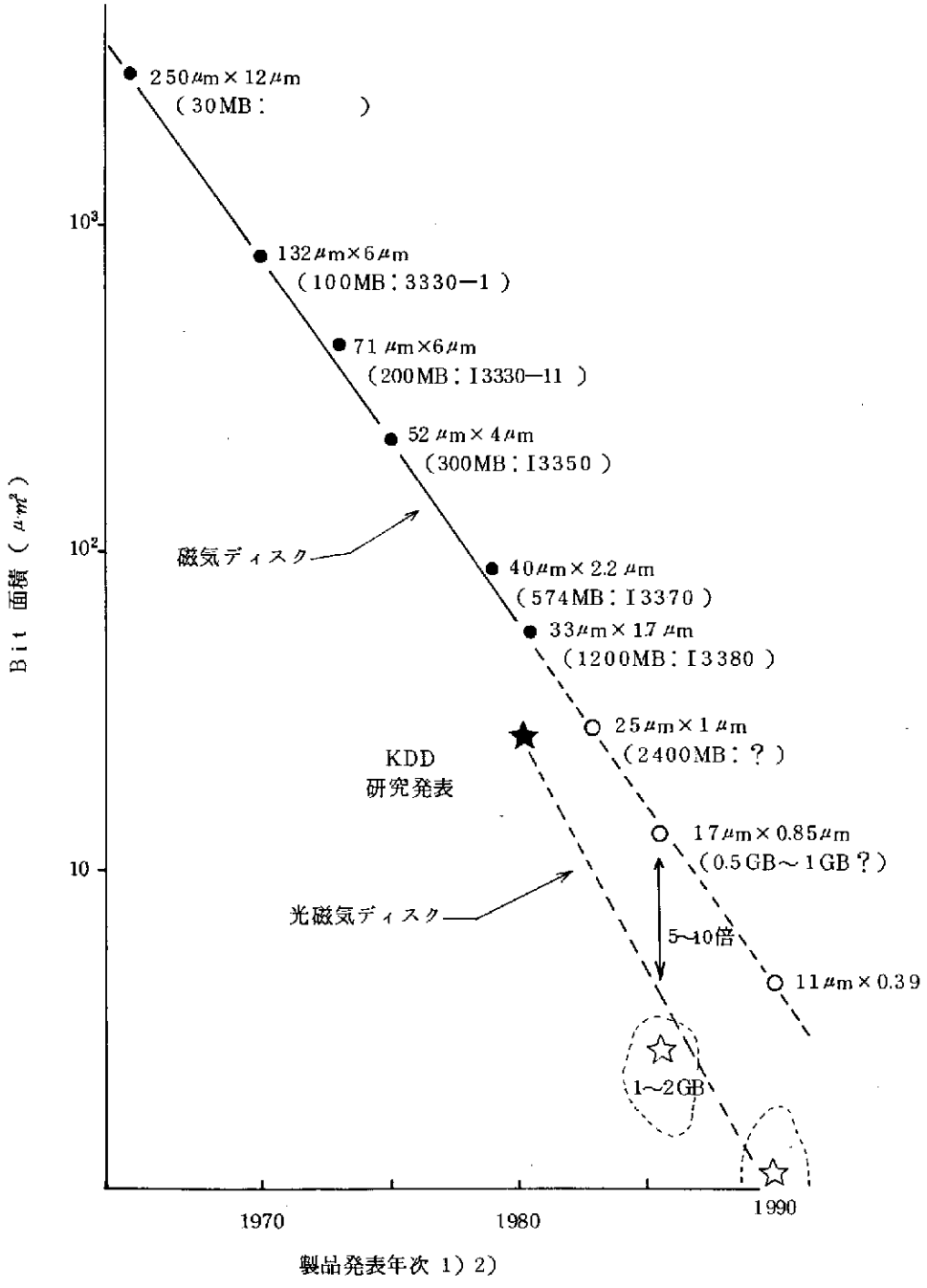


図4.1.3　Bit面積（μm^2）

光磁気ディスク／磁気ディスク

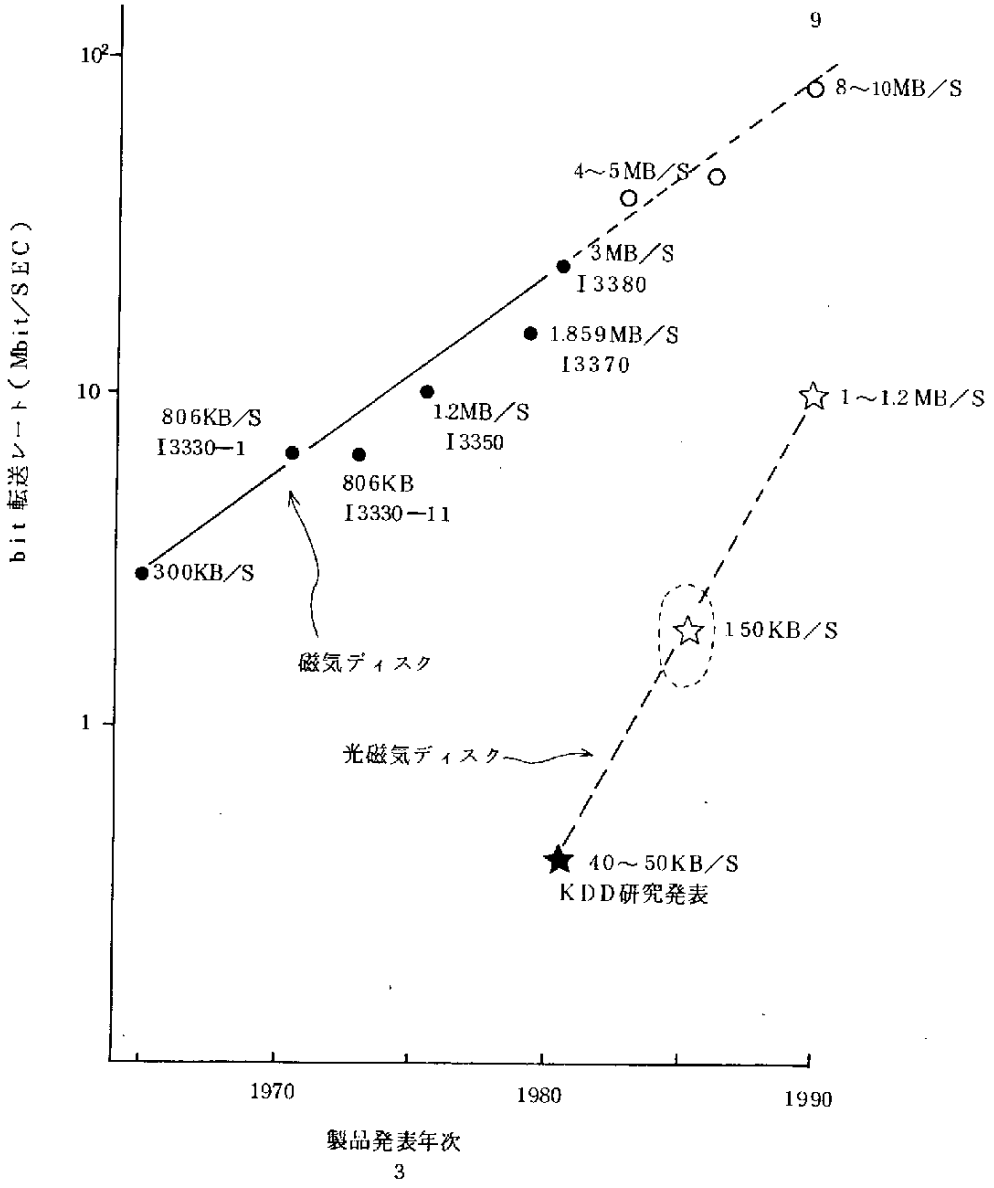


図 4.1.4　転送レート（Mbit／SEC）

光磁気ディスク／磁気ディスク

C．光磁気ディスク[2),3)]

最近，製品化が発表された光ディスクは書き替えが不可能で，信頼性も劣ることからアナログ情報のデータバンク的利用が考えられている。しかし１９８０年代後半にも実用化が期待されている光磁気ディスクは，書き替えも可能となり，信頼性も高いことから，大容量外部メモリとして用いられる可能性がある。図4.1.3と図4.1.4に磁気ディスクの性能に比較した光磁気ディスクの性能を示す。

データベースマシンにとっても光磁気ディスクの大容量性は魅力的であり，その利用が積極的に検討されてしかるべきであろう。

4.1.2 大容量化技術

A．ディスク・キャッシュ

コンピュータシステムにおける磁気ディスクなどの２次記憶のアクセス時間と，主記憶へのアクセス時間には10^5倍以上もの大きな隔りがある。この２次記憶におけるアクセス時間の遅さは，主としてヘッドの移動時間あるいは回転体待ち時間という機械的動作に起因するもので，従来からこのアクセス性能の低さを補うべく，ハードウェア，ソフトウェア両面から種々アーキテクチャが提案，実施されてきた。例えば，

(1) セクタ方式

(2) ブロックマルチプレキシング転送

(3) マルチヘッド

等にその例を求めることができる。

しかしこれらのアプローチも，ヘッド移動時間，回転体待ち時間に対する部分的改善ではあるものの，本質的にアクセス時間に関する主記憶との大きなギャップを埋めるには到っていない。

現実的な問題として，磁気ディスクなどの２次記憶に対するアクセス性能の改善度が，処理装置本体の性能改善度に及ばず，システムとして，主記憶を含む演算処理装置と２次記憶との間の性能バランスが壊れ，大型システムにおいては演算処理装置の実行時間の大部分を２次記憶アクセスのために費いし，演算処理装置の演算性能を十分発揮させ得ないシステムが多くなってきている。

この問題の根本的な解決策として，機械的アクセスを伴わないＲＡＭ，ＣＣＤあるいは磁気バブルのような記憶素子の利用が考えられている。具体的には

(1) 固体ディスク

(2) ディスク・キャッシュ

等のアーキテクチャが提唱されており，一部実用の段階に入っている例もある。しかし，本来，外部記憶に対しては，大容量かつ低価格でなければならないという宿命的条件が課せられており，現状では，ＲＡＭ，ＣＣＤ，磁気バブルなどの割高な素子を使用する固体ディスクの全面的使用は，経済的に困難であるといわざるをえない。

一方，ディスク・キャッシュは今後ますます大容量化と高速化が要求される磁気ディスクに対し，比較的小容量のＲＡＭなどの使用により，安価に磁気ディスクの実効的性能を向上させることができ，今後非常に有効なアーキテクチャとして脚光を浴びてくる可能性が大きいと思われる。

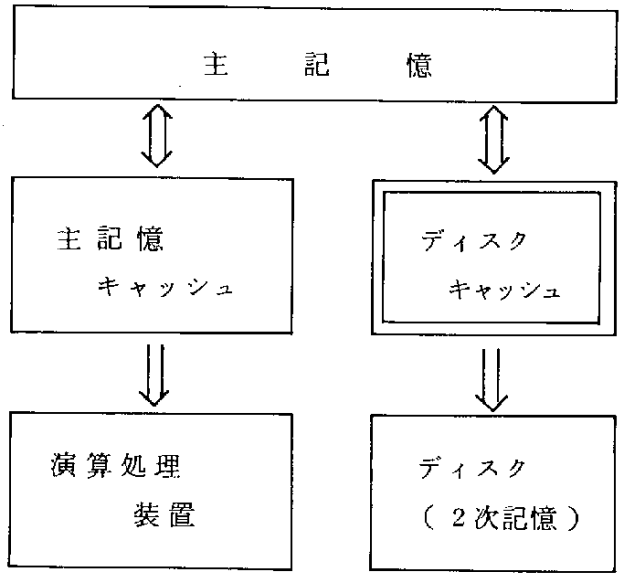


図4.1.5　ディスクキャッシュの位置づけ

ディスクキャッシュは，主記憶と演算処理装置の間に位置する主記憶キャッシュの概念を，主記憶とディスクとの間に適用したもので，アクセス頻度の高い磁気ディスク上のデータを，高速アクセスが可能なディスク・キャッシュ上に移しておき，プログラムからの要求に対して実効的なアクセス時間を大巾に短縮し，システム全体の効率を上げようとするものである。

ディスクキャッシュに適用するアーキテクチャも，

・ＬＲＵ　（Least Recentry Used）方式

・First In First Out 方式

など種々考えられるが，一定時間帯で見ると，アクセスの傾向が，一定記憶領域に集中する局在性の有無などファイルの構成，ファイルのアクセス方法の特性など十分検討された上で決定されるべきであろう。

現在までに，ＡＣＯＳ１０００などの装置にディスク・キャッシュの適用例が報告されている。

ＡＣＯＳ１０００の場合のディスク・キャッシュについて，その概略を示す。

- 使用記憶素子 ： ６４Ｋビットメモリ素子
- メモリ容量 ： 最大３２Ｍバイト
- データ転送レート ： 4.8Ｍバイト／秒
- データ転送開始までの時間 ： 1ms以下
- リプレースメントアルゴリズム ： ＬＲＵ方式

また方式上の特長として，このディスク・キャッシュは，中央処理装置内に接続されておりシステム内の全てのディスクで共通に利用できる形態を採用している。

B．連想ディスクコントローラ

連想ディスクコントローラの基本的な考え方は，連想プロセッサ，連想メモリの概念が出発点となっている。

連想プロセッサを定義づけるならば

- 格納されているデータがそのロケーションアドレスではなく，その内容または内容の一部によって検索できる。
- 多数の変数集合に対する，算術的，論理的演算を１つの命令で行うことができる。

の２つの特性を持つプロセッサであるということができる。

連想プロセッサの使用が比較的効果があると考えられる用途として，

- 変化の早いデータベースの検索，ストア。
- 大規模データベースの高速サーチ。
- 多量のデータに対する，算術的，論理的演算。

などがあげられる。

連想ディスクコントローラとは，内容呼び出し（Content Addressing）など，データベース用のハードウェアを備えたインテリジェント・ファイル・コントローラの一種と考えることができる。

従来のデータベースは全てホストＣＰＵ上でソフトウェアによって形成され，実行する形式が

採用されていたが，データベースが取扱うべきデータ量の膨張に伴って，データベースソフトウェアは膨大かつ複雑なものになってきた。

そこでデータベース処理を他のデータベース専用マシンに譲りホストの処理負担を軽減しようとする機能分散の概念を具現化したものがデータベースマシンである。この考えの延長上にあって，大容量データファイルである磁気ディスクに近いディスクコントローラで，データベース処理をやってしまい，ホストＣＰＵとの間のデータ転送量をできるだけ減らそうとするのが，連想ディスクコントローラであり，従来のディスクコントローラに検索機能などのハードウェアを付加したものである。このようなインテリジェント・ディスクはこれまでＣＡＦＳ（Content Addressable File System）[4]やＩＦＣ（Intelligent File Controller）[5]などが報告されており，これらのインテリジェント・ディスクに共通する点は，従来のディスクコントローラに検索機能，サーチ機能などの簡単なハードウェアを付加するだけで容易に実現でき，しかも効果も比較的大きいのが共通した特長となっており，データベースマシンとしての１つの方向を示唆していると思われる。

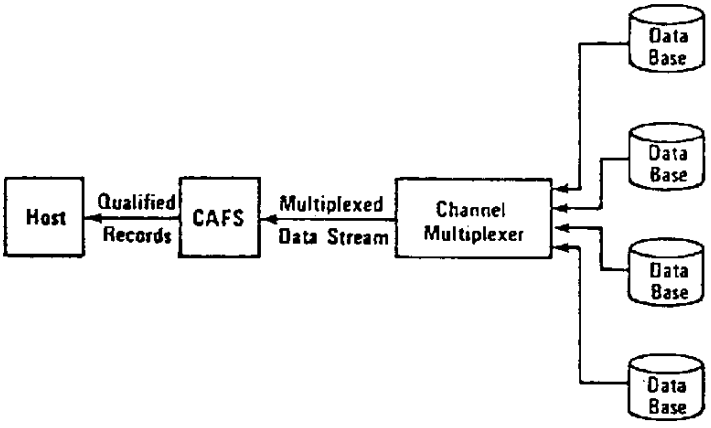


図4.1.6(a) ＣＡＦＳ　ＳＹＳＴＥＭ

インテリジェント・ディスク・コントローラの動作概念はおおむね次のようなものである。

まず最初にＣＰＵより検索条件がコントローラに知らされ，その後順次ディスク装置より読み出されてくるデータを検索の条件と逐次比較し，検索条件に合致したデータのみを抽出しＣＰＵ側に送る。

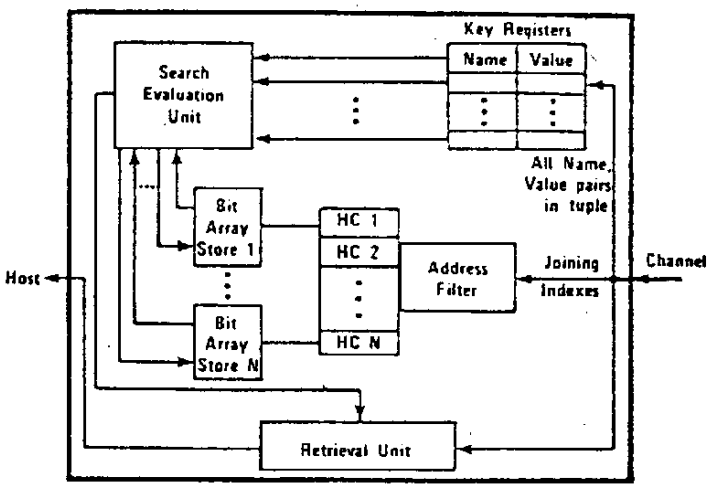


図4.1.6(b) ＣＡＦＳ　アーキテクチャ

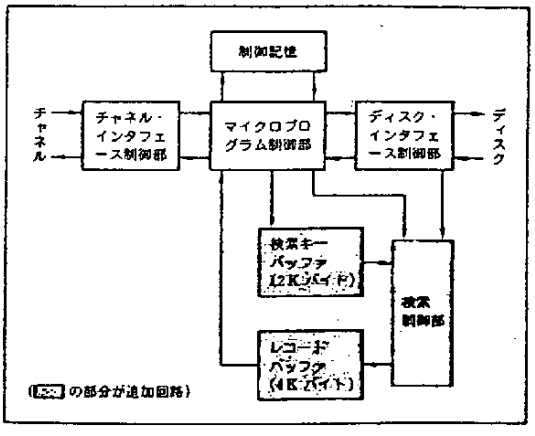


図4.1.7(a) ＩＦＣの 構 成

図4.1.6にＣＡＦＳ，図4.1.7にＩＦＣについてその実際例を示す。

ＣＡＦＳは，通常のディスク制御装置に，連想サーチ機能，レコード検索機能を付加したものであり，連想サーチ部は，ディスク装置より送られてくるデータに対して検索条件を適用する部分で，レコード検索部は，条件を満たしたデータのうち指定されたデータを抜き出し，ＣＰＵへ送る。必要ならばここでデータに対し一定の加工を加えた上でＣＰＵへ送ることもできる。

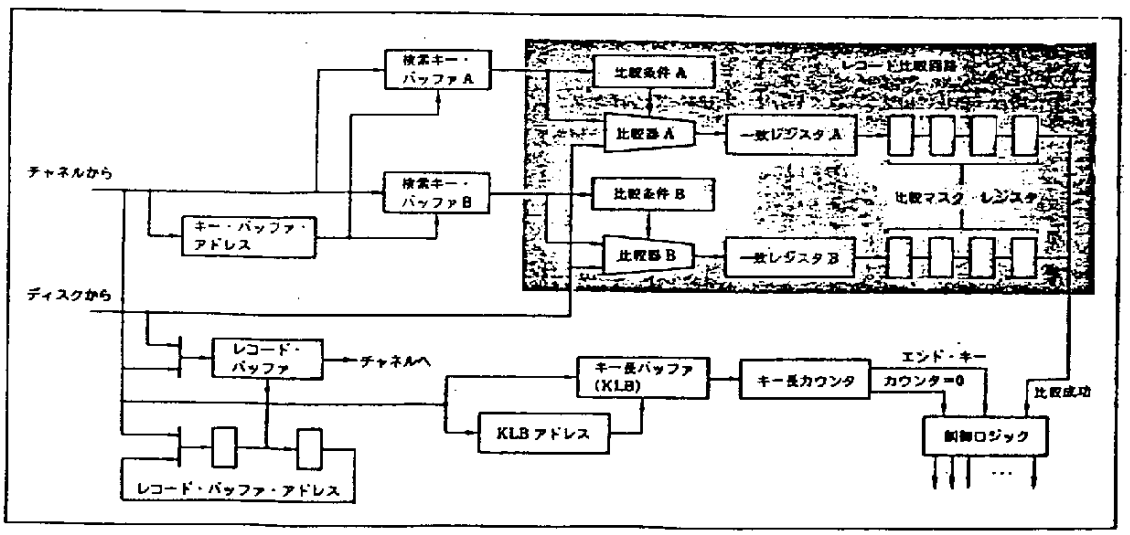


図4.1.7(b)　ＩＦＣ　アーキテクチャ

ＩＦＣは汎用の磁気ディスク制御装置に比較的小規模なハードウェアからなる内容による検索機能を付加したもので，その基本構成はＣＡＦＳのそれと大きな相異点は見られない。このような小規模なハードウェアの付加のみで，ファイルの応答時間は，1/4.5～1/13に減り，ＣＰＵ時間は1/11～1/45に減ったことが報告されている。

4.1.3　大容量化技術の実際例

大容量ファイルに対する高速連想アクセスが可能な記憶システムは，データベースにとって不可欠の要素であり，ＲＡＭ，磁気バブル，ＣＣＤあるいは磁気ディスクなどの記憶素子のそれぞれの特長を生かす形で多くのデータベースマシンが発表されている。

しかしいずれのシステムをとってみても，容量，アクセス時間の点で十分であるとはいえず，さらに大容量，高速アクセスが可能な記憶素子の開発が望まれている。

データベースシステムによる記憶機能の大容量化，アクセスの高速化の要求に対する1つの解決策として記憶機能の階層化という考え方がある。

小容量だがアクセス速度の速い記憶素子と，大容量だがアクセス速度の遅い記憶素子を，それぞれの記憶素子の欠点を補い合う形で階層状に配置し，システムとして実効的に，大容量かつ高速アクセスが可能な記憶システムである。

記憶階層システムの実際例について説明する。

図4.1.8は，DBC[6]の構造を示したものであり，データベースに関する構造情報を扱う構造ループとで構成され，ホストとのインタフェース制御やシステム全体の制御などをDBCCPが行う。

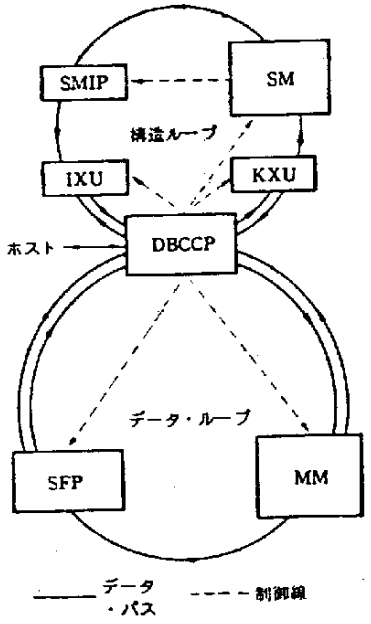


図4.1.8 DBCの構造

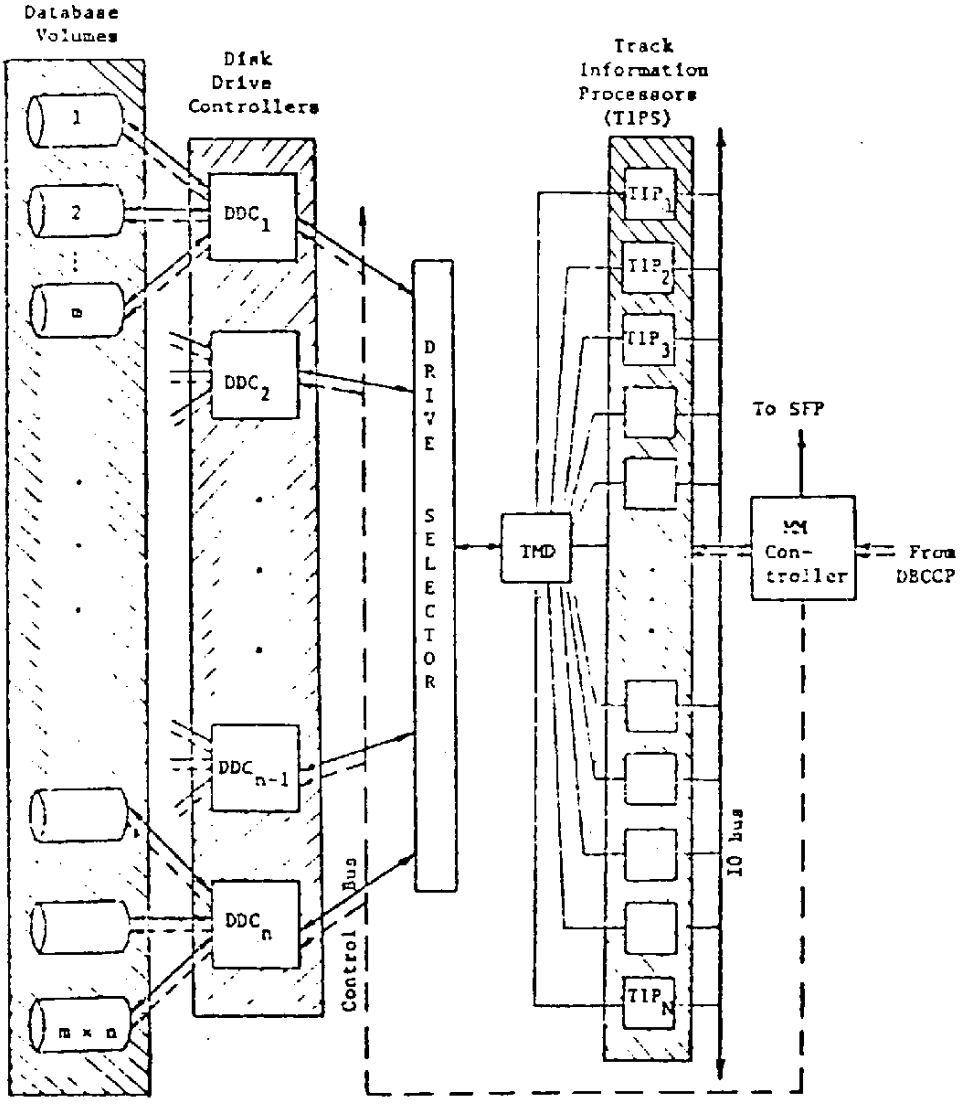


図4.1.9　MM（Mass Memory）の構造

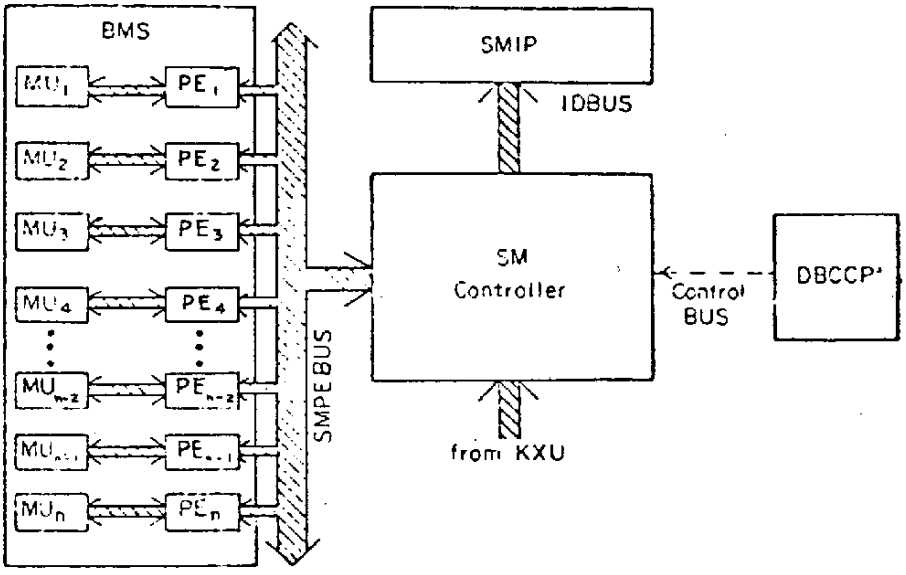


図4.1.10　SM（Structure Memory）の構造

図4.1.9に示すMMはディスクで構成され，10^{10}バイト以上の記憶機能を有する。図4.1.10に示すSMは，CCD，磁気バブルなどで構成される10^{7}～10^{9}バイト程度の記憶機能であり，構造情報の検索と更新を行う。MMとSMの2つの記憶階層を有するのがこのDBCの特徴といえる。

INFOPLEXは，100Gバイト程度の超大型データ・ベースを対象とし，システムを構成する各レベルメモリを，表4.1.1に示すように高速小容量（100ns，32Kバイト）から低速大容量（1秒，100Gバイト）まで6階層に分化し，各階層間でデータのステージングを行うというマシンである。

表4.1.1　INFOPLEXにおける記憶の階層化

Storage Level	Random Access Time	Sequential Transfer Rate (bytes/sec)	Unit Capacity (bytes)	System Price (per byte)	Technology
1. Cache	100 ns	100M	32K	50c	Bipolar LSI random-access memory
2. Main	1 μs	16M	512K	10c	Metal oxide semiconductor LSI random-access memory, ferrite core
3. Block	100 μs	8M	2M	2¢	LSI shift registers, bulk ferrite core, charge-coupled devices, magnetic bubbles
4. Backing	2 ms	2M	10M	0.5c	Fixed-head magnetic disk and drums, charge-coupled devices, magnetic bubbles, electron beam addressed
5. Secondary	25 ms	1M	100M	0.02c	Moving-head disks
6. Mass	1 sec.	1M	100B	0.0005c	Automated tape handlers, optical (laser) beam

4.1.4　モジュール間の各種結合方式とその評価

本プロジェクトにおいて開発の目標となる高性能な関係データベースマシンシステムのシステム形態として，複数のプロセッサ，機能モジュール，記憶モジュールが結合された形の複合システムが想定されている。

このような複合システムにおいてはそれぞれのプロセッサ，モジュール個々の性能が，システム全体の処理性能を大きく左右することはいうまでもないが，処理性能を決めるもう1つの大きな要素としてプロセッサ，モジュール間を結ぶ結合方式と結合インタフェースのデータ転送性能をあげねばならない。

結合インタフェースのデータ転送性能がシステム全体としてのボトルネックにならないような結合方式を検討することが必要となろう。

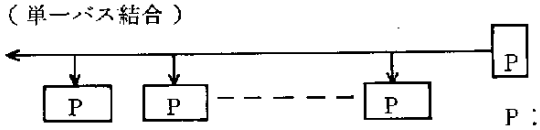
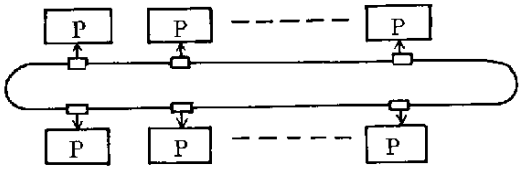
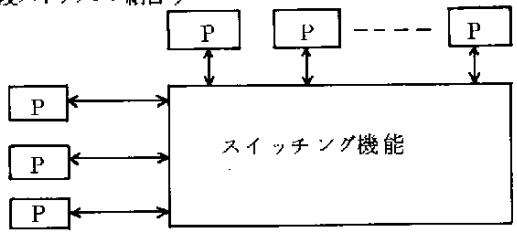
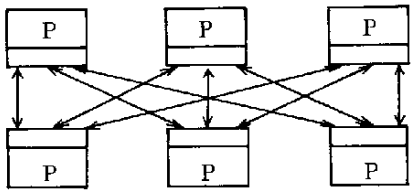
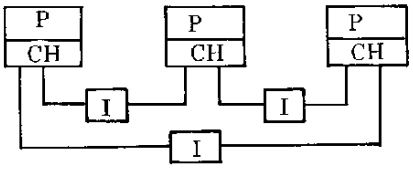
ここでは，各プロセッサ，各モジュールは比較的近距離な位置に配置されると想定した上で，このような条件下での一般的に考えられる結合技術[7]について述べる。

一般的に，1つのシステムの中で複数のプロセッサ,モジュールを分散して配置する複合システムの目的は，どういう機能をどのように分散するかによって，

(1)　それぞれの処理に適した機能を専用に備えた専用プロセッサによる機能分散。

(2)　同一プロセッサを多数使用することによって総合的に高い処理性能を得ることができる負

表4.1.2　プロセッサ間結合方式の形態による分類

分　類	形　態
共通バス結合	（単一バス結合） 
	（リングバス結合） 
	（多段スイッチング結合） 
マルチポート結合	
チャネル結合	

荷分散。

(3) 予備のプロセッサを備えシステムとしての高信頼性を目指す危険分散。

の3項目に分類することができ，本プロジェクトが目標とする関係データベースマシンシステムは，(1)の考え方に基礎を置いたものと考えることができる。

このように分散されたプロセッサ，モジュール間を結合する方式として，表4.1.2　結合方式による分類に示すように，3つの方式に分類することができる。

A．共通バス結合方式

一般的に共通バス結合方式は，共通のバスに複数のプロセッサ（モジュール）を接続した結合方式で，他の2つの結合方式に比べ最も単純で経済的な結合方式であるといえるが，一方欠点も多く複数のプロセッサによるバス専有要求に対する競合性，信頼性，データ転送性能などに問題がある。しかしこれらの問題点に対しては，例えばバスの多重化，バス幅の拡大などによる解決策が種々考えられその結合インタフェースに要求される性能を引き出すことは比較的要易と思われる。拡張性については結合手順のやり方に大きく左右されるが，一般的には他の3つの方式に比べより柔軟性に富むということがいえる。

共通バス結合方式は，その接続形態からみて，

(1) 単一バス結合方式

(2) リングバス結合方式

(3) 多段スイッチング結合方式

の3つに分類することができる。

(1)の単一バス結合方式は最も単純な結合方式であるといえるが，その典型的な例としてＩＯインタフェースにおけるいもづる接続方式をあげることができる。この方式の特徴は本体処理装置と任意のＩＯ間では双方向のデータ交換が可能であるが，ＩＯ間ではデータの交換ができないという制約が伴うという側面をもつ。

例として複数のプロセッサをバスで接続したＲＡＰ.2 システム[8]の例を図4.1.11に示す。

(2)のリングバス結合方式はノード数が非常に多いシステムにとっては有効な方式であり，拡張性にも優れ任意のノード間での任意方向のデータ交換が可能であり，他の結合方式に比べて最も柔軟性のある結合方式であるといえよう。またリングバス上の障害に対しても障害ノード部のバイパス機能，交替リングバス機能，ループバック機能などの方式を導入することにより高い信頼性を確保することも可能である。

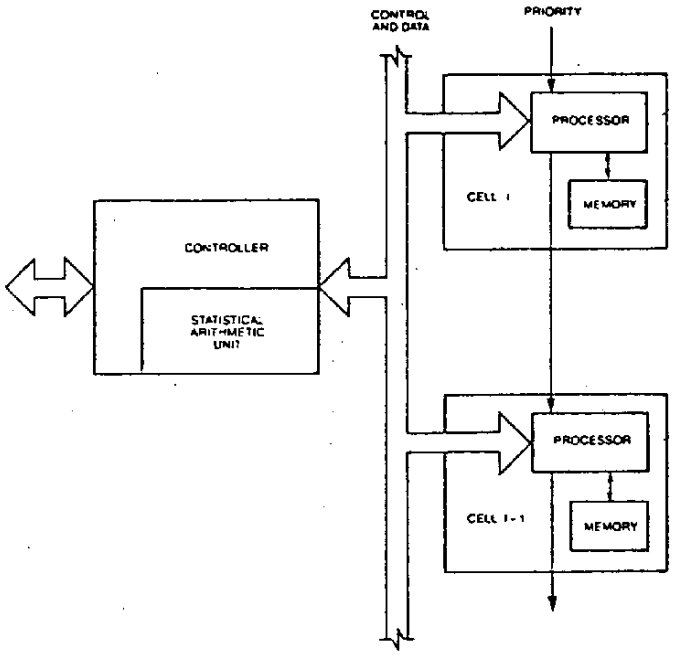


図 4.1.1 1　RAP.2 system architecture.

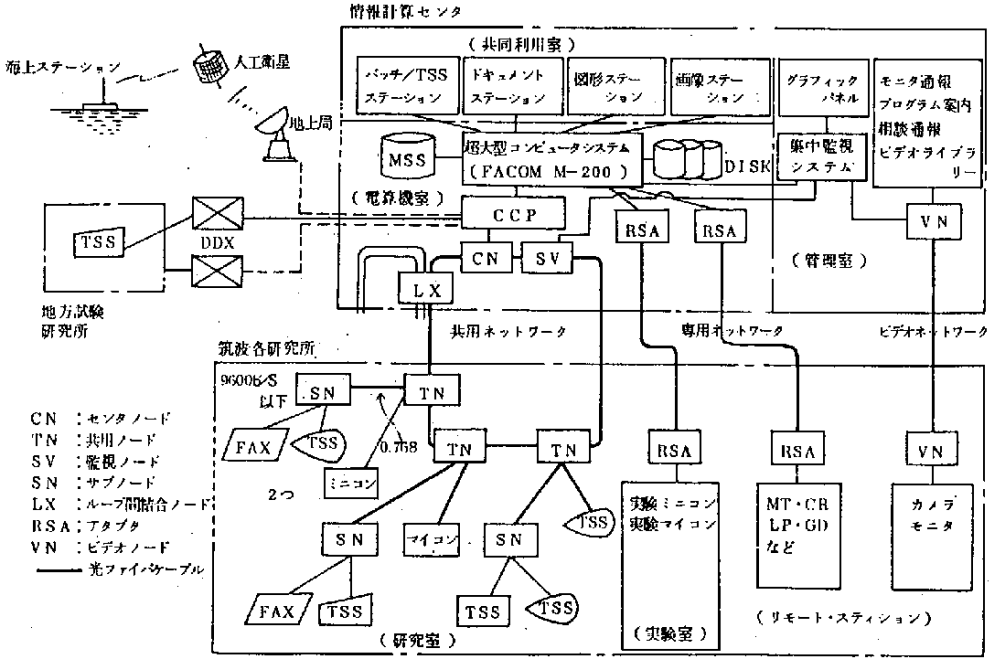


図 4.1.1 2　複合分散型研究情報システム（ＲＩＰＳ－ネット）の構成

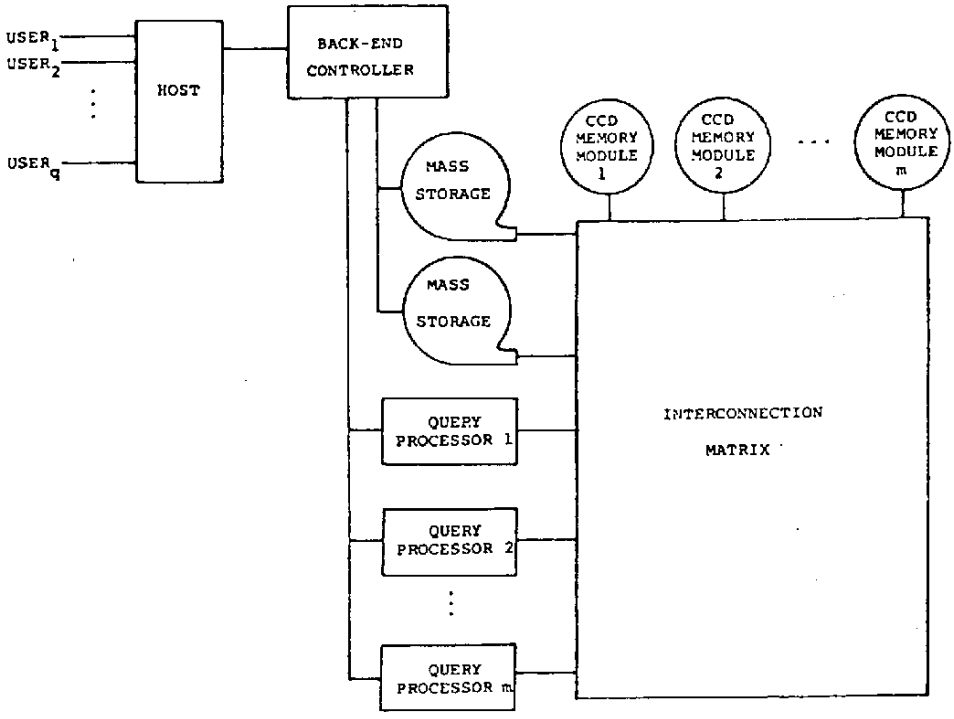


図4.1.13　DIRECT system architecture.

図4.1.12にＲＩＰＳ[9]におけるリングバス結合の実施例を示す。

(3)の多段スイッチング結合方式はデータ交換パスのスイッチング機能が必要となり，データ交換が必要なプロセッサの数が非常に多いシステムにとってはハードウェア的にも経済的にも実現が困難になってくる。しかし一方で同時に複数の交換パスを提供し得るという多重処理性能の面では他の方式に比較し最も優れた方式であるといえよう。

図4.1.13にＤＩＲＥＣＴシステムにおける多段スイッチング結合方式の実施例を示す。

B　マルチポート型結合方式[10]

マルチポート型結合方式はそれぞれのプロセッサにデータ交換が必要な相手プロセッサの数だけ入出力ポートを装備させるもので，従来から最も一般的に行われてきた方式であり，接続可能台数がポート数で制限されるため拡張性に問題があり，ポート数を多く必要とするシステムにとっては大きな問題である。しかし，データ交換手順が他の方式に比べ単純であること，データ交換における同時処理性にも優れていることからデータ交換性能は最もよいとされている。

C　チャネル結合方式

それぞれのプロセッサのチャネルを介してプロセッサ間のデータ交換を行う結合方式であり，

互いに相手プロセッサを一種の入出力装置とみている点に特徴がある。この方式では一般的にCHとCHの間にインターフェース変換装置が必要であり，データ交換のための手順がソフトウェアも含め非常に複雑である。しかし相手プロセッサを入出力装置とみなすため，インタフェース変換機能を付加すれば比較的容易に拡張できるという利点がある。

ー参考文献ー

1) "The Fifth Generation"
INFOTECH STATE OF THE ART REPORT SERIES 9 №1

2) Nobutake Inamura "Magnetic and Magneto-Optic Properties of Amorphous Gd-Tb-Fe Film for Magneto-Optical Disk Memory"
Third International Conference on Ferrite.

3) 角田, "新コンピュータ時代を支える大容量メモリ"
日本の科学と技術 '81／レーザーⅡ

4) D.K. Hsiao, S. E. Madnick
"Data Base Machine Architecture in the Context of Information Technology Evaluation"
Proc. VLDB 1977 PP63～84

5) 石塚, 宮寺, 高橋 "インテリジェントファイル制御機構の実験システムについて"
情報処理学会 計算機アーキテクチャ研究会資料 39 1984. 9. 17

6) Olin. Bray, Harvay A. Freeman "Disk Base Computer"
Lexirgton Books Series in Computer Science Lexington Books Massachusetts Toront 1979.

7) 生田, 武川 "最近の大型ミニコンとその応用(5) 大型ミニコンによる複合計算機システム"
処報処理 Vol 23 1982 №1

8) Stewart A. Schuster, Hob. Nguyen, Eseo A. Ozarahan, and Kenneth C. Smith "RAP. 2—An Association Processor for Data base and its Applications."

9) 矢田, 本田, 他 "RIPSーネットー研究情報システムー"
情報処理学会 分散処理システム研究会資料 10 1981. 9. 3

10) Philip H. Enslow JR. "Multiprocessor Oranization - A Survey"
Computing Surveys, Vol. 9, No. 1, March 1977

4.2 アーキテクチャ

4.2.1 並列処理，パイプライン処理技術

A．データベースマシンと並列処理

データベースマシンにおける並列処理には

(1) ホストコンピュータとデータベースマシンの間の並列処理

(2) 検索要求の解析と処理の間の並列処理

(3) データのアクセスと処理の間の並列処理

(4) データの転送と処理の間の並列処理

(5) データの並列アクセスと並列転送

(6) データの並列処理

等がある。(1)～(3)は機能分散型の並列処理である。(1)～(5)の各々に，処理対象を独立な部分に分割して，各々独立に処理する同時処理型の並列処理と，一連の処理の流れを，いくつかのステージに分割し，ステージ毎に処理機能を割り当てて流れ作業をするパイプライン処理型の並列処理とがある。

B．ホストとデータベースマシンの間の並列処理

計算機システムから，データベース処理機能だけを独立させ，バックエンドプロセッサに処理させるという考え方は，ＣＣＡ社のData Computerや，ソフトウェアＡＧ社のADABAS マシン等の商用機で，その効果が既に実証されている。これらのシステムでは，汎用計算機をバックエンドマシンとして用い，ＤＢＭＳを常駐させ，ＤＢＭＳへのコマンドのレベルをホストコンピュータとのインタフェースとしている。データベースマシンをこのようなソフトウェアバックエンドマシンではなく，専用ハードウェアによる並列処理型のマシンとして実現する場合にも，ホストコンピュータとデータベースマシンの間の機能分担や，これらの間のインタフェースに関しては，ソフトウェアバックエンドマシンと異ならない。

C．検索要求の解析と処理との並列処理

検索要求言語の構文解析，処理のスケジュールの最適化等と，データベース処理とは独立に並列処理することができる。検索要求の解析を行うサブシステムは検索要求解析サブシステムと呼ばれる。

検索要求解析サブシステムは，エンドユーザ言語の解析，関係代数言語の解析等を行い，データベースマシンの各種処理モジュールに最適なスケジュールでもって，処理の制御コマンドを発する。関係がセグメント化されていたり，転置ファイルのような補助ファイルが存在する場合に

は，どの関係がどのような形でセグメント化され，どのファイル装置に格納されているかというような情報をデータディクショナリ，データディレクトリ（ＤＤ／Ｄ）として管理し，関係に対する処理要求をセグメント・ファイルに対する処理コマンドの系列に変更することも，検索要求解析サブシステムの仕事である。したがって，検索要求解析サブシステム自体が機能分散型の並列処理システムとして実現されることもあり得る。

D．データのアクセスと処理との並列処理

ディスクキャッシュや大容量のバッファメモリを用い，ディスクの先行アクセス等のスケジューリングによって，ファイルの２次記憶装置からの読み出しや，２次記憶装置への書き込みと，バッファ内のデータの高速処理との並列処理が可能である。Bray の分類によるところのSPISやMPＩS型のデータベースマシンで用いられている手法である。各２次記憶装置毎に専用バッファを用意する方法と，多バンク構成のバッファを用意するが，各バッファと２次記憶装置との結合関係はスイッチング・ネットワークを用いて自由に設定できるようにする方法とがある。前者では，ダブルバッファリングの手法が通常用いられる。前者はセレクションやリストリクションの処理に用いられ，後者は，ジョイン等の交差演算をこのバッファ上で処理する場合に用いられる方法である。

E．データの転送と処理との並列処理

データベースマシンでは，大量データの転送が，ホストコンピュータとデータベースマシン，２次記憶装置と処理系，処理系内部の各種機能モジュールの間に生じる。この転送時間は無視できない。そこで，データの転送経路に処理モジュールを置いて，データがこのモジュールを通過する間に種々の処理を実行してしまおうとする試みがいくつか提案されている。

１つは，２次記憶装置から処理系へのデータ転送途中で，セレクション，リストリクション，セミジョイン，プロジェクション等の処理を施してしまおうとする考えで，ＩＣＬ社のCAFSにその例を見ることができる。ＲＡＰのように，ＣＣＤや磁気バブルのような固体ディスクと，シフトレジスタを用いた比較的小容量のバッファを用い，固体ディスク中のデータが，ディスクの回転に同期して，バッファに読み出され，バッファ上でシフトされ，このバッファを通過して再び固体ディスクへと戻る間にセレクション，リストリクション，セミジョイン，プロジェクション等を処理する試みもある。これを on - the - fly 処理と呼ぶ。

他のタイプは，処理系の中の各種機能モジュール間の転送と，機能モジュールによる処理とを重畳させようという試みである。北海道大学が開発したサーチエンジン，ソートエンジンの他，東京大学，京都大学，広島大学等でこの種のソータの研究がなされている。CMUのKung の提唱するSystollic　Array 処理は，もともとは，ＡＬＵのような処理モジュールを繰り返し構

造を持つように並べ，メモリから読み出されたデータをメモリに戻す前に，できるだけ加工しようとの考えから提案されたものであるが，その効果として，転送と処理を重畳することができる。

データの転送に重畳して処理を行うことを本報告書ではデータストリーム処理〔TANA80〕と呼んでいる。

F．データの並列アクセスと並列転送

データベースマシンの処理性能を決める因子の1つは，2次記憶装置のアクセス時間と，その転送速度である。これらの因子をよくするために，複数ボリュウムの並列アクセス，並列転送が用いられる他，個々のボリュウムのアクセス時間を短縮するために，トラック毎にヘッドを設ける等の多重ヘッドのディスクの提案や，転送速度を向上するために，シリンダ内の全トラックの並列読み出しの提案がなされている。シリンダ内トラックの並列読み出しは検討の余地があるが，多重ヘッドディスクは，コストの面で普及が疑わしい。

磁気ディスク装置における上述のような改良が高価なものになることから，CCDや磁気バブル等の固体ディスクの採用が一時盛んに検討されたが，MOSメモリの急速な大容量化と低廉化により，CCDや磁気バブルのMOSメモリに対する競争力が弱まり，2次記憶メモリや，低速のバッファメモリへ，これらのデバイスを採用することのメリットが弱くなりつつあるとの見方もある。

このようなことから，並列アクセス，並列転送に関しては，ボリュウム間の並列度を上げることによる効果に検討を絞ってよいと考える。検討項目には，関係データ等のファイルの最適なセグメンテーション等の問題も含まれる。

G．データの並列処理

RAPに代表される初期のデータベースマシンでは，各関係のタプルの集合をいくつかに分割し，小さくなった部分関係を独立のディスクに格納し，これらを並列アクセス，並列転送し，部分関係毎に独立のプロセッサでセレクション，リストリクション，セミジョイン，プロジェクションを処理することにより，プロセッサの多重度分の並列処理によって処理速度を向上することを目指していた。

現在は，処理速度向上の努力はジョイン演算の高速化に向けられている。ジョイン演算の高速化には，ソート処理が有効であり，O（n log n）の時間複雑度の最適なソートアルゴリズムを，多重度がO（log n）のパイプライン処理で処理することにより，ソートの処理時間をO(n)にしようとする試みが種々発表されている。このようなソート処理を用いると，ジョイン演算もO(n)で処理できる〔TANA80〕。

ところが，関係が大きく，これをセグメントに分割する場合には，セグメントとセグメントの

ジョインはO(n)で処理できたとしても，ジョインすべきセグメントの対は，２つの関係のタプル数をn，m，セグメントの大きさをBとするとき，（n×m）／B対あり，ジョインに要する時間はO（n×m）となる。そこで大容量のステージングバッファを用い，各関係を，ジョイン属性の値の区間分けによって，動的にセグメントに分ける手法が武蔵野通信研究所や東京大学〔KIZU81〕によって提案されている。これを動的クラスタリングと呼んでいる。これに対し，ソート処理と動的クラスタリングを融合した機能を持つネットワークが北海道大学より提案されている〔TANA82〕。

―参考文献―

1）〔KIZU81〕 喜連川優他：可変構造多重処理データベースマシンにおけるHash の適用，情報処理学会第２３回全国大会予稿集，pp.561－562，1981

2）〔TANA80〕 Y. Tanaka, Y. Nozaka, and A. Masuyama : Pipeline Searching and Sorting Modules as Components of a Data Flow Database Computer, IFIP Congress 80, 1980.

3）〔TANA82〕 田中譲：データフロー制御データストリーム処理方式データベースマシン（基本構想），情報処理研資，データベース管理システム28－3，1982.

4.2.2 新アーキテクチャ

A．データベースマシンと新アーキテクチャ

７０年代のコンピュータアーキテクチャの研究は，データフローアーキテクチャ，オブジェクト指向アーキテクチャを始め，リダクションマシン，抽象データ型マシン，関数言語型マシンといった新しい概念を育成してきた。さらに，ＶＬＳＩ技術の進歩に伴い，ＶＬＳＩに適したマシンアーキテクチャの研究も進み，高度な並列処理やパイプライン処理を行う専用ＶＬＳＩの研究や，このようにハードウェアによって実現されるアルゴリズムの研究が盛んになっている。

B．データフローアーキテクチャ

ＭＩＴのDennis等が中心となって進められてきたデータフローマシンの研究は，計算アルゴリズムが本来持っている並列処理度をうまく引き出すことと，ＣＰＵとメモリの間に存在するvon Neumanボトルネックの分散による解消を目指したものである。データベースマシンにデータフローアーキテクチャを導入する試みもいくつかある。

１つは，ＭＩＴのDennisのデータフローマシンをジョイン等の関係代数演算の高速処理に応用しようとする試みである〔BORA80〕。この種の演算では，処理に内在する並列性は明確な

形で記述することができ，データフロー制御を行うよりは，前もってスケジューリングされた並列処理や，パイプライン処理を行う方がよいと考えられる。

他の形でのデータフロー制御の導入は，セグメント化されたファイルの先行アクセス，読み出し，処理，書き込みをデータフローで制御しようとする試みである。先に述べたタイプのデータフロー制御の導入が，個々のデータ項目に対するデータフロー制御により，関係代数演算処理の高速化を目指すものであるのに対して，このタイプでは，セグメント化された関係や，転置関係を単位とするデータフロー制御を行い，２次記憶装置から各セグメントを読み出す時間のばらつきを吸収するとともに，先行アクセスによって，等価的にアクセス時間を短縮し，マシンのスループットを上げるだけでなく，応答時間をも改善しようとする試みである。この試みは，北海道大学で研究されている〔ＴＡＮＡ８２〕。今後解決すべき課題は多いが，可動ヘッドディスクを用いる限り，検討すべき試みの１つである。

C．オブジェクト指向アーキテクチャ

ＩＢＭのSystem３８や，インテル社のiAPX４３２等，オブジェクト指向アーキテクチャを持つ商用マシンが現れてきている。オブジェクト指向アーキテクチャのマシンでは，データやプログラム等のデータ集合に対し，これらのデータ型や属性を記述した記述子を管理し，論理的には一様にオブジェクトとして扱い，オブジェクト名を付与して，種々のタイプのオブジェクトを一様に名前で呼ぶことを目指す。

データベースシステムでは，データディクショナリ／ディレクトリ（ＤＤ／Ｄ）が一種のオブジェクト記述子を管理したものと見ることができる。セグメント化された関係や，転置関係もオブジェクトと見做すことができる。関係名で参照される関係に対する処理は，この関係を構成するセグメントのセグメント名を用いて指定される処理へと展開され，さらに，このセグメントを格納する物理記憶媒体へのアクセス，セグメントの転送，処理の各々の指示するコマンドへと展開されねばならない。この過程を，柔軟に実現するためには，オブジェクト指向アーキテクチャの導入が望ましい。

D．データストリームアーキテクチャ

データベース処理では，大量のデータの各種モジュール間での転送が必然的に生じる。これは，他の分野の計算処理と比較した場合，際立った特徴である。このような大量データの転送に要する時間は数ｍｓｅｃから数百ｍｓｅｃ に達することも充分予想される。従来の高速処理用機能モジュールは，主として，機能モジュール内に揃ったオペランドデータに対し，高度な並列処理アルゴリズムを用いて高速の処理を施すことを目指して開発されてきた。これに対し，上述のように機能モジュールへのデータの入力と機能モジュールからのデータの出力のためのデータ転送時間

が無視できない処理では，データの入力転送中とデータの出力転送中に，できる限り多くの処理を重畳させて実行し，転送時間の有効利用を図ることが望ましい。これをデータストリームアーキテクチャと呼んでいる。

—参考文献—

1) 〔BORA80〕 H. Boral and D. J. Dewitt : Design Considerations for Data—Flow Database Machines, ACM SIGMOD'80, pp 94—104, 1980,

2) 〔TANA82〕 田中譲：データフロー制御データストリーム処理方式データベースマシン（基本構想），情報処理研資，データベース管理システム28－3，1982

4.2.3 各種処理技法とその評価

3.4では各マシンについて概説したが，ここでは，データベースマシンに利用される各種処理技法についてまとめる。

初期のデータベースマシンはほとんど全てが，フィルタープロセッサと位置づけられるが，以後ジョインとソートに適したアーキテクチャへと開発の中心が移ってきたように思われる。データベースマシンの代表的な要素技術としては以下の4つがあげられる。

(1) ジョイン

(2) ソート

(3) フィルター

(4) 大容量化技法

A. ジョイン

データベースマシン，特に，関係代数マシンの設計では，ジョインをいかに高速に処理するかという問題が最も大きな興味の対象とされてきた。ジョインの処理技法は次のように分類できる。

a. $\theta(N \cdot M / n \cdot k)$ ジョイン

RAPに代表されるセルラロジックタイプのDBMでは，カーディナリティN，Mのリレーションをジョインするのに$\theta(N \cdot M / n \cdot k)$時間必要となる。そこでターゲットリレーションを有するセル数をn，各セルが有する比較器の数をkとする。ソースリレーションの各タプルに対し，ターゲットリレーションをフルスキャンするため，基本的には両リレーションの大きさの積に比例した処理負荷となり，これを多数のプロセッサを用いて高速化することになる。このタイプのマシンでは処理の高速化のためには，nまたはkを大きくする必要があり，連想プロセッサを用いたRELACSやジョインプロセッサを有するDIALOGなどではkを充分

大きくしたマシンとみなすことができる。kを大きくする際，必ずしも比較器をk個設ける必要はなく，Hashを用い擬似的な連想性をもたせる手法も考えられる。この際，Hashは単一関数ではなく，並列Hashにすることにより性能を向上させることが可能となる。

b. ページレベル　$\theta(M' \cdot N' / n)$ ジョイン

DIRECTではa)の如きタプルレベルの制御ではなく，ページレベルの処理方式を採っている。基本的にはページ数の積で効き，処理負荷そのものは変わらない。M', N'はそれぞれソースリレーション，ターゲットリレーションのページ数を示す。

c. $\theta(\frac{N \cdot M}{n^2 \cdot k})$ ジョイン

DPNET[1]を用いたマシンではシリンダを構成するトラック間で動的なクラスタリングを行うため，a)に比べて理想的には$\frac{1}{n}$倍高速化される。プロセッサはRAP同様，並列比較を基本としているため，kはそれほど大きな値はとれないが，擬似的にkをnkにしたものと考えることができる。nkにソースリレーションが入り切らない場合には，ディスクを何度も回転することになる。

d. $\theta(N+M)$ ジョイン

KUNGらのシストリックアレイを用いたデータベースマシン，SONGらのTree Machineではジョインを$\theta(N+M)$時間で実現できる。また，$\theta(N)$ソータ（SOE）を用いた田中らのマシンも同様である。一般に，データベースを一定の大きさに分割できたとすると，分割されたページ内では，これらの方式により高速に処理することができる。もちろん，この処理技法を，リレーション全体に対して適用することも可能であるが，大容量のリレーションに対する処理を行うためにはかなりコストが高くなると考えられる。aに比べると処理時間は和に比例するため，かなり高速化されたことになる。bのページ内処理に適用可能である。

e. $\theta(\frac{N+M}{n})$ ジョイン

dではO(N+M)の処理が可能となったが，bに適用したとしてもページレベルでの処理は$\theta(M' \cdot N' / n)$であり，両リレーションのページ数の積に比例した時間がかかり，大規模なリレーションのジョインに適さない。またdを直接リレーション全体に適用したとしても，大きなリレーションに対してはN，M自体がかなり大きな値となり，より高速化が望まれる。以上の点から，和のオーダの処理が可能で，かつバンクパラレリズムを反映することができる$\theta(\frac{N+M}{n})$の処理技法が要求されることになる。東大のGRACEがこのタイプに属する。GRACEでは動的クラスタリングを用い，データベースを独立なバケットに分割し，バケット

レベルで $\theta(n)$ の処理に落とすとともに，バケット内では $\theta(m)$ ソータを用いて処理しており，全体として $\theta(N+M)$ の負荷としているが，さらにn台のプロセッサを用い，バケットを並列に処理することにより $\theta(\frac{N+M}{n})$ の処理時間を達成している。ＧＲＡＣＥによって提案されている動的クラスタリングの方式がｃのそれと異なる点は，ＤＰＮＥＴでは，クラスタリングとジョインが同時になされるのに対し，ＧＲＡＣＥでは，一旦リレーション全体をクラスタリングした後，ジョイン処理が施されるという点である。したがって，関係代数木が低い場合（ジョインが1回）には，ジョインとクラスタリングを同時に行うＤＰＮＥＴの方式が高速である。しかしながら，ＧＲＡＣＥでは，クラスタリング処理がジョイン処理と重量されており，クラスタリング自体の時間的オーバヘッドはなくなるため，問い合わせ木が高い場合には，両者の差はなくなる。むしろＧＲＡＣＥ方式の方がスケジューリングが容易になる。また，ＧＲＡＣＥでは，バケット内の処理にソータを用いているため，その容量はかなり大きくすることが可能でありＤＰＮＥＴの如くソースリレーションの大きさが比較器の大きさによって制限されることはない。また，両方式では，バケットの対応方式も異なっている。なお，この方式は東大[2]につづいて広島大学[3]からも発表されている。さらにその後，北大[4]からも同種の提案がなされている。これは，ＧＲＡＣＥのステージング用バッファが単なるバルクメモリ（バブル，ＲＡＭ）であるのに対し，メモリ内に生成される各バケットにソータを用いようとするものである。

f. Joinability Filter

ＬＥＥＣＨ，ＣＡＦＳ等のマシンではExplicit Joinを直接マシン上で実行することはせず，ジョインのとれる可能のあるタプルのみをホストへ転送するという方式を採用している。ジョインアトリビュートにハッシュを施し，両リレーション間でのシノニムをジョイン候補とするこの手法は，図4.2.1に示される如くHashed Bit Arrayを用いて処理される。これによりジョイン対象のリレーションサイズを大幅に小さくすることが可能となる。ハッシュ関数の不完全性により，ホストでは正確なジョイン処理が必要となり，このフィルターは前処理としての効果しかもたらさない。しかし，カップリングインデックス等の完全ハッシュ関数を用いる場合には，セミジョインに対する $\theta(N+M)$ アルゴリズムと見なすことができる。また，一般にハッシュ関数は順序を維持しないため，本方式はEqui Join処理に限られ，θ－Joinには適用できない。

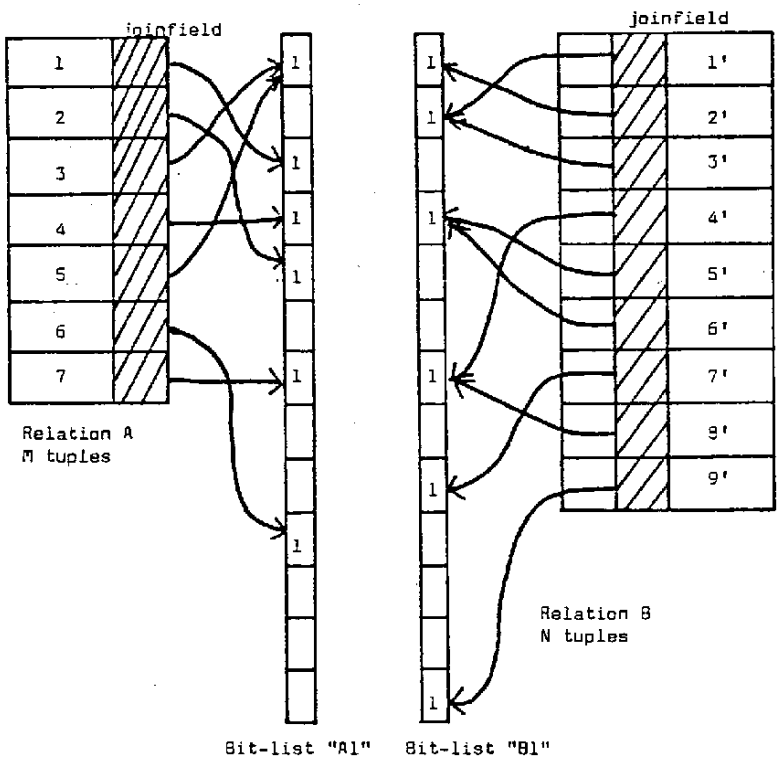


図 4.2.1　Joinability Filter

B．ソート

種々の関係代数演算は，その処理対象がソートされている場合にはθ（ｎ）で処理でき，したがってデータベースマシンでは，ソートの高速化が大きな課題と考えられてきた。特に，二次記憶からデータを読み出す際，データ転送と重畳してソートを行うことにより，実効的にソート時間をなくすことが可能となり，これを目指すといくつかのパイプラインソータが開発された。パイプラインマージソータ[5]，パイプラインピープソータ（ＳＯＥ）[6]，パイプラインバブルソータ，リバウンドソータ[7]などがそれである。ソートデータ数をｎとするとこれらの内，後の２つは比較器がｎ個必要であるのに対し，先の２つは log ｎ個ですませることが可能となり，大容量ソートに適していると考えられる。各々の処理の様子を図 4.2.2 ～ 4.2.5 に示す。

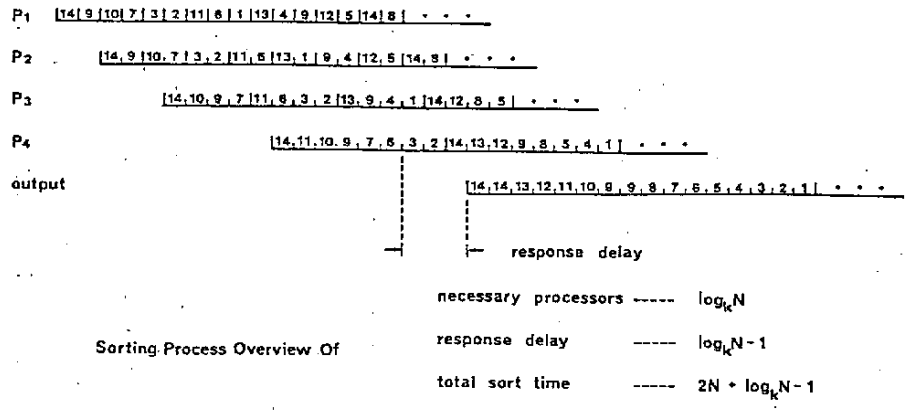


図 4.2.2　Pipeline Merge Sorter

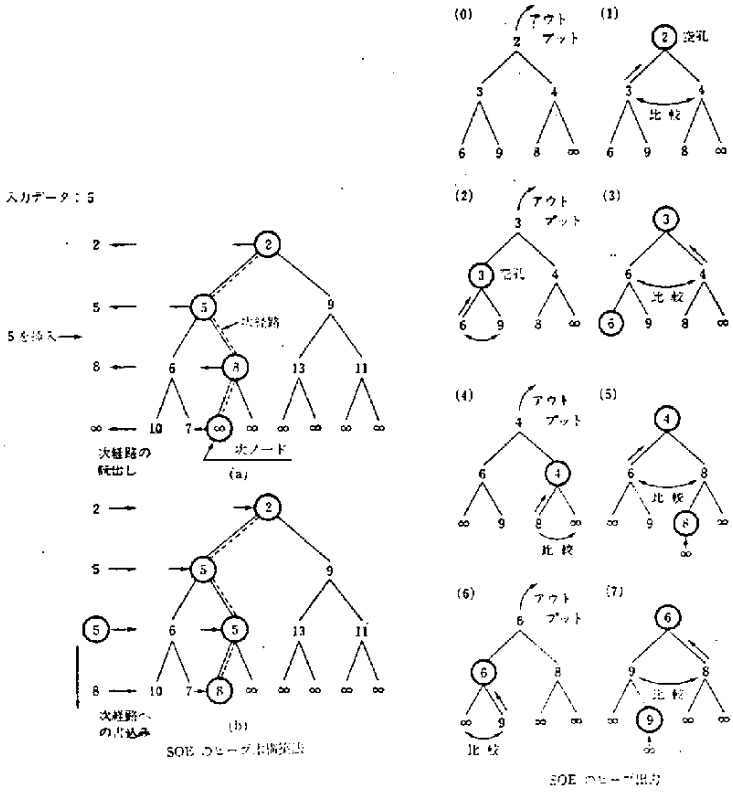


図 4.2.3　パイプラインヒープソータ（SOE）

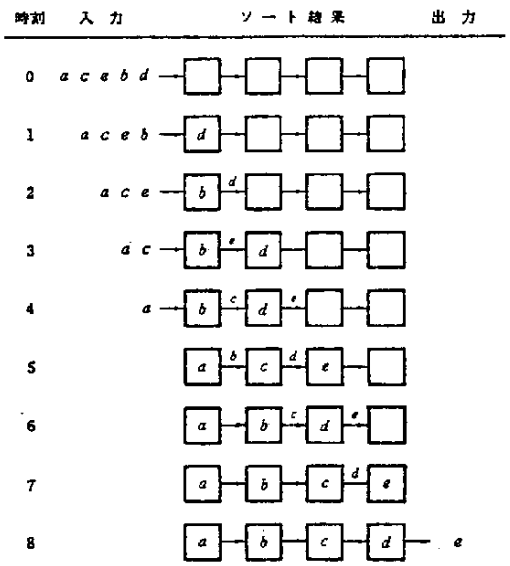


図4.2.4　パイプラインバブルソータ

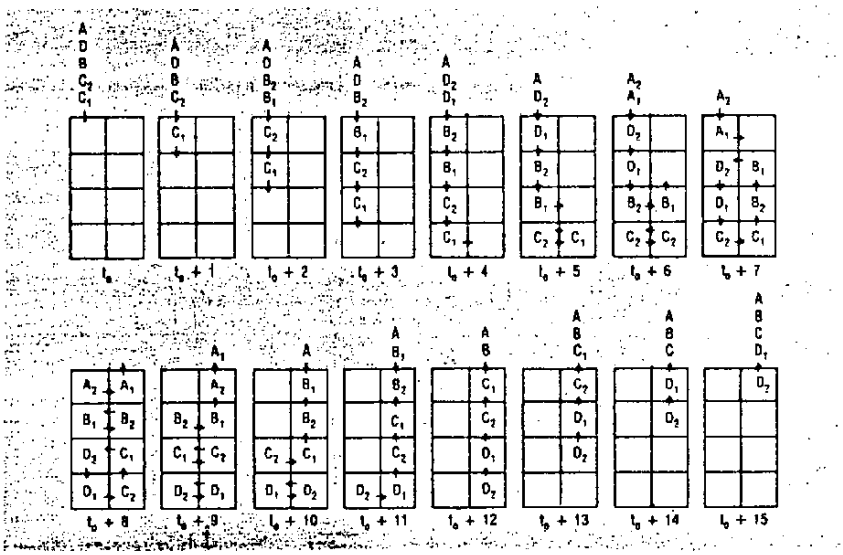


図4.2.5　リバウンドソータ

データベースマシンでは，データをブロックとして扱い，バンクパラレリズムを反映した並列アルゴリズムが望まれる。前川のアルゴリズムはブロック化されたデータに対するソートおよびジョインを効率よく行うことができる。今，全体でk・n個のレコードがnコのブロックにkコずつ分配されていて，それをソートすることを考えると，n個のブロックは並列に動作できるため，ソート時間は

$$\frac{k \cdot n \log_2(kn)}{n} = k \log_2 k + k \log_2 n$$

と表わされる。第1項がブロック内ソート処理，第2項がブロック単位のソート処理に要する時間である。実際には，理想的に処理されるわけではなく，

$$\alpha k \log_2 k + \beta k \log_2 n \quad (\alpha, \beta \geq 1)$$

と表わされる。通常，レコードの転送はレコードの比較に比べてはるかに長い時間を要するため，第2項が主要項となり，プロセッサ間結合方式がマシン構成上重要な課題となる。前川らは[8] Super imposed tree なる結合方式により（図4.2.6）βを1～1.5とできることをシミュレーションによって示している。前川のアルゴリズムは基本的には2進木を辿る操作を並列に行

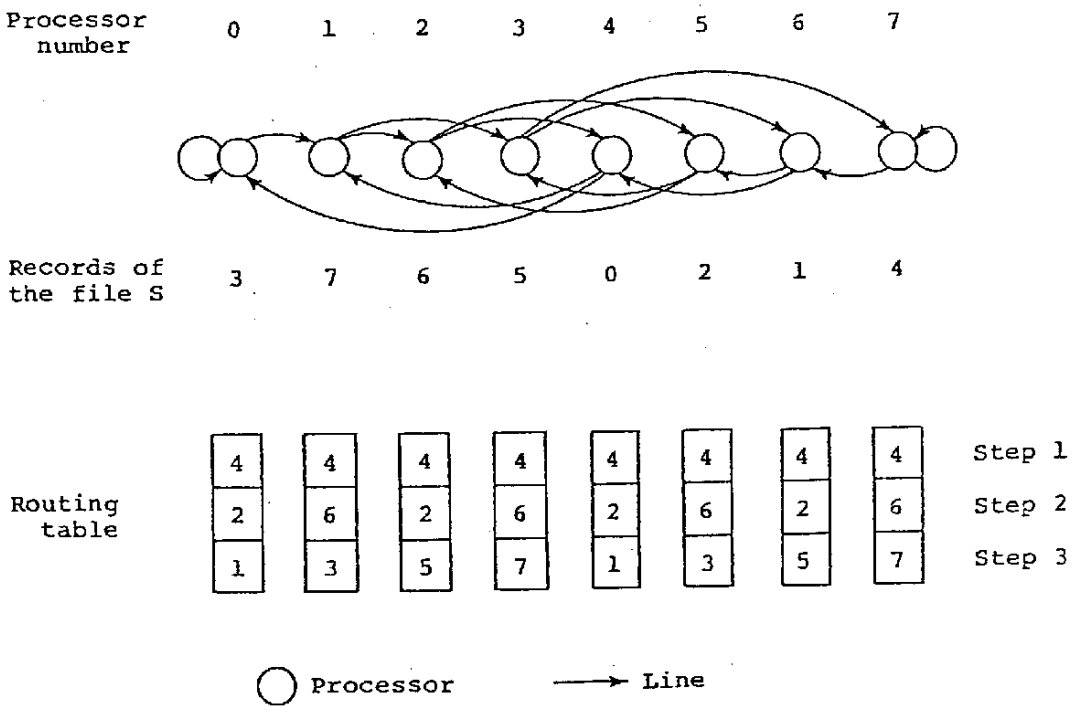


図4.2.6　前川のソートアルゴリズム

えるようにしたものであるが，この操作のためにルーティングテーブルを始めに作る必要がある。ファイルの統計情報によりルーティングテーブルが得られている場合には，理想的な性能が期待される。なお，この方式では，データの配置によりいくつかのプロセッサにデータが集中することが考えられ，プロセッサ内バッファは有限であるため，デッドロックが生ずることがある。

2次記憶から一旦，データをバッファ上にステージングしてその上で種々の処理を施すMPIS型マシンのアプローチとは別に，媒体上で直接ソートを行う方式も考えられる。Edelberg らによるインテリジェントメモリ[9]は，ジャイロソートと呼ばれるアルゴリズムによりi＞jのとき，トラックi上のデータは，トラックj上のどのデータよりも大きくなるように並べかえを行うものである。

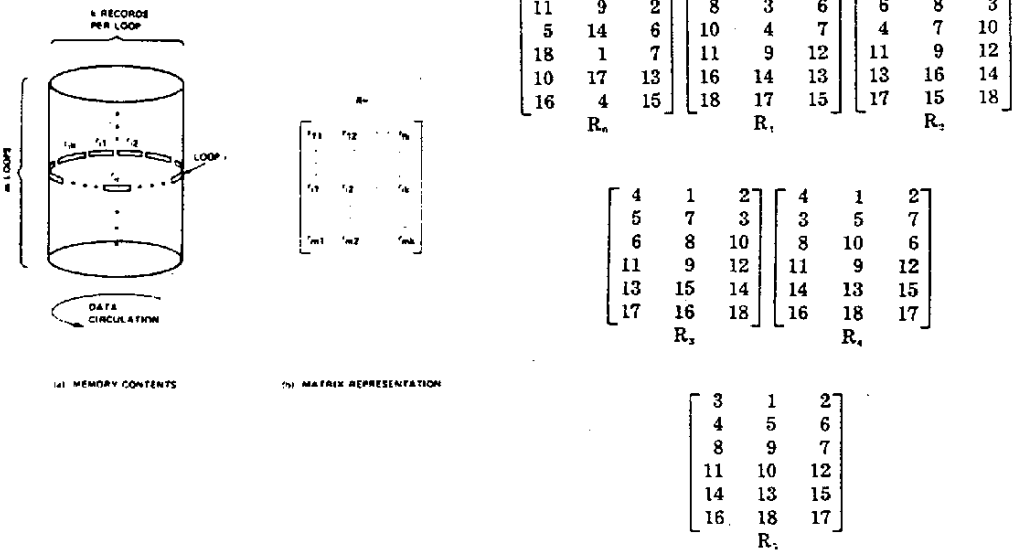


図4.2.7　ジャイロソートとその処理の流れ

なお，ソータを実装する際には，その対象とするレコードの長さの変動を柔軟に吸収できる機能が必要となる。レコード長Lに対して，Nコのレコードを行うためには，N・Lだけのメモリ容量は最低必要となるが，ハードウェア設計長Lと実際のソートレコード長Xとは必ずしも一致せず，レコード長変動に対しても高いメモリ効率を維持し，できるだけ多くのレコードをソートできることが望ましい。10)ではLength Tuningにより，SOEではビットスライス的手法によって対処している。

C．フィルタ

データベースマシンでは，2次記憶上からのデータ流に対し，その流れに追従した形でセレク

ションやリストリクションを行い，実効データ転送量の低減を図ることが常套手段となっている。RAP，CASSM，CAFSなどではトラックからのデータ流に対し，種々のアトリビュートを含む条件式を流れにそって評価する。RAP，CASSMでは，比較器の数はそれほど多くないためマークビットを用い一回転で評価できない問い合わせに対しては，何回も回転しながら処理を行う。CAFSにはマークビット機構はない。RARESではトラック並列にデータが格納される点に特徴がある。またRAPのようにディスクのトラックにプロセッサを固定するか，あるいはCAFSやDBCのTIPのようにプロセッサとトラックの間に結合機構を設けるかによってその対応関係の柔軟性に差はあるが，フィルタ機能としてはいずれも同程度のものである。

サーチプロセッサは他のマシンと異なり，複数トラックのデータを時分割的に多重化し，同時に処理することができる。一方で1シリンダ分のサーチを一回転で行うため，そのロジックは高速化する必要がある。

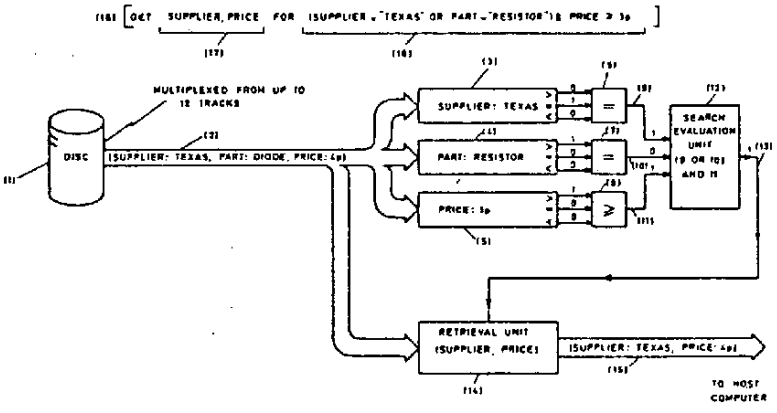
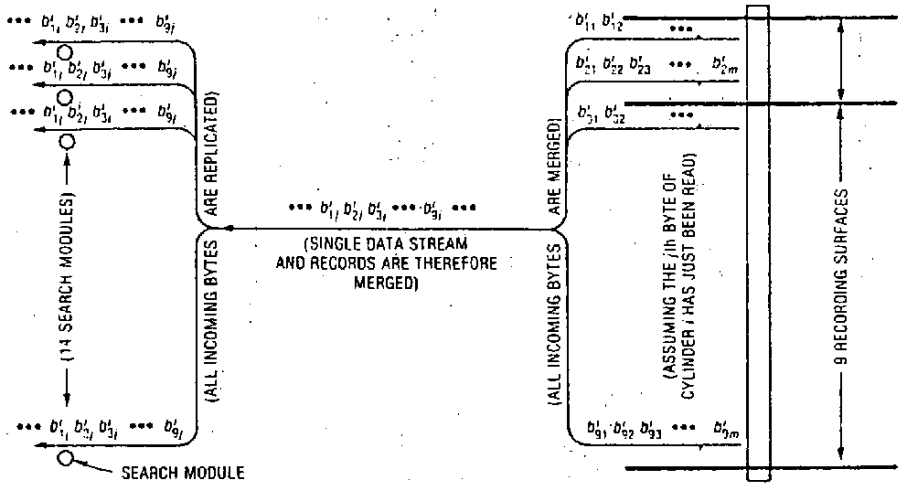


図4.2.8　CAFSのフィルター処理



NOTES: 1. Records are always stored sequentially on a track, e.g., b'_{11}, b'_{12}, and b'_{13} may form a small record.
2. Input to a search module consists of merged records, e.g., b'_{12}, b'_{22}, . . . b'_{92}.
3. Records are replicated and interleaved for each data stream.

図4.2.9　サーチプロセッサのフィルタ処理

D．大容量化技法

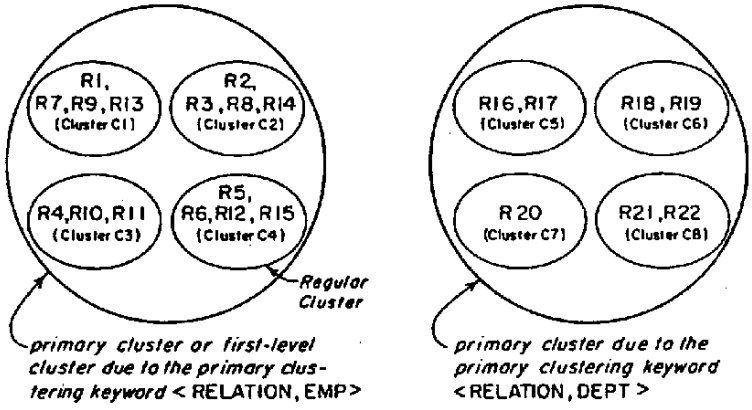
データベースマシンの開発目的の1つは，データベース管理ソフトウェアをより簡素化することであり，したがって，複雑な操作が伴う補助的記憶構造は用いず，単純な全数サーチによる処理方式が主流である。しかしながら巨大なデータベース処理に対してはやはり，検索空間はある程度絞り込むことが望ましい。ＤＢＣでは，クラスタリングインデックスを用い，２次元のクラスタリングを実現している。物理的なクラスタリングは，多次元にすることは不可能であり，多くのアクセスパスを構成するには，交代インデックス等を設けるしか手段はない。しかし，これはアクセス量が少ない場合にしか有効ではなく，多い場合には意味をもたない。ＤＢＣでは２つのインデックスに対してのみクラスタリングを適用しており，そのアトリビュートが検索条件に含まれる場合には高速化される。クラスタリングの単位は１シリンダであり，その中はＴＩＰにより全数サーチされる。したがって，追加もシリンダ内であればどこでもよく，更新に対してもそれ程大きな負荷とはならない。インデックスのレベルを高くしたこのアプローチは自然な手法であり，ＭＰＩＳ型の多くのマシンに対して適用可能と考えられる。

EMP Relation

	ENO	NAME	DNO	JOB	PNO
R1.	1	HAYES	100	MGR	10
R2.	2	NAYAK	100	ENGG	20
R3.	3	ROSEN	100	ENGG	20
R4.	4	KERNS	100	TECH	10
R5.	5	GROVE	100	SEC	10
R6.	6	PERRY	100	SEC	20
R7.	7	GHOSH	200	MGR	30
R8.	8	SLOAN	200	ENGG	30
R9.	9	PARDO	300	MGR	30
R10.	10	PRICE	300	TECH	20
R11.	11	WHITE	300	TECH	30
R12.	12	KLINE	300	SEC	30
R13.	13	HSIAO	400	MGR	40
R14.	14	PRATT	400	ENGG	40
R15.	15	BOONE	400	SEC	40

DEPT Relation

	DNO	MGR	FLOOR
R16.	100	HAYES	1
R17.	100	NKOMO	2
R18.	200	GHOSH	1
R19.	200	GHOSH	2
R20.	300	PARDO	1
R21.	400	HSIAO	1
R22.	400	HSIAO	2



Cluster id	Clustering Keywords	
C1	(RELATION, EMP),	(JOB, MGR)
C2	(RELATION, EMP),	(JOB, ENGG)
C3	(RELATION, EMP),	(JOB, TECH)
C4	(RELATION, EMP),	(JOB, SEC)
C5	(RELATION, DEPT),	(DNO, 100)
C6	(RELATION, DEPT),	(DNO, 200)
C7	(RELATION, DEPT),	(DNO, 300)
C8	(RELATION, DEPT),	(DNO, 400)

It has been given that the primary clustering attribute is RELATION and the secondary clustering attributes are JOB and DNO.

（サンプルデータベースによるクラスタリングの実例）

図 4.2.10　ＤＢＣにおけるクラスタリング技法

インデックスと共によく利用される技法としてHashがある。インデックスの場合には$\theta(\log N)$のアクセスステップが要されるのに対してHashでは$\theta(1)$となり，きわめて高速なアクセスが可能となる。しかし，従来の直接編成法ではHash関数が固定的であるため，その分散に偏りがある場合には特定のバケットにデータが集中し，シノニムチェイニングによりアクセスタイムが大きく低下する場合が多かった。したがって，ある程度以上ロードファクタを増加することはできないため，ファイルの再編成をくり返さねばならず，そのオーバヘッドは無視できなかった。これに対しDynamic Hashing[11]，Virtual Hashing[12]，Linear Hashing[13]，Extendible Hashing[14]などの技法では，ハッシュ関数をデータの挿入と共に動的に変化し，オーバフローバケットをスプリットすることにより再編成を不要としている。以下，各技法について簡単に説明する。

(1) Virtual Hashing

Litwinは$R(C, N)=C \bmod N$なるHash関数が$R(C, 2N)=R(C, N)$ or $R(C, N)+N$となる性質に着目し，オーバフロー発生と共にバケットアドレスm+Nなるバケットを新たにファイルに追加し，$h_0(C)=R(C, N)=m$なるCに対しては，ハッシュ関数として$h_1(C)=R(C, 2N)$を用いることによりバケットスプリットを行うことを提案している。一般に，$R(C, 2^j N)=h_j(C)=m$，なるバケットmでオーバフローが生じた場合には$C \varepsilon h_j^{-1}(m)$に対してハッシュ関数が$h_{j+1}(C)$へ変更される。コア内に保持すべき情報は，ハッシングの対象となっているバケットを区別するbit tableであるが，これは1バケットにつき1bitですむため，小容量のコアで十分な大きさのファイルをアクセスできることになる。Litwinはロードファクターが一定値を越さない場合にはスプリットせずチェイニングを行う方式も提案しており，この場合でも，入力キーがランダムであると，成功サーチの際のディスクアクセスは平均1回ですむことがシミュレーションによって示されている。（図4.2.11，図4.2.12，図4.2.13）

(2) Linear Hashing

Linear Hashingは，バケットスプリットシーケンスを固定化し，Virtual Hashingで用いるbit tableを不要としたものである。当然Virtual Hashingに比べてアクセス性能は劣るが，成功サーチには平均2回を越えるアクセスを行う必要はない。

(3) Dynamic Hashing

Dynamic Hashingは$h_0: \{C\} \rightarrow \{0, \cdots\cdots N-1\}$なるHash関数を用意し，オーバフローが発生するとバケット対応にキーによって定まる乱数に対するTrieを作ってゆく方法である。B－treeにしたものが富士通RDB－V1にみられる。（図4.2.14）

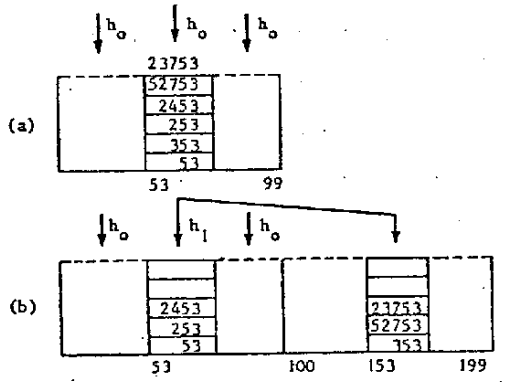
(4) Extendible Hashing

Extendible Hashingでは図4.2.15に示される如くオーバフローのたびにハッシュ値のビット長を広げ（ハッシュ関数は一定）バケットスプリットを行う手法である。

(5) Trie Hashing

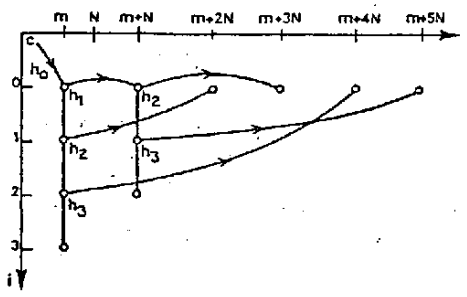
キー値をそのまま利用し，ブロックレベルのTrieを構成する手法である。オーバフローするとバケットはスプリットしてTrieが成長する。Trieがバランス化することはないが，これを1ノード6バイト程度のコンパクトな表現にすることによって，コア内で充分大きなファイルに対するTrieを保持することが可能である。Trieのためキー順アクセスができる点にも特長がある。

一般的特性として，Virtual Hashingを除いて他の手法では，ロードファクタを平均70%に維持することが可能であり，これはB－トリーのそれとほぼ等しい。またアクセス性能に関してはDynamic Hashing，Extendible Hashing，Trie Hashingではサーチに1回，スプリットに3回のディスクアクセスですむ。Virtual Hashing，Liner Hashingはこれに比べると性能が落ち，サーチには平均2回程度のアクセスが必要となる。コアに有すべき情報量はLiner Hashingの場合，ほとんどでなく純粋なHashに近い。次いでVirtual Hashingが多く，さらにDynamic Hashing，Extendible Hashing，Trie Hashingではかなり増加する。



(a) – A collision occurs for m = 53.
(b) – The split by h_1 resolves the collision and avoids creating an overflow record

図4.2.11　Virtual Hashingにおけるバケット分割



- A structure graph. Page m was split three times, by h_1, h_2, h_3. Page $m+N$ was split two times, first time by h_2 than by h_3.

図4.2.12　Virtual Hashinにおけるバケットグラフ

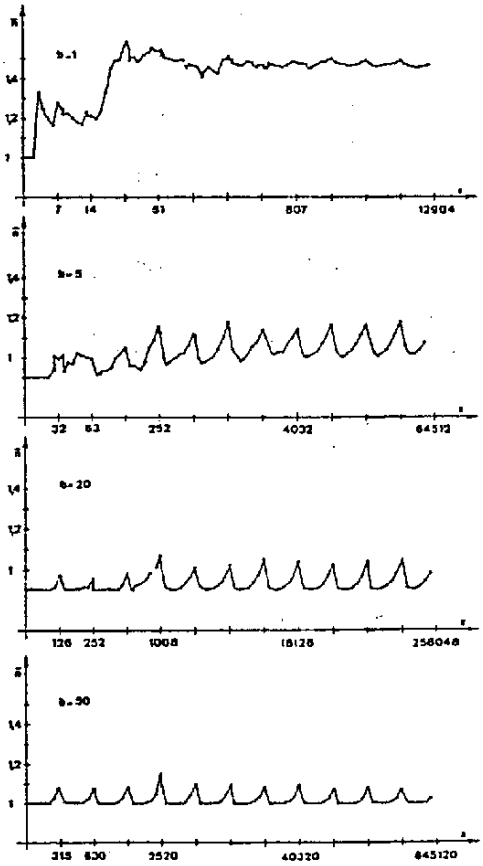


図4.2.13　Xレコードのインサートに対する成功サーチの平均アクセス時間

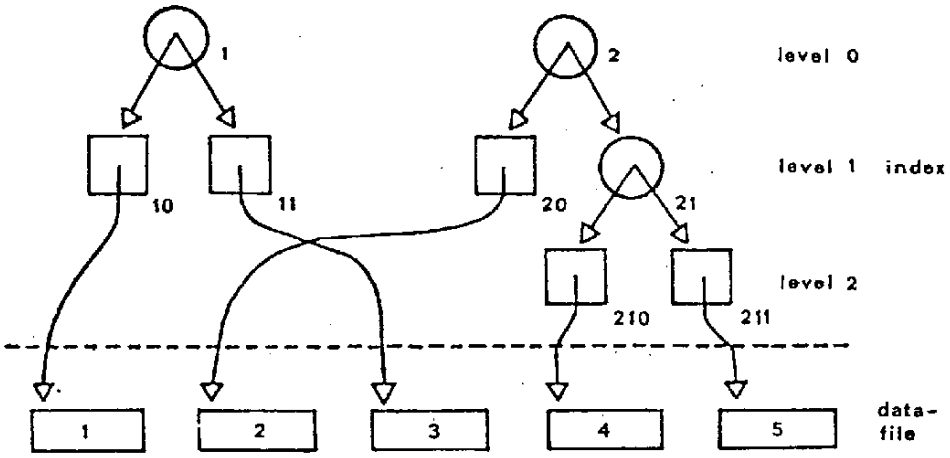


図4.2.14　ダイナミックハッシング（3回のバケットスプリットを行った後の状態）

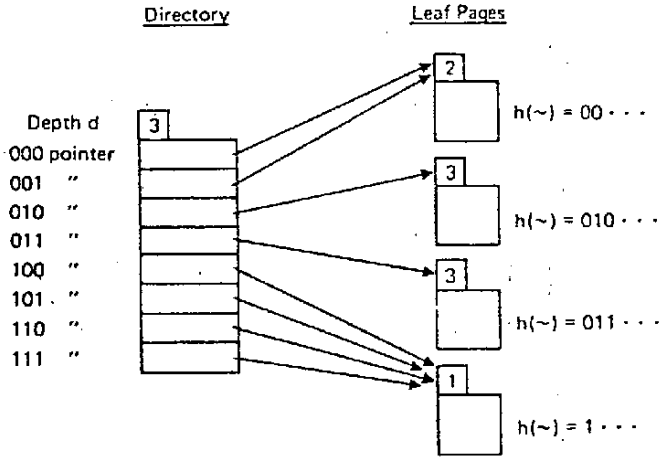


図4.2.15　Extendible Hashing

—参考文献—

各種データベースマシンに関しては3章を参照のこと。

1) 小田他　データ分配ネットワークを用いたデータベース計算機の構成法，信学技法　ＥＣ８０－７２，1980.

2) 喜連川他　「可変構造多重処理データベースマシンにおけるHashの適用」２３回情処全国大会，「HashとSortによる関係代数マシン」　信学技法　ＥＣ８１－３５　1981.

3) 上林他　「記憶階層に関係代数演算の前処理機構を組み込んだ関係データベースマシンの基本アーキテクチャ」　信学技法ＥＣ８１－４７　1981.

4) 田中他　「データフロー制御データストリーム処理方式データベースマシン」　情処，データベース管理２８－３　1982.

5) S.Todd, "Algorithm and Hardware for a Merge Sort Using Multiple Processors" IBM J.RES. DEVELOP vol 22 №5 1978

6) TANAKA, Y., NOZAKA, Y. AND MASUYAMA, A.
"PIPELINE SEARCHING AND SORTING MODULES AS COMPONENTS OF A DATA FLOW DATABASE COMPUTER" PROCEEDINGS OF IFIP CONGRESS pp.427－432 1980.10.06－09, 14－17

7) CHEN, T. C., LUM, V. Y. AND TUNG, C.
"THE REBOUND SORTER:AN EFFICIENT SORT ENGINE FOR LARGE FILES"
PROCEEDINGS OF THE 4TH INTERNATIONAL CONFERENCE ON VERY LARCE DATA BASES pp312－318 1978.09.13－15

8) MAEKAWA, M.
"QUICK PARALLEL JOIN AND SORTING ALGORITHMS"
PROCEEDINGS OF THE 13TH IBM COMPUTER SCIECE SYMPOSIUM, THE WORKING CONFERENCE ON DATABASE ENGINEERING, SOFTWARE ENGINEERING SERIES №1, IBM JAPAN pp.111－11－111－46
1979.11.17－19

9) M.Edelberg and L. R. Schissler:Intelligent Memory, Proc. AFIPS NCC, Vol.45, pp.393－400, 1976.

10) 喜連川他　「可変構造多重処理データベースマシンにおけるソートモジュール」
信学技法ＥＣ８１－１５　1981.

11) Larson, P. 「Dynamic Hashing」 BIT 18 (1978)

12) Litwin, W. 「Virtual Hashing: a Dynamically Changing Hashing」 proc 4th Conf on VLDB. 1978.

13) Litwin, W. 「Linear Hashing: a new tool for file and table addressing」 Proc. 6th Conf on VLDB. 1980.

14) Fagin. R. et al 「Extendible Hashing」 ACM TODS Vol.4. No. 3, 1979.

4.3 ソフトウェア

4.3.1 マルチユーザの支援技法

データベースシステムは，複数アプリケーションによる総合的なデータ共有を目指している。このために，データベースマシン（ＲＤＢＭ）も，複数利用者からのアクセス要求を有効に，かつデータベースのインテグリティ，セキュリティを保ちながら処理出来ねばならない。

データベースに対するアクセスは，検索演算だけの場合と，更新演算も含まれた場合とがある。検索演算の場合には，異なったユーザからの検索演算は，どのような順序で実行されても，結果には影響がない。問題は，個々のユーザ検索演算のアクセスコストを最少化するとともに，システム全体の目標（ｅ・ｇ・スループットの最大化）を達成せねばならない。リレーショナルＤＢＳ，ＤＤＢＳでは，リレーショナル検索演算に対して，結合，制限等の中間処理に表われる中間結果を最少化するアルゴリズムが開発されている。これらは，リレーションのカーディナリティ，属性の選択度（selectivity）といった統計情報によって，中間結果のサイズの平均値を見積ろうとするものである〔ＨＥＶＮＡ７８，ＳＥＬＩＰ７９〕。見積りは，リレーション内での属性値の均一分散と，属性間の値の分散の独立性の仮定にもとづいている。問題としては，この仮定が実際のデータベースにおいて成り立つかどうかである。これらの手法は，単一ユーザ検索演算に対するものである。複ユーザ検索演算のもとでは，リソースの競合，バッファ内に参照ページの存在する確率が重要な要因になる。他のユーザが利用する確率の高いページのバッファ内への保存等の手法が考えられる。これは，ＲＤＢＭにおいて，２次記憶ディスクから，ＳＤＫへのセグメントのステージングにおいて必要な技術である。

従来のＤＢＳでは，組単位（レコード単位）のアクセスが基本となっているが，ＲＤＢＭでは，セグメント単位のアクセスが基本となっている。従来のＤＢＳでは主要なコストは，２次記憶と主記憶のページングであった。ＲＤＢＭでは，より複雑な記憶階層をなしていることから，各モジュールの処理コスト，モジュール間の転送コストを，まず明らかにする必要がある。

更新演算を含むアクセス（トランザクションと呼ぶ）が複数ユーザから出される場合には，各オブジェクトに対する検索と更新，また更新と更新の演算の実行順序が問題となる。この問題は，同時実行制御（concarrency control）問題と呼ばれている。実行順序を保つ手法は，同期手法と呼ばれ，ロッキング〔ＥＳＷＡＫ７６　〕，時間順序〔ＲＥＥＤＤ７８　，ＢＥＲＬＰ７８，ＴＨＯＭＲ７９　〕手法が従来のデータベースシステム（ＤＢＳ），分散型データベースシステム（ＤＤＢＳ）において開発されてきている。ロックは，オブジェクト（ロックの単位（granularity））を，他のユーザのトランザクションからのアクセスを禁止するために用いる。ロックの単位としては，

一般に，組が用いられる。ロックの単位と，並行度とは相関があり，一般に単位が小さくなれば，並行度は高まる。他に，階層的なオブジェクトに対して，並行度を高めるための階層ロック方式〔GRAYJ75〕がある。RDBMでは，演算の単位はセグメント（リレーションの部分）であるとともに，基本演算は組単位ではなく集合単位の更新演算が基本となる。この意味で，ロック方式は，基本的な再考察が必要であろう。セグメント単位のロックでは，セグメント間のインテグリティ構造にもとづいた構造的なロックが必要となろう。

ロック方式の他の問題は，デッドロックが発生し得るために，これの検出と解除とが必要になる。ロックの要求関係を表したWait－for グラフのサイクル検出法と，サイクル内のあるトランザクションのバックアウト手法が用いられている。いつデッドロックを検出するか（e.g．週期的，ロックの要式の都度），どのトランザクションをバックアウトさせるかが問題である。原則的に，RDBMにおいても用いれるが，デッドロックの検出は，1つのモジュールで，RDBM全体をモニタして集中して行われる必要がある。

これに対して，時間順序法は，競合する演算を　トランザクションの生成の時間順に実行させることによって，演算の実行の同期をとろうとする方法である。各オブジェクトごとに，過去の更新記録をバーションとして保存する方法〔REEDD78〕，各オブジェクト各に最新の演算時間（readとWriteの時間）を記録しておく方法〔BERNP78〕とが提案されている。この方法の最大の問題は，時刻情報の格納と管理オーバヘッドである。オブジェクトの単位，演算単位の問題は，ロックと同様である。

同時実行では，複数のトランザクションの演算が，各演算のI／O時間のために，インタリーヴして実行される。この一連の演算実行はスケジュールと呼ばれる。スケジュールはデータベースのインテグリティを保たねばならない。各トランザクションの実行が正しくインテグリティを保つと仮定する。この時，トランザクションがシリアルに実行されればスケジュールはインテグリティを保つ。これをシリアルスケジュールと呼ぶ。一般のスケジュールが正しいとは，このスケジュールがシリアルスケジュールと等価（i．e．同一入力に対して，同一の出力）である時，正しいという。しかし，逆は必ずしも真ではない。これを，シリアライザブル（serializable）という〔ESWAK76〕。また，全てのトランザクションが（well－formed）2フェースロック（2PL）構造をしている時，このスケジュールは全てシリアライザブルであることが知られている〔ESWAK76〕。同様に，この逆も必ずしも真ではない。2PLトランザクションとは，全てのアクセスされるオブジェクトは，その前に必ずただ一度ロックし，トランザクション終了前に必ずロックを解放し，かつ最初の解放（unlock）演算後はどのオブジェクトに対してもロックをしないトランザクションである。より並行度を高めるためには，2PL構造ではないトランザクション

を考えることが必要である。この時には，トランザクションと，トランザクション相互関連の意味論的な解析が必要になる。ＲＤＢＭにおける集合演算にもとづいたトランザクション定義方法の研究が今後必要である。

4.3.2 柔軟性に関する技法

ＲＤＢＭは，ユーザのアプリケーションの変化と多様化と，ハードウェア技術の変化に対して柔軟でなければならない。

データの利用要求の変化に対して，そのパフォーマンスを改良するために，データベースの物理構成をチューニングさせる方法がある。ＲＤＢＭでは，リレーションのセグメントの変更，セグメントの格納配置の変更がこれである。また，利用状況をモニタしながら，現在のデータ利用要求に合うように自動チューニングする方法も必要である。

データの利用要求の変化が，データベースのスキーマの変更を要求する場合には，データベースの物理構成の再構成が求められる。現在のＤＢＭＳでは，時間を要する処理である。ＲＤＢＭにおいて，各ＳＤＭの格納データを再構成する手法が必要である。

ＲＤＢＭでは，可変長のデータを扱かえることが必要である。可変長データの処理方法としては，固定長データにエンコードして，ＲＤＢＭに格納し，処理を固定長で行う方法も考えられる。そしてユーザに出力する時は，逆にデコードして出力する方法も考えられる。

また，ハードウェアモジュールの追加，削除が容易であることが必要である。モジュールの追加は，他のモジュールに対する変更なく行なえる必要がある。このためには，インタフェースの高水準化も必要である。

4.3.3 ＲＡＳＩＳ

ＲＤＢＭが複数のユーザ下で動作する時，そのＲＡＳＩＳ（reliability，availability，serviceability，integrity and security）を保障できねばならない。特に，複数ユーザのもとでのデータベースに対する更新に対して，データベースのセキュリティ，インテグリティを，ＲＤＢＭのハードウェア，ソフトウェアの障害のもとでも保障する必要がある。

ＲＤＢＭのＲＡＳＩＳを保障するための技術としては，以下のものがある。

A．素子の高信頼化

高信頼性素子を開発し，用いる。ＶＬＳＩ化によってサブシステムの信頼性を向上させる。

B．冗長構成

システムの信頼性を高めるための技術としてシステムの構成要素を冗長化する方式がある。冗

長化の方式としては，並列冗長方式と待機冗長方式とがある〔INOSH77〕。前者は同じ要素を複数個並列運転させるもので，この中の少なくとも1つが正常動作していれば，システムの機能は正しく同じ性能で動作できる。一方後者は，各要素を複数個用意しておき，この中の1つが使用中の時には，他を待機状態にしておく。使用中のものが障害を起したならば，直ちに待機中の要素に切り替えて使用する。さらに，n者択r系と呼ばれる方式で，n個の要素のうち，少なくともr個が正常動作していれば，システムは正しく機能しているとみなすものである。

Tandem社のNonstopマシンは，プロセッサ，メモリ，2次記憶，プログラムも全て並（直列）2重化することによって，システムの高信頼化を実現している〔GRAYJ81〕。

システムの冗長方式は，並列，待機，n者択r方式を組みあわせた方式が実際には用いられる。

C．自動故障診断技術

修理時間短縮のためには，故障の起きた場所と原因を明らかにする自動故障診断技術が有効である。システム構成要素の診断は，ある診断データを入力して，その結果を調べて，故障があれば，その場所，原因を結果の解析から明らかにすることによってなされる。

故障診断方式としては，IBM360のFLT（Farlt Loading Test），マイクロ診断方式，№1ESS方式が有名である。

D．データベースのインテグリティ

データベースは，あるインテグリティ条件を満足するデータの集合である。データベースに対する更新は，データベースの内容の変化をもたらすが，この変化結果もまたインテグリティ条件を満足するものでなければならない。各更新演算に対して，チェックすべきインテグリティ条件は何であり，この条件をどのように記述し，いつどのようにチェックするかが問題となる。

現在インテグリティ条件としては，現在以下のものがある〔DATEC73〕。

(i) 組制約　例　Sal ＜ 500,000

(ii) 集合制約　例　従業員の平均給料は，200,000円以上

(iii) 静的制約　データベースの正しい状態を記述する。

(iv) 遷移制約　データベースの正しい状態変化の記述

例　new sal ＞ old sal

(v) immediate　各演算各の制約

(vi) 遅　延　トランザクションごとの制約

(vii) 属性値の範囲，形式

(viii) キー，FD，MVD

これらは，問い合わせ言語を用いて記述されたり（e.g・QUEL〔STONM76〕，QBE〔ZLO

OM77〕),アサーションとして表す(System R〔ASTRM79　〕)方法が用いられている。これらの場合,インテグリティは,問い合わせ修正(query modification)手法〔STONM76〕等の論理式の操作によってなされる。これらの手法を,RDBMで用いるならば,ソフトウェアシステムによって処理することになる。記号処理マシン,定理証明マシンがあれば,高速に論理式の処理を行える。式のレベルで表わせるインテグリティ以外(例,追加に対する主キー,FD)は,データベースのアクセスが必要になる。

他に,オブジェクト間の更新の依存性(update dependency)のサポートが更新において重要である。例えば,ある組の消去が,他の組の消去をもたらすような場合である。このサポートとしては,System Rのトリガー機構が,RDBMでも用いれる。

E. セキュリティ

データベースのセキュリティは,利用資格のないユーザからの不当なアクセスを防止することに関している。セキュリティ機構としては,パスワードによる利用者の認識方法と,インテグリティと同様に,対象となるオブジェクトを論理式で記述し,これを問い合せ修正手法によって論理式を処理してしまう方法がある。他に,オブジェクト毎に,許される演算と,その利用者を明確に定義し,ケーパビリティアーキテクチャ的に実現する方法も考えられる。データベースでは,対象となるオブジェクトの構造が複雑であるために,全ての場合には,この方式は適していない。

他に,セキュリティの権限委譲がある(System R)〔FAGIR78　〕。

セキュリティについては,インテグリティと同様に,理論的に未解決の問題が多く,今後,基礎的な研究を行い,RDBMへの適用を考えていく必要がある。

F. 障害回復(recovery)

データベースシステムは,種々の障害に対して,データベースのインテグリティを正しく保つ必要がある。障害では,トランザクションの概念が重要である。トランザクションとは,ユーザから見た,原子的(atomic)な実行単位である。従って,トランザクションの実行は,完全に実行されたか(commitされたか),または全く実行されなかったかである。コミットされたトランザクションによるデータベースの変化は,物理的にデータベースに対してなされている。このため他のトランザクションからも見えることができる。しかし,コミットされていないトランザクションによるデータベースの変化は,全くデータベース上に反映されないし,他のトランザクションによっても見ることは出来ない。この原子性を保障するために,2フェーズコミット手法がある。これは,更新する全てのデータベース媒体(RDBMではSDM)が,更新データを受け取りログにセーヴしたのを確認した後に,物理な書き込みを行うものである。1つでも媒体が,更新できなければ,トランザクションは,バックアウトされる。RDBMで,更新データ

が，複数のＳＤＭにまたがっている場合には，この手法を用いることができる。

システムの障害から回復するための手法としては，次のものが現在試みられている。

(i) ログ（ log ）

ログには，各トランザクションの演算の実行前の値と，実行後の値がセーヴされる。前者をundo ログ（またはbefore 値），後者をredo ログ（またはafter 値）と呼ぶ〔ＩＭＳ，System Ｒ〕。データベースのbuffer に対する更新を行う以前に，ログに書き込まれる（write－ahead protocol （WAP））。

データベースを，ある状態に戻すには，undo ログを用いて，逆演算を順々に行っていく（ backward recovery ）。逆に過去の状態から現在に戻すには，redo ログを用いる（ forward recovery ）。

ログを格納する媒体は，通常ＭＴが用いられている。ログ媒体のスピードが，システムのパフォーマンスに影響することから，ＲＤＢＭでは，高信頼な高速記憶（ e.g・ＳＤＭ）をログに用いる必要がある。

(ii) バックアップコピー

データベース全体を，ＲＤＢＭとは切り離された記録媒体に格納する。媒体としては，ＭＴ，ＭＳＳが考えられる。バックアップを取ることは，多くの時間を費やす作業であり，この間，システム上でトランザクションを実行出来ない。周期的にバックアップを取る方法，または夜間に取る等の方法がある。記憶媒体（ e.g・ＳＤＭ）の障害時には，このコピーからバックアップされる。ログを用いて，現在までで，コミットされたトランザクションを全て再実行させる。

(iii) チェックポイント

チェックポイントは，トランザクションが１つも動いていない，ログ上のポイントである。従って，ある時にシステムが障害を起こしたが，データベースは障害を受けていない時，ログを用いて，この点まで全てのトランザクションを undo させ，次にこの点からコミットされたトランザクションのみをredo させることによってデータベースを回復できる。

(iv) differential ファイル

ＩＮＧＲＥＳで用いられている方法で，データベースの更新結果は，differentialファイルに格納しておく。検索は，まずdifferential fileをアクセスし，見つからなければデータベースを見にいく。適当な時に，データベースとdifferential ファイルとをマージする。データベースのバックアップがあれば，データベースが障害を起こしても，このバックアップとdifferential ファイルによって現状に復旧できる。

(v) careful replacement

データベースのwrite を行う時，更新すべきオブジェクト（page）をコピーして，コピーしたものを更新する。これによって，必ず，更新前のオブジェクト値が残るので，信頼性を高められる。

これらの手法は，RDBMにも適用できるが，多くはソフトの問題である。RDBMでは，記憶階層が複雑なので，各階層ごと，及び階層間の障害のモデル化が必要である。ログ，バックアップは，基本的な手法として，RDBMにも適用可能である。各SDMが，ローカルな処理能力によって更新を行う場合には，各SDMごとに障害回復機構（ログ等）が必要である。

4.3.4 高水準言語インタフェース

関係データベースマシンは，ユーザにとって利用が容易である必要がある。このため，データベースマシンにとって易性性向上技術は重要である。

ユーザにとっての易用性は，ユーザインタフェースの問題である。このためには，高水準なデータベースアクセス言語が必要になる。現在，ユーザ言語としては，SEQUEL〔CHAMD76〕QUEL〔HELDG75〕等の関係計算言語と共に，QBE〔ZLOOM77〕のような2次元言語が，リレーショナルDBMSではサポートされている。他に，PL／I等の高級言語に，関係計算機能を組み込んだ言語がある。知識情報処理に向けたRDBMでは，より容易な高水準言語が望まれる。例えば，関数型言語がある。インタフェースも，QBE，SDMS〔HEROC80〕的な2次元的なものが必要である。

言語と共に，種々の高水準データ構造のサポートも必要である。ユーザに対するビューのサポート機能も必要である。ビューとしては，リレーショナルなビューのサポートを考える。リレーショナルなスキーマに対するCODASYLモデルのような手続的モデルのサポートは，手続から非手続的意味記述の生成が必要となり困難となり，ここでは考えない。ビューを通しての更新に対しては，データ抽象化技法の適用が考えられる。これはビューを抽象データ型として，これに対して許される演算をexplicitに定義してしまう方法である。

4.3.5 分散データベースシステム関連技術

分散データベースシステム（DDBS）は，複数の自立的なデータベースシステム（DBS）を，通信ネットワークによって結合するとともに，ユーザに対して各DBSの異種性とデータの所在の不可視性を提供するシステムである〔TAKIM78〕。DDBSとRDBMとの関連については，次の2点が考えられる。

(1) ＲＤＢＭを，ＤＤＢＳのノードシステムとする。

(2) ＤＤＢＳにおける同時実行制御，障害回復制御方式を，ＲＤＢＭの更新方式に適用する。

A．ノードプロセッサとしてのＲＤＢＭ

ＤＤＢＳのノードプロセッサとしてのＲＤＢＭについて考える。ＤＤＢＳでは，各ノードのＲＤＢＭが保有しているデータ記述／演算に関する層（ローカル概念スキーマ層（ＬＣＳ））と，ＤＤＢＳ全体の異種及び分散が見えない仮想データベース記述／演算に関する層（全体概念スキーマ層（ＧＣＳ））とがある〔ＴＡＫＩＭ７８，７９〕。ＤＤＢＳ実現上の問題は，以下の２点である。

(1) ＧＣＳとＬＣＳとのデータ構造の対応づけ問題，e.g.ＬＣＳリレーション（ＲＤＢ内のリレーション）のビューとしてＧＣＳを定義する。

(2) ＧＣＳ層の演算を，関連するＲＤＢと通信ネットワークを用いての通信処理し，結果を得る問題

(1)は，主にDD/Dの問題となる。統合的（bottom−up）にＧＣＳを形成する場合には，データベースの意味との対応づけである。現在，十分な設計支援ツールもなく，困難な問題である。

一方，分割的（top−down）的に，ＧＣＳ仮想データベースから，ＬＣＳのＲＤＢＭにデータベースを分割する場合には，想定される利用形態のもとで，信頼性，アベイラビリティ，応答性等を最適とするように決定される。データの最適配置では，複数ノードでの同一データの冗長保持方法が問題となる。データを冗長保持させると，ノード障害に対する信頼性の向上とともに必要なデータが自分のノードに存在する確率が高まり，通信が不要となり良い応答性をもたらす。一方，更新演算に対しては，冗長コピー間のデータの一致性を保つための通信オーバヘッドが増大し，システム全体のパフォーマンスを低下させてしまう。従って，信頼性，応答性，アベイラビリティの要求とともに，データの検索と更新頻度とによって，冗長性とデータの分配が決定されるべきである。

(2)では，検索演算に対して，通信処理コストを最少とするための最適化が問題となる。最適化の目標としては，以下の２点がある。

(i) 応答時間

(ii) 全処理時間

応答時間は，各ノードが処理能力を有していることから，通信処理の並行度を高めることによって達成できる。一方，全処理時間は，１つの演算の全通信処理コストであり，システムのスループットを与える。一般に，これらの目標間には，トレードオフが存在している。

第２には，通信と処理のコスト比率である。ARPANET，CYCLADESといった広域パケット変換ネットワークでは，高々５０Kbps の回線速度しか有しないために，各ノードでの処理速度

は，通信速度に比して無視し得る。一方，Ethernet等のローカルネットワークでは，10Mbps程度の帯域を有しているために，通信と処理のコストは，互いに，比較し得るものとなる。1Gbps程度の光ファイバを用いた通信ネットワークが用いられると，通信と処理とのコスト関係は逆転したものになろう。この場合には，通信量を減少させることよりも，処理量を減少させるような通信方法が必要になる。

第3に，通信処理のスケジュール方法がある。スケジュールの決定方法としては，以下の2種がある。

(i) 静 的

(ii) 動 的

(i)は，通信処理のスケジュールを，その実行の前に全て決定してしまう方法である。スケジュールは，ある目標関数を最少とするように決定される。現在は，通信処理中の関係演算の中間結果の大きさを，カーディナリティ，選択度（selectivity）〔HEVNA78〕といった統計情報を用いて見積ることによって，応答時間／全処理時間を見積ることが試みられている。この見積りの前提には，以下がある。

(a) 各リレーションにおいて，各属性の値は，均一に分散している。正確には，結合を取る2つの属性のうち，少なくとも一方の値が均一分散している。

(b) 各リレーションこの属性間の値の分散には，相互関連がない（i.e.独立である）。

この仮定が，実際にどの程度満足されるかは実際のデータベースの値の分散によっている。分散に偏りがあり，各アプリケーションごとに，アクセスする値が異なっている場合には，経験的な選択度を用いることが有効である。

選択度には，こうした平均値と，実際にアクセスする範囲との間の相関性の問題と共に，選択度，カーディナリティといった統計情報の管理形態にも問題がある。データの更新頻度が高ければ，統計情報自体も更新されねばならない。DDBSの場合には，各ノードがDD／Dとして通信処理のためのパフォーマンス情報を有している。この時，データと，冗長なパフォーマンス情報（i.e.選択度）との間の一致性を保つ問題が生じる。これは，後述するような同時更新制御（concurrency control）問題となり，一致性を保つための通信オーバヘッドが生じてしまう。しかし，検索中心の固定業務データベースに対しては，有効な手法である。

マルチユーザ下では，リソースの競合による待ち状態が生ずる可能性がある。この時，1つの処理のリソース待ちによる遅れによって，全体の処理が遅れてしまうことが生じてしまうことがある。マルチユーザ下のスケジューリングでは，こうした待ち行列による動的な遅延をどのように反映させていくかが問題である。

他のスケジューリング方法は，動的決定法と呼ばれるものである。これは，前述した統計情報の管理と正確さへの問題点を解決することを目指している。動的決定法では，通信処理の実行情況を見ながら，次に行うべき最適な方針を決定し実行させる方法である。静的決定法では実行前の状態で全てのスケジュールを決定していたが，ここでは各実行ステップ各に，その時点での最適解を求めようとする。ここでの問題は，各実行ステップ各の状態情報をどのようにモニタするかである。モニタのための制御情報の通信と同期は，システム全体のパフォマンスの低下をもたらす。決定の正しさと，そのための情報獲得のためのオーバヘッドとのトレードオフが存在する。

実際には，静的決定と動的決定との混合形態が可能である。即ち，あらかじめ通信処理スケジュールの戦略を，現在の統計情報を基にして決定しておき，途中で予定よりコストが悪化した時に，その時点から再度スケジューリングをやり直す方法である。

第４に，通信処理の各ノードでの分散実行の制御方式がある。制御方式としては，以下の２点がある。

(i) 集中制御

(ii) 分散制御

集中制御では，１つの検索演算に対して１つのコントローラが，その実行の制御を行う。即ち，あるノードから他のノードへのデータの転送，各ノードでの演算実行の開始／終了，互いの実行の同期，障害回復の制御を，１つのコントローラが集中して行う。コントローラは，各ノードでの実行の全体を把握できるので，障害回復等の全体的視野の必要な制御（e.g.デッドロック検出）には適している。一方では，コントローラ近傍に，制御トラフィックが集中しパフォーマンスボトルネックになることと，コントローラが障害を起こした場合の信頼性とに問題がある。

これに対して，分散制御では，各ノードは自立的に処理を行う。例えば，データフロースキーマにもとづいて，必要なデータが到着した時点で，自分の処理を行い結果を他のノードに出力していく方式である。演算の実行とデータの転送は，各ノードの自主判断によって行われる。このため，集中制御の場合のような制御情報が不要になり，より高い並列性をもたらす。しかし，障害回復，デッドロック管理のような全体的視野の必要な処理が困難である。

実際には，なるべく分散制御を行い，全体的視野の必要な障害回復，デッドロック検出等のための集中制御機能をもたす方式が考えられる。

B．ＤＤＢＳにおける更新制御方式

分散データベースシステム（ＤＤＢＳ）は処理能力とデータベースとを有したデータベースシステム（ＤＢＳ）を通信ネットワークを介して結合したシステムとしてモデル化できる。ユーザレベルでの１つの原子的な実行単位（これをトランザクションと呼ぶ）は，これらの複数のＤＢ

S上で，実行されることになる。DDBSにおける更新制御の目的は複数トランザクションを同時に実行した時，データベースのインテグリティを保つと共に，各トランザクションの原子性を保障することである。

DDBSでは，各トンザクションの演算は，複数のDBS上で分散されて実行される。トランザクションのスケジュールは，DBSと異なり，各DBSのスケジュールの集合となる。DDBSでは，競合する演算の相対的実行順序を，どのDBSでも同一にすることが必要である。このための手法として，4.6で述べたように，2フェーズロック（2PL）と，タイムスタンプ順序づけ（T／O）の2つの方法がある。前者は，オブジェクトをロックし，他のトランザクションからのアクセスを禁止することによって，実行同期を取る方法である。後者は，トランザクションを，あらかじめタイムスタンプ（時刻）によって順序づけ，競合演算が，この順に各DBSで実行させようとするものである。これらのDDBSにおける同時実行の問題は，集中DBSと較べて，各演算実行の同期を取るための通信オーバヘッドである。もう1つは，データの冗長コピーの存在が考えられる点である。

a．2PL同期方式

DDBSでは，各論理オブジェクトXに対して，冗長コピー χ_1，……，χn が存在し得る。write に対して，各コピーの一致性を保つとともに，read に対しては有効なアクセスを保障せねばならない。このために以下のような方法が提案されている。ここで，論理オブジェクトXに対して，χ_1，……，χn を各々異なったDBSに存在するXの冗長コピーとする。

(i) 主コピー方式（primary copy 2PL）

χ_1を，主コピーとする。Xに対するreadは，主コピーχ_1をRlockしてから，ある1つのχiをreadする。vriteに対しては，χ_1をWlockして，全てのχ_1，……，χnをwriteする。即ち，χ_1において，競合演算の制御を行う。writeに対して，全てのコピーをWlock する必要がない利点がある。ただし主コピーが，パフォーマンスのボトルネックとなり得る。

(ii) 多数決方式（votirg 2PL）

ww一同期のためには，全てのコピーのWlockを試みる。過半数のコピーにWlock を行えたならば，全コピーへのwriteを行う。Wlockできなければ，トランザクションをバックアウトさせる。

rw一同期には，全てのコピーへのRlockが必要となる。このため，read に対して，多くの通信オーバヘッドが必要になる。

(iii) 集中制御

2PLのためのスケジューラが，1つのノードに集中化される。各トランザクションのread，write要求は，この集中スケジューラに出され，必要なlockはこれがセットする。後述するデッドロックの制御には適しているが，集中スケジューラがパフォーコンスボトルネックになり得ることと信頼性の問題とがある。

ロック方式では，デッドロックが生じ得る。デッドロックに対する解としては，以下がある。

(イ) デッドロック検出(deadlock detection)

(ロ) デッドロック防止(deadlock prevention)

(イ) デッドロック検出

デッドロックは，トランザクションのwait-forグラフがサイクルを持つことによって示される。wait－for グラフは，トランザクションを節とする有向グラフである。有向辺Ti→Tj(Ti，Tj は各々トランザクション)は，TiはTj によって現在所有されているロックの解放を待っていることを示している。

DDBSでは，各DBS内のデッドロックとDBS間のデッドロックを考える必要がある。DBSのデッドロックは，DBS内のwait－for グラフから検出できる。DBS間のデッドロックの検出には，DDBS全体の集中制御が必要である。1つのデッドロック検出ノードに，周期的に各ノードDBSのwait－for グラフを送出する方法がある。しかし，この周期内に，デッドロック状態が存在し得る。一方，周期を短くすると，wait－forグラフ送出のオーバヘッドが増大してしまう。また，phantom deadlock 問題が生じる可能性もある。

送られたローカルなwait－for グラフの集合に，DBSノード間のwait－for辺を付加することによって，DDBS全体のグラフを形成できる。

グラフ内にサイクルが存在すれば，サイクル内のあるトランザクションをバックアウト(backout) し，再スタートされる。このバックアウトされるトランザクションは，backout－restartのコストが最少のものを選ぶ。トランザクションがバックアウトされると，それが現在有しているロックが全て解放されるので，wait－for グラフのサイクルが切断され，デッドロック状能が解除される。

(ロ) デッドロック防止

デッドロック防止方法は，デッドロックが起りそうな時に，前もってあるトランザクションをバックアウトしてしまう方法である。

1つの方法は，各トランザクションに優先度を与えるものである。T_iとT_jを各々異なったトランザクションとする。Tiがロック要求を出した時，Tjがすでにそのオブジェクトのロックを有していたとする（即ち，wait－forグラフ表現でTi → Tj ）。トランザクションTの優先度として，タイムスタンプ（トランザクションの初期化時のＤＢＳのローカル時間＋ＤＢＳ番号）を用いて，これをts（Ｔ）と記す。TiがTj より古ければ，ts（Ti）＜ts（Tj）となる。この時，以下の２つの方法がある。

(α) wait－die

Ti→Tjの時，Tiの方がTjより古い（ts（Ti）＜ts（Tj））ならば，Tiをバックアウトして，同一のタイムスタンプで再スタートさせる。Tiの方がTjより若ければそのままwaitさせる。

(β) wound－wait方式

(α)と逆に，Tiの方がTjより若い時（ts（Ti）＞ts（Tj）），Tjをバックアウトして再スタートさせる。TiがTjより古ければ，そのままTiをwaitさせる。

どちらの方式によっても，デッドロックを防止できる。

防止におけるもう1つの問題は，同一のトランザクションが，繰り返し再スタートされることがある（cyclic restart）。(b)のwound－wait方式の方が，より再スタートが少なくてすむ。

b．タイムスタンプ方式（Ｔ／Ｏ）

タイムスタンプ順序づけ（timestamp ordering（Ｔ／Ｏ））とは，トランザクションのserialization 順序（タイムスタンプ順）をあらかじめ定め，実行をこれにもとづくように制御する方式である。各トランザクション（Ｔ）は，初期化時に，タイムスタンプが与えられる。Ｔの各演算opは，全て同一のタイムスタンプを有する。ts（op）をopのタイムスタンプとする。各ＤＢＳは，各トランザクションからの競合する演算が，タイムスタンプの順で実行される様に，スケジュールを行う。各競合演算があらかじめ定められた順（タイムスタンプ順）で実行されるので，→rwr と →ww 関係は，サイクルをなさず，各関係→はタイムスタンプ順になっている。このため，生成されるスケジュールは，serializable であり，正しい。

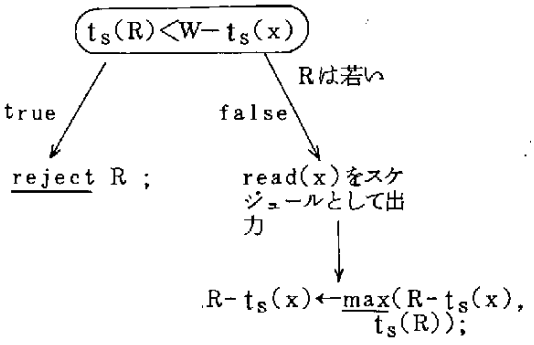
Ｔ／Ｏの実現方法について述べる。χをオブジェクトとする。各χは，以下の情報を有している。

R－ts（χ）≜　χに対する最後のread 演算のタイムスタンプの値

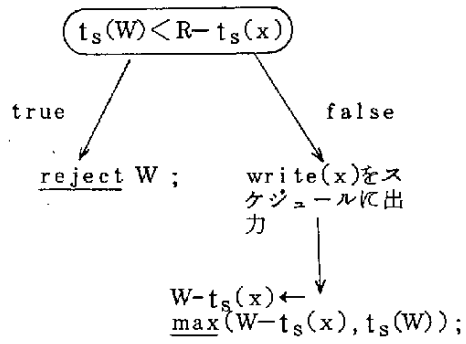
W－ts（χ）≜　χに対する最後のwrite 演算のタイムスタンプの値

r－w同期方法は，以下のように示される。ここでRとWを，各々，あるトランザクションのread（χ）とwrite（χ）演算とする。各データベースシステムは，T／Oスケジューラを有し以下を行う。

Rに対して



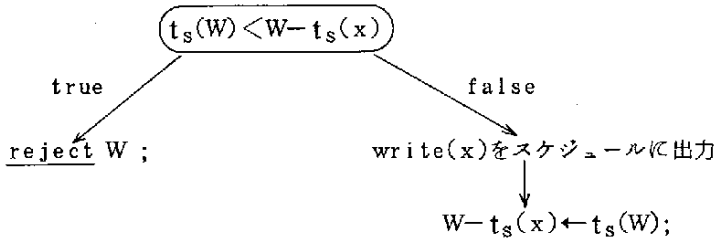
Wに対して



ある演算がrejectされると，これを出したトランザクションがバックアウトされる。新たなタイムスタンプを与えられて，トランザクションは再スタートされる。

w－w同期は，以下のようである。

W（write（χ））に対して



2フェーズコミット（aを参照）とあわせてT／Oを用いることができる。

(i) マルチバーションT／O

各オブジェクトに対して，過去のread時間，write時間とその値のヒストリを格納しておく。readは，このヒストリ内から時間に合う値を読むことである。writeはヒストリに値を時間とともに書き込むことである。タイムスタンプ方式は，この方式の1つで，最近の値のみを残すものと考えられる。マルチバーション方式は，read，write演算が，バッファされたりrejectされることが少くなり，応答性が向上する。問題は，各オブジェクトに対するヒストリの格納と管理オーバヘッドである。

(ii) conservative T／O

ＳＤＤ－１のｒｗ一同期法に用いられている。この方法では，オブジェクトにタイムスタンプは不要である。そのかわりに，各ＤＢＳは，各トランザクションごとのキューを持つ。各トランザクションからの演算は，発進順に各ＤＢＳに到着するものとする。

従って，各ＤＢＳで，ある演算が将来再スタートを起しそうな時，再スタートの原因になる他のトランザクションの演算を先に実行させてしまうことができる。例えば，Ｒ＝read(X)を実行する時，各キューを調べて，　ｓ（Ｒ）　よりも小さいタイムスタンプをもったwriteがあれば，Ｒをｂｕｆｆｅｒする。

トランザクションをクラス分けすることによって，各トランザクションクラスごとに競合するクラスを限定する方法もとりこまれている。

Ｔ／Ｏ方式では，タイムスタンプの管理が問題である。タイムスタンプを格納するためのオーバヘッドが生ずることから，不要な古いタイムスタンプを"忘れる"機能が必要である。タイムスタンプは，各オブジェクトＸと，これに対するＷ－ｓ　とＲ－ｓ　を表として格納することによって管理される。先に述べたように，ＳＤＤ－１の方法では，この表は不要である。

Ｔ／Ｏ方式は，２ＰＬで問題となったデッドロックが生じない利点がある。

c．２ＰＬとＴ／Ｏ

ｒｗ，ｗｗ同期は，トランザクションの全順序を保持する限り独立して扱かえる。可能な同期手法の組合せは，表4.3－1のように４８通りある。２ＰＬとＴ／Ｏとの混合した形態では２ＰＬにおけるｇｒｏｗｉｎｇ　ｐｈａｓｅ の終り（最初のｕｎｌｏｃｋ演算の直前）（ｉ，ｅ　ｌｏｅｋｅｄ　ｐｏｉｎｔ）に，トランザクションのタイムスタンプを与える。

表4.3－1

Method	rw technique	ww technique	Method	rw technique	ww technique
1	Basic T/O	Basic T/O	1	Basic 2PL	Basic T/O
2	Basic T/O	Thomas Write Rule (TWR)	2	Basic 2PL	TWR
3	Basic T/O	Multiversion T/O	3	Basic 2PL	Multiversion T/O
4	Basic T/O	Conservative T/O	4	Basic 2PL	Conservative T/O
5	Multiversion T/O	Basic T/O	5	Primary copy 2PL	Basic T/O
6	Multiversion T/O	TWR	6	Primary copy 2PL	TWR
7	Multiversion T/O	Multiversion T/O	7	Primary copy 2PL	Multiversion T/O
8	Multiversion T/O	Conservative T/O	8	Primary copy 2PL	Conservative T/O
9	Conservative T/O	Basic T/O	9	Centralized 2PL	Basic T/O
10	Conservative T/O	TWR	10	Centralized 2PL	TWR
11	Conservative T/O	Multiversion T/O	11	Centralized 2PL	Multiversion T/O
12	Conservative T/O	Conservative T/O	12	Centralized 2PL	Conservative T/O

Method	rw technique	ww technique	Method	rw technique	ww technique
1	Basic 2PL	Basic 2PL	13	Basic T/O	Basic 2PL
2	Basic 2PL	Primary copy 2PL	14	Basic T/O	Primary copy 2PL
3	Basic 2PL	Voting 2PL	15	Basic T/O	Voting 2PL
4	Basic 2PL	Centralized 2PL	16	Basic T/O	Centralized 2PL
5	Primary copy 2PL	Basic 2PL	17	Multiversion T/O	Basic 2PL
6	Primary copy 2PL	Primary copy 2PL	18	Multiversion T/O	Primary copy 2PL
7	Primary copy 2PL	Voting 2PL	19	Multiversion T/O	Voting 2PL
8	Primary copy 2PL	Centralized 2PL	20	Multiversion T/O	Centralized 2PL
9	Centralized 2PL	Basic 2PL	21	Conservative T/O	Basic 2PL
10	Centralized 2PL	Primary copy 2PL	22	Conservative T/O	Primary copy 2PL
11	Centralized 2PL	Voting 2PL	23	Conservative T/O	Voting 2PL
12	Centralized 2PL	Centralized 2PL	24	Conservative T/O	Centralized 2PL

d．信頼性制御

ＤＤＢＳの障害としては，ノードの障害とネットワーク分割とがある。ノードの障害は，そのＤＢＳの障害である。ＤＤＢＳでは，データを冗長化させることによって，これに対応できる。ノードが回復した時，このＤＢＳ内の冗長コピーを，他のコピーの値と同一にする（converge）必要がある。ＤＢ全体が冗長化されている時には，正しいノードからのＤＢの完全再ロード方法と，他のノードにあるログ内の演算の実行による方法とがある。前者は，通信オーバヘッドの点から現実的ではない。１つの方法は，コピーを順序づける（e．g χ_1, χ_2, ……, χn と順序づける）。トランザクションのアクセス演算は，常に χ_1 に対してなされ，この演算は順々に χ_2, χ_3, ……χn と伝わっていく。１つ前のノードに，更新のログを備えておき，これによって復旧し，他のコピーと値を同一にする。

ネットワーク分割〔ＹＡＭＡＨ８０〕は，互いに通信不能なノードグループが複数存在し，各グループは正常に動作出来る時に起こる。分割の検出と，復旧方法が問題である。解の１つは，分割が発生したら処理を全て止めてしまうものである。他は，１つだけグループを正常に動作させ，他は止めてしまうものである。後者では自分のいるグループが主なのか，従（止める）なのかを決めねばならない。グループのノード数が，全ノードの過半数の時，主とする方法がある。復旧は，ノード障害と同様である。検出で復旧には，ノード間での通信が必要となり，通信遅延が，パフォーマンス上問題である。

4.3.6　ホストとのインタフェース

ＲＤＢＭのホストとのインタフェースを向にするかは重要である。ＲＤＢＭが，アーキテクチャとして，直接実行できるレベルを，ホストインタフェースは示している。インタフェースの問題としては，

(i) 言語レベル

(ii) ホストとＲＤＢＭとの結合方式

の２点がある。

(i) ユーザ高水準言語

ユーザの高水準言語（ｅ．ｇ ＳＥＱＵＥＬ，ＱＵＥＬ）を，インタフェースとする。この時，ＲＤＢＭは，この言語問合せを入力として，ＳＶＣが最適な関係代数演算シーケンスを生成する必要がある。ホストとＲＤＢＭとは，問い合わせと，結果データの転送だけなので，高速な結合は不要である。

(ii) リレーションに対する関係代数プログラム

ユーザの高水準問合せから，ホストは，リレーションにもとづいた関係代数のシーケンス（ｔｒｅｅ）を生成する。ＲＤＢＭは，これから，セグメントにもとづいた関係演算のシーケンスを生成しストリーム処理を行う。この時，ユーザ問合せから関係代数生成における最適化と，ＲＤＢＭでの最適化との適合性が問題である。前者の最適化が，後者のものを十分に反映できなければならない。ＤＤ／Ｄが共有されねばならない。

ホスト－ＲＤＢＭ結合は，(i)と反様である。

(iii) セグメントに対する関係演算

セグメントに対する関係演算を，１つづつＲＤＢＭにもたし実行させる。ユーザ問合せから，これへの変換は，ホストが行う。

ＲＤＢＭでは，最終的に(i)の形態が望ましい。中間的には，(iii)の形態になると思われる。

－参考文献－

1) 〔ＢＥＲＮＰ７８〕

Ｂ Ｂｅｒｎｓｔｅｉｎ，Ｐ．Ａ．，Ｇｏｏｄｍａｎ，Ｎ．，Ｒｏｔｈｎｉｅ，Ｊ．Ｂ．，ａｎｄ Ｐａｐａｄｉｍｉｔｒｉｏｕ，Ｃ．Ａ．"Ｔｈｅ ｃｏｎｃｕｒｒｅｎｃｙ ｃｏｎｔｒｏｌ ｍｅｃｈａｎｉｓｍ ｏｆ ＳＤＤ－１：Ａ ｓｙｓｔｅｍ ｆｏｒ ｄｉｓｔｒｉｂｕｔｅｄ ｄａｔａｂａｓｅｓ（ｔｈｅ ｆｕｌｌｙ ｒｅｄｕｎｄａｎｔ ｃａｓｅ），" ＩＥＥＥ Ｔｒａｎｓ．Ｓｏｆｔｗ．Ｅｎｇ．ＳＥ－４，３ Ｍａｙ １９７８ １５４－１６８．

2) 〔ＤＡＴＥＣ７３〕

Ｄａｔｅ，Ｃ．Ｊ．，"Ｉｎｔｒｏｄｒｕｔｉｏｎ ｔｏ Ｄａｔａｂａｓｅ Ｓｙｓｔｅｍｓ" １９７３

3) [ESWAK 76]

Eswaran, K.P., et.al, "The Notions of Consistency and Predicate Locks in a Database System," CACM, Vol. 19, Nov. pp 624-633

4) [GRAY J75]

Gray, J., et.al, "Granularity of Locks in a Shared Database," Proc. of the 1 st VLDB, 1975, pp. 428-451

5) [GRAY J78]

Gray, J., et.al, "Notes on Database Operating Systems," in Operating systems — on advanced course, New York, 1978

6) [GRAY J81]

Gray, J., et.al, "The Rocovery Manager of the System R Database Manager," ACM Comp. Survey, 1981

7) [HEVNA 78]

Hevener, A.R., et.al, "Query Processing on a Distributed Database," Proc." Proc. of the Berkeley Workshop, 1978, pp. 91-107

8) [FAGIR 78]

Fagin, R., "On an Anthorization Mechanism," ACM TODS, Vol. 3, №3, Sept. 1978 pp. 310-319

9) [HELDG 75]

Held, G., et, al., "INGRES-A Relational Database System " AFIPS Conf. Proc., May 1976, pp. 407-416

10) [CHAMD 76]

Chamberlm, D. D., et. al., "SEQUEL 2 : A Unified Approach to Data Definition, Manipulation, and Control," IBM Journal of R&D., Vol. 20, №6, Nov. 1976

11) [HEROC 80]

Herot, C.F., "Spatial Management of Data." ACM TODS, Vol.5, №4, Dec. 1980, pp. 493-514

12) [YAMAH 80]

Yamazaki, H., et. al., "A Graph Theoretic Approach for Fault Detection and Recovery in a Distributed Database," ICCC '80,

pp. 237-242

13) [TAKIM78]

Takizawa, M., et. al, "Resource Integration and Data Sharing on Heterog neous Resource Sharin ," Proc. ICCC '78, Kyoto, Sept. 1978

14) [TAKIM 79]

Takizawa, M. and Hamanaka, E., "The Four-Schema Concept as The Gross Architecture of Distributed Database and Heterogeniety Problems," JIP of IPSJ, Vol. 2 №3, Nov. 1979. pp. 134-142

15) [TAKIM 80]

Takizawa, M. and Hamanaka, E., "Query Translation in Distributed Database," Proc. of the IFIP '80, Tokyo-Malbourne, Oct. 1980, pp. 451-456.

16) [SELIP 79]

Selinger, P.G. et. al., "Access Path Selection in a Relational Database Management," ACM SIGMOD, 1979, pp. 23-24

17) [STONM 76]

Stonebraker, M., et. al, "The Design and Implementation of INGRES," ACM TODS, Vol. 1. №3, 1976, pp. 189-222

18) [ZLOOM77]

Zloof. M., "Query-By-Example; A Data Base Language." IBM Systems Journal, №4, 1977, pp.324-343

19) [ASTRM79]

Astrahan, M., et. al,, "System R; A Reletional Database Management System," IEEE Compon 79, Feb. 1979, pp. 72-76

5. プロトタイプデータベースマシンの概要

5.1 システムイメージ

5.1.1 基本構想

プロトタイプデータベースマシンは，

- データストリーム処理
- データフロー制御
- オブジェクト指向

の3つの基本概念にもとづく，高並列度の関係データベースマシンを目指す。

大容量データベースを扱うためには，記憶階層の最下層メモリに可動ヘッドディスクを用いることが不可欠であるとの考えに立つ。関係，および処理速度向上のために用いられる転置関係等は，セグメント化されてディスクに格納される。どの関係が，どのようにセグメント化され，どこに格納されているかはデータ記述子によって管理され，オブジェクト指向アーキテクチャによって，セグメントの管理が行われる。セグメントの読み出しは，データフロー制御によるセグメントの先行アクセスとプリステージングの導入により，スループットが上げられる。ステージングされたセグメントは，各種機能モジュール中を通過して転送されて行く間に，転送に重畳した処理が行われ，関係代数演算が実行されていく。転送に重畳した処理をデータストリーム処理と呼ぶ。

5.1.2 種々の考え方

基本構想を定めても，そのようなデータベースマシンの実現方式は種々存在する。そうした中で，アーキテクチャの検討において重要なポイントが2つある。

その1つは，データベースの属性値はもともと可変長であるが，これをどのように取り扱うかという点である。これには，大きく分けて，次の3つの方式が考えられる。

(1) 可変長データをそのまま処理する。

(2) 属性毎に，データの最大長を決め，各データをこの長さに固定して処理する。

(3) 可変長データを固定長の符号に変換して処理する。

1は処理モジュールや，結合ネットワークの設計を難しくする。当面は，2の方式を採用するが，従来，オーバー・ヘッドが大きすぎるとされていた3の方式が，そうでもないとの新しい考え方も発表されており〔TANA82〕，この方式についての検討も行う必要がある。

第2のポイントは，転置ファイルを導入するか否かである。一般に転置ファイルの導入は，検索

速度を向上するが，更新の際のオーバー・ヘッドを増すとされている。しかし，これは単一プロセッサでの処理経験に基づいて言われていることであり，関係と，転置関係の並列処理についても検討する必要がある。

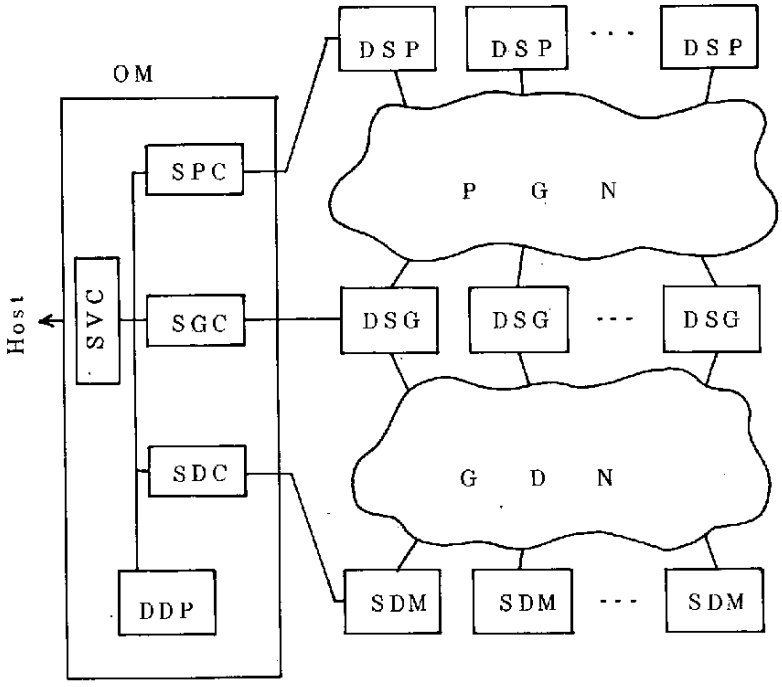
5.1.3 プロトタイプマシンのイメージ

当面のプロトタイプマシンのイメージを図5.1.1に示す。

ＳＤＭ（Secondary Data Manager）は，関係をセグメント化して格納し，タプルの連想検索，連想更新の機能を持つ２次記憶システムである。ＤＳＧ（Data Stream Generator）は処理内容に応じて，タプルの適当なクラスタリングを行うためのステージングバッファである。ＤＳＰ（Data Stream Processor）はクラスタリングされたタプル集合を，データストリーム処理でソート，マージし，重複データの除去や，統計処理等を行い，再び，ＤＳＧを用いてクラスタリングし直す機能を持つ。セグメント化された関係の管理は，ＤＤＰ（Directory Dictionary Processor）が行う。セグメントのアクセス，ＳＤＭ内のバッファへのプリステージングはＳＤＣ（Secondary Data Controller）によって制御される。ＤＳＧ群，ＤＳＰ群は各々ＳＧＣ（Stream Generation Controller），ＳＰＣ（Stream Processing Controller）によって制御される。検索要求や更新要求の処理は，ＳＶＣ，ＳＤＣ，ＳＧＣ，ＳＰＣがデータフロー制御で互いに制御しあい，ＤＳＰ，ＤＳＧ，ＤＳＭを制御することによって実行される。

プロトタイプデータベースマシンではＤＳＰの個数は１００程度，ＤＳＧは２００程度のバンクより成る。ＳＤＭは１台当り数ＧＢの容量を持ち，ＳＤＭの台数は，データベースの規模によって決まる。

最近，北海道大学は，ネットワークとソータを融合したＳＤネットワークを用いたアーキテクチャを提案している〔ＴＡＮＡ８２〕。符号化／復号化の採用，転置関係の利用と専用サブシステムの採用等，以下に詳細に示すアーキテクチャでは採用していない方式を採用している。その良否については，現時点では充分な評価資料がないこともあり，プロトタイプデータベースマシンの基礎研究においては，併せて検討を要する。



SVC : Supervisor Controller
SPC : Stream Processirj Controller
SGC : Stream Generation Controller
SDC : Secondary Data Controller
DDP : Directory Dictionary Processor
} OM

OM : Object Manager
DSP : Data Stream Processor
SDM : Secondary Data Manager
PGN : Processor Generator Network
GDN : Generator Disk Network
} Network

図 5.1.1　プロトタイプデータベースマシンのイメージ

ー参考文献ー

1) 〔TANA82〕 田中譲：データフロー制御データストリーム処理方式データベースマシン（基本構想），情報処理研資，データベース管理システム28－3，1982

5.2 アーキテクチャ

5.2.1 はじめに

A. 性能要求

3章で述べた如くデータベースの応用は多岐にわたり，これら全てに対して充分な性能を有するデータベースマシンは現在，存在しない。第5世代システム，知識ベースマシンの構成要素となるデータベースマシンとしての要求を明確にすることが必要である。現時点では，知識の格納方式，管理方式及び操作方式について充分な知見が得られておらず，関係代数をそのインタフェースとする関係代数マシンによってその方式を模索する段階にあるといえる。従って，ビジネスアプリケーションの如く，定型的な処理のみを想定したマシンでは充分とはいえない。複雑な知識操作を高速に実現するためには多段の関係代数木を効率よく実行する機能が必要と考えられる。すなわち，セレクションやリストリクションによる単純な条件検索ばかりではなく，ジョイン，プロジェクション，ディビジョンなど，処理負荷の重い関係代数オペレータを多く含む問い合わせを高速に実行出来る必要がある。またリレーションが小さい場合には，その操作は容易であるが，対象とする知識はエキスパートシステム等では容量が限られているものの，応用分野を広くすると急激に増大すると考えられる。従って，データベースマシンとしては，大容量のデータベースを高速に操作できる必要がある。

以上の如く，大容量のデータベースに対して，非定型的な負荷の重い関係代数操作を，効率よく実現することが要求される。

B. 抽象アーキテクチャ

前節で示される如き高い性能を実現するためには，データ操作系，およびデータ記憶系全体が統合化され，高速の並列処理により関係代数演算を実現する必要がある。

プロトタイプデータベースマシンの全体象を図5.2.1に示す。これは，今後の研究により変更されることがありうる。

全体は，4つの基本構成要素

DSP　Data Stream Processor

DSG　Data Stream Generator

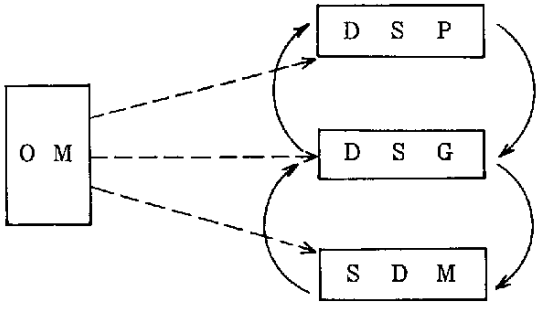
SDM　Secondary Data Manager

OM　Object Manager

からなる。

リレーションは，2次記憶系であるSDMに格納されており，処理に必要な部分だけがDSG

にステージングされる。その後，ＤＳＧはＤＳＰ群に対してデータストリームを生成し，ＤＳＰは適当なストリームに作用することによって新しいストリームを生成し，ＤＳＧに返す。多数のＤＳＧ，ＤＳＰが並列動作することによってこれらの操作を高速に実現出来る。そして，ＤＳＰは処理を繰り返しながら，結果データストリームを生成してゆく。ＤＳＰはＤＳＧからのデータ流に対して遅れることなく処理を行い，すなわちＯ（ｎ）の処理を実現し，これにより，ソースＤＳＧから，デスティネーションＤＳＧに対し，データ流を流す間に処理を終える。ソースとデスティネーションの対応は必ずしも一対一ではない。この対応が固定的であるならば，従来のCellular Logic Type ＤＢＭで実現出来ることとなり，従って，セル間の相互作用が，このマシンでは，ダイナミックなストリーム生成によって実現されることになる。１つのリレーションは複数のＤＳＧ上にのせられることになり，従って複数のデータストリームに対し，多くのＤＳＰが操作し，かつ，それらがＯ（ｎ）で処理完了するため，処理時間はＤＳＧのページサイズによって決定され，基本的にはリレーションの大きさに依存しない処理速度が得られる。ＯＭは，ストリームの管理，Name Mapping，ステージングコントロールなど問い合わせの実行管理を行う。



Abstract Architecture of
Relational Database Machine

図5.2.1

5.2.2 処理方式

A．データストリーム処理

データベースマシンではあくまでデータ流が主体であり，この流れにそって，そして流れを乱すことなく必要とされる処理が施されていくことが望ましい。この発想は，７０年代初頭の先駆的なマシン以来，引き継がれてきており，自然な考え方といえる。CASSM，RAPでは磁気ディスクを記憶媒体としメモリからのデータ流の通り道に簡単なプロセッサを置き，プロセッサ内の

バッファをデータが通り過ぎてゆく間に処理が行われる。回転型メモリの媒体は，ＲＡＰ２ではＣＣＤに，ＥＤＣでは磁気バブルに置き換えられたものの，処理方式自体は何ら変化していない。データベースマシンのアーキテクチャ上の分類では，ＤＳ型とＩＳ型とに分けたが，方式上は，ＣＡＦＳであれ，ＤＢＣのＴＩＰであれ，ＲＡＰであれ，基本的には"Data Stream Manipulator"として位置づけられる。このように，データベースマシンでは，データストリームが"Data Stream Manipulator"を流れていく間に順次処理されていくと考えるデータストリーム処理が基本処理方式となる。

データストリームとはデータの流れであるが，ビットシリアルなデータ流，バイトシリアルなデータ流など種々の形態があり得る。さらに，"Data Stream Manipulator"にとっては，流れに対する処理単位が存在する。ディスクからのデータ流に対する処理では，誤りチェックが必要であり，少なくとも１ブロックのバッファは必要となる。流れに対する処理単位には種々のレベルが考えられ，ＲＡＰ等のマシンではその単位が比較的小さく，データ流に密着した処理がなされているのに対し，ＤＩＲＥＣＴマシンでは，１６ＫＢの１ページがデータ流の単位とみなすことも出来る。ＤＩＲＥＣＴはページ内ではＬＳＩ１１を用いて処理しており，必ずしもデータ流に追従出来ているわけではない。ＫＵＮＧのマシンではページ内処理をシストリックアレイによって行っており，ビットレベルのデータストリーム処理が実現されている。

データストリーム処理ではデータストリームが主体であり，従ってそのデータ流をいかに発生するかという点が重要になる。これまでの多くのマシンではデータ流は回転型メモリの１トラックに対応し，記憶媒体に固定化されたものとなっていた。このようにデータ流が固定的であることは処理方式上，大きな制限が課せられることになる。ジョインやプロジェクションをＲＡＰ上で実現しようとするとほとんど効果がないことは周知の事実であり，これは記憶媒体に固定化されたデータストリームに対するフルスキャンによる処理方式の限界とみなすことが出来る。ＣＡＦＳやＣＡＳＳＭに於けるHashed Bit Array方式は確かに有効な手法ではあるが，基本的には篩い落としの効果しかなく，実際のジョイン処理はホストマシンで行わなくてはならない。3.3節で評価した如く，これは単一プロセッサの場合かなり大きい負荷となる。記憶媒体に固定化されたデータ流に対して，on the fly処理だけで実現出来る機能は限られている。即ち，データ流が能動的主体となり，プロセッサは受動的な存在ととらえられる環境では，種々の処理を効率よく行うためにはデータ流自体を処理に応じて動的に生成することが必要になる。

B．クラスタリング技法を用いたデータストリーム処理方式

本マシンではデータ流を発生する主体をData Stream Generator（ＤＳＧ）と呼んでおり，これによってデータ流を動的に生成する。以下，その処理技法について説明を行なう。

Data Stream Manipulatorに相当するData Stream Processor（ＤＳＰ）はＤＳＧによって生成されたデータ流を取り込みながら処理を施し，ＤＳＧに送り返す。ＤＳＰはＤＳＧからのデータ流に殆ど遅れることなく処理を行い，データ流を乱さない様にする必要がある。図5.2.2に示される如く，ＤＳＧがリレーションを構成するタプルからなるデータ流を押し出し始めると，ＤＳＰはその流れに対して処理を施し，処理結果からなるデータ流をＤＳＧに発生する。このようにして処理が進み，ＤＳＧが当該オペレーションに必要なデータを送出し終えると，ほとんど遅れることなく処理結果は再びＤＳＧ上に生成されており，ＤＳＧは次のオペレーションに対して同じようにデータ流を押し出すことが出来る。これを繰り返し，データ流を循環させることにより，一連の関係代数オペレーションからなる問い合わせを処理してゆくことが出来る。

マシンは複数台のＤＳＧ，ＤＳＰから構成され，ＤＳＧ群から生成される複数本のデータ流に対して，活性化されたＤＳＰ群が処理を施すことになる。この際，ＤＳＧが発生したデータ流は一旦，ＤＳＰの中に取り込まれ，ＤＳＧはＤＳＰによって新たに生成されたデータ流を入力する

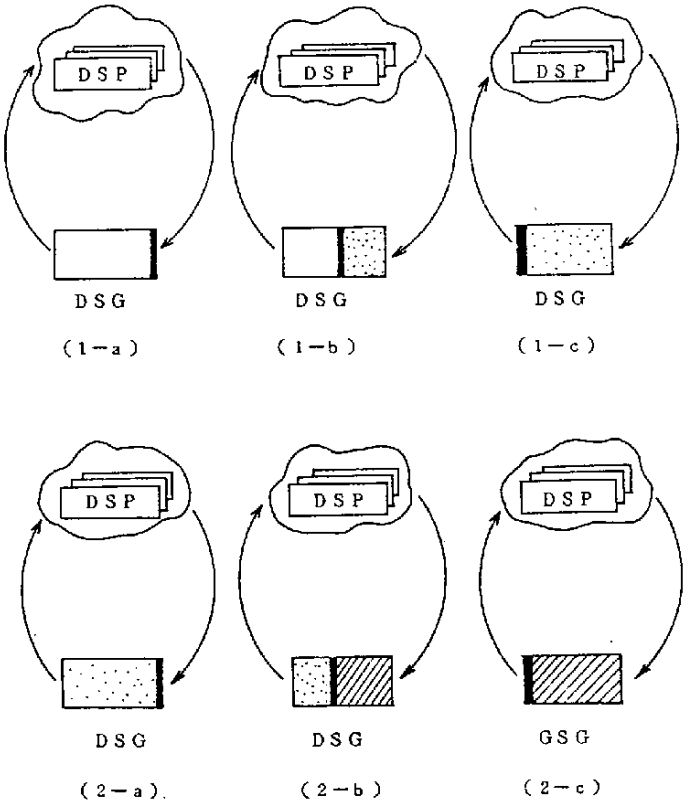
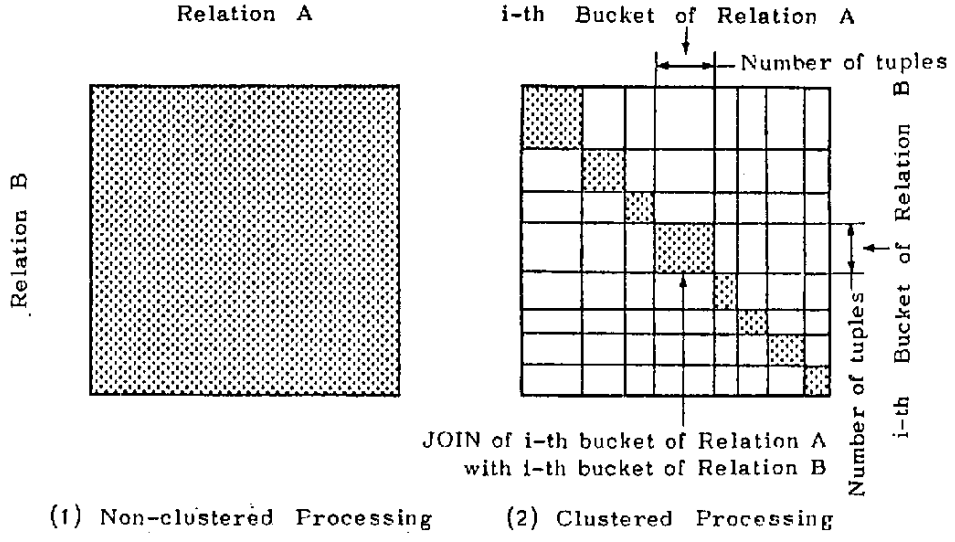


図5.2.2　ＤＳＧとＤＳＰによる処理形態

ことになる。従って，これまでのセルラロジックタイプマシンの如く，データ流がセルに固定化されることはなく，動的に変化することになる。従来機におけるセル間の交信が本マシンでは動的なストリーム生成によって実現されている。

以上の如き円滑なデータストリーム処理を実現するために，本マシンではクラスタリング技法を導入している。図5.2.3に示される如く，ジョインやプロジェクションなどでは当該リレーションがクラスタリングされていない場合には，両リレーションのCardinalityの積に比例した処理時間が必要となるが，もし，ジョインあるいはプロジェクションアトリビュートに関してクラスタリングされている場合には，クラスタリングによって生成された各バケットは互いに独立であり，バケット間でマッチングがとれる可能性はないため，処理をバケット内で閉じた形にすることが出来る。



Clustering Effects By Hashing In JOIN Operation

図5.2.3

すなわち，ジョインの場合，両リレーションのCardinalityをN，Mとすると，O（N×M）の処理負荷を，S個のバケットにクラスタリングすることにより，O（$\sum_{i=1}^{S} n_i \cdot m_i$）の負荷に落とすことが出来る。図5.2.3では影の部分が処理負荷を表わし，クラスタリングにより大きく負荷を減少出来ることがわかる。クラスタリングはHashにより実現出来る。このように，クラ

スタリングを導入することにより，パケットレベルでO(s)の処理に変換出来ることが示されたが，さらにパケット内での処理をO（mi×ni）ではなくO（mi+ni）時間で処理出来るならば，パケットレベルでO(s)，パケット内でO（mi+ni）の処理が実現され，全体として流れに追従した高速なデータストリーム処理が可能となることがわかる。パケットの大きさは限られておりページ内でのO(n)処理は基本的にはリレーションがソートされていれば実現可能である。またO(n)ソータはすでにいくつか発表されている。シストリックバブルソータ，パイプラインヒープソータ，パイプラインマージソータなどがそれである。パケット内処理のO(n)化については，シストリックアレイを始め，いくつかの方式が考えられるが，ここではO(n)ソータを用いることにする。これはソート自体が基本的な演算と考えられ，ジョイン，プロジェクション，ディビジョン，イメージ処理など，いずれに対しても適用出来るからである。なお，DSGからのデータ流はシリアルに送られてくるため，これ以上高速に処理する必要はない。

以上の如きHashとSortにもとづく関係代数処理は，DSG，DSPを用いて次のように行われる。クラスタリングによって生成された各パケットは互いに独立に処理可能であり，これらをそれぞれ異なるDSPに割り当てることによって，DSPは互いに交信することなく処理をすすめられ，パケットの多重処理が可能となる。そして，DSPにパケットを効率よく割り付けるため，DSGはパケットシリアルにデータ流を生成することが望まれる。即ちDSGはパケットシリアルなデータ流を発生し，一方DSPは適当なパケットを取り込み処理を施せばよい。DSP内での処理はパケットの大きさに比例した時間，すなわち，パケットの入力に要した時間とほぼ同じ時間で完了するため，データ流を殆ど乱すことなくデータストリーム処理を実現できる。パケットシリアルなデータ流の生成により，DSPはパケットレベルのパイプラインを構成する事が可能になる。従って，DSPはパケット数台用意する必要はない。図5.2.4にDSGが2台，DSPが4台の場合についてデータストリーム処理の様子を示す。なお，この場合，DSPは1つのパケットの処理中には次のパケットの入力は出来ないと仮定しているが，DSPがパケットの連続処理可能な場合にはこの半分ですむことになる。

DSGでは当該オペレーションに関するパケットIdを各タプル対応に保持しており，この値をもとにパケットシリアルなデータ流を発生する。このパケットIdの付加自体はon the fly処理が可能であり，2次記憶からDSGへのステージング時，および，結果リレーションの生成時に行える。多段の関係代数木で表現される複雑な問い合わせに対しては，当該オペレーションの実行中に，DSPが次オペレーションに関するパケットIdの生成を行い，DSGからの当該オペレーションに関するパケットシリアルなデータ流はDSPによって次オペレーションに適したデータ流に変換されることになる。もっともこの段階では，DSGに返されるデータ流はパケッ

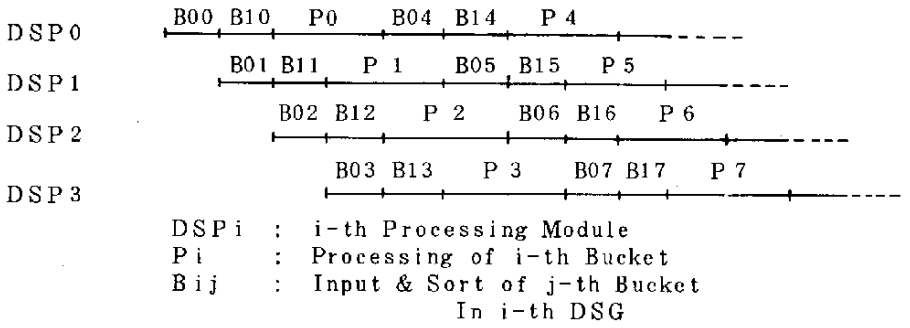


図5.2.4 Pipeline Overuiew of Bucket Processing

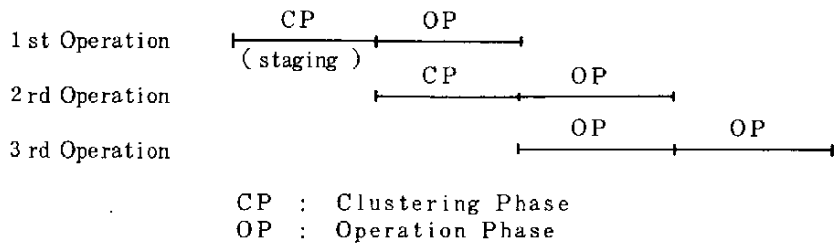


図5.2.5 Operator Level Pipeline Processing

トIdに関してランダムであり，DSGはデータ流生成時にバケットシリアルにする必要がある。このように，クラスタリングを用いたデータストリーム処理では，次オペレーションに関するクラスタリング処理を当該オペレーションの真処理に重畳させることが可能となり，オペレータレベルのパイプラインを実現している。これにより，図5.2.5で示される如く，関係代数演算を連続して実行出来ることになる。

以上の如く，本マシンでは動的なクラスタリング機能にもとづくデータストリーム処理を基本方式としており，これにより一連の関係代数演算を高速に実現する。

C. 種々の関係代数演算処理

クラスタリング技法を用いたデータストリーム処理により各種関係代数演算子を高速に実行することが出来る。

a. ジョイン

2つのリレーションのジョインアトリビュートに関してHashを施すことにより両リレーションを互いに独立なバケットに分割する。同一バケットIdを有するバケット間でしかジョインがとれることはなく，処理負荷はバケットレベルでO(s)に落とすことができ，各DSPは対応するバケットを取り込み処理を行う。DSP内では，O(n)ソータによって生成されたソート

出力に対してジョインを行えばよい。各々のリレーションに対して別々にバケットを生成する必要はなく，２つをまとめて取り扱うことも出来る。ソートはジョインアトリビュートのあとにリレーションIdを付加しておけばよい。

b．プロジェクション

リレーション内の重複タプルを除去する作業がプロジェクションの処理負荷を重くしている。プロジェクションアトリビュートに関してHashを施し，当該リレーションを独立なバケットに分割する。異なるバケットIdを有するバケットに同一タプルが分散されることはないため，各ＤＳＰはバケットを取り込み独立に重複除去処理を行える。ＤＳＰ内ではO(n)ソータによって生成されたソート出力に対して容易に重複を取り除くことが出来る。

c．ディビジョン

被除リレーションを，商アトリビュートに関してHashを施し，互いに独立なバケットに分割する。これにより，商となるアトリビュートに関してクラスタリングされ，ＤＳＰ内でソートすることにより除リレーションが含まれるかどうかチェックすることが出来る。ここで除リレーション自身は比較的小さいと仮定している。別の手法としては，除リレーションと被除リレーションに対しあらかじめインプリシットジョインを施した後，商アトリビュートに関してクラスタリングを行い，商アトリビュートのイメージセットに属するタプル数を数えることによって処理することも可能である。

d．集合演算

Intersection及びDifference は２つのリレーションの全アトリビュートに関してHashを施し，基本的にはジョインと同様に取り扱うことが出来る。Unionでは重複タプルを除去すればよく，プロジェクションと同様に処理することが出来る。

e．Aggregate Operation

3.3.節で評価に用いた３番目のＱ３の如き問い合わせに対しても，by clauseのアトリビュートに関してHashを施し，ディビジョンと同じようにして取り扱うことが出来る。

f．セレクション，リストリクション

セレクション，リストリクション演算丈からなる単純な問い合わせはＳＤＭ上で直接処理する事が出来る。そうでない場合には，一般に，この演算は関係代数木における下降ルールに従ってなるべく早く処理をする方が望ましい。これにより対象データストリームを短くすることが出来るからである。従って，ベースリレーションに対するセレクションはデータスージングの際に処理することとする。下降出来ないものに対しては，その前の演算に重ね合わせて処理する。また，アトリビュート選択だけのプロジェクションも同様に取り扱うものとする。クラ

スタリングは，交差的演算に対して適用されるものであり，1タプル内で処理が完結するセレクション演算は，元来データ流の順序と無関係に処理可能であるため，交差的演算と同時に行えることは明らかである。

D．2次記憶とデータステージング

本マシンでは最下層メモリを磁気ディスクとし，2次記憶管理を行う部位をSecondary Data Manager（ＳＤＭ）と呼ぶ。ＤＳＰを用いてデータ処理を行うには，まずＳＤＭから所要データをＤＳＧ上にステージングする必要がある。この後，すでに述べたクラスタリングにもとづくデータストリーム処理がなされることになる。データステージングの際には，フィルタリングとクラスタリングの2つの処理がなされる。すなわち，ベースリレーションに対して，セレクション，リストリクションを施し必要なタプルのみを選択し，さらに，以後の条件判定に必要なアトリビュートのみをそのタプルから取り出すフィルタリング処理がなされる。これは，従来のＣＡＦＳやサーチプロセッサに見られるＤＳ型機能と同じものであり，これを用いて処理対象を絞り込むと同時に，ＳＤＭ側でこの操作行うことによってＳＤＭとＤＳＧ間のデータ転送量を削減している。またデータステージング時には，選択されたタプルに対して次オペレーションに関するクラスタリングがなされる。ステージングとクラスタリングを重畳させることにより，ＤＳＧはステージングされた後，即座にデータ流を生成出来る。一般に，関係代数木を実行する際，ベースリレーションが参照される場合には，一時点前に当該オペレーションに対するクラスタリングを施しながらステージングすればよい。

また，セレクション，リストリクション文からなる単純な問い合わせは，ＤＩＲＥＣＴの様に一端ＤＳＧ上にステージングした後，ＤＳＰを用いて処理するのではなく，ＳＤＭからのデータにフィルターを通して直接ＤＳＧ上に結果を生成することが出来る。本マシンは，ＤＳＧ上のデータにＤＳＰが処理を施すという点ではＩＳ型のマシンであり，ＳＤＭ上で直接フィルタリングを行うという点ではＤＳ型のマシンと考えられる。ＭＰＣＳ型の本マシンでは，両機能を柔軟に使い分けることによって，負荷の重い複雑な問い合わせにも単純な問い合わせにも高い性能を示すと考えられる。

E．更新処理

単一リレーションに対する簡単な更新処理は，ＲＡＰ１やＣＡＳＳＭ同様，ＳＤＭ上で直接行うことが出来る。一方複数のリレーションにまたがる条件を含むような複雑な更新処理の場合には，更新すべきタプルを決定するのにジョイン等が必要とされ，このため，ＤＳＧ上にステージングした後ＤＳＰで処理することになる。このようにして更新すべきタプルが決定されると，ＤＳＰは当該リレーションが物理的にクラスタリングされているキーに関して，更新タプルの集合にク

ラスタリングを施す。即ち，ベースリレーションの更新はベースリレーションと更新タプル集合とのジョインによってその対応を決定でき，最終段階において，ＤＳＧはＤＳＰに対してではなく，ＳＤＭに対してデータ流を生成することになる。更新情報を得た各ＳＤＭは，その管理するリレーションに対して更新を行う。

また更新時にはセマンティックインテグリティに関するアサーションチェックを行う必要がある。これも簡単な場合にはＳＤＭ上で実現出来るが，複数のリレーションに及ぶ複雑なアサーションの場合にはＤＳＧ上に必要なリレーションをステージングして高速にチェックすることが可能である。アサーションのチェックは基本的には問い合わせ処理と同様に扱うことが出来る。

F．スタティッククラスタリングを用いた記憶構造と動的クラスタリング処理方式

これまでは，２次記憶媒体上ではジョインアトリビュートに関してスタティッククラスタリングがなされていない場合について考察してきたが，そうでない場合についても動的クラスタリング手法は同様に，適用可能である。タプルワイズな記憶形式では，スタティッククラスタリングに用いることの出来るキーはわずかであるのに対し，カラムワイズな方式ではカラム毎にクラスタリングを施すことができる。しかし，この場合には，タプルの横方向の対応関係をとるためには毎回ジョインが必要となる。そして，このジョイン処理に，先のダイナミッククラスタリングを適用することが出来る。カラムワイズなクラスタリングを行う場合の一例として以下のような方式が考えられる。リレーションの各アトリビュート毎に，タプルＩｄを付加して全体をバイナリリレーションに分解し，各々のバイナリリレーションに対して，アトリビュート値に関してクラスタリングしたものと，タプルｉｄに関してクラスタリングしたものの２つを設けることとする。そして，前者のファイルを操作するプロセッサをＴＳ，後者のそれをＴＭと呼ぶことにする。例えば，条件アトリビュートと出力アトリビュートの異なるセレクション処理を行う場合には，まずＴＳが当該アトリビュートに関する条件式をスタティッククラスタリングが施されたファイル上で評価し，対応するタプルＩｄの集合をＴＭに送る。ＴＭでは，タプルＩｄ同志のジョインが施され，対応する出力アトリビュート値をとり出し結果とする。この処理のタプルＩｄ同志のジョインでは，ＴＳから送出されるタプルＩｄに対し，動的にクラスタリングが施されることになる。また，ジョインを含む問い合わせの場合には，アトリビュート間の対応をとるために必要なタプルＩｄ間のジョインに加え，真のジョインに対応するアトリビュート値上でのジョインも必要となる。ここでも同様に動的クラスタリングがなされる。

このようにリレーションの縦方向分割を行うと，その対応をとるためにはジョインが必要となり，これには先に述べた動的クラスタリングを用いたデータストリーム処理が同じように適用できる。スタティックリラスタリングを行うからといって，動的クラスタリングが不要になるわ

けではない。上記のような分割されたバイナリーリレーションの各々のアトリビュートに関してクラスタリングされたファイルを設ける冗長な構成法では，片側のリレーションはすでにクラスタリングされている点が異なるだけである。スタティッククラスタリングと，動的クラスタリングを統合し，高い性能を得ることが期待される。

G．データベースのコード化とその処理方式

データベース操作の対象をレコード本体とするか，あるいはそれをエンコードしたコード化レコードとするかという２つの方向が考えられる。コード化を行い対象データ構造を固定化することによりマシンアーキテクチャはより簡素化することが可能である。例えば，ＲＡＰ１ではコード化されたデータベースを対象としていた。しかしながら，コード化データベースの処理では入出力時に常にエンコード，デコードの処理が必要となり，それだけオーバヘッドが増加することになるし，また，コード化表の管理が大きな負荷となり得る。ＣＡＦＳにおけるカップリングインデックスの如く一意な対応関係をもたせることにより，処理は高速，かつ，簡素化されるものの，その代償がコード化表の維持管理負荷という形で現われることになる。これは，２次記憶構造におけるインデックスの管理と同種の問題が生じていると考えられる。また，ジョイン操作のような場合には，ジョインアトリビュートに関して同じコード化がなされている必要があり，予期されていなかった場合にはジョイン出来ない事態が生ずることもありうる。このように，コード化することによりマシン内での処理構造は簡素化される反面，いくつかの問題も存在し，これらに関して今後詳細に検討を重ね，全体としての評価を行う必要がある。一方，レコード本体をそのまま操作する直接的な手法は，エンコード，デコードに関する問題はなく処理手順は簡単になるものの，ハードウェア構成という点では対象が一般化されるためより複雑になる可能性がある。

データベースマシン構築に際し，どちらか一方に限定する必要はないと考えられる。方式の適否はアプリケーションのタイプにも依存すると考えられ，ジョイン負荷が大きな割合を占める問い合わせが多いアプリケーションでは問い合わせ処理の高速化が重要であるのに対し，単純な問い合わせだけしか発せられない場合には変換オーバヘッドが問題になってくると考えられる。またアーキテクチャ上も，問い合わせのエンコード，デコードの処理自体は，コード化表とのジョインで表わされることになり，コード化処理方式自体は比較的自然な形で，コード化を意図しないマシンで実現できることになる。逆にコード化を意識したマシンでもデータ処理幅は限られているが処理をくり返すことによって比較的長さの大きい一般レコードに対する演算を実現できる場合もある。コード化は，関係データベースマシン上だけでの問題ではなく，知識ベースシステム全体との関係も考慮する必要があり，統合化された処理方式を採用することにより，高い性能

が期待される。

5.2.3 アーキラクチャ

A. 全 体 像

プロトタイプデータベースマシンの全体像を図5.2.6に示す。全体は大きく，DSP，DSG，SDM，OMから構成され，DSPとDSG間は，PGNネットワークによりDSGとSDMの間はGDNネットワークにより結合されている。OMは，さらに5つの構成要素SPC，SGC，SDC，SVC，DDPから成る。各略称は，次のような意味をもつ。

DSP	Data Stream Processor
DSG	Data Stream Generator
SDM	Secondary Data Manager
OM	Object Manager
SVC	Supervisor Controller
SPC	Stream Processing Controller
SGC	Stream Generation Controller
SDC	Secondary Data Controller
DDP	Directory & Dictionary Processor
PGN	Processor Generator Network
GDN	Generator Disk Network

以下，各構成要素について説明を加える。

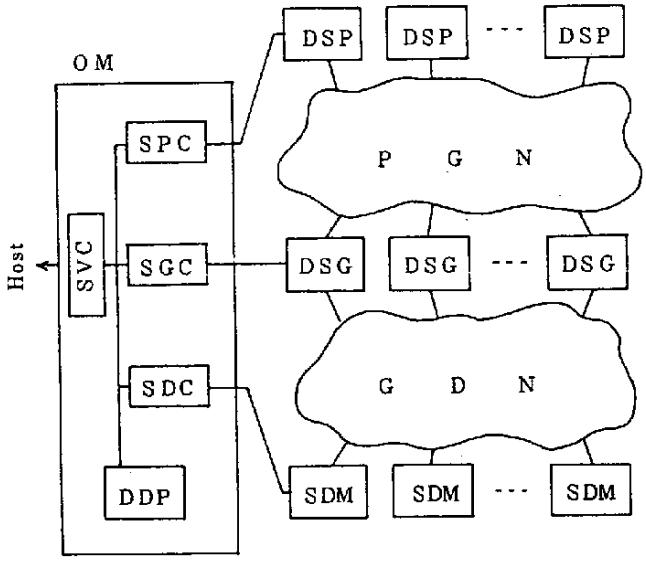


図5.2.6 Total Architecture of Relational Database Machine

B. Secondary Data Manager（SDM）

SDMは2次記憶系でのデータ管理を司る部位である。以下にその諸機能および構成について述べる。

a. 記憶構造

リレーションの物理的記憶媒体へのマッピングには，1タプルを1レコードとする一般的な方式や，アトリビュート毎のファイル構成をとるトランスポーズドタイプのものなど，いくつかの方式が考えられる。レコードという単位を定めた時点での記憶構造，あるいはファイル編成は，基本的には従来までに開発されてきた多くのファイル編成法，Sequential，Direct，Indexed Sequential，VSAM，Dynamic Hashなどを適用することが出来る。プライマリーキーに関しては，インデックス，あるいはハッシュを用いることにより高速にアクセスすることが可能であるのに対し，2次キーに対しては，レコード指向の構成では，交代インデックスを設けるなどの手法が考えられるが，物理的にはクラスタリングされていないため，大量データのアクセスを行う場合には直接適用出来るわけではなく，さらに整列化が必要となる。また，DBCで行っている如く，2次元クラスタリング程度までは，クラスタリングキーを拡大することによって適用可能であるが，多次元に拡張することは一般に困難である。

2次キーにインデックスを生成することはやめ，別のファイルとすることにより，擬似的にこの問題を解決出来る。ファイルのVertical Partitioning（トランスポーズドファイル）ではフィールド間の対応関係を論理ポインタ（タプルID）をもつことによって維持しつつ，フィールド毎の物理的なクラスタリングを実現している。しかし，この方式ではフィールド間の対応をとろうとする場合には，関係代数演算のジョインに相当するオペレーションが毎回必要となる。一般に，条件によるフィルタリングにおいて当該アトリビュートのデータはクラスタリングされていても，対応するフィールドの空間ではクラスタリングされているわけではなく，タプルIDによるジョインによって対応をとろうとする場合には，交代インデックスと同様の手間が必要となる。なお，この段階の操作は，アクセス量が比較的多い場合，タプルIdの重複が許されないため，ビットマップによりフルスキャンすることで実現することも考えられる。またファイル構成としては，アトリビュート値とタプルIdの2つに対して各々クラスタリングされたファイルを重複して維持し，条件によるタプル選択においては前者を，アトリビュート間の対応をとる場合には後者を用いる手法も存在する。以上の如く，記憶構造としてどのような形が最適であるかという問題，即ち，スタティックなファイル構成法の問題は，データベースの性質，問い合わせのタイプにも大きく依存すると考えられ，種々の方式を共用させることが望ましい。必ずしも全てのアトリビュートに関して完全に分割する必要はなく，

また無意味に長い，多くのアトリビュートから成るレコードを生成することも避けるべきであろう。ＳＤＭでは，最下層記憶媒体上においてこれらの構成をより効率よく実現するため，インデックス，ハッシュ，アトリビュート間セミジョインなどを，ハードウェア的に強化することが望まれる。

補助データ構造に関しては，ＩＳＡＭのオーバフローに対する弱点を，ＶＳＡＭでは，インデックスを可変にしバランス化させることによって解決し，また直接編成でのオーバフロー，あるいは，シノニムチェイニングによるアクセスタイムの低下は，ダイナミックハッシュ，バーチュアルハッシュなど，バケットスプリットを導入したノンオーバフロータイプのハッシュにより改善されている。ＳＤＭはこれらの技法を反映した記憶管理機構を導入し，更新，追加，削除等の操作に対し，速いアクセスと高いロードファクタを維持する必要がある。

補助データに対する記憶媒体は，ＤＢＣで採用している如く，磁気バブルやＣＣＤなど，より高速のアクセスデバイスを用いることが考えられるが，近年，半導体技術の進歩はめざましく，ＲＡＭの利用がより現実的である。また補助データに対するアクセスパターンはほぼ一定であり，従って高度にファームウェア化された専用プロセッサを開発することが可能である。

b．フィルタリング機構

データベースマシンでは，その主体はデータ流であり，その流れにそって処理が施されて行く。このようなデータストリーム処理ではデータ流の長さが処理負荷を決定し，従って，不要なデータは出来る限り下層の記憶上でろ過し，必要なデータだけから成るストリームを構成することが，システムのデータトラヒックを軽減し，パフォーマンスを向上する上で重要と考えられる。プロトタイプデータベースマシンではディスクにフィルター機構を付加し，セレクションやプロジェクションなど１タプル内で処理可能な操作はその場で行い，条件を満たすタプルのみを上層の記憶系に出力することとする。不要アトリビュートの除去などタプルの整形もこの段階で行う。このような，条件検索およびタプル整形専用のプロセッサをここではフィルタープロセッサと呼ぶことにする。また，ＣＡＦＳに見られるHashed Bil Arrayによる篩い落とし操作も，フィルターの拡張機能ととらえることが出来，カップリングインデックスを用いた完全セミジョイン機能なども備えることとする。

c．クラスタリング機構

記憶媒体上では静的なクラスタリングによりアクセスタイムの向上を図れるが，全てのアトリビュートに関してクラスタリングを行うことは出来ないため，ジョイン等処理負荷の重いオペレーションに対しては，5.2.2.B で述べた如く，動的にクラスタリングを行う必要がある。ＳＤＭではフィルターをかけた後ハッシュを施し，バケットＩｄを付加してＤＳＧに送出する。

ＳＤＭからのステージング操作に重畳してオーバヘッドなくクラスタリングを実現する。ハッシュ関数の設定はデータ対象に強く依存し，何らかの統計情報を管理しておくことが望ましいと考えられる。しかし，ベースリレーションに対するハッシュとはいえ，フィルタリングを施した後の分布は必ずしも予測容易ではない。従って，理想的なバケット分割が生成されると考えるのではなく，むしろ生成されたバケットの調整・統合等を行う柔軟なバケット操作方式を採用すべきである。

d．ステージング制御

Set orientedな関係代数処理の環境下では，従来のnavigational なデータベース処理に比べて次の処理対象を予測することは容易であり，従って先回りステージングにより実効的にアクセスタイムを小さくし，処理速度を改善することが可能である。ＳＶＣ，ＳＤＭ間でこの先行制御を行う。ＳＤＭでは可動ヘッドディスクを最下層記憶デバイスとしており，ヘッドスケジューリングとセミコンダクターディスクを用いた先回りバッファリングにより，実効的なデータベース処理速度を高速化することが出来る。

e．記憶管理機構

ＳＶＣは処理に必要とされるオブジェクトがどのＳＤＭ群に格納されているかという情報は管理しているが，ＳＤＭ内のどのディスクのどのトラック上に存在するかは関知しない。従って，ＳＶＣによって与えられる論理識別子から，ディスク空間上の物理アドレスへの対応を管理する必要がある。さらに，ディスクボリュームのエクステント管理，配置管理などの諸機能もＳＤＭになくてはならない。ＳＤＭは単なるディスク集団ではなく，そのインタフェースは高度化されている。

f．更新機構

単一リレーションに対する単純な更新処理では，必ずしもＤＳＧにステージングした後行うことは得策と考えられず，ＳＤＭ側で処理する方が効率的である。即ち，更新操作をセミコンダクターディスク上で直接行う機能が必要となる。複数リレーションにまたがる複雑な更新操作の場合にはＤＳＧ，ＤＳＰを用いる必要があるが，この場合にも当該リレーション全体を置き換えるのではない場合には，ＳＤＭへ逆方向クラスタリングされた更新タプル集合を用いてセミコンダクターディスク上で更新処理を行う。また，インテグリティ維持のため，複雑なアサーションチェックが必要となる場合には，Change SetをＳＤＭ上に生成し，高速にバックアウト可能な機能をもたせることとする。また，インデックスや統計情報など，補助データ構造の維持が更新に対して従来大きな負担となっていたが，これらに対しては専用ハードウェアを設けることにより，ボトルネックとならないよう配慮する必要がある。

g. ＳＤＭの構成

ＳＤＭの基本構成を図5.2.7に示す。ディスクはデータベースの格納媒体であり，１つのＳＤＭ当り数GB～数十ＧＢ程度の容量を有するものとする。セミコンダクターディスクは数十M～数百ＭＢバイト程度の半導体メモリから成り，先回りステージングによりディスクのアクセスタイムを実効的に短縮化出来る。磁気ディスク，セミコンダクターディスクの容量はデータベースの性質に依存すると考えられ，最適容量に関して充分検討を行う必要がある。

連想ディスクコントローラ（ＡＤＣ）は，すでに述べたフィルタリング機構，クラスタリング機構，およびローカルアプデイト機構を有する。フィルタプロセッサに関しては，ディスクからのデータ流に対し直接処理を施す方式と，データを一旦セミコンダクターディスクにステージングした後その上で操作する方式の二通りが考えられる。単純な処理はデータステージングと同時に行うことが出来，時間的無駄がないのに対し，複雑な処理はデータ流に追従出来ないため，

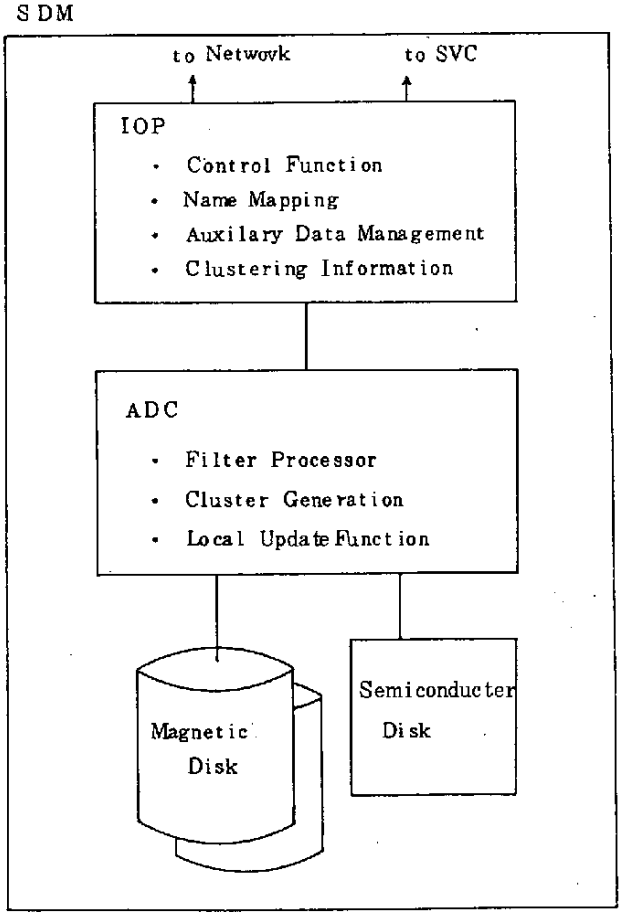


図5.2.7 Global Architecture of ＳＤＭ

RAM上で行った方が効率的である。両手法を融合することが望しい。更新はセミコンダクターディスク上のデータに対して行われ，またクラスタリングのためのハッシュもフィルタを通った後のRAM上タプルに対して施される。

I／Oプロセッサ（IOP）はSDM各部を制御する機能をもつ。SVCにより発行されたコマンドを解析し，論理名から物理アドレスを生成すると共に，ADCに対して適切なディスクコマンドを発生する。物理アドレスの生成に必要な補助データ，および，クラスタリングに用いられる統計情報なども管理する。

C. Data Stream Generator（DSG）

DSGの役割は，当該関係代数オペレーションに適したデータストリームを生成することにある。以下，その諸機能および構成について述べる。

a．データストリーム入力とバケット管理

データストリームを生成するためには，まずそのためのデータを入力する必要がある。DSGへの入力としてはSDMからのベースリレーションのステージング，およびDSPからの中間リレーションの生成の2つの場合が考えられる。いずれの場合にもDSGに入力されるタプルには，次オペレーションに関するクラスタリングを行うためのバケットIdがタグとして付加されている。これらはSDMのADCに属するクラスタジェネレータ，およびDSPのそれによって生成される。DSGは大容量バッファを有しており，入力完了後，直ちにバケットシリアルなデータ流を生成出来るように記憶管理を行う必要がある。バケットId毎にリストを生成したり，あるいは，バケットIdをキーとするちらし編成をRAM上で実現したりすることによりタプルを分類しておく。この操作はタプルが入力される毎に入力ストリームに遅れることなく行う必要がある。また，DSGは当該バケットIdに属するタプル数を管理しており，この数はバケットサイズの調整や，バケットオーバフロー（一台のDSPにバケットが入り切らない事をいう）の生じた場合のスケジューリング等に用いられる。

b．データストリーム生成

次オペレーションに必要な全てのデータがDSG上に入力されると，DSGはDSPに対してバケットシリアルなデータストリームを生成する。バケットサイズの分布の偏りが激しい場合にはそれらを統合し調整したり，バケット出力シーケンスのスケジュールを行うこともある。DSGは当該バケットを構成するデータを連続して出力する必要があり，このためリスト構造によってバケット管理を行っている場合には，そのポインタを辿りながらデータを取り出すインテリジェントなDMA機構が必要となる。当該バケットの出力が終了すると次バケットの処理に移る。バケットの切換をすみやかに行うためのバケットディレクトリサポート機構も必要

と考えられる。

c．結果リレーションの管理

問い合わせの処理結果はＤＳＧに生成され，その後，ＳＶＣを介してホストに転送される。ＤＳＧはデータストリームの生成や入力の他に，結果リレーションの記憶管理を行う。セレクション文の単純な問い合わせも，一旦ＤＳＧ上に生成される。

d．ＳＤＭへのデータストリーム生成

検索を中心とする問い合わせの場合，ＳＤＭから取り出されたデータはＤＳＧとＤＳＰを用いて処理され，ＤＳＧからホストへ転送されるため，ＳＤＭにデータが戻されることはないが，複雑な条件節を含む更新要求に対してはＤＳＧからＳＤＭに対してデータストリームが生成される。

e．ＤＳＧの構成

ＤＳＧの基本構成を図 5.2.8 に示す。

メモリバンクは中間リレーションのバケット，および結果リレーションを貯えておくもので，データ流を乱すことなく読み書き出来る必要があり，容量としては数十ＭＢ程度のものが望まれる。プロセッサはバケットIdを参照しながらバケットシリアルにデータ流を生成したり，入力したりする。また結果リレーションの管理も行う。これらは各々マイクロプログラム化された専用ハードウェアによって実現される。また，ＳＶＣからのコマンドを解析するためのコマンド処理機構も含まれる。

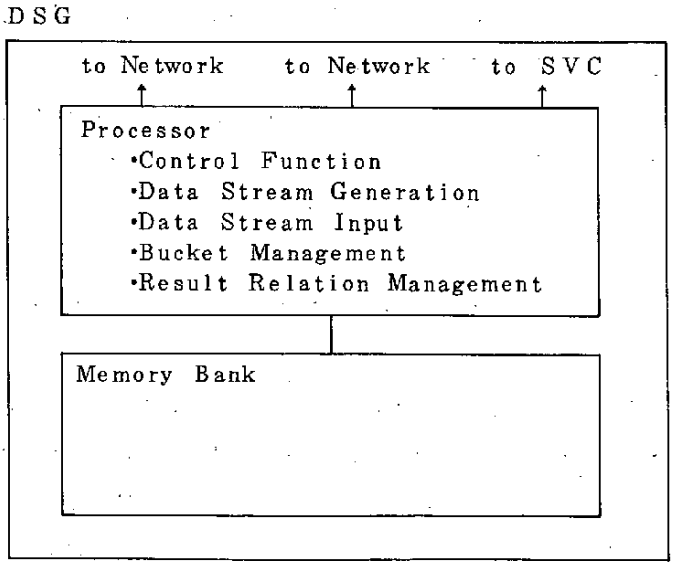


図5.2.8 Global Architecture of DSG

D. Data Stream Processor（ＤＳＰ）

ＤＳＰの役割は，割り当てられたパケットに対して指定された関係代数演算を施すことにある。以下，その諸機能，および構成について述べる。

a．ソート機構

ジョイン，プロジェクション，ディビジョンなど，種々の関係代数演算，および，集合演算処理は，対象とするデータがソートされている場合，O(n)時間で実行することが可能となる。このため，ＤＳＰはＤＳＧから送られてくるデータストリームを入力すると同時に，当該アトリビュートに関してソートを行なう。ここで用いられるソータは入力データ流に同期したソートが行え，データ入力が完了した時点で，ほとんど遅れることなくソートされたデータストリームが得られることが望ましい。ＤＳＧから送られてくるレコードは固定長としてもリレーション毎に変動するため，ソータはその変動に対処できる構造であることが望ましい。このためハードウェアはより複雑になるがlog N プロセッサによるハードウェアアルゴリズムでは，それ程大きな問題にはならない。ソータがキーだけではなく，レコード全体を取り扱う場合には外部メモリは不要となる。さらに，ＤＳＧからはワードスライスに，データ流を下位バイトから転送し，ＤＳＰでディジタルソートを行うことも考えられる。この場合にはソータの容量は小さく，レコード本体は別のメモリバッファ上にとられることになる。ソータ自身の取り扱うレコードは固定長に限ることが出来る。また，この場合には，処理上レコード全体のワードスライス分の時間遅れが生ずることになる。ソータの構成をレコード指向ととらえるか，ワードスライスととらえるかはソータだけの設計にとどまらず，処理系全体に影響を与える。例えば，ＤＳＧはレコード指向の出力をし，ＤＳＰはワードスライスのソートを行うことも可能であるが，これでは処理時間が大幅に増加することになる。従って，各部の整合性について今後より詳細に検討する必要がある。アルゴリズム的には，ソート対象を完全固定長とするか可変固定長とするかは問題ではなく，パイプライン化されたバブルソート，ヒープソート，マージソートなどいくつかの方式が考えられるが，実装上差が生ずることになる。アルゴリズムによってはパケットのソートを連続的に行えるものもあるが，データ入力とソート出力を同時に行えない場合には，２台用意する等の工夫も必要となる。

b．タプル操作機構

ジョイン，プロジェクション，ディビジョンなどの処理は，ソータからのソートされたデータ流に対して容易に行うことが出来る。プロジェクションではソート出力に対し重複タプルを取り除けばよく，ジョインでは２つのリレーションをまとめてソートすると，同一アトリビュートを有する両リレーションのタプルがソート順に出力され，その直積をとればよい。これら

の操作をソータからのデータ流に遅れることなく実現することが望まれる。ジョインでは一般にその結果は小さいと考えられるが，もとのバケットよりも大きくなる時は処理が追従出来なくなることもあり得る。直積をとる操作と，ジョイン候補を取り出す操作はバッファを介して分離することとし，処理負荷のゆらぎをなるべく吸収する必要がある。

また，セレクションやリストリクション，およびタプル整形操作もデータ流を乱すことなく行わねばならない。これらは，ＳＤＭにおけるフィルタープロセッサと同様の機構によって実現できる。

c．クラスタリング機構

タプル操作を施され，結果タプルとしてＤＳＧに戻される前に，次オペレーションに関するアトリビュートに対しハッシュを施し，クラスタリングを行う必要がある。ここで，結果タプルにバケットＩｄが付加された後に出力される。機能はＳＤＭのそれと同等のものであるが，一般にベースリレーションに対しては何らかの統計情報が有効な場合もあるが，中間リレーションに対しては予測不可能であり，分布テーブルの如き情報はＤＳＰでは用いられない。

d．算術演算機構

問い合わせの条件節，あるいは，出力節には算術演算が含まれる場合があり，整数，浮動小数点データに対する四則演算機構を設けるものとする。また集計，平均，最大，最小などの統計処理機構も備える。これは，Aggregate関数やディビジョンにおけるイメージ処理にも利用する。一般に，これらの諸機構はデータベースによっては不要であったり，あるいは，さらに複雑な機能が望まれる場合もあり，柔軟に対処すべきであろう。なお，統計演算処理など，各バケット毎に求めた値をさらにまとめる必要がある場合には，ＤＳＰ群全体としての算術演算部が用意されており，ここで処理される。算術演算機構はそれがない場合，全てホストで処理せねばならず，大きな負担となることが知られており（3.3節）ここでは各ＤＳＰ毎に設けることとする。

e．諸　機　能

すでに述べた機能に加え，サーチ，ストリングマッチ機構など，応用によっては導入する必要があろう。

f．ＤＳＰの構成

ＤＳＰの基本構成を図5.2.9に示す。

プロセッサはＳＶＣからのコマンド解析，および，ＤＳＰ全体の制御を司る。メモリは，作業用のＲＡＭであり，容量はソータの構成に依存する。ソータは，データ流に同期したO(n)ハードウェアソータであり，いくつかの実装方式が考えられる。ソータの出力はＯＦＰ（on the

fly processor）に導かれ，タプル操作，クラスタリング操作が施される。また，算術演算，統計演算は，MP（Math Processor）によってなされる。その他，サーチモジュール，ストリングマッチ用モジュールなどが付加されることがある。上記各ユニットは，専用化されたハードウェアにより構成され，ソータやフィルタプロセッサに関しては，ＶＬＳＩ化を試みる。

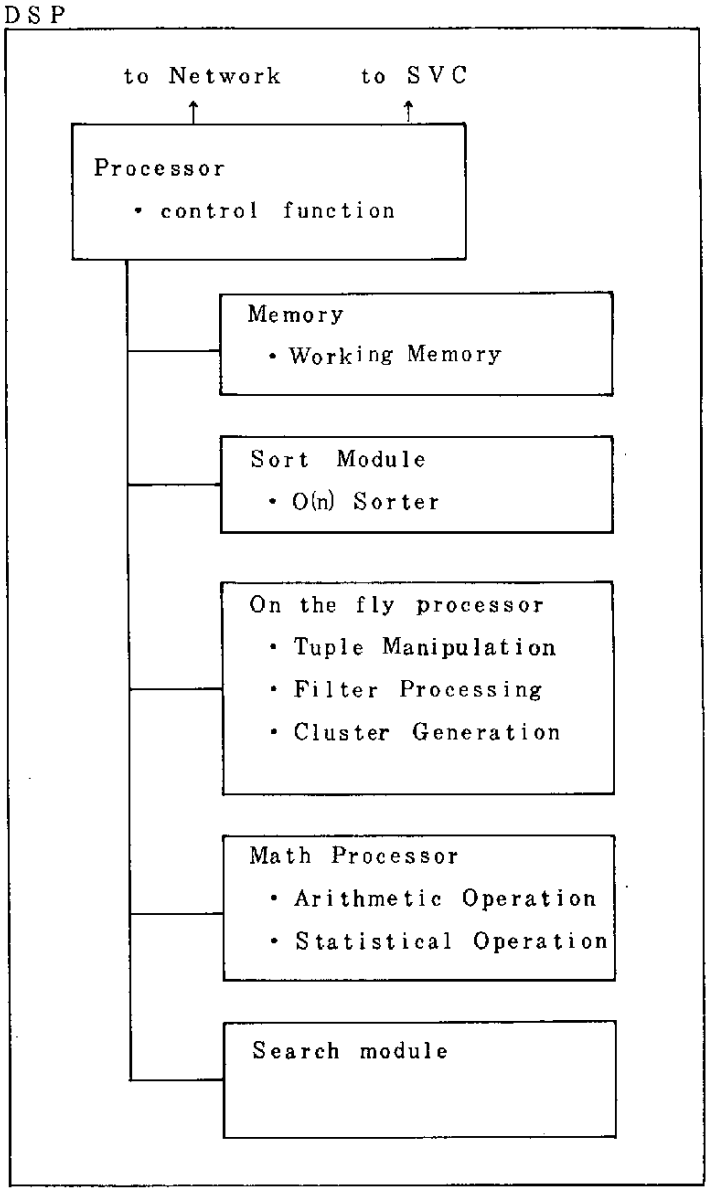


図5.2.9 Global Architecture of DSP

E. Object Manager（OM）

OMは，DSP，DSG，SDM各々の制御を行うとともに，ホストインタフェースを提供する部位であり，SPC（Stream Processing Controller），SGC（Stream Generation Controller），SDC（Secondary Data Controller），SVC（Supervisor Controller），およびDDP（Dictronary & Directory Processor）からなる。以下，各構成要素について述べる。

SDCはSDMの実行管理を行うとともに，SDM系全体に対するリレーションの記憶管理を行う。すなわち，各SDM毎の容量管理，および，リレーションのアクセス頻度等をもとに，最適な配置をとるよう制御する。また，データベースのイニシャルローディング，あるいは，リカバリ時のローディングなどは，SDCを介してなされる。問い合わせ実行に際しては，SDMに対してコマンドを発行し，先回リステージングを行う。SDCは，リレーション名，セグメントid，フィルタリング情報，クラスタリングアトリビュートおよび，データストリーム送出先のDSG idリストなどをSDMに送る。

SGCは，DSGの実行管理を行うとともに，結果リレーションの収集を行う。結果リレーションは，DSG上に一旦生成された後，SGCを介してホストへ転送される。SGCは，各DSG上の断片を収集する機能を有する。また問い合わせの実行に際しては，SDM，DSPからのデータ流に対する入力指令，DSPへのデータ出力指令を発行する。バケットIdリスト，クラスタリングキー，送出先DSPidリストなどの情報がDSGに送られる。またバケットオーバフロー時には，スケジューリング指令を発行し，例外事象にも対処する。

SPCは，DSP群の実行管理を行う。DSPに対し，オペレーションの種類，条件アトリビュート，出力フォーマット，クラスタリングアトリビュート，ハッシュ関数，結果タプル送出用DSG idリストなど当該バケットを処理するのに必要とされる種々の情報を送出する。バケットオーバフロー時にはDSP間の協調も必要となり，SPCはこの制御も行う。

SVCは，問い合わせ実行に関する統合的な制御を行う。すなわち，当該オペレーションに対し，DSG，DSP資源の割り付けを行い，SDC，SGC，SPCに対して適切な指令を発行する。中間リレーションの大きさは処理が進むにつれて変化するため，問い合わせレベルの静的なスケジューリングに加えて，オペレーションレベルの動的なスケジューリングを行うことが望ましい。

上記各制御モジュールでは，データストリームをオブジェクトとみなし，オブジェクト指向の制御アーキテクチャにより，各DSP，DSG，SDMを管理，制御し，データフローコントロールにもとづき，データストリームの流れを制御する。

またSPC，SGC，SDCでは各々DSP，DSG，SDMの動作を監視し，障害が検出される

とシステムを再構成することにより，柔軟に対処する必要がある。

ＤＤＰは，ＤＤ／Ｄを管理するための要素であり，ＳＤＭと同様の構成をとる。データベースに関する各種のメタ情報，スキーマ，インテグリティ，セキュリティ，および各種統計情報などが，統一的に管理される。

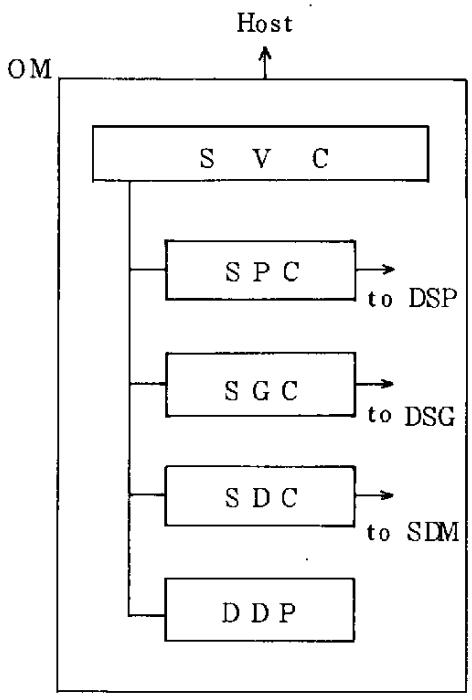


図5.2.10 Global Architecture of OM

F. PGN, GDN

ＰＧＮは，ＤＳＧとＤＳＰの間のデータ転送を司るネットワークであり，ＧＤＮはＤＳＧとＳＤＭ間用のネットワークである。共に両方向性のネットワークが必要となる。これらのネットワークでは，各システム構成要素から発生されたデータストリームを効率よく転送し，システムのボトルネックとならないように設計される必要がある。特に，ＤＳＰとＤＳＧの間のデータトラヒックはＳＤＭとＤＳＧ間のそれに比べると高くなると考えられる。また，データストリーム処理では，記憶媒体からのデータストリームに適合した処理がなされており，データストリーム自体は比較的長いため，ネットワークのスイッチング制御に伴うオーバヘッドに対する制約は緩和されるものと考えられる。ネットワークの実現形態としては，ＤＳＰ，ＤＳＧなど各モジュールを多数結合したシステムを構成するためには単一バス結合方式などでは不充分である。マトリック

ススイッチ方式も，O（n^2）の結合点数が必要となり，多数のモジュール間の結合には適さない。リングバスは多モジュールの結合には適しているが，現時点では充分な転送レートが確保されていない。しかし将来，光ファイバの利用によってかなり高い転送能力が期待できる。また，種々のスイッチングネットワークを利用することも有望である。高い転送レートを有し，かつ，結合モジュール数の増加に対する影響の少ない柔軟な結合ネットワークを採用することが望ましい。

5.3 ソフトウェア

本節では，プロトタイプ関係データベースマシンのソフトウェアシステムについて述べる。ソフトウェアシステムとしては，以下について考える。

(1) 基本ソフトウェア

(2) データベース管理システム（ＤＢＭＳ）

5.3.1 基本ソフトウェアシステム

基本ソフトウェアとしては，関係データベースマシンの基本モジュールSDM，DSG，DSP，およびＳＶＣの各々に存在し，各々の動作を制御する役目を持っている。これらの機能については，5.2を参照されたい。

SVCは，SDM，DSG，DSPの個々の実行制御とともに，各モジュール間での通信の制御を行う。ＳＶＣでは，リレーションのセグメントをオブジェクトとした一元的なオブジェクト指向のリソース管理がなされる。オブジェクトに，種々のセキュリティ情報（アクセス権，許される演算，有資格者のリスト等）を与えることも可能である。このために，関係データベースマシン内のリソースをオブジェクト指向管理するためのオブジェクト指向言語が必要となる。この言語を用いたリソースの記述は，ＤＤ／Ｄ（data dictionary ／ directory）内のメタ情報として格納され，データベースマシンの運用において用いられる。

関係データベースマシンの各モジュール，i.e. SDM，DSG，DSP，によるオブジェクト（セグメント）のストリーム処理は，データフロースキームにもとづいて制御される。このために，データフロー言語がまず必要になる。ＳＶＣソフトウェアは，このデータフロー言語によって記述される。データベースシステムでは，共有されたデータに対するreadとwrite演算に対して，データベースのインテグリティを保持される必要がある。こうした，同時実行制御，障害管理制御の組み込まれたデータフロー言語が必要になる。

モジュール間のインタフェースとしては，次のものがある。

(1) SVC−SDM，SVC−DSG，SVC−DSP

(2) SDM−DSG

(3) DSG−DSP

これらのインタフェースとして，モジュール間の通信プロトコル（コマンド形式，伝送手順）の設定が必要になる。あるモジュールから，他のモジュールへのメッセージの転送に対しては，正常受信に対する acknowledgement（ＡＣＫ）転送と，タイムアウトと再送による障害管理が考えられる。

5.3.2 データベース管理ソフトウェア

5.3.1で述べたオブジェクト指向なりソース記述言語と，データフロー制御言語にもとづいて，データベース管理ソフトウェアが記述されねばならない。データベース管理ソフトウェアとしては，以下の点を考えねばならない。

(1) ユーザインタフェース

(2) 問い合わせ／更新演算変換

(3) トランザクション管理

(4) DD／D

(5) セキュリティ管理

これらの管理ソフトウェアは，基本ソフトウェアにもとづいて，データフロー言語によって記述され，データベースマシンのSVC内に存在する。

A．ユーザインタフェース

ユーザインタフェースは，ユーザとの接点として，RDBMへの容易なアクセスを可能とする必要がある。このため，以下の機能が必要である。

(1) 高水準データ操作と定義機能

(2) ビューの定義とアクセス機能

(3) QBE，SDMS的な2次元インタフェース機能

(4) トランザクション定義機能

高水準言語としては，従来のSEQUEL，QUELといった言語の設定も可能である。この時，トランザクション定義のためには，レコード単位アクセスにもとづいた親言語（例，PL／I）インタフェースも必要になる。今後，関数型言語，単一割り当て言語と関連した高水準言語によって，従来の問い合わせと共に，トランザクションも1元的に扱かえることが望ましい。

ビューとしては，検索利用に対しては，従来のように，問い合わせ言語と同形式で任意に定義できる。更新を行うためには，このようなビューのデータ構造の定義と共に，許される演算をビューの意味から明確に定義させる方法がある。これは，ソフトウェア工学におけるデータ抽象化技法である。即ち，ビューを抽象データ型として定義し，許される抽象演算を，トランザクションとしてベース言語（ユーザ言語）によって記述してしまうものである。

他に，QBE，SDMS的な2次元インタフェースが，必要である。

B．検索／更新演算変換

ユーザインタフェースに入力された高水準アクセス要求から，RDBMで実行可能な形式を生成する問題である。検索演算に対して，RDBM上での最適なアクセスシーケンスの生成が問題

になる。非手続的な問い合わせから，手続きを生成する問題は，可能な手続きの中から，ある目標を満足する最適なものを見つけることである。この問題は，全ての可能な手を調べることが不可能なために，一般にヒューリスティクスによって解かれる。

最適化の目標としては，応答時間の最少化，全処理時間の最少化といった個々の問い合わせに関するものとともに，リソースの最大有効利用，最大スループット等のＲＤＢＭ全体に関するものがある。複数ユーザ下での最適化では，各々の問い合わせごとのローカル最適化と，ＲＤＢＭ全体のグローバル最適化とのかね合いが問題である。

ヒューリスティクスとしては，処理の中間結果のサイズの期待値を，カーディナリティ，選択度といった統計情報によって見積る方法が１つである。この時，選択度と，実際のデータベースとの適合性が問題である。

他の方法として，シーケンスの生成を，処理情況を見ながら動的に決めていく方法も考えられる。この時には，問い合わせのスケジューラが，実行情況をモニタできねばならない。

aggregate 関数，quantifier を持った問い合わせの最適化法が必要である。

更新演算の変換は，更新データを生成するための検索演算と，物理的な更新演算とから成る。前者に対しては，検索における最適化手法が用いれる。後者に対しては，後述するトランザクションの管理（ i．e. 同時実行制御，障害回復制御）が重要となる。

C．トランザクション管理

複数ユーザがデータベースに対する更新を同時に行う時，データベースのインテグリティを保つための同時実行制御（ concurrency control）と，ＲＤＢＭの種々の障害に対しても，データベースのインテグリティを保つための障害回復（ recovery）制御とが必要となる。ユーザから見た原子的な実行単位はトランザクションと呼ばれる。一般に，トランザクションは，一連の更新演算から成り立つプログラムである。

a．トランザクション

ＲＤＢＭでは，セグメントを処理単位とした集合演算が，システムの基本演算となっている。リレーションに対する更新も，従来の１つのリレーションの組単位の更新に加えて，複数リレーションを含めた集合単位の更新が行われる。従来のトランザクションにおける組（レコード）単位の基本演算（ read, write ）より，これらの集合演算にもとづいたトランザクションの定義が必要となる。

b．同時実行制御

データベースに対する競合する２つの演算とは，互いの実行順序を替え時に，その結果が異るものである。例えば，ある組Ｘに対するreadとwrite，またwriteとwriteが，競合演

算の例である。同時実行制御は，各トランザクションの競合する演算の相対的実行順序を，一定に保つことを目的としている。このための手法として，２フェーズロック（２ＰＬ），タイムスタンプ順序づけ（Ｔ／Ｏ）が，現在のデータベースシステム（ＤＢＳ），分散型データベースシステム（ＤＤＢＳ）において開発されている。ＲＤＢＳにおいて，ＳＤＭにおいてローカルに更新が行われる場合には，ＤＤＢＳにおける２ＰＬ，Ｔ／Ｏ手法の適用が考えれる。即ち，各ＳＤＭに，２ＰＬまたはＴ／Ｏのスケジューラを設け，競合する演算の相対的実行順序が，各ＳＤＭで同一となるようにスケジュールする方法である。

２ＰＬでは，オブジェクトをロックすることによって，他のトランザクションの実行を禁止することによって，実行同期をとろうとする方法である。ＲＤＢＭでは，オブジェクトはセグメントであり，ロックの単位が従来とは異ってくる。あるセグメントの更新が，他のセグメントにも影響する場合には，これらもロックする必要がある。新たな集合演算に対するロックモードと，各種演算の競合関係の定義が，新たに求められる。

ＳＤＭで，分散して更新を行う場合には，デッドロックの検出のための集中コントローラが必要になる。各ＳＤＭでのロック要求関係を表すwait－for グラフを集めることによって，ＳＤＭ間のデッドロックの検出と解除を行う。解除は，デッドロックサイクル内のあるトランザクションをバックアウトすることによって行う。

Ｔ／Ｏでは，あらかじめ各トランザクションに時間（タイムスタンプ）を与えることによって順序づける。各ＳＤＭでは，このタイムスタンプの順に，競合する演算を実行させる。Ｔ／Ｏでは，デッドロックが生じない利点がある。タイムスタンプを，オブジェクトごとに格納するためのオーバヘッドが問題である。

これ等の同期手法と，新たな処理単位（セグメント）と集合演算との結びつけが，ＲＤＢＭでは必要になる。

c．障害回復制御

障害回復では，トランザクションの原子性（atomicity）が重要である。トランザクションの実行は，完全に正常終了する（コミットされる）か，全く何も実行されないかである。トランザクションの実行中に障害が発生すれば，このトランザクションの実行中のデータベースへの変化演算は，何等データベースに影響を与えてはならない。この原子性を保障するためには，２フェーズコミット手法が用いれる。即ち，トランザクションの更新データの存在する全てのＳＤＭが，更新データを各々のログまたは信頼性の高いエリアにセーブしたことが確認された後に，初めて２次記憶への物理的な更新が行われる。１つでのＳＤＭが更新データをセーヴできない時は，トランザクションをバックアウトすることによって，原子性が保障される。

システムの障害に対しては，各トランザクションの演算の実行前後の値をセーヴしたログ，データベースのバックアップコピー，チェックポイント方式によって障害回復を行う。トランザクションが集合単位の更新を行うことから，ログに対する格納情報が多くなる。大規模データベースのバックアップ形成は，多くの時間を費すことから，データベースの部分単位のバックアップ方法，ＳＤＭ単位のバックアップとログ管理とローカル障害回復の強化が必要になる。

ＲＤＢＭは，従来のデータベースシステム（ＤＢＳ）に対して，記憶階層が複雑なことから，各階層モジュール，および階層間の障害の原因，頻度等のモデル化が必要である。

D．ＤＤ／Ｄ

ＲＤＢＭにおけるＤＤ／Ｄは，ユーザインタフェース，マシンインタフェースにおけるオブジェクトの記述と，オブジェクトに対して許される演算の記述，階層間の対応情報（naming情報），セキュリティとインテグリティの記述が格納される。さらに，各セグメント，リレーションに関するパフォーマンス情報（カーディナリティ，選択度，値の分散等），利用情況の統計情報，アカウント等の運用情報も，ＤＤ／Ｄとして一元的に管理される。

ＲＤＢＭにおけるソフトウェアシステムは，ＤＤ／Ｄ内の情報を用いて，問い合わせの変換／最適化，トランザクションの管理（同時実行制御，障害回復），セキュリティ管理，インテグリティ管理を行う。ＤＤ／Ｄに対するアクセスと記述（定義）インタフェースは，マシンインタフェースまたはユーザインタフェースと同一であることが望ましい。システムが，ＤＤ／Ｄを，通常のリレーションとしてアクセスできることは，ＲＤＢＭのソフトウェア開発上望ましい。

E．セキュリティ管理

データベースのセキュリティとは，資格のない利用者によるデータへの演算を禁止することである。セキュリティ管理機構としては，パスワードによる資格チェック，関係計算言語形式またはアサーションによるオブジェクトに対する許される演算と許される資格者とを設定する方式を考える。これらのセキュリティ情報は，ＤＤ／Ｄ内に格納される。ＳＶＣにおいて，ユーザのトランザクションの解析時に，ＤＤ／Ｄ情報を用いてチェックを行うものとする。

5.4 機能モジュール

5.4.1 関係代数演算処理モジュール

関係代数演算を始めとするデータベース演算をデータストリーム処理によって高速処理するモジュールには，以下のようなものが考えられる。これらは，ＤＳＰにおいて用いられる。

(1) サーチモジュール

(2) ソートモジュール

(3) オンザフライ・プロセッサ

(4) マスプロセッサ

サーチモジュールはキー順に並んだデータ項目を持つ探索表から，多数の探索キーを順々に見つけ出し，対応するレコード部を順々に出力する。複数の探索キーを一括して順々に探索する過程のことを一括探索と呼ぶ。

ソートモジュールは，キーとレコードとの対の集合をキー順に並べ換えるモジュールである。

オンザフライプロセッサは，セレクションやリストクションの機能の他，重複データを除去する機能や，ハッシングの機能を持つ。

マスプロセッサは，指定された属性の値に四則演算を施したり，ある属性についてgroup-byされた関係に対し，各グループ毎に，指定された一定の属性についての総和や，総数，平均等の統計量を計算する機能を有する。

A．サーチモジュール

探索テーブルの一括探索をデータストリーム処理するモジュールである。ソートモジュールが数多く提案されているのに対し，サーチモジュールの提案は少ない。例としては，北海道大学が開発したサーチエンジンがある〔ＴANA８０〕。

サーチエンジンは，２分木探索をパイプライン処理する。探索テーブルの大きさをｍ，一括探索するキィーの個数をｎとすると，１台のプロセッサでこれを処理する場合の時間複雑度はＯ（ｎ log ｍ）である。これに対し，サーチエンジンは，Ｏ（log ｍ）個の比較器によるパイプライン処理により，Ｏ（ｎ log ｍ）をO(n)に改善する。

サーチモジュールのＶＬＳＩ化に際しては，サーチエンジンに見られるようなビット・スライス機能の付加を留意すべきである。これによって，ピン数が多数増加するようなアーキテクチャは適当でない。

B．ソートモジュール

キーとレコードとの対の集合をキー順に並べかえるソートモジュールは，多数の種類が提

案されている。

矢島等による一覧表〔ＹＡＪＩ８１〕を表5.4.1に示す。

表5.4.1　ソートのアルゴリズム〔ＹＡＪＩ８１〕

アルゴリズム名	開発者，年	プロセッサ数のオーダ	ステップ数のオーダ	プロセッサ数×ステップ数
バブルソート		1	n	n^2
選択ソート，挿入ソート等		1	nloge	nlogn
分散ソート	Dobosiewicz, 78	1	nlogn	nlogn
バイトニックソート	Batcher, 68	n	$(\log n)^2$	$n(\log n)^2$
リバウンドソート	Chen, Lum, Tung, 78	n	n	n^2
並列計数ソート	安浦，高木，81	n	n	n^2
並列マージソート	Todd, 78	logn	n	nlogn
並列ヒープソート	田中譲，30	logn	n	nlogn
格子結合上のソート	Tompson, Kung, 77 Nassimi, Sahni, 79	n	$n^{1/2}$	$n^{3/2}$
並列分散ソート	前川，79，81 inslow, Chow, 81	n n	logn 1～logn	n～nlogn
log n ステップの並列ソート	Hirschberg, 78 Preparata, 78	$n^{1+\alpha}$	logn	$n^{1+\alpha}\log n$
定時間ソート Ⅰ	Muller 75 Preparata,	n	1	n^2
定時間ソート Ⅱ	Haggkvist, Hell, 81	$n^{3/5}\log n$	2	$n^{3/5}\log n$

この内，データストリーム処理に用いることのできるアルゴリズムは，ステップ数が，O(n)となっているものである。これらの内，比較器（表ではプロセッサ）の数がO(n)以上のものは，大容量ソータの実現には適していない。その結果，Todd のパイプラインマージソート〔ＴＯＤＤ

78〕や，田中のパイプライン・ヒープソート〔TANA80〕が，ここでの目的に適していると考えられる。

サーチモジュールの場合と同様に，ソートモジュールのＶＬＳＩ化に関してもビットスライス機能の付加に留意が必要である。さらに，ソートモジュールでは，ソート容量の多倍化機能の付加にも留意が必要である。これらの機能の付加によるピン数の増加が大きくなるようなアーキテクチャは適当でない。

C．オンザフライプロセッサ

セレクションやリストリクションのデータストリーム処理は，ＣＡＦＳに見ることができる〔BABB79〕。また，セミジョインのデータストリーム処理の例は，ハッシュ値でビット・マップを作成するアルゴリズムを用いたLEECH〔MCGR76〕やCAFSに見られる。

このような処理の他，ハッシングや，重複データの除去を行うモジュールをオンザフライプロセッサと呼ぶ。実現手法には２つ考えられる。１つは，これらの機能を持つ専用ＶＬＳＩプロセッサを考えることであり，他の１つは，ＶＬＳＩ化されたサーチモジュール，ソートモジュールと，ハッシング用の専用ＶＬＳＩを，汎用ＶＬＳＩプロセッサによって制御するというアーキテクチャを考えることである。

D．マスプロセッサ

入力されるデータ集合に対し，四則演算や初等関数の計算の他，総和，総数，平均，分散等の統計演算の計算をパイプライン処理する専用プロセッサの開発が必要である。

―参考文献―

1) 〔BABB79〕

E. Babb : Inplementing a Relational Database by Means of a Specialized Hardware, ACM TODS Vol. 4, No 1, pp1―29, 1979

2) 〔MCGR76〕

D. R. McGregor, R. H. Thomson, and W. N. Dawson : High Performance Hardware for Database Systems for Large Data Bases, North―Holland, pp. 103―116, 1976.

3) 〔TANA80〕

Y. Tanaka, Y. Nozaka, and A. Masuyama : Pipeline Searching

and Sorting Modules as Components of a Data Flow Data base Computer, IFIP Congress 80, 1980.

4) 〔TODD78〕

S. Todd : Algorithm and Hardware for a Merge Sort Using Multiple Processors, IBM J. of R&D, Vol. 22, No. 5, 1978.

5) 〔YAJI81〕

矢島 三, 安浦寛人, 上林弥彦：ハードウェアアルゴリズムの設計とその問題点, 電子通信学会研資, オートマトンと言語, 1981年12月

5.4.2 結合ネットワーク

プロトタイプデータベースマシンでは, 各々が100台程度の要素モジュールからなるサブシステム間を, 結合が自由に変えられる結合ネットワークで結合する 要がある。PGN(Processor Generator Netwsrk), GDN(Generator Disk Network)がそれである。特にPGNは, 100×100程度の結合ネットワークであり, 多段スイッチングネットワークの採用等が検討されねばならない。基本素子である2×2のネットワークは, データの衝突を避けるために, バッファと, ハンドシェーク機能を持たねばならない。このバッファの大きさをどの程度にするかが大きな問題となる。8×8程度のネットワークのVLSI化についても検討を要する。

多段スイッチングネットワークは, データフローマシンの研究開発においても, 開発が検討されることになっており, そちらの研究開発との密接な連係が必要である。

5.4.3 2次記憶モジュール

A. 連想ディスクコントローラ(ADC：Assosiative Disk Controller)

プロトタイプデータベースマシンにおける2次記憶システムは, 図5.4.1に示す如く半導体ディスク(SDK：Semicouductor Disk), 大容量ディスク(DK), 大容量記憶(MSS：Mass Storage System)の3段階の記憶階層を形成することになると思われる。

連想ディスクコントローラは, この3段階の記憶装置を統括的に制御する役割を持つものである。

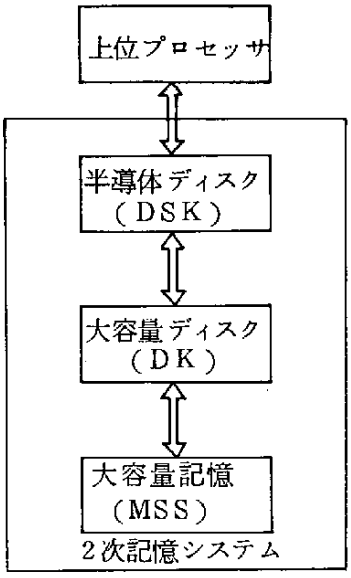


図5.4.1　2次記憶システム概念図

a．構　　成

図5.4.2に連想ディスクコントローラと各記憶装置との接続関係を示す。

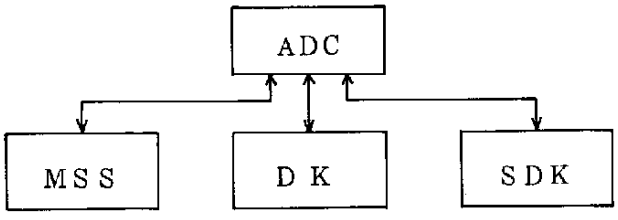


図5.4.2　ADC接続図

1台の連想ディスクコントローラに接続可能な，各記憶装置の記憶容量の目標仕様は下記とする。

(1)　SDK　　最大2GB

(2) DK　　スピンドル当たり300～1,200MB最大　32スピンドル

(3) MSS　　10～100GB

b．機　　能

連想ディスクコントローラの持つ機能としては，従来のディスクコントローラの機能に加えて，新たにデータベースマシン指向の機能を付加し，効率の良いデータベース処理を実現することが目的となる。

(1) 入出力制御機能

半導体ディスク，大容量ディスク，MSSの各入出力制御を実行する。

(2) ステージング制御機能

半導体ディスク，大容量ディスク，MSSの3段階の記憶階層間のステージング制御を実行する。特に大容量ディスクから半導体ディスクへのステージング制御は，上位プロセッサからの指令により，予じめ必要な関係データの先行ステージング機能を持たせる他，使用頻度の高い関係データに対しては，ディスクキャッシュの機能も持つようにする。

(3) 連想アクセス機能

ディスク内の関係データアクセス時に予じめ上位プロセッサより与えられたキーにもとづいて連想アクセスを可能とし，フィルタリング機能を持つものとする。

(4) ローカル更新処理機能

1台の連想ディスクコントローラ下にある関係データベース内部で処理可能なローカルなデータ処理は，更新処理も含め連想ディスクコントローラ内のファームウェアで処理可能な構造とする。

このローカル更新処理は過度にやり過ぎると，連想ディスクコントローラの処理能力がシステム全体のネックとなってしまうため，ローカル更新処理についてはどのような仕様でどの程度やるかは，今後の実験機，プロトタイプシステムを通しての研究課題となる。

(5) RAS機能

ディスクに対するバックアップファイルは，連想ディスクコントローラで自動的にMSSに対しとり，ディスク障害に対する回復処理は，連想ディスクコントローラのみで自動的に処理可能なことが望ましい。しかし，チェックポイントリトライのためのダンプ情報やジャーナル情報をとるには，より上位装置での制御が必要となり，システム全体に対するRASISをどうするかは，今後の研究課題である。

B．半導体ディスク（SDK：Semiconductor Disk）

半導体ディスクはプロトタイプデータベースマシンにおける大容量記憶（MSS），大容量デ

ィスク（ＤＫ）とともに２次記憶階層システムの一部を構成する記憶装置であり，２５６Ｋビット～１Ｍビット／チップ程度のＭＯＳメモリ素子を使用し，最大２Ｇバイトの容量を有するものとする。

半導体メモリの特長はディスク装置などと異なり，インターリーブ数，データ転送幅，メモリバンク容量などの論理構造を変えることにより，それぞれの目的に合わせた性能を比較的容易に実現できる点にあるといえる。

関係データベースマシンに採り入れる半導体ディスクが，どの程度の容量と，アクセス性能を必要とし，どのような論理構造を持つべきか，実験機システムなどを使って十分な検討が行われるべきであろう。

半導体ディスクは，ディスク，ＭＳＳと共に連想ディスクコントローラ（ＡＤＣ）の制御を受けるものとし，半導体ディスク自身では，インテリジェントな機能は持たないものとする。

図5.4.3に２５６ＫビットＭＯＳメモリ素子を使った２Ｇバイトの容量を有する半導体ディスクの一構成例を示す。

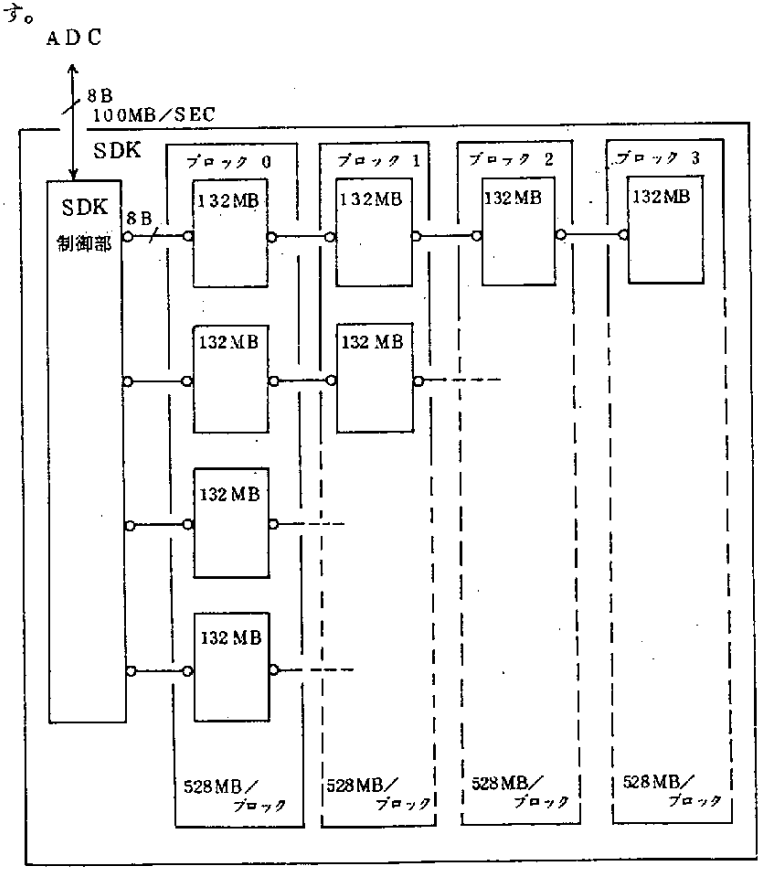


図5.4.3　ＳＤＫの構成例

6. 研究開発内容と開発計画

6.1 関係データベースマシンの方式研究

6.1.1 関係データベースマシンのアーキテクチャ研究

A. 目　　標

大量の知識を記憶し，それを高速に処理する高度知識ベースマシンの構築に当たり，その基本構成要素と考えられる関係データベースマシンを開発する必要がある。高速かつ大容量の関係データベースマシンアーキテクチャを決定するには，以下の点に関して詳細な検討を行い，種々の方式を開発する必要がある。

(1) 関係データベースマシンの知識ベースマシンにおける位置付けの明確化

(2) 高速関係代数処理方式の確立

(3) 関係データベースデータ構造の確立

(4) システム制御技法の確立

(5) 階層記憶システム構造の確立

(6) 演算モジュール構造の明確化

(7) システム評価技法の確立

(8) 各種支援機能に関するハードウェア化技法の確立

B. 研究開発のポイント

関係データベースマシンアーキテクチャの研究開発のポイントは，

a. 関係データベースマシンインタフェースの検討

b. 関係演算処理方式の検討

c. 関係データ構造の検討

d. コード化データベースと可変長レコード処理方式の検討

e. 演算モジュールの検討

f. 階層記憶系構造の検討

g. モジュール間結合方式の検討

h. システム制御系の検討

i. システム評価方式の検討

j. RASISに関する検討

の9点である。以下に，それぞれのポイントについて述べる。

a．関係データベースマシンインタフェースの検討

知識ベースマシンの構成要素としての関係データベースマシンの位置付けを明らかにすることが重要である。即ち知識処理における関係データベースマシンの果たすべき役割と必要とされる機能を明らかにし，マシンインタフェースを設定する必要がある。知識操作に関しては，現在不明な点が多いが，関係代数演算と論理プログラシングの関係等を検討し，従来のデータベース操作以外に必要とされる知識ベース固有な機能を明らかにすることが必要である。これをもとに関係データベースマシンのインタフェースを定め，知識操作方式を関係代数マシンとしてのアーキテクチャに融合することが出来る。

b．関係代数演算処理方式の検討

全ての関係代数演算をデータストリームに追従した形で実現する処理方式を明確化する。すなわち，現在得られている各種処理技法を比較評価し，それぞれの特長を統合化した効率の良い処理アルゴリズムを開発する。さらに，１つの演算子だけではなく，複数の演算子から構成される問い合わせレベルでの処理方式についても，問い合わせ木の縦方向パイプライン制御など種々の技法について検討する。また，システム内では多くの問い合わせを同時に処理することが必要となるが，問い合わせ内の並列処理，問い合わせ間の並列処理について検討し，レスポンス，スループット，それぞれを最大にする処理方式について研究を行う。特にマルチユーザ環境では多くのデータストリームが発生することになり，これらを効率よく制御する方式を開発する必要があり，データフロー制御方式の適用について考察する。

c．関係データ構造の検討

関係データベースマシンではリレーションの表現形式が，関係代数演算処理方式，マシンアーキテクチャに大きな影響を及ぼす。データ構造を大きく２つに分けるとすれば，リレーションの１タプルを１レコードとするタプル指向方式と，リレーションを各アトリビュート毎に分割し，キー又はタプルＩＤを付加したものを１レコードとみなすカラム指向方式の２つの手法があげられる。後者では関係を全てバイナリリレーションから構成することになる。これまでの多くのマシンはタプル指向のデータ構造を採用しており，比較的自然な形式と考えられる。タプル指向の記憶形式を採用している場合には１タプルを構成するアトリビュートはその隣接性により結合されているのに対し，カラム指向の記憶形式を採用した場合にはアトリビュート毎の対応関係を動的に生成する必要があり，その負荷は大きい。逆に，カラム指向の場合にはそのアトリビュートに関してクラスタリングを行うことが出来，しかもレコードには余分なアトリビュート情報は付加されておらず，当該アトリビュートに関するアクセスは高速で，かつ

小容量のデータ転送ですむ。２つの表現形式に対し，処理方式は異なったものになる可能性があるが，それぞれ特長を有しており，必ずしも常に一方が他方より優れているわけではない。問い合わせの種類，アトリビュートの性質などにも依存する。今後，より詳細な検討を重ね，両方式を統合した最適な表現形式を得る手法を確立することが必要である。

d．コード化データベースと可変長レコード処理方式の検討

(1) コード化データベース

データベース操作の対象をレコード本体ではなく，コード化されたレコードとする方式が考えられる。コード化を行い，対象データ構造を固定化することによりアーキテクチャはより簡素化することが可能であるが，入力出時に，常にエンコード，デコードの処理が必要となり，それだけオーバヘッドが増加することになるし，また，コード化表の管理が大きな負荷となり得る。一方，レコード本体をそのまま操作する直接的な手法は，エンコード，デコードに関する問題はなく処理手順は簡単になるものの，ハードウェアアーキテクチャの観点からは，処理対象が一般化されるため，より複雑になる可能性がある。両手法の適否は，アプリケーションのタイプにも依存すると考えられ，データベースマシン構築に際し一方に限定する必要はないと考えられる。アーキテクチャ上では，エンコード，デコード処理自体はコード化表とのジョインで処理でき，コード化処理方式自体は，比較的自然な形で，コード化を意図しないマシン上で実現出来る。コード化技法は，データベースマシンの各構成要素の内部構成に大きな影響を与えると考えられ，今後より詳細な検討を重ねるとともに，マシンレベルの全体的な評価を行う必要がある。

(2) 可変長レコード処理方式

レコード長に関しては，種々の自由度を設定することが出来る。

① 可変長キーフィールドレコード

② 固定長キーフィールドレコード

③ 固定長レコード

④ コード化レコード

前項ではエンコードレコード方式に関する課題について検討したが，一般レコードを扱う場合でもそのレコード形式を固定長とするのか，あるいは，可変長レコードを許すのかという点でアーキテクチャ上大きな違いが生ずると考えられる。即ち，一般に固定長形式を採用することにより処理方式が簡素化され，メモリ管理が容易になるのに対し，可変長レコードを対象とする場合にはその制御はより柔軟性が必要となり，一般的に複雑になることが多い。レコードの記憶構造は別として，処理の上では①のレベルまで一般化することはかなり困難

と考えられる。関係代数マシンでは，ジョイン，プロジェクション，ディビジョンなど複雑な処理を高速に実現することが要求され，このためには，よりデータ構造を簡素化することが望まれるが，一方で，２次記憶に密着した処理部位では，記憶媒体上でのデータ構造を直接処理できることが要求される。２次記憶上では記憶効率を考慮する必要があり，必ずしも固定長レコードを採用することは望ましいとはいえず，記憶媒体からのデータストリームを直接処理するためには可変長レコードを処理出来る機構も開発する必要がある。

以上の如く，レコード形式は，データベースマシン上その部位によっても異なる可能性があり，種々の形式に対する処理方式を検討する必要があると考えられる。

e．階層記憶系構造の検討

大容量データベースを構築するにあたっては，メモリ階層化技法，データベース編成法，アクセス制御方式等について詳細に検討する必要がある。スタティッククラスタリング技法を用いたデータベース編成法，および，インデックス等の補助的データ構造の管理とそれを用いた高速アクセス機構について，その方式を確立する。さらに，実効的アクセスタイムを短くするための，先行ステージング技法とセミコンダクターディスクの方式を検討する。また２次記憶からのデータストリームに対するフィルタリング機構（不要タプルの除去，不要アトリビュートの除去），ハッシュビットアレイを用いたセミジョイン機構，ダイナミッククラスタリング機構などを備えた連想ディスクコントローラのアーキテクチャを確立する。セミコンダクターディスク上での更新機構についても検討する。光メモリなど新しい記憶デバイスの導入についても考慮すべきである。

f．演算モジュールの検討

関係代数演算の基本となるデータ操作，例えば，ソート，フィルタ，タプルフォーマッティング，ハッシュ，ビットマップなどについてＶＬＳＩ化に適したデータストリーム処理アルゴリズムを開発し，そのアーキテクチャを明確化する。データ長の拡大や容量の増大に対する柔軟性について，充分検討する必要がある。エンコードされた固定長データのみを扱うのではなく，一般レコードも処理出来ることが望ましく，プログラマビリティの導入についても検討する必要がある。

g．モジュール間結合方式の検討

データストリームプロセッサとデータストリームジェネレータ間の結合方式の確立

データストリームジェネレータとセカンダリデータマネジャ間の結合方式

ホストコンピュータとオブジェクトマネジャー間の結合方式

オブジェクトマネジャーとデータストリームプロセッサ，データストリームジェネレータ，

セカンダリデータマネジャ間での制御データ授受，および，結果リレーションの転送のための結合方式

データベースローディンのためのオブジェクトマネジャ，セカンダリデータマネジャ間の結合方式

以上の各結合に関して，授受されるデータタイプ，データトラヒック，データストリーム制御方式などを詳細に検討し，柔軟性があり，しかも，高性能な結合機構を開発する必要がある。

h．システム制御系の検討

多くの問い合わせを並行して効率よく処理するための制御機構の開発を行う。従来開発されてきた多くのマシンでは，問い合わせレベルのマシン制御機構については充分考慮されておらず，また，複数の問い合わせの実行に関する制御手法にいたっては，ほとんど研究されていない。従って，システムベルでのデータベースマシン制御方式を早急に開発する必要がある。

プロトタイプデータベースマシンではオブジェクトマネジャがシステムレベルの制御を行う。すなわち，処理対象をオブジェクトとみなし，2次記憶系に対するオブジェクトのステージング制御，およびオブジェクト処理に関するプロセッサのアロケーションとデータストリーム制御部の駆動を司るオブジェクトマネジャにおける，制御技法を確立する。さらに，オブジェクト管理部のアーキテクチャを明確にし，制御機構のハードウェア化について検討する。

i．システム評価方式の検討

システム評価用データベースを作成するとともに，評価用ベンチマークについて検討する。単純な問い合わせ，複雑な条件式を含む問い合わせ，ジョイン数の多い問い合わせなど評価用の問い合わせをいくつか作成する。その際，リレーションの大きさ，タプルの長さ，アトリビュート内の値の分布などをパラメータとすることが望ましい。

また，処理速度，スループット，記憶効率，並列度，データベース参照のローカリティなど評価対象データを明確化し，データ採取技法を確立する。

以上のようにして詳細なシステム評価を行うことにより，ボトルネックを明らかに出来，アーキテクチャの改善を図ることが可能となる。

j．RASISに関する検討

データベースの管理に必要とされる機能はデータ操作だけではない。RASISの維持はデータベースシステムではきわめて重要な課題といえる。しかしながら，データベースマシンでは従来，ほとんど検討されていない。セキュリティ，インテグリティの維持，リカバリー機能，システム高信頼化などに関して詳細な検討を行う必要がある。インテグリティーに関しては，種々のアサーションを高速にチェック出来るようなハードウェア機構の開発を行い，また，セ

キュリティーに関しては，ケーパビリティ管理機構，エンクリプション技法などについて検討する必要がある。リカバリーに関しては，多数のモジュールの内，一部が障害を起こしても再構成可能であることが望ましく，結合ネットワークにおいてもその障害を局在化できることが期待される。また，２次記憶系においては，高速にバックアップを行えるハードウェア機構，チェックポイントからのリスタート機構などを設けることが必要である。

C．作　　業

前述の目標を達成するために，以下の作業を行う。

a．関係データベースマシンのアーキテクチャに関する仕様検討

前期３年で開発する実験機のアーキテクチャの詳細を検討するとともにプロトタイプデータベースマシンの仕様を決定する。

b．高速関係代数演算処理方式の研究

データフロー制御，および，データストリーム処理方式にもとづく関係代数演算メカニズムを確立する。

c．関係データベースデータ構造の研究

関係代数演算に適したデータ構造について，演算処理系，２次記憶系の両面から検討を行い，最適データ構造を求める。

d．システム制御技法の研究

断片化されたリレーションの管理と処理のオブジェクト概念にもとづく制御方式を検討する。

e．２次記憶システム構造，制御方式の研究

大容量化のための記憶の階層化，補助記憶構造の導入，アクセス制御方式の検討を行う。

f．演算モジュールの研究

関係代数演算を高速に行うデータストリームアルゴリズム検討し，ＶＬＳＩ化に適した演算モジュールの開発を行う。

g．結合方式の研究

データストリームプロセッサとデータストリームジェネレータ間の結合，データストリームジェネレータとセカンダリデータマネジャ間の結合，スーパバイザコントローラとホストの結合についてその方式を検討する。

h．各種支援機能のハードウェア化に関する研究

セキュリティ，インテグリティ，コンカレンシィ等の制御，問い合わせ変換操作，プロセッサ群実行管理機構等集中化されたシステム構成部位における各種機能の高速化について検討する。

i．システム評価技法の研究

アーキテクチャ上の性能評価，2次記憶系における記憶効率の評価，コンカレンシーレベルの評価等，システム評価基準を明確化するとともに，評価データ採取ツールの開発を行う。また，評価用データベースを生成する。

D．詳細スケジュール

a．線　　表

	57年	58年	59年
a．関係データベースマシンのアーキテクチャに関する仕様検討	実験機仕様の詳細化	アーキテクチャ検討	プロトタイプマシンのアーキテクチャ仕様決定
b．～i．	基本検討	第1次アーキテクチャ	第1次アーキテクチャの評価，及びプロトタイプアーキテクチャの概容決定
b．高速関係代数演算方式の研究	処理アルゴリズムの開発	比較評価	アーキテクチャ改良
c．関係データベースデータ構造の研究	各種データ構造の検討	比較評価	データ構造の改良
d．システム制御技法の研究	オブジェクト指向方式の検討	各種制御技法の評価	システム制御技法の改良
e．2次記憶システム構造制御方式の研究	各種記憶構造の検討	比較評価	2次記憶システムアーキテクチャの改良
f．演算モジュールの研究	演算アルゴリズムの開発	VLSI化検討	演算モジュールの評価
g．結合方式の研究	各モジュール間結合方式の開発	比較評価 VLSI化検討	結合方式の改良
h．各種支援機能のハードウェア化に関する研究	高速化すべき支援機能の明確化	各種方式の比較検討	アーキテクチャの改良
i．システム評価技法の研究	評価項目の明確化	評価技法および評価ツールの比較検討	・評価用ツールの開発 ・評価用ハードウェアのアーキテクチャへの組込み

b．マンパワー　（各年度同一とする）

	研究者	技術者
(1)	1	0
(2)	3	3
(3) (5)	1	1
(4)	1	1
(6)	6. 2. 1 参照	
(7)	1	1
(8)	6. 2. 2 参照	
(9)	1	3
計	7	9（人）

c．所要リソース

(1) 計算機利用　（各プロジェクト合計）

５７年	５８年	５９年
１０h	５０h	５０h

(2) シミュレーション用高速マシン

VLSI CADシステム

評価用データベース

6.1.2 関係データベースマシンのソフトウェア研究

A．目　　標

関係データベースマシンは，SDM，DSG，DSPによるストリーム処理を，オブジェクト指向のリソース管理とデータフロースキームとによって，実行制御を行うシステムである。また，この基本機能の上に，複数ユーザのサポート，同時実行制御，障害回復管理，セキュリティ管理，演算変換，DD／D，容易なユーザインタフェース等のデータベース管理機能を実現せねばならない。このため以下の課題の解決が必要となる。

(1) 各機能モジュールの機能の明確化と，そのソフトウェア開発

(2) 各モジュール間のインタフェース，通信プロトフルの確立

(3) マシン言語の設定

(4) データベース管理ソフトウェアの，(3)による開発

B．ポイント

a．各機能モジュールのソフトウェア開発関係データベースマシンの基本機能モジュール，ＳＤＭ，ＤＳＧ，ＤＳＰの各機能を明確にし，対応したソフトウェアを開発する。ここでは，この3つのモジュールのソフトについては，アーキテクチャレベルによってなされるものとする。

b．各機能モジュール間のインタフェース，通プロトコルの確立

各機能モジュール間ＳＶＣ－ＳＤＭ，ＳＶＣ－ＤＳＧ，ＳＤＭ－ＤＳＰ，ＤＳ －ＤＳＧ，ＤＳＧ－ＤＳＰのインタフェース／プロトフルを確立する。通信のためのメッセージのフォーマット（コマンド体系，パケット構成）と，転送と応答の体系を確立する。転送とAcknowledge-mern応答と，タイムアウト，再送方式等が考えられる。通信のオーバヘッドと共に，高信頼な通信プロトコルの設定が必要になる。

c．マシン言語の設定

ＳＶＣのソフトウェア記述言語の設定が必要である。ＳＶＣは，ＳＤＭ，ＤＳＰ，ＤＳＧによるストリーム処理を，オブジェクト指向のリソース管理と，データフロースキームによる実行制御を行うモジュールである。このため，マシン言語としては，以下の機能が必要である。

(1) オブジェクト指向のリソース管理機能（リレーションのセグメントをオブジェクトとする）

(2) データフロースキームにもとづいた制御機能

関係データベースマシン内のリソースは，オブジェクトとして統一的に管理される。リレーションのセグメントがオブジェクトとしてストリーム処理される。これらのオブジェクトには，許される演算，利用者等が定義される。この定義は，オブジェクト指向な言語によってなされ，ＤＤ／Ｄに格納され，運用時に用いられる。

関係データベースマシンでの演算は，オブジェクトとしてのセグメントが揃った時点で動作するように，データフロー的に実行が制御される。データベースマシンでは，オブジェクトが複数のユーザによって共有され，オブジェクトに対する演算（検索，更新）に対してオブジェクト内およびオブジェクト間のインテグリティが保たれる必要がある。データフロー言語と，このような同時実行制御は，現在，未解決の問題であるが，データベースマシンの実現には，まず第1に研究される必要がある。

後述するデータベース管理ソフトウェアは，このマシン言語によって記述される。

d．データベース管理ソフトウェアの開発データベース管理ソフトウェアを，(3)で述べたデータベースマシンのマシン言語によって記述する。このデータベース管理ソフトウェアは，ＳＶＣ内に存在する。オブジェクト指向でデータフロースキームにもとづいたマシン言語によって記述されるべきデータベース管理ソフトウェアとしては，以下のものがある。

・ユーザインタフェース機能

・検索／更新演算変機能

・トランザクション管理機能

・障害回復機能

・セキュリティ管理機能

・ＤＤ／Ｄ機能

(1) ユーザインタフェース

ユーザインタフェースの設定が必要である。高水準なユーザ言語（e.g. 関数型言語）とデータ構造の設定，ビューのサポート，ＱＢＥ，ＳＤＭＳのような２次元インタフェースが必要になる。同時に，ＲＤＢＭの管理用（e.g. インテグリティセキュリティの記述）の高水準インタフェースも求められる。同一の言語体系であることが望ましい。

(2) 検索／更新変換機能

ユーザインタフェースと，マシンインタフェースとの変換機構が必要となる。検索に対しては，マシンインタフェースレベルでの最適なアクセス要求に変換する問題がある。複数ユザ下での最適化技法の開発が今後の課題としてある。更新要求に対しては，(B)で述べるトランザクション管理機能が必要である。

(3) トランザクション管理機能

複数ユーザのもとで，種々の一重の検索／更新演算を，システムの障害に対しても，データベースのインテグリティを保ちながら有効に実行させるためのトランザクション管理方式の確立が必要である。現在のＤＢＭＳにおいても，System R 等で実験的に，種々の方式が試みられている段階である。

現在，トランザクションの管理方式としては，数多くの方式が提案されているが，それらの評価方式も定まっていない。このため，既存方式の分類，整理を行うとともに，評価方式を明確にする必要がある。特に，応答性，オーバヘッド，完全性等の評価規準の確立が必要である。

トランザクション管理では，ＤＤＢＳにおける方式の適用が有望である。これは，ＲＤＢＭにおいて，ＳＤＭにおいてローカルな更新を並行に行えるからである。ＤＤＢＳにおける２フェーズロック（２ＰＬ），タイムスタンプ順序づけ（Ｔ／Ｏ）方式を，ＳＤＭでの演算の実行の同期方式として適用を試みる。可能な方式の中から，必要な並行度，応答性，同期オーバヘッド，更新と検索の頻度等からＲＤＢＭにおける方式を選択する必要がある。ＤＤＢＳとＲＤＢＭでは，処理単位が異っている。ＲＤＢＭでは，リレーションのセグメントが

ストリームとして処理される。一方，ＤＤＢＳでは，組単位の処理がなされる。ロック，タイムスタンプの対象となる単位が異なっている。また，演算単位も組のread，writeではなく，ストリームに対する集合演算である。このような，新たな演算，オブジェクトに対応した，同時更新方式の基礎的な研究が必要である。

ＲＤＢＭでは，従来のコンピュータシステムに対して，SDM, DSG, DSP というより多くの記憶階層から構成されている。このため，まず，ＲＤＢＭにおける障害をモデル化する必要がある。各モジュールの信頼性障害の種類，多くのモジュールへの影響を明らかにし，障害回復手順を設定せねばならない。

従来のＤＢＭＳにおけるトランザクションに対して，セグメントを処理単位とした集合演算から成るトランザクション概念の明確化が必要である。

また，データベースにおけるインテグリティも，従来の定義域の範囲，型，長さ，統計量，主キー等に加えて，オブジェクト間の更新依存性（e.g. referential dependency）〔田中(Y)81〕の組み込みが必要である。従来の静的なデータモデルに対して，動的振舞いを組み込んだデータモデルの研究が必要である。

(4) セキュリティ管理方式

データベースを，利用資格のないユーザからアクセスされることに対するセキュリティ管理方式の確立が必要である。パスワード，述語，アサーション等の従来の技術とともにセグメントをオブジェクトとしたセキュリティ機構が必要である。セキュリティの権限委譲，チェック機構，暗号化等の検討も求められる。セキュリティについては，今後，基礎的な研究が必要である。

(5) ＤＤ／Ｄ機構

ＤＤ／Ｄは，ＲＤＢＭにおける各階層のオブジェクトとそれに対する演算の記述と，階層間の対応情報から成っている。ＲＤＢＭは，ＤＤ／Ｄ情報を用いて動作する。問い合わせの変換，最適化も，ＤＤ／Ｄを用いてなされる。また，システムの運用上の統計情報，アカウンティングも管理される。ＤＤ／Ｄ情報の内容，記述方法，格納方法，アクセス方法，パフォマンス情報（e.g.カーディナリティ選択度）とデータベースとの一致性の保持方法等が検討されねばならない。

e. ソフトウェア開発支援ツール

既存ＤＢＭＳは，巨大でかつ複雑なソフトウェアの１つであり，膨大な開発コストを要している。ＲＤＢＭにおけるソフトウェアも，機能的には従来のＯＳとＤＢＭＳを合わせたものと，ストリーム処理等の新たな機能を加えたものとなることから，かなりの大きさで複雑なものと

なる。このために，ソフトウェアの高級記述言語，言語プロセッサ，種々の方式のシュミレータが必要である。

(1) 基礎研究

基礎研究では，現在，まだ未解決な課題の基礎的研究を行い，その成果を，関係データベースマシンのソフトウェア仕様作成に生かす。

(i) マシン言語の設定

オブジェクト指向でデータフロースキームにもとづき，かつデータベースシステムにおける同時実行制御を記述し得るマシン言語の仕様を決定する。

(ii) (i)のシミュレータとコンパイラの開発

マシン言語のソフトウェアシミュレータとコンパイラを開発する。

(iii) 更新処理方式の研究

データベースに対する動的な更新に対するインテグリティのモデル化，オブジェクトベースの同時実行制御方式，障害回復方式，セキュリティ管理方式を研究し，これのマシン言語による実現化を研究する。

(2) 各機能モジュールの機能設計

関係データベースマシンの各機能モジュールの機能，インタフェースを明確化し，そのソフトウェアの基本設計を行う。

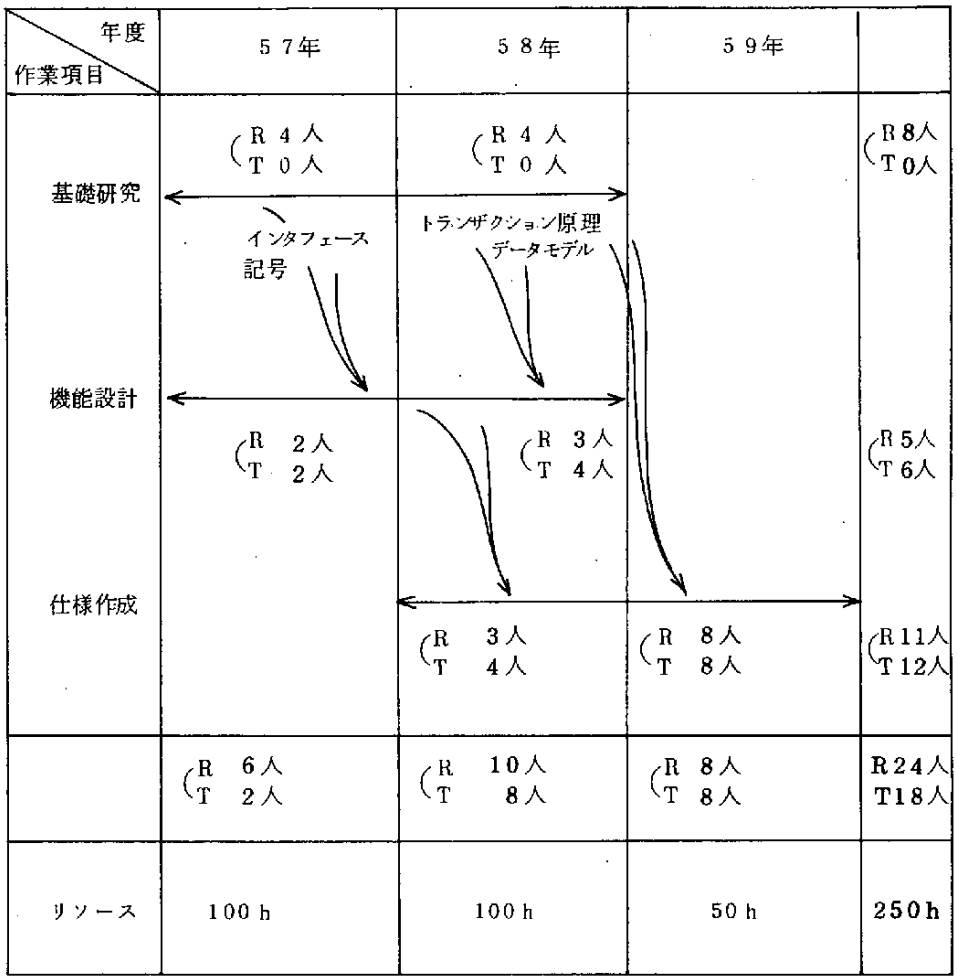
ソフトウェアシステムのユーザインタフェース，ＤＤ／Ｄ，変換方式の基本設計を行う。

(3) プロトタイプ関係データベースマシンのソフトウェアの基本仕様書作成

(1)および(2)にもとづいて，プロトタイプ関係データベースマシンのソフトウェアシステムの基本仕様書を作成する。ＳＶＣ内のソフトウェアは，(1)で設定される新しいマシン言語によってなされるものとする。

D．スケジュール

(1) 線表とマンパワー



R：研究者

T：技術者

6.2 要素技術の開発と要素モジュールの開発

6.2.1 関係代数演算処理モジュール

A. 目　　標

関係代数演算をデータストリーム処理によって高速に処理するＤＳＰ（データストリーム・プロセッサ）で用いる機能モジュールを開発する。

このような機能モジュールには，

(1) サーチモジュール（Search Module）

(2) ソートモジュール（Sort Module）

(3) オンザフライプロセッサ（On － the － Fly Processor）

(4) マスプロセッサ（Math Processor）

の５種類がある。

これらはいずれも，データストリーム処理でデータを処理する。

前期には，これらの機能モジュールをディスクリートな論理回路で実現し，実験機で用いる。中期以後のマシンの構成要素として，前期より，ＶＬＳＩ化の準備を始め，一部ＶＬＳＩの試作を行う。

以下に，開発の目安として，各モジュールの仕様例を示す。

a. サーチモジュール

(1) 機能仕様

値の順序で並んだキィーの集合 $\{K_i \mid K_i \leqq K_j \quad \text{if} \quad i < j,\ 0 \leqq i \leqq n\}$ を格納し，探索キィーの入力系列 $K_0, K_1, \cdots\cdots, K_m$ に対し，３つ組の系列 $(K_0, f_0, i_0), (K_1, f_1, i_1), \cdots\cdots, (K_m, f_m, i_m)$ を出力する。f_j は発見フラグ，i_j は発見番地と呼ばれ，

$$\text{i)}\ k_j \in \{K_i\} \text{ のとき}$$
$$f_j = 1$$
$$i_j = \min\{i \mid k_j = K_i\}$$
$$\text{ii)}\ k_j \notin \{K_i\} \text{ のとき}$$
$$f_j = 0$$
$$i_j = \min\{i \mid k_j < K_i\}$$

と定義される。

(2) 性能仕様

- キーの語長：32bit
- 格納テーブルの最大の大きさ：128K語
- 処理速度：

（テーブル格納）（1～10）×10^6 keys／sec

（テーブル探索）（1～10）×10^6 keys／sec

サーチモジュールを構成するためのVLSIの仕様を以下に示す。

(i) 機能仕様

上述の通り

(ii) 性能仕様

- キーの語長：16bit
- 格納テーブルの最大の大きさ：4K語
- 処理速度：

（テーブル格納）（1～10）×10^6 keys／sec

（テーブル探索）（1～10）×10^6 keys／sec

(iii) 拡張機能

- 語長の多倍長化機能：複数個のVLSIを用いて，キーの語長を16bitの整数倍にする機能

b．ソートモジュール

(1) 機能仕様

キーとレコードとの対の系列（k_0, r_0），（k_1, r_1），……（k_n, r_n）の入力に対し，pをi＜jなら$k_{p(i)} \leqq k_{p(j)}$となる置換として，系列（$k_{p(0)}$, $r_{p(0)}$），……（$k_{p(n)}$, $r_{p(n)}$）を出力する。

(2) 性能仕様

- キー，レコードの語長：32bit
- ソート可能な最大語数：128K語
- 処理速度：（1～10）×10^6 keys／sec

ソートモジュールを構成するためのVLSIの仕様を以下に示す。

(i) 機能仕様

上述の通り

(ii) 性能仕様

- キー，レコードの語長：16bit
- ソート可能な最大語数：4K語
- 処理速度：(1～10)×10^6 keys／sec

(iii) 拡張機能

- 語長の多倍長化機能：複数個のVLSIを用いて，キーとレコードの語長を16bitの整数倍にする機能
- ソート容量の多倍化機能

c．オンザフライプロセッサ

データストリームプロセッサ(DSP)，および連想ディスクコントローラの構成要素として用いる。

(1) 機能仕様

- セレクション
- リストリクション
- セミジョイン
- ハッシング
- 重複データ除去

(2) 性能仕様

- 処理速度：(1～10)×10^6 keys／sec

d．マスプロセッサ

(1) 機能仕様

- 入力データ系列d_1，d_2，……d_nに対して，
  - i 定数との四則演算
  - ii d_{2i}とd_{2i+1}との四則演算

  を，整数型，浮動小数点型について行う。
- 入力データ系列d_1，d_2，……，d_nに対して，
  - i $\Sigma_i d_i$
  - ii count($\{d_i\}$)
  - iii average($\{d_i\}$)
  - iv variance($\{d_i\}$)
  - v max($\{d_i\}$)，min($\{d_i\}$)

  を計算する。

・ 入力データ系列d_1, d_2, ……d_nに対して，sin, cos, tan, log, exp, $\sin^{-1}$, $\cos^{-1}$, $\tan^{-1}$, square, cube, root等の計算を行う。

(2) 性能仕様

・ 処理速度は一律に（1～10）×10^6 data／sec となることが望ましい。

B. ポイント

前期に試作する実験機に使用する機能モジュールはディスクリート論理回路で構成する。中期以後の開発計画を成功させるためには，これらのモジュールで本質的な機能を抜き出し，ビットスライス等の拡張機能を持ったＶＬＳＩとして実現することが必要である。

本節で述べる機能モジュールは，すべてデータストリーム処理方式で処理を実行することが重要であり，各々の機能をデータストリーム処理方式で処理するためのハードウェア・アルゴリズムの研究がポイントとなる。

C. 作業内容

開発の作業は，各機能モジュールとも以下の順序で行う。

a. ハードウェア・アルゴリズムの確立

データストリーム処理を行うためのハードウェアアルゴリズムを確立する。

b. ディスクリートタイプの開発

前期の実験機の構成モジュールとして使用するための各モジュールをディスクリート論理回路で実現する。

c. ＶＬＳＩ化

各モジュールの基本機能の抜き出しと，そのＶＬＳＩ化を検討する。そのためには，アルゴリズムの選択を行うことが必要である。検討結果にもとづいて，ＶＬＳＩの設計試作を行う。前期では，ＶＬＳＩのＣＡＤシステム，およびプロセスが，プロジェクト内部には完備されていないと考えられ，ＶＬＳＩの開発は発注によらなければならないと考える。

D. 開発計画

	57年	58年	59年
サーチ・モジュール	ディスクリート・タイプ試作	VLSI 設計	VLSI 開発
研究者	3 人	3 人	3 人
技術者	5 人	5 人	5 人

		57年	58年	59年
ソート・モジュール		ディスクリート・タイプ試作	VLSI 設計	VLSI 開発
研究者		3 人	3 人	3 人
技術者		5 人	5 人	5 人
オン・ザ・フライ・プロセッサ		ディスクリート・タイプ試作	VLSI 設計	VLSI 開発
研究者		3 人	3 人	3 人
技術者		5 人	5 人	5 人
マス・プロセッサ		ディスクリート・タイプ試作	VLSI 設計	VLSI 開発
研究者		3 人	3 人	3 人
技術者		5 人	5 人	5 人
計算機使用時間	TSS	400h	400h	400h
	CPU	20h	20h	20h

6.2.2 結合ネットワーク

A. 目　　標

前期には，DSP群とDSG群を結合するPGN（Processor Generator Network），および，DSG群とSDM群を結合するGDN（Gemerator Disk Network）をマトリックススイッチバスで実現する。

プロトタイプデータベースマシンでは，最大100×100程度の結合が必要となると考えられる。このような結合を実現するための結合方式を検討する。そのような結合方式の1つと考えられる多段スイッチング・ネットワークを実験評価するために，2×2の基本ネットワークのVLSIを試作する。

B. ポイント

前期に開発する実験機では，結合ネットワークには重点を置かず，マトリックススイッチ・バスを採用する。

前期のポイントは，これとは独立に，プロトタイプデータベースマシンの結合方式の基礎研究にある。特に，多段スイッチングネットワークの採用の可能性を評価するために，2×2の基本

素子のＶＬＳＩを試作する。

データフローマシン開発グループとも連係を保つ。

C．作業内容

a．マトリックススイッチ・パスの開発

b．100×100程度の結合のための結合方式の研究

c．多段スイッチングネットワークの評価実験のための２×２の基本素子のＶＬＳＩ開発（発注による）

D．開発計画

		57年	58年	59年
マトリックス・スイッチ・パス		開　発		
研究者		1　人		
技術者		3　人		
結合方式の研究		・基礎研究 ・２×２のスイッチングネットワーク設計	・基礎研究 ・２×２のスイッチングネットワーク設計	・基礎研究 ・２×２のスイッチングネットワーク試作
研究者		3　人	3　人	3　人
技術者		5　人	5　人	5　人
計算機使用時間	TSS	100h	100h	100h
	CPU	5h	5h	5h

6.2.3　記憶階層システムの開発

A．目　　標

関係データベースマシンにおける記憶階層システムの開発において，その目標とするところは，近年の益々大容量化してくるデータベース処理を実行するために，いかに高速で効率の良い，かつ関係データ処理指向の記憶階層システムを実現するかである。具体的には下記の項目に関し，各々研究し明らかにせねばならない。

(a)　データベースの大容量化に適合した記憶階層方式

(b)　大容量データベースの高速アクセスを実現するための記憶階層方式

(c)　検索と更新を連想的に処理することの出来る連想ディスクコントローラのアーキテクチャ

(d) 連想ディスクコントローラの利用技術と効果

(e) 検索と更新処理を高速化するための半導体ディスクの方式

(f) 高速半導体ディスクの利用技術と効果

(g) 関係データのセグメンテーション技法とアドレスマッピング方式

(h) 記憶階層システムと上位プロセッサ間のインタフェース仕様

上記項目に関し前期，中期に分けて研究するものとする。

前期における記憶階層システムを図 6.2.1 に示す。前期における主要研究項目は，連想ディスクコントローラと半導体ディスクの機能確認と各記憶装置の容量およびスループットの最適化調査とする。このためディスクの入出力制御等の従来と同様の機能については，入出力制御プロセッサとして汎用プロセッサを使用し実験を行う。

中期における記憶階層システムを図 6.2.2 に示す。中期マシンにおける目標は，実際の大規模な関係データ処理を，プロトタイプマシンに組み込まれた中期記憶階層システムにおいて実行し，その機能・性能を確認することである。

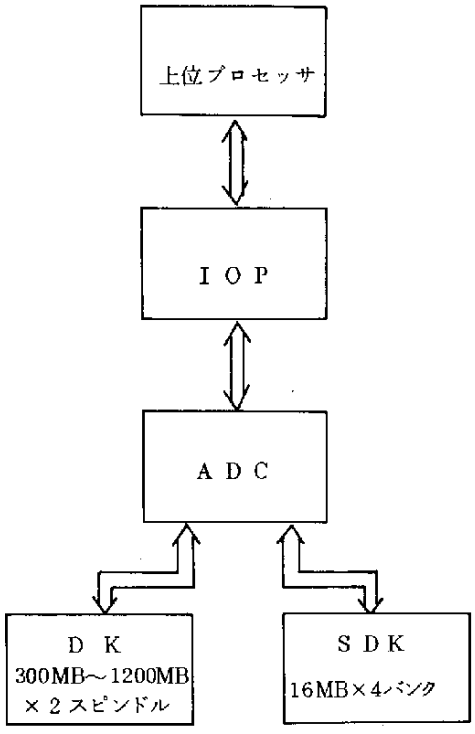


図 6.2.1　前期記憶階層システム

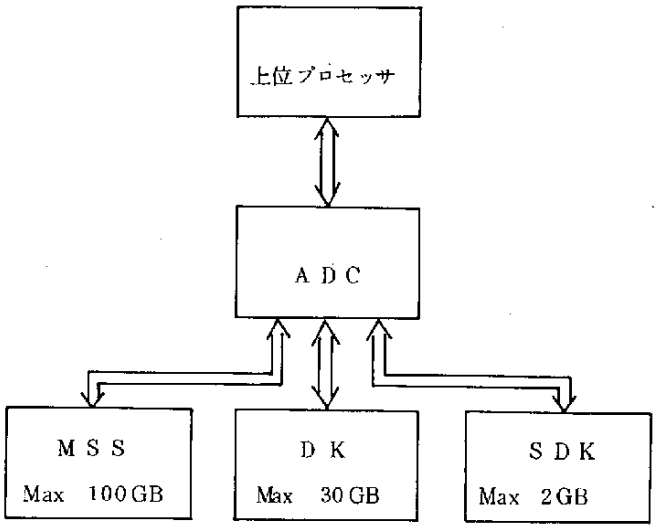


図6.2.2　中期記憶階層システム

前記および中期の記憶階層システムの主要相違点は，第１に中期では大規模な関係データ処理が扱えるように，半導体ディスクおよび大容量ディスクの容量を大幅に拡張すると共に，ＭＳＳの接続を実施する。第２に連想ディスクコントローラとして，前期ではＩＯＰとＡＤＣに分離していたものを，ＶＬＳＩ化することにより１台の専用プロセッサとしてまとめ，大幅に性能およびデータ転送効率を上げるものとすることである。第３にはＭＳＳを接続することにより，関係データベースマシンにおける本格的なシステム全体のＲＡＳＩＳ機能を確認することである。

B．ポイント

記憶階層システムは，実際の連想ディスクコントローラ，半導体ディスク，大容量ディスク装置を設計し，これらの装置を関係データベースマシンの実験機に組み込み，動作させることにより評価するものとする。記憶階層システムの開発および評価にあたってのポイントを，以下に記述する。

今回の記憶階層システムにおける第１の特長となるものは，連想ディスクコントローラと半導体ディスクの組み合わせである。これは従来システムにおけるディスクキャッシュに代表されるような，ディスクファイルと処理装置の主記憶との中間に中間メモリを設けることにより，実効的なファイルアクセス時間を短縮しようという技術と，ＣＡＦＳあるいはＩＦＣに見られるような，ディスクファイルの制御部にあたる位置にデータベース処理指向の機能を付加し，実効的なデータベース処理速度を高める技術とを融合し，両方の利点を兼ね備えた記憶階層システムを実現することを狙ったものである。

半導体ディスクに関しては，最新の半導体技術を活用することにより，従来の高速ディスクあるいはディスクキャッシュと比較し容量およびアクセス速度の両面において優れたものとする。半導体ディスクの制御は，ファイルアクセスの高速化を目的とした事前ステージングおよびローカルなファイル更新のためのワーキングエリア等に使用可能とするため，ソフトウェア制御も可能とする構造にする。

連想ディスクコントローラは，接続される半導体ディスクおよび大容量ディスク装置に対し連想アクセスを可能とし，直接に接続されているディスク群に含まれるデータベースの範囲内で，ローカルなファイルの検索および更新処理を可能とする。

これらの連想ディスクコントローラに要求される処理は，将来多種多様に変化することも考えられ，連想ディスクコントローラの制御方式は，柔軟性の高いマイクロプログラム制御とし，高度にファームウェア化するものとする。

連想ディスクコントローラのアーキテクチャ上での最大のポイントとなる項目は，いかにディスク群に対する連想アクセスを可能とするかの方式である。連想アクセス処理に必要な連想メモリの仕様に関しても，その容量，比較する際のデータ幅，処理速度等の要求仕様も，そこで処理される関係データの大きさ等に密接に関連し，最適な仕様を決定することが研究課題となる。

データベースの大容量化技術に対する今回の記憶階層システムのポイントは，いかに半導体ディスクを利用し，大容量ディスク群に対する実効的なアクセス処理の高速化を実現するかである。今回の半導体ディスクは，従来のディスクキャッシュと異なり，一度使用したデータエリアを再度使用する際のアクセス時間を高速化するのみでなく，ソフト的にあらかじめ使用するデータエリアを，事前に大容量ディスクより半導体ディスクへステージングすることにより，実効的なアクセス時間を短縮することも目的とする。このため，関係データ処理におけるソフトウェア面からの半導体ディスクの利用技術の検討も，研究課題の１つである。

また，一台の連想ディスクコントローラに接続される半導体ディスクおよび大容量ディスクの容量の関係も，そこで処理される関係データの大きさ，処理量によりその最適容量が異なるため，ハードウェアのアーキテクチャ上は，比較的柔軟に容量を変えられるよう，考慮しておく必要がある。半導体ディスクと大容量ディスクの最適容量の決定も，今回の実験システムの研究課題である。

半導体ディスクのアーキテクチャ上の特長は，記憶媒体がＭＯＳメモリであることから，従来のディスクと比較しアクセスが高速であること，およびアクセス方法に制約が無という利点があり，データ転送幅，インタリーブ数，メモリバンク容量等の論理構造を変えることにより，種々の仕様の半導体ディスクを容易に実現出来ることである。関係データベースマシンにとって，ど

のような論理構造を持った半導体ディスクが最適であるかが，実験システムにおける半導体ディスクのアーキテクチャ上の研究課題となる。

また，半導体ディスクにおける記憶素子に比較論理を実装することにより，半導体ディスク全体を連想メモリ化することも技術的には可能であり，このようなディスク全体を連想メモリ化したものが，関係データベースマシンにとって有効であるかどうかの検討も今後の研究課題である。さらに場合によってはロジックインメモリあるいはBORAM等の，関係データベースマシン専用の半導体素子開発の提案も必要となろう。

実験機においてはＭＳＳを接続しないが，大容量ディスクのスピンドル間で自動的にバックアップファイルをとる等のＲＡＳ機能は，確認可能である。しかし，実際のＭＳＳの接続およびＭＳＳを含めたＲＡＳ機能確認は，今後の研究課題となる。

C．作　　業

実験機における記憶階層システムは，図6.2.1に示すように入出力制御プロセッサ，連想ディスクコントローラ，半導体ディスク，大容量ディスクを各々数台を試作するものとし，ＭＳＳは接続しない。実験機のアーキテクチャは，充分に拡張性を考慮し，その最大構成時には可能な限りプロトタイプマシンのイメージに合致するようにする。

(a) 入出力制御プロセッサ

上位プロセッサより発行される指令を解析し，リレーション名より実際にアクセスするディスクアドレスを生成し，実行する内容にもとづいてディスク制御コマンド群を生成し，連想ディスクコントローラに対し指令を発行する。

ディスク制御コマンド群には従来のディスクと互換性を持ったコマンド群に加えて，連想ディスクコントローラにおけるローカルなファイルの検索および更新処理を指令するためのマクロコマンドを新設する。

入出力制御プロセッサには汎用プロセッサを利用し，その制御ソフトウェアのみを開発するものとする。入出力制御プロセッサと連想ディスクコントローラの接続は，従来の入出力インタフェースを使用しスループットは３ＭＢ／秒程度とする。この連想ディスクコントローラから上位プロセッサへのデータ転送能力に対する必要性は，連想ディスクコントローラにおけるフィルタリング機能およびローカルな更新機能がどの程度有効に働くかにより変るので，実験機において研究成果が中期プロトタイプマシンにおける連想ディスクコントローラのデータ転送能力に反映されねばならない。

(b) 連想ディスクコントローラ（ＡＤＣ）

実験機においては6.2.3で示した連想ディスクコントローラを数台試作するものとする。フ

ファームウェアに対する各種要求に柔軟に応えられるように，マイクロプログラム格納用メモリは可能な限り充分大きくとることが望ましい。

(c) 半導体ディスク（ＳＤＫ）

実験機においては図5.4.3で示した半導体ディスクと同一の論理構造で１６ＭＢ×４バンク程度のものを試作するものとする。

但し，論理構造および実装構造ともに容易に容量拡張が実施可能なようにしておく。

(d) 大容量ディスク

実験機においては300～1200MB/スピンドルのディスクを数スピンドル程度接続するものとする。但し，連想ディスクコントローラ１台には最大３２スピンドルまでのディスクを接続可能な構造にしておく。

D. 詳細スケジュール

	５７年	５８年	５９年
ＩＯＰ（ソフトウェア）	基本設計	制御ソフト設計	試作／評価
	研　２人 技　１人	研　４人 技　５人	研　４人 技　６人
ＡＤＣ	基本設計	論理設計 ファームウェア設計	試作／評価
	研　３人 技　２人	研　４人 技　５人	研　４人 技　６人
ＳＤＫ	基本設計	論理設計	試作／評価
	研　１人 技　１人	研　２人 技　２人	研　２人 技　２人
合計	研　６人 技　４人	研　１０人 技　１２人	研　１０人 技　１４人

研：研究者
技：技術者

6.3 関係データベースマシン実験機の開発

6.3.1 実験機設計と開発

A. 目　　標

知識情報処理で要求されるプロトタイプデータベースマシンの開発に先行して、それに必要な基礎技術を構築することを目的に、関係データベースマシン実験機を開発する。

すでに、プロトタイプデータベースマシンについて検討してきたが、これらの検討成果をベースにして、　　前期3年間に実現できる形態で実験機の設計および開発を進めることとする。

このために、

1）プロトタイプデータベースマシンについて検討することにより得られたいくつかのマシンイメージの中から、実験機として要求される機能を満すものを選び実現する。また、同検討時に提案された各種処理技法についても、適宜取捨選択することにより、実験機に採用する。

2）プロトタイプデータベースマシンは本格的なＶＬＳＩで構成されるものと考えられるが、実験機においては、将来ＶＬＳＩ化される各種機能モジュールについて、既存のディスクリートＩＣを用いて構成することにより得られたものを用いることとする。

3）実験機を構成する各種プロセッサの規模や結合ネットワークの機能・規模については、プロトタイプデータベースマシンの開発に必要な基礎的データを得るために要求される機能・規模にとどめることとする。

これらの目的を実現するために、以下の目標を達成することを念頭において、研究を進めることが要請される。

- 複雑な関係演算処理を高速に実行できる方式を確立し、これを組み込んだ実験機を開発する。

  処理速度としては、汎用大型計算機の数倍～十数倍程度を目標とする。
- 大容量のデータベースを蓄積し、処理できる記憶機構を備える。

  容量としては、数ＧＢ～十数ＧＢ程度を目標とする。
- データベースマシンを効率良く構成するための機能モジュールを明確にするとともに、試作し、実験機に組み込む。そして、これをもとに、さらに、高性能な機能モジュール開発に備える。
- 定量的な詳細な評価を行えるように、実験機の動作状況に関するデータを収集できる機構を備えるようにする。
- 開発された実験機を用いて、各種の制御方式、構成技法、等について、シミュレーション

評価を行えるような構造を実現する。

・ 知識情報処理の実験を行うためのツールとして使用できるシステムとして作成する。

B．研究開発のポイント

関係データベースマシン実験機の研究開発のポイントは知識ベースマシンとして利用できる関係データベースマシン・プロトタイプに先行して、実験機として開発するものであり、実験機として必要な機能を明確にし、実験機のアーキテクチャ、ハードウェア構成およびソフトウェアを研究開発することである。そして、この実験機にもとづいて、プロトタイプ開発の基礎を築くことである。以下に、それぞれのポイントについて述べる。

a．必要機能・性能の明確化

知識ベースマシンとして利用できる関係データベースマシンプロトタイプを開発することが最終目的であり、従って、知識ベースマシンとして必要な機能が実験的にも実現されていることが必要である。このために、知識ベースマシンとしての必要機能・性能を明確にし、それを実現し、方式の評価を行えることが要求される。

そのために、特に、知識情報処理でのデータベースクイアリの特徴分析、頻度解析を行うとともに、それを反映する構成・方式を採用することが第1のポイントである。さらに、知識情報処理で要求されるデータベースの規模、それに対するトランザクションの要求速度を明らかにし、それらを満たすような構成・方式を採用することが第2のポイントである。

b．処理方式の確立

前述の機能・性能の明確化に続いて、関係データベースマシン実験を研究開発する時に重要なポイントは関係演算子を処理するためにどのような処理方式を採用するかである。

知識情報処理で要求されると予想される大容量のデータベースに対して、複数個の関係演算子から成る複雑なクイアリを効率良く処理する方式として、ロジックパートラック[2)]、連想処理[3)]、動的クラスタリング[5)]、データストリーム処理[4),5)]、等の処理技法を組み合わせた高度な並列処理を行える方式を選択することが求められる。

c．実験機のプロセッサ構成の検討

高性能な実験機を実現する時に考慮しなければならないポイントはプロセッサをどのように構成するかであり、これについては、大規模、高速化するＶＬＳＩを駆使した構成を考えることである。このような観点から考えると、シストリックアレイ[6)]を用いる方式、高速のソート、サーチエンジンを用いたデータストリーム処理方式、等にもとづいたプロセッサ構成を追求することが要望される。

また、知識情報処理で要求されるといわれる大容量のデータベースを蓄積、管理するために、

新しいデバイスを用いた2次記憶装置（例えば，半導体デイスク，等）を活用したプロセッサ構成を考えることが重要である。

d．機能モジュール，記憶モジュールの設計・試作

本実験機を開発することの1つの目的として，プロトタイプを開発する時の核となる構成モジュールを確立し，これらを設計・試作することである。従って，プロトタイプとして活用できそうな機能モジュールや記憶モジュールの機能・性能仕様を明確化することが重要なポイントである。

e．結合方式と結合ネットワーク

高速処理を実現する時には，複数個のプロセッサ，機能モジュールが結合し，動作する構造が予想される。このような構造において，達成される性能を左右するのはプロセッサ自体のスピードとともに，プロセッサ間の結合方式，結合ネットワークであると考えられる。

従って，関係データベースマシン実験機を実現するに際して，高性能な結合方式，結合ネットワークを開発することが重要なポイントと考えられる。

f．評価データの収集機能

関係データベースマシン実験機は各種の機能モジュール，記憶モジュール，結合ネットワーク，等から構成されるが，完成された実験機をもとに各種の評価を行い，プロトタイプ開発に必要な評価データを収集できる機能を備えていることが要求される。

このためには，実験機を構成しているモジュールやネットワークについての機能，性能，容量を変更できる柔軟な構造を備えるとともに，これらの使用状況についてのデータを収集するための機構を組み込むことが必要である。

C．作業内容と進め方

以上述べた目標を達成するために行わなければならない作業内容と作業の進め方について述べる。作業内容の具体的な作業項目については以降で述べる。

a．システム基本検討

関係データベースマシン実験機の開発に当たり，先ず，実験機システムの基本検討を行う。

基本検討としては、

- システム構成の検討
- データベースの配置と管理方式の検討
- クイアリの処理方式の検討

等を中心に行い，関係データベースマシン実験機システムと実験機ハードウェアのイメージを明白にする。この作業は，6.1，6.2で述べた方式研究，要素技術・要素モジュールの開発と

関連をもって進める。

b．実験機のハードウェア構成の設計

システム構成および実験機ハードウェアのイメージが明らかになると，これにもとづいて実験機のハードウェア構成を検討し，設計を行う。

実験機のハードウェア構成の検討，設計に当たっては，先ず，要求される処理を行うために必要な機能の分析を行い，それを効率良く処理する機能モジュールを明白にする。そして，これらの機能モジュール間の結合インタフェースを規定し，それに必要な結合ネットワークを明らかにする。

次に，設計・試作された機能モジュール，結合ネットワークを融合し，結合・検査を行い，実験機ハードウェアを完成させる。また，同時に開発されるソフトウェアと結合して，総合システム検査を行い，システムとして，完成させる。

c．機能モジュールの設計・試作

ハードウェア構成の設計において規定された各種の機能モジュールについて，詳細な仕様検討を行い，これにもとづいて，設計および試作を行う。

ここで使用する機能モジュールの一部は，6.2で述べられた要素技術・要素モジュールの開発において試作されるものを利用する方針で作業を進める。

d．結合インタフェースの設計・試作

ハードウェア構成の設計において定められた結合インタフェースについて，詳細な仕様検討を行い，設計および試作を行う。

ここで用いられる結合インタフェースとしては，6.2で述べられた要素技術・要素モジュールの開発において試作されるものを利用する方針で作業を進める。

e．ソフトウェアシステムの設計・試作システム基本検討の中でのシステム構成の検討により得られた検討結果にもとづいて，関係データベースマシン実験機を動作させるのに必要なソフトウェアを明確にし，詳細設計・作成を行う。

これを行う時には，出来る限り，既存のシステムを流用することを念頭におき，関係データベースマシン実験機として固有のものを中心に試作するように作業を進める。

D．作業項目

以上述べた作業内容を実施するための具体的作業項目とそれぞれの実現例について詳細に検討する。

a．システム構成

関係データベースマシン実験機を実現する形態として，全ての機能を新たに設計し，開発する方式と，前記目的・目標を達成するために必要な最低限の機能について開発し，他の機能に

ついては，既存のものを使用する方式とが考えられる。

前期3年間において，関係データベースマシン実験機を完成させるとともに，知識情報処理に要求される機能・性能を満たすような斬新で高性能な方式を採用したマシンを実現するためには，関係データベースマシンの中核となる機能の研究開発に集中することが望ましいと考えられる。例えば，各種の入出力装置のための制御機能，応用プログラム開発のためのサポートツール，等については，出来る限り，既存のものを流用することが賢明である。

このようなことを考慮して，関係データベースマシン実験システムの構成を決定することが必要であり，図6.3.1にシステム構成の一例を示す。

本システム構成では，関係データベースマシン実験機はホストプロセッサに接続されて動作し，バックエンドプロセッサとして働く。そして，関係データベースマシン実験システムは関係データベースの処理だけを行い，他の機能は全て，ホストプロセッサに依頼することになる。

ホストプロセッサは例えば，高性能な汎用ミニコンピュータ（実験環境にも依存するが，場合によって，大型機が要求される可能性もある）で実現し，入出力装置の制御，マン・マシンインタフェースの制御，各種の高級言語プログラムのサポート，等を行うとともに，関係データベースマシン実験機の制御を行う。

ホストプロセッサと関係データベースマシン実験機との結合として，I／Oインタフェース，主記憶インタフェース，内部バスインタフェース，等が考えられ，情報の転送速度と転送量に対する要求をもとに決定される必要がある。

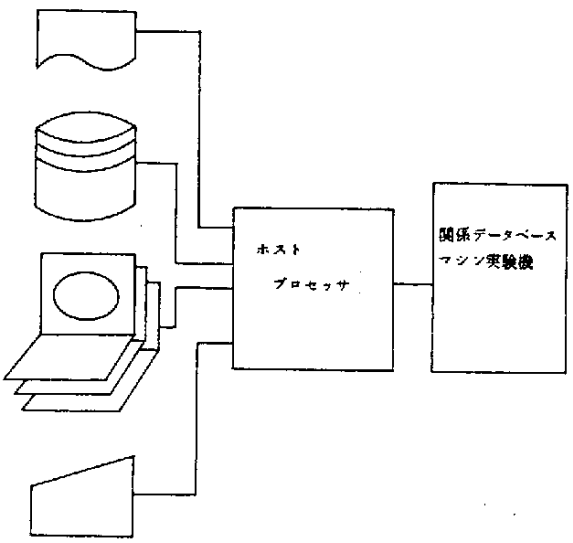


図6.3.1　関係データベースマシン実験システムの
システム構成

b：データベースの記憶割り付けと管理方式

関係データベースシステムでは，データベースは複数個のリレーションから構成される。また，各リレーションは複数個のタプルから成り，タプル集合として見做すことができる。そして，1つのリレーションをとってみても，本報告書が示されているように，大きいもので十Gバイトぐらいのものがあり，これ全体を1つとして取り扱うことは能率的でない。また，本関係データベースマシン実験機で導入されると考えられるデータストリーム方式等では，1つのリレーションを分割して，記憶割り付けを行うとともに，管理を行うことが要求される。さらに，リレーションの分割は，1つのリレーションの1部をサブリレーションと見做し，これに対してアクセスする場合のアクセス権の検証の単位としても必要とされる。

このような観点から，1つのリレーションは複数個に分割し，記憶割り付けと管理を行うことが望ましいと考えられる。

1つのリレーションを分割する方式として，リレーションのタプル方向に分割する（タプルワイズ：tuple － wise）のと，リレーションのカラム方向に分割する（カラムワイズ：Column － wise）のとが可能である。いずれの分割が良いかは，行われる関係演算に依存するが，一般的には，タプルワイズの分割がよく用いられている。

図 6.3.2 は，1つのリレーションをタプルワイズに分割した時の例を示す。複数個のタプルから成るサブリレーションは記憶割り付けや管理，および処理の基本単位となり，その中のタプルは記憶装置内で連続して配置される。

従って，このサブリレーションは処理，制御の観点から1つのオブジェクトとして扱うことが望ましく，このために，オブジェクト記述子による規定が有効と考えられる。また，複数個のサブリレーションから成る1つのリレーション全体も1つのオブジェクトとして取り扱われる。

オブジェクト記述子は複数個のタプルから成るサブリレーションを規定するための情報として，記憶アドレス，タプルの構成を示すアトリビュート記述，記憶保護のためのアクセス権，等を備えることが要求される。

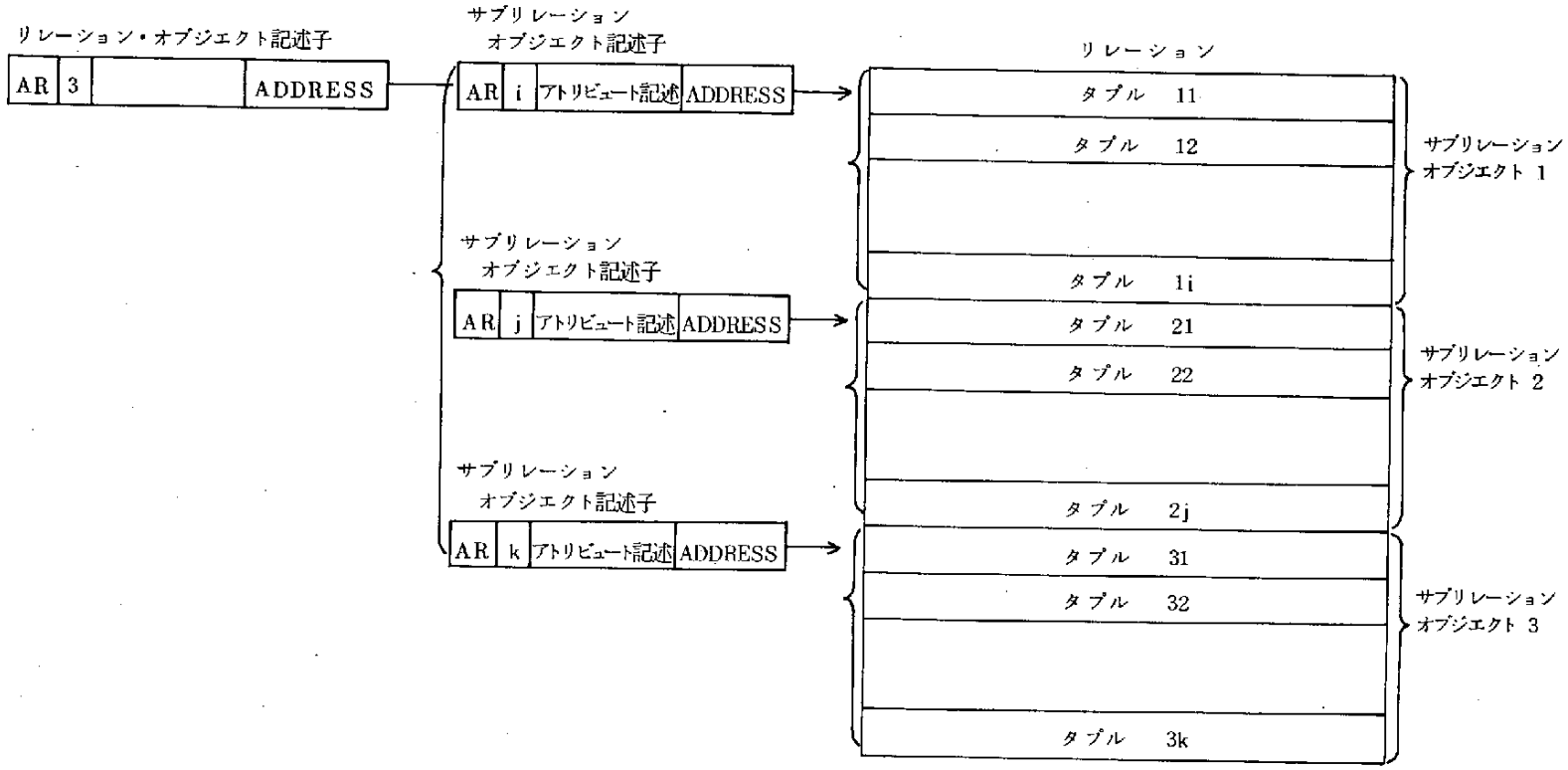


図6.3.2　リレーションをタプルワイズに分割した例

c、クイアリ（Query）展開と処理方式

知識情報処理においては，大量の知識（facts）が関係形式のデータベースとして蓄積され，これに対して，多数の関係演算子から構成されたクイアリの実行が要求されると考えられる。このような大量のデータベースに対する複雑なクイアリを高速に処理する方式として，複数個のオペレーションを並列に実行することと，大量のデータベースを分割し，分割されたデータベースを並列に処理すること，が有効と考えられる。

このような2つの並列処理を行う方式として各種考えられるが，図 6.3.3 に処理方式の一例を示す。ここでは，4つのリレーション（R_1，R_2，R_3，R_4）に関する3 joins 4 restrictions〔$R_{12} := R_1(A_1=A_2)R_2$，$R_{34} := R_3(A_3=A_4)R_4$，$R_{12}(A_5=A_6)R_{34}$〕の場合を示している。

各リレーションは4つのサブリレーションに分割されて記憶装置に割り付けられている。

このクイアリは以下のステップで処理される。

(1) 4つのリレーションに対するRestrictionsが並列に，かつ，1つのリレーションは4つに分割されて並列に実行される。そして，Restrictionの結果，得られたタプルは，次のJoinのために，Join で用いられるアトリビュートについてハッシュを行う。その後，そのハッシュ値によって分割しておく。

(2) 次に，R_1とR_2，R_3とR_4のJoin オペレーションを並列に実行する。この時，1つのJoinユニットでは，リレーションR_1，R_2（又は，R_3，R_4）のJoinアトリビュートに関するハッシュ値が一定の範囲のタプルだけが送られてくる。このために，ハッシュ値について，ソートを行い，等しいものについてのマージを行うことにより，Join オペレーションを達成する。従って，ハッシュ値で分割された単位で並列に実行されることになる。次に，これによって得られたタプル集合（リレーションR_{12}とR_{34}）に対して，次のJoin のために，その時用いられるアトリビュートについてハッシュを行って，そのハッシュ値により分割しておく。

(3) 次に，2段目のJoin オペレーションをステップ(2)と同様に実行する。この時には，1つのJoin ユニットには，リレーションR_{12}とR_{34}のハッシュ値がある範囲のタプルが送られてくることになる。

以上，1つのクイアリ例をもとに，処理方式について述べたが，この処理方式を一般的に示すと，図 6.3.4 のようになる。図 6.3.4 (a)は全体イメージ，図 6.3.4 (b)は論理的な処理イメージ，図 6.3.4 (c)は詳細イメージを示している。Data Stream Processor（DSP）は処理を行う部分で，図 6.3.3 のJoin ユニットに対応し，Data Stream

Generator（ＤＳＧ）はリレーションの蓄積と該当するアトリビュートについてのハッシュ値に従って，特定のタプル集合を取り出す処理，等を行う。

従って，本方式では，ハッシュ値をもとにグループ化して処理するために，オペレーションの対象にならないタプルを前もって除くことになり，高い処理効率を達成できる。例えば，タプルレベルのソートはオーダ（$\frac{N}{m}$）〔ただし，Nはタプル数，mはプロセッサの台数〕で処理できることになる。

また，ＤＳＰでは，特定のタプル集合が連続して送り込まれ，これに対してパイプライン形式で処理されて，送り出されるため，データストリーム方式と考えられる。

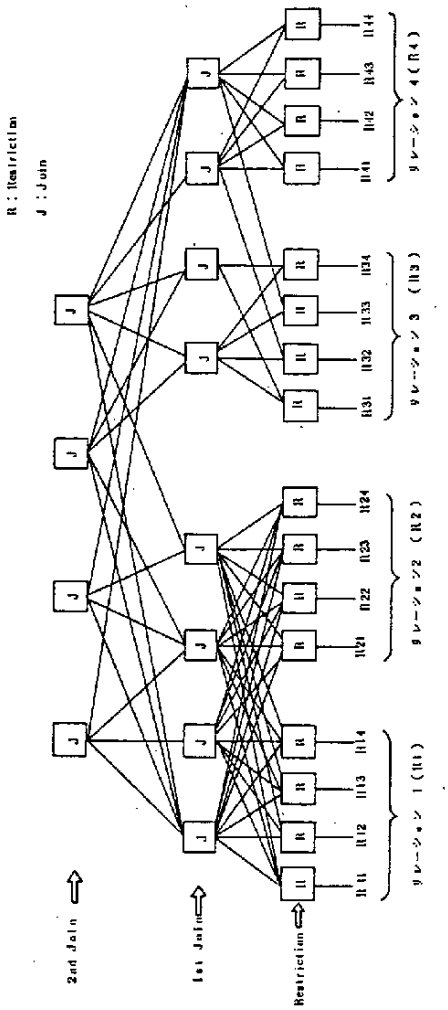
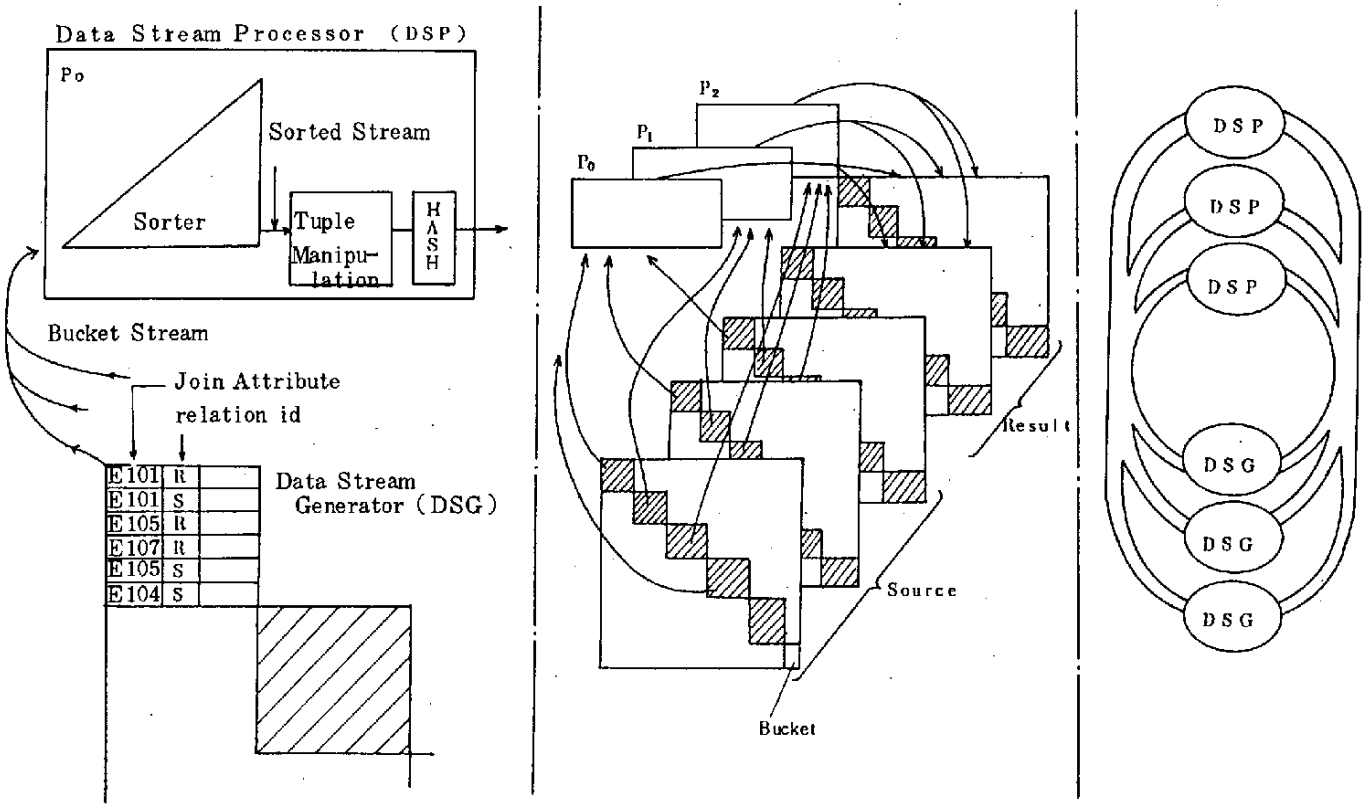


図6.3.3　処理方式の例（３joins 4 restrictions）



(a)全体イメージ　　(b)論理イメージ　　(c)詳細イメージ

図 6.3.4　データストリーム方式の処理イメージ

d．実験機のハードウェア構成

これまでに述べた処理方式を実現するために，関係データベースマシン実験機のハードウェア構成について検討する。目標の機能と性能を達成するための要素技術として，

- データストリーム方式
- オブジェクト指向
- マルチプロセッサ方式
- データフロー制御
- パイプライン処理方式

等を導入することが望ましい。

これを考慮して実現する場合のハードウェア構成の一例を図 6.3.5 に示す。

本構成では，データベースの蓄積とフイルタリング処理などを行うSecondary Data Manager（ＳＤＭ），タプルで構成されるデータストリームの生成と管理を行う Data Stream Generator（ＤＳＧ），関係演算を実行するData Stream Processor（ＤＳＰ）が関係演算処理の主要部を実行する。これらの他に，ＳＶＣ（Supervisor Controller）は実験機ハードウェア全体の制御とホストプロセッサとのインタフェースを制御するものである。ＳＰＣ（Stream Processing Controller），SGC（Stream Generation Controller），SDC（Secondary Data Controller）は，各々ＤＳＰ，ＤＳＧ，ＳＤＭを制御すると共に，ＳＶＣとのインタフェース制御を行う。DDT（Data Directory Dictionary Processor）はＳＶＣの指示のもとで，ＳＤＭに蓄積されているデータベースのデイレクトリ管理，等を行う。

SPCに接続されたMath Processor（MP）は，集計演算，等の処理を行うために用いられる。

ＤＳＰ，ＤＳＧ，ＳＤＭの間は，各々，結合ネットワークを介して接続される。

本ハードウェア構成において，実験機として実現する場合には，その目的が達成される規模を選定することが要求されるが，例えば，ＤＳＰは８個，ＤＳＧは１６個，ＳＤＭは４個程度の構成が望まれる。

これらのハードウェアを実現する場合には，Processor，Controller，等において，共用できる可能性があるものについては，出来る限り，共用することが望ましい。

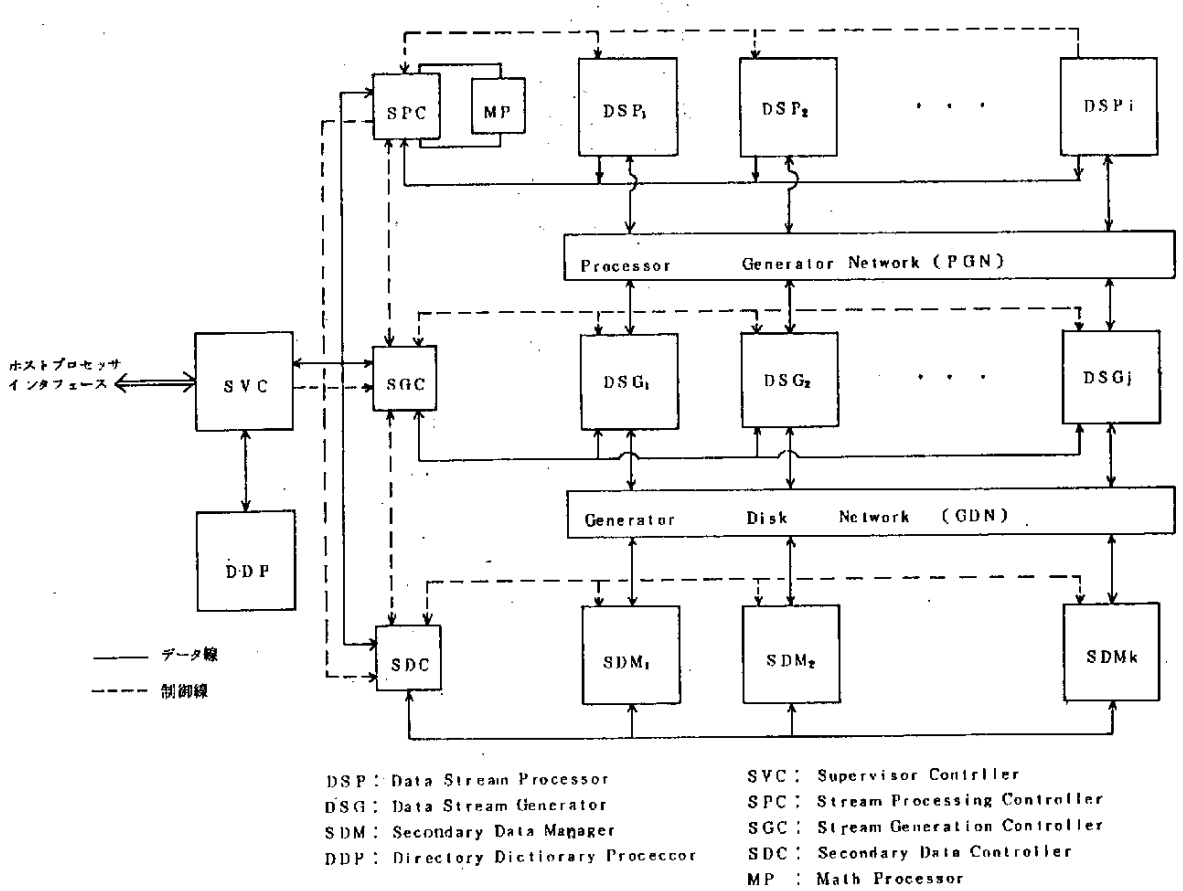


図6.3.5　関係データベースマシン実験機のハードウェア構成例

e．機能モジュールの仕様設計

図 6.3.5 で示した関係データベースマシン実験機ハードウェアを構成する機能モジュールについて設計する。

(1) Secondary Data Manager（SDM）

図 6.3.6 はSDMにおける処理の流れとデータが変わる過程とを示している。

SDMに要求される機能は以下の通りである。

- データベースの蓄積と管理

大容量のデータベースを扱うことができるためには，仮想記憶方式が実現されていることが要求されるため，1つのSDM内での仮想記憶制御を行う必要がある。また，高速のデータベースアクセスを可能とするために，全データベースを大容量の記憶デバイスに蓄積するとともに，使用頻度の高い一部のデータベースを高速記憶デバイスに蓄積し管理することが望ましい。

- フイルタリング処理

関係代数演算においては，1つのリレーションに対して，特定のカラムやタプルを選択抽出するSelectionやRestriction等の演算がJoinやProjection の処理に先行して施されることが多い。このような処理を高速化するために，データベース記憶デバイスより取り出し，JoinやProjection等の処理を行う演算ユニットへ送る過程で処理を完了してしまう方式が有効である。図 6.3.6 におけるFilteringがこのための処理を行う。

- クラスタリング処理

関係演算のJoin やProjection に共通する前処理操作として，指定されたアトリビュートに関してリレーションを再構成し，クラスタ化することである。そして，クラスタ化することにより，JoinやProjectionにおいて，各クラスタ内での並列処理を可能とするものである。

クラスタ化において重要なことは，各クラスタに属するタプル数のバラツキを小さくすることが望ましい。

この処理は図 6.3.6 におけるClustering 部で行われるが，これを実現する方式として，Hash 関数を用いる方式，等が考えられている。

クラスタ処理が完了すると，グループを示すタグ（Hash 方式ではHash 値）が付加されて出力されるが，このタグに応じて分解され，それに対応するData Stream Generatorに各々送られる。

・ ローカル・アップデート

以上の処理の他，ＳＤＭに蓄積されたデータベースをユーザの要求に応じて更新することが必要になるために，ＳＤＭで独立に更新できる（Local Update）ことが望ましい。

図 6.3.7 はこれまで述べた機能を実現するためのＳＤＭのハードウェアの構成例を示す。

ＤＩＳＫは全体のデータベースを蓄積するための記憶デバイスで，記憶容量は数ＧＢ（例，2.4ＧＢ）程度のものが望ましい。

Semiconductor Disk（ＳＤＫ）は半導体ディスクで構成され，記憶容量は数十ＭＢ（例，６４ＭＢ）程度のものが望ましい。そして，SDKは，Local Update，等のワーキングエリアとして使用される。

ＡＤＣ（Associative Disk Controller）はDISK，SDKの制御，等を行い高性能なマイクロプログラム制御ユニットで実現される。

ＩＯＰ（Input Output Processor）はＳＤＭ全体の制御，ＧＤＮやＳＤＣとのインタフェース制御，およびHash処理，等を行うものである。

ＳＤＭは，6.2で試作されるものを利用する方向で進めることが望ましい。

(2) Data Stream Generator（ＤＳＧ）

ＤＳＧはＳＤＭやＤＳＰ（Data Stream Processor）から送られてくるサブリレーションの蓄積，ＤＳＰが要求するサブリレーションの生成，等の処理を行い，ＤＳＰのワーキング用メモリとしての役割を果す。ＤＳＧでは以下の機能を満たすことが要求される。

・ サブリレーションの管理

特定アトリビュートにもとづいてクラスタ化されたサブリレーションが複数個に分割されてＤＳＧの中で保存される。従って，クラスタ値とそれに該当するサブリレーションの記憶位置との関係を管理し，要求があった場合に高速にそのサブリレーションをみつけることが可能であることが必要である。

・ サブリレーションの生成

ＤＳＰでの処理が要求されると，指定されたクラスタ値に対応するサブリレーションを取り出し，データストリームとして要求のＤＳＰへ高速に送り出すことが必要である。従って，グループ値から，該当するサブリレーションをみつける高速な機構を備えることが要求される。

・ 結果リレーションの管理

ＳＤＭやＤＳＰで処理され，クラスタ化されたサブリレーションがＤＳＧへ送られてく

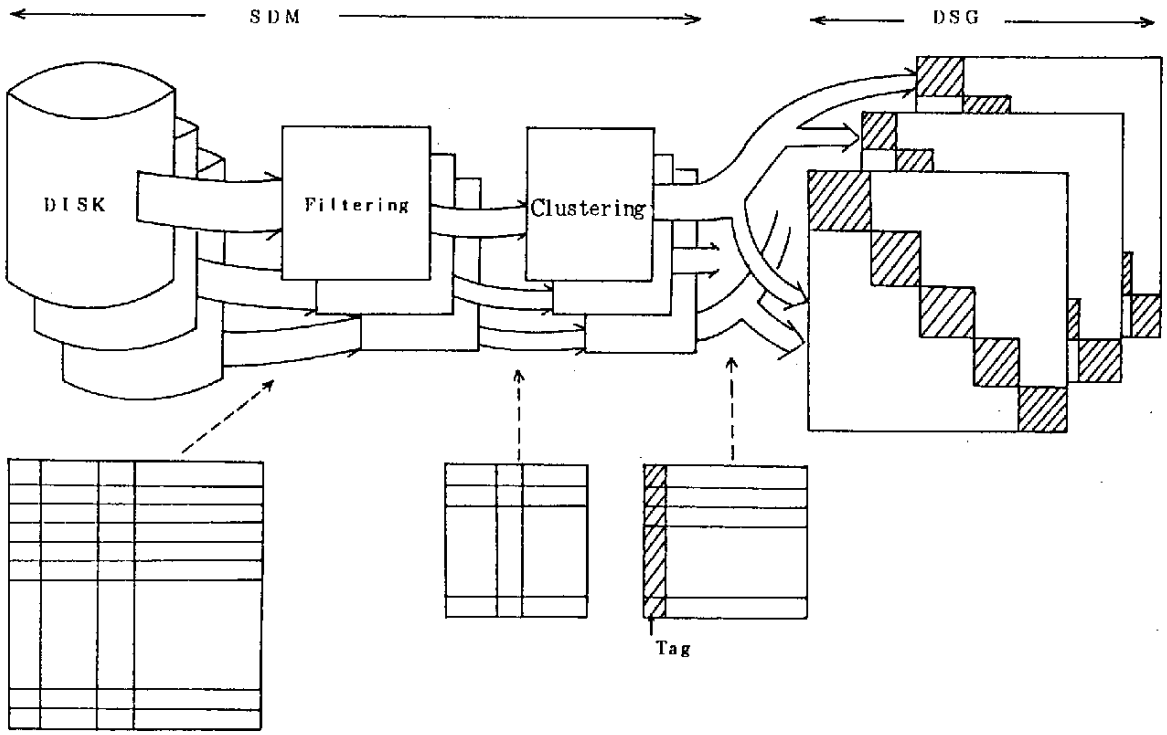
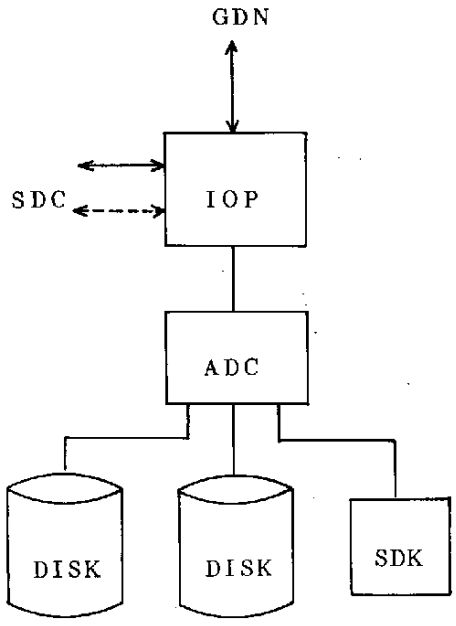


図6.3.6　ＳＤＭの処理の流れ図



IOP：Input Output Processor
ADC：Associative Disk Controller
SDK：Semiconductor Disk

図6.3.7　Secondary Data Manager（SDM）のハードウェア構成例

るので，クラスタ値に応じて区分けして，DSG内に蓄積し，管理することが必要である。

以上の処理を行うDSGのハードウェア構成例を図6.38に示す。

Bufferはサブリレーションを蓄積するもので，高速に書き込みおよび読み出しできることが必要であり，容量として十数MB（例，16MB）程度のものが望まれる。

Buffer Management Unit（BMU）はBufferに蓄積されたサブリレーションとクラスタ値との対応づけを行うもので，要求されたクラスタ値のサブリレーションがBufferのどこにあるかを見つけるために用いられる。これを高速に行うためには，例えば，連想メモリなどが考えられ，また，クラスタ値は区間で指定されることも生じるため，区間検索ができることも必要である。

Microprogram Controller（MPC）はマイクロプログラム制御ユニットで，BufferやBMUを制御するものである。SDMやDSPからの要求を解読し，BMUに

もとづいて指定のサブリレーションをBufferから取り出してＤＳＰへ送ったり，逆に，ＤＳＰからのサブリレーションをBufferに蓄積する。

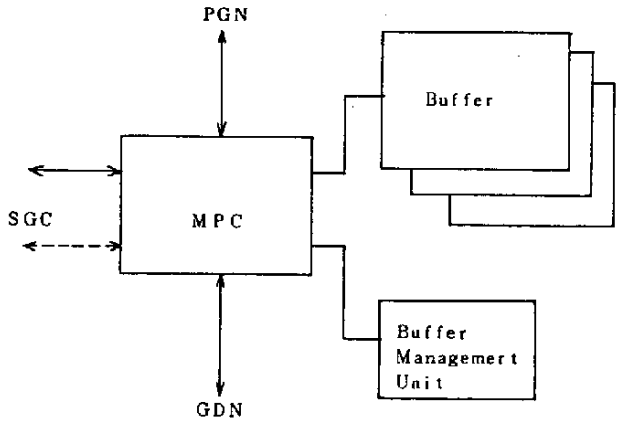


図6.3.8　Data Stream Generator(DSG)
のハードウェア構成例

(3) Data Stream Processor（ＤＳＰ）

Data Stream Processor（ＤＳＰ）は関係データベースマシンの中核をなすもので関係演算のためのTuple Manipulationを中心に行う。

図 6.3.9 は1つのＤＳＰでの処理の流れを示しており，クラスタ化された1つのサブリレーションに対して，関係演算処理が行われ，以下の機能から成っている。

・ ソーテイング（Sorting）

あるアトリビュートにもとづいてクラスタ化されたサブリレーションがＤＳＧから送られてくる。このサブリレーションを構成する複数個のタプルはクラスタリングされて付加されたタグを持っており，そのタグ値はグループ内に属する範囲の値を示している。そして，サブリレーションを構成するタプルは処理された順に送られてくるため，タグ値はバラバラになっている。このために，次のTuple Manipulationを効率良く行うために，タグ値にもとづいて，各タプルをソーテイングして，順序正しくしておくことが要求される。

ソートの方式としては，各種考えられているが，実行速度とハードウェア量にもとづいて決定されるべきであり，例えば，Ｋ－way（通常は２－way）のMerge－Sortが有効であると考えられる。

・ タプル操作（Tuple Manipulation）

関係代数演算の中核処理で，Join／Concatenation，Selection／Restriction，Projection，Arithmetic Operation，等の処理を行うものである。

これらの処理の中で，Join，Selection，Projection，等は前もってソーティングが行われているために高速に実現される。

Arithmetic Operationとしては，SUM，AVERAGE，等の集計演算を主に行うためのものである。

・ クラスタリング（Clustering）

複雑なクイアリを実行する時には，1つの関係演算が行われた後，別の関係演算が続いて行われる。このために，次のTuple Manipulationを効率良く行う前準備として，次に使用されるアトリビュートにもとづいて，すでに生成されたサブリレーションをクラスタリングしておくことが効果的である。

ここでのクラスタリングはＳＤＭでのクラスタリングと等価なものと考えられる。

図 6.3.10 は以上の処理を行うＤＳＰのハードウェア構成例を示す。

Sort Module（SOM）は指定アトリビュートに対してソートを行うもので，K－wayのMerge－Sort方式，等が有効である。Working Memory（WM）は中間データを蓄積するもので，記憶容量として数ＭＢ（例，１ＭＢ）程度のものが望ましい。On－the－fly Processor（ＯＦＰ）は重複タプルの除去，qualification，ハッシュ処理を行うもので，Math Processor（ＭＰ），Arithmetic Operationを実行するものである。

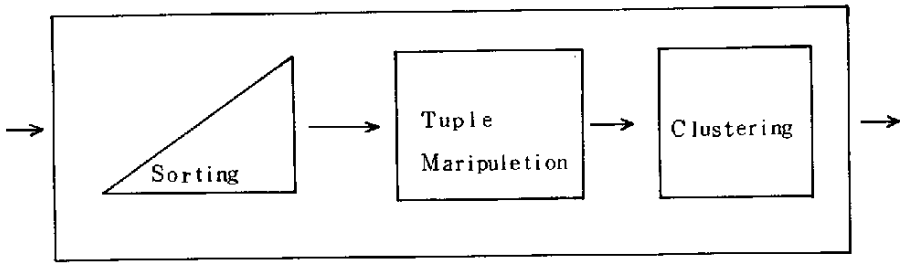
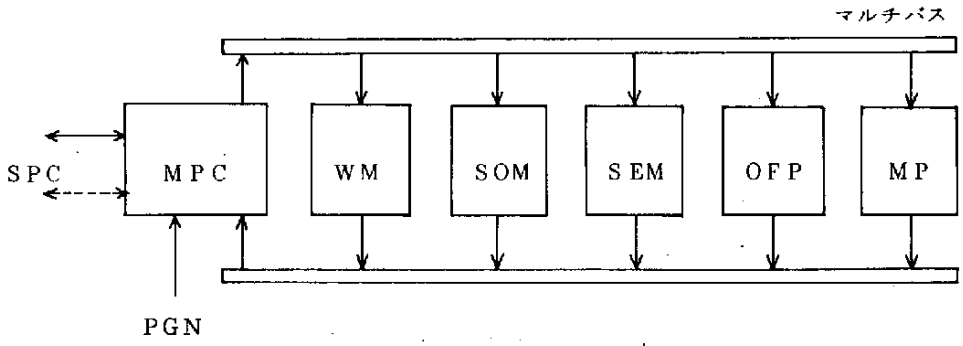


図6.3.9 Data Stream Processor(DSP)での処理の流れ



W M：Working Memory　　OFP：On－the－fly Processor
SOM：Sort Module　　MP：Math Processor
SEM：Search Module

図6.3.10　DSPのハードウェア構成例

Microprogram Controller（MPC）はDSGのものと同様に構成される。

DSPを構成するこれらの機能モジュールは，高度な並列処理を可能とするために，マルチバス，等の高性能な結合インタフェースを介して接続されることが適当である。

(4) Secondary Data Controller（SDC）

SVCの指示のもとで，独立に動作し，主にSDMの制御を行う。SDCは，各SDMに対して，その中に蓄積されているリレーションを取り出し，処理を行った結果を，GDNを介して，該当する各DSGへロードするように指示する。そして，次の処理を行うようにSGCに指示する。

また，SDCは，SVCからの指示のもとで，データベースの初期ローディングの制御も行う。

以上のような処理を行うSDCは，マイクロプログラム制御コントローラで実現され，例えば，ビットスライスのマイクロプロセッサを用いて構成することができる。

(5) Stream Generation Controller（SGC）

SGCはSVCの指示のもとで独立に動作し，主に，DSGの制御を行うものである。

SDCやSPCの指示を受けると，各DSGに対して，該当するリレーションを蓄積したり，保存されているリレーションを取り出したり，等の処理を行うように要求する。

また，一連の処理の結果によって求められたデータは，DSGに蓄積されているので，これらを外部へ取り出す時の制御を各DSGに対して行う。

以上の処理を行うSGCは，SDCと同様に，マイクロプログラム制御コントローラによ

って実現できる。

(6) Stream Processing Controller（ＳＰＣ）

ＳＰＣはＳＶＣの指示のもとで独立に動作し，主にＤＳＰの制御を行う。即，ＳＧＣから処理の要求がくると，ＰＧＮを介して送られてくるデータストリーム形式のデータに対して，各ＤＳＰがその中に備えている処理モジュールを使用して処理するように指示する。

また，ＤＳＰでの処理結果をＰＧＮを介して該当するＤＳＧへ送るように各ＤＳＰを制御すると共に，ＳＧＣに対して，次の処理を行うように指示する。また，各ＤＳＰ中に求められたデータを用いて，集計演算，等の処理をＭＰを用いて行うように制御する。

以上の処理を行うＳＰＣは，ＳＤＣと同様に，マイクロプログラム制御コントローラによって実現することが可能である。

(7) Supervisor Controller（ＳＶＣ）

ＳＶＣは，ユーザが要求するクイアリを実験機ハードウェアが処理できるコマンド列へ変換する処理，実験機ハードウェア全体の制御，ホストプロセッサとのインタフェース処理，エラー回復処理，等を行うものであり，主に以下の機能が要求される。

- クイアリ変換

  ユーザが指示するクイアリは高レベルのインタフェース言語でなされるものと考えられるため，これを，実験機ハードウェアが処理できるコマンド列へ変換することが必要になる。

- コマンド列の送出

  １つのユーザのクイアリは複数個のリレーションに対する複数個の関係演算子で表現されており，関係演算子の列で構成されていると考えられる。これらの演算子列を解読し，それらの処理を行う機能モジュールであるＳＤＭ，ＤＳＧ，ＤＳＰへコマンドの列として送り出し，実行させるものである。

- リレーションの記憶管理

  知識情報処理で要求されるような巨大データベースを蓄積でき，かつ効率良く管理できるためには，仮想記憶制御方式が採用されていることが望ましい。これらの巨大データベースを構成する多数のリレーションは分割されて，複数個のＳＤＭに分けられて蓄積されることになるが，この時必要となるシステム全体の記憶管理を行うものである。

  記憶管理の方式としては，汎用計算機，等で用いられている方式が有効であるが，仮想アドレス空間と実アドレス空間とのマッピングを高速に行う方式が望ましい。

- ホストプロセッサインタフェース

関係データベースマシンとホストプロセッサとの交信のための制御を行うものである。第1の処理は，ホストプロセッサを介して送られてくるユーザクイアリを受け取り，内部高速記憶に一時的に蓄積する。第2の処理は，クイアリの処理の結果，それを満たす結果リレーションをホストプロセッサを介してユーザに渡すことである。第3の処理は，実験機を構成するＳＤＭ，ＤＳＧ，ＤＳＰ，等の機能モジュールで発生した障害をホストプロセッサに通知する処理である。

- 障害処理

実験機ハードウェアで発生する障害の検出，診断，その結果に対する対策，等の処理を行う。また，対策に対する指示や障害に関する報告処理を行うために，ホストプロセッサとの交信処理も行う。

- ＤＤＰに対する制御

ＳＤＭに貯えられているデータベースに対するデイレクトリ管理，等を行うData Directory／Dictionary Processor（ＤＤＰ）に対する制御を行う。

ＳＶＣは以上述べたような処理を行うことが要求されるが，実験機としての性能はこのＳＶＣにも大きく依存すると考えられるため，高性能なミニコンピュータ，等をベースに構成することが望ましいと思われる。

さらに，実験機を能率良く実現するために，ホストプロセッサとして高性能な汎用ミニコンピュータが用いられることを考えると，ＳＶＣとしての機能をホストプロセッサ上で実現することが適当と考えられる。

(8) Data Directory／Dictionary Processor（ＤＤＰ）

ＳＤＭ中には多量のデータベースが蓄積され，時々刻々に更新されながら用いられる。このために，ＳＤＭ中に蓄積されているデータベースがどのような形で貯えられているかを管理することが必要になる。ＤＤＰはこの処理を専門に行うプロセッサであり，データベースに対するディレクトリを管理する，等の方式により，この機能を実現することができる。

従って，大量のディレクトリの作成，更新，検索，等を効率良く出来ることが要求されるため，すでに述べたＳＤＭを用いることにより実現できるものと考えられる。

(9) Math Processor（ＭＰ）

関係処理の1つとして要求される集計演算を行うものであり，2進データばかりでなく，浮動小数点データについても処理できることが望ましい。

ＭＰは専用のプロセッサを実現するか，これに合った市販の処理ユニット（例えば，イン

テル社のNumeric Data Processor 8087），等を用いて作ることが適当である。

f．機能モジュール間インタフェース

関係データベースマシン実験機を構成する機能モジュールは互いに緊密な関係を持って動作するために，機能モジュールの内部仕様とともに，モジュール間インタフェースを高性能に実現することが要求され，このインタフェースが実験機の性能に大きく依存する。このために，機能モジュール間インタフェースは単純であることと，高速であることが必要と考えられる。

- 制御信号インタフェース

関係代数演算子やリレーション記述を示す制御コマンドや演算子の実行開始信号，等の制御信号インタフェースはＳＶＣとＳＰＣ，ＳＧＣ，ＳＤＣ間，およびＳＰＣとＤＳＰ，ＳＧＣとＤＳＧ，ＳＤＣとＳＤＭ間，等で必要になる。これらは，高速なことが要求されるため，専用信号ラインで構成することが望ましい。

- データ信号インタフェース

ソースリレーションや結果リレーションを転送するためのインタフェースで，大量のデータストリームを連続的に送るものである。

これには，ＳＤＭとＤＳＧ間の結合ネットワーク（ＧＤＮ）とＤＳＧとＤＳＰ間の結合ネットワーク（ＰＧＮ）が主なものと考えられる。いずれの結合ネットワークも，複数の機能モジュールと複数の機能モジュール間で自由に結合される機能を備えるとともに，文字列データ，等に対する高速転送が必要になる。

従って，これらを満足する方式として，バス結合，マトリックス結合，トリー結合，ネットワーク結合，多段結合，等が考えられるが，多段スイッチング・ネットワークの方式が有望であると思われる。また，データの幅としては，少なくとも１バイト程度は必要であり，転送速度は数十Ｍバイト／秒程度は要求されるだろう。

g．ホストプロセッサ間インタフェース

関係データベースマシン実験機とホストプロセッサは，クイアリ転送，データベースの転送，等のためのインタフェースを準備することが必要である。

大容量のデータを転送するデータベース転送では，入出力インタフェースを用いて実現できるが，さらに高速化するために，例えば，ＤＭＡを用いる方式も考えられる。

クイアリ転送では，データ量はそれ程多くないと考えられ，その反面，高速性が要求されると考えられるので，バス結合，メモリ結合，等の専用インタフェースで実現できることが望まれる。また，割り込みや障害通知，等の状態信号は，高速性が要求されるので，プロセッサ間の専用ラインを設けることが必要と考えられる。

従って，このような点を考慮して，ホストプロセッサを選択することが望まれる。

次に，ホストプロセッサとの機能インタフェースについて検討する。

システム構成の検討においても論じたように，関係データベースマシン実験機の本体では関係演算処理を高速に行うことに集中し，他の処理はホストプロセッサに依頼することが望まれる。

従って，ユーザの高水準のクイアリはＳＶＣ機能をホストプロセッサ内で実現する時には，ホストプロセッサ上のソフトウェアにより，関係演算子列へ変換され，実験機が実行できる形でホストプロセッサから送ることになる。また，クイアリを満たすリレーションが生成されると，実験機からホストプロセッサへ送られ，ホストプロセッサ上の編集ソフトウェアによってユーザが見易い形に加工されて出力される方式が望ましい。

また，実験機で発生した障害情報，等についても，ホストプロセッサ側のソフトウェアにより分析する方式が得策と考えられる。

h．ホストプロセッサ上でのソフトウェアシステム

ホストプロセッサインタフェースで述べた機能をホストプロセッサにおいて実現するためには，図 6.3.1 1 に示すような構成のソフトウェアが必要と考えられる。ここでは，ホストプロセッサとして，ＳＶＣを含む構成の場合として，述べる。

- クイアリトランスレータ

  ユーザインタフェースとして提供するクイアリの仕様を決めるとともに，これを関係演算子列へ変換するトランスレータを作成することが要請される。

- 編集プログラム

  結果リレーションをユーザが見易い形に変形して出力するもので，ホストプロセッサが備えているエディタ機能を用いて実現することが望まれる。

- 障害解析プログラム

  実験機から送られてくる障害情報を，例えば，内部辞書をもとに分析し，分析結果を表示すると共に，指示に従って，以降の処理を制御するものである。

以上の他，ホストプロセッサに備えられていると考えられる入出力処理プログラム，各種ユーティリティが用いられる。

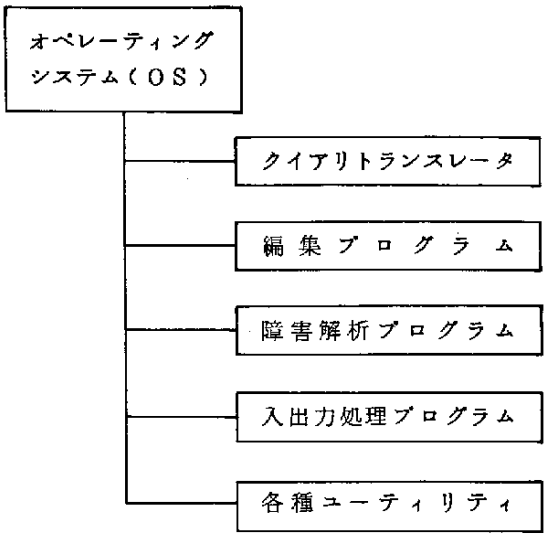


図6.3.1 1　ホストプロセッサ上のソフトウェア構成例

E. 開発計画

表6.3.1に開発スケジュール，表6.3.2に人員計画，表6.3.3に経費計画を示す。

表6.3.1　開発スケジュール

	57年	58年	59年
1) システム基本検討 <作業項目1～3>			
2) ハードウェア構成 <作業項目4.および6.3.2のシミュレーション>	基本・設計およびソフトウェア・シミュレーション	詳細設計	試作・結合・検査・シミュレーション実験・評価
3) 機能モジュール <作業項目 5 >	仕様検討	設計・試	作
4) 結合インタフェース <作業項目6. 7>	仕様検討	設計・試	作
5) ソフトウェアシステム <作業項目 8 >	基本設計	詳細設計・	作成

表6.3.2　人　員　計　画

	57年		58年		59年	
1) システム基本検討						
2) ハードウェア構成	研	2人	研	2人	研	2人
	技	1人	技	3人	技	2人
3) 機能モジュール	研	2人	研	5人	研	2人
	技	1人	技	10人	技	6人
4) 結合インタフェース	研	1人	研	2人	研	1人
	技	1人	技	4人	技	3人
5) ソフトウェアシステム	研	2人	研	2人	研	2人
	技	0人	技	6人	技	4人
合　計	研	7人	研	11人	研	7人
	技	3人	技	23人	技	15人

注）研：研究者
　　技：技術者　サポーティングスタッフ

表6.3.3　経　費　計　画

	57年	58年	59年
1) 計算機使用料			
シミュレーション	50h	50h	10h
ソフト/ファーム/ハード開発	50h	100h	50h

6.3.2　シミュレーション

A. 目　　標

プロトタイプデータベースマシンを開発するためには，各種の処理方式，構成法について，適用される環境下において評価を行い，それに適したものを選定することが重要である。

この選定に当り，開発を行った関係データベースマシン実験機を用いて各種のシミュレーショ

ン評価を行い，定量的な分析結果にもとづいて決定することが望ましい。

このために，以下の目標を達成することを念頭において，シミュレーションを行う。

- 処理速度に対する評価・分析
- データベース量に対する評価・分析
- 処理方式の分析
- ハードウェア構成の評価
- 機能モジュールの評価

B．研究開発のポイント

本研究開発を進める時の第1のポイントとして，プロトタイプデータベースマシンが適用されると考えられる知識情報処理の応用分野を忠実に反映する環境下でシミュレーションを行うことである。これには，クイアリの特性と，データベースの量や構成，について特に注意が必要である。

第2のポイントは，基礎的な処理方式や構成技法，等の主要な設計パラメータに対するシミュレーションを中心に作業を行うことが望ましい。

第3のポイントは，効率良いシミュレーションを行うことであり，これには，先ず，マクロなシミュレーションを行い，続いて，詳細なミクロなシミュレーションを行う形で進めることが賢明である。

C．作業内容

本シミュレーションを実施するために必要な作業項目の要点を次に述べる。

(1) 応用プログラム，データベースの準備

知識情報処理の応用の標準的なプログラムを準備するとともに，これに用いられるデータベースを作成する。また，クイアリ形式についても規定する。

(2) 評価データ収集のための準備

実験機が高速に動作するので，詳細な評価データを収集するために，高速のハードウェア，等が必要になる。このために，動作速度を分析し，要求を明確にし，出来る限り市販のハードウェアモニタなどのツールを使用する方向で検討することが望ましい。

(3) 測定パラメータの仕様検討

シミュレーションを効率良く行うために，測定されるパラメータを出来る限りしぼり，必要項目に集中する方針で測定パラメータの仕様を決める。

(4) 測定および整理・分析

準備した応用プログラムとデータベースを実験機に実装し，各種の方式，構成技法について，

特性データを測定する。そして，測定結果を分析することにより，方式，技法の評価・解析を行う。

D. 開発計画

6.3.1の開発計画で述べた通り，実験機設計・開発の一環として実施する。

—参考文献—

1) E. A. Ozkarahan, et. al. : RAP—An Associative Processor for Database Management, AFIPS NCC, pp379—387, Vol 144, 1975

2) S. W. Y. Su : CASSM—a Cellular System for Very Large Databases, Proc. Int. Conf. on VLDB, 1975

3) E. J. Oliver, et. al. : RELACS—A Relational Associative Computer System, 5th Workshop on Computer Architecture for Non—Numeric Processing, 1980

4) Y. Tanaka, et. al. : Pipelined Searching and Sorting Modules as Components of a Data Flow Data Base Computer, Proc. IFIP 80, 1980

5) 喜連川, 他：HashとSortによる関係代数マシン，電子通信学会電子計算機研究会，EC81—35. 1981

6) H. T. Kung, et. al. : Systolic (VLSI) Arrays for Relational Data Base Operation, Proc. ACM SIGMOD, 1980

7. 研究開発計画と体制

本章では，関係データベースマシン（以下ＲＤＢＭと略す）の研究開発の計画および体制についてまとめる。

7.1 研究開発計画の相互関連

ＲＤＢＭの研究開発は，ＲＤＢＭの理想像を追求する(1)ＲＤＢＭの方式研究，理想的なＲＤＢＭに不可欠な要素技術・要素モジュールを研究・開発・評価する(2)要素技術・要素モジュールの開発，そして，(2)で開発された技術等をもとにＲＤＢＭの開発を試みる(3)ＲＤＢＭ実験機の開発を３本の柱にして推進する。

ＲＤＢＭの前期研究開発計画の概要は表７－１のとおりである。表７－１の研究開発項目は，それぞれ，(1)の①が6.1.1項，(1)の②が6.1.2項，(2)の①が6.2.1項，(2)の②が6.2.2項，(2)の③が6.2.3項，(3)が6.3節と対応している。

また，表７－１中の"○　"記号は各研究開発項目間での技術導入および成果の反映の流れを示している。すなわち，５７年度にはＲＤＢＭ実験機の基本検討と設計を行い，関係代数演算処理モジュール，モジュール間結合ネットワーク，記憶階層システムは，それぞれ，５７年度末，58年度末，５９年度に試作を完了して実験機に導入される。また，実験機の開発結果がＲＤＢＭの方式および要素技術の研究開発に有効な指針を与えることをも示している。

7.2 研究開発体制

6章で示した研究を実行するのに必要な研究者と技術者との所要人数を表７－２に示す。また，その実行過程で必要となる計算機使用時間の推定値を表７－３に示す。

表７－２で示した人数は，いずれも必要最小限に近い数であり，サポート用人員を含めればもっと多くを必要としよう。データベースマシンは，研究が開始されてから何年か経つが，その必要とする機能の膨大さのため，部分的な実験は行われてはいるが，完全な形での実装実験は余り多くない。特に全体の制御ソフトウェアの部分はその実装に多大のマンパワーを必要とする。従って，ここで示した値はかなり控え目な値である。

表７－３では計算機使用時間として汎用大型機におけるＣＰＵ時間が示されているが，研究の中で行う機能ＬＳＩの試作用のＣＡＤ利用時間は含まれていない。

表7.1　関係データベースマシン前期開発計画

研究開発項目＼年度		57年度	58年度	59年度
(3) 関係データベースマシン実験機の開発		システム基本検討とハードウェア，ソフトウェアの基本設計と要素技術の仕様検討	詳細設計と試作と評価	
(2) 要素技術，要素モジュールの開発	① 関係代数演算処理モジュール	ディスクリートタイプ試作	改良およびVLSI設計	VLSI開発
	② モジュール間結合ネットワーク	設計および開発	改良およびVLSI設計	VLSI開発
	③ 記憶階層システム	基本設計	制御ソフトウェア設計と論理設計	試作と評価
(1) 関係データベースマシンの方式研究	① アーキテクチャの研究	基本検討	第1次アーキテクチャ	アーキテクチャの評価とプロトタイプアーキテクチャ概様設定
	② ソフトウェアの研究	基礎研究と機能設計	基礎研究と機能・仕様設計	仕様設計

（注1）　○～→　は技術導入および成果の反映を意味する。

表7.2　研　究　開　発　人　員

研究開発項目＼年度			５７年度	５８年度	５９年度
(3)　関係データベースマシン実験機の開発		研	7人	11人	7人
		技	3人	23人	15人
(2) 要素技術，要素モジュールの開発	①　関係代数演算処理モジュール	研	12	12	12
		技	20	20	20
	②　モジュール間結合ネットワーク	研	4	3	3
		技	8	5	5
	③　記憶階層システム	研	6	10	10
		技	4	12	14
(1) 関係データベースマシンの方式研究	①　アーキテクチャの研究	研	7	7	7
		技	9	9	9
	②　ソフトウェアの研究	研	6	10	8
		技	2	8	8
合　計		研	42人	53人	47人
		技	46人	77人	71人

（注１）　研は研究者，技は技術者の略記である。

表7.3　計　算　機　使　用　時　間

研究開発項目＼年度			５７年度	５８年度	５９年度
(3)　関係データベースマシン実験機の開発		CPU	100	150	60
		端　末	2,000	3,000	1,200
(2) 要素技術，要素モジュールの開発	①　関係代数演算処理モジュール	CPU	20	20	20
		端　末	400	400	400
	②　モジュール間結合ネットワーク	CPU	5	5	5
		端　末	100	100	100
	③　記憶階層システム	CPU	50	80	50
		端　末	1,000	1,600	1,000
(1) 関係データベースマシンの方式研究	①　アーキテクチャの研究	CPU	10	50	50
		端　末	200	1,000	1,000
	②　ソフトウェアの研究	CPU	100	100	50
		端　末	2,000	2,000	1,000
合　計		CPU	285	405	235
		端　末	5,700	8,100	4,700

（注１）　単位は，汎用大型計算機を使用した場合の１時間。

昭和57年2月発行

発行所　財団法人 日本情報処理開発協会
社団法人 日本電子工業振興協会
東京都港区芝公園3—5—8
機械振興会館内
TEL 03(434)8211(大代表)

印刷所　有限会社 サンコピー印刷部
東京都大田区西蒲田4—23—7
TEL 03(754)7076