資　料

ハイエンドコンピューティング技術に関する調査研究Ⅳ

平成 15 年 3 月

財団法人　日本情報処理開発協会
先 端 情 報 技 術 研 究 所

KEIRIN

この事業は、競輪の補助金を受けて実施したものです。

ハイエンドコンピューティング技術調査ワーキンググループ

主査	山口　喜教	筑波大学 電子・情報工学系（学術情報処理センター）　教授
委員	天野　英晴	慶應義塾大学 理工学部情報工学科　教授
委員	笠原　博徳	早稲田大学 理工学部　電気電子情報工学科　教授
委員	久門　耕一	(株)富士通研究所 IT コア研究所　グリッド＆バイオ研究部　部長
委員	妹尾　義樹	日本電気(株) インターネットシステム研究所　研究部長
委員	関口　智嗣	独立行政法人　産業技術総合研究所 グリッド研究センター　センター長
委員	佐藤　裕幸	三菱電機(株) 情報技術総合研究所 電子システム技術部 チームリーダ
委員	佐藤　真琴	(株)日立製作所 システム開発研究所　主任研究員
委員	近山　隆	東京大学 新領域創成科学研究科　教授
委員	中島　浩	豊橋技術科学大学 情報工学系　教授
委員	福井　義成	理化学研究所 情報基盤研究部　情報環境室　プロジェクトリーダ
委員	横川　三津夫	独立行政法人　産業技術総合研究所 グリッド研究センター　副センター長

幹事	石井　英志	(財)日本情報処理開発協会 先端情報技術研究所 技術調査部 主任研究員

ハイエンドコンピューティング技術に関する調査研究 Ⅳ

目次

第 1 章 まえがき

1.1 調査の活動方針（山口主査）……1

1.2 調査報告の概要 ……3

1.2.1 アーキテクチャ＆新計算モデル ……3

1.2.2 基本ソフトウェア＆ミドルウェア ……5

1.2.3 応用システム＆応用分野 ……7

1.3 その他の活動 ……9

第 2 章 米国のハイエンドコンピューティング研究開発動向

2.1 概況 ……11

2.2 FY2003 Blue Book にみる政府支援の IT 研究開発 ……12

2.2.1 概要 ……12

2.2.2 NITRD 予算規模 ……13

2.2.3 ハイエンドコンピューティング研究開発の内容 ……15

2.3 ハイエンドコンピューティング分野の新しい動向 ……20

2.3.1 SC2002 に見る動向 ……20

2.3.2 超並列コンピューティング ……23

2.3.3 グリッド・コンピューティング ……25

2.3.4 将来のコンピューティングに関する研究 ……27

第 3 章 ハイエンドコンピューティング研究開発の動向

3.1 概要 ……31

3.1.1 調査方針 ……31

3.1.2 外部講師 ……31

3.2 アーキテクチャ＆新計算モデル

3.2.1 粗粒度 Reconfigurable Device の挑戦（天野委員） ……32

3.2.2 CPU の高速化とアプリケーション性能の高速化（久門委員） ……42

3.2.3 高性能プロセッサの高速シミュレータ（中島委員） ……48

3.2.4　大規模アプリケーションと
大規模計算に望ましいプロセッサ（福井委員）……56
3.2.5　ソフトウェア可制御オンチップメモリを用いた
高性能・低消費電力プロセッサ（中村講師）……70
3.3　基本ソフトウェア＆ミドルウェア
3.3.1　アドバンスト並列化コンパイラプロジェクトにおける
マルチグレイン並列処理（笠原委員）……87
3.3.2　HPF の 10 年と並列プログラミングインタフェース（妹尾委員）……103
3.3.3　Software Design and Productivity (SDP) Coordination Group
が目指すもの（近山委員）……120
3.4　応用システム＆応用分野
3.4.1　センサ・データ処理の高速化（佐藤(裕)委員）……130
3.4.2　グリッド技術の概要と動向（関口委員）……143
3.4.3　非圧縮性一様等方性乱流の大規模直接数値シミュレーション（横川委員）153
3.4.4　Web サービスの技術動向（藤田講師）……160

第 4 章　あとがき（山口主査）……173

付表

付表 1　ハイパフォーマンス・コンピュータ世界の Top 20 ……175
付表 2　クラスタコンピュータ世界の Top 20 ……176
付表 3　日本製ハイパフォーマンス・コンピュータ Top 20 ……177

平成 14 年度ワーキンググループ活動記録……179

第 1 章 まえがき

1.1 調査の活動方針

本報告書は、先端情報技術研究所(AITEC)内に設置された「ハイエンドコンピューティング技術調査(HECC)ワーキンググループ」での調査や議論に基づき、各委員の報告を中心にまとめたものである。このワーキンググループは、平成 8 年に設定された「ペタフロップスマシン技術調査ワーキンググループ」を 3 年間続けた後に、先端的なコンピュータ技術の調査をより広範囲な視点で行うことを目的として設置されたものである。その委員は大学、メーカ等で活躍中の研究者でアーキテクチャ、ソフトウェア、アプリケーションの各分野において実際に研究や開発に携わっている方々から構成されている。このワーキンググループでは､ハイエンドアーキテクチャ､ハイエンドハードウエアコンポーネンツ、アルゴリズム研究を含む基礎研究、ソフトウェアおよびハイエンドアプリケーションなどを対象として、調査や議論を行ってきた。

本年度は、HECC ワーキンググループとしては 4 年目の調査であり、本報告書が当ワーキンググループととしての最後の報告書である。

今年度も、昨年度の調査と同様に、HECC 領域の開発動向を重要な中心課題と認識しつつも、これのみにとらわれず対象を拡大し、コンテンツやユーザインタフェースの実現基盤としてのプラットフォーム技術全般を視野に入れて、調査・検討を行った。調査は、各委員の専門およびその周辺分野において、今後、研究的に急速に発展したり、あるいは市場にインパクトを与えそうな分野やテーマの抽出と、それらの技術に関する、先端研究の調査・評価を行うことを主眼に、今後重要になると思われる領域にはどのようなものがあるかを抽出するという点に力点が置かれている。

また、昨年と同様に、一般的に Blue Book と呼ばれている「IT 研究開発(NITRD)に関する大統領予算教書補足資料」の 2003 年度版 を中心に、米国におけるハイエンドコンピューティング研究開発動向について調査をおこなった。これについては、本報告書の第 2 章にまとめられている。

HECC での 4 年間の報告書は、各委員の主体的な調査によるところが大きいため、調査の焦点が絞りきれていないという批判はあるかも知れないが、それとは逆に、情報処理技術やハイエンドコンピュータの分野において、今後重要になると思われる領域にはどのようなものがあるかを抽出して紹介するという点においては、一定の役割を果たすことができたのではないかと考えている。たとえば、最近注目されているグリッドと呼ばれるグローバルコンピューティング技術などについては、近未来の基盤技術としてインパクトを与

える可能性がある技術として数年前から取り上げてきたことなどがその良い例である。

本年度は、最終年度ということもあり、ワーキンググループとしては 3 回の会合を持ったのみであったが、限られた会合の中で、2 人の外部講師を招いて話を伺うことができた。1 人は、東京大学の中村宏氏であり、省エネルギーを指向した高性能プロセッサに関して講演をしていただいた。もう 1 人は、日本電気の藤田悟氏であり、「Web サービスの技術動向」と題して、21 世紀にインターネットを介して広く展開されると考えられている Web サービスに関して、その技術的な課題と標準化活動の状況などに関して講演をしていただいた。これらの講演を基に、中村氏と藤田氏には講演の概要を、本報告書の中にまとめていただいた。ここで、あらためて感謝したい。

本ワーキンググループの各委員は、情報処理の分野において最先端の研究や開発に従事している方々であり、情報処理分野における重要と思われる分野について各委員の独自の判断で調査を行い、本報告書に記述していただいていることを申し添えておきたい。

本報告書が、わが国のハイエンドコンピューティングをはじめとする先端的な情報処理分野においてその技術開発や技術政策の一助になれば幸いである。

（山口　喜教　主査）

1.2 調査報告の概要

本節では、第 3 章に記載する各委員および外部講師による調査報告の概要を示す。

1.2.1 アーキテクチャ＆新計算モデル

(1) 粗粒度 Reconfigurable Device の挑戦

天野　英晴　委員

Reconfigurable System は FPGA/PLD などの書き換え可能なデバイスを用いて問題アルゴリズムを直接ハードウェアで実行するシステムである。最近、実用化が進んでいるものの、構成情報の書き換えに時間を要するため、汎用の書き換えデバイスを用いた場合の限界が見え始めた。そこで、最近、高速に構成を変更可能な動的適応型デバイスが登場した。NEC の DRP、IP Flex の DN Matrix、Quicksilver の ACM などがこの代表であり、高速な書き換え機能に加えて、粗粒度のアーキテクチャ、高位言語によるプログラムなどの能力を持つ。これらのデバイスの動向をまとめると共に、特に NEC の DRP を用いた動的適応型ハードウェアについて紹介する。

(2) CPU の高速化とアプリケーション性能の高速化

久門　耕一　委員

CPU の周波数、トランジスタ数はムーアの法則にほぼ従い向上してきた。しかし、計算機の性能という意味では必ずしも上記の法則にしたがった向上が得られているとは言えない。良く知られた原因として、メモリアクセス時間がクロック周波数の向上に見合うだけ短縮していないことが挙げられている。しかし、これ以外にも、高周波数を達成するための長大なパイプライン構造が原因となった非通常処理時のパイプラインブレークによるペナルティの増加が挙げられる。この状況は、HPC アプリケーションではなくビジネス向のサーバ処理に頻繁に出現するため、ビジネスサーバの性能向上に大きいな障害となる。

このレポートでは、Linux オペレーティングシステムを使ったサーバシステムにおける、計算機アーキテクチャの持つ役割について述べる。

(3) 高性能プロセッサの高速シミュレータ

中島　浩　委員

マイクロプロセッサの高度化・複雑化に伴い、複数のパイプラインの動作など、ハードウェア機構の挙動をクロックレベルで忠実に再現する cycle accurate なシミュレーションには、非常に長い時間を要するようになっている。たとえば最も広く用いられている

SimpleScalar では、実機との実行時間比（slow down）が 5,000～10,000 と極めて大きく、実用的なアプリケーションはもちろん、SPEC などのベンチマークの実行でさえ膨大な時間が必要である。そのため、アーキテクチャレベルのシミュレータの高速化については様々な研究がなされている。

今回の報告では Wisconsin 大学の FastSim や筆者らが開発中の高速シミュレータを中心に、高速性と正確性の両立をめざしたシミュレータの研究動向について概観する。

(4) 大規模アプリケーションと大規模計算に望ましいプロセッサ

― 計算機アーキテクチャに対する素朴な疑問 ―

福井　義成　委員

計算機の歴史、アプリケーションの開発・保守、高速化などの立場から見た現在の計算機への素朴な疑問について述べる。計算機が開発されて以来、計算機の歴史は高速化の歴史でもあった。計算機を高速化する手段はいくつか考えられる。(1)素子の高速化、(2)メモリの大容量化、(3)キャッシュメモリ、(4)命令バッファ、(5)メモリのインタリーブ、(6)ループを効果的に行う命令、(7)ベクトル命令、(8)パイプライン化等があったが、この中で、すべての問題（プログラム）に満遍なく効果があるのは(1)素子の高速化だけである。その他の項目は、何らかの「仮定」を満足した場合のみに高速化の効果があるものばかりである。

このようなことを考慮して、1 つの計算機のアイデアを考えて見た。計算機設計の素人の発想であるので、具体的なことはともかくとして、1 つの考えかたとして見ていただきたい。対象は科学・技術計算のハイパフォーマンスコンピューティング(HPC)を対象に考える。

プロセッサに比較して、メモリが遅いことは残念ながら受け入れることとする。従ってなんらかの形での階層的なメモリ構造は受け入れざるを得ないと考える。しかし、キャッシュ機構は投機的で、HPC 分野では障害が多い。そこで、場所により速度に差のあるメモリ構成を考え、番地が小さいところほど速いメモリを配置する。半導体技術の進歩したときに、速度差のあるメモリ間の境界が変わるだけなので、そのことを考慮したプログラムにすることで、その恩恵が自然な形でユーザに得られるアーキテクチャを狙う。ベクトル機構も従来のような重厚長大なものではなく、数%のゲートの増加で、150%の性能向上が得られれば良いとする発想である。利用可能なゲートを如何に使うかという視点が重要であると考える。

(5) ソフトウェア可制御オンチップメモリを用いた高性能・低消費電力プロセッサ

中村　宏　講師

近年のマイクロプロセッサにおいて、キャッシュメモリと主記憶からなるメモリシステムは性能上のボトルネックであると同時に消費電力が大きい。そこで、高性能かつ低消費電力なマイクロプロセッサを目的として、ソフトウェアから制御可能なオンチップメモリを採用するアーキテクチャ SCIMA を提案している。このオンチップメモリは、通常のキャッシュメモリと同じハードウェア機構で実現されており、オンチップメモリとキャッシュメモリは再構成可能である。従って、オンチップメモリとキャッシュメモリの総容量は実装時に決定されるものの、処理の特徴に応じて両者の容量を変更することが可能である。

SCIMA のオンチップメモリは、ソフトウェアによる明示的なデータのアロケーションとリプレースメントが可能であり、データの再利用性を最大限に活用することができる。このため、性能低下と消費電力増大を招くチップ間のトラフィックを低減することができる。また、SCIMA のオンチップメモリは、キャッシュと違い与えられたアドレスにより格納場所が一意に決まるため、全てのウェイをアクセスしたりタグ部をアクセスする必要がなく、低消費電力化を実現できる。NAS PARALLEL ベンチマーク、および計算物理学の実アプリケーションを用いて、従来のメモリ階層を持つプロセッサに対する SCIMA が性能と消費電力の優位性を定量的に評価した。その結果、SCIMA は性能と消費電力の両面で優れていることがわかった。

1.2.2 基本ソフトウェア＆ミドルウェア

(1) アドバンスト並列化コンパイラプロジェクトにおけるマルチグレイン並列処理

笠原　博徳　委員

ここでは、経済産業省ミレニアムプロジェクト官民共同研究開発プロジェクトとして、新規産業創出型産業科学技術研究開発制度（産技制度）に基づき、新エネルギー・産業技術総合開発機構(NEDO)からアドバンスト並列化コンパイラ技術研究研究体が委託を受け、2000 年 9 月より 3 年度計画で活動を行っているアドバンスト並列化コンパイラ技術研究開発プロジェクトにおけるマルチグレイン並列処理について述べる。

本プロジェクトの目標は、従来 HPC(High Performance Computer)の代名詞ともなっているマルチプロセッサシステム（複数の CPU を有機的に利用し高性能を達成するコンピュータで、今後マイクロプロセッサ、各種携帯情報機器、ホームサーバ等幅広い情報機器で使われていくと考えられている）の実行効率を 2 倍程度に高め、価格性能比及び使いやすさを向上させることにより、コンピュータ及び各種 IT 機器の国際競争力を強化しようというものである。

(2) HPF の 10 年と並列プログラミングインタフェース

妹尾　義樹　委員

分散メモリ並列システム向けの並列プログラミング言語 HPF（High Performance Fortran）について、その最初の仕様が定められてから、この春でちょうど 10 年になる。昨年 11 月に米国バルチモアで行われた SC2002 国際会議において、地球シミュレータ上で HPF を用いたプラズマシミュレーション並列化の成果が Gordon Bell Award の言語賞を受賞した。陽解法の TVD 差分スキームを用いたプログラムで、14.9 テラフロップスを達成したことが認められた。ようやくここまで来たか、と感慨深いものがあるが、HPF の本当の実用化という意味では、まだまだ課題が残っているのも事実である。

本報告では、この 10 年間の HPF の歩みを総括するとともに、最近の並列プログラミングインタフェースの話題と、今後の方向について、私見を述べる。特に、HPF と MPI の中庸を狙ったといえる、Co-Array Fortran や GA (Global Arrays)を紹介し、MPI や OpenMP を含めた種々のプログラミングインタフェースの比較を行い、今後の方向性について考えてみたい。

(3) Software Design and Productivity (SDP) Coordination Group が目指すもの

近山　隆　委員

米国の政府機関である Interagency Working Group for Information Technology Research and Development (ITRD) 傘下の Software Design and Productivity(SDP) Coordinating Group の活動の方向性を定めるため、2001 年 8 月 18 日～19 日に Arlington において行なわれた“Planning Workshop on New Visions for Software Design and Productivity”について紹介する。

このワークショップでは、

・ソフトウェアの将来とソフトウェア研究

・ソフトウェア開発の新パラダイム

・実世界のためのソフトウェア

・ネットワーク中心の分散ソフトウェア

の 4 領域について、様々な研究テーマが提案された。どの領域についても、提案の中心は抽象化を武器に複雑性・規模・不均質等の実世界の問題に挑戦する課題で、従来乖離していたソフトウェアの理論と実践を止揚し、新たな枠組を構築しようとするものである。

1.2.3 応用システム＆応用分野

(1) センサ･データ処理の高速化

佐藤　裕幸　委員

レーダなどのセンサからのデータ処理と言えば、従来は、DSP（Digital Signal Processor）を用いた処理や FFT（Fast Fourier Transform）などの信号処理が中心であった。すなわち、信号レベルでのデータをいかに高速に処理するかが課題であった。一方近年では、信号レベルではなく、より抽象度の高いデータを処理することが多くなってきている。例えば、観測された信号データを位置情報として扱い、目標追尾を行うことが挙げられる。目標追尾とは、レーダ等のセンサの観測結果から目標の航跡を抽出し、位置、速度等の運動諸元を推定する問題である。目標追尾の抱える課題の 1 つに、追尾目標の近傍に他の目標やクラッタ等の不要信号が存在する高密度環境下での追尾性能の確保が挙げられる。高密度環境下では、センサから得られた観測値のいずれが各追尾目標のものであるかを判定する相関処理が重要となる。この相関処理は、いわゆる組み合わせ問題であり、従来の情報処理の分野で用いられてきた手法が適用でき、それらを用いて高速化・並列化が試みられている。

(2) グリッド技術の概要と動向

関口　智嗣　委員

昨年あたりから、グリッド・コンピューティングという言葉が、科学技術専門メディアのみならず、一般のメディア上にも頻繁に登場するようになった。大学・公的研究機関などの大規模な計算機システムを対象としたものばかりでなく、一般家庭のパソコンを対象としたグリッド技術が構築され、また、科学技術計算の分野で始まったグリッドの波が、徐々にビジネスの世界にも押し寄せてきている。すなわち、従来から科学技術における応用としてバイオインフォマティクス、創薬デザイン、ライフサイエンス、高エネルギー物理や天文観測などの大規模科学実験などが知られていたが、今後はWebサービス、ユーティリティ提供、アミューズメントなどが次のターゲットになると考えられている。

本報告では、こうした大きな構想とは別に、現実の問題としてグリッドをどうすれば毎日の研究生活に使えるようになるのか、そうした日々のグリッドを作るためには何が必要なのかを念頭に置きながら、技術の現状を整理し、今後のグリッド技術動向全般について概観する。

(3) 非圧縮性一様等方性乱流の大規模直接数値シミュレーション

横川　三津夫　委員

近年のスーパーコンピュータの発展により、ナビエ・ストークス方程式の大規模な直接数値シミュレーション（DNS）が可能となってきた。しかし、乱流現象の解明とそのモデル化のためには、さらに大規模な DNS を行う必要がある。ピーク性能 40Tflops の地球シミュレータのハードウェアの性能評価を行うと共に、超大規模な乱流 DNS を実現することを目指し、地球シミュレータ用に最適化した DNS コードを開発した。このコードにより、512 ノードを用いた格子点数 2048^3 の DNS において 16.4Tflops の超高速計算と、従来の DNS と比較して桁違いに大規模な格子点数 4096^3 の DNS を実現することができ、SC2002 においてゴードンベル賞特別賞を受賞した。大規模 DNS によって得られた乱流場が、乱流の統計力学の構築など今後の乱流研究において大きな成果が期待される。

(4) Web サービスの技術動向

藤田　悟　講師

近年、サービス指向アーキテクチャという言葉の下、インターネットを通した疎結合のシステム統合技術である Web サービスが注目を集めてきた。これまでのソフトウェアコンポーネントを束ねたシステム統合技術に比べると、サービスという、より抽象度の高いコンポーネントのレベルの部品を統合することと、インターネットのような制限の多いネットワークの接続環境を前提にしていることが特徴的である。基盤となる標準化技術としては、XML(データ表現)、SOAP(メッセージング)、WSDL(インタフェース定義)、UDDI(サービスディレクトリ)があり、それぞれ、現在も改訂が進んでいる。しかし、これらの基盤技術だけでは、実ビジネスで利用するシステムとしては課題も多い。そこで、セキュリティ、信頼性、相互接続性などについても、様々な技術開発と標準化活動が並行して進められている状況にある。本報告では、Web サービスを取り巻く技術開発の動向について紹介し、今後への期待について述べる。

1.3 その他の活動

本ワーキンググループでは、平成 14 年度の活動として、委員、外部講師による調査以外に、当研究所の石井英志主任研究員による米国のハイエンドコンピューティング研究開発に関する調査を行った。主な参考資料としては、2002 年 8 月に公開された米国の FY2003 Blue Book と、2002 年 11 月に米国メリーランド州ボルチモアで開催された、ハイエンドコンピューティングに関する世界最大の国際会議である SC2002 に参加し、入手した情報をベースとした。

この調査結果は、第 2 章「米国のハイエンドコンピューティング研究開発動向」に記載した。

第 2 章　米国のハイエンドコンピューティング研究開発動向

2.1 概況

2002 年は、世界経済において、当初期待されていた景気の回復は見られなかった。米国では 2001 年末のエンロン社に続き、6 月には業界第 2 位の長距離通信企業であったワールドコム社における巨額の粉飾決算が発覚し、経営破綻に陥るなど、経営危機に見舞われる企業が相次いだ。NASDAQ 指数は 2002 年 1 月に一時 2,000 ポイント台まで回復したが、その後下降を続け、2003 年 2 月時点では 1,300 ポイント付近を低迷している。また、2001 年 9 月 11 日の同時多発テロ発生以降の不穏な中東アジア情勢や、米国によるイラク攻撃の可能性の増大（2003 年 2 月末現在）など、不安定な国際政治の動向が世界経済へ与える影響も懸念されている。こういった状況から、2003 年以降の景気回復の見通しは、依然として不透明なものとなっている。

このように、現在の経済の低迷は世界規模となっているためきわめて深刻なものであり、特に米国において、近年の経済成長を牽引してきた IT 産業の不振がその大きな要因であることは、疑いようのない事実であろう。しかし、この逆風下にあっても、米国政府の IT 研究開発投資への意欲はいささかも衰えを見せていないようである。政府の IT 研究開発の最重点課題が、「テロの脅威に対抗するための研究開発」に移ったとしても、その根底にあるのは、国を挙げて情報技術分野（およびその波及分野）で世界をリードするという従来からの強固な意志であることに変わりはない。

以下、本章ではハイエンドコンピューティングの米国における最新動向を中心に概観する。主な情報源としては 2002 年 11 月に開催された SC2002 コンファレンス、通称 Blue Book と呼ばれている「IT 研究開発(NITRD)に関する大統領予算教書補足資料」の 2003 年度版（FY[Fiscal Year]2003 Blue Book)、および米国政府関係機関サイトなどの Web 上で公開されている情報を元にしている。

FY2003 Blue Book に関しては、過去の分も含めて下記サイトに全文が掲載されている。

英語原本：http://www.itrd.gov/pubs/bb.html

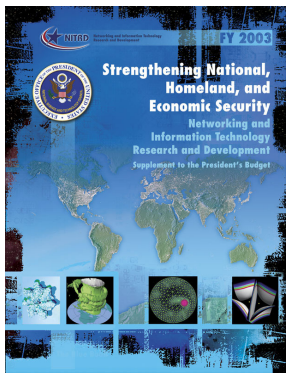
日本語訳：http://www.icot.or.jp/FTS/Ronbun/Ronbun-J.html

米国のハイエンドコンピューティング開発の背景、経緯は過去 6 期に渡る当研究所の報告書（ペタフロップスマシン技術に関する調査研究、同Ⅱ、同Ⅲ、およびハイエンドコンピューティング技術に関する調査研究Ⅰ、同Ⅱ、同Ⅲ）にまとめられているので、これらを参照いただきたい。上記の報告書はいずれも当研究所のホームページ(http://www.icot.or.jp/)から参照が可能である。

2.2　FY2003 Blue Book にみる政府支援の IT 研究開発

2.2.1 概要

2002 年 8 月に公開された FY2003 Blue Book：”Strengthening National, Homeland, and Economic Security”は、同時多発テロ以降の米国の立場を色濃く反映したものになっている。ここでは、政府支援による IT 研究開発の最重要目的は「米国の国家、国土および経済の安全保障の強化」のためであり、巻頭の 10 ページにわたって、NITRD 計画により開発された先進技術が国家、国民および経済の安全の確保にいかに貢献しているかが強調されている。



FY2003 Blue Book

ただし、内容を読む限り、その研究内容は従来からそれほど変化があったようには見えない。また、NITRD 計画の推進体制は図 2.1 に示す通りで、昨年度からの変更はごくわずかである。同時多発テロ以降、軍事、安全保障のための予算が増加しているが、従来からの IT 研究開発に対する影響は少ないようである。

NITRD計画への参加省庁は下記の12省庁である。

1) National Science Foundation (NSF)
2) Defense Advanced Research Projects Agency (DARPA)
3) National Institutes of Health (NIH)
4) National Aeronautics and Space Administration (NASA)
5) Department of Energy, Office of Science (DOE/SC)
6) National Security Agency (NSA)
7) National Institute of Standards and Technology (NIST)
8) National Oceanic and Atmospheric Administration (NOAA)
9) Agency for Healthcare Research and Quality (AHRQ)
10) Department of Defense, Office of the Director, Defense Research and Engineering (ODDR&E)
11) Environmental Protection Agency (EPA)
12) Department of Energy, National Nuclear Security Administration (DOE/NNSA)

参加省庁に関しては2002年度から若干の変更があるが、本質的な変更ではない(下記)。

DOE/OS：Department of Energy Office of Science

→ DOE/SC:：Department of Energy Office of Science　　（略称変更のみ）

ODUSD(S&T)：Office of the Deputy Under Secretary of Defense for Science and Technology

→ ODDR&E：Office of the Director, Defense Research and Engineering

（DoD 内の組織変更のみと思われる）

また、LSN Coordinating Group 内のチームが再編成され、Middleware and Grid Infrastructure Coordination(MAGIC)が追加された（2.2.3 節(3)参照）。

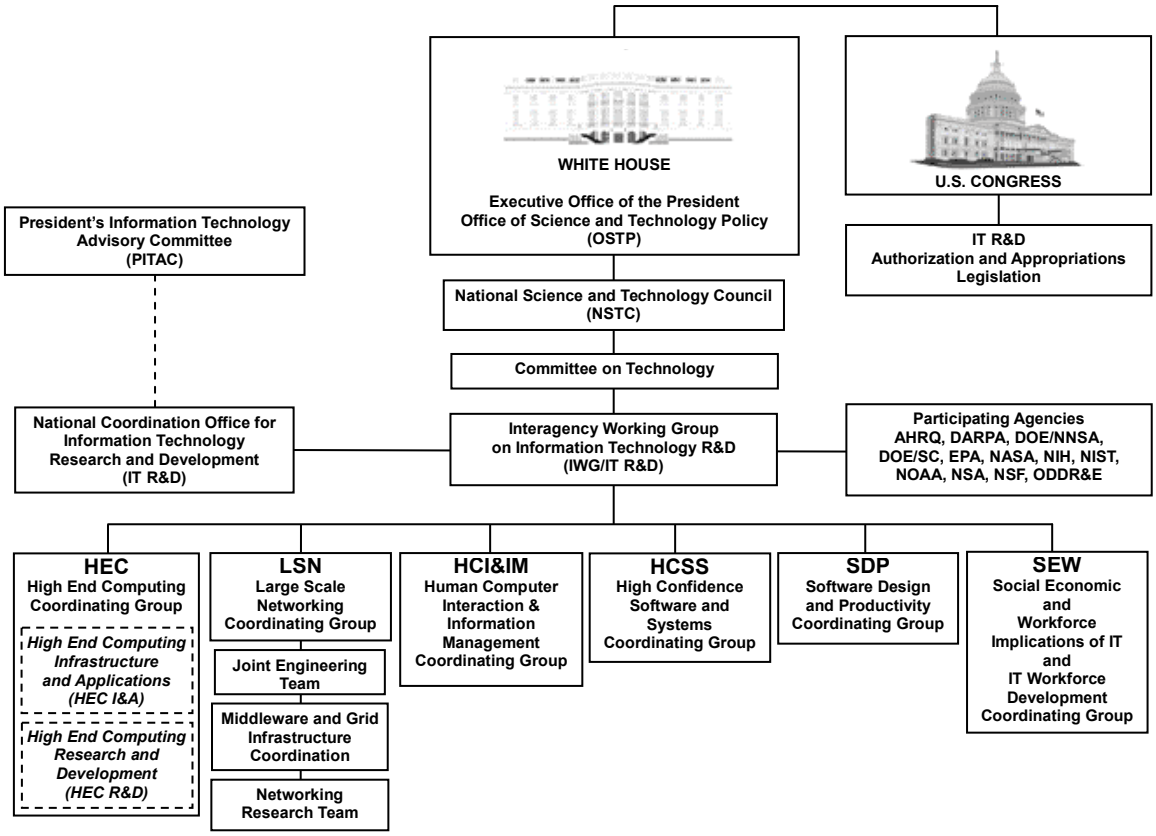


図 2.1　NITRD 推進体制

2.2.2 NITRD 予算規模

FY2003 Blue Book に掲載されている、NITRD 関連の省庁および PCA ごとの 2003 年度予算要求額を表 2.1 に示す。ハイエンドコンピューティング関係(HEC)予算は全体の半分近く(44.8%)を占めており、当初より NITRD 計画の中心的存在となっている。

表 2.1 NITRD 計画 FY2003 予算要求額（単位：100 万ドル）

省庁	HECI&A	HECR&D	HCI&IM	LSN	SDP	HCSS	SEW	合計		比率
NSF	215.2	68.3	132.2	102.4	45.8	50.1	64.8	679		35.9%
NIH	88.2	8.9	90.8	117.5	5.8	3.7	11.8	327		17.3%
DARPA	16.8	81.9	35.5	29.2	60			223		11.8%
NASA	68.4	26	23.8	4.5	40	43.3	7	213		11.3%
DOE/SC	98.5	39.3	16.4	28.1			3.5	186		9.8%
NSA		31.9		1.3		28.1		61		3.2%
NIST	3.5		3.2	6.2	7.5	2		22		1.2%
NOAA	13.3	1.8	0.5	2.8	1.5			20		1.1%
ODDR&E		1.8	1.8	6.3	1.9	1		13		
AHRQ			5	4				9		0.5%
EPA	1.8							2		0.1%
小計	505.7	259.9	309.2	302.3	162.5	128.2	87.1	1,755		92.9%
DOE/NNSA	41.4	39.5		14.7	34.2		4.3	134		7.1%
合計	547.1	299.4	309.2	317	196.7	128.2	91.4	1,889		100.0%
比率	29.0%	15.8%	16.4%	16.8%	10.4%	6.8%	4.8%	100.0%		

また、FY2004（2003 年 10 月～2004 年 9 月）大統領予算教書における NITRD 関連予算要求額が 2003 年 2 月 3 日に OSTP のサイトに掲載された。これを表 2.2 に示す[1]。
（http://www.ostp.gov/html/budget/2004/OSTP%20NITRD%201-pager%20(OMB).pdf）

表 2.2 NITRD 計画 FY2004 予算概要（単位：100 万ドル）

省庁	2002 実績	2003 要求	2004 要求	増加率 '03 to '04
National Science Foundation	662	678	724	7%
Defense	439	442	461	4%
Health and Human Services	347	374	441	18%
Energy	306	310	317	2%
NASA	181	213	195	-8%
Commerce	36	38	39	3%
Environmental Protection Agency	2	2	2	0%
合計	**1,973**	**2,057**	**2,179**	**6%**

表 2.2 を見ると、NITRD 計画の予算は FY2004 予算案においても全体的には順調な伸びを示しており、あいかわらず Health and Human Services（バイオ、医療関係）の大きな伸びが目立っている。その一方で NASA 関係の予算が 8%削減されているのが目に付く（た

[1] この要求額は大統領から議会に提出された予算案にもとづくものであり、議会での審議により大きく変わる可能性を持っている。例えば表 2.2 における FY2003 の予算額は表 2.1 の Blue Book 記載額からかなり増加していることがわかる。

だし、NASA の FY2004 予算は、2003 年 2 月 1 日のスペースシャトル、コロンビア号事故の影響で大きく変わる可能性あり）。

本資料では 2004 年度における NITRD 計画の重点項目として以下を挙げている。

- ネットワークの「信用」（セキュリティ、信頼性、およびプライバシー）
- 高信頼(high-assurance)ソフトウェアおよびシステム
- マイクロ、組み込み型センサ技術
- ハイエンドコンピューティング・プラットフォームのコスト、サイズ、および消費電力を低減するための革命的アーキテクチャ
- 情報技術による社会的、経済的インパクト

2.2.3 ハイエンドコンピューティング研究開発の内容

以下に、NITRD 計画における主要なハイエンドコンピューティングおよび高速ネットワーキング関係の研究開発の内容を、FY2003 Blue Book にもとづいて記載する。これらの内容は NITRD 計画の 7 つの領域(Program Component Area：PCA)のうち、HEC I&A、HEC R&D および LSN に相当している。

(1) ハイエンドコンピューティング

－複雑さのフロンティアを探求する新技術－

主要な研究課題

- 高性能コンピュータ・アーキテクチャー科学的発見と国家安全保障のための最大の能力をもたらすために、ハードウェア設計とアーキテクチャ、システム・ソフトウェア、科学的応用などの間の複雑さのトレードオフを理解し、それを克服する
- 次世代スーパーコンピュータ・プラットフォームを実現するための革命的手法－新しいアーキテクチャ、量子及び生物分子コンポーネント、ハイブリッド技術、チップ上の再構成可能なシステム、processor-in-memory（PIM）技術を含む革新的なプロセッシング及びストレージ概念
- システムソフトウェアー先進的なプログラミング環境、コンパイラ、ライブラリ、ミドルウェア、及びパフォーマンス・エンジニアリング技術
- 科学計算とシミュレーションのための新しいハイエンド・アルゴリズムとコード；分散型テラスケール・コンピューティング環境のための、統合化・最適化されたソフトウェア基盤

2003 年度における各機関の代表的活動

政府機関	活動の内容
NSF	テラスケール基盤；異機種分散型ハイエンドシステムのためのシステム・ソフトウェア、ミドルウェア、ソフトウェア環境、ライブラリ、可視化、データ管理、及びアルゴリズム；グリッド資源管理；量子及びバイオコンピュータの概念
DARPA	多様なアーキテクチャ；ハイエンドかつ生産的で、堅牢なインテリジェントコンピューティング・システム；プロセッサ-イン-メモリアレイを含むチップ内及びチップ間通信のための光学材料の超高密度集積技術
DARPA/NSF	テラスケール・コンピューティングとストレージに用いる生物分子構造
NASA	ハイエンドソフトウェア及びシステムのチューニングと管理技術；インフォメーション・パワー・グリッドの技術とツール；情報物理学（センシング、処理及び記憶システムの特性）；量子及びナノスケール技術
NIH	ハイエンド生物医学コンピューティング；立体分子構造を決定するためのツール；測定データのイメージを表示及び分析するための方法
DOE/SC	テラスケールのモデル化とシミュレーション・アプリケーションのためのスケーラブルな数学的アルゴリズムとソフトウェア基盤（オペレーティングシステム、コンポーネント技術、最適な数学的ソルバ）；テラスケール科学のためのパートナーシップ
DOE/NNSA	米国の核貯蔵量管理のためのスーパーコンピュータ用モデル化とシミュレーションを可能にする高速計算、テラバイトデータ記憶と検索、及び可視化における科学及び工学上の革新
NSA	ハイエンドシステム製造企業との共同研究；オペレーティングシステムとプログラミング言語の改良；特殊目的のデバイスのための基盤技術（電力制御、冷却、相互接続、スイッチ、および設計ツール）；コンピュータメモリ性能；量子情報システムの基礎物理
NOAA	強化されたモジュラー・オーシャン・モデル（MOM）、フレキシブル・モデリング・システム（FMS）、スケーラブル・モデリング・システム（SMS）による改良された気候及び気象モデル
NIST	量子コンピューティング、安全な量子通信、最適化と計算幾何学、フォトニクス、ナノテクノロジー、オプトエレクトロニクス、及び新しいチップの設計と製造方法に関する研究
ODDR&E	量子通信とメモリに関する大学を中心とした研究
EPA	空気、水、及び土壌の相互作用のような複雑な環境現象をモデル化するためのパラダイム、技法及びツール

(2) ハイエンドコンピューティング能力

－発見のためのテラスケール・インフラストラクチャー－

1) NSF の分散型テラスケール施設(Distributed Terascale Facility：DTF)

NSF から$53M の資金提供を受けて 4 ヶ所の研究所（NCSA、SDSC、ANL、Caltech）にある Linux クラスタを 40 ギガビットの高速ネットワークで接続してグリッド環境（TeraGrid）を構築する。2003 年から運用を開始する予定。

（注：Blue Book には書かれていないが、2002 年 10 月に$35M の追加資金援助と、上記の 4 サイトに加えて Pittsburgh Supercomputing Center(PSC)の参加が決定された。これで合計 5 サイトとなり、ピーク性能の合計は 20TFlops、ストレージ容量の合計は 1 ペタバイト近くになるという。）

2）DOE の先進的コンピューティングによる科学的発見（SciDAC）

2001 年度から始まった Scientific Discovery through Advanced Computing (SciDAC)プログラムは、DOE のミッションに関連する科学分野（気象、核エネルギー、化学等）の基礎研究を、テラスケール・コンピュータを利用して推進するための、ソフトウェアおよびハードウェア基盤を開発することを目的としている。DOE による $57M の資金提供により、合計 51 のプロジェクト（3～5 年計画）が DOE の 13 ヶ所の研究所と 50 以上の大学で進行中である。

3）革新的なアーキテクチャにおける長期的研究

現在のハイエンド・プラットフォームは性能のスケーラビリティ、製造コスト、運用コスト等において限界に達しようとしている。DARPA の高生産性コンピューティング・システム（High Productivity Computing Systems : HPCS）プログラムは、民間企業や大学と協力して 2010 年までに性能、コスト、ソフトウェアの生産性／移植性、堅牢性、信頼性等を飛躍的に向上させる新しいアーキテクチャやコンポーネント技術を開発することを目標としたものである（2.3.4 節(1)参照）。

(3) 大規模ネットワーキング

－未来のインターネット：ダイナミックな柔軟性、広帯域幅、及びセキュリティー

主要な研究課題

- 信用：セキュリティ、プライバシー、および信頼度
- 適応性のある、ダイナミックでスマートなネットワーク
- ネットワーク性能の測定とモデル化
- 何十億個ものワイヤレス機器とセンサを含む、異種間ネットワークトラフィックの大幅な増加に対応する拡張性のある技術
- ミドルウェアのような垂直統合とサポーティングのツールやサービスを含む、ネットワーキング・アプリケーション
- 革命的な研究：複雑さの理論、一般化された制御理論、接続性が指数関数的に拡大する状況で、ネットワーク機能の進展に取り組むための異なったモデル

2003 年度における各機関の代表的活動

政府機関	活動の内容
NSF	ネットワーク化されたアプリケーションの性能を最適化するミドルウェアを中心とした研究；大学間における高性能な接続；ネットワーク・モニタリング、問題検出と解決、アクティブ／インテリジェントネットワークを支援する先進的自動化ツール、協調的アプリケーション、及び革新的アクセス方法のような戦略的インターネット技術
NIH	スケーラブルでネットワークを意識したワイヤレスの地理情報システム(GIS)、及びネットワーク化された医療関連環境におけるセキュリティ技術のアプリケーションの実証実験
DARPA	何百ものノードから成るネットワーク上における、ミリ秒単位から時間単位までのレンジにわたる振る舞いを予測できる、拡張性のあるネットワークのモデリング及びシミュレーションツール；ハイブリッドの光/RF 自己回復型ネットワークの実証実験
NASA	先進のコンピューティング、ネットワーキング及び協調技術を開発、実証するための、高速テストベッドネットワークの導入；グリッド環境のためのネットワークサービスの統合（QoS、受動的モニタリング、リソース確保）；ハイブリッドな衛星/モバイル、無線/アドホック・ネットワークアプリケーションと「未来のオフィス」の労働環境の実証
DOE/SC	分散型ハイエンド科学アプリケーションに対して、テラバイト/秒のスループットで TCP による信頼性の高い転送を可能にする高性能転送プロトコルの研究；大規模な科学的共同研究のためのエンド・ツー・エンド性能モニタリング、ネットワーク診断、及び拡張性のあるサイバー・セキュリティ・サービスの開発
NSA	バーストスイッチ技術、供給(Provisioning)、メッセージ受け渡し、低出力無線ネット、高速システムのファイヤーウオール、セキュリティと相互運用性の問題などを含む、先進的ネットワークトポロジーとプロトコル、ネットワークコンバージェンス、全光ネットワーク、及びネットワーク管理に関する研究
NIST	産業用プロトコルのモデル、新しい仕様の検証、適応制御メカニズムを含む、パーベイシブ・コンピューティング・デバイスのネットワーク通信のための標準規格；アドホック無線ネットワークの基準とプロトコル；応答性に優れたスイッチング・インフラストラクチャーの開発手法；インターネット・インフラストラクチャー・セキュリティのためのプロトコルと標準規格
NOAA	過酷な気象現象の予測と警報及び危険物への対応を支援する、拡張性のあるネットワーク能力とアプリケーションの早期導入
ODDR&E	リアルタイム耐故障ネットワーク・プロトコルに関する大学を拠点とした研究

1）グリッドのためのネットワーク向けミドルウェア”MAGIC”

ネットワーク環境でのアプリケーションの効率的実行には、ミドルウェアが重要な役割を果たす。最近注目を集めるグリッド（Grid）技術においては、NITRD 計画の下で開発された Globus と呼ばれるミドルウェアパッケージが標準的に利用されるなど、政府支援

によるミドルウェアの研究が継続して行われている。

① NSF ミドルウェアイニシアチブ(NMI)

NMI は NSF からの資金提供により 2001 年 9 月から開始されたイニシアチブであり、ネットワーク環境での再使用可能で拡張性に富んだミドルウェアを開発、配備、およびサポートすることを目的としている。

② MAGIC(Middleware And Grid Infrastructure Coordination)

MAGIC は、NITRD 組織の中の大規模ネットワーキング調整部会の内部に新たに設けられたチームである。本チームは、従来からある JET および NRT を主としてグリッドという観点から結びつけるもので、以下の役割を担っている。

- 省庁間のミドルウェアとグリッドに関する活動の調整
- 相互運用可能なグリッド技術の研究とその配備の促進・奨励
- 使用に適した、広く利用可能なミドルウェアツールとサービスの開発推進
- 上記技術における有効な国際協調に関するフォーラムの開催

(4) 先進のネットワーキング・アプリケーション

－人材と IT 資源を結びつけ米国の科学面でのリーダーシップを目指す－

2003 年度における各機関の代表的活動

政府機関	活動の内容
NSF	地震工学シミュレーションのためのネットワーク (Network for Earthquake Engineering Simulation：NEES) 国立バーチャル天文台(National Virtual Observatory：NVO) グリッド物理学ネットワーク(Grid Physics Network：GriPhyN)
DARPA	ネットワークを基本とした総合監視装置 (Network-Based Total Surveillance System：NBTS)
NASA	航空安全シミュレーション
DOE	研究者を科学とコンピューティング機能に接続
NIST	製造のためのソフトウェア相互運用性テストベッド 国際 iGrid デモンストレーション

2.3 ハイエンドコンピューティング分野の新しい動向

2.3.1 SC2002 に見る動向

2002 年 11 月 16 日から 11 月 22 日の間、米国メリーランド州ボルチモアにおいて第 15 回 International Conference on High Performance Networking and Computing（通称 SC2002）が開催された。この会議は毎年 11 月に開催されるスーパーコンピューティング関連の国際会議（開催地は米国）であり、会議の性格から、展示等は企業よりむしろ大学や国の研究機関からの参加が目立つのが特徴である。同時多発テロの影響により低調であった前回とは違い、今回は展示参加が 221 団体、参加者が 7000 人を超えて史上最大規模となった。

今回のテーマは”From Terabytes to Insights”（テラバイトから洞察へ）であり、ハイエンドのコンピューティング能力がもたらす科学研究の飛躍的進歩といったところに主眼が置かれていたようである。基調講演や招待講演でも科学研究に対するハイエンドコンピューティングの貢献の大きさと、さらなる高性能化への期待が述べられていた。

SC2002 の主要な話題は、やはり地球シミュレータをはじめとするハイエンドプラットフォームとグリッドおよび高速ネットワーク関連であった。展示関係では CRAY X1、IBM p655、Origin 3900 などのサーバ新製品の展示と科学技術アプリケーションのデモによる高性能のアピールが目立った。

なお、SC2002 での講演、発表および展示内容については、東京大学の小柳教授のレポートが詳しいので、そちらを参照されたい。

（http://olab.is.s.u-tokyo.ac.jp/~oyanagi/reports/SC2002）

次回（第 16 回）SC2003 は、アリゾナ州フェニックスで開催される予定である。

SC2003
Phoenix, AZ
November
15 - 21, 2003

参照サイトは http://www.sc-conference.org/sc2003/

テーマは”Igniting Innovation”（革新に火をつけよう）である。

(1) 地球シミュレータの与えたインパクト

SC2002 における最大の話題は、2002 年 6 月にその圧倒的な性能により Top 500 リスト

の第 1 位に躍り出た地球シミュレータであった。会期中には地球シミュレータセンター長の佐藤哲也氏の招待講演があったほか、米国政府の NSF や DOE のディレクターの講演でもしばしば言及されていた。また、投稿された論文の中から選ばれて、Gordon Bell Awards を受賞した 5 件の論文うちの 3 件が地球シミュレータ関連であった。さらに、パネルディスカッションではそのものずばりの”The 40Tflop/s Earth Simulator System: Its Impact on the Future Development of Supercomputing”というタイトルのセッションまで設けられ、活発な議論が行われるなど、その影響の大きさがうかがえた。

上記のように地球シミュレータが大きく取り上げられた背景には、「危機感を煽って米国政府からの R&D 予算獲得の口実とする」という理由も存在したであろうが、以下のような議論を呼び起こすきっかけを作ったということもあったと思われる。

① 実効性能に関する議論

現在の主流は、汎用スカラープロセッサと汎用ネットワーク、すなわち Commodity Off-The-Shelf(COTS)製品を使用した超並列スカラー型システム（PC クラスタを含む）である。しかしこの方式における実効性能（実アプリケーション全体を通しての平均 Flops）は、一般的にピーク性能の 10 パーセント程度と言われており、ベクトル型システムより劣っている。米国が捨てたベクトル方式を採用した地球シミュレータが、実アプリケーションの「全球大気大循環シミュレーション」でピーク性能の 66 パーセント（26.58TFlops）という驚異的な実効性能を実現したことが、システムアーキテクチャと実効性能についての議論をあらためて呼び起こしているように見える。

② ハイエンド領域も COTS でいいのか？

米国のハイエンドコンピュータの主流は、上記の COTS 製品を使用したものである。安価な点において、この方式の一般市場での優位は、もはや動かしがたいところまで来ている。しかし、国家の安全保障や科学研究などの最高性能を必要とする領域で、COTS をベースとしたシステムしか解がないことへの問題意識も高まりつつある。

地球シミュレータは、わが国ではさほど話題になっていないが、それとは対照的に、米国の関係者には非常に大きなショックを与えたようである。科学技術における米国のリーダーシップの危機といった論調まであり、DOE などが中心となって 2 年以内に地球シミュレータの性能を凌駕するシステムを開発しようと躍起になっている。

一方、わが国は地球シミュレータ以後、国が主導する高性能プラットフォーム開発の具体的計画を持っていない。このままだと 2～3 年後には米国が再び世界一の座を取り戻し、

ハイエンドコンピューティング分野における日本発のセンセーションも、一時的な現象に終わるのではなかろうか。

(2) Top 500 にみる性能動向

毎年 6 月にドイツで開催される ISC(International Supercomputer Conference)と 11 月に開催される SC において、"Top 500 Supercomputer Sites"（LINPACK ベンチマークによる性能測定結果の上位 500 サイト）が発表されている。

SC2002 で発表された Top 500 には、地球シミュレータに続き、新たに 2 位と 3 位にロスアラモス国立研究所(LANL)の ASCI Q がランクされた。また、5 位にローレンス・リバモア国立研究所(LLNL)の NetworX 社製 PC クラスタ（Xeon 2.4GHz x 2,304 プロセッサ）、8 位に NOAA の HPTi/Aspen Systems 社製 PC クラスタ（Xeon 2.2GHz x 1,536 プロセッサ）と 2 台のシステムが PC クラスタとしては初めて Top 10 内にランクインするなど、全般的に PC クラスタの増加が目立っている（付表 1、2 参照）。

図 2.2 は 1993 年から 2002 年まで 10 年間の Top 500 の 1 位と 500 位の性能推移をグラフにしたものである。過去 10 年間は年率 1.9～2.0 倍の割合で性能が向上しており、今後もこの傾向が続くとすれば、2005 年に 1 位が 100TFlops、500 位が 1TFlops に達し、さらに 2009 年には 1 位が 1PFlops（ペタフロップス）に達すると予想される（あくまでも LINPACK ベンチマークの性能であり、理論ピーク性能ではないことに注意）。

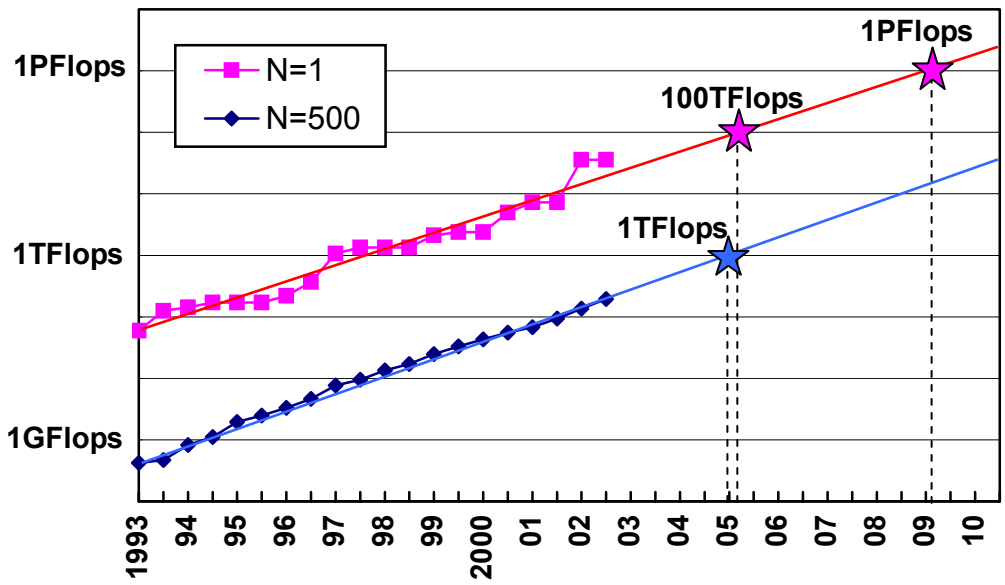


図 2.2 Top 500 の性能トレンド

2.3.2 超並列コンピューティング

(1) ASCI 計画

DOE の国家核安全保障管理局(National Nuclear Security Administration : NNSA)が実施している ASCI 計画（Advanced Simulation and Computing Initiative[2]）は、1995 年の発足以来、米国のハイエンドコンピュータ開発の中心的な役割を担っており、当ワーキンググループでは過去の報告書において ASCI 計画の状況を毎年報告してきた。最近になっても”Red Storm”、”Purple”、”Blue Gene/L”などの計画が次々と発表されるなど、相変わらず積極的な投資が目立っている。以下では本計画の下で開発されているシステムの最新状況について述べる。

1） ASCI Q

ロスアラモス国立研究所（LANL）に設置される ASCI Q は、HP（旧コンパック）の Alpha サーバ ES45 をベースにしたクラスタシステムで、当初 2002 年にピーク性能 30TFlops のシステムが完成する予定であった。現在 1/3 規模のシステム（4,096 プロセッサ／1,024 ノード）が 2 セット（QA、QB）完成した段階にあり、これらの 2 セットがそれぞれ LINPACK ベンチマークで 7.73TFlops を記録して Top 500 リストの 2 位と 3 位に登場している。しかし、残りの 1/3 である QC が設置され、さらにシステム全体が統合されるのがいつになるかは不明である。一足先に完成した地球シミュレータや後述する ASCI Purple と Blue Gene/L の発表の影響で影が薄くなった感は否めない。

2） ASCI Red Storm

Red Storm は CRAY, Inc.により開発、製造される予定のスカラー型システムである。サンディア国立研究所に納入され、2004 年 8 月から運用を開始する。AMD の Opteron プロセッサ(SledgeHammer)を 10,368 個使用し、ピーク性能は 41.5TFlops（LINPACK 性能目標は 14TFlops 以上、ASCI Red の 7～8 倍）と発表されている。
（SC2002 での発表資料による。）

3） ASCI Purple と Blue Gene/L

2002 年 11 月 19 日に IBM により開発、製造される ASCI Purple および Blue Gene/L（昨年度の報告書で紹介済み）がローレンス・リバモア国立研究所(LLNL)に納入されることが発表された。

---

[2] 以前は Accelerated Strategic Computing Initiative と呼ばれていた。

ASCI Purple は IBM の POWER5 プロセッサを 12,544 個使用し（64 CPU x 196 ノード）、ピーク性能は 100TFlops、主記憶容量 50 テラバイト、ディスク容量 2 ペタバイト（約 2,000 兆バイト）という巨大なシステムである。ASCI Purple は 2004 年末までに運用を開始する予定である。

Blue Gene/L は IBM が最終的にペタフロップスを目指している Blue Gene 計画の最初のシステムであり、2005 年までに完成する予定である。Blue Gene/L は 1 チップ上に 2 プロセッサ(700MHz)を実装し、これと 256MB のメモリで 1 ノードを構成する。さらに 65,536 ノード（131,072 プロセッサ）を 3 次元トーラス接続して、ピーク性能 360TFlops を実現するとしている[3]。過去このように多数のノード（65,536 ノード）から成るシステムは例が無く、さらにノードあたりのメモリが 256MB と小さいため、高い実効性能を出すためには、これまで以上に高度なタスク割り当て制御が必要と思われる。

図 2.3 に SC2002 で発表された Blue Gene/L の構成概念図を示す（昨年度報告書に掲載したものから若干の変更あり）。詳しくは SC2002 における発表論文[1]を参照されたい。

また、Blue Gene/L の後にもきわめて意欲的な計画が続いている。Blue Gene/P は 2006

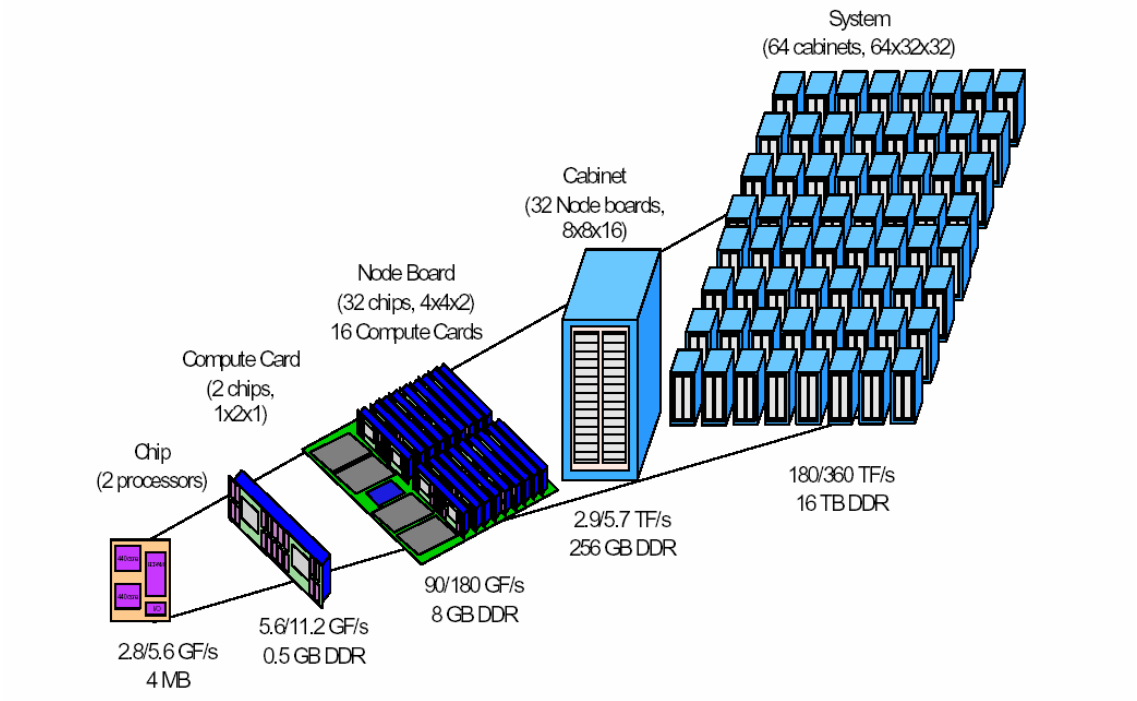


図 2.3 Blue Gene/L の構成概念図（出典：文献[1]）

[3] 1 ノード（2 プロセッサ）中の片方のプロセッサは主にノード間メッセージ転送用に使用されるため、通常演算に使用されるのは 65,536 プロセッサである。したがってノード間メッセージ転送がきわめて少ない特殊なアプリケーション以外は、ピーク性能 180TFlops とするのが妥当と思われる。

年～2007 年にピーク性能 1PFlops（実効性能 300TFlops）、Blue Gene/Q は 2007 年～2008 年にピーク性能 3PFlops（実効性能 1PFlops）を実現するとしている。にわかには信じがたい計画だが、もしこれが予定通り実現されるとすれば、わが国のメーカはとても太刀打ちできないに違いない。

(2) その他

ASCI 計画とは別に、地球シミュレータ対抗を強く打ち出しているものに、ローレンス・バークレイ国立研究所(LBNL)、アルゴンヌ国立研究所(ANL)および IBM が共同で提案している Blue Planet 計画というものがある[2]。その目標は、2005 年末までに地球シミュレータの 2 倍の実効性能のシステムを、半分のハードウェアコストで実現するというもので、ピーク性能は 150TFlops、少なくとも数種類の実科学計算コードで 40～50TFlops の実効性能を実現することを目指している。

IBM の POWER5 を科学技術計算向けに改良したものを 16,384 プロセッサ使用し、また Virtual Vector Architecture(ViVA)と呼ぶ疑似ベクトル処理（CRAY X1 等でも使用している、ノード内の複数 CPU を連動させてベクトル処理を行う機能）を採用する。プロセッサを POWER6、7 とアップグレードすることにより 2009 年にはペタフロップスを目指すとしている。この計画が実行に移されるかどうかは、今のところ不明であるが、もし実施されるとすれば、将来は Blue Gene 計画と統合されるものと思われる。

2.3.3 グリッド・コンピューティング

2002 年は、グリッド・コンピューティングが研究段階から実用段階へと移行する節目の年であったと言えるかも知れない。IBM などの大手メーカがグリッド・コンピューティングビジネスへの本格的投資を発表して以来、「グリッド・コンピューティング」は技術系／ビジネス系のメディアに頻繁に登場するようになった。また、わが国も含めて、科学研究コミュニティでのグリッド構築が一段と加速している。以下にグリッド・コンピューティングに関する最近の動向を、①基盤技術の標準化の促進、②科学研究コミュニティのグリッド形成の加速、③ビジネス化へ向けた動きの本格化、という 3 つの視点から述べる。

(1) 基盤技術の標準化

2000 年 11 月にグリッド基盤技術の世界的標準化団体である Global Grid Forum(GGF)が結成され、インターネットにおける IETF と同様の手続きで標準化の作業が進められて

いる[4]。　GGF では 2001 年より毎年 3 回づつワークショップを開催しているが、2002 年 7 月の第 5 回ワークショップ(GGF5)参加者は約 900 人とそれ以前（500 人以下）の約 2 倍であり、グリッド・コンピューティングに対する関心が急速に高まっていることを示している。

わが国では 2002 年 6 月にグリッド協議会が設立され、GGF に対する標準化の提案のほか、グリッド・コンピューティングの啓蒙活動、情報共有などが実施されている。

(2) 科学研究コミュニティのグリッド形成

グリッド技術は、現在の一般的なうたい文句である大規模分散コンピューティング（計算）への応用面のほかに（それ以上に？）、遠隔地にある実験装置、計算資源へのアクセスやデータベースの共有、遠隔地間でのコラボレーションなどにも応用される[3]。最近になって、様々な科学研究コミュニティの間で、主として後者の目的によるグリッドの構築が活発化してきている。以前はグリッドそのものの実証実験的意味合いが強かったが、今は科学研究への実質的貢献を目指した応用面主体のものに変わりつつある。以下に米政府の支援により構築されている（されつつある）主要な科学研究グリッドを示す。

プロジェクト名	支援機関	分野／参照サイト
Earth System Grid (ESG)	DOE	気候変動研究 http://www.earthsystemgrid.org/
Grid Physics Network (GriPhyN)	NSF	天文学、高エネルギー素粒子物理学 http://www.griphyn.org/
National Virtual Observatory (NVO)	NSF	天文学（仮想天文台） http://www.us-vo.org/
Network for Earthquake Engineering Simulation (NEES)	NSF	地震工学 http://www.nees.org/
Biomedical Informatics Research Network (BIRN)	NIH	バイオ医学研究 http://birn.ncrr.nih.gov/

(3) ビジネス化へ向けた動き

グリッド・コンピューティング技術は、従来から政府支援のもとで大学と国研により研究開発が行われてきた。その後、昨年の報告書にあるように、AVAKI、Platform Computing、Entropia などのベンチャー企業を中心に企業化が始まった。2002 年 2 月には Platform Computing 社がオープン・ソースである Globus Toolkit の業界初の商業版であ

[4] それ以前は米国の Grid Forum、ヨーロッパの European Grid Forum(eGrid)、アジア太平洋の Asia-Pacific Grid Forum(ApGrid)が、それぞれの地区のグリッド開発を推進していた。

る”Platform Globus”を発表している。また、最近になって、米国の大手メーカのグリッドビジネスへの参入が本格化しており、現在、グリッド・コンピューティングのビジネス化を目指している主要な大手メーカとして、IBM、Sun、HP、マイクロソフトなどが挙げられる。特に IBM は Globus Project と共同で、グリッドと Web サービスとの統合化を目指す Open Grid Services Architecture (OGSA) の仕様を提案しており、GGF は 2003 年にこれを標準として認定する見込みである。IBM のグリッド事業責任者である Tom Hawk 氏は「2005 年にはグリッド関係のビジネス分野向けの売り上げが科学技術分野を追い越すことになるだろう」との見解を明らかにしている(日経 IT Pro 2002 年 5 月 13 日記事)。

以上のように、グリッド・コンピューティングのビジネス化の動きが活発になっているが、実際はグリッドというキーワードが先行し、後からビジネスの為の応用を考えているというのが正確な状況であろう。これをビッグビジネスに成長させるためには、セキュリティや信頼性などの技術的課題を克服することと並んで、（どのような技術にも言えることであるが）いかに技術の特性を生かしたビジネスモデルを作り、キラーアプリケーションを生み出せるかが鍵になると思われる。

2.3.4 将来のコンピューティングに関する研究

(1) DARPA の”High Productivity Computing System”プログラム

DARPA は 2002 年より”High Productivity Computing System”（高生産性コンピューティング・システム：HPCS）プログラムを開始した。

本プログラムは 2010 年までの 9 年間にわたる長期計画であり、現在のコンピュータシステムと、製品化がまだ当分先と言われている量子コンピュータとの間のギャップを埋めるものと位置づけられている。計画はフェーズ 1（概念研究：1 年)、フェーズ 2（R&D：3 年)、フェーズ 3（大規模開発：5 年）に分かれている（図 2.4 参照)。フェーズ 1 については BAA(Broad Agency Announcement)により企業や大学からの提案が公募され、CRAY、IBM、SGI、Sun 等の企業が提案を行っている。以下に HPCS プログラムの生まれた背景とその目的について記す。

1) 背景

- 現在の商用高性能コンピュータ技術のトレンドでは国家の安全保障に必要とされる能力とのギャップが大きくなるばかりである。
- 政府主導による研究開発が無ければ、ハイエンドコンピューティングは主要な関心が一般大衆やビジネス市場に向いた企業の製品からしか得られず、これでは重要な

国家安全保障に関するアプリケーションが効果的に実行できない。

- 以下のような重要な分野において、米国の国防総省（DoD）や産業界の優位性を確保する必要がある。
  －軍事目的の気象、海洋予測
  －大気中の汚染物質の分布解析に関する計画演習
  －暗号解読
  －軍用プラットフォームの分析
  －サバイバビリティー／ステルス設計
  －諜報／監視／偵察システム
  －巨大な航空機、船舶および構造の仮想生産／故障解析
  －先端的バイオテクノロジー

2）目的

- ハイエンドコンピューティングの新しいビジョンである「高生産性コンピューティング(HPCS)」を実現するための次世代のハイエンド・プログラミング環境、ソフトウェアツール、アーキテクチャ、およびハードウェアコンポーネントを生み出すことに焦点を当てた研究開発プログラムを設け、実効性能、スケーラビリティ、ソフトウェアツールや環境、および増加する物理的制約などの課題に取り組む。
- 今日の 80 年代後半の技術を元にした高性能コンピューティング（HPC）と将来の量子コンピューティングの間を埋める。
- 国家安全保障と産業界のユーザコミュニティのために、経済的な高生産性コンピューティング・システムを以下のような設計条件でこの 10 年の後半（2007 年～2010 年）に実現する。
  －性能：計算効率と重要な国家安全保障のアプリケーション性能を 10～40 倍向上
  －生産性：アプリケーションの開発、運用、および保守コストを低減
  －移植性：研究および業務用 HPCS アプリケーションソフトウェアがシステム仕様に影響されないようにする
  －堅牢性：HPCS ユーザにより高い信頼性を提供し、犯罪行為によるリスクを低減

3）プログラム計画

図 2.4 に本プログラムの計画を示す。

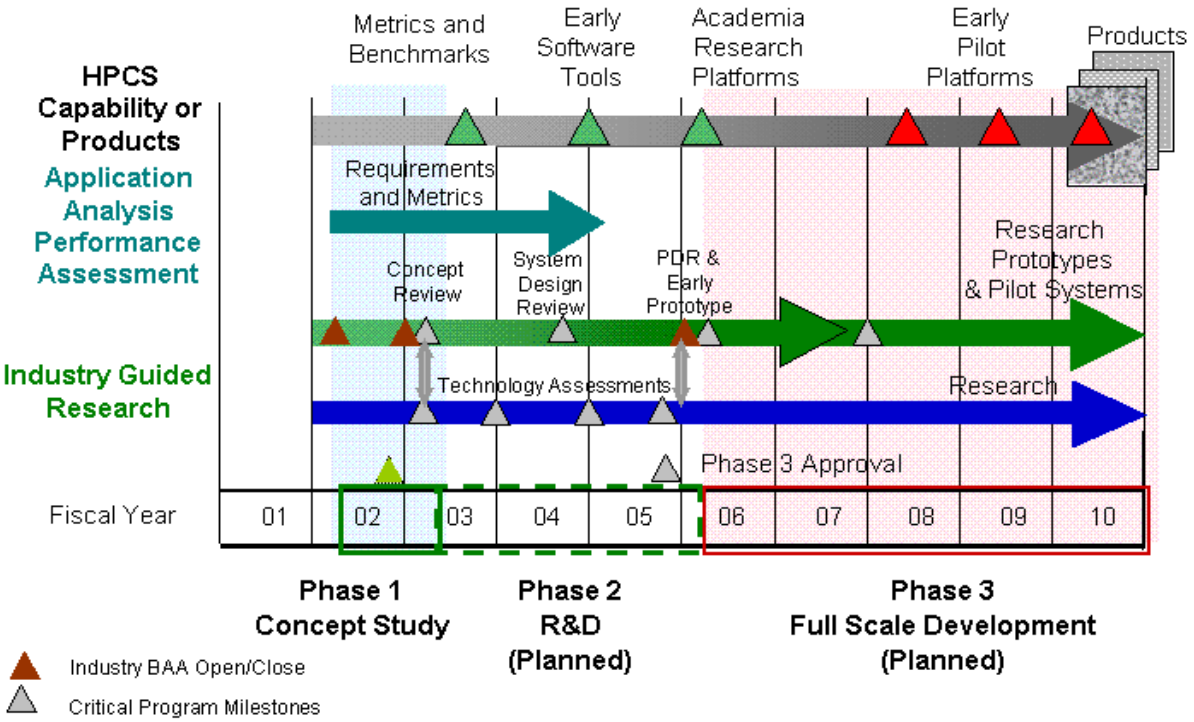


図 2.4　プログラム計画－DARPA のサイトより
（http://www.darpa.mil/ipto/research/hpcs/plan.html）

(2) 量子コンピュータの研究動向

量子コンピュータの実用化は数十年先（20 年以内から 50 年以上まで種々の意見がある）と言われている。ドッグイヤーと呼ばれる近年の IT の進展速度からすれば、はるか遠い未来の話と言ってもいい（このことを考えれば、前述の HPCS プログラムのようなものが生まれてくるのもうなずけることであろう）。しかし、米国政府はこの遠い将来の実用化に向けて、金額的にはまだそれほど大きくないにしても、この 10 年間継続して着実に研究支援を行ってきている[4]。

以下に米国連邦政府の資金による主要な量子コンピュータ研究プログラム（公募）および政府機関における研究の一部を挙げておく。

- NSA　ARDA[5](Advanced Research and Development Activity)
  (http://www.ic-arda.org/Quantum/)
- NSF　Quantum and Biologically Inspired Computing(QuBIC) Program
  (http://www.nsf.gov/pubs/2002/nsf02017/nsf02017.html)

[5] ARDA は NSA などの情報機関が実施する先端的研究に対するファンディング組織である。

- NASA　JPL Quantum Computing Technology Group
  (http://cs.jpl.nasa.gov/qct/qat.html)
- NIST　Physics Laboratory, Quantum Information Program
  (http://qubit.nist.gov/)
- DARPA　Quantum Information Science and Technology(QuIST) Program
  (http://www.darpa.mil/ipto/research/quist/)
- DOE　LANL
  (http://qso.lanl.gov/qc/)

その他、Caltech、Stanford、MIT、UCB など多くの大学でも研究が行われている他、民間企業では IBM の活発な研究が目立っている。

また、日本においても政府系研究機関や富士通、NEC などの民間企業でも研究が行われ、米国と対等な成果を出しつつある。

参考文献

[1] NR. Adiga, et al.: "An Overview of the BlueGene/L Supercomputer" http://sc-2002.org/paperpdfs/pap.pap207.pdf

[2] Creating Science-Driven Computer Architecture: A New Path to Scientific Leadership - A Strategic Proposal from ANL and LBNL－, pp.31-36, Oct. 2002 http://www.nersc.gov/news/ArchDevProposal.5.00.pdf

[3] 関口智嗣, "Grid による世界戦略とブロードバンドコンピューティング技術" ハイエンドコンピューティング技術に関する調査研究 II, 先端情報技術研究所(2001) pp.42-43, 2001 年 3 月

[4] 総務省, "21 世紀の革命的な量子情報通信技術の創生に向けて", 2-5-1, 2000 年 6 月 http://www.yusei.go.jp/policyreports/japanese/group/tsusin/00623x01.html#20

（石井　英志　幹事）

第 3 章　ハイエンドコンピューティング研究開発の動向

3.1　概要

本章では、各委員および外部講師による、ハイエンドコンピューティングに関する研究開発動向の調査報告をまとめている。

3.1.1　調査方針

各委員の専門およびその周辺分野において、今後、研究が急速に発展しそうな、あるいは市場にインパクトを与えそうな分野やテーマを抽出し、それらの技術に関する(米国をはじめとする)先端研究と、わが国のそれとの格差の調査・評価を行うことを主眼とした。したがって、今後重要になると思われる領域にはどのようなものがあるかを抽出するという点に力点を置くことにした。また、調査範囲は多岐にわたっており、当ワーキンググループ委員の専門分野で網羅できない部分は外部講師に依頼した。

3.1.2　外部講師

今年度は下記の 2 名の方に外部講師として講演を依頼し、さらに調査報告をまとめていただいた。1 件は、今後のハイエンドコンピューティングにとってますます重要となる、プロセッサの性能と消費電力の問題への取り組みに関する論文であり、もう 1 件は、グリッドと並ぶネットワーク利用技術の重要テーマである Web サービス（最近は両者の統合を目指す動きもある）に関する論文である。

(1) 東京大学　先端科学技術研究センター
　　中村　宏　助教授
「ソフトウェア可制御オンチップメモリを用いた高性能・低消費電力プロセッサ」

(2) 日本電気(株)　インターネット基盤開発本部
　　藤田　悟　マネージャー
「Web サービスの技術動向」

3.2 アーキテクチャ＆新計算モデル

3.2.1 粗粒度 Reconfigurable Device の挑戦

天野 英晴 委員

1. はじめに

リコンフィギャブルコンピューティング(Reconfigurable Computing)とは、解法アルゴリズムを、書き換え可能な IC(FPGA,CPLD)上で直接ハードウェア化して実行する方式である[1][2]。専用システムの高速性と書き換え可能な柔軟性を併せ持つ方式として、1990年代はじめから研究が続けられてきた。しかし、数値演算、データアクセスの高速性、規模の限界、プログラミング環境等様々な面で問題があり、商用システムとして一般的に使われるに至らなかった。

しかし、2000 年代に入ってから、FPGA、CPLD の大容量化、高性能化が進むと共に Chameleon[3]などの専用チップが登場したことにより、通信処理をはじめとする一部のアプリケーションで一定の地位を確立するに至った。そして 2002 年には、NEC の DRP[4]、Quicksilver の ACM[5]、IPFlex の DNA チップ[6]など新しいリコンフィギャブルコンピューティング専用チップが発表され、話題になっている。これらのチップは、動作中に高速に構成を変更する動的再構成機能(Dynamic Reconfigurable)を持つ他、粗粒度構成、C 言語ベースの設計環境などの共通の特徴を持つ。

Reconfigurable System については、1998 年度の報告書[7]、2001 年度の報告書[8]に、その概観と将来予測について述べたが、今回はこれらの新しいデバイスを紹介し、その特徴と将来予測を述べる。

2. Reconfigurable System の概観

書き換え可能なデバイスは、ユーザが手元でプログラミングするデバイスであり、90 年代に入って、システムをまるごと実装可能な規模の FPGA(Field Programmable Gate Array)あるいは CPLD(Complex Programmable Logic Device)が登場するに至った。このうち、内部の配線データ(Configuration Data)を SRAM に格納するタイプが特に発展し、100MHz 近い周波数で動作する高速性、100 万ゲートを越える実装密度、RAM や演算器、コア CPU を内蔵するものも登場し、ASIC に代わってシステムインテグレーションの主役の座についている。

これらの FPGA や CPLD は、配線データを入れ換えることにより、電源を入れたままでハードウェア構成を変えることができる。この性質を利用し、状況に応じてハードウェア構成を変化させたり、解法アルゴリズムを直接ハードウェア化させることのできるシステムを Reconfigurable System(可変構造システム)あるいは Custom Computing Machine：CCM(直接解法マシン)と呼ぶ。

初期の Reconfigurable System は、汎用の FPGA や CPLD を用いて構成され、独立動作を行なうスタンドアローン型と、汎用 CPU との分散協調動作を目的とするコプロセッサ型に分けられる。これらの比較的古典的なシステムについてのサーベイは、2000 年度の報告書[9]を参照されたい。しかし、これらの古典的なシステムには、以下の問題点があった。

a) FPGA による演算処理は、IEEE 浮動小数点演算を実行する場合、専用の高速プロセッサや DSP に比べて 10 倍程度遅い。また、チップ面積は専用チップの 10 倍必要である。
b) FPGA は、メモリとの接続が脆弱で、大量のデータを扱うアプリケーションでは汎用プロセッサに比べ不利である。
c) 構成の変更に時間がかかる。このため、解ける問題のサイズが FPGA で実現できるサイズを越えるとお手上げになってしまう。
d) プログラミングが困難である。またプログラム=構成情報の生成に時間がかかる。

2002 年に登場した新しいデバイスは、これらの問題点の本質的な解決を目指している。以下、個々のデバイスを簡単に紹介する。

3. 新しい Reconfigurable Device

表 1 に今回紹介する新しいデバイスを一覧する。このうち NEC の DRP は、筆者らが実際に研究で用いているため、詳細まで記述してあるが、他は文献や web に基づくもので、対象によっては情報が多くない。

表 1　最近の粗粒度 Reconfigurable Device

名称	製造	構成要素	接続	再構成	プログラム
DRP	NEC	8bit FF/DMU /ALU	格子状バス	16 セットの Multicontext 1clock で切り替え	Cyber-C
DNA Matrix	IP Flex	32bit ALU	格子状バス	Configuration RAM より 1clock でセット	C Matlab

Xpp	PACT Informations technologie	32bit ALU	階層型格子状バス	Partial Configuration	Xpp C NML
ACM	Quicksilver	異種類 PE	H-Tree	Configuration RAM より 1clock でセット	C
PipeRench	CMU	8bit PE	直線構造	ストライプを 1 クロックで切り替え	C 他

これらのデバイスは､いずれも数十ゲート相当の基本構成要素から成る細粒度の FPGA、CPLD に比べ、8bit-32bit の演算器レベルが基本構成要素である。すなわち、粗粒度の Reconfigurable Device である。粗粒度にすることにより、専用の高速プロセッサや DSP に迫る演算性能を得ることができ、a)の問題を解決する。

また、分散した内蔵 RAM との結合を密にすること、もっぱらストリームデータ処理に特化することで、b)の問題を解決する。さらに粗粒度にすることで、構成情報の量を減らすことができることから、構成の変更に要する時間を削減できる。このことで、1 クロックあるいは数マイクロ秒の時間で動的に再構成を行なうことを可能としている。動作中に短時間で再構成を行なうことにより、c)のサイズの制約を突破し、新しい応用分野を広げることができる｡また､プログラムは粗粒度の演算器間のデータフローに帰着されるため、比較的容易に C などの高級言語からの生成が可能であり､このことにより d)の問題点を解決する。最後に、これらのデバイスで最もポイントになるのは、並列処理による性能向上である。以下、それぞれのデバイスを簡単に紹介する。

3.1 NEC DRP[4]

a) アーキテクチャ

NEC により 2002 年に発表された DRP(Dynamically Reconfigurable Processor)は､4X2 の Tile と呼ばれる reconfigurable unit から構成される。各 Tile は図 1 に示す通り、8X8 の PE(Processor Element)アレイ、状態制御を行なうシーケンサである STC(State Transition Controller)から構成される。また、8bitX256 エントリのメモリ 8 セットを両側に持ち、これらを制御するメモリコントローラを 2 セット持つ。

各 PE は 8bit の ALU, シフトやデータ制御、簡単な論理演算を行なう DMU(Data Management Unit)、8bit の Flip Flop、レジスタファイルから構成される。DRP は、上記構成の Core の周辺部に 32 ビットの乗算器を 8 セット、メモリモジュール、PCI バスインタフェース、
SDRAM/CAM/SDRAM インタフェースを搭載したシステム LSI となっており、単独で

PCI バスへの接続や外部メモリの制御が可能である。

DRP は、同社が 1998 年に開発した DRL 同様、1 クロックでコンテキスト切り替えが可能なマルチコンテキストデバイスで、Tile 単位の部分再構成が可能である。一方で、DRL が LUT(Look Up Table)を構成要素とした細粒度のリコンフィギャブルデバイスであったのに比べ、DRP は 8bit の PE を構成要素としたリコンフィギャブルなプロセッサである点が全く異なっている。

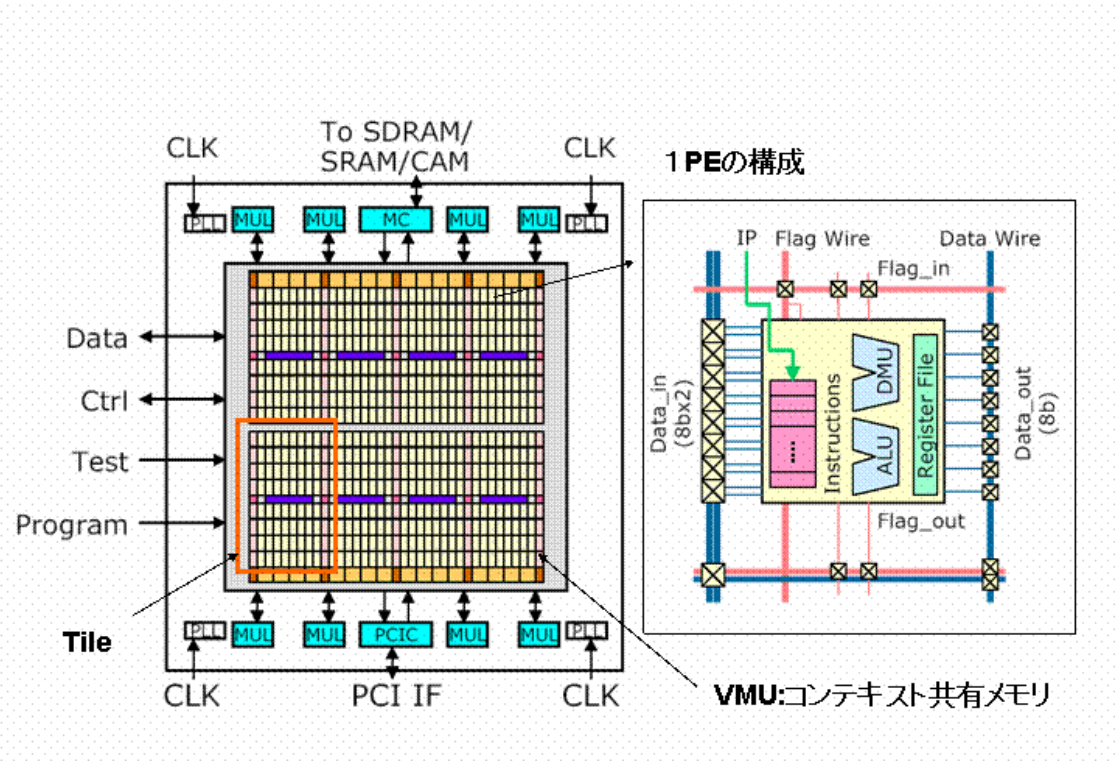


図 1　DRP-1 の構成

b）再構成

DRP は内部のメモリに最大 16 コンテキスト分の情報を蓄えることができ、最大で 5 コンテキストの中から次のコンテキストを選択し、動的に再構成をすることができる。次のコンテキストの選択は STC に信号を送ることで行ない、動作を止めることなしに 1 クロックで行うことができる。

さらに、動作中に、コンテキスト切り替えの対象外のコンテキストに対してコンフィギュレーション ロードが可能である。この特徴により、仮想ハードウェアや動的適応ハードウェアの効率的な実現が可能となる。

これらの、コンテキスト切り替えや、Configuration data ロードは、Tile 単位で行なうことができ、Tile ごとに異なるコンテキスト構成を取ることができる。

c)プログラム

C 言語に制約を加えた Cyber C からの構成情報の生成が可能である。

3.2 IP Flex DNA Matrix[6]

a) アーキテクチャ

DNA Matrix は、IP Flex 社が富士通と共に 2002 年に開発したチップで、32bit の ALU が縦横のバスで接続された構造を持つ可変構造のプロセッサアレイである。デジタル家電向けとして新聞発表が行われたように、DNA Matrix は、制御用の RISC である DAP と用途に応じて様々な形で組み合わせることで多様な応用が可能である。

b) 再構成

演算器間の接続はオンチップの Configuration RAM 上に格納され、1 クロックで切り替えが可能である。

c) プログラム

C 言語および Matlab のエントリがある。

3.3 PACT XPP[9]

a) アーキテクチャ

ドイツの PACT Informations technologie により発表された PACT XPP(eXtreme Processing Platform)は、階層的な構造を持つ再構成可能なプロセッサアレイである。

図 2 に示すように、32bit の ALU、FREG(Forward Register Object), BREG(Backward Register Object)から構成される PAE(Processing Array Element)を 2 次元アレイ状に接続し、PAC(Processing Array Cluster)を構成する。この PAC を複数接続することでシステムを構成する。バスの縦横の端には、I/O が装備され、さらに Configuration 用 RAM とマネージャが Configuration bus を介して Configuration data を流し込む。それぞれのアレイには互いに同期をとってパイプライン動作を容易に行なうことができる。

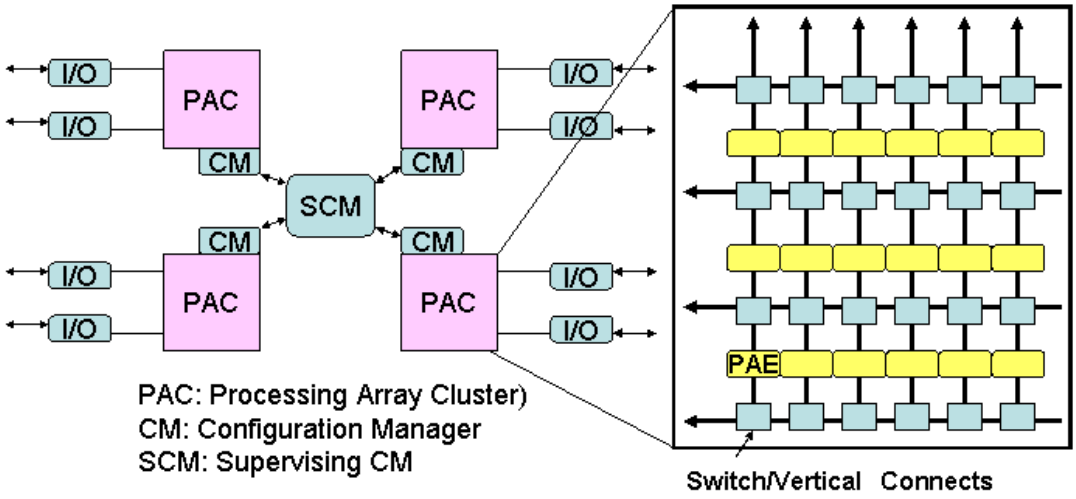


図 2　PACT　XPP の構成

b）再構成

Partial reconfiguration が可能である。また、一部を動作させながら、RAM より高速に Configuration 情報を流し込むことによりマイクロ秒オーダーで再構成が可能である。

c）プログラム

上位レベルの記述エントリとして XPP C が開発されている。XPP C は、標準 C のサブセットで、特殊な I/O ライブラリを用いる。記述は、XPP C Complier により構造化された HDL である NML(Native Mapping Language)に変換される。この NML を用いて直接記述を行なうことも可能である。

3.4　Quicksilver ACM[5]

a）構成

Quicksilver 社から 2002 年に発表され、FPL2002 の基調講演で発表が行なわれたが、詳細な情報が公開されていないので不明な点が多い。図 3 に示すように、算術計算用、ビット処理用、状態マシン、スカラープロセッサの 4 種類のプロセッサがクラスタを構成し、これらのクラスタを 4 つずつまとめて、階層型ツリー構造で接続される構造を持つ。

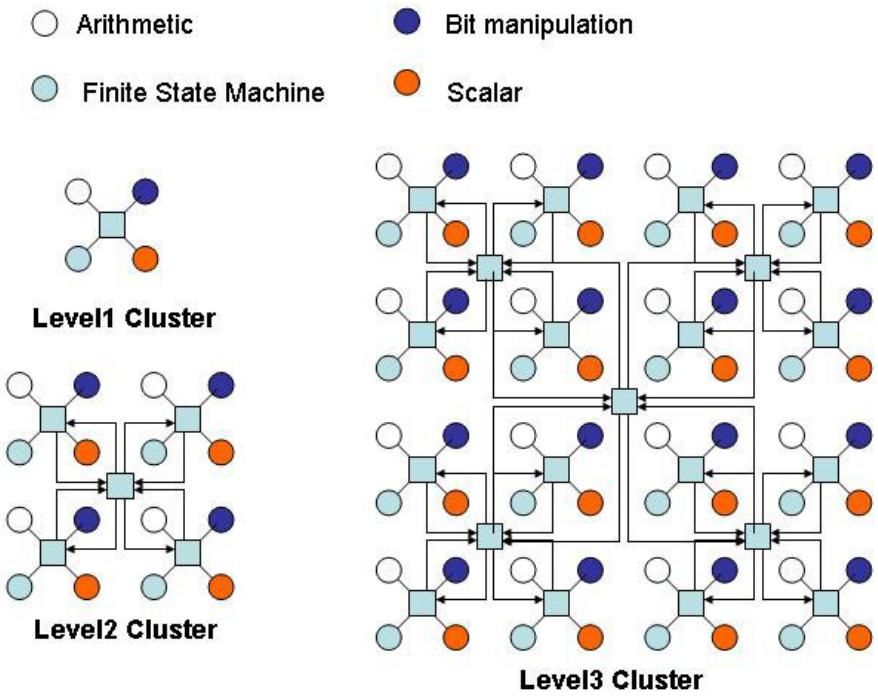


図 3　Quicksilver　ACM の構造

b）再構成

演算器間の接続はオンチップの Configuration RAM 上に格納され、1 クロックで切り替えが可能である。

c）プログラム

C 言語でのプログラムが可能である。

3.5　CMU　PipeRench[10]

a）アーキテクチャ

1997 年から CMU で続けられているプロジェクトで 2002 年にチップが完成して話題となった。図 4 に示す、直線構造のパイプラインを基本とし、各 Stripe には 8bit 構成の PE を 16 セット持つ。順にパイプラインを切り替えていくことで、仮想的に長大なパイプライン構造を実現することができる。

b）再構成

Stripe 単位に 1 クロックで構成を変更することが可能である。2002 年に完成したチップは、チップ内に実 Stripe を 16 保持し、これを切り替えるための仮想 Stripe を 256 セット保持することができる。

c)プログラム

ホスト上の API を用いて様々な環境からホストと協調動作する環境が作成されている。

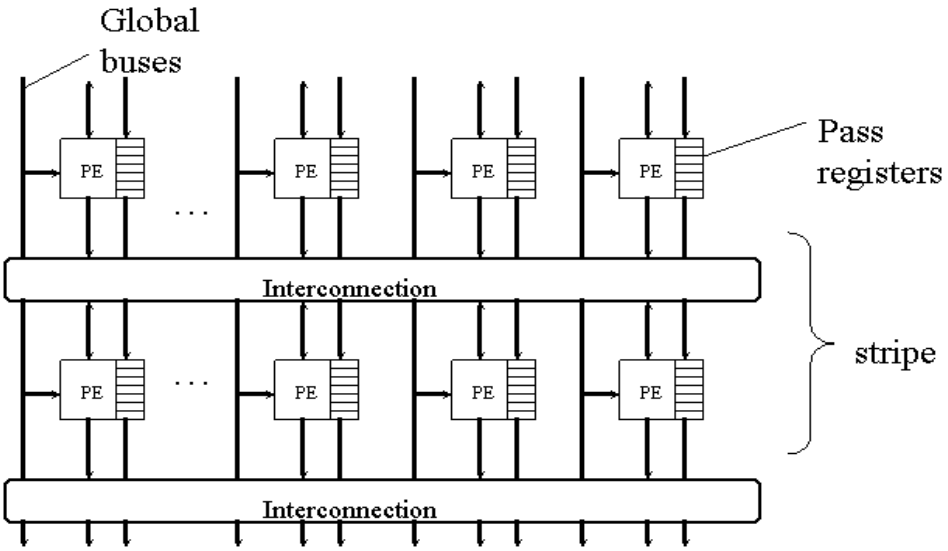


図 4　PipeRench の構造

4. 粗粒度 Reconfigurable Device の将来と問題点

粗粒度 Reconfigurable Device の当面の応用分野はストリームデータを扱う通信における符号化、暗号化、復号化、音声、画像処理への応用が期待されており、実際にかなりの性能向上が実現されている[11][12]。

さらに、これらの粗粒度 Reconfigurable Device は、要素プロセッサの機能と配線構造が可変なオンチップマルチプロセッサ、あるいは可変構造シストリックアレイとも考えられ、SIMD、データパラレル、VLSI アルゴリズムを容易に実装することができる。すなわち、いまや Reconfigurable Device とオンチップマルチプロセッサとの境界は限りなくアナログ的になりつつある。

また、動的再構成機能を利用することにより、今まで実現が困難であった仮想ハードウェア[13]の実現が期待される。

さらには、動的適応型ハードウェアの実現も可能である。動的適応型ハードウェアは、利用状況に応じて、自律的に動的に構成を変えることで、性能、電力消費を改善するハードウェアであり、その理想とする所は、設計者が細々とした指定をしなくても、状況に応じて、最適な構成にチューニングされていく。

一般的な再構成可能なデバイスでは構成の入れ替えに要する時間と電力消費が大きいた

め、動的再構成の利点が帳消しになってしまう場合が多い。しかし、DRP のように多数のコンテキストをチップ内で保持し、1 クロックで入れ替えが可能であれば、チューニングのコストが小さいため、適応の効果が上回る場合が多くなる。筆者らは DRP 上に動的適応型のスイッチを実現する研究を進めている[14]。

これは、到着するパケットが少ない場合、スイッチ中のクロスバが、より単純な構成に変更され、場合によっては配線のみになってしまうスイッチで、限定された条件で性能向上が確認されている。

一方で、ここで紹介した粗粒度 Reconfigurable Device は粗粒度にしたが故の弱点を抱え込んでいる。すなわち、細粒度の要素から構成される FPGA、CPLD と比べて、アプリケーションが演算器間のデータフロー上にうまく載らない場合、実現が極めて難しくなる点である。すなわち、今まで Reconfigurable System が得意としたパターンマッチング、検索処理、短いデータ幅での特殊な演算処理の効率が悪化する可能性がある。これを防ぐためには制御処理を行なう部分と演算アレイから成るデータパスの組合わせを工夫した構成にする必要がある。

5. おわりに

この委員会の報告書も今年度で最後となるとのことだが、1998 年から 2001 年までの報告書を読み返すと、かなり正確に現状が予測されていることがわかる。これらの報告書が示す通り、今後、Reconfigurable Device は、VLSI アルゴリズムやマルチプロセッサアーキテクチャで培われた並列処理技術との組み合わせが重要になっていくであろう。

DRP を開発した NEC、DNA チップを開発した IP Flex および富士通、PCA を開発中の NTT など、この分野での日本企業の貢献は大きく、研究面でも日本の大学、企業から、多くの業績が挙げられている。しかし、やはり本格的商用化の段階で米国ベンチャー企業の進出に押されつつあるのが現状である。

1998 年の段階で、報告書で提言したように、国を中心とした大型プロジェクトが実行されていれば、状況はかなり違ったものになったであろう。この点既に機を逸した感があるが、この分野は、今後、アーキテクチャ、ソフトウェア、VLSI チップ、CAD 等を巻き込んで大きく発展するだろう。

参考文献

[1] "Special Issue on Configurable System," IEEE Computer, April 2000.

[2] H.Amano, Y.Shiata: Reconfigurable Systems: New activities in Asia," Proc. on FPL2000 (LNCS1896), pp.585-595.

[3] http://www.chameleonsystems.com/

[4] M. Motomura: "A Dynamically Reconfigurable Processor Architecture," Microprocessor Forum, Oct. 2002.

[5] http://www.qstech.com/

[6] http://www.ipflex.com

[7] 天野英晴: "直接解法マシン(Custom Computing Machines): 打倒プログラム格納型計算機への最後の希望," ペタフロップスマシン技術に関する調査研究 II (日本情報処理開発協会 先端情報技術研究所編, 1998.

[8] 天野英晴: "実用化が進むリコンフィギャブルコンピューティング," ハイエンドコンピューティング技術に関する調査研究 II (日本情報処理開発協会 先端情報技術研究所編, 2001.

[9] http://www.pactcorp.com

[10] http://www.ece.cmu.edu/research/piperench

[11] 山田ほか, "マルチコンテキストリコンフィギャブルデバイス上でのデータドリブンアプリケーションの実装,"　FPGA/PLD　Conference and Exhibit. 2003.

[12] 出口ほか, "DRP でのウェーブレットフィルタの実装," 信学報 CPSY　2003 年 1 月

[13] X.P.Ling, H.Amano: "WASMII:A Data Driven Computer on a Virtual Hardware," Proc. of FCCM pp33-42, 1993.

[14] 天野 : "マルチコンテキストデバイスを用いた動的適応型ハードウェアの提案," 情処アーキテクチャ研報 150-12、2002 年 11 月

3.2.2 CPU の高速化とアプリケーション性能の高速化

久門 耕一 委員

1. はじめに

CPU の周波数、トランジスタ数はムーアの法則にほぼ従い向上してきた。しかし、計算機の性能という意味では、必ずしも上記の法則にしたがった向上が得られているとは言えない。良く知られた原因として、メモリアクセス時間がクロック周波数の向上に見合うだけ短縮していないことが挙げられている。しかし、これ以外にも、高周波数を達成するための長大なパイプライン構造が原因となった、非通常処理時のパイプラインブレークによるペナルティの増加が挙げられる。この状況は、HPC アプリケーションではなくビジネス向のサーバ処理に頻繁に出現するため、ビジネスサーバの性能向上に大きな障害となる。

このレポートでは、Linux オペレーティングシステムを使ったサーバシステムにおける計算機アーキテクチャの持つ役割について述べる。

2. CPU 性能の向上とそのペナルティ

1970 年代にインテルの Gordon Moor が提唱したいわゆるムーアの法則は、この法則をひとつの目標として努力が行われてきたこともあり、その後 30 年近く経た現在でも成り立ってきた。CPU の性能を向上させるには単に CPU のクロック周波数を向上させるだけではなく、処理するデータを処理に見合う速度で提供することが重要であるが、当時から 1000 倍近く CPU のクロック速度の向上が得られているのに対し、メモリ速度の向上は 100 倍程度に留まっているため、メモリアクセスがシステムのボトルネックになっていることは良く知られている[1]。

この解決のために、CPU クロックと同期して動く高速なメモリを用いたキャッシュメモリが導入され、主記憶へのアクセス頻度を低減させ、透過的にメモリアクセス時間の短縮が図られてきた。しかし、CPU 速度とミスマッチを起こしているのは、主記憶へのアクセス速度だけではない。複数の CPU を使用したマルチプロセッサシステムにおいては、CPU 間の通信がある CPU が書き換えたデータを他の CPU が読み出すことにより引き起こされる並列キャッシュ間のコヒーレンスミスを発生させるため、キャッシュサイズを無限に増大させてもキャッシュミスをなくすことは出来ない。また主記憶外にあるデータ、たとえばディスク上のデータへのアクセス、ネットワークから流入するデータに関しても、キャッシュへのデータ読み込みが CPU からのアクセスによるオンデマンド読み込みを行う限りキャッシュミスを引き起こす。

このようなキャッシュミスに起因する性能劣化以外に、近年の急激な CPU クロックの向上を達成するためのパイプラインの多段化、命令レベル並列度を引き出すためのスーパースカラのための内部状態の複雑化が原因となって、パイプラインの乱れが引き起こす大幅な性能劣化が急激に大きくなってきた。次章でインテルの 32 ビットプロセッサを例に取りサーバアプリケーションが近年の CPU でどのような性能となるのかを示しながら問題を分析する。

3. 各種インテルプロセッサによるアプリケーションの性能

キャッシュミス時のメモリアクセス時間を隠蔽する重要な手段として、キャッシュ中に命令が実際にデータを必要とする前に準備しておくプリフェッチがある。インテルの P4 マイクロアーキテクチャを採用するプロセッサは、規則的なメモリアクセスパターンをハードウェアにより検出し、先行してメモリアクセスを発行するハードウェアプリフェッチ機構を備えている。図 1 は、P4 アーキテクチャを採用する Xeon プロセッサ上で、流体系シミュレーションプログラムのベンチマークである姫野ベンチマーク[2]を実行した場合のハードウェアプリフェッチ効果をキャッシュミス率として表したものである。

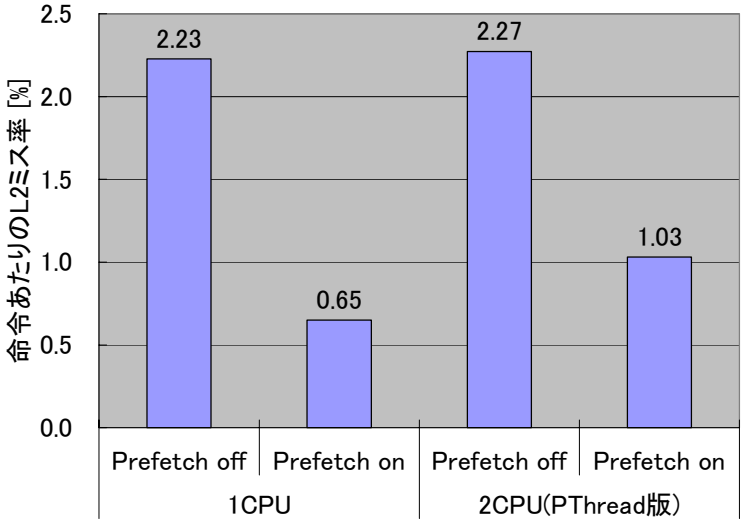


図 1　姫野ベンチマークによるメモリプリフェッチの効果

プリフェッチを行うことによりキャッシュミス率はプリフェッチを行わない場合の約 1/3 に減少している。この結果、プリフェッチによりアプリケーション性能は 1CPU 時に約 2 倍、2CPU 時に 1.5 倍向上する。プリフェッチの効果をサーバアプリケーションにおいて評価するため、ZivDavis 社の提供する WebBenchmark を実行した場合の PrefetchOn/off によるサーバ性能の変化を図 2 に示す。姫野ベンチの場合とは異なり、プ

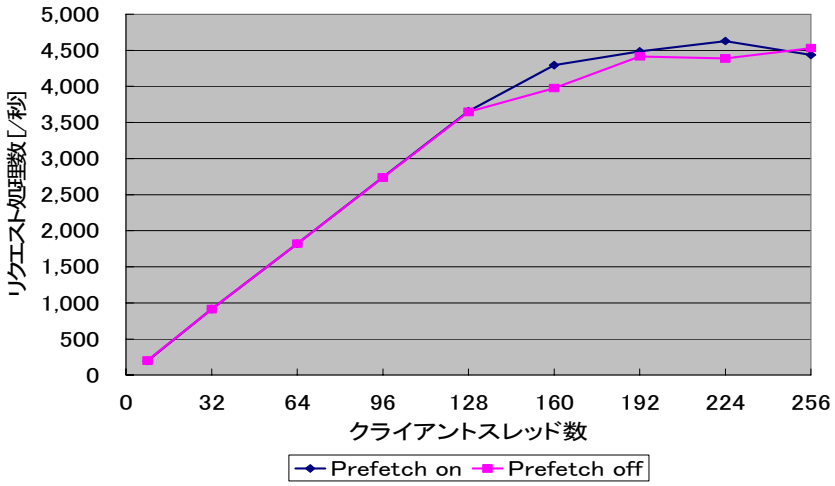


図 2　WebBehcmark 実行中の User･OS 命令の実行に必要な平均クロック数（CPI）

リフェッチの効果はほとんど得られず、最大でも 5%程度の性能向上にとどまっている。これは、HPC 系のプログラムのキャッシュミスは、規則的なメモリアクセスに基づくものが多いため、ハードウェアプリフェッチが可能であるのに対し、Web サーバのようなサーバプログラムにおけるキャッシュミスは、アクセスパターンが不規則なためプリフェッチの効果が期待しにくいことが主な原因である。

更に、サーバ系のアプリケーションプログラムの実行特性が HPC 系のプログラムと大きく異なることとして、ディスク、ネットワークへのアクセスが多いため、プログラムの実行時間に占める OS 実行時間の割合が大きいことが上げられる。このように、一般に OS 中の命令実行は User 部分の命令とはかなり異なった特性を示す。

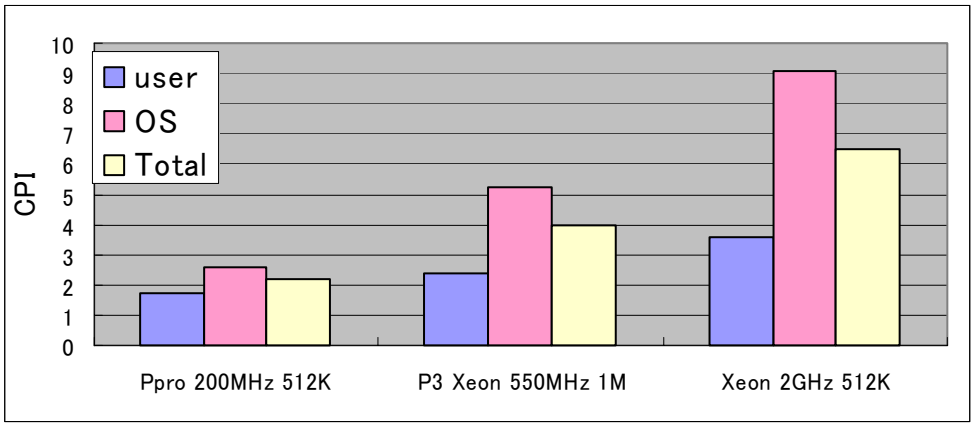


図 3　WebBench 実行時の OS/User 毎の CPI

図 3 は前述の WebBenchmark を実行した時の平均命令実行時間を、OS 部分、User 部

分、総合と分けて示したものである。実行に使用した CPU は、PentiumPro 200MHz（L2Cache 512KB）、Pentium-lll 550MHz（L2Cache 1MB）、Xeon 2GHz（L2Cache 512KB）の 3 種類である。これらの CPU はいずれもスーパースカラ構造を持っており、理想的には 1 クロックあたり 3 命令の実行が可能である。PentiumPro と Pentium-lll はほぼ同じ P6 マイクロアーキテクチャで、パイプラインステージは 10 段程度であるのに対し、Xeon は P4 マイクロアーキテクチャで、20 段以上のパイプラインステージに細分化することにより、高周波数化を達成している。これらいずれの CPU でも CPI が 1 を超えており、1 クロックあたりの平均命令実行数は 0.5～0.1 程度と、3 に比較し大変に少ないことがわかる。更に、User 部分の CPI と OS 部分の CPI を比較すると OS 部分のほうが大きく、新しい世代の CPU になるとその差が広がっていることも分かる。

User 部分の CPI は 200MH z のプロセッサと 2GHz のプロセッサで CPI は約 2 倍で、10 倍のクロック速度で命令は 5 倍高速に実行されていることが分かる。一方、OS 部分の CPI は 4 倍弱となっており、10 倍のクロックであっても実際の命令実行速度は 2.5 倍程度しか高速化されない。

このように、プログラム全体を平均した CPI は OS 部分の CPI の増加と共に大きくなっており、プロセッサが新しくなるほど OS 実行時間の割合が増加することが分かる。WebBench のようなサーバアプリケーションでは OS 実行時間の割合が比較的高いため、OS 部分が User 部分に比べ高速化しにくいことは、処理全体の高速化にとってのボトルネックになる。

4. CPU 性能と OS 実行時の性能

OS での CPI 増加が大きいことを検証するため、Linux カーネルのシステムコールの中でも単純な処理である gettimeofday()の実行時間を計測したのが図 4 である。

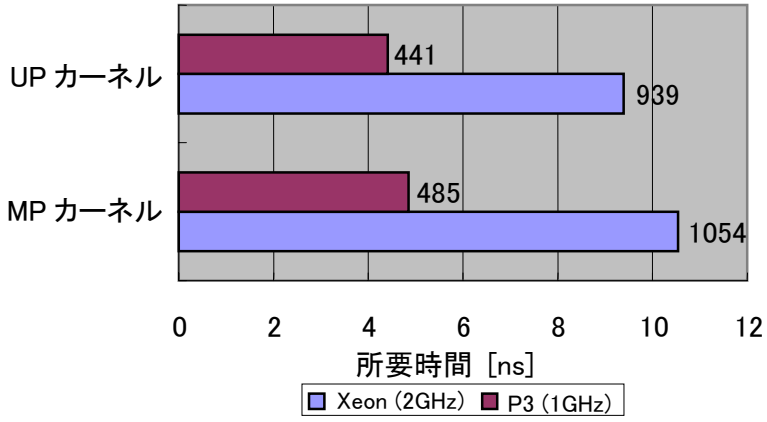


図 4　Gettimeofday()処理の実行時間

2GHz の Xeon と 1GHz の Pentium-lll では、クロックが高速な Xeon のほうが 2 倍の実行時間が掛かっている。それぞれの実行に必要なクロック数は、Pentium-lll が 441 クロック、Xeon では 1872 クロック必要で、Xeon は 4 倍以上のクロック数が掛かる。詳細な命令レベルの実行時間を分析することにより、これらの時間の大半は OS を呼び出すための動作モード切り替えによりパイプラインが乱されることと、内部状態の退避、復帰のために掛かることが分かっている。

また、図 4 のマルチプロセッサ用の Linux カーネルと単一 CPU 用のカーネルの実行時間の比較から、マルチプロセッサカーネルを使用した場合には、それぞれの実行に必要なクロック数は、それぞれ 44 クロックと 236 クロック増加する。これは、複数プロセッサ間での競合を避けるためにスピンロックを行うコードが追加されるためで、1 プロセッサ上での実行ではロック競合は起きないにもかかわらずこれだけのクロックが使用され、Xeon のほうがスピンロックコードの実行に多くのクロック数が掛かることが分かる。この増加は、ロック変数へのアクセスが命令実行のシリアライズを伴う Lock プリフィクス付きメモリアクセスでパイプライン実行が乱されたために発生している。

5. 今後の CPU の行方

近年のコンピュータシステムの性能向上は、CPU の周波数向上と、メモリ速度の見かけ上の高速化を行うキャッシュメモリの増大により達成されてきた。パーソナルコンピュータの高性能化はマルチメディア処理を高速に行うために使用され、ビデオ編集などのマルチメディア用アプリケーションプログラムは HPC アプリケーションと同等の計算手法を用いており、最近の高速 CPU はビジネスアプリケーションよりも HPC アプリケーションの実行に向いた CPU ということが出来る。

HPC 向けの高性能プロセッサであるベクトルプロセッサは CPU からメモリまでを長いパイプラインとして扱うことにより、メモリアクセスレイテンシをパイプライン処理の中に埋め込んできた。P6 マイクロアーキテクチャから P4 マイクロアーキテクチャに替わり、CPU の周波数が 10 倍以上高速化されてきたが、メモリアクセスレイテンシはほとんど変わらずほぼ 200ns で短縮されてこなかったが、P4 のハードウェアプリフェッチ機構は IA32 命令互換を保ちながらベクトルプロセッサと同様のメモリアクセス時間隠蔽手段をとることで、HPC アプリケーションを高速化したと考えることが出来る。

一方、その代償として長いパイプラインステージと複雑な内部状態の管理からパイプラインが乱されたときのペナルティはクロック速度の向上を帳消しにし、性能低下が起きるほど大きくなっている。小さなトランザクションが多発するサーバアプリケーションにとっては、プリフェッチによるレイテンシ隠蔽よりはキャッシュの増大による実効アクセス

レイテンシの短縮、さらには主記憶アクセスレイテンシを実際に短くする直接的な手段、またパイプラインの多段化よりは短パイプラインによる実行レイテンシの短縮が重要であることを示している。

今後、マルチメディア向け高性能プロセッサと、サーバ向け高性能プロセッサは 2 分化してゆくと考えられる。しかし、マルチメディアアプリケーションが今後も進展することは間違いが無く、その意味で現在の MPU を用いた PC クラスタによる HPC コンピューティングは当面高速化が続くことが予想される。これに対し、ビジネスサーバ向けの CPU の方向性は現時点ではあまり明確とはいえない状況である。

参考文献

[1] Himeno Benchmark http://w3cic.riken.go.jp/HPC/HimenoBMT/

[2] “Hitting the Memory Wall: Implications of the Obvious”, Wm. A. Wulf , et al. ftp://ftp.cs.virginia.edu/pub/techreports/CS-94-48.ps.Z

3.2.3 高性能プロセッサの高速シミュレータ

中島 浩 委員

1. はじめに

高性能プロセッサをはじめとする高性能システムの研究・開発に、シミュレータが不可欠のツールであることはいうまでもない。本稿の著者は平成 12 年度の報告書[1]において、高性能システムのシミュレーション技術の動向を概観し、その中でプロセッサの複雑化への対応が重要な課題であることを示した。すなわち、スーパスカラーなど複雑化した命令実行機構を忠実にシミュレーションすると、実機との実行時間比である slowdown (SD) が 5,000~10,000 という非常に大きな値となり、アーキテクチャの迅速な評価に支障をもたらしていることを指摘した。

また[1]では、この性能上の問題を解決するアプローチとして、シミュレーションの精度を一定の範囲で犠牲にしたうえで、処理を簡略化することによって高速化する手法をいくつか紹介した。これらはいずれも精度と性能がトレードオフの関係にあり、たとえば Rice 大学の DirecRSIM[2]は誤差が 1~2%と小さいが SD は 1000 以上と大きく、FastILP[3]は SD が 100 程度と高速であるが誤差が 10%と大きい。

一方、最近になって精度をまったく落とさない cycle accurate なシミュレーションを、計算再利用の技術を用いて高速化する研究が注目されている。その一つが Wisconsin 大学のグループが開発している FastSim[4,5,6]であり、また著者らが開発中の TurboScalar（仮称）[7]も同様のアプローチを採用している。本稿ではこれら二つのシミュレータを概観し、計算再利用技術がシミュレーションの高速化に有用であることを示す。

2. 高性能プロセッサのアーキテクチャシミュレーションの課題

アーキテクチャレベルあるいは命令レベルのシミュレータは古くから研究されており、命令の動作だけをシミュレートするものであれば 10~100 という非常に小さな SD を達成しているものが数多く知られている。また単一パイプラインなど、命令実行に要するサイクル数が比較的単純に定まるものについては、キャッシュや MMU を含めたシミュレータであっても同程度の SD が達成されている[1,8]。

一方、スーパスカラーのように命令実行機構が複雑である場合、ある命令の実行完了に要するサイクル数は他の多くの命令の動作に依存するため、ワークロードの実行サイクルを精密にシミュレートする cycle accurate なシミュレーションには、大きな計算量を要する。すなわち、命令パイプラインの各ステージなどのハードウェア機構の動作を詳細にシ

ミュレートする必要があり、必然的に SD が大きな値となってしまう。

たとえば代表的なスーパスカラーシミュレータである SimpleScalar[9]は、分岐予測を含む命令フェッチ、renaming register を含む命令発行機構、ロード・ストアバッファ、キャッシュ、TLB を含むメモリアクセス機構、複数サイクルの遅延を持つ演算機構、リオーダバッファなど、命令実行のための種々のハードウェア機構の動作を忠実にシミュレートすることができる。またこれらの機構の構成パラメータを自由に設定できるほか、研究者・設計者が独自の機構を C 言語でコーディングすることもできる。このように SimpleScalar はマイクロアーキテクチャを柔軟かつ詳細に記述できるという優れた特徴を持っているが、1 サイクルのシミュレーションに多くの機構の処理が含まれるため、SD が 5,000~10,000 という大きな値となってしまう。

このほか、SimOS [10]の動的スケジューリングモードや RSIM[11]など、命令の動的スケジューリングを精密にシミュレートするシミュレータの SD は、いずれも 5,000~10,000 という大きな値である。

3. 命令スケジューリング計算における計算再利用

前節で述べたように、スーパスカラーなど高度で複雑な命令実行機構を持つプロセッサのシミュレータには、ハードウェア機構の動作、特に命令スケジューリングの処理に大きな計算量が必要なため、命令動作のシミュレーションの 100 倍もの時間を要するという問題がある。一方、プログラムや命令の実行過程には繰り返しによる時間的局所性があり、キャッシュや分岐予測機構など局所性を利用した機構が数多く知られている。

計算再利用の技術もその一つであり、ある計算の入力が過去の同じ計算の入力に一致すれば、計算を省略して過去の計算結果を出力することにより、計算を高速化する手法である。たとえば関数の入力引数と関数内で参照する状態変数の値を記憶しておき、同じ関数が再び起動された場合に入力引数・状態変数の値が一致すれば、やはり記憶しておいた関数の返値、出力引数の値および状態変数の更新値を用いて、計算を省略して関数を実行する手法などが知られている[12]。

この手法は、再利用可能な場合には計算を大幅に高速化できるが、再利用ができない場合には入力値の比較と入出力値の保存というオーバヘッドを伴う投機的なものであるため、その有効性は再利用可否の確率とそれを左右する比較・保存対象の選択に強く依存する。すなわち多数の入出力値を保存しておけば再利用確率は高くできるが、保存や比較のオーバヘッドは必然的に大きな値となり、結果的に性能が向上しない（あるいは低下する）ことが十分に考えられる。したがって再利用されやすいもの、すなわち関数の例では実行頻度が高い関数の出現頻度が高い入力値を選択的に保存し、比較・保存のオーバヘッドを極

力小さくしつつ再利用確率を高く保つことが重要である。

さて前述のように命令の実行過程には時間的局所性があるため、シミュレータにおける命令スケジューリング計算に計算再利用技術を適用するのは妥当なアプローチであると考えられる。すなわちループを形成する命令列の実行過程において、命令スケジューリングの結果が一つあるいは少数のパターンとなることは十分予想される。したがって入力である命令列と命令実行機構の状態と、出力であるスケジューリング結果を保存しておけば、同一の命令列・状態に対してスケジューリング計算を完全に省略することができる。

たとえば図 1 に示す簡単なループを、2 命令同時実行の仮想的なスーパスカラーで実行したときの命令パイプラインの状態が、図 2 の(a)と(b)を繰り返すものとする。すなわちループが一定回数回転すると、パイプラインには$i-1$回目の BNEZ と i 回目の LW、および i 回目の ADD と SUB が、それぞれペアとなって実行される状態が継続するものとする。このとき入力命令列として同じものが供給される限り、すなわちループが継続しかつ LW がキャッシュヒットする限り、スケジューリング計算の結果は(a)から(b)あるいは(b)から(a)への遷移となる。したがって、この二つの状態とその間の遷移をもたらす命令列を保存・比較すれば、スケジューリング計算を完全に省略することができる。

```
L:                          do {
  LW    Rx,a(Ri)              x=a[i];
  ADD   Rs,Rs,Rx              s=s+x;
  SUBI  Ri,Ri,#1            } while (--i != 0);
  BNEZ  Ri,L
```

図 1 ループ命令列の例

decode	execute	commit
BNEZ(i-1)	ADD(i-1)	BNEZ(i-2)
LW(i)	SUBI(i-1)	LW(i-1)
ADD(i)		

(a)

decode	execute	commit
ADD(i)	BNEZ(i-1)	ADD(i-1)
SUB(i)	LW(i)	SUBI(i-1)
BNEZ(i)		

(b)

図 2 仮想的スーパスカラーのパイプライン状態

4. FastSim

FastSim[4,5]は Wisconsin 大学で開発されているシミュレータであり、fast-forwarding と呼ぶ計算再利用技術を特徴としている。FastSim は図 3 のように構成されており、ターゲットマシンの実行形式①を fs と呼ぶツールによりシミュレータ用の実行形式②に変換し、それを直接実行することによって命令の動作をシミュレートする。このシミュレーションは命令の論理的な挙動、すなわち演算、ロード・ストア、分岐などの操作とそれに伴

うレジスタやメモリの更新のみを対象とする。キャッシュの動作や命令スケジューリングなどの物理的挙動は、実行されたロード、ストア、条件分岐の情報を保持するキュー③を経由して駆動されるキャッシュ／メモリシミュレータとマイクロアーキテクチャシミュレータ④によりシミュレートされる。すなわち分岐命令の情報によりスケジュールすべき命令列が明らかになり、それらの中で遅延が一定ではないロード・ストアをキャッシュ／メモリシミュレータでシミュレートすれば、命令スケジューリングに必要な情報を全て入手することができる。

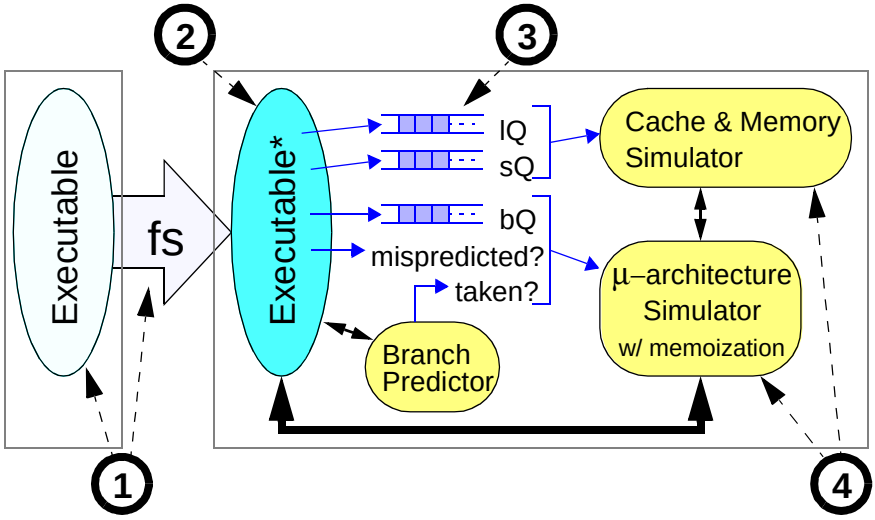


図 3　FastSim の構成[5]

FastSim の特徴である fast-forwarding は、マイクロアーキテクチャシミュレータの内部状態であるスケジュールされた命令群の状態を記憶しておき、それらの状態間の遷移をロード・ストアと分岐命令の挙動によって決定するという、一種の有限状態機械を動的に構築することにより行う。状態を記憶する領域である memoization cache は初期的には空であり、ロード・ストアあるいは分岐命令をスケジューリングするたびに内部状態が保存される。その際、すでに保存されている状態と等しい状態であれば、前状態から現状態へのリンクがロード・ストアの遅延や分岐先情報とともに形成される（図 4）。したがって記憶済の状態にあるときに、遅延が記憶済のロード・ストア、あるいは分岐先が記憶済の分岐命令をスケジューリングしようとすると、実際のスケジューリング計算は行わずに単に次状態へ遷移することによってシミュレーションが行われる。

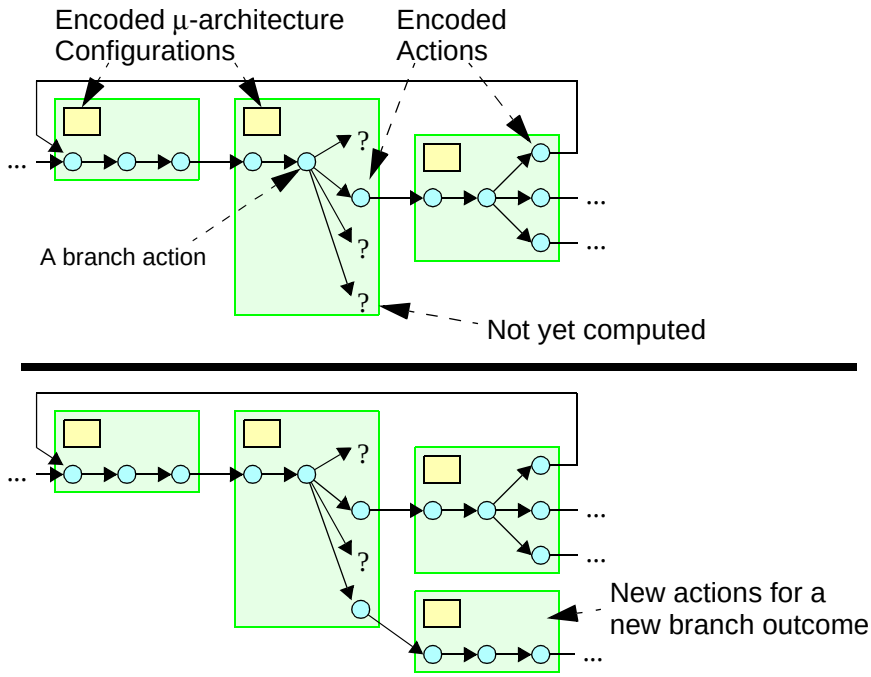


図 4　fast-forwarding [5]

この fast-forwarding はきわめて効果的であり、シミュレータを 5～12 倍に加速すること、また SimpleScalar に比べて 9～15 倍の性能が得られることが報告されている。一方、問題点としては、挙動が異なる全てのロード・ストアと分岐命令に対して内部状態を記憶するため、膨大な記憶領域が必要であることがあげられる。SPEC95 ベンチマークを用いた性能評価によれば、3MB~900MB の記憶領域が消費されており、明らかに過大なメモリ消費である。またこの評価に用いられたターゲットマシン UltraSPARC は同時発行命令数や命令ウィンドウが小さく、ループ中での内部状態が安定しやすい傾向があり、より高度なプロセッサではさらに大きな記憶領域が必要であると予想される。

もう一つの問題点は FastSim の構成方式が特異であるため、一般の研究者・設計者がシミュレータを構築することが困難なことである。この問題に対して Wisconsin 大学のグループは Fasile というマシン記述言語を用意することによって解決しようと試みている [5,6]。この言語では、ターゲットマシンの ISA や命令スケジューラの機構に加え、memoization のために保存・比較すべき情報を記述できるようになっている。したがってシミュレータの構築はある程度容易になると思われるが、現時点では SimpleScalar の 2 倍程度の性能しか得られておらず、Fasile での記述の最適化が課題となっている。

5.　TurboScalar

TuboScalar（仮称）は本稿の著者らが開発中のシミュレータであり[7]、SimpleScalar に計算再利用の機構を組み込むことにより、マシン記述の容易性と高速化を同時に達成することを目的としている。TurboScalar と FastSim の最大の相違点は、計算再利用の対象をループにおける命令スケジューリング計算に限定していることである。ループ中では同一の命令スケジューリングが繰り返し出現することは容易に予想でき、またループであるので実行頻度も当然高いものとなる。したがって FastSim のように膨大な記憶領域を必要とすることなく、かつ計算再利用の効果を十分に発揮できるものと予想される。

TurboScalar は図 5 に示すように、先行実行、詳細実行、高速実行の 3 つの命令シミュレータから構成されている。先行実行は SimpleScalar の sim-safe をベースとしたもので、命令のエミュレーションを行って命令の論理的挙動をシミュレートするほか、詳細実行における命令スケジューリングに必要な情報である分岐命令とロード・ストアの情報を記録する。またループの検出も同時に行われ、詳細実行から高速実行への切り替え候補として記録される。

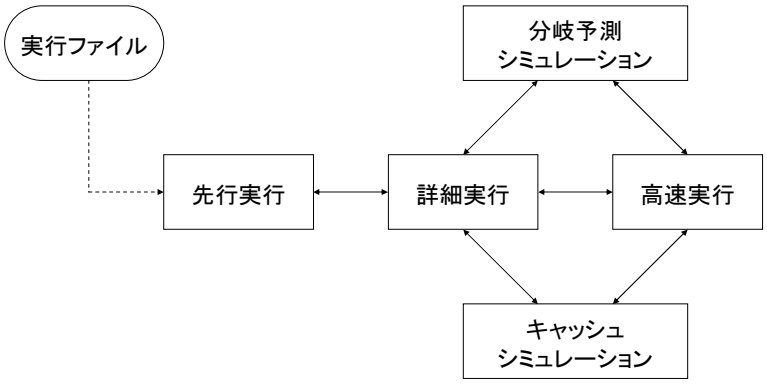


図 5　TurboScalar の構成[7]

詳細実行は SimpleScalar の sim-outorder をベースとし、命令スケジューリングを行うとともに、ループが回転するたびにスケジューリング状態の保存と比較を行う。またループ中で行われたロード・ストアの遅延も保存する。このループ回転時の状態が過去の状態と一致すると、前状態から現状態へのリンクがロード・ストア遅延とともに形成され、高速実行へ移行する。高速実行ではループが継続し、かつロード・ストア遅延が過去に形成されたリンクの値と一致している間、スケジューリング計算を省略して状態遷移のみを行

う。

上記のように TurboScalar ではループ回転時の状態のみを保存・比較するため、必要な記憶領域や保存・比較のオーバヘッドは FastSim に比べて格段に小さい。また予備的な評価によれば、計算再利用をループに限定しても効果は十分に発揮され、SimpleScalar の 10 倍程度の高速化が達成できる見込みである。なおシミュレータ構築の手間については、ベースシミュレータが広く利用されているものであるため、計算再利用のための API を提供することにより比較的容易であると考えられる。

参考文献

[1] 中島浩：ハイエンドコンピュータ研究のためのシミュレーション技術．（HECC-WG の報告書），pp. 117--125, 日本情報処理開発協会, March 2001.

[2] M. Durbhakula, V. S. Pai, and S. Adve. Improving the Accuracy vs. Speed Tradeoff for Simulating Shared-Memory Multiprocessors with ILP Processors. In Proc. 5th IEEE Symp. High-Performance Computer Architecture, January 1999.

[3] D. J. Sorin, V. S. Pai, S. V. Adve, M. K. Vernon, and D. A. Wood. Analytic Evaluation of Shared-Memory Systems with ILP Processors. In Proc. 25th Intl. Symp. Computer Architecture, June 1998.

[4] Eric Schnarr and James Larus. Fast Out-Of-Order Processor Simulation Using Memoization. In Proc. ASPLOS-VIII, October 1998.

[5] Eric Schnarr. Applying Programming Language Implementation Techniques To Processor Simulation. Ph.D. Dissertation, University of Wisconsin-Madison, Computer Sciences Department, 2000.

[6] Eric Schnarr, Mark Hill, and James Larus. Facile: A Language and Compiler For High-Performance Processor Simulators. In Proc. PLDI 2001, June 2001.

[7] 中田尚、大野和彦、中島浩：高性能マイクロプロセッサの高速シミュレーションの構想．情報処理学会計算機アーキテクチャ研究会, 149-27, August 2002.

[8] D. Burger and T. M. Austin. The SimpleScalar Tool Set, Version 2.0. http://www.cs.wisc.edu/ mscalar/simplescalar.html.

[9] R. A. Uhlig and T. N. Mudge. Trace-Driven Memory Simulation: A Survey. ACM Computing Surveys, Vol. 29, No. 2, pp. 128--170, June 1997.

[10] M. Rosenblum, S. A. Herrod, E. Witchel, and A. Gupta. Complete Computer

System Simulation: The SimOS Approach. IEEE Parallel & Distributed Technology, Vol. 3, No. 4, pp. 34--43, 1995.

[11] V. S. Pai, P. Ranganathan, and S. V. Adve. RSIM: An Execution-Driven Simulator for ILP-Based Shared-Memory Multiprocessors and Uniprocessors. In Proc. WS. Computer Architecture Education, February 1997.

[12] 中島康彦、緒方勝也、正西申悟、五島正裕、森眞一郎、北村俊明、富田眞治：関数値再利用および並列事前実行による高速化技術. 情報処理学会論文誌ハイパフォーマンスコンピューティング、Vol. 43, No. SIG 6, pp. 1-12, September 2002.

3.2.4 大規模アプリケーションと大規模計算に望ましいプロセッサ
― 計算機アーキテクチャに対する素朴な疑問 ―

福井 義成 委員

1. はじめに

計算機の歴史、アプリケーションの開発・保守、高速化などの立場から見た現在の計算機への素朴な疑問について述べる。計算機が開発されて以来、計算機の歴史は高速化の歴史でもあった。計算機を高速化する手段はいくつか考えられる。思いつくまま挙げてみると以下のようなものがある。

(1) 素子の高速化
(2) メモリの大容量化
(3) キャッシュメモリ（データ、命令）
(4) 命令バッファの使用
(5) メモリのインタリーブ
(6) ループを効果的に行う命令を用意する
(7) ベクトル命令
(8) パイプライン化
(9) （並列処理）

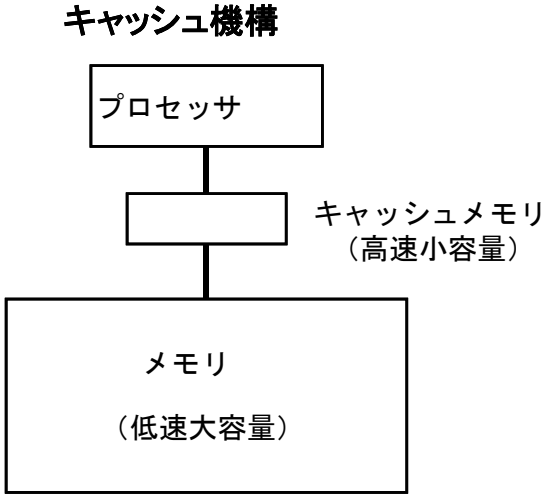
この中で、すべての問題（プログラム）に満遍なく効果があるのは(1)素子の高速化だけである。その他の項目は、何らかの「仮定」を満足した場合のみに高速化の効果があるものばかりである。その様子を以下に述べる。

(1) 素子の高速化
すべての問題に効果がある。

(2) メモリの大容量化
メモリが大きいと速く計算できる場合がある。メモリ不足で外部記憶を使用していたのを、すべてメモリ上で処理できる場合等。不必要に大きくしても効果はない。

(3) キャッシュメモリ
メモリ参照が局所的であれば効果がある。メモリ参照が局所的でないプ

ログラムには効果が少ない。

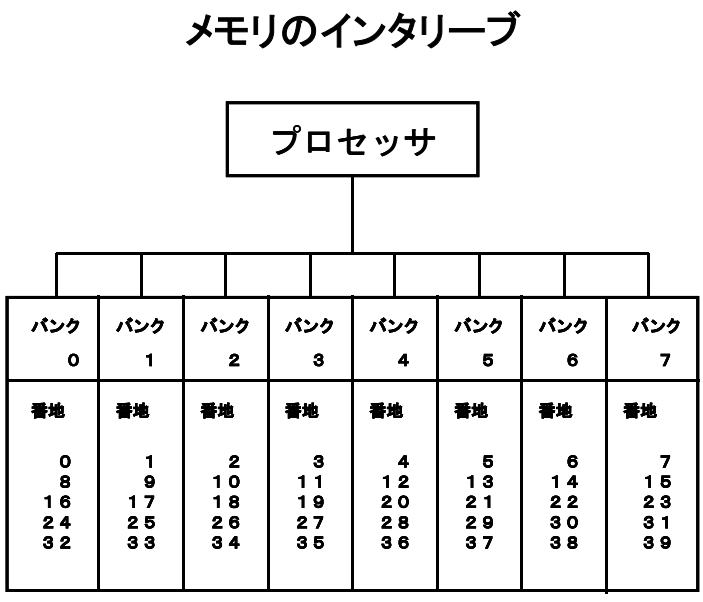


(4)　命令バッファの使用（命令キャッシュ）

ループが命令バッファ内に入りきると命令のためのメモリへの参照が減り、高速化が可能になる。長いループや分岐が多いとダメ。

(5)　メモリのインタリーブ

メモリ参照を分散すると高速にデータを扱える。同じメモリバンクを連続的に使用すると効果がない。



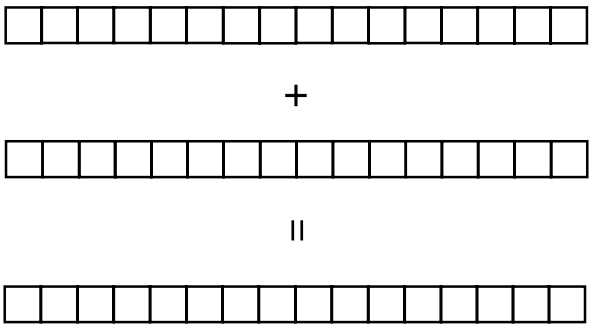
(6) ループを効果的に行う命令を用意する

ループを速く実行できる。その命令にあてはまらないと効果がない。

(7) ベクトル命令

ループを効果的に行う命令を発展させたものであり、ベクトル化できると10倍以上速くなる。ベクトル化できないと効果がない。

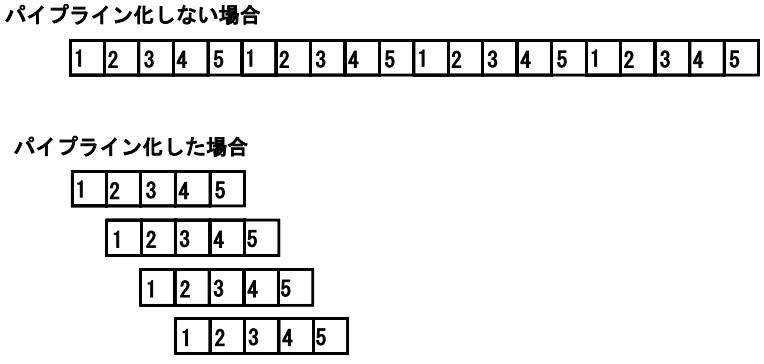
ベクトル機構



(8) パイプライン化

1つの演算を複数に分けて処理し、時間的な並列性を上げることにより、高速化できるが、同じ種類の演算がなければ効果がない。

パイプライン化



以上のように、(1)以外の項目は、各々の「仮定」を満足しなければ、高速化の効果が得られない。逆に言えば、(2)～(8)については、問題（プログラム）の性質により、高速化の効果が異なると言える。(9)については別の機会に考える。

2. キャッシュについての疑問

キャッシュ機構は、IBM360/81 から商用機に採用されたと思われる。現在ではキャッシュ機構はプロセッサとメモリの間の速度差を埋める（隠蔽する）手法と考えられている。しかし、プロセッサとメモリの間の速度差を隠蔽する手法は、キャッシュ機構だけではない。ベクトル機構もプロセッサとメモリの間の速度差を隠蔽する手段である。スカラープロセッサでも、キャッシュ機構ではなく、大量のレジスタを準備する方法等も考えられる。その場合の問題点はレジスタを指定するための、ビット数と大量のレジスタを効率良く利用する為のコンパイラ技術であろう。

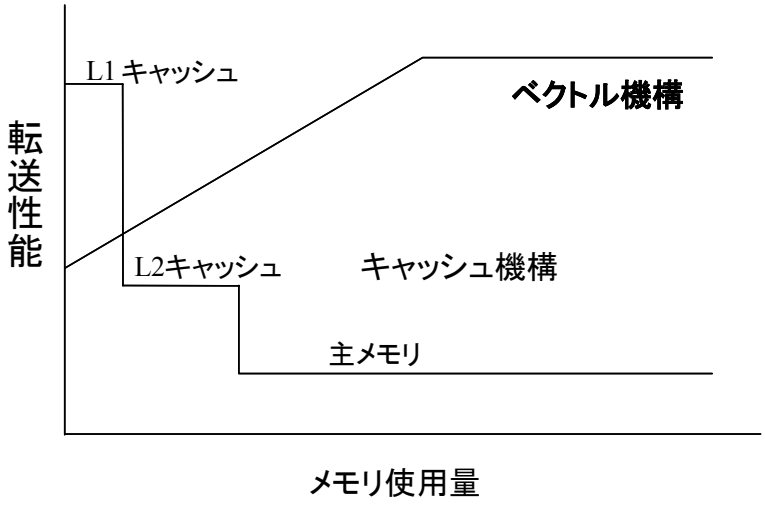
筆者の目にはキャッシュ機構は、ハードウェアの余裕が出来たが、アーキテクチャを変更出来なくなった状況での妥協策と映る。アーキテクチャが変更できる状況で命令セットを変更し、レジスタ数を増加させる方向も可能だからである。レジスタ数を増加させる方向をとった場合、問題となるのは、ビット数と大量のレジスタを効率良く利用する為のコンパイラ技術である。コンパイラの最適化能力不足により、せっかくの多数のレジスタのアロケーションができないことも起きる可能性がある。しかし、まったく自動的にコンパイラが、レジスタのアロケーションを行うのではなく、何らかの指示をユーザが行うことで、レジスタを効果的に使うことができない状況を回避することも可能である。

キャッシュ機構は、高速のプロセッサと低速のメモリの間にユーザがコントロールできない高速のメモリを配置し、頻繁に使用するデータを高速のメモリに置いて、高速処理を図ろうとする方法である。使用するデータがキャッシュ容量を越えた場合、最も使われていないデータを削除し、その後に新たなデータを移動することが行われている。このアルゴリズムは一見良さそうであるが、最も使われていないデータ（最も過去に使用されたデータ）は、削除された直後に使われることもある。行列形式のデータを左上から右下に演算をしていく場合を考えると、右下まで演算が終了した後は、また左上の要素を演算するのが自然である。この場合、「最も使われていないデータ（最も過去に使用されたデータ）」を削除し、新たなデータを移動する方式は、常にデータの移動を発生させる恐れがある。常にデータの移動ばかりを行って、キャッシュ機構のメリットがなくなる。これを防ぐには、キャッシュ機構の動作を考慮したプログラムが必要になり、これは本末転倒である。この原因はキャッシュ機構が「投機的」な方式であることに由来する。ユーザのデータの使い方とキャッシュ機構の前提としているデータの使用方法が一致している場合はキャッシュ機構が有効に働くが、一致しない場合は有効に働かないどころか、キャッシュ機構が無い場合以上に遅くなることさえある。

ユーザの多くは自分の計算におけるデータの動きを理解しており、それに従ったプログラムを作成する。キャッシュ機構がある場合は、さらにキャッシュ機構の動きを意識した

プログラムを作成する必要がある。「キャッシュ機構の動き」が常に同じものであればまだしも、キャッシュサイズやキャッシュ機構のアルゴリズムが代わると「自分の解きたい問題は変わらない」にも関わらず、プログラムの修正を行わないと高速な計算ができないことになる。ユーザの立場からは、自分の意志で計算機をコントロールしたいのに、キャッシュ機構に阻まれて自由にならないこともある。キャッシュ機構は、リソースに余裕ができたが、アーキテクチャを変更できない場合の妥協策である。動きが投機的で、プログラマの自由にならない。その結果、チューニングの効果がキャッシュサイズに依存し、その効果が計算モデルに依存する。

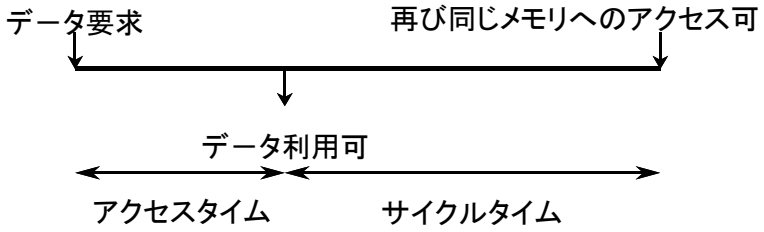
キャッシュ機構とベクトル機構の模式図



3. メモリの遅さの隠蔽方法

現在の計算機においては、プロセッサの速度とメモリの速度に大きな差があり、その差が毎年広がりつつある。プロセッサとメモリの速度差が大きい環境で、如何に効率良く計算を行うかを考える。

DRAMの動作



プロセッサとメモリの速度差を隠蔽する手法の１つはキャッシュ機構であるが、キャッシュ機構は、キャッシュ機構のアルゴリズムと計算におけるデータの動きが異なった場合、ユーザに不要な努力を強いることになる。ユーザが自覚しているデータの動きを、素直に反映させることが困難な場合がある。自分でデータの動きを理解しているユーザにとって、計算機が自分の思うように動いてくれないことは歯がゆいことである。科学・技術計算の場合、データの動きは比較的単純である。特に、偏微分方程式を解く場合、多数のメッシュに分割し、計算を行うので、決まった形の繰り返し演算になり、他の分野よりデータの動きが予見しやすい。

ベクトル計算機も、プロセッサとメモリの速度差を隠蔽する手法の１つである。ベクトル計算機では、以下の効果が得られる。

(1)　ベクトルレジスタを利用したメモリの遅さの隠蔽
(2)　１組の命令で多数の演算をすることによる命令処理の低減
　　－インストラクション・バッファの負荷低減

一般的には、キャッシュメモリとベクトルレジスタは異なる性質のものと考えられている。はたして、キャッシュメモリとベクトルレジスタは異なる役割のものであろうか？見方を変えるとキャッシュメモリとベクトルレジスタは同じ目的のためのものである。現在の計算機では、プロセッサの速度とメモリの速度に大きな差がある。キャッシュメモリとベクトルレジスタは、このプロセッサとメモリの速度差を隠蔽するという点でまったく同じ役割を果たしている。

キャッシュメモリは、メインメモリとの間でアドレスが連続なメモリを決まったサイズ（キャッシュライン：32 バイトなど）で転送する。データの転送アルゴリズムはハードウェアで決まっており、プログラマが直接関与することはできない。ベクトルレジスタは、メモリからプロセッサへまとまったデータをブロック転送する点は同じである。ベクトルレジスタは、キャッシュと同じように、アドレスが連続なメモリを転送することもできる。違いは決まったサイズではなく、転送サイズを変更できる点にある。また、リストベクトル（英語では scatter／gather）を使用すれば、連続アドレスのメモリでなくとも転送可能な点も異なる。また、データ転送をプログラマの意志で行うことができる。

	キャッシュ	ベクトル
転送サイズ	固定	任意
メモリの分布	連続領域	連続領域／任意の領域
転送アルゴリズム	プログラマが支配できない	プログラマが支配できる

このように見るとキャッシュメモリはベクトルレジスタのサブセットになっている。

キャッシュメモリの利用は投機的実行と呼ばれることがある。キャッシュメモリはメモリを連続するブロック（キャッシュライン）に分け、メモリとの間でブロック単位で転送される。キャッシュへ転送されたメモリは必ずしもすべて使われるとは限らない。全て使われる場合もあれば、最悪の場合、ブロックの中の１記憶単位（4 バイトや 8 バイトなど）しか使われないこともある。そのため、投機的実行と呼ばれる。キャッシュメモリはプロセッサとメモリの速度差を隠蔽するための手法であるが､使われるデータがプログラム(主としてループ）を実行する前に確実に分かっていれば、事前にデータをメモリから転送する作業を行っておき、プロセッサでデータが必要になった時にレジスタ等に準備されていれば、プロセッサとメモリの速度差を隠蔽することができる。最近のプロセッサでは事前にデータをレジスタに転送する命令（プリフェッチ命令）を持っているものもある。プリフェッチ命令はデータが確実に使われることが分かっていてデータをあらかじめレジスタへもってきておく命令であるため、投資的実行と呼ばれる。解釈を広くすれば、ベクトルレジスタへのデータのロード命令も同じであると考えられる。

このように、プロセッサとメモリの速度差を隠蔽する方式は、キャッシュ機構だけではない。ベクトル方式もプロセッサとメモリの速度差を隠蔽する手段であり、他にも色々考えることは可能である。たとえば、ベクトル方式でなくても、高速な記憶装置（レジスタ等）をプロセッサの近くに大量に配置することによっても、プロセッサとメモリの速度差を隠蔽することは可能になる。

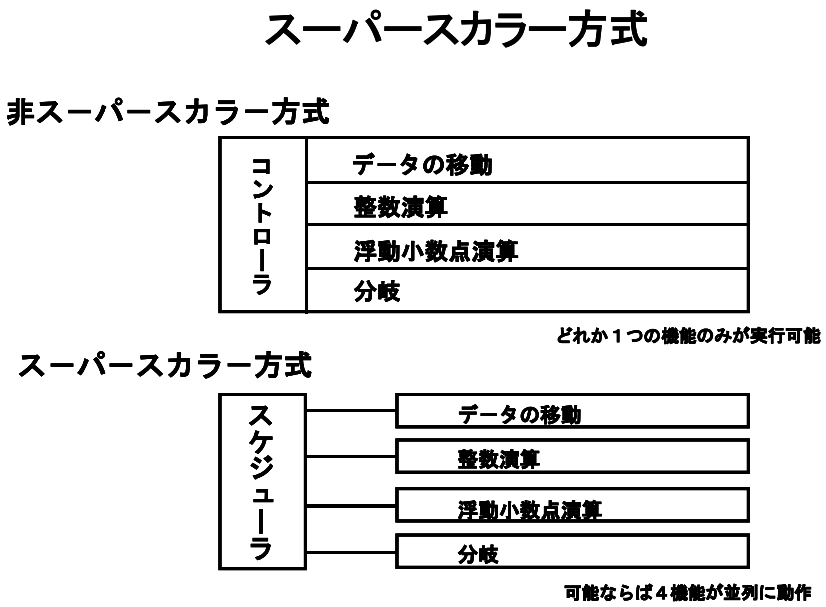
4. プロセッサの動向からキャッシュメモリの今後を推定

並列化以外の最近のプロセッサの動向を考えると、スカラープロセッサでは、

- パイプライン
- スーパースカラー
- VLIW

等が行われてきた。この流れを見ると、スーパースカラーまでは、プロセッサ内の演算順序の変更、並列化等はすべてハードウェアが行ってきた。しかし、VLIW からは、ハードウェアが行ってきた並列動作の制御等を、コンパイラを中心としたソフトウェアに依存する方向になってきたと考えることができる。この延長線上には、ソフトウェアコントロールのキャッシュ機構が来ると考えることもできる。ベクトル機構も、見方を変えるとソフ

トウェアコントロールのキャッシュと考えることができる。



5. 温故知新

最近、マルチスレッドをサポートするプロセッサが出現している。良く知られているのは、インテルプロセッサのハイパースレッドであろう。10 数年前には、バートン・スミスの MTA が出現している。ハイパースレッドや MTA は、CDC6600 の pp を思い出させる。pp は 1 つの周辺処理装置を時分割で 10 個の周辺処理装置として見せるようになっていた。速度の遅さを複数の「スレッド」で隠すという観点から同じ概念と言える。実際、バートン・スミスに MTA は CDC6600 の pp を思い出させると話したとたんに、彼は、CDC6600 の pp のパワーポイントを出してきた。

また、メモリについては、HITAC-401 という大昔の計算機が、最初のアドレス（4,096 まで?）がコアメモリでその後は、ドラムであったそうである（残念ながら、実際に見たことはないが)。

このように計算機を速くする努力は昔から行われており、根本的な点では、共通することが多い。これからの計算機を考える上でも、参考になることが多い。

6. 軽いベクトル機構

現在のベクトル型計算機は重厚長大なものと考えている人が多い。現在存在するベクトル型計算機の特色について考えて見る。

(1) プロセッサへのデータ供給能力が大きい。これを実現するために、ベクトル計算機のコストが高くなっている。
(2) ベクトルレジスタというプロセッサに近い高速メモリをもっている。しかし、その量はキャッシュメモリよりは小さい。
(3) ベクトル命令では 1 回に扱えるデータが大きい。ベクトル計算機ではベクトル化がうまく出きると FLOPS 値は上がるが MIPS 値は低下する。すなわち、ベクトル命令で一度に多くのデータを処理できるため、実行命令数は少なくて良い。
(4) ベクトルレジスタのデータはユーザの意思で再利用が可能。

(1)はベクトル計算機のコスト高の原因である。しかし、ベクトル計算機のメリットはベクトルレジスタを使うことにより、メモリの遅さを隠蔽する効果を期待するのであれば、この部分はコストが多大にならない範囲で妥協した「軽い」ベクトル計算機があっても良いのではないか？CRAY には擬似ベクトルというベクトルレジスタの使い方があった。ベクトル化はできない計算を、データだけベクトルレジスタに入れ、計算はスカラーでやる方式であった。これでもかなりの効果があった。計算のベクトル化よりも、データ転送のベクトル化の方が効果があった。ベクトル機能をメモリの遅さを隠蔽させることに中心を置いた方式を考える。

また、実行する命令数が減少するのも、メモリへのアクセスが減り、高速化への効果もある。

7. ユーザにとって望ましい計算機

計算機のユーザにとって望ましい計算機について考えてみる。以下のような要件が必要である。重複する項目もあるが、色々な角度から考える。

(1) 正しい結果を与えてくれる。
(2) 高速化を行ったプログラムの資産が生きる。
(3) プログラマの意思が尊重される。
(4) 高速化のコストが小さい。
(5) 半導体技術が進歩した場合、自然にその効果を享受できる。

このような用件を満たす計算機について考えてみる。(1)については自明である。(2)については、キャッシュ機構はメモリの容量が変わることによって高速化の効果が変わってしまうため、十分な性能を発揮させるためには高速化の変更が必要になる。この条件に反す

る。(3)はキャッシュ機構がプログラマのやりたいことを阻むことを懸念している。これは、高速化のコストにも反映する。

現在のプロセッサは、ほとんどが、1 つの LSI で作製することができるようになっている。半導体技術が進歩すると、1 つの LSI の中に入れることができるゲート数が増加する。そのような場合に、自然に増えたゲート数の効果を享受できることが望ましい。あまり大きな変更をしなくても、ゲート数の増加の効果で、自然に速くなる方式であって欲しい。すなわち、時間が経つと 1 つの LSI に入れることができるゲート数が増加することを前提にしたアーキテクチャであるべきである。現在は増加したゲート数が安易にキャッシュメモリの量を増やすことに使われている。それは、計算機が変わるとプログラマに高速化の変更を強いることになる。

上記の用件を満たす 1 つの考え方を例として述べる。新たなプロセッサを作成するのは非常なコストが掛かるので、それも考慮し、なるべくシンプルなものを考える（リスクプロセッサの原点？）。

また、マイクロプロセッサで良く性能比較に使われる「クロック」であるが、クロックは何をベースにするかで大きく異なる。例えば、パイプライン化されたプロセッサの場合、パイプラインの段数を深くすることで見かけのクロックを速くすることは、難しいことではない。例えば、クロックが 2GHz で、乗算を 10 段のパイプラインで計算するプロセッサと 1GHz で 5 段のパイプラインで計算するプロセッサは、1 つの乗算で見る限り同じ性能である。したがって、基本的に LSI の性能を比較するにはゲートあたりの遅延を使うのが公平であろう。同じゲート遅延の LSI であっても、クリティカルパスを減らすことで、実際のプロセッサの性能を上げることができるであろう。キャッシュ機構のコヒーレンシィをとる部分は、クリティカルパスになると予想できる。したがって、キャッシュ機構を無くすことで、クリティカルパスを減らせるのではなかろうか？

8. 素人のたわごと？

以上のようなことを考慮して、1 つの計算機のアイデアを考えて見た。計算機設計の素人の発想であるので、具体的なことはともかくとして、1 つの考えかたとして見ていただきたい。対象としては、科学・技術計算のハイパフォーマンスコンピューティング（HPC）を考える。

プロセッサに比較して、メモリが遅いことは残念ながら受け入れることとする。現在のプロセッサの場合、階層的なメモリ構成になっている。

・ レジスタ
・ L1 キャッシュ
・ L2 キャッシュ
・ L3 キャッシュ
・ 主メモリ
・ ディスク等

現在のメモリでは、高速性と容量（コスト）は相反する要素である。従って、なんらかの形での階層的なメモリ構造は受け入れざるを得ない。しかし、キャッシュ機構は投機的な機構で、HPC 分野では障害が多い。そこで、場所により速度に差のあるメモリ構成を考える。

・ レジスタ
・ L1 キャッシュ
・ L2 キャッシュ
・ L3 キャッシュ
・ 主メモリ
・ ディスク

と同じ順序で速度差のあるメモリ要素を下記のように配置する。番地が小さいところほど速いメモリを配置する。

・ レジスタ
・ プロセッサ内のメモリ
・ プロセッサ外のメモリ
・ DRAM

このようにすると、LSI の集積度が上がると、各レベルの境界が移動する。そのような状況も踏まえたアーキテクチャにしたい。そのためには、上記のレジスタ、プロセッサ内のメモリ、プロセッサ外のメモリ、DRAM に連続した番地を振ることにする。ユーザは、低い番地ほど速いメモリだということを理解し、プログラムを書くことにする。これを支援するために、コンパイラ、アセンブラでデータをメモリ上のどこに配置するかというディレクティブをサポートする。このようにすれば、LSI の集積度が上がった場合でも、再

コンパイル程度で集積度の向上の恩恵を得ることができるのではないか。本当に自分でプログラムを書いているユーザは、自分が使っているデータのどれが頻繁に使われるかを理解しているはずなので、ユーザにとって負担にはならない。

多くのレジスタ、メモリを実装すると、レジスタやメモリを指定する部分（ビット数）が大きくなり、命令語長も長くなってしまう。レジスタ数は 256 個とし、8 ビットで 1 つのレジスタを指定することにする。レジスタの種類は 8 ビット、16 ビット、32 ビット、64 ビット、128 ビットとする。8 ビットは文字処理用、16 ビットは短長の整数、漢字等を想定する。32 ビット、64 ビットは整数と浮動少数点数を想定し、128 ビットは浮動少数点数を想定する。

プロセッサ内のメモリは、4M バイト、プロセッサ外の高速メモリは 64M バイトを、DRAM は 8G バイト以上を想定する。プロセッサ内のメモリにレジスタを配置する。

- 32 ビットレジスタ　　1,024 バイト
- 32 ビットレジスタ　　1,024 バイト　インデックス用
- 64 ビットレジスタ　　2,048 バイト
- 128 ビットレジスタ　　4,096 バイト

合計　8,192 バイト（8K バイト）

これらの配置の自由度を増すため、レジスタのスタート・アドレス・レジスタ、レジスタ数・レジスタを用意する。この自由度を増すための仕様は、インプリメントの際に決定することにする。

このスカラー・レジスタ群の後にベクトルレジスタを配置する。ベクトルレジスタの数は 16 個（4 ビット）とする。ベクトル・データ長レジスタを用意し、32 ビットデータ、64 ビットデータ、128 ビットデータ、256 ビットデータをサポートする。このベクトル機構は従来のような重厚長大なものを狙うのではない。長いベクトル長のデータのロード／ストアを可能にすることにより、メモリのレーテンシィを隠蔽することと、1 つの命令で多数のデータを処理することによる効果を目的とする。数%のゲートの増加で、150%の性能向上が得られれば良いとする発想である。

利用可能なゲートを如何に使うかという視点が重要であると考える。「与えられた面積（体積）に実装可能なゲートをどのように使うか」というクイズである。そのクイズの 1974 年の時点での、最高の回答が、CRAY-1 であったと思う。

異種メモリの組合は、歴史的には結構なされている。たとえば、HITAC-401 では、コアメモリとドラムが使われていたようである。4,096 語以下は、コアメモリで、それより

上はドラムであったようである。また、CDC7600 や FACOM230-75 では、通常のメモリ(SCM)と LCM（FORTRAN の COMMON だけで使えるメモリ）があった。CRAY-2 ではローカル・メモリ（16 ビット・アドレッシング：SRAM）とコモン・メモリ（32 ビット・アドレッシング：DRAM）という構成であった。

この方式は、タスク切り替えに向いていない。しかし、HPC 分野でひたすら計算をすることを目的としている。このような例は CDC6600 で、セントラル・プロセッサは演算だけに専念し、割り込み処理も行わなかった。割り込み処理や入出力処理は、ペリフェラル・プロセッサ(pp)が行っていた。HPC 分野を狙うプロセッサであるため、設計コストが小さいことが重要であろう。

9. まとめ

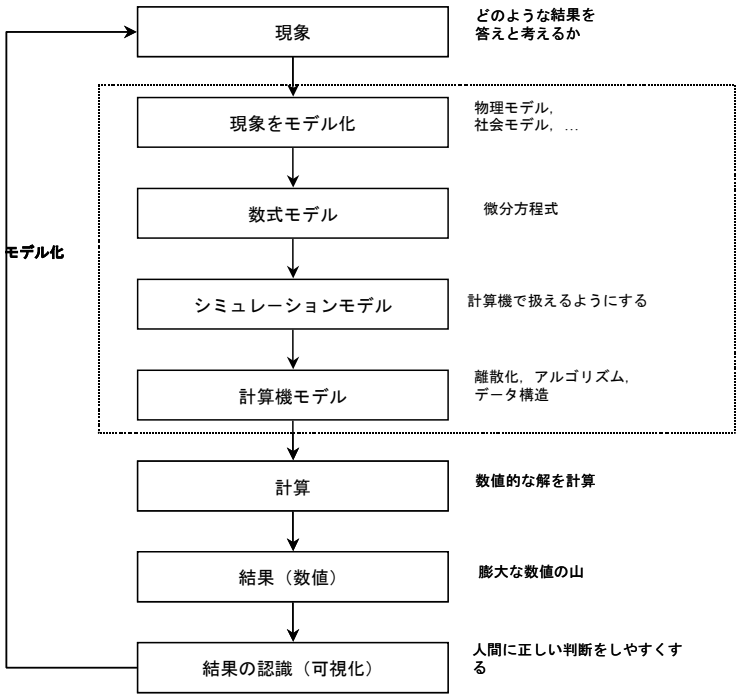
計算機設計には素人である一人のユーザのたわごとについて述べてきた。前項で述べた計算機は、計算機設計の専門家からは、問題だらけであろう。前項に述べた計算機の具体的な形ではなく、その後ろにある考え方を今後の計算機の参考にしていただければ幸いである。参考にしていただきたい考え方は以下の点である。

- HPC 分野ではプログラマが自由にならない機構は避けて欲しい。
  キャッシュのような投機的な機構ではなく、投資的機構が望ましい。
- 同じことの別表現ではあるが、どこが速いメモリか分かっていれば、ユーザは使い分ける。
- 半導体技術の進歩したときに、その恩恵が自然な形でユーザに得られるアーキテクチャであって欲しい。

10. おまけ

高性能の計算機を使いこなすためには、対象となる問題の記述も非常に重要である。この部分のユーザの負荷を軽減し、マクロなレベルで記述することにより、高速化も容易になる。

残念ながら、現在のプログラムの作成は、殆どが FORTRAN や C のような手続き型言語で行われている。しかし、本当のエンドユーザにとっては、大規模プログラムの作成には負担が大きい。これを回避するためには、問題向き言語のような、人間の思考に近いモデルの記述が重要である。問題向き言語の例としては、CSMP-Ⅲ（連続系シミュレーション言語：制御系向け言語）や DEQSOL（偏微分方程式向け問題向き言語）等がある。



シミュレーションの手順

振動子モデルを問題向き言語で記述した例

$$m\frac{d^2x}{dt^2} + c\frac{dx}{dt} + kx = 0$$

X=INTGRL(X0,V)

V=INTGRL(V0,A)

A=F/M

F=-C*V-K*X

3.2.5 ソフトウェア可制御オンチップメモリを用いた高性能・低消費電力プロセッサ

中村 宏 講師

1. はじめに

近年、計算機システムの消費電力、あるいは消費エネルギー削減の要求が高まっている。例えば、モバイル計算機におけるバッテリ駆動時間延長の要求はもちろんのこと、ハイパフォーマンスコンピューティング(HPC)分野においても、大規模な並列計算機を構成する上で、放熱や設置面積などの面から消費電力を可能な限り小さく抑えることが重要となっている。今後、消費電力が全体の性能を制限する要素の１つになると考えるため、低消費電力化・低消費エネルギー化は今日の計算機システムにとって非常に重要な課題である。

著者らはこれまで、HPC 分野やメディア系のアプリケーションなどをターゲットとしたプロセッサアーキテクチャ SCIMA(Software Controlled Integrated Memory Architecture)を提案してきた[1]。SCIMA はチップ上に実装するメモリとして、従来のキャッシュに加えてアドレス指定可能な主記憶の一部(以降では、そのメモリを SCM(Software Controlled Memory と呼ぶ)を SRAM として搭載するアーキテクチャである。さらに、キャッシュと SCM でハードウェアを共有し、アプリケーションの性質に応じてそれらの容量比を動的に変更可変な実装(キャッシュ・SCM 統合機構と呼ぶ)についても提案している。

SCM は、ISA(命令セットアーキテクチャ)上に定義された命令により主記憶との間でデータ転送が行われる。従って、ユーザ(コンパイラ)による明示的なデータのアロケーション、およびリプレースメントが可能となる。キャッシュではこれらの制御がハードウェアによって決められたアルゴリズムで自動的に行われるため、個々のプログラムにとって最適なデータアロケーション、リプレースメントを行わせることは難しい。そのため、コンフリクトミスによりプロセッサと主記憶間のデータ転送(メモリトラフィック)が増加し、性能が大きく低下することがある。一方、SCIMA ではソフトウェア制御の SCM を用いることで、メモリトラフィックを最小限に抑えることが可能となる。この特徴により、SCIMA では高性能を達成できることがわかっている[1][2]。一方、消費電力の面においても、オフチップメモリトラフィックの削減は効果が大きいと考えられる。負荷容量の高い外部のメモリバスや I/O パッドを駆動するために、多くの電力が消費されるからである。SCIMA ではまた、ロード/ストア命令、すなわちレジスタ・SCM 間データ転送のための SCM アクセスの際にも、従来のキャッシュアクセスに比べ消費電力を削減できる[3]。従来の連想キャッシュでは、アクセスすべきデータは高々1 つのウェイにしか存在しないにも関わら

ず、キャッシュアクセス時間の増加を防ぐために、タグの検索と同時に全てのウェイを並列にアクセスする。このため、該当するデータが存在しないウェイでは無駄に電力が消費されている。しかし、SCM では、キャッシュとは異なりアクセスすべき位置がアドレスから一意に決定されるため、アクセスすべきデータが存在するウェイのみを選択的にアクセスすることができる。

本稿では、ソフトウェア制御のオンチップメモリを用いる SCIMA のアーキテクチャの概要、オンチップメモリを用いた性能向上手法を述べ、その後で、オフチップメモリトラフィック削減による低消費電力化に加え、選択的ウェイアクセスによる低消費電力化についても評価を行ない、NAS Parallel Benchmarks、および計算物理学の実アプリケーションを用いて、従来のメモリ階層を持つプロセッサに対する、性能と消費電力面での SCIMA の優位性を評価・報告する。

2. SCIMA

SCIMA の構成を図 1 に示す。SCIMA はプロセッサ上にキャッシュだけでなく、アドレス指定可能な SCM (Software Controlled Memory)を搭載する。従来のキャッシュでは、データのアロケーションやリプレースメントがハードウェアで暗黙的に制御されるのに対し、SCM はそれらの制御をソフトウェアで明示的に行う。

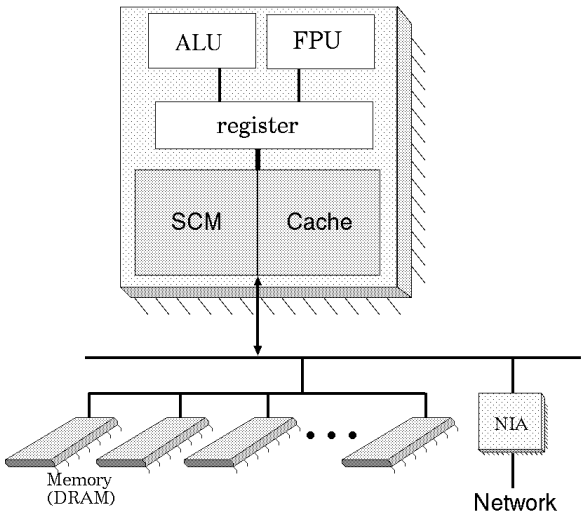


図 1　SCIMA の構成図

SCIMA では論理アドレス空間上に SCM 領域をマッピングする。SCM 領域は 1 つの大きな連続領域であるため、この管理を TLB ではなく専用レジスタで行い、TLB ミスの頻発を防ぐ。導入するレジスタは、SCM 領域の先頭アドレスを保持する ASR (On-Chip

Address Start Register)とオンチップメモリの容量を表す AMR (On-Chip Address Mask Register)である。

SCM へのデータ転送を制御するため、page-load/page-store と呼ぶ主記憶・SCM 間のデータ転送命令を追加する。本命令によるデータ転送は大きな粒度で行ない、オフチップメモリレーテンシの影響を抑えることを狙う。また、この命令にはブロックストライド転送の機能を付け加える。この機能によりオフチップメモリ上の不連続領域のデータをパッキングしてオンチップメモリに持ってくることが可能となり、プロセッサ・主記憶間のバンド幅の有効利用につながる。また、これらのデータを再利用する場合、オンチップメモリ上では連続するデータとなるため処理効率が向上する。HPC アプリケーションでは多次元配列要素に対するアクセスなどが多いために有用な機能であると考えられる。

レジスタ・SCM 間のデータ転送は従来の load/store 命令により行う。前節で述べたアドレスマッピング機構により、load/store の対象アドレスが SCM 領域か否かを判定し、SCM 領域であれば SCM へのアクセスを、そうでない場合は通常のキャッシュアクセスを行なう。

キャッシュと SCM に割り振られる容量をアプリケーションの性質に応じて変えるべく、総容量一定のもと SCM とキャッシュの容量比を実行時に再構成できる機構を提案している[2]。具体的には、n-way 連想キャッシュの連続する一部の way を SCM に割り当てる。

3. ソフトウェア可制御メモリを用いた性能最適化

本章ではまず、SCIMA の性能向上要因について整理する。また、それに基づき SCIMA の性能を最適にするためにユーザ、あるいはコンパイラが SCM をどのように用いるべきかの指針を整理・検討する。

3.1 SCIMA の性能向上要因の分析

プロセッサと主記憶の性能格差により、プロセッサは実行時間の多くを本来の計算処理にではなく主記憶からのデータ待ち、すなわち無駄なストール時間として費やしている。この実行時間を解析するために、本論文ではプロセッサの実行時間を CPU-busy time (T_b)、latency-stall (T_l)、throughput-stall (T_t)の 3 つに分類する。CPU-busy time とはプロセッサが実際に計算処理を行っている時間であり、latency-stall は主記憶のアクセスレーテンシがもたらすストール時間を、また throughput-stall はオフチップメモリのスループット不足に起因するストール時間を指す。

ここで、プロセッサの総実行時間を T、オフチップメモリスループットが無限大と仮定した場合の実行時間を T_∞、オフチップメモリスループットが無限大かつオフチップメモ

リレーテンシが 0 であると仮定した場合の実行時間を T_p とする。この T、T_∞、T_p を用い、T_b、T_l、T_t を以下のようにここでは定義する。

- $T_b = T_p$
- $T_l = T_\infty - T_p$
- $T_t = T - T_\infty$

表 1 は、SCIMA のソフトウェア可制御メモリ(SCM)の特徴が上記で分類した実行時間のそれぞれに対しどのように影響するのかを示したものである。また、キャッシュアーキテクチャにおける代表的なレーテンシ隠蔽技術についても同様に示している。表中の「p-load/p-store (大粒度転送)」、及び「p-load/p-store (ストライド転送)」はそれぞれ 2 章で述べた page-load/page-store 命令の大粒度転送、及びストライド転送の機能を表す。

表 1　SCM 利用による実行時間短縮の分析

SCMの特徴	T_b	T_l	T_t
ソフトウェア可制御性	-	-	↓
p-load/p-store(大粒度転送)	↑	↓	-
p-load/p-store(大粒度転送)	↑	↓	↓
キャッシュにおけるレーテンシ隠蔽技術	**T_b**	**T_l**	**T_t**
ラインサイズ増大	-	↓	↑
ノンブロッキングキャッシュ	-	↓	↑
キャッシュプリフェッチ	↑	↓	↑

まず、ソフトウェア制御が可能なため、データの再利用性を最大限に活用することで、オフチップメモリトラフィックを最小限に抑えられる。これは、throughput-stall の短縮につながる。また、page-load/page-store 命令の大粒度転送により、メモリアクセス回数を削減することで、latency-stall を短縮することができる。さらに、ストライド転送機能は、無駄なトラフィック及びアクセス回数を削減でき、latency-stall、throughput-stall の短縮に有効である。

キャッシュにおけるレーテンシ隠蔽技術は、latency-stall を短縮させることができるが、throughput-stall を増大させてしまう。これは、プロセッサ・主記憶間のデータトラフィックを増大させてしまうためであり、これらレーテンシ隠蔽技術がプロセッサの総実行時間短縮に常に有効であるとは限らないことを示している。

一方、SCM を用いることで latency-stall 及び throughput-stall の両者の短縮が可能となる。今後オフチップメモリのレーテンシ増大とスループット不足がより深刻化すると思

われるが、将来的に SCIMA の有効性はさらに増すと考えられる。

3.2 最適化の指針

オンチップメモリを利用するためには、オンチップメモリ経由でアクセスするデータの選択とデータ転送のタイミングを指定しなければならない。後者についてはユーザによる最適化も可能であるが、コンパイラによるスケジューリング支援は既存技術の延長として、現在でも十分可能であると考えられる。一方、アクセスするデータの選択は性能最適化のためには特に重要となるが、これまでその指針は整理されていない。そのため、本節では主に「オンチップメモリ経由でアクセスするデータの選択」について検討を行う。

オンチップメモリ経由でアクセスするデータを決定する場合、再利用性やデータアクセスの特徴に関して同じ配列内のデータは同じ性質を持つことが多いため、配列を単位として考えるのが妥当である。そこで、プログラム中での配列について再利用性とアクセスの連続性の観点からその特徴を分類する。そして、横軸に再利用性を、縦軸に連続性をとった場合に、SCM を利用する戦略をまとめたものを図 2 に示す。ここで各配列の再利用性は、``対象とする配列のある要素が参照されてから再び参照される間にアクセスされる、自身の配列内の他のデータの総数(容量)"を用いて定義する。ブロッキングなどを行なうことで、この値がオンチップメモリサイズ以下になるならば、その配列は再利用性があると判断し、そうでなければ再利用性なしと判断する。図 2 を以下に詳述する。

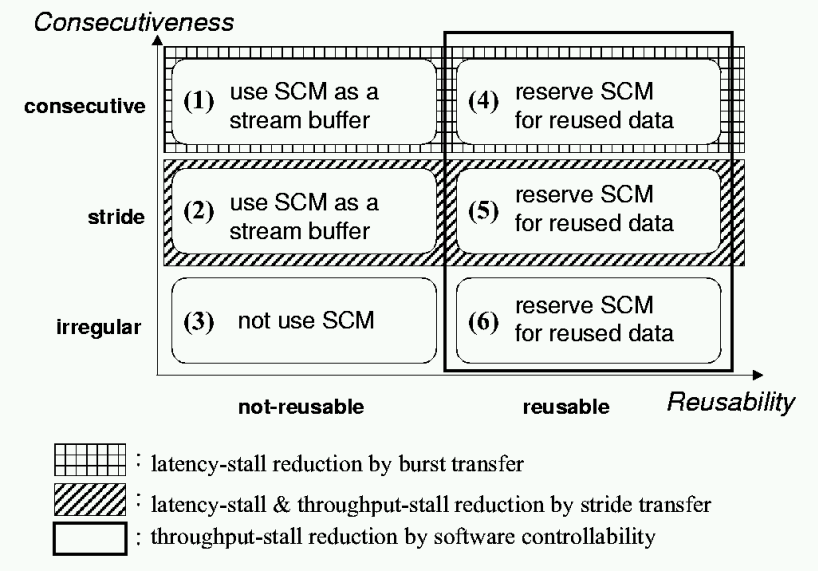


図 2 配列の特徴に対するオンチップメモリの利用法

(1) 再利用性はないが連続アクセスされる配列: page-load/page-store 命令により、転送回数を減らし latency-stall の短縮を図る。オンチップメモリ上に page サイズ分のバッファ領域を確保し、この領域をストリームバッファとして用いながらアクセスする。これは、page-load/page-store 命令の転送サイズの上限が page サイズであるためである。なお、更なる最適化として、page サイズのストリームバッファを 2 つ確保し、交互にそのバッファ領域に転送しながら演算を行なうことで、演算と転送とをオーバーラップさせることも効果的である。

(2) 再利用性はないがストライドアクセスされる配列: page-load/page-store のストライド転送の機能により、無駄なデータ転送を防ぎ効率的なデータ転送を行う。(1)と同様にオンチップメモリ上に page サイズ分のバッファ領域を確保し、この領域をストリームバッファとして用いる。

(3) 再利用性がなくアクセスが不規則な配列: このような配列は SCIMA のオンチップメモリを用いても性能向上は見込めない。そのため、SCM は利用しない。

(4) 再利用性があり連続アクセスされる配列: ブロッキング手法を用いるなど、オンチップメモリサイズに分割してアクセスすることで他のデータとの干渉を防ぎながら再利用性を最大限に活用する。また、page-load/page-store 命令による latency-stall 短縮の効果も期待される。オンチップメモリ上に、ループ中でアクセスされるワーキングセット分のオンチップメモリ領域を割り当てる。

(5) 再利用性がありストライドアクセスされる配列: page-load/page-store のストライド転送機能により、不連続なデータをパッキングしてオンチップメモリに載せ、チップ内記憶領域の有効利用を図ると同時に、他のデータとの干渉を防ぎながら再利用性を最大限に活用する。(4)と同様に、オンチップメモリ上にループ中でアクセスされるワーキングセット分のオンチップメモリ領域を割り当てる。

(6) 再利用性はあるがアクセスが不規則な配列: アクセスされる範囲が予めわかり、かつその範囲がオンチップメモリに載る程度の大きさであれば、オンチップメモリ上に固定してデータを載せ再利用性を活用する。

(1)～(6)は独立な最適化であるが、まず(1)、(2)が重要である。なぜなら、再利用性がないデータに対してはキャッシュは無力であるため、まずストリームバッファを提供して latency-stall を短縮することは効果が大きいためである。これは、HPC 分野においてベクトル計算機のベクトルロード・ストア機構が非常に有効であることからも裏付けられる。その次に、キャッシュでもある程度は効果的に扱える再利用性のあるデータに対し(4)～(6)を行い、さらなる最適化を行うのが妥当であると思われる。以上の戦略に従い、オンチッ

プメモリを用いる最適化を行なう際に、最適化の対象となる配列をオンチップメモリへ割り当てる際は、まず図 2 の(1)(2)の特徴を持つ配列に対し、最適化戦略(1)、(2)に従い page サイズ分の領域を割り当てる。次に、図 2 の(4)(5)(6)の特徴を持つ配列に対し、残りのオンチップメモリ領域にそれらの配列のすべてのワーキングセットが収まるようにブロックサイズを縮小し、収まりきるブロックサイズになった段階で、各配列のワーキングセット分の領域をオンチップメモリに割り当てる。

4. SCIMA における低消費電力化

4.1 メモリトラフィック削減による低消費電力化

SCIMA は、SCM・主記憶間のデータ転送を、page-load/page-store 命令により明示的に行うことで、SCM のデータアロケーション・リプレースメントをユーザから制御可能である。そのため、必要なデータのみを必要なタイミングで転送することができる。一方、キャッシュはハードウェア制御により決められたアルゴリズムでそれらの制御が行なわれるため、個々のプログラムに最適なデータアロケーション、リプレースメントを行わせることは難しい。例えば、ソフトウェア的な手法でデータの再利用性を向上させるキャッシュブロッキング(タイリング)[4]を適用する場合、キャッシュではラインコンフリクトによる同一配列のブロック内データの干渉(self interference)や、異なる配列間のデータの干渉(cross interference)により、オフチップメモリトラフィックが増加してしまう。SCM を利用することで、SCIMA ではキャッシュに比べ以下の点でトラフィックが削減できる。

(1) コンフリクトミスが生じないため、再利用性のあるデータが偶然に再利用性のないデータにより追い出されることがなく、従ってデータの再利用性を最大限に活用できる。
(2) 転送サイズが可変であるため、ストライド転送となる場合に、必要ないデータの転送が抑制できる。

プロセッサ・主記憶間のデータ転送では、負荷容量の高い外部のメモリバスや I/O パッドを駆動するため、消費される電力が多い。キャッシュに比べオフチップメモリトラフィックの削減が期待できる SCIMA では、低消費電力化を図ることができると期待される。

4.2 選択的ウェイアクセスによる低消費電力化

従来の N ウェイ連想キャッシュでは、キャッシュアクセスが発生すると、アクセスされたアドレスの Index 部をデコードすることで、該当するキャッシュ内のセットを決定する。次に、該当セットのタグ、およびデータアレイの全てのウェイを並列にアクセスし、それぞれのデータを読み出す。そして、読み出されたタグとアクセスされたアドレスの Tag 部分を比較し、一致するものがあればキャッシュヒットとなり、該当ウェイのデータが選

択される。このように、選択すべきデータがタグアレイの内容に依存するため、高速化のためには全てのウェイを並列に読み出す必要がある。

SCIMA のキャッシュ・SCM 統合機構を図 3 に示す。従来のキャッシュの機構に対し、SCIMA ではアクセスされたアドレスが SCM 領域に対するものかどうかの判定(SCM-test)と、その際にどのウェイのデータをアクセスすべきかの選択(way-select)を行なう回路が追加される。本機構において、SCM-test 回路でキャッシュアクセスと判定された場合は、従来のキャッシュアクセスと同様の動作となる。一方、SCM アクセスと判定された場合は、way-select 回路により決定されるウェイ内のデータが選択される。この時、アクセスすべきセットは、SCM-test の判定結果によらず、アドレスの Index 部のみで決定される。従って、選択すべきデータのウェイをキャッシュアクセス時より早い段階で決定することができ、SCM アクセス時には 1 つのウェイのみを選択的にアクセスすることができるため、消費電力の削減につながる。さらに、選択すべきデータがタグの内容に依存しないため、タグアレイへのアクセスを抑制することによる消費電力の削減も期待される。

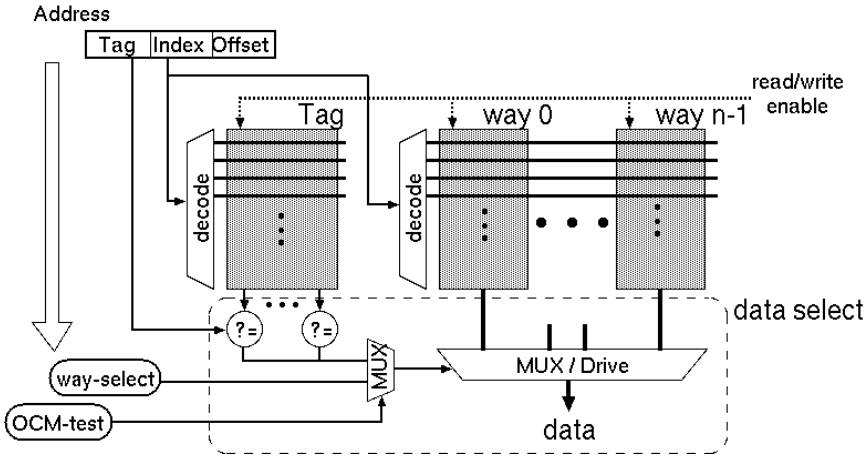


図 3　SCIMA の SCM アクセス

5. 消費エネルギー評価

5.1 評価モデル

SCIMA の低消費電力化の効果について調べるため、オフチップメモリバス、およびキャッシュ/SCM アクセスにおける消費エネルギーを評価する。なお、本稿で以降バスの消費エネルギーと表記した場合には、プロセッサや DIMM の I/O パッドで消費されるエネルギーも含まれるものとする。

本評価では、主記憶として SDRAM[5]を想定する。図 4 は評価におけるプロセッサと

主記憶のモデルを表している。ここでは、64Mx4 SDRAM デバイス(4bit 幅)を 16 個並べ、64bit 幅のメモリモジュール(DIMM)を構成する場合について考える。この場合、アドレスバスのビット幅は 15 ビットである。本稿でモデルとする SDRAM のアクセスタイミングを図 5 に示す。SDRAM では row アドレスに続いて column アドレスを発行することで、クロックに同期して連続な数データ(burst length 分のデータ)が転送される。また、burst length 以上の連続データを転送する際、row アドレスに変更がなければ、column アドレスのみを発行することで連続デするータの転送が可能と仮定する。なお、本稿では burst length を 4 として評価を行なう。

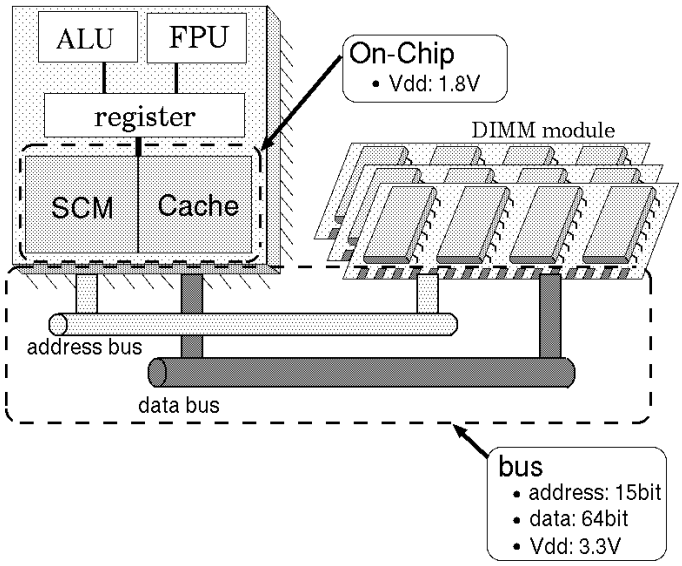


図 4 評価モデル

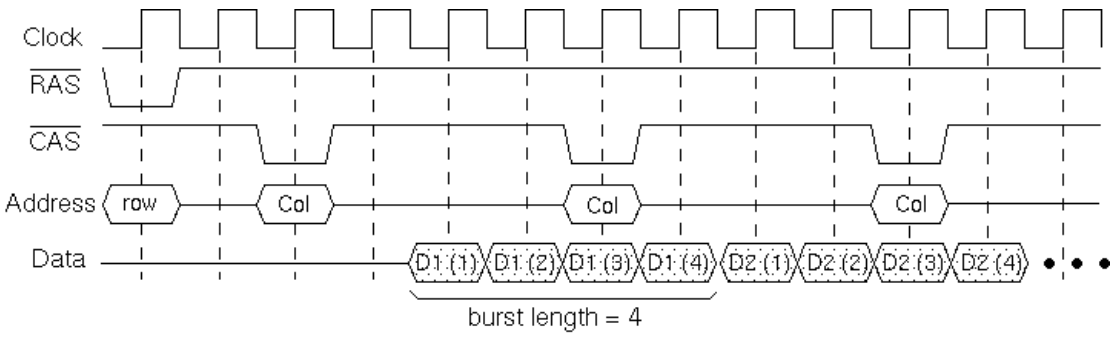


図 5 SDRAM のアクセスタイミング

5.2 評価方法

評価におけるベンチマークプログラムとして、NAS Parallel Benchmarks の中から CG、FT の 2 つのカーネル、および筑波大学の計算物理学研究センターで行われている QCD(量子色力学)計算[6]を用いる。評価では、ロード/ストア回数、メモリトラフィック、ビット遷移回数などの情報をシミュレータで採取し、消費エネルギーを求める。ここで、アドレスバスについては、実際にビット遷移回数をシミュレーションにより求める。データバスやキャッシュ/SCM アクセス時のビット遷移においては、遷移確率を 0.5 と仮定してビット遷移回数を算出した。消費エネルギーは、文献[7]で提案されたキャッシュ消費エネルギーモデルを参考にして求めた。その際、チップ内におけるキャッシュ／SCM アクセス時の負荷容量のパラメータは文献[8]のものを、またバスの負荷容量については[5]のものを採用した。

また本評価では、チップ内およびバスの駆動電圧をそれぞれ 1.8V、3.3V と仮定して評価を行った。なお、アドレスのデコードに必要な消費エネルギーやタグ比較回路の消費エネルギーは評価には含めないものとする。これは、小規模な組合わせ回路における消費電力は、SRAM アレイアクセスやオフチップメモリへのアクセスに必要な消費電力に比べて非常に小さいと言われており[7]、妥当な仮定と思われる。

表 2 に評価におけるメモリ構成の仮定を示す。キャッシュ、および SCM の構成は、合計 64KB、4-way のキャッシュを、キャッシュ・SCM 統合機構により、容量比を再構成した場合を想定する。また、データアレイの各ウェイは、内部でサブアレイに分割されるが、本稿の評価では CACIT¥cite[10]により得られた最適な分割数を用いる。

表 2　評価におけるメモリ構成の仮定

メモリサイズ	
Cacheモデル	キャッシュ:64KB(4way), SCM:0KB
SCIMAモデル	キャッシュ:16KB(1way), SCM:48KB
総ウェイ数	4way
ラインサイズ	32B　または 128B
サブアレイ分割数	行方向2分割、列方向1分割

このような条件のもと、キャッシュアーキテクチャ(Cache と表記)と SCIMA の消費エネルギーについて比較評価を行なう。まず SCIMA のトラフィックの削減による消費電力削減の効果を調べるため、バスの消費エネルギーのみを評価する。次に、キャッシュ/SCM アクセスの消費エネルギーも含め、メモリシステム全体の消費エネルギーについて比較を行う。その際、SCIMA では 4.2 節で述べた選択的ウェイアクセスを行った場合について、

またキャッシュにおいては、MRU アルゴリズムによるウェイ予測[9]を行った場合について評価を行う。

6. 評価結果

6.1 バスの消費エネルギー

まず、SCIMA によるメモリトラフィック削減の効果を見るため、図 6 に Cache と SCIMA のメモリトラフィックを示す。図 6 は、各ベンチマークプログラムにおいて、ラインサイズ 32B の場合を基準とした相対的なメモリトラフィックである。なお各棒グラフは、キャッシュミスによるトラフィック(cache miss)と、page-load/page-store によるトラフィック(page-load/store)の内訳も示している。この図から、すべてのプログラムにおいて SCIMA はキャッシュに比べオフチップメモリメモリトラフィックが削減されていることがわかる。これは、SCM を用いることで、必要なデータがコンフリクトなどにより追い出されることなく、再利用性を最大限に活用できた結果である。

次に、図 7 に各プログラムにおけるバスの相対消費エネルギーを示す。図中、*Ebus_addr* はアドレスバスの消費エネルギーを、また *Ebus_data* はデータバスの消費エネルギーを示している。図 7 から、SCIMA では Cache に比べ 1%から 61%もの消費エネルギーが減少していることがわかる。これは、トラフィックの削減に比例して、データバス(Ebus¥_data)の消費エネルギーが減少しているためである。さらに、アドレスバスの消費エネルギーが減少していることも理由の 1 つである。

アドレスバスのみの消費エネルギーを比較したものを図 8 に示す。SCIMA ではアドレスバスの消費エネルギーも大きく削減されているが、これには次の理由がある。まず、トラフィックが削減されたことにより、そもそもアドレス発行の回数が削減されたためである。また、SCIMA では大きな連続領域を一度に転送できるため、row アドレスを 1 回発行した後、column アドレスのみで連続領域の転送ができ、row アドレスの発行が抑制されたことも挙げられる。さらに、連続な column アドレスのシーケンスが発行された場合、ビットの遷移回数が少なくなることも理由の 1 つである。

表 3 に、CG における row アドレス/column アドレスの発行回数、およびアドレスバスにおけるビット遷移回数を示す。CG では Cache と SCIMA でメモリトラフィックに大きな違いはないにも関わらず、row アドレスの発行回数が大きく削減されており、またビット遷移回数も Cache の半分程度となっている。

以上のことから、SCIMA ではトラフィック削減により、実際にデータバス、アドレスバス共に消費エネルギーを削減することができると言える。

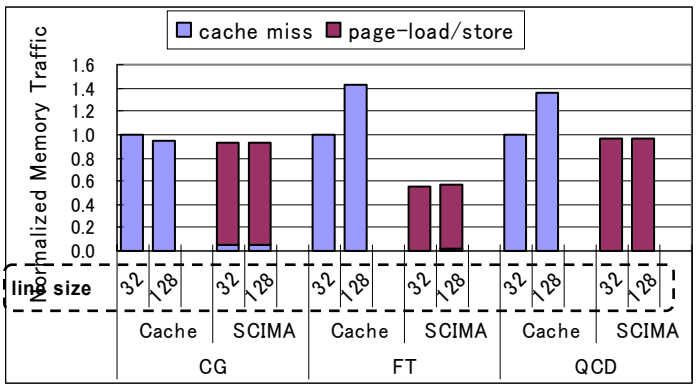


図 6 メモリトラフィック

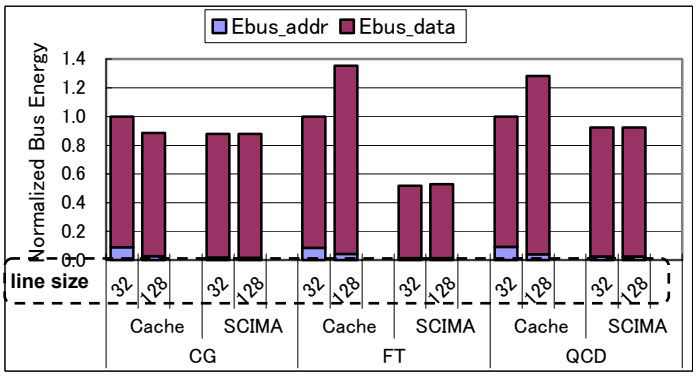


図 7　バスの消費エネルギー

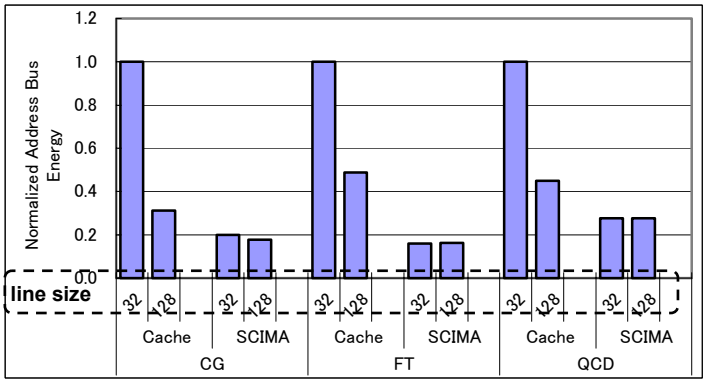


図 8　アドレスバスの消費エネルギー

表 3　CG におけるアドレスバス発行回数・ビット遷移回数

モデル	Cache		SCIMA	
ラインサイズ	32バイト	128バイト	32バイト	128バイト
rowアドレス発行回数	243932	57351	23276	14588
columnアドレス発行回数	243932	229404	230355	231288
ビット遷移回数(アドレスバス	3019750	943794	604155	534096

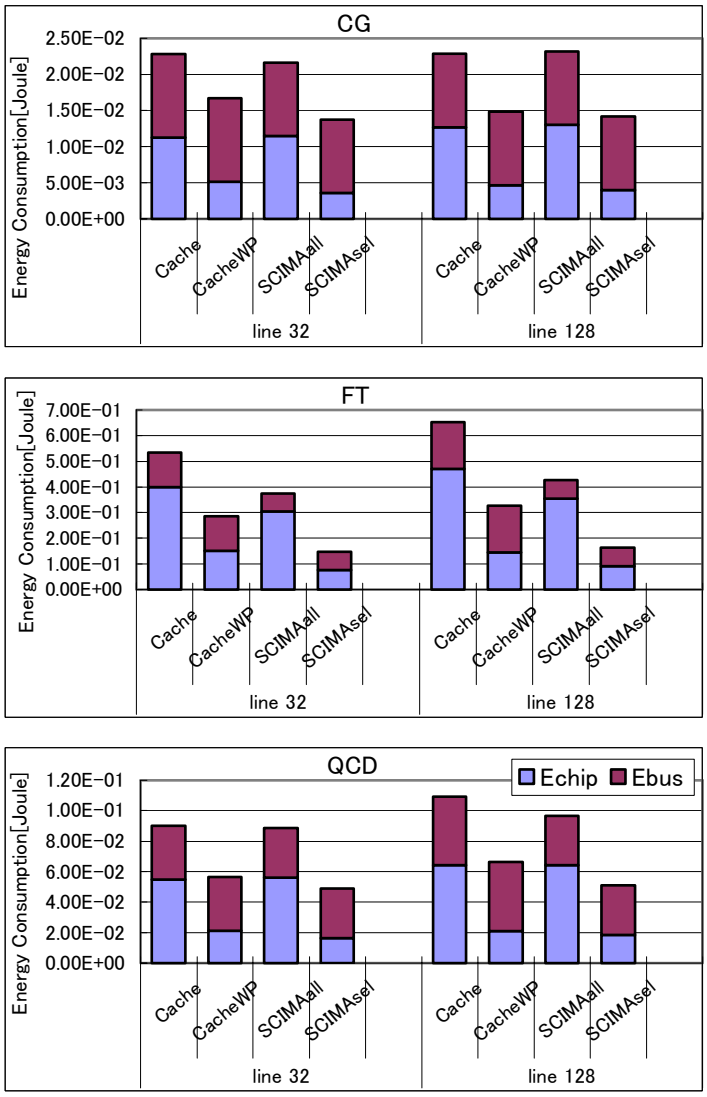


図 9　メモリシステム全体の消費エネルギー

6.2　メモリシステム全体の消費エネルギー

本節では、バスの消費エネルギーに加え、キャッシュ/SCM アクセスの際の消費エネルギーを評価し、メモリシステム全体の消費エネルギーについて検討する。

図 9 に、各ベンチマークプログラムにおける、キャッシュ/SCM アクセスにおける消費エネルギー(Echip)と、バスにおける消費エネルギー(Ebus)の合計を示す。Cache モデルにおいて、Cache と CacheWP はそれぞれウェイ予測を行なわなかった場合と行なった場合を表している。また、SCIMAall, SCIMAsel は、SCIMA モデルにおいて SCM アクセス時にも全ウェイを並列にアクセスした場合と、選択的にウェイをアクセスした場合の違いである。SCIMA では、アドレスからアクセスすべきウェイを一意に決定できるため、常に選択的ウェイアクセスが可能であるが、比較のために SCIMAall も評価した。

また、表 4 に、32B のキャッシュラインにおける両モデルのキャッシュ/SCM アクセス回数を示す。表 4 には、CacheWP モデルでのウェイ予測の際のヒット率(WP ヒット率)と、SCIMA モデルのキャッシュ/SCM アクセス回数中に SCM アクセスが占める割合(SCM アクセスの割合)も示している。

表 4　キャッシュ/SCM アクセス回数

	Cache	SCIMA
	アクセス回数（WPヒット率）	アクセス回数（SCMアクセスの割合）
CG	1686219 (69%)	1731899 (96%)
FT	65400698 (84%)	50253470 (92%)
QCD	8620112 (89%)	8806997 (55%)

まず、選択的ウェイアクセスを行なわない Cache と SCIMAall を比較する。FT を除き、キャッシュ/SCM アクセス回数はほとんど変わらないため(表 4 参照)、チップ内の消費エネルギー(Echip)に差はない。しかし、SCIMA ではバスの消費エネルギーが削減されているため、合計の消費エネルギーは SCIMA の方が少なくなっている。なお、FT においてチップ内の消費エネルギーが SCIMA で少ないのは、Cache モデルの場合にコンフリクトミス頻発による性能低下を防ぐ目的で、計算領域を一時的な scratch 配列にコピーするため、キャッシュアクセス回数が SCIMA よりも多くなるためである。

次に、全ウェイを並列にアクセスせずに、データの存在するウェイをのみ選択的にアクセスすることで、キャッシュ/SCM アクセス時の消費エネルギーを削減した CacheWP と SCIMAsel について比較する。

CacheWP では、ウェイ予測が成功した場合、今回仮定した連想度 4 のキャッシュではタグアレイアクセスの消費エネルギーは変わらないものの、データアレイアクセスの消費エネルギーが 4 分の 1 になる。今回評価したベンチマークプログラムでは、ウェイ予測のヒット率が高いため(表 4 参照)、チップ内の消費エネルギーが 60%～72%ほども削減され

ている。

また、SCIMAsel では、SCM アクセスの場合にデータアレイアクセスの消費エネルギーが約 4 分の 1 になるが、その他にタグアレイアクセスの消費エネルギーも削減できる。表 4 より、SCIMA ではロード/ストア命令の大部分が SCM へのアクセスであるため、チップ内の消費エネルギー削減率は 75%～77%と CacheWP の場合に比べ、より高くなっている。

合計の消費エネルギーについて比較すると、SCIMAsel ではチップ内部、およびバスの消費エネルギーの両方が CacheWP に比べ削減されるため、両モデル間の消費エネルギーの差はさらに大きくなる。例えば FT では、約 50%もの消費エネルギーが、また最も低い CG の 128B ラインサイズの場合でさえ 5%の消費エネルギーが CacheWP に比べ削減されている。従って、SCIMA はメモリシステム全体として見た場合、キャッシュアーキテクチャに比べ低消費エネルギー化を達成できるアーキテクチャであると結論付けることができる。

6.3 性能と消費エネルギーの関係

次に、クロックレベルの性能評価を用いた Cache モデルと SCIMA モデルの性能評価結果を図 10 に示す。この図からわかるように、SCIMA はキャッシュアーキテクチャに比べ高性能が達成できる。従って、SCIMA は SCM を用いることで、高性能かつ低消費電力を実現できるアーキテクチャである。この性能と消費エネルギーの関係について議論するため、図 11 に Cache モデルと SCIMA モデルの ED 積(Energy-Delay Product)を示す。両モデルとも、消費エネルギーは選択的ウェイアクセスを行なった場合を想定している。

図 11 より、SCIMA は Cache に比べ ED 積を大きく改善できることがわかる。この結果から、性能と消費エネルギーの両方を考えた場合、キャッシュアーキテクチャに対する SCIMA の優位性はさらに増すと考えられる。

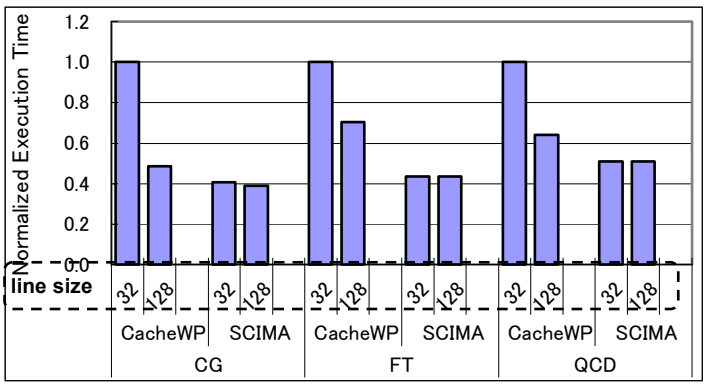


図 10 性能評価結果(COLPFig12)

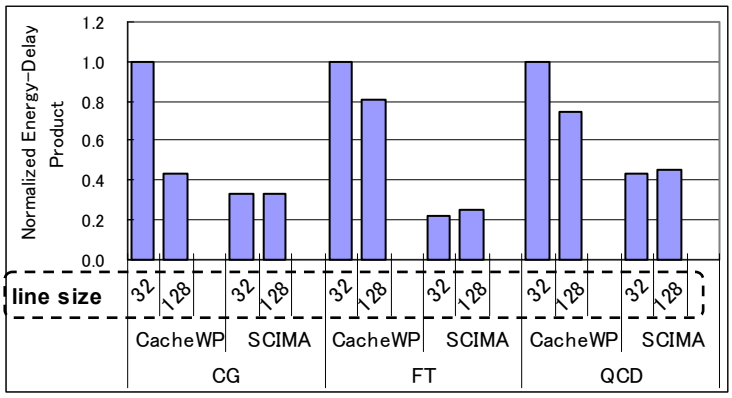


図 11　ED 積 (fig9, SWoPP02)

7. まとめと今後の課題

本稿では、ソフトウェア制御のオンチップメモリを用いる SCIMA のアーキテクチャの概要、オンチップメモリを用いた性能向上手法、ならびに、オンチップメモリ利用による低消費電力化について評価を行なった。また、キャッシュ/SCM アクセスにおける消費電力を含めて評価し、メモリシステム全体の低消費電力化の効果についても検討した。

評価結果より、SCIMA ではトラフィック削減により、キャッシュに比べ実際にデータバス、アドレスバスの消費エネルギーを削減できることがわかった。また、SCIMA ではキャッシュ/SCM アクセスに費やされる消費エネルギーも大きく削減できることから、メモリシステム全体として見た場合、キャッシュアーキテクチャに比べ大幅な消費エネルギー削減効果が得られることがわかった。今後は、さらに詳細なモデルで評価を行なう予定である。

参考文献

[1] 中村 宏、近藤 正章、大河原 英喜、朴 泰祐, “ハイパフォーマンスコンピューテティング向けアーキテクチャ SCIMA”、情報処理学会論文誌、Vol. 41、No. SIG 5(HPS 1)、pp.15-27, 2000 年 8 月

[2] 近藤 正章、中村 宏、朴 泰祐, “SCIMA における性能最適化手法の検討”、 情報処理学会論文誌、 Vol. 42、 No. SIG 12(HPS 4)、 pp.37-48、 2001 年 11 月

[3] 近藤 正章、 大根田 拓、 田中 慎一、 中村 宏、 “ソフトウェア可制御オンチップメモリを用いた低消費電力化の検討”、 並列処理シンポジウム JSPP2002、pp.285-288、2002 年 5 月

[4] M.Lam, E.Rothberg and M.Wolf, “The cache performance and optimizations of Blocked Algorithms”, Proc. ASPLOS-IV, pp.63-74, 1991

[5] PC SDRAM Specification、 Revision 1.7, Intel Corporation, Nov. 1999

[6] S.Aoki, et al. “Performance of lattice QCD programs on CP-PACS”, Parallel Computing 25、 pp.1243-1255, 1999。

[7] M.B.Kamble, and K.Ghose, “Analytical Energy Dissipation Models For Low Power Caches”, Proc. of 1998 Internatilnal Symposium on Low Power Electronics and Design, pp.143-148, Aug. 1998

[8] M.B.Kamble, and K.Ghose, “Energy-Efficiency of VLSI Caches: A comparative Study”, Proc. of the IEEE 10th Int'l. Conf. on VLSI Design, pp.261-267, Jan. 1997

[9] K.Inoue, T.Ishihara, and K. Murakami, “Way-Predicting Set-Associative Cache for High Performance and Low Energy Consumption”, Proc. of the Int'l. Symp. on Low Power Electronics and Design, pp. 273-275, Aug. 1999.

[10] S. Wilton and N. Jouppi, “CACTI: An Enhanced Access and Cycle Time Model for On-Chip Caches”, WRL Research Report 93/5, July 1994.

3.3 基本ソフトウェア＆ミドルウェア

3.3.1 アドバンスト並列化コンパイラプロジェクトにおけるマルチグレイン並列処理

笠原　博徳　委員

1. プロジェクトの構成

アドバンスト並列化コンパイラ技術研究体は、図 1 に示すように早稲田大学　笠原博徳を研究開発責任者（PL：プロジェクトリーダ）、早稲田大学山名早人と産業総合研究所小池汎平をサブリーダ、アドバンスト並列化コンパイラ技術グループリーダ JIPDEC(富士通より出向)堀田浩一郎、並列コンパイラ性能評価技術グループリーダ JIPDEC(日立より出向)安崎篤郎、管理法人（財）日本情報処理開発協会(JIPDEC)、JIPDEC に出向（兼務）して研究を行う㈱日立製作所及び富士通㈱からの研究員 23 名、及び共同研究先として産業技術総合研究所、早稲田大学、再委託先として電気通信大学、東京工業大学、東邦大学で構成されている。これらのメンバは、産技制度初の試みである各研究拠点をネットワーク等の利用による「ネットワーク集中研究所方式」で研究開発を行っている。ただし、より密な共同研究開発・当該分野の人材育成を目指し、ネットワークベースの議論に加え、物理的に一カ所に集まり、講演会・開発技術に関する討論を行う技術研究会も頻繁に開催しており、その数は 3 年間で 46 回に達している。

さらに、本プロジェクトでは海外との協調、目標設定の妥当性、成果の海外へのアピール及び国内評価委員会の参考として世界的権威による評価を自主的に受けることを目標に国際協調委員会を設置し、平成 13 年 9 月第 1 回委員会を開催し、第 2 回委員会は APC プロジェクトの最終成果報告会を兼ね平成 15 年 3 月 20 日に早稲田大学理工学部で開催される国際シンポジウム(APC2003)に併設して開催する予定である。委員会の中核となるインターナショナルアドバイザリボードは Univ. of Illinois, Prof. Padua (Parafrase, Cedar Fortran, Polaris 等並列コンパイラの世界的権威)、Stanford Univ, Prof. Lam (Suif, National Compiler Infrastructure, Suif Explorer 等コンパイラ、チューニングツール分野の権威)、Purdue Univ Prof. Eigenmann (Polaris コンパイラ, SPEC HPC 等、コンパイラ及び性能評価の権威)、Ecole des Mines de Paris, Prof. Irigoin (PIPS Parallelizer 手続き間解析の権威)で構成されている。ボードメンバとは協調委員会の席だけではなく、ネットワークベースで、技術動向、プロジェクトの進め方、世界への成果のアピール等について議論して戴くと共に、プロジェクトの中間評価も行って戴いた。その評価概要は以下である。

1）粗粒度タスク並列処理・アフィン変換等世界をリードする研究テーマで興味深い。
2）3 年間で性能を 2 倍向上させるという数値目標は、世界の従来のコンパイラによる性能向上が年平均数%であることを考えると非常に難しい目標であり、数値目標にとらわれずに機能の向上を目指すべき。
3）粗粒度タスク並列化等で予想を上回る成果が出ている。
4）成果を海外でよりアピールすべき。

2. 開発技術

APC では、共有メモリ型マルチプロセッサを中心としたマルチプロセッサの実効性能、価格性能比、使いやすさを向上させ、ハイエンドサーバ自身及びそれを使用する各種研究開発の競争力強化を支援すると共に、今後の携帯電話、PDA、ゲーム等チップマルチプロセッサを使用する各種の情報機器の競争力強化に資することを目的としている。特に、並列処理ではユーザは並列化チューニングに非常に大きな時間を費やさねばならず、本来のモデル開発、ゲームを含めたアプリケーション開発に十分な時間をとれないという問題を抱えている。この問題を軽減し、ソフトウェア開発の生産性を向上させハードウェアと合わせ市場を開拓していくことが今後の IT 機器開発では重要であり、そのためにも本自動並列化コンパイラの開発を急ぐ必要がある。

本プロジェクトでは、従来の市販ループ並列化コンパイラの性能を同一マシン上で概ね 2 倍上回るということを目標としているが、ハードウェアと異なりコンパイラの性能は 100%を上限とした実行効率で評価されるため 2 倍は非常に難しい値である。これを達成するためには、従来のループ並列化ではなく新たな並列処理方式の導入が必須であり、APC ではマルチグレイン並列化をコア技術として次に示すコンパイラ技術の研究開発を行っている。また、コンパイラの数値目標の評価も従来世界のプロジェクトで例がないため開発技術による性能向上を客観的に示すために性能評価手法に関する研究も行っている。

2.1 自動マルチグレイン並列化技術の開発

マルチグレイン並列化は、従来のループ並列化に加え、サブルーティン・ループ間などの粗粒度タスク間並列性、近細粒度並列性等を階層的に使用する方式で、低オーバーヘッドでプログラム全域にわたるより大きな並列性を利用することが可能である。また、このマルチグレイン並列処理の性能を向上させるデータ依存解析技術、キャッシュあるいは分散共有メモリの有効利用によりメモリアクセスオーバーヘッドの軽減を目指すデータ自動分散技術等に関する研究・開発を行っている。

まず、プログラム中のサブルーティン、ループ、基本ブロック間の並列性を利用する粗粒度並列処理は、マクロデータフロー処理とも呼ばれ[1][6][7]、APC コンパイラのコアの一つとなる早稲田大学 OSCAR マルチグレイン Fortran 並列化コンパイラで初めてが実現された[1]。

OSCAR コンパイラでは、図 2 に示すように、粗粒度タスク（マクロタスク）として、単一の FORTRAN プログラムより RB (Repetition Block)、SB (Subroutine Block)、BPA(Block of Pseudo Assignment Statements)の 3 種類のマクロタスクを生成する。RB は各階層での最外側ナチュラルループであり、SB はサブルーティン、BPA はスケジューリングオーバーヘッドあるいは並列性を考慮し融合あるいは分割された基本ブロックである。

次にマクロタスク間の制御フロー[1]とデータ依存を解析し、図 3(a)のようなマクロフローグラフ(MFG)を生成する。MFG では、各ノードがマクロタスク(MT)、点線のエッジが制御フロー、実線のエッジがデータ依存、ノード内の小円が条件分岐文を表している。また、MT7 のループ(RB)は、内部で階層的に MT 及び MFG を定義できることを示している。

次に、マクロタスク間制御依存[1][6]及びデータ依存より各マクロタスクが最も早く実行できる条件（最早実行可能条件）[1]すなわちマクロタスク間の並列性を検出する。この並列性をグラフ表現したのが図 3(b)に示すマクロタスクグラフ(MTG)である。MTG でも、ノードは MT、実線のエッジがデータ依存、ノード内の小円が条件分岐文を表す。ただし点線のエッジは拡張された制御依存を表し、矢印のついたエッジは元の MFG における分岐先、実線の円弧は AND 関係、点線の円弧は OR 関係を表している。例えば、MT6 へのエッジは、MT2 中の条件分岐が MT4 の方向に分岐するか、MT3 の実行が終了した時、MT6 が最も早く実行が可能になることを示している。

次にコンパイラは、MTG 上の MT をプロセッサグループへスタティックに割当てを行う（スタティックスケジューリング）か、実行時に割当てを行うためのダイナミックスケジューリングプログラムを Dynamic CP アルゴリズムを用いて生成し、これを生成する並列化プログラム中に埋め込む。これは、従来のマルチプロセッサのように OS あるいはランタイムライブラリに粗粒度タスクの生成、スケジューリングを依頼すると数千から数万クロックという大きなオーバーヘッドが生じてしまう可能性があり、これを避けるためである。このような並列化プログラムを OpenMP を用いて生成するために、APC コンパイラでは図 4 に示すワンタイムシングルレベルスレッド生成法を開発した。図 4 は、第一階層のマクロタスクグラフがデータ依存エッジのみをもっているため第一階層の 4 つのマクロタスクをコンパイラがスタティックスケジューリングを適用し、各 4 スレッドからなる

2 つのスレッドグループに割り当てる様子を示している。また、サブルーティンであるマクロタスク MT1_3 及び並列化できないループであるマクロタスクタスク MT1_4 内では、階層的にマクロタスクグラフを生成し、各グラフの並列度を推定後、その並列性にあったスレッドグループを生成し階層的に並列処理する際の並列プログラムのイメージを示している。この例の場合、これらのマクロタスクグラフは条件分岐を含んでいるため、ダイナミックスケジューリングが適用される。図中では、マクロタスクグラフ MTG1_3 に対しては、一つのスレッドにスケジューリングを任せる集中型ダイナミックスケジューリングを適用する場合のコードイメージが示されており、マクロタスクグラフ MTG1_4 に対しては 4 スレッドを 2 スレッドずつの 2 グループに分け、各スレッドグループがスケジューリングを行う分散型ダイナミックスケジューリングの場合のコードイメージが示されている。

また、APC コンパイラは、図 5 の SPEC CFP95 SU2COR プログラムのように多重にネストされたプログラムに対しても、各階層のマクロタスクグラフの並列性をループ並列性も含め推定し、各階層で利用できる並列性に見合ったスレッドグループ数を自動的に決定できるようになっている[7]。なおこの図 5 の例では、サブルーティン LOOPS 中に図 6 に示すような粗粒度タスク並列性が大きい部分を見つけ、この図 6 で表されるマクロタスクグラフに 7 スレッドから成る 2 つのスレッドグループを割り当てていることが示されている。

さらに、このような粗粒度並列性を抽出する際には、サブルーティン間にわたるデータ依存を正確に解析することが重要となる。APC プロジェクトではこのインタープロシージャ解析の高度化に関する研究開発も行った。APC コンパイラは、サブルーティンにわたる定数の伝搬を行い、必要に応じクローンと呼ぶサブルーティンのコピーを作成し、その結果不要となる条件分岐文等を削除しプロシージャ間のデータ依存を正しく解析できるようにしている[7]。これにより粗粒度タスク間の並列性をより多く引き出せると共に、ループボディ中にサブルーティンコールを含むループの並列化もより高精度に行うことができるようになった[7]。

また APC では、不完全ネストループより並列性を引き出すと共に、データローカリティを高め、さらにイタレーション間同期オーバーヘッドを減らす手法としてスタンフォード大 Suif グループにより提案されたがコンパイラには実装できなかった Affine Partitioning[8]を実現する実用的な手法開発し、パイプライン化された並列コードを OpenMP を用いて生成することに成功した[7]。

2.2 メモリ利用最適化技術

実際のマルチプロセッサシステム上で効果的な並列処理を実現するためには、並列性の

抽出のみならず、メモリアクセスオーバーヘッドを如何に抑えるかが重要な課題となっている。これはプロセッサの動作速度向上に対しメモリのアクセス速度向上が追いついていないために生じるもので、主記憶共有メモリ型マルチプロセッサではキャッシュの有効利用が、分散共有メモリ型マルチプロセッサでは近接メモリの有効利用が性能に大きく影響する。

このため APC コンパイラでは、主記憶共有型マルチプロセッサ用の分散キャッシュメモリの有効利用を目指したデータローカライゼーション技術と、分散共有メモリ型マルチプロセッサのためのファーストタッチ制御方式を開発した。

データローカライゼーションは、図 7 に示すようにマクロタスクグラフ上でデータ依存が存在する複数の異なるループにわたりイタレーション間のデータ依存を解析し（インターループデータ依存解析)、プロセッサ間データ転送が最小になり、さらにデータがキャッシュに収まるようにループとデータを分割（ループ整合分割）する[6]。さらに一度キャッシュ上に載せたデータを複数のループにわたり使用できるように、前のループが終了してから次のループを実行するという通常の実行順序ではなく、図 8 に示すように前のループの一部（整合分割されたループの一部）を実行すると同一のデータにアクセルする次のループの一部、またその次のループの一部を実行するという方式で、キャッシュ上のデータを長期間有効利用することを可能にしている。

さらに、プログラム中の配列サイズがキャッシュサイズの整数倍あるいは整数分の 1 のサイズだと配列間でラインコンフリクトが頻発することがある。これを避けるため、本データローカライゼーション手法では、図 9 に示すように配列サイズをコンパイラが拡張する配列間パディングを行い、ローカライゼーション後のラインコンフリクトを最小化する方式も実装している[7]。

また、分散共有メモリ型マルチプロセッサ用の自動データ分散では、SGI 社 Origin2000 の OS が、最初に当該データを触ったプロセッサ上のメモリにデータを割り付けるファーストタッチ方式を採用していることから、これを利用しコンパイラが望むデータ分散を実現する方式を開発した。これは、プログラム中で最も処理時間を消費するループ（メインループ）で要求されるデータ分散に合わせ、コンパイラがその分散を実現するダミーのループ（ファーストタッチ制御ループ）を生成し、メインループ内での遠隔分散共有メモリへのアクセスを最小化する。この方式を用いると、配列へのアクセスパターンが実行時まで分からないため人手でもデータ分散が難しかった配列の間接参照を伴うプログラムに対しても最適なデータ分散を実現できる。これにより、SGI 社コンパイラに対し、32 プロセッサ Origin2000 上で NAS Parallel Benchmarks CG プログラムを 6.6 倍高速に実行できることを確かめている。

3. APC マルチグレイン並列化コンパイラの性能

ここでは、前章までで述べたコンパイラ技術を統合した APC コンパイラを市販の主記憶共有メモリ型マルチプロセッサシステム上で性能評価した結果について述べる。

APC コンパイラは、逐次型 FORTRAN プログラムを入力すると OpenMP 指示文を用いて並列化された FORTRAN プログラムを生成する。今回評価に使用したマシンは、Power4 チップを搭載した IBM pSeries690 16 プロセッサ・ハイエンドサーバ、IBM RS6000 604e high-node 8 プロセッサ・ミッドレンジサーバ、SUN Ultra80 4 プロセッサデスクトップワークステーションである。これらのマシンを使用するにあたって、使用するコンパイラは、pSeries690 では最新の IBM コンパイラ XL Fortran Ver.8.1、RS6000 では APC プロジェクト開始時のコンパイラ XL Fortran Ver. 7.1、Ultra80 では SUN Forte Ver.6 update 2 を用いた。評価においては、SPEC CFP95 及び SPEC CFP2000 の内の全 FORTRAN77 プログラム 16 本に対し APC コンパイラを適用し、生成された OpenMP コードを上述の各社コンパイラを用いてコンパイルして実行した。本アドバンスト並列化コンパイラプロジェクトにおいてはプロジェクトの基本計画としてプロジェクト開始当時のループ並列化コンパイラの性能を概ね 2 倍上回るという数値目標を与えられており、この性能評価を客観的に行うために性能評価を担当するグループがプロジェクト内に設置されている。コンパイラ開発プロジェクトで数値目標をもった研究開発は世界でも例がなく、2 倍をどのように評価するかが問題になるが、本プロジェクトにおいては使用可能なプロセッサ数までのプロセッサを用いた際の商用コンパイラの最高性能と APC コンパイラの最高性能を比較する方式をとることとした。これはループ並列性のみを使っている市販コンパイラに比べ、APC コンパイラはより大きな並列性及びメモリアクセスオーバーヘッドの軽減が可能であるためプロセッサ数と共に性能が向上する場合が多いが、市販コンパイラの場合プロセッサを増やしすぎると逆に性能が低下してしまう場合があり、例えば同じ 16 プロセッサ同士の性能を比較すると見かけ上 APC コンパイラの性能が非常によく見えてしまうためである。

このように各マシンが持つ最大プロセッサ数までの最高性能を、逐次処理に対するスピードアップ率として表示したのが、図 10 から図 12 である。

図 10 は、16 プロセッサ pSeries690 上での性能であり、各バー2 本の組の内、左側が XL Fortran Ver.8.1 ループ並列化コンパイラを用いた時のスピードアップ率であり、右側が APC コンパイラを用いた時のスピードアップ率である。図中プログラム名の下に書かれている数値は逐次処理時間[s]を表しており、各バー上の数字は並列処理時の処理時間[s]を、またカッコ内の数字は APC コンパイラの XL Fortran コンパイラに対する速度向上率を示している。この図を見ると分かるように、APC コンパイラは SPEC CFP95 turb3d

プログラムに対し、pSeries690 上で最高 10.7 倍の高速化を達成できることが分かる。また16プログラムの平均速度向上率を計算すると算術平均で3.5倍という数値が得られており、最新のマシン及び最新の市販コンパイラに対してプロジェクト目標を大幅に上回る性能を得ることができた。

また、図 11 は 8 プロセッサ IBM RS6000 上での性能を表している。プロセッサ数が少ないこともあり、pSeries690 と比べ性能向上率は小さいが、最大で 6.2 倍、平均で 2.4 倍の性能を得ることができた。これらの IBM マシンで最大性能が得られた turb3d では、クローニングを伴うインタープロシージャ解析、ワンタイムシングルレベルスレッド生成を用いたマルチグレイン並列化が有効に働き上記のような性能が得られている。また、tomcatv、swim 等のプログラムではデータローカライゼーションを用いたキャッシュ最適化が有効に働き、良好な速度向上が得られた。

図 12 は上記 2 マシンと比べ価格の低い 4 プロセッサ主記憶共有型ワークステーション上での性能を表している。このマシンは、上記 2 マシンに比べメモリ性能が低いため、キャッシュ最適化が性能に大きく影響する。このマシン上では、レベル 2 キャッシュがダイレクトマップ方式であるためパディングによるラインコンフリクト除去を併用したキャッシュ最適化が極めて有効で、SPEC CFP95 swim では 4 プロセッサで逐次処理に比べ 9.3 倍の速度向上、Forte コンパイラに比べ 5.4 倍の速度向上を得ることができた。全 16 プログラムに対する速度向上の平均でも 2.0 倍という数値を得られている。

以上のように、APC コンパイラは複数の機種の主記憶共有メモリ型マルチプロセッサに対して極めて高い性能を示すことが確かめられた。

4. まとめ

アドバンスト並列化コンパイラプロジェクトは 2003 年 3 月に実質 2 年半の研究開発を終了する。このような短い研究開発期間を選んだのは、競争の激しいこの分野でマルチグレイン並列化という新しい技術の優位性を示すには、参加メンバーが所有しているコンパイラ技術をフル活用し短期間での開発実証が重要と考えたためである。

プロジェクトを通じ、従来例を見ない数値目標は大きなプレッシャーであったが、最新の 16 プロセッササーバー上で、そのマシン向けの最新のコンパイラと比べ、平均 3.5 倍の速度向上を達成できたことは、開発者としても予想しなかった好成績である。これは、同一のハードウェア上で、APC コンパイラを一度通すだけで自動的に性能向上を得られることを意味しており、この技術をさらに磨けばマルチプロセッサを使用する各種アプリケーションのユーザの方の計算速度向上及びプログラム開発期間の短縮が可能になると考えている。

また、このコンパイラは高性能コンピュータの利用者のみならず、今後のチップマルチプロセッサの性能向上、価格性能比向上、アプリケーション開発期間の短縮にも役立つものと期待できる。特に、ゲーム、携帯電話、PDA などの SoC 分野ではハードウェア・ソフトウェアの開発期間短縮、価格性能比の向上、アプリケーションソフトウェアの質が市場を分ける重要な要素となるため、自動並列化コンパイラは今後この分野でも有効に使用できるものと期待される。

参考文献

［1］ 笠原博徳, 並列処理技術, コロナ社, 1991

[2] Banerjee U., Loop Parallelization, Kluwer Academic Pub., 1994

[3] Wolfe M., High Performance Compilers for Parallel Computing, Addison Wesley, 1996

[4] Eigenmann, R., Hoeflinger, J. and Padua, D.: On the Automatic Parallelization of the Perfect Benchmarks, IEEE Trans. on parallel and distributed systems, Vol.9, No.1 (1998).

[5] Hall, M. W., Anderson, J. M., Amarasinghe, S. P., Murphy, B. R., Liao, S.-W., Bugnion, E. and Lam, M. S.: Maximizing Multiprocessor Performance with the SUIF Compiler, IEEE Computer (1996).

[6] Kasahara H., Obata M., Ishizaka K., Kimura K., Kaminaga H., Nakano H., Nagasawa K., Murai A., Itagaki H., and Shirako J., “Multigrain Automatic Parallelization in Japanese Millenium Project IT21 Advanced Parallelizing Compiler”, Proc. of IEEE PARELEC, Warsaw, Poland, Sep. 23, 2002.

[7] APC2003:アドバンスト並列化コンパイラ国際シンポジウム資料, （財）日本情報処理開発協会(JIPDEC), 早稲田大学理工学部, Mar.20, 2003.

[8] Lim, A. W. and Lam., M. S.: Cache Optimizations With Affine Partitioning, Proceedings of the Tenth SIAM Conference on Parallel Processing for Scientific Computing (2001).

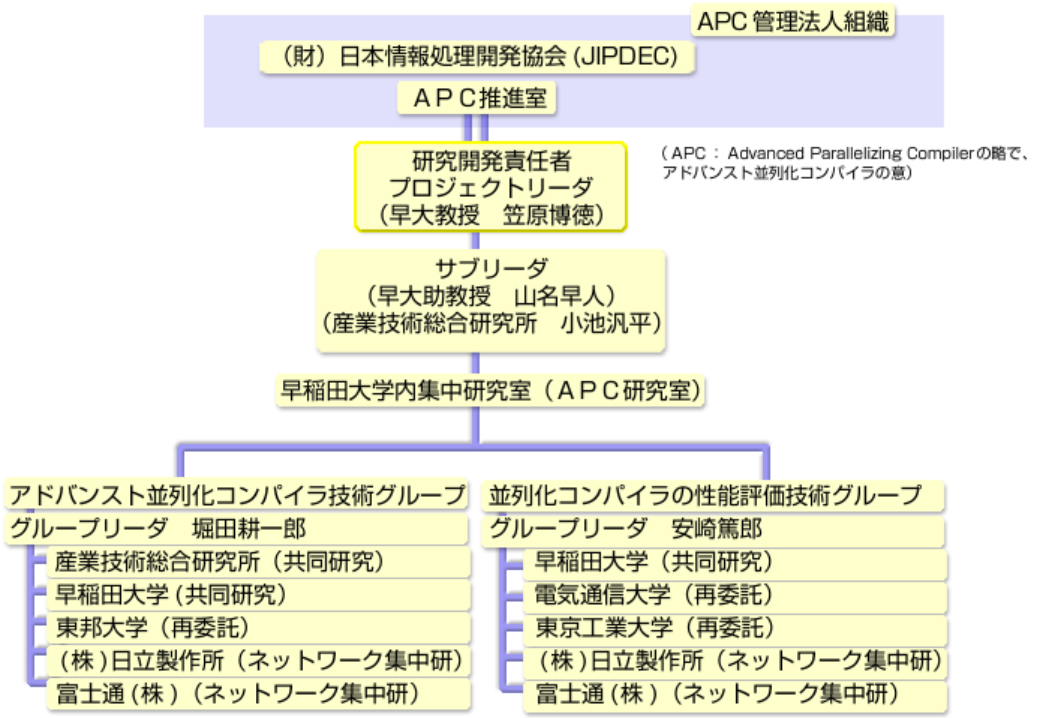


図 1　APC プロジェクト組織構成

■Macro-tasks (MTs)

- ↗ Block of Pseudo Assignments (**BPA**): Basic Block (BB)
- ↗ Repetition Block (**RB**) : outermost natural loop
- ↗ Subroutine Block (**SB**): subroutine

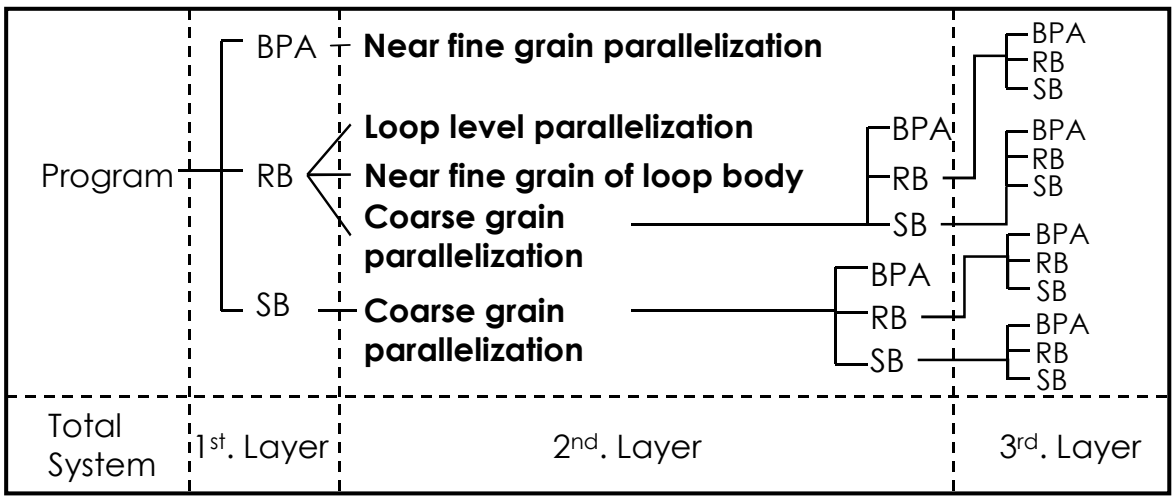
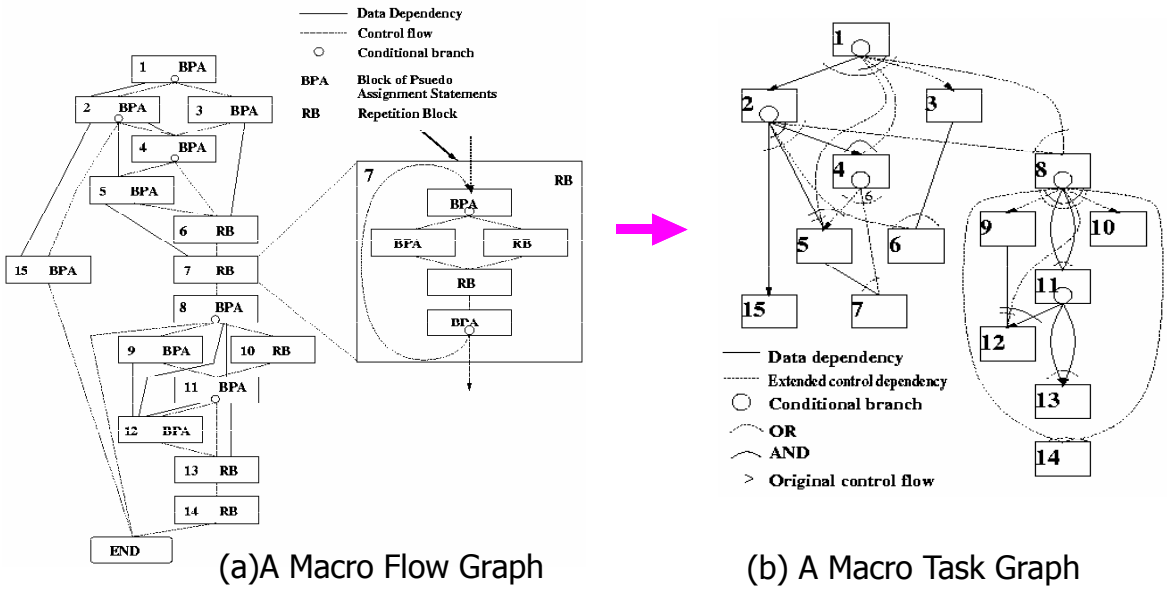


図 2　マクロタスクの階層的定義



(a)A Macro Flow Graph　　(b) A Macro Task Graph

図 3　マクロタスクフローグラフからの最早実行可能条件解析による
マクロタスクタスクグラフの生成

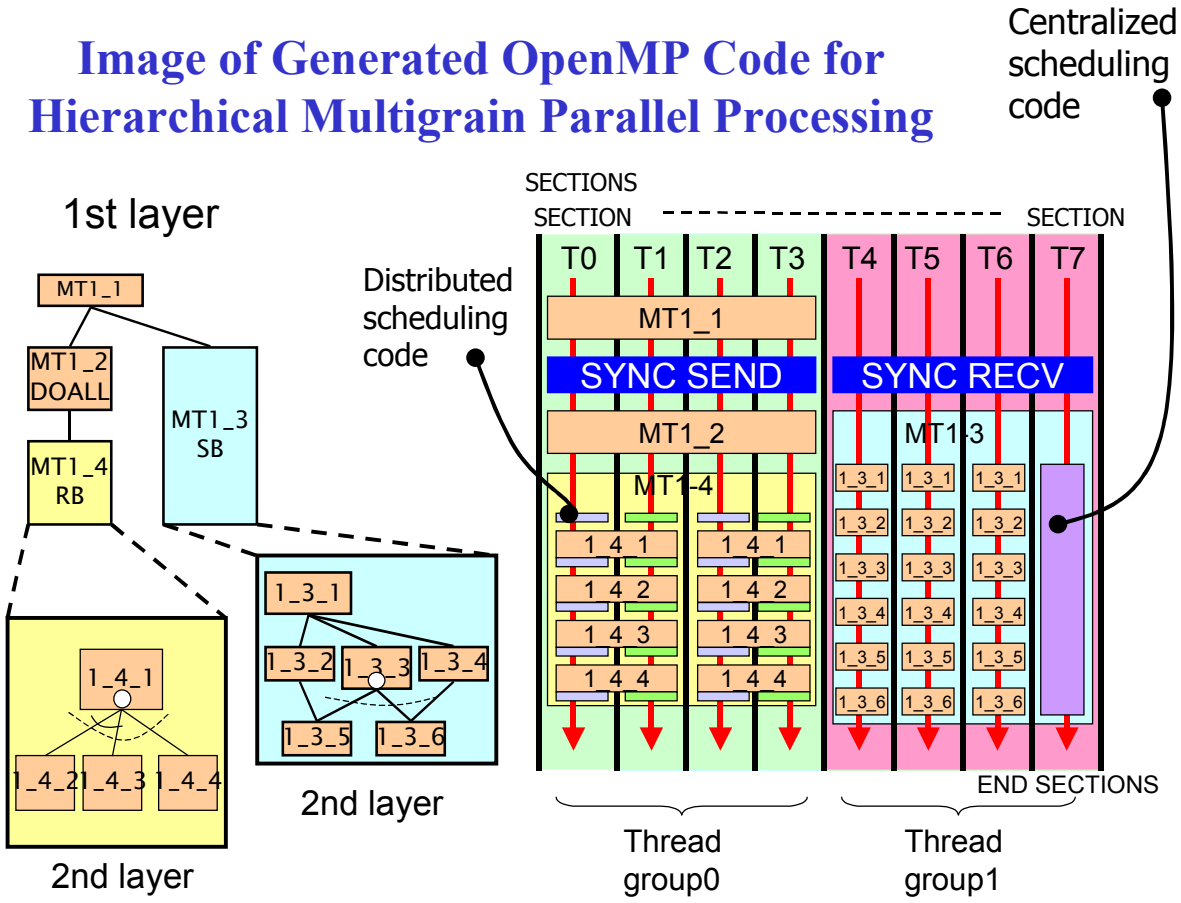


図 4　OpenMP を用いた階層マルチグレイン並列処理のための生成コードのイメージ

- **Using 14 processors**
  - **Coarse grain parallelization within DO400 of subroutine LOOPS**

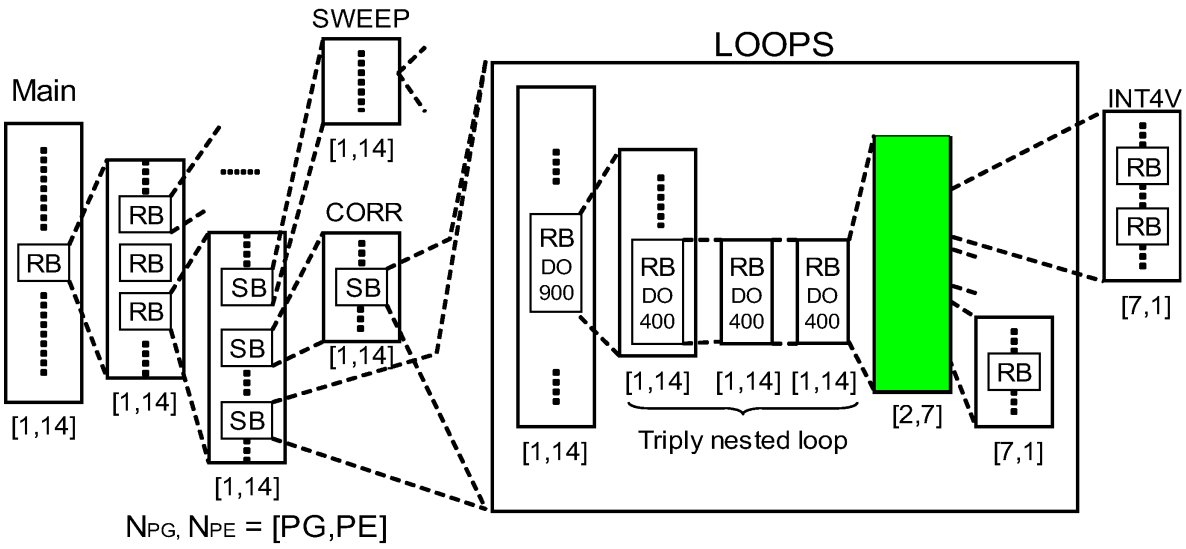


図 5　SPEC CFP95 SU2COR における並列処理階層及びプロセッサグループ(PG)数、PG内プロセッサ数の自動決定

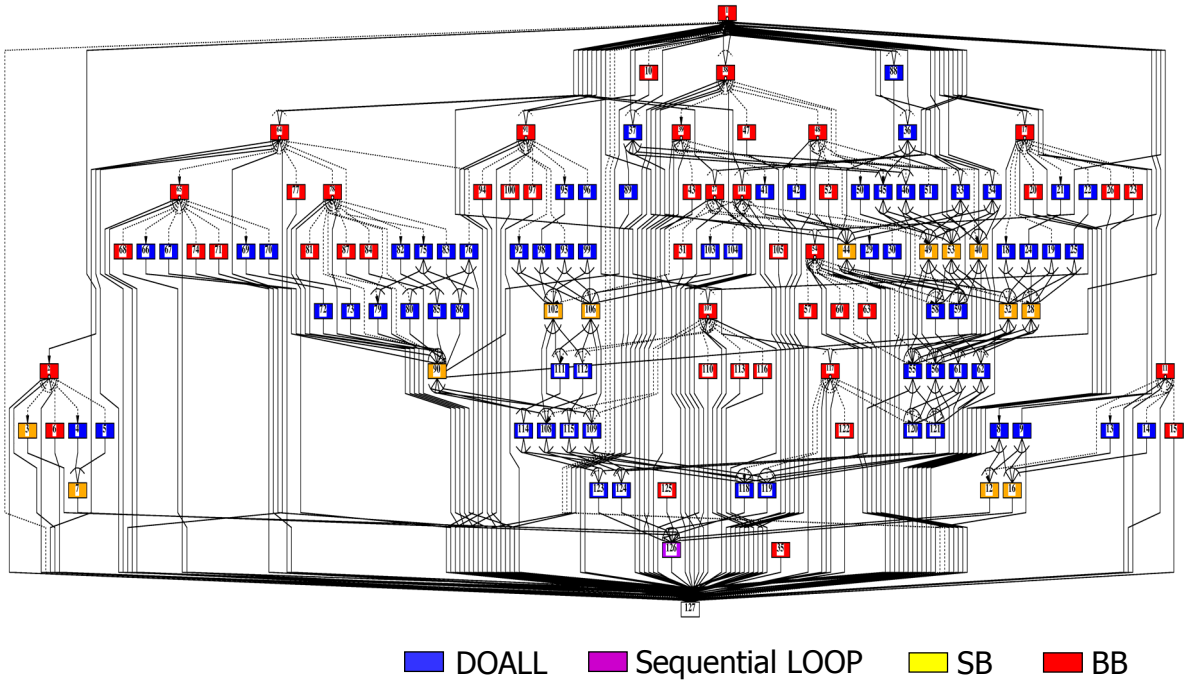


図 6　図 5 の LOOPS 部分のマクロタスクタスクグラフ

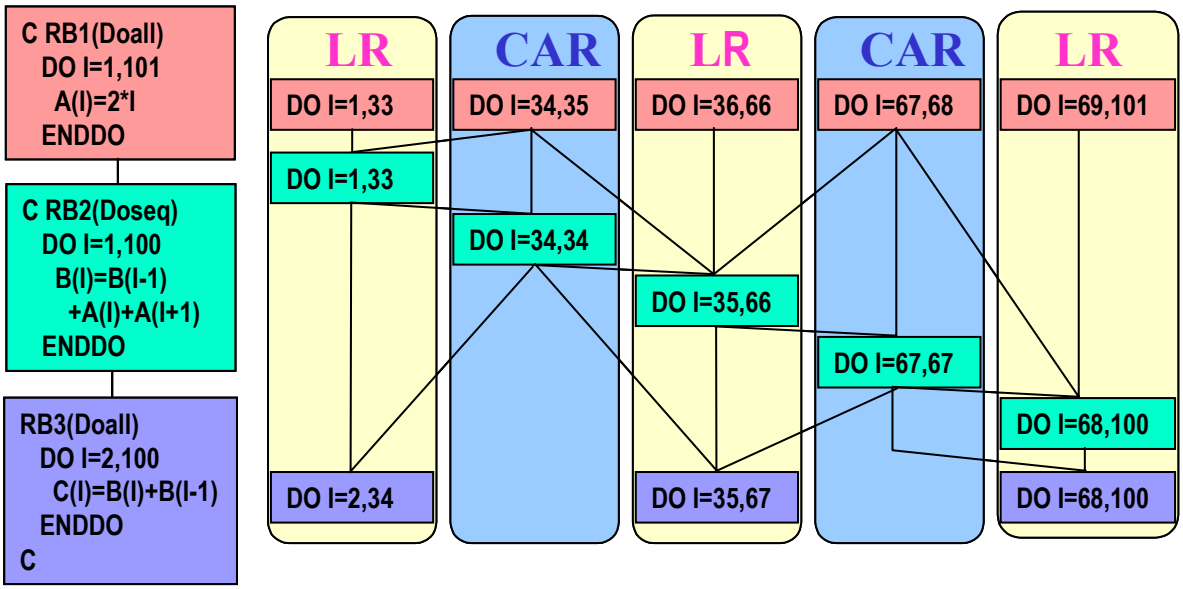


図 7　データローカライゼーションにおける整合分割

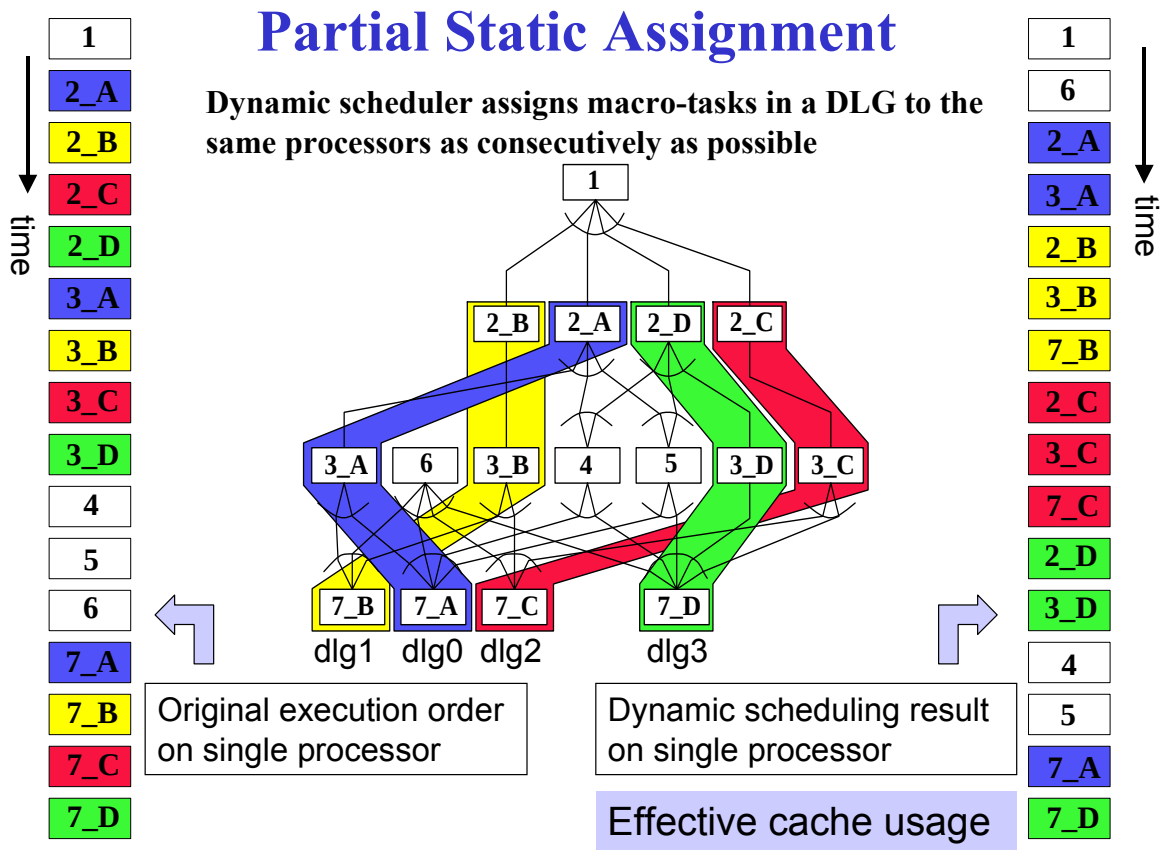


図 8　データローカライゼーショングループ DLG と 1 プロセッサへのスケジューリング例

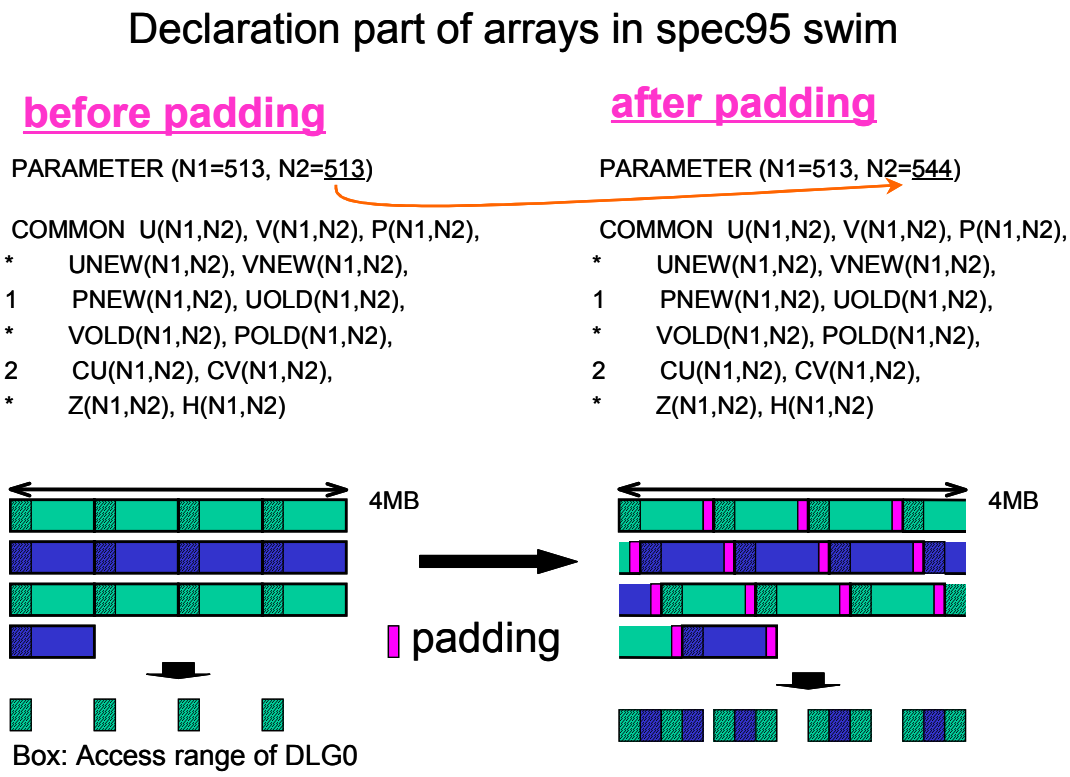
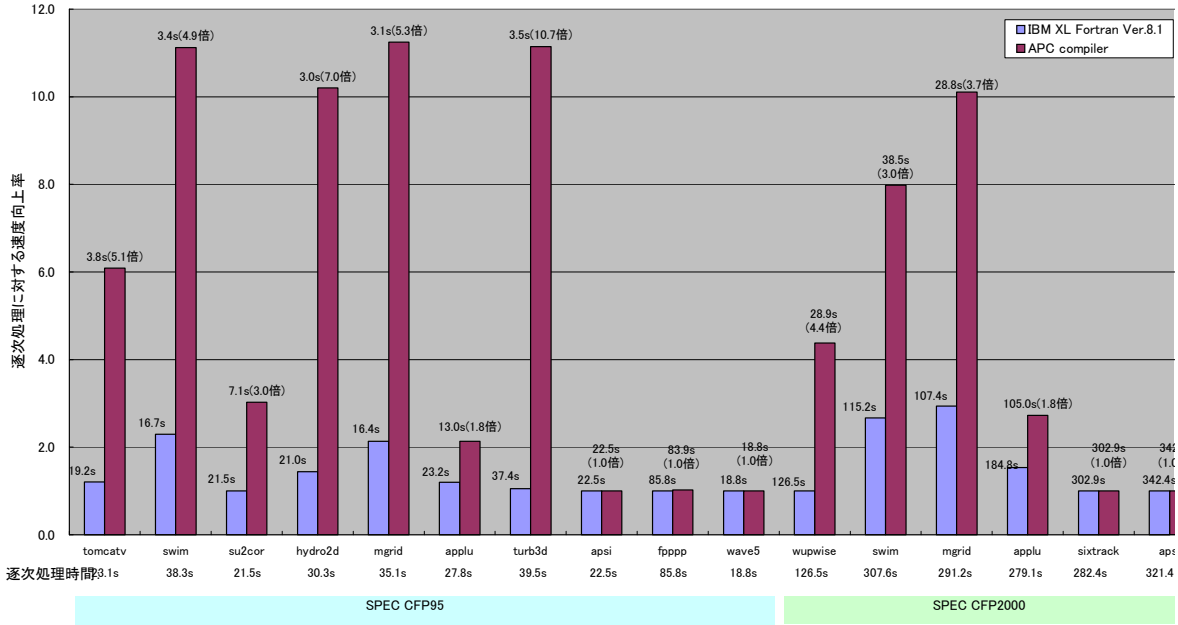
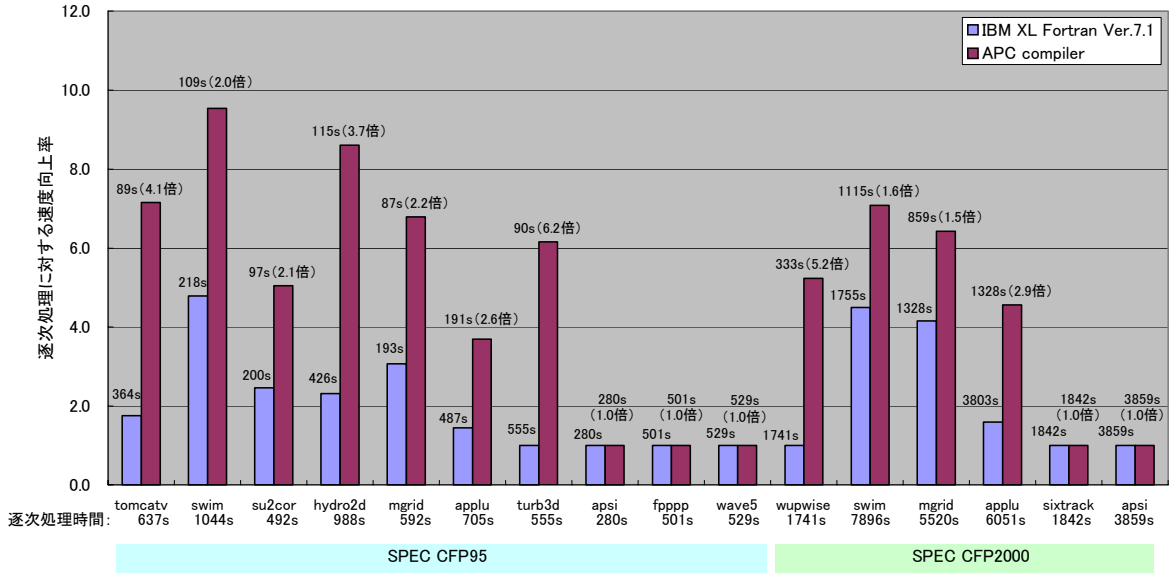


図 9　配列間パディングを用いたラインコンフリクトの除去



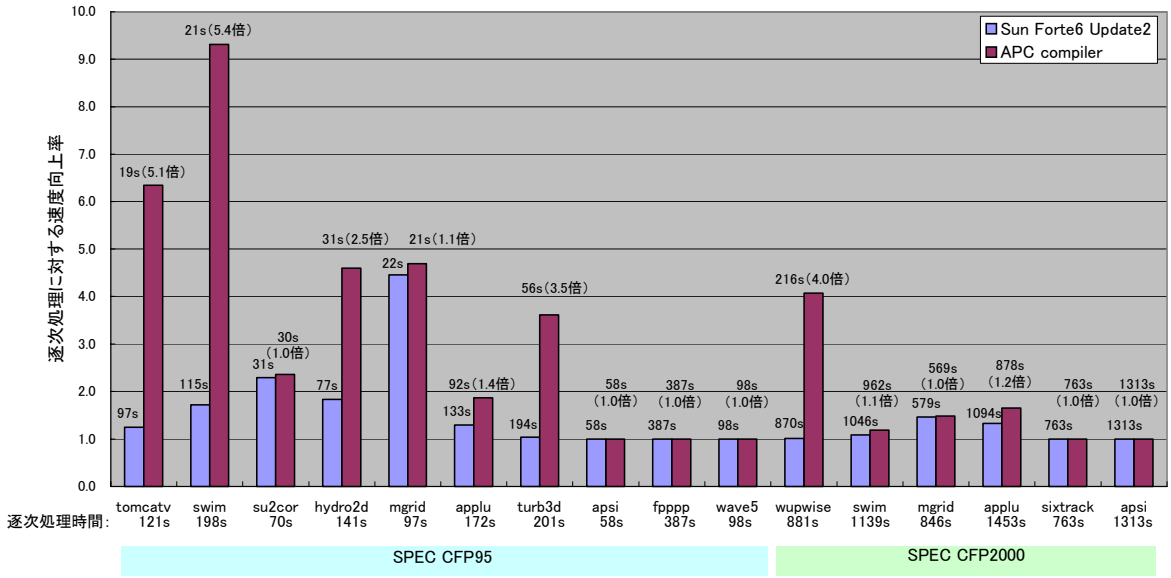
*各バーの数字はIBM社製最新コンパイラXL Fortran Ver. 8.1(青)と、開発したAPCコンパイラ(赤)による各プログラムの処理時間[s:秒]を示す。
APCコンパイラによる処理時間横のカッコ内は、IBMコンパイラに比べAPCコンパイラが何倍処理速度を向上できるかを示している。

図 10　IBM ハイエンドサーバ pSeries690 16 プロセッサ上での APC コンパイラの性能



*各バーの数字はIBM社製コンパイラXL Fortran Ver. 7.1(青)と、開発したAPCコンパイラ(赤)による各プログラムの処理時間[s:秒]を示す。
APCコンパイラによる処理時間横のカッコ内は、IBMコンパイラに比べAPCコンパイラが何倍処理速度を向上できるかを示している。

図 11　IBM RS6000 604e high node 8 プロセッサ上での APC コンパイラの性能



*各バーの数字はSun社製コンパイラForte 6 Update 2(青)と、開発したAPCコンパイラ(赤)による各プログラムの処理時間[s:秒]を示す。
APCコンパイラによる処理時間横のカッコ内は、IBMコンパイラに比べAPCコンパイラが何倍処理速度を向上できるかを示している。

図 12　Sun Ultra80 4 プロセッサワークステーション上での APC コンパイラの性能

3.3.2 HPF の 10 年と並列プログラミングインタフェース

妹尾 義樹 委員

1. はじめに

分散メモリ並列システム向けの並列プログラミング言語 HPF（High Performance Fortran）[1]について、その最初の仕様が定められてから、この春でちょうど 10 年になる。昨年 11 月に米国バルチモアで行われた SC2002 国際会議において、地球シミュレータ上で HPF を用いたプラズマシミュレーション並列化の成果が Gordon Bell Award の言語賞を受賞した[2]。陽解法の TVD 差分スキームを用いたプログラムで、並列化が比較的容易なコードではあるが、手続きにまたがった SHIFT 通信やリダクション演算を含む実用コードであり、MPI[3]で記述すればプログラムの全体構造の変換や、通信のための大量の MPI 呼び出しが必要になる。このコードに対して 30 数行程度（include ファイルに記述した共通の宣言文を複数手続きで引用しているので、実質的には 10 行程度）のデータマッピング指示挿入するだけで 14.9 テラフロップスを達成したことが認められた。ようやくここまで来たか、と感慨深いものがあるが、HPF の本当の実用化という意味では、まだまだ課題が残っているのも事実である。

本報告では、この 10 年間の HPF の歩みを総括するとともに、最近の並列プログラミングインタフェースの話題と、今後の方向について、私見を述べる。特に、HPF と MPI の中庸を狙ったといえる、Co-Array Fortran[4]や GA (Global Arrays)[5]を紹介し、MPI や OpenMP[6]を含めた種々のプログラミングインタフェースの比較を行い、今後の方向性について考えてみたい。

2. 分散メモリ向けプログラミングインタフェースと HPF

2.1 分散メモリプログラミングインタフェース

並列計算機のアーキテクチャは大別して、共有メモリ型と分散メモリ型に分けることができる。共有メモリモデルは、複数のプロセッサ間で主記憶のデータが共有されるシステムである。このため、プロセッサ間でのデータの明示的な受渡しは不要である。そのためプログラミングは容易になるが、接続できるプロセッサ台数に限界があるなどの欠点がある。分散メモリモデルは、プロセッサと主記憶から構成されるシステムが複数個互いに接続されたシステムである。共有メモリモデルに比較して、大規模なシステムを構築可能であるという特徴がある。しかし、他のプロセッサのデータを利用するためには、明示的にデータの授受を行う必要があるなど、プログラミングは共有メモリモデルに比較すると難

しい。また、この 2 つの中間に位置づけられる分散共有メモリアーキテクチャ[7][8][9]もある。物理的には、分散メモリとして構成されているシステムを基にしているため、HW または SW 制御によりプログラムからデータを共有メモリとして扱うことができるが、データが分散メモリのどこに格納されているかによって、性能差が生ずる。

分散メモリシステム（あるいは、分散共有メモリ）上の並列化における最も重要な２つのポイントは、データ分割（処理対象データの分散メモリ上への配置）と、計算マッピング（計算処理の各要素プロセッサ(以後 PE と記す)への分配）である。並列化の良し悪しは、並列性の抽出のみならず、いかにデータアクセスの局所性を高め、プロセッサ間通信を減らすかに大きく依存する。プログラマの立場からすると、データ分割と計算マッピングの両方を指定するというのは、大変な重荷になる。これが分散メモリマシン上での並列化の難しさの本質なのであるが、データ分割を主に考えてプログラミングする場合には、決めたデータ分割に合わせるように計算マッピングを指定する必要があるし、逆に、計算マッピングを主に並列化を決めた場合は、これに合わせたデータ分割およびデータ転送処理の記述が必要となる。

このデータ分割と計算マッピングをユーザがプログラム上でどのように記述するかによって､並列プログラミングインタフェースを類型化することができる｡詳細は後述するが、MPI（Message Passing Interface）は両者をすべて利用者が指定するインタフェースであり、商用コンパイラとしては存在しないが、完全自動並列化コンパイラというものがあれば、これは何も指定しないというインタフェースになる。主に片方を指定する方法が中庸として考えられ、HPF などのデータパラレル言語は、データ分割をユーザが明示的に指示し、それ以外の仕事を処理系（コンパイラ）に任せようというものである。逆に、計算マッピングだけを指示するのが、共有メモリ用システム用のインタフェースとして発展した OpenMP などの言語である。

2.2 HPF

HPF(High Performance Fortran)は、その名の通り高性能コンピューティング用に開発されたプログラミング言語である。 Fortran 言語に最小限の指示文を付加することにより、分散メモリ並列システムで簡単に高い性能を得ることを目指している。

HPF の本質的な考え方は、データ分割をユーザが明示的に指示し、それ以外の仕事を処理系（コンパイラ）に任せようというものである。これにより、ユーザは煩わしいプログラムの SPMD 化や通信管理から解放され、従来の Fortran プログラミングと非常に親和性の高い方法で分散メモリシステムを利用することが可能となる。HPF では、プログラムは従来の逐次言語と同様にシングルストリームのセマンティックスで記述され、データは

すべて大域的に扱われる。これに対して、MPI などのメッセージパッシングプログラムでは、SPMD (Single Program Multiple Data Stream)形式であり、プログラムは、並列実行に参加するすべてのプロセッサがそれぞれ独自の制御を持つ。つまり、IF 文などによるプログラムの実行経路がプロセッサごとに異なることをユーザが意識してプログラムする必要がある。また、プログラム上で宣言されたデータは、すべてのプロセッサが独自のコピーを持つことが前提であり、プロセッサ 1 上の配列 A とプロセッサ 2 上の配列 A は別々のデータとして扱われる。

HPF 言語の標準化活動は、ISO や ANSI などの公式な標準制定機関ではなく、Rice 大学の Ken Kennedy 教授を中心とするプライベートな組織 HPFF(HPF Forum)で行なわれた。ANSI を中心に行なわれた Fortran90 の仕様制定に 10 年以上かかった反省からである。言語仕様に組み込むべき新機能ごとにサブワーキンググループを作り、ここで仕様案が作られ、これらが参加者の投票を経て正式に仕様に組み込まれる。活動は 1991 年に開始され、1993 年に HPF1.0 が制定された。その後も HPFF2 において、仕様の曖昧性のチェックや、HPF1.0 で不十分であった機能拡張が検討され、1997 年 1 月に HPF2.0[10][11]の仕様がまとめられた。

HPFF のホームページが、http://www.vcpc.univie.ac.at/information/mirror/HPFF/ にあり、HPF 言語の仕様や関連研究プロジェクトの情報が得られる。

2.3 HPF のこれまでの歩み

筆者は、1987 年あたりから並列化言語の研究開発に携わるようになり、以降 15 年以上にわたり、並列マシンアーキテクチャ、言語インタフェース、コンパイラ技術の変遷とともに歩んできた。この 15 年を振り返るとともに、HPF の歩みを総括してみたい。

(1) 1990 以前の並列化インタフェースの状況

1985 年から 1990 年にかけてという時代は、商用システムとしてベクトルマシンの全盛期であり、CRAY、NEC、富士通、日立のハイエンドスーパーコンピュータに加え、Alliant や Convex などのミニスーパーといわれるシステムも広く使われた。また、分散メモリを採用したスカラプロセッサベースの高並列システムも多く開発・商用化された時期でもある。今では、ほとんどが消えてしまったが、Thinking Machine(CM5)や Ncube、FPS、Cenju、AP1000 など、種々の工夫をこらしたマシンがいろいろと現れ、Supercomputing(SC)国際会議では、種々のアーキテクチャ、デバイス、コンパイラ、通信ライブラリなどが所狭しと出展され、楽しい時代であった。

この時代、自動ベクトル化技術は、すでに成熟しており、共有メモリ向けの自動並列化

技術に各社がしのぎを削っていた。ベクトル化も自動並列化もデータ依存関係解析という意味では、ほとんど同様の技術で良いのだが、ベクトル化が局所的なループネストだけを対象に最適化できるのに対し、並列化はできるだけ並列処理単位を大きくし、並列処理や同期などのオーバヘッドを減らす必要があり、このために手続きをまたがる最適化が必須となる。手続き間解析は、技術的には完成の域に達し、Convex などいくつかの商用コンパイラで実装されたが、手続き間の解析時間が実用の大規模なプログラムになると膨大になることや、手続きごとに別ファイルにして、変更があったファイルだけ再コンパイルする make の環境で、オブジェクトの管理がややこしくなるなどの理由であまり日の目を見ることはなかった。一方で手続きをインライン展開する手法は、展開すべき手続きをユーザが指定するなどの方法で、簡便に用いることができ、また効果も大きいことから、現在でも広く用いられている。

自動並列化のもうひとつの問題は、ベクトル化に比べて最適化の選択肢が多いということである。ループネストの並列化ひとつにしても、ループ入れ替えやアンローリング、並列化すべきループの選択などが複雑に絡み合い、選択によって性能が大幅に変動するケースが多い。また、変な並列化を行うと、性能が極端に劣化するという問題もある。このため、並列化で最初にユーザに普及したのは、CRAY-XMP や NEC の SX-3 などでサポートされたマイクロタスキングとマクロタスキングというインタフェースである[12]。マイクロタスキングとは、プログラム中の主にループに対して並列化指示文を挿入するインタフェースであり、ワークシェアリングといって、並列化対象以外の部分は、全プロセッサで重複して実行されるというプログラミングモデルを採用している。OpenMP の原型となったインタフェースでもある。自動並列化（autotasking）との相違点は、コンパイラが勝手に並列化を判断することが一切ないという点と、ワークシェアリングモデルのため、指示文を無視して逐次実行したものと、プログラムの意味が変わってしまうという点である。自動並列化もループに並列化の指示文を挿入していくが、あくまでもコンパイラの自動処理の助けとなる情報を埋め込むという形式で、指示文を無視した実行が、並列処理と（指示文が正しく指定されていれば）一致する。

マクロタスキングとはスレッド並列処理の枠組みであり、手続きを非同期に呼び出し、呼び出された手続きを別々のプロセッサで実行することで並列処理を実現する。呼び出された手続き間の同期処理や排他制御は、イベントやロックという特別な変数を用いたライブラリ呼び出しで実現されている。Pthread ライブラリと同様のものと考えてもらえると分かりやすい。

1980 年台の後半には、マクロタスキング/マイクロタスキング/自動並列化の技術が共有メモリシステムを対象としてほぼ確立され、計算機システムベンダや、Kuck & Assciates、

PSR、APR などのベンチャー企業により製品化されたシステムが多数の実用コード開発に用いられるようになった。

この一方で、性能の追求とともに、分散メモリ型のマシンも多く開発され、計算機ベンダそれぞれが固有のメッセージパッシングライブラリを提供した。PVM[13]や PARAMACS などのたくさんのシステムで共通的に使えるライブラリも開発された。他方、これらのシステム上での並列化コンパイラを実現しようという研究活動も世界各地で立ち上がり、データマッピングを明示指定するという考え方の元になった Purdue 大学の KALI コンパイラ、Rice 大学の Fortran D[14]、欧州では Hans Zima らの SUPERB など多くの研究がなされた。また、SIMD マシンの Thinking Machine 上インタフェースとして、Forall 文を用いた並列化が特徴である CM Fortran も、この時代に実用化されている。Fortran D と CM Fortran は HPF の言語仕様開発に大きな影響を与えた。日本では、この時期スーパーコン大プロが最終段階にあり、共有メモリとローカルメモリの階層構造を持ったベクトル型のスーパーコンの上で、Phil や Paragram という並列言語が開発された。我々は、Cenju や SX-3 をターゲットにして並列化支援環境 PCASE の開発を行い、自動並列化エンジンを搭載した半自動の並列化システムを実現して、後の HPF 開発の礎を築いた。

データ依存解析などの並列化のための基礎理論は、Illinois 大学などの研究活動によって、この時期にはすでに完成されていたが、これを実用化するための技術が非常に進歩した時期であった。リダクションなどの、プログラム中の特定構造（イディオム）の認識、ループ融合・分割・入れ替えなどのループリストラクチャリング技術、先に述べた手続き間解析・インライン展開、ループ長などの実行時パラメータに基づく最適化切り替え、種々の指示行の導入によるユーザによる最適化促進技術などである。キャッシュ最適化やスーパースカラ処理など RISC 用のコンパイル技術も大きく発展した時代であり、コンパイラ技術者にとってはやりがいのある時期であった。

(2) HPF 誕生前後の状況

分散メモリマシン用の標準のプログラミングインタフェースを作ろうという機運が高まり、データパラレル言語 HPF の言語仕様を定めるべく、HPFF（HPF フォーラム）の設立が決まったのは 1991 年 11 月の Supercomputing 91 国際会議の BOF であった。欧米のアカデミアおよび企業のコンパイラ技術者および NASA などのユーザが 1 月半に一度のペースで集まり、基本仕様が 1 年間でまとめられた。リーダは Rice 大学の Ken Kennedy であり、言語仕様のベースとなったのは Fortran-D であった。言語仕様策定のポイントの 1 つとなったのは、ベース言語を Fortran77 にするのか Fortran90 にするのかという点であった。安定した Fortran90 処理系がまだない状況で、これをベースとするのはリスクが高

いし、HPF 処理系の早期実装の妨げになるとベンダから反対意見があったが、結局 Fortran90 がベースに選定された。今、振り返ってみると、この後、巨大仕様の Fortran90 の処理系開発に各ベンダとも手間取り、この選択が HPF の実装を大きく遅らせる要因となった。F77、F90 の両方での実装が可能な仕様にしておけば、その後の米国での HPF の事情はかなり変わったのではと思う。

HPFF における言語策定作業は、Fortran90 の仕様策定が 10 年近くかかったことと比べると、よく 1 年と少しでまとめられたと思う。毎回の会合は普通 3 日間行われ、この中で検討項目ごとのワーキンググループに別れ、それぞれ仕様案が検討される。毎日の最後には全体で仕様案の採択が深夜まで行われる。技術課題は宿題として参加者が分担して持ち帰り、次回以降に引き続き検討される。会議を開いて、この中で実のある議論を尽くして、参加者で協力して新たなものを作っていくというやり方は、なかなか日本ではできない。参考にすべきだと思う。

1993 年の春に HPF1.0 言語仕様が定まり、そのあとすぐに、APR（Applied Parallel Research）や DEC などが、フル実装ではないが、HPF コンパイラの開発に成功した。アカデミアでは、ドイツ GMD の Thomas Brandes が Adaptor[15]という HPF システムを早期に開発した。やはり、Fortran90 への対応と HPF 開発の両立は難しく、初期の HPF は性能・品質の両面で問題が多く、期待が大きかった分、欧米でのユーザ離れを招く要因となった。我々は、1992 年の段階で、簡単なデータ分散指示を受理する Cenju 用のコンパイラの開発に成功し、これを元に HPF コンパイラの開発を開始した[16]。Fortran77 ベースであり、HPF Conforming とは言えないものではあったが、最初の HPF 処理系が動き出したのは 1994 年の夏であった。アカデミアでは、Vienna 大学の Vienna Fortran Compiler[17]、Rice 大学の D system[14]などで、通信最適化の種々の方式（Communication Vectorization, Communication Aggregation, Communication Coalescing）や非定型処理への対応技術[18]などが開発された。

(3) HPF の成熟への歩み

HPFF は、1993 年から 1 年間は言語拡張の活動をストップし、言語のバグフィックスに努めた。アカデミアは、それまでの HPF の弱点と言われた、非定型データ構造への対応や計算マッピング指示、タスク並列処理への対応などの HPF 拡張活動の継続を求めたが、ベンダサイドからの、「成熟したコンパイラが完成しないうちに、次々と新規言語機能を追加するのは、ベンダの処理系開発が追いつかず、マイナスに作用する」との意見を採用した結果である。

そして、1994 年から HPFF が再開され、HPF2.0 の仕様策定作業が始まった。筆者は

HPF1.0 策定作業には、あまり参加できなかったが、Rice 大学での勤務の機会を得たこともあり、HPF2.0 の作業にはずっと参加することができた。アカデミア主導で、非定型分散（GEN_BLOCK、Indirect 分散）やタスク並列化など応用範囲を広げるための新機能追加が検討され、ベンダ中心でコンパイラ能力不足を補うための計算マッピング指示（ON 指定）や MPI など他の並列化インタフェースとの共存機能（Extrinsic インタフェース）が検討された。最大の議論となったのは、言語仕様の肥大化にどう対応するかということであった。HPF1.0 では、すべての機能の実装が大変で、どのベンダも自コンパイラを HPF1.0 準拠と言えなかったためである。その結果、HPF1.0 から、実行時分散変更機能（ダイナミック分散）など、実装が大変な機能を除いた、より基本機能だけのセットを HPF2.0 と定義し、HPF1.0 から除いた機能と、新規に策定した機能を、HPF2.0 Approved Extensions として定義することとなった。

1996 年に HPF2.0 が定まり、1997 年に HPFF は散会し、活動はユーザ主体で HPF を利用技術を検討する、あらたに設立された HPF User Group(HUG)[19]に引き継がれた。ただし、欧米では、並列アプリケーション開発プロジェクトが、1995 年ごろから多数立ち上がり、これらがすべて MPI を採用したことにより、徐々に HPF に対するユーザの興味は薄れていった。

この時期、日本では、NEC、富士通、日立が HPF や VPP Fortran コンパイラ[20]を開発しており、また地球シミュレータプロジェクトも開始の時期であった。地球シミュレータプロジェクトの生みの親である故三好さんは、この壮大なマシン用の並列プログラミングインタフェースをどうするか検討を開始していた時期であり、また上記 3 社も、欧米で HPF に暗雲が立ち込める中で、将来のコンパイラ開発をどうするかの岐路にあった。三好さんには、「欧米は MPI ベースで良いが、日本で分散メモリマシンを普及させるには、ユーザフレンドリーな並列化インタフェースが不可欠」との思いがあった。これらの人たちの思惑が一致し、1996 年から、日本主導で次世代のデータ並列言語の開発するための活動が開始された。まったく白紙の状態から言語仕様検討が開始されたが、議論の末、一応世界標準として認知されている HPF がもっとも有望だとの結論に達し、これをベースにインタフェース検討を行うことになった。三好さんと 3 社の技術者 15 名程度を中心に 1996 年の夏から、月に 2 度の検討会を繰り返し、半年で HPF2.0 の問題点の洗い出し、不足機能の検討を行った。1997 年 1 月までに、主な HPF への拡張機能を固め、この時点から、日本の主な HPC ユーザを交えたおよそ 40 人で、JAHPF（Japan Association of HPF）がスタートした。約 2 年の議論を経て、HPF/JA 1.0 仕様[21]が定められた。拡張は、新機能の採用よりも、実用性を優先させ、ユーザの手軽な指示追加で高性能を得られるようなものを中心に採用した。主な拡張機能は、複数の SHIFT 転送を最適化するための

SHADOW 機能拡張、実用コードで現れる複雑な Reduction 並列化機能、通信スケジュール再利用機能、非同期データ転送機能などである。VPP Fortran や 3 社のコンパイラ開発の知見、JAHPF でのユーザコードの並列化実践による必要機能洗い出し、HPF+[22]などの欧米での研究成果などが検討のベースになった。ユーザプログラムの並列化もユーザ・コンパイラ技術者の共同作業として多数進められ、プラズマ流体コードや粒子コードなどで多数の成果を納めた。東京で開かれた HUG2000 がこの集大成である。2001 年から、仕様検討から、HPF の普及促進に重点を移す目的で、JAHPF は HPF 推進協議会（HPFPC、www.hpfpc.org 参照）として体制の刷新を行った。図 1 にこれら活動を年表形式にまとめておく。地球シミュレータ用の HPF コンパイラは SX 用の処理系 HPF/SX V2[23][24]を拡張して 2002 年に完成したが、これまで述べてきたコンパイラ技術開発、HPF 活動の集大成である。富士通からも HPF コンパイラが正式に製品化された[25]。MHD シミュレーションで大規模 VPP システムを用いて 300GFLOPS 以上の性能が達成された[26]。

HPFPC は現在も活動を継続しており、地道ながら新たな HPF ユーザの獲得や、フリーの PC クラスタ用処理系の開発、ドキュメント整備などの活動を行っている。HPF/JA 仕様を装備したコンパイラは NEC および富士通から製品化されており、多くの利用者の並列化負荷を大幅に軽減できる、安定した処理系になったと思っている。もう 5 年早く、現在のコンパイラが実現できていれば、HPF の現状はかなり変わったのにと思う。NEC も実は F77 ベースから F90 ベースに移行するのに、余計に 3 年を費やしている。他社も同じであろうと思うと、最初に F90 ベースを選択したことが残念である。

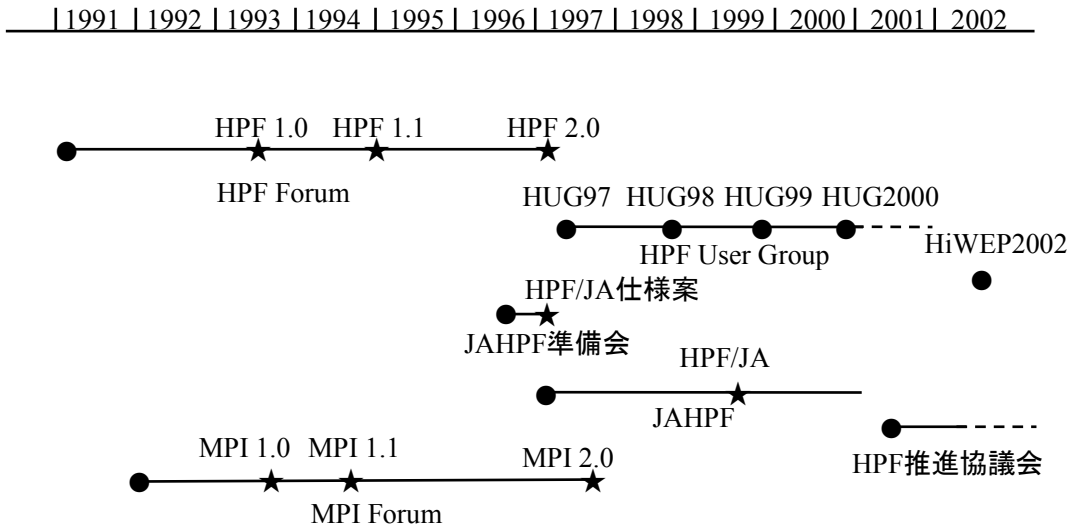


図 1 HPF についての各種活動の年表

3. 種々のプログラミングインタフェースの比較

3.1 MPI（Message Passing Interface）

メッセージパッシングは、プロセッサ間のデータ転送を、ユーザがライブラリを用いて明示的に記述する枠組である。SPMD(Single Program Multiple Data Stream)形式に基づいて、複数の制御の流れを意識したプログラミングが必要になる。

MPI(Message Passing Interface)は、分散メモリ型のプログラミングインタフェースとして、現在、最も標準的に用いられているものである。仕様は MPIF(MPI Forum)と呼ばれるプライベートなフォーラムで検討され、1994 年 6 月に MPI1.0 がまとめられた。その後、並列 I/O やプロセスの動的生成・消滅、1 方向通信(One Sided Communication)などの機能をさらに追加した MPI2.0[4]の言語仕様が 1997 年 7 月に定められた。MPI の詳細についてはホームページ http://www.mcs.anl.gov/mpi/ を参照されたい。

MPI は、現在、大半の商用並列マシン上でサポートされている。これらの MPI には Argonne 国立研究所と Mississippi 州立大学との共同で開発された MPICH[5](http://www.mcs.anl.gov/mpi/mpich/ 参照)をもとにしたものが多い。MPICH は仮想デバイスの概念を導入することにより、システム依存部がコンパクトに切り離された構造になっており、移植性が高い。ベンダは、システム依存部を対象ハードウェアに適するように開発することで、効率の良い MPI を実装することができる。

3.2 VPP Fortran

VPP Fortran は、もともと NWT(Numerical Wind Tunnel)をターゲットとして開発された言語であり、日本におけるデータ並列言語としては、もっとも成功したものである。指示行形式を中心に Fortran に最小限の拡張を加え、データマッピングと計算マッピングの両方を利用者に指定させる。言語拡張のそれぞれの要素を見ると、非常に HPF と非常に良く似た構成になっているが、言語の設計コンセプトはかなり異なる。もっとも大きな相違は、HPF がデータマッピングのみを指定させ、計算マッピングをコンパイラの自動処理に期待しているのに対して、VPP Fortran では、コンパイラの自動処理に期待するものは非常に少ない。データマッピングも計算マッピングも性能に影響を与えるものは、すべて利用者に明示的に指定させ、コンパイラに負担をできるだけかけずに、高い性能を得られることを目指して設計されている。

データマッピングについては、ローカルデータとグローバルデータという 2 種類が用意されている。グローバルデータは、HPF の分散データと同様に、分散メモリ上に block 分割または cyclic 分散を用いて配置することができ、プロセッサをまたがってのアクセスが可能である。ローカルデータは、プロセッサごとに重複して領域が用意されるもので、他

のプロセッサのデータアクセスはできない。グローバルデータへのアクセスは低速であるため、分散配置されたグローバルデータのうち、自プロセッサが持つ領域（オーナー領域）とローカルデータを Fortran の EQUIVALENCE 文で結合し、計算処理などで頻繁にアクセスする場合には、ローカルデータとしてアクセスすることが推奨される。

計算マッピングについては、複数プロセッサで実行すべき部分を PARALLEL REGION として指定し、この部分が SPMD 的に実行される。下記の例では、配列 a,b の分散を Partition D として宣言したものにしたがって指定し、さらに同じ分散をもとに SPREAD DO として DO ループのマッピングに指定することにより、計算マッピングとデータ分散の両方を指定している。

(VPP Fortran の例)

```
!XOCL PROCESSOR P(4)
      dimension a(400),b(400)
!XOCL INDEX PARTITION D=(P,INDEX=1:1000)
!XOCL GLOBAL a(/D(overlap=(1,1))), b(/D)
!XOCL PARALLEL REGION
!XOCL SPREAD DO REGIDENT(a,b) /D
      do i = 2, 399
        dif(i) = u(i+1) - 2*u(i) + u(i-1)
      end do
!XOCL END SPREAD
!XOCL END PARALLEL
```

3.3 Co-Array Fortran

Co-Array Fortran[*]は、もともと F--という名で呼ばれていたもので、CRAY が開発したデータパラレル言語である。Fortran95 に、データマッピングを記述するための簡単な言語拡張が施されている。プログラミングモデルは MPI と同様に SPMD であり、プログラムは image と呼ばれる、独立したメモリイメージを持つ実行ストリームがプログラムを重複して実行する。Fortran にデータマッピングを追加したという意味では、HPF と同じであるが、大きな違いは、HPF がシングル実行イメージであるのに対して、Co-Array Fortran は SPMD をベースにしている点である。言語拡張のポイントは、co-array と呼ばれる新たな配列を付加して、どの実行イメージ上のデータかを指定した配列アクセスを許すという点である。

(例)

```
REAL, DIMENSION(N)[*] :: X,Y
X(1) = Y(1)[Q]
```

この例では、配列 X、Y というサイズ N の co-array が定義されており、配列宣言に[*]が付け加えられることで明示的に co-array であることが示される。co-array は、すべての実行 image の上に宣言されたサイズ（この場合は N）の領域が重複して確保される。co-array へのアクセスにはローカルアクセスとグローバルアクセスの 2 通りの方法がある。グローバルアクセスとは、配列参照に[ ]指定を追加するもので、上記の例では、Y(1)[Q]の参照がこれにあたる。[ ]の中には、実行 image を指定し、この場合には、image Q の Y(1)が参照される。ローカルアクセスは、[ ]指定なしでの配列参照であり、この場合は左辺の X(1)がこれにあたる。ローカルアクセスは、実行中の image 上に確保された co-array をデータ転送なしでアクセスする手段であり、たとえば、実行中の自 image が me なら、X(1)と X(1)[me]は同じものである。ただし、X(1)[me]へのアクセスはデータ転送を含めたアクセスが仮定されるので、(コンパイラの最適化にも依存するが)性能が劣化することを覚悟する必要がある。

ローカル/グローバルという co-array へのデータアクセスは、非常に VPP Fortran のそれと近い。VPP Fortran でもローカルアクセスとグローバルアクセスを許すが、こちらはグローバル変数とローカル変数を別々に宣言して、この両者を equivalence 文で結合することにより実現している。

Co-array Fortran のもう一つの特徴は、並列化のモデルが SPMD ということである。image というのは、論理的実行単位（MPI での rank、HPF でいうところの論理プロセッサ）であり、その数は、必ずしもプログラムを実行するシステムのプロセッサ数と一致する必要はないが、簡単のために image 数とプロセッサ数が同一として議論を進める。SPMD モデルにおいては MPI と同様に、記述されたプログラムの複製が、すべてのプロセッサ上で非同期的に実行される。ある image が生成したデータを別の image で co-array を介して引用する際には、明示的に同期を挿入する必要がある。

3.4　Global Arrays

GA(Global Arrays)は、ライブラリ呼び出しの形で共有メモリビューを提供するインタフェースであり、Fortran や C、C++などの言語と一緒に用いることができる。MPI 処理系などのメッセージパッシングインタフェースの上にも実装されており、この場合、メッセージパッシングインタフェースと混在して用いることができる。

基本的な機能は、グローバル配列（GA）の宣言、および GA への一方向通信によるアクセス（put と get)、データ一致制御のための同期、総和などの上位機能、GA の格納アドレスのなどの情報取得機能である。機能的には、後述の Co-Array Fortran と非常に似ているが、これらの GA 操作をすべてライブラリ呼び出しの形式で実装されている点が異なる。メッセージパッシング関数が、プロセッサ識別子（MPI では rank）とアドレスの組によりデータ転送の対象を指定するのに対し、GA では、配列要素番号によるアクセスが可能であり、明示的にデータ格納先のプロセッサを意識する必要はない。具体的には、

nga_create(type, ndim, dims, array_name, chunk, g_a)

で、n 次元の GA を宣言でき、これに対して

subroutine NGA_Put(g_a, lo, hi, buf, ld)

subroutine NGA_Get(g_a, lo, hi, buf, ld)

で put/get 操作が可能である。

同期用のルーチンも用意されており、データ一致制御の必要に応じて挿入する必要がある。また、総和などの collective 通信を行うルーチンも用意されている。

3.5 OpenMP ＋ DSM

DSM（分散共有メモリシステム）システムとは、物理的に分散配置されたメモリを共有メモリとして構成したシステムであり、論理的にはすべてのアドレス空間を通常のロード、ストア命令でアクセスできる。ただし、物理的には分散配置されているので、自プロセッサに直接接続されたメモリへのアクセスは高速だが、他のプロセッサに接続されたメモリへのアクセスは低速となる。ハードウェアサポートを持つ分散共有メモリマシンと、ハード的には分散メモリシステムで、この上に純粋にソフトウェアだけで DSM を実現するソフトウェア DSM がある。

ソフトウェア DSM を実装し、この上で共有メモリ用の並列化インタフェースである OpenMP［4］を用いることにより、分散メモリシステムのプログラミングインタフェースを実現することができる。OpenMP は、主に、ループの各繰り返しや計算処理のまとまりを単位として、これらを別々のプロセッサで実行させることをユーザに明示的に指示させる。たとえば、ループを並列化する場合、ループを!$OMP OMP DO ～ !$OMP END DO で囲むことにより、各繰り返しを並列実行させることができる。

OpenMP は共有メモリが対象であるため、データの分散メモリ上への配置については、ユーザは指定できない。ある計算処理とそれに用いられるデータが同じプロセッサとメモリにあれば高速処理ができるが、これらを全く考慮しなければ、大半の計算処理でのメモリアクセスがネットワーク経由となり効率低下を招く[5]。このための制御（Affinity 制御

と呼ばれる）をコンパイラ、ユーザでどのように分担するか、またその際の言語インタフェースはどうすべきかが重要である。OpenMP を拡張して、DSM への配列のデータマッピングを指定される拡張や、First Touch という、最初にアクセスしたプロセッサメモリにデータを配置するような制御をコンパイラで行う方法が提案され、一部のコンパイラには実装されている。

4. プログラミングインタフェースの比較

前節で説明した種々の並列化インタフェースを比較すると、これらの違いのポイントとしては、下記のものが挙げられる。

(1) 並列化インタフェースの形式（言語拡張、指示行追加、ライブラリ呼び出し）

最も融通の利くのが、言語そのものに手を入れる言語拡張だが、開発コストは最も高い。指示行形式にすると、コンパイラに手を入れる必要がある点では、言語拡張と同じだが、既存言語と指示行言語の処理（たとえば構文解析処理など）を比較的切り分けられるため、開発コストが下がる。ライブラリ呼び出しは、コンパイラに手をいれずに済むため、もっとも手軽な実装が可能だが、ライブラリ呼び出しのオーバヘッドが必要になったり、ライブラリ化することで、コンパイラの最適化が阻害されたりする場合がある。

(2) 並列化指定（計算マッピング指定　AND/OR　データ分散指定）

計算マッピングとデータ分散指定をどのように切り分けるか。先に述べたとおり、分散メモリシステムの分類においては、言語機能そのものという意味では最も大きな違いが出る部分である。HPF の取ったデータ分散指定をさせるという考え方は、シンプルで筋の良いものだと思う。問題点は、手続き間の扱いであろう。Fortran90 できれいに書かれたコードは良いが、データマッピング指定は、コンパイラなどの処理系が配列の論理イメージと物理的アドレス上運のイメージのマッピングを管理する必要があるため、手続き間の扱いに弱い。最適化のために手続き間で異なる形状の配列として、Fortran の Reference 呼び出しをされると、コンパイラはお手上げとなる。これは GA や Co-Array Fortran、VPP Fortran のいずれにも当てはまる。データ並列言語を普及させるには、手続き間の配列をどのように扱うべきかというプログラミング方法論を確立し、ユーザが物理的メモリイメージを考慮してコーディングするケースを減らしていくしかないかもしれない。

(3) コンパイラの自動化への期待度

コンパイラの自動化にどこまでを期待するかによっても分類できる。特殊な例を除けば、

ライブラリ呼び出し形式の MPI などは、コンパイラに何も期待しないし、完全自動並列化はすべてを求める。VPP Fortran や GA や Co-Array Fortran も、コンパイラにほとんど期待しない形式になっている。最近、HPF と MPI の中庸を求める動きの中で、これらのインタフェースが再度注目を浴びている。

(4) インタフェースの美しさ

インタフェースの完備性、仕様の美しさもインタフェースのよしあしを決める上で大きな要素となる。シンプルな美を持つ仕様がもっとも望ましいが、特定ケースの対応ができず、実用上問題がでる場合が多い。かといって、実用性を追求していくと、インタフェースの肥大化や複雑化を招く。特定機能を追加することで、おもいもかけなかった副作用が他の機能に発生することも、よくある。HPF も、JAHPF も言語標準化活動において、大半のエネルギーを言語の完備性を求めることに費やしている。計算機言語は、要素機能の無限の組み合わせによりプログラムを記述できるため、ミスのない、きれいな開発は非常に困難である。これらの活動には、忍耐強い論理的思考能力が要求されるが、HPF では CM Fortran を開発した Guy Steel が、複雑な仕様を纏め上げる過程で大きな役割を果たした。活動の最終局面では、言語仕様の細部の詰めを、なかなか皆で追いきれず、Guy の能力に依存する局面が多かった。ある意味で、HPF は言語の完全性、美しさを追求しすぎたきらいがある。その点、VPP Fortran は、完備製や美しさをある程度犠牲にして、実用性を追求した言語であると言える。

5. HPF の今後について

利用者に最も大きな期待を寄せられた時期に、実用レベルのコンパイラを提供することができなかったことが最大の原因で、HPF は欧米では、忘れ去られようとしている。3 年から 5 年遅れた。この機を逸したことで、これから再び HPF が爆発的に利用が広まるということは考えにくいが、分散メモリマシンにおいて、データ分散配置を指定させ、残りをコンパイラが処理するという考え方は的を射たものだと思っている。今後は、HPF でコード開発・並列化を加速できるユーザを地道に見つけ、少しずつ発展させていくことになると思う。

並列化インタフェースとしては、当面は MPI が中心となることは間違いないが、MPI と似通ったプログラミングモデルでありながら、データパラレルの考え方を導入して記述量の削減を狙う GA や Co-Array Fortran も、インタフェースがシンプルで利用者の習得が容易なことから、今後の発展が期待できる。また、OpenMP にデータマッピング機能を付加するアプローチも面白いと思う[27][28][29]。MPI に比べて、とりあえず動く並列コ

ードを作成することが非常に容易である点を評価したい。並列化インタフェースは、そのもののよしあしよりも、どこで用いられるかが、その後の存続という意味では重要である。Linda という共有メモリスペースを提供する処理系は、ほとんどの人に忘れられているが、GAUSSIAN の並列化に利用されているというだけで、現在でもかなりのニーズがある。

HPF の今後についての最大の懸案は、言語やコンパイラの問題というより、Fortran で並列プログラム開発を行うという用途が、今後どれだけ広がるかにあると思われる。新規コード開発において、Fortran の利用割合が減っていることや、並列コード開発も、スクラッチから開発するより、既存の MPI プログラムを改造していくケースが多い。並列化によりユーザの研究自体が大いに加速されるような逐次コードをできるだけ多く発掘して HPF を用いて並列化し、その有効性を地道にユーザにアピールしていく活動を継続することが重要である。特定用途であれ何であれ、何とか HPF を一人前の市民権をもった言語インタフェースとして確立されていくことを願っている。

参考文献

[1] High Performance Fortran Forum. 1997. *High Performance Fortran Language Specification. Version 2.0.* Technical Report TR, Rice University, January 1997.

[2] H. Sakagami, H. Murai, Y. Seo and M. Yokokawa, 1*4.9 TFLOPS Three-dimensional Fluid Simulation for Fusion Science with HPF on the Earth Simulator*, SC2002, http://sc-2002.org/abstracts_papers/ab_paper17.html.

[3] Message Passing Interface Forum 1994. *MPI: A Message-Passing Interface Standard,* The International Journal of Supercomputing Applications and High Performance Computing, Vol.8, No. 3/4, pp. 165-416.

[4] http://www.co-array.org/welcome.htm

[5] http://www.emsl.pnl.gov:2080/docs/global/ga.html

[6] OpenMP Architecture Review Board, OpenMP Fortran Application Program Interface, Version 2.0, November 2000. (http://www.openmp.org)

[7] Lenoski, D., et. al., *The Stanford Dash Multiprocessor*, IEEE Computer, 25(3), March 1992, pp. 63-79.

[8] J. Laudon and D. Lenoski, *The SGI Origin ccNUMA Highly Scalable Server*, SGI White Paper, March 1997.

[9] 細見他、*並列計算機 Cenju-4 の分散共有メモリ機構*、情処論文誌 Vol.41 No.05-17, 2000 年 4 月.

[10] High Performance Fortran Forum. 1997. *High Performance Fortran Language Specification. Version 2.0.* Technical Report TR, Rice University, January 1997. http://www.crpc.rice.edu/HPFF/

[11] High Performance Fortran Forum and RIST 1999. *High Performance Fortran 2.0 (In Japanese)*, Springer-Verlag Tokyo, ISBN4-431-70822-7.

[12] http://www.nas.nasa.gov/Groups/HSP/UserGuide/

[13] http://www.csm.ornl.gov/pvm/pvm_home.html

[14] http://www.cs.rice.edu/~dsystem/

[15] http://www.labri.fr/Perso/~counilh/adaptor/

[16] T. Kamachi, et. al., *Generating Realignment-Based Communication for HPF Programs*, IPPS '96, pp.361-371, 1996.

[17] Barbara Chapman, Piyush Mehrotra, Hans Zima, *Programming in Vienna Fortran,* Scientific Programming, 1992, http://www.par.univie.ac.at/research/report/node15.html

[18] R. Das, J. Saltz, R. von Hanxleden 1993, *Slicing Analysis and Indirect Access to Distributed Arrays*, Proc. of the 6th Workshop on Languages and Compilers for Parallel Computing, Portland, OR.

[19] http://www.vcpc.univie.ac.at/information/HUG/

[20] Fujitsu Ltd., *VPP Fortran Programming, UXP/V, UXP/M Ver. 1.2*, April 1997.

[21] Japan Association of High Performance Fortran, 1999, *HPF/JA Language Specification Version 1.0*, RIST (English version is available at http://www.tokyo.rist.or.jp/jahpf/index-e.html).

[22] Siegfried Benkner, Erwin Laure, and Hans Zima. 1999. *HPF+: An Extension of HPF for Advanced Industrial Applications.* Technical Report TR 99-1, Institute for Software Technology and Parallel Systems, University of Vienna. http://www.par.univie.ac.at/project/hpf+/

[23] H. Murai, et. al., Implementation and Evaluation of an HPF Compiler for Vector Parallel Machines, HPF/SX V2, 4th HPF User Group Meeting, Oct. 2000, Tokyo.

[24] K. Suehiro, et. al., Benchmarking Results of HPF/SX V2, 4th HPF User Group Meeting, Oct. 2000, Tokyo.

[25] Fujitsu Ltd., *HPF Programming: Parallel programming (HPF version)*, UXP/V Ver. 1.0, June 2000.

[26] T. Ogino, *Global MHD Simulation Code of Earth's Magnetosphere Using HPF/JA*,

4th HPF User Group Meeting, Oct. 2000, Tokyo.

[27] Barbara Chapman, *HPF Features for Locality Control on ccNUMA Architectures*, HUG2000 Invited Presentation, Tokyo, Oct. 2000.
http://rist03.tokyo.rist.or.jp/jahpf/hug2000/presen/Chapman.pdf

[28] 廣岡 孝志、太田 寛、菊池 純男、ファーストタッチ制御による分散共有メモリ向け自動データ分散方法、情処論文誌 Vol. 41 No.05 – 020, 2000 年 4 月.

[29] K. Kusano, S. Satoh and M. Sato, *Performance Evaluation of the Omni OpenMP Compiler*, WOMPEI 2000, Oct. 2000, Tokyo.
http://pdplab.trc.rwcp.or.jp/Omni/home.ja.html

3.3.3 Software Design and Productivity (SDP) Coordination Group が目指すもの

近山 隆 委員

1. はじめに

本稿では米国の政府機関である Interagency Working Group for Information Technology Research and Development (ITRD) 傘下の Software Design and Productivity (SDP) Coordinating Group の活動の方向性を定めるため、2001 年 8 月 18 日～19 日に Arlington において行なわれた“Planning Workshop on New Visions for Software Design and Productivity”について紹介する。

2. 活動の枠組と経緯

ITRD の活動の発端は 1991 年に設置された President's Information Technology Advisory Committee（PITAC）に遡る。PITAC は全米の大学・研究所・情報産業の有識者から構成され、大統領・国会・連邦政府機関に対し IT における米国の優越を保つ方法を直接答申する、他の政府諸組織とは独立した調査機関である。

PITAC は 1999 年 2 月に、現状を分析した調査結果として、以下のような事項を報告している。

- ソフトウェア需要は生産能力を上回っている
- 国家は脆弱なソフトウェアに依存している
- 信頼できる安全なソフトを作る技術が不十分である
- ソフトウェア基礎研究への国家投資は不足している

その上で PITAC は「連邦政府はソフトウェアの基礎研究に absolute priority を与えるべきである」と結論付けており、より具体的には以下のような政策の推進を提言しており、これが ITRD の活動の根拠となっている。

- ソフトウェア開発メソッド・要素技術の基礎研究に投資すべきである
- 諸領域においてソフトウェアの国立ライブラリを提供すべきである
- あらゆる情報技術研究イニシアチブにおいて、ソフトウェア研究を重要な要素とすべきである
- ヒューマンコンピュータインタフェースの基礎研究に投資すべきである

3. Planning Workshop

Arlington で行なわれた Planning Workshop は、PITAC の答申に沿った研究開発につ

いてより具体的な方向を探るためのもので、大学、NSF, DARPA, DOE, 産業界から、第一線の研究者を集めて開催された。

ワークショップは二日間に亘ったが、第一日は導入セッションに続き 4 領域についてのパネル、第二日は朝に各パネルに対する課題が示された後、パネルごとに分かれての検討が行なわれ、昼にはその結果を報告する、という日程であった。各パネルの領域は以下の通りである。

- ソフトウェアの将来とソフトウェア研究
- ソフトウェア開発の新パラダイム
- 実世界のためのソフトウェア
- ネットワーク中心の分散ソフトウェア

3.1 導入セッションと各パネルへの課題

導入セッションでは上述の背景経緯説明と共に、主要な研究領域として以下のものが示された。

- ソフトウェア
- スケーラブルな情報インフラ
- ハイエンドコンピューティング（ソフトウェアの研究開発を含む）
- 社会経済的および総労働力上の影響

その上で、検討すべき主要なポイントとして、下記が示された。

- 解くべき基礎的な問題は何なのか
  「Java をセキュアにする」など、技術的詳細ではなく
- 真の障壁と挑戦は何なのか:
  「絶対の安全」「完璧な保証」など、非現実的な課題ではなく
- どの方向がもっとも有望か:
  「究極の形式仕様記述」などの理想論ではなく、現実的技術では?
- 何をもって理想的な成果とすべきか:
  無限の生産性、でも最低の品質、では困る
- どのように研究を進めるべきか:
  「理論上の力技」とか「永遠に作っては壊しの繰返し」では困る

そして、「真の障壁と挑戦」として「無限の複雑性を知的に制御すること」をあげている。つまり、現実のシステムは、

- 分散し、
- 不均一で、

- ハイブリッドで、
- コンポーネントは信頼できない、

ということを覚悟した上での対処法を求めている。

各パネルへの要請として、Software Design and Productivity (SDP)の研究計画を策定することを求め、そこではこの「真の問題」への挑戦を、遠い将来の理想的への道を探るのではなく「明日の問題」を解くことを求めている。

第一日はこの導入セッションの後に４パネルが開かれ､参加者は各々の所信を発表した。その内容は参加者の多様な研究領域を反映した多岐に渡るものであるが、ここでは個別には立ち入らないこととする。

第二日にはパネルごとにその検討結果のとりまとめを行なったが、それに先立ってとりまとめにおいて留意してほしい事項として、以下が掲げられている。

- ソフトウェアプロセス・ソフトウェア作成の基盤となる科学は何か
- サイズ、複雑さ、合成可能性、試験可能性などの根本的限界はあるのか
- 生産性向上のために評価すべき研究項目は何か
  - 要素技術?　高度な抽象化?　利用者教育?
  - オープン SW?　開発プロセスの自動化?
  - 新たな開発アプローチ?
- どうすれば変化に対応するソフトウェアを作れるか
- 大量の既存コードを（非難するのではなく）うまく利用するにはどうすればよいのか
- サイバーワールドの「都市計画」はどうすべきか
  荒廃したスラムや浅薄な歓楽街ばかりの街にはしたくない
- これまでの基本仮定を崩すような新技術は出てくるのか
- 産業への技術注入の障壁を低くするにはどうすればよいか

その上で、連邦予算を 5 億ドル投入すべき研究方向を検討するよう求めている。

以下では、二日目昼のセッションで、各パネルの結論として提示された内容について概説する。

3.2 ソフトウェア研究の将来

「ソフトウェア研究の将来」についてのパネルでは、まず現状の分析として、以下をあげている。

- 現在のインフラはまったく不適切である
  信頼性、安全性、可用性のどの面をとっても低い

- 過去の大規模ソフトウェアプロジェクトの半数は失敗であった
- 応用ソフトウェア同士が情報をやりとりできない状況にある
- 過去のソフトウェア資産は 1 兆ドル相当もある

その上で、以下の目標を掲げた研究をすべきであるとしている。

- 問題領域の（コンピュータの、ではない）専門家が使いこなせるソフトウェアの実現
- 既製品ソフト業界の共通基盤となるアーキテクチャの開発
- ソフトウェア産業に対し、鍵となる概念･技術を提供
- プログラマの意図を機械が理解できる形で記録する方法を開発
- ソフトウェア開発の現状をしっかり認識

特に注目すべきは以下の領域であるとする。

- ネットワーク埋め込み型
- 対人インタラクション
- グループインタラクションの計算機支援
- 問題領域専門家のための環境

  教員; エンジニア; 金融; 科学者

そして、この基礎となる科学として、以下をあげる。

- ソフトウェア物理: 並行処理のモデル、など
- ソフトウェア経済学: ソフトウェアの柔軟性の価値を定量化
- ソフトウェア行動科学:
  - 人間の認知・記憶に関する知見をソフトウェア設計に応用
  - ソフトウェアと人間の適切な役割分担

鍵となるポイントは以下であるとする。

- ソフトウェア開発のプレイグラウンドを変えよう
  - 「プログラム」の再定義
  - 「プログラマ」の再定義: 開業医などと同様の専門家として
- 鍵となる進歩の提供
  - 軽い形式的枠組
  - 並行性･耐故障性･安全性などのモデル
  - 「価値」に基礎を置いたデザイン
  - 分散オープンソースウェアテスト方式
- ソフトウェアの経済学を変えよう
- ハードウェアをソフトウェア側から定義しよう（現在は逆）

3.3 ソフトウェア開発の新たなパラダイム

このパネルでは、まずソフトウェアの複雑が指数的に増大していることを指摘する。

- 組込型、モバイル、浸透性、大域性
- 長寿命、複数バージョン混在
- 共同開発
- 利用者ごとのカスタマイズ

そして、これらの問題は「ゴルディオスの結び目」である、とする。「ゴルディオスの結び目」とは、フリージアの王ゴルディオスの戦車の轅（ながえ）と軛（くびき）をつないだ非常に複雑な結び目であり、アジアを支配する者のみがこれを解くという神託があったが、アレクサンダー大王がこれを剣で両断した、とされるものである。すなわち、従来の考え方に沿った漸進的な改良では本質的な解決には至らず、なんらかのまったく新しいアプローチが必要な問題である、というわけである。しかし残念ながら、どのような剣をもって、どのように切ればよいか、という具体像を示しているわけではない。

PITAC のソフトウェアの国立ライブラリの提言に対しては「国立ソフトウェア考古学センター」の設立を提言している。そこでは、ソフトウェアの古典を Web 上に公開するもので、古典的価値のあるソフトウェアの蒐集と保存、その芸術的本質のリバースエンジニアリングを行なう、としている。こうした活動を通じ、ソフトウェア分野における芸術的価値を持つ表現を奨励し、また既存技術を特許化することで、将来の研究の基礎ともなる、ソフトウェアの基本構成の宝庫を構築しようとするものである。これはいわばソフトウェア版のゲノム保存プロジェクトとでも呼ぶべきものである。

ソフトウェアを見るにあたって、その多面性に注目すべきである、ともしている。すなわち、ソフトウェアは多様な異なる意図を表現したものであり、ソフトウェアに関するこれまでの技術進歩の多くは、こうした意図のある側面をよりよく理解した、あるいは、ある側面を上手に表現した、というものである、とする。新たなシステムは共存する異なる視点が互いに影響しあって生まれるものであるが、従来はこのことについての認識が不足していた、とする。これに関連する研究分野としては:

- デザインパターン
- 協調開発環境
- アスペクト指向プログラミング
- 特定領域用言語

があげられる。こうしたソフトウェアの多面性に着目した研究においてなすべきこととして:

- どの側面が肝要か
  性能、価格、使い勝手、過去からの連続性、などの側面がある
- 諸側面をどう表現するか
- 諸側面をどう変形・統合しシステムとするか
- 特定領域用言語のためのツール整備の加速

をあげている。多面的なソフトウェアを構築する上で役立つ機構として、設計上の選択とトレードオフの明確化をあげている。この明確化によって、開発プロセスのメリットはそのままに、システム設計を考えるようにしむけることができる。また、ソフトウェアに対するさまざまな要請に対し、他の要請と独立に考えることを容易にしながら、他の側面との折り合いを明示的につけられるような機構も重要である、としている。

協調開発環境としては、Web を仮想会議場としてのソフトウェア開発が今後進むであろうことを予想し、ソフトウェア開発の社会学、ソフトウェアプロセスの利用、考古学センターや多面的ソフトウェアを利用するプラットホームの創造が研究課題として重要であるとしている。

3.4 実世界のためのソフトウェア

このパネルも、まず現状分析を行なっている。挙げられているのは以下の点である。

- 今日のソフトウェアは多くの問題点を持っている
  - 偶然的複雑性、予測不能性、合成不能性、脆弱性を持っているため、実世界との相互作用に向かない
- その根本原因は、ソフトウェアが「コードの集まり」という考え方にある
  - 大規模システムの構築には人力を注ぎ込む
  - このため、システム全体像の理解が喪失される
  - 仕様とその実現が遊離してしまう

そして近未来の重要な問題として、今後十年程度のうちに、いたるところにでもコンピュータがある、という状況になるに違いないのにもかかわらず、今日のようなソフトウェアの作り方をしていたら、世界は危険極まりない場所になるだろう、逆にこれを回避しようとすれば、新技術の実施に支障をきたすことになるだろう、と警告している。

問題の根本は、人間が以下を表現するモデリング言語がないことにある、とする。

- 複雑な機能の実現
- 設計の理解
- 質問の定式化
- 振舞いの予測

そして、根本的な問題はプログラムコードでも「仕様」でもなく、「モデル」あるいは「設計」の問題である、という。そこで、こうしたモデリングを行なう言語が重要な研究対象であり、より具体的なテーマとして以下を挙げる。

- システムのモデリングのための「言語」
- 有用な抽象化を見つけること
- 計算的システム理論
- 互いに組合せて使える抽象化技法
- 時間・並行・能力などの表現方法

また、現状ではシステムを組合せて利用する組織的方法が欠如していることが、大きな問題である、とする。この問題を解決するに必要な要素技術として、以下を挙げる。

- 要素システムの枠組
- 要素システムの組合せに対する意味論
- その場での（on-the-fly）システム合成
- 既成のソフトウェア要素との統合

より具体的な研究テーマとしては、以下を挙げている。

- 意味の枠組と理論
- 方法論とツール
- 実験台と挑戦的問題
- 参照実装（reference implementation）
- アーキテクチャ枠組の定義
- 分散・分割の戦略
- 粒度とモジュラー性の制御戦略

また、これらの背景となる技術として、異なる軸での抽象化の間の変換についての理論の欠如をあげている。必要な技術としては、以下である。

- 異なる抽象化の間の関係の解明
- 抽象記述の生成器（異なる抽象化からの変換器）
- 複数の視点を許す抽象化（multi-view abstraction）
- 物理的環境の抽象化
- 証明つきの変換

具体的な研究テーマとしてあげているものは、以下のものである。

- オープンな生成器・変換器のインフラ（方法論、ライブラリ）
- 生成器の理論的基礎付け
- 抽象化変換器のパラメタ化

過去のソフトウェア資産の扱いについても、まず過去の遺産を扱う方法論の欠如を第一に挙げている。すなわち、システムを徐々に現代化する方法や、新しいものを古いものと統合する方法論の欠如である。具体的な研究テーマとしてあげているのは、以下のものである。

- 遺産コードの部品化
- 遺産からの抽象化の抽出
- 漸次現代化、リバースエンジニアリング
- 再設計を安価に ⇔ ソフトウェア遺産からの進化

3.5 ネットワーク中心の分散ソフトウェア

最後のパネルは、ネットワークを中心とした分散ソフトウェアについてである。ここでも、まず近未来の状況分析から入る。

- なにもかもがコンピュータであり、
- なにもかもがネットワークコンピュータであり、
- すべてが相互依存する可能性をもち、
- 実世界と接続し、
- ますますヘテロに、
- 接続はますます多様で不安定に

なるであろうと予測する。

こうした状況での分散ソフトウェアの応用として、以下のような多様な例を挙げる。

- 交通制御: 自動車数千台からセンサデータを入手し、利用
- 戦域管理: 敵対環境で多様な粒度の調整を行なう
- サプライチェーン管理
- 科学データの共同解析
- 大規模なデータ交換システムで、数千ユーザ、数千資源に対する、ソフトリアルタイムでの問合せ最適化
- 家庭内電力管理

これらを実現する技術的な課題としては、以下を挙げる。

- 資源を意識したプログラミング

  サービスクオリティ制御、時間、相互依存関係、エネルギー消費、大きさ、などを意識する
  - 現状では工学的トレードオフを考慮する計算モデルが欠如
  - 複雑なシステムではネットワーク端点間の特性の把握が困難

- 動的な資源の振舞い: 故障･不均一性 等
  - ここでも資源に関する計算モデルや端点間特性の欠如が問題
- ソフトウェア遺産の扱い
  - 設計時に想定しなかった利用形態への対応が重要
- 危機・信用・制御の管理
  - ポリシー･保障･管理のドメイン設定
  - 極度に動的なシステムの安全性確保と確認手法
  - プライバシー確保
- スケールの問題
  - 構成要素数; 構成要素の大きさ
  - 単一の計算に関係する要素数
  - ネットワーク中心システムは長時間無停止動作の要請あり

そして､これらの技術を実現するための研究方向とアプローチとして､以下を提言する。

- 「問題」を扱うプログラミングモデルの研究
  - 工学的トレードオフの扱い
  - 大規模システムの運用手法
  - 動的な資源の振舞いの解析
  - 危機・信頼・安全の管理技法
- 「問題」の基本機構の研究
  - 資源のトレードオフ: QoS、実時間性、等
  - 適応的振舞い
  - さまざまな次元への拡大
  - 分散管理
- 「問題」の全体を扱える、多レベルの資源管理
- ネットワーク中心の（小規模な）要素を、動的に大規模システムに合成する技術

こうした研究の成果が与える効果としては、以下があるとする。

- 社会的な効果
  - 現状では作れない規模のものを作れるようになる
  - ネットワーク化システムの信頼性・制御性が向上する
    工学的基礎がしっかりした設計による実現は、デバッギングによる実現よりもはるかに信頼性・制御性を高くできる
  - ソフトウェア開発コストの抑制

- 教育への効果
  - 新たなソフトウェアの作成・システムの設計にあたれる人材の供給

また、研究へのアプローチとして、以下が重要であるとしている。

- チームワークを持った研究は必須である
- 本当の問題を見つけ、中間成果を検証するには、大規模な検証プロジェクトが必要である
- 国際協力が望ましい

4. おわりに

この報告をまとめる上で感じたところを述べて、本稿の結語としたい。

このワークショップは基本的には予算計画のためのものであり、種々の提言も技術的には必ずしも新規性の高いものではない。しかしながら、真剣に技術的な諸側面を議論した様子がうかがわれ、その結論には強い方向性がある。

共通に感じられるのは、抽象化を武器に複雑性･規模･不均質等の実世界の問題に挑戦しようという姿勢である。これは、従来乖離していたソフトウェアの理論と実践を止揚し、新たな枠組を構築しようとするものである。容易には解決の糸口さえつかめない実世界の複雑な問題に直面したとき、研究者は往々にして匙を投げ、理想化・単純化した問題のみを扱う象牙の塔の中に引きこもろうとする傾向が見られるものであるが、SDP ではそうした傾向に厳しい態度をもって臨んでいる。プロジェクトでは実施に当たる可能性が高いと見られるこのワークショップ参加者も、学術的な世界に閉じた研究を提案することはしていない。逆に、理論的な基礎付けなく、ソフトウェアを開発するような提言も行なわれておらず、むしろそうしたアプローチではカオスを招くことを強く意識した提言となっている。

具体的な研究方向を策定する上で、このワークショップは多くの示唆に富む提言を行なっているように思える。筆者はその詳細な検討の任に堪えないが、強く感じられたのは、このような検討の場が連邦政府の研究予算の使途策定の場であると同時に、一線の研究者間の意識合わせの場でもあったに違いない、ということである。2 日間という短期間ではあっても、一線の研究者のみが集まり合宿して行なう交流は、非常に密度の高いものであろうと推察できる。最終的な研究予算配分に至る前段階として、このような集中的な検討の場が設けられることは、研究を適切な方向性をもって活性化上で重要な役割を果たすであろう。

3.4 応用システム＆応用分野

3.4.1 センサ・データ処理の高速化

佐藤 裕幸 委員

1. はじめに

レーダなどのセンサからのデータ処理と言えば、従来は、DSP（Digital Signal Processor）を用いた処理や FFT（Fast Fourier Transform）などの信号処理が中心であった。すなわち、信号レベルでのデータをいかに高速に処理するかが課題であった。一方近年では、信号レベルではなく、より抽象度の高いデータを処理することが多くなってきている。例えば、観測された信号データを位置情報として扱い、目標追尾を行うことが挙げられる。目標追尾とは、レーダ等のセンサの観測結果から目標の航跡を抽出し、位置や速度等の運動諸元を推定する問題である。目標追尾の抱える課題の１つに、追尾目標の近傍に他の目標やクラッタ等の不要信号が存在する高密度環境下での追尾性能の確保が挙げられる。高密度環境下では、センサから得られた観測値のいずれが各追尾目標のものであるかを判定する相関処理が重要となる。この相関処理は、いわゆる組み合わせ問題であり、従来の情報処理の分野で用いられてきた手法が適用でき、それらを用いて高速化・並列化が試みられている。

ここでは、この追尾処理の並列処理を用いた高速化について紹介する。

2. 目標追尾とは

目標追尾とは、レーダ等のセンサの観測結果から目標の航跡を抽出し、位置，速度等の運動諸元を推定する問題である。目標追尾の抱える課題の１つに、追尾目標の近傍に他の目標やクラッタ等の不要信号が存在する高密度環境下での追尾性能の確保が挙げられる。高密度環境下では、センサから得られた観測値のいずれが各追尾目標のものであるかを判

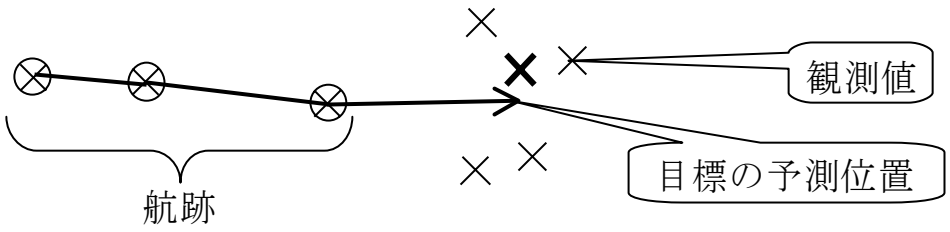


図 1 NN(Nearest Neighbor)方式による目標追尾

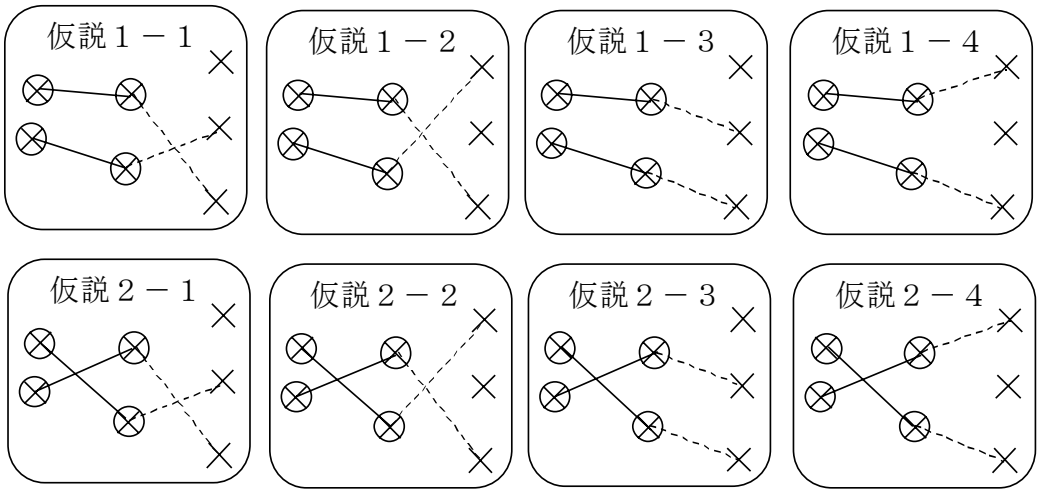


図 2　航跡と観測値の組み合わせ

定する相関処理が重要となる。従来は、各追尾目標に対して、予測位置に最も近い観測値を選択し、追尾計算に使用する方式が採られていた（図 1 参照）が、この方式では高密度環境下では追尾性能が不十分であった。そこで、より高度な相関方式を採用した航跡型 MHT（Multiple Hypothesis Tracking）[1] が提案された。

この航跡型 MHT では、観測値の時系列データから成る航跡のいずれが追尾目標であるかについて、他の目標やクラッタが存在する可能性及び追尾目標が探知されなかった可能性も考慮して、複数の仮説を構成しながら追尾処理を行う（図 2 を参照)。このように複数の仮説を保持しながら処理を進めていくため、追尾精度は優れるが、処理負荷が高く、また仮説の数が組合せ爆発を起こし膨大になる危険性があるという問題点があった。そこで、仮説数を抑制するために、信頼度の高いものから順に固定数の仮説しか生成しない N ベスト探索アルゴリズムを用いることで、航跡型 MHT が高速化されている[2] [3]。

この N ベスト版航跡型 MHT において、追尾精度を向上させるには、N の値、すなわち仮説生成上限数をできるだけ大きくしなければならない。そのためには、生成仮説数を抑えた N ベスト版航跡型 MHT においても、処理の高速化は重要な課題である。そこで、並列処理により、N ベスト版航跡型 MHT の高速化を行っている。

3. 航跡型 MHT の概要

図 3 の処理フローに基づいて、航跡型 MHT の概要を述べる。

3.1 観測ベクトルの入力

1 サンプリング時刻の観測ベクトル（センサによる目標の X,Y,Z 座標など）を入力する。

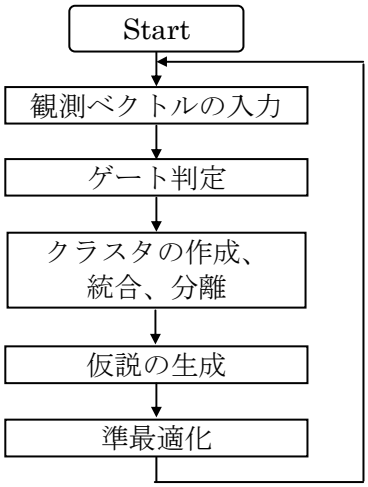


図 3 航跡型 MHT の処理フロー

3.2 ゲート判定

各サンプリング時刻から高々一つの観測ベクトルを選んだ時系列データ（0 観測ベクトルは、目標が探知できない場合を想定）を航跡と呼ぶ。1 サンプリング前のどの航跡とどの観測ベクトルが対応しているのかの相関処理を行う範囲をゲートと呼ぶ。すなわち、各 1 サンプリング前の航跡に対して、現サンプリング時刻の観測ベクトルが存在すると予測される空間の領域をゲートと定め、この領域の観測ベクトルのみ相関処理の対象とする。

3.3 クラスタの作成、統合、分離

互いに観測ベクトルを共有しない航跡の集合をクラスタと呼ぶ。すなわち、航跡 T_i と T_j が少なくとも一つの観測ベクトルを共有する場合に限り、航跡 T_i と T_j を類似航跡と呼び、T_i～T_j と書く。航跡 T_i, T_j において

$$T_i=T_{i1}\sim T_{i2}\sim\cdots\sim T_{in-1},\ T_{in}=T_j \tag{1}$$

となる T_{i1}, T_{i2}, …, T_{in} が存在する場合に限り

$$T_i\equiv T_j \tag{2}$$

と定義する。各サンプリング時刻の全航跡は、式(2)の関係により、互いに素な複数の部分集合に分けられ、この部分集合をクラスタと呼ぶ。以下の 3.4、3.5 の処理は、各クラスタ毎に独立して処理することが可能である。

どのクラスタにも属さない観測ベクトル（どの航跡のゲート内にも入らない観測ベクトル）がある場合は、新たにクラスタを生成する。また、異なるクラスタに属する複数の航跡のゲート内に同じ観測ベクトルが入っている場合、それらのクラスタの統合を行う。更に、後述の準最適化処理により過去のデータを捨てた結果、互いの航跡が共有する観測ベ

クトルを持たなくなった場合は、クラスタを分離する。

3.4 仮説の生成

各クラスタにおいて、0 個以上の航跡を選択して構成した航跡の集合を仮説と呼ぶ。各仮説は、クラスタ内のどの航跡の組合せを選ぶのが正しいのかを表わすものであり、最終的に求めたい目標の位置、速度等の運動諸元の基となる解候補である。各仮説を構成する際の基準は、同一仮説内に属する複数の航跡間で観測ベクトルを共有しないように航跡を選択することである。なお、仮説に含まれるどの航跡にも属さないクラスタ内の観測ベクトルは、この仮説において誤信号と扱ったことに相当する。

仮説を生成するには、まず現サンプリング時刻における観測ベクトルの入力を反映した航跡を生成して、それらを選択する。これは、各観測ベクトルが、どの既存の航跡と対応するのか、新たな航跡なのか、誤信号なのかを表わす選択木として考えることができる。

以下に、例を用いて説明する。1 サンプリング前のある仮説の航跡が T_1 及び T_2 であり、現サンプリング時刻では、航跡 T_1 がゲート内に 2 つの観測ベクトル Z_1 及び Z_2 を捕らえ、

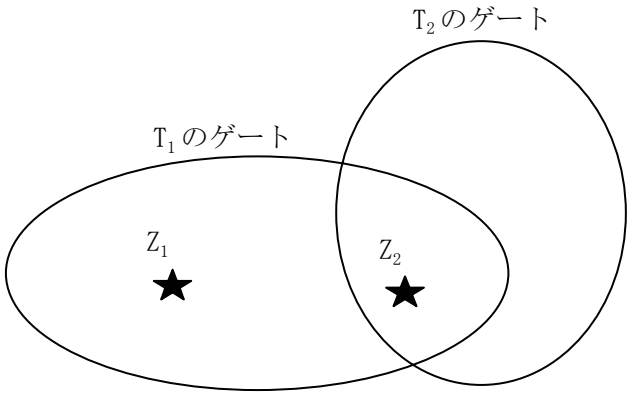


図 4 ゲートと観測ベクトル

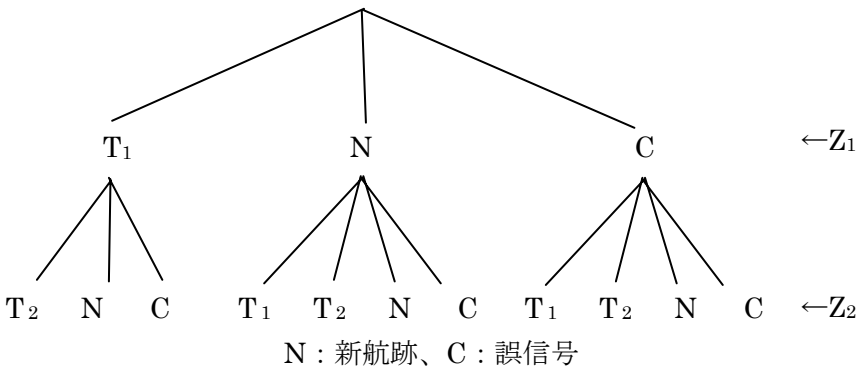


図 5 航跡と観測ベクトルの組合せ

航跡 T_2 が 1 つの観測ベクトル Z_2 を捕らえている（図 4)。この状況下での航跡と観測ベクトルの可能な組合せは、図 5 に示すような選択木で表わすことができる。この木の根から各葉までを辿る枝がそれぞれの仮説であり、この例では 11 個の仮説が生成されることになる。

1 つの親仮説から生成される子仮説の数は、その仮説内の航跡と観測ベクトルとの可能な組合せ数になるので、航跡数や観測ベクトル数が少しでも多くなると、直ぐに組合せ爆発を起こしてしまう可能性がある。そこで、仮説の生成段階で信頼度の高いものから所定数しか生成しないようにしている。

前述のように、考えられる仮説は図 5 のように選択木で表わすこともできるが、各列を誤信号、既存航跡、新航跡、各行を観測ベクトルとし、各要素を信頼度（相関度）とする行列とし、互いに同一の列からの要素を選択しないように各行について 1 要素ずつ選択することでも、仮説を表わすことができる。そして、選択した各要素の値の総和が最も大きくなるように選択する方法を見つけることが、最も信頼度の高い仮説を生成することになる。このような選択問題を効率的に行うアルゴリズムとして Munkres のアルゴリズム[5]がある。更に、最高信頼度仮説を生成した後は、選択した枝を境界とする部分木に分割する。例えば図 6 において、太い枝が選択された仮説であるとすると、その枝を境界とする部分木は、四角で囲った 2 個所となる。そして、それぞれの部分木で最高信頼度仮説を生成した後、それらの中で最も信頼度が高い仮説を選択する。これを仮説数が規定数 N になるまで繰り返す。この方法を Murty アルゴリズム[6] と呼び、これにより効率的に高信頼度の仮説から生成することができる。

なお、仮説の信頼度、すなわちその仮説が真である確率は、文献[4] に詳しいが、目標の探知確率、ゲート内捕捉確率、誤信号発生率、新目標発生率、仮説を構成する航跡と対応する観測ベクトルとの相関度及び基となる親仮説の信頼度の項から構成されている。

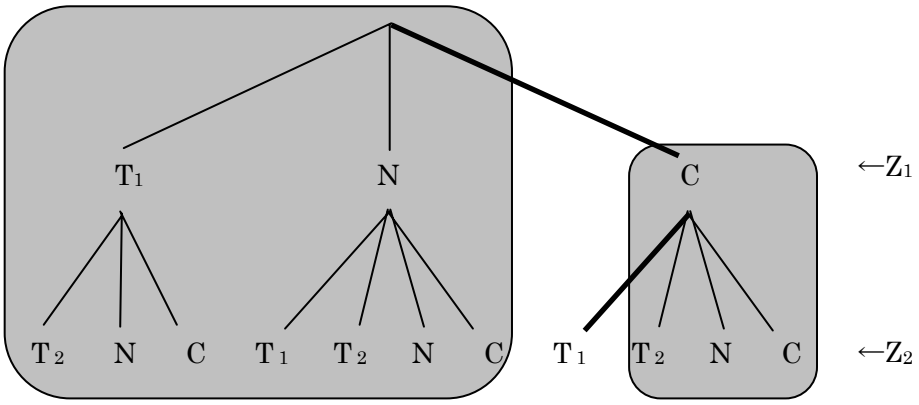


図 6 選択木の分割

3.5 準最適化

古いデータを捨てることにより、最新 N サンプリングの観測ベクトル構成が同一である航跡を同一視した結果、内容（構成する航跡）が同一となった仮説同士を統合する。これを N スキャンリミットと呼び、仮説数の抑制を信頼度ではない別の観点で行っていることに相当する。

4. 航跡型 MHT の並列化

航跡型 MHT では、個々のデータ（観測ベクトル、クラスタ、航跡、仮説など）毎に独立して処理しているのではない。従って、全体を統一的に並列化することはできず、それぞれの処理を個別に並列化するしかない。そこで、どの処理に時間を要しているのかを解析し、最も負荷の高い処理から並列化することにした。

4.1 負荷解析

図 7 に各サンプリング時刻毎の実行時間（左軸で棒グラフ）と仮説生成の全実行時間に対する割合（右軸で折線グラフ）を示す（生成仮説上限数は 3,000 個）。棒グラフの黒い部分が仮説生成の時間であり、圧倒的に仮説生成に時間を要していることが分かる。実行時間を要しているサンプリング時刻 24 秒以降の仮説生成時間の割合の平均は、92%である。なお、「その他」の内で最も負荷の高い処理は、クラスタ統合である。クラスタ統合では、お互いのクラスタ内の仮説同士を組合せるので、組合せ数が多くなると処理時間がかかっている。

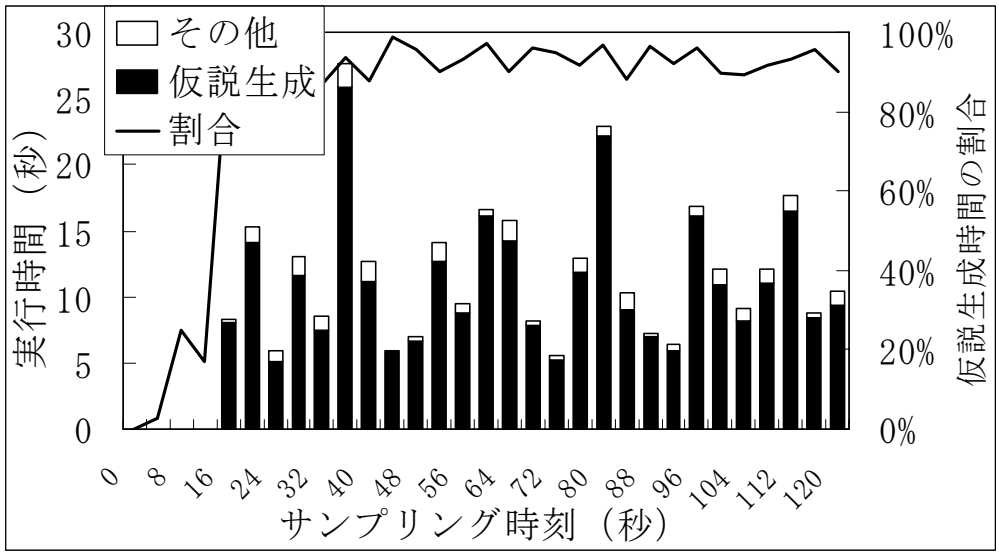


図 7 各サンプリング時刻毎の実行時間

以上の負荷の解析結果から、仮説の生成部分を並列化することにした。逐次の航跡型MHTプログラムにおけるこの仮説の生成方式は、以下の通りである。

(a) 全ての親仮説について、それぞれの最も信頼度の高い子仮説を生成する。

(b) N 個の親仮説から生成された N 個の子仮説の中で最も信頼度の高い子仮説を採用する。採用された子仮説を生成した親仮説は、次に信頼度の高い子仮説を生成する。

(c) (b)の処理を子仮説が N 個になるまで繰り返す。

4.2 並列化方式

この仮説生成において、各親仮説で子仮説を生成する過程は独立して実行できる。そこで、その親仮説毎の子仮説生成の部分を以下にように並列化する（図 8)。なお、子仮説を生成するプロセッサをクライアントと呼び、各クライアントで生成された子仮説を信頼度の高い順にマージするプロセッサをサーバと呼ぶことにする。サーバの処理は子仮説のマージだけであり負荷が低いので、サーバのプロセッサはどれか 1 つのクライアントのプロセッサと同じあっても良い(ある 1 つのプロセッサはサーバでありクライアントでもある)。

(a) N 個の親仮説をクライアント台数 P で割り、各クライアントに分配する。

(b) 各クライアントでは、サーバからの要求に基づき、分配された親仮説群を基に逐次版と全く同様に子仮説を信頼度の高い順に幾つか生成し、それらをサーバへ送信する。

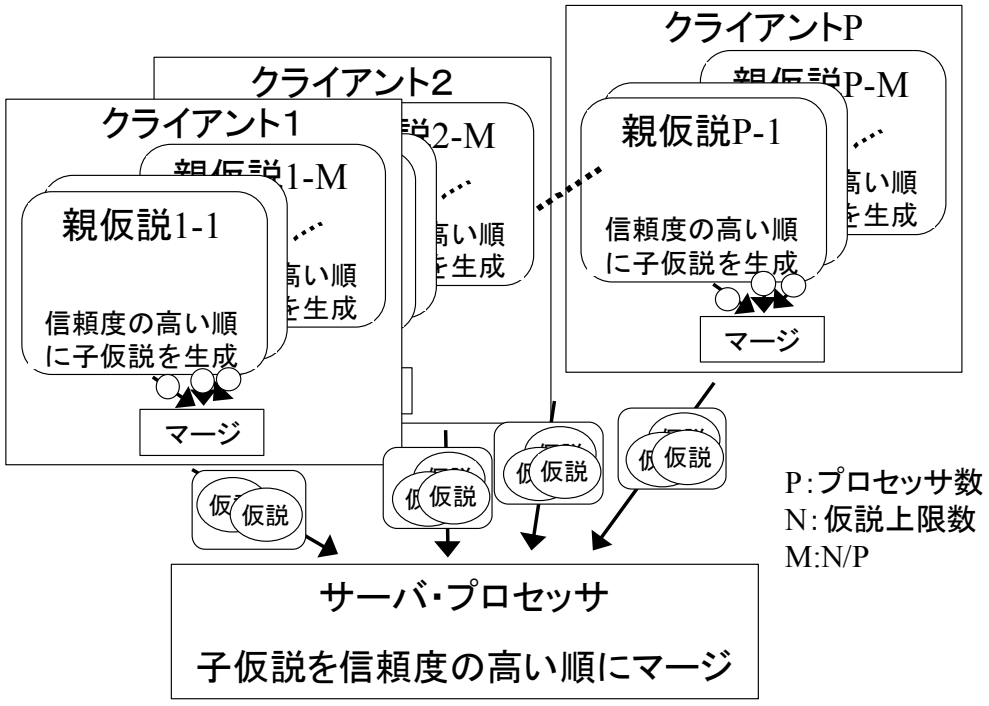


図 8 仮説生成の並列化

(c) サーバは、各クライアントから送られてきた子仮説群（各クライアント内では信頼度順にソートされている）を信頼度順にマージ・ソートする。このマージの際に、あるクライアントからの子仮説群を使い果たしたら、次の子仮説の生成をそのクライアントに要求する。

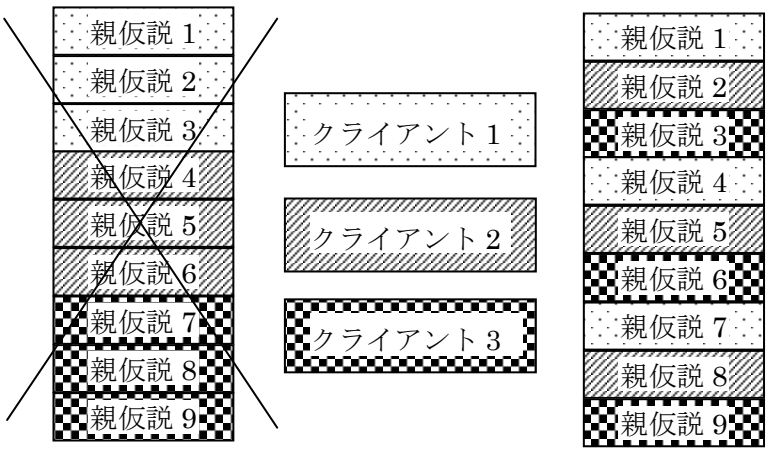
(d) (c)のサーバでのマージ処理を子仮説数が N 個になるまで繰り返す。

この並列処理方式では、親仮説から信頼度の高い順にある個数の子仮説を生成するという単位で並列化しているので、子仮説の生成方法（Munkres アルゴリズムや Murty アルゴリズム）には全く依存していない。従って、その生成方法を別のアルゴリズムに変更したとしても、本並列処理方式はそのまま適用できる。

4.3 並列化方式の特長

(1) ラウンドロビンによる静的負荷分散

親仮説の各クライアントへの分配は、静的に割り付ける（処理を最初に割り付け、負荷状況によって移動させない）。前述したように、子仮説の信頼度は、親仮説の信頼度と航跡と観測ベクトルの相関度等から算出される。そのため、親仮説の信頼度がそこから生成される子仮説の信頼度に影響する。従って、信頼度の高い順に並んでいる親仮説を N をクライアント数 P で割ったブロック毎に各クライアントに割り付けてしまうと、上位のクライアントに高信頼度の親仮説が集中し、それらのクライアントから生成される子仮説のみが採用される可能性が高くなり、クライアント間で負荷のアンバランスが起き易くなる。そこで、ブロックではなく、高信頼度の親仮説から 1 つずつ順に各クライ



(1)ブロック割付　　(2)ラウンドロビン割付

図 9 親仮説の分配方法

アントへ割り付けるようにする（図 9)。これにより、各クライアントの親仮説の信頼度が均衡化され、それに従って各クライアントの負荷も均等になる。

(2) 複数仮説単位での通信

各クライアントは子仮説を一つ生成したら直ぐにサーバへ送るのではなく、複数個生成してから送信する。これは、子仮説を一つ生成する処理が粒度として小さすぎる可能性があり、一度に生成する子仮説の数でその粒度を調整できるようにするためである。現在、一度に生成する子仮説の数は、N/P/10 としている（N：親仮説数、P：クライアント数)。すなわち、各クライアント毎に平均 10 回程度サーバへ子仮説群を送信することになる。この数は、本来、並列処理環境のプロセッサ速度と通信速度の比率により調整すべきものである。

(3) 仮説生成のダブル・バッファリング

各クライアントは、生成した子仮説群をサーバへ送信した後は、次の生成要求がサーバから来るまで暇となり、クライアント・プロセッサが遊休状態になる可能性がある。そこで、各クライアントは、次にサーバから子仮説生成の要求が来る前に、次の子仮説群を前もって生成して溜めておく（バッファリングする）ようにしている。これにより、各クライアントの遊休状態が緩和されるだけでなく、早めに生成を開始しているので、サーバからの生成要求に対してのレスポンスも早くなる。

5. 評価実験

今回の並列化による効果を評価するために、シミュレーションによる計測実験を行った。誤信号密度及び新目標密度は同じ値を取り、それぞれ低、中、高の 3 種類のデータを用いた。サンプリング間隔は 4 秒で、シミュレーション時間は初探知の時刻 0 秒から 120 秒までであり、測定は乱数のシードを変えて 30 回行い、その平均を取った。また、仮説の生成上限数（=N）は 3,000 個である。実験に用いた並列処理環境を表 1 に示す。Alpha21264 プロセッサを 2 つ持つマシンを Myrinet で 3 台接続した環境である（CPU 総数は 6)。本計測でクライアント台数を増やす場合は、各マシンに 1 つずつ割り当てていき、次に 2 つ目の CPU に割り当てるようにしている。

表 1 並列処理環境の主な仕様

CPU/クロック	（Alpha21264/667MHz×2）×3
メモリ	512MB
キャッシュ	1次:8KB, 2次:96KB, 3次(外部):2MB
OS	SuSE Linux 6.3
ネットワーク	Myrinet(MPICH-1.2.0, GM-1.1.3.14)

5.1 実行時間

表 2 に、各クライアント数毎の 1 サンプリング時刻当たりの全実行時間及び仮説生成時間の平均を示す。クライアント数が 0 とは、従来の逐次の航跡型 MHT プログラムでの実行時間であり、クライアント数が 1 とは、仮説の生成のみを別プロセッサで実行した場合の実行時間である。また、図 10 に、各クライアント数毎の高速化率（全実行時間の逐次プログラムとの比率）及び並列化効果（仮説生成時間の 1 クライアントとの比率）を示す。これらの結果から、以下のことが言える。

- クライアント数が 1 台の場合は、仮説の生成が別プロセッサで実行されるだけで、並列実行は全く行われないし、仮説生成に必要な入力データと生成された仮説データを通信しているにもかかわらず、逐次プログラム（クライアント台数 0 台）より速くなっている。同様に、高速化率及び並列化効果ともに、クライアント数以上の効果が出ている場合がある。
- サーバと一つのクライアントが同一 CPU で動作するクライアント数 6 の場合は、かえって遅くなっている。別途 MPI を使用しない版で計測した場合は、遅くなっていなかったので、MPI の共有メモリ機能が関係しているのではないかと考えられる。
- 誤信号及び新目標密度が高くなると組合せ数が多くなるので、全体及び仮説生成ともに実行時間は長くなるが、並列化効果は密度によってそれほど変わらない。高速化率は、密度が低い方が効果が出ている。これは、密度が高くなると仮説生成以外の処理割合が増えているからだと考えられる。
- 並列化効果は、ほぼ直線になっており、更にクライアント（プロセッサ）数を増やしても効果は持続すると考えられる。ただし、高速化率の方は密度が中以上で頭打ちの傾向が見られる。これは、今回並列化しなかった部分（恐らくクラスタ統合）が次にボトルネックになっているからだと思われる。

前記のクライアント数以上の効果が出ている理由としては、プロセッサ数を増やすことによりキャッシュメモリ量も増加し、キャッシュヒット率が向上するためであると考えられる。すなわち、クライアント数が増えるに従って個々のクライアントで扱う仮説数が少なくなるが、クライアント当たりのキャッシュメモリ量は変わらないので、キャッシュヒット率が向上するのである。試しに、キャッシュがヒットし難いように、データを飛び飛びに格納するように変更したら、実行時間がかなり長くなった。

表 2 全実行時間と仮説生成時間

全実行時間（単位：秒）

クライアント数	0	1	2	3	4	5	6
誤信号密度:低	10.39	6.57	2.31	1.50	1.15	0.95	3.31
誤信号密度:中	12.52	10.49	3.89	2.58	2.20	1.78	4.46
誤信号密度:高	17.05	16.75	7.67	5.27	4.80	4.16	7.34

仮説生成時間（単位：秒）

クライアント数	0	1	2	3	4	5	6
誤信号密度:低	9.98	6.20	1.94	1.13	0.78	0.58	2.76
誤信号密度:中	11.61	9.53	2.92	1.62	1.23	0.81	3.00
誤信号密度:高	15.21	14.69	5.60	3.19	2.73	2.07	4.21

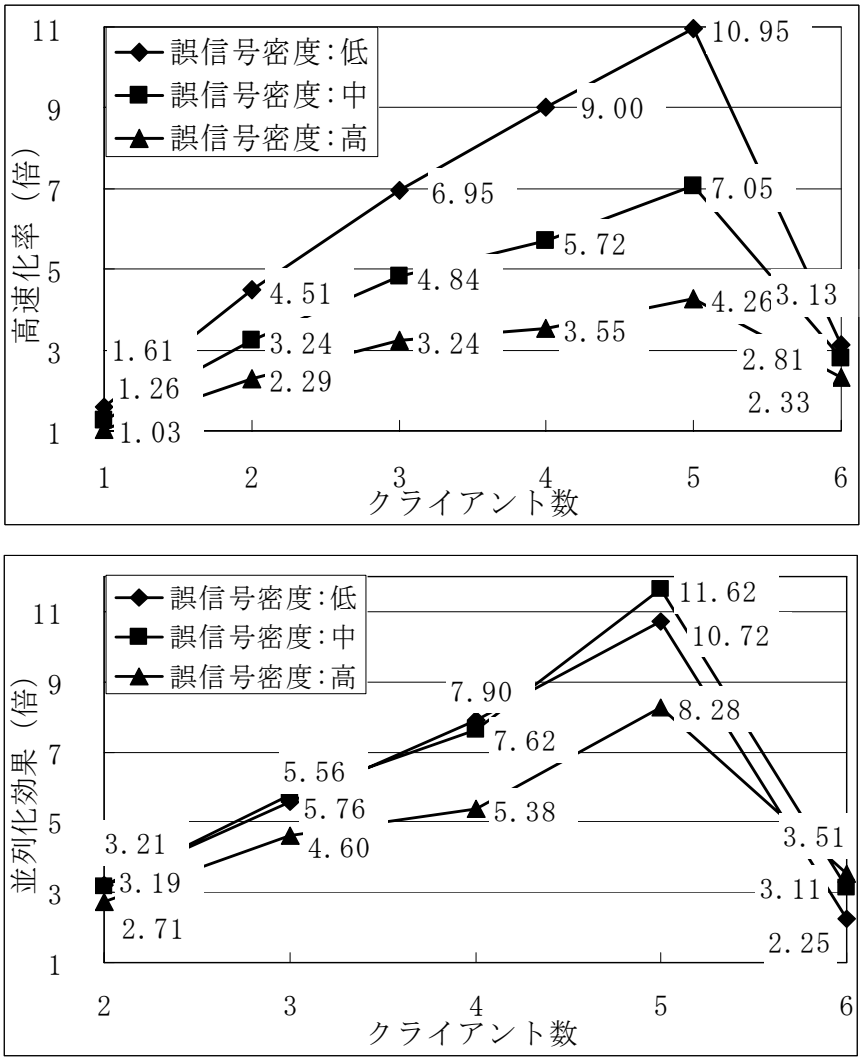


図 10 全実行時間の高速化率と仮説生成部分の並列化効果

5.2 通信量と負荷の均等具合

次に、並列化効果を妨げる要因である通信オーバヘッドと各プロセッサでの負荷の不均衡を調べるために、サーバと各クライアント間での通信量と各クライアントで生成した子仮説の数を計測した。なお、計測に用いたデータは、前節の誤信号密度:中であり、クライアント数は 5 台である。

(1) 入力データ（サーバ→各クライアント）量

仮説生成を始める前にサーバから各クライアントに送られる仮説生成に必要なデータは、どの仮説がどの航跡を保持しているかという情報と各航跡と観測ベクトルの相関度情報である。前者はバイトデータ（本来はビットデータでも可）の 2 次元配列であり、後者はダブルワードの 2 次元配列である。それぞれの要素数は、仮説数、航跡数、観測ベクトル数で決まる。

この入力データの 1 クライアントの 1 並列実行当たりの平均データ量は、89K バイトであった。通信には Myrinet の MPI を使用しており、実効性能は別途計測により 134MB/s（レイテンシーは 40μ秒）なので、この入力データの通信時間は 0.7 ミリ秒程度である。5 台実行で仮説生成時間が 810 ミリ秒なので、この通信時間を 5 倍したとしても、通信は全くオーバヘッドにはなっていないと言える。

(2) 出力データ（各クライアント→サーバ）量

出力データは生成した子仮説の情報そのものであり、基となった親仮説の識別子と航跡と観測ベクトルの組である。

この出力データの 1 クライアントの 1 並列実行当たりの平均データ量は、16K バイトであった。これは、入力データよりかなり小さいので、全くオーバヘッドはないと考えて良い。

(3) 子仮説生成数

各クライアントで生成する子仮説の数を計測し、それが各クライアントでどの程度異っているかを調べた。その結果、各クライアントで生成する子仮説の数は、1 クライアントの 1 並列実行当たりの平均で 657 個であった。これは、仮説上限数 3,000 個をクライアント数 5 で割った 600 に、先行して生成した数 60 を加えた数 660 に近い数字である。また、各クライアントの生成数の標準偏差（ばらつき）の平均は、55 個であった。これは、657 個に対して 12%であり、静的負荷分散にもかかわらず、それほど負荷が不均衡になっていないと言える。

次に、ラウンドロビンによる分配の効果を知るために、ブロック割付した場合も計測した。その結果、各クライアントで生成する子仮説の数の平均は変わらないものの、標準偏

差は 963 個と圧倒的に大きくなっていた。また、全実行時間は 1.78→2.16 秒、仮説生成時間は 0.81→1.20 秒に増えており、ラウンドロビン分配による効果を確認することができた。

6. おわりに

以上、信頼度の高いものから順に固定数の仮説しか生成しない N ベスト版航跡型 MHT の並列化について報告した。並列化に際しては、最も負荷の高かった仮説生成部分を並列化し、各プロセッサで負荷が均等化されるように処理を分配し、並列処理粒度を調整することによりプロセッサ間の通信オーバヘッドが出ないようにし、各プロセッサでは先行して仮説生成を行うことにより遊休状態にならないようにしている。その結果、キャッシュメモリの増量効果もあり、6 プロセッサを使用して、全実行時間で 4 倍～11 倍、並列化した仮説生成部分に関しては 8 倍～11 倍の高速化を確認した。

このようにセンサ・データ処理においても、従来の情報処理分野の技術が活かせるようになってきている。今後は、更にデータの抽象度を上げ、センサ配置の最適化やセンサ･データに基づく意思決定支援等の AI 的な要素を含んだ処理も必要とされると考えられる。

参考文献

[1] 小菅義夫, 辻道信吾, 立花康夫, ”航跡型多重仮説相関方式を用いた多目標追尾,” 信学論(B-II), vol.J79-B-II, no.10, pp.677-685, Oct. 1996.

[2] 小幡康, 系正義, 辻道信吾, 小菅義夫, “N ベスト解探索アルゴリズムによる航跡型 MHT の高速化(1)－N ベスト解探索アルゴリズムの導入方式－,” 2001 信学総大, B-2-32, Mar. 2001.

[3] 系正義, 小幡康, 辻道信吾, 小菅義夫, “N ベスト解探索アルゴリズムによる航跡型 MHT の高速化(2)－演算時間および追尾維持性能の評価－,” 2001 信学総大, B-2-33, Mar. 2001.

[4] 系正義, 辻道信吾, 小菅義夫, “航跡型 MHT による追尾維持性能のシミュレーション評価,” 信学技報 SANE99-117, SAT99-154, P.71-78, Fep. 2000.

[5] F.Bourgeois, J.C Lassale:”An Extension of the Munkres Algorithm for the Assignment Problem to Rectangular Matrices,” Comm. Of the ACM, vol.14, no.12, Dec. 1971.

[6] K.T.Murty, “An Algorithm for Ranking al the Assignments in Order of Increasing Cost,” Oper. Res., 16:682-687, 1968.

3.4.2 グリッド技術の概要と動向

関口　智嗣　委員

1. はじめに

平成15年3月4日から7日にかけて第7回グローバルグリッドフォーラム(GGF7)が東京・新宿京王プラザホテルにて開催された。総勢約800人、海外から500名もの参加を見た当会議はアジア・太平洋地区では初めての開催となった。

これらをきっかけに新聞、雑誌、Webニュースなどのメディアを通じても、グリッド・コンピューティングという言葉を良く見かけるようになってきた。大学・公的研究機関などの大規模な計算機システムが対象となるばかりではなく、一般家庭のパソコンを対象としたグリッド技術が構築されてきている。元来は、科学技術計算の分野で始まったグリッドの波であるが、徐々にビジネスの世界に押し寄せてきている。すなわち、科学技術における応用としてはバイオインフォマティクス、創薬デザイン、ライフサイエンス、高エネルギー物理や天文観測などの大規模科学実験などが知られていたが、今後はWebサービス、ユーティリティ提供、アミューズメントなどが次のターゲットと考えられている。こうした大きな構想とは別に、現実の問題としてグリッドをどうすれば毎日の研究生活に使えるようになるのか、そうした日々のグリッドを作るためにはなにが必要なのかを念頭に置きながら、技術の現状を整理し、今後のグリッド技術動向全般について概観する。

グリッドとは、「遊休PCをネットワークで繋いだもの」でもなければ、「スーパーコンピュータをネットワークで繋いだもの」でもない。新たなネットワークを利用した技術として、安全に・安定して・安易に様々な情報サービスを享受できるようにするための技術である。「安全に」とはセキュリティ面での配慮があること、「安定して」は必要な計算資源、記憶資源、ネットワーク資源が必要なだけ提供されるように計算資源が仮想化されていること、「安易に」とはユーザは特に端末の裏側で何が起きているか気にする必要なく情報サービスへのアクセスを可能とすることを目指している。

グリッドを大きく分類すると図1のように考えられる。ここでは、PCを中心としたいわゆる分散計算型のグリッドと、狭義のグリッドとして、柔軟な情報サービスを目指したものがある。こちらは、ビジネスグリッド、コンピューティンググリッド、データグリッドなどの主な目的に応じた分類が可能である。しかし、これらの境界は曖昧であり、明確な分類ではない。通常は複数の目的を同時にひとつのシステムとして実現される。

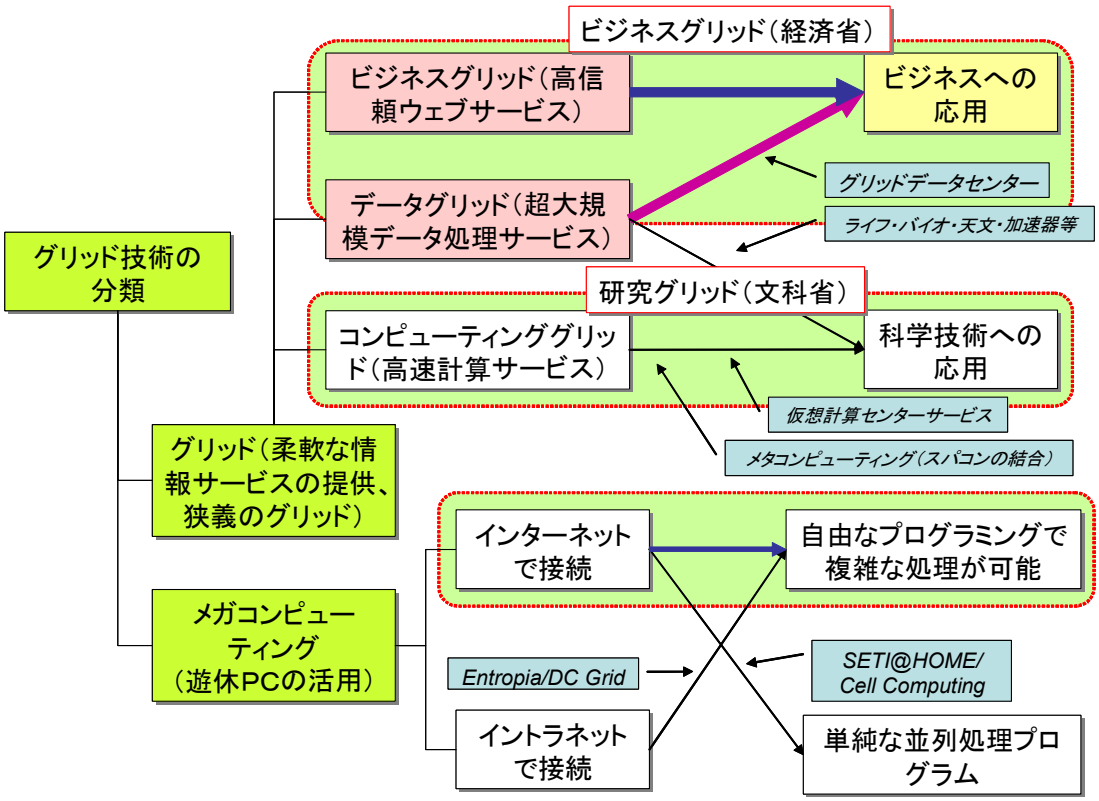


図1 グリッド技術の分類

グリッド・コンピューティングの考え方そのものは、十年以上前から芽生えていたが、実用的な例題に活用され始めたのは科学技術分野においてさえ、ここ数年のことである。大学、研究所などのスーパーコンピュータを高速ネットワークで接続し、大規模なグリッド環境を構築する試みが世界各地で進められている。米 NSF の基金による TeraGrid は、カリフォルニア州にある SDSC、イリノイ州にある NCSA など四つのデータセンターを、高速伝送網で接続するプロジェクトである（図 2 参照）。光ファイバー通信で異なる波長の光信号を複数のチャネルで伝送する WDM (wavelength division multiplexing)通信方式を用い、将来は 50-80Gbps（1Gbps は 1 秒間に 10^9 ビットを転送）を目指す。各センターには Itanium2 による大規模な Linux クラスタと RAID 装置を配備し、トータルで 13.6TFlops（1TFlops は 1 秒間に 10^{12} 回の浮動小数点演算を実行する性能）の演算性能と 450TB（1TB は 10^{12} バイトの容量）のストレージを有するグリッド環境を構築する。

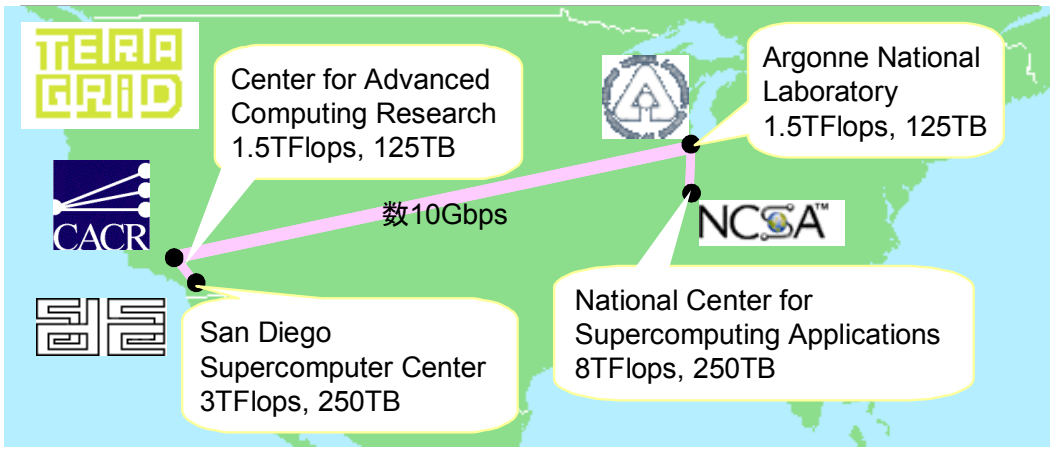


図 2　米国 TeraGrid プロジェクトとして計画中のグリッド環境

2. 海外における動き

グリッド・コンピューティングの世界は、2002 年になってから急速な展開を示している。2002 年 2 月にカナダのトロントで行なわれたグリッド技術の標準化を進める国際会議 Global Grid Forum(GGF)4 において、米 IBM 社がグリッド基盤の新しいアーキテクチャとして、Web サービス技術を取り込んだ OGSA を発表したことが要因の一つである。

GGF は、1999 年に米国を中心に始まった Grid Forum にヨーロッパ、アジア諸国が加わり 2000 年 11 月組織された世界レベルの標準化団体である。GGF には表 1 に示すように 7 つのエリアがあり、エリア毎に設置されたワーキンググループ（WG）とリサーチグループ（RG）が主な活動である。WG は仕様書、ガイドライン、推奨事項といったドキュメントの提供を目指し、非常に特定の技術に焦点を当てている。RG はドキュメントを作成するには時期尚早なテーマについて、長期的に議論を進めている。この会議は年に 3 回の頻度で会合が行われており、2002 年 7 月に第 5 回目の会議 GGF5 がスコットランドのエジンバラで開催された。図 3 に示すように、GGF への参加者数は GGF4 から急速に増加し、グリッドへの関心の高さを示している。OGSA の提案が、今回企業から多くの参加者を得て急増した理由の一つと考えられる。

表 1　GGF におけるエリア、WG、および RG（出展 GGF）

AREA	Working Groups (23)	Research Groups (20)
Applications, Programming Models, Environments	• Grid Checkpoint/Recovery	• Advanced Collaborative Environments (ACE) • Advanced Programming Models (APM) • Applications and Test Beds (APPS) • Grid Computing Environments (GCE) • Grid User Services (GUS) • Life Sciences • Production Grid Management
Architectures	• Open Grid Service Infrastructure (OGSI) • Open Source Software (OSS) • New Productivity Initiative (NPI) • Open Grid Services Architecture (OGSA)	• Accounting Models (ACCT) • Grid Protocol Architecture (GPA) • Service Management Frameworks (JINI) • Semantic Grid (SEM-RG)
Data	• GridFTP • Data Access and Integration Services (DAIS) • *OGSI Data Replication Services -WG*	• Data Replication (REPL) • Persistent Archives (PA) • Grid High-Performance Networking • Data Transport
Information Systems and Performance	• Discovery and Monitoring Event Description (DAMED) • Network Measurement (NM) • Grid Information Retrieval (GIR) • CIM based Grid Schema (CGS)	• Relational Grid Information Services (RGIS) • Grid Benchmarking (GB)
Peer-to-Peer		• Appliance Aggregation • Relation of OGSA/Globus and Peer to Peer
Scheduling and Resource Management	• Scheduling Attributes (SA) • Scheduling Dictionary (SD) • Distributed Resource Management Application API (DRMAA) • Grid Resource Allocation Agreement Protocol (GRAAP) • OGSA Resource Usage Service • Grid Economic Services Architecture • Usage Record	
Security	• Grid Security Infrastructure (GSI) • Grid Certificate Policy (GCP) • Open Grid Service Architecture(OGSA) Security (OGSA-SEC) • CAOps • Authorization Frameworks and Mechanisms	• Site Authentication, Authorization, and Accounting Requirements

OGSA はグリッドと Web サービスの良い点を統合したコンセプトであり、J2EE や.NET などを用いるビジネス分野との親和性が高い。SOAP、WSDL、UDDI などの Web サービス技術を用いて、グリッド基盤を構築することにより、ビジネスでの情報システム利用が、科学技術計算と同じ基盤上で実現されるようになる（図 4 参照）。その実現のためには様々

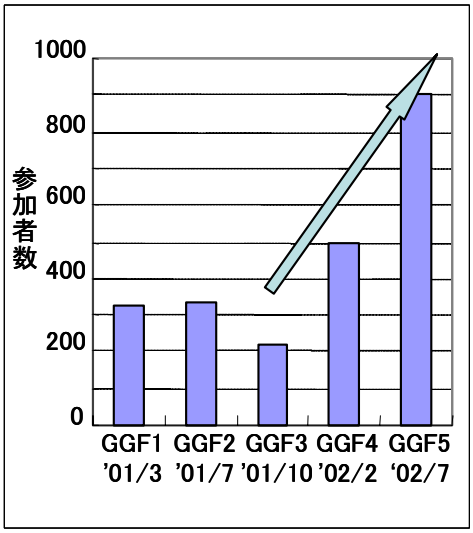


図 3　GGF への参加人数

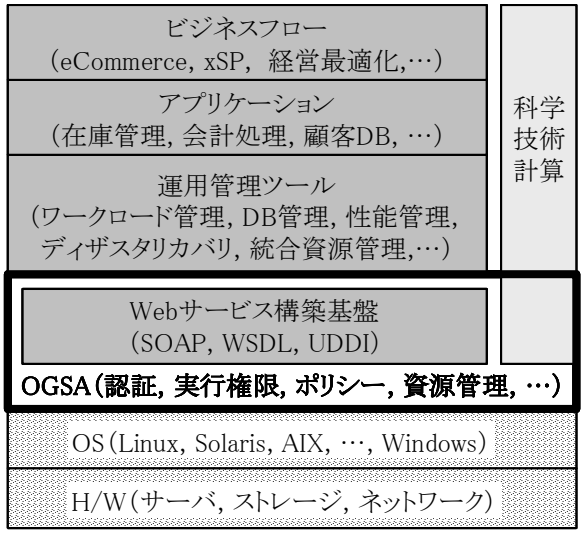


図 4　OGSA の位置づけ

な仕様拡張が必要である。例えば、計算機資源の仮想化を行うには、実行されるサービス（アプリケーション）が特定の計算機資源に留まることを前提にはできない。すなわち、サービスの一時的な生存を可能にしなければならず、サービスを記述する WSDL にサービスの生成・消滅・生存時間などの仕様追加が必要となる。このようなグリッド向けの拡張仕様を GSDL して GGF の OGSI-WG などで検討されている。

OGSA のミドルウェアへの実装については、主として Globus プロジェクトで進められている。Globus プロジェクトは、米国アルゴンヌ国立研究所と南カリフォルニア大学を中心に、米国の大学、IBM 社、マイクロソフト社といった企業も参加する最も大規模なミドルウェア開発チームである。GGF と並行してグリッドミドルウェア Globus Toolkit を開発し、ある種のデファクトスタンダードとなっている。

Globus が提供するのは、資源管理、資源情報サービス、データ管理の機能といった、グリッドにおけるプリミティブな機能である。これらの要素サービスを束ねて、ワークロード管理、データベース管理、ディザスタリカバリー、性能管理、統合資源管理などを提供するのが、上位の管理ツールである。しかし、管理ツールレイヤーにおける標準化についてはまだ手がつけられていない。

3. ビジネスにおけるグリッド技術の応用

OGSA がビジネス分野でも使われる基盤であることは既に述べた。では、具体的にどのように利用されるのか。現在のグリッド技術、および Web サービス技術と融合した将来のグリッド技術を用いることで、どのようなビジネス形態が現れるかを述べる。

企業情報システムの仮想化が現時点で容易に実現可能な利用方法である。企業における計算機資源は、一般に平日の夜間や休日などを合わせると 50%以上は休止状態であり、ロードバランスも不均一になり易い。グリッド技術を用いて計算機資源を仮想化・統合化することにより、業務アプリケーションの実行効率をあげることができる。ユーザは使用するマシンを意識する必要はなく、投入されたアプリケーションジョブは資源の空き状況に応じてスケジューリングされ、効率よく順次実行される。このようなグリッド環境を構築するためのミドルウェアは、研究ベースとしては Globus プロジェクトによる Globus Toolkit、UNICORE プロジェクトによる UNICORE などがある。またプラットフォームコンピューティング社、アバキ社、エントロピア社などからもツールやソリューションが提供されており、車体設計を行う自動車メーカーやリスク管理評価を行う金融機関など多くの海外企業で利用が始まっている。

一方で、Web サービス技術の登場によりアプリケーションを連携・統合することが可能となった。共通的な要素機能としての顧客情報管理、取引先情報管理、在庫管理、従業員

情報管理、資産管理などを SOAP、WSDL を用いてサービス化し、SCM、CRM、ERP、給与計算などの業務アプリケーションから呼び出すように再構築できる（図 5 参照）。要素機能を再利用することで、業務システム開発の工数が削減可能である。将来 OGSA として、Web サービス技術がグリッドの基盤として確立した後には、各業務アプリケーションを実行するサーバを固定する必要がなくなり、負荷状況に応じて必要なサービスを空いているサーバ上に起動、相互に連携実行させることが可能となる。IT への投資削減を進める企業にとって、グリッド技術は資源を有効活用する重要なツールとなるだろう。

また、グリッド技術を用いて統合・仮想化された計算機資源に対して、必要な時に必要なアプリケーションを起動して利用することは、電気、ガス、水道と同様にユーティリティの一つとして見なすことができる。米 IBM 社は、「e-business on demand™」のコンセプトを掲げユーティリティ事業に乗り出している。このサービスの基盤として 7 月に発表した「Linux Virtual Services」では、同社 zSeries で構成した Linux 搭載メインフレームを仮想サーバとし、各顧客の要求に応じて資源を割り当てる。ユーザは必要な分だけの処理能力を買うことができる。まさしくユーティリティ事業である。

ヒューレットーパッカード社は、コンパック社との合併で強化されたグリッド技術により、ユーティリティデータセンター（UDC）のソリューションを提示している。UC（Utility Controller）ポータルを用いて、ビジネスに合わせたサービス設計とシステム構築のテンプレート記述を行うと、UC ソフトウェアが UDC 内にプールしてあるクラスタ資源を適切にネットワーク接続し、必要なアプリケーションをインストールする。ビジネスの立ち上げがフレキシブルに且つ短期間に実施可能である。

分散リソース管理ツール「Sun Grid Engine」と Web サービスのフレームワーク「Sun Open Net Environment（Sun ONE）」の統合を行ってきたサン・マイクロシステムズも、ユーティリティコンピューティングを実現する次世代データセンター構想をこの 9 月に発

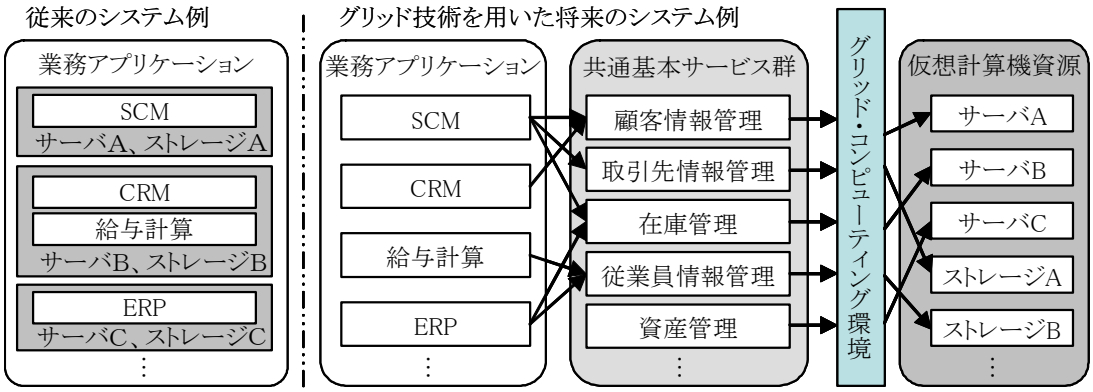


図 5 Web サービスによる共通機能の再利用とグリッドによる計算機資源の仮想化により、資源の動的な負荷分散が可能となる

表した。ネットワークコンピューティングのアーキテクチャ「N1」を発展させ、自ら新しい事業を起こす決意である。

4. ビジネス応用への課題

ユーティリティ事業にまで至るグリッド技術のビジネス応用実現にあたっては、まだまだ多くの課題が残されている。特に、ミッションクリティカルな分野への応用となるため、システムの安定稼動、セキュリティ、相互運用性といった課題が重要である。また、ユーティリティとして普及・浸透するためには、技術的課題の他にも、十分な数の利用者が参入すると、サービス提供者の格付け、利用者の与信、利用しやすいビジネスモデルとそれに適した課金システムなど、解決すべき社会的課題も多い。

システムの安定稼動に向けては、具体的な活動が始まっている。要求されたアプリケーションの実行時に、プロセッサやメモリなどのハードウェア資源を必要な量確保できなければ、処理性能の低下が生じる可能性がある。また、ハードウェアの一部に障害が発生した場合、人手による修復を待っていたのではビジネスチャンスを失う可能性がある。IBM 社は早い段階からサーバの自律機能を強化するプロジェクト eLiza を進めている。処理能力が低下したら、アプリケーションを停止することなく自動的にプロセッサやメモリを追加する。障害が生じたら、システムが自動的に障害を検知して修復を行う。自律機能の一つである動的資源配置を、NEC は NX7000/superdome の同一筐体内を対象に、ミドルウェア「HA/GlobalMaster」として製品化した。ヒューレットーパッカード、サン・マイクロシステムズ、富士通なども同様の機能実現を計画している。これらシステムの自律機能は、ビジネスへのグリッド技術利用において重要な要素である。

ハード資源の統合・仮想化を行うことは、異なるユーザが資源を共有する可能性があることを示している。ユーザの認証、資源の制御、情報の改ざん／漏洩防止などセキュリティ上の課題解決はビジネス上特に重要である。

セキュリティについては、これまで既存の技術の流用を基本としてきた。例えば Globus では、GSI と呼ばれる SSL をベースとした認証機構で実現している。Globus 独自の第三者認証機関 CA を運営しており、全てのユーザに SSL で用いられる X.509 の証明書を発行している。また、グローバル ID とローカル ID との対応表による利用可能な資源の制限といった方法が用いられている。

GGF では、インターフェースやプロトコルについてガイドラインを設けるだけで、具体的なツールや技術については規定していない。現在は、どうやって CA を信用するか、異なるユーザ ID で複数の資源にアクセスする必要がある場合の管理方法など、について議論している。

ビジネスの世界では、様々な組織階層が存在し、それらの間で開示可能な情報は様々に異なる。従って、グループ内、企業内、提携企業間、無制限公開など、資源を共有する範囲によってセキュリティのレベルを変える必要がある。

また、初めての利用者や初めてのアプリケーションを迎え入れる場合、どうやって相手を信用するか、逆にサービス提供者の信用を利用者に与えることも重要な問題である。利用者の与信、提供者の格付け、アプリケーションが当該プラットフォームで稼動しウィルスなどを内包していないことの保証、などを行う第三者機関の存在が必要である。

オープンなシステムの世界では、相互運用性は欠かすことのできない課題である。オープンソースである Linux によるクラスタシステムは、コストパフォーマンスの面だけでなく、プラットフォームの均一化が図りやすいという面からも、グリッド環境構築に採用され易い。しかし、その他のプラットフォーム（ハードウェア、OS、ミドルウェア含む）を提供するハードおよびソフトベンダーも多く、ヘテロな環境に対して稼動することは必須の条件である。例え仕様が標準化されようとも、実装したツールが連携できるとは限らない。

ビジネスの世界でグリッド技術が花開くのは 5 年先とも言われているが、早期実現を目指して、これらの課題解決に向けた研究開発が世界各国で進められている。

5. 使えるグリッド

「使えるグリッド」を構築するためには、一般のユーザがどのようなことをグリッドに期待しているか十分な調査が必要である。グリッドに期待されるものとして、複数のサイトのコンピュータを同時に利用して、大規模な計算を実施するというものがある。しかしながら、その「使える度」は下記のようなものであった。これは米国 AVAKI社の CTOである Andrew Grimshaw氏がバイオインフォマティクス関連の市場調査を通じて整理したものであり、同様な傾向は筆者がCBI学会において講演中に問いかけた回答でも確認された。

- MPIでプログラムを書いている または興味がある - is very rare
- MPI で異機種計算機や遠隔地の計算機の利用をしている または興味がある - no interest at all
- 複数のアプリケーションを複数の拠点で同時に稼働させている または興味がある - basically no interest
- Remote visualization を行っているまたは興味がある - no interest
- 並列スケジューラを使っているまたは興味がある - little interest
- 新たにアプリケーションを作成したいと思っている - almost nil

確かに、研究の最先端としては複数サイトを利用して大規模計算を実現するというグランドチャレンジを一部の先進的なユーザは想定している。しかし、これはむしろ限定されたシナリオと考えられる。ユーザの多くはグリッドによりいかに日々のコスト(TCO)削減に結びつくかが導入あるいは使用に踏み込むきっかけと考えられる。このような、日々使えるグリッドを作って行く必要がある。逆説的であるが、グリッドを使っていることをユーザがいかに意識しないで済ませるかが必要となる。

我々は JSTの計算科学活用型プロジェクトの課題としてGridLib という仮想スーパーコンピュータセンターの実現を目指している。これは日々のグリッドの例としてユーザが計算サービスを提供する機器に関してOS、機種、ライブラリ、関数、ライブラリ毎のインターフェース、異なる運用ポリシー、スケジューリング、などの違いを隠蔽することが可能となる。具体的にはGSI (Grid Security Infrastructure)と Grid RPC 技術を用いることで実現を目指している。図6にグリッドASPのイメージを示す。現在は、ASPの例として量子化学計算用ソフトウェア GAUSSIAN, 熱流体コード PHENICS、構造解析コード NASTRAN 等を対象としている。GAUSSIAN は Quantum Chemistry Grid の一環としてすでに産総研・先端情報計算センターにおいてユーザに公開を始めたところである。

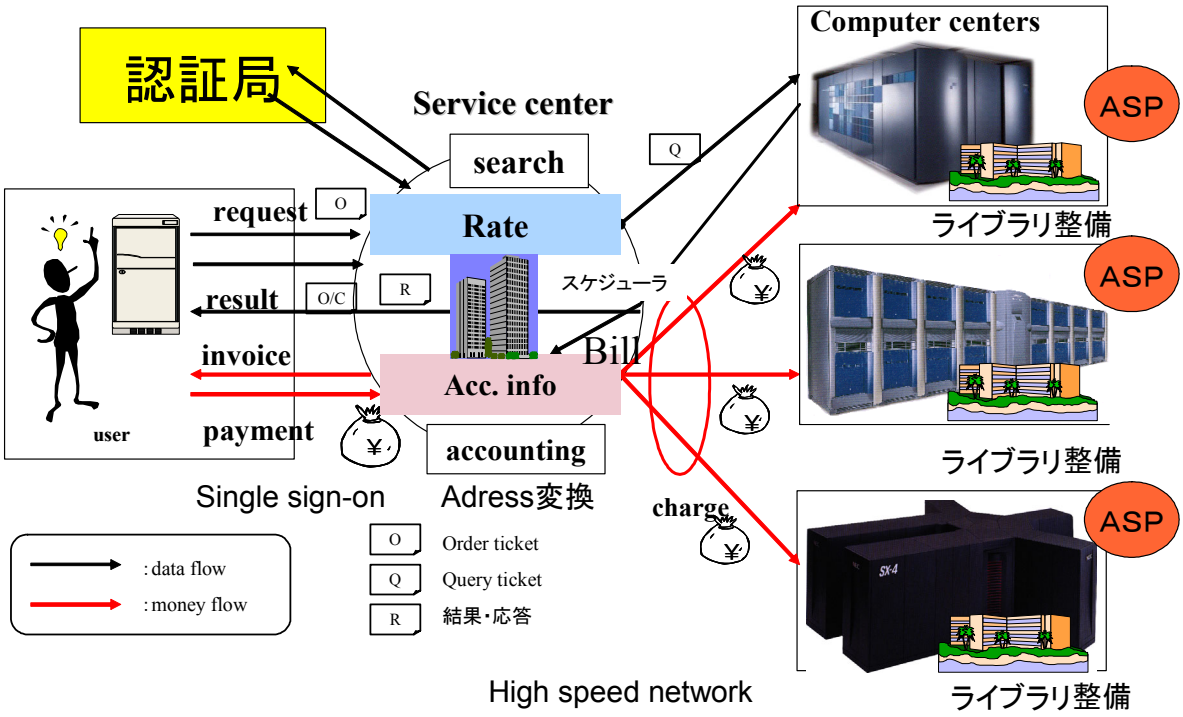


図6 グリッドASP

6. おわりに

グリッド技術はインターネットの急速な普及をもたらしたアクセス技術の整備とバックボーン技術の高速化、高信頼化が前提となっている。一方で、コンピュータのサービス提供技術はメインフレーム集中型モデルからクライアント・サーバ型モデルを経て、オープン型へとネットワーク技術の変遷とともに推移してきた。グリッド技術の根幹は仮想的な組織における資源共有の実現であり、これを活用した新たなサービスモデルにより現状のサービスが「グリッド化」されてくる。例えば、将来Webサービス技術がグリッドの基盤として確立した後には、各業務アプリケーションを実行するサーバを固定する必要がなくなるため、負荷状況に応じて必要なサービスを空いているサーバ上に起動、相互に連携実行させることが可能となる。ITへの投資削減を進める企業にとって、グリッド技術は資源を有効活用する重要なツールとなることが期待される。今後、グリッド技術は情報基盤技術として当然のように産業基盤となる科学技術計算の分野ばかりではなく、ITビジネスにも広がってくる。応用範囲が拡大するにつれ、多くのベンダーが事業に参入してくるであろう。競争と協調、がグリッド技術の今後の課題である。

3.4.3 非圧縮性一様等方性乱流の大規模直接数値シミュレーション

横川　三津夫　委員

1. はじめに

乱流の微細渦構造を直接数値シミュレーション（DNS）によって捉えようとする研究は、スーパーコンピュータの出現により飛躍的に発展した[1]。近年では、さらに計算性能を増大させるために複数の計算プロセッサを協調動作させる並列計算機が開発され、乱流研究ばかりでなく多くの計算科学の分野においてその高い計算能力を発揮している。特に、並列ベクトル型スーパーコンピュータ上でフーリエ・スペクトル法による 10^9 自由度を持つ非圧縮性一様等方性乱流シミュレーションが実現したが、これは計算機ハードウェア及び並列プログラミングなどソフトウェア技術の進展に依るところが大きい。しかし、21 世紀を迎えた時点では、計算性能及びメモリ容量の制約から計算格子点数 1024^3 の DNS が限界であった。

超高速並列計算機システム「地球シミュレータ」は、数値シミュレーションにより地球変動現象の解明を目指すこと、すなわち仮想地球を計算機上に実現することを目的として、1997 年から開発してきた SMP（Symmetric Multi Processors）クラスタ型の大規模並列計算機システムである[2]。2002 年 2 月に完成し稼動し始めた。計算機システムの性能に対する一つの指標であるフロップス値（1 秒間に可能な浮動小数点演算数）で示せば、地球シミュレータの理論ピーク性能は 40 テラフロップスであり、実効性能では世界的標準ベンチマークテストプログラム LINPACK において 35.86 テラフロップス（ピーク性能比 87.5%）を達成した。現時点において世界最高速のスーパーコンピュータである。地球シミュレータの高い計算性能と大きなメモリ空間を用いれば、10^{11} 自由度乱流 DNS が実現可能であり、乱流研究が大きく進展するに違いない。

2. 地球シミュレータ

地球シミュレータは、640 台の独立動作可能な計算機（ノードと呼ぶ）が単段クロスバスイッチによって結合されている（図 1）。各ノードは、ピーク性能 8 ギガフロップスのベクトル型計算プロセッサ 8 個が 16 ギガバイトのメモリシステムを共有する SMP であり、ノード間データ転送のための制御機構を持つ。計算プロセッサの総数は 5120 個、メモリ容量は 10 テラバイトである[3]。

大規模科学技術計算向きのスーパーコンピュータの要件として、ノード単体に対しては、浮動小数点演算性能、高速動作可能なメモリ素子、それらの性能に見合うプロセッサとメ

モリ間のデータ転送性能が挙げられる。また、複数ノードをネットワークで結合したクラスタにおいては、ノード間の高いデータ転送性能が重要である。地球シミュレータでは、ノード単体で 64 ギガフロップスの演算性能、同一バンクアクセスタイムが 24 ナノ秒の高速 DRAM 素子、8 個のベクトルプロセッサとメモリ間における 256 ギガバイト/秒のデータ転送性能を有しており、バランスよいシステムに成っている。また任意の 2 つのノード間最大転送性能は 12.3 ギガバイト/秒であり、ノード間ネットワーク全体では約 8 テラバイト/秒の類を見ない高いデータ転送性能を持っている[4]。

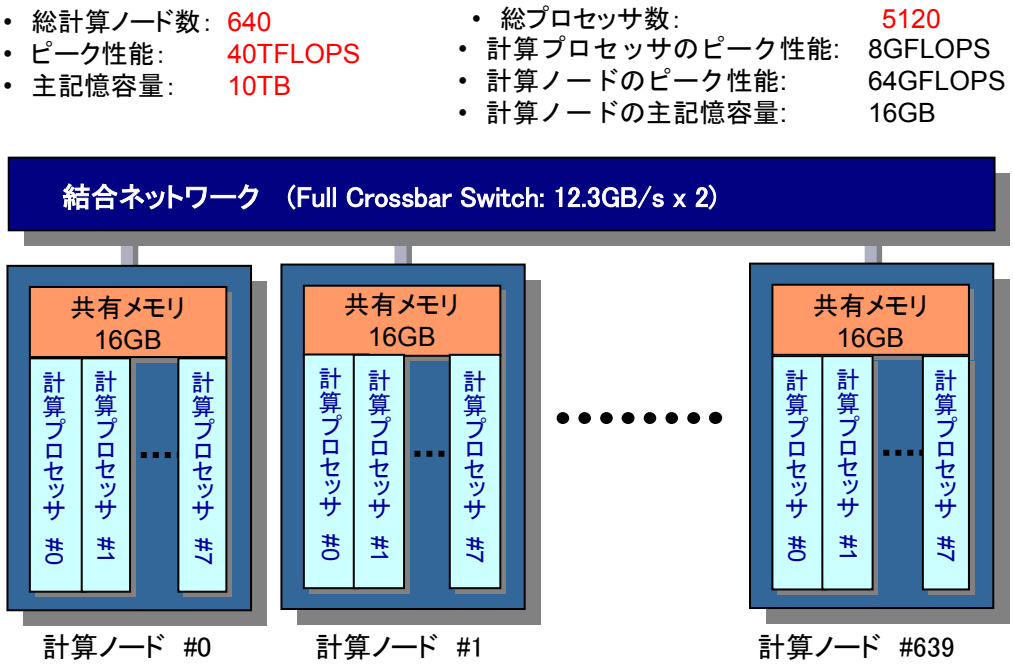


図 1 地球シミュレータのハードウェア構成

これらの性能を実現するために最先端の計算機製造技術が用いられた。ベクトル型計算プロセッサには、0.15 ミクロンの CMOS LSI 加工技術と銅配線技術が用いられ、従来複数の LSI で構成されていたベクトルプロセッサ部を約 2cm 四方の 1 チップ大型 LSI で実現した。また、配線幅及び配線間隔 25 ミクロンのビルドアップ基板加工技術や大型 LSI を搭載するためのベアチップ実装技術など、最先端のパッケージング技術を用いて高密度実装を実現するとともに、大型 LSI から発生する約 140W の発熱に対して沸騰型ヒートシンクにより冷却した。これらの技術により開発開始時点の同じ性能のスーパーコンピュータと比較して、体積で 1/50、消費電力で 1/10 を達成した。

3. 一様等方性乱流シミュレーション

地球シミュレータ上で大規模乱流シミュレーションを行うために、フーリエ・スペクトル法に基づく並列直接数値シミュレーション（DNS: Direct Numerical Simulation）コード Trans7 を開発し、非圧縮一様等方性乱流の世界最大規模直接数値シミュレーション（計算格子点数 4096^3）を実現した[5]、[6]。この成果により、2002 年米国ボルチモアで開催された国際会議 SC2002 において、ゴードンベル賞特別賞を受賞することができた。ここでは、コード概要、並列化手法、数値シミュレーション結果の一部について述べる。

3.1 基礎方程式と数値計算法

3 次元立方領域 $\Omega = [0,2\pi]\times[0,2\pi]\times[0,2\pi]$ を基本周期領域として、単位密度の非圧縮性流体の運動を考える。流体の運動は外力のある 3 次元ナビエ・ストークス方程式と連続の式で表される。

$$\frac{\partial u}{\partial t} + (u\cdot\nabla)u = -\nabla p + \nu\Delta u + f \qquad (1)$$

$$\nabla\cdot u = 0 \qquad (2)$$

ここで、$\boldsymbol{u}$ は速度、p は圧力、ν は動粘性係数、f は $\nabla\cdot f=0$ を満たす外力である。これを渦度形式で表現した方程式を基礎方程式とした。

この NS 方程式を離散化する方法は幾つかあるが、ここではスペクトル法を用いた。スペクトル法は、物理空間の変数を直交関数によって級数展開するものであり、物理空間の変数が十分滑らかであるという条件の下では、項数の増加に対して解が指数的に収束するという性質があり、精度が高く、乱流の振舞いの解析や大気大循環モデルに用いられている方法である。特に、乱流解析では渦のスペクトル成分に関する統計量が重要な物理量となるので、スペクトル法は最適な手法となっている。ここでは、周期境界条件に対応するために 3 次元フーリエ級数展開を用いた。式(2)及び渦度方程式をフーリエ級数により離散化し、整理すると次の式が得られる。

$$\left(\frac{d}{dt}+\nu|k|^2\right)\tilde{u}_k = \tilde{s}_k - k\frac{(k\cdot\tilde{s}_k)}{|k|^2} + \tilde{f}_k \qquad (3)$$

ここで、$\tilde{s}_k$ は$-(\mathrm{u}\times\omega)$のフーリエ係数である。非線形項 $\tilde{s}_k$ の計算には擬スペクトル法が有効である。すなわち、フーリエ空間における畳み込みを計算するために、u のフー

リエ係数及びωのフーリエ係数を実空間に変換し、実空間で各格子点(格子点数 N^3)の外積を求めた後、それのフーリエ係数を求める方法である。この 1 回の評価のために、3 次元フーリエ変換を 9 回適用しなければならないが、高速フーリエ変換（FFT）を用いれば、非線形項の計算量を $O(N^4)$から $O(N^3 \log N)$にすることができ、フーリエ空間における畳み込み演算(直接和)を行った場合の計算量 $O(N^6)$と比べると N が大きい場合には大幅に計算時間の節約が可能である。また、物理空間の値は実変数であるから、実 FFT を適用して計算量をさらに減らすことが可能である。

しかし､擬スペクトル法を用いた非線形項の計算ではエイリアジング誤差が生じるため、畳み込みの正確な値を得るためにはその誤差を除去する必要がある。高波数モードの打ち切りだけでエイリアジング誤差を除去できる 3/2 則は簡便な方法であり、大気大循環モデルでも用いられているが、畳み込みの正確な値が求まるモード数(有効モード数と呼ぶ)はフーリエ項数 N^3 の$(2/3)^3$ になり、3 次元実 FFT で N^3 の計算を行う割には十分な解像度が得られない。一方、phase shift 法は、実空間において（1,1,1）方向に半メッシュだけずらした(phase shift した)格子点でも非線形項の評価を行うことによってエイリアジング誤差を除去する方法である。波数 $k > \sqrt{2}N/3$ の高波数モード打ち切りと合わせて完全にエイリアジングエラーが除去可能である。開発したコードではこの方法を用いた。

また、式(3)の常微分方程式の時間積分には 4 次精度の 4 段ルンゲ・クッタ法を用いた。

3.2 並列化手法

前節の数値計算法に基づく逐次版 DNS コード Trans7 を開発した。このコードは Fortran で書かれており、主要な変数の個数は格子点数 N^3 の 25 倍である。エネルギースペクトルの最小値と最大値が表現できるように変数はすべて倍精度変数とすると、N=512 では約 25GB が必要となり、地球シミュレータの一つの計算ノードで実行できない。より大きな N に対するメモリ容量を見積もると、N=4096 で約 12.5TB になり地球シミュレータ全体でも計算不可能である。このため、格子点数 4096^3 では 3 次元 FFT 部分を含む非線形項の計算を倍精度演算、そのほかの部分は単精度演算として実行した。

コードの並列化では、スペクトル空間のフーリエ係数を k_3 方向、物理空間の変数を y 方向に分割する領域分割法を用いた。この時､3 次元実 FFT においてｚ方向に 1 次元 FFT を適用する部分とｙ方向に 1 次元 FFT を適用する部分の間でデータ転置を行うこととし、MPI ライブラリを用いてそれを実現した。データ転置では、全ての MPI プロセス間同士でのデータ交換が必要である。MPI プロセス数を n_{mpi} とすると、1 回の 3 次元実 FFT において、ひとつの MPI プロセスから、他の分割領域の計算を担当する（n_{mpi}-1）個のうち任意のひとつの MPI プロセスに送られるデータ個数は $N^3/2(n_{mpi})^2$ である。各 MPI プロセ

スは自分以外の転送先に対して、$N^3/2(n_{mpi})^2$ 個のデータを$(n_{mpi}-1)$ 回転送し、$(n_{mpi}-1)$個の他の MPI プロセスから送られてくるデータを受けとって、データ転置が完了する。

また、地球シミュレータのハードウェア性能を十分発揮させるために、地球シミュレータ特有の並列プログラミング技法、すなわちプロセッサ内ベクトル処理、ノード内（共有メモリ）並列処理、ノード間（分散メモリ）並列処理の 3 レベルの並列処理を考慮したプログラミング技法を適用した。これにより高効率の計算性能が得られる。本コードでは、計算時間の 90%以上が 3 次元高速フーリエ変換（FFT）によって消費されており、この部分の高速化が高い自由度の DNS 達成のポイントであった。並列化 3 次元 FFT では、 3 つの空間軸の内、ひとつの軸（例えば z 軸）に対し領域分割法によるノード間並列処理を行い、その他の軸（ x , y 軸）の処理をノード内並列処理とベクトル処理で行った。特に 2 基底 FFT で見られるメモリアクセスのボトルネックを避けるために、ループ内演算数が多くメモリアクセスの少ないベクトル処理可能な 4 基底 FFT を用い、この部分の実効性能をピーク性能の約 60%に引き上げた。また、分割方向である z 軸に対する FFT では、その計算を 1 つのノード内で実行出来るように、領域分割軸を変更するためのデータ転置を行うが、高速なノード間クロスバスイッチとリモートメモリへの直接アクセス可能なハードウェア（RDMA 機構）によって高速に実行することが出来た。

3.3 シミュレーション結果

Trans7 による計算結果の妥当性を検証するために、Trans7 を用いて N=512(格子点数 512^3)の DNS を行った。速度場の初期条件は、エネルギースペクトル $E(k)=a\,k^4\exp\{-b\,k^2\}$ に従う一様等方的なランダムな速度場で、周期境界条件を満足するように与えた。$\Delta t = 0.001$、動粘度 $\nu = 0.000763$ とし、十分乱流場が発達しエンストロフィー(渦度の 2 乗平均の 1/2)が安定する t=10 まで時間発展させた。乱流の統計的準定常状態を得るための外力 f として、低波数領域($|k| < 2.5$)に負の粘性を仮定し、その値は全エネルギーが一定に保たれるように各時間ステップで調整した。t=10 でのテーラー長λに基づくレイノルズ数は、$R_\lambda \fallingdotseq 164$ であった。この結果、波数領域 $4 < k < 16$ 近傍において慣性小領域におけるコルモゴロフのエネルギースペクトルが確認された。

さらに、格子点数 2048^3、4096^3 の世界最大規模の数値シミュレーションを実施した[7]、[8]。図 2 は規格化した乱流場のエネルギースペクトルである。格子点数 2048^3 の DNS では、$E(k) \fallingdotseq k^{-5/3}$ となるコルモゴロフの $k^{-5/3}$ 則がより広い波数領域で確認できる。512 台のノードを用いた時の格子点数 2048^3 の DNS の計算時間は、1 時間ステップあたり約 7 秒、約 16 テラフロップスの高い実効性能（ピーク性能比 50%）を達成した。これは地球シミュレータのハードウェア性能を最大限に引き出す注意深い並列プログラミング技法による

ところが大きい。格子点数4096^3のDNSの1時間ステップの計算時間は約30秒であった。詳細なシミュレーションの結果については、参考文献[6]、[7]をご覧頂きたい。

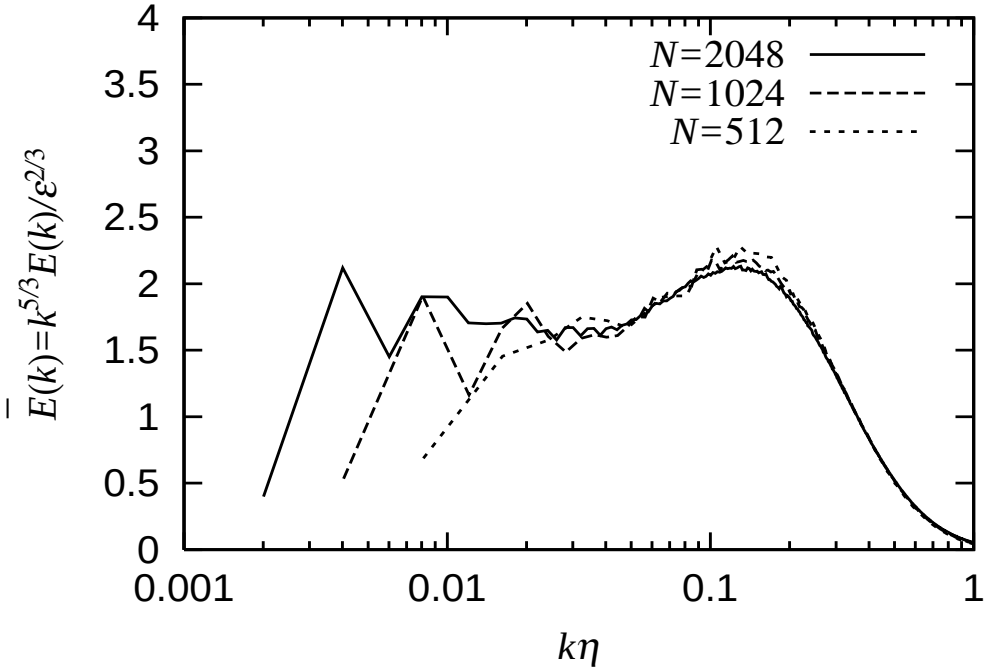


図 2 規格化した乱流場のエネルギースペクトル

4. まとめ

地球シミュレータハードウェアの性能評価を行うと共に、超大規模な乱流 DNS を実現することを目指し、地球シミュレータ用に最適化した DNS コードを開発した。その結果、512 ノードを用いた格子点数 2048^3 の DNS において、16.4Tflops の超高速計算と従来の DNS と比較して桁違いに大規模な格子点数 4096^3 の DNS を実現することが出来た。DNS によって得られた乱流場が、乱流の統計力学の構築など今後の乱流研究において大きな成果が期待される。

参考文献

[1] 後藤，石原，乱流の謎にせまる計算科学，パリティ，Vol.17，No.10，pp.27-33 (2002)

[2] 横川，谷，地球シミュレータ計画，情報処理, Vol.41, No.4, pp.369-374 (2000)

[3] S. Habata, M. Yokokawa, and S. Kitawaki, “The Earth Simulator System,” NEC Research and Development, Vol.44, No.1, pp.21-26 (2003)

[4] 上原，田村，横川，地球シミュレータの計算ノード上での MPI 性能評価，「計算科学とハイパフォーマンスコンピューティング」シンポジウム（HPCS2002），pp.73-80 (2002)

[5] 横川，斉藤，石原，金田，地球シミュレータ上の一様等方性乱流シミュレーション，「計算科学とハイパフォーマンスコンピューティング」シンポジウム（HPCS2002），pp.125-131 (2002)

[6] M. Yokokawa, K. Itakura, A. Uno, T. Ishihara, and Y. Kaneda, “16.4-Tflops Direct Numerical Simulation of Turbulence by a Fourier Spectral Method on the Earth Simulator,” Proc. Of the IEEE/ACM SC2002 Conference (CD-ROM), Baltimore, November 16-12 (2002).

[7] Y. Kaneda, T. Ishihara, M. Yokokawa, K. Itakura, and A. Uno, “Energy dissipation rate and energy spectrum in high resolution direct numerical simulations of turbulence in a periodic box,” Physics of Fluids, Vol.15, No.2, L21-L24 (2003).

[8] 横川，「地球シミュレータ」による世界最大規模乱流シミュレーション，パリティ，Vol.17，No.10，丸善 (2002)

[9] 宇野，板倉，横川，石原，金田，地球シミュレータ上での流体コードのスケーラビリティ評価，情処研報 02-HPC-91，pp.55-60 (2002)

3.4.4 Web サービスの技術動向

藤田 悟 講師

1. はじめに

インターネットを通じてシステムを連携させる技術として、Web サービスが注目を集めてきている。Web サービスという概念は、元々、サービス指向アーキテクチャという発想から生まれたものである。すなわち、大きなビジネスを動かすシステムを設計する時に、全てをゼロから作り始めるのではなく、既存のサービスを利用して、それらを組み合わせることでビジネス全体を構成していくという考えに基づく。このサービス指向アーキテクチャを、サービス提供者、サービス要求者、そして、サービス仲介者の 3 者の関係として図 1 に示す。まず、サービス提供者とは文字通りサービスを保有していて、第三者に対してそれを提供する役割である。サービス提供者は、第三者から自分のサービスを見つけてもらうために、サービス仲介者にサービスの登録を行い、サービスを公開する必要がある。このサービス公開時に重要なことは、アクセスするインタフェース情報と、エンドポイントを明確にすることである。一方、サービス利用者は、サービス仲介者に対して、サービス検索の要求を出す。検索結果として、必要なサービスにアクセスするためのインタフェースとエンドポイントを取得する。後は直接にサービス利用者からサービス提供者に対してサービスの結合を行い、相互に通信を行って、サービスを利用することができる。

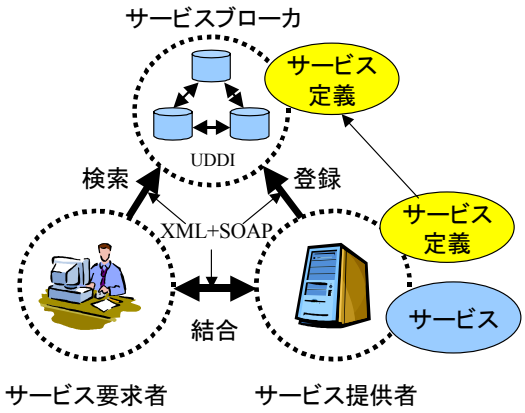


図1 Web サービスアーキテクチャ

このサービス指向アーキテクチャを支える重要な技術は、インタフェース定義と、インターネットを通じた通信手順と、サービス登録/検索技術である。インターネットという疎結合な環境で利用することを考慮し、また、計算機や OS に非依存な技術を利用するとい

う考えから、全てを XML(eXtensible Markup Language)で記述する標準化が進められてきた。これがインタフェース定義言語である WSDL (Web Services Description Language)[1]と、通信規約である SOAP (Simple Object Access Protocol)[2]と、サービスディレクトリ UDDI (Universal Discription, Discovery, and Integration)[3] である。本報告では、これらの Web サービスの基本となっている技術を中心に概説した後、いくつかの課題を解決するための技術動向について紹介する。

本論に入る前に、SOAP が、Web サービスの必要条件というわけではないことも述べておきたい。例えば、WSDL によるインタフェースを中心にして、CORBA、Java RMI 等の状況に応じた通信規約を用いるものも Web サービスに含めるという考え方もある。また、UDDI のようなサービスディレクトリを設けずに、個別のサービスがインタフェース定義を持つという考え方(WSIL: Web Services Inspection Language)[4]もある。W3C の Web Services Architecture WG[5] による Web サービスの定義を次に示すが、この定義の中にも、SOAP というキーワードはでは、直接現れていない。

「Web サービスとは、URI で識別できるソフトウェアシステムであり、インタフェースとバインディングが XML を用いて定義され、記述されているものである。他のソフトウェアシステムは、この定義を検索できる。これらシステムは、定義に示された方法で、インターネットプロトコルによって伝送される XML ベースのメッセージを利用して、Web サービスと相互通信することができる。」

それでは、SOAP は Web サービスにとって重要でないかというとそうは言っておらず、W3C による Web サービスの定義文に続く説明にも、参照アーキテクチャとして、また、Web サービスを汎用につなぐための上位レベルの表現として、SOAP、WSDL が標準的技術であることが示されている。SOAP の出現により、Web サービスという考え方が広まり、Web サービス自体は、SOAP に限定されない広い概念として拡張されてきていると考えた方が良いであろう。

2. Web サービスの基本技術

2.1 SOAP

SOAP は、XML を用いてメッセージ交換するための通信規約である。下位の通信層は通常 HTTP を用いるが、これに限定した仕様ではなく、例えば SMTP を用いることもできる。SOAP の語源は、Simple Object Access Protocol の頭文字をとったものであるが、オブジェクト指向的な側面は少なく、また、最近は simple でもなくなったということから、SOAP という名前だけ残り、その語源は問わないようになってきている。

SOAP によって送受信される XML は、ルートに Envelope 要素があり、その下位要素

に、Header と Body 要素が存在する構造を持つ。Header 部は、主にメッセージの制御のために扱われ、例えば、署名情報などが格納される。Body 要素に、通信したい内容が格納される。この Body の中身は、XML Schema[6] で定義した任意の XML 文書が記述できるが、特に RPC(Remote Procedure Call)を実現するために用いる表現の規約を持っている。図 2 にその典型的な SOAP RPC に対する XML 文書の送受信例を示す。

```
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">

  <SOAP-ENV:Header>
  </SOAP-ENV:Header>

  <SOAP-ENV:Body>
    <m:GetLastTradePrice xmlns:m="Some-URI">
      <m:CompanyName>NEC</m:CompanyName>
    </m:GetLastTradePrice>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

(1) RPC 要求メッセージ

```
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">

  <SOAP-ENV:Header>
  </SOAP-ENV:Header>

  <SOAP-ENV:Body>
    <m:GetLastTradePriceResponse xmlns:m="Some-URI">
      <m:price>449</m:price>
    </m:GetLastTradePriceResponse>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

(2) RPC 応答メッセージ

図2 SOAP のメッセージ例

これは、株価を検索する例であり、要求側では、Body 要素の中に、メソッド名に相当する GetLastTradePrice という要素を作成し、その中に、引数に相当する CompanyName 要素を添える形で XML 文書を作成している。一方、応答側は、GerLastTradePriceResponse という返答を表す要素中に、price 要素を用いて現在の株価を返却する。このように、SOAP RPC はメソッドコールと返却値の関係を、直感的に XML でシリアライズしたものと考えて良い仕様になっている。この RPC に用いる引数や返却値に利用できる型は、XML Schema で定義された基本データ型(byte, short, integer, double など)と、その複合型(構造体と配列)である。構造体は、XML Schema の文書型定義を用いて表現し、配列は、SOAP encoding による特別な表現が用意されている。

この他、SOAP には、SOAP Messages with Attachments [7]という仕様があり、MIME などでエンコーディングした複数のパートにまたがる文書を転送することができる。文書の本体で SOAP による通信方法を規定し、添付ファイルで付属情報としての画像やテキストなどを付加して転送することが可能になっている。

2.2　WSDL

WSDL は、Web サービスのサービスインタフェースを XML で定義するための言語である。CORBA の IDL や、Java の interface に近いものであると考えて良い。WSDL の XML 文書は、definitions 要素によって記述されており、その中に次のような要素が含まれている。

- types
  引数や返却値に現れる型を宣言する部分。XML Schema による文書型の規定を行う。
- message
  交換する一つのメッセージのフォーマットを規定する。message 要素の中に part 要素があり、これが引数のひとつずつに対応する。
- portType
  まとまった操作の集合を表す。portType の中には、一つの操作を表す operation 要素がある。さらに、operation 要素の中には、input 要素、output 要素を書くことができ、これが、それぞれ message 要素と結び付けられる。operation 要素が、input 要素だけを持つと、受信だけの一方向メッセージの定義となり、input 要素に続いて output 要素が定義されていると、要求-応答型の RPC に相当するサービス定義となる。

- binding
  転送プロトコルへのバインディングを行う。例えば、SOAP binding では、WSDL で定義したインタフェースと SOAP メッセージの関係を定義する。
- service
  service 要素は port 要素を持ち、サービスのエンドポイントなどを指定する。

SOAP の説明で用いたメッセージ例について、そのインタフェース定義を行う WSDL の例を図 3 に掲載する。

図 3 のように、WSDL は非常に複雑であり、これを人手で記述するのは困難である。通常は、例えば、Java の interface を定義し、その interface 定義から、WSDL を自動生成することができる。また、逆に、WSDL 定義から、Java の stub クラスを自動生成することもできる。言い換えれば、Java でサービスのプログラムを作成し、そのインタフェース定義から WSDL を生成し、呼び出し側は WSDL から Java のアクセスクラスを生成して、それに対してサービス呼び出しを行うクライアントプログラムを作成するという形になり、プログラマは、Java のプログラミングをするだけで、クライアントとサーバの間の RPC を SOAP と WSDL を用いて通信する Web サービスプログラムが書けるようになっている。

```
<?xml version="1.0"?>
<definitions name="StockQuote"
   targetNamespace="http://example.com/stockquote.wsdl"
             xmlns:tns="http://example.com/stockquote.wsdl"
             xmlns:xsd1="http://example.com/stockquote.xsd"
             xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
             xmlns="http://schemas.xmlsoap.org/wsdl/">
   <types>
      <schema targetNamespace="http://example.com/stockquote.xsd"
             xmlns="http://www.w3.org/1999/XMLSchema">
         <element name="TradePriceRequest">
            <complexType>
               <all>
                  <element name=“CompanyName" type="string"/>
               </all>
            </complexType>
         </element>
         <element name="TradePriceResult">
            <complexType>
               <all>
                  <element name="price" type="integer"/>
               </all>
            </complexType>
         </element>
      </schema>
   </types>

   <message name="GetLastTradePrice">
      <part name=“TradePriceRequest" element="xsd1:TradePriceRequest"/>
   </message>
   <message name="GetLastTradePriceResponse">
      <part name=“TradePriceResult" element="xsd1:TradePriceResult"/>
   </message>

   <portType name="StockQuotePortType">
      <operation name="GetLastTradePrice">
         <input message="tns:GetLastTradePrice"/>
         <output message="tns:GetLastTradePriceResponse"/>
      </operation>
   </portType>

   <binding name="StockQuoteSoapBinding" type="tns:StockQuotePortType">
      <soap:binding style="document"
                    transport="http://schemas.xmlsoap.org/soap/http"/>
      <operation name="GetLastTradePrice">
         <soap:operation soapAction="http://example.com/GetLastTradePrice"/>
            <input>
               <soap:body use="literal"
                  namespace="http://example.com/stockquote.xsd"
                  encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
            </input>
            <output>
               <soap:body use="literal"
                  namespace="http://example.com/stockquote.xsd"
                  encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
            </output>
         </soap>
      </operation>
   </binding>
</definitions>
```

図3　WSDL の記述例

2.3 UDDI

UDDI は、Web サービスの公開と検索を行うレジストリの仕様を規定しており、主にレジストリのデータ構造と、レジストリへのアクセスインタフェースの定義を行っている。まず、データ構造は、businessEntity, businessService, bindingTemplate の 3 階層からなる XML 文書と、tModel を表現する XML 文書からなる。

- BusinessEntity
  サービスを提供する企業に関する情報(企業名、所在地、企業コードなど)を記述する。
- businessService
  サービスの内容や種類に関する情報を記述する。
- bindingTemplate
  サービスに接続するための情報(tModel へのリンク、エンドポイント)を記述する。
- tModel
  サービスにアクセスするためのインタフェース情報(WSDL 等)へのリンクを記述する。

これらの情報を登録、検索するための API としては、次のものが用意されている。ただし、XXX で示される部分には、検索の対象となる businessEntity、businessService、bindingTemplate、tModel を入れたものが用意されている。

- 参照 API (Inquiry API)
  - find_XXX: キーワードなどを用いた検索のための API。検索の結果は対象となった要素のキーがリスト形式で返される。
  - get_XXX: find_XXX で検索されたキーから、その実体（内容）を取り出すときに使用する API。
- 発行 API (Publication API)
  - save_XXX: オブジェクトを登録するための API。
  - delete_XXX: レジストリのエントリを削除するための API。

UDDI は、サービス提供者とサービス要求者を結びつける役割を果たす。当初は、世界共通のグローバル UDDI を立ち上げたが、その後、企業内や団体内においてプライベートな用途の UDDI を立ち上げる要求が高まってきている。また、Web サービスは、必ずしも UDDI を必要としていない現状もある。Web サービスを利用していると言われる多くのシステムは、WSDL と SOAP を用いて直接に相手と接続しており、UDDI を利用した

動的なサービス検索、サービス呼び出しというレベルには至っていない。

3. Web サービスと、他の技術との比較

Web サービスの特徴を述べるために、他の類似技術との比較を述べる。

- Web サイト

  Web サイトは、主に人に対してサービスを提供し、ブラウザを通して人と対話することを特徴とする。一方、Web サービスは、主にシステムとシステムを結ぶための技術であり、人が介在しない通信を行う。
- オブジェクト指向

  オブジェクト指向は、ソフトウェアの設計と開発において、オブジェクトというコンポーネント単位を用いて抽象化する技術である。Web サービスにおけるサービス指向とは、オブジェクトより上位のサービスという大きな枠組でビジネスを抽象化するものである。
- CORBA

  Web サービスは、HTTP などのインターネットプロトコルを基本としていることから、CORBA よりも疎結合なシステム連携技術であると言える。
- EDI(Electronic Data Interchange)

  EDI は専用線、専用プロトコルを利用して発展してきた。Web サービスは、汎用でかつ標準技術に基づくプロトコルによる EDI を可能にする。
- SCM(Supply Chain Management)

  SCM は、固定的な企業間連携の枠組みあったのに対し、Web サービスでは、動的な企業間連携の枠組みを提供する。

4. Web サービスの技術動向

Web サービスはインターネットを利用した簡単で疎結合のシステム連携技術として開発が始まったものの、実際にビジネスに利用するためには、セキュリティやトランザクション、高信頼なメッセージ交換など、様々な周辺技術が必要になってきている。また、相互接続性の確保も重要であり、様々な相互接続実験が実施されている。この節では、このような Web サービスを取り巻く技術動向について、その代表的なもののいくつかを取り上げて簡単な説明を加える。詳細な説明は、標準化団体のページを参照されたい。

4.1 WS-Security[8]

Web サービスのセキュリティを確保するために OASIS で標準化が進められている技術で、特に秘匿性(メッセージの盗聴防止)と完全性(メッセージの改ざん防止)のための情報を付与する文書形式について定めている。具体的には、基盤技術として XML 署名や XML 暗号を用いて、これらに関する情報を SOAP Header の中の Security 要素に埋め込む形式を用いている。

4.2 SAML (Security Assertion Markup Language)[9]

SAML は、認証／認定情報を交換するための仕様を定めており、V1.0 の仕様が OASIS の標準として承認された。この技術を用いることにより、一度だけの認証で複数の企業が提供するサービスを利用できるようにする Single Sign On の環境が実現できるようになる。

4.3 Liberty Alliance[10]

ネットワーク上に分散管理された個人情報を、ネットワークアイデンティティを中心に連携させる仕様作りを行っている。特に、利用者の ID 管理を集中化させないで、各サービスサイトが別々の ID 管理を行いながらも、互いに情報交換して、サービス連携をスムーズに行う技術に着目している。SAML をベースにして、その上に、プライバシー情報管理、サービス連携に関する仕様をまとめている。2003 年 1 月現在、V1.1 の仕様が公開されている。

4.4 WS-Transaction[11], BTP (Business Transaction Protocol)[12]

複数の Web サービスを一つのトランザクションとしてまとめる技術が注目を集めている。これは、通常のデータベースのトランザクションと違って、長期間に渡るトランザクションとなることが特徴である。このため、リソースをロックするような技術ではなく、一旦はデータをコミットした後で、問題があれば後でキャンセルできる Conpensation と呼ばれる仕掛けが用いられる。OASIS では、BTP と呼ばれる標準化を進めてきたが、最近、新たに IBM と Microsoft 社から、WS-Transaction という新しい技術提案があり、トランザクションの標準化動向は混沌としている。

4.5 WS-I (Web Services Interoperability Organization)[13]

Web サービスの相互運用性を確保するために設立された団体。SOAP, WSDL, UDDI, XML Schema を中心に、それぞれの仕様の間で矛盾なく運用できるための詳細な規程をま

とめた Basic Profile と、それに基づくテストツールとサンプルアプリケーションの開発を進めている。2003 年 1 月現在、Basic Profile 1.0 の仕様が公開されている。

4.6　ebXML[14]

ebXML は Web サービスを補完する技術として参照されることが多い。SOAP をベースにした企業間取引のための XML 通信技術の体系を策定している。元々は、UN/CEFACT と OASIS が共同で、ebXML イニシアチブの活動を行い、2001 年 5 月に V1.0 の仕様を公開した。その後は、UN/CEFACT と OASIS は別々に、ebXML の個別仕様の策定作業を進めている。2003 年 1 月現在では、部分的に V2.0 の仕様が公開されている。

ebXML は、以下に示すように、メッセージの規定からビジネスオブジェクトの規定まで、幅広い領域に対して標準を定めている。

- メッセージサービス仕様(Message Service Specification)
  SOAP Messages with Attachments をベースにした企業間取引のためのメッセージ交換仕様を定める。高信頼メッセージ(メッセージ再送、重複受信防止、メッセージ順序保証)、署名付きメッセージ、同期/非同期メッセージなどをサポートした仕様である。SOAP の Header 部分に、メッセージを制御する情報(送受信会社情報、契約番号、対話番号、署名など)を記述し、Body 部分に必要な文書への参照ポイントを格納する。実際に取引の内容を表す文書は、添付ファイルとして交換される。
- パートナープロファイル/契約(Collaboration-Partner Profile and Agreement)
  ある企業の受付け可能なメッセージ仕様について記述するプロファイル(CPP)と、このプロファイルに基づいて企業間で取り決めた通信に関する契約(CPA)を定義する。具体的には、高信頼メッセージ使用の有無や、署名の有無、サービスのエンドポイントなどの情報が記述される。
- ビジネスプロセス仕様(Business Process Specification Schema)
  メッセージ交換のシーケンスを定義するための言語。
- レジストリ(Registry)
  企業プロファイル(CPP)や、ビジネスメッセージを記述するためのコアコンポーネントなどを格納するためのディレクトリ。
- コアコンポーネント(Core Components)
  ビジネスメッセージを記述するための共通オブジェクトや共通プロセスを規定する。

ebXML 技術も相互接続性が重要なことから、様々な相互接続実験の活動が行われてい

る。日本では、電子商取引推進協議会(ECOM)が中心に、相互接続実験を実施し、NEC、富士通、日立、NTT、インフォテリアの 5 社による基本メッセー陣の相互接続実験の成功が報告されている。

5. まとめ

以上、Web サービスの基本技術の紹介と、最近の技術動向について述べた。Web サービスは、インターネット環境における疎結合なサービス統合アーキテクチャを提供することから、特にビジネス領域で注目を集めており、各社が開発ツールの提供を開始している。Web サービスの基本となる技術には、SOAP、WSDL、UDDI などがあり、それを紹介したが、これらは Web サービスの必須技術ではないことも注釈した。また、ビジネスに利用するための Web サービスの強化が進められており、セキュリティ、トランザクション、高信頼メッセージなどにおいて、標準化が進められつつある。WS-I のような、様々な相互接続実験の活動も並行して進められている。

多くの Web サービスの技術は、標準化途上であり、今後も仕様の改版が進められていくであろう。まだ未成熟な部分も多いが、インターネットを通じてシステムとシステムが連携するという非常に魅力的な技術である。今後とも、その動向には目を離せないものがある。

参考文献

[1] Web Services Description Language (WSDL) 1.1, http://www.w3.org/TR/wsdl

[2] Simple Object Access Protocol (SOAP) 1.1, http://www.w3.org/TR/SOAP/

[3] UDDI Version 3.0, http://uddi.org/pubs/uddi-v3.00-published-20020719.pdf

[4] Specification: Web Services Inspection Language (WS-Inspection) 1.0, http://www-106.ibm.com/developerworks/webservices/library/ws-wsilspec.html

[5] Web Services Architecture Working Group, http://www.w3.org/2002/ws/arch/

[6] XML Schema, http://www.w3.org/XML/Schema

[7] SOAP Messages with Attachments: http://www.w3.org/TR/SOAP-attachments

[8] WS-Security, http://www.oasis-open.org/committees/wss/

[9] SAML 1.0 Specification Set, http://www.oasis-open.org/committees/security/

[10] Liberty Alliance, http://www.projectliberty.org/

[11] Web Services Transaction (WS-Transaction), http://www-106.ibm.com/developerworks/library/ws-transpec/

[12] Business Transaction Protocol (BTP),
http://www.oasis-open.org/committees/business-transactions/
[13] Web Services Interoperability Organization (WS-I), http://www.ws-i.org/
[14] ebXML, http://www.ebxml.org/

第 4 章 あとがき

本報告書は、AITEC における HECC ワーキンググループとして最後の報告書となる。HECC ワーキンググループは 1999 年（平成 11 年）から活動をはじめ、本年度で 4 年目となるが、その前身である「ペタフロップスマシン技術調査ワーキンググループ」が、1996 年（平成 8 年）から既に 3 年間の調査を行っており、合計すると 7 年に渡り、ハイエンドコンピュータやその周辺技術について技術調査や議論を行ってきたことになる。そもそも、「ペタフロップスマシン技術調査ワーキンググループ(以下ペタフロップス WG と略)」を始めるきっかけは、米国における HPCC 計画によるハイエンドコンピュータや高速通信網計画、およびそれに続いて 1996 年から開始された ASCI（戦略的コンピューティング加速構想）などによって、米国におけるハイエンドコンピュータの開発が戦略的に行われ始めたという事情がある。このような状況の中、米国ではペタフロップスという、テラ(tera)の 1000 倍の性能を有する計算機システムの研究開発に対する検討も行われはじめていた。すなわち 1991 年に Purdue 大学で開催された HPCC のグランドチャレンジにおいて、初めてペタフロップスコンピュータについて議論が行われ、その後、関連のワークショップが毎年開催されるようになった。特に、1994 年 2 月にカリフォルニア州パサディナ(Pasadena)で開かれたワークショップ「The Workshop on Enabling Technologies for Peta(FL)OPS Computing」で、ペタフロップス級の超高性能計算機を開発するための技術的な検討が行われ、ペタフロップスレベルのシステム、アプリケーション開発を進展させる議論が行われた。このような背景の下、AITEC 内に、本 WG の前身である、ペタフロップス WG が設置されたわけである。

ここで、本ワーキンググループの 7 年間の調査検討の軌跡を、少し振り返ってみたいと思う。最初の年のペタフロップス WG においては、当初のもくろみはいかにしてペタフロップスマシンを開発するか、あるいはそのようなプロジェクトを立ち上げる必要があるかというものであったが、我々の議論は少し違う方向に向かってしまった。すなわち、議論の中心は、日米の技術格差が特に基幹ソフトウェアや産業基盤ソフトウェアにおいて顕著であるという認識が得られ、その打開のために何をなすべきかという点が強調されたことである。この認識は、現時点では、昨年発表された「地球シミュレータ」の開発能力などを見ると、ハードウェア開発などにおいては、米国に引けをとらないということが証明されたが、IT における他の分野や総合力をみると、やはり遅れをとっている感は否めない。この点に関して、1997 年（平成 9 年）の報告書のあとがきに書いた次の指摘は、現状でもやはり当てはまるのではないだろうか。「日本の技術開発で欠けているものが自ずと明

らかになってくる。それは、共通した仕様や共通のインタフェースの抽出とそれをいかに効率よくインプリメントするかといった戦略である。これは、いわば構成力（一般的には構想力と呼ばれるかも知れない）の差であると言い換えられる。したがって、これから我が国における情報処理技術を活性化し、米国に対抗するためには、情報処理技術における構成力を高める必要があるのではないだろうか。」

ペタフロップス WG の 3 年目（1998 年）の調査においては、調査の対象をペタフロップスに限定することなく、近未来の情報処理産業を見据えて、どのような技術開発を行うべきかについてヒアリングや議論を行った。これが、1999 年から始まる HECC-WG のさきがけとなるものであった。そして、1999 年から 4 年間、HECC-WG として、高性能コンピューティングに関連する情報処理技術のあるべき姿や方向性に関して、調査や検討を行ってきた。

HECC における 4 年間の調査などを振り返ってみると、ハイエンドコンピューティングからグリッド（グローバルコンピューティング）、そして P2P や Web サービスなどへと調査の対象が拡大してきたわけであるが、それは計算機システムとネットワークとの関係がより密接になり、離れがたいものになってきていることの証明でもあった。この傾向はますます強いものとなり、これからの情報処理技術の流れの中で、ますますネットワークを介した情報の通信とプロセッサによる情報の処理が一体化されたシステムが構成される方向に向かうであろう。その 1 つの表れとして、たとえばユビキタスコンピューティングという考え方が広く言われるようになってきていることが挙げられる。このようなユビキタスコンピューティングが情報システムの主流になるとすると、昨年の報告書のあとがきでも指摘したように、「脱 PC」という状況が出現すると考えられる。そのような状況においては、PC 時代を WinTel という 2 大企業が制覇したような状況にはなっていかないと思われる。これは、我が国にとっては挽回のチャンスであり、その時に備えて、半導体技術や組み込みシステムさらにはネットワーク技術やソフトウェア、そして最も重要なアプリケーションにいたるまで、技術のポテンシャルを高めておく必要があると考えられる。

最後に、7 年間の長きにわたり、本ワーキンググループの活動について様々な局面から支えていただいた内田所長をはじめとする AITEC の関係者の皆様に感謝の言葉を述べ、本ワーキンググループとしての報告書のあとがきを結ぶこととする。

（山口　喜教　主査）

付表1　ハイパフォーマンス・コンピュータ世界のTop 20

順位	メーカ	Computer	R_{max} (GFlops)	設置場所	国	設置年	使用分野	CPU数
1	NEC	Earth Simulator	35860	Earth Simulator Center	Japan	2002	Research	5120
2	HP	ASCI Q - AlphaServer SC ES45/1.25 GHz	7727	Los Alamos National Laboratory	USA	2002	Research	4096
3	HP	ASCI Q - AlphaServer SC ES45/1.25 GHz	7727	Los Alamos National Laboratory	USA	2002	Research	4096
4	IBM	ASCI White SP Power3 375MHz	7226	Lawrence Livermore National Laboratory	USA	2000	Research	8192
5	Linux NetworX	MCR Linux Cluster Xeon 2.4GHz - Quadrics	5694	Lawrence Livermore National Laboratory	USA	2002	Research	2304
6	HP	AlphaServer SC ES45/1GHz	4463	Pittsburgh Supercomputing Center	USA	2001	Academic	3016
7	HP	AlphaServer SC ES45/1GHz	3980	Commissariat a l'Energie Atomique (CEA)	France	2001	Research	2560
8	HPTi	Aspen Systems, Dual Xeon 2.2GHz - Myrinet2000	3337	Forecast Systems Laboratory - NOAA	USA	2002	Research	1536
9	IBM	pSeries 690 Turbo 1.3GHz	3241	HPCx	UK	2002	Academic	1280
10	IBM	pSeries 690 Turbo 1.3GHz	3164	NCAR(National Center for Atmospheric Research)	USA	2002	Research	1216
11	IBM	pSeries 690 Turbo 1.3GHz	3160	Naval Oceanographic Office(NAVOCEANO)	USA	2002	Research	1184
12	IBM	SP Power3 375 MHz/16Way	3052	NERSC/LBNL	USA	2002	Research	6656
13	IBM	pSeries 690 Turbo 1.3GHz	2560	ECMWF	UK	2002	Research	960
14	IBM	pSeries 690 Turbo 1.3GHz	2560	ECMWF	UK	2002	Research	960
15	Intel	ASCI Red	2379	Sandia National Laboratories	USA	1999	Research	9632
16	IBM	pSeries 690 Turbo 1.3GHz	2310	Oak Ridge National Laboratory	USA	2002	Research	864
17	Atipa Technology	P4 Xeon 1.8GHz - Myrinet	2207	Louisiana State University	USA	2002	Academic	1024
18	HP	AlphaServer SC ES45/1GHz	2164	NASA/Goddard Space Flight Center	USA	2002	Research	1392
19	IBM	ASCI Blue-Pacific SST, IBM SP 604e	2144	Lawrence Livermore National Laboratory	USA	1999	Research	5808
20	IBM	pSeries 690 Turbo 1.3GHz	2140	US Army Research Laboratory(ARL)	USA	2002	Research	800

出典：http://www.top500.org/における2002年11月版
（全てのカテゴリーの総合順位でRmax順にソート）

付表2　クラスタコンピュータ世界のTOP20

順位	Top500	メーカ	Computer	R_{max} (GFlops)	設置場所	国	設置年	使用分野	CPU数
1	2	HP	ASCI Q - AlphaServer SC ES45/1.25 GHz	7727	Los Alamos National Laboratory	USA	2002	Research	4096
2	3	HP	ASCI Q - AlphaServer SC ES45/1.25 GHz	7727	Los Alamos National Laboratory	USA	2002	Research	4096
3	5	Linux NetworX	MCR Linux Cluster Xeon 2.4GHz - Quadrics	5694	Lawrence Livermore National Laboratory	USA	2002	Research	2304
4	6	HP	AlphaServer SC ES45/1GHz	4463	Pittsburgh Supercomputing Center	USA	2001	Academic	3016
5	7	HP	AlphaServer SC ES45/1GHz	3980	Commissariat a l'Energie Atomique (CEA)	France	2001	Research	2560
6	8	HPTi	Aspen Systems, Dual Xeon 2.2GHz - Myrinet2000	3337	Forecast Systems Laboratory - NOAA	USA	2002	Research	1536
7	17	Atipa Technology	P4 Xeon 1.8GHz - Myrinet	2207	Louisiana State University	USA	2002	Academic	1024
8	18	HP	AlphaServer SC ES45/1GHz	2164	NASA/Goddard Space Flight Center	USA	2002	Research	1392
9	22	Dell	PowerEdge 2650 Cluster P4 Xeon 2.4GHz	2004	University at Buffalo, SUNY, Center for Computational Res.	USA	2002	Academic	600
10	32	Dell	Vplant Cluster P4 Xeon 2.4/2.0GHz - Myrinet	1272	Sandia National Laboratories	USA	2002	Research	660
11	42	HP	rx5670 Itanium2 Cluster	1090	Energy Company	USA	2002	Industry	545
12	43	Legend Group	DeepComp 1800 - P4 Xeon 2GHz - Myrinet	1046	Academy of Mathematics and System Science	China	2002	Academic	512
13	46	Linux NetworX	LCRC Xeon 2.4GHz - Myrinet	1007	Argonne National Laboratory	USA	2002	Research	361
14	47	HP	AlphaServer SC ES40/833MHz	1007	Japan Atomic Energy Research	Japan	2002	Research	812
15	48	Self-made	Cplant/Ross Cluster	996.9	Sandia National Laboratories	USA	2002	Research	1800
16	51	Self-made	Pentium Xeon Cluster 2.2GHz - SCI3D	960.4	National Supercomputer Center(NSC)	Sweden	2002	Academic	400
17	57	HP	AlphaServer SC ES45/1GHz	862.2	ERDC MSRC	USA	2002	Research	512
18	58	HP	AlphaServer SC ES45/1GHz	862.2	Hewlett-Packard	USA	2002	Vendor	512
19	59	HP	AlphaServer SC ES45/1GHz	862.2	Wright-Patterson Air Force Base/DoD ASC	USA	2002	Research	512
20	61	HP	rx2600 Itanium2 Cluster - Quadrics	851.0	Pacific Northwest National Laboratory	USA	2002	Research	256

出典：http://www.top500.org/における2002年11月版
（Computerクラスを Clusterにしてソートしたもの）

付表3　日本製ハイパフォーマンスコンピュータ　Top20

順位	Top500	メーカ	Computer	R_{max} (GFlops)	設置場所	設置国	設置年	使用分野	CPU数
1	1	NEC	Earth Simulator	35860.0	Earth Simulator Center	Japan	2002	Research	5120
2	26	Hitachi	SR8000/MPP	1709.1	University of Tokyo	Japan	2001	Academic	1152
3	27	Hitachi	SR8000-F1/168	1653.0	Leibniz Rechenzentrum	Germany	2002	Academic	168
4	34	NEC	SX-5/128M 3.2ns	1192.0	Osaka University	Japan	2001	Academic	128
5	49	NEC	SX-6/128M16	982.0	DKRZ - Deutsches Klimarechenzentrum	Germany	2002	Research	128
6	50	NEC	SX-6/128M16	982.0	NEC Fuchu Plant	Japan	2002	Vendor	128
7	53	Hitachi	SR8000-F1/100	917.0	High Energy Accelerator Research Organization /KEK	Japan	2000	Research	100
8	55	Fujitsu	VPP5000/100	886.0	ECMWF	UK	2000	Research	100
9	56	Hitachi	SR8000/128	873.0	University of Tokyo	Japan	1999	Academic	128
10	67	Hitachi	SR8000-G1/64	790.7	Institute for Materials Research /Tohoku University	Japan	2001	Academic	64
11	75	Fujitsu	VPP5000/80	730.0	University of Tsukuba	Japan	2001	Research	80
12	77	Hitachi	SR8000-E1/80	691.3	Japan Meteorological Agency	Japan	2000	Research	80
13	86	NEC	Magi Cluster P3 933MHz	654.0	CBRC - Tsukuba Advanced Computing Center - TACC/AIST	Japan	2000	Research	1040
14	94	Hitachi	SR8000-F1/60	577.0	University of Tokyo /Institute for Solid State Physics	Japan	2000	Academic	60
15	97	Fujitsu	VPP5000/64	563.0	Japan Stomic Energy Research	Japan	2001	Research	64
16	98	Fujitsu	VPP5000/64	563.0	Kyushu University	Japan	2000	Academic	64
17	119	NEC	SX-6/64M8	495.2	NAL	Japan	2002	Research	64
18	120	NEC	SX-6/64M8	495.2	National Institute for Environmental Studies	Japan	2002	Research	64
19	122	Fujitsu	VPP5000/56	492.0	Nagoya University	Japan	1999	Academic	56
20	123	Fujitsu	VPP800/63	482.0	Kyoto University	Japan	1999	Academic	63

出典：http://www.top500.org/における2002年11月版
（Manufacturer LocationをJapanとしてソートしたもの）

平成 14年度ワーキンググループ活動記録

開催日	討議内容
第 1 回 平成 14 年 9 月 17 日	・山口主査　挨拶 ・内田所長　挨拶 ・WG 幹事「平成 14 年度活動方針案」 ・招待講演 東京大学 先端科学技術研究センター　助教授　中村　宏氏 「ソフトウェア可制御オンチップメモリを用いた高性能・低消費電力プロセッサ」 ・委員講演　山口喜教主査 「2010 年の IT 技術」
第 2 回 平成 14 年 10 月 15 日	・招待講演 日本電気(株) インターネット基盤開発本部 マネージャー　藤田　悟氏 「Web サービスの技術動向の紹介」 ・委員講演　近山　隆委員 「米国 NSF SDP の目指すもの」
第 3 回 平成 14 年 11 月 12 日	・委員講演　関口智嗣委員 「グリッドよどこへ行く」 ・委員講演　久門耕一委員 「高性能 CPU と OS との関係（Linux カーネルのチューニング）」

ハイエンドコンピューティング技術に関する調査研究Ⅳ

© 平成 15 年 3 月発行

発行所　財団法人　日本情報処理開発協会
先端情報技術研究所
東京都港区芝 2 丁目 3 番 3 号
芝二丁目大門ビル 4 階
TEL(03)3456-2511