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Ⅱ

―　高性能プラットフォーム技術を中心とする　―

平成１３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　教授
委員	秋山　　泰	電子技術総合研究所 生命情報科学ラボ　主任研究官
委員	天野　英晴	慶應義塾大学 理工学部 情報工学科　助教授
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 コンピュータシステム研究部　主管研究員
委員	妹尾　義樹	日本電気(株) 情報通信メディア研究本部　研究マネージャー
委員	関口　智嗣	電子技術総合研究所 情報アーキテクチャ部　主任研究官
委員	田中　義一	(株)日立製作所 中央研究所 エンタープライズシステム研究部　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	新部　　裕	電子技術総合研究所 情報アーキテクチャ部　主任研究官
委員	福井　義成	(株)東芝 IS センター エンジニアリングシステム部 主査
委員	古市　昌一	三菱電機(株)　情報技術総合研究所 電子システム部　主席研究員
委員	横川　三津夫	日本原子力研究所 地球シミュレータ開発特別チーム　副主任研究員
幹事	小林　茂	(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
幹事	宮田　哲治	(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
幹事	若杉　康仁	(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　Ⅱ
－ 高性能プラットフォーム技術を中心にして －

目　次

第１章　まえがき

1.1　活動方針と調査の考え方

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ワーキンググループ」、略して HECC（High End Computing and Computation）WG の議論に基づき、各委員の報告をまとめたものである。このワーキンググループは、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ており、その委員は大学、メーカ等の若手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13 名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ト、アルゴリズム研究を含む基礎研究、ソフトウエ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、調査や議論を行うこととしている。

本年度は、HECC ワーキンググループとしては 2 年目の調査となる。昨年度は、議論の手がかりとして、ハイエンド・コンピューティングの分野で先頭を走っている米国の研究開発計画を参考にして、議論をすすめた。そして、そのための具体的な材料として、米国連邦政府のコンピューティング・情報・通信委員会（CIC）がまとめた通称 Blue Book と呼ばれているドキュメントを参考資料とした（平成 11 年度の報告書を参照）。昨年度の報告書では、この Blue Book で指摘されている技術項目について、市場動向などを視野に入れて見たときに市場にインパクトを与えそうな分野やテーマについて、それらの技術的内容および成果の調査・評価と我が国の技術との格差などについて各委員を中心に議論し、各委員の HECC に関する見解をまとめた。

今年度の調査において、本ワーキンググループでは、従来同様に 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・検討した。そのために、各分野の専門家の最新知識を集積して研究開発のリーディングエッジを浮かび上がらせるとともに、今後注力すべき技術分野の検討に必要な元データとしての利用を可能にするように努めた。特に、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、（米国をはじめとする）先端研究とわが国との格差の調査・評価を行うことを主眼にした。したがって、昨年度の調査や報告で目指した、網羅的なマップの完成よりも、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。たとえば、プ

ラットフォーム技術研究の代表的なものは、計算性能の追求である。計算速度や記憶容量等の劇的な向上は、単に処理速度を速めるにとどまらず、コンテンツやユーザインタフェースに質的な変化をもたらし、情報技術の社会的・経済的な影響力が強い。しかし、それ以外にもコンパイラ技術、並列処理技術やグローバルコンピューティング技術など、新たな基盤技術が含む可能性という意味で、いずれも同様の重要性を備えており、次時代をなす情報技術の一次近似予測のためには、これらの動向を広く把握する必要がある。

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、情報処理分野において調査が必要な分野についてかなりの部分を網羅しているが、専門的な分野において、把握しきれない部分があるため、これを外部の講師を招いてヒアリングを行うことにした。今年度は、次世代の I/O アーキテクチャとして注目されている InfiniBand に関して、富士通研究所の岸本光弘氏に、またグローバルコンピューティングにおけるセキュリティ技術に関して、奈良先端大の門林雄基助教授に、講演をお願いした。この講演を基に、岸本氏と門林先生には講演の概要を、本報告書内にまとめていただいた。ここで、あらためて感謝したい。

情報処理の分野の研究開発は、ハードウェアからソフトウェアおよびアプリケーションまで、その最先端技術をすばやく取り入れ、新しいシステムの開発や規格の提案に生かすことがますます重要になってきている。そのためには、より幅広く最先端の情報処理技術や情報処理に関連した技術を的確に捉えることも必要である。本ワーキンググループの委員は、情報処理の各分野において実際に最先端の研究開発に従事している方々であるので、この点は各委員の経験に基づきながら、委員個人の感性に基づいてどのような題材を選択するかについて判断していただいた。本報告書が、わが国の情報処理研究開発に向けた技術開発や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

各委員及び講師による調査報告の概要を以下に示す（本文は第 3 章に記載）。

1.2.1　ハードウェア／アーキテクチャ関連

（１）実用化が進むリコンフィギャブルコンピューティング（天野委員）

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リコンフィギャブルコンピューティングは、高速性と柔軟性を併せ持つシステムとして、90 年代の初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など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ことは少なかった。しかし、最近 CPU と FPGA との混載が可能になり、通信処理というキラーアプリケーションを開拓したことにより、一気に実用化が進んでいる。日本でも新しい構成のチップやシステムが次々に提案され注目されつつある。本稿では、最近の情勢を分析すると共に、日本で現在やるべき研究、開発の方向性を探る。

（２）次世代 I/O アーキテクチャ（InfiniBand）の概要（岸本講師）

InfiniBand は新しい I/O アーキテクチャとして、1999 年夏から検討が始まり、2000 年 10 月に第 1 版が公開になっている。本稿では、次世代の I/O アーキテクチャである InfiniBand の概要と狙いを説明する。

サーバコンピュータに要求される I/O 性能は、年々着実に上昇している。そして、I/O をコンピュータにつなぎ込むところは、I/O バスが受け持っている。これまで ISA、PCI 等がサーバで使われているが、I/O に要求されるバンド幅は年代と共に急速に増大しており、66MHz x 64bit の PCI をもってしても不足し始めている。それを越える仕様として、InfiniBand が提案されている。

InfiniBand は 1 本の線の周波数が 2.5GHz で、その線が 1 本のものの他に、4 本、12 本束ねたもの（バンド幅にして 500MB、2GB、6GB）が今回の仕様に入っている。

InfiniBand の主要な特徴は、ムーアの法則以上のバンド幅向上、応答時間の短縮と割り込み回数の削減、コモディティ化による低価格化、各種ネットワークの一本化、数千ノードを接続できるスケーラビリティ、高機能（メッセージベース）通信機能、高信頼／高可用性の実現である。

1.2.2　グローバルコンピューティング関連

（１）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（関口委員）

高速なネットワークの発展はこれまでWebというHTML文書へのアクセス手段を提供してきたが、どちらかというと静的に与えられたデータに対するアクセス技術であった。これに対して、Grid はネットワーク上の情報資源に対する高性能かつ柔軟なアクセスを目指し、分散した情報資源を統一的に扱うための手段である。

Grid は世界的に大きな技術潮流となっており、我が国も現時点での先進性とこれまでのポテンシャルを活かして大いに貢献することで、世界標準構築への参画を積極的に推進する必要がある。現時点で我々が持ちうるハイエンドコンピューティングの技術シーズとしてはGridとクラスタであることは疑いもない。また、こうした技術において先行してきた強みを活かし、国内のコンピュータベンダーを巻き込んだプロジェクトとして欧米に対抗する道筋を見つけだしていくことが必要である。

本稿では、Global Grid Forum を通じたGrid技術に関する世界戦略と、次のGridとその応用技術であるブロードバンドコンピューティングによるData Farmingについて概略を述べる。

（２）Gridおけるセキュリティ技術（門林講師）

ここではGridという大きな概念の一実現であるGlobusについて、そのセキュリティアーキテクチャを概観し、その利点と欠点について考察する。むろん、ここでの考察は断片的なものでしかない。包括的なセキュリティアーキテクチャに関する考察や、ソフトウェアの実装にまで踏み込んだセキュリティ検査といったことは近い将来において取り組むべき課題として残されているということに注意されたい。

つぎに、Grid技術の標準化団体であるGrid ForumにおけるGridのセキュリティアーキテクチャ標準化について、潜在的な問題点の有無を点検する。最後に、Gridにおけるセキュリティ研究の方向性について議論をおこなう。

1.2.3　プログラミング／コンパイラ関連

（１）変貌するプログラム言語の位置づけ（近山委員）

高性能計算・通信の実現には、ハードウェアとソフトウェアの両面が重要であることは言うまでもない。ソフトウェアの記述には通常なんらかのプログラム言語を用いるわけだが、解くべき問題とハードウェアアーキテクチャの両者の複雑化や、コンパイル技術の進展に伴い、プログラム言語の表面上の記述と、その記述内容の計算機による実現との間の乖離が拡大してきている。このため、従来常識とされてきた、手続き型言語は

計算機による計算手順を記述し、宣言型言語は計算機とは独立に問題仕様を記述する、といった言語族についての認識は適切性を失いつつある。

本稿では近年のプログラム言語技術と計算機アーキテクチャ技術により、プログラム言語の位置づけがどのように変化してきたか、今後さらにどのような変化が予測されるかについて述べる。

（２）ハードウェアとソフトウェアの協調（笠原委員）

ここでは、実効性能、価格性能比及び使いやすさに優れた次世代マイクロプロセッサ（例えばシングルチップ・マルチプロセッサ）から HPC（High Performance Computer）に至るマルチプロセッサシステムの開発では、マルチグレイン並列化コンパイラのようなソフトウェアによる先端並列化技術の性能を有効に引き出すアーキテクチャ開発が重要であることを述べる。

（３）分散共有メモリ向け並列プログラミングインタフェースの動向（妹尾委員）

分散共有メモリは、物理的には分散メモリの構成を採用しつつ、この上で共有メモリのインタフェースを論理的に提供するシステムであり、ハードウェアとしてはスケーラブルな実装が可能でかつ OpenMP などの使いやすい共有メモリ並列化インタフェースが提供できる点で注目されている。しかしながら高い性能を達成するためには、データアクセスのローカリティーを抽出する必要があり、データマッピングと計算マッピングをどのように制御するかが課題となっている。本稿では、これらの課題および、現在の商用システム上でどのような解決策が提供されているかについてプログラミングインタフェースを中心に説明する。また、今後さらに高性能を得るための方法について、最近の研究事例を紹介する。

（４）科学計算における C++の状況（田中委員）

C++には、科学計算に魅力的な機能がいくつもある。テンプレートを使った汎化プログラミング、記述性の高い演算子の多重定義、コード再利用のためのオブジェクト指向的機能である。しかし、これらの魅力的機能にもかかわらず性能に大きな問題があるため、科学計算の一般ユーザは、C++を本格的には使用していないように思われる。実際に、C++を使用してプログラムを書くと、 FORTRAN に比べ何十倍も遅くなることはよくあることである。この状況は、命令レベル並列方式によって性能をあげるハイエンドプロセッサでは特にひどくなってきている。

本稿では、大きな抽象化ギャップに対するコンパイラによる最適化の困難さと、これを打破するためのアクティブライブラリ構築の試みを、具体的な例に基づいて紹介する。

（５）GCC（GNU コンパイラコレクション）とフリー（自由）ソフトウェア（新部委員）

フリーソフトウェアのコンパイラ開発プロジェクトとして、GNU コンパイラコレクションについて述べる。最近の状況について、技術的な進展と公共のソフトウェアとしての社会的な側面の両方の点から論じる。

技術的な進展としては、新しい言語のサポート、新しいプロセッサのサポートなどの新機能、およびスケジューラの改善、レジスタアロケーションの改善などの効率の向上について述べる。

社会的な側面としては、開発体制の変容、組織を越えた分散協調の開発体制、自由にアクセス可能なソフトウェアの配布と、書き込み権限の制限とパッチの査読の仕組み、および Steering Committee による意志決定の仕組みについて述べる。また、GCC を含むフリー（自由）ソフトウェアの広がりについても述べる。

1.2.4　シミュレーション関連

（１）並列分散シミュレーション技術とエージェント技術（古市委員）

異機種分散シミュレータの統合を可能とするため、米国防総省が 1995 年に提案していた HLA（High Level Architecture）が、2000 年 9 月に IEEE により標準仕様 IEEE Std. 1516 となった。これにより、訓練用のシミュレータ同士をネットワークで連接してのチーム訓練が容易になるばかりでなく、大規模な演算が必要となるような高機能で高精度なシミュレーションシステムを、従来と比べて効率良く実現することが可能となる。しかし、ハイパフォーマンスコンピューティングの分野に HLA がどのように関わることができ、またどのような応用分野に適用可能かは未知数である。そこで、この分野における HLA の応用の研究例を２例調査したので報告する。

（２）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（中島委員）

ハイエンド・コンピュータのアーキテクチャは進化途上にあり、様々なアイデアが次々と提案されている。このようなアイデアを特に性能面で評価するためには、アーキテクチャレベル（または命令セットレベル）のシミュレータは不可欠のツールである。一方、並列マシンの PE 数増加などシステムの大規模化や、マルチスレッド・プロセッサなどプロセッサの複雑化により、シミュレーションに要する時間は増加の一途をたどっている。したがってシミュレータの高速化技術は緊急の課題であり、様々な提案がなされている。今回の報告では並列マシンをターゲットとするシミュレータを中心に、MINT などの代表的なシミュレータの高速化技術や、最近の研究動向について概観する。

1.2.5　並列処理／アプリケーション関連

（１）最適化と並列計算機（福井委員）

将来の高性能計算機（HECC）では並列計算が必須である。並列計算のやり方には、１つの問題を複数の演算要素で計算を実行する正統な方法と、パラメトック・スタディのように個々の計算では並列化は行わず、異なるパラメータに対する計算を並列（併行？）に行わせる「自明な並列化」がある。

ここでは「自明な並列化」よりは、簡単ではないが、ある目的関数を最大化（最小化）するための最適化問題に並列計算を利用することについて考える。最適化を行う手法として、シンプレックス法のような素朴な方法を用いると並列計算の効果は期待できないが、並列計算の効果が期待できる方法について述べる。

（２）高性能計算（機）雑感（横川委員）

本稿では、大規模科学技術計算のための高性能計算、あるいは高性能計算機について、以下の観点から思うことを述べる。

・ベクトル計算機
・コストパフォーマンス
・計算機科学者と計算科学者とのギャップ
・高性能計算機の開発資金

1.3　その他活動

本 WG は定例の 5 回の活動と、米国主体の海外調査も行っているのでその報告を付属資料に添付した。

・付属資料 1 は福井委員の SC2000 参加報告，今回が 3 回目の参加である。
　ASCI のアプリケーションについて、なかなか得られない情報を聞いてきた。

・付属資料 2 は若杉幹事の SC2000 参加報告、今回が初参加。
　全体的参加報告。個人的な印象、主観が強いので、一つには纏めず個別に報告した。

・付属資料 3 古市委員、小林幹事の米国ハイパフォーマンスコンピューティング技術動向海外調査報告。これは合同で一つに纏めてある。
　米国の主要な研究所 5 カ所を訪問し、最新動向を纏めてある。

・付属資料 5 には定例会議の実績を纏めてある。
　平成 12 年 10 月から平成 13 年 2 月にかけて 5 回実施した。

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

昨年の報告書作成時には米国はニューエコノミーとし、未曾有の経済成長をうたい、クリントン政権は最後の仕事である 2001 年 1 月の FY2001 Economic Outlook でも過去の経済成長ぶりを誇った。

2001 年 1 月 21 日より民主党クリントン大統領から共和党ブッシュ大統領に交代した。8 年に続く民主党の政権とアメリカの好景気の重なり、景気低迷の始まりと共和党ブッシュ政権の始まりは過去の政策の大幅な見直しがあることが予想される。

ただ現時点ではこれの詳細はは分かっていない。ブッシュ大統領による米国大幅減税の影響はあるのか？ 商務省管轄の ATP（Advanced Technical Program）は 1994 年に民主党政権下で発足し、今年共和党のブッシュ政権下で見直しが行われることになった。大統領の技術的諮問機関である PITAC（President's Information Technology Advisory Committee）に関しては 2/11 までの任期を 6/1 まで延ばしている。

米国競争力協議会（Council on Competitiveness）は先端技術に対する投資がまだまだ不十分であるという勧告を 2001 年 2 月ブッシュ大統領に対して行っている。

経済面では、米国の景気は 2000 年 3 月の株価のピーク以降、ハイテック企業の株式の低迷を受け 2001 年 3 月にはダウ平均はでは 1 万ドルを切り、ナスダックにおいてはピークの 3 分の 1 の 1600 ドルを低迷した。これは IT を基調とする経済発展が過大期待から見直しモードに入ったのであろう。

2000 年 11 月に米国ダラスで開催された SC2000 High Performance Networking and Computing コンファレンスでの基調講演では 2014 年にペタフロップスを実現と予測した。この時の構成はかなりの超並列を想定しており HTMT の構想は触れられていなかった。

超伝導関連の研究は多いが、まだ素子レベルのものが多くコンピュータ本体までの実用化には時間が掛かりそうである。

ペタフロップスを追求する、米国の最近の動向はあくまで高性能を追求する動きから Grid に代表されるグローバルコンピューティングさらに既存のハードウェアコンポーネントを利用するワークステーションクラスタに流れはますます加速しているように見える。

TOP500 を模したランキング Top500Cluster も計画されている。

ハードウェアの進歩に比べてそれを活用するソフトウェアの進歩が追いつかない。こ

れは PC から HECC に渡って言えることであろう。2000 年 9 月には PITAC は政府支援の研究成果をオープンソースにすることを勧めている。

翻って日本でも、2001 年はハイテックバブルがはじけて全世界的株価低下を迎えている、米国景気低下の影響を受け株式市場はバブル以来の最安値日経平均 12000 円台を 2001 年 3 月に記録した。中長期的ゼロ金利政策が復活した。

以下、本章ではハイエンドコンピューティングの新しい流れを中心に概観する。主な情報は源は 2000 年 11 月に開催された SC2000 コンファレンス及び通称 Blue Book と呼ばれている大統領予算教書への科学技術に関する補足ドキュメント（Blue Book 2001）及び Web で公開されている情報をもとにした。

Blue Book 2001 に関しては、下記サイトを参照されたい。（過去の分もある。）

英語原本：http://www.itrd.gov/home.html

日本語訳：http://www.icot.or.jp/FTS/Ronbun/Bluebook2001-J.PDF

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 4 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）にまとめられているのでこれを参照下さい。

報告書はいずれも AITEC のホームページ（ http://www.icot.or.jp/ ）から参照可能。

2.2 Blue Book 2001 にみる政府支援の研究開発

2.2.1 構成

今期の Blue Book いつもより遅れて 2000 年 9 月に開示された。昨年の CIC による発行から、組織替えがあり Interagency Working Group on IT R&D（関係省庁間情報技術 R&D に関する WG）からの発行となっている。

この情報から米国の国家支援研究プロジェクトは概観できるのでここで簡単にまとめておく。内容はコンポーネント毎に分けられており、昨年とほぼ同じ体裁を取るが詳細には一部変わっている。

2001 年度の研究は次のプログラムコンポーネントに分類された。

(1) High End Computing and Computation (HECC)

- **High End Computing Infrastructure and Application (HEC I&A)**
  現状で政府が利用するインフラとかアプリケーションを目的に比較的近い将来。
- **High End Computing Research and Development (HEC R&D)**
  ペタフロップスとか量子コンピュータを追い求める将来的な研究がターゲット

(2) Human Computer Interface and Information Management

昨年の Human Centered System がこれに変わった。内容が追加されている。

(3) Large Scale Networking (LSN)

昨年と同じ分類。

(4) Software Design and Productivity

PITAC の勧告を受けて、2001 年度から新規コンポーネントとして追加された。ソフトウエアの需要が開発能力を完全に上回っているのとソフトウエアの設計、維持管理の困難さ等の問題をいかに克服するかが研究される。

(5) High Confidence Software and System

これも PITAC の勧告を反映して、昨年の HES（High Confidence System）にソフトウエアが明示的に追加された。

(6) Social, Economic and Workforce Implication of IT and IT Workforce Development

昨年の ETHR（Education Training and Human Resourse）に当たるもの。範囲が広くなっている。

の計６つの分類である。

本報告書が扱う範囲は、基本は**(1)HECC** の範囲であるが、これにはとらわれず副題でつけたように高性能プラットフォーム技術関連分野としている。

Blue Book2001 のハイエンドコンピューティング部分に対しては内容を整理し一覧できるように付表 4.1,4.2 に纏めたので参照ください。テーマ、管轄省庁、内容等を簡潔に整理した。

以下に簡単に HECC 研究開発の対象分野について概説する。

2.2.2 High End Computing Infrastructure and Application (HEC I&A)

これは、既存の技術で実現可能なハイエンドコンピューティングの実現であり、政府機関の担当する分野で必要とされるもの。総数約 20 の項目が紹介されている。

（1）環境とツールの整備

ハイエンドコンピューティング（並列処理、数値解析、大規模データセット等）環境を簡単に構築できるような、インフラストラクチャやツールキットの整備。

（2）モデリングツール

オブジェクト指向で再利用性に優れたモデリングツール、有限要素ソフトウェア等がある。

（3）アプリケーション

具体的な問題に対応するアプリケーションの研究開発で生物医学、航空宇宙応用化学、量子物理学、気象等の分野を網羅している。

2.2.3 High End Computing Research and Application (HEC R&D)

現在の技術では実現できない部分、研究開発であり、

(1)Tera フロップスを目指す HTMT

(2)Beowulf クラスタの次世代

(3)グローバルコンピューティング

Globus、Legion 等

(4)さらに現在の技術では実現できない次世代コンピューティングの研究開発

量子コンピュータ、DNA ストレージ、超伝導エレクトロニクス等 14 テーマの紹介があり 10 年先を目指す研究に力を入れているのが分かる。

2.2.4 研究施設

HECC の研究は NSF, NASA, DOE, NIH, NOAA, EPA 等の省庁にまたがり行われている。付表 4.3 に研究施設一覧をつけたので参照ください。

NSF 支援の PACI（Partnership for Advance Computational Infrastructure）では各研究施設のコンピュータを接続し、全米規模の Computational Grid が構築され、共同運用が可能になっている。

イリノイ大学(UIUC)が中心の Alliance(National Computational Science Alliance)とサンディエゴ大学（UCSD）が纏める NPACI（National Partnership for Advanced Computational Infrastructure）がある。

2.2.5 HECC の予算規模

2001 年度は 2 月に “A Blueprint for New Beginning”として予算概要が発表されたが詳細の枠組みがでていない。データは多少古いが BlueBook2001 での情報でその規模をみると次のようになる。

2001 年度の予算要求

単位：100 万ドル

関係機関	(HECI&A)	(HEC R&D)	(HCI&IM)	(LSN)a	(SDP)	(HCSS)	(SEW)	合計 b
合計	761.6	291.4	334.7	368.8	160.5	98.3	120.9	2,137

HECC 関連である HEC I&及び HEC R&D は合計で 1053M ドルであり、IT 関連予算の約 50%を占める。2002 年度の詳細予算はまだ公開されていないが、ブッシュ政権になり、見直しが行われると思われる。各省庁毎の詳細は付表 4.4 を参照ください。

2.3 新しい動向

ハイエンドコンピューティングを代表する ASCI プロジェクトおよび今後の新しい流れと思われるものをあげる。

2.3.1 ASCI プロジェクト

現時点での世界最速のコンピュータは TOP500 のサイトではベスト 4 までが ASCI マシンである。付録 4.5 に TOP10 までをあげた。No.1 が本年度納入の White で、さらにそれを上回るマシンの受注がきまった。

（１）ASCI White

IBM が Laurence Livermore National Laboratory に ASCI White を納入、現時点で最速のコンピュータとなった。

スペックは下記

・プロセッサ：　IBM Power Chip3 (8192 個)
・速度：　12.3 Tera Flops
・メモリ容量：　6 Tera Bytes
・ディスク容量：160 Tera Bytes
・設置場所：　DOE の Lawrence Livermore National Laboratory
・目的：　複雑な 3 次元シミュレーションを用い、核兵器の保管の安全性を実際の実験を行うことなく確かめる。

2001 年中の完全インストールが計画されている。

参照サイト：　http://www.llnl.gov/asci/news/white_news.html

（２）ASCI “Q”

さらに DOE は COMPAQ 社にコードネーム ASCI “Q”を発注した。

スペックは次の通り。

・プロセッサ：　COMPAQ　アルファチップ（12,000 個）
・速度：　30 Tera Flops
・メモリ容量：　12 Tera Bytes
・ディスク容量：600 Tera Bytes
・設置場所：　DOE の Los Alamos National Laboratory

これは SC2000 の展示でもモックアップが 2 カ所（COMPAQ 社、Los Alamos National Laboratory）に飾られていた。最終完成は 2002 年予定。さらに将来的にはマシンを追加し、100Tflops を 2004 年を目指している。

（3）ASCI のソフトウェア対応

ハードウェア高速化の華々しい話題に比し、ソフトウェアの情報があまり開示されていない、あるいは進歩があまり無いのであろうか？

超並列でプロセッサが 1 万個の規模になれば効率よいプログラムの開発は並大抵でないであろう。付録資料 5.1 の福井委員の報告によると ASCI の現在の並列処理の効率は 1000 プロセッサで 10%から 12%位。研究者はこのぐらい出ればいい方との感じを持っている。超並列ソフトウェアは今後ますます重要な分野となるであろう。

2.3.2 Cluster と Linux

クラスタシステムへの加速が激しい、特に予算を確保が厳しい研究所では PC を寄せ集めても構築可能なクラスタシステムに魅力を感じているようだ。

ネットワークも Mirinet, Giganet 等が De-Facto 標準化されコストが下がってきている。システム構築が一層容易になってきている。

SC2000 展示でも多くのクラスタが出展され、ベンチャ企業を中心にマーケット拡大を狙っているのが伝わってくる。日本の RWCP が開発したクラスタシステムも SC2000 会場には展示されており頑張っている。

BlueBook2001 での紹介は Beowulf プロジェクトである。ここでの使用 OS は Red Hat 版の Linux であり、すべての開発ライブラリ等はこの上に移行している。

Top500 サイトもこの流れに、クラスタ用のランキングを準備している。

参照サイト：http://clusters.top500.org/

2.3.3 グローバルコンピューティング

HTMT に代表される 1 台で高速化する手法も、かなり困難が伴いコスト的にも見合わないというのが色々見えてきており、こちらは世界のコンピュータをつないで使おうと言う発想からきている。Web がデータアクセスの世界では成功しているように、Computational Grid といって計算機の計算能力資源までも使おうというもの。

昨年の BlueBook でも Globus が紹介されているが、2001BlueBook では Globus のほか Legion、さらに ACCESS DC が別に紹介されており、研究の盛り上がりがみられる。

2000 年 8 月には横浜で Inet200 が開催され、その展示会場の目玉は Grid のインターナショナル版の I-Grid であった。

(1) ACCESS DC (Alliance Center for Collaboration, Education, Science and Software)

ワシントン DC に最新鋭の設備を持つセンターを設置し、民間企業、教育機関、政府系機関の実践的導入教育の機会を提供している。２つのコンポーネントで構成される。

(a)Computational Grid

スーパーノードと呼ばれる高性能並列コンピュータのグループで構成されて

いる。

大学、研究所のコンピュータが接続され大規模並列、並列ベクター、分散共用メモリ、共用メモリ対称型マルチプロセッサ、クラスタワークステーション等が含まれている。

(b)Access Grid

Grid ユーザの大規模な遠隔会議、共同チームセッション、セミナー、講義、個別指導、トレーニング等の機能を提供している。デスクトップ対デスクトップではないグループ対グループのための各種装置を完備する最初のものである。

(2) Legion

オブジェクト指向のメタシステムソフトウェアであり、共同研究の為の共有仮想ワークスペースを提供する。数百万台の接続されたホストのデータ、物理的資源があたかも自分のマシンにあるかのようにアクセスできる。

2.3.4　量子コンピューティング

まだ、汎用の量子コンピュータができるというところまではきていないが、半導体素子の限界が見えているだけに研究は進んでいる。Blue Book でも昨年あたりからかなり紹介されており 2001 版では“量子コンピューティング”、“量子情報とコンピュテーション”、“アンサンブル量子コンピュータの磁気共鳴”等が紹介されている。

ビッグニュースは IBM が 2000 年 8 月に 5 量子ドットによる量子コンピュータの実験レベルに成功したと言う報道であろう。理論でしか無かったのが実験で確かめられたのは大きいと思う。研究の加速が期待できる。

SC2000 展示会では、量子コンピュータの展示はほとんど無かった（パネルを１つみただけ)。まだまだ展示会で見せられるところにはきていない。

2.3.5　ヒューマンゲノム解析

2001 年 2 月には米国セレラ社及び国際ヒューマンゲノム計画チームは人間の遺伝子の塩基配列の解析結果を同時公開した。遺伝子の数は当初予想されていた 10 万個を下回る 3 から 4 万個であることがわかった。この数でできていることは思っていたより遺伝子の仕組みが複雑であると推定される。

具体的な成果データはセレラ社はサイエンス誌 2001 年 2 月 16 日号、国際チームはネイチャ誌 2 月 15 日号で公開された。

これからは遺伝子の本格的解析に向けて拍車がかかっている。

遺伝子解析には現在の最高速スーパーコンピュータでも足りないので DOE の Sandia 研究所と米セレラ社、コンパック社は専用のスーパーコンピュータの開発に向け 2001 年１月共同研究提携をした。目標性能は 100TeraFlops である。

第３章　ハイエンドコンピューティング研究開発の新しい動向

3.1　概要

本章では、平成 12 年度のハイエンド・コンピューティング技術調査ワーキンググループの活動に基づき、委員及び講師の調査報告をまとめた。

今年度活動方針

(1) 調査対象の拡大

今年度の調査において、本ワーキンググループでは、従来同様に 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。

(2) 重要技術分野の抽出

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、(米国をはじめとする) 先端研究とわが国との格差の調査・評価を行うことを主眼とした。したがって、昨年度の調査や報告で目指した、網羅的なマップの完成よりも、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いた。

したがって、今年度の調査は、ハードウェア／アーキテクチャ、グローバルコンピューティング、プログラミング／コンパイラ、シミュレーション、並列処理／アプリケーションなどと多岐に渡っており、ワーキンググループの各委員の専門分野で網羅できない部分は外部講師による講演を実施した。今年度は下記の 2 名の方に外部講師をお願いし、最新状況を講演して頂いた。また、講演内容は本章の報告としてまとめて頂いた。

(1) (株) 富士通研究所　コンピュータシステム研究所
　　岸本 光弘　主管研究員
　　「次世代 I/O アーキテクチャ (InfiniBand) の概要」

(2) 奈良先端科学技術大学院大学　情報科学研究科
　　門林 雄基　助教授
　　「Grid におけるセキュリティ技術」

3.2　ハードウェア／アーキテクチャ関連

3.2.1　実用化が進むリコンフィギャブルコンピューティング

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][3]。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、最近、CPU と FPGA の混載が可能になったことと、通信処理というキラーアプリケーションの開拓に成功したことにより、一気に実用化が進んでいる。

リコンフィギャブルマシンについては、平成 10 年度の報告書[4]で、その概観と将来予測について述べたが、ここでは、主にその後の進展を紹介し、最近の情勢を分析すると共に、未解決の課題と研究の方向性についてまとめる。

2. Reconfigurable System とは

PLD（Programmable Logic Device）とは、74 シリーズなどのディジタル IC があらかじめ機能が定まっているのに対して、ユーザが手元でプログラミングするデバイスであり、1970 年代から小規模の書き換え不能なものが一部の組合せ回路に用いられていた。80 年代に CMOS 技術の導入により、再書き込み可能で、順序回路も含む広い範囲の回路を実装することができるようになり、ディジタル回路の実装用デバイスとしての地位を確立した。さらに、90 年代に入って、大規模な FPGA（Field Progarmmable Gate Array）あるいは CPLD（Complex Programmable Logic Device）が登場するに至って、システムをまるごと実装可能なデバイスとして進化を遂げた。最近は、100MHz 近い周波数で動作する高速性、100 万ゲートに迫る実装密度、内蔵 RAM を持つものものも出現し、ASIC に代わってシステムインテグレーションの主役になりつつある。

このうち、内部の配線データ（Configuration Data）を SRAM に格納するタイプは、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。

Reconfigurable System が実現可能になったのは、数千ゲート以上の規模で 10MHz 以上の動作が可能な SRAM 型 FPGA が安価に利用可能になった 1990 年代に入ってからである。**図 1** に Reconfigurable System の発展の歴史を簡単にまとめる。開発例の登場、学会の立ち上げは、ほとんど 90 年代の前半に行われており、非常に若い分野であることがわかる。また、2000 年代に突入して、急激に新しいシステムの開発例が増加していることがわかる。

図 1 に示すように Reconfigurable System についてのアクティビティは次の 3 種類に分類することができる。

・スタンドアローン型：かなり多数の FPGA を用いて、大規模な問題を処理する。
・コプロセッサ型　　：CPU と密結合し、一部の処理を高速化する。
・新デバイス　　：FPGA 自体に Reconfigurable System に特化した機能を持つデバイスの研究

以下、それぞれの代表的なシステムとアクティビティを紹介する。なお、詳細なサーベイは文献[1]を参照されたい。

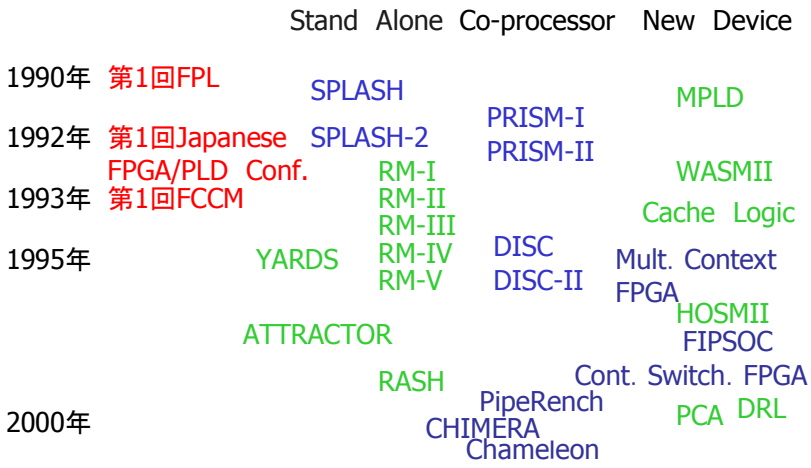


図１　Reconfigurable　System の発達

3. スタンドアローン型の動向

スタンドアローン型 Reconfigurable Machine で最も早い時期に登場したのは、米国計算センターで開発された Splash-1/2 である。Splash は、FPGA の計算セルを直線状およびクロスバを用いて接続したシステムである。各ボード上では Xilinx 社の FPGA である XC4010 が計算用に 16 個（X1--X16）、制御用に 1 個（X0）使用され、これらがクロスバを介して結合されている。また、各 FPGA は 256K×16 ビットのローカルメモリを持っている。計算用の 16 個の FPGA はリニアシストリックアレイを構成して

おり、その結合の幅は 36 ビットである。X0 の FPGA はクロスバの制御にも用いられる。この構成は、流れてきたデータに対して計算を行なうシストリックアルゴリズムあるいは、全体で単一の命令を実行する SIMD（Single Instruction stream and Multiple Data stream）的な動作に特化している。このため、パターンマッチングや画像処理などのアプリケーションでは大幅な性能向上を示している。Splash 1 は、DNA の塩基配列間の距離を求める計算で、当時のスーパーコンピュータ CRAY-2 の 330 倍の性能を記録して大きな注目を集め、Recongigrable System の Flagship 的な役割を果たした。一方で、FPGA 間の結合の自由度が低い点や任意のメモリへのアクセスが困難な点などから、適用分野はある程度限定されている。現在、Annapolis Micro Systems 社より WILDFIRE として商用化されている。

日本でも、神戸大学の沼らが、かなり早い時期にスタンドアローン型の Reconfigurable Machine、RM-I,II,III,IV を次々と開発してフロンティア的な役割を果たした。RM-IV は、FPGA とメモリを組み合わせたモジュールを FPIC と呼ばれるプログラマブルなスイッチで接続した構成を持つ。この構成は、Splash に比べると汎用性が高く、沼らは、このシステムに論理シミュレーションエンジンを搭載し、当時のワークステーションの 10-100 倍の性能を記録した。さらに故障シミュレーション、Wavelet 変換への応用を行った。我々慶應大学でもスタンドアローン型の Reconfigurable Testbed FLEMING 上で、ニューラルネットワークエミュレーション、待ち行列解析、マルコフ解析、電力潮流計算等のアプリケーションの開発を行った。

最近、この型での最大規模の商用マシンは、三菱電機による RASH である。RASH は、**図 2** に示すように、Compact PCI バス上に、Altera 社 FLEX10K シリーズ 8 個、SRAM, PCI interface から成る Reconfigurable Board を最大 6 枚搭載し、1 つの UNIT を構成する。さらに複数 UNIT を Ethernet で接続してコンソールを共有することによって、大型システムの実現が可能である。RASH は、様々な分野に用いることが可能だが、暗号解読のアプリケーションが発表されており、並列探索の利用により、汎用ワークステーションの 100 倍以上の性能を達成している。

後に紹介するコプロセッサ型同様、スタンドアローン型も通信処理に応用されている。NTT で開発された YARDS、ATTRACTER は IP パケットを ATM 上で交換するために、プロトコル変換処理と蓄積、交換を 1 つのシステムで行う。また、RWCP と慶應大学が共同に開発した RHiNET/NI では、ネットワークインタフェースの構成を変化させて、状況に応じて様々な通信プリミティブを実行する。

通信制御、ネットワーク制御等、ある程度専用目的のスタンドアローン型は、それぞれの分野で利用されているが、RASH 等の汎用目的のスタンドアローン型は、商用化はされてはいるものの、大きな成功を収めているとは言い難い状況で、最近元気なのは小規模システムであるコプロセッサ型である。

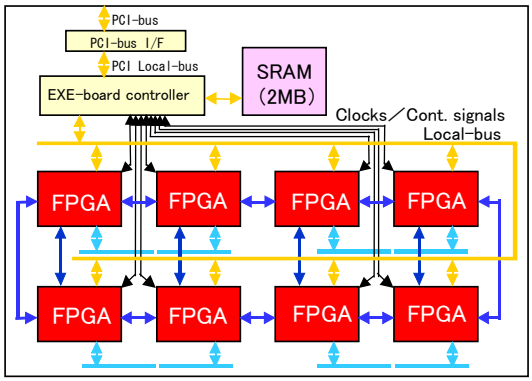


図２　RASH（三菱）の Reconfigurable　Board

4. コプロセッサ型の動向

コプロセッサ型の Reconfigurable Machine は、一般的なプロセッサと密結合し、その機能の一部を高速化するのが目的である。この型の元祖といえるのは Brown 大学によって 90 年代の前半に研究された Plism I/II である。Plism は、CPU と FPGA を共有メモリで接続したシンプルな構成を持ち、プログラムの大部分を占めるループを検出して、その部分を FPGA で実行することにより、高速化を図った。研究の主眼はソフトウェア技術に置かれていた。実際に Plism の単純な構成では CPU と FPGA の転送容量が制限されたため、多様なプログラムへの適用が困難だった。

90 年代の終りから 2000 年代に入って、CPU と FPGA を同一チップ上に混載することが可能になり、様々な接続形態が可能になり、様々な提案がなされた。MIPS コアと Reconfigurable Array を強結合した Garp、汎用 PC のメモリを直接 Configuration 用メモリとして使うことで、動的書き換え機能を利用できる DISC、スーパスカラプロセッサのデータパスを可変構造化した CHIMAERA などが大学の研究レベルで提案された。

この中で最近注目されているのは、Chameleon 社から商用化された Chameleon CS2112[6]である。このチップは、**図 3** に示すように、CPU、PCI インタフェース、DRAM インタフェース、メモリおよび Reconfigurable 部を混載した構成を持つ。この Reconfigurable 部は、**図 4** に示すように、3 つの Tile から構成された Slice を 4 つ持つ粗粒度の FPGA である。各 Tile は 108 の DPU を持つ。この DPU は、**図 5** に示すようにマルチプレクサ、ALU、シフタから構成され、これらのユニット内の構成と接続形態を変化させることによって、積和の並列演算、パイプライン処理が実現できる。構成データは、それぞれのユニットのメモリから 1 クロックでロードされるので、高速な動的変更も可能である。Chameleon は、アプリケーションの対象を通信処理、信号処理に絞っている。この分野は高性能な DSP と競合するが、DSP が専ら演算の高速化に

特化しているのに対して、Chameleon は、Reconfigurable 部を利用することで、プロトコル変換、パターンマッチング等の処理も併せて高速化可能である。演算処理自体も、並列処理とパイプライン処理をうまく利用すれば DSP よりも数倍から数十倍の高速化を果たすことができる。

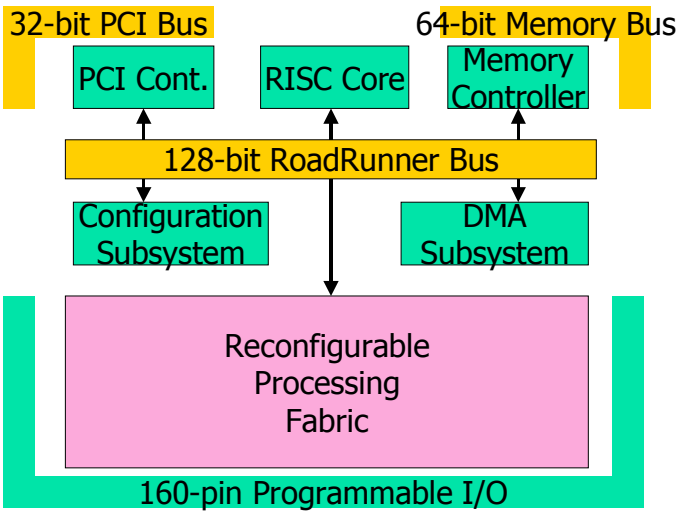
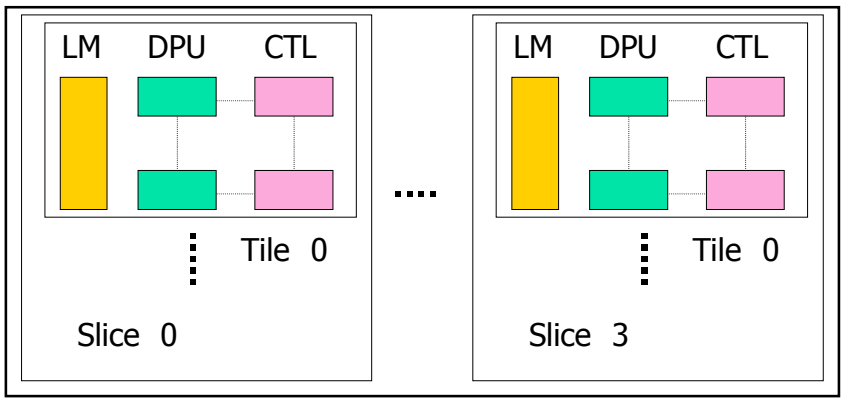


図 3　Chameleon CS2112 の構成



108のDPU(Data Path Unit)が4つのSlice(各3Tile)を構成

1Tile: 9DPU＝32bit ALU X 7 16bit + 16bit乗算器 X 2

図 4　Reconfigurable ProcessingFabric の構造

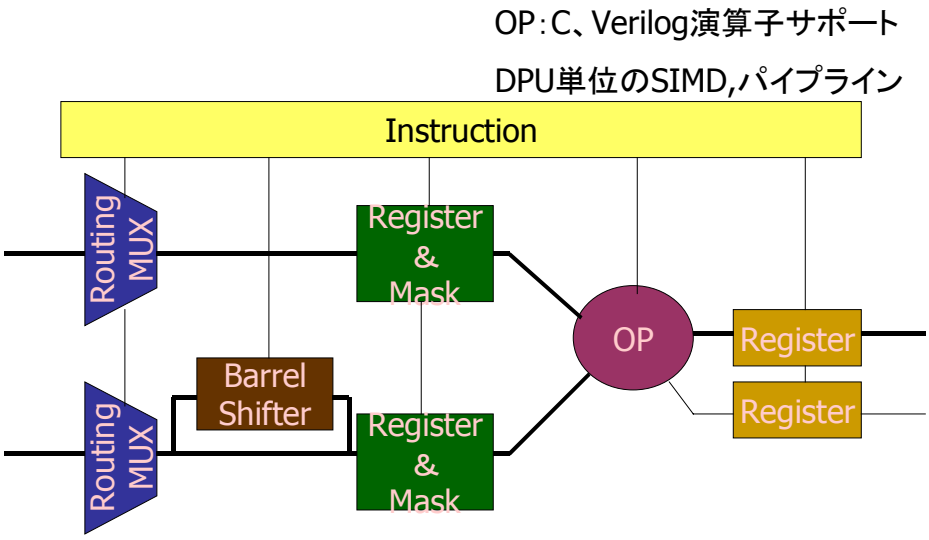


図 5　DPU の構成

5. Reconfigurable Machine の問題点と新しいチップの動向

Reconfigurable Machine の問題点として当初から挙げられていたのは、以下の点である。

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) キラーアプリケーションがない。

d) 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。

e) プログラミングが困難である。また、プログラムのロードに手間と時間がかかる。

2000 年代になって、半導体技術の発達と Reconfigurable Machine の研究の進展と共に、これらの問題は解決の道筋が付けられつつある。

a) 浮動小数点演算の性能の問題

10 倍遅く、10 倍面積を必要とする、というのは、FPGA が汎用 CPU より劣ったプロセスを用いていた 90 年代中ごろまでの話である。現在は、最新のプロセスを用いた汎用 CPU が市場に登場するよりも早く、最新のプロセスを用いた FPGA が市場に登場

する。したがって、この差は縮まりつつある。とはいえ、GHz オーダのクロックを用いる高性能プロセッサや DSP に浮動小数点演算で勝てないのは明らかである。しかし、以下のような逃げ道はある。

a-1: 並列処理と問題にあった構成の利用：パイプライン化された浮動小数点演算器は、かなりのゲート数を要するため、FPGA 上に複数実装するのは、困難であったが、最近の大規模 FPGA を用いれば、ある程度は可能になっている。複数の演算器間をチップ上でアプリケーションに合わせて無駄なく接続し、並列処理することにより性能向上を図る。

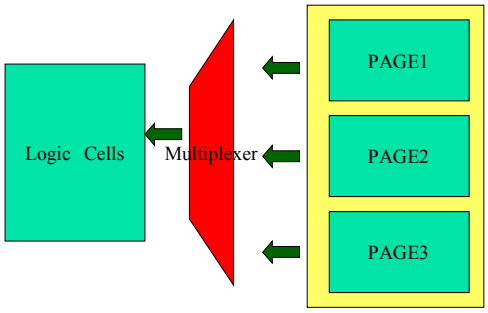
a-2: 粗粒度の FPGA の利用：FPGA の再構成のユニットを ALU 等特殊化された大きなものにすれば、専用プロセッサとの間の性能差は縮まる。このような FPGA を粗粒度 FPGA と呼ぶ。Chameleon の再構成ユニットを始め、広島市立大、RWCP などでも粗粒度 FPGA が開発されている。粗粒度にすることにより、ユニット間の配線を決める CAD も簡単化することができる。一方で、目的が特化されてしまい、例えば ALU でユニットを構成してしまうと、演算以外の処理ができにくくなる。

a-3: 逃げるが勝ち: IEEE 標準にこだわると勝ち目がないが、画像処理や信号処理のアプリケーションでは、IEEE 標準の浮動小数を使う必要がなく、より短い構成の数や固定小数点でも十分な場合も多い。このような場合は、Reconfigurable Machine は目的に応じた長さの数を用いて高速計算を図ることができる。逆に IEEE 標準よりも精度を大きく取る必要がある場合も、Reconfigurable Machine は有利である。

b）メモリとの接続の問題

浮動小数点演算と共に Reconfigurable Machine の苦手とする処理が大規模なデータを扱う処理である。汎用プロセッサは、その誕生時にフォンノイマンが指摘して以来、CPU とメモリ間の通信路が性能のボトルネックになると言われており、逆にこの高速化と洗練には 60 年の歴史がある。その間、キャッシュ、仮想記憶、データ配置最適化、プリフェッチ、当機的アクセス発行など膨大な技術が開発されている。FPGA にメモリをつないだだけの Reconfigurable Machine が勝てるわけはない。

しかし、基本的に Reconfigurable Machine はフォンノイマンボトルネックからは免れているため、演算回路とメモリを接続する大容量の接続路さえ確保できれば、大規模データを並列に扱うことによって汎用 CPU に対抗できる可能性を持つ。このために注目されるのは DRAM 混載型 FPGA である。**図 6** に NEC の開発した DRAM 混載型 FPGA を示す。この構成では、DRAM の Column Buffer を直接 FPGA で扱うメモリとして扱うことができるため、大容量のデータ転送を実現することが可能である。大容量の SRAM 型メモリと、高速データ転送機構を併せ持つ FPGA でも同様の機構は実現可能である。



ページ間の共有
Partial Configuration との組み合わせ
DRL(NEC)は実際に利用可能

図 6　マルチコンテキスト FPGA の基本構造

c) キラーアプリケーションがない。

キラーアプリケーションがない、ないと言われて 10 年間、様々な試みがなされた結果、最近ようやくネットワークプロセッサ、ネットワークインタフェース、ソフトウェア無線等の通信処理、プロトコル処理、信号処理が Reconfigurable Machine のキラーアプリケーションなのではないか、と言われるようになった。このアプリケーションは、次々に到着するデータを処理すること、処理の種類が多様であり、変更の機会が多いこと、演算を伴うが必ずしも IEEE 標準データの必要がないこと、など Reconfigurable Machine に適した点が大きい。基地局用に大規模なシステムが必要な場合はスタンドアローン型が、その他は、プロセッサを混載したコプロセッサ型が主に用いられると考えられる。Chameleon もこの分野をターゲットとしており、NTT でも Proteus 3 など通信処理に特化したチップの開発が行なわれている。しかし DSP、専用システム LSI チップ等ライバルも強力であり、この分野を制覇できるかは疑問である。

d) サイズの問題

FPGA のサイズの問題は、仮想ハードウェアの研究と特殊な FPGA チップの登場により、解決の見通しは付いている。この問題の解決のための最も簡単な方法は、マルチコンテキスト FPGA の利用である。マルチコンテキスト FPGA とは図 6 に示すように、チップ内に複数の配置配線データ用メモリを持ち、これを高速に切替えることで、チップの構成を一気に変えることのできる FPGA である。NEC はマルチコンテキストでかつ部分的な再構成可能なチップである DRL を開発した。このチップは、利用していないデータ用メモリに新たな構成データを外部からロードして、適切な時期に切替えることにより、仮想的に大規模なハードウェアを実現することができる。我々慶應大学は 90 年代の前半から、データ駆動型制御により、仮想ハードウェアを実現する方法を研

究しており、この方法を DRL 上で用いることによって、小規模ながらある程度実用的な問題を解くことのできる仮想ハードウェアシステムを実現した。

より大規模で実用的な仮想ハードウェアシステムの実現のためには、チップ内で大量のデータが格納可能な DRAM 混載型や、チップ外とのデータの高速転送機能を持つ FPGA が必要である。

CMU で開発中の PiPERench は、パイプラインの構成を動的に変更することにより、仮想ハードウェアと同様に小規模な実回路で大規模なハードウェアを実現することができる。さらに、NTT で開発された PCA（Plastic Cell Architecture）は、より柔軟な構成を持っている。このチップは、**図 7** に示すように通信を制御してコマンドを実行する部分と再構成可能な部分を持つ。処理が進んで、あるハードウェアが新しいハードウェアを必要とすると、ちょうどソフトウェアのプロセスが子供のプロセスを Fork するように、非同期メッセージを送ってハードウェアを新たに生成することができる。

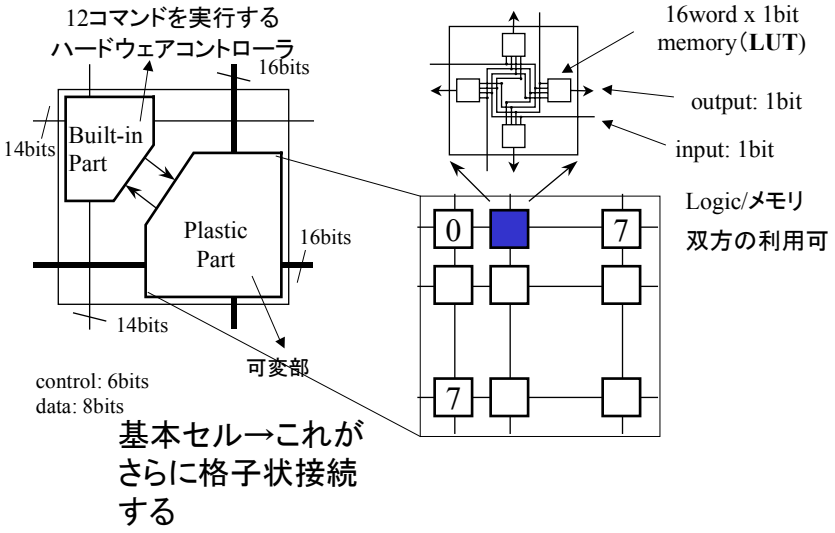


図 7　PCA(Plastic Cell Architecture：NTT)

e) プログラミングが困難である。また、プログラムのロードに手間と時間がかかる。

Reconfigurable Machine のプログラミングは、最も性能を出すためには、ユーザが VHDL や Verilog-HDL などのハードウェア記述言語でハードウェア構成をじかに記述しなければならない。これは流石にしんどい。われわれの慶應大学のプロジェクトでは、DFC などのデータフロー言語からグラフを生成して、自動的に分割する方法を開発した。しかし、データフロー言語は記述上の制約が多く、決してプログラム開発がしやすいとは言えない。この問題の解決は、System-C, Spec-C などの CAD 用高位記述言語の発達を待つよりない。幸いにしてこの分野の研究は最近活発であり、現状よりは、は

るかにましなプログラム環境が実現されると思われる。

また、Reconfigurable Machine は、開発したプログラムを実行するには Reconfigurable 部の配置、配線が必須なので、場合によっては、数時間を要する処理になってしまう。これもプログラム開発時のシミュレーション環境等 CAD の整備によりある程度解決は可能である。ただ、Reconfigurable Machine は、コンパイル時間という点では普通の CPU には絶対にかなわないので、ユーザが非常に頻繁にプログラムをし直さなければならない分野には向いていない。しかし、大部分のユーザは、実際にはそうそう頻繁にプログラムを開発したりしないので、この点はさほど問題にはならないのではないかと思う。

6. おわりに

Reconfigurable Machine の現状を紹介し、問題点と解決の道筋を示した。このように Reconfigurable Machine をめぐる情勢は 1998 年に報告した時点に比べはるかに明るくなっていると言える。また、**図 1** などを見ると、日本の大学、企業が特に新しいチップの開発に関してかなり健闘していることがわかる。しかし、問題点も抱えている。

a) 配置配線用の CAD が弱い。

DRL など日本企業の開発するチップの最大の弱点は配置配線用の CAD である。新しいチップを開発しても利用可能な CAD がなければ利用者は非常に制限される。これが日本企業が優れた機能を持つ新しい FPGA チップを開発しても一般的に商用化できない最大の原因である。CAD 開発を支援する何らかの枠組が必要であろう。

b) Chameleon タイプは成功の確率が最も高いが、今から乗り出しても既に遅い。

Core CPU、PCI, DRAM 等の標準インタフェースと粗粒度 FPGA を組み合わせる方式は現状で成功の可能性が最も高い。しかし、この分野には Reconfigurable Machine 以外のライバルも多く、また Chameleon がある程度の成功を収めれば米国ベンチャーがよってたかって開発に乗り出す可能性がある。今から開発を始めるには、ソフトウェア無線等やや違った所を狙い、アナログ混載等の独自技術が必要になる。

c) DRAM 技術、大規模 SRAM 混載技術で勝負

先に述べたように、DRAM の混載、あるいは思い切って大きな SRAM の混載と、配置配線データの外部との高速な転送能力を持ったチップは有望である。また、この分野のデバイスは比較的、日本が強いので希望が持てる。

d) プラスαを

Reconfigrable Machine は性能と柔軟性だけで勝負すると、どうしても専用チップと

の競争がしんどい。ここで、もう一つプラスαがあれば競争で優位に立つのが楽になる。

ｉ）同一性能を達成するのに消費電力がうんと小さければメリットになる。現状でもGHzオーダのクロックで動く高性能プロセッサと同程度の性能が数十MHzで実現可能な場合がある（というより、FPGAの動作速度が数十MHzを越えられないのだが）。消費電力は周波数に比例するため、本質的にReconfigurable Systemは優位に立てる可能性はあると思う。ただし、現在のFPGAは省電力という点があまり考慮されていないので、問題は残る。また、仮想ハードウェアは、実際に動いている部分が小さいので、消費電力という点でも注目される。

ⅱ）マルチコンテキストFPGAやPCAなど、新しい構成を持つFPGAは、ハードウェアのコンテキストを切替えることができる点で、いわばOSの機能の一部を持っているのと同様である。これをさらに改良することで、専用チップの開発に比べ、はるかにスマートに信頼性の高いシステムの設計を短期間で可能にすることが考えられる。すなわち、複雑なシステムを簡単に実現することができるという点で、専用チップに優位に立てる可能性がある。これにはCADとチップの両方を視野に入れた研究が必要になる。

参考文献

今、最も完全なサーベイは文献[1]であり、ここに登場したシステムは全てサーベイされているが入手性が悪いので、web にて公開を依頼中。入門用には文献[2]があり、国内のアクティビティは文献[3]がある。

[1] 柴田裕一郎: "データ駆動型仮想ハードウェアに関する研究",慶應義塾大学理工学博士論文 2001年（http://www.am.ics.keio.ac.jp/ にて公開予定）.

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

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

[4] 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）, pp.35-43,（平成10年3月）.

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

3.2.2　次世代 I/O アーキテクチャ（InfiniBand）の概要

岸本 光弘講師

InfiniBand は新しい I/O アーキテクチャとして、1999 年夏から検討が始まり、2000 年 10 月に第 1 版が公開になっている。米国のメーカが中心となって仕様策定しているが、日本からも何人かが標準化作業に参加している。ここでは、次世代の I/O アーキテクチャである InfiniBand の概要と狙いを説明する。

1. InfiniBand の概要

1.1　性能概要

サーバコンピュータに要求される I/O 性能は、年々着実に上昇している（図 1 参照）。そして、I/O をコンピュータにつなぎ込むところは、I/O バスが受け持っている。これまで ISA、PCI 等がサーバで使われているが、I/O に要求されるバンド幅は年代と共に急速に増大しており、66MHz x 64bit の PCI をもってしても不足し始めている。それを越える仕様として、InfiniBand が提案されている。

InfiniBand は 1 本の線の周波数が 2.5GHz で、その線が 1 本のものの他に、4 本、12 本束ねたもの(バンド幅にして 500MB、2GB、6GB)が今回の仕様に入っている。

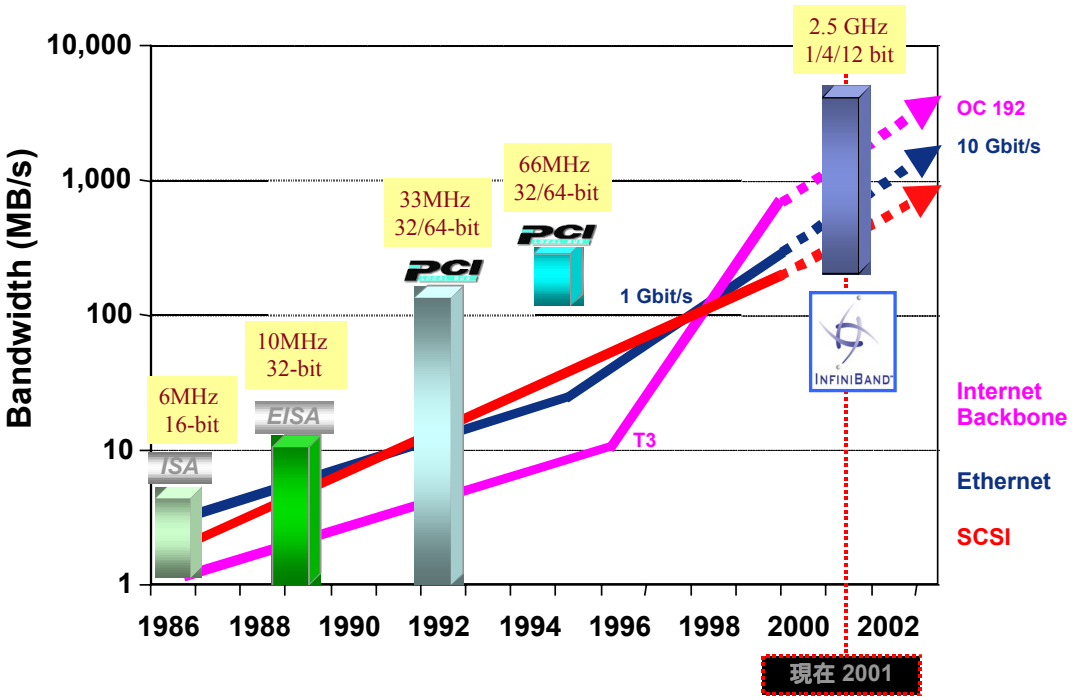


図 1　サーバに要求される I/O 性能の上昇

1.2　技術の特徴

InfiniBand は次のような主要な特徴を備えている。

- ムーアの法則以上のバンド幅向上
- 応答時間の短縮と割り込み回数の削減
- コモディティ化による低価格化
- 各種ネットワークの一本化
- 数千ノードを接続できるスケーラビリティ
- 高機能（メッセージベース）通信機能
- 高信頼／高可用性の実現

1.3　アーキテクチャ概要

従来は CPU とメモリがメモリコントローラにつながり、そこからブリッジを介して PCI につながっていた。このようなバス型のアーキテクチャだと、接続のケーブル長さやバンド幅、あるいは故障時のアイソレーションなど、色々な制約がある。そこで、InfiniBand はバスをやめてメインフレーム式のチャネルにしようと考えた(図 2 参照)。

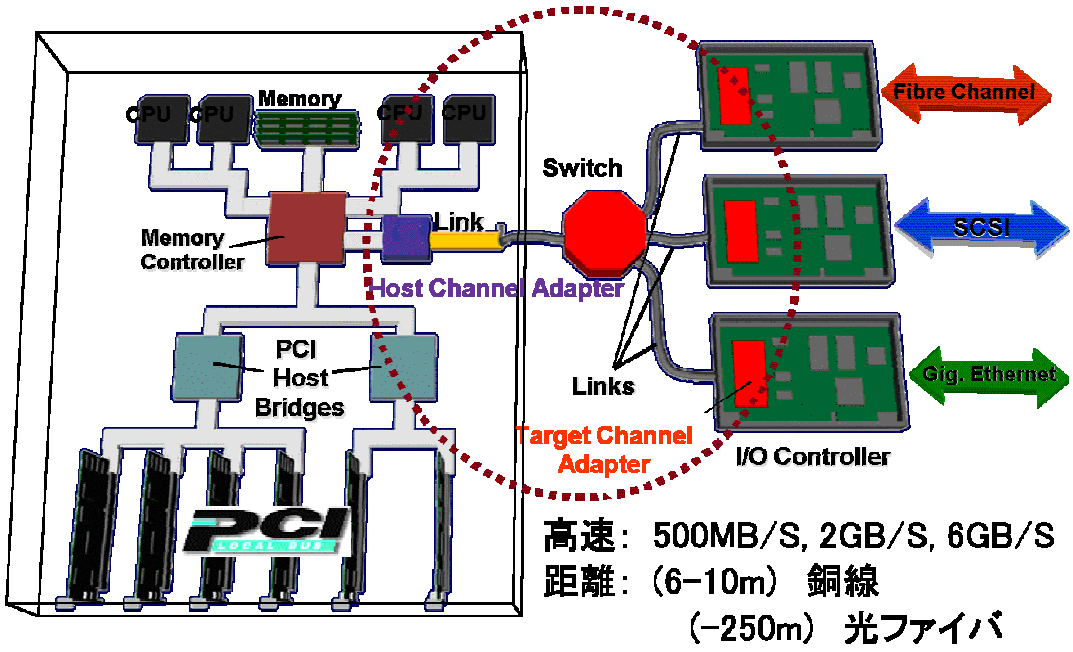


図 2　InfiniBand ハードウェア構成

メモリコントローラにはホスト側チャネルアダプタ（HCA）を、一方の I/O コントローラにはターゲットチャネルアダプタ（TCA）をつけ、両者をリンクでつなぐ。途中にスイッチを入れることによって、1 本の線で多数の I/O コントローラ接続を可能にする。かつ、リンクであるから、I/O だけでなくホスト間のインターコネクトとしても使用可能となる（図 3 参照）。

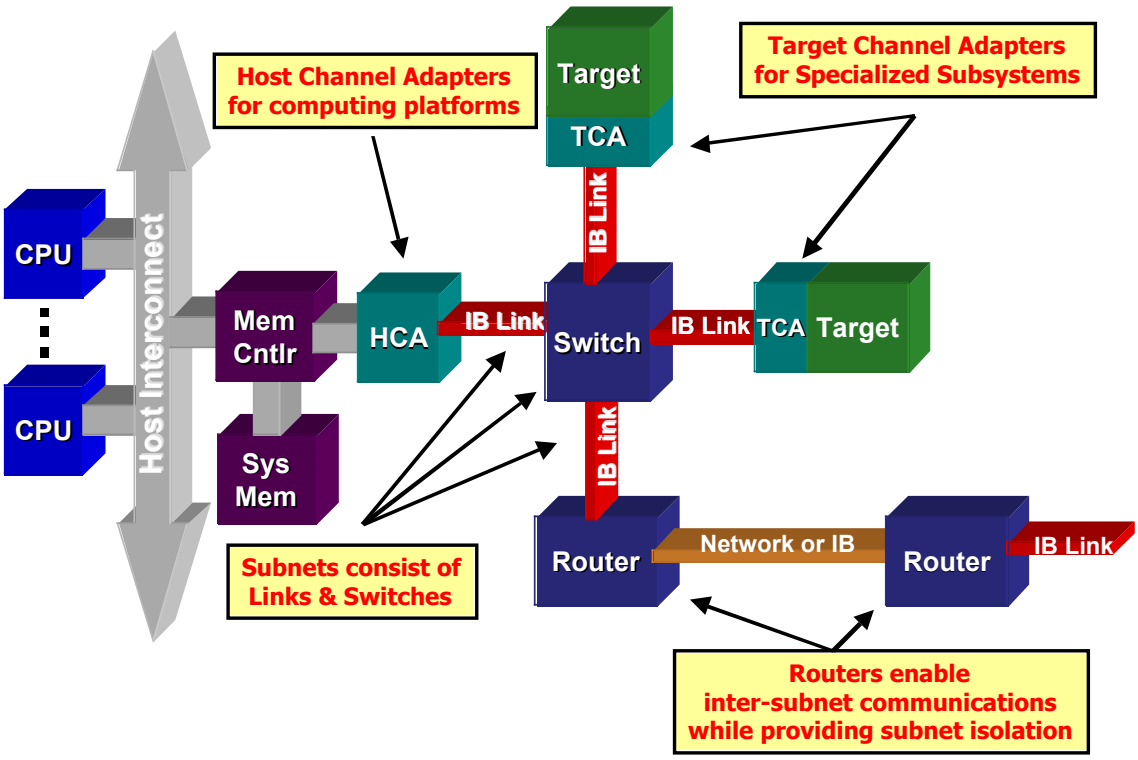


図 3　InfiniBand ブロック図

InfiniBand のネットワークはリンクとスイッチから構成される。インターネットと同様に、最小単位であるサブネットを多数つないでスケーラビリティをかせぐ。

最小のサブネットはスイッチを中心に接続された構成である。HCA は高機能を想定しているが、TCA は安価な物から高価な物まで幅があり、サブセット機能も認める。将来はルータ接続によりサブネットを越えた構成も可能とする。

1.4　電気的インタフェース

銅線には x1/x4/x12 のものがある。各通信路は送信/受信用に別々のケーブルを使う。ディファレンシャルシグナルで 8B/10B コーディングして送るので、バイト単位でいうと 2.5MHz で 250MB の転送速度となり、それが登り下り 2 本で 500MB と称している。x4/x12 ではそれぞれの線がバイト毎にシリアルに送るので、スキュー（電圧遷移タイ

ミングの変動）等に強くなっている。

x16 でなく x12 となった理由は、光コネクタに x12 で良いものがあるから、それに合わせたということである。

1.5　フォームファクタ

実装ではバックパネルに InfiniBand をはわせ、そこにコントローラを差し込む形が多い。アダプタのフォームファクタには、

{Single/Double-wide} x {Single/Double-height}

の組み合わせで、計 4 種が定義されている（**図 4** 参照）。

冗長性を考えた I/O コネクタの場合は、Double-height のものが使われるだろう。コネクタは標準のものが 銅線/光ファイバー、x1/x4/x12 用に各々決まっている。

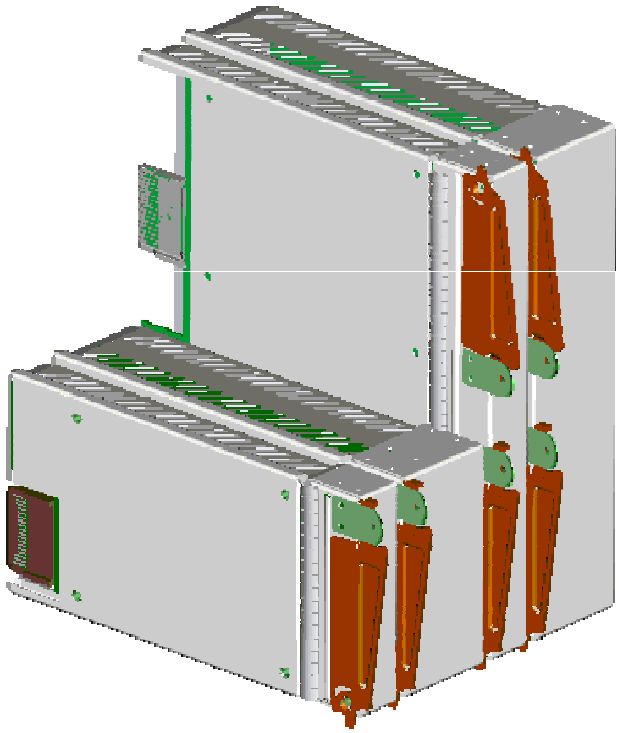


図 4　４種類のフォームファクタ

2.　InfiniBand の狙い

2.1　コンピュータルームのインターコネクト用途

InfiniBand は、I/O バスの後継として、まずコンピュータルームのインターコネクトを狙う。

- ケーブルの種類と数を減らす。
- サーバ内部、およびサーバ間のインターコネクトとして使う。
- LAN は狙わず、300m 以下に収まるネットワークを対象とする。
  - 銅線では 6m-10m/1 本、光では 250m/1 本が規格になっている。
    それをルータでつないで 300m まで。
- スイッチは単純化し、安価に作れるようになっている。
  - 色々なデータが流れるため、大量のディスクデータやネットワークの少量データまで、サービスレベル変更をハードウェアで実現する規格になっている。
- I/O だけでなく、アプリ同士のプロセス間通信にも使える。
- 24 時間、週 7 日使える RAS 機能

2.2　メリット、アドバンテージ

InfiniBand には以下のメリット、アドバンテージがある。

<Easy clustering>

ホスト間のネットワーク、アプリケーション間の IPC、ストレージ I/O、ネットワーク I/O などをすべて一本の InfiniBand でつなげるようにする。また、アップグレードや拡張を単純にする。

<Scalability>

サブネットの中で数千、そのサブネットをさらにつないで大きな InfiniBand ネットワークを組めるようにする。

サブネット内のスイッチは単純･安価なものにし、コスト効果比を高めて普及を図る。x1/x4/x12 のバリエーションは、将来のスケーラビリティも狙う。既に幾つかの会社はプロトタイプ開発をしており、来年後半には x1（500MB）の製品サンプルが、2002 年には x4（2GB）が出てくると思われる。

<Performance>

通信路が太くなるだけでなく、サーバ CPU を通信から開放することによってアプリケーションに集中させられる。

<RAS>

パス（ノード間経路）の冗長化:複数の経路を張れる。さらにサブネット自身も 2 重化することによって、single point of failure の無い経路をサーバ、I/O 間に張ることができる。

イン・バンドの管理が可能であり、イーサ、シリアル等の別ルートが不要である。エラーの管理は、物理レイヤからソフトウェアレイヤまで、きれいに層構造となっている。

<Flexibility>

アーキテクチャ的に任意のトポロジーが可能である。データ転送のサイズは、ユーザアプリレベルでは 2GB まで可能となっているが、ハードウェアレベルでは、安価なスイッチや HCA を使う場合の 256B から最高 4KB まで、5 段階の転送パケットサイズが使い分けられる。

色々な機器を組み合わせると、リンクの幅や転送単位などがばらばらになってしまうが、これを最初に自動的にネゴシエーションする仕様が物理的レベルで定められている。4 種類のフォームファクタ、規格化されたコネクタも、様々な構成の実現に役立つ。

現在の典型的な大規模データセンタのシステムでは、何台かのサーバの後ろに PCI カードスロットをたくさん用意して、ファイバーチャネルやクラスタリング用のインターコネクトのカード、ギガビットイーサなどを挿す。その結果、非常に煩雑な構造となり、接続の種別毎の管理も必要となる。InfiniBand では CPU 以外を I/O として分離し、すっきりした構造となる。

現在のサーバでは CPU と I/O コントローラが同じボードに載っているが、これはユーザ、メーカ、双方にとって好ましくない。なぜならユーザにとっては、CPU、I/O いずれを増設したい場合も余分なものを購入せねばならず、メーカにとっては、CPU を置くための特性の高いボードに PCI の安価な口を並べる形となっている。分離によってこれが解決できる。

InfiniBand からファイバーやギガビットにつなぐためのコントローラはもちろん必要だが、ファイバーチャネルストレージやギガビットスイッチも、従来通りつなげる。RAS、スケーラビリティも、従来よりも格段に改善される。

3. スケジュール

3.1 仕様

昨年 2000 年の 10 月に Spec1.3 が出た。次の 2 冊がある。

Vol1:（ソフトウェアから見た）ハードウェア仕様
Vol2:コネクタの形状、電気特性など

2001 年末から 2002 年初めに 1.0 のアネックス版仕様（補遺）ができ、2.0 は 2 年後くらいになるというのが、アーキテクチャチームのラフな合意事項となっている（図 5 参照）。

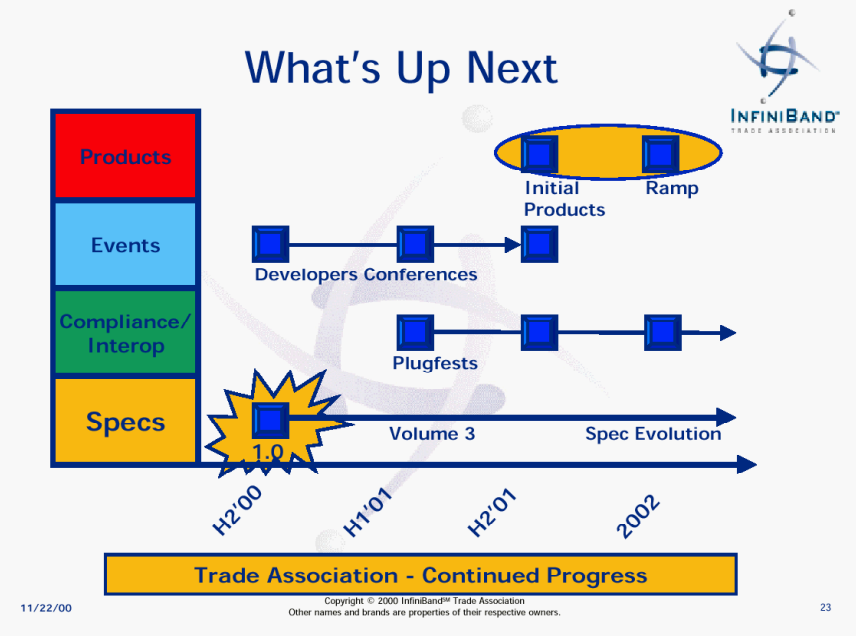


図 5　今後の予定スケジュール

3.2　製品開発

40 社以上が製品開発開始している。さらに I/O ベンダーを巻き込むために、デベロッパーズカンファレンスを半年に一度開催することにしている（**図 5** 参照）。

インターオペラビリティ確保のために、Plugfests も定期的に実施する。2001 年の後半に最初の製品ができ、2002 年には大量に登場するものと期待している（**図 5** 参照）。

4.　機構

4.1　アーキテクチャレイヤ

OSI の 7 階層で言えば、トランスポートレイヤまでをハードウェアでサポートする。実際のソフトウェアはトランスポートレイヤを直にたたくことになる。

「トランザクション」は、ディスクとのデータ I/O を行なう。このとき「トランスポートレイヤ」では、データの要求コマンドに対して、実際のデータメッセージと、完了コードが送られるという手順になる（**図 6** 参照）。

「ネットワークレイヤ」では、サブネットをまたぐパケットハンドリングを行なう。特に新しいものを作るのではなく、IPv6 のルーティングをそのまま使う。このパケットを利用するのが「ルータ」であり、その仕様が定義されているが、記述が曖昧なので本当に 1、2 年中に出てくるのか心配もある。

サブネット内でパケットを送るのは「リンクレイヤ」のスイッチの仕事になる。その下に MAC レイヤがある。

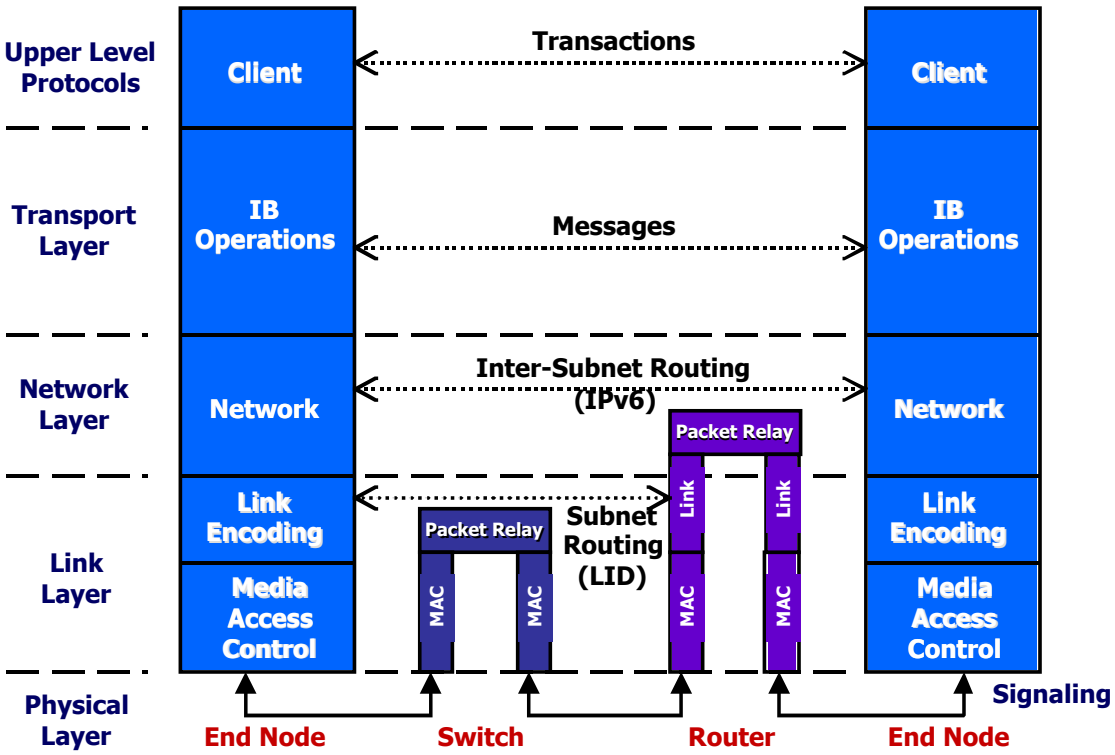


図 6　アーキテクチャ階層

4.2　パケット転送

ディスク I/O（トランザクション）のデータは複数のメッセージに分割され、メッセージはさらにパケットに分割される（図 7 参照）。メッセージは最大で 2GB まで可能である。これをハードウェアが自動的に適当な転送単位のパケットに分割し、パケットの順序と、欠落時の再送を保証しながら転送する。

転送のメカニズムは(昔、Intel が言った virtual interface architecture を拡張した)、メッセージ交換による。

Queue-Pair（QP）が最小の転送実行の単位であり、HCA は複数の QP をハードウェアで実装している。そのオーダーは最小では 256、最大で 32K くらいである。QP は、send/receive-queue のペアである。ソフトウェアは QP に send/receive を指示し、これを受けてハードウェアが実際のデータ転送を行なう。複数の QP それぞれにサービスレベルを設定できるので、たとえばディスク I/O の優先度を落とし、ネットワーク I/O の優先度を上げるなどが可能である。

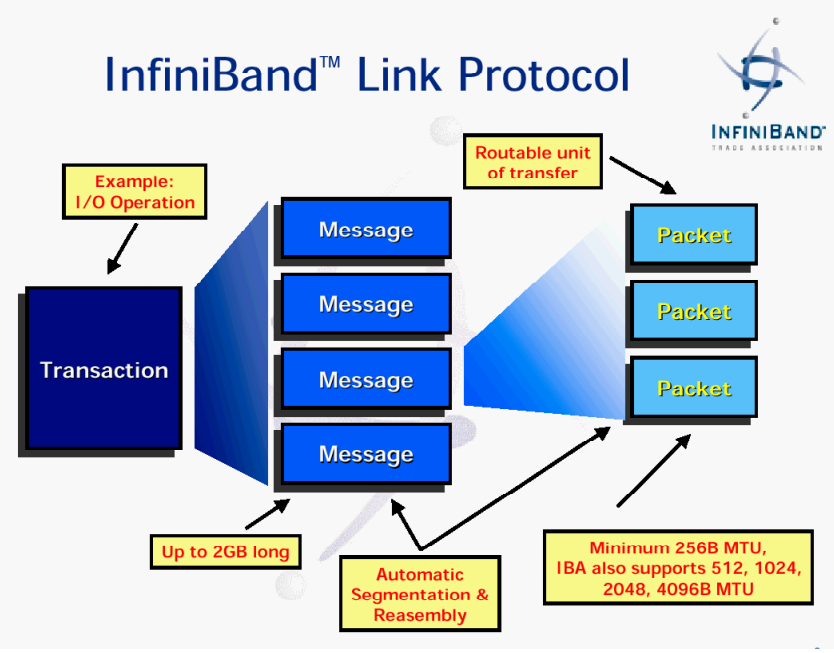


図７　リンクプロトコル

ホストと I/O、あるいはホスト間の接続を規定した「パーティション」をハードウェアレベルで定義し、パケットを色分けすることによって、パケット交換を物理的にパーティショニングできる。

4.3　通信サービスの種類

トランスポートサービスとして 5 種類（最後のものは特殊なので 4 種類と言ってもよいが）用意されている。

・Reliable Connection
　queue 同士の接続に基づく。
　TCP と同様。パケットの順序と送達を保証する。
・Reliable Datagram
　「信頼性のある UDP」。InfiniBand に非常に特徴的であり、自分の QP とリモートの QP との間で、N 対 N のデータ転送ができる。
・Unreliable Connection
　queue 同士の接続に基づく。パケット欠落が有り得る。
・Unreliable Datagram
　UDP と同じ。
・Raw Datagram
　InfiniBand のサービス提供は無く、IPv6 やイーサのパケットをそのまま送る。

また、これとは別に転送機能が定義されている。

・Send

自分のデータバッファは指定するが、相手側のバッファは指定しない。

・RDMA Read&Write

リモート DMA。発行側が相手側のバッファも指定する。

・Atomic

fetch&add、compare&swap がオペレーションとして定義されている。

・Bind Window

メモリ保護オペレーション（後述）をハードウェアが直接サポートする。

・Multi-Cast

Datagram のマルチキャスト通信。

ただし、これらの機能と、先の 5 種類のサービスのすべての組み合わせが可能なわけではない（**表 1** 参照）。

Reliable Datagram の実現のために、各ノードには EE(End-to-End)-Contexts があり、これが送受信両 QP 間を仲介して順序保証や送達保証をする。

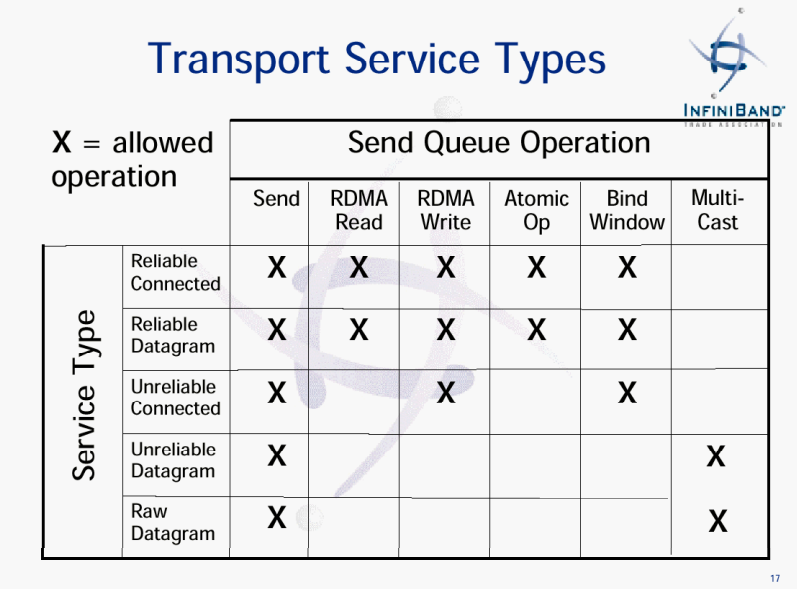


表 1　トランスポートサービスタイプと利用可能な転送機能

4.4　メモリマネジメント

データ転送用のアドレスとして、ソフトウェアからは、送信相手先の仮想アドレスを指定する。仮想アドレスから物理アドレスへの変換は、HCA の役割である。IOMMU を HCA が持っていると考えればよい。その際、Read/Write の可否、リモートアクセス/ローカルアクセスの可否などのアクセス権を細かく制御できる。

4.5　verb アーキテクチャ

InfiniBand では、ハードウェアとソフトウェアのインタフェースとして、API を定義する代わりに「verb（動詞）」によって機能だけを定義している。これには、具体的な API の定義は OS ベンダの担当範囲としつつ、何ができるかだけを強く規定する意図がある。

verb の機能には、たとえば QP・CQ（Completion Queues:データ転送の完了通知用）の操作や、送受信要求、メモリリージョンやウィンドウの操作に関するものなどがある。verb は、HCA と HCA ドライバの間に置かれる。

verb の使用手順は、例えばディスクの読み出し要求なら、次のようになる。

・オープン
・CQ、QP 生成
・コネクション生成
・メモリ登録
・要求発行
・要求送信完了
・I/O
・I/O 完了
・コネクション切断
・Takedown

4.6　マネジメント

InfiniBand では、インバンドで Unreliable Datagram を使ってマネジメントを行なう。マネジメント用のパケットを MAD（MAnagement Datagram）と呼んでいる。HCA、TCA、およびスイッチは、すべて MAD を処理するエージェント SMA（Subnet Management Agent）を載せていなければならない。各装置で実際の処理を行なう SMA の上にサブネットマネージャがいる。サブネットマネージャは各サブネットにつき１つだが、マスターの他にバックアップがいくつかいても良い。パブリックなルーティング制御などは、このマネージャが行なう。またコネクションを張るときには、コネクションマネージャがサブネットマネージャから必要情報を引き出す（**図 8** 参照）。

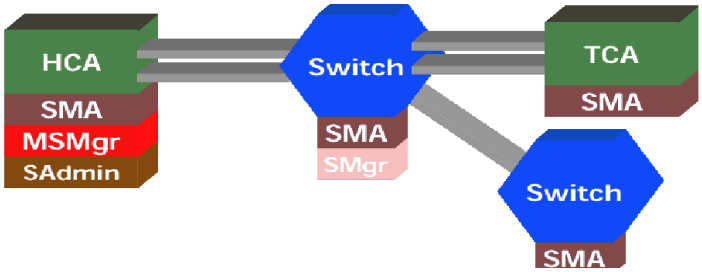


図 8　マネージメントモデル

サブネットマネージャ以外にもいくつかのマネージャがいる（図 9 参照）。これらを General Service と呼ぶ。

・General Services

Subnet Administration
Connection Management
Performance manegement
SNMP Tunneling
Baseboard Manegemnet
Device Manegement
Vendor Specific

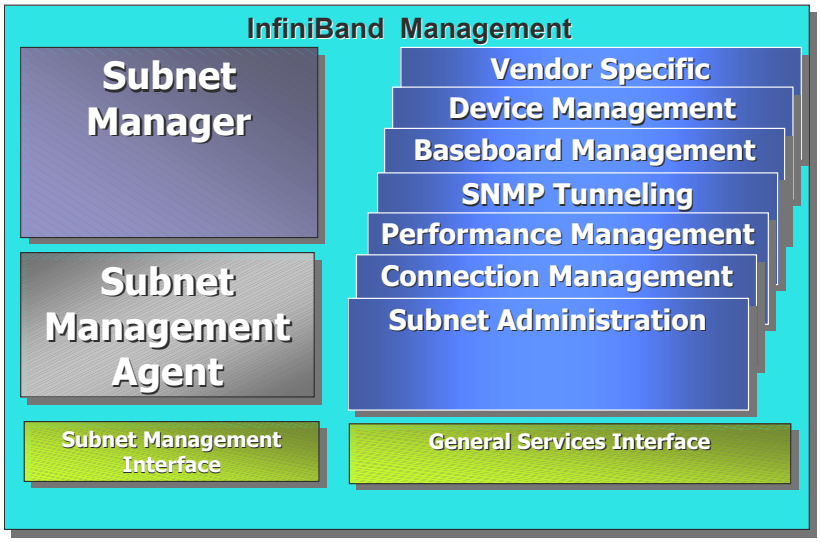


図 9　マネージメント機能一覧

5.　略語一覧

CA:　Channel Adapter
CI:　Channel Interface
CQ:　Completion Queue
EEC:　End to End Context
HCA:　Host Channel Adapter
IB:　InfiniBand
L_Key:　Local Key
MAD:　Management Datagram
QP:　Queue Pair
R_Key:　Remote Key
RC:　Reliable Connection
RD:　Reliable Datagram
RDD:　Reliable Dagagram Domain
RDMA:　Remote DMA
SA:　Subnet Administrator
SL:　Service Level
SMA:　Subnet Management Agent
SMgr:　Subnet Manager
TCA:　Target Channel Adapter
UC:　Unreliable Connection
UD:　Unreliable Datagram
VI:　Virtual Interface
VL:　Virtual Lane
WC:　Work Completion
WQE:　Work Queue Entry
WR:　Work Request

3.3　グローバルコンピューティング関連

3.3.1　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

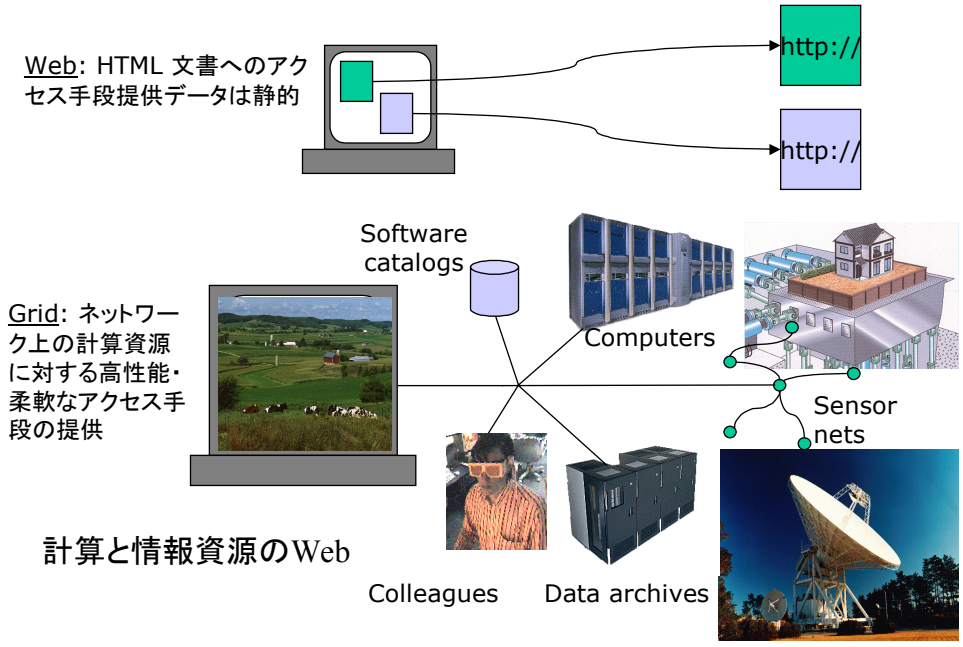
関口　智嗣委員

1. Gridによる世界戦略

Grid（グリッド）とは分散した情報資源を統一的に扱うためのアクセス手段である。高速なネットワークの発展はこれまでWebというHTML文書へのアクセス手段を提供してきたが、どちらかというと静的に与えられたデータに対するアクセス技術であった。これに対して、Grid はネットワーク上の情報資源に対する高性能かつ柔軟なアクセス手段を提供することを目指している（**図 1**）。ネットワーク上の情報資源とは、高性能コンピュータ、データアーカイブ、大規模センサーや観測装置、多種多様なソフトウエアと加えて遠隔地にいる仲間たちを含む。これらの情報資源を必要なときに必要な資源にアクセスし、自分の手元の端末を通して自由自在に操ることができるものである。

Gridによりできることの例として、

(1) 世界中のスーパーコンピュータを同時に複数台使用することでこれまで実現できなかった超大規模計算を行ってみること（Metacomputing）



（By courtesy of Ian Foster）

図 1　The Grid : The Web on Steroids

(2) 世界中の PC などの余剰時間を使ってこれまでできなかった発見的計算を行ってみること（Throughput computing, SETI@HOME）
(3) 席にいながらにして、遠隔地の共同研究者と同じ画面イメージ、同じ計算結果等を共有して高度なコラボレーションを行うこと（Access Grid）（**図 2**）
(4) 高エネルギー実験装置、実大規模 3 次元振動台、高感度望遠鏡等の超大型実験施設から得られるデータを遠隔地からアクセス、処理を行うことで大規模科学を遂行すること（Data Grid）（**図 3**）

などが挙げられる。これらの技術を実現するためのソフトウェア、ハードウェアインフラストラクチャが求められている。

これらをまとめると、**図 4** にあるように、超高速ネットワークで接続される多様な情報資源を統一的なユーザインタフェースで扱うために共通的なミドルウェアが必須の技術となってきている。上位への API ならびに下位とのプロトコルと API はミドルウェアをどのように構築するかという設計の根幹を担っている。すなわち、このミドルウェアを征するものが Grid 技術を征すると言っても過言ではないだろう。しかしながら、このような大規模になった Grid 技術においてはもはや単独のソフトウェアパッケージがすべてを支配することや、今からまったく新たにミドルウェアパッケージを作ることは困難である。

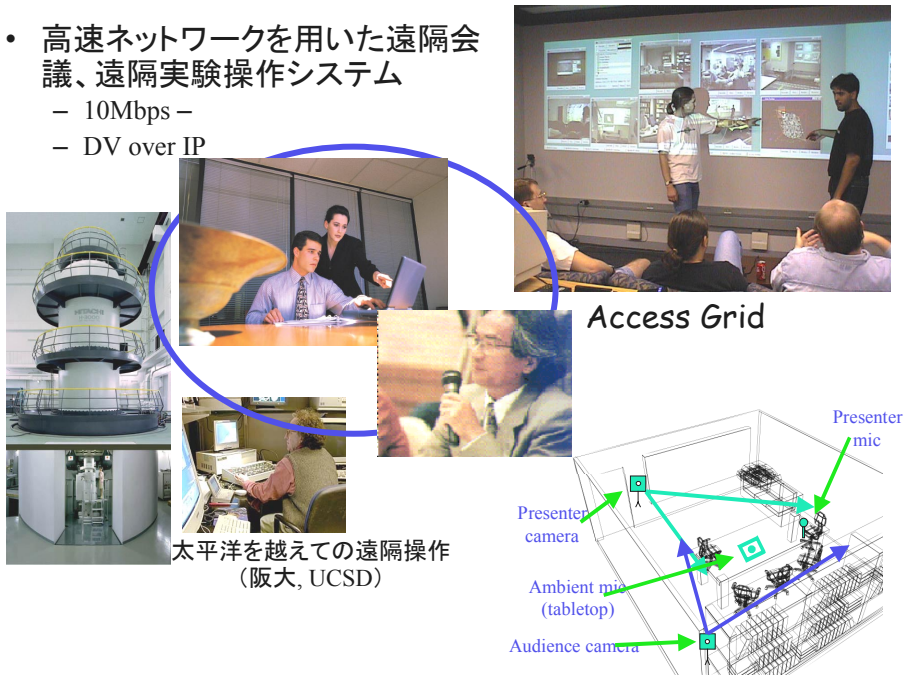


図 2　ネットワークを使ったコラボレーション

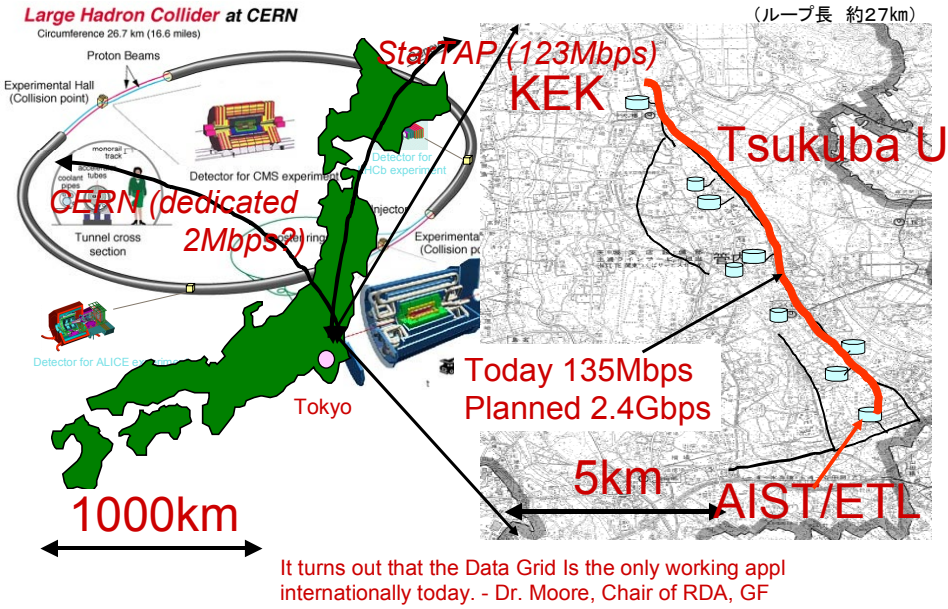


図３　大容量データ処理の例(e-science)

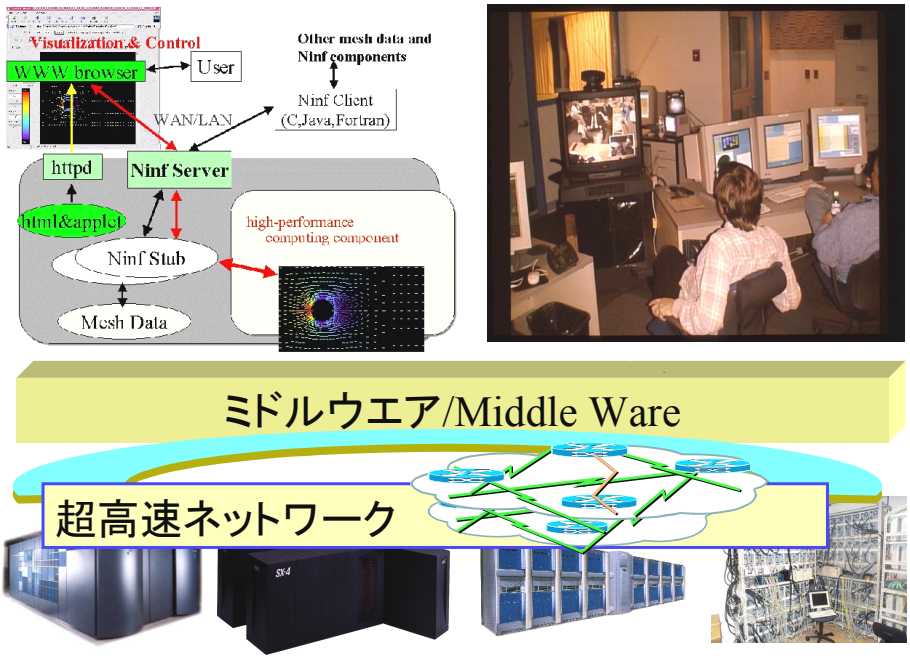


図４　Grid ミドルウェア

これらの標準化ならびに国際的協力関係の下、共同で Grid 技術に関する情報交換を行うフォーラムが設置された（図 5）。これまで、Grid Forum として米国を中心に 5 回開催された（http://www.gridforum.org/）。この間、ヨーロッパにおいて eGrid が立ち上がり、また 2000 年 6 月に Asia-Pacific における Grid インフラとしての ApGrid（http://www.apgrid.org/）が立ち上がった。アジア太平洋地区においては欧米での Grid の立ち上がりとそれぞれに巨額な予算（数百万ドル以上）が手当てされ、その勢いはアジア太平洋の各国にもいずれかへの所属を迫るものであった。しかし、ApGrid により、アジア太平洋諸国は独自の国際学術ネットワークである APAN（図 6）を中心にひとつのテストベッドを構築することとなった。こうして、米－欧－亜の 3 極体制を整えることにより、日本を含めたアジア・太平洋地区からの Grid に対する技術発信を行う橋頭堡を築くことができた。これらは Global Grid Forum において世界における統一的な Grid インフラを議論し定めていく場へと繋がっている。

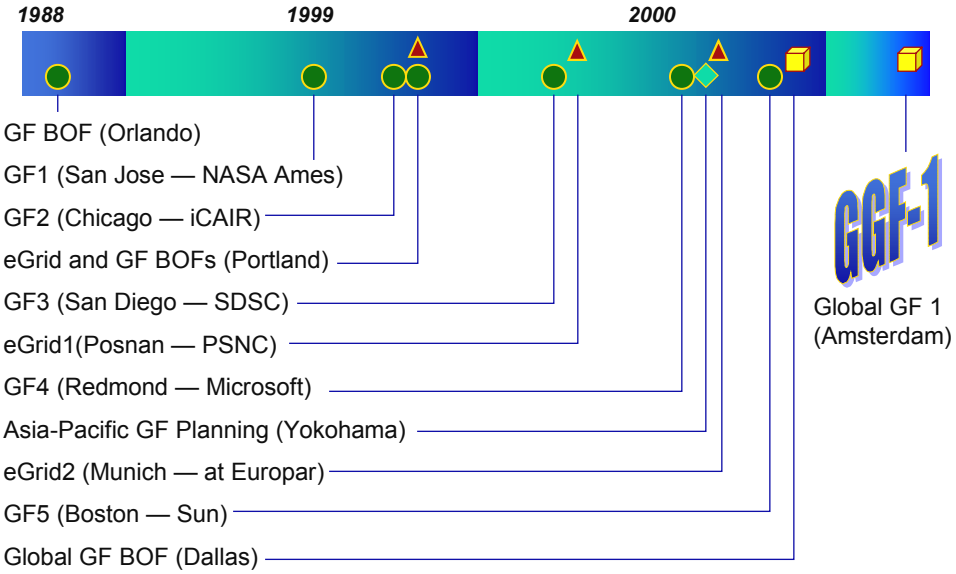


図 5　Global Grid Forum History (by C. Catlett, ANL)

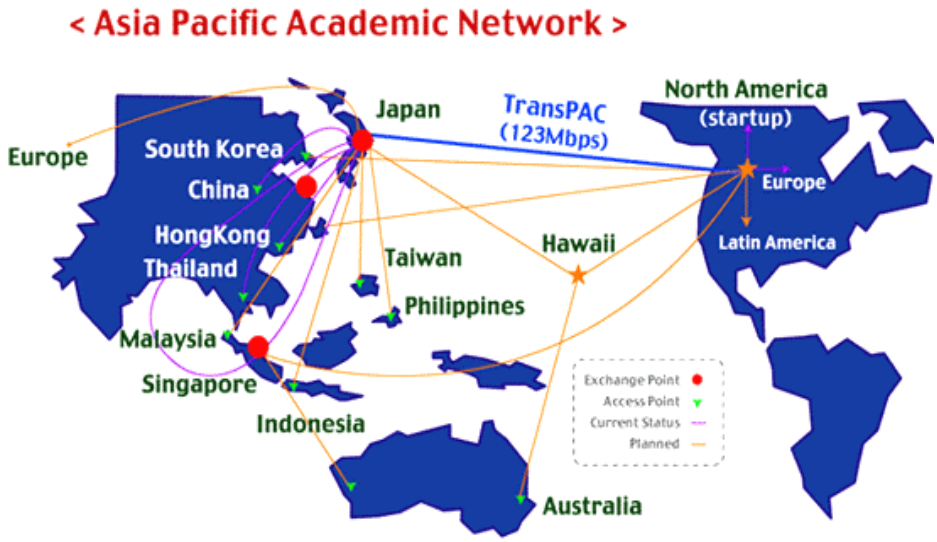


図６　ApGrid と APAN

2. Grid からブロードバンドへ

「国家産業技術戦略」において情報通信関連技術は「高機能で柔軟かつ安全な情報通信環境を活用して、時間や場所の制約を受けずに、必要とする情報・知識を誰もが自由自在に創造・流通・共有できる高度な情報通信社会」を目指すとある。具体的には、ネットワーク関連技術として、2010 年までにアクセスサービスとして事業所:100Gbps 級を目指し、高度コンピューティング関連技術として 10Pflops の演算能力ならびに 1TB の主記憶、1PB（ペタバイト）級の大容量高速記憶装置が不可欠であるとされている。この目標値により、高性能計算技術（HPC）におけるコンピュータの演算速度・信頼性の向上、シミュレーション技術における多方面への応用、大容量・高速記憶装置の確立を目指すことが急務である。

ブロードバンドコンピューティング技術研究開発は現在の一般的なインターネットとではなくその 1000 倍以上の高速ブロードバンドネットワークを指向し大容量、高性能システム技術開発である。具体的には 1PB 級の大容量高速記憶システムを構築する。この実現方式としては複数のサイトに高性能ノードを設置しこれを高速ネットワークで接続した分散処理方式による。個々のサイトにおいてはコモディティである PC 技術を基本としたノードを実現する。これらのサイトをアクセスサービスとしての 10Gbps 級大容量高速ネットワークを用いた分散処理技術を開発することでペタバイト級の大容量記憶を実現する。

この大容量記憶システムは、いわゆる e-business において超高速検索エンジンの実現、これによる検索精度向上、金融ビジネスのリスク分散、データマイニングサービス他への応用が期待される。また e-Science においてはすでに超大容量データ処理（データ管理と分散処理）が必須でありポストゲノムサイエンスはもとより、高エネルギー物

理、天文データ処理他での応用が期待されている。これらは、ペタバイト級のデータ処理を行うためのパイロットアプリケーションであり、現状民間におけるデータハウスの100 倍を超えるシステムの実現はビジネス系ならびに科学技術系における応用分野の開拓が期待される。

このような大規模記憶システムは世界の標準がまだ確立しておらず、我が国が先行している光ネットワーク技術ならびに Grid と呼ばれる分散コンピューティング技術においてデファクト標準の確立を目指していく。特に、標準化がソフトウエアとして実現されている分散コンピューティングのサイト技術において、本研究開発で実現されるハードウエアによる分散ノード構築技術は先駆的なものとなり、標準化戦略の遂行により世界的に市場をリードすることが可能となる。

これを産学官連携の体制においてサイトにおけるノード構築技術、サイト間の光ネットワーク技術、ストレージシステム構築技術の開発が求められている。

3. 具体的研究課題

3.1　実現イメージ（ハードウエア）（図 7）

(1) ノード: PC+Disk の pair で構成

それぞれ、単独で高性能ノードかつ柔軟性のある再構成可能な計算ノードとして実現し、高性能 PC の実現や、ビジネス用サーバへの技術転用を可能とする。

(2) サイト：ノード間を高速インタフェースで接続

次世代 PC 用標準インタフェース技術である Infiniband 等で接続したクラスタノードとして実現を目指す。高性能クラスタ構築技術はビジネスサーバ、インターネットサーバ、Web server farm として技術転用が可能となる。

(3) 中距離サイト：サイト間を DWDM 技術により接続

Infiniband を次世代高速ネットワーク技術である DWDM 上に実現することで接続を行う。次世代の光インターコネクト技術、フォトニックネットワーク技術の開発を行う。

(4) 広域ファーミング：サイト間を高速ネットワーク技術により接続

インターネットに用いられる Ethernet プロトコルを基本にして、SDH 等の高速通信基盤上の 10Gbps, 40Gbps 等のネットワーク技術を用いて、広域分散サイト間の接続を行うための技術開発を行う。

(5) 高信頼ディスクアレイ技術

単にドライブ数を増加して接続するだけでは信頼性の高いストレージの構築ができない。テープドライブへのバックアップも現実的に不可能だとすれば、大容量化を実現すると同時に RAID 技術による高信頼化を実現する必要がある。同時に、高速インタフェースへの対応、設置床面積の節約のため小型化、低消費電力化が必須である。

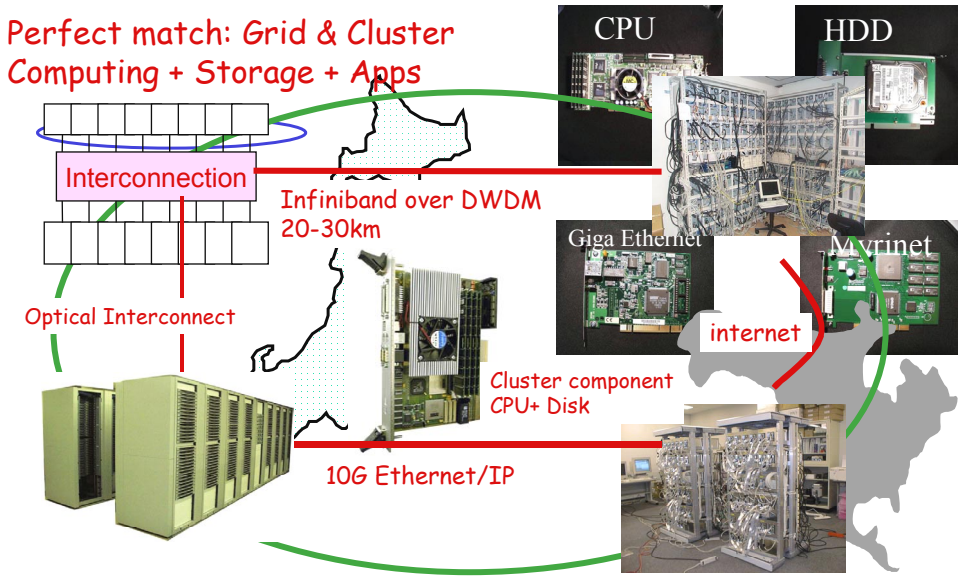


図 7　ブロードバンドコンピューティング
実現イメージ ― hardware ―

3.2　実現イメージ（ソフトウエア）（図 8）

(1) Grid 基盤技術

広域ネットワークの利用技術を促進するためのミドルウエアが Grid と称される。ブロードバンドコンピューティングにおいては高速ネットワークを基盤とした分散処理技術が基本であり、このための Grid 技術開発が必須である。しかし、Grid は世界でのデファクト標準化が GLOBAL GRID FORUM において進行しており、その活動を通じた標準化への貢献を実現する。国内からも電総研（関口）、東工大（松岡）が運営委員（8 名中の 2 名）としてチャネルを有している利点を活かす。

(2) Grid セキュリティ基盤

広域ネットワークの利用技術を促進するためのミドルウエアが Grid と称される。広域分散環境における認証のためのセキュリティ技術が必要であり、IPv6、IP sec 等の先進的インターネット技術をブロードバンドコンピューティングに導入する。

(3) データファーミング技術

大容量データは単なる生データとして保持するのではなく、関連づけられた意味を持たされたデータとして処理される。これを広域に分散させることと、それぞれのデータに対する処理を関連づける。このためのプロトコルの確立が必須で、今後の世界的標準化を目指す部分である。

(4) クラスタシステム技術

ブロードバンドコンピューティングのサイトにおけるノードは PC クラスタ技術である。このクラスタにおける負荷分散技術、高信頼化技術、開発環境等のシステム構築技術は即効性のある技術開発である。

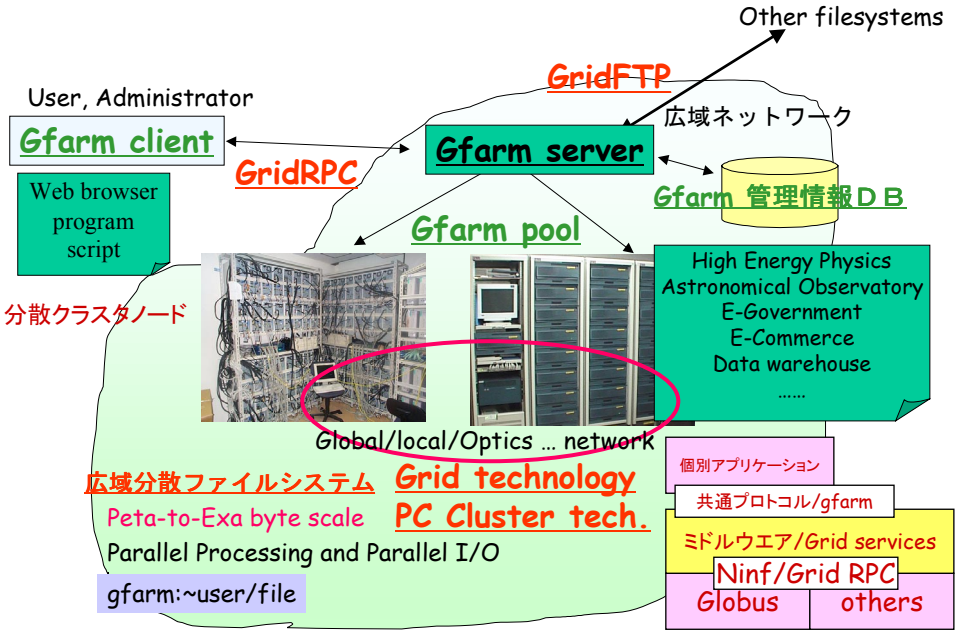


図８　ブロードバンドコンピューティング
実現イメージ ― software ―

4. おわりに

Global Grid Forum を通じた Grid 技術に関する世界戦略と、次の Grid とその応用技術であるブロードバンドコンピューティングによる Data Farming について概略を述べた。Grid は世界的に大きな技術潮流となっており、我が国も現時点での先進性とこれまでのポテンシャルを活かして大いに貢献することで、世界標準構築への参画を積極的に推進する必要がある。現時点で我々が持ちうるハイエンドコンピューティングの技術シーズとしては Grid とクラスタであることは疑いもない。また、こうした技術において先行してきた強みを活かし、国内のコンピュータベンダーを巻き込んだプロジェクトとして欧米に対抗する道筋を見つけだしていくことが必要である。

3.3.2 Gridにおけるセキュリティ技術

門林 雄基講師

1. はじめに

Grid は分散計算、可視化、ディレクトリサービス、認証など実証計算機科学における近年の技術革新の集大成であるととらえることができる。Grid 研究ではこれらの基盤技術を単一のミドルウェアに統合する作業が活発におこわれているが、このうち、最も取り組みが遅れているのがセキュリティ技術の統合作業であると考えられる。

Grid の積極的な応用が始まっている研究分野では、セキュリティ技術の重要性は十分認識されていると考えられる。応用分野として、高エネルギー研究、宇宙科学研究、デバイス技術、バイオテクノロジー研究、ナノテクノロジー研究などが考えられ、またこのうちのいくつかでは応用が始まっているからである。

これまでの研究機関や国立大学に対するサイバー・アタックは高エネルギー研究、バイオテクノロジー研究、宇宙科学研究などをおこう研究部門に対するものが顕著であったことを鑑みれば、Grid 技術がこれらの研究部門で広く用いられることになったとき、Grid の個々の基盤技術を精密に分析し、またソフトウェアのバグを洗い出すなどの手段によって、研究の基盤であるところの Grid に対するサイバー・アタックが最も効率的な攻撃手段となりうる。

このようにセキュリティやプライバシを最も必要とする応用研究分野において Grid 技術を実際に使っていくためには、Grid のセキュリティアーキテクチャを精査し、また Grid のコンポーネントとなるソフトウェアそれぞれについてセキュリティ検査をおこうといった作業が必要不可欠であると考えられる。

現在のところ、このような作業が体系的かつ組織的におこわれているという情報はまったくなく、またそのような作業の必要性も広く認識されているとは言い難い。このような現状において、Grid をセキュリティがもっとも重要視される各応用研究に対して実用に供することは、拙速というほかない。

ここでは Grid という大きな概念の一実現である Globus について、そのセキュリティアーキテクチャを概観し、その利点と欠点について考察する。むろん、ここでの考察は断片的なものでしかない。包括的なセキュリティアーキテクチャに関する考察や、ソフトウェアの実装にまで踏み込んだセキュリティ検査といったことは近い将来において取り組むべき課題として残されているということに注意されたい。

つぎに、Grid 技術の標準化団体である Grid Forum における Grid のセキュリティアーキテクチャ標準化について、潜在的な問題点の有無を点検する。最後に、Grid におけるセキュリティ研究の方向性について議論をおこなう。

2. Globus におけるセキュリティ

Globus は USC/ISI（南カルフォルニア大学 情報科学研究所）を中心とする全米の研究者によって活発に設計、開発、改良がすすめられている Grid のミドルウェアである。

Globus のセキュリティアーキテクチャの大部分は X.509 の電子証明書にもとづくセキュリティアーキテクチャから派生したものである。X.509 のセキュリティアーキテクチャは歴史的には ITU-T の X.509 作業部会によって標準化がおこなわれ、4 年ごとに改版されてきたが、近年は電子商取引をはじめとして、X.509 標準のインターネットへの応用が盛んになってきたため、IETF の PKIX 作業部会に標準化が引き継がれ、本稿の執筆時点においても活発に機能拡張のための提案作業や規格書の改版作業が PKIX 作業部会を中心としておこなわれている。

Globus では X.509 の電子証明書（Digital certificate）をもちいたホストの認証とユーザの認証をおこなっている。それぞれの計算機や、計算機の利用者にたいして DN（Distinguished Name）と呼ばれる識別子を付与し、それぞれの DN に対して電子証明書を発行する。電子証明書は利用者の計算機においていったん生成され、これを CA（Certificate Authority）において CA の秘密鍵で署名することによって正式なものとなる。CA は秘密鍵と公開鍵の組をもっており、公開鍵をふくむ情報を CA の電子証明書として広くすべてのホストから参照可能とすることで、どのホストにおいてもユーザの認証が可能となる。ユーザからアクセス権限をもとめられたときに、ホストはみずからが持つ CA の公開鍵をもちいてユーザの電子証明書を確認することができる。

DN はグローバルに一意な識別子であるので、これに付帯する電子証明書をもちいることでグローバルなユーザやホストの認証をおこなうことができる。しかしながら、DN は実際にユーザプログラムを実行するうえで個々の計算機のユーザ ID にあるため、Globus では個々のユーザの DN からユーザ ID への変換情報をひとつのファイルにおさめる方式をとっている。変換情報をおさめたファイルは grid-mapfile と名付けられ、これをシステム管理者が利用権限を許可するユーザごとに書き込むことで個々のホストにおけるアクセス制限が実現される。

このように Globus のセキュリティアーキテクチャは既存の優れたアーキテクチャを応用することによってグローバルな認証を実現したものであるが、問題点がないわけではない。以下では、本稿の執筆時点における Globus のセキュリティアーキテクチャにおける問題点について議論する。

Globus のセキュリティアーキテクチャでは認証のみをおこなっており、暗号化に関しては考慮されていない。Globus ではジョブの投入、ノード間のメッセージ交換、ファイル転送など、さまざまなコンポーネントにおいてネットワーク上でデータをやりとりするが、このうち、ジョブ投入時の認証以外にはセキュリティを確保するための操作はおこなわれない。現在のところ、Globus の実装では OpenSSL の認証部分のみを用いて認証をおこなっており、ファイル転送における内容の機密保持や、ノード間のメッ

セージ交換におけるなりすまし攻撃への対処は考慮されていない。

もちろん、このようなセキュリティアーキテクチャを採用したことには理由がある。認証操作を最小限にとどめることは設計上の選択肢であったと考えられる。認証操作を最小限にとどめることで、ファイル転送やノード間のメッセージ転送における認証のオーバーヘッドをなくし、分散計算を高速化することができる。また、暗号化をおこなわないことでメッセージの送受信における計算負荷をなくすこともできる。しかしながら、常にメッセージの暗号化をおこなわないことと、常にメッセージの認証をおこなわないことは Globus が用いられる状況によっては危険性をともなう。したがって、アーキテクチャとしてはプログラマに対し選択肢を用意すべきだと考えられる。

また、Globus における X.509 セキュリティアーキテクチャの実現方式にも問題点がある。X.509 では CA を階層化して運営することによって、証明書発行や証明書の維持管理業務の負荷を分散させることが意図されているが、Globus においては、本稿の執筆時点では CA は階層化されておらず、ひとつの CA によってすべてのノードとユーザの証明書が発行され、管理されている。直感的にも明らかなように、単一の CA によるグローバルなユーザ認証とホスト認証には規模の限界がある。

X.509 では証明書の秘密鍵が漏洩した場合や利用者が解雇された場合などを想定し、証明書の取り消し（Certificate revocation）という操作が規定され、証明書の有効性を問い合わせるプロトコルが規格化されているが、本稿の執筆時点では Globus には証明書の取り消し操作が用意されていない。

以上の議論からわかるように、Globus で用いているのは X.509 セキュリティアーキテクチャのサブセットであり、大規模な運用や現実に起こりうることに対応できる運用を可能とするものではない。

Globus セキュリティアーキテクチャのもう一つの問題点として、権限委譲モデルが非常に単純であるということが挙げられる。grid-mapfile にエントリがあるユーザは、権限を与えられた計算機上で通常のユーザとして振る舞うことができ、あたえられたユーザ ID に対して許される範囲でファイルの読み書きやプロセスの実行が可能となる。

3. Grid Forum におけるセキュリティ標準化

現在、Globus, Legion などにおける Grid の設計、実装経験にもとづき、Grid Forum において Grid アプリケーションプログラミングインターフェースの標準化が積極的にすすめられている。

Grid Forum において Grid Security Protocols として標準化がすすめられているものは、PKIX と Kerberos を土台としたものである。また、アプリケーションプログラミングインタフェースとしては GSS-API がたたき台として考えられている。

このように、専門家によって設計された既存のセキュリティアーキテクチャにもとづいて標準化作業をすすめることで、Grid Security Protocols の標準化を早期に完了させ

ることができるという利点がある。しかしそのことは同時に、PKIX や Kerberos の抱える問題に並行して取り組まなければならないということも意味する。

PKIX では電子証明書の本人確認を効率的に行う方法が必要であり、一般的な解決策はない。特にグローバルに分散して存在する Grid の利用者をどのようにして効率的に、かつ間違いなく本人確認するのかという問いに対して解決策を見いだす必要がある。このような問題が X.509 だけでなく、電子証明書を用いる PKI（Public Key Infrastructure）全般について指摘されている。詳しくは、

D. Davis, “Compliance Defects in Public-Key Cryptography”,
Proc. 6th USENIX Security Symposium, pp.171-178, 1996.
http://world.std.com/~dtd/

あるいは

C. Ellison and B. Schneier, “Ten Risks of PKI”, Computer Security,
Journal, 16(1), pp. 1-7, 2000.
http://www.counterpane.com/pki-risks.html

を参照されたい。

このほか、PKIX に基づくセキュリティアーキテクチャでは米国の Federal PKI やわが国の GPKI（Government PKI, 政府認証基盤）と同じ問題に直面することになる。例えば複数の Root CA が存在する場合、それらをまたがったユーザ認証やホスト認証をおこなうためには Root CA 同士をつなぐ Bridge CA が必要だと GPKI では考えられている。これと同じ問題を Grid においても解決する必要がある。

また、証明書の有効性問い合わせについても問題点があることが知られている。証明書の有効性問い合わせ間隔をあまり短くとると、証明書検証サーバの負荷が高くなり現実的ではない。逆に、有効性問い合わせ間隔が長すぎると、無効な証明書をもつユーザのアクセスを許可してしまうことがあり、好ましくない。

また、CA にもとづくシステムでは CA の秘密鍵の漏洩がもっとも深刻な脅威であり、この問題にたいしてどう対処するのかといった合意形成と、緊急時行動規定の作成が必要である。

以上のような電子証明書の問題点のほかに、個々のノードは従来のインターネット・セキュリティの諸問題を抱えていることは言うまでもない。オペレーティングシステムに標準で付属するデーモンのいくつかにはセキュリティホールがあり、これらを悪用することにより管理者権限を奪取される危険性がある。また、DNS を用いてドメイン名から IP アドレスを検索する際に、なりすまし攻撃によって DNS サーバの応答を誤った応答にすり替えることが可能である。これによってデータの漏洩が起こる可能性がある。

さらに、TCP/IP におけるよく知られたセキュリティ上の諸問題を悪用した攻撃も想定しておくべきである。たとえば、IP 始点アドレスを詐称することにより分散計算の

メッセージ交換に悪影響を与えることが可能である。また、TCP コネクションの乗っ取り攻撃によって認証後のコネクションを乗っ取られ、計算機の制御を奪われる可能性もある。

これらの諸問題は Grid セキュリティアーキテクチャの構築とは別に解決すべきだと考えられるが、これらの諸問題を抱えたまま Grid 技術を実用化することは非現実的である。

4. Grid におけるセキュリティ研究の方向性

Grid を実証計算機科学における技術革新の集大成であるととらえれば、Grid におけるセキュリティ研究のあり方は明確であると考えられる。つまり、近年のインターネット・セキュリティ関連の技術革新を積極的に取り入れ、より安全な Grid セキュリティアーキテクチャを構築するのがとるべき方向であろう。

まず、ネットワーク層セキュリティの標準である IPsec,および IPsec のための鍵交換プロトコルである IKE を考慮してセキュリティアーキテクチャを設計すべきである。IPsec によって、悪意あるノードからのがい乱トラフィックの挿入を防止することができ、これによってコネクションの乗っ取りや始点アドレスの詐称といった問題を解決することができる。また、DNS 応答の偽造とすり替えについては DNS のセキュリティ拡張である DNSSEC を早期に運用することによって解決すべきである。

また、デーモン等のソフトウェアのバグにより管理者権限が奪取されるという問題に対しては、より安全なオペレーティングシステムを採用することによって回避、もしくは脅威を軽減することができる。そのような安全なオペレーティングシステムとしては、たとえば Immunix を挙げることができる。Immunix では、Stack protection という独自の方式によってスタック上での意図しないコードの実行を防ぐことができ、これによってソフトウェアのバグを悪用した管理者権限の奪取をかなりの程度、防止することができる。

しかしながら Stack protection などの対策方式は特定の状況におけるプログラムの誤動作を防ぐことができるだけであり、完全な対策方式は現在のところ実現不可能だと考えられている。より本質的には、Grid を構成するソフトウェアのコードを一行ずつ検査し、セキュリティホールの有無を人手によって確認するという膨大な作業が必要となる。

3.4　プログラミング／コンパイラ関連

3.4.1　変貌するプログラム言語の位置づけ

近山 隆委員

高性能計算・通信の実現には、ハードウェアとソフトウェアの両面が重要であることは言うまでもない。ソフトウェアの記述には通常なんらかのプログラム言語を用いるわけだが、解くべき問題とハードウェアアーキテクチャの両者の複雑化や、コンパイル技術の進展に伴い、プログラム言語の表面上の記述と、その記述内容の計算機による実現との間の乖離が拡大してきている。このため、従来常識とされてきた、手続き型言語は計算機による計算手順を記述し、宣言型言語ではプログラムを実行する計算機と独立に問題仕様を記述する、といった言語族についての認識は適切性を失いつつある。
本稿では近年のプログラム言語技術と計算機アーキテクチャ技術により、プログラム言語の位置づけがどのように変化してきたか、今後さらにどのような変化が予測されるかについて述べる。

1. 手続き型言語と宣言型言語

計算機のソフトウェアはなんらかのプログラム言語で記述するのが通例である。高性能計算通信システムにも、なんらかのプログラム言語によるソフトウェア記述が欠かせない。プログラム言語は、その設計原理、利用目的とする対象分野、利用者とのインタフェース方法などによって、実に数多くのものが使われてきている。こうしたプログラム言語を分類する軸として代表的なもののひとつが、手続き型か宣言型かという視点からの、ふたつの言語族への分類である。

手続き型（procedural）あるいは命令型（imperative）と称される言語族は、機械語を直接用いた記述から自然に発展してきた体系といえる。

計算機の動作を直接的に指示する機械語命令列によるプログラム記述は、たとえば分岐命令の飛び先の命令番地なども、その番地を直接数値として記述するため、プログラムのごく小さな変更にあたっても、かなり大幅なプログラムの修正が必要であった。また、機械語命令のオペレーションコードも直接数値で指定するため、プログラムの読解性がごく低いという問題があった。そこで、こうした指定に文字列による「名前」を用いるようにしたアセンブリ語が使われるようになった。

ついては、ほとんど同じような機能を持つが、詳細においては異なる部分のある異機種計算機に対し、個別にプログラムを作成する労力が問題となった。ある抽象レベル、たとえばメモリ内容を読み出す、加算を行う、といったレベルでは同じ操作を行うにもかかわらず、機種が異なれば異なる機械命令体系を持っているため、アセンブリ語によ

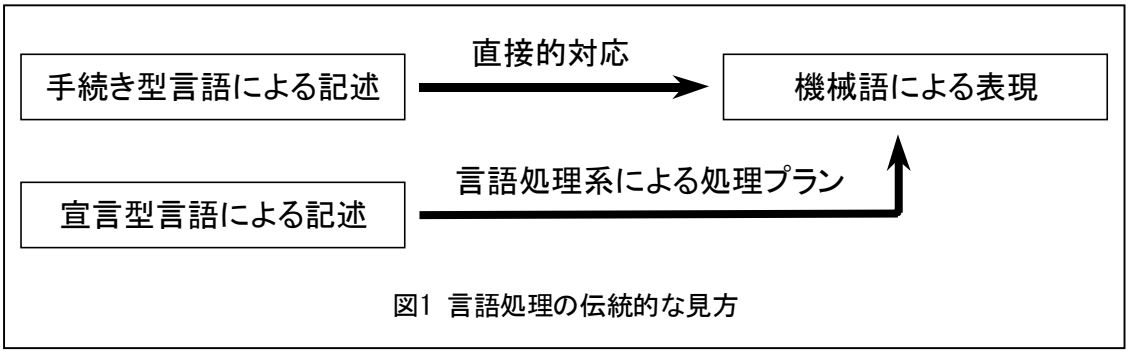


図1 言語処理の伝統的な見方

る記述では異なるプログラムとならざるを得ない。そこで、通常の計算機システムならば必ず持っているような処理機能を抽象化し、そうした機能の系列を記述できるような言語を設計した結果が、Fortran などの手続き型プログラム言語である。

このような生い立ちのものであるから手続き型言語は、記憶装置の内容を読み出して CPU を用いた処理を行い、その結果をまた記憶装置に蓄えるという、実際の計算機が共通して持つ機能を、ある程度高い抽象レベルで記述することが出来るような言語になっている。こうした意味で手続き型言語の記述は、どのように計算を進めるか（how）の記述であるということが出来る。

一方これに対し、まったく異なるアプローチから設計されたのが、関数型や論理型などの宣言型（declarative）、記述型（descriptive）、あるいは非手続き型（non-procedural）と呼ばれる言語族である。宣言型言語によるプログラムの記述は、実際の計算機による計算の過程そのものをいったん離れ、数学や論理学といった形式的体系に基づいて、計算すべきものは何であるか（what）を記述する。記述するのは何を計算するかであって、どうやって計算するかではなく、計算の具体的手順は言語処理系に委ねられる。このことが両言語族にさまざまな違いが生じる根源となっている。

2. 言語族の特徴

プログラム言語を手続き型と宣言型の両言語族に分類するとき、同じ言語族に属するプログラム言語には、共通した長所・短所がみられる。

2.1 伝統的な見方

宣言型言語は形式的体系に拠った意味論をもっており、具体的な計算機の機械語から出発して抽象化していった手続き型言語とは異なる性格を持つ。このため、宣言型言語は手続き型言語に比して意味論の抽象度が高く、より高い抽象レベルでの記述が可能で、このため同じ内容のプログラムをより簡潔に記述できるとされてきた。

たとえば、論理式で表された一定の制約を満たす解を求めるような問題を解く場合、Prolog のような論理型言語を用いれば、ほとんど制約論理式の形式を計算機入力に便利な形式に変換するだけの操作でプログラムを構成できることが多い。同じ問題に対す

るプログラムを C などの手続き型言語で記述するには、制約論理式を評価手続きとして記述しなければならないのはもちろんのことであるが、その式を満たすような解候補をどのようにして生成するか、どのような順で制約充足の成否を判定するかなど、解法を手順の詳細にわたるまで指定する必要がある。

一方、手続き型言語プログラムを機械語に翻訳するのは、抽象化の際に通ってきたパスを逆にたどるだけなので、コンパイラなどの言語処理系を構成することは容易である。宣言型言語は計算機と直接的な関係のない形式体系に基づいた意味論を持つため、処理系の構成は容易ではないとされる。効率的な処理の実現のための手がかりは乏しく、そのまま素直に機械語に翻訳すればよい手続き型言語とは大きな違いがある。

たとえば論理型言語 Prolog のある程度効率的な処理方式が確立するまでには、言語が提案されてから四半世紀近くもの年月を要しており、この間に同じ計算機ハードウェア上での Prolog 処理系の速度性能は二桁以上も向上している。もちろん Fortran などの手続き型言語においても処理系の技術進歩による速度性能向上は小さくないが、せいぜい数倍程度にすぎず、当初から最適に近い処理系が構成可能であったことがわかる。

2.2　宣言型言語による記述は容易か

前節に述べた手続き型・論理型の両言語族についての長所短所は、一般に広く認められてきたものである。しかしながら、記述の容易性に関していわれてきた長短については、これを精査すると、必ずしも承認しがたい点も明らかになる。

第一に、宣言型言語による記述が手続き型言語による記述よりも高いレベルであるとは限らない。たとえば、実際に利用されている関数型や論理型の言語は、整数値データを扱うためのプリミティブを備えているが、これは関数型・論理型言語の本質的な機能とはいいがたく、現実の計算機がもつ四則演算機能を直接的に利用するための付加的機能という面が強い。実際、多くの Prolog 処理系では、整数値データに対する四則演算は、通常の Prolog の述語が備えている入出力関係に関する双方向性を持たず、いわばままこ扱いである。

宣言型言語においては、むしろ自然数をペアノの公理に基づいて、宣言的な基本機能だけを用いて記述する方が、自然な記述であるといえよう。こう考えると、数値演算を必要とする応用について、少なくとも数値演算の部分についてのみ考えれば、機械語の四則演算命令をほぼそのまま利用することを想定して設計されている手続き型言語の方が高い抽象度を持つといえる。

同様なことは他の多くの機能についても言える。宣言型言語は採用した形式体系に沿った機能を持っているため、どのような問題に対してもその形式体系の抽象レベルでの記述が可能になる。一方、手続き型言語は計算機の物理的なハードウェアに近い機能については、むしろ宣言型言語よりも抽象度の高い記述が可能である。前述の論理型言語による制約充足問題の記述などは、計算機のハードウェア機能との直接的なマッチング

が難しく、形式論理という論理型言語の依拠する形式体系とのマッチングがよかったため、論理型言語による記述が容易であったのだ、ともいえよう。逆に、たとえば時系列が重要な要素となるグラフィック・ユーザ・インタフェースを、手順を記述の基本要素としない宣言的な枠組で記述することは、不可能ではないにせよ独自の難しさがある。抽象度の高低は単純に一次元に並べられる全順序関係ではなく、問題領域によって異なる複雑な基準なのである。

また、近代的なプログラム言語ならば何を用いても、何らかの形でプログラムライブラリを提供することができる。ライブラリを適切に構築すれば、問題領域に対する抽象度の高いプリミティブを提供できる。どのようなライブラリが利用可能であるかは、その言語のもつ歴史的社会的背景に依存するところが大きいが、少なく言語自体の問題としても、プログラムライブラリを適切な形で提供しやすいか否かは重要な特性であり、これは言語プリミティブの抽象度とは簡単な相関にはない。

第二に、仮に抽象度の高い簡潔な記述が可能であっても、それがプログラマにとって容易であるとは限らない。抽象概念は抽象概念を扱い慣れている者にとっては容易に扱えても、抽象概念を苦手とするものにとっては、むしろその簡潔性ゆえに理解も利用も難しくなることがある。筆者自身の経験からしても、用いる体系に適切な抽象度がどのレベルであるかには大きな個人差があり、簡潔な抽象レベルの記述よりも、煩瑣な具体レベルの記述の方が、はるかに理解しやすい、といった例は少なくない。

以上のとおり、記述の抽象レベルの高低や難易と、手続き型・宣言型の言語族の優劣との関係は、従来いわれてきたような単純な関係にはなさそうで、必ずしも宣言型が有利とは限らない。要は解くべき問題と言語の依拠するモデルとのマッチングの問題であるといえよう。

2.3 最適化

手続き型言語と宣言型言語の間での記述性の優劣については、前節に述べたとおり問題とのマッチングの問題であると考えられるが、性能上の優劣については、明らかに手続き型言語の優位性が認められるように見える。実際、大規模数値計算はいうに及ばず、宣言型言語が記述の簡潔さの面では明らかに優位にある組合せ的な問題においても、大規模な問題を解くための実用システムは手続き型言語で記述するのが通例である。

性能の問題を議論するためには、プログラムに対するいわゆる「最適化」の問題を避けて通れない。プログラムの性能を比較する際には、最適化を施した結果のプログラムの性能について云々するべきであって、最適化を施さない稚拙な処理系による場合の性能について云々しても意味はない。

手続き型言語プログラムに対する最適化とは、その言語によって記述された手続きを、同じ効果をもつ等価な手続きで、より効率的なものに変換することと認識できる。これは手続きから手続きへの直接的な変換であり、計算機の動作に対応付けて語られる手続

き型言語の意味論の中に閉じた変換であるといえる。

一方、本来手続きを書き表すものではない宣言型言語プログラムに対して「最適化」という言葉を用いる場合、その意味合いはかなり違ってこざるを得ない。宣言型言語に対しても手続き的な意味論を与える場合があるが、それは意味論の根幹となっている形式体系によっては説明しがたい付加的な機能に対して意味づけするなどの目的のためであり、本来の言語の性質からすれば補助的な役割を果たすものにすぎない。宣言型言語プログラムに対する「最適化」を云々する場合は、実はなんらかの標準的な言語実装方式を仮定し、その方式による処理手続きよりも効率的な実装を行うことを指すと考えられる。

2.4　手続き型言語に対する最適化技術の進展

最近の手続き型言語プログラムの最適化系は、前述したような手続きから手続きへの変換という最適化を、直接的に行っているわけではない。機械語にして数命令程度の範囲でのいわゆるピープホール最適化については、あらゆる場合を尽くした探索もそれほど大きな手間をかけずに出来るため、機械語命令列から機械語命令列への直接的な変換が十分可能である。しかし、より大きな範囲、たとえば手続きひとつ程度、機械語命令にして百命令オーダ程度以上になると、探索空間が広くなり過ぎ、記述を機械語よりは若干高い抽象レベルにおいても、このような手続きから手続きへの直接的な変換は計算量的に困難である。そこで、手続きとして記述されたプログラムを、いったん抽象的な意味領域に写像し、その領域上での表現が意味するところを、コード列という手続きの領域になるべく効率的になるように写像し直す、というアプローチが普通になっている。こうした抽象意味領域としてはデータフローグラフなどが用いられることが多い。

これを自然言語に対する翻訳の場合にたとえていうなら、手続き型言語プログラムに対する素朴な機械語生成は逐語訳に、手続きから手続きへという最適化は直訳に、いったん抽象意味領域に写像してからのコード生成は意訳に対応するといえよう。

データフローグラフは、手続き型言語プログラムに記述されていた多くの手続き的詳細を捨象している。これに基づいて手続きを再構成することによって、手続き間の直接的な変換では難しかったさまざまな最適化が可能になる。

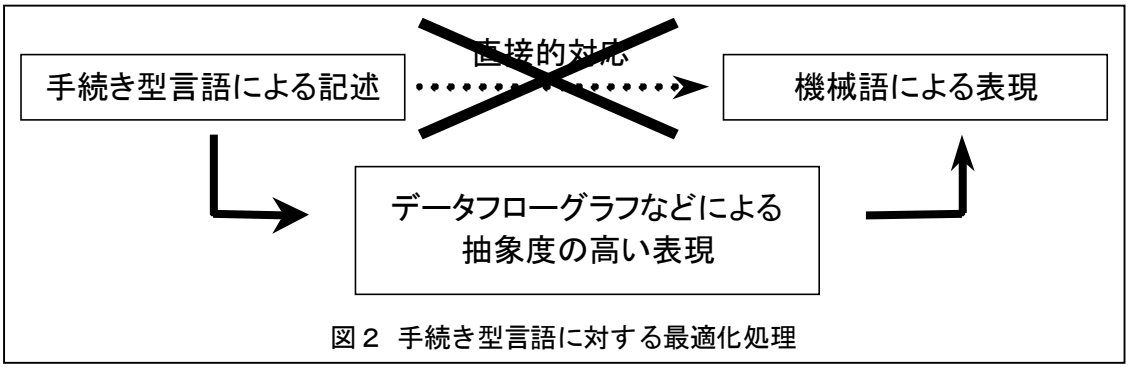


図２ 手続き型言語に対する最適化処理

たとえば、データフローグラフには実行順という概念はなく、データの依存関係があるだけである。もちろんこのデータ依存関係は元のプログラム中の実行順記述をもとに抽出したものであるが、プログラム中に記述した実行順はデータフロー依存関係にどのように影響するかを解釈された後は無視される。このことによって、実行順制約は緩和され、対象とする計算機の性質に依存して実行順を入れ換える最適化、たとえばパイプラインストールが生じにくい実行順の選択などが、自然な形で可能になってくる。共通部分式の括り出しや、さらにはコンパイル時に静的部分評価を行うなども、実行順についての自由度増大を一般化したものと見ることができる。

また、データフローグラフには元のプログラムにあった記憶場所の名前としての変数という概念はなくなっている。計算の途中結果である値というものがあるだけで、これをどこにどのような形で記憶しておくかは、データフローグラフを実現するコードの生成法に依存している。プログラム中のひとつの変数には、実行の状況によってさまざまな値が入る。それぞれの状況における変数値をどこに記憶するかは、データフローグラフを元に考える限りにおいては、まったく独立した問題であり、元のプログラムで同一の変数に格納されていたということは特段の影響を与えない。このため、プログラム中の同一の変数を、あるときはレジスタ上、あるときはメモリ中など、効率上最適な場所に配置することが自然に可能になる。さらに、変数に入っている値をこれ以上使わないという状況では、その変数の値はどこにも記憶していないという状況も起こりうる。

こうした最適化は手続きから手続きへの変換という考え方によって不可能というわけではない。実際、ごく低レベルの機械語命令列としてしかプログラムを認識できないCPU の中で、上述の最適化はそれぞれ out-of-order 実行、register renaming といった言葉で知られる最適化として、ハードウェアレベルで実現されている。こうしたハードウェアによる最適化は、どのようなキャッシュミスが生じたかなどといった実行時にしか得にくい情報に基づく最適化が可能である利点はあるが、ハードウェア資源の制約上ごく狭い範囲のみに注目したピープホールの最適化しか行なうことができず、その効果はおのずと限定されている。近年の最適化コンパイラが行うような、ある程度広範囲にわたる最適化を施すには、いったんデータフローグラフのような高い抽象度のレベルに写像した方が、見通しのよいコード生成を整合性よく行うのに有利になっているのである。

3. 宣言型と手続き型のギャップ縮小

前節に述べたような手続き型言語に対する高度な最適化技術の進展を見ると、手続き型言語の宣言型言語に対する性能面での優位性の根源であった、計算機ハードウェアとの直接的な関係は崩れつつあるといえる。

宣言型言語はもともと計算機のハードウェアとは遠い形式的体系でその意味が与えられるため、実際の計算機の持つ機能との間には小さからぬギャップがあった。一方、

手続き型言語の場合このギャップはなきに等しく、プログラム中に用いられた言語機能を逐語訳するような形で機械語に翻訳することができた。ところが、前述したとおり、効率性を追求するための最適化を施すにあたっては、この言語の持つ機能と計算機のハードウェア機能の対応関係を直接的に利用するよりも、いったんデータフローグラフのような抽象度の高いレベルに持ち上げた方が有利なのである。

こうしてみると、手続き型言語で記述するのは手続きそのものではなく、手続きという形式を借りて解くべき問題を記述しているのだ、とも見ることができる。たとえば C 言語ならば、四則演算や論理演算を行える演算装置、関数やブロックに入る際に自動的に拡張され名前文字列と関係付けられるようなメモリ装置、if-then-else, while, switch などのような実行制御装置を備えた仮想機械を考え、その仮想機械上でどのように計算を行うかを記述する。しかし、この記述は実際の計算手続きの記述ではなく、プログラムに記述した手続きを仮想機械上で実行した場合に得られる結果という形式で、結果として得るべきものを指定する記述と考えるわけである。言語処理系はプログラムを解析してどのような結果を得ればよいのかを解釈し、実際の計算手続きを決めるにあたっては問題記述に用いた手続きはいったん忘れ、指定された結果を得るためにもっとも効率的な手続きを、別途ゼロから組み立て直す、という考え方である。

データフローグラフは計算機ハードウェアと直接対応する形式でないとはいえ、宣言型言語の基礎となっている形式体系、たとえば論理型言語の基礎である一階述語論理に比べれば、はるかに計算機のハードウェアに近い表現である。そうではあっても、手続き型言語が計算機ハードウェアによる処理と一対一に直接的に対応するという考え方は、既に成り立たなくなっていると考えられる。

従来の手続き型言語に対する最適化処理は、主としてひとつの手続きの中に閉じた解析・最適化であり、手続き間にわたる広域的な最適化は現在のところ比較的限定されたものでしかない。今後、ある程度大規模なソフトウェアシステム全体にわたるような最適化を行う場合、現在よく用いられているデータフローグラフのようなレベルよりも、さらに高い抽象レベルの表現に依らなければ、妥当なコスト範囲での解析が実現できなくなる可能性が高い。このことを考えると、遠くない将来、宣言型言語と手続き型言語の差異は、計算機ハードウェアとの疎遠によってではなく、問題記述の方法論の違いという視点から語られることとなり、手続き型言語は仮想的な抽象機械上での「手続き」という形式体系にのっとった宣言型言語の一種である、とみなされるようになるものとも予測できる。

3.4.2　ハードウェアとソフトウェアの協調

笠原 博徳委員

1. はじめに

現在世界最速のスーパーコンピュータ（HPC）は 10TFLOPS を越え、来年 3 月には我が国の地球シミュレータが 40TFLOPS のピーク性能を達成する予定である。このようにスーパーコンピュータのピーク性能は年々急速に向上しているが、プログラムを実行したときの実効性能はピーク性能の伸びに比例した向上が難しく、ハードウェアピーク性能とソフトウェア実行時の実効性能の差が年々大きくなっている。また、使い勝手的にも、超高性能を達成するために用いられている分散メモリ型マルチプロセッサアーキテクチャでは十分な能力を持つ自動並列化コンパイラがないため、ユーザは問題中の並列性を自分で抽出し、MPI、HPF、PVM などの拡張言語あるいはライブラリを用い並列プログラムを作成しなければならず、一般のユーザにはハードウェアを使いこなせないという問題が生じている。さらに、これらのような価格性能比の劣化・使いにくさ等の問題にも起因して世界の HPC 市場には低迷感があり、産業界ではハイエンドコンピューティングにおける将来のビジネスモデルが描けないというような不安感さえ生じ始めている。

この価格性能比、使いやすさの問題を解決し、スーパーコンピュータのマーケットを拡大するためには、ユーザが使い慣れている Fortran、C 等の逐次型言語で書かれたプログラムを自動的に並列化する自動並列化コンパイラ[4-10,18]の開発が重要となる。

一方、汎用マイクロプロセッサの分野では、我が国の産業界が世界をリードしているとは言い難い状況にあることが認識されている。しかし、従来我が国の電気産業を支える一つの柱となっていた DRAM よる利益確保が将来的に困難と予想される状況では、より付加価値の高い汎用マイクロプロセッサの開発を検討することが必要である。

その際、海外企業が大きなシェアをもち、さらに命令レベルの並列性[1-3]の限界から将来的な実効性能の向上が難しいと予測されるスーパースカラや VLIW ではなく、21 世紀初頭の有力なアーキテクチャの一つとなると考えられるシングルチップ・マルチプロセッサ[2,24]について検討を行うことは、産業界がこの分野で一定のシェアを獲得するために重要と考えられる。また、このようなシングルチップ・マルチプロセッサに関する検討は、上述の HPC の価格性能比向上に対しても重要な役割を果たすものと思われる。

ただ、このようなシングルチップ・マルチプロセッサの研究開発を行う際にも、単に従来の主記憶共有アーキテクチャでプロセッサを集積しただけでは、近隣諸国の技術レベルの向上も著しい現在、世界に対する国産プロセッサの優位性は得られない。21 世紀初頭に十分な利益を上げることを可能とするためにはアーキテクチャの独自性、高性能化、低価格化、低消費電力化等がキーファクタとなる。

このためには、プログラム中の命令レベル並列性、ループ並列性、粗粒度並列性をフルに使用できるマルチグレイン並列処理[4]のように、真に実行すべき命令列からより多くの並列性を抽出し、システムの高価格性能比を達成すると共に、誰にでも使えるユーザフレンドリなシステムの構築を可能とする新しい自動並列化コンパイル技術と、それを生かせるようなアーキテクチャの開発が重要である。

本稿では日本独自の並列化手法であるマルチグレイン並列化コンパイラと、それを産業界と共に実用化するために 2000 年度より開始された経済産業省ミレニアムプロジェクト“アドバンスト並列化コンパイラ（APC)”、及びそのようなコンパイラ技術をサポートするマルチプロセッサアーキテクチャとその実現に向け STARC プロジェクトの一環として 2000 年度に開始された“自動並列化コンパイラ協調型 OSCAR 型シングルチップ・マルチプロセッサアーキテクチャ”について述べる。

2. マルチグレイン並列化コンパイラ

マルチグレイン並列処理[4,8,25]とは、サブルーティン、ループ、基本ブロック間の粗粒度並列性[16]、ループイタレーション間の中粒度並列性（ループ並列性）[5-9,18,19]、ステートメントあるいは命令間の（近）細粒度並列性[11,14]を階層的に利用する並列処理方式である。この方式により、従来の市販マルチプロセッサシステム用自動並列化コンパイラで用いられていたループ並列化、あるいはスーパースカラ、VLIW における命令レベル並列化のような局所的で単一粒度の並列化とは異なり、プログラム全域に渡るグローバルかつ複数粒度によるフレキシブルな並列処理が可能となる。

2.1　粗粒度並列処理

単一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[8,16,17]、OSCAR マルチグレイン Fortran 並列化コンパイラで世界で初めて自動並列化が実現された[4,25]。

OSCAR コンパイラでは、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[7,8]とデータ依存[5,7,8]を解析し、**図 1** 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[7,8]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）[4,8,26]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 2** 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 をプロセッサクラスタ（プロセッサのグループ）への割当てを行う（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コードを Dynamic CP アルゴリズムを用いて生成し、これを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の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。

また、このスタティックスケジューリング及びダイナミックスケジューリングコードの生成の時には、各プロセッサ上のローカルメモリあるいは分散共有メモリを有効に使用し[19-22]、プロセッサ間データ転送量を最小化するためのデータローカライゼーション手法[17,27]も用いられる。

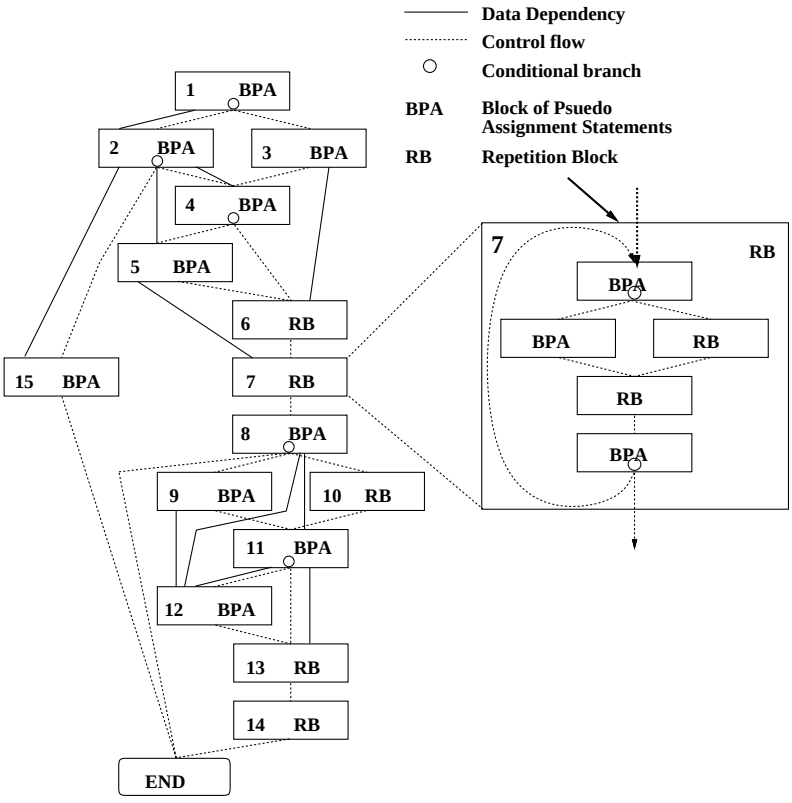


図 1　マクロフローグラフ(MFG)

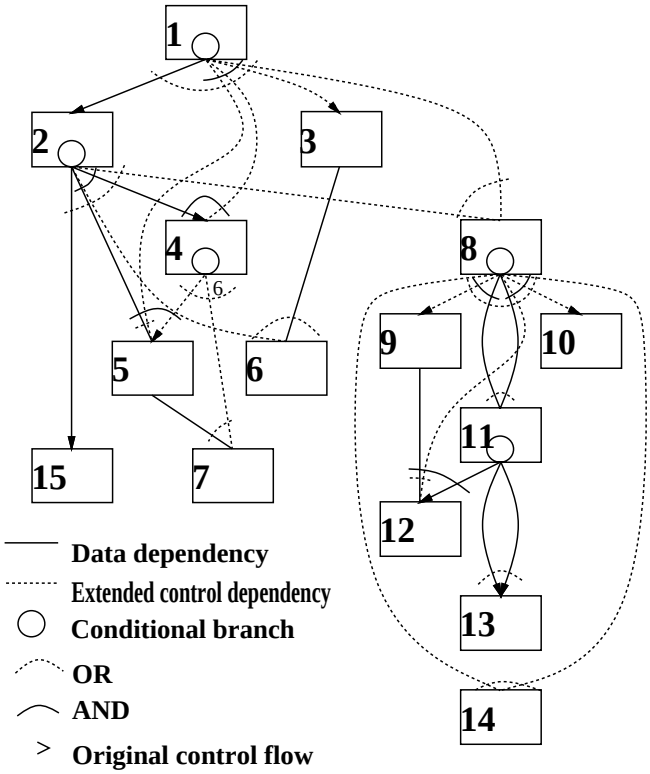


図２　マクロタスクグラフ(MTG)

データローカライゼーションは、MTG 上でデータ依存のあ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)、データ転送が最小になるようにループとデータを分割（ループ整合分割）後、それらのループとデータが同一のプロセッサにスケジューリングされるように指定するパーシャルスタティックスケジューリングアルゴリズムを用いてダイナミックスケジューリングコードを生成する。

またこの際、データローカライゼーションによっても除去できなかったプロセッサ間データ転送を、データ転送とマクロタスク処理をオーバーラップして行うことにより、データ転送オーバーヘッドを隠蔽しようとするプレロード・ポストストアスケジューリングアルゴリズムも使用される[15]。

2.2　ループ並列処理

マルチグレイン並列化では、マクロデータフロー処理によりプロセッサクラスタ（PC）に割当てられるループ（RB）は、その RB が Doall、あるいは Doacross ループの場合 PC 内のプロセッサによりイタレーションレベルで並列処理される。

ループ・リストラクチャリングとしては、以下のような従来の技術をそのまま利用できる。

(a) ステートメントの実行順序の変更
(b) ループディストリビューション
(c) ノードスプリッティング、スカラエクスパンション
(d) ループインターチェンジ
(e) ループアンローリング
(f) ストリップマイニング
(g) アレイプライベタイゼーション
(h) ユニモジュラー変換(ループリバーサル、パーミュテーション、スキューイング)

また、ループ並列化が適用できないループに関しては、**図 3** に示す階層的なマクロタスクの定義のように、ループボディ部を **2.3** で述べる（近）細粒度並列処理か、ボディ部を階層的にマクロタスクに分割しマクロデータフロー処理を適用する。

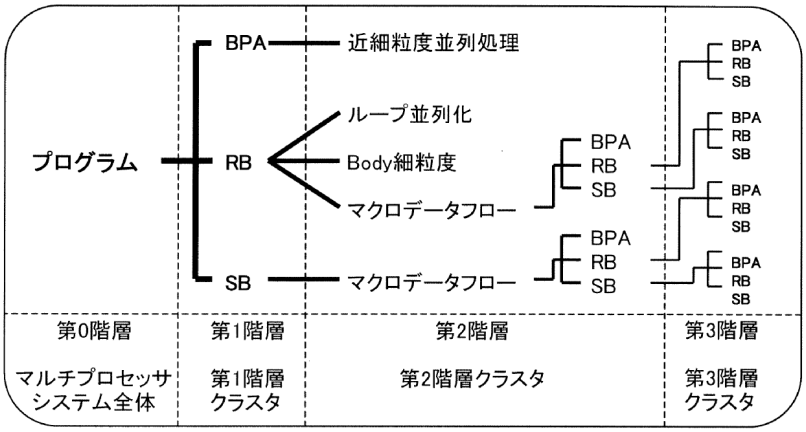


図 3　階層的なマクロタスクの定義

2.3　（近）細粒度並列処理

PC に割当てられる MT が BPA、またはループ並列化あるいは階層的にマクロデータフロー処理を適用できない RB 等の場合には、BPA 内部のステートメントあるいは命令を（近）細粒度タスクとして PC 内プロセッサで並列処理する。

マルチプロセッサシステムあるいはシングルチップ・マルチプロセッサ上での近細粒度並列処理[11,14]では、プロセッサ間の負荷バランスだけでなくプロセッサ間データ転送をも最小にするようにタスクをプロセッサにスケジューリングしなければ、効率良い並列処理は実現できない。さらに、この近細粒度並列処理で要求されるスケジューリングでは、**図 4** に示すようにタスク間にはデータ依存による実行順序制約があるため強 NP 完全な非常に難しいスケジューリング問題[13]となるが、CP/DT/MISF（Critical Path/Data Transfer/Most Immediate Successors First）法等のスタティックヒューリ

スティック・アルゴリズムの使用により実マルチプロセッサシステム上でも効率良い並列処理が行えることが確かめられている。

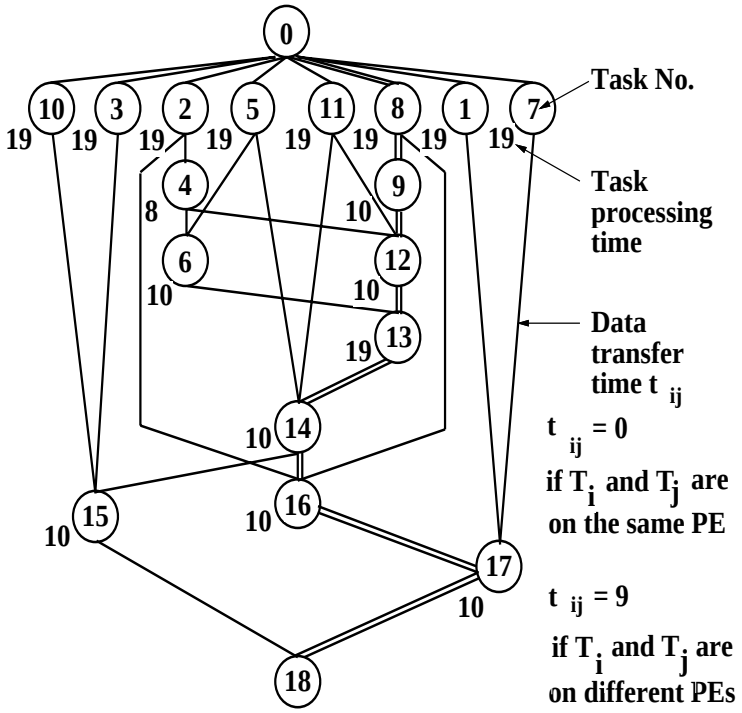


図４　近細粒度タスクグラフ

次にコンパイラは、図５に示すようにタスクスケジューリング、データ分割・配置決定後、各プロセッサで実行される並列化マシンコードあるいは拡張 Fortran 言語等を生成する。各プロセッサで実行されるべき並列マシンコードは、そのプロセッサに割当てられたタスクコード、プロセッサ間でのデータ転送のためのコード、及び同期のためのコード等から構成される。

このマシンコード生成時には、同一プロセッサに割当てられたタスク間でのデータ授受のためのレジスタ利用の最適化、データ転送オーバーヘッド最小化を目指したプロセッサ間データ転送モードの最適化、同期オーバーヘッドの最小化を目指した冗長な同期コードの削除等が行われる。特にシングルチップ・マルチプロセッサでは、コード生成時に厳密なコード実行スケジューリングを行うことにより、実行時のデータ転送タイミングを含めた全ての命令実行をコンパイラが制御し、全ての同期コードを除去して並列実行を可能とする無同期並列化[11]のような究極的な最適化も行える。

このようなマルチグレイン自動並列化の市販主記憶共有型マルチプロセッサシステムシステム上での性能評価を、図５に示した OSCAR マルチグレイン自動並列化コンパイラの OpenMP 並列化プログラム自動生成機能を用いて行った結果を簡単に紹介する。図６は、OSCAR コンパイラが生成した OpenMP コードを IBM 自動並列化コンパイラ

である XL Fortran コンパイラでコンパイルし、IBM RS6000　604e ハイノード 8 プロセッサ SMP 上で実行した時の逐次処理に対するスピードアップ率を表している。図より、OSCAR コンパイラによる Perfect Club ベンチマーク及び SPEC ベンチマークからの 5 本のプログラムに対するスピードアップは、IBM XL Fortran コンパイラによる性能を 2 倍程度上回っており、マルチグレイン並列化技術のポテンシャルの高さを理解することができる[28]。

なお、このように高いポテンシャルを有するマルチグレイン並列化を世界に先駆けて実用化すべく、経済産業省ミレニアムプロジェクトの一環として平成 2000 年度より 3 年間計画で、JIPDEC を中心に日立、富士通、産総研、早稲田大で研究共同体を形成し、電通大、東工大、東邦大の再委託先と共にマルチグレイン自動並列化コンパイラを開発するアドバンスト並列化コンパイラプロジェクトが**図 7** のように開始されている。(http://www.apc.waseda.ac.jp 参照)

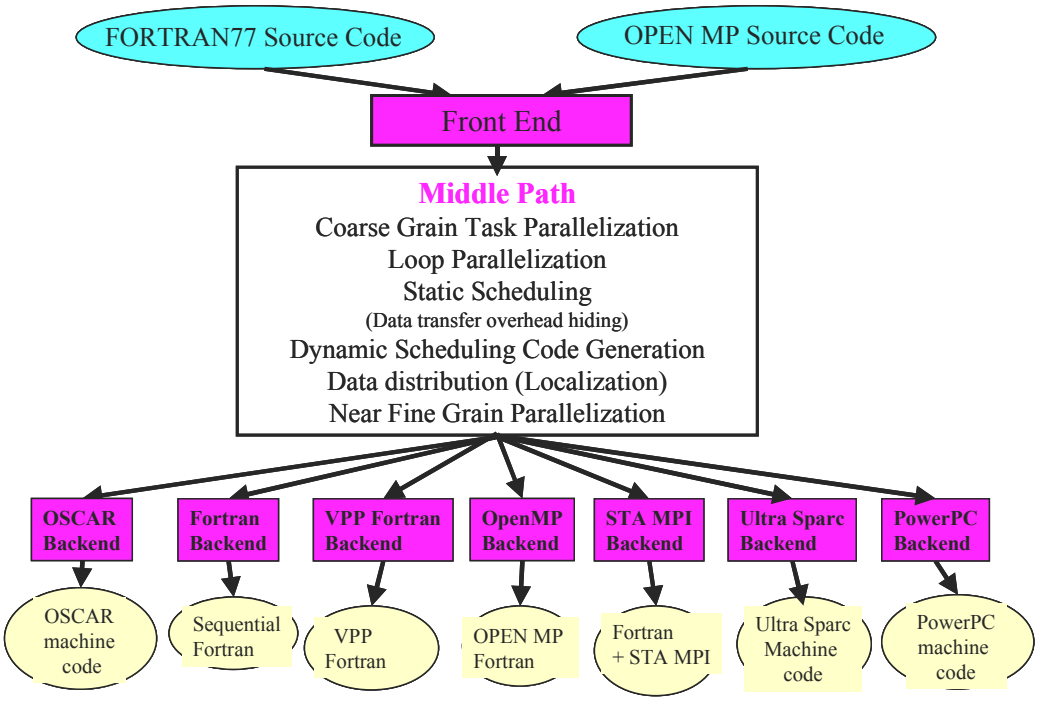


図 5　OSCAR Multigrain Compiler の構成

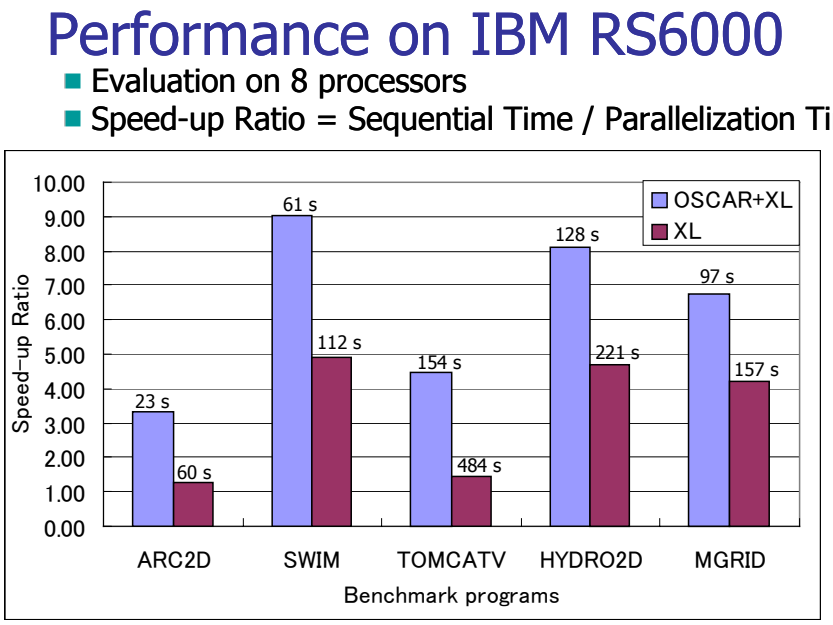


図 6　市販 SMP 上での OSCAR マルチグレイン並列化コンパイラの性能

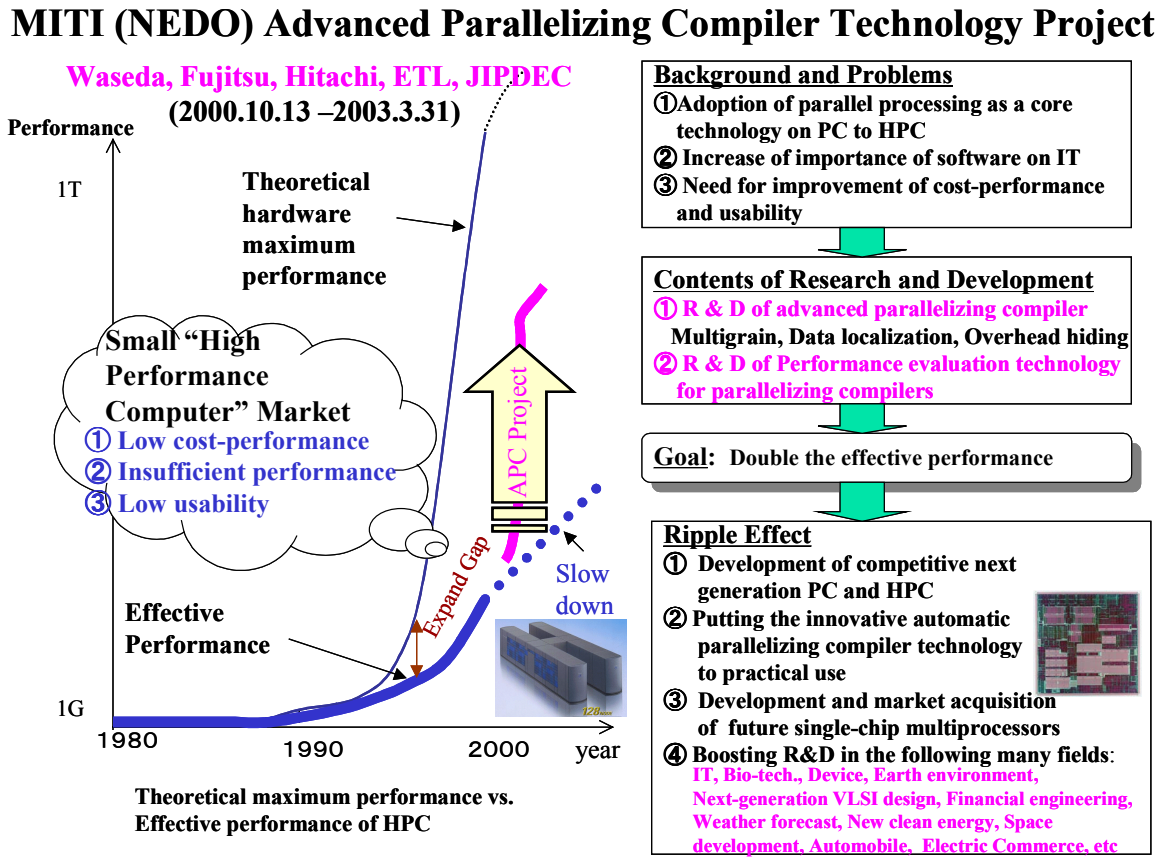


図 7　経済産業省ミレニアムプロジェクト
“アドバンスト並列化コンパイラ”

3. アーキテクチャサポート

2.で述べたマルチグレイン並列処理をマルチプロセッサシステム上で実現しようとする時に望まれるアーキテクチャサポートについて述べる。

3.1 集中・分散共有メモリとローカルメモリ

粗粒度並列処理では、条件分岐に対処するためにダイナミックスケジューリングが使用される。この場合、マクロタスクがどのプロセッサで実行されるかはコンパイル時には分からない。したがって、ダイナミックにスケジューリングされるマクロタスク間の共有データは（集中）共有メモリに配置できることが好ましい。

また、粒度によらずスタティックスケジューリングが適用できる場合には、あるマクロタスクが定義する共有データをどのプロセッサが必要とするかはコンパイル時に分かるため、生産側のプロセッサが消費側のプロセッサ上の分散共有メモリにデータと同期用のフラグを直接書き込めることが好ましい。これにより集中共有メモリを介してデータ転送する場合に比べ同期を含めたオーバーヘッドが 1/2 以下に軽減される。

さらにタスクローカルなデータあるいはデータローカライゼーション手法によりマクロタスク間でローカルにデータの授受が可能な場合には、各プロセッサ上のローカルメモリあるいは分散共有メモリにそれらのデータを割当てることによりデータ転送オーバーヘッドを顕著に削減できる。

また、プログラムコードはデータと比べ小さいことが多く、さらにスタティックスケジューリングを行う場合には各プロセッサ毎に別々のプログラムを生成した法が効率良い並列処理が行えるため、できる限りローカルメモリにおくことが望ましい。

3.2 可変プロセッサクラスタリングと同期

マルチグレイン並列処理では、**図 3** のようなプログラム中の何階層目にどの粒度の並列性がどの程度あるのかによって、階層的なプロセッサクラスタの構成もソフトウェア的に変えられるクロスバのようなネットワークが望ましい。また、それに応じクラスタ内でバリア同期が高速にとれること（可変バリア）が望ましい。

また、近細粒度並列処理においては分散あるいは集中共有メモリ、共有キャッシュ、共有レジスタ等を用いた細粒度データ転送及びデータ同期を高速にとることを可能とするアーキテクチャサポートが必要である。さらに、プロセッサをクラスタリングした場合でも各プロセッサから集中共有メモリへ直接アクセスできることが望ましい。

3.3 処理とデータ転送オーバーラッピング用データ転送ユニット

データローカライゼーションあるいはデータ転送を考慮したスケジューリングで除去できなかったプロセッサ間データ転送は、マクロタスクの実行とオーバーラップして行なわれる。すなわち、スタティックスケジューリングモードでも、ダイナミックスケ

ジューリングモードでもマクロタスクの実行が始まる前にデータが自プロセッサ上のメモリにロードされていることが好ましい。このためには、コンパイラが制御できる高機能なデータ転送ユニットが必要である。

3.4　ソフトウェア制御キャッシュ

マクロデータフロー処理のようなダイナミックスケジューリング環境下での、データローカライゼーション及び高度データ転送ユニットを用いた処理とデータ転送のオーバーラッピングスケジューリング技術の融合は、コンパイラとコンパイラがプログラム情報を埋め込んだダイナミックスケジューリングコードによって、ノーミスヒットのキャッシュの実現にもつながると考えられる。

現在筆者らは、以上のような要求をほぼ満たしているマルチプロセッサ OSCAR をベースに、次世代のスーパーコンピュータアーキテクチャ及び図 8 に示すようなシングルチップ・マルチプロセッサ[23,24]の開発・評価を STARC プロジェクトの一環として行っている[29]。このような、コンパイラと協調して効率的な並列処理を可能とするシングルチップ・マルチプロセッサの性能を近細粒度並列処理を対象に評価した結果、図 9 に示すように microSPARC のような単純なシングルイシュープロセッサ 4 台を OSCAR 型シングルチップ・マルチプロセッサアーキテクチャで 1 チップ上に集積したプロセッサは、ほぼ同程度のトランジスタ数を必要とする 4 イシューの UltraSPARC 程度のスーパースカラプロセッサ 1 台に比べ、2 倍以上の性能向上を可能とすることが確かめられている[29]。

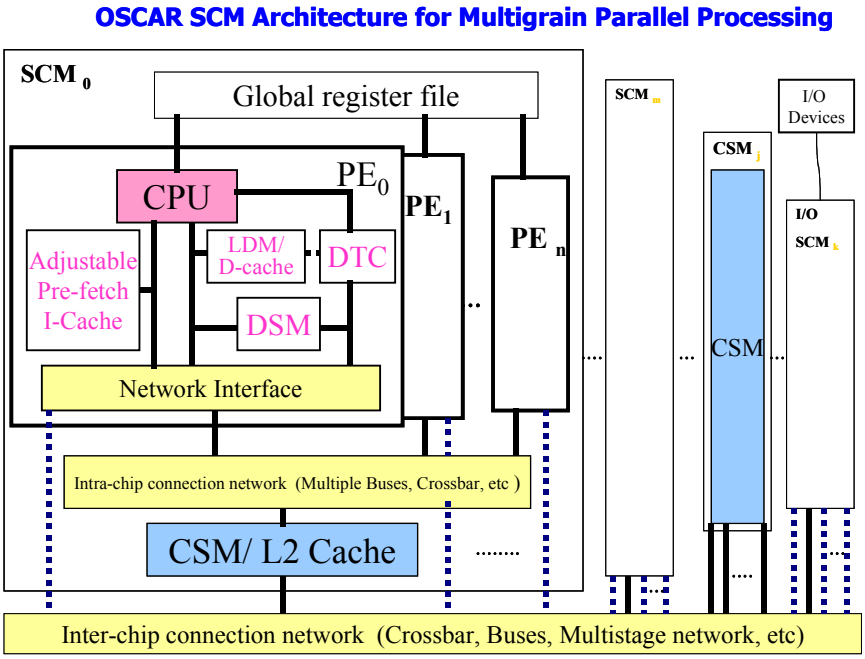


図 8　マルチグレイン並列化をサポートする
OSCAR 型シングルチップ・マルチプロセッサ・アーキテクチャ

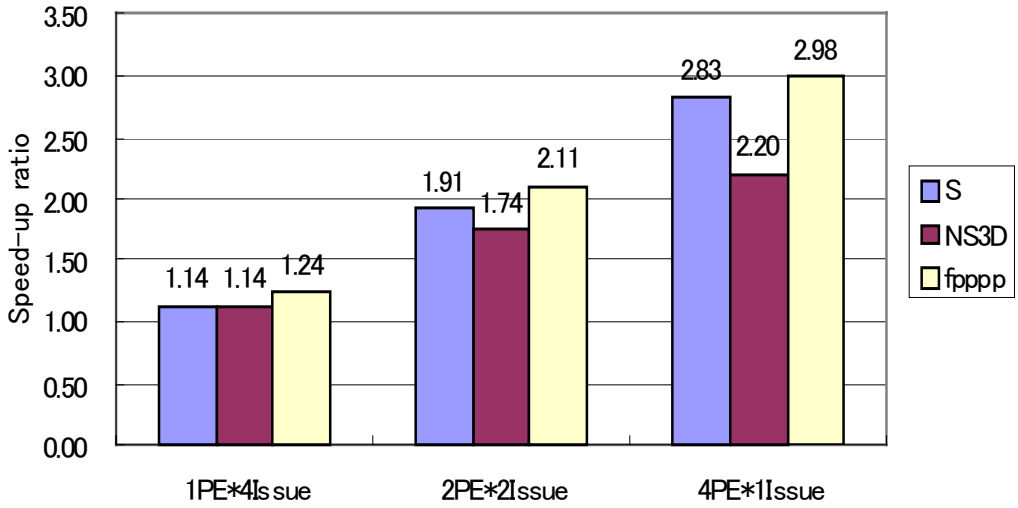


図 9　近細粒度並列処理におけるシングル・イシュー・プロセッサ集積シングルチップ・マルチプロセッサと４イシュースーパースカラプロセッサの性能比較

4. まとめ

本稿では、今後シングルチップ・マルチプロセッサからスーパーコンピュータ、さらには本稿では述べなかったがメタコンピューテイングまで適用可能な、フレキシブルかつ強力な並列処理手法であるマルチグレイン並列処理手法と、そのためのアーキテクチャ・サポートについて述べた。

マルチグレイン並列化は、データローカライゼーション（データ自動分散技術)、データプレロード・ポストストアスケジューリング技術（処理とデータ転送のオーバーラッピング技術）等の技術と共に、マルチプロセッサシステムの実効性能を向上させシステムの価格性能比の向上を可能とするだけではなく、逐次プログラム全域から細粒度から粗粒度に至る全ての並列性を自動的に抽出することを可能とする使い易いマルチプロセッサシステムの構築を可能とする。

このようなマルチグレイン並列化コンパイラとそれを効果的に実現するアーキテクチャの実用化により、ハイパフォーマンスコンピュータの市場を拡大、及びシングルチップ・マルチプロセッサによる汎用マイクロプロセッサ市場への参入を可能とすることを目指している。

参考文献

[1] D.Burger, J.R.Goodman, "Billion Transistor Architectures," IEEE Computer, Vol.30, No.9, pp.47-49, Sep.,1997.

[2] L.Hammond, et.al, "A Single-Chip Multiprocessor", IEEE Computer, Vol.30, No.9, pp.79-85, Sep.,1997.

[3] M.H.Lipasti, J.P.Shen, "Superspeculative Microarchitecture for Beyond AD 2000," IEEE Computer, Vol.30, No.9, Vol.30, No.9,pp59-66, Sep.1997.

[4] H.Kasahara, et.al., "A Multi-Grain Parallelizing Compilation Scheme for OSCAR (OptimallyScheduled Advanced Multiprocessor)," Proc. Languages and Compilers for Parallel Computing, Lect. Notes Comput Sci., Springer-Verlag, Vol.589, 1992.

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

[6] D.J.Lilja, "Exploiting the Parallelism Available in loops," IEEE Computer, pp.13-26, Vol.27, No.2, Feb.1994.

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

[8] 笠原, 並列処理技術,コロナ社, 1991 年.

[9] 笠原,情報処理学会編情報処理ハンドブック第 3 編計算機アーキテクチャ 7 章・並列計算機 6 節・基本ソフトウェア, オーム社,1995 年.

[10] 笠原, “並列処理のためのシステムソフトウェア”, 情報処理 Vol.34, No.9, 1993 年 9 月.

[11] 笠原,“マルチプロセッサシステム上での近細粒度並列処理”,情報処理,Vol.37, No.7, Jul. 1996.

[12] B. Blume, R.Eigenmann, D.Padua, et.al., "Polaris: Then Next Generation on Parallelizing Compilers," 7th Annual Workshop on Languages and Compilers for Parallel Computing, 1993.

[13] H.Kasahara, S.Narita, "Practical Multiprocessor Scheduling Algorithms for Efficient Parallel Processing," IEEE Trans. on Computers, Vol.C-33, No.11, Nov. 1984.

[14] H.Kasahara, H.Honda, et.al., "Parallel processing of Near Fine Grain Tasks Using Static Scheduling on OSCAR (Optimally Scheduled Advanced Multiprocessor)," Proc. of IEEE Supercomputing '90, Nov. 1990.

[15] 藤原,笠原等,"データプレロードおよびポストストアを考慮したマルチプロセッサスケジューリングアルゴリズム", 電子情報通信学会論文誌 D‐1, Vol.75, No.8, pp.495-503, 1992 年 8 月.

[16] 笠原,吉田,本多,合田等,"Fortran マクロデータフロー処理のマクロタスク生成手法

",　電子情報通信学会論文誌Ｄ－１, Vol.75, No.8, 1992 年 8 月.
[17] Yoshida, H.Kasahara, et.al., "Data-Localization for Fortran Macrodataflow Computation Using Partial Static Task Assignment," Proc. ACM Int. Conf. on Supercomputing, May 1996.
[18] U. Banerjee, R. Eigenmann, A. Nicolau, D.Padua, "Automatic program parallelization," Proceedings of IEEE, Vol.81, No.2, pp.211-243, Fev. 1993.
[19] P.Tu and D.Padua, "Automatic Array Privatization," 6th Annual Workshop on Languages and Compilers for Parallel Computing, 1993
[20] M.Gupta and P.Banerjee, "Demonstration of Automatic Data Partitioning Techiniques for Parallelizing Compilers on Multicomputers," IEEE Trans.on Parallel and Ditributed System, Vol.3, No.2, pp.179-193, 1992.
[21] J.M.Anderson amd M.S.Lam, "Global Optimizations for Parallelism and Locality on Scalable Parallel Machines," Proc. of the SIGPLAN '93 Conference on Programming Language Design and Implementation, pp.112-125,1993.
[22] Agrawal, D.A.Kranz, V.Natarajan, "Automatic partitioning of parallel loops and data arrays for distributed shared-memory multiprocessors", IEEE Trans. on Parallel and Distributed System, Vol.6, No.9,pp.943-962, 1995.
[23] 小幡,笠原等,"科学技術計算プログラムにおけるマルチグレイン並列性の評価",情報処理学会第 56 回全国大会, 2E-07,Mar. 1998.
[24] 木村,笠原等,"マルチグレイン並列処理用シングルチップマルチプロセッサアーキテクチャ",情報処理学会第 56 回全国大会, 1N-03,Mar. 1998.
[25] H.Kasahara, et.al, "OSCAR Multigrain Architecture and its performance", Proc. International Workshop on Innovative Architecture for Futer Generation High-Performance Processors and Sytems, IEEE Press, Oct. 1997
[26] 本多,笠原等,"Fortran プログラム粗粒度タスク間の並列性検出手法", 電子情報通信学会論文誌Ｄ－１ Vol.73, No.12, pp.951-960, 1990 年 12 月.
[27] H.Kasahara, A.Yoshida, et.al, "A Data-Localization Compilation Scheme Using Partial Static Task Assignment for Fortran Coarse Grain Parallel Processing, ", Journal on Parallel Computing (Special Issue on Languages and Compilers for Parallel Computing), May. 1998 (to be appeared).
[28] 笠原, 小幡, 石坂, "共有メモリマルチプロセッサシステム上での粗粒度タスク並列処理", 情報処理学会論文誌, Vol. 42, No. 4, Apr., 2001.
[29] 木村, 加藤, 笠原, "近細粒度並列処理用シングルチップマルチプロセッサにおけるプロセッサコアの評価", 情報処理学会論文誌, Vol. 42, No. 4, Apr., 2001.

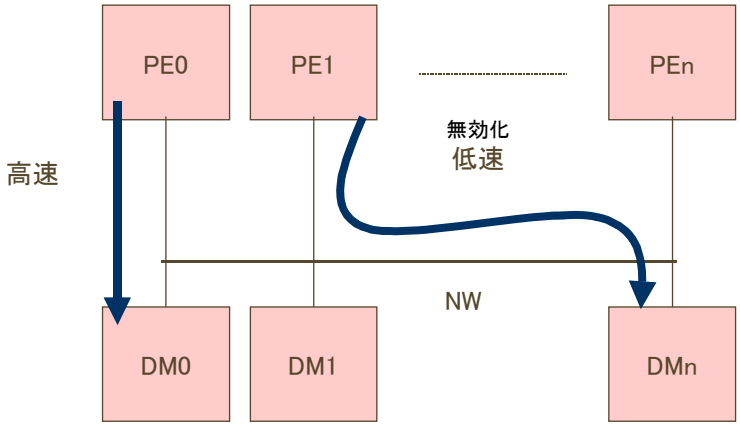
3.4.3　分散共有メモリ向け並列プログラミングインタフェースの動向

妹尾 義樹委員

今後の有力な並列システムである分散共有メモリシステムと、OpenMP を用いた並列プログラミングの実際と問題点、今後の課題について述べる。

1. 分散共有メモリシステムと OpenMP

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、図１に示すように、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。このようなマシンとしては、スタンフォード大学で研究された DASH［1］や、商用機としては SGI の Origin シリーズ［2］、NEC の Cenju-4［3］などがある。



メモリは分散配置しており、自分のプロセッサに直接接続されたプロセッサへのアクセスは高速だが、他のメモリへのアクセスは低速となる。

図1　分散共有メモリ

このような DSM システム上の並列プログラミングインタフェースとしては、OpenMP［4］が有望と目されている。OpenMP は、共有メモリシステム向けの並列化インタフェースであり、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が明示的に指示することにより

並列化を行うことができる。たとえば、ループの並列化の例では以下のように、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することをコンパイラに伝える。

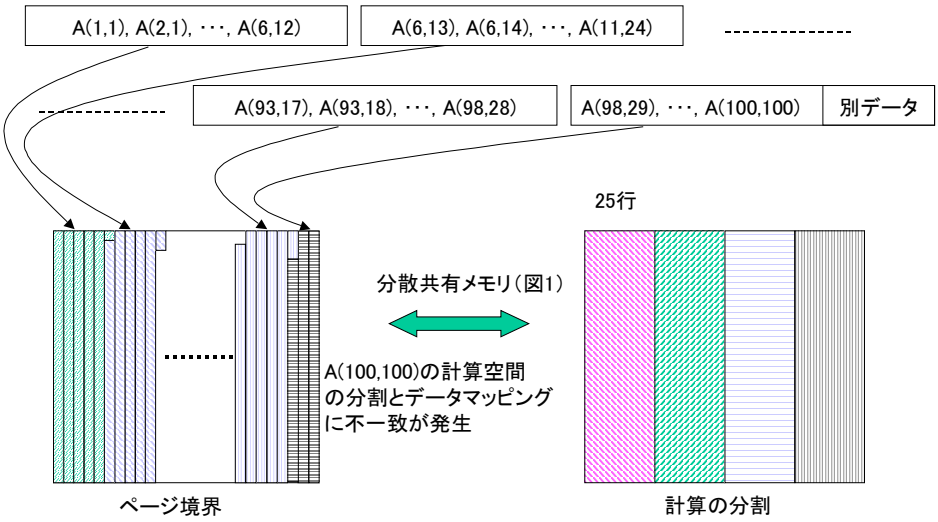
（例 1）OpenMP を用いたループ並列化

```
!$OMP DO
        do i = 1, N
            X(i) = Y（i）  + Z(i)
        enddo
!$OMP END DO
```

ここで注意すべきは、OpenMP の並列化は、計算処理のプロセッサへの振り分けが主眼であり、データを分散メモリ上にどのように配置するかについては、何もユーザが記述する術がない点である。**図 1** に示すとおり、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。データ階層を対象とするコンパイラの最適化については、文献[6]によくまとまっている。

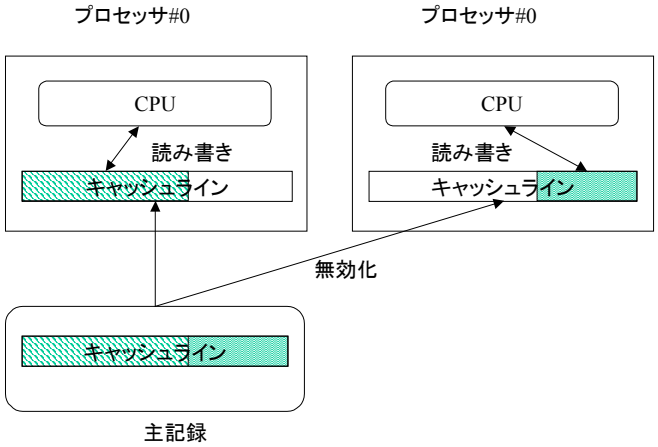
DSM システムでは、一般にデータの（物理的）分散メモリ上への配置は、ページ単位に行われる。このことが 2 つの困難な問題を発生させる。1 つは、データのマッピングがページ単位にしか行えないことであり、2 つ目はこれと関連するが、キャッシュの False Sharing という問題である。

一つ目のページ単位のマッピングという問題は、Fortran プログラムにデータ配置の連続性を仮定したものが多いという問題と関連がある。たとえば、A（100,100）という倍精度（8 バイト）の 2 次元配列をマッピングすることを考えてみる。Fortran では 1 次元目（列）方向が連続に配置されるデータレイアウトになる。ページが仮に 4KB だとすると、倍精度データが 512 要素。これは、配列 A が 5 列と少しということになる。4 プロセッサの実行では、2 次元目で分割し、25 列ずつをプロセッサに割り付けると効率が良いが、ページ単位の分割では、これができない。境界部分では、どちらかのプロセッサメモリに両方のプロセッサがアクセスするデータがページとして配置されることになる（**図 2** 参照）。これを防ぐためには、データ境界部分について、余分な未使用領域を確保し、データの分割サイズとページサイズを合致させることが考えられる。ただし、この場合には論理空間上のデータの連続性が損なわれるため、メモリの連続性を仮定したプログラムは不正動作をすることになる。また、プログラムの並列化の都合で、1 次元目で分割したいこともありうるが、これもデータの連続性を保ったまま分割することはできない。



ページ単位のマッピングでは、任意の配列の計算空間の分割とデータマッピングとのAffinityをいつも満足することができない。どうしても境界部分で不整合が発生する。

図2　配列の計算マッピングとデータマッピング



複数のプロセッサが同一のキャッシュラインをアクセス（読み書き）することにより他方のプロセッサのキャッシュを無効化し合い、極端に性能が劣化することがある。実際の同一データを共有する場合にはやむを得ないが、別々のデータをアクセスしているにもかかわらず対象データが同一キャッシュライン上にあるだけで性能が劣化する。

図3　False Sharing

2 つ目の問題、False Sharing は、キャッシュの一貫性制御の単位であるキャッシュラインを、複数のプロセッサで同時に利用することにより発生する。キャッシュラインは、32 バイトや 64 バイトなどの単位で管理されるが、この 1 ラインの中に複数プロセ

ッサがアクセスするデータの境界が来ると、双方の処理で、たとえデータの受け渡しをする必要がなくとも、キャッシュライン単位でデータの一貫性制御を行うために、双方でデータを更新するたびに、他方が保持するキャッシュデータを消し合うことになる。数値計算では、問題の境界部分の処理を目的として、N の計算領域のために、N+1 のデータ領域を確保することが良くある。このような場合にも False Sharing が発生することになる(**図 3** 参照)。

2. MPI と OpenMP の比較

DSM システム上の並列化インタフェースとしては、多くのマシンが MPI（Message Passing Interface）[7]もサポートしている。MPICH[8]でも、lfshmem というロックフリーの同期機構を用いた実装がフリーソフトとして提供されている。OS や MPI の実装にも依存するが、MPI は SPMD（Single Program Multiple Data Stream）の形式で並列実行が実現されるため、各プロセッサには、プロセスまたはスレッドの上で同一のプログラムがロードされ実行される。各プロセッサが用いるデータは基本的にプロセッサローカルなものだけであるため、それぞれの利用するデータを別ページとして、プロセッサに直接接続されたメモリに貼り付けることが容易である。プロセッサ間のデータの共有は、すべて MPI ライブラリを介して行われるため、Affinity 制御の問題や複数プロセッサによる同一データ（キャッシュライン）参照の問題はすべてライブラリの中に分離することができる。

一方で、MPI はもともと分散メモリマシン用に開発されたインタフェースであり、OpenMP で必要となる計算処理の分割だけでなく、対象データの分割、プロセッサ間のデータ転送、同期などをすべてユーザが明示的に記述する必要がある。それだけプログラミングは複雑になりユーザの負担は飛躍的に増大する。また、分散メモリで動作可能な MPI プログラムを DSM システムで動作させることは、DSM に余計な HW コストをかけて共有メモリを構築する意味が薄れてしまう。

3. DSM 上の Affinity 制御

そこで、DSM 上で OpenMP プログラムを動作させ、さらに Affinity 制御を行う方法がいろいろと検討されている。代表的な手法が、First Touch Method と明示的マッピング指定である。これらは、SGI の Origin システム[9]や、Compaq のシステム[10]で実装されている。

First Touch Method とは、データをメモリ上に確保する際に、実際にデータがアクセスされるまでは、ページの登録だけをしておいて、物理メモリには unmapped の状態にしておく。そして、最初のアクセスに最も適した形でページを配置する（アクセスしたプロセッサのメモリにページを貼り付ける）方法である。たとえば、下記のプログラムを考える。

（例 2）First Touch Method による Affinity 制御

```
!$OMP DO
      DO I=1,N                    ! A の初期化ループ
           A(I)=0.0D0
      ENDDO

       ・・・

!$OMP DO
      DO I=1,N
         ・・・  =  A(I) ＋   ・・・          ! A(I)を用いた計算処理
      ENDDO
    ・・・
```

最初に配列 A(I)の初期化処理が行われ、あとの計算主要部にて A(I)がアクセスされる。この場合、計算処理における複数プロセッサからの A(I)のアクセスパターンが初期化部のそれと同一であれば、初期化時点で、A(I)をアクセスしたプロセッサのメモリに対応するデータをマッピングすれば、計算主要部における Affinity を得ることができる。一般的に、この制御はシステムにより自動的に行うことができるため、ユーザは特に何も指定する必要がない。さらに、計算途中でマッピングを変更できる機能を持つシステムもある。Compaq では、MIGRATE_NEXT_TOUCH という機構により、データマッピングを一旦白紙に戻し、後続のループ処理に Affinity を取る形でデータを再構成することができる。

データの明示的分散配置指定は、HPF（High Performance Fortran）[11]と同様の方法でユーザが陽にデータマッピングを与える方法である。ただし、ページ単位のマッピング（データ配置の連続性を保証する場合）には、厳密にユーザの指定通りに配置されるわけではなく、ページ境界を考えて、できる範囲でのマッピングとなる。データの連続性を保証しなくて良い場合には、メモリ上のデータの詰め替えや非利用エリアでページを埋めるなどして、厳密なユーザ指定に従うこともできる。

参考までに、SGI システムでサポートされている Affinity 制御のための OpenMP の拡張仕様を説明する[9]。

!$SGI DISTRIBUTE

配列に明示的マッピング指示を与える。ただしデータの論理空間上の連続性を保持して配置する。

!$SGI DISTRIBUTE_RESHAPE

配列に明示的マッピング指示を与える。ただし、データの連続性は保持しない。データ配置の連続性を仮定したプログラムでは結果不正を起こす可能性がある。

!$SGI DYNAMIC

配列データのマッピングを変更する可能性があることを指示する。

!$SGI REDISTRIBUTE

実行途中で、データのマッピングを変更する。

!$SGI PAGE_PLACE

HPF でいうところの GEN_BLOCK に相当。配列を任意の数の部分に分割し、それぞれの分割片の先頭アドレス、サイズ、割り付けるべきプロセッサ番号を与え、その分割片を明示的にプロセッサメモリにマッピングする。

!$SGI AFFINITY(idx)=data(array(expr))

DO ループに対する指示であり、ループ空間の計算マッピングに合わせて、指定した配列をマッピングすることを指定。

参考文献[9]には、上記指示を用いた種々の最適化例が掲載されているので、興味のある方は参照されたい。

4. 擬似 SPMD プログラミングによる高速化

Houston 大学の Barbara Chapman らは、ここまでで述べた Affinity 制御によりある程度の高速化は可能だが、16 プロセッサ以上の並列実行では、さらなる最適化が必要だと主張している[12]。もっとも確実に Affinity 制御を実現する方法は、実は先にも述べた通り、MPI によるプログラミングである。データのプロセッサ間での共有は通信部分に隠蔽できる。この考え方を OpenMP プログラミングに適用する。SPMD モデルにおけるそれぞれのローカルデータ領域は、OpenMP の Private 変数（スレッドごとに独立に確保される変数）を用いて表現できる。そして、通信バッファ領域として Shared 変数を確保しておき、プロセッサ間でのデータの共有をバッファエリアとローカルデータ領域のデータ移送部分に隠蔽するのである。**図 4** に擬似 SPMD プログラミングの概要を示す。上の図は通常の OpenMP プログラミングであり、Shared データを全プロセッサがアクセスする。それに比べ、下図では、プロセッサごとに Private 領域を確保し、計算処理では、これだけを用いる。必要に応じて、HPF でいうところの SHADOW 領域と同様の袖領域も確保される。Shared データは、通信バッファ領域だけであり、プロセッサ間のデータのやりとりは、この領域へのアクセスに押し込めることができる。

彼女たちは、本手法を Origin2000 上で、Jacobi 反復などのテストコードを用いて評価し、単なる First Touch 手法による方法に比べて明示的データマッピングを行った方が高速だが、8 プロセッサ程度で高速化が飽和する。これに比べ、擬似 SPMD コーディングを行えば、32 プロセッサまでスケーラブルな性能向上が得られることを示している（データ等は、文献[12]に併記した URL より参照可能）。

このプログラミングテクニックは有効であるが、プログラミングが大変である。共有メモリでの容易な並列化を狙って標準化が進められた OpenMP であるにもかかわらず、

MPI と同じ程度複雑なプログラミングが要求される。プログラムによっては、MPI を用いればトップダウンに手続き処理単位のマクロなレベルで並列化できる場合があるが、OpenMP では、それぞれの DO ループについてボトムアップの並列化対応も必要になるので、こちらがより大変になるケースもある。何らかの言語処理系の支援によって、この SPMD コーディングを容易にする方策が必須である。HPF コンパイラは、逐次コードにデータマッピング指示を加えたものを SPMD コードに自動変換するシステムであるので、ここで蓄積された技術が活かせるのではと期待している。

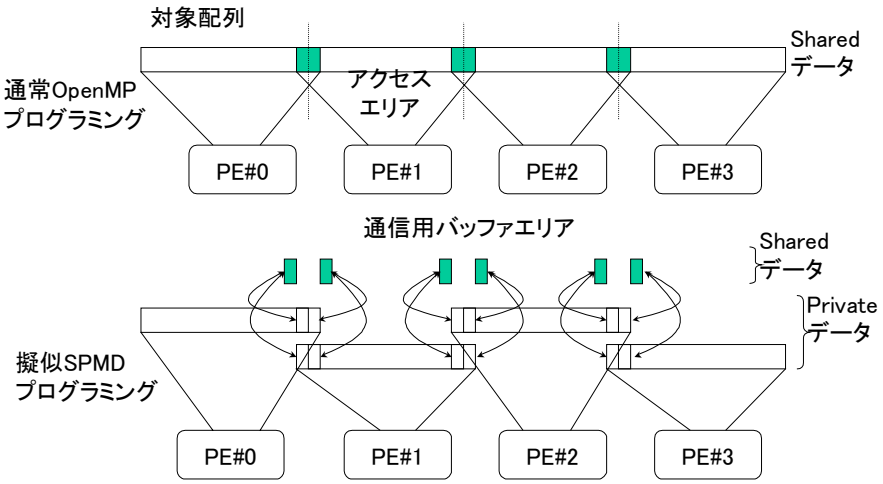


図４　擬似 SPMD プログラミング

5. 言語インタフェース標準化の動向と関連研究

OpenMP は共有メモリ用のデファクト標準を目指して作られた言語であり、シンプルできれいな言語インタフェースである。これに、DSM 向けのデータマッピング指定機能を付加するかどうかが OpenMP の ARB（Architecture Review Board）などで議論されている。DSM という特定のクラスの計算システムだけのために機能付加を行って、言語の簡潔さを損なうことは避けたい。かといって、DSM システムのベンダがそれぞれ独自の指示行を追加していくと、言語の Portability が損なわれる。昨年末から、Barbara Chapman らは HPF の基本機能から OpenMP に付加すべきデータマッピング指示を整理して、OpenMP 拡張仕様案をとりまとめる作業に着手している。また、前節でも述べた通り、DSM ではプロセッサ間で境界データをやりとりする処理が性能ボ

トルネックとなるケースが多い。日本で HPF の拡張機能としてとりまとめられた HPF/JA[13]に SHADOW 通信の明示的最適化機能が含まれるが、これが OpenMP では有効だという議論もなされている。

この他には、日立で、自動データレイアウト機能を用いた、最適 First Touch コードを自動挿入する手法が研究されている[14]。手続き間解析が必須であり、またコンパイル時の静的な情報だけでは難しいケースもあるが、面白い試みだと思う。

また、RWCP の佐藤らのグループが、Omni-OpenMP という処理系を開発し、フリーソフトとして公開している[15]。共有メモリマシンや、クラスタに実装されたソフトウェア DSM 上でもかなり良い性能を達成している。

6. まとめ

DSM というシステムは、メモリを分散配置することで、スケーラブルに接続台数を増やせる HW を実現可能にしつつ、利用者には共有メモリビューを提供するという非常に野心的なシステムである。今後、さらに種々のベンダから DSM システムが商用化されることが予想される。問題は、性能のスケーラビリティーが得られるかどうかであり、プログラミングインタフェース、コンパイラ最適化手法の両面からさらなる研究が期待される。

ある意味で、DSM 上の OpenMP というのは、機能要件として、元の共有メモリ上のインタフェースと MPI との間に位置するものが要求される。この意味では HPF の技術と共通するものが多い。HPF との違いは、HPF が分散メモリを対象にすることで、通信最適化において、正常動作を保証するために、保守的な実装を強制されることがあるが、DSM 上 OpenMP の通信最適化は、最悪でも共有メモリ機能があるので、うまくやれば Optimistic な最適化ができるかもしれない。一方で、OS のジョブスケジューリング機能やページ管理、ページマイグレーション Policy などと整合性をとった最適化が要求されるという点に難しさがある。今後、MPI や HPF、そして OpenMP がそれぞれどのように進化していくのか、あるいは全く別のプログラミングパラダイムが現れるのか、非常に興味深いところである。

参考文献

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

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

[3] 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.

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

[5] T. Faulkner, Performance Implications of Process and Memory Placement using a Multi-Level Parallel Programming Model on the Cray Origin 2000., 1998, http://www.nas.nasa.gov/~faulkner/home.html.

[6] J. Anderson, Data and Computation Transformations for Multiprocessors , In Proc. Fifth ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP '95), Santa Barbara, CA, July 19--21, 1995. http://suif.stanford.edu/papers/anderson95/paper.html

[7] 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. http://www-unix.mcs.anl.gov/mpi/

[8] http://www-unix.mcs.anl.gov/mpi/mpich/indexold.html

[9] Silicon Graphics Inc., MIPSPro Fortran 90 Commands and Directives Reference Manual. Document number 007-3696-003. (http://techpubs.sgi.com/library/　から検索可能)

[10] J. Bircsak, et. al., Extendinf OpenMP for NUMA Machines, In SC2000, Dalls, Texas, November 2000.

[11] 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/

[12] 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

[13] 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).

[14] 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.

[15] 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

3.4.4　科学計算における C++の状況

田中 義一委員

1. はじめに

C++には、科学計算に魅力的な機能がいくつもある。テンプレートを使った汎化プログラミング、記述性の高い演算子の多重定義、コード再利用のためのオブジェクト指向的機能である。しかし、この魅力的機能にもかかわらず性能に大きな問題があるため、科学計算の一般ユーザは、C++を本格的には使用していないように思われる[1]。実際に、C++を使用してプログラムを書くと、FORTRAN に比べ何十倍も遅くなることはよくあることである。この状況は、命令レベル並列方式によって性能をあげるハイエンドプロセッサでは特にひどくなってきている。

本稿では、大きな抽象化ギャップに対するコンパイラによる最適化の困難さと、これを打破するためのアクティブライブラリ構築の試みを、具体的な例に基づいて紹介する。

2. 変数のエリアス関係の増加

一般に、C++で記述すると、生データやポインタをクラスでラップすることが多くなる。例えば、以下のプログラムのように、C や FORTRAN ではデータ dataa などのポインタを直接アクセスしていたが、C++では小さな管理オブジェクト a を経由してアクセスすることが多い。

```
class A{double *dataa;}; class B{double *datab;};   class C{double *datac;   };
void foo( A &a, B &b, C &c, int n){
 for(int i=0;i<n;i++)a.dataa(i)=b.datab(i)+c.datac(i);}
```

性能のキーとなるループ内のロード・ストア命令の数を予想すると、最適化されたコードでは浮動小数点ロード 2 と浮動小数点ストア 1 つである。ところが、実際の C++コンパイラは、dataa, datab,datac フィールド間にエリアスの関係にあることから、書き換えの可能性があると判断する。従って、ループ内にはこれらのフィールドをロードする命令が追加され、ロード命令は 5 つとストア命令が 1 つとなる。データを直接アクセスするのであれば｛ void foo(double *dataa,..) ｝、dataa,datab,datac は別名の関

---

[1] 研究者レベルでは、C++の持つオブジェクト指向プログラミング機能を生かした並列アプリケーション記述の研究が盛んである。例えば、C++の言語仕様を変更することなく C++の持つテンプレートおよびクラス機能を用いて並列処理を実現するアプローチをとる HPC++、同じくテンプレートやクラス機能を用いてアプリケーション寄りのライブラリを提供する POOMA があげられる[1][2]。

係にないため、そのような問題は発生しない。このように、C++で書かれたプログラムは、変数間のエリアス関係を形成しやすいので、冗長なコードや、並列度の低いコードが生成される。

エリアスの問題に対して、最近のコンパイラはコンパイルオプションや言語拡張で対応している。コンパイルオプションの例としては、エリアスに関する条件を緩和するオプションがあげられる。例えば、ポインタの自己参照がないとか、型の異なるポインタ同士は別名の関係にないなどである。すなわち、トリッキーなプログラムでなければ通常は守られているエリアス条件を仮定するコンパイラオプションである[2]。

ここでは、命令数の観点から述べたが、スーパスカラプロセッサや VLIW プロセッサ向けに並列度の高いコードを生成するためには、さらにポインタの指す先がエリアスの関係がないことが必要である。このために、restrict ポインタと呼ばれるポインタ仕様がデファクトスタンダードとなりつつある。

3. 大量の一時オブジェクト

C++では、一時オブジェクトの導入がたびたび必要になる。特に、多重定義演算子を含む式を処理する場合に発生しやすい。一時オブジェクトが導入されると、必要なコンストラクタやデストラクタ呼び出しや、オブジェクト間のコピー演算（関数の引数や返り値渡す）が発生するので性能上問題が多い。インライン展開により、一部のコピーは消去できることもあるが、複雑にネストされたオブジェクト間のエリアスの問題で削除できないものも存在する。コピーコンストラクタが定義されている場合に、名前付返却値（NRV）最適化などにより、関数の返り値に対してコピーの発生が抑止できることがある[3]。

以下に、一時オブジェクトの問題の典型として、STL（Standard Template Library）[4]の開発者である Stepanov が作成したベンチマークプログラムの説明と性能結果を示す。プログラムの内容は、「2000 個の double 型データの総和という単純な処理」を STL 風な記述をしたものである。このベンチマークの中には、大きなデータを扱うプログラムにおける、小さなオブジェクトの典型的な 2 つの扱い方が書かれている。すなわち、たくさんの小さなオブジェクトから構成される大きなデータセット（例えば double 型配列）と、その大きなデータセットのアクセスするための小さなオブジェクト（例えばコンテナとイテレータ）の振る舞いである。ベンチマークは、以下に示す 13 個のループからなる[6]。

[2] また、過去のトリッキーなプログラムの実行は保証せずに、ポインタに関して緩和したエリアス条件を前提としてコードを生成するコンパイラもある。

test0: c スタイルのループ
test1-test12: 関数オブジェクトを使用した STL スタイル記述

```
template<class Iterator, class Number>
Number accumulate(Iterator first, Iterator last, Number result){
  while(first!=last)result = plus(result, *first++);   return result;}
```

test1,3,5,---- データは直接見える double 型変数　　パラメータ Number <-double
test2,4,6,---- データはクラス Double の中の double 型変数（ラッピング）
　　パラメータ Number <- Double
test1: 反復子として通常ポインタ パラメータ Iterator <- *double
test2: 反復子として通常ポインタ パラメータ Iterator <- *Double
test3: 反復子としてクラス double_pointer 型変数（中にポインタ*double を含む）
　　パラメータ Iterator <- double_pointer
test4: 反復子としてクラス Double_pointer 型変数（中にポインタ*Double を含む）
　　パラメータ Iterator <- Double_pointer
..... と test12 までつづく（**図 1** 参照）。

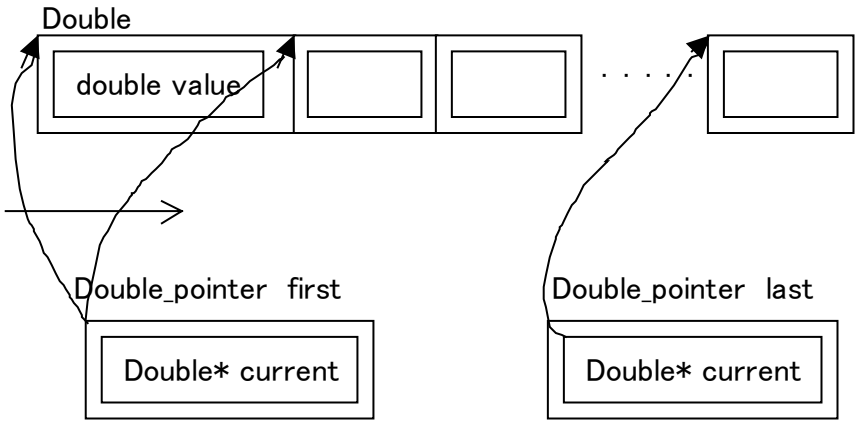


図 1　test4 のデータ構造と test4 のインライン展開後の擬似プログラム

```
accumulate(Double_pointer first, Double_pointer last, Double result)
|while(oprator!=(&first, &last)){          仮引数を p1,p2 と書く
|      |int t1=0
|      |t1 = ! operator=(p1, p2)            仮引数を p21,p22 と書く
|      |      | return (*p21.current==*p22.current)      current は Double*
|      |return t1
|   Double_pointer   t3;
|   次の operator()関数の第３引数を求める関数;
|   operator*( t3=operator++(int)(&first)   ,&t3) operator*の仮引数を p5 と書く
|         |                |Doube_pointer t4;      oprator++関数の仮引数を p3 と書く
|         |                | t4 =*p3;
|         |                | operator++()(&*p3)   仮引数を p4 と書く       前置型＋＋
|         |                |       |++*p4.current          Double_pointer 変数 first の current
|         |                |       |return p4                フィールドの++
|         |                |return t4;
|         | return (*p5).current             *p5     ++前の Double_pointer のコピー
|   result=operator()(&plus, &result,  第３引数   Double*)   第２、第３仮引数を p7,p8
|         |  return operator+(p7,p8)      仮引数を p9,p10
|         |       |double t10=0;
|         |       |Double t11;
|         |       |return Double( &t11,(t10=*p9.value+*p10.value, &t10)),t11
|         |       |             | value=*p11     Double コンストラクタの引数   double *p11
|         |       |             |                  p11 は加算結果
| }
|return result;
```

表 1 に、ある高性能スーパスカラプロセッサでの、３つの異なるコンパイラによる test0-test4 の性能と、ループ 1 回あたりの実行命令数を示す。コンパイラ A は、C++の最適化では世界的に著名なコンパイラである。そのコンパイラでも test0 以外はあまりよい性能とはなっていない。一時オブジェクトは完全に除去されているため、無駄なコードはない。しかし、最近の計算機では、高速性の前提である命令レベル並列方式を生かすソフトウェアパイプラインに基づくコードが生成できなければ、命令数以上の性能差が発生する（本例は、総和と言う依存関係にある処理であるため、その傾向がさらに強くなっている）。このことは、同一周波数の他のスーパスカラプロセッサでも確かめることができた。メーカ製コンパイラを用いた場合、test0 は 714mflops でループ 1 回あたり 2.1 個の命令、test4 は 17mflops で 7.5 個の命令であった、命令数が 3.5 倍に増加しただけで、性能は 1/42 倍というひどい状態になっている。このように抽象度が高いソースプログラムに対して、合格点の最適化コードを生成するには、1 つの見逃しも許されないということを意味している。

表１　Stepanov ベンチマークの性能

	コンパイラA		コンパイラB		コンパイラC	
	性能	命令数	性能	命令数	性能	命令数
test0	556mflops	1.8	625mflops	1.8	31mflops	6
test1	91mflops	5	556mflops	1.8	33mflops	6
test2	91mflops	5	455mflops	2.3	33mflops	6
test3	89mflops	5	21mflops	10	33mflops	7
test4	93mflops	5	11mflops	15	33mflops	7

4. オブジェクト指向モデルと数値計算

C++で書かれた物理シミュレーションプログラムを見ると、以下のようなスタイルで書かれたプログラムを散見することが多い。このプログラム断片は、流体力学のシミュレーションにおいて単純な SOR 法によって圧力を求める部分を表している。ここでは、2 次元セル（i, j）点での残差を求めており、この外側に領域を巡るループがある。

```
Poisson::zansa( int i, int j)
{      double diffx = 0, diffy=0;
      if( grid().cell(i,j).isInsideFluid() )
      {   diffx = grid().P(i-1,j    ) - 2 * grid().P(i,j) + grid().P(i+1,j    );
          diffy = grid().P(i    ,j-1) - 2 * grid().P(i,j) + grid().P(i    ,j+1); }
      else {
         double p = grid().P(i,j);
         if( grid().cell(i,j).isFluid(Cell::DOWN ))    diffy += grid().P(i ,j-1) - p;
         if( grid().cell(i,j).isFluid(Cell::LEFT   ))    diffx += grid().P(i-1,j ) - p;
         ..... }
       return diffx * dx + diffy * dy - so()(i,j);
}
```

この断片から、セル i,j の状態は、すべて i,j の物理量を表すオブジェクトが知っているというモデルで書かれていることがわかる。従って、各セルごとに、「このセルは流体を含んでいますか isInsideFluid」、「このセルの左隣は流体を含んでいますか isFluid(Cell::LEFT)」、「隣のセルの圧力はＰ(i+1,j)」、、、というメンバ関数による問い合わせが発生することになる。このようなプログラムの機械語レベルの構造を見ると、最内側ループは、物理量の属性（計算のための属性）の種類だけ分岐処理を含んだ複雑な処理になる。

スーパコンピュータ向けにプログラムを FORTRAN で書く場合、高速化を望むプログラマは、「最内側ループはできるだけ単純」に書くというプログラム指針に従っていた。単純という意味は、次のループ繰り返し i+1 での処理が、データを読まなくても静

的に予測(線形)[3] できるということにつきる。上記の例題が長方形の領域の場合には、内部領域内と特殊処理の境界を別々のループにして、内部領域内は性能向上が達成できるシンプルなループ構造にするアプローチがとられていた。すなわち、内側ループにある条件文を外側に追い出したことに相当する。また、任意の形状の場合であっても、間接ベクトルを使用して、同様な処理を記述することが行われてきた。逆に、FORTRAN では、徹底的にセル i, j に着目したオブジェクト指向モデルで書くのは煩雑すぎると思われる。これが、FORTRAN がスーパコンピュータと相性がよい理由と考えられる。

上記の例題のような物理問題において、解法に対して素直なオブジェクト指向モデルによって設計されたプログラムを、FORTRAN のごとく書くのは、データと手続きが融合されているので困難と思われる。高速化のためには、物理解法モデルだけでなく、計算の規則性に基づいて計算領域を分割するような、計算の特性に基づくクラス設計を導入する必要がある。

5. 式テンプレート

ベクトル、行列、配列演算が多くの科学計算の基礎である。C++では演算子多重定義により、各クラスの演算をカスタマイズできる。これにより、STL[4]などでは

```
void foo(vector<complex> &y, vector<complex> &x, complex a) {
  y += a*x; }
```

のように、積和ベクトル演算を非常に自然な方法で表現できる。しかし、演算子の多重定義の評価は 2 項演算であるので、上記の式では、まず t1= a*x、次に t2=y+t1 を行い、最後に y=t2 と代入される。ここで、t1,t2 のオブジェクトは中間結果のための一時オブジェクト（ベクトル）である。2 項演算の式評価では、1 つのループで式を評価するのではなく、3 つの分かれたループで評価するためにオーバヘッドが大きい。また、一時オブジェクトは動的に取られるために、さらに性能が悪化する。例えば、上記スタイルと、C による低レベルの記述による性能は、ベクトル長が 1000 の場合、あるスーパスカラプロセッサでは、「ベクトルクラスで記述 333Mflops」「低レベルで記述 640Mflops」であった。

上記の問題は、FORTRAN のアレイ演算でも発生することがある。しかし、通常コンパイラが、FORTRAN からローレベルの内部言語に変換する際に、左辺と右辺でエリアスの可能性がないと判断した時は、1 つのループにすることが一般的である。同様のことを C++で行うことは一般的に困難と考えられる。FORTRAN のアレイ演算は、言語の組み込みであるのに対して、C++ではユーザ（クラスライブラリ）が定義するも

[3] 長方形領域の「右隣のセル圧力の大きさはP(i+1,j)」ですら、ループ内では予測できない。領域は、計算上、一般に内部領域と境界領域に分けられるが、セルが境界点にある場合は境界条件モデルによって、右隣の定義が異なる。

のであるからである。コンパイラのフロントエンドが、各演算子がどのメンバ関数での定義であるかを見つけて、必要なら一時オブジェクトを生成しながら対応する関数を生成する。従って、C++の場合は、一度複数ループを生成してから、再度インライン展開後に隣接するループを融合する方法を取らざるを得ない。しかし、ループの間には、例外を投げうる一時オブジェクトの動的割り当て文の存在もあり、ループ融合ができるかどうかを判定する一般的方式は考えにくい。

上記の問題に対して、blitz クラスライブラリ[5]では、2 項ごとの評価を式テンプレートと呼ばれる技術で解決した。以下では、簡単に double 型のベクトル加算演算だけが可能なクラスライブラリのみを示す。ここでは、明らかなコンストラクタやデストラクタ等や、public/private は削除してある[4]。

```
template<class Op>
   class vector{
      vector & operator = (Vop<Op> const &expr){                    (1)
      for(int k = 0; k != s_; ++k ) a_[k] = expr[k];     return *this; }
   double const& operator[](int k) const { return a_[k]; }
   double& operator[](int k) { return a_[k];}
   double *a_;
  int    s_;
};
   template<class Op>
    class Vop {
       double operator[](int k)    const { return op_[k]; }
       int size() const { return op_.size();}
       Op ope() const { return op_; }
       Op    const    op_;
    };
   template<class Va, class Vb>
   class Vadd{
      double operator[](int k) const {return a_[k] + b_[k];}
      int size() const {return a_.size();}
      Va    const &a_; Vb    const &b_;
 };
   Vop<Vadd<vector,vector > >    operator+(vector const &a, vector const &b){      (2)
      return Vop<Vadd<vector, vector> > (Vadd<vector,vector >(a,b));
   }
   template<class Va>
Vop<Vadd<Va,vector > > operator+(Vop<Va> const &a, vector const &b){          (3)
   return Vop<Vadd<Va, vector> >    (Vadd<Va,vector>(a.ope(),b));
}
```

[4] ここで示したコードは実際の blitz の実装とは異なる。式テンプレートをわかりやすく説明するためのものである[7]。

上記のクラスライブラリが与えられたとき

```
vector x(1000),y(1000),z(1000),w(1000)
x=y+z+w:
```

において、ベクトル演算が効率的に実装されることを示す。まず、y+z を考える。y,z が vector 型なので、(2)の operator+が使われ、Vadd<vector,vector>オブジェクトを格納する Vop<Vadd<vector,vector> >型のオブジェクトが作られる。ここで、この加算は Vop 型の小さなオブジェクトを作ることだけを行う。y+z の評価は先送り（遅延評価）にされている。このオブジェクトは閉包（closure）と呼ばれるものと密接に関係している。同様に (y+z)+w は Vop<Vadd<vector,vector> >型と vector 型の加算なので(3)の operator+が使われ、Vop<Vadd<Vadd<vector,vector>,vector> >型オブジェクトを生成する。すなわち、以下のようにコンパイル時にテンプレートの具現化が行われる。

```
=y+z+w;
= Vop<Vadd<vector,vector> >( Vadd<vector,vector>(y,z))+w
    （yz= Vop<Vadd<vector,vector> >( Vadd<vector,vector>(y,z))とすると）
= Vop< Vadd<Vadd<vector,vector>,vector> >( Vadd<Vadd<vector,vector>,vector>
     (yz.op_ , w))        （=yzw とおく）
```

そして、これが vector に代入されたときに、(1)の代入演算により添え字演算子が以下のように展開され、演算評価が行われる（ここまで評価が遅延された)。このため、従来の vector クラスを使用した時のように、ループが分割されることはなく、C により低レベルで記述した場合と同様な性能が実現できることになる。

```
x.operator=(Vop<Vadd<Vadd<vector,vector>,vector> >   const &expr
    ) for(int k = 0; k != s_; ++k) a_[k] = expr[k]
                                    = yzw.op_[k] = yz.op_[k]+w[k]
                                     =y[k]+z[k]+w[k]
```

6. アクティブライブラリ

ここまで述べてきたように、C++で書かれた抽象化のレイヤを、従来のようなコンパイラの最適化（transformational approach）のみで解決することは難しい。コンパイラはプログラムの意味がわからないので、できることが限られるためである。**5.** で述べた blitz ライブラリ[5]は、抽象化レイヤに対する最適化技術をライブラリで行うことを目的としている。ここでライブラリとは、従来の意味での種種の機能の集積である伝統的なパッシブなライブラリでなく、C++の持つ高度のテンプレート機能により、コン

パイル時に、コードをオプティマイズするアクティブなライブラリである。すなわち、最適化コードはテンプレート技術を使用してライブラリ自身によって生成（generative approach）される。

5.で述べた式テンプレート技術は、評価をする前にいくつかの情報を 1 つの関数にまとめなければならない性質をもつ様様な問題にて応用ができる。

例えば、

```
Array<double, 3> A,B,C,D;
    A=B+C+D;
```

とすると、まず、**5.**で述べたように、式テンプレートが式のパースツリーを生成することで、テンポラリ変数を生成させないで以下のように 3 重ネストループを生成できる。

```
for(int i=0; i< N1; i++)
for(int j=0;j<N2;j++)
for(int k=0;k<N3;k++)
    A(i,j,k)=B(i,j,k)+C(i,j,k)+D(i,j,k);
```

そしてパースツリー操作の際に、blitz ライブラリでは、次のような最適化を行っている。ループ交換、ループ一重化、アンローリング、配列ステンシル（5 点差分、7 点差分など）を検出したあとのタイリング[5] などである。アレイクラスにおいてインデクスをトラバースするイテレータをいくつか用意して、適用しているように思われる。これらの最適化は、高レベルに抽象化されたソースプログラムに適用されるので、汎用性が高い。一方、これらの最適化は、コンパイラでも行われるが、低レベルに変換されたソースプログラムへの適用であるため、汎用性は小さい。

7. おわりに

C++を FORTRAN なみの性能で使うにはまだ障害が多い。今後は、C++のような抽象度の高い言語に対する最適化は、コンパイラではなく、テンプレート機能によるアクティブライブラリによる最適化が有望である。ユーザは、これらの最適化されたライブラリを使って、初めて高速性を享受できる可能性がある。 blitz のようなライブラリが機能拡張[6]，標準化され、商品化されることが望まれる。なお、テンプレートを駆使したアクティブライブラリのソースプログラムを理解することは非常に困難で、一般のユ

[5]空間のトラバースの例として Hilbert curve によるものがあり、これを配列ステンシルに適用すると、大規模プログラムにおいて、キャッシュの再利用性をすこし高める実験結果が得られている。もちろん、タイリングが望ましいが、2 次元までしかまだ対応していないようである。

ーザが自分で構築することは無謀と思われる。また、テンプレートプログラミングの最大の問題は、膨大なテンプレートの具体化のため（部分計算）コンパイル時間が膨大になることである。何らかの方策（プリコンパイルヘッダなど）が考えられないと、実用的にはならないと思われる。

参考文献

[1] http://www.extreme.indiana.edu/hpc++/index.html
[2] http://ww.acl.langl.gov/pooma/
[3] S.B.Lippman：C++オブジェクトモデル，トッパン(1997)
[4] D.R.Musser, and A.Sani: STL Tutorial and Reference Guide, Addison-Wesley (1996).
[5] http://www.oonemerics.org/blitz
[6] http://www.kai.com/benchmarks/stepanov
[7] D. Vandevoorde: C++ Solutions Companion to the C++ Programming Language, Addison-Wesley (1998).

[6] blitz ライブラリは疎行列や、対称行列には対応していないよう。

3.4.5 GCC（GNU コンパイラコレクション）と フリー（自由）ソフトウェア

新部 裕委員

1. GCC（GNU コンパイラコレクション）

GCC は、GNU プロジェクト[1]の開発環境の一つであり、プログラミング言語 C, C++, Fortran を始めとするコンパイラのコレクションである。多くの言語、プラットフォーム、およびターゲットをサポートする優秀なコンパイラであるだけでなく、ソースコードが自由に利用可能であり、開放型の開発体制を持つという特徴を持つ。GCC は、GNU/Linux を始めとする Unix 系の OS の開発環境として用いられるだけでなく、組み込み用途など広く用いられている。

1999 年より、GCC は GNU コンパイラコレクション（GNU CompilerCollection）の略となったが、以前はプログラミング言語 C のコンパイラとして、GNU C コンパイラ（GNU C Compiler）の略という位置付けであった。1999 年に行なわれた開発の統合により、略称の名前はそのままで、その意味が変更された。

以下では、GNU コンパイラコレクションの開発の歴史を概観し、最近の技術的な進展について述べ、開発の体制について説明する。

1.1 GNU コンパイラコレクションの開発の歴史

GCC の開発は GNU プロジェクトの最初の段階の 1986 年に始まった。1986 年のリリースが目標とされていたが、実際の最初のリリースはベータバージョンの 0.9 で、1987 年 3 月である。1.0 のリリースは 1987 年 5 月である。その後、1.x シリーズは 1.42（1992 年 9 月）まで開発が続けられた。1.x シリーズの保守は Richard M. Stallman によって行なわれた。この間、プログラミング言語 C++のサポートが Michael Tiemann により加えられている。

その後、RISC プロセッサのサポートなどが加えられ、2.0 が 1992 年 2 月にリリースされた。2.x シリーズは 2.8.1（1998 年 3 月）まで開発が続けられた。2.x シリーズの保守は、ニューヨーク大学ウルトラコンピュータ研究所の Richard Kenner によって行なわれた。

2.x シリーズの開発の途中の 1997 年、その開発の方式をめぐって、開発者間で合意がとれず、GCC の正式開発から分化する形で、3.0 への実験的試みとして EGCS と呼ばれる活動がはじまった。EGCS は、Experimental/Enhanced GNU Compiler System の略である。

EGCS では、それまで別々に開発されてきた GNU Fortran, C++ランタイムライブラリ、テストスイートを統合したものである。この統合に加えて、新たに IBM Haifa 命

令スケジューラが導入され、エイリアス解析、例外処理、レジスタ割り当てなどに改善が行なわれた。また、UltraSPARC, M32R, H8/S などのプロセッサのポートが加えられた。

EGCS は 1.0（1997 年 12 月）から 1.1.2（1999 年 3 月）までリリースされた。この開発は、EGCS Steering Committee を方向性の意志決定機関とし、各開発者がそれぞれの責任で進めるネットワークを利用した分散開発体制により行なわれた。リリースに関する責任者は EGCS Steering Committee から指名された Jeffery Law であった。

1999 年 4 月に EGCS の成功を受けて、Free Software Foundation は、当時の EGCS の Steering Committee を GCC Steering Committee とし、GCC のオフィシャルメンテナーとした。これによって、その後の GCC のリリースは、GCC Steering Committee に任されることとなった。この時点で、GNU C コンパイラ（GNU C Compiler）という名前を、GNU コンパイラコレクション（GNU Compiler Collection）とした。

GCC Steering Committee は、それまでの GCC 2.x シリーズと EGCS を統合したものを、バージョン 2.95 としてリリースすることとし、引続き Jeffery Law をリリースに関する責任者とした。2.95 は、1999 年 7 月にリリースされた。Jeffery Law は 2.95.2（1999 年 10 月）までのリリースを担当した。2.95.3 のリリース（2001 年 3 月）は、Bernd Schmidt が担当した。2.95 では、VM のバイトコードではなくネイティブの機械語を出力する Java コンパイラが加えられるとともに、新たに Chill という言語のサポートが加えられた。オプティマイズの改善としては、大域共通部分式削除が加えられた。また、SH-4, StrongARM などのポートが加えられた。

GCC Steering Committee は、GCC 3.0 シリーズのリリースの担当者として、Mark Mitchell を指名した。3.0 では、C プリプロセッサが統合され、C++のランタイムライブラリ、Java のランタイムライブラリが統合される。また、GCC のサポートライブラリである libgcc がシェアードライブラリをサポートするターゲットではシェアードライブラリとして提供される。言語としては、Chill のサポートが却下された。また、IA-64, M*Core, AVR, picoJava などのポートが加えられる。3.0 のリリースは 2001 年半ばに予定されている。

1.2　最近の技術的な進展

本節では、1997 年の EGCS 移行に加えられた技術的な進展について述べる。

C 言語, C++言語のサポートは、言語の規格化の進展にともない進められている。言語の変化はそれほど大きくないため、C++については進んでいるといえるが、全体から見るとこの部分は比較的大きな進展ではないといえる。

近年の GCC の開発におけるもっとも大きな進展は、プロジェクトの名前の変更を及ぼすに至った統合であろう。この統合には、言語のフロントエンドの統合、言語のランタイムライブラリの統合、およびテストスイートの統合がある。

また、オプティマイズについても、いくつもの機能が導入され、改善が続けられている。ターゲットアーキテクチャについては 20 を越える新しいアーキテクチャがサポートされた。ターゲット OS のサポートも新しく追加されている。

以下では、それぞれについて詳しく述べる。

1.2.1 これまでの言語サポートの強化

C 言語のサポートとしては、ISO C99 サポートがあげられる。プログラミング言語 C は 1999 年の規格 ISO/IEC 9899:1999 (E) でいくつかの拡張が行なわれたが、GCC の C 言語サポートはこれを追随している。サポートが行なわれた主なものには、以下の 3 つがある。

* for ループ内での宣言のサポート
  for (int i = 0; i < 10; i++) /* ... */という形が有効になった。
* ブーリアン_Bool タイプと<stdbool.h>のサポート
  ブーリアンのタイプが正式にサポートされた。
* _Pragma 演算子のサポート
  プリプロセッサの#pragma ではマクロ展開と組み合わすことができないという欠点があったが、これを補う_Pragma 演算子が導入された。

C++言語のサポートとしては、ISO/ANSI C++のサポートが大幅に改善された。ランタイムライブラリの libstdc++と併せて、C++テンプレートの実装の改善、例外処理の改善が行なわれ、新 C++ ABI が導入された。

C, C++共通のサポートとして、新しい Dwarf2 と呼ばれるデバッグフォーマットがサポートされた。

また、実装として、C プリプロセッサが統合され、一つのプロセスで、プリプロセッサの処理とコンパイルが行なわれるようになった。これにより、カラム単位のエラーメッセージのサポートが可能となった。将来的には、マクロのデバッガサポートなども考えられる。（スタンドアローンの CPP はこれまで通り提供される。）

その他、GCC サポートライブラリの libgcc ががシェアードライブラリをサポートするターゲットではシェアードライブラリとして提供される予定である。これは、例外処理のサポートの際、単一のエントリが必要なためである。

1.2.2 言語システムの統合

EGCS 以前では、GCC の開発は GNU プロジェクトの最も重要な部分であることから、安定した実装が求められ、開発としての機能の追加や拡張が難しいという傾向があった。これが、EGCS 以降、取り込まれていった。

EGCS 以前は、プログラミング言語 C と C++および ObjC がサポートされていた。これに加える形で、追加された言語(フロントエンド、ランタイムライブラリ)には、以下のものがある。

* Fortran
  以前、g77 として GCC とは別に開発されていたものである。コンパイラドライバ g77,フロントエンド f771 と、ランタイムライブラリ libf2c からなる。

* Chill
  CCITT (ITU の前身)で定義された, Modula 2 ファミリの言語である。コンパイラドライバ chill,フロントエンド cc1chill と、ランタイムライブラリ libchill からなる。CCITT High-Level Language の略。ITU の Z.200 として規格がある。
  http://www.itu.int/itudoc/itu-t/rec/z.html
  GCC 2.95 までサポートされたが, GCC 3.0 ではサポートがなくなる予定。

* Java
  Sun Microsystems で開発されたプログラミング言語 Java である。バーチャルマシン(VM)ではなく、ネイティブの機械語を出力するコンパイラである。コンパイラドライバ gcj,フロントエンド jc1 と、ランタイムライブラリ libgcj からなる。

* C++ランタイムライブラリ
  以前は GCC とは別に開発されていたものである。ランタイムライブラリは libstdc++という名前で、ISO/ANSI C++で定義される機能をサポートする。STL も含まれる。
  ここで、ランタイムライブラリが統合されたことが重要である。この統合によって、GCC は単なるコンパイラだけでなく、言語のインフラストラクチャを提供するものとなった。ただし、GCC にはプログラミング言語 C のランタイムライブラリは、含まれていないことに注意すること。（C のランタイムライブラリはシステムに大きく依存し、一般にシステムで提供されるものである。GNU/Linux で用いられる GNU C ライブラリ、組み込み用途に用いられる Newlib は別のソフトウェアとして開発、配布されている。）

言語サポートについては、以上の言語の他に、以下のものが開発されている。これらは、将来 GCC に取り込まれていくものと考えられる。

* ADA
  プログラミング言語 ADA のサポートを行なう GNU Ada Translator (GNAT) という名前のプロジェクトとして実施されている。http://www.gnat.com/

* Mercury
  プログラミング言語 Mercury は論理型/関数型言語であり、メルボルン大学で開発されている。http://www.cs.mu.oz.au/mercury/download/gcc-backend.html

* Pascal
  プログラミング言語 Pascal のサポートとして GNU Pascal Compiler (GPC) がある。http://agnes.dida.physik.uni-essen.de/~gnu-pascal/

* Modula-2
  プログラミング言語 Modula-2 のサポートして GNU Modula-2 がある。
  http://floppsie.comp.glam.ac.uk/Glamorgan/gaius/web/GNUModula2.html

* Cobol
  まだ全くの開発の端緒であるが、Cobol For GCC というプロジェクトがある。
  http://CobolForGCC.sourceforge.net/

1.2.3 テストスイートの統合

テストスイートは基本的には、過去のバグに関するテストケースを集積したものである。例えば、プログラミング言語 C の場合、15000 以上のテストケースが含まれる。

ここで、テストスイートを動かすシステムは GNU プロジェクトの DejaGNU と呼ばれるソフトウェアが用いられる。DejaGNU は POSIX 1003.3 で定義されるテストのフレームワークを実現するソフトウェアである。

改善/拡張を行なった開発者は、テストスイートを実施し、後退（リグレッション）がないことを確認することが推奨されており、リグレッションがないことがパッチの提出の一つの条件となっている。

GCC システムの統合として、テストスイートが GCC に含まれたことは大きな意義を持つ。GCC は時とともにシステムとして大きくなってきたが、大きなシステムの安定性を保つ一つの方策としてテストスイートが有効に働いている。

2000 年 7 月には開発体制が強化され、毎日、自動的に GCC のシステムの構築とテストスイートの実施が行なわれるようになった。そして、テストスイートの実施においてリグレッションが発見されると、該当する可能性がある開発者に対して、注意を換気するメールが自動的に送信される仕組みまで実装された。メールの送信は、ChangeLog のエントリによって行なわれるため、メールの送信はバグを導入してしまった開発者以外の人（同日に変更を行なった開発者）にも送られる。

1.2.4 オプティマイズの改善および拡張

オプティマイズの改善は、さまざまな拡張が行なわれている。以下に、新たに加えられたオプティマイズを列挙する。

* SSA 表現（Static Single Assignment）の導入
  SSA 表現では、それぞれのレジスタは一度しか代入されないという形式のため、レジスタへの代入とその使用の関係が明確になり、ある種のオプティマイズが可能、あるいは簡単になる。
* 大域共通部分式の除去（GCSE），部分冗長除去（Fred Chow's PRE）パスの導入
* IBM Haifa 命令スケジューラの導入
* ソフトウェアパイプラインおよびブランチ最適化の導入

* 基本ブロック順序変換パスの導入
* RTL ベースの末尾再帰呼出の除去（将来は Tree ベースに）の導入
* if 文の変換パスの導入
* 遅延コード移動（Steven Muchnick's LCM: Lazy Code Motion）の導入
* 大域コード上昇（Global Code Hoisting/Unification）パスの導入
* インライン展開の改善
  RTL ベースのインライン展開であったが、Tree ベースのものとなった。コンパイル時間の改善（特にテンプレートを多用する C++のプログラム）、セマンティックスの明確化。
* 大域定数伝搬の改善
* エイリアス解析の改善
* 局所デッドストア除去の改善
* ループの外にロード/ストアを出す改善
* 大域 Null ポインタテスト除去の改善

1.2.5　ターゲットアーキテクチャの拡張

通常の 32-bit アーキテクチャのターゲット他に、IA-64 を始めとする 64-bit アーキテクチャのサポートも増えてきている。AVR というワンチップマイクロコントローラのサポートもなされている。以下に新たに加えられたターゲットアーキテクチャを列挙する。

* Alpha EV6
* Intel Itanium (IA-64)
* Intel XScale
* StrongARM 110, ARM9 (Arm & Thumb)
* NEC V850
* Mitsubishi D30V
* Mitsubishi M32R
* Motorola M*Core
* picoJava
* Matsushita AM33
* Matsushita MN102 & MN103
* Fujitsu FR30
* PowerPC 750
* Sparclite86x
* Hypersparc
* UltraSPARC

* TMS320C[34]x
* Hitachi H8/S
* Hitachi SH-4
* Atmel AVR

1.2.6 ターゲット OS のサポート

以下に新たに加えられたターゲット OS を列挙する。FreeBSD や OpenBSD が追加されたのは、以前の開発体制では GCC の側での正式のサポートがなかったが、EGCS の統合とともに開発が広がったことを示すものであると言える。

* i386-elf
  ELF フォーマットの実行形式のジェネリックなターゲット
* FreeBSD
* HP/UX 11
* Interix
* Irix6.2 & Irix6.3
* OpenBSD
* RTEMS
  組み込み向けリアルタイム OS
* SCO Openserver 5

1.2.7 その他の改善および拡張

その他の改善および拡張について以下に列挙する。

* CPP Makefile の依存関係ルール生成（-MD）の改良
* CFG の計算における Lengauer&Tarjan アルゴリズムの実装
* コントロールフロー解析の書き直し
* SPARC バックエンドの書き直し
* i386 バックエンドの書き直し
* SuperH バックエンドにおける PIC のサポート

まだ取り込まれていないが、今後取り込まれるであろう機能に、Bounded ポインタの導入およびレジスタアロケーションの改善がある。

Bounded ポインタは、ポインタに対し、上限、下限のチェックを行なうものである。コンパイラだけでなく利用するライブラリも変更する必要があるが、細粒度のチェックが可能で、この確認により、多くのバグが見つかっている。

レジスタアロケーションの改善に関しては、ソフトウェアの特許の問題が関係すると言われており、まだ調査段階である。

1.3　開発体制

GCC は当初、Stallman 一人が開発し、保守を行なっていた[2]。その開発体制は、Eric S. Raymond がその論文[3]で Linux と対比して批判したように、中央集権的で必ずしも効率が良いとは言えないものであった。が、EGCS の活動からその開発体制は大きく変わり、現在の体制はかえって Linux よりも整った開放型の開発体制となったと言える。

GCC の現在の開発体制は、GNU プロジェクトの元、Steering Commitee による（方向性の）意志決定、複数のメンテナーによる共同保守、開放型の開発という構成を取っている。

以下では、ミッション、Steering Commitee, メンテナー、および開放型の開発の仕組みについて述べる。

1.3.1　GCC のミッション

GCC は Free Software Foundation が進めている GNU プロジェクトの一つのプロジェクトであり、GNU システムで用いられるコンパイラの開発を行なうプロジェクトである。GCC の開発の統合の際に、下記のようなミッションと方策が明確に定義された（1999-04-22）。

* GNU システムのコンパイラを改善する。
* GNU システム以外のプラットフォームをサポートする。
* 複数のアーキテクチャ、さまざまな環境をサポートする。
* フリーソフトウェアのプロジェクトとして、コンパイラのコピーライトは FSF で GNU GPL の配布条件に従う。その他のライブラリなどは、GNUGPL 以外でも良く、コピーライトの帰属は FSF でなくとも良い。
* 開放型の開発環境とし、協調とコミュニケーションをエンカレッジする。利用者とより緊密に仕事をする。
* コードが誰にでもいつでも利用可能。誰でも開発に参加できる。
* 開放型のメーリングリスト。
* CVS による開発しやすい環境、複数のメンテナーによる査読。

1.3.2　GCC Steering Committee

GCC の開発の方向性は 1998 年に設置された GCC（当時は EGCS）Steering Committee によって決定される。この Committee は、特定の一個人、グループ、あるいは組織が GCC 開発のコントロールを取ってしまうことを阻止する意図の元に設置された。その役割は、ミッションステートメントに掲げられた原則を元にして GCC の開発の意志決定をすることにある。例えば、ブランチ作成、リリース担当の指名などが

Steering Committee により決定されている。
現在のメンバを以下に示す。

* Per Bothner (independent consultant)
* Joe Buck (Synopsys)
* David Edelsohn (IBM)
* Kaveh R. Ghazi
* Torbjorn Granlund (Swox AB)
* Jeffrey A. Law (Red Hat)
* Marc Lehmann (Technische Universitedt Karlsruhe)
* Jason Merrill (Red Hat)
* David Miller (Red Hat)
* Mark Mitchell (CodeSourcery)
* Toon Moene (Koninklijk Nederlands Meteorologisch Instituut)
* Gerald Pfeifer (Vienna University of Technology)
* Joel Sherrill (OAR Corporation)
* Jim Wilson (Red Hat)

1.3.3 メンテナー

メンテナーと呼ばれる保守の担当が決まっている。その担当者の名と連絡先は、gcc/MAINTAINERS というファイルに記述されており、総括的メンテナー13 人、各部分のメンテナー49 人、承認の後書き込みが認められるメンテナー63 人という構成である。

総括的メンテナー13 人の名前を以下に示す。このうち、8 人が Red Hat の所属である。

* Craig Burley
* John Carr
* Richard Earnshaw
* Richard Henderson
* Geoffrey Keating
* Richard Kenner
* Jeff Law
* Jason Merrill
* Michael Meissner
* David S. Miller
* Mark Mitchell
* Bernd Schmidt
* Jim Wilson

総括的メンテナーは、GCC の開発のすべてについて変更する権限を持つ。各部分のメンテナーは担当部分について変更する権限を持つ。承認の後書き込みが認められるメンテナーは、独自の変更の権限はなく、上位のメンテナーの承認の後、変更することができる。

1.3.4　開放型の開発

GCC の開発は広く一般に開放されているものであり、メンテナーだけが開発を行なうのではない。1.4 節で述べるように、開発段階のソースコードも広く一般に開放されており、誰もがアクセスすることができる。改善あるいは拡張を行なう開発者は、自分自身で拡張を行ない、GCC の開発チームに差分（パッチ）を提出することが可能である。

提出されたパッチはグラウンドルール（コピーライトの帰属、コーディングスタイルの遵守など）に従う限り、メンテナーによって査読され、技術的メリットを元にしてその採否が判断される。

1.4　開放型の開発とそれを支えるサービス

GCC は、開発環境として非常に重要な位置を占めるソフトウェアであり、そのユーザと緊密な情報交換があることが望まれる。ソースコードは、開発段階のものが誰でも利用可能となっており、一般からの改善や拡張を受け入れやすくするために、ネットワークを利用した種々のサービスが提供される。

ネットワークを利用したサービスには以下のものがある。

* FTP によるリリースの配布
* FTP による開発段階のスナップショットの提供（毎週）
* CVS による開発段階のソースコードの提供（cvsweb, rsync, CVSup を含む）
* メーリングリストによる情報交換（パッチの提出とその査読を含む）
* 毎日の構築による構築実験とそのバイナリの提供
* テストスイートの実施と該当範囲への連絡
* ベンチマークスイートの実施
* GNATS による PR（Problem Report）の管理

ここで注目すべきは、これらのサービスは主にボランタリベースで行なわれているということである。

以下、CVS のアクセスとメーリングリストについてその詳細を説明する。

1.4.1　CVS による開発段階のソースコードの提供

CVS はネットワークを利用した分散開発を可能とするツールで、特にフリーソフトウェアの開発に広く用いられている[4]。

GCC の CVS は gcc.gnu.org というホストで提供され、使い方は以下の通りである。

$ cvs login -d :pserver:anoncvs@gcc.gnu.org:/cvs/gcc
パスワードが聞かれるので、"anoncvs"と打つ。

GCC のソースコードは:

$ cvs -z 9 -d :pserver:anoncvs@gcc.gnu.org:/cvs/gcc co gcc

GCC のドキュメントは:

$ cvs -z 9 -d :pserver:anoncvs@gcc.gnu.org:/cvs/gcc co wwwdocs

---

WWW でブラウズするインタフェースも提供されており、以下の URL で利用可能である。

http://gcc.gnu.org/cgi-bin/cvsweb.cgi/

ここで、現在、利用可能のブランチは以下の通り:

* 最新のスナップショット: gcc_latest_snapshot
* 特定の日付の版: gcc_ss_yyyymmdd
* 3.0 ブランチ: gcc-3_0-branch
* 2.95.3 リリース: gcc-2_95_3-release
* 2.95.2 リリース: gcc-2_95_2-release
* 2.95 ブランチ: gcc-2_95-branch

1.4.2 メーリングリスト

GCC の関連メーリングリストは、ホスト gcc.gnu.org で運営され、以下のものがある。

* gcc-announce
  アナウンス用（Read Only）
* gcc
  GCC の開発あるいはテストに関する議論
  http://gcc.gnu.org/ml/gcc/
* gcc-help
  GCC の構築あるいは使い方に関するヘルプ
* gcc-bugs
  GCC の bug に関する提出用および議論
  http://gcc.gnu.org/ml/gcc-help/

* gcc-patches
  GCC のパッチの提出および議論
* gcc-testresults
  GCC のテスト結果のレポート
* libstdc++
  standard C++ライブラリの開発に関する議論
  libstdc++に関するパッチはここと gcc-patches の両方に。
* java-announce
  アナウンス（Read Only）
* java
  GCC の Java フロントエンドおよび実行時ライブラリに開発に関する議論
* java-patches
  Java の実行時ライブラリのパッチの提出用
  Java フロントエンドはここと gcc-patches の両方に。
* gcc-prs
  GNATS database の Problem Report
* gcc-cvs
  GCC CVS repository の checkin の情報
* gcc-cvs-wwwdocs
  GCC webpages CVS repository の checkin の情報
* gcc-regression
  GCC compilers の regression 結果
* java-prs
  GNATS database の Problem Reports
* java-cvs
  Java コンパイラおよび runtime の checkin
  gcc-cvs にもメッセージは行く。

2. フリー（自由）ソフトウェアと GNU プロジェクト

フリー（自由）ソフトウェアの運動は、文献[1]に述べられているように、15 年以上の歴史を持っている。今では、さまざまなフリー（自由）ソフトウェアの開発プロジェクトがあるが、その最初は GNU プロジェクトである。GNU プロジェクトは、1984 年に始まった。Unix コンパチのフリー（自由）の OS を作るプロジェクトであり、GCC もその一部である。GNU とは“GNU's NotUnix”の略である。その考え方は、ソフトウェアは、自由に利用可能、改変可能、再配布可能であるべきであり、その自由に制限を加えることは良くないことであるとする。

GNU プロジェクトは、エディタの GNU Emacs,デバッガの GDB,開発環境の GCC

と開発を進めてきたが、OS としては、カーネルが欠けていた。1991 年にカーネルの Linux が出現し、Linux をカーネルとする Unix コンパチの自由の OS,すなわち GNU システムが実現されることとなった。

Linux 出現以後のフリー（自由）ソフトウェアの利用とその開発は大きく広がり、これまでの閉鎖的で独占市場をモデルとするソフトウェアを凌駕するものとなりつつある。

特に、1997 年、Eric S. Raymond が提唱したオープンソースの運動は、Netscape にブラウザソフトウェアの公開を惹起させ、フリー（自由）ソフトウェアの可能性を大きく広げたと言える。ここで Raymond は、フリー（自由）ソフトウェアの運動があまりにも野心的で、ビジネスと相容れない部分があるとし、独自の用語"Open Source"を用いた。フリー（自由）ソフトウェアが、ソフトウェアの自由を第一とするのに対し（しばしばその経済性を無視することもある）、オープンソースはその開放型の開発を重視し、その優秀な機能と性能を誇る。

その後、フリー（自由）ソフトウェアの考え方、オープンソースの開発体制は、大きな発展を遂げている。フリー（自由）ソフトウェア開発に関連したサイトである SourceForge, Freshmeat, GNOME プロジェクト、および GNU プロジェクトの Savannah を見てみると、現在では、**表 1** に示すようにネットワークを利用した大規模の開発が行なわれている。

ここで、SourceForge は開放型の開発をサポートするサイトであり、フリー（自由）ソフトウェアの開発に対して、ネットワークサービスを提供する。Freshmeat はフリー（自由）ソフトウェアのカタログのデータベースを保持するサイトであり、フリー（自由）ソフトウェアの利用者にその利用と開発の情報を提供する。GNOME は GNU プロジェクトの一部で、デスクトップ環境の開発のプロジェクトであり、GNU Savannah は GNU プロジェクトのソフトウェアに対して、開放型の開発をサポートするサービスである。

表 1　フリー（自由）ソフトウェア開発のサイト

サイト名	プロジェクトの数	開発者の数	URL
SourceForge	17,572	138,109	http://sourceforge.net/
Freshmeat	13,229	74,400	http://freshmeat.net/
GNOME	665	不明	http://www.gnome.org/applist/
GNU Savannah	87	378	http://savannah.gnu.org/

特に注目すべきは、その元となった GNU プロジェクトよりと比較にならない程、多くのフリー（自由）ソフトウェアのプロジェクトが実施されていると言うことであり、

その開発者の数が膨大になってきていると言うことである。

今後、これらのプロジェクトから GCC を越えるような素晴らしいソフトウェアが生まれることを期待したい。

参考文献

[1] Richard M. Stallman, "The GNU Project", Open Sources, O'Reilly, 1999.
（http://www.gnu.org/gnu/the-gnu-project.html）で利用可能

[2] Richard M. Stallman, "GNU Manifesto", Dr. Dobb's Journal, 1985.
（http://www.gnu.org/gnu/manifesto.html）で利用可能

[3] Eric S. Raymond, "The Cathedral and the Bazaar", O'Reilly, 1999, 2001.
（http://www.tuxedo.org/~esr/writings/cathedral-bazaar/）で利用可能

[4] Karl Franz Fogel, "Open Source Development with CVS", Coriolis, 2000.

[5] Gary V. Vaughan, Ben Elliston, Tom Tromey and Ian Lance Taylor,
"GNU Autoconf, Automake and Libtool", New Riders, 2000.

[6] Richard M. Stallman, "Using and Porting the GNU Compiler Collection
for GCC 2.95", Free Software Foundation, 1999.

[7] Hans-Peter Nilsson, "Porting GCC for Dunces", 1998.
（ftp://ftp.axis.se/pub/users/hp/pgccfd/）

3.5　シミュレーション関連

3.5.1　並列分散シミュレーション技術とエージェント技術

古市 昌一委員

1. はじめに

異機種分散シミュレータの統合を可能とするため、米国防総省が 1995 年に提案していた HLA（High Level Architecture）が、2000 年 9 月に IEEE により標準仕様 IEEE Std. 1516 となった。これにより、訓練用のシミュレータ同士をネットワークで連接してチーム訓練が容易になると共に、これまでに開発した多数のシミュレータに HLA インタフェースを拡張することにより、新たに開発するシミュレーションシステムの一部として、レガシーなシミュレータを活用することが可能となる。

一方、HLA をハイパフォーマンスコンピューティングにおけるプログラミングモデルと実行環境として活用しようとする研究も一部で行われている。本調査報告では、まず HLA の概説を行った後、このような応用分野の例を 2 例紹介する。

2. HLA の概要

2.1　HLA が標準化されるまでの動き

2.1.1　DMSO による HLA の提案

1990 年代初頭、世界で最もヘビーなシミュレーションシステムユーザである米国防総省では、現場で運用中のシミュレータの種類が数百種を越えて膨大となり、今後の開発・保守コストを低減することが急務であった。そして、1990 年よりこれを解決するための研究開発へ巨額の投資が開始された。

その第 1 段が 1991 年 6 月の DMSO（Defense Modeling and Simulation Office、ディムソーと発音する）設立である。DMSO は、従来は陸・海・空の各軍が独自に実施してきたモデリング＆シミュレーション（以下 M&S）を統合的に扱う機関で、その第 1 の成果は、1994 年 1 月に発行した M&S プロジェクトの憲法的役割を果たす計画書「DoD Modeling and Simulation Master Plan」（DoD Directive 5000.59-P）である。

その後大学や軍の分散シミュレーション研究者を中心に、HLA の仕様を策定するためのチーム AMG（Architecture Management Group）が設立され、その第 1 回会合が 1995 年 3 月に DMSO で実施された。そして 2 ヶ月後の 1995 年 5 月には「HLA -Version 0.0 Interface Specification -」が公開され、続いて 0.1 版が 7 月、0.2 版 が 10 月、そして 1 年後の 1996 年 8 月に HLA 仕様の第 1.0 版が完成した。

2.1.2　IEEE 標準化までの道のり

HLA 仕様の第 1.0 版完成直後の 1996 年 9 月、米国防総省は 2000 年までに HLA 仕様を IEEE 標準とすることと、2001 年以降は HLA に準拠しないシミュレータを調達しないことを宣言した。当時 HLA をベースとした分散シミュレーションシステムは存在せず、大変冒険的な発言であったが、この宣言を契機に研究開発は一気に加速していった。

まず、HLA の IEEE 標準化を推進するために学会 SISO（Simulation Interoperability Standards Organization Inc., http://www.sisostds.org/）が設立され、技術的な議論は SISO が年 2 回開催する SIW（Simulation Interoperability Workshop）で行われるようになった。SIW は毎回 1,000 人以上が参加する大変大きな国際会議で、1 割以上は米国以外からの参加者である。

2.1.3　IEEE による HLA の承認

その後 HLA の仕様は改版を重ね、1998 年 2 月の 1.3 版 Draft 9 が SISO としての HLA 仕様の最終案としてまとまった。これをベースに IEEE 1516 Draft 1（Draft Standard for Modeling and Simulation High Level Architecture,http://www.dmso.mil/index.php?page=121）が 1998 年 4 月に作成され、改訂を重ねて 2000 年 5 月の Draft 5 が、2000 年 9 月に IEEE 標準化委員会により採択された。

2.2　HLA 仕様概説

HLA は、ルール（IEEE 1516）、インタフェース仕様（IEEE 1516.1）、オブジェクトモデルテンプレート仕様（IEEE 1516.2）の 3 つの仕様から成り立っている。

2.2.1　IEEE 1516 ルール

「ルール」は、HLA が規定する範囲を明確にするため、用語の定義やそれぞれの動作の基本を規定したものである。例えば、個々のシミュレータをフェデレートと呼び、その集合である分散シミュレーション全体をフェデレーションと呼ぶ事や、フェデレート間は直接情報を交換することなく、RTI（Run-Time Infrastructure）と呼ぶソフトウェアを介して行う事など、10 種類のルールが規定されている。

2.2.2　IEEE 1516.1 インタフェース(I/F)仕様

「I/F 仕様」は、シミュレータと RTI 間のサービスの仕様を規定したもので、各言語（C++, Java, Ada, CORBA IDL）毎の API（Application Program Interface）を定めている。また、このインタフェース仕様で規定されたサービスを提供するソフトウェアを RTI と呼ぶが、RTI の実装法や通信プロトコルの詳細に関しては規定していない。CORBA 処理系の上に実装した DMSO RTI-NG 1.3 を代表に、スウェーデンの Pitch

社が Java 上に実装した pRTI や、筆者らが TCP/IP Socket 上に開発した eRTI（experimental RTI）等、いくつかの方式による RTI の実装が試みられ、それぞれ性能面やポータビリティ等で特徴を持たせている。

2.2.3　IEEE 1516.2 オブジェクトモデルテンプレート仕様

この仕様は、個々のシミュレータ間でのデータ交換を行う時に基本となる、外部から参照可能なオブジェクトやその属性を定義する SOM（Simulation Object Model）と、分散シミュレーション全体で共通するオブジェクトやその属性を定義する FOM（Federation Object Model）の定義方法を規定する。FOM は、異機種分散シミュレーションにおいて柔軟性のあるデータ交換仕様を実現し、シミュレータの再利用の要となる仕様である。

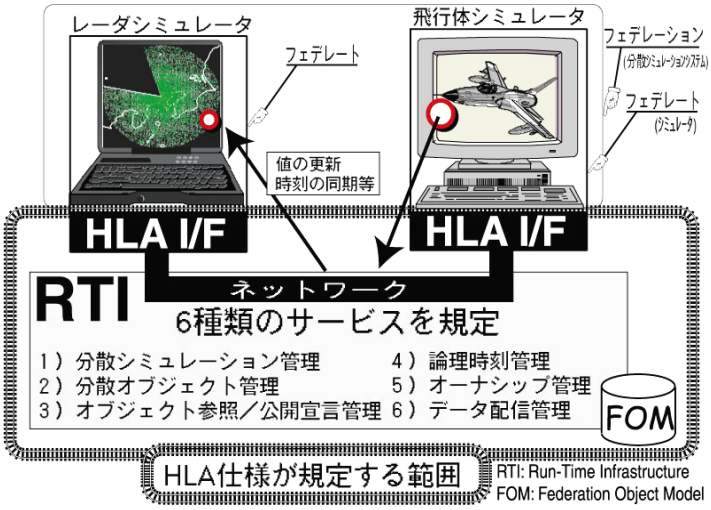


図 1　分散シミュレーション統合アーキテクチャ HLA

2.2.4　HLA 分散シミュレーションの構成

図 1 に、HLA に基づく分散シミュレーションシステムの構成を示す。図に示すように、HLA に準拠した各シミュレータは、I/F 仕様により規定された 6 種類のサービスに属する各 API を使って RTI とメッセージを交換することにより、他のシミュレータとのデータ交換や時刻の同期を行う。

3. ハイパフォーマンスコンピューティングへの応用例

3.1　ジョージア工科大におけるネットワークシミュレータへの応用例

Dr. Fujimoto が率いるジョージア工科大学の Modeling and Simulation Center では、HLA をハイパフォーマンスコンピューティングへ応用するための研究を行っている。

元々 Dr. Fujimoto は並列離散イベント駆動型シミュレーション（PDES: Parallel Discrete Event Simulation）の研究の権威であり、HLA の仕様策定にあたっても中心的なメンバとして活躍してきた。

PDES の研究は様々な大学や企業でこれまで行われてきており、ジョージア工科大学の TimeWarp OS、UCLA の Maisie と PARSEC, NASA の SPEEDES や、三菱電機（株）情報技術総合研究所の OSim 等、様々な開発実行環境が実現されてきた。しかし、これまでは特定のアプリケーションで利用されてきたものの、広いユーザに利用されるようなシミュレーションシステムに適用されることがなく、PDES 自体が一般に広まるまでには至っていない。

これまで普及しなかった理由の一つは、従来の PDES 研究の特徴は、それぞれが独自の開発実行環境を構築し、その上で様々なアプリを開発してその先進性や性能や効果を競っていたからである。このような独自環境の上で作られたシステムは、各ノード（プロセッサ）間の同期と通信プロトコルが独自であるため、あるシステム上に作ったシミュレータを別システムへ移植することが事実上大変困難であることと、各システムの開発環境自体が製品レベルの充実したものが少なく、その結果 PDES の普及はこれまですすまなかったと考えられる。また、キラーアプリに相当するような、広く使われるアプリケーションが生まれてこなかったという問題点もあった。

PDES の研究分野においては、既に研究テーマとして成熟している部分が存在する。例えば、分散シミュレーション全体を管理する方法や、時刻の同期方式の一部等である。このような部分に関しては、必ずしも多くの研究者がそれぞれ独自の方法によるシステムを開発するよりも、仕様の標準化を図ることにより、別々のシステム上で開発したシステム同士のインターオペラビリティを図ることが重要である。

Dr. Fujimoto は、米国防総省が異機種シミュレータ間のインターオペラビリティを実現したいという要望をうまく活用し、HLA 標準仕様の設計とその普及に協力することにより、PDES 研究と実用化の促進を目指したようである。

HLA ベースの PDES システム実用化の第一歩として Dr. Fujimoto が選んだ応用は、シミュレーション技術を応用したアプリケーションとして今後最も需要が増えるだろうと予測される、ネットワークシミュレータである。ネットワークシミュレータの中で現在最も良く使われているのは、パブリックドメインの NS2（Network Simulator, http://www.isi.edu/nsnam/ns/）である。UDP, TCP, RTP 等のプロトコルから、各種のルーティング方式が 1 パッケージの中に実装され、ユーザレベルで自由に拡張することができ、また、拡張機能が更に NS2 にフィードバックされて次のバージョンに盛り込まれることから、ユーザ数を伸ばしているようである。このように、既に充分多数のユーザを持つアプリケーションを対象とし、HLA をベースとした PDES 化を Dr. Fujimoto は選んだ。

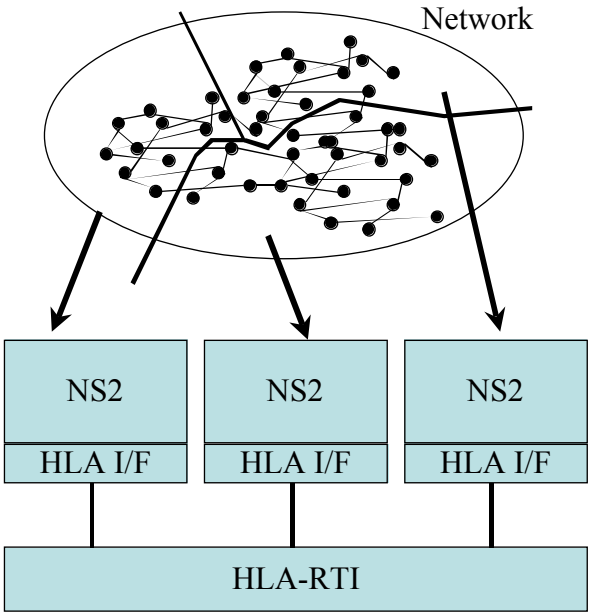


図 2　PDNS の構成概念図

一般的に、既存のシステムを並列化するためには、そのシステムの中で最も性能上のボトルネックとなる部分を切り出して並列化する場合が多い。しかし、この手法で並列化すると、並列化されたシステムへユーザのプログラムを移植する際、プログラムの変更が必要となる場合が多い。

そこで Dr. Fujimoto が採用した方法は、NS2 にはほとんど手を加えず、HLA のインタフェースのみを新たに付加し、多数の NS2 を HLA により連接することにより、全体として一つの PDES システムを構成する方法である。これは、PDNS（Parallel and Distributed Network Simulator）として一般公開されている。PDNS の対象はネットワークのシミュレータであり、ネットワーク分割ツールを用いて多数のサブネットワークを生成し、各サブネットを NS2 にマッピングすることにより並列分散化する。

※PDNS（http://www.cc.gatech.edu/computing/compass/pdns/）

本例は、HLA を利用することにより、既存のシミュレータを組み合わせて大規模な問題を高速に実行可能な分散シミュレーションシステムを、効率良く開発できた代表例であると共に、ユーザ数が多いことから、PDES や HLA を普及させるためのキラーアプリとなる可能性を秘めていると予想する。同様な例が今後も多数現れることにより、更に HLA 及び並列分散シミュレーション技術が普及するであろう。

3.2　ミシガン大学と南カリフォルニア大学におけるエージェント技術への応用例

日本で 80 年代にブームとなった人工知能の研究は、米国のいくつかの大学ではかつての日本以上に研究が活発化している。特に活発なのは、理論的側面からの研究色が強い認知科学の分野ではなく、もう少し工学寄りのエージェン技術の分野である。ここでは、なぜ米国ではこのようにエージェント技術の研究が活性化しているかという理由と、ハイパフォーマンスコンピューティングとの関係に関して説明する。

80 年代から現在に至るまで、人工知能の研究に関する DARPA からの研究資金と米国防総省からの開発資金は年々増額されてきた。

例えば、DARPA の中で先進的な情報技術開発のプロジェクトの計画を策定する機関である DARPA/ITO（Information Technology Office）が現在取り仕切っているプロジェクトの一部をリストアップすると以下の通りとなる。自律協調システムやロボット技術等、システムの自動化に関する技術開発プロジェクトが多数有ることがわかる。

※DARPA/ITO（http://www.darpa.mil/ito/ResearchAreas.html）

Networking & Distributed Systems：Active Networks,Next Generation Internet, Network Modeling and Simulation,Sensor Information Technology, Ubiquitous Computing 等

Embedded & Autonomous Systems：Autonomous Negotiating Teams, Mobile Autonomous Robot S/W,Software for Distributed Robotics 等

Intelligent Software：Communicator,Evolutionary Design of Complex Software, Information Management 等

また、米国防総省では既にエージェント技術に関する研究成果の一部である CGF（Computer Generated Forces）の一部を実用化し、ModSAF（Modular Semi-Automated Force, http://www.modsaf.org/）という名称のソフトウェアとして広く訓練用のシミュレーションシステムのインテリジェント化と半自動化に貢献している。

ModSAF は、軍事訓練に登場するエンティティをモデル化したソフトウェアで、兵器の特性から過去の事例に至るまで詳細にモデル化されており、米軍や NATO 軍における軍事訓練では必ず使われている。モデル化があまりに詳細なため、ソフトウェア自体は輸出規制がされており、我々が一般的に利用することはできない。このように、ModSAF では兵器の物理的特性を中心としたモデル化が行われているが、人間の意思決定の部分に関しては単純にモデル化されているため、実際の訓練で用いる際には、多数の ModSAF を一人の人間が操作することが多い。

一方、意思決定の部分の自動化に関しては、カーネギーメロン大学（CMU）を中心に研究された Soar（the basic cycle of taking a State applying an Operator, And

generating Result）グループを中心に研究が進み、ModSAF と Soar を連接することにより、全自動による知的軍隊模擬（intelligent force）を実現している。

かつて Soar の研究の中心は CMU であったが、現在はミシガン大学アンナーバ校の AI ラボが Soar システムの研究開発とその上のアプリケーションの研究開発の中心となっている。また、南カリフォルニア大学の ISI（Information Sciences Institute）では分散協調エージェントの研究のためのツールとして Soar を用い、ロボカップサッカーやレスキュープロジェクトで活躍している。更に、南カリフォルニア大学の ICT（Institute for Creative Technologies）では、米陸軍とハリウッドの映画製作会社と共同で、Soar を用いた陸軍兵士のミッションリハーサルシステムを実現するための技術開発を行っている。

ここでは、Soar の応用として Soar/IFOR（Intelligent FORce）システムの中の TacAir-Soar を簡単に紹介する。TacAir-Soar は、戦闘機パイロットの訓練用シミュレータのために研究開発されたシステムで、敵側の複数の戦闘機の模擬を ModSAF で行い、敵側パイロットの意思決定部分を Soar により行うものである。TacAir-Soar では、米軍と英国軍が保有する全ての戦闘機とそのパイロットをモデル化しており、約 5,000 種類のルール数と約 500 種類のオペレータから構成されている。おそらく、これまでに開発された人工知能応用システムとしては最も規模の大きいものであろう。

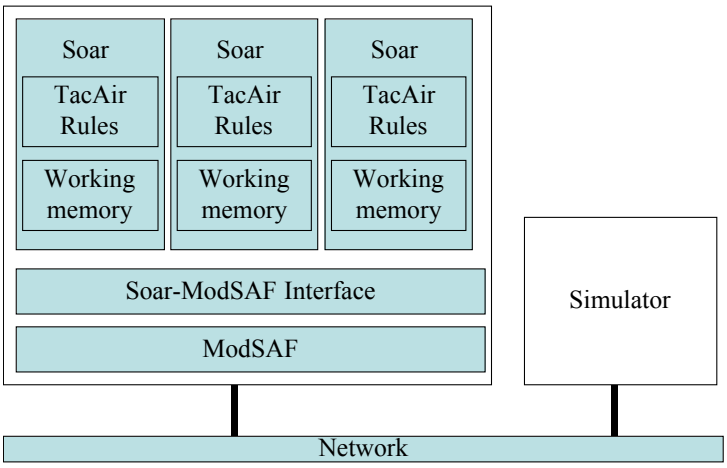


図 3　TacAir-Soar システムの構成概念図

図 3 に TacAir-Soar のシステム構成を示す。図の例では、1 つの ModSAF と 3 つの Soar エンジンが Soar-ModSAF インタフェースを介して連接されて、1 台のワークステーションの上で実行される。最新の ModSAF はまだマルチスレッド化されておらず、Soar に実装された Soar スケジューラにより、疑似的にマルチタスキングされる。訓練生は図中右側のシミュレータを操作し、TacAir-Soar により制御される敵側戦闘機と戦

うことにより戦闘訓練を行う。

HLA 仕様の策定中に開発された TacAir-Soar で用いられた ModSAF は HLA インタフェースを備えておらず、シミュレータとの連接には DIS（Distributed Interactive Simulation）仕様を用いていた。しかし、最新版の ModSAF は HLA 化されており、HLA 準拠のシミュレータから簡単に TacAir-Soar を利用することが可能になる。

TacAir-Soar は既に実際の米空軍パイロットの戦闘訓練で利用されてその効果は実証済みであり、現在は海軍や陸軍でも同様のシステムの開発が進んでいる。特に、陸軍ではミッションリハーサルシステムへの本技術の適用を本格的に開始しており、1999 年には南カリフォルニア大学に 4430 万ドルを投じて ICT（Institute for Creative Technologies）を設立した。ICT では、単に知的エージェント技術やシミュレーション技術の研究だけでなく、映像面でのリアリティを高めるため、ハリウッドの映画製作会社からも多数の人材を招いた共同研究が行われている。ICT の研究内容に関しては、本報告書の付録の海外調査報告に詳しく説明してある。

このように、Soar をベースとした知的エージェントシステムは様々な応用に適用される方向にあり、利用目的が拡大するにつれて、知的エージェント部分の大規模化が促進すると考えられる。しかしながら、現状では Soar の研究者の中には Soar の並列化に興味のある研究者はおらず、Soar をベースとしたハイパフォーマンスコンピューティングの研究は、まだ始まっていないようである。

4. おわりに

HECC のアプリケーション分野の一つとして、HLA をベースとした並列分散シミュレーション技術をとらえ、その中で将来有望と考えられる応用分野の例を 2 例紹介した。

2000 年 9 月に HLA が IEEE 標準となった後、HLA に対する関心がこれまで以上に高まるとともに、軍事訓練用シミュレータの分野では全世界的に普及が始まった。更に、本報告で紹介したような並列分散型のネットワークシミュレータ PDNS の例のように、既に普及しているシミュレータを HLA の特長を利用することによって並列化し、ユーザにとっては HLA を意識することなく性能を向上することができる、という大変良い例が出てきている。同様の例が今後多数出現することが、ハイパフォーマンスコンピュータの需要を高める重要なポイントとなるであろう。

また、米国においては人工知能技術を応用した知的エージェント技術の研究がかつて以上に活性化しており、米陸軍が南カリフォルニア大学に投資して設立した ICT のように、大学とハリウッドの映画業界と軍隊が共同で研究開発を行うまでに至っている。今回調査した範囲では、知的エージェント技術自体を並列処理により高度化・高速化する試みは行われていないようである。しかし、同様の応用が多数出現し、その有効性が示されることによって、ハイパフォーマンスコンピュータの応用として重要な位置を占める可能性を秘めていると予想する。

参考文献

[1] 古市 昌一, 和泉 秀幸, 分散シミュレーションのための統合基盤アーキテクチャ HLA の紹介, 情報処理, 2000.

[2] George F. Riley, Richard M. Fujimoto, Mostafa H. Ammar,, A Generic Framework for Parallelization of Network Simulations, in Proc. of Seventh International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunications Systems, 1999.

[3] Randolph M. Jones, John E. Laird, Paul E. Nielsen, Karen J. Coulter, Patrick Kenny and Frank V. Koss, Automated Intelligent Pilots for Combat Flight Simulation, AI Magazine, 1999.

3.5.2　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

中島 浩委員

ハイエンド・コンピュータのアーキテクチャは進化途上にあり、様々なアイデアが次々と提案されている。このようなアイデアを特に性能面で評価するためには、アーキテクチャレベル（または命令セットレベル）のシミュレータは不可欠のツールである。一方、並列マシンの PE（Processor Element）数増加などシステムの大規模化や、マルチスレッド・プロセッサなどプロセッサの複雑化により、シミュレーションに要する時間は増加の一途をたどっている。したがってシミュレータの高速化技術は緊急の課題であり、様々な提案がなされている。

本節では共有メモリ・マルチプロセッサをターゲットとするシミュレータを中心に、以下の観点から代表的なシミュレータの高速化技術や、最近の研究動向について概観する。

- 高速化の基本技術
- プロセッサの複雑化への対応
- 統合的シミュレーション
- シミュレータの分散・並列化

1. 高速化の基本技術

これまでに提案されている多くの並列マシン・シミュレータは、単一プロセッサの上で動作する。したがって 1 プロセッサあたりの実機／シミュレータの実行時間比である slowdown factor（SF）を S とすると、N プロセッサから成る並列マシンに対する SF は S×N となる。たとえば S=50 のシミュレータは比較的高速な部類に属するが、N =16 であれば SF は 800 となり、実機が 1 分で実行するワークロードに 13 時間以上を要することとなる。したがって高速化技術の基本的な目的は、いかに S を小さくするかにある。

共有メモリ・マルチプロセッサの多くは**図 1** に示すように、トレース駆動型と実行駆動型のいずれかの形態であり、いずれの場合もプロセッサの挙動をシミュレートするメモリ参照履歴を生成するフロントエンドと、それを消費しながらメモリシステムをシミュレートするバックエンドに二分される。各々の形態に固有の高速化技法はもちろんあるが、多くの研究の主眼はいかにフロントエンドの SF を小さくするかにある。逆にいえばバックエンドの高速化はユーザ（アーキテクト）任せであり、研究されていない多くの課題が存在すると思われる。

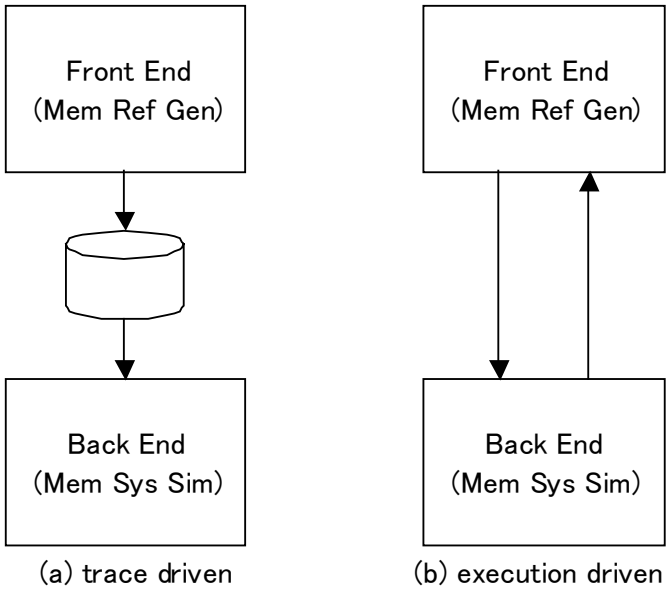


図 1　シミュレーション実行方式

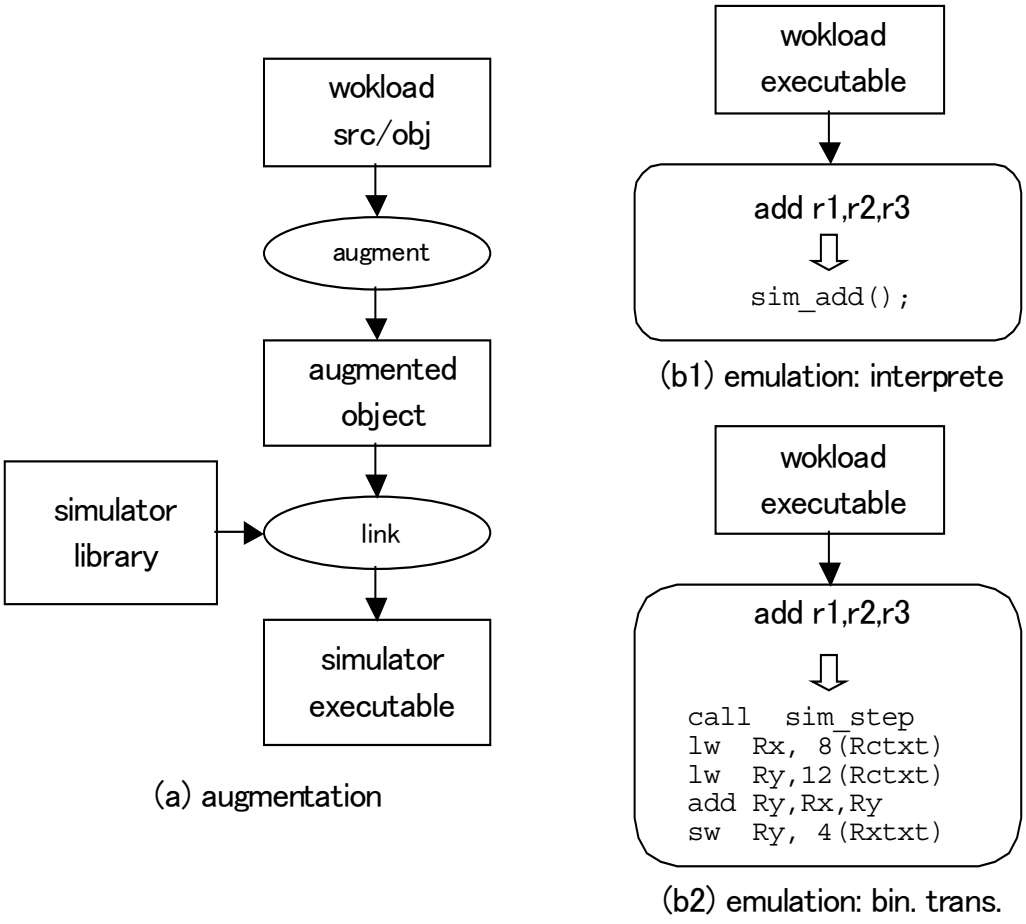


図 2　シミュレーション実行方式(2)

フロントエンドでのプロセッサ・シミュレーションの方式は、**図 2** に示すように Augmentation 方式と Emulation 方式に大別される。前者はワークロード・プログラムのソースあるいはオブジェクトに、メモリ参照履歴生成などのためのコードを付加し、それをシミュレータ・ライブラリとリンクして得られる実行形式を直接実行する。これに対し Emulation 方式ではワークロードの実行形式を入力し、命令の解釈実行あるいは等価な命令列の実行によりシミュレーションを行なう。

表１　代表的なシミュレータとその性能

	T/E	Seq/Para	Mem Sim.	SF
MPTrace [2]	T:aug	P:9～12/9～12	No	1.9～2.6 1000～（w/ backend）
Cerberus [3]	E:aug	Seq	No Yes	
SimICS [4]	E:int	Seq	Yes	50～200
MINT [6]	E:int+bt	Seq	No Yes	18～65 60～200
SimOS [5] (DE) (BT) (DC) (DS)	E:de E:bt E:int E:int	Seq Seq P:4/4 Seq Seq	No L2C L2C Yes Yes	1.1～1.9 9～11 10～36 180～230 3900～6400
RSIM [8]	E:int	Seq	Yes	5000～10000
DirectRSIM [9]	E:aug+int	Seq	Yes	1500～3000
FastILP [10]	T::aug+int	Seq	No	50～100
TDS-NF [11]	T:int	Seq	Yes	1500～3000
SIMCA [12]	E:int	Seq	Yes	40000～50000
WWT [16]	E:de	P:4～64/4～64	Yes	52～250
WWT-II [17]	E:aug	Seq P:32/2～8	Yes Yes	114～241 132～468×16～4
T/E: T … トレース駆動、E … 実行駆動、de … direct execution、 aug … augmentation、 int … interpretation、 bt … binary translation P:n/m … n プロセッサの対象システムを m プロセッサのホストでシミュレート SimOS: DE … direct execution、 BT … binary translation、 DC … Detailed CPU DS … DC + dynamic scheduling				

Augmentation 方式で最高速のトレース駆動型のフロントエンドである MPTrace [2] は、road map と呼ぶ独特の解析情報を用いて、履歴生成のみであれば 2～3、履歴のファイル書込みを含めても 10 程度という、驚異的に小さな SF を達成している。すなわち通常のフロントエンドがメモリアドレスやアクセスタイプの完全な列を直接生成す

るのに対し、MPTrace では完全な列を得るために必要な最小限の情報（たとえばベース・レジスタの値）のみを生成する。road map はこの最小限の情報に基づく補完方法を指示する一種の命令列であり、たとえば履歴中のレジスタ値を参照してメモリアドレスを求める方法が示されている。またワークロードのデータ依存解析を行なうことにより、履歴生成やそのために必要なレジスタ等の退避／復帰を最小限に抑えている。この結果前述のようにフロントエンドの SF は極めて小さいが、逆に履歴を消費するバックエンドでは road map を用いた一種の interpretation が必要となり、バックエンドを含めた SF は（おそらく）1000 以上の大きな値となる。

Augmentation 方式で通常の履歴を生成する高速フロントエンドとしては、Cerberus [3] が挙げられる。Cerberus は実行駆動型であり、MIPS R3000 の 1 命令を 8 命令に展開したコードを直接実行する。この 8 命令の中にはシミュレータ・ライブラリ中の関数呼出が少なくとも一つは存在し、並行実行している対象プロセッサのスケジューリング、キャッシュ・シミュレータの呼出、浮動小数点演算などを行なう。Cerberus のフロントエンドのみの SF は、対象が単一プロセッサの場合は 20～40、マルチプロセッサの場合はプロセッサあたり 40～50 と、比較的高速である。しかし簡単なコヒーレント・キャッシュを付加した場合の SF は 1000～2000 となり、バックエンドあるいはフロントエンドとバックエンドのインタフェースに大きなオーバヘッドがあると思われる。

Emulation 方式の基本的な実行方法は、命令フェッチ、デコード、実行をソフトウェアでシミュレートする interpretation である。この方法は対象プロセッサの動作を最も正確に再現する方法であるが、Augmentation 方式に比べてやや性能が劣るのが普通である。しかし MMU や OS 機能も含めた詳細なシミュレーションが可能な SimICS [4] や SimOS [5]（詳細実行モード）の SF は、コヒーレント・キャッシュのシミュレーションも含め 50～200 であり、Cerberus よりも高速である。

より高速な方式は binary translation と呼ばれ、簡単にいえば Augmentation 方式で静的に展開されたコードを、シミュレーション実行時に動的に展開して実行する方式である。SimOS の binary translation モードでは基本ブロックごとにコード展開を行ない、展開結果を一種のキャッシュに保存する。したがってこの展開コードキャッシュにヒットした基本ブロックは直接実行することができ、簡単なキャッシュのシミュレーションを含めても 10 程度の小さな SF が達成されている。

また、代表的なシミュレータの一つである MINT [6]は、基本ブロック中のメモリ参照を含まない一定の長さ以上の部分命令列に対して binary translation を行ない、それ以外は interpretation を行なうという、二つの方式を混在させた手法を用いている。MINT の SF は履歴生成のみであれば 20～70、コヒーレント・キャッシュのシミュレーションを行なった場合は 100～200 であり、前者は Cerberus と、後者は SimICS や SimOS とほぼ同等である。

2. プロセッサの複雑化への対応

前節で示したSFの値は、いずれも1クロック／命令や単純なパイプライン構造を前提としたシミュレータの性能である。一方、スーパスカラーなどの複雑な構造をもったプロセッサの挙動を忠実にシミュレートしようとすると、より大きなSFとなってしまう。実際、SimICSの動的スケジューリング・モードでは5000程度、代表的なILPプロセッサ・シミュレータであるSimpleScalar [7]では10000程度、また共有メモリ・マルチプロセッサ用のRSIM [8]では20KIPSという性能値[9]から換算して5000～10000という大きな値になっている。

このような低い性能はシミュレーションの精度とのトレードオフであり、精度をある程度犠牲にすればより高い性能を得ることができる。たとえばRSIMを開発したRice大学のグループの報告[9]によれば、スーパスカラー・プロセッサの命令発行レートのピーク値に合わせてプロセッサとL1キャッシュのクロック速度を加速した単純なプロセッサを用いれば、シミュレーション速度を1桁程度改善できる。ただし精度はさほど高くなく、RSIMの結果に対して対象システムの実行時間に30～70%程度の誤差が生じる（増加する）と報告されている。

シミュレーション精度を高めるアプローチのいくつかは、やはりRice大学のグループから提案されている。一つはAugmentationとEmulationを組み合わせたDirectRSIM [9]である。このシミュレータでは命令の「論理的な実行」、すなわち各命令が行なう演算などの操作とそれにより定まる実行パスのトレースをAugmentation方式により行い、「物理的な実行」すなわちメモリ参照命令の実行タイミングを定めるためのILP機構のシミュレーションをEmulation方式により行なう。また後者のタイミング管理を一部簡略化するなどの工夫も行い、誤差を1～2%に留めつつRSIMの3倍程度の高速化を達成している。

もう一つの提案は、シミュレータをシステムの性能モデルへの入力パラメータ生成ツールと考え、モデルを用いた解析に必要なだけの精度のシミュレーションを高速に行なうというものである[10]。この提案の中で用いられているシミュレータFastILPは、DirectRSIMをさらに簡略化し、かつ1プロセッサ分のメモリ参照履歴だけを用いることにより、RSIMの100倍程度（すなわちSFが50～100）の高速化を達成している。また解析の精度は比較的高く、誤差は10%程度と報告されている。

RSIMをベースとした別のグループの提案として、特定の構成によるRSIMの実行結果を用いたトレース駆動シミュレーションを行なうTDS-NF [11]がある。このシミュレータではネットワークの構造だけを変化させることができ、RSIMの3～4倍の高速化と10%以下の誤差が報告されている。

この他、マルチスレッド・プロセッサのシミュレータの性能も大きな問題であり、スーパスカラー用のシミュレータをベースにマルチスレッド化するような方法では、極めて大きなSFとなることが容易に推測できる。たとえばSIMCA [12]の性能は

SimpleScalar の 1/4～1/5 であり、SF は 40000 から 50000 という非常に大きな値となっている。

3. 統合的シミュレーション

多くのシミュレータは、SPEC や SPLASH-2 といったベンチマークなど、単一のアプリケーションを実行した際のシステム性能を示すことを目的としている。したがって OS 機能はシミュレータ内部で直接シミュレートすることが多く、OS の性能や OS 機能がアプリケーションに与える性能上の影響については、一般に高い精度が得られない。

このような状況の中で例外的なものは SimOS であり、IRIX をほぼ完全にシミュレートする機能を持っている。また、SimOS には前述の binary translation、詳細実行、動的スケジューリングの他に、OS の動作の再現だけを目的とした direct execution というモードがあり、単一プロセッサの場合には 2 以下という極めて小さな SF を達成している。このモードは割込やトラップを signal で模倣しつつ、単に OS のコードをユーザ・プログラムとして実行するだけであるので、SF が小さいのは当然ではあるが、性能を細かく解析したいアプリケーションの実行環境設定などの高速化への貢献は大きい。

また、MINT（あるいは他のシミュレータ）をフロントエンドとして用い、共有メモリ・マルチプロセッサの性能と密接に関係する OS 機能である、スレッド・マイグレーション[13]やマルチプロセス環境でのスケジューリング[14]を対象とするシミュレータ構築に関する報告もある。しかし、これらはいずれも対象システムの性能のみを報告しており、シミュレータ自体の性能は明らかにされていない。

4. シミュレータの分散・並列化

マルチプロセッサを対象とするシミュレータの場合、問題には明らかに並列性が内在している。しかしこの並列性を活用して実際にシミュレータを分散・並列化した例は少なく、成功例もごく限られている。この理由の一つは、対象システムの論理構造、特にプロセッサの論理構造が複雑であるため、Time Warp [15]のような一般的な分散・並列離散シミュレーションの技術が適用困難なことにある。Time Warp ではシミュレータ・ノード間の局所時刻のずれにより生じる因果関係の逆転を解消するための巻き戻し処理のためには、クロックごとにプロセッサの状態（あるいはその変化）を保存する必要がある。したがって、ノード間の通信遅延が大きなシステム、たとえば PC クラスタなどでノードあたりの SF を小さくしようとすると、膨大な状態保存が必要となってしまう。また共有メモリ・マルチプロセッサではプロセッサ間通信の頻度が高く、これをノード間通信に直接マッピングすると、致命的な通信オーバヘッドが生じる。たとえば 1GIPS のプロセッサからなる対象システムを、同じ性能のプロセッサを用いて SF が 100 となるようにシミュレーションしようとすると、キャッシュ・ミス率を 10%、メモ

リ参照命令の出現頻度を 25%とした場合、4μs に 1 回の細粒度通信が生じてしまう。

これまで述べたシミュレータのいくつかは、時刻管理の簡略化と共有メモリ・マルチプロセッサの利用により、上記のような問題を回避しながら並列化を行なっている。たとえば MPTrace では、共有メモリ機構の実現をシミュレータ・ホストである共有メモリ・マルチプロセッサに完全に委任し、メモリ参照履歴の生成と保存のみを行なう。したがって対象システムの結合網やメモリでの競合による遅延を考慮したタイミング・シミュレーションは、履歴を解析するバックエンドに委ねられる。また SimOS の binary translation モードは、コヒーレントな 2 次キャッシュの並列シミュレーションが可能であるが、キャッシュミス時の遅延は固定としており、false sharing など複数プロセッサのアクセス競合に起因するキャッシュ動作の非決定性も考慮していない。

厳密な時刻管理を行ないながら分散メモリマシンを用いて共有メモリ・マルチプロセッサ・シミュレータの成功例は、Wisconsin Wind Tunnel (WWT) [16]にほぼ限定される。WWT では、並列離散シミュレーション技法の一つである conservative 法に基づく同期的時刻管理を行なっている。この方法では、対象システムのプロセッサ間（キャッシュ間）の通信遅延の最小値 Q を時刻管理の quantum として用い、シミュレータの時刻が Q だけ進むごとに全てのシミュレータ・ノードで同期を取りつつプロセッサ間通信事象の授受を行なう。この方法は Q が比較的大きく、全ノードでのバリア同期の性能がよい場合には有効であり、WWT では 64PE の CM-5 を用いて 64PE の DASH（Q=100）のシミュレーションを SF=50～100 で実現している。このような高い性能は、CM-5 のバリア同期が高速であることのほかに、対象システムのコードを直接実行すること（direct execution）や、キャッシュミスの検出に ECC トラップを用いていることに負っている。

また、WWT の後継システムである WWT-II [17]は、Augmentation 方式を用いることにより WS クラスタを含めた広範囲のホストに実装可能としている。WWT-II を 67MHz の SuperSPARC を Myrinet で結合したクラスタに実装した例では、32PE の COMA（Q はやはり 100 程度と思われる）の 8 ノードでのシミュレーションにおいて 1200～1500（対象を 8PE とすれば 300 程度）の SF を達成している。しかし WWT(-II) の手法は Q が小さいシステム、たとえばバス結合された SMP を対象とした場合にオーバヘッドが大きく、適用範囲が限定される。

WWT とは全く違うアプローチとして、本報告の筆者らによる Shaman [18]があげられる。Shaman は PC クラスタへの実装を想定し、フロントエンドに複数ノードを、バックエンドには別のノード（当面は単一ノード）を割り当てる。したがってフロントエンドからバックエンドへ、LAN などを介して供給するメモリ参照履歴の量をいかに小さくするかが研究の主題である。Shaman のフロントエンドは Lazy Release Consistency の技法を用いたソフトウェア分散共有メモリにより、対象システムの共有メモリをシミュレートする。その際、対象システムのキャッシュを部分的にシミュレー

トし、対象システムで確実にヒットするメモリ参照を履歴から除去することによって、履歴の大幅な圧縮を行なっている。Shaman は開発途上であるので SF の値は未測定であるが（ノードあたりで 100 程度と予想している）、履歴の削減率が 95%以上であることが確かめられており、高効率の分散シミュレーションが実現できると考えている。

参考文献

[1] 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.

[2] D. K. Keppel, E. J. Koldinger, S. J. Eggers, and H. M. Levy. Techniques for Efficient Inline Tracing on a Shared-Memory Multiprocessor. In Proc. ACM SIGMETRICS Conf. Measurement and Modeling of Computer Systems, 1990.

[3] J. B. Rothman and A. J. Smith. Multiprocessor Memory Reference Generator Using Cerberus. In Proc. MASCOTS'99, pp. 278--287, October 1999.

[4] P. Magnusson and B. Werner. Efficient Memory Simulation in SimICS. In Proc. 28th Annual Simulation Symp., 1995.

[5] 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.

[6] J. E. Veenstra and R. J. Fowler. MINT: A Front End for Efficient Simulation of Shared-Memory Multiprocessors. In Proc. MASCOTS'94, pp. 201--207, February 1994.

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

[8] 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.

[9] 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.

[10] 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.

[11] V. Puente, J. M. Prellezo, C. Izu, J. A. Gregorio, and R. Beivide. A Case Study of Trace-Driven Simulation for Analyzing Interconnection Networks:

cc-NUMAs with ILP Processors. In Proc. 8th Euromicro Workshop on Parallel and Distributed Processing, January 2000.

[12] J. Huang and D. J. Lilja. An Efficient Strategy for Developing a Simulator for a Novel Concurrent Multithreaded Processor Architecture. In Proc. MASCOTS'98, July 1998.

[13] F. Gabbay and A. Mendelson. Smart: An Advanced Shared-Memory Simulator --- Towards a System-Level Simulation Environment. In Proc. MASCOTS'97, January 1997.

[14] H. J. Choi, H. J. Suh, S. T. Jhang, and C. S. Jhon. A Method for an Integrated Simulation Linked with Scheduling Policies on a Program-Driven Simulator. In Proc. MASCOTS'00, August 2000.

[15] D. R. Jefferson. Virtual Time. ACM Trans. Programming Languages and Systems, Vol. 7, No. 3, pp. 404--425, July 1985.

[16] S. K. Reinhardt, M. D. Hill, J. R. Larus, A. R. Lebeck, J. C. Lewis, and D. A. Wood. The Wisconsin Wind Tunnel: Virtual Prototyping of Parallel Computers. In Proc. ACM SIGMETRICS Conf. Measurement and Modeling of Computer Systems, pp. 48--60, May 1993.

[17] S. S. Mukherjee, S. K. Reinhardt, B. Falsafi, M. Litzkow, S. Huss-Lederman, M. D. Hill, J. R. Larus, and D. A. Wood. Wisconsin Wind Tunnel II: A Fast and Portable Parallel Architecture Simulator. In Proc. Workshop on Performance Analysis and Its Impact on Design, June 1997.

[18] S. Imafuku, K. Ohno, and H. Nakashima. Reference Filtering for Distributed Simulation of Shared Memory Multiprocessors. In Proc. 34th Annual Simulation Symp., May 2001. (to appear).

3.6　並列処理／アプリケーション

3.6.1　最適化と並列計算機

福井 義成委員

1. 並列化と矛盾

ここでは将来の高性能計算機(HECC)が並列計算機となることを前提として、HECCに向いたアプリケーションについて考える。並列化は本来、大きな計算（時間がかかる計算）を 1CPU より早く行うことに目的がある。現実的に並列化の効果を期待した問題は、1CPU では１ヶ月以上もかかるような規模である。このような計算を並列計算で実行するときは、各 CPU が負担する計算量は当然、その CPU の能力を十分使いきる程度の大きさになっている。非常に多数（数千？）の CPU による並列計算が実現できたとしても、各 CPU の計算量が数秒程度の問題であっては現実的には意味がない。計算機の有効利用という点からも問題である。人間にとって問題なく待てる時間（状況によって異なる）以内に終了する計算に並列処理を適用する意味はない（ただし、並列化を利用した消費電力の低減の場合を除く)。

並列計算機の性能を発揮させるためは、当然のことながら、並列化が必要である。並列計算は色々な手法があるが、どの方法を適用するにしても、計算の粒度が大きいほど（同期をとる必要が少ないほど)、並列化の効果が大きい。しかし、普通に行われている並列化では１つの問題（計算単位）をいくつか（CPU 数以上）に分割して、分割した各部分（プロセス、スレッド等）を別々の CPU を割り当てて計算をさせることになる。すなわち、並列化自身が計算の粒度を小さくしていることになる（**図１**)。これは自己矛盾である。ここでは CPU 数が増えても、計算の粒度が小さくならないような分野を考える。

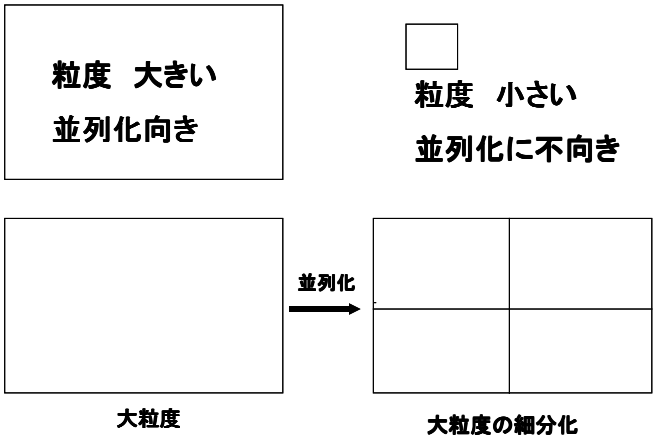


図１　普通の並列化

2. 性能評価

本来の目的に入る前に、性能評価についてふれる。上記のような非常に大規模な問題を並列計算の目的とするのであれば、性能評価も目的にあったものでなければならない。良く見かける並列計算の性能測定は、1 つの固定された問題を分割し、各 CPU に配分して計算している（**図 2**)。本来、並列計算機で解きたい問題は、非常に大きな問題であり、CPU 数が増えても、なおかつ各 CPU が分担する計算が十分に大きなものであるはずである。CPU 数が増えた場合、1CPU が分担する演算量を固定で、CPU 数が増えるにしたがい、計算可能な問題が大きくなるというような問題（**図 3**）での評価が重要である。

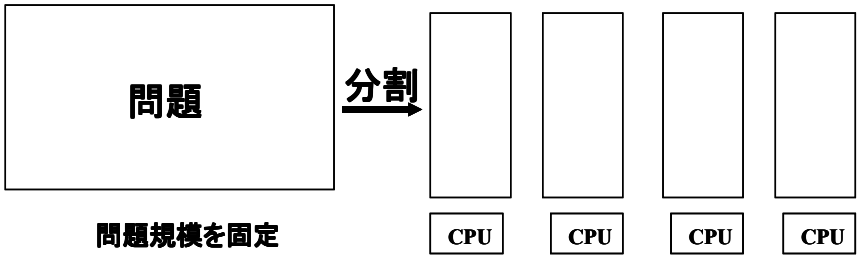


図 2　問題規模を固定し、並列化する場合

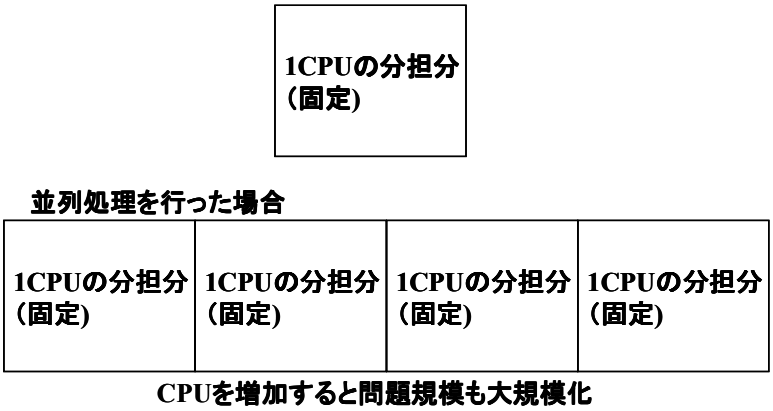


図 3　1CPU が行なう計算規模を固定した場合

3. 問題の粒度を小さくしないアプリケーション

普通に並列化を行えば、問題の粒度が小さくなる。しかし、同じ問題で含まれるパラメータの値だけが異なる問題を別々の CPU で計算すれば、問題の粒度を小さくしなくて済む。これは良く知られている「自明な並列化」である。「自明な並列化」は並列化

の研究者にとっては、興味がないものであろうが、実用的には重要な方法である。研究者にとって面白い対象と世の中で必要とされる対象は必ずしも一致するとは限らない。

「自明な並列化」を行う主な目的は

(1) パラメトリック・スタディ
(2) 最適化
(3) 許容範囲（歩留まり）の探索

等がある。この 3 つは相互に関係している。最適化の場合、製品の開発において、最適化を行い、最高の品質と低コストの製品を作るのが目的である（製品の性能を表す目的関数の値が小さいほど、製品の性能が良いとする)。しかし、最適な設計点が求まったとしても、実際の製品の製造には、バラツキがつきものであり、1 点だけの最適化では製造の役には立たない（**図 4**)。設計パラメータに対して、製品の性能を表す目的関数の形が問題である。製品の設計点で、設計パラメータに対して、目的関数が鍋底のような形であれば、製造工程で設計パラメータが多少変動しても、製品の性能には大きな影響は与えない（**図 5**)。しかし、製品の設計点で、設計パラメータに対して、目的関数が急激に変化するような形をしていると、設計パラメータの少しの変動が製品の性能に大きな変動が出てしまう（**図 6**)。良い設計は、設計点を鍋底の中心に置くことである。

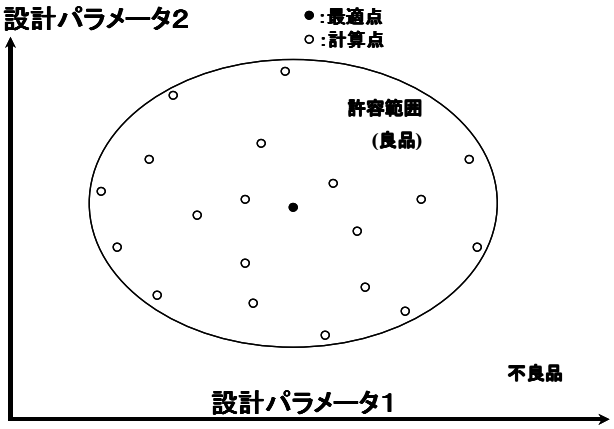


図 4　最適点と許容範囲

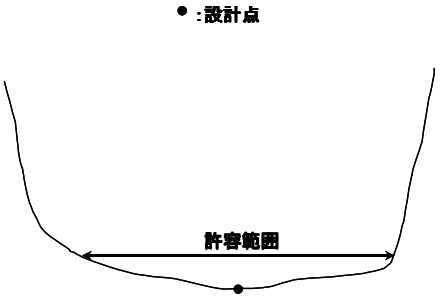


図 5　設計パラメータの変動に強い場合

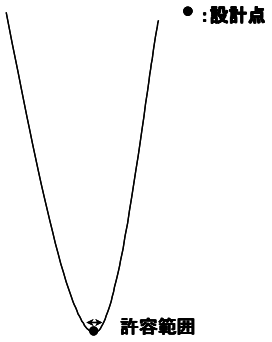


図 6　設計パラメータの変動に弱い場合

「設計点を鍋底の中心に置く」ためには最適点の計算と共に、最適点付近での感度分析あるいは許容範囲の確認が必要となる。そのためには、最適点付近で多くの設計パラメータの組に対する計算が必要となる。この各々の計算は互いに独立なため、並列に処理可能となっている（自明な並列処理)。

上記の（1)、(3）の各設計パラメータに対する計算は完全に独立である。(2）の最適化は最適化の手法により、並列化可能かどうかが決まる。

理想的な手順としては、まず、設計パラメータの最適の場所を探し、その付近での設計パラメータのバラツキに対する製品の品質（歩留まり）の変動状況を確認することが重要である。

4. 最適化

前章で述べた「自明な並列処理」の内、最適化は最近、製品開発という面から最適設計､最適実験として注目されている｡製品の製造において歩留まりを向上させること(不良品を出さないこと）は非常に重要である（半導体の製造等で重要)。また、製品の性能を少しでも向上させることも重要である（発電関係の機器等)。

製品の設計・開発に関して、最適化を行うには、

(1) 実験・試作の繰り返しで行う（図 7）
(2) 実験・試作と計算機シミュレーションを組み合わせて行う（図 8）
（実験・試作なしで済ますことが理想であるが）

の 2 つのやり方がある。実験・試作の繰り返しだけで最適化を行うことは不可能ではないが、非常にコストがかかる。また、時間も非常に長くかかる。近年、計算機の性能も向上し、また、計算モデルの精度も向上したため、実験・試作の代わりに計算機シミュレーションで代用させることが可能になってきている。シミュレーションによる計算結果の精度が向上し、十分、実験・試作の代わりに出来る分野が増えつつある。

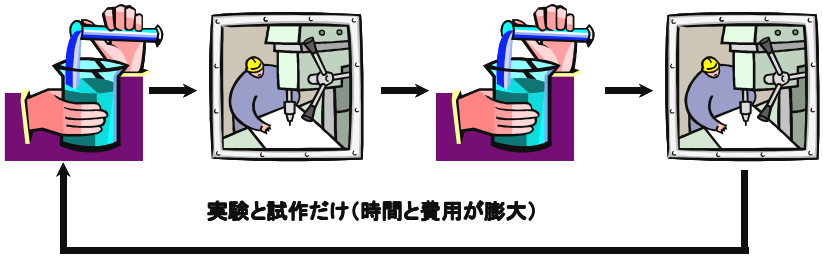


図 7　実験・試作の繰り返し

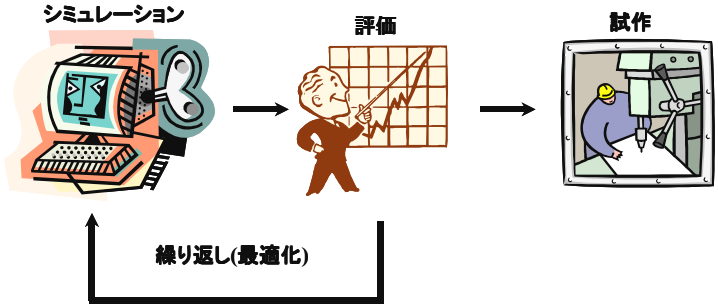


図 8　実験・試作と計算機シミュレーションの組み合わせ

計算機シミュレーションが実験・試作の代わりとすることができるようになったことで、製品の最適化も計算機で行うことが可能になってきている。最適化は以下に述べるように、だれでもが容易に使いこなせるようになっているわけではないが、実用上、非常に重要なテーマとなっている。それだけに、HECC のような高性能計算機の環境をサポートすることが重要な分野といえる。

計算機シミュレーションで製品の最適化を行う場合、次の点に注意しなければならない。

(1) 目的関数の定義
(2) 局所的最適ではなく、全体最適な解を見つける

目的関数をどのように定義するかは非常に難しい問題である（**図９**）。計算結果のどのような値が最終製品の歩留まり・性能に関係しているかを見極めなければならない。例えば、ある部分の電圧だけが製品の性能に関係しているような場合は、目的関数の定義は比較的簡単である。しかし、出力波形の形、しかも波形の形だけが重要で、その相対的位置はあまり問題にならないような場合は電圧だけの場合に比べ、目的関数の定義が難しくなる。現実の問題において、目的関数の定義は複雑であり、目的関数の決定は慎重に行わなければならない（それだから面白いといえる）。

f(p1,p2,･･･, pn) → 最小化
f:目的関数
p1,p2,･･･,pn:設計パラメータ
どのように目的関数を決めるか？

図 9　目的関数の定義

また、全体最適な解であるが、目的関数の形が単峰性の問題の場合は、目的関数の形が単純であるため、全体最適の解を見つけることは易しい。しかし、ほとんどの問題では、目的関数が多峰性のため、局所的最適ではなく、全体最適な解を見つけることは難しい（計算量が多くなる）（**図 10**）。多くの最適化の手法は局所的に最適な場所を求めるような手法である。また、**図 11** の例のように、全体最適点のところが、設計パラメータの変動に敏感で、その次の局所最適点の部分が鍋底のような形をしている場合は考えどころであろう。

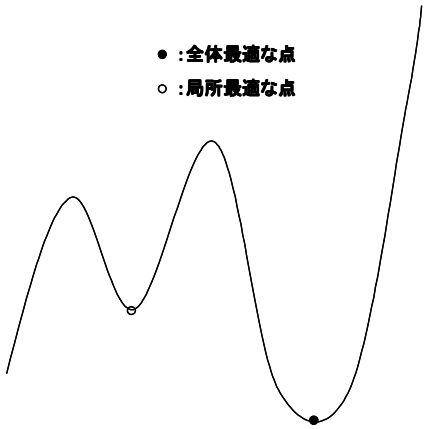


図 10　局所最適点と全体最適点

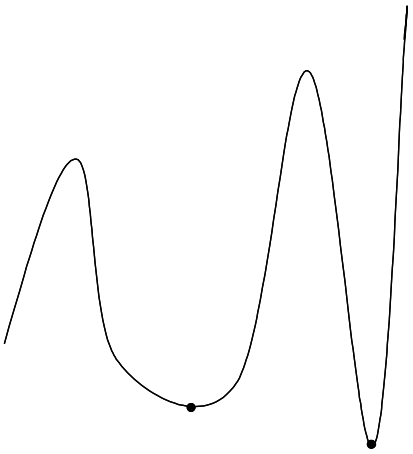


図 11　どこが望ましい点か

最近では、製品の性能向上と開発期間の短縮が求められている。前に述べたように、計算機の性能向上と計算モデルの高精度化により、実験・試作の代わりに計算機シミュレーションを利用した最適化が行われるようになっている。最適化のためには、設計パラメータを変えたシミュレーションを複数行う必要がある。分野にも依存するが、最適化までには、バラツキの評価も含めて、1,000～10,000 ケースのシミュレーションをしたいという分野もある。例えば、5 つの設計パラメータがあり、各パラメータ毎に 5 つを選ぶとこの組み合わせだけでも、3,125 回の計算が必要になる。5 つの設計パラメータは、現実の問題では、決して多い方ではない。

このような例ではたとえ 1 つのシミュレーションが 1 時間で終わっても、全体では、1 ヵ月から 1 年程度の計算時間となってしまう。これは HECC 向きの計算になる。ただ問題は総当り法では、すべてのシミュレーションが独立に行われるが、最適化の手法では、必ずしも、すべてのシミュレーションが独立に行えるわけでもない。

古典的なシンプレックス法で考えてみると、最初のシンプレックスの計算には（設計パラメータの数）＋1 回のシミュレーションが必要で、これは独立に計算することができる。しかし、その後の新たな計算点に相当するシミュレーションは逐次に行わなければならず、各設計パラメータに対するシミュレーションを並列計算は行うことはできない。目的関数の傾斜を使って最適点を求める方法（多変数の場合はヤコビヤン行列）では、数値的に微係数を計算する場合では、少なくとも（設計変数の数）＋1 回の計算が必要となり、この計算は並列にできる。また、最近、設計最適化のアプリケーションで良く使われる応答局面法では、応答局面を構成する点の数だけの並列性がある。

HECC が数千から数万の CPU で構成される計算機とすると、最適化手法において、100 オーダの設計パラメータの組を並列に計算することができれば、その意義は大きい。1,000 CPU の計算機であれば、各シミュレーションを 10 CPU で、10,000 CPU の計算機であれば、各各シミュレーションを 100 CPU で並列計算すればよいことになり、1 つの計算を 1,000 CPU あるいは 10,000 CPU で並列化することよりも、並列化が非常

にやり易くなる。

次世代高性能計算機（HECC）は、1 つのノードが複数の CPU から構成され、そのノードをクラスターの構成要素とする方式となると思われる。最適化の場合、各ノード内では通常の並列化を行い（共有メモリーで可能：並列化コストの低減)、各ノード間で別の設計パラメータの計算を行うことで、ソフトの生産性と並列化の効率の向上という一石二鳥を得ることも可能になる（図 12)。最適化は HECC のハードウェア構成にも適合しているといる。今後は各設計パラメータに対する計算をなるべく並列（併行）に計算できるような最適化のアルゴリズムを開発することが重要である。

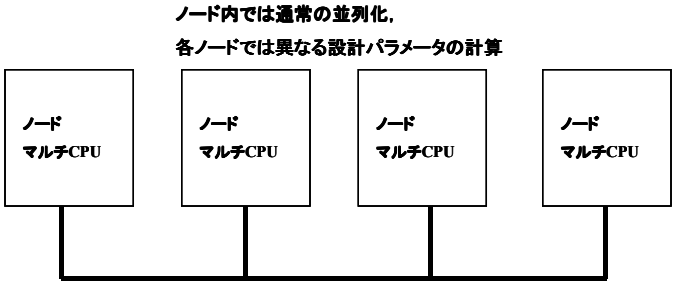


図 12　HECC の構造と最適化

5. まとめ

並列化は粒度を小さくする。しかし、パラメトリック・スタディや最適化のような場合は計算粒度を小さくすることなく、問題が大きくなる。これは HECC 向きである。

3.6.2　高性能計算（機）雑感

横川 三津夫委員

高性能計算機を、「高い処理能力をもつ計算機」と定義することはあいまいである。処理を行うべき対象を明確にしなければ、高性能計算機が具備すべき要件について議論することはできないし、処理能力の程度についても議論できない。私は「高性能計算機」という言葉に、「大規模科学技術計算のための」、あるいは最近では「計算科学のための」高速計算機をイメージするのであるが、通常は“高い処理能力”＝“理論最大性能が高い”という図式で解釈されることが多いと思う。

電子計算機（この名前自体が古めかしいが）は、その開発当初まさしく計算のみを行う機械であった。しかし、その汎用の処理能力によって様々な情報を加工できる万能の機械となってしまったところに、高性能計算機（ハイエンドコンピュータ、或いはスーパーコンピュータ）の悲劇があるように思える。すなわち、「パソコンの能力を高くしたもの」、「誰もが使いこなせるもの」などと、特に努力せずともその能力を手に入れられるものと考えられてしまっているのではないだろうか。

本稿では、大規模科学技術計算のための高性能計算、あるいは高性能計算機についていろいろな観点から考えてみることにする。

1. ベクトル計算機

2001 年 3 月 1 日付け朝刊に掲載された見出しはまさに衝撃的であった。「NEC スパコン、対立の米大手に供給へ」（読売）、「NEC クレイ社と提携」（朝日）、「NEC“昨日の敵”と和解」（日経）など主要な新聞はもとより、インターネットの主要ニュースサイトにおいてもこの情報は大きく取り上げられた。そもそもこの報道の発端は、1996 年 5 月に米国大気研究センタ NCAR（National Center for Atmospheric Research）が NEC 製スパコンを選定したことにある。この選定結果に関し、米国クレイ社が 1996 年 7 月、米国商務省に対しダンピング提訴を行った結果、日本製スパコンに高率の課税がなされ事実上米国市場から締め出されたのであった。そのクレイ社が、NEC 製スパコンを自社ブランドで販売する提携を発表したのである。この合意はダンピング課税の撤回が条件とされているが、この動きは米国市場で有利であったスカラ並列型計算機がその実効性能において予想以上に対ピーク性能比が悪く、高度な解析を必要とする（ベクトル計算機を必要とする）研究者、技術者の要求を満たすことが出来なかったと言って良いのではないだろうか。

事実、ホワイトハウス科学技術政策オフィス（OSTP: Office of Science and Technology Policy）の環境部門が発足させた「気候モデリングに関する臨時作業部会」は、USGCRP（United States Global Change Research Program）に対する報告書の

中で、高性能計算に関して以下のように指摘している[1]。

- 米国の研究者が利用できない日本製共有メモリ型ベクトル計算機は、米国の研究者が利用できる計算機能力をはるかに越える使い勝手と高性能を有している。
- 米国製の並列計算機（普通は分散メモリ型であるが）は使いにくく、加えて気候科学分野のアルゴリズムに対してこの種の計算機で高性能を達成する事は本質的に限界である。
- 米国以外の国に導入された日本製コンピュータにより、少なくとも 3～5 年間にわたって、米国の科学者の計算能力が大幅に劣ることは確実である。
- 米国のソフトウェア開発への投資が不十分である。
- コモディティプロセッサを用いた分散メモリ型並列計算機の利用を推進している米国の政策は、ソフトウェアへの投資額を増大させる。

米国クレイ社 CEO のロットソルク氏は、米国市場においてスカラ型並列計算機が主流になったことについて、「ダンピング提訴とクレイの SGI 傘下入りが同時期に重なったため、ねじれ現象が起きてしまっていた。」と述べている[2]。この発言の妥当性はともかく、高性能計算（大規模科学技術計算）の分野において、ベクトル型を必要としている分野が厳然として存在していると言える。これは、マイクロプロセッサを用いたスカラ型並列計算機だけの開発に投資した米国のスパコン政策が曲がり角に来ていると言って良い。

2. コストパフォーマンス

コモディティプロセッサを用いたスカラ型並列計算機が優位であると主張する者は、よく高性能計算機の価格（本稿では「基本コスト」と呼ぶことにする）に対するピーク性能比（コストパフォーマンス）の優位性を根拠に議論する。しかし、基本コストとピーク性能に基づくコストパフォーマンスの議論だけでは、一般にコストパフォーマンスが悪いと言われているベクトル型計算機を、応用分野のユーザが依然として手に入れたいと思っていることについて説明できない。応用分野のユーザは、各自の応用ソフトウェアの実効性能と実効コストを用いた真のコストパフォーマンスの議論をしたいのである。ここで、実効コストとは、基本コストに並列計算機上での応用ソフトウェア開発の経費を加えたコストとする。すなわち、スカラ型並列計算機上での応用ソフトウェアの開発及びチューニングは労力がかかるが実効性能があがらず、結局実効コストに基づくコストパフォーマンスが悪いことに気がついたと考えることが出来る。もちろん、ベクトル型の計算機であっても単体プロセッサだけでは既にそのピーク性能が限界に近く、実効性能を高くするためには並列構成を取らなければならないので、並列化した応用ソフトウェアの開発は必要である。しかし、スカラ型並列計算機と同等の実効性能を

ベクトル型並列計算機で得ようとする場合には少ないプロセッサ数（並列性）で実現できるので、当面数万～数 10 万のプロセッサを結合した“超”並列計算機のような高い並列性を考慮しながら応用ソフトウェア開発を行う必要はない。

より一般的には、高性能計算機の実効コストは、計算機シミュレーションを用いる作業全体の必要経費（本稿では「総コスト」と呼ぶことにする）の中で考えるべきであり、たとえ基本コストがある程度高くても、それを用いた計算機シミュレーションにより開発に必要な期間の短縮や経費削減が得られれば、総コストが節約できるかもしれない。例えば、製品開発における計算機シミュレーションの直接的メリットは、製品の実質的コストの削減である。今、コストパフォーマンスを考える時に、計算機シミュレーションを行わず、実験等で製品開発する場合のコストを 100 としてみよう。計算機シミュレーションを用いた場合に、計算機本体に必要な価格を含めた計算機シミュレーション全体のコストが 30、そのため実験等のコストが 50 に減少したとすれば、同じ製品を開発するのに実質的に 20 のコスト削減が得られたことになる。このような観点でコストを評価するならば、高性能計算機の基本コストは総コストの中に埋もれて見えなくなり、基本コストは問題にならない。

また、災害などの人命に関わり、緊急を要する計算機シミュレーションには、短 TAT（ターンアラウンドタイム）が要求され、そのために高性能計算機が必要であるが、このような状況下ではもはや基本コストに基づくコストパフォーマンスの議論は意味をなさない。

3. 計算機科学者と計算科学者とのギャップ

計算機そのもの、あるいはその上でのソフトウェア（ここでは基本ソフトウェアと呼ぶことにする）を研究対象にしている研究者と、計算機シミュレーションの結果が重要で計算機は道具であるとしか考えない応用分野の研究者との間には、計算機に向かう姿勢に大きなギャップがある。例えば、前者は計算機あるいは基本ソフトウェアが完成すれば研究或いは開発目標が達成されるので、実効コストを用いたコストパフォーマンスの議論をすることがなく、したがって基本コストとピーク性能比によるコストパフォーマンスの議論の中から抜け出すことが出来ない。一方、後者は成果を得ることを目的としているので、応用ソフトウェア開発にかかる実効コストと実効性能によるコストパフォーマンスの議論が重要であり、基本コストとピーク性能比によるコストパフォーマンスの議論は意味をなさない。

また、応用ソフトウェア開発環境に必要なツール開発に関する考え方も、前者と後者では異なっている。前者は、計算機そのもの、あるいは基本ソフトウェアそのものを研究する時に、その利用方法をモデル化してそれに適したツールを開発している。モデル化に基づくツールの開発は科学の方法論としても妥当であろう。しかし、実際の計算機シミュレーションをする応用ソフトウェアの研究者にとっては、使い勝手や処理時間の

観点から実用に耐えないツールは意味がなく、すなわちツール開発の最終過程では、実際の現場環境でその実用性評価を行うべきである。大規模科学技術計算においては、

- 通常、計算結果の出力が多く、また長時間実行した後にエラーが起きることが多いので、実時間トレースによってデバッグ機能はあまり役に立たない、
- 数千の並列プロセスのデバッグに、十数個の並列プロセスで評価したデバッグツールが役に立つとは思えない、
- 分散されて得られる大量の出力結果からデバッグにつながる情報をどう取り出すかは応用ソフトウェアの研究者の深い洞察によることが多く、単純なパターンマッチング等の定型的な作業では行えない、

などの応用ソフトウェアの開発環境ツールが現実の作業と遊離していると指摘できる。実際に応用分野の研究者が数万行、数十万行に及ぶ独立したメッセージを見ていく良い方法は考えられず、一昔前の方法のようにプログラムの適当な場所へデバッグ出力を挿入し、出力されたデータを紙の上で丁寧に追っていくしか方法がないのかもしれない。計算機シミュレーション結果の画像処理は役に立つかもしれないが、それは大まかなバグ部分の絞込みであって本質的な解決にはならないと思う。

近年、ソフトウェアの重要性が認識され、高性能計算を支える基盤ソフトウェアに多大な投資をしている。しかし、これらのツールは現実の大規模科学技術計算（計算機シミュレーション）に本当に役に立つのか考えてみるのも重要でないだろうか？

4. 高性能計算機の開発資金

原子力委員会 ITER 計画懇談会の「研究の資源配分と国際協力の責任分担に関する検討報告書」という興味深い資料がある[3]。報告書の目的を考慮すれば内容を若干差し引いて読む必要があると思うが、政府負担資源の使途について「国策遂行のための研究・開発に優先度がある」と述べられている。さらに、資源配分すべきカテゴリとして以下のものがあげられている。

① 国民全体という見地からの広義の安全保障
② 国家という規模で行われる国際的機能
③ より下位組織の自発選択では困難な、しかし学問上必要と認められる研究・開発の支援
④ 先導的産業技術の開発支援
⑤ 国民（の一部）の要求への対応

高性能計算機の開発はどのカテゴリに属するであろうか。過去、国策として計算機産業育成に重点をおいた時期があるが、もはや高性能計算機の開発を先導的産業技術に分類させることには無理があるように思われる。したがって、計算機シミュレーションを広義の安全保障を確保するために用いる手段と考え、そのために必要な高性能計算機を開発するとの説明が妥当である。

ペタフロップスマシン技術に関する調査研究Ⅱの中で、私は「高性能計算機は、パソコンのように誰でも容易に使えるような大衆のツールではなく、大規模な実験装置とみなすべきである」との提言をした[4]。しかも、この実験装置は特殊なものであり、十分な開発費がなければ成り立たない。これを用いて多方面の問題に対し計算機シミュレーションを行い広義の安全保障に貢献することができるならば、高性能計算機開発の資金を負担する意義は大きい。多くの人にその意義を理解してもらうためにも、計算科学によって得られた結果に基づく有効性について声高に説明する必要があるし、それが継続的に高性能計算機を開発できる環境につながるものと考えられる。

ここで、重要なことは高性能計算機アーキテクチャのオプションを複数持つことである。スカラ型並列計算機もベクトル型並列計算機のどちらも重要である。米国のように一つのオプションに片寄ってしまえば、他のオプションを開発する技術は離散、衰退してしまい、再度その技術を復活させる事は容易ではない。

5. おわりに

先に挙げた原子力委員会 ITER 計画懇談会の報告書では、「米国の『戦略』に全て付き合う必要はない、という視点は重要である」とも述べられている。高性能計算機について（もちろん他の分野についても）、米国の研究開発動向が気になるところではあるが、「米国でペタフロップスマシンと云えば、日本でもペタフロップスマシン」という風にことさら騒ぎ立てる必要もない。高性能計算機の開発は国家レベルで行うべき研究開発の施策であり、諸外国の研究・開発を参考にしつつも、それに振り回されることなく我が国独自の方針をもって進めるべきである。

参考文献

[1] “High-End Climate Science: Development of Modeling and Related Computing Capabilities,”USRGCP, Subcomittee on Global Change Research, December (2000).

[2] 日経産業新聞，平成 13 年 3 月 19 日朝刊

[3] http://www-atm.jst.go.jp/jicst/NC/iter01/siryo/siry12/siryo2.htm

[4] ペタフロップスマシン技術に関する調査研究Ⅱ，先端情報技術研究所，平成 10 年.

第４章　あとがき

本年度における HECC ワーキンググループの調査は、過去３年間の「ペタフロップスワーキンググループ」における調査活動の結果および昨年の「HECC ワーキンググループ」の報告書を踏まえ、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を探るように努めるとともに、今後注力すべき技術分野の検討に必要な元データを抽出するように努めた。この報告書は、それらの議論を踏まえて各委員の見解をまとめたものであり、必ずしも各委員の意見が同じ方向を向いているわけではない点はご了解願いたい。

本報告書の内容は、ハードウェア／アーキテクチャ、グローバルコンピューティング、プログラミング／コンパイラ、シミュレーション、並列処理／アプリケーションなどと多岐に渡っているが、各項目における技術はそれ単独では存在しない。情報処理の分野の研究開発は､ハードウェアからソフトウェアおよびアプリケーションまで、その最先端技術をすばやく取り入れ、新しいシステムの開発や規格の提案に生かすと同時に、情報処理システムのニーズを的確に把握することがますます重要になってきている。たとえば、数値計算をサポートするための計算機システムは、汎用のマイクロプロセッサをベースにしたいわゆるスカラ型の並列計算機とベクトルプロセッサを主体にしたベクトル並列計算機に大別される。米国では、主としてスカラ型並列計算機システムがハイパフォーマンス計算を行うために用いられており、価格性能比が高いだけではなく、いくつかのベンチマークテストなどにおいては、高い性能を引き出すことが実証されている。このタイプの計算機システムにおいては、コンパイラ技術や最適化技術、あるいは並列化のためのプログラミングツールなどの果たす重要性が高い。このことによって、米国では並列化のためのソフトウェア技術や並列化されたプログラムのライブラリなどが蓄積されてきたことは事実である。しかしながら、すべてのプログラムやアルゴリズムが全て効率よく並列化されるというわけではなく、プロセッサ数が多くなるに従って、むしろ効率的な並列化ができないアプリケーションが存在するという事実もユーザの間で認識されてきている。また、従来から我が国のスーパーコンピュータの主流であった並列ベクトル計算機の方が、超並列計算機に比べて一般的に並列化の効率が高いという認識もある。本報告書の中で、横川委員の報告にも書かれていたように、2001 年の 2 月の末、NEC と米国クレイ社との提携が発表された。このことの背景には、第 1 に NEC が 454%のダンピング課税を課せられるなどしてベクトル型を得意とした日本製スーパーコンピューターが米国市場から排除されてきたという事実があげられる。さらに、ハイエンドのベクトル型スーパー

コンピュータをもはや米国では製造することができずに（あるいはコスト的に見合わないとの判断で)、日本製を導入せざるを得なかった状況と、米国のある種のユーザから気象予報や自動車の衝突試験などのシミュレーション解析の分野で優位性のあるベクトル型を求める声が高まっていたということがその背景にあると考えられる。

上に述べたことは、必ずしもベクトル型スーパーコンピュータの優位性やスカラ型の並列計算機の導入に伴う、並列化のためのソフトウェア技術が不必要であったということを述べたのではない。情報処理産業を全般的に捉えると、より汎用的な技術である並列化ソフトウェア技術に関して積極的な支援を行う必要があるのはいうまでもない。そのような意味において、わが国においても遅ればせながら、本ワーキンググループの委員である早稲田大学の笠原教授が中心になって進めている「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」が通産省（現在は経済産業省）の下で開始されたことは喜ばしいことである。このプロジェクトではマルチプロセッサ・コンパイラの性能を高めることを目指して産学で共同研究が進められているが、わが国での大学での基礎技術に基づいて民間が共同開発を行うことは、情報処理技術の特にソフトウェアの分野ではなかなか見られなかった形態であり、その成果に期待したい。

最後に、昨年の報告書のあとがきで、わが国においても、省庁横断的な形で情報処理に関する提言がまとめられているということを紹介した。米国政府に対する PITAC などの提言と比較すると、総花的で、概念的な面はあるが、このような方向性が現れたことは、大きな一歩であった。この提言を受けた形で、わが国においても、昨年（2000年)、森内閣のもとで IT の推進ということが叫ばれるとともに、補正予算や各種の戦略会議がつくられ、国をあげて情報処理の推進を図るということが強調された。このような形で IT 基本戦略が出されたこと自体は期待が持てるが、インフラ作りが強調されていて、それを実際に担い利用するはずの肝心の R&D の推進と IT 技術者･研究者の育成については、それほど重要視されていないように見える。米国の IT 関連の R&D 政策を見てみると、20 億ドル程度のそれほど大きいとは思えない予算規模ながら、連邦政府の推進体制に、公的研究機関、大学、民間企業の一体となった実施体制がうまく噛み合って、予算規模をはるかに越えるような成果を上げているという点であり、今後はこのような基本戦略を見習いつつわが国に即した R&D 政策を立案すべきであると考える。そのような目的の際に、本報告書が参考になれば幸いである。

（山口喜教主査）

付属資料１

SC2000 参加報告（その 1）

—— SC2000 High Performance Networking and Computing ——

報告者　福井　義成

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（HECC - WG）の活動の一環として、2000 年 11 月 4 日～10 日に、米国テキサス州ダラスにて開催された“SC2000”に参加したので報告する。

この会議は、1988 年のフロリダ州オーランドで最初に開催され、その後、毎年 11 月頃に米国で開催されている。今回は通算、13 回目にあたる。この会議の履歴については、http://www.supercom.org/を参考にしていただきたい。

筆者は第 1 回目と第 2 回目（ネバダ州リノ）に参加した。今回は約 10 年ぶりの参加であり、昨年（SC99）からの変化については残念ながらわからなかった。ここでは全般的な報告ではなく、筆者の主観による報告とする。SC2000 については全般的な報告は下記を参考にしていただきたい。

・小柳義夫先生（東大）のレポート

http://phase.hpcc.gr.jp/phase/tech_report/SC2000/oyanagi.txt

・Hokke グループのレポート

http://phase.hpcc.gr.jp/phase/tech_report/SC2000/

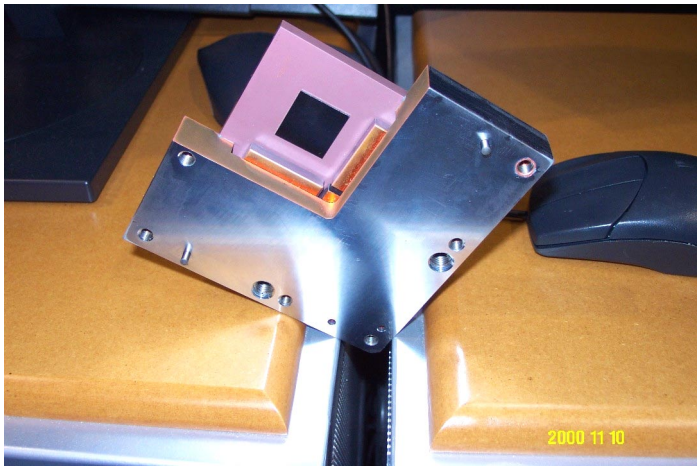
会議名：SC2000　- SC2000 High Performance Networking and Computing -
日　程：2000 年 11 月 5 日～11 日
場　所：テキサス州ダラス
参加者：福井義成　若杉康仁

会場風景

1. 概要

今回は、大規模な並列アプリケーションの状況調査を中心にした。そのため、講演の聴講と共に ASCI を中心にした大規模な並列アプリケーションの関係者にコンタクトをとった。会議の前後に関連する研究機関等の訪問を行いたかったが、スケジュールの関係で断念した。

会場の Dallas Convention Center は非常に広く、SC2000 で使用しているのは一部であった。第 1 回、第 2 回に比べて、Exhibition が盛大になっていた。第 1 回などは講演会場のホテルのホールで、小規模なものであった。今回の展示では、IBM,SGI、COMPAQ などが、入口の近くで大々的に展示を行っていた。日本のメーカはややおとなしい感じであった。日本の研究機関では RWC が広い場所を占めており、人数も多かった。IBM が Power4 の試作機を展示していたのが、印象的であった。



IBM の Power4

10 年前にもダラスで開催された並列・分散処理関係の会議（12 月）に参加したが、その時の天候は非常に良く、日本の 5 月頃の感じであった。今回は非常に寒く、11 月 8 日の夜には雪になってしまった。飛行機の中で着るために持っていったジャンバーを背広の上から着てちょうど良いくらいであった。

2. 大規模アプリケーションの状況

会議の後、毎年、参加している方に話を伺ったところ、ASCI の状況は昨年とあまり変わっていなかったとのことであるが、現状、特に並列化の効率について調査した結果を報告する。

ASCI のアプリケーションは関係する 3 つの研究所で以下のように分担している。

・Sandia
  ・AZTEC　An iterative sparse linear solver package
  ・MPSALSA　Massively Parallel Numerical Methods for Advanced Simulation of Chemically Reacting Flows
  ・CUBIT　Mesh Generation Research Project
  ・ALEGRA　A Three-Dimensional, Multi-Material, Arbitrary-Lagrangian-Eulerian Code for Solid Dynamics
  ・Computational Sciences, Computer Sciences, and Mathematics Center
・LLNL
  ・ASCI Turb　strongly compressible three-dimensional hydrodynamic flows
  ・CASC　high performance computing, computational physics, numerical mathematics, and computer science
  ・MeshTV　Scientific Visualization and Graphical Analysis Software
  ・Python　Framework
・LANL
  ・Antero　one-, two-, and three-dimensional unstructured mesh Arbitrary Lagrange-Eulerian (ALE) full-physics code suitable for stockpile-relevant simulations
  ・Blanca　Stockpile-relevent Caluclation
  ・Crestone　3-D codes to model and understand issues involved with stockpile aging
  ・Shavano　nuclear device performance
  ・Telluride　casting processes

使用している言語・ツールは C++、F 90、MPI、Pooma（Parallel Object Oriented Methods and Applications）とのことであった。

関係する講演を聞くと共に、知り合いに、大規模アプリケーションの関係者への紹介を依頼した。講演での発表等の上手くいった場合の性能ではなく、現実的な値を知りたいと色々な人に接触した。

(1) SGI の Bob Bishop の主催の夕食会。

出席者は、HPC 関係の責任者が多かった。出席メンバーは下記の通りであった。

Bill Feiereisen	:NASA Ames Director NAS Division
Milt Halem	:NASA Goddard Space Flight Center
Dona Crawford	:Sandia National Laboratory
Ed Oliver	:Department of Energy

Stephen Younger	:Los Alamos National Laboratory
Andy Uraskie	:Office Chief, HPC NSA
Cray Henry	:DoD Modernization Office Office Chief, HPC
John Mulholland	:Scientific Systems Analysis Communications Security Establishment
Carol Hopkins	:Canadian Meteorological Centre
Frank Williams	:University of Alaska Artic Region Super Computing Center
Professor Ron	:UK Research Councils Technology Watch Panel

このメンバー及び元 CRAY の Derek Robb（元 ASCI Blue Mountain の SGI 側責任者、現在 SGI)、David Slowinski（e の計算の世界記録保持者、元東大先端研、現在、IBM Watson Research Center ACTC）らに関係者を紹介してもらった。

(2) Caltech の Shron Brunett

Caltech の Shron Brunett に Simulation of Dynamic Response of Materials の実行性能について、色々と質問した。ASCI Blue Mountain を使用しており、1,000 プロセッサオーダーで 10～12%の効率とのことであった。

(3) John Morrison（CCN Division Office Acting Division Director）
ACL(Advanced Computing Lab.)

アプリケーションの組織等について、話をしてもらった。実効効率の点では気候関係（Climate Systems）では大規模では 7～9%とのことであった。

(4) IBM Geert Wenes:Consulting I/T Architect

1,000 プロセッサで、実効効率 10～12%は悪くない。現実的には、大規模な計算は夜だけ実行しているので、あまり気にしていないとのことであった。

(5) IBM John M. Levesque:Director ACTC Watson Res. Ctr.

行列計算のようにメモリーネックを回避できるアプリケーションでは、実効効率は、30～40%出せるが、メモリーの比重が大きいアプリケーションでは、15～30%であれば良いほうである。

これ以外に、CRAY、IBM、SGI の製品計画をじっくり聞く機会（各々2～3 時間）があり、その時にも大規模計算の実効効率について聞いたが、知りたい回答は得られなかった。

3. まとめ

ASCI のアプリケーションの状態は昨年と大きな変化はなかったようである。現実的なアプリケーションの場合、1,000 CPU 規模の並列化の効率は、10～20%程度のもあるということである。並列化に向いた問題であれば、かなり良い効率を出すことも可能であるが、並列化に向かない問題であれば、10～20%程度の場合もあることを認識する必要がある。十分に並列化費用をかけることができれば、効率をあげることは可能であるが、要はどこまでコストをかけられるかとうことである。

また、日本との違いで強く感じたことは、

（1） 実際に大規模な環境で大規模な計算をやることが大切
（2） 各機関の間のコラボレーションが日本より出来ている

であった。ASCI の RED, Blue Pacicic, blue Mountain, White, Q とどんなものであれ、実際に使えるものがあるとソフトの技術は進むという印象であった。

付録資料 2

SC2000 参加報告（その２）

報告者　若杉　康仁

１．はじめに

コンファレンス開始前日（11/6）に Registration 後、午後 6 時半よりアルコール類を含む飲み物、食べ物が Buffet スタイルで用意され、出展者が事前にお披露目を行った。これはなかなかいいスタイルである。飲みながら会場を回り、気軽に質問できてうれしい。

今回の展示は企業 95 社、研究所 60。会場のネットワーク Scinet は 9.6Gbit/sec で接続されている。展示マシンは汎用のチップを活用したクラスタシステムが目に付く。既に研究開発分野の OS は Linux が主流になりつつある。

最近は日本ではスーパーコンピュータに対する取り組みがその市場の狭さ、企業の余裕の無さからかあまり積極的になされていない。メインフレームを米国で販売しないと決定した企業にスーパーコンピュータの開発の余力は感じられないのは残念である。

翻って、米国では好調な（だった？）経済に支えられて、国の予算を元に超並列を主体とする High End Computing の研究開発が積極的であると感じた。

今回の 11/7 の開会式には本年（2000 年度）ノーベル賞受賞者のキルビー博士が招かれ、ご高齢（77 歳）のため“ようこそおいで下さいました。・・・”の簡単な挨拶ですまされたが会場は大変盛り上がった。博士は地元ダラスの Texas Instrument 社で IC を 1958 年に発明され、これによりノーベル物理学賞を他の 2 名と共同で受賞された。その縁の地での開催。

次回の SC2001 は 11/10/2001 よりコロラド州デンバー市で開催予定である。SC Global といってインターネット、Access Grid を活用した展示になる予定である。

展示会場内の Access Grid

2. 講演

2.1 “Petaflops in the Year 2000”（基調講演）

Steven J Wallach （11/7 8:30 – 10:00）

ペタフロップスマシンを実現するには電力消費量が問題になっており、すべての接続をファイバーで実現した AON（All Optical Network）になるだろう。

そのころの OS は LINUX が 99%で NT は 1%ぐらいではないかとも予測。

構成としては 4CPU/die, 128die/box これを 64 個 40Ghz の光ケーブルでつなぎ 1 ペタフロップスを実現する。1CPU=30Gflops, 1die=120Gflops 総計 8192die で約 1Peta となる。

印象としては算術計算のみの性能計算でソフトウエアを含めた実行性能では大いに疑問が残るがその当たりはあまり触れられなかった。

たとえば地球シミュレータが 40TFlops であり、これを 25 個つなげば実現するか？

2.2 “COTS Cluster for HPC”

Dr. Thomas Stirling of NASA (11/8 8:30 – 9:15)

今回の展示でも既に 20 のクラスターがあり、今後の High Performance Computer の主流になると予測。流れは Single -> SMP -> MPP -> Cluster である。既に TOP500 の 31 番 Compaq Alpha Server SC は 0.5Tflops の性能で動いている。これは NASA の Beowulf プロジェクトの成果でもある。PCI インターフェースは重要な標準。Myrinet, Giganet 等、高速ネットワークは重要。Open Source Software が主流である

2.3 “A Small Dose of Infosec”

Dr. Eugene Spafford （11/8 9:15 – 10:00）

I love you ウィルスをはじめ最近の被害の増大を訴え、2004 年には 15 分に 1 回ウィルスが発生すると予測。更に COTS を使うことが多くなり Security に対する不安はます一方である。

2.4 “Numbers, Lots of Numbers, And Insight, too: Scientific Computing 2000”

Dr. Margaret Wright (11/9 8:30 – 9:15)

手書きの OHP を用いたのが印象的。理学系の風習であろうか？　内容はいまいち。

2.5 “Parallel / Distributed Programming: Research Success - Application Failure”

Dr. J. C. Browne （11/9 9:15 – 10:30）

並列計算は自然であるが、Von Neumain 流（現在のコンピュータ）Sequential である。性能を上げるには並列は必須である。これを効率よく記述するにはコンポーネントモデルによるデザインパターンが必要。会場で現在どんな言語で記述しているかの即席

挙手のアンケートを採っていたが、MPI と OpenMP では MPI が圧倒的に多かったのが印象に残った。MPI を Fortran, C, C++で利用。

3. テクニカルセッション

テクニカルセッションは 4 つの会場で同時進行するので興味のある物を選択して聞くことになる。タイトルのみが頼りなので期待して行っても外れもあったりするので慌てて会場を移動も多々。注目としては、どんな事例が実際に行われているか期待したが以外と少ない印象を持った。研究から外れるのだろう。分類は Panel（質疑）, Paper（登録論文）が 2 本、Masterworks（招待）の計 4 本のセッションである。

展示会場でも展示者によるセッションが 30 分単位で開かれていたが、時間がとれないため参加は出来なかった。会議の合間に会場を見るため立ち止まっていると展示が見られない羽目になる。

受講した物は、題名と講演者の羅列になるが参考までに次に記す。

3.1 Computational Biosciences: Genomic (11/7 10:30 - 12:00)

- From First Assembly Towards a New Cyber-Pharmaceutical Computing Paradigm
  （Celera Genomic 社の Sorin Istrail 氏）
- Unveiling the Human Genome
  （Whitehead 社/MIT　Jill Mesirov 氏）

3.2 MPI / OpenMP (11/7 1:30 - 3:00)

- Performance of Hybrid Message-Passing and Shared-Memory Parallelism for Discrete-Element Modeling
- A Comparison of Three Programming Models for Adaptive Application on Origin2000
- MPI versus MPI+OpenMP on the IBM SP for the NAS Benchmarks

3.3 Cluster Infrastructure (11/7 15:30 - 17:00)

- PM2: High Performance Communication Middleware for Heterogeneous Network Environments（これは RWCP からの発表である。）
- Performance and Interoperability Issues in Incorporating Cluster Management Systems Within a Wide-Area Network-Computing Environment
- Architectural and Performance Evaluation of GigaNet and Myrinet Interconnect on Cluster of Small-Scale SMP Server

3.4 IEEE Award Winners (11/8 10:30 - 12:00)

- Seymour Cray Computer Engineering Award
  Dr. Glen J Culler が受賞
- Sidney Fernbach Award
  Dr. Steve Attaway が受賞

"Parallel Computing"として講演があった。

5 年前は Simulation にパラレルは使用しなかった。

ASCI Red で 2D/3D をやるようになった。（これは意外な気がした。）

3.5 Masterworks (11/9 15:30 - 17:00)

- Moving a Large Commercial Application from SMP to DMP
- Application Driven Strategy of High Performance Company

3.6 Panel "Computational Grid" (11/10 8:30 - 10:00)

パネル討論会。何故か Ian Foster が目立っていた。次のパネルもかけ持ちで。

- Dr. Ian Foster （Argonne National Lab.）
  Global Grid Future
  Commercial Sector へ IETF を介して
- Dr.Snir (IBM)
  科学からビジネスへ。（本音はうまくいかないとかの発言も）。
- Dr. Fox (Syracuse Univ.)
- Dr. Pancake (Oregon State Univ.)
  No economical Model

3.7 Panel "Mega Computer" (11/10 10:30 - 12:00)

- Dr. Ian Foster
  Mega は Grid の特殊ケースであり、Secured Controlled Resource Sharing と定義づけていた。
- Mr. Chien (Entropia 社)
  Entropia 社の宣伝。1997 年設立。Entropia is Grid と豪語。時期を得た会社で Grid 環境をビジネスにつないでいる気がした。100,000 台のマシンを 80 カ国にわたりつないでいる。
- Andrew Grimshaw (Virginia Univ.)
  Legion を開発しているグループからの発言。

4. BOF (Birds of a Feather)

正式会議が終了後に、やや非公式な感じのセッションが開かれる。並行してかなりの数が開かれる。仲間内の会合のような感じで新しい情報が聞けそうなので以下に出席した。

4.1 The ASCI Simulation and Computer Science Technology Prospectus Organizer (11/7 15:30 - 19:00)

- Road Map Toward 2005（LLNL より Marry）

  1000CPU までのテストは終了した。これは約半分の CPU に当たる。言語は HPC Java

- Software Interoperability（LANL より John）

  2002 年までに標準インタフェースを作りたい。Class ライブラリを使うコンポーネントモデル。これは OMG スタイルの WG を作って推進したい。

  2005 年にプロトタイプができる予定。

- Linear Solver / Non Linear Solver

  特になし

- Storage & File System

  500TB までのテストは完了。

- Scale, Complexity, remoteness exploit data
- Data mining, date Processing, Query & Pattern Recognition
- Visualization（SAL より）

  特になし

- ASCI Grid について

  Distributed collection of xxxx Hardware resource の構築。

  2002 年にサービスをスタートする。Kerberos Secured Web を利用、2005 年に Grid のアクセスが完了する計画。

- ASCI Network Architecture

  LLNL、LANL、SNL 研究所間のネットワーク説明。

4.2 “Global Grid Forum” (11/8 17:30 - 19:00)

Global Grid Forum の 2001 年度の開催予定はは３月にアムステルダム、７月ワシントン DC、10 月ヨーロッパ（場所未定)。

色々な WG がありそれぞれから発表があった。

- Security WG

  PKI for Grid Identification, GSI Road Map, Global Security API 等

- Performance WG

  標準化, シナリオの使用, Event, ディレクトリサービスのフォーマット,Time stamp, Communication protocol 等

- Remote Data Access WG
  Convergent architecture, File base access, Object base access, Collection base access, Information Base Grid, Knowledge base Grid について
- Grid Information Service WG
  Definition of Meta model, Task force（JNI, RDB, GXD）について
- Advanced Programming Model WG
  Active Middleware Model 等
- Distributed Accounting WG
- Grid Computing Environment WG
  Grid リソースにアクセスするツール
- Application and Tool WG
  色々なＷＧがあるので Grid Application WG にマージを検討。
- その他 SC Global より
  次回の SC2001 は最初の真のグローバルな technical Conference になる予定であると宣伝があった。

5. 展示

目に付いた展示を記す。

5.1 米国メーカ

(1)IBM

会場の一番良いところで展示していた。Power チップを使ったクラスタシステムが中心。現状は Power3（333MHZ）だが新しい Power4（1.1GHZ-1.4GHZ）チップのみがが展示されていた。（福井氏の報告の写真参照）

1 枚のボードに Power4 チップを 2 個のせる 2Way から 64Way までのバリエーションを考えている。出荷は 2001 年以降になる。

(2)SGI

新 Server では Linux がメイン OS になる。クラスタも 2004 年には 2048 個までのプロセッサをつなぐことを考えている。

(3)Compaq

ASCI で採用された Alpha チップによる超並列マシンが縮小モックアップで展示されていた。世界一の性能（予定）とうたっていたが。同じモックアップは LosAlamos 研究所にもあった。

(4)SUN

Sun Ray Appliance のしゃれたクライアントが目をひいた。サーバは目新しさ無し。

(5)Cray

CraySV1 シリーズ。特に目立つ展示無く商談中心？

5.2　日本メーカ

(1)日立

SR8000 のモックアップを展示。基本的に米国での販売が関税のため出来ないので、対象のお客は日本と欧州とのこと。

(2)Fujitsu

VPP5000 シリーズ

(3)NEC

SX-5/8 シリーズ

5.3　研究所(日本)

(1)地球シミュレータ

40Tflops で世界最速を目指す日の丸地球シミュレータがモックアップを展示。科技庁関連がブースを構える中にあったがこのコーナは人だかりは少ない気がした。パネル展示プラスアルファであり予算面からして仕方のないところか。

(2)Real World Computer Partnership

ワークステーションクラスタの実機を展示動作させていて迫力があった。なかなか意欲的に展示をしていた。

5.4　米国研究所関連

(1)ASCI 関連

ASCI のブースを出していた。会場ではショーコーナーもあり興味深い展示をしていたがコンファレンスと重なりみられなかった。別途入手した CD-ROM で内容はもう少し詳細にわかったが、昨年も参加した人は同じものだとおっしゃっていた。毎年更新は大変であろう。

COMPAQ 社のブースでは ASCI を展示していた。

ASCI “Q”

(2) Access Grid 関連

Argonne 国立研究所はじめ会場内で数カ所見かけた。デモ効果は大きい。

5.5 Linux Cluster 関連

ハードウエアの展示ではこれが目に付く、既存のチップを使いラック型にマウントするのがほとんど、まだニッチ産業的だが内容的には PC の技術でできるので標準化が進めばさらに値段は低下する。スケーラブルな構成が得意なので、低価格な分野からの参入が可能。見知らぬメーカが一発を狙ってと活気を感じる。

6. 雑感

ハイエンドコンピューティング自体のマーケットの少なさからか、専用マシンによるスーパーコンピュータと言う雰囲気はなく、汎用チップを利用したマシンへ移行する速度が加速している気がした。ただそのしわ寄せはソフトウエアにきており性能を効率よく出すのは非常に難しくなっていると思う。日本も内容的にかなり頑張っているという気がしたがプレゼンスは低いと感じる。

付属資料３

米国のハイパフォーマンスコンピューティング技術動向調査報告

HECC ワーキンググループ活動の一環として、米国におけるハイパフォーマンスコンピューティングの技術動向調査を目的に実施した海外調査について報告する。

【日程】　2001 年 2 月 25 日（日）から 3 月 11 日（日）
【調査員】 古市 昌一 委員、小林 茂 主任研究員
【調査の目的】
米国において大規模並列マシン上のソフトウェアを実際に研究開発している機関を訪問し、言語／開発実行環境／パフォーマンス解析・チューニングツール／ソフトウェア生産性向上法等、大規模並列ソフトウェア開発上の課題が現場ではどのように解決されているかを、研究所見学と研究者へのインタビューにより調査する。また、並列マシンの応用分野の一つとして、米国防総省を中心に活発に研究が行われている、知的エージェントの研究場所も訪問する。

【主な調査先】

1. インテル社 Microcomputer Software ラボ（サンタクララ）
2. イリノイ大学 NCSA（アーバナ・シャンペン）
3. ジョージア工科大学 M&S センタ（アトランタ）
4. ミシガン大学 AI ラボ（アンナーバ）
5. 南カリフォルニア大学 ISI と ICT（ロスアンジェルス）

1. インテル社 Microcomputer Software Laboratory（サンタクララ）

http://www.intel.com/

＜訪問目的＞
インテル社におけるソフトウェア研究開発の現状と将来戦略
＜面会者＞
Mr. Hideki Saito, Senior Software Engineer

インテル社における研究開発組織は TRL（Technology and Research Labs）と呼ばれる。サンタクララの本社敷地内の SC12（Santa Clara Campus 第 12 ビル）にある MSL（Microcomputer Software Lab.）や MRL（Microprocessor Research Lab.）をはじめ、オレゴン州、イリノイ州、アリゾナ州等の米国内だけではなく、中国、イスラエル、ロシア等、世界中に点在する研究所群から構成されている。

（http://www.intel.com/labs/map.htm）

今回訪問したMSLは、主としてIA-64アーキテクチャ向けのコンパイラやツールの研究開発を行っている研究所で、オレゴン州の研究所のメンバも加えると、100名を越える研究者から構成されている。プロセッサ専業メーカにおけるソフトウェアの研究所は、小規模な研究グループに違いないと訪問前には想像していたが、実際には、日本の主要なコンピュータメーカの研究所の各研究部よりも大規模であるのに驚いた。

インテルの総社員数は2000年現在約65,000人で、近年ソフトウェア技術者の採用に力を入れているらしい。なぜ今インテルがソフトウェアの研究開発に力を注いでいるのか、その答えは、SC12建物内の至るところに貼ってあるポスターに書かれたスローガン（Our Mission 2001）に託されていると感じた。

「Preeminent building block supplier to the world-wide Internet economy」

すなわち、インテルはCPUの専業メーカであるが、インターネットエコノミーを支えるキラーアプリを考案／開発することを企業目標として掲げている。高性能CPUの開発は、そのための部品開発であるという位置付けであろう。昨年頃から日本のパソコンショップでは、インテル製のPC接続型の顕微鏡、デジカメやインターネットTV電話用のカメラ等が目に付くが、これらの製品群も、インターネットエコノミーのキラーアプリを生むための部品群の一つなのであろう。

では、そのキラーアプリとはどのようなものとなるのであろうか？ 単純に想像すると、デジタルビデオの編集や個人レベルの動画のストリーミング配信（いわゆるTV電話またはインターネット動画放送）のように、従来は高価な機材や高度な技術を必要とされてきた分野を、誰もが安価で簡単に使いこなせるようにするためのもの、ということになる。短期的には、このようなソフトがインターネットエコノミーを支える一つの分野を構成することは間違いないであろう。しかし、キラーアプリがどのような分野のものとなるかは全く想像がつかないのが面白いところであり、数年後の世界のインターネットエコノミーがどのようになっているか、大変楽しみである。

筆者は常日頃より「高性能計算機の研究開発を続けるためには、まずアプリケーションを開発せよ」と感じているが、インテル社の訪問を通して、研究開発における需要と供給の法則の重要性を再認識することができた。

図１　インテル本社ビル前にて、Mr. Hideki Saitoと古市委員（向かって左）

2. イリノイ大学 NCSA（アーバナ・シャンペン）

http://www.ncsa.uiuc.edu/

http://charm.cs.uiuc.edu/

＊NCSA: National Center for Supercomputing Application

＜訪問目的＞

米国におけるスーパーコンピュータアプリケーションとハイパフォーマンスコンピュータの現状と今後の動向。

＜面会者＞

Dr. Daniel Reed, Director of the NCSA,
Professor and Head of Dpartment of Computer Science

Dr. L.V. Kale, Director of Parallel Programming Lab.,
Professor, Department of Computer Science

Dr. Saburo Muroga, Professor, Department of Computer Science

NCSA Telnet や NCSA Mosaic 等、現在我々が使っているソフトを多数生み出したことで有名な NCSA であるが、本来の業務は、大規模な演算を必要とするアプリケーションの開発支援と実行環境の提供である。長年 Dr. Larry Smarr が NCSA の所長を務めてきたが、2000 年の秋に Dr. Smarr は UCSD（Univ.of California San Diego 校）に移籍した。その後任として、イリノイ大学の DCS（Department of Computer Science）の学部長である Dr. Daniel Reed が NCSA の所長を兼任している。2001 年 5 月 1 日付けで Dr. Daniel Reed は DCS のヘッドを退き、NCSA の専任所長となる予定である。

NCSA の業務は、1) 並列アプリケーションの開発支援と 2) HPC（High Performance Computer）の調達、開発と運用である。

NCSA に並列アプリの開発支援を依託しているのは、NIH（保健省）、DOE（エネルギー省）をはじめとする国の機関や、大手自動車メーカ、大手小売業、大手製薬メーカ、大手石油メーカ、大手化学メーカなど多数の民間企業である。かつては数値演算系のアプリケーションが多く、ベクタースパコンが大活躍していたが、現在ではデータマイニングに代表される非定型アプリケーションが多く、最も活躍しているのは SGI 社の Origin 2000 である。何式かはベクタースパコンもあるだろうと

図 2　NCSA の所長室にて、Dr. Daniel Reed、古市委員と小林研究員（中央）

予想していたが、現在 NCSA では現在ベクタースパコンを 1 台も保有していないとのことである。

NCSA が所有する多数の Origin 2000 の中で、最大構成のものは 512 プロセッサのシステムが一式で、他に 256 プロセッサのシステム約 10 式が 24 時間体制で稼動している。Dr. Daniel Reed によると、運用中の実アプリケーションの動特性を解析したところ、粗粒度のプロセスがメッセージパッシングによりデータ通信と同期を行うタイプのものが多く、「将来は全て Linux クラスタへ移行可能である」という大胆な意見をもらった。この意見は Dr. Daniel Reed の個人的なものではなく、今後 NCSA が計画している調達品は全て Linux ベースのクラスタであるとのことである。

Dr. Daniel Reed のインタビューを終えた後、テクニカルマネージャのブライアン氏の案内で、NCSA の計算機室とオペレーションルームを見学させてもらった。計算機室は 3 階建ての建物の 2 階と 3 階で、各フロア 50m x 50m 程度の通常の空調の部屋に Origin 2000 がぎっしりと設置されていた。

計算機室には、他にも多数の手作り Linux クラスタが並んでいた。更に奥には 2 月に搬入されたばかりで現在調整中の IBM の Linux クラスタが置かれ、IBM 社の技術者によって最終調整が行われていた。この IBM のクラスタは、1GHz のペンティアム III を 2 基搭載する eServer 330 を 1 ノードとし、ミリネット社のネットワークスイッチで 256 ノードから構成される。総プロセッサ数は 512 個で、ピーク性能は 1 テラ FLOPS、OS は Red Hat Linux である。

更に奥には、インテル社の Itanium 搭載パソコンをラックに多数積み上げて Linux クラスタを構成していた。このクラスタは本年夏に稼動予定の 800MHz Itanium を 2 基搭載するボードを 1 ノードとし、最大 160 ノードから構成するとのこと。ピーク性能は 1 テラ FLOPS で、OS は Turbo Linux である.

これらのクラスタに関する報道記事は以下に詳しく解説されている。

http://www.hotwired.co.jp/news/news/20010117301.html
http://biztech.nikkeibp.co.jp/wcs/show/leaf?CID=onair/biztech/comp/121213

今回、ブライアン氏の案内で、Itanium クラスタのために新築中のビルの工事現場の中に入れてもらうことができた。ビルは 3 階建ての体育館程の大きさであるが、中は 2 階建てとなっている。1 階には特注の空調システムが入っており、Itanium クラスタを設置する 2 階へ冷風を送る仕組みとなっている。320 個の Itanium の発熱を冷却するためには、これだけの施設が必要なのかと、正直言ってびっくりした。

NCSA のオペレーションルームでは、約 5 人のオペレータが 24 時間体制でシステムの稼働状況をモニタリングしている。今後システムが全て Linux クラスタとなった後、オペレーションルームの風景がどのように変化するか、楽しみである。オペレータのボスであるブライアンは、「Stable if boring、安定したシステム（Origin 2000）はつまらないよ」と言ったが、１年後にブライアンは何と言うだろう？

図 3　NCSA のオペレーションルーム

3. ジョージア工科大学 M&S センタ（アトランタ）

http://www.cc.gatech.edu/computing/pads/

＜訪問目的＞

並列分散シミュレーションの研究開発の現状と、ハイパフォーマンスコンピューテングへの HLA 適用の現状と将来に関してする調査。

＜面会者＞

Dr. Richard M. Fujimoto, Director of Modeling and Simulation Center,
Professor, College of Computing

Mr. Thom McLean, PhD candidate

Dr. Fujimoto は PDES（Parallel Distributed Simulation）研究の世界的第一人者で、ACM 学会の Transaction on Modeling and Computer Simulation の編集を務めている他、1990 年からは IEEE と ACM 共催で毎年開催される国際会議 PADS（Parallel and Distributed Simulation）のステアリングコミッティを務めている。Dr. Fujimoto を更に有名にしているのは、DoD（米国防総省）が 1995 年に提案した HLA（High Level Architecture）の仕様検討委員会である AMG（Architecture Management Group）のメンバとして、HLA 仕様の中核を設計したことである。HLA のアカデミック側からの最も強力なシンパとして、DoD 及び HLA コミュニティメンバから高い信頼を集めている。

2000 年 9 月に HLA は IEEE 1515 としての標準化が完了し、今後分散シミュレータの接続仕様として広く利用されていくことになると予想される。しかし、HLA は元来異機種シミュレータをネットワークを介して統合するために出てきた標準仕様であり、ハイパフォーマンスコンピューティングのためのプログラム開発実行環境としての適

用を考えている研究者は少ない。Dr. Fujimoto はそのような研究者の一人であり、HLA 分散シミュレーションの実行時に必須となるソフトウェア RTI（Run-Time Infrastructure）と、その上のアプリケーションの研究開発を行っている。最近の研究成果である PDNS（Parallel and Distributed Network Simulator）は、コンピュータネットワークのシミュレータとして広く普及している逐次型の NS2（Network Simulator）をベースに、HLA 仕様で拡張することにより分散並列化したシミュレータである。この PDNS はネットワークシミュレーションにおけるキラーアプリとなり得る可能性を秘めており、HLA の実用化に大きく貢献していると考えられている。

これまで HLA は仕様の標準化が最も大きな課題であった。しかし、標準化を完了した後の課題は、HLA をベースとした多くのアプリケーションを開発し、多くのユーザに提供することであり、Dr. Fujimoto は NS2 を題材としてこれを実現しようとしている。今後 HLA が更に普及し、ネットワークシミュレータ以外にも同様のアプリケーションが多数出てくることを Dr. Fujimoto は願っているとのことである。筆者は、そのようなアプリの一つが分散協調型のエージェント技術であると確信しており、そこでミシガン大学の AI ラボと南カリフォルニア大学を訪問した。

図４　Dr. Fujimoto, Mr. McLean と共に

4. ミシガン大学 AI ラボ（アンナーバ）　http://ai.eecs.umich.edu/

＜訪問目的＞

ルールベースの知的エージェント開発実行環境 Soar の研究開発動向と、官学民共同プロジェクトの成功例の調査。

＜面会者＞

Dr. Mike van Lent, Post-doctor research assistant

Mr. Scott Wallace, Ph.D. Candidate

日本で 80 年代にブームとなった AI（人工知能）の研究は、米国のいくつかの大学ではかつて以上に活発に研究が行われている。人工知能の研究分野でも、工学よりの知的エージェントや分散協調エージェントの分野の研究開発が最も盛んである。

ミシガン大学アンナーバ校の AI ラボは、米国防総省からの研究開発資金を獲得して最も成功している大学であり、エージェント開発実行環境である Soar（the basic cycle of taking a State applying an Operator, And generating Result の頭文字を取ったものであるらしい）の開発元である。米軍が保有する訓練用のシミュレーションシステムの多くでは、現実の人間に近い意思決定を行うための仕組みとして Soar を用いた知的エージェントを利用している。AI ラボの運営はほとんど軍からの研究依頼でまかなっているようである。

知的エージェントの開発上で最も重要なのは、現場の専門家の知識をいかにしてルール化するか、という部分で、このための手法を知識獲得という。AI ラボでは様々な知識獲得手法の研究をしているが、検証のためには学生が直接米空軍のパイロットにインタビューしてルール化し、その結果をパイロットに評価してもらうところまで、大学が実際に行ったとのことである。

このように、現在軍向けの研究が最も比重が重いようであるが、大学院の学生の間で最も人気があるのは、エンタテイメント用のゲームのインテリジェント化らしい。例えば、Quake というゲームの敵側の登場人物を Soar により自動化したデモを見せてもらったが、人間同士で対戦するよりも断然強い敵が実際に実現できていた。近いうちにユーザが Soar を用いてユーザが独自に拡張可能なゲームが出現するのではないかと予想される。また、学部の学生に最も人気のある授業はゲームプログラミングという授業らしく、AI ラボはミシガン大学でも最も人気のある研究室の 1 つらしい。

AI ラボ訪問の後、Soar の考案者らが中心となって 1998 年に設立したベンチャー企業 Soar Technology Inc. のオフィスを訪問した。社長の Dr. Jim Rosbe 氏によると、設立した当時の社員は数名だったらしいが、軍からの知的エージェント開発のプロジェクトは繁盛していて、今日現在社員は 17 人。まだまだ増やしたいと語っていた。

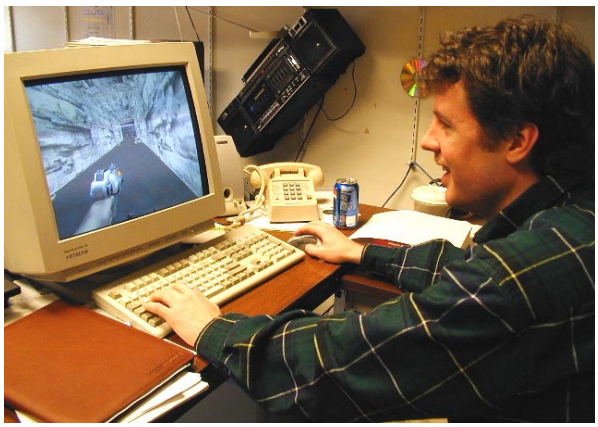


図 5　Dr. Mike Lent が操作するゲーム Quake の敵は Soar による知的エージェント

ミシガン訪問は、AI の研究は軍用のシミュレーション技術とエンタテイメント技術の両面から近い将来ブレイクするのではないか、と予感させた。続いて訪問した南カリフォルニア大学では、更にこの予感を実感することとなった。

5. 南カリフォルニア大学 ISI と ICT（ロスアンジェルス）

http://www.isi.edu/

http://www.ict.usc.edu/

*ISI:　Information Sciences Institute

*ICT:　Institute for Creative Technologies

＜訪問目的＞

ルールベースの知的エージェント開発実行環境 Soar を応用したシステムの研究開発動向の調査。特に、米陸軍とハリウッド映画業界の共同研究施設における分散協調エージェントの研究方法の調査

＜面会者＞

Dr. Paul S. Rosenbloom, Director of New Directions,
Deputy Director of Intelligent Systems Div.

Dr. Lewis Johnson, Director of CARTE, Professor, ISI

Dr. Milind Tambe, Research Associate Professor, ISI

Dr. Johathan Gratch, Project Leader, ICT

ISI は約 30 年前に米 RAND 社のメンバ 3 人が中心となって南カリフォルニア大学（USC）の College of Engineering の組織の１つとして設立された研究所である。USC の本体はロスアンジェルスのダウンタウンのど真ん中に位置するが、ISI はベニスビーチのすぐ隣にあるマリナ・デル・レイでもひときわ眺望の良い場所に位置する。なぜキャンパスに作らなかったのかと問うと、軍の予算で設立した研究所のため、物理的に異なる場所に作ったとの答えだった。

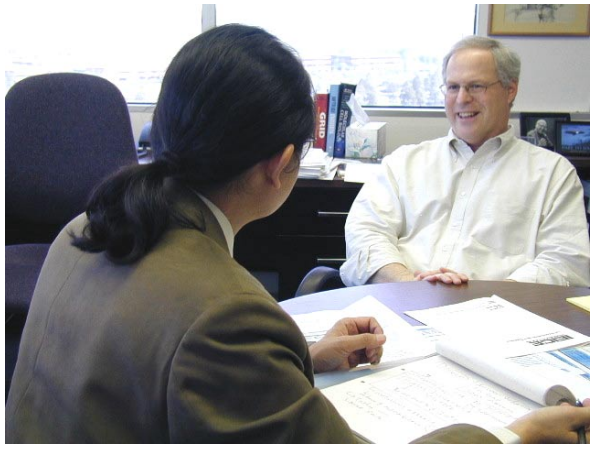


図 6　Dr. Paul Rosenbloom

Dr. Paul Rosenbloom は、カーネギーメロン大学時代に現ミシガン大学の Dr. John E. Laird らと一緒に Soar を考案した人物である。Dr. Paul Rosenbloom によると、Soar がここまで発展してきたのは全て米国防総省の長期的な研究資金サポートがあってのことだという。そして近年、その技術をゲーム業界やハリウッドの映画業界が活用しようとしている動きに、大変満足しているとのことだ。ただ、Dr. Paul Rosenbloom 自身は数年前から人工知能の第１線の研究からは退かれたそうで、現在は Director of New

Directions の職務で忙しい。New Directions の中では、Virtual Human、Virtual Organization、Rapid Prototyping の応用に関するプロジェクトの立ち上げが検討されているとのことであった。

ISI で続けて訪問した Dr. Milind Tambe 氏は、ロボカップサッカーの世界大会で毎回上位入賞を果たしているチームのリーダである。ロボカップはあくまでも研究の成果をアピールする場の一つでしかなく、氏の研究の主要テーマは、マルチセンサによるマルチムービングターゲットの認識と追尾であり、研究依頼元は米国防総省である。ここでも、軍用技術とエンタテイメント技術との奇妙な融合に大変驚かされた。

続いて訪問した ICT は、米陸軍が 4430 万ドルを投じて 1999 年末に南カリフォルニア大学に設立した研究所で、先に訪問した ISI とは道を隔てた向かいのビルに位置する。ICT は南カリフォルニア大学の映画・テレビ学部（フランシスコッポラ監督が卒業したことでも有名）を中心に、複数の学部のメンバから構成される研究組織であり、所長はハリウッドのパラマウントテレビジョンの Dick Rindheim 氏である。2000 年度の研究予算は 2200 万ドルで、全て米陸軍から出ている。

ICT の目標は、陸軍の兵士に対してリアルな訓練環境を提供するための技術開発であり、最新のエージェント技術と VR（仮想現実）技術を融合することにより、どこまでリアルな訓練環境を実現できるかを試すことである。Dr. Jonathan Gratch は ICT の中のエージェント技術開発プロジェクトのリーダで、Soar を用いて人間の感情表現の研究を中心に現在行っている。氏に対するインタビューの後、陸軍によるユーゴスラビア平和維持活動をシナリオとしたミッションリハーサルのデモを見せてもらった。

ミッションリハーサル室には円筒型の 300 インチの巨大スクリーンがあり、その中ではリアルなユーゴスラビアの街路が表現されている。ミッションのプロローグで、交差点で乗用車が少年をはねるという交通事故が発生する。はねられた少年は路上に倒れ、一緒に居た母親は呆然とする。そこへ PKO 活動中の兵士が続々と集まるところから訓練は始まる。

このミッションリハーサルの中では、観客のうち一人はコマンダ役となって、兵士達に自然言語（英語）で命令を発する。すると、スクリーン上の兵士（Soar で記述されたエージェント）は命令を理解してそれぞれの行動をとる。

図 7　Dr. Jonathan Gritch

1 回目のリハーサルでは、事故処理のために兵士を 2 人残し、他の隊員を他の現場へと派遣する。すると、はねられた少年の母親は正気を失っ

て兵士に反抗し、事故処理をスムーズに行うことができない。

そこで、事故が発生した時点までシミュレーションを一度ロールバックさせ、2 回目のリハーサルを行った。今度は事故処理のために兵士を多数配置することにより母親を安心させることができ、その結果スムーズに事故処理を行うことができた。

図 8　ミッションリハーサル室の円筒形スクリーンに投影された画像

このような訓練は、従来はテキストやマニュアルを読むことによって行ってきたわけだが、実際に現場の各人間の感情や行動を画像を見て体験することにより臨場感を高めることができ、より訓練の効果が増すことを実体験することができた。

ハリウッドでは、将来この技術を映画製作に利用するだろうが、それよりも大きな市場は、やはりゲームであろう。既にゲームメーカが ICT への投資を開始している。AI とハイパフォーマンスコンピューティング技術は、現在軍用の技術としての実用化が促進されている。それとともに、更には我々の身近なゲームやハリウッド映画に適用される時がすぐそこまでやってきているようである。

まとめ

今回の海外調査を通して得られた結論を 2 つにまとめると、以下の通りとなる。

今後のハイパフォーマンスコンピュータの主流は Linux クラスタとなる。

米国では 80 年代の人工知能ブーム以降も着実に人工知能の研究が続けられて来た。90 年代に入って分散協調型エージェント技術として人工知能技術の一部は実用化し、2000 年代には軍用技術とエンタテイメント技術が融合する勢いで研究が加速している。

Soar プロジェクトは、米国における官学民共同プロジェクトの中で大変うまくいった例の一つとして参考になる。

付属資料４　米国におけるハイエンドコンピューティング関連

付表4.1　HECC インフラストラクチャとアプリケーション

分類	研究テーマ	管轄／研究所	内容	その他
コンピューティング環境とツールキット				
	アーキテクチャ適応 コンピューティング環境	NASA	データ並列コンピューティング	構造的並列処理、LINUX
	Kelp	NSF/NPACI,UCSD	ポータブル科学アプリケーション	分散メモリ並列コンピュータ、データ並列
	不規則科学向け並列アルゴリズムと アプリケーションソフトウエア	NSF/ニューメキシコ大	並列コンピューティング	計算機科学
	先進的テストおよびシミュレーション ツールキット	DOE	数値解析法、ソルバ、数値ツール	MPI使用
	スケーラブルなビジュアライゼーション ツールキット	NSF/NPACI	大規模データセットの視覚化	大規模シミュレーション
	幾何学的問題を研究するためのツール	NSF/NYCポリテック大	計算幾何学	幾何学的アルゴリズム
モデリングツール				
	NASA地球システムモデリングフレーム ワーク	NASA	再利用、移植性、修正の容易性	ESS計画に利用
	マイクロマグネティックモデリング	NIST	マイクロマグネティック計算ツール	オブジェクト指向フレームワーク
	現実的材料のミクロ構造のモデリング	NIST	オブジェクト指向有限要素ソフトウエア	使い易いGUI
	環境モデリングのための数値 ／データ処理	EPA	数値アルゴリズムの高速化	並列コンピューティング
HECCアプリケーション				
生物医学アプリケーション				
	神経科学イメージング	NIH	脳、神経システムの構造解析	シナプスの詳細モデル
	Mcell	NSF/NPACI、UCSD	細胞生理学向けモンテカルロ シミュレータ	NetSolveクラスタで利用可能
	タンパク質の折り畳み	NIH/Alliance	相互作用シミュレーション	計算生物学
	ポータブルな生物医学情報の検索と融合	NSF、NCI/NCSA	ポータブルな情報検索プログラム	Emergeトランスレータ
航空宇宙アプリケーション				
	Computational Aerosciences	NASA	航空機設計時間短縮	1000倍のパフォーマンス向上追求
	地球および宇宙科学計画（ESS）	NASA	相対論的宇宙物理学から地球気象	高密度中性子星の爆発
応用化学				
	燃焼の理解	DOE/LBNL	化石燃料の燃焼予測モデル	メタン燃焼研究に光をもたらす
量子物理学アプリケーション				
	電子の衝突によるイオン化	DOE/LBNL、LLNL、 UCD	量子システムの散乱現象	水素原子のイオン化を解明
気象アプリケーション				
	ハリケーンの強度予測	NOAA/GFDL	数値シミュレーションでの流体力学	大規模ハリケーンで課題があり
	ハリケーンと地球温暖化	NOAA/GFDL	地球気象モデルの高解像度化	ハリケーン予測モデルとの関連付け

付表4.2　HECC　研究と開発

分類	研究テーマ	管轄／研究所	内容	その他
HTMT	ハイブリッド技術マルチスレッド	DARPA,NASA,NSA,NSF	ペタフロップス規模コンピュータ	超伝導ロジック、光通信
Beowulf	ワークステーションクラスタとLinuxによる高性能	NASA	Linuxをベースにソフトウエアを開発	Red Hut Linuxの採用
	混在する分散共有メモリ環境での気象予報	NOAA	MPIベース気象予測プログラム	グローバル・スペクトル・モデル
	バーチャルインターフェースアーキテクチャのためのMVICH-MPI	DOE,NERSC	クラスタ向けポータブル高性能通信	クラスタ用ネットワークの業界標準
	分散ストレージシステム(DPSS)	DOE,DARPA	ネットワーク対応ストライプ・ディスクアレイ	低コスト汎用デバイスの採用
	Globus	DARPA,DOE,NSF	Gridの統合実行環境	Webアクセスのような操作性を目指す
	Globus：スマートイスツールメントアプリケーション	DARPA,DOE,NSF	大規模分散対話シミュレーション	対話型戦車シミュレーション
	Legion：ワールドワイドなバーチャルコンピュータ	DOD,NSF,DOE/バージニア大学	オブジェクト指向メタシステム・ソフトウエア	共有仮想ワークスペース
	JavaNumerics	NIST	数値ソフトウエアコンポーネント	Java Grande Forumとの連携
高度ハードウエアコンポーネントの研究（現行研究では不可能なものを目指す。）				
	大規模集積回路フォトニクスプログラム	DARPA	光学特性をチップ相互接続に	通信帯域の確保
	量子コンピューティング	NSA	シリコンベース原子スピン量子コン	超伝導量子ビットデコヒーレンス性
	量子位相によるデータの保存と検索	NSF/ミシガン大学	データベースに原子の量子位相利用	セシウム原子で研究
	量子情報とコンピュテーション	DARPA/CaTec,MIT,USC	量子ゲート、量子デバイス、量子コンピュータ	概念、調査、実験研究
	アンサンブル量子コンピュータの核磁気共鳴（NMR）	DARPA	NMR分光の超微視的観察アンサンブル平均	実装方法、エラー修正
	DNAによるデータ保存	DARPA,NSF/デューク大学	DNA組み換え技術の利用	NP探索問題の解決
	集積回路開発のための高度な顕微鏡ツール	NSA	表面直接イメージング機能	原子力顕微鏡
	3次元マルチチップモジュールによるキューブコンピュータ	NSA,DARPA	ダイヤモンドベース5層スタック3次元	エアゾール・スプレー冷却
	光テープ	NSA,NASA,DOE	１テラバイトデータをカセットテープに	LOTSテープドライブの改良も
	超伝導エレクトロニクス	NSA	超伝導クロスバースイッチ	HTMTでの利用
	スマートメモリ	NSA	コンピュテーショナル・タイル	ワイヤリング再構成可能チップ
	ベンダーの協力	NSA	SGI/Cray, Compaq, Sun	
	分子電子工学	DARPA	分子／ナノ粒子による電子デバイス	高度な計算能力、高密度メモリ
	ナノクリスタルデバイス	NSF	NC合成による薄膜	電子スイッチング、ストレージデバイス
今後の構想				
バーチャル科学の測定とキャリブレーション		NIST	モデリングとシミュレーションに基づく体系	テスト、キャリブレーションのツール
DOEの戦略的コンピューティング加速機構(ASCI)		DOE/LANL, LLNL, SNL		
	概要		2004年までに基本部分の実装	2010年までにR&Dの完全実装
	PathForward		30TFlopsコンピュータ、1Tバイト記憶装置	
	ASCIコンピューティングプラットフォーム		Option White 10 TFlops	2004年に100TFlops
	Virtual Interructive Environment for Weapons Simulation (Views)		高レベルの科学的データ分析機能	可視化サーバ
	科学データ管理計画		メタデータによる共通データモデル	Data Discovery等多数のツール開発
	問題解決環境		ASCI分散コンピューティング環境	

付表4.3　IT R&Dの施設

管轄	補足	大学／研究所	施設／センタ	その他
NSF	PACI (Partnership for Advanced Computational Infrastracture)			Gridを構築 実験アーキテクチャ、イネーブリング技術
	Allience (National Computational Science Allience)			分散共有メモリ、PCクラスタ
		イリノイ大学/UIUC	National Center for Supercomputer Application	
		ボストン大学		
		ケンタッキー大学		
		ニューメキシコ大学	アルバカーキHPCC	
		ニューメキシコ大学	マウイHPCC	
		ウィスコンシン大学		
	NPACI (National Partnership for Advanced Computational Infrastracture)			全米のコンピュータインフラの構築
		サンディエゴ大/UCSD	San Diego Supercomputer Center	
		テキサス大学		
		ミシガン大学		
		カリフォルニア工科大学	及びNASAジェット推進研究所	HTMT推進
NSF他	NCAR（国立大気研究センタ）			大気科学、海洋科学
NASA	米国航空宇宙局		Ames研究センタ	カリフォルニア州モフェット・フィールド
			Glenn研究センタ	オハイオ州クリーブランド
			Goddard宇宙航空センタ	メリーランド州グリーンベルト
			ジェット推進研所	カリフォルニア州パサディナ
			Langley研究センタ	バージニア州ラングレー
DOE	米国エネルギー省/科学局	LBNL	NERSC	国立エネルギー研究所スーパーコンピュータセンタ
		ANL		
		LALN	Advanced Computing Lab	ASCI Blue Mountine (SGI) 1.6TFlops
		ORNL		
		PNNL		
NIH	米国国立衛生研究所			
	Center for Information Technology			
	National Center for Research Resource		NCRR Computing Resource Center	
			NCRR Scientific Visualization Center	
NOAA	商務省海洋大気局		Forecast System Lab	コロラド州ボルダー
			Geophysical Fluid Dynamic Lab	ニュージャージ州プリンストン
EPA	米国環境保護庁		National Environmental Supercomputer Center	ノースカロライナ州

付表 4.4　2001 年度予算要求

(単位:100 万ドル)

	ハイエンド・コンピューティング基盤およびアプリケーション	ハイエンド・コンピューティング研究開発	ヒューマン・コンピュータ・インターフェイスおよび情報管理	大規模ネットワーク	ソフトウェアの設計および生産性	高信頼ソフトウェアおよびシステム	社会、経済、および労働力の面から見た IT 労働力開発	
関係機関	(HECI&A)	(HEC R&D)	(HCI&IM)	(LSN)[a]	(SDP)	(HCSS)	(SEW)	合計
NSF	285.2	102.1	135.8	111.2	39.5	20.5	45.3	740
DARPA	54.6	56.5	48.0	85.3	55.0	8.0	0.0	307
NASA	129.1	25.8	17.9	19.5	20.0	9.1	8.3	230
NIH	34.5	3.4	99.6	65.6	0.7	6.5	7.0	217
DOE Office of Science	106.0	30.5	16.6	32.0	0.0	0.0	4.6	190
NSA	0.0	32.9	0.0	1.9	0.0	44.7	0.0	80
NIST	3.5	0.0	6.2	4.2	2.0	8.5	0.0	24
NOAA	13.3	1.8	0.5	2.7	1.5	0.0	0.0	20
AHRQ	0.0	0.0	8.1	7.4	0.0	0.0	0.0	16
OSD/URI	0.0	2.0	2.0	4.0	1.0	1.0	0.0	10
EPA	3.6	0.0	0.0	0.0	0.6	0.0	0.0	4
小計	629.8	254.9	334.7	333.8	120.3	98.3	65.2	1,838
DOE ASCI	131.8	36.5	0.0	35.0	40.2	0.0	55.7	299
合計	761.6	291.4	334.7	368.8	160.5	98.3	120.9	2,137

(注)

・本表は BlueBook2001 から抜粋したもの。

・NGI の予算は LSN 予算の一部。

・切り捨てのため小計の和は合計と一致しない。

・これらの合計は大統領の 2001 年度の予算とは異なる。

付表4.5　Top10　High Performance Computer

順位	メーカ	マシン	Rmax (GFlops)	設置場所	国	設置年	研究分野	プロセッサ数	Rpeak (GFlops)
1	IBM	ASCI White, SP Power3 375 MHz	4938	Lawrence Livermore National Laboratory (Livermore)	USA	2000	Research Energy	8192	12288
2	Intel	ASCI Red	2379	Sandia National Labs (Albuquerque)	USA	1999	Research	9632	3207
3	IBM	ASCI Blue-Pacific SST, IBM SP 604e	2144	Lawrence Livermore National Laboratory (Livermore)	USA	1999	Research Energy	5808	3868
4	SGI	ASCI Blue Mountain	1608	Los Alamos National Laboratory (Los Alamo)	USA	1998	Research	6144	3072
5	IBM	SP Power3 375 MHz	1417	Naval Oceanographic Office (NAVOCEANO) (Bay Saint Louis)	USA	2000	Research Aerospace	1336	2004
6	IBM	SP Power3 375 MHz	1179	National Centers for Environmental Prediction (Camp Spring)	USA	2000	Research Weather	1104	1656
7	Hitachi	SR8000-F1/112	1035	Leibniz Rechenzentrum (Muenchen)	Germany	2000	Academic	112	1344
8	IBM	SP Power3 375 MHz 8 way	929	UCSD/San Diego Supercomputer Center (San Diego)	USA	2000	Research	1152	1728
9	Hitachi	SR8000-F1/100	917	High Energy Accelerator Research Organization /KEK (Tsukuba)	Japan	2000	Research	100	1200
10	Cray	T3E1200	892	Government	USA	1998	Classified	1084	1300.8

（注）　本資料は　http://www.top500.org/list/2000/11/から入手したもの。
2000年11月時点のもの。SC2000に先立ち発表された。
性能はLINPACKで計測され、Rpeakはピーク性能、Rmaxは実効性能でRmaxの順にソートしてある。

付属資料５　平成１２年度ワーキンググループ活動記録

開催場所：先端情報技術研究所（AITEC）内会議室
開催時間：18:00　～　　（第 5 回のみ 17:00　～　）

開催日	討議内容
第 1 回 2000.10.03	・山口主査　WG 今期活動方針説明 ・WG 幹事「平成 13 年度活動方針案」 ・新部講師　三菱総研　主任研究員　（注 1） 「GNU/LINUX の最近の動向」 ・Blue Book の紹介（英語版の配布）
第 2 回 2000.11.20	・門林講師　奈良先端大学　助教授　（注 2） 「グローバルコンピューティングにおけるセキュリティ技術」 ・田中委員 「スーパーコンピュータのハードウエアアーキテクチャとコンパイラのはざま」 ・福井委員「SC2000 印象記」
第 3 回 2000.12.18	・岸本講師「インフィニバンド・アーキテクチャ」　（注 3） ・近山委員「コードを短くする方法」
第 4 回 2001.01.30	・笠原委員 「通産省（NEDO）ミレニアムプロジェクト：アドバンスト並列コンパイラプロジェクトの概要」 ・妹尾委員 「並列化ツールの最近の話題について」
第 5 回 2001.02.27	・天野委員 「いよいよ商業化を迎えた Reconfigurable System」 ・報告書の書き方について。

（注 1）新部講師は当時招待で、その後ワーキンググループ員になっていただいた。
（注 2）招待講師、講演内容をベースに報告書として纏めていただいた。
（注 3）招待講師、講演内容をベースに報告書として纏めていただいた。

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
―　高性能プラットフォーム技術を中心とする　―
©平成１３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
芝東京海上ビルディング４階
TEL(03)3456-2511