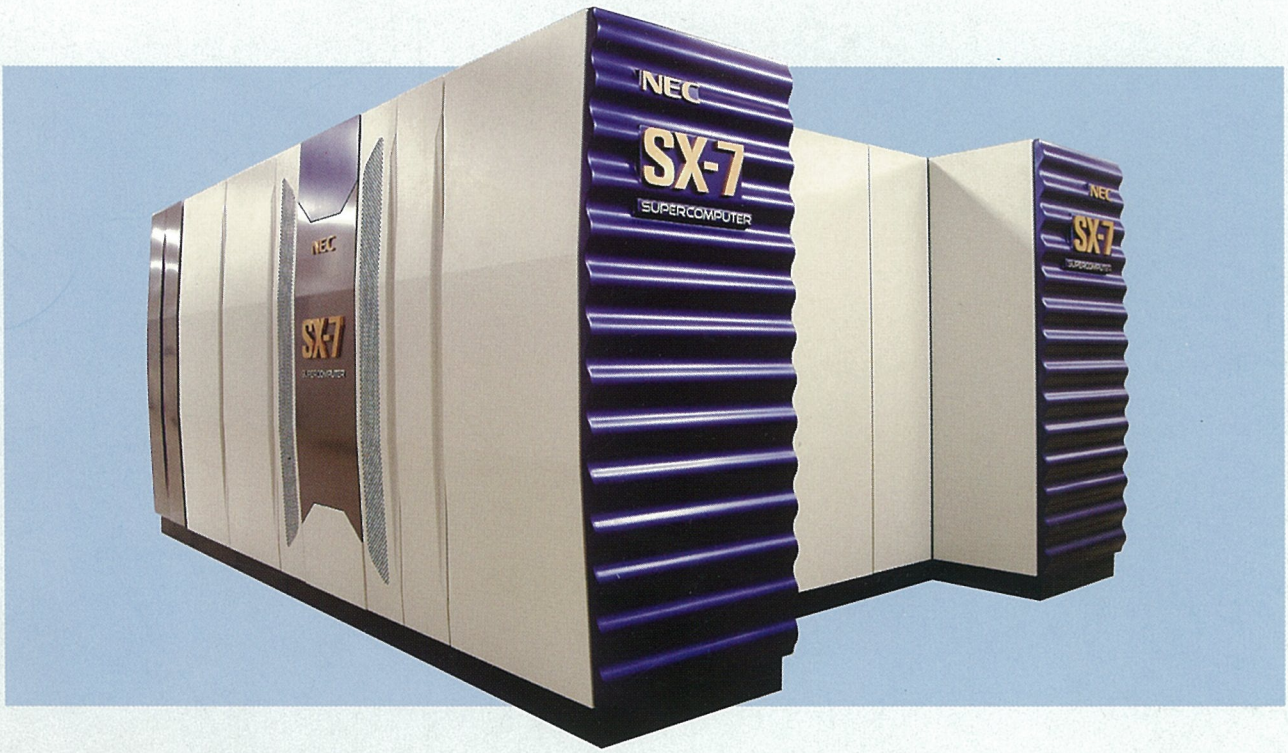
高速化推進研究活動報告

〈第２号〉



東北大学情報シナジーセンター

2003年3月

目　次

1　高速化推進研究活動報告（第2号）の刊行にあたって

センター長　根元　義章

東北大学情報シナジーセンターは、1969年に、大学の教員、その他の研究者が学術研究等のために利用する全国共同利用施設として設置され、30数年が経過した。本センターは、わが国の計算機の揺籃期に、期待をもって誕生し、時代とともに進展をとげ、学術研究の発展に大きく貢献してきた。これも文部科学省をはじめ、関係各位のご支援、ご協力の賜物であり、厚く感謝申し上げる。

本センターの目的は、「最先端かつ世界最大級のコンピュータシステムの導入と利用環境の構築」ということに尽きる。計算機の揺籃期においては、大型計算機センター、計算機メーカ、利用者が一体となって、コンピュータのハードウェア、ソフトウェア、プログラミング言語、アプリケーション等を開発した。また、これら最先端の機器や技術を使いこなすために、上記3者が協力して利用環境を整備し、さらにわかりやすいマニュアルの作成やプログラミング相談、講習会等で利用技術等の普及に努めてきた。その後も現在にいたるまで、種々の情報技術やネットワーク技術の進展にあわせて、上記3者の協力による情報技術の開発が進められてきている。

本センターの大規模科学計算システムとしては、ベクトル並列型の大規模処理に適したスーパーコンピュータと汎用処理指向の並列コンピュータの2つが設置されている。2003年1月から運用を開始したスーパーコンピュータSX-7は、科学技術計算用の高速コンピュータとして1986年に第1号機SX-1が導入されて以来、5代目のスーパーコンピュータである。また2002年1月から運用している並列コンピュータTX7/AzusAは、以前の大型汎用コンピュータの流れを受け、さらに機能・性能を向上させたシステムで、8代目となる。これにより情報シナジーセンターの大規模科学計算システムは、並列コンピュータ358GFLOPS, スーパーコンピュータ2119GFLOPSの性能を持つ世界で有数のシステムとなっている。

一方、大学の利用者は、より規模の大きい3次元シミュレーションを高速に行うことを切望しており、そのためにはシステムの持つ潜在的な処理能力を最大限に引き出すことができるように計算方法やプログラミングを工夫する必要がある。そこで本センターでは1997年より、利用者、本センター、日本電気㈱の3者により高速化推進のための研究会を立ち上げ、プログラムの高速化手法や新しい計算モデルに関する共同研究を進め、その結果を2001年3月に高速化推進活動報告として刊行した。第2号となる本報告は、2001年4月から現在に至るまでの約2年の間に行った研究・開発の成果をまとめたものである。第1号の刊行から今日まで、大規模科学計算をとりまく環境も変化した。また利用者の常に増大する計算要求に応えるために、本センターでは今後も継続的に本活動を実施していくことがますます重要であることを痛感している。本共同研究で、本センターの技術系の職員を中心とするスタッフが中心的役割を果たしたことを特に記しておきたい。

最後に、本研究会に積極的に参加し、目覚しい成果を挙げていただいた、本センター利用者、日本電気㈱側のスタッフならびに本センター側スタッフに厚くお礼を申し上げる。

2　高速化推進研究活動報告

スーパーコンピューティング研究部　教授　小林　広明

2．1　はじめに

ライフサイエンス、情報通信、環境・エネルギー、ナノテクノロジー・材料、産業基盤、安全工学、航空・宇宙工学などの先端科学分野において、計算機シミュレーションは理論、および実験と並んで重要な役割を演じている。半導体技術、コンピュータ構成技術、そしてソフトウェア技術の進歩に伴い、コンピュータの機能、および性能は飛躍的に向上してきたが、その潜在的な処理能力の恩恵を最大限に享受し、大規模・高精度・高速計算機シミュレーションを実現するためには、コンピュータシステム、およびプログラミング技術に関する高度な専門知識を必要とするのが現状である。

東北大学情報シナジーセンターでは、常に最先端の大規模科学計算システムの導入整備を行い、センター教官・技官、利用者、日本電気㈱が共同でプログラムの高速化技術および新しいシミュレーション技術に関する研究・開発を行ってきた。本節では、スーパーコンピュータ、および計算機シミュレーションに関する技術動向と、本センターが取組む高速化推進研究活動について説明する。

2．2　シミュレーションとコンピュータの処理能力

1976 年に Seymour Cray がスーパーコンピュータという新しいコンピュータのカテゴリに属する科学技術計算用コンピュータを世に送り出して以来、スーパーコンピュータは、その時代の最高性能を有するコンピュータと位置付けられるとともに、先端科学分野での必要不可欠な解析・実験装置として受け入れられてきた。スーパーコンピュータの高性能性は、

- 「18 ヶ月で半導体チップ上に集積されるトランジスタ数は 2 倍」という Moore の法則により技術革新を遂げる半導体技術の進歩
- 大規模科学技術計算に内在する数値計算に高度に特化されたベクトルパイプライン方式、および並列処理方式の導入
- 自動ベクトル化、自動並列化、およびに並列プログラムライブラリの整備といったハードウェアの性能を効率よく引き出すためソフトウェア技術の発達

によることが大きい。

スーパーコンピュータの性能向上に応じて、シミュレーション技術も、大規模化、高精度化が進んでいる。一般に、コンピュータシミュレーションでは、計算機上でモデル化したい対象を複数の点の集合で表現する。すなわち、各点での離散的な値（代表値）をあるモデルに基づいた数値計算により求め、離散点でモデル化した対象の力学的性質やエネルギー分布などの現象を解析する。従って、対象を表現する点の数が多いほど、より高精度に対象の振舞いを計算することができる。

シミュレーションのモデルを構築するための点の設定には様々な方法があるが、よく用いられるのは、図 2.1 に示すように、点を 1 次元、2 次元、3 次元の格子状に等間隔に配置する方法である（各点を格子点と呼ぶ）。1 次元モデルでは、ある一定方向のみに格子点を配置し、横のつながりだけで対象を表現する。2 次元モデルでは対象を面として表現し、2 方向に格子点を配置する。3 次元では、対象を縦、横、高さといった 3 次元

でとらえるために、3 方向に格子点を配置する。各次元方向に N 個の点を配置すると、1 次元ではＮ点、2 次元ではＮ×Ｎ個、3 次元ではＮ×Ｎ×Ｎ個の格子点を必要とする。

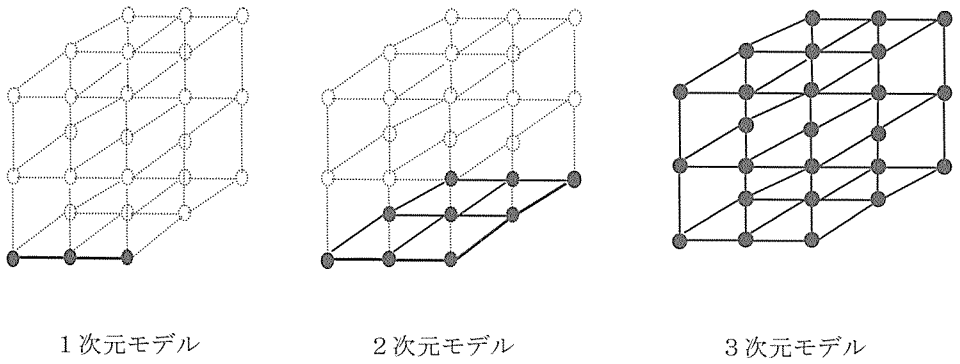


図2.1　シミュレーションモデル

次元ごとに定義する格子点の数を増加させることと、次元数そのものを増加させることにより、シミュレーションモデルは大規模な現象を高い精度で解析することができるようになる。しかしながら、モデルの大規模化、高精度化に伴い、計算量、および必要メモリ量も著しく増加する。ここで、各点で必要とする計算量、およびメモリ量がモデルの格子点の次元数によらずに一定であると仮定すると、図 2.2 に示すように 2 次元、3 次元モデルでは格子点の数の増加に伴い必要計算量、および必要メモリ量は急速に増加する。従って、コンピュータの処理能力に応じて、扱うことができるシミュレーションモデルが制限される。逆に言えば、より高性能のスーパーコンピュータを利用すれば、設定する格子点の数を増加させ、併せて点の配置を 1 次元から 2 次元、2 次元から 3 次元へとシミュレーションモデルの規模を拡大していくことができる。

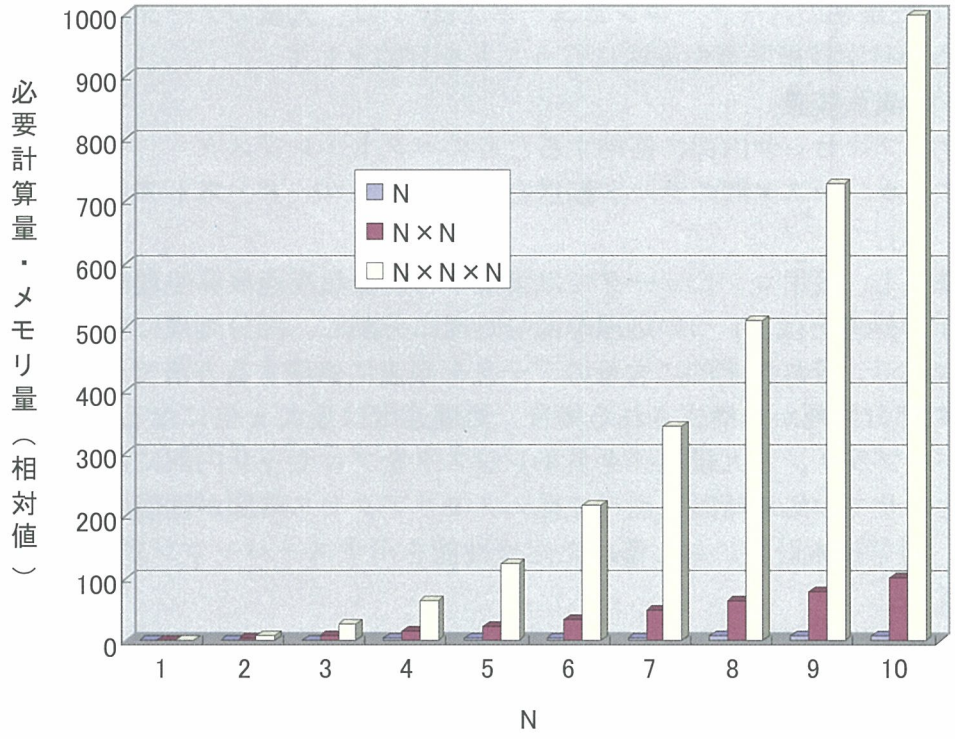


図 2.2　シミュレーションモデルの精度と計算量・メモリ量の関係

東北大学情報シナジーセンターでは、最新のスーパーコンピュータ SX-7 を 2003 年 1 月に導入し、運用を開始しているが、新しい計算方法の開発やプログラムの高速化のための種々の工夫により、中規模 3 次元モデルによるシミュレーションが可能な状況になってきた。

しかし、今後さらにより詳細な現象をとらえるようにシミュレーションモデルを大規模化、高精度化していくためには、常に最新のスーパーコンピュータを利用して単位時間当たりに扱うことのできる計算量とメモリ量の両方を向上させることに加え、3 次元モデルであれば計算量、およびメモリ量を入力データ数の 3 乗（時間軸をも考慮することになれば 4 乗）に比例して増加させる格子点を適切に設定したシミュレーションモデルの研究・開発、そしてシステムの潜在的な性能を最大限に引き出すための高度な計算モデル、およびプログラミング技術の研究・開発が必要不可欠である。

２．３　スーパーコンピュータの技術動向

スーパーコンピュータはその時代の最高性能を維持するために、様々な革新的処理方式を導入し、進化してきた。図 2.3 にスーパーコンピュータの構成方式の発展の流れを示す。以下、スーパーコンピュータの誕生から現在に至るまでのスーパーコンピュータの特徴を簡単に説明する。

（１）ベクトルプロセッサ（Vector Processor）

1976 年に誕生した最初のスーパーコンピュータ Cray-1 は、大量のデータに対する規則的な処理が中心の科学技術計算を高速に行うことを目的として、

- パイプライン化演算装置
- 大量のデータをプロセッサ内部に格納するためのベクトルレジスタ
- メモリとベクトルレジスタ間のデータ転送を高速に行うロード・ストアパイプライン

を持つことを特徴とし、汎用コンピュータでは実現不可能な超高速演算処理能力を実現した。パイプライン処理とは、1 つの処理を部分処理に分割し、部分処理に専用の装置を用意することにより、流れ作業的に大量のデータを高速に処理する方法である。パイプラインが n 個の部分処理から構成される場合、処理速度は最大 n 倍になる。さらに、ロード・ストアパイプラインと大規模ベクトルレジスタをプロセッサ内部に導入することにより、プロセッサでの処理時間に比べて長いメモリアクセス時間が性能向上の妨げになること防ぐ。科学技術計算に対し優れた処理性能を示すスーパーコンピュータは、それ以降しばらくは、

- パイプラインの動作周波数の向上
- プロセッサ内部のパイプライン演算器の並列化
- 自動ベクトル化などのソフトウェア技術の進歩

により、性能を向上させてきた。

（２）共有メモリ型マルチプロセッサ（Shared-Memory Multiprocessors）

1 つのプロセッサに複数の演算器を導入することにより性能を向上させることが可能であるが、そこには限界がある。そこで、プロセッサ内部での並列処理に加え、複数のベクトルプロセッサによる並列処理でさらに性能を高めたスーパーコンピュータが誕生した。共有メモリ型マルチプロセッサと分類されるこのタイプのスーパーコンピュータは、プログラムの開発を容易にするために、複数のベクトルプロセッサで 1 つのメモリを共有し、1 つの OS（オペレーティングシステム）で動作する（このような処理形態を、SMP: 対称型マルチプロセッシングと呼ぶ)。共有メモリ型マルチプロセッサは、それまでの「1 プロセッサー1 メモリモデル」のプログラムモデルの自然な拡張として、「複数プロセッサー1 メモリモデル」を提供し、これにより逐次型プログラムから並列型プログラムへ移行するための並列プログラムライブラリ整備、ならびに自動並列化などのプログラミング技術の研究・開発が盛んに行われた。これと並行して、ベクトルプロセッサの代わりに比較的安価なマイクロプロセッサを用いた共有メモリ型並列コンピュータも作られ、汎用目的の中規模のサーバシステムとしての利用も始まった。

（３）分散メモリ型パラレルプロセッサ（Distributed Memory Parallel Processors）

単一メモリ空間でのプログラミングモデルを提供する共有メモリ型並列コンピュータは、並列処理による高速化と並列プログラムに対する比較的わかりやすいプログラミング環境を実現したが、プロセッサ数を増やしていくと、共有メモリ上でプロセッサ同士のメモリ競合が起こり、プロセッサ台数に応じた性能向上に限界が生じる。そこで、メモリを共有せずに、プロセッサ、キャッシュメモリ、局所メモリで 1 つの計算ノード

を構成し、複数の計算ノードをネットワークで相互接続する分散メモリ型パラレルプロセッサが開発された。分散メモリ型パラレルプロセッサでは、各計算ノードは局所メモリを利用して割り当てられた部分計算を処理し、他のノードのメモリを使うことはできない。そして他のノードとデータ交換が必要な場合のみに、ノード間通信を明示的に行う。分散メモリ型パラレルプロセッサは、基本的にノード間で共有するハードウェア資源を持たないために、ノードを増加させることにより高性能コンピュータを構成することが可能となる。その一方、プログラマは通信という新しい要素を意識してプログラムすることを余儀なくされる。超並列処理の可能性を求めて、数千から数万のプロセッサからなる分散メモリ型パラレルプロセッサシステムが、新しいタイプのスーパーコンピュータとして商用化された。しかしながら、超並列処理のためのプログラミング技術が未成熟であったことから、システムの持つ潜在的な能力を引き出すことができず、実際にプログラムを実行したときの性能（実効性能）の低さが問題となり、このタイプの初期のスーパーコンピュータシステム、例えば最大 16,000 の SPARC プロセッサから構成される Connection Machine などは市場から消滅した。

（4）分散共有メモリ型パラレルプロセッサ（Distributed Shared Memory Parallel Processors）

共有メモリ型のプログラミングの容易さと分散メモリ型のシステム拡張性（スケーラビリティ）の両方の利点は、分散共有メモリ型パラレルプロセッサとして統合された。分散共有メモリ型パラレルプロセッサは、SMP を行う共有メモリ型マルチプロセッサを 1 つの計算ノードとし、複数の計算ノードをネットワーク経由で相互接続することにより構成される。メモリはノードに分散しておかれるが、分散メモリに対しても SMP を可能とする OS の支援の下でノードに分散するメモリを単一のメモリ空間としてプログラマに提供する。さらに、計算処理を複数のレベルに分割し、頻繁なデータ交換が必要なレベルの処理はノード内の SMP で行い、比較的通信頻度が低いレベルの処理はノード間のメッセージ交換を用いた大規模並列処理、といった通信処理を最適化した階層的な並列プログラミングをも可能としている。

現在、スーパーコンピュータのほとんどは分散メモリ型のシステムであるが、プロセッサの構成の違いから、プロセッサとしてベクトルプロセッサを用いるベクトル並列型スーパーコンピュータ（Vector-Parallel Supercomputer）と、スカラプロセッサを用いるスカラ並列型スーパーコンピュータ（Scalar-Parallel Supercomputer）の 2 つに分類することができる。ベクトル並列型スーパーコンピュータの代表的なものとしては、1 計算ノード（32 ベクトルプロセッサ）の性能が世界的にトップクラスに位置づけられる東北大学情報シナジーセンターSX-7（NEC 製）やシステム総合性能世界第 1 位にランクされている地球シミュレータ（国家プロジェクトとして開発、http://www.es.jamstec.go.jp/esc/jp/）などがある。

一方、スカラ並列型スーパーコンピュータは、Intel Pentium/Itanium や HP/Compaq Alpha など PC やワークステーションにも使われる高性能汎用マイクロプロセッサをベースに設計され、大量のプロセッサによる超並列処理方式により、ベクトル並列型スーパーコンピュータに匹敵する高い性能を実現する。東北大学情報シナジーセンターの並列コン

ピュータ TX7/AzusA（112 Intel Itanium CPU）や米国の国家プロジェクトとして開発された ASCI-Blue/White/Q/Purple（ただし、Purple は 2005 年稼動予定）がこのタイプのスーパーコンピュータである。

スカラ並列型スーパーコンピュータは、その潜在的処理能力がかなり高いものもあるが、メモリから CPU へのデータ供給が CPU 内部にある 100K バイト程度のオンチップキャッシュと数 M バイト程度のオフチップキャッシュに依存していることもあり、大量のデータ処理時の実効性能はベクトル処理に比べて低くなる傾向がある。しかしながら、並列性は高いがベクトル化に不向きな不規則なデータ操作を伴う処理や、小規模のデータに対する参照の局所性の高い処理に対しては、キャッシュが有効に働くことからスカラ並列型が非常に効率良く適用できる。したがって、扱う問題によってベクトル並列型処理を適用するべきかスカラ並列型処理を適用するべきかを適切に判断する必要がある。

東北大学情報シナジーセンターのベクトル並列型スーパーコンピュータ SX-7、およびスカラ並列コンピュータ AzusA の特徴、およびそれぞれの高速化技法については、第 3 章以降で説明する。

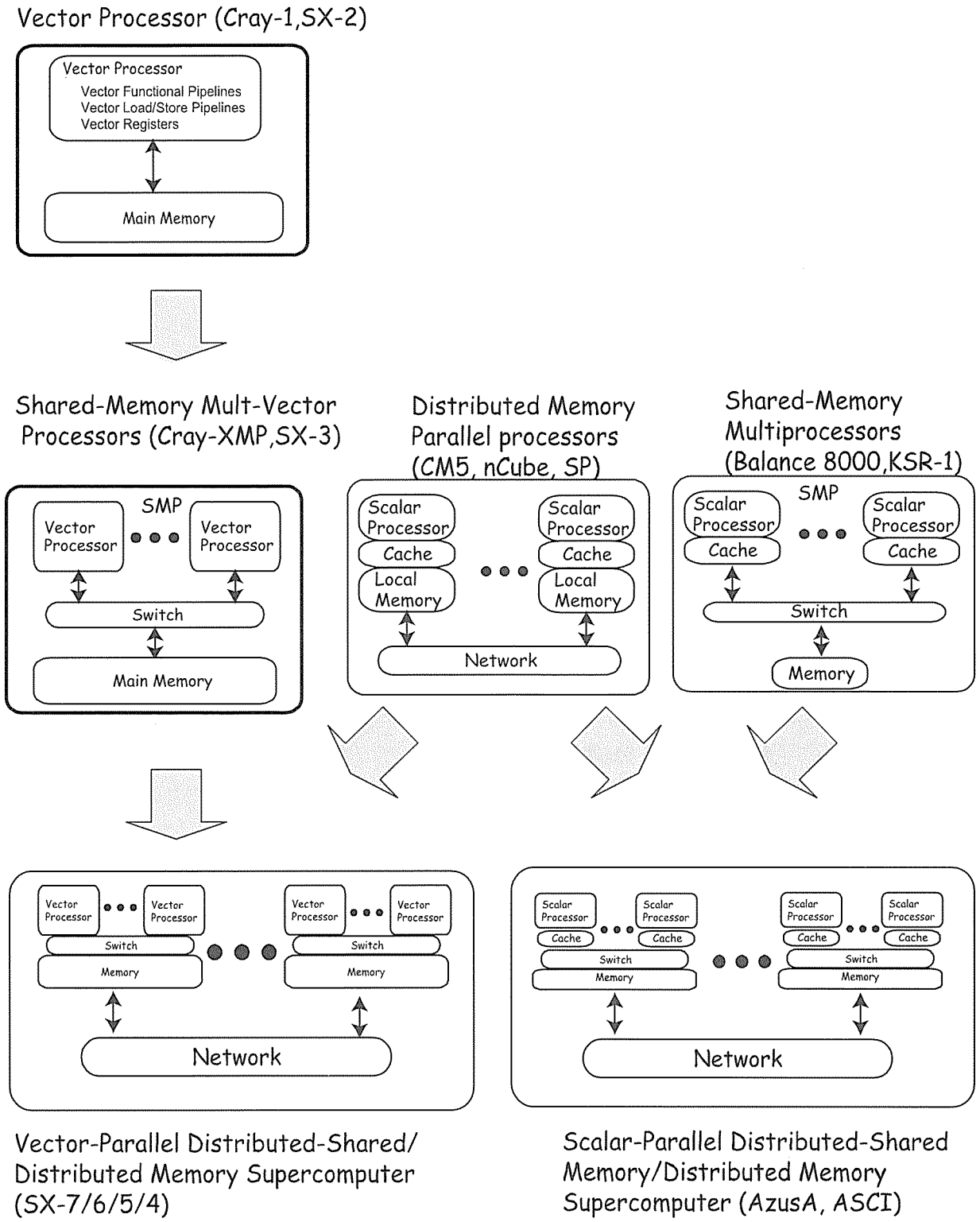
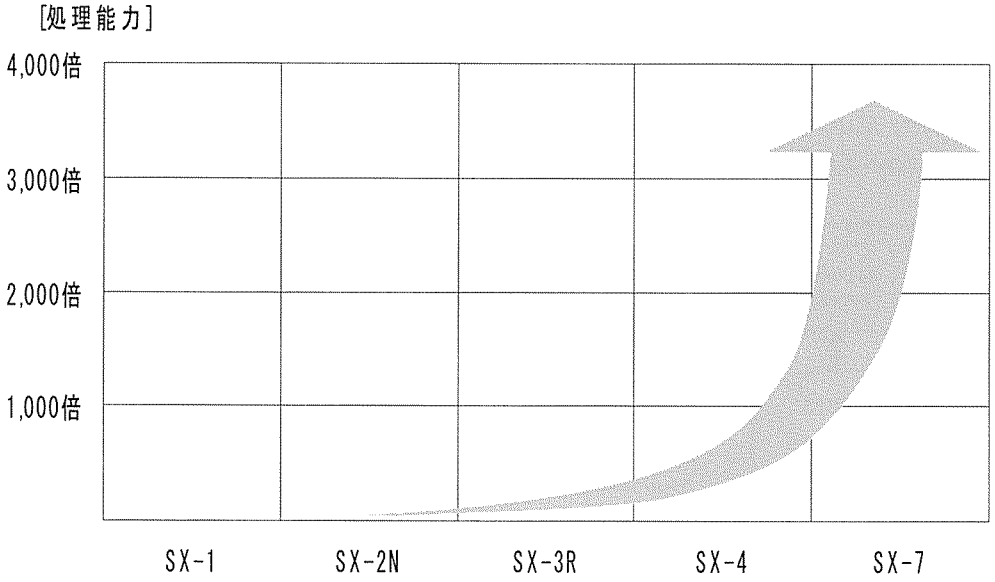


図 2.3　スーパーコンピュータの構成方式の変遷

２．４　東北大学情報シナジーセンターにおける大規模科学計算システムの変遷

東北大学情報シナジーセンターは 1986 年の SX-1（NEC 製、0.57GFLOPS）以来、常に世界最高クラスのスーパーコンピュータを導入し、最先端の学術研究を強力に支援・推進してきた。図 2.4 に示すように、シミュレーションも SX-1 の小規模 1 次元モデルから、SX-4 の小規模の 3 次元モデルへ発展し、2003 年 1 月導入した SX-7（NEC 製、2119GFLOPS）システムでは中規模の 3 次元モデルへと期待できる。



導入年	形式	性能(GFLOPS)	記憶容量	シミュレーションモデル
1986年	SX-1	0.57	256MB	小規模１次元モデル
1990年	SX-2N	1.14	256MB	中規模１次元モデル
1994年	SX-3	25.6	4GB	中・大規模２次元モデル
1998年	SX-4	256	32GB	小規模３次元モデル
2003年	SX-7	2119	1920GB	中規模３次元モデル

図 2.4　本センターのベクトル型スーパーコンピュータの変遷

SX-7 システムは、8.83GFLOPS のベクトル演算プロセッサ（CPU）を 240CPU 備え、8 ノードで構成される。利用者は、自動並列化と、自動ベクトル化を利用することによって、MPI（メッセージ通信ライブラリ）などの知識を有さなくても、1 ノードの演算処理能力（282.56GFLOPS）と 256GB のメモリをフルに利用することができる。さらに、より高度な大規模並列処理を必要とする場合には、MPI を使用した並列プログラミングにも対応可能となっている。

さらに、ベクトル処理に不向きな計算を高速化することを目的として、1998年に並列演算サーバ（Exemplar）、そして2002年に並列コンピュータ（TX7/AzusA）を導入した（表2.1）。

2002年1月から運用を開始した並列コンピュータTX7/AzusAは、7ノード（112プロセッサ）から構成され、358GFLOPSの性能を有する。利用者は、ベクトル化処理に向いていない問題に対して、複数プロセッサによる並列処理とプロセッサ内部でのスーパースカラ処理の組み合わせにより得られる優れた処理性能を利用できる。

表2.1　本センターのスカラー並列コンピュータの変遷

導入年	形式	性能(GFLOPS)	記憶容量
1998年	Exemplar/X	34	12GB
2002年	TX7/AzusA	358	176GB

２．５　大規模科学計算システムの高速化推進研究体制

1998年1月に導入のSX-4システムは本センターでは初のベクトル並列型スーパーコンピュータであり、より規模の大きい3次元シミュレーションなどを実現するためには、自動並列化、自動ベクトル化に頼るだけでなく、プログラムをあらゆる面から検討し、高速化する必要があった。このため、利用者、本センター、日本電気㈱の3者によりシュミレーションプログラムの高速化推進のための研究会を1998年に立ち上げ、高速化技術の研究・開発を推進してきた。この高速化推進研究は2001年3月に報告書第1号を発行し、それ以降も、3者による研究・開発活動を継続してきた。

高速化推進研究のためには、利用者プログラムの計算アルゴリズムとデータ構造、さらには研究目的や研究内容を熟知する必要があり、センター側スタッフと日本電気側スタッフが長時間をかけてそれらを理解・修得し、ベクトル型あるいはスカラ型のコンピュータアーキテクチャを考慮したアルゴリズム、プログラミング方法、データ構造について改善策を利用者に提案してきた。本報告書は、2001年4月から現在に至るまでの高速化推進研究活動をまとめたものである。

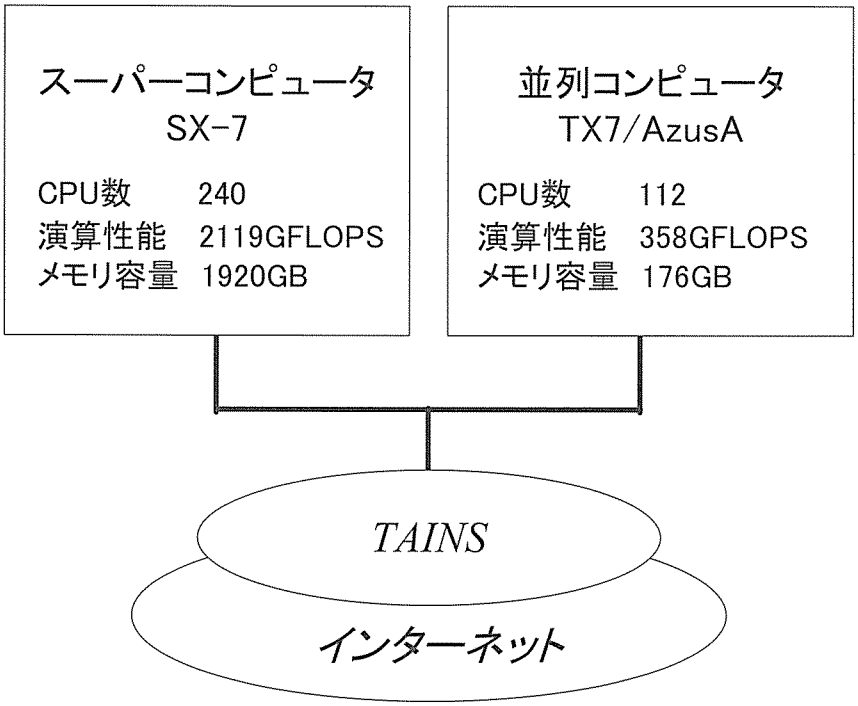
表2.2に本プロジェクトの参加した本センター側のスタッフと日本電気側のスタッフを示す。

表2.2 東北大学と日本電気㈱参加者

東北大学	日本電気㈱
研究部	第一コンピュータソフトウェア事業部
小林　広明　後藤　英昭	橋本　ユキ子、左近　章一
岡部　公起　滝沢　寛之	コンピュータ事業部
システム管理掛	西川　岳、渋谷　俊輝
伊藤　英一　大泉　健治	第一官庁システム開発事業部
システム運用掛	橋本　敏昭、神山　典、熊沢　浩市
小野　敏	金野　浩伸

３　大規模科学計算システムの特長

大規模科学計算システムとしては、スーパーコンピュータ SX-7 システムと、並列コンピュータ TX7/AzusA システムが設置されている。システムは、TAINS（東北大学総合情報ネットワークシステムの略称）に接続しており、センターの利用者は、TAINS を介して学外・学内から、大規模科学計算システムを使用することができる。



３．１　スーパーコンピュータ　ＳＸ－７の特長

SX-7 システムは、１ノードが 32 個の CPU、256GB の主記憶装置で構成され、システム全体のベクトル演算性能が 2119GFLOPS のベクトル型スーパーコンピュータである。ベクトル型スーパーコンピュータは、科学計算分野における基本的なデータ構造の配列、すなわちベクトル型のデータをパイプライン演算器で高速に実行できる。

（１）ハードウエアの特長

SX-7 の主な特長は以下の通りである。

①ベクトル演算性能は、8.83GFLOPS の世界最速の１チップベクトルプロセッサ（CPU）であり、最先端の超高速・高集積の CMOS テクノロジと最先端 LSI 設計技術による CPU である。

②CPU はベクトルユニットと、スカラユニットを備え、ベクトルユニットは５多重８並列のベクトル演算パイプラインによりベクトル演算を高速実行できる。

③CPU のスカラユニットにおける命令処理方式は、4way スーパースカラを採用しており、スカラ最大性能値は 1.1GFLOPS を実現している。
④大規模演算に不可欠なメモリからの高速データ供給能力を超高速クロスバスイッチにより実現し、実プログラムでの高速演算を可能にしている。
⑤32 個のベクトルプロセッサ（CPU）は、共有メモリ方式のアーキテクチャを採用であり、大規模な共有メモリを利用できる。

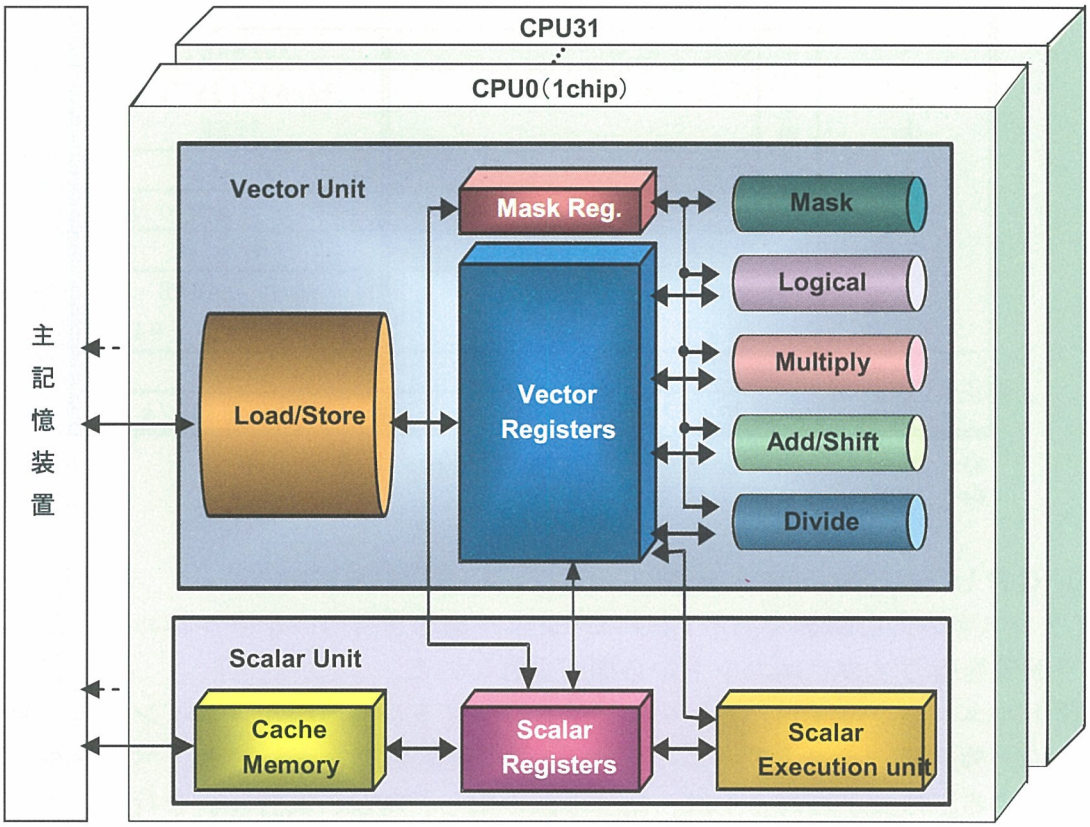


図3．1　CPUの内部構造

表 3.1　ＳＸ－７ノードの主要諸元

項目			SX-7
理論最大演算性能			282.56GFLOPS
CPU数			32
CPU	レジスタ	ベクトルレジスタ	144KB
		ベクトルマスクレジスタ	256bit×16
		スカラレジスタ	64bit×128
	データ形式	固定小数点	32/64bit
		浮動小数点*	32/64/128**bit IEEE
		論理	64bit
	ベクトル演算パイプライン		5種類×8セット
	スカラ演算パイプライン		1セット
	キャッシュメモリ		命令：64KB オペランド：64KB
主記憶装置	容量		256GB
	最大データ転送能力		1130.24GB/s

*)　SX-7では国際標準であるIEEE754規格のみサポート
**)　128bitはスカラ命令のみサポート

（２）ＣＰＵ

SX-7 システムの心臓部である CPU はベクトル演算を行うベクトルユニット、およびスカラ演算を行うスカラユニットから構成されている。

ベクトルユニットは CPU あたり８セットのベクトル演算パイプラインが単一命令ストリーム／複数データストリーム（SIMD）型の並列処理を実行でき、ベクトル演算を高速実行する。スカラユニットは、命令の読み出し、解読、実行制御を行い、スカラ演算パイプラインはベクトル演算パイプラインと同時並行にスカラ演算を実行できる。

ベクトルユニットは、５多重８並列のベクトルパイプラインとベクトルレジスタから構成されている。ベクトルパイプラインは加減算／シフト、乗算、除算、論理演算、マスク演算の５種類を有し、この５種類のベクトル演算パイプラインはデータの依存関係が無ければ、５種の演算が同時に実行できる。さらに各演算用のパイプラインは８個（８並列）が用意され、１つのベクトル演算命令により８個のベクトル演算のパイプラインが SIMD（Single Instruction Multiple Data）型の並列処理を実行できる。図 3.2 にベクトルレジスタに格納されるデータ要素番号と８並列のベクトル演算パイプラインの関係を示している。

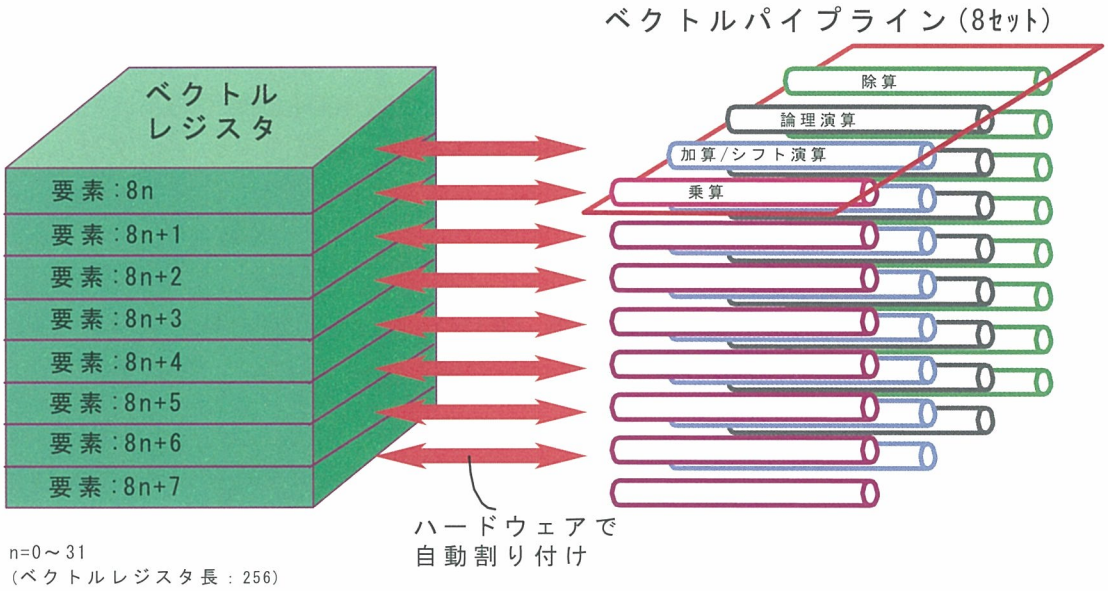


図3.2　ＳＸ－７のベクトルユニットの構成

スカラーユニットは 4way スーパースカラ方式を採用している。最大４個の命令を同時に解読し、ハードウェア資源のぶつかりが無ければ最大４個の命令の処理を同時に起動することができる。スカラ浮動小数点演算は加算・除算の２演算を１クロックで同時に実行しているので、スカラ理論最大性能は 1.1GFLOPS となり、実アプリケーションでの高い実効性能が期待できる。

（３）主記憶装置

SX-7 システムでは高いベクトル演算性能に加えて、主記憶からベクトルパイプラインに対しベクトル演算性能に見合った高いデータ供給能力を有している。

CPU あたりのデータ供給能力は 35.32GB／秒であり、ノードあたりでは、1130.24GB／秒の総合データ供給能力を有している。このデータ供給能力の実現にはデータを格納する主記憶部に独立動作可能な複数のバンクからなる多バンク構成を採用している。

また、主記憶容量はノードあたり 256GB、システムとしては 1920GB を備えている。この主記憶容量の拡大によって、大規模科学計算システムとしての基本的な要件である大規模データ配列を主記憶上へ展開し、プログラムの実行時間を大幅に短縮することができるようになった。

３．２　並列コンピュータ　ＴＸ７／ＡｚｕｓＡの特長

TX7/AzusAは、１ノード16CPUで構成され、理論最大性能は51.2GFLOPSの並列演算サーバである。並列コンピュータは、TX7/AzusAサーバを７台で構成され、CPU数は112個、メモリは176GBを備え、システム全体の理論最大性能は358.4GFLOPSである。ベクトル化が難しくベクトル化率の低いプログラム、あるいはスカラー演算向きのプログラムのシステムを提供している。

（１）ハードウエア構成

TX7/AzusAは、プロセッサ（CPU）／メモリ／IOコントロラを持つセルカード４枚を中核に、セルカード間を接続するデータクロスバなどから構成される。（図3.3）

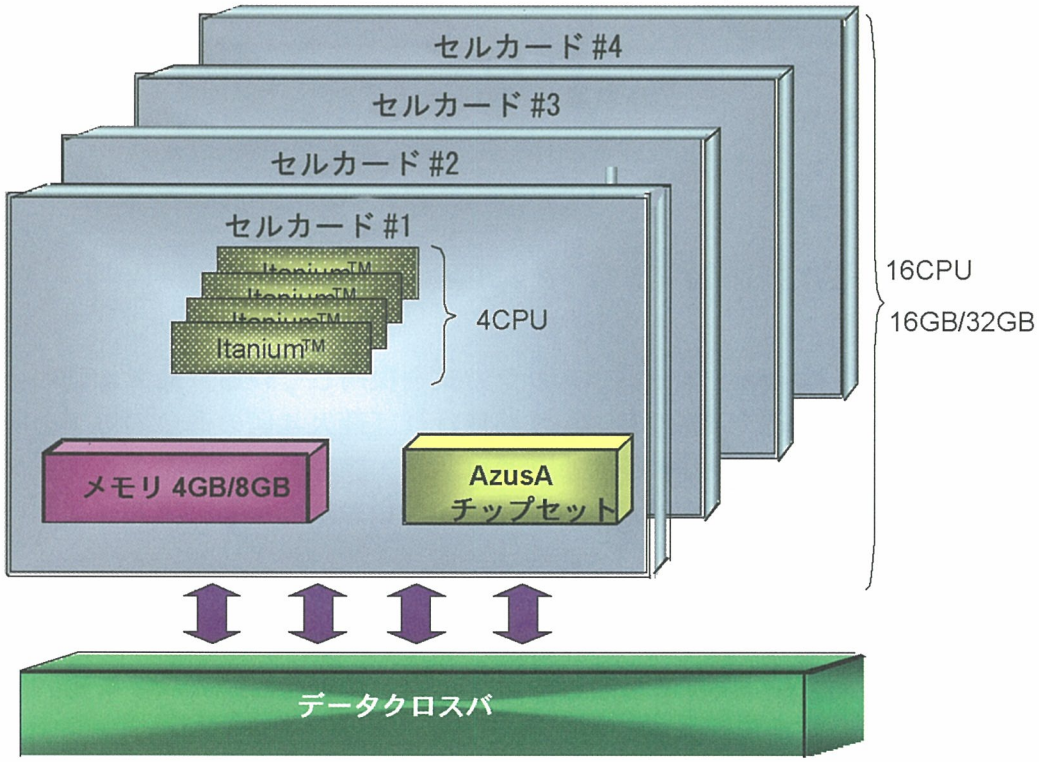


図3.3　ＴＸ７／ＡｚｕｓＡシステム構成

各セルカードには４つのプロセッサ（CPU）とメモリ（4GB／8GB）を搭載し、４セルカードにより、16CPU、16GB/32GBのシステムになる。４枚のセルカードに分散されたcc-NUMA（Cache Coherent Non-Uniform Memory Access）型の構造をしているが、ソフトウェアから見たメモリアクセス時の振る舞いは16プロセッサのSMPシステムに近いものとなっている。

（２）プロセッサ

AzusAの搭載しているプロセッサは、Itanium™（IA-64）であり、そのプロセッサ仕様を表3.3に示す。

プロセッサは命令の並列処理を効率化することで高性能を達成しており、その特徴とし

ては「明示的に並列性を記述した命令のサポート」、「命令レベルの並列性をさらに強化する機能のサポート」の２つをあげることができる。

明示的な並列性を記述するために、Itanium™アーキテクチャでは３つの命令と３つの命令の組み合わせを表現するテンプレートからなる「命令バンドル」という概念が採用されている。プロセッサは命令をバンドル単位で実行していきますが、プロセッサは１サイクル当たり２バンドルを同時実行可能であるため、バンドル当たり１つの倍精度浮動小数点演算命令を含むことが可能である。加えて、積和演算命令をサポートしているため１サイクルあたり４つの倍精度浮動小数点演算が可能であり、TX7/AzusA、800MHzの場合、理論最大性能は、3.2GFLOPS（倍精度）である。

さらに、命令レベルの並列性をさらに強化するために、メモリレイテンシの影響を最小限に抑えるための機能である「スペキュレーション」、命令の分岐を排除するための機能である「プレディケーション」、さらには分岐予測による分岐コストの削減といった先進的な機能を豊富にサポートしている。

表3.3　Itanium™の仕様

項目	仕様
汎用レジスタ数	整数 128、浮動小数点 128
パイプライン段数	10 段
実行ユニット	ALU　×4 分岐ユニット　×3 単精度浮動小数点ユニット　×2 倍精度浮動小数点ユニット　×2
アドレス空間	仮想メモリ空間　16PB(54bit) 実メモリ空間　16TB(44bit)

（３）プロセッサのキャッシュ構成

一般的なスカラ型マイクロプロセッサと同様に Itanium™（IA-64）もキャッシュメモリの有効活用が性能向上のかぎとなる。そこで表 3.4 および図 3.4 にキャッシュメモリの仕様と構造を示す。

表 3.4　Itanium™プロセッサのキャッシュ仕様

	Level1（命令）	Level1（データ）	Level2（命令）	Level3（命令）
容量	16KB	16KB	96KB	4MB
実装場所	On die	On die	On die	パッケージ内
ラインサイズ	32Byte	32Byte	64Byte	64Byte
アソシビティ	4way	4way	6way	2way

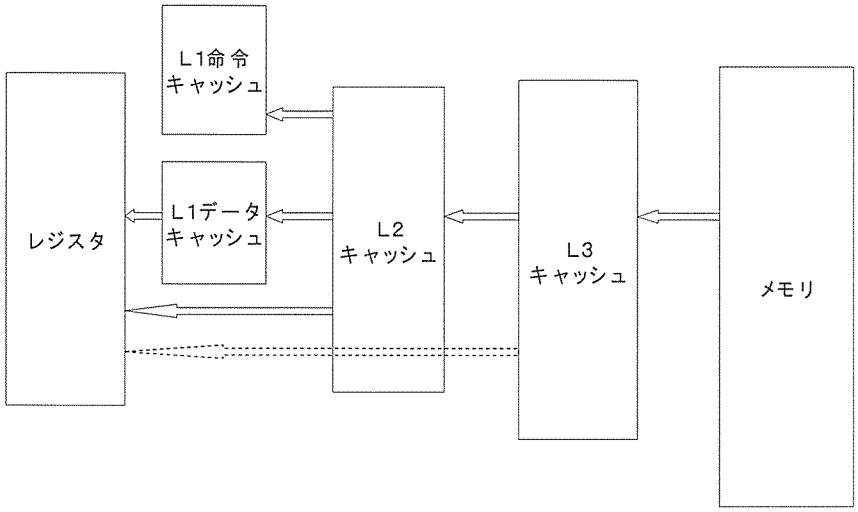


図 3.4　キャッシュメモリの階層

TX7/AzusA は、図 3.4 のようにプロセッサとメモリとの間に L1、L2、L3 の 3 階層のキャッシュを持っているが、浮動小数データについては、L1 キャッシュを用いないため、L2 キャッシュが最も CPU に近い高速なキャッシュである。L2 キャッシュのアソシビティは 6way であることから、16KB のキャッシュを 6 個持っていることになる、すなわち、異なる 6 個までの配列データ（倍精度浮動小数ならば 2048 要素×6 個）は、キャッシュを有効に活用することができる。

プロセッサからキャッシュとメモリへのアクセスによる演算性能は、データが L2 キャッシュ、L3 キャッシュ、あるいはメモリのどこにあるかによって異なり、データの所在による演算性能比を表 3.5 に示す。

TX7/AzusA の 1 CPU の倍精度浮動小数理論演算性能は、最大 3.2GFLOPS なので、倍精度浮動小数の計算が連続的に行なわれる場合、L2 キャッシュにデータがあれば、1 つのデータに対して 1 演算で行なうので、データ供給のバンド幅と演算の速度がつり合っている。

表 3.5　　キャッシュとメモリの特性

	サイズ	バンド幅	バンド幅：倍精度演算性能
L1 データ	16KB	12.8GB/s	—
L2	96KB	25.6GB/s	1：1
L3	4MB	12.8GB/s	1：2
メモリ		2.1GB/s	1 ：12

しかし、１つの倍精度要素に対して、データがL3キャッシュにある場合は２演算、メモリにある場合は12演算を必要としており、データ供給スピードと演算性能がつり合わない。逆に言うと、L3キャッシュやメモリにデータがある場合、１データに対して上記の数程度の演算を行えない場合、データ供給ネックとなり、最大演算性能を引き出すことができない。

（４）セルカード間のメモリアクセス

TX7/AzusAは、あるプロセッサからのメモリアクセス時、自セルカード内のメモリからのデータ読み出しに対して、他セルカードのキャッシュメモリまたはメモリからの読み出しの場合はカード間のパスを双方向に備えるバンド幅が処理のボトルネックとならないように工夫して、アクセス時間の低下を小さくしている。

４　スーパーコンピュータ　ＳＸ－７の高速化技法

４．１　高速化の必要性

SX-7 は、ベクトル型スーパーコンピュータ SX シリーズの最新機種で、32CPU の共有メモリ方式を採用し、SX-4 で開発されたソフトウェア資産の移行性を容易にしている。

SX-7 システムは、理論最大演算性能が 8.83GFLOPS の単体性能 CPU を 240CPU 搭載し、システム全体で 2TFLOPS 超（2119.2GFLOPS）の総合性能を備えている。ノードは 282.6GFLOPS の演算性能、256GB のメモリを搭載し、性能を増強、規模を拡大している。

この超高速演算性能を実現しているハードウェアの特長としては以下のものがある。

①32 個のベクトルプロセッサ（CPU）が同時動作できる強力なベクトル演算性能
②共有メモリ方式のアーキテクチャの 32CPU と 256GB メモリの接続
③共有メモリからの高速データ供給能力は超高速クロスバスイッチにより実現

このような SX-7 のハードウェア性能を十分に引き出すためには、

・個々の CPU の中で効率的な計算を行う　＞　ベクトル化
・複数の CPU を効率的に使用する　＞　並列化

が大変重要になる。

SX システムの Fortran90 コンパイラである FORTRAN90/SX は、SX-7 のハードウェア性能を十分に引き出すための、高度な自動ベクトル化機能、自動並列化機能を備えている。

しかし、こうしたコンパイラによるベクトル化、並列化の能力には限度があるため、常に最適な実行プログラムが生成できるわけではない。SX-7 のハードウェアの特性の発揮できるようにプログラミングにおける工夫を行うことによって、性能を大きく改善できる。

４．２　ベクトル化による高速化

SX-7 のベクトル命令は、複数の組のデータに対する演算処理を一つの命令で一度に行うことができる。

ループ中で計算される行列の要素など、規則的にならべた配列データ（ベクトルデータ）に対してベクトル命令を適用することを自動ベクトル化と呼び、ベクトル化することによってベクトル演算機構をもちいた高速な演算が可能となる。これに対して、通常の演算命令はスカラ命令により一度に一組のデータに対する演算処理を行っているなども違いがある。

ベクトル化によるプログラムの実行性能の向上を行うには、

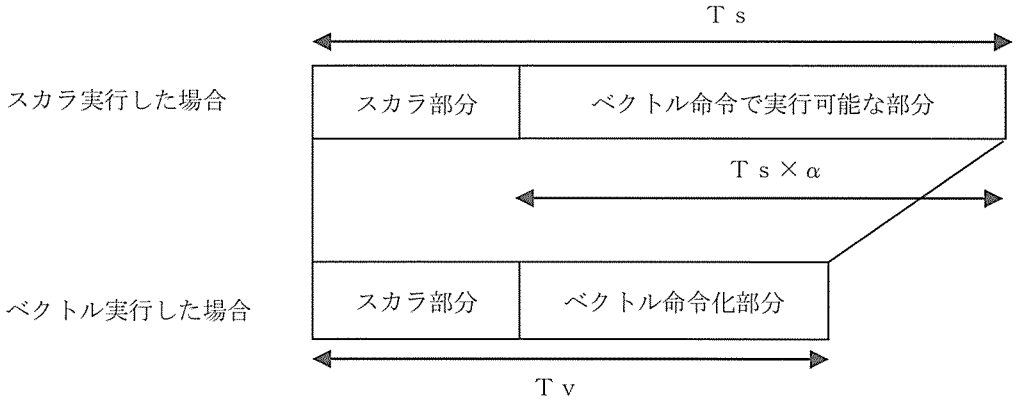
・ベクトル率の向上、

・ベクトル長の向上、
・メモリアクセスの改善

これら、３点に着目したプログラムの工夫を行うことで、実行性能の向上を図ることが重要である。

４．２．１　ベクトル化率

ベクトル化率とは、プログラムをスカラ命令だけで実行させた場合の実行時間（Ts）に占める、ベクトル命令で実行可能な部分（Ts×α）の時間の割合である。



α　：　ベクトル化率
Ｔｓ　：　スカラ実行したときの実行時間
Ｔｖ　：　ベクトル実行したときの実行時間

図 4.1　ベクトル化率と加速率

プログラムをベクトル化することにより実行性能は向上しますが、プログラムの一部だけをベクトル化しても、ベクトル化の効果はあまり期待できません。

図 4.1 からもわかる通り、ベクトル化率を上げることは、ベクトル実行可能部分をいかに増やすかが大きな鍵となる。

ベクトル化率と、プログラム全体の実行時間の理想的な性能向上の関係を表すのに、アムダールの法則がある。ここでは、ベクトル実行したときの加速率を 50 倍と仮定した時の、ベクトル化率による性能倍率を図 4.2 に示す。

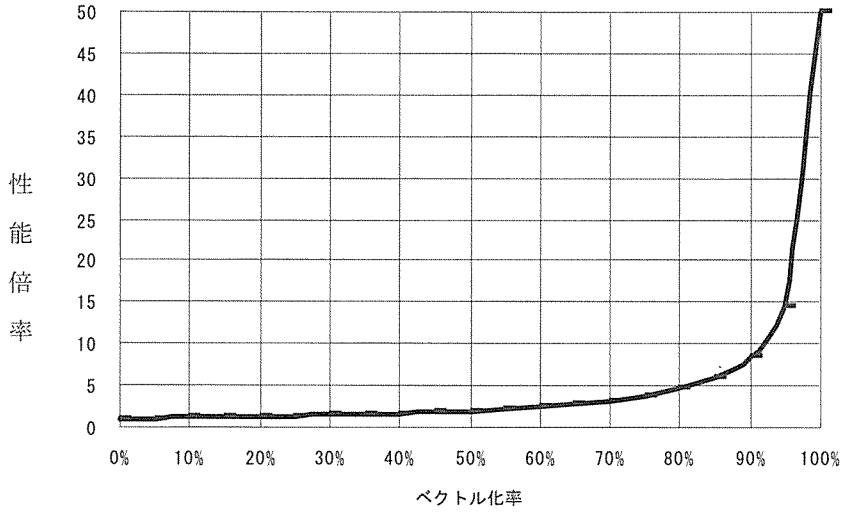


図4.2　ベクトル化率におけるアムダールの法則

このグラフを見るとわかるとおり、たとえば、ベクトル化部分の実行時間が非常に小さくなったとしても、ベクトル化率が 50%程度では、高々２倍の性能にしかならないことはわかると思います。一般にはベクトル化率が 95%以上ないと、ベクトル化による大きな性能向上効果は期待できない。

すなわち、ベクトル化による高速化技法の一つは、ベクトル化率を高め、できる限り 100%に近づけることである。このベクトル化率は正確に求めることは困難であるため、SX-7 では、ベクトル化率に近い値として、次のように表示されるプログラム情報（Program Information）のベクトル演算率を用いる。

```
***** Program Information ******
    V. Op. Ratio (%)      :              99.643774      <  ベクトル演算率
```

ベクトル演算率は、実行された命令の数をハードウェアがカウントすることで、プログラムで処理された全演算数に占める、ベクトル演算命令で処理された数の割合を求めたものである。

SX-7 では、このベクトル演算率を高くすることを目標に、高速化のチューニングを行う。

なお、プログラム情報は、proginf 機能を使用すると、プログラムの実行が終了したときに次のような内容で出力され、プログラム全体の分析に用いることができる。この情報は、実行時オプションの指定によりますが、東北大学情報シナジーセンターでは、”出力”が既定値となっている。

```
***** Program Information ******
Real Time (sec)        :      1570.275969
User Time (sec)        :      6217.915341
Sys Time (sec)         :        19.211721
Vector Time (sec)      :      5822.816669
Inst. Count            :     82652532167.
V. Inst. Count         :     44241631526.
V. Element Count       :  10744313583322.
FLOP Count             :   4702924898916.
MOPS                   :      1734.138195
MFLOPS                 :       756.350745
VLEN                   :       242.855275    <  平均ベクトル長
V. Op. Ratio (%)       :        99.643774    <  ベクトル演算率
Memory Size (MB)       :      1664.031250
MIPS                   :        13.292644
I-Cache (sec)          :         2.249134
O-Cache (sec)          :        60.878483
Bank (sec)             :        43.971155    <  バンクコンフリクト時間
```

図 4.3　Program Information（Proginf 情報）の例

SX システムの FORTRAN90/SX では、プログラム中の DO ループ、IF、GOTO ループ、配列式をベクトル化の対象にしており、これらがベクトル化できるためには、以下の条件を満たしている必要がある。

（ａ）ループまたは配列式内にベクトル化可能な文、型、演算がある

（ｂ）ループ中の変数や配列が定義・引用される順序がベクトル化によって変化しない

この条件を満たさずベクトル化できないものとは何か、CALL 文、入出力文などの文、文字型や４倍精度実数型、構造型などの型、文字比較、文字代入、ポインタ代入などが該当し、(a) の条件を満たさないのでベクトル化されません。

また、次のように、前の繰り返し回で定義された値を後の繰り返し回で参照するようなループは、(b) の条件を満たさないため、ベクトル化されない。

```
do   i=3,n
     x(i)=x(i-2)*2.0+y(i)
end  do
```

図 4.4　ベクトル化できないループ

また、プログラムを高速化するためには、FORTRAN90/SX の自動ベクトル化機能に加えて、指示行の挿入や、プログラムの書き換えにより、ベクトル化できない部分を減らし、ベクトル化率をできる限り 100%に近くづけていくことも重要である。

ここでは、ベクトル化率（ベクトル演算率）を向上させる手法の一つとして、ベクトル化指示行と、依存関係を無くすことについて、簡単に使用例を示す。

（１）指示行の挿入によるベクトル化率の向上

配列や変数の定義・引用関係がコンパイル時に不明で、ベクトル化しても結果が変わらないかどうかをコンパイラが判断できないケースでは、ベクトル化しても問題がなければ、ベクトル化指示行 NODEP を挿入することでベクトル化を促進することができる。

次の例では、配列 ix(i)の複数の要素が同じ値を持っているとベクトル化できません。しかし、同じ値をもつことがないとわかっているならば、ベクトル化指示行 NODEP を指定することでベクトル化できる。

```
!cdir  nodep
       do  i=1,n
         a(ix(i))=a(ix(i)+b(i))
       enddo
```

図 4.5　指示行によるベクトル化

（２）ソースプログラムの書き換えによるベクトル化率の向上

配列や変数の定義・引用関係に依存が無いようにプログラムを書き換えることで、ベクトル化ができるようになる場合がある。次の例では、変数 x の定義・引用関係に依存があるためベクトル化できませんが、右のループのように書き換えることにより、ベクトル化が可能となる。

```
do  i=1,n                          a(1)=x
  a(i)=x          ------>          do  i=2,n
  x=b(i)+c(i)                        x=b(i-1)+c(i-1)
enddo                                a(i)=x
                                   enddo
                                   x=b(n)+c(n)
```

図 4.6　ループの書き換えによるベクトル化

４．２．２　ベクトル長

個々のベクトル命令は、演算処理に入る前にある程度の準備処理が必要となります。この準備処理に要する時間を立ち上がり時間と呼びます。即ち、実際のベクトル演算に要する時間が余りにも小さいと、立ち上がり時間の影響が大きくなり、ベクトル化による高速化が行えない。

ベクトル化した場合とベクトル化しない場合とで実行時間が等しくなるループの繰り返し数(ループ長)を交差ループ長と呼び、立ち上がり時間と交差ループ長には、図 4.7 に示す関係がある。

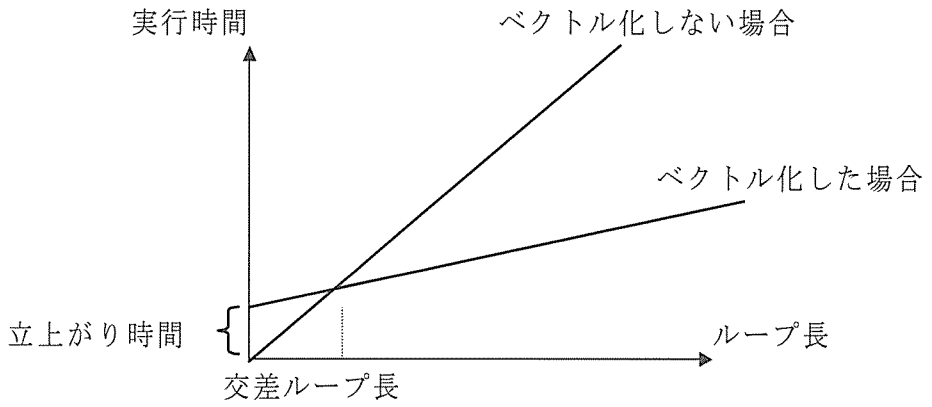


図4.7　立ち上り時間と交差ループ長

この図を見ると、ループ長をできるだけ長くすることがベクトル化の効率が高まり、高速化の効果が大きいことがわかる。

FORTRAN90/SX コンパイラの自動ベクトル化では、ループの繰り返し回数が５未満の場合には、ベクトル化による効果が少ないと判断し、ベクトル化はしない。

このループ長について、ベクトル演算を効率的に実行する要素であるが、ベクトル命令では、一度に最大 256 個の要素を演算でき、１個のベクトル命令で処理する要素数はループ長で決まるため、特にループ長が 100 以下の場合には、ループ長を長くする必要がある。多重ループの場合、コンパイラは基本的には最内側ループをベクトル化するので、できるだけ最内側ループのループ長が長くなるようにする工夫が必要になる。

なお、プログラム全体についての平均ベクトル長は、Proginf 情報（図 4.3 参照）に次のように表示されるので、VLEN の値によってベクトル長が充分かをチェックすることができる。

```
***** Program Information ******
      VLEN              :          242.855275          <  平均ベクトル長
```

コンパイル時にループ長がわかる場合には、コンパイラが自動的にループの入れ換えを行うが、ループ長が実行時にならないとわからない場合は、入れ換えを行わない。例えば以下の多重ループで、m が 10、n が 1000 だとわかっているならば、右のように書いたほうが、コンパイラの自動入れ換えによってベクトル演算の効率は高くなる。

```
do   i=1,n                                  do   j=1,m
  do   j=1,m                                  do   i=1,n
    a(i,j)=b(i,j)+c(i)*d(j)                     a(i,j)=b(j,i)+c(i)*d(j)
  enddo                          ---->      enddo
enddo                                       enddo
```

図 4.8　ループの入れ換えによるループ長の拡大

４．２．３　メモリアクセスの改善

SX-7 は、主記憶とベクトルレジスタとの間のデータの転送を高速に行うために、主記憶を 32 個の独立したモジュール（これをメモリバンク・グループと呼ぶ）に分割して、別々のグループに対して、並列に読み出しまたは書き込みができるようにしている。しかし、同一のメモリバンク・グループに対しては、同時には１つの読み出しまたは書き込みしかできないため、連続して同じグループのデータをアクセスすると性能が低下する。この状態を、proginf 情報（図 4.3 参照）では、"Bank(sec)" バンクコンフリクト時間として、次のように表示される。

```
***** Program Information*****
      Bank (sec)                  43.971155      <   バンクコンフリクト時間
```

例えば、一つの配列を 32 要素飛び（又は 32 の倍数要素飛び）にアクセスすると、バンクコンフリクトが発生する。これを防ぐためには、配列宣言を奇数になるように変更するか、ループを入れ換えて、配列のアクセスが連続するように変更する。

次の配列 a は２次元目でベクトル化されるが、１次元目の配列宣言の大きさが 32 の倍数となっているため、同じメモリバンク・グループのデータを連続してアクセスすることになり性能が低下する。この場合、配列の宣言が奇数になるように変更すると、バンクコンフリクトは発生しなくなる。

```
real,dimension(1024,1000)  a        real,dimension(1025,1000)  a
do   i=1,n                  ------->    do   i=1,n
  a(k,i)=0.0                                a(k,i)=0.0
enddo                                     enddo
```

図 4.9　メモリアクセスの競合させないループ

４．３　並列処理における高速化

並列処理は、一つの仕事をいくつかの小さな仕事に分割し、それを複数のタスク（CPU）で並列に実行することにより、仕事の終了までの経過時間を短縮することである。

ここで注意しなければならないことは、並列処理では、CPU 時間が削減されるわけではなく、逆に、仕事を各タスクで並列実行させるための処理（オーバーヘッド）が必要になり、CPU 時間は増加することになる。

CPU 時間と経過時間の関係は、以下のようになる。

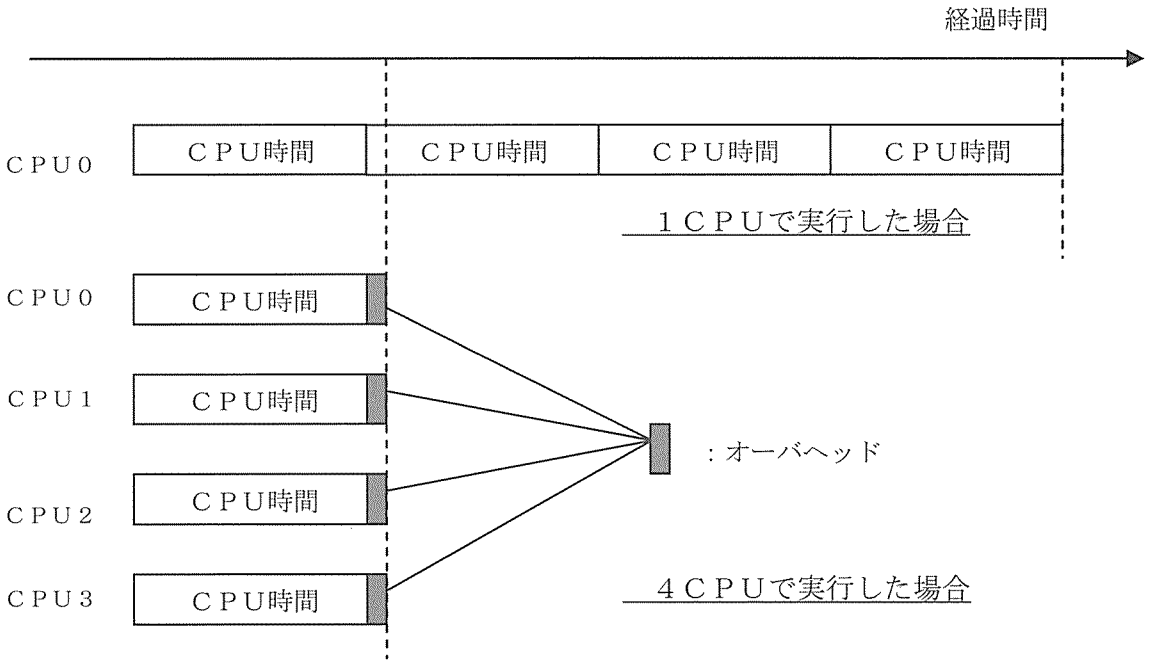


図4.10　並列化におけるCPU時間と経過時間

このように、並列化した場合並列処理のオーバーヘッド時間があるので、並列に実行される仕事量（粒度と呼ぶ）が十分に大きくなければ、並列化の効果は期待できない。当然ですが、並列処理のオーバーヘッド時間よりも並列実行される部分の実行時間が小さければ、並列化したことにより、実行時間（経過時間）が逆に多くなってしまうことになる。これは、高速なベクトル演算を使用することによりCPU時間が短縮され、同時に経過時間も短縮されるため、ほとんど全ての場合に性能向上が計れるベクトル化と大きく異なるところである。

並列化により性能向上させるためには、下記に述べるように並列化率の向上と並列化効率の向上が重要となる。

４．３．１　並列化率

並列化率とは、あるプログラムを１CPUで実行した場合の実行時間に対する、並列実行可能な部分の実行時間の割合を示したものである。

並列化率（α）は、シングル実行を並列実行にした場合で、次のように表すことができる。

Ｔｓ：　シングル実行したときの実行時間
Ｔｐ：　並列実行したときの実行時間
α　：　並列化率
ｎ　：　並列実行したCPU台数

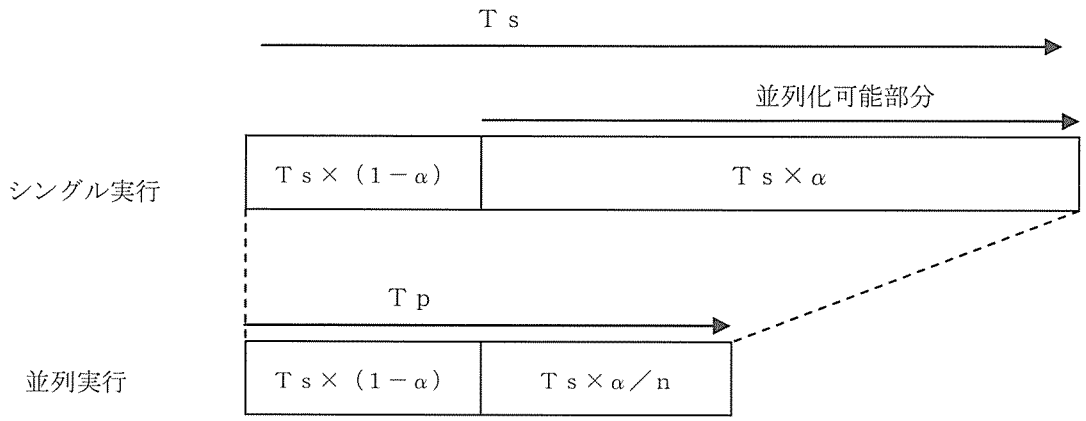


図4.11　並列化率

あるジョブを、シングル実行したときの実行時間、Ts に対して、並列化率（α）ならば、CPU 台数（ｎ）で並列実行したとき、実行時間は Tp になる。

図 4.11 からもわかる通り、並列化率を上げることは、並列化可能部分をいかに増やすかが大きな鍵となる。そのためには、プログラムを書き変えたり、指示行を挿入したりして、並列化を促進していくことが必要である。

並列化率（α）と実行時間の理想的な CPU 台数効果（性能倍率）を表したアムダールの法則は、図 4.12 に示す。

CPU 台数分の並列化効果を上げるためには、95%近い並列化率に上げなければならないことがわかる。

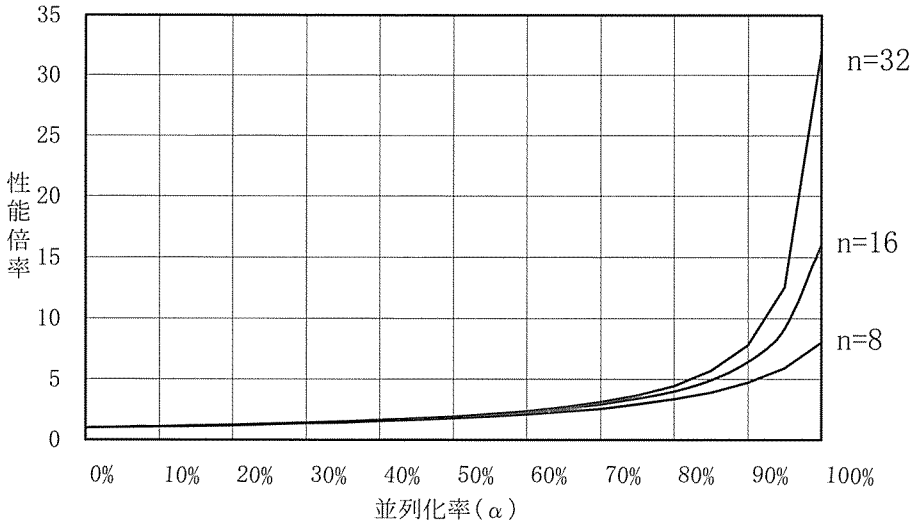


図 4.12　並列化におけるアムダールの法則

FORTRAN90/SX の自動並列化機能は、プログラムを解析し、十分性能が向上するだけの粒度をもっていれば、並列化できるようにループやデータの定義を書き直し

て１ノード内の 32CPU を用いた並列プログラムを生成している。この場合、コンパイラは、例えば do ループの繰り返しを並列実行しても結果不正になるような文を含んでいないかなどを調査し、結果不正にならないチェックを行っている。このような、ループが並列化できるためには、以下の条件を満たしている必要がある。

（ａ）ループ内に call 文や入出力文などの並列化できない文がない

（ｂ）ループ中の変数や配列がループの繰り返しにまたがって定義・引用されていない

この条件に沿って、コンパイラは自動並列化を行うが、並列化効率を更に上げるためには、コンパイラが自動並列化できないループに対して、プログラムの書き換えを行うこと、あるいは並列化指示行を挿入したりして、並列化を促進してゆくことが必要である。

４．３．２　並列化効率

並列化効率とは、同じように並列化した場合でも、効果的な並列化手法を採用できているか、より性能の良い並列化が行われているかということを表している。

この並列化効率を左右する大きな要因としては次のようなものが考えられる。

・並列化されているループの（ベクトル化後の）実行時間（粒度）は十分に大きいか

・並列処理のオーバヘッドは最適か（大きすぎないか）

・並列実行されるループの仕事量（負荷バランス）は各タスクに対して均一か

前に説明した通り、並列処理されるループは、十分な仕事量をもっていなければ、並列化処理のオーバヘッドの割合が大きくなり、その効果を上げることはできない。したがって、同じ多重ループを並列化する場合でも、より演算量が多くなる繰り返しに着目して並列化ができるように注意を払うことが必要である。これらについては、指示行による指定や、ループの入れ換えなどを工夫することによって可能となる。

SX-7 では、内側ループはベクトル化により高速化を行い、なるべく外側のループを並列化により高速化するようにループの入れ換えを行うなどの工夫が必要になる。

一方、いくら並列化率が高く、並列の粒度が大きく、並列オーバヘッドの割合が低くても、個々のタスクの負荷バランスが悪いと、経過時間は最も負荷の高いタスクによって決まってしまうため、各タスクの仕事量（処理量）を均等にする必要がある。

FORTARN90/SX では、ループの並列実行方法は以下の２つの方法がある。

for　：　ループを指定した数に分割する

by　：ループを指定した数の繰り返しをもつループに分割する。

並列化されるループの特性に合わせ、これらを使い分けることにより、ループの処理が、できるだけ均等に各タスク上で処理されるように分割方法を工夫することが必要になる。

５　並列コンピュータ　ＡｚｕｓＡの高速化技法

５．１　高速化の必要性

TX7/AzusA（本章ではAzusAと略す）は、Itanium™（IA-64）プロセッサを16個搭載した並列コンピュータシステムである。EPIC（Explicitly Parallel Instruction Computing）アーキテクチャを採用し、１クロックに２つの浮動小数点命令を実行可能で、また積和命令（乗算と加減算を１命令で行う）をサポートしているので、１クロックに最大４つの浮動小数点演算を実行することができる。この他、科学技術計算向きには、ソフトウェアパイプライニング支援機能、多数のレジスタ、レジスタローテーション、ループ専用ブランチ命令、３レベルのキャッシュなどにより高速化を図っている。このため、AzusAの演算性能を最大限に引き出には、ハードウェアの特長を生かしたプログラムミングの工夫が必要であり、特にキャッシュの有効利用は重要である。

例えば図5.1は、積和演算a(i)=a(i)*x+yの性能をループ長を変えながら測定したものであるが、このループの外側にダミーのループを入れて計測し、キャッシュの効果を見ている。ループ長12,000まではデータがL2キャッシュ（96KB）に収まっているので高い性能がでている。ループ長12,000を境に性能は急速に低下し始めているが、これはデータがL2キャッシュを外れ、L3キャッシュのアクセスが発生していることによる。さらにループ長を長くするとデータはL3キャッシュを外れ、メモリへのアクセスにより性能が低下している。

```
do   k = 1,kk   !ダミーのループ
  do   i = 1,n      ! 計測ループ．n=ループ長
    a(i) = a(i) * x + y
  enddo
enddo
```

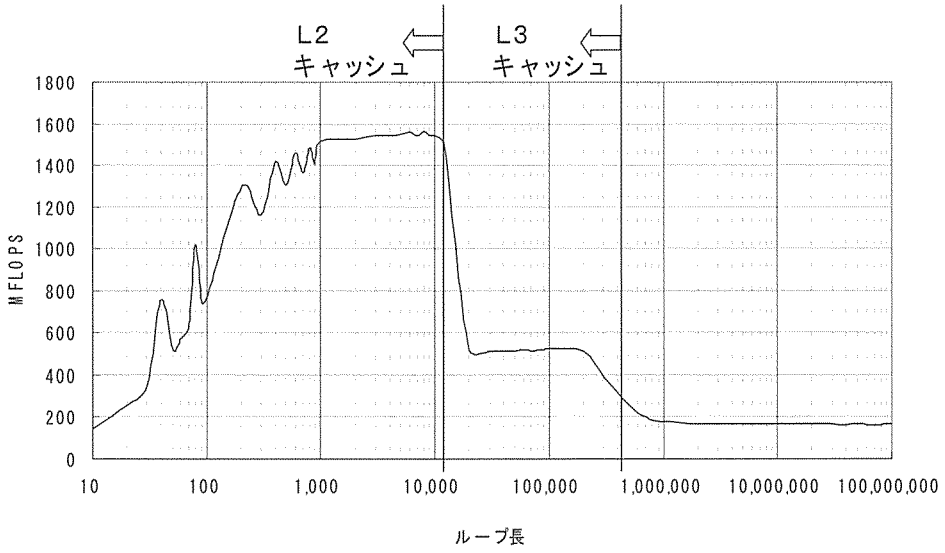


図 5.1　積和演算の性能

このように、データがL2キャッシュに乗っている間は高い性能が出るが、L2キャッシュを外れてL3キャッシュの領域に入ると急に性能が低下する。さらにL3キャッシュを外れると著しく性能が低下している。したがって、高速化のためには、できるだけメモリにデータを取りに行く頻度を少なくすること、CPUのレジスタやキャッシュ上にデータを保持しそれを再利用するようにすることが高い処理性能を実現する上で重要である。

また、AzusAはプロセッサとメモリの構成として、４枚のセルカードによるcc-NUMA型の構造を採っている。一般に、cc-NUMAアーキテクチャでは、同一セル内のアクセスに比べ、異なるセルのメモリへのアクセスは性能が低下する。このため、並列処理の場合は動作するCPUとメモリを同一セルに保つことが高速化に有効となる。

AzusAにおける、プログラムの高速化のポイントは、以下の通りである。

①キャッシュの有効利用
②メモリアクセスの削減
③ソフトウェアパイプライニング（ＳＷＰ）の促進
④ファーストタッチ　－並列処理性能の向上－

AzusAシステムのFortran95コンパイラは、ハードウェア性能を十分に引き出すための、ハードウェアに密着した最適化、ループ最適化、ソフトウェアパイプライニング、キャッシュ利用の最適化などの高度な最適化機能、及び自動並列化機能を備えている。

しかし、こうしたコンパイラによる最適化、自動並列化の能力には限度があるため、常に最適な実行プログラムが生成できるわけではない。最大限の性能を求める場合には、ハードウェアの特性を発揮できるようなプログラミングにおける工夫を行うことが有効である。

５．２　キャッシュの有効利用

AzusAは、並列処理機能を利用することによって、プログラムを高速に実行することができるが、性能を最大限に引き出すには、単一CPUでの性能をできる限り向上させておくことが重要である。そこで、以下では単一CPUでの性能向上のためのキャッシュの効率的な利用法について述べる。

(1) 配列の連続アクセス

AzusAでは、データをメモリからキャッシュにロードするとき、連続する64バイト（倍精度実数型データで８要素分）で行う。このため、Fortranでは、配列を１次元目で連続的に使用する（１次元目を増加／減少させる）とキャッシュの有効利用が図れる。つぎのプログラムは、２次元配列aの全要素の総和をとるプログラムである。

```
s = 0.0d0
do   j = 1,n
  do   i = 1,n
       s =   s + a(i,j)
  enddo
enddo
```

まず、j=1,i=1 のとき、a(1,1)の加算を行うために、a(1,1)がキャッシュにロードされ、同時に a(2,1)～a(8,1)もキャッシュにロードされる。つぎに、i=2 のときは、a(2,1)がすでにキャッシュにロードされているので、a(2,1)の加算が直ぐに行える。以下、i=8 まではキャッシュのデータを利用して配列 a の加算が行われる。すなわち、メモリからキャッシュへのロードは、８要素につき１回ですみ、キャッシュの有効利用が行われている。この様子を示したものが、図 5.2 である。

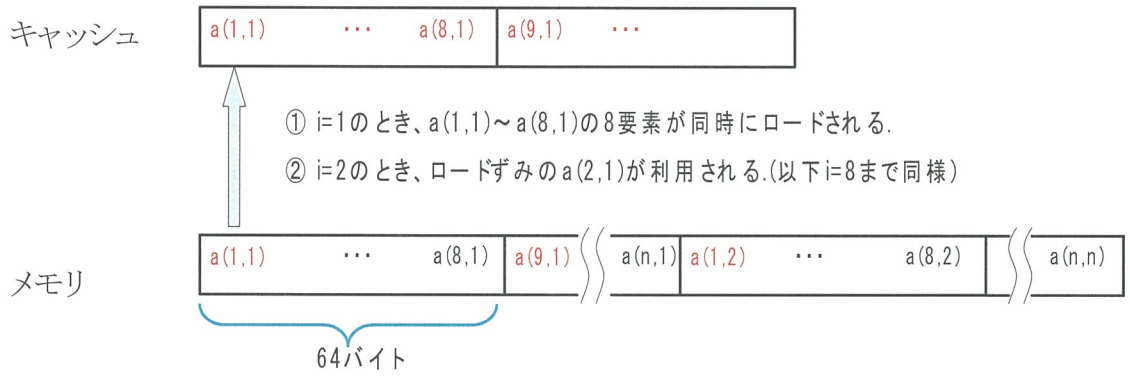


図5.2 連続アクセス

一方、つぎのプログラムは、内側ループが j であり配列 a の２次元目を動かすループとなっている。

```
s = 0.0d0
do   i = 1,n
  do   j = 1,n
       s =   s + a(i,j)
  enddo
enddo
```

i=1,j=1 のとき、a(1,1)がキャッシュにロードされ、同時に a(2,1)～a(8,1)もロードされるのは、最初の例と同様である。つぎに、j=2 のときは a(1,2)が必要になるが、a(1,2)はキャッシュにはまだロードされていないので、a(1,2)のロードが必要になる（配列 a に関する

とびアクセスが行われており、j=1 のときにロードした、a(2,1)～a(8,1)は利用されない）。以下、内側ループの j が増加するたびに、常に使用する配列要素をキャッシュにロードしなければならない。この様子を示したものが、図 5.3 である。このようなとびアクセスは性能低下を招くので、配 1 次元目を連続的に使用するようにしなければならない。

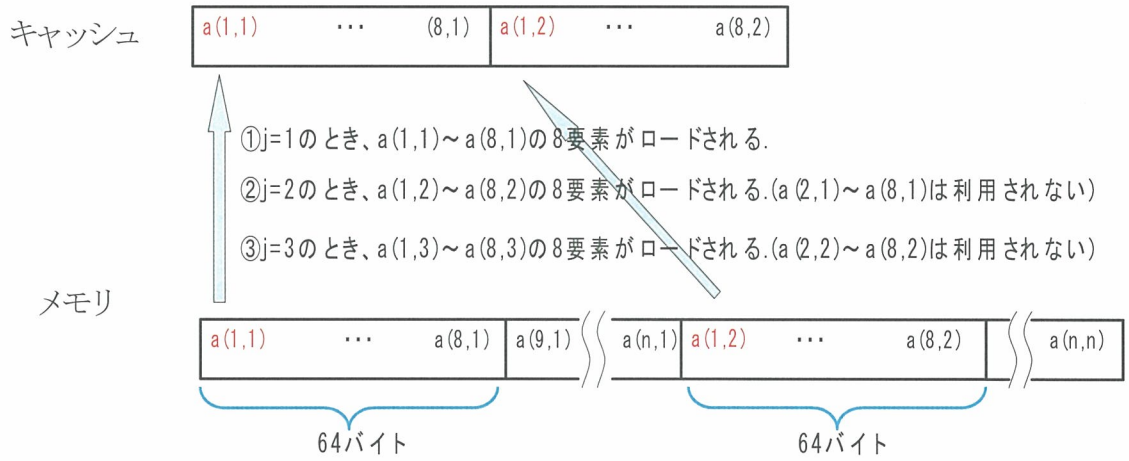


図5.3 とびアクセス

(2) キャッシュ位置の競合回避

配列の大きさが 2 のべき乗である 2 つ以上の配列が、キャッシュ上の同じ位置を占めることによって性能が低下する場合がある。これをキャッシュ競合という。

キャッシュの大きさを 16K バイト（2 の 14 乗）としたとき、メモリもキャッシュと同じ 16K バイトの大きさで区切られ、それぞれの先頭はキャッシュ上の同じ位置にロードされる。例えば、以下のプログラムは、倍精度実数型の配列 a,b の大きさがキャッシュと同じ 16K バイト（2048×8 バイト）であるため、配列要素 a(1)と配列要素 b(1)はキャッシュ上の同じ位置を占めることになる。

まず、i=1 のとき a(1)がキャッシュにロードされる（同時に a(2)～a(8)もロードされる）。つぎに、b(1)が引用されると b(1)～b(8)がキャッシュにロードされるが、b(1)は a(1)とキャッシュ上の同じ位置を占めるため、a(1)～a(8)をキャッシュから追い出してしまう。すなわち、配列 a と b の間でキャッシュ位置の競合が発生している。i が 2 に進んだとき、a(2)はキャッシュから追い出されているため、a(2)を再ロードしなければない。この様子を示したものが、図 5.4 である。

```
parameter N=2048
real(8) a(N),b(N)
    do i = 1,N
      a(i) = a(i) + b(i)
    enddo
```

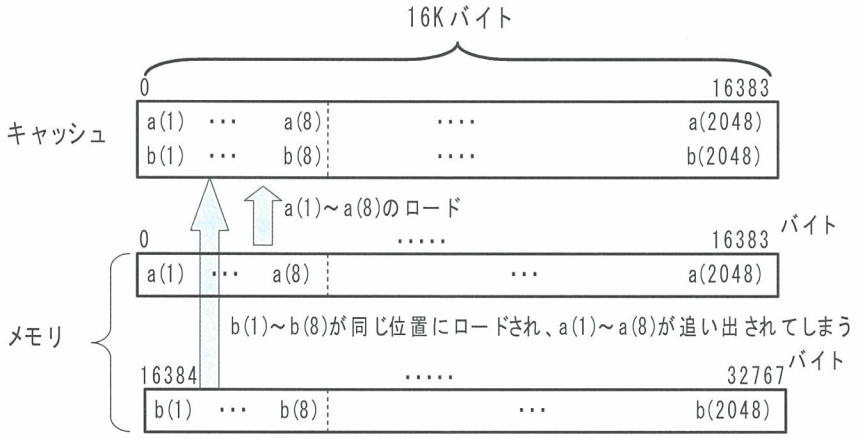


図5.4 キャッシュ位置の競合

キャッシュ位置の競合を避けるには、配列宣言において、２のべき乗とならないよう小さな値を足しこむことが有効である。つぎのプログラムでは、配列 a に８要素分の値を足しこんでいるので、図 5.5 のように配列 b(1)～b(8)は、キャッシュ上で配列 a(1)～a(8)と異なる位置を占めることになり、a(1)～a(8)の再利用が可能となる。

```
parameter N=2048,mod=８
real(8) a(N+mod),b(N)
 do   i = 1,N
     a(i) =   a(i) + b(i)
 enddo
```

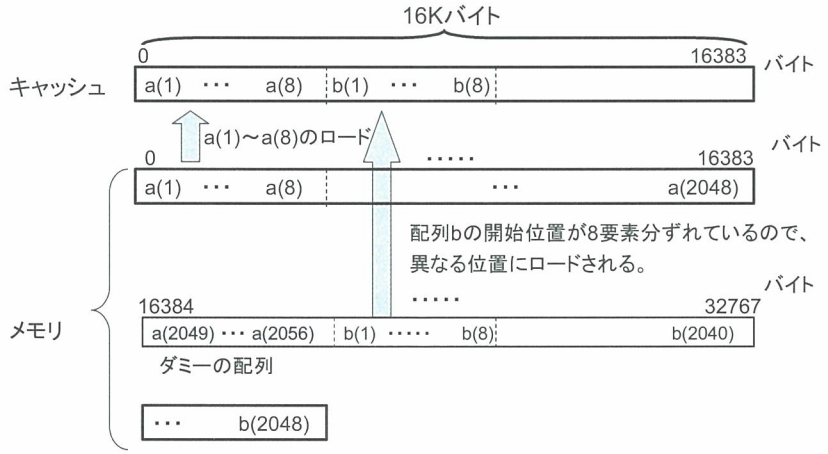


図5.5 キャッシュ競合の回避

AzusA は、L3 キャッシュとして 2M バイトのキャッシュを２個（計 4Mbyte)、L2 キャッシュは 16K バイトのキャッシュを６個（計 96Kbyte）持っている。すなわち、L2 キャッシュでは、異なる配列を６個保持できるようにキャッシュ競合の対策を行っているが、配列サイズが 16K バイトの整数倍にならないようにすることが、高速化のためには必要である。

(3) キャッシュブロッキング

プログラム中で同一のデータを何度も使う場合、それをキャッシュ上にとどめて置いて再利用することは、性能を出す上で非常に重要である。しかし、キャッシュの大きさには限りがあるので、データがキャッシュに収まるようにループを変形しキャッシュの再利用を図る技法がある。これをキャッシュブロッキングという。

図 5.6 の（Ｉ）のプログラムにおいて、配列 b をキャッシュの２倍の大きさと仮定する。このとき内側の j のループでは、配列 b(1)の要素から順にキャッシュにロードされていき、j=m/2 でキャッシュが一杯となる。すなわち、配列 b の前半部分がキャッシュに置かれている。その後、j=m/2+1 からは、配列 b の後半部分が順にキャッシュにロードされていくが、すでにキャッシュは一杯になっているので、配列 b の前半部分は順にキャッシュから追い出されていく。そして、j=m になったとき、キャッシュは配列 b の後半部分にすべて置き換わってしまう。つぎに外側ループの i が２に進んだとき、配列要素 b(1)が再び必要になるが、キャッシュは配列 b の後半部分のデータに置き換わっているため、再ロードが行われることになる。すなわち、内側 j のループでは、外側 i のループが増分するたびに、配列 b の前半部分、後半部分がキャッシュに交互にロードされることになり、キャッシュに置かれたデータの再利用が行われていない。

```
（Ｉ）do i=1,n                    （Ⅱ）  do j2=1,m,m/2
        do j=1,m                            do i=1,n
          ... = ... *b(j)                     do j=j2,j2+m/2-1
                                                ... = ... *b(j)
```

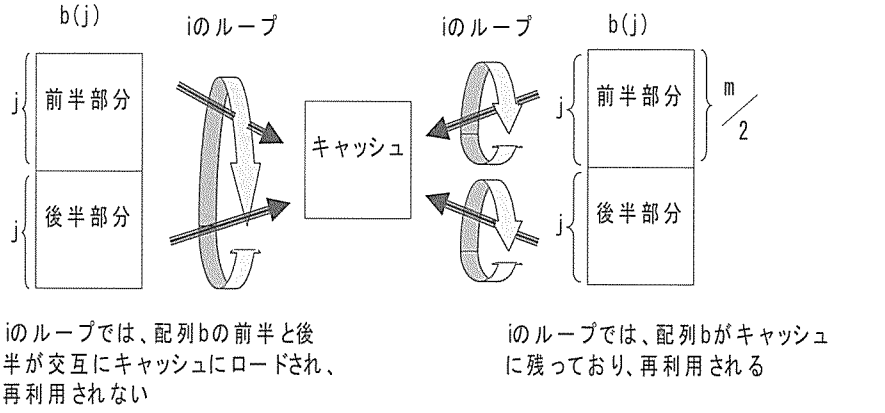


図5. 6 キャッシュブロッキング

一方、図5.6の（Ⅱ）の例では、内側ループの長さを半分にし、配列bがちょうどキャッシュに収まるようにしている(配列bの大きさがキャッシュの２倍の大きさと仮定しているため)。そして、半分の長さの内側ループを２回実行するために、最外側に新たにj2のループを設けている。内側のjのループの実行において、配列bは１番目の要素からキャッシュにロードされていき、ループが終了したとき(j=j2+m/2-1)、配列 b の前半部分がキャッシュに置かれている。つぎに外側iのループが２のとき、再びjのループが実行されることになるが、キャッシュには配列bの前半部分がすべて残っているのでキャッシュへのロードは不要であり、キャッシュのデータが再利用される。以下iのループが終了するまでキャッシュのデータを再利用しながら計算が行われる。この後、j のループの後半部分を実行するときも同様に配列bの後半部分が一度キャッシュにロードされた後は、そのままキャッシュに置かれた状態で、iのループを繰り返すことになる。

この２つを比較すると、（Ⅰ）ではキャッシュへのロードが i のループの度に発生していたが、（Ⅱ）ではキャッシュへのロードは２回ですむので性能が向上する。

５．３ メモリアクセスの削減　　－ アンローリング －

キャッシュの有効利用の他に、メモリアクセスそのものを減らしてしまうことも高速化には効果がある。例えば、図5.7の（Ⅰ）のプログラムにおいて、内側のjのループで使用するb(j)は、外側ループのiが増加するたびに毎回メモリからロードしなければならない。一方、（Ⅱ）のプログラムでは、外側ループを４個おきに回し、その分を内側ループに展開している。この手法をアンローリングという。これによりb(j)のロードは1/4に削減されている。

このような単純な構造ではコンパイラによって自動的にアンローリングされるが、複雑なループ構造の場合は、アンローリングされないことがある。実際にアンローリングが行われたかは、オプション-opt_reportを指定したときに出力される最適化リストで調べることができる。

（Ⅰ）
```
do i=1,n
  do j=1,m
    a(j,i)=a(j,i)*b(j)
  enddo
enddo
```

⇨

（Ⅱ）
```
do i=1,n-3,4
  do j=1,m
    a(j,i)=a(j,i)*b(j)
    a(j,i+1)=a(j,i+1)*b(j)
    a(j,i+2)=a(j,i+2)*b(j)
    a(j,i+3)=a(j,i+3)*b(j)
  enddo
enddo
```

図5.7　アンローリング

５．４ ソフトウェアパイプライニング（ＳＷＰ）の促進

コンパイラは、CPU の性能を活かすために、ソフトウェアパイプライニングの技法（図5.8）を用いたループの最適化を行い、さらに命令並べかえ（スケジューリング）を行って、できるだけ少数の命令でかつ１サイクルでできるだけ多くの命令を同時に実行できるような高速のコードを生成する。ソフトウェアによるパイプライニングとは、ハードウェアによるパイプライン化と同じような形で、ループの繰り返しをオーバーラップさせことによって、依存関係のない命令を同時に実行するテクニックである。

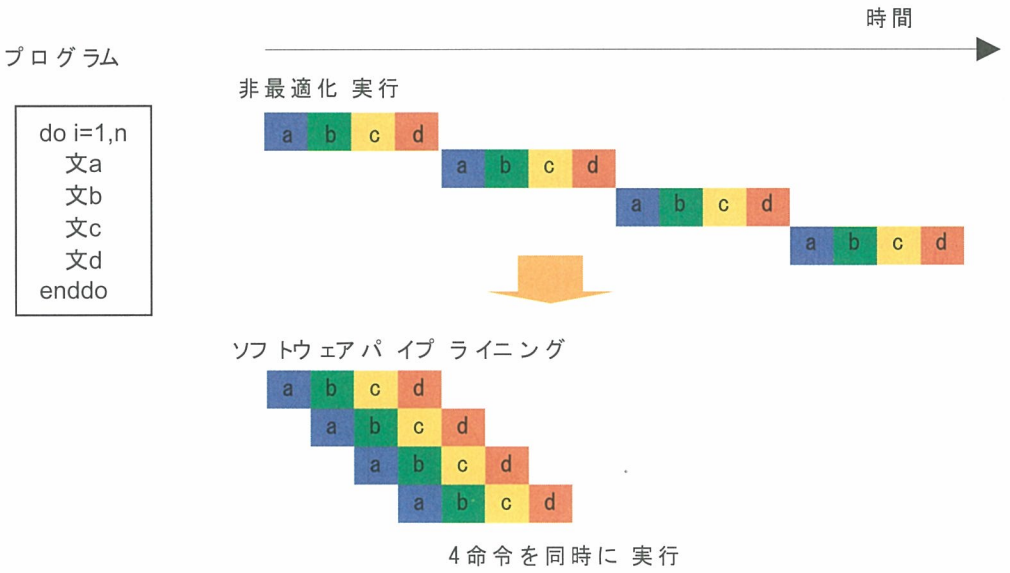


図 5.8　ソフトウェアパイプライニング

しかし、ループの条件によっては、コンパイラでソフトウェアパイプライニング化が行われない、または行われても効果がでない場合がある。コンパイル時に-opt_report オプションを指定することにより、ソフトウェアパイプライニング化を行っているか否か、また、できない場合はその理由を表示することができる。

主な理由は以下の通りである。

- ループの繰り返しをまたいだデータ依存関係がある
- ループ中に複雑な if 文がある
- ループ中に call 文または関数呼び出しがある

このレポートに沿って、適宜対処しソフトウェアパイプライニング化を促進し、プログラムの高速化を図ることができる。

５．５　ファーストタッチ　－ 並列処理性能の向上 －

AzusAでは、４つのCPUと4GB/8GBのメモリから成るセルを４個接続したcc-NUMAアーキテクチャを採っているので、並列処理の場合は動作するCPUとメモリを同一セルに保つことが高速化に有効となる。

図5.9の（Ｉ）のプログラムにおいて、ループ１はコンパイラの自動並列化機能により、最外側のkのループで並列化され実行される。そして、その領域が実行を受け持つCPUの属するセルのメモリに分散して確保される。すなわち、最外側kのループで並列化されたとき、k=1の実行はCPU#1が受け持ち、配列a(1,1,1) ～ a(16,16,1)をセルCell#1のメモリ上に確保する。また、k =16の実行はCPU#16が受け持ち、配列a(1,1,16) ～ a(16,16,16)をCell#4のメモリ上に確保する。このときの関係を示したものが図5.9である。このように、ある配列へのストアが初めてであれば（ファーストタッチという）、実行を受け持つCPUの属するセルのメモリにその配列の領域が分散して確保される。

つぎのループ２はループ１と同様、最外側の k のループで並列化される。そのため、CPU#1 で使用する配列要素 a は、ループ１のファーストタッチで確保した Cell#1 のメモリ内の配列（図 5.10）をアクセスすることになり、ループ１と同様に CPU とメモリは同一セル内に保たれることになる。

```
（Ⅰ）
do k = 1,16                    ┐
  do j = 1,16                  │
    do i = 1,16                │
      a(i,j,k) = ...           │ ループ１
    enddo                      │
  enddo                        │
enddo                          ┘
do k = 1,16                    ┐
  do j = 1,16                  │
    do i = 1,16                │
      a(i,j,k) = ...           │ ループ２
    enddo                      │
  enddo                        │
enddo                          ┘

（Ⅱ）
do k = 1,16                    ┐
  do j = 1,16                  │
    do i = 1,16                │
      a(i,j,k) = ...           │ ループ１
    enddo                      │
  enddo                        │
enddo                          ┘
do i = 1,16                    ┐
  do j = 1,16                  │
    do k = 1,16                │
      a(i,j,k) = ...           │ ループ２
    enddo                      │
  enddo                        │
enddo                          ┘
```

図 5.9　多重ループ

kの値	CPU番号	該当CPUが処理する配列a	CPUを搭載しているセル番号
1	CPU #1	a(1,1,1) ～ a(16,16,1)	Cell#1
2	CPU #2	a(1,1,2) ～ a(16,16,2)	Cell#1
3	CPU #3	a(1,1,3) ～ a(16,16,3)	Cell#1
4	CPU #4	a(1,1,4) ～ a(16,16,4)	Cell#1
5	CPU #5	a(1,1,5) ～ a(16,16,5)	Cell#2
6	CPU #6	a(1,1,6) ～ a(16,16,6)	Cell#2
7	CPU #7	a(1,1,7) ～ a(16,16,7)	Cell#2
8	CPU #8	a(1,1,8) ～ a(16,16,8)	Cell#2
9	CPU #9	a(1,1,9) ～ a(16,16,9)	Cell#3
10	CPU #10	a(1,1,10) ～ a(16,16,10)	Cell#3
11	CPU #11	a(1,1,11) ～ a(16,16,11)	Cell#3
12	CPU #12	a(1,1,12) ～ a(16,16,12)	Cell#3
13	CPU #13	a(1,1,13) ～ a(16,16,13)	Cell#4
14	CPU #14	a(1,1,14) ～ a(16,16,14)	Cell#4
15	CPU #15	a(1,1,15) ～ a(16,16,15)	Cell#4
16	CPU #16	a(1,1,16) ～ a(16,16,16)	Cell#4

図 5.10 配列とセルの関係

一方、（II）のプログラムにおいて、ループ１の構造はプログラム（I）と同様最外側ループ k で並列化され、メモリも k に関してセルに分散される。しかし、ループ２は、ループ１とは異なり最外側の i のループであるため、i で並列化されることになる。このとき、i=1 の実行は、CPU#1 が受け持ち、同時に配列要素 a(1,1,1)～a(1,16,16)の演算を受け持つことになるが、配列 a はループ１で図 5.10 のようにメモリに割り付けられているので、CPU#1 はすべてのセルのメモリへのアクセスが必要になり性能が低下する（他の CPU も同様）。このように cc-NUMA アーキテェクチャを採っている AzusA ではファーストタッチした時のループ構造と、それ以降のループを同じ構造にすることが必要である。

尚、OpenMP により並列化し実行する場合は、本体部分の計算にのみ並列化指示行を入れ並列実行し、初期化部分に指示行を入れなかった場合、これと同様のことが起きるので、初期化部分も並列化の指定が必要である。

6　高速化推進研究活動の成果

本センターの大規模科学計算システムは、コンピュータの処理能力を順次整備してきているが、利用者ジョブは、このシステムの強化に応じて規模が拡大してきている。このような状況から、利用者プログラムのシミュレーションモデルは大規模化、高精度化されてきていると言える。従って、スーパーコンピュータの運用設計は、ジョブの大規模化、長時間化に対して的確に対応するため、1ノード32台のCPUを一つのジョブで占有して使用できるように並列処理を運用の中心にして、さらに長時間ジョブは実行時間の推定が困難であるため、CPU時間を制限しないような利用環境を整備してきた。これにより従来不可能であった大規模・長時間ジョブの実行できるようになっている。実際2000年には、100時間以上のジョブで占めるCPU時間の割合は、全体の約75%に達し、2001年以降も同じようなジョブ状況が継続している。並列コンピュータの運用設計は、並列処理向けに16CPUの占有と、長時間ジョブへの対応としてCPU時間を制限しないような利用環境を整備してきた。これによりスカラー向きのジョブ規模も大きくなり、2002年末にはCPU利用状況は約80%に達している。

現在、10,000時間を越えるジョブも日常的に実行されており、このような大規模・長時間ジョブは、スーパーコンピュータでの実行であっても32台のCPUを使用して2週間を要することもあり、ベクトル化率と並列化率を極限まで高めておく必要がある。このような高速化を推進してゆくためには、スーパーコンピュータのハードウェア、コンパイラについての深い知識が要求されることと、さらに自動ベクトル化、自動並列化、および高速化チューニングでは利用者プログラムに積極的に関わっていく必要がある。そこで、本センターでは高速化を受け持つ担当者を置き、利用者プログラムの高速化を支援してきている。

6．1　スーパーコンピュータの高速化推進研究活動（２００１～２００２）

今回、高速化支援を行ったプログラムのうち主なものについて概略を表6.1に報告する。また、高速化推進研究活動の成果の一部を研究論文として本報告の最後に掲載する。

表6.1　高速化支援性能向上比

プログラム番号	オリジナル比	
	単体性能	並列性能
1	—	1.5倍
2	3.5 倍	—
3	1.9 倍	—
4	4.1 倍	—
5	2.0 倍	3.8倍
6	1.4 倍	4.6倍
7	1.1 倍	1.1倍

６．２　スーパーコンピュータ　ＳＸの使用状況

本センターの SX-4 システムは 2002 年 12 月まで５年間運用しているが、この期間の利用者ジョブのベクトル化率と並列化率の状況と、ジョブの CPU 時間使用分布状況を表 6.2 に示す。

表6.2　ＳＸ　ＣＰＵ使用状況分布

ベクトル化率＼並列化率	0%	10%	20%	30%	40%	50%	60%	70%	80%	90%
90%	7	8	10	11	17	35	44	83	137	553
80%	0	2	0	1	1	3	2	6	4	21
70%	0	0	2	1	0	1	4	2	1	9
60%	0	0	0	0	0	0	1	0	3	8
50%	0	0	0	0	0	0	0	0	0	7
40%	0	0	0	0	0	0	0	0	0	2
30%	0	0	0	0	0	0	1	0	0	1
20%	0	0	0	0	0	0	0	0	0	1
10%	0	0	0	0	0	0	0	0	0	1
0%	1	0	0	0	0	0	0	0	0	2

並列化率

備考：マトリックスの値は、ジョブCPU時間の1000分率（全CPU時間比）

表 6.2 は、SX-4 で実行した利用者ジョブを、ベクトル化率と並列化率とで分類した CPU 時間の割合を、全 CPU 時間に対して 1000 分率で表したものである。この表から、高速化推進の主要な指標であるベクトル化率と並列化率は次のような状況である。

ベクトル化率と並列化率の分類	CPU 時間の割合
・ベクトル化率と並列化率の双方が 90%以上のジョブ	約 55%
・ベクトル化率が 90%以上のジョブ	約 90%
・並列化率が 90%以上のジョブ	約 60%

このように、ベクトル化率、並列化率が高い効率的な実行がされているジョブの割合が高い状況は、高速化推進研究活動の成果が現れていると考えられる。

さらに、本センターでは利用者からの高速化要求に対応して、コンパイラの開発部門及びハードウェア開発部門との連携をとり、高速化の最善施策を検討し、具体的な解決を実施してきている、また、同時に科学計算ライブラリの開発部門との連携により、ライブラリ適用での支援、さらには必要に応じてベクトル化、並列化での個別支援を受ける体制で推進しており、高速化の最適解を目指している。

６．３　今後の取り組み

2003年1月から稼動しているSX-7システムでは、8GBメモリを超えるジョブ数の増加、あるいは新たな利用者の大規模なジョブなど、シミュレーションのジョブ特性に規模拡大の変化が見られている。今後の稼動状況推移を見なければ充分な傾向を計れないが、SX-7システムではベクトル演算性能が８倍、メモリ容量が32倍に拡大したことによって、利用者プログラムのシミュレーションモデルの規模が大きくなり、ジョブの大規模化・長時間化が進みつつあると考えられる。

しかし、高速化を必要とするジョブも一部見うけられる傾向があり、今後も、高速化推進研究活動を継続して実施して行く必要がある。

研究論文

1　王　建青　　藤原　修
「携帯電話に対する人体ドシメトリの計算機システムシミュレーション」
SENAC　Vol.36，No.1　（2003.1）

2　遠藤新一，松原史卓，佐々木崇徳
「超薄膜磁性体の磁区のコンピュータシミュレーション」
SENAC　Vol.36，No.1　（2003.1）

携帯電話に対する人体ドシメトリの計算機シミュレーション

王　建青　　藤原　修
名古屋工業大学 電気情報工学科

1．まえがき

携帯電話はその利便性のゆえに爆発的に普及しており，その勢いは留まることを知らない．一方，携帯電話使用時には枢要器官の頭部は局所的に強い電磁界に曝(さら)されることになり，その安全利用を目的とする局所吸収指針が世界各国において相次いで制定されている．一般に電波と呼ばれる高周波電磁波の人体影響は，内部組織の温度上昇に起因するものとされ，携帯電話を対象とした上述の安全指針もこれを共通のベースとしている．そのとき，評価尺度としては単位体重当たりの吸収電力，すなわちSAR（Specific Absorption Rate，単位W/kg）が用いられ，SARを定量することを「ドシメトリ（Dosimetry)」という．しかしながら，人体内部のSARは直接測定が不可能なため，ドシメトリ評価には数値モデルによる計算機シミュレーションかファントムと呼ばれる擬似人体モデルでの測定に頼らざるを得ない．携帯電話のドシメトリの計算評価には人体頭部を解剖学的に詳細に模擬した数値モデルが用いられ，そのドシメトリのシミュレーションも数値モデルの分解能の向上とともに大規模となり，大型計算機の利用が欠かせないものとなっている．

本文では，携帯電話に対する人体頭部ドシメトリの計算手法の概要を述べ，東北大学情報シナジーセンターのスーパーコンピュータを利用した筆者らのシミュレーション例[1]を紹介する．また，プログラムのベクトル化と並列化の向上手法についても述べる．

2．計算法

ドシメトリの指標となるSARは，生体組織内の電界をE，導電率をσ，密度をρとすれば，$SAR=\sigma E^2/2\rho$で計算される．従って，携帯電話による頭部ドシメトリの計算評価は，結局生体組織内の電界Eを求めることに帰着し，一般にFDTD（Finite-Difference Time-Domain）法[2]が用いられる．FDTD法とは，電界E，磁界Hに関するマクスウェルの方程式を時間領域と空間領域とで差分化し，その差分式を時間領域で逐次計算することで計算領域内の電磁界を数値的に求める手法をいう．電

界E，磁界Hに関するマクスウェルの方程式についてFDTD定式化を行うと，例えば電界Eと磁界Hのz成分はそれぞれ

$$E_z^{n+1}(i,j,k)=\frac{1-\sigma\Delta t/2\varepsilon}{1+\sigma\Delta t/2\varepsilon}E_z^n(i,j,k)+\frac{\Delta t/\varepsilon}{1+\sigma\Delta t/2\varepsilon}\left[\frac{H_y^{n+1/2}(i,j,k)-H_y^{n+1/2}(i-1,j,k)}{\Delta x}-\frac{H_x^{n+1/2}(i,j,k)-H_x^{n+1/2}(i,j-1,k)}{\Delta y}\right] \tag{1}$$

$$H_z^{n+1/2}(i,j,k)=H_z^{n-1/2}(i,j,k)+\frac{\Delta t}{\mu}\left[\frac{E_y^n(i+1,j,k)-E_y^n(i,j,k)}{\Delta x}-\frac{E_x^n(i,j+1,k)-E_x^n(i,j,k)}{\Delta y}\right] \tag{2}$$

で表せる．ここで，式(1)，式(2)のμ，ε，σはそれぞれ媒質の透磁率，誘電率，導電率である．空間領域における差分化として，計算対象を三次元の微小立方体か直方体（セル）に分割し，各セルに生体組織と対応する電気定数（誘電率εと導電率σ）を割り付ける．

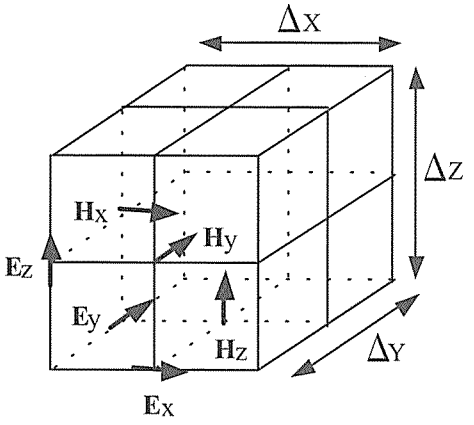


図１　FDTD単位セル内の電磁界配置

```
CALL SETUP
T=0
DO N=1,N_TIMESTEP
  CALL E_FIELD_CALCULATION
  CALL E_FIELD_PML
  T=T+DT/2
  CALL H_FIELD_CALCULATION
  CALL H_FIELD_PML
  T=T+DT/2
END DO
```

図２　FDTD　プログラム例

図１は単位セル内の電磁界各成分の空間配置を示す．基本的には電界はセルの各辺に沿って，磁界は面の中心に垂直に割り当てられ，電界と磁界は空間的に相互に配置される．これは，電界の回転が磁界を，磁界の回転が電界を作るというマクスウェルの方程式を満たすような配置となっている．一方，時間軸においては電界と磁界は時

間的に相互に配置されることになる．例えば，電界を t=nΔt (Δt: 時間ステップ)の整数次の時刻に，磁界を t=(n+1/2)Δt の半整数次の時刻にそれぞれ割当て，t=(n-1)Δt の電界 E^{n-1} と t=(n-1/2)Δt の磁界 $H^{n-1/2}$ とから E^n を，$H^{n-1/2}$ と E^n とから $H^{n+1/2}$ をそれぞれ計算する，というように電界，磁界が順次計算される．特に，式(1)，(2)からもわかるように，(i, j, k) での電界を計算するためにはその前のΔt/2 時刻の隣接磁界，同様に (i, j, k) での磁界を計算するためにはその前のΔt/2 時刻の隣接電界が用いられるため，FDTD 法は大型計算機の得意とする大規模並列計算に特に適しているといえる．なお，計算機で取り扱える解析領域は有限であるため，解析領域を仮想な境界で閉じておく必要がある．この仮想的な境界を吸収境界といい，その条件を吸収境界条件という．代表的な吸収境界条件は，境界に仮想的な吸収媒質を置いて入射波を減衰させる PML（Perfectly Matched Layer）である．

FDTD 計算のプログラム例は図 2 に示す．

3．頭部内 SAR のシミュレーション例

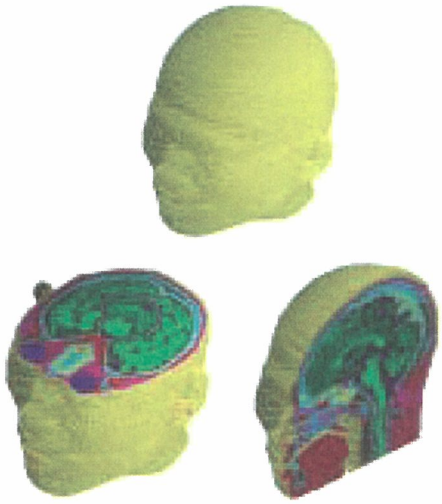


図 3　頭部数値モデル

図 3 は日本人成人男性の頭部 MRI(Magnetic Resonance Imaging)データから筆者らが製作した頭部数値モデルを示す．MRI データは，各水平断面において 256 x 256 ピクセル（約 1 mm 四方）の空間分解能，9 ビットグレースケールの濃淡分解能を有する．この MRI データを基に，各ピクセルを放射線医師の指導の下で，17 種類の RGB(Red-Green-Blue)コードのいずれかに指定することで皮膚，脂肪，筋肉，骨，脳など 17 種類の組織を同定した．こうして得られた頭部数値モデルは，一辺 2mm の立

方体セルを約 53 万個集積して構成されている．

図 4 は携帯電話機による頭部垂直断面内の SAR 空間分布を示す．なお，図(a)は携帯電話機のアンテナが伸張時，図(b)は収納時のものである．図から，SAR の最大値は携帯電話機側の頭部表面耳付近で生じ，そこから遠ざかるに従って SAR は減衰し，頭部内部にはホットスポットは形成されないことがわかる．また，アンテナ収納時には短いアンテナ部及び筐体上方に電流が集中した結果，高 SAR 領域は耳付近に集中するが，アンテナ引き出し時にはそれが頭部上方に分散され，曝露(ばくろ)領域が広くなるもののピーク SAR 値は低くなる．

このように，頭部内ＳＡＲ分布をシミュレーションすることで，人体に対する安全性評価を行うことが可能となる．

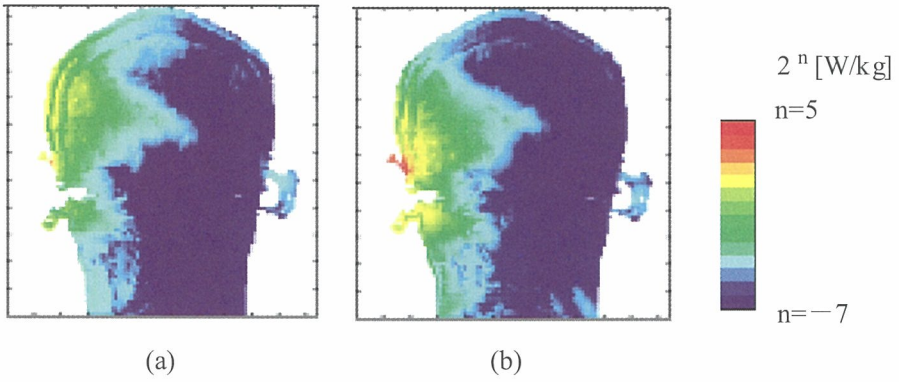


図４　携帯電話による頭部垂直断面内のSAR空間分布
(a)アンテナ伸張時；(b)アンテナ収納時

４．FDTD コードの性能向上

前述ドシメトリで用いた筆者らの FDTD コードの性能向上にあたっては，東北大学情報シナジーセンターから有益な助言と指導を頂いた．この FDTD コードは当初 Fortran 77 で作成されたもので，PC Unix 上で使用することを想定し，演算は全て単精度で行われている．しかし，スーパーコンピュータ(NEC SX-4)は演算を倍精度で行うために，型宣言が単精度の場合には，単精度⇔倍精度間の変換が必要になり，単精度版のままでは演算が遅くなる．これに対処するために，プログラムの先頭に全ての変数を倍精度に指定するように

IMPLICIT REAL*8　(A-H,O-Z)

を追加することにした．

FDTD 計算における電界と磁界の計算部は，式(1)と(2)からもわかるように高ベクト

ル化率と高並列化率が容易であるが，PML 吸収境界条件の部分は，一般に PML の層

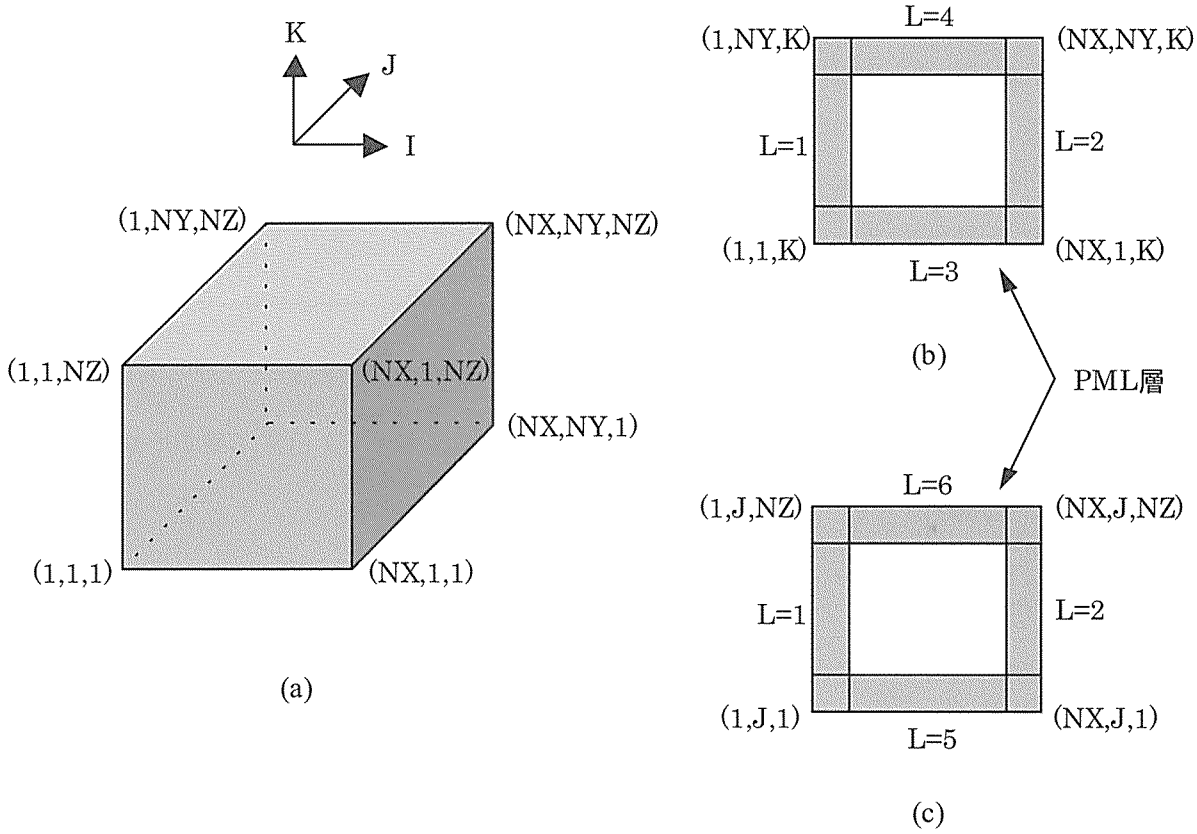


図５　FDTD解析空間とPML各層の配置
(a) 全FDTD解析空間；(b) I J平面断面図；(c) I K平面断面図

数 M が 10 前後（筆者らのプログラムでは M=12）しかなく，ベクトル化と並列化においては工夫を要する．図５に FDTD 解析領域と PML 媒質の配置を示す．PML 媒質層は解析領域を囲んで L=1 から 6 まで計 6 面がある．図 6 に電界に対する E_FIELD_PML の例を示す．このプログラムは文献２で紹介されたもので，広く用いられているようである．PML 媒質層（図中グレーの領域）が解析領域を囲んで 6 面があるので，各 PML 媒質の始点の座標，終点の座標を LPMLII(L,1), LPMLJJ(L,1), . . . , LPMLKK(L,2)によって定めている．ただし，L=1 は，I=1 に接する PML 層，L=2 は I=NX に接する層，同様に，例えば，L=6 は K=NZ に接する層としている．このプログラムは見かけ上非常にコンパクトに書かれているが，次のような問題点がある．

（１）　ベクトル化率を向上させるためには，最内側のループの長さが最大になるようなループ構成にする必要があるが，図 6 のプログラムでは L=5 と 6 のときに最内側の K に関するループは 11 か 12 の長さしかなく，ベクトル長は極めて短

い．例えば，E_FIELD_PML の平均ベクトル長は理想値 256 に対して僅か 31 であった．

（2） 並列化の効率を向上させるためには，できるだけ外側のループで並列化する必要があるが，図 6 のプログラムでは，最も外側の L のループは長さが 6 と短い．これでは，例え 16 や 32 個の CPU で並列演算を行わせても並列化の効果は実質的に発揮されない（後述の［備考］参照）．

```
DO L=1,6
  I0=LPMLII(L,1)
  I1=LPMLII(L,2)
  J0=LPMLJJ(L,1)
  J1=LPMLJJ(L,2)
  K0=LPMLKK(L,1)
  K1=LPMLKK(L,2)

  L1=LPMLST(L)
  DO I=I0,I1-1
    DO J=J0+1,J1-1
      DO K=K0+1,K1-1
        EX_FIELD_CALCULATION
        L1=L1+1
      END DO
    END DO
  END DO

  L2=LPMLST(L)
  DO I=I0+1,I1-1
    DO J=J0,J1-1
      DO K=K0+1,K1-1
        EY_FIELD_CALCULATION
        L2=L2+1
      END DO
    END DO
  END DO

  L3=LPMLST(L)
  DO I=I0+1,I1-1
    DO J=J0+1,J1-1
      DO K=K0,K1-1
        EZ_FIELD_CALCULATION
        L3=L3+1
      END DO
    END DO
  END DO
END DO
```

図 6　E_FIELD_PML のプログラム例（文献 2 より）

```
DO K=2,M
  DO J=2,NY-1
    DO I=1,NX-1
      EX_FIELD_CALCULATION
    END DO
  END DO
END DO

DO K=M+1,NZ-M
  DO J=2,M
    DO I=1,NX-1
      EX_FIELD_CALCULATION
    END DO
  END DO
  DO I=1,M
    DO J=M+1,NY-M
      EX_FIELD_CALCULATION
    END DO
  END DO
  DO I=NX-M,NX-1
    DO J=M+1,NY-M
      EX_FIELD_CALCULATION
    END DO
  END DO
  DO J=NY-M+1,NY-1
    DO I=1,NX-1
      EX_FIELD_CALCULATION
    END DO
  END DO
END DO

DO K=NZ-M+1,NZ-1
  DO J=2,NY-1
    DO I=1,NX-1
      EX_FIELD_CALCULATION
    END DO
  END DO
END DO

以下EY,EZ同様
```

図 7　E_FIELD_PML のプログラム例（改良版）

実際に，SX-4 でのトレース解析結果によれば，FDTD 全計算時間の 6 割が PML の部分で費やされていることがわかった．この問題を対処する手段として，まず，変数 L をなくし，従来の L，I，J，K に関する 4 重ループを I，J，K に関する 3 重ループに変更した．その結果，プログラム上では I，J，K に関する 3 重ループが従来の 3 つから 18 になる．つぎに，ベクトル化率を向上させるために，最内側のループ変数をループ長の長いものにした．即ち，L=1 と 2 に対応する PML 層については J を最内側のループ変数，L=3～6 に対応する PML 層については I を最内側のループ変数にし，ループ長が PML の層数 M しかないような変数を最内側のループにしない．さらに，並列化の効率を上げるために，最外側のループをできるだけループ長の長いものに変更した．このような考え方に基づいた電界に関する PML のプログラム例を図 7 に示す．なお，磁界に関する PML も全く同様である．

表 1 にベクトル化と並列化の向上効果を示す．このとき NX=NY=NZ=251，時間ステップ数 N_TIMESTEP=100 とした．表から，プログラム全体の平均ベクトル長は 75.8 から 231.2 に 3 倍以上（PML だけの平均ベクトル長が 242 まで）向上したこと，またこのベクトル化の向上で演算時間が 51.1%に低減されたことがわかる．さらに，従来のプログラムでは L が最大 6 なので，16 並列しても 8 並列時と同様な演算時間を要したが，プログラムの改良で並列化の効果が向上し，改良前に比べて 8 並列では 74%，16 並列では 82%の演算時間の短縮ができた．

表 1　実行時間と平均ベクトル長

	従来			改良後		
	並列せず	8並列	16並列	並列せず	8並列	16並列
実行時間（相対値）	100.0	27.6	27.7	51.1	7.2	4.9
平均ベクトル長	75.8	75.8	75.8	231.2	231.2	231.2

［備考］

図 6 のプログラムは，自動並列化のオプション-Pauto だけでは I のループでの並列化となり，粒度（並列実行される部分の時間）が小さくて並列化の効率は悪い．そこで，つぎの並列化指示行を「DO L=1,6」の直前に指定して，L のループで並列化し，粒度を大きくすることもできる．ただし，この場合でも，L の長さは 6 なので，6 個

の CPU しか使われない.

```
!cdir paralleldo private(I0,I1,J0,J1,K0,K1,L1,L2,L3)
```

ここで,「private()」は並列化に際して CPU ごとに確保すべき変数を指定したものである.

5. むすび

スーパーコンピュータを利用した携帯電話に対する頭部ドシメトリに関する筆者らの FDTD 計算例を紹介し, PML 吸収境界条件のベクトル化と並列化の向上手法を述べた. FDTD 法は極めて並列化に向くアルゴリズムであり, スーパーコンピュータ上でその威力が一層発揮される. 人体を解析対象とする研究分野では, 人体数値モデルの分解能や組織数は年を追うごと高精度かつ大規模になっており, スーパーコンピュータの利用価値は一層高まるものと予想される.

謝辞

本研究での計算の一部は, 東北大学情報シナジーセンターのスーパーコンピュータを使用して行われたものである. 計算コードのベクトル化・並列化にあたっては, 情報シナジーセンターから有益な指導と助言をいただいた.

文献

[1] J. Wang and O. Fujiwara, "Dosimetry in the human head for portable telephones," in *The Review of Radio Science 1999-2002*, Edited by W.R. Stone, Wiley-Interscience, 2002, pp.51-63.

[2] 宇野　亨, FDTD 法による電磁界およびアンテナ解析, コロナ社, 1998.

超薄膜磁性体の磁区のコンピュータシミュレーション

遠藤新一　松原史卓　佐々木崇徳
東北大学大学院工学研究科応用物理学専攻

1　はじめに

近年のエピタキシャル結晶成長技術は Cu, Ag 等の非磁性金属基板上に厚さ数原子層の超薄膜磁性体を作ることを可能にした. これらの超薄膜磁性体では通常のバルク磁性体とは異なる様々な性質が観測されている. Cu 基板上に成長した超薄膜 Fe/Cu では, 膜厚が大きくなると磁化の方向が膜面に垂直から平行に変わる再配列転移が観測される [1]. この再配列転移は温度を上げていったときにも観測される. 逆に Ni/Cu では磁化の方向が膜面に平行から垂直に変わる再配列転移が起こる [2]. 又, 異なる成長条件下で作製した超薄膜 Fe/Cu では膜厚を増していくと表面層のみ磁性を持つ不思議な現象も報告されている [3]. また, このような超薄膜磁性体でも磁区が観測されるが, その形状は膜厚や温度に依存して様々な様相を示す [4].

従来, 磁性体の巨視的な磁気構造 (磁区) の研究はマイクロマグネティックモデルを使ってなされてきた [5]. これは, 磁性体を N 個の小磁石に分割して, これらの小磁石の安定な配向を求めるものである. しかし, 数原子層からなる超薄膜では, このモデルが使えないことは明らかである. 他方, ミクロなハミルトニアンをもとにした超薄膜磁性体の磁区構造の研究も行われている [6]. これは磁気構造の第一原理計算とも言われるものであり, マイクロマグネティックモデルによる研究と相補的なものである. このミクロなハミルトニアンに基づく計算には莫大な計算時間がかかるため, この方面からの研究は２次元モデルに限られてきた. 超薄膜磁性体の磁区構造の本質を明らかにするためには, 厚さを持った積層膜の大規模シミュレーションを実行する必要がある. 最近筆者らの研究室で開発された方法によってこの計算時間が大幅に短縮され, 積層膜の磁気構造の研究が可能になりつつある. ここではその一端を紹介する.

2 モデル及び方法

簡単のため, $L \times L$ の2次元正方格子が L_z 層積み重なった積層格子を考える. ここで, 積層面方向には周期境界条件, 積層面と垂直な方向には開放端境界条件を課す. 系のエネルギーは次のハミルトニアンで与えられる.

$$\begin{aligned} \mathcal{H} &= -J\sum_{\langle ij\rangle} \boldsymbol{S}_i \cdot \boldsymbol{S}_j - K \sum_{i\in \text{surface}} (S_i^z)^2 \\ &+ D\sum_{i>j} \left\{ \frac{\boldsymbol{S}_i \cdot \boldsymbol{S}_j}{r_{ij}^3} - 3\frac{(\boldsymbol{r}_{ij}\cdot \boldsymbol{S}_i)(\boldsymbol{r}_{ij}\cdot \boldsymbol{S}_j)}{r_{ij}^5} \right\}. \end{aligned} \tag{1}$$

ここで, $\boldsymbol{S}_i(|\boldsymbol{S}_i| = 1)$ は古典ベクトルスピンであり, $\langle ij\rangle$ は最近接スピン対を表す. 第一項は交換相互作用エネルギー, 第二項は表面異方性エネルギー, 第三項は磁気双極子相互作用エネルギーを表し, J, K 及び $D(\equiv (g\mu_B S)^2/a^3)$ はそれらの強度を表す定数である. a は正方格子の格子定数であり, また $c = a$ とする. ここで c は積層面間の距離である. 変数 $\boldsymbol{r}_{ij}$ は i スピンと j スピンの相対位置ベクトルを格子定数 a で計ったものである. 以下, x 軸と y 軸を面内にとり z 軸をそれらと垂直方向にとる.

ここではモンテカルロ (MC) 法を用いる. この方法の最大の利点は熱揺らぎが自然に取り込まれ, 有限温度の性質が容易に調べられる点にある. MC法は以下の手順で実行される: (1) 一つのスピンに着目し, このスピンにかかる他の全てのスピンからの有効場を計算する. (2) この有効場をもとに, ボルツマン重率を求め, それに従って, このスピンの方向を再配置 (更新) する. (3) この操作を全てのスピンに対して実行する (1MCステップ). (4) これを多数回繰り返して物理量の平均値を求める. 実行上の最大の問題は磁気双極子相互作用の取り扱いである. この相互作用は交換相互作用に比較して非常に小さい $(D/J \sim 10^{-3})$ が全てのスピン間に働くために無視できない. 事実この磁気双極子相互作用が磁区形成の原因となっている. 計算時間のほとんどが双極子場の計算に費やされることは言うまでもない. 我々の研究室ではこの困難を緩和する効率的なアルゴリズム, 双極子場不連続更新MC (DUDMC) 法を開発した. この詳細は既に報告してあるのでそちらを参照して戴きたい [7]. この方法により, 積層膜の磁気構造の研究が可能になりつつある.

さて, モンテカルロ法では磁気構造だけでなく比熱や磁化等の温度特性も調べられる. これらの物理量の測定は, 磁気構造の変化を見る上で非常に重要である. 比熱 C, 積層面に平行な磁化 (面内磁化) $M^{\parallel}$, 積層面に垂直な磁化 (面直

磁化) $M^{\perp}$ は以下の式を用いて計算される.

$$C = \frac{1}{NT^2}(\langle \mathcal{H}^2 \rangle - \langle \mathcal{H} \rangle^2), \tag{2}$$

$$M^{\parallel} = \frac{1}{N}\langle \sqrt{(M^x)^2 + (M^y)^2} \rangle, \tag{3}$$

$$M^{\perp} = \frac{1}{N}\langle |M^z| \rangle, \tag{4}$$

ここで $M^{\nu} = \sum_i S_i^{\nu}$ $(\nu = x, y, z)$, N はスピン数, T は温度 ($k_{\mathrm{B}} = 1$ の単位系をとる) であり, また $\langle \cdots \rangle$ はMCステップによる平均を表す. 計算は J を基準として $K = J$, $D = 0.05J$ の場合について行った. 又, 平面格子サイズは 48×48, 64×64, 96×96 とし、膜厚は $L_z = 2 \sim 6$ とした. D 及び K の値は実際の超薄膜磁性体のものに比べてまだまだ大きいが, 現在の計算機パワーではこのあたりが限界である.

3 結果

3.1 比熱及び磁化

膜厚 L_z を大きくしていくと $L_z \sim 4$ で磁気的性質が大きく変わることが分かった. 図 1 に $L_z = 3$ と $L_z = 5$ の比熱の温度依存性を示す. 薄い膜 ($L_z = 3$)

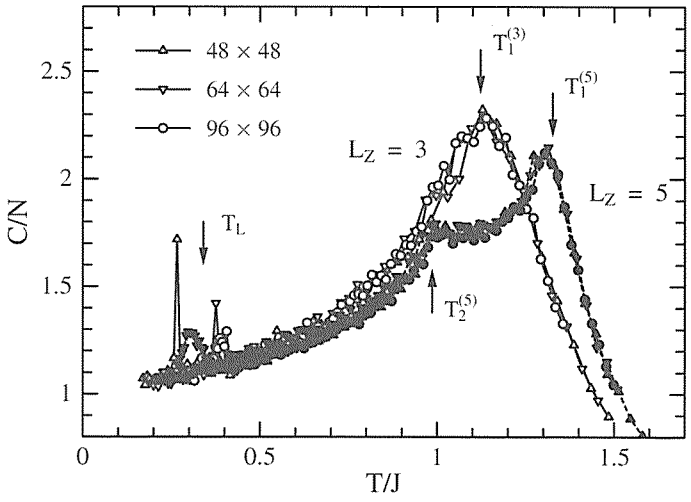


図 1: 比熱の温度依存性. 様々な平面格子サイズについて示してある.

では比熱は 高温側 ($T_1^{(3)} \sim 1.1J$) に大きなピークを持ち、低温側 ($T_L \sim 0.4J$)

にスパイク状の小さなピークを持つ. これらのピークの平面格子サイズ依存性はほとんどない. このことは, $T_1^{(3)}$ で磁気秩序が生じるが, この出現は連続的である (相転移ではない！) ことを示している. また, 低温側 T_L でも磁気構造の変化が起こることを示している. これに対して, 比較的厚い膜 ($L_z \geq 5$) では高温側 $T_1^{(5)}(\sim 1.3J)$, $T_2^{(5)}(\sim 1.0J)$ に２つのピークが見られ, 高温領域で磁気構造の逐次的変化が起こっていることを示している.

図２に磁化の温度変化を示す. 磁化の出現は厚い膜 ($L_z \geq 5$) でのみ認められ, その方向は膜面に平行である. 磁化の成長は高温側のピーク温度 $T_1^{(5)}$ で始まり, 低温側のピーク温度 $T_2^{(5)}$ で飽和に達する. このときの飽和値は通常の磁性体で期待される値の約 40% である. 一般に比熱のピークは磁気秩序の発生に対応するが, この場合は逆である. このことは $T_2^{(5)}$ で面直成分の秩序 (磁区) が発生していることを示唆している. 一方, 薄い膜 $L_z = 3$ では, $T_1^{(3)}$ 以下で面直成分の磁区が形成されていると考えられる.

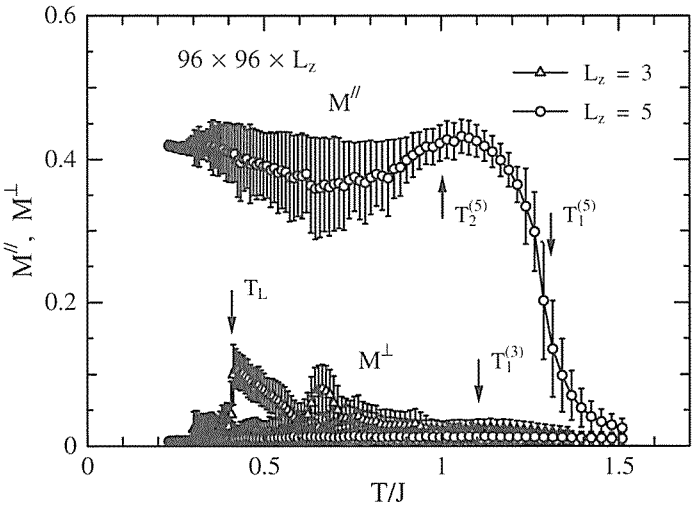


図 2: 磁化の膜面に平行な成分 $M^{\parallel}$ と膜面に垂直な成分 $M^{\perp}$ の温度依存性. 矢印は比熱にピークが見られる温度を示してある.

3.2 磁区 I：膜厚依存性

図３に様々な膜厚 L_z の薄膜で見られた典型的な磁区模様を示す. 図３ (a) - (e) は xy 面内のスピン構造を示し, 図３ (f) は xz 面内のそれを示している. 特徴は膜面に平行な成分は磁区を作らず膜面に垂直な成分が磁区を作ることで

ある. 膜厚が小さいときはバブル状の磁区が見られ, 大きいときはストライプ状の磁区になる. よく見てみると, スピンは磁区境界 (磁壁) で格子面に平行に向き, しかも磁壁に沿った方向を取っている. すなわち, 薄い膜でもブロッホ型の磁壁の出現が見られる. 面白いことに磁壁も静磁エネルギーを下げるように閉じたループを形作っている. このように薄い膜でも複雑なスピン構造を持つ磁壁が生じていることは興味深い.

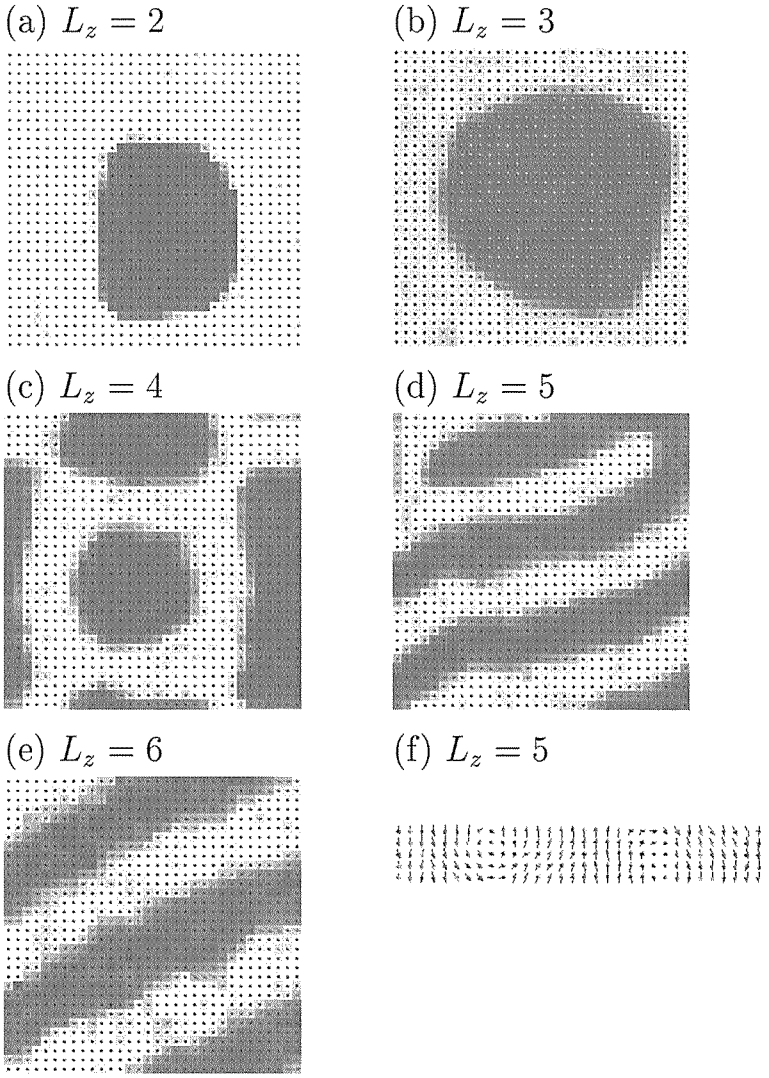


図 3: 様々な膜厚 L_z の積層薄膜の中心面におけるスピン構造のスナップショット, 温度は $T \sim 0.5J$. 各印は中心面 9 スピンの平均値を示している (ただし、(f) の z 方向は除く). また, 面直成分は明暗で, 面内成分は矢印で示している. (f) は $L_z = 5$ の場合の断面の一部である.

3.3 磁区 II：温度依存性

図4に薄い膜 ($L_z = 3$) の磁区構造の温度依存性を示す．比熱のピーク温度 $T_1^{(3)}$ 直下から面直成分 ($S_i^{\perp}$) の不規則な形をした磁区が出現していることが分かる．さらに温度を下げると磁区の形はバブル状に変わり，T_L 以下の低温ではストライプ状になる．低温における比熱のスパイク状のピークはこの磁区構造の変化に対応している．これに対して，面内成分 ($\boldsymbol{S}_i^{\parallel}$) の秩序化は低温でも認められない．

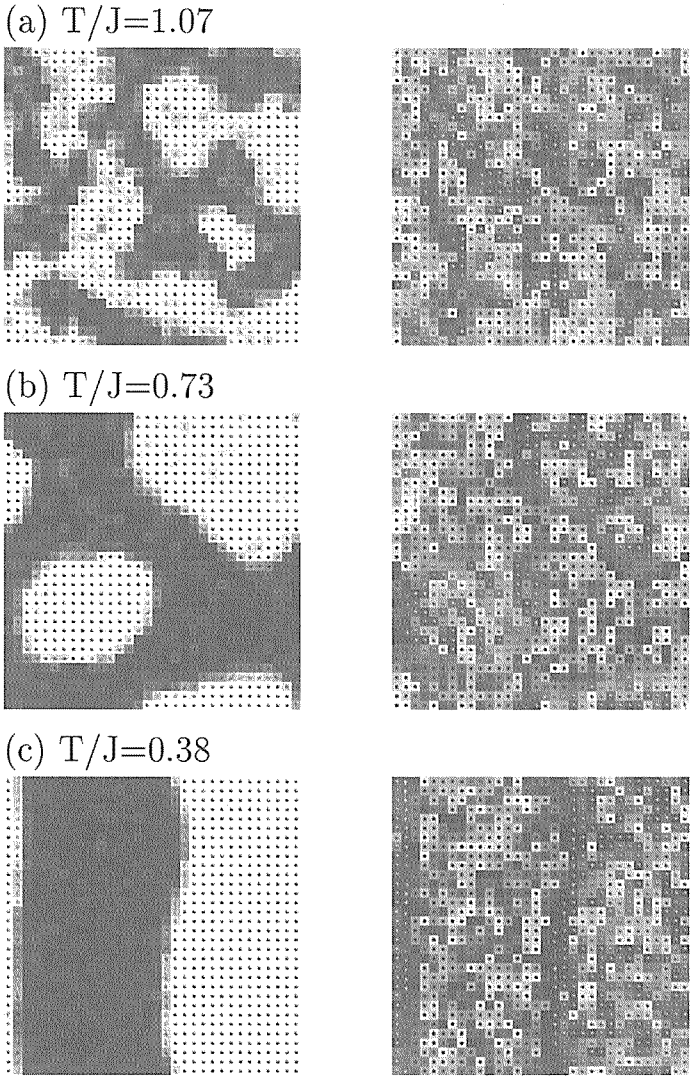


図 4: 薄い膜 ($L_z = 3$) の中心面のスピン構造のスナップショットの温度依存性．各矢印は中心面の９スピンの平均値．左図は面直成分を明暗で示したもの，右図は面内成分を色調と矢印で示したもの．比熱のピーク温度は $T_1^{(3)} \sim 1.1J$, $T_L \sim 0.4J$ である．

図５に比較的厚い膜 ($L_z = 5$) のスピン構造の温度依存性を示す．この場合は，比熱の第一ピーク温度 $T_1^{(5)}$ の下から面内強磁性秩序が始まっている．これに対して，面直成分の秩序は比熱の第二ピーク温度 $T_2^{(5)}$ 以下で起こっている．磁区の形状はストライプ型で温度が下がると共に明瞭になり，かつ幅が広がってくる．またストライプの方向は面内磁化 $M^{\parallel}$ の方向と同じである．興味深い点は面内強磁性と面直磁区が発生する温度が異なることであるが，一度発生した面内磁化 $M^{\parallel}$ は低温でも消失しない．すなわち，スピン再配列転移は再現されていない．

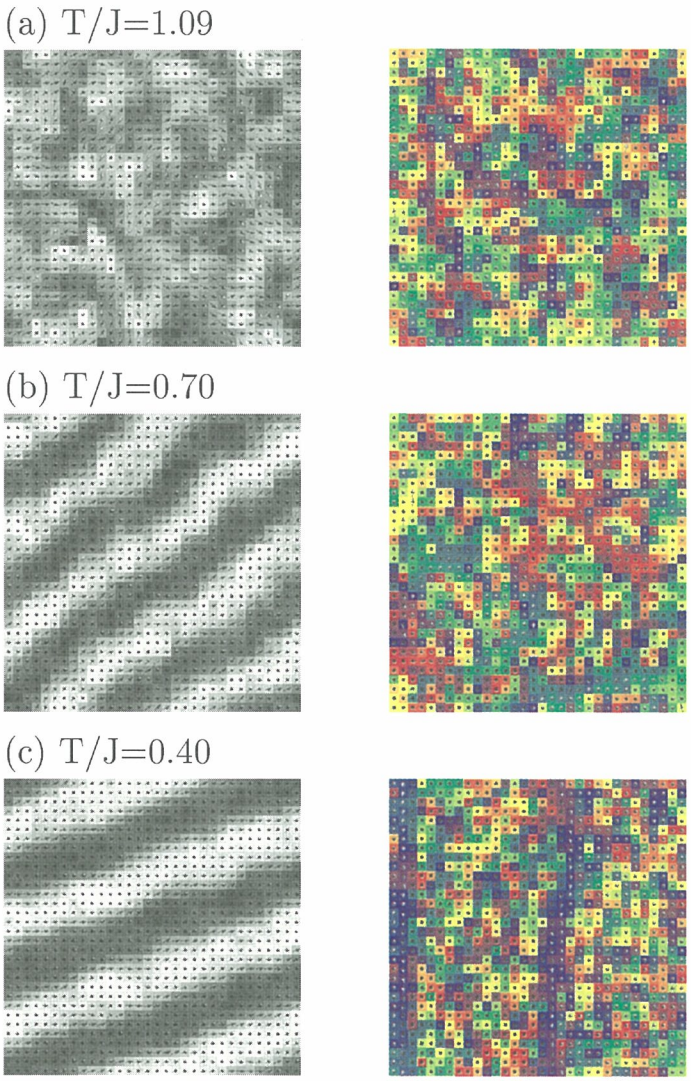


図 5: 比較的厚い膜 ($L_z = 5$) の内部のスピン構造のスナップショットの温度依存性．各印は中心面の９スピンの平均値．左図は面直成分を明暗で示したもの，右図は面内成分を色調と矢印で示したもの．比熱のピーク温度は $T_1^{(5)} \sim 1.3J$, $T_2^{(5)} \sim 1.0J$ である．

4 まとめ

本小稿では, 超薄膜磁性体の磁区構造研究の一端を紹介してきた. 膜厚や温度によって変わる超薄膜の磁区構造の理論的研究はミクロなモデルを基にしたコンピュータシミュレーションによって初めて可能になったものである. また, 超薄膜では表面乱れも無視できない. ミクロなモデルのシミュレーションにより, スピン再配列転移がこの表面乱れに起因することも明らかになりつつある. 今後ミクロなモデルのシミュレーション需要は益々高まっていくであろう. しかし, 今のところ我々の扱える格子面のサイズは 100×100 (一辺数 $10nm$) 程度であり, 実際の微小磁性体 (一辺 $1\mu m$ 程度) に比べて２桁ほど小さい. この問題を克服するにはアルゴリズムの更なる改良と計算機パワーの飛躍が必要である. 前者は我々研究者一人一人の努力に負うものであるが, 後者はコンピュータ産業, 計算機センター等の研究支援機関の努力に負うものであり更なるご尽力を期待したい. 平成１４年暮れより東北大学情報シナジーセンターのスーパーコンピュータシステムが更新されてＳＸ－７になり, 性能が８倍になる予定と聞いている (既に稼働中のことと思う). これによって、更なる大規模シミュレーションに挑戦できることを楽しみにして筆を置きたい.

この超薄膜磁性体シミュレーションソフトウエアの開発には, 情報シナジーセンターの岡部公起氏に多大の援助を戴いた. ここに深く感謝致します.

参考文献

[1] R. Allenspach and A. Bischof: Phys. Rev. Lett. **69** (1992) 3385.

[2] M. Farle et al. : Phys. Rev. B **56** (1997) 5100.

[3] J. Thomassen et al.: Phys. Rev. Lett.**69** (1992) 3831.

[4] 例えば, R. Allenspash: J. Magn. Magn. Mater. **129** (1994) 160.

[5] 例えば, J. H. J. van Opheusden and E. M. C. M. Reuvekamp: J. Magn. Magn. Mater. **88** (1990) 247.

[6] 例えば, A. B. MacIsaac et al.: Phys. Rev. Lett. **80** (1998) 616.

[7] 松原史卓, 鈴木伸夫, 佐々木淳哉：SENAC Vol.32 No.1 (1999.1).

平成１５年５月発行

編集・発行　東北大学情報シナジーセンター
仙台市青葉区荒巻字青葉
〒980－8578