高速化推進研究活動報告

第 4 号

平成20年11月

国立大学法人東北大学
サイバーサイエンスセンター

高速化推進研究活動報告　第4号

目　次

1　高速化推進研究活動報告第4号の刊行にあたって …… 1
2　高速化推進研究活動報告
　2.1　はじめに …… 3
　2.2　東北大学サイバーサイエンスセンターにおける大規模科学計算システムの変遷 3
　2.3　高速化推進研究体制および共同研究の取り組み …… 8
3　高速化推進研究活動の成果
　3.1　高速化推進研究活動（2005年度～2007年度） …… 11
　3.2　スーパーコンピュータSX-7のベクトル化・並列化の状況 …… 14
　3.3　今後の取り組み …… 15
4　SX-9高速化技法
　4.1　SX-9の特長 …… 16
　4.2　ベクトル処理による高速化 …… 18
　4.3　高速化技法 …… 21
　4.4　並列処理による高速化 …… 27
　4.5　高速化事例 …… 30
　　4.5.1　平面法によるベクトル化の事例 …… 30
　　4.5.2　MPI化の事例 …… 33

付録
参考資料 …… 40
研究論文 …… 64
論文リスト …… 69

高速化推進研究活動参加者

東北大学サイバーサイエンスセンター

スーパーコンピューティング研究部

小林広明、後藤英昭、滝沢寛之、江川隆輔、岡部公起

情報部情報基盤課

伊藤英一、大泉健治、小野敏

日本電気株式会社

HPC 販売推進本部

小久保達信、十川直幸、久保克維、緒方隆盛

第一コンピュータソフトウェア事業部

橋本ユキ子、横谷雄司、工藤淑裕、田村正典、早坂武

NEC コンピュータテクノ

磯部洋子

NEC システムテクノロジー

野口孝明、曽我隆、塩田和永、伊藤学、山口健太、山本秀喜

第一官公システム事業部

撫佐昭裕

NEC ソフトウェア東北

神山典、佐藤佳彦、金野浩伸、岡崎昌夫、山形正明、吉田智、菅雄一郎、下村陽一

1 高速化推進研究活動報告の刊行にあたって

東北大学理事(教育・情報システム担当)
根元 義章

21 世紀を迎え、情報処理技術、情報通信技術、応用技術の量的な面及び質的な面での格段の進歩により、高度な情報ネットワークシステムが実現し、これらのシステムを活用することにより、社会活動に大きな変化を与えてきたことを身近に実感できる。当然のこととして、大学における全ての業務においても、高度な情報システムの構築とその高度な利用を実現することが必須となっている。大学の情報システムは多種多様であるが、大規模科学計算の実施を可能とするスーパーコンピュータは重要な設備である。大学に設置されるスーパーコンピュータは研究目的に利用されるものであり、最先端かつ世界最大級の能力を持つ必要がある。最高級のスーパーコンピュータは極めて高価であり、その利用にあたっては研究者の共同利用の形態が望ましい。東北大学サイバーサイエンスセンターは、その前身の大型計算機センターが 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て以来、約 40 年が経過している。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、計算科学・計算機科学に関する共同利用・共同研究を通して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」ということを合言葉に、その実現に努力してきている。その原点を、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た、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の開発に見ることができる。また、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。当時の多くの関係各位のご努力に敬意を示したい。

その後、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。特に、1997 年 9 月より、利用者、本センター、日本電気㈱(以下、NEC)の 3 者により高速化推進のための研究会を立ち上げ、高性能計算に関する共同研究を進め、その成果を「高速化推進研究活動報告」にまとめている。高速化を実現するには、導入されたシステムの能力を極限まで利用することが必要である。そのためには、利用者である研究者のプログラムを、極限までチューンナップすることが必要であり、このためには利用者、センターの職員、そしてメーカの連携が必要であり、研究会がその役割を果たしてきている。

これまで、3 巻にわたり、その活動成果を「高速化推進研究活動報告」として発行している。得られた成果は、関連分野で広く活用されていると聞く。

今回発行する第 4 号は、2005 年度から現在までに得られた成果が収録されている。多くの研究者の方々に参考になるものと考える。なお、本共同研究では、本センターの技術系の職員を中心とするスタッフも中心的役割を果たしていることを特に記しておきたい。

さて、本センターが出版している広報誌の名前にもなっている「SENAC-1 (Sendai Automatic Computer 1)」が誕生して本年 11 月で 50 年になる。SENAC-1 は、本センターの前身の大型計算機

センターの初代センター長である大泉充郎先生が、諸外国に後れを取っていた我が国のコンピュータの研究開発の状況を打破し、国産コンピュータ技術の確立を目的として本格的なコンピュータの設計・開発をNECと共同で行ったものである。その成果はNECが出荷した第1号のコンピュータNEAC-1102に反映され、NECにおけるコンピュータ開発の原点としても位置付けられる。併せて、SENAC-1の開発を通して、数多くの研究者・技術者が育成されるなど、学界・産業界へ多大な貢献をもたらした。産学連携共同研究の成功例の1つであるSENAC-1の50歳の誕生日をお祝いすると共に、本センターには、SENAC-1 開発・センター設立の精神を引き継ぎ、利用者にとって真に役に立つ学術情報基盤の整備・運用・研究開発に熱意を持って引き続き取り組むことを期待している。

2　高速化推進研究活動報告

サイバーサイエンスセンター　センター長
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、コンピュータ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。

東北大学サイバーサイエンスセンター（以下、本センター）は、その前身の一つである大型計算機センターの設立（1969 年）以来、研究室のレベルをはるかに超える最高性能の計算機システムを設置して、最先端の学術研究を強力に支援・推進してきた。加えて、利用者にとって使い勝手の良いシステムの構築、他では実行できない大規模プログラムの実行環境の整備を行うと共に、本センター教員・技術職員、利用者、日本電気㈱（以下、NEC）が一体となって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・開発に取り組み、計算科学・計算機科学の発展に貢献してきた。本章では、本センターの特徴的な取り組みである高速化推進研究活動について説明する。

2.2　東北大学サイバーサイエンスセンターにおける大規模科学計算システムの変遷

本センターは 1986 年の SX-1（NEC 製、0.57Gflop/s）以来、常に最先端のベクトル型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。表 2.1 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 による小規模の 3 次元モデル、SX-7 による中規模の 3 次元モデルへと規模を拡大していき、2008 年 3 月に導入した最新のスーパーコンピュータ SX-9（NEC 製、26.2Tflop/s）は、単一 CPU 性能で 100Gflop/s を超えるベクトルプロセッサから構成され、単体 SMP ノードで 1.6Tflop/s、1TB メモリを実現するなど、従来シミュレーションモデルのさらなる大規模化・高精度化に加え、マルチスケール・マルチフィジックスシミュレーションなどシミュレーションの質的変化をもたらす新たな計算モデルへの展開も期待できるシステムとなっている。

表 2.1 本センターのベクトル型スーパーコンピュータの変遷

導入年	システム名	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
2008 年	SX-9	26214	16384GB	大規模 3 次元モデル マルチスケールマルチフィジックス

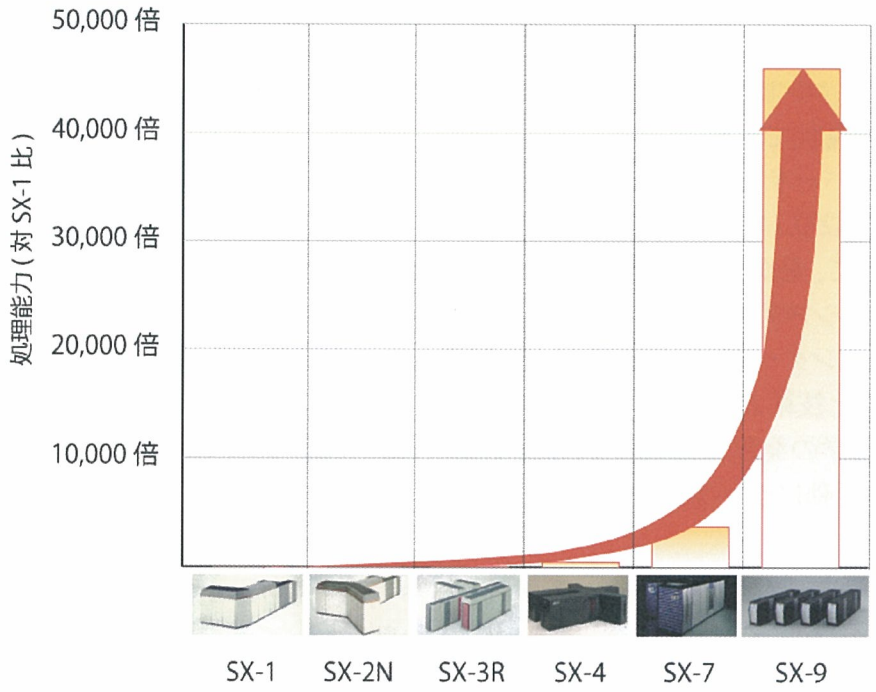


図 2.1 本センターで導入したベクトルスーパーコンピュータの性能向上

SX-9 システムは、102.4GFLOPS のベクトル演算プロセッサ (CPU) を 256CPU 備え、16 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、従来の SX-7 のシステム全体に相当する、1 ノード (16CPU) の演算処理能力 (1.6TFLOPS) と 1TB のメモリをフルに利用することができる。さらに、SX-9 のノードは 256GB/s (双方向) という高速な転送能力を持つ専用のネットワークシステム IXS (Internode Xbar Switch) で相互接続されているため、MPI を使用した並列プログラム対してもマルチノード (通常 4 ノード) による効率よい実行環境を提供している。表 2.2 に SX-7 と SX-9 の CPU、ノード、システム、それぞれのレベルでの性能比較を示す。SX-9 システムは、SX-7 システムに比べて、ピーク性能値において、CPU レベルで 11.6 倍、ノードレベルで 5.8 倍、システムレベルで 12.5 倍の性能向上を達成すると共に、CPU あたり 256GB/s のスカラ型スーパーコンピュータでは得ることのできない高いメモリバンド幅を有している。

図 2.2 は、同じ総ピーク性能 1Tflop/s を有する、1) 100Gflop/s の CPU10 個のシステムと、2) 10Gflop/s の CPU100 個のシステムにおいて、並列処理効率とそのとき得られる実効性能の関係をアムダールの法則に基づき示したものである。図からわかるように、高い単一 CPU 性能を有するシステムでは、比較的低い並列処理効率においても高い実効性能を得ることができ、このことは高い CPU 性能を持つシステムでは、比較的少ない労力でも高い実効性能が得られる可能性を示唆しているといえる。

表 2.2 SX-7 と SX-9 の性能比較

性能項目		SX-7 (2003 年)	SX-9 (2008 年)	SX-7 比
CPU あたり	クロック周波数	1.1GHz	3.2GHz	2.9 倍
	ベクトル演算性能	8.83Gflops	102.4Gflops	11.6 倍
	メモリハンド幅	35.32GB/s	256GB/s	7.3 倍
ノードあたり	ベクトル演算性能	282Gflops	1.6Tflops	5.8 倍
	メモリ容量	256GB	1TB	4 倍
	メモリバンド幅	1.13TB/s	4TB/s	3.5 倍
	メモリバンク数	16384	32768	2 倍
	ノード間接続速度	32GB/s *	256GB/s	8 倍
システム全体	ベクトル演算性能	2.1Tflops	26.2Tflops	12.5 倍
	メモリ容量	2TB	16TB	8 倍

*SX-7C (SX-8)

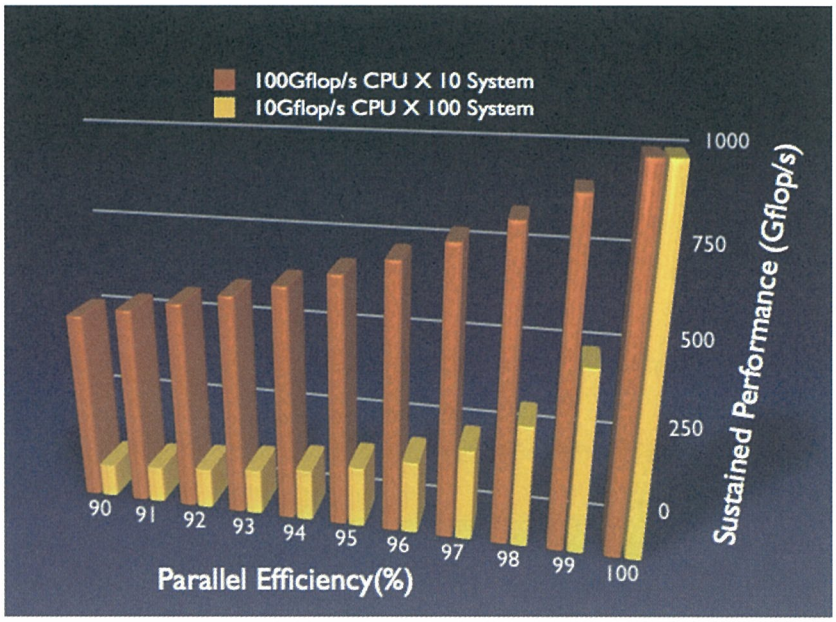


図 2.2 単一 CPU 性能の違いによる並列処理効率と実効性能との関係

図 2.3 は、本センターの利用者がそれぞれの研究分野で実際に研究開発している実アプリケーションコードを用いて、ベクトル型スーパーコンピュータ SX-9、SX-8R、SX-8 (SX-7C)、SX-7 とスカラ型並列コンピュータ TX-7/i9610 の単一 CPU の実効性能を比較したものである。なお、図

中の倍率はSX-7に対するSX-9の性能向上率である。図からわかるように、SX-9はTX7/i7610のItaniumIIに比べて圧倒的な実効性能を達成すると共に、SX-7と比べても実アプリケーションの実行において4.4倍から8倍の性能向上を達成している。

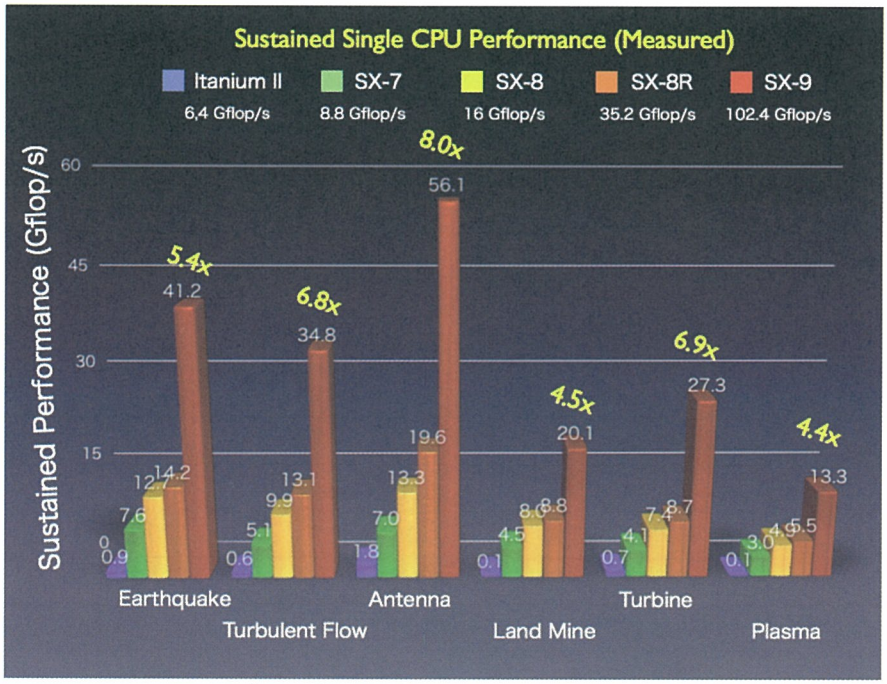


図 2.3 実アプリケーションによる性能評価結果

さらに、ベクトル処理に不向きな計算の高速化やGaussianなどのアプリケーションを提供する計算環境として、1998年に並列演算サーバ(Exemplar)、2002年に並列コンピュータ(TX7/AzusA)、2006年に(TX-7/i9610)をそれぞれ導入してきた(表2.3)。

2006年4月から運用している並列コンピュータTX7/i9610は、3ノード(192プロセッサ)から構成され、1.23Tflop/sの性能を有する。TX7/i9610の1ノードは、32CPU(コア数64)のデュアルコアのItaniumIIプロセッサから構成され、409.6Gflop/s、512GBの計算環境を提供する。ノード間は2GB/sのInfini-Bandで相互接続される。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スレッド並列処理とプロセッサ内部での命令レベル並列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 2.3 本センターのスカラ型並列コンピュータの変遷

導入年	システム名	性能(GFLOPS)	記憶容量
1998年	Exemplar/X	34	12GB
2002年	TX7/AzusA	358	176GB
2006年	TX7/i9610	1228	1536GB

図2.4に本報告書で取り上げる本センターの大規模科学計算システムの構成を示す。図2.5は、主にベクトル型スーパーコンピュータで2007年に実行されたアプリケーションの計算時間に基づく研究分野の分類である。図に示されるように、本センターのベクトル型スーパーコンピュ

ータは、次世代航空機設計、高性能アンテナ、次世代磁気記録デバイス、地震解析などその社会的影響力の高い成果が期待される分野で重要な貢献を行っている。

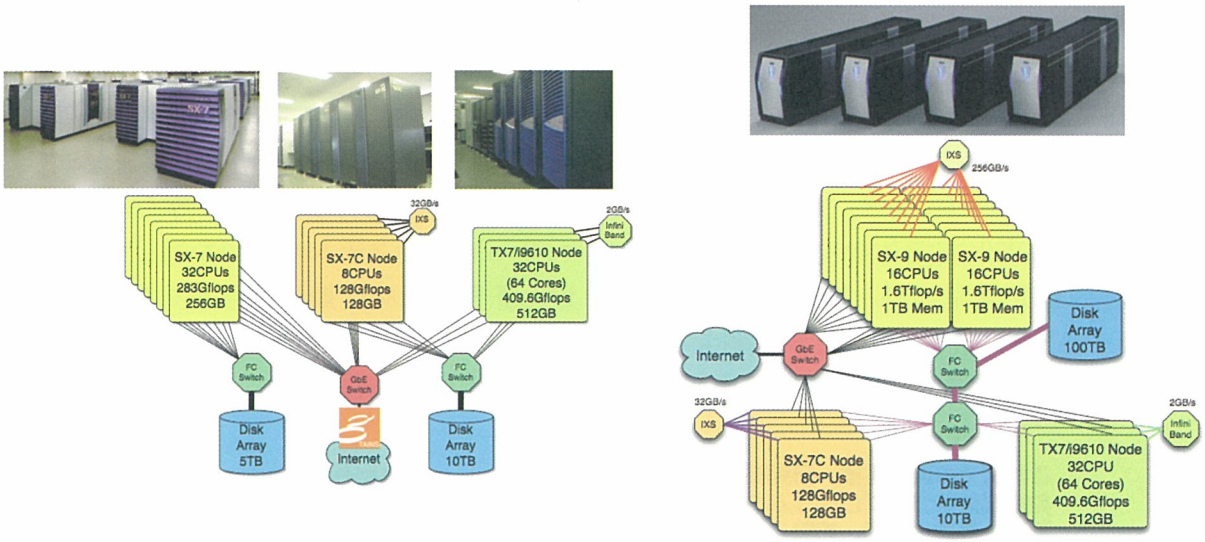


図 2.4 大規模科学計算システム
（左：～2008 年度　右：2008 年度～）

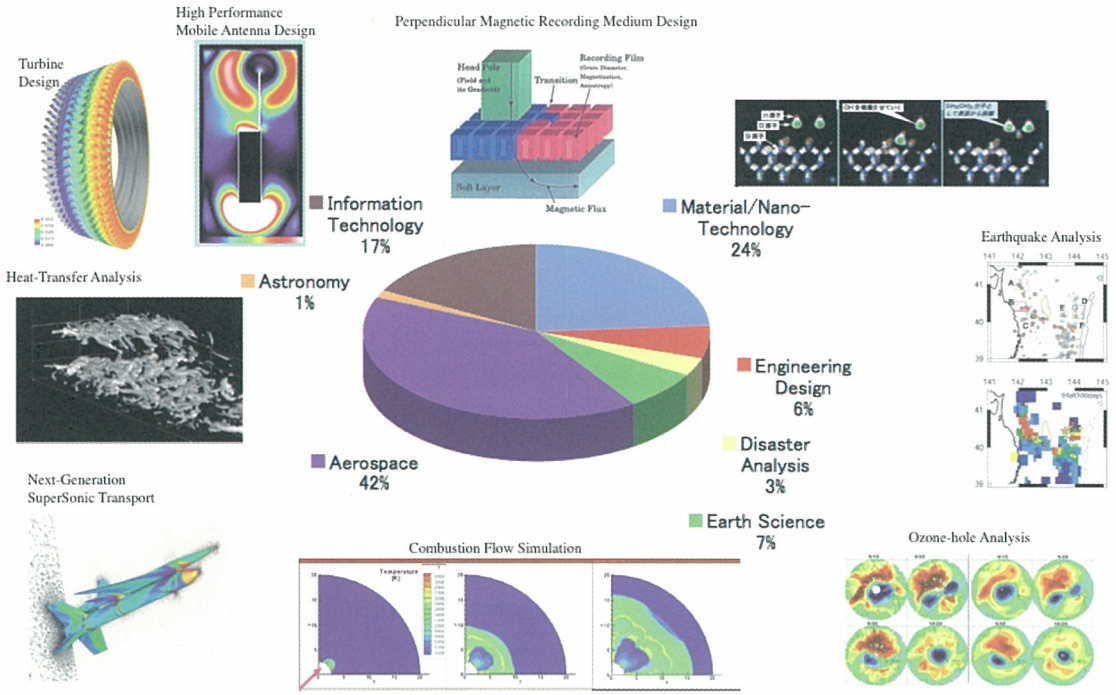


図 2.5 ベクトル型スーパーコンピュータの利用分野

2.3 高速化推進研究体制および共同研究の取り組み

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。本センターでは、「最先端、かつ世界最大級のコンピュータシステムの導入、利用環境の整備・運用・研究・開発」という本センターの使命を果たすために、1997 年に 7 センターの中でいち早く図 2.6 に示すように本センターの教員・技術職員、利用者、メーカの技術者が一体となって、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の研究開発に取り組む高速化支援体制を整備し、スーパーコンピュータシステムの運用を通した臨床学的・実証的研究開発活動を推進してきた。これにより、研究成果を次期情報基盤の整備・運用へ還元すると共に計算科学・計算機科学分野で活躍できる人材育成を行い、日本の計算科学コミュニティへの支援を通して先端学術研究に幅広く貢献している(図 2.7)。

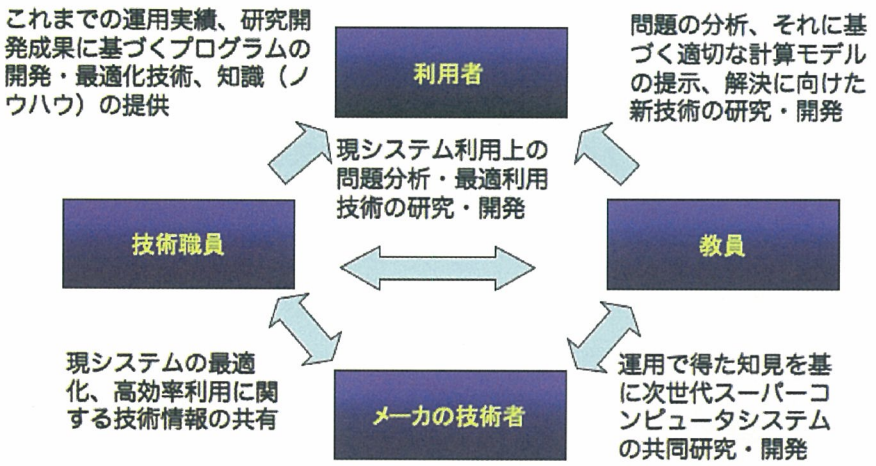


図 2.6 高速化支援体制

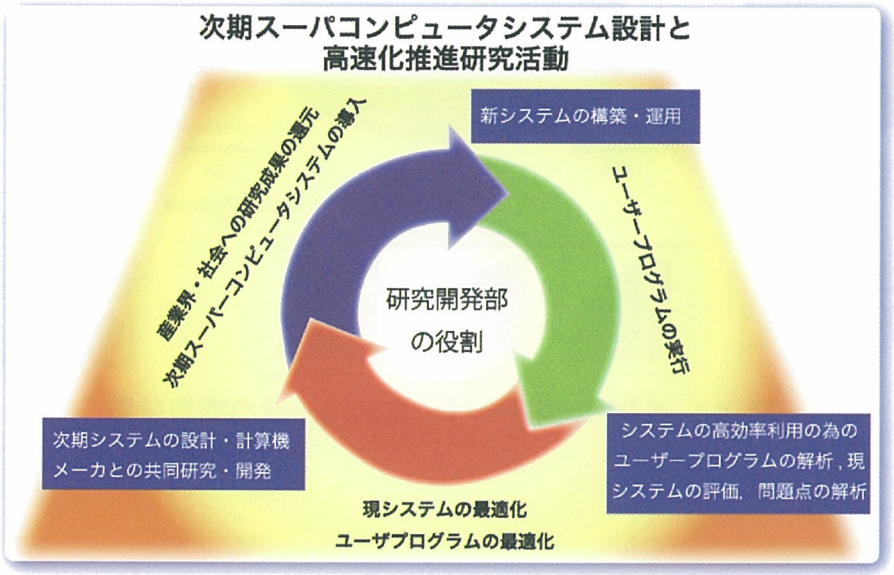


図 2.7 運用を通じた臨床学的研究開発サイクル

本高速化支援体制の下、平成11年度からは、毎年10件程度、大規模計算利用者との共同研究を実施し、大学の多様な研究分野で活用される様々なシミュレーション計算モデルの大規模化、高精度化、高効率化に関する研究開発に取り組み、その成果を、高速化推進研究活動報告として、2001年、2003年、2005年に出版し、広く日本の大規模科学計算コミュニティに還元してきた。

また、高速化支援を通じて、次世代スーパーコンピュータに関する研究開発をNECと共同で行っている。最近の成果としては、次世代ベクトルスーパーコンピュータで問題になるオフチップメモリバンド幅制限による実効性能の低下という問題に対して、ベクトルプロセッサ向けオンチップキャッシュ(図 2.8)の提案が挙げられる。特に、図 2.9 に示すように、本センター利用者の開発した実プログラムを用いた性能評価により、提案キャッシュがオフチップメモリバンド幅に制限があるシステムにおいても高い実効性能を維持できることを明らかにしたことは、実用性の高い研究成果として国内外で高く評価されている。さらに、本キャッシュアーキテクチャの特性を生かし、マルチベクトルコア向けに展開する研究開発を進めている。共同研究の成果は、学術論文誌やSCなどスーパーコンピュータに関する著名な国際会議の論文として毎年発表しており(図 2.10)、理研次世代スーパーコンピューティングシンポジウム 2008 では特別賞を受賞した(http://www.nsc.riken.jp/symposium2008.html)。

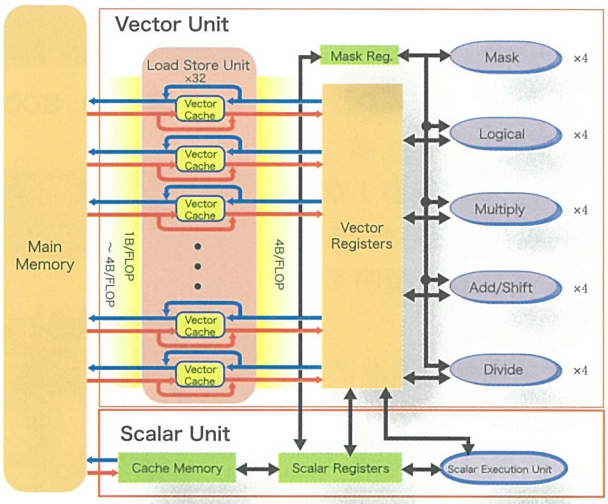


図 2.8 ベクトルキャッシュの構造

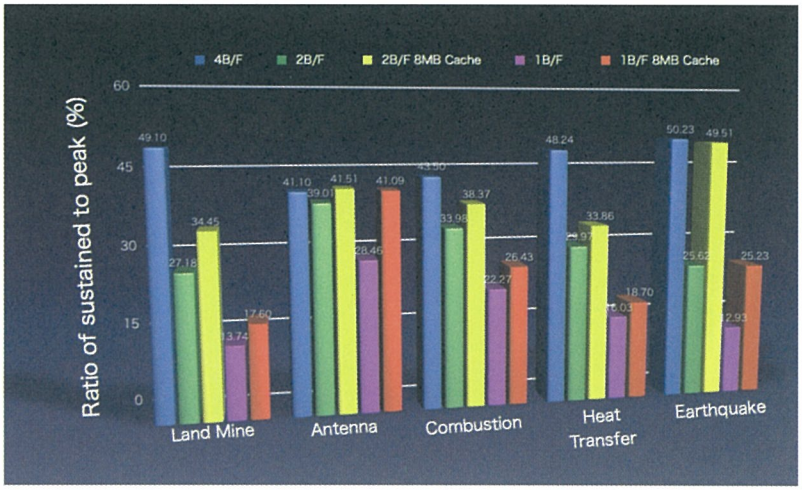
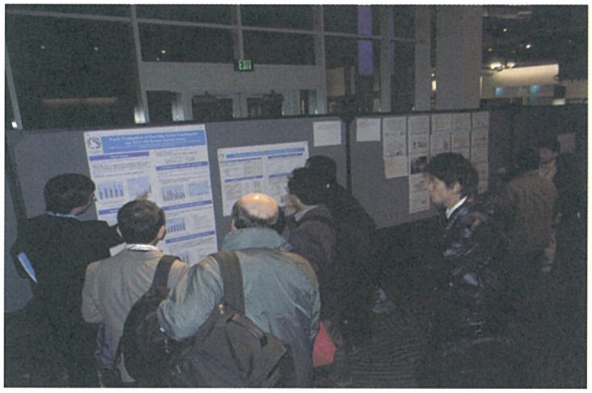


図 2.9 ベクトルキャッシュの効果

さらに、2006 年からはドイツシュットガルト大学高性能計算センターと共同で高性能計算に関する国際ワークショップ Teraflop workshop を企画し、本センターや利用者の研究成果発表に加えて、国内外の著名な研究者技術者を招聘し、毎年 11 月に東北大学で開催し、成果の国際発信に努めている（図 2.11）。

これら国内外で高く評価されている活動・成果はいずれも、本センターの教員・技術職員・NEC 技術者・そして利用者が密に連携した高速化支援体制・共同研究体制がベースになっている。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、本センター側スタッフと NEC 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 3 章以降では、2005 年 4 月から現在に至るまでの高速化推進研究活動について詳細に説明する。

最後に、本高速化推進研究活動の推進には、利用者の協力を欠かすことができない。本報告書に掲載されている成果は、東北大学の長谷川昭名誉教授、澤谷邦男教授、佐藤源之教授、山本悟教授、中橋和博教授、高橋俊様、飯島雅英准教授、東京理科大学の塚原隆裕博士、海洋研究開発機構の有吉慶介博士の多大なる協力により得られたものである。改めて感謝の意を表したい。



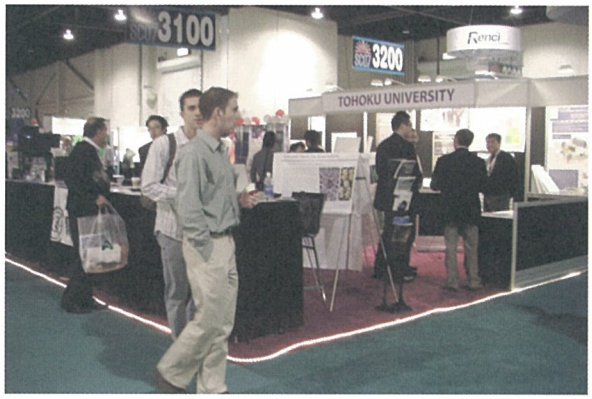


図 2.10　SC07 での研究発表

図 2.11　テラフロップワークショップ

3 高速化推進研究活動の成果

情報部情報基盤課
伊藤英一、大泉健治、小野敏

スーパーコンピュータシステム SX-7 は、2003 年 1 月にサービスを開始し 2008 年 2 月に終了した。この間に利用者プログラムのシミュレーションモデルの規模はより大きくなりジョブの大規模化・長時間化の傾向は一層強くなった。

本センターでは、従来から研究室では実行不可能な大規模・長時間ジョブの実行を可能とするため、スーパーコンピュータでは、1 ノードの全 CPU を1つのジョブで占有する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ことから CPU 時間を制限しないような利用環境を整備してきた。

このような大規模・長時間ジョブにおいては、ベクトル化率と並列化率を極限まで高めておくことが必要である。そのためにはスーパーコンピュータのハードウェアやコンパイラについての深い知識が要求される。また、自動ベクトル化、自動並列化の機能強化には、利用者プログラムに積極的に係わっていく必要もある。そこで、本センターでは高速化を専門に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いる。

以下では、2005 年度から 2007 年度までの SX-7 システムでの高速化を行ったプログラムのうち主なものについて概略を述べる。

3.1 高速化推進研究活動(2005 年度～2007 年度)

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表 3.1.1 に 2005 年度(平成 17 年度)、表 3.1.2 に 2006 年度(平成 18 年度)、表 3.1.3 に 2007 年度(平成 18 年度)の高速化支援性能向上比と概略を報告する。ここで並列性能は原版(8CPU)に対しての性能向上比である。

表 3.1.1 2005 年度 (17 年度)の高速化支援性能向上比

プログラム番号	主な改善点	性能向上比	
		単体性能	並列性能(8 並列)
1	バンクコンフリクトの削減 並列版 ASL に置き換え	―	1.7 倍 (4 並列)
2	アンローリング ファイル出力時間の低減	1.0 倍	1.0 倍
3	並列化指示行による並列化促進	1.1 倍	1.3 倍
4	作業配列によるベクトル化の促進、ベクトル長の拡大	1.2 倍	1.4 倍
5	アンローリング 作業配列によるベクトル化の促進、ベクトル長の拡大	2.2 倍	―
6	インライン展開	―	8.3 倍

	ループ交換によるベクトル長の拡大		
7	アンローリング 作業配列使用による演算量の削減 ループ入れ換えによるベクトル長の拡大	1.2 倍	2.6 倍
8	res=whole の利用による並列化	2.4 倍	12.3 倍

表 3.1.2 2006 年度 (18 年度)の高速化支援性能向上比

プログラム番号	主な改善点	性能向上比	
		単体性能	並列性能 (8 並列)
1	バンクコンフリクトの削減 作業配列の使用による並列化	―	37 倍 (16 並列)
2	リストベクトルの削減 バンクコンフリクトの削減 ファイル出力時間の低減	1.7 倍	2.2 倍
3	Makefile の修正 環境変数の変更	―	12 倍 (16 並列)
4	各 CPU 負荷を均等にするための処理の分割 総和演算の並列化	―	1.6 倍 (4 並列)
5	配列次元の入れ替えによるバンクコンフリクトの削減 行列積ライブラリへの置換(複素数の実部・虚部の分離)	2.2	1.4
6	行列積ライブラリへの置換(複素数の実部・虚部の分離) 3 重ループの外側を並列化、内側を行列積になるよう改善	8.3	8.5
7	ループ交換による並列粒度の拡大 ベクトル阻害要因の write 文抑止	5.6	49.2
8	リストベクトルの抑止 ループの unroll	1.2	1.3
9	外側ループの outerunroll 行列積ライブラリへの置換(ループの分割)	―	1.1
10	ループの一重化 ループ交換	1.4	1.6
11	ループ交換 IF 文を含むループの最適化	1.6	2.9
12	バンクコンフリクトの削減 ループ交換	2.2	2.1
13	並列化オプション指定	―	3.6
14	ループ交換	2.3	1.4

15	ソース修正による並列化阻害要因の排除 一重ループの外側に並列化用のループを用意し、 並列化促進	—	1.7
16	ソース修正による並列化阻害要因の排除	—	2.0 (32 並列)

表 3.1.3 2007 年度 (19 年度)の高速化支援性能向上比

プログラム番号	主な改善点	性能向上比	
		単体性能	並列性能 (8 並列)
1	指示行による並列化促進 並列版 ASL の利用	—	5 倍
2	ループ交換によるベクトル長拡大 if 文による条件付き代入改善によるベクトル化 バンクコンフリクトの削減	160 倍	—
3	並列化制御オプションの追加	—	1.1 倍 (32 並列)
4	並列版 ASL の利用	—	1.5 倍 (4 並列)
5	ループ分割によるベクトル化、並列化促進 ループ外へ代入式を移動し、演算数を削減	1.9	—
6	ループ融合によるベクトルロードの削減と並列粒度の拡大	1.1	1.1
7	並列化指示行による並列化	—	1.1
8	並列化指示行による並列化	—	1.4
9	作業配列使用によるベクトル化	1.1	—
10	作業配列使用によるベクトル化	1.1	—

3.2 スーパーコンピュータ SX-7 のベクトル化・並列化の状況

SX-7 システムの 2005 年度から 2007 年度までの 3 年間の利用者ジョブのベクトル化・並列化の状況を表 3.2.1 に示す。表は SX-7 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、CPU 時間の割合を全 CPU 時間に対して千分率で表したものである。この表より、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。

表 3.2.1 ベクトル化率と並列化率の状況

ベクトル化率 \ 並列化率	0%	10%	20%	30%	40%	50%	60%	70%	80%	90%
90%	15	16	20	27	65	28	24	67	147	502
80%	1	3	3	1	2	2	1	2	13	23
70%	3	1	0	0	0	0	0	0	0	8
60%	2	1	0	0	0	0	0	0	0	1
50%	1	0	0	0	0	0	0	0	0	2
40%	1	4	0	0	0	0	0	0	0	1
30%	0	0	0	0	0	0	0	0	0	0
20%	0	2	0	0	0	0	0	0	0	20
10%	1	0	0	0	0	0	0	0	0	10
0%	2	1	0	0	0	0	0	0	0	2

備考：マトリックスの値は、ジョブ CPU 時間の千分率（全 CPU 時間比）

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 50%
・ベクトル化率が 90%以上のジョブ	約 91%
・並列化率が 80%以上のジョブ	約 73%

このように、ベクトル化率、並列化率がともに高く SX-7 の性能が十分に活かされている状況は、高速化推進研究活動の成果であると考えられる。

表 3.2.3 は 2005 年度から 2007 年度までの 3 年間について、ジョブの CPU 時間の分布を示したものである。

表 3.2.2 CPU 時間使用分布

時間	2005 年度	2006 年度	2007 年度
0 ～ 99	13%	14%	8%
100 ～ 999	46%	38%	28%
1000 ～1999	18%	13%	14%
2000 ～	23%	35%	50%

3.3 今後の取り組み

本センターのスーパーコンピュータシステムは2008年3月にスーパーコンピュータSX-9に更新した。それまでのSX-7に比べて演算性能が12.8倍、メモリ容量が8.8倍と拡大した。運用開始して半年の時点ですでにジョブの大規模化、長時間化が進んでいる。SX-9においてもSX-7と同様 3.2 節で述べたようなベクトル化率と並列化率はともに高いところに集中すると推測される。このため SX-9 システムにおいてはシステムの更新に伴い、新たなチューニングに力を入れていく必要があると考えている。

4 SX-9 高速化技法

スーパーコンピューティング研究部
小林広明、江川隆輔、岡部公起
情報部情報基盤課
伊藤英一、大泉健治、小野敏
日本電気株式会社
撫佐昭裕
NEC システムテクノロジー株式会社
曽我隆、塩田和永、山口健太
NEC ソフトウェア東北株式会社
神山典、金野浩伸、下村陽一

4.1 SX-9 の特長

SX-9 は、ノードあたり 16 個の CPU、1TB の主記憶装置を搭載し、1.6TFLOPS のベクトル演算性能を有したベクトル型スーパーコンピュータである。サイバーサイエンスセンターでは、2008 年 3 月に SX-9 16 ノードのシステムを導入し、高速クロスバースイッチ IXS(Internode Xbar Switch)で全ノードを接続することによって、最大256CPU による大規模シミュレーションを行うことが可能となった。SX-9 の主な特長は以下のとおりである。また SX-9 のノード主要諸元と CPU の内部構造は表 4.1.1、図 4.1.1 の通りである。

① ベクトルユニットは 8 セットのベクトル演算パイプライン(マスク演算、論理演算、乗算×2、加算×2、除算／Sqrt)を有し、CPU あたり 102.4GFLOPS の単一コア世界一の演算性能を実現している。
② 共有メモリ方式のアーキテクチャにより、1TB の大規模な主記憶を 16 個の CPU で共有することができる。
③ 主記憶装置と CPU 間のデータ転送は、CPU あたり 256GB/S、ノードあたり 4TB/S を実現している。
④ オンチップメモリ ADB (Assignable Data Buffer)を有している。
⑤ プログラムを高速実行するため FORTRAN と C コンパイラは、自動ベクトル化機能と自動並列化機能を有している。
⑥ ノードあたり 128GB/S の高速クロスバースイッチ IXS(Internode Xbar Switch)によりマルチノードシステムを構成している。

表 4.1.1　SX-9 のノード主要諸元

項目			SX-9
理論最大演算性能			1.6TFLOPS
CPU 数			16
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
		*128bitはスカラ命令のみサポート	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		命令:32KB
			オペランド:32KB
主記憶装置	容量		1TB
	最大データ転送能力		4TB/S

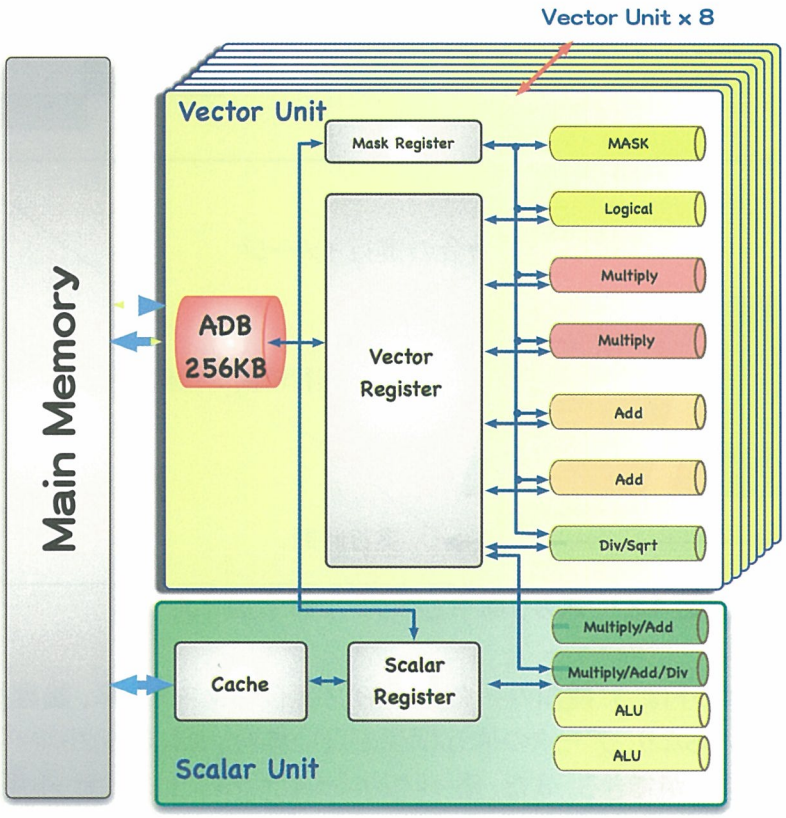


図 4.1.1　CPU の内部構造

4.2 ベクトル処理による高速化

SX-9 における高速演算機能としてベクトル処理がある。SX-9 では、FORTRAN、C コンパイラの自動ベクトル化機能によって、ユーザは意識することなくベクトル処理を利用することができる。

科学技術計算プログラムの多くは、特定の do ループ(for ループ)に実行の大部分が集中している。しかも、このループは配列データ(ベクトル)に同一演算を繰り返し実行させることが多い。この規則性に着目して高速化を図った方式がベクトル処理である。ベクトル処理は、do ループによる配列データへの繰り返し演算を、ベクトルユニットを用いて一括に実行するものである。一方、このベクトル処理に対してスカラ処理といわれる演算は、一組のデータを逐次的に実行していくものである。図 4.2.1 にスカラ命令とベクトル命令の処理のイメージ図を示す。

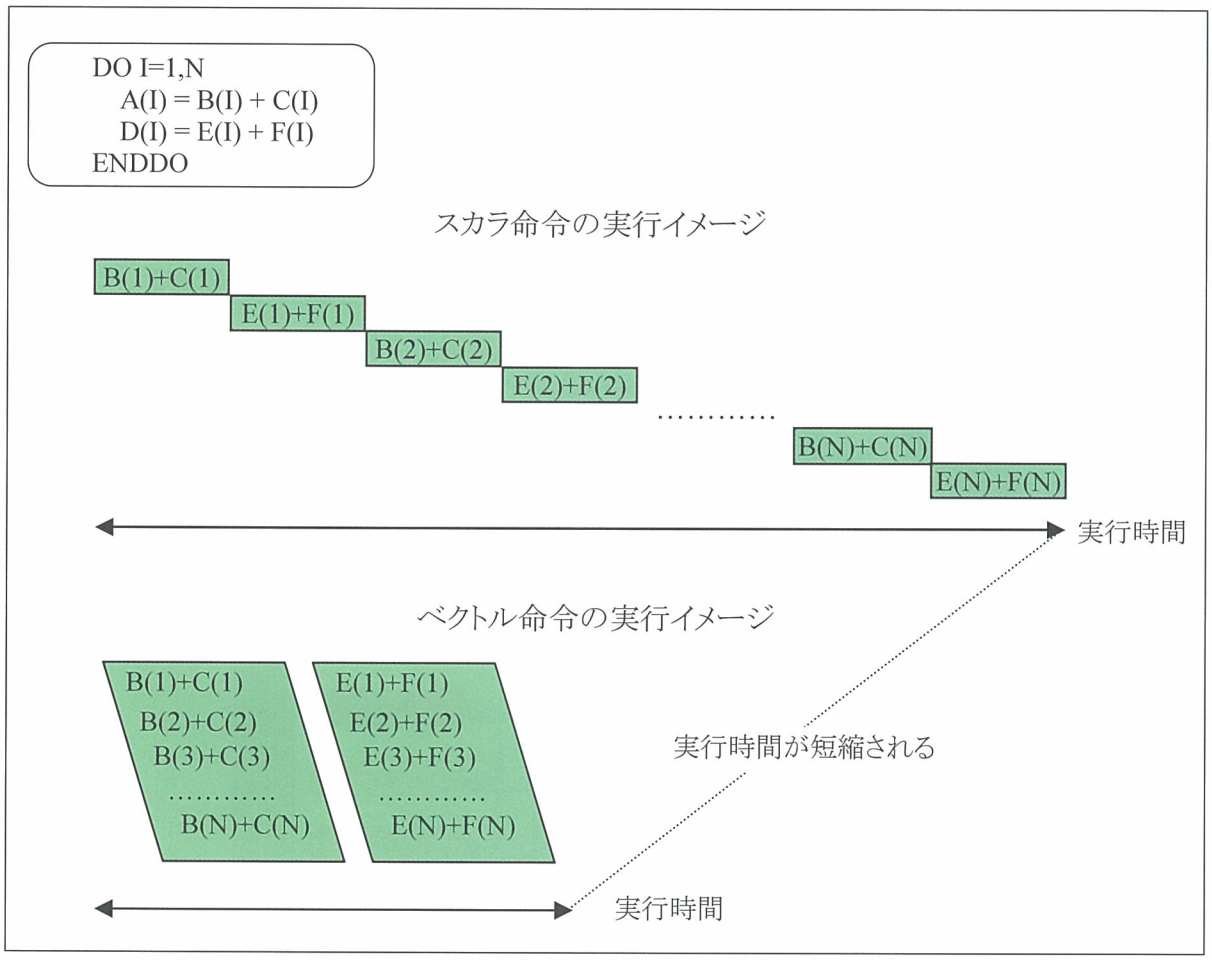


図 4.2.1 スカラ命令とベクトル命令のイメージ

SX-9 のベクトルユニットは 5 種類のベクトルパイプライン(マスク演算、論理演算、乗算×2、加算×2、除算／Sqrt)からなり、データの依存関係がない場合、これらベクトルパイプラインは 5 種類同時に実行することができる。また、各ベクトルパイプラインは 8 セット用意され、SIMD (Single Instruction Multiple Data)型の並列処理を行っている。ベクトルユニットのこれらの動作が SX-9 の高速演算を実現しているのである。

表 4.2.1 に SX-9 とサイバーサイエンスセンターの並列コンピュータ TX7/i9610 の主要な諸元の比較を示す。表中の B/flop はメモリバンド幅を演算性能で割った値であり、CPU への演算

データの供給割合を表している。SX-9 はスカラ型の並列コンピュータ TX7/i9610 と比較して、演算性能当たりのデータ供給量が大きいことがわかる。

表 4.2.1 SX-9 と TX7/i9610 の比較

システム名	CPU 数（コア数）	動作周波数（GHz）	CPU（コア）あたり			
			演算性能（Gflop/s）	メモリバンド幅（GB/s）	B/flop	キャッシュ
SX-9	16	3.2	102.4	256	2.5	ADB:256KB
TX7/i9610（Itanium Ⅱ）	32（64）	1.6	12.8（6.4）	8.5（4.25）	0.66	L1:16KB L2:256KB L3:12MB

図 4.2.2 に実アプリケーションによる SX-9 と TX7/i9610 の 1CPU（TX7/i9610 は 1 コア）の実効性能の比較を示す。グラフは、TX7/i9610 の性能値を 1 とした時の SX-9 の性能値を表している。

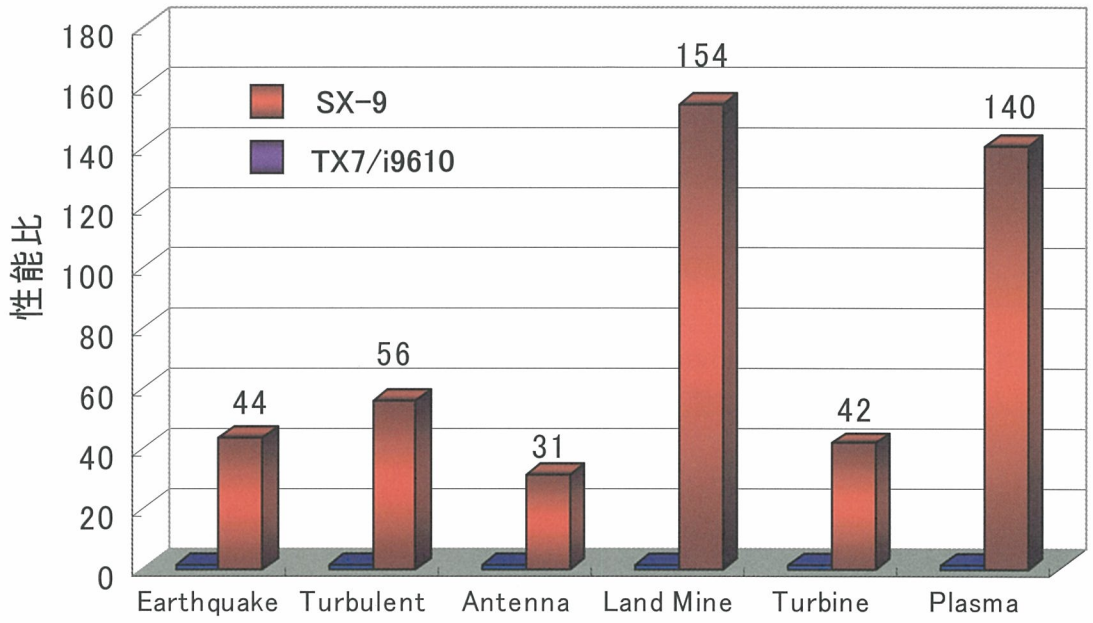


図 4.2.2 TX7/i9610 の性能を 1 としたときの SX-9 の性能倍率

メモリ負荷の小さい Antenna コードでは、SX-9、TX7/i9610 とも高い実効性能であるため、性能差は 31 倍であった。しかしメモリ負荷の高い Land Mine コードでは TX7/i9610 はキャッシュヒット率が 77%程度に留まり、実効性能も 130Mflop/s と低いため SX-9 との性能差が 154 倍と高くなっている。Plasma コードは間接参照（リストベクトル）を使用しているコードで、TX7/i9610 ではキャッシュを有効に使用できないため、SX-9 との性能差が 140 倍になっている。演算性能（ピーク性能）比が 16 倍に対して、実アプリケーションではいずれも SX-9 が大きく性能を上回っていることが明らかになった。これはメモリバンド幅の性能が、アプリケーションの実行に大きく影響するためである。

SX-9 では、組み込み関数の SQRT と MAX/MIN をハードウェアの命令で実行する。SQRT を使用する実アプリケーションの実行において、ハードウェアで行う場合と従来通りのソフトウェアで行う場合を比較した。図 4.2.3 に使用したソースのイメージを示す。このコードでは、SQRT が 136 億回実行される。

```
                    :
1957                A = A / B - C/DSQRT(B)
1958                B = B / B - C/DSQRT(B)
                    :
```

図 4.2.3 SQRT を使用したソースコードのイメージ

SQRT をソフトウェアで実行した場合の 13.4 秒に対して、SQRT をハードウェアで実行すると 1.81 秒となり、7.4 倍の向上になった。ただし、従来ソフトウェアで実行した場合と比較して、誤差が生じる場合がある。これは近似方法の違いによるものである。本コードを実行した時の SQRT の演算結果の差異を、図 4.2.4 に示す。

```
【SW Result】   0.57251621211382042900E-02
【HW Result】   0.57251621211380577030E-02
```

図 4.2.4 SQRT の実行結果差異の比較

コンパイルオプション-Wf,-pvctl novsqrt を指定することにより、SQRT は従来と同じソフトウェアでの実行になる。コンパイラの規定値では、SQRT はハードウェアによる実行になる。

4.3 高速化技法

(1)メモリアクセスの効率化

表 4.2.1 で示したように、SX-9 は高いメモリバンド幅を有しており、CPU に大量にデータを供給することができ、図 4.2.2 に示したようにスカラ型機に比較して、高い実効性能を実現することを可能にしていることがわかった。しかしながらメモリアクセスの方法を替えることにより、より高い性能を得る場合がある。図 4.3.1 に SX-9 の CPU と主記憶の構造を示す。

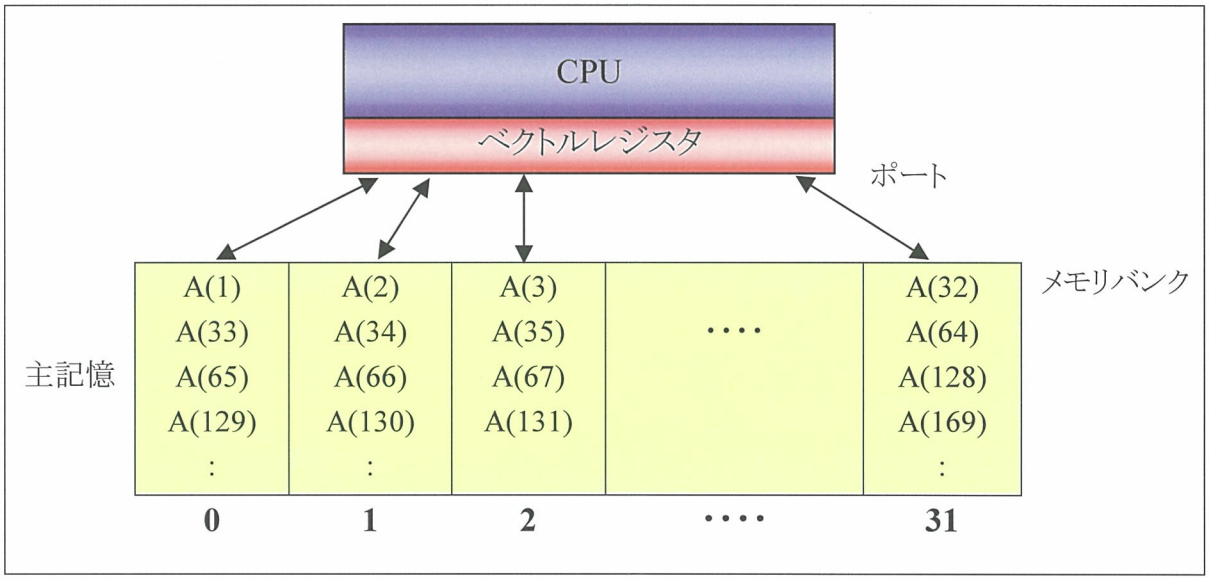


図 4.3.1 CPU と主記憶の構造

1TB の主記憶は、32 のメモリバンクグループに分割され、CPU からは各メモリバンクグループからデータが供給される。配列データのアドレスはメモリバンクグループ順に確保されるので、配列データが連続でアクセスされる場合は、すべてのポートからデータ供給の命令が発行され、効率よくデータが供給される。しかしながら飛びアクセスになる場合は、ポートに空きが生じる可能性があり、データの供給が効率的に行われないことになる。図 4.3.2 にループの構造とメモリアクセス順序の関係を示す。

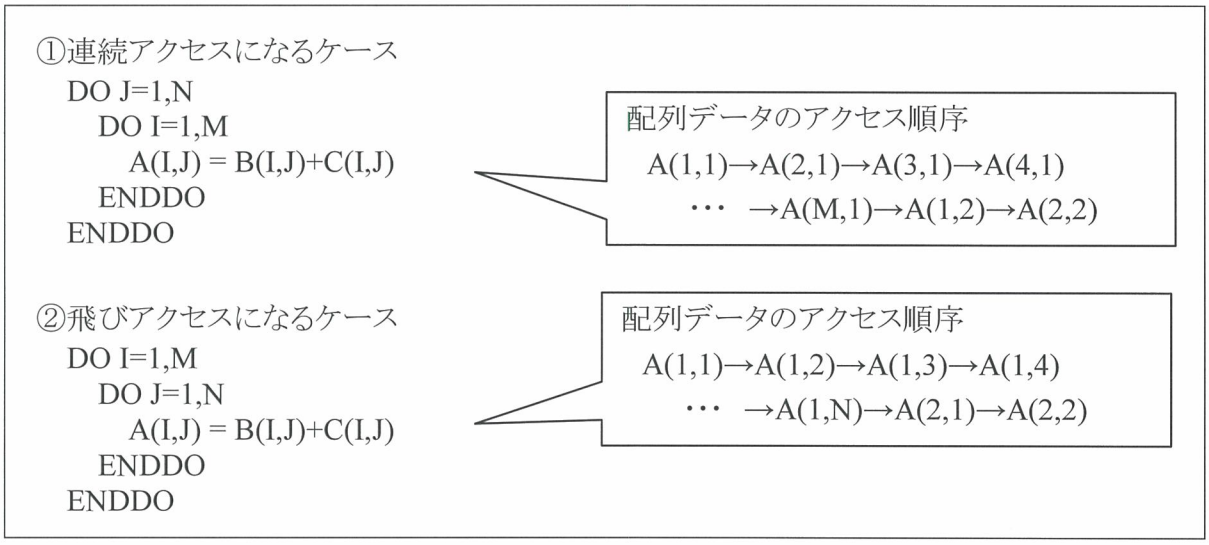


図 4.3.2 ループ構造とメモリアクセス順序

図 4.3.2 の②に示したように配列データが飛びアクセスになる場合、飛びアクセスの間隔(ストライド)は、配列 A の 1 次元目の定義の大きさになる。このストライドが 2 のべき乗倍になると、同じポートにアクセスが集中することになり、ポートの競合による待ち時間が発生する。特に 32 飛びでアクセスを行うとすべてのメモリアクセスが同じポートに集中することになる。コンパイラはメモリアクセスが連続になるようにループの交換などの最適化を行うが、コンパイラで回避できない場合は、配列の定義を奇数にするなどの方法でポートの競合を回避することができる。

SX-9からプログラム実行解析情報(PROGINF)において、バンクコンフリクト時間をCPU側のポートの競合する時間と主記憶側のメモリバンク側で競合する時間に分けて出力する。上記のように、配列データの飛びアクセスによるCPUのポートの競合による時間は「CPUポート競合」にて出力される。また自動並列化、OpenMP並列化されたプログラムの複数のタスクから同時に同一のメモリバンクにアクセスした場合や、同一の配列データ、スカラ変数を複数のタスクで参照している場合、別のプロセスが、たまたま同一のメモリバンクに繰り返しアクセスするような場合による待ち時間は、「メモリネットワーク競合」にて出力される。F_PROGINF=DETAIL指定時に以下のように出力される。

```
バンクコンフリクト時間
  CPUポート競合 (秒)          :          0.455945
  メモリネットワーク競合 (秒)  :          2.416176
```

(2)メモリアクセス回数の削減

図 4.3.3 に示すような差分式の計算では、同じ配列データのアドレスの前後を参照する(この例では、配列 b、配列 c の 1 次元目、2 次元目のアドレスの-1、+1 のデータを参照している)。この場合、ループをアンローリングすることで、配列データのロード回数を削減することが可能になる。最内ループはベクトル化の対象となっているので、外側のループに対して、OUTERUNROLL 指示行を指定して、アンローリングを行う。この例では 30 行目のループの先頭に outerunroll=4 を挿入し、4 段のアンロールを指定した。

```
28:+------>          do k = 2,nz-1
29: |            !cdir outerunroll=4
30: |+----->           do j = 2, ny - 1
31: ||V---->              do i = 2, nx - 1
32: |||                      x = a(i,j) * ( (b(i,j,k)     * c(i,j,k)
33: |||              &                          - b(i-1,j,k) * c(i-1,j,k)
34: |||              &                          - b(i+1,j,k) * c(i+1,j,k) )
35: |||              &                      +   ( b(i,j,k)     + c(i,j,k)
36: |||              &                      +     b(i,j-1,k) + c(i,j-1,k)
37: |||              &                      +     b(i,j+1,k) + c(i,j+1,k) ))
38: |||                       d(i,j,k) = d(i,j,k) + x
39: ||V----                  end do
40: |+-----                end do
41:+------                end do
```

図 4.3.3 外側ループのアンロールを指定した例

4 回の実行で配列 b は 20 要素、配列 c も 20 要素をロードする必要があったが、外側のルー

プを4段アンロールすることで、それぞれ14要素をロードするだけになり、12回のロードを削減することが可能になる。図 4.3.4 にアンロールしたソースイメージを示す。

```
30: |+-----> do j = 2, ny – 1, 4
31: ||V----->   do i = 2, nx - 1
32: |||           x1 = a(i,j) * ( (b(i,j,k) * c(i,j,k)
33: |||     &                       - b(i-1,j,k) * c(i-1,j,k)
34: |||     &                       - b(i+1,j,k) * c(i+1,j,k) )
35: |||     &                   +   ( b(i,j,k) + c(i,j,k)
36: |||     &                   +     b(i,j-1,k) + c(i,j-1,k)
37: |||     &                   +     b(i,j+1,k) + c(i,j+1,k) ))
39: |||           x2 = a(i,j+1) * ( (b(i,j+1,k) * c(i,j+1,k)
40: |||     &                       - b(i-1,j+1,k) * c(i-1,j+1,k)
41: |||     &                       - b(i+1,j+1,k) * c(i+1,j+1,k) )
42: |||     &                   +   ( b(i,j+1,k) + c(i,j+1,k)
43: |||     &                   +     b(i,j,k) + c(i,j,k)
44: |||     &                   +     b(i,j+2,k) + c(i,j+2,k) ))
46: |||           x3 = a(i,j+2) * ( (b(i,j+2,k) * c(i,j+2,k)
47: |||     &                       - b(i-1,j+2,k) * c(i-1,j+2,k)
48: |||     &                       - b(i+1,j+2,k) * c(i+1,j+2,k) )
49: |||     &                   +   ( b(i,j+2,k) + c(i,j+2,k)
50: |||     &                   +     b(i,j+1,k) + c(i,j+1,k)
51: |||     &                   +     b(i,j+3,k) + c(i,j+3,k) ))
53: |||           x4 = a(i,j+3) * ( (b(i,j+3,k) * c(i,j+3,k)
54: |||     &                       - b(i-1,j+3,k) * c(i-1,j+3,k)
55: |||     &                       - b(i+1,j+3,k) * c(i+1,j+3,k) )
56: |||     &                   +   ( b(i,j+3,k) + c(i,j+3,k)
57: |||     &                   +     b(i,j+2,k) + c(i,j+2,k)
58: |||     &                   +     b(i,j+4,k) + c(i,j+4,k) ))
                                :
64: ||V----   end do
65: |+-----  end do
```

図 4.3.4 外側ループのアンロールのソースイメージ

図 4.3.5 に外側ループのアンロールによる主記憶からのロード回数の削減による改善結果を示す。約 45%の改善となった。

PROG.UNIT	FREQUENCY	EXCLUSIVE TIME[sec](%)	AVER.TIME [msec]	MOPS	MFLOPS	V.OP RATIO	AVER. V.LEN	VECTOR TIME	I-CACHE MISS	O-CACHE MISS	BANK CONF
sub	20000	19.201(99.9)	0.960	33306.2	12744.5	97.87	106.3	19.174	0.0001	0.0018	0.4725
sub	20000	13.227(99.9)	0.661	47955.7	18501.4	98.68	106.3	13.199	0.0000	0.0001	0.5413

図 4.3.5 外側ループのアンロールによる効果

(3)ADB の利用

SX-9 では、メモリアクセス高速化のためにADB（Assignable Data Buffer)と呼ばれるキャッシュが、主記憶とベクトルレジスタの間に追加された。ADBは通常のキャッシュとは異なり、選択的にデータをバッファリングすることができる。たとえば、他の配列や変数のアクセスによりデータ

がADBから追い出されないように、頻繁にアクセスされる配列のみをADBにバッファリングしておくことで、その配列を長期間に渡って高速にアクセスできるようになる。図4.3.6にADBの概念図を示す。

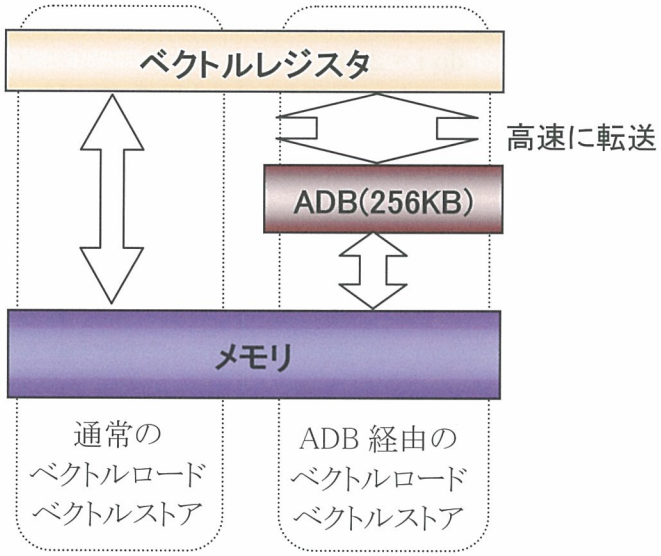


図 4.3.6　ADB の概念図

ある配列・ポインタの指示先をON_ADB指示行で指定したとき、それらのベクトルロード・ベクトルストアはADB経由で行われる。最初のベクトルロードまたはベクトルストアを行うとき、データがADBにバッファリングされる。次にそれらのデータを利用するときには、ADBにバッファリングされたデータがロードされる。ADBからは、主記憶からデータをロードするよりも高速にデータをロードできるので、2回目以降のベクトルロードを高速化できる。図4.3.7にADBの使用方法を示す。

```
      SUBROUTINE SUB
      COMMON A(4096),B(4096),C(4096)
!CDIR ON_ADB(A)
!CDIR ON_ADB(C)
      DO I=1,4096
            = A(I) + ...      ①
        C(I) = ...            ②
            = A(I) * ...      ③
      END DO
            :
!CDIR ON_ADB(A)
!CDIR ON_ADB(C)
      DO I=1,4096
        B(I) = A(I) + C(I)    ④
      END DO
      END SUBROUTINE
```

①の配列Aのベクトルロード時に配列Aのデータ がADBにバッファリングされる。
②の配列Cのベクトルストア時に配列Cのデータ がADBにバッファリングされる。
③の配列Aの2番目参照では①でADBにバッファリングされた配列Aのデータがベクトルロードされる。
④の2番目のループの配列A，配列Cの参照では、①②でADBにバッファリングされた配列A、配列Cのデータがベクトルロードされる。配列BはON_ADB指示行で指定されていないため、ADBを経由せず、主記憶に直接ベクトルストアされる。このとき，配列Bのデータ はADBにバッファリングされない．

図 4.3.7　ADB の使用方法

図 4.3.8 に示すソースコード 1 における ADB の使用例を紹介する。

```
      do 10 k=0,Nz
      do 10 i=0,Nx
!cdir on_adb(H_x,H_y,H_z)
      do 10 j=0,Ny
        E_x(i,j,k)=
     &    C_x_a(i,j,k)*E_x(i,j,k)
     &   +C_x_b(i,j,k)*((H_z(i,j,k)-H_z(i,j-1,k))/dy
     &   -(H_y(i,j,k)-H_y(i,j,k-1))/dz-E_x_Current(i,j,k))
        E_y(i,j,k)=
     &    C_y_a(i,j,k)*E_y(i,j,k)
     &   +C_y_b(i,j,k)*((H_x(i,j,k)-H_x(i,j,k-1))/dz
     &   -(H_z(i,j,k)-H_z(i-1,j,k))/dx-E_y_Current(i,j,k))
        E_z(i,j,k)=16
     &    C_z_a(i,j,k)*E_z(i,j,k)
     &   +C_z_b(i,j,k)*((H_y(i,j,k)-H_y(i-1,j,k))/dx
     &   -(H_x(i,j,k)-H_x(i,j-1,k))/dy-E_z_Current(i,j,k))
   10 CONTINUE
```

図 4.3.8　ADB を使用したソースコード 1

インデックス i のループ長が 50 と小さいため、コンパイラはインデックス j のループをベクトル化の対象としている。各配列の大きさは 858MB であるため、256KB の ADB にすべてのデータを載せることはできない。しかし、このループは差分式になっており、例えば、図 4.3.8 の下から 3 行目の *H_y(i,j,k) - H_y(i-1,j,k)* に着目すると、*H_y(i,j,k)* のデータを次の *i+1* の *H_y(i-1,j,k)* が再利用することになる。図 4.3.8 の赤字が再利用性のある部分で、再利用に必要な ADB の容量は 36KB になる。配列 *H_x, H_y, H_z* を ADB に載せることを選択した。実行結果を図 4.3.9 に示す。再利用性のある 3 つの配列を ADB に保持することにより、本コードの実効性能が 15% 向上することが明らかになった。これは再利用性のあるデータを ADB からプロセッサへ供給することにより、メモリへのアクセス回数が減少したためである。

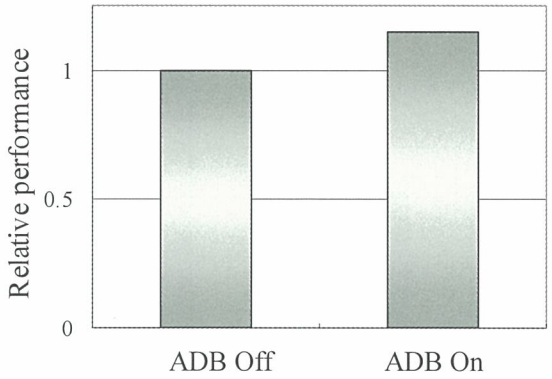


図 4.3.9　ADB による性能向上（ソースコード 1）

次に間接参照（リストベクトル）を使用したプログラムに対して、ADB を適用した例を紹介する。

図 4.3.10 にソースコードを示す。

```
!cdir on_adb(dvecw),nodep
      do ii=jj,min(jj+lvec-1,iplast)
        kk = ii - jj + 1
        aa     = xp1(ii)/delx
        ic     = int( aa+half )
        raa1 = ic
        dd1    = fact1*(aa+half-raa1)
        dd2    = fact1*(raa1-aa+half)
        dvecw(ic    ,kk,1)=dvecw(ic    ,kk,1)+dd1*vp1(1,ii)
        dvecw(ic-1,kk,1)=dvecw(ic-1,kk,1) +dd2*vp1(1,ii)
        dvecw(ic    ,kk,2)=dvecw(ic    ,kk,2)+dd1*vp1(2,ii)
        dvecw(ic-1,kk,2)=dvecw(ic-1,kk,2) +dd2*vp1(2,ii)
        dvecw(ic    ,kk,3)=dvecw(ic    ,kk,3)+dd1*vp1(3,ii)
        dvecw(ic-1,kk,3)=dvecw(ic-1,kk,3) +dd2*vp1(3,ii)
        dvecw(ic    ,kk,4)=dvecw(ic    ,kk,4)+dd1
        dvecw(ic-1,kk,4)=dvecw(ic-1,kk,4) +dd2
      enddo
```

図 4.3.10　ADB を使用したソースコード 2

このループの計算に使用される配列 *dvecw* は 1 次元目のアドレスが変数 *ic* による間接参照になっている。このため、ベクトルギャザ命令におけるメモリレイテンシが見えてしまう。配列 *dvecw* の大きさは 254KB であり、この配列全体を ADB に載せることにより、メモリレイテンシの削減を期待できる。

実行結果を図 4.3.11 に示す。この結果より、比較的小容量の配列を繰り返しアクセスするループの場合、その配列を ADB に保持して再利用することによってメモリアクセス回数を削減可能であり、その結果として大幅な速度向上が得られた。

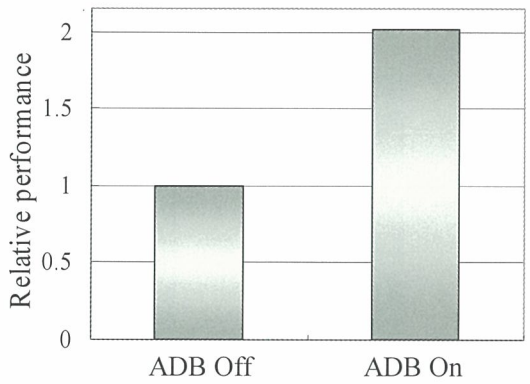


図 4.3.11　ADB による性能向上（ソースコード 2）

4.4 並列処理による高速化

SX-9における高速演算機能として並列処理がある。SX-9ではFORTRAN、Cコンパイラによる自動並列化とMPIによる並列化が可能である。自動並列化では1ノード16CPUまでの並列処理が可能であり、MPIでは、ノードを跨った並列化が可能である。

(1)自動並列化

SX-9は、4.1の特長の通り、共有メモリ方式のアーキテクチャにより、1TBの大規模な主記憶を16個のCPUで共有することができる。SX-9のコンパイラは、この共有メモリ方式を使用した自動並列化機能を提供している。一般に、並列化を行う時には、並列化しても結果が変わらないことを保証するために、データの依存関係の解析を行い、細心の注意を払ってプログラムの変形や指示行の挿入を行わなければならないが、自動並列機能を利用すると、それらの作業をコンパイラが自動的に行う。

自動並列化の基本的な方針は、外側のループを分割して、複数のCPUで並列に実行することである。図4.4.1に、コンパイラがループを4つのタスク(仕事)に分割して実行するイメージを示す。

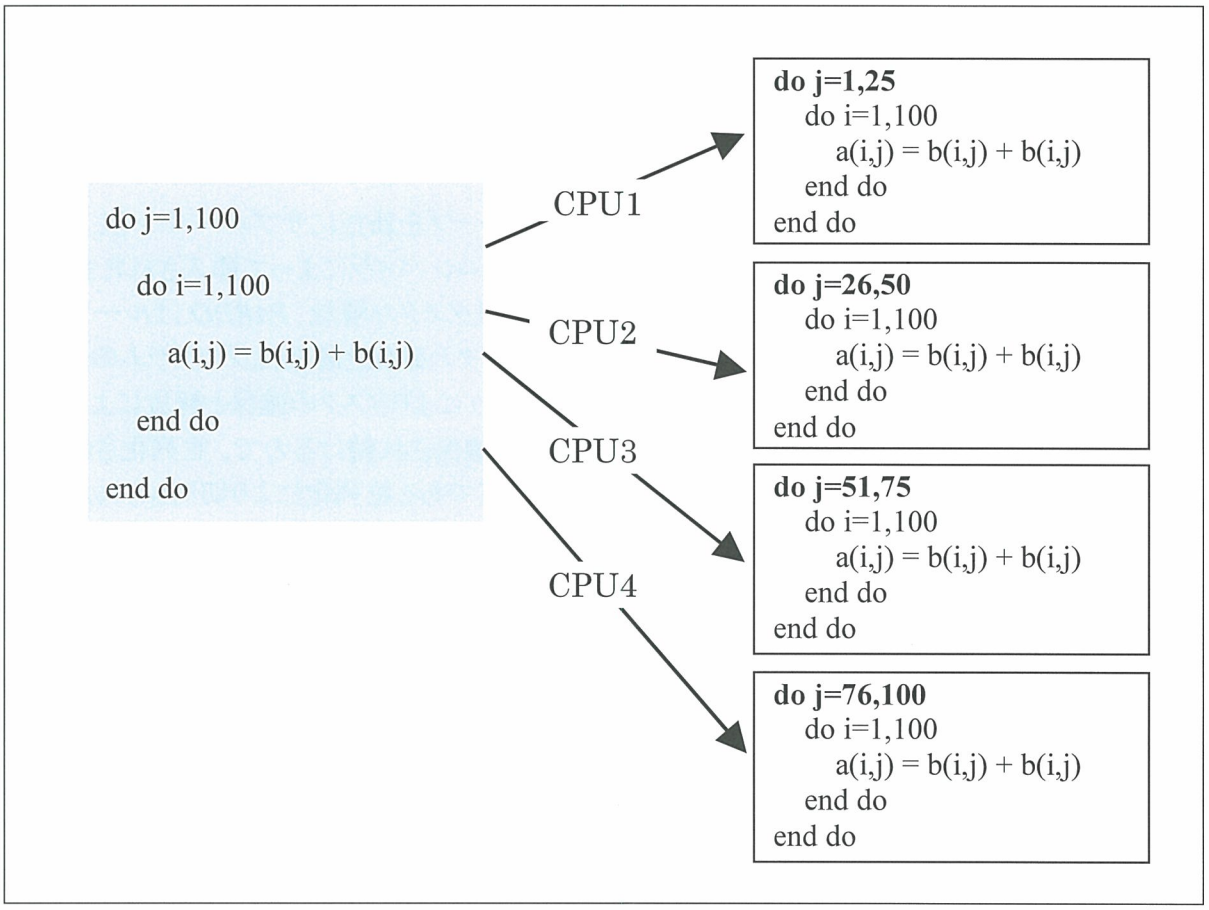


図 4.4.1 並列実行のイメージ

FORTRAN90/SXコンパイラでは、コンパイルオプション-Pautoを指定するだけで、自動並列化機能を利用することができる。コンパイラは、依存関係の解析、タスクの生成、データの分割などをすべて自動で行う。コンパイルオプション-Wf,pvctl fullmsgを指定することで、自動並列

化に関する詳細なメッセージを出力することが可能になる。

-Pauto –Wf,-pvctl fullmsg

コンパイラの自動並列化機能は、プログラムを解析し、並列化した場合の効果も考慮して並列化を行う。たとえば、並列化すれば十分性能が向上するだけの粒度(演算量)をもっているか、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、並列化が可能な場合は、ループやデータの定義の書き直しを行う。図4.4.2にコンパイラが行う自動並列化の内部的なイメージを示す。

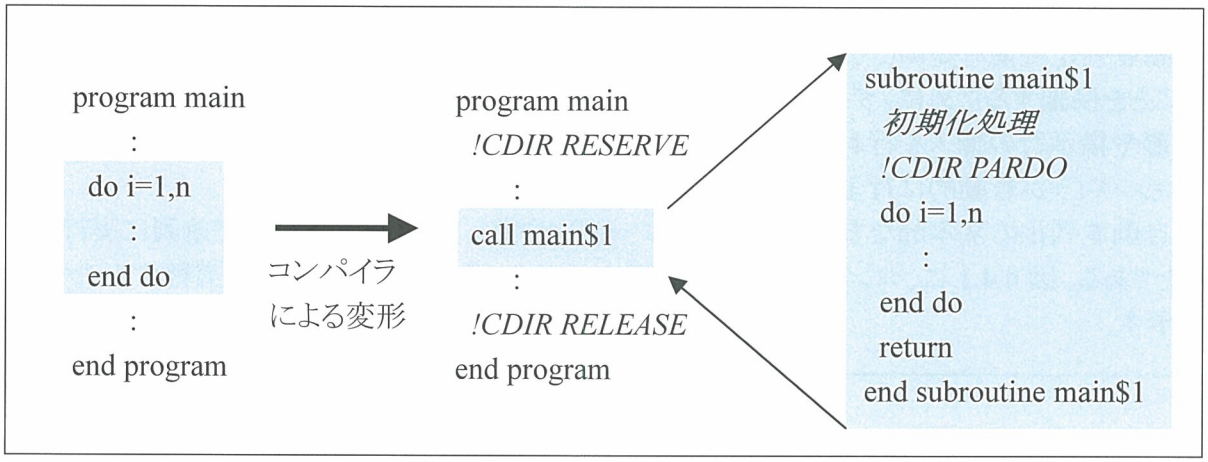


図 4.4.2 自動並列化によるコンパイラの変形イメージ

コンパイラは並列実行可能なループを検索し、そのループを新たにサブルーチンとして切り出して並列化する。図 4.4.2 の中で*!CDIR* で始まる行はコンパイラによって挿入された並列制御のためのもので、*RESERVE* はタスクの確保、*RELEASE* はタスクの解放、*PARDO* はループを並列実行することを指示する。SX-9 のコンパイラから、タスクの確保と解放はプログラムの最初と最後に行う**-Wf,-pvctl res=whole** が規定値となった。これによりタスクの確保と解放によるオーバヘッドが削減されるが、プログラムの実行中、タスクが確保され続けるので、並列化されていない部分でも CPU 時間は消費される。これまでの規定値である並列化により切り出されたループ単位でタスクの確保と解放を行いたい場合は、コンパイルオプション**-Wf,-pvctl res=parunit** を指定する。

(2)MPI による並列化

SX-9 を複数ノード使用して、大規模シミュレーションを行うためには、分散並列化を行う必要がある。分散並列化を行うためには、利用者が MPI(Message Passing Interface)ライブラリを用いて、プログラムの並列化を行う必要がある。SX-9 が提供する MPI ライブラリは、MPI フォーラムにより開発・規格化された MPI-1、MPI-2 の仕様に準拠しており、他のプラットフォーム上の MPI ライブラリで並列実行されたプログラムコードをそのまま利用することができる。

分散並列化を行う際に、最初に検討しなければならないのは、並列化の対象が何であるかを見極めることである。シミュレーションの対象となる物理現象の中で、空間的な並列性がある場合は、領域分割法による並列化が可能になる。また、現象の並列性がある場合には、処理の並列化が可能になる。

領域分割法とは、シミュレーションの対象となるある空間をできるだけ均等に分割して、複数

のプロセス(CPU)で分担して、並列に実行するものである。したがって、空間的並列性があること(分割して計算することが可能)が必要になる。分割した空間の境界面でデータの参照が必要になる場合、MPI ライブラリを用いてデータの転送を行う。図 4.4.3 に領域分割法のイメージを示す。

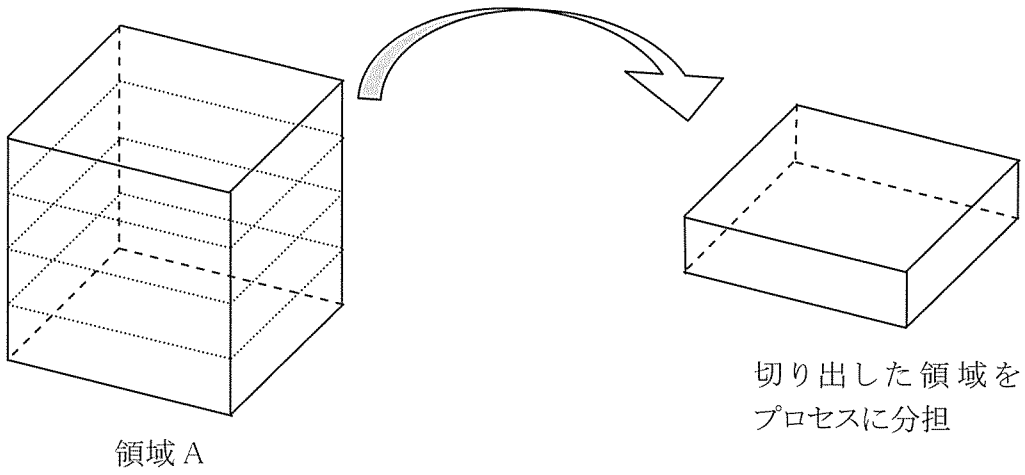


図 4.4.3 領域分割のイメージ

分散並列化で高い並列性能を得るために必要なことは、次の 2 点である。

・各プロセスの処理の均衡(インバランスを起こさない)

・転送の効率化(少ない転送回数)

分割した領域で、処理内容が均一でない(領域ごとに IF 分岐などで異なる処理を実行する必要。収束に至るまでの繰り返し回数が異なるなど)場合、インバランスが発生する。データ転送のタイミングでプロセスの同期を取る必要がある場合、処理時間の長いプロセスを待ち合わせる必要があり、先に処理を終えたプロセスに待ちが発生する。インバランスを解消するためには、領域の分割方法の変更を検討する必要がある。大きく領域を分割するブロック分割だけなく、小さく領域を分けて複数の領域をプロセスに分担することや、サイクリックにプロセスに分担することを検討する。

また MPI には様々なデータ転送方法が提供されているが、大きく分類すると 1 対 1 通信と集団通信に分けられる。必要に応じて最適な転送方法を選択する必要がある。

・1 対 1 通信 … 領域の境界面のデータの転送(MPI_SEND/MPI_RECV)。

・集団通信 … 領域を各プロセスに配る(MPI_SCATTER)あるいは各プロセスから集める(MPI_GATGER)。
入力データを各プロセスに配る(MPI_BCAST)。
総和や MAX/MIN などのリダクション処理(MPI_REDUCE)。

4.5 高速化事例

4.5.1 平面法によるベクトル化の事例

多重ループのベクトル化において、ベクトル化の対象となる配列の全ての次元に対して依存関係がある場合、そのループはループ構造を入れ替えたとしてもベクトル化することはできない。しかし、平面法を適用し配列のアクセス順序を工夫することで依存関係を取り除くことができ、ベクトル化が可能になる。ここでは、東北大学大学院工学研究科中橋研究室で開発されたシミュレーションコードに平面法を適用した事例を紹介する。

(1)シミュレーションコードの特徴

今回平面法を適用したシミュレーションコードは非圧縮性流体解析を行うプログラムで、最もコストの高いループは圧力のポアソン方程式を SOR(Successive Over Relaxization)法で計算している部分である。図 4.5.1 にこのループの概要を示す。配列 *pcnt* は *i, j, k* 全ての方向にベクトル化不可の依存関係があり、このままの形ではベクトル化することはできない。このような構造のループをベクトル化する方法の 1 つとして、ベクトル化不可の依存関係がある部分を漸化式の形に修正する手法が挙げられる。コンパイラは、漸化式のパターンであることを認識すると専用のベクトル命令を使用するためベクトル化することができる。しかし、式が複雑であればプログラムを大幅に変形する必要があり、その結果、演算順序が変わり結果に誤差が発生してしまう場合がある。そこで、今回は平面法を使用したプログラムチューニングによりベクトル化を行った。

```
do k = 1, n
    do j = 1, n
        do i = 1, n
              :
          p0        = pcnt ( i, j, k )
          pxm       = pcnt ( i – 1, j, k )
          pym       = pcnt ( i, j – 1, k )
          pzm       = pcnt ( i, j, k – 1 )
              :
          p_a       = c0 * ( pxm * xm + pym * ym + pzm * zm +  . . .  )
          pcnt ( i, j, k ) = p0 + cnst_p * ( p_a – p0 )
        end do
    end do
end do
```

図 4.5.1 ベクトル化不可のループの概要

(2)平面法

平面法は、ベクトル化の対象となる配列の全ての次元について依存関係が存在する、つまり、ループの入れ替えを行ったとしても依存関係が解消されないような場合に、配列の依存関係がなくなるような演算順となるリストをあらかじめ作成し、ベクトル化を可能にするチューニング手法である。計算式そのものを修正するのではなく配列アクセスの順序を変更するため計算結果に影響はない。本チューニングでは図 4.5.1 に示したような三次元配列に対して平面法を適用しているが、まずは簡単のために二次元配列の場合について考える。図 4.5.2 は *i* 方向、*j* 方向に依存関係がある二重ループの例である。

```
do j = 1, n
    do i = 1, n
        tmp = array( i ) * work( i – 1, j ) + array( j ) * work( i, j – 1)
        work( i, j ) = const * ( tmp – work( i, j ) )
    end do
end do
```

図 4.5.2　i, j 方向に依存関係のある二重ループの例

ベクトル化は原則として最内側ループに適用されるため、この場合 *i* 方向がベクトル化の方向となる。ベクトル化が行われた場合の配列 *work* の演算順を模式的に表したのが図 4.5.3(a)である。横軸が *i* 方向、縦軸が *j* 方向を表す。図中の数字は配列 *work* の演算が行われる順序を表し、同じ数字はベクトル化が行われた場合に同時に演算される要素であることを示す。図 4.5.3(b)は配列 *work* の演算順を具体的に示したものである。*n* が 256 以下であれば *i* 方向の演算は各 *j* の値について一回のベクトル命令で同時に実行される。この演算順では、*i* 方向に依存関係がありベクトル化することはできない。仮にループを入れ替えたとしても、*j* 方向に対しても依存関係があるため、ベクトル化することはできない。

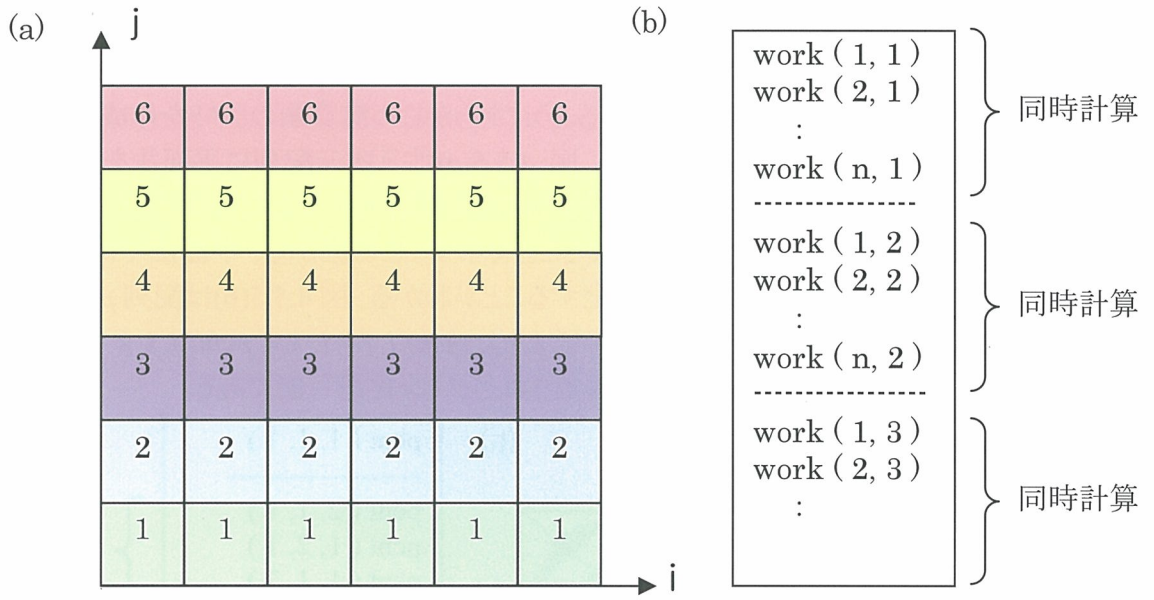


図 4.5.3（a）配列 work の演算順の模式図（b）配列 work の演算順

平面法では、配列 *work* の演算順が図 4.5.4(a)のようになるリストをあらかじめ作成する。つまり、*i* と *j* の和が同じとなる配列の演算が同時に行われるようにする。図 4.5.4(b)は配列 *work* の計算順を示し、平面法を適用すると配列の演算に依存関係がなくなることがわかる。つまり、*work(i, j)*の演算には *work(i – 1, j)*、 *work (i, j – 1)*の演算結果が必要であるが、平面法を適用した演算順ではそれらは直前の繰り返しですでに演算されているためベクトル化を行うことができる。

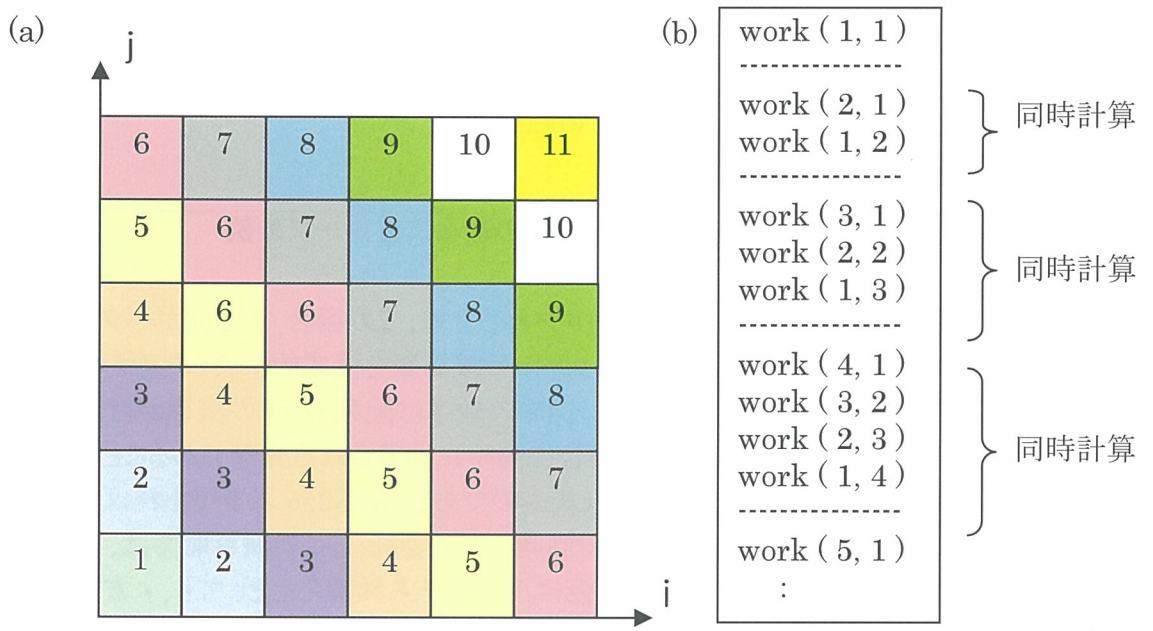


図 4.5.4 平面法適用時の（a）配列 work の演算順の模式図 （b）配列 work の演算順

次に、図 4.5.1 に示したような三次元配列の場合を考える。この場合も、添え字の和($i + j + k$)が同じとなる配列の演算が同時に行われるようにあらかじめ演算順のリストを作成し、配列の依存関係を取り除いた上でベクトル化を行う。図 4.5.5(a)は三次元配列に平面法を適用しベクトル化を行った場合に、同時に演算される配列の位置を模式的に示したものである。図中の同一色の平面上にある配列が同時に演算されることを示す。この図から、平面法では同時に実行される配列要素数（ベクトル長）が毎回変化することがわかる。図 4.5.5(b)は配列 *pcnt* の演算順を示す。*pcnt(i, j, k)*の演算時には*pcnt(i – 1, j, k)*、*pcnt(i, j – 1, k)*、*pcnt(i, j, k – 1)*の演算はすでに終了しており、配列の依存関係が解消されベクトル化が可能である。

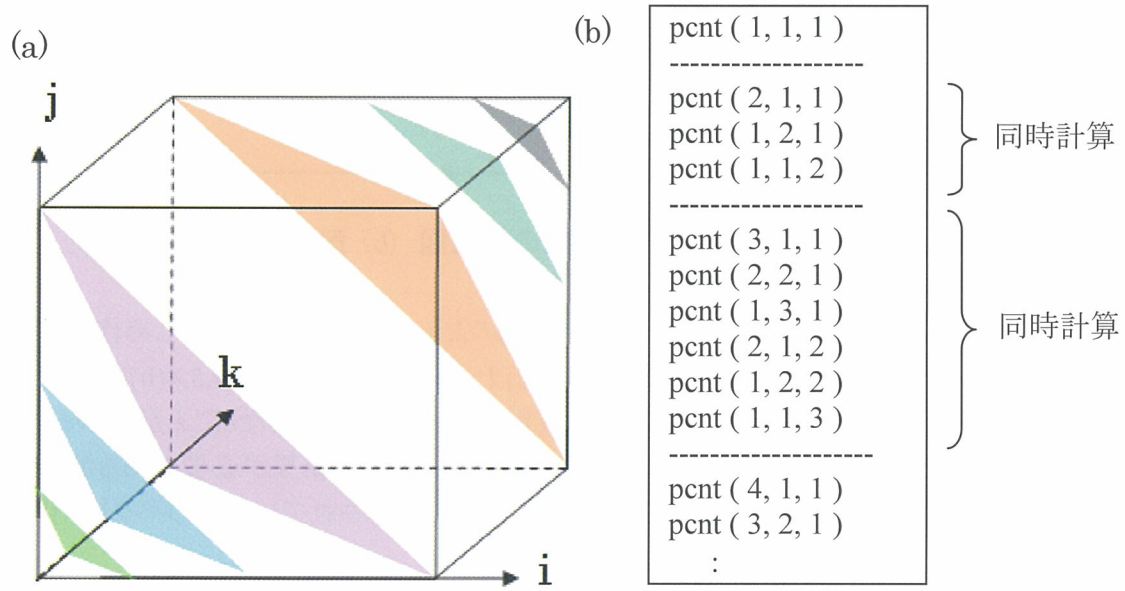


図 4.5.5 平面法適用時の（a）同時に演算される配列 pcnt の位置の模式図
（b）配列 pcnt の演算順）

(3)平面法適用による性能向上

表 4.5.1 はオリジナルのコードと平面法を適用した場合の測定結果を比較したものである。測定は SX-9 で行い、測定区間は図 4.5.1 に示したオリジナルコードでベクトル化不可であったループで，計算ステップは 20 とした。平面法の適用により、約 13.6 倍の性能向上が得られた。平均ベクトル長、ベクトル演算率についてもチューニングにより改善されている。

表 4.5.1 SX-9 での測定結果

	実行時間(秒)	平均ベクトル長	ベクトル演算率
オリジナル	5,644.39	32.0	46.28%
平面法	414.40	187.8	99.90%

配列の全ての次元に対して依存関係が存在し、そのままの形ではベクトル化することができない多重ループについて、平面法を適用しベクトル化を行った事例を紹介した。平面法は配列の依存関係がなくなるように配列アクセスの順番を変更し、ベクトル化を可能にするチューニング手法であり、今回は三次元配列に対して平面法を適用しベクトル化を行った。

4.5.2 MPI 化の事例

MPI を用いて分散並列化を行った事例を紹介する。ここでは、4.5.1 と同じ東北大学大学院工学研究科中橋研究室で開発されたシミュレーションコードを用いて並列化を行い、SX-9 で大規模シミュレーションを実行した事例について説明する。

(1)並列性の検討

中橋研究室で開発された Building-Cube(直交格子積み上げ)法は、計算空間にさまざまなサイズの立方体(cube)を積み上げ、個々の cube 内には高密度な等間隔直交格子(cell)を用いる計算法である。さまざまなサイズの cube を用意することで流れの局所性に適合した格子密度を可能にしている、cube 内に等間隔直交格子を用いることで高精度化を可能にするとともに、アルゴリズムの単純さを確保する。また空間を cube で分割することは、大規模並列化や大規模データの後処理でも有利である。図 4.5.6 に Building-Cube 法の概念図を示す。

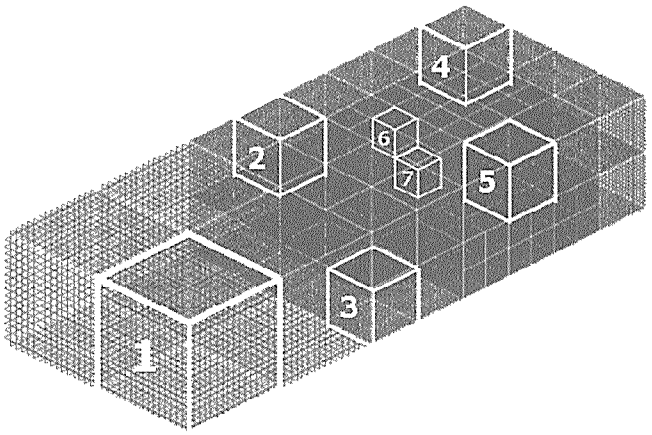


図 4.5.6 Building-Cube 法の概念図

cube 形状の小領域を積み上げるこの方法には幾つかの利点があり、並列化の検討において以下の点に注目した。

① 計算コードは、cube 内の計算を行うサブルーチンと、cube 間の情報交換や計算順序などを扱う部分とに分けられ、単純なプログラム構造になる。

② 各 cube には、同数の直交格子 cell を内包していることから、各 cube 内の計算量は均等になる。

MPI を用いて分散並列化を行う際にまず並列化の対象の検討が必要になる。本シミュレーションコードでは、計算空間に cube を積み上げる手法であることから cube 単位に独立して計算を行っており、並列性がある。また各 cube に内包される cell の数が同じで、計算量も均等である性質から、複数の cube を MPI プロセスに割り当てる(ブロック分割)こととした。

(2)転送方法の検討

空間(領域)の分割を行うことで、各 MPI プロセスは自分に割り当てられた領域についてのみ計算を行う。プログラム実行の過程で、自分が担当しない領域のデータを参照する場合や計算結果を集約する場合は、MPI プロセス間の通信によりデータの転送を行う。通信手段としては、次の方法に大別される。

- 一対一通信
- 集団通信

並列性能の向上を図るためには、通信時間をできるだけ短くすることが必要である。そのためには、できるだけ少ない通信回数でデータ転送を実現できるかを検討する。本シミュレーションコードでは、計算の過程で隣接する cube 間の情報交換を行う必要がある。隣接する cube を同じ MPI プロセスが担当する場合は、データの転送は必要ないが、別の MPI プロセスが担当する場合は、データの転送が必要になる。また隣接する cube の大きさが等しいとは限らず、1/4 の大きさの cube が隣接する場合は、4 つの cube の情報が必要となる。6 方向に面しているので、最大で 24 個の cube の情報が必要になる。図 4.5.7 に隣接する cube の情報交換の概念図を示す。

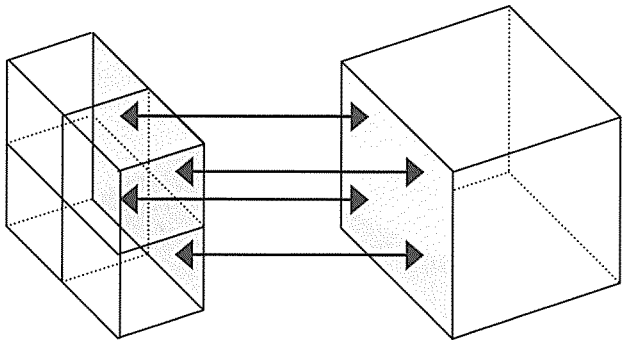


図 4.5.7 隣接する cube の情報交換

まず集団通信により、すべての cube 情報を全 MPI プロセスで更新する方法を検討した。隣接する cube 以外の必要ない情報についてもデータ転送することになるが、一度の通信で多くのデータを転送するので、通信回数の削減が可能になる。各 MPI プロセスに分割した cube の情報をルートプロセスに集めて、全 MPI プロセスに対して再配信を行う集団通信の MPI_ALLGATHERV を用いて、データ転送を行った。この結果、すべての MPI プロセスが更

新された全 cube 情報を持つことになる。図 4.5.8 に MPI_ALLGATHERV を用いて cube の情報交換を行う手順を示す。

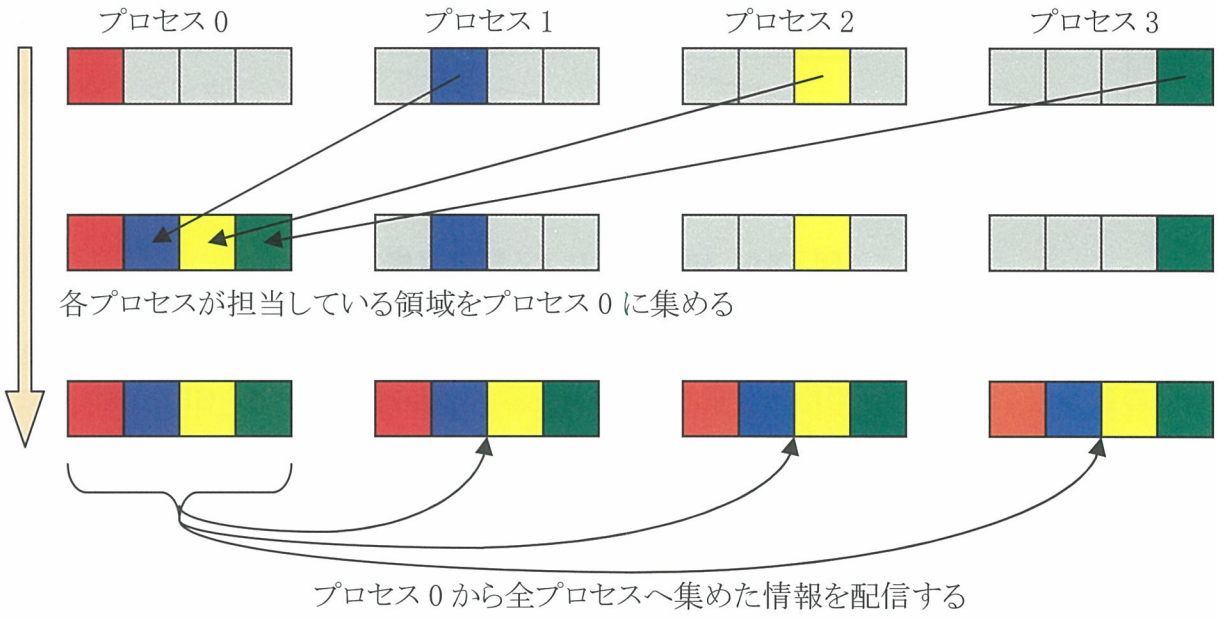


図 4.5.8 MPI_ALLGATHERV による転送手順

各 cube が内包する cell は、3 次元の方向を持ち、今回シミュレーションに使用したモデルにおいては、32×32×32 の大きさになる。実際の配列定義は、各次元とも糊代を設けており、倍精度実数型で 1 つの cube のサイズが約 400Kbyte になる。総数 5,930 個の cube に対して集団通信 MPI_ALLGATHERV を用いて通信すると、256MPI プロセス時における 1 回のデータ通信量が約 130GByte 程度となるため，通信時間によるオーバヘッドから並列性能が得られないことが予想される。

隣接する cube の情報だけを MPI プロセス間で通信することを検討した。cube ごとに隣接する cube の番号を検索し、その cube がどの MPI プロセスに存在するかを判定することによりデータ転送の必要の有無を決定する。またできるだけデータの転送回数を削減するため、cube 番号が連続する場合は、一度にまとめて転送するように考慮した。データ転送方法は、1 対 1 通信の MPI_SEND/MPI_RECV もしくは MPI_ISEND/MPI_IRECV を用いることも可能であるが、MPI2.0 仕様の単方向通信 MPI_GET を使用した。単方向通信を用いることの利点は、データ転送を行う際に他の MPI プロセスとの協調を必要としない通信形態であることから、ある MPI プロセスが単独で他の MPI プロセスに対してデータ転送を行うことができることにある。

単方向通信 MPI_GET は、MPI プロセスごとに予めウィンドウ領域を用意しておき、任意の MPI プロセスから、別の MPI プロセスのウィンドウ領域に対して、データを読み込むことができる。これを隣接する cube の情報交換に応用した。図 4.5.9 に MPI_PUT/MPI_GET を用いて cube の情報交換を行う手順を示す。ウィンドウ領域に情報交換が必要になる配列を定義した。

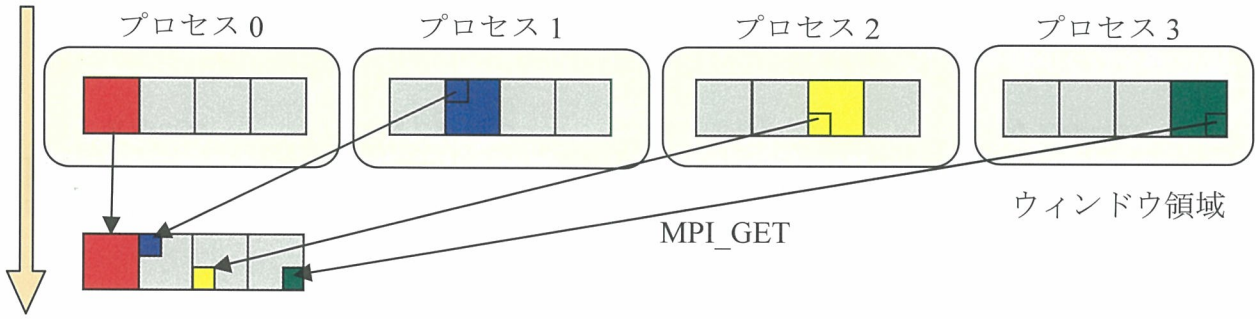


図 4.5.9　MPI_GET による転送手順

表 4.5.2 に集団通信 MPI_ALLGATHERV を用いた場合と、単方向通信 MPI_GET を用いた場合の各 MPI プロセスでの平均転送量と総転送量を示す。単方向通信 MPI_GET を用いることで、64MPI プロセス時にデータ通信量を 1/25 に削減することができた。但し、1 回のデータ転送サイズは、平均で約 900Kbyte と小さくなる。

表 4.5.2　MPI_ALLGATHERV と MPI_GET の転送量

	MPI プロセス数	4	16	32	64
MPI_ALLGATHERV	平均転送量(byte)	2.7G	2.4G	2.3G	2.3G
	総転送量(byte)	10.4T	37.1T	72.1T	142.2T
MPI_GET	平均転送量(byte)	0.91M	0.99M	0.96M	0.89M
	総転送量(byte)	1.0T	2.1T	2.8T	3.4T

収束判定やエラー判定の処理には、集団通信の MPI_ALLREDUCE を使用している。図 4.5.10 に MPI_ALLREDUCE を用いてエラー判定を行った例を示す。

(逐次版)

```
do i=1,n_cube
          :
   err    = err + err0 * err0
enddo
if(err.lt. eps_p ) exit
```

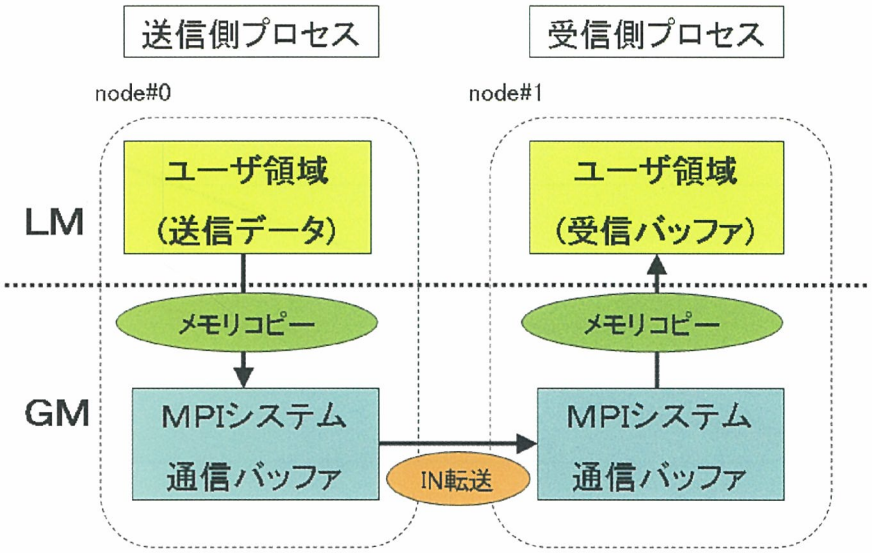
(MPI 版)

```
do i=1,n_cube
          :
   err    = err + err0 * err0
enddo
call MPI_ALLREDUCE(err,err2,1,
+      MPI_DOUBLE_PRECISION, MPI_SUM,
+      MPI_COMM_WORLD,ierr)
 if( err2 .lt. eps_p ) exit
```

図 4.5.10 MPI_ALLREDUCE によるエラー判定

(3)グローバルメモリ

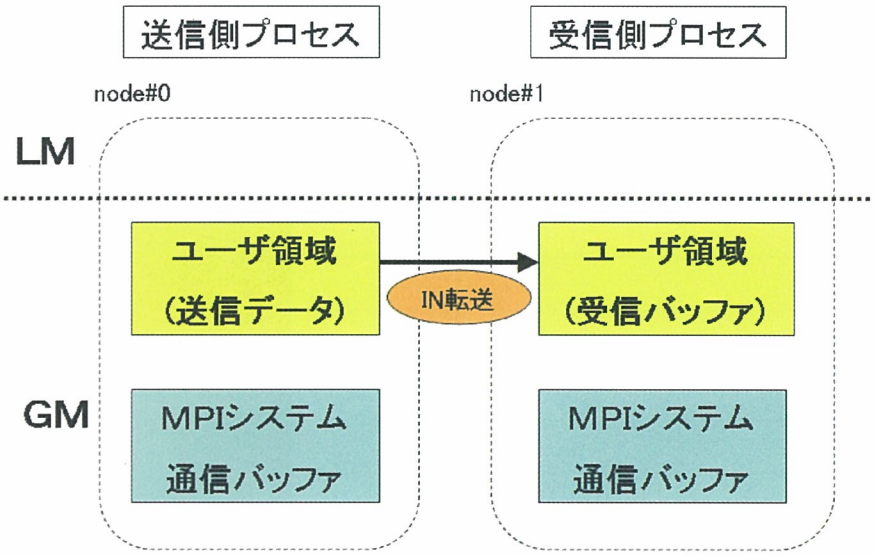
通常のユーザプログラムが使用するメモリ空間はローカルメモリにある。MPI がデータ通信を行う場合、図 4.5.11 に示すようにローカルメモリにあるデータは、グローバルメモリ上にある MPI 通信バッファにメモリコピーされ、MPI システム通信バッファ間で転送が行われる。受信側で MPI システム通信バッファから、ローカルメモリにあるユーザ領域へのメモリコピーが完了した時点でデータ通信が終了する。



LM : Local Memory　GM : Global Memory

図 4.5.11　通常の MPI データ転送手順

SX-9 では、グローバルメモリ機能を提供しており、図 4.5.12 に示すようにユーザプログラムが使用するメモリ空間をグローバルメモリ上に割り付けておくことで、直接グローバルメモリ空間でのデータ転送を行うことが可能になり、MPI システム通信バッファへのメモリコピーを省略した高速な通信を実現することができる。



LM : Local Memory　GM : Global Memory

図 4.5.12　グローバルメモリを使用した MPI データ転送手順

グローバルメモリは、通信するデータを格納する配列を動的に確保することと、コンパイルオプション**-gmalloc** を指定するだけで、使用が可能になる。図 4.5.13 にグローバルメモリの使用による SX-9 における結果の比較を示す。

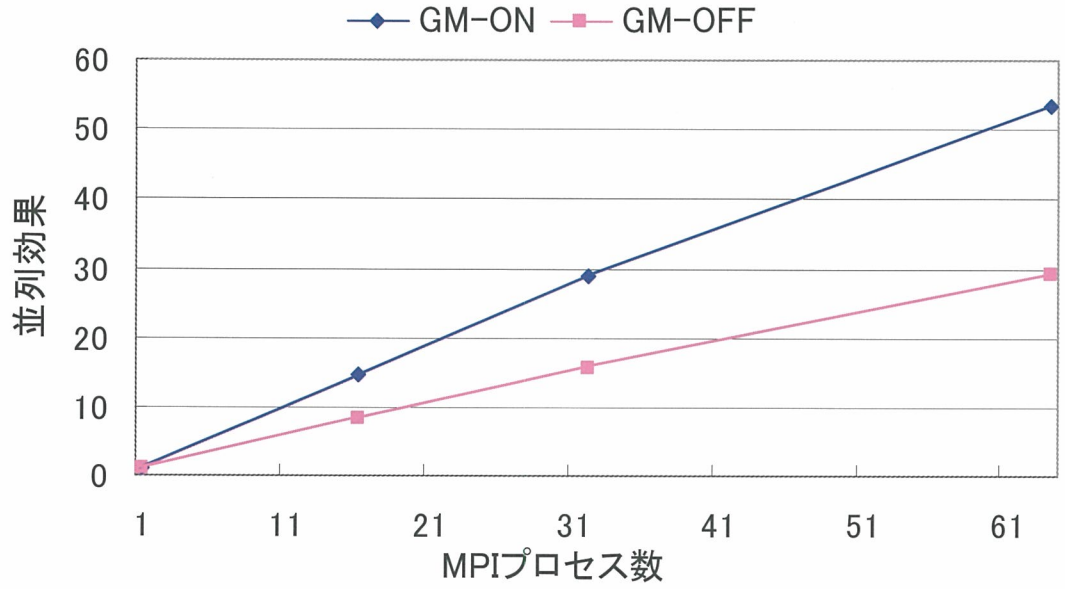


図 4.5.13 グローバルメモリによる実行時間の比較

(4)SX-9 と TX7/i9610 の実効性能比較

サイバーサイエンスセンターの SX-9 と TX7/i9610 の実効性能を比較した。MPI プロセス数は同じ 16 である。図 4.5.14 に TX7/i9610 の性能を 1.0 とした比較を示す。SX-9 は約 84 倍の性能となった。

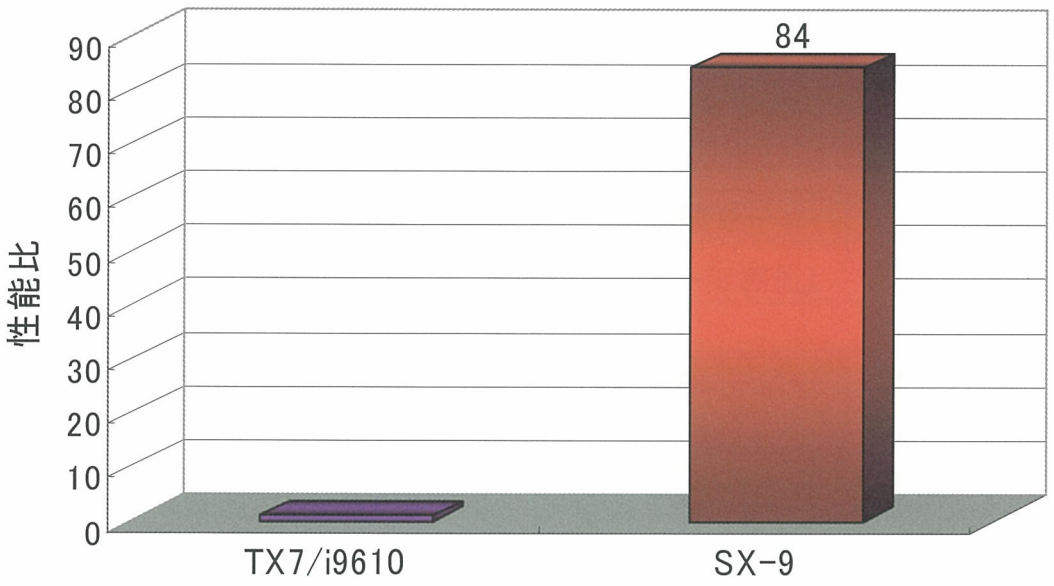


図 4.5.14 TX7/i9610 の性能を 1 としたときの SX-9 の性能倍率

(6)SX-9 大規模システムの評価

全システム 16 ノード 256CPU を使用して、大規模並列実行を行った。並列化は MPI でのみ行い、計算ステップは 20 として実行した。図 4.5.15 に測定結果を示す。

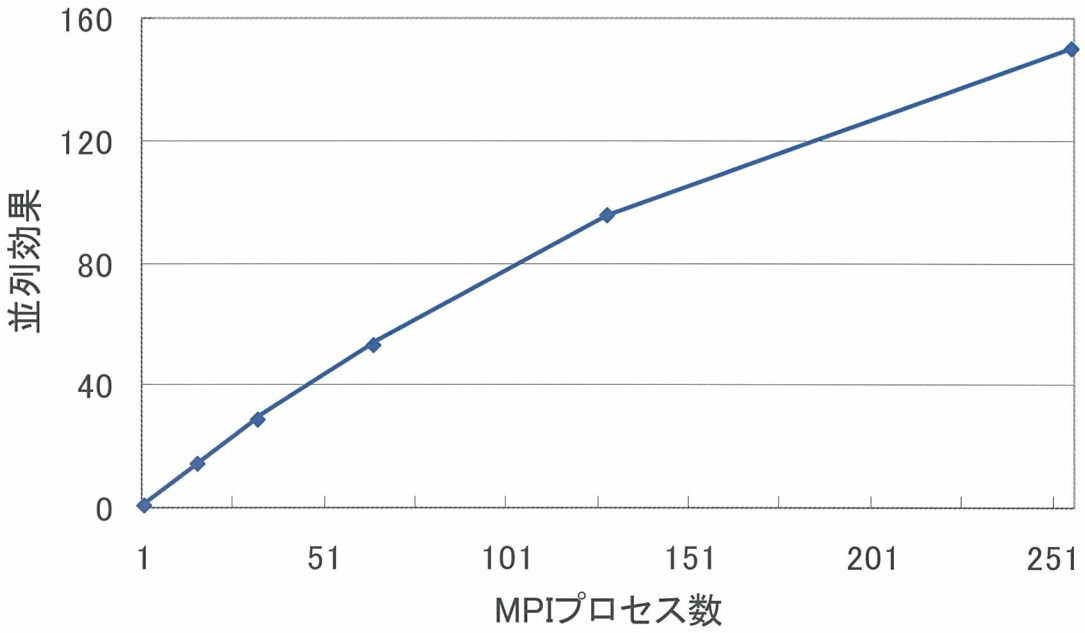


図 4.5.15　SX-9　256CPU の実行結果

16 ノード 256CPU における測定の結果から、並列化率 99.72%、SX-9 の 1CPU と比較した並列効果 150 倍を達成することができた。

本シミュレーションコードは、今までスカラ型のコンピュータにおいて、1 ノード内で実行されていた。サイバーサイエンスセンターの TX7/i9610 を 64 コア使用して実行した場合と比較して、5 万ステップ実行するのに約 36 日必要であったものが、SX-9 を 16 ノード使用するとことにより約 10 時間で終了する。SX-9 による分散並列化を利用することにより、大規模なシミュレーションを短時間で行うことができる。

以下に示す検討を行うことで，さらに大規模なシミュレーションを実現することが可能になる。

① メモリの削減…各 MPI プロセスの実行に必要ない配列の定義が存在する。削減することにより，さらに大規模なシミュレーションを可能にすることができる。

② 転送時間の削減…MPI_GET を用いることで大幅な改善を図ることができたが、256CPU での実行時に半分近い時間をデータ転送に要している。自動並列化と組み合わせた多階層の並列化の検討が必要と考える。

付　　録

参考資料

1. 稲坂純、萩原孝“スーパーコンピュータ SX-9 のハードウェア”東北大学サイバーサイエンスセンター大規模科学計算システム広報 SENAC Vol.41,No.3,2008-7

2. 江川隆輔、大泉健治、伊藤英一、岡部公起、小林広明“新大規模科学計算システムスーパーコンピュータSX-9の性能評価”東北大学サイバーサイエンスセンター大規模科学計算システム広報 SENAC Vol.41,No.2,2008-4

3. 情報基盤課システム管理係、サイバーサイエンスセンタースーパーコンピューティング研究部“スーパーコンピュータシステムSX-9利用ガイド”東北大学サイバーサイエンスセンター大規模科学計算システム広報 SENAC Vol.41,No.2,2008-4

研究論文

4. A.Musa, Y.Sato, R.Egawa, H.Takizawa, K.Okabe, H.Kobayashi, “Early Evaluation of On-Chip Vector Caching for the NEC SX Vector Architecture,” Poster Presentation at SC07, 2007.

5. 佐藤義永、撫佐昭裕、江川隆輔、滝沢寛之、岡部公起、小林広明“ベクトルプロセッサ用キャッシュメモリにおけるMSHRの性能評価”次世代スーパーコンピューティング・シンポジウム 2008,9/16-17,2008.(特別賞受賞)

6. Akihiro Musa, Yoshiei Sato, Takashi Soga, Ryusuke Egawa, Hiroyuki Takizawa, Koki Okabe, and Hiroaki Kobayashi, SC08 Poster Presentation at SC08, 2008.

論文リスト

スーパーコンピュータ SX-9 のハードウェア

稲坂 純　　萩原 孝
日本電気株式会社 コンピュータ事業部

1. はじめに

本年 4 月よりセンターに導入された新しいスーパーコンピュータシステムである SX-9 のハードウェアについて紹介します。SX-9 は、旧システムの SX-7 のアーキテクチャを継承し、単一 CPU 性能を約 12 倍(8.8GFLOPS→102.4GFLOPS)、ノード性能で約 6 倍(281.6GFLOPS→1638.4GFLOPS)に引き上げています。また SX-7 向けに開発したソフトウェア資産を継承するシステムです。ノードあたり 16 台の中央処理装置(Central Processing Unit、CPU)を有し、ノード間を専用の超高速のノード間接続装置(Inter-node Crossbar Switch、IXS)により接続し、システム全体で16ノード、総CPU数256、総合演算性能26.2TFLOPS、総メモリ容量16Tバイトを有する、実効演算性能、スケーラビリティ、及び使いやすさに優れたスーパーコンピュータです。

特長は以下の通りです:

① CPU あたり 100GF を超えるベクトル演算性能
② ノードあたり 1.6TFLOPS の演算性能と 1TB の大容量メモリ
③ 専用のノード間接続装置(IXS)により、ノードあたり 128GB/s×2 のノード間データ転送バンド幅
④ 65nm 銅配線技術を用いた超高速、高集積 CMOS LSI による高密度実装
⑤ 省電力設計による消費電力・発熱量の削減、及び高密度実装による設置面積の削減

本稿では、上記特性を備えた SX-9 のシステム構成(アーキテクチャ)、及びテクノロジの概要について説明します。

2. SX-9 システム構成

SX-9 は、CPU と主記憶装置(Main Memory Unit:MMU)を密結合した共有メモリ型のシングルノード 16 台を超高速なノード間接続装置(IXS)によりクラスタ接続することにより、分散並列処理を可能としたシステムです。SX-9 のシステム諸元を表1、システム構成を図 1 に示します。

表 1 SX-9 システム主要諸元

中央処理装置 (CPU)	CPU 数	256
	ベクトル性能	26.2TFLOPS
主記憶装置 (MMU)	容量	16TB
	データ転送性能	64 TB/s
入出力機構 (IOF)	スロット数 (チャネル数)	128 (512)
	総入出力性能	256 GB/s
ノード間接続装置	データ転送性能	128 GB/s×2／ノード

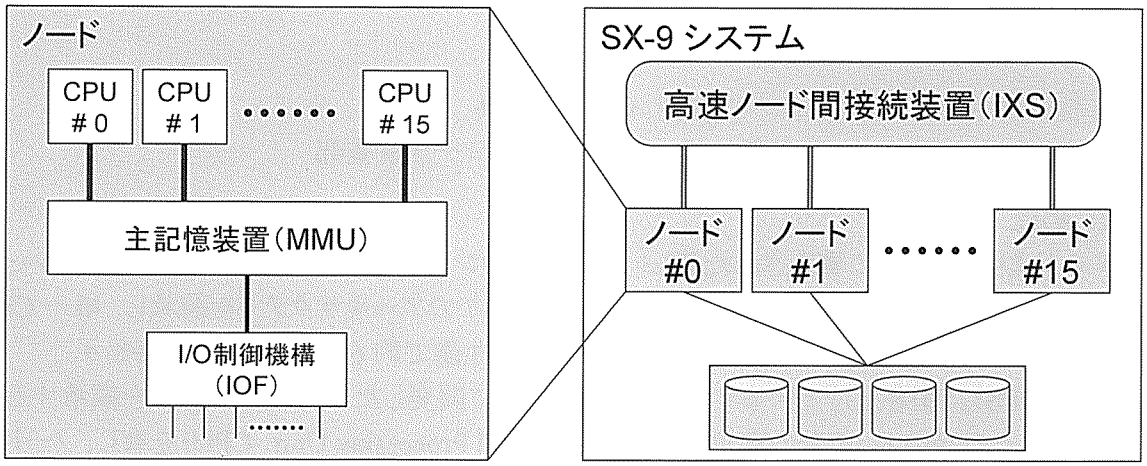


図1　SX-9 システム構成

各ノードは CPU 数 16、総合演算性能 1638.4GFLOPS、主記憶容量 1T バイト、8 スロットの入出力用の PCI スロットを持ち、演算性能、メモリスループット性能、入出力性能などのトータルバランスに優れ、高い実効性能を実現します。また、ノード内の共有メモリを利用した並列処理用に通信レジスタ(Communication Register:CR)を備えており、自動並列処理や、Open MP 指示行による並列処理時の CPU 間同期制御を高速に実行することが可能です。

SX-9 システム全体は、ノード間を超高速に接続する専用の IXS により、16 ノードをクラスタ接続したシステムであり、表 1 に示すように総 CPU 数 256、総合主記憶容量 16T バイト、総入出力チャネル数 256 の構成であり、総合演算性能 26.2TFLOPS という高い演算性能を実現します。

また SX-9 の 1 ノードにおける保守エリアを含む設置面積、及び消費電力はそれぞれ約 $2m^2$、及び約 30KVA であり、設置環境及び性能あたりの消費電力は前システム SX-7 と比較して大幅に改善しています。次節以降、SX-9 システムのハードウェアについて説明します。

中央処理装置(CPU)

SX-9 の CPU は従来の SX アーキテクチャを継承しつつ、さらなる機能・性能の強化を図っています。図 2 に CPU の構成を示します。CPU は、スカラユニット部、及びベクトルユニット部により構成され、プロセッサ／メモリ間ネットワークを介して MMU と接続されます。スカラユニットは命令の解読、ベクトルユニットへのベクトル命令の供給・起動、及びスカラ命令の実行を行います。一方、ベクトルユニットはベクトル演算部、及びベクトル制御部から構成されます。ベクトル演算部は 8 セットのベクトルパイプラインを備え、各ベクトルパイプラインは、乗算器×2、加算／シフト演算器×2、及び除算／平方根演算器×1、論理演算器×1 の 6 種のそれぞれ独立に動作可能な演算パイプライン、マスク演算パイプライン、ロード／ストアパイプライン、マスクレジスタ、及びベクトルレジスタにより構成されています。したがって、科学技術計算で主に利用される乗算と加算は、3.2ns のクロックサイクルあたり 32 個が同時に処理することが可能であり、最大 102.4GFLOPS のベクトル演算性能を実現します。

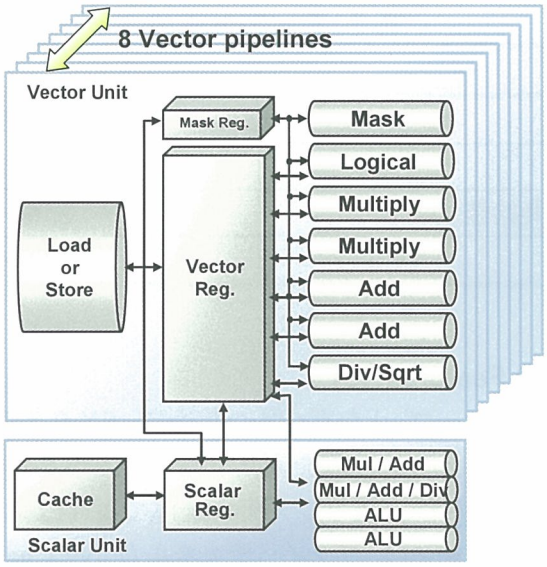


図 2　CPU の内部構成

スカラユニットは 64 ビット RISC アーキテクチャであり、L1 キャッシュ、命令同時デコード数 4、最大命令同時発行数 6 であるスーパスカラアーキテクチャ、アウトオブオーダー実行、及び命令投機実行の採用により短ベクトル時のベクトル命令発行性能、及びスカラ性能を向上させ、3.2GFLOPS の最大演算性能を実現しています。

主記憶装置（MMU）

スーパーコンピュータにおいて高い実効性能を実現するためには、高い演算性能に見合うだけの高いデータ供給性能が必要となります。SX-9 の MMU は、高速、かつ均一にメモリアクセス可能な共有メモリ方式を採用しています。MMU は 32768 個のメモリバンクを 64 バンク毎にメモリバンクを管理する制御部で管理し、その制御部にメモリバンクキャッシュ機構を備えてバンク競合を最小限に抑えつつ、高いメモリスループット性能を実現しています。

これにより SX-9 は表 2 に示すように、シングルノードにおいて、1T バイトのメモリ容量、及び 4T バイト／秒のメモリスループット性能、システム全体では 16T バイトの総メモリ容量、及び 64T バイト／秒の総合メモリスループット性能を実現します。また、

SX-9 は、従来 SX シリーズ同様に 3 次元実装構造の MMU カード実装を引き続き採用しています。これにより、RAM とメモリ制御部間の物理的距離を短くし、RAM の高速動作、及び RAM と CPU 間の高速信号伝送を実現しています。一方、メモリ信頼性確保のための ECC（Error Check and Correct）符号、タイミング、パリティ、2 重化回路などの採用による高いメモリ故障検出率の実現、擬似障害によるチェック回路の診断機能、エラー内容から即座にエラー箇所を指摘するビルトイン機能などにより、RAS（Reliability, Availability, Serviceability）機能の充実を図り信頼性を高めています。

表 2　主記憶装置諸元

	諸元
主記憶装置容量	1T バイト
インターリーブ数	32768
CPU あたりのデータ供給能力	256G バイト／秒
最大データ供給能力	4T バイト／秒

入出力機構(Input/Output Features:IOF)

入出力機構はシステムのスループットを高く保つために、SX-9 の高いプロセッサ性能、及びメモリ性能に見合った高速なデータ転送性能を備えており、ノードあたり 8 スロット(但し、1 スロットはシステム制御用)、16G バイト／秒、SX-9 システム全体では 112 スロット、256G バイト／秒の性能を有します。入出力インタフェースは、ファイバチャネル(Fiber Channel: FC)、シリアル SCSI (Serial Attached SCSI: SAS)などの汎用インタフェースをサポートしており、様々な周辺機器を接続することが可能です。また、ネットワークインタフェースとして、ジャンボフレームに対応したギガビット・イーサネットを備えています。

I/O 処理において、CPU は全ての I/O 装置に対して対等にアクセスすることが可能であり、実行負荷の低い CPU を I/O 制御に割り当てるなど、CPU の効率的利用を可能としています。

ノード間接続装置(IXS)

SX-9 は図 1 で示したように、共有メモリ型のシングルノード 16 台を超高速専用クロスバスイッチを介して結合することにより、分散並列処理を可能としています。各ノードは、ノード間通信制御部(Remote Control Unit: RCU)を介して IXS とケーブル接続され、分散並列処理において低通信レイテンシ、及び高通信スループットを実現します。

RCU のデータ受信部、及び送信部はそれぞれ独立に動作可能であり、ノードあたり 128G バイト／秒×2(双方向)の通信バンド幅を実現します。また、RCU は CPU とは独立に動作するデータムーバを持つことにより、異なるノードのメモリ間でデータ転送を行なうリモートメモリアクセスを CPU 動作とは完全に独立して行なうことが可能です。SX-9 の IXS は、従来の SX シリーズの IXS で採用していた回線交換型から、パケット交換型にデータ交換方式を変更しました。これにより、従来の回線制御オーバーヘッドを削減し、小データサイズの転送性能の向上を図っています。

3. SX-9 のテクノロジ

LSI 技術

これまで SX シリーズでは CMOS テクノロジによる高集積化，及びプロセッサのパラレル化により高性能化を実現しつつ、コストパフォーマンスを向上させてきました。

SX-9 では、さらに高い性能を実現するために LSI 技術及び回路技術を発展させています。また、システムの性能向上のためには、LSI 間信号伝送の高速化も非常に重要です。SX-9 では、新規のマルチチャネル・シリアル・インタフェースを開発し、LSI 間の高速データ転送を実現しています。また、インタフェース回路の低消費電力化、小面積化により LSI への多チャネルの搭載を実現しています。表 3 及び図 3 に SX-9 の CPU LSI 諸元、及び CPU チップ外観をそれぞれ示します。

表3　CPUチップ諸元

テクノロジノード	65nm
搭載トランジスタ数	3億5千万トランジスタ
電源電圧	1.0V
ピン数（内信号ピン）	8,960（1,791）
配線層構成	銅11層
I/Oインタフェース	CML（Current Mode Logic）
実装形態	ベアチップ実装

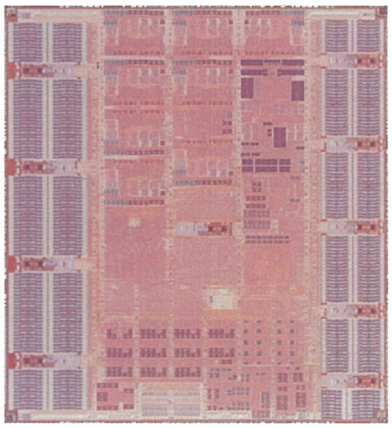


図 3　CPU チップ外観

SX-9 に使われる LSI の共通仕様として、65nm CMOS プロセス、11 層銅配線、及び低誘電率層間絶縁膜などの採用による配線遅延の改善、MIM（Metal-insulator-metal）プロセスの開発による大容量オンチップキャパシタの実現、ゲート酸化膜の薄膜化による高性能な低電圧電源の実現などを行なっています。さらに、新規のマルチチャネル・シリアル・インタフェースの開発により、CPU-MMU 間の低転送レイテンシを実現し、同時に LSI 上の電源分離、アナログ回路の削減、制御信号のデジタル化などによりノイズ耐力及びエラーレートの格段の向上を実現しています。

LSI 内部の RAM 回路は、デバイス性能を最大限引き出すために専用設計されています。また、

LSIの低消費電力化のために、読み出し回路のダイナミックパワーの削減、及びリーク電流の少ないトランジスタの使い分けによるスタティックパワーの削減を実現しています。また、高速なクロック動作を実現するために、LSI外部からクロックを逓倍するAPLL(Analog Phase-Locked Loop)回路を採用しています。

高速システムにおける処理能力の向上には、LSI内信号伝送の高速化とともに、LSI間信号伝送の高速化が必要となります。同様に、信号伝送の高速化を妨げる要因となる電源ノイズ対策も重要です。SX-9では高速、かつ安定した信号伝送を実現するために、信号伝送時の減衰が小さい低損失材料を使用した基板、伝送信号の波形を改善するイコライズ機能を備えた回路、及び波形ひずみの少ないソケットやコネクタなどを採用しています。また、トランジスタが高速化し、電源電流の時間変化が大きくなることにより電源ノイズが増加するため、デカップリング用コンデンサの搭載数最適化などにより電源ノイズの低減を実現しています。

実装技術

SX-9は、世界最高性能、高コストパフォーマンス、及び優れた設置性を実現するために、高密度LSI実装技術、高密度接続技術、高効率冷却技術、及び高性能電源モジュール技術により、従来のワンチッププロセッサをさらに進化させました。

CPU、及びMMUモジュールの諸元を表4に、CPU、及びMMUモジュールの外観を図4にそれぞれ示します。超高速動作が要求されるCPU／MMUモジュールは、高密度実装により大型で多ピンのLSIを搭載可能としています。CPUモジュールは、CPU LSIをビルドアップ基板表面にベアチップ実装し、裏面には高速シリアル信号を伝送するケーブルを接続するための高密度コネクタを搭載しています。MMUモジュールは、メモリ制御用のHUB LSIとSDRAMをプリント配線基板に実装しています。

次にシステム実装技術について述べます。SX-9はルータ(RTR)モジュールと呼ばれるメインボードならびに高周波多芯ケーブルを介して、CPUモジュールとMMUモジュールを相互接続するケーブルインタフェース接続構造を採用しています。これにより、メモリユニットの高密度化とCPUモジュールとMMUモジュール間の効率的な接続を実現することができました。RTRモジュールは図5に示すように32枚のMMUモジュールと2個のルーティングスイッチ LSIを搭載しています。MMUモジュールから送信されたデータは ルーティングスイッチLSIでルーティングされ、高速シリアルインタフェース信号に変換された後、RTRモジュール裏面に搭載されたコネクタを経由し、高周波信号伝送用に開発した多芯ケーブルを介してCPUモジュールに伝送されます。

表 4　CPU／MMU モジュール諸元

項目		CPU モジュール	MMU モジュール
搭載 LSI（形態）		CPU LSI×1 （ベアチップ）	HUB LSI×1 （FBGA）
	ピン数	8960	840
	IO ピッチ（μm）	168	168
搭載 RAM		—	μBGA×24
配線基板	種類	ビルドアップ プリント配線基板	プリント配線基板
	基板サイズ（mm）	140×112.5	110×65
	基板厚（mm）	1.6	1.56
	基板層数	4ビルド－8コア－4ビルド	12
	配線密度（μm）	配線幅／間隙 ＝18／20	配線幅／間隙 ＝80／80
モジュール	入力端子数	2628	170
	冷却	空冷	
	消費電力（W）	240	41

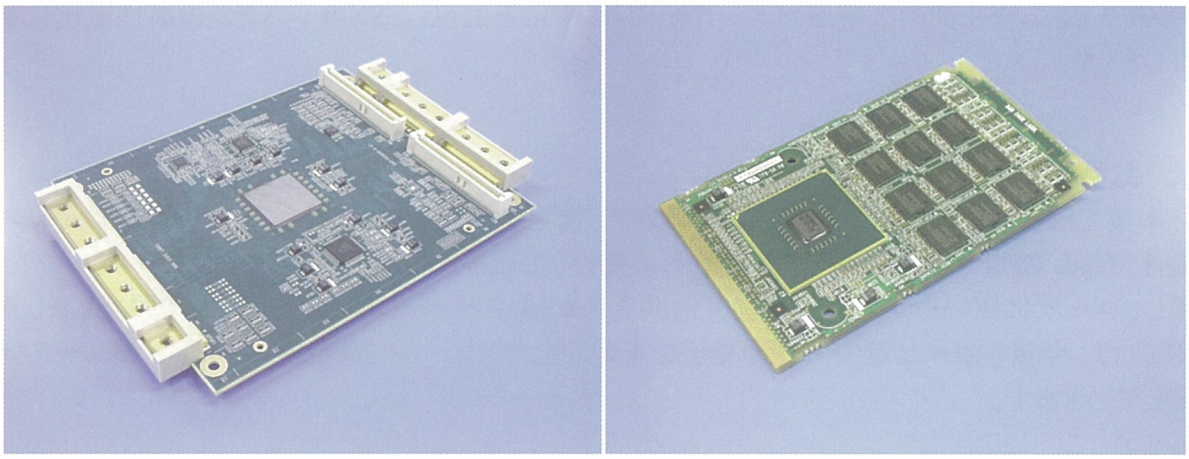


図 4　CPU モジュール、MMU モジュールの外観

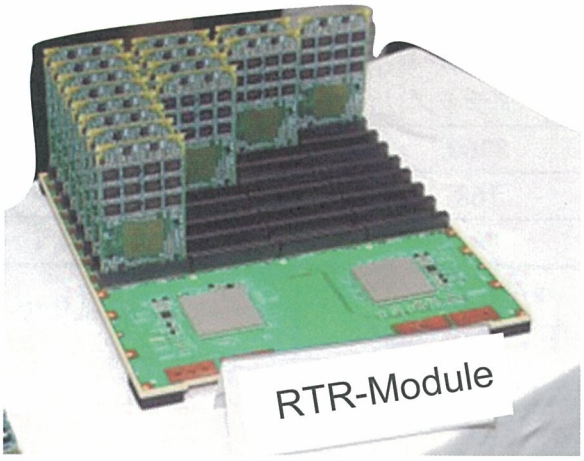


図 5 RTR モジュールの外観

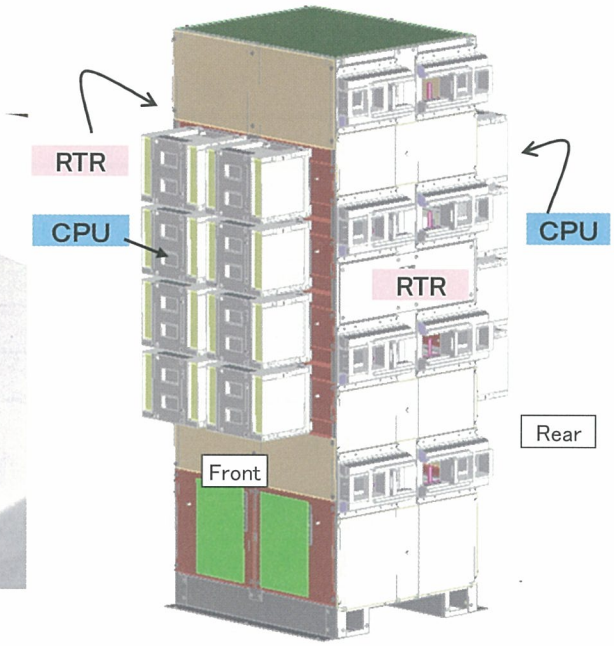


図 6 布線ボックスの外観図

CPU モジュールと RTR モジュール間の高密度多芯ケーブルは図 6 に示す布線ボックスと呼ばれるユニットに収納されています。このケーブルはノード内に搭載される CPU モジュール 16 ユニットと RTR モジュール 16 ユニット間を 1 対 1 で相互に接続することで、CPU チップと MMU モジュール間は独立した専用の高速信号伝送媒体を有することが可能となり、高性能 CPU と大容量のメモリモジュール間の広帯域のデータ転送バンド幅を実現し、SX-9 システムの高性能化に寄与しました。

4. おわりに

本稿では SX-9 のハードウェア概要について説明しました。SX-9 は、スーパーコンピュータの要件である CPU の高い演算性能と、それに見合う主記憶からのデータ供給能力のバランスを重視し、ユーザに使いやすい大規模な分散共有メモリ型スーパーコンピュータとして開発しました。NEC は、今後も様々な研究分野の発展を支える強力なツールであるスーパーコンピュータを開発していきます。

新大規模科学計算システム
スーパーコンピュータ SX-9 の性能評価

江川隆輔* 大泉健治** 伊藤英一** 岡部公起* 小林広明*
*東北大学サイバーサイエンスセンター スーパーコンピューティング研究部
**東北大学情報部情報基盤課システム管理係

本稿では，2008 年 3 月に導入した当センター大規模科学技術計算システムの主力計算機であるベクトル型スーパーコンピュータ SX-9 の概要について説明し，実アプリケーションを用いた新システム SX-9 の性能評価について述べます．評価の結果，スーパーコンピュータ SX-9 システムが従来の SX-7 システムと比較して約 10 倍の高速な処理が可能である事を確認しました．

1. はじめに

東北大学サイバーサイエンスセンター（旧情報シナジーセンター）では，2008 年 3 月に大規模科学計算システムの主力計算システムであるベクトル型スーパーコンピュータ SX-7 を，最新のベクトル型スーパーコンピュータ NEC SX-9 へとリプレースしました．新システム SX-9 の導入に伴い，当センターの大規模科学計算システムは，ベクトル型スーパーコンピュータ SX-9 と SX-7C，並列コンピュータ TX7/i9610 の 3 つのシステムから構成されることとなります．当センターの大規模科学計算システムの構成を図 1 に示します． 今回導入した SX-9 システム（図 2）は 16 ノードからなり，1 ノードあたり 1.6Tflop/s(Tflop/s:1 秒間に 1 兆回の浮動小数点演算)，システム全体では，26.2Tflop/s の高い理論性能を有しています．この新システムに従来の SX-7C,TX-7/i9610 の 640Gflop/s，1.23Tflop/s を加えて，当センターが提供する演算能力は，約 28Tflop/s に達し，ユーザの皆様の幅広いご要望に応えらえる充実した計算機環境を提供します．

以下本稿では，この度導入した新システム SX-9 の概要，SX-9 の実アプリケーションを用いた性能評価，および当センターにおける SX-9 の運用について説明します．

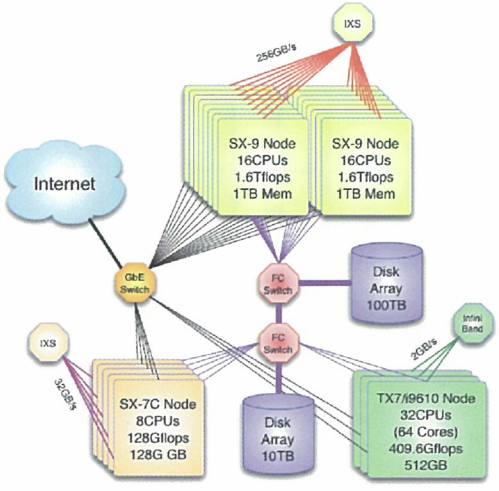


図 1 新大規模科学計算システムの構成．

図 2 SX-9(サイバーサイエンスセンター内)．

2. スーパーコンピュータ SX-9

SX-9 は世界最速の 1 チップベクトルプロセッサを搭載し，1 プロセッサあたり 102.4Gflop/s (1Gflops/s：1 秒間に 10 億回の浮動小数点演算)の高いベクトル演算性能を実現しています．これは半導体加工技術の進歩による動作周波数の向上と大幅なベクトルパイプラインの増強によるものです．SX-9 の CPU は 65nm の CMOS 加工技術を用いて設計されており，3.2GHz という高速な周波数で動作します．また，図 3 に示すように SX-9 の CPU には SX-7 の 2 倍にあたる 8 つのベクトルユニットが搭載されており，各ベクトルユニットの浮動小数点乗算器と浮動小数点加算器が各 2 器，スカラユニットの乗加算器も各 2 器と，SX-7 と比べて大幅な演算能力の増強が行われています．

高い単一 CPU 性能がもたらす効果を確かめるために，図 4 に 100Gflop/s の CPU を 10 個有するシステム(100Gx10)，10Gflop/s の CPU を 100 個有するシステム(10Gx100)の性能と並列化効率の関係を示します．双方のシステムとも理論性能は同じ 1Tflop/s であるにも関わらず，並列化効率が 100%に満たない場合は，常に 100Gx10 システムが高い性能を示します．つまり，高い実効性能を実現するためには，高い並列化効率が必要であり，高い並列化効率を得るためにプログラマは多大なる労力を費やさなければなりません．このことからも，高い単一 CPU 性能を有する SX-9 は，プログラマに与える負荷を極力抑え，高い演算能力の提供できるシステムであると言えるでしょう．

16 個の CPU が搭載される各ノードは，1T バイトのメモリを共有する SMP(Symmetrical Multi Processing)構成をとり，1.6Tflop/s のピーク性能を持っています．各 SMP ノード間は，図 5 に示す高性能なノード間転送装置(IXS：Internode Crossbar Switch)により，双方向で最大 256G バイト/秒の接続が可能となり，大規模，かつ高速な並列計算を可能にします．従来のスーパーコンピュータシステム SX-7 と新スーパーコンピュータ SX-9 の性能を表1に示します．この表からも明らかなように，大幅な CPU 性能の向上，高速な CPU 間の通信性能に基づき SX-9 システムは SX-7 システム に比べて，約 12.5 倍の性能向上を実現しています．次節では，東北大学情報シナジーセンターでこれまで運用されてきた実アプリケーションを用いた SX-9 の性能評価を行い，新スーパーコンピュータシステム SX-9 の有用性を検証します．

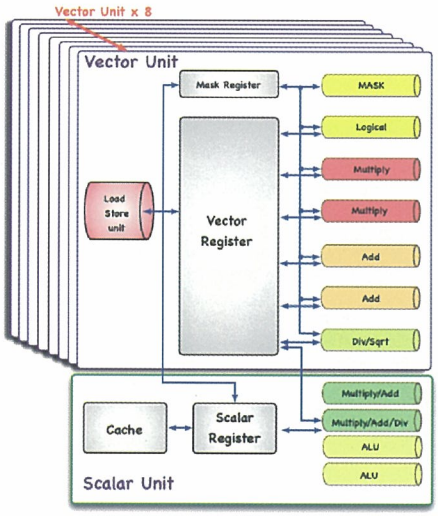


図 3 SX-9 ベクトルユニット．

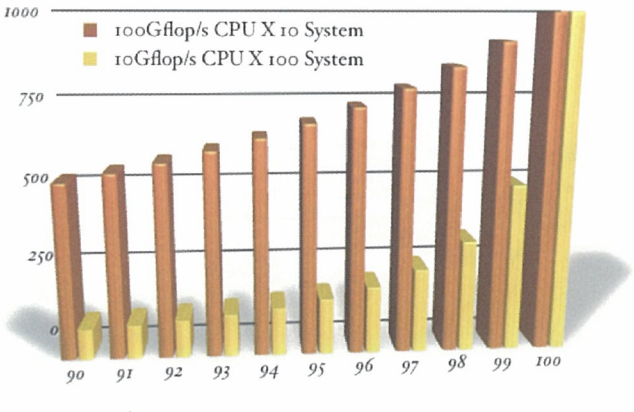


図 4 高い CPU 性能の有効性．

表 1 SX-7 と SX-9 の性能比較.

	項目	SX-7	SX-9	向上比
CPU	動作周波数	0.551GHz	3.2GHz	5.6倍
	ベクトル演算性能	8.83Gflops	102.4Gflop/s	11.6倍
	メモリバンド幅	64GB/s	256GB/s	4.0倍
SMPノード	ベクトル演算性能	282Gflop/s	1.6Tflops	5.8倍
	メモリ容量	256GB	1TB	4.0倍
	メモリバンド幅	1.13TB/s	4TB/s	3.5倍
	メモリバンク	16K	32K	2.0倍
	ノード間通信速度	32GB/s	256GB/s	8.0倍
システム	ベクトル演算性能	2.1Tflops	26.2Tflop/s	12.5倍
	メモリ容量	2TB	16TB	8.0倍

図 5 IXS ユニット.

3. 性能評価

3.1 SX-9 シングル CPU 性能

ここでは, SX-9 のシングル CPU 性能の評価について説明します. 前節では SX-9 の理論性能は, これまでの SX-7 システムと比較して大きく向上していることを述べました. しかし, 計算機本来の有用性を検証するには, 計算機の理論性能だけではなく, 実際にアプリケーションを実行した際に得られる実効性能と併せて評価する必要があります. そこで我々は, これまで, 東北大学情報シナジーセンターにおいて運用実績のある 6 つの実アプリケーションを用いて SX-9 の性能評価を行いました.

はじめに, Intel ItaniumII (開発コード:montecito)を搭載した並列計算機 TX-7/i9610 と SX-7, SX-8, SX-8R, SX-9 の CPU 性能比較を行います. それぞれの CPU の理論性能は ItaniumII:6.4Gflop/s, SX-7:8.83Gflop/s, SX-8:16Gflop/s, SX-8R:35.2Gflop/s, SX-9:102.4Gflop/s になります. また, 評価には Earthquake[1], Turbulent Flow[2], Antenna[3], Land Mine[4], Turbine[5], Plasma[6]の 6 つアプリケーションを用います. 図 6 の評価結果が示すとおり, 全てのアプリケーションにおいて, ベクトル型のプロセッサが ItaniumII を凌ぐ性能を実現していることが分かります. また, SX-9 は, その理論性能値が示すとおり, 既存のベクトル型スーパーコンピュータを大きく凌ぐ演算性能を有していることが, 実アプリケーションを用いた評価により確認することができます.

次に, ItaniumII(motecito)1CPU と SX-9 1CPU, ItaniumII 16CPU と SX-9 1CPU との比較を行います. 図 7, 図 8 は, それぞれ ItaniumII の 1CPU と 16CPU に対する性能比を示しており, SX-9 の1CPU は ItaniumII の 1CPU と比較して最大 154 倍,平均して約 77 倍の実効性能を示しています. ここで, Itanium II と SX-9 の理論性能はそれぞれ 6.4Gflop/s, 102.4Gflop/s と約 16 倍の性能差しかありません. しかし, 今回の評価では, 平均 77 倍と ItaniumII を大きく上回る実効性能を示していることから, ベクトル型のプロセッサが当センターで実行されてきた科学技術計算プログラムに対して, 高い有用性を示している事が確認できます.

また, SX-9 の 1CPU(102.4Gflop/s)と同じ理論性能を持つ ItaniumII の 16CPU(6.4 x 16 = 102.4Gflop/s)と比較した場合も, 図 8 に示すように最大で約 28 倍, 平均で約 10 倍の実効性能を示していることがわかります. これは, 図 4 で説明したように, 高いシングル CPU 性能を持つ計算システムが高速処理の実現に効果的に機能することを, 実アプリケーションを用

いた評価で確認できたと言えます．この評価において，Land Mine は他のアプリケーションに比べて特に高い実効性能比を示しています．これは，Land Mine を ItaniumII で実行した場合，キャッシュヒット率が約 70%と低く，全計算時間の内，約 90%以上をメモリアクセスが占めているためです． このようなメモリインテンシブなプログラムでは，高いメモリバンド幅をもつベクトル型プロセッサは，特にその有効性を発揮します．

これらの結果より，SX-9 ではユーザがこれまで SX-7 で実行してきたアプリケーションを特に変更することなく，容易に 10 倍近い速度で実行可能になることが確認できます．

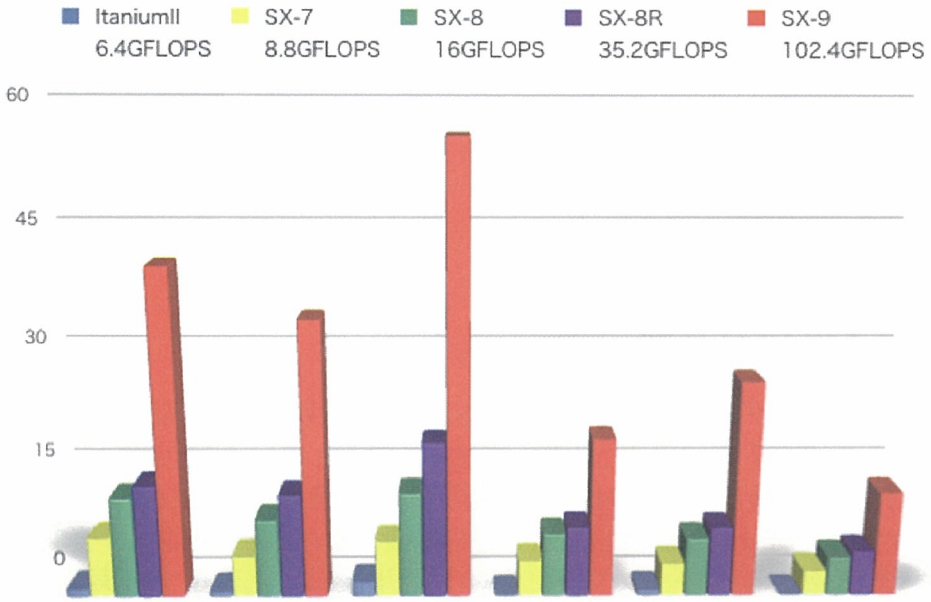


図 6 SX-9 と SX シリーズ，ItaniumII の 1CPU あたりの性能．

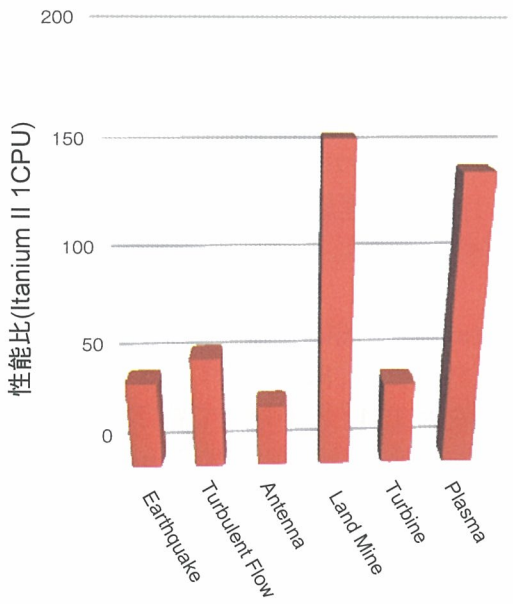


図 7 SX-9 vs. Itanium II.

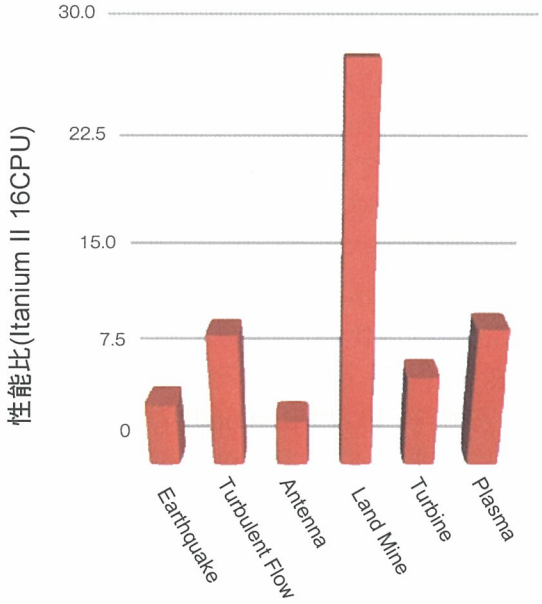


図 8 SX-9 vs. 16 Itanium II.

3.2 シングルノード性能

次に 16CPU からなる SX-9 のシングル SMP ノードの性能を評価します. 評価には前節で用いたベンチマークのうち Antenna を用います. SX-7 は 1 ノード 32CPU で構成されているため, 1CPU,16CPU,32CPU で実行した場合を, 一方, SX-9 は 1 ノードが最大 16CPU で構成されていることから, 1CPU, 8CPU, 16CPU で実行した場合の性能を評価します. 図 9 は, SX-7 の 1CPU における実効性能を 1 とした場合の各 CPU 数における実行時の実効性能との比を表しています. 1 ノードあたりの CPU 数が半分しかないにもかかわらず, SX-9 の 1 ノードの性能は SX-7 の 1 ノードの性能の約 5 倍, 同じ CPU 数の場合は約 10 倍の性能を持つことが確認できます. また, SX-9 のシングルノード内の使用する CPU 数の増加に伴い, SX-7 よりも高い割合で実効性能が向上し, 高いスケーラビリティを有していることもわかります. この結果からも, SX-9 のシングルノードでは SX-7 用に最適化されたアプリケーションを, 容易に SX-9 上で高速に処理することが可能だと言えます.

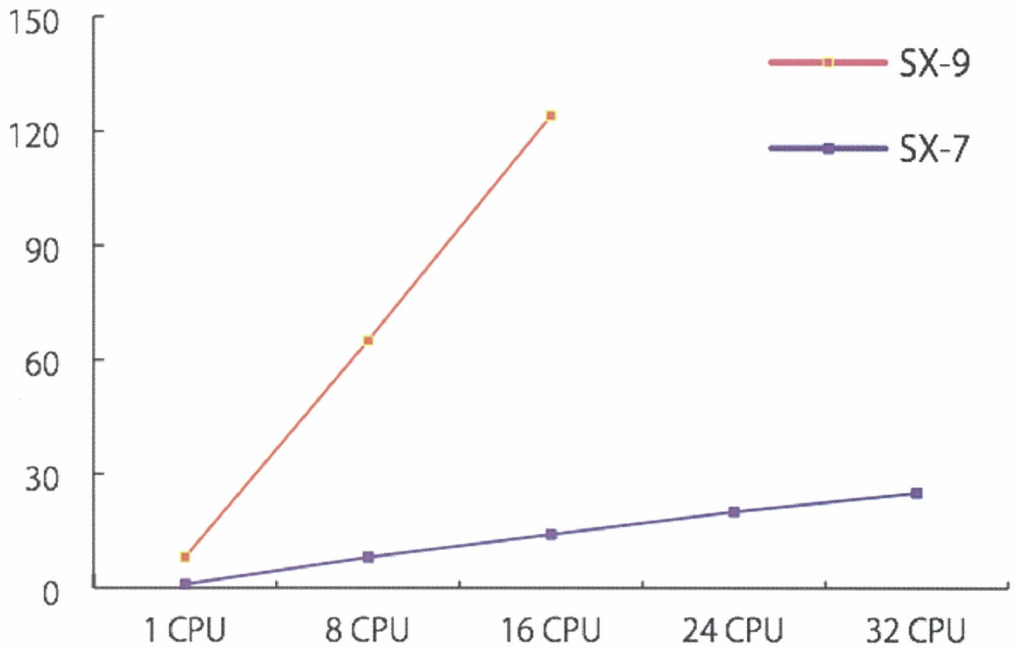


図 9 SX-7 と SX-9 のシングル SMP ノード性能.

3.3 CPU 間通信性能

ここでは, SX-9 の CPU 間通信性能の評価を行います. 複数のノードを跨いだ高速な CPU 間通信は, 複数のノードによる高い実効性能実現の為には必要不可欠です. そこで, CPU 間の通信速度を HPC Challenge ベンチマークで提供されている ping pong カーネルベンチマークを用いて評価します[7]. 図 10 は横軸に代表的な大型計算システムとその CPU 数を示し, 縦軸にはメモリバンド幅を示しています. SX-9 はサイバーサイエンスセンターで導入した 16 ノード, 256CPU の構成です. この結果からも明らかなように SX-9 は他の大型計算システムと比較して, 高速なデータ通信を実現可能なことが確認できます. SX-9 の高い CPU・ノード間通信性能は, 複数ノード・CPU を用いた高速処理の実現を可能にします.

また，当センターでは 4 ノード，64CPU を用いた MPI 並列処理環境の提供を予定しています．4 ノードの MPI 実行環境では表 1 に示した 256GB/s のバンド幅を有する IXS（Internode Crossbar Switch）により，約 6.5Tflop/s の理論性能を持つこととなります．これは，現在我々が SX-7C により提供している MPI 環境よりも，10 倍以上高速な環境になります．

以上，実アプリケーションを用いたSX-9の実効性能の評価により，SX-9は，高いシングルCPU性能と高速なノード間通信に基づき，従来のSX-7システムを大きく凌ぐ実効性能を持つことを示しました．

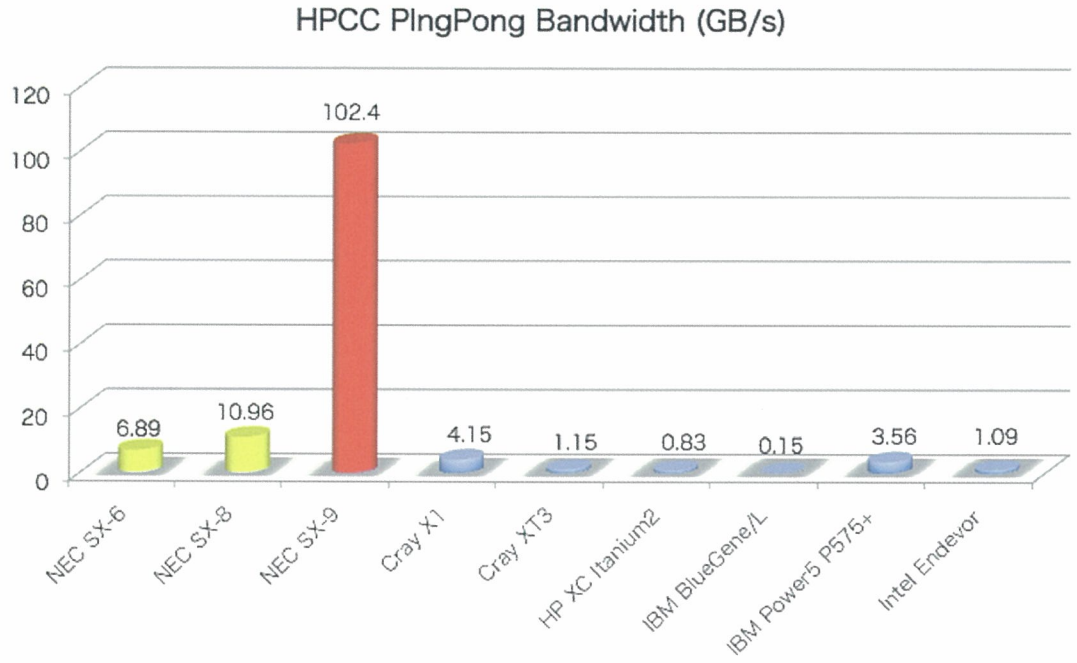


図 10　SX-9 通信性能．

4. SX-9 の運用

SX-9 の一般運用は 2008 年 4 月 4 日の開始を予定しています．また，基本的な運用方針，および環境は SX-7 と同様のものをユーザの皆様に提供することを前提に準備を進めております．プログラミング言語は Fortran と C/C++への対応を準備しており，ユーザの皆様には従来通り自動ベクトル化，自動並列化機能を提供することで，既存のプログラムをそのまま新システムで実行出来る環境を提供します．

また，自動並列化機能に加えて OpenMP や MPI による並列化プログラムの利用可能な環境，複数の job クラスを提供することで，ユーザの皆様には効率のよいシステム利用をして頂きたいと考えております．前節でも触れましたが，将来的には SX-9 の 4 ノード，64CPU を用いた MPI 環境を提供する予定でおります．さらに大規模な MPI プログラミング環境を必要とする場合は、別途特別に用意することも検討いたしますので、どうぞお問い合わせください。また，当センターの大規模科学技術計算システムを有効に活用していただくために，全てのユーザの皆様に対して平成 20 年 4 月 1 日ー平成 21 年 3 月 31 日の期間，利用負担金割引制度（施行）を用意いたしております．特別な申請は一切不要です．

その他，計算機利用，利用可能環境に関しては随時大規模科学計算システムウェブ（http://www.isc.tohoku.ac.jp）において情報を提供しておりますので，是非ご確認ください．

また，ご不明な点，ご質問等ございましたら，お気軽にセンターまでお問い合わせ下さい．

【問い合わせ先】
- 利用相談室　sodan05@isc.tohoku.ac.jp
- システム管理係　022-795-6252

【システム利用申請】
- http://www.cc.tohoku.ac.jp/guide/riyou.html をご覧下さい．
- 窓口（庶務係）　uketuke@isc.tohoku.ac.jp　022-795-3406

5. おわりに

本稿では，2008 年 4 月に運用開始予定のスーパーコンピュータ SX-9 の概要と，SX-9 の性能評価について説明しました．これまで当センターSX-7 で実行されてきた実アプリケーションを用いた性能評価の結果，従来のアプリケーションに特に変更を加えることなく，SX-9 システムは SX-7 システムに比べて約 10 倍の実行性能を実現する事を示しました．2011 年には総性能 10Pflop/s を超える次世代スーパーコンピュータの運用が予定されておりますが、次世代スーパーコンピュータ基本アーキテクチャの 1 つがベクトル並列型となっております。従いまして、次世代スーパーコンピュータへの応用プログラムの拡張を視野に入れながら、この高性能な SX-9 システムを皆様の研究推進の為の強力なツールとしてご活用いただければ幸いです．

※本稿は東北大学サイバーサイエンスセンター大規模科学計算システム広報 SENAC, Vol.41 No.2 2008-4 の原稿に一部加筆訂正を行ったものです．

謝辞

本稿を執筆するにあたり，多くの方々にご協力・ご支援を賜りました．東北大学の長谷川昭教授，澤谷邦男教授，佐藤源之教授，山本悟教授，笹尾泰洋助教，土屋史紀助教，東京理科大の塚原隆裕研究員，海洋研究開発機構海洋工学センターの有吉慶介研究員には評価プログラムを提供して頂きました．また，日本電気(株)の撫佐昭裕様，NEC システムテクノロジーの曽我隆様，塩田和永様,NEC ソフトウェア東北の下村陽一様には性能測定において，多大なる協力をいただきました．皆様に心より感謝申し上げます．

引用文献

[1] Ariyoshi, K., T. Matsuzawa, and A. Hasegawa, “The key frictional parameters controlling spatial variation in the speed of postseismic slip propagation on a sub-duction plate boundary,” *Earth and Planetary Science Letters*, vol. 256, pp.136--146, (2007)

[2] 塚原 隆裕, 岩本 薫, 河村 洋, “平行平板間乱流における流体線の直接数値シミュレーション”, 東北大学情報シナジーセンター大規模科学計算システム広報SENAC, Vol. 39, No. 1, pp. 47-58 (2006).

[3] Takagi, Y., H. Sato, Y. Wagatsuma, K. Mizuno, and K. Sawaya, “Study of high gain and broadband antipodal fermi anetnna with corrugation,” *In proceedings of 2004 International Symposium on Antennas and Propagation, Volume 1,* pp. 69 --27 (2004).

[4] Kobayashi, T., X. Feng, and M. Sato, "Fdtd simulation on array antenna sar-gpr for land mine detection," *In Proceedings of SSR2003: 1st International Symposium on Systems and Human Science*, pp. 279--283, (2003).
[5]笹尾泰洋,堀田翼,山本悟," タービン多段静動翼列流れの大規模並列計算を実現する数値タービンの研究, " 東北大学情報シナジーセンター大規模科学計算システム広報 SENAC, Vol. 40 No.3, pp.15--24, (2007).
[6] 加藤雄人, 小野高幸, 飯島雅英, " 不均質媒質中でのプラズマ波動の伝搬についての計算機実験, " 東北大学情報シナジーセンター大規模科学計算システム広報SENAC, Vol. 37, No.1,pp13−19, (2004).
[7]HPC CHALLENGE HOME Page, http://icl.cs.utk.edu/hpcc/index.html.

スーパーコンピュータシステムSX-9

利用ガイド

情報基盤課　システム管理係
サイバーサイエンスセンター　スーパーコンピューティング研究部

1章◆ はじめに
2章◆ システムの特徴
3章◆ システムの構成
4章◆ プログラミング　~逐次処理、ノード内並列処理~
5章◆ プログラミング　~MPI並列処理~
6章◆ その他
7章◆ おわりに

東北大学

1章　はじめに

2008年3月、本センターはスーパーコンピュータシステムSX-9を導入し、新たなシステム構成で運用を開始しています。本稿は、SX-9システムでのプログラミング利用ガイドとして、プログラムの作成からコンパイル、実行等の使い方をご紹介します。

2章　システムの特徴

ハードウェア

スーパーコンピュータシステムSX-9（日本電気㈱製）は、前システムSX-7と同じくベクトル並列型スーパーコンピュータを継承しています。1ノードあたり16CPU構成で、1CPUあたりの演算性能は100G FLOPS[1]超と大幅に性能向上しています。物理メモリはノードあたり1Tバイトの共有メモリ型、バンド幅は1CPUあたり256Gバイト/秒のデータ転送能力を持ち、大規模かつ高性能な処理を可能にします。本センターでは、全16ノードのシステム構成でSX-9を運用します。

●図1　SX-9　1ノード

●図2　本センターのSX-9システム

[1] Floating point number Operations Per Second（浮動少数点演算性能）1秒あたり100ギガ回、つまり1000億回を超える浮動少数点演算が可能。

●ハードウェアの特徴

世界最速[2]ベクトルプロセッサ
1CPUあたり102.4G FLOPS

大規模な共有メモリ型並列処理
1ノードあたり　16CPU、1Tバイト

圧倒的なデータ転送性能（メモリバンド幅）
1CPUあたり256Gバイト/秒

■ベクトル計算機って？

ベクトル演算を行う専用のハードウェアを持つコンピュータです。ベクトル演算は、ループ中で繰り返し処理されるような配列データの演算に対して、逐次的ではなく一括して演算実行するため高速に演算することができます。

ベクトル計算機は科学技術用の数値計算に適しており、大量のデータを繰り返し処理するような大規模計算に向いていると言われています。本センターでは、流体解析や気象解析、電磁界解析をはじめとする大規模シミュレーションにご利用いただいています。

ソフトウェア

UNIXに準拠したSUPER-UXオペレーティングシステムを継承し、ベクトル処理、並列処理に対応したプログラムも従来どおり演算時間に制限無く実行可能です。また、SX-9に最適化された科学技術用数値演算ライブラリ ASL/SX も引き続きご利用できます。

■プログラミング言語

プログラミング言語はFortranとC/C++を用意しています。それぞれ自動ベクトル化機能、自動並列化機能を有していますので、既存のプログラムを修正することなくベクトル化、並列化することができます。OpenMPやMPIによる並列化プログラムも利用可能です。

■並列化プログラミング

並列化の方法として**表1**の3種類を用意しています。

自動並列化は、単一CPUで動作する逐次処理プログラムを並列化します。プログラムを改めて書き直

[2] 浮動小数点演算性能(2007年10月時点)

す必要はなく、お持ちのプログラムをそのまま利用することができます。コンパイラが並列化可能な箇所を自動的に判断し並列化します。

OpenMP は、自動並列化と同じく逐次処理プログラムを並列化します。ただ、並列化の判断は自動ではなくユーザが行います。ソース中の並列化したい箇所に並列化指示行を追加します。

MPI は、メッセージ交換用ライブラリによりプロセッサ間通信を行います。データの分割、処理方法等の並列処理手順を明示的に記述した MPI プログラムを作成する必要があります。

一般的に、共有メモリのノード内では自動並列化あるいは OpenMP による並列処理を、分散メモリのノード間では MPI による並列処理という手段を用います。したがって、自動並列化や OpenMP は 16 並列までのノード内並列処理ができます。それ以上の並列数で実行する場合は、複数のノードを必要とするため MPI による並列処理を行います。4 ノード使用する 64 並列と 2 ノード使用する 32 並列は、MPI 並列処理用として用意しています。

自動並列化と OpenMP によるノード内並列処理手順は 4 章で、MPI による並列処理手順は 5 章で、それぞれ紹介します。

●表1　並列化の種類、特徴、並列数

	逐次処理プログラムの並列化	並列化の指示	最大並列数
自動並列化	そのままで可能。	自動（コンパイラ）	16 並列ノード内
OpenMP	指示行を追記することで可能。	手動（ユーザ）	16 並列ノード内
MPI	できない。MPI 用プログラムを作成する。	通信ライブラリを用いプログラミングする。（ユーザ）	64 並列4 ノード

■互換性

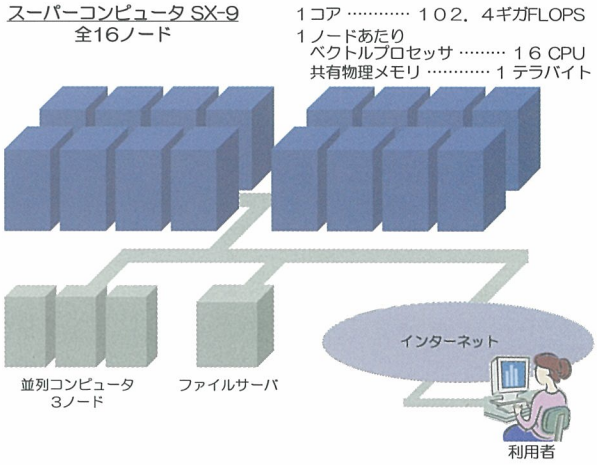
前システム SX-7 とプログラムの互換性はありますが、SX-9 システムで再コンパイルしてから実行してください。SX-9 の性能を十分に引き出すために、最適化機能が強化された SX-9 用コンパイラで再コンパイルします。

3章　システムの構成

本センターでは、SX-9 システム 16 ノードと既設の並列コンピュータシステム（TX7/i9610）の 2 つのシステムをサービスしています。

並列コンピュータシステムは、従来どおり運用し利用方法にも変更はありません。

●図3　システム構成図



4章　プログラミング
～ 逐次処理、ノード内並列処理 ～

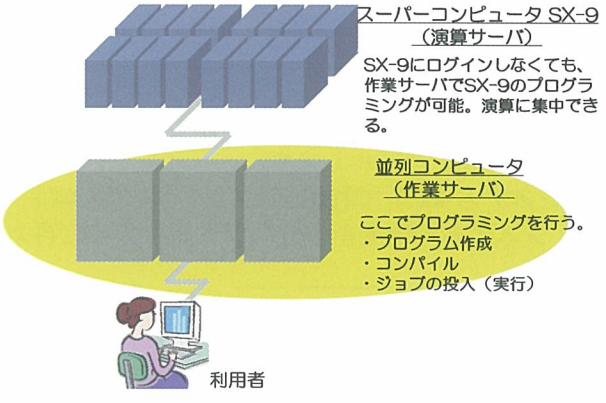
本章では、従来の単一 CPU で実行する逐次処理と、自動並列化および OpenMP によるノード内並列処理について利用手順を紹介します。

MPI による並列化プログラミング手順については、次章で紹介しますが、コンパイルコマンド、ジョブクラス等に違いがある以外は、同じ手順ですのでこの章と合わせてご覧ください。

4-1 フロントエンドサーバ（作業サーバ）

SX-9 用のプログラミングは、既設の並列コンピュータ上で全ての作業を行えるようにしています。ソースファイル作成、コンパイル、実行等一連のプログラミング作業が可能です。SX-9 システムのパフォーマンスをプログラム演算に専念させるために、作業サーバと演算サーバの 2 段構成にしています。

●図4　作業サーバと演算サーバ



4-2 ログイン

作業を行うため並列コンピュータにログインします。リモート接続は、ssh コマンドまたは SSH 対応リモート接続ソフト[3]をご利用ください。

並列コンピュータの OS は Linux です。ログイン名とパスワードを入力することでログインすることができます。アカウント希望の場合は、受付（庶務係）に利用申請し利用者番号と初期パスワードを発行してもらいます。

並列コンピュータホスト名

```
gen.isc.tohoku.ac.jp
```

●リスト 1　ssh コマンドによる接続例

```
yourhost$ ssh gen.isc.tohoku.ac.jp -l 利用者番号
Password: パスワード
```

4-3 ホームディレクトリ

ホームディレクトリは、プログラムファイル等を置く自分専用のディスク領域です。ディレクトリ名は、/uhome/利用者番号　です。容量の制限は特に設けていませんが、利用サイズに応じてファイル負担経費が発生します。

なお、ホームディレクトリはスーパーコンピュータシステムと並列コンピュータシステムで共有しています。

ホームディレクトリ

```
/uhome/利用者番号
```

4-4 ソースファイル作成

ソースファイルを作成するためのエディタは、emacs か vi をご利用ください。

研究室等のサーバ、パソコンから転送する場合は、SSH 対応のファイル転送ソフト[4]で並列コンピュータに転送してください。この際、ソース（テキスト）ファイルは ASCII モードで転送します。

4-5 コンパイラ

SX-9 用に最適化された コンパイラ Fortran および C/C++を用意しています。両言語とも並列コンピュータ上でコンパイルします。

前システム SX-7 で利用していたプログラムは、SX-9 システムの性能を十分に引き出すために再コンパイルしてから実行してください。

Fortran コンパイラの仕様

Fortran90/SX（日本電気）	ISO/IEC 1539-1:1997 自動ベクトル化、自動並列化、OpenMP 対応

C/C++コンパイラの仕様

C++/SX（日本電気）	ISO/IEC 14882:1998 自動ベクトル化、自動並列化、OpenMP 対応

4-6 コンパイルする

通常のコンパイルと同様に、それぞれの言語用コマンドでコンパイルします。この項目では単一 CPU で実行する逐次処理と、自動並列化または OpenMP による並列処理、それぞれのコンパイル手順について解説します。

■Fortran プログラムのコンパイル

sxf90 コマンドでコンパイルします。利用したい機能があれば適当なオプションと、Fortran プログラムソースファイル名を指定します。

ソースファイルの拡張子は、自由形式（フリーフォーマット）なら.f90 か.F90、固定形式（7 カラム目から記述）なら.f か.F を付けます（表 3）。

並列化コンパイルは、ここで並列数を意識する必要はありません。実行する時点 **4-8-3 ジョブを投入** で希望する並列数を選択することができます。

コンパイル【逐次処理】

```
sxf90 オプション ソースファイル名
```

コンパイル【自動並列化】

```
sxf90 -Pauto オプション ソースファイル名
```

OpenMP プログラムなら、-Pauto の箇所を -Popenmp にします。

表 2 は、主なオプションです。詳細は、man コマンド man sxf90 でご覧ください。

●表2　主なオプション

-Pauto	自動並列化機能を利用する。
-Popenmp	OpenMP 対応機能を利用する。
-Wl,-h 4T_memlayout	512G バイト以上のメモリを使用する。
-R5	ベクトル化／並列化状況を表示した編集リストの出力
-ftrace	簡易性能解析機能を利用する
-eC	配列要素参照時、添字の値が範囲内であるか検査する（バウンズチェック）。

[3] Windows であれば、TeraTerm(TTSSH), Putty 等のフリーソフトがあります。

[4] Windows であれば、WinSCP 等のフリーソフトがあります。

●表3　拡張子とフォーマット

拡張子	フォーマット
f90, F90	自由形式
f, F	固定形式

■C プログラムのコンパイル

sxcc コマンドでコンパイルします。利用したい機能があれば適当なオプションと、C プログラムソースファイル名を指定します。

並列化コンパイルは、ここで並列数を意識する必要はありません。実行する時点 **4-8-3 ジョブを投入** で希望する並列数を選択することができます。

コンパイル【逐次処理】

```
sxcc　オプション　ソースファイル名
```

コンパイル【自動並列化】

```
sxcc -Pauto オプション　ソースファイル名
```

OpenMP プログラムなら、-Pauto の箇所を -Popenmp にします。

表 4 は、主なオプションです。詳細は、man コマンド man sxcc でご覧ください。

●表4　主なオプション

-Pauto	自動並列化機能を利用する。
-Popenmp	OpenMP 対応機能を利用する。
-size_t64	4G バイト以上の配列、構造体を利用する。
-Wl,-h,4T_memlayout	512G バイト以上のメモリを使用する。
-ftrace	簡易性能解析機能を利用する。

4-7 プログラムを実行する

コンパイルして作成された実行形式ファイルを実行するには、バッチ処理と会話型処理の 2 通りの方法があります。通常はバッチ処理で実行します。

■バッチ処理

バッチ処理は、実行の手続きをジョブという単位でジョブ管理システムに登録し一括に処理します。ジョブ管理システムは NQS Ⅱ (Network Queuing System Ⅱ)を用意しており、ジョブの操作は NQS Ⅱ のコマンドを用い行います。通常のプログラム（長時間実行するプログラム、並列実行するプログラム等）はバッチ処理で実行します。

■会話型処理

会話型処理は、コマンドラインでプログラムを実行する形式です。演算 CPU 時間や使用できるメモリサイズに制限がありますので、短時間の演算やデバッグ作業にご利用ください。負担金の単価はバッチ処理に比べ割高に設定しています。SX-9 システムに直接ログインして実行します。

4-8 バッチ処理で実行する

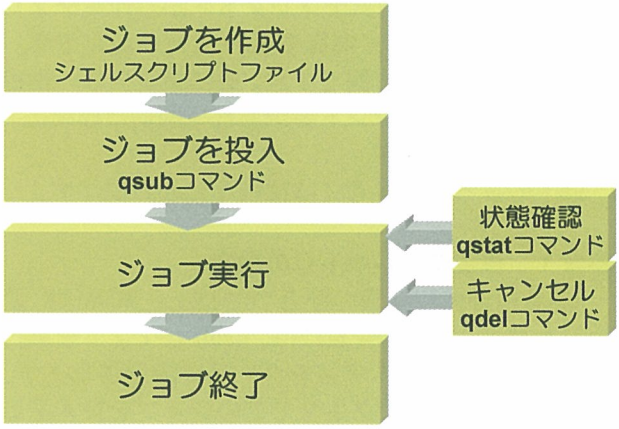
バッチ処理は並列コンピュータ上で作業を行います。

■4-8-1 バッチ処理の概要

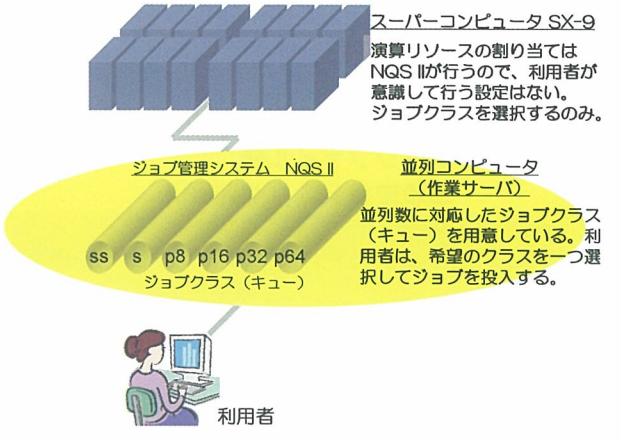
NQS Ⅱにプログラムの実行を依頼するため、実行の手続きを書いたジョブ（バッチリクエスト）を作成します。このジョブを NQS Ⅱに投入することで、プログラムの実行が可能になります。この際、NQS Ⅱではジョブクラスと呼ばれるキューが並列数に応じて設定してあり、利用者は目的のジョブクラスを１つ選択し投入します。順番が回ってくると NQS Ⅱは自動的にジョブを実行します。

ジョブ投入後は、ジョブの状態や込み具合の確認、また投入済みのジョブをキャンセルすることも可能です。プログラムの実行が終了するとジョブは NQS Ⅱの情報から消え、標準出力ファイルと標準エラー出力ファイルが出力されます。

●図 5　ジョブ（バッチリクエスト）の流れ



●図6　ジョブクラスの構成



■4-8-2 ジョブを作成

ジョブ投入用のシェルスクリプトファイルを作成します。プログラムの実行手続きを、通常のシェルスクリプトと同じ形式で記述します。csh スクリプトと sh スクリプト、どちらでも記述できます(以降、解説は csh スクリプトで記述します)。適当なファイル名を付け作成します。

リスト 2 はジョブファイルの一例で、ホームディレクトリ直下 work ディレクトリの a.out を実行する手続きを記述しています。

●リスト 2　ジョブファイル例

```
# test job.2              コメント行
cd work                   作業ディレクトリへ移動
a.out                     実行形式ファイル名
```

1行目：# が先頭の行は、コメント行です。動作には影響しません。

2行目：cd work で作業ディレクトリ（実行形式ファイルのあるディレクトリ）へ移動します。省略するとホームディレクトリを指定したことになります。

3行目：a.out はコンパイルして作成した実行形式ファイル名です。あらかじめコンパイルし作成しておきます。自動並列や OpenMP による並列処理も、同じ形式で指定します。

基本的には、ホームディレクトリから作業ディレクトリへ移動、そして実行する、の2行の設定です。これは最低限の設定例ですが、他に環境変数の指定、ファイルの操作コマンド等があれば適切な箇所に手続きを記述します。

以下の参考1と参考2は便利な記述方法です。

参考1：作業ディレクトリの指定

NQSⅡ用の環境変数のひとつに PBS_O_WORKDIR 変数があります。この変数には、qsub コマンドを実行した時点のカレントディレクトリが設定されます。つまり、work ディレクトリでこのジョブを投入する（qsub を実行する）と、$PBS_O_WORKDIR にはカレントディレクトリの work が設定され cd work と同じことになります。PBS_O_WORKDIR 変数を設定することで、ディレクトリの具体名を記述する必要がなくなります。

●リスト 3　ジョブファイル例
環境変数 PBS_O_WORKDIR

```
# test job.3
cd $PBS_O_WORKDIR         ディレクトリを環境変数で指定
a.out
```

参考2：データファイルの指定

Fortran プログラムのファイル指定方法です。通常、ソース中 OPEN 文でファイル装置番号にファイル名を割り当てますが、環境変数 F_FF*nn* でファイル名を割り当てることもできます。(*nn* はファイル装置番号)

●リスト 4　ジョブファイル例
環境変数 F_FF*nn*

```
# test job.4
setenv F_FF11 datafile    装置番号 11 に、datafile を割り当てる。
cd $PBS_O_WORKDIR
a.out < infile > outfile
```

■4-8-3 ジョブを投入

プログラムの実行は、作成したジョブを NQSⅡに投入することで行います。

```
qsub オプション ジョブファイル
```

利用したい機能があれば、**表 5** に代表されるオプションを指定します。ここで、必ず指定するオプション-q があります。-q の後にジョブクラス名を指定することで、逐次処理または並列処理かを利用者が選択できるようになっております。本センターの SX-9 システムは、**表 6** のジョブクラスを用意しております。

●表5　主なオプション

-q　(必須)	投入するジョブクラス名を指定します。
-N	ジョブ名（リクエスト名）を指定します。指定がなければ、ジョブファイル名がリクエスト名になります。
-o	標準出力のファイル名を指定します。指定がなければ、ジョブ投入時のディレクトリに「ジョブ名.o リクエスト ID」のファイル名で出力されます。
-e	標準エラー出力のファイル名を指定します。指定がなければ、ジョブ投入時のディレクトリに「ジョブ名.e リクエスト ID」のファイル名で出力されます。
-jo	標準エラー出力を標準出力と同じファイルへ出力します。

-l cputim_job=hh:mm:ss	実行打ち切りの CPU 時間を指定します。設定時間は、時：分：秒を hh:mm:ss の形式で指定します。この指定がなければ無制限となります。並列処理で実行するときは、各プロセスの合計 CPU 時間を指定します。
-m b	ジョブの実行が開始したときにメールが送られます。
-m e	ジョブの実行が終了したときにメールが送られます。
-M メールアドレス	ジョブに関するメールの送信先を指定します。指定がなければ、「利用者番号@gen.isc.tohoku.ac.jp」宛に送られます。

詳細は、man コマンド manqsub でご覧ください。

●表 6　ジョブクラス

ジョブクラス	利用可能 CPU 台数	経過 CPU 時間	メモリサイズ制限 (GBytes)
ss	4	1 時間	256
s	4	無制限	32
p8	8	無制限	512
p16	16	無制限	1024

単一 CPU で実行する逐次処理プログラムは s か ss クラスに投入します。ss クラスは時間制限がありますので、経過 CPU 時間が 1 時間以内の短いプログラムを投入します。並列化プログラムは並列数に応じて p8, p16 のいずれかに投入します。並列用のキュー(p8, p16)を利用するには、並列用のオブジェクト[5]を作成しておく必要があります。

s と ss クラスは逐次処理用のクラスですが、4 並列までの実行も可能です。これは並列化プログラムのデバッグを想定しており、特に ss クラスはメモリサイズを 256G バイトと大きく設定しています。

ジョブが正常に投入されると、システムからリスト 5 のメッセージが返ります。この例では 1234.job がリクエスト ID で、ジョブの状況確認やキャンセル等、ジョブの操作の際に必要になります。

●リスト 5　qsub コマンド実行時のメッセージ例

```
gen$ qsub -q p8 job-a
Request 1234.job submitted to queue : p8.
```

[5] 自動並列化なら-Pauto オプション、OpenMP なら-Popenmp オプションを付けてコンパイルしていること。

参考：オプションの埋め込み

毎回同じオプションを記述するのが面倒なとき、ジョブファイルに記述しておく方法もあります。

設定方法は、最初のコマンドより前の行に、#PBS という文字列を先頭に指定します。#PBS の後に空白を一文字以上入れ、指定したいオプションを続けます。一行に複数のオプション指定も可能です。リスト 6 の例は、2 行目で p8 クラスのキューを指定(-q)、3 行目で標準エラー出力を標準出力ファイルにひとまとめにする(-jo)、さらにジョブ名を reqname とする(-N)を、それぞれ指定しています。

また、埋め込みオプションとコマンド列に同じオプションを指定した場合は、コマンド列の方を有効とします。

●リスト 6　オプションの埋め込み

```
# test job.6
#PBS -q p8                    埋め込みオプション
#PBS -jo -N reqname           (複数)
cd $PBS_O_WORKDIR
a.out
```

■4-8-4 ジョブの状態確認

その１．自分のジョブ状態を確認する

```
qstat
```

qstat コマンドは、投入されたジョブの状態を表示します（リスト 7)。状態は STT（ステータス）項目に表示され、たいていは実行中(RUN)か実行待ち(QUE)のどちらかの状態になります。実行中の場合は、さらに CPU 時間、使用メモリサイズ等の情報が表示されます。また、リソースに空きがなければ実行待ち状態になり、順番が回ってくると自動的に実行状態に入ります。待ち状態中は、リクエスト ID の前に待ち順を表示します。システム内に自分のジョブが存在しない場合は、ヘッダのみ表示されます（リスト 8)。

その２．システム全体のジョブ状態を確認する

```
qstat -Q
```

-Q オプションで、システム全体の情報を表示します。各ジョブクラスの件数が表示されますので、混雑具合いがわかります（リスト 9)。

●リスト 7　qstat コマンド例

```
gen$ qstat
RequestID      ReqName  UserName Queue    Pri STT S  Memory  ACCPU Elapse R H M Jobs
-------------- -------- -------- -------  --- --- - -------  ----  ------ - - - ----
   353.job       reqname  x2***9  p8        0  RUN  -  732.1B 42350 43012   Y Y Y   1
 2:359.job       jobA     x2***9  p16       0  QUE  -    0.0B  0.0     0   Y Y Y   1
 5:366.job       jobB     x2***9  p16       0  QUE  -    0.0B  0.0     0   Y Y Y   1
```

主な項目

RequestID	リクエスト ID 待ち状態(QUE)のリクエストは、先頭に待ち順の番号がつきます。
ReqName	ジョブ（リクエスト）名
UserName	ジョブの所有者
Queue	ジョブクラス（キュー）名
STT	ステータス　(QUE：待ち、RUN：実行中)
Memory	使用メモリサイズ (Byte)
ACCPU	演算時間(sec)／並列演算の場合、使用 CPU の総演算時間 (sec)
Elapse	経過時間　(sec)

●リスト 8　投入したジョブがない場合（ヘッダのみ）

```
gen$ qstat
RequestID      ReqName  UserName Queue    Pri STT S  Memory  ACCPU Elapse R H M Jobs
-------------- -------- -------- -------  --- --- - -------  ----  ------ - - - ----
```

●リスト 9　-Q オプションの例(一部抜粋)

```
gen$ qstat -Q
[EXECUTION QUEUE] Batch Server Host: job
=======================================
QueueName       SCH JSVs ENA STS  PRI  TOT ARR WAI QUE PRR RUN POR EXT HLD HOL RST SUS MIG STG CHK
--------------- --- ---- ------ ---- ---------------------------------------------------------------
a16               0    0 ENA ACT   32    0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
a32               0    0 ENA ACT   32    0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
a64               0    0 ENA ACT   32    0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
a8                0    0 ENA ACT   32    0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
am                0    0 ENA ACT   32    0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
as                0    0 ENA ACT   32    0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
p16               1    4 ENA ACT   32    1   0   0   0   0   1   0   0   0   0   0   0   0   0   0
p32               1    4 ENA ACT   32    0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
p64               1    2 ENA ACT   32    4   0   0   2   0   2   0   0   0   0   0   0   0   0   0
p8                1    0 ENA ACT   32    0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
s                 0    1 ENA ACT   32    0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
ss                0    1 ENA ACT   32    0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
--------------- --- ---- ------ ---- ---------------------------------------------------------------
 <TOTAL>                             5   0   0   2   0   3   0   0   0   0   0   0   0   0   0
--------------- --- ---- ------ ---- ---------------------------------------------------------------
```

主な項目

QueueName	ジョブクラス（キュー）名
TOT	ジョブ（リクエスト）の総数
QUE	待ちの件数
RUN	実行中の件数

■4-8-5 ジョブのキャンセル

投入済みのジョブを取り消します。

```
qdel リクエスト ID
```

●リスト 10　qdel コマンドの例

```
gen$ qdel 1234.job
Request 1234.job was deleted.
```

4-9　会話型処理で実行する

会話型処理は、短時間の演算やデバッグ作業に使用します。並列処理も 4 並列まで実行可能です。

手順は、一般的な UNIX を利用するように SX-9 にログインしコマンドラインで実行形式ファイル名を入力します（**リスト 11**）。CPU 時間と使用メモリサイズに制限があり、課金単価はバッチ処理より割高に設定しています（**表 7**）。

ホスト名

```
super.isc.tohoku.ac.jp
```

●リスト 11　会話型処理の例

```
yourhost$ ssh super.isc.tohoku.ac.jp -l 利用者番号
（パスワードを入力しログインする）

super$ a.out     実行
```

●表 7　会話型処理の制限値

演算 CPU 時間	1 時間
メモリサイズ	8 ギガ Bytes
利用可能 CPU 数	4

並列処理の場合、演算 CPU 時間は総 CPU 時間を対象にします。

5章　プログラミング
～ MPI 並列処理 ～

本章では、MPI による並列化プログラミング手順について紹介します。基本的な手順は前章と同じですので、コンパイルコマンド、利用できるジョブクラス等の異なる点について紹介します。

5-1 コンパイルする

MPI プログラムは、MPI 用コマンド sxmpif90 や sxmpicc コマンドでコンパイルします。

■MPI 並列 Fortran プログラムのコンパイル

sxmpif90 コマンドでコンパイルします。利用したい機能があれば適当なオプションと、Fortran ソースファイル名を指定します。

MPI 並列 Fortran プログラムをコンパイル

```
sxmpif90 オプション　ソースファイル名
```

オプションは、sxf90 コマンドと共通です。man sxf90 でご覧ください。

■MPI 並列 C プログラムのコンパイル

sxmpicc コマンドでコンパイルします。利用したい機能があれば適当なオプションと、C ソースファイル名を指定します。

MPI 並列 C プログラムをコンパイル

```
sxmpicc オプション　ソースファイル名
```

オプションは、sxcc コマンドと共通です。man sxcc でご覧ください。

5-2 バッチ処理で実行する

MPI プログラムの実行は、mpirun コマンドを用いますのでジョブファイルにもそのように記述します。

■ジョブを作成

MPI プログラムの実行コマンド

```
mpirun オプション　実行形式ファイル名
```

●表8　主なオプション

-np	MPI 並列数
-nn	使用ノード数

●リスト 12　ジョブファイルの例
64MPI 並列

```
# test job.12
cd $PBS_O_WORKDIR
mpirun -np 64 -nn 4 a.out    64 並列で実行
```

64MPI 並列より少ない並列数の場合、表 9 のオプションになります。-nn オプションを使用ノード数に合わせ設定します。

●表9　MPI 並列数とオプション

8 並列	mpirun -np 8 -nn 1 a.out
16 並列	mpirun -np 16 -nn 1 a.out
32 並列	mpirun -np 32 -nn 2 a.out
64 並列	mpirun -np 64 -nn 4 a.out

8 並列と 16 並列の場合、つまり使用ノード数が 1 のとき -nn 1 は省略できます。

■ジョブクラス

MPI プログラムは最大 64 並列まで実行可能です。64 並列は 4 ノード、32 並列は 2 ノードを占有して実行します。ノード内並列化プログラムと同様に 16 並列、8 並列も実行できます。ご希望の並列数のジョブクラスに投入します。

●表10　MPI プログラムのジョブクラス

ジョブクラス	利用可能 CPU 台数	経過 CPU 時間	メモリサイズ制限 (GBytes)
ss	4	1 時間	256
s	4	無制限	32
p8	8	無制限	512
p16	16	無制限	1024
p32	32	無制限	1024×2
p64	64	無制限	1024×4

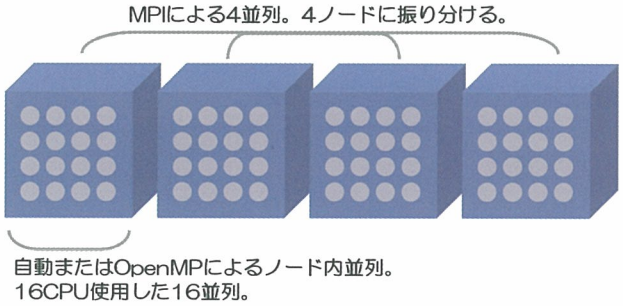
p64 クラス、p32 クラスのメモリサイズは、1 ノード(16CPU)当たり 1024G バイトに収まるようにしてください。その他、プログラムの実行手順は **4-8 バッチ処理で実行する** と同じです。

5-3 ハイブリッド並列

MPI と自動並列、または MPI と OpenMP を組み合わせたハイブリッド並列と呼ばれる並列化手法もあります (**図 7**)。通常、ノード間は MPI 並列化しノード内は自動並列化や OpenMP による並列化する形をとります。

例えば、64CPU を使ったハイブリッド並列を考えると、使用するノード数にあわせ 4 並列で動作する MPI プログラムを作成します。さらに、自動並列化オプション-Pauto をつけコンパイルすると、MPI と自動並列のハイブリッド並列用オブジェクトが作成されます。ノード間は MPI の 4 並列、ノード内は自動並列化の 16 並列、計 64 並列で実行できます。実行時のオプションは、-np で MPI 並列数の 4 を、- nn で使用ノード数 4 を指定します (**リスト 13**)。

●図 7　ハイブリッド並列　64 並列の例



MPI 4並列 × ノード内 16並列 ＝ 64並列

ハイブリッド並列化コンパイル(MPI+自動並列)

sxmpif90 -Pauto オプション ソースファイル名

ノード内が OpenMP 並列なら、-Popenmp オプションを使います。

ハイブリッド並列の実行コマンド

mpirun -np 総プロセス数 -nn 使用ノード数　実行形式ファイル名

●リスト 13　ジョブファイルの例
ハイブリッド並列

```
# test job.13
cd $PBS_O_WORKDIR
mpirun -np 4 -nn 4 a.out    4 ノード利用の
                            64 並列で実行
```

5-4 環境変数の設定 (p64, p32 クラス)

p64 または p32 クラスの実行で環境変数の設定を行う場合、環境変数 MPIEXPORT により各ノードに設定を反映させる必要があります。MPIEXPORT の設定がないと各 MPI プロセスに環境変数が反映されません。設定は、利用する環境変数名を MPIEXPORT の後に記述します(**リスト 14**)。

●リスト 14　環境変数 MPIEXPORT の指定例

```
# test job.14
#PBS -q p64
#PBS -jo -N reqname
setenv F_FF11 datafile
setenv F_FF12 datafile-B
setenv F_UFMTENDIAN  30,40
setenv MPIEXPORT        (次行へ続いています)
   "F_FF11 F_F12 F_UFMTENDIAN"

cd $PBS_O_WORKDIR
mpirun -np 64 -nn 4 a.out
```

" "（ダブルクォーテーション）内に、空白で区切って記述します。

5-5 標準入出力ファイルを指定する

標準入出力ファイルは通常リダイレクション記号（<,>）で指定しますが、mpirun による MPI プログラムの実行では利用できません。

× 利用できません

```
mpirun -np 64 a.out < infile > outfile
```

この場合、標準入力ファイル指定は環境変数 F_FF05 で行います（**リスト 15**）。

標準出力および標準エラー出力は、MPI 並列数分複数出力されます。同一ファイルに入り混じり出力されないように、MPI プロセスごとにファイルを分け出力させます（**リスト 16**）。

●リスト 15　標準入力ファイルの指定例

```
# test job.15
#PBS -q p64
#PBS -jo -N reqname
setenv F_FF05 infile  標準入力は装置番号05
setenv MPIEXPORT "F_FF05"
cd $PBS_O_WORKDIR
mpirun -np 64 -nn 4 a.out
```

●リスト 16　標準出力、標準エラーファイルの指定例

```
# test job.16
#PBS -q p64
#PBS -jo -N reqname
setenv MPISEPSELECT 4
setenv MPIEXPORT "MPISEPSELECT"
cd $PBS_O_WORKDIR
mpirun -np 64 -nn 4      (次行へ続いています)
    /usr/lib/mpi/mpisep.sh a.out
```

シェルスクリプト /usr/lib/mpi/mpisep.sh を実行形式ファイル a.out の前に記述し、環境変数 MPISEPSELECT に 1～4 の値を指定することで、**表 11** の動作を選択します。

●表11　MPISEPSELECT 値と動作

MPISEPSELECT	動作
1	各 MPI プロセスの標準出力のみを stdout.uuu:rrr へ格納
2	各 MPI プロセスの標準エラーのみを stderr.uuu:rrr へ格納（規定値）
3	各 MPI プロセスの標準出力と標準エラーを各々 stdout.uuu:rrr および stderr.uuu:rrr へ格納
4	各 MPI プロセスの標準出力と標準エラーを同一ファイル std.uuu:rrr へ格納

実行時、std.*や stderr.*の同名ファイルが存在する場合は、上書きでなく追加出力します。

5-6 MPI 用実行性能情報

MPIPROGINF 環境変数の設定により、プログラム性能情報（PROGINFO）の表示形式を変更することができます。DETAIL は Min/Max/Ave にまとめたもの、ALL_DETAIL は全てのランクについて、それぞれ性能情報を表示します。

DETAIL	性能情報を集約形式で出力します。
ALL_DETAIL	性能情報を拡張形式で出力します。全ランクの情報を出力します。

性能情報はプログラム実行終了後、標準エラー出力ファイルに出力されます。

●リスト 17　ジョブファイルの例
MPIPROGINF 環境変数

```
# test job.17
#PBS -q p64
#PBS -jo -N reqname
setenv MPIPROGINF DETAIL      実行性能情報
                               表示の指定
cd $PBS_O_WORKDIR
mpirun -np 64 -nn 4 a.out
```

6章　その他

使用メモリサイズ

実行に必要なメモリサイズを、事前に確認することができます。ただし allocate 等で動的に確保するメモリサイズは含まれません。

並列コンピュータで実行します。

逐次プログラムの場合

```
sxsize 実行形式ファイル名
```

並列化プログラムの場合
（自動並列または OpenMP）

```
sxsize -fl 並列数 実行形式ファイル名
```

●リスト 18　sxsize コマンド表示例

```
gen$ size a.out.s
1046912 + 140272 + 418928 = 1606112
                          逐次プログラム a.ous.s を実行すると、1,606,112 バイト使用します

gen$ size -fl 16 a.out.p
1046912(.text) + 140272(.data) + 418928(.bss) + 181855(.comment) + 4496(.whoami)
+ 1048576(logical task region) * 16 = 18569679
                          並列化プログラム a.out.p を 16 並列実行すると、18,569,679 バイト使用します
```

バイナリファイルの扱い　[Fortran]

データファイルを入力するなど他のサーバで作成したバイナリファイルを扱う場合、注意が必要です。SX-9 システムは Big-Endian 仕様ですので、Little-Endian 仕様[6]のバイナリファイルを扱うには、Endian を合わせる必要があります。変換は環境変数 F_FUFMTENDIAN を用い、Big-Endian に変換したい Little-Endian ファイルの装置番号を指定します。

環境変数 F_FUFMTENDIAN

```
setenv F_UFMTENDIAN u[,u]…
```

u は Little-Endian ファイルの装置番号です。複数ある場合は、,（カンマ）で区切って羅列します。

●リスト 19　ジョブファイル例

```
# test job.19
setenv F_UFMTENDIAN 30,40
cd $PBS_O_WORKDIR
a.out
```

装置番号 30,40 を Big-Endian に変換する

バウンズチェック　[Fortran]

配列の添字について、宣言している範囲内を使っているかを検査します。

プログラムの実行中に、適切な値を処理しているはずなのにエラー番号 250(オーバーフロー)や 252(ゼロ割り)、253(演算例外)等が発生する、または実行エラーは発生しないが実行する毎に動作や演算結果が異なるという場合、配列の添字検査をしてみてください。配列で参照されている添字が許される範囲にあるかどうかを調べ、範囲外であれば配列の大きさ、添字の値を表示します。

チェック方法は、-eC オプションを付けコンパイルしてから実行します。範囲外添字が見つかると、標準エラー出力ファイルにエラーメッセージを出力します（リスト 20）。

バウンズチェックのオプション

```
sxf90 -eC ソースファイル名
```

-eC オプションを指定すると、最適化やベクトル化および並列化はされませんので通常の実行に比べ時間がかかります。添字検査を行う時のみ、このオプションを用いてください。

[6] プロセッサによって仕様が異なります。ちなみに、本センターの並列コンピュータシステムは、Intel 製 Itanium2 なので Little-Endian 仕様です。

●リスト 20　エラーメッセージ例

```
*240  Subscript  error  array=b  size=10  subscript=11  eln=756  PROG=main
ELN=756(400021981)
```

配列 b はサイズ 10 の宣言であるが、添字 11 を使っているためエラーが発生している。
エラー発生箇所は、main ルーチン 756 行目。

パスワードの変更

ログイン名とパスワードは、スーパーコンピュータシステム、並列コンピュータシステムで共通です。初めてご利用の場合、初期パスワードが設定されていますので、ログイン後 yppasswd コマンドで速やかに変更してください（**リスト 21**）。並列コンピュータでコマンドを実行します。

パスワードはセキュリティ保護のため、ときどき変更することをお勧めします。

●リスト 21　パスワードの変更

```
gen$ yppasswd
yppasswd is deprecated, use
/usr/bin/passwd instead

Changing password for x2xxx9.
Old Password: 現在のパスワード
New password: 新しいパスワード
Re-enter new password: 新しいパスワード
（再度）
Changing NIS password for x2xxx9 on
xxx.isc.tohoku.ac.jp.
 Password changed
```

ログインシェルの変更

ログインシェルは、規定値で csh を設定しています。tcsh 等に変更したい場合は ypchsh コマンドを実行後、再ログインしてください（**リスト 22**）。

並列コンピュータでコマンドを実行します。

●リスト 22　ログインシェルの変更

```
gen$ ypchsh
ypchsh is deprecated, use /usr/bin/chsh
instead

Changing login shell for x2xxx9.
Password: パスワード
Enter the new value, or press return for
the default.
Login Shell [/bin/csh]: /bin/tcsh
 Shell changed.

（一度ログアウトして、再ログインする）
```

マニュアル

冊子マニュアルは、本センター本館 1 階 利用者相談室に備えております。

■冊子

[1]FORTRAN90/SX 言語説明書
[2]FORTRAN90/SX プログラミングの手引
[3]FORTRAN90/SX 並列処理機能利用の手引
[4]MPI/SX 利用の手引
[5]C++/SX プログラミングの手引
[6]科学技術計算ライブラリ ASL/SX 利用の手引
（基本機能編 1/4）
[7]科学技術計算ライブラリ ASL/SX 利用の手引
（基本機能編 2/4）
[8]科学技術計算ライブラリ ASL/SX 利用の手引
（基本機能編 3/4）
[9]科学技術計算ライブラリ ASL/SX 利用の手引
（基本機能編 4/4）
[10]科学技術計算ライブラリ ASL/SX 利用の手引
（高速機能編）
[11]科学技術計算ライブラリ ASL/SX 利用の手引
（並列処理機能編）

■オンライン

冊子番号[1]～[4]についてはオンラインマニュアルも用意しております。並列コンピュータ上で mansxf90 コマンドを実行します。HTML 形式のマニュアルがご覧になれます。

```
mansxf90
```

利用負担金

利用負担金は、主に演算負担経費とファイル負担経費からなります。演算負担経費は、演算した CPU 時間に応じて課金され、ファイル負担経費は週一回集計するファイル使用量を基に算出されます。

請求書は 4 半期（3 ヶ月）ごと[7]に、利用者を取りまとめている支払い責任者に発行します。

また、平成 20 年度も利用者全員に適用される利用負担金割引制度があります。演算した分だけ割引率が上がり、長時間利用するほどお得な制度になっ

[7] 年度末は、2 月中旬にも請求しています。

ています。詳しくは、以下のWebサイトをご覧ください。

http://www.cc.tohoku.ac.jp/　“利用負担金”

●表12　負担額単価表

		負担額
演算負担経費	スーパーコンピュータ	バッチ処理　0.4円／秒
		会話型処理　2　円／秒
	並列コンピュータ	バッチ処理　0.1円／秒
		会話型処理　0.2円／秒
ファイル負担経費		1Mバイト　0.1円／日

並列演算の場合、最長のCPU時間を演算時間とします。

現在の利用額を表示するには、以下2つのコマンドがあります。

■利用額の表示【利用者】

```
kakin
```

前日までの利用額を表示します（リスト23）。kakinコマンドを実行した個人の利用額です。並列コンピュータで実行します。

■利用額の表示【支払責任者】

```
skakin
```

前日までの利用額と負担額を、支払い責任者単位で表示します（リスト24）。利用額は実際利用した額、負担額は利用額から割引制度で減算した額になります。負担額の方を四半期ごとにご請求します。

このコマンドも並列コンピュータで実行します。

●リスト23　kakinコマンド表示例

```
利用者番号=x2xxxx
                    ---------- 利 用 額 ----------
月     スーパー     並 列     ファイル     そ の 他     調 整 額     合 計
 4        2262       592        58286            0            0     61140
 5           6         8        19421            0            0     19435
 6           0         0            0            0            0         0
 7           0         0            0            0            0         0
 8           0         0            0            0            0         0
 9           0         0            0            0            0         0
10           0         0            0            0            0         0
11           0         0            0            0            0         0
12           0         0            0            0            0         0
 1           0         0            0            0            0         0
 2           0         0            0            0            0         0
 3           0         0            0            0            0         0
合計      2268       600        77707            0            0     80575
```

●リスト24　skakinコマンド表示例

```
6月 12日 現在の利用額および負担額は次のとおりです。
  支払責任者 ： 東北　太郎                              (u20000)

                  合計          演算        ファイル        出力        その他
利用額       2,159,948     1,443,157         711,391       5,400             0
負担額       1,038,370       321,579         711,391       5,400             0
-------------------------------------------------------------------------------
  月別利用額
        4月       1,597,898
        5月         562,050
-------------------------------------------------------------------------------
  利用者別利用額
        x10000       123,456
        x10001             0
        x10002     1,656,431
```

お知らせ、お問い合わせ

■システム運用に関するお知らせ

大規模科学計算システム
http://www.cc.tohoku.ac.jp/

利用方法やシステムの運用について最新情報をお知らせします。

■システム利用に関するお問い合わせ

利用相談室　　sodan05@isc.tohoku.ac.jp
システム管理係　022-795-6252

■システム利用申請

詳細は、以下のウェブページをご参考ください。
http://www.cc.tohoku.ac.jp/guide/riyou.html

窓口（庶務係）　uketuke@isc.tohoku.ac.jp
022-795-3406

7章　おわりに

本稿は、SX-9システムのプログラミング利用ガイドとして基本的な手順を紹介しました。

研究室のサーバでは実現できなかったプログラムやアイデアを、ぜひ最新鋭のスーパーコンピュータシステムでお試しください。研究の強力なツールとしてご活用いただければ幸いです。ご不明な点、ご質問等ございましたら、お気軽にセンターまでお問い合わせください。

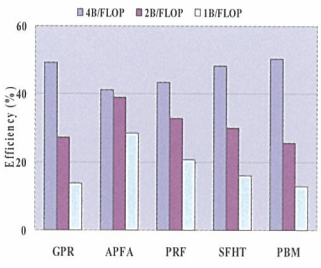
Early Evaluation of On-Chip Vector Caching for the NEC SX Vector Architecture

Akihiro Musa[1,2], Yoshici Sato[1], Ryusuke Egawa[1], Hiroyuki Takizawa[1],Koki Okabe[1] and Hiroaki Kobayashi[1]
1: Tohoku University, 2: NEC Corporation

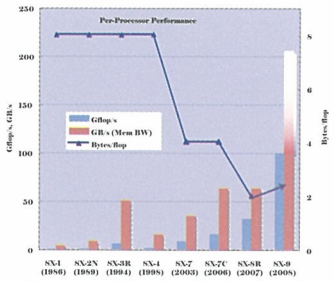
TOHOKU UNIVERSITY

Motivation

- **SX has high computational efficiency for scientific applications.**
  - **High efficiency is supported by the balance between processor performance and memory bandwidth: 4B/FLOP *1.**
- **On future SX, the B/FLOP < 4 due to Memory Wall Problem**
  - **Efficiency will decrease on scientific applications.**



Computational efficiency of SX-7 when changing memory bandwidth

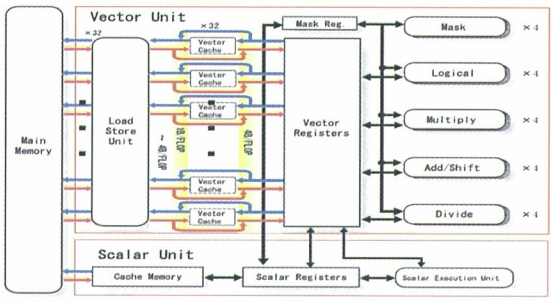


SX computational performance and memory bandwidth trend

*1 B/FLOP : the ratio of memory bandwidth to floating-point operations

On-Chip Vector Cache

- **Propose on-chip vector cache to cover a lack of memory bandwidth of future SX**
- **Vector Unit employs an on-chip vector cache.**
  - **32 sub-caches**
  - **Bypassing mechanism**



SX-7 processor architecture with vector cache

Summary of sub-cache structure

Line size	8 Byte
Set associative	2 Way
Write policy	Write Through

Evaluation of vector cache

- **Evaluated Applications**

Summary of leading scientific applications

Areas	Names and Descriptions	Methods Subdivisions
Electro-magnetic Analysis	GPR simulation: Simulation of Array Antenna Ground Penetrating Radar	FDTD 50x750x750
	APFA simulation: Simulation of Anti-Podal Fermi Antenna	FDTD 612x105x505
CFD/Heat Analysis	PRF simulation: Simulation of Premixed Reactive Flow in Combustion	DNS 513x513
	SFHT simulation: Simulation of Separated Flow and Heat Transfer	SMAC 711x91x221
Seismology	PBM simulation: Simulation of Plate Boundary Model on Seismic Slow Slip	friction law 32400x32400

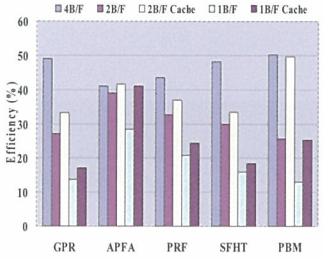
- **SX simulator with cache mechanism**
  - **Timing simulator**
  - **Single processor of the NEC SX-7 architecture**

Simulation parameters

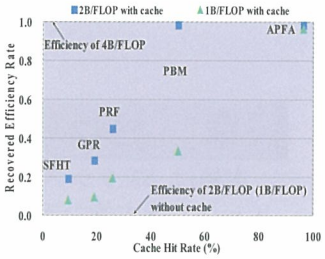
Cache size	32KB ~ 2MB
Bandwidth (Memory ~ Cache)	1, 2, 4 B/FLOP
Bandwidth (Cache ~ Register)	4 B/FLOP

(1) Efficiency and Cache Hit Rate

- **Cache can improve efficiency of the 2B/FLOP and 1B/FLOP**
- **Recovered efficiency rate goes up as cache hit rate increases.**
  - **PBM: Efficiency of 2B/FLOP with 50% cache hit rate is comparable to that of 4B/FLOP.**



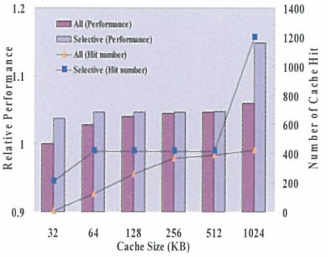
Effect of on-chip cache



Recovered efficiency rate

(2) Effect of selective caching

- **High locality of reference is selectively cached.**
  - **9% higher performance than the all caching on the 1MB cache in SFHT**



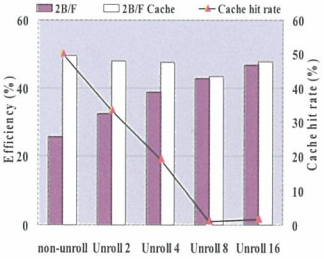
Performance of selective caching in SFHT

```
do 110 k=KST(I),NK,2
  do 110 j=JST(I),NJ,4
    do 110 i=IST(I),NI,2
      resA=(AN(i,j,k)*Phi(i,j+1,k)+AS(i,j,k)*Phi(i,j-1,k)
     &     +AE(i,j,k)*Phi(i+1,j,k)+AW(i,j,k)*Phi(i-1,j,k)
     &     +AT(i,j,k)*Phi(i,j,k+1)+AB(i,j,k)*Phi(i,j,k-1)
     &     +AP(i,j,k)*Phi(i,j,k)-DIV(i,j,k))/AP(i,j,k)
      resB=(AN(i,j+2,k)*Phi(i,j+3,k)+AS(i,j,k)*Phi(i,j+1,k)
     &     +AE(i,j+2,k)*Phi(i+1,j+2,k)
     &     +AW(i,j+2,k)*Phi(i-1,j+2,k)+AT(i,j+2,k)*Phi(i,j+2,k+1)
     &     +AB(i,j+2,k)*Phi(i,j+2,k-1)+AP(i,j+2,k)*Phi(i,j+2,k)
     &     -DIV(i,j+2,k))/AP(i,j+2,k)
110 continue
```

High cost kernel of SFHT
(Color arrays are selectively cached)

(3) Effect of unrolling and caching

- **Conflict between unrolling and caching**
  - **The efficiency of caching without unrolling is 3% higher than that of unrolling without caching on 2B/FLOP.**
  - **More discussions on the tradeoff will be addressed as our future work.**



Effects of unrolling and caching in PBM

```
do i=1,ncells
  do j=1,ncells
    wf_dip(i)=wf_dip(i)+gd_dip(j,i)*wary(j)
  end do
end do
```

```
do i=1,ncells,4
  do j=1,ncells
    wf_dip(i)=wf_dip(i)+gd_dip(j,i)*wary(j)
    wf_dip(i+1)=wf_dip(i+1)+gd_dip(j,i+1)*wary(j)
    wf_dip(i+2)=wf_dip(i+2)+gd_dip(j,i+2)*wary(j)
    wf_dip(i+3)=wf_dip(i+3)+gd_dip(j,i+3)*wary(j)
  end do
end do
```

Outer unrolling of PBM

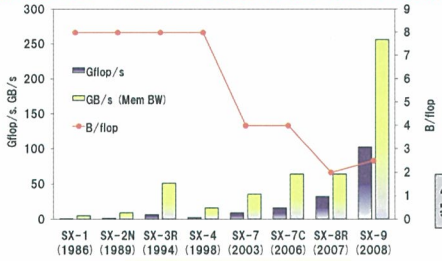
NEC

MSHRを加えたベクトルプロセッサ用キャッシュメモリの性能評価

佐藤義永[1], 撫佐昭裕[1, 2], 江川隆輔[1], 滝沢寛之[1], 岡部公起[1], 小林広明[1]
1：東北大学, 2：日本電気株式会社

ベクトルスーパーコンピュータのさらなる性能向上に向けて

演算性能あたりのメモリバンド幅(B/flop)が低下していくにつれて実行効率が減少する傾向

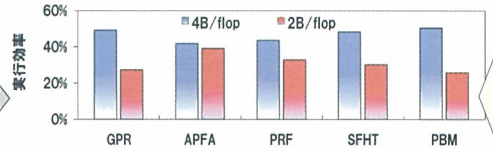


SXシリーズの性能トレンド

次世代ベクトルスーパーコンピュータ
高いB/flopを維持するためには高いメモリバンド幅が要求される．しかし，ピンバンド幅の物理的限界によりメモリバンド幅の劇的な向上は困難
実行効率を高める新しいアーキテクチャの実現が不可欠

SX-4 → SX-7 → SX-7C → SX-9 → ??

4B/flop→2B/flop 実行効率を比較



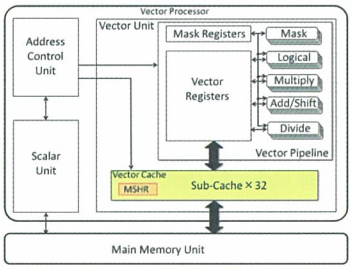
B/flopと実行効率の関係

東北大学サイバーサイエンスセンターで利用されているアプリケーション
GPR：超帯域アンテナ解析
APFA：不均質媒体中における電磁伝搬と散乱
PRF：二次元予混合火炎の不安定性解析
SFHT：非定常剥離流れの熱伝導解析
PBM：プレート境界面上における滑りシミュレーション
全てB/flop低下で実行効率低下の傾向

MSHRを有するベクトルキャッシュを備えたベクトルプロセッサ

ベクトルキャッシュを利用して不足するメモリバンド幅を補い，高い実行効率を実現する

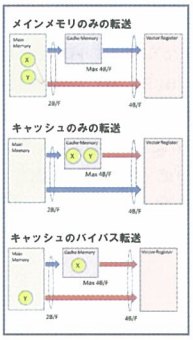
- SXが持つ32個のメモリポート毎にベクトルキャッシュを設置
- メモリからレジスタへの直接転送を可能とするバイパスメカニズムを採用
- MSHRの利用によるキャッシュヒット率の向上



キャッシュを加えたベクトルプロセッサ

ベクトルキャッシュの仕様

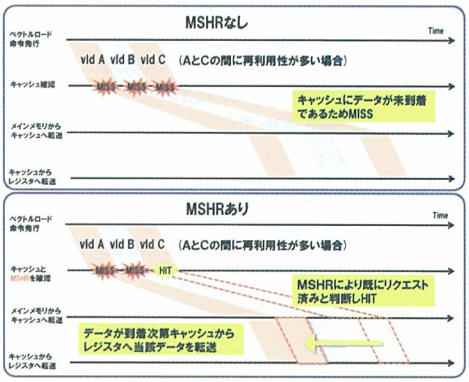
Line size	8 Bytes
Associativity	2 Ways
Write policy	Write through



バイパスメカニズム

MSHR - Miss Status Handling Register

MSHRとは？
・キャッシュミスを起こしたベクトルロード命令のメモリアドレスを記録するレジスタ
MSHRが保持する情報
・キャッシュミスしたロード命令に関する情報
(メモリアドレス、キャッシュラインオフセット、ターゲットレジスタ)
MSHRの動作
①キャッシュミスが生じたメモリロード命令はMSHRを確認
②・MSHRに情報が保持されていない場合
MSHRに情報を追加しメモリからキャッシュへデータを転送
・MSHRに既に情報が保持されている場合
データがキャッシュへ届くまで待機，届き次第データを転送
MSHR利用による利点
・キャッシュヒット率の向上
・重複するメモリアクセス削減によるメモリバンド幅への負荷低減



姫野ベンチマークによる性能評価

●姫野ベンチマーク主要カーネル

姫野ベンチマーク：ポアソン方程式解法をヤコビの反復法で解く際の主要ループで構成

```
DO K=2,kmax-1
   DO J=2,jmax-1
      DO I=2,imax-1
         S0=a(I,J,K,1)*p(I+1,J,K)+a(I,J,K,2)*p(J+1,K)
     1      +a(I,J,K,3)*p(I,J,K+1)
     2      +b(I,J,K,1)*(p(I+1,J+1,K)-p(I+1,J-1,K)
     3      -p(I-1,J+1,K)+p(I-1,J-1,K))
     4      +b(I,J,K,2)*(p(I,J+1,K+1)+p(I,J-1,K+1)
     5      -p(I,J+1,K-1)-p(I,J-1,K-1))
     6      +b(I,J,K,3)*(p(I+1,J,K+1)+p(I-1,J,K+1)
     7      -p(I+1,J,K-1)-p(I-1,J,K-1))
     8      +c(I,J,K,1)*p(I-1,J,K)+c(I,J,K,2)*p(I,J-1,K)
     9      +c(I,J,K,3)*p(I,J,K-1)+wrk1(I,J,K)
         SS=(S0*a(I,J,K,4)-p(I,J,K))*bnd(I,J,K)
         GOSA1=GOSA1+SS*SS
         wrk2(I,J,K)=p(I,J,K)+OMEGA *SS
      enddo
   enddo
enddo
```

サイズSS	I: 64 J: 32 K: 32
サイズS	I: 128 J: 64 K: 64
サイズM	I: 256 J:128 K:128
サイズL	I: 512 J:256 K:256
サイズXL	I:1024 J:512 K:512

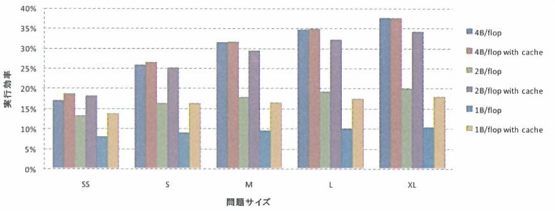
シミュレーションパラメータ

Cache Size	32KB ~ 2MB
Bandwidth (memory-cache)	4B/flop, 2B/flop 1B/flop
Bandwidth (cache-register)	4B/flop

●問題サイズと実行効率

・B/flopの低下による性能低下を軽減可能
・レイテンシの短縮により4B/flopの場合でも実行効率が向上

問題サイズ大きくなると演算時間が増加しメモリロード時間が隠ぺいされ，実行時間に対する演算時間の割合が増加するため，実行効率は高くなっていく．しかし，B/flopが減少するとメモリロード時間が増加し，隠ぺいできなくなるために実行効率が大きく低下する．これに対し，ベクトルキャッシュは，このような状況においてもメモリロード時間を削減し，実行効率の向上に貢献する．
SSやSなど問題サイズが小さい場合はベクトル長が64，128と短く，メモリレイテンシが表面化するが，キャッシュを導入することでレイテンシが削減され，実行効率が向上する．

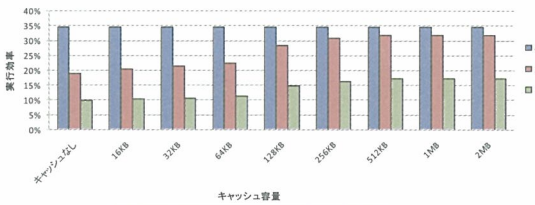


問題サイズと実行効率

●キャッシュサイズと実行効率

・ヒット率45%で4B/flopと同等の性能(2B/flop時)
・B/flopが小さいほどキャッシュサイズの影響が大きい

問題サイズLの場合，キャッシュ容量が256KB以上あればヒット率45%程度となり十分な実行効率が得られる．これは，キャッシュのバイパスによりキャッシュとメインメモリ両転送の合計が4B/flopに達するためである．キャッシュ容量が256KBを下回る場合，2B/flop以下では実行効率が低下していく．



キャッシュサイズと実行効率

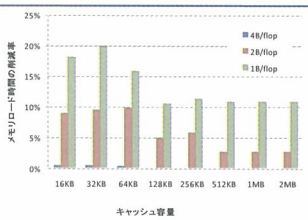
キャッシュサイズとヒット率

●MSHRの有効性

・MSHRの導入でB/flopが低い時の実行効率向上
・MSHRは全実行時間に対するメモリロード時間の割合を削減する効果

キャッシュミスが増える256KBを下回る場合において高い効果を示し，MSHRを用いることで最大20%の性能向上が得られる．1B/flopの場合では最大で20%のメモリロード時間を削減でき，その結果として2MBのベクトルキャッシュにMSHRを導入することで，キャッシュメモリを用いない場合と比較して94%の実行効率の向上を得られることが確認できる．

キャッシュサイズとMSHRの効果



メモリロード時間の削減割合

まとめ

●姫野ベンチマークにおいてMSHRを備えるベクトルキャッシュを利用することで2B/flopのシステムでも4B/flopのシステムと同等の実行効率を達成
- MSHRを備えるベクトルキャッシュは高いB/flopが求められる大規模科学技術計算の高速化に有望
- 問題サイズが小さい場合，メモリアクセスレイテンシ短縮の効果による実効性能向上も期待可能

●スカラプロセッサと比較して少ないプロセッサ数で同等の演算性能を達成できるため，台数効果が高くないアプリケーションでも高い実効性能を達成可能

本提案のベクトルキャッシュはペタフロップス級の計算においても高い実行効率を達成するという目標に対し大きなブレークスルーを与える技術

Caching on a Chip Multi Vector Processor

Akihiro Musa[1,2], Yoshiei Sato[1], Ryusuke Egawa[1], Hiroyuki Takizawa[1],
Koki Okabe[1] and Hiroaki Kobayashi[1]
[1] Tohoku University,
Sendai, 980-8578, Japan
[2] NEC Corporation,
Tokyo, 108-8001, Japan
{musa, yoshiei}@sc.isc.tohoku.ac.jp
{egawa,tacky, okabe, koba}@isc.tohoku.ac.jp

ABSTRACT

Chip multiprocessors (CMPs) have become the mainstream in processor architectures. CMP architectures will be applied to vector processor design in the near future. The computational efficiency of vector supercomputers relies on their high memory bandwidth. Therefore, we propose an on-chip shared cache to maintain the effective memory bandwidth for a chip multi vector processor (CMVP). We evaluate the performance of the CMVP based on the NEC SX vector architecture using real scientific applications. Especially, the caching effects on the sustained performance are examined when the rate of the bandwidth to the performance decreases. The experimental results indicate that an 8 MB on-chip shared cache can improve the performance of a four-core CMVP by 15% to 40%, compared with that without the cache. This is because the shared cache can increase cache hit rates of multi threads.

Caching on a Chip Multi Vector Processor

The continued increase in the number of transistors on a chip named Moore's law has led to the development of chip multiprocessors. In this work, we discuss the design of a chip multi vector processor (CMVP), especially, examining the effects of an on-chip cache when the off-chip memory bandwidth is limited. To keep a higher sustained performance, a vector processor (core) generally requires that the ratio of the memory bandwidth to the arithmetic performance (B/FLOP) is at least 4. However, vector supercomputers have been encountering the memory wall problem due to the limited pin bandwidth. In this paper, therefore, we propose a shared on-chip non-blocking vector cache as a solution to the memory wall problem in the future vector supercomputers.

As shown in Figure 1, we design CMVP with a shared cache based on the NEC vector architecture. The core has four parallel vector pipe sets, each of which contains five types of vector arithmetic pipes (Mask, Logical, Add/Shift, Multiply, Divide), and 144 KB vector registers. This core is interconnected to the vector cache through 32 memory ports. The cache is an 8 MB data cache that consists of 32 2-way set-associative sub-caches with miss status handling registers. The memory bandwidth between the core and the cache can provide data to vector registers at the rate of 4 B/FLOP.

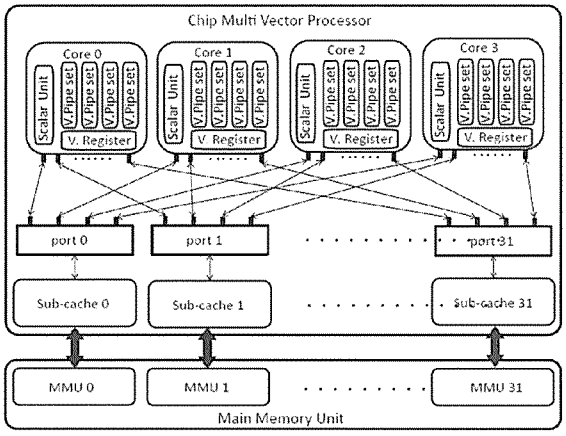


Figure 1: Block diagram of vector processor

We evaluate the effects of the cache on CMVP when the off-chip memory bandwidth per the CMVP flop/s is changed from 1 to 4. For the evaluation, we use four scientific applications as shown in Table 1, and the NEC SX timing simulator. The parameters of CMVP are shown in Table 2. These application programs are representative of scientific simulations.

Table 1: Benchmark Programs

No.	Descriptions	Methods	Subdivisions	Memory sizes
1	Solution of Poisson Equation	Jacobi (relaxation)	SS: 64 x 64 x 64 S: 128 x 64 x 64 M: 256 x 128 x 128 L: 512 x 256 x 256 XL: 1024 x 512 x 512	48 MB 96 MB 496 MB 3.6 GB 28.8 GB
2	Simulation of Array Antenna Ground Penetrating Radar	FDTD	50 x 750 x 750	8.9 GB
3	Simulation of Anti-Podal Fermi Antenna	FDTD FFT	612 x 105 x 505	12 GB
4	Simulation of Separated Flow and Heat Transfer	SMAC	711 x 91 x 221	6.6 GB

Table 2: Summary of setting parameters

Parameters	Remarks
Cores	1, 4
Number of Pipe Set	4
Pipe Types	Mask, Logical, Add/Shift, Multiply, Divide
Cache size (Sub-cache)	8 MB (256 KB)
Associativity	2-way
Cache Policy	LRU, Write-through
Cache Bank Cycle	5% of memory cycles
Cache Latency	15% of memory latency
Line size	8 B
MSHR Entries per Sub-cache	256
Bandwidth: Memory - CPU	1, 2, 4 B/FLOP
Bandwidth: Cache - Core	4 B/FLOP

Figure 2 shows the computational efficiency in the execution of Program 1 on a single core processor when changing the memory bandwidth from 1 B/FLOP to 4 B/FLOP. These results clarify that the computational efficiency goes down as the memory bandwidth decreases. Thus, the efficiency of vector supercomputers strongly depends on their memory performance. As the cache can compensate the performance degradation due to the reduction in the B/FLOP rate, it can boost the computational performance. In the cases of SS and S, the 2 B/FLOP system with the cache can achieve almost the same performance as the 4 B/FLOP system. Here, the cache hit rates of SS, S and XL are 57%, 52% and 51%, respectively. Moreover, for SS, S, and M, the performance of the 4 B/FLOP system with the cache is higher than that without the cache, because the cache can decrease the memory access latency.

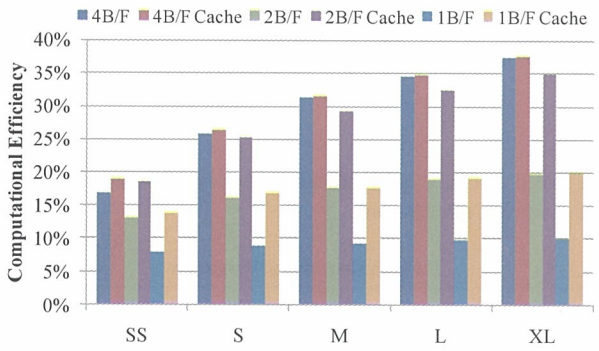


Figure 2: Computational efficiency of Program 1 on single core

Figure 3 shows the scalabilities of the four programs when changing the B/FLOP rate from 1 to 4 in the four-core CMVP without the cache. The scalability of the vector system decreases with the B/FLOP rate, except Program 3, which is computation intensive.

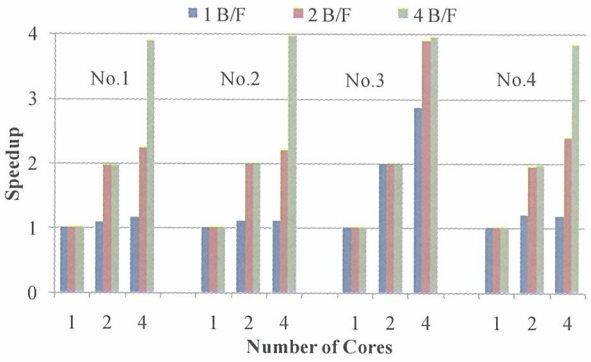


Figure 3: Scalability of Programs

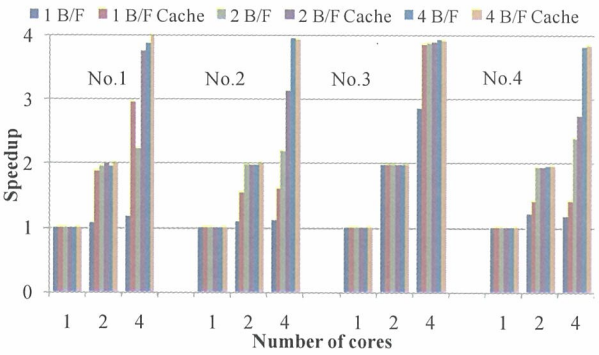


Figure 4: Scalability of Programs with cache

Figure 4 shows the scalabilities of the four programs boosted by the cache mechanism. In the cases of using the cache, the performances of the 1 B/FLOP and 2 B/FLOP systems for Program 3 are comparable to that of the 4 B/FLOP system, even if the number of cores increases. Meanwhile, for Programs 2 and 4, the speedup ratios of the 1 B/FLOP system are smaller than those of the 2 B/FLOP system. This is because the 1 B/FLOP system cannot provide data to vector registers at the rate of 4 B/FLOP even if all the data are on the cache, even though the 2 B/FLOP system can. Accordingly, these results suggest that the off-chip memory bandwidth per core should satisfy at least 2 B/FLOP to achieve a high scalability.

For multi-threaded programs of various difference schemes, a thread can reuse the data previously loaded by another thread. Figure 5 shows a kernel loop of Program 2. Suppose that the outermost loop, index *k*, is parallelized for multi-threading. Then, in the case of a single thread, arrays *H_y(i,j,k-1)* and *H_x(i,j,k-1)* are not reused. In the case of multiple threads, however, they are reused by another thread. Figure 6 shows the cache hit rates and the improved efficiency per core. The performance gain per core increases with the cache hit rate. Consequently, sharing the cache among vector cores increases the cache hit rate of difference schemes, resulting in performance improvement.

```
DO 10 k=0,Nz ; DO 10 i=0,Nx; DO 10 j=0,Ny
      E_x(i,j,k) = C_x_a(i,j,k)*E_x(i,j,k)
&           + C_x_b(i,j,k) * (( H_z(i,j,k) - H_z(i,j-1,k))/dy
&           - (H_y(i,j,k) - H_y(i,j,k-1))/dz - E_x_Current(i,j,k))
      E_z(i,j,k) = C_z_a(i,j,k)*E_z(i,j,k)
&           + C_z_b(i,j,k) * ((H_y(i,j,k)-H_y(i-1,j,k) )/dx
&           - (H_x(i,j,k)-H_x(i,j-1,k) )/dy - E_z_Current(i,j,k))
      E_y(i,j,k) = C_y_a(i,j,k)*E_y(i,j,k)
&           + C_y_b(i,j,k) * ((H_x(i,j,k) - H_x(i,j,k-1) )/dz
&           - (H_z(i,j,k) - H_z(i-1,j,k))/dx - E_y_Current(i,j,k))
10 CONTINU
```
Figure 5: High cost kernel loop of program 2

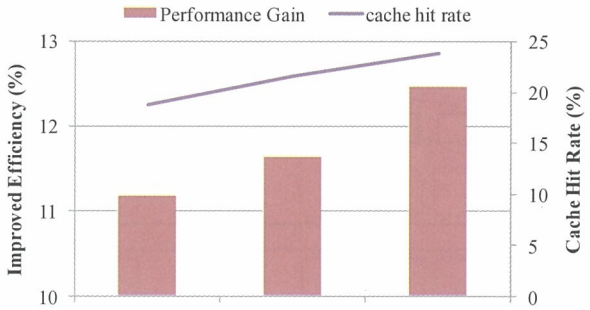


Figure 6: Cache hit and Improved efficiency

論文リスト

1. Akihiro Musa, Yoshiei Sato, Takashi Soga, Ryusuke Egawa, Hiroyuki Takizawa, Koki Okabe, and Hiroaki Kobayashi, SC08 Poster Presentation at SC08, 2008.

2. 佐藤義永、撫佐昭裕、江川隆輔、滝沢寛之、岡部 公起、小林広明, "ベクトルプロセッサ用キャッシュメモリにおけるMSHRの性能評価"、次世代スーパーコンピューティング・シンポジウム 2008,9/16-17, 2008.(特別賞受賞)

3. M.Resch, H.Kobayashi et al.(Ed.), High-Performance Computing on Vector Systems 2008, Reading, Springer-Verlag, 2008.

4. A.Musa, Y.Sato, T.Soga, R.Egawa, H.Takizawa, K.Okabe, H.Kobayashi, "Effects of MSHR and Prefetch Mechanism on an On-Chip Cache of the Vector Architecture, " To be presented at ISPA2008.

5. A.Musa, Y.Sato, T.Soga, K.Okabe, R.Egawa, H.Takizawa, H.Kobayashi,"A Shared Cache for a Chip Multi Vector Processor,"To be presented at the MEDEA workshop (PACT 08), 2008.

6. A.Musa, Y.Sato, R.Egawa, H.Takizawa, K.Okabe, H.Kobayashi, "Early Evaluation of On-Chip Vector Caching for the NEC SX Vector Architecture," Poster Presentation at SC07, 2007.

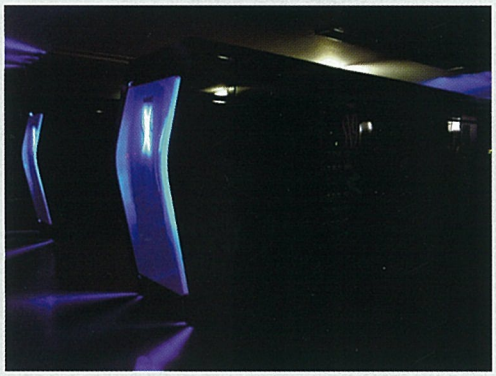
7. H.Kobayashi, A.Musa, Y.Sato, H.Takizawa, K.Okabe, "The Potential of On-Chip Memory Systems for Future Vector Architectures," M.Resch et al. (ED.) High Performance Computing on Vector Systems 2007, Springer-Verlag, pp.247-264, 2007.

8. H.Kobayashi,The Potential of On-Chip Memory Systems for Future Vector Architectures, Invited Talk at the 16th CCSE Workshop on High- Performance Computing on Vector Based Architectures - Recent Achievements and Future Directions-, Tokyo, Japan, Apr. 23, 2007.

9. A.Musa, Y.Sato, R.Egawa, H.Takizawa, K.Okabe, H.Kobayashi, "An On- Chip Cache Design for Vector Processors," Proceedings of the MEDEA workshop (PACT 07), pp.17-24, 2007.

10. H.Kobayashi, "Implication of Memroy Performance in HEC systems," M.Resh, T.Bonisch, K.Benkert, and T Furui (ED.) High Performance Computing on Vector Systems, Springer-Verlag, pp.21-50, 2006.

11. A.Musa, H.Takizawa, K.Okabe, T.Soga, H.Kobayashi, "Implications of Memory Performance for Highly Efficient Supercomputing of Scientific Applications, " Proceedings of International Symposium on Parallel and Distributed Processing and Application (ISPA06), pp. 845-858, 2006



Cyberscience Center Tohoku University

高速化推進研究活動報告　第4号
平成20年11月　発行
編集・発行　国立大学法人東北大学
サイバーサイエンスセンター
〒980-8578　仙台市青葉区荒巻字青葉6-3
TEL 022-795-3407
http://www.isc.tohoku.ac.jp