高速化推進研究活動報告

第 3 号

平成17年3月

東北大学情報シナジーセンター

高速化推進研究活動参加者

東北大学情報シナジーセンター

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，岡部公起
システム管理係
伊藤英一，大泉健治
システム運用係
小野敏

日本電気株式会社

ＨＰＣ販売推進本部
小久保達信，小林義昭，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，山本秀喜
第一官庁システム開発事業部
撫佐昭裕，石井繭子，塚本龍治，吉村健二
神山典，金野浩伸，岡崎昌夫，山形正明，吉田智，菅雄一郎
ＮＥＣ情報システムズ
浅見暁，小林一夫，後藤記一
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
曽我隆，松村佳明，伊藤学

高速化推進研究活動報告　第３号

目次

1　高速化推進研究活動報告第３号の刊行にあたって

センター長　根元　義章

ここに高速化推進研究活動報告の第3号をお届けする．報告書は2年間ごとの成果をとりまとめ，その成果を広く紹介することでスーパーコンピュータのより良い利用環境の構築に貢献することを目的として作成されている．センターとして1997年に公式に高速化推進に取り組んでから8年が経過したことになる．この取り組みは他にはない本センターの独自のものである．この取り組みが極めて重要であるとの関係各方面から評価をいただくに至り，これまでのセンターの活動の正当性が示されていると考える．

振り返ってみると，8年以上前になると思うが，当時最高速のシステムへの機種変更が決定した折，ある利用者のプログラムの実行に要する処理時間に性能改善の効果が現れない状況があった．最先端のシステムを全体として効率よく運用するのがセンターの責務であり，利用者と協力してその原因を追究した．その結果，プログラムが導入されたシステムの特徴を十分に活用できる構造ではないことが判明した．もちろん導入システムには，最適化の手法が備えられているが，それを活用できていないか，あるいは最適化の適応範囲を逸脱しているかのどちらかである．最先端の機種を効果的に利用できなければ，利用者の研究活動の進展，ならびにセンターのシステムの効率的な運用に支障をきたす．

利用者の願いは，時間的，経済的に効率良く処理結果を獲得し研究を推進することである．これに対するセンターの使命は，絶えず最高性能のシステムを導入し，その性能を十分に利用できる環境を整備することにある．このような利用環境を実現するには，単にシステムを導入し運用するだけでは，不十分である．以前より，センターはメーカとの間で，システムを効率的に運用することを絶えず真剣に検討してきた．新しく導入されたシステムを最大限有効に利用するには，プログラムをチューンナップし，高速化を図ることが必要であった．現在このことは当然であると考え始められているが，本センターでは，ハードウェアの導入運用のみがセンターの責務ではなく，サービスの質的な改善を図ることを目指し，8年前から高速化支援を業務として取り組んできた．高速化を行うには，利用者によるプログラム開発に加えて，専門知識の活用，時にはシステムへの改善が要求され，システムを運用管理しているセンターとシステムを開発したメーカの共同作業として取り組むことが必須である．幸いなことに，メーカの全面的な支援を得ることができ，何の違和感もなく高速化推進プロジェクトがスタートできている．その結果，センターの共同利用施設としての責務として，高速化推進が一つのプロジェクトとして確立され，これまで多くの成果が得られている．高速化支援により利用が一層促進され，システムの極限までの活用にもつながっている．高価なシステムを最大限に利用できることは極めて大きい意義がある．

量の時代から質の時代，知の時代への転換が必須である現在，如何にそれを実現していくかがいたるところで強く問われている．新たな評価基準により一層の進展が期待されている．激動，そして閉塞感の漂う時代，どうしてもそれが必要である．

大規模科学計算の実行環境については，スーパーコンピュータのあり方，共同利用センターの果たすべき役割について継続的に論議がなされ，高度な利用環境が維持されなければならない．そのためには，量のみならず質も面での優位性が立証されなければならい．ここで報告した高速化推進プロジェクトは，実績のある事例として，今後の模範となるものと確信する．

いうまでもなく，多くの研究分野でシミュレーションの重要性，有用性が強く認識され，センターのスーパーコンピュータの利用は増加の一途をたどっている．SX-7 に更新した後のスーパーコンピュータの運用状況をみると，すでに SX-7 の能力の限界を超えつつあることからも、このことは理解できる．また利用者の計算規模は確実に巨大化している．これらの状況のもと，より一層高速化に取り組み新たな高速化手法を開発することが急務である．本プロジェクトでは，これまでは 1 つのノードに閉じた高速化推進を中心としてきたが，利用者の研究の進展に歩調を合わせ，複数のノードを使用しての超大規模計算に対する高速化推進が今後の重要なテーマである．

最後に，本プロジェクトにご協力いただいた利用者および日本電気（株）の関係者に感謝する．

2　高速化推進研究活動報告

スーパーコンピューティング研究部　小林広明

2．1　　はじめに

コンピュータシミュレーションは，伝統的な科学技術研究の方法である理論，実験に加え，先端科学技術の発展に大きく貢献する第3の方法として，その重要性はますます大きくなっている．特に，国が定める先端科学の4つの重点研究分野であるライフサイエンス，情報通信，環境，ナノテク・材料においては，スーパーコンピュータを用いた大規模・高精度シミュレーションは必要不可欠であり，コンピュータシミュレーションを支える高性能・高効率スーパーコンピュータの開発・整備・運用は，日本の先進科学分野での国際的リーダシップ，産業界の製品開発における国際競争力の維持・発展，さらには災害に強い安心・安全な社会形成・維持に大きく貢献している．

東北大学情報シナジーセンターは，計算科学のための学術情報基盤として常に最高性能のスーパーコンピュータシステムを導入・運用し，利用者に提供することにより，最先端の学術研究を強力に支援・推進している．加えて，研究者にとってプログラムの開発が容易なシステムの構築，他では実行できない大規模・長時間ジョブの実行環境の整備，専門的な立場からの利用者プログラムの高速化の支援等も行っている．特に，スーパーコンピュータの性能・機能を極限まで引き出すために，プログラムの高速化技術，およびシミュレーションの大規模化，高精度化に関する研究・開発を，センター教員・技術職員，システム利用者である研究者，システムベンダである日本電気（株）（以下NEC）の技術者が一体となって取り組んでいる．本報告書はこのような研究開発により得られた成果をまとめたものである．

本節では，情報シナジーセンターの大規模科学計算システムの整備・運用状況，およびプログラム高速化推進支援体制および実施状況について述べるとともに，その活動の1つとしてHPCチャレンジベンチマークによるSX-7スーパーコンピュータの性能評価の取り組みについて説明する．

2．2　　情報シナジーセンターの大規模科学計算システム

情報シナジーセンターは，一般の研究室レベルの計算機では実現不可能な大規模・高精度シミュレーション計算環境を提供し，研究の生産性を高めるために計算結果をできる限り短時間で得られるようにすることを最優先に考えて，スーパーコンピュータシステムの設計，および運用を行っている．スーパーコンピュータは，現在ベクトル並列型とスカラ並列型の2つの方式に分類することができるが，その適用範囲は様々である．それぞれの方式とHPC(high-performance computing, 高性能計算)を必要とする代表的な分野とをデータ規模とメモリバンド幅の観点で関係つけたものを図2.2.1に示す．

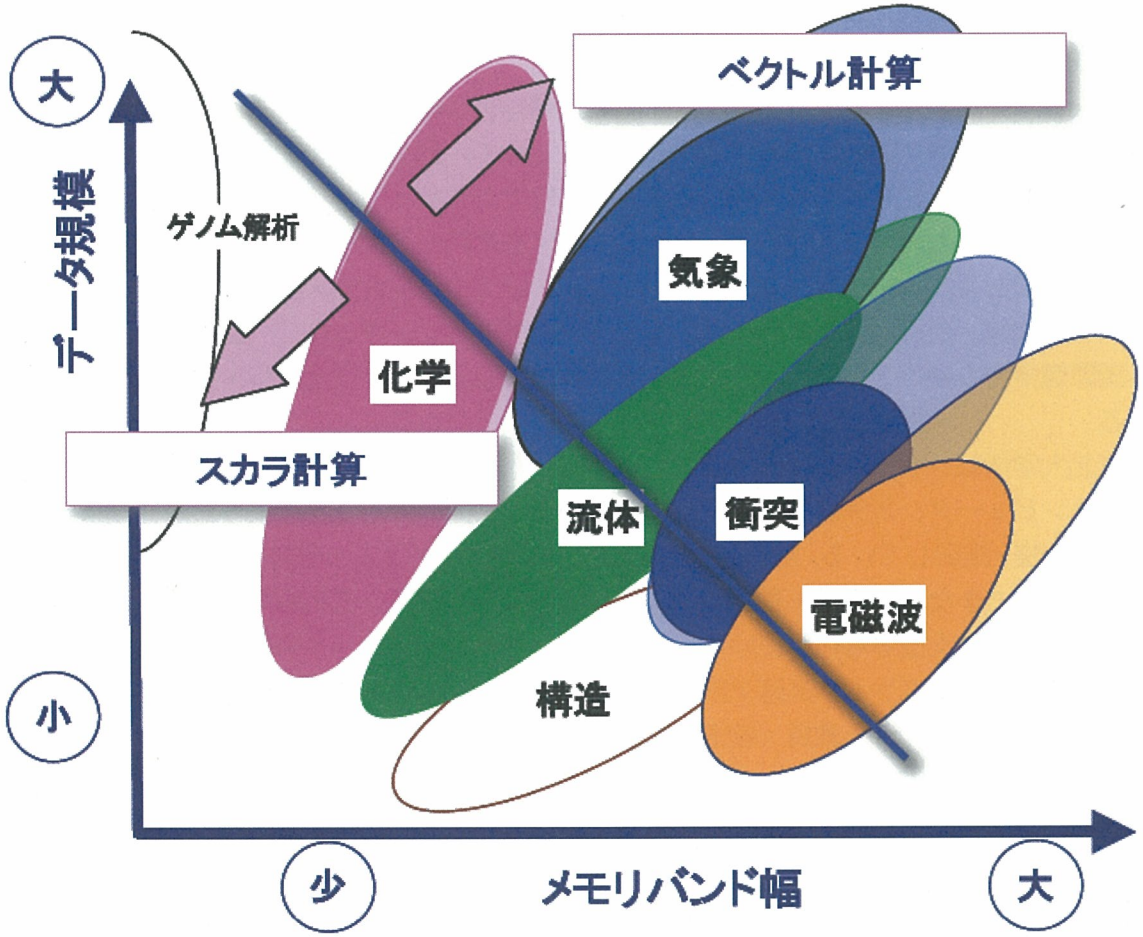


図 2.2.1　データ処理の観点から分類した HPC 分野

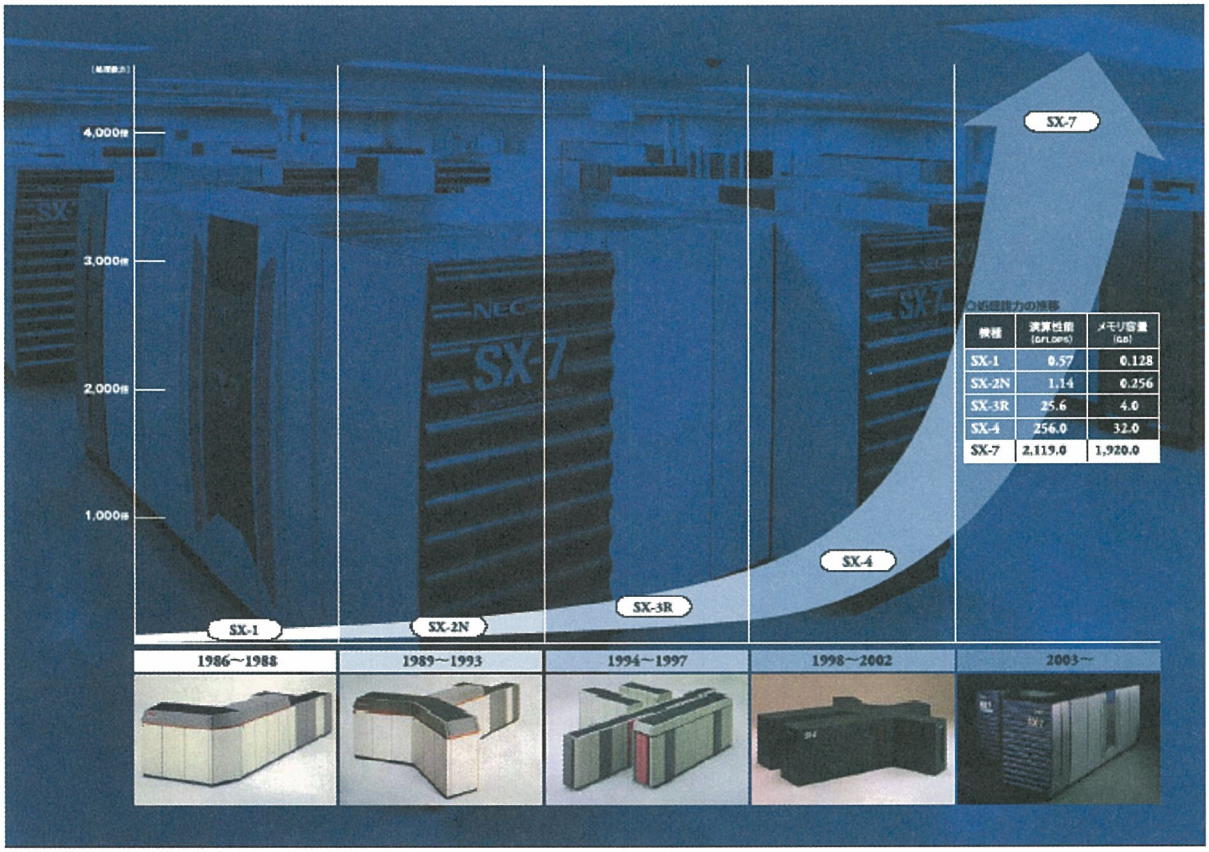
ここで，メモリバンド幅とは，単位時間あたりに演算器に供給できるデータ量，すなわち演算器に対するデータ供給能力を意味する．コンピュータの急速な性能向上に伴い，シミュレーションの大規模化と高精度化が進み，扱うデータサイズの巨大化はとどまることを知らない．そのような大規模データを効率よく処理するためには，演算性能ばかりでなく，それに見合った高いデータ供給能力が求められる．

低コストのスーパーコンピュータとして商用部品を多用したスカラ並列型スーパーコンピュータが普及しつつあるが，このタイプのシステムは，キャッシュメモリが有効に機能することにより，その高い演算性能が発揮される．逆に言えば，キャッシュサイズによりデータ供給能力が制限されるために，キャッシュミスを頻繁に引き起こす大量のデータを扱うプログラムの実行については，その能力を生かすことができなくなる．従って，スカラ並列型スーパーコンピュータは，演算に対してデータの局所化が容易，かつ大規模並列性を有するゲノム解析やデータマイニングなどでは威力を発揮するが，流体，気象，電磁解析などデータの局所化が非常に難しく，並列化

によりさらなるメモリ性能，CPU 間通信性能が求められる問題に対しては，高い実効性能（実際にアプリケーションを実行したときに得ることができる性能）が得られないという特徴がある．

一方，ベクトル型スーパーコンピュータは，ベクトル命令，ベクトル演算パイプライン，ベクトルロードストアユニット，ベクトルレジスタ，高速クロスバスイッチといった大規模データを効率よく扱うための機構がハードウェアとソフトウェアの両面から用意されており，先端科学分野における大規模・高精度シミュレーションを高効率で実行することが可能である．このようなことから，情報シナジーセンターは，本センターの大規模科学計算システムとして，世界最高クラスの性能を有するベクトル並列型スーパーコンピュータを中心に据え，加えて，ベクトル並列型スーパーコンピュータを必要としないプログラム実行，および汎用アプリケーションサービス用に対してはスカラ並列型コンピュータを用意することにより，学術研究者が求める様々な計算要求に対応できる世界有数の優れた計算環境を実現している．

スーパーコンピュータ SX-7 は，図 2.2.2 に示すように，科学技術計算専用の高速コンピュータとして 1986 年に第 1 号機 SX-1 が導入されて以来，5 代目のベクトル並列型スーパーコンピュータである．ベクトル並列型スーパーコンピュータ導入のためのシステム設計においては，情報シナジーセンターでは，利用者のシミュレーションモデルの開発，実行，結果取得までのサイクルを最短にすることができるシステムを実現することを最優先に考えて，利用者の実プログラムの検討を日夜行い，図 2.2.3 に示すような要求要件を設定した．これに基づき開発された現在運用中のベクトル並列型スーパーコンピュータ SX-7 は，流体解析・航空機設計，電磁界解析・アンテナ設計，地震解析，地雷探索，気象予測，半導体設計，燃焼・伝熱シミュレーション，ナノプロセス技術・ナノ電子デバイスの開発などに求められる高精度・大規模科学技術計算の高速実行に威力を発揮し，学会ばかりでなく新聞紙上をにぎわすような先端科学技術分野の優れた成果を生み出し，この分野の研究促進に大いに貢献している．また，2.4 節で述べるように，SX-7 は，スーパーコンピュータの新しい国際性能指標 HPCC (HPC Challenge)ベンチマークの 28 項目のうち 16 項目で世界最高性能を達成(2004 年 11 月登録時)し，ベクトル並列型スーパーコンピュータの優れた潜在能力を国内外に示した．



導入年	名称	性能(GFLOPS)	記憶容量
1986	SX-1	0.57	128MB
1990	SX-2N	1.14	256MB
1994	SX-3	25.6	4GB
1998	SX-4	256	32GB
2003	SX-7	2,119	1,920GB

図 2.2.2　情報シナジーセンターベクトル型スーパーコンピュータの変遷

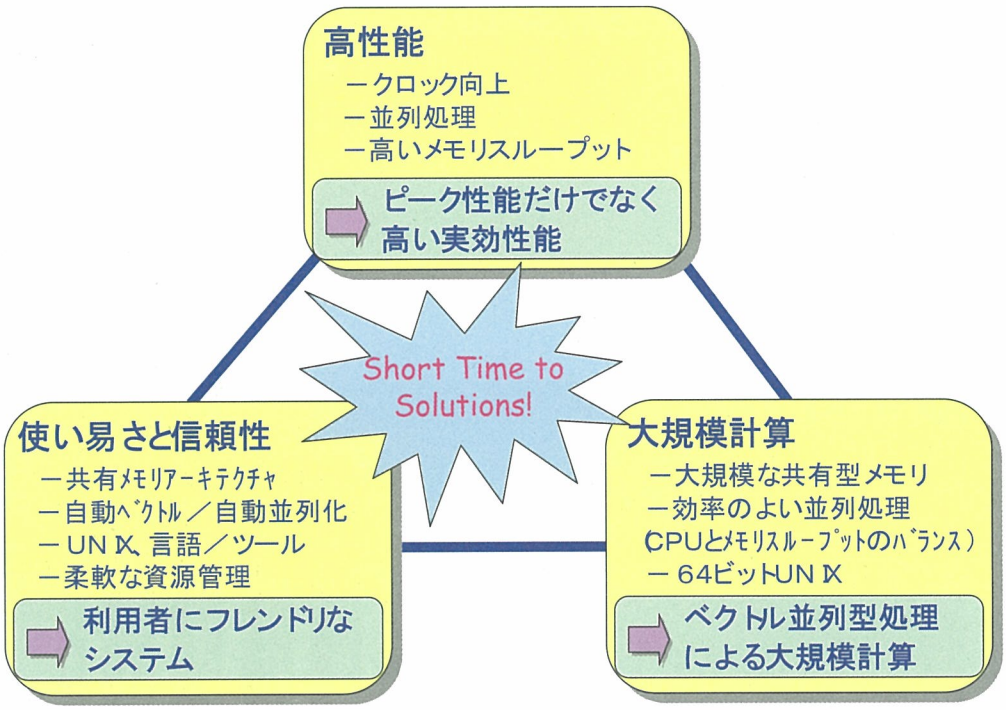


図 2.2.3　大規模科学計算システム SX-7 の設計における要求要件

一方，ベクトル処理を必要としない，あるいはベクトル化の効果が少ない計算に適したシステムとして，1998 年に並列演算サーバ(Exemplar)，2002 年にスカラ並列型コンピュータ TX7/AzusA を導入し，運用している（表 2.2.1）．スカラ並列型コンピュータでは，Gaussian をはじめとする標準的な数多くのアプリケーションも提供し，利用者がプログラムを開発することなしに，シミュレーションを行える環境を整えている．

表 2.2.1　情報シナジーセンタースカラ並列コンピュータの変遷

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

このように，情報シナジーセンターの大規模科学計算システムは，様々な研究者が必要とする多様な計算要求に応じて，ベクトル型スーパーコンピュータ，およびスカラ型並列コンピュータを適切に使い分けることができる万能型の異機種複合システムに仕上がっている．2003 年度から 2004 年度の CPU 利用状況は，ベクトル型スーパーコンピュータで年平均 90%，並列コンピュータで年平均 80%であり，特にベクトル型スーパーコンピュータでは，毎年 11 月から 2 月は常に 95%を超える CPU 利用率で活用されており，利用者の大規模科学計算システムへの期待が非常に大きいことを伺わせる．このことを端的に表すデータとして，図 2.2.4 に SX-7 の年間処理量を GFLOPS(Giga-Floating Point Operations)で求めたものを示す．極限処理量とは，保守などによりサービスを停止することを想定して年間サービス日を 340 日，24 時間運転，90%の CPU 利用率を想定して設定された提供可能な処理量の上限を与えている．グラフが示すように，処理量は毎年提供可能な極限処理量に達しており，さらにシステムの更新毎にはシステムの上限まで利用が一気に

増加することから，利用者は，提供されるスーパーコンピュータの性能に合わせてシミュレーションの規模，精度を設定していることがわかる．このような利用者の増え続ける計算需要に応えていくために，現在，2006年3月予定の並列コンピュータの更新に向けて仕様策定作業を行っており，並列コンピュータの大幅な性能向上に加え，ベクトル型スーパーコンピュータの処理能力増強も計画している．

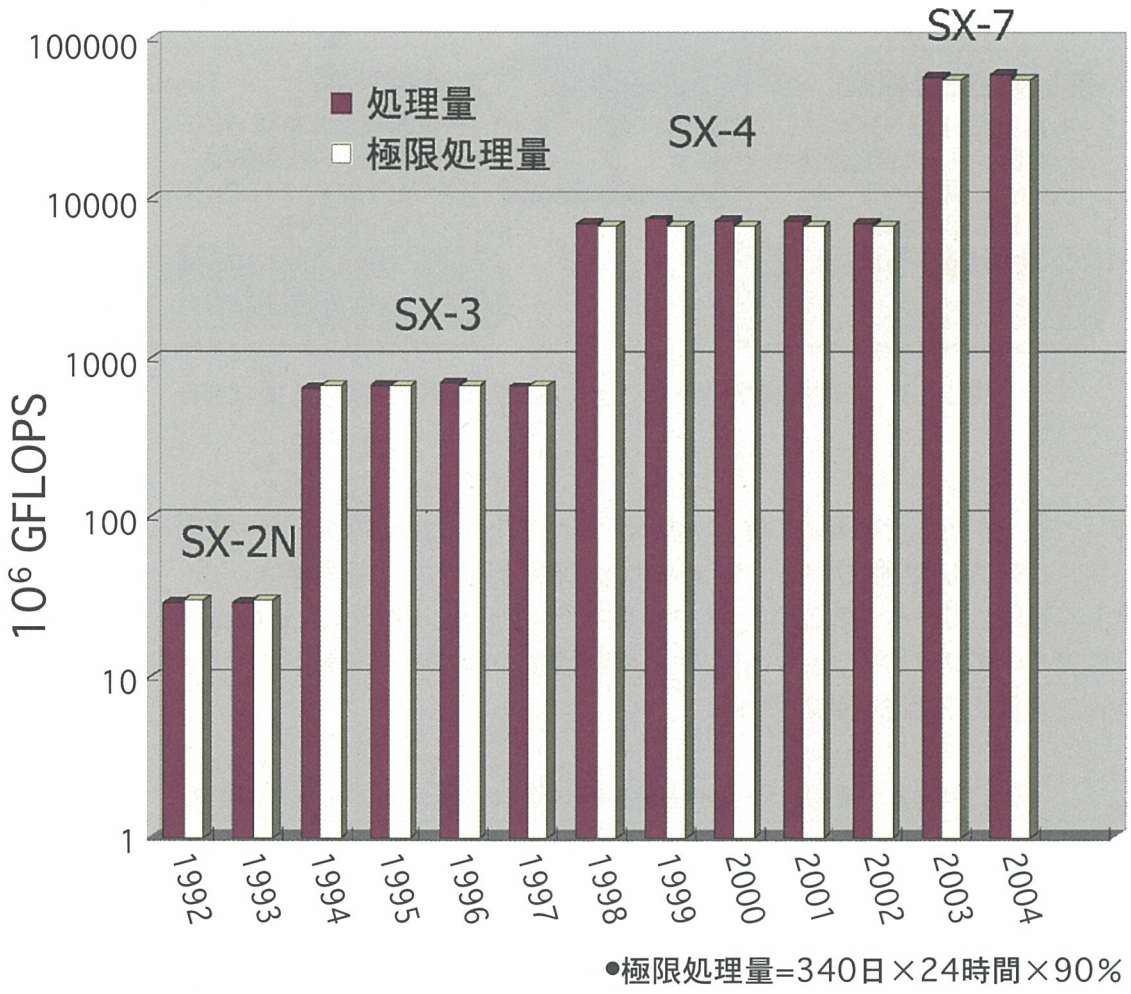


図 2.2.4　年間処理量

２．３　　高速化支援体制

スーパーコンピュータの性能・機能を極限まで引き出すためには，スーパーコンピュータに関する専門的知識を用いてプログラムをいろいろな角度から解析し，最適化する必要がある．従って，利用者プログラムに対する高速化支援は，高性能・高機能なスーパーコンピュータを整備・運用することと並んで極めて重要なサービスの１つであると我々は考えている．本センターでは，図 2.3.1 に示すように，センターの教員，技術職員，NEC の技術者が一体となり，利用者に対するプログラムの高速化支援体制を整え，利用者プログラムの解析，プログラム最適化指導，高速化技術の蓄積，講習会や出版等による高速化技術の社会への還元に取り組んでいる．

2003 年度は 8 人の利用者, 2004 年は 9 人の利用者と共同研究をそれぞれ実施し, 18 本(2003 年), 20 本(2004)のプログラムの解析・高速化に取り組み，平均単体性能 6.7 倍／平均並列性能 18.6 倍(2003 年)，平均単体性能 2.9 倍／平均並列性能 4.5 倍(2004 年)の成果を得た．その成果の一部は，本報告書の 3 章に示してある．

利用者プログラムの解析・高速化を通して得た技術力や研究開発成果は，利用者，センターの教員・技術職員，NEC の技術者と共有し，図 2.3.2 に示すような研究・開発サイクルを通してそれぞれの技術力を高めるとともに，得られた成果・知見を次期システムの設計・開発へと還元させ，常に高性能・高効率なスーパーコンピュータの整備・運用へとつなげている．

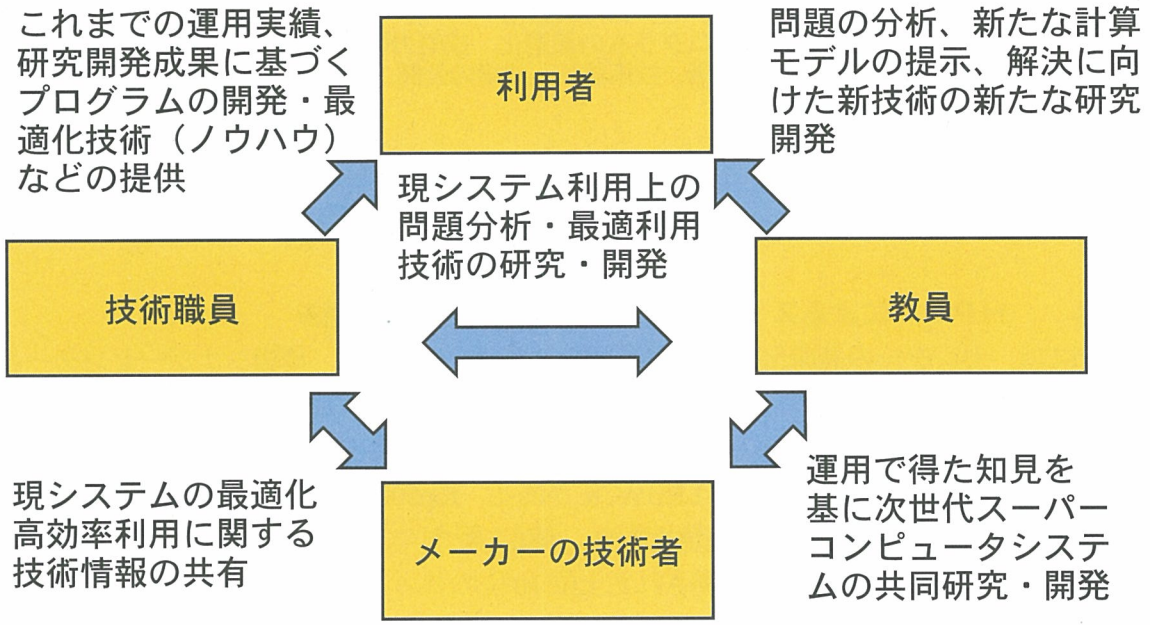


図 2.3.1　利用者プログラムの解析・高速化の支援体制

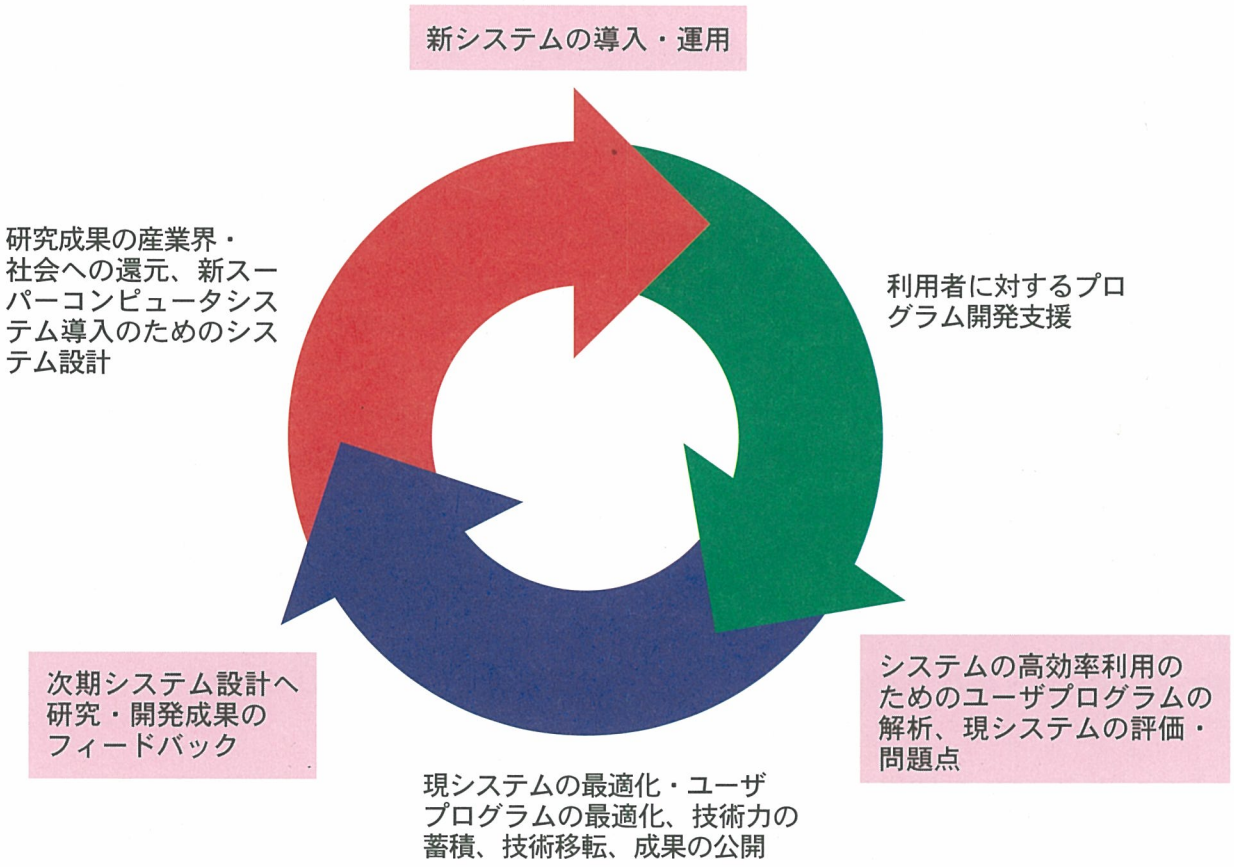


図 2.3.2　スーパーコンピュータの導入・整備・研究・開発サイクル

2．4　　ＨＰＣＣによるスーパーコンピュータ評価の取り組み

本節では，センターの高速化推進支援体制の成果の 1 つとして，情報シナジーセンターと NEC が共同で取り組んだ HPC チャレンジベンチマークによる SX-7 の評価について述べる．

スーパーコンピュータの性能評価の指標として国際的に有名なものに Top500 ランキング (http://www.top500.org/)で用いられる LINPACK がある．Top500 ランキングでは，LINPACK ベンチマークの実行時の最大浮動小数点演算性能(R_{max} 値)に基づいて HPC システムの性能を評価し，順位付けしている．2004 年 11 月に公開された上位 10 位のリストを表 2.4.1 に示す．このランキングでは，ついに IBM BlueGene/L および SGI Columbia が 2 年半首位を保っていた地球シミュレータの性能を超え，再び米国のこの分野での優位性を示すことになった．もう 1 つ特徴的なことは，商用プロセッサおよび汎用ネットワーク機器を用いた COTS (Commercial-Off-the-Shelf)ベースのクラスタ型システム（ピンク色の網掛けで表示．スカラ並列型に分類される．）がランキングの 6 割(全体では 500 システムのうち 294 台)を占め，ネットワーク等を専用設計されたシステムを加えればスカラ並列型スーパーコンピュータは全体として 96%にも達することである．一方，ランキングが開始された 1993 年当時は全体の 67%を占めていたベクトル型スーパーコンピュータは，2004 年 11 月時は全体の 4%までに減少している．汎用部品を多用するスカラ並列型スーパーコンピュータは，専用ハードウェアに基づくベクトル型スーパーコンピュータと比べ，カタログ性能当たりのコストが低いために，1995 年頃から HPC 市場で急成長し，それに伴って Top500 での割合を増

やしてきた．では，近年のこのようなTop500におけるCOTSベースのクラスタ型システム（スカラ並列型スーパーコンピュータ）の優位性は，ベクトル型スーパーコンピュータがもはやHPC分野で必要とされていないことを意味するのであろうか？

表2.4.1　2004年11月現在のTop500ランキング(上位10システム)

Rank	Manufacture	Computer	Rmax [TF/s]	Installation Site	Country	Year	# of Proc
1	IBM	BlueGene/L	70.72	DOE/IBM	USA	2004	32768
2	SGI	Columbia Altix, Infiniband	51.87	NASA Ames	USA	2004	10160
3	NEC	Earth-Simulator	35.86	Earth Simulator Center	Japan	2002	5120
4	IBM	MareNostrum BladeCenter JS20, Myrinet	20,.53	Barcelona Supercomputer Center	Spain	2004	3564
5	CCD	Thunder Itanium2, Quadrics	19.94	Laurence Livermore National Laboratory	USA	2004	4096
6	HP	ASCI Q AlphaServer SC, Quadrics	13.88	Los Alamos National Laboratory	USA	2002	8192
7	Self-Made	System X Apple XServe, Infiniband	12.25	Virginia Tech	USA	2004	2200
8	IBM/LLNL	BlueGene/L DD1 500MHz	11.68	Lawrence Livermore National Laboratory	USA	2004	8192
9	IBM	pSeries 655	10.31	Naval Oceanographic Office	USA	2004	2944
10	Dell	Tungsten PowerEdge, Myrinet	9.82	NCSA	USA	2003	2500

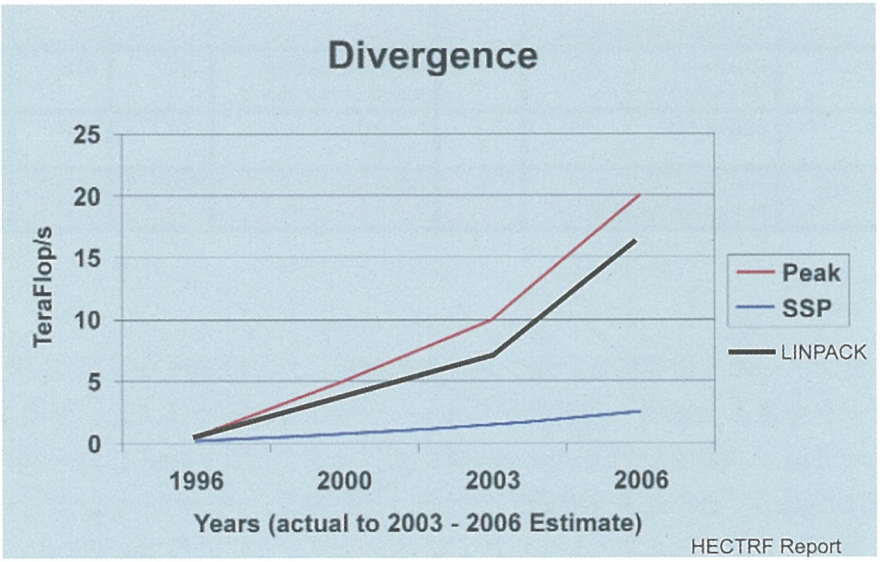
私たちは，センターの大規模科学計算システムの運用，およびHPCに関する研究・開発を通して，大量のデータを扱う大規模・高精度シミュレーションプログラムでは，ベクトル並列型スーパーコンピュータは，スカラ並列型スーパーコンピュータとは比べものにならないほど，高い実行効率で高速に処理ができることを認識している．それゆえ，スーパーコンピュータの非常に限られた性能を評価しているにすぎないLINPACKベンチマークによるTop500ランキングだけを盲目的に崇拝し，それをもってスーパーコンピュータの価値を議論することは，将来のスーパーコンピュータの研究開発の方向性を見誤らせると，常に警鐘をならしてきた．

LINPACKベンチマークは，プログラム自体が50行ほどの単純なプログラム構成であり，演算当たりのメモリ使用量，およびプロセッサ間の通信量も小さく，データの分割，および並列化が容易であるという特徴を持つ．従って，スカラ並列型スーパーコンピュータにおいてもプロセッサ数を増やしてもキャッシュが有効に働き，プロセッサ台数に比例した性能値をLINPACKベンチマークで得ることができる．その結果，大量のプロセッサを汎用ネットワークで相互接続した安価なクラスタ型システムでさえ，Top500の上位のスコアを得ることが可能になっている．

実際の研究開発で利用される実用的なシミュレーションプログラムの実行において，クラスタ型システムでは高い実効性能（ピーク性能に対するプログラム実行性能の比）を得ることが非常に難しく，LINPACKを用いたTop500ランキングは実効性能の観点からスーパーコンピュータを正しく評価していない．研究者が研究・開発する実用的なシミュレーションプログラムは，数万

行を超える複雑なプログラム構造を持ち，1)演算当たりのデータアクセス量が多い，2)データ参照の局所性が小さい，3)並列化のためにデータを分割すればするほどプロセッサ間通信が多くなる，などの特徴を有するのが一般的である．従って，高い演算処理能力を有するスーパーコンピュータであっても，演算性能に見合ったメモリ性能，およびネットワーク性能を有していなければ，プロセッサ(CPU)数を増加させても，ピーク性能に見合った性能向上を得ることは難しくなる．

同様なことは，米国国家科学技術会議（NSTC）の HECRTF がまとめた「米国政府の HEC 計画」（Federal Plan for High-End Computing）報告書でも議論され，スカラ並列型スーパーコンピュータを導入している米国スーパーコンピュータセンターのピーク性能と実効性能の著しい乖離を問題視している（図 2.4.1 参照, HECRTF の報告書のグラフに, LINPACK の性能を重ね合わせたもの）．図からわかるように，スーパーコンピュータのピーク性能は，ムーアの法則に示される半導体技術の進歩と超並列処理の台数効果により，年を追う毎に飛躍的に向上しているが，実用的なシミュレーションプログラムの実行を通して得られる実効性能は，それに見合って向上していない．結果として，図に示すような実効性能とピーク性能の乖離が近年著しくなっており，スーパーコンピュータの現実的な科学技術計算に対する実効性能をより適切に評価できる新しいベンチマークが強く求められている．



*SSP: Sustained System Performance

図 2.4.1　米国スーパーコンピュータセンターにおけるピーク性能と実効性能の乖離

このような背景のもと，HPC (High Performance Computing) チャレンジは(以下 HPCC)，総合的な HPC システム性能評価の試みとして，米国 DARPA (Defense Advanced Research Projects Agency) の HPCS (High-Productivity Computing Systems) プロジェクトの支援を受けて，Tennessee 大学の J. Dongarra 博士 により，2003 年 11 月 SC2003(2003 年 Supercomputing Conference)において提唱された HPC システムのベンチマークセットである．Linpack ベンチマークの単一性能指標のみで評価する Top500 を補完するものとして，より総合的で Linpack も含んだ 7 セットのベンチマークの集まりとなっている．このベンチマークセットでは，これまでスーパーコンピュータの性能評価で重要視されてきた総演算性能の評価に加えて，アプリケーション実行におけるスーパーコンピュータの実効性能を引き出す上で重要なメモリアクセス性能とネットワーク性能の評価と，多

くのアプリケーションで頻繁に使用されるカーネルコードを用いた性能評価が可能になっている．これにより，従来のノード数に頼るスーパーコンピュータの数量的な評価ばかりでなく，ノードの「質」を評価することが可能になる．

我々は，HPCC ベンチマークがスーパーコンピュータの実効性能を正しく評価するための試みであると高く評価し，

- 東北大学・NEC 共同で研究した成果としてベクトル型スーパーコンピュータの HPC 分野における優位性，および大規模科学計算に関する技術力の高さを示す
- スーパーコンピュータの評価のあり方について一石を投じる

ことを目的として，HPCC ベンチマークによる SX-7 の評価に取り組んだ．

図 2.4.2 に HPCC の 28 の評価項目における SX-7 のランキングを示す(評価結果の詳細は,「HPC チャレンジでの SX システムの性能評価」参照)．円チャートの 100%がその項目の最高性能を意味しており，SX-7 は，Cray, IBM, SGI, SUN の代表的なスーパーコンピュータ，および Opteron/Xeon ベースのクラスタ型システムの性能と比較して，16 の項目で世界最高性能値を達成した．表 2.4.2 は，Lazou が提唱した，オリンピックのメダル集計方式にならって，各項目の 1 位に金，2 位に銀，3 位に銅を与えた場合に，その獲得数を集計し，金，銀，銅の数でソートしたものである．これからわかるように，SX-7 は他のスカラ並列型スーパーコンピュータと比べて圧倒的に優れた性能を示している（SX-7 の 2 つのエントリは，2 種類の並列処理形態で測定したことによる)．

情報シナジーセンターのベクトル型スーパーコンピュータ SX-7 が HPCC で優れた性能値を達成することができたのは，1 ノード 32CPU の大きな SMP 型並列処理方式とベクトル型アーキテクチャだけが達成できる極めて高いメモリ性能のシナジー効果によるものである．このことは，ベクトル並列型スーパーコンピュータは HPC 分野において優位性があることを示し，最先端の科学技術分野の研究推進の重要な学術情報基盤であることを意味している．情報シナジーセンターの SX-7 の HPCC ベンチマーク評価結果に対し，HPCC プロジェクトリーダの J. Dongarra 博士は以下のようなコメントを寄せ，ベクトル並列型スーパーコンピュータを高く評価している．”We are impressed with the continuing high performance of the SX family of processors. The SX-7 lives up to the expectations.”

SX-7 の HPCC 評価結果の反響は大きく，日刊工業新聞，河北新報，朝日新聞，NHK ニュース／ラジオ，文部科学省科学技術政策研究所報告，HPCwire，supercomputingonline.com などで大きく取り上げられ，日本の HPC 分野におけるスーパーコンピュータ／スーパーコンピューティング技術力の高さを国内外に示すことができた．HPC チャレンジベンチマークによるスーパーコンピュータの評価は，今後，ベンダやユーザからのフィードバックを得ながら，スーパーコンピュータの総合的な評価指標として確立されていくと思われ，我々も多くの貢献をして行きたいと考えている．今後とも，全国共同利用施設である情報シナジーセンターは，日本の学術情報基盤を支えるという重要な役割を果たすことを最優先に考え，ベクトル並列型スーパーコンピュータを主軸とした大規模科学計算システムの整備・運用・研究・開発に全力を尽くす所存である．

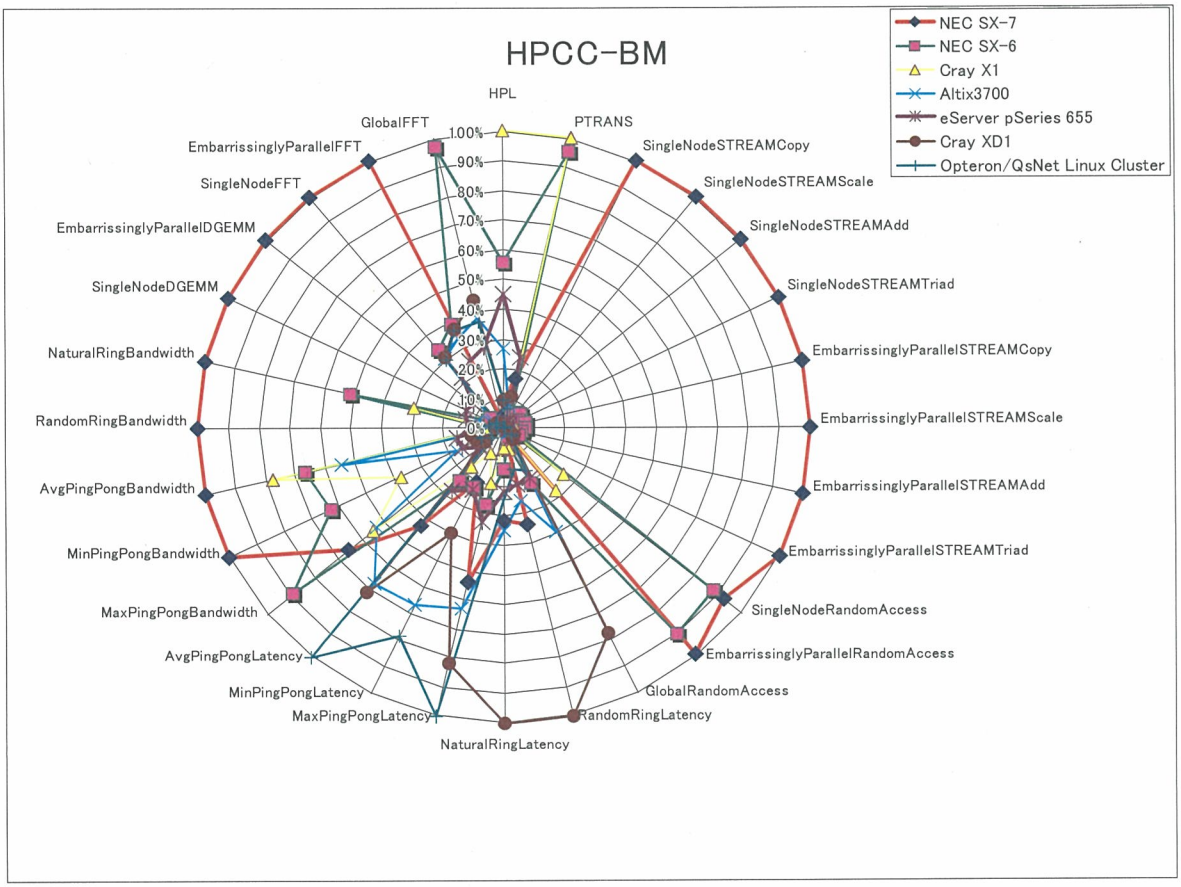


図 2.4.2　HPCC 評価結果(2004 年 11 月登録時)

表 2.4.2　HPC チャレンジによるランキング（オリンピック形式，2004 年 11 月登録時）

RANK	SYSTEM S	Gold	Silver	Bronze
1	SX-7 (16SMP/2MPI) 東北大	16	3	1
2	SX-6+(4node)	3	5	4
3	Cray XD1(64PE)	2	3	0
4	Opteron(64PE)	2	1	0
5	Cray X1(252PE base)	2	0	0
6	SX-6(16node)	1	1	3
7	PowerEdge(64PE)	1	1	0
8	T3E(512PE)	1	0	0
9	SX-7 (32MPI) 東北大	0	3	0
10	SX-6(24node)	0	2	3
11	Cray X1(252PE opt)	0	2	0
12	SX-6(4node)	0	1	5
13	SX-6(8node)	0	1	5

RANK	SYSTEMS	Gold	Silver	Bronze
14	Altix 3700(128PE)	0	1	1
15	eServer 655	0	1	1
16	Cray X1(60PE)	0	1	0
17	655(128PE/4SMP)	0	1	0
18	p690(64PE)	0	1	0
19	Altix(32PE)	0	0	2
20	Cray X1(32PE base)	0	0	1
21	Origin 3900(500PE)	0	0	1
22	655(64PE/4SMP)	0	0	1
23	Altix 3700(32PE)	0	0	1
24	Altix 3700(128PE)	0	1	1
25	eServer 655	0	1	1

3　高速化推進研究活動の成果

システム管理係　伊藤英一，大泉健治
システム運用係　小野敏

本センターのスーパーコンピュータシステムは，2003 年 1 月にスーパーコンピュータ SX-7 に変更され，それまでの SX-4 システムに比べてベクトル性能が 8 倍，メモリ容量が 32 倍と拡大した．SX-7 システムは，2003 年 1 月にサービスを開始してから 2005 年 3 月で 2 年 3 ヶ月を経過した．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・長時間化の傾向が一層強くなっている．

本センターでは，従来からスーパーコンピュータは，ジョブの大規模化・長時間化に対して的確に対応するため，1 ノード 32 台の CPU を一つのジョブで占有し使用できるように並列処理を運用の中心にして，さらに長時間ジョブは実行時間の推定が困難であるため CPU 時間を制限しないような利用環境を整備してきた．以前からも研究室では不可能な大規模・長時間ジョブの実行を可能としていたが，SX-7 システムに更新後もこの考え方をさらに推し進め，SX-4 では不可能と思われた大きな規模のシミュレーションを可能としている．

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きている．

3．1　高速化推進研究活動（２００３年度～２００４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 2003 年度（15 年度），表 3.1.2 に 2004 年度（16 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　2003 年度（15 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（配列宣言の変更） 重複演算の削減 ループ交換によるベクトル長の拡大 ベクトル化されたループの融合	9.5 倍	—
2	IF 節 ELSE 節で共通する演算の IF ブロック外への移動 ループ構造の変更と式の変形による演算数の削減 バンクコンフリクトの削減（ループ交換） 外側ループの融合による粒度の拡大	8.9 倍	16.2 倍

	並列化指示行 paralleldo による並列化促進		
3	バンクコンフリクトの削減（配列宣言の変更） ループ交換によるベクトル長の拡大	6.6 倍	25.1 倍
4	リストベクトルループのベクトル化	2.7 倍	—
5	インライン展開によるベクトル化	3.6 倍	—
6	IF 文の依存関係解消によるベクトル化 ポインタ配列を静的な宣言に変更	3.2 倍	22.4 倍
7	ループ一重化によるベクトル長の拡大 バンクコンフリクトの削減（配列宣言の変更）	1.3 倍	—
8	漸化式をベクトル化可能な形に変形	26 倍	—
9	三重ループの一重によるベクトル長の拡大 ループ交換によるベクトル長の拡大	1.63 倍	1.79 倍
10	リストベクトルループのベクトル化指示行 nodep によるベクトル化	6.2 倍	—
11	漸化式をベクトル化可能な形に変形 バンクコンフリクトの削減（配列宣言の変更）	3.8 倍	—
12	並列版 ASL に置き換え	—	2.1 倍
13	インライン展開によるベクトル化	3.5 倍	—
14	ループ交換によるベクトル長の拡大 スカラ変数に関する依存関係の除去による並列化	41.3 倍	118 倍
15	ループの一重化によるベクトル長の拡大 ベクトル化されたループの融合 重複演算の削減	6.9 倍	6.2 倍
16	ループ交換によるベクトル長の拡大 外側ループの融合による粒度の拡大 ループ交換による並列化の促進 重複演算の削減	2.6 倍	5.8 倍
17	ループ交換によるベクトル長の拡大 並列化指示行 paralleldo による並列化の促進 インライン展開	—	1.4 倍
18	マクロ演算が適用される形に式を変形 インライン展開 ループ交換によるベクトル長の拡大	1.7 倍	3.0 倍
19	配列の動的割当を静的割当に変更 ループ交換による並列バランスの向上	5.4 倍	70.6 倍 (16 並列)

表 3.1.2　2004 年度（16 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	ASL 並列版の適用 並列化指示行 paralleldo による並列化促進	—	5.2 倍
2	インライン展開	3.59 倍	—

3	インライン展開 作業配列を用いたベクトル化の促進 IF 節 ELSE 節で共通する演算の IF ブロック外への移動	1.2 倍	1.4 倍
4	ループ交換による並列バランスの向上 並列化指示行 paralleldo による並列化促進 粒度の小さいループの並列化抑止	—	2.4 倍
5	スカラ変数に関する依存関係の除去による並列化	—	5.0 倍
6	作業配列を用いたベクトル化の促進 ループ交換によるベクトル長の拡大 ループ交換の抑止による粒度の拡大	4.1 倍	6.7 倍
7	インライン展開 pow 関数を二乗に置き換えベクトル化促進 乱数生成をループ本体と分離することによるベクトル化	7.6 倍	—
8	ベクトル化されたループ融合によるロード／ストア回数の削減 ループの融合による粒度を拡大 インライン展開	1.2 倍	1.8 倍
9	ループ分割によるベクトル化	1.67 倍	—
10	重複演算の削減 インライン展開によるベクトル化促進 並列化指示行 paralleldo による並列化促進	—	22.0 倍
11	バンクコンフリクトの削減（配列宣言の変更） ループ交換によるベクトル長の拡大	1.8 倍	—
12	２重ループの１重化 重複演算の削減 WRITE 文をループ外に出すことによるベクトル化促進 COMMON 宣言による利用メモリ節約	1.8 倍	—
13	ループ交換によるベクトル化 ループ交換によるベクトル長の拡大	2.0 倍	2.5 倍
14	外側ループアンロールによるロード／ストア回数の削減 ループ交換によるベクトル長の拡大	1.1 倍	1.6 倍
15	手動インライン展開 並列化指示行 paralleldo による並列化促進	1.1 倍	2.3 倍
16	バンクコンフリクトの削減（配列宣言の変更）行列積ライブラリに適用される形に式を変形	4.5 倍	5.7 倍
17	外側ループアンロールによるロード／ストア回数の削減 ループ交換による並列バランスの向上	1.9 倍	2.0 倍
18	バンクコンフリクトの削減（配列宣言の変更）ループ融合による粒度の拡大	1.3 倍	1.2 倍
19	ループ交換によるベクトル長の拡大	2.5 倍	2.2 倍

	バンクコンフリクトの削減（配列宣言の変更）		
20	ループ一重化によるベクトル長の拡大 ストリップマイニング処理によるロード／ストア回数の削減 外側ループ交換による並列バランスの向上	6.9 倍	6.3 倍
21	ASL 並列版の適用 インライン展開 並列化指示行 paralleldo による並列化	1.1 倍	9.0 倍 (16 並列)

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 2003 年 1 月から 2004 年 12 月まで 2 年間の利用者ジョブのベクトル化率と並列化率の状況を表 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%	21	17	10	5	15	21	36	71	153	567
80%	2	1	2	1	3	0	7	3	3	18
70%	0	1	0	0	2	1	1	0	2	4
60%	1	0	0	0	0	0	0	0	0	1
50%	0	1	1	0	0	0	0	0	0	1
40%	0	3	1	0	0	0	0	0	0	2
30%	1	0	0	0	0	0	0	0	0	1
20%	0	1	0	0	0	0	0	0	0	2
10%	4	1	0	0	0	0	0	0	1	3
0%	1	1	0	0	0	0	0	0	1	3

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 57%
・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 76%

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 3.2.2 は 2002 年 1 月から 2004 年 12 月までの 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．2002 年(SX-4)は 100 時間以上のジョブで占める CPU 時間の割合は 84%であった．SX-7 に更新した 2003 年には，これが 92%に増加している．このことは，ノードあたりの性能が SX-4 の 4.4 倍の SX-7 に更新したことで，利用者のモデルがさらに大きくなったことを表わしている．この傾向は，2004 年も引き継がれており，高速化推進研究活動の重要性は一層高まっている．

表 3.2.2　CPU 時間使用分布

時間	2002 SX-4	2003 SX-7	2004 SX-7
0～　99	16%	8%	8%
100～ 999	37%	35%	37%
1000～1999	12%	18%	19%
2000～	35%	39%	36%

３．３　今後の取り組み

SX-7 システムの運用を開始してから 2 年 3 ヶ月を経過し，ジョブの大規模化，長時間化が一層進んでいる．3.2 節に述べたようにベクトル化率，並列化率はとも高いところに集中はしている．並列化率 80%未満のジョブの割合が 24%あり，今後は並列処理のチューニングに力を入ていく必要があると考えている．

４　ＳＸ－７高速化技法

スーパーコンピューティング研究部　岡部公起
システム管理係　伊藤英一
日本電気株式会社　撫佐昭裕，石井繭子，吉村健二，金野浩伸
NEC システムテクノロジー株式会社　曽我隆

４．１　ＳＸ－７の特長

SX-7 は，1 ノードあたり 32 個の CPU，256GB の主記憶を搭載し，282.5GFLOPS 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．SX-7 の CPU はベクトルユニットとスカラユニットからなり，主な特長は以下のとおりである．

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，加算／シフト演算，除算）を有し，1CPU あたり 8.83GFLOPS の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，256GB の大規模な主記憶を 32 個の CPU で共有することができる．
③ 主記憶と CPU 間のデータ転送は，1CPU あたり 35.32GB/S を実現している．
④ 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．
⑤ プログラムの性能解析を行うツールを有している．
⑥ SX-7 向けに高速化した数値計算ライブラリ ASL/SX を有している．

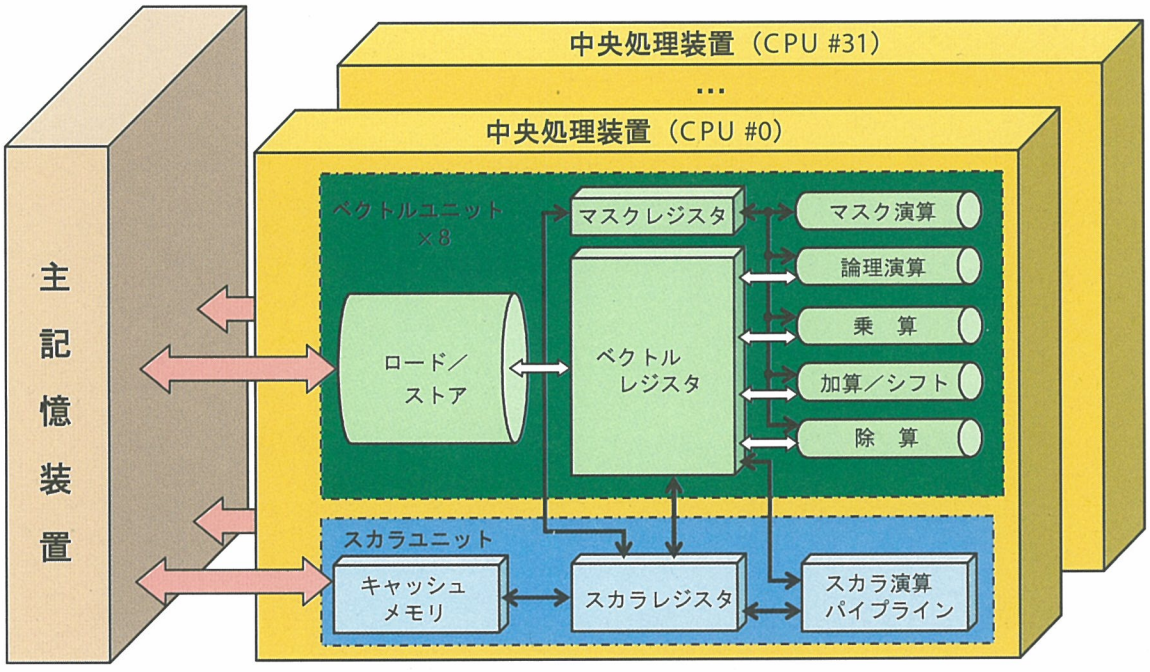


図 4.1.1　CPU の内部構造

表4.1.1　SX-7ノードの主要諸元

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

* 128bitはスカラ命令のみサポート

4．2　ベクトル処理

SX-7 における高速演算機能としてベクトル処理がある．SX-7 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．

SX-7 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，加算／シフト演算，除算）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-7 の高速演算を実現しているのである．

（1）ベクトル化率

プログラムを高速に実行するためには，SX-7 のベクトルユニットをできるだけ多く利用する必要がある．図 4.2.1 に示すようにプログラムのベクトル処理可能部分を増やすことによって，プログラムの実行性能が向上し，実行時間を短縮することができる．

４．３　並列処理

SX-7 における高速演算機能として並列処理がある．SX-7 では FORTRAN，C コンパイラの自動並列化機能によって，1 ノード 32CPU までの並列処理が可能である．ここでは高速演算を目的とした並列処理の概要を説明する．

並列化機能は一つのプログラムを複数の処理単位に分割し，それぞれを異なる CPU で同時実行することによってプログラムの実行時間（経過時間）を短縮するものである．図 4.3.1 は並列実行可能な do ループを分割し，4CPU を用いて経過時間の短縮を行っている概念図である．

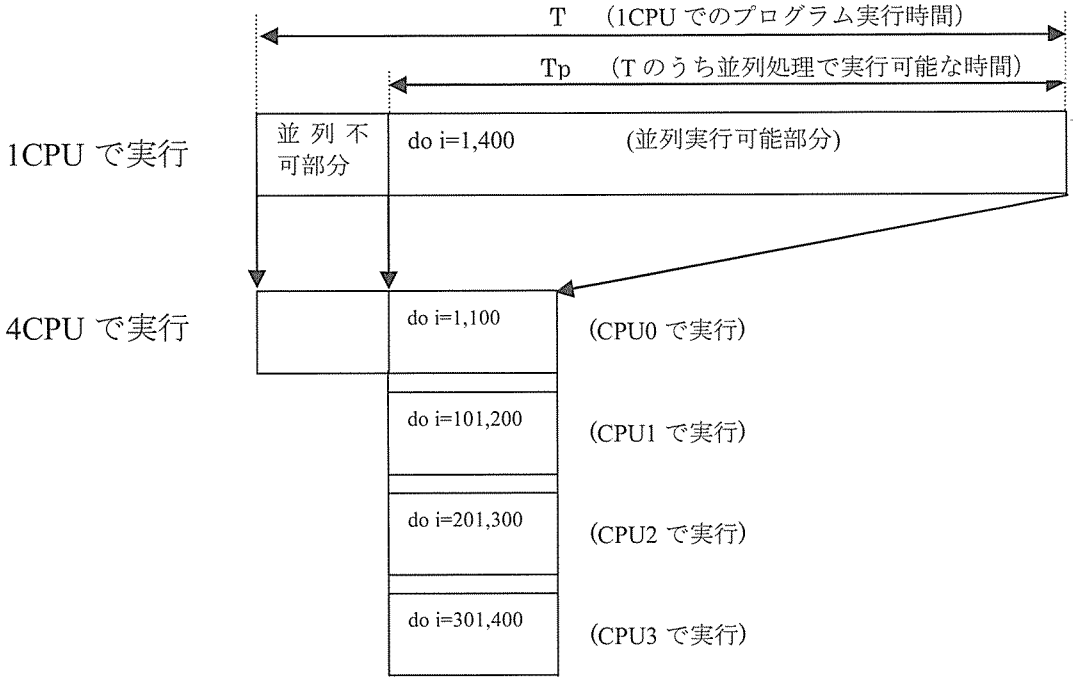


図 4.3.1　並列処理による実行時間短縮

（１）並列化率

並列処理においてプログラムをより短時間に処理するためには，図 4.3.1 に示した並列実行可能部分を増やす必要がある．

並列処理を行ったときの性能向上倍率は並列化率から求めることができる（アムダールの法則）．プログラム内における並列処理可能部分の割合を並列化率と呼び，図 4.3.1 中の T，Tp を用いて

$$並列化率　\alpha = Tp / T$$

である．並列化率 α のプログラムを n 台の CPU で演算を行ったときの理想的な性能向上倍率 P は，次のように表される．

$$P = 1 / ((1-\alpha) + \alpha / n)$$

図 4.3.2 に CPU8 台, 16 台, 32 台の性能向上倍率を示す. 並列性能を向上させるためには, 並列化率 95%以上が必要なことがわかる. また, 並列化率が低い場合には, CPU 台数を増やしても性能が変わらないこともわかる.

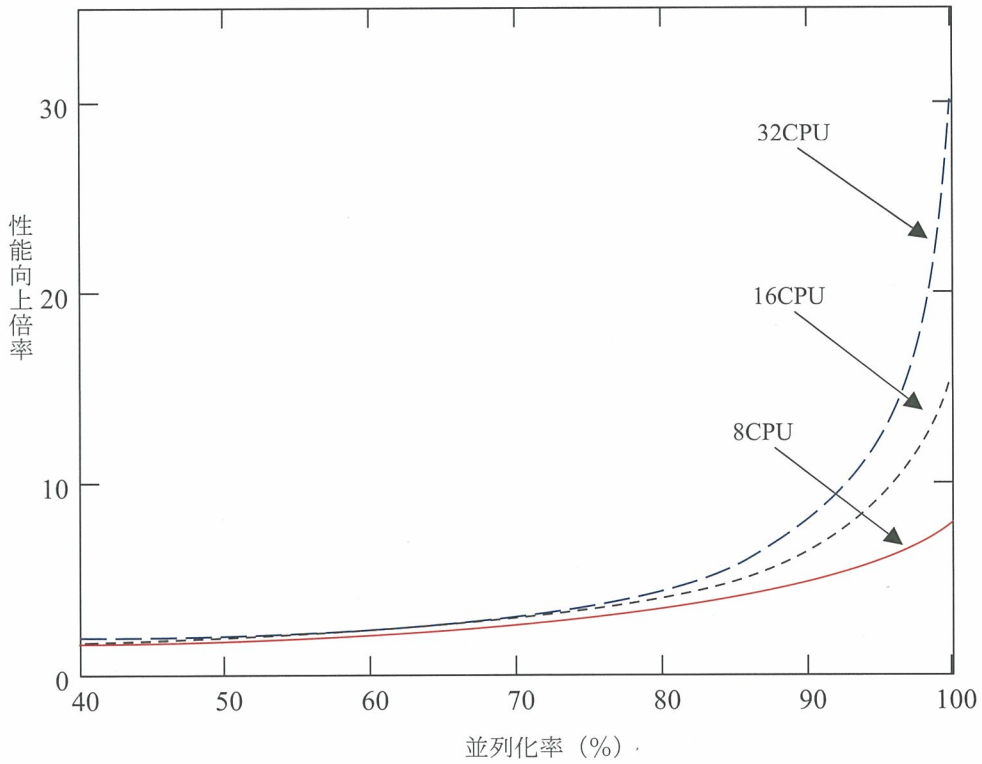


図 4.3.2　並列化率と性能倍率

（2）粒度

並列処理では, 並列実行するための制御処理（タスク生成やバリヤ同期など）の時間が必要になる. この時間をオーバヘッドという. 図 4.3.3 は並列処理の粒度とオーバヘッドの概念図である. 図の a, b は, それぞれ同じ演算量を異なった粒度（1 つの並列単位の演算量）で並列化した場合の例である. 図の a は, 粒度が大きい場合であり, オーバヘッドが少ない. b は粒度が小さく, 相対的にオーバヘッドが増加している. a, b を比較した場合, a は b よりオーバヘッドが少ない分, 実行時間が短くなっている.

並列処理では, 粒度が小さい場合, 並列化を行ってもオーバヘッドにより速くならないことがある. 並列処理のためには, 粒度を大きくし, オーバヘッドを削減することが必要である. 粒度を大きくするためには,

- 並列化する do ループに多くの演算を集める
- 多重 do ループの一番外側で並列化を行う

などの工夫が必要である. また, 粒度の小さな do ループは並列化をしないなどの処置も必要である.

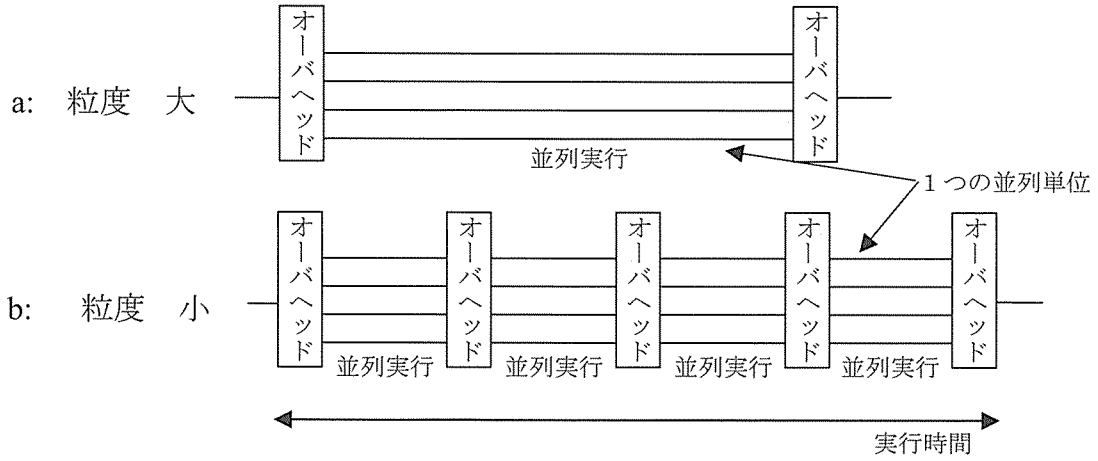


図 4.3.3 粒度とオーバヘッド

（3）負荷バランス

並列処理の効率を上げるためには，それぞれの CPU で実行する演算量（負荷バランス）が均等である必要がある．図 4.3.4 は負荷バランスの概念図である．図の a，b は，それぞれ同じ演算量を異なった負荷バランスで並列化した場合の例である．図の a は，並列処理を行う各 CPU に演算が均等に割り付けられた例で，効率的な並列処理を行っている．b は，CPU ごとにばらつきがあるので実行時間が長くなっている．

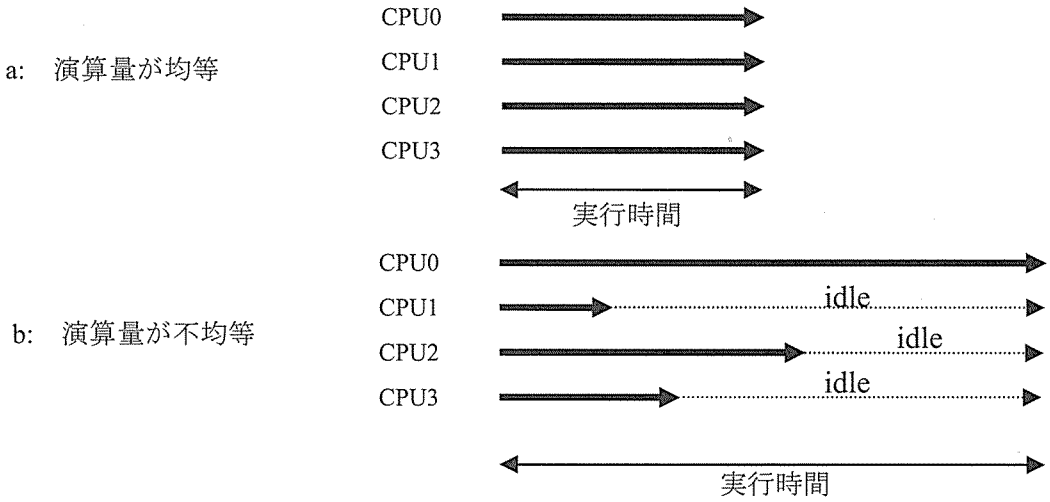


図 3.3.4 負荷バランス

SX-7 の FORTRAN，C コンパイラは， do ループの負荷バランスを調整するオプション（for=整数値，by=整数値）をサポートしている．コンパイラの既定値は，for=（使用できる CPU 数）である．図 4.3.5 に，for=4 と by=4 の例を示す．for=4 では，ループ長 100 の do ループが，ループ長 25 の do ループ 4 個に分割される．by=4 では，ループ長 100 の do ループが，ループ長 4 の do ループ 25 個に分割される．

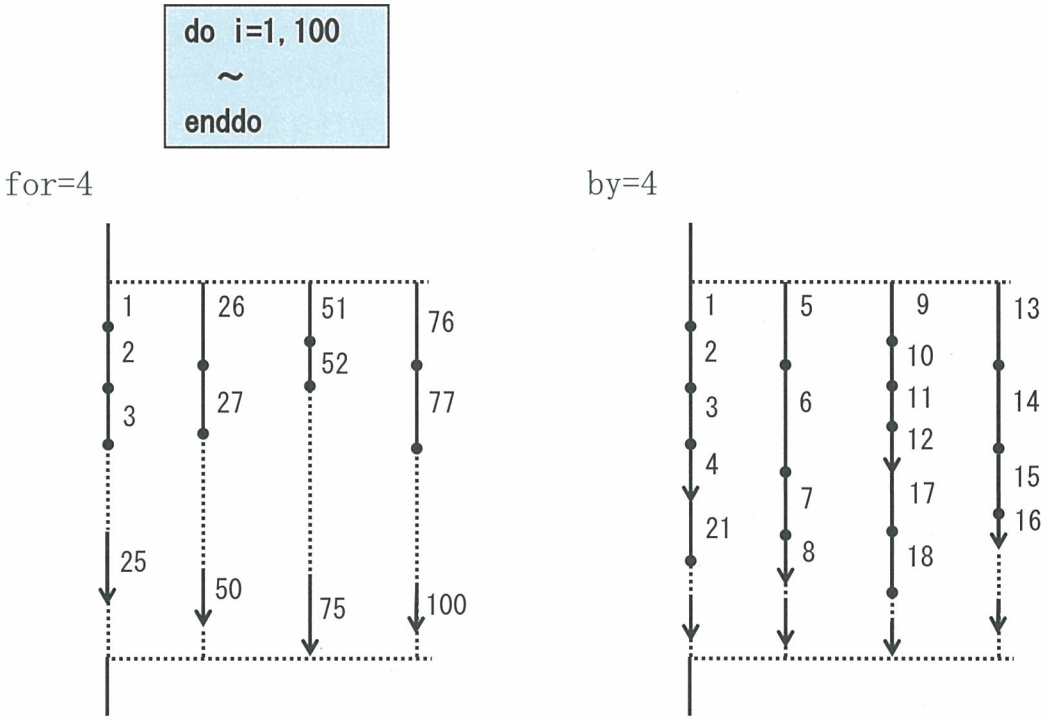


図 4.3.5　ループ長 100 のループを分割する場合

４．４　プログラムの解析

SX-7 のコンパイラは様々な最適化機能を有しており，SX-7 の性能を引き出すオブジェクトの生成を行う．しかし，プログラムによっては最適化が十分に行えない場合がある．そのような場合でも利用者が少しの工夫(チューニング)を行うだけで，高い性能を引き出せる可能性がある．ここでは，プログラム解析を行うために利用できる「編集リスト」，「プログラム情報」，および「ftrace 情報」を説明する．

（１）編集リスト

プログラムのどの部分がベクトル化，並列化されているかを知るためには，編集リストを採取する．編集リストは，コンパイル時にコンパイルオプション「-R5」を指定した場合に，「ソースファイル名.L」というファイル名で作成される．図 4.4.1 に編集リストの出力例を示す．

```
FILE NAME: list.f
PROGRAM NAME: sub
FORMAT LIST

    ①        ②        ③            ④
  LINE    LOOP          FORTRAN STATEMENT

   35:                        subroutine sub(a,b,c,d,x,z,ix)
   36:                        real,dimension(100)::a,b,c,d,x,y,z
   37:                        integer ix(100)
   38:  V------>              do I=1,100
   39:  |           I            call func(x,a,b,I)
   40:  |                        y(I)=c(I) + d(I)
   41:  |           S            z(ix(I))=z(ix(I))+x(I)+y(I)
   42:  V------               enddo
   43:                        end
```

図 4.4.1　編集リストの出力例

①　外部行番号
②　ループ情報
③　スカラ情報，インライン展開情報
④　原始プログラム

以下に主なループ情報，スカラ情報，インライン展開情報の出力イメージを示す．

- do ループ全体がベクトル化される場合

```
V------>            do i=1,100
|                      a(i)=b(i)+c(i)
V------             enddo
```

ベクトル化される do ループに「V」が表示される．

- doループが部分的にベクトル化される場合

```
V------>          do i=1,100
|                    a(i)=b(i)+c(i)
|       S            z(ix(i))=z(ix(i))+x(i)+y(i)
V------           enddo
```

ベクトル化できない行（スカラ処理される行）には「S」が表示される．

- ベクトル化されない場合

```
+------>          do i=1,100
|                    print*,a(i)
+------           enddo
```

ベクトル化されないdoループには「+」が表示される．

- 副プログラムの呼び出しがインライン展開される場合

```
        I         call sub
```

インライン展開される副プログラムがある行には「I」が表示される．

- 多重doループが一重化される場合

```
W------>          do j=1,100
|*----->          do i=1,100
||                   d(i,j)=d(i,j)+b(i,j)+c(i,j)
|*-----            enddo
W------           enddo
```

一重化されるdoループの外側ループには「W」，内側ループには「*」が表示される．

- doループが並列化される場合

```
P------>          do i=1,100
|                    :
|                    :
P------           enddo
```

並列化されるdoループには「P」が表示される．

- 多重doループにおいて，doループの入れ替えが行われる場合

```
X ------>         do i=1,100
|+----->          do j=1,20
||                   d(i,j)=d(i,j)+b(i,j)+c(i,j)
|+-----            enddo
X ------          enddo
```

doループの入れ替えが行われる場合，ベクトル化されるdoループに「X」が表示され，ベクトル化されないdoループには「+」が表示される．

（2）プログラム情報（Program Information）

プログラム全体の性能を得るには，プログラム情報(以下 Proginf)を採取する．本センターでは Proginf の採取を行う設定になっているので，プログラムを実行すると標準エラー出力に Proginf が出力される．これは，ハードウェア側で計測している情報を整理して出力されるだけなので，オーバヘッドはない． 図 4.4.2 に Proginf の出力例を示す．

```
******  プログラム 情報  ******
経過時間（秒）                :      6.774000
ユーザ時間（秒）              :      4.911325
システム時間（秒）            :      0.230647
ベクトル命令実行時間（秒）    :      4.846089
全命令実行数                  :    380150140.
ベクトル命令実行数            :    205713526.
ベクトル命令実行要素数        :  52468073111.
浮動小数点データ実行要素数    :  14758745137.
MOPS 値                       :  10718.595612
MFLOPS 値                     :   3005.043295
平均ベクトル長                :    255.054075 ①
ベクトル演算率（%）           :     99.668639 ②
メモリ使用量（MB）            :   1820.031250
MIPS 値                       :     77.402761
命令キャッシュミス（秒）      :      0.014333
オペランドキャッシュミス（秒）:      0.010467
バンクコンフリクト時間（秒）  :      0.244283 ③
```

図 4.4.2　Proginf の出力例

最も注目すべき項目は，①平均ベクトル長，②ベクトル演算率，③バンクコンフリクト時間である．

平均ベクトル長は，1 つのベクトル命令で演算を行った演算要素数の平均値である．SX-7 の場合，1 つのベクトル命令で最大 256 要素の演算を行うので，平均ベクトル長は 256 を超えることはないが，256 に近い値であるほど効率の高い演算を行うことができる．

ベクトル演算率は，ベクトル命令を使用して演算が行われた割合を示し，

$$\text{ベクトル演算率} = \frac{(\text{ベクトル命令実行要素数}) \times 100}{(\text{全命令実行数} - \text{ベクトル命令実行数} + \text{ベクトル命令実行要素数})}$$

となる．厳密にはベクトル化率とは言えないが，ほぼベクトル化率に近い値と考えて良い．したがってベクトル演算率が 100%に近づくほど高い性能を得ることができる．

バンクコンフリクト時間は，CPU がメモリアクセスを行う際にメモリバンクが競合して待ちになる時間である．全体時間に対するバンクコンフリクト時間の割合が高い場合は，メモリバンクの競合が発生する原因を究明し回避する手立てが必要となる．

並列処理時の Proginf には，CPU が同時に実行した時間(Concurrent Time)が併せて出力される．負荷バランスが各 CPU に均一であるときは，この数値が図 4.4.3 のように揃っ

た値となる.

```
最大同時実行可能プロセッサ数          :          4.
  1台以上で実行した時間（秒）         :      1.061972
  2台以上で実行した時間（秒）         :      1.058297
  3台以上で実行した時間（秒）         :      1.057302
  4台以上で実行した時間（秒）         :      0.945636
```

図 4.4.3　実行時の Concurrent Time の例

（3）ftrace 情報（FLOW TRACE ANALYSIS LIST）

Proginf ではプログラム全体の性能情報しか得られないため，プログラムの最適化を検討するサブルーチン，ループを絞り込むことが難しい．SX-7 のコンパイラの機能として，サブルーチンや関数といった副プログラム単位の性能情報を採取する ftrace がある．コンパイルオプションに-ftrace を追加したロードモジュールを実行すると，標準エラー出力に ftrace 情報が出力される．図 4.4.4 に ftrace 情報の採取方法を示す．

```
%sxf90 -ftrace test.f
%./a.out
```

図 4.4.4　ftrace 情報の採取方法

ftrace 情報は，Proginf と異なり，副プログラムの入口と出口で計測ポイントを設定するため，プログラム実行時にオーバヘッドが加わる．特に副プログラムの呼び出し回数が非常に多い場合や，副プログラムの呼び出しのネストが深い場合はプログラムの実行時間が長くなる場合がある．図 4.4.5 に ftrace 情報の出力例を示す．

```
①          ②          ③                    ④                              ⑤    ⑥                          ⑦
PROG.UNIT  FREQUENCY  EXCLUSIVE            AVER.TIME    MOPS  MFLOPS  V.OP  AVER.    VECTOR  I-CACHE  O-CACHE  BANK
                      TIME[sec](  %  )       [msec]                    RATIO V.LEN     TIME    MISS     MISS     CONF

cccc             128    90.620( 65.5)      707.966  6282.9  4645.1  99.58  256.0   88.793  0.0002   0.0008   0.0001
bbbb             100    33.221( 24.0)      332.209 11836.7  7722.6  99.81  250.0   33.209  0.0004   0.0063   0.0000
dddd              15    14.092( 10.2)      939.465  6057.3  1142.9  99.06  256.0   14.092  0.0000   0.0000   0.0000
main               1     0.291(  0.2)      291.189  7574.0  3694.6  98.44  252.7    0.291  0.0002   0.0002   0.0000
aaaa               1     0.039(  0.0)       39.028  1890.4   164.6  96.06  125.2    0.039  0.0000   0.0000   0.0000
-------------------------------------------------------------------------------------------------------------------
total            245   138.263(100.0)      564.338  7595.8  5024.3  99.62  253.7  136.424  0.0009   0.0072   0.0001
```

図 4.4.5　ftrace 情報の出力例

① PROG.UNIT…副プログラム (サブルーチンあるいは関数)名.
② FREQUENCY…副プログラムが実行された回数.
③ EXCLUSIVE TIME…副プログラムの実行された時間と全体時間からの割合.
④ AVER.TIME…副プログラムの一回当たりの実行時間.
⑤ V.OP RATIO…副プログラムのベクトル演算率.
⑥ AVER V.LEN…副プログラムの平均ベクトル長.
⑦ BANK CONF…副プログラムのバンクコンフリクト時間.

出力される順番は，副プログラムの実行時間の多いものからとなる.

ftrace 情報は並列化実行時にも採取することが可能であり，CPU ごとの情報を見ること

ができる．図 4.4.6 に，4CPU 利用して実行した場合の ftrace 情報の出力例を示す．図のとおり，並列化されたサブルーチンには，サブルーチン名の後ろに「$数字」が付く．CPU ごとの性能情報は，各サブルーチンの性能情報の下に，「-micro 数字」として表示される．

```
PROG.UNIT  FREQUENCY  EXCLUSIVE       AVER.TIME   MOPS MFLOPS V.OP  AVER.   VECTOR I-CACHE O-CACHE   BANK
                      TIME[sec](  %)    [msec]                 RATIO V.LEN    TIME   MISS    MISS     CONF

main_$8          120  109.173( 80.6)   909.778 11724.6 5294.1 99.82 254.9  109.152 0.0015  0.0016  0.0000
 -micro1          30   27.290( 20.2)   909.677 11725.9 5294.7 99.82 254.9   27.288 0.0007  0.0008  0.0000
 -micro2          30   27.294( 20.2)   909.815 11724.1 5293.9 99.82 254.9   27.288 0.0003  0.0003  0.0000
 -micro3          30   27.294( 20.2)   909.809 11724.2 5294.0 99.82 254.9   27.288 0.0003  0.0003  0.0000
 -micro4          30   27.294( 20.2)   909.813 11724.2 5293.9 99.82 254.9   27.288 0.0003  0.0003  0.0000
  :
```

図 4.4.6　並列化実行時の ftrace 情報の出力例

また，副プログラム単位だけではなく，指定した範囲の性能情報を見ることも可能である．ftrace_region_begin，ftrace_region_end に挟まれた範囲の情報が出力される．図 4.4.7 に ftrace 情報範囲指定方法の例を示す．

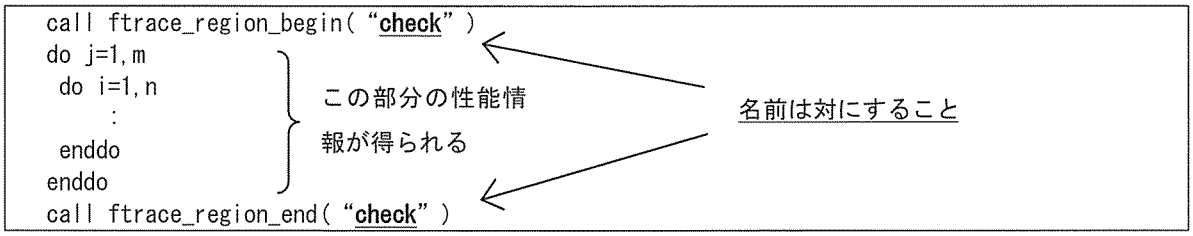


図 4.4.7　ftrace 情報範囲指定方法の例

これにより，do ループ単位にまで絞り込んで解析を行うことが可能になる．図 4.4.8 に ftrace 情報範囲指定の出力例を示す．

```
PROG.UNIT  FREQUENCY  EXCLUSIVE       AVER.TIME   MOPS MFLOPS V.OP  AVER.   VECTOR I-CACHE O-CACHE   BANK
                      TIME[sec](  %)    [msec]                 RATIO V.LEN    TIME   MISS    MISS     CONF

func2            128   93.898( 66.0)   733.581 6063.5 4482.9 99.58 256.0   92.085 0.0017  0.0026  0.0001
bbbb             100   33.301( 23.4)   333.012 11808.2 7703.9 99.81 250.0   33.290 0.0002  0.0075  0.0000
func1             15   14.668( 10.3)   977.854 5819.5 1098.1 99.06 256.0   14.668 0.0011  0.0013  0.0001
main               1    0.296(  0.2)   296.430 7440.1 3629.3 98.44 252.7    0.296 0.0003  0.0002  0.0000
aaaa               1    0.043(  0.0)    43.081 1712.5  149.1 96.06 125.2    0.043 0.0000  0.0000  0.0000
-------------------------------------------------------------------------------------------------------------
total            245  142.207(100.0)   580.436 7385.1 4885.0 99.62 253.7  140.381 0.0034  0.0116  0.0002

check            100   33.119( 23.3)   331.189 11833.6 7746.2 99.82 250.0   33.108 0.0002  0.0073  0.0000
-------------------------------------------------------------------------------------------------------------
```

図 4.4.8　ftrace 情報範囲指定の出力例

４．５　ベクトル化チューニング事例

４．５．１　入出力文を含む do ループのベクトル化

SX-7 では，do ループの一部にベクトル化対象外の文があると，ベクトル化されない．以下に，do ループ内からベクトル化の阻害要因の一つである入出力文の除去を行い，ベクトル化を促進した例を示す．

（１）性能の分析

図 4.5.1.1 に Proginf を示す．図中①に示すように，ベクトル演算率が 0.7%と低いことがわかる．

```
******  プログラム 情報  ******
経過時間（秒）                :    1130.708891
ユーザ時間（秒）              :    1096.143670
システム時間（秒）            :       1.267607
ベクトル命令実行時間（秒）    :       0.256928
全命令実行数                  :  349195261811.
ベクトル命令実行数            :       9008178.
ベクトル命令実行要素数        :    2290285204.
浮動小数点データ実行要素数    :   81404252932.
MOPS 値                       :     320.648240
MFLOPS 値                     :      74.264218
平均ベクトル長                :     254.245110
ベクトル演算率（%）           :       0.651618  ①
メモリ使用量（MB）            :      64.031250
MIPS 値                       :     318.567056
命令キャッシュミス（秒）      :       0.101621
オペランドキャッシュミス（秒）:     128.705834
バンクコンフリクト時間（秒）  :       0.000129
```

図 4.5.1.1　Proginf

図 4.5.1.2 に ftrace 情報を示す．図中①に示すように，コストのほぼ 100.0%を占めるサブルーチン func のベクトル演算率が 0.0%であり，ほとんどベクトル化されていないことがわかる．

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
func	100	1095.888(100.0)	10958.877	318.6	73.5	0.00 ①	69.2	0.003	0.1007	128.7055	0.0001
main	1	0.254(0.0)	254.169	9072.2	3435.5	99.22	255.0	0.254	0.0001	0.0001	0.0000
total	101	1096.142(100.0)	10852.890	320.6	74.3	0.65	254.2	0.257	0.1008	128.7055	0.0001

図 4.5.1.2　ftrace 情報

図 4.5.1.3 にサブルーチン func の編集リストを示す. 図に示すように do ループ内に write 文が含まれている. 入出力文はベクトル化対象外の処理であるため, 図中「+」記号が示すように, do ループ全体がベクトル化されていない.

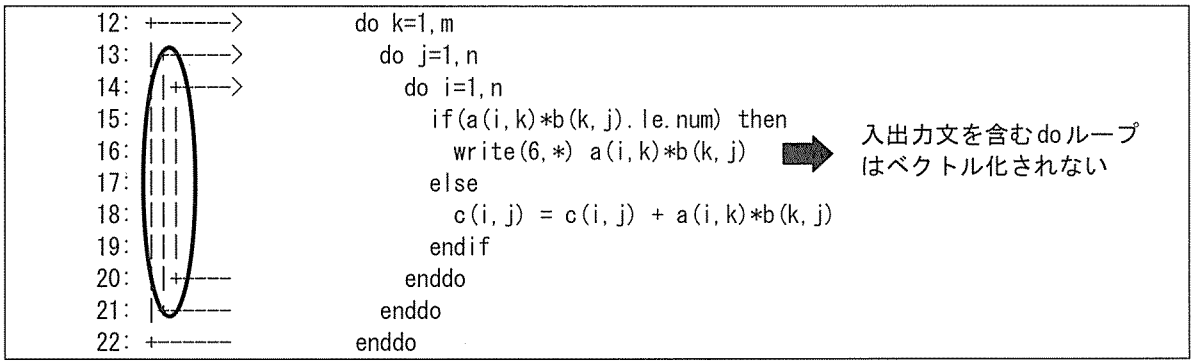


図 4.5.1.3　サブルーチン func の編集リスト

（2）入出力文の do ループ外への移動

do ループ内から, ベクトル化対象外の文である write 文を別の do ループに切り分けることにした.

図 4.5.1.4 にソース修正後の編集リストを示す. 図に示すように, 作業配列 wk を設け, write 文に渡すための値はその中に格納した. これにより, do ループ内にはベクトル化対象外の文がなくなり, 図中「V」記号が示すように, ベクトル化されるようになった.

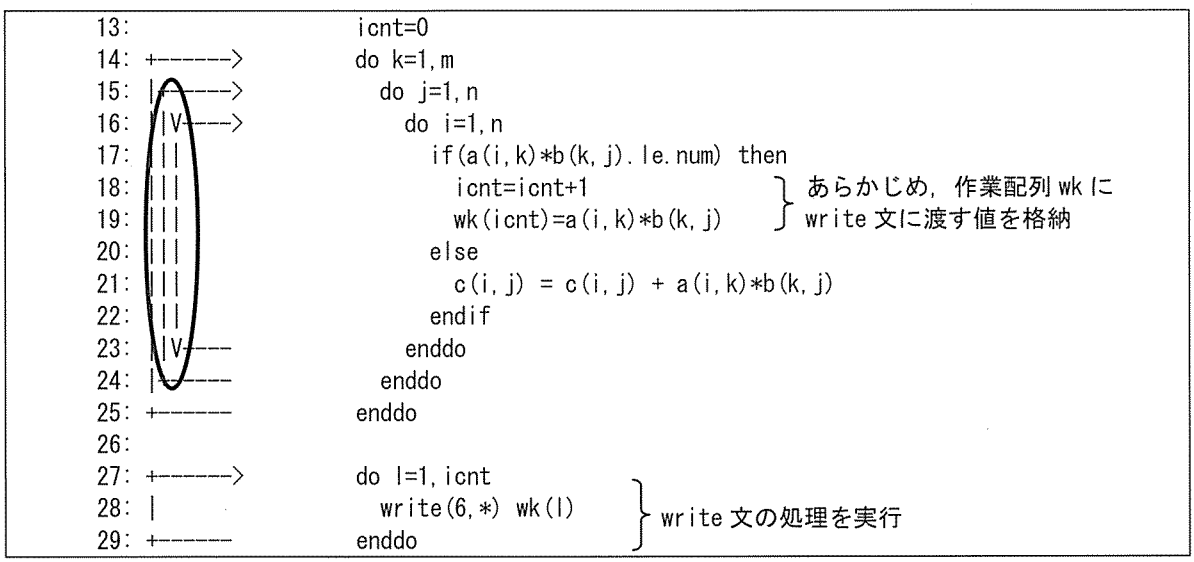


図 4.5.1.4　ソース修正後の編集リスト

（３）ソース修正による効果

図 4.5.1.5 にソース修正後の ftrace 情報を示す．図に示すように，サブルーチン func のベクトル演算率が 99.1%に向上し（修正前は 0.0%），プログラム全体のベクトル演算率は 99.1%に向上した（修正前は 0.7%）．また，実行時間は 48.4 秒に短縮された（修正前は 1096.1 秒）．

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
func	100	48.137(99.5)	481.373	4501.9	1673.0	99.10	256.0	43.210	0.0256	0.3238	0.0000
main	1	0.254(0.5)	253.695	9089.1	3441.9	99.22	255.0	0.254	0.0001	0.0000	0.0000
total	101	48.391(100.0)	479.119	4526.0	1682.2	99.10	256.0	43.464	0.0257	0.3238	0.0000

図 4.5.1.5　ソース修正後の ftrace 情報

４．５．２　副プログラムを含む do ループのベクトル化

副プログラムの引用(CALL 文や関数呼び出し)を含む do ループはベクトル化できない．この場合，副プログラムをインライン展開することによりベクトル化が可能になる．インライン展開はコンパイラオプション-pi によって自動的に行われるが，自動インライン展開阻害要因がある場合，手動でソース修正を行う必要がある．以下に手動でインライン展開を行った例を示す．また，自動インライン展開阻害要因の詳細については 4.5.8 章(4)で紹介する．

（１）性能の分析

図 4.5.2.1 に Proginf を示す．図中①に示すように，ベクトル演算率が 2.1%と低いことがわかる．

```
******  プログラム 情報  ******
経過時間（秒）                :    1188.872381
ユーザ時間（秒）              :    1183.763346
システム時間（秒）            :       2.394112
ベクトル命令実行時間（秒）    :       2.296196
全命令実行数                  :  230944773788.
ベクトル命令実行数            :      38487182.
ベクトル命令実行要素数        :    5048783549.
浮動小数点データ実行要素数    :    1259607979.
MOPS 値                       :     199.326218
MFLOPS 値                     :       1.064071
平均ベクトル長                :     131.180910
ベクトル演算率（%）           :       2.139722  ①
メモリ使用量（MB）            :    4848.031250
MIPS 値                       :     195.093702
命令キャッシュミス（秒）      :       0.993977
オペランドキャッシュミス（秒）:     149.308159
バンクコンフリクト時間（秒）  :       0.000010
```

図 4.5.2.1　Proginf

図 4.5.2.2 に ftrace 情報を示す．①のようにサブルーチン sub のベクトル演算率は 0.0%であり，②のようにサブルーチン main のベクトル演算率は 29.82%である．よって，ともにほとんどベクトル化されていないことがわかる．

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	209715200	125.325(59.1)	0.001	47.5	5.0	0.00 ①	0.0	0.000	0.0615	41.6679	0.0000
main	1	86.776(40.9)	86776.407	195.1	7.3	29.82 ②	131.2	2.296	0.1341	2.0274	0.0000
total	209715201	212.102(100.0)	0.001	107.9	5.9	22.06	131.2	2.296	0.1956	43.6953	0.0000

図 4.5.2.2　ftrace 情報

図 4.5.2.3 に編集リストを示す．図中の「+」記号が示すように，call 文を含む do ループはベクトル化されていない．また，呼び出される側のサブルーチン sub には do ループがないのでベクトル化されない．

```
    1:                  program main
...
    9:
   10: +-------->        do j=1,m
   11: |+------->           do i=1,n
...                          ...
   16: ||                    call sub(a(i,j),b(j,i),x(i,j))
...                          ...
   22: |+-----              enddo
   23: +-------           enddo
...
   30:                  subroutine sub(a,b,x)
   31:                          x = a*b
   32:                        if(x.le.0.25) then
   33:                          x = x + 0.25
   34:                        else
   35:                          x = x - 0.25
   36:                        endif
   37:                  end
...
```

call 文を含む do ループはベクトル化されない

call 文により呼び出されるサブルーチン sub

図 4.5.2.3　編集リスト

（2）インライン展開

図 4.5.2.4 にソース修正後の編集リストを示す．call 文を含む do ループをベクトル化するために，サブルーチン sub をインライン展開した．インライン展開を行ったことにより，図中の「V」記号が表すように do ループの内側がベクトル化されるようになった．

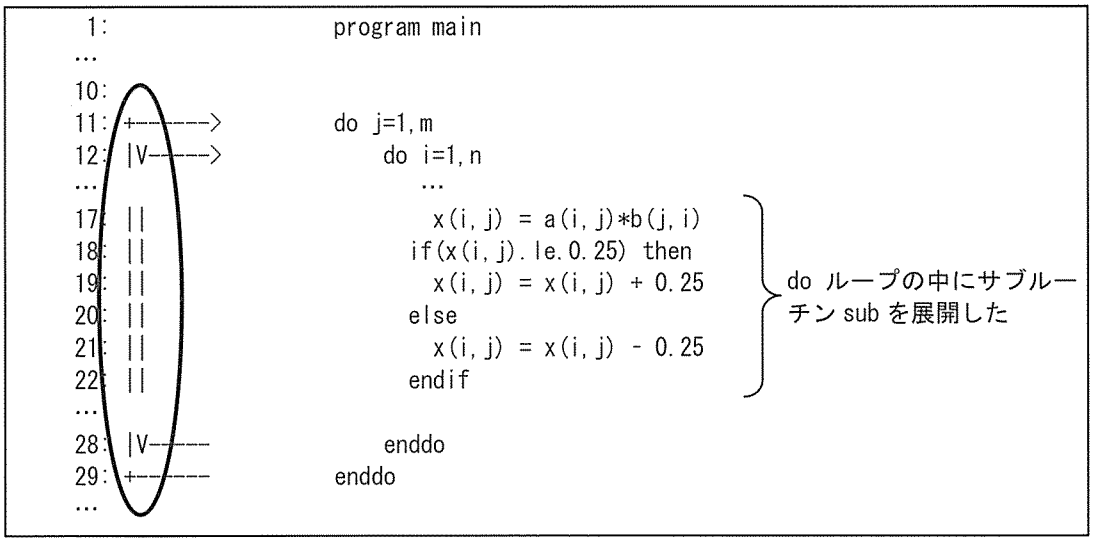


図 4.5.2.4　ソース修正後の編集リスト

（3）インライン展開の効果

図 4.5.2.5 にソース修正後の ftrace 情報を示す．インライン展開により sub は main に吸収されていることがわかる．

図中②に示すように，プログラム全体のベクトル演算率は 98.0%に向上した（修正前は 22.1%）．この結果，図中①のように全体の実行時間は 6.8 秒に短縮された（修正前は 212.1 秒）．

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
main	1	6.773(100.0)	6773.252	1108.1	216.9	98.00	154.9	6.773	0.0006	0.0005	3.6073
total	1	6.773(100.0) ①	6773.252	1108.1	216.9	98.00 ②	154.9	6.773	0.0006	0.0005	3.6073

図 4.5.2.5　ソース修正後の ftrace 情報

４．５．３　重なりのあるリストベクトルのベクトル化

定義・参照を行っている配列の添え字が間接参照（リストベクトル）の場合，添え字の重なりの有無をコンパイラは判断することができず，ベクトル化を行うことができない．添え字に重なりがないことがわかっていればNODEP 指示行を挿入することでベクトル化できるようになる．NODEP は配列の左辺と右辺に依存関係がないことを指定する指示行である．

添え字に重なりがあるかどうかわからない場合，止まり木の手法を用いてソースを修正することでベクトル化できるようになる．

以下に，止まり木の手法を用いたベクトル化の例を示す．

（1）　性能の分析

図 4.5.3.1 に Proginf を示す．図中①に示すように，ベクトル演算率が 84.9%と低いことがわかる．

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_a	97659	60.365(45.6)	0.618	265.7	91.4	61.20	39.0	16.887	0.0545	12.1005	0.0006
sub_c	200	32.624(24.7)	163.119	299.6	28.2	22.38	209.6	1.733	0.0608	2.3870	0.0000
sub_b	603	14.673(11.1) ①	24.335	5350.5	1254.7	99.80 ②	255.6 ③	14.505	0.0135	0.0084	0.6498
sub_d	401	9.378(7.1)	23.385	6561.4	2059.6	99.80	255.9	9.374	0.0042	0.0022	1.2822
total	643807	132.303(100.0) ④	0.190	1453.3	445.9	91.55	183.3	52.999	0.2737	14.6271	2.1547

図 4.5.3.5　ソース修正後の ftrace 情報

４．５．４　ベクトル長の拡大

平均ベクトル長が小さいと，図 4.2.3 に示すようにベクトル処理の効果が低く，実行時間が長くなる．以下に，平均ベクトル長を拡大し，ベクトル処理の効率を上げた例を示す．

（１） 性能の分析

図 4.5.4.1 に Proginf を示す．図中①に示すように，平均ベクトル長は 129.9 であり，十分に大きいとは言えない．

******　プログラム　情報　******			
経過時間（秒）	:	337.754345	
ユーザ時間（秒）	:	329.712521	
システム時間（秒）	:	0.094685	
ベクトル命令実行時間（秒）	:	308.082955	
全命令実行数	:	35179138465.	
ベクトル命令実行数	:	10413430205.	
ベクトル命令実行要素数	:	1352288967785.	
浮動小数点データ実行要素数	:	365195719423.	
MOPS 値	:	4176.531331	
MFLOPS 値	:	1107.618594	
平均ベクトル長	:	129.860088	①
ベクトル演算率（%）	:	98.201545	
メモリ使用量（MB）	:	48.031250	
MIPS 値	:	106.696398	
命令キャッシュミス（秒）	:	0.005801	
オペランドキャッシュミス（秒）	:	0.013908	
バンクコンフリクト時間（秒）	:	0.000002	

図 4.5.4.1　Proginf

図 4.5.4.2 に ftrace 情報を示す．図中①に示すように，サブルーチン solve はコストの 100.0%を占めており，②のようにその平均ベクトル長は 129.9 である．

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
solve	2000	326.709(100.0) ①	163.354	4214.9	1117.8	98.20	129.9 ②	306.012	0.0066	0.0107	0.0000
output	1	0.044(0.0)	43.716	262.6	7.7	3.27	12.0	0.002	0.0002	0.0005	0.0000
main	1	0.002(0.0)	1.773	56.5	0.0	0.00	0.0	0.000	0.0004	0.0004	0.0000
input	1	0.000(0.0)	0.211	762.5	37.0	89.58	51.2	0.000	0.0000	0.0000	0.0000
total	2003	326.755(100.0)	163.133	4214.3	1117.6	98.20	129.9	306.014	0.0073	0.0116	0.0000

図 4.5.4.2　ftrace 情報

図 4.5.4.3 にサブルーチン solve の編集リストを示す．図中の「V」記号が表すように，多重ループの最内ループがベクトル化の対象になっている．最内ループのループ長は nmax-1=30 であるにも関わらず，solve の平均ベクトル長が 30 ではなく図 4.5.4.2 の②に示すように 129.9 になっているのは，最内ループ内の演算をコンパイラが総和型のマクロ演算に置き換えているためである．マクロ演算への置き換えは，図 4.5.4.3 の上部に示したコンパイルメッセージよりわかる．

マクロ演算への置き換えにより平均ベクトル長が大きくなってはいるが，129.9 ではまだ十分な大きさではなく，演算性能を上げるためにはベクトル長の拡大が必要となる．

```
f90: vec(1): sample554.f, line 49: ループ全体をベクトル化する
f90: vec(26): sample554.f, line 50: Sum/InnerProd のマクロ演算としてベクトル化を行う
f90: vec(26): sample554.f, line 52: Sum/InnerProd のマクロ演算としてベクトル化を行う
f90: vec(26): sample554.f, line 54: Sum/InnerProd のマクロ演算としてベクトル化を行う
f90: vec(26): sample554.f, line 56: Sum/InnerProd のマクロ演算としてベクトル化を行う


    3:                    parameter(mmax=63,nmax=31)
   ...
   46: +------>           do m=1,mmax
   47: |+----->            do k=1,nmax
   48: ||+---->             do j=1,mmax-1
   49: |||V--->              do i=1,nmax-1
   50: |||                    ph1(k,m)=ph1(k,m)
   51: |||                +             +chg1(k,m)*engy(i,   j  )+chg2(k,m)*dens(i,   j  )
   52: |||                    ph2(k,m)=ph2(k,m)
   53: |||                +             +chg1(k,m)*engy(i+1, j  )+chg2(k,m)*dens(i+1, j  )
   54: |||                    ph3(k,m)=ph3(k,m)
   55: |||                +             +chg1(k,m)*engy(i,   j+1)+chg2(k,m)*dens(i,   j+1)
   56: |||                    ph4(k,m)=ph4(k,m)
   57: |||                +             +chg1(k,m)*engy(i+1, j+1)+chg2(k,m)*dens(i+1, j+1)
   58: |||V---              enddo
   59: ||+-----            enddo
   60: |+------           enddo
   61: +-------          enddo
```

図 4.5.4.3　サブルーチン solve の編集リスト

（２）ベクトル長の拡大方法

図 4.5.4.4 にソース修正後の編集リストを示す．図 4.5.4.3 中の外側 2 つのループを融合し，さらに，融合したループを内側ループと入れ替えることでベクトル化の対象としている．ループを融合することができるのは，ループ内にある配列 ph1,ph2,ph3,ph4 の添え字 k が 1 から nmax まで，添え字 m は 1 から mmax まで，いずれも連続して参照されているため，k*m を 1 つの変数 km として扱うことができるためである．

```
46: +------>          do j=1,mmax-1
47: |+----->           do i=1,nmax-1
48: ||V---->            do km=1,nmax*mmax
49: |||  .                ph1(km,1)=ph1(km,1)
50: |||                +          +chg1(km,1)*engy(i,  j  )+chg2(km,1)*dens(i,  j  )
51: |||                   ph2(km,1)=ph2(km,1)
52: |||                +          +chg1(km,1)*engy(i+1,j  )+chg2(km,1)*dens(i+1,j  )
53: |||                   ph3(km,1)=ph3(km,1)
54: |||                +          +chg1(km,1)*engy(i,  j+1)+chg2(km,1)*dens(i,  j+1)
55: |||                   ph4(km,1)=ph4(km,1)
56: |||                +          +chg1(km,1)*engy(i+1,j+1)+chg2(km,1)*dens(i+1,j+1)
57: ||V----            enddo
58: |+-----           enddo
59: +------          enddo
```

図 4.5.4.4　ソース修正後の編集リスト

（３）ループ融合およびループ入れ換えの効果

図 4.5.4.5 にソース修正後の ftrace 情報を示す．図中②に示すようにサブルーチン solve の平均ベクトル長は 244.1 に改善された(修正前は 129.9)．これは，コンパイラによる総和型のマクロ演算への置き換えよりもループ融合の方がベクトル長拡大の効果が大きかったことを表している．これにより，図中①に示すように，実行時間は 21.8 秒に短縮された（修正前は 326.7 秒）．

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
solve	2000	21.798(99.8) ①	10.899	8689.7	5332.8	99.73	244.1 ②	21.795	0.0021	0.0029	0.0002
output	1	0.045(0.2)	44.785	256.4	7.5	3.27	12.0	0.005	0.0003	0.0005	0.0000
main	1	0.002(0.0)	1.782	56.2	0.0	0.00	0.0	0.000	0.0005	0.0003	0.0000
input	1	0.000(0.0)	0.263	611.2	29.7	89.58	51.2	0.000	0.0000	0.0000	0.0000
total	2003	21.844(100.0)	10.906	8671.6	5321.4	99.72	244.1	21.800	0.0029	0.0038	0.0002

図 4.5.4.5　ソース修正後の ftrace 情報

４．５．５　バンクコンフリクトの回避

バンクコンフリクトはSX-7の演算性能を低下させる要因である．これは，配列データのアクセス方法によって発生する場合がある．以下に，配列の使い方の変更によってバンクコンフリクトを回避した例を示す．

（１）性能の分析

図4.5.5.1にProginfを示す．図中①に示す平均ベクトル長，および②に示すベクトル演算率はともに高い値を示している．しかし，③に示すバンクコンフリクト時間は1124.3秒と，ユーザ時間の約3分の1を占めている．

```
******    プログラム　情報    ******
経過時間（秒）                :     3651.261524
ユーザ時間（秒）              :     3644.885959
システム時間（秒）            :        3.021369
ベクトル命令実行時間（秒）    :     3625.232560
全命令実行数                  :   305051263468.
ベクトル命令実行数            :    93131375684.
ベクトル命令実行要素数        :  23742486609381.
浮動小数点データ実行要素数    :   9489525377288.
MOPS 値                       :     6572.059254
MFLOPS 値                     :     2603.517773
平均ベクトル長                :      254.935423  ①
ベクトル演算率（%）           :       99.115320  ②
メモリ使用量（MB）            :    37248.031250
MIPS 値                       :       83.692951
命令キャッシュミス（秒）      :        0.582369
オペランドキャッシュミス（秒）:        0.226319
バンクコンフリクト時間（秒）  :     1124.310413  ③
```

図4.5.5.1　Proginf

図4.5.5.2にftrace情報を示す．図中①に示すようにサブルーチンprog_doのバンクコンフリクト時間は1124.1秒である．これは全バンクコンフリクト時間のほとんどを占めていることがわかる．

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
prog_do	4495	3629.111(99.6)	807.366	6595.6	2613.7	99.13	255.1	3621.125	0.1110	0.1305	1124.1265 ①
prog_sim	1	7.927(0.2)	7926.550	227.0	0.7	4.41	1.2	1.518	0.0055	0.0540	0.0001
prog_read	1	4.999(0.1)	4999.002	248.2	1.3	0.03	231.3	0.000	0.4113	0.0236	0.0000
…											
total	5945	3644.825(100.0)	613.091	6572.2	2603.6	99.12	254.9	3625.232	0.5561	0.2160	1124.3104

図4.5.5.2　ftrace情報

図 4.5.5.3 にサブルーチン prog_do の編集リストを示す．図中に示すように二重 do ループの内側 do ループがベクトル化されている．内側 do ループ内で，配列 para_g(i,j)は，内側 do ループの変数 j が配列要素の二次元目となっている．よって，内側 do ループのアクセスは，i のサイズ NUM(=98700)要素飛びであることがわかる．これがバンクコンフリクト発生の原因となっている．

```
    2:                    PARAMETER ( NUM = 98700 )
    3:                    real para_g( NUM, NUM )
...
  252: |V----->              do j = 1, ncells
  253: ||                      para_g(i,j)=wbuf(j)
  254: |V-----               end do
...
 1030: +------>             do 6100 i = 1, ncells
 1031: |                      para_f = F0
 1032: |V----->              do 6110 j = 1, ncells
 1033: ||                       para_f = para_f + para_g( i, j ) * wary(j)
 1034: |V-----       6110    end do
...
 1054: +------       6100 end do
```

図 4.5.5.3　サブルーチン prog_do の編集リスト

（2）バンクコンフリクト回避の方法

配列が一次元目からアクセスされるように配列の使い方を変更し，配列へのアクセスが連続となるようにする．

図 4.5.5.4 にソース修正後のサブルーチン prog_do の編集リストを示す．図に示すように，配列 para_g にデータを格納する際，一次元目からデータを格納するようにした．これにより，para_g(i,j)を para_g(j,i)と置き換えて，内側 do ループで一次元目の要素にアクセスするようにした．

```
    2:                    PARAMETER ( NUM = 98700 )
    3:                    real para_g( NUM, NUM )
...
  252: |V----->              do j = 1, ncells
  253: ||                      para_g(j,i)=wbuf(j)
  254: |V-----               end do
...
...
 1030: +------>             do 6100 i = 1, ncells
 1031: |                      para_f = F0
 1032: |V----->              do 6110 j = 1, ncells
 1033: ||                       para_f = para_f + para_g( j, i ) * wary(j)
 1034: |V-----       6110    end do
...
 1054: +------       6100 end do
```

para_g に，一次元目からデータを格納する

図 4.5.5.4　ソース修正後のサブルーチン prog_do の編集リスト

（３）ソース修正による効果

図 4.5.5.5 にソース修正後の ftrace 情報を示す．図中①に示すように，サブルーチン prog_do のバンクコンフリクト時間が 0.1 秒に短縮された（修正前は 1124.1 秒）．これに伴い，実行時間は 2476.2 秒に短縮された（修正前は 3644.8 秒）．

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
prog_do	4495	2460.317(99.4)	547.345	9425.9	3735.3	99.13	255.1	2452.804	0.4070	0.4658	0.0622 ①
prog_sim	1	8.110(0.3)	8109.692	221.8	0.7	4.41	1.2	1.595	0.0077	0.0682	0.0001
prog_read	1	5.203(0.2)	5202.641	238.5	1.2	0.03	231.3	0.000	0.4970	0.0419	0.0000
…											
total	5945	2476.232(100.0)	416.523	9372.6	3712.9	99.11	254.9	2456.819	0.9397	0.5902	0.0647

図 4.5.5.5　ソース修正後の ftrace 情報

４．５．６　行列積ループのライブラリ置き換え

do ループ中に行列積の演算が含まれている場合，SX のコンパイラはより高速な演算を行うためにライブラリへの置き換えを行う．以下に，do ループ内の演算を行列積の形に修正した例を示す．この修正によりコンパイラがライブラリへの置き換えを行うことが可能になった．

（１）性能の分析

図 4.5.6.1 に Proginf を示す．図中に示す①平均ベクトル長は 248.2，②ベクトル演算率は 98.0%，③バンクコンフリクト時間は 0.0 秒と，特に問題はない．

```
      ******    プログラム 情報    ******
経過時間（秒）                :      161.639781
ユーザ時間（秒）              :      159.905034
システム時間（秒）            :        0.487791
ベクトル命令実行時間（秒）    :      121.182032
全命令実行数                  :    20336217809.
ベクトル命令実行数            :     3400523371.
ベクトル命令実行要素数        :   844017228578.
浮動小数点データ実行要素数    :   513889260236.
MOPS 値                       :     5384.151476
MFLOPS 値                     :     3213.715344
平均ベクトル長                :      248.202155 ①
ベクトル演算率（%）           :       98.032913 ②
メモリ使用量（MB）            :      192.031250
MIPS 値                       :      127.176846
命令キャッシュミス（秒）      :        0.464495
オペランドキャッシュミス（秒）:        3.940623
バンクコンフリクト時間（秒）  :        0.000556 ③
```

図 4.5.6.1　Proginf

図 4.5.6.2 に ftrace 情報を示す．サブルーチン sub2 と sub3 では，②，⑤のベクトル演算率および③，⑥の平均ベクトル長にさほど差はない．しかし，図中①，④の MFLOPS 値を見ると，sub2 は sub3 に比べてはるかに演算効率が悪いことがわかる．

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
sub2	100	73.284(51.7)	732.839	6254.7	3507.7 ①	98.43 ②	250.0 ③	73.235	0.0173	0.0431	0.0000
sub4	1	35.266(24.9)	35265.901	263.7	7.9	3.00	10.9	1.980	0.0731	0.4528	0.0000
sub3	100	33.255(23.4)	332.547	11824.3	7714.6 ④	99.81 ⑤	250.0 ⑥	33.237	0.0127	0.0167	0.0000
sub1	1	0.035(0.0)	34.935	2111.9	183.8	96.06	125.2	0.035	0.0000	0.0000	0.0000
main	1	0.000(0.0)	0.419	28.2	0.0	0.00	0.0	0.000	0.0002	0.0001	0.0000
total	203	141.840(100.0)	698.718	6069.9	3623.0	98.03	248.2	108.487	0.1033	0.5127	0.0000

図 4.5.6.2　ftrace 情報

図 4.5.6.3 にサブルーチン sub2 の編集リスト，図 4.5.6.4 にサブルーチン sub3 の編集リストを示す．どちらも多重ループの処理であり，図中の「V」記号が表すようにベクトル化されている．さらに，図 4.5.6.4 中の矢印に示すコンパイルメッセージから，sub3 のループ中の処理が行列積のライブラリに置き換えられていることがわかる．これは sub3 の tflow(i,l,k)*eigen(j,l)の部分が行列積のパターンになっているため，コンパイラがライブラリへの置き換えを行ったことを表している．これに対して，sub2 の eigen(j,l)*vw(l)*pfk(i,l,k)の部分は行列積のパターンになっておらず，コンパイラによる置き換えは行われていない．

```
 47: +------>          do k=1,nz
 48: |+----->            do j=1,ny
 49: ||+---->              do l=1,lmax
 50: |||V--->                do i=1,nx
 51: |  ||                    tflow(i,j,k)=tflow(i,j,k)
 52: |  ||           +              +eigen(j,l)*vw(l)*pfk(i,l,k)
 53: |  ||V---                 enddo
 54: |  ||+----               enddo
 55: |  |+-----              enddo
 56: |  +------             enddo
```

図 4.5.6.3　サブルーチン sub2 の編集リスト

```
80  opt  (1800): 行列積ループをライブラリ呼び出しに置換した.
      :
 73: +------>          do k=1,nz
 74: |*---->            do j=1,ny
 75: ||V--->              do l=1,lmax
 76: |||*-->                do i=1,nx
 77: ||||                    pflow(i,j,k)=pflow(i,j,k)
 78: ||||           +              +tflow(i,l,k)*eigen(j,l)
 79: |||*---                 enddo
 80: ||V----               enddo
 81: |*-----              enddo
 82: +------             enddo
```

図 4.5.6.4　サブルーチン sub3 の編集リスト

（２）行列積のライブラリへ置き換え方法

サブルーチン sub2 のループ内の処理も行列積のパターンになるようにソースを修正した．図 4.5.6.5 にソース修正後の編集リストを示す．eigen(j,l)*vw(l)の演算を予め別のループで作業配列 wk(j,l)に置き換え，実際の演算部分では，wk(j,l)*pfk(i,l,k) となるようにした．これにより行列積のパターンとなり，コンパイラはライブラリへの置き換えを行っている．

```
60  opt  (1800): 行列積ループをライブラリ呼び出しに置換した.
      :
48: +------>          do l=1,lmax
49: |V----->            do j=1,ny                          予め eigen(j,l)*vw(l)
50: ||                    wk(j,l)=eigen(j,l)*vw(l)  <=     の演算結果を wk(j,l)に
51: |V-----             enddo                              格納しておく
52: +------           enddo
53: +------>          do k=1,nz
54: |*----->            do j=1,ny
55: ||V---->              do l=1,lmax
56: |||*--->                do i=1,nx
57: ||||                    tflow(i,j,k)=tflow(i,j,k)              wk(j,l)*pfk(i,l,j)という
58: ||||          +                    +wk(j,l)*pfk(i,l,k)  <=     行列積のパターンになって
59: |||*---               enddo                                   いる
60: ||V----              enddo
61: |*-----             enddo
62: +------           enddo
```

図 4.5.6.5　ソース修正後の編集リスト

（３）ソース修正による効果

図 4.5.6.6 にソース修正後の ftrace 情報を示す．図中②に示すように sub2 の MFLOPS 値は 7711.6 に向上した（修正前は 3507.7）．また，①に示すように sub2 の実行時間は 33.3 秒に短縮された（修正前は 73.3 秒）．これにより④に示すように全体の実行時間は 101.6 秒に短縮された（修正前は 141.8 秒）．

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
sub4	1	34.984(34.4)	34984.389	265.8	7.9	3.00	10.9	1.868	0.0765	0.6183	0.0000
sub3	100	33.280(32.8)	332.796	11815.5	7708.8	99.81	250.0	33.268	0.0015	0.0071	0.0001
sub2	100	33.268(32.8)	332.680	11820.0	7711.6	99.81	250.0	33.256	0.0014	0.0072	0.0000
		①			②						
sub1	1	0.034(0.0)	34.242	2154.6	187.6	96.06	125.2	0.034	0.0000	0.0000	0.0000
main	1	0.000(0.0)	0.358	33.0	0.0	0.00	0.0	0.000	0.0002	0.0001	0.0000
total	203	101.567(100.0)	500.328	7835.4	5054.6	98.68	248.1	68.427	0.0796	0.6327	0.0001
		③									

図 4.5.6.6　ソース修正後の ftrace 情報

４．５．７　重複演算の削除

ベクトル演算率が高く，ベクトル長が長くても，高速化の余地が残っている場合がある．その一つとして，一度の計算でよいにもかかわらず，重複して計算されている場合がある．この場合，重複演算を削除することで，演算量の削減を行うことができる．

（１）性能の分析

図 4.5.7.1 に Proginf を示す．図中①に示す平均ベクトル長，および②に示すベクトル演算率はともに高い値を示している．

```
******  プログラム 情報  ******
経過時間（秒）                :      180.034357
ユーザ時間（秒）              :      112.506836
システム時間（秒）            :        0.234196
ベクトル命令実行時間（秒）    :      109.624226
全命令実行数                  :     6414937517.
ベクトル命令実行数            :     2957973850.
ベクトル命令実行要素数        :   757212925864.
浮動小数点データ実行要素数    :   411205216914.
MOPS 値                       :     6761.099289
MFLOPS 値                     :     3654.935391
平均ベクトル長                :      255.990406  ①
ベクトル演算率（%）           :       99.545537  ②
メモリ使用量（MB）            :     8256.031250
MIPS 値                       :       57.018202
命令キャッシュミス（秒）      :        0.008210
オペランドキャッシュミス（秒）:        0.950829
バンクコンフリクト時間（秒）  :        0.189765
```

図 4.5.7.1　Proginf

図 4.5.7.2 に ftrace 情報を示す．図中に示すとおり，コストが一番高いサブルーチンは，func2 である．

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
func2	103	98.165(87.3)	953.059	6856.9	4013.9	99.61	256.0	95.285	0.0066	0.9498	0.1193
func1	15	14.049(12.5)	936.569	6076.0	1146.5	99.06	256.0	14.048	0.0004	0.0005	0.0705
main	1	0.291(0.3)	290.931	7580.7	3697.9	98.44	252.7	0.291	0.0001	0.0001	0.0000
total	119	112.505(100.0)	945.416	6761.2	3655.0	99.55	256.0	109.624	0.0072	0.9504	0.1898

図 4.5.7.2　ftrace 情報

図 4.5.7.3 にサブルーチン func2 の編集リストを示す．図中の計算 y (i)*i-z(i+1)は，最内側 do ループの変数 i で求めることができる．しかし外側 do ループによって m*m 回分繰り返し計算されている．これは，y(i)*i-z(i)の計算を m*m 回重複して行っていることになる．

```
    53:                    parameter(m=1024)
    54:                    dimension a(m,m),b(m,m),y(m),z(m+1),x(m)
    55: +------->          do k=1,m
    56: |+------>            do j=1,m
    57: ||V---->              do i=1,m
    58: |||                      x(k) = x(k) + a(i,k)*b(k,j) * ( y(i) * i - z(i+1) )
    59: ||V-----              enddo
    60: |+------            enddo
         ...
    64: +-------           enddo
```

図 4.5.7.3　サブルーチン func2 の編集リスト

（２）重複演算削除の方法

図 4.5.7.4 にソース修正後のサブルーチン func2 の編集リストを示す．図中に示すように，三重 do ループの前にあらかじめ y(i)*i - z(i)を計算して作業配列 wrk に格納しておく．そして本体 do ループでは，上記作業配列 wrk を引用するようにした．

```
    53:                    parameter(m=1024)
    54:                    dimension a(m,m),b(m,m),y(m),z(m+1),x(m),wrk(m)
    55: V------->          do i=1,m                          } あらかじめ，計算を行い，
    56: |                      wrk(i) =  y(i) * i - z(i+1)   } 作業配列 wrk に格納
    57: V------            enddo
    58: +------->          do k=1,m
    59: |
    60: |+------>            do j=1,m
    61: ||V---->              do i=1,m
    62: |||                      x(k) = x(k) + a(i,k)*b(k,j) * wrk(i)
    63: ||V-----              enddo
   ...
    68: +-------           enddo
```

図 4.5.7.4　ソース修正後のサブルーチン func2 の編集リスト

（３）ソース修正による効果

図 4.5.7.5 にソース修正後の ftrace 情報を示す．図に示すように，サブルーチン func2 の実行時間が 74.0 秒に短縮された（修正前は 98.1 秒）．またプログラム全体の実行時間が 88.1 秒に短縮された（修正前は 112.5 秒）．

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
func2	103	74.013(83.9)	718.575	6190.1	4576.6	99.58	256.0	72.520	0.0080	0.0378	0.0470
func1	15	13.883(15.7)	925.534	6148.5	1160.1	99.06	256.0	13.882	0.0031	0.0033	0.0000
main	1	0.292(0.3)	291.734	7559.8	3687.8	98.44	252.7	0.291	0.0002	0.0001	0.0000
total	119	88.188(100.0)	741.075	6188.1	4035.8	99.49	256.0	86.694	0.0113	0.0412	0.0471

図 4.5.7.5　ソース修正後の ftrace 情報

４．５．８　インライン展開

一回当たりの実行時間が短く，呼び出し回数の多い副プログラムは，呼び出し時のオーバヘッドが大きくなるので高速化を妨げる要因となる．以下に，副プログラムの自動インライン展開によるオーバヘッドの削減を行った例を示す．

（１）性能の分析

図 4.5.8.1 に ftrace 情報を示す．図中①に示すように，サブルーチン check は呼び出し回数が 111,765,600 回と多く，②のように一回当たりの実行時間が 0.001msec と短い．一般に，実行時間が短く，呼び出される回数が多い副プログラムは，インライン展開することで呼び出しのためのオーバヘッドが削減され，性能の向上が得られる．図中③，④のサブルーチン volume，param についても同様である．

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
solver	234650	152.913(25.6)	0.652	603.0	169.9	85.15	86.7	50.673	0.1158	41.9971	0.0131
calca	3610	142.035(23.8)	39.345	95.1	19.9	1.01	255.4	0.028	0.4892	50.2056	0.0002
calcb	3610	124.860(20.9)	34.587	105.3	22.2	1.04	255.4	0.029	0.7522	38.0813	0.0002
check	111765600 ①	56.564(9.5)	0.001 ②	75.1	0.0	0.00	0.0	0.000	0.0254	0.6623	0.0001
calcc	3610	51.250(8.6)	14.197	234.3	28.4	62.14	149.3	4.012	0.1474	22.7819	0.0000
volume ③	18627600	40.866(6.8)	0.002	154.8	0.4	0.00	0.0	0.000	0.2581	9.5769	0.0001
param ④	28013600	12.500(2.1)	0.000	56.0	0.0	0.00	0.0	0.000	0.0139	0.0147	0.0000
flawb	1173250	4.330(0.7)	0.004	101.0	24.6	0.00	0.0	0.000	0.3445	0.9191	0.0000
:											
total	161316393	597.195(100.0)	0.004	252.7	56.0	61.75	88.7	58.595	2.6638	166.1046	0.0149

図 4.5.8.1　ftrace 情報

（２）インライン展開の指定

SX-7 の FORTRAN コンパイラは，-pi オプションを指定することで自動インライン展開を行うことができる．以下，代表的なサブオプションについて説明する．

-pi　自動インライン展開を行うことを指定(既定値ではないので指定が必要)

line=α：　自動インライン展開の対象となる副プログラムの最大行数を指定（既定値は α=50 であるので，51 行以上の副プログラムを展開する場合は指定が必要）

nest=β：　自動インライン展開の対象となる副プログラムのネストの深さを指定（既定値は β=1）

expin=ファイル名/ディレクトリ名：
展開したい副プログラムと，展開される場所を記述したソースコードが別のファイルに記述されている場合，自動インライン展開の対象とならないので，明示的にどのファイルに記述されているのかを指定する．

exp=副プログラム名：
指定された副プログラムがインライン展開の対象となることを指定する．

（3）インライン展開の効果

図 4.5.8.2 に自動インライン展開オプションを追加した場合の ftrace 情報を示す. 図のようにサブルーチン check, volume, param がインライン展開され, ①に示すように全体の実行時間が 322.5 秒に向上した(オプション指定前は 592.7 秒).

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
calcc	3610	124.411(38.2)	34.463	648.6	167.5	86.15	89.2	44.137	0.1938	31.4907	0.0234
calca	3610	93.255(28.6)	25.832	245.0	61.8	41.19	87.0	5.907	0.3027	34.8644	0.0002
calcb	3610	92.126(28.3)	25.520	242.6	61.6	41.68	86.1	5.882	0.2967	34.6228	0.0132
flawa	1173250	4.145(1.3)	0.004	105.5	25.7	0.00	0.0	0.000	0.0495	1.0749	0.0000
flawb	1173250	3.844(1.2)	0.003	112.9	27.3	0.00	0.0	0.000	0.0456	0.8623	0.0000
:											
total	2611845	325.703(100.0) ①	0.125	413.4	100.3	70.69	87.4	59.789	1.3123	103.9001	0.0379

図 4.5.8.2　オプション追加時の ftrace 情報

（4）自動インライン展開の阻害要因

以下に挙げるような場合は, コンパイラは自動インライン展開ができない. インライン展開を阻害する要因を取り除くか, 手動で展開する必要がある.

- 展開するルーチンが同じソース上で見つからない(-pi expin オプションを用いる)
- 展開ルーチンに構文エラーがある(構文エラーを修正する)
- 呼び出し手順で使用される引数が展開ルーチンの引数に一致しない(引数の個数と型を一致させる)
- 呼び出しルーチンと展開ルーチンの共通ブロックに衝突がある
- 展開するルーチンに NAMELIST が含まれる
- 展開するルーチンに SAVE 属性を持つ変数が含まれる
- 関数が展開されていて, それが DO WHILE 文または ELSE IF 文から呼び出される
- 展開ルーチンで引用される関数名が, 呼び出し元ルーチンで使用される非関数名と衝突する
- 展開するルーチンに並列化制御オプション, 強制並列化指示オプションまたは OpenMP 指示行が指定されている

４．５．９　乱数生成ルーチン

数値シミュレーションの多くで乱数を用いており，この際，計算機の演算によって生成される疑似乱数を使用することが多い．SX-7 上ではいくつかの乱数生成ルーチンを提供している．乱数を生成する場合，乱数生成に要する実行時間だけでなく，生成される乱数の性質等にも注意を払う必要がある．

（１）提供される乱数生成ルーチン

一度の呼び出しで１つの乱数を生成するものもあるが，大規模シミュレーション等で使用する乱数を生成する場合は，一度に複数の乱数を生成する一様乱数を使用することになる．以下，SX-7 で提供される一様乱数生成ルーチンである．

表 4.5.9.1　SX-7 で提供される一様乱数生成ルーチン

ルーチン名	乱数の周期	乱数の種類
random_number	2^{127}-1	
DJUFSP(ASL/SX)	2^{31}	線形合同法
DJUFLP(ASL/SX)	2^{250}-1	M 系列法
DJUFLR(ASL/SX)	可変	M 系列法

（２）性能値

乱数生成に要する時間について，100,000 個の一様乱数の生成を 5,000 回行うというベンチマークにて比較した結果が，図 4.5.9.1 である．なお，①に示す DJUFLR の周期は 2^{1396}-1 である．

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
random_number	5000	2.533(0.7)	0.507	2235.7	197.4	97.26	126.1	2.529	0.0011	0.0009	0.0000
DJUFSP	5000	0.571(0.2)	0.114	9735.5	875.6	98.94	255.8	0.564	0.0020	0.0018	0.0000
DJUFLP	5000	1.955(0.6)	0.391	1949.7	255.7	99.52	126.5	1.951	0.0001	0.0020	0.0000
DJUFLR ①	5000	0.889(0.3)	0.178	4699.4	562.6	95.86	233.1	0.622	0.0003	0.0534	0.0000

図 4.5.9.1　一様乱数の性能値

この結果では，DJUFSP ルーチンが最も高い性能であるが，周期が 2^{31} と短周期であるため，長周期の乱数を必要とするモンテカルロ法を用いた大規模なシミュレーションでは，DJUFLR ルーチンの利用をすすめる．

（３）乱数の検定

図 4.5.9.2 に 4 種類の乱数生成ルーチンについての検定結果を示す．100,000×5,000 個の乱数を生成し，0.0 から 1.0 までを 20 分割した各領域への分布を調べたものである．各ルーチンとも乱数値が分散されていることがわかる．

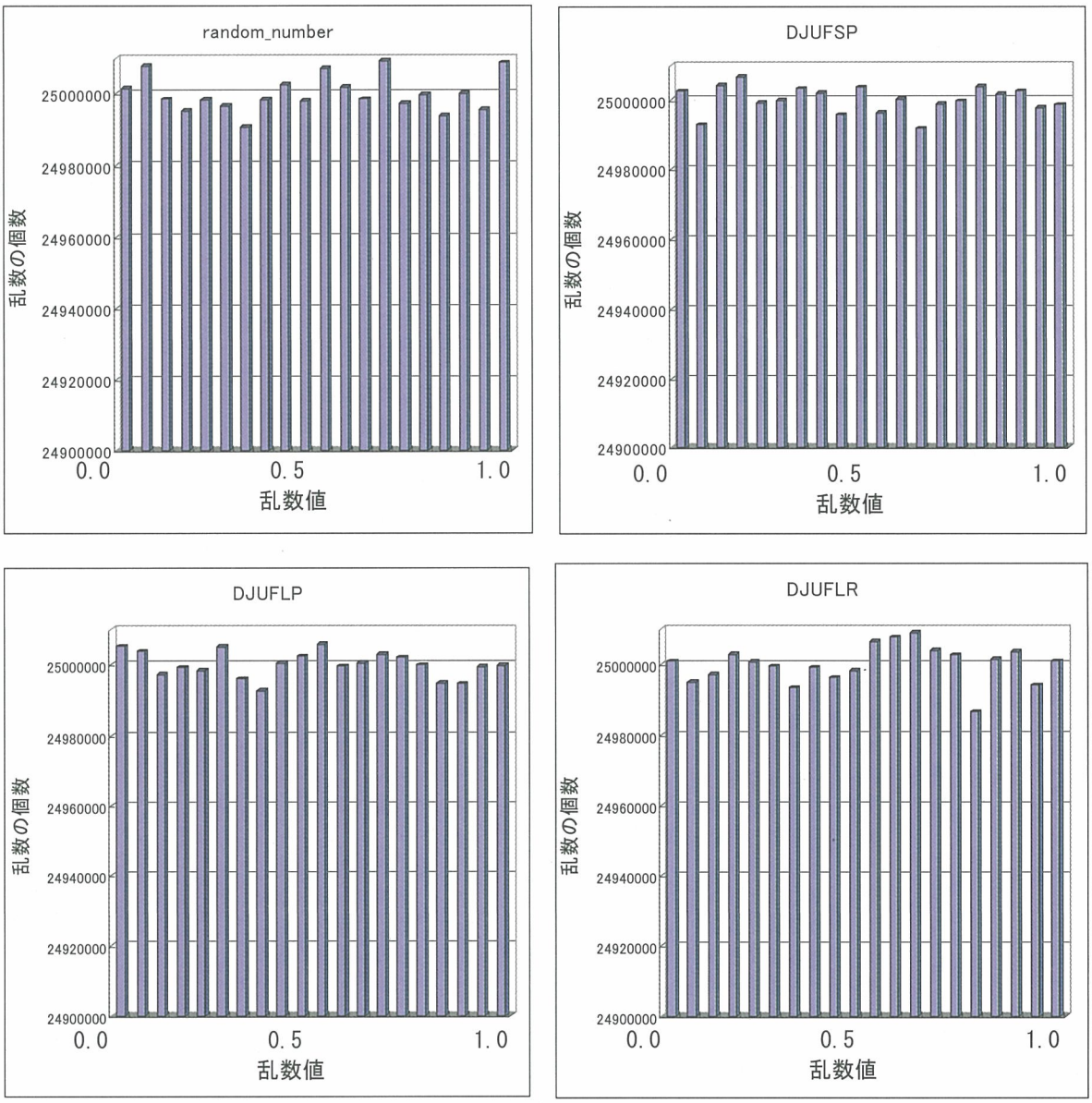


図 4.5.9.2　乱数の検定結果

ASL/SX の DJUFSP，DJUFLR については，並列処理機能版(それぞれ，QJUFSP,QJUFLR)が用意されているので，並列化されたコードで使用する時はそちらを使用されたい．

４．６　並列化チューニング事例

４．６．１　ループ入れ換えによる並列化

多重 do ループの場合，内側 do ループをベクトル化し，外側 do ループを並列化する．しかし，do ループ内に定義・参照のデータ依存関係があるため，ベクトル化は可能だが，並列化できないケースがある．このようなとき，do ループの入れ替えを行うことで，並列化できる場合がある．以下に，do ループの入れ替えを行うことで，並列化を促進する例を示す．

（１）性能の分析

図 4.6.1.1 に Proginf を示す．図中①～④に示すように，ほとんどの部分が 1 つの CPU だけで実行されており，並列化できていないことがわかる．

```
      ******    プログラム 情報    ******
経過時間（秒）                :      321.469443
ユーザ時間（秒）              :      321.638126
システム時間（秒）            :        0.051050
ベクトル命令実行時間（秒）    :      321.381345
全命令実行数                  :    14232484592.
ベクトル命令実行数            :     8364659062.
ベクトル命令実行要素数        :  2137085789479.
浮動小数点データ実行要素数    :   854810382473.
MOPS 値                       :     6662.623121
MFLOPS 値                     :     2657.677413
MOPS 値   （実行時間換算）    :     6667.874798
MFLOPS 値（実行時間換算）     :     2659.772272
平均ベクトル長                :      255.489886
ベクトル演算率（%）           :       99.726180
メモリ使用量（MB）            :      208.000000
最大同時実行可能プロセッサ数  :              4.
   1 台以上で実行した時間（秒）:      321.384801  ①
   2 台以上で実行した時間（秒）:        0.084926  ②
   3 台以上で実行した時間（秒）:        0.084758  ③
   4 台以上で実行した時間（秒）:        0.084139  ④
イベントビジー回数            :              0.
イベント待ち時間（秒）        :        0.000000
ロックビジー回数              :              0.
ロック待ち時間（秒）          :        0.000000
バリアビジー回数              :              0.
バリア待ち時間（秒）          :        0.000000
MIPS 値                       :       44.249992
MIPS 値（実行時間換算）       :       44.284871
命令キャッシュミス（秒）      :        0.002689
オペランドキャッシュミス（秒）:        0.009786
バンクコンフリクト時間（秒）  :        0.000026
```

図 4.6.1.1　Proginf

図 4.6.1.2 に ftrace 情報を示す．図中①に示すように，最もコストの高いサブルーチン subb は，並列化されていないことがわかる（並列化されている場合は，サブルーチン名の後ろに「$数字」が付く）．

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
subb ①	51200	321.369(100.0)	6.277	6668.0	2659.9	99.73	255.5	321.346	0.0010	0.0024	0.0000
suba	1	0.035(0.0)	34.798	2088.1	179.9	96.82	116.0	0.035	0.0000	0.0000	0.0000
main	1	0.032(0.0)	32.192	417.4	33.4	0.04	239.0	0.000	0.0004	0.0002	0.0000
suba$1	4	0.001(0.0)	0.358	3685.3	0.1	98.96	255.8	0.001	0.0001	0.0001	0.0000
-micro1	1	0.001(0.0)	1.168	4511.2	0.0	99.11	255.8	0.001	0.0000	0.0000	0.0000
-micro2	1	0.000(0.0)	0.008	47.6	2.9	0.00	0.0	0.000	0.0000	0.0000	0.0000
-micro3	1	0.000(0.0)	0.107	30.6	0.2	0.00	0.0	0.000	0.0000	0.0000	0.0000
-micro4	1	0.000(0.0)	0.149	30.5	0.2	0.00	0.0	0.000	0.0000	0.0000	0.0000
subc	1	0.001(0.0)	0.787	106.2	0.1	40.95	232.8	0.000	0.0003	0.0002	0.0000
total	51207	321.438(100.0)	6.277	6666.8	2659.3	99.73	255.5	321.381	0.0017	0.0029	0.0000

図 4.6.1.2　ftrace 情報

図 4.6.1.3 に編集リストを示す．図中の do ループは，外側の do ループの添え字 j に対して，定義・参照のデータ依存関係が生じており並列化されていない．

```
  3:                  parameter(nx=1022,ny=1021)
    ...
  8: +------->        do j=1,ny
  9: |V----->          do i=1,nx
 10: ||                 u(i,j)   = u(i,j)   - u1(i,j)*u2(i,j)/dd
 11: ||                 v(i,j)   = v(i,j)   - v1(i,j)*v2(i,j)/dd
 12: ||                 w(i,j)   = w(i,j)   - w1(i,j)*w2(i,j)/dd
 13: ||                 e(i,j)   = e(i,j)   - e1(i,j)*e2(i,j)/dd
 14: ||                 u(i,j+1) = u(i,j+1) + u1(i,j)*u2(i,j)/dd
 15: ||                 v(i,j+1) = v(i,j+1) + v1(i,j)*v2(i,j)/dd
 16: ||                 w(i,j+1) = w(i,j+1) + w1(i,j)*w2(i,j)/dd
 17: ||                 e(i,j+1) = e(i,j+1) + e1(i,j)*e2(i,j)/dd
 18: |V-----          enddo
 19: +-------        enddo
```

図 4.6.1.3　編集リスト

（２）並列化阻害要因の除去方法

外側 do ループの添え字 j については，並列化を阻害する依存関係は存在するが，ベクトル化を阻害するものではない．内側 do ループの添え字 i については，ベクトル化も並列化も阻害する依存関係はない．また，nx=1022，ny=1021 であり，２つの do ループは十分にベクトル長がある．

図 4.6.1.4 にソース修正後の編集リストを示す．内側と外側の do ループを入れ替えることにより，ベクトル化と並列化を行うことにした．またコンパイラは自動的に，配列の一次元目の添え字 i の do ループをベクトル化の対象とするように do ループの入れ替えを行うので，それを抑止するコンパイラ指示行を挿入した(図中の矢印).

```
 8: P------>          do i=1,nx
 9: |           !cdir noloopchg        <== ループ入れ換えの抑止
10: |V----->            do j=1,ny
11: ||                     u(i,j)   = u(i,j)   - u1(i,j)*u2(i,j)/dd
12: ||                     v(i,j)   = v(i,j)   - v1(i,j)*v2(i,j)/dd
13: ||                     w(i,j)   = w(i,j)   - w1(i,j)*w2(i,j)/dd
14: ||                     e(i,j)   = e(i,j)   - e1(i,j)*e2(i,j)/dd
15: ||                     u(i,j+1) = u(i,j+1) + u1(i,j)*u2(i,j)/dd
16: ||                     v(i,j+1) = v(i,j+1) + v1(i,j)*v2(i,j)/dd
17: ||                     w(i,j+1) = w(i,j+1) + w1(i,j)*w2(i,j)/dd
18: ||                     e(i,j+1) = e(i,j+1) + e1(i,j)*e2(i,j)/dd
19: |V-----              enddo
20: P------            enddo
```

図 4.6.1.4　ソース修正後の編集リスト

（3）ソース修正による効果

図 4.6.1.5 にソース修正後の Proginf を示す．図①～④に示すように，各 CPU が実行した時間に差がなく，並列化できていることがわかる．図中①に示すように実行時間も 121.0 秒に向上した（修正前は 321.4 秒）．

```
      ******    プログラム 情報    ******
経過時間（秒）                :      121.043601
ユーザ時間（秒）              :      483.995969
システム時間（秒）            :        0.051845
ベクトル命令実行時間（秒）    :      481.036157
全命令実行数                  :    14099358619.
ベクトル命令実行数            :     8372851062.
ベクトル命令実行要素数        :  2137085789479.
浮動小数点データ実行要素数    :   854810945670.
MOPS 値                       :     4427.335007
MFLOPS 値                     :     1766.153027
MOPS 値    (実行時間換算)     :    17703.545838
MFLOPS 値 (実行時間換算)      :     7062.300688
平均ベクトル長                :      255.239915
ベクトル演算率 (%)            :       99.732757
メモリ使用量 (MB)             :      208.000000
最大同時実行可能プロセッサ数  :              4.
   1台以上で実行した時間（秒）:      121.038594  ①
   2台以上で実行した時間（秒）:      121.007163  ②
   3台以上で実行した時間（秒）:      121.006951  ③
   4台以上で実行した時間（秒）:      120.943812  ④
   :
```

図 4.6.1.5　ソース修正後の Proginf

４．６．２　負荷バランスの調整

自動並列化が行われても，各 CPU に割り当てられた負荷バランスが不均等であると効率のよい並列化を行うことができない．以下に，コンパイラ指示行の挿入により，負荷バランスの不均等を解消した例を示す．

（１）性能の分析

図 4.6.2.1 に Proginf を示す．図中①～④に示すように各 CPU が実行した時間にばらつきがあり，各 CPU に割り当てられた負荷バランスが不均等であることがわかる．

```
******  プログラム 情報  ******
経過時間（秒）                :      86.464324
ユーザ時間（秒）              :     228.643402
システム時間（秒）            :       0.350031
ベクトル命令実行時間（秒）    :     183.478110
全命令実行数                  :   12978973353.
ベクトル命令実行数            :    6482442272.
ベクトル命令実行要素数        :1641243648398.
浮動小数点データ実行要素数    :  929323558981.
MOPS 値                       :    7206.594046
MFLOPS 値                     :    4064.510722
MOPS 値   （実行時間換算）    :   19077.496715
MFLOPS 値（実行時間換算）     :   10759.686123
平均ベクトル長                :     253.182918
ベクトル演算率（%）           :      99.605731
メモリ使用量（MB）            :   18544.000000
最大同時実行可能プロセッサ数  :          4.
   1台以上で実行した時間（秒）:      86.370880  ①
   2台以上で実行した時間（秒）:      61.128696  ②
   3台以上で実行した時間（秒）:      41.689851  ③
   4台以上で実行した時間（秒）:      39.454017  ④
イベントビジー回数            :          0.
イベント待ち時間（秒）        :       0.000000
ロックビジー回数              :          0.
ロック待ち時間（秒）          :       0.000000
バリアビジー回数              :          0.
バリア待ち時間（秒）          :       0.000000
MIPS 値                       :      56.765134
MIPS 値（実行時間換算）       :     150.270246
命令キャッシュミス（秒）      :       0.032893
オペランドキャッシュミス（秒）:       1.532640
バンクコンフリクト時間（秒）  :       0.011514
```

図 4.6.2.1　Proginf

図 4.6.2.2 に ftrace 情報を示す．図中①～④に示すように，サブルーチン solve について，各 CPU の実行時間にばらつきがあることがわかる．

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
solve$1	2048	220.987(96.3)	107.904	7373.7	4198.1	99.63	256.0	175.160	0.0163	1.4971	0.0092
-micro1 ①	512	41.531(18.1)	81.115	8713.1	4964.0	99.70	256.0	38.971	0.0041	0.0057	0.0030
-micro2 ②	512	55.307(24.1)	08.022	6085.4	3461.8	99.55	256.0	36.138	0.0039	0.6680	0.0021
-micro3 ③	512	59.403(25.9)	116.022	6788.7	3863.6	99.60	256.0	43.320	0.0041	0.5630	0.0020
-micro4 ④	512	64.745(28.2)	126.456	8151.6	4642.5	99.67	256.0	56.732	0.0041	0.2605	0.0021
input	1	8.516(3.7)	8515.812	2142.1	189.1	97.27	126.2	8.515	0.0009	0.0011	0.0000
solve	512	0.060(0.0)	0.116	90.1	0.4	54.77	239.0	0.001	0.0107	0.0097	0.0017
main	1	0.001(0.0)	0.840	157.6	12.9	0.00	0.0	0.000	0.0003	0.0001	0.0000
output	1	0.000(0.0)	0.000	0.0	0.0	0.00	0.0	0.000	0.0000	0.0000	0.0000
total	2563	229.563(100.0)	89.568	7177.7	4048.2	99.61	253.2	183.677	0.0282	1.5080	0.0108

図 4.6.2.2　ftrace 情報

図 4.6.2.3 にサブルーチン solve の編集リストを示す．並列化対象の do ループ中に if 文があるため，if 文の真偽によって，各 CPU の演算の負荷にばらつきが出る．

```
 70: P------>          do k=1,nz
 71: |                   if(id(k).eq.0) then
 72: |W----->              do j=1,ny
 73: ||*---->                do i=1,nx
 74: |||                       v3(i,j,k) = R1*v1(i,j,k) + R2*v2(i,j,k) + R3 * R4
 75: |||                       u3(i,j,k) = R1*u1(i,j,k) + R2*u2(i,j,k) - R3 * R4
 76: |||                       w3(i,j,k) = R1*w1(i,j,k) + R2*w2(i,j,k) + R3 * R4
 77: ||*-----                enddo
 78: |W-----                enddo
 79: |                     endif
 80: P------           enddo
```

図 4.6.2.3　編集リスト

（２）負荷バランスの調整方法

図 4.6.2.4 にサブルーチン solve を修正した編集リストを示す．並列化対象となる do ループの分割方法を，コンパイラ指示行 concur(by=1)を使用して変更した．

```
69:              !cdir concur(by=1)
70: P------>          do k=1,nz
71: |                   if(id(k).eq.0) then
72: |W----->.            do j=1,ny
73: ||*---->               do i=1,nx
74: |||                      v3(i,j,k) = R1*v1(i,j,k) + R2*v2(i,j,k) + R3 * R4
75: |||                      u3(i,j,k) = R1*u1(i,j,k) + R2*u2(i,j,k) - R3 * R4
76: |||                      w3(i,j,k) = R1*w1(i,j,k) + R2*w2(i,j,k) + R3 * R4
77: ||*----                 enddo
78: |W-----                enddo
79: |                     endif
80: P------              enddo
```

図 4.6.2.4　指示行挿入後の編集リスト

（３）分割方法変更による効果

図 4.6.2.5 に指示行挿入後の ftrace 情報を示す．図中の①～④に示すようにサブルーチン solve の各 CPU の実行時間がほぼ均等になった．図 4.6.2.6 に指示行挿入後の Proginf を示す．図中の①に示すように，実行時間も 57.5 秒に短縮された(修正前は 86.4 秒)．

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
solve$1	2048	195.437(95.8)	95.428	8334.6	4746.9	99.67	256.0	176.180	0.0107	0.7037	0.0287
-micro1 ①	512	48.878(24.0)	95.466	8778.2	5000.4	99.69	256.0	46.397	0.0026	0.0916	0.0075
-micro2 ②	512	48.860(24.0)	95.430	8121.8	4625.2	99.66	256.0	42.926	0.0024	0.2160	0.0071
-micro3 ③	512	48.847(23.9)	95.403	8210.6	4676.0	99.66	256.0	43.381	0.0028	0.1997	0.0071
-micro4 ④	512	48.852(24.0)	95.413	8227.8	4685.8	99.66	256.0	43.476	0.0028	0.1964	0.0071
input	1	8.493(4.2)	8492.617	2148.0	189.6	97.27	126.2	8.493	0.0000	0.0000	0.0000
solve	512	0.030(0.0)	0.059	149.0	0.7	64.84	239.0	0.001	0.0100	0.0092	0.0017
main	1	0.001(0.0)	0.830	159.5	13.0	0.00	0.0	0.000	0.0004	0.0001	0.0000
output	1	0.000(0.0)	0.001	226.0	21.0	0.00	0.0	0.000	0.0000	0.0000	0.0000
total	2563	203.961(100.0)	79.579	8075.8	4556.4	99.64	253.2	184.673	0.0211	0.7131	0.0305

図 4.6.2.5　指示行挿入後の ftrace 情報

```
      ******    プログラム  情報    ******
経過時間（秒）                :      57.549276
ユーザ時間（秒）              :     204.576108
システム時間（秒）            :       0.080975
   :
最大同時実行可能プロセッサ数  :              4.
   1台以上で実行した時間（秒）:      57.544710  ①
   2台以上で実行した時間（秒）:      49.042544
   3台以上で実行した時間（秒）:      49.042457
   4台以上で実行した時間（秒）:      48.946444
```

図 4.6.2.6　指示行挿入後の Proginf

４．６．３　並列化指示行による並列化（並列コンピュータ AzusA による高速化）

ベクトル演算率が低い場合は，SX-7 よりも AzusA の方が高速化が期待できる．ここでは AzusA による高速化の例を示す．

（１）性能の分析

図 4.6.3.1 に SX-7 で実行したときの ftrace 情報を示す．図中①に示すようにコストの 56.3%を占めているサブルーチン sub_a の②ベクトル演算率は 62.18%と低く，③平均ベクトル長も 40.1 と小さい．

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_a	2	1619.526(56.3) ①	809763.079	568.8	95.4	62.18 ②	40.1 ③	300.121	86.9083	256.0777	0.0033
sub_b	2	520.635(18.1) ④ ⑤	260317.737	7528.0	4277.8	99.52 ⑥	246.1 ⑦	520.440	0.0028	0.1089	90.7350 ⑧
sub_c	781743062	316.736(11.0)	0.000	66.6	12.3	0.00	0.0	0.000	0.0382	30.8979	0.0010
sub_d	335089316	283.083(9.8)	0.001	66.3	0.0	0.00	0.0	0.000	0.0267	11.8700	0.0007
sub_e	111751232	128.940(4.5)	0.001	72.8	10.4	0.00	0.0	0.000	29.0002	0.5667	0.0003
sub_f	2	5.561(0.2)	2780.668	4300.5	1648.3	99.68	249.3	5.497	0.0000	0.0000	2.9719
main	1	0.796(0.0)	795.889	184.3	3.6	0.27	242.9	0.000	0.1464	0.0186	0.0000
…											
total	1228606041	2875.304(100.0)	0.002	1709.0	833.4	91.52	148.7	826.059	116.1227	299.5404	93.7122

図 4.6.3.1　SX-7 の ftrace 情報

図 4.6.3.2 に sub_a の編集リストを示す．sub_a は 3 重 do ループで構成され，ベクトル化の対象となる do ループは最内側の 2 つである．2 つの do ループはループ長(n3,n4)が共に小さく，演算量も少ない．このため並列化が可能であればスカラ性能の高い AzusA での方が適していると思われる．

```
    1:                  subroutine sub_a
                          …
  569: +------>         do k=1,n1                ⇐ ベクトル化されないループ
       |                  …
  601: |+----->           do j=1,n2                ⇐ ベクトル化されないループ
  602: ||                    if( ) then
  603: ||                        write(*,*) "error xxxxxx"
  604: ||                        stop
  605: ||                    endif
       ||                    …
  630: ||V---->              do i=1,n3              ⇐ ベクトル化されるループ
    :  |||                    …
  635: ||V-----              enddo
  636: ||V---->              do i=1,n4              ⇐ ベクトル化されるループ
    :  |||                    …
  642: ||V-----              enddo
       ||                    …
 1529: |+------           enddo
       |                   …
 1547: +-------          enddo
    :                       …
```

図 4.6.3.2　サブルーチン sub_a の編集リスト

一方，図 4.6.3.1 中⑤に示すようにコストの 18.1%を占めるサブルーチン sub_b の⑥ベクトル演算率は 99.52%と高く，⑦平均ベクトル長も 246.1 と大きい．しかし，④実行時間 520.63 秒に対して⑧バンクコンフリクトが 90.73 秒も発生している．

図 4.6.3.3 に sub_b のソースを示す．sub_b では連立一次方程式（ガウスの消去法）を解いており，実行時間の大きいループで配列の二次元目を動かしている．これがバンクコンフリクトの原因であった．バンクコンフリクトを削減するには配列の一次元目を動かすようにプログラムを書き換える方法もあるが，ここではメーカ提供の数値計算ライブラリ ASL に置き換えることにした．また，sub_c，sub_d，sub_e は一回あたりの実行時間が短いが，呼び出し回数が非常に多いので，呼び出しのためのオーバヘッドを削減するためインライン展開を行うことにした．

```
      do  i=1,n-1
            …
            do  ii=i+1,n
                  r=a(ii,i)/a(i,i)
                  do  j=i+1,n
                        a(ii,j)=a(ii,j)-r*a(i,j)
                  enddo
                  …
            enddo
               …
      enddo
```

図 4.6.3.3　サブルーチン sub_b のソース

（２）　並列化指示行の利用

図 4.6.3.2 で示したサブルーチン sub_a は，write 文(603 行目)があるため自動並列化の対象とならない．しかし，この write 文はパラメータが不適当な場合に実行され，プログラムが停止するように記述されているので，通常実行されることはない．そこで並列化指示行を用いて最外側の k のループ（569 行目）を並列化することとした．

図 4.6.3.4 に並列化指示行 paralleldo の追加とソース修正後のサブルーチン sub_a のリストを示す．ここで k のループの本体部分（図 4.6.3.2　570～1546 行）をサブルーチン化しているのはつぎの理由による．paralleldo を使用するときにはループ内でのみ使用する変数を並列化指示行で指定する必要がある．しかし，本体部分は約 1,000 行あり，多数の変数を使用しているので，それらを指定するのは大変である．そこでループの本体部分を別サブルーチン body とした（これにより，サブルーチン body で使用する変数はそれぞれの CPU ごとに確保される）．

HPC チャレンジでの SX システムの性能評価

小林 広明[1)] , 滝沢 寛之[1)2)], 小久保 達信[3)], 岡部 公起[1)],
伊藤 英一[1)] , 小林 義昭[3)], 浅見 暁[4)], 小林 一夫[4)]
後藤 記一[4)], 片海 健亮[5)], 深田 大輔[5)]

1) 東北大学情報シナジーセンター, 2) 東北大学大学院情報科学研究科,
3) 日本電気株式会社, 4)NEC情報システムズ, 5) NECソフト

1. はじめに

HPC (High Performance Computing) チャレンジは、総合的な HPC システム性能評価の試みとして、米国 DARPA (Defense Advanced Research Projects Agency) の HPCS (High-Productivity Computing Systems) プロジェクトの支援を受けて[1]、Tennessee 大学の J. Dongarra 博士 により、2003 年 11 月 SC2003(2003 年 Supercomputing Conference)において提唱された HPC システムのベンチマーク(BM)セットです[2]。Linpack HPC の単一性能指標のみで評価する Top500 を補完するものとして、より総合的で Linpack(HPL)も含んだ 7 セットの BM コードの集まりとなっています。この BM セットでは、これまでスーパーコンピュータの性能評価で重要視されてきた総演算性能の評価に加えて、アプリケーション実行におけるスーパーコンピュータの実効性能を引き出す上で重要なメモリアクセス性能とネットワーク性能の評価と、多くのアプリケーションで頻繁に使用されるカーネルコードを用いた性能評価が可能になっています。これにより、従来のノード数に頼るスーパーコンピュータの数量的な評価ばかりでなく、ノードの「質」を評価することが可能になります。本報告では、NEC と東北大学情報シナジーセンターが共同で行った HPC チャレンジベンチマークを使った SX-7 の評価結果について述べ、28 評価項目の中、16 項目で最高性能を出したベクトル型スーパーコンピュータの HPC 分野における優位性を明らかにします。

2. HPC チャレンジとは

テストの概要は、HPC チャレンジの Web ページを参照するのが一番です。

http://icl.cs.utk.edu/hpcc/index.html

この Web ページの情報をもとに説明します。HPC チャレンジは 7 つのカテゴリーの BM セットとなっています。まずは、評価方法について説明し、次にそれぞれのテストついて説明します。

2. 1 評価方法

評価方法は、全ノード総合性能を評価するテスト(G: Global system performance)と、ノード単体のシングル環境でのテスト(SN: Single Environment)、多重負荷環境でのテスト(EP: Embarrassingly Parallel)の 3 つがあります。この 3 つのテストは、計算機システムが複数の CPU

から構成される並列計算機となっていることで、その特徴を評価することを目的としています。最近の並列計算機の構成方法は、SMP によるメモリ共有型計算機、MPI による分散メモリ型計算機、そしてその組み合わせによるハイブリッド型並列計算機があります。特に後者の組み合わせの並列計算機では、メモリが共有される単位をノードとして定義することが多く、そのノードをネットワークで繋ぎ一つのシステムとなっています。このように計算機システムが複雑な構成となった場合、全体と部分の評価が必要で、上記 3 つのテストで対応します。

全ノード総合性能テストでは、計算機システム全体を使ってのテスト(G)となり、一つのシステム全体でどの程度性能があるかを評価するものとなります。更に全ノード総合性能だけではなく、各ノードの性能についても評価するものが次の 2 つです。すなわち、ノード内ではメモリは共有されているため、複数プロセスを同時に使ってテストをするとメモリ利用の奪い合いが起こります。これを避ける一番簡単な評価方法が、プロセスを 1 つとしてテストすることで、これがシングル環境でのテスト(SN)になります。この場合、単体ノードの性能が最大限に発揮されます。逆に、複数のプロセスを同時に使ったテストの場合、メモリ利用の奪い合いが起こり、シングル環境のテストよりも性能が劣化することになります。その性能劣化具合を調べるのが多重負荷時のテスト(EP)になります。これらのテストで、計算機システムの各ノードの性能評価を行えます。

2．2 HPCC ベンチマークプログラム

2．2．1 HPL：連立1次方程式（LU 分解と後退代入）の計算プログラム

いわゆる Linpack の MPI で記述された分散並列版で、ノード全体の演算性能を評価します。オリジナルコードは、Netlib の HPL です。全ノードのトータルな演算性能が大きいほど高い性能となり、全ノード総合性能に依存します。性能の単位は Tflop/s（Tera floating-point operations per second:一秒間に浮動小数点演算を何回行うかを、10 の 12 乗の単位で表す）で表示されます。テスト項目は 1 項目で、全ノードを使った総合性能(G)のテストを行います。

2．2．2 DGEMM：実数行列 A,B の行列積 C=AB を計算するカーネルプログラム

DGEMM は Netlib の BLAS(Basic Linear Algebra Subprograms)の一つの機能として提供され、さまざまな数値計算に出てきます。例えば連立 1 次方程式 LU 分解計算（Linpack）では、演算の主要部分が DGEMM となり、性能を左右する一番重要な要素となります。本 BM による評価結果は、全ノード総合性能には依存せず、ノード単体の演算性能に依存します。性能の単位は Gflop/s（Giga floating point operations per second:一秒間に浮動小数点演算を何回行うかを、10 の 9 乗の単位で表す）で表示されます。テストの項目は 2 項目で、ノード単体のシングル環境(SN)、多重負荷環境(EP)のテストを行います。

2．2．3 STREAM：メモリバンド幅の評価プログラム

複写(Copy)、定数倍(Scale)、総和(Add)、積和(Triad)の 4 つの計算プログラムからなっており、ノード単体のメモリ性能を測定します。オリジナルコードは、J. D. McCalpin 博士の STREAM memory bandwidth benchmark です。全ノード総合性能には依存せず、ノード単体のメモリ性能に依存します。性能の単位は GB/s（Giga Bytes per second:一秒間に転送するメモリサイズ（バイト）を 10 の 9 乗の単位で表す）で表示されます。テストの項目は 8 項目で、シングル環境(SN)および多重負荷環境(EP)での、各ノード単体の 4 項目（複写、定数倍、総和、積和）のテストを行います。

2．2．4 PTRANS：行列の転置 $A=A+B^T$ を行うプログラム

PTRANS (Parallel matrix TRANSpose)は、全ネットワーク転送性能を行列の転置で評価します。オリジナルコードは Netlib の PARKBENCH (PARallel Kernels and BENCHmarks)です。全ネットワーク転送性能が大きいほど高い性能となり、全ネットワーク総合転送性能に依存します。性能の単位は GB/s で表示されます。テスト項目は 1 項目で、全ノードを使った総合性能(G)のテストを行います。

2．2．5 RandomAccess：整数データの間接参照（インダイレクトアクセス）性能を評価するプログラム

オリジナルコードは、DARPA HPCS Discrete Math Benchmarks です。ノード単体のメモリ間接参照のテストと、全ノードの MPI 通信での参照テストとなっています。ノード単体の性能に依存する項目と、全ノード総合性能に依存する項目の両方があります。性能の単位は Gup/s (Giga updates per second:一秒間に更新する要素数を 10 の 9 乗の単位で表す）で表示されます。テストの項目は 3 項目で、ノード単体のシングル環境(SN)、多重負荷環境(EP)のテスト、および全ノードを使った総合性能(G)のテストを行います。

2．2．6 FFTE ：離散フーリエ変換の性能評価を行うカーネルプログラム

一次元離散フーリエ変換を高速フーリエ変換で計算するカーネルプログラムです。オリジナルコードは、筑波大学の高橋大介博士が開発した FFTE です。ノード単体の FFT のテストと、全ノードを使った FFT のテストとなっています。ノード単体の性能に依存する項目と、全ノード総合性能に依存する項目の両方があります。性能の単位は Gflop/s で表示されます。テストの項目は 3 項目で、ノード単体のシングル環境(SN)、多重負荷環境(EP)のテスト、および全ノードを使った全環境(G)テストを行います。

2．2．7 Communication bandwidth and latency：データ転送能力を評価するプログラム

オリジナルコードは、HLRS(ドイツ High Performance Computing Center in Stuttgart)が開発の b_eff (effective bandwidth benchmark)です。

一般に、データ転送能力の評価は、プロセス間のデータ転送速度（バンド幅 BW: bandwidth)とプロセス間のデータ転送の立ち上がり時間（セットアップ時間, SetupTime）で行います。データ転送時間 T(レイテンシ, Latency)は次式で得られます。

Latency=SetupTime + (Data Size)/BW

転送するデータ量(Data Size)が小さい場合は、レイテンシにおいてセットアップ時間が支配的になるために、レイテンシが小さい、すなわちセットアップ時間が短いほど優れた性能を示します。一方、転送データ量が大きい場合は、レイテンシにおけるセットアップ時間の割合は相対的に小さくなるために、バンド幅が高いほどデータ転送能力が高くなります。したがって、セットアップ時間は小さく、バンド幅は大きいほど、データサイズに関係なくデータ転送能力が優れていることになります。実際の測定では、Setup 時間は最小単位のデータ転送により、バンド幅は、MB オーダのデータ転送により、それぞれ近似的に求めます。

データ転送能力を評価するため、ネットワークの転送スキームは、Ping-Pong, Ring(Naturally

ordered, Randomly ordered)が用意されています。ネットワーク全体のポイント間の性能を評価するため、ノード間通信が多いほど性能が落ちることもあります。性能の単位はレイテンシがマイクロ秒、バンド幅が GB/s で表示されます。テストの項目は、バンド幅とレイテンシそれぞれ 5 項目ありますが、2004 年 12 月現在、ブラウザから詳細表示される項目をあげるとレイテンシが 2 項目、バンド幅が 3 項目となっています。測定項目は全部で 10 項目ですので、今後表示される項目が変更される可能性があります。

3. 実行ルール

HPC チャレンジには、ベースラインランとオプティマイズランの 2 つ実行ルールがあります。2004 年 12 月現在、全登録された結果は 49 件あるうち、ベースラインランは 45 件、オプティマイズランが 4 件となっており、多くの公表された値は、ベースラインランとなっています。

3. 1 コードの変更が許されないベースラインラン

基本実行ルールであるベースラインランは、コンパイラによる高度な最適化と、高性能の BLAS ライブラリ、MPI ライブラリを使うことで HPC チャレンジベンチマークを評価します。HPC チャレンジの評価結果には、使用したコンパイラおよびライブラリのバージョン、そしてコンパイル時のオプションが公開されます。

3. 2 コードの最適化が許されるオプティマイズラン

コードの修正を伴う最適化は、2 つのレベルが許可されています。一つは、限られた部分(特定のサブルーチン単位)のコード最適化であり、もう一つは、アルゴリズムの見直しを含んだ根本的な最適化です。しかし、後者は HPCC プロジェクト主催グループとの協議が必要で、また、その結果は公開されることになっています。2004 年 12 月現在のオプティマイズランの中で、前者の限られた部分の最適化の方しか登録されていません。

4. ベンチマークコードの入手方法

ベンチマークコードは HPCC のサイトで公開され、誰でもダウンロード可能です。

http://icl.cs.utk.edu/hpcc/software/index.html

最初のバージョン 0.3 α は 2003 年 11 月 5 日に公表されています。この時点では、大テスト項目は 5 項目でした。0.6 α が 2004 年 5 月 31 日に更新され、この時点でテスト 2 項目(DGEMM と FFTE)が追加され、2004 年 12 月時点と同じ大テスト項目が 7 項目となりました。最新のバージョンは 0.8 β で 2004 年 10 月 19 日に更新されています。0.6 α から見て大きな変更は無く、出力の変更などマイナーチェンジとなっています。

5. 各テスト項目の詳細分析とその評価結果

HPC チャレンジは、C 言語で書かれており、make(コンパイルとリンク)すると一つの実行形式(ロードモジュール)ができます。これを実行すると全項目のテストがまとめて行われ、結果が表

示されます。入力データのサイズやパラメータを制御するのが、「hpccinf.txt」という入力ファイルです。この入力ファイルの内容は次のようになっています(分かりやすいように、行番号をつけてあります)。

```
 1    HPLinpack benchmark input file
 2    Innovative Computing Laboratory, University of Tennessee
 3    HPL.out      output file name (if any)
 4    8            device out (6=stdout,7=stderr,file)
 5    1            # of problems sizes (N)
 6    30000        Ns
 7    1            # of NBs
 8    64           NBs
 9    1            PMAP process mapping (0=Row-,1=Column-major)
10    1            # of process grids (P x Q)
11    1            Ps
12    32           Qs
13    16.0         threshold
14    1            # of panel fact
15    2            PFACTs (0=left, 1=Crout, 2=Right)
16    1            # of recursive stopping criterium
17    44           NBMINs (>= 1)
18    1            # of panels in recursion
19    3            NDIVs
20    1            # of recursive panel fact.
21    2            RFACTs (0=left, 1=Crout, 2=Right)
22    1            # of broadcast
23    0            BCASTs (0=1rg,1=1rM,2=2rg,3=2rM,4=Lng,5=LnM)
24    1            # of lookahead depth
25    1            DEPTHs (>=0)
26    2            SWAP (0=bin-exch,1=long,2=mix)
27    64           swapping threshold
28    0            L1 in (0=transposed,1=no-transposed) form
29    0            U  in (0=transposed,1=no-transposed) form
30    1            Equilibration (0=no,1=yes)
31    16           memory alignment in double (> 0)
32    ##### This line (no. 32) is ignored (it serves as a separator). ######
33    0                       Number of additional problem sizes for PTRANS
34    1200 10000 30000                values of N
35    2                       number of additional blocking sizes for PTRANS
36    134   471                       values of NB
```

1～31 行目は HPL に関連したパラメータとなっています。33～36 行目が PTRANS に関連したパラメータです。以降、各テスト項目について説明しますが、HPL に関してはパラメータの決め方をより詳しく説明します。

今回、我々はスーパーコンピュータの「質」を評価するという観点から、SX-7 の 1 ノード(32 CPU)を用いて、その性能を 2 つ並列処理形態で評価しました。一つはノード内 32CPU の各 CPU をそれぞれ独立のノードとして見立てて、1CPU に 1MPI プロセスを割り当て、性能評価を行います。この場合、MPI による 32CPU の分散並列処理となります。もう一つは、SX-7 の共有並列を最大限に活かし、32CPU を 2 つの 16CPU グループに分け、それぞれに MPI プロセスを割り当てると共に、1 つの MPI プロセス内では 16CPU による SMP 共有並列処理を行うハイブリ

ッド型の並列処理形態です。HPC チャレンジは、プロセス間の通信などの評価を行うため、最低限 MPI の 2 プロセスのテストが必要となります。

また、評価の途中では、SMP 共有並列と MPI 分散並列の組み合わせの割合を変化させて、さまざまなパターンでの評価も同時に行いました。

5．1 HPL

- SMP 並列処理の導入効果

HPL のオリジナルコードは MPI で分散並列化されたもとなっていますが、SMP 並列化された BLAS とコンパイラによる自動 SMP 並列を使用することで、HPL を SMP 共有並列と MPI 分散並列を組み合わせたハイブリッドの並列化が可能となります。コードの変更が許されないベースラインランでは、一部のコードで SMP 共有並列のオーバヘッドに比べて処理時間が短い部分があり、コンパイラの判断により自動での並列化が行われないため、MPI 分散並列だけの結果と比較するとハイブリッド並列は多少並列性能が劣化しています。

- 実効効率(ピーク性能比)

ベクトル型スーパーコンピュータが高性能を発揮しており、SX-7 の MPI 分散並列では実効効率が 89.5%、SMP と MPI の共有分散ハイブリッド並列では実効効率が 76.9%となっています。スカラ型スーパーコンピュータでは SGI Altix がやや高性能で、実効効率が 67.7%となっています。 ただし HPC チャレンジは、実効効率が評価の対象とはなっていません。

-性能評価結果

HPL を実行するには、入力データのサイズ、解法のパラメータ、ブロックサイズのパラメータを決める必要があります。まずは、HPL のサブルーチン毎の計算時間の内訳(今後、このような内訳を「コスト分布」と呼びます)について説明します。表 1 は、入力データサイズを N=30,000、解法パラメータを Right Looking、ブロックサイズパラメータを NB=64 とし P=1、Q=32 として測定した結果です。HPL 全体では、データ生成と結果検証もありますが、性能測定に関係する部分のコストのみを抜き出します。実行は MPI 32 プロセスで、単位は秒です。

表 1　HPL のコスト分布

HPL　(Target portion for measurement)			
94.3　(sec)			
HPL_dgemm calculation	HPL_bcast_1ring communication	HPL_dtrsm calculation	HPL_pdupdateTT calculation
71.8	19.9	1.6	0.9

一番計算時間が長い、すなわち計算コストの一番大きいのは HPL_dgemm で、これは行列積となります。行列積の計算は、BLAS ライブラリの DGEMM を使っています。この時の測定では DGEMM の実効効率は 97%と高い性能が発揮されています。また、HPL_bcast_1ring はデータ転送の部分で、MPI の SEND と RECV の関数を使って転送が行われています。HPL のコストのほとんどがこれら 2 つの部分で占められます。LU 分解のオーダ評価を行うと、演算部分は N の 3 乗に比例しており、転送部分は N の 2 乗に比例しています。したがって、N が大きくなるとほとんどのコストが DGEMM になり、ベースラインランでの HPL の実効効率は 90%程度になると予想されます。

次に、ブロックパラメータ NB, P, Q, NBMIN が HPL 実行結果にどのように影響を及ぼすかについて検討します。まずは、NB=64, NBMIN=64 に固定して評価します。この時、実行時間を節約するため、小さなサイズのデータ N=20,000 で評価した結果が表 2 となります。P と Q が行列データの分散方法を決めるパラメータで、P と Q の積が MPI のプロセス数になります。なお、項目 T/V は HPL の解法を表します。HPL は LU 分解の解法に、3 つの選択肢(Left looking, Crout, Right looking)がありますが、本実験では一番性能が良かった Right Looking(外積型ガウスの消去法)を採用しました。表 2 中 T/V の項目の R の次の数字が NBMIN のサイズとなっています。

表 2　ブロックパラメータ(P, Q)の HPL 性能に対する影響

T/V	N	NB	P	Q	Time	Gflop/s
W10R3R64	20000	64	1	24	33.80	1.578e+02(82.1%)
W10R3R64	20000	64	2	12	33.91	1.573e+02(81.9%)
W10R3R64	20000	64	3	8	35.37	1.508e+02(78.5%)
W10R3R64	20000	64	3	6	46.73	1.141e+02(79.2%)
W10R3R64	20000	64	4	4	51.79	1.030e+02(81.1%)
W10R3R64	20000	64	6	3	48.70	1.095e+02(76.0%)
W10R3R64	20000	64	8	2	55.01	9.696e+01(67.3%)
W10R3R64	20000	64	12	1	78.96	6.756e+01(70.3%)

この結果を見ると、P=1 に固定するのが一番良い性能となることが分かります。以降 P=1 に固定して評価を続けます。

次に、入力データサイズを N=10,000 に縮小し、プロセス数を 48 に固定(P=1, Q=64)して、NBMIN と NB のパラメータ依存性を調べた結果が表 3 となります(注:このテストは SX-6 6 ノードを使って評価しています)。NBMIN の組み合わせは NBMIN=32、44、64、144 で行い、NB について NB=64、128、256 の±1 前後のパラメータで評価しました。

表 3 プロックパラメータ(NBMIN, NB)の HPL 性能に対する影響

T/V(Change:NBMIN)	N	NB	P	Q	Time	Gflop/s
WC10R3R32	10000	63	1	48	3.16	2.11E+02
WC10R3R44	10000	63	1	48	3.13	2.13E+02
WC10R3R64	10000	63	1	48	3.14	2.12E+02
WC10R3R144	10000	63	1	48	3.75	1.78E+02
WC10R3R32	10000	64	1	48	3.14	2.12E+02
WC10R3R44	10000	64	1	48	3.12	2.14E+02
WC10R3R64	10000	64	1	48	3.1	2.15E+02
WC10R3R144	10000	64	1	48	3.08	2.17E+02
WC10R3R32	10000	65	1	48	3.22	2.07E+02
WC10R3R44	10000	65	1	48	3.37	1.98E+02
WC10R3R64	10000	65	1	48	3.18	2.10E+02
WC10R3R144	10000	65	1	48	3.16	2.11E+02
WC10R3R32	10000	127	1	48	3.86	1.73E+02
WC10R3R44	10000	127	1	48	3.89	1.72E+02
WC10R3R64	10000	127	1	48	4.06	1.64E+02
WC10R3R144	10000	127	1	48	4.2	1.59E+02
WC10R3R32	10000	128	1	48	3.91	1.71E+02
WC10R3R44	10000	128	1	48	3.61	1.85E+02
WC10R3R64	10000	128	1	48	3.98	1.67E+02
WC10R3R144	10000	128	1	48	4.17	1.60E+02
WC10R3R32	10000	129	1	48	4.06	1.64E+02
WC10R3R44	10000	129	1	48	4.04	1.65E+02
WC10R3R64	10000	129	1	48	4.16	1.60E+02
WC10R3R144	10000	129	1	48	4.28	1.56E+02
WC10R3R32	10000	255	1	48	5.97	1.12E+02
WC10R3R44	10000	255	1	48	6.05	1.10E+02
WC10R3R64	10000	255	1	48	6.16	1.08E+02
WC10R3R144	10000	255	1	48	6.55	1.02E+02
WC10R3R32	10000	256	1	48	5.92	1.13E+02
WC10R3R44	10000	256	1	48	6.22	1.07E+02
WC10R3R64	10000	256	1	48	6.19	1.08E+02
WC10R3R144	10000	256	1	48	6.5	1.03E+02
WC10R3R32	10000	257	1	48	6.13	1.09E+02
WC10R3R44	10000	257	1	48	6.13	1.09E+02
WC10R3R64	10000	257	1	48	6.14	1.09E+02
WC10R3R144	10000	257	1	48	6.56	1.02E+02

結果を見ると、表 3 から NBMIN に関してはパラメータ依存性がはっきりしませんが、NBMIN=64、144 は性能劣化していることが確認されました。NB に関しては、サイズを大きくすると性能劣化していること分かりました。以上の結果から HPL のブロックパラメータは、NBMIN=32 または 44、NB=64 が望ましいことが分かり、入力データを大きくする時には、このパ

ラメータを採用することにしました。

最後に、2004 年 12 月現在登録されている HPL の値を図 1 に示します。全体ノードが大きいほど高性能となるため、シングルノードの SX-7 の順位は 49 位中 MPI 並列版が 31 位、SMP +MPI 並列版が 39 位となっています。トップは 252CPU 構成の Cray X1 であり、24 ノード(192CPU)構成の SX-6 が、Cray X1 に続くグループとなっています。

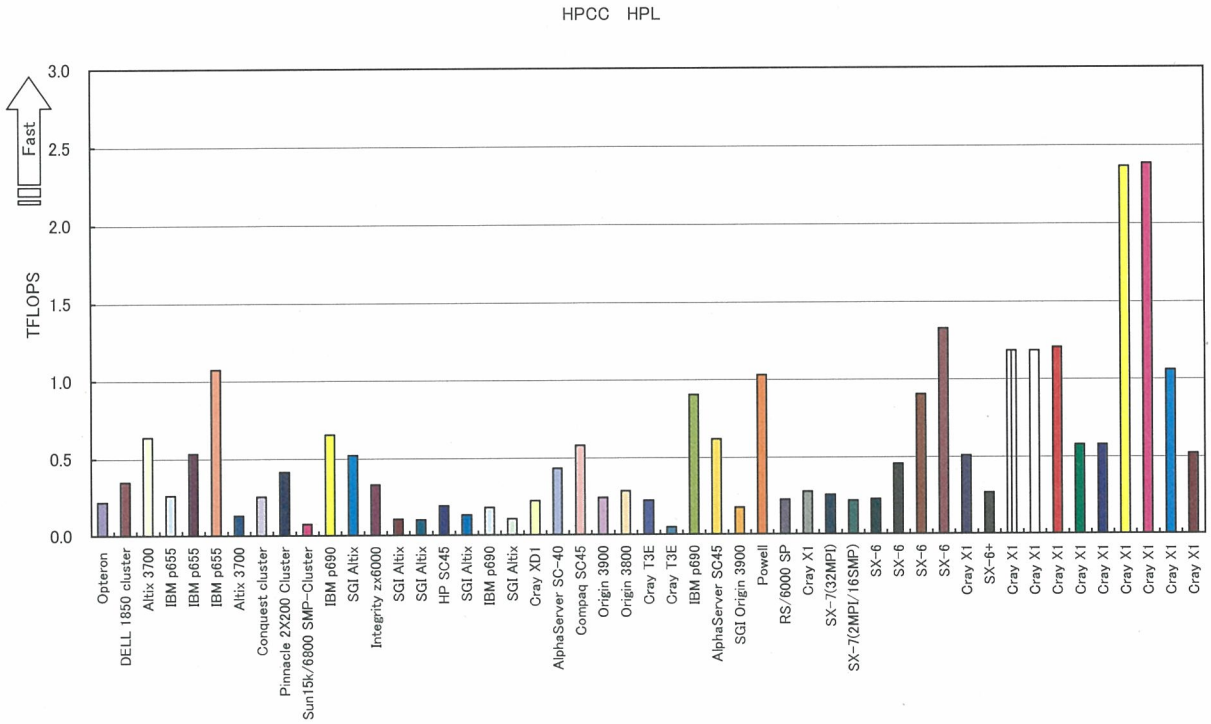


図 1 HPL の結果

5．2 DGEMM

- データサイズ

DGEMM のデータサイズは、HLP のデータサイズ N と MPI のプロセス数から次の式で決められます。

DGEMM データサイズ ＝HPL N／2／MPI プロセス数の平方根

- SMP 並列処理の導入効果

十分に SMP 並列化された BLAS を利用することで、DGEMM のテストは高度に並列化された結果が得られます。

- 実効効率

シングル環境(SN)での DGEMM の実効効率はピーク性能に近い 99%以上の性能を発揮します。また多重負荷環境(EP)においても実効効率は 97%以上となっています。

- 性能評価結果

2004 年 12 月現在登録されているシングル環境(SN)の結果を図 2 に示します。データのないプラットフォームもありますが、これは古いバージョンの HPCC の結果で、DGEMM の項目が入っていない時期のものとなっています。SX-7 は、高性能な 16CPU を使った SMP 共有並列処理により、他のシステムと比較して圧倒的に優れた演算性能を示しています。

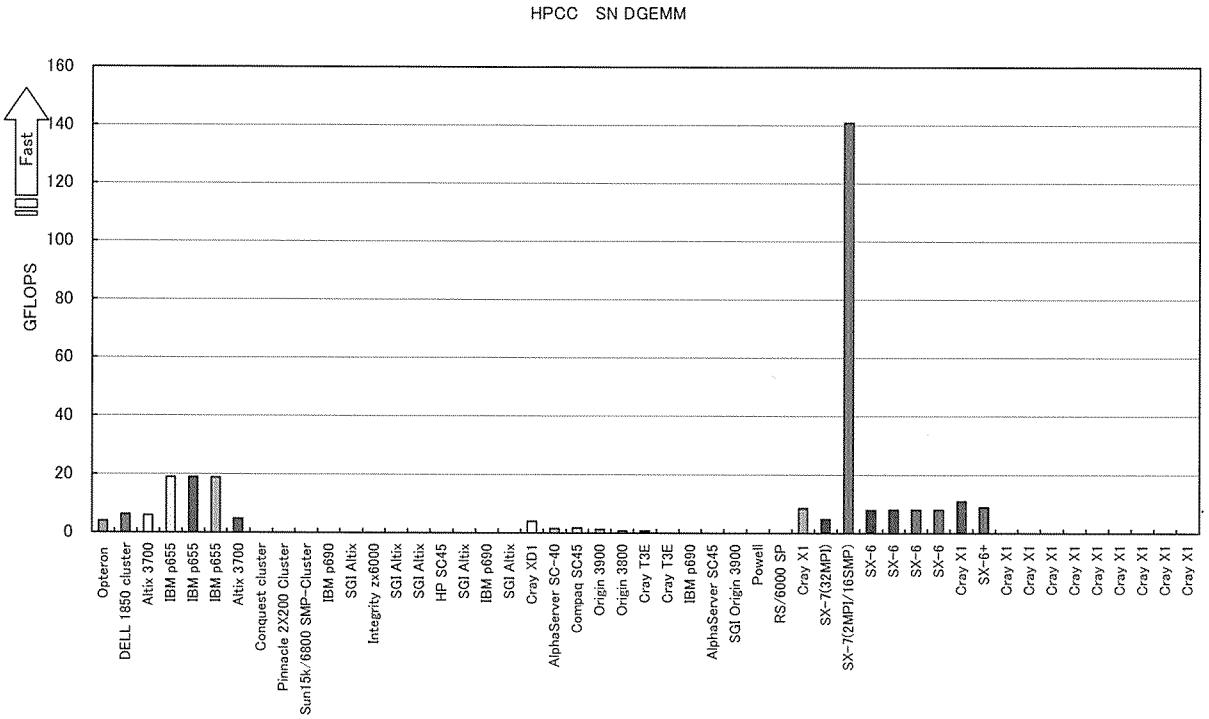


図 2　DGEMM の結果(シングル環境)

5. 3 STREAM

- データサイズ

STREAM の配列サイズは、HLP のデータサイズ N と MPI のプロセス数から次の式で決められます。

STREAM 配列サイズ ＝HPL N^2／MPI プロセス数／3

- SMP 並列処理の導入効果

オリジナルコードに OpenMP による SMP 並列化の指示行が入っており、並列化されます。実行するとノード内において並列化された結果が得られます。表 4、5 は全 32CPU 中、SMP 並列に 16CPU、多重負荷として 2 プロセス使用して実行した結果です。STREAM 配列サイズ =620,166,666 としました。多重負荷環境(EP_STREAM)は、シングル環境(SN_STREAM)での実行に比べてメモリ負荷がかかるため、性能が劣化していることが分かります。

表 4　多重負荷環境での STREAM 性能

EP_STREAM_Copy	389.791 GB/s
EP_STREAM_Scale	348.593 GB/s
EP_STREAM_Add	428.084 GB/s
EP_STREAM_Triad	492.161 GB/s

表 5　シングル環境での STREAM 性能

SN_STREAM_Copy	537.486 GB/s
SN_STREAM_Scale	379.734 GB/s
SN_STREAM_Add	437.240 GB/s
SN_STREAM_Triad	556.609 GB/s

-性能評価結果

2004 年 12 月現在登録されているシングル環境の積和の結果(SN_STREAM_Triad)と多重負荷環境の積和の結果(EP_STREAM_Triad)を図 3、4 に示します。ベクトルロードストアユニットを有するベクトル機のメモリ性能の良さが際立っています。特に SX-7 は共有並列を最大限に活かされた結果となっており、16CPU を使った SMP 共有並列の高性能が発揮されて、圧倒的に優れたメモリ性能が示されています。

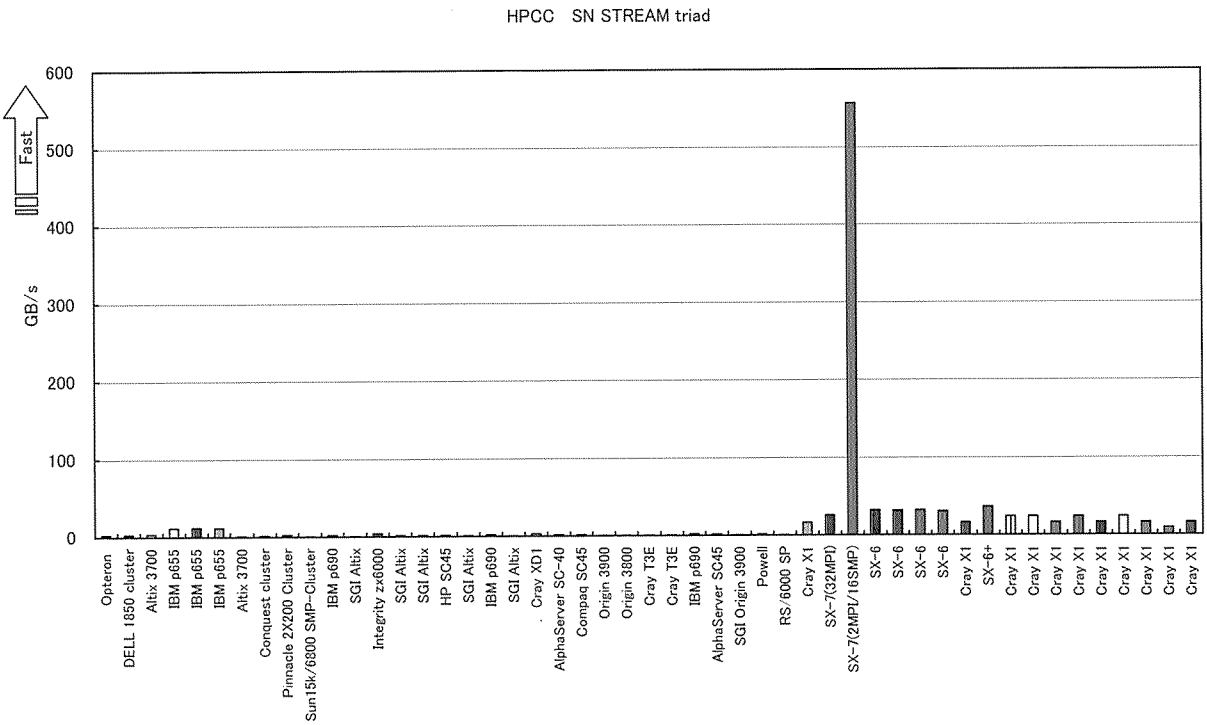


図 3 シングル環境 SN_STREAM_triad の結果

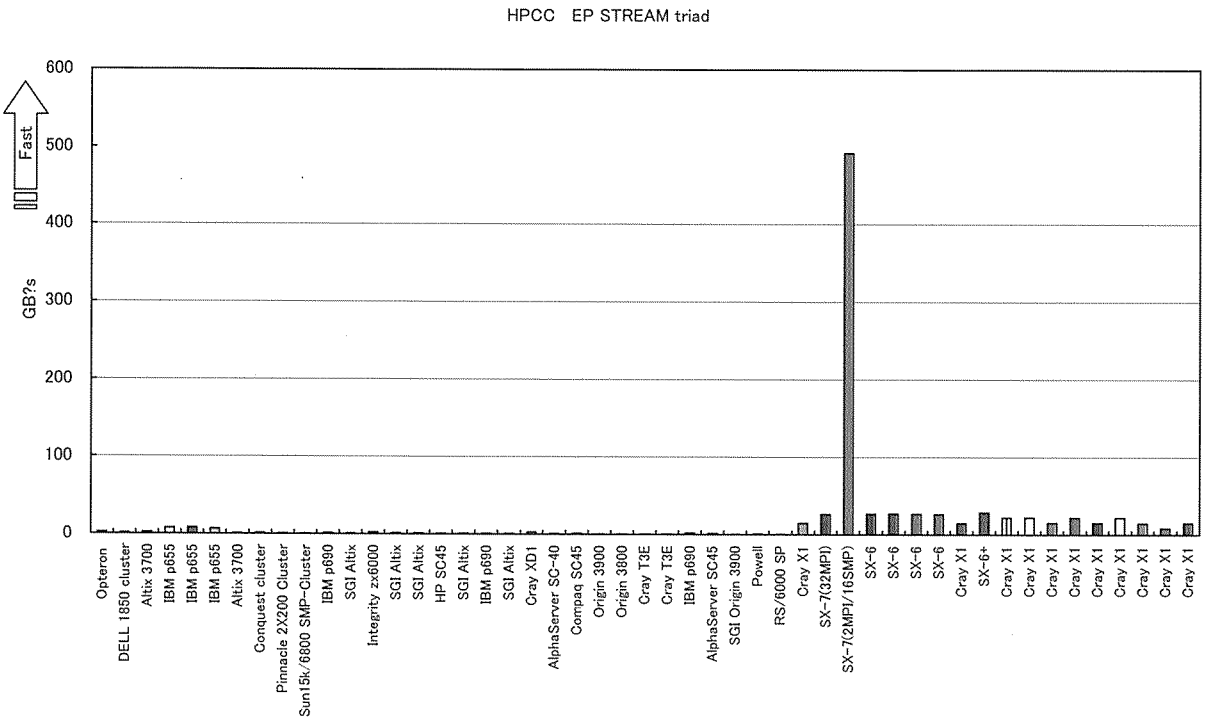


図 4　多重負荷環境 EP_STREAM_triad の結果

5．4 PTRANS

- データサイズ

PTRANS のサイズは、HLP のデータサイズと同じ N、または個別に PTRANS のサイズ N を決めることができます。転置される配列のサイズは、N／2 となっています。

-性能評価結果

PTRANS は MPI のデータ転送による通信部分(sendrecv)が、コストの大部分となっています。測定区間の転送処理全体のコスト分布(実行時間の内訳)は表 6 のようになります。この時使用したパラメータは、N=90,000、NB=471、P=1、Q=32 としました。

表 6　PTRANS のコスト分布

ptr_trans (Target portion for measurement)				
0.651　(sec)				
intrans-1,2	dtr2mx	dtr2b	sendrecv	intrans-4
0.000	0.006	0.038	0.541	0.066

まずは、通信について説明します。行列の転置を行うため、次の図 5 に示されているような転送方法に最適化されています。すなわち、データ転送は、(自分のランク番号－1)のランクから始まって、ランク番号を減少させていく方向に、それぞれのプロセスがデータを転送します。更にランク 0 のプロセスの後に、最後のランクから始まってランク番号を減らす方向に同じ方法でデータ転送が行われます。これに対応して、データの受信は、(自分のランク番号+1)のランクから始まってランク番号が増えていく方向に行われます。このようなスケジューリングに基づいて、

データの送受信の競合を避けるように最適化が行われています。

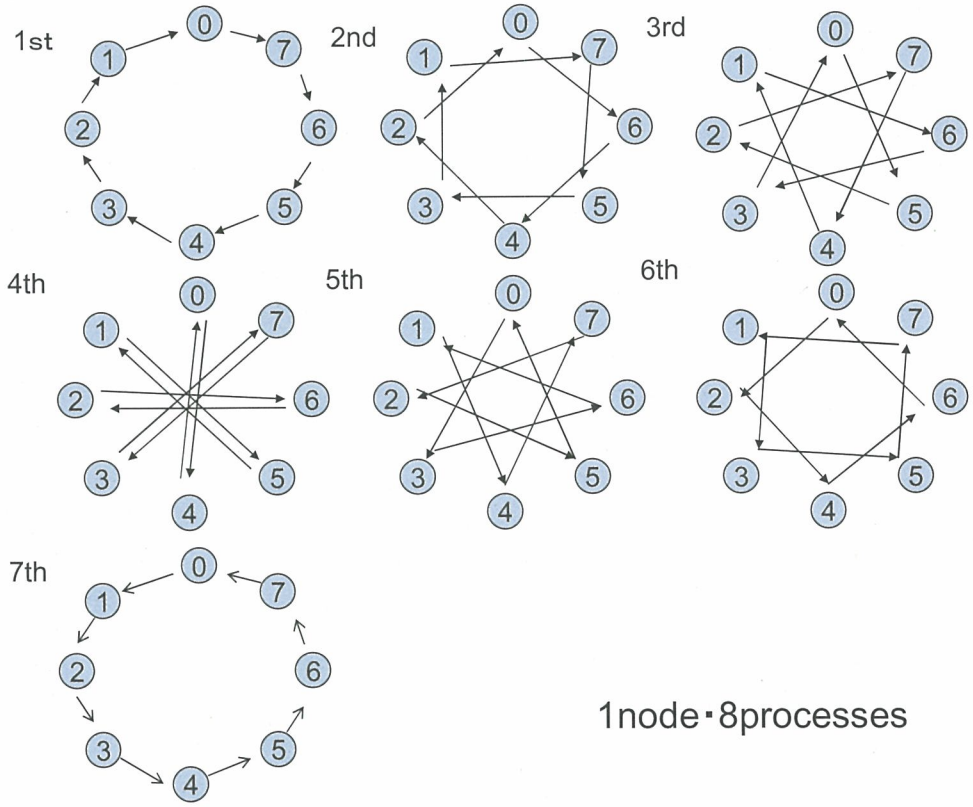


図 5　最適な転置方法

PTRANS の性能に関して、通信コストが最も大きな要素でありますが、通信以外の処理コストの時間も BM 計測区間の大きな要素になっています。このコストを調べるため、N=10,000 に固定して NB を色々な組み合わせで評価しました。まずは、評価区間内の性能(Performance)と転送時間(transmission)、バンク競合時間(bank)を調査しました。表 7 に結果を示します。

表 7　PTRANS コスト分布

N	Performance (GB/s)	transmission (sec)	bank (sec)	Number of processes
10000	7.423	0.024	0.0062	4(NB= 90)
10000	7.871	0.023	0.0066	4(NB=100)
10000	7.035	0.026	0.0111	4(NB=104)
10000	8.543	0.022	0.0064	4(NB=105)
10000	11.231	0.014	0.0011	8(NB=105)
10000	7.091	0.026	0.0112	4(NB=106)
10000	6.496	0.029	0.0118	4(NB=110)
10000	6.424	0.029	0.0119	4(NB=120)
10000	5.972	0.031	0.0131	4(NB=200)

測定区間に含まれるバンク競合時間が、転送時間と同じ程度の時間になるため、性能全体に影響していることがわかります。また、コスト分布から、非通信部分のコストは 17%も占められていることがわかり、この部分を高速化することにより、全体の性能を改善させることが出来ます。これを調べるため、NB、P、Qの様々な組み合わせを用いて測定した結果が以下の表8となります。一番右側の RESID が検証結果で、0.00 が正しい結果であることを示しています。

表 8　PTRANS ブロックパラメータの評価

TIME	M	N	MB	NB	P	Q	TIME	CHECK	GB/s	RESID
WALL	45,000	45,000	150	150	1	8	0.94	PASSED	17.266	0.00
WALL	45,000	45,000	187	187	1	8	0.84	PASSED	19.237	0.00
WALL	45,000	45,000	241	241	1	8	1.16	PASSED	13.952	0.00
WALL	45,000	45,000	255	255	1	8	0.82	PASSED	19.828	0.00
WALL	45,000	45,000	255	255	2	4	4.15	PASSED	3.901	0.00
WALL	45,000	45,000	471	471	1	8	0.93	PASSED	17.356	0.00
WALL	45,000	45,000	105	105	1	16	1.09	PASSED	14.889	0.00
WALL	45,000	45,000	143	143	1	16	1.05	PASSED	15.427	0.00
WALL	40,000	40,000	147	147	1	16	0.89	PASSED	14.336	0.00
WALL	45,000	45,000	150	150	1	16	1.01	PASSED	15.991	0.00
WALL	45,000	45,000	187	187	1	16	1.45	PASSED	11.166	0.00
WALL	45,000	45,000	200	200	1	16	1.15	PASSED	14.050	0.00
WALL	45,000	45,000	241	241	1	16	1.01	PASSED	15.992	0.00
WALL	45,000	45,000	255	255	1	16	1.13	PASSED	14.386	0.00
WALL	45,000	45,000	300	300	1	16	1.21	PASSED	13.437	0.00
WALL	45,000	45,000	450	450	1	16	1.12	PASSED	14.491	0.00
WALL	45,000	45,000	471	471	1	16	0.93	PASSED	17.353	0.00
WALL	45,000	45,000	150	150	1	32	0.76	PASSED	21.208	0.00
WALL	45,000	45,000	187	187	1	32	0.81	PASSED	19.944	0.00
WALL	45,000	45,000	241	241	1	32	0.71	PASSED	22.757	0.00
WALL	45,000	45,000	255	255	1	32	0.77	PASSED	20.995	0.00
WALL	45,000	45,000	471	471	1	32	0.66	PASSED	24.569	0.00

表 8 を見ると、最も性能を支配しているパラメータは NB で、適切な NB の値を選ぶことが、PTRANS の性能を決める重要な要素となっていることがわかります。コードを解析すると、通信と通信以外の両方にパラメータNBが影響しています。NB値は、通信以外の処理ではベクトル長になっており、このNB値によってループのストライドが変化するために、バンク競合が発生するコードになっています。また、バンクコンフリクトを回避するため、NB を奇数にすることが考えられますが、通信処理の影響で必ずしも奇数とすることが最良の結果を得ることにならないことも分かりました。

2004年12月現在登録されているPTRANSの値を図6に示します。ノード数が大きいほど高性能となるため、シングルノードの SX-7 の順位は低くなっています。トップは 252CPU 構成の Cray X1 であり、24 ノード(192CPU)構成の SX-6 が、それに続きます。

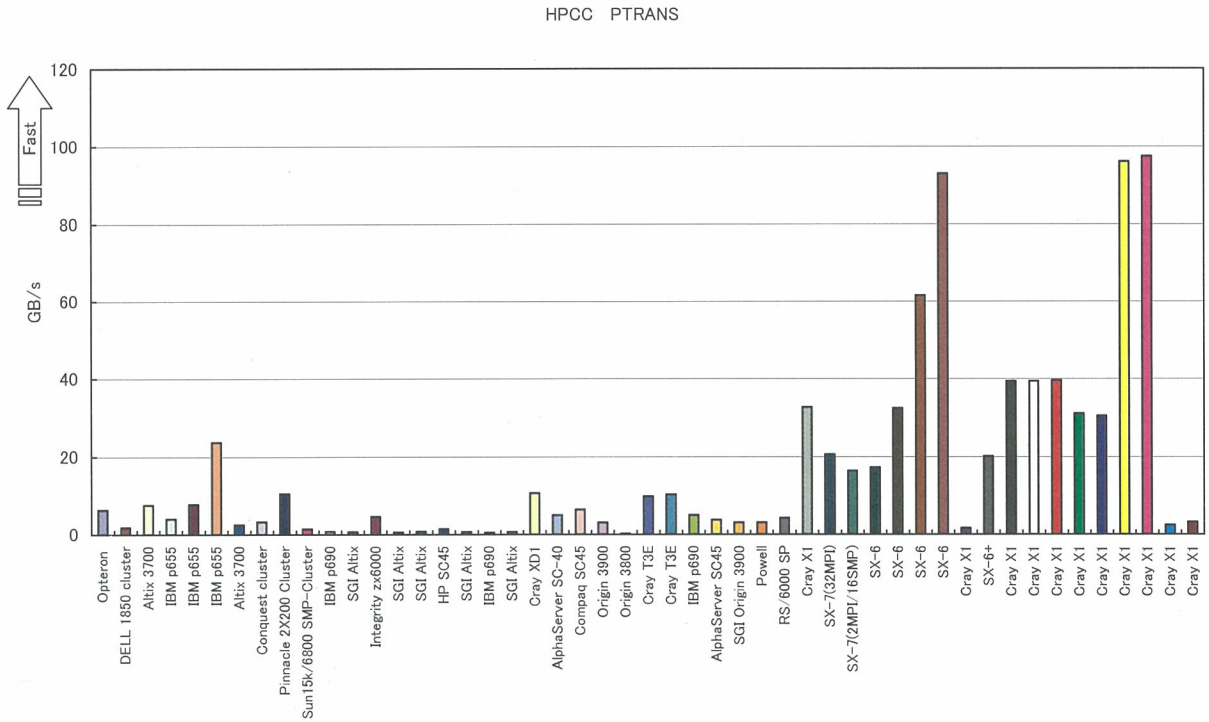


図6 PTRANSの結果

5．5 RandomAccess

- データサイズ

ノード単体テストの RandomAccess のテーブルサイズ(rand の範囲)と更新領域サイズ(配列 Table の大きさ)は、HLP のデータサイズ N と MPI のプロセス数から次の式で決められます。

テーブルサイズ= HPL N^2／MPI プロセス数

更新領域サイズ＝テーブルサイズ×4

- メモリアクセス方法

ノード単体テストでは、ノード内のメモリのランダムアクセス(インダイレクトアクセス)性能を評価します。図 7 で示すように、乱数で作成されたテーブル rand(i)に基づき、配列データを順次格納して、メモリのランダムアクセス性能を評価します。

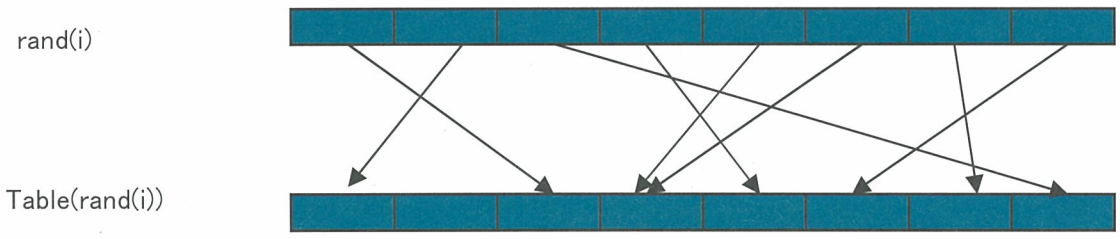


図7 ランダムアクセスの方法

- 性能評価結果

シングル環境(SN)と多重負荷環境(EP)の両方において、ベクトル機はメモリランダムアクセス性能が高いため、SX の性能は高い順位に位置しています。なお、ランダムアクセスのループ長は 128 と短いループ長に固定されているため SMP 並列化が困難となっています。また全ノードテストでは、ノード間の MPI 転送性能の評価を目的としており、同時データアクセス性が評価され、トータル転送パスが多いほど高性能となっています。2004 年 12 月現在登録されているシングル環境の結果を図 8 に示します。ノード単位テストでは、SX、Cray X1 などのベクトル機がトップグループになっており、ベクトル型スーパーコンピュータのメモリアクセス性能の良さが評価されています。

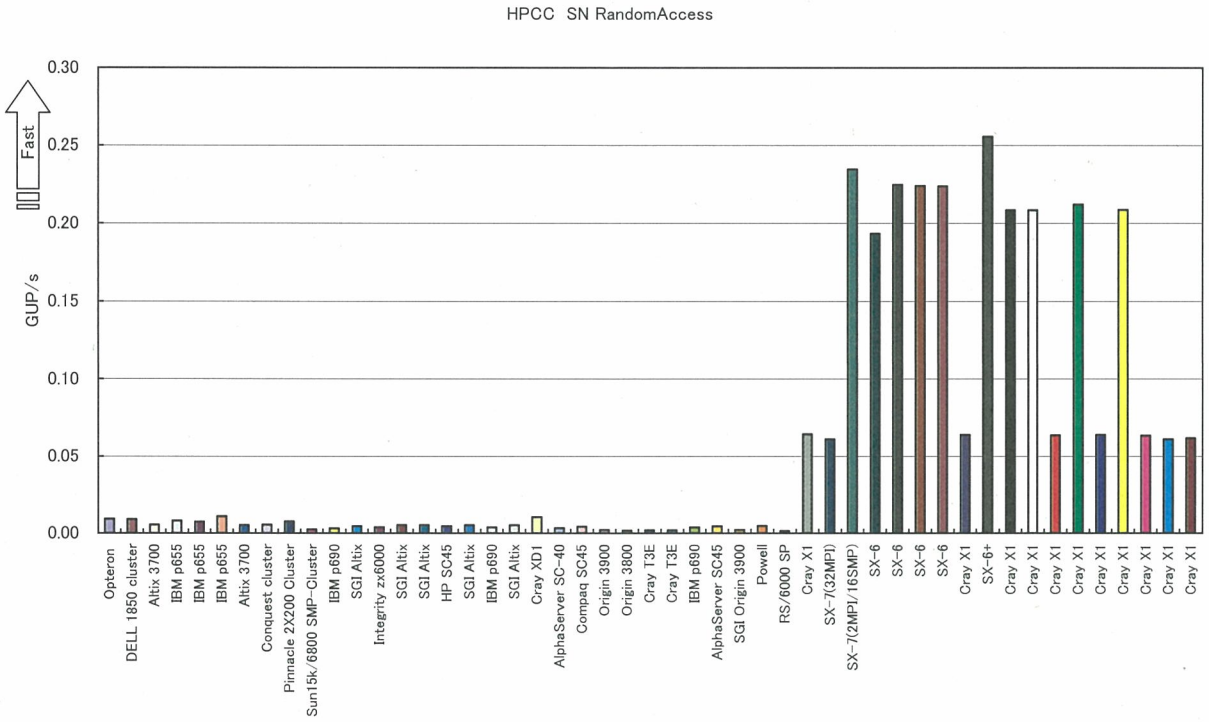


図 8 シングル環境の RandomAccess の結果

また、図 9 は MPI による全ノードテストの結果です。全ノードテストは、ネットワーク全体の総合性能が評価されるため、ノード(CPU)数の少ない SX は高い性能とはなっていません。

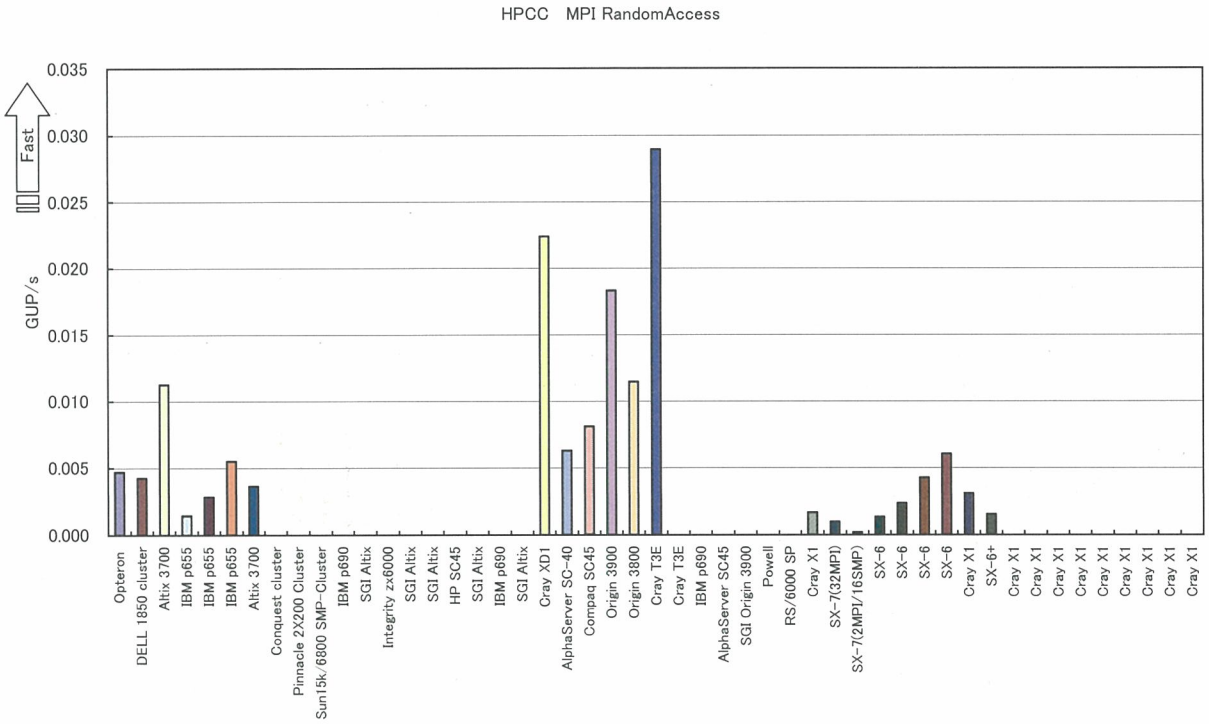


図 9　全ノードテストの RandomAccess の結果

5．6 FFTE

- データサイズ

FFT のサイズは、HLP のデータサイズ N と MPI のプロセス数から、次の計算式の値を超えない最大の 2 のべき乗の値となっています。

N^2／プロセス数／2／(struct fftw_complex のサイズ)　(SN_FFTE, EP_FFTE)

N^2／プロセス数／3／(struct fftw_complex のサイズ)　(G_FFTE)

-性能評価結果

FFT のコードの中で、L2SIZE というパラメータがあり、キャッシュサイズを指定するのに使われています。L2SIZE の値は使用したコンピュータハードウェアに合わせて変更する必要がありますが、ベースラインランの実行ルールでは、このパラメータの変更は許されておらず、固定のサイズとなっています。このように現時点では、FFT のコードは、十分最適化されておらず、登録された各マシンの評価結果は不十分な値となっています。

2004 年 12 月現在登録されているシングル環境の結果を図 10 に示します。シングル環境では、SX-7 がトップとなっています。また、図 11 に示す全ノードテストでは、SX システムがトップグループとなっています。

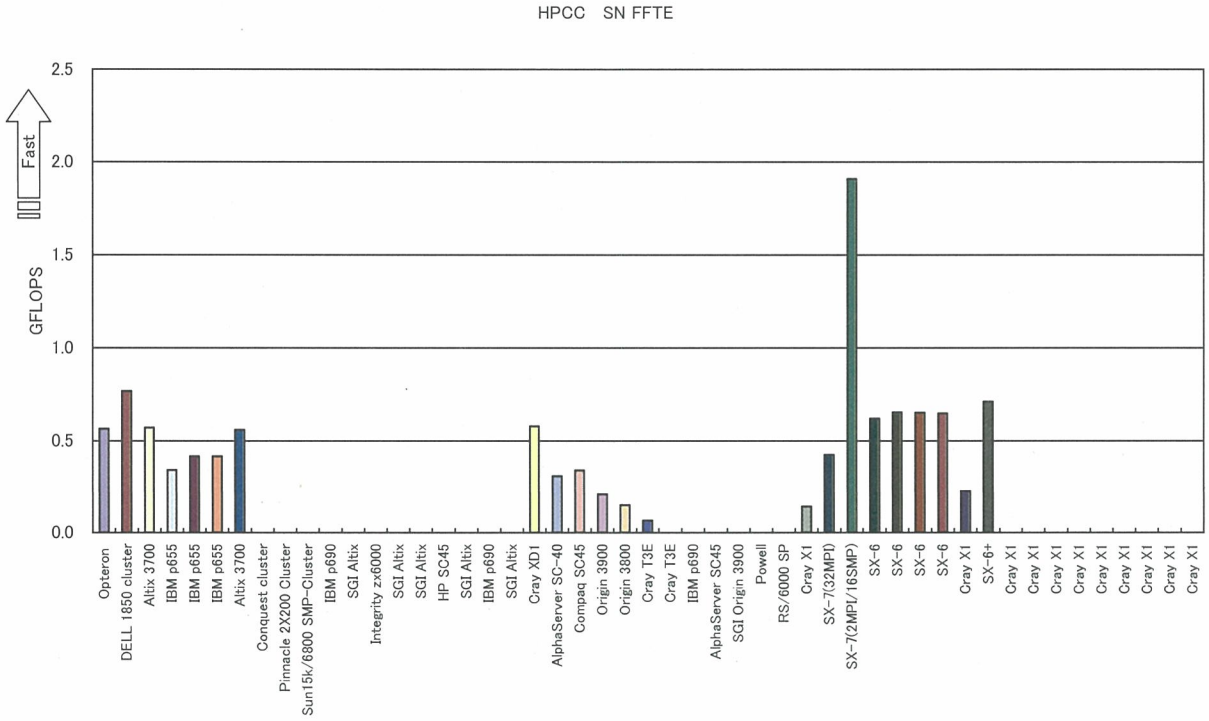


図 10 シングル環境の SN_FFTE の結果

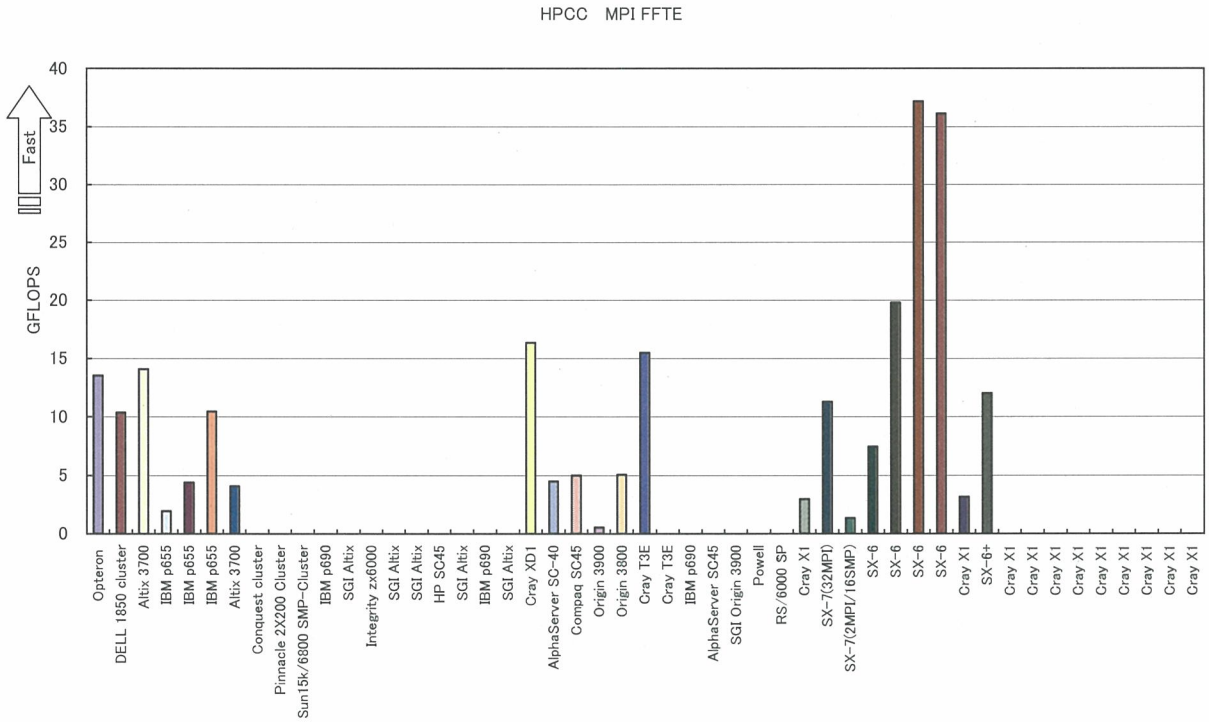
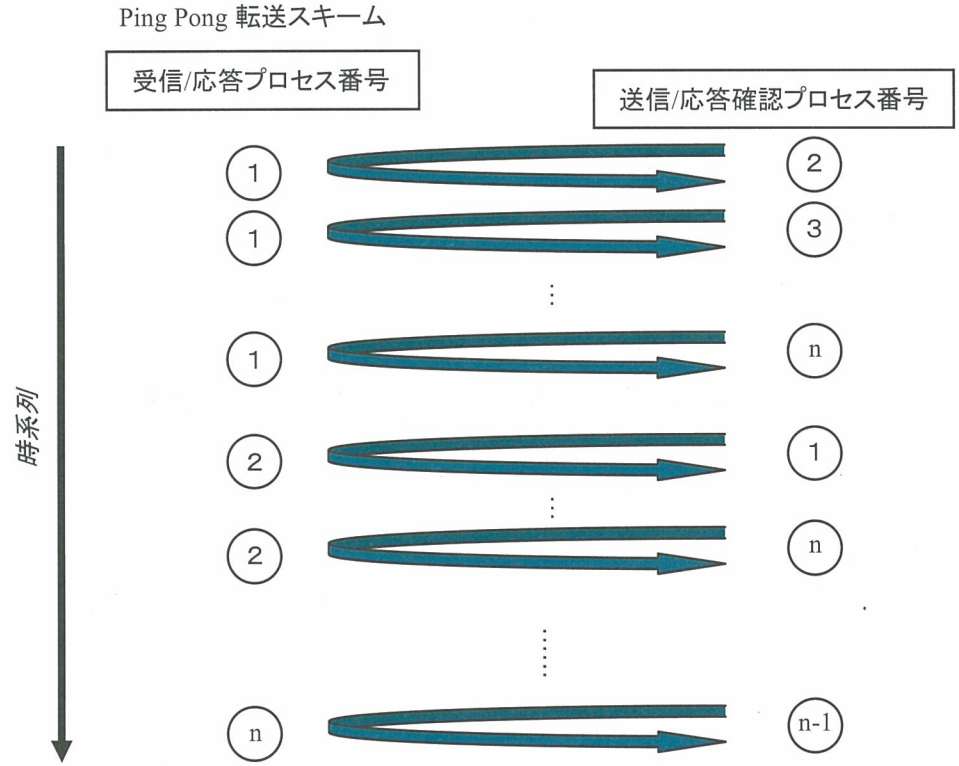


図 11 全ノードテストの G_FFTE の結果

5. 7 Communication bandwidth and latency

- 転送スキーム

バンド幅およびレイテンシの評価のスキームは Ping-Pong 転送スキームと Ring 転送スキームの 2 つあります。このうち、Ring 転送スキームはデータを MPI のランク順に転送する方法と、ランダムに転送する方法があります。この 2 つのスキームを使って、バンド幅テストでは 2M バイトのデータを転送し、レイテンシーテストでは 8 バイトのデータを転送してデータ転送性能の評価を行います。図 12 は、これら 2 つの転送スキームを簡単に説明したものです。



Ring 転送スキーム

Naturally ordered （MPI_COMM_WORLDでのランクの並びでとなりに転送）

Randomly ordered （乱数を使って作ったランクの並びでとなりに転送）

図 12 転送スキーム

-性能評価結果

Communication bandwidth and latency の内、2004 年 12 月現在登録されているレイテンシ 5 項目の内、Randomly ordered Ring のレイテンシの性能を図 13 に示します。SX-7 はノード内の転送のため、比較的高性能の部類に属します。最速は、Cray XD1 になっています。

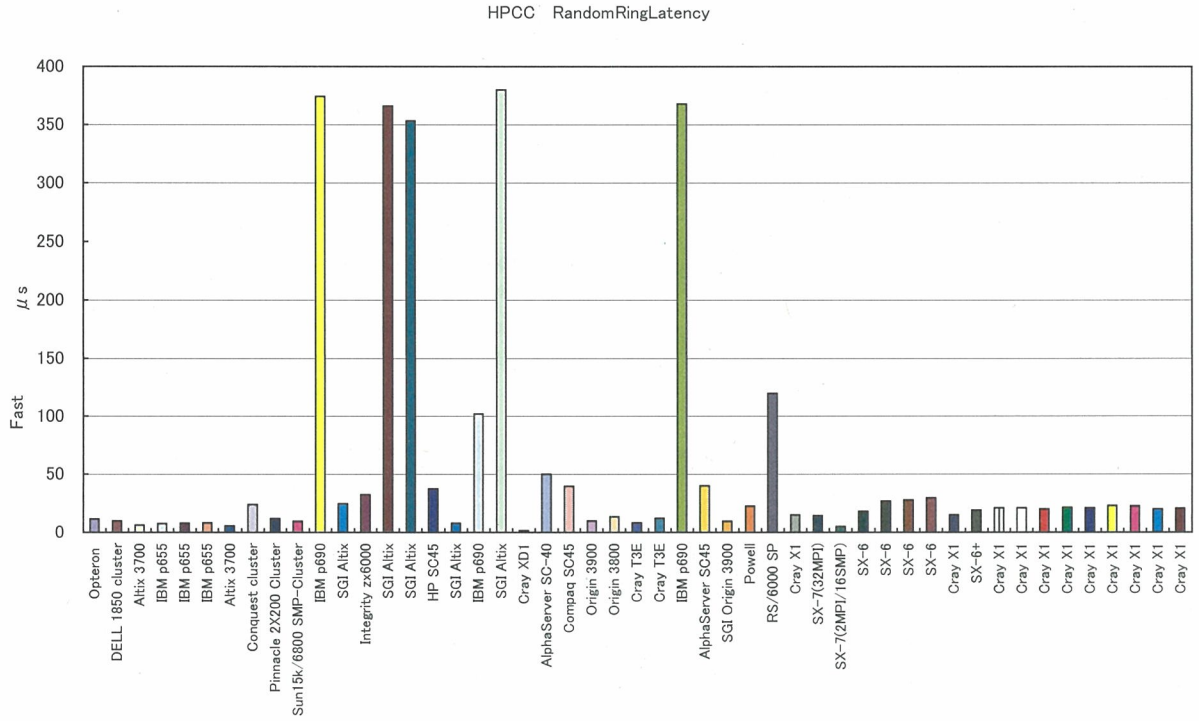


図 13　Randomly ordered Ring のレイテンシの性能

バンド幅の転送性能を見ると、Naturally Ordered Ring と Randomly Ordered Ring の性能に大きな差があります。この差はノード間の転送データ量の合計の差によるものであり、Naturally Ordered Ring の場合、ノード間転送を最小にするための最適化を簡単に行えますが、Randomly Ordered Ring は通信パターンが複雑なため最適化が困難になっています。2004 年 12 月現在登録されているバンド幅 5 項目の内、Naturally Ordered Ring のバンド幅の結果を図 14 に示します。SX はトップグループになっていることが分かります。

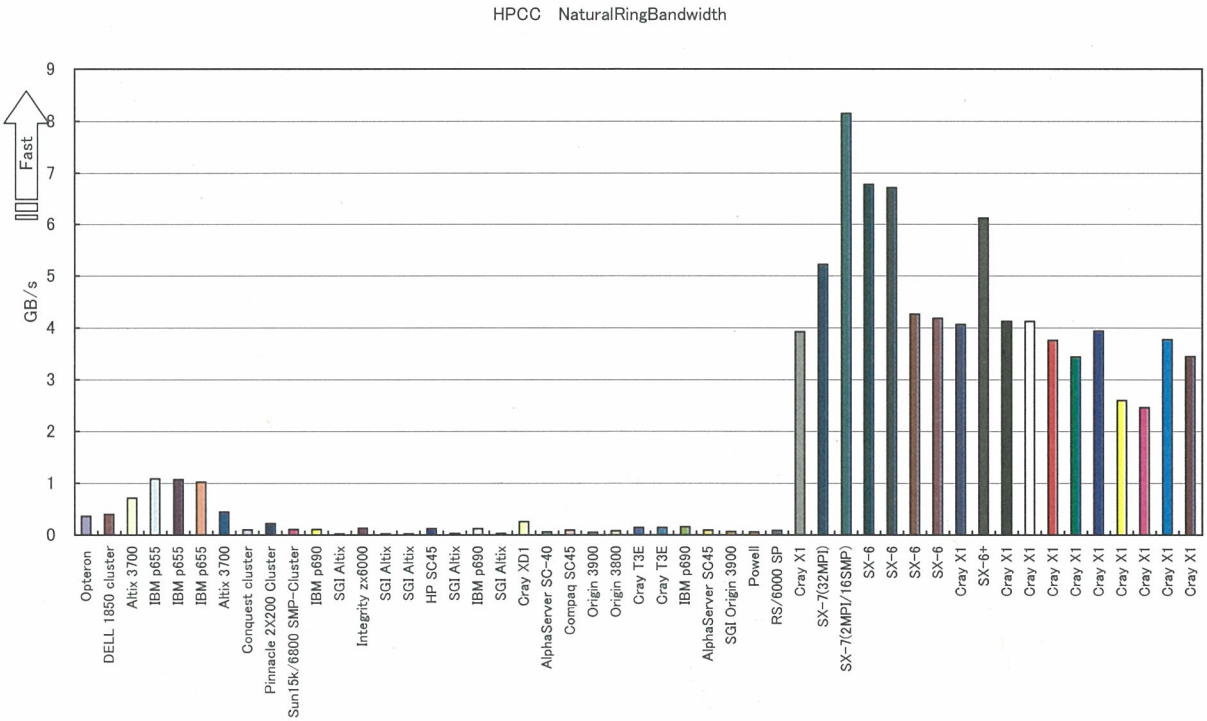


図 14 Naturally Ordered Ring のバンド幅の性能

6. 考察

6. 1 HPCC のアプローチに対して評価できる点

1)Linpack 以外の本格的な HPC 領域の BM プログラムとなっている点があげられます。性能評価として、Linpack のような演算に限定された測定指標と異なり、メモリバンド幅性能、ネットワーク性能、基本カーネル(まだ不十分ながら行列 DGEMM、FFT)が含まれており、より総合評価となる測定指標へのアプローチとなっています。

2)主要な HPC ユーザおよびベンダが参画している点をあげられます。具体的には、CRAY X1、IBM Power5、SGI Altix など、ベクトル型およびスカラ型の主要なプラットフォームの性能が登録済みとなっています。また、SX の結果も、東北大からだけでなく、HLRS(独)、DKRZ(独)からも登録されています。

3)運営・評価体制がそれなりに確立されている点があげられます。HPC チャレンジプログラムは DoD/DARPA の援助を受け実施されており、委員も J. Dongarra 博士(Linpack ベンチマーク運営)、McCalpin 博士(STREAM ベンチマークの運営者)、と HPC ベンチマークの中心メンバーが参加しています。

6. 2 HPCC のアプローチにおける課題

1)総合指標の確立

複数の評価指標の集合体であり、総合指標が存在せず一義的な解釈困難となっている点があげられます。従って、総合指標の作成には、なお試行を要すると思います。これに対する 1 つ

の答えとして、我々は、図 15 に示すような評価方法を考えました。登録されている全結果の項目の順位をレーダーチャートで表し、外側が第 1 位として順位で正規化しています。外側にあるほど高順位であることを示しています。SX-7 の登録された結果(SMP 実行の方)で、SX-7 単体は 32CPU 構成であるため、HPL では順序が 49 位中 39 位としながら、メモリバンド幅や、ネットワーク性能、行列積、FFT で 1 位であることがわかります。このような方法で総合評価するのも一つの方法だと思います。

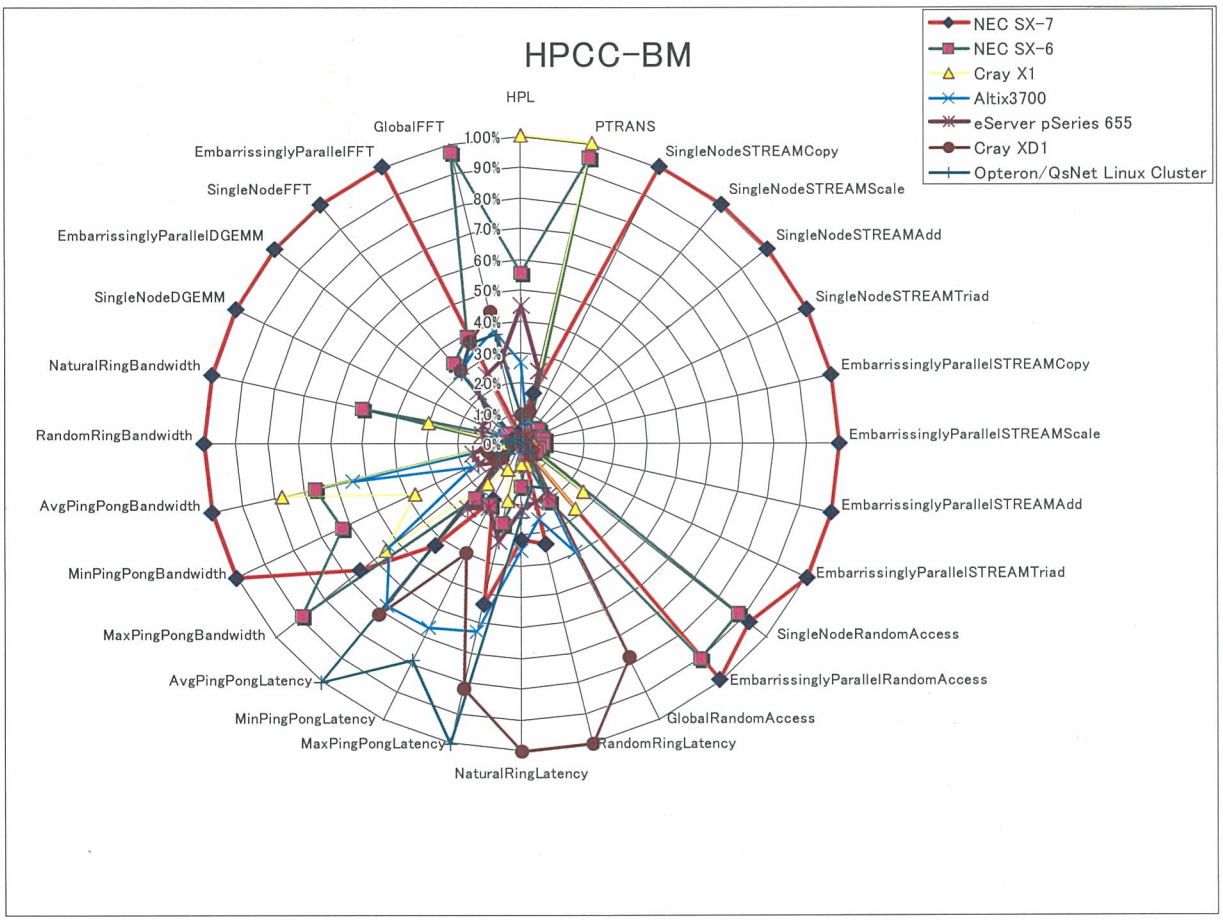


図 15 HPC チャレンジ順位のレーダーチャート

2)評価項目間での一貫性、公平性の確保

計測対象機の最大性能(演算、メモリ、ネットワーク)を「量」で評価する項目があげられます。Top500 の Linpack HPC のように、「量」のみで大規模なシステムを評価することも一つの見方ですが、この「量」でのみの性能評価とすると、Top500 と同じように今後単一ボリュームの大きなシステムしか性能評価のレースに参加できないという問題もでてきます。実際の計算では、ボリュームが大きくなると効率が下がる傾向にありますので、これを評価する指標として量による総性能評価に加えてそのときの稼働率、すなわち実効効率を評価することが大切だと考えます。必要な計算結果を効率よく得ることも一つの評価項目になるとなりますので、最大性能を評価する場合、「量」とその「実効効率」も評価の対象となることを期待しています。

3)簡明性の向上

指標の解釈には、内容の理解が必要なため、評価結果がわかりにくいという問題点があります。今後 HPC コミュニティやメディアを通したより一層の広報・普及活動を推進する必要があると思います。

7. おわりに

以上、HPCチャレンジによるSX-7の評価について述べました。様々な角度からスパコンを評価するHPCチャレンジベンチマークにおいて、情報シナジーセンターのSX-7は、28評価項目中16項目で最高性能を得ました。ベクトル型スーパーコンピュータのメモリ性能の高さに加え、SX-7では、SMP並列で32までと大きな共有並列化することができることから、HPC分野で高い潜在能力を持っていることを明らかにしました。情報シナジーセンターのSX-7のHPCチャレンジベンチマーク評価結果の登録[3]に対し、J. Dongarra 博士からは以下のようなコメントを受けています。" We are impressed with the continuing high performance of the SX family of processors. The SX-7 lives up to the expectations."

情報シナジーセンターの SX システムは、8 ノード(240CPU)からなる総性能 2.1Tflop/s のシステムですが、24 時間フル稼働の現在、常に 85%以上の CPU 利用率(2003～2004 年の実績で毎年度 6 月以降 90%以上、11 月以降 95%以上、実行待ちジョブ毎日 70～80 件)で動作しており、学内外の多くの研究者に活用されております。このような高性能なスーパーコンピュータコンピュータシステムを利用して、学会論文ばかりでなく、新聞紙上を賑わすような数多くの研究成果が生み出されております[4][5][6]。

HPCチャレンジベンチマークによるスーパーコンピュータの評価の試みはまだ始まったばかりで、今後、ベンダやユーザからのフィードバックを得ながら様々な改良が加えられ、スーパーコンピュータの総合的な評価指標として確立されていくと思われます。現在、HPCを支えるスーパーコンピュータシステムとして、ベクトル型スーパーコンピュータのようなカスタム設計によるもの、スカラ並列スーパーコンピュータのCOTS (Commercial Off-the-Shelf, 商用量産品)ベースのもの、そしてPCクラスタやGridなど様々なものがありますが、米国では、市場性重視でのHPCシステムの研究開発の危うさを2004年6月にHPC特別委員会報告書で指摘し[7]、米国システムが中心のTop500リスト中の296システムを占める高性能クラスタデザインによるスーパーコンピュータでは、国家安全保障の要求水準を満たすには不十分といった議論がなされています。また、2004年11月に米国ピッツバーグで開催されたSC2004では、ベクトル型スーパーコンピュータ(地球シミュレータ, CRAY-X1)とスカラ並列型スーパーコンピュータ(SGI Altix, IBM Power3/4)の実用的なアプリケーションを用いた性能比較の報告が米国Lawrence Berkeley National Laboratoryの研究グループからあり、運用開始後3年近くたった今でもベクトル並列型である地球シミュレータの実効性能の高さが示されました[8]。そのような背景の中、スーパーコンピュータの新しい評価ベンチマークの研究開発プロジェクトであるHPCチャレンジベンチマークが重要視されております。加えて、同年11月には「高性能計算再生法」が可決され、大統領署名をもって今後3年間に総額1億6600万ドルの予算がDOE(エネルギー省)が中心となって、HECS(High-End Computing Systems)の研究開発に投入されることが決まりました[9]。米国では、IBM BlueGene/Lが2004年11月のtop500ランキングで1位になった現在においても積極的、かつ継続的にHECS/HPCS研究開発計画が推進されています。日本においても、日本の先進科学技術分野における国際競争力を失わないために、実効性能に優れたHECS/HPCSの研究

開発を国策として継続的に支援するとともに、産学官の精力的な取り組みが必要不可欠と思います。

謝辞

今回の実験でご協力いただいた日本電気株式会社第一官庁システム開発事業部の撫佐昭裕氏、神山　典氏、金野浩伸氏に深く感謝いたします。

参考文献

[1] DARPA HPCS Program, http://www.highproductivity.org/.
[2] HPC Challenge, http://icl.cs.utk.edu/hpcc/index.html.
[3] HPCC 評価結果の全登録リスト, http://icl.cs.utk.edu/hpcc/hpcc_results_all.cgi.
[4] 東北大学大型計算機センター年報(昭和 54 年度～平成 10 年度).
[5] 東北大学大型計算機センター1999 年度～2000 年度の歩み.
[6] 東北大学情報シナジーセンター年報(平成 13 年度～平成 15 年度).
[7] Federal Plan for High-End Computing, Report of the High-End Computing Revitalization Task Force (HECRTF), May 10, 2004.
[8] Leonid Oliker, et al., "Scientific Computations on Modern Parallel Vector Systems," Proceedings of SC 2004, CD-ROM, 2004.
[9] Department of Energy High-End Computing Revitalization Act of 2004, http://thomas.loc.gov/, 2004.

Information Synergy Center Tohoku University

高速化推進研究活動報告　第3号
平成17年3月　発行
編集・発行　東北大学情報シナジーセンター
〒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/