ISSN 0286-7419

TOHOKU
UNIVERSITY

東北大学
サイバーサイエンスセンター

大規模科学計算システム広報

SENAC

Vol.45 No.2 2012−4

Supercomputing System
Cyberscience Center
Tohoku University
www.ss.isc.tohoku.ac.jp

大規模科学計算システム関連案内

＜大規模科学計算システム関連業務は、サイバーサイエンスセンター本館内の情報部情報基盤課が担当しています。＞

http://www.ss.isc.tohoku.ac.jp/

階	係・室名	電話番号(内線)* e-mail	主なサービス内容	サービス時間 平 日
一階	共同利用支援係 （受 付）	022-795-3406(3406) FAX:022-795-6099 uketuke@isc.tohoku.ac.jp	各種申請、講習会、利用相談、広報、センター業務全般に関する質問や要望の受付	8:30～17:15
	利用相談室	022-795-6153(6153) sodan05@isc.tohoku.ac.jp 相談員不在時 022-795-3406(3406)	計算機利用全般に関する相談 大判プリンタ、利用者端末等の利用	8:30～17:15 8:30～21:00
	利用者談話室	(3444)	各センター広報の閲覧	8:30～21:00
	展 示 室（分散コンピュータ博物館）		歴代の大型計算機等の展示	9:00～17:00
三階	庶務係	022-795-3407(3407) syomu@isc.tohoku.ac.jp	庶務に関すること	8:30～17:15
	会計係	022-795-3405(3405) kaikei@isc.tohoku.ac.jp	会計に関すること、負担金の請求に関すること	8:30～17:15
	共同研究支援係	022-795-6252(6252) rs-sec@isc.tohoku.ac.jp	共同研究、計算機システムに関すること	8:30～17:15
	共同利用支援係	022-795-6251(6251) uketuke@isc.tohoku.ac.jp	ライブラリ、アプリケーションに関すること	8:30～17:15
	ネットワーク係	022-795-6253(6253) net-sec@isc.tohoku.ac.jp	ネットワークに関すること	8:30～17:15
四階	研究開発部	022-795-6095(6095)		
五階	端末機室	(3445)	PC 端末機(X 端末)	8:30～21:00

＊（ ）内は東北大学内のみの内線電話番号です。青葉山・川内地区以外からは頭に 92 を加えます。

本誌の名前「SENAC」の由来

昭和 33 年に東北地区の最初の電子計算機として、東北大学電気通信研究所において完成されたパラメトロン式計算機の名前で SENAC-1(SENdai Automatic Computer-1)からとって命名された。

［お知らせ］

平成 24 年度サイバーサイエンスセンター講習会案内

No.	名　称	開催日時	講　師	内 容 概 略
1	ＵＮＩＸ入門	5 月 22 日(火) 13:00-16:10	江川　隆輔	・UNIX システムの基本的な使い方 ・エディタの使い方 ・プログラムの実行方法
2	スーパーコンピュータと並列コンピュータの基本的な利用法	5 月 23 日(水) 13:30-16:00	小野　敏	・ｽｰﾊﾟｰｺﾝﾋﾟｭｰﾀ、並列ｺﾝﾋﾟｭｰﾀの紹介 ・見学 ・利用法
3	スーパーコンピュータと並列コンピュータの高速化技法の基礎（実習形式）	5 月 24 日(木) 10:00-17:00	ＮＥＣ	・実習によるプログラムの高速化を目的とした最適化および並列化の基礎
4	ＭＰＩプログラミング入門(実習形式)	5 月 25 日(金) 10:00-17:00	ＮＥＣ	・MPI による並列プログラミングの基礎 ・利用法
5	MATLAB入門	6 月 15 日(金) 13:00-17:00	陳　　国躍 (秋田県立大)	・MATLAB の基本的な使い方
6	ネットワークと セキュリティ入門	8 月 2 日(木) 13:30-16:00	水木　敬明	・ネットワークの基本的な仕組み ・ネットワークの危険性と安全対策
7	Ｇａｕｓｓｉａｎ入門	8 月 7 日(火) 13:00-17:00	岸本　直樹 (理学研究科)	・Gaussian の基本的な使い方
8	Ｍａｔｈｅｍａｔｉｃａ入門	9 月 4 日(火) 13:00-17:00	横井　渉央 (情報科学研究科)	・Mathematica の基本的な使い方
9	ＵＮＩＸ入門	9 月 10 日(月) 13:00-16:10	後藤　英昭	・UNIX システムの基本的な使い方 ・エディタの使い方 ・プログラムの実行方法
10	スーパーコンピュータと並列コンピュータの基本的な利用法	9 月 11 日(火) 13:30-16:00	大泉　健治	・ｽｰﾊﾟｰｺﾝﾋﾟｭｰﾀ、並列ｺﾝﾋﾟｭｰﾀの紹介 ・見学 ・利用法
11	スーパーコンピュータと並列コンピュータの高速化技法の基礎（実習形式）	9 月 12 日(水) 10:00-17:00	ＮＥＣ	・実習によるプログラムの高速化を目的とした最適化および並列化の基礎
12	ＭＰＩプログラミング入門(実習形式)	9 月 13 日(木) 10:00-17:00	ＮＥＣ	・MPI による並列プログラミングの基礎 ・利用法
13	Ｍａｒｃ入門	9 月 14 日(金) 13:00-17:00	内藤　英樹 (工学研究科)	・Marc の基本的な使い方

備考: プログラムは予定のものです。若干変更になる場合がありますのでお含みおきください。詳細は、大規模科学計算システムウェブサイト（http://www.ss.isc.tohoku.ac.jp/）の「講習会」でご確認ください。このページから受講申込みが可能です。受講資格は、センター利用有資格者（教員、技術系職員、学生等）となりますが、No.6 に限り事務職員（非常勤職員も含む）の受講も可能です。

問合せ先：共同利用支援係(795-3406,uketuke@isc.tohoku.ac.jp)

利用負担金割引制度の実施について

大規模科学計算システムを有効に活用していただくために、平成 11 年度より、利用額に応じて負担額を軽減する割引制度を実施してきました。平成 24 年度は以下のとおり実施しますのでご活用願います。

平成 24 年度の利用負担金割引制度の内容

1. 実施期間は平成 24 年 4 月 1 日から平成 25 年 3 月 31 日までです。
2. 実施の対象はスーパーコンピュータ、並列コンピュータの演算負担経費です。したがって、ファイル負担経費および出力負担経費は含まれません。
3. 支払責任者ごとに集計した累計利用額に応じて負担額はつぎのように減額されます。
4. 申請などは不要で、すべての支払責任者（利用者）が適用となります。

利 用 額	負 担 額
10 万円まで	利用額と同じ
10 万円を超え 100 万円まで	10 万円
100 万円を超え 500 万円まで	(100 万円を超える利用額の 1/2) ＋ 10 万円
500 万円を超え 1000 万円まで	(500 万円を超える利用額の 1/3) ＋ 210 万円
1000 万円を超え 2000 万円まで	(1000 万円を超える利用額の 1/4) ＋ 375 万円
2000 万円以上	(2000 万円を超える利用額の 1/5) ＋ 625 万円

請求書は 4 半期ごとに発行されますが、割引制度は 1 年間の利用額の累計に対して適用されます。
(請求額　＝　4 月からの利用額の累計に割引制度を適用した金額　－　請求済額)

負担金項目と負担額

区 分	項　目	負　担　額
演 算 負担経費	スーパー コンピュータ	バッチ処理　演算時間　1 秒につき 0.4 円
		会話型処理　演算時間　1 秒につき　2 円
	並 列 コンピュータ	バッチ処理　演算時間　1 秒につき 0.1 円
		会話型処理　演算時間　1 秒につき 0.2 円
ファイル負担経費		1MB・ 日につき 0.1 円
出 力 負 担 経 費		大判プリンタによるカラープリンタ用紙 1 枚につき 600 円

備考
1. 負担額算定の基礎となる測定数量に端数が出た場合は、切り上げる。
2. 並列コンピュータで並列処理した場合の演算時間は経過時間とする。

[大規模科学計算システム]

初めてプログラムするための基礎知識

—パソコンもスパコンも基礎は同じー

福田優子
大阪大学レーザーエネルギー学研究センター

１．はじめに

大学もしくは大学院で初めてパーソナルコンピュータ(以下、パソコン)、場合によってはスーパーコンピュータ（以下、スパコン）などを用いて計算しようという方を対象に、知っておいていただきたい基礎知識を説明します。「京」が 2011 年 6 月と 11 月に Top500 で世界一になりましたが、急激にパソコンや研究室のクラスタマシンなどが進展し、ほとんどの方が大規模なシステムを利用せず、パソコンで研究を進められ、正しい基礎を勉強されないままに大学を卒業していかれることを残念に感じています。せっかく大学にいるのだから、その間に最先端のスパコン(HPC: High Performance Computer)につながる知識も勉強してください。あくまで概念の記述に重点をおいていますので、最新の情報や詳細は、マニュアルや各センターなどの WEB やテキストなどを参照してください。それらを理解できる下地を作りたいというのが、この記事の目的です。

プロセッサのマルチコア化や分散メモリによる並列化は、世の中の流れになっています。手元のパソコンだけで当面はこと足りそうだとしても、最初に読んでいただければ参考になるテキストとしてホームページで公開しているものから抜粋し、加筆修正しました。元のテキスト[1]も機会があればご参照いただけたら幸いです。

私は、理工系の情報系以外の学生の方がスパコンを使ってシミュレーションをされるのをサポートしてきましたが、基礎的なことは学習したことがない、あるいは習ったことはあるけど忘れたなど、基礎的な概念がつかめていない方も多いです。ほんの 10 年前までは、パソコンがパワフルだったわけではないので、計算しようとされる初心者の皆さんには端末室に来て作業していただき、分からないことをその都度説明することができました。しかし、最近はみなさん研究室（ひょっとしたら自宅)からネットワーク経由で利用されるので、直接指導する機会が減っています。また、分からないことは研究室の先輩に教えてもらうというのが難しい方も増えているように感じています。たとえ大型の計算機を使う必要がなくても、気軽に質問していただきたいと思っていますが、質問するのも難しいですよね。講習会も活用していただくとよいのですが、都合があわなくて、受講する機会がないままに卒業される方も多いでしょう。スパコンや計算機システムは、どんどん変化しています。研究室のまわりの方が（たとえ教授の先生でも）最新の正しい情報をご存じとは限りません。気楽に各センターのシステム管理者や相談窓口にお問い合わせください。

スパコンと一口で言いますが、定義は時代とともに変わりますし、種類もいろいろあります。東北大学に導入されている NEC 製の SX というスパコンはベクトル並列型と呼ばれるもので、システムについて特殊なことを勉強しなくても、少し勉強して素直にプログラムを作ると簡単に性能を引き出すことができます。ベクトル化は簡単です。理解してプログラミングを行うと、パソコンなどのプログラミングの基礎にもなりますし、並列化への発展も可能です。ベクトル化と並列

化については、東北大学サイバーサイエンスセンターや大阪大学サイバーメディアセンターでも毎年何回か、講習会が開催されています。一度は勉強しておかれることを強くお薦めします。

Fortran の超初心者用のテキストの重要性も感じていましたが、摂南大学の田口先生が、ご自分の研究室の学生のために作られていた入門書の研究室独自の部分をはぶき、Fortran の初歩から説明したテキスト[2]を提供してくださいました。レーザー研のホームページ[3]でも公開していますし、次号以降の SENAC に投稿させていただく予定ですので、ぜひ一度目を通してください。

２．計算機はややこしい？　ー用語の説明ー

計算機は難しいという方と話をしていると、用語が混乱しているためと思われることがよくあります。メーカーによって同じものを違う用語で呼んだり、場合によっては同じ用語を異なる意味で用いるなど、たしかに分かりにくいと思われることが多々あります。みな、自分の用語がデファクトスタンダードだと思われているようですし、計算機の世界はどんどん変化していますのでそのようなものだと思わないと仕方ないと思います。ただ、この記事は、概念をつかんでいただくことを目的としていますので、この記事内の用語は以下のように統一いたします。

・計算機

スパコン、クラスタマシン、ワークステーション、パソコンの総称として用いています。ワークステーションとは OS が Linux や UNIX で複数の人で利用している計算機を示し、パソコンやワークステーションも含めて説明したい場合に計算機と呼ぶことにします。また、複数の計算機を並べて、ひとつのシステムとしたものをクラスタマシンと呼びます。

・CPU、プロセッサ、コア（中央処理装置）

計算機の心臓部分であり、演算をする装置のこと。プロセッサと呼ばれることもあります。チップの周波数をあげることで計算機の性能は向上してきましたが、近年では、CPU をたくさん並べることで性能を向上しようという動きが加速してきており、CPU の中のコアが 2 つあるのはデュアルコア、複数のコアをもつものはマルチコアと呼ばれます。

・メモリ（主記憶装置）

計算するときにデータやプログラムを記憶するところ。電源が落ちるとデータは消えてしまいますが、CPU と高速に通信できます。電源が切れても保存したいデータは、ディスクなどに保存します。

・ジョブ

計算機に処理させるひとかたまりの仕事のことを意味し、ここでは主にプログラムの実行のこと。

・デフォルト（既定値）

計算機に何かやりなさいとか、ここを利用しなさいなどと指示する際に、様々なオプションがありますが、明示的に指定しない時に、自動的に採択される値のこと。

・サイバーサイエンスセンター、阪大 CMC、SX

東北大学サイバーサイエンスセンター、大阪大学サイバーメディアセンター（CMC)、もしくはそのスパコンシステムのこと。SX は、サイバーサイエンスセンター、阪大 CMC に設置されているスパコンのこと。

３．パソコンやスパコンを利用する前に（基礎の基礎）

３．１　ハードディスクは壊れる！セーブは常識

ハードディスクは壊れるものと思っていて間違いありません。プログラムや大事なデータなど消えたら困るものは必ずセーブするようにしましょう。セーブというと、USB や DVD に保存するなど他のメディアに保存することと思われるかもしれませんが、他のコンピュータにコピーしておき、物理的に 2 箇所以上においておくこともセーブになります。地理的に離れた箇所に保存するのがよいかもしれません。自分の財産は自分で守る、危機管理意識を常に持ちましょう。パソコンのハードディスクが壊れることはよくあります。

３．２　研究を始める前に自分の身体は自分で守る

研究を始め、画面に向かって仕事をするようになると夢中になって時間を忘れるかもしれません。ゲームやインターネットでも同じです。当然のことながら、長時間画面に向かって作業すると眼などに悪い影響があります。自分の身体や眼は自分で守るしかありません。VDT 作業指針というものがあり、イスの高さや画面の高さ、適度に休憩をとるなどが紹介されています。WEB で「VDT 症候群」「VDT　作業　対策」などで調べると、たくさんヒットしますので一度は目を通しておき、1 時間画面に向かって作業したら、10 分は眼を休めるなど、自分で工夫して、自分で自分の身体を守ってください。長年、講習会では、頭は休めなくていいですよと話をしてきましたが、最近は、頭も適当に休ませてあげないといけないと思っています。

３．３　プログラムを作る

パソコンやスパコンに仕事をさせるためには、通常プログラムを作成します。商用のプログラムやソフトを利用する場合もありますし、エクセルで大抵のことを済ます方もいますが、ここでは基本的に自分でプログラミングすることを前提にしています。先輩から引き継いだものなど既存のプログラムを利用する場合も多いでしょうが、中身を理解するようにしましょう。シミュレーションの条件を変える、測定する物理量を追加するなどプログラムの修正、追加が必要な場合にもあてはまります。

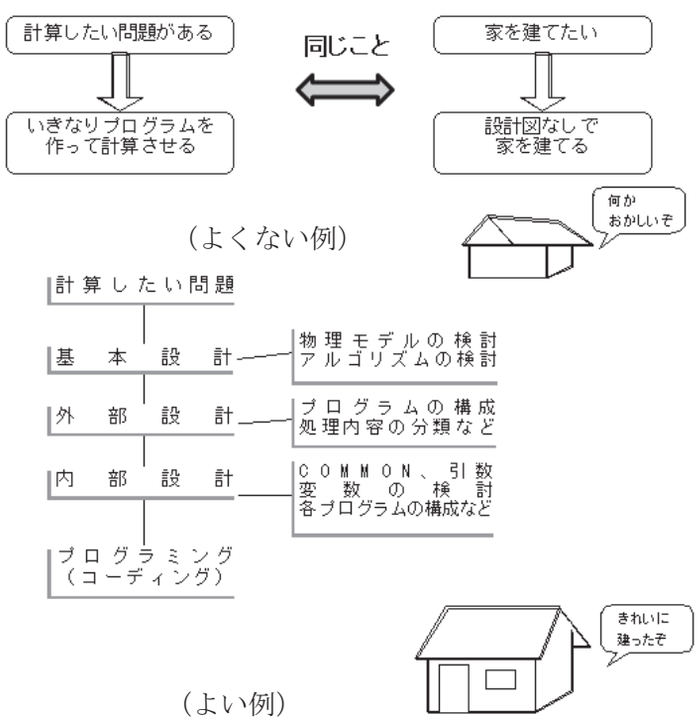


図１　プログラム作成の概念図

図1に、よくない例としてあげているように、思いつくままにプログラムを書くということは、設計図なしに家を建てるのと同じことです。あらかじめ考えるべきことは紙に書き出し、簡単なフローチャート（流れ図）などを作っておくと、プログラミング作業は楽になり、エラーも減ります。ドキュメントはあとで作ろうと思う方も多いと思いますが、画面に向かうときには、ドキュメントは完成していて、ただタイピングするだけという状態が理想です。

３．４　誰が見ても分かりやすいプログラムを作りましょう

プログラミングには、それぞれ人の流儀がありますが、よいプログラムとは誰が見ても分かりやすいプログラムです。計算機は年々速くなりますし、少々回り道をさせても文句も言わず、言われたとおり（これが問題のときもあるのですが）計算してくれます。実際に計算機を使用した際に最も効率が悪いのは人間です。

・コメントをたくさんつける

コメントとは、プログラムの実行は行なわれない注釈のことで、覚えのために書いておくメモのことです。特に単位はよく間違えますので、m（メートル）なのかcm（センチメートル）なのか、g（グラム）なのかkg（キログラム）なのかなどを、プログラム中にコメントで書くようにしましょう。修正箇所には日付とともに簡単な概要を書くようにしましょう。

```
! コメントをたくさん書きましょう
!
!  初期値設定
!
      a=0.1d0    ! cm/sec
!
!  収束条件変更 （2011.7.8  by Y.O.F)
     :
     :
```

コメント例）！以降、行末までコメント

・ルールに従った変数名やファイル名をつける

レーザー研の西原研究室では、伝統的にローカルな変数は「Z」で始めるというルールがありました。このルールに従うことで、変数を見ただけで、他では使われていないということが分かります。ファイル名、ディレクトリ名なども一時的なものは「z」で始めるというルールに従うことで、あとで安心して消すことができます。

核融合科学研究所の坂上先生は、ローカル変数に加えて、仮引数、COMMON 変数や、変数の型（REAL*4, REAL*8, INTEGER, CHARACTER）などでも区別するルールを作り、分かりやすくされているそうです。参考にしてください。

３．５　自分のプログラムのメモリ容量に注意しましょう

計算機に計算させようという場合には速さが気になりますが、いわゆる計算機の心臓であるCPUの他に、メモリをどれだけ使うかということも重要です。計算のための変数、配列の他に、計算機に対する命令などもすべてメモリ上に展開されます。計算機を使って計算しようとする方は、メモリ容量にも注意を払うようにしてください。2Gバイトのメモリを搭載しているパソコンで5Gバイトのメモリを必要とする計算をしようとしたら、動かない、もしくは動いてもとても遅いということになります。

一般的には以下のような方法で、プログラムが実行に必要とするメモリ容量を知ることができますが、これはファイルのサイズとは異なりますので、注意してください。

- Linux,UNIX 標準の size コマンドを用いる
  (SX の場合は sxsize コマンド)
- 大規模なプログラムの場合は概算できる場合が多い。自分で大規模な配列サイズを計算する。例：変数の数×配列数×8 バイト＋‥‥

実際には、処理系が勝手にとる一時メモリも無視できない場合がありますし、配列の動的割付けという機能を用いた場合も size コマンドではわかりません。以下のような方法で調べることができる場合もあります。

- Linux の top コマンド
- プログラム実行中の情報、あるいは終了後に出力される情報を参照する（SX の場合は実行終了時の program information に詳細が表示されます)

３．６　メモリとスワップとキャッシュ

パソコンでいろいろ仕事をさせている場合も同様ですが、多数の仕事を同時に計算機にさせると、メモリに入りきらない分は、メモリよりもはるかにアクセス速度の遅いディスク装置などに追い出されます。この状態をスワップと呼びます(図 2)。スワップが発生するとシステムの効率は非常に悪くなります。前節で 2G バイトのメモリを搭載しているパソコンで 5G バイトのメモリを必要とする計算をしようとしたら、とても遅い場合があると書きました。このような場合は、メモリに入りきらない分をディスクに入れたり出したりしながら計算しようとしますので遅くなります。メモリを増設したらパソコンが速くなったという経験をお持ちの方も多いのではないでしょうか。

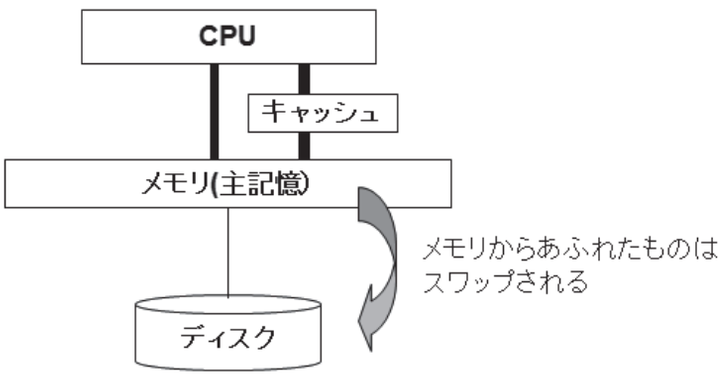


図 2　CPU とメモリとスワップの概念図

さらに、キャッシュという言葉も聞いたことがありますか。CPU で高速に計算するためには、メモリからいかに高速にデータを供給するかが重要ですので、CPU とメモリの間のデータ転送の遅延を隠ぺいするために用いられます。パソコンなどでは特に、ここを効率よく利用することも高速化では重要です。計算速度（＝計算時間）にだけまどわされないように注意してください。

４．プログラミングするときに気をつけてほしいこと

４．１　メモリのアクセスは連続に

3 章の最後で説明したように、メモリやキャッシュを有効に利用することはプログラミングの基本ですが、実際にプログラムを作るときには、極力メモリに連続アクセスするように心がけてください。そうすれば、難しいことは考えなくても有効利用できます。ここでは Fortran の説明はしませんので、Fortran がわからないという方は、詳細は田口先生のテキスト[2]などを勉強してください。プログラムのイメージを紹介します。Fortran では、1 次元配列は a(100)のように宣言すると、メモリ上には図 3 上のように配置されます。プログラムでは図 3 下のように書くと、a に連続的にアクセスすることになります。

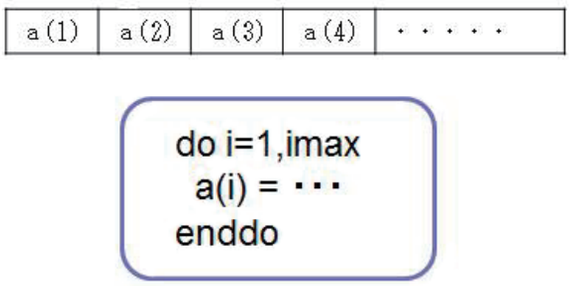


図 3　1 次元配列のメモリ上の配置（上）とプログラム（下）

a(100, 50)のような 2 次元以上の配列でも、メモリ上では 2 次元ではなく、1 次元に並んでいます。2 次元以上の配列の場合には、図 4 の左のプログラムのように、配列の 1 次元目を内側のループにすると、メモリに連続的にアクセスすることになります。しかし、右のように配列の 2 次元目を内側のループに書くと、100 個おきに不連続にアクセスすることになりますので、配列の 1 次元目を内側ループで回すようにしましょう。C 言語の場合は、Fortran と異なりますので注意してください。

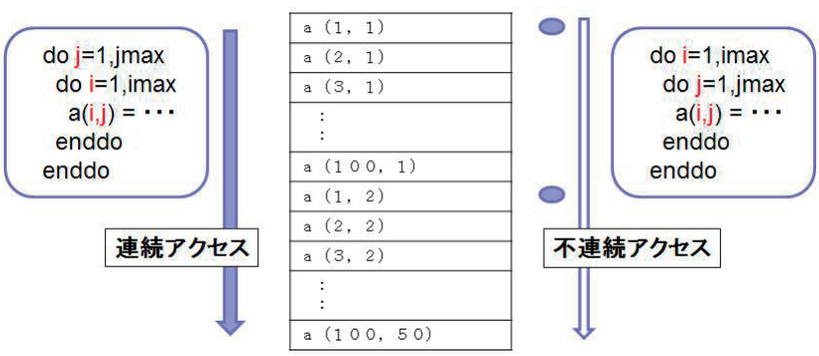


図 4　Fortran　2 次元配列のメモリ上の配置

４．２　ループ長は長くなるように

マルチコアプロセッサが時代の流れとなり、パソコンでも 4 コア搭載などが当たり前になってきました。ベクトル化という概念はスパコンだけでなく、パソコンでプログラムする際にも知っておいた方がよい知識です。パソコンのコンパイラでも、「ベクトル化できました」のようにメッセージを表示する場合があります。4.1 に記したようにメモリに連続にアクセスするだけでなく、ループ長が長くなるように気をつけることも重要です。

たとえば(3, 3, 10000)のような配列を宣言したほうが、物理的にピッタリくる場合でも、(10000, 3, 3)のように配列を宣言し、1 次元目の 10000 でループを回すようにしてください。そう

することで、メモリに連続にアクセスしますし、ループ長も長くなり効率のよいプログラムとなります。
（注意：ベクトルマシンは上記のことがあてはまりますが、スカラーマシンの場合は、キャッシュメモリにデータがのるかどうかの影響が大きいので、一概には言えない場合もあります[1]。）

４．３　バンクコンフリクトは避けましょう

メモリは、バンクと呼ばれる幾つかのグループに分かれており、異なるバンク間を並列にアクセスできるようになっていますが、同じバンクへのアクセスが集中するとメモリアクセスの性能が低下します。バンクの数は、機種によって異なりますが、2 のべき乗の場合が多いので、一般的には 2 のべき乗の間隔でのアクセスは避けた方がよいでしょう。具体的には、2 次元目以降で a という配列にアクセスする場合は、右のように 1 次元目の配列の宣言を奇数にします。1 次元目でアクセスする場合は、連続アクセスになりますので、このようなことは不要です。

```
real a(1024,1000)
do i=1,1000
  a(1,i) = ・・・
enddo
```

↓

```
real a(1025,1000)
do i=1,1000
  a(1,i) = ・・・
enddo
```

４．４　入出力は効率的に

入出力（read、write、print など）は効率が悪いものです。はじめのうちは、分かりやすいように、いたるところに write 文を挿入してもいいと思いますが、本格的に計算するようになったら、注意してください。

図 5 左のようにループの一番内側に write 文を書くと 15 回 write 文を実行し、15 レコード出力されますが、右のように書けば、1 回の実行で 1 レコードで出力されます。なるべくこのように書くほうが効率はよくなります。

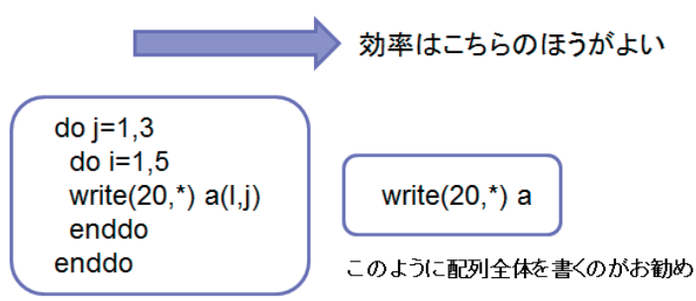


図 5　write 文の形式と効率

さらに、本格的に計算するようになると、計算の重いループの中に入出力文を入れるのも効率が悪くなります。配列を宣言してその中に格納し、上のように、まとめて出力するようにしましょう。配列（変数）に格納するということは、メモリ上にデータを保存するということであり、メモリ容量がその分必要になります。しかし、最近はメモリ容量も大きくなりましたので、入出力の効率化を意識したほうがより実効性能があがります。

５．プログラム実行の概念

プログラムができあがり、計算機で計算させるためには機械語に翻訳する必要があります。プログラム言語は人間に分かる言語になっており、標準的な Fortran の仕様で書いていれば機種依存はありません。パソコンでもスパコンでも実行させることができます。という意味で、なるべ

く標準的な仕様でプログラミングするほうがいいでしょう。機械語に翻訳するためには、コンパイラを利用しますが、これは機種によって対応するものが異なります。パソコン用の機械語は、スパコンでは動かないことは言うまでもありません。機種だけでなく、コンパイラのバージョンや、32 ビット用か 64 ビット用かなども注意が必要です。自分が実行しようとしている計算機のハードウェアとソフトウェアの概要は知っておいてください。

５．１　コンパイル

図 6 はプログラムを作成してから、計算機で実行させるまでの翻訳する作業（**コンパイル**）の概念を示しています。機械語に翻訳されたものをオブジェクトモジュールと呼びます。これでもまだ実行はできません。**リンク**（結合）という作業をすると実行形式である**ロードモジュール**（LM、実行オブジェクトファイル、実行形式、実行ファイル、パソコンでは exe（エグゼ）などと呼ばれることもあります）が作られます。計算機はこのロードモジュールを実行することができます。通常コンパイルとリンクの作業をあわせてコンパイルと呼び、そのためのツールをコンパイラと呼びます。

図 6 に示すように、メインプログラム、サブルーチンなどのプログラム単位にソースプログラムを保存したときは、make というツールを使うと便利です。コンパイルすると「ファイル名.o」という名前でオブジェクトモジュールができますが、それとの関連もわかり易くなります。

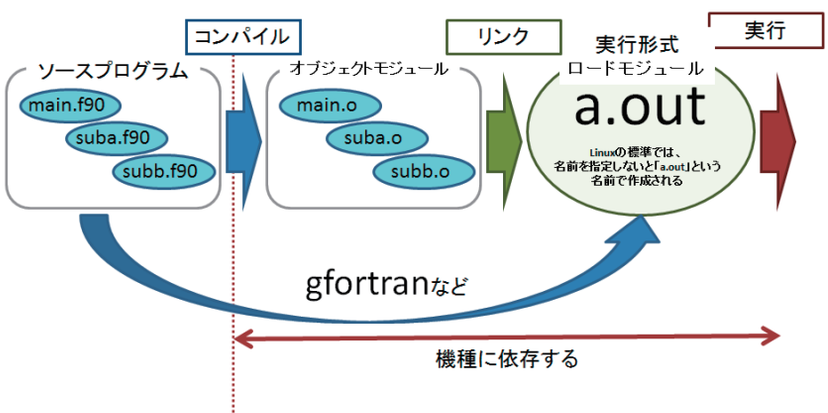


図 6　コンパイルの概念

例えば source.f90 というプログラムを作成し、Linux 上で gfortran source.f90 のように実行するとコンパイルとリンクが行われ、a.out という名前のロードモジュールができます。オブジェクトモジュール（source.o）は、明示的に残すという指定をしないと残らない場合もあります。コンパイルしたのに、a.out ができないという場合は、何かエラーが発生しています。原因をきちんと調べて対処してください。

よく使う関数など汎用的なものはライブラリとして保存し、リンクするだけで再利用するというようなことも行われています。個人的にオブジェクトの形でライブラリ（例：libabc.a など）として保存して利用されている方も多いですし、科学技術計算のための複雑な関数などは、市販されているものや、フリーのものもいろいろあります。

５．２　デバッグしましょう

できたプログラムは、正しく計算できているかどうか必ずチェックしましょう。理論計算で解が分かっている問題を計算させ、プログラムが正しいかどうかチェックすべきです。プログラムを修正したときのために、チェックする仕組みを入れておくべきでしょう。また、部分ごとに思ったとおりの答えをだしているかどうかチェックしてから全体をチェックしてください。いきなり全部動かして、どこでエラーがおこっているか分からないというようなことは避けてください。それらしく、計算機が答えをだしてくると「正しい」と思いがちですので、注意するようにしましょう。

５．３　ＣＰＵ時間とエラプス時間

計算機で計算させたときの時間には、CPU 時間とエラプス（Elapse）時間と呼ばれるものがあります。CPU 時間は実際に CPU が演算処理した時間のことです。エラプス時間は経過時間とも呼ばれ、計算機が処理を開始してから終了するまでの時間です。特に並列化したプログラムの場合は、このエラプス時間が重要になります。複数の人間で利用するような計算機では、「ひとつの仕事が終わってから、次の仕事を実行する」というようなやり方ではなく、「投入された複数の仕事を少しずつ実行する」ことにより処理します。このようにして、同時にいろいろな処理がすすんでいるように見えます。これをタイムシェアリングと呼びます。もともとは大型計算機で開発された手法ですが、現在ではパソコンでもこのような手法が用いられています。そのため、計算時間を考えるときに、この両方の時間に注意を払う必要があります。他に何も計算していないはずなのに、CPU 時間とエラプス時間に大きな違いがある場合は、入出力の効率が悪いとか、メモリが足りなくてスワップが発生しているなどが考えられます。

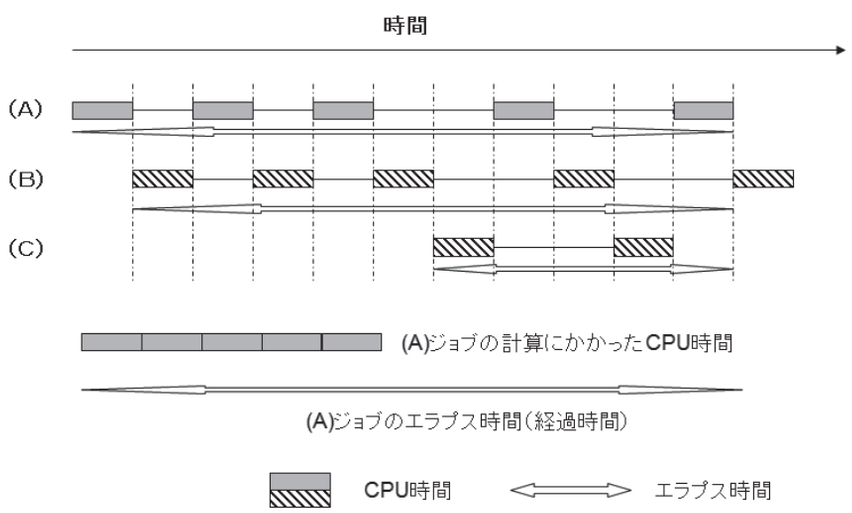


図７　CPU 時間とエラプス時間

ここではひとつの CPU で 3 個のジョブを実行する様子を使って CPU 時間とエラプス時間について説明します。図 7 は、(A)、(B)、(C) の 3 個のジョブが実行されているときの計算機の様子を示しています。それぞれボックスの部分が実際に CPU を使って計算した時間を示し、その合計がそのジョブの CPU 時間となります。エラプス時間は矢印で示される開始から終了までの時間を示します。複数のジョブが同時に実行されているために、ひとつひとつのジョブの CPU 時間は短くても、エラプス時間が長

くなっていることが分かります。同時に多数のジョブを実行させるほど、一個ずつのジョブの終了までにかかるエラプス時間は長くなります。

６．スパコンの動向とプログラミング（ベクトル化＆並列化の概念）

６．１　スパコンとは

「スパコン」の定義は時代とともに変化します。あまりにも多様化したので、「高性能コンピュータ（High Performance Computer)」と呼ばれるようにもなりました。

・その時代で最も高速な処理能力をもつコンピュータの総称
・主に科学・技術分野に利用される
・スーパーコンピュータセンターに設置されているコンピュータ

などと思うとイメージがわくかもしれません。

プロセッサの単体性能は、限界に達してきていますので、マルチコア＆超並列が時代の流れと言わざるをえません。2011 年 6 月と 11 月に 2 期連続で世界一となった「京」は、864 筐体（CPU 数 88128 個）の構成で、LINPACK（リンパック）と呼ばれるベンチマークにおいて、世界最高性能の 10.51 ペタフロップス（毎秒 1 京 510 兆回の浮動小数点演算数）を達成しました。しかし、このような超並列の計算機を研究で使いこなすためには、並列化プログラミングが必要であり、相当の技術力を必要とします。ここではそのイメージとプログラミングについて紹介しますが、

「ベクトル化」➔「(自動並列化)」➔「分散メモリ並列化」

を考えるのが基本です。

（注意：パソコンでもベクトル化を活用すべき方向になってきていますが、キャッシュの影響の方が大きいので、パソコンの場合は、「まずベクトル化」はちょっと言い過ぎかもしれません。しかし、今後の計算機の動向はそういう方向に進むように思いますし、ベクトル化は基本的にシンプルにプログラミングすればよいので、作業の効率を考えてまず「ベクトル化」をお勧めします。そうしておけば、パソコンでは時間がかかって困るようになったときに、サイバーサイエンスセンターや阪大 CMC のスパコンを使えば、簡単に高速化が期待できます。）

６．２　ベクトル化とは

ベクトル化は、繰り返し処理される配列データを、一括で演算させることです。SIMD 演算とプログラミングの考え方は基本的に同じですし、ベクトル化は、ループの演算で性能を発揮します。イメージは、図 8 に示すようなもので、配列データを一括して演算し、計算を高速化するものですが、コンパイルによりベクトル化前とは計算順序が異なることもあり、プログラミング上注意すべきことがあります。

ベクトル化できるようにプログラムするのが基本と思って間違いありません。ベクトル化については、一度は講習会などで勉強されることを強くお勧めします。

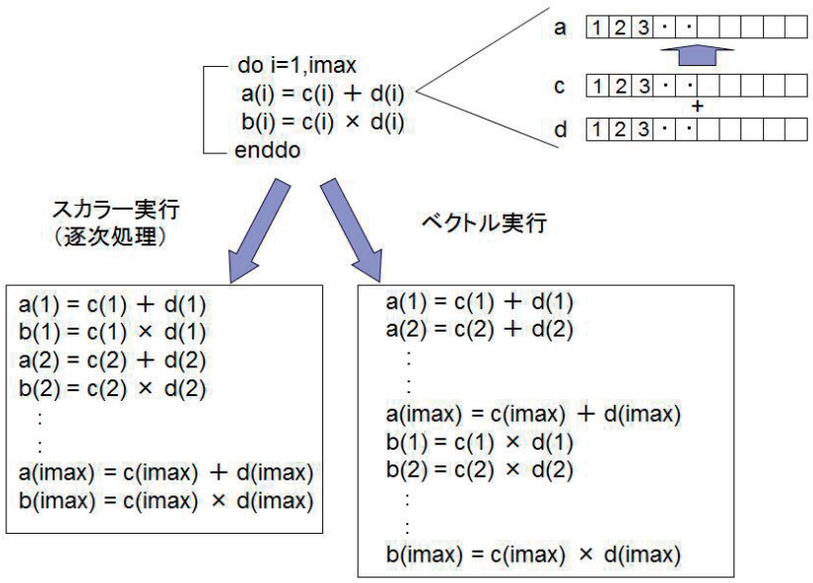


図 8　ベクトル化のイメージ

６．３　並列化とは

図 9 に、並列実行した場合の CPU 時間とエラプス時間のイメージを示します。並列化の目的は、CPU 時間を短くすることではなく、エラプス時間を短くしようとするものです。ある計算を 4 並列で実行したときは、理想的には図 9 に示すようにエラプス時間が 1/4 になるはずです。

並列化を考えるときには、メモリが共有か分散かでプログラミングが大きく変わります。メモリが共有ということは、どの CPU からも同じメモリが見えているので、変数のアクセスが簡単ですが、分散メモリのときは CPU から見えている変数がどのメモリにあるかを意識してプログラミングしないといけないので、いろいろ気をつけないといけません。

図 10 に、CPU と主記憶（メモリ）とノードのイメージを示しています。一つの CPU を使って計算する場合はシングル実行と呼びます。この場合は、ベクトル化でスピードをかせぎます。ノード内の複数の CPU を使って並列計算するときには、メモリが共通ですので、自動並列や OpenMP などで並列化できます。

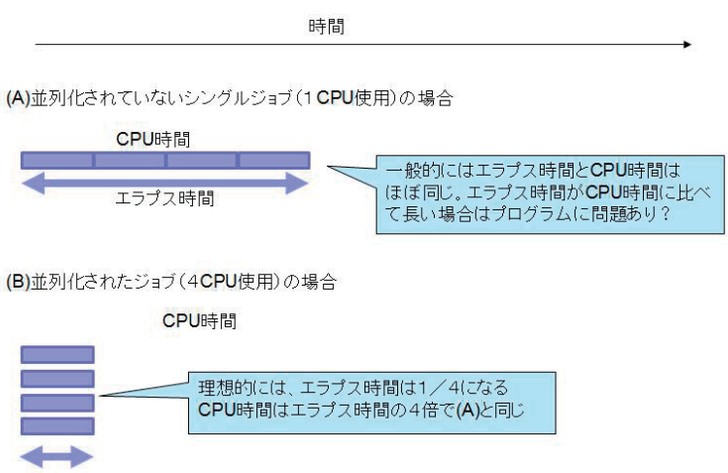


図 9　並列実行の CPU 時間とエラプス時間

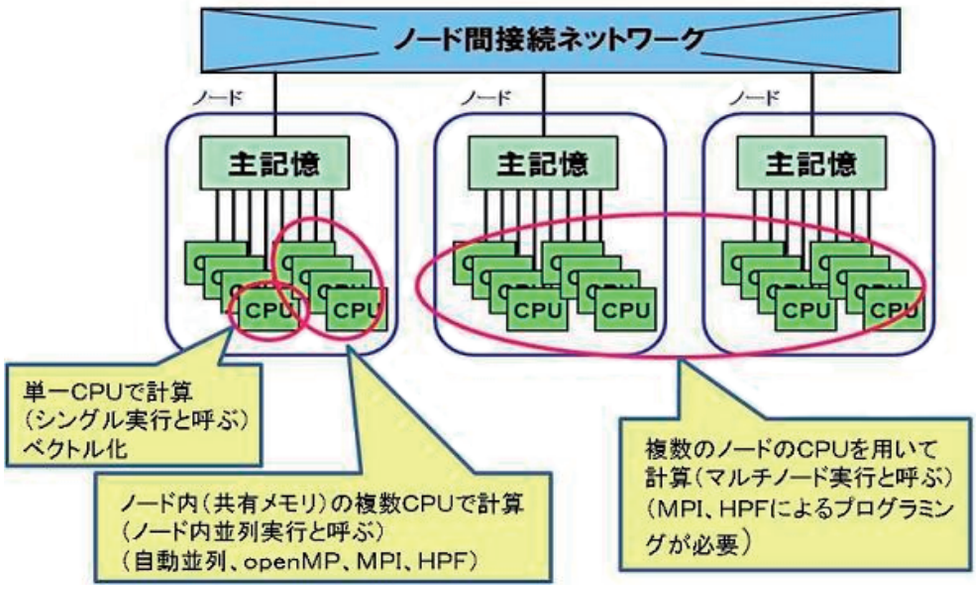


図 10　共有メモリと分散メモリ並列のプログラム方法

複数のノードを利用する、分散メモリに対応した並列化プログラムを作成するためには、現在のところ MPI(Message Passing Interface)と呼ばれるメッセージ通信のためのライブラリを用いるのが主流です。最初から並列プログラムとしてプログラムを記述する必要があり、初心者にはかなりハードルは高いと言わざるをえません。HPF*と呼ばれる言語もあります[4][5][6]ので、興味のある方は参考文献を参照してください。(* サイバーサイエンスセンターではサービスしておりません。)

また、ノード内は自動並列や OpenMP で並列化し、ノード間は MPI で並列するような、ハイブリッド並列と呼ばれる方法もあります。

７．スパコンセンターなどを利用するための概念と基礎知識

ここからは、サイバーサイエンスセンターや、阪大 CMC のスパコンのように、多数の利用者で共有する大規模なシステムを利用するための基礎知識を説明します。パソコンで利用するための基礎知識があれば、それほど新しい知識が必要なわけではありません。詳細は、各センターの説明を参照し、不明点はそれぞれのセンターにお問い合わせください。利用者からの質問は、システム管理者にとっても勉強になりますので、遠慮はいりませんよ。

スパコンの OS は UNIX をベースとしています。UNIX が分からないから使えないという声もききますが、スパコンを利用するだけなら、必要なコマンドはそれほど多くはありません。UNIX、Linux はいろいろ種類がありますが、利用者にとって基本的な使い方は同じですし、一度覚えるととても簡単で便利です。東北大学では UNIX 入門という講習会も開催されていますので、一度勉強されるとスパコンセンターの敷居はうんと低くなります。レーザー研のテキスト[1]でもすこし紹介していますので、参考にしていただけたら幸いです。

７．１　利用の流れ

スパコンに仕事をさせるための、通常の作業手順は図 13 のようになります。ここでフロント端末と呼んでいるのは、並列コンピュータを指します。スパコンの作業用端末として利用します。

データの解析をどこで、何を使って行うのがよいかは一概には決められませんが、大容量のファイルを解析したり、転送したりしたい方は、事前にシステム管理者と相談するようにしてください。データ量に応じてふさわしい使い方をしないと、システムに負荷をかけ他の人に迷惑をかけることがあります。

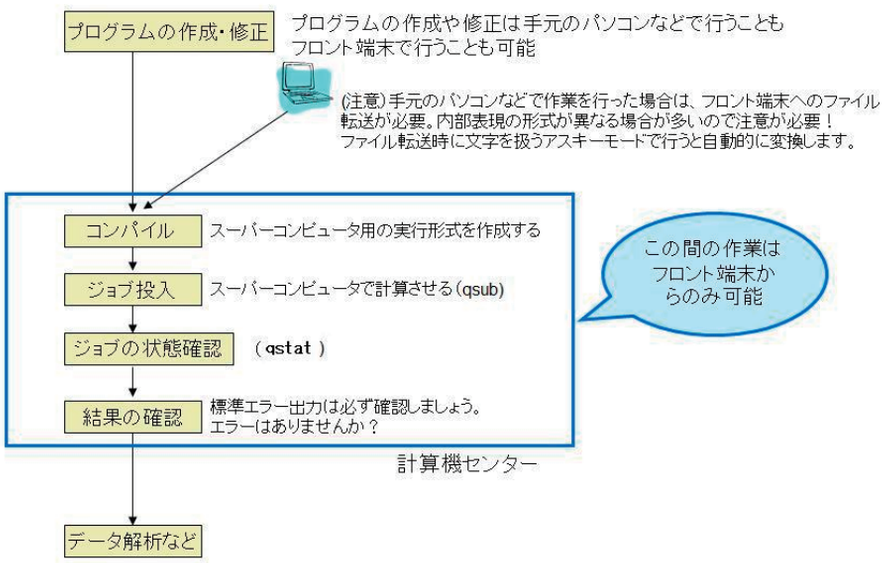


図 13　スーパーコンピュータ利用の流れ

７．２　リモートログインとファイル転送

7.1 のような利用の流れですが、今やセンターの端末室に行って利用するということはほとんどなく、手元のパソコンからリモートログインして遠隔地のセンターのシステムを利用するのが一般的です。そのために、手元のパソコンに、リモートログインとファイル転送するための環境を用意する必要があります。一般的には、リモートログインには、「SSH クライアント」ソフト、ファイル転送には、「sftp」や「scp」を利用します。それをキーワードにご自分のパソコンにあう適当なソフトをインストールしてください。まわりの方に聞いてみるのもよいかもしれません。

レーザー研では、Windows では「Tera Term」と「Winscp」を利用している人が多いようです。Mac OS X はデフォルトで、「ssh」「sftp」が搭載されていますので、terminal を起動して、以下のように入力すればサイバーサイエンスセンターにログインすることができます。

```
ssh username@gen.isc.tohoku.ac.jp
```

Windows に Cygwin をインストールして利用されている方も多いですが、初心者にはインストールにちょっと手間がかかるかもしれません。

７．３　セルフ環境とクロス環境

コンパイラは、計算機の機種に依存し、Linux 端末などでは、5.1 で説明したコンパイルと実行は、通常同一の計算機上で行います。このような環境を**セルフ環境**と呼びます。一方、スパコンはロードモジュールを高速に実行するには適していますが、コンパイルという作業には不向きです。そのため**クロス環境**と呼ばれる環境を利用します。

セルフ環境：コンパイルと実行までを同じ計算機で行うこと。セルフ環境のコンパイラをセルフコンパイラと呼ぶ。

クロス環境：コンパイルと実行を異なる計算機で行うこと。クロス環境のコンパイラをクロスコンパイラと呼ぶ。

サイバーサイエンスセンターではフロントと呼ばれる端末（並列コンピュータ）が SX のクロス環境のための Linux 端末です。SX 用のロードモジュールを作るためのクロスコンパイラは **sxf90** というコマンドを利用します。図 14 にセルフコンパイラとクロスコンパイラの違いを示しています。

無事にコンパイルが終わったら、ベクトル化、並列化、最適化などコンパイラが何をしているかを確認するようにしましょう。**sxf90** でコンパイルした場合は、-R5 オプションをつけると、編集リストが出力されコンパイラが自分のプログラムをどう変形したのかを調べることができます。

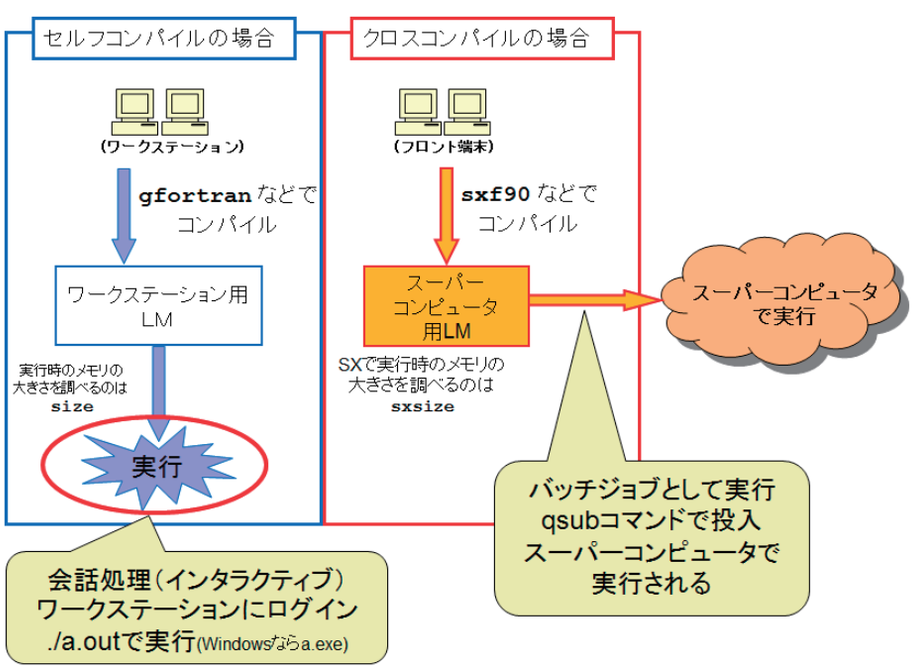


図 14　セルフコンパイラとクロスコンパイラ

また、実際に実行させなくても、図 14 で示すように size コマンドや sxsize コマンドを利用することにより、実行させたときに、どのくらいのメモリ容量を必要とするかの目安を調べることができます。

７．４　スパコンで計算させるにはバッチ処理

プログラムを計算機で実行させる（計算させる）方法には、以下の 2 通りの方法があります。

・　会話処理（インタラクティブや対話処理とも呼ばれる）
通常のパソコンやワークステーションの作業。利用者と計算機が、会話をするように入出力を繰り返す処理方法で、デバッグや短時間の計算に適している。

・　バッチ処理（一括処理）

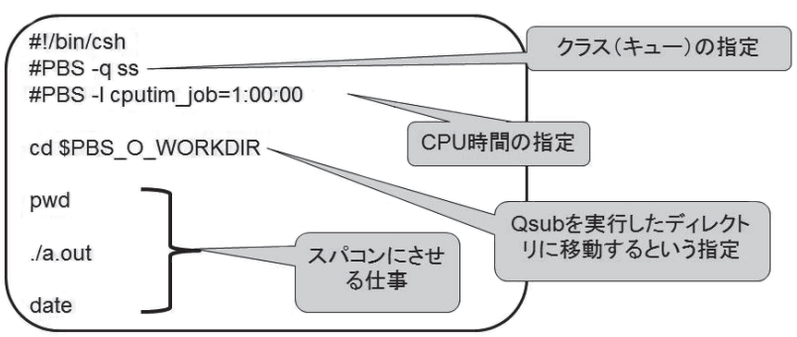
イメージは図 14 に、特徴は表 1 に示します。7.3 のクロスコンパイルにより、できたロードモジュールをスパコンで実行させるためには、バッチ処理（NQS2 など）を用います。

表 1　会話処理とバッチ処理

	会話処理	バッチ処理
実行のさせ方	計算機に向かってコマンドを入力	コマンドなどを書いて、システムにわたす
いつ実行されるか	コマンドを入力するとすぐに実行される	計算機システムが状況に応じて実行する
端末の画面	占有され、ログインしたままにしておく	計算機システムにまかせておけばよいので、ログアウトして帰ってもよい
計算機全体の効率	考慮できない	考慮できる

「どのクラスで計算させるのか」
「どのディレクトリにある、どのロードモジュールを実行させるのか」
「どのディレクトリにデータを吐き出すのか」
「計算を実行せよ」

などスパコンにさせたい仕事の命令を記述しておくと、スパコンが自分の都合に合わせて実行します(スケジューリング)。このひとかたまりの計算をジョブと呼びます。ここでは、スパコンにさせる命令を記述したものを NQS ファイルと呼びます。以下に、簡単な NQS ファイルの例をのせます。このファイルは、それぞれのセンターの指示に従って作成してください。



考え方は、システムにより異なりますので、それぞれのセンターの説明を参照して、自分のプログラムに合わせて適切に指定するようにしてください。

7．5　いよいよスパコンで計算させましょう（ジョブ実行）

ロードモジュール、NQS ファイル、入力データなどが用意できたら、いよいよスパコンに計算させます。図 15 は NQS を用いたジョブ実行の概念を示します。

①は､用意した NQS ファイルを qsub コマンドを使ってフロント端末からスパコンに投入します。これをジョブの投入と言います。②NQS サーバーはジョブを受け取ると各々のジョブにリクエス

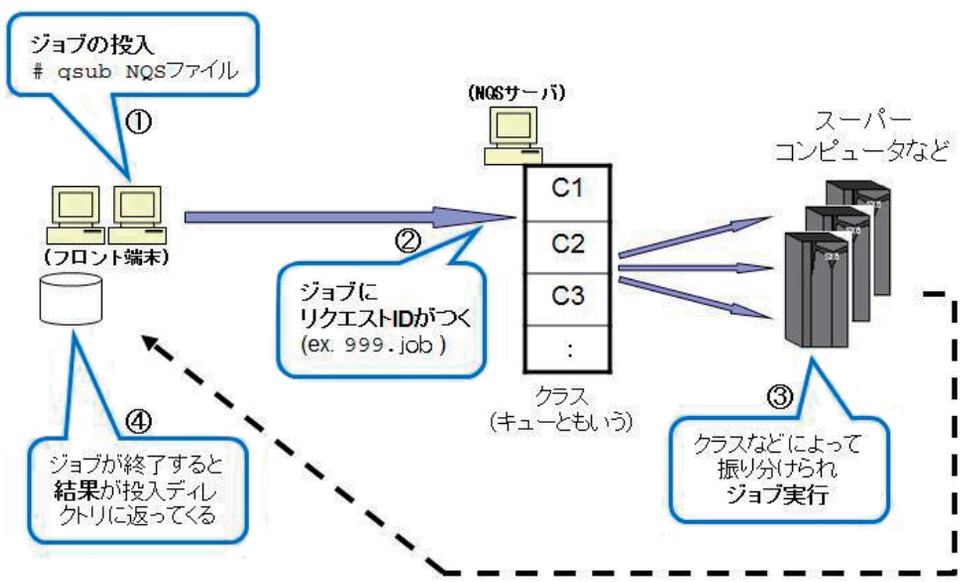


図 15　NQS を用いたジョブ実行の概念

ト id と呼ばれる番号をつけます。この例では、999.job がリクエスト id（ジョブ番号と呼ぶこともある）であり、ジョブの状態表示や、強制終了、結果の確認などに利用します。③で、スパコンがジョブを実行し、④で結果を投入ホストに返します。これらの作業のためにスパコンに入る（ログインする）必要はありません。表 2 のコマンドをフロント端末から入力するだけでスパコンに仕事をさせることができるのです。

表 2　よく使う NQS コマンドの例

ジョブの投入	qsub　NQS ファイル
ジョブの状態表示	qstat
ジョブの強制終了	qdel　リクエスト ID

④では特に指定しない限り、標準出力と標準エラー出力と呼ばれる 2 つのファイルが NQS ファイルを投入したディレクトリに返ってきます。

標準出力は、プログラム中で write (6,*) のように書かせたものです。思ったようにプログラムが動いているか、確認するための小容量の出力に利用してください。基本的には画面に出力させるイメージですので、ここに大量に出力すると、システムによっては全て出力できない場合があります。大容量の出力は別ファイルにするようにしてください。標準エラー出力には、ゼロ割りや、計算機の内部表現で表せる大きさ以上の数値になってしまったなどのエラーが記録されますので、ジョブ終了後、結果が返ってきたら必ず確認するようにしましょう。SX では、どのくらいの CPU 時間がかかった、メモリを利用したなどの実行時の詳細情報もここに出力されますので、確認するようにしましょう。

７．６　ディスク構成を知り、正しく工夫してディスクを利用しましょう

スパコンのような大規模なシステムでは、多人数で共有するのですから、自分のプログラムから出力される予定のデータ容量を把握し、正しくディスクを使うようにしましょう。大容量のファイルを必要とするのに、分からないという方はシステム管理者に問い合わせるようにしてください。

せっかくシミュレーションしたのだから、すべてのデータをいつまでも保存したいと思われるでしょうが、スパコンのディスクに、いつまでも皆さんの大容量のデータを保存することはできません。データをどのように解析し、可視化し、保存する必要があるのかをよく考える必要があります。時代とともに状況は変化しますが、現在は以下のような方法が考えられます。スパコンで計算させるのは意外に簡単なのですが、あとのデータ処理が実は難しい場合があります。どのような方法をとるとしても、プログラムや NQS ファイルなど、実行するための環境を保存しておくということは重要なことです。

- 必要になったときに再実行できるように、プログラム、ロードモジュール、入力データファイル、NQS ファイルなどを保存しておき、膨大な生データは保存しない。
- 手元のパソコンにファイル転送して保存しておく。
- 生データは膨大なので、可視化して小さくなった画像データのみを保存しておく。
- 科学技術計算は倍精度（実数型の場合、有効桁数は 10 進で約 16 けた）で行う必要があるが、単精度または工夫して精度を落として保存することにより、ファイルの容量を小さくする。(倍精度のまますべてを保存する必要があるかよく考えましょう)
- 計算領域すべてを保存するのではなく、注目している現象の部分のみを保存する。
- 時間発展による変化を観測したい場合、変化のみを保存する。
- 現象に応じた圧縮方式を用いる。ここでいう圧縮は compress や gzip コマンドによる圧縮ではありません、プログラムなどによる工夫を意味しています。間違ってもスパコンに compress や gzip の仕事をさせないでください。

などです。もっといい方法をご存知の方はぜひ教えてください。

それぞれのシステムで若干異なりますが、スパコンで計算させるために必要な概念は終了です。イメージはつかめましたでしょうか？

８．よくある質問

今までに質問の多かった項目です。より詳しい説明は田口先生のテキスト[2]にも掲載されていますし、WEB などで調べることもできます。

８．１　計算機の内部表現

スパコンといえども、数値を表すのはビット（1 か 0）が基本です。1 バイトは 8 ビットで、単精度（シングル）の数値は 4 バイト（32 ビット）、倍精度（ダブル）の数値は 8 バイト（64 ビット）で表現されます。

計算機内部で 1 と 0 をどう組み合わせて、どう数値を表現するかを内部表現と呼びます。図 16 は SX の float0 形式の整数と実数（単精度と倍精度）を説明したものです。単精度と倍精度では有効けた数や表現範囲が異なり、科学技術計算は倍精度で行うのが基本です。単精度では有効けた数が約 7 けたで 10^{-38} から 10^{37} までの範囲の数字が表せますが、倍精度では約 16 けたの有効けた数で、10^{-308} から 10^{308} の範囲の数字を表すことができることを示しています。

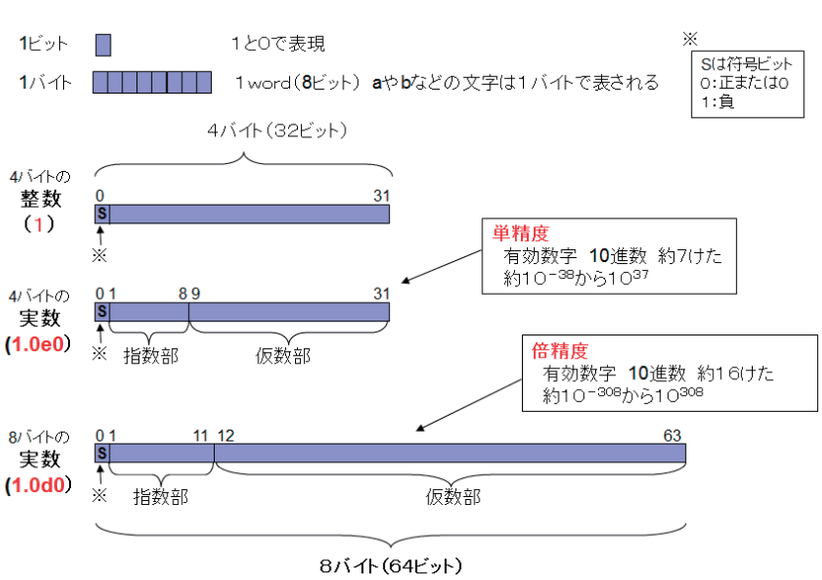


図 16　SX のデータの内部表現の例

気をつけていただきたいのは、整数の「1」と実数の「1.0」は内部表現が異なり、実数の「1.0」は正確には「1」ではないということや、シングルの「1.0e0」とダブルの「1.0d0」も異なるということです。このようなことは、計算機で計算するときには、当然気をつけるべきことですが、案外意識しない方がおられるのでご注意ください。もちろん、数字で表された「1」と文字の「1」も異なります。

８．２　書式つきデータと書式なしデータ（アスキーとバイナリ）

Fortran から入出力するには read 文と write 文を利用しますが、その扱うデータには「書式つき」と「書式なし」と呼ばれる 2 種類のデータがあります。前者を「アスキー」、後者を「バイナリ」と呼ぶこともあります。

・書式つきファイル（文字形式）

テキストファイルのため、テキストエディタで見ることができる。

```
write(n, * ) a,b
write(n,100)a,b
```

書式つき write 文

機種に依存しないので、移植が容易。
編集処理が入るため、アクセスが遅い。
書式なしファイルに比べてファイル容量が 2〜3 倍大きくなる。

プログラム中で、(n,*) ,(n,100)のように書式を指定して書き出したデータを書式付きと呼びます。 n は外部装置番号と呼ばれる番号で、ファイルを指定します。利用者が任意につけることができますが、番号は 1-4, 7-99 を用いると、どんな機種でも使用できます。

・書式なしファイル

バイナリファイルのため、テキストエディタで見ることができない。
機種に依存する。(計算機の内部表現形式)
編集処理が入らないため、アクセスが速い。
書式つきファイルに比べてファイル容量が小さくなる。

```
write(n) a,b
```

書式なし write 文

プログラム中では右上のように、ファイルの番号 n のみを指定し、書式つきの時に指定した「＊」や「100」の書式の指定をせずに a, b のデータを書き出すと、書式なしと言われるバイナリの形式でデータを出力します。

例えば、「1.2345678^{10}」という数字を書式つきで文字として、有効けた数 4 けたで出力しろという書式で、プログラムに書いて出力すると、「+0.1235E+11」となり、11 文字 (11 バイト) も必要です。これを、書式をつけずに、単精度のバイナリのままで出力すると、約 1/3 の 4 バイトで、有効けた数約 7 けたを表すことができるのです。

```
write(6,100) a
100 format(1x,e11.4)
```

有効数字 4 けたの書式つき指定

ちなみに、「write(6,*) a」　とすると「+1.2345678E+10」のようになり、15 文字 (15 バイト) と 4 倍も大きくなります。(注：実際には＋は表示されませんが、符号の文字も必要なので、ここではわざと＋と記載しています)

８．３　プログラム中の read、write とファイルの関係

プログラム中に、read 文があれば読みにいきますし、write 文があれば書きます。では、いったいどこのデータを読み書きするのでしょうか？Fortran プログラム中では装置番号を用いてファイルを指定しています。その装置番号がどのファイルに該当するかは、指定方法や、実行のさせ方によります。

基本的に read も write も同じですが、Fortran では５番と６番の装置番号は特別な意味があり、それぞれ標準入力、標準出力に対応します。ワークステーションやパソコンで会話処理で実行している場合や、スパコンでもインタラクティブで実行している場合は、それぞれキーボードと画面に相当します。インタラクティブで実行する場合は、何も指定しなければ、read 文になると、プログラムはそこでとまり、キーボードからのデータ入力を待ち、write 文で書かせたものは、

画面に表示されます。

NQS を利用して、ジョブとして実行する場合には、それぞれファイルが該当します。「read(5,*)」に対応するファイルは、NQS ファイル中で「setenv F_FF05 inputfile」のように指定すると、inputfile というファイルから読み込みます。「write(6,*)」のように出力したものは、ジョブを qsub で投入したディレクトリに、”*jobname.o999*”のように o（オー）とリクエスト番号が付加されたファイルが標準出力として返ってきます。

一般的には 5,6 以外の 1-99 の任意の数字をその他の装置番号として用います。プログラム中には、「write(8,*)」のように記載しておき、実行時にどう指定するかによって、実際に入出力するファイルが決まります。

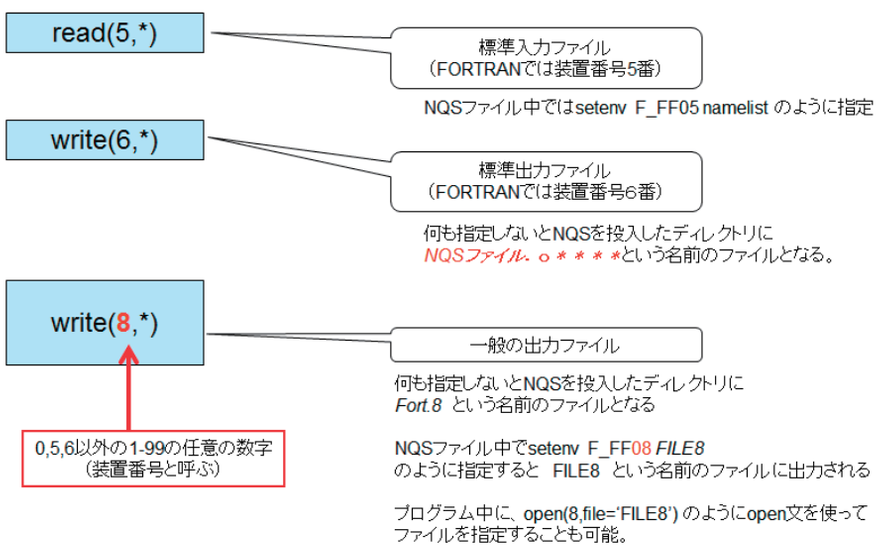


図 17　標準入出力と READ,WRITE 文の関係

NQS ファイル中に「setenv F_FF08 outdata1」のように指定すると「outdata1」というファイルに出力されます。

NQS ファイル中に何も指定しないで実行させると、「fort.8」のように装置番号が付加されたファイルが作成されます。プログラム中に、open(8,file=‘FILE8’）のように open 文を使って指定することもできます。

ファイル名は上記のように指定しますが、どのディレクトリかという指定は、インタラクティブ（会話型）の場合は、ロードモジュールを実行させたディレクトリにファイルが入出力されます。NQS の場合は何も指定しないとスパコンのホームディレクトリに入出力され、ディスク容量の制限でエラーになったりしますので注意してください。どのディスク（ファイルシステム、ディレクトリ）を利用すべきかをよく考え、システムにふさわしい場所を使いましょう。

8．4　ビッグエンディアンとリトルエンディアン

スパコンで大容量のデータを扱う場合は書式なしファイル（バイナリデータファイル）を用いますが、データサイズが小さい、またはパソコンにダウンロードして処理する場合は書式つきファイルを用いるのが一般的です。容量の大きいデータを扱う場合は、バイナリデータを利用する

ようにしましょう。ただ、同じ IEEE フォーマットと呼ばれるバイナリデータでもバイトの並びが異なるリトルエンディアン、ビッグエンディアンという形式があり、注意が必要です。

ビッグエンディアンとリトルエンディアンの変換方法はいろいろありますが、以下のようにソフトウェアの機能を使う方法や、コンパイルオプションや環境変数を利用するなどの方法があります。サイバーサイエンスセンターと阪大 CMC の SX(ビッグエンディアン)で計算したバイナリデータを他のワークステーションやパソコン（リトルエンディアン）で解析するなどの場合に利用している方法です。

- 種々Fortran のコンパイルオプションを利用。
- Fortran の環境変数を利用。(SX もこの方法で変換できます)
- IDL や AVS というソフトウェアを用いて可視化などの作業を行うときに、ソフトウェアの機能を利用して変換。

以上のようなものですが、機種によってエンディアンが異なるということを知っておくことが重要です。自分の利用している環境や、ファイルの大きさ、用途にもよりますので、分からないときは、システム管理者などにご相談ください。

謝辞

この記事のもとになるテキストの作成にあたっては、東北大学サイバーサイエンスセンター、大阪大学サイバーメディアセンター、地球シミュレータセンター、NEC の皆様の応援とご助言をいただきました。 摂南大学の田口先生と核融合研究所(NIFS)の坂上先生からもコメントをいただきました。東北大学サイバーサイエンスセンターの共同利用支援係、共同研究支援係の皆様には、丁寧に原稿のチェックと修正をしていただきました。最後に執筆の機会を与えていただきました、東北大学サイバーサイエンスセンター小林広明センター長に謝意を表します。

参考文献

[1] 「パソコン＆スーパーコンピュータで計算するための基礎知識」、福田優子(※)

[2] 「Fortran でシミュレーションしよう」摂南大学理工学部　田口俊弘(※)

[3] 大阪大学レーザーエネルギー学研究センター　公開テキスト（※を公開しています）
http://www.ile.osaka-u.ac.jp/research/cmp/text.html

[4] HPF 推進協議会　http://www.hpfpc.org/

[5] 「PC クラスタで並列プログラミング　–High Performance Fortran で楽々並列化」、津田孝夫監修、岩下英俊・坂上仁志・妹尾義樹・林康晴共著、培風館、2011 年 3 月初版

[6] 「HPF 講習会テキスト 6 種類」（※）

[7] 東北大学サイバーサイエンスセンター　http://www.ss.isc.tohoku.ac.jp/

[8] 大阪大学サイバーメディアセンター　http://www.hpc.cmc.osaka-u.ac.jp/

［大規模科学計算システム］ 高速化推進研究活動報告第 5 号より転載

スーパーコンピュータ SX-9 の高速化

スーパーコンピューティング研究部　江川隆輔　岡部公起
情報部情報基盤課　伊藤英一　小野敏　山下毅
日本電気株式会社　撫佐昭裕　神山典　小久保達信
吉村健二　遠藤清隆　小沢実希
坂本英顕　金野浩伸　坂口祐太
NEC システムテクノロジー株式会社　曽我隆

4.1 SX-9 の特徴

SX-9 は高速ベクトルプロセッサと大規模主記憶装置を有するベクトル型スーパーコンピュータである. 東北大学サイバーサイエンスセンターでは 18 ノードの SX-9 を導入しており, そのうち 16 ノードがノード間接続装置(以下, IXS)に接続されたマルチノードシステムである. 表 4.1-1 に SX-9 の主要諸元を, 表 4.1-2 に SX-9 マルチノードシステムの主要諸元を示す.

表 4.1-1 SX-9 主要諸元

項目			諸元
最大ベクトル演算性能			1.6TFLOPS
CPU 数			16
CPU	レジスタ	ベクトルレジスタ	144KB
		ベクトルマスクレジスタ	256bit×16
		スカラレジスタ	64bit×128
	データ形式	固定小数点	32/64bit
		浮動小数点	32/64/128bit
		*128bit はスカラ命令のみサポート	IEEE
		論理	64bit
	ベクトル演算パイプライン		5 種類×8 セット
	ADB		256KB
	スカラ演算パイプライン		1 セット
	キャッシュ		命令:32KB
			オペランド:32KB
主記憶装置	容量		1TB
	最大データ転送能力		4TB/s

表 4.1.2 SX-9 マルチノードシステム主要諸元

項目	諸元
ノード数	16
最大ベクトル演算性能	26.2TFLOPS
CPU 数	256
主記憶装置容量	16TB
IXS 転送性能(片方向)	128GB/s (ノード当たり) 2TB/s (システム全体)

(1) CPU の特徴

SX-9 に搭載されている CPU は, 単一チップで 102.4GFLOPS のベクトル演算性能を有するベクトル

プロセッサである. 図 4.1-1 に SX-9 のプロセッサ構成を示す. CPU 内部はベクトルユニットとスカラユニットに分かれており, ベクトルユニットは 8 つのベクトルパイプラインセットで構成されている. 各ベクトルパイプラインセットはマスク演算器, 論理演算器, 乗算器×2, 加算器×2, 除算器/平方根演算器を有している. また, SX-9 から MAX/MIN 関数がハードウェア命令化され, 最大値・最小値の計算時間が短縮されている.

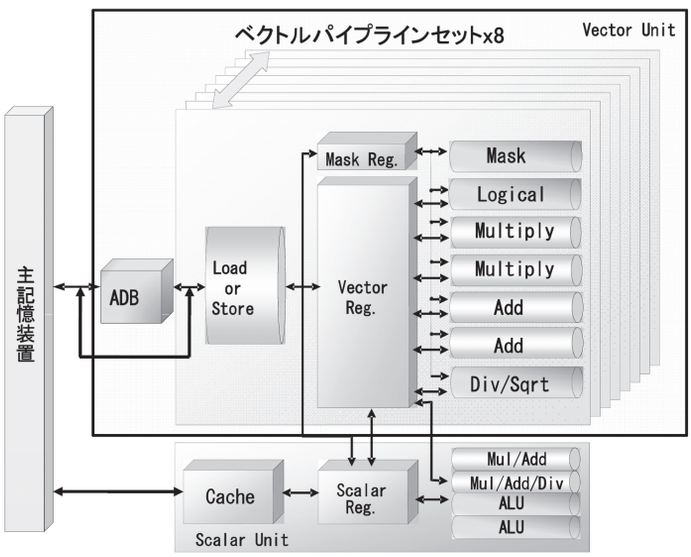


図 4.1-1 SX-9 プロセッサの構成図

(2) メモリシステムの特徴

SX-9 は, CPU と主記憶装置(MMU)間の距離が論理的に等しい共有メモリ型計算機である. 図 4.1-2 に SX-9 のメモリ構成を示す. SX-9 は CPU と主記憶装置間にネットワークルータ(以下, RTR)という内部スイッチング機構を配することで, 広いメモリバンド幅と大規模共有メモリを実現している. また, 主記憶装置はノードあたり 32,768 個のメモリバンクで構成されている. バンクのアクセスに関しては SX-7C までの SX シリーズでは一度に 1 演算要素(8Byte)を処理していたが, SX-9 から 2 演算要素(16Byte)を処理するように強化し, ロード・ストアの処理時間の短縮を図っている.

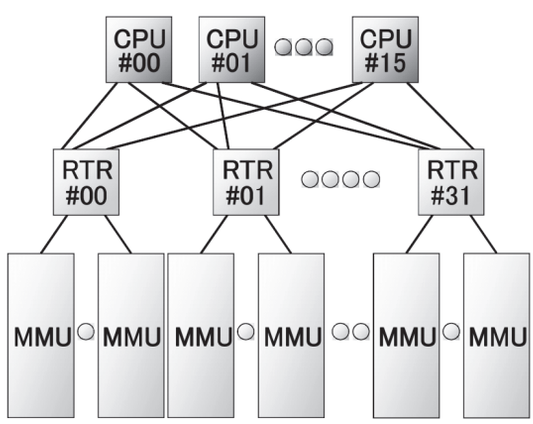


図 4.1-2 SX-9 のメモリ構成図

SX-9 では CPU－主記憶装置間に ADB（Assignable Data Buffer）と呼ばれるベクトルデータを選択的にバッファする機構を有している．図 4.1-3 に ADB の概念図を示す．ADB は CPU－主記憶装置間よりメモリバンド幅が広く，メモリレイテンシ（遅延）が短いという特徴を持っている．表 4.1-3 に SX-9 の主記憶装置と ADB のデータ供給性能を示す．データ供給性能は，演算性能あたりのデータ供給量を表す指標としてベクトル演算性能あたりのメモリバンド幅（Byte/FLOP）である．

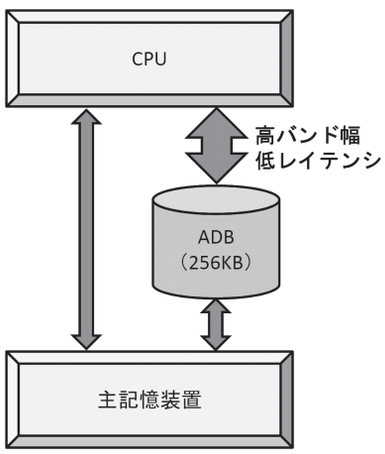


図 4.1-3 ADB の概念図

表 4.1-3 SX-9 の主記憶装置と ADB のデータ供給性能

	CPU-主記憶装置間	CPU-ADB 間
バンド幅	256.0GB/s	409.6GB/s
ベクトル演算性能あたりのバンド幅	2.5Byte/FLOP	4.0Byte/FLOP

(3) ソフトウェアの特徴

① コンパイラ

SX-9 用コンパイラとして FORTRAN90/SX と C++/SX が利用可能である．FORTRAN90/SX は ISO/IEC 1539-1:1997，ISO/IEC1539:1991 に準拠している．C++/SX は ISO/IEC 9899:1999，ISO/IEC 14882:1998 に準拠している．

FORTRAN90/SX，C++/SX は自動ベクトル化機能，自動並列化機能を有しており，ユーザは意識することなく，ベクトル化，並列化することができる．

② MPI ライブラリ

SX-9 用 MPI ライブラリは MPI-1.2 および MPI-2 に準拠した機能を提供している．

③ 科学技術計算ライブラリ

SX-9 用ライブラリとして，科学技術計算ライブラリ ASL，数値計算ライブラリ MathKeisan が利用可能である．

4.2 ベクトル処理による高速化

(1) ベクトル処理の概要

科学技術計算プログラムの多くは特定の DO ループに実行の大部分が集中し，DO ループ内の配

列データ(ベクトル)に対し繰り返し演算が行われている. 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, スカラ処理は繰り返し演算を逐次的に実行するものである.

(2) ベクトル化率

ベクトル型スーパーコンピュータではプログラム中のベクトル処理可能な部分を高速に実行している. 図 4.2-1 に同一のプログラムをスカラ処理する場合とベクトル処理する場合の実行時間に関する概念図を示す.

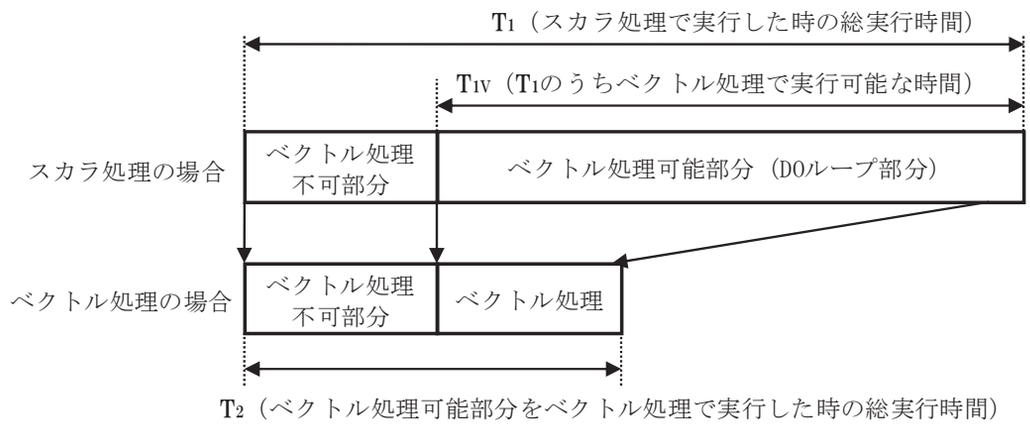


図 4.2-1　ベクトル処理による実行時間短縮のイメージ

ここで, あるプログラムをスカラ処理で実行する場合の総実行時間を T_1, そのプログラムでベクトル処理が可能な部分の実行時間を T_{1V} とすると, ベクトル化率 α は以下の式で定義される.

$$\text{ベクトル化率}\quad \alpha = \frac{T_{1V}}{T_1}$$

また, そのプログラムをベクトル処理する場合の性能向上比 P は, スカラ処理性能とベクトル処理性能の比を β とすると,

$$\text{性能向上比}\quad P = \frac{1}{(1-\alpha)+\frac{\alpha}{\beta}}$$

と表される.

この式から, ベクトル化率と性能向上比の関係は図 4.2-2 のようになる(アムダールの法則). この図からベクトル化率 80%程度では大きな性能向上は見られず, ベクトル化率 90%を超えたあたりから急激に性能が向上していることがわかる. ベクトル型スーパーコンピュータで高い実効性能を得るためにはベクトル化率を 100%にできる限り近付ける必要がある.

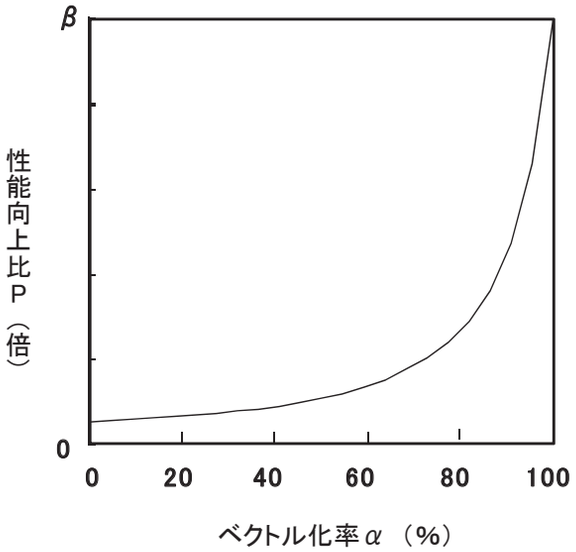


図 4.2-2 ベクトル化率と性能向上比（アムダールの法則）

ベクトル化率を求めるため，プログラムをスカラ実行する場合とベクトル実行する場合のそれぞれの実行時間が必要である．しかし，これらの実行時間は同時に得られないのでベクトル化率の算出は困難である．そのため SX-9 ではベクトル化率の代わりにベクトル演算率をベクトル化の指標としている．後述の PROGINF や FTRACE などの性能解析機能に出力される情報はベクトル演算率である．ベクトル演算率は，プログラムで処理される全演算要素数に対するベクトル演算命令で処理される演算要素数の割合で算出し，ほぼベクトル化率とみなせる指標である．

(3) ベクトル長

ベクトル処理を効率的に行う上で重要な指標にベクトル長がある．ベクトル長はベクトル化対象の DO ループの繰り返し回数のことであり，効率良くベクトル処理を行うためには，できるだけループ長を長くする必要がある．

一般に命令を発行してから結果が返ってくるまでに遅延時間が存在する．この遅延時間を立ち上がり時間と呼ぶ．図 4.2-3 にベクトル処理における立ち上がり時間および演算時間の概念図を示す．図 4.2-4 にベクトル長と立ち上がり時間の関係を示す．立ち上がり時間はベクトル長が異なる場合でも一定である．よって総演算量が等しい場合，短ベクトル長処理を繰り返すよりも長ベクトル長処理を一括して行う方が実行時間を短くすることができる．このように高速化を図るためには，DO ループのベクトル長を十分に確保し，演算時間に対する立ち上がり時間の占める割合を低くすることが重要である．

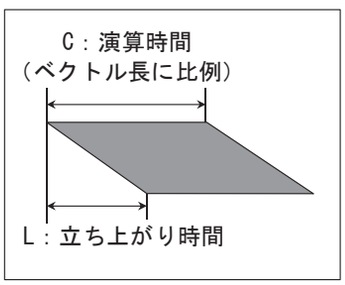


図 4.2-3　立ち上がり時間

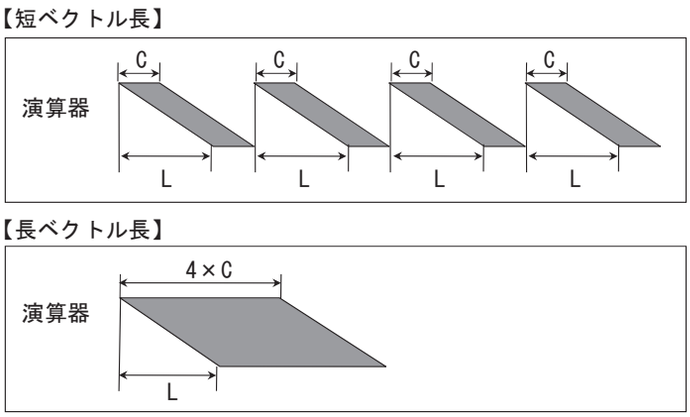


図 4.2-4　ベクトル長と立ち上がり時間

(4)　メモリアクセス

ベクトルユニットはベクトル処理によって複数の演算要素を一括で処理することが可能である．この処理を高速に行うためには，主記憶装置からベクトルユニットに対して，データを滞りなく供給することが必要である．

データ供給の遅延の要因にバンクコンフリクトがある．バンクコンフリクトは CPU ポート競合とメモリネットワーク競合の 2 つに分類される．CPU ポート競合は CPU 内における同一のポートにロード・ストアが集中した時に発生する競合を指す．メモリネットワーク競合は同一のメモリバンクへのアクセスで発生する競合や CPU と主記憶装置間の経路上で発生する競合を指す．バンクコンフリクトが発生するとデータ供給能力が低下し，高速化の妨げの要因になる．バンクコンフリクトを回避するためには，メモリアクセスの間隔が偶数となることを避け，可能ならば連続アクセスとすることや，ループのアンロールを行うなどしてメモリアクセス回数を減らすことが重要である．

また，メモリアクセスの効率化では，前述の ADB の活用があげられる．ADB を利用することで主記憶装置からのデータ供給性能を補うことや，メモリレイテンシを短くすることが可能となり性能向上を図れる場合がある．後述のチューニング例 4.3.8(4)を参考に ADB の利用を試みて頂きたい．

4.3 高速化技法

4.3.1 多重 DO ループの一重化による高速化

(1)　多重 DO ループの一重化の概要

多重 DO ループの一重化とは，図 4.3-1 のような多重構造の DO ループを，図 4.3-2 のように一重の DO ループに書き換えることをいう．図 4.3-1 では，内側の DO ループはベクトル長 10 でベクトル処理され，外側の DO ループはこのベクトル処理が 20 回実行されるので，ベクトル処理の立ち上がり時間は 20 回分必要となる(4.2.(3) 図 4.2-4 参照)．一方，図 4.3-2 では，ベクトル長 200 のベクトル処理を1回だけ実行すればよいので，立ち上がり時間は1回分だけですみ，高速化することができる．ここで，配列 a(ij,1)，b(ij,1)，c(ij,1)は一重化された DO ループで，元の配列 a(i,j)，b(i,j),c(i,j)を連続してアクセスするために書きかえたものである．

多重 DO ループの一重化は，ここで示した例のように，内側の DO ループの長さが短いときに特に効果がある．

```
!多重 DO ループ
real,dimension(10,20)::a,b,c
do j=1,20
  do i=1,10
    a(i,j)=b(i,j)+c(i,j)
  enddo
enddo
```

図 4.3-1 多重 DO ループの例

```
!図 4.3-1 の多重 DO ループの一重化
do ij = 1, 10*20
  a(ij,1) = b(ij,1) + c(ij,1)
enddo
```

図 4.3-2 多重 DO ループの一重化の例

(2) コンパイラによる DO ループの一重化

多重 DO ループは，コンパイラによって自動的に一重化される場合がある．4.3.1.(1)であげた図 4.3-1 から図 4.3-2 への変形は，コンパイラが自動的に行うものである．コンパイラが変換可能なのは，多重ループが以下の条件を満たす場合である．

① 密な多重 DO ループ(注)である
② 指標変数を含む添字式が，その配列の次元の下限から上限までの値をとる
③ 各 DO ループの指標変数は，ループ中に現れる配列要素の添字中に左から連続して同じ形式かつ同じ順序で現れている
④ DO ループ中に現れる配列要素の各添字式は異なるループの指標変数を含まない
⑤ 一重化しても定義・引用の順序関係が正しく保たれる
⑥ ループ中に現れるすべての配列はポインタでも形状引継ぎ配列でもない

(注)密な多重 DO ループとは，直接の入れ子関係をなす DO ループのうち，外側ループの DO 文と内側ループの DO 文の間および内側ループの ENDDO 文と外側ループの ENDDO 文の間に，実行文が現れないもの(およびそれと等価な構造の多重ループ)．

(3) 自動的に一重化されない多重 DO ループのチューニング

DO ループ中にポインタや形状引継ぎ配列がある場合は上述⑥の条件を満たさないため，コンパイラによる自動的なループの一重化は行われない．このような場合，ループはコンパイル時にオプション「-pvctl collapse」(注 1)を指定することにより一重化することができる．

この他，自動的に一重化されない例として，図 4.3-3 のように DO ループの範囲の外に冗長な領域を持つ配列が含まれる場合をあげられる．ループの指標変数が配列 work の下限から上限までの値をとらず，上述②の条件を満たさないため，ループの一重化は行われない．図 4.3-4 のように配列 work の冗長な領域をなくすことで，コンパイラにより自動的に一重化されるようになる．なお，図 4.3-3 と図 4.3-4 は，コンパイラによる編集リスト(注 2)の形式で記載している．

```
!オリジナル
42:              subroutine sample_f(a,b,work,p)
43:                implicit none
44:                include 'parameter.h'
45:                real*8,intent(in)  :: a(imax,jmax,kmax,3)
46:                real*8,intent(in)  :: b(imax,jmax,kmax,3)
47:                real*8,intent(in)  :: work(imax+1,jmax+1,kmax+1,3)
48:                real*8,intent(out) :: p(imax,jmax,kmax)
49:                integer :: i,j,k
50:
51: +------>       do k=1,kmax
52: |+----->         do j=1,jmax
53: ||V---->           do i=1,imax
54: |||                 p(i,j,k)=work(i,j,k,1)*(a(i,j,k,1)+b(i,j,k,1)) &
55: |||                         +work(i,j,k,2)*(a(i,j,k,2)+b(i,j,k,2)) &
56: |||                         +work(i,j,k,3)*(a(i,j,k,3)+b(i,j,k,3))
57: ||V----           enddo
58: |+-----         enddo
59: +------       enddo
```

図 4.3-3　多重 DO ループが一重化されないコードの例

```
!チューニング
42:              subroutine sample_f(a,b,work,p)
43:                implicit none
44:                include 'parameter.h'
45:                real*8,intent(in)  :: a(imax,jmax,kmax,3)
46:                real*8,intent(in)  :: b(imax,jmax,kmax,3)
47:                real*8,intent(in)  :: work(imax,jmax,kmax,3)
48:                real*8,intent(out) :: p(imax,jmax,kmax)
49:                integer :: i,j,k
50:
51: W------>       do k=1,kmax
52: |*----->         do j=1,jmax
53: ||*---->           do i=1,imax
54: |||                 p(i,j,k)=work(i,j,k,1)*(a(i,j,k,1)+b(i,j,k,1)) &
55: |||                         +work(i,j,k,2)*(a(i,j,k,2)+b(i,j,k,2)) &
56: |||                         +work(i,j,k,3)*(a(i,j,k,3)+b(i,j,k,3))
57: ||*----           enddo
58: |*-----         enddo
59: W------       enddo
```

図 4.3-4　多重 DO ループが一重化されないコードのチューニング例

(注 1)「-pvctl collapse」は，次元数が 2 以上の配列式，または多重 DO ループ中に現われる形状引き継ぎ配列，ポインタ配列の全要素がメモリ上の連続領域にあると仮定し，一重化を行うことを指示するコンパイルオプションである．

(注 2)編集リストとは，ソースコードと共に，ループと配列式のベクトル化情報，手続きのインライン展開情報，ループと配列式の並列化情報を出力する機能である．以下に編集リストに用いられる記号の意味を示す．

V：ベクトル化されたループ・配列式
W：一重化されてベクトル化されたループ・配列式
P：並列化されたループ
Y：並列化され，かつベクトル化されたループ
I：インライン展開された手続き
+：最適化されなかったループ

(4) 性能評価

(3)の図 4.3-3, 図 4.3-4 で示したチューニング前後のコードについて, 問題サイズを「imax= 50, jmax=200, kmax=1000」として, SX-9 1CPU を用いて性能評価を行う. SX-9 では, 性能情報として PROGINF(注 1)と FTRACE(注 2)がある. PROGINF からはプログラム全体の性能情報が取得でき, FTRACE からは手続(サブルーチンや関数)ごとの性能情報を取得することができる.

① PROGINF

図 4.3-5, 図 4.3-6 はチューニング前後の PROGINF である. 実効性能を比較するため, MFLOPS(1 秒間に実行された浮動小数点演算数を 100 万単位で示した値)の項目を見る. チューニング前は8,578MFLOPSであるのに対し, チューニング後には20,063MFLOPSとなっており, 実効性能が 2.33 倍向上していることが分かる. また, A.V.Length(平均ベクトル長(注 3))の項目を見ると, チューニング前は 49.9 であるのに対し, チューニング後は 252.3 となっている. チューニングにより多重 DO ループが一重化されることで, コード全体の平均ベクトル長が長くなっていることを確認できる.

```
    ******  Program Information  ******
Real Time (sec)                :            4.699699
User Time (sec)                :            4.672104
Sys  Time (sec)                :            0.023863
Vector Time (sec)              :            4.667600
Inst. Count                    :         3615133579.
V. Inst. Count                 :         1806406030.
V. Element Count               :        90320127249.
FLOP Count                     :        40080201114.
MOPS                           :        19718.922095
MFLOPS                         :         8578.619208
A. V. Length                   :           49.999904
V. Op. Ratio (%)               :           98.036742
Memory Size (MB)               :          960.031250
MIPS                           :          773.769929
I-Cache (sec)                  :            0.000384
O-Cache (sec)                  :            0.000478
Bank Conflict Time
  CPU Port Conf. (sec)         :            0.053537
  Memory Network Conf. (sec)   :            3.597606
```

図 4.3-5 チューニング前の PROGINF

```
    ******  Program Information  ******
Real Time (sec)                :            2.028015
User Time (sec)                :            1.997679
Sys  Time (sec)                :            0.025952
Vector Time (sec)              :            1.993263
Inst. Count                    :          695335689.
V. Inst. Count                 :          357973030.
V. Element Count               :        90320127249.
FLOP Count                     :        40080201166.
MOPS                           :        45381.410080
MFLOPS                         :        20063.384140
A. V. Length                   :          252.309866
V. Op. Ratio (%)               :           99.627871
Memory Size (MB)               :          960.031250
MIPS                           :          348.071782
I-Cache (sec)                  :            0.000325
O-Cache (sec)                  :            0.000416
Bank Conflict Time
  CPU Port Conf. (sec)         :            0.010343
  Memory Network Conf. (sec)   :            0.779060
```

図 4.3-6 チューニング後の PROGINF

② FTRACE

図 4.3-7, 図 4.3-8 はチューニング前後の FTRACE の情報である. ①PROGINF と同様に, 実効性能と平均ベクトル長を確認することができるが, FTRACE ではこれらの情報を手続(サブルーチンや関数)単位で取得することができる.

PROC.NAME	FREQUENCY	EXCLUSIVE TIME[sec](%)	AVER.TIME [msec]	MOPS	MFLOPS	V.OP RATIO	AVER. V.LEN	VECTOR TIME	I-CACHE MISS	O-CACHE MISS	BANK CONFLICT CPU PORT	BANK CONFLICT NETWORK
sample_f	500	4.638(99.5)	9.276	19794.6	8624.8	98.04	50.0	4.637	0.000	0.000	0.051	3.577
tp_sample	1	0.022(0.5)	22.060	14752.3	3635.5	98.33	50.0	0.022	0.000	0.000	0.000	0.015
total	501	4.660(100.0)	9.301	19770.7	8601.2	98.04	50.0	4.659	0.000	0.000	0.051	3.593

図 4.3-7 チューニング前の FTRACE 情報

PROC.NAME	FREQUENCY	EXCLUSIVE TIME[sec](%)	AVER.TIME [msec]	MOPS	MFLOPS	V.OP RATIO	AVER. V.LEN	VECTOR TIME	I-CACHE MISS	O-CACHE MISS	BANK CONFLICT CPU PORT	BANK CONFLICT NETWORK
sample_f	500	1.959(98.9)	3.918	46114.9	20420.2	99.63	256.0	1.959	0.000	0.000	0.011	0.715
tp_sample	1	0.022(1.1)	22.036	14749.9	3639.4	98.45	50.0	0.022	0.000	0.000	0.000	0.015
total	501	1.981(100.0)	3.954	45765.9	20233.5	99.63	252.3	1.980	0.000	0.000	0.011	0.730

図 4.3-8 チューニング後の FTRACE 情報

(注 1)プログラム実行解析情報:プログラム全体の性能情報が標準エラー出力ファイルに出力される. 以下に PROGINF 機能で取得可能な情報を示す.

Real Time	:経過時間(プログラム実行に要した時間)
User Time	:ユーザ時間(プログラム実行に要した CPU 時間の内, ユーザルーチンが実行に要した時間)
Sys Time	:システム時間(プログラム実行に要した CPU 時間の内, システムルーチンが実行に要した時間)
Vector Time	:ベクトル命令実行時間
Inst. Count	:全命令実行数
V. Inst. Count	:ベクトル命令実行数
V. Element Count	:ベクトル命令で実行された演算要素の個数
FLOP Count	:実行された浮動小数点演算の個数
MOPS	:1 秒間に実行された演算数を 100 万単位で示した値
MFLOPS	:1 秒間に実行された浮動小数点演算数を 100 万単位で示した値
A. V. Length	:平均ベクトル長(注 3 参照)
V. Op. Ratio	:全実行命令の個数の内, ベクトル命令の占める割合
Memory Size	:メモリ使用量
MIPS	:1 秒間に実行された命令数を 100 万単位で示した値
I-Cache	:命令キャッシュミスにより発生した合計時間
O-Cache	:オペランドキャッシュミスにより発生した合計時間
Bank Conflict Time	:バンクコンフリクト時間
CPU Port Conf.	:CPU ポート競合時間
Memory Network Conf.	:メモリネットワーク競合時間

（注 2）性能解析情報：コンパイル時にオプション「-ftrace」を指定することで，手続（サブルーチンや関数）ごとの性能解析情報を採取することができ，プログラム実行後に解析情報が標準出力ファイルに出力される．特に，チューニング対象とする手続を選択する際に活用すると良い．以下に FTRACE 機能で取得可能な情報を示す．

PROC.NAME	：手続名
FREQUENCY	：手続の呼び出し回数
EXCLUSIVE TIME	：手続の実行に要した占有の CPU 時間と，手続全体の実行に要した CPU 時間に対する比率
AVER.TIME	：手続の 1 回の実行に要した平均 CPU 時間
MOPS	：1 秒間に実行された演算数を 100 万単位で示した値
MFLOPS	：1 秒間に実行された浮動小数点演算数を 100 万単位で示した値
V.OP RATIO	：ベクトル演算率
AVER V.LEN	：平均ベクトル長（注 3 参照）
VECTOR TIME	：ベクトル命令実行時間
I-CACHE MISS	：命令キャッシュミスにより発生した合計時間
O-CACHE MISS	：オペランドキャッシュミスにより発生した合計時間
BANK CONFLICT	：バンクコンフリクト時間
CPU PORT	： CPU ポート競合時間
NETWORK	： メモリネットワーク競合時間

（注 3）SX-9 の性能情報の一つに，平均ベクトル長がある．SX-9 は一つのベクトル命令で最大 256 組の演算要素を処理することができるので，例えば DO ループの長さが 1000 の場合，256，256，256，232 という 4 つの処理単位に DO ループを分割して処理する．この場合，平均ベクトル長は 250 になる．

4.3.2 多重 DO ループのアンローリングによる高速化

(1) 多重 DO ループのアンローリングの概要

DO ループのアンローリングとは，ループ本体の演算を n 倍にし，繰り返し数を 1/n にする処理である．このとき n をアンローリングの段数という．

図 4.3-9 の外側の j のループについて，2 段のアンローリングをしたのが図 4.3-10 である．外側の j のループは増分 2 とし，内側の i のループ内に j+1 の演算を追加して，内側ループの演算を 2 倍にしている．これにより，図 4.3-10 の①の **x(i)**のストアと②の **x(i)**のロードが不要になる．つまり，図 4.3-10 は図 4.3-11 のように変形できる．よって，多重 DO ループ全体で考えると，x(i)のロード回数，ストア回数がアンローリング前の半分になる．

多重 DO ループのアンローリングでは，一般に，ベクトル化されない外側ループについてアンローリングを行うことで性能向上が期待できる．特にロード・ストア回数が減少するような場合には，大きな効果が得られることが多い．

```
!多重 DO ループ
do  j=1,100
    do  i=1,n
        x(i)=x(i)+a(i,j)*b(j)
    enddo
enddo
```

図 4.3-9　多重 DO ループの例

```
!図 4.3-9 の多重 DO ループを，外側ループについて 2 段でアンローリング
do  j=1,99,2
    do  i=1,n
        x(i)=x(i)+a(i,j)*b(j)        ・・・①
        x(i)=x(i)+a(i,j+1)*b(j+1)    ・・・②
  enddo
enddo
```

図 4.3-10　外側 DO ループのアンローリング後

```
!図 4.3-10 の x(i)のロード・ストアを削減した結果
do  j=1,99,2
    do  i=1,n
        x(i)=x(i)+a(i,j)*b(j)+a(i,j+1)*b(j+1)
  enddo
enddo
```

図 4.3-11　x(i)のロード・ストア削減の結果

(2)　差分コードのアンローリング

図 4.3-12 は，差分コードのチューニング例である．多重 DO ループにおいてコンパイラが外側ループのアンローリングを行うように，コンパイラ指示行 OUTERUNROLL(注 1)を挿入している．

```
!チューニング
 :
47: +------>      do k=2,kmax-1
48: |             !CDIR OUTERUNROLL=4
49: |+----->        do j=2,jmax-1
50: ||V---->           do i=2,imax-1
51: |||                 s0=c(i,j,k,1)*(a(i+1,j+1,k)-a(i+1,j-1,k) &
52: |||                               -a(i-1,j+1,k)+a(i-1,j-1,k)) &
53: |||                   +c(i,j,k,2)*(a(i,j+1,k+1)-a(i,j-1,k+1) &
54: |||                               -a(i,j+1,k-1)+a(i,j-1,k-1)) &
55: |||                   +c(i,j,k,3)*(a(i+1,j,k+1)-a(i-1,j,k+1) &
56: |||                               -a(i+1,j,k-1)+a(i-1,j,k-1))
57: |||                 ss=s0*b(i,j,k)
58: |||                 p(i,j,k)=a(i,j,k)+omega*ss
59: ||V----           enddo
60: |+-----         enddo
61: +------       enddo
 :
```

図 4.3-12　差分コードのチューニング例

図 4.3-13 は図 4.3-12 のチューニング例について，アンローリング部分の変形結果を変形リスト(注 2)より抜粋したものである．

図 4.3-13 アンローリング後の変形リスト

この 4 段のアンローリングの例では，配列 a，b，c は全部で 68 個の要素がロードされるが，そのうち配列 a は 8 個の要素が 2 回参照される(注 3)．重複する部分は再度ロードされることが無いため，DO ループ内のロード回数が 68 回から 60 回に減る．同様に，8 段でアンローリングした場合は，配列 a，b，c は全部で 136 個の要素がロードされるが，そのうち配列 a は 24 個の要素が 2 回参照されるため，DO ループ内のロード回数が 136 回から 112 回に減る．

(注 1)直後の外側 DO ループに対し，強制的に n 段のアンローリングを行うことを指定する指示行である．これを指定した場合，コンパイラはループ内のデータの依存関係をチェックせずにアンローリングを行う．なお OUTERUNROLL の段数は，指定した値以下の 2 のべき乗の値となる．

(注 2)変形リスト:コンパイラが行った最適化の結果，実際に生成されたオブジェクトコードの構造をソースコードのイメージで表示したリストである．

(注 3)図 4.3-13 では，変形リストを抜き出して行番号を追記し，2 回参照される配列がわかりやすいよう，配列ごとに形式を変えて強調表記している．例えば，行番号 4 と 10 の矢印で示した a(i,j+2,k)の配列は **a(i,j+2,k)**と表記している．

(3) 性能評価

(2)の図 4.3-12，図 4.3-13 で示しているチューニング前後のコードについて，問題サイズを「imax=256, jmax=256, kmax=256」とし，SX-9 1CPU を用いて性能評価を行う．表 4.3-1 が性能測定結果である．4 段でアンローリングする場合には，チューニング前に比べて 1.16 倍，8 段でアンローリングする場合には，1.30 倍性能向上していることが分かる．

表 4.3-1 アンローリングの性能比較

	性能
チューニング前	16.29GFLOPS
アンローリング後(4 段)	18.91GFLOPS
アンローリング後(8 段)	20.21GFLOPS

(4)　アンローリング段数による性能の違い

(3)での評価結果からもわかるとおり，アンローリング段数によって性能に差が生じている．図 4.3-14 は，(2)の差分コードについて，段数を 2，4，8，16，32 と変えて性能測定している結果である．

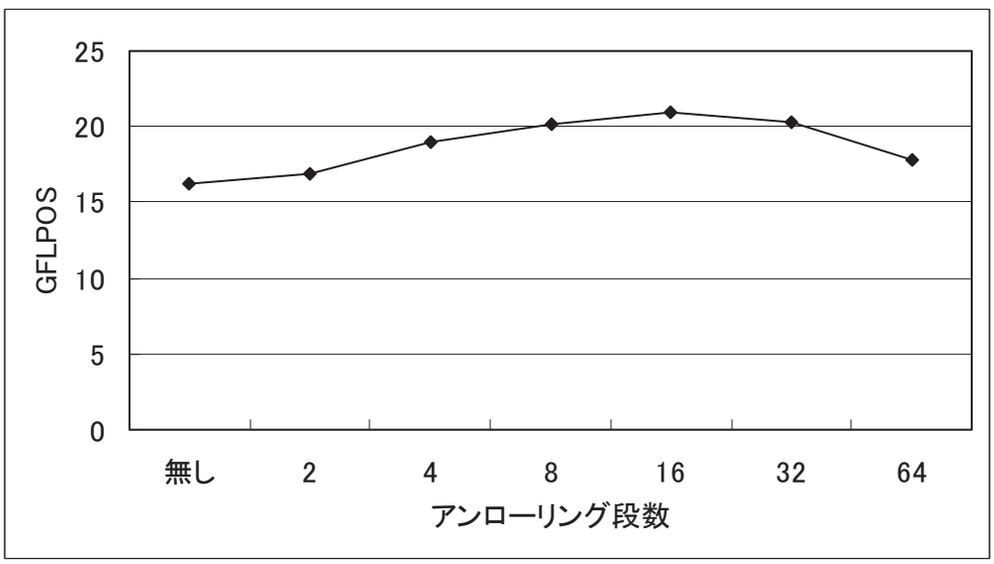


図 4.3-14　アンローリング段数による性能差

一般に，アンローリング段数を増やすと性能が向上する．しかし，段数が増えるに伴いループ中の中間変数が増加する．そのため段数を増やしすぎると途中で中間変数を保存するレジスタが足りなくなり，性能が劣化する．図 4.3-14 のとおり，(2)の差分コードの例では，16 段でアンローリングすると最も性能が高くなる．最適なアンローリング段数はコードにより異なる．

(5)　アンローリングの自動/手動に関する補足

外側の DO ループをアンローリングすることで，内側の DO ループのベクトル演算パイプラインを有効に利用できる場合や，ロード・ストア回数を減らす効果がある場合，コンパイラは自動的にアンローリングを行う．コンパイラによって自動的にアンローリングされる場合でも，デフォルトの段数（通常 4 段）以外で実行すると更に性能向上が得られる場合がある．多重ループでは，ロード・ストア回数が最も削減される次元でアンロールするのが有効である．

4.3.3　IF 文を含む DO ループのチューニング

(1)　IF 文を含む DO ループのベクトル処理の概要

図 4.3-15 のように IF 文を含む DO ループは，つぎのように処理される．

①　条件式「flag(n).eq.1」の真偽に応じて，マスクベクトルを作成

②　条件式の真偽にかかわらず，b(n)+c(n) を演算

③　マスクベクトルが真である要素のみ，b(n)+c(n)の演算結果を a(n)に代入

このように，IF 文を含まない DO ループに比べ，条件分岐のための処理が加わるので，分岐の数が多いと性能劣化の要因となる．

```
real,dimension(nsp)::a,b,c
integer::flag(nsp)

do n=1,nsp
  if(flag(n).eq.1)then
    a(n)=b(n)+c(n)
  endif
enddo
```

図 4.3-15 IF 文を含んだ DO ループの例

なお, IF 文を含んでいても, 図 4.3-16 のようにループの繰り返しの間, 条件式の結果が不変の場合, コンパイラは IF 文を DO ループの外に出してベクトル化するので, IF 文のオーバヘッドは発生しない.

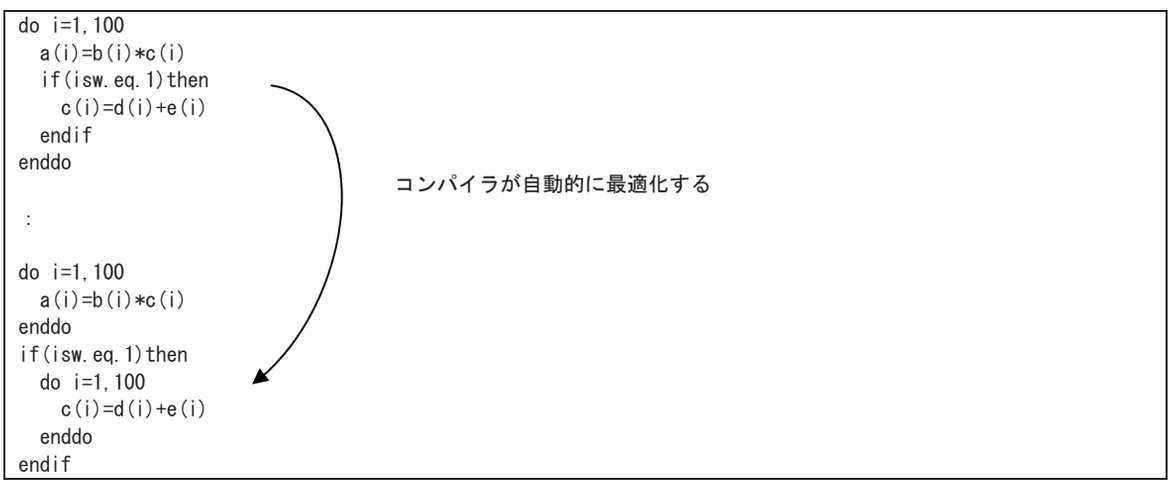


図 4.3-16 IF 文が DO ループの外に展開される変形イメージ

(2) マスクベクトルを用いてベクトル化しているループの性能

図 4.3-17 の DO ループは, マスクベクトルを用いてベクトル化するので TRUE, FALSE のどちらの場合も演算が実行される. 演算結果は, マスクベクトルを用いて TRUE に対応する要素のみ代入される. このため, flag_v の真の割合(以下, 真率)に関係なく, DO ループの処理時間はほぼ一定になる.

```
do n=1,nsp
  if(flag_v(n) .eq. 1) then
    tmpx3(n)=(tmpx1(n)-tmpx2(n))*2.0d0
    tmpy3(n)=(tmpy1(n)-tmpy2(n))*2.0d0
    tmpz3(n)=(tmpz1(n)-tmpz2(n))*2.0d0
  endif
enddo
```

図 4.3-17 一般的な IF 文を含む DO ループ

図 4.3-18 は, 以下の 3 パターンで図 4.3-17 の DO ループを SX-9 1CPU で実行するときの性能値である. それぞれの実行時間がほとんど変わらないことがわかる.

- Rfv_vt :flag_v の値が全て TRUE の場合
- Rfv_vf :flag_v の値が全て FALSE の場合
- Rfv_50 :flag_v の値がランダムに TRUE/FALSE となる場合

PROC.NAME	FREQUENCY	EXCLUSIVE TIME[sec](%)	AVER.TIME [msec]	MOPS	MFLOPS	V.OP RATIO	AVER. V.LEN	VECTOR TIME	I-CACHE MISS	O-CACHE MISS	BANK CONFLICT CPU PORT	BANK CONFLICT NETWORK
Rfv_vt	30	0.757(0.3)	25.237	48061.8	15954.8	99.59	256.0	0.757	0.000	0.000	0.006	0.327
Rfv_vf	30	0.755(0.3)	25.174	48182.2	15994.8	99.59	256.0	0.755	0.000	0.000	0.000	0.327
Rfv_50	30	0.758(0.3)	25.262	48015.4	15939.4	99.59	256.0	0.758	0.000	0.000	0.014	0.329

図 4.3-18　FTRACE 情報の一覧

(3)　事例1:IF 文をループの外に移動するチューニング

図 4.3-19 の多重ループは，TRUE 側の演算量が FALSE 側の演算量よりも大きい IF 文の例である．マスクベクトルを用いるベクトル化は，条件式の結果に関わらず，どちらの演算も実行しているため，本来必要な演算数より多くなる．以下に，ループの入れ替えによって IF 文をループの外に移動するチューニングの例を示す．

```
!オリジナル
153: +------>     do k=1,nz
154: |+----->       do j=1,ny
155: ||V---->         do i=1,nx
156: |||               if(lc(i,k).eq.0) then
157: |||                 p(1,i,j,k) = ff*va(1,i,j,k)*(f1*vb(i,j,k)+gv10) + d1*c(i,j,k)
158: |||                 p(2,i,j,k) = ff*va(2,i,j,k)*(f2*vb(i,j,k)+gv10) + d2*c(i,j,k)
159: |||                 p(3,i,j,k) = ff*va(3,i,j,k)*(f3*vb(i,j,k)+gv10) + d3*c(i,j,k)
160: |||               else
161: |||                 p(1,i,j,k) = d1*c(i,j,k)
162: |||                 p(2,i,j,k) = d2*c(i,j,k)
163: |||                 p(3,i,j,k) = d3*c(i,j,k)
164: |||               endif
165: ||V----         enddo
166: |+-----       enddo
167: +------     enddo
```

図 4.3-19　TRUE 側の演算量が FALSE 側の演算量よりも大きい IF 文のコード

この IF 文の条件式「lc(i,k).eq.0」はループ変数 j に依存しておらず，ループ内の配列に依存関係がない．このため DO ループ j を IF 文の内側に移動できる．

チューニング後のコードを図 4.3-20 に示す．IF 文は DO ループ j よりも外側にあり，マスクベクトル処理とならない．このチューニングで IF 文の条件式により内側の DO ループが選択され，真の時だけ実行される．

```
!チューニング
153: +------>     do k=1,nz
154: |+----->         do i=1,nx
155: ||                 if(lc(i,k).eq.0) then
156: ||V---->       do j=1,ny
157: |||                 p(1,i,j,k) = ff*va(1,i,j,k)*(f1*vb(i,j,k)+gv10) + d1*c(i,j,k)
158: |||                 p(2,i,j,k) = ff*va(2,i,j,k)*(f2*vb(i,j,k)+gv10) + d2*c(i,j,k)
159: |||                 p(3,i,j,k) = ff*va(3,i,j,k)*(f3*vb(i,j,k)+gv10) + d3*c(i,j,k)
160: ||V----        enddo
161: ||                 else
162: ||V---->       do j=1,ny
163: |||                 p(1,i,j,k) = d1*c(i,j,k)
164: |||                 p(2,i,j,k) = d2*c(i,j,k)
165: |||                 p(3,i,j,k) = d3*c(i,j,k)
166: ||V----        enddo
167: ||                endif
168: |+-----         enddo
169: +------     enddo
```

図 4.3-20　IF 文をループの外に移動したコード

チューニング前後の各ケースについてSX-9 1CPUを用いて性能評価を行う. 問題サイズは「nx=256, ny=256, nz=128」としている. IF 文の条件式の真率による評価を行うため, チューニング前後で真率をそれぞれ以下に設定する.

- Res_vt : IF 文の条件式が全て TRUE の場合
- Res_vf : IF 文の条件式が全て FALSE の場合
- Res_25 : IF 文の条件式が真率 25%となる場合
- Res_50 : IF 文の条件式が真率 50%となる場合
- Res_75 : IF 文の条件式が真率 75%となる場合

```
!オリジナル
PROC.NAME  FREQUENCY  EXCLUSIVE       AVER.TIME     MOPS    MFLOPS V.OP  AVER.    VECTOR I-CACHE O-CACHE   BANK CONFLICT
                      TIME[sec](  %)    [msec]                    RATIO V.LEN     TIME    MISS    MISS CPU PORT  NETWORK

Res_vt           30    0.145( 11.8)     4.836  67836.6  36428.7 99.74 256.0    0.145   0.000   0.000    0.053    0.035
Res_vf           30    0.145( 11.8)     4.827  67955.3  36492.4 99.74 256.0    0.145   0.000   0.000    0.052    0.037
Res_25           30    0.163( 13.2)     5.421  60511.5  32495.0 99.74 256.0    0.163   0.000   0.000    0.067    0.054
Res_50           30    0.172( 13.9)     5.719  57358.6  30801.9 99.74 256.0    0.172   0.000   0.000    0.072    0.062
Res_75           30    0.163( 13.2)     5.427  60447.5  32460.7 99.74 256.0    0.163   0.000   0.000    0.068    0.049
```

図 4.3-21 チューニング前の FTRACE 情報

```
!チューニング
PROC.NAME  FREQUENCY  EXCLUSIVE       AVER.TIME     MOPS    MFLOPS V.OP  AVER.    VECTOR I-CACHE O-CACHE   BANK CONFLICT
                      TIME[sec](  %)    [msec]                    RATIO V.LEN     TIME    MISS    MISS CPU PORT  NETWORK

Res_vt           30    0.123( 13.9)     4.116  61321.9  36687.8 99.71 256.0    0.123   0.000   0.000    0.064    0.044
Res_vf           30    0.095( 10.6)     3.151  18804.9   7987.7 99.11 256.0    0.094   0.000   0.000    0.045    0.060
Res_25           30    0.105( 11.8)     3.514  30567.0  16091.8 99.46 256.0    0.105   0.000   0.000    0.050    0.057
Res_50           30    0.115( 13.0)     3.844  40563.3  22932.7 99.60 256.0    0.115   0.000   0.000    0.054    0.055
Res_75           30    0.120( 13.4)     3.983  51256.7  30021.8 99.67 256.0    0.119   0.000   0.000    0.059    0.049
```

図 4.3-22 チューニング後の FTRACE 情報

チューニング前は, 実行時間にわずかな差が出ているが, 全ての場合で全浮動小数点演算数(MFLOPS×時間)の変化がない. このことから, IF 文の条件式の真率に関係なく TRUE/FALSE 両方の演算がなされていることがわかる.

チューニング後は, 全浮動小数点演算数は IF 文の条件式の真率よって増減している. 演算密度の違いで MFLOPS 値が低下している場合もあるが, どの場合でも処理時間の短縮が確認できる.

(4) 事例2:IF 文を他の処理に置き換えるチューニング

図 4.3-23 の例は, 粒子コードに見られる周期的境界条件で位置情報の補正を行うコードである. DO ループ中に多数の IF 文があるため, 多くのマスク付きベクトル演算を行い, 演算時間が長くなっている. この根本的な解決として IF 文に代わる処理への置き換えを実施する.

```
!チューニング前
54: V------>      do n=1,nsp
55: |               if(pa(n,1) < 0.0d0) pa(n,1)=pa(n,1)+xdim
56: |               if(pa(n,1) >= xdim) pa(n,1)=pa(n,1)-xdim
57: |               if(pa(n,2) < 0.0d0) pa(n,2)=pa(n,2)+ydim
58: |               if(pa(n,2) >= ydim) pa(n,2)=pa(n,2)-ydim
59: |               if(pa(n,3) < 0.0d0) pa(n,3)=pa(n,3)+zdim
60: |               if(pa(n,3) >= zdim) pa(n,3)=pa(n,3)-zdim
61: V------       enddo
```

図 4.3-23 IF 文による周期的境界条件のコード

図 4.3-23 のコードは, 位置情報が各方向の最小値から最大値の範囲に収まっているかを, IF 文の

条件式で判定している. 収まっていない場合は, 位置情報に各辺の長さを加減して周期的境界条件を満たす補正がなされる.

図 4.3-24 にチューニング後のコードを示す. ここでは, mod 関数を使用して周期的境界条件を満たす同等な補正を行っている. 全要素一律に各辺の長さを加算し, その長さで割った余りを求めることで周期的境界条件が満たされ, IF 文による場合分けが不要となる. ただし, 周期内の条件にある位置は, mod 関数による補正計算が行われるため, チューニング前のコードと比べて若干の誤差が生じる. どちらのコードもループ長は同じである.

```
!チューニング後
54: V------>      do n=1,nsp
55: |               pa(n,1)=dmod(pa(n,1)+xdim,xdim)
56: |               pa(n,2)=dmod(pa(n,2)+ydim,ydim)
57: |               pa(n,3)=dmod(pa(n,3)+zdim,zdim)
58: V------       enddo
```

図 4.3-24　mod 関数による周期的境界条件のコード

図 4.3-23 と図 4.3-24 の各場合について SX-9 1CPU を用いて性能評価を行う. 問題サイズは「nx=256, ny=128, nz=64, ns=32」としている. 要素数 nsp は nx, ny, nz, ns の積となり, 各方向の最大値を実数型変数 xdim, ydim, zdim で与えている.

```
PROC.NAME  FREQUENCY  EXCLUSIVE       AVER.TIME     MOPS   MFLOPS V.OP  AVER.    VECTOR I-CACHE O-CACHE   BANK CONFLICT
                      TIME[sec](  %)    [msec]                   RATIO V.LEN     TIME    MISS    MISS CPU PORT  NETWORK

sample_f          30     9.557( 85.8)   318.582   6327.7   1895.8 99.87 256.0     9.557   0.000   0.000    0.021    8.658
```

図 4.3-25　チューニング前の FTRACE 情報

```
PROC.NAME  FREQUENCY  EXCLUSIVE       AVER.TIME     MOPS   MFLOPS V.OP  AVER.    VECTOR I-CACHE O-CACHE   BANK CONFLICT
                      TIME[sec](  %)    [msec]                   RATIO V.LEN     TIME    MISS    MISS CPU PORT  NETWORK

sample_f          30     1.720( 51.5)    57.345  52707.6  17554.0 99.91 256.0     1.720   0.000   0.000    0.004    0.029
```

図 4.3-26　チューニング後の FTRACE 情報

図 4.3-25 と図 4.3-26 がそれぞれチューニング前後の性能値で, ベクトル演算率やベクトル長に大きな差異は見られない. チューニング前は配列 pa のロード・ストア回数が多く, 条件分岐のオーバヘッドで待ち時間も発生し, 「BANK CONFLICT/NETWORK」が大きくなっている. チューニング後は mod 関数を用い IF 文の条件分岐が取り除かれたことで, ベクトル演算の密度が上がり MOPS および MFLOPS の値が向上している.

4.3.4 DO ループ内の作業配列の削除による高速化

(1) 作業配列の削除の概要

一般にプログラムは入力データと出力結果の他にも, 途中経過を作業配列に保存する場合がある. しかし, 不必要な作業配列の利用はメモリ負荷を増加させ, 演算性能を低下させることになる.

図 4.3-27 の例は, ループ内でのみ使用する途中の演算結果 a(n)+b(n)を作業配列 work(n)に保存している. この途中の演算結果をその後の処理で利用することがなければ, 作業配列 work(n)を使わずスカラ変数に書き換え, 作業配列への保存を回避できる. このようにすることで, 途中の演算結果は作業レジスタに保存され, メモリ負荷が下がり演算性能が向上する.

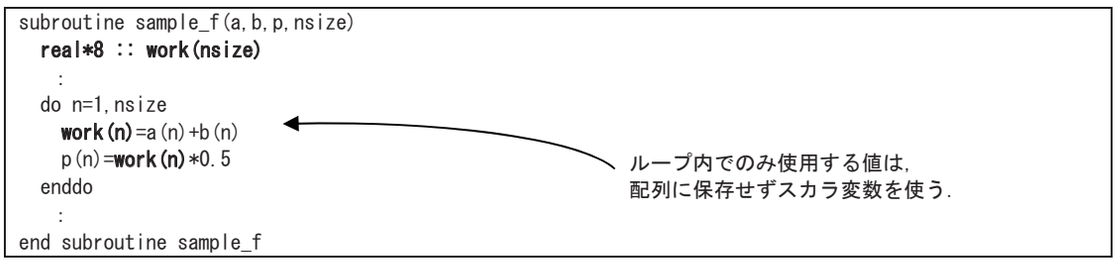


図 4.3-27 作業配列からスカラ変数に書き換えることができる例

(2) 作業配列を削減するチューニングの事例

図 4.3-28 のサブルーチン sample_f1, sample_f2 は, プログラム中の複数の箇所で同じ演算となる部分を括りだし, 共通部品としてサブルーチンにしたものである. このプログラムは, 3 次元配列のデータを 1 次元作業配列にコピーしサブルーチン sample_f1 へ渡し, サブルーチン sample_f1, sample_f2 で計算し, 1 次元作業配列の結果を 3 次元配列に書き戻している.

```
!オリジナル
 34: |+----->       do k=1,kmax
 35: ||+---->       do j=1,jmax
 36: |||
 37: |||V--->         do i=1,imax
 38: ||||              wi(1,i)=a(1,i,j,k)
 39: ||||              wi(2,i)=a(2,i,j,k)
 40: ||||              wi(3,i)=a(3,i,j,k)
 41: ||||              wv(i)  =b(i,j,k)
 42: |||V---           enddo
 43: |||
 44: |||               call sample_f1(imax,wi,c ,wt)
 45: |||               call sample_f2(imax,wv,wt,wo)
 46: |||
 47: |||V--->         do i=1,imax
 48: ||||              p(i,j,k)=wo(i)
 49: |||V---           enddo
 50: |||
 51: ||+----        enddo
 52: |+-----        enddo

 85:             subroutine sample_f1(lmax,wi,wc,wt)
 :
 97: V------>     do l=2,lmax-1
 98: |              wt(l)=wc(1)*(wi(1,l+1)-wi(1,l-1)) &
 99: |                   +wc(2)*(wi(2,l+1)-wi(2,l-1)) &
100: |                   +wc(3)*(wi(3,l+1)-wi(3,l-1))
101: V------       enddo

106:            subroutine sample_f2(lmax,wv,wt,wo)
 :
118: V------>     do l=2,lmax-1
119: |              wo(l)=wt(l)*wv(l)
120: V------       enddo
```

図 4.3-28 作業配列を削減前のコード

図 4.3-29 のチューニングは, 作業配列 wi,wv と元の入力配列 a,b の 1 列が同じ値を参照していることに着目する. この場合, 作業配列 wi,wv を使わないで配列 a,b の 1 列を直接引き渡すことができる. 同様に引数の作業配列 wo と出力配列 p の 1 列が同じ値を更新しているので, 出力データも作業配列 wo を使わないで配列 p へ直接出力できる. このようにサブルーチン呼び出し前後に作業配列 wi, wv, wo を使用しないでデータを引き渡すことができるため, 作業配列へのメモリアクセス回数が削減できる.

```
!チューニング①
 58: |+-----> do k=1,kmax
 59: ||+----> do j=1,jmax
 60: |||
 61: |||        call sample_f1(imax,a(1,1,j,k),c ,wt      )
 62: |||        call sample_f2(imax,b(1,j,k)  ,wt,p(1,j,k))
 63: |||
 64: ||+----   enddo
 65: |+-----   enddo

 85:          subroutine sample_f1(lmax,wi,wc,wt)
 :
 97: V------>   do l=2,lmax-1
 98: |            wt(l)=wc(1)*(wi(1,l+1)-wi(1,l-1)) &
 99: |                 +wc(2)*(wi(2,l+1)-wi(2,l-1)) &
100: |                 +wc(3)*(wi(3,l+1)-wi(3,l-1))
101: V------    enddo

106:         subroutine sample_f2(lmax,wv,wt,wo)
 :
118: V------>   do l=2,lmax-1
119: |            wo(l)=wt(l)*wv(l)
120: V------    enddo
```

図 4.3-29 ［第 1 段階］入力データと出力データを引数にする修正

図 4.3-30 の更なるチューニングは，サブルーチン sample_f1，sample_f2 を 1 つのサブルーチン sample_f にまとめ，これらのサブルーチンそれぞれにあった DO ループを融合している．このループ融合によって，sample_f1 から sample_f2 に受け渡している途中の結果を，作業配列 wt を使わずに演算処理できる．

```
!チューニング②
 71: |+-----> do k=1,kmax
 72: ||+----> do j=1,jmax
 73: |||
 74: |||        call sample_f(imax,a(1,1,j,k),b(1,j,k),c,p(1,j,k))
 75: |||
 76: ||+----   enddo
 77: |+-----   enddo

127:         subroutine sample_f(lmax,wi,wv,wc,wo)
 :
141: V------>   do l=2,lmax-1
142: |            s=wc(1)*(wi(1,l+1)-wi(1,l-1)) &
143: |             +wc(2)*(wi(2,l+1)-wi(2,l-1)) &
144: |             +wc(3)*(wi(3,l+1)-wi(3,l-1))
145: |            wo(l)=s*wv(l)
146: V------    enddo
```

図 4.3-30 ［第 2 段階］サブルーチンにまたがる 2 つの DO ループを融合する修正

(3) 性能評価

チューニング前後の各場合について SX-9 1CPU を用いて性能評価を行う．(2)で示したコードを用い，問題サイズは「imax=1000, jmax=50, kmax=50」としている．図 4.3-31 は，チューニング前後の性能値である．

- case_000：図 4.3-28 チューニング前の結果
- case_001：図 4.3-29 チューニング［第 1 段階］での結果

- case_002:図 4.3-30 チューニング[第 2 段階]での結果

PROC.NAME	FREQUENCY	EXCLUSIVE TIME[sec](%)	AVER.TIME [msec]	MOPS	MFLOPS	V.OP RATIO	AVER. V.LEN	VECTOR TIME	I-CACHE MISS	O-CACHE MISS	BANK CONFLICT CPU PORT	BANK CONFLICT NETWORK
case_000	1000	5.841(100.0)	5.841	12542.9	3844.5	98.83	249.7	5.840	0.000	0.000	1.862	4.740
case_001	1000	1.821(100.0)	1.821	26312.6	12331.1	98.93	249.5	1.820	0.000	0.000	0.520	1.145
case_002	1000	1.064(100.0)	1.064	40150.0	21097.7	99.26	249.5	1.064	0.000	0.000	0.516	0.423

図 4.3-31 FTRACE 情報一覧

case_000→case_001→case_002 とチューニングの段階を経て,「BANK CONFLICT/ NETWORK」の時間が短縮されている. これは作業配列へのデータ転送が減っているためである. データ転送が削減されメモリ負荷が下がったことで, 結果的に演算器の待ち時間が減少し, MFLOPS 値が向上している.

4.3.5 組み込み関数 MAX/MIN の利用による高速化

(1) 組み込み関数 MAX/MIN の利用による高速化の概要

最大値・最小値を求める計算はマクロ演算を利用してベクトル化され, 図 4.3-32 に示すようにコンパイル診断メッセージに「Macro operation Max/Min」と表示される. また, 4.1 節で述べたとおり, SX-9 から MAX/MIN 関数がハードウェア命令化され, 最大値・最小値を求める計算時間の短縮が行われる. 本節では, 最大値を求めるコードでベクトル化される場合とベクトル化されない場合を示し, さらにハードウェア命令が使われた場合と, 使われない場合の性能の違いを示す.

```
    61   vec   (  26): Macro operation Max/Min.

 :(※中略)

57: W------>      do k=1,kmax
58: |*----->        do j=1,jmax
59: ||*---->          do i=1,imax
60: |||                 tmp=r1(i,j,k)*r2(i,j,k)
61: |||                 s=max(s,tmp)
62: ||*----           enddo
63: |*-----         enddo
64: W------       enddo
```

行番号 61 番で最大値・最小値のマクロ演算を適用するメッセージ

図 4.3-32 MAX/MIN 関数利用の例とコンパイル診断メッセージ

(2) 最大値を求めるコードの事例

次に示す図は, 3 種類の最大値を求めるコードで, 以下の特徴がある.

- コード①(図 4.3-33)は, 組み込み関数 MAX を利用して最大値を求める
- コード②(図 4.3-34)は, 大小関係を IF 文で判断し, 大きい値で変数 s を更新し最大値を求める
- コード③(図 4.3-35)は, ループの 1 回目で初期値を設定し, 2 回目以降のループで最大値を求める

コード①は, 組み込み関数 MAX を利用しているので SX-9 のハードウェア命令が使われベクトル化される. コード②はコンパイラが最大値を求める処理と認識できるのでベクトル化される. コード③は条件分岐が複雑であり, コンパイラが最大値を求める処理と認識できないのでベクトル化されない.

```
!コード①
131:              subroutine sample_b1(r1,r2,s)
:
143:                s=-999d0
144: V------>       do i=1,imax*jmax*kmax
145: |                tmp=r1(i)*r2(i)
146: |                s=max(s,tmp)
147: V------        enddo
```

図 4.3-33　最大値を求めるコード①

```
!コード②
154:              subroutine sample_b2(r1,r2,s)
:
166:                s=-999d0
167: V------>       do i=1,imax*jmax*kmax
168: |                tmp=r1(i)*r2(i)
169: |                if(tmp .gt. s) s=tmp
170: V------        enddo
```

図 4.3-34　最大値を求めるコード②

```
!コード③
177:              subroutine sample_b3(r1,r2,s)
:
190:                lflag=.true.
191: +------>       do i=1,imax*jmax*kmax
192: |                tmp=r1(i)*r2(i)
193: |                if(lflag) then
194: |                  s=tmp
195: |                  lflag=.false.
196: |                else
197: |                  s=max(s,tmp)
198: |                endif
199: +------        enddo
```

図 4.3-35　最大値を求めるコード③

(3)　性能評価

各コードを SX-9 1CPU を用いて性能評価を行う. 問題サイズは「imax*jmax*kmax=256^3」で十分に長いループ長としている.

表 4.3-2 3 種類コードの性能

	性能値
コード①	20,999MFLOPS
コード②	11,373MFLOPS
コード③	93MFLOPS

コード①は MAX/MIN 関数でベクトル化され SX-9 で強化されたハードウェア命令が使われるため, 高い性能を示している. コード②はベクトル化されるが SX-9 のハードウェア命令が使われてないため, コード①と比較して半分程度の性能である. コード③はベクトル化されていないので性能が出ていない. SX-9 で最大値・最小値を求める場合, 高い性能を得るには MAX/MIN 関数を使用してベクトル化することが必要である.

4.3.6 科学技術計算ライブラリASLの利用による高速化

(1) 科学技術計算ライブラリ利用の概要

数値計算などで利用している連立 1 次方程式, 固有値・固有ベクトル, 高速フーリエ変換(FFT), 特殊関数, 乱数などの機能は計算ライブラリとして準備されている. このような計算ライブラリをユーザが作成することはできるが, ハードウェアの性能を引き出すために複雑な最適化が必要であり, 高い性能を得るのは容易ではない. また一般の計算ライブラリは, 特定のハードウェアに向けての最適化は通常行われておらず, SX-9 の最大性能を得ることは難しい. SX-9 には科学技術計算ライブラリ ASL があり, これは広範な分野の数値シミュレーションプログラムの作成を強力に支援する計算ライブラリである. ASL はハードウェアの性能を最大限に引き出す最適化が行われており, ユーザはこれらのライブラリを使うことにより, SX-9 の性能を引き出すことが容易に行える.

これから科学技術計算ライブラリ ASL の FFT とフリーソフトウェアの FFT ライブラリとの性能を比較する. 比較対象のフリーソフトウェアは, NETLIB の FFTPACK, および筑波大学高橋大介氏作成の FFTE 4.1 としている. それぞれの FFT ライブラリは, 以下の URL からダウンロードできる.

FFTPACK	http://www.netlib.org/fftpack/
FFTE	http://www.ffte.jp/

(2) ASL と FFTPACK の性能評価

ここでは1次元複素数フーリエ変換を ASL のサブルーチン JFCOSI/JFCOSF と FFTPACK のサブルーチン ZFFTI/ZFFTF/ZFFTB を用いて比較する. それぞれの評価コードを図 4.3-36 と図 4.3-37 に示す. これらサブルーチンは周期2πの1周期を n 等分した n 個の複素数データが与えられたとき, n 個のデータに対する離散フーリエ変換(順変換と逆変換)を行う.

```
! 1次元複素数 FFT (ASL) の評価コード
    integer, parameter :: ld=2**26
    complex(kind=8) :: c(ld),trigs((3*ld)/2),wc(ld)
    integer :: ifax(128)
    real(kind=8) :: t1,t2,tm,perf(-1:1)
    do ii=11,26
     n=2**ii
     do isw=1,-1,-2
       do i=1,n
        c(i) = dcmplx(cos(sqrt(2.d0)*i),sin(sqrt(2.d0)*i))
       enddo

       call JFCOSI(n,ifax,trigs,ierr)
       call clock(t1)
       call JFCOSF(n,c,ld,isw,ifax,trigs,wc,ierr)
       call clock(t2)
       tm=(t2-t1)
       perf(isw)=5.d0*n*log(dble(n))/log(2.d0)/tm*1.d-9

     enddo
     write(6,*) n,perf(1),perf(-1)
    enddo
    end
```

ASL の FFT サブルーチン

図 4.3-36 1 次元複素数 FFT(ASL)の評価コード

```
! 1 次元複素数 FFT (FFTPACK) の評価コード
    integer, parameter :: ld=2**26
    complex(kind=8) :: w(ld*2+15),cx(ld)
    real(kind=8) :: t1,t2,tm,perf(-1:1)
    do ii=11,26
     n=2**ii
     do isw=1,-1,-2
       do i=1,n
        cx(i) = dcmplx(cos(sqrt(2.d0)*i),sin(sqrt(2.d0)*i))
       enddo

       call ZFFTI (n,w)
       call clock(t1)
       if(isw.eq.1)then
        call ZFFTF (n,cx,w)
       else
        call ZFFTB (n,cx,w)
       endif
       call clock(t2)
       tm=(t2-t1)
       perf(isw)=5.d0*n*log(dble(n))/log(2.d0)/tm*1.d-9

     enddo
     write(6,*) n,perf(1),perf(-1)
    enddo
    end
```

FFTPACK のサブルーチン

図 4.3-37　1 次元複素数 FFT（FFTPACK）の評価コード

ここで性能測定プログラムに使う評価コードについて説明する. この評価コードでは, 必要なデータ領域を確保し, FFT の入力データおよびワーク領域の初期化を行って1次元複素数フーリエ変換のサブルーチンを呼び出している. 実行時間の計測は, フーリエ変換のサブルーチンを呼び出す前後で, タイマールーチン（CLOCK）を挟んで行っている.

つぎに性能の算出方法について説明する. FFT の代表的な計算方法である Cooley-Tukey のアルゴリズムからデータ数 n の1次元複素数 FFT の演算数を数え上げると, 次式で表わされる.

$$演算数＝5n\times\log_2(n)$$

この Cooley-Tukey のアルゴリズムの演算数は基数が 2 の場合の結果であるが, 基数が 2 以外でも大きく違わないため, 演算数を求めるには良い近似値となる. この方法で FFT の演算数を求め, 実行時間で除する事で性能(GFLOPS 値)を算出している.

図 4.3-38 は, データ数を 2^{11} (=2,048) から 2^{26}(=67,108,864)まで 2 のべき乗で増加させた時の FFT の性能である. 図中, fwd は順変換を bwd は逆変換を意味している.

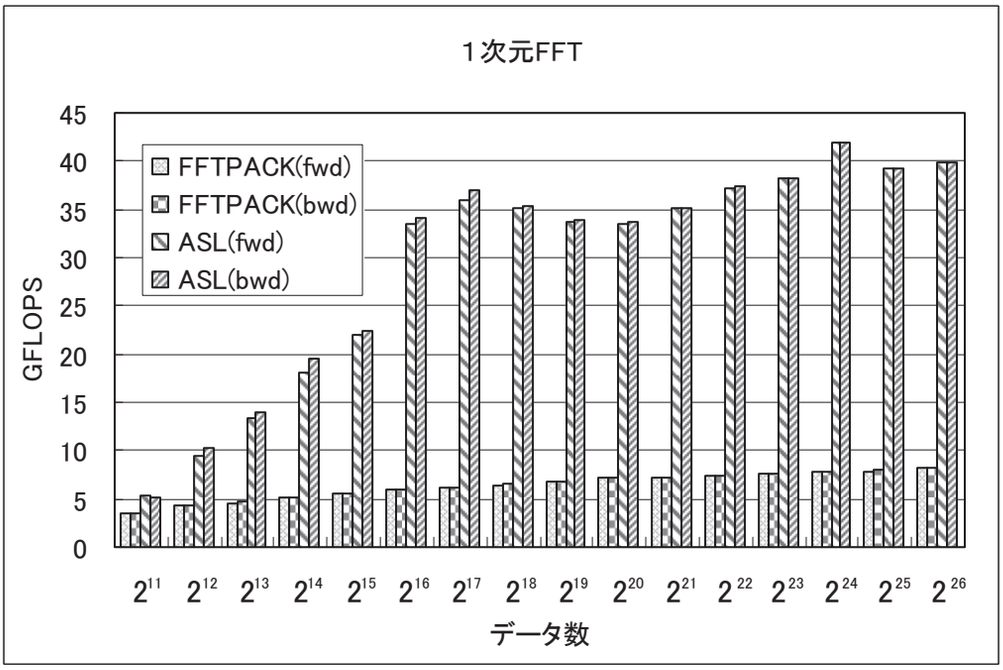


図 4.3-38 1 次元複素数 FFT の性能比較

FFTPACK は，データ数を増加しても大きな性能向上が得られない． ASL は，データ数を増加すると 2^{17}(=131,072)で 36GFLOPS と性能が向上し，最大性能は 2^{24}(=16,777,216)で 41GFLOPS となる．ASL と FFTPACK の性能を比較すると，データ数が大きいところで約 5 倍 ASL が高速である．

(3) ASL と FFTE の性能評価

ここでは2次元複素数フーリエ変換を ASL のサブルーチン ZFC2FB/ZFC2BF と，FFTE はベクトル用のサブルーチン ZFFT2D を用いて行っている(注 1)．それぞれの評価コードを，図 4.3-39 と図 4.3-40 に示す．これらのサブルーチンは，2次元の各次元のデータで離散フーリエ変換(順変換と逆変換)を行う．

```
! ２次元複素数 FFT （ASL） の評価コード
    integer, parameter :: ld=10241
    complex(kind=8) :: c(ld*ld),trigs(2*ld),wc(ld*ld)
    integer :: ifax(40)
    real(kind=8):: t1,t2,tm,perf(-1:1)
    do ii=1024,10240,1024
    do isw=1,-1,-2
      nx=ii
      ny=ii
      lx=nx+1
      ly=ny+1
      call init(c,lx,ly,nx,ny)

      call ZFC2FB(nx,ny,c,lx,ly,0,ifax,trigs,wc,ierr)
      call clock(t1)
      call ZFC2BF(nx,ny,c,lx,ly,isw,ifax,trigs,wc,ierr)
      call clock(t2)
      tm=(t2-t1)
      perf(isw)=5.d0*nx*ny*(log(dble(nx))+log(dble(ny)))/log(2.d0)/tm*1.d-9

     enddo
     write(6,*) ii,perf(1),perf(-1)
    enddo
    end
    subroutine init(c,lx,ly,nx,ny)
    complex(kind=8) :: c(lx,ly)
    do j=1,ny
     do i=1,nx
      c(i,j) = dcmplx(cos(sqrt(2.d0)*i),sin(sqrt(2.d0)*j))
     enddo
    enddo
    end
```

ASL の FFT サブルーチン

図 4.3-39　2 次元複素数 FFT(ASL)の評価コード

```
! ２次元複素数 FFT （FFTE） の評価コード
    integer, parameter :: ld=10240
    complex(kind=8) :: a(ld*ld)
    real(kind=8) :: t1,t2,tm,perf(-1:1)
    do ii=1024,10240,1024
     if(mod(ii,7).eq.0) cycle
     do isw=1,-1,-2
      nx=ii
      ny=ii
      call init(a,nx,nx,nx,ny)

      call ZFFT2D(a,nx,ny,0)
      call clock(t1)
      call ZFFT2D(a,nx,ny,-isw)
      call clock(t2)
      tm=(t2-t1)
      perf(isw)=5.d0*nx*ny*(log(dble(nx))+log(dble(ny)))/log(2.d0)/tm*1.d-9

     enddo
     write(6,*) ii,perf(1),perf(-1)
    enddo
    end
    subroutine init(c,lx,ly,nx,ny)
    complex(kind=8) :: c(lx,ly)
    do j=1,ny
     do i=1,nx
      c(i,j) = dcmplx(cos(sqrt(2.d0)*i),sin(sqrt(2.d0)*j))
     enddo
    enddo
    end
```

FFTE のサブルーチン

図 4.3-40　2 次元複素数 FFT(FFTE)の評価コード

図 4.3-41 は 2 次元データの各次元とも 1,024 の増分で 1,024×1,024 から 10,240×10,240 までデータ数を増加させた時の FFT の性能である. ただし, FFTE はデータ数に 2, 3, 5 以外の約数が入ると計算できないため, 約数に 7 が入るデータ数は除外している(注 2). 実行時間の計測は, (2)と同様, フーリエ変換のサブルーチンを呼び出す前後で行い, タイマールーチンを呼ぶ前に FFT の入力データとワーク領域の初期化を行っている. 演算数は1次元の FFT での算出方法を2次元の FFT に拡張し, 性能を求めている.

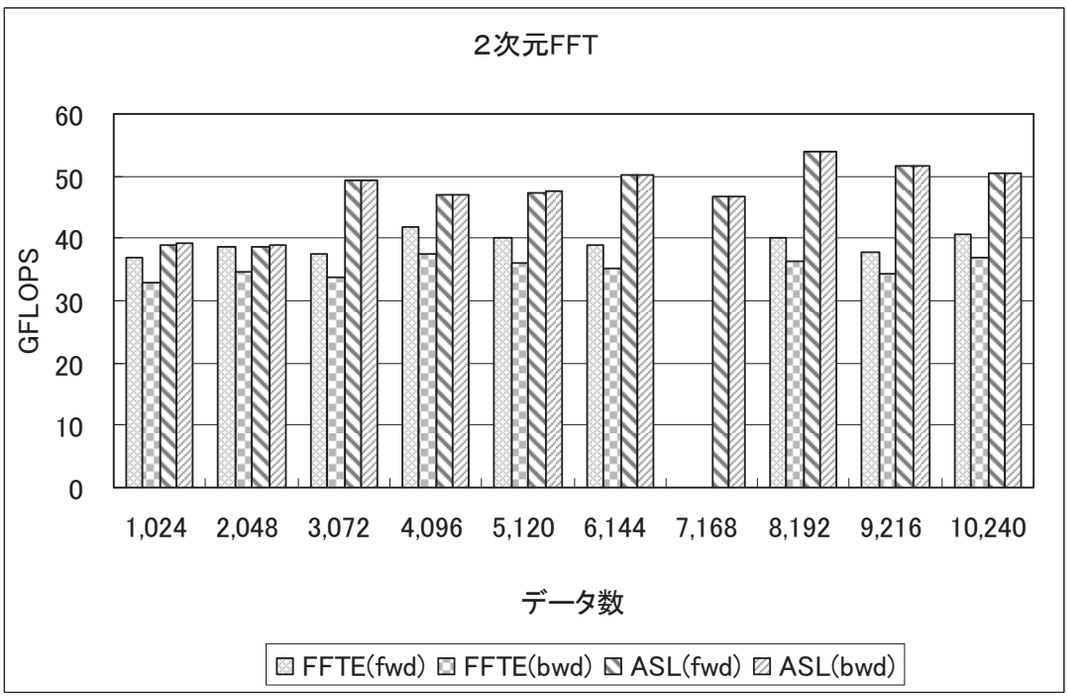


図 4.3-41 2 次元複素数 FFT の性能比較

FFTE はデータ数により 32～42GFLOPS の性能である. ASL はデータ数が 8,192×8,192 のとき, 最大性能の 54GFLOPS となる. ASL と FFTE の性能を比較すると 1.1～1.4 倍 ASL が高速である.

(注 1) FFTE は, ベクトル用およびスカラ用に最適化された FFT のサブルーチンが用意されている. それぞれのプラットフォームに適しているサブルーチンを利用すること.

(注 2) FFTE は, データ数に 2,3,5 以外の約数が入っていると FFT の計算ができない仕様である. 例えば 7,168 は約数に 7 が入り計算できない. 一方 ASL の FFT は, データ数に約数の制約はなく任意のデータ数で FFT の計算ができる.

(4) 性能評価のまとめ

科学技術計算ライブラリ ASL の FFT サブルーチンとフリーソフトウェアの FFT ライブラリである FFTPACK, FFTE との間で性能を比較している. その結果, ASL の FFT サブルーチンが高い性能を示している. FFTE にはベクトル用に最適化されている FFT サブルーチンが用意されており, 実行効率が 30%～40%と比較的よい性能が得られているが, ASL の FFT サブルーチンは実行効率が最大で 50%を超え, より高速である.

SX で FFT の計算を行う場合は, SX のハードウェア向けに高度な最適化が施されている科学技術計

算ライブラリASLを使うことで，フリーソフトウェアのFFTライブラリを上回る性能が引き出せることがわかる．

4.3.7 リストベクトルを含む DO ループの最適化

(1) リストベクトルの概要

リストベクトルとは，図 4.3-42 に示すように間接参照される配列 IX にしたがって，配列 A の参照または更新をベクトル処理で行うものである．図 4.3-43 に図 4.3-42 に示したコードのメモリアクセスパターンを示す．リストベクトルを用いることで，連続でないランダムなメモリアクセスをベクトル処理として行うことができる．リストベクトルはDOループ中のIF文の除去や，ベクトル化できないデータ参照関係を回避する平面法などに活用されている．平面法については，「高速化活動報告第 4 号」を参照のこと．

```
!参照
do  i=1,n
  T(i)=A(IX(i))
enddo

!更新
do  i=1,n
  A(IX(i))=T(i)
enddo
```

図 4.3-42 リストベクトルのコード

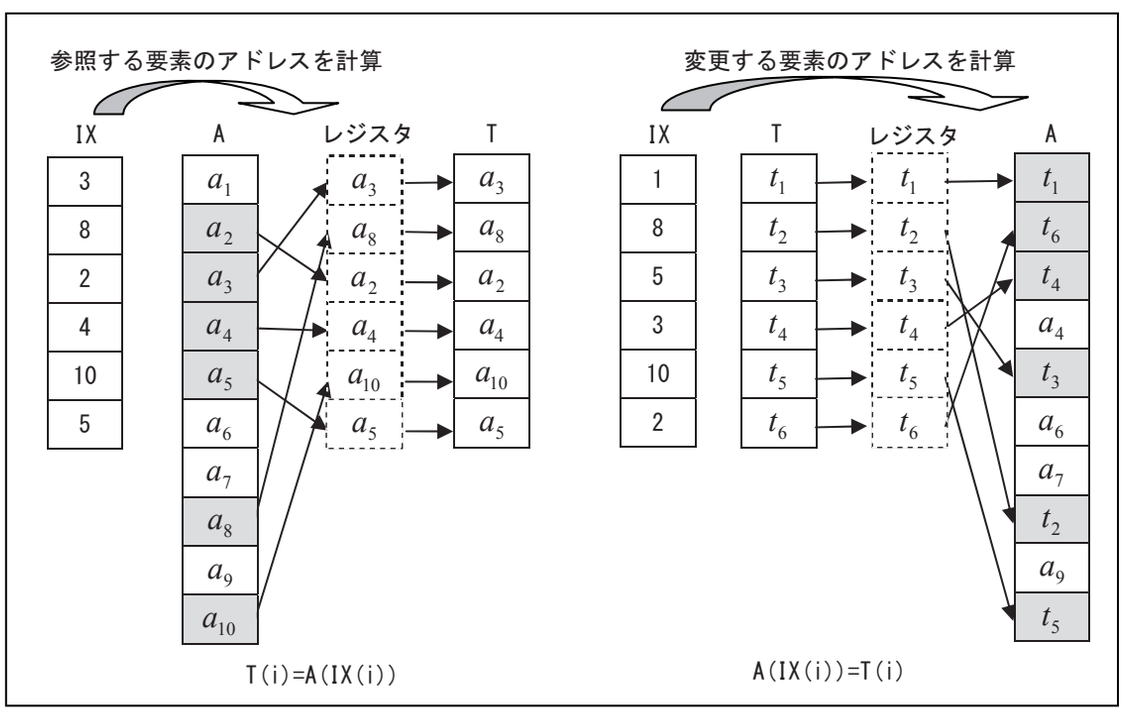


図 4.3-43 リストベクトルのメモリアクセス

(2) 事例1:リスト参照のメモリレイテンシの隠ぺい

リストベクトルによる参照や更新が同一 DO ループ内に複数回現れるプログラムの高速化の手法について説明する．

図 4.3-44 にリストベクトルによる参照や更新が同一 DO ループ内に複数回現れるコード(注)を示す．

間接参照 n2 による配列 p の参照は，間接参照 n1 による同配列 p への更新の完了を待つためメモリレイテンシ（遅延）が表面化し，演算性能を発揮できない．そこで配列 p のリストベクトルによる参照と更新のタイミングをずらすため，一度別の作業配列に連続データとして保存する．

```
121: +------>       do icol=1,ncol
122: |              !cdir nodep
123: |V----->         do edge=esep(icol),esep(icol+1)-1
124: ||                 n1=e2n(edge,1)
125: ||                 n2=e2n(edge,2)
 :
141: ||                 x1=u(n1,1)
142: ||                 y1=u(n1,2)
143: ||                 x2=u(n2,1)
144: ||                 y2=u(n2,2)
 :
161: ||                 !userfunc はインライン化によりベクトル化されるユーザ定義関数
162: ||                 v=userfunc(x1,y1,z1,x2)
 :
191: ||                 p(n1)=p(n1)+v
192: ||                 p(n2)=p(n2)-v
 :
209: |V-----          enddo
210: +------        enddo
```

図 4.3-44 リストベクトルが同一 DO ループに複数回現れるコード

図 4.3-45 に作業配列を使って最適化するコードを示す．先ず配列 p を間接参照 n1,n2 により参照し，それぞれの作業配列 pn1,pn2 に連続データとして保存する．次に作業配列 pn1,pn2 を使って連続アクセスのデータで演算する．最後に演算結果を，作業配列から元の配列 p へ書き戻している．このようにすることで，配列 p の間接参照による参照を更新の前にまとめて行い，更新完了の待ちによる間接参照時のメモリレイテンシの影響を少なくできる．

```
121: +------>       do icol=1,ncol
122: |              ! リストベクトルによる参照処理の集約
123: |V----->         do edge=egsep(icol),egsep(icol+1)-1
124: ||                 n1=e2n(edge,1)
125: ||                 n2=e2n(edge,2)
 :
141: ||                 x1_a(edge)=u(n1,1)
142: ||                 y1_a(edge)=u(n1,2)
143: ||                 x2_a(edge)=u(n2,1)
144: ||                 y2_a(edge)=u(n2,2)
 :
161: ||                 pn1(edge)=p(n1)
162: ||                 pn2(edge)=p(n2)
 :
179: |V-----          enddo
180: |              ! 連続ベクトルによる演算
181: |V----->         do edge=egsep(icol),egsep(icol+1)-1
182: ||                 v=userfunc(x1_a(edge),y1_a(edge),x2_a(edge),y2_a(edge))
 :
211: ||                 pn1(edge)=pn1(edge)+v
212: ||                 pn2(edge)=pn2(edge)-v
 :
229: |V-----          enddo
230: |              ! リストベクトルによる更新処理の集約
231: |              !cdir nodep
232: |V----->         do edge=egsep(icol),egsep(icol+1)-1
233: ||                 n1=e2n(edge,1)
234: ||                 n2=e2n(edge,2)
 :
250: ||                 p(n1)=pn1(edge)
251: ||                 p(n2)=pn2(edge)
 :
268: |V-----          enddo
269: +------       enddo
```

図 4.3-45　作業配列を使った演算コード

チューニング前後のコードを使って，SX-9 1CPU で性能評価を行う．図 4.3-46 に FTRACE の情報を示す．org がチューニング前のコード，tune がチューニング後のコードを表している．配列 p の間接参照を更新より先に一括して行っていることで，メモリレイテンシを隠ぺいでき，図 4.3-46 で示されるようにメモリネットワーク競合時間（NETWORK）が半減し，CPU 時間比にして 1.95 倍に性能が向上している．

PROC.NAME	FREQUENCY	EXCLUSIVE TIME[sec](%)	AVER.TIME [msec]	MOPS	MFLOPS	V.OP RATIO	AVER. V.LEN	VECTOR TIME	I-CACHE MISS	O-CACHE MISS	BANK CONFLICT CPU PORT	BANK CONFLICT NETWORK
org	100	24.849(12.8)	248.493	6508.8	1817.5	99.95	250.1	24.843	0.001	0.001	2.039	18.972
tune	100	12.707(6.5)	127.074	16339.5	3554.2	99.80	250.1	12.640	0.016	0.012	2.301	8.792

図 4.3-46　チューニング前後の FTRACE 情報

（注）本コードは，間接参照 n1, n2 がそれぞれ最内の DO ループの中で重複する値を取らないことと，間接参照 n1 と n2 の間にも値の重複がないことを想定している．いずれも重複がある場合，ベクトル化によりデータの不整合が起こる．コンパイラはデータ依存性を判断できないため，図 4.3-44 の 122 行目および図 4.3-45 の 231 行目に指示行「!cdir nodep」を追加し，コンパイラに重複がないことを指示することでベクトル化を行っている．

(3) 事例2:配列の次元の入れ替えによるリストベクトルの回避

リストベクトル処理はメモリアクセスのレイテンシを隠ぺいすることが難しく，メモリを連続にアクセスする処理と比べて，処理時間を多く必要とし実効性能が低下する．

図 4.3-47 にリストベクトルのコードを示す．最内の i の DO ループの配列 ain はリスト il2，il1，ir1，ir2 により間接参照され，リストベクトルとなっている．

```
68: +------>            do k=1,kmax
69: |+----->              do j=1,jmax
70: ||V---->                do i=2,imax
71: |||                       il2=ilir(i,1)
72: |||                       il1=ilir(i,2)
73: |||                       ir1=ilir(i,3)
74: |||                       ir2=ilir(i,4)
75: |||                       stbc=wstbc(i)
76: |||                       dqr0=ain(i,j,k)
77: |||                       dqr1=ain(ir1,j,k)
78: |||                       dqr2=ain(ir2,j,k)
79: |||                       dql0=ain(i-1,j,k)
80: |||                       dql1=ain(il1,j,k)
81: |||                       dql2=ain(il2,j,k)
```

図 4.3-47 リストベクトル処理のコード

配列 ain に対して，i,j,k の次元から j,k,i の次元へ入れ替えを行うとともに，i の DO ループを最外に移動させる．図 4.3-48 に配列の次元を入れ替えたコードを示す．この変更により，配列 ain の参照は j,k の DO ループに対し連続ベクトルとなり，リストベクトルが回避される．加えてこの変更により，添え字 j が配列の下限から上限までの値を取ることで j,k の DO ループが一重化でき，ベクトル長が長くなる（注）．

```
68: V----->            do i=2,imax
69: |                    il2=ilir(i,1)
70: |                    il1=ilir(i,2)
71: |                    ir1=ilir(i,3)
72: |                    ir2=ilir(i,4)
73: |                    stbc=wstbc(i)
74: |W---->              do k=1,kmax
75: ||*--->                do j=1,jmax
76: |||                      dqr0=ain(j,k,i)
77: |||                      dqr1=ain(j,k,ir1)
78: |||                      dqr2=ain(j,k,ir2)
79: |||                      dql0=ain(j,k,i-1)
80: |||                      dql1=ain(j,k,il1)
81: |||                      dql2=ain(j,k,il2)
```

図 4.3-48 配列の次元を入れ替えたコード

チューニング前後のコードを使って，SX-9 1CPU で性能評価を行う．図 4.3-49 に FTRACE の情報を示す． org がリストベクトル処理主体のコード，tune が配列の次元を入れ替えたコードを表している．リストベクトルの回避とベクトル長が長くなることで，メモリネットワーク競合時間が大幅に削減され，CPU 時間比にして 11.41 倍に性能が向上している．

PROC.NAME	FREQUENCY	EXCLUSIVE TIME[sec](%)	AVER.TIME [msec]	MOPS	MFLOPS	V.OP RATIO	AVER. V.LEN	VECTOR TIME	I-CACHE MISS	O-CACHE MISS	BANK CONFLICT CPU PORT	BANK CONFLICT NETWORK
org	1	33.964(30.1)	33963.691	7920.3	1748.9	99.55	99.0	33.964	0.000	0.000	1.875	29.967
tune	1	2.977(2.6)	2976.662	46796.1	19955.2	99.50	250.0	2.976	0.000	0.000	0.010	1.418

図 4.3-49 チューニング前後の FTRACE 情報

(注)図 4.3-48 のコードにおける j,k の DO ループは，ループ長 jmax×kmax の一重ループとして処理される．例えば ain(j,k,i)への参照は，1～jmax×kmax の値を取るループ変数 jk を用いることで ain(jk,1,i)への参照と同様に処理される．

(4)　事例3：IF 文の変更によるリストベクトルの回避

図 4.3-50 のコードは，22 行目の IF 文で決定される変数 j を用いて配列 c を間接参照しているため，リストベクトルとなる(注)．この場合，IF 文の処理内容を変更することでリストベクトルを回避できる．

```
21: V------>        do i=1,n
22: |                 if ( x(i) .ge. 0.0 ) then
23: |                   j=2
24: |                 else
25: |                   j=1
26: |                 endif
27: |                 z1(i)=z1(i)+y1(i)*c(1,j)
28: |                 z2(i)=z2(i)+y2(i)*c(2,j)
29: V------         enddo
```

図 4.3-50　IF 文で決定した変数を用いて間接参照を行うコード

図 4.3-51 にリストベクトルの回避コードを示す．22 行目以降の IF 文を配列 c の値を直接参照し，新たに導入した作業変数 c1,c2 に格納する．これらの作業変数 c1,c2 を用いた 29 行目以降の演算は連続ベクトルとなるため，リストベクトルが回避される．加えて，IF 文の処理は配列 c の値を定数としてレジスタ上で利用できるため，メモリアクセスも大幅に低減される．

```
21: V------>        do i=1,n
22: |                 if ( x(i) .ge. 0.0 ) then
23: |                   c1=c(1,2)
24: |                   c2=c(2,2)
25: |                 else
26: |                   c1=c(1,1)
27: |                   c2=c(2,1)
28: |                 endif
29: |                 z1(i)=z1(i)+y1(i)*c1
30: |                 z2(i)=z2(i)+y2(i)*c2
31: V------         enddo
```

図 4.3-51　IF 文内で配列を直接参照するコード

チューニング前後のコードを使って，SX-9 1CPU で性能評価を行う．図 4.3-52 に FTRACE の情報を示す．org がリストベクトルのコード，tune がリストベクトル回避コードを表している．リストベクトルの回避とメモリアクセスの低減によりメモリネットワーク競合時間が大幅に削減され，CPU 時間比にして 9.90 倍に性能が向上している．

PROC.NAME	FREQUENCY	EXCLUSIVE TIME[sec](%)	AVER.TIME [msec]	MOPS	MFLOPS	V.OP RATIO	AVER. V.LEN	VECTOR TIME	I-CACHE MISS	O-CACHE MISS	BANK CONFLICT CPU PORT	BANK CONFLICT NETWORK
org	1	22.664(27.6)	22663.970	8405.8	1764.9	99.73	255.8	22.664	0.000	0.000	1.275	20.002
tune	1	2.287(2.8)	2286.695	70192.4	17492.5	99.68	255.8	2.287	0.000	0.000	0.023	0.693

図 4.3-52　チューニング前後の FTRACE 情報

(注)ループ変数 i の値それぞれに応じた変数 j の値は，ベクトルレジスタもしくはコンパイラが自動生成する作業配列に，一度ベクトルデータとして格納される．そのため j による配列 c の間接参照は，この格納されたベクトルデータを元にリストベクトルとして処理される．

4.3.8 SOR 法のベクトル化（Red Black 法）

(1) SOR 法の概要

SOR 法（Successive Over-Relaxation method）とは，n 元連立方程式 Ax=b を反復法で解く手法の一つである．図 4.3-53 に示すとおり，ある要素 p(i,j,k)の計算には近傍の要素 p(i-1,j,k)，p(i+1,j,k)，p(i,j-1,k)，p(i,j+1,k)，p(i,j,k-1)，p(i,j,k+1)が必要な計算を行う．

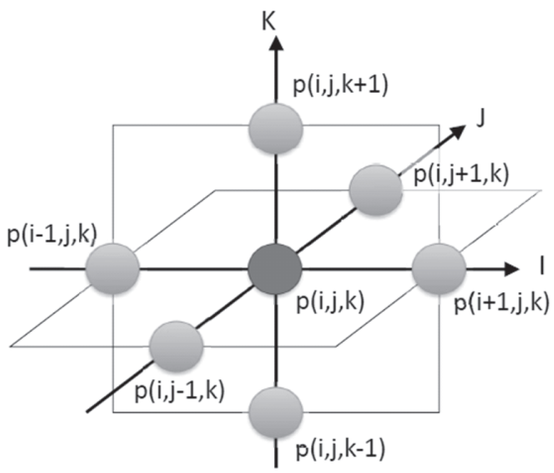


図 4.3-53 SOR 法計算の関係図

図 4.3-54 に SOR 法のコードを示す．p(i,j,k)の値を計算する際に，p(i-1,j,k)の要素を参照する必要がある．ベクトル化の対象となる最内の DO ループでは，p(i-1,j,k)の値は 1 回前で定義されており（注 1），この定義・参照関係がベクトル化を阻害する依存関係になる．そのためベクトル化できず（注 2），SX-9 で高い実効性能を得ることができない．

（注 1）p(i+1,j,k)は参照された後で値が更新されるので，ベクトル化を阻害する要因にはならない．

（注 2）図 4.3-54 の例のように，コンパイラはベクトル化可能な部分のみベクトル化を行う．これは部分ベクトル化と呼ばれ編集リストでは「S」で示される．

```
115  +-----> do k = n_bnd + 1, n_bnd + n_cell
116  |+----->  do j = n_bnd + 1, n_bnd + n_cell
117  ||V--->   do i = n_bnd + 1, n_bnd + n_cell
 :
141  |||  S      p_a = c0
142  |||  S  .   * ( p(i-1,j  ,k  ) * xm + p(i+1,j  ,k  ) * xp
143  |||  S  .     + p(i  ,j-1,k  ) * ym + p(i  ,j+1,k  ) * yp
144  |||  S  .     + p(i  ,j  ,k-1) * zm + p(i  ,j  ,k+1) * zp
145  |||  S  .     - rhs * ds * ds )
 :
148  |||  S      p(i,j,k) = p(i,j,k) + cnst_p * ( p_a - p0 )
149  ||V---     end do
150  |+----    end do
151  +-----   end do
```

図 4.3-54 SOR 法のコード

(2) Red Black 法

定義・参照の依存関係が存在することでベクトル化ができない DO ループに対して，依存関係が生じない順序で配列要素をアクセスすることによりベクトル化を可能にする手法の一つに Red Black 法が

ある. 図 4.3-55 に 3 次元の Red Black 法の概念図を示す. 配列要素を赤・黒と順番に色付けを行い, 赤の要素の計算と黒の要素の計算を行うループを分けることにより, ベクトル化を阻害する依存関係を排除できる.

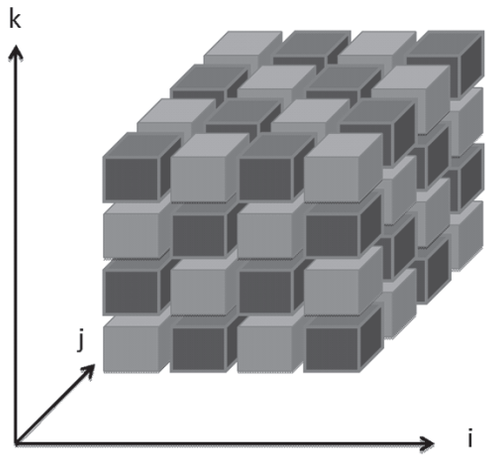


図 4.3-55 Red Black 法の概念図

3 次元の場合, 配列のアクセスは以下のように行われる(各次元のサイズが 8 の場合).

赤のループ:p(1,1,1),p(3,1,1),…,p(7,1,1),p(2,2,1),p(4,2,1),….

黒のループ:p(2,1,1),p(4,1,1),…,p(8,1,1),p(1,2,1),p(3,2,1),….

図 4.3-56 に Red Black 法のコードを示す. 最内 DO ループは上記に示した配列のアクセスのとおり増分 2 で繰り返される. 先頭要素から始まるループが赤のループ, 後続の先頭要素+1 から始まるループが黒のループに該当する. j, k の DO ループも増分 2 のアクセスになるように IF 文で処理を分岐している.

```
113 +-----> 　　do k = n_bnd + 1, n_bnd + n_cell
114 |+-----> 　　do j = n_bnd + 1, n_bnd + n_cell
115 ||　　　　　if(mod(k,2).eq.0.and.mod(j,2).eq.0.or.
116 ||　　+　　mod(k,2).eq.1.and.mod(j,2).eq.1) then
117 ||V---> 　　do i = n_bnd + 1, n_bnd + n_cell,2
 :
133 |||　　　　　p_a = c0
134 |||　　.　　* ( p(i-1,j  ,k  ) * xm + p(i+1,j  ,k  ) * xp
135 |||　　.　　  + p(i  ,j-1,k  ) * ym + p(i  ,j+1,k  ) * yp
136 |||　　.　　  + p(i  ,j  ,k-1) * zm + p(i  ,j  ,k+1) * zp
137 |||　　.　　  - rhs * ds * ds )
138 |||　　　　　p(i,j,k) = p(i,j,k) + cnst_p * ( p_a - p0 )
139 ||V---　　　end do
140 ||　　　　　else if(mod(k,2).eq.1.and.mod(j,2).eq.0.or.
141 ||　　+　　mod(k,2).eq.0.and.mod(j,2).eq.1) then
142 ||V---> 　　do i = n_bnd + 2, n_bnd + n_cell,2
 :
```

赤のループの実行

黒のループの実行

図 4.3-56 Red Black 法のコード

(3)　DO ループの一重化

図 4.3-56 に示したコードにおいて, 最内の DO ループがベクトル化の対象になるため, ベクトル長は

i の DO ループの半分のループ長になる(増分 2 でアクセスする)ため，十分なベクトル長が得られない．そこで，2 つのループを含めて 3 重ループを一重化することで，ベクトル長の拡大を図る．この例では，各次元に計算を行わない袖要素を含んでいるため，単純にDOループの一重化ができない．そこで予め計算を行う要素の場合を「1」，行わない要素の場合を「0」とするマスクテーブルを用意しておき，IF 文の判定で真になる場合のみ計算を実行するようにする．図 4.3-57 にDOループの一重化を行うコードを示す．コンパイラが配列pの依存関係を解析できないので，コンパイラ指示行NODEPを指定して，配列 p のアクセスに重なりがないことを明示する．

```
277 +-----> do icolor = 1, 2
278 |     !cdir nodep(p)
279 |V----> do ijk=icolor,max_cell*max_cell*max_cell,2
280 ||       if(matbl(ijk,1,1,icolor).eq.1.or.
281 ||    +     matbl(ijk,1,1,icolor+2).eq.1 ) then
 :
297 ||         p_a = c0
298 ||    .    * ( p(ijk-1,                1,1) * xm + p(ijk+1,                1,1) * xp
299 ||    .      + p(ijk-max_cell,         1,1) * ym + p(ijk+max_cell,         1,1) * yp
300 ||    .      + p(ijk-max_cell*max_cell,1,1) * zm + p(ijk+max_cell*max_cell,1,1) * zp
301 ||    .      - rhs * ds * ds )
 :
304 ||         p(ijk,1,1) = p(ijk,1,1) + cnst_p * ( p_a - p0 )
305 ||       endif
306 |V----   end do
307 +-----   end do
```

マスクテーブル配列 matbl の値が真の場合，ループの処理が実行される

図 4.3-57 マスクテーブルを使用するコード

(4) ADB の利用

SOR 法の計算は，4.3.8(1)で示しているように，ある要素の計算に近傍の 6 つの要素を使用するステンシル計算である．各要素は異なる繰り返しにおける計算で再び参照される．ON_ADB 指示行で，これらの要素を格納する配列 p を ADB に乗せることを指示する．これにより，1 回目のロード命令はメモリから行われるが，2 回目以降のロード命令は ADB から行われることになる．この結果，メモリのアクセス回数を削減することができ，処理効率の向上が期待される．図 4.3-58 に ON_ADB 指示行を指定したコードを示す．

```
277 +-----> do icolor = 1, 2
278 |     !cdir nodep(p)
279 |     !cdir on_adb(p)
280 |V----> do ijk=icolor,max_cell*max_cell*max_cell,2
281 ||       if(matbl(ijk,1,1,icolor).eq.1.or.
282 ||    +     matbl(ijk,1,1,icolor+2).eq.1 ) then
 :
298 ||         p_a = c0
299 ||    .    * ( p(ijk-1,                1,1) * xm + p(ijk+1,                1,1) * xp
300 ||    .      + p(ijk-max_cell,         1,1) * ym + p(ijk+max_cell,         1,1) * yp
301 ||    .      + p(ijk-max_cell*max_cell,1,1) * zm + p(ijk+max_cell*max_cell,1,1) * zp
302 ||    .      - rhs * ds * ds )
 :
305 ||         p(ijk,1,1) = p(ijk,1,1) + cnst_p * ( p_a - p0 )
306 ||       endif
307 |V----   end do
308 +-----   end do
```

図 4.3-58 ON_ADB 指示行の挿入

(5) 性能評価

SX-9 1CPU を用いて性能評価を行う. 評価を行うプログラムは以下のとおりである.

① SOR 法を最適化行わず記述したプログラム

② Red Black 法を用いてベクトル化を行ったプログラム

③ ②に対してマスクテーブルを用いてループの一重化を行ったプログラム

④ ③に対して再利用性の高い配列 p を ADB に乗せることを指示したプログラム

表 4.3-3 に, 各次元のループ長が 64 の場合の性能特性を示す.

表 4.3-3 性能特性

プログラム	ベクトル演算率(%)	平均ベクトル長	実効性能(MFLOPS)
①	51.86	64.0	189.5
②	90.11	54.4	970.1
③	99.59	255.9	8,247.3
④	99.59	255.9	18,940.6

①のプログラムは, ベクトル化できない DO ループ構造であるため, コンパイラは部分的にしかベクトル化を行わず, ベクトル演算率は 51.9%となり 200MFLOPS に満たない性能値となる. ②の 3 次元ループのままの Red Black 法のプログラムでは, ベクトル化は行えるものの, 平均ベクトル長が短いことと, ベクトルループを外側で何度も繰り返すため, 高い実効性能が得られない. ③の 3 次元ループに対してマスクテーブルを用いて一重化したプログラムでは, 平均ベクトル長が 255.9 となり, 8.3GFLOPS の性能値となる. さらに④の ADB に再利用性の高い配列要素を乗せるプログラムでは, メモリへのアクセス回数を削減することができ, ③と比較して 2 倍近い性能向上が得られる.

図 4.3-59 に, それぞれのプログラムで次元サイズを変更した場合の性能グラフを示す. すべてのケースにおいて, ④が最も高い性能であることが分かる. ベクトル型プロセッサの SX-9 では高いベクトル演算率, 十分な平均ベクトル長だけでなく, メモリ負荷を軽減する最適化が有効であることが示される. 特に再利用性の高い要素を ADB に載せることで実効性能の向上が期待される.

Red Black 法は, 元の SOR 法と計算順序が異なるため, 結果に差異が生じる. そのため SOR 法のような反復計算において, 収束までの繰り返し回数が元の SOR 法の計算より増える場合がある. 極端に収束回数が増加する場合は, Red Black 法による高速化の効果が相殺される可能性があるので注意されたい.

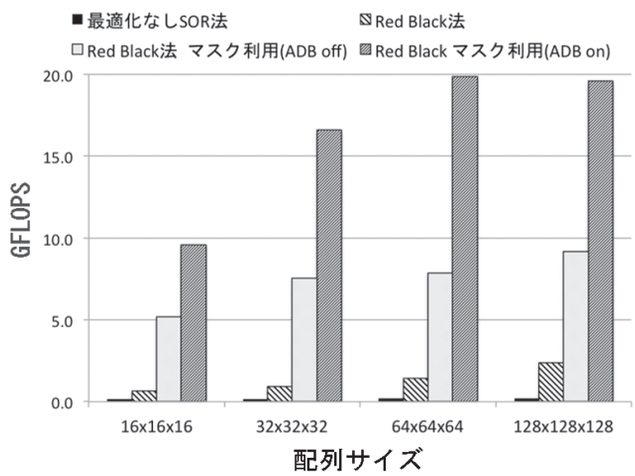


図 4.3-59 配列サイズと性能値

［大規模科学計算システム］

ライブラリ・アプリケーションの紹介

（2012 年 4 月）

共同利用支援係

はじめに

本センター大規模科学計算システムでは、プログラミングのための科学技術計算ライブラリや、分子軌道計算、構造解析、数式処理、グラフ処理、統計データ解析等の各アプリケーションソフトウェアを、利用者の幅広い要望にお応えしてサービスしています。

この稿では、スーパーコンピュータ・並列コンピュータ上でサービスしているライブラリプログラムとアプリケーションソフトウェアの紹介をします。

サービス一覧表

システム	プログラミング言語	ライブラリ	アプリケーション
スーパーコンピュータ SX-9 **super.isc.tohoku.ac.jp**	Fortran90/SX C++/SX	**ASL** **MathKeisan for SX**	
並列コンピュータ Express5800 **gen.isc.tohoku.ac.jp**	Fortran95 C++	**ASL** **Math Kernel Library**	**Gaussian09,03** **MSC.Marc** **MSC.Marc Mentat** **SAS** **Mathematica** **MATLAB**

ライブラリ、アプリケーションの紹介は、以下の URL の本センター大規模科学計算システムホームページにも掲載しています。

大規模科学計算システム ホームページ（以下 「ホームページ」）
http://www.ss.isc.tohoku.ac.jp/

本稿中の内容は 2012 年 4 月現在のものですので、ライブラリ、アプリケーションの今後のバージョンアップや利用方法の最新情報については、ホームページを随時ご確認下さい。

ご利用の前に

リモートログイン

スーパーコンピュータ、並列コンピュータへリモートログインする手順です。SSH（Secure SHell）接続を行います。アプリケーションを利用する際は、並列コンピュータにログインします。GUI アプリケーションを利用する際は、**GUI アプリケーションを利用する方法**を合わせてご参照下さい。

	ホスト名（FQDN）	OS	日本語環境
スーパーコンピュータ	super.isc.tohoku.ac.jp	UNIX	EUC-JP
並列コンピュータ	gen.isc.tohoku.ac.jp	Linux	UTF-8

SSH は通信路上のデータを暗号化することで安全性を高めたプログラムです。利用している端末が UNIX, Linux, OS X の場合は通常 SSH クライアントソフトがインストールされているので、すぐに利用できます。インストールされていない場合は端末の管理者にご相談下さい。super, gen ともにプロトコル version 1,2 両方ご利用できます。

【UNIX, Linux からのログイン】

「ターミナル」、「端末」、「terminal」などと呼ばれるアプリケーションを起動します。コマンドを入力するプロンプトが表示され、コマンドの待ち受け状態になります。

並列コンピュータへのログイン例

```
(yourhost)$ ssh 利用者番号@gen.isc.tohoku.ac.jp

The authenticity of host 'gen.isc.tohoku.ac.jp (xx.xx.xx.xx)' can't be
 established.
RSA key fingerprint is fd:c2:9a:11:xx:xx:xx:xx:xx:xx:xx:xx:cd:53:9f.
Are you sure you want to continue connecting (yes/no)? yes  ※1
Warning: Permanently added 'gen.isc.tohoku.ac.jp,xx.xx.xx.xx' (RSA) to
 the list of known hosts.
利用者番号@gen.isc.tohoku.ac.jp's password: パスワード  ※2
Last login: Mon Apr  1 09:26:11 2012 from gen.isc.tohoku.ac.jp

[利用者番号@gen ~]$
```

※1 初めての接続時は問い合わせがありますので、**yes** を入力します。
※2 入力した文字は表示されません。

【OS X からのログイン】

「ターミナル.app」を起動します。接続方法は上記と同じです。

【Windows からのログイン】

１．SSH クライアントソフトのダウンロードとインストール

SSH クライアントソフトの一つである「Tera Term」というフリーソフトをインストールします。以下のページからダウンロード出来ます。2012 年 4 月現在の最新版は 4.73 です。ダウンロード後インストール作業を行って下さい。

Tera Term ダウンロードページ:　　http://sourceforge.jp/projects/ttssh2/

２．サーバへの接続

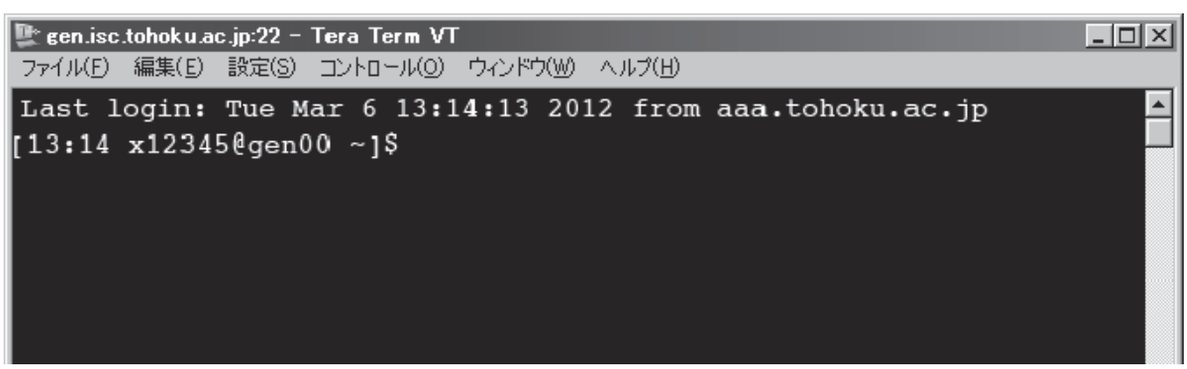
インストールされた Tera Term を起動すると「新しい接続」ダイアログが表示されます。以下の様に入力・設定し、「OK」ボタンを押して下さい。初めて接続する場合、「セキュリティ警告」のウィンドウが表示されます。表示されているホスト名が正しいことを確認し、「続行」ボタンを押して下さい。

ホスト(T):　　gen.isc.tohoku.ac.jp　（並列コンピュータの場合）
サービス:　　SSH
SSH バージョン　　SSH2

「SSH 認証」ウィンドウが表示されたら以下の様に入力・設定し、「OK」ボタンを押して下さい。

ユーザ名(N):　　利用者番号
パスフレーズ:　　パスワード
「プレインテキストを使う(L)」にチェック

以下のプロンプトが表示されるとログインは完了です。



※ 日本語環境を変更する場合は、ツールバーメニューの「設定(S)」→「端末(T)」の「漢字−受信(K)」と「漢字−送信(J)」で設定変更して下さい。

シェルの初期設定

大規模科学計算システムでは、 お勧めの初期環境設定を用意しています。 これによりパスなどの基本的な設定、また各アプリケーションの環境変数等が自動的に設定されます。これは、利用登録時に個々の ID にあらかじめ行っていますので、 通常は作業の必要はありません。

ライブラリやアプリケーションが利用できないという場合には、 この設定が変更されていることが考えられます。 .cshrc ファイル(csh を利用する場合、センターの規定値) または .login ファイル(sh を利用する場合)に、センターで用意している初期設定ファイル /usr/skel/Cshrc または/usr/skel/Login を読み込む設定となっていることを確認して下さい。設定を変更した場合は、設定を反映させるためにログインし直して下さい。

ファイル転送

【コマンドラインでのファイル転送】

ローカル端末から「scp」、「sftp」コマンドが利用できます。どちらのコマンドも通信経路上は暗号化されていますので安全性の高いファイル転送が出来ます。利用方法についてはそれぞれのマニュアルをご参照下さい。

【アプリケーションを利用したファイル転送】

ファイル転送を行う代表的なアプリケーションは Linux では「gftp」、Windows では「WinSCP」、OS X では「Cyberduck」などです。利用方法についてはそれぞれのマニュアルをご参照下さい。アプリケーションの設定において、転送プロトコルは SSH2 を選択して下さい。通信経路上は暗号化されます。

【入出力端末を利用したファイル転送】

センター1F の利用相談室に設置された入出力端末を利用して、USB 接続（USB3.0 対応）の HDD にホームディレクトリのデータをコピーすることが出来ます。センター内ネットワークからのアクセスで、高速なファイルのコピーが可能です。利用方法はセンターまでお問い合わせ下さい。

GUI アプリケーションを利用する方法

GUI を用いたアプリケーション（MSC. Mentat, Mathematica, MATLAB, SAS）の実行には、ローカルマシンに X Window System 環境の設定が必要です。（Mathematica の実行にはフォントパスの設定が必要になる場合があります。**「Mathematica の利用方法」**をご参照下さい。）

【UNIX, Linux からの利用】

標準で X Window System がインストールされています。ローカル端末から以下の様にログインして下さい。X Forwarding によりローカル画面にアプリケーション画面が表示されます。

例：Matlab を起動する場合

```
(yourhost) $ ssh -X 利用者番号@gen.isc.tohoku.ac.jp ※1
利用者番号@gen.isc.tohoku.ac.jp's password: パスワード
[利用者番号@gen ~]$ matlab
```

※ 1　大文字の“X”です。

【Windows からの利用】

１． 商用のアプリケーションを利用する場合

Windows 用 X サーバは、X サーバソフトとしていくつかのメーカから販売されています。

・ASTEC-X （アステック・エックス）
・Exceed（Open Text Exceed オープンテキスト・エクシード）

それぞれの利用方法について詳しくは各社の HP をご参照下さい。どちらのソフトも無料評価版があります。

※ フォントパスの追加（Mathematica を利用する場合）
Mathematica を利用する場合には、X サーバに tcp/localhost:7100 のフォントを追加する設定をします。 X サーバにフォントサーバを追加する設定は、各 X サーバソフトのマニュアルをご参照下さい。

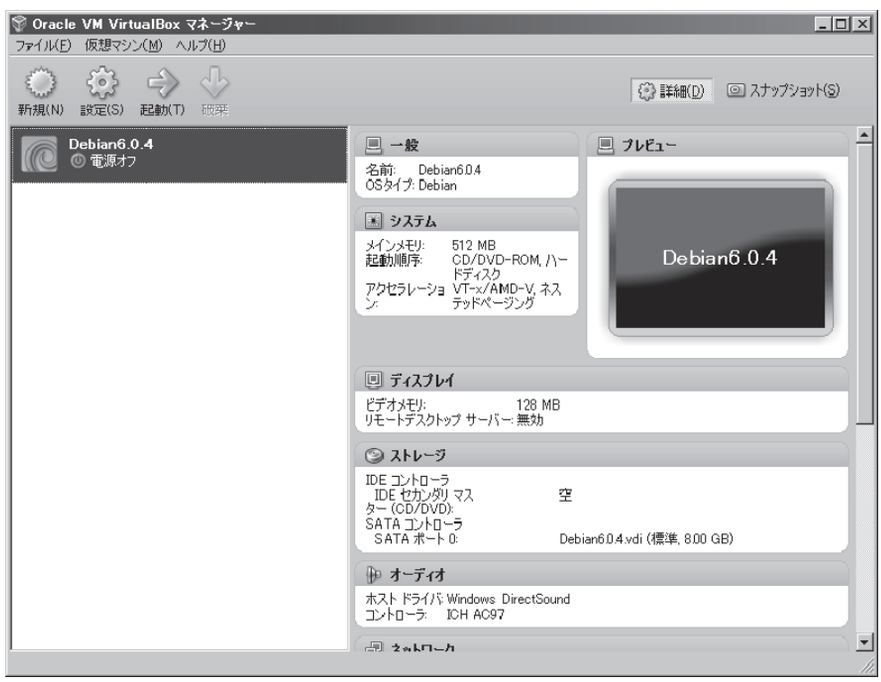
２． Windows に仮想的な Linux をインストールする場合

Windows に「Oracle VM VirtualBox」（以下「VirtualBox」）という仮想化ソフトウェアをインストールし、その環境に Linux をインストールします。

「VirtualBox」は以下のページからダウンロード出来ます。「VirtualBox platform packages」（現在使用している OS に合ったもの）と「VirtualBox Extension Pack」の両方をダウンロードし、インストールを行って下さい。インストール方法の詳細はマニュアルをご参照下さい。2012 年 4 月 12 日現在の最新版は 4.1.12 です。

VirtualBox ダウンロード: https://www.virtualbox.org/wiki/Downloads

VirtualBox 4.1.12 の起動画面

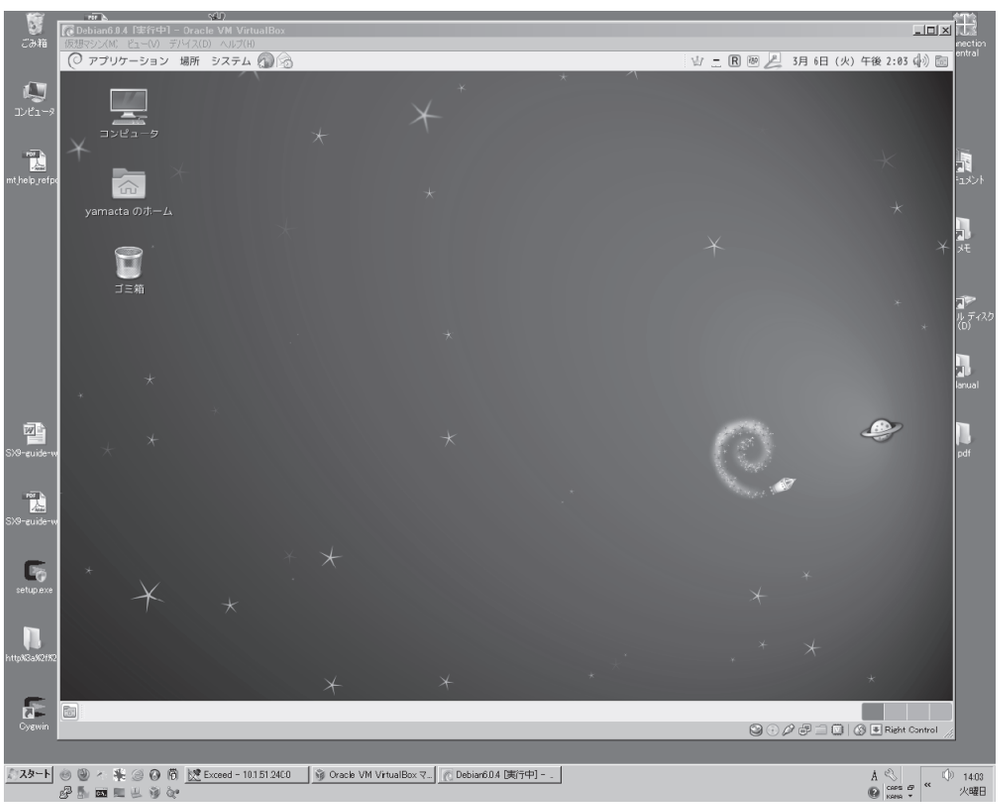


Linuxのディストリビューション、バージョンによってはGUIアプリケーションが正しく表示されない場合があります。センターで動作確認を行っているのは、Debian 6.0.4 (2012.01.28 リリース)です。以下のページからダウンロードし、Virtual Boxの仮想環境にインストールして下さい。X Window Systemの利用だけならインストール CD(470MB)のインストールで可能です。インストール方法の詳細は各マニュアルをご参照下さい。

Debian ダウンロード:　　http://www.debian.or.jp/using/

Linuxをインストール後起動し、Linuxの端末から【UNIX, Linuxからの利用】と同様に利用できます。

Windows 上で動作する仮想Linux(Debian 6.0.4)



【OS Xからの利用】

OS Xでは標準でX Window System環境の「X11.app」がインストールされていますので、OS Xの端末から【UNIX, Linux からの利用】と同様に利用可能ですが、GUI アプリケーションによっては表示の不具合がある場合があります。その場合は、**Windows に仮想的な Linux をインストールする場合** と同様の方法で、Linuxをインストールしてご利用下さい。

ライブラリ

スーパーコンピュータ SX-9

Fortran90/SX,C++/SX 用

科学技術計算ライブラリ **ASL**

数学ライブラリ **MathKeisan for SX**

並列コンピュータ Express5800

Fortran95,C++ 用

科学技術計算ライブラリ **ASL**

数学ライブラリ **Math Kernel Library**

科学技術計算ライブラリ ASL

ASL(Advanced Scientific Library)は、科学技術計算の広範な分野の数値シミュレーションプログラムの作成を強力に支援する数学ライブラリです。ASL を用いることによって、難解な数値計算アルゴリズムの詳細に煩わされること無く高度な科学技術計算プログラムを作成することができます。次のような数値計算分野に対応しています。

＜基本機能＞

格納モードの変換,基本行列演算,最小二乗法,固有値・固有ベクトル,連立 1 次方程式(直接法),連立 1 次方程式(反復法),対称連立一次方程式(反復法),非対称連立一次方程式(反復法),フーリエ変換とその応用／時系列分析,微分方程式とその応用,数値微分,数値積分,3 次元境界要素法用の数値積分法,近似・補間,スプライン関数,特殊関数,乱数,ソート・順位付け,方程式の根,極値問題・最適化

＜並列処理機能＞

基本行列演算,連立 1 次方程式(直接法),固有値・固有ベクトル,フーリエ変換とその応用／時系列分析,乱数,ソート・順位付け

プログラミング言語・コンパイラ

スーパーコンピュータ

Fortran90/SX ・ sxf90,sxmpif90
C++/SX ・ sxcc,sxmpicc

並列コンピュータ

Fortran95 ・ f95,mpif95
C++ ・ cc,mpicc

利用方法

ASL ライブラリは自動的にリンクされます。設定は特に必要ありません。

コンパイルはすべて並列コンピュータ上で行います。
プログラムのコンパイルについては、ホームページの「スーパーコンピュータシステム SX-9 利用ガイド」、「並列コンピュータの利用ガイド」をご参照下さい。

スーパーコンピュータ用のコンパイル

```
[gen00 ~]$ sxf90 source.f       ベクトル版 ASL がリンクされます
[gen00 ~]$ sxf90 -Pauto source.f  並列版 ASL がリンクされます

[gen00 ~]$ sxcc source.c
```

並列コンピュータ用のコンパイル

```
[gen00 ~]$ f95 source.f
```

＊C、C++言語の場合は、オブジェクトを作成した後、f95 で ASL ライブラリをリンクする。

```
[gen00 ~]$ cc -c source.c
[gen00 ~]$ f95 -cxxlib -nofor_main source.o -laslcint -lasl
```

マニュアル

PDF 形式のマニュアルを提供しています。

各マニュアルは、並列コンピュータ(gen.isc.tohoku.ac.jp)の以下のディレクトリにあります。並列コンピュータにログイン後、acroread コマンドでご覧下さい。

[FORTRAN]

/usr/ap/ASL-man-super/PDF/ASL/pdf/

[C/C++]

/usr/ap/ASL-man-super/PDF/CINT/pdf/

1main.pdf : 基本機能編 第 1 分冊
(格納モードの変換,基本行列演算,最小二乗法,固有値・固有ベクトル)
2main.pdf : 基本機能編 第 2 分冊
(連立1次方程式(直接法))
3main.pdf : 基本機能編 第 3 分冊
(連立1次方程式(反復法),対称連立一次方程式(反復法),非対称連立一次方程式(反復法))
4main.pdf : 基本機能編 第 4 分冊
(フーリエ変換とその応用／時系列分析)
5main.pdf : 基本機能編 第 5 分冊
(微分方程式とその応用,数値微分,数値積分,3 次元境界要素法用の数値積分法
近似・補間,スプライン関数)
6main.pdf : 基本機能編 第 6 分冊
(特殊関数,乱数,ソート・順位付け,方程式の根,極値問題・最適化)
8main.pdf : 並列処理機能編
(基本行列演算,連立一次方程式(直接法),固有値・固有ベクトル,連立一次方程式(反復法)
フーリエ変換とその応用／時系列分析,乱数,ソート・順位付け)

数学ライブラリ集　MathKeisan for SX , Math Kernel Library

MathKeisan for SX は NEC のハイパフォーマンス・コンピュータ用に高度に最適化された数学ライブラリ集です。Math Kernel Library は、Intel プラットフォーム用に最適化された数値演算ライブラリです。

MathKeisan for SX および Math Kernel Library に含まれるライブラリは以下のとおりです。
※MathKeisan for SX および Math Kernel Library のいくつかのサブルーチンについては、同機能のものが ASL にも含まれています。ASL は、日本電気製マシン用に最適化されたライブラリですので、同機能であれば、ASL の利用をお薦めします。

BLAS	ベクトル、行列の基本演算
LAPACK	高性能コンピュータ用連立一次方程式、固有値解析
ScaLAPACK	連立一次方程式、固有値解析（MPI による並列版、PBLAS を含む）
BLACS	ベクトル、行列の基本演算のためのメッセージパッシングライブラリ
PARBLAS	共有メモリ用の並列版 BLAS（for SX のみ）
CBLAS	BLAS の C インタフェース
SBLAS	スパース BLAS(ACM Algorithm 692 参照)
FFT	HP VECLIB 並びに SGI/CRAY LIBSCI 3.1 のインタフェースを持つ FFT
PARFFT	HP VECLIB 並びに SGI/CRAY LIBSCI 3.1 のインタフェースを持つ共有メモリ用の並列版 FFT（for SX のみ）
METIS	行列、グラフの並べ変え、分割ライブラリ
ParMETIS	行列、グラフの並べ変え、分割ライブラリの並列版(MPI による並列版)
SOLVER	対称疎行列線形問題の直接法ソルバ
ARPACK	大規模固有値解析

プログラミング言語・コンパイラ

MathKeisan for SX（スーパーコンピュータ）

Fortran90/SX ・ sxf90,sxmpif90
C++/SX ・ sxcc,sxmpicc

Math Kernel Library（並列コンピュータ）

Fortran95 ・ f95,mpif95
C++ ・ cc,mpicc

利用方法

各ライブラリをリンクするには、コンパイル時にオプションを指定します。
それぞれのリンク用オプションについてはマニュアルをご参照下さい。

スーパーコンピュータで LAPACK をリンクする例

```
[gen00 ~]$ sxf90 source.f -llapack -lblas
```

マニュアル

[MathKeisan]

HTML 形式のマニュアルを提供しています。
並列コンピュータ(gen.isc.tohoku.ac.jp)にログイン後、以下のコマンドでご覧下さい。

```
スーパーコンピュータ SX-9 用
[gen00 ~]$ w3m /usr/ap/MathKeisan-man/SX-9/J/index.html
```

[Math Kernel Library]

以下の URL を参考にして下さい。
インテル MKL ： http://www.xlsoft.com/jp/products/intel/perflib/mkl/

アプリケーション

非経験的分子軌道計算プログラム	**Gaussian09, 03**
Gaussian プリポストシステム	**GaussView**
汎用構造解析プログラム	**MSC.Marc / MSC.Marc Mentat**
構造解析用汎用プリポストソフトウェア	**MSC.Patran**
数式処理プログラム	**Mathematica**
科学技術計算言語	**MATLAB**
データ解析システム	**SAS**

非経験的分子軌道計算プログラム Gaussian09,03

Gaussianは、Carnegie-Mellon大学のPopleを中心として開発された分子軌道計算プログラムパッケージです。広範囲にわたる非経験的モデルおよび半経験的モデルをサポートしています。

本センターのGaussianには、以下のような特長があります。

＊ 最大16並列までの並列処理が行え、実行時間の短縮が可能です。
＊ スクラッチファイル(テンポラリファイル)を専用の高速ディスクに置くことにより、ファイル入出力時間が短縮されます。

サービスホスト・バージョン

gen.isc.tohoku.ac.jp ・ Gaussian09 RevC.01
Gaussian03 RevE.01

利用方法

Gaussianのプリポストシステムとして GaussView を提供しています。
以下はGaussian利用方法の概要です。

実行コマンド

Gaussianのインプットファイルは、拡張子を .com とします。(例: e2_01.com)

※インプットファイルをWindowsのエディタで作成した場合、拡張子.comのファイルはWindowsでは実行ファイルと認識されるため、 誤ってダブルクリックなどでインプットファイルを実行しないようご注意下さい。 また、ファイル転送ソフトで genに転送する際にはアスキーモードを指定し、転送して下さい。

gen.isc.tohoku.ac.jp にログイン後、 subg09(gaussian03 利用時には、subg03)コマンドに、キュー名と入力プログラム名を指定することにより、バッチリクエストとして実行が行われます。

e2_01.com を解析するコマンド例
(subg09 コマンドに入力ファイルを指定する際は拡張子 .com を省きます)

```
[gen00 ~]$ subg09 a16 e2_01
```

subg09 コマンドで指定できるキュー

キュー名 (ジョブクラス)	利用可能CPU数 (並列数)	CPU時間	メモリサイズ制限 (GBytes)
as	1	無制限	16
a8	8	無制限	128
a16	16	無制限	256

データの大きさなどにより投入するキューを選択して下さい。

8 および 16 並列実行の指定

本センターでサービスしている Gaussian では、8 または 16 並列での並列処理が可能です。大きな分子の解析にぜひご活用下さい。

8または16並列で実行するには、ルートセクションにLink 0コマンドの**%NProc=並列数**を追加します。手入力の場合は、テキストエディタで先頭行に追加、 GaussView 等ではインプットファイル作成画面の Link 0 section の項に追加して下さい。

実行時には、subg09 コマンドでキュー **a8** または **a16** を指定して下さい。

使用メモリ量の指定

実行して「メモリ量が足りない」というエラーになった場合は、 Link 0 コマンド **%Mem=** で使用メモリ量を増やして下さい。

16 並列、メモリ 16GB の設定をしたインプットファイル e2_01.com を実行する例

```
[gen00 ~]$ cat e2_01.com ← インプットファイルの内容を表示

 %NProc=16    ← 並列数
 %Mem=16Gb    ← メモリ量
 # RHF/6-31G(d) Pop=Full Test

 Formaldehyde Single Point

 0 1
 C   0.   0.   0.
 O   0.  1.22  0.
 H  .94  -.54  0.
 H -.94  -.54  0.

[gen00 ~]$ subg09 a16 e2_01
```

実行結果

計算が終了すると、インプットファイル名に拡張子.log がつけられた結果ファイル (例: e2_01.log)が作成されます。計算結果をはじめ、CPU 時間などの計算機使用量に関する情報もここに含まれます。

正常終了ならば、このファイルの末尾に「Normal termination of Gaussian 09.」というメッセージが出力されます。

ファイルの末尾を表示する tail コマンドで確認できます。

```
[gen00 ~]$ tail e2_01.log
  :
 Job cpu time:  0 days  0 hours  0 minutes 30.7 seconds.
 File lengths (MBytes):  RWF=   11 Int=    0 D2E=    0 Chk=    8 Scr=    1
 Normal termination of Gaussian 09 at Mon Nov 1 12:00:00 2006.
```

結果ファイルの詳細な見方は、マニュアル等をご参照下さい。

チェックポイントファイル

チェックポイントファイルは、デフォルトで作成される結果ファイル（.log ファイル）より詳細な結果が出力され、計算のやり直しや結果を画像表示するためなどに使用されます。チェックポイントファイルを出力するには、ルートセクションに Link 0 コマンドの **%Chk=*チェックポイントファイル名*** を追加します。

マニュアル

本センター本館 1 階 利用相談室に以下の資料を備えてあります。

電子構造論による化学の探求 第二版,ガウシアン社,1998
Gaussian 09 User's Reference
Gaussian 09 IOps Reference
Gaussian 09 Online Manual, http://www.gaussian.com/
Gaussian プログラムによる量子化学計算マニュアル ： 堀憲次, 丸善出版
すぐできる量子化学計算ビギナーズマニュアル ： 武次鉄也, 講談社
すぐできる分子シミュレーションビギナーズマニュアル ： 長岡正隆, 講談社
Gaussian プログラムで学ぶ情報化学・計算化学実験 ： 堀憲次, 丸善出版

Gaussian プリポストシステム GaussView

GaussView は、分子軌道計算プログラム Gaussian のプリポストシステムです。
Windows XP、Windows Vista, Windows 7, Linux 搭載のパソコンなどで動作し、入力データの作成、計算結果の可視化を 3 次元的に行うことができます。

バージョン

5.0.9

お申し込み

利用ご希望の方に、 GaussView の CD-ROM を貸し出しいたします。

利用条件
・東北大学内の方

CD-ROM は、お手数ですが Gaussian 利用申請書をホームページよりダウンロードしてご記入の上、当センターまで直接取りにいらして下さい。

利用方法

インストール方法、データ作成方法などについては同梱マニュアルまたは以下のHPをご参照下さい。

ヒューリンクス Gauss View 5： http://www.hulinks.co.jp/software/gaussview/

並列コンピュータ gen.isc.tohoku.ac.jp の Gaussian で解析を実行する手順

1. 入力データ作成後、Gaussian のインプットファイル「.com」としてエクスポートします。
2. インプットファイルを gen.isc.tohoku.ac.jp に転送します。
3. gen.isc.tohoku.ac.jp にログインします。
4. subg09 コマンドにより解析を実行します。
5. 結果ファイルを転送し GaussView で表示します。

チェックポイントファイル(.chk)は、Gaussian のユーティリティコマンド formchk により書式付(.fchk)に変換後転送して下さい。

汎用構造解析プログラム MSC.Marc / MSC.Marc Mentat

MSC.Marc は有限要素法による非線形汎用構造解析プログラムです。世界中で広く利用され最も評価を受けているプログラムの一つで、その扱える解析は以下の通り非常に広範囲にわたっています。

非線形／大変形／接触／弾塑性／剛塑性／破壊／熱伝導／動的非線形／境界非線形流体と固体の連成／電気伝導と熱伝導の連成／熱と応力の連成

MSC.Marc Mentat は、汎用構造解析プログラム Marc の会話型プリ／ポストプロセッサとして、有限要素モデルの作成および解析結果の表示が行えます。

サービスホスト・バージョン

gen.isc.tohoku.ac.jp ・ MSC.Marc /Mentat 2010.2

利用方法

Marc のプリポストプロセッサとして、Mentat の他に MSC.Patran も提供しています。

run_marc コマンドでの解析実行

実行コマンド

Marc の入力ファイルは、拡張子を .dat とします。（例： job_name.dat ）

gen.isc.tohoku.ac.jp にログイン後、 run_marc コマンドに入力ファイル名を指定し実行することにより、バッチリクエストとして解析が行われます。
（バッチリクエストは **am** (Marc 専用、CPU 時間無制限、最大メモリ 16GB)というキューに投入されます）

job_name.dat を解析するコマンド例
（run_marc コマンドに入力ファイルを指定する際は拡張子 .dat を省きます）

```
[gen00 ~]$ run_marc -jid job_name -v n
```

run_marc の入力オプション

オプション	説明
-jid (-j) *job_name* (必須)	入力ファイル名 *job_name*.dat を指定
-cpu *秒数*	cpu 時間の制限
-ver yes(デフォルト) (-v) no	バッチリクエスト投入前に確認する。 バッチリクエストをただちに投入する。
-user (-u) *user_name*	ユーザサブルーチン *user_name*.f を指定

その他のオプションは、「マニュアル C 編 プログラム入力 付録 B 表 B-2」をご参照下さい。

解析結果

バッチリクエストが終了すると、主に以下のようなファイルが作成されます。

job_name.out	(解析結果)
job_name.log	(解析ログ)
job_name.t16	(ポストファイル)
job_name.sts	(ステータスレポートファイル)
job_name.batch_err_log	(エラーログ)

解析時の指定によって、この他にもファイルが作成されます。

それらのファイルの概要は、「マニュアル C 編プログラム入力 付録 B 表 B-1」をご参照下さい。

終了番号 (exit number)

解析結果ファイル(job_name.out)の末尾にある marc exit number により、正常に終了したか、エラー終了か、またエラー終了の場合はその原因がわかります。

終了番号を確認する
(tail コマンドで job_name.out の末尾を表示)

```
[gen00 ~]$ tail job_name.out

************************************************************

                  MSC.Marc Exit number 3004

                    check marc exit passed
[gen00 ~]$
```

終了番号	説明
3004	正常終了
13	入力データにデータエラーが検出された。
2004	剛体変位が発生している、または全体剛性マトリクスが非正定マトリクスになっている。
3002	指定したリサイクル数内で収束しない。

この他の番号については、「マニュアル C 編 プログラム入力 付録 A」をご参照下さい。

プリポストプロセッサ Mentat からの解析実行

Mentat の起動

Mentat の起動には、並列コンピュータに接続する際に X forwarding の設定を行う必要があります。

```
yourhost$ ssh -X 利用者番号@gen.isc.tohoku.ac.jp
  :
[gen00 ~]$ mentat
```

解析実行

Mentat 上でモデルを作成し、解析のための設定を行った後、

メインメニュー [JOBS] -> [RUN] -> [submit1]

という操作をすることで、バッチリクエストとして解析を実行します。
(バッチリクエストは **am** (Marc 専用、CPU 時間無制限、最大メモリ 16GB)というキューに投入されます)

スタティックメニュー [FILES] -> MARC INPUT FILE [WRITE]

とすることで、run_marc コマンド用入力ファイル(.dat ファイル)を作成することができます。

サンプルプログラム

Marc

マニュアル E 編に掲載されている例題が、並列コンピュータ
gen.isc.tohoku.ac.jp の/usr/ap/MSC2010/marc2010.2/demo/にあります。
コピーしてご利用下さい。

Mentat

マニュアル「ユーザガイド」 に掲載されている例題のプロシジャファイルが、
並列コンピュータ gen.isc.tohoku.ac.jp の
/usr/ap/MSC2010/mentat2010.2/examples/marc_ug/にあります。
コピーしてご利用下さい。

マニュアル

PDF 形式のマニュアルを提供しています。

各マニュアルは、並列コンピュータ(gen.isc.tohoku.ac.jp) の以下のディレクトリにあります。
並列コンピュータにログイン後、acroread コマンドでご覧下さい。

リリースガイド /usr/ap/MSC2010/mentat2010.2/doc/
- release_guide2010r2.pdf : 2010r2 英語版
- release_guide2010r1.pdf : 2010r1 英語版
- release_guide2010r1_jpn.pdf : 2010r1 日本語版

和文(MSC.Marc2003 版) /usr/ap/MSC2010/mentat2010.2/doc/japanese/
- vola.pdf : A 編 理論およびユーザー情報
- volb.pdf : B 編 要素ライブラリ
- volc.pdf : C 編 プログラム入力
- vold.pdf : D 編 ユーザサブルーチンおよび特別ルーチン
- vole.pdf : E 編 例題集
- new_features.pdf : 新機能ガイド
- marc_ug.pdf : ユーザガイド
- mt_help_ref.pdf : Mentat 2003 ヘルプリファレンス
- xsec_adden.pdf : ドキュメント補足資料

English version /usr/ap/MSC2010/mentat2010.2/doc/
- vola.pdf : Volume A:Theory and User Information
- volb.pdf : Volume B:Element Library
- volc.pdf : Volume C:Program Input
- vold.pdf : Volume D:User Subroutines and Special Routines
- vole.pdf : Volume E:Demonstration Problems
- release_guide.pdf : release Guide

有限要素法プログラム汎用プリポストソフトウェア　MSC.Patran

MSC.Patran は、有限要素法構造解析プログラム MSC.Nastran 用として開発されたプリポストソフトウェアです。本センターでは Marc の利用をサポートするためにサービスしています。

MSC.Patran は多くの CAD に対応するダイレクトインターフェースを介して、正確で迅速な CAD 形状のインポートが可能です。さらに優れた特長として、高水準のメッシュ作成機能や可視化機能に加え、 Marc との親和性が高いことが挙げられます。

バージョン

MSC.Patran2012 Windows 版, Linux 版

お申し込み

利用条件(以下の条件をすべて満たしている方)

- 大規模科学計算システムの利用者番号を持っている方
- 本センターでサービスしている Marc のプリポストとして利用する方
- 東北大学内の方

利用ご希望の方は、共同利用支援係までお問い合わせ下さい。

数式処理プログラム　Mathematica

Mathematica は Stephen Wolfram によって作られた、プログラミング言語を備えた数式処理システムです。Mathematica の機能は、数値計算、記号計算、グラフィックスという 3 つに大別でき、この 3 つが一体となって使いやすいインタフェースを提供しています。

サービスホスト・バージョン

gen.isc.tohoku.ac.jp ・ version 8.0.4

利用方法

Mathematica の起動

[GUI 版]

GUI 版の Mathematica の起動には、並列コンピュータに接続する際に X forwarding の設定と、フォントパスの設定を行う必要があります。

```
yourhost$ ssh -X -L 7100:gen.isc.tohoku.ac.jp:7100 利用者番号@gen.isc.tohoku.ac.jp
 :
[gen00 ~]$ mathematica
```

[テキスト版]

```
yourhost$ ssh 利用者番号@gen.isc.tohoku.ac.jp
  :
[gen00 ~]$ math
```

Mathematica の基本的な使い方は、 マニュアル・参考資料 や、Web などをご参照下さい。

マニュアル・参考資料

参考資料

本センター本館1階 利用相談室に、以下の資料を備えてあります。
スティーブンウルフラム Mathematica ブック（日本語版）：トッパン
Mathematica 方法と応用 ： J.W. グレイ, サイエンティスト社
Mathematica プログラミング技法 ： R. メーダー, トッパン
入門 Mathematica ： 日本 Mathematica ユーザー会, 東京電機大学出版局
はやわかり Mathematica ： 榊原進, 共立出版
もっと Mathematica で数学を ： 吉田孝之, 培風館

科学技術計算言語 MATLAB

MATLAB は高機能な数値計算機能と多彩な可視化機能を備えた技術計算ソフトウェアです。科学的、工学的分野の様々な数値計算(特に行列演算)、データ解析、シミュレーション、およびビジュアライゼーションのための統合環境を提供しています。

サービスホスト・バージョン

gen.isc.tohoku.ac.jp ・ Version 7.14.0 (R2012a)

Toolbox

センターで導入している Toolbox です。

MATLAB
Simulink
Communications Blockset
Communications ToolboxControl
System Toolbox
Extended Symbolic Math
Fixed-Point Toolbox
Fuzzy Logic Toolbox
Image Processing Toolbox
MATLAB Compiler
Model Predictive Control Toolbox
Neural Network Toolbox
Optimization Toolbox
Partial Differential Equation Toolbox
Real-Time Workshop

Robust Control Toolbox
Signal Processing Blockset
Signal Processing Toolbox
Simulink Accelerator
Simulink Control Design
Simulink Fixed Point
Simulink Response Optimization
Simulink Verification and Validation
Spline Toolbox
Statistics Toolbox
Symbolic Math Toolbox
System Identification Toolbox
Wavelet Toolbox

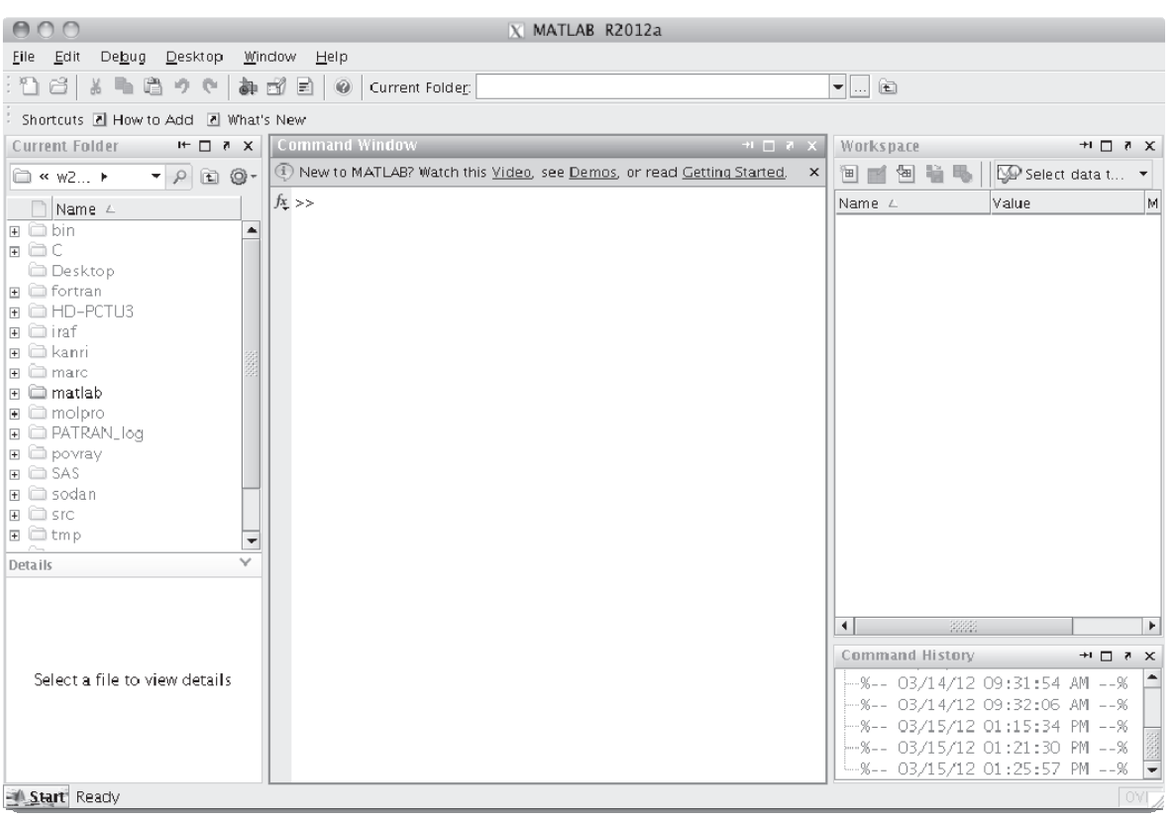
利用方法

MATLAB の起動

[GUI 版]

GUI 版の MATLAB の起動には、並列コンピュータに接続する際に X forwarding の設定を行う必要があります。

```
yourhost$ ssh -X 利用者番号@gen.isc.tohoku.ac.jp
 :
[gen00 ~]$ matlab
```



[テキスト版]

GUI を使用せず、コマンドライン上で起動することも出来ます。

```
(yourhost)$ ssh  利用者番号@gen.isc.tohoku.ac.jp
  :
 [gen00 ~]$ matlab -nojvm -nosplash -nodesktop -nodisplay


                    < M A T L A B (R) >
          Copyright 1984-2012 The MathWorks, Inc.
        Version 7.14.0.739 (R2012a) 64-bit (glnxa64)
                     February 9, 2012

  To get started, type one of these: helpwin, helpdesk, or demo.
  For product information, visit www.mathworks.com.

>>
```

[バッチ処理]

Matlab の組み込み並列処理機能を使用し、32 並列までの処理が可能です。最大メモリも 512GB まで利用可能です。大規模な計算にご利用下さい。

関数 test を実行するためには以下の様なバッチリクエスト用シェルスクリプトファイルを作成し、job-m というファイル名で保存した例です。ジョブは **m32** クラスに投入します。

```
[gen00 ~] cat job-m   ←ジョブファイルの中身を表示

#PBS -q m32
cd $PBS_O_WORKDIR
matlab -nojvm -nosplash -nodesktop -r test
```

以下のコマンドでジョブを投入します。

```
[gen00 ~]$ qsub job-m
Request 1234.job submitted to queue: m32.
```

MATLAB の基本的な使い方は、 マニュアル・参考資料などをご参照下さい。

サンプルプログラム

MATLAB には豊富なデモがありますので、ご利用下さい。
MATLAB 上で、demo コマンドを実行すると、デモ画面が開きます。

マニュアル・参考資料

マニュアル

日本語オンラインマニュアルが公開されています。以下のページをご参照下さい。
http://www.mathworks.co.jp/help/ja_JP/techdoc/index.html

参考資料

本センター本館1階 利用相談室に、以下の資料を備えてあります。

MATLAB による制御理論の基礎 ： 野波健蔵, 東京電機大学出版局
MATLAB による制御のためのシステム同定 ： 足立修一, 東京電機大学出版局
だれでもわかる MATLAB ： 池原雅章, 培風館
はやわかり MATLAB 第 2 版 ： 芦野隆一, 共立出版
最新 MATLAB ハンドブック第 3 版 ： 小林一行, 秀和システム
MATLAB グラフィックス集 ： 小国 力, 朝倉書店
MATLAB と利用の実際 ： 小国 力, サイエンス社
MATLAB の総合応用 ： 高谷邦夫, 森北出版
最新使える！MATLAB ： 青山貴伸, 講談社
使える！MATLAB/Simulink プログラミング ： 青山貴伸, 講談社
MATLAB による画像＆映像信号処理 ： 村松正吾, CQ 出版

Matlab によるグラフ描画 ： 西村竜一 (広報誌 SENAC Vol.37 No.1 (2004-1))
高機能数値計算・可視化機能ソフト MATLAB の基本的な使い方 ： 陳国曜 他
(広報誌 SENAC Vol.29 No.4 (1996-10))

データ解析システム SAS

SAS(Statistical Analysis System) は、基本システムである BaseSAS ソフトウェアを中心とした汎用統計パッケージです。

サービスホスト・バージョン

gen.isc.tohoku.ac.jp ・ SAS 9.2

導入プロダクト

当センターで導入しているSASプロダクトです。

Base SAS
SAS/ETS
SAS/GRAPH
SAS/STAT

利用方法

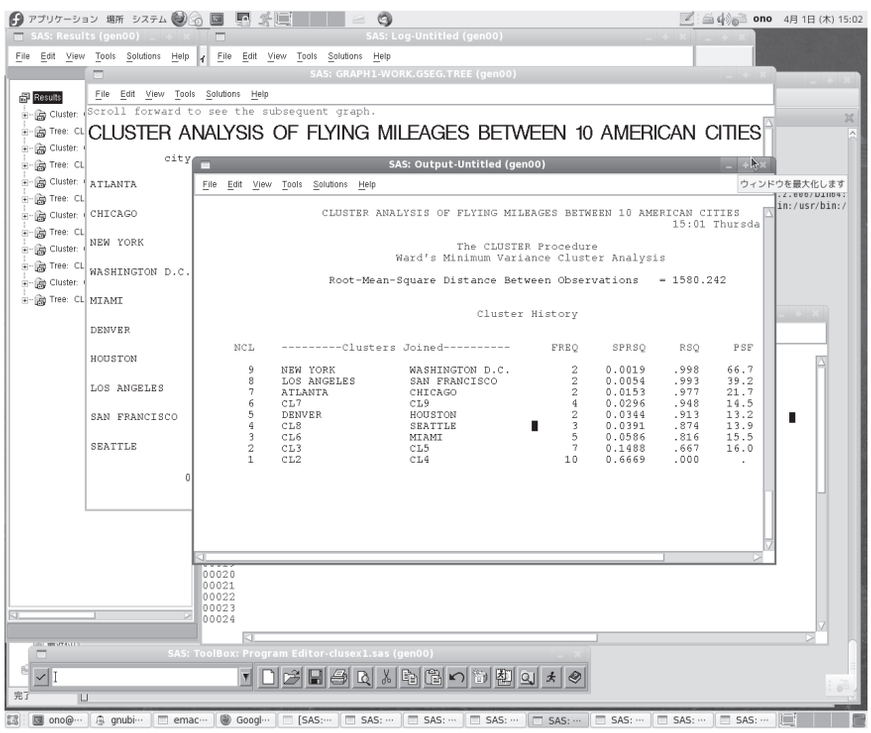
【対話型ディスプレイマネージャでの実行】

対話型ディスプレイマネージャとは、ウィンドウ画面を使って対話形式で SAS システムに命令を与えたり、メッセージを受け取ったりすることの出来る処理モードです。

SAS の起動

SAS の起動には、並列コンピュータに接続する際に X forwarding の設定を行う必要があります。

```
yourhost$ ssh -X 利用者番号@gen.isc.tohoku.ac.jp
  :
[gen00 ~]$ sas
```



Log、**Output**、**Program Editor**、**ToolBox**、**Results**、**Explorer** の 6 つのウィンドウが開きます。
Program Editor ウィンドウで、SAS プログラムを作成した後、 **Program Editor** ウィンドウ上のメニュー「Run」→「Submit」で、プログラムが実行されます。 結果は **Output** ウィンドウに出力されます。

【コマンドでの実行】

X Window System 環境がなくても SAS の利用が可能です。

実行コマンド

SAS の入力ファイルは、拡張子を .sas とします。（例： test01.sas ）

gen.isc.tohoku.ac.jp にログイン後、sas コマンドに入力ファイル名を指定し実行することにより、会話型処理として実行が行われます。

例）test01.sas を実行する
（sas コマンドに入力ファイルを指定する際は拡張子 .sas を省きます）

```
[gen00 ~]$ sas test01
```

実行結果

実行後、カレントディレクトリに 2 つのファイルが作成されます。

test01.lst	（実行結果）
test01.log	（ログ）

SAS の基本的な使い方は、参考資料などをご参照下さい。

参考資料

本センター本館1階 利用相談室に、以下の資料を備えてあります。

SAS によるデータ解析入門[第 2 版] ： 市川伸一, 大学出版会
SAS による共分散構造分析 ： 野田秀樹, 東京大学出版会
SAS による実験データの解析 ： 高橋行雄, 東京大学出版会
SAS による統計分析 ： 高柳良太, オーム社
データ解析のための SAS 入門 ： 宮岡悦良, 朝倉書店
実用 SAS 生物統計ハンドブック ： サイエンティスト社

SAS による統計分析入門 ： 八巻邦次（広報誌 SENAC Vol.35 No.2 (2002-7)）

［報　告］

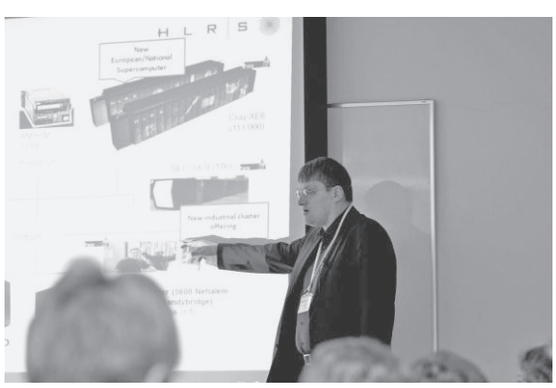
第 15 回高性能シミュレーションに関するワークショップ(WSSP)報告

東北大学サイバーサイエンスセンター　小林　広明

東北大学サイバーサイエンスセンターとドイツのシュトゥットガルト大学高性能計算センター(HLRS)は、2012 年 3 月 22 日(木)～3 月 23 日(金)に第 15 回 Workshop on Sustained Simulation Performance (WSSP)を開催しました。本ワークショップは、サイバーサイエンスセンターと HLRS との間の高性能計算に関する組織的連携協定に基づき両センターのスーパーコンピュータシステムの利用者、並びに国際的に活躍する計算科学者・計算機科学者を招いて、毎年、春と秋にシュトゥットガルト大学と東北大学で交互に開催しているものです。

今回のワークショップでは、昨年の東日本大震災を受け、被災地でもある仙台において、特に**「震災からの日本の復興・再生を目指して　- 安全・安心、ものづくりに資するスーパーコンピューティング技術 -」**をテーマとして、スーパーコンピュータを用いた 3.11 東日本大震災のシミュレーション解析、福島原発事故の放射線汚染マップ可視化技術、我が国の地震津波観測網構築などの安全・安心に関するスパコン活用成果や、国産近距離ジェット機 MRJ の設計、ナノデバイス設計、海水の淡水化システムの開発など先進ものづくりを加速するスーパーコンピューティング技術の講演が行われました。その他、エクサスケール時代のシステムアーキテクチャに関する講演や、ドイツ HLRS および GRS の研究グループから、EU のスパコン戦略や最先端のシミュレーション技術とその産業応用に関する発表も行われました。

2 日間のワークショップでは、延べ約 150 名の参加者を得て、活発な議論が交わされました。第 16 回ワークショップは、2012 年秋にシュトゥットガルト大学で行われる予定です。また、2012 年度に開催されたワークショップの論文集が後日 Springer 社から出版されます。講演予稿集および論文集にご興味がございましたら、センターまでお問い合わせください。

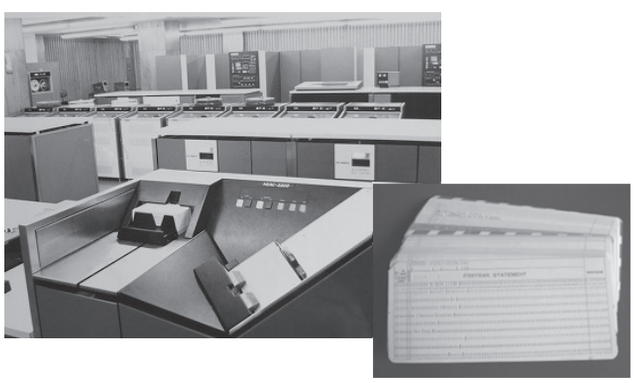


［展示室便り④］

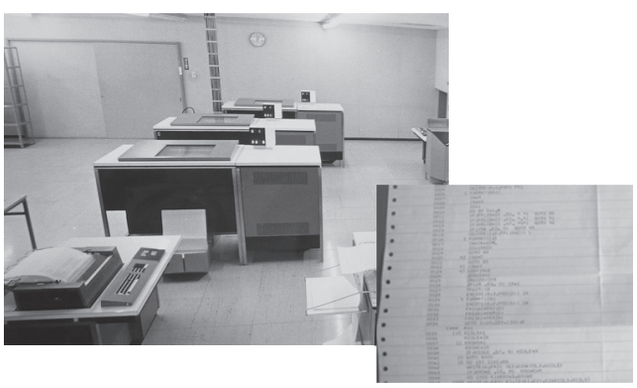
NEAC シリーズ 2200

展示品１　計算機室

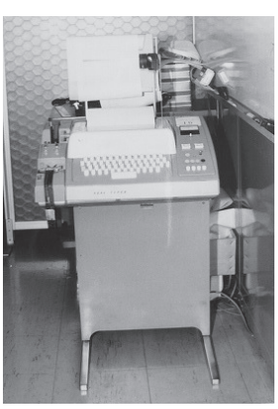
展示品２
磁気テープ装置とディスクパック装置



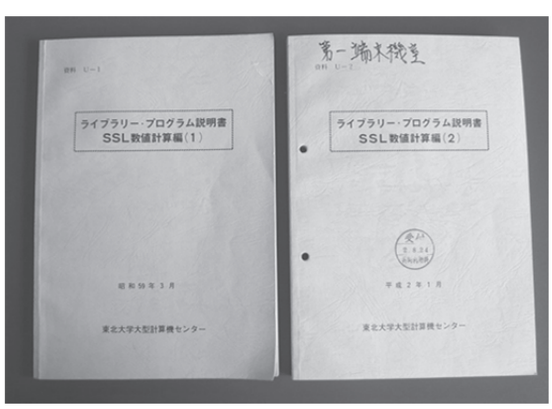
展示品３　カードリーダ装置とパンチカード



展示品４　ラインプリンタ装置と連続紙



展示品５　TSS 端末



展示品６　SSL 説明書

今回は、大型計算機センター発足時に導入された日本電気（株）製 NEAC シリーズ 2200 です。センターで使用した計算機は、このシリーズのモデル 500（1969 年～1971 年）とモデル 500/700（1971 年~1976 年）でした。これらの計算機は、1 語が 48 ビットで構成され、浮動小数点は指数部 12 ビット、仮数部 36 ビットで表現しました。利用形態はバッチ処理と会話処理（TSS）でし

た。東北大学ではセンター発足時から TSS（タイムシェアリングシステム）サービスを計画し、その成果について大泉充郎初代センター長は大型計算機センター「十年史」の中で、以下のようなことを書かれています。

TSS の利用の発展は誠に目覚しいものがあり、遠隔 5 大学(弘前、岩手、秋田、山形、福島)は勿論、青葉山、更にセンター内端末機に迄利用者が殺到し、TSS はパンク状態になることが多く、利用者からの苦情が多くなった。しかし、TSS は東北大に定着し、計算機利用の一形態としての地位を得、・・・・。

展示室には、計算機関連の装置に関するものは保存されておりませんが、当時の計算機室の様子を写した写真、プログラムとデータの入力のために使用したパンチカード、計算結果を印刷した B4 版の連続紙が展示されています。

展示品 1 はモデル 700 を設置した計算機室の全景です。手前左右の装置は操作卓でオペレータがここからシステムの操作をします。中央奥の装置は、CPU とメモリが組み込まれた中央処理装置群です。展示品 2 は、奥の列がおなじみの磁気テープ装置、その手前の洗濯機に似た物は磁気ディスクパック装置です。現在はパソコンなどに組み込まれハードディスクと呼んでいるものです。展示品 3 の写真は、カードリーダ装置とパンチカードです。カードリーダ装置の右側にカードを積み、読み込まれると左上に出てきます。この時代は、プログラム 1 行を、パンチカード機で一枚のカードに作り入力媒体としました。展示品 4 はラインプリンタ装置（中央の 3 台）と実行結果を印刷した連続紙です。この装置は、連続紙、インクリボン、金属の活字ドラム、金属のハンマーにより高速に印字するものです。印刷時には凄まじい音が発生しました。展示品 5 は速度が 50 ボーの TSS 端末機です。展示品 6 はライブラリー・プログラム説明書　SSL 数値計算編（1）、（2）です。大型計算機センター発足時、センター教職員と利用者の協同により NEAC シリーズ 2200 で使用するライブラリを開発・登録し、利用者へ提供いたしました。それが科学計算サブプログラムライブラリ SSL（Scientific Subprogram Library）です。展示品は、つぎのシステムであった ACOS 時（昭和 59 年 3 月）に改版された説明書です。NEAC シリーズ 2200 用としては「ライブラリー・プログラム説明書」No.1(昭和 46 年 11 月)、No.2(昭和 47 年 11 月)、No.3(昭和 49 年 3 月)、No.4(昭和 50 年 3 月)の 4 分冊が発行されました。SSL に収められているプログラムは、

（1）数値計算用の汎用的なサブルーチン
（2）統計計算用の汎用的なサブルーチン
（3）特定分野のサブルーチン
（4）その他

に分類されていました。読者の中には、これらを利用された方も大勢いらっしゃるのではないでしょうか。

[共同利用支援係]

［退職のご挨拶］

センターでの思い出

東北大学情報部情報基盤課　高橋洋一

この 3 月で退職を迎えました。昭和 48 年 1 月に、サイバーサイエンスセンターの前身である大型計算機センターの運用部に配属となりました。運用部は、事務部と研究開発部の中間で計算機を利用者にサービスする部署でした。

運用部の仕事は、利用者への利用支援、システムの操作、システム管理、運用に必要な技術開発などが主な業務でした。私はシステムの操作が主な仕事でしたが、ほかに定期的なファイルバックアップ作業や課金・統計に必要なプログラムの開発なども手がけておりました。

この頃の計算機システムは、NEAC2200-700 (OS MOD7)と NEAC2200-500 (OS MOD4)で、計算機システムの起動、停止、OS の監視等のシステム操作は、すべて手動にて行っていました。NEAC2200-700 はバッチ処理専用で、NEAC2200-500 は TSS サービス専用となっておりました。サービス時間は 10:00～17:00 まで、繁忙期（12 月～2 月）は 10:00～21:00 まで延長していました。昭和 50 年には 24:00 まで、昭和 53 年には 24 時間のサービスとなりましたが、処理しきれない状態が終日続くこともありました。ジョブの入力は、パンチされたカードを読取装置で、手動で入力しました。その入力時やラインプリンタ出力時の騒音は非常に大きく、健康に配慮しながらの操作でした。

昭和 51 年には、計算機システムも ACOS シリーズに代わり操作もだんだんと改善され、自動運転も出来るようになり、昭和 63 年には、スーパーコンピュータの SX-1 が導入されました。その後は、汎用コンピュータもスーパーコンピュータも数年周期で更新され、システムの操作は自動運転が主流になり、サービス時間も 24 時間運転が当たり前となりました。

私が一番印象に残っているのは、平成 6 年にセンターが片平より青葉山に移転する際の建物建設についてのことです。建物の設計では、初めての経験であり、また施設部の考え方と合わないところも多数あったため、何度も施設部に通い打ち合わせを行いました。中でも大きな問題になったことは、計算機を設置する広いマシン室の真ん中に柱があったことです。この柱があると数年ごとに更新する計算機を効率よく設置出来ない為、柱をとることを要求して設計からの見直しとなりました。また、計算機の更新時の搬入口の設置場所でも二転三転しました。このことで学んだのは、絶対必要なものは、妥協しないということでした。

もう一つの思い出は、青葉山移転後の計算機更新時、分散処理の運用に即した利用者管理、課金管理も見据えた運転管理システムの構築に携わったことです。この運転管理は、ソフトウェア、ハードウェアの改造が必要だったため、メーカの技術者も含めプロジェク

トを立ち上げ幾度となく討論を交わして構築し、平成 9 年の 3 月より運用しました。その後も計算機が導入される度に、色々なことがありましたが今となっては良い思い出です。

話は変わって、大型計算機センター時代から今でも続く親睦会のびっと会（カタカナのビットではない）について触れます。その昔にＴ先生が職員ひとりひとりがやわらかいびっとで構成されるようにと名付けた様に伝え聞きました。現在は、歓送迎会がびっと会の主な行事に成りましたが、私が勤め始めた頃は、職員の人数も今より多く春のスポーツ大会、花見、ビール祭り、秋のスポーツ大会、芋煮会、時にはバス旅行など盛りだくさんでした。特にスポーツ大会では、全職員が 2〜3 チームに分かれ、ソフトボール、バレーボール、時にはバドミントン、卓球など昼休みなどを利用し 2 週間ほどの日程で行ったことが思い出されます。

人生でもっとも忘れられないことがあの大震災（2011.3.11）です。丁度その日にびっと会の行事が予定されていました。大震災で人生観が変わりました。これから先、何が本当に大事なのかを残りの人生で考えていきます。

長い間大変お世話になりました。どうもありがとうございました。

みなさまのますますの発展をお祈りいたします。

[Web 版大規模科学計算システムニュース]より

大規模科学計算システムニュースに掲載された記事の一部を転載しています。　http://www.ss.isc.tohoku.ac.jp/tayori/

Gaussian09 のバージョンアップについて（No.128）

Gaussian09 を B.01 から C.01 にバージョンアップしましたのでお知らせいたします。

バージョン名　　：　Gaussian09 C.01
サービスホスト　　：　gen.isc.tohoku.ac.jp

B.01 から C.01 のバグフィクスとマイナーチェンジについての詳細は以下のページをご覧下さい。

Gaussian 09 Revision C.01 Release Notes
http://www.gaussian.com/g_tech/rel_notes.pdf
HULINKS | Gaussian 09 | Gaussian 09 Revisions B.01 から C.01 への変更点
http://www.hulinks.co.jp/software/gaussian/section01_v09_c01.html

（共同利用支援係）

計算科学・計算機科学人材育成のための スーパーコンピュータ無償提供制度について（No.129）

東北大学サイバーサイエンスセンターでは、計算科学・計算機科学分野での教育貢献・人材育成を目的として、大学院・学部での講義実習等の教育目的での利用について、無料（ただし、利用状況によっては上限を設定する場合があります）でベクトル並列型スーパーコンピュータ SX-9（1 ノード，16CPU，1.6Tflop/s，1TB メモリ）システムをご利用いただける制度を用意しております。

利用を希望される場合は、以下の情報を添えて、edu-prog@isc.tohoku.ac.jp までお申し込みください。

・講義担当者氏名
・同所属
・同連絡先（住所，電話，電子メール）
・講義名
・講義実施日時（１セメスターの中で実習を予定している回数）
・センターでの実習利用希望の有無（必要であれば予定日）
・講師派遣の有無
・講義シラバス
・講義ウェブ（もし用意されていれば）
・受講者数（予定）
・必要とする理由（利用目的：例えば、高速数値実験の研修を行うなど）

・期待できる教育効果
・その他（センターへの要望等）

なお、講義終了後、報告書（広報誌 SENAC へ掲載）の提出をお願いいたします。たくさんのお申し込みをお待ちしております。不明な点は、edu-prog@isc.tohoku.ac.jp までお問い合わせください。

（スーパーコンピューティング研究部，共同利用支援係）

民間企業利用サービスについて(No.129)

東北大学サイバーサイエンスセンターでは、文部科学省が平成 19 年度から開始した先端研究施設共用促進事業（旧「先端研究施設共用イノベーション創出事業」）を通して、産学連携共同研究におけるサイバーサイエンスセンターのスーパーコンピュータ学術利用支援を行ってまいりました。平成 23 年度からは、自主事業の制度のもと大学で開発された応用ソフトウェアとスーパーコンピュータを民間企業へ提供しており、平成 24 年度についても引き続き同制度を利用する課題を募集しております。

本サービスにおける利用課題区分は、以下の 2 通りがあります。

・大規模計算利用(有償利用)
・トライアルユース(無償利用)

詳細については以下を参照し、利用を希望される場合は共同利用支援係まで申し込みください。
http://www.ss.isc.tohoku.ac.jp/minkan/

問い合わせ先
東北大学サイバーサイエンスセンター
情報部情報基盤課 共同利用支援係
電話: 022(795)3406
E-mail : uketuke@isc.tohoku.ac.jp
(共同利用支援係)

コンパイラアップデートのお知らせ（No.131）

■2012 年 3 月 26 日にスーパーコンピュータ用のコンパイラ、ライブラリを下記のバージョンへアップデートします。
新リビジョンでは、実行性能の向上が期待されますので、ユーザプログラムを再コンパイルして実行することをお勧めいたします。

FORTRAN90/SX	Rev.450
C++/SX	Rev.092
MPI ライブラリ	library 8.0.15
MPI デーモン	daemon 7.3.3

■コンパイル方法は下記となります。
なお、コンパイルコマンドに変更はありません。

Fortran コマンド名：sxf90
C/C++コマンド名：sxcc
オプションの説明は、man コマンドでも参照できます。

MPI を利用する場合
Fotran コマンド名:sxmpif90
C/C++コマンド名:sxmpicc
オプションの説明は、man コマンドでも参照できます。

■FORTRAN90/SX のコンパイラについて
【主な追加機能・強化機能】

(1)　自動ベクトルデータ・バッファリング機能を追加した。この機能はベクトル化される配列式、またはループ内に現われる配列をコンパイラが自動的に選択し、ADB にバッファリングする。

(2)　ADB に関する新規ベクトル化診断メッセージ、および、新規編集リスト表示を追加した。

(3)　ON_ADB 指示オプションを改善し、以下のように複数の配列変数指定を可能とした。

```
ON_ADB[(配列変数指定[,配列変数指定]...)]
```

(4)　OpenMP Version 3.0 における COLLAPSE 句をサポートした。

(5)　OpenMP の WORKSHARE 構文中にコンパイラ指示オプションを指定できるように、コンパイラを改善した。

(6)　手続き中に OpenMP の THREADPRIVATE 指示行が指定されている場合に、その手続きから呼ばれる手続きがインライン展開されるようにインライン展開機能を強化した。

(7)　ネストした手続呼出しがインライン展開される場合のベクトル化機能を強化した。

(8)　組込み手続 SYSTEM_CLOCK、SYTEM_CLOCKX、ETIME の性能を改善した。
また、この改善に伴い新規実行時オプション F_CLOCK を追加した。

(9)　配列の参照/定義間に重なりがないことを前提とした最適化をサポートした。
また、この最適化に関する新規コンパイラオプション -pvctl [no]conflict
およびコンパイラ指示オプション NOCONFLICT を追加した。

(10)　ベクトルレジスタ割付け最適化をサポートした。
また、この最適化を行うことを指定する新規コンパイラオプション
-pvctl vregs=N およびコンパイラ指示オプション ALLOC_ON_VREG を追加した。

(11) -f2003 オプションが指定された場合、Fortran2003 規格に従って、基本文字型および C 文字種別の文字型のスカラ実引数と仮配列の結合を許すように、コンパイラを改善した。

(12) "-f2003 cbind" オプションによる Fotran2003 の C 相互運用機能と、"-A idbl/idbl4/idbl8" 詳細オプションによる明示的な型パラメタ指定のない型の精度自動拡張機能を、同時に利用できるようにコンパイラを改善した。

詳細についてはリリースメモ、紹介資料もご覧ください。
Rev440 リリースメモ
Rev441 リリースメモ
Rev450 リリースメモ
F2003_JPN.pdf

■C++/SX について
C++/SX の新機能等については下記資料をご覧ください。
C++言語プリプロセッサ利用の 手引.pdf
Rev.092 新機能のご紹介.pdf
互換性留意事項.pdf
Rev092 リリースメモ

■MPI/SX について
MPI/SX については下記のリリースメモをご覧ください。
MPI リリースメモ

(共同利用支援係, 共同研究支援係)

MATLAB のバージョンアップについて (No.132)

数値計算言語「MATLAB」のバージョンアップを行いましたのでお知らせいたします。

MATLAB は、アルゴリズム開発、データの可視化、数値計算を行うための高レベルなテクニカルコンピューティング言語と対話型環境です。MATLAB を利用することにより、C、C++、Fortran といった伝統的なプログラミング言語よりも短時間で科学技術計算の問題を解決することが可能です。

バージョン名	:	MATLAB R2012a (Ver. 7.14.0)
バージョンアップ日	:	2012 年 3 月 27 日 (火)
サービスホスト	:	gen (並列コンピュータ)
起動コマンド	:	matlab (GUI 版)
	:	matlab -nojvm -nosplash -nodesktop -nodisplay (テキスト版)

新機能の概要、機能の詳細については MathWorks 社のページをご覧下さい。
http://www.mathworks.co.jp/products/new_products/latest_features.html

(共同利用支援係)

Mathematica のバージョンアップについて (No.132)

数式処理プログラム「Mathematica」のバージョンアップを行いましたのでお知らせいたします。

Mathematica は Stephen Wolfram によって作られた、 プログラミング言語を備えた数式処理システムです。 Mathematica の機能は、数値計算、記号計算、グラフィックスという 3 つに大別でき、 この 3 つが 一体となって使いやすいインターフェイスを提供しています。

バージョン名	：	Mathematica 8.0.4
バージョンアップ日	：	2012 年 3 月 27 日 (火)
サービスホスト	：	gen (並列コンピュータ)
起動コマンド	：	mathematica　(GUI 版)
	：	math　(テキスト版)

新機能の概要、機能の詳細については Wolfram 社のページをご覧下さい。
http://www.wolfram.com/mathematica/new-in-8/

(共同利用支援係)

平成 23 年度 研究開発公募の報告 (No.133)

平成 23 年度のライブラリ研究開発が、つぎのように終了しましたのでお知らせいたします。

No.	開 発 課 題	開発者	所　属	報　　告	備考
10-01	超 SIMD ビット演算による低消費電力流体シミュレーションコードの開発	松岡　浩	東北大学 電気通信研究所	低消費電力を実現できる流体シミュレーションコードを開発した。新しい格子ガス法シミュレーションコードとして”多段２体衝突法”を考案しその効果を SENAC Vol.44,No3 において発表した。	終了
10-02	大規模線形常微分方程式系のための高速ソルバの開発	中村真輔	秋田県立大学 システム科学技術学部	放物型の偏微分方程式の空間方向を離散化した場合に現れる、線形定係数の大規模常微分方程式系の高速ソルバを開発した。	終了
11-01	３次元データのコンパクト差分用並列 LU 分解ソルバーの開発	三浦英昭	核融合科学研究所	各ノード毎に LU 分解を行うソルバーを作成し、ベクトル化などの調整を行った。	終了

(スーパーコンピューティング研究部, 共同研究支援係)

平成 24 年度共同研究について(No.133)

本センターでは、大規模科学計算システムの利用者と共同でプログラムやアルゴリズムを開発する共同研究を行っています。特にプログラムの MPI 化に積極的に取り組む課題を募集しました。

ライブラリ・共同研究専門部会で審査の結果、以下の 14 件が共同研究課題として採択されましたのでお知らせします。

No.	申 請 者	所　　属	研 究 課 題
1	有吉 慶介	海洋研究開発機構 地震津波・防災研究プロジェクト	海溝型巨大地震サイクルの大規模シミュレーションの開発
2	稲津 大祐	東北大学大学院 理学研究科 地震・噴火予知研究観測センター	海底鉛直地殻変動検出のための数 km スケールを解像する全球海底圧力モデリング
3	岩崎 俊樹	東北大学大学院 理学研究科地球物理学専攻	気象・気候の数値シミュレーション
4	岩長 祐伸	物質・材料研究機構	プラズモニック構造を有する複合光･電子素子の数値的設計
5	宇野　亨	東京農工大学大学院 工学研究科先端電気電子部門	大規模有限周期構造モデルを用いたプラズモンポラリトン発生メカニズムの解明とその応用技術開発
6	小野 高幸	東北大学大学院 理学研究科地球物理学専攻	惑星磁気圏における電磁プラズマ不安定についての計算機実験
7	河野 裕彦	東北大学大学院 理学研究科化学専攻	量子化学計算に基づくナノカーボンの光化学過程の理論的研究
8	佐々木 大輔	東北大学大学院 工学研究科航空宇宙工学専攻	工学問題に対する Building-Cube 法の高度化に関する研究
9	澤谷 邦男	東北大学大学院 工学研究科電気・通信工学専攻	モーメント法の高速化アルゴリズムに関する研究
10	茂田 正哉	東北大学大学院 工学研究科 機械システムデザイン工学専攻	プラズマ流によるナノ粒子群創製プロセスの数値シミュレーション
11	島田 照久	東北大学大学院 理学研究科 大気海洋変動観測研究センター	ダウンスケーリングを用いた東北地方の局地気候と温暖化予測に関する研究
12	豊国 源知	東北大学大学院 理学研究科 地震・噴火予知研究観測センター	球座標系 2.5 次元差分法によるローカルスケール･グローバルスケールの地震波伝播モデリング
13	森川 良忠	大阪大学大学院 工学研究科 精密科学・応用物理学専攻	超高速第一原理電子状態計算コードの開発
14	山本　悟	東北大学大学院 情報科学研究科情報基礎科学専攻	数値タービンの汎用化と大規模並列計算に関する研究

（スーパーコンピューティング研究部，共同研究支援係）

大規模ファイル領域(short 領域)の利用開始について (No.133)

○目的

スーパーコンピュータや並列コンピュータで演算を実行する際、大規模な入出力ファイルを必要とする利用者に、大規模ファイル領域(以下、short 領域)を短期間開放します。

○利用期間

利用期間は、3ヶ月毎（四半期毎）とします。

4～6 月、7～9 月、10～12 月、1～3 月

継続して利用希望の方は、四半期末毎に次の３ヶ月分を申請していただきます。その際、過去の演算利用実績を考慮して継続利用の審査を行います。一定時間の演算利用がなければ、継続利用をお断りする場合もございますのでご承知おきください。目安としては、スーパーコンピュータ、並列コンピュータどちらか支払責任者単位で３ヶ月間の合計が 100 経過時間程度を予定しています。

○利用申請

ご利用希望の際は、支払責任者が登録する利用者番号をまとめて電子メールにて申込み願います。

申込み先： sys-sec@isc.tohoku.ac.jp

例：

件名：short 領域利用申請（4～6 月）

	氏名		利用者番号
支払責任者	○○	○○	u2xxxx
希望利用者	□□	□□	w2xxxx
	□□	□□	w2xxxx

○利用負担額と利用可能なサイズ

short 領域の利用負担額は原則無料とし、4 テラバイトのパーティションを複数の利用者で共有してご利用いただきます。ディスク資源には限りがありますので、利用期間中においても使用済みのファイルは速やかに削除・退避するなど、ご協力をお願いいたします。

また、データのバックアップサービスは行いませんので、他のサーバやパソコンに転送するなどして、各自データのバックアップを行うようお願いいたします。

（共同利用支援係，共同研究支援係）

— SENAC 執筆要項 —

１．お寄せいただきたい投稿内容

次のような内容の投稿のうち、当センターで適当と判定したものを掲載します。その際に原稿の修正をお願いすることもありますのであらかじめご了承ください。

・一般利用者の方々が関心をもたれる事項に関する論説
・センターの計算機を利用して行った研究論文の概要
・プログラミングの実例と解説
・センターに対する意見、要望
・利用者相互の情報交換

２．執筆にあたってご注意いただく事項

(1)原稿は横書きです。
(2)術語以外は、「常用漢字」を用い、かなは「現代かなづかい」を用いるものとします。
(3)学術あるいは技術に関する原稿の場合､200 字～400 字程度のアブストラクトをつけてください。
(4)参考文献は通し番号を付し末尾に一括記載し、本文中の該当箇所に引用番号を記入ください。
　・雑誌：著者, タイトル, 雑誌名, 巻, 号, ページ, 発行年
　・書籍：著者, 書名, ページ, 発行所, 発行年

３．原稿の提出方法

原稿のファイル形式は Word を標準としますが、PDF での提出も可能です。サイズ*は以下を参照してください。ファイルは電子メールで提出してください。

－Word の場合－
　・用紙サイズ：A4
　・余白：上＝30㎜　下＝25㎜　左右＝25㎜　綴じ代＝0
　・標準の文字数（45 文字 47 行）

＜文字サイズ等の目安＞
　・表題＝ゴシック体 14pt 中央　・副題＝明朝体 12pt 中央
　・氏名＝明朝体 10.5pt 中央
　・所属＝明朝体 10.5pt 中央
　・本文＝明朝体 10.5pt
　・章・見出し番号＝ゴシック体 11pt～12pt
　　*余白サイズ、文字数、文字サイズは目安とお考えください。

４．その他

(1)執筆者には、希望があれば別刷 50 部を進呈します。50 部を超える分については、著者の実費負担とします。別刷の希望部数等は投稿の際に申し出てください。
(2)投稿予定の原稿が 15 ページを超す場合は以下まで前もってご連絡ください。
(3)初回の校正は、執筆者が行って、誤植の防止をはかるものとします。
(4)原稿の提出先は次のとおりです。

東北大学サイバーサイエンスセンター内　情報部情報基盤課共同利用支援係
e-mail　uketuke@isc.tohoku.ac.jp
TEL　022-795-3406

編 集 後 記

このSENACはVolume 45，つまり45年目であり，当センターの前身だった大型計算機センターの設置は1969年のことでした。直後の1973年から永年にわたってシステムの運用管理や整備に携われた高橋洋一技術専門職員がこのほど定年を迎えられました。お話しによると，初期はラインプリンターの印刷用紙を補給する力仕事などもあったとのことです。そういえば，プログラムや計算結果を11×15インチにミシン目で折りたたまれた用紙に長い印刷出力として受け取っていた時代を覚えています。レーザプリンタやA4判サイズも登場しましたが，あまり長続きせずに，画面上で仕事をするスタイルに移ってしまい，今ではプリンターの提供はポスター用のものだけになっています。このような移り変わりもあった長い間のご貢献にこの欄を借りてお礼を述べたく思います。ほかに，この年度末で，当センター（情報基盤課）から課長補佐の佐藤均，会計係の佐藤正行，庶務係の赤間友紀が転出しました。前職員の方々のご健康とご活躍を祈っております。(H.S)

大震災から1年以上が経過し，ここ青葉山も雪が多かったことを除けば，落ち着いた春を迎えたような感じがします。震災被害を受けた本センターの建物の補修工事もようやく始まりました。建物自体は，見た目は大きな被害はありませんが，建物内部の壁は一面，工事業者が印した亀裂や破損個所の目印だらけになっています。大地震の痕跡がまだまだ身近にあります。今年は復興元年でもあり，また，HPCI が動き出す年でもあります。スパコンについてはいろいろ話題になりますが，是非ご利用いただき，防災分野をはじめあらゆる分野で研究成果があがることを期待したいと思います。(T.H)

サイバーサイエンスセンター前
整備中の青葉山新キャンパス

平成24年4月発行
編集・発行　東北大学
サイバーサイエンスセンター
仙台市青葉区荒巻字青葉6-3
郵便番号　980-8578
印　刷　東北大学生活協同組合
プリントコープ

システム一覧

計算機システム	ホスト名	機　種
スーパーコンピュータ	super.isc.tohoku.ac.jp	SX-9
並列コンピュータ	gen.isc.tohoku.ac.jp	Express5800

サービス時間

利用システム名	利用時間帯
スーパーコンピュータ	連　続　運　転
並列コンピュータ	連　続　運　転
館 内 利 用	月曜日～金曜日は 8:30～21:00、 土・日・祝日は閉館

ジョブクラスと制限値

計算機システム	処　理	ジョブクラス	ＣＰＵ時間	メモリ容量
スーパー コンピュータ	会話型	(4cpu)	1 時間	8GB
	バッチ 処理	ss　(4cpu)	1 時間	256GB
		s　(4cpu)	無制限	32GB
		p8　(8cpu)	〃	512GB
		p16　(16cpu)	〃	1024GB
		p32　(32cpu)	〃	1024GB×2
		p64　(64cpu)	〃	1024GB×4
並列 コンピュータ	会話型	(4 並列)	1 時間	8GB
	バッチ 処理	as　(非並列)	無制限	16GB
		am　(Marc 専用)	〃	16GB
		am2　(Marc 専用)	〃	128GB
		a8　(8 並列)	〃	128GB
		a16　(16 並列)	〃	256GB
		a32　(32 並列)	〃	512GB

目次

東北大学サイバーサイエンスセンター
大規模科学計算システム広報　Vol.45 No.2 2012–4