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

2011 年 10 月 17 日　第 2.1 版

（初版　2011 年 7 月 8 日）

大阪大学

レーザーエネルギー学研究センター

高性能計算機室　福田優子

はじめに

このテキストは、大学もしくは大学院で初めてパーソナルコンピュータ(以下パソコンと略します)、場合によってはスーパーコンピュータ（以下、スパコンと略します）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ています。「京」が 2011 年 6 月に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。その性能を引き出すために最低必要な基礎的な知識や概念を記載したいと思っています。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、このテキストの目標です。

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、最初にこのテキストを読んでいただけば、参考になるというテキストにしたいと思っています。このテキストは、もともと大学のセンターなどの大規模計算機システムを始めて利用する大学生＆大学院生を対象に作成していた「スパコン利用の基礎」と、「スパコンとは-高速化の基礎-」として作っていたものを合体してタイトルも変更し初版としたものです。

パソコンしか利用しないという方は、４章または５章までお読みください。計算機センターなどの大規模なシステムを利用される方は、パソコン以外にも知っておいていただきたいことを６章にまとめましたので、６章を読んでから各センターの説明書に進んでいただくと、分かりやすいと思います。7 章には、質問の多い項目の説明を付録としてまとめました。

私は、理工系の情報系以外の学生の方が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 年前までは端末室に来て作業されるので、分からないことはその都度説明することができましたが、最近はみなさん研究室（ひょっとしたら自宅）にこもって、ご自分のパソコンからネットワーク経由で利用されるので、どこで困っているのかも見えなくなってしまいました。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は、気軽に質問していただきたいと思っています。このテキストは、どこが分からないか分からない、そんな方のために書きました。これを見ても分からんということはぜひ、どこが分からないか教えてください。また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。気楽にシステム管理者や問い合わせ先にお問い合わせください。

スーパーコンピュータ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。大阪大学に導入されている NEC 製の SX というスーパーコンピュータ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、パーソナルコンピュータ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を提供してくださいました。このテキストと合わせて勉強していただくとよいと思います。(2011.6.24 初版)

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。これからも、最新の状況に対応できるように改版を進めたいと思っていますので、ご理解とご協力をお願いいたします。
（2011/9/13 の第 2 版からは軽微な誤字などを修正し、V2.1 としました）

2011/10/17　大阪大学
レーザーエネルギー学研究センター
（阪大レーザー研）
高性能計算機室　福田優子
（fukuda-yko＠ile.osaka-u.ac.jp）

用語の説明：

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、このテキストは、概念をつかんでいただくことを目的としていますので、このテキスト内の用語は統一したいと思っています。分かりにくい用語は連絡いただいたら説明するようにしたいと思いますので、お気軽にご連絡ください。

計算機

スーパーコンピュータ、クラスタ、ワークステーション、パーソナルコンピュータ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と呼びます。

CPU、プロセッサ、コア

計算機の心臓部分である演算をする装置のこと（中央処理装置)。プロセッサと呼ばれることもあります。また、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、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の中の実際に演算などの処理をする部分（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うが、CPUと高速に通信できる。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する。

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。

阪大 CMC、SX

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、CMC や阪大レーザー研に設置されているスパコンのこと。

参考文献など)

・各センターの大規模計算機システムなどの HP

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

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

HPF 推進協議会　http://www.hpfpc.org/

平成 22 年度 HPF 推進協議会総会のページに「PC クラスタで並列プログラミング -High Performance Fortran で楽々並列化」の本の紹介が掲載されています。

XcalableMP　http://www.xcalablemp.org/

・公開テキスト(阪大レーザー研のホームページ)

http://www.ile.osaka-u.ac.jp/research/cmp/text.html

公開予定

「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」 このテキスト

「Fortran でシミュレーションをしよう」 摂南大学理工学部電気電子工学科　田口俊弘

公開中

「HPF 講習会テキスト 6 種類」

1　パソコンやスパコンを利用する前に　–基礎の基礎-

1.1 計算機は壊れる！セーブは常識

計算機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、他の計算機にコピーしておき、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。災害のことを考えるとさらに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。災いは忘れたころにやってきます。

1.2 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、長時間画面にむ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「VTD 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います。

1.3 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、エクセルでたいていのことをすます方もおられるようですが、このテキスト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。先輩から引きついだもの

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 1-1 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。

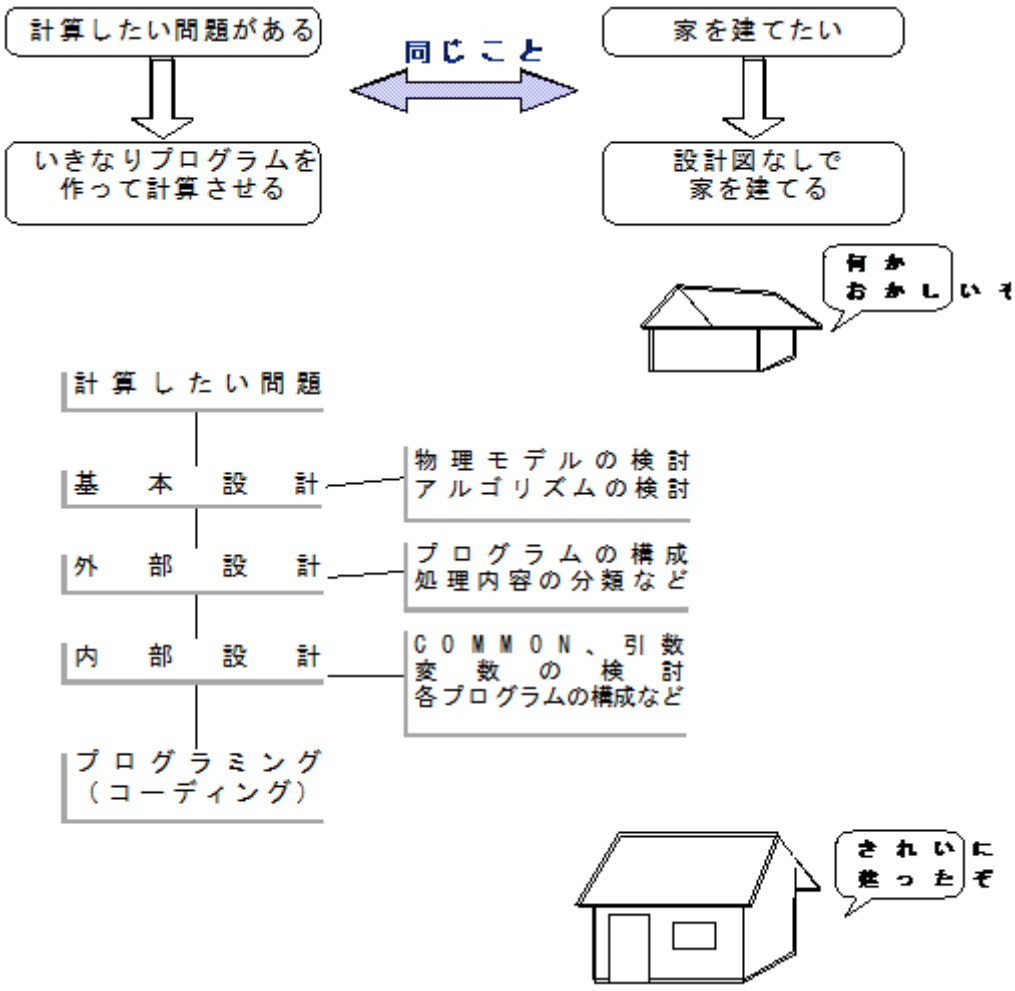


図 1-1　プログラム作成の概念図

1.4 誰が見ても分かりやすいプログラミングを作りましょう

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、プログラムの実行とは関係ない注釈のことで、覚えのために書いておくメモのことです。プログラムの半分がコメントになるのが望ましい形です。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。

コメント例）！以降行末までがコメント

```
! 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)
     :
     :
```

・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。

NIFS 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 などの型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。

1.5 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 CPU（演算処理装置）の他に、メモリ（主記憶）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 のメモリ搭載のパソコンで５G 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、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。

- Windows のタスクマネージャー
- 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 proginfo detail で詳細が表示されます)

1.6 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。スワップがおこるとシステムの効率は非常に悪くなります。前節で２G 搭載のパソコンで５G 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が、このような場合は、メモリに入りきらない分を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。

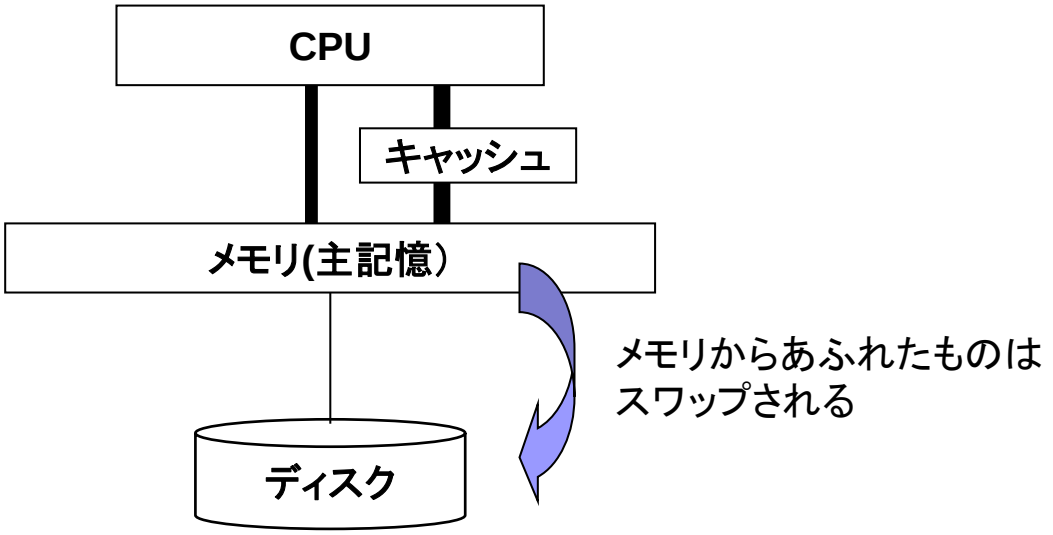


図 1-2　CPU とメモリとスワップ

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が、CPU とメモリの間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。CPU の計算速度（≠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。

2　プログラミングするときに気をつけてほしいこと

2.1 メモリのアクセスは連続に

１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、実際にはプログラムを作るときに、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。このテキストでは FORTRAN の説明はしませんので、FORTRAN がわからんという方は、詳細は田口先生のテキストを勉強していただきたいですが、ここでは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図 2-1 のように配置されます

a (1)	a (2)	a (3)	a (4)	・・・・・

図 2-1　FORTRAN 1 次元配列のメモリ上の配置

プログラムでは以下のように書くと、 a に連続的にアクセスすることになります。

```
do i=1,imax
a(i) = ・・・
enddo
```

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、図の左のプログラムのように、内側からループを回すようにすると、メモリに連続的にアクセスすることになりますが、右のように外側のループを内側に書くと、100 個おきに不連続にアクセスすることになりますので、基本的には内側のループを内側で回すようにしましょう。Ｃの場合は、逆ですので注意してください。

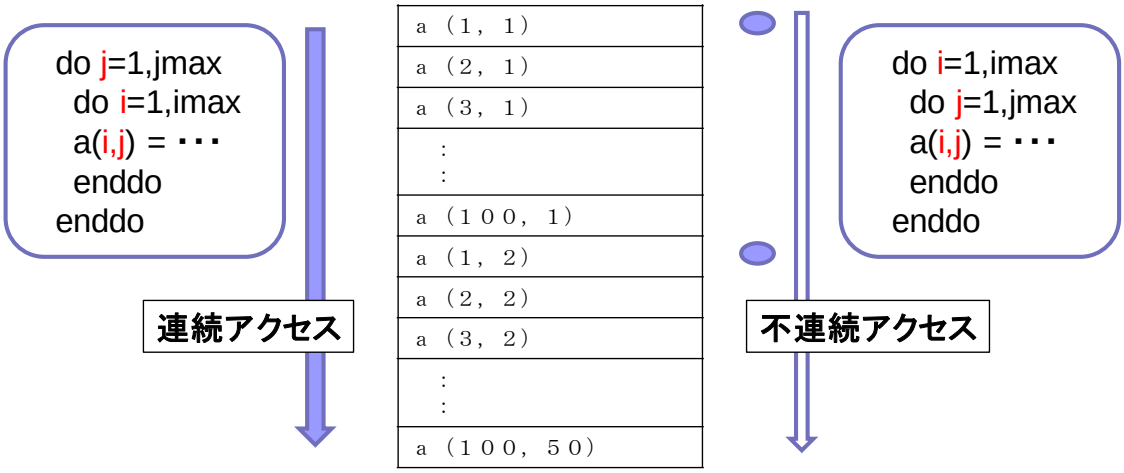


図 2-2　FORTRAN　2 次元配列のメモリ上の配置

2.2 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。詳細は、ここでは説明できませんが、2.1 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。

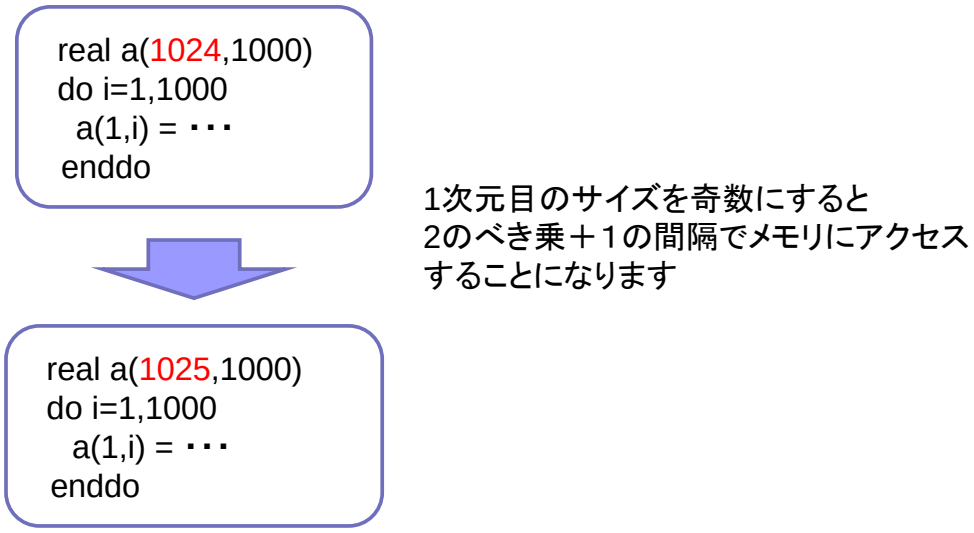
たとえば(3, 3, 10000)のような配列を宣言したほうが、物理的にピッタリくる場合でも、(10000, 3, 3)のように配列を宣言し、一番内側の 10000 でループを回すようにしてください。そうすることで、メモリに連続にアクセスし、効率のよいプログラムとなります。

（注意：ベクトルマシンについては上記はあてはまりますが、スカラーマシンの場合は、キャッシュにのるかどうかが効いて、一概には言えない場合もあります。付録 7.5 を参照してください。）

2.3 バンクコンフリクトを避けましょう

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、以下のように 1 次元目の配列の宣言を奇数にします。



2.4 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 2-3 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。

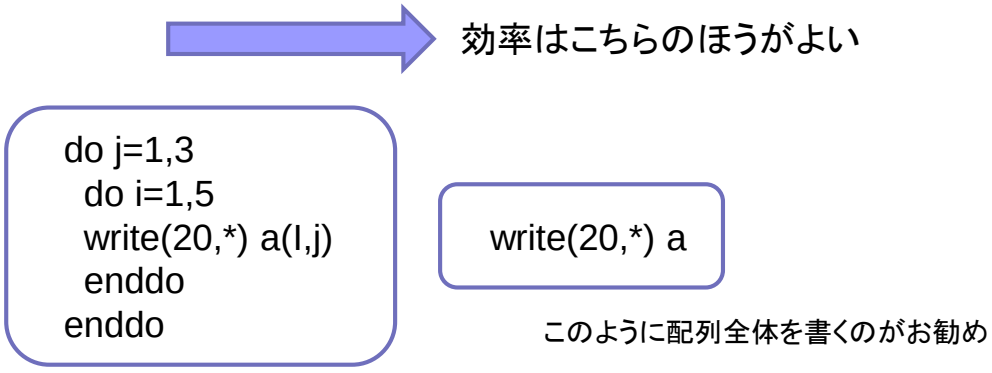


図 2-3 write 文の形式と効率

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。配列を切ってその中に格納し、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、メモリ容量は必要になりますが、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、実効性能があがります。

3　プログラム実行の概念

プログラムができあがったら、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、一般的な FORTRAN 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。

3.1 コンパイル

図 3-1 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。Linux では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。

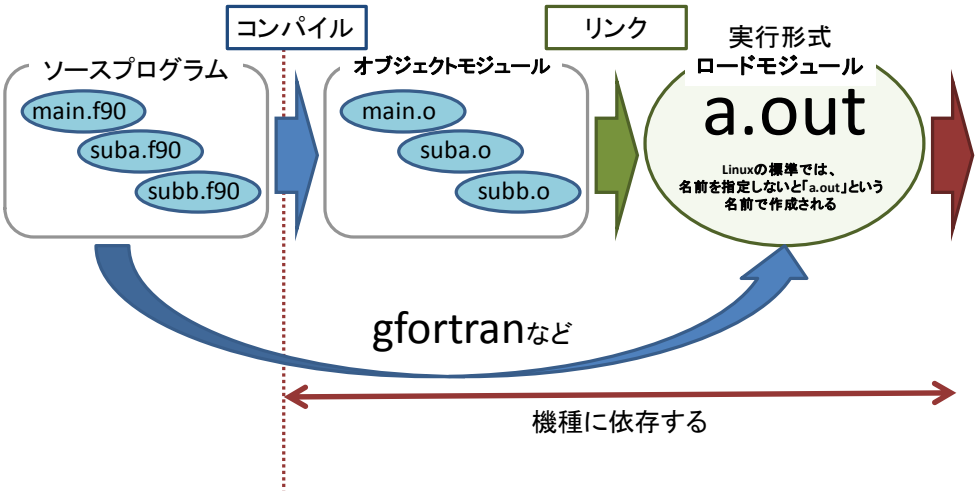


図 3-1　コンパイルの概念

Linux では、図 3-1 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するようにすると、make というツール(5.6 参照)を使うときに便利です。コンパイルすると「*.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。

一般に source.f90 というプログラムを作成し、Linux 上で gfortran source.f90 と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（soutce.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、何かよくないことがおこっています。エラーをきちんと調べて対処してくださいね。

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一般的に行われています。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。

3.2 デバッグしましょう

できたプログラムは、必ず正しく計算できているかどうか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も、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やめてください。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。

3.3 ＣＰＵ時間とエラプス時間

さて、計算機で計算させたときの時間には、CPU 時間とエラプス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コンピュータでは、「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悪いことが起こっていることも考えられます。概念を理解しておいていただきたいので、ここで説明することにします。

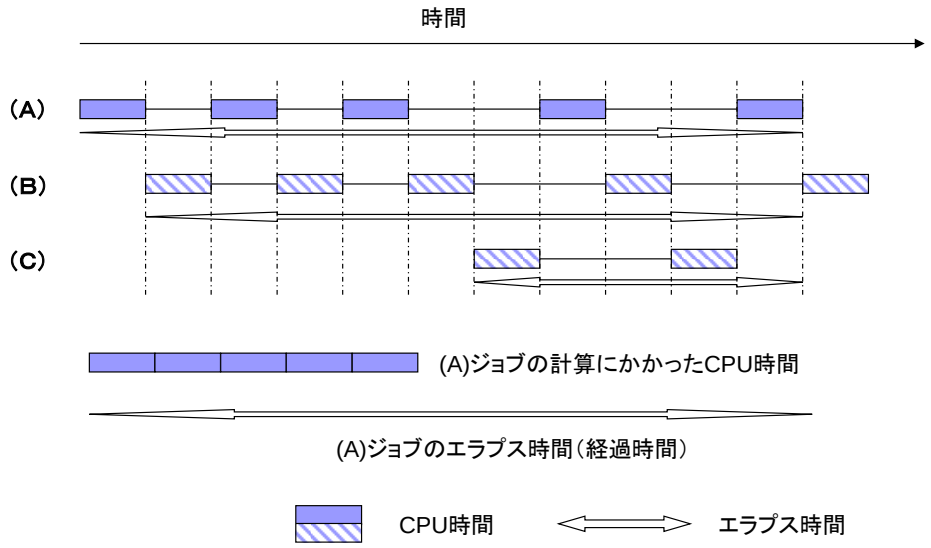


図 3-2 CPU 時間とエラプス時間

ここではひとつの CPU で３つの仕事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 3-2 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、一個ずつのジョブにかかるエラプス時間は長くなります

4　パソコン＆スパコンの動向とプログラミング（ベクトル＆並列の概念）

4.1 スパコンとは

「スパコンとは」の定義は時代とともに変化します。あまりのも多様化したので、「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 月に世界一となった「京」のシステムは、整備途中段階のものですが、最終的には 1 つの筐体に 96 個の CPU がのり、それを 864 台並べ合計 83000 個の CPU の構成になる予定です。LINPACK（リンパック）と呼ばれるベンチマークでは、世界最高性能の 8.162 ペタフロップス（毎秒 8,162 兆回の浮動小数点演算数）を達成し、TOP500 リストの首位を獲得しました。しかし、このような超並列の計算機を実際の研究に使いこなすためには、並列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、人間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、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。）

4.2 ベクトル化とは

ベクトル化は簡単です。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、簡単なプログラミングで性能を発揮しますので、一度は勉強してください。イメージは、図 4-1 に示すようなもので、一度に計算するので速いというものですが、計算順序が異なることにより、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い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しておいてください。

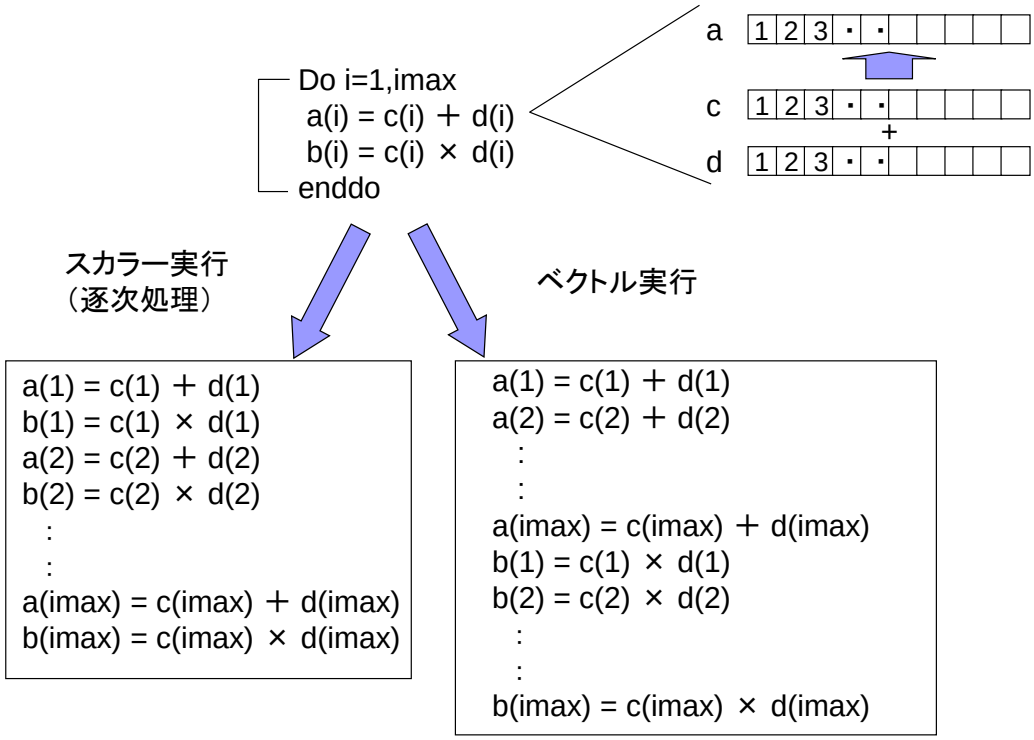


図 4-1　ベクトル化のイメージ

4.3 並列化とは

図 4-2 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が、CPU 時間は短くならず、エラプス時間を短くしようとするものです。

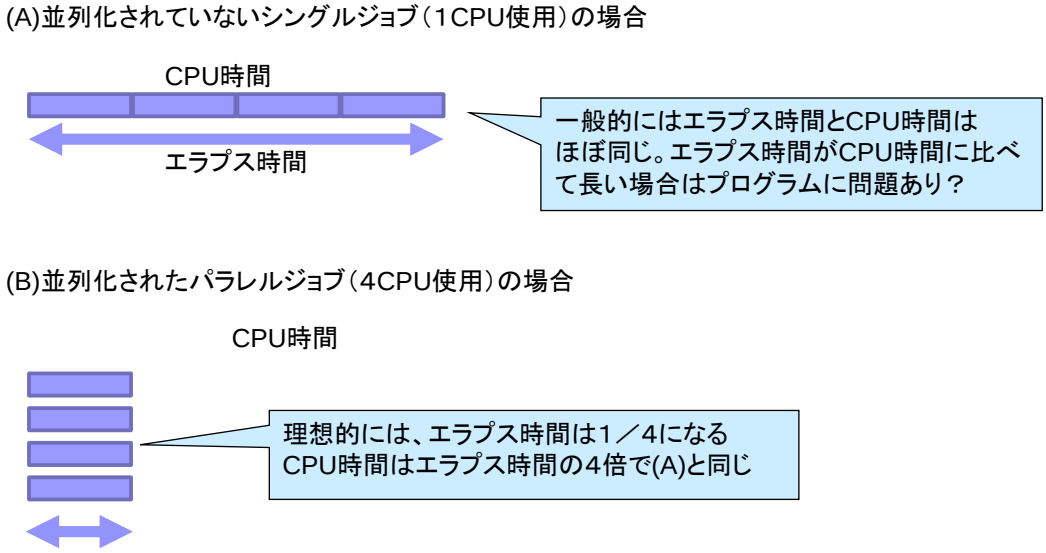


図 4-2　並列実行時の CPU 時間とエラプス時間

並列を考えるときには、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 4-3 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と呼びます。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 と呼ばれる標準的な指示文などで並列化できます。あとで説明する分散メモリに対応した MPI や HPF などで並列化することも可能です。

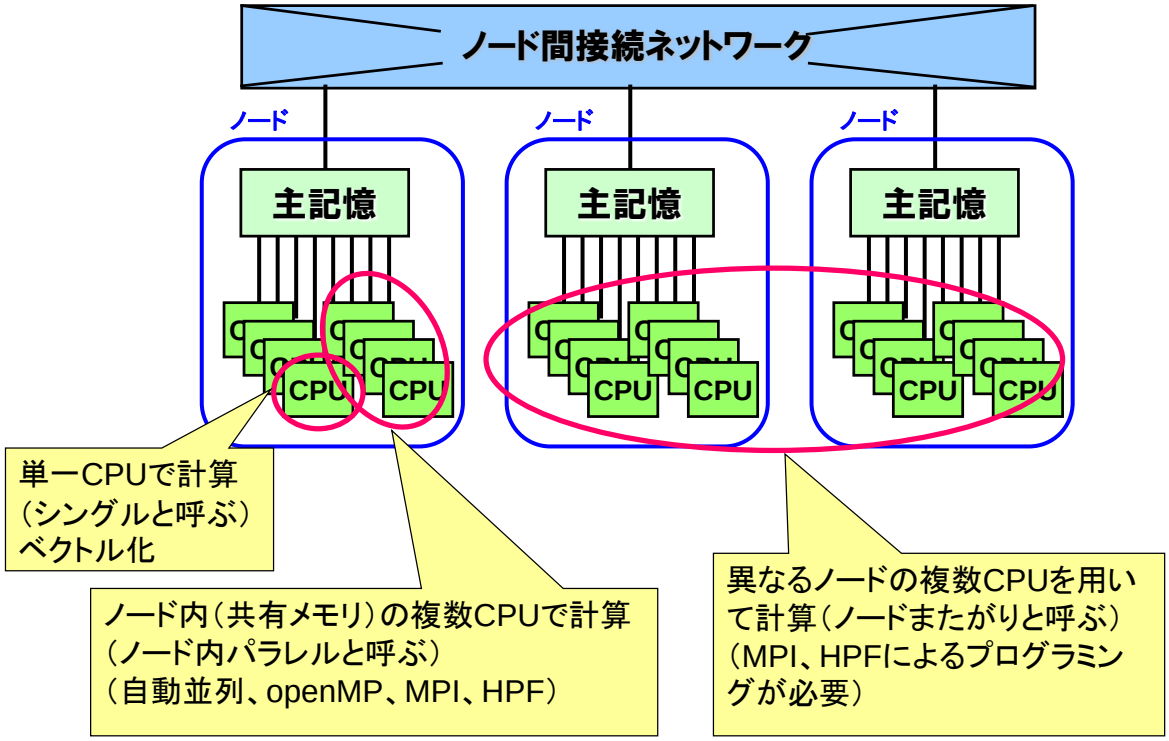


図 4-3　共有メモリと分散メモリ並列のプログラム方法

ノードをまたがり、分散メモリに対応した並列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。ただ、研究室単位でもクラスタが普及し、先輩から引き継いだプログラムが、いきなり MPI のプログラムという学生さんもおられますので、ここで紹介しておきます。

ノード内は自動並列や OpenMP で並列化し、ノードまたがりは MPI で並列するような場合は、ハイブリッド並列と呼びます。

4.4 ＭＰＩとＨＰＦとＸｃａｌａｂｌｅＭＰ

プログラムを並列化して高速性能を得るのは、かなりの労力が必要です。これを簡単にするために開発されたのが HPF(High Performance Fortran)と呼ばれる言語です。通常の FORTRAN プログラムに最小限の指示文を追加することにより、分散メモリ並列で簡単に性能がでることを目指して開発されました。ただ、あまりにも自動化を目指したために、うまく並列化で性能がでない場合のチューニングが難しかったこともあり、残念ながら普及はあまり進んでいない状況です。ただし、地球シミュレータセンターや阪大 CMC で使用できる HPF コンパイラ(HPF/SX V2)では、チューニングすべき場所を簡単に見つけられる機能が強化されていることもあり、現在はチューニングがかなり容易になってきています。HPF による分散並列化の普及活動を行っている HPF 推進協議会（HPFPC）のホームページでは、フリーの HPF コンパイラ（fhpf）やサンプルプログラムが公開されていますので、これらをダウンロードして試してみることもできます。2011 年 3 月には、「PC クラスタで並列プログラミング(High Performance Fortran で楽々並列化)」という本も出版され、並列化の基礎から fhpf のインストール方法、実際のプログラムの並列化方法まで含めて説明されています。HPF は、並列の初心者用の入門としてもよいと言われていますし、地球シミュレータセンター上では、実プログラムの HPF による高速計算の研究成果が、2002 年 Gordon Bell Award 言語章を受賞しており、簡単にノードにまたがった並列化ができるので、根強い利用者がおられるとのことです。図 4-4 に、簡単な Fortran プログラムを MPI と HPF で分散メモリに対応できる並列プログラムに書きかえた例を示します。HPF の場合は!HPF 指示文を追加しているだけですが、MPI の場合には、SEND や RECV を逐一指定している様子がわかります。

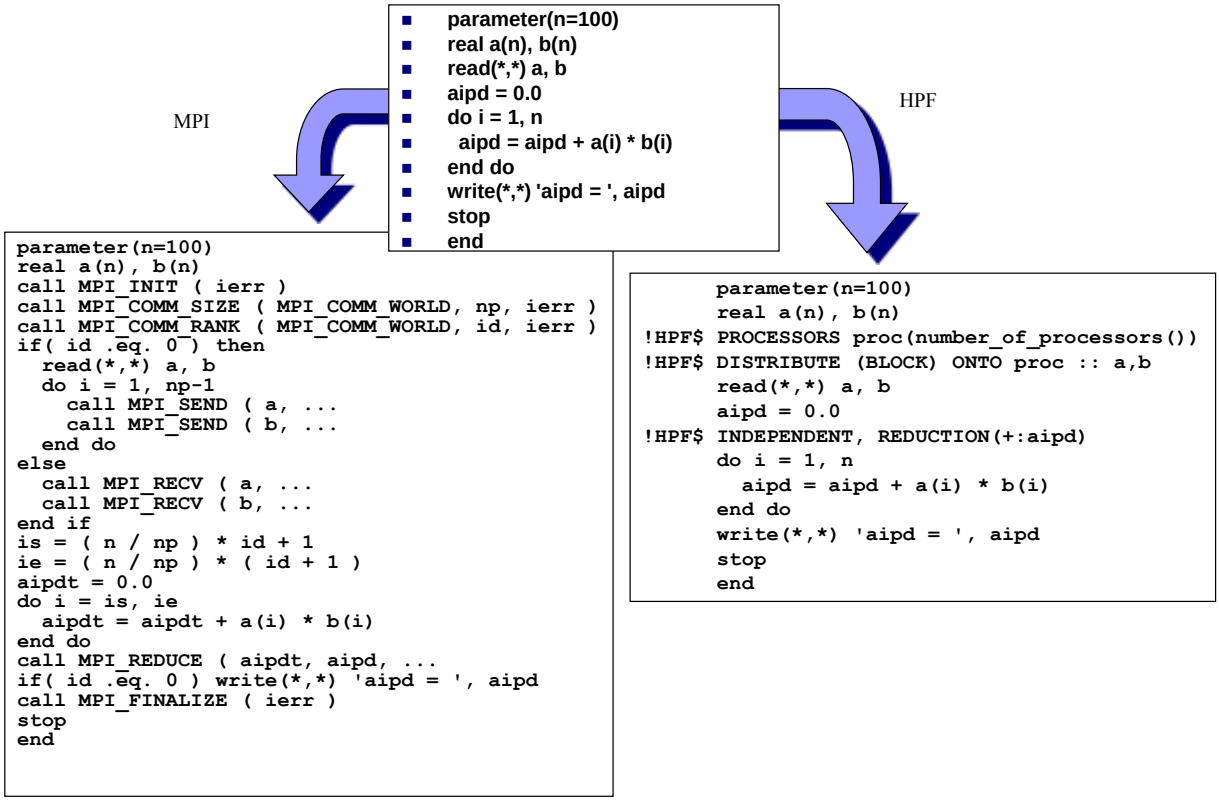


図 4-4　MPI と HPF の比較

阪大レーザー研では、2007 年に HPF 推進協議会の協力で HPF の講習会を行いました。ほとんどの参加者がすぐにコードを書き換えて HPF で動かすことができましたが、今のところ定着はしていません。まだ、大規模な計算をするのにどうしても分散並列に対応したプログラミングが必要という時代にはなっていないということもあり、ほとんどの研究者はベクトル化と自動並列で研究を進めておられます。ただし、ベクトル化と自動並列だけではすまない大規模な計算をする人は MPI でプログラミングされていますが、プログラミングだけでなく、デバッグやプログラムの改造の際にもいろいろ苦労されていますので、規則的なデータ構造のプログラムであれば、一度 HPF を試してみられるとよいと思います。興味のある方は、レーザー研のホームページの公開テキスト「HPF 応用編２　HPF/SX　V2 プログラミング」などには、HPF/SX V2 を使ったチューニング方法が解説されていますので参考にしてみてください。

HPF の経験を生かして、超スケーラブルな並列化を支援する言語として「XcalableMP」という並列言語の開発も進められているそうです。FORTRAN 版も

作成し、「京」で使用される予定もあるそうです。今後の超並列スパコンについての動向についての予想は難しいですが、まだまだこれからという状況もあります。HPF から XcalableMP への移行は簡単ということですし、分散メモリ対応の並列の入門としては、HPF を検討する価値はあると思います。

5　知っていると便利（シミュレーションに最低必要な Linux コマンド）

スパコンの OS は UNIX をベースにしています。UNIX がわからないので、使えないという方もおられますが、UNIX は一度覚えるととても簡単で便利です。最近では、Linux のほうが接する機会が多いと思いますが、利用者としては、それほどたくさん知らなくても、大丈夫です。ここでは、最低これだけ知っていればなんとかなるというコマンドをご紹介します。

Linux,UNIX といっても、実はいろいろな種類があります。システム管理者にとっては、設定ファイルが異なるなどいろいろ違いがあるのですが、利用者にとっては基本の使い方はほとんど同じです。ここでご紹介するようなコマンドは非常に一般的ですので、どのような Linux でもほとんどそのまま使えます。なるべく基本的なコマンドを使うことで、システムが異なっても同様に使えるというメリットがあります。

5.1 ディレクトリとファイル

Linux を使うためには、ディレクトリとファイルを理解しておく必要があります。Linux のファイル構造は分類、整理して、管理しやすいように、「階層構造」になっています。ファイルとは、書類のことで文書、プログラム、データなどのことです。

ディレクトリとは、Windows でいうフォルダのことで、ファイルやディレクトリを格納します。それぞれのディレクトリの間は自由に移動することができます。階層の一番上のディレクトリを「ルートディレクトリ」、ログインした時に最初にいるディレクトリを「ホームディレクトリ」と呼びます。ディレクトリ構造や、ホームディレクトリはシステム管理者によって決められています。

自分のホームディレクトリの下は、自由にディレクトリやファイルを作ったり消したりすることができます。自分で分かりやすいように名前をつけ管理するようにしましょう。また、あとで述べるパーミッションに気をつけましょう。基本的には、自分しか「読めない、書けない、実行できない」という設定にします。

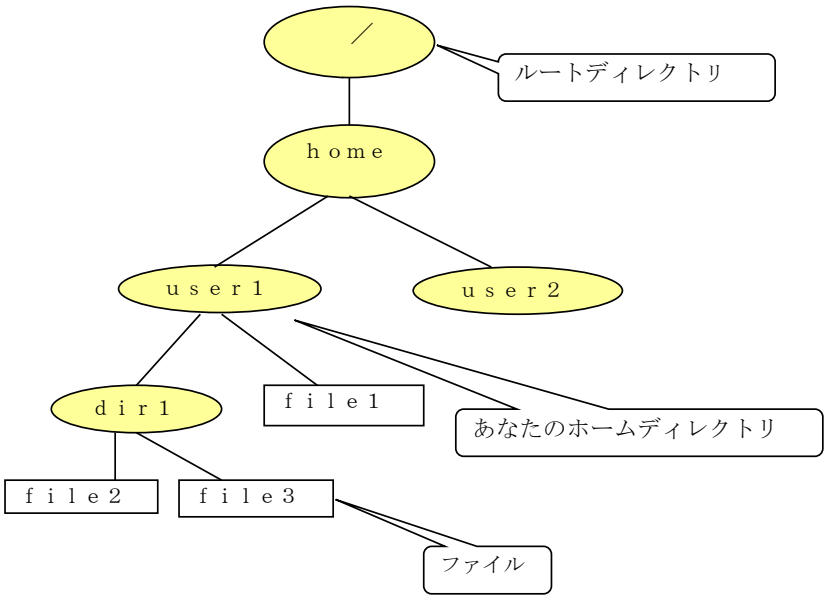


図 5-1　ディレクトリの概念図

Linux ではファイルはディレクトリとファイル名で指定します。たとえば図 5-1 で、ホームディレクトリの下の dir1 の下のファイル file は

　　／ｈｏｍｅ／ｕｓｅｒ１／ｄｉｒ１／ｆｉｌｅ

という指定をします。これを「絶対パス」と呼びます。

／ｈｏｍｅ／ｕｓｅｒ１というホームディレクトリからは

　　ｄｉｒ１／ｆｉｌｅ

という指定方法も可能です。これを「相対パス」と呼びます。

［／］から始まると絶対パスでルートディレクトリからの指定、

［／］がなければ、現在いるディレクトリ（カレントディレクトリ）からの指定となります。

5.2 ディレクトリの操作・表示に関するコマンド

コマンド名	概略
ｐｗｄ	Ｐｒｉｎｔ　Ｗｏｒｋｉｎｇ　Ｄｉｒｅｃｔｏｒｙ カレントディレクトリの表示（今どこにいるの？）
ｃｄ	Ｃｈａｎｇｅ　Ｄｉｒｅｃｔｏｒｙ ディレクトリの移動
ｍｋｄｉｒ	ＭａＫｅ　ＤＩＲｅｃｔｏｒｙ ディレクトリの作成
ｒｍｄｉｒ	ＲｅＭｏｖｅ　ＤＩＲｅｃｔｏｒｙ ディレクトリの削除
ｌｓ	ＬｉＳｔ　（リスト） ディレクトリの内容をリスト出力する

表 5-1

使用例）

・ カレントディレクトリを表示する
　ｐｗｄ
・ カレントディレクトリ下のｗｏｒｋディレクトリへ移動する
　ｃｄ　ｗｏｒｋ
・ ホームディレクトリへ移動する
　ｃｄ
・ カレントディレクトリ下へｗｏｒｋディレクトリを作成する
　ｍｋｄｉｒ　ｗｏｒｋ

5.3 ファイルの操作・表示に関するコマンド

コマンド名	概略
ｃａｔ	ｃｏｎＣＡＴｅｎａｔｅ　ファイルの内容の表示
ｍｏｒｅ	ＭＯＲＥ　　ファイルの内容を画面単位にとめながら表示 （続きを見るのはスペース、終了はｑ）
ｖｉ	ＵＮＩＸの標準的な画面エディタ どこへいっても使えます
ｃｐ	ＣｏＰｙ　ファイルのコピー

ｍｖ	ＭｏＶｅ　ファイル名の変更
ｒｍ	ＲｅＭｏｖｅ　ファイルの削除

表 5-2

使用例）

・ カレントディレクトリのｆｉｌｅの内容の表示
　ｃａｔ　ｆｉｌｅ
　ｍｏｒｅ　ｆｉｌｅ
・ カレントディレクトリの file を file2 という名前でコピー
　ｃｐ　ｆｉｌｅ　ｆｉｌｅ２
・ /home/user2/sample をカレントディレクトリの file という名前でコピー
　ｃｐ　／ｈｏｍｅ／ｕｓｅｒ２／ｓａｍｐｌｅ　ｆｉｌｅ
・ カレントディレクトリの file を sample に名前を変える
　ｍｖ　ｆｉｌｅ　ｓａｍｐｌｅ
・ カレントディレクトリの file を削除する
　ｒｍ　ｆｉｌｅ

5.4 ファイルのアクセス許可に関するコマンド

コマンド名	概略
ｌｓ	ＬｉＳｔ　（ｌｓ　－ｌとするとファイルのオーナー、アクセス権などの詳細が表示されます）
ｃｈｍｏｄ	Ｃｈａｎｇｅ　ＭＯＤｅ ファイルやディレクトリのアクセス許可の変更

表 5-3

Linux では、ファイルやディレクトリに対して

読み取り許可（Read）
書き込み許可（Write）
実行許可（eXecute）

の３つの権利を、利用者自身、同じグループの人、その他の人に対してそれぞれ設定されています。ｌｓ　－ｌとすると「ｄｒｗｘｒｗｘｒｗｘ」の

ように表示されます。「ｄｒｗ-------」に表示される最初の１文字がｄはディレクトリであることを示し、－ならファイルです。その次は３個ごとにｒｗｘならそれぞれ Read 権、Write 権、実行権があり、―なら権利がないことを示します。

（ls -l の出力例）

```
-rw-------    1 user    grp          21  9月 16 日 13:58 filea
-rwx------    1 user    grp          21  9月 16 日 13:58 fileb
-rw-r--r--    1 user    grp         276  9月 16 日 13:58 filec
drwx------    3 user    grp         512 12月 25  2003 dir1/
drwxr-xr-x    3 user    grp         512 12月 25  2003 dir2/
```

上記の出力例の

ｆｉｌｅａは自分のみ、「Read」「Write」可能

ｆｉｌｅｂは自分のみ、「Read」「Write」「実行」可能

ｆｉｌｅｃは自分のみ、「Read」「Write」可能、
　　同じグループとその他の人は「Read」のみ可能

ｄｉｒ１は自分のみ「Read」「Write」「実行」可能
　　そのディレクトリに移動して新規にファイルやディレクトリを作成可能

ｄｉｒ２は自分は「Read」「Write」「実行」可能
　　その他の人は、そのディレクトリに移動はできるが、新規ファイルの作成はできない。

ｒｗｘをそれぞれ許可を１、禁止を０で表すことができます。

－－－は	０００	８進数では０
ｒ－－は	１００	４
ｒｗ－は	１１０	６
ｒｗｘは	１１１	７

となり、以下のようにアクセス許可を変更できます。

使用例）

・ ｆｉｌｅａのアクセス許可を rwxr--r--に変更する
　ｃｈｍｏｄ　７４４　ｆｉｌｅａ
・ ｆａｉｌａのアクセス許可を rw-------に変更する

ｃｈｍｏｄ　６００
ｍｏｒｅ　ｆｉｌｅ

5.5 その他の便利なコマンド

コマンド名	概略
ｍａｎ	ＭＡＮｕａｌ　マニュアルの表示
ｇｒｅｐ	文字列の検索（とっても便利！） Global Regular-Expression Print
ｈｅａｄ	ＨＥＡＤ　ファイルの先頭の表示
ｔａｉｌ	ＴＡＩＬ　ファイルの末尾の表示
ｓｅｄ	Ｓｔｒｅａｍ　ＥＤｉｔｏｒ　文字列の一括編集
ａｗｋ	パターン検索と処理のための言語。ｓｅｄと組み合わせていろいろな編集処理が可能

表 5-4

使用例）
・ｆｉｌｅａの中で、文字列“error”を含む行を出力する
　ｇｒｅｐ　ｅｒｒｏｒ　ｆｉｌｅａ
・ ｇｒｅｐの使い方を調べる
　ｍａｎ　ｇｒｅｐ

5.6 ｍａｋｅ

プログラムをコンパイルするのには、gfortran などのコマンドを利用すると説明しましたが、Linux では「make」というツールを利用するとプログラムの管理が非常に便利です。ｍａｋｅには以下のような利点があります。

・ ”make”と入力するだけで、コンパイルとリンクを行う。
・ 修正したファイルのみを自動的に再コンパイルするので、2 度目以降のコンパイルは時間が短縮できる。(変更したファイルのみを自動的にコンパ

イルする）
・ プログラムの管理が容易

などです。3.1 で説明しましたように、プログラムは、機能別に複数のファイルから１つのプログラムが形成されています。「makefile」によって、複数のファイルを管理します。「make」は奥が深いですし、市販の本も複数出版されていますので、興味のある方は勉強してください。実際には、研究室で先輩から引き継ぐ場合も多いでしょうから、そんなものがあるということは知っておいてください。

6　スパコンセンターなどを利用するための基礎知識

ここからは、大阪大学サイバーメディアセンター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。

6.1 システム全体のメモリ管理

システムによって、考え方も方式も異なりますが、スパコンでは効率を重視し、通常、スワップが起こらないようにプログラムを実行するように設定されている場合が多いです。CPU だけでなくメモリも貴重な資源であり、有効にシステム全体を活用するように工夫して運用されています。

図 6-1 は、トータル 64GB のメモリが実装されていて、A と B の 2 つのジョブが各々30GB、20GB とあわせて 50GB を使うと宣言している場合を示しています。この場合、実際には 20GB しか使っていませんが、システムが空きとして認識している 14GB 以上を使うと宣言したジョブは実行されません。

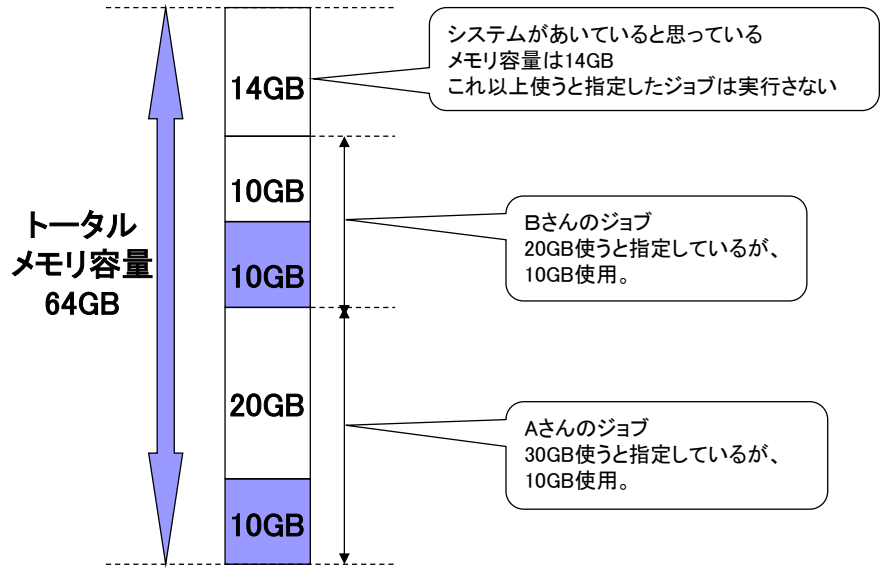


図 6-1　システム全体のメモリ容量とジョブのメモリ容量

大きすぎる値を指定すると自分も他の人も困ります。コンパイルしたあとで調べた LM サイズで調べた目安や、SX で実行した場合は PROGINFO で表示される実際に使用したメモリサイズに少し余裕をもたせたサイズを指定するようにしてください。動的アロケートを利用して大きなメモリを必要とするかもしれないプログラムもありますので、どれだけのメモリを必要とするかは、利用者の方でないと分かりません。適切にシステムに教えてあげるようにしましょう。だからと言って、あまりにぎりぎりの値を設定すると、ほんの少し超えただけでもジョブがアボートするかもしれませんので、ご注意ください。計算機は融通がききません。最近は、ノードを占有するなど、複数のジョブでメモリを共有する場合は少なくなっていると思いますが、利用者が正しく指定して、必要量を計算機に教えてあげることは重要です。

6.2 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 6-2 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、スパコンを使うために用意されている端末のことです。

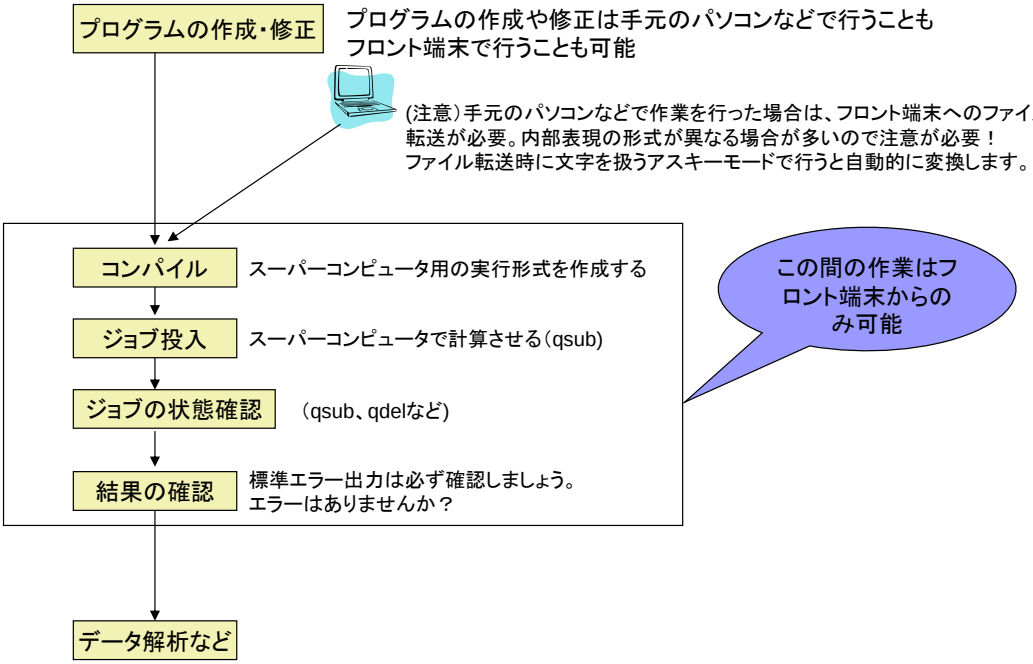


図 6-2　スーパーコンピュータ利用の流れ

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。

6.3 リモートログインとファイル転送

6.2 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」、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阪大 CMC にログインすることができます。

```
ssh username@login.hpc.cmc.osaka-u.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がちょっと手間かもしれません。

6.4 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、3.1 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。このような使い方を**セルフ環境**と呼びます。しかし、スパコンはロードモジュール高速実行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

阪大 CMC ではフロントと呼ばれる端末が 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 6-3 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。

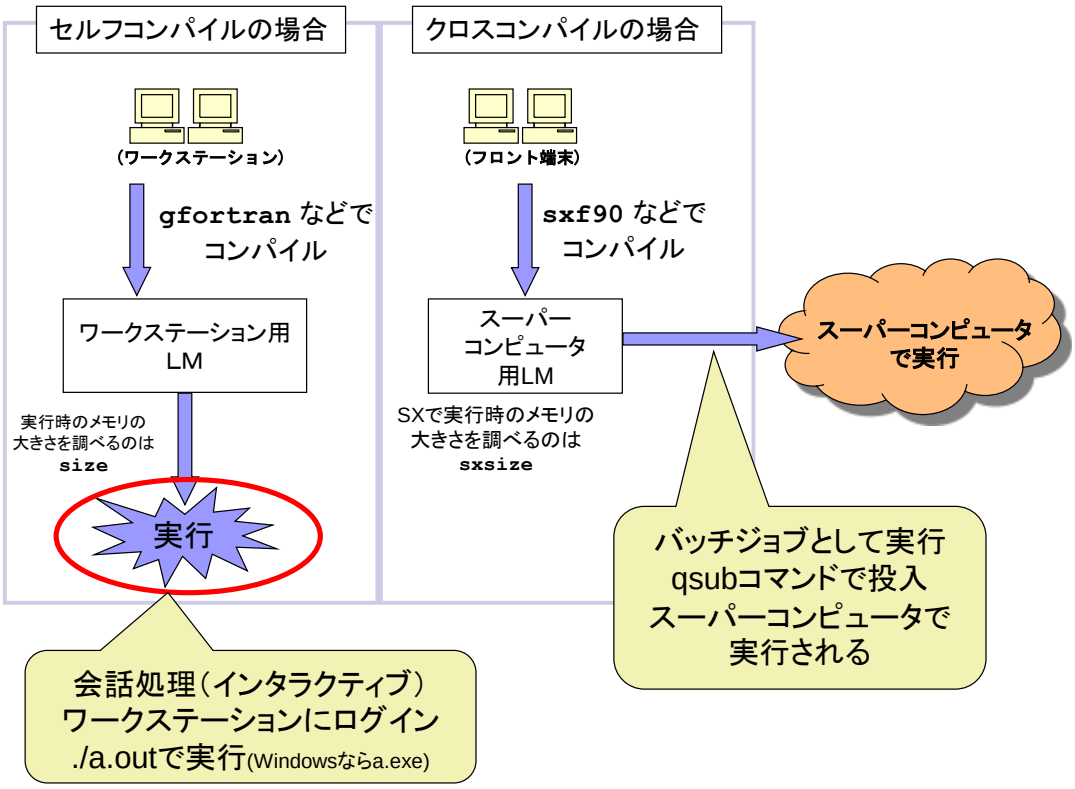


図 6-3　セルフコンパイラとクロスコンパイラ

無事にコンパイルが終わったら、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。ベクトル化、並列化、最適化などを行います。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いじったのかを調べることができます。

また、実際に実行させなくても、図 6-3 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。

6.5 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）

それぞれ、表 6-1 に示すような特徴があります。

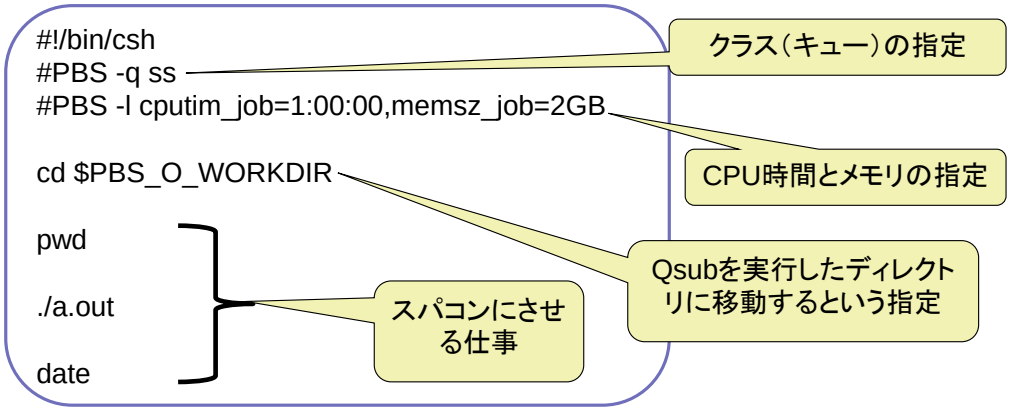
	会話処理	バッチ処理
実行のさせ方	計算機にむかってコマンドを入力	コマンドなどを書いて、システムにわたす
いつ実行されるか	コマンドを入力するとすぐに実行される	計算機システムが状況に応じて実行する
端末の画面	占有され、ログインしたままにしておく	計算機システムにまかせておけばよいので、ログアウトして帰ってもよい
計算機全体の効率	考慮できない	考慮できる

表 6-1　会話処理とバッチ処理

6.4 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。

「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。このテキストでは、この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。このようなファイルは、それぞれのセンターの指示に従って作成してください。



6.6 適切なクラス（キュー）を決定する。

1 章にも記述しましたが、スーパーコンピュータで計算させる場合は、計算時間（CPU 時間）だけでなくメモリ容量も重要なファクタとなります。計算に必要な CPU 時間、CPU の数、メモリの容量などにより、システムで決められたクラス（キューとも呼びます）を指定してジョブを投入するようにしてください。ジョブの特性によってなるべく効率よく計算を行い、システム全体の効率を高め、利用者全体がハッピーになるようにしたいと考えていますが、そのためには、利用者が適切なクラスにジョブを投入することが必要です。

その考え方は、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指定するようにしてください。

6.7 いよいよスーパーコンピュータで計算させる（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、いよいよスーパーコンピュータに計算させます。図 6-4 は NQS を用いたジョブ実行の概念を示します。①はフロント端末から、用意した NQS ファイルを qsub コマンドを使って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ト id と呼ばれる番号をつけます。この例では、999.cmc 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 6-3 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。

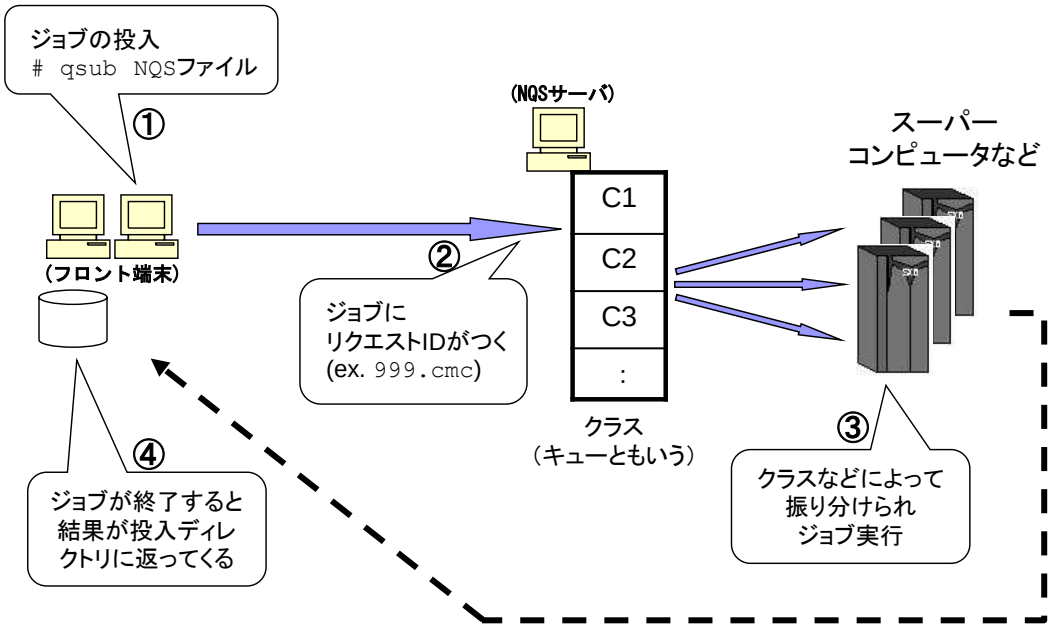


図 6-4　NQS を用いたジョブ実行の概要

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*

表 6-3　よく使う NQS コマンドの例

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よくないことがおこる場合があります。大容量の出力はファイルにするようにしてください。標準エラー出力は、ゼロで割った（ゼロディビ）とか、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（オーバーフロー）などのエラーが表示されますので、

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、確認するくせをつけましょう。

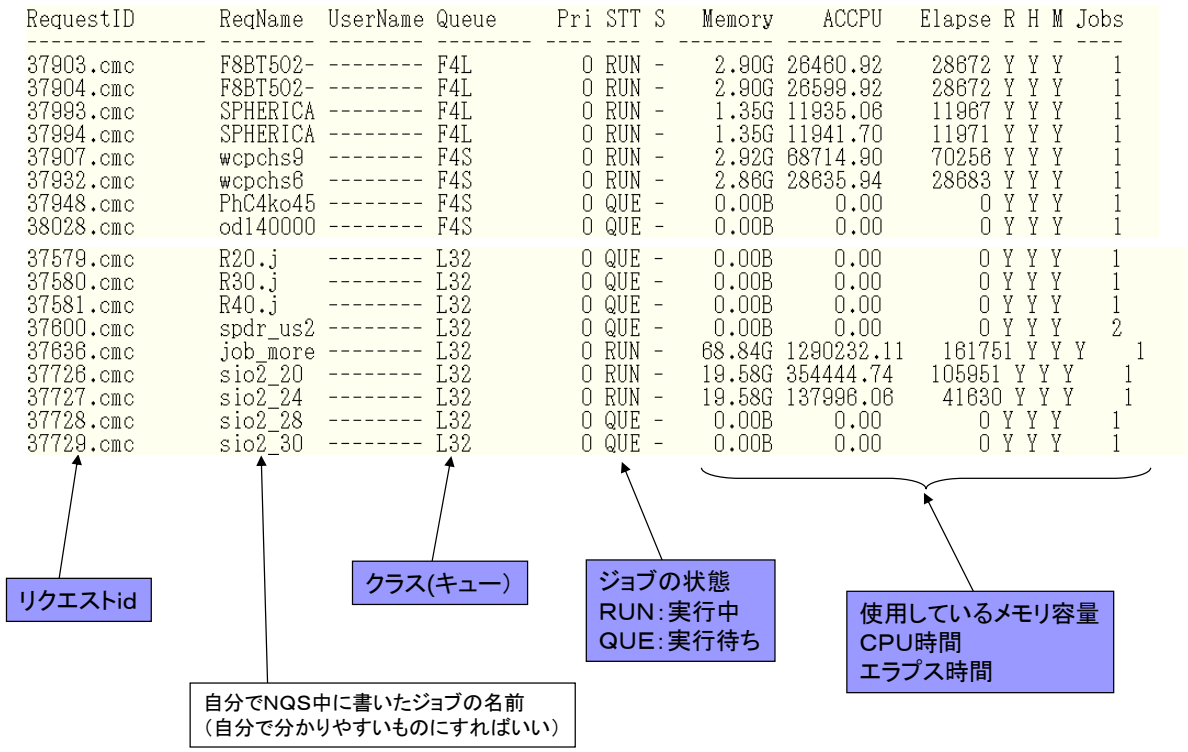


図 6-5　スーパーコンピュータで実行中ジョブの確認例

標準の qstat コマンドでは、自分のジョブの状態しか表示されませんが、阪大 CMC とレーザー研では、jobr というコマンドが用意されており、システムで実行中の全てのジョブの状態を表示させることができ、図 6-5 はその表示例です。コマンドを入力した時点の、各々のジョブのリクエスト id やクラス、使用しているメモリ容量、CPU 時間、エラプス時間などが表示されます。たくさんのジョブが実行中（RUN）や待っている（QUE）ことがわかります。

6.8 ディスク構成を知り、正しく工夫してディスクを使いましょう

スパコンのような大規模なシステムでは、通常プログラムや NQS ファイル、小さな入力データなどを保存するための領域と、スパコンで計算したときに

出力される膨大な出力データを保存するための領域は別に用意されています。また、多人数で共有するのですから、一人当たり使用できるディスク容量もｑｕｏｔａという機能を用いて制限されています。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、スーパーコンピュータのディスクに、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。以下のような方法がありますが、時代とともに状況は変化します。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 ファイルなどの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ランできるように、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 桁）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。間違ってもスーパーコンピュータに compress や gzip の仕事

をさせないで下さい。
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。

6.9 リランとリスタート

長時間のジョブを実行させようとする方は、リランとリスタートについて知っておいて下さい。

リランとは：
スパコンでは、何かの障害があってジョブが中断してしまった場合は、通常は、自動的に最初から計算をやり直す設定になっています。実行していたジョブがアボートし、最初からやり直すことをリランと言います。せっかく途中まで計算していたのにリランになることで、ファイルが上書きされる場合もあります。それは困るという方は、リランしない設定でジョブを実行することができます。

リスタートとは：
実行中のジョブの状態を保存しておき（チェックポイント)、途中からジョブを再開することをリスタートと言います。チェックポイントはシステムに負荷がかかるのでむやみには行いません。
大規模な計算は、計算時間もかかる場合が多いですが、最近のセンターでは多数の人が公平に使うために、あまりに長時間のジョブは許可されない場合があります。長時間の計算が必要な方は、メモリ上の計算を継続するために必要なデータを、自分で適宜ディスクにファイルとして出力して保存し、そのファイルを読み込むことで計算を続行できるようにされています。ただ、メモリが大きいプログラムの場合には、これを頻繁に行うとシステムに負荷がかかりますので、注意しましょう。

阪大CMCやレーザー研のSXでは､2日以上かかるようなジョブの場合には、システム側で自動的にチェックポイントをとり、障害によりジョブがアボートしたような場合には、チェックポイントを採取した時点からリスタートすることができます。このような事態に陥った場合は、システム管理者から連

絡がありますので、あわてないようにしてください。リスタートするためには、そのジョブが必要とするファイルがすべて残っていることが必要です。あわててジョブに関連するファイルを削除してしまうと、せっかく途中からリスタートできるはずだったのに、できなくなってしまうことがあります。

長時間ジョブに関連するファイルは消したりしないようにしましょう。

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？

7　付録

今までに質問の多かった項目について、以前のテキストで説明していたものなどです。もっと詳しい説明は田口先生のテキスト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。

7.1 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 7-1 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^{38} 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。

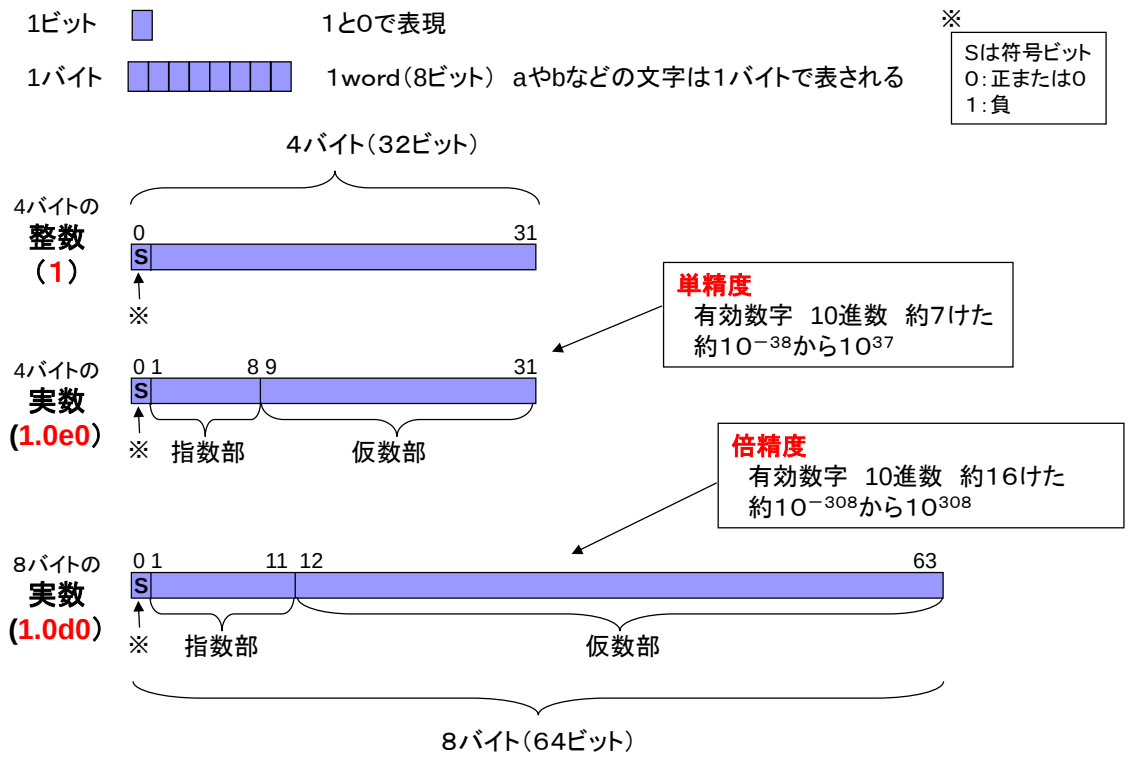


図 7-1　SX のデータ内部表現の例

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。

7.2 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、エディタで見ることができる。

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、以下のよう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 は外部装置番頭と呼ばれる番号で、ファイルを指定します。

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

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、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
```

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、以下のようにプログラムの中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, * ) a
100 format(1x,e11.4)
```

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）

7.3 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

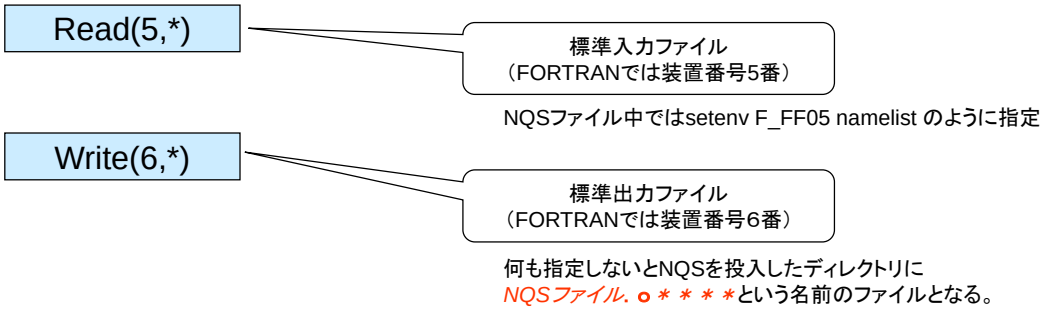


図 7-2　標準入出力と READ,WRITE 文の関係

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、スーパーコンピュータ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、「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。

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。

7.4 ビッグエンディアンとリトルエンディアン

スパコンで計算した結果を再びスパコンで使用する大容量のデータの場合は書式なしファイル（バイナリ）を用いるのがよく、データサイズが小さくパソコンに持っていっ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が、バイナリデータも互換性のある場合が多くなっています。容量の大きいデータを扱う方は、なるべくバイナリ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。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で解析するなどの場合に利用されている方法です。

- 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。
- FORTRAN の環境変数を利用。(SX もこの方法で変換できます)
- 種々fortran のコンパイルオプションを利用。

以上のようなものですが、機種によって異なる場合があるということを知っておくことが重要です。このような処理はスーパーコンピュータではなく、ワークステーションやパソコンでやるべきだと思っていますが、摂南大学の田口先生はスパコンで行っておられるそうです。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。

7.5 スカラ向きブロック化チューニング例

2.2 でループ長は長くなるようにしましょうと書きましたが、スカラマシンでは実際には、キャッシュに乗るか乗らないかで性能がかなり異なるため、必ずしも長い方がいいとは言えず、以下のように、キャッシュに収まるようにループ長を短くする（ブロック化）チューニング手法が知られています。しかし、私はわかりやすくプログラムを書くべきだと思っていますので、こんな風なチューニングをしないと性能がでないようなら、阪大 CMC の SX を利用されることをお勧めします。(特に初心者の方)

```
DO K = 1,10000
  DO J = 1,10000
    DO I = 1,10000
      A(J,K) = A(J,K) + B(J,I) * C(I,K)
    ENDDO
  ENDDO
ENDDO
```

↓

```
DO KK = 1,10000,1000
  DO JJ = 1,10000,1000
    DO II = 1,10000,1000
      DO K = KK,MIN(KK+1000-1,10000)
        DO J = JJ,MIN(JJ+1000-1,10000)
          DO I = II,MIN(II+1000-1,10000)
           A(J,K) = A(J,K) + B(J,I) * C(I,K)
          ENDDO
        ENDDO
      ENDDO
    ENDDO
  ENDDO
ENDDO
```