[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。

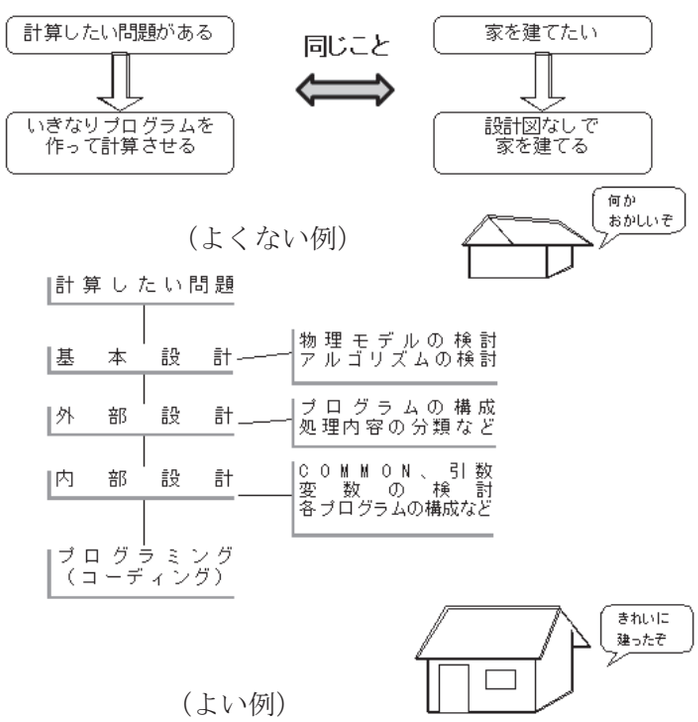


図１　プログラム作成の概念図

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。

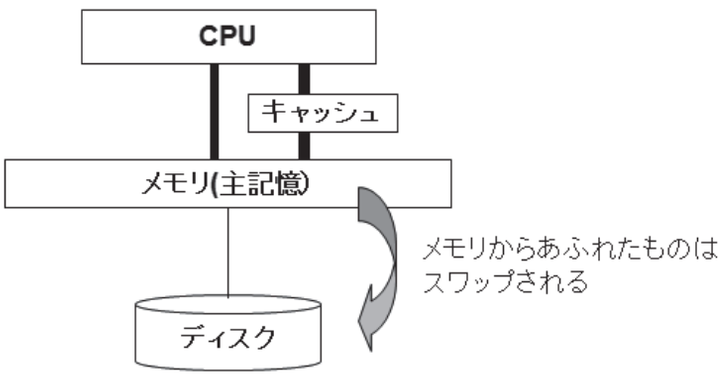


図 2　CPU とメモリとスワップの概念図

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。

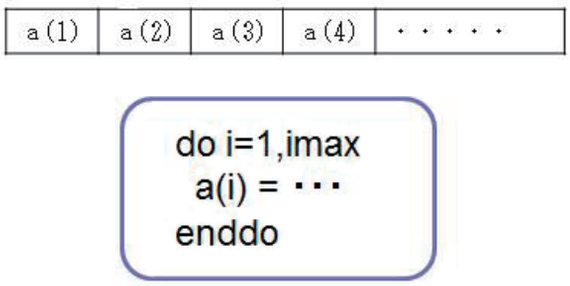


図 3　1 次元配列のメモリ上の配置（上）とプログラム（下）

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。

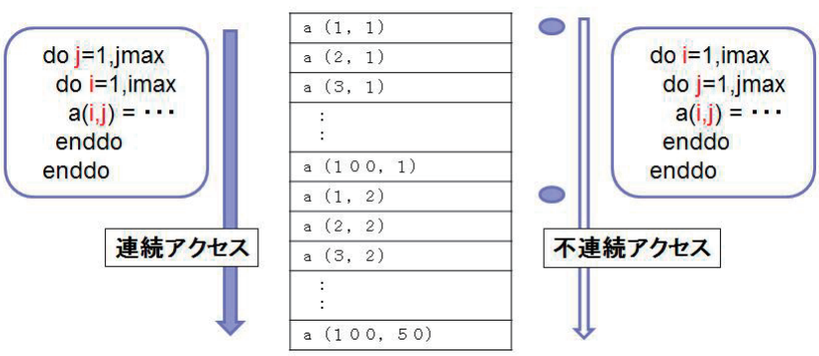


図 4　Fortran　2 次元配列のメモリ上の配置

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。

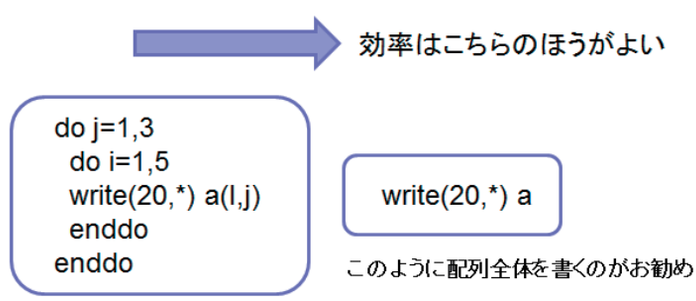


図 5　write 文の形式と効率

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。

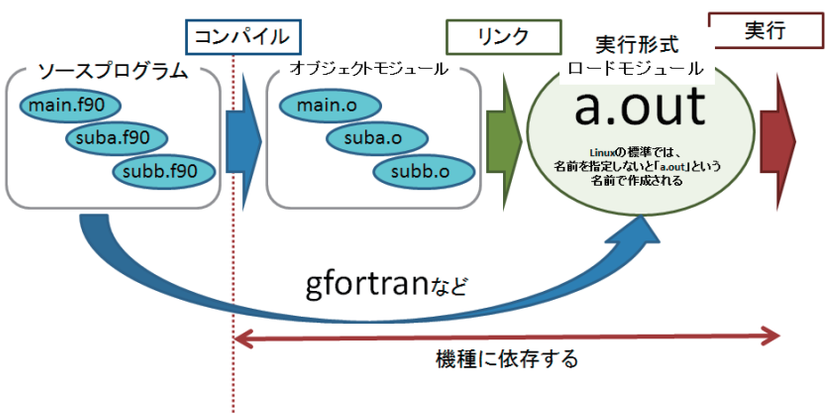


図 6　コンパイルの概念

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。

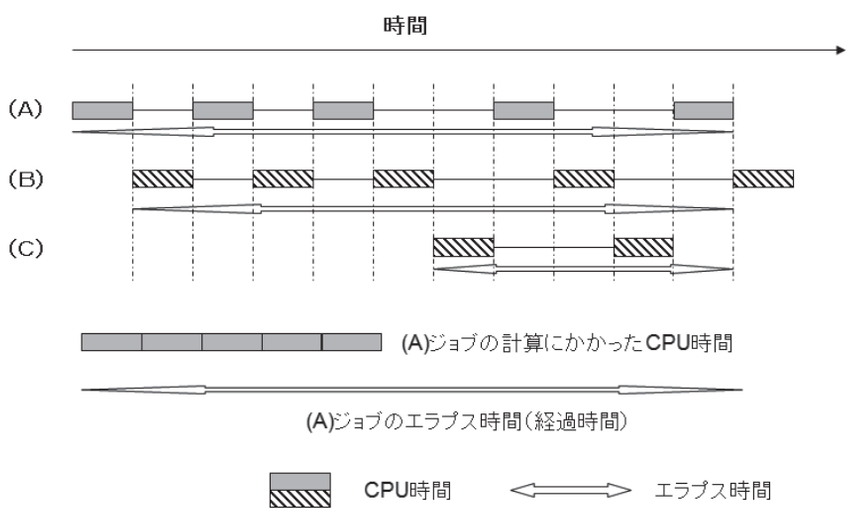


図７　CPU 時間とエラプス時間

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。

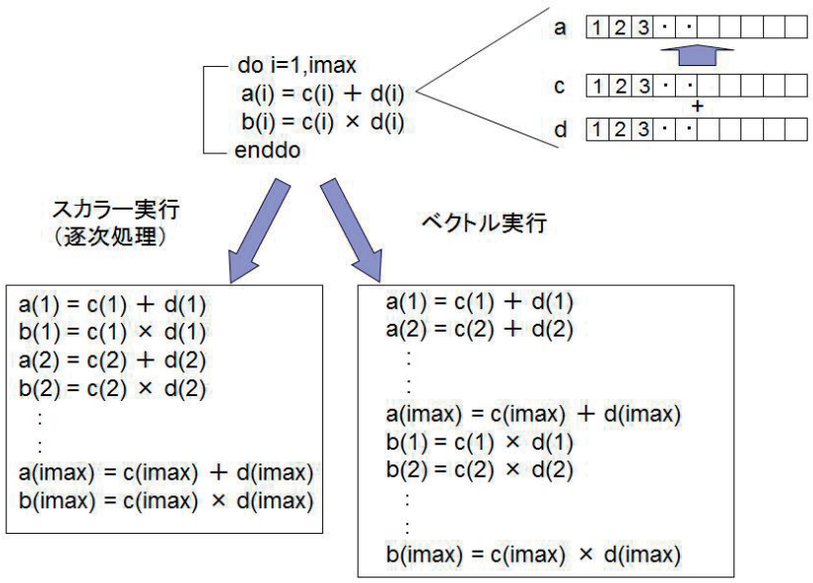


図 8　ベクトル化のイメージ

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。

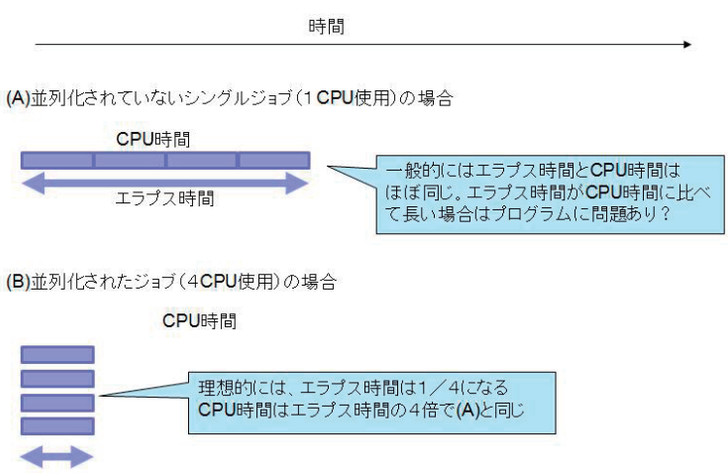


図 9　並列実行の CPU 時間とエラプス時間

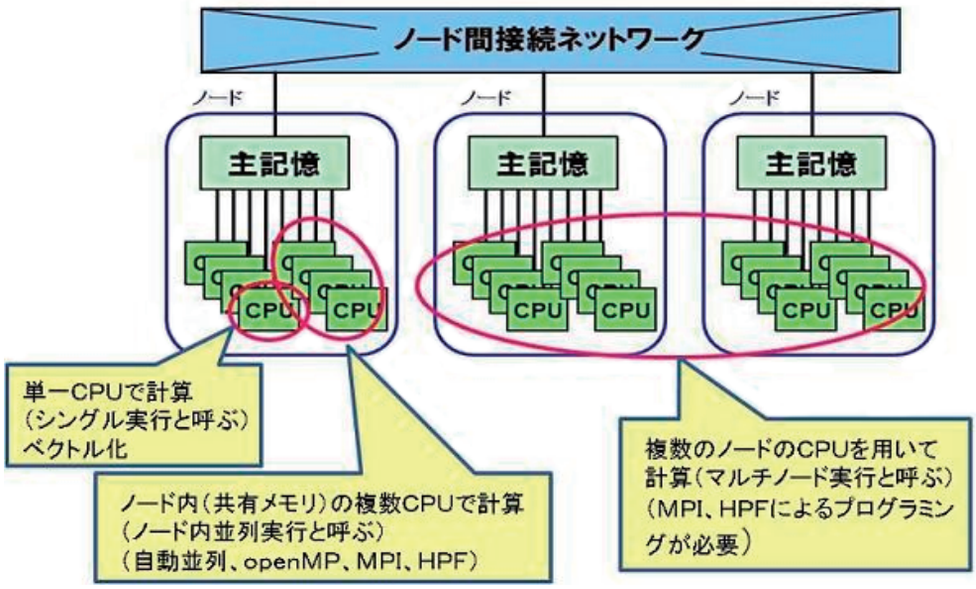


図 10　共有メモリと分散メモリ並列のプログラム方法

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。

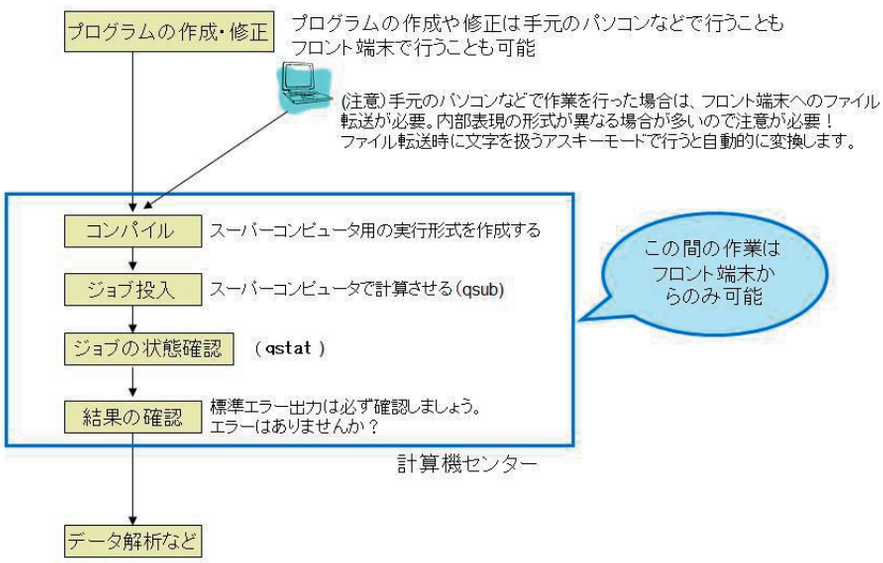


図 13　スーパーコンピュータ利用の流れ

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。

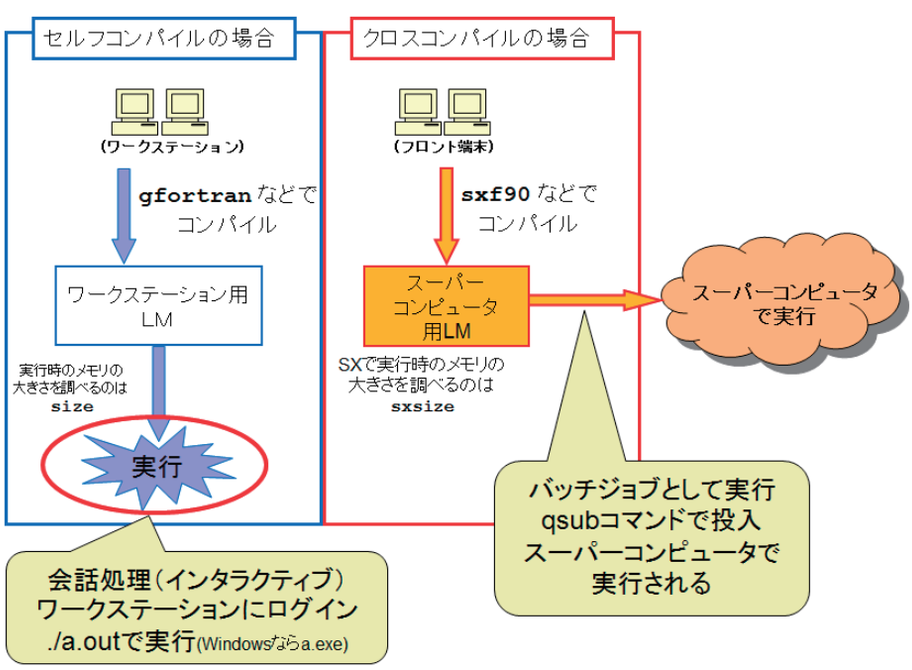


図 14　セルフコンパイラとクロスコンパイラ

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）

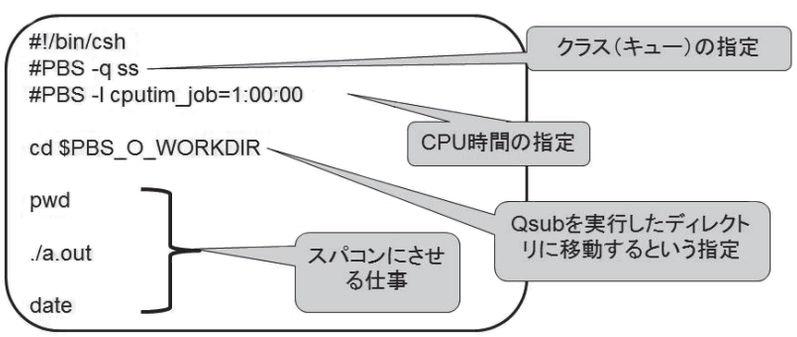
イメージは図 14 に、特徴は表 1 に示します。7.3 のクロスコンパイルにより、できたロードモジュールをスパコンで実行させるためには、バッチ処理（NQS2 など）を用います。

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

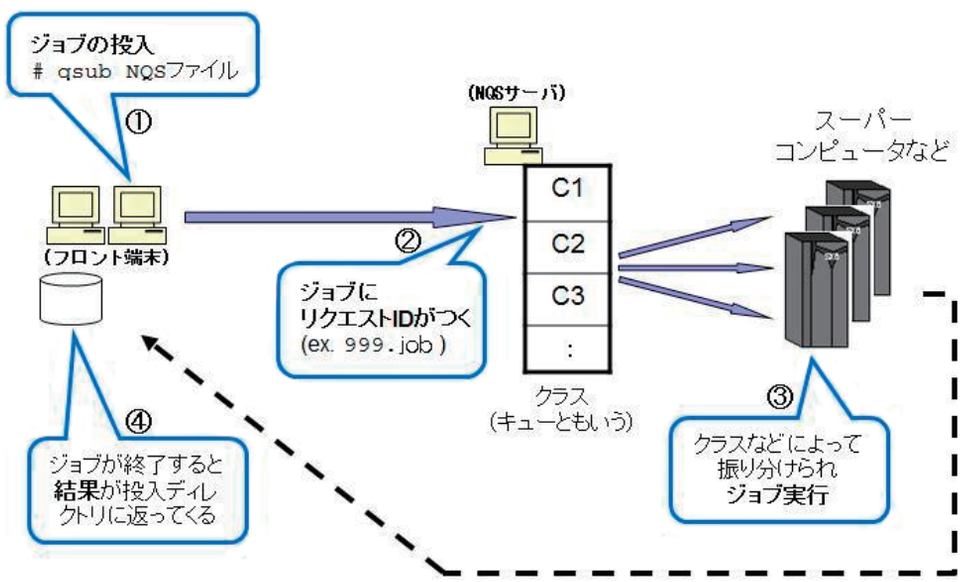


図 15　NQS を用いたジョブ実行の概念

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。

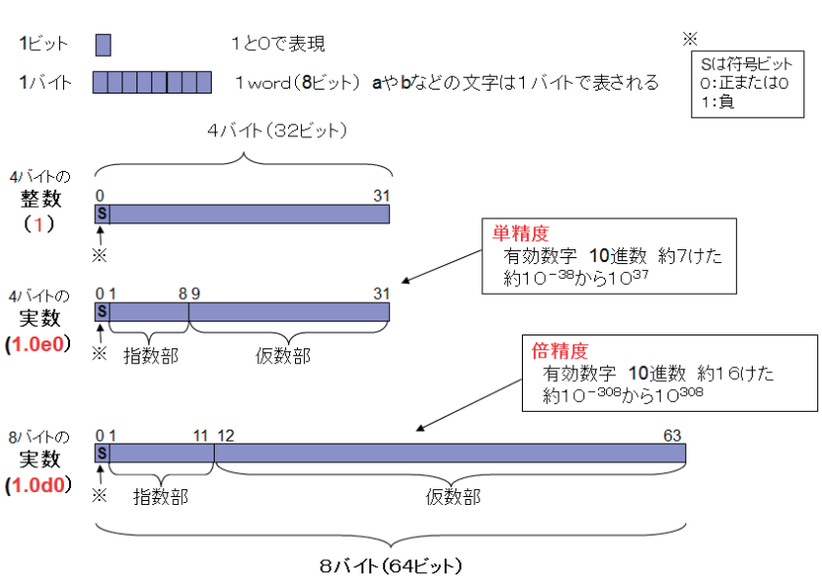


図 16　SX のデータの内部表現の例

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。

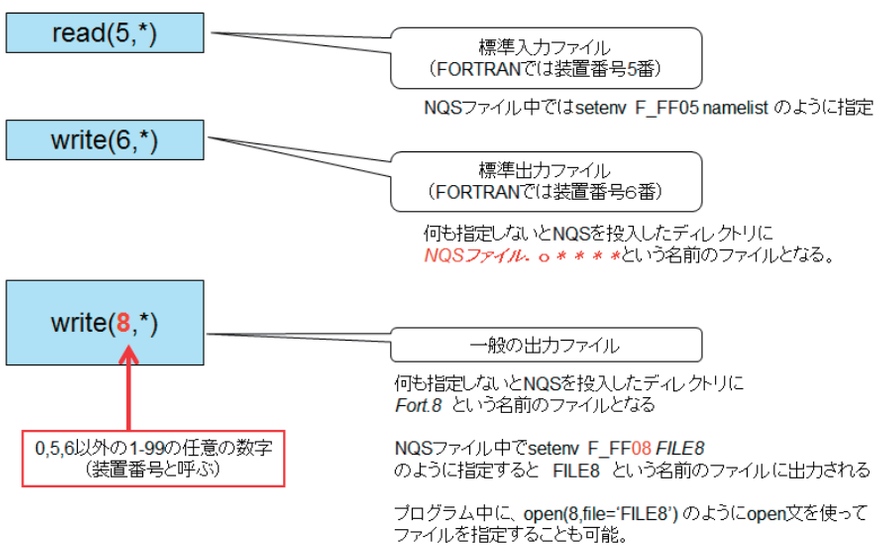


図 17　標準入出力と READ,WRITE 文の関係

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/