講義にスーパーコンピュータをとりいれてみませんか？：講義用無料アカウントを使った演習事例

南里，豪志
九州大学情報基盤研究開発センター

https://doi.org/10.15017/1470719

出版情報：九州大学情報統括本部ITマガジン. 1 (2), pp.25-41, 2007-12. 九州大学情報統括本部
バージョン：
権利関係：

講義にスーパーコンピュータをとりいれてみませんか? ～ 講義用無料アカウントを使った演習事例 ～

南里 豪志*

九州大学情報基盤研究開発センター (以下, 本センター) では, スーパーコンピュータを気軽に学生の皆さんに利用してもらえるよう, 今年の6月に稼働を開始した新スーパーコンピュータ Fujitsu PRIMEQUEST 580(図 1) のアカウントを学生向けに無料で発行するサービスを始めました. このアカウントは, 最大で 16CPU コア, 主記憶 22.4GB を利用した計算が行えるもので, 講義で利用する場合は講義を担当する教員用のアカウントも無料で配布します.

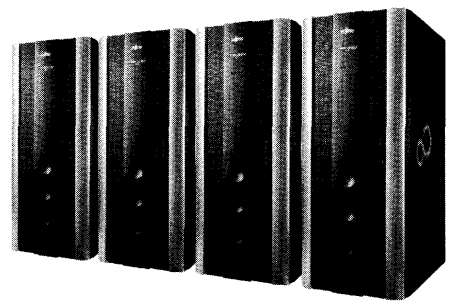

図 1: Fujitsu PRIMEQUEST 580

スーパーコンピュータと聞くと, 用途が限られた特別な計算機というイメージがあるかもしれませんが, 実際には様々な分野で広く使われており, 利用法も一般的な UNIX サーバとほとんど変わりません. 本稿では, 講義等でスーパーコンピュータを扱うことを想定して, スーパーコンピュータに関する一般的な話や具体的な利用法, さらに実際に筆者が講義で行った演習の事例等を紹介します.

1 スーパーコンピュータ四方山話

1.1 スーパーコンピュータとは何か?

スーパーコンピュータという名称は広く使われていますが, そもそもスーパーコンピュータとはどういう計算機か, という疑問があるかもしれません. 一般にスーパーコンピュータは, その時期に利用可能な最新の計算機技術を用いて, 出来るだけ高い演算性能を得られるよう設計, 構築された計算機, という感じで認識されていますが, 実はスーパーコンピュータとそれ以外の計算機と区別するための明確な定義はありません.

では, 実際のところ本センターのスーパーコンピュータシステムを構成している計算機群は, 皆さんが普段使われている PC(パーソナルコンピュータ) とどの程度違うものでしょうか? 本センターで運用している各計算機の仕様を表 1 に示します.

* 九州大学情報基盤研究開発センター
E-mail: nanri@cc.kyushu-u.ac.jp

表 1: 情報基盤研究開発センターの計算機群の仕様

	PRIMEQUEST	PRIMERGY	eServer p5	SR11000
CPU	Intel Itanium2	Intel Xeon	IBM Power5	IBM Power5+
クロック周波数	1.6GHz	3.0GHz	1.9GHz	2.3GHz
ノード内 CPU コア数	64	4	64	16
ノード内メモリ容量	128GB	8GB	128GB～512GB	128GB
総 CPU コア数	2048	1536	416	368
総メモリ容量	4TB	3TB	2TB	2.9TB

まず, 各計算機に搭載されている CPU のクロック周波数を見てみると, どれも 1.6GHz ～ 3.0GHz と, 市販の PC と同等かそれより遅いように見えます. 実際のところ, 1 つの CPU コアだけで比較すると, スーパーコンピュータと市販の PC の性能差はほとんどありません[1].

次にノード内の CPU コア数とメモリ容量を見てみると, PRIMERGY は PC クラスタなので 4CPU コア, 8GB と市販の PC 並みですが, それ以外の計算機は 16～64CPU コア, 128～512GB を搭載しています. ノード内のメモリはノード内の CPU コアで共有されているため, プログラミングが容易な共有メモリ型の並列処理に利用できます. すなわち, PC では実行できない大規模な計算を手軽にかつ高速に行えます.

一方各計算機の総 CPU コア数や総メモリ容量を見ると, それぞれ 400～2000 個の CPU コアと 2～4TB のメモリで構成された, 非常に大規模な計算機であることが分かります.

このように, 仕様だけ見ても, PC の限界を超えた計算が出来るということが分かって頂けると思います.

1.2　天気予報とスーパーコンピュータ

さて, このように大規模で高性能な計算機は誰が使うのでしょうか? 前述したように, スーパーコンピュータは様々な分野で利用されていますが, 特に計算機に高い演算性能を求めている分野として頻繁に取り上げられるのが, 天気予報です.

現在の天気予報には計算機によるシミュレーション結果が活用されています. このシミュレーションでは, まず予測の対象となる地域を, 東西, 南北, 及び鉛直方向にそれぞれ一定間隔の格子で区切ります. 次にその各要素毎に気温, 気圧, 風速, 風向等の初期値を設定して, 流体力学や熱力学に基づいた方程式を用いて時間発展させることにより, 将来の状態を予測します[2].

このシミュレーションにおいて, 空間を区切る刻み幅や時間発展させる間隔は, 天気予報の精度に大きく影響します. 例えば, 東西方向及び南北方向の格子を 100km 間隔から 10km 間隔に縮めると, 予報の精度を格段に向上させることができると期待できますが, 計算すべき要素数は 100 倍になります. この場合, 計算アルゴリズムにもよりますが, 計算に要する時間は少なくとも 100 倍以上となります. すなわち, 100 倍の要素数によるシミュレーションを以前と同じ時間で計算するためには 100 倍以上高速な計算機が必要になります. 逆に言えば, 計算機の演算性能が高くなるとその分予報の精度を向上させることができます.

[1]ただし, このクロック周波数を単純に比較することはできません. 例えば PRIMEQUEST に搭載されている Itanium2 や eServer p5, SR11000 に 搭載されている Power5 は, メモリアクセスを効率良く行うための機構等, 様々な高速化技術が備わっており, クロック周波数が低くてもプログラムによっては PC を凌ぐ性能を発揮します.

[2]計算機による天気予報について興味のある方は, 以下の Web ページも参照してください.
http://www.jma.go.jp/jma/kishou/know/yohou.html

1.3 スーパーコンピュータの世界ランク

世界のスーパーコンピュータ事情を知る上でとても参考になるサイトがTOP500リスト(http://www.top500.org)です. これは, 世界中で実際に導入された計算機システムについて, 演算性能の上位500位を公開しているものです. 本来, 計算機の演算性能はプログラムによって異なるので単純な比較は難しいのですが, このTOP500ではLinpackベンチマークと呼ばれるプログラムを公開し, このプログラムの実行に要した時間を指標として用いています.

なお, このリストは自己申告制なので, 計算機の性能計測等の作業も自分で行わないといけません. また, 上位に入ったからといって特に賞金や景品があるわけでもなく, 順位を示す認定証が一枚送られてくるだけです. それでもスーパーコンピュータを導入した多くの機関では, 単に演算性能を計測するだけでなく, 計算機に合わせたLinpackのチューニングまで行って, このリストの上位への登録を目指します[3].

このリストは半年毎に更新されており, 1993年の公開開始からのデータを見ることができます.

例えば2002年6月から2004年の6月までTOP500リストの1位の座にいた「地球シミュレータ」(http://www.es.jamstec.go.jp/index.html)と呼ばれる計算機は, 日本で開発された計算機です. この計算機が登場した2002年6月の時点では, 2位から7位までの計算機の演算性能を合計してもかなわないほどの, 圧倒的な性能を誇っていました. そのためTOP500リスト創始者のJack Dongarra教授(テネシー大学)にこれは「Computenik」である, と言わしめました. すなわち, 宇宙開発競争時代にソ連が米国に先んじて打ち上げた人工衛星スプートニクと同じくらい, 米国にとって脅威となる計算機だ, ということです.

その後, 危機感を持った米国が開発したIBM BlueGene/L (http://www-06.ibm.com/jp/solutions/deepcomputing/solutions/bluegene.shtml)は, 2004年11月の登場時に同リストの1位となり, その後システムの拡張を経ながら現在まで1位の座を守り続けています.

1.4 並列計算機

近年利用されているほとんどのスーパーコンピュータは, 1.1節で紹介したように搭載されているCPUコアの1基あたりの性能は一般のPCやサーバとほとんど変わりません[4]. その代わり, 数百～数万基のCPUコアを搭載することにより, システム全体として演算性能の向上を図っています. このようにCPUコアを複数搭載した計算機のことを並列計算機と呼びます.

1.4.1 並列計算機の種類

並列計算機は, 内部の構造によって大きく三つの型に分類されます. まず, 各CPUコアがそれぞれ独立したメモリ空間を持つ型の並列計算機を分散メモリ型(図2(a))と呼び, 逆に全てのCPUコアが一つのメモリ空間を共有する型の並列計算機を共有メモリ型(図2(b))と呼びます. さらに, 小～中規模の共有メモリ型並列計算機をネットワークで接続した型があり, これを混合型, もしくはハイブリッド型等と呼びます.

[3]本センターのスーパーコンピュータについても, 保守作業の時間を使ってLinpackの性能計測, チューニングを行っているところです.

[4]前述の地球シミュレータは別格で, CPUコアとしてベクトルプロセッサと呼ばれる高性能プロセッサを用いているため, プログラムによってはCPUコア1基でPCの数倍～数十倍の性能が得られます. 同様のプロセッサ技術がNEC社のSXシリーズに用いられています.

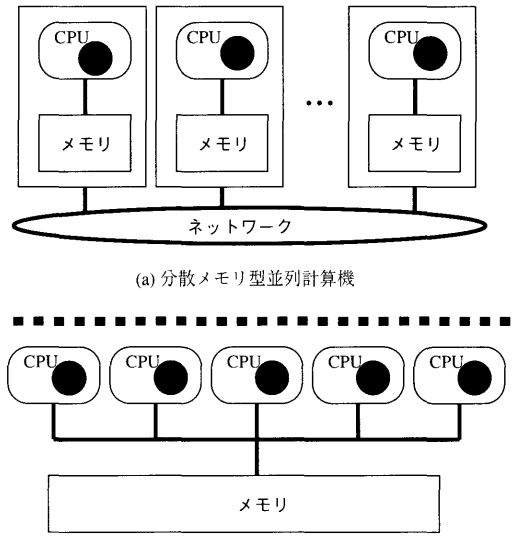

図 2: 分散メモリ型並列計算機と共有メモリ型並列計算機

ちなみに, 本センターの PRIMEQUEST580 は, CPU コアを 64 基, メモリを 128GB 搭載した計算ノード 32 台をネットワークで接続した, 混合型の並列計算機です. 現在スーパーコンピュータと呼ばれている計算機は, ほとんどがこのような混合型で構成されています.

1.4.2　並列プログラミングモデル

並列計算機の性能を活かすためにはプログラムを並列化する必要があります. プログラムの並列化とは, 並列計算機の複数の CPU コアを使ってプログラムを実行するために, プログラム中の処理を複数の同時実行可能な部分に分割するものです. また, プログラムを並列化するための記述方法を並列プログラミングモデルと呼び, このモデルにも並列計算機と同様に分散メモリ型, 共有メモリ型及びそれらを合わせた混合型があります.

まず分散メモリ型の並列プログラミングモデルでは, 分割したそれぞれの処理の流れ (プロセス) が独立したメモリ空間を所有します. そのため, 他のプロセスの計算結果を参照するにはプロセス間でデータを転送する必要があります. この分散メモリ型の並列プログラミングモデルとして最も広く用いられているのは MPI(Message Passing Interface) です.

一方共有メモリ型の並列プログラミングモデルでは, プロセスよりも小さいスレッドと呼ばれる単位で処理を分割します. このモデルでは全スレッドが一つのメモリ空間を共有するので, 比較的簡単にプログラムの並列化を行えます. 共有メモリ型の並列プログラミングモデルとしては, OpenMP が最も標準的に使われています.

また, もっと簡単に並列化を行う方法として, コンパイラによる自動並列化があります. これは, コンパイル時のプログラム解析結果をもとにコンパイラがプログラム中の並列化可能な部分を抽出して, 自動的に並列化を行うもので, プログラマはプログラムに一切変更を加える必要がありません. ただし, ほとんどの場合並列化の対象となるのはプログラム中のループのみで, しかも構造が比較的単純なものに限られます.

MPIによって並列化されたプログラムは, モデルとしては分散メモリ型ですが, 共有メモリ型並列計算機でもメモリ空間を仮想的に分割することによって実行できます. 一方 OpenMP や自動並列化によって並列化されたプログラムは, 基本的に共有メモリ型並列計算機でしか実行できません. すなわち, 分散メモリ型並列計算機で実行する予定がある場合は MPI, そうでなければ, 自動並列化, もしくは OpenMP で並列化する, という選択が一般的です.

ちなみに, 本記事で紹介している学生用無料アカウントで利用できるのは計算ノード内の 16 基の CPU コアなので, 共有メモリ型並列計算機として扱えます. すなわち, 自動並列化, OpenMP, MPI のどの並列化手法も選択可能です.

一方, 本センターの有料アカウントを利用すると, 複数の計算ノードに跨った大規模な並列計算も可能となります. この場合の並列化手法としては, 計算ノードの中も外も全て MPI のみで並列化する手法か, もしくは計算ノード内の処理を自動並列化や OpenMP で並列化し, 計算ノードに跨る処理を MPI で並列化する混合型の手法のどちらかを選択することになります. どちらの手法が適しているかはプログラムの構造や計算機の構成に依存しますが, いくつかのプログラムで試した限りでは, MPI のみによる並列化でも十分な性能が得られるようです.

1.4.3 MPI

MPI では, プロセス間通信を行うためのサブルーチン等が定義されており, C や C++, Fortran のプログラムから呼び出すことによって並列プログラムを記述できます. MPI によるプログラムの並列化では, 処理の分割だけでなくデータの各プロセスへの配置やプロセス間通信の指示が必要なので, 通常はプログラム全体にわたる設計の変更が必要となります.

```
localN = N / P
allocate(a(localN))
 ...

! 部分和の計算
local_sum = 0d0
do i = 1, localN
  local_sum = local_sum + a(i)
end do

! 全プロセスの部分和を 1つのプロセスに転送して総和を計算
call MPI_Reduce(local_sum, sum, 1, MPI_DOUBLE_PRECISION, MPI_SUM, &
                0, MPI_COMM_WORLD, ierr)
```

図 3: MPI プログラム例 (ベクトルの総和)

図 3 は, MPI による並列プログラムの例です. このプログラムでは要素数 N のベクトルの総和を並列に計算しています. そのため, 要素数をプロセス数 P で割ることにより各プロセスに割り当てるベクトルの大きさ localN を計算し, それを用いてベクトル a を配置し, 部分和を計算して変数 local_sum に格納しています. その後, 各プロセスの計算結果を取りまとめるため, MPI_Reduce という MPI の通信サブルーチンを呼び出しています. ここでは, 全プロセスの local_sum を 0 番のプロセスに転送し, そこで総和を計算して変数 sum に格納します. このよ

うに MPI では, データや処理の分割方法, データの転送等をプログラムに明記する必要があります.

MPI の利点の一つとして, 計算機の種類によらず実行できる点が挙げられます. MPI で並列化されたプログラムは分散メモリ型並列計算機で実行できるのは当然ですが, 前述の通り共有メモリ型並列計算機でも, 共有メモリを仮想的に分割し, 分散メモリ型と見なして実行することが可能です. すなわち, MPI はほとんど全ての並列計算機で利用可能な並列プログラミングモデルです.

MPI による並列プログラミングの詳細については, 以下の資料を参照してください.

1. P. パチェコ, 「MPI 並列プログラミング」, 培風館, ISBN-10: 456301544X, 2001 年 10 月
2. 青山 幸也, 「並列プログラミング入門 MPI 版」,
   http://accc.riken.jp/hpc/training/mpi/mpi_all_2007-02-07.pdf

1.4.4 OpenMP

OpenMP は, プログラム中に特殊なコメント行を追加することによって並列化に関する指示を行う並列プログラミングモデルです. この指示行はコンパイラによって解釈され, 自動的に適切な命令に変換されます. このためプログラマは, 並列化に関する細かい指示を与える必要がありません. また, OpenMP ではスレッド間でメモリが共有されているため, ほとんどの場合プログラムの構造を大きく変える必要がなく, MPI より容易に並列化を行えます.

図 4 は OpenMP による並列プログラムの例です. これも図 3 と同様にベクトルの総和を計算していますが, ほとんど並列化前のプログラムと変わりません. 唯一の違いが, `!$omp` で始まる行です. これは OpenMP の指示行と呼ばれ, これによって並列化に関する指示を行います. ここでは, 直後の `do` 文をスレッドに分割して並列実行するよう指示されています. また, `sum` という変数が総和を格納する変数であることも指示されており, これによってループの終了後に全スレッドの計算結果が取りまとめられ, 総和が計算されます.

なお, `!$omp` という行は感嘆符で始まるため, OpenMP を解釈できないコンパイラでコンパイルすると, 注釈行として無視されます. そのため, この場合は並列化前のループがそのまま実行されます. このように, 並列化前後のプログラムを一つのファイルで共存させることができるのも, OpenMP の特徴の一つです.

```
! 総和の計算
!$omp parallel do reduction(+:sum)
do i = 1, N
  sum = sum + a(i)
end do
```

図 4: OpenMP プログラム例 (ベクトルの総和)

一方 OpenMP の欠点としては, 共有メモリ型並列計算機以外での利用が困難であることが挙げられます. OpenMP で並列化されたプログラムを, 分散メモリ型並列計算機で実行させるための技術もいくつか開発されていますが, まだ計算機の性能を十分に活かせるほどではありません.

OpenMP による並列プログラミングについての詳細は, 以下を参照してください.

1. 牛島 省,「OpenMP による並列プログラミングと数値計算法」, 丸善, ASIN: 4621077171, 2006 年 5 月

2. Michael J. Quinn, "Parallel Programming in C with MPI and OpenMP," McGraw-Hill Education, ASIN: 0071232656, 2003 年 9 月

3. 「インテル コンパイラー OpenMP 入門」, http://jp.xlsoft.com/documents/intel/compiler/525J-001.pdf

4. 「インテル C/C++ コンパイラー OpenMP 活用ガイド」, http://jp.xlsoft.com/documents/intel/compiler/526J-001.pdf

5. 「インテル Fortran コンパイラー OpenMP 活用ガイド」, http://jp.xlsoft.com/documents/intel/compiler/527J-001.pdf

2 講義でスーパーコンピュータを使うための準備

次に, 実際に本センターのスーパーコンピュータを使うための手続きや, 必要なソフトウェア, 計算機の構成等の情報を紹介します.

2.1 利用申請

まず本センターのスーパーコンピュータを利用するためには, 利用申請が必要です. 利用申請書は以下のページからダウンロードできます.

http://www.cc.kyushu-u.ac.jp/scp/guidance.html

学生用無料アカウントの申請書もここで入手できます. 「各種申請書のダウンロード」から, 「講義等のための申請書類」をダウンロードして記入して下さい. なお, 「指導用登録番号」は空欄で構いません.

その後, 情報基盤研究開発センター共同利用係 (e-mail:kyodo@cc.kyushu-u.ac.jp) 宛に申請書を送ると, 約 1〜2 週間で利用承認書が届きます. 利用承認書は担当教員及び全学生に 1 枚ずつ送られてきますので, 学生に配布してください.

ただし, 承認書にはユーザ ID と初期パスワードが書いてありますので, 配布にあたって学生には承認書を大事に扱うように注意してください. また, 承認書を配布したらすぐにパスワードを変更させてください. ちなみにパスワードの変更は, 計算機にログインして passwd というコマンドを実行することによって行えます.

2.2 スーパーコンピュータへのアクセス

本センターの PRIMEQUEST 580 は, 図 5 に示す通りフロントエンドサーバとバックエンドサーバで構成されています. このうち, 利用者はまずフロントエンドサーバ (tatara.cc.kyushu-u.ac.jp) にログインして作業をすることになります.

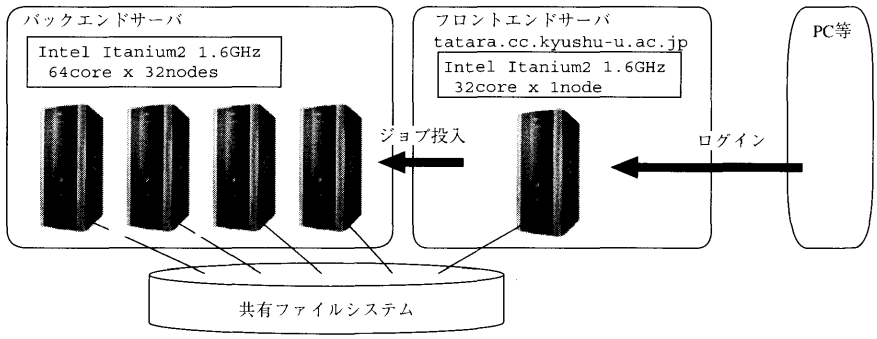

図 5: スーパーコンピュータの構成

フロントエンドサーバへのアクセスには, SSH 接続[5] に対応した端末ソフトウェアとファイル転送ソフトウェアが必要です. 対応ソフトウェアのインストール方法は, 以下を参照してください.

- 端末ソフトウェア (UTF-8 及び SSH2 に対応した TeraTerm Pro):
  `http://sourceforge.jp/projects/ttssh2/`
- ファイル転送ソフトウェア (WinSCP):
  `http://www.tab2.jp/~winscp/`

ちなみに, 本センターが管理する教育用 PC(情報基盤研究開発センター, 文系地区分室, 農学部分室, 病院地区分室, 六本松 130, 131, 135, 136 番教室及び伊都キャンパス情報講義室) には, 既に TeraTerm Pro と WinSCP がインストールされています.

また, PRIMEQUEST 580 では日本語の文字コードとして UTF-8 を採用しています. 上記の TeraTerm Pro 等, UTF-8 をサポートしている端末ソフトウェアでは, 文字コードを適切に選択してください.

一方, UTF-8 に対応していない端末ソフトウェアでは, 日本語が正しく表示できませんので, 最初のログイン時に以下を実行し, メッセージを英語表記に変更してください.

```
tatara% echo unsetenv LANG >> ~/.cshrc
tatara% source ~/.cshrc
```

2.3 フロントエンドサーバとバックエンドサーバ

前節でも紹介した通り, 本センターのスーパーコンピュータはフロントエンドサーバ部とバックエンドサーバ部に分かれています. このうちフロントエンドサーバは多数の利用者が同時にログインして直接操作するため, 計算機が処理能力オーバーで停止したりすることがないよう, 多少厳しく使用可能メモリ量や計算時間を制限しています. 具体的には, プログラム 1 つ当たりの使用メモリ量を 1GB 以内, 使用 CPU 時間を 1 時間以内としています. そのためフロント

[5]ネットワーク上で通信内容を盗み見られることがないよう, 暗号化して通信を行う接続です.

エンドサーバでは, 主にプログラムの作成やコンパイル, 規模の小さなプログラムの実行等を行い, 本格的なプログラムの実行にはバックエンドサーバを用います.

バックエンドサーバは, フロントエンドサーバからバッチ処理と呼ばれる方法で間接的に利用します. このバッチ処理についての詳細は後述しますが, 直接的な利用を禁止することによってバックエンドサーバでは, フロントエンドサーバよりも大きなプログラムを長時間実行できます. 例えば学生用無料アカウントの場合, メモリ使用量 22.4GB 以内のプログラムを, 最長 1 週間まで継続して実行させることができます.

2.4 スーパーコンピュータの基本操作

2.4.1 スーパーコンピュータの OS

先にも書いた通り, 最近のスーパーコンピュータでは OS として UNIX や Linux が使用されており, 本センターの PRIMEQUEST でも Linux が稼働しています. そのため, UNIX の基本的な利用法を知っていれば特に違和感無く操作することができます.

ちなみに UNIX の基本的な利用法については, 以下のページ等を参照してください.

- UNIX 利用法:
  `http://www.cc.kyushu-u.ac.jp/ec/manual/Tebiki/UNIX.pdf`
- エディタ Mule(Emacs) 利用法:
  `http://www.cc.kyushu-u.ac.jp/ec/manual/Tebiki/Mule.pdf`

2.4.2 バッチ処理

スーパーコンピュータに限らず多数の利用者が共同で利用する計算機では, それぞれの利用者からの処理を全て直接実行していると, 処理能力を超えてシステムダウン等を起こす危険性が高くなります. そこでこのような計算機では, 処理要求を交通整理する仕組みとして, バッチ処理と呼ばれる方法が採用されています.

バッチ処理とは, 利用者がコマンドを直接実行する代わりに, 実行して欲しいコマンド等の処理内容を専用の書式で記述し, ジョブとしてシステムに投入するものです. 利用者から投入されたジョブはバッチジョブシステムが受理し, 計算機の空き状況によって自動的に実行を開始します.

なお, 2.3 節で書いた通り, 本センターの PRIMEQUEST 580 ではフロントエンドサーバで対話的な利用をある程度許していますが, バックエンドサーバはバッチジョブシステムを経由した間接利用しか行えません.

3 スーパーコンピュータを使った演習例

この無料アカウントを用いて, 実際に筆者が担当する講義でスーパーコンピュータによる演習を行ってみました. 本節では, この演習の内容を紹介します.

3.1　演習の方針

今回はプログラミングの経験がほとんど無い学生を対象としたFortran入門の講義の中での演習でした．また，スーパーコンピュータが稼働を開始した時点で残りの講義数が少なくなっており，プログラムの並列化について十分な時間を割くことができませんでした．そこで今回の講義では，予め用意したプログラムの処理速度を計測する演習を通して，スーパーコンピュータの性能を実際に確認させることと，バッチ処理による計算機利用法の習得を図りました．そのため，並列化手法としては自動並列化のみを扱っています．既にC言語もしくはFortranを習得している学生が対象であれば，数回の講義を通じてMPIやOpenMPによる簡単な並列プログラミングを扱うことも可能だと思います．

3.2　演習に用いたプログラム

図6に，今回の講義で用いたプログラムを示します．このプログラムでは密行列の行列積を計算しています．プログラム自体は非常に簡単なもので，特に変わったことはありませんが，サブルーチン `mm` の計算に要した時間を計測するための時間計測ルーチンとして，経過時間を計測する `system_clock` を用いています．

経過時間とCPU時間

プログラムの性能を評価する際，実行時間として経過時間を計測する場合と，CPU時間を計測する場合があります．経過時間とは，ストップウォッチで計測するように，プログラムの実行に要した時間を計測したものです．一方CPU時間とは，そのプログラムを実行するためにCPUが動作した時間の合計値です．

こう書くと，経過時間とCPU時間は同じもののように見えるかも知れません．しかし経過時間には，ファイルの入出力や他CPUとの同期待ち時間等，CPUが動作しない処理も含まれるため，一般に経過時間とCPU時間は一致しません．また，自動並列化やOpenMPによる並列プログラムのように共有メモリ型並列計算機で並列実行されるプログラムでは，全CPUで使用された合計の時間がCPU時間として報告されます．

並列プログラムの性能評価においては，CPUの動作時間だけでなくスレッド間の同期待ち時間等，CPU時間に含まれない時間が重要であるため，並列プログラムの性能を評価する際の実行時間としては，一般に経過時間を使用します．

```
program mmtest
implicit none
integer, parameter :: n=4000
real(8),dimension(n,n) :: a, b, c
real(8) :: time1, time2, total
integer :: icount, irate
intrinsic dble

  call random_number(a)  ! Initialize a and b with
  call random_number(b)  !  a random number generator
  call system_clock(icount,irate)  ! Get current time info.
  time1 = dble(icount)/dble(irate) ! Calculate time in seconds
  call mm(c, a, b, n)              ! C = A * B
  call system_clock(icount,irate)
  time2 = dble(icount)/dble(irate)
  total = time2 - time1          ! Elapsed time for mm

  write(*,'(A6,F8.4,2x,e)') "Size", n, " Time=", total, c(n,n)
stop
end program

subroutine mm(c, a, b, n)
implicit none
integer :: n
real(8), dimension(n, n) :: a, b, c
integer :: i, j, k

  c = 0d0                ! Initialize C as zero

  ! Matrix multiplication
  do i = 1, n
    do j = 1, n
      do k = 1, n
        c(i, j) = c(i, j) + a(i, k) * b(k, j)
      end do
    end do
  end do

end subroutine
```

図 6: プログラム例: 行列積

3.3 プログラムのコンパイル

本センターの PRIMEQUEST 580 で利用できる主なコンパイルコマンドを表 2 に示します.

表 2: PRIMEQUEST で利用可能なコンパイルコマンド

言語	コマンド	ソースファイルの拡張子
FORTRAN77	frt	.f
Fortran90/95	frt または f90 または f95	.f90 または .f95
C	fcc	.c
C++	FCC	.cc または .C

また, コンパイルは通常の UNIX と同様に以下の形式で行います.

```
コンパイルコマンド　コンパイルオプション　ソースファイル名
```

主なコンパイルオプションについては, 以下のページを参照してください.

http://www.cc.kyushu-u.ac.jp/scp/system/general/PQ/how_to_use/index.html

ちなみに, 今回筆者が行った講義では, 以下の二通りのコンパイルを行いました.

```
tatara% f90 -Kfast mm.f90 -o mm
tatara% f90 -Kfast,parallel mm.f90 -o mm-p
```

-K は, 主にプログラムの高速化のためのオプションを指定するもので, ここでは以下の 2 つを用いています. また, 上記のように 2 つ以上のオプションを列挙することもできます.

オプション	意味
fast	最適化
parallel	自動並列化

ここで「最適化」とは, 計算機の特性に合わせてプログラムを変形し, 高速化を図る機能です. 例えば図 6 のプログラムで, サブルーチン mm 内の 3 つのループ i,j,k, は, 互いに入れ替えても結果はほとんど変わりません. また, Fortran では多次元配列は c(1, 1), c(2, 1), c(3, 1), ... のように, 第 1 次元方向, すなわち最も左側の添字について連続になるようメモリに配置されているので, 第 1 次元に対して連続にアクセスを行った方が効率が良くなります. そこでコンパイラは, 最適化機能が有効になると, ループ i が最も内側になるようプログラムを変形します. コンパイラには他にも様々な変形技術が用意されており, それらを適用して高速化を図ります.

一方「自動並列化」は, 1.4.2 節で説明した通り, コンパイル時の解析によってプログラム中の並列化可能な部分を抽出し, 自動的に並列化する機能です. ここで並列化可能な部分とは, 並列化によってプログラムの意味が変わらない部分のことです.

このように最適化機能や自動並列化機能は, 基本的にプログラムの結果をなるべく変えない範囲で適用されます. ただし, 計算機で実数の計算を行う際の丸め誤差が計算の順序に依存するため, 最適化や自動並列化によって計算結果が若干変わる可能性があります.

並列化の可否

並列化とは, 複数のCPUに処理を割り当てることができるよう, プログラムの処理を複数に分割することであり, 分割されたそれぞれの処理が実際にどのような順番で行われるか, 予測できません. そのため, どのような順番で実行されてもプログラムの意味が変わらないと分かる場合のみ, 並列化可能と判断できます.

例えば図6のプログラムで, ループ i は i=500 での計算が i=1 での計算よりも早く計算されても, 計算結果は変わりません. そのため, ループ i を 1〜400, 401〜800 のように400回ずつに分割してそれぞれ別のCPUに計算させるような並列化が可能です.

一方以下のループでは, i=100 の計算が実行される前に i=101 の計算が行われるとプログラムの意味が変わりますので, 並列化できません.

```
do i = 1, 1000
  c[i] = a[i] + c[i-1]
end do
```

3.4 プログラムの実行

3.4.1 ジョブの投入

前述の通り, フロントエンドでも簡単なプログラムは実行できますが, 大きなプログラムを実行するにはバッチジョブシステムにジョブを投入する必要があります. バッチジョブシステムの利用法は以下のページを参照してください.

http://www.cc.kyushu-u.ac.jp/scp/system/general/PQ/how_to_use/06_NQS.html

このページにもある通り, まず実行要求を記述したジョブリクエストファイルを作成します. この実行要求の内容としては, 実行して欲しいコマンドをそのまま列挙します. 例えば今回の講義では, 以下のファイルを作成しました.

```
#!/bin/csh
cd $QSUB_WORKDIR
./mm
```

次に, このファイルを test.sh という名前で保存し, 以下の通りバッチジョブシステムに投入しました.

```
tatara% qsub -q pdbg1 test.sh
```

ジョブクラス 上の例では, qsub コマンドへのオプション -q pdbg1 により, ジョブを投入するジョブクラスとして pdbg1 を指定しています. ジョブクラスとは, バッチジョブシステムで管理される計算機資源の利用枠のようなものです. 通常, バッチジョブシステムではいくつかの大きさのジョブクラスを用意し, 利用者はその中から自分のジョブにあったジョブクラスを選択します. ちなみに, ジョブクラスの大きさを複数用意するのは, 小さいジョブクラスだけだと大きいジョブが実行できず, 大きいジョブクラスだけだと小さいジョブを実行する場合に無駄

に計算機資源を確保してしまうためです．学生用無料アカウントで利用可能なジョブクラスは以下の通りです．

ジョブクラス名	利用可能な資源
pdbg1	1CPU コア，メモリ量 1.4GB，1 時間
pdbg8	8CPU コア，メモリ量 11.2GB，1 時間
p16	16CPU コア，メモリ量 22.4GB，1 週間

3.4.2　ジョブの確認と削除

バッチジョブシステムに投入した自分のジョブは，qps コマンドで確認できます．一方，計算機の空き状況を知りたい場合は，qps -a を実行して，現在投入されている全てのジョブの情報を表示します．qps コマンドで表示される主な情報としては，ID (ジョブの番号)，Que (ジョブクラス)，Owner (投入した人の ID)，State (ジョブの状態．RUNNING は実行中．QUEUED は待ち状態)，Create Time (投入日時)，Cpu Time (現在までの使用 CPU 時間) 等があります．

投入したジョブを削除するには，以下のように qdel コマンドを用います．

```
tatara% qdel -k ジョブ番号
```

ここで -k は，ジョブが既に実行を開始していても，強制的に終了させることを指示します．

3.4.3　並列プログラムの実行

次に，自動並列化したプログラムを実行してみます．これは，3.3 節でコンパイルしたプログラムのうち mm-p を用います．

自動並列化や OpenMP によって並列化されたプログラムは，1.4.2 節で紹介した通り複数のスレッドによって並列実行されるので，実行時に使用するスレッド数を指定します．このスレッド数の指定には，setenv OMP_NUM_THREADS という環境変数を用います．以下は，スレッド数を 4 に指定して並列実行する場合のジョブリクエストファイルの例です．なお，スレッド数を指定しない場合，利用可能な最大の CPU コア数と同じ数が指定されたものとして実行されます．

```
#!/bin/csh
cd $QSUB_WORKDIR
setenv OMP_NUM_THREADS 4
./mm-p
```

このジョブを投入する場合，qsub コマンドの -lp オプションを用いて，少なくともそのジョブで使用するスレッド数と同じ数の CPU コアを確保します．例えば，先ほどのように 4 スレッドを利用して並列実行するジョブは，-lp オプションに渡す CPU コア数として 4 以上を指定します．

```
tatara% qsub -q pdbg8 -lp 4 test.sh
```

CPU コア数及びメモリ量の指定　このように, 使用する CPU コア数の指定は qsub コマンドのオプション-lp で行います. 一方, CPU コア 1 個あたりのメモリ量の指定は -lm オプションで行えます. 例えばジョブ投入時に以下のように指定することにより, 最大 5.6GB(4 × 1.4GB) のメモリを使用できます.

```
tatara% qsub -q pdbg8 -lp 4 -lm 1.4gb test.sh
```

なお, こうして確保されたメモリは共有メモリなので, 実際に実行する際のスレッド数とは関係なく利用できます. また, -lm オプションで指定可能な最大のメモリ量は 1.4GB なので, 例えば並列化されていないプログラムであっても 1.4GB を超えるメモリを使用するプログラムでは -lp で指定する CPU コア数を増やして必要なメモリを確保する必要があります.

3.4.4 並列化による効果

プログラムの並列化を行うと, スレッド数に比例して性能が向上すると期待してしまいますが, 実際には必ずしもそうなりません. 例えば各スレッドに割り当てる処理が均等でなかったり, 並列化に伴ってスレッドを生成したり削除したりするコストが必要になったりして, 理想的な性能向上が得られないことが良くあります. ここでは, 並列化の効果を評価する方法を紹介します.

並列プログラムの性能を評価する指標として最も良く用いられるのは, 並列化前後の経過時間の比で計算される性能向上率です. すなわち, 並列化前のプログラムを実行した際の経過時間 T_{seq} と並列化後のプログラムを実行した際の経過時間 T_{par} から, 性能向上率 $R_{speedup}$ は, 以下の式で計算できます.

$$R_{speedup} = T_{seq}/T_{par}$$

この性能向上率は, 並列実行時のスレッド数に応じて変化します. そこで, 使用するスレッド数 $N_{threads}$ を変化させた場合の性能向上率の変化を調べることにより, そのプログラムの並列処理に関する特性を見ることができます. また, この特性を並列化の効率 E_{par} として以下の式で数値化して評価する場合もあります.

$$E_{par} = R_{speedup}/N_{threads}$$

一方, 性能向上率の計算において, 「並列化前のプログラムを実行した場合の経過時間」ではなく「並列化後のプログラムを 1 スレッドで実行した場合の経過時間」を用いることもあります. この評価方法は, プログラムの並列化に関する傾向を確認するには良いのですが, 並列化前のプログラムとの比較を行わないため, 並列化によって本当に速くなったかどうかを検証できません.

今回の講義では, 最終的に以下のようにジョブリクエストファイルを書き換え, スレッド数を変化させた場合の経過時間の変化を計測し, 評価に用いました.

```
#!/bin/csh
cd $QSUB_WORKDIR
setenv OMP_NUM_THREADS 1
echo $OMP_NUM_THREADS
./mm-p
setenv OMP_NUM_THREADS 2
echo $OMP_NUM_THREADS
./mm-p
setenv OMP_NUM_THREADS 4
echo $OMP_NUM_THREADS
./mm-p
setenv OMP_NUM_THREADS 8
echo $OMP_NUM_THREADS
./mm-p
setenv OMP_NUM_THREADS 16
echo $OMP_NUM_THREADS
./mm-p
```

3.5　教育用計算機での実験

スーパーコンピュータでの実験だけでは相対的な速さが分からないので, 図 6 のプログラムを, 教育用計算機システムのアプリケーションサーバでも以下の通りコンパイル, 実行してみました. ただし, このプログラムをそのまま実行すると最大で 10 分程度かかってしまうので, 講義ではパラメータの n の値を 1000 に変えてコンパイル, 実行しました.

```
% f90 -Kfast mm.f90 -o mm
% ./mm
% f90 -Kfast,parallel mm.f90 -o mm-p
% setenv OMP_NUM_THREADS 1
% ./mm-p
% setenv OMP_NUM_THREADS 2
% ./mm-p
% setenv OMP_NUM_THREADS 4
% ./mm-p
```

プログラムから明らかなように, このプログラムの実行には n の 3 乗に比例した時間がかかります. これは, n の値を 100, 500, 1000 のように変化させてコンパイル, 実行することによって確認できます. その結果をもとに n が 4000 の場合のアプリケーションサーバでの処理時間を予測して, スーパーコンピュータでの処理時間と比較することにより, 相対的な速さを確認できます.

4 おわりに

本記事では, スーパーコンピュータを講義で用いることを想定して基本情報と演習の事例を紹介しました. 実際の利用を考えると, プログラムの並列化やチューニングの方法, 高速な数値計算ライブラリの利用等, まだまだ講義で扱うべき事がたくさんあります. 本センターの Web ページ (`http://www.cc.kyushu-u.ac.jp/scp/`) 等も参照しながら, 適宜取り入れていただければと思います. また, メールでの質問も随時受け付けておりますので, 南里 (`nanri@cc.kyushu-u.ac.jp`) もしくは質問窓口 (`request@cc.kyushu-u.ac.jp`) までお気軽にお問い合わせ下さい.