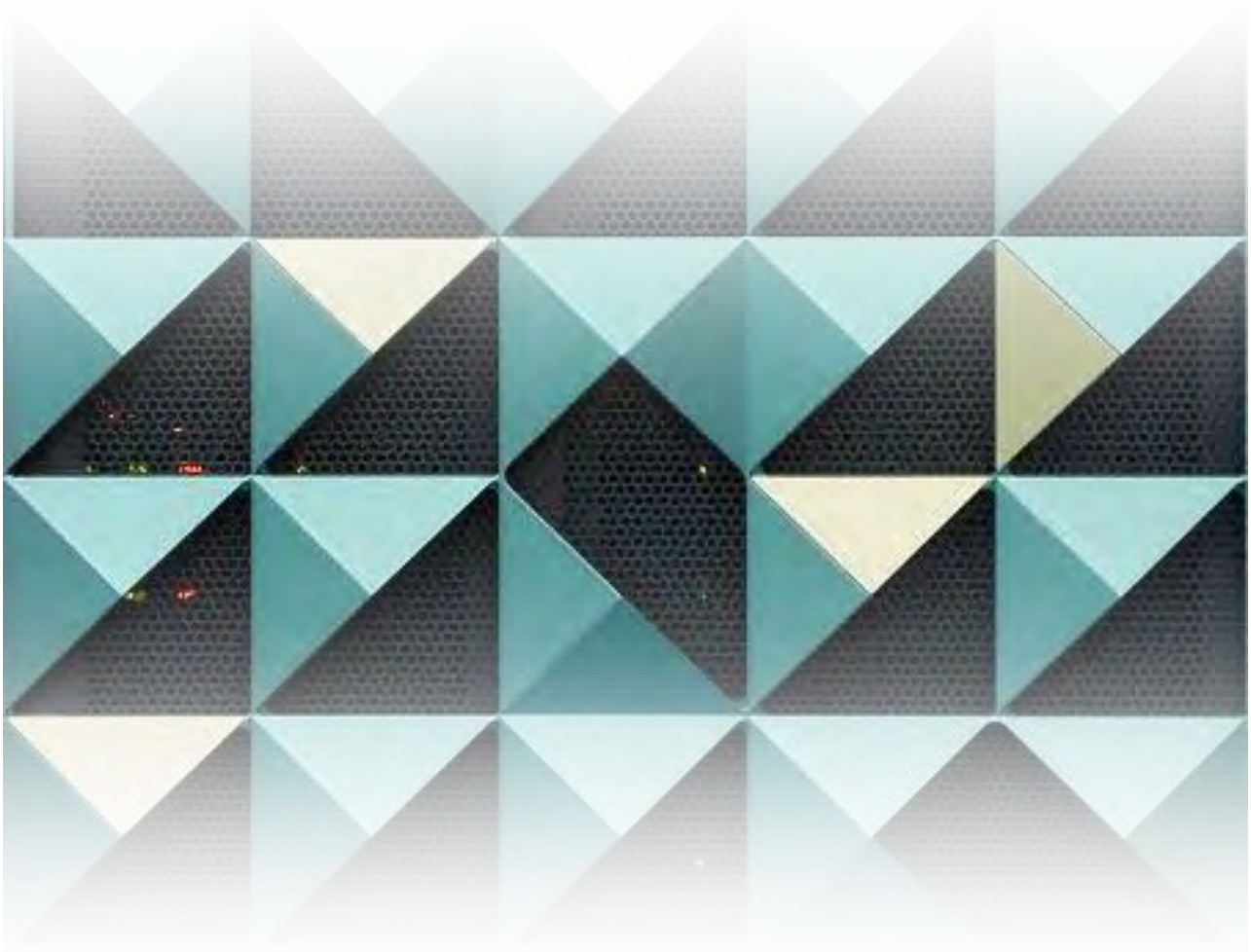
SR16000

– HITACHI SR16000 モデル M1 –

利用の手引き

第 1 版



東京大学情報基盤センター
Information Technology Center, The University of Tokyo

・ 本利用の手引は 2017 年 4 月時点での情報をもとにしています。
・ システムの現状と本手引の内容が矛盾する場合はシステムの現状を優先します。
・ 利用状況や利用者からの意見に応じてサービス内容は変更する可能性があります。
・ 本利用の手引は、予告なく改定を行います。最新版は本センターの Web ページから入手できます。
・ 本手引に記載されている会社名および製品名などはそれぞれ各社の商標、登録商標、商品名です。

はじめに

本利用の手引きは、2011 年 10 月より運用を開始した「SR16000 システム (SMP) (Yayoi) (大規模 SMP 並列スーパーコンピュータシステム)」の利用方法を説明したものです。

SR16000 システムは、2005 年からサービスを行ってきた SR11000 システムの後継システムとして導入され、既存プログラムとの高い互換性、大容量の (ノード) メモリ空間が確保できるなどの特徴を持ったシステムですが、従来システムとは異なる機能、サービスも提供しています。

本利用の手引きが SR16000 システムを使用するにあたっての参考となり、有益な計算結果、研究の手助けとなれば幸いです。

2012 年 11 月
東京大学 情報システム部
情報基盤課 スーパーコンピューティングチーム

予備知識

本利用の手引きは、本センターの SR16000 固有の機能や設定を中心に解説します。内容は利用者の方々が UNIX 系オペレーティングシステムの使用経験をお持ちで、並列計算の基本的な概念を理解されていることを前提としています。UNIX オペレーティングシステムの使用経験がない方は市販の UNIX 参考書もご用意いただくと本システムを使用するに役立ちます。並列計算につきましては多くの参考書が入手可能である他、本センターでも講習会を行っておりますのでそれらをご利用ください。学生の方であれば各大学で並列プログラミングの授業が開講されていることと思いますので、それらを受講することをお勧めします。

お問い合わせ

システムや本手引についてのお問い合わせは、次のメールアドレスまでお願いします。

受付・登録関係	uketsuke@cc.u-tokyo.ac.jp
技術的内容	soudan@cc.u-tokyo.ac.jp
要望など	voice@cc.u-tokyo.ac.jp

目次

第6章　コンパイラ

第7章　プログラムの実行

第8章　数値計算ライブラリ

第9章　使用例

第10章　マニュアルの閲覧

付録 A　Gaussian16 の使い方

1

システム概要

1.1 SR16000 システムについて

スーパーコンピューティング部門では、以下のスーパーコンピューターシステムにより、高度かつ大規模な計算サービスを提供しています。

▶ FX10 スーパーコンピュータシステム（Oakleaf-FX）
（大規模超並列スーパーコンピューターシステム）

▶FX10 スーパーコンピュータシステム（Oakbridge-FX）
（長時間ジョブ実行用並列スーパーコンピューターシステム）

▶ SR16000 システム（SMP）（Yayoi）
（大規模 SMP 並列スーパーコンピューターシステム）

この利用の手引きは「SR16000 システム」について解説します。SR16000 は 1 ノードあたり 32 個コアを備えたノードを複数台搭載しており、ノードを並列に動作されるのと同時にノード内でも並列処理を行うことが可能なシステムです。

1.2 ハードウェア構成

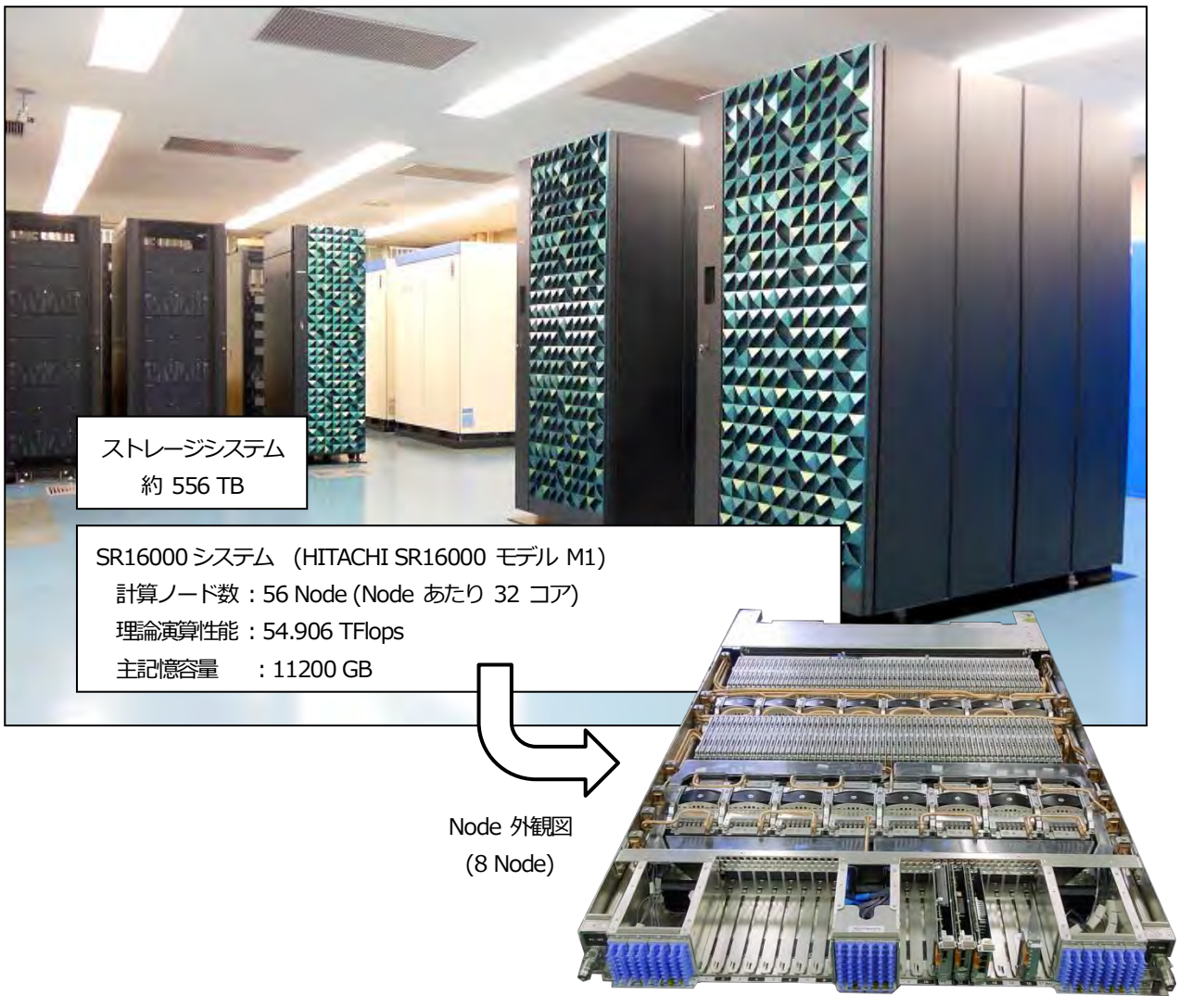
本章ではSR16000のハードウェア構成を紹介します。

1.2.1 全体構成

SR16000は1ノードあたり32個のコアを備えた計算機（計算ノード）を56台、高速なネットワークで接続したシステムです。全体構成は以下の表のようになっています。

表1-2-1　SR16000全体性能

	項目	仕様
全体性能 (計算ノード)	理論演算性能	54.906 TFlops
	主記憶容量	11200 GB
	ノード数	56
	ノード間ネットワーク	階層型完全結合
	ノード間転送性能	96GB/s (単方向) × 双方向
ストレージ容量		556 TB



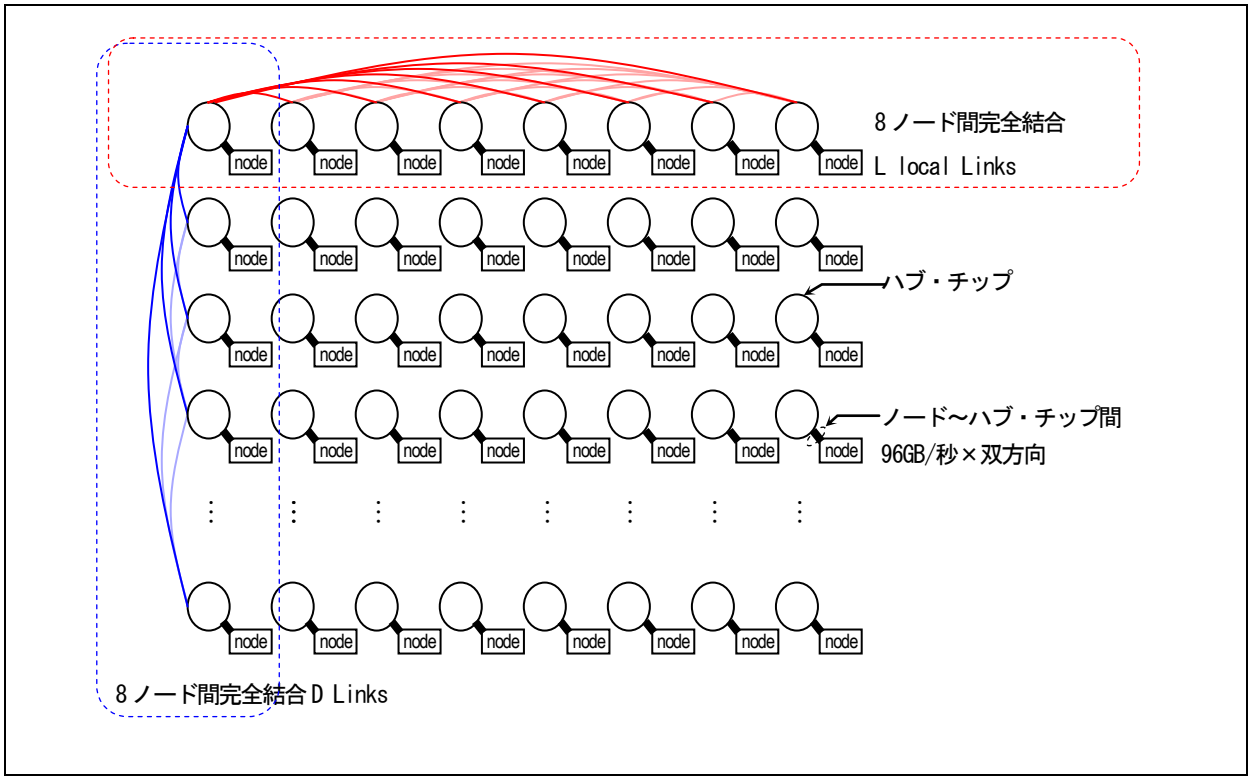


図 1-2-3　ネットワーク構成

1.2.2 ノードの構成

SR16000 のノードは、4 個の POWER7 プロセッサ(32 個コア)、200GB のメモリを搭載しています。POWER7 プロセッサの特徴として、1 コアあたり 30.64GFlops、ノードあたりでは 980.48GFlops の高い演算性能を有しています。
また、1 台のコア内で複数のスレッドを実行できる SMT(Simultaneous Multi-Threading)機能を有しており、1 台のコアを仮想的に 2 または 4 台のコアとして扱うことができます。なお、本センターでは SMT 設定を 2（1 台のコアを仮想的に 2 台のコアとして作動）としています。

表 1-2-4　SR16000 全体性能

ノード	理論演算性能	980.48 GFlops
	プロセッサ数 (コア数)	4 (32)
	主記憶容量	200 GB
プロセッサ	プロセッサ (周波数)	Power7 (3.83 GHz)
	理論演算性能	30.64 GFlops

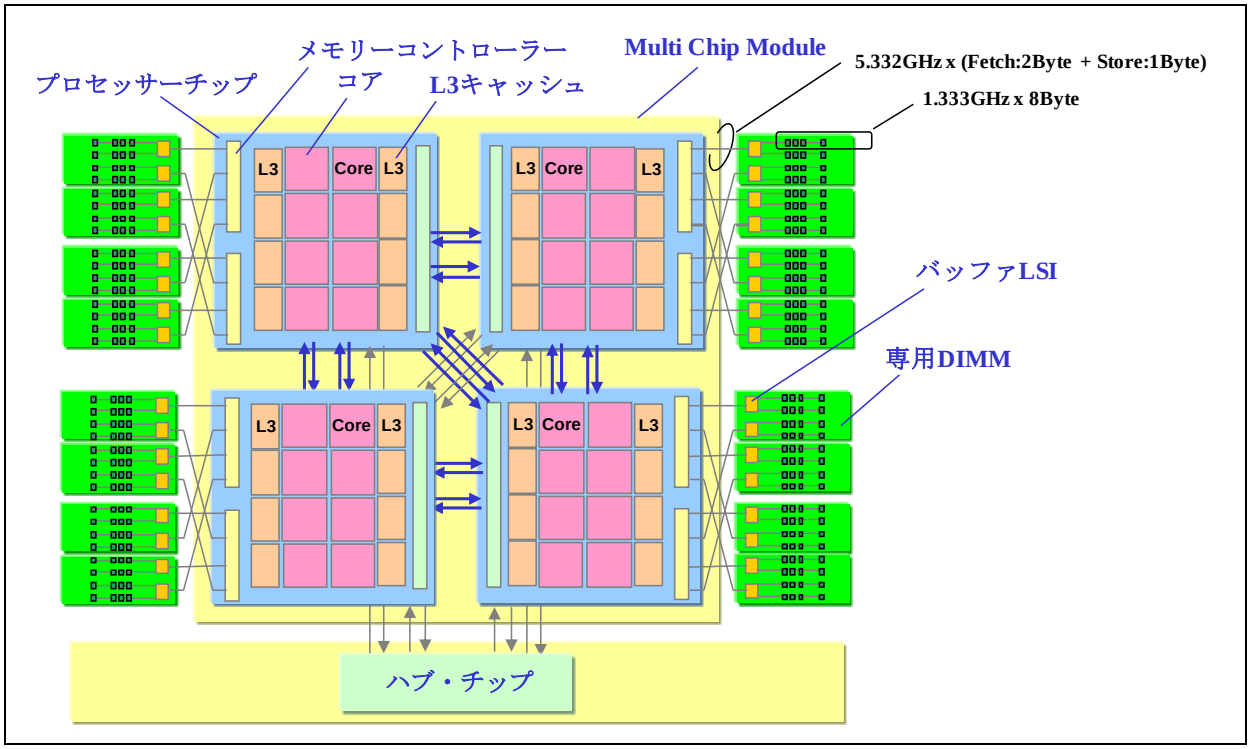


図 1-2-5　ノード構成図

1.3 ソフトウェア構成

SR16000 にインストールされている主なソフトウェアは以下の表のとおりです。

表 1-3-1　ソフトウェア一覧

項目	ソフトウェア名
OS	AIX 7.1
バッチシステム	NQS 互換機能
コンパイラ	日立製作所製　最適化 FORTRAN90，　最適化 C，　最適化標準 C++ IBM XL C/C++ Enterprise Edition for AIX Java，　GNU コンパイラ
並列化支援	MPI，　OpenMP
数値計算ライブラリなど	MSL2，　MATRIX/MPP，　MATRIX/MPP/SSS，　BLAS，　LAPACK，　ScaLAPACK，　FFTW，　SuperLU
分子計算アプリケーション	Gaussian16
フリーソフトウェア	bash，　tcsh，　zsh，　emacs，　autoconf，　automake，　bzip2，　cvs，　gawk，　gmake，　gzip，　make，　less，　sed，　tar，　vim など

これらのソフトウェアはログインノードまたは計算ノードで使用できます。ログインノードでは SSH によるログインおよびコマンドの対話的実行が可能です。主にプログラムの作成・編集、コンパイル、バッチジョブの投入に使用します。ログインノードの資源は多くのユーザーで共有しますので重い処理は行わないようにしてください。計算ノードはバッチジョブシステム(NQS)を通じて使用します。ジョブを投入してから実行されるまでに待ち時間がありますが、自分の順番が回ってきた際には計算ノードの資源を占有できます。通常、すべてのユーザープログラムはバッチジョブとして計算ノードで実行します。

SR16000 にインストールしていないソフトウェアは、ユーザー個人のホームディレクトリにインストールしていただくことで利用可能です（ただし、管理者権限が必要なソフトウェアは使用できません）。

1.4 サービス概要

SR16000 では研究者が個人単位で利用するための「パーソナルコース」によるサービスのみ行っています。パーソナルコースには、最大で利用できるノード数により複数のコースを用意しています。また、上位コースは下位コースを含みますので、例えばパーソナルコース 3 の利用者が 4 ノードのジョブを実行することも可能となっています。

表 1-4-1　パーソナルコースの種類

コース名	利用可能ノード数	利用可能ディスク容量	資源割当方式
パーソナルコース 1	1	500 GB	パーソナルコースの他の利用者と共有し、Fair Share 方式によってジョブの実行順序を決定
パーソナルコース 2	4		
パーソナルコース 3	8		

2

ログイン方法

2.1 概要

SR16000 には SSH2 を用いてログインします。ログイン自体にはパスワードを使用せず鍵による認証を行います。初回ログイン時には鍵の生成とシステムへの登録作業が必要となります。登録操作は次のようになります。

1. Windows の場合はターミナルソフトを入手する。
2. 認証に用いる鍵を生成する。
3. Web ブラウザを用いて SR16000 に公開鍵を登録する。
4. SSH ログインする。

手順は手元のマシンが Windows か UNIX 系オペレーティングシステム（Mac OS X を含む）で異なりますので、OS ごとに分けて説明します。すでに公開鍵をお持ちの方は改めて鍵を生成する必要はありません。SR16000 への鍵の登録のみで使用できます。

2.2 接続情報

ログインに必要な接続情報は以下の表のようになります。

表 2-2-1　接続情報

ホスト名	yayoi.cc.u-tokyo.ac.jp ※以下のホストの何れかに接続します。 また、どのホストに接続しても同じ環境です。負荷分散にご協力ください。 yayoi-1.cc.u-tokyo.ac.jp yayoi-2.cc.u-tokyo.ac.jp
接続方法	SSH Protocol Version 2
認証方法	鍵による認証（センター発行のパスワードは SSH ログインには使いません。） 初回は Web による鍵登録が必要です。本章で説明します。

2.3 Windows システムからのログイン方法

Windows で使用できるターミナルソフトには PuTTY や Tera Term Pro などがあります。PuTTY がもっとも鍵の扱いが容易なので、PuTTY を本センターの推奨ターミナルソフトとし接続方法を説明します。他のターミナルソフトについては解説しませんが、多くのソフトは鍵での認証に対応しているので SR16000 へのログインに使用することができます。また Cygwin を使用される方は UNIX 向けの解説をご覧ください。

2.3.1 PuTTY の入手

2008 年 5 月現在 PuTTY は日本語化されたものが次のページより入手できます。

http://hp.vector.co.jp/authors/VA024651/

このページより日本語化 PuTTY をダウンロードし、適切なフォルダに展開してください。

2.3.2 鍵の生成

PuTTY の配布ファイルを展開すると puttygen.exe という実行可能ファイルがあるのでそれを実行します。「PuTTY Key Generator」 の画面が開くので <Generate>ボタンをクリックします。すると画面の上の方にバーが出現するので、バーが右端に到達するまでバーの下あたりでマウスを適当に動かします。これにより公開鍵(public key)、秘密鍵(private key)のペアが生成されます。

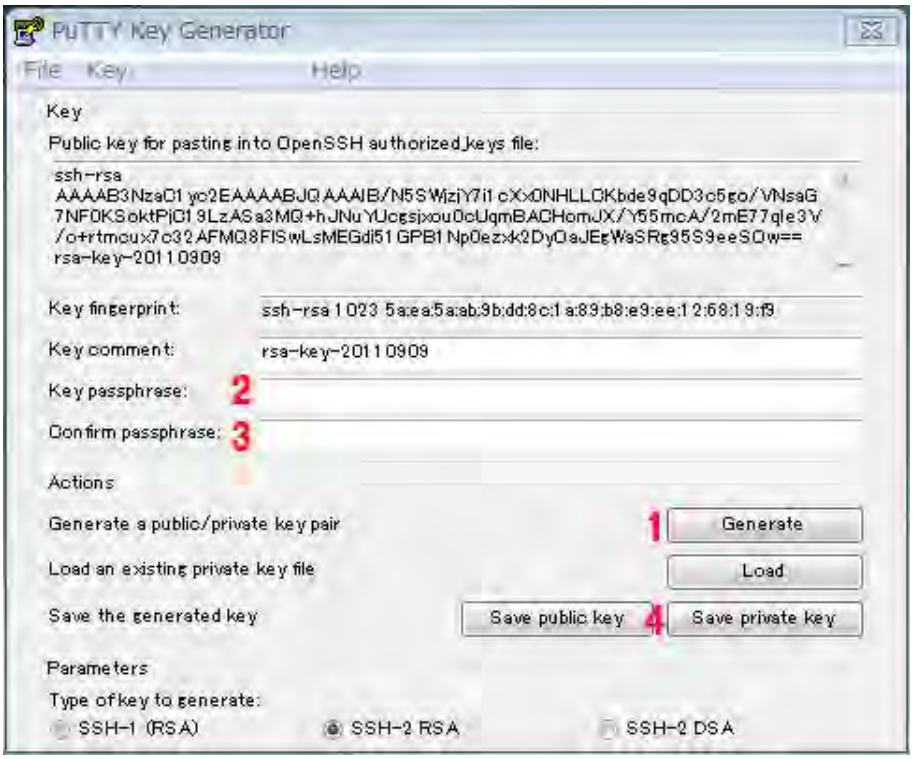


図 2-3-1　鍵の生成

[Key passphrase] と [Confirm passphrase] に適当な他人に知られない文字列を入力します。センターから通知されたパスワードとは無関係なので、センターから発行されたパスワードを入力しないように注意してください。また、入力しなくても鍵としては使えますが、セキュリティー強化のため SR16000 に登録する鍵はパスフレーズで保護されていることを必須とします。パスフレーズはメモしたりファイルに書いたりしなくて済むよう、他人には推測されず、自分には覚えやすいものを設定してください。パスフレーズの入力が終了したら <Save private key> ボタンをクリックして鍵を保存します。この秘密鍵ファイルは絶対に他人に読まれることがないよう、アクセス権には十分注意してください。この鍵を他人に読まれると SR16000 に不正侵入される危険性があります。<Save public key>ボタンは今回使用する必要はありません（<Save private key> ボタンで public key も同時に保存されています）。以上で鍵の生成は完了ですが、次の登録作業のためにこの画面は開いたままにしておいてください。

2.3.3 鍵の SR16000 への登録

SR16000 には puttygen の画面の上の方にある"Public key for pasting into OpenSSH authorized_keys file" と書かれた欄にある文字列を登録します。もし、前のステップの後 puttygen を終了してしまった場合は、再び puttygen を起動した後 <Load>ボタンを押して前のステップで保存した鍵のファイルを読み込めば登録に必要な文字列が得られます。鍵の登録のためには、次の URL にアクセスしてください。

https://yayoi.cc.u-tokyo.ac.jp/

ユーザー名とパスワードを求められるので、センターから通知されたユーザー名とパスワードを入力してください。ここで入力するパスワードは鍵の保護のために入力したパスフレーズではなく、センター発行のパスワードです。認証に成功するとユーザーページが開きますので「公開鍵登録」のリンクをクリックしてください。開いたページの "Public Key"の欄に PuTTY Key Generator ウインドウの "Public key for pasting into OpenSSH authorized_keys file" に表示されている文字列をコピー&ペーストし、"Password" の欄にセンター発行のパスワードを入力してください（すでに認証済みですが、セキュリティー上非常に重要な部分なので再度パスワード入力をお願いしております）。<登録>をクリックして完了画面が出れば登録完了です。登録まで少し時間がかかることがありますが、<登録>ボタンを何度もクリックしないようお願いします。登録作業を 2 回以上行うと、最初に登録した鍵は削除されます。また、一度でも SSH でログインすると Web 経由での公開鍵登録はできなくなります。複数のコンピュータからログインするために公開鍵を 2 個以上登録する方法や、ログインした後の鍵の変更方法は後の「鍵について」の節で説明します。

2.3.4 ログイン

PuTTY を起動すると以下のような画面が出ます。図に示した順番で以下のように情報を設定します。

1. ホスト名を入力する。
2. 「SSH」の「認証」画面に移動して秘密鍵ファイルを設定する。
3. 「セッション」画面に戻ってセッション名を入力し、保存する。
4. <開く>ボタンで接続する。ユーザー名を入力すると鍵のパスフレーズを聞かれるので、鍵作成の時に設定したパスフレーズを入力する。

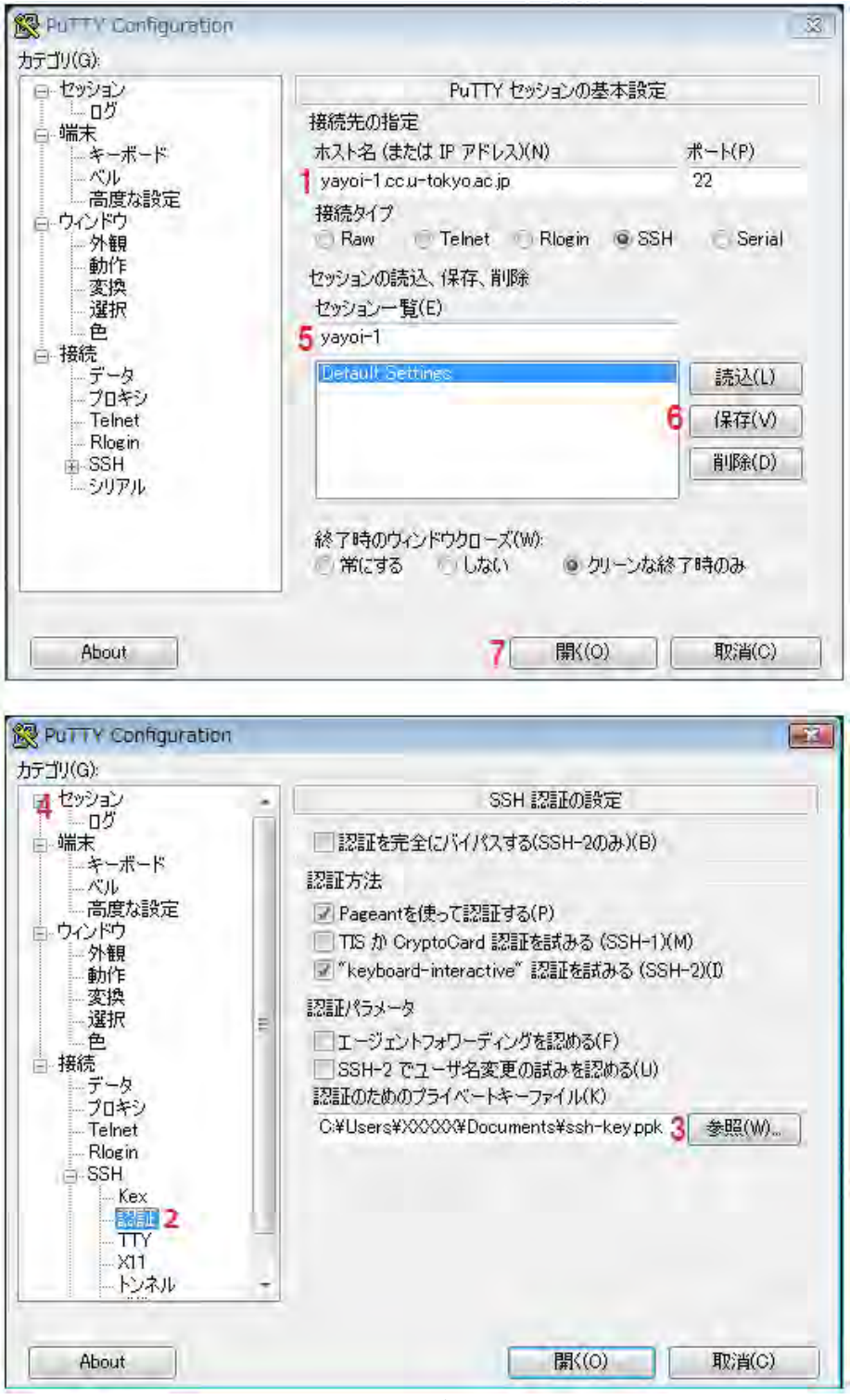


図 2-3-2　ログイン方法

2.3.5 ファイルコピー

ファイルコピーにはWinSCPというソフトが使用できます。WinSCPはPuTTY Key Generator で生成して保存した秘密鍵がそのまま使えます。ホスト名やユーザー名を登録する「セッション登録画面」に秘密鍵ファイルを指定する欄がありますので、秘密鍵ファイルを指定してください。

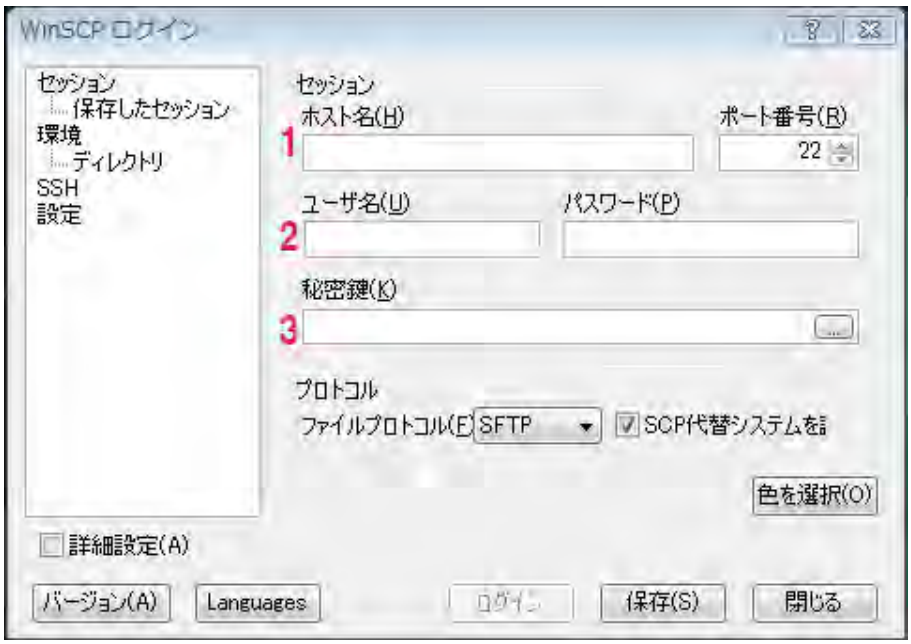


図2-3-3　WinSCPによるファイルコピー

2.4 UNIX システムからのログイン方法

Linux や Mac OS X 、Windows 上の Cygwin からのログインには OS にデフォルトでインストールされている OpenSSH が使用できます。

2.4.1 鍵の生成

次のコマンドを入力してください。

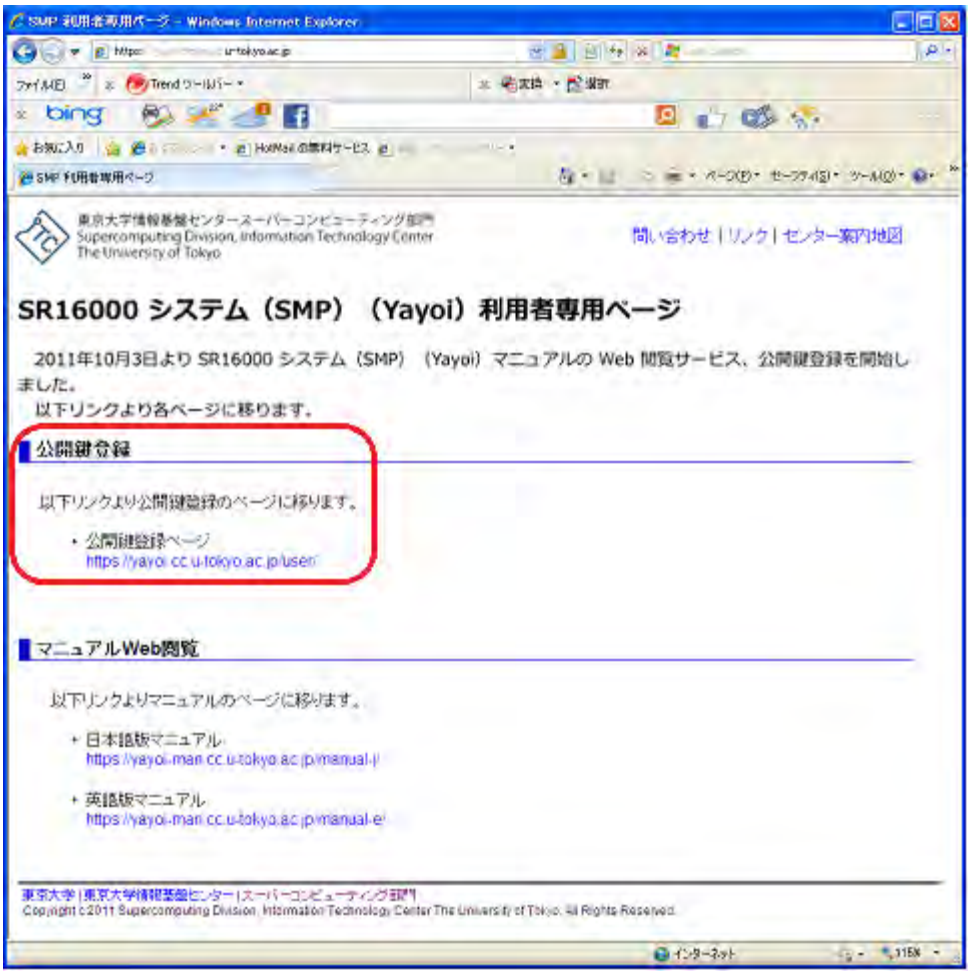
```
$ ssh-keygen -t rsa
```

最初に鍵の保存場所を聞かれます。これはそのままリターン(エンター)キーで大丈夫です。次にパスフレーズを求められるので入力してください。センターから通知されたパスワードとは無関係なので、センターから発行されたパスワードを入力しないように注意してください。また、入力しなくても鍵としては使えますが、セキュリティー強化のため SR16000 に登録する鍵はパスフレーズで保護されていることを必須とします。パスフレーズはメモしたりファイルに書いたりしなくて済むよう、他人には推測されず、自分には覚えやすいものを設定してください。確認のためもう一度パスフレーズを入力したら鍵生成は完了です。

2.4.2 鍵の SR16000 への登録

鍵の登録のために、次の URL にアクセスしてください。

https://yayoi.cc.u-tokyo.ac.jp/



ユーザー名とパスワードを求められるので、センターから通知されたユーザー名とパスワードを入力してください。ここで入力するパスワードは鍵の保護のために入力したパスフレーズではなく、センター発行のパスワードです。認証に成功すると上記の図のような、利用者専用ページが開きますので「公開鍵登録」のリンクをクリックしてください。開いたページの "Public Key"の欄に ~/.ssh/id_rsa.pub の内容をコピー&ペーストし、"Password" の欄にセンター発行のパスワードを入力してください（すでに認証済みですが、セキュリティー上非常に重要な部分なので再度パスワード入力をお願いしております）。「登録」をクリックして完了画面が出れば登録完了です。登録まで少し時間がかかることがありますが、「登録」ボタンを何度もクリックしないようお願いします。登録作業を 2 回以上行うと、最初に登録した鍵は削除されます。また、一度でもログインすると Web 経由での公開鍵登録はできなくなります。複数のコンピュータからログインするために公開鍵を 2 個以上登録する方法や、ログインした後の鍵の変更方法は次節「鍵について」で説明します。

(注) 公開鍵の文字列中に不正な文字（半角スペースなど）が含まれている公開鍵を登録した場合、正しいパスフレーズを用いて SSH ログインしようとしてもエラーになり、ログインできません。現在のところ、Mac OS X や UNIX(Linux)のターミナル上で、less や emacs を使用して文字列（公開鍵）を表示させ、コピー&ペーストすると、SSH 公開鍵では使用されない半角スペースが混在してしまうという問題が発生する場合があることを確認しております。したがって、SSH ログインできない場合、以下のようにターミナル上で cat で出力した文字列（公開鍵）をコピー&ペーストして、再度、公開鍵登録作業をし、SSH ログインを試してみてください。

```
$ cat ~/.ssh/id_rsa.pub
```

2.4.3 ログイン

次のコマンドを実行し、パスフレーズを入力すればログインできます。自動的に公開鍵認証が行われます。

```
$ ssh {ユーザー名}@yayoi-1.cc.u-tokyo.ac.jp
```

(注) 正しい公開鍵を登録したにもかかわらず、SSH ログインの際に「Agent admitted failure to sign using the key.」というエラーメッセージが表示されログインできない場合は、お使いのクライアントマシン上で ssh エージェントが有効になっていると思われますので、お使いのクライアントマシンの管理者にお問い合わせください。もし、ssh エージェントを利用されている場合は、登録した公開鍵の対となる秘密鍵が何らかの原因により、エージェントに認識されていないため、お使いのクライアントマシン上で ssh-add コマンドを利用するケースである可能性があります。

```
$ ssh-add {登録した公開鍵の対となる秘密鍵ファイル}
```

2.4.4 ファイルコピー

次のコマンドを実行し、パスフレーズを入力します。自動的に公開鍵認証が行われます。

```
$ scp file {ユーザー名}@yayoi-1.cc.u-tokyo.ac.jp:
```

2.5 鍵について（Windows、UNIX 共通）

2.5.1 Web 登録システムの有効期間

Web による鍵登録が可能なのは最初に SSH でのログインに成功するまでです。常に登録システムを開いておくと、悪意を持った第三者にパスワードが漏れた場合に登録システムを通じて攻撃者が生成した鍵が登録されてしまい、システムに不正侵入されてしまうためです。鍵が未登録なはずなのに登録システムに登録を拒否された場合は、何者かによって鍵を登録されてしまったことになるので、至急センターまでご連絡ください。

2.5.2 Web 登録システムに入力できる鍵

Web 登録システムは OpenSSH 形式の鍵を受け付けます。Windows では推奨ソフトである PuTTY の他、Tera Term Pro もこの形式の鍵を生成します。

2.5.3 鍵の追加と変更

SSH ログインした後に、他のマシン用の鍵を追加したり、登録済みの鍵を変更する場合は、SR16000 にログインし ~/.ssh/authorized_keys に書かれている公開鍵をエディタで直接編集してください。一行に一個の公開鍵を書きます。鍵の文字列は長いため、多くの場合は複数行にわたって表示されることになりますが、データ上は一個の鍵は一行でなくてはなりません。鍵の途中に改行を入れないようご注意ください。このファイルを削除したり、書かれている公開鍵を壊してしまうとログインできなくなります。

2.5.4 秘密鍵の管理

鍵の生成方法で説明したとおり、秘密鍵を守ることは SR16000 のセキュリティーを守る上で非常に重要なので、パスフレーズや秘密鍵が他人に漏れることのないよう十分ご注意ください。万が一秘密鍵が他人に漏れた場合は必ずセンターに連絡し、適切な処置についてご相談ください。複数のマシンから SR16000 にログインする場合は、それぞれのマシンで鍵を生成して、上で説明した方法で SR16000 の方に鍵を追加します。秘密鍵をコピーして他のマシンで使用してはいけません。

2.5.5 秘密鍵を失った場合

鍵の生成方法で説明したとおり、秘密鍵は SR16000 にログイン、セキュリティーを守るために非常に重要なものです。そのため、秘密鍵を紛失する (鍵を生成したマシンが故障するなどして、秘密鍵を失ってしまう) と、SR16000 へのログインができなくなってしまいます。この場合には、新たに SR16000 にログインするためのマシンで、鍵の生成と SR16000 側への公開鍵登録が必要となりますが、SR16000 へのログインが一度でも行ったことがある場合には、Web 経由での公開鍵登録はできない状態です。そのため、ユーザー名と再度公開鍵登録を行えるようにしてほしい旨と紛失した理由をご記入の上、本センター宛てまでメールでご連絡ください。

2.5.6 別システムへのログインに使用している鍵の流用

１台のクライアントマシンから SR16000 と他の公開鍵認証を用いるシステムにログインする場合、他のシステムに登録した鍵を SR16000 にも登録することは差し支えありません。ただし、他のシステム用に登録していた鍵がパスフレーズで保護されていない場合は SR16000 には登録できません。一般的に、パスフレーズのない鍵の使用は非常に危険なので、その場合は、別のシステムに登録していた鍵の方をパスフレーズ付きに変更し、それを SR16000 にも登録することをお勧めします。

お使いの UNIX システム上で複数の鍵を使い分けたいときは ~/.ssh/config に次のように書きます。
（例）yayoi-*.cc.u-tokyo.ac.jp には秘密鍵 ~/.ssh/id_rsa_smp を使用する時

```
Host yayoi-*.cc.u-tokyo.ac.jp
  User {ユーザー名}
  IdentityFile ~/.ssh/id_rsa_smp
```

3

ログインノードの使用

3.1 概要

利用者はすべての作業をログインノードで行います。本章ではログインノードを使用するにあたって重要なコマンドの使い方およびログインノードで使用できるソフトウェアについて紹介します。

3.2 シェルの種類

ログインノードでは標準で csh が設定されています。本章ではすべて csh を前提に説明します。ログインシェルは chsh コマンドで変更可能です。変更可能なログインシェルについては以下をご覧ください。

```
/bin/bash, /bin/tcsh, /bin/bssh, /bin/csh, /bin/zsh
```

3.3 初期設定ファイル

ログイン時に一回だけ実行されるスクリプトが ~/.cshrc です。このファイルに設定を書き込むことでユーザー独自の環境を自動的に設定できます。通常、コマンドサーチパスを設定する環境変数 PATH の設定をカスタマイズしたり、シェルの動作をカスタマイズするためのシェル変数をセットしたりします。~/.cshrc で設定することの多い重要な環境変数は次の 2 種類です。

PATH
: コマンドを探すディレクトリを設定します。自分のホームディレクトリにソフトをインストールした場合はインストール先に PATH を通すと便利な場合があります。

LD_LIBRARY_PATH
: 動的ライブラリの検索対象ディレクトリを設定します。自分のホームディレクトリにライブラリをインストールした場合、設定が必要な場合があります。いずれの環境変数もシステムデフォルトの値を消してしまわないように注意が必要です (センター初期値に戻したい場合には、/usr/local/skel/.cshrc にあるファイルをコピーするか、参考にしてください)。次に例を示します。

```
setenv PATH /home/p11000/software/bin:${PATH}
setenv LD_LIBRARY_PATH=/home/p11000/software/lib:${LD_LIBRARY_PATH}
```

3.4 利用できるプログラム

ログインノードには計算ノードとまったく同じ仕様の計算機が使われています。したがって 32 個のコアがありますが、多くのユーザーで共有する資源ですので一人同時 1 コアの使用をルールとさせていただきます。並列化されたプログラムの実行や並列でのプログラムコンパイルはご遠慮ください。これは同時に複数のターミナルからログインすることを禁止するものではありません。

3.5 プログラムの強制終了

ログインノードでは他のユーザーと譲り合って資源をご使用ください。特に暴走して CPU を浪費しているだけのプロセスを残さないようにご注意ください。ログインノードでプログラムが正常終了しない場合、フォアグラウンドで実行している場合は Ctrl + C でプログラムを停止できます。バックグラウンド実行の場合は ps コマンドでそのプロセスのプロセス ID を調べ、kill コマンドを実行します。

```
[p11000@yayoi-1 p110000]% ps
      PID      TTY    TIME CMD
  3277180 pts/16   0:05 ./a.out
  4129444 pts/16   0:00 ps
  3998580 pts/16   0:00 -csh
[p11000@yayoi-1 p110000]% kill 3277180
```

ただし、この方法の kill はプログラムによって拒否されることがあります。どうしても強制終了が必要なときは kill コマンドに –KILL オプションを付けます。また、プロセス番号の -1 は自分に権限の及ぶ全プロセスを意味します。大量のプロセスが暴走を始めたときなど、マシン上の自分の全プロセスを緊急強制終了する必要がある場合は次のコマンドを入力します。これを実行した場合は暴走プロセスのみならず、ログインシェルも終了するので、強制ログアウトとなります。また、すべてのプログラムが直ちに強制終了となるので終了処理が定義されているプログラムであっても終了処理は実行されません。

```
$ kill -KILL -1
```

3.6 独自アプリケーションのインストール

ログインノードではシステム標準のアプリケーションの他、ユーザーが自身でインストールしたプログラムも（並列化されていなければ）実行可能です。ファイルシステムは計算ノードとも共有されているのでログインノードでインストール作業を行い、バッチジョブからそれを使用することもできます。ユーザーディレクトリにプログラムをインストールする際に特別な連絡をセンターにいただく必要はありません。

3.7 制限事項

SR16000 では利用者の秘密保持の観点から通常の UNIX システムでは利用可能なコマンドの一部が利用できません。例えば w、ps など、他の利用者が実行しているプロセスの名前が取得できるコマンドは利用できない、あるいは自プロセスの情報取得のみに限定されます。また、ログイン中の利用者がわかるようなコマンドも実行できません。同一グループ内の利用者であっても、他の利用者の情報は取得できません。不便なこともあるかと思いますがご理解とご協力をお願いいたします。

3.8 メール転送

バッチジョブの終了通知などのメールはジョブを投入したログインノードに電子メールで通知されます。通知先はアカウント申請の際に記入していただいた連絡用メールアドレスではありませんのでご注意ください。この通知メールは利用者の方が普段使用されているメールアドレスに転送することをお勧めします。転送する場合は次の設定を ~/.forward に記述してください。

```
\p11000
p11000@mail.ecc.u-tokyo.ac.jp
```

1行目は SR16000 にもメールを残す場合に必要です。バックスラッシュ（端末によっては円記号）に続けて自分のログイン名を記述してください。SR16000 にメールを残す必要がない場合はこの行は不要です。2行目は転送先のメールアドレスです。任意のメールアドレスが設定できます。SR16000 にメールを残した場合は mail コマンドで確認ができます。POP や IMAP などメール取り込みのためのサービスはありません。

3.9 quota 情報 (la コマンド)

SR16000 では、現在保有しているディスク使用量、4 月からの CPU 時間積算値などは、la コマンドを実行することで、確認できます。
ただし、表示される値については、システムが定期的に集計した結果を表示するため、最新の使用量 (CPU 利用実績) とは異なる場合がありますのでご注意ください。

```
[p11000@yayoi-1 p11000]% la -d (ディスク使用量の確認)
disk resource      user(p11000)
 file system       limit(MB)    occupied(MB)
 /home                  512000             1
```

3.10 無入力打ち切り (autologout 機能)

SR16000 では、2 時間端末を操作しなかった場合に、端末を強制終了いたします。警告メッセージは、強制終了される 30 分前より出力されます。

```
Warning message from login manager at Sat Oct   1 12:40:00 2011
          *** Session autologout message ***
 Session pts/12 (p11000@yayoi-1) is not accessed; going logout in 30 minutes.
 Warning: If you input any string, autologout is canceled.
             Once logout time arrives, your jobs will be killed.
             Please prepare your jobs for logout.

[[ 30 分前に警告メッセージが出力される。以降 10 分間隔で出力される ]]

Warning message from login manager at Sat Oct   1 12:50:00 2011
          *** Session autologout message ***
 Session pts/12 (p11000@yayoi-1) is not accessed; going logout in 20 minutes.
 Warning: If you input any string, autologout is canceled.
             Once logout time arrives, your jobs will be killed.
             Please prepare your jobs for logout.

Warning message from login manager at Sat Oct   1 13:00:00 2011
          *** Session autologout message ***
 Session pts/12 (p11000@yayoi-1) is not accessed; going logout in 10 minutes.
 Warning: If you input any string, autologout is canceled.
             Once logout time arrives, your jobs will be killed.
             Please prepare your jobs for logout.

[[ 無入力時間が 2 時間になった時点で以下のメッセージを出力し、強制終了 ]]

Warning message from login manager at Sat Oct   1 13:10:00 2011
          *** Session logout time has arrived ***
 Session pts/12 (p11000@yayoi-1) going logout IMMEDIATELY.

Connection to yayoi-1 closed by remote host.
Connection to yayoi-1 closed.
```

3.11 初期パスワードの変更

センターから発行された初期パスワードは、最初に SR16000 を利用するための鍵登録と、Web 経由で閲覧可能な利用者マニュアルを参照する場合に使用します。
鍵登録後 (SSH ログイン完了後) にセンターから発行された初期パスワードを変更したい場合には、**chuser_passwd** コマンドを実行することで、変更することができます。**passwd コマンドを実行しても、初期パスワードは変更されませんのでご注意ください。**

```
[p11000@yayoi-1 p11000]% chuser_passwd      ← パスワード変更用コマンドの実行
p11000 change password start

        <<Password Policy>>
        6 characters or more
        Character complexity

NewPassword:                                ← 新しいパスワードを入力 (画面には表示されません)
RetypePassword:                             ← もう一度、新しいパスワードを入力 (画面には表示されません)
Wait a moment ...                           ← 登録までしばらくお待ちください

p11000 change password finished             ← パスワードの変更が成功した
[p11000@yayoi-1 p11000]%
```

変更するパスワードには、利用者番号や回文、または容易に推測できるような文字列は設定できません。パスワードの設定には、6 文字以上、英大文字や数字、記号等を入れるようにしてください。
パスワード変更時に以下のようなエラーメッセージが出力された場合には、登録しようとしたパスワード文字列、文字数について修正のうえ再登録をお願いします。

エラーメッセージ	対処方法
Error : Enter at least 6 characters. Please try again.	パスワードが 6 文字未満の場合に出力されます。6 文字以上のパスワードを設定してください。
Error : Cannot set a simple password. Please try again.	パスワードが利用者番号と同じなど、容易に推測できるような文字列が設定されている場合に出力されます。
Error : The password does not meet the password policy requirements. Check the minimum password length and password complexity.	パスワードが容易に推測できるような文字列が入力された場合に出力されます。
Error : Password incorrect. Please try again	最初と 2 度目パスワードが異なっている場合に出力されます。
You cannot try any more. Sorry.	パスワード変更を 3 回以上間違った場合に出力されます。この場合、コマンドを終了します。

※ 上記のエラーメッセージが複数出力される場合があります。
上記のエラーメッセージ以外が出力される場合がありますが、その場合は、表示されるメッセージに従い対処をお願いいたします。

4

ファイルシステム

4.1 概要

SR16000 では約 500GB のストレージが使用できます。利用者データの格納場所としては並列ファイルシステム上にある、ホームディレクトリ (/home)、計算中の一時データ格納領域としては 5 日で削除される Quota(容量)制限のない領域 (/short) などが使用できます。本章ではこれらのディスク領域の特徴や典型的な使い方を紹介します。
センターではホームディレクトリなどのファイルシステムのバックアップ等は取得しておりませんので、利用者の皆様でバックアップの取得等をお願いいたします。

4.2 使用可能領域

利用者データの保存には以下のディレクトリが使用できます。

- /home/ログイン名

一時データの格納場所としては次の場所が使用できます。

- /short/ログイン名
- /tmp

/home/ログイン名 （各コースの上限容量まで使用可）

/home はホームディレクトリであり，ここに作られたファイルは利用者自身で削除するまで保存されます。この領域にはすべてのノードからアクセスできます。ファイルシステムには GPFS (General Parallel File System)が使用されています。GPFS は複数のファイルサーバに負荷を分散する機能を持ち、多数のノードからの大量入出力を処理することができます。ホームディレクトリは、500 GB 単位で容量の追加を行うことも出来ます (別途負担金が発生します)。

/short/ログイン名 （Quota 制限なし）

一時的に必要なデータ等を置くためのディレクトリです。ここに作られたファイルは作成（または最終更新）から 5 日後に削除されます。ファイルシステムは GPFS で、すべてのノードから同じファイルにアクセスできます。ただし、利用者全員でご利用頂くファイルシステムですので、ファイルシステムを圧迫する (大容量、大量ファイルを作成するなど、ファイルシステムの大部分を占有している) 場合には、保存期間前でも削除される場合があります。

/tmp （Quota 制限なし）

短時間の作業のために使用するディレクトリです。ここに作られたファイルはログインノードでは 2 日以内、計算ノードではジョブ終了時に削除されます。計算ノード、ログインノードとも、各ノードのローカルディスクなので他のノードからのアクセスはできません。

ファイルシステム		Quota 制限	目的・用途
/home/利用者番号	GPFS	500 GB	ホームディレクトリ。500 GB 単位で増量可能 (追加負担金が必要)。
/short/利用者番号	GPFS	なし	短期利用。5 日間で削除。
/tmp	-	なし	一時利用。計算ノードでは、ジョブ終了時に削除。ログインノードでは、2 日以内に削除。

5

並列計算

5.1 概要

スーパーコンピューターは多数のプロセッサを同時に使用することで高い性能を得る計算機です。多数のプロセッサを用いるためにはアプリケーションが並列化されている必要があります。プログラムの並列化には主に２種類の方法があります。

- 共有メモリを用いた並列化
- ネットワーク通信による並列化

前者は主に複数のスレッドを用いる並列化のことであり、自動並列化、OpenMP などの並列化手法があります。後者は複数のコンピューターを用いて通信をしながら並列に計算を進める方法であり、MPI 通信ライブラリを用いて並列計算に必要な通信を記述するのが一般的です。
SR16000 では両方の並列化手法が使用できます。ただし、本センターでは複数のノードを用いた並列計算を行っていただくことに重点を置いているため、共有メモリを用いた並列化のみを行い、ノード間の通信を行わないアプリケーションを実行するためのみのコースは用意しておりません（最適化などのため１ノードジョブを流す必要性もありますので、単一ノードジョブも実行は可能です）。つまり、本システムにおいては主に MPI 通信によるアプリケーションの並列化と、さらに必要に応じて、ノード内での並列化を最適化するために共有メモリを用いた並列化を行うことになります。
本章では共有メモリを用いた並列化、MPI による並列化、およびその双方を用いた並列化についてその概念を説明します。本章で説明する概念を理解することはコンパイラやバッチジョブシステムを正しく使用するために重要となります。

5.2 共有メモリを用いた並列化

前述のように SR16000 は共有メモリを用いた並列化のみを行った単一ノードアプリケーションに重点を置いてはいませんが、MPI による通信を行うプロセスをさらに共有メモリを用いて並列化する手法は一般的です。本節ではこの共有メモリを用いた並列化の方法を紹介します。

5.2.1 自動並列化

自動並列化はプログラムの並列化をすべてコンパイラに任せる方法です。コンパイラはプログラムの並列性を自動的に見つけ出し並列化されたコードを生成します。

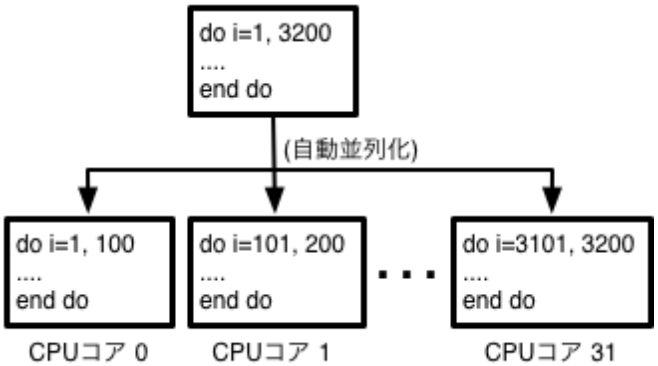


図 5-2-1　自動並列化のイメージ

この方法では逐次プログラムを書き換えることなく並列化が可能です。もっとも容易な並列化手法ですが、並列性を意識せずに書かれたプログラムは本質的に並列化が難しいコードになる傾向が強いため高い並列性能が得られない場合があります。
性能向上のために自動並列化のためのコンパイラへのヒントをソースコード中に記述できるコンパイラも存在しますが、そのようなコードはコンパイラ依存になってしまいます。ソースコードに並列化指示を記述する場合は次に説明する OpenMP を使用することをお勧めします。SR16000 で導入された日立製作所製のコンパイラでは自動並列化の一種として「要素並列化」と呼ばれる機能が提供されており、IBM 製 C/C++コンパイラでも自動並列化機能が提供されております。利用方法は次章で紹介します。

5.2.2 OpenMP

OpenMP はプログラマによってコンパイラに並列化を指示するための言語拡張仕様です。多くのコンパイラに実装されているためほとんどの並列計算機で使用可能な一般的な方法となっています。この方法ではユーザーが並列化方法を制御できるため正しく OpenMP コードを挿入した場合の性能は自動並列化より高くなることが期待できます。反面、誤った並列化指示は計算結果の誤りにもつながるので注意が必要です。

SR16000 では日立製作所製の最適化コンパイラ、IBM 製 C/C++コンパイラに OpenMP オプションを付けることで使用できます。使用方法の詳細は次章で紹介します。

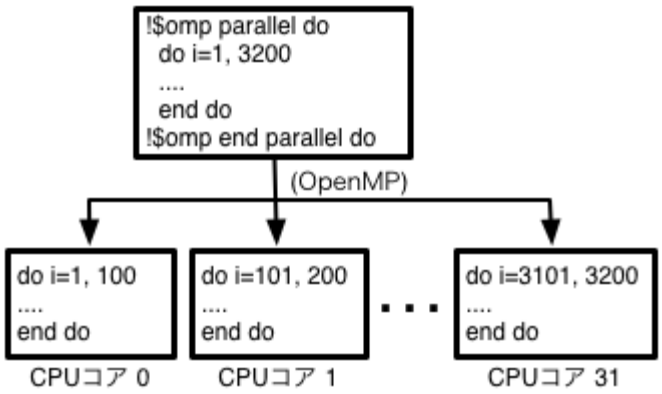


図 5-2-2　OpenMP 使用イメージ

5.2.3 並列化されたライブラリの使用

BLAS などのライブラリの中にはライブラリ自体が並列化されているものがあります。ライブラリの中での計算が実行時間の大半を占めるようなアプリケーションでは並列化されたライブラリをリンクするだけで十分な並列化効率が得られる場合もあります。

並列化されたライブラリを使用するときは呼び出し元の並列化方法に注意する必要があります。例えば OpenMP で並列化された部分からさらに並列化されたライブラリを呼び出すとプロセッサ数の 2 乗の数のスレッドが生成され性能が大きく低下する可能性があります。

5.2.4 日立製作所独自仕様の並列化

日立製作所製のコンパイラでは SECTION 型要素並列化と呼ばれる並列化手法が提供されています。旧システムの HA8000 クラスタシステムや SR11000 など、本センターの他のマシンでこの SECTION 型要素並列化を使用して並列化を行ったプログラムをお持ちの方は SR16000 でも引き続き SECTION 型要素並列化にて使用できます。新しく作成するプログラムに関しては特殊な事情がない限り、なるべく汎用的な方法での並列化をお勧めします。

5.3 MPI による並列化

MPI による並列プログラムは SPMD(Single Program Multiple Data)モデルと呼ばれ、単一のプログラムを多数のノードで実行し、それぞれのノード(CPU コア)で動くプログラムが自身のノードで行うべき計算を判断して実行することにより並列に計算を行います。図 5-3-1 に MPI を使ったプログラムのイメージ(1 ノードあたり 1 プロセスで実行した場合の例)を示します。
この例でのプログラムの実行は各ノードで行われますが、図をよく見ると 2 つのノードで実行されるプログラムはまったく同じ内容であることがわかります。このように MPI では基本的にすべてのノードで同じプログラムを実行し、プログラムの中で自身のランク(ノード番号あるいはプロセス番号)を調べてそれに応じた処理を行います。

ノード 0

```
MPI_Init(&argv, &argc);
MPI_comm_rank(MPI_COMM_WORLD,&myrank);
if (myrank == 0) {
/* ノード 0 の計算 */
  /* ノード 1 から結果を受け取る */
  MPI_Recv(&result, ..., 1.....);
  printf("result = %d", ...);
} else if (myrank == 1) {
/* ノード 1 の計算 */
  /* ノード 0 に結果を送る */
  MPI_Send(&result, ..., 0, .....);
}
MPI_Finalize();
```

ノード 1

```
MPI_Init(&argv, &argc);
MPI_comm_rank(MPI_COMM_WORLD,&myrank);
if (myrank == 0) {
/* ノード 0 の計算 */
  /* ノード 1 から結果を受け取る */
  MPI_Recv(&result, ..., 1.....);
  printf("result = %d", ...);
} else if (myrank == 1) {
/* ノード 1 の計算 */
  /* ノード 0 に結果を送る */
  MPI_Send(&result, ..., 0, .....);
}
MPI_Finalize();
```

図 5-3-1　MPI のプログラム例

SR16000 ではノード間およびノード内のプロセス間通信に MPI が使用できます。ノード内通信、ノード間通信ともに MPI で行う場合は、ノード内外の区別をすることなくすべての並列化作業を MPI 通信によって行います。このようなプログラムを 1 ノードあたり複数のプロセスが配置されるような設定で実行すれば MPI 通信ライブラリが自動的にノード内およびノード間それぞれに適した方法で通信を行います。このような実行形態を一般的にピュア MPI（またはフラット MPI）と呼びます。

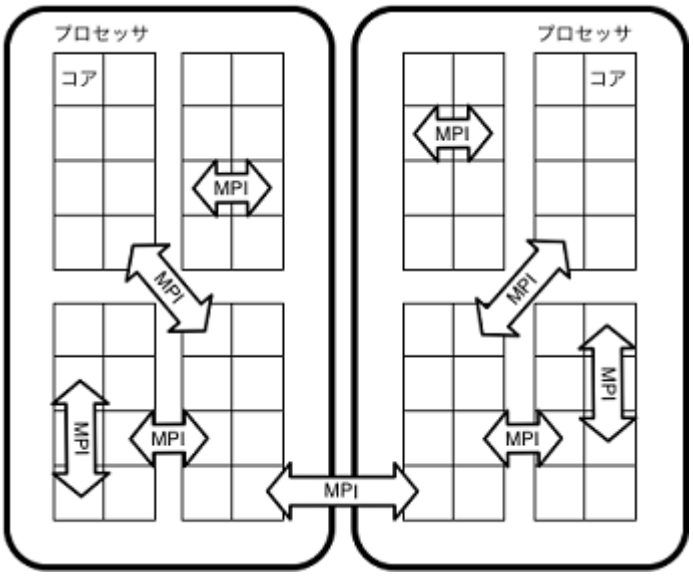


図 5-3-2　ピュア MPI での実行イメージ

5.4 共有メモリによる並列化と MPI 並列化の併用

前節で説明したピュアMPIでは各プロセスが行う通信が細かくなりすぎるなどの問題で高い性能が得られない場合があります。このような場合は OpenMP や自動並列化などの共有メモリを用いた並列化と MPI を併用することになります。例えば、１ノードにOpenMPで８スレッドに並列化したプロセスを４つ生成することで32個の物理コアを使用するような方法が考えられます。このような実行方法を「MPI+OpenMPハイブリッド実行」などと呼びます。
このような場合は、MPIプログラムを自動並列化したり、MPIプログラムにOpenMP指示行を加えたりしてプログラムをコンパイルします。実行時にはプロセスあたりの並列化数（スレッド数）とMPIプロセス数双方を正しく設定する必要があります。

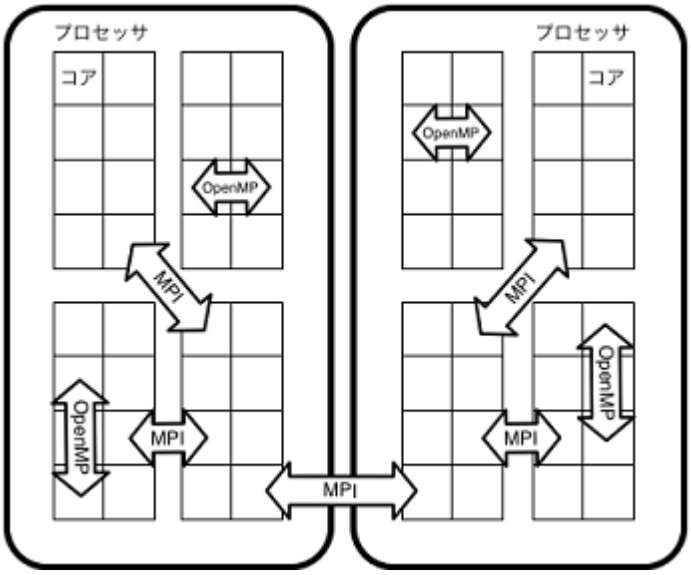


図5-4-1　MPIとOpenMPの併用イメージ

6

コンパイラ

6.1 概要

SR16000 では日立製作所製最適化 C、C++、FORTRAN90 コンパイラを標準のコンパイラとしています。このコンパイラは本センターでサービスを提供していた過去のスーパーコンピュータ (SR11000, HA8000 クラスタシステムなど)のコンパイラと高い互換性を持つため、本センターを利用したことのある利用者はプログラムやコンパイル方法を変更することなく SR16000 を利用できます。また、このコンパイラは高速な逐次コードを生成するだけでなく、高度な自動並列化機能を有することで知られており、本システムで初めてスーパーコンピュータを利用するユーザーにもお勧めできるコンパイラです。
SR16000 には日立製作所のコンパイラの他、IBM 製 C/C++コンパイラもインストールされています。
さらに本システムでは GNU Compiler Collection (gcc) も利用できます。機能、性能とも商用コンパイラには及ばないことが多いですが、広く使われているコンパイラであるため多くのソフトウェアでサポートされているという利点があります。
本章では SR16000 で使用できるこれらのコンパイラの使い方を、並列プログラムのコンパイル方法および最適化オプションを中心に紹介します。コンパイラの使い方は並列化の手法によって使用するコマンドと必須オプションが異なります。本章では以下の場合についてコマンドと必須オプションを列挙します。

並列化なし
: 並列化を行わない場合のコンパイル方法です。逐次実行が前提のライブラリや逐次部分のみを含んだプログラムのコンパイル方法です。

自動並列化
: コンパイラによって自動的に並列化するときに使用するコンパイル方法です。

OpenMP
: 並列化のための OpenMP 指示文を含んだプログラムのコンパイル方法です。

MPI
: ピュア MPI 実行する並列プログラムのコンパイル方法です。

MPI + 自動並列化
: MPI によって並列化したプログラムをさらに自動並列化によってノード内を並列化するときに使用するコンパイル方法です。

MPI + OpenMP
: MPI によって並列化したプログラムをさらに OpenMP によってノード内を並列化するときに使用するコンパイル方法です。

6.2 日立製作所製 最適化 C/C++コンパイラ

本節では日立製作所の最適化 C、C++コンパイラについてコンパイルに使用するコマンドおよび重要な最適化オプションについて紹介します。

6.2.1 推奨オプション

難しい説明を飛ばしてとりあえず使ってみたい方は次のようにコンパイルして下さい。典型的な数値計算プログラムでは通常安全なオプションですが場合によっては誤った結果となるやや危険なオプションを含んでいます。

```
$ cc -Os -noparallel +Op program.c
$ mpicc -Os -noparallel +Op mpi-program.c
```

6.2.2 コマンド

C 言語のコンパイルコマンドは cc、C++言語のコンパイルコマンドは sCC となります。以下に使い方を示します。角括弧は省略可能（またはデフォルトで設定されている）ことを意味します。

並列化なし

```
$ cc [-noparallel] [options] source_file
$ sCC [-noparallel] [options] source_file
```

自動並列化

```
$ cc -parallel [options] source_file
$ sCC -parallel [options] source_file
```

OpenMP

```
$ cc -parallel -omp [options] source_file
$ sCC -parallel -omp [options] source_file
```

MPI

```
$ mpicc [-noparallel] [options] source_file
$ mpisCC [-noparallel] [options] source_file
```

MPI + 自動並列化

```
$ mpicc -parallel [options] source_file
$ mpisCC -parallel [options] source_file
```

MPI + OpenMP

```
$ mpicc -parallel -omp [options] source_file
$ mpisCC -parallel -omp [options] source_file
```

6.2.3 最適化オプション

以下に示すレベルに従って最適化することができます。オプション無指定時はレベル 3 でコンパイルされます。なお、最適化機能には副作用を含むものがあり、プログラムによっては計算結果が異なったり、エラーが生じる場合があります。

-O0	原始プログラムどおりのコンパイル、レジスタの効果的な使用方法を中心に文の実行順序を変更しない一文単位の最適化（べき演算の乗算化、文関数のインライン化、局所的なレジスタの割り当て等）
-O3 デフォルト	プログラム全体での大域的な最適化（分岐命令の最適化、命令の並べ替え、演算子の変更、外部手続きインライン展開等）
-O4	制御構造の変換、演算順序の変更を含むプログラム全体の最適化（演算順序の変更、除算の乗算化、ループ展開・交換・分配・融合等）
-Os	一般的に性能向上が期待できる多くのオプションをまとめて設定（後述） -parallel が設定されるので注意

これらの最適化オプションの他、性能に大きく影響することが知られているオプションに次のものがあります。

+Op	関数仮引数内のポインタ間に依存関係がないことを仮定する。例えば配列へのポインタ、p および q を取る関数で、p のどの要素を変更しても q の要素に影響がないことを仮定する

6.2.4 自動並列化オプション

自動並列化には並列化のレベルに応じて次のようなオプションがあります。

-noparallel -parallel=0	並列化をしない
-parallel=1	SECTION 型要素並列化、強制ループ並列化
-parallel -parallel=2	変数・配列のプライベート化、リダクション並列化
-parallel=3	ループ分配、ループ分割、ループの一重化、サイクリック分割
-parallel=4	パイプライン並列化、インダクション並列化

6.2.5 その他のオプション

分割コンパイル、リンクなどに使用するオプションは gcc など一般的なコンパイラと共通です。

-I<dir>	ヘッダファイルを探すディレクトリを指定する
-c	コンパイルのみを行う
-L<dir>	ライブラリを探すディレクトリを指定する
-l<lib>	ライブラリをリンクする

6.3 日立製作所製 最適化 FORTRAN90 コンパイラ

本節では SR16000 の標準 FORTRAN コンパイラである日立製作所製最適化 FORTRAN90 コンパイラの使い方を紹介します。

6.3.1 推奨オプション

難しい説明を飛ばしてとりあえず使ってみたい方は次のようにコンパイルして下さい。典型的な数値計算プログラムでは通常安全なオプションですが場合によっては誤った結果となるやや危険なオプションを含んでいます。

```
$ f90 -Oss -noparallel program.f90
$ mpif90 -Oss -noparallel mpi-program.f90
```

6.3.2 コマンド

日立製作所製最適化 FORTRAN90 コンパイラは f90 コマンドで起動します。f77 コマンドはありませんが、FORTRAN77 プログラムも f90 コマンドでコンパイルすることができます。

並列化なし

```
$ f90 [-noparallel] [options] source_file
```

自動並列化

```
$ f90 -parallel [options] source_file
```

OpenMP

```
$ f90 -parallel -omp [options] source_file
```

MPI

```
$ mpif90 [-noparallel] [options] source_file
```

MPI + 自動並列化

```
$ mpif90 -parallel [options] source_file
```

MPI + OpenMP

```
$ mpif90 -parallel -omp [options] source_file
```

言語仕様と形式はソースファイルの拡張子によって次のように設定されます。

f	固定形式
f77, f90, f95	各言語の自由形式
F	固定形式（コンパイル前にＣプリプロセッサを使用）
F77, F90, F95	各言語の自由形式（コンパイル前にＣプリプロセッサを使用）

6.3.3 言語オプション

拡張子に関わらず規格や形式を指定するには次のオプションを用います。

-fixed	ファイルの拡張子によらず固定形式を用いる
-free	ファイルの拡張子によらず自由形式を用いる
-hf77	ファイルの拡張子によらず FORTRAN77 としてコンパイルする
-hf90	ファイルの拡張子によらず Fortran90 としてコンパイルする
-hf95	ファイルの拡張子によらず Fortran95 としてコンパイルする

6.3.4 最適化オプション

以下に示すレベルに従って最適化することができます。オプション無指定時はレベル 3 でコンパイルされます。なお、最適化機能には副作用を含むものがあり、プログラムによっては計算結果が異なったり、エラーが生じる場合があります。

-O0	原始プログラムどおりのコンパイル、レジスタの効果的な使用方法を中心に文の実行順序を変更しない一文単位の最適化（べき演算の乗算化、文関数のインライン化、局所的なレジスタの割り当て等）
-O3 デフォルト	プログラム全体での大域的な最適化（分岐命令の最適化、命令の並べ替え、演算子の変更、外部手続きインライン展開等）
-O4	制御構造の変換、演算順序の変更を含むプログラム全体の最適化（演算順序の変更、除算の乗算化、ループ展開・交換・分配・融合等）
-Os	ソフトウェアパイプライン、要素並列化を含む各種最適化オプションをまとめて設定(後述 6.4.1)。-parallel が設定されるので注意
-Oss	一般的に性能向上が期待できるほとんどの最適化オプションをまとめて設定(後述 6.4.1)。-parallel が設定されるので注意

-Oss に含まれませんが、次のオプションも高速化につながる可能性があります。

-autoinline	関数のインライン展開を行う

6.3.5 自動並列化オプション

自動並列化には並列化のレベルに応じて次のようなオプションがあります。

-noparallel -parallel=0	並列化をしない
-parallel=1	SECTION 型要素並列化、強制ループ並列化
-parallel -parallel=2	変数・配列のプライベート化、リダクション並列化
-parallel=3	ループ分配、ループ分割、ループの一重化、サイクリック分割
-parallel=4	パイプライン並列化、インダクション並列化

6.3.6 その他のオプション

分割コンパイル、リンクなどに使用するオプションは一般的なコンパイラと共通です。

-I<dir>	ヘッダファイルを探すディレクトリを指定する
-c	コンパイルのみを行う
-L<dir>	ライブラリを探すディレクトリを指定する
-l<lib>	ライブラリをリンクする

6.4 日立製作所製 最適化コンパイラ オプションの詳細

6.4.1 最適化の内容

前節までで日立製作所製最適化コンパイラの基本的なオプションを紹介しましたが、-Os などの最適化オプションは複数の最適化技術を組み合わせて使用することを示しており、アプリケーションによっては計算結果が変わってしまう最適化技術を含んでいます。以下に最適化オプション -Os または -Oss が行う最適化の種類と、それぞれの最適化手法を個別に選択するためのオプションをまとめます。

表 6-4-1　-Os に含まれるオプション

レベル	含まれるオプション	詳細
-Os C、FORTRAN 共通	-O4	プログラム全体の最適化(前述 6.3.4)
	-approx	除算を逆数による乗算に置きかえる
	-disbracket	括弧や文の順序によって明示された演算順序を変更する
	-divmove	条件式下で除算割り込みが起こる可能性があるループ不変式をループ外に移動する
	-expmove	条件式下にあるループ不変式をループ外に移動する
	-invariant_if	ループ不変条件展開最適化をする
	-ischedule=3	命令の実行順序を変更する。分岐予測を行い投機的に実行順序を変更する
	-loopdistribute	ループ分配による最適化をする
	-loopexpand	ループ展開による最適化をする
	-loopfuse	ループ融合による最適化をする
	-loopinterchange	ループ交換による最適化をする
	-loopreroll	ループ巻き戻しによる最適化をする
	-parallel=2	自動並列化を行う(前述 6.3.5)
	-prefetch	最内側ループ中の配列に対してプリフェッチ最適化をする
	-simd	SIMD 命令による最適化を行う
	-workarray	作業配列を利用した最適化をする

レベル	含まれるオプション	詳細
-Os FORTRAN のみ	-rapidcall	組込み関数に対して呼び出し時の引数チェックを行わない。引数を値渡しとする
	-scope	ループの最適化でスコープ分割をする
	-swpl	ソフトウェアパイプラインによる最適化をする
	-noargchk	サブルーチンおよび関数を引用する際、引数の個数と型、属性、名称の重なりをチェックしない
	-noagochk	プログラム実行時に、割当て形 GO TO 文の文番号並びの有無をチェックしない
	-nosubchk	配列参照の添字の値が、宣言の範囲内にあるかどうかをチェックしない
	-nolcheck	新 FORTRAN 規格から拡張した仕様に対して、エラーメッセージを出力しない
	-nodochk	DO 文の繰り返し回数 0 をチェックしない
-Os C のみ	-autoinline	自動的に関数をインライン展開する

表 6-4-2　-Oss に含まれるオプション

レベル	含まれるオプション	内容
-Oss FORTRAN のみ	-Os	(表 6-4-1)
	-ifswpl	条件分岐を含むループのソフトウェアパイプラインによる最適化をする
	-parallel=4	(前述 6.3.5)

6.4.2 最適化の個別設定と打ち消し

特定の最適化技術のみを使用する場合は表 6-4-1 および表 6-4-2 に示したオプションを個別に設定します。また -Oss や -Os が含む最適化技術の一部を無効化したいときは各オプションに no を付けたオプションを -Oss または -Os の後に置きます。特に、自動並列化を抑止するための -noparallel はピュア MPI プログラムのコンパイル時には必須となります。

```
$ mpif90 -Oss -noparallel mpi-program.f90
```

6.4.3 並列化報告

チューニングのヒントのために並列化の報告を出力することができます。次のように -loglist オプションを付けてコンパイルして下さい。並列化ログはソースファイル名の拡張子を .log にしたファイルとして生成され、元のソースファイルにコメントを加える形で並列化状況が書き込まれます。

```
$ mpif90 -Oss -loglist mpi-program.f90
```

6.5 日立製作所製 最適化コンパイラ 実行時オプションと環境変数

本節は実行時に設定する環境変数または実行時オプションについて説明します。

6.5.1 自動並列化および OpenMP のスレッド数指定

自動並列化または OpenMP による並列化を行った場合、実行時のスレッド数は次の表にある環境変数で指定します。環境変数を設定しなかった場合は理論コア数(SR16000 では常に 64)となります。

環境変数	内容
HF_PRUNST_THREADNUM	自動並列化されたプログラムの実行時スレッド数
OMP_NUM_THREADS	OpenMP で並列化されたプログラムの実行時スレッド数

環境変数の具体的な指定方法は後の「使用例」の章に例を載せます。

6.5.2 FORTRAN の実行時オプション

FORTRAN プログラムの実行時にはプログラムの動作を変更するいくつかのオプションが指定できます。オプションは次のように -F に続けてオプションを書きます。

```
$ ./a.out -F'RUNST(UFLOW),RUNST(OFLOW)'
```

ただし、この方法はシェルの特殊文字との相性が悪く、特にバッチジョブでは正常にプログラムに渡らないことが多いので実行時オプションは上のようにプログラムの引数として指定するのではなく、次の例のように環境変数 HF_90USEROPTS に設定するようにしてください。

```
export HF_90USEROPTS='-FRUNST(UFLOW),RUNST(OFLOW)'
```

主なオプションは次の通りです。詳しくはオンラインマニュアルをご覧下さい。

オプション	内容
RUNST(UFLOW)	浮動小数点演算中にアンダーフローが発生したらエラーとしてプログラムを終了する
RUNST(OFLOW)	浮動小数点演算中にオーバーフローが発生したらエラーとしてプログラムを終了する
PORT(ENDIANINOUT(ALL))	入出力のエンディアンを変換する

6.6 IBM 製 C/C++コンパイラ

本節では IBM 製 C/C++コンパイラについてコンパイルに使用するコマンドおよび重要な最適化オプションについて紹介します。

6.6.1 推奨オプション

とりあえず使ってみたい方は次のようにコンパイルして下さい。

```
$ xlc -qsmp=noauto program.c
```

6.6.2 コマンド

C 言語のコンパイルコマンドは xlc、C++言語のコンパイルコマンドは xlC となります。また、MPI を使用する場合には C 言語のコンパイルコマンドは mpixlc、C++言語のコンパイルコマンドは mpixlC となります。以下に使い方を示します。角括弧は省略可能を意味します。

並列化なし

```
$ xlc -qsmp=noauto [options] source_file
$ xlC -qsmp=noauto [options] source_file
```

自動並列化

```
$ xlc −qsmp=auto [options] source_file
$ xlC −qsmp=auto [options] source_file
```

OpenMP

```
$ xlc −qsmp=omp [options] source_file
$ xlC −qsmp=omp [options] source_file
```

MPI

```
$ mpixlc [options] source_file
$ mpixlC [options] source_file
```

MPI + 自動並列化

```
$ mpixlc −qsmp=auto [options] source_file
$ mpixlC −qsmp=auto [options] source_file
```

MPI + OpenMP

```
$ mpixlc −qsmp=omp [options] source_file
$ mpixlC −qsmp=omp [options] source_file
```

6.6.3 最適化オプション

以下に示すレベルに従って最適化することができます。オプション無指定時はレベル 2 でコンパイルされます。なお、最適化機能には副作用を含むものがあり、プログラムによっては計算結果が異なったり、エラーが生じる場合があります。

-O0	原始プログラムどおりのコンパイル。定数の畳み込みなど、局所的な最適化のみ
-O	コンパイル速度と実行時性能の組み合わせで最良となるように最適化
-O2 デフォルト	-O と同じ
-O3	このオプションは-O2 に加え、実行時性能を向上させるため、メモリとコンパイル時間の両方を必要とする最適化を行う。プログラムの意味を若干変更する可能性のある積極的な最適化(計算順序の並べ替え、浮動小数点式の書き換えなど)を行う
-O4	このオプションは-O3 に加え、コンパイル時のマシンの特性に適したオプションを設定する最適化を行う
-O5	このオプションは-O4 に加え、プロシージャ間のデータ・フローを考慮した最適化を行う

6.6.4 自動並列化オプション

自動並列化には次のようなオプションがあります。

-qnosmp (デフォルト)	並列化なしでコンパイルする
-qsmp	サブオプションなしで-qsmp と指定すると、暗黙的に最適化オプション-O2 が有効となる。また、-qsmp を指定すると暗黙的に以下のようにサブオプションを指定したことに相当する -qsmp=auto:explicit:opt:noomp:norec_locks:nonested_par:schedule=runtime:nostackcheck :threshold=100:ostls

-qsmp 指定時のサブオプションのうち、主なものは次のようになります。

-qsmp=auto (デフォルト)	プログラム・コードの自動並列化および最適化を行う
-qsmp=noauto	プログラム・コードの自動並列化を行わない。ただし、SMP または OpenMP ディレクティブによる明示的な並列化指示に対する最適化は行う
-qsmp=noomp (デフォルト)	-qsmp=auto と同じ動作を行う。
-qsmp=omp	OpenMP 標準に対し厳密に準拠して並列化を行う。自動並列化は行わず、OpenMP API に準拠しない言語構成要素がコードに含まれている場合、コンパイラは警告メッセージを出力する(OpenMP 並列化プラグマだけが認識される)
-qsmp=opt (デフォルト)	プログラム・コードの自動並列化および最適化を行う
-qsmp=noopt	自動並列化は行うが、コードの並列化に必要な最小の最適化のみを行う。並列化されたプログラムをデバッグするときにはこの設定を使用する

6.6.5 その他のオプション

分割コンパイル、リンクなどに使用するオプションは gcc など一般的なコンパイラと共通です。

-I<dir>	ヘッダファイルを探すディレクトリを指定する
-c	コンパイルのみを行う
-L<dir>	ライブラリを探すディレクトリを指定する
-l<lib>	ライブラリをリンクする

プログラムのデバッグに利用できるオプションは次のものがあります。

-qinfo	診断メッセージを有効化する
-qheapdebug	デバッグ・バージョンのメモリ管理関数を使用する
-qinitauto=hex_value	未初期化の自動変数を特定の値に初期化する。(デバッグ専用) hex_value は 2 桁の 16 進数
-qoptdebug	元のソースコードより最適化されたプログラムを、ファイル・サフィックス .optdbg つきで作成する。最適化レベル –O3 以上と組み合わせて使用する

アーキテクチャに最適化するオプションは次のものがありますが、デフォルト(何も指定しない場合)では 64bit、POWER7 に最適化されるようになっています。

-q64 (デフォルト)	64bit アドレッシングモデル用のコンパイルを行う
-qarch=pwr7 (デフォルト)	POWER7 アーキテクチャに最適化したコンパイルを行う
-qtune=pwr7 (デフォルト)	POWER7 アーキテクチャに最適化した命令シーケンスを生成する
-qhot	より積極的な HOT 変換を使用可能にして、ループ構成体および配列言語を最適化する。 -O3 と組み合わせて使用する

6.7 GNU Compiler Collection

SR16000 には GNU Compiler Collection (GCC) がインストールされています。一般的に数値計算用途には前節までに紹介した商用コンパイラの方が高性能です。しかし性能がそれほど重要でないツール類やフリーソフトウェアのコンパイルには、世界中のユーザーによってテストされている GCC を使うのが最もトラブルが少ないと期待されます。
GCC のコマンドは C 言語、C++、FORTRAN がそれぞれ gcc、 g++、 g95 です。詳しい使い方は info gcc コマンドなどでご確認下さい。

6.8 分割コンパイルとリンク

規模の大きなプログラムは複数のソースコードに分割し、分割コンパイルを行うことをお勧めします。分割コンパイルは各ソースコードのコンパイル時に -c オプションをつけてコンパイルし、生成されたオブジェクトファイルを最後にリンクします。次の例では上2行がコンパイルのみを行っており、最後の行がリンクを行っています。

```
$ mpicc -parallel -omp -Os -c a.c
$ mpicc -parallel -omp -Os -c b.c
$ mpicc -parallel -omp a.o b.o -o program
```

このようにするとソースの変更があった場合も変更したソースのみをコンパイルするだけで済みます。分割コンパイルをするときは、変更があったファイルのみを自動的にコンパイルできる make と呼ばれるツールを使用するのが一般的です。なお各コンパイラ共、自動並列化、OpenMP 関連のオプションはコンパイル時、リンク時双方に必要です。

7

プログラムの実行

7.1 概要

SR16000 ではすべてのプログラムがバッチジョブシステムを通じて実行されます。SR16000 では 日立製作所製のバッチジョブシステム (NQS 互換機能、以降 NQS) が導入されており、旧システムの HA8000 クラスタシステムと同じジョブスクリプトが使用できます (FX10 スーパーコンピュータシステムとは互換性はありませんのでご注意ください)。
ジョブを実行するにはまず実行するプログラムとは別に「ジョブスクリプト」と呼ばれるファイルを作成します。ジョブスクリプトには希望する実行時間に見合ったキューの名前、使用 CPU 数などを記述した上で、実際にプログラムを実行するためのコマンドを書きます。このファイルが完成したら qsub コマンドでこのジョブを NQS に登録し実行を待ちます。ジョブに資源が割り当てられると、ジョブスクリプトに記述したコマンドが自動的に実行されます。実行結果はファイルに残りますのでジョブ投入後はログインしている必要はありません。ジョブの状態は qstat コマンドで確認できるほか、開始と終了はログインノードにメールで通知されます。「第 3 章 ログインノードの使用」で説明した転送設定を行うかジョブスクリプトに送信先を設定することで、普段使用しているメールアドレスにジョブ完了通知を送信することもできます。

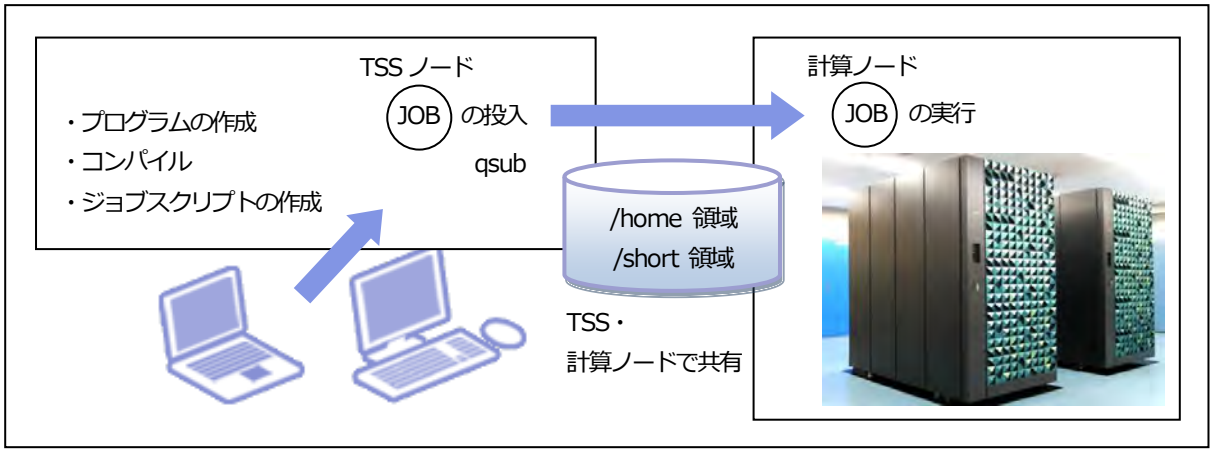


図 7-1-1 ジョブ実行概念図

7.2 パイプキューとバッチキュー

バッチジョブシステムの待ち行列には「バッチキュー」と呼ばれるジョブの待ち行列と「パイプキュー」と呼ばれる他のキューにジョブを転送するためのキューがあります。バッチキューは要求されたプロセッサや計算時間別に準備されており、実行待ちのジョブはバッチキューに入ります。一方パイプキューはジョブの振り分けを行うキューであり、パイプキューに投入されたジョブは必要としている資源量に応じて適切なバッチキューに自動的に転送されます。
SR16000 では図 7-2-1 のようにパイプキューとバッチキューが準備されています (実際に利用できるキューは申込コースによって異なります)。点線で囲まれたバッチキューは必ずパイプキュー経由で使用する必要があり、直接キューを指定してジョブを投入することはできません。debug キューは直接バッチキュー名を指定してジョブを投入します。
※ 利用できるコースの詳細は、「ジョブクラス制限値」をご覧ください。

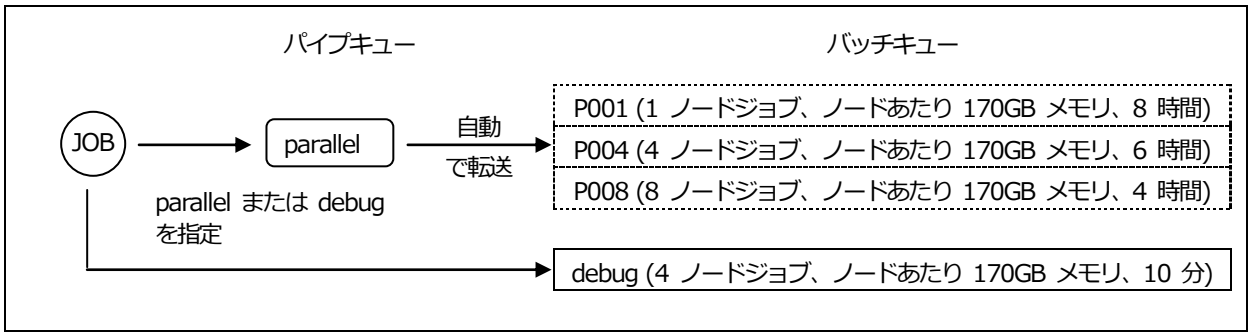


図 7-2-1 SR16000 のキュー (ジョブクラス)

7.3 ジョブスクリプトの書き方

ジョブスクリプトは次のような構成のテキストファイルです。

```
#!/bin/csh                  /bin/csh を指定する
#@$-q parallel
#@$-N 4                     #@$- から始まる行でバッチジョブの属性を設定する
#@$-J T4
#@$-lT 2:00:00

cd program
mpirun ./a.out              実行するシェルスクリプトを記述する
date
```

このようにジョブスクリプトには #@$- から始まるバッチジョブのオプション指定部分とその下のシェルスクリプト部分があります。バッチジョブシステムはオプション部分を使用して投入するバッチキューを決定し、順番が回ってきた際にはスクリプト下部のシェルスクリプトを実行します。「第 9 章 使用例」でも具体的なジョブスクリプトの例を紹介します。ジョブスクリプトは、大文字、小文字が区別されるのでご注意ください。

7.3.1 主要オプション

次に挙げる 2 項目はすべてのジョブスクリプトに必要です。

オプション	内 容	指 定 例
#@$-q	パイプキューの名前	#@$-q parallel
#@$-N	ノード数	#@$-N 4

q および N オプションの値を用いてバッチジョブキューが決定されます。N の値はバッチキューのノード数に一致している必要はありません。例えば　3 ノードを指定した場合は　4 ノード用のバッチキューに投入されます。

次の 2 項目は必須ではありませんが、指定することをお勧めします。

オプション	内 容	指 定 例
#@$-J	ノードあたりのプロセス数	#@$-J T4
#@$-lT	予想実行時間	#@$-lT 2:00:00

J オプションには T に続けてノードあたりのプロセス数を書きます。省略時は 1 となります。T が必要なのは SR11000 以前のシステムとの互換性のためです。SR11000 で使用している S および SS の指定も可能ですが、新たに作成するスクリプトでは使用しないでください。lT（小文字エル、大文字ティー）オプションはジョブ実行の最大時間であり、この時間を超えるとジョブは強制終了となります。lT オプションを省略すると各パイプキューの最大時間とみなされますが、効率的な資源利用のために指定をお願いします。

7.3.2 その他のオプション

必須オプションの他に以下のオプションが使用できます。

オプション	内 容	指 定 例
#@$-lM	ノードあたりのメモリーサイズ	#@$-lM 4GB
#@$-e	標準エラー出力ファイル名	#@$-e file.e
#@$-o	標準出力ファイル名	#@$-o file.o
#@$-eo	標準エラー出力の内容を標準出力ファイルに出力	#@$-eo
#@$-mu	メールの送信先	#@$-mu xxx@xxx.jp
#@$-lc	コアファイルサイズ	#@$-lc 0MB
#@$-ld	プロセス毎のデーターサイズ	#@$-ld 2GB
#@$-ls	プロセス毎のスタックサイズ	#@$-ls 192MB
#@$-lm	プロセス毎のメモリーサイズ	#@$-lm 2GB
#@$-lt	プロセス毎の CPU 時間制限	#@$-lt 30:00

7.3.3 シェルスクリプト部分

シェルスクリプト部分には先頭行で指定したシェルが解釈できるシェルスクリプトを記述します。シェルスクリプトは常にホームディレクトリをカレントディレクトリとして開始します（ジョブスクリプトのあるディレクトリではありません）。したがって通常はプログラムが存在するディレクトリへの移動が必要です。また、このスクリプトは端末から切り離された状態で実行されるのでエディタなどの対話的なプログラムの起動を含めることはできません。

7.4 バッチジョブシステムの使い方

7.4.1 ジョブの投入

ジョブの実行のためにはスクリプトファイルをバッチシステムに投入 (サブミット) します。コマンドは qsub を使用します。以下の例で job.sh は既存のスクリプトファイル名とします。

```
[p11000@yayoi-1 p11000]% qsub job.sh
Request “Request ID” submitted to queue: P004.
```

NQS はジョブを識別するリクエスト ID (リクエスト番号 + ジョブ投入ホスト名) 付加し、バッチシステムにジョブを送り込みます。利用者一人が複数のジョブを投入することもできますが数に制限があります。

7.4.2 ジョブの確認

ジョブの状態（待機中や実行中）は qstat コマンドで知ることができます。

```
[p11000@yayoi-1 p11000]% qstat
  2012/07/23 (Mon) 11:54:00:      REQUESTS on SR16000
  NQS schedule stop time : 2012/07/27 (Fri)   9:00:00 (Remain:   93h   6m   0s)
     REQUEST        NAME        OWNER        QUEUE      PRI    NODE   E-TIME    MEM      STATE
  39550.yayoi       STDIN       p11000        P004       63       2      600s    170GB   QUEUED
```

リクエスト ID、スクリプト名、ログイン名が表示されている行が先程投入したジョブの状況です。ジョブはバッチキュー P004 で待機（QUEUED）していることがわかります。出力の 2 行目には次の計画停止までの時間も表示されています。ジョブの予想実行時間が計画停止までの時間より長い場合は、計画停止からの復旧後までスケジューリングされません。ジョブが実行を開始すると qstat の結果は実行中（RUNNING）になります。待機中、または実行中のジョブがない時は No requests. となります。

```
[p11000@yayoi-1 p11000]% qstat
2012/07/24 (Tue)   7:44:15:      REQUESTS on SR16000
NQS schedule stop time : 2012/07/27 (Fri)   9:00:00 (Remain:   73h 15m 45s)
    REQUEST        NAME        OWNER        QUEUE       PRI NODE E-TIME     MEM      STATE
No requests.
```

7.4.3 キューの状態確認

キューごと (システム全体) の実行ジョブ数、待ちジョブ数は以下のように qstat -b で確認できます。

```
[p11000@yayoi-1 p11000]%   qstat -b
2012/07/23 (Mon) 11:54:00:      REQUESTS on SR16000
NQS schedule stop time : 2012/07/27 (Fri)   9:00:00 (Remain:   93h   6m   0s)
QUEUE NAME        STATUS      TOTAL    RUNNING   RUNLIMIT   QUEUED   HELD      OTHER
P001              AVAILBL       9          2          2          7          0          0
P004              AVAILBL      17          4          4         13          0          0
P008              AVAILBL      23          2          2         21          0          0
debug             AVAILBL       1          0          4          1          0          0
```

RUNNING は実行中のジョブ数を示しています。例では、P001 では 2 本実行されていることが確認できます。RUNLIMIT は各キューで同時に実行できるジョブの数、QUEUED が実行 (順番) 待ちのジョブの数を表示しています (RUNLIMIT はシステム全体の計算資源の空き状況などにより適宜変更しています。そのため、RUNLIMIT 数よりも RUNNING 数が多くなっている場合もあります)。

7.4.4 ジョブのキャンセルまたは強制終了

投入したジョブをキャンセルしたい時、または実行中のジョブを強制終了したい時は qdel コマンドを使用します。

```
[p11000@yayoi-1 p11000]% qdel 39550.yayoi
```

7.4.5 ジョブ実行履歴の確認

投入したジョブの実行履歴等を確認するには、qlogf コマンドを使用します。オプションを指定せずに実行すると、当日のジョブ実行履歴が表示されます (オプションには、検索範囲指定やリクエスト ID 指定ができます)。

```
[p11000@yayoi-1 p11000]% qlogf -h
Usage: qlogf [-days] [-d date] [request-id ...]
          date must be 'dd/mm/yy' form.

[p11000@yayoi-1 p11000] qlogf
DATE          TIME        REQUEST       NAME      OWNER     QUEUE     EVENT
2012/07/23    11:16:38    39550.yayoi   STDIN     p11000    P004      QUEUED(priority:0)
2012/07/23    11:16:38    39550.yayoi   STDIN     p11000    P004      QUEUED(priority:63)
2012/07/23    11:58:13    39550.yayoi   STDIN     p11000    P004      STARTED
2012/07/23    11:58:19    39550.yayoi   STDIN     p11000    P004      ENDED(exit:0000)
        :
        :
```

7.4.6 ジョブチェイン機能

ジョブの実行終了時の状態により後続ジョブの実行または抑止を行うことができます。チェイン機能によるバッチジョブは、qstat コマンドの STATE 項目が CHAIN 表示となります。ただし、チェインジョブとして登録されたバッチジョブが実行条件を満たさなくなった場合は、STATE 項目が、NEVERRUN 表示に変更し、バッチジョブ実行は行われません。

指定方法 : qsub -c[EXIT コード] リクエスト ID

チェインジョブが設定された場合の表示例

```
[p11000@yayoi-1 p11000]% qstat
2012/07/20 (Fri) 10:50:33:      REQUESTS on SR16000
NQS schedule stop time : 2012/07/27 (Fri)   9:00:00 (Remain: 166h   9m 27s)
 REQUEST        NAME      OWNER      QUEUE      PRI   NODE   E-TIME    MEM     STATE
18619.yayoi     STDIN     p11000     P001       63      1     240m    170GB   QUEUED
18620.yayoi     STDIN     p11000     P001       62      1     240m    170GB   CHAIN
```

チェインジョブが設定された場合の表示例

```
[p11000@yayoi-1 p11000]% qstat
2012/07/20 (Fri) 10:51:33:      REQUESTS on SR16000
NQS schedule stop time : 2012/07/27 (Fri)   9:00:00 (Remain: 166h   8m 27s)
  REQUEST        NAME      OWNER      QUEUE      PRI   NODE   E-TIME    MEM     STATE
18620.yayoi     STDIN     p11000     P001       62      1     240m    170GB   NEVERRUN
```

7.4.7 ジョブの終了通知と結果確認

ジョブが終了するとジョブスクリプトの -mu オプションで指定したメールアドレス (省略時は SR16000 のアカウント) にメールが届きます。ジョブの出力は次のファイルに保存されています。

標準出力： スクリプトファイル名.oN （N はジョブ番号）
標準エラー出力： スクリプトファイル名.eN （N はジョブ番号）

7.4.8 システム障害時の動作

システム障害でジョブが途中で異常終了した場合などには、終了したジョブの名前とジョブ番号を個別にご連絡しています。お手数ですが再度実行をお願いいたします。ファイル状態などが再実行可能な状態かどうかセンターでは判断できませんので、自動的に再実行を行うことはいたしません。

7.5 ジョブスケジューリングのルール

実行待ちのジョブは「フェアシェアスケジューリング」と呼ばれる方式で実行順序が決まります。この方式では過去に使った計算資源が少ない利用者が優先されます。ジョブの投入順序と実行順は基本的には一致しません。

7.5.1 フェアシェアスケジューリングの詳細

フェアシェアスケジューリングシステムは、4 月から現在までに実行済みのバッチジョブにて使用した CPU の使用量 (以下、 CPU 使用量とする) と投入したジョブに設定した実行予定時間より算出される CPU の予定使用量 (以下、CPU 予定使用量とする) から各ジョブの実行優先度を決定しています。具体的なスケジューリングルールは次のようになっています。

- 各キュー毎に実行待ちのジョブにそれぞれ優先順位 (63 が最優先で 0 まで) を付け、順位の高いジョブから順に実行します。
- 順位付けの要素は、4 月から現在までに実行したバッチジョブの CPU 使用量を利用者毎に積算した値と、投入したバッチジョブで要求している CPU 予定使用量を合計したものであり、その値が少ないジョブから順位付けを行います。CPU 使用量とは「ジョブの実行時間×使用したノード数」です。
- いずれかのジョブが終了する毎に当該利用者の CPU 使用量を積算し、順位の付け替えを行います。大規模計算を行う利用者が不利とならないよう一定時間毎に CPU 使用量に一定の逓減率を掛け逓減する仕組みとしています。
- キューに並んでいる際の待ち時間は順位付けの要素に含みません。
- ジョブの実行は 2 つまで同時に可能です。ただし同一のキューでは 1 つとなります。
- ジョブの優先順位は qstat コマンドの "PRI" の値で確認が可能です。
  以下の例では、PRI が 50 となっているので、優先順位が 14 番目となっている (14 番目に実行 (13 本のジョブが実行待ち) される) ことがわかります。

```
[p11000@yayoi-1 p11000]% qstat
2012/07/24 (Tue) 15:48:42:      REQUESTS on SR16000
NQS schedule   stop time : 2012/07/27 (Fri)   9:00:00 (Remain:   65h 11m 18s)
    REQUEST      NAME       OWNER       QUEUE     PRI   NODE   E-TIME   MEM      STATE
76541.yayoi      job.001    p11000      P004       50     4      180m    170GB    QUEUED
76542.yayoi      job.002    p11000      P004       49     4      180m    170GB    QUEUED
76543.yayoi      job.003    p11000      P004       48     4      180m    170GB    QUEUED
```

参考：実行開始までの目安 (概算)

ジョブ実行開始までに係る時間の概算 (計算方法) は以下の通りです。

ジョブ実行待ちの状態を確認する。以下の例の場合、PRI が 63 となっているので、ジョブ実行優先順位は最上位となっていることが確認できます。

```
[p11000@yayoi-1 p11000]% qstat
2012/07/23 (Mon) 11:54:00:      REQUESTS on SR16000
NQS schedule stop time : 2012/07/27 (Fri)   9:00:00 (Remain:   93h   6m   0s)
    REQUEST      NAME        OWNER      QUEUE      PRI   NODE   E-TIME    MEM      STATE
18619.yayoi      STDIN       p11000     P001        63     1      240m     170GB   QUEUED
```

キューの同時実行数を確認する。以下の例の場合、投入したジョブクラスの同時実行数 (RUNLIMIT) は 1 本となっていることが確認できます (システム資源の空き状況などにより、適宜同時実行数 (RUNLIMIT)を変更しています。そのため同時実行数 (RUNLIMIT) よりも多く実行中 (RUNNING) のジョブがある場合があります)。

```
[p11000@yayoi-1 p11000]%   qstat -b
2012/07/23 (Mon) 11:54:00:      REQUESTS on SR16000
NQS schedule stop time : 2012/07/27 (Fri)   9:00:00 (Remain:   93h   6m   0s)
QUEUE NAME        STATUS      TOTAL    RUNNING   RUNLIMIT   QUEUED   HELD     OTHER
P001              AVAILBL       9          2          1          7          0          0
P004              AVAILBL      17          4          4         13          0          0
P008              AVAILBL      23          2          2         21          0          0
debug             AVAILBL       1          0          4          1          0          0
```

このような状態で実行待ちとなっている場合には、現在実行しているジョブが終了しだいジョブ実行が開始されることになります。例では P001 キューにジョブ実行待ちとなっていますので、最大 8 時間 (P001 キューの最大経過時間制限値) の実行待ち時間が発生することになります。

8

数値計算ライブラリ

8.1 概要

SR16000 には数値計算ライブラリとして MATRIX/MPP、MATRIX/MPP/SSS、MSL2、ESSL、Parallel ESSL、BLAS、LAPACK、ScaLAPACK、Parallel NetCDF、FFTW、SuperLU、SuperLU_DIST、STL(Standard Template Library)、Boost C++ があります。これらのライブラリを利用するにはコンパイラのオプションとして

-L ライブラリ検索パス名
-l ライブラリ名

を指定します。-l オプションはプログラムファイル名の後ろに指定します。このオプションは左から順に処理されるのでライブラリ名を指定する順序には注意して下さい。なお、センター提供の数値計算ライブラリの内、MATRIX/MPP、MATRIX/MPP/SSS、MSL2、ESSL、Parallel ESSL の検索パスは標準で設定されていますので、-L オプションは省略できます。

(注) 数値計算ライブラリ(ESSL/PESSL/BLAS/LAPACK/ScaLAPACK/FFTW/SuperLU/SuperLU_DIST/STL)において、数学関数を呼び出している場合、-lm オプションが必要になります。
(注) Fortran プログラムを取り扱う場合には -i,L オプションが必要となる場合があります。-i,L オプションは C プログラムでは必要ありません (ライブラリが C 言語で作成されている場合)。
(注) C プログラムを取り扱う場合には -lf90s オプションが必要となる場合があります。-lf90s オプションは Fortran プログラムでは必要ありません (ライブラリが Fortran 言語で作成されている場合)。

8.2 MATRIX/MPP、MATRIX/MPP/SSS

MATRIX/MPP は基本配列演算、連立 1 次方程式、逆行列、固有値・固有ベクトル、高速 Fourier 変換、擬似乱数等に関する副プログラムライブラリです。並列処理用インターフェースを用いることにより、データを各ノードに分散して配置、並列に実行することができます。MATRIX/MPP を使用する場合にはコンパイル時にオプションとして以下のライブラリを指定します。C プログラムから利用する場合は、-lf90s オプションも同時に指定して下さい（要素並列版は–parallel オプションも同時に指定して下さい）。

MATRIX/MPP (ver03-00)	要素並列版	-lmatmpp
	スカラー版	-lmatmpp_sc
MATRIX/MPP/SSS（スカイライン法、ver02-00）	要素並列版	-lmatmpps
	スカラー版	-lmatmpps_sc

(参考マニュアル)
「行列計算副プログラムライブラリ MATRIX/MPP」(3000-3-C56)
「行列計算副プログラムライブラリ 疎行列解法 MATRIX/MPP/SSS」(3000-3-C97)

8.3 MSL2

MSL2 は行列計算（連立 1 次方程式、逆行列、固有値・固有ベクトル等）、関数計算（非線形方程式、常微分方程式、数値積分等）、統計計算（分布関数、回帰分析、多変量解析等）に関する副プログラムライブラリです。MSL2 を使用するためにはコンパイル時にオプションとして以下のライブラリを指定します。C プログラムから利用する場合は、-lf90s オプションも同時に指定して下さい（要素並列版は–parallel オプションも同時に指定して下さい）。

MSL2 (ver02-00)	要素並列版	-lMSL2P
	スカラー版	-lMSL2

(参考マニュアル)
「数値計算副プログラムライブラリ MSL2 操作」(3000-3-C95)
「数値計算副プログラムライブラリ MSL2 行列計算」(3000-3-C92)
「数値計算副プログラムライブラリ MSL2 関数計算」(3000-3-C93)
「数値計算副プログラムライブラリ MSL2 統計計算」(3000-3-C94)

8.4 ESSL

ESSL(Engineering and Scientific Subroutine Library)は行列演算、連立 1 次方程式、固有値解析、Fourier 変換、乱数生成等に関する数値計算ライブラリです。BLAS、LAPACK の API も有しています。また、FFTW 3.1.2 のラッパーサンプルが /usr/lpp/essl.rte.common/FFTW3 にあります。ESSL を使用する場合には、リンク時にオプションとして以下を指定します。Fortran プログラムから利用する場合は、-i,L オプションも同時に指定して下さい (要素並列版は -parallel オプションも同時に指定して下さい)。

ESSL (ver5.1)	要素並列(32/64bit 共通、32bit 整数型対応)版	-lesslsmp
	スカラー(32/64bit 共通、32bit 整数型対応)版	-lessl

8.5 Parallel ESSL (PESSL)

PESSL は、MPI での分散メモリ並列処理による線形代数方程式、Sparse サブルーチン、ScaLAPACK のサブセット、Fourier 変換、一様乱数生成等に関する数値計算ライブラリです。PESSL を使用する場合には、リンク時にオプションとして以下を指定します。Fortran プログラムから利用する場合は、-i,L オプションも同時に指定して下さい。

PESSL (ver4.1)	MPI 版	-lpesslsmp -lesslsmp –lblacssmp

8.6 BLAS・LAPACK・ScaLAPACK

SR16000 の標準 BLAS、LAPACK、ScaLAPACK は本家 netlib より入手できるソースをコンパイルしたものとなります。これらは日立製作所製のコンパイラでコンパイルしたプログラムと正常にリンクできることが確認されております。性能を要求する場合は、ESSL、Parallel ESSL の利用をお奨めします。
システム標準の BLAS、LAPACK、ScaLAPACK を使用する場合にはコンパイル時にオプションとして以下を指定します。C プログラムから利用する場合は、-lf90s オプションも同時に指定して下さい(要素並列版は　-parallel オプションも同時に指定して下さい)。

	要素並列版	スカラー版
BLAS	-L/usr/local/lib -lblas	-L/usr/local/lib -lblas_sc
LAPACK (ver3.3.1)	-L/usr/local/lib -llapack -lblas	-L/usr/local/lib -llapack_sc -lblas_sc
ScaLAPACK (ver1.8.0)	-L/usr/local/lib -lscalapack -lblacs -llapack –lblas	無し

8.7 Parallel NetCDF

SR16000 には、アプリケーションに対し共通のデータアクセス方法を提供する I/O ライブラリ NetCDF(Network Common Data Form) ver1.1.1 がインストール(MPI 版のみ)されています。NetCDF を使用する場合には、リンク時にオプションとして以下を指定します。MPI 版のみの提供となりますので、ご注意ください。

Parallel NetCDF (ver1.1.1)	MPI 版(32/64bit 共通)	-I/usr/local/include -L/usr/local/lib -lpnetcdf

8.8 FFTW

FFTW(Fastest Fourier Transform in the West) は離散 Fourier 変換を計算するライブラリです。ver3.3 の MPI 版、要素並列版、スカラー版がインストールされています。FFTW を使用する場合には、リンク時にオプションとして以下を指定します。Fortran プログラムから利用する場合は、-i,L オプションも同時に指定して下さい（要素並列版は –parallel オプションも同時に指定して下さい)。

FFTW (ver3.3)	MPI 版 (32/64bit 共通)	-I/usr/local/include -L/usr/local/lib -lfftw3_mpi -lfftw3
	要素並列版 (32/64bit 共通)	-I/usr/local/include -L/usr/local/lib -lfftw3
	スカラー版 (32/64bit 共通)	-I/usr/local/include -L/usr/local/lib -lfftw3_sc

8.9 SuperLU、SuperLU_DIST

疎行列の直接ソルバである SuperLU がインストールされています。ver4.2 の要素並列版及びスカラー版、メッセージパッシング並列ライブラリの SuperLU_DIST (MPI 版、ver2.5)を用意しています。SuperLU を使用する場合には、リンク時にオプションとして以下を指定します（要素並列版は –parallel オプションも同時に指定して下さい)。

SuperLU (ver4.2)	要素並列版	-I/usr/local/include -L/usr/local/lib -lsuperlu -lblas
	スカラー版	-I/usr/local/include -L/usr/local/lib -lsuperlu_sc -lblas_sc
SuperLU_DIST (ver2.5)	MPI 版	-I/usr/local/include -L/usr/local/lib –lsuperludist -lparmetis -lmetis -lblas

8.10 C++ ライブラリ

C++ライブラリとして、日立最適化 C++ コンパイラにて標準ライブラリの一つである STL(Standard Template Library) を使用するためにはリンク時にオプションとして以下を指定します。

STL (ver4.6.2)	64bit	-I/opt/STLport-4.6.2/stlport -L/opt/STLport-4.6.2/lib64 -lstlport_sCC
	32bit	-I/opt/STLport-4.6.2/stlport -L/opt/STLport-4.6.2/lib -lstlport_sCC

また、Boost C++ ライブラリがインストールされています。リンク時にオプションとして以下を指定します。本ライブラリは日立最適化C++コンパイラには対応しておりません。IBM C++ コンパイラ、GNU C++ コンパイラをご利用ください。

Boost (ver1.47.0)	要素並列版	(ヘッダ検索パスの指定のみで利用可能なBoostライブラリを使用する場合 [例:Unordered Library など]) -I/usr/local/include (ライブラリ検索パスも指定する必要があるBoostライブラリを使用する場合 [例:Date-Time など]) -I/usr/local/include -L/usr/local/lib/boost -lboost_{ライブラリ名}

(注) Boost Accumulators ライブラリについては IBM C++ コンパイラが対応しておりません。GNU C++ コンパイラをお使いください。

(注) 使用する Boost ライブラリにより、-lpthread オプションが適宜必要になるケースがあります。

9

使用例

9.1 概要

本章ではこれまでの各章を踏まえ、SR16000 の使用例として、極めて簡単な例から一般的なパターンまでプログラムのコンパイルと実行方法を紹介いたします。

9.2 初歩的な例

本節はこれから初めて本センターの SR16000 にてスーパーコンピューターを使用しようとする方を対象にしています。しかしながら、Fortran 及び C 言語の文法や UNIX コマンド、ホスト名などの概念について解説するものでは無く、それらは既知として想定しておりますのでご注意ください。また、MPI 等についてセンターでは講習会を定期的に開催しておりますので、そちらの参加もご検討ください。
ここで、本節については MPI、OpenMP などのノード間、ノード内並列に詳しかったり、すでに本システムにて十分に計算を行われた経験があったりする方は読み飛ばしていただいて問題はありません。

9.2.1 プログラムの作成とコンパイル

プログラムソースの例を示し、日立最適化 FORTRAN90 コンパイラを用いてコンパイルしてみます。

まず、エディタである vi や emacs を用いて、以下のように Fortran90 でプログラムを作成し、ファイル名「gethostn.f90」として保存しておきます。

```
program gethostn
  integer statnum,hostnm
  character*12 name

  statnum = hostnm(name)

  write(*,*) 'HOSTNAME is "',name,'"(',statnum,')'
end
```

ここでは例として、実行する計算ノードのホスト名を取得し、それを標準出力するプログラムになっています。具体的には日立最適化 FORTRAN90 コンパイラで利用可能なサービスサブルーチンの一つである HOSTNM を用いて、実行時に name 変数にホスト名を格納し、ホスト名が取得できた場合に statnum に「0」を格納します。その後、write 文にてそれぞれの値を出力します。

日立最適化 FORTRAN90 コンパイラにおいて HOSTNM のようなコンパイラでサービスするサブルーチンについては日立より用意されているマニュアル「最適化 FORTRAN 言語」及び「最適化 FORTRAN 使用の手引」に記載があります。今回の場合、「最適化 FORTRAN 言語」の索引から先頭の文字 H の箇所を参照してください。マニュアルにはこの他にも各オプションの解説や、プログラミングに必要な注意書きなどが記載してあります。適宜参照するようにしてください。

それでは、このプログラムをコンパイルするために、以下のようなオプションを含むコマンドラインを実行してください。-lf90c により、サブルーチンの HOSTNM の呼び出しが可能になるからです。

```
% f90 gethostn.f90 -lf90c
f90: compile start : gethostn.f90

*OFORT90 V03-01-/B entered.
*program name = GETHOSTN
*end of compilation : GETHOSTN
*program units = 0001, no diagnostics generated.
```

このコマンドラインでは上記の様に特にエラーを出力せず、実行オブジェクトとして、a.out が生成されます。ここで、プログラムソース中の statnum = hostnm(name) の一行上に意図的に間違った文法となる statnum == 1 という行を追加してコンパイルすると次のようなエラーになります。

```
% f90 gethostn.f90 -lf90c
f90: compile start : gethostn.f90

*OFORT90 V03-01-/B entered.
    KCHF049K   12                5
                unrecognizable fortran statement.
*program name = GETHOSTN
*program units = 0001, 0001    diagnostics generated, highest severity code is    12
```

この KCHF の一行が日立最適化 FORTRAN90 コンパイラにおける診断メッセージのコードを表しており、その一行下が診断メッセージ本文となります。今回の場合はコード:KCHF049K のエラー(Fortran で分類できない文がある)が発生しており、エラーレベルは 12(文法の誤りがあり、オブジェクトモジュールは生成しない)でエラー個所はソースコードの 5 行目ということを表してします。もし、診断メッセージだけでは不十分であった場合はデバッグオプションである -debug をコンパイルオプションに追加して試してみるとより詳細な情報が得られることがあります。通常はこのメッセージを参考にプログラム作成を行ったり、本センターの SR16000 の環境で正常に動作するよう、手持ちのプログラムのデバッグ作業を実施します。

また、この診断メッセージのコードを元にマニュアルの関連する記述を参照することができます。上記の例の場合は「最適化 FORTRAN 使用の手引」の「11 章コンパイル時の診断メッセージ(11.1 診断メッセージ)」の KCHF049K の項を参照してください。

次に、コンパイル可能な状態のプログラムソースを使って、試しにオプション「-lf90c」を外してコンパイルしてみます。

```
% f90 gethostn.f90
f90: compile start : gethostn.f90

*OFORT90 V03-01-/B entered.
*program name = GETHOSTN
*end of compilation : GETHOSTN
*program units = 0001, no diagnostics generated.
ld: 0711-317 ERROR: Undefined symbol: .HOSTNM
ld: 0711-345 Use the -bloadmap or -bnoquiet option to obtain more information.
find: bad status-- /tmp/.20120806165543720948F90
```

先ほどとは異なるエラーが出ました。ld で Undefined symbol なので、HOSTNM が解釈できていないことになります。HOSTNM はプログラムソース中の hostnm と異なり、大文字と小文字の差がありますが、今回のサービスサブルーチンの場合、適宜コンパイラが解釈してくれるので、問題となる箇所ではありません。サブルーチン名の大文字と小文字を解釈してくれるオプション「-i,L」を用いると次のようになります。このオプションは今後外部のライブラリを用いる場合には必要なケースがあります。

```
% f90 gethostn.f90 -i,L
f90: compile start : gethostn.f90

*OFORT90 V03-01-/B entered.
*program name = gethostn
*end of compilation : gethostn
*program units = 0001, no diagnostics generated.
ld: 0711-317 ERROR: Undefined symbol: .hostnm
ld: 0711-345 Use the -bloadmap or -bnoquiet option to obtain more information.
find: bad status-- /tmp/.20120806171837537 4832F90
```

以上より、このようなすでにコンパイル可能な例、あるいは手持ちのプログラムを用いて、オプションを変更したり、プログラムの記述をわざと変更したりすることで、本システムではどの様なエラーメッセージが出力されるかを把握しておくことは今後のプログラムのデバッグ作業にも有用となります。

9.2.2 ジョブ実行

9.2.1 でコンパイル、生成された a.out を、ジョブスクリプトを記述して、計算ノードで実行してみます。その際、ジョブスクリプトのパラメータを少し変更して出力の変化を見てみます。

ジョブスクリプト(job.csh)は以下のようにします。parallel キュー(-q parallel)において 1 ノード(-N 1)を用い、1 プロセス/1 ノード(-J T01,T は task の「t」)で実行してみます。$QSUB_WORKDIR は qsub コマンドを実行した際のカレントディレクトリが代入されます。今回の場合、job.csh と a.out は同じディレクトリに保存されているとします。よって、このジョブスクリプトは、計算ノードにおいて $QSUB_WORKDIR にカレントディレクトリを変更し、$QSUB_WORKDIR に置かれている a.out を実行するというジョブスクリプトになります。

```
#!/bin/csh
#@$-q parallel
#@$-N 1
#@$-J T01

cd $QSUB_WORKDIR
./a.out
```

qsub job.csh にてジョブ実行を開始し、問題なく終了すると job.csh.oXXXXX (XXXXX は数字)というテキストファイルが生成されるので、それを参照してみます。以下のような 1 レコードの出力になっているはずです。

```
HOSTNAME is "htcfWWcWWpWW"(              0 )
```

「htcfWWcWWpWW」が取得された計算ノードのホスト名です。a.out はこの計算ノード上で実行されました。次に 2 ノード(-N 2)を用いて実行してみます。

```
#!/bin/csh
#@$-q parallel
#@$-N 2
#@$-J T01

cd $QSUB_WORKDIR
./a.out
```

出力されたファイルには先ほどと同様に 1 レコードしか得られていないはずです。これは 1 ノード(-N 1)を用い、2 プロセス/1 ノード(-J T02)で実行しても同様です。また、下記のように a.out を 2 つ実行するようにしても、-N や -J を変化させても同じホスト名で出力されたレコードが 2 つ出力されます。

```
#!/bin/csh
#@$-q parallel
#@$-N 2
#@$-J T01

cd $QSUB_WORKDIR
./a.out &
./a.out &

wait
```

以上より、単純に a.out を実行する限りでは NQS のパラメータを操作しても、ある 1 ノード上でしか実行されないことが分かります。ここで、本センターのシステムは、1 つの a.out を、複数の計算ノードを跨いだすべての CPU を使って実行する仕組みではありません。これは例えば OpenMP でノード内並列(スレッド並列)されたプログラムが生成するスレッドにおいても同様です。逆に並列化されていなかったり、スレッド並列化されていても 1 ノード使えれば十分なプログラムの場合は、これまでの 1 ノード、1 プロセス実行で基本的な対応ができます。次に、NQS オプションで計算ノード数個分、プロセス数個分の a.out を実行するためには mpirun コマンドを使います(MPI 実行)。

```
#!/bin/csh
#@$-q parallel
#@$-N 2
#@$-J T01

cd $QSUB_WORKDIR
mpirun ./a.out
```

この場合は以下の出力が得られるはずです。各計算ノードで a.out が実行できました。

```
HOSTNAME is "htcfWWcWWpWW"(              0 )
HOSTNAME is "htcfXXcXXpXX"(              0 )
```

また、 1 ノード(-N 1)を用い、2 プロセス/1 ノード(-J T02)で実行すると a.out をバックグラウンドで 2 つ記述して実行せずとも以下の出力が得られます。

```
HOSTNAME is "htcfWWcWWpWW"(              0 )
HOSTNAME is "htcfWWcWWpWW"(              0 )
```

このように mpirun を使うと、プログラムを並列に実行できますが、単純に同じ a.out を計算ノード数分あるいはプロセス個数分実行しただけで、プログラム自体が MPI で並列化されたわけではありません。一つのプログラムを効率よく複数計算ノードあるいは複数プロセスを使って実行するためには MPI を用いて適切にプログラムソースを書き換える必要があります。

9.3 OpenMP

9.3.1 OpenMP プログラムとコンパイル

OpenMP は、共有メモリを用いた並列化を記述するための指示文、ライブラリ、及び環境変数を規定したもので、並列実行単位にスレッドを用います。各スレッドが各 CPU に分散され並列実行されます。ここでは以下のようなサンプルプログラムを作成し、ファイル名「omp_sample.f90」として保存しておきます。

OpenMP についてはマニュアル「最適化 FORTRAN 使用の手引」の「7. OpenMP」に詳細な記述があります。コマンドなどについては FAQ 「OpenMP は使えますか。使い方を教えてください。」にまとめてありますので、ご覧ください。

```
program omp_sample
   implicit none
   include 'omp_lib.h'

   double precision t1, t2, diff
   integer, parameter :: N = 1000
   integer i, j, k, nt
   double precision :: a(N,N), b(N,N), c(N,N)
   double precision tmp_val

   t1 = 0.0d0
   t2 = 0.0d0

!$omp parallel
!$omp master
   nt = omp_get_num_threads()
!$omp end master
!$omp end parallel

   do i=1, N
      do j=1, N
         a(j,i) = dble(i+j)
         b(j,i) = dble(i+j)
         c(j,i) = 0.0d0
      end do
   end do

   t1 = omp_get_wtime()
!$omp parallel
!$omp do private(j,k,tmp_val)
   do i=1, N
      do j=1, N
         tmp_val = 0.0d0
         do k=1, N
            tmp_val = tmp_val + a(k,i) * b(j,k)
         end do
         c(j,i) = tmp_val
      end do
   end do
!$omp end do
!$omp end parallel
   t2 = omp_get_wtime()

   diff = t2 - t1
   print *, nt, 'thread(s), Etime is:', diff, '[s]'
end program omp_sample
```

このプログラムは適当な行列積を実行し、その計算にかかった経過時間を表示するものです。実行時のスレッド数を取得する関数(omp_get_num_threads)と経過時間を取得する関数(omp_get_wtime)を使用するため、まず omp_lib.h をインクルードします。OpenMP 実行時ライブラリのヘッダファイルである omp_lib.h に対しては標準でライブラリパスが通っているので、omp_lib.h をフルパスなどで記述する必要はありません。

コンパイルには、日立最適化 FORTRAN90 コンパイラで OpenMP ディレクティブを有効にするオプション -omp を使います。

```
% f90 -omp -Oss omp_sample.f90
f90: compile start : ./omp_sample.f90

*OFORT90 V03-01-/B entered.
    KCHF475K   00                    C
                the variable is defined, but is never referred.
*program name = OMP_SAMPLE
*end of compilation : OMP_SAMPLE
*end of compilation : _user_parallel_func_1__hf_main
*end of compilation : _user_parallel_func_2__hf_main
*program units = 0001, 0001   diagnostics generated, highest severity code is   00
```

行列積の結果を保存した配列 c の出力などを行っていないため、KCHF475K が出力されます。これはエラーではなく警告メッセージ相当の出力であり、コンパイルは中断せず、オブジェクトファイルは生成されます。詳細は「最適化 FORTRAN 使用の手引」を参照してください。

-Oss で最適化しない場合は -Oss に含まれていた -parallel を明記します。

```
% f90 -omp -parallel omp_sample.f90
```

9.3.2 ジョブ実行

以下のようなジョブスクリプト(job.csh)を記述します。parallel キュー(-q parallel)において 1 ノード(-N 1)を用い、1 プロセス/1 ノード(-J T01)で実行します。

```
#!/bin/csh
#@$-q   parallel
#@$-N   1
#@$-J   T01

setenv OMP_NUM_THREADS 8

cd $QSUB_WORKDIR
./a.out
```

OpenMP を使用したプログラムを実行する際のスレッド数は環境変数の $OMP_NUM_THREADS に与えます。上記では 1 プロセス立ち上げたプログラムから 8 つのスレッドを生成し、並列実行するものです。出力ファイル(job.csh.oXXXXX)を確認します。

```
 8 thread(s), Etime is:   0.412459999999999938        [s]
```

$OMP_NUM_THREADS を 1 から 64 まで変化させてそれぞれ実行し、結果を比較してみてください。ここで、基本的に性能面での理由から、指定する $OMP_NUM_THREADS の値が 32(CPU コア数) x SMT(=2) = 64 を超えないように注意してください。

9.4 MPI

9.4.1 簡単な MPI プログラムとコンパイル

MPI(Message Passing Interface) は MPI Forum によって標準化されたメッセージ通信ライブラリのインタフェース規約です。ノード間及びノード内のプロセス間通信に MPI 通信ライブラリを持ちいたメッセージ通信が可能です。MPI は多くの計算機に実装されており、移植性の高いインタフェースとなります。

本センターの SR16000 では MPI 2.1 をサポートしています。ここでは以下のようなサンプルプログラムを作成し、ファイル名「mpi_sample.f90」として保存しておきます。コマンドなどの詳細については、FAQ 「MPI プログラムをコンパイルする際のコマンド名について教えてください。」をご覧ください。

```
program getmpirank
   implicit none
   include 'mpif.h'

   integer :: size,rank,ierr

   call MPI_INIT(ierr)
   call MPI_COMM_RANK(MPI_COMM_WORLD,rank,ierr)
   call MPI_COMM_SIZE(MPI_COMM_WORLD,size,ierr)

   write(*,*) 'proc=',rank, 'size=',size

   call MPI_FINALIZE(ierr)
end program getmpirank
```

MPI 関数を使用するため、まず mpif.h をインクルードします。MPI ライブラリのヘッダファイルである mpif.h に対しては標準でライブラリパスが通っているので、mpif.h をフルパスなどで記述する必要はありません。上記では、MPI 実行環境の初期化(MPI_INIT)と実行環境の終了(MPI_FINALIZE)、MPI コミュニケータから、call したプロセスのランクの取得(MPI_COMM_RANK)と全プロセス数の取得(MPI_COMM_SIZE)というプログラム制御関連の MPI 関数のみ用いており、通信を行う関数(MPI_Send など)を使用していない単純な例となっています。また、MPI コミュニケータには全プロセスで構成される MPI_COMM_WORLD を指定しています。取得できるランク番号は 0 から始まる点ご注意ください。

コンパイルには、MPI を使用するにあたって必要なヘッダファイルやライブラリ、コンパイルオプションを設定し、f90 コンパイラを呼び出す mpif90 コマンドを使います。今回は以下のように -Oss で最適化するものの、このオプションには自動並列化オプションが含まれるため、自動並列化を -noparallel オプションで無効にします。ピュア MPI なので、各プロセスは単一スレッドでの動作となります。

```
% mpif90 -Oss -noparallel mpi_sample.f90
f90: compile start : mpi_sample.f90

*OFORT90 V03-01-/B entered.
*program name = GETMPIRANK
*end of compilation : GETMPIRANK
*program units = 0001, no diagnostics generated.
```

9.4.2 ジョブ実行(ピュア MPI)

以下のようなジョブスクリプト(job.csh)を記述します。parallel キュー(-q parallel)において 2 ノード(-N 2)を用い、32 プロセス/1 ノード(-J T32)で実行します。

```
#!/bin/csh
#@$-q   parallel
#@$-N   2
#@$-J   T32

cd $QSUB_WORKDIR
mpirun ./a.out
```

合計 64 プロセスでの実行なので、出力されるランク番号は 0 から 63 の 64 個になっています。出力ファイル(job.csh.oXXXXX)を確認します。

```
proc=              0 size=             64
 proc=             1 size=             64
....
 proc=            62 size=             64
 proc=            63 size=             64
```

ここで、指定するプロセス数(-J)には 32(CPU コア数) x SMT(=2) = 64 を超えた指定を行うことはできません。

9.4.3 OpenMP + MPI ハイブリッド実行

OpenMP + MPI ハイブリッド実行についてはコンパイル方法とジョブスクリプトの例を示すに留めることにします。プログラムソース(hybrid_sample.f90)には OpenMP ディレクティブが書かれているとすると、次のようにしてコンパイルします。なお、コマンドなどについては、FAQ 「MPI 並列化したプログラムですが、ノード内を OpenMP で並列化していますが利用可能でしょうか。また、コンパイル方法について教えてください。」にもまとめてありますので、ご覧ください。

```
% mpif90 -omp -Oss hybrid_sample.f90
```

-Oss で最適化しない場合は -Oss に含まれていた -parallel を明記します。

```
% mpif90 -omp -parallel hybrid_sample.f90
```

このプログラムを 1 ノードあたり 8 プロセス、 1 プロセスあたり 4 スレッド、8 ノードで実行する際のジョブスクリプトは以下のようになります。

```
#!/bin/csh
#@$-q   parallel
#@$-N   8
#@$-J   T08

setenv OMP_NUM_THREADS 4

cd $QSUB_WORKDIR
mpirun ./a.out
```

parallel キュー(-q parallel)において 8 ノード(-N 8)を用い、8 プロセス/1 ノード(-J T08)での実行となります。また、各プロセスは OpenMP のスレッド 4 本で並列化するので、環境変数 OMP_NUM_THREADS に 4 を設定します。

ここで、プログラムの作りにもよりますが、基本的に性能面での理由から、指定するプロセス数(-J) x OMP_NUM_THREADS の値が 32(CPU コア数) x SMT(=2) = 64 を超えないように注意してください。プログラムの作り次第で計算ノード上のメモリを使い果たしてしまい、一部プロセスがダウンしてしまうなどの原因になります。

9.5 より高度な実行

立ち上げるプロセスやスレッドを適切な CPU に割り付け、かつ、物理的に最適なメモリを使うようにするにはどのようにすればよいかや SMT についてなどが日立より公開されているマニュアル「大規模 SMP 並列 スーパーコンピューターシステム ジョブ実行方法」や「SR16000 性能を引き出す利用方法」に書かれていますので、こちらを参照の上ご活用ください。

10

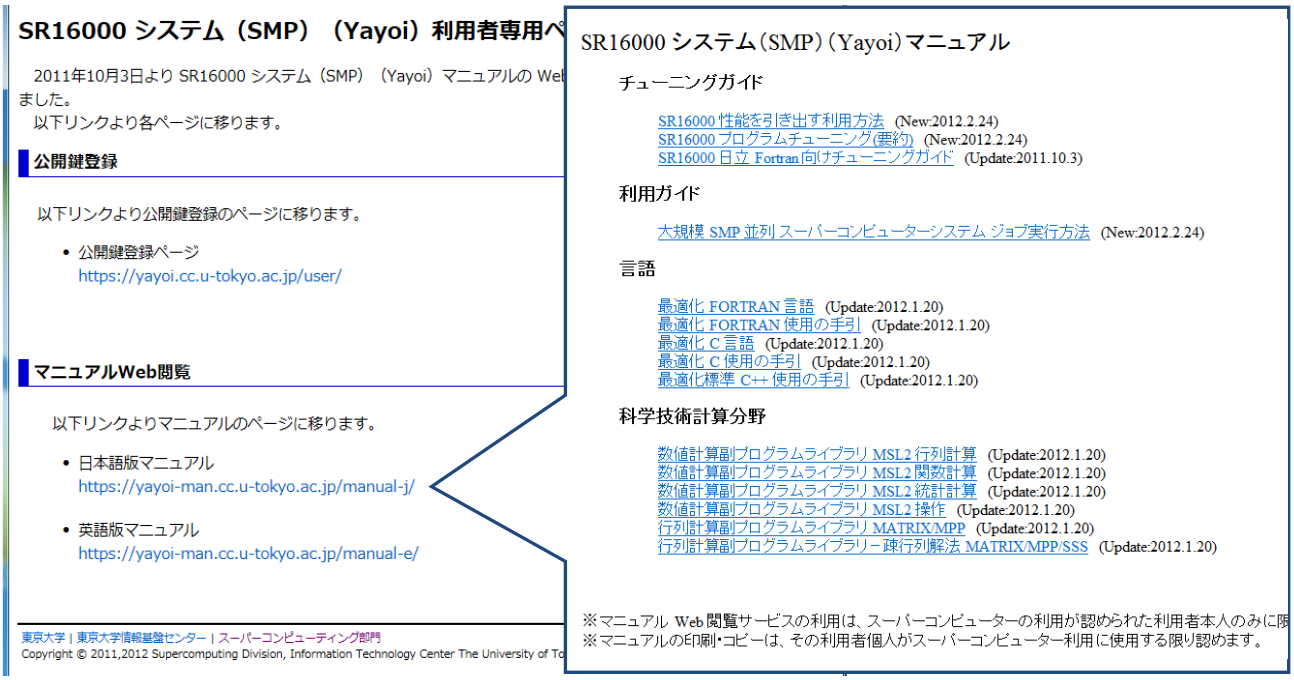
マニュアルの閲覧

10.1 Web 経由でのマニュアルの閲覧

本システムに導入されているコンパイラと数値計算ライブラリのマニュアルは Web 経由での閲覧が可能です。ただし本システムのアカウントが必要です。マニュアルは次のページよりアクセスできます。

https://yayoi-man.cc.u-tokyo.ac.jp/

この URL にアクセスするとユーザー名とパスワードを求められますので、センター発行のパスワードを入力してください（ログインに使用している鍵のパスフレーズではありません）。このページからは以下のマニュアルが HTML および PDF 形式で閲覧できます。また、AIX のマニュアルについては IBM 社の Web ページで参照できます。



チューニングガイド

- SR16000 性能を引き出す利用方法
- SR16000 プログラムチューニング(要約)
- SR16000 日立 Fortran 向けチューニングガイド

利用ガイド

- 大規模 SMP 並列 スーパーコンピューターシステム ジョブ実行方法

言語

- 最適化 FORTRAN 言語
- 最適化 FORTRAN 使用の手引
- 最適化 C 言語
- 最適化 C 使用の手引
- 最適化 C++ 使用の手引

科学技術計算分野

- 数値計算副プログラムライブラリ　MSL2 行列計算
- 数値計算副プログラムライブラリ　MSL2 関数計算
- 数値計算副プログラムライブラリ　MSL2 統計計算
- 数値計算副プログラムライブラリ　MSL2 操作
- 数値計算副プログラムライブラリ　MATRIX/MPP
- 数値計算副プログラムライブラリー疎行列解法 MATRIX/MPP/SSS

付録 A

Gaussian16 の使い方

A.1 Gaussian16

一般的非経験分子軌道計算プログラム Gaussian16 を SR16000 で公開しました。Gaussian16 を実行するためのジョブの作成と使用方法について簡単に説明します。但し、Gaussian16 自体の内容や計算方法、入力ファイルの書き方に関しては触れませんので詳細は以下に示す Gaussian 社の Web ページを参照して下さい。

本センターにおける Gaussian16 のバージョンは次の通りです。（2017 年 4 月 1 日現在）

Gaussian16 Revision A.03

Gaussian は 1 ノードによる逐次実行版(並列版ではありません)ですが、プログラムは要素並列化されており、64bit モードで動作します。また、ソースプログラムは非公開です。

（参考文献）

Gaussian Inc.（ **http://gaussian.com/** ）
Technical Support（ **http://gaussian.com/techsupport/** ）
Gaussian16 Users Reference（ **http://gaussian.com/man/** ）

Gaussian16 を使用して得られた計算結果や成果を公表する場合には Gaussian 社の Web ページに従って文献の引用を行うようにして下さい。

A.2 スクリプトファイルの作成

Gaussian16 はバッチジョブ（NQS）で実行する必要があります。バッチジョブで実行するには環境変数の設定や入出力ファイルを記述するスクリプトファイルを作成する必要があります(ログインノードでは実行できません)。以下にスクリプトファイルの例を挙げ説明します。

記入例	説明
#!/bin/csh	(1) シェルの宣言
#@$-q parallel	(2) キューの選択
#@$-N 1	(3) ノード数
#@$-lT 00:30:00	(4) 経過制限時間（ここでは 30 分）
#@$-ls 1536MB	(5) スタックサイズ
source /usr/local/g16/bsd/g16.login	(6) 環境設定
cd gaussian/test	(7) 入力ファイルの場所
g16 < test0178.com > test0178.log	(8) プログラムの起動

(1) スクリプトファイルを C シェル「/bin/csh」で記述することを宣言します。（本スクリプトでは source 等 C シェルのコマンドを使用しています。）
(2) 選択するキューは「parallel」あるいは「debug」となります。
(3) 並列版ではないためノード数は「1」ノード固定です。
(4) ジョブの実行時間を制限します。通常は省略可能です。（計画停止時刻（qstat コマンドで確認）が迫っているときはこれを残り時間以内に設定しないとジョブが実行しません。）
(5) スタックサイズの標準は 1GB，最大は 2GB です。標準で足りない場合は 2GB 以内の値を指定してください。
(6) コマンドパスなど各種環境変数を設定します。
(7) 入力ファイルのある場所までディレクトリを移動します。
(8) プログラム g16 を起動します。入力ファイル及び出力ファイルを以下のようにリダイレクションで指定することができます。
　　g16 < 入力ファイル > 出力ファイル
「> 出力ファイル」を省略すると実行結果はジョブの標準出力ファイルに出力します。

A.3 プログラムの実行

スクリプトファイルが完成したらファイル（以下の例では job.csh）に保存し，バッチシステムにジョブを投入します。ジョブの投入は以下のように qsub コマンドで行います。実行状況は qstat コマンドで確認して下さい。

```
[p11000@yayoi-1 p110000]% qsub   job.csh
Request 15086.yayoi submitted to queue: P001.

[p11000@yayoi-1 p110000]% qstat
  2011/11/27 (Sun) 20:21:50:      REQUESTS on SR16000
  NQS schedule stop time : 2011/12/23 (Fri)   9:00:00 (Remain: 612h 38m 10s)
     REQUEST      NAME      OWNER       QUEUE      PRI   NODE   E-TIME    MEM       STATE
  15086.yayoi      job.sh     p11000      P001       63      1     1800s     170GB     QUEUED
```

A.2 のスクリプトファイルの場合，ジョブが終了すると以下のファイルが作成されます。エラーメッセージが出ていないか各ファイルを確認して下さい。

```
job.csh.o15086      ← ジョブの標準出力
job.csh.e15086      ← ジョブの標準エラー出力（空の場合は作成されません）
test0178.log        ← Gaussian16 の出力ファイル
```

エラーがなければ出力ファイルに計算結果が出力され，標準出力のファイルは空になります。

ジョブをキャンセルする場合には以下のようにします。

```
[p11000@yayoi-1 p110000]% qdel 15086
deleting request 15086.yayoi.
```

A.4 実行時のエラー

Gaussian16 実行後は，出力ファイル（Gaussian16 の出力ファイル，ジョブの標準・エラー出力ファイル）のメッセージを確認して下さい。なお，問題解決には，入力ファイル中のルートセクションの記述を #（標準）や#T（簡略化した出力）ではなく#P（詳細の出力）としてみることも必要です。

A.5 テスト用入力ファイル

Gaussian16 が用意しているテスト用の入力ファイルが以下にあります。

/usr/local/g16/tests/com/

A.6 注意事項

センターでは、Gaussian16 で利用できるコア数を 1 としています (標準値)。必要に応じて使用するコア数 (プロセッサ数) を変更してください。ただし、Gaussian プログラム、問題サイズ等によっては、コア数を変更することで、実行時間が長くなる場合もありますのでご注意ください。

入力ファイルに %Nproc=4 と指定した場合、log ファイルに以下のメッセージが出力される
(4 プロセッサが使用された)

```
 ******************************************
 Gaussian 16:   IBM64-G16RevA.03 25-Dec-2016
                10-Mar-2017
 ******************************************
 %Nproc=4
 Will use up to      4 processors via shared memory.
```

入力ファイルに %cpu=33,35,37,39 と指定した場合、log ファイルに以下のメッセージが出力される
(使用するコア番号を指定して実行)

```
 ******************************************
 Gaussian 16:   IBM64-G16 RevA.03 25-Dec-2016
                10-Mar-2017
 ******************************************
 %cpu=33,35,37,39
 SetSPE:   set environment variable "XLSMPOPTS" = "procs=33,35,37,39"
 Will use up to      4 processors via shared memory.
```