T2K Open Supercomputer

HA8000 クラスタシステム
利用の手引き

第２版

全国共同利用施設
東京大学情報基盤センター

- 本利用の手引は 2009 年 3 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き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ホーム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

本利用の手引は 2008 年 6 月より運用を開始した「HA8000 クラスタシステム」の利用方法を解説するものです。本システムは筑波大学、東京大学、京都大学の 3 大学が共同で仕様を策定して調達を行った「T2K オープンスーパーコンピュータ」の東京大学版となります。

T2K オープンスーパーコンピュータは広く使われているハードウェアおよびソフトウェアを中心に構成されていることを特徴としており、使用開始にあたってのハードルが従来のスーパーコンピュータより低くなるように設計されています。しかしながらスーパーコンピュータを初めて利用する方には大型並列計算機ならではの使用手順に慣れていただく必要があり、また熟練した利用者の方にも本センター固有の情報を知っていただく必要があります。本利用の手引は HA8000 クラスタシステムのすべての利用者のために、本システム固有の機能を中心にシステムの使い方全般について解説するものです。

「利用の手引き」は試験運用開始前に第一版を発行しましたが、試験運用開始から 8 ヶ月の間に当初導入されていなかったソフトウェアが導入されたり、利用者の皆様から多くの質問をいただく事柄が記述されていないなどの問題が出てきました。そこでこれらの問題を修正すべく 2009 年度を迎えるにあたって大幅に加筆修正した第二版を発行することと致しました。本利用の手引が HA8000 クラスタシステムを使用するにあたっての参考となり、有益な計算結果を得るための手助けとなれば幸いです。

2009 年 3 月
東京大学情報基盤センター
「利用の手引き」制作代表　　松葉　浩也

予備知識

本利用の手引は、本センターの HA8000 クラスタシステム固有の機能や設定を中心に解説します。内容は読者の方々が 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 Linux などの参考書もご用意いただくと本システムを使用するのに役立ちます。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

1

システム概要

1.1 HA8000 クラスタシステムについて

本センターでは

- HITACHI SR11000/J2
- HA8000 クラスタシステム（T2K オープンスーパーコンピュータ）

の２式のスーパーコンピュータシステムをサービスしています。この利用の手引は「HA8000 クラスタシステム」について解説します。

HA8000 クラスタシステムは「T2K オープンスーパーコンピュータ」として筑波大学、京都大学と共同で仕様を策定したスーパーコンピュータです。仕様を共同で策定したため 3 大学の計算センターに構成の似たスーパーコンピュータが導入されていますが、本センターでは特に東京大学情報基盤センターに導入された T2K オープンスーパーコンピュータを意味する名称として「HA8000 クラスタシステム」を用います。この名称は本センターの T2K オープンスーパーコンピュータの各ノードである計算機の製品名である「HITACHI HA8000-tc/RS425」の名前に由来します。

1.2 HA8000 クラスタシステムの特徴

従来のスーパーコンピュータは「ベクトル化」に代表される特殊な最適化が必要でプログラムの汎用性が低くなる傾向がありましたが、HA8000 クラスタシステムは多くの人に手軽に使っていただける環境を提供するため特殊な技術を使わない設計となっています。具体的には特徴は次の 3 点です。

- パーソナルコンピュータやクラスタシステムなどで広く使われている x86 系プロセッサを搭載しているため、特殊なソフトウェアや最適化が不要
- Linux オペレーティングシステムを採用しているため、一般的な PC クラスタとほぼ同じ操作で使用可能
- 多数のプロセッサと高性能ノード間接続による高性能計算が可能

つまり、オープンスーパーコンピュータは、PC クラスタの利点を備えつつプロセッサ数と通信性能を強化した並列計算機ということになります。

1.3 ハードウェア構成

本章では HA8000 クラスタシステムのハードウェア構成を紹介します。

1.3.1 全体構成

HA8000 クラスタシステムは 16 個のプロセッサを持つ計算機 952 台を高速ネットワークで接続したクラスタ型並列計算機です。全体構成は下の表のようになっています。

表 1　HA8000 クラスタシステム全体性能

	項目	仕様
全体性能	理論演算性能	140.1344TFLOPS
	主記憶容量	31.25TB
	ノード数	952（512 ＋ 128 ＋ 256 ＋ 56）
	ノード間ネットワーク性能	5 GB/s ×双方向（計算ノード群 A） 2.5 GB/s ×双方向（計算ノード群 B）
	ストレージ容量	1PB (RAID6: 実効容量)

運用上、一個のアプリケーションが 952 台全体を使用することはないので、図 1 のように 952 台は 512 台、128 台、256 台、56 台に分割されて接続されています。また、512 台のクラスタと 128 台のクラスタはノード間が 5GB/s で接続されており、256 台のクラスタと 56 台のクラスタはノード間が 2.5GB/s で接続されています。5GB/s 接続のクラスタをタイプ A、2.5GB/s のクラスタをタイプ B と呼びます。

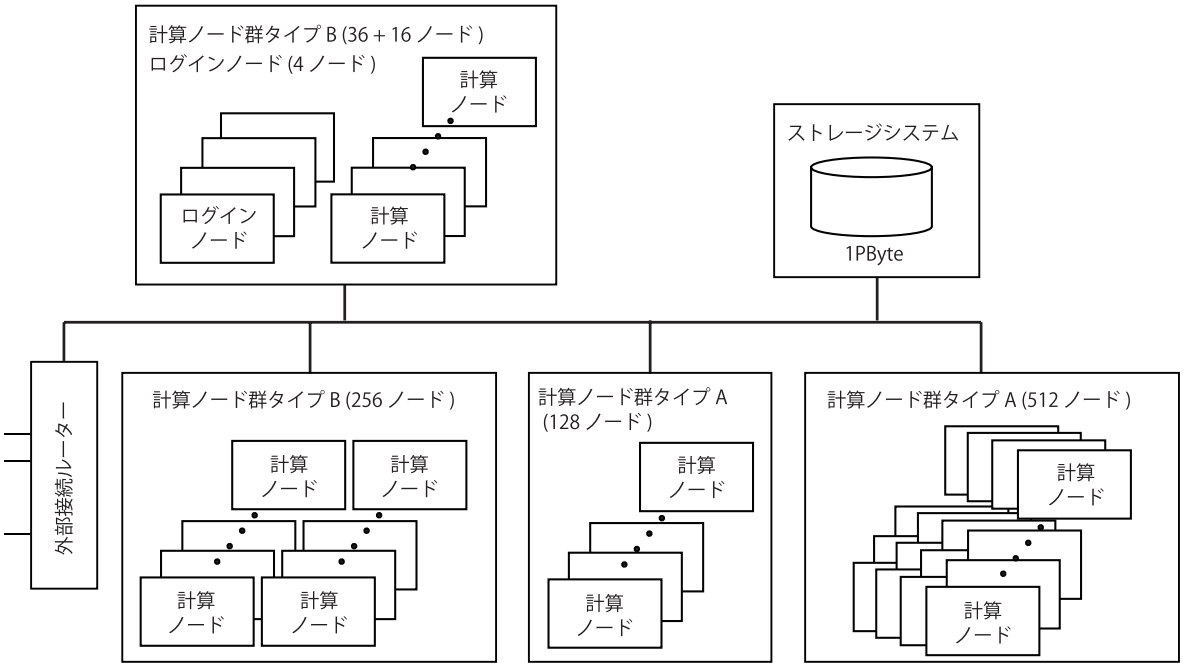

図 1　HA8000 クラスタシステム全体構成

このように HA8000 クラスタシステムは 4 組のクラスタとストレージシステム、外部接続ルーターがネットワークで接続された構成になっています。クラスタ間およびクラスタとストレージシステムの間は 1.25GB/s から 7.5GB/s のネットワークで接続されています。

1.3.2 クラスタ内の構成

それぞれのクラスタは AMD Quad Core Opteron プロセッサを 4 個（合計 16 コア）搭載したノードを Myrinet-10G インターコネクトで接続したものとなっています。Myrinet-10G は 1 リンク 1 方向あたり 1.25GB/s のバンド幅を持ち、双方向同時通信が可能です。タイプ A のクラスタは 1 ノードあたり 4 本の Myrinet-10G を搭載しているので 5GB/s、タイプ B は 2 本なので 2.5GB/s となります。各クラスタ内では Myrinet-10G は「フルバイセクションバンド幅」が確保される方法で接続されています。フルバイセクションバンド幅の構成とはクラスタ内の任意の半数のノードが同時に残り半分のノードにデータを送信してもネットワーク内での競合が発生しないネットワーク構成を意味します。図 2 に 16 ポートのスイッチを 24 個使用して 128 ノードのフルバイセクションネットワークを構成する例を挙げます。実際の HA8000 クラスタシステムでは 32 ポートのチップを多数搭載した 512 ポートスイッチを使用しているため、図 2 は実際の配線を示しているわけではありませんが、方式としては実際のシステムと同じです。多くの接続でバンド幅が確保されていることがおわかりいただけるかと思います。

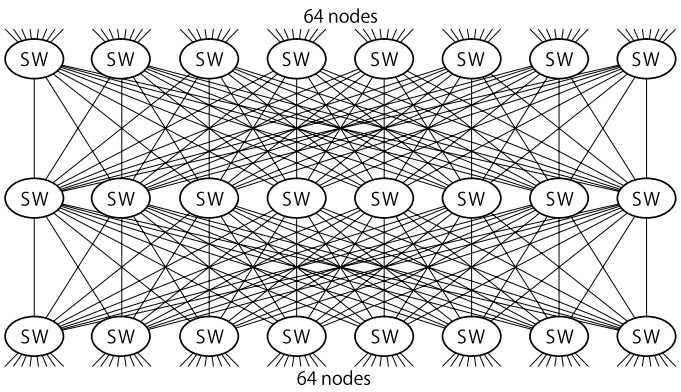

図 2　ネットワーク構成

1.3.3 ノードの構成

HA8000 クラスタシステムの各ノードはクアッドコアプロセッサ 4 個、つまり 16 コアの CPU を搭載しています。表 2 にノードの主な仕様を示します。AMD Quad Core Opteron 8356 プロセッサは 1 クロックに 4 回の浮動小数点演算ができるので、2.3GHz の場合、1 コアあたりの演算性能は 2.3 x 4 = 9.2GFlops です。各ノードは 16 個のプロセッサコアを搭載しているため、演算性能は 9.2 x 16 = 147.2GFlops となります。

表 2　HA8000 ノードの諸元

ノード	理論演算性能	147.2GFLOPS
	プロセッサ数（コア数）	4 (16)
	主記憶容量	32GB (936 ノード) 128GB (16 ノード)
	ローカルディスク容量	250GB (RAID1 OS 領域含む)
プロセッサ	プロセッサ（周波数）	AMD Quad Core Opteron 8356 (2.3GHz)
	キャッシュメモリ	L2: 512KB/ コア L3: 2MB/ プロセッサ
	理論演算性能	9.2GFLOPS

各ノードは図 3 のように構成されています。32GB（一部 128GB）のメモリは 4 つに分割されて各プロセッサに接続されていますが、すべてのプロセッサからすべてのメモリにアクセスができます。ただし近くのメモリにアクセスする場合と遠くのメモリにアクセスする場合で性能が異なるので、アプリケーションの最適化の際には注意が必要です。

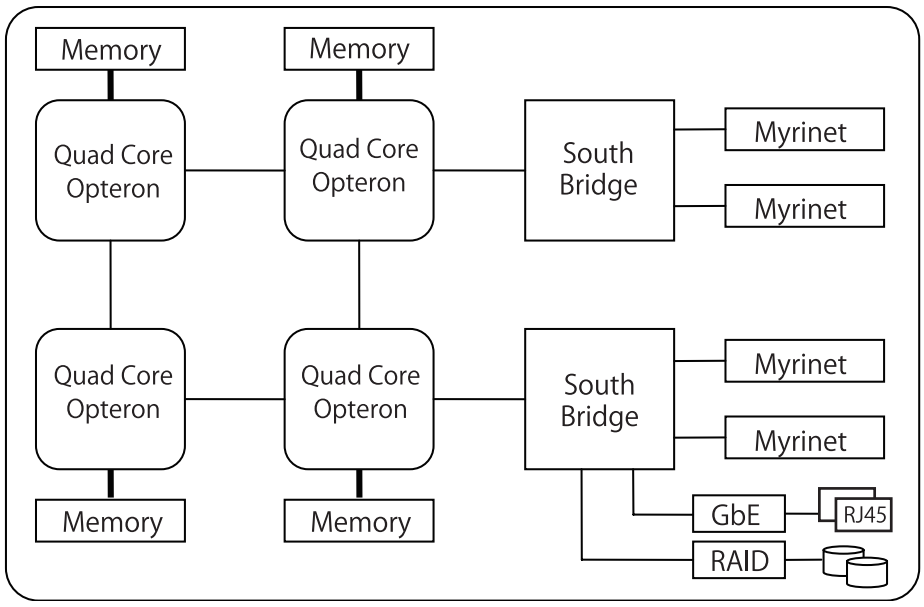

図 3　ノード構成

1.4 ソフトウェア構成

HA8000 クラスタシステムにインストールされている主なソフトウェアは表 3 のとおりです。

表 3　ソフトウェア一覧

項目	ソフトウェア名
OS	RedHat Enterprise Linux 5.1
シェル	bash/sh tcsh/csh zsh
バッチシステム	日立製 NQS
コンパイラ	日立製作所製　最適化 Fortran 77/90/95 日立製作所製　最適化 C　最適化標準 C++ Intel Compiler C/C++/Fortran Version 11 PGI Conpiler 7.1 (アカデミックユーザーのみ)
並列計算通信ライブラリ	MPICH-MX (MPI 1.2)
数値計算ライブラリ	MSL2、MATRIX/MPP、MATRIX/MPP/SSS BLAS、LAPACK、ScaLAPACK Intel Math Kernel Library、ACML GotoBLAS
分子計算アプリケーション	Gaussian03

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。

本システムは広く使われている x86 系プロセッサと Linux の組み合わせで構成されているので、多数のフリーソフトウェアや市販のソフトウェアが動くことが期待できます。本センターがサービスとして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）。

1.5 サービス概要

HA8000 クラスタシステム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」、研究グループ単位でまとまった資源を確保するための「専用キューコース」、スーパーコンピュータの一部を占有利用できる「ノード固定コース」を準備しています。それぞれのコースのサービス内容は次の通りです。

1.5.1 パーソナルコース

パーソナルコースは希望される資源量に応じて下のようなコースがあります。

表 4　　パーソナルコースの種類

コース名	使用可能 ノード数	使用可能 ディスク容量	資源割り当て方式
パーソナル 1	4	100GB	パーソナルコースの他の利用者と共有し、Fair Share 方式によって実行順序を決定
パーソナル 2	8	100GB	
パーソナル 3	16	100GB	
パーソナル 4	32	100GB	
パーソナル 5	64	100GB	

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 5 の利用者が 16 ノードのジョブを実行することは可能です。また、デバッグのための 4 ノード 5 分のキューやパラメーターチューニングなどのために

4,8,16 ノードについては実行可能時間が短い代わりに待ち時間も短い short キューが準備されています。

さらに、利用者のプログラムの大規模化を支援するため、申し込みコースの上限より大きなジョブを月に一回だけ実行できます（大規模ジョブ実行サービス）。このジョブは申込コースより 1 レベル高いコースの上限ノード数（パーソナル 1 の方は 8 ノード、など）を利用できます。パーソナル 5 の方は 256 ノードジョブが実行できます。

1.5.2 専用キューコース

専用キューコースは研究グループなどで利用されるためのコースです。8 ノード単位でご希望のノード数が申込みいただけます（利用者数によってはご希望に添えない場合があります）。標準ディスク容量は 8 ノードあたり 4TB です。また、64 ノード以上をお申し込みのグループは月に一回の 256 ノードジョブが実行可能です。

このコースではバッチジョブキューがご希望のポリシーで作成できます。例えば、16 ノードを申し込みの上で 8 ノードジョブが同時に 2 本流れるようなキューと 16 ノードジョブ用のキューを作成するなどの設定が可能です。次に紹介するノード占有との違いは、物理ノードを他の専用キューコースの方と共有することです。例えば 16 ノードを申し込むグループが 20 グループあっても、320 ノードでなく 240 ノード程度しか割り当てを行いませんので、自グループのジョブが一個も実行されない時間帯が存在します。公平な資源分配のため、ジョブの実行時間は原則 48 時間以内とさせていただきますが、特別な理由がある場合は「制限時間延長申込書」をご提出下さい。センターで延長の可否を決定させていただきます。

1.5.3　ノード固定コース

ノード固定コースは、ノード数を確定する必要のある商用ライブラリの利用や、利用環境をカスタマイズする必要があるユーザー向けのコースです。例えば利用ノードが一定しない専用キューコースではライセンス代が非常に高価になるソフトウェアをご利用の場合、あるいはローカルディスクの使用ポリシーをカスタマイズするなどの必要がある場合にお申し込みいただけます。8 ノード単位での申し込みができます。標準ディスク容量は 8 ノードあたり 4TB です。64 ノード以上ご利用のグループは月に一回 256 ノードジョブが実行できます（占有利用しているノードとは別のノードを使用します）。

なお、ノード固定コースにお申込の場合は、「ノード固定コース 申込時添付書類」のご提出をお願いします。センターでご使用の可否を決定させていただきます。

2

ログイン方法

2.1 概要

HA8000 クラスタシステムには 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 HA8000 クラスタシステムに公開鍵を登録する
4. 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）かで異なりますので、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。HA8000 クラスタシステム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表 5 のようになります。

表 5　接続情報

ホスト名	ha8000-1.cc.u-tokyo.ac.jp ha8000-2.cc.u-tokyo.ac.jp ha8000-3.cc.u-tokyo.ac.jp （どれに接続しても同じです。負荷分散にご協力下さい）
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 HA8000 クラスタシステム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動かします。

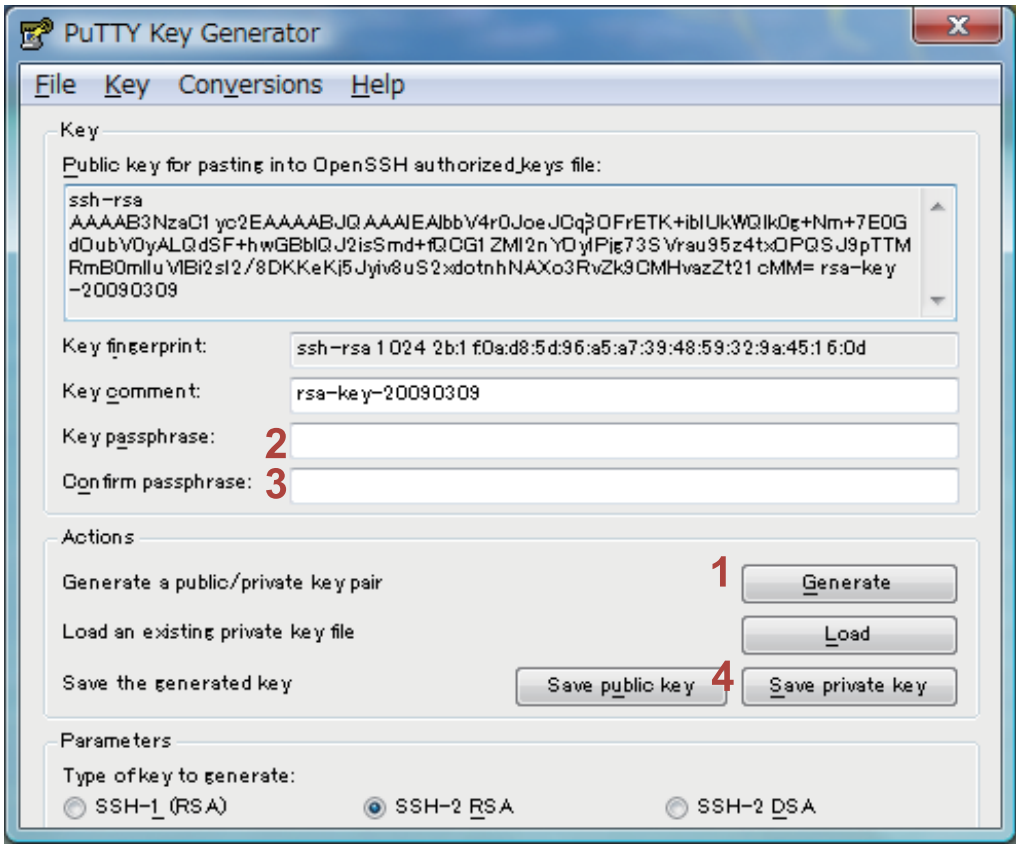

図 4　鍵の生成

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 HA8000 クラスタシステム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 HA8000 クラスタシステムに不正侵入される危険性があります。Save public key は必要ありません（Save private key 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 HA8000 クラスタシステムへの登録

HA8000 クラスタシステムには 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://ha8000.cc.u-tokyo.ac.jp/user/
```

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 を起動すると図 6 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
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。

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。

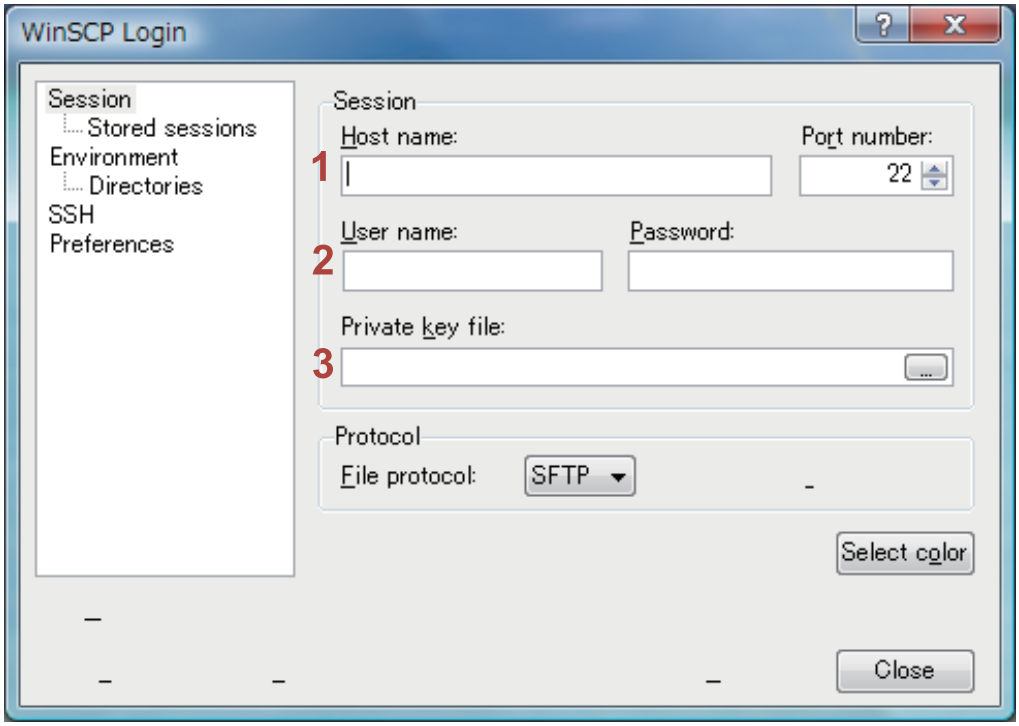

図 5　WinSCP によるファイルコピー

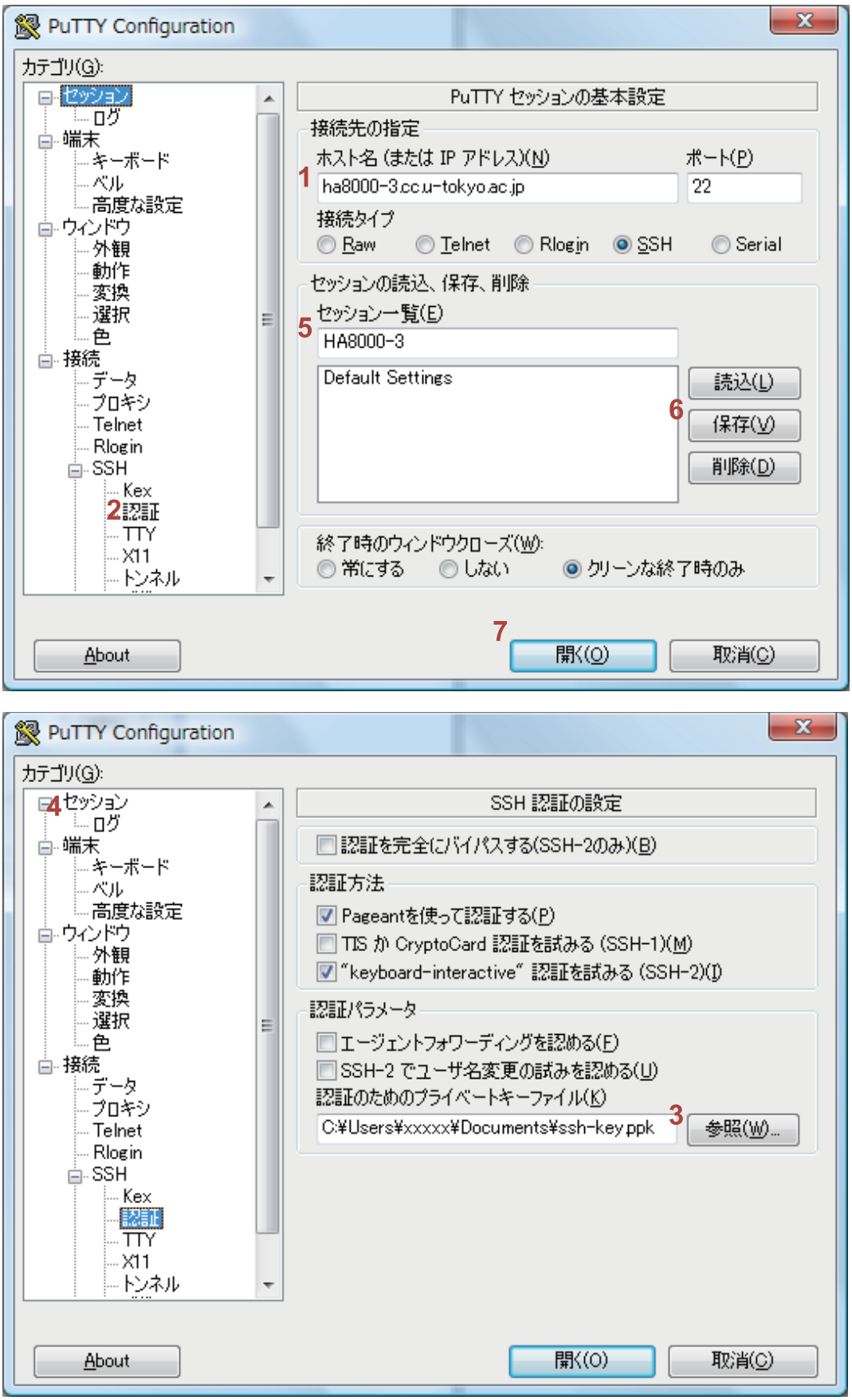
PuTTY Configuration
カテゴリ(G):
セッション
ログ
端末
キーボード
ベル
高度な設定
ウィンドウ
外観
動作
変換
選択
色
接続
データ
プロキシ
Telnet
Rlogin
SSH
Kex
認証
TTY
X11
トンネル
PuTTY セッションの基本設定
接続先の指定
ホスト名 (または IP アドレス)(N)
ポート(P)
1
ha8000-3.cc.u-tokyo.ac.jp
22
接続タイプ
Raw
Telnet
Rlogin
SSH
Serial
セッションの読込、保存、削除
セッション一覧(E)
5
HA8000-3
Default Settings
読込(L)
6
保存(V)
削除(D)
終了時のウィンドウクローズ(W):
常にする
しない
クリーンな終了時のみ
About
7
開く(O)
取消(C)
PuTTY Configuration
カテゴリ(G):
4
セッション
ログ
端末
キーボード
ベル
高度な設定
ウィンドウ
外観
動作
変換
選択
色
接続
データ
プロキシ
Telnet
Rlogin
SSH
Kex
認証
TTY
X11
トンネル
SSH 認証の設定
認証を完全にバイパスする(SSH-2のみ)(B)
認証方法
Pageantを使って認証する(P)
TIS か CryptoCard 認証を試みる (SSH-1)(M)
"keyboard-interactive" 認証を試みる (SSH-2)(I)
認証パラメータ
エージェントフォワーディングを認める(F)
SSH-2 でユーザ名変更の試みを認める(U)
認証のためのプライベートキーファイル(K)
C:¥Users¥xxxxx¥Documents¥ssh-key.ppk
3
参照(W)...
About
開く(O)
取消(C)

図 6　ログイン方法

2.4 UNIX システムからのログイン方法

Linux や MacOS 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下さい。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 HA8000 クラスタシステム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 HA8000 クラスタシステムへの登録

鍵の登録のために、次の URL にアクセスしてください。

`https://ha8000.cc.u-tokyo.ac.jp/user/`

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 ~/.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。

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 z00000@ha8000-3.cc.u-tokyo.ac.jp
```

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 z00000@ha8000-3.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下さい。

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

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 HA8000 クラスタシステムにログインし、~/.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下さい。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 HA8000 クラスタシステム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下さい。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下さい。複数のマシンから HA8000 クラスタシステム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 HA8000 クラスタシステム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 別システムへのログインに使用している鍵の流用

一台のクライアントマシンから HA8000 クラスタシステム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 HA8000 クラスタシステム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 HA8000 クラスタシステム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 HA8000 クラスタシステムにも登録することをお勧めします。

UNIX システム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）ha8000-*.cc.u-tokyo.ac.jp には秘密鍵 ~/.ssh/id_rsa_ha8000 を使用するとき

```
Host ha8000-*.cc.u-tokyo.ac.jp
        User z00000
        IdentityFile ~/.ssh/id_rsa_ha8000
```

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 bash または tcsh が使用できます。デフォルトは bash です。本章ではすべて bash 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。

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 ~/.bash_profile 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。~/.bash_profile 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。次に例を示します。

```
export PATH=/home/z00000/software/bin:$PATH
export LD_LIBRARY_PATH=/home/z00000/software/lib:$LD_LIBRARY_PATH
```

3.4 利用できるプログラム

ログインノードでは RedHat Enterprise Linux に含まれているほぼすべてのプログラムが使用できます。各プログラムの使用方法及び UNIX システムの一般的な使用方法については本手引きでは触れませんので市販の参考書などをご覧ください。

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 16 個の CPU 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 1CPU 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。

```
$ ps
  PID TTY          TIME CMD
13295 pts/0    00:00:00 bash
13500 pts/0    00:00:00 a.out
$ kill 13500
```

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

HA8000 クラスタシステム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 top、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。

```
\z00000
g00000@mail.ecc.u-tokyo.ac.jp
```

1 行目は HA8000 クラスタシステム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。HA8000 クラスタシステム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。

HA8000 クラスタシステム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。

4

ファイルシステム

4.1 概要

HA8000 クラスタシステムでは合計 1PB の大容量ストレージと各ノードのローカルディスクが使用できます。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のホームディレクトリおよび小容量 NFS 領域が使用できます。申込制で大容量 NFS 領域も使用できます。また、計算中の一時データ格納領域としては 5 日で削除される容量制限のない領域および各計算ノードのローカルディスクが使用できます。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。

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
- /nfs/all/ ログイン名
- /nfs/ グループ名 / ログイン名（申込制）

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

それぞれのディレクトリには次のような特徴があります。

/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 HSFS(Hitachi Striping File System) が使用されています。HSFS 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。そのため /home は並列アプリケーションからのデータ入出力に向いています。一方、HSFS はファイル操作のレスポンスがよくないため、プログラムの編集やコンパイルなどの日常作業には不向きです。ログインノードにおける作業時など、細かいファイル操作が多いときは NFS 領域をご利用ください。

/nfs/all/ ログイン名　（１アカウントあたり 10GB まで使用可）

/nfs/all はログインノードにおける作業時などファイル操作に対するレスポンスを重視する場合に使用する領域です。プログラムの編集やコンパイル作業などにお使い下さい。ファイルシステムには NFS を用いています。この領域にはすべてのノードからアクセスできますが、NFS はサーバの負荷を分散する機能を持たないため、多数のノードからの同時アクセスを苦手とします。そのため、並列アプリケーションからのデータ入出力にこの領域を使うことはご遠慮ください。

/nfs/ グループ名 / ログイン名　（申込制）

上記の NFS 領域では容量不足である利用者、または並列アプリケーションからの入出力に NFS 領域を使用したい利用者のための領域です。申込制となっており、容量は申込時にセンターとの話し合いにより決定させていただきます。パーソナルコースの方もお申し込み可能です（「/nfs/personal/ ログイン名」などの名前になります）。この領域と /home の合計容量が各コースのディスク容量（ディスク増量をされている方は増量後の容量）となるよう、各ディレクトリの使用量制限値を設定させていただきます。/nfs/all とは別のサーバ、別のディスクを使用します。/nfs/all とは異なり、MPI などの並列アプリケーションからご利用いただくことができます。NFS が負荷分散機能を持たないこと、また他の利用者の影響も受けることをご理解の上ご利用ください。

/short/ ログイン名　（容量制限なし）

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。バックアップはありません。ファイルシステムは HSFS 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。

/tmp　（ノードあたり 140GB）

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。

4.3 並列ファイルシステム HSFS について

4.3.1 HSFS 概要

HA8000 クラスタシステムのように大量のノードが存在するシステムでは NFS のようなファイルサーバが一台しか配置できないようなネットワークファイルシステムではサーバが過負荷となる可能性があります。そこで /home や /short には日立製作所の HSFS(Hitachi Striping File System) と呼ばれる並列ファイルシステムが使われています。このシステムでは NFS とは異なり、複数のクライアントからのアクセスを複数のサーバで処理することができるため、大量のノードからのアクセスに耐えるだけのサーバを配置できます。本システムでは 952 台のノードが 16 台のファイルサーバに接続されています。

下図のように HSFS はディレクトリ構造やファイルの実体の存在場所を記録したメタデータサーバとファイルの実体を格納するサーバを持ちます。HSFS を使用するノードはメタデータサーバにファイルの場所を問い合わせた上で実体を持つサーバと通信を行いファイルにアクセスします。これらの操作はシステムにより自動的に行われるため利用者は HSFS の仕組みをそれほど意識する必要はなく、ローカルディスクのファイルを扱うのと同じ操作でファイルを扱うことができます。

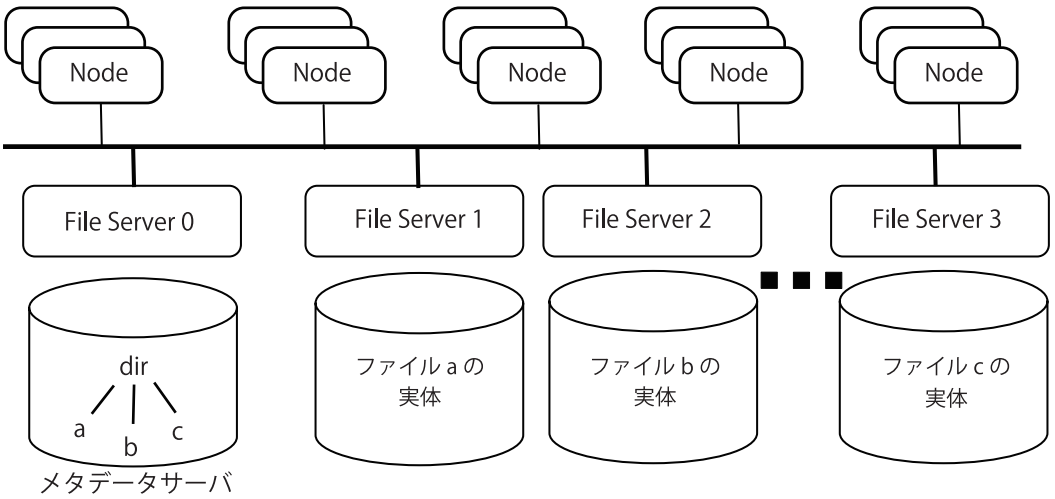

図 7　HSFS の動作概要

4.3.2 HSFS の使い方

HA8000 クラスタシステムの HSFS を高い性能で使うため、以下の点に注意してご利用ください。

同時アクセスは別のファイルに対して行う

HA8000 クラスタシステムの HSFS は「ファイルストライプモード」と呼ばれるモードに設定されています。このモードではファイル単位で実体を保存するサーバが選択されます。単一のファイルが複数のサーバに分割

されて格納されることはありません。そのため、同一ファイルに多数のノードからアクセスすると一つのサーバに負荷が集中し、高い性能は得られません。

大きなバッファで入出力を行う

HSFS は大規模ファイルの入出力に特化して最適化されており、小規模の入出力では高い性能が得られません。特に C 言語をご利用の方は 4MB 以上のデータ入出力をまとめてファイルシステムに要求するようプログラミングしてください。日立製作所製 Fortran をご利用の場合は実行時ライブラリが自動的にバッファリングして大規模リクエストを作成するので、特殊な工夫は基本的には不要です。

複数ノードからアクセスする

ネットワークやプロセッサの負荷分散の観点から、並列アプリケーションからの入出力は複数のノードから行うようにプログラミングしてください。代表ノードのみが入出力を行うタイプのアプリケーションは HA8000 クラスタシステムでは高い性能は得られません。

5

並列計算

5.1 概要

スーパーコンピュータ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。

HA8000 クラスタシステムでは両方の並列化手法が使用できます。ただし、本システム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を基本とし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コースは準備していません（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）。つまり本システムでは MPI 通信によるアプリケーションの並列化は必須となり、さらにノード内での並列化を最適化するために必要に応じて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 HA8000 クラスタシステムで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は想定されていませんが、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。

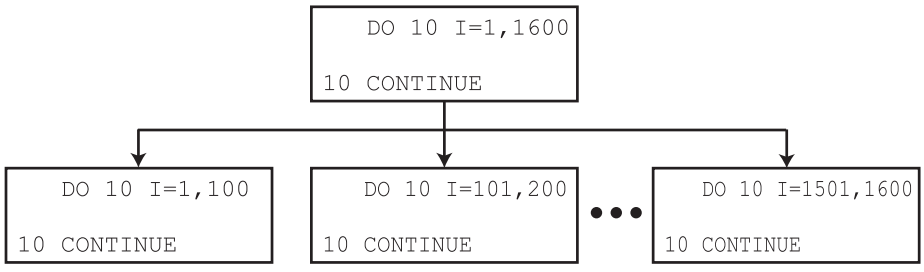

図 8　自動並列化のイメージ

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。HA8000 クラスタシステムを導入した日立製作所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います。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。

HA8000 クラスタシステムでは日立製作所製の最適化コンパイラ、Intel コンパイラ、または PGI 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。

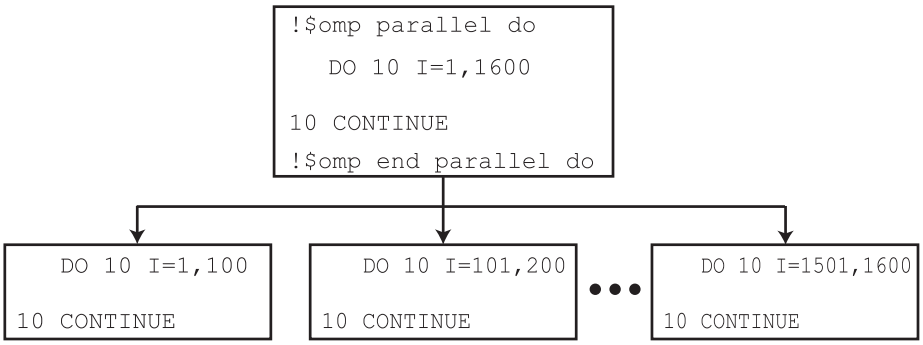

図 9　OpenMP 使用イメージ

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。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したプログラムをお持ちの方は HA8000 クラスタシステムでも引き続き使用できます。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 10 に MPI を使ったプログラムのイメージを示します。

プログラムの実行は各ノードで行われますが、図をよく見ると二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 では基本的には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ノード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 10　MPI のプログラム例

HA8000 クラスタシステム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。コンパイル、実行方法については次章以降で詳しく紹介します。

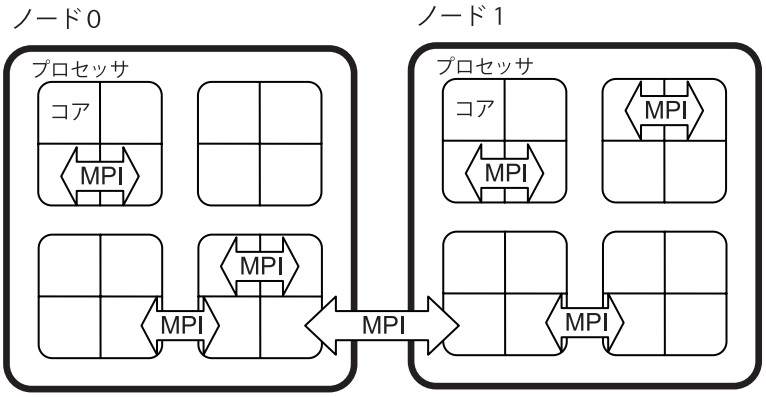

図 11　ピュア MPI での実行イメージ

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 4 スレッドに並列化したプロセスを 4 つ生成することで 16 個の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。コンパイル、実行方法については次章以降で説明します。

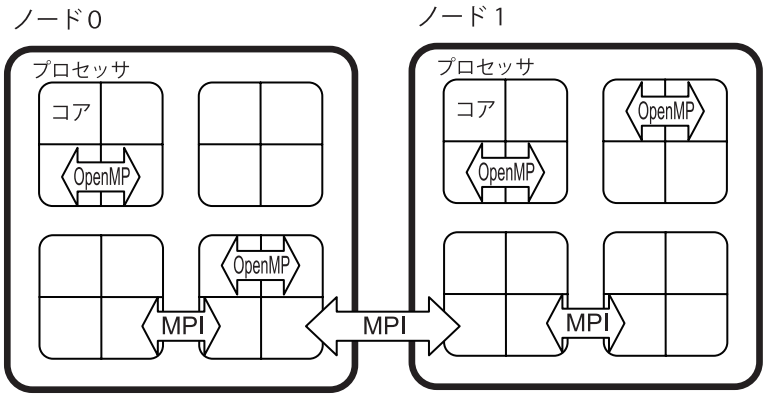

図 12　MPI と OpenMP の併用イメージ

6

コンパイラ

6.1 概要

HA8000 クラスタシステムでは日立製作所製最適化 C,C++,Fortran 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の過去のスーパーコンピュータおよび SR11000 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 HA8000 クラスタシステムを利用できます。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。

HA8000 クラスタシステムには日立製作所のコンパイラの他、Intel Compiler Version 11 と PGI Compiler Version 7.1 もインストールされています（PGI コンパイラはアカデミックライセンスで導入されているため企業ユーザーの方はご利用になれません）。これらのコンパイラは Opteron プロセッサで動作する高速なコードを生成できるコンパイラとして広く使われている製品です。

さらに本システムでは RedHat Enterprise Linux に標準で付属する 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 HA8000 クラスタシステム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
$  mpicxx [-noparallel] [options] source_file
```

MPI + 自動並列化

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

MPI + OpenMP

```
$  mpicc -parallel -omp [options] source_file
$  mpicxx -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 FORTRAN コンパイラ

本節では HA8000 クラスタシステムの標準 FORTRAN コンパイラである日立製作所製最適化 FORTRAN 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 FORTRAN コンパイラは f90 コマンドで起動します。f77 コマンドもインストールされていますがこれは昔の日立製作所製の計算機で使われていたコンパイラとの互換性を維持するためのコマンドで、一部の機能が使用できません。特別な理由がない限り FORTRAN77 プログラムも f90 コマンドでコンパイルして下さい。

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したが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。-parallel が設定されるので要注意
-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。-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　-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 言語 Fortran 共通	-O4	プログラム全体の最適化
	-approx	除算で，逆数による乗数化を許可する
	-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
	-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 7　-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 のすべて
	-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）

6.4.2 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 および表 7 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 の後に置くきます。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コア数 (HA8000 クラスタシステムでは常に 16) 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 Intel コンパイラ

本節では Intel コンパイラの使い方を紹介します。

6.6.1 基本的な使い方と推奨オプション

とりあえず使ってみたい方のため、解説の前に基本的な使い方の例を示します。

```
$ source /opt/itc/mpi/mpiswitch.sh mpich-mx-intel11
$ icc -xSSE3 -O3 -ipo -no-prec-div program.c
$ mpicc -xSSE3 -O3 -ipo -no-prec-div mpi-program.c
```

6.6.2 環境設定

次のコマンドで Intel コンパイラに必要な PATH などの環境変数が設定されます。また MPI 用のコンパイルコマンド（mpif90 など）が Intel コンパイラを使用するように切り替わります。

```
$ source /opt/itc/mpi/mpiswitch.sh mpich-mx-intel11  （通常の環境）
$ source /opt/itc/mpi/mpiswitch.csh mpich-mx-intel11（tcsh に変更している場合）
```

Intel コンパイラには PATH などの環境変数を設定する環境設定シェルスクリプトが付属していますが、HA8000

クラスタシステムではそれらを明示的に呼び出す必要はありません。

6.6.3 コマンド

Intel コンパイラは次のように呼び出して使用します。上から順に C 言語、C++ 言語、FORTRAN です。MPI を使う場合のコマンドは前節で説明した方法で Intel コンパイラ用の環境に切り替わっていることを前提とします。MPI プログラムのコンパイラはコンパイラの種類によらず同じコマンド名なので、前節の設定を行っていない場合は日立製作所最適化コンパイラを使う MPI コンパイラが動きます（日立製コンパイラに Intel コンパイラのオプションを渡すことになるので多くの場合はエラーになります）。

並列化なし

```
$ icc [option] source_file
$ icpc [option] source_file
$ ifort [option] source_file
```

自動並列化

```
$ icc [option] -parallel source_file
$ icpc [option] -parallel source_file
$ ifort [option] -parallel source_file
```

OpenMP

```
$ icc [option] -openmp source_file
$ icpc [option] -openmp source_file
$ ifort [option] -openmp source_file
```

MPI

```
$ mpicc [option] source_file
$ mpicxx [option] source_file
$ mpif90 [option] source_file
```

MPI + 自動並列化

```
$ mpicc [option] -parallel source_file
$ mpicxx [option] -parallel source_file
$ mpif90 [option] -parallel source_file
```

MPI + OpenMP

```
$ mpicc [option] -openmp source_file
$ mpicxx [option] -openmp source_file
$ mpif90 [option] -openmp source_file
```

6.6.4 オプション

主なオプションを次の表にまとめます。

オプション	内容
-O	最適化を行います
-O3	強い最適化を行います。通常はこのオプションがお勧めです
-xSSE3	SSE3 ベクトル演算命令を生成します
-ipo	複数の関数にまたがっての最適化を行います
-no-prec-div	除算を逆数の乗算に変換します。計算精度がわずかに落ちます
-fixed	(FORTRAN) 固定フォーマットを仮定します
-free	(FORTRAN) 自由フォーマットを仮定します
-convert big_endian	(FORTRAN) unformatted データをビッグエンディアンと仮定します

6.6.5 実行時オプションおよび環境変数

OpenMP または自動並列化でコンパイルしたプログラムの実行時スレッド数は OMP_NUM_THREADS 環境変数で指定できます。自動並列化の場合もこの環境変数を参照します。省略時はコア数になります。

6.7 PGI コンパイラ

本節では　PGI コンパイラの使い方を紹介します。PGI コンパイラはアカデミックライセンスで導入されておりますので企業ユーザーの方はご利用になれません。ご了承下さい。

6.7.1 基本的な使い方と推奨オプション

とりあえず使ってみたい方のため、解説の前に基本的な使い方の例を示します。

```
$ source /opt/itc/mpi/mpiswitch.sh mpich-mx-pgi
$ pgcc -fast -O3 -tp=barcelona-64 program.c
$ mpicc -fast -O3 -tp=barcelona-64 mpi-program.c
```

6.7.2 環境設定

次のコマンドで PGI コンパイラに必要な PATH などの環境変数が設定されます。また MPI 用のコンパイルコマンド（mpif90 など）が PGI コンパイラを使用するように切り替わります。

```
$ source /opt/itc/mpi/mpiswitch.sh mpich-mx-pgi　（通常の環境）
$ source /opt/itc/mpi/mpiswitch.csh mpich-mx-pgi（tcsh に変更している場合）
```

6.7.3 コマンド

PGI コンパイラは次のように呼び出して使用します。上から順に C 言語、C++ 言語、FORTRAN77, Fortran90 です。MPI を使う場合のコマンドは前節で説明した方法で PGI コンパイラ用の環境に切り替わっていることを前提とします。MPI プログラムのコンパイラはコンパイラの種類によらず同じコマンド名なので、前節の設定を行っていない場合は日立製作所最適化コンパイラを使う MPI コンパイラが動きます（日立製コンパイラに PGI コンパイラのオプションを渡すことになるので多くの場合はエラーになります）。

並列化なし

```
$ pgcc [option] source_file
$ pgCC [option] source_file
$ pgf77 [option] source_file
$ pgf90 [option] source_file
```

自動並列化

-Mcouncur オプションがありますが性能が良くないのでお勧めしません

OpenMP

```
$ pgcc [option] -mp source_file
$ pgCC [option] -mp source_file
$ pgf77 [option] -mp source_file
```

```
$ pgf90 [option] -mp source_file
```

MPI

```
$ mpicc [option] source_file
$ mpicxx [option] source_file
$ mpif77 [option] source_file
$ mpif90 [option] source_file
```

MPI + OpenMP

```
$ mpicc [option] -mp source_file
$ mpicxx [option] -mp source_file
$ mpif77 [option] -mp source_file
$ mpif90 [option] -mp source_file
```

6.7.4 オプション

主なオプションを次の表にまとめます。

オプション	内容
-fast	通常効果のあるすべての最適化オプションを有効にします
-O3	強い最適化を行います。-fast のあとに付けると効果的です
-tp=barcelona-64	HA8000 クラスタシステムで使われているプロセッサ向けのコードを生成します
-Mfixed	(FORTRAN) 固定フォーマットを仮定します
-Mfree	(FORTRAN) 自由フォーマットを仮定します
-Mbyteswapio	(FORTRAN) unformatted データをビッグエンディアンと仮定します

6.7.5 実行時オプションおよび環境変数

OpenMP または自動並列化でコンパイルしたプログラムの実行時スレッド数は OMP_NUM_THREADS 環境変数で指定できます。自動並列化の場合もこの環境変数を参照します。省略時はコア数になります。

6.8 GNU Compiler Collection

HA8000 クラスタシステムには RedHat Linux 標準の 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++, gfortran です。詳しい使い方は info gcc コマンドなどでご確認下さい。また、GCC を使用する MPI も準備してあります。

```
$ source /opt/itc/mpi/mpiswitch.sh mpich-mx-gcc
```

を実行すると GCC 用の MPI 環境に切り替わります。

6.9 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

HA8000 クラスタシステムではすべてのプログラムがバッチジョブシステム (NQS) を通じて実行されます。HA8000 クラスタシステムでは 日立製作所製の NQS が導入されており、本センターの他のスーパーコンピュータと同じジョブスクリプトが使用できます。

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。3 章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。

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。

HA8000 クラスタシステムでは図 13 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 キューや専用キューコースの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センターホームページの「ジョブクラス制限値」をご覧ください

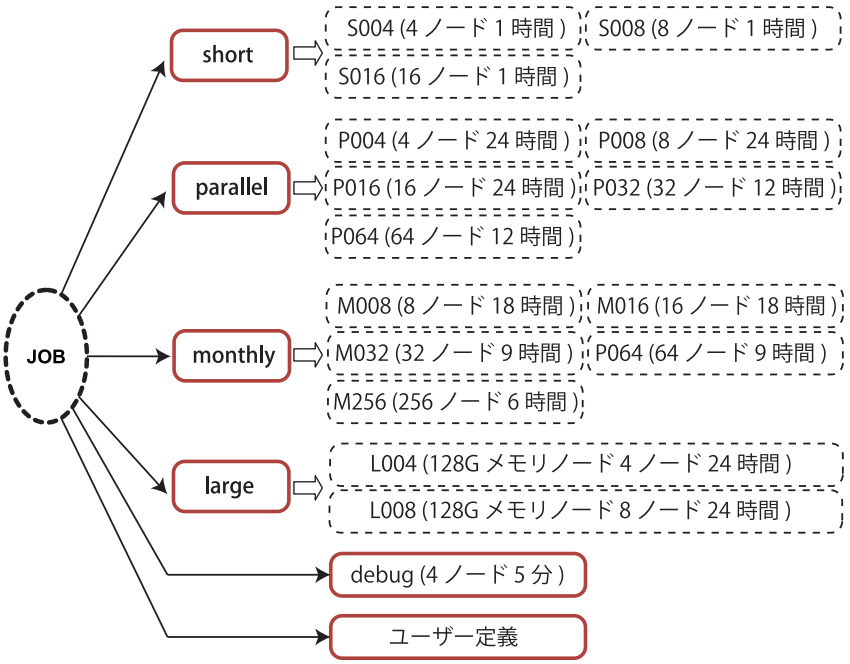

図 13　HA8000 クラスタシステムのキュー

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/bash

#@$-q parallel
#@$-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 8:00:00

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。9 章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 10 ノードを指定した場合は 16 ノード用のバッチキューに挿入されます。

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 8:00:00

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ことをお勧めします。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。

```
$ 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。

```
$ qstat
2008/06/01 (Sun) 13:47:21:    BATCH QUEUES on HA8000
NQS schedule stop time : 2008/06/27 (Fri) 9:00:00 (Remain: 474h 42m 39s)
   REQUEST   NAME      OWNER    QUEUE  PRI NICE     CPU    MEM   STATE
1425.batch1  job.sh    p08000   P004    63  0    unlimit   28GB  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ときは No requests. となります。

7.4.3 キューの状態確認

キューごとの実行ジョブ，待ちジョブ数は以下のように qstat -b で確認できます。

```
$ qstat -b
2008/06/01 (Sun) 13:47:21:    BATCH QUEUES on HA8000
NQS schedule stop time : 2008/06/27 (Fri) 9:00:00 (Remain: 474h 42m 39s)
QUEUE NAME   STATUS    TOTAL   RUNNING    RUNLIMIT  QUEUED HELD  IN-TRANSIT
S004         AVAILBL      1         1           6       0    0           0
S008         AVAILBL      7         3           3       4    0           0
```

RUNNING が実行中のジョブの数、RUNLIMIT が同時に実行できるジョブの数、QUEUED が順番待ちのジョブの数です。

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とき、または実行中のジョブを強制終了したいときは qdel コマンドを使用します。

```
$ qdel 1425.batch1
```

7.4.5 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 HA8000 クラスタシステムのアカウント）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）

7.4.6 システム障害時の動作

システム障害でジョブが途中で終了してしまったときは異常終了したジョブの名前とジョブ番号を個別にご連絡します。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が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致しません。

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。専用キューコースまたはノード固定コースでは投入順に実行される「FIFO スケジューリング」のキューを作成することもできます。

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×使用したコア数」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。

8

数値計算ライブラリ

8.1 概要

HA8000 には数値計算ライブラリとして MATRIX/MPP，MATRIX/MPP/SSS，MSL2，BLAS，LAPACK，ScaLAPACK 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。

8.2 MATRIX/MPP

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、高速フーリエ変換、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。（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	（要素並列版）	`-lmatmpp`
	（スカラー版）	`-lmatmpp_sc`
MATRIX/MPP/SSS（スカイライン法）	（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-E75)
「行列計算副プログラムライブラリ - スカイライン法 - MATRIX/MPP/SSS」(3000-3-E76)

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。
（要素並列版は –parallel オプションも同時に指定して下さい。）

MSL2	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-E74)
「数値計算副プログラムライブラリ MSL2 行列計算」(3000-3-E71)
「数値計算副プログラムライブラリ MSL2 関数計算」(3000-3-E72)
「数値計算副プログラムライブラリ MSL2 統計計算」(3000-3-E73)

8.4 Intel Math Kernel Library

本センターでは、Intel コンパイラと一緒に、Intel Math Kernel Library Version 10.1（MKL) がインストールされています。MKL を使用する場合には、Intel コンパイラをご利用ください。GNU コンパイラ、PGI コンパイラでも利用可能ですが、一部機能が使えない場合があります。日立コンパイラはサポート対象ではありません。

MKL では BLAS、LAPACK、ScaLAPACK をはじめ、フーリエ変換ルーチンなど多くの数値演算ライブラリを提供しています。また、OpenMP に対応しており、スレッド化されたルーチンではマルチコア・マルチプロセッサシステムで最適なパフォーマンスを得ることが可能です。

8.4.1 リンク方法

MKL を利用するためには環境変数の設定が必要となります。「6.6 Intel コンパイラ」の環境設定を行った上で、MKL の環境設定を行います。次のコマンドで Intel コンパイラの設定と MKL の設定が行われます。MKL のバージョンアップが行われた場合、バージョン番号の部分（10.1.0.015）が変更となります。

```
（通常の環境）
$ source /opt/itc/mpi/mpiswitch.sh mpich-mx-intel11
$ source /opt/intel/mkl/10.1.0.015/tools/environment/mklvarsem64t.sh
（ログインシェルを csh 系に変更している場合）
$ source /opt/itc/mpi/mpiswitch.csh mpich-mx-intel11
$ source /opt/intel/mkl/10.1.0.015/tools/environment/mklvarsem64t.csh
```

OpenMP を使用する MKL の動的リンクでのコンパイルでは、リンク時に以下のようにライブラリを指定してください。紙面の都合上以下では２行で書かれていますが、実際には改行を入れずに入力してください。

```
$ icc program.c -lmkl_intel_lp64 -Wl,--start-group -lmkl_intel_thread
-lmkl_core -Wl,--end-group -liomp5 -lpthread -lm
```

OpenMP を使用しない MKL の動的リンクでは、以下のようになります（同じく改行は入れません)。

```
$ icc program.c -lmkl_intel_lp64 -Wl,--start-group -lmkl_sequential
-lmkl_core -Wl,--end-group -lpthread -lm
```

静的リンクでのコンパイルでは、上記オプションの先頭に -static を追加してください。

8.4.2 実行時の設定

動的リンクのプログラム実行時には、LD_LIBRARY_PATH に Intel コンパイラのライブラリ、MKL のライブラリのパスが設定されている必要がありますので、バッチスクリプトの最初に環境変数設定用のコマンドを実行するように記述してください。

OpenMP を使用する場合、スレッド数の設定には OpenMP 環境変数 OMP_NUM_THREADS、もしくは MKL 環境変数 MKL_NUM_THREADS が利用できます。MKL 環境変数は OpenMP 環境変数よりも優先して参照されます。どちらの変数も指定されない場合は、デフォルトのスレッド数が選択されます。実際の利用例に関しては、パッケージ付属のサンプルソースが /opt/intel/mkl/10.1.0.015/examples 以下にあるので、プログラム開発の参考にしてください。

8.4.3 マニュアル

MKL の詳細は、/opt/intel/mkl/10.1.0.015/doc 以下にドキュメントがあります。mklman.pdf が Ver.10.1 のリファレンスマニュアル（英語版）となります。また関数リファレンス部分に関しては、Ver.9 のものですが日本語版のリファレンスマニュアル（mklman90_j.pdf）も参照可能です。Web 上の資料としては以下のページで情報が入手できます（2009 年 2 月現在）。

```
http://www.intel.com/cd/software/products/ijkk/jpn/329226.htm
http://www.xlsoft.com/jp/products/intel/perflib/mkl/
```

8.5 AMD Core Math Library

本センターでは、PGI コンパイラと一緒に、AMD Core Math Library Version 3.6 (ACML) がインストールされています。ACML を使用する場合には、PGI コンパイラをご利用ください。GNU コンパイラ、Intel コンパイラでも利用可能ですが、一部機能が使えない場合があります。日立コンパイラはサポート対象ではありません。

ACML では BLAS、LAPACK に加え、フーリエ変換ルーチンなど多くの数値演算ライブラリを提供しています。また、OpenMP に対応しており、スレッド化されたルーチンではマルチコア・マルチプロセッサシステムでよりパフォーマンスを得ることが可能です。

8.5.1 リンク方法

OpenMP 版 ACML のコンパイルは以下のようになります。

```
$ pgf77 -mp source.f   -lacml_mp
$ pgf95 -mp source.f90 -lacml_mp
$ pgcc  -mp source.c   -lacml_mp
```

シリアル版 ACML の場合は次の通りです。

```
$ pgf77 source.f   -lacml
$ pgf95 source.f90 -lacml
$ pgcc  source.c   -lacml
```

8.5.2 実行時の設定

OpenMP を使用する場合、スレッド数の設定には OpenMP 環境変数 OMP_NUM_THREADS を使用します。環境変数が指定されない場合は、デフォルトのスレッド数（本センターでは 16）が選択されます。

8.5.3 マニュアル

ACML の詳細は、AMD のホームページをご参照ください。

```
http://developer.amd.com/cpu/libraries/acml/Pages/default.aspx
```

2009 年 2 月現在の最新版は、ACML Version 4.2 で、上記ホームページより入手可能です。各種コンパイラで使用できるライブラリが提供されています。使用方法などに関しては以下のアドレスから情報が入手できます（URL はすべて 2009 年 2 月現在）。

```
http://developer.amd.com/cpu/libraries/acml/Pages/default.aspx
http://www.softek.co.jp/SPG/Pgi/TIPS/acml.html
```

8.6 GotoBLAS

HA8000 クラスタシステムには Goto BLAS ver.1.26 がインストールされています。Goto BLAS 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。（並列版は、日立コンパイラでは -parallel オプションも同時に指定して下さい）。

GotoBLAS	並列版	`-L/opt/itc/lib -lgoto_smp`
	スカラー版	`-L/opt/itc/lib -lgoto`
	16 並列限定版	`-L/opt/itc/lib -lgoto_smp16`

GotoBLAS は、GNU コンパイラおよびインテルコンパイラではサポート対象ですが、日立コンパイラはサポート対象ではありません。日立コンパイラでも多くの場合は動きますが、一部複素数系の関数が利用できません。また結果の正当性などには十分ご注意ください。

並列版におけるコア数は、OpenMP 環境変数 OMP_NUM_THREADS、もしくは、GotoBLAS 用環境変数 GOTO_NUM_THREADS にスレッド数を設定します。

8.7 BLAS・LAPACK・ScaLAPACK

HA8000 クラスタシステムの標準 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。ただし、これらは Opteron プロセッサ向けに最適化された実装よりは性能が悪くなります。可能な限り、前節までに紹介した MKL、ACML、GotoBLAS などの最適化されたライブラリをご使用ください。

システム標準の BLAS 等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。（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	`-lblas`	`-lblas_sc`
LAPACK	`-llapack` `-lblas`	`-llapack_sc` `-lblas_sc`

ScaLAPACK	`-lscalapack`
	`-lblacsBASE`
	`-lblacsF77`
	`-lblas`
		`-lscalapack_sc`
		`-lblacsBASE_sc`
		`-lblacsF77_sc`
		`-lblas_sc`

なお、ScaLAPACK の通信関数には MPI を使用していますので、使用時には mpif90 や mpicc などの MPI 用コマンドをご使用ください。

9

使用例

9.1 概要

本章では HA8000 クラスタシステムを使用例として、一般的な次の 2 パターンについて、プログラムのコンパイルと実行方法を紹介します。

- ピュア MPI で 32 ノード 512 プロセスのプログラムを実行する場合
- ノード内は OpenMP 4 スレッド× 4 プロセスで、32 ノード 128 プロセス、512 スレッドで実行する場合

9.2 ピュア MPI 実行の例

本節ではピュア MPI プログラムのコンパイルおよび実行例を紹介します。

9.2.1 プログラムのコンパイル

ソースプログラムは program.f と仮定します。コンパイルは次のように行います。

```
$ mpif90 -Oss -noparallel -o program program.f
```

各プロセスは単一スレッドで動作するので `-Oss` に含まれる自動並列化オプションを無効にするよう `-noparallel` をつけることが重要です。

9.2.2 ジョブスクリプトの作成

ジョブスクリプトは次のようになります。

```
#!/bin/bash
#@$-q  parallel
#@$-N  32
#@$-J  T16
#@$-lT 8:00:00

cd program/
mpirun ./program
```

パイプキューの名前は適宜変更が必要です。上の例では 32 ノード実行で各ノードに 16 プロセスを生成する設定が書かれています。シェルスクリプト部分では実行可能ファイルが存在するディレクトリに移動し、mpirun コマンドによって MPI プログラムを起動しています。

9.2.3 プログラムの実行

前のステップで作成したジョブスクリプト program.job を引数にして qsub コマンドを実行します。

```
$ qsub program.job
```

9.2.4 結果の確認

実行が終了するとカレントディレクトリに program.oXXXXX と program.eXXXXX というファイルができま

す。XXXXX はジョブ番号です。これらがそれぞれ標準出力と標準エラー出力の内容です。

9.3 OpenMP + MPI ハイブリッド実行の例

本節では OpenMP と MPI のハイブリッドで並列化して実行する例を紹介します。

9.3.1 プログラムのコンパイル

ソースプログラムは program_mp.f と仮定し、OpenMP ディレクティブが書かれていることを仮定します。このプログラムを次のようにコンパイルします。

```
$ mpif90 -omp -Oss -o program_mp program_mp.f
```

9.3.2 ジョブスクリプトの作成

プログラムを 1 ノードあたり 4 プロセス、1 プロセスあたり 4 スレッドとし、32 ノードで実行するためのジョブスクリプトの例を示します。J オプションにはノードあたりの MPI プロセス数を設定するのでこの場合は 4 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 OMP_NUM_THREADS 環境変数を 4 に設定します。

```
#!/bin/bash
#@$-q parallel
#@$-N  32
#@$-J T4
#@$-lT 8:00:00

export OMP_NUM_THREADS=4
cd program/
mpirun ./program_mp
```

9.3.3 プログラムの実行と結果の確認

プログラムの実行方法と結果の確認方法は前節の例と同じです。

9.4 NUMA 最適化実行の例

最後の例として HA8000 クラスタシステムでなるべく高いアプリケーション性能を得るために少し工夫をした実行方法を紹介します。HA8000 クラスタシステムの各ノードは図 14 のように各プロセッサにメモリが直結されています。計算中の CPU がメモリアクセスを行う際、自身に直結されているメモリにアクセスするのは高速ですが、他の CPU に接続されているメモリにアクセスする場合は時間がかかります。このように共有メモリ計算機でありながら、アクセスするメモリの場所によって性能が異なる計算機アーキテクチャを「NUMA(Non-Uniform Memory Access) アーキテクチャ」と言います。本節では NUMA アーキテクチャ計算機でのアプリケーション実行を高速化するために、プロセスが使用する CPU とメモリの配置を固定し、なるべく高速なメモリを使用して並列ジョブを実行する方法を紹介します。

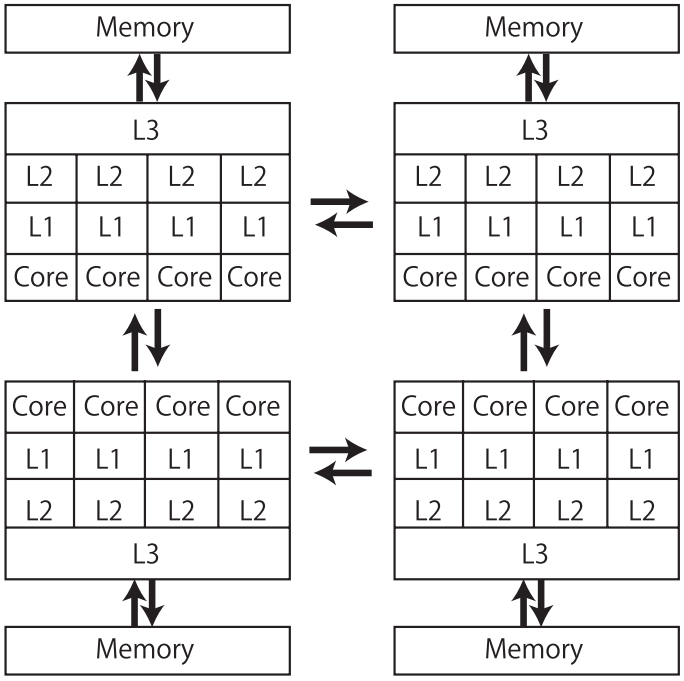

図 14　NUMA 構成

9.4.1 プログラムのコンパイル

プログラムのコンパイル時に NUMA 最適化を行う必要はありません。前に挙げた例と同じようにコンパイルできます。利用者自身で日立製でないコンパイラや独自のライブラリをインストールした場合は、それらのコンパイラやライブラリが NUMA 最適化を行わないことを確認してください。異なる二通りの最適化を行うとそれらが衝突してうまく動かない場合があります。

9.4.2 NUMA 最適化スクリプトの準備

次のシェルスクリプトを準備してください。ここでは ~/numarun という名前にします。実行権限を付けておいてください。ここでは前節の OpenMP と MPI のハイブリッド実行を NUMA 最適化します。このスクリプトは環境変数から MPI のランク番号を調べ、それを 4 で割った余りで使用する CPU を決定します (--cpunodebind オプション)。各 MPI プロセスに属す全スレッドがこの CPU に固定されるため、スレッドは同じ CPU の別のコアに割り当てられることになります。また、使用するメモリの位置も固定し (--membind オプション)CPU に直結されたメモリを優先的に使用します。

```
#!/bin/bash
MYRANK=$MXMPI_ID
NODE=$(expr $MYRANK % 4)
/usr/bin/numactl --membind $NODE --cpunodebind $NODE $@
```

このスクリプトは本システムの MPI である MPICH-MX に強く依存しています。使用する MPI によってランク番号が格納される環境変数が異なるので、お手元のクラスタなどに応用する場合は注意が必要です。

9.4.3 ジョブ投入

ジョブスクリプトは次のように書きます。前節の OpenMP 4 スレッド + MPI のハイブリッド実行の例を NUMA 最適化します。

```
#!/bin/bash
#@$-q  parallel
#@$-N  32
#@$-J  T4
#@$-lT 8:00:00

export OMP_NUM_THREADS=4
cd program/
mpirun ~/numarun ./program_mp
```

ジョブスクリプトも通常の実行とほとんど差がありません。mpirun で実行するプログラムを先ほど作成したシェルスクリプトにし、そのスクリプトを経由してアプリケーションの実行可能ファイルを実行しています。これで各プロセスがなるだけ高速にアクセスできるメモリを使用して実行できます。

ピュア MPI 実行の時もまったく同じ方法で NUMA 最適化が可能です。ただし、上の numarun を使った場合はランク番号に対して CPU がサイクリックに割り当てられます。隣り合うランクが同じ CPU 内の別のコアを使用するように割り当てたい場合は numarun を変更する必要があります。

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://ha8000.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。

- 最適化 C　言語
- 最適化 C　使用の手引
- 最適化 C++ 使用の手引
- 最適化 Fortran　言語
- 最適化 Fortran 使用の手引
- MATRIX/MPP
- MATRIX/MPP/SSS
- MSL2 関数計算
- MSL2 行列計算
- MSL2 統計計算
- MSL2 操作

10.2 マニュアルの購入

マニュアルの CD-ROM 及び印刷物を購入することもできます。

```
http://www.hitachi-imdx.jp/
```

の「オンラインマニュアル販売」からお申し込みください。詳しくは下記までお問い合わせください。

(株) 日立インターメディックス
電話 03-5281-5054

索引

記号

欧字

A

B

C

F

G

H

I

K

L

M

N

O

P

Q

R

S

T

W

かな

え

か

き

こ

さ

し

す

せ

HA8000 クラスタシステム　利用の手引き

2009 年 3 月 5 日　第 2 版発行

本文
　松葉　浩也　　（第 1，2，3，4，9 章、第 5，6，7 章の一部）
　黒田　久泰　　（第 5，6，8 章の一部）
　片桐　孝洋　　（第 5，6，8 章の一部）
　吉廣　保　　（第 6，8 章の一部）
　システム運用係　　（第 7 章の一部）

編集・表紙・組版
　松葉　浩也

発行
　東京大学情報基盤センター　スーパーコンピューティング部門
　〒 113-8658
　東京都文京区弥生 2-11-16
　TEL: 03-5841-2717

RYTB-20090317