統計数理研究所
計算統計学支援システム
スーパーコンピュータシステム
利用者ガイド

2.1 版

はじめに

本手引書は統計数理研究所にて稼動する計算統計学支援システムの利用方法についてまとめたものです。

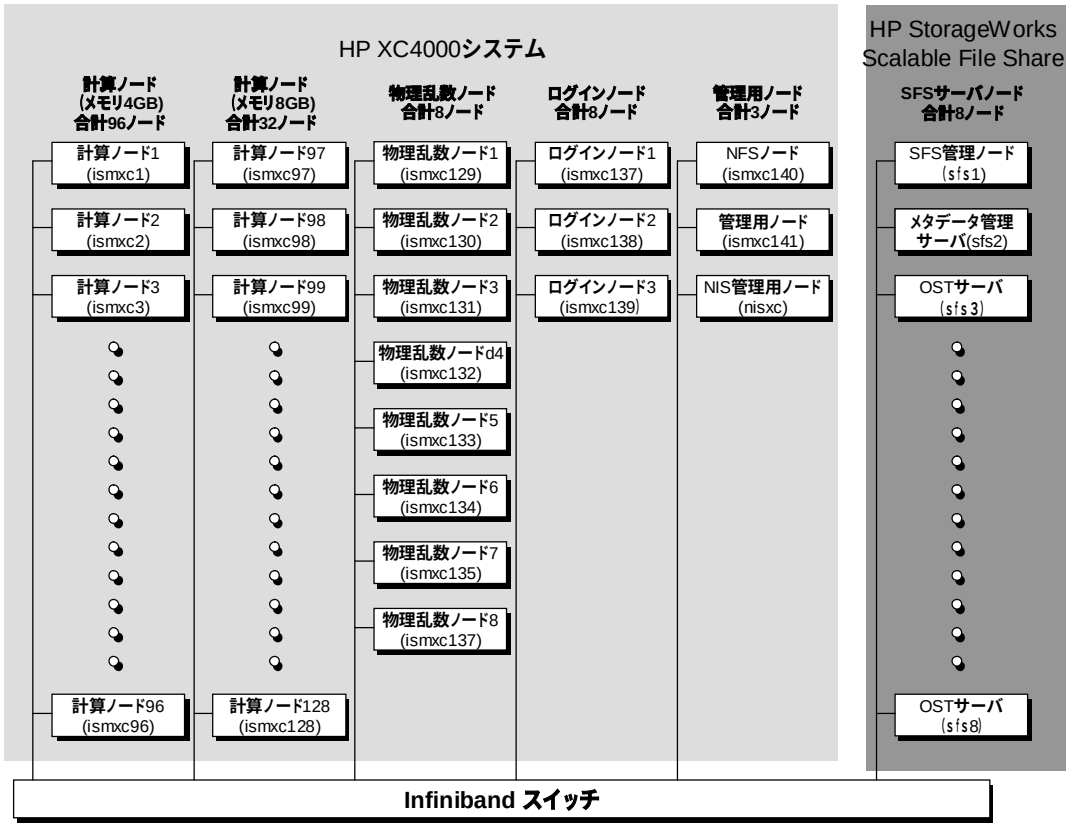
改版履歴

版数	内容	発行日
1.0	初版	2005 年 12 月 19 日
1.1	プログラム実行方法の記述を加筆。章立てを変更	2006 年 1 月 4 日
2.0	ユーザ説明会向けに内容を刷新	2006 年 5 月 12 日
2.1	講習会時に要望された項目を追加	2006 年 5 月 29 日

1 スーパーコンピュータ HP XC4000 の概要

1.1 ハードウェア構成

スーパーコンピュータとしては米国ヒューレット･パッカード社が開発した HP XC4000（以降 XC4000 と表記）とそのストレージを管理する Storageworks SFS サーバが導入されています。XC4000 は下図のように役割の異なる合計 141 台の SMP サーバから構成された PC クラスタシステムで、ノード間は 10Gbps の転送速度を持つ低遅延な Infiniband により相互接続されています。XC4000 ではプログラミングスキームとして、OpenMP（ノード内並列）と MPI（ノード間並列）が利用できます。



各ノードの種別とその役割を下表に示します。

ノード種別	役割
計算ノード ProLiant DL145G2	ユーザが投入したバッチジョブの実行。最大 32 ノード（64CPU）使った MPI 並列計算が可能。
物理乱数ノード ProLiant DL585	物理乱数ボードを搭載し、XC4000 内あるいはネット経由で乱数を生成するために使用する。乱数生成のための API（C,Fortran 用）が用意されている。
ログインノード ProLiant DL385	ユーザがログインして、プログラムの開発、デバッグ、ジョブ投入に使用する。ログインノードは 3 台用意されており、クラスタ機能による負荷分散が行われている。
NFS ノード ProLiant DL385	SFS ファイルシステム上で管理されているユーザのホーム領域を、他のシステムに NFS でエクスポートするために使用。
管理用ノード ProLiant DL385	クラスタの管理やシステムの状態監視を行うために使用する。

各ノードのスペックとその台数を下表に示します。

ノード種別	ノード数	スペック
計算ノード 1 ProLiant DL145G2	96	CPU: Opteron252 (2.6GHz 1MB キャッシュ) x 2 メモリ: 4GB ディスク: 80GB ネットワーク： Infiniband
計算ノード 2 ProLiant DL145G2	32	CPU: Opteron252 (2.6GHz 1MB キャッシュ) x 2 メモリ: 8GB ディスク: 80GB ネットワーク： Infiniband
物理乱数ノード ProLiant DL585	8	CPU: Opteron850 (2.4GHz 1MB キャッシュ) x 2 メモリ: 4GB ディスク: 36GB ネットワーク： Infiniband その他： 物理乱数生成ボード
ログインノード ProLiant DL385	3	CPU: Opteron252 (2.6GHz 1MB キャッシュ) x 2 メモリ: 4GB ディスク: 36GB ネットワーク： Infiniband
NFS ノード ProLiant DL385	1	CPU: Opteron252 (2.6GHz 1MB キャッシュ) x 2 メモリ: 4GB ディスク: 36GB ネットワーク： Infiniband
管理用ノード ProLiant DL385	1	CPU: Opteron252 (2.6GHz 1MB キャッシュ) x 2 メモリ: 4GB ディスク: 36GB ネットワーク： Infiniband

ストレージ

外部ストレージとしては 12TB の容量を持つ HP SFS（StorageWorks Scalable File Share based on Lustre technology）が用意されています。HP SFS は数千台規模の PC クラスタシステムに接続することを前提に開発されたパラレル・ファイル・システムで、NFS 等で問題になっていたファイル入出力によるボトルネックを回避しています。

1.2　ソフトウェア構成

XC4000 は RedHat 社の RedHat Linux Enterprise Linux AS 3 (EM64T) をベースに、MPI 環境を提供する HP　MPI、リソース管理ユーティリティ SLURM、ジョブ管理ツール LSF、そしてクラスタ管理ツールの LVS といったものを組み合わせて作られた PC クラスタ環境です。

統計数理研究所の XC では開発環境として、GNU コンパイラに加えて、Intel 社の Intel コンパイラ (C++/Fortran)、PGI 社の PGI コンパイラ (C++/Fortran)、Pathscale 社の Pathscale EKO Compiler Suite (C++/Fortran) がインストールされており、MPI 及び OpenMP による並列プログラムの作成が可能となっています。

ソフトウェア種別	ソフトウェア名
オペレーティングシステム	RedHat Enterprise Linux AS3.0　Update 4 ベース
コンパイラ	gcc　3.2.3 Intel C/C++ 8.1、9.0　(EM64T 版) Intel Fortran 8.1、9.0 (EM64T 版) Pathscale　2.2.1 PGI Fortran　5.1-6,5.2-4,6.0-8
ライブラリ	NAG ACML HP MLIB GOTO BLAS Library
ノード間通信ライブラリ	HP MPI version　v1.2
デバッカ	gdb (cgdb) v6.1 idb 8.1、9.0(インテル版) pgdbg(PGI 版)
ジョブ管理システム	Platform LSF(Load Sharing Facility)
リソース管理システム	SLURM　(Simple Linux Utility for Resource Management)
クラスタソフトウェア	Linux Virtual Server
システム管理ツール	Nagios, SuperMon, Parallel Distributed Shell
環境設定ツール	Module 3.1.6

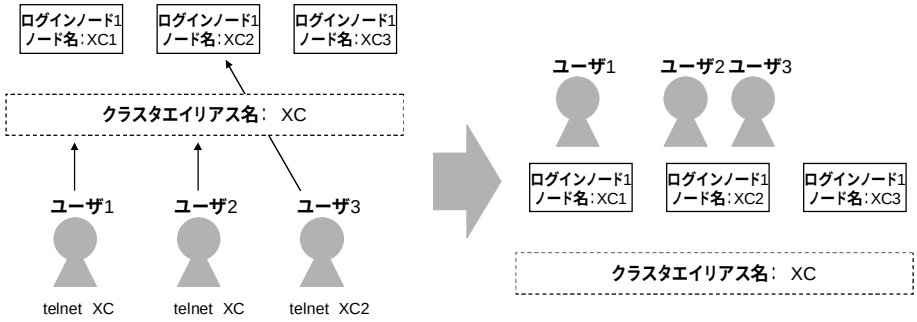
2 スーパーコンピュータの利用方法

2.1 システムへのログイン方法

XC4000 には 3 台のログインノードが用意されています。プログラムの作成、コンパイル、バッチジョブの投入などは全てこのログインノード上で行います。
ログインノードに接続するには、一般的な Linux マシンと同じく、SSH や Telnet 等が利用できます。XC4000 にログインする時に、ホスト名として ｓｍｘｃ.ism.ac.jp を指定すると、3 台のノードのいずれかにログインできます。(次節クラスタエイリアスを参照)

2.1.1 クラスタエイリアス

XC4000 では複数のログインノードを設けて、そこにひとつのホスト名 (IP アドレス) を付けることができます。これにより、ユーザから複数のノードがあたかも 1 台であるかのように見えてきます。この機能はクラスタエイリアスと呼ばれ、XC4000 上の LVS (Linux Virtual Server) と呼ばれるソフトウェアによって実現されています。例えば下図では XC1～3 の 3 台のログインノードに XC というひとつの名前が付けられています。この XC という名前がクラスタエイリアス名にあたります。



SSH や telnet、FTP にクラスタエイリアスを指定すると、ログインノードのいずれかに接続され、一部のノードに負荷が集中しないようになっています。それに対して、特定のノードを利用したい場合は、各ノードのホスト名を明示的に指定します。上の図ではユーザ 1、2 がクラスタエイリアス名 XC を使って telnet した場合、それぞれ順に XC1、XC2 にログインしたことを表しています。ユーザ 3 は明示的に XC2 を指定したため、XC2 にログインします。

XC４０００では各ログインノードの設定は基本的に同じになっています。ただし/tmp 等一部の領域は個々のノードで独立に持っているため注意が必要です。

計算統計学支援システムにおけるログインノードとその IP アドレスは下表のようになっています。

ホスト名	IP アドレス	備考
ismxc.ism.ac.jp	133.58.105.12	クラスタエイリアス名として ismxc1～3 をまとめたもの
ismxc1.ism.ac.jp	133.58.105.21	ログインノード 1
ismxc2.ism.ac.jp	133.58.105.22	ログインノード 2
ismxc3.ism.ac.jp	133.58.105.23	ログインノード 3

2.1.2 XC4000 にログインする場合の例 (UNIX,Linux)

UNIX (Linux) システムから XC4000 に接続する例を以下に示します。

ssh コマンドの場合

```
% ssh -l ユーザ名 ismxc.ism.ac.jp      （クラスタエイリアスを指定した場合）

% ssh –l ユーザ名 ismxc1.ism.ac.jp（クラスタエイリアスを使用しないで直接ノード名
                                    を指定する場合）
```

SSH を使用する場合、ログイン時に以下のようなメッセージが出る場合があります。
このような場合には、"yes"とタイプして次に進んでください。

```
The authenticity of host 'ismxc (XXX.XXX.XXX.XXX)' can't be established.
RSA key fingerprint is XX:XX:XX:XX:XX:XX:XX:XX:XX:XX:XX:XX:XX:XX:XX:XX.
Are you sure you want to continue connecting (yes/no)?
```

システムに接続するとパスワード入力を促されます。正しいパスワードを入力すると、
システムにログインできます。

```
ユーザ名@ismxc's password: ＜パスワード入力＞
[ユーザ名@ismxc ユーザ名]$
```

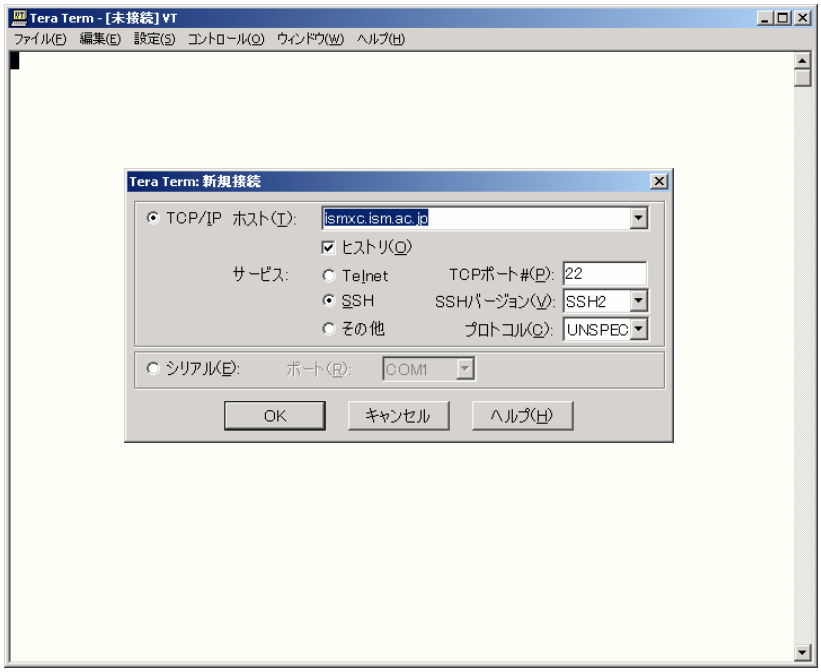
2.1.3 XC4000 にログインする場合の例 (Windows)

Windows から XC4000 に接続するには、TeraTerm や PuTTy 等の仮想端末ソフトを使うと便利です。各ソフトは以下のサイトから無料でダウンロードできます。

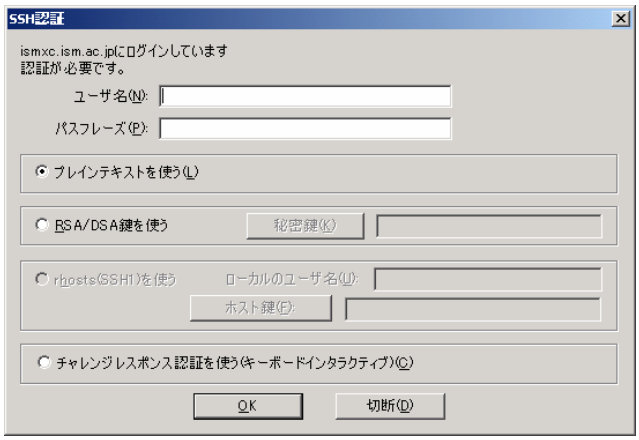
アプリケーション名	ダウンロードサイト
TeraTerm UTF8 対応版	http://sourceforge.jp/projects/ttssh2/
PuTTY	http://www.chiark.greenend.org.uk/~sgtatham/putty/
PuTTY 用日本語パッチ	http://hp.vector.co.jp/authors/VA024651/

各アプリケーションのインストール使用法については、インストールサイトの情報を参考にしてください。

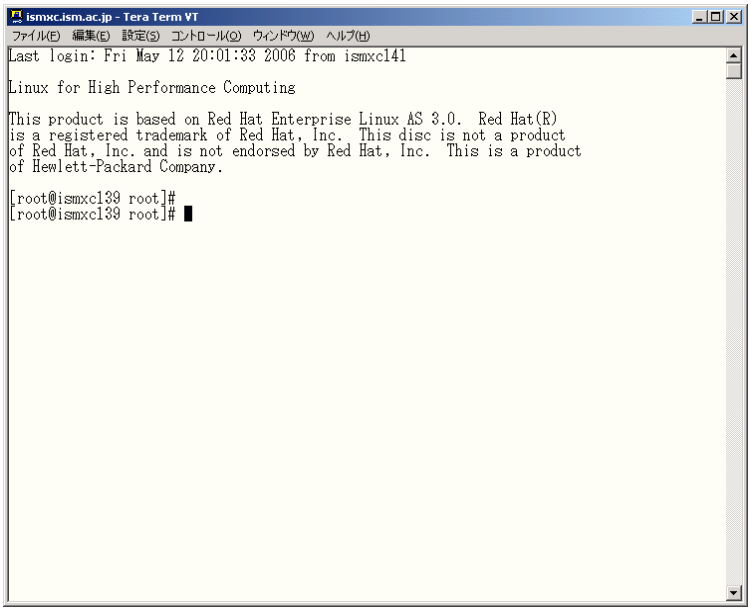
① TeraTerm を起動し、新規接続ダイアログにホスト名(ismxc.ism.ac.jp)を入力します。



② SSH 認証ダイアログにユーザ名、パスフレーズ（パスワード）を入力します。



③ 認証に成功すると仮想端末画面が表示されます。



2.1.4 パスワードの変更方法

XC4000 は NIS を使ってパスワード管理を行っています。そのためパスワードを変更する場合には、passwd コマンドの変わりに yppasswd コマンドを使用します。システムのセキュリティを維持するために、定期的にパスワードを変更することをお勧めします。

```
% yppasswd
Changing NIS account information for ismtest on nisxc.
Please enter old password: XXXXXXX （古いパスワードを入力）
Changing NIS password for ismtest on nisxc.
Please enter new password:  XXXXXXX      （新しいパスワードを入力:非表示）
Please retype new password: XXXXXXX      （新しいパスワードを再入力：非表示）

The NIS password has been changed on nisxc.
```

2.2 スーパーコンピュータで利用できるコマンド

XC4000 のオペレーティング・システムは、RedHat Linux Enterprise Linux AS 3.0 がベースになっています。そのため Linux の一般的なコマンドがそのまま利用できます。
XC4000 用として特別に用意されたコマンドとしては、PATH や環境変数等の設定を切り替えるための module コマンド、ジョブの投入、削除等を行うための LSF コマンド群、並列プログラムを実行するための SLURM コマンド等が用意されています。これらのコマンドについては後章で後述します。

2.3　ユーザの初期環境

ユーザディレクトリは、以前あったものをそのまま移行しています。ユーザ個々の環境に関わる設定ファイル（.cshrc、.profile、.login など）については、XC4000 用の変更を行っていないため、ログイン後各自で Linux に合わせた修正を行う必要があります。
自分だけジョブが実行できないなど、他のユーザと異なる挙動を示したときは、まず各自の環境を確認してください。
各シェルとその設定ファイルは以下の表を参考にしてください。

シェル種別	関連する主な設定ファイル名
C シェル（tcsh）	~/.tcshrc、~/.cshrc、~/.login
B シェル（bash）	~/.bash_profile、~/.bash_login、~/.profile、~/.bashrc
K シェル（ksh）	~/.profile

3 プログラムの開発

3.1 スーパーコンピュータで利用できる開発環境

本システムには、開発環境として以下のソフトウェアがインストールされています。

ソフトウェア種別	ソフトウェア名
コンパイラ	gcc 3.2.3 Intel C/C++ 8.1、9.0 （EM64T 版） Intel Fortran 8.1、9.0 (EM64T 版) Pathscale 2.2.1 PGI Fortran 5.1-6,5.2-4,6.0-8
ライブラリ	NAG ACML HP MLIB GOTO BLAS Library
ノード間通信ライブラリ	HP MPI version v1.2
デバッカ	gdb (cgdb) v6.1 idb 8.1、9.0(インテル版） pgdbg(PGI 版)
環境設定ツール	Module 3.1.6

上記コンパイラ、ツールおよび、ライブラリは、GNU 関連のものを除き、全て/opt 以下のディレクトリにインストールされています。

3.2 プログラムのコンパイル方法

ここではスーパーコンピュータにおいて、プログラムのコンパイルから実行までに必要な手順を説明します。

3.2.1 モジュールコマンドによる環境設定

スーパーコンピュータでは、コンパイラや各種開発環境に必要な設定を行うために、Module と呼ばれるユーティリティが用意されています。ユーザが使用するコンパイラやライブラリの環境変数（PATH、MANPATH、LD_LIBRARY_PATH 等）はすべてこの Module を使って設定できます。

従来このような設定はユーザごとに.bashrc や.login 等のファイルで行うのが一般的でしたが、module コマンドを使用することで、コンパイラのバージョンを切り替えて利用するなど柔軟な使い方が可能になります。

2006 年 1 月 4 日時点で、スーパーコンピュータには以下のモジュールが用意されています。

モジュール名	説明
acml/2.7.0	AMD Core Math Library (各種コンパイラ用)
goto/1.00	後藤 BLAS ライブラリ
icc/8.1	インテル C コンパイラ(V8.1)
icc/9.0	インテル C コンパイラ(V9.0)
ifort/8.1	インテル Fortran コンパイラ (V8.1)
ifort/9.0	インテル Fortran コンパイラ (V9.0)
idb/8.1	インテルデバッガ (V8.1)
idb/9.0	インテルデバッガ (V9.0)
intel/8.1	インテル C、インテル Fotran(V8.1)
intel/9.0	インテル C、インテル Fotran(V9.0)
mlib/intel	HP MLIB (Math Library) インテルコンパイラ用
mlib/pgi	HP MLIB (Math Library) PGI コンパイラ用
mpi/hp	HP MPI 開発環境
pathscale/2.2.1	Pathscale コンパイラ (V2.2.1)
pgi/5.1-6	PGI コンパイラ (V5.1-6)
pgi/5.2-4	PGI コンパイラ (V5.2-4)
pgi/6.0-8	PGI コンパイラ (V6.0-8)
intel/8.1、9.0 はインテル C、インテル Fortran を同時にロードするためのモジュール	

3.2.2　モジュールコマンドの使い方

■　システムに用意されているモジュールの一覧表示

モジュールの一覧を表示するには”modules avail”と入力します。

```
% module avail
--------------------------------- /opt/modules/version ---------------------------------
3.1.6

---------------------------- /opt/modules/3.1.6/modulefiles ----------------------------
dot          module-cvs   module-info modules      null         use.own

------------------------------- /opt/modules/modulefiles -------------------------------
acml/2.7.0/gnu64                  idb/9.0/default
acml/2.7.0/pathscale64            ifort/8.1/default
acml/2.7.0/pathscale64_mp         ifort/9.0/default
acml/2.7.0/pgi64                  intel/8.1
acml/2.7.0/pgi64_large_arrays     intel/9.0
acml/2.7.0/pgi64_mp               mlib/intel/8.1
acml/2.7.0/pgi64_mp_large_arrays mlib/pgi/5.1
goto/1.00/default                 mpi/hp/default
hptc                              pathscale/2.2.1/default
icc/8.1/default                   pgi/5.1-6/default
icc/9.0/default                   pgi/5.2-4/default
idb/8.1/default                   pgi/6.0-8/default
```

■　モジュールのロード

モジュールを現在の環境に組み込むには、”module load モジュール名” と入力します。例えば PGI コンパイラ（V6.0-8）の実行環境をロードするには、以下のようにします。

```
% module load pgi/6.0-8
```

上記の module avail の例では、PGI コンパイラのバージョン（5.1-6、5.2-4、6.0-8）毎にモジュールが用意されています。このとき、module load pgi とすることで、管理者が推奨するデフォルトバージョンが自動的にロードされます。

■　ロードされたモジュールの一覧表示

現在の環境にロードされているモジュールの一覧を表示するには、”module list”と入力します。例では、現在の環境に Intel C++コンパイラ　バージョン 9.0 が組み込まれていることを示しています。

```
% module list
Currently Loaded Modulefiles:
1) icc/9.0
```

■　モジュールのアンロード

ロード済みのモジュールを、環境から外すには”module unload モジュール名”を入力します。例えば複数のバージョンを切り替えて使いたい場合には、今使っているバージョンをアンロードしてから、再度別のバージョンをロードします。

```
% module unload icc/9.0
```

■ ログイン時の自動組み込み

ログイン時にモジュールが自動的に組み込まれるようにするには、.bashrc や.cshrc などの環境設定ファイルを使用します。例えば、ログインシェルに bash を使用している場合には、ユーザのホームディレクトリにある ~/.bashrc　ファイルに以下のように module のロードコマンドを組み込みます。

```
# if the ‘module ‘ command is defined, $MODULESHOME
# will be set
if [ -n “$MODULESHOME” ] ; then
      module load ロードしたいモジュール名
fi
```

3.2.3　シリアルプログラムのコンパイルと実行方法

シリアルプログラムとは MPI や OpenMP 等の並列化手法を使わないものを言います。これらのプログラムは通常の方法でコンパイル/リンク方法を使って作成します。
次の例ではホスト名を表示するシリアルプログラム(ファイル名 hw_hostname.c)のコンパイル/実行例を示しています。

```
#include <unistd.h>
#include <stdio.h>

int main()
{
    char name[100];
    gethostname(name,sizeof(name));
    printf("%s says Hello!\n",name);
    return 0;
}
```

このプログラムは以下のようにコンパイル/リンクします。

```
プログラムのコンパイル
% module load icc/9.0 ①
% icc hw_hostname.c –o hw_hostname

プログラムの実行（ログインノードで実行）
% ./hw_hostname
ismxc137 says Hello!

プログラムの実行（q1 キューの 1CPU でジョブを実行）
% bsub -I -q q1 -n 1 ./hw_hostname     ②
Job <1234> is submitted to queue <q1>.
<<Waiting for dispatch ...>>
<<Starting on lsfhost.localdomain>>
ismxc95 says Hello!
```

①インテルコンパイラを使用するために、該当するモジュールをロードします。

②LSF の会話型ジョブを使ってジョブを実行します。この例では q1 というキューで 1CPU 使用しています。

3.2.4 並列プログラムのコンパイルと実行方法 (MPI を用いた並列化)

スーパーコンピュータで MPI プログラムのコンパイルやリンクを行うには、HP-MPI が提供するユーティリティ(mpicc,mpif77 等)を使用します。これを利用すると必要なライブラリや検索パスを設定した上で、実際のコンパイラを呼び出してくれます。このユーティリティの利用には、予め HP-MPI のモジュールをロードしておく必要があります。

HP-MPI ではコンパイラ毎に、以下のようなスクリプトを用意しています。各スクリプトともに gcc（含む g77)、Intel コンパイラ、PGI コンパイラに対応します。複数のコンパイラが使用できる場合、Intel コンパイラ、PGI コンパイラ、gcc の順でコンパイラを選択するため、module コマンドを使って使用したいコンパイラを選択してください。

スクリプト名	説明
mpicc	C コンパイラ用スクリプト
mpic++	C++コンパイラ用スクリプト
mpif77	FORTRAN 77 用スクリプト
mpif90	FORTRAN 90/95 用スクリプト
各スクリプトともに/opt/hpmpi/bin に置かれています。	

3.2.4.1 HP-MPI を使ったコンパイル例

HP-MPI が提供するスクリプトを使った並列プログラムのコンパイル例を示します。以下は C 言語で書かれた MPI プログラムの例(hello_world.c)です。

```
#include <stdio.h>
#include <mpi.h>

int main(argc,argv)
int argc; char *argv[];
{
    int rank, size, len;
    char name[MPI_MAX_PROCESSOR_NAME];
    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    MPI_Comm_size(MPI_COMM_WORLD, &size);
    MPI_Get_processor_name(name, &len);
    printf("Hello world! I'm %d of %d on %s\n",rank, size, name);
    MPI_Finalize();

    exit(0);
}
```

このプログラムをコンパイルして実行すると、画面に"*Hello world! I'm r of s on host*"と表示されます。ここで表示される r,s,host は以下の通りです。

r: プロセスのランク
s: コミュニケータのサイズ
host： プログラムを実行したホスト名

① HP-MPI を使用する場合、最初に HP-MPI 用のモジュールをロードします。これのモジュールを組み込まないと、後述の mpicc、mpif90 等のコマンドが利用できません。

```
% module load mpi
```

② C で書かれたプログラムをコンパイルするには mpicc を使用します。これを実行する前に、予め使用するコンパイラに合わせたモジュールをロードしておいてください。例ではインテル C++を使ってコンパイルしています。

```
C によるコンパイル
% module load icc/9.0
% mpicc -o hello_world hello_world.c

プログラムの実行（q8 キューの 8CPU でバッチジョブを実行）
% bsub -q q8 -n 8 mpirun -srun ./hello_world

プログラムの実行（q8 キューの 4CPU で会話型ジョブを実行）
% bsub -I -q q8 -n 4 mpirun -srun ./hello_world
Hello world!
Hello world! I'm 1 of 4 on ismxc1
Hello world! I'm 3 of 4 on ismxc2
Hello world! I'm 0 of 4 on ismxc1
Hello world! I'm 2 of 4 on ismxc2
```

③ FORTRAN90 で書かれたプログラムをコンパイルするには mpif90 を使用します。例ではインテル Fortran のモジュールを使ってコンパイルしています。

```
Fortran によるコンパイル
% module load ifort/9.0
% mpif90 –o hello_world hello_world.f

プログラムの実行（q8 キューの 8CPU でバッチジョブを実行）
% bsub  -q q8  -n 8 mpirun –srun ./hello_world

プログラムの実行（q8 キューの 4CPU で会話型ジョブを実行）
% bsub –I –q q8 –n 4 mpirun –srun ./hello_world
Hello world!
Hello world! I'm 1 of 4 on ismxc1
Hello world! I'm 3 of 4 on ismxc2
Hello world! I'm 0 of 4 on ismxc1
Hello world! I'm 2 of 4 on ismxc2
```

mpicc、mpic++、mpif77、mpif90 のいずれの場合も、そのコンパイラがサポートしているオプションを指定することができます。例えば PGI FORTRAN を使って 2GB 以上のメモリを使用するプログラムをコンパイルする場合は、以下のようになります。

```
PGI コンパイラ用モジュールをロード
% module load pgi/6.0-8

HP-MPI 用モジュールをロード
% module load mpi

PGI コンパイラで 2GB 以上のメモリを使用するために、 -mcmodel=medium、 -i8 のオプションを指定
% mpif90 –o hello_world –mcmodel=medium –i8 hello_world.f
```

⚠注意

MPI プログラムをコンパイルする場合、 -static やこれを暗黙的に含むオプション（インテルコンパイラの -fast）は使用しないでください。XC でリンクされる HP MPI ライブラリは Infiniband 以外にも Myrinet や Ethernet といった様々なインターコネクトをサポートしています。使用するインターコネクトによって、必要なライブラリをダイナミックリンクする方式をとっているため、 -static を使ってライブラリを静的にリンクするとプログラムが実行できなくなる場合があります。
尚シリアルプログラムの場合は、静的リンクも可能です。

3.3 ライブラリの利用

計算統計学支援システムでは、以下のライブラリが利用できます。

ライブラリ種別	概要
ACML (AMD Core Math Library)	AMD 社が自社の Opteron、Athlon プロセッサ向けに開発・提供している数値演算ライブラリ。BLAS1,2,3、LAPACK、ScaLAPACK、FFT 等をサポート
NAG 数値演算ライブラリ	NAG 社が提供する商用数値演算ライブラリ。BLAS1,2,3,LAPACK, FFT、最適化、最小二乗法、固有値問題、偏微分方程式、常微分方程式、曲面・曲線フィッティング等の科学技術計算ルーチン群と、分散分析、時系列予測、主成分分析、クラスタ分析などの統計計算ルーチン群を１２００以上提供。
GOTO BLAS Library 1.0	テキサス大学の後藤氏によるもの BLAS のライブラリの中では最速と呼ばれている。
HP MLIB	HP 純正の数値演算ライブラリ。BLAS1,2,3 スパース BLAS、LAPACK、ScaLAPACK、SuperLU_DIST をサポート

本書では ACML を使ったコンパイル・リンク方法を説明します。

ACML では使用するコンパイラ及び、作成するプログラムの種別 (並列版、大容量メモリ版) によって、リンクするライブラリが分かれています。

3.3.1 GNU コンパイラとのコンパイル・リンク

g77 とのコンパイル・リンク(動的リンク)

```
% g77 -m64 sample.f -L/opt/acml2.7.0/gnu64/lib -lacml
```

g77 とのコンパイル・リンク (静的リンク)

```
% g77 -m64 sample.f -L/opt/acml2.7.0/gnu64/lib -static -lacml
または
% g77 -m64 sample.f /opt/acml2.7.0/gnu64/lib/libacml.a
```

3.3.2 PGI コンパイラとのコンパイル・リンク

PGI Fortran とのコンパイル・リンク (シリアル)

```
% pgf77 -tp=k8-64 -Mcache_align sample.f -L/opt/acml2.7.0/pgi64/lib -lacml
```

PGI Fortran とのコンパイル・リンク (並列)

```
% pgf77 -tp=k8-64 -Mcache_align -mp sample.f -L/opt/acml2.7.0/pgi64_mp/lib -lacml
```

PGI Fortran とのコンパイル・リンク (シリアル、メモリ 2GB 以上使用する場合)

```
% pgf77 -tp=k8-64 -Mcache_align sample.f -L/opt/acml2.7.0/pgi64_large_arrays/lib -lacml
```

PGI Fortran とのコンパイル・リンク (並列、メモリ 2GB 以上使用する場合)

```
% pgf77 -tp=k8-64 -Mcache_align -mp sample.f -L/opt/acml2.7.0/pgi64_mp_large_arrays/lib -lacml
```

3.3.3 Pathscale コンパイラとのコンパイル・リンク

PGI Fortran とのコンパイル・リンク (シリアル)

```
% pathf90 sample.f -L/opt/acml2.7.0/pathscale64/lib -lacml
```

PGI Fortran とのコンパイル・リンク (並列)

```
% pathf90 -mp sample.f -L/opt/acml2.7.0/pathscale64_mp/lib -lacml
```

3.3.4 Intel コンパイラとのコンパイル・リンク

Intel Fortran とのコンパイル・リンク

```
% ifc sample.f -L/opt/acml2.7.0/gnu64/lib -lacml /usr/liba/libg2c.so
```

4 プログラムの実行

4.1 スーパーコンピュータのバッチジョブソフトウェア

XC4000 のバッチジョブソフトウェアとしては LSF(Load Sharing Facility)が導入されています。バッチジョブの投入はログインノードから LSF のコマンドを使って行います。投入されたジョブは、指定されたキューの設定に従って計算ノードへ送られ、ジョブが実行されます。

4.2 キュー構成

現在のキュー構成は下表の通りです。

キュー名	経過時間制限	メモリ制限(GB)	最大並列度	最大CPU 数	最大CPU 数(ユーザ当り)	実行ホスト	備考
q1	120H	2	1	8	2	ismxc1～96	
q8	120H	16	8	32	16	ismxc1～96	default キュー
q8t	360H	16	8	16	8	ismxc1～96	
q16	120H	32	16	32	16	ismxc1～96	
q16t	360H	32	16	16	16	ismxc1～96	
q16m	120H	64	16	32	16	ismxc97～128	
q32	120H	64	32	64	32	ismxc1～96	
q32m	120H	128	32	64	32	ismxc97～128	
q64	120H	128	64	128	64	ismxc1～96	
q64m	120H	256	64	64	64	ismxc97～128	
q128	120H	256	128	128	128	ismxc1～96	通常時 Close

項目名	説明
キュー名	キューに付けられた名前
経過時間制限	そのキューでジョブを流すことが出来る最大経過時間 (walltime)
メモリ制限	1ジョブ当り利用可能な最大メモリ容量
最大並列度	1ジョブ当り同時に利用可能な最大 CPU 数
最大 CPU 数	そのキューで使用できる CPU 数の合計 (全ジョブの)
最大 CPU 数 (ユーザ当り)	そのキューで 1 ユーザが使用できる CPU の合計
実行ホスト	ジョブが実行されるジョブの一覧

- q128 キューは通常利用不可 (CLOSE) の状態になっており、特別な申請があった場合のみ利用が許可されます。
- キュー名を指定しないと、自動的に q8 キューにジョブが投入されます。
- 経過時間制限を越えたジョブは自動的に Kill されます。
- キュー名の末尾のアルファベットが t (time) は 長時間ジョブ用のキュー、m (memory)は、 大容量メモリを必要とするジョブのためのキューをあらわします。このキューを構成するノードは 8GB のメモリを搭載しており、4GB 搭載する他のノードの 2 倍の容量を有します。

4.3　ジョブの実行

XC でのジョブの実行は、SLURM というソフトウェア（コマンド名は srun）を介して行うのが基本です。また並列ジョブを実行する場合には HP-MPI（コマンド名は mpirun）を使います。またこれらジョブをバッチで動作させたい場合は LSF というソフトウェア（コマンド名は bsub）を使います。

4.3.1　バッチジョブの投入

バッチジョブを投入するには bsub コマンドを使用します。

例では MPI で書かれたプログラム"./hello_world"プログラムを、bsub コマンドを使って実行しています。bsub で投入されたジョブは、任意のキューに入れられ実行の順番を待ちます。

```
% bsub mpirun -srun ./hello_world
```

-n オプションを使うと使用するプロセッサ数を指定できます。また、 -ext "SLURM[nodes=数字]"オプションを指定すると、利用するノード数を指定できます。以下の例では、4 ノードで 4 プロセッサ、つまり１ノードあたり１プロセッサを確保しています。

```
% bsub –n 4 –ext “SLURM[nodes=4]" mpirun -srun ./hello_world
```

bsub に -o オプション、 -e オプションを指定すると、標準出力およびエラー出力の内容を、ファイルに記録できます。出力先には、/home または、/work 上のファイルを指定してください（/tmp 等はクラスタ内で共有されていないため、特定のノードからしかファイルが見えないことがあります）。このオプションを付けないと、標準出力/エラー出力の結果が得られないので注意してください。

```
% bsub –n 4 –o test.out –e test.err mpirun -srun ./hello_world
```

大容量メモリ(8GB)搭載するノードを 1 台占有して、1 つのジョブを実行したい場合

```
% bsub –n 1 –ext “SLURM[node=1] mpirun  -srun ./hello_world_mpi
```

bsub でスクリプトを実行する場合、その中で#BSUB を付けて bsub のオプションを指定できます。＃BSUB を使って指定した情報は、bsub コマンドのオプションで指定されたものより優先順位が低くなります。

```
% bsub –q q8 –o outfile –e errfile –t 13:00 –ext “SLURM[node=4]” mpirun
 –srun ./helo_world_mpi

上記と同じことを行うスクリプトを作成（my_script）

% cat my_script (my_script の中身を表示)
#!/bin/sh
#BSUB –q q8
#BSUB –o outfile –e errfile
#BSUB –t 13:00
#BSUB –ext “SLURM[nodes=4]"
srun ./hello_world_mpi

% bsub < ./my_script  （my_script を実行）

% bsub - q q16  < ./my_script  （q16 で実行する。スクリプト中の指定が上書きされる）
```

bsub の主なオプションを以下に示します。

オプション	説明
-q キュー名	ジョブを投入するキューを指定します。
-J ジョブ名	ジョブ名の指定。ジョブを中断する場合等に、このジョブ名を指定することができます。
-n CPU 数	並列計算に使用する CPU 数を指定します
-I	ジョブを会話型モードで実行します。
-c [hour:]munites	ジョブが使用する最大 CPU 時間を指定します。CPU 時間を制限することで、ジョブの暴走等によるリソースの浪費を抑えます。
-t [[month:]day]hour:munites	指定した時間を越えたジョブを強制終了します。
-o ファイル名	指定したファイルに標準出力の内容が記録されます。
-e ファイル名	指定したファイルに標準エラー出力の内容が記録されます。

上記以外にも様々なオプションが用意されています。詳細についてはオンラインマニュアル(man bsub)を参照して下さい。

4.3.1.1 会話型での実行

bsub に -I（Interactive）オプションを付けると、確保した計算ノード上でプログラムを会話的に実行することができます。
以下の例ではシリアルプログラム hw_hostname を、キューq1 で実行しています。実行結果は標準出力に表示されます。

```
% bsub -q q1 -I ./hw_hostname
Job <7574> is submitted to queue <q1>.
<<Waiting for dispatch ...>>
<<Starting on lsfhost.localdomain>>
ismxc95 says Hello!
```

実行プログラム名を指定しないと、プロンプトが表示されます。コマンドを入力して、最後に Ctrl -D を入力すると入力した内容が実行されます。

```
% bsub -q q1 -I
bsub> hostname
bsub> ./hw_hostname
bsub> Ctrl -D
Job <7577> is submitted to queue <q1>.
<<Waiting for dispatch ...>>
<<Starting on lsfhost.localdomain>>
ismxc95
ismxc95 says Hello!
```

4.3.2 ジョブのステータス

サブミットしたジョブの状態を参照するには bjobs コマンドを使用します。このコマンドは通常自分のジョブの情報のみを表示します。他のユーザの状態を表示するには” -u all “オプションを付けてください。

```
$ bjobs -u all
JOBID USER STAT QUEUE   FROM_HOST   EXEC_HOST   JOB_NAME   SUBMIT_TIME
84    test RUN  normal  lsfhost.loc 4*lsfhost.l *bin/xterm date and time stamp
```

ここで表示される各項目の意味を以下に示します。

項目名	説明
JOBID	ジョブ番号（システムが自動的に割り当てます）
USER	ユーザ名
STAT	ジョブの状態
QUEUE	ジョブが投入されたキューの名前
FROM_HOST	ジョブをサブミットしたホスト名
EXEC_HOST	ジョブを実行しているホスト名
JOB_NAME	ジョブの名称 bsub コマンドの-J オプションで指定
SUBMIT_TIME	ジョブが投入された時間

特定のジョブ番号の詳細を確認したいときには”bjobs JOBID”で確認できます。
bjobsには他にもオプションがありますので、詳細についてはオンラインマニュアル(man bjobs)を参照して下さい。
ジョブを投入したもののいつまでたってもジョブが実行されない場合は、bjobs に -p または -lp オプションを付けて実行してください。下図のように実行されない理由が出力されます。

```
% bjobs -p
JOBID USER STAT QUEUE FROM_HOST JOB_NAME SUBMIT_TIME
7678 user1 PEND priority ismxc103 verilog Oct 28 13:08
Queue ’s resource requirements not satisfied:3 hosts;
Unable to reach slave lsbatch server: 1 host;
Not enough job slots: 1 host;

% bjobs -lp
Job Id <7678>, User <user1>, Project <default>, Status <PEND>, Queue <priority>,
Command
<verilog>
Mon Oct 28 13:08:11: Submitted from host <hostD>,CWD <$HOME>, Requested
Resources
<type==any && swp>35>;
PENDING REASONS:
Queue ’s resource requirements not satisfied: hostb, hostk, hostv;
Unable to reach slave lsbatch server: hostH;
Not enough job slots: hostF;
SCHEDULING PARAMETERS:
r15s r1m r15m ut pg io ls it tmp swp mem
loadSched - 0.7 1.0 - 4.0 - - - - - -
loadStop - 1.5 2.5 - 8.0 - - - - - -
```

補足
ジョブの実行状況を知るもうひとつの方法として、SLURM（リソース管理システム）が用意している squeue コマンドがあります。

```
% squeue
  JOBID PARTITION     NAME     USER  ST       TIME  NODES NODELIST
   4194       lsf XC_Clust hogehoge   R 5:21:44:24      4 ismxc[5-8]
   4457       lsf XC_Clust foo        R 1:22:44:50     16 ismxc[4,9-10,12-24]
   4475       lsf XC_Clust var        R   20:20:56      1 ismxc128
```

squeue コマンドは上の例のように、現在使用しているノード情報も表示されます。尚、squeue コマンドで表示される JOBID は、LSF の JOBID とは異なるので注意してください。

4.3.3 キュー情報の表示

キューの情報を表示します。

```
% bqueues
QUEUE_NAME      PRIO STATUS          MAX JL/U JL/P JL/H NJOBS  PEND   RUN  SUSP
q1                20  Open:Active      8    2    -    -     0     0     0     0
q8                20  Open:Active     32   16    -    -     8     0     8     0
q8t               20  Open:Active     16    8    -    -     0     0     0     0
q16               20  Open:Active     32   16    -    -     0     0     0     0
q16t              20  Open:Active     16   16    -    -     0     0     0     0
q16m              20  Open:Active     32   16    -    -     2     0     2     0
q32               20  Open:Active     64   32    -    -    32     0    32     0
q32m              20  Open:Active     64   32    -    -     0     0     0     0
q64               20  Open:Active    128   64    -    -     0     0     0     0
q64m              20  Open:Active     64   64    -    -     0     0     0     0
q128              20  Closed:Inact   128  128    -    -     0     0     0     0
```

項目名	説明
QUEUE_NAME	キューに付けられた名前
PRIO	キューのプライオリティ
STATUS	キューの状態（Open：ジョブの受付可能、Close：ジョブの受付不可能、Active：キュー内のジョブを実行可能、Inactive：ジョブを実行不可能）
MAX	そのキューが利用できるジョブスロット数
JL/U	そのキューでユーザが利用できるジョブスロット数
JL/P	キューからプロセッサが利用できるジョブスロット数（-の場合は無制限）
JL/H	ホストがキューから割り当てられるジョブスロット数。（-の場合は無制限）
NJOBS	現在キューが保持しているジョブスロット数。実行中及び保留されたジョブが使用しているジョブスロット数の合計
PEND	保留中のジョブが使用しているジョブスロット数
RUN	実行中のジョブが使用しているジョブスロット数
SUSPEND	中断ジョブが使用しているジョブスロット数

4.3.4 ジョブの強制終了

ジョブを強制終了するには bkill コマンドを使用します。 ここで指定する JOBID は、前述の bjobs コマンドで確認してください。

```
% bkill  JOBID
```

4.3.5 ジョブの中断

ジョブを中断するには bstop コマンドを使用します。 中断したジョブは、後述の bresume コマンドで再開することができます。

```
% bstop  JOBID
```

4.3.6 ジョブの再開

bstop コマンドで中断されたジョブを再開するには bresume コマンドを使用します。

```
% bresume  JOBID
```

4.3.7 ジョブの中間結果を参照

実行中のジョブの標準出力、標準エラー出力はそのジョブが完了するまで表示されません。bpeek コマンドを使うと、実行中のジョブの各出力を画面に表示することが出来ます。

```
%  bpeek  JOBID

%  bpeek –f JOBID (bpeep の表示を tail –f を使って表示します)
```

4.4　LSF と PBS のコマンド比較

XC4000 で採用されている LSF（Load Sharing Facility）は研究所内の他システムで使用されている PBS とほぼ同じ機能を提供しています。但し PBS と LSF では以下に示すように各機能を提供するコマンドの名前が異なりますので注意してください。
また、当然各コマンドのオプション等も異なるため、両システムの man を見比べるようにしてください。

機能	LSF	PBS
ジョブの投入	bsub	qsub
ジョブ情報の表示	bjobs	qstat
ジョブのキャンセル	bkill	qdel
キュー情報の確認	bqueues	qstat −q
ジョブの途中結果の閲覧	bpeek	該当コマンドなし

そ

5　付録

5.1　付録1　コンパイルから、ジョブ実行までの流れ

Fortran90 のプログラムをコンパイルし、それをジョブとして実行するまでの例を、以下に示します。

① システムにログインしたら、必要なモジュールをロードします。スーパーコンピュータシステムにどのようなモジュールが用意されているかは、module avail と実行してください。MPI のプログラムを作成する場合、mpi モジュールは必須となります。また、利用したいコンパイラに合わせてそれにあったモジュールをロードします。

```
% module avail ←  システムに用意されているモジュールを表示します

 ---------------------------------- /opt/modules/version  ----------------------------------
3.1.6

 ----------------------------- /opt/modules/3.1.6/modulefiles  -----------------------------
dot          module-cvs   module-info modules      null         use.own

 ------------------------------- /opt/modules/modulefiles  -------------------------------
acml/2.7.0/pathscale64             ifort/9.0/default
acml/2.7.0/pathscale64_mp          intel/9.0
acml/2.7.0/pgi64                   mlib/intel/8.1
acml/2.7.0/pgi64_large_arrays      mlib/pgi/5.1
acml/2.7.0/pgi64_mp                mpi/hp/default
acml/2.7.0/pgi64_mp_large_arrays pathscale/2.2.1/default
goto/1.00/default                  pgi/5.1-6/default
hptc                               pgi/5.2-4/default
icc/9.0/default                    pgi/6.0-8/default
idb/9.0/default

% module load mpi ← 並列プログラムを作成するために MPI 用のモジュールをロードします。
%module list ←モジュールが自分の環境にロードされたことを確認します。
Currently Loaded Modulefiles:
  1) mpi/hp/default

% module load ifort/9.0 ← インテル Fortran 用のモジュールをロードします。(この部分は利用するコンパイラに合わせて必要なモジュール名に変更してください)
```

② mpif90（C の場合は mpicc）コマンドを使ってプログラムをコンパイルします。これは各コンパイラのラッパースクリプトであるため、使用するコンパイラのオプションをそのまま指定することができます。またこのスクリプトを使用することにより、並列計算に必要なライブラリ等が全てリンクされます。

```
% mpif90 –o test test.f90
```

③ bsub コマンドを使ってジョブをキューに投入します。並列プログラムでは mpirun に -srun オプションを付けて実行します。以下の例では、キューq8 に 8 並列でジョブを投入しています。ジョブの標準出力、標準エラー出力は、それぞれ test.out test.err というファイルに記録されます。

```
% bsub –q q8 –n 8 –o test.out –e test.err mpirun –srun ./test
```

5.2　付録 2　開発環境

開発言語

■　標準コンパイラ

HP　XC システムでは　標準的な Linux　が提供する各種コンパイラ（(gcc、g++、g77) やライブラリを提供しています。

コンパイラ	バージョン	コマンド
C/C++	3.2.3	gcc
Fortran 77	3.2.3	g77

■　インテルコンパイラ

HP XC システムでは、インテルコンパイラ、Ver8 および 9 を提供しています。

コンパイラ	バージョン	コマンド
C/C++	9.0.021	icc
Fotran	9.0.021	ifort
互換性を考慮し、V7.0 以前のifcコマンドもサポートされています。		

■　PGI コンパイラ

HP XC システムでは、PGI コンパイラ　V6 を提供しています。

コンパイラ	バージョン	コマンド
C	5.1-6, 5.2-4, 6.0-8	pgcc
C++	5.1-6, 5.2-4, 6.0-8	pgCC
Fortran77	5.1-6, 5.2-4, 6.0-8	pgf77
Fortran90	5.1-6, 5.2-4, 6.0-8	pgf90
Fortran95	5.1-6, 5.2-4, 6.0-8	pgf95

■　Pathscale コンパイラ

HP XC システムでは、Pathscale コンパイラ V2 を提供しています。

コンパイラ	バージョン	コマンド
C	2.2.1	pathcc
C++	2.2.1	pathCC
Fortran90	2.2.1	pathf90
Fortran95	2.2.1	pathf95

コンパイル方法

5.2.1 GNU コンパイラ

5.2.1.1 gcc (GNU C)

使い方

```
gcc [ -O レベル] [ -c | -o 出力ファイル名] prog.c [ -lm]
```

主なオプションの説明

オプション	説明
-O レベル	実行時間の最適化レベルの指定。レベルには 0-3 の数字を指定します。レベル 0 は最適化を行いません。数が大きいほど最適化のレベルが上がります。
-c	コンパイルのみを行ない、オブジェクトファイルを作成します。
-g	デバッグを行う際に指定します。これによりデバッグ情報が作成されます。
-o 出力ファイル	出力ファイルの指定。省略すると実行ファイル名が a.out となります。
-lm	数学ライブラリとのリンク
-mcmodel	2GB を超えるデータを扱う場合は –mcmodel=medium を指定します。

コンパイル例

test.c というソースファイルを実行し、test という実行ファイルを生成します。最適化レベルとしては 2 を使用します。

```
gcc  -O2  -o test test.c
```

5.2.1.2 g++ (GNU C++)

使い方

```
g++ [ -O レベル] [ -c | -o 出力ファイル名] prog.cxx [ -lm]
```

主なオプションの説明

オプション	説明
-O レベル	実行時間の最適化レベルの指定。レベルには 0-3 の数字を指定します。レベル 0 は最適化を行いません。数が大きいほど最適化のレベルが上がります。
-c	コンパイルのみを行ない、オブジェクトファイルを作成します。
-g	デバッグを行う際に指定します。これによりデバッグ情報が作成されます。
-o 出力ファイル	出力ファイルの指定。省略すると実行ファイル名が a.out となります。
-lm	数学ライブラリとのリンク
-mcmodel	2GB を超えるデータを扱う場合は –mcmodel=medium を指定します。

コンパイル例

test.cxx というソースファイルを実行し、test という実行ファイルを生成します。最適化レベルとしては 2 を使用します。

```
g++  -O2  -o test test.cxx
```

5.2.1.3 g77 (GNU Fortran77)

使い方

```
g77 [ -O レベル] [ -c | -o 出力ファイル名] prog.f [ -lm]
```

主なオプションの説明

オプション	説明
-O レベル	実行時間の最適化レベルの指定。レベルには 0 -3 の数字を指定します。レベル 0 は最適化を行いません。数が大きいほど最適化のレベルが上がります。
-c	コンパイルのみを行ない、オブジェクトファイルを作成します。
-g	デバッグを行う際に指定します。これによりデバッグ情報が作成されます。
-o 出力ファイル	出力ファイルの指定。省略すると実行ファイル名が a.out となります。
-lm	数学ライブラリとのリンク

コンパイル例

test.f というソースファイルを実行し、test という実行ファイルを生成します。最適化レベルとしては 2 を使用します。

```
g77  -O2  -o test test.f
```

5.2.2 Intel コンパイラ

5.2.2.1 Intel Fortran

使い方

Intel Fortran を実行する前に、”module load ifort/9.0“ を実行して、実行環境をロードしてください。

```
ifort [ -O レベル] [option] [ -c  |  -o 出力ファイル] prog.f
```

*Intel Fortran7.X 以前で使用されていた ifc コマンドも互換性のため用意されています。

主なオプションの説明

オプション	説明
-O レベル	レベルには 0-3 の数字を指定します。レベル 0 は最適化を行いません。数が大きいほど最適化のレベルが上がります。 デフォルトの最適化レベルとして 2 が使用されます。 また -g を指定した場合は最適化レベルは 0 がデフォルトになります。
-fast	いくつかの最適化オプションを一まとめにして設定します。 ifort では、 -fast を設定すると、 -ipo, -O3, –no-prec-div, -static, xP が指定されます。
-c	コンパイルのみを行ない、オブジェクトファイルを作成します。
-g	デバッグを行う際に指定します。これによりデバッグ情報が作成されます。
-o 出力ファイル	出力ファイルの指定。省略すると実行ファイル名が a.out となります。
-lm	数学ライブラリとのリンク
-parallel	自動並列化（マルチスレッド化）されたプログラムを生成します。
-par_report{0 |1 |2 |3}	自動並列化時の診断レポートのレベルを指定します。(0：診断情報を表示しない、1（デフォルト）：正常に並列化されたループの表示、2：ループの並列化が成功・不成功を表示、3：並列化の妨げになると考えられる依存関係の表示)
-par_threshld[n]	ループの並列化による効果が現れる確率に基づいてループの自動並列化のしきい値を設定します （ n=0 から n= 100 。デフォルト ： n=75 ）。 このオプションは、コンパイル時に計算量が確定できないループに使用します。 0 - 計算量にかかわらず並列化を行います。 100 - ループは並列実行が有効であることが確実な場合にのみ

	並列化されます。

コンパイル例

test.f というソースファイルをコンパイルし、test という実行ファイルを生成します。最適化レベルとしては 2 を使用します。

```
ifort -O2 -o test test.f
```

test.f というソースファイルを自動並列化します。

```
ifort –parallel –per-report3 –par-threshold0 –O3 test.f
```

5.2.2.2 Intel C++

使い方

Intel Fortran を実行する前に、”module load icc/9.0“ を実行して、実行環境をロードしてください。

```
icc [ -O レベル] [option] [ -c | -o 出力ファイル] prog.c
```

主なオプションの説明

オプション	説明
-O レベル	レベルには 0 -3 の数字を指定します。レベル 0 は最適化を行いません。数が大きいほど最適化のレベルが上がります。 デフォルトの最適化レベルとして 2 が使用されます。 また -g を指定した場合は最適化レベルは 0 がデフォルトになります。
-fast	いくつかの最適化オプションを一まとめにして設定します。 ifort では、 -fast を設定すると、 -ipo, -O3, –no -prec -div, -static, xP が指定されます。
-c	コンパイルのみを行ない、オブジェクトファイルを作成します。
-g	デバッグを行う際に指定します。これによりデバッグ情報が作成されます。
-o 出力ファイル	出力ファイルの指定。省略すると実行ファイル名が a.out となります。
-lm	数学ライブラリとのリンク
-parallel	自動並列化（マルチスレッド化）されたプログラムを生成します。
-par_report{0 |1 |2 |3}	自動並列化時の診断レポートのレベルを指定します。(0：診断情報を表示しない、1（デフォルト）：正常に並列化されたループの表示、2：ループの並列化が成功・不成功を表示、3：並列化の妨げになると考えられる依存関係の表示)
-par_threshld[n]	ループの並列化による効果が現れる確率に基づいてループの自動並列化のしきい値を設定します （ n=0 から n= 100 。デフォルト ： n=75 ）。 このオプションは、コンパイル時に計算量が確定できないループに使用します。 0 - 計算量にかかわらず並列化を行います。 100 - ループは並列実行が有効であることが確実な場合にのみ並列化されます。
-openmp	OpenMP ディレクティブを用いた 並列プログラムを並列コンパイルする場合に使用します。

コンパイル例

test.c というソースファイルをコンパイルし、test という実行ファイルを生成します。最適化レベルとしては 2 を使用します。

```
icc -O2 -o test test.c
```

test.c というソースファイルを自動並列化します。

```
icc -parallel -par-report3 -par-threshold0 -O3 test.c
```

5.2.3 PGI コンパイラ

5.2.3.1 pgf77 (PGI Fortran 77)

使い方

PGI Fortran を実行する前に、"module load pgi/6.0-8" を実行して、実行環境をロードしてください。

```
pgf77 [-On | -fast] [option] [-c | -o outfile] prog.f
```

主なオプションの説明

オプション	説明
-O レベル	レベルには 0-3 の数字を指定します。レベル 0 は最適化を行いません。数が大きいほど最適化のレベルが上がります。 -O をつけなかった場合、自動的に -O1 が使用されます。また -O のみでレベルを指定しなかった場合は -O2 が使用されます。
-fast	コンパイラオプションとして、 -O2 -Munroll=c:1 -Mnoframe –Mlre を指定したのと同じになります。
-fastsse	SSE/SSE2 機能を備えた CPU において、SSE のベクトル化機能を使用します。
-c	コンパイルのみを行ない、オブジェクトファイルを作成します。
-g	デバッグを行う際に指定します。これによりデバッグ情報が作成されます。
-o 出力ファイル	出力ファイルの指定。省略すると実行ファイル名が a.out となります。
-mcmodel={small medium}	メモリモデルを指定します。2GB 以上のメモリを使用する場合は、 -mcmodl=large を指定してください。
-Mlarge_array	単一オブジェクト（配列等）が 2GB を超える場合、さらに、そのインデックス計算においても 64 ビット整数空間のサイズで計算されます。
-i8	2GB を超えるオブジェクトの配列変数のインデックスをプログラム内部で使用している場合には、必須となります。従来のデフォルトの INTEGER*4 では、2GB 以上のアドレス値を処理できないため、8 バイト整数に暗黙的にコンパイラが置き換えてコードを生成します。
-Mconcur	-Mconcur オプションは、プログラム内のループレベルの並列性を抽出し、自動並列化を行うためのオプションです。
-mp	OpenMP ディレクティブを用いた 並列プログラムを並列コンパイルする場合に使用します。

コンパイル例

test.f というソースファイルをコンパイルし、test という実行ファイルを生成します。最適化レベルとしては 2 を使用します。

```
pgf77 -O2 -o test test.f
```

test.f というソースファイルを自動並列化します。

```
pgf77 –Mconcur test.f
```

test.f というソースファイルを 2GB 以上使用プログラムとして生成します。

```
pgf77 –mcmodel=medium –i8 –Mlarge_array test.f（V6.0 以前）
pgf77 –mcmodel=medium –i8 test.f（V6.0 以降）
```

5.2.3.2 pgf90 (PGI Fortran 90)

使い方

PGI Fortran を実行する前に、”module load pgi/6.0-8“ を実行して、実行環境をロードしてください。

```
pgf90 [-On | -fast] [option] [-c | -o outfile] prog.f90
```

主なオプションの説明

オプション	説明
-O レベル	レベルには 0-3 の数字を指定します。レベル 0 は最適化を行いません。数が大きいほど最適化のレベルが上がります。 -O をつけなかった場合、自動的に -O1 が使用されます。また -O のみでレベルを指定しなかった場合は -O2 が使用されます。
-fast	コンパイラオプションとして、 -O2 -Munroll=c:1 -Mnoframe –Mlre を指定したのと同じになります。
-fastsse	SSE/SSE2 機能を備えた CPU において、SSE のベクトル化機能を使用します。
-c	コンパイルのみを行ない、オブジェクトファイルを作成します。
-g	デバッグを行う際に指定します。これによりデバッグ情報が作成されます。
-o 出力ファイル	出力ファイルの指定。省略すると実行ファイル名が a.out となります。
-mcmodel={small |medium}	メモリモデルを指定します。2GB 以上のメモリを使用する場合は、 -mcmodl=large を指定してください。
-Mlarge_array	単一オブジェクト（配列等）が 2GB を超える場合、さらに、そのインデックス計算においても 64 ビット整数空間のサイズで計算されます。このオプションは PGI コンパイラ 6.0 以降では必要なくなりましたが、それ以前のバージョンには必ず指定してください。
-i8	2GB を超えるオブジェクトの配列変数のインデックスをプログラム内部で使用している場合には、必須となります。従来のデフォルトの INTEGER*4 では、2GB 以上のアドレス値を処理できないため、8 バイト整数に暗黙的にコンパイラが置き換えてコードを生成します。
-Mconcur	-Mconcur オプションは、プログラム内のループレベルの並列性を抽出し、自動並列化を行うためのオプションです。
-mp	OpenMP ディレクティブを用いた 並列プログラムを並列コンパイルする場合に使用します。

コンパイル例

test.f90 というソースファイルをコンパイルし、test という実行ファイルを生成します。最適化レベルとしては 2 を使用します。

```
pgf90 -O2 -o test test.f90
```

test.f90 というソースファイルを自動並列化します。

```
pgf90 –Mconcur test.f90
```

test.f90 というソースファイルを 2GB 以上使用プログラムとして生成します。

```
pgf90 –mcmodel=medium –i8 –Mlarge_array test.f90   (V6.0 以前)
pgf90 –mcmodel=medium –i8 test.f90   (V6.0 以降)
```

5.2.3.3 pgf95 (PGI Fortran 95)

使い方

PGI Fortran を実行する前に、”module load pgi/6.0-8“ を実行して、実行環境をロードしてください。

```
pgf95 [ -On | -fast] [option] [ -c  | -o outfile] prog.f95
```

主なオプションの説明

オプション	説明
-O レベル	レベルには 0-3 の数字を指定します。レベル 0 は最適化を行いません。数が大きいほど最適化のレベルが上がります。 -O をつけなかった場合、自動的に -O1 が使用されます。また -O のみでレベルを指定しなかった場合は -O2 が使用されます。
-fast	コンパイラオプションとして、 -O2 -Munroll=c:1 -Mnoframe –Mlre を指定したのと同じになります。
-fastsse	SSE/SSE2 機能を備えた CPU において、SSE のベクトル化機能を使用します。
-c	コンパイルのみを行ない、オブジェクトファイルを作成します。
-g	デバッグを行う際に指定します。これによりデバッグ情報が作成されます。
-o 出力ファイル	出力ファイルの指定。省略すると実行ファイル名が a.out となります。
-mcmodel={small |medium}	メモリモデルを指定します。2GB 以上のメモリを使用する場合は、 -mcmodl=large を指定してください。
-Mlarge_array	単一オブジェクト（配列等）が 2GB を超える場合、さらに、そのインデックス計算においても 64 ビット整数空間のサイズで計算されます。このオプションは PGI コンパイラ 6.0 以降は必要なくなりましたが、それ以前のバージョンには必ず指定してください。
-i8	2GB を超えるオブジェクトの配列変数のインデックスをプログラム内部で使用している場合には、必須となります。従来のデフォルトの INTEGER*4 では、2GB 以上のアドレス値を処理できないため、8 バイト整数に暗黙的にコンパイラが置き換えてコードを生成します。
-Mconcur	-Mconcur オプションは、プログラム内のループレベルの並列性を抽出し、自動並列化を行うためのオプションです。
-mp	OpenMP ディレクティブを用いた 並列プログラムを並列コンパイルする場合に使用します。

コンパイル例

test.f95 というソースファイルをコンパイルし、test という実行ファイルを生成します。最適化レベルとしては 2 を使用します。

```
pgf95 -O2 -o test test.f95
```

test.f95 というソースファイルを自動並列化します。

```
pgf95 –Mconcur test.f95
```

test.f95 というソースファイルを 2GB 以上使用プログラムとして生成します。

```
pgf95 –mcmodel=medium –i8 –Mlarge_array test.f95(V6.0 以前)
pgf95 –mcmodel=medium –i8 test.f95 (V6.0 以降)
```

5.2.3.4 pgcc (PGI C)

使い方

PGI C

を実行する前に、”module load pgi/6.0 -8 “ を実行して、実行環境をロードしてください。

```
pgcc [ -On | -fast] [option] [ -c  | -o outfile] prog.c
```

主なオプションの説明

オプション	説明
-O レベル	レベルには 0 -3 の数字を指定します。レベル 0 は最適化を行いません。数が大きいほど最適化のレベルが上がります。 -O をつけなかった場合、自動的に -O1 が使用されます。また -O のみでレベルを指定しなかった場合は -O2 が使用されます。
-fast	コンパイラオプションとして、 -O2 -Munroll=c:1 -Mnoframe –Mlre を指定したのと同じになります。
-fastsse	SSE/SSE2 機能を備えた CPU において、SSE のベクトル化機能を使用します。
-c	コンパイルのみを行ない、オブジェクトファイルを作成します。
-g	デバッグを行う際に指定します。これによりデバッグ情報が作成されます。
-o 出力ファイル	出力ファイルの指定。省略すると実行ファイル名が a.out となります。
-mcmodel={small |medium}	メモリモデルを指定します。2GB 以上のメモリを使用する場合は、 -mcmodl=large を指定してください。
-Mlarge_array	単一オブジェクト（配列等）が 2GB を超える場合、さらに、そのインデックス計算においても 64 ビット整数空間のサイズで計算されます。このオプションは PGI コンパイラ 6.0 以降では必要なくなりましたが、それ以前のバージョンには必ず指定してください。
-Mconcur	-Mconcur オプションは、プログラム内のループレベルの並列性を抽出し、自動並列化を行うためのオプションです。
-mp	OpenMP ディレクティブを用いた 並列プログラムを並列コンパイルする場合に使用します。

コンパイル例

test.c というソースファイルをコンパイルし、test という実行ファイルを生成します。最適化レベルとしては 2 を使用します。

```
pgcc -O2 -o test test.c
```

test.c というソースファイルを自動並列化します。

```
pgcc –Mconcur test.c
```

test.c というソースファイルを 2GB 以上使用プログラムとして生成します。

```
pgcc –mcmodel=medium test.c
```

5.2.3.5 pgCC (PGI C++)

使い方

PGI Fortran を実行する前に、”module load pgi/6.0-8“ を実行して、実行環境をロードしてください。

```
pgCC [ -On | -fast] [option] [ -c  | -o outfile] prog.f95
```

主なオプションの説明

オプション	説明
-O レベル	レベルには 0-3 の数字を指定します。レベル 0 は最適化を行いません。数が大きいほど最適化のレベルが上がります。 -O をつけなかった場合、自動的に -O1 が使用されます。また -O のみでレベルを指定しなかった場合は -O2 が使用されます。
-fast	コンパイラオプションとして、 -O2 -Munroll=c:1 -Mnoframe –Mlre を指定したのと同じになります。
-fastsse	SSE/SSE2 機能を備えた CPU において、SSE のベクトル化機能を使用します。
-c	コンパイルのみを行ない、オブジェクトファイルを作成します。
-g	デバッグを行う際に指定します。これによりデバッグ情報が作成されます。
-o 出力ファイル	出力ファイルの指定。省略すると実行ファイル名が a.out となります。
-mcmodel={small |medium}	メモリモデルを指定します。2GB 以上のメモリを使用する場合は、 -mcmodl=large を指定してください。
-Mlarge_array	単一オブジェクト（配列等）が 2GB を超える場合、さらに、そのインデックス計算においても 64 ビット整数空間のサイズで計算されます。このオプションは PGI コンパイラ 6.0 以降では必要なくなりましたが、それ以前のバージョンには必ず指定してください。
-Mconcur	-Mconcur オプションは、プログラム内のループレベルの並列性を抽出し、自動並列化を行うためのオプションです。
-mp	OpenMP ディレクティブを用いた 並列プログラムを並列コンパイルする場合に使用します。

コンパイル例

test.c というソースファイルをコンパイルし、test という実行ファイルを生成します。最適化レベルとしては 2 を使用します。

```
pgCC  -O2  -o test test.c
```

test.CC というソースファイルを自動並列化します。

```
pgcc –Mconcur test.c
```

test.c というソースファイルを 2GB 以上使用プログラムとして生成します。

```
pgCC –mcmodel=medium test.c
```

5.2.4 Pathscale コンパイラ

5.2.4.1 pathcc (Pathscale C)

使い方

Pathscale コンパイラを実行する前に、”module load pathscale“ を実行して、実行環境をロードしてください。

```
pathcc [ -On | -fast] [option] [ -c  | -o outfile] prog.c
```

主なオプションの説明

オプション	説明
-O レベル	レベルには 0 -3 の数字を指定します。レベル 0 は最適化を行いません。数が大きいほど最適化のレベルが上がります。 -O を付けなくても、デフォルトで -O2 が使用されます。
-show -defaults	コンパイラのデフォルトオプションを表示します。
-Ofast	コンパイラオプションとして、 -O3 -ipa –OPT:Ofast –fno -math -errno を指定したのと同じになります。
-c	コンパイルのみを行ない、オブジェクトファイルを作成します。
-g	デバッグを行う際に指定します。これによりデバッグ情報が作成されます。
-o 出力ファイル	出力ファイルの指定。省略すると実行ファイル名が a.out となります。
-mcmodel={small medium}	メモリモデルを指定します。2GB 以上のメモリを使用する場合は、 -mcmodl=large を指定してください。
-apo	プログラムの自動並列化を行います。
-mp	OpenMP ディレクティブを用いた 並列プログラムを並列コンパイルする場合に使用します。

コンパイル例

test.c というソースファイルをコンパイルし、test という実行ファイルを生成します。最適化レベルとしては 2 を使用します。

```
pathcc  -O2  -o test test.c
```

test.c というソースファイルを 2GB 以上使用プログラムとして生成します。

```
pathcc –mcmodel=medium test.c
```

5.2.4.2 pathCC (Pathscale C++)

使い方

Pathscale コンパイラを実行する前に、”module load pathscale“ を実行して、実行環境をロードしてください。

```
pathCC [ -On | -fast] [option] [ -c  | -o outfile] prog.c
```

主なオプションの説明

オプション	説明
-O レベル	レベルには 0 -3 の数字を指定します。レベル 0 は最適化を行いません。数が大きいほど最適化のレベルが上がります。 -O を付けなくても、デフォルトで -O2 が使用されます。
-show -defaults	コンパイラのデフォルトオプションを表示します。
-Ofast	コンパイラオプションとして、 -O3 -ipa –OPT:Ofast –fno -math -errno を指定したのと同じになります。
-c	コンパイルのみを行ない、オブジェクトファイルを作成します。
-g	デバッグを行う際に指定します。これによりデバッグ情報が作成されます。
-o 出力ファイル	出力ファイルの指定。省略すると実行ファイル名が a.out となります。
-mcmodel={small medium}	メモリモデルを指定します。2GB 以上のメモリを使用する場合は、 -mcmodl=large を指定してください。
-apo	プログラムの自動並列化を行います。
-mp	OpenMP ディレクティブを用いた 並列プログラムを並列コンパイルする場合に使用します。

コンパイル例

test.c というソースファイルをコンパイルし、test という実行ファイルを生成します。最適化レベルとしては 2 を使用します。

```
pathCC  -O2  -o test test.c
```

test.c というソースファイルを 2GB 以上使用プログラムとして生成します。

```
pathCC –mcmodel=medium test.c
```

5.2.4.3 pathf90 (Pathscale Fortran90)

使い方

Pathscale コンパイラを実行する前に、”module load pathscale“ を実行して、実行環境をロードしてください。

```
pathf90 [ -On | -fast] [option] [ -c  | -o outfile] prog.f90
```

主なオプションの説明

オプション	説明
-O レベル	レベルには 0 -3 の数字を指定します。レベル 0 は最適化を行いません。数が大きいほど最適化のレベルが上がります。 -O を付けなくても、デフォルトで -O2 が使用されます。
-show -defaults	コンパイラのデフォルトオプションを表示します。
-Ofast	コンパイラオプションとして、 -O3 -ipa –OPT:Ofast –fno -math -errno を指定したのと同じになります。
-c	コンパイルのみを行ない、オブジェクトファイルを作成します。
-g	デバッグを行う際に指定します。これによりデバッグ情報が作成されます。
-o 出力ファイル	出力ファイルの指定。省略すると実行ファイル名が a.out となります。
-mcmodel={small |medium}	メモリモデルを指定します。2GB 以上のメモリを使用する場合は、 -mcmodl=large を指定してください。
-apo	プログラムの自動並列化を行います。
-mp	OpenMP ディレクティブを用いた 並列プログラムを並列コンパイルする場合に使用します。

コンパイル例

test.c というソースファイルをコンパイルし、test という実行ファイルを生成します。最適化レベルとしては 2 を使用します。

```
pathf90  -O2  -o test test.f90
```

test.c というソースファイルを 2GB 以上使用プログラムとして生成します。

```
pathf90 –mcmodel=medium test.f90
```

5.2.4.4 pathf95 (Pathscale Fortran95)

使い方

Pathscale コンパイラを実行する前に、”module load pathscale“ を実行して、実行環境をロードしてください。

```
pathf95 [ -On | -fast] [option] [ -c  | -o outfile] prog.f95
```

主なオプションの説明

オプション	説明
-O レベル	レベルには 0 -3 の数字を指定します。レベル 0 は最適化を行いません。数が大きいほど最適化のレベルが上がります。 -O を付けなくても、デフォルトで -O2 が使用されます。
-show -defaults	コンパイラのデフォルトオプションを表示します。
-Ofast	コンパイラオプションとして、 -O3 -ipa –OPT:Ofast –fno -math -errno を指定したのと同じになります。
-c	コンパイルのみを行ない、オブジェクトファイルを作成します。
-g	デバッグを行う際に指定します。これによりデバッグ情報が作成されます。
-o 出力ファイル	出力ファイルの指定。省略すると実行ファイル名が a.out となります。
-mcmodel={small |medium}	メモリモデルを指定します。2GB 以上のメモリを使用する場合は、 -mcmodl=large を指定してください。
-apo	プログラムの自動並列化を行います。
-mp	OpenMP ディレクティブを用いた 並列プログラムを並列コンパイルする場合に使用します。

コンパイル例

test.c というソースファイルをコンパイルし、test という実行ファイルを生成します。最適化レベルとしては 2 を使用します。

```
pathf95  -O2  -o test test.f95
```

test.c というソースファイルを 2GB 以上使用プログラムとして生成します。

```
pathf95 –mcmodel=medium test.f95
```

5.3　付録3　ドキュメント

XC4000 にインストールされている各種システムソフトウェア（コンパイラ、ライブラリ等）のマニュアルは、/home/doc/ismxc の下にあります。
閲覧には PDF ビューア（Adobe Reader）等が必要ですので、各自の PC にダウンロードして御利用ください。

ディレクトリ名	説明
ACML	AMD が提供する ACML 数値演算ライブラリ付属のドキュメント
HP-MLIB	XC4000 で使用される MPI(HP MPI)付属のドキュメント
Intel_Compiler	インテル･コンパイラ付属のドキュメント
LSF	バッチ管理システム（Load Sharing Facility）付属のドキュメント
MLIB	HP MLIB 数値演算ライブラリ付属のドキュメント
Nagios	XC4000 のシステム管理・監視ツール（Nagios）の付属ドキュメント
Pathscale	Pathscale コンパイラ付属のドキュメント
PGI	PGI コンパイラ付属のドキュメント
Random	物理乱数発生プログラムの付属ドキュメント
SLURM	XC4000 のリソース管理ユーティリティ SLURM の付属っドキュメント
XC_Software	XC4000 付属のユーザガイド、システム管理者ガイド

5.4　御質問について

XC4000 システムの利用に関する御質問及びトラブル等の障害については、以下のメーリングリストまで御連絡ください。

統計数理研究所　XC4000 サポート用メーリングリスト
ism-xc@sgi.co.jp