統計数理研究所

統計科学スーパーコンピュータシステム

並列計算機サブシステム利用の手引き

第 1.2 版

改訂履歴

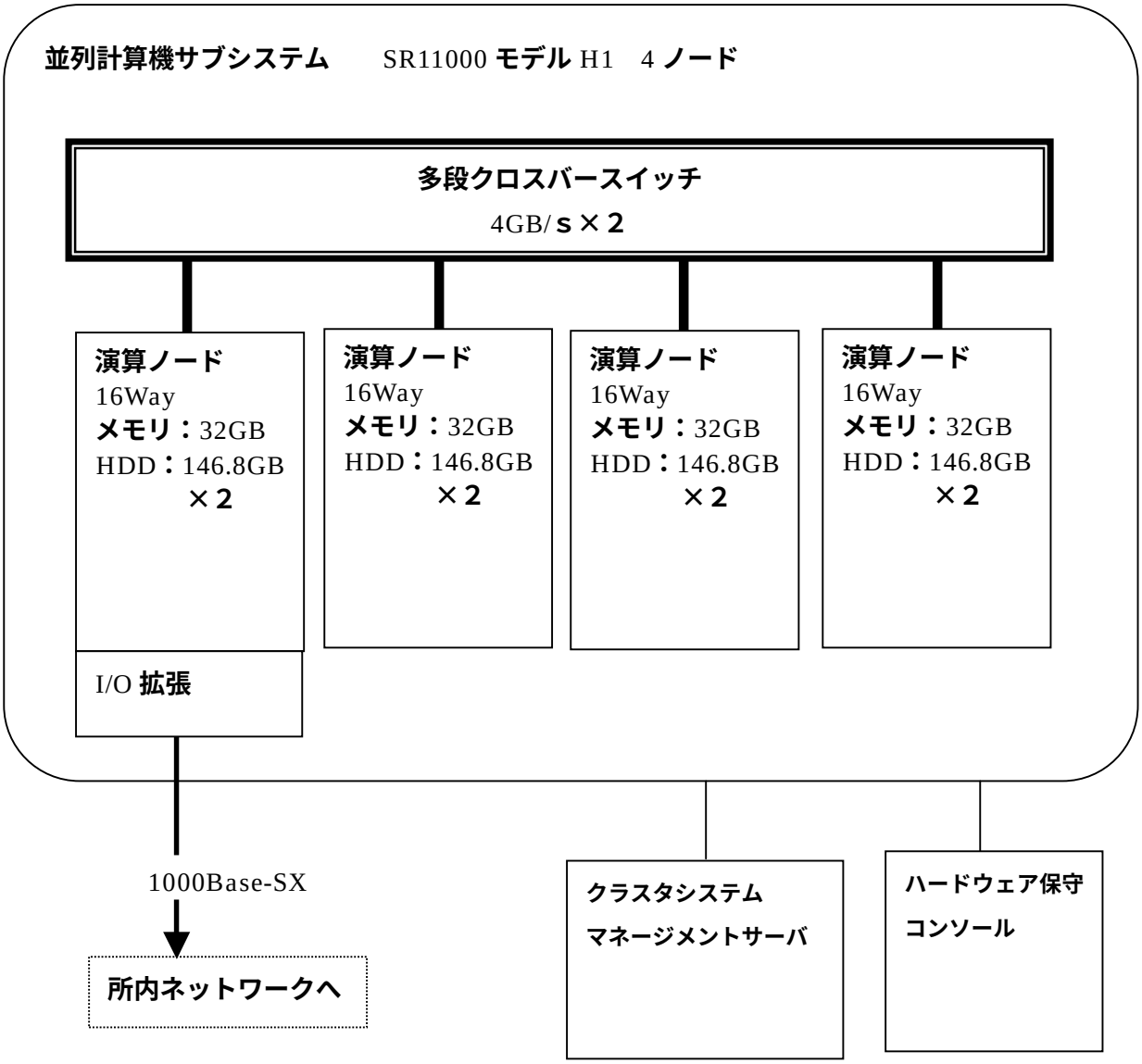
版数	年月日	該当項目	概要説明
1.0	2004.02.27	全項目	新規作成
1.1	2004.03.12		ライブラリ指定方法変更
1.2	2004.06.28		要素並列を使用しないノード内 MPI の利用方法 ノード内要素並列と MPI の組み合わせ利用方法追加

― 目次 ―

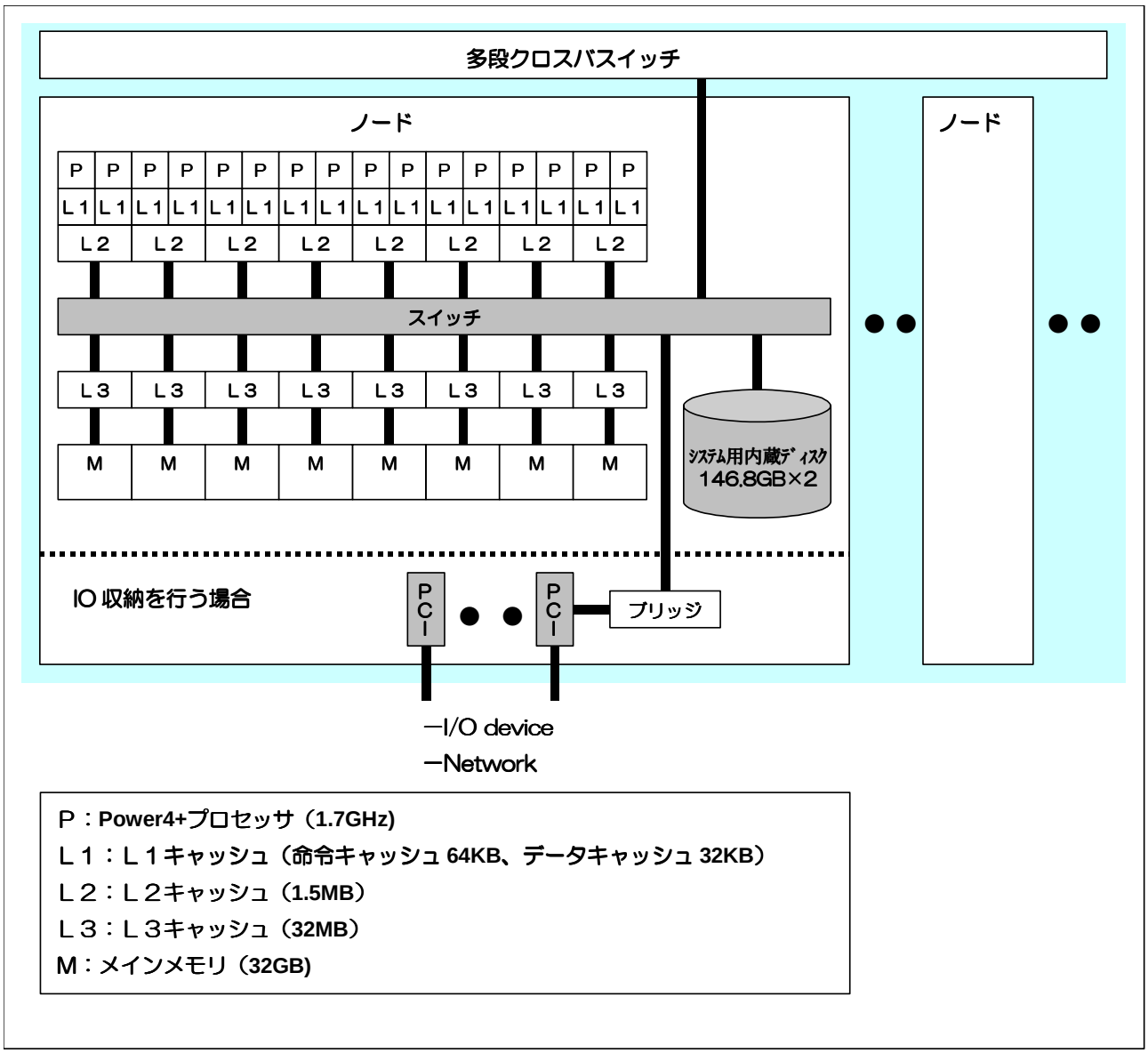
1. システム構成

1.1 システム構成図

（1）システム全体構成図



（2）演算ノード構成図



1.2 ソフトウェア構成

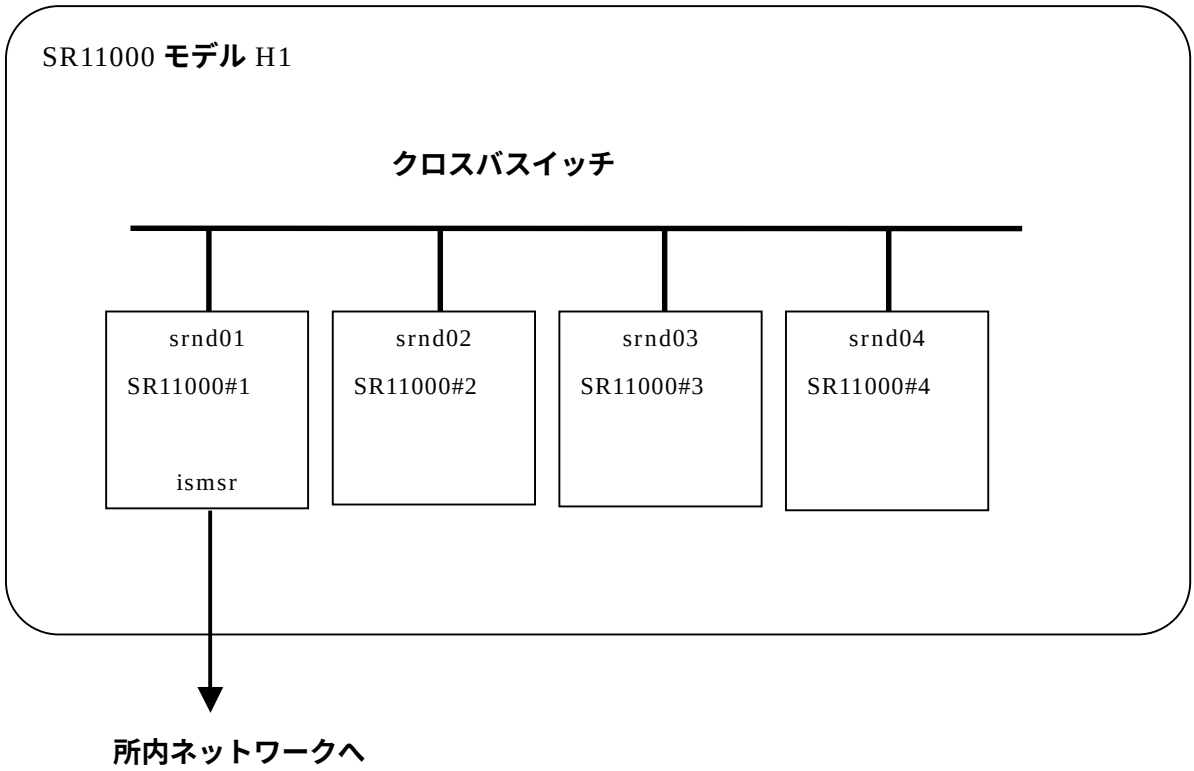
プロダクト名	機能
AIX 5L 関連	OS・クラスタシステム基本ソフトウェア
LoadLeveler	バッチジョブ実行管理
Parallel Environment	並列プログラム、開発・実効環境
最適化 FORTRAN90	Fortran コンパイラ
C for AIX	C コンパイラ
MATRIX/MPP	行列計算用数値計算ライブラリ
MSL2	数値計算副プログラムライブラリ

1.3. ネットワーク構成

（1）IP アドレス一覧

機器名	ホスト名	IP アドレス	用途
SR11000 #1	ismsr srnd01	133.58.226.111 133.58.228.111	ユーザログイン、ジョブ投入
SR11000 #2	srnd02	133.58.228.112	ジョブ実行
SR11000 #3	srnd03	133.58.228.113	ジョブ実行
SR11000 #4	srnd04	133.58.228.114	ジョブ実行

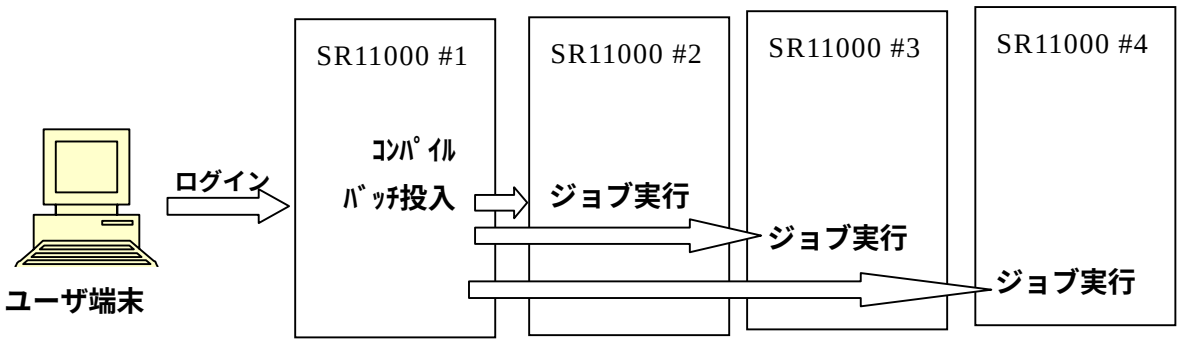
（2）論理ネットワーク構成図



2. システム利用

2.1.システム利用イメージ

利用者は、SR11000#１（ismsr.ism.ac.jp)にログインし、プログラム開発、バッチジョブの投入を行います。



2.2. ログイン方法

プログラムの作成、コンパイル、バッチジョブ投入などは以下のホストにログインして行います。

ホスト名	IP アドレス
ismsr.ism.ac.jp	133.58.226.111

telnet、ftp、ssh を利用してアクセスできます。

基本的な UNIX コマンド及び、エディタとして vi 、emacs が利用可能です。

言語環境は、環境変数“LANG”により設定されます。

LANG	言語
ja_JP	日本語（EUC code)
Ja_JP	日本語（SJIS code)
C	7 ビット ASCII 文字セット

2.3. ファイルシステム

一般ユーザが利用可能なファイルシステムは以下のとおりです。

名称	マウントポイント	利用目的・備考
HOME 領域	/home0	・ユーザホームディレクトリ
(NFS)	/home1	・ファイルサーバ(ismorg)のディスクを NFS マウントした領域

3. 開発環境

3.1. ツール一覧

コマンド名	内容
f90	Fortran 言語で書かれたプログラムをコンパイルする
mpif90_r	MPI 並列化された Fortran プログラムをコンパイルする
xlc	C 言語で書かれたプログラムをコンパイルする
mpcc_r	MPI 並列化された C プログラムをコンパイルする

3.2. Fortran コンパイラ

(1)利用方法

プログラムをコンパイルリンケージする例

```
%  f90   [オプション]  プログラムファイル名
```

MPI 並列化されたプログラムをコンパイルする例

```
%  mpif90_r   [オプション]  プログラムファイル名
```

（2）主なオプション(man f90 コマンドで詳細な説明が参照できます)

コンパイラオプション	内容	
-o *ファイル名*	指定された*ファイル名*で実行ロードモジュールを作成します。 省略した場合、a.out が作成されます。	
-O*x* (デフォルト：-Os)	最適化オプション(大文字の O)	
	x=0	(数字の 0）最適化を行いません。デバッグ時に使用
	x=3	演算順序を変更しない範囲で最適化を行います。
	x=4	制御構造変換、演算順序の変換などプログラム全体にわたる最適化を行います。
	x=s	実行スピードが最も速くなるようコンパイルオプションを自動的に設定します。
-omp	OpenMP 指示行を有効にします。	
-parallel[=N]	ノード内自動並列化オプション N=0~4 で自動並列化レベルを指定する。N=0 は、自動並列を行わない。	
-64	64 ビットアドレッシングモードのオブジェクト及びロードモジュールを作成します。 （デフォルトは 32 ビットアドレッシングモード）	
-loglist	性能チューニング用最適化ログを、.log ファイルに出力します。	

3.3. C コンパイラ

（1）利用方法

プログラムをコンパイルリンケージする例

```
%  xlc   [オプション]  プログラムファイル名
```

MPI 並列化されたプログラムをコンパイルする例

```
%  mpcc_r   [オプション]  プログラムファイル名
```

（2）主なオプション（オプション無しの xls コマンドで詳細な説明が参照できます）

コンパイラオプション	内容
-o *ファイル名*	指定された*ファイル名*で実行ロードモジュールを作成します。 省略した場合、a.out が作成されます。
-q64	64 ビットのコンパイラモードを選択する。 （デフォルトは 32 ビットモード）
-O または -O*n*	指定した最適化レベル（*n=2,3,4,5* ）でコンパイルする。 *n* を省略した場合 *n=2* と同じ
-qsmp=auto	プログラムの自動並列化（SMP ノード内並列）を行う。
-qsmp=omp	OpenMP 指示行による並列化を行います。自動並列化は使用不可となります。

3.4. MSL２の使い方

```
%  f90   [オプション]  プログラムファイル名 -lMSL2
```

3.5. MATRIX/MPP の使い方

```
%  f90   [オプション]  プログラムファイル名 -lmatmpp
```

3.6. 時間計測関数

（1） Fortran **関数（**xclock)

xclock **関数は時刻、経過時間、及び** CPU **時間が測定できる。どの時間を計測するかは、第** 2 **引数** q **で指定する**

使用方法：call　xclock(p,q)

引数の形式	機能
q=1	**サブルーチンが引用されたときの時刻が、**p **の左端から** 12 **バイトの領域に文字列で与えられる。文字列の形式は以下の通り。** HH△MM△SS△△△△（HH：**時、**MM：**分、**SS：**秒、**△：**空白**）
q=2	**サブルーチンが引用されたときの時刻が、整数型** 4 **バイトの形式で** p **に与えられる。単位は秒である。**
q=3	**インタバルタイマ（**CPU **時間測定用タイマ）を起動する。**p **の値は変更しない。**
q=4	**インタバルタイマ起動後の** CPU **時間が、実数型** 8 **バイトの形式で** p **に与えられる。値** 1.0 **は、**1 **秒に相当する。インタバルタイマの値は更新される。**
q=5	q=4 **を指定した時と同じ。ただし、インタバルタイマの値は更新されない。**q=3 **で呼び出された時点からの継続した値となる。**
q=6	CPU **時間最大値からの残り時間が、実数型** 8 **バイト形式で** p **に与えられる。値** 1.0 **は、**1 **秒に相当する。**
q=7	**経過時間計測タイマを起動する。**p **の値は変更しない。**
q=8	**経過時間計測タイマ起動後の経過時間が実数型** 8 **バイト形式で** p **に与えられる。経過時間計測タイマは起動時（**q=7**）からの継続した値となる。**

使用例）

```
% cat sample.f
      program sample
      real*8      t1,t2,dummy
      call xclock(dummy,7)          経過時間計測タイマ起動……①
        call sub1( )
      call xclock(t1,8)             ①からの経過時間を t1 に格納
      print   *,t1                  sub1( )の経過時間を印刷
        call sub2( )
      call xclock(t2,8)             ①からの経過時間を t2 に格納
      print   *,t2 - t1             sub2( )の経過時間を印刷
      end
```

（２）C 関数（gettimeofday**）**

gettimeofday **は、**1970 **年** 1 **月** 1 **日** 00:00:00 **からの経過時間を秒数とマイクロ秒数で表します。**

返り値は、実行に成功すれば 0**、失敗すれば－**1 **を返します。**

```
#include <sys/time.h>
int gettimeofday ( Tp, Tzp)
struct timeval *Tp;
void *Tzp;
```

＊timeval **構造体は、**/usr/include/sys/time.h **で定義されています。**

```
struct timeval {
        time_t          tv_sec;         /* seconds */
        long            tv_usec;        /* microseconds */
};
```

使用例）

```
#include<stdio.h>
#include<stdlib.h>
#include<sys/time.h>
int main(void)
{
   int i;
   float s=0;
   double ts, te;
   double elapsed();
   ts=elapsed();
   for(i=0;i<1000000;i++);
   s=s+(float)i;
   te=elapsed();
   printf("%10.6e¥n",te-ts);
   exit(EXIT_SUCCESS);
}
double elapsed()
{
   struct timeval tp;
   void *tzp;
   gettimeofday(&tp,tzp);
   return ( (double) tp.tv_sec + (double) tp.tv_usec * 1.e-6 );
}
```

4. プログラム実行方法

4.1. LoadLeveler の概要

並列計算機サブシステムのバッチジョブソフトウェアは、LoadLeveler V3.2 を使用して運用しています。バッチジョブを投入する場合、JCF（ジョブコマンドファイル）を作成し、実行プログラム、実行条件を指定してサブミットします。

LoadLeveler の主なコマンド

コマンド名	機能	使用方法
llcancel	ジョブの取り消し	llcancel <option> <joblist>
llclass	利用可能なジョブクラスの表示	llclass <option> <class name>
llq	現在キューイング及び実行しているジョブの表示	llq <option> <job number/name>
llstatus	クラスタ内のマシン状況に関する情報を表示	llstatus <option> <host>
llsubmit	バッチジョブをサブミットする	llsubmit <option> <JCF filename>

4.2. LoadLeveler キュー構成

本システムに設定されているバッチジョブクラスは、下表のとおりです。

#	クラス名	用途	最大使用ﾉｰﾄﾞ数	ﾉｰﾄﾞ多重度	経過時間上限(h)	CPU 時間上限(h)	ﾒﾓﾘ上限(GB/ﾉｰﾄﾞ)
1	S1	シングルジョブ実行	1	1	24	ー	25
2	P4	並列ジョブ実行	4	1	24	ー	25

バッチジョブクラスは、今後変更されることがあります。現在設定されているバッチジョブクラスは、“ llclass ”コマンドで確認することができます。

4.3. ジョブスクリプトの作成

バッチジョブを実行するためには、次のような JCF スクリプトを作成して、llsubmit コマンドによりジョブを投入します。

（例１）sample.f をコンパイルして実行させる JCF スクリプト(*JCFSAMPLE*)

#!/bin/csh -f	スクリプトが csh 構文であることの宣言
#@ class = S1	ジョブクラスの指定（必須）
#@ output = $(jobid).out	標準出力ファイル（必須）
#@ error = $(jobid).err	標準エラー出力ファイル（必須）
#@ environment = COPY_ALL	ユーザ環境の引継ぎ（任意）
#@ queue	ジョブステップの実行（必須）
hostname	実行ステートメント：ホスト名表示
unlimit	：資源制限値解除
f90 -o sample.exe　sample.f	：コンパイル
./sample.exe	：実行
echo "TEST END"	：終了メッセージ書込み

4.4. バッチジョブの投入

llsubmit コマンドで作成した JCF スクリプトを投入する。

```
%　llsubmit JCFSAMPLE
llsubmit: The job "ismsr-cx1.ism.ac.jp.5651" has been submitted.
```

ジョブ ID　5561 でサブミットされました。

4.5. ジョブのステータス

投入されたジョブの状態を確認するためには、llq コマンドを使用します。

```
% llq
Id                       Owner      Submitted   ST PRI Class        Running On
------------------------ ---------- ----------- -- --- ------------ -----------
ismsr-cx1.5662.0         hitachi     2/26 07:41 R  50  S1           ismsr-cx2
ismsr-cx1.5664.0         hitachi     2/26 07:42 R  50  S1           ismsr-cx3
ismsr-cx1.5663.0         hitachi     2/26 07:42 I  50  P4

3 job step(s) in queue, 1 waiting, 0 pending, 2 running, 0 held, 0 preempted
```

Id：ジョブ ID

Owner：ジョブ投入ユーザ名

Submitted：ジョブ投入時間

ST：状態：R（実行中)、I（実行待ち）

Class：投入ジョブクラス

RunningON：ジョブが実行されているノード名（ホスト名）

（制御用インターフェースを使用しているため実際のホスト名と異なる場合があります。）

4.6. ジョブのキャンセル

実行待ちまたは実行中のジョブをキャンセルするためには、llcancel コマンドを使用します。

```
% llcancel 5664                ……キャンセルするジョブの ID を指定します。
llcancel: Cancel command has been sent to the central manager.
```

他利用者のジョブは、キャンセルできません。

4.6. 並列プログラムの実行

並列ジョブを実行するために Parallel Environment（poe、mpiexec）を使用します。

（例 2）並列プログラム sample.exe を poe コマンドで 4 並列実行させる JCF

#!/bin/csh -f #@ class = P4 #@ job_type = parallel #@ node = 4 #@ output = $(jobid).out #@ error = $(jobid).err #@ environment = LANG=C; ¥ #　NLSPATH=/usr/lib/nls/msg/C/pepoe.cat #@ queue unlimit poe ./sample.exe –procs 4	スクリプトが csh 構文であることの宣言 ジョブクラスの指定（必須） 並列ジョブの指定（必須） 使用ノード数を指定（必須） 標準出力ファイル（必須） 標準エラー出力ファイル（必須） ユーザ環境の設定（必須） (継続行：上からの続き) ジョブステップの実行（必須） 実行ステートメント -procs 4 は、省略可能

(注)class,jyob_type,node,environ は必ず指定してください。

4.7. 主な JCF ステートメント

JCF ステートメントは、JCF スクリプト内で、＃@に続けて記述します。

ステートメント	構文	機能
class	class = *<CLASS Name>*	実行ジョブクラスを指定します。 P4 or S1 省略時は S1 になります。
job_type	job_type=<parallel/serial>	ジョブのタイプ（並列・逐次）を指定します。 省略時は、serial（逐次）になります。
node	node=<1-4>	使用ノード数を指定します。 省略時は node=1 です。
output	output=*<file name>*	標準出力ファイルを指定します。 省略時は、/dev/null です。
error	error=*<file name>*	標準エラー出力ファイルを指定します。 省略時は、/dev/null です。
input	input=*<file name>*	標準入力ファイルを指定します。 省略時は、/dev/null です。
environment	environment=*<env1;env2…>*	ジョブ実行時の初期環境変数を指定します。 COPY_ALL を指定すると現在の環境変数をすべて引き継ぎます。
queue	queue	ジョブステップのひとつのコピーをキューに入れます。この設定値は必須です。

4.8. 要素並列を使用しない MPI プログラム実行方法

1 ノード 16CPU をバラバラに MPI プログラムで実行する方法を示す。

(1)コンパイル

コンパイルに mpif90 を使用しコンパイルオプションの最後に" -noparallel "を追加する。

(例）MPI プログラム sample.f をコンパイルする。

```
% mpif90 -Os -64 -noparallel sample.f
```

(2)実行方法

(例）1 ノード 16CPU を使用するための JCF

#!/bin/csh -f	スクリプトが csh 構文であることの宣言
#@ class = P4	ジョブクラスの指定（必須）
#@ job_type = parallel	並列ジョブの指定（必須）
#@ node = 1	使用ノード数１を指定（必須）
#@ total_tasks = 16	実行プロセス数を指定
#@ resources=ConsumableCpus(1)	プロセスあたりの CPU 数１を指定
#@ output = $(jobid).out	標準出力ファイル（必須）
#@ error = $(jobid).err	標準エラー出力ファイル（必須）
#@ environment = LANG=C; ¥	ユーザ環境の設定（必須）
#　NLSPATH=/usr/lib/nls/msg/C/pepoe.cat	(継続行：上からの続き)
#@ queue	ジョブステップの実行（必須）
unlimit	実行ステートメント
poe ./sample.exe –procs 16	-procs 16 は、省略可能

(例)3 ノード 48CPU を使用するための JCF

#!/bin/csh -f	スクリプトが csh 構文であることの宣言
#@ class = P4	ジョブクラスの指定（必須）
#@ job_type = parallel	並列ジョブの指定（必須）
#@ node = 3	使用ノード数を指定（必須）
#@ total_tasks = 48	実行プロセス数を指定
#@ resources=ConsumableCpus(1)	プロセスあたりの CPU 数１を指定
#@ output = $(jobid).out	標準出力ファイル（任意）
#@ error = $(jobid).err	標準エラー出力ファイル（任意）
#@ environment = LANG=C; ¥	ユーザ環境の設定（必須）
#　NLSPATH=/usr/lib/nls/msg/C/pepoe.cat	(継続行：上からの続き)
#@ queue	ジョブステップの実行（必須）
unlimit	実行ステートメント
poe ./sample.exe –procs 48	-procs 48 は、省略可能

注意事項：並列プログラム実行のプロセス数は、#@ total_tasks = N パラメータが優先される。

4.9. 要素並列と MPI を組合せて実行する方法

1 ノード 16CPU を要素並列（SMP)と MPI を組み合わせて実行する方法を示す。

(1)コンパイル

コンパイルに mpif90 を使用する。

(例）MPI プログラム sample.f をコンパイルする。

```
% mpif90 -Os -64 sample.f
```

(2)実行方法：実行時オプションで要素並列で使用する CPU 数を指定する

実行時オプション：PRUNST(THREADNUM(*n*))…*n に要素並列で使用する CPU 数を指定*

指定を省略した場合は、n=16 を仮定します。

(例）要素並列２CPU×８MPI 並列で実行するための JCF

#!/bin/csh -f	スクリプトが csh 構文であることの宣言
#@ class = P4	ジョブクラスの指定（必須）
#@ job_type = parallel	並列ジョブの指定（必須）
#@ node = 1	**使用ノード数１を指定（必須）**
#@ total_tasks = 8	**実行プロセス数を指定**
#@ resources=ConsumableCpus(2)	**プロセスあたりの CPU 数 2 を指定**
#@ output = $(jobid).out	標準出力ファイル（必須）
#@ error = $(jobid).err	標準エラー出力ファイル（必須）
#@ environment = LANG=C; ¥	ユーザ環境の設定（必須）
# NLSPATH=/usr/lib/nls/msg/C/pepoe.cat	(継続行：上からの続き)
#@ queue	ジョブステップの実行（必須）
unlimit	実行ステートメント
poe ./sample.exe -F'PRUNST(THREADNUM(2))'	**実行時オプションを指定**