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
並列計算機利用の手引

1.8 版

はじめに

本手引書は統計数理研究所にて稼動する統計科学並列計算機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	2004 年 1 月 13 日
1.1	項目 2.7 Web によるバッチジョブの利用の追加	2004 年 2 月 23 日
1.2	項目 4.2.2 C のサンプルプログラム追加	2004 年 3 月 5 日
1.3	項目 2.3.1 に dpalce コマンドの利用を追加 項目 2.6 に dplace をテンプレートに追加 項目 11.1 時間計測関数の説明変更	2004 年 7 月 1 日
1.4	OS バージョンアップに伴い、OpenMP ジョブ実行時の dplace のオプションを変更 PBS キュークラス情報の追加（q1、q1r に関して） コンパイラバージョンアップに伴うコマンド変更 StackSize 制限に関する記述削除（Intel コンパイラの制限解除）	2005 年 3 月 25 日
1.5	項目 15 に各種マニュアルについてを追加	2005 年 8 月 19 日
1.6	histx の注意事項、histx 使用時の dplace のオプションを変更	2005 年 12 月 1 日
1.7	2.2 キュー構成変更を反映	2006 年 5 月 16 日
1.8	項目 14 に PGPLOT の利用方法を追加	2006 年 7 月 24 日

1　並列計算機フロントエンドへのログイン

プログラムの作成、コンパイル、インターラクティブなデバッグ、バッチジョブ投入などはフロントエンドマシンにログインして行います。フロントエンドマシンのホスト名は以下のとおりです。

ホスト名	IP アドレス
ismaltx.ism.ac.jp	133.58.226.30

フロントエンドマシン上で利用可能なエディタは vi、および emacs などです。

フロントエンドでのインターラクティブによるプログラム実行は CPUTIME で 2 時間に制限されています。長時間のプログラム実行はバッチをご利用下さい。

バッチジョブの投入は PBS を用いて行います。

フロントエンドマシンは、通常の telnet、ftp の他に、SSH（Secure Shell）も利用可能です。rlogin、rsh は利用できません。なるべく SSH の利用をお願いします。

2 バッチジョブの投入方法

2.1 並列計算機のバッチジョブソフトウェア

並列計算機のバッチジョブソフトウェアは PBS(Portable Batch System)を使用し、運用をしています。バッチジョブを投入する場合は並列計算機フロントエンド(ismaltx.ism.sc.jp)からジョブ投入を行います。並列計算機フロントエンド上で投入されたジョブは、ジョブクラスの設定により実行ホストへ送られてジョブ実行されます。

2.2 キュー構成

キュー名	経過時間制限	メモリ制限(GB)	最大並列度	最大ジョブ本数	最大ジョブ本数(1 ユーザ)	実行ホスト	備考
q8r	60H	48	8	4	2	ismaltx4	
q16r	60H	96	16	2	2	ismaltx4	
q32r	120H	192	32	2	1	ismaltx4	
q8	60H	64	8	4	2	ismaltx3 ismaltx4	default キュー
q16	60H	128	16	2	2	ismaltx3 ismaltx4	
q32	120H	256	32	1	1	ismaltx3 ismaltx4	
q64	120H	512	64	1	1	ismaltx1 ismaltx2	
q64s	24H	512	64	1	1	ismaltx1 ismaltx2	
q128	120H	1024	128	1	1	ismaltx1 ismaltx2	通常時 Close
q1	60H	8	1	8	4	ismaltx3 ismaltx4	
q1r	60H	8	1	8	4	ismaltx4	

q1r・q8r・q16r・q32r ジョブクラスは乱数発生ボード 16 枚を搭載し、OpenMP、MPI プログラムが実行可能です。

q8・q16・q32・q64・q64s ジョブクラスは OpenMP、MPI プログラムが実行可能です。

q128 のジョブクラスは MPI プログラムのみ実行が可能です。また q128 キュークラスは通常は CLOSE の状態で利用のリクエストがあった場合に OPEN になります。

2.3 ジョブスクリプトの作成

ジョブ投入する際に使用するスクリプトファイルの注意点を以下に示します。

2.3.1 並列処理していないプログラムの例

```
1 #!/bin/csh
2 #PBS -q q8
3 #PBS -m abe
4 #PBS -j oe
5 #PBS -l ncpus=1
6 cd /home0/sgi/prog1
7 dplace ./a.out
```

#PBS は、シェルとしてはコメント扱いですが PBS では解釈されます。

1 行目:スクリプトを csh の文法で書くように宣言。

2 行目:キュー名の指定(**必須**)

3 行目:異常終了(a)とジョブ実行時(b)とジョブ終了(e)時にメールで送信。

記述がない場合にはメール送信は行われません。

4 行目:stdout と stderr を同一ファイルに出力。

5 行目:利用する CPU 数(並列度)を指定。

指定がない場合には最大並列度の値が設定されます。

6 行目:実行ファイルがあるディレクトリへの移動。

7 行目:プログラムの実行。

注:並列処理されていないプログラムの実行時には dplace コマンドを指定してください。

2.3.2 OpenMP 並列化ジョブの例

```
1 #!/bin/csh
2 #PBS -q q8
3 #PBS -m ae
4 #PBS -j oe
5 #PBS -l ncpus=8
6 setenv OMP_NUM_THREADS 8
7 cd /home0/sgi/test1
8 dplace -x2 ./test_prog
```

6 行目:OpenMP 並列ジョブの並列度を指定する環境変数。(**必須**)

7 行目:実行ファイルがあるディレクトリへの移動。

8 行目:プログラムの実行。

注:自動並列化など OpenMP 並列プログラムの実行時には dplace -x2 コマンドとオプションを必ず指定して下さい。

2.3.3 MPI 並列化ジョブの例

```
1 #!/bin/csh
2 #PBS -q q8
3 #PBS -m ae
4 #PBS -j oe
5 #PBS -l ncpus=8
6 cd /home0/sgi/test1
7 mpirun -np 8 dplace -s1 ./test_mpi < prog.dat
```

7 行目:MPI 並列ジョブを実行するには mpirun コマンドになり、並列度は-np オプションの後に指定します。

注:MPI 並列プログラムの実行時には dplace -s1 コマンドとオプションを必ず指定して下さい。

2.3.4 q128 キュークラスへのジョブ投入

q128 のジョブクラスは MPI プログラムのみ実行可能なキュークラスになっており、2 台のホストを利用しジョブを実行します。他のジョブクラスとは違い mpirun コマンドの引数に"$PBS_SGINODE"の環境変数を使いプログラムを実行して下さい。

以下がジョブスクリプトファイルのサンプルになります。

```
1 #!/bin/csh
2 #PBS -q q128
3 #PBS -m ae
4 #PBS -j oe
5 #PBS -l ncpus=128
6 cd /home0/sgi/test1
7 mpirun $PBS_SGINODE dplace -s1 ./test_mpi < prog.dat
```

2.4　バッチジョブの投入

バッチ投入するコマンドは qsub コマンドです。qsub コマンドはフロントエンドのホストからのみ実行が行えます。

2.4.1　　qsub コマンド使用例

```
%qsub [options]ジョブスクリプトファイル名
%cat subfile
#!/bin/csh
#PBS -q q8
#PBS -m ae
#PBS -j oe
#PBS -l ncpus=8
setenv OMP_NUM_THREADS 8
cd /home0/sgi/test1
dplace -x2 ./test_prog
%qsub subfile
10.ismaltx.ism.ac.jp
```

10 のジョブ番号で投入されていることになります。

qsub の主なオプションは以下になります。

オプション	説明
-q destination	ジョブを投入するキューの指定。
-N name	ジョブ名の指定。指定しないとジョブスクリプトの名前か、STDIN に設定されます。(空白文字は使うことはできません)
-a date_time	指定した日時にジョブを投入するように指定します。date_time は[[[[CC]YY]MM]DD]hhmm[.SS]というような形式で指定します。CCYY は西暦、MMDD は月日、hhmm.SS は時分秒です。
-j oe | eo	stdout(標準出力),stderr(標準エラー出力)の出力ファイルを統合。"oe"を指定した場合これらの出力をまとめて stdout に出力します。"eo"を指定した場合出力をまとめて stderr に出力します。
-o path	stdout の出力ファイル名を指定。無指定だと"ジョブ名.o ジョブ番号"となります。投入時のディレクトリに対する相対パス、絶対パス、ホスト名:パスというような指定方法があります。
-e path	stderr の出力ファイル名を指定。無指定だと"ジョブ名.e ジョブ番号"となります。投入時のディレクトリに対する相対パス、絶対パス、ホスト名:パスというような指定方法があります。

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 qsub)を参照して下さい。

2.5　ジョブのステータス

ジョブのステータスを参照は qstat コマンドです。qstat コマンドはフロントエンドのホストからのみ実行が行えます。

2.5.1 qstat コマンド使用例

```
% qstat
                                                  Elapsed   Elapsed      Cputime/
Job ID          Username Queue    Jobname    S Walltime  Cputime      Walltime
--------------- -------- -------- ---------- - --------- ------------ --------
171.ismaltx     sgi      q8       ISM_test   R  00:00:20     00:02:40       8.0
```

Job ID :ジョブ番号

Username :ユーザ名

Queue :投入キュー

Jobname :-N ジョブ名で投入されたジョブ名

S :動作状況の表示

Q :キューイング状態

R :ランニング状態

Elapsed Walltime :実行時間

Elapsed Cputime :実行 CPU 時間

Cputime/Walltime :並列度状況

全てのジョブと他ユーザの投入ジョブが表示されます。

特定のジョブ番号の詳細を確認したいときには"qstat -f JobID"で確認できます。

メモリ使用量を確認する際には"qstat -a"で確認することができます。

```
% qstat -a

ismaltx: SGI Altix SuperCluster
                                                                  Elapsed   Used
Job ID          Username Queue    Jobname    SessID Nds Cpu S WallTime  Memory
--------------- -------- -------- ---------- ------ --- --- - --------- ------
171.ismaltx     sgi      q8       ISM_test    16508   1   8 R  00:15:15 1467mb
```

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 qstat)を参照して下さい。

2.5.2 ジョブのキャンセル

ジョブの中断などでキャンセルするには qdel コマンドです。qdel コマンドはフロントエンドのホストからのみ実行が行えます。

```
% qdel ジョブ ID
```

2.6　ジョブスクリプトファイルのテンプレート

ジョブスクリプトファイルのテンプレートは/home0/sample/template 配下にあります。

またジョブスクリプトファイルの中では”##PBS”はコメント行扱いになります。テンプレートファイルをコピーし利用する際には、有効にしたい PBS オプション行の先頭#を削除して下さい。

template_single.pbs は並列処理していないプログラムを実行する際に使用してください。

template_single.pbs

```
#!/bin/csh
##PBS -q q8
##PBS -q q8r

##PBS -N job_name
##PBS -j oe
##PBS -m ae
#PBS -l ncpus=1

#dplace exe_file_name
```

template_single.pbs 使用例

```
% pwd
  /home0/sgi/prog1
% cp /home0/sample/template/template_single.pbs   jobfile.csh

% vi jobfile.csh
#!/bin/csh
#PBS -q q8       ※(#を１つにすることで適用となります)
##PBS -q q8r

#PBS -N single_job
#PBS -j oe
##PBS -m ae
#PBS -l ncpus=1

※以下にジョブ実行スクリプトを記述して下さい。
cd /home0/sgi/prog1

dplace ./a.out

% qsub jobfile.csh
  200.ismaltx.ism.ac.jp

% qstat
                                                  Elapsed   Elapsed      Cputime/
Job ID          Username Queue    Jobname    S Walltime  Cputime      Walltime
--------------- -------- -------- ---------- - --------- ---------- --------
200.ismaltx        sgi      q8    single_job  R  00:01:20   00:00:57      0.7
```

template_Openmp.pbs は OpenMP で並列化されたプログラムを実行する際使用してください。

ジョブスクリプトファイルの中では"##PBS"はコメント行扱いになります。テンプレートファイルをコピーし利用する際には、有効にしたい PBS オプション行の先頭#を削除して下さい。

template_Openmp.pbs

```
#!/bin/csh
##PBS -q q8
##PBS -l ncpus=8
#setenv OMP_NUM_THREADS 8

##PBS -q q16
##PBS -l ncpus=16
#setenv OMP_NUM_THREADS 16

##PBS -q q32
##PBS -l ncpus=32
#setenv OMP_NUM_THREADS 32

##PBS -q q8r
##PBS -l ncpus=8
#setenv OMP_NUM_THREADS 8

##PBS -q q16r
##PBS -l ncpus=16
#setenv OMP_NUM_THREADS 16

##PBS -q q32r
##PBS -l ncpus=32
#setenv OMP_NUM_THREADS 32

##PBS -q q32m
##PBS -l ncpus=32
#setenv OMP_NUM_THREADS 32

##PBS -q q64
##PBS -l ncpus=64
#setenv OMP_NUM_THREADS 64
##PBS -N job_name
##PBS -m ae
##PBS -j oe

# dplace -x2 exe_file_name
```

template.Openmp.pbs 使用例

```
% pwd
  /home0/sgi/prog2
% cp /home0/sample/template/template_Openmp.pbs   jobfile.csh

% vi jobfile.csh
#!/bin/csh
#PBS -q q8
#PBS -l ncpus=8      ※(#を１つにすることで適用となります)
setenv OMP_NUM_THREADS 8

##PBS -q q16
##PBS -l ncpus=16
#setenv OMP_NUM_THREADS 16

##PBS -q q32
##PBS -l ncpus=32
#setenv OMP_NUM_THREADS 32

##PBS -q q8r
##PBS -l ncpus=8
#setenv OMP_NUM_THREADS 8

##PBS -q q16r
##PBS -l ncpus=16
#setenv OMP_NUM_THREADS 16

##PBS -q q32r
##PBS -l ncpus=32
#setenv OMP_NUM_THREADS 32

##PBS -q q32m
##PBS -l ncpus=32
#setenv OMP_NUM_THREADS 32

##PBS -q q64
##PBS -l ncpus=64
#setenv OMP_NUM_THREADS 64

#PBS -N Openmp_job
##PBS -m ae
#PBS -j oe
※以下にジョブ実行スクリプトを記述して下さい。
cd /home0/sgi/prog2

dplace -x2 ./a.out    ※OpenMP プログラム実行の際には dplace -x2 を指定
```

```
% qsub jobfile.csh
  201.ismaltx.ism.ac.jp
% qstat
                                                  Elapsed   Elapsed      Cputime/
Job ID          Username Queue    Jobname    S Walltime  Cputime      Walltime
--------------- -------- -------- ---------- - --------- ---------- --------
201.ismaltx        sgi      q8    Openmp_job  R  00:01:20   00:08:40      8.0
```

template_MPI.pbs は MPI で並列化されたプログラムを実行する際使用してください。

ジョブスクリプトファイルの中では"##PBS"はコメント行扱いになります。テンプレートファイルをコピーし利用する際には、有効にしたい PBS オプション行の先頭#を削除して下さい。

template_MPI.pbs

```
#!/bin/csh
##PBS -q q8
##PBS -q q16
##PBS -q q32
##PBS -q q8r
##PBS -q q16r
##PBS -q q32r
##PBS -q q32m
##PBS -q q64
##PBS -q q128
##PBS -l ncpus=XX
##PBS -N job_name
##PBS -m ae
##PBS -j oe

#mpirun -np xx dplace -s1 exe_file_name
```

template_MPI.pbs 使用例

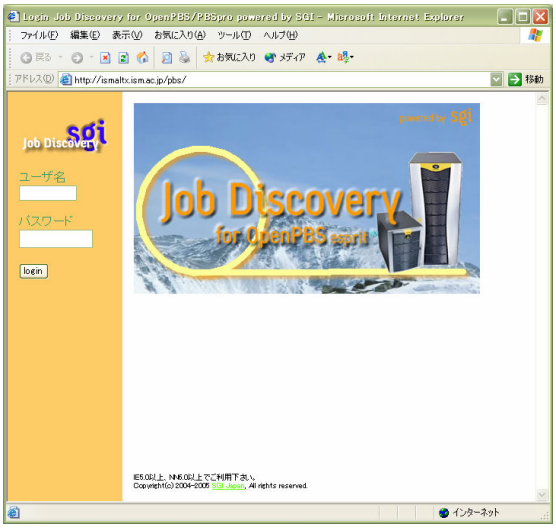
```
% pwd
  /home0/sgi/prog3
% cp /home0/sample/template/template_MPI.pbs  jobfile.csh

% vi jobfile.csh
#!/bin/csh
#PBS -q q8      ※(#を１つにすることで適用となります)
##PBS -q q16
##PBS -q q32
##PBS -q q8r
##PBS -q q16r
##PBS -q q32r
##PBS -q q32m
##PBS -q q64
##PBS -q q128
#PBS -l ncpus=8      ※(#を１つにすることで適用となります)
#PBS -N MPI_job
##PBS -m ae
##PBS -j oe
※以下にジョブ実行スクリプトを記述して下さい。
cd /home0/sgi/prog3

mpirun -np 8 dplace -s1 ./a.out
※MPI プログラム実行の際には dplace - s1 を指定

% qsub jobfile.csh
  202.ismaltx.ism.ac.jp
%qstat
                                                  Elapsed   Elapsed      Cputime/
Job ID          Username Queue    Jobname    S Walltime  Cputime      Walltime
--------------- -------- -------- ---------- - --------- ------------ --------
201.ismaltx        sgi      q8    MPI_job     R  00:20:20   00:50:40      8.0
```

2.7　Web によるバッチジョブの利用

並列計算機では Web によるバッチジョブの投入、参照、削除などを行える機能があります。

2.7.1　　ログイン方法

URL は http://ismaltx.ism.ac.jp/pbs です。アクセスしますと以下の画面が表示されます。



ユーザ名、パスワードは ismaltx のユーザ名、パスワードを入力し、login ボタンをクリックします。ログインに成功するとジョブステータスの参照画面が表示されます。

2.7.2　ジョブの状態(Job Status)

JobStatus をクリックするとジョブステータスの参照画面が表示され、計算ホストの状態、ジョブの状況、キューの状況、バッチジョブサーバの CPU とメモリの使用量などが参照できます。
ジョブの削除などが行えます。

計算ホストの状態は以下になります。

計算ホスト	 offline(メンテナンス中) down(システム停止中)　  free (リソースがあいている状態なのでジョブの新規実行が可能です)　  full / busy (すべてのリソースを使用中のため新規ジョブは投入待ちになります) 画像はホストの種類により異なります。
空き CPU / 総 CPU	計算ホストの CPU 数に対する 利用可能 CPU の表示
キューとの関連	キューごとのジョブ情報の「関連」に表示されているものと同一の計算ホストでジョブが実行されます
Jobs (Job ID/並列数)	Job ID が 200、並列数(ncpus)の指定が 8 のときは 200/8 と表示されます

ジョブの状況は以下になります。

Job ID	ジョブ ID（これをクリックするとジョブの詳細が閲覧できます）
ユーザ名	実行ユーザ名
キュー	ジョブが投入されているキュー名
Session ID	計算ホストのプロセスを管理するＩＤ
ホスト数	ジョブ実行しているホスト数
並列数	ユーザが指定した並列数
Stat	ジョブのステータス よく表示されるステータス

	R ： ジョブ実行中 Q ： 実行待ち E ： ジョブの終了処理中 障害時や管理者の設定によるステータス W ： ファイル転送時に起こるエラー S ： その他の障害よる停止, H ： qhold による停止
経過時間	実行中のジョブの実経過時間
メモリ使用量	ジョブで利用された最大実メモリ使用量
実行ホスト	ジョブが実行中の計算ホスト
削除	自分のジョブを削除するときにボタンが表示されます

キューの状況は以下になります。

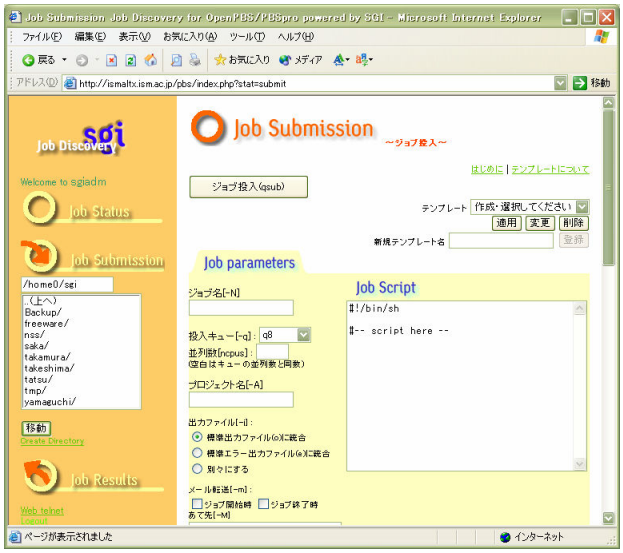
キュー	キュー名
現在のジョブ本数	投入された総ジョブ数、実行待ち、実行中、 ファイル転送中のジョブ数の表示。
投入、実行	投入が ○ の時はジョブの受付が可能です。 実行が ○ の時はジョブの実行が可能です。
最大本数	投入 ： キューあたりの最大投入本数 実行 ： 実行可能な最大本数 Group ： グループあたりの最大実行本数 User ： ユーザあたりの最大実行本数 0 が表示されている場合には制限がありません。
リソース制限	最大並列数、最大計算時間(実時間)、 ジョブ全体で利用可能な最大実メモリの表示
関連	同じ関連を持つホスト上でジョブは実行されます

PBS Server の状況は以下のとおりです。

interactive	telnet, ssh などでログインして利用している システムリソース (ＯＳなどシステムリソースも含む)
batch(PBS)	PBS から利用しているシステムリソース

2.7.3　ジョブの投入(Job Submission)

Job Submisson をクリックするとジョブを投入に必要な画面が表示されます。



各メニューの説明は以下になります。

Job Submission のメニュー下に現在のディレクトリ名とディレクトリリストが表示されていますので、ジョブを実行したいディレクトリに移動してください。

新規ディレクトリを作成したい場合は Create Directory をクリックするとディレクトリを作成できます。

次に、PBS の各パラメータおよびスクリプトを入力してください。設定終了後 ジョブの投入(qsub)ボタンをクリックするとジョブが投入され、現在の設定を元にジョブ名とジョブＩＤからスクリプトファイルを作成されます。

各項目の説明は以下になります。

ジョブ名	ステータスの表示などでジョブの内容がわかりやすくなるような名前をつけます。(英数字で 15 文字以内)
投入キュー	画面下に表示されているキューから選択してください。
並列数	キューの並列数以下の並列数を指定。(空白時はキューの並列度と同数になります)
出力ファイル	ジョブの標準・標準エラー出力ファイルの形式について指定します。
メール転送	サーバに障害があったりリソース制限に該当したときはその旨がユーザにメールで通知されます。さらにジョブの開始、終了時にもメールを送信することが可能です。
あて先	メールのあて先を PBS サーバのアカウントから任意のアドレスに転送することができます。
プロジェクト 名	一人のユーザがプロジェクト毎に稼動統計を行うときに指定します。**プロジェクト名を指定されていないサイトではこの値は必要ありません。**
障害時の扱い	万一、計算サーバの障害が発生したときに実行されていたジョブは、終了してキューから削除されます。 再計算を選択することにより障害後はじめから計算しなおすことができます。

2.7.4　ファイルの転送、ファイルの修正

画面の下の部分には PBS Local Files, File Servers, Your Client のタブがあります。サーバのファイル修正、ブラウザを起動している PC 端末などからのファイル転送、別のファイルサーバからのファイル転送の設定をすることができます。**File Servers および Stagein の check box は並列計算機構成では必要ありませんので利用しないで下さい。**

現在いるディレクトリのファイル一覧(ファイル名、サイズ、最終更新日のリストが表示されています。ファイル名をクリックするとファイルをダウンロードすることができます。

注意) Internet Explorer では、ファイル名をクリックするとファイルを保存するかのウィンドウが表示されますがこのとき開くを選択するとファイルの中身が表示されないなどのエラーとなります。必ずいったん保存してからそのファイルを開くようにしてください。

Netscape Navigator では、クリックすると保存するかどうかきかれますが、右クリックしたときはデフォルトのファイル名を down.php として保存しようとします。ファイルそのものは問題ありませんのでファイル名を適宜設定してください。

また、ファイルには次のアイコンが表示されているものがあります。

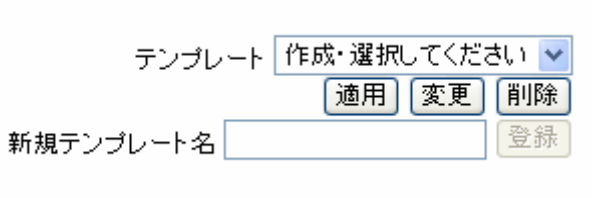
1MB 未満のファイルを直接編集することが可能です。設定ファイルなどの修正が行えます。

アイコンがあるファイルは PBS スクリプトです。過去に実行した PBS のジョブスクリプトをテンプレートとして利用することができアイコンをクリックするとこのスクリプトを基にしたパラメータの設定が可能となります。前に実行したジョブを元に新しいジョブを作成するときなどで利用できます。

ブラウザを表示しているクライアント PC/WS からファイルを転送することができます。最大 10 ファイル、最大サイズは合計で 4ＭＢまでです。

2.7.5　テンプレートの活用

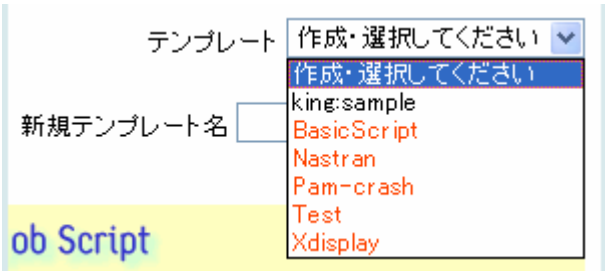
キューやジョブ名などの PBS のパラメータ、ジョブスクリプトの構成がパターン化されているときテンプレートを作成することができます。



パラメータを設定してから新規テンプレート名を入力して[登録]をクリックするとテンプレートリストに新規テンプレートが登録されます。

テンプレートファイルはユーザのホームディレクトリの.pbstmpl/ ディレクトリ以下に shell script として保存します。

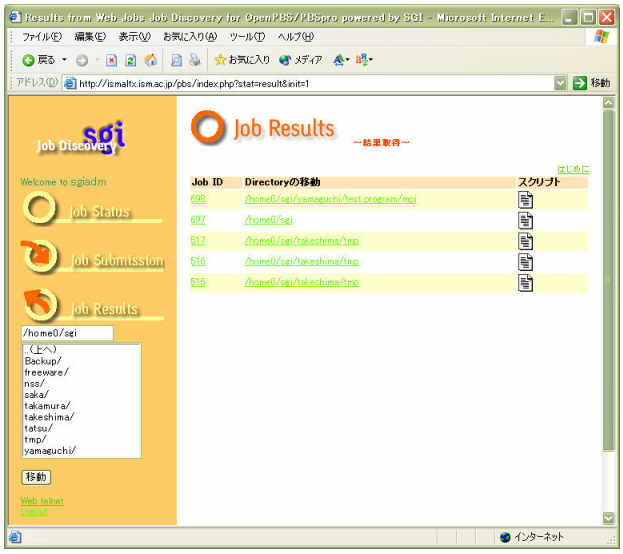
ユーザが登録したテンプレート名は黒で表示されます。別に、赤文字のテンプレートもあります。これらは、システムテンプレートと呼ばれ、PBS の設定の右側でスクリプトのほかにアプリケーションに特化した入力をすることが可能となります。



並列計算機ではシステムテンプレートを利用する必要はありません。

2.7.6　ジョブの結果(Job Results)

結果一覧を見るときには Job Results より行います。
ここではジョブ一覧のほかにファイルのダウンロード、削除が可能です。
Job Results にアクセスすると最近投入された最大 20 件のジョブ一覧が表示されます。



投入した Job ID、投入したディレクトリ、ジョブスクリプトのアイコンが表示され、ファイルをダウンロードするときはディレクトリ名をクリックするか、左のディレクトリナビゲーションで移動してください。

ファイルの取得・削除はディレクトリ、ファイルの一覧からファイル名をクリックするとファイルをダウンロードすることができます。
またファイル･ディレクトリのチェックボックスにチェックを入れて[ファイルを削除]を選択すると該当するファイル、ディレクトリが削除されます。
[圧縮してダウンロード]を選択すると pbsweb.tar.gz ファイルとして圧縮されます。

3 コンパイラ

並列計算機では以下のコンパイラが利用可能です。

3.1.1 Intel Compiler

Intel Fortran Compiler は Fortran 77 90 95 をサポートしますので、どの規格に沿って書かれたソースプログラムでもコンパイル可能です。Fortran95 は Fortran90 に対して、Fortran90 は Fortran77 に対して上位互換であるため、それぞれの規格が混在しているようなソースプログラムもコンパイルできます。

コマンド	内容
ifort	Fortran 77 90 95 をサポート
icc	C C++ をサポート

3.1.2 GNU コンパイラ

GNU コンパイラは、GNU プロジェクトで開発されたコンパイラです。GNU コンパイラでは以下の言語が利用可能です。

コマンド	内容
gcc	C をサポート
g++	C++ をサポート
g77	Fortran77 をサポート

最良のパフォーマンスを得るために Intel Compiler を推奨致しますが、GNU ツールである g77、gcc、g++もお使い頂けます。このガイドでは、Intel Compiler ifort、icc のオプションをご紹介します。尚、－help オプションを指定して頂くとコンパイラオプションの一覧が表示されます。

4 Intel コンパイラにおけるプログラムの最適化方法

4.1 シングルプロセッサプログラムの最適化方法

本章では、Intel コンパイラでのシングルプロセッサプログラムの最適化についてご説明します。まず正しい結果が得られていることを確認してください。他のシステムからプログラムを移植する場合には、結果の妥当性に対して検討が必要です。一般には、プログラムが高級言語で書かれていて、それらが言語規格に正しく準拠している場合には、その移植には何ら問題はありません。デバッグの方法については「デバッグ」を参照してください。

4.1.1 データ型

Intel コンパイラのデータ型の一覧を以下に示します。

C	bit	Fortran
char	8	CHARACTER
short int	16	INTEGER(KIND=2)
int	32	INTEGER(KIND=4)
long int	64	-
long long int	64	INTEGER(KIND=8)
pointer	64	POINTER
float	32	REAL(KIND=4)
double	64	REAL(KIND=8)/double precision
long double	128	REAL(KIND=16)

4.1.2 オプション

以下に Intel コンパイラの各種オプションについて説明します。

4.1.2.1 最適化、プロセッサ特化、浮動小数点演算関連のオプション

オプション	内容
最適化レベルオプション	
-O0	すべての最適化を無効にします。デバッグを行うときに指定してください。
-O1	保守的な最適化を行います。
-O2	ソフトウェアパイプラインによる最適化を行います。多くの場合有効です。デフォルトの最適化レベルです。最適化のオプションを何も指定しないとこのオプションで最適化されます。
-O3	積極的な最適化を行います。-O2 の最適化に加えて、ループの交換やデータプリフェッチを行います。
プロセッサに特化したオプション	
-tpp2	Itanium2 に適した命令レベルのスケジューリングを行います。デフォルトで ON になっています。このオプションは指定しなくともデフォルトで付加されます。
浮動小数点演算に関するオプション	
-ftz	アンダーフローが発生したときに、値をゼロに置き換えます。このオプションは指定しなくともデフォルトで付加されます。

4.1.2.2 プロシージャ間解析オプション(IPO)

インライン展開を有効にします。プログラムのなかで小さいサイズの関数を何回も呼び出しているような場合に性能向上の効果があります。関数コールを含むループの最適化にも効果的です。

オプション	内容
-ip	1つのソースファイルにあるプロシージャ間の解析、最適化を行います。
-ipo	複数のソースファイルにあるプロシージャ間の解析、最適化を行います。リンク時にもオプションとして指定してください。

コンパイル例

```
% ifort  -c  -ipo  exm1.f
```

リンク例

```
% ifort  -o  exm1  -ipo  exm1.o
```

4.1.2.3 エイリアス解析オプション

プログラム中の異なるポインタ変数が同じメモリを指している可能性があるとき、コンパイラは積極的な最適化を行うことができません。エイリアスがないとわかっている場合には次のオプションを指定してください。

-fno-alias

どの 2 つのポインタも決して同じメモリ領域を指すことはない(エイリアスがない)とコンパイラは仮定します。積極的な最適化の可能性が増えます。

例)

```
int x
foo(int *p, int *q)
{
   while (q!= NULL)
   {
      x+= *p * q->cost;
      q = q->next;
   }
}
```

変数 x を更新することによって、*p も更新される可能性があります(コンパイラには判断できない)。 しかし、プログラマがエイリアスはないと判断できれば-fno_alias を指定します。コンパイラは積極的な最適化を行うことができます。

4.1.2.4 最適化レポートオプション

オプション	内容
-opt_report	最適化レポートを標準エラー出力に表示
-opt_report_file ファイル名	最適化レポートを指定したファイルに出力

次ページに最適化レポートを使用した場合の例を示します。

例）

```
% cat ex1.f
      1          program ex1
      2          parameter(n=500)
      3          real a(n,n),b(n,n),c(n,n)
      4          common /data/a,b,c
      5
      6          do i=1,n
      7          do j=1,n
      8            a(i,j)=i+k
      9            b(i,j)=i+j+k
     10            c(i,j)=0.0
     11          enddo
     12          enddo
     13          time1=dclock()
     14          do i=1,n
     15          do j=1,n
     16          do k=1,n
     17            c(i,j)=c(i,j)+a(i,k)*b(k,j)
     18          enddo
     19          enddo
     20          enddo
     21
     22          print *,c(n,n)
     23          stop
     24          end

% ifort -opt_report ex1.f
================================================================================
SWP REPORT LOG OPENED ON Fri Feb 14 04:22:15 2003
================================================================================
Optimization Report for: main()
Phase : Extend Insertion

% ifort -O3 -opt_report ex1.f -Vaxlib

================================================================================
SWP REPORT LOG OPENED ON Fri Feb 14 04:22:29 2003
================================================================================
HLO REPORT LOG OPENED ON Fri Feb 14 04:22:29 2003
================================================
Total #of lines prefetched in main for loop in line 7=3
#of Array Refs Scalar Replaced in main at line 16=28
Total #of lines prefetched in main for loop in line 16=8
#of Array Refs Scalar Replaced in main at line 16=3
Total #of lines prefetched in main for loop in line 16=5
Total #of lines prefetched in main for loop in line 16=2

LOOP INTERCHANGE in main at line 6
LOOP INTERCHANGE in main at line 7
LOOP INTERCHANGE in main at line 14
LOOP INTERCHANGE in main at line 15
LOOP INTERCHANGE in main at line 16
        /* －O3 のとき、ループの最適化を行います。 */
Block, Unroll, Jam Report:
(loop line numbers, unroll factors and type of transformation)
Loop at line 7 unrolled without remainder by 4
Loop at line 14 unrolled and jammed by 4
Loop at line 15 unrolled and jammed by 4
```

4.1.3 makefile について

以下に makefile の例を示します。

```
SHELL= /bin/csh
BIN     = a.out
FFLAGS  = -c
CFLAGS  = -c
LFLAGS  =
LIB     = -lscs  －lmpi
F90     = ifort
CC      = icc

OBJ= ¥
main.o    routine1.o    routine2.o    prog.o

$(BIN): $(OBJ)
        $(F90) $(LFLAGS) -o $(BIN) $(OBJ) $(LIB)

.f.o:
        $(F90) $(FFLAGS) $*.f

.c.o:
        $(CC) $(CFLAGS) $*.c

clean:
        -rm -f core $(BIN) $(OBJ)
```

Intel Compiler のコンパイルオプションはデフォルトで最適化を進めるようなレベルが設定されています。そのため、特にオプションを指定しなくてもコンパイラが最適化した実行モジュールを生成します。デフォルトのコンパイラ・オプションの内容は「最適化オプション」をご参照ください。

マクロ名 LIB で指定しているのは、科学技術計算ライブラリ（SCSL）と MPI のリンクです。SCSL ライブラリについては「科学技術計算ライブラリ」を、MPI については「MPI」をご参照ください。

4.2　並列プログラムの最適化方法

本章では、Intel コンパイラでのシングルプロセッサプログラムの最適化についてご説明します。

4.2.1　　自動並列化

Intel コンパイラでは「-parallel」オプションの指定により、コンパイラがソースコード内のループに対して、自動並列化の解析を行います。

オプション	内容
-parallel	自動並列化の指定です。
-par_report{0|1|2|3}	並列化されたループを表示し、並列化されなかったループについてはなぜ並列化されなかったかを説明します。末尾の数字が大きいほど詳細な情報を出力します。
-override_limits	1300 行を越える大きなソースコードで自動並列化オプション -parallel を指定するときには、このオプションも一緒に指定してください。指定しない場合、コンパイルが止まってしまうこともあります。

次ページの自動並列化、並列化解析オプションの例を参照してください。

例）

```
% cat ex8.f
1.           program ex8
2.           parameter(n=1000)
3.           real a(n,n),b(n,n)
4.           do j=1,n
5.           do i=1,n
6.             a(i,j)=1.
7.             b(i,j)=1.
8.           enddo
9.           enddo
10.          do j=1,n
11.          do i=1,n
12.            a(i,j)=a(i,j)*b(i,j)
13           enddo
14.          enddo
15.          do j=1,n
16.          do i=1,n
17.            print *,a(i,j)
18.          enddo
19.          enddo
20.          stop
21.        end

% ifort -parallel -par_report3 ex8.f

   serial loop: line 5: not a parallel candidate due to insufficent work
   serial loop: line 16: not a parallel candidate due to statement at line 17
   serial loop: line 15: not a parallel candidate due to statement at line 17
     /*  並列化できなかったループについて、その理由が出力されます。*/
ex8.f(4) : (col. 0) remark: LOOP WAS AUTO-PARALLELIZED.
   parallel loop: line 4
      shared: {"a", "b"}
      private: {"j", "i"}
      first private: { }
      reductions: { }
ex8.f(10) : (col. 0) remark: LOOP WAS AUTO-PARALLELIZED.
   parallel loop: line 10
      shared: {"a", "b"}
      private: {"j", "i"}
      first private: { }
      reductions: { }
21 Lines Compiled
```

4.2.2 OpenMP プログラムのコンパイル

Intel コンパイラでは「-openmp」オプションの指定により、ソースコード内の OpenMP 指示行を有効にします。

サンプルプログラム 1（Fortran）

```
% cat ex9.f
1.           program ex9
2.           parameter(n=1000)
3.           real a(n,n),b(n,n)
4.           do j=1,n
5.           do i=1,n
6.             a(i,j)=1.
7.             b(i,j)=1.
8.           enddo
9.           enddo
10.     !$OMP PARALLEL DO
11.          do j=1,n
12.          do i=1,n
13.              a(i,j)=a(i,j)*b(i,j)
14.          enddo
15.          enddo
16.          do j=1,n
17.          do i=1,n
18.              print *,a(i,j)
19.          enddo
20.          enddo
21.          stop
22.          end
% ifort -openmp ex9.f
ex9.f(10) : (col. 0) remark: OpenMP DEFINED LOOP WAS PARALLELIZED.
23 Lines Compiled
%
```

サンプルプログラム 2(C)

```
1.#include<stdio.h>
2.
3.#define n 1000
4.
5.int main(void)
6.{
7.        float a[n][n],b[n][n];
8.        int i,j;
9.        for (i=0;i<n;i++){
10.                for (j=0;j<n;j++){
11.                        a[i][j]=1;
12.                        b[i][j]=1;
13.                }
14.       }
15.
16.#pragma omp parallel for
17.        for (i=0;i<n;i++){
18.                for (j=0;j<n;j++){
19.                        a[i][j]=a[i][j]*b[i][j];
20.                }
21.       }
22.
23.        for (j=0;j<n;j++){
24.                for (i=0;i<n;i++){
25.                        b[i][j]=a[i][j];
26.                }
27.         }
28.}

% icc -openmp ex9.c
ex9.c(16) : (col. 1) remark: OpenMP DEFINED LOOP WAS PARALLELIZED.
```

-openmp と-parallel を同時に指定した場合

-openmp と-parallel を同時に（同じソースファイルに）指定した場合、-parallel オプションは OpenMP 指示行を含まないルーチンにだけ有効になります。OpenMP 指示行を含むルーチンでは-openmp だけが有効です。

指定オプション	OpenMP 指示行のあるルーチン	OpenMP 指示行のないルーチン
-openmp	指示行のあるループのみ並列化。その他のループは並列化しない（自動並列化なし）	並列化しない
-parallel	指示行は無効。全てのループに対して自動並列化を試みる	全てのループに対して自動並列化を試みる
-openmp と-prallel	指示行のあるループのみ並列化。その他のループは並列化しない（自動並列化なし）	全てのループに対して自動並列化を試みる

例）-openmp と-parallel を指定した場合(Fotran)(% ifort -openmp - parallel ex10.f)

```
1.        program ex10
2.        parameter(n=1000)
3.        real a(n,n),b(n,n)
4.        do j=1,n
5.        do i=1,n
6.          a(i,j)=1.
7.          b(i,j)=1.
8.        enddo
9.        enddo
10.        call comp(a,b,n)
   ... ... ... ...
20.       subroutine comp(a,b,n)
21.       integer n
22.       real a(n,n),b(n,n)
23.     !$OMP PARALLEL DO
24.           do j=1,n
25.           do i=1,n
26.              a(i,j)=a(i,j)*b(i,j)
27.           enddo
28.           enddo
29.           do j=1,n
30.           do i=1,n
31.              b(i,j)=a(i,j)
32.           enddo
33.           enddo
34.           return
35.           end

ex10 メインルーチン       : 4 行目のループは自動並列化
comp ルーチン             : 24 行目のループは-openmp による並列化
                          : 29 行目のループは並列化されない
```

例）-openmp と-parallel を指定した場合(C)(% ifort -openmp -parallel ex10.c)

```
 1. #include<stdio.h>
 2.
 3. #define N 1000
 4.
 5. float comp(float a[N][N],float b[N][N]);
 6.
 7. int main(void)
 8. {
 9.         int i,j;
10.         float a[N][N],b[N][N];
11.         for (i=0;i<N;i++){
12.                 for (j=0;j<N;j++){
13.                         a[i][j]=1+i;
14.                         b[i][j]=1+j;
15.                 }
16.         }
17.         comp(a,b);
18.    … … … …
30. }
31. float comp(float a[N][N],float b[N][N])
32. {
33.         int i,j;
34. #pragma omp parallel for
35.          for (i=0;i<N;i++){
36.                 for (j=0;j<N;j++){
37.                         a[i][j]=a[i][j]*b[i][j];
38.                 }
39.         }
40.         for (i=0;i<N;i++){
41.                 for (j=0;j<N;j++){
42.                         b[i][j]=a[i][j];
43.                 }
44.         }
45. }
```

ex10 メインルーチン　　：11 行目のループは自動並列化
comp ルーチン　　：35 行目のループは-openmp による並列化
　　：40 行目のループは並列化されない

例） -openmp だけを指定した場合(Fortran)(% ifort -openmp ex10.f)

```
1.        program ex10
2.        parameter(n=1000)
3.        real a(n,n),b(n,n)
4.        do j=1,n
5.        do i=1,n
6.          a(i,j)=1.
7.          b(i,j)=1.
8.        enddo
9.        enddo
10.        call comp(a,b,n)
   ... ... ... ...
20.       subroutine comp(a,b,n)
21.       integer n
22.       real a(n,n),b(n,n)
23.     !$OMP PARALLEL DO
24.           do j=1,n
25.           do i=1,n
26.              a(i,j)=a(i,j)*b(i,j)
27.           enddo
28.           enddo
29.           do j=1,n
30.           do i=1,n
31.              b(i,j)=a(i,j)
32.           enddo
33.           enddo
34.           return
35.           end

ex10 メインルーチン        : 4 行目のループは並列化されない
comp ルーチン              : 24 行目のループは-openmp による並列化
                           : 29 行目のループは並列化されない
```

例）-openmp だけを指定した場合(C)(% ifort -openmp ex10.c)

```
 1. #include<stdio.h>
 2.
 3. #define N 1000
 4.
 5. float comp(float a[N][N],float b[N][N]);
 6.
 7. int main(void)
 8. {
 9.         int i,j;
10.         float a[N][N],b[N][N];
11.         for (i=0;i<N;i++){
12.                 for (j=0;j<N;j++){
13.                         a[i][j]=1+i;
14.                         b[i][j]=1+j;
15.                 }
16.         }
17.         comp(a,b);
18.    … … … …
30. }
31. float comp(float a[N][N],float b[N][N])
32. {
33.         int i,j;
34. #pragma omp parallel for
35.          for (i=0;i<N;i++){
36.                 for (j=0;j<N;j++){
37.                         a[i][j]=a[i][j]*b[i][j];
38.                 }
39.         }
40.         for (i=0;i<N;i++){
41.                 for (j=0;j<N;j++){
42.                         b[i][j]=a[i][j];
43.                 }
44.         }
45. }
```

```
ex10 メインルーチン       : 11 行目のループは並列化されない
comp ルーチン             : 35 行目のループは-openmp による並列化
                          : 40 行目のループは並列化されない
```

例） -parallel だけを指定したとき(Fortran)(% ifort -parallel ex10.f)

```
1.        program ex10
2.        parameter(n=1000)
3.        real a(n,n),b(n,n)
4.        do j=1,n
5.        do i=1,n
6.          a(i,j)=1.
7.          b(i,j)=1.
8.        enddo
9.        enddo
10.        call comp(a,b,n)
   ... ... ... ...
20.       subroutine comp(a,b,n)
21.       integer n
22.       real a(n,n),b(n,n)
23.     !$OMP PARALLEL DO
24.           do j=1,n
25.           do i=1,n
26.              a(i,j)=a(i,j)*b(i,j)
27.           enddo
28.           enddo
29.           do j=1,n
30.           do i=1,n
31.              b(i,j)=a(i,j)
32.           enddo
33.           enddo
34.           return
35.           end

ex10 メインルーチン        : 4 行目のループは自動並列化
comp ルーチン              : 24 行目のループは自動並列化
                           : 29 行目のループは自動並列化
```

例）-parallel だけを指定したとき(C)(% icc -parallel ex10.c)

```
 1. #include<stdio.h>
 2.
 3. #define N 1000
 4.
 5. float comp(float a[N][N],float b[N][N]);
 6.
 7. int main(void)
 8. {
 9.         int i,j;
10.         float a[N][N],b[N][N];
11.         for (i=0;i<N;i++){
12.                 for (j=0;j<N;j++){
13.                         a[i][j]=1+i;
14.                         b[i][j]=1+j;
15.                 }
16.         }
17.         comp(a,b);
18.    … … … …
30. }
31. float comp(float a[N][N],float b[N][N])
32. {
33.         int i,j;
34. #pragma omp parallel for
35.          for (i=0;i<N;i++){
36.                 for (j=0;j<N;j++){
37.                         a[i][j]=a[i][j]*b[i][j];
38.                 }
39.         }
40.         for (i=0;i<N;i++){
41.                 for (j=0;j<N;j++){
42.                         b[i][j]=a[i][j];
43.                 }
44.         }
45. }

ex10 メインルーチン       : 11 行目のループは自動並列化
comp ルーチン             : 35 行目のループは自動並列化
                          : 40 行目のループは自動並列化
```

OpenMP の詳細に関しましては以下をご参照下さい

http://www.openmp.org/

4.2.3 MPI プログラムのコンパイル

コマンド行の末尾に MPI ライブラリのリンクを指示してください。

```
% ifort  ex_mpi.f  -lmpi
```

（実行時には mpirun コマンドを使ってジョブを起動してください。）

4.2.3.1 MPI のデータ型

Fortran プログラムでの MPI のデータ型とそれに対応する Fortran でのデータの型

MPI Datatype	Fortran Datatype	データ長（バイト）
MPI_DATATYPE_NULL	対応する型はありません	-
MPI_INTEGER	integer	4
MPI_REAL	real	4
MPI_DOUBLE_PRECISION	double precision	8
MPI_COMPLEX	complex	8
MPI_DOUBLE_COMPLEX	double complex	16
MPI_LOGICAL	logical	4
MPI_CHARACTER	character	1
MPI_INTEGER1	integer*1	1
MPI_INTEGER2	integer*2	2
MPI_INTEGER4	integer*4	4
MPI_INTEGER8	integer*8	8
MPI_REAL4	real*4	4
MPI_REAL8	real*8	8
MPI_REAL16	real*16	16

C プログラムでの MPI のデータ型とそれに対応する C でのデータの型

MPI Datatype	Fortran Datatype	データ長（バイト）
MPI_CHAR	char	1
MPI_SHORT	short	2
MPI_INT	int	4
MPI_LONG	long	8
MPI_UNSIGNED_CHAR	unsigned char	1
MPI_UNSIGNED_SHORT	unsinged short	2
MPI_UNSIGNED	unsinged int	4
MPI_UNSIGNED_LONG	unsinged long	8
MPI_FLOAT	float	4
MPI_DOUBLE	double	8
MPI_LONG_DOUBLE	long double	16

サンプルプログラム

例)Fortran プログラム

```
% cat simple1.f
        program simple1
        include 'mpif.h'
        call mpi_init(istat)
        call mpi_comm_size(mpi_comm_world, num_procs, ierr)
        call mpi_comm_rank(mpi_comm_world, my_proc, jerr)
        if (my_proc .eq. 0)
     &     write(6,1) 'I am process ',myproc,'.  Total number of proce
     &sses: ',num_procs
        format(a,i1,a,i1)
        call mpi_finalize(ierr)
        stop
        end
% ifort simple1.f -lmpi

% mpirun -np 4 dplace -s1 ./a.out
I am process 0.  Total number of processes: 4
```

例)C プログラム

```
% cat simple1.c
#include <mpi.h>
#include <stdio.h>
main(argc, argv)
int     argc;
char    *argv[];
{
        int     num_procs;
        int     my_proc;

        MPI_Init(&argc, &argv);
        MPI_Comm_size(MPI_COMM_WORLD, &num_procs);
        MPI_Comm_rank(MPI_COMM_WORLD, &my_proc);

        if (my_proc == 0)
                printf("I am process %d.  Total number of processes: %d¥n",
                        my_proc,num_procs);
        MPI_Finalize();
}
% icc -w simple1.c -lmpi
% mpirun -np 4 dplace -s1./a.out
I am process 0.  Total number of processes: 4
```

4.2.3.2 MPI モジュールについて

現在の MPT のバージョンでは MPI モジュールをサポートしていません。USE MPI 文は include 'mpif.h'に修正してください。

4.3　並列ループのデータ分散

Altix システムでは、データ配置に「ファーストタッチ」を採用しています。ファーストタッチとは、データは、最初にそのデータにタッチ（書き込みまたは読み込み）したプロセスのローカルメモリに配置されるというものです。もし、配列（配列 A とする）を初期化しているループが並列化されていないと 1CPU 実行であるために、配列 A はある1つのノードに配置されます。配列 A を処理する並列実行領域では、複数のプロセッサが1つのノードへアクセスすることになります。このような状況ではプログラムの性能向上は望めません。可能なかぎり初期化ループも並列化してください。

例）

```
real*8  A(n), B(n), C(n), D(n)
!$omp parallel do private(i)
    do  i=1, n
      A(i) = 0.
      B(i) = i/2
      C(i) = i/3
      D(i) = i/7
    enddo
!$omp parallel do private(i)
    do  i=1, n
    A(i) = B(i) + C(i) + D(i)
    enddo
```

計算をしている 2 番目のループを並列化するときは、配列を初期化している1番目のループも並列化すれば、より性能が向上します。

5　VAST/toOpenMP における自動並列化方法

自動並列化ツールとして VAST/toOpenMP をご利用。VAST/toOpenMP は Fortran プログラム用の自動並列化ツールです。以下に VAST/toOpenMP の利用方法を説明します。本資料では、以下のサンプルプログラムを OpenMP による並列化させた例を示します。

サンプルプログラム

```
$cat auto.f
           program aut_test
           parameter(n=1000)
           real a(n,n),b(n,n)
           do j=1,n
           do i=1,n
             a(i,j)=1.
             b(i,j)=1.
           enddo
           enddo
           do j=1,n
           do i=1,n
              a(i,j)=a(i,j)*b(i,j)
           enddo
           enddo
           do j=1,n
           do i=1,n
               print *,a(i,j)
           enddo
```

```
            enddo
            stop
            end
```

並列コードの生成

「vastfomp」を利用して、自動並列処理を行います。ここで「-0」オプションは出力ファイル、つまり「vastfomp 」コマンドにより OpenMP ディレクティブが挿入されたコードのファイル名を指定するオプションです。このオプションをつけない場合の出力ファイルは、元のソースコードの先頭に「V」がついたファイルが出力されます。今回の場合は「Vauto.f」というファイルが出力されます。

```
$ vastfomp - O VAST_auto.f auto.f
```

生成された OpenMP プログラムは以下のような内容になります。

```
more VAST_auto.f
           program aut_test
           parameter(n=1000)
C...Translated by Crescent Bay Software VAST      7.1F  12:13:52   1/ 8/2004
           real a(n,n),b(n,n)
!$OMP PARALLEL
!$OMP DO
      do j = 1, 1000
         do i = 1, 1000
            a(i,j) = 1.
            b(i,j) = 1.
         end do
      end do
!$OMP DO
      do j = 1, 1000
         do i = 1, 1000
            a(i,j) = a(i,j)*b(i,j)
         end do
      end do
!$OMP END PARALLEL
           do j=1,n
           do i=1,n
               print *,a(i,j)
           enddo
           enddo
           stop
           end
```

あとは通常の OpenMP プログラムと同様にコンパイル、実行します。

```
$ icc -openmp VAST_auto.f -o exec
$ setenv OMP_NUM_THREADS 4
$ ./exec
```

以下に、vastomp コマンドの主要なオプションを示します。詳細に関しては、VAST/toOMP のマニュアルをご参照ください。

オプション	内容
-O ファイル名	出力（生成）ファイル名の指定。指定しないと「V 元のソースファイル名」で生成される
-l ファイル名	ファイル名で指定した名前で最適化および並列化の解析リストを出力する。解析リスト中のコードの意味は以下の通りです。 P：並列化された N：該当するループは最適化されなかった(最適化の対象外となった) V：最適化された R：unrollされた
-J	callingツリーで最下部レベルのサブルーチンをインライン展開した後、並列化を行う。ループ中にサブルーチンコールがある場合は、通常は並列化されないが、インライン展開後、依存性がないことが分かれば並列化可能となる場合があるため、比較的小さなサブルーチンがループ中にある場合に有効となる。
-I サブルーチン名	多重ループ中にサブルーチンsub1とsub2があって、並列化できないような場合、sub1とsub2をインライン展開後に並列化を行う。-Iで指定されたサブルーチンのみをインライン展開した後、並列化を行う。
-C サブルーチン名	ループ中にサブルーチンsub1とsub2がある場合、それらのサブルーチンに並列化を阻害する依存性がないことを明示的に示すことによって、サブルーチンを含んだ形でループを並列化します。サブルーチンはインライン展開されません。 このオプションは、依存性がないことが分かっている場合にのみ利用ですますので、十分に注意の上使用してください。

注意事項

「-C」オプションは、もし依存性があった場合、計算結果が異なってしまいます。依存性に十分に注意の上オプションを指定してください。また、「-J」や「-I」オプションの場合などでも計算結果についてはオリジナル版と検証しながら使用することを推奨いたします。

6　並列プログラムの実行：dplace コマンド

dplace コマンドは、複数のプロセスをアイドル状態の CPU にラウンドロビン方式で割り当てます。並列プログラムの実行時には、dplace コマンドを指定されることを推奨します。自動並列化／OpenMP プログラムと MPI プログラムでオプションが異なりますのでご注意ください。数字のあるオプションですが、実行 CPU 数がいくつであっても変わりません。

6.1　自動並列化／OpenMP プログラムでの dplace コマンド

環境変数　OMP_NUM_THREADS に実行プロセス数を指定してください。デフォルトでは dplace コマンドの-x2 オプションを使用してください。

例）

```
% setenv OMP_NUM_THREADS 4
% dplace  -x2 ./a.out
```

6.2　MPI プログラムで deplace コマンド

dplace コマンドの-s1 オプションを使用してください。

例）

```
% mpirun -np 4 dplace  -s1 ./a.out
```

6.3　dplace のオプション

dplace のオプション、-x -s ともに CPU に配置しないプロセスを指定するものです。

オプション	内容
-x	CPU に配置しないプロセスを指定します。ビットマスクになっています。例えば 6 を指定すると、2 番目と 3 番目のプロセスを CPU に配置しません。 自動並列化／OpenMP プログラムを実行すると、OMP_NUM_THREADS で指定した数のプロセスの他に 2 つの管理プロセスが起動されます。-x2 オプションはこの2つの管理プロセス以外のプロセスを CPU に割り当てることを指示するものです。
-s	-s オプションを指定すると最初の n 個のプロセスを CPU に配置しません。-s1 オプションで MPI プログラムは mpirun で N+1 個のプロセスが起動されます。最初のプロセスは実際にはインアクティブな状態です。-s1 を指定してインアクティブなプロセスを CPU に割り当てません。

7 デバッグ

7.1 デバッグオプション

オプション	内容
-mp, -mp1	IEEE754 規格に則った浮動小数点演算コードを生成します。例えば、－mp オプションは乗加算命令を採用しません。－mp1 オプションは－mp オプションよりも最適化を進めます。
-IPF_fltacc	浮動小数点の精度を保つような最適化を行います。つまり、精度を落とす可能性がある最適化は無効にします。
-ftz	ゼロ割などの例外処理で不正な値になったとき、値をゼロにして計算を続行します。 注意：最適化オプション-O2 のときは-ftz は無効ですが、－O3 で有効になります。例外処理の発生するようなプログラムでは、-O2 ではエラーになるが、-O3 ではエラーにならないという場合があります。
-r8	real/compelx 型で宣言された変数を real*8/complex*16 型の変数として取り扱います。
-i8	integer 型で宣言された変数を integer*8 型の変数として取り扱います。

7.2 デバッグツール

デバッグツールをお使いになる場合には、コンパイル時に「-g」オプションをご指定ください。-g を指定するとデバッグシンボルを有効にします。

7.2.1 gdb

GNU プロジェクトのデバッガです。C、C++、Fortran95 で書かれたプログラムのデバッグに使用できます。

7.2.2 idb

Intel 製のデバッガです。C、C++、Fortran77、Fortran90 で書かれたプログラムのデバッグに使用できます。現時点では並列プログラムのデバッグには使用できません。詳細はマニュアルをご覧ください。

7.2.3 ddd

コマンドラインデバッガに接続する GUI です。起動後、画面の各ペインに次の情報が表示されます。

- 配列表示
- ソースコード
- 逆アセンブルコード
- コマンドを入力するウィンドウ

8 科学技術計算ライブラリ

8.1 SCSL ライブラリ

科学技術計算ライブラリとしては SCSL ライブラリをご利用頂けます。

以下に SCSL に含まれる内容を示します。

ライブラリ名	内容
BLAS	基本的な線形計算
LAPACK	密行列計算パッケージ
FFT	1 次元、2 次元、3 次元の FFT
psldlt, psldu	直接法スパースソルバー

8.1.1 SCSL 利用方法

コンパイル時に以下のように SCSL のオプションを付加し、SCSL ライブラリをリンクさせてください。

オプション	内容
-lscs	SCSL 逐次版をリンクします。
-lscs_mp	SCSL 並列版をリンクします。

次ページにてライブラリルーチンへのリンク方法を示します。逐次版、並列版も基本的に利用方法は同様です。

例）

```
% cat ex7.f
    program ex7
    integer       M, N, LDA, NRHS, LDB, INFO
    character   TRANS
    parameter   (M=3, N=3, LDA=3, NRHS=1, LDB=3, TRANS= 'N' )
    integer       IPVT(n)
    real            A (LDA,LDA), B(LDB), X(n)
C    A=( 1.0    3.0    3.0    )
C         1.0    3.0    4.0    )
C         1.0    4.0    3.0    )
C
C         B=( 1.0
C             4.0
C            -1.0   )
data   A/1.0, 1.0, 1.0, 3.0, 3.0, 4.0, 3.0, 4.0, 3.0/
data   B/1.0, 4.0, -1.0/
C   compute an LU factorization of a general M-by-N matrix A
        call  SGETRF(M, N, A, LDA, IPVT, INFO)
C   solve a system of linear equations A*X=B
        call  SGETRS(TRANS, N, NRHS, A, LDA, IPVT, B, LDB, INFO)
        do  i=1,n
            print *, B(I)
        enddo
        stop
        end
% ifort ex7.f -lscs
% ./a.out
-2.000000
-2.000000
  3.000000
```

なお、SCSL ライブラリの詳細に関しては、SCSL ライブラリのマニュアルをご参照ください

9 数値計算ライブラリ

9.1 IMSL ライブラリ

IMSL は、数値計算、統計解析用の Fortran ライブラリです。

以下に IMSL の数値計算機能、統計解析機能を示します。

数値計算機能	統計解析機能
● 線形連立方程式 ● 固有システム解析 ● 補間と近似 ● 積分と微分 ● 微分方程式 ● 変換 ● 非線形方程式 ● 最適化 ● マトリックス/ベクトル 操作 ● 特殊関数 ● ユーティリティ	● 基本統計 ● 回帰 ● 相関と共分散 ● 分散分析 ● カテゴリデータと離散データ解析 ● ノンパラメトリック統計 ● 適合性の検定 ● 時系列と予測 ● 多変量解析 ● 生存解析 ● 確率分布関数とその逆関数 ● 乱数発生

なお、並列化対応している関数は以下の通りとなります。

数値計算機能	統計解析機能
● 連立一次方程式 ● 固有値解析 ● 補完と近似 ● 微分方程式 ● 変換 ● 最適化 ● 基本的な行列とベクトル演算	● 回帰 ● カテゴリー・データと離散データの解析 ● 時系列解析と予測 ● 共分散構造と因子分析 ● 判別分析 ● 次元尺度法 ● 乱数発生

9.1.1 IMSL 利用方法

コンパイルやリンク時に、下記のように指定して下さい。

STATIC リンクの例

```
% icc -o  test test.f  $LINK_F90_STATIC
```

SHARED リンクの例

```
$ ifort -o  test test.f  $LINK_F90_SHARED
```

詳細に関しては IMSL のマニュアルをご参照ください

10 物理乱数発生ボードの利用方法

東芝製物理乱数発生ボードを並列計算機のうち ismaltx4 に 16 枚搭載してます。
利用方法はブートストラップシステムの物理乱数発生ボードに準じています。
以下に詳細を述べます。

10.1 物理乱数発生ボード

10.1.1 特徴

- 一様乱数(32bit 符号付整数、単精度浮動少数、倍精度浮動少数)
- 読み出すデータ型は以下の 3 種類

区間[-2^31, 2^31) の 32-bit 符号付整数型
区間[0,1)の単精度浮動小数点型
区間[0,1)の倍精度浮動小数点型

- 高速転送(最大 133Mbyte/sec)

10.1.2 仕様

作成中

10.2 RMWS-2 ライブラリ

RMWS-2 ライブラリは、C 言語および FORTRAN 言語から物理乱数発生ボードを使用するためのインターフェースを提供します。
ユーザーは実装されているボードの枚数を考慮する必要がなく、複数のボードを並列的に使用する機能を持っています。

10.2.1 コンパイル及びリンクについて

- 環境設定

読み込み用ライブラリは以下の場所にインストールしました。
ismaltx:/usr/local/lib/libtrmlib.so
ismaltx4:/usr/local/lib/libtrmlib.so
ライブラリを利用するためには環境変数 LD_LIBRARY_PATH に上記のパスが含まれている必要があります。システムのデフォルトの環境では既に設定されています。

- 使い方

gcc [option | filename] -ltrmlib

g77 [option | filename] -ltrmlib

その他のコンパイラの使用についても、同様に-trmlib でリンクを行なうことで利用可能です。

10.2.2 C 用ライブラリ

int RM_Init(void)	バッファの初期化を行います
int RM_Finish(void)	バッファのクリーンを行います
int RM_read_int(int*, int)	32bit 符号付整数の読み込みを行ないます
int RM_read_float(float*, int)	単精度浮動小数点の読み込みを行ないます
int RM_read_double(double*, int)	倍精度浮動小数点の読み込みを行ないます

[注意点]

読み込み関数の引数について

第 1 引数 読み込み用配列を指定します。

第 2 引数 読み込み個数(>0)を指定します。

返り値については読み込み成功時には "0"を失敗時には "-1"を返します。

読み込み関数は"RM_Init" と"RM_Finish" の間にはさんで使用します。これはプログラムの最初と最後に一度だけ行います。

10.2.3 Fortran 用ライブラリ

rminit(integer)	バッファの初期化を行います
rmfinish(integer)	バッファのクリーンを行います
rmri(integer, integer, integer)	32bit 符号付整数の読み込みを行います
rmrf(integer, real*4, integer)	単精度浮動小数点の読み込みを行います
rmrd(integer, real*8, integer)	倍精度浮動小数点の読み込みを行います

[注意点]

読み込み関数の引数について

第 1 引数 チェックサム(成功時 0, 失敗時 -1)

第2引数 読み込み用配列を指定します。

第 3 引数 読み込む個数(>0)を指定します。

"rminit","rmfinish"の引数はチェックサム(成功時 0, 失敗時 -1)

読み込み関数は "rminit" と"rmfinish" の間にはさんで使用します。これはプログラムの最初と最後に一度たげ使用します。また、引数はチェックサムになっています(成功時 0, 失敗時 -1)

11 時間計測関数

11.1 Fortran 関数(dclock, etime)

dclock 関数、etime 関数をご紹介します。どちらもリンク時には 「-Vaxlib」 を指定してください。「-Vaxlib」オプションは VAX/VMS FORTRAN の拡張組み込み関数を有効にします。

11.1.1 dclock 関数

経過時間を秒単位で返します。real*8 型の値を返します。

```
real*8  dclock, time1
time1 = dclock()
```

例）

```
% cat ex2_1.f
        program   ex2_1
        real*8    dclock, t1, t2
        t1 = dclock()
        call  sub()
        t2 = dclock()
        print  *, t2 -  t1
        end
```

11.1.2 etime 関数

etime 関数は、プログラムの始めからの経過時間(ユーザ時間＋システム時間)を秒単位で返します。また配列の 1 番目の要素にユーザ時間 2 番目の要素にシステム時間を返します。

```
real*4  etime, time4
real*4  tarray(2)
time4 = etime(trray)
```

例）

```
% cat ex2_2.f
    program   ex2_2
    real*4    etime, time4, tarray(2),t1,t2
    t1 = etime(tarray)
    call sub
    t2 = etime(tarray)
    print *, t2 -  t1, tarray(1), tarray(2)
    end
```

11.2 C 関数(gettimeofday)

gettimeofday は、1970 年 1 月 1 日 00:00:00 からの経過時間をマイクロ秒単位で計測します。

返り値は整数型で、実行に成功すれば 0、失敗すると-1 を返します。

```
      #include<sys/time.h>
      struct timeval tp;
      struct timezone tzp;
      gettimeofday(&tp,&tzp);
```

※ timeval 構造体は/usr/include/sys/time.h に定義されています。

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

※ timezone 構造体の使用は廃止 (obsolete) されました。 Linux では使用されません。

例）

elapsed.c は時間を秒単位で返すルーチンです。

```
% cat elapsed.c
#include<sys/time.h>
#include<stdio.h>

double elapsed()
 {
    struct timeval tp;
    struct timezone tzp;
    gettimeofday(&tp,&tzp);
    return ( (double) tp.tv_sec + (double) tp.tv_usec * 1.e-6 );
}
```

elapsed.c を使ったプログラム例です。

```
% cat ctime.c
#include<stdio.h>
#include<stdlib.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)
}

%  icc ctime.c celapsed.c
% a.out
1.337051e-03
```

12 性能解析ツール

プログラムの性能向上を妨げている要因を知りたいとき、プログラムのどこで実行時間がかかっているのかを知りたいときなどに有用な2つの性能解析ツールをご紹介します。どちらも実行時に指定しますと、実行終了後に解析結果を出力します。

lipfpm コマンドは、プログラム全体のふるまいを調べるときにお使い頂けます。命令の実行、変数のロード／ストア、キャッシュミスなどの現象をイベントと呼んでいますが、lipfpm はプログラム実行全体のイベントカウントを知るためのツールです。

histx コマンドはプログラムのどのルーチンで実行時間がかかっているのか、あるいはソースコードのどの行の実行に時間がかかっているのかを調べるときにお使い頂けます。

MPI プログラムの可視化と解析を行う場合は、Vampir および VampirTrace をご使用ください。

12.1 lipfpm （Linux IPF Performance Monito）コマンド

Itanium2 には 4 つのイベントカウンタがあるので同時に 4 つのイベントをカウントすることができます。 デフォルトでは CPU サイクルをカウントします。再コンパイルは必要ありません。

例）

```
% lipfpm ./a.out
   1.000000
lipfpm summary
======= =======
CPU Cycles.................................................. 19934472
```

次は 3 次キャッシュにおけるキャッシュミスのカウントの例です。イベントの名前を-e オプションと共に指定してください。

例）

```
% lipfpm -e L3_MISSES ./a.out
   1.000000

lipfpm summary
======= =======
L3 Misses................................................... 126448
CPU Cycles.................................................. 19944147
```

オプションを付けずに libfpm を実行するとオプションの内容などが表示されます。 -i オプションで全イベントの名前がインターラクティブにアルファベット順に表示されます。

- space で次のイベント、backspace で前のイベントが表示されます。
- enter でイベントの選択ができます。Esc で抜けます。

12.1.1 MFLOPS 値計測

MFLOPS 値を計測するには、カウントするイベントとして FP_OPS_RETIRED を指定してください。

例）

```
$ lipfpm -e FP_OPS_RETIRED ./a.out

lipfpm summary
======= =======
Retired FP Operations....................................... 3002108147
CPU Cycles.................................................. 3380038606
Average FLOPS per Cycle..................................... 0.888188
```

並列計算機の CPU は動作周波数 1.3GHz ですので、「Average FLOPS per Cycle」*1300 で MFLOPS 値になります。

12.1.2　自動並列化／OpenMP プログラムの解析

並列プログラムを解析するときは、-f オプションを追加してください。また、-o オプションで結果を格納するファイルを指定してください。結果ファイルが指定されないと、各スレッドがばらばらに標準エラー出力に結果を書き出してしまいます。自動並列化／OpenMP プログラムを実行すると、OMP_NUM_THREADS で指定した数のプロセスの他に 2 つの管理プロセスが起動されます。そのため解析結果ファイルも実行スレッド数＋2 個作られます。

例）

```
% setenv OMP_NUM_THREADS 4
% lipfpm -f -o cnt ./a.out
   1.000000
% ls cnt*
cnt.a.outpara.15863  cnt.a.outpara.15865  cnt.a.outpara.15867
cnt.a.outpara.15864  cnt.a.outpara.15866  cnt.a.outpara.15868
% cat cnt*
lipfpm summary
======= =======
CPU Cycles.................................................. 26742864
lipfpm summary
======= =======
CPU Cycles.................................................. 70897
lipfpm summary
======= =======
CPU Cycles.................................................. 58572
lipfpm summary
======= =======
CPU Cycles.................................................. 3935939
lipfpm summary
======= =======
CPU Cycles.................................................. 3755507
lipfpm summary
======= =======
CPU Cycles.................................................. 4086317
```

dplace と共に実行するとき次のように histx を指定します。

```
% dplace - x5 lipfpm -f -o 解析結果ファイル名　./a.out
/* dplace のオプション -x の値は 5 です　*/
```

12.1.3 MPI プログラムの解析

MPI プログラムも-f オプションを指定してください。

例）

```
% mpirun -np N dplace -s3 lipfpm -f -o out a.out
```

サンプルプログラム （次ページも含む）

```
% cat test.f
          program main
          parameter(n=800)
          real a(n,n),b(n,n),c(n,n)
          call init(n,a,b,c)
          call matmul(n,a,b,c)
          print *,c(1,1)
          stop
          end
c
c******* initialization **************
c
          subroutine init(n,a,b,c)
          integer n
          real a(n,n),b(n,n),c(n,n)
          do j=1,n
          do i=1,n
             a(i,j)=1.
             b(i,j)=1.
             c(i,j)=0.
          enddo
          enddo
          return
          end
c
c******* calculation *****************
c
          subroutine matmul(n,a,b,c)
          integer n
          real a(n,n),b(n,n),c(n,n)
          do k=1,n
          do j=1,n
          do i=1,n
             c(j,k)=c(j,k)+a(k,i)*b(i,j)
          enddo
          enddo
          enddo
          return
          end

最適化オプション -O2 でコンパイルし、MFLOPS 値を計測。
%  ifort test.f
%  lipfpm -e FP_OPS_RETIRED ./a.out
   800.0000

lipfpm summary
======= =======
Retired FP Operations..................................... 1025923874
CPU Cycles................................................ 4239252657
Average FLOPS per Cycle................................... 0.242006


※  242MFLOPS
```

最適化オプション -O3 でコンパイルし、MFLOPS 値を計測。

```
%  ifort -O3 test.f
%  lipfpm -e FP_OPS_RETIRED ./a.out
   800.0000

lipfpm summary
======= =======
Retired FP Operations..................................... 1024003874
CPU Cycles................................................ 593508649
Average FLOPS per Cycle................................... 1.72534
```

※ 1725MFLOPS

12.2 histx (HISTogram eXecution)コマンド

histx コマンドは、プログラムのどこで実行時間が消費されているか、どこでイベントが発生しているかを知るためのツールです。プログラムの実行が終わると解析結果ファイルが作られるので、iprep コマンドを使って整形してください。

なお、histx は histx は静的ライブラリがリンクされているバイナリを解析することができないのでご注意ください。

例）

```
% histx -o out ./a.out
/* －o オプションは必須です。解析結果を収めるファイル名の先頭にこの名前が付きます。*/
% ls out*
out.a.out.16066
% iprep out.a.out.16066
Functions sorted by count
========= ====== == ====
       Count          Excl. %   Incl. %   Name
-------------------   -------   -------   --------------------
                1562    40.815    40.815   a.out:yaver_
                1501    39.221    80.037   a.out:zaver_
                 737    19.258    99.294   a.out:xaver_
                  26     0.679    99.974   a.out:main
ソースコードのライン毎の解析を行いたいときは、-g オプションを付けてコンパイルしてください。histx には、-l オプションを指定してください。



% ifort -g -o a.out  prog.f
% histx -l -o out ./a.out
% ls out*
out.a.outg.17204
% iprep out.a.outg.17204
Source lines sorted by count
====== ===== ====== == ====

       Count          Excl. %   Incl. %   Name
-------------------   -------   -------   --------------------
               41185    32.678    32.678   [prog.f:82] (a.out)
               41059    32.578    65.255   [prog.f:63] (a.out)
               39580    31.404    96.660   [prog.f:44] (a.out)
                1247     0.989    97.649   [prog.f:64] (a.out)
                1206     0.957    98.606   [prog.f:83] (a.out)
                1183     0.939    99.545   [prog.f:45] (a.out)
                 136     0.108    99.652   [prog.f:18] (a.out)
                 127     0.101    99.753   [prog.f:43] (a.out)
        ...   ...   ...

Count : サンプリングのカウント数
Excl. : 全体に対する割合い
Incl. : 積算
Name  : []の中はソースファイル名と行番号

オプションを指定せずに histx コマンドを実行すると、使い方の概要が表示されます。
histx はコールスタックのサンプリングを採ることもできます。「付録 histx を使ったコールスタックの解析」を参照ください。
```

12.2.1 histx を用いた自動並列化／OpenMP プログラムの解析

自動並列化／OpenMP コードを histx コマンドを使って解析するときは -f オプションを指定してください。lipfpm コマンドと同様に実際の実行スレッド数＋2 の解析結果ファイルが作成されます。

```
% histx -f -o out ./a.out
```

dplace と共に実行するとき次のように histx を指定します。

```
% dplace - x5 histx -f -o 解析結果ファイル名  ./a.out
/* dplace のオプション -x の値は 13 です  */
```

12.2.2 histx を用いた MPI プログラムの解析

MPI プログラムを解析するときは -f オプションを指定してください。

```
% mpirun -np N dplace -s3 histx -f - o 解析結果ファイル名 ./a.out
```

12.2.3 histx を用いたコールスタックの解析

```
% histx - o cs - s callstack10 ./prog
```

csrep コマンドを使って整形します。標準出力に表示されます。

```
% csrep < cs.prog.12345 > butterfly.12345
```

または、

```
% csrep < cs.prog.* > butterfly.all
```

例）

コールスタックに多く現われたルーチンから表示されます。

```
98.1% libc.so.6.1:__libc_start_main [libc-start.c:129]
 97.5% a.out:_start
 48.7% a.out:main [prog.c:93]
 48.7% a.out:main [prog.c:70]
 36.8% a.out:f1 [prog.c:23]
 34.4% a.out:f2 [prog.c:33]
 31.7% libm.so.6.1:cos
 17.9% libm.so.6.1:asin
 14.8% libm.so.6.1:acos
```

各ルーチン毎に、コール元のルーチン、コール先のルーチンの情報が表示されます。

```
  100.00% ( 48.74%) libc.so.6.1:__libc_start_main [libc-start.c:129]

  ....... ( 48.74%) a.out:main [prog.c:70]

   75.43% ( 36.76%) a.out:f1 [prog.c:23]
   24.57% ( 11.97%) a.out:f1 [prog.c:19]
```

上記の例で注目しているのは「main」です。上下が空行になっていることでわかります。main よりも上に表示されているのは、main をコールしたルーチン、下に表示されているのは、main がコールしたルーチンです。

サンプルプログラム（3 ページを含む）

```
% cat test.f
          program main
          parameter(n=800)
          real a(n,n),b(n,n),c(n,n)
          call init(n,a,b,c)
          call matmul(n,a,b,c)
          print *,c(1,1)
          stop
          end
c
c******* initialization **************
c
          subroutine init(n,a,b,c)
          integer n
          real a(n,n),b(n,n),c(n,n)
          do j=1,n
          do i=1,n
             a(i,j)=1.
             b(i,j)=1.
             c(i,j)=0.
          enddo
          enddo
          return
          end
c
c******* calculation ****************
c
          subroutine matmul(n,a,b,c)
          integer n
          real a(n,n),b(n,n),c(n,n)
          do k=1,n
          do j=1,n
          do i=1,n
             c(j,k)=c(j,k)+a(k,i)*b(i,j)
          enddo
          enddo
          enddo
          return
          end

行毎の解析を行なうために、-g　オプションでコンパイル。
histx には、-l　オプションをつける。

% ifort -O3 -g test.f
%  histx -l -o OUT ./a.out
   800.0000
```

```
% iprep OUT.a.out.1793
Functions sorted by count
========= ====== == ====

       Count           Excl. %   Incl. %   Name
-------------------   -------   -------   --------------------
                 515    93.466    93.466   a.out:matmul_ [test.f:27]
                  35     6.352    99.819   a.out:init_ [test.f:12]
                   1     0.181   100.000   libc.so.6.1:flockfile

Source lines sorted by count
====== ===== ====== == ====

       Count           Excl. %   Incl. %   Name
-------------------   -------   -------   --------------------
                 620    43.448    43.448   [test.f:33] (a.out)
                 364    25.508    68.956   [test.f:38] (a.out)
                 362    25.368    94.324   [test.f:32] (a.out)
                  29     2.032    96.356   [test.f:17] (a.out)
                  23     1.612    97.968   [test.f:18] (a.out)
                  11     0.771    98.739   [test.f:19] (a.out)
                  10     0.701    99.439   [test.f:16] (a.out)
                   7     0.491    99.930   [test.f:23] (a.out)
                   1     0.070   100.000   [soinit.c:30] (libc.so.6.1)
%
```

自動並列化オプションを指定してコンパイル。

```
% ifort -parallel -O3 -g test.f

test.f(30) : (col. 0) remark: LOOP WAS AUTO-PARALLELIZED.

% setenv OMP_NUM_THREADS 2
% histx -l -o OUT ./a.out
   800.0000
% ls -lt OUT*
-rw-r--r--    1 sgi      sgi           947  7ｷ 17 20:59 OUT.a.out.1927
-rw-r--r--    1 sgi      sgi            46  7ｷ 17 20:59 OUT.a.out.1928
-rw-r--r--    1 sgi      sgi            46  7ｷ 17 20:59 OUT.a.out.1929
-rw-r--r--    1 sgi      sgi           301  7ｷ 17 20:59 OUT.a.out.1930
```

環境変数 OMP_NUM_THREADS で指定した数＋２の解析結果ファイルが生成されますが、
そのうち解析結果があるのは OMP_NUM_THREADS で指定した数のファイルです。

```
%  iprep OUT.a.out.1927
Functions sorted by count
========= ====== == ====

       Count           Excl. %   Incl. %   Name
-------------------   -------   -------   --------------------
                1616    95.396    95.396   a.out:matmul_ [test.f:27]
                  35     2.066    97.462   a.out:init_ [test.f:12]
                  27     1.594    99.055   a.out:__kmp_wait_sleep
                   8     0.472    99.528   a.out:__kmp_yield
                   3     0.177    99.705   a.out:__kmp_static_yield
                   1     0.059    99.764   a.out:__kmp_ia64_pause
                   1     0.059    99.823   ld-linux-ia64.so.2:strcmp
                   1     0.059    99.882   libc.so.6.1:sched_yield
                   1     0.059    99.941   a.out:memset
                   1     0.059   100.000   a.out:f_fioinit

Source lines sorted by count
====== ===== ====== == ====

       Count           Excl. %   Incl. %   Name
```

```
-------------------   -------   -------   --------------------
                1597    40.025    40.025   [test.f:38] (a.out)
                1382    34.637    74.662   [test.f:32] (a.out)
                 929    23.283    97.945   [test.f:33] (a.out)
                  23     0.576    98.521   [test.f:17] (a.out)
                  19     0.476    98.997   [test.f:18] (a.out)
                  14     0.351    99.348   [test.f:19] (a.out)
                  14     0.351    99.699   [test.f:16] (a.out)
                  12     0.301   100.000   [test.f:23] (a.out)
%  iprep OUT.a.out.1930
Functions sorted by count
========= ====== == ====

       Count           Excl. %   Incl. %   Name
-------------------   -------   -------   --------------------
                1644    99.939    99.939   a.out:matmul_ [test.f:27]
                   1     0.061   100.000   a.out:__kmp_wait_sleep

Source lines sorted by count
====== ===== ====== == ====

       Count           Excl. %   Incl. %   Name
-------------------   -------   -------   --------------------
                1628    40.997    40.997   [test.f:38] (a.out)
                1412    35.558    76.555   [test.f:32] (a.out)
                 931    23.445   100.000   [test.f:33] (a.out)
```

12.3 Vampir および VampirTrace を用いた MPI プログラムの解析

Vampir は、MPI プログラムの可視化と解析を行うツールです。Vampirtrace は、MPI のプロファイル取得ライブラリです。使用は、Vampirtrace のライブラリをリンクし、プログラム実行時に、MPI に関する情報を取得します。使用方法は、以下のようになります。なお、Vampir/Vampirtrace は、ismaltx で利用することができます。

利用方法

以下のような MPI プログラムを解析する例を示します。今回は C 言語のプログラムですが Fortran プログラムでも、オプション等は同様に利用できます。

テストプログラム

```
cat myname.c
#include <mpi.h>
#include <stdio.h>
main(argc, argv)
int        argc;
char       *argv[];
{          int        num_procs;
           int        my_proc;
           MPI_Init(&argc, &argv);
           MPI_Comm_size(MPI_COMM_WORLD, &num_procs);
           MPI_Comm_rank(MPI_COMM_WORLD, &my_proc);
           printf("I am process %d.   / Total processes: %d¥n",
                                  my_proc,num_procs);
           MPI_Finalize();
}
```

オブジェクトファイルの作成

まず、オブジェクトファイルを作成します

```
$ icc -c myname.c
```

Vampirtrace のライブラリをリンクさせる

以下の赤字で示されたオプションを追加することにより、Vampirtrace のライブラリをリンクさせます

```
$ icc myname.o -L/usr/local/VT/lib -lVT -lmpi -o myname
```

MPI プログラムの実行

上記で作成された実行ファイルを run 実行します。実行が完了すると、トレースファイルが、出力されます。

```
$ mpirun -np 2 ./myname
I am process 1.  / Total processes: 2
I am process 0.  / Total processes: 2
[0] Vampirtrace INFO: Writing tracefile myname.stf in /home/sgi/yamaguchi/test_program/mpi
```

今回の例で作成されるトレースファイルは以下の通りです。

```
myname.stf.sts 、 myname.stf.pr.0.anc 、 myname.stf.pr.0 、 myname.stf.msg.anc 、 myname.stf.msg 、 myname.stf.gop.anc 、 myname.stf.pr.0 、 myname.stf.msg.anc 、 myname.stf.msg 、myname.stf.gop.anc 、myname.stf.gop 、myname.stf.frm 、myname.stf.dcl 、 myname.stf、myname.prot
```

プロファイルの参照

ご利用の端末に X サーバの設定を行い、vampir を実行するホストで環境変数「DISPLAY」を設定した後、以下を実行します。

```
$  export DISPLAY=XXXX:0
※  csh 系は setenv DISPLAY XXXX:0 となります
$ vampir ./myname.stf
```

GUI の利用方法の詳細や、各種オプションの詳細は、Vampir、Vampiretrace のマニュアルをご参照願います。

13 バイナリデータの取り扱い

Fortran プログラムにおけるバイナリデータファイルの入出力を対象とした、エンディアンの変換方法を説明します。

並列計算機ではバイナリデータの形式としてリトルエンディアンを採用しています。エンディアン形式変換機能を使って、ビッグエンディアン形式のファイルを読み込む、あるいはビッグエンディアン形式のファイルを書き出すといった処理が可能になります。
環境変数 F_UFMTENDIAN にファイルのユニット番号を指定しますと、該当のユニット番号のデータに対してプログラム実行時に次のような変換を行います。

- READ を実行するとき:ビッグ→リトル
- WRITE を実行するとき:リトル→ビッグ

環境変数のパラメータ(空白は不可)

```
% setenv F_UFMTENDIAN MODE | [MODE; ]  EXCEPTION
MODE = big | little
EXCEPTION = big:ユニット番号 | little:ユニット番号 | ユニット番号
```

MODE はファイルの形式を指定します。little を指定すると変換を行いません。big を指定すると変換を行います。省略すると little が設定されます。EXCEPTION には MODE 以外の形式を持つファイルを指定します。

- ユニット番号 10 と 20 のファイルだけを変換したいとき

```
% setenv F_UFMTENDIAN 10,20
```

全てのファイルはリトルエンディアン形式であることが前提で、10 と 20 だけがビッグエンディアン形式を採用するということを設定しています。ユニット番号 10 と 20 のファイルは READ/WRITE 文によって変換されます。

- 全てのファイルを変換したいとき

```
% setenv F_UFMTENDIAN big
```

READ 文によって入力ファイルはビッグ→リトル変換が行われ、WRITE 文によってリトル→ビッグ変換が行われます。

- 上記の条件に加えてユニット番号 8 だけ変換したくないとき

```
% setenv F_UFMTENDIAN ” big;little:8”
```

MODE で big が指定されているので、全てのファイルについて変換されます。しかし、ユニット番号 8 のファイルだけは変換が行われません。

- ユニット番号 10 から 19 までのファイルを変換したいとき

```
% setenv F_UFMTENDIAN 10-19
```

注意:ここで紹介した操作は C/C++プログラムには適用できません。C/C++プログラムで作られたデータファイルを変換する場合には、それぞれ変換プログラムが必要です。

14 PGPLOT の利用方法

並列計算機では、様々なグラフィックデバイス上に図やグラフを描くことができる科学技術用グラフィックライブラリとして PGPLOT をご利用いただけます。

14.1 PGPLOT を用いたプログラムのコンパイル方法

下記の例では、PGPLOT ライブラリを用いたサンプルプログラム pgdemo.f をコンパイルし、そこで作成された実行ファイル pgdemo を実行しております。

```
% g77 -o pgdemo pgdemo.f -L/usr/local/pgplot -L/usr/X11R6/lib -fno-backslash -lpgplot -
 lX11
%./pgdemo    //プログラムの実行はX Window Systemが使用可能な環境でご利用ください。
```

ここで使用されているコンパイルオプションの詳細は次のようになります。

オプション	説明
-o *file*	出力先、及び出力ファイル名を *file* に指定します。
-l *library*	名前が *library* であるライブラリをリンク時に使用します。
-L *dir*	ディレクトリ *dir* を ‘-l’ による検索が行なわれるディレクトリのリストに加えます。
-fno-backslash	バックスラッシュを処理しません。 PGPLOT のプログラムによっては必要になります。

並列計算機では、PGPLOT のサンプルプログラムが/usr/local/pgplot/example に格納されておりますのでご利用下さい。

15 各種マニュアルについて

主な各種マニュアルは以下のディレクトリに格納されております。

- Compiler について　ismaltx:/home0/doc/compiler 以下
- Openmp について　ismaltx:/home0/doc/openmp 以下
- MPI(MPT)について　ismaltx:/home0/doc/mpi 以下
- プログラムミングガイドについて　ismaltx:/home0/doc/ programming_guide 以下
- SCSL について　ismaltx:/home0/doc/scsl 以下
- IMSL について　ismaltx:/home0/doc/imsl 以下
- VAST について　ismaltx:/home0/doc/vast 以下
- 講習会資料について　ismaltx:/home0/doc/workshop 以下
- 利用の手引きについて（本資料）　ismaltx:/home0/doc/user_guide 以下