UNIX システム利用入門
中級編

− Fortran プログラムと NQS の利用方法 −

システム運用掛　宮嵜 洋

1　システム概要

1.1　システム構成

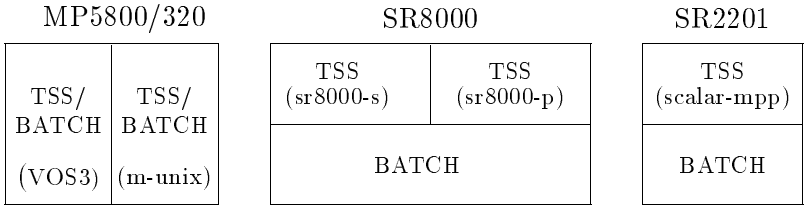
東京大学情報基盤センターの大型計算機システムにはベクトル並列型スーパーコンピューター、超並列型スーパーコンピューター、汎用大型計算機があり、UNIX システムにおいては、それぞれ以下のように利用が可能です。

ベクトル並列型スーパーコンピューター（OS: HI-UX/MPP(UNIX)）
　HITACHI　SR8000（ホスト名 **sr8000-s, sr8000-p**）
　　総ノード数：128　ノード演算性能：8GFLOPS
　　主記憶容量：1 ノードにつき 8GB　拡張記憶容量：24GB（主記憶の一部を統合して構成）
　　言語プロセッサー：FORTRAN77, FORTRAN90, C, C++
　　並列化支援：MPI, PVM, PARALLELWARE, HPF, リモート DMA 転送

超並列型スーパーコンピューター（OS: HI-UX/MPP(UNIX)）
　HITACHI　SR2201（ホスト名 **scalar-mpp**）
　　総ノード数：1024　ノード演算性能：256MFLOPS
　　主記憶容量：1 ノードにつき 256MB
　　言語プロセッサー：FORTRAN77, FORTRAN90, C, C++
　　並列化支援：MPI, PVM, PARALLELWARE, HPF, リモート DMA 転送

汎用大型計算機（OS:VOS3/FS, HI-OSF/1-MJ(UNIX)）
　HITACHI　MP5800/320（ホスト名 **m-unix**）
　　（UNIX 用として）主記憶容量：2GB 拡張記憶容量：2GB
　　言語プロセッサー：FORTRAN77, C, C++

汎用大型計算機（MP5800/320）は計算機資源を VOS3 と OSF/1（UNIX）に分割して各々別のシステムとして運用しています。ベクトル並列型スーパーコンピューター（SR8000）、超並列型スーパーコンピューター（SR2201）は計算機資源をインタラクティブ処理サブシステム（TSS）、バッチ処理サブシステム（BATCH）に分割しており、ジョブの形態によって処理を分担するというサブシステム構成をとっています。現在、ログインセッションは TSS 、バッチジョブは BATCH で実行されるよう以下の構成で運用しています。



SR8000 では利用者がノードを共有（要素並列実行不可）して利用する形態とノードを排他的に占有（要素並列実行可能）して利用する形態があるため、ログインできるサブシステム（TSS）として 2 つのホスト名 sr8000-s、sr8000-p を用意しています。

1.2 磁気ディスク

UNIX システムで利用者が使用できるディスク（ファイルシステム）として以下のディレクトリーを用意しています。

/home

各利用者のホームディレクトリー。ここに作成されたファイルは利用者が削除するまで保存されます。（VOS3 の長期データーセットに相当します。）

/short

利用者が作業用に利用するディレクトリー。ここに作成されたファイルは最終アクセスから 15 日後に削除されます。（VOS3 の短期データーセットに相当します。）

/para-io

利用者が大容量入出力に利用するディレクトリー。ストライピングファイル機構により、データーをブロック単位に分割して並列に入出力するので高速です。通常のファイルと同様にアクセスできますが、バッチジョブからのみ使用可能で 1 つの大容量ファイルを扱うときに有効です。ここに作成されたファイルは最終アクセスから 5 日後に削除されます。

/tmp

利用者およびシステムが短時間の作業用に利用するディレクトリーです。ここに作成されたファイルは作成終了後しばらくして（平均 1 時間以内。最低 30 分、最大 1 時間 30 分以内で）削除されます。

UNIX システムのディスク容量

<table>
<tr><td rowspan="2">m-unix</td><td colspan="2">sr8000-s　sr8000-p</td><td colspan="2">scalar-mpp</td><td rowspan="2">保存期間</td></tr>
<tr><td>TSS</td><td>NQS</td><td>TSS</td><td>NQS</td></tr>
<tr><td>/home
110GB</td><td colspan="2">/home
128GB</td><td colspan="2">/home
160GB</td><td>削除せず</td></tr>
<tr><td rowspan="3">/short/m-unix
35GB</td><td colspan="2">/short/sr8000-s
192GB</td><td colspan="2" rowspan="2">/short/single
80GB</td><td rowspan="3">１５日間</td></tr>
<tr><td colspan="2">/short/sr8000-p
192GB</td></tr>
<tr><td colspan="2">/short/sr8000-bt
192GB</td><td colspan="2">/short/multi
80GB</td></tr>
<tr><td>/tmp
500MB</td><td>/tmp
5GB</td><td>/tmp
5GB</td><td>/tmp
8GB</td><td>/tmp
8GB</td><td>３０分～
１時間半</td></tr>
<tr><td>なし</td><td>なし</td><td>/para-io
160GB</td><td>なし</td><td>/para-io
128GB</td><td>５日間</td></tr>
</table>

ファイルへのアクセスは使用するファイルシステムがローカルな磁気ディスクか、ネットワーク経由（NFS 経由）でマウントされているかによって入出力性能が異なります。以下にログインしているホスト、バッチジョブと各ファイルシステムとの関係を示します。

SR8000	sr8000-s	sr8000-p	バッチジョブ
/home	○	○	◎
/short/sr8000-s	◎	○	○
/short/sr8000-p	○	◎	○
/short/sr8000-bt	○	○	◎
/para-io	×	×	◎

SR2201	scalar-mpp	バッチジョブ
/home	○	◎
/short/single	◎	○
/short/multi	○	◎
/para-io	×	◎

◎ローカル　○ NFS 経由　×アクセス不可

現在、利用者毎のファイル使用量の上限値を設定しているディレクトリーは /home（および scalar-mpp の/short）です。上限値は利用者自身の宣言（newuser）により設定されます。その他のディレクトリーの上限値は制限していませんが、課金対象となっていないディレクトリーでも、限られた計算機資源を有効に利用するため、不要なファイルは削除するよう心掛けて下さい。

2　コンパイル、実行方法

2.1　コンパイラー – f77, f90

FORTRAN プログラムのコンパイルにおいて FORTRAN77 オブジェクト生成には f77 コマンドを、FORTRAN90 オブジェクト生成には f90 コマンドを使用します。(m-unix では FORTRAN90 をサポートしていないため f90 コマンドは使用できません。)

以下に f77, f90 コマンドの指定方法と主なオプションを示します。

```
f77 または f90
    [-c]
    [-o ロードモジュールファイル名 ]
    [-W0,' コンパイルオプション' ]
    [-Wl,' リンケージオプション' ]
    [-I インクルードディレクトリー名 ]
    [-L ライブラリーディレクトリー名 ]
    [-l ライブラリー名]
    [-i,{N|E}[,L][,U][,P][,LT] ]          … 言語使用の拡張
    [-32|64]                              … 32/64 ビットモード
    [-オプション]
    ファイル名 ..
```

使用例

```
% f77  -i,U  -W0,'vos3(nosymnchk)'   -lmatmpp   main.f   sub.f
```

FORTRAN プログラムのソースファイル名は通常 .f で終わる名前を使用します。コンパイルが正常に終了するとロードモジュール a.out が作成されます。(ロードモジュール名は -o オプションで指定できます。) ロードモジュールは実行ファイルとも呼ばれ、ロードモジュール名をコマンド同様に入力することでプログラムの実行を開始することができます。(-c オプションを指定した場合はオブジェクトモジュール.o が作成されます。)

いまソースプログラムの中身が、

```
% cat a.f
read(5,*) a,b
write(6,*) a*b,a/b
stop
end
```

であるファイル a.f があるとします。これをコンパイル、実行するには以下のようにします。

```
% f77 a.f                              …  a.out が作成される
% a.out
?
2 3
  6.00000000     0.66666627
%
```

装置番号 5, 6 番に対する入出力は、リダイレクト機能を用いることができます。たとえば、5 番の入力を a.data から読み込ませるには、

```
% a.out < a.data
  6.00000000     0.66666627
%
```

および 6 番の出力を a.result に書き出すには、

```
% a.out < a.data > a.result
```

とします。

2.2 オプションと機能

計算機の性能を最大限に引き出すため、コンパイラーにはハードウェアに適応したオブジェクトを生成する各種機能が備わっています。これらの機能は利用者がコンパイルオプションやコメント文のパラメーターで指示することにより利用できます。効果的に適用された場合には性能が大幅にアップします。

最適化 (m-unix, scalar-mpp, sr8000-s,p)

最適化とはプログラムの実行速度向上の目的で、演算子の変更、ループ構造の変換、演算順序の変更などをコンパイラーが自動的に行ない、レベルに応じた最適化オブジェクトを生成する機能です。

最適化機能とレベル（上位のレベルは下位の機能を含む）

- 原始プログラム通りのコンパイル、一文単位での最適化 … レベル 0
- 演算順序はそのままでプログラム全体での大域的な最適化 … 〃 3
- 制御構造、演算順序の変更を含むプログラム全体の最適化 … 〃 4
- 実行速度が最も速くなるコンパイルオプションの自動設定 … 〃 S
- さらなる実行速度の向上（演算精度は落ちる） … 〃 SS

使用例

```
% f77 -W0,'opt(o(4))'  a.f        （または  % f77  -opt=4   a.f ）
```

… オプション無指定時の最適化レベルは sr8000-s,p、scalar-mpp が S（f77）、3（f90）、 m-unix が 4

擬似ベクトル化 (scalar-mpp, sr8000-s,p)

擬似ベクトル化とはメモリー上のデーターを先読みすることで演算をパイプライン的に並列実行するオブジェクトを生成する機能です。データー転送のオーバーヘッド（主記憶アクセスレイテンシー）を隠蔽できるのでプログラムを高速に実行することができます。

擬似ベクトル化が適用される主な条件

- 最内側 DO ループである
- DO ループに以下の文を含まない
  - ループ脱出文 (GOTO 文、EXIT 文など)
  - 割り当て GOTO 文、SWITCH 文
  - 関数、手続き呼び出し（一部数学関数とインライン展開、別ループ化した場合を除く）
  - 入出力文
  - 終了・停止文 (STOP 文、PAUSE 文、RETURN 文など)
- コンパイラーに不明なデーターの依存関係がない
- 実行性能向上が見込める 等

使用例

```
% f77  -W0,'pvec'  a.f          （または  % f77 -pvec   a.f  ）
```

… オプション pvec はデフォルトで指定されているため、無指定時でも擬似ベクトル化を試みる (f77)
f90 では指定が必要

要素並列化 (sr8000-s,p)

要素並列化とはプログラムを並列処理単位に分割して、ノード内の複数の論理プロセッサーで並列実行するためのオブジェクトを生成する機能です。ノード間通信を行う並列化とは異なり、利用者が並列処理自体をコーディングすることなく、プログラムを高速化することができます。

要素並列化機能とレベル（上位のレベルは下位の機能を含む）

- SECTION 型要素並列化、ユーザー指示による DO 型要素並列化 … レベル 1
- 変数・配列のプライベート化、リダクション変数要素並列化 … 〃 2
- ループ分配、ループ分割、ループの一重化 … 〃 3
- パイプライン要素並列化、圧縮・拡張ループの要素並列化 … 〃 4

要素並列化が適用される主な条件（DO 型自動要素並列化）

- DO ループに以下の文を含まない
  - ループ脱出文 (GOTO 文、EXIT 文など)
  - 割り当て GOTO 文、SWITCH 文
  - 関数、手続き呼び出し（インライン展開、別ループ化した場合を除く）
  - 入出力文
  - 終了・停止文 (STOP 文、PAUSE 文、RETURN 文など)
- コンパイラーに不明なデーターの依存関係がない
- 実行性能向上が見込める 等

使用例

```
% f77 -parallel   a.f      （または % f77  -W0,'mp'  a.f）
```

… 要素並列化レベルは -parallel のとき 2、-parallel=n とすることで レベル n(0〜4) を指定する

オブジェクトのリンク時にも -parallel オプションが必要です。

```
% f77 -parallel   a.o   b.o
```

なお、要素並列化されたプログラムを実行できるのは SR8000 の要素並列実行可能なノード（占有ノード）のみです。(sr8000-p またはバッチキュー A-ES 〜 F-ES、P001 〜 P016）

64 ビットアドレッシングモード (sr8000-s,p)

sr8000-s, sr8000-p では 64 ビットアドレッシングモードのオブジェクトモジュールを作成することができます。 メモリー使用量が 2GB を超えるときは -64 オプションを指定します。

使用例

```
% f77 -64  m.f      … 2GB を超える配列を使用したプログラム m.f
```

オブジェクトのリンク時にも -64 オプションが必要です。このときリンクするオブジェクトは全て-64 オプションを指定してコンパイルしたものでなければなりません。

```
% f77 -64   a.o   b.o
```

なお、実行時には必要なメモリー使用量 (例 2200MB) を確保する指定をして下さい。

- TSS 環境
  ```
  % limit addressspace 2200
  ```
  （必要に応じて 64datasize、64stacksize も設定）
- NQS スクリプトの記述
  ```
  #@$-lM 2200mb
  ```

-64 オプションの指定等について詳細は参考マニュアルの「64 ビットアドレッシングモードのオブジェクトモジュールの作成」を参照して下さい。

参考マニュアル　「最適化 FORTRAN77 使用の手引」
「最適化 FORTRAN90 使用の手引」

オプションに関する注意事項

オプションの指定方法

コンパイラーへのオプションを示す "-W0" の W は大文字、0 は数字（ゼロ）で指定して下さい。opt，pvec，mp や -opt，-pvec，-parallel は、大文字でも小文字でもかまいません。なお、-W0 を使用しないコンパイルオプション（-opt，-pvec，-parallel 等）が指定できるのは sr8000-s, sr8000-p のみです。

最適化、擬似ベクトル化、要素並列化の適用

コンパイラーが判断できない場合には利用者の明示的な指示が必要です。

- オプション (opt，pvec，mp) のサブオプション指定
- パラメーター (*SOPTION,*VOPTION,*POPTION) 指定

適用条件、サブオプション、パラメーターの指定等について詳細は参考マニュアルの「最適化機能」、「擬似ベクトル化機能」、「要素並列化機能」を参照して下さい。

参考マニュアル　「最適化 FORTRAN77 使用の手引」
「最適化 FORTRAN90 使用の手引」
参考資料　スーパーコンピューティングニュース（Vol.1 No.3 1999.9）
「SR8000 の有効利用法」

変数名、外部手続き名の文字数

f77 では変数名、外部手続き名を 8 文字以内に制限しています。プログラム中で 8 文字を超える名称を使用している場合は以下のコンパイルオプションを指定して下さい。

```
-i,U  -W0,'vos3(nosymnchk)'  （または -i,U -W0,'testmode(n(0))'）
```

2.3　診断メッセージ

プログラムをコンパイルするとき、診断メッセージ出力用のオプションを指定することで、プログラムに擬似ベクトル化、要素並列化、ループ最適化の処理が適用されたか否かを調べることができます。

いまソースプログラムの中身が、

```
      parameter (n=1000)
      real a(n), b(n), c(n)
      do 10 i=1,n
        a(i) = i
        b(i) = i*2.0
10    continue
      do 20 i=1,n
        c(i) = a(i) + b(i)
20    continue
      write(6,*) c(1),c(n)
      stop
      end
```

であるファイル v.f があるとします。以下のようにサブオプション diag(1)、loopdiag(1) を指定することで診断メッセージを出力することが出来ます。

擬似ベクトル化と診断メッセージ（scalar-mpp の例）

```
% f77 -W0,'pvec(diag(1)), opt(loopdiag(1))' v.f
f77: compile start : v.f

*OFORT77 V02-06-/D entered.
*program name = MAIN
   KCHF1809C
             the do 10 loop is unrolled 4 times. line=3
   KCHF1734C
             the do 10 loop is not pseudo-vectorized. it contains no array
             data for pseudo-vectorization. line=3
   KCHF1809C
             the do 20 loop is unrolled 4 times. line=7
   KCHF1700C
             the do 20 loop is pseudo-vectorized. line=7
*program units = 0001, no diagnostics generated.
```

擬似ベクトル化、要素並列化と診断メッセージ（sr8000-s,p の例）

```
% f77 -W0,'pvec(diag(1)),mp(diag(1))'  v.f
f77: compile start : v.f

*OFORT77 V01-00 entered.
*program name = MAIN
(diagnosis for loop structure)
   KCHF2000C
             the do 10 loop is parallelized.line=3
   KCHF2011C
             the variable(s) or array(s) in do 10 loop is applied to tlocal
             transformation.name=I line=3
   KCHF2000C
             the do 20 loop is parallelized.line=7
   KCHF2011C
             the variable(s) or array(s) in do 20 loop is applied to tlocal
             transformation.name=I line=7
   KCHF2009C
             the do 20 loop is synchronized by barrier at the entry of the
             loop.line=7
   KCHF1700C
             the do 20 loop is pseudo-vectorized. line=7
*program units = 0001, no diagnostics generated.
```

【参考】
sr8000-s、sr8000-p ではコンパイル時のログメッセージを作成することが出来ます。f77 のコンパイルオプションとして -optlog を指定してコンパイルした後、loggen コマンドでログメッセージファイル（ここでは v.log）を作成します。ログメッセージはソースコードに診断メッセージを付加したものでプログラムのチューニングに便利です。

使用例

```
% f77 -optlog -parallel  v.f              … コンパイル
% loggen v.f                              … ログメッセージの作成
```

```
% cat v.log                                        … ログメッセージ
            parameter (n=100000)
            real a(n), b(n), c(n)

** ループ入口で並列処理 開始
** 並列手続き名 : _parallel_func_1_MAIN
** 並列ループ
**
**     [DO 10]
**     最内側ループ展開 (2 倍) を行った。
XX     対象となる配列参照が無いので擬似ベクトル化を適用しなかった。
**
            do 10 i=1,n-10
              a(i) = i
              b(i) = i*2.0
 10         continue

** 並列処理を継続
** 並列ループ
** --- ループ入口でバリア出力 ---
** ループ出口で並列処理 終了
**
**     [DO 20]
**     最内側ループ展開 (8 倍) を行った。
**     擬似ベクトル化を適用した。
**
            do 20 i=1,n
              c(i) = a(i) + b(i)
 20         continue
            write(6,*) c(1),c(n)
            stop
            end
```

参考資料　スーパーコンピューティングニュース（Vol.1 No.3 1999.9）
「SR8000 の有効利用法」

2.4　実行

コンパイラーで作成したロードモジュールは通常 a.out （またはロードモジュール名）で実行します。ただし、使用するアプリケーションによって実行方法が異なりますので、それぞれの使用例を参考にして下さい。

limit （TSS 環境の資源制限）

ログインした環境（TSS 環境）の計算機資源はデフォルト値によって制限されています。このため、プログラムのサイズによっては資源不足となる場合がありますので必要に応じて変更して下さい。（変更はセッション内のみ有効です。）ただし、コマンド入力などもこれらのサイズの影響を受けますので無闇に大きく（または小さく）設定しないよう心掛けて下さい。

（limit コマンドの例）

```
% limit
cputime         1:00:00                 … 実行制限時間（CTIME）
datasize        131072 kbytes           … データーサイズ
stacksize       1024 kbytes             … スタックサイズ
64datasize      131072 kbytes           … データーサイズ（64bit）
64stacksize     1024 kbytes             … スタックサイズ（64bit）
addressspace    512 Mbytes              … 仮想メモリーサイズ
```

（設定方法）

```
% limit cputime  7200                   … 2 時間を設定
% limit datasize 2048                   … 2048MB を設定
% limit addressspace unlimited          … 設定できる最大値を設定
```

なお、NQS 環境での資源制限については「6.1 キューとジョブクラス制限値」を参照して下さい。

3　並列化プログラム

3.1　並列化関連コマンドと通信ライブラリー

分散メモリー型並列計算機である SR8000 と SR2201 では各ノードに分散されているメモリー空間を相互にアクセスするためにノード間通信を行う必要があります。このためメッセージ通信ライブラリーとしてリモート DMA 転送、MPI、PVM、PARALLELWARE が提供されており、これらを利用したプログラムの並列化、並列実行を支援する各種コマンドが用意されています。

prun （並列実行コマンド）

プログラムを複数のノードで並列に実行するコマンドです。定義ファイルを用いるとデーターのファイル名にノード番号を付加することができるので複数の異なる入出力データーに対してプログラムを並列に実行することができます。

使用例

（定義ファイル sample.def による例）

```
% cat a.f                                              … サンプルプログラム
        read(5,*) a,b
        write(6,*) a+b,a-b
        stop
        end
% cat sample.def                                       … 定義ファイルの例
*4  a.out < data.%n > result_a.%n                          a.out を 4 ノード、b.out を
*2  b.out < data.%n > result_b.%n                          2 ノードで実行する。
                                                           *4 は 4 ノードの意味、
                                                           %n はノード番号（1〜4）

% ls
a.f          b.f          data.1       data.3       sample.def
a.out        b.out        data.2       data.4
                                                      … 入力ファイル data.1〜4 を用意
% prun -f sample.def                                  … 並列実行
% ls
a.f          b.out        data.3       result_a.2   result_b.1
a.out        data.1       data.4       result_a.3   result_b.2
b.f          data.2       result_a.1   result_a.4   sample.def
                                                      … result_a.1〜2、result_b.1〜4が

% cat  data.2                                             作成される
2 3
% cat result_a.2
   5.00000000      -1.00000000


                                                           a.out を 4 ノードで実行
```

定義ファイルの記述

(リダイレクト出力指定)

>	上書き	標準出力
>&	〃	標準エラー出力
>>	追記	標準出力
>>&	〃	標準エラー出力

(ファイル名修飾)

%n	1〜並列プロセス数の数値
%r	各プロセスが動作している相対ノード番号
%a	各プロセスが動作している絶対ノード座標
%t	prun が起動した時刻
%d	prun が起動した日付
%p	起動した並列プロセスのプロセス ID

（実行ファイル a.out による例）

```
% prun -n 4 a.out                                     … プログラム（リモート DMA 転送等）
```

コマンド、オプション、定義ファイルの詳細はオンラインコマンド “man prun” を参照して下さい。

リモート DMA 転送

リモート DMA(Direct Memory Access) 転送はリモート DMA 転送ライブラリーを用いてノード間のメッセージ通信を行なうインターフェースです。通常のノード間通信と異なり OS を介さず、直接ユーザー空間のデーターを転送するため高速な通信ができます。通信性能を最大限に引き出す場合にはこの通信を使用します。

使用例

```
% cat rdma_sample.f                                          … サンプルプログラム
      program sample

      integer rank, size
      integer node(4)

      call $rgetnod(node)

      rank=node(1)
      size=node(2)

      write(*,*)  'node=',rank,'size=',size
      stop
      end

% f77  -rdma   rdma_sample.f                                 … f77 コマンドでコンパイル
f77: compile start : rdma_sample.f
          … -rdma オプションは静的データー自動リモート DMA 機能を使用する場合の例
              (プログラム中で *COPTION パラメーターを定義しているときは不要)
*OFORT77 V02-06-/D entered.
*program name = SAMPLE
*program units = 0001, no diagnostics generated.
% ls
a.out            rdma_sample.f    rdma_sendrecv.f    … a.out が作成される

% prun -n 4 a.out                                           … prun コマンドで実行
node=           2 size=           4
node=           1 size=           4                        4 ノードで実行の例、
node=           3 size=           4                        sr8000-p は 2 ノードまで可
node=           0 size=           4
```

参考マニュアル 「リモート DMA 転送使用の手引 -FORTRAN-」
「リモート DMA 転送使用の手引 -FORTRAN77-」

MPI

MPI(Message-Passing Interface) は MPI forum によって標準化されたメッセージ通信ライブラリーのインターフェース規約です。ノード間およびノード内のプロセス間通信に MPI 通信ライブラリーを用いたメッセージ通信ができます。多くの計算機に実装されており移植性の高いインターフェースです。

使用例　… MPI 機能で提供されているコマンドを使用する場合、環境変数 path の設定が必要です。
[環境設定]　% set path=($path /usr/mpi/bin)

```
% cat mpi_sample.f                                          … サンプルプログラム
      program sample
      include 'mpif.h'

      integer size, rank, ierr

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

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

      call MPI_FINALIZE(ierr)
```

```
      stop
      end

%  mpif77 mpi_sample.f                                  … mpif77 コマンドでコンパイル
f77: compile start : mpi_sample.f
          … f77 コマンドでコンパイルする場合には以下の指定が必要
               (scalar-mpp)
               %  f77 -i,U -W0,'vos3(nosymnchk)' -I/usr/mpi/include
                      -L/usr/mpi/lib/hmpp2/cml -lfmpi -lmpi  mpi_sample.f
               (sr8000-s,p)
               %  f77 -i,U -W0,'vos3(nosymnchk)' -I/usr/mpi/include
                      -L/usr/mpi/lib -lfmpi -lmpi  mpi_sample.f

% ls
a.out           mpi_sample.f    mpi_sample.o    mpi_sendrecv.f
                                                   … a.out が作成される
*OFORT77 V02-06-/D entered.
*program name = SAMPLE
*program units = 0001, no diagnostics generated.

% mpirun -n 4 a.out                                  … mpirun コマンドで実行
node=           3 size=           4
node=           1 size=           4                   4 ノードで実行の例、
node=           2 size=           4                   sr8000-p は 2 ノードまで可
node=           0 size=           4
```

参考マニュアル 「MPI・PVM 使用の手引」

PVM

PVM(Parallel Virtual Machine) は複数の計算機による仮想的な並列計算機のメッセージ通信ライブラリーとして発展してきたインターフェースです。ここではノード間通信として PVM 通信ライブラリーを用いたメッセージ通信ができます。

使用例　… PVM 機能を使用する場合、環境変数 PATH，PVM_ROOT，PVM_ARCH の設定が必要です。
[環境設定]　% source /usr/pvm3/pvm_config

```
% cat host.f                                          … サンプルプログラム
      program host                                      （ホストプログラム）
      include 'fpvm3.h'                                 ノードプログラム node を
      integer mytid, tids(4), numt                      4 ノードで実行する例
      call pvmfmytid(mytid)                             sr8000-p は 2 ノードまで可
      call pvmfspawn('node', PVMDEFAULT, '*', 4, tids, numt)
      write(*,*) 'parent =', mytid
      write(*,*) numt, 'tasks:', tids
      call pvmfexit(into)
      stop
      end
% cat node.f                                           … サンプルプログラム
      program node                                      （ノードプログラム）
      include 'fpvm3.h'
      call pvmfmytid(mytid)
      write(*,*) 'task id=',mytid
      stop
      end

% f77 -i,U -W0,'vos3(nosymnchk)' host.f -I/usr/pvm3/include
          -L/usr/pvm3/lib/$PVM_ARCH -lfpvm3 -lpvm3 -o host
                                                     … f77 コマンドでコンパイル
% f77 -i,U -W0,'vos3(nosymnchk)' node.f -I/usr/pvm3/include
          -L/usr/pvm3/lib/$PVM_ARCH -lfpvm3 -lpvm3 -o node
                                                     … f77 コマンドでコンパイル

% cp host node $HOME/pvm3/bin/$PVM_ARCH              … PVM の実行ファイルを置く
```

```
% ls $HOME/pvm3/bin/$PVM_ARCH
host  node

% /usr/pvm3/lib/pvm -batch start                    … PVM デーモンの起動
pvmd still running.

%  ~/pvm3/bin/HMPP/host                             … プログラムの実行
parent =      274434
  4 tasks:      393217      425985      458753      491521

% /usr/pvm3/lib/pvm -batch end                      … PVM デーモンの終了
pvmd already running.

% ls -l /tmp/pvml.*                                 … 結果は /tmp に出力される
-rw-------   1 b30000   system       180 Aug 17 18:51 /tmp/pvml.10391
-rw-------   1 b30000   system        22 Aug 17 18:51 /tmp/pvml.10391.t60001
               :                        :
-rw-------   1 b30000   system        22 Aug 17 18:51 /tmp/pvml.10391.t78001
% cat /tmp/pvml.10391.t60001                        … 出力先は環境変数 PVM_LOGDIR
 task id=      393217                                 で変更できる
```

参考マニュアル 「MPI・PVM 使用の手引」

PARALLELWARE

PARALLELWARE(Express) は ParaSoft 社によって開発された並列処理プログラム開発環境ソフトウェアであり、メッセージ通信ライブラリーとデバッガー等の開発支援ツールから構成されています。

使用例　… PARALLELWARE 機能を使用する場合、環境変数 EXPRESS の設定および環境ファイルが必要です。

[環境設定]
```
% setenv EXPRESS $HOME/.express.cst
% cp /usr/local/skel/.express.cst $HOME/.express.cst
```

（ホスト-ノードモデル）

```
% cat host.f                                        … サンプルプログラム
      program host                                     （ホストプログラム）

      call kxinit()                                    ノードプログラム node を
      pgid=kxrun(4,'node')                              4 ノードで実行する例
      call kxclos(pgid)                                 sr8000-p は 2 ノードまで可
      stop
      end
% cat node.f                                        … サンプルプログラム
      program node                                     （ノードプログラム）
      integer rank, size
      integer env(4)

      call kxinit()
      call kxpara(env)

      rank=env(1)
      size=env(2)

      write(6,*) 'node=',rank, 'size=',size
      stop
      end
% ls
cubix.f  host.f   node.f
% srf77 -kXH host.f -o host                        … srf77 コマンドでコンパイル
f77: compile start : host.f                          （ホストプログラム）

*OFORT77 V02-06-/D entered.
*program name = HOST
*program units = 0001, no diagnostics generated.
% srf77 -kXN node.f -o node                        … srf77 コマンドでコンパイル
f77: compile start : node.f                          （ノードプログラム）
```

```
*OFORT77 V02-06-/D entered.
*program name = NODE
*program units = 0001, no diagnostics generated.
% ls
cubix.f  host     host.f   host.o   node     node.f   node.o
                                                    … 実行ファイルhost, nodeを作成
% host                                               … プログラムの実行
Using partition: "ALL"
Allocated 4 nodes, origin at 0, process id 897194.
Loading "node" into all processors ...
Loaded, starting ....
node=           0 size=           4
node=           2 size=           4
node=           3 size=           4
node=           1 size=           4
```

（cubix モデル）

```
% cat cubix.f                                      … サンプルプログラム
      program cubix

      integer rank, size
      integer env(4)

      call kxinit()
      call kxpara(env)

      rank=env(1)
      size=env(2)

      call kmulti(6)
      write(6,*) 'node=',rank, 'size=',size
      stop
      end
% ls
cubix.f  host.f   node.f
% srf77 -kcubix cubix.f -o node_cubix             … srf77 コマンドでコンパイル
f77: compile start : cubix_cbx.f

*OFORT77 V02-06-/D entered.
   KCHF233C  00                  TIORET
             the variable is defined but never referred.
*program name = CBXMAIN
*program units = 0001, 0001  diagnostics generated, highest severity code is  00
% ls
node_cubix   cubix.f      cubix_cbx.o  host.f       node.f
                                                    … 実行ファイルが作成される
% cubix -n 4 node_cubix                            … cubix コマンドで実行
Cubix Version 3.2.5 -- Copyright (C) 1988-1996 ParaSoft Corp.
All Rights Reserved Copyright (C) 1996,1998, Hitachi Ltd.
HI-UX/MPP PARALLELWARE 02-04-/A
Using partition: "ALL"
Allocated 4 nodes, origin at 0, process id 897257.
Loading "node_cubix" into all processors ...
Loaded, starting ....
Execution Terminated:                               4 ノードで実行の例、
  node=  0  size=  4                                sr8000-p は 2 ノードまで可
  node=  1  size=  4
  node=  2  size=  4
  node=  3  size=  4
System 0:1   User 0:0
CUBIX: exit status 0
```

参考マニュアル 「PARALLELWARE ユーザーズガイド -FORTRAN-」
「PARALLELWARE リファレンス -FORTRAN-」

Parallel FORTRAN

Parallel FORTRAN は並列処理機能を拡張した Fortran 言語 である HPF(High Performance Fortran) の指示文を含むプログラムを Fortran90 ソースプログラムに変換するトランスレーターです。データー分散等の並列化指示文をプログラム中にコメントで記述する形式です。なお、使用する通信ライブラリーはオプションで選択します。

使用例

```
% cat hpf_sample.f                                    … サンプルプログラム
       integer a(100)
!hpf$ processors p(4)                                  4 ノードで実行の例
!hpf$ distribute a(block) onto p                       sr8000-p では 2 ノードまで可
       do 10 i=1,100
       a(i)=i
 10    continue
       write(*,*)  (a(i),i=1,100)
       end
```

（通信ライブラリーにリモート DMA 転送を使用する場合）

```
%  pf90 -Wt,'comlib(rdma),pfmsg(1)' hpf_sample.f … pf90 コマンドでトランスレート
pf90: translate start : hpf_sample.f               オプション comlib(rdma) は
*Parallel FORTRAN V02-05 entered.                  通信ライブラリーの指定
*program name = MAIN

*Parallel FORTRAN V02-05  diagnosis loop distribution
   KCPF3301T 00  Loop at line 4 is distributed.  … オプション pfmsg(1) により
*program name=MAIN                                 診断メッセージ出力
*program units = 0001, no diagnostics generated.
% ls
pf_node       hpf_sample.f                        … pf_node ディレクトリーの下に
% ls pf_node                                       Fortran90 ソースプログラム
hpf_sample.f                                       pf_node/hpf_sample.f が作成
% f90 pf_node/hpf_sample.f -lhpf                   … f90 コマンドでコンパイル
f90: compile start : pf_node/hpf_sample.f          オプション -lhpf が必要

*OFORT90 V02-06-/D entered.
*program name = MAIN
*program units = 0001, 0005  diagnostics generated, highest severity code is  00
% ls
a.out         pf_node       hpf_sample.f  hpf_sample.o
% prun -n 4 a.out                                  … prun コマンドで実行
         1           2           3           4           5           6
                :                       :
```

（通信ライブラリーに MPI を使用する場合）

```
%  pf90 -Wt,'comlib(mpi)' hpf_sample.f             … pf90 コマンドでトランスレート
pf90: translate start : hpf_sample.f               オプション comlib(mpi) は
*Parallel FORTRAN V02-05 entered.                  通信ライブラリーの指定
*program name = MAIN
*program units = 0001, no diagnostics generated.
% ls
pf_node       hpf_sample.f
% mpif90 pf_node/hpf_sample.f -lhpf                … mpif90 コマンドで実行
f90: compile start : pf_node/hpf_sample.f          オプション -lhpf が必要

*OFORT90 V02-06-/D entered.
*program name = MAIN
*program units = 0001, 0005  diagnostics generated, highest severity code is  00
% mpirun -n 4 a.out                                … mpirun コマンドで実行
         1           2           3           4           5           6
                :                       :
```

参考マニュアル　「Parallel FORTRAN 言語」
「Parallel FORTRAN 使用の手引」

4　数値計算ライブラリー

UNIX システムには数値計算ライブラリーとして以下のものがあります。

MATRIX/M、MATRIX/MPP
MATRIX/M/SSS、MATRIX/MPP/SSS（スカイライン法）
MSL2
BLAS（scalar-mpp のみ）
ScaLAPACK/PBLAS（scalar-mpp のみ）

これらのライブラリーを利用するには f77、f90 コマンドのオプションとして、
　　-l ライブラリー名　　-L ディレクトリー名
を指定します。ただし、センター提供の数値計算ライブラリーのディレクトリーは標準で設定されていますので、-L オプションは必要ありません。

MATRIX

基本配列演算、連立１次方程式、逆行列、固有値・固有ベクトル、高速フーリエ変換、擬似乱数等に関する副プログラムライブラリーです。並列処理用インターフェース（scalar-mpp,sr8000-s,p）を用いることにより、データーを各ノードに分散して配置、並列に実行することができます。MATRIX を使用するためにはコンパイル時にオプションとして以下のライブラリー名を指定します。

システム名	ライブラリー指定	ライブラリー名
m-unix	-lmat	MATRIX/M
scalar-mpp	-lmatmpp	MATRIX/MPP
sr8000-s,p	-lmatmpp	〃　（要素並列処理用）
	-lmatmpp_sc	〃　（スカラー処理用）
（スカイライン法）		
m-unix	-lmats	MATRIX/M/SSS
scalar-mpp	-lmatmpps	MATRIX/MPP/SSS
sr8000-s,p	-lmatmpps	〃　（要素並列処理用）
	-lmatmpps_sc	〃　（スカラー処理用）

使用例

（1 ノード実行の場合）

```
% cat hsru1m.f                                  … サンプルプログラム
      parameter ( n=10 )
      implicit real*4(a-h,o-z)
      dimension x(n)
      ix=0
      call hsru1m(n,ix,x,ier)
      do 10 i=1,n
        write(6,*) i,x(i)
   10 continue
      end

% f77 hsru1m.f -lmat                           … コンパイル（m-unix）
f77: compile start : hsru1m.f                   (scalar-mpp)
                                                  -lmatmpp
*FORT77/HAP V03-5A 開始                         (sr8000-s,p)
*プログラム名 = MAIN                             -lmatmpp -parallel
*プログラム数 = 0001                               または -lmatmpp_sc

% a.out                                        … 実行
          1  0.148270369
          2  0.158839464
                  :
         10  0.629399776
```

（複数ノード実行の場合）

```
% cat    hdru3mpp.f                                  … サンプルプログラム
      parameter ( n=10,npu=4 )                          （4 ノードの例）
      implicit real*8(a-h,o-z)
      dimension x(n),lstpu(0:npu-1),iopt2(2),ienv(4)
      do 10 k=0,npu-1
         lstpu(k)=k
   10 continue
      iwksize=max(128,(npu+1)*8)
      call hmatinit(iwksize,lstpu,npu,ier)    … 初期処理ルーチン
      call hkxpara(ienv)                      … ノード論理番号取得
      me=ienv(1)
      ix=0
      iopt1=1
      iopt2(1)=1
      iopt2(2)=1
      call hdru3mpp(n,ix,lstpu,npu,iopt1,iopt2,x,ier)
      write(6,*) me, n, ier
      do 20 i=1,n
        write(6,*) i,x(i)
   20 continue
      end

% f77 hdru3mpp.f -lmatmpp                     … コンパイル（scalar-mpp）
f77: compile start : hdru1mpp.f
                                                  (sr8000-s,p)
*OFORT77 V02-06-/D entered.                         -lmatmpp -parallel
*program name = MAIN
*program units = 0001, no diagnostics generated.

% prun -n 4 a.out                            … prun コマンドで実行
          1          10           0
          3          10           0            4 ノードで実行の例、sr8000-p は
          2          10           0            2 ノードまで実行可
          1  0.49669231942671920 9E-001
                    :
```

参考マニュアル　「行列計算副プログラムライブラリ MATRIX/M 」
「行列計算副プログラムライブラリ -スカイライン法- MATRIX/M/SSS 」
「行列計算副プログラムライブラリ MATRIX/MPP 」
「行列計算副プログラムライブラリ -スカイライン法-MATRIX/MPP/SSS 」

MSL2

行列計算（連立１次方程式、逆行列、固有値・固有ベクトル等)、関数計算（非線形方程式、常微分方程式、数値積分等)、統計計算（分布関数、回帰分析、多変量解析等）に関する副プログラムライブラリーです。MSL2 を使用するためにはコンパイル時にオプションとして以下のライブラリー名を指定します。

システム名	ライブラリー指定
m-unix	-lmsl2
scalar-mpp	-lMSL2
sr8000-s,p	-lMSL2

使用例

```
% cat msgu1m.f                                  … サンプルプログラム
      parameter ( n=10 )
      implicit real*4(a-h,o-z)
      dimension x(n)
      ix=0
      call msgu1m(n,ix,x,ier)                   … (sr8000-s,p) msgu1m、￥sgu1m
      do 10 i=1,n                                  (m-unix)      ￥sgu1m
        write(6,*) i,x(i)                          (scalar-mpp) msgu1m
```

```
   10 continue
      end
% f77 msgu1m.f -lMSL2                              … コンパイル
f77: compile start : msgu1m.f                        m-unix の場合は -lmsl2
*OFORT77 V02-06-/D entered.
*program name = MAIN
*program units = 0001, no diagnostics generated.
% a.out                                            … 実行
          1  0.148270369
          2  0.158839539
             :
         10  0.629399896
```

参考マニュアル　「数値計算副プログラムライブラリ MSL2 操作 」
「数値計算副プログラムライブラリ MSL2 行列計算 」
「数値計算副プログラムライブラリ MSL2 関数計算 」
「数値計算副プログラムライブラリ MSL2 統計計算 」

BLAS(scalar-mpp)

BLAS（Basic Linear Algebra Subprogram）はベクトル、行列に関する基本演算ライブラリーです。このパッケージは scalar-mpp のみサポートしています。

使用例

（BLAS ライブラリーの例）

```
% cat blas.f                                       … サンプルプログラム
      parameter ( n=4 )
      implicit real*8(a-h,o-z)
      dimension x(n)
      do 10 i=1,n
        x(i)=1d0*i
   10 continue
      alpha=2d0
      incx=1
      call dscal(n,alpha,x,incx)
      do 20 i=1,n
        write(*,*) x(i)
   20 continue
      stop
      end
% f77 blas.f -lblas                                … コンパイル
f77: compile start : blas.f
*OFORT77 V02-06-/D entered.
*program name = MAIN
*program units = 0001, no diagnostics generated.
% a.out                                            … 実行
  2.00000000000000000
  4.00000000000000000
  6.00000000000000000
  8.00000000000000000
```

ScaLAPACK ／ PBLAS(scalar-mpp)

ScaLAPACK（Scalable Linear Algebra PACKage）は PBLAS（並列版 BLAS）を含む行列計算ライブラリーです。このパッケージは scalar-mpp のみサポートしています。なお、並列版の通信関数には PVM を使用しており、実行方法等については以下の資料をご覧下さい。（PVM 環境下で `-lscalapack -lblas` を指定してコンパイルします。）

参考資料　センターニュース (Vol.30 No.2 1998.3)
「HITACHI SR2201 用 ScaLAPACK, PBLAS ライブラリーのご紹介」
センターニュース (Vol.28 No.6 1996.11)
「HITACHI SR2201 用 高速版 BLAS ライブラリーのご紹介」

5　ファイル入出力

5.1　データー形式

UNIX システムの浮動小数点形式はシステムによって以下のように表現範囲が異なっており、データー互換のためには変換が必要です。また表現範囲が異なるため、変換の際に誤差が生じる場合があるので注意して下さい。なお、SR8000 形式と SR2201 形式の単精度、倍精度浮動小数点形式は同一であり、IEEE 形式に準拠しています。

精度	SR8000 形式 (sr8000-s,p)	SR2201 形式 (scalar-mpp)	M 形式 (m-unix)
単精度	$\pm 1.175495 \times 10^{-38}$〜 $\pm 3.402823 \times 10^{38}$	$\pm 1.175495 \times 10^{-38}$〜 $\pm 3.402823 \times 10^{38}$	$\pm 5.397606 \times 10^{-79}$〜 $\pm 7.237005 \times 10^{75}$
倍精度	$\pm 2.225074 \times 10^{-308}$〜 $\pm 1.797693 \times 10^{308}$	$\pm 2.225074 \times 10^{-308}$〜 $\pm 1.797693 \times 10^{308}$	
拡張精度	$\pm 2.225074 \times 10^{-308}$〜 $\pm 1.797693 \times 10^{308}$	$\pm 3.37 \times 10^{-4932}$〜 $\pm 1.189731 \times 10^{4932}$	

例えばワークステーションなどで作成した IEEE 形式の書式なしファイルを m-unix のプログラムで読み込む場合は実行時のオプションで

```
% a.out -F'port(ieee,dstduf)'
```

とします。また、VOS3 や m-unix で作成した M 形式の書式なしファイルを sr8000-s, sr8000-p, scalar-mpp のプログラムで読み込む場合は

```
% a.out -F'port(host)'
```

として下さい。なお、これらのデーター形式の変換には環境変数による方法（「5.3 ファイル接続」を参照）もあります。

5.2　ファイル形式

書式なし入出力文で入出力する場合のファイル形式には FORTRAN 固有ファイルと標準書式なしファイルがあり、デフォルトでは以下のようになっています。

書式付き	標準テキストファイル	
書式なし	(m-unix)	FORTRAN 固有ファイル
	(scalar-mpp)	FORTRAN 固有ファイル
	(sr8000-s,p)	標準書式なしファイル（入力はファイル形式を自動判別）

例えばワークステーションなどで作成した標準書式なしファイルを scalar-mpp のプログラムで読み込む場合は実行時のオプションで

```
% a.out -F'port(dstduf)'
```
（または `% a.out -F'port(stduf)'`）

… 業界標準書式なし `dstduf` と標準書式なし `stduf` ではファイル形式は同一ですがファイル終了記録検出後の BACKSPACE 文の動作が異なります。

とします。また、sr8000-s, sr8000-p のプログラムで書式なしファイルを FORTRAN 固有ファイル形式で出力するときは以下のようにします。

```
% a.out -F'port(fortuf)'
```

データー形式、ファイル形式、実行時オプション (`port` 等）についての詳細は参考マニュアルの「入出力文とファイル」、「プログラムの実行」、「付録」を参照して下さい。

参考マニュアル　「最適化 FORTRAN77 使用の手引」
「最適化 FORTRAN90 使用の手引」

5.3 ファイル接続

標準入出力（外部装置識別子 5, 6, 7）以外の装置番号を使用するとき、入出力文とファイルを接続するには、以下の方法があります。

OPEN 文

```
OPEN(装置番号,FILE=' ファイル名',...)
 ファイル名　ファイル名（* を指定すると端末になる）
```

READ, WRITE 文

1. 環境変数を指定する場合

   OPEN 文を書かない（または、先行する OPEN 文がない）場合、環境変数 FT*nn*F*xxx* により入出力ファイル名を設定できます。

   ここで、*nn* は装置番号、*xxx* は FORTRAN 順序番号です。（FORTRAN 順序番号は、同一の装置番号を使用して、異なるファイルを順次処理する場合などに利用するもので、プログラムの実行開始時点では 001 です。）環境変数の英字は大文字でなければいけません。設定方法は以下のとおりです。

   使用例
   （TSS の例）
   ```
   % setenv FT50F001 a.data
   % setenv FT60F001 a.result
   % a.out
   ```
   （NQS スクリプトの例）
   ```
   #!/bin/csh
   cd work
   setenv FT50F001 a.data
   setenv FT60F001 a.result
   a.out
   ```

   環境変数 FT*nn*F*xxx* による入出力時の浮動小数点形式は各システムのデーター形式に従います。なお、以下のように環境変数を設定することでデーター形式を変換して入出力することができます。

	SR8000 形式	SR2201 形式	M 形式
(m-unix)	なし	FT*nn*E*xxx*	FT*nn*F*xxx*
(scalar-mpp)	なし	FT*nn*F*xxx* または FT*nn*E*xxx*	FT*nn*M*xxx*
(sr8000-s,p)	FT*nn*F*xxx* または FT*nn*S*xxx*	FT*nn*E*xxx*	FT*nn*M*xxx*

   使用例
   （m-unix での例）
   ```
   #!/bin/csh
   cd work
   setenv FT50F001 a.data          … M 形式で入力
   setenv FT60E001 a.result        … SR2201 形式で出力
   a.out
   ```
   （sr8000-s,p での例）
   ```
   #!/bin/csh
   cd work
   setenv FT50E001 a.data          … SR2201 形式で入力
   setenv FT60F001 a.result        … SR8000 形式で出力
   a.out
   ```

2. 環境変数を指定しない場合

上記方法（環境変数に設定する）によらない場合、READ, WRITE 文により特定ファイル ft.*nn*（*nn* は装置番号）に接続します。READ 文の場合、ファイルが存在しないとエラーになります。WRITE 文の場合、ファイルが存在しないと新しく作成します。すでに存在する場合は、先頭から書き込まれます。

使用例

（端末から入力、ファイル ft.60 へ出力）

```
% cat u.f
      read(5,*) a,b
      write(60,*) a*b,a/b
      stop
      end
% f77 u.f
% a.out
?
2 3
% cat ft.60                          … ファイルがなければ作成される
  6.00000000    0.66666627
```

（ファイル ft.50 が存在しないのでエラーとなる）

```
% cat u.f
      read(50,*) a,b
      write(60,*) a*b,a/b
      stop
      end
% f77 u.f
% a.out
KCHF348R the read statement for unit id 50 is invalid. the
file does not exist.
```

（ファイル ft.50 から入力、ファイル ft.60 へ出力）

```
% cat ft.50                          … 入力ファイル
2 3
% a.out
% cat ft.60
  6.00000000    0.66666627
```

DEFINE FILE 文

DEFINE FILE 文は特定のファイル ft.*nn*（*nn* は装置番号）に接続されます。DEFINE FILE 文については参考マニュアルの「JIS FORTRAN より拡張している言語使用」を参照して下さい。

参考マニュアル　「最適化 FORTRAN77 言語」
　　「最適化 FORTRAN90 言語」

6 NQS

UNIX システムにおけるバッチジョブを計算機に投入するための機能を NQS（Network Queueing System）といいます。本センターの大型計算機システムは多くの利用者が同時に計算機を使用するため、1 ジョブに割り当てる計算機資源を制限しており、実行時間、メモリー使用量、ノード数などジョブの規模により、ジョブクラス制限値を設定しています。NQS ではこのジョブクラス制限値に従って、利用者がスムーズにジョブを投入、実行できるようなコマンド環境を提供しています。

6.1 キューとジョブクラス制限値 (1999 年 5 月 1 日現在)

NQS ではジョブクラス制限値を設定した各ジョブクラスの待ち行列をキュー（バッチキュー）と呼びます。投入したジョブは利用者の宣言値に従って適当なバッチキューに並び、計算機資源が確保できるまで待ちます。順番がきて実行に必要な資源が確保できるとジョブは実行を開始します。ここでは各キューの名称と制限値について説明します。

MP5800 ジョブクラス制限値 (m-unix)

キュー名	制限時間 CTIME	メモリーの大きさ（MB）
A	2 分	1024
B	10 〃	〃
C	60 〃	〃
D	180 〃	〃
E	600 〃	〃
F	2700 〃	〃
L	600 〃	1760
TSS	無制限	128

オプション記述例

```
#@$-lT 1:00:00 … 制限時間（CTIME）
#@$-lM 1024MB  … メモリー使用量
```

m-unix のジョブクラスは CTIME（CPU 時間）により区分しています。利用者がキュー名を指定する必要はなく、ジョブ制限時間（CTIME）、メモリー使用量を指定することによって実行可能なバッチキュー A〜L に自動的にキューイングされます。なお、キュー L を実行する場合には超大型計算利用届の提出が必要です。

SR2201 ジョブクラス制限値 (scalar-mpp)

キュー名	制限時間 ETIME	ノード数	メモリーの大きさ（MB/node）
P001	128 時間	1	224
P002	128 〃	2	〃
P004	64 〃	4	〃
P008	32 〃	8	〃
P016	16 〃	16	〃
P032	8 〃	32	〃
P064	4 〃	64	〃
P128	2 〃	128	〃
P256	2 〃	256	〃
TSS	(16 〃)	32	192
P1024	1 〃	1024	224

オプション記述例

```
#@$-N  8        … ノード数
#@$-lT 1:00:00 … 制限時間（ETIME）
#@$-lM 224MB    … メモリー使用量
```

scalar-mpp のジョブクラスはノード数と ETIME（経過時間）により区分しています。利用者がキュー名を指定する必要はなく、ノード数、ジョブ制限時間 (ETIME)、メモリー使用量を指定することによって実行可能なバッチキュー P001〜P256 に自動的にキューイングされます。（使用できるノード数の最大値は登録によって異なります。）なお、ノード数の指定を 257 以上とするとキュー P1024 にキューイングされます。このキューのジョブは原則として毎月の第 1 土日曜日（金曜 19:00〜月曜 9:00）に実行されます。

SR8000 ジョブクラス制限値 (sr8000-s,sr8000-p)

キュー名	制限時間		メモリーの大きさ（MB/node）		ノード数
	CTIME	ETIME	仮想メモリー	区分 ES	
single					
A	16 分	10 分	512(7168)	0(0)	1
B	80 〃	50 〃	〃	〃	〃
C	480 〃	300 〃	〃	〃	〃
D	1440 〃	900 〃	〃	〃	〃
E	4800 〃	3000 〃	〃	〃	〃
F	21600 〃	13500 〃	〃	〃	〃
A-ES	16 分	10 分	512(3072)	0(16384)	1
B-ES	80 〃	50 〃	〃	〃	〃
C-ES	480 〃	300 〃	〃	〃	〃
D-ES	1440 〃	900 〃	〃	〃	〃
E-ES	4800 〃	3000 〃	〃	〃	〃
F-ES	21600 〃	13500 〃	〃	〃	〃
parallel					
P001	無制限	64 時間	7168(7168)	0(0)	1
P002	〃	32 〃	〃	〃	2
P004	〃	16 〃	〃	〃	4
P008	〃	8 〃	〃	〃	8
P016	〃	4 〃	〃	〃	16
TSS					
sr8000-s	無制限	(18 時間)	128(512)	—	1
sr8000-p	〃	〃	〃(7168)	—	2

sr8000-s、sr8000-p のジョブクラスはパイプキューと呼ばれるキューを指定します。なお、ログインしているシステムが sr8000-s、sr8000-p のどちらであってもバッチシステムは共通です。

シングルノードで実行するジョブクラス — パイプキュー single

- スカラージョブ実行（ノード共有：要素並列ジョブ実行不可）
  次の指定によって実行可能なバッチキュー A〜F にキューイングされます。

```
#@$-q  single   … パイプキュー
#@$-lT 1:00:00  … 制限時間 (ETIME)
#@$-lM 7192MB   … メモリー使用量
```

- 拡張記憶 ES 使用ジョブ実行（ノード占有：要素並列ジョブ実行可）
  次の指定によって実行可能なバッチキュー A-ES〜F-ES にキューイングされます。

```
#@$-q  single   … パイプキュー
#@$-lT 1:00:00  … 制限時間 (ETIME)
#@$-lV 2GB      … ES 使用量
#@$-lM 2048MB   … メモリー使用量
```

複数ノードで実行するジョブクラス — パイプキュー parallel

- ノード間並列ジョブ実行（ノード占有：要素並列ジョブ実行可）
  次の指定によって実行可能なバッチキュー P001〜P016 にキューイングされます。

```
#@$-q  parallel … パイプキュー
#@$-N  2        … ノード数
#@$-lT 1:00:00  … 制限時間 (ETIME)
#@$-lM 7192MB   … メモリー使用量
```

ジョブクラス制限値の内容は変更となる場合があります。最新の情報はスーパーコンピューティングニュースの表紙裏または Web ページ (http://www.cc.u-tokyo.ac.jp) をご覧下さい。

6.2 NQS コマンドとスクリプトファイル

qsub - ジョブの投入

qsub コマンドによりバッチジョブをサブミットします。ジョブはスクリプトファイルの形態で記述し、qsub コマンドのオプション、あるいはシェルスクリプト内のコメント文により、ジョブ制限時間、ノード数、メモリー使用量などについて、ユーザーが利用したいと思う最大値を宣言します。ジョブは次のようにサブミットします。(スクリプトファイル job.csh)

```
%  qsub  job.csh
Request 6128.sr8000-s.cc.u-tokyo.ac.jp submitted to queue: single.
```

*ジョブはパイプキュー(ここでは single)を経由し、バッチキューにキューイングされます。(リクエスト番号 6128)　複数のジョブを投入できますが同時に投入するジョブ数は 8 ジョブまでとして下さい。

以下のオプション指定により(ユーザーが使いたいと思う)最大値を宣言します。これらの指定により、NQS は適当なバッチキューを自動選択します。(リソースの有効利用のため、無闇に大きな値を指定するのではなく、実際に必要な最小限の量を指定してください。)

-q queue	パイプキューの指定
-N node	ジョブが使用するノード数
-lT time	ジョブ全体での制限時間*
-lM mem	ジョブ全体でのメモリー使用量
-lV mem	ジョブ全体での ES 使用量*

*制限時間 -lT は m-unix では CTIME(CPU 時間)、sr8000-s,p、scalar-mpp では ETIME(経過時間) です。また、ES 使用量 -lV は sr8000-s,p のみ使用できます。

また、必要に応じて以下の指定ができます。

-AC account_number	課金番号(支払いコード)の指定
-lt time	プロセス毎の制限時間 CTIME(CPU 時間)
-ld mem	プロセス毎のデーターサイズ
-ls mem	プロセス毎のスタックサイズ
-mu mail_address	ジョブ実行に関するメール送信先アドレス
-nr	システムダウン時のジョブリスタート禁止

スクリプトファイルの記述

NQS ジョブのスクリプトファイルはシェルスクリプトで記述します。スクリプトファイルはシェルスクリプトと同様に指定されたシェル(ここでは C シェル)を起動して実行に移ります。使用するシェルの指定はスクリプトファイルの先頭行に以下のように記述して下さい。

```
#!/bin/csh
```

qsub コマンドにおいて、コマンドラインで指定できるオプションは、サブミットするスクリプト内で、

```
#@$[option]
```

で等価な指定ができます。例えば、ノード数を 2 、ジョブ全体のメモリー使用量を 100MB に、制限時間 (ETIME) を 15 分に、設定したい場合、シェルスクリプト内で

```
#@$-N  2
#@$-lM 100mb
#@$-lT 15:00
```

と記述することになります。(メモリーの単位の mb は、大文字と小文字を区別しませんので、MB，Mb，mB でも OK です。しかし、-N，-lT，-lV，-q の N，T や V は必ず大文字で、l，q は必ず小文字で書いてください。)

qsub コマンドでは、現在のディレクトリー（current working directory）などの環境は引き継がれず、再度ログインし直したような形になります。したがって、実行したいプログラムが ~/work などにある場合には、

```
cd ~/work   （または /home/利用者番号/work）
実行するコマンド名
        :
```

などのように、NQS で実行されるスクリプト内で、ディレクトリーを移動してからコマンドを起動して下さい。実行するコマンド名にはプログラム (a.out 等の実行ファイル）や UNIX コマンド (cd，setenv 等) を記述します。
実際のスクリプトファイルの例は以下のようになります。

```
#!/bin/csh
#@$-q single       … sr8000-s または sr8000-p のシングルノード
#@$-lT 1:00:00    … 実行制限時間 (ETIME) は 1 時間
#@$-lM 1024mb     … メモリー使用量は 1024MB
cd ~/work         … a.out のあるディレクトリーへ移動
a.out < a.data    … 実行 (入力データー a.data)
```

"qsub スクリプトファイル名" のような形式でジョブを投入した場合（オプションで特に指定がなければ)、実行結果は、

```
標準出力：        スクリプトファイル名.oN
標準エラー出力：  スクリプトファイル名.eN
```

に出力されます。ここで、N はリクエスト番号で、1 から 5 けたで出力されます。(スクリプトファイル名は最初の 7 文字を使用します。）スクリプトファイル名を指定しなかった場合には、

```
標準出力：        STDIN.oN
標準エラー出力：  STDIN.eN
```

に出力されます。(スクリプトファイル名を指定しない場合、qsub に引き続く一連のコマンドを一つのジョブとして処理します。しかし、初心者の方はこのジョブ入力時にトラブルを起こしやすいので、スクリプトファイル名を指定する方法による利用をお勧めします。)

qstat - キューの状態

qstat コマンドにより、現在のキューの状態を知ることができます。

```
% qstat
1999/08/11 (Wed) 17:49:37:   BATCHPIPE REQUESTS on sr8000-bt.cc.u-tokyo.ac.jp
NQS schedule stop time : 1999/08/12 (Thu) 18:00:00 (Remain:  24h 10m 23s)
   REQUEST      NAME      OWNER      QUEUE     PRI NICE   CPU    MEM    STATE
 6128.sr8000-s  STDIN     a30000     A         31  20     960    7168   RUNNING
 6127.sr8000-s  STDIN     a30000     P002      31  20     UNLIM  7168   QUEUED
```

(STATE が RUNNING 表示は実行中、QUEUED 表示は実行待ちを示します。)

```
% qstat -b
1999/08/11 (Wed) 17:28:32:   BATCH PIPE  QUEUES on m-unix.cc.u-tokyo.ac.jp
QUEUE NAME      STATUS    TOTAL   RUNNING  RUNLIMIT  QUEUED  HELD    IN-TRANSIT
A               AVAILBL   0       0        3         0       0       0
B               AVAILBL   0       0        3         0       0       0
C               AVAILBL   0       0        2         0       0       0
```

```
D               AVAILBL   0       0        2         0       0       0
E               AVAILBL   0       0        1         0       0       0
F               AVAILBL   0       0        1         0       0       0
L               STOPPED   0       0        1         0       0       0
```

（STATUS の AVAILBL 表示が実行可能、STOPPED 表示は実行不可能状態を示します。）

```
1999/08/11 (Wed) 17:29:44:   BATCH PIPE  QUEUES on sr8000-bt.cc.u-tokyo.ac.jp
NQS schedule stop time : 1999/08/12 (Thu) 18:00:00 (Remain:  24h 30m 16s)
QUEUE NAME      STATUS    TOTAL   RUNNING  RUNLIMIT  QUEUED  HELD    IN-TRANSIT
A               AVAILBL   0       0        2         0       0       0
B               AVAILBL   0       0        2         0       0       0
C               AVAILBL   0       0        2         0       0       0
D               AVAILBL   2       2        2         0       0       0
E               AVAILBL   0       0        2         0       0       0
F               AVAILBL   1       0        2         1       0       0
A-ES            AVAILBL   0       0        2         0       0       0
B-ES            AVAILBL   0       0        2         0       0       0
C-ES            AVAILBL   0       0        2         0       0       0
D-ES            AVAILBL   0       0        2         0       0       0
E-ES            AVAILBL   0       0        2         0       0       0
F-ES            AVAILBL   2       0        2         2       0       0
P001            AVAILBL   3       2        2         1       0       0
P002            AVAILBL   0       0        2         0       0       0
P004            AVAILBL   0       0        2         0       0       0
P008            AVAILBL   0       0        2         0       0       0
P016            AVAILBL   0       0        2         0       0       0
```

（RUNNING が実行中のジョブ数、QUEUED が実行待ちのジョブ数です。）

```
1999/08/11 (Wed) 17:28:25:   BATCH PIPE  QUEUES on scalar-mpp-bt.cc.u-tokyo.ac.jp
NQS schedule stop time : 1999/09/03 (Fri) 18:00:00 (Remain:  552h 31m 35s)
QUEUE NAME      STATUS    TOTAL   RUNNING  RUNLIMIT  QUEUED  HELD    IN-TRANSIT
P001            AVAILBL   4       3        16        1       0       0
P002            AVAILBL   2       2        8         0       0       0
P004            AVAILBL   3       3        8         0       0       0
P008            AVAILBL   3       2        4         1       0       0
P016            AVAILBL   5       4        4         1       0       0
P032            AVAILBL   7       3        3         4       0       0
P064            AVAILBL   13      3        3         10      0       0
```

（NQS schedule stop time の行が計画停止時刻（残り実行可能時間）を示しています。）

計画停止時刻 (scalar-mpp, sr8000-s,p)

システム停止が予定されている場合には、その予定時刻までに全てのジョブが実行を終了するよう NQS に計画停止時刻を設定しています。このときキューの状態が実行可能状態 (AVAILVL) と表示していても計画停止時刻を超過するジョブは実行されず待ち状態 (QUEUED) となります。計画停止時刻までにジョブが終了するよう実行制限時間 (-lT オプション) を設定して下さい。計画停止時刻は qstat コマンドで確認できます。

```
NQS schedule stop time : 1999/09/03 (Fri) 18:00:00 (Remain:  2h 3m 35s)
                         計画停止時刻                  残り実行可能時間
```

残り実行可能時間が 2 時間の場合、以下のように制限時間を 2 時間以内に設定します。

```
#@$-N 1
#@$-lT 1:50:00      … 2 時間以内なので実行可能
```

qdel - ジョブの削除

qdel コマンドにより、実行中ないしは、待機中のジョブをキャンセルすることができます。

キャンセルしたいジョブのリクエスト ID（リクエスト番号.ホスト名）を qstat で確認します。

```
   REQUEST      NAME      OWNER      QUEUE     PRI NICE   CPU    MEM    STATE
6128.sr8000-s   STDIN     a30000     A         31  20     960    7168   RUNNING
```

（REQUEST 部分がリクエスト ID の一部です。）

リクエスト ID を指定して実行中のジョブをキャンセルします。

```
  % qdel -k 6128.sr8000-s.cc.u-tokyo.ac.jp
  Request 6128.sr8000-s.cc.u-tokyo.ac.jp is running, and has been signalled.
```

以下のように省略することもできます。

```
  % qdel -k 6128
```

なお、実行待ちジョブの場合には -k の指定を省略できます。

参考マニュアル 「NQS ユーザーズガイド」

6.3 NQS ジョブ実行例

以下に実行例を示します。

例 1 - コンパイルと実行の例

```
 sr8000-s で FORTRAN プログラムをコンパイル、NQS で実行する
% ls
go1.csh   test.dat  test.f

% cat test.f                              … ソースプログラム test.f とデーター
c                                            test.data は予め作成しておきます
      parameter (nn=10)
      implicit real*8 (a-g,o-z)
      dimension a(nn,nn), x(nn), b(nn)
c
c     aij = a(j,i)
      read (*,*) ((a(j,i), j=1,nn), i=1,nn)
      read (*,*) (x(j), j=1,nn)
c
      do 100 i=1,nn
       b(i) = 0.0d0
  100 continue
c
      do 200 i=1,nn
       do 210 j=1,nn
        b(i) = b(i) + a(j,i) * x(j)
  210  continue
  200 continue
c
      do 300 i=1,nn
       write(*,*) b(i)
  300 continue
      stop
      end

% f77 test.f -o program           … ソースプログラム test.f をコンパイルして
f77: compile start : test.f           実行ファイル program を作成します

*OFORT77 V01-00 entered.
*program name = MAIN
*program units = 0001, no diagnostics generated.

% ls
go1.csh    program    test.data  test.f

% cat go1.csh                     … ジョブ go1.csh は予め作成しておきます
#!/bin/csh
#@$-q single                     … 1 ノードで実行するジョブ
#@$-lT 1:00:00                       制限時間 ETIME（経過時間）で 1 時間
#@$-lM 100mb                         メモリー使用量 100MB
cd ~/test
program < test.data
% qsub go1.csh                   … ジョブ go1.csh をサブミットします
Request 6180.sr8000-s.cc.u-tokyo.ac.jp submitted to queue: single.
```

```
% qstat                                  … ジョブの状態を確認します
   （省略）
     :
1999/08/12 (Thu) 14:18:50:   BATCHPIPE REQUESTS on sr8000-bt.cc.u-tokyo.ac.jp
NQS schedule stop time : 1999/08/12 (Thu) 18:00:00 (Remain:  3h 41m 10s)
   REQUEST      NAME      OWNER      QUEUE     PRI NICE   CPU    MEM    STATE
 6180.sr8000-s  go1.csh   a30000     C         31  20     28800  7168   RUNNING
 6183.sr8000-s  go2.csh   a30000     P001      31  20     UNLIM  7168   RUNNING
       :
                                        … リクエスト ID 6180.sr8000-s.. のジョブは
                                          ジョブクラス C で実行中（RUNNING）である
                                          ことがわかります
% ls
go1.csh         go1.csh.o6180  test.data
go1.csh.e6180   program        test.f
                                        … ジョブが終了すると２つのファイル
                                          go1.csh.o6180、go1.csh.e6180 が作成されます
% cat  go1.csh.e6180                    … 標準エラー出力ファイル（エラーのとき
                                          エラーメッセージが出力されます）
% cat  go1.csh.o6180                    … 標準出力ファイル（実行結果が入ります）
   1.00000000000000000
   2.00000000000000000
   3.00000000000000000
           :
   1.00000000000000000

% qdel 6183.sr8000-s.cc.u-tokyo.ac.jp    … リクエスト ID 6183.sr8000-s.. の
                                          ジョブのキャンセルを試みます
Request 6183.sr8000-s.cc.u-tokyo.ac.jp is running.
                                        … ジョブは実行中なのでキャンセル
                                          できません

% qdel -k 6183.sr8000-s.cc.u-tokyo.ac.jp … 実行中のジョブのキャンセルには
                                          -k オプションを指定します。
Request 6183.sr8000-s.cc.u-tokyo.ac.jp is running, and has been signalled.
                                        … ジョブはキャンセルされました
```

例２ - リモート DMA 転送プログラムの例

scalar-mpp でリモート DMA 転送による並列プログラムをコンパイルし、NQS で実行させる

```
% cat rdma_sendrecv.f                   … サンプルプログラム
      program sample

      integer node(4), msg(4)
      integer key, objid, stat
      integer cdesc(3), rdesc
      integer sendbuf, recvbuf(3)

      call $rgetnod(node)
      call $rstatic(key,objid)

      if ( node(1) .eq. 0 ) then
        do 10 i=1, node(2)-1
           stat=$rcreate(objid, recvbuf(i), 4, i, 0, cdesc(i))
           call $rread(cdesc(i), -1, stat)
           write(*,*) 'data=', recvbuf(i), 'from node', i
 10     continue
      else
        stat=$rright(0, key, node(1), -1, rdesc)
        sendbuf=node(1)
        call $rsetmsg(msg, 4, sendbuf)
        call $rwrite(msg, rdesc, 0, stat)
      endif
      stop
      end
```

```
% f77 rdma_sendrecv.f -rdma                          … コンパイル
f77: compile start : rdma_sendrecv.f

*OFORT77 V02-06-/D entered.
*program name = SAMPLE
*program units = 0001, no diagnostics generated.

% cat rdma.csh                                       … スクリプトファイル
#!/bin/csh
#@$-N 4
cd ~/sample_parallel/rdma
prun -n 4 a.out

% qsub rdma.csh                                      … サブミット
Request 29590.scalar-mpp.cc.u-tokyo.ac.jp submitted to queue: multi.

% qstat                                              … ジョブの状態確認
1999/08/23 (Mon) 18:49:29:   BATCHPIPE REQUESTS on scalar-mpp-bt.cc.u-tokyo.ac.jp
NQS schedule stop time : 1999/09/03 (Fri) 18:00:00 (Remain:  263h 10m 31s)
   REQUEST      NAME      OWNER      QUEUE     PRI NICE   CPU    MEM    STATE
29591.scalar-m  rdma.csh  a30000     P004      31  20     230400 224    RUNNING
          :
% ls                                                … 実行終了後
README           rdma.cs.e29590   rdma.csh
a.out            rdma.cs.o29590   rdma_sendrecv.f
% cat     rdma.cs.e29590                            … エラー出力ファイル
% cat     rdma.cs.o29590                            … 結果出力ファイル
 data=           1 from node          1               （実行結果）
 data=           2 from node          2
 data=           3 from node          3
```

例 3 - MPI プログラムの例

sr8000-s で MPI による並列プログラムをコンパイルし、NQS で実行させる

```
% cat mpi_sendrecv.f                                  … サンプルプログラム
      program sample
      include 'mpif.h'

      integer size, rank
      integer buf(1), count, tag, stat(MPI_STATUS_SIZE), ierr
      integer dest, src

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

      buf(1)=rank
      count=1
      tag=1

      if ( rank .eq. 0 ) then
        do 10 src=1, size-1
           call MPI_RECV(buf, count, MPI_INTEGER,
     +                   src, tag, MPI_COMM_WORLD, stat, ierr)
           write(*,*) 'data=', buf(1), 'from node', src
 10     continue
      else
        dest=0
           call MPI_SEND(buf, count, MPI_INTEGER,
     +                   dest, tag, MPI_COMM_WORLD, ierr)
      endif

      call MPI_FINALIZE(ierr)
      stop
      end

% set path=($path /usr/mpi/bin)                       … 環境変数の設定

% mpif77 mpi_sendrecv.f                              … コンパイル
f77: compile start : mpi_sendrecv.f
```

```
*OFORT77 V01-00 entered.
*program name = SAMPLE
*end of compilation : SAMPLE
*program units = 0001, no diagnostics generated.

% cat mpi.csh                                          … スクリプトファイル
#!/bin/csh
#@$-q parallel
#@$-N 4
cd ~/sample_parallel/mpi
/usr/mpi/bin/mpirun -n 4 a.out

% qsub mpi.csh                                         … サブミット
Request 6731.sr8000-s.cc.u-tokyo.ac.jp submitted to queue: parallel.

% qstat                                                … ジョブの状態確認
1999/08/23 (Mon) 18:54:58:   BATCHPIPE REQUESTS on sr8000-bt.cc.u-tokyo.ac.jp
NQS schedule stop time : 1999/08/31 (Tue) 07:00:00 (Remain:  180h 5m 2s)
   REQUEST      NAME      OWNER      QUEUE     PRI NICE   CPU    MEM    STATE
 6731.sr8000-s  mpi.csh   a30000     P004      0   20     UNLIM  7168   RUNNING
                :
% ls                                                  … 実行終了後
README           mpi.csh.e6731    mpi.csh
a.out            mpi.csh.o6731    mpi_sendrecv.f
% cat     mpi.csh.e29590                              … エラー出力ファイル
% cat     mpi.csh.o29590                              … 結果出力ファイル
 data=           1 from node          1                 （実行結果）
 data=           2 from node          2
 data=           3 from node          3
```

参考資料（本文中の参考マニュアルの一覧）

マニュアル名称	HI-UX/MPP (SR8000)	HI-UX/MPP (SR2201)	HI-OSF/1 (MP5800)
最適化 FORTRAN90 言語	6A30-3-310	6A20-3-310	—
最適化 FORTRAN90 使用の手引	6A30-3-311	6A20-3-311	—
最適化 FORTRAN77 言語	6A30-3-313	6A20-3-313	8X20-3-110
最適化 FORTRAN77 使用の手引	6A30-3-314	6A20-3-314	8X20-3-111
MPI・PVM 使用の手引	6A30-3-026	6A20-3-026	—
リモート DMA 転送使用の手引 -FORTRAN-	6A30-3-312	6A20-3-312	—
リモート DMA 転送使用の手引 -FORTRAN77-	6A30-3-315	6A20-3-315	—
Parallel FORTRAN 言語	6A30-3-320	6A20-3-320	—
Parallel FORTRAN 使用の手引	6A30-3-321	6A20-3-321	—
PARALLELWARE ユーザーズガイド -FORTRAN-	6A30-3-400	6A20-3-400	—
PARALLELWARE リファレンス -FORTRAN-	6A30-3-401	6A20-3-401	—
OSCNQS NQS ユーザーズガイド	6A30-3-230	8X20-3-230	8X20-3-230
行列計算副プログラムライブラリ MATRIX/MPP	6A30-7-600	6A20-7-600	—
MATRIX/M	—	—	8X20-7-504
行列計算副プログラムライブラリ MATRIX/MPP/SSS	6A30-7-601	6A20-7-601	—
スカイライン法 MATRIX/M/SSS	—	—	8X20-7-505
数値計算副プログラムライブラリ MSL2 行列計算	6A30-7-610	6A20-7-610	8X20-7-500
数値計算副プログラムライブラリ MSL2 関数計算	6A30-7-611	6A20-7-611	8X20-7-501
数値計算副プログラムライブラリ MSL2 統計計算	6A30-7-612	6A20-7-612	8X20-7-502
数値計算副プログラムライブラリ MSL2 操作	6A30-7-613	6A20-7-613	8X20-7-503