UNIX システム利用入門　～後編～

システム運用係

注）本稿は「東大センターにおける利用入門 第 2 版（2004 年 4 月）」の内容を最新の情報に書き換え，再編集したものです。なお，前編についてはスーパーコンピューティングニュース Vol.7 No.5 2005.9 を御覧下さい。

1.　システム概要

1.1.　システム構成

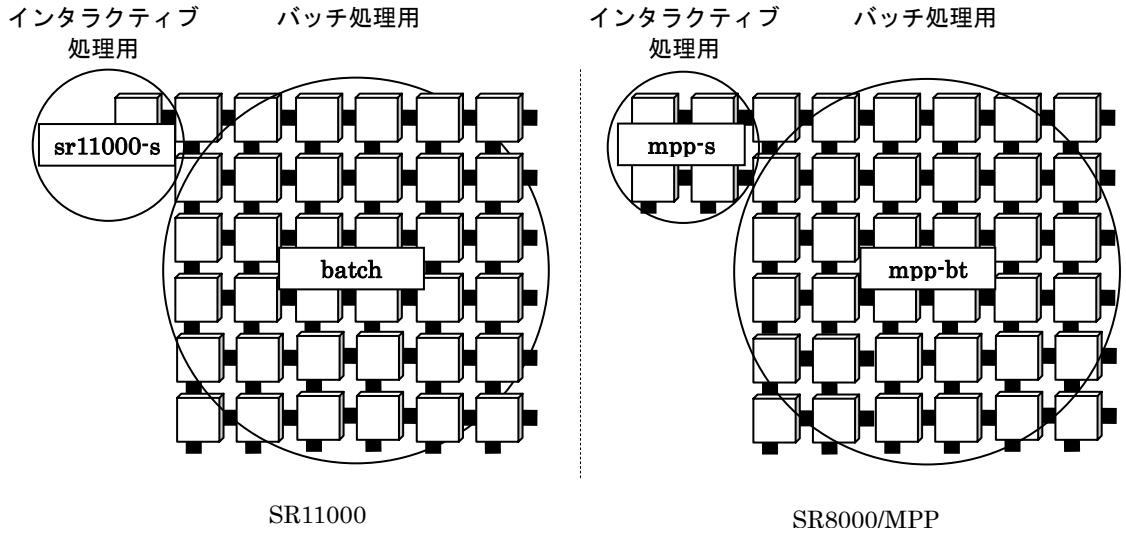
本センターでは 2 台のスーパーコンピューターを運用しています。これらのシステムは 1 ノード当り 8 台または 16 台の演算プロセッサーを備えたノードを複数台搭載しており，ノードを並列に動作させるのと同時にノード内でも並列処理が可能です。

	SR11000	SR8000/MPP
ノード数	44（16cpu/node）	144（8cpu/node）
1 プロセッサー当りの理論性能	9.2GFLOPS	1.8GFLOPS
1 ノード当りの理論演算性能	147.2GFLOPS	14.4GFLOPS
1 ノード当りの主記憶容量	128GB	16GB
演算プロセッサー数	704	1152

☞ SR11000 は 2005 年 3 月，SR8000/MPP は 2001 年 4 月よりサービス開始。なお SR8000/MPP は 2007 年 3 月上旬でサービス終了予定。

☞ SR11000 の並列実行用ノードは，物理的なノードを論理分割し 8 プロセッサーで構成する SMP を 1 ノードとして運用（1 ノード当りの理論演算性能，主記憶容量は 73.6GFLOPS，64GB）。

各ノードはクロスバーネットワークと呼ばれる内部ネットワークで結合されており，互いに通信することができます。本センターでは搭載された多くのノードと機能を活かし，様々なバッチ処理やインタラクティブ処理を行えるよう，通常はそれぞれのシステムを次頁の図のようにインタラクティブ処理用，バッチ処理用の 2 つに分割して運用しています。



SR11000　　SR8000/MPP

注）図は実際のノードの配置や数とは異なります。

☞ SR11000 は SMP クラスター型システムであり複数台のシステム（ノード）で構成されますが，本センターでは SR11000 のバッチ用システムを総称して"batch"と呼んでいます。

☞ SR8000/MPP では月に一度，128 ノードの大規模バッチジョブが実行できる環境を提供しています。また，SR11000 では月に一度，全ての利用者を対象とする 8 ノードジョブの実行サービスを行っています。詳細はウェブページ（http://www.cc.u-tokyo.ac.jp/）を御覧下さい。

- インタラクティブ処理（TSS）用

  sr11000-s.cc.u-tokyo.ac.jp　（SR11000）

  mpp-s.cc.u-tokyo.ac.jp　（SR8000/MPP）

  スカラージョブを対話的に実行できます。主にプログラムの作成・編集，コンパイル，バッチジョブの投入に使用します。

- バッチ処理用（ログイン不可）

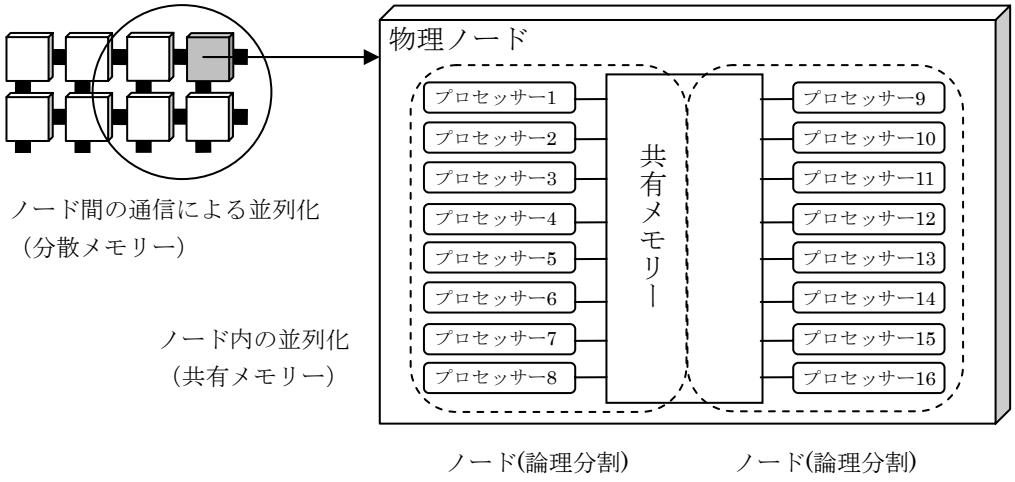
  batch.cc.u-tokyo.ac.jp　（SR11000）

  mpp-bt.cc.u-tokyo.ac.jp　（SR8000/MPP）

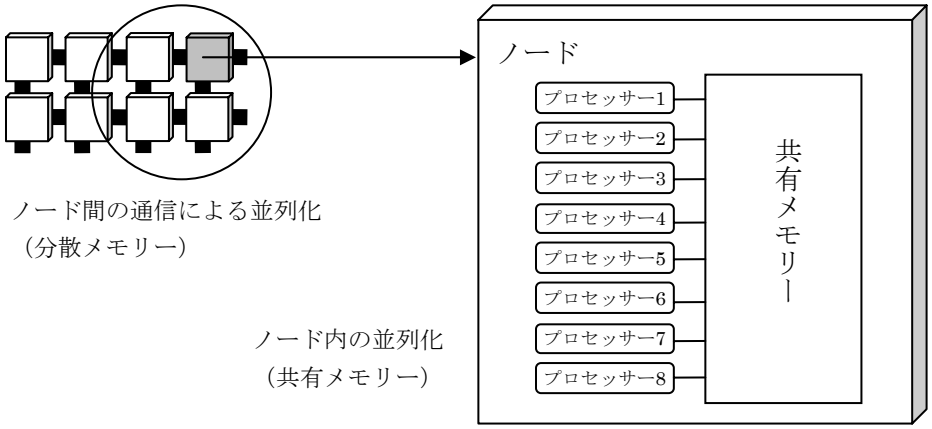
  スカラージョブ，要素並列ジョブ，ノード並列ジョブ，拡張記憶（ES）使用ジョブを実行できます。ジョブはインタラクティブ処理用サブシステムから投入します。

1.2. ノード構成

SR11000 は SMP クラスター型のシステムです。物理的には 1 ノード当たり 16 台の CPU を搭載していますが，これを 8 台の CPU で構成される 2 ノードに論理分割して運用しています。ノード内では 8 台の CPU がメモリーを共有する共有メモリー型，ノード間は分散メモリー型になります。ノード単体でも並列実行やスカラージョブの多重処理が可能です。



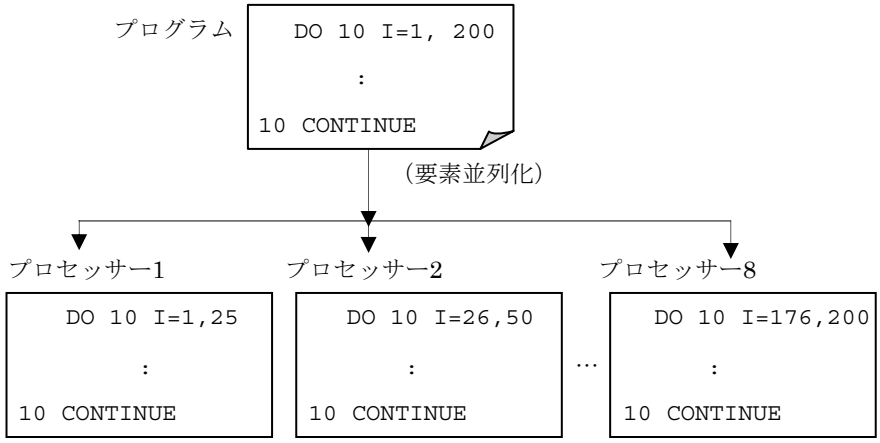
SR8000/MPP は複数のノードを搭載した分散メモリー型の並列計算機です。各ノードは 8 台の CPU と共有メモリーで構成されています。SR11000 と同様にノード単体でも並列実行やスカラージョブの多重処理が可能です。



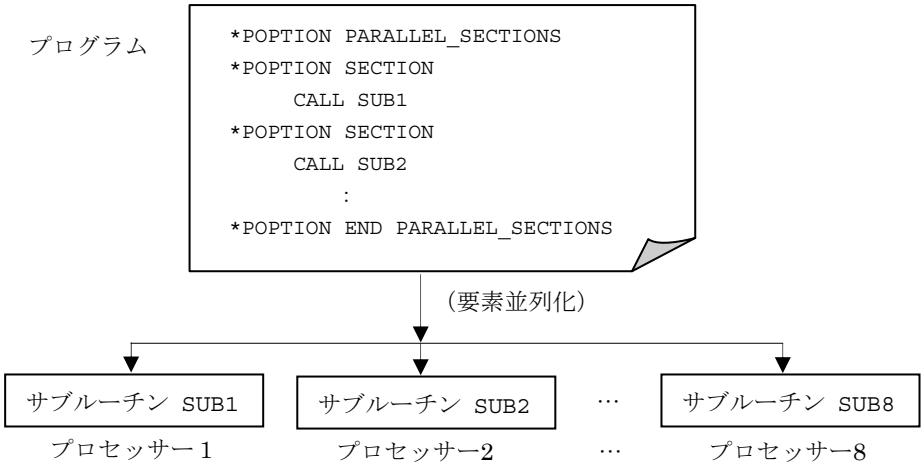
1.3. 要素並列処理

各ノードは，並列処理単位（スレッド）に分割したプログラムをノード内の複数のプロセッサーで並列実行する「要素並列」処理機能を備えています。コンパイルオプションの指定やソースプログラム中に指示文を記述することにより，コンパイラーは要素並列化を施したオブジェクトを生成します。以下に要素並列化変換の例を示します。

- ループ要素並列化（自動並列化）
  DO ループをスレッドに分割し，複数プロセッサーで並列に実行します。コンパイルオプションの指定によりコンパイラーが判断して自動的に変換します。



- SECTION 型要素並列化
  関数，サブルーチンなどの文の集まりをスレッドに分割し，複数プロセッサーで並列に実行します。利用者がソースプログラム中に指示文を記述することで並列化を指示します。

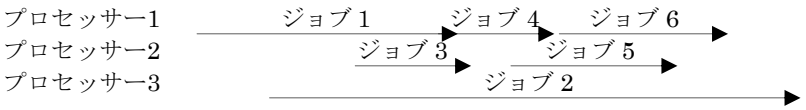


1.4. ジョブの形態

SR11000 および SR8000/MPP システムの各ノードは複数プロセッサーで構成されているため，要素並列処理だけでなく，以下に示すジョブ処理が可能です。さらにメッセージ通信ライブラリー（MPI）を組み合わせて使用することで複数ノードによる大規模な並列プログラムを実行することができます。

(1) スカラージョブ

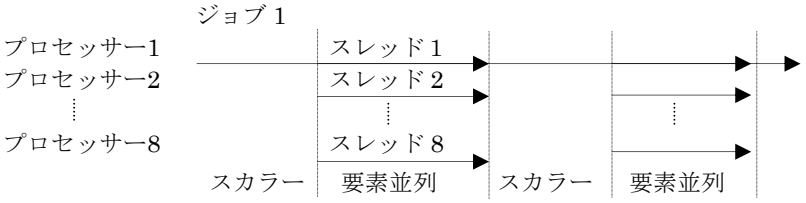
スカラープログラムを実行するジョブです。1 プロセッサーのみ使用するので他のジョブとノードを共有し，ノード内で多重実行します。



☞ スカラージョブクラス（A～F）または sr11000-s，mpp-s で実行します。

(2) 要素並列ジョブ

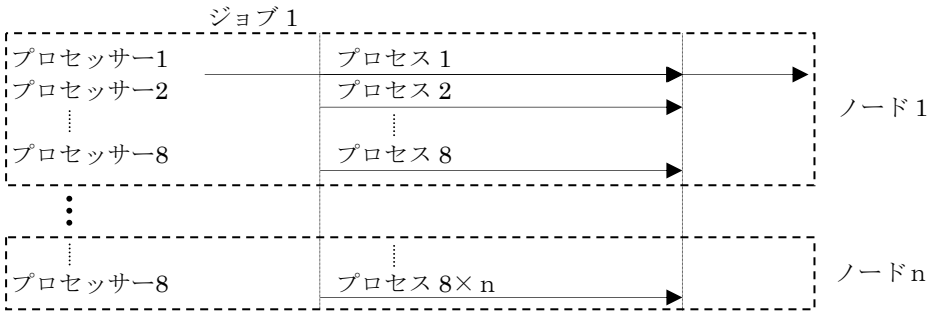
要素並列プログラムを実行するジョブです。要素並列化により分割されたスレッドが各プロセッサーに配置され並列に実行します。



☞ 並列ジョブクラス（P001～P008 等）で実行します。

(3) MPI ジョブ

MPI のプロセスが各プロセッサーに配置され，共有メモリーを使用して通信します。ノード内は最大 8 プロセスの並列実行ができます。n 台のノードを使用すると 8×n プロセス並列が実現できます。



☞ 並列ジョブクラス（P001～P008 等）で実行します。実行にはジョブタイプ SS（#@$-JSS）の指定が必要です。

2. コンパイルとオプション　※ 以降は SR11000 を基準に説明します。

2.1. コンパイルとリンク

FORTRAN プログラムのコンパイルには f90（最適化 FORTRAN90）または f77（最適化 FORTRAN77）コマンドを使用します。また，リンクにも f90，f77 コマンドを使用します。

```
f90 または f77
    ファイル名..
    [-c]
    [-o ロードモジュールファイル名]
    [-コンパイルオプション..]
    [-リンケージオプション..]
    [-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,言語仕様拡張オプション]
```

オプション

オプションは空白で区切り複数指定できます。左から右へ順に処理するので同一または背反するオプションは後ろに指定したものが有効になります。特にライブラリー名を指定する–l オプションはオプションの順序がコンパイル，リンクに影響するので注意が必要です。

☞ オプションの指定方法には –W0,'コンパイルオプション', –Wl,'リンケージオプション'と記述する方法がありますが，ここでは －コンパイルオプション，－リンケージオプションの形式を使用します。

なお，本センターでは幾つかのオプションをデフォルトとして設定しています。オプション無指定時にも以下のオプションが仮定されますので御注意下さい。（デフォルトオプションは都合により変更となる場合があります。）

SR11000	f90:	–e
	f77:	–e
SR8000/MPP	f90:	–i,P
	f77:	–i,P –Os –pvec –noparallel –symnchk

デフォルトオプションは環境変数 F90OPTS（または F77OPTS）で利用者毎に変更することができます。（特に必要がない場合には変更しないで下さい。）

```
% setenv  F90OPTS  ''                          デフォルトオプションを設定しない
% setenv  F90OPTS  '-O4  -parallel'            デフォルトオプションの設定例
```

ファイル名

以下のサフィックスを持つファイル名を指定するとファイルが FORTRAN ソースプログラムファイルであるとして FORTRAN コンパイラーを起動します。

.f （固定形式） .f77 .f90 .f95 （各言語仕様の自由形式）

以下は C 言語プリプロセッサーを通した後，FORTRAN コンパイラーに入力します。

.F（固定形式） .F77 .F90 .F95 （各言語仕様の自由形式）

ただし，オプションにより固定形式，自由形式の指定（–fixed，–free），または言語仕様の切り替え（–hf77，–hf90，–hf95），C 言語プリプロセッサー（–cpp）を指定した場合にはオプションが優先されます。サフィックスが .c または .i の場合は C 言語ソースプログラムであるとして C コンパイラーを起動します。また上記以外，オブジェクトファイル .o 等のファイルが指定された場合にはリンケージエディターを起動します。

コンパイル，リンクの例

```
% f90  main.f  sub.f                          main.f, sub.f をコンパイルして実行ファイル
main.f:                                       を作成
f90: compile start : main.f

*OFORT90 V01-04 entered.
*program name = MAIN                          （main.f のコンパイルメッセージ）
*end of compilation : MAIN
*program  units  =  0001,  no  diagnostics
generated.
sub.f:
f90: compile start : sub.f

*OFORT90 V01-04 entered.
*program name = SUB                           （sub.f のコンパイルメッセージ）
*end of compilation : SUB
*program  units  =  0001,  no  diagnostics
generated.                                    （この後リンクが行われる）
% ls
a.out   main.f   sub.f                        実行ファイル a.out ができる
%
```

オブジェクトモジュール作成後，リンクする例

```
% f90  -c  main.f  sub.f                      main.f, sub.f  をコンパイルしてオブジェク
 （省略）                                     トモジュールを作成
% ls                                          オブジェクトモジュール main.o, sub.o ができ
main.f  main.o  sub.f  sub.o                  る
%
% f90  -o  program  main.o  sub.o             main.o, sub.o  をリンクして実行ファイルを
% ls                                          作成
main.f  main.o  program  sub.f  sub.o         実行ファイル program ができる
%
```

2.2. オプションと機能

最適化 FORTRAN90（または最適化 FORTRAN77）で使用される主なオプションとその機能について説明します。代表的な使用例は次のとおりです。

```
% f90  -64  -Oss  program.f                    自動並列化を行うコンパイル
% f90  -64  -Oss  -omp  program.f              OpenMP 使用時のコンパイル
% f90  -64  -Oss  -noparallel  program.f       スカラープログラムのコンパイル
```

最適化オプション

最適化とは演算子の変更，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い，プログラムの実行速度を向上させる機能です。各種オプションが最適化機能毎に用意されていますが，以下に示すレベルに従って最適化することができます。

なお，最適化機能には副作用を含むものがあり，最適化によっては計算結果が異なったり，エラーが生じる場合がありますので十分理解した上でオプションを使用して下さい。

–O 最適化レベル ***最適化オプション***

% f90 -O4 program.f	最適化オプション-O4 でコンパイル
% f90 -Oss program.f	最適化オプション-Oss でコンパイル

オプション無指定の場合，SR11000 では f90 および f77 ともに -03，SR8000/MPP では f90 が -03，f77 が -0s -noparallel になります。

最適化オプション（SR11000）

-O0 (レベル 0)	原始プログラムどおりのコンパイル，レジスタの効果的な使用方法を中心に文の実行順序を変更しない一文単位の最適化（べき演算の乗算化，文関数のインライン化，局所的なレジスタの割り当て等）
-O3 (レベル 3)	プログラム全体での大域的な最適化（分岐命令の最適化，命令の並べ替え，演算子の変更，不変式のループ外への移動，外部手続きインライン展開等）
-O4 (レベル 4)	制御構造の変換，演算順序の変更を含むプログラム全体の最適化（演算順序の変更，除算の乗算化，ループ展開・交換・分配・融合等）
-Os	実行速度が速くなるようなコンパイルオプション※の自動設定（ソフトウェアパイプライン，要素並列化を含む各種最適化オプションの設定） ※ -nolcheck，-nodochk，-approx，-noconvcheck，-disbracket，-divmove，-expmove，-fma，-invariant_if，-ischedule=3，-loopdistribute，-loopexpand，-loopfuse，-loopinterchange，-loopreroll，-O4，-parallel=2，-prefetch，-prod，-rapidcall，-scope，-swpl，-noargchk，-nobreak，-noerstmt，-noagochk，-nosubchk，-workarray
-Oss	-Os に加えてさらに実行速度が速くなるようなコンパイルオプション※の自動設定（-Os に加えてライブラリーコード使用，ベクトル数学関数の引用等） ※ -approxlib，-ifswpl，-parallel=4，-pvfunc=2

高いレベルの最適化は，それより低いレベルの機能を含んでいます。また，一般にレベルが高い最適化ほどコンパイルに時間がかかります。

注）上記は SR11000 の説明です。SR8000/MPP についてはマニュアルを御覧下さい。

–loopdiag ***最適化ループ診断メッセージ***

```
% f90 -Os -loopdiag program.f                program.f をコンパイルしたときのループ
f90: compile start : program.f               構造診断メッセージを出力する

*OFORT90 V01-04 entered.
*program name = MAIN
*end of compilation : MAIN
   KCHF1805K
     the do 10 loop is fused with the following   (DO10 ループと後続のループを融合)
one. line=11
   KCHF1809K
     the do 10 loop is unrolled 4 times.          (DO10 ループを 4 回展開)
line=11
*program units = 0001, no diagnostics
generated.
```

オプションの詳細および個別の最適化オプションにつきましてはマニュアル「最適化 FORTRAN90 使用の手引（3000-3-C22)」または「最適化 FORTRAN77 使用の手引（3000-3-C24)」を御覧下さい。

最適化オプションと仮定されるオプションの関係を次に示します。

最適化 オプション	仮定される オプション	内容
-O3 （レベル 3）	-convcheck	データー型変換関数（aint,anint,nint）の引数範囲をチェックする
	-expmove	条件式下にあるループ不変式をループ外に移動する
	-ischedule=1	命令の実行順序を変更する
	-nomathinline	数学関数のインライン化をしない
	-scope	ループの最適化でスコープ分割をする
-O4 （レベル 4）	-O3	最適化レベル 3
	-arraycomm=1	演算操作が増加しない範囲で配列要素を共通化する
	-fma	浮動小数点乗加減算命令を使用する最適化をする
	-swpl	ソフトウェアパイプラインによる最適化をする
-Os	-O4	最適化レベル 4
	-nolcheck	新 FORTRAN 規格（JIS X 3001-1:1994 または JIS X 3001-1:1998）から拡張した仕様に対して，エラーメッセージを出力しない
	-nodochk	DO 文の繰り返し回数 0 をチェックしない
	-approx	除算で，逆数による乗数化を許可する
	-noconvcheck	データー型変換関数（aint,anint,nint）の引数範囲をチェックしない（注：-03,-04 と異なる）
	-disbracket	括弧や文の順序によって明示された演算順序を変更する
	-divmove	条件式下で除算割り込みが起こる可能性があるループ不変式をループ外に移動する
	-invariant_if	ループ不変条件展開最適化をする
	-ischedule=3	-ischedule=1 に加え，より大域的に命令の実行順序を変更する。また，分岐予測を行い投機的に実行順序を変更する。（注：-03,-04 と異なる）
	-loopdistribute	ループ分配による最適化をする
	-loopexpand	ループ展開による最適化をする
	-loopfuse	ループ融合による最適化をする
	-loopinterchange	ループ交換による最適化をする
	-loopreroll	ループ巻き戻しによる最適化をする
	-parallel=2	※ 次頁を参照
	-prefetch	最内側ループ中の配列に対してプリフェッチ最適化をする
	-prod	乗算の代わりに組込み関数 DPROD，QPROD を引用する
	-rapidcall	組込み関数に対して呼び出し時の引数チェックを行わない，引数を値渡しとする
	-noargchk	サブルーチンおよび関数を引用する際，引数の個数と型，属性，名称の重なりをチェックしない
	-nobreak	プログラム実行中の端末の割り込みシグナル（SIGINT）およびタイマ割り込み（SIGALRM）を受け付けない
	-noerstmt	トレースバックマップ中にエラーが発生した文の行番号を出力しない
	-noagochk	プログラム実行時に，割当て形 GO TO 文の文番号並びの有無をチェックしない
	-nosubchk	配列参照の添字の値が，宣言の範囲内にあるかどうかをチェックしない
	-workarray	作業配列を利用した最適化をしない
-Oss	-Os	最適化レベル S
	-approxlib	除算，逆数演算，SQRT 組込み関数呼び出しを実行時ライブラリーの呼び出しで計算する
	-ifswpl	条件分岐を含むループのソフトウェアパイプラインによる最適化をする
	-parallel=4	※ 次頁を参照
	-pvfunc=2	ベクトル数学関数を引用する（一時配列の導入含む）

演算順序が変更となる最適化（–approx，–disbracket 等）は浮動小数点演算での精度誤差が発生する場合があります。また，–expmove，–divmove は例外の発生する可能性があります。これらを回避するためには –no*option* を指定して原因となる最適化機能を抑止して下さい（例：–Os –noapprox)。反対に –no*option* を有効にするためには –*option* を指定して下さい（例：–Os –argchk)。その他の最適化においても副作用を含む場合がありますので御注意下さい。

☞ 最適化の副作用について詳細はマニュアル「最適化 FORTRAN90 使用の手引（3000-3-C22)」または「最適化 FORTRAN77 使用の手引（3000-3-C24)」を参照して下さい。

要素並列化オプション

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。メッセージ通信を行う並列化とは異なり，コンパイルオプションや指示文により利用者が並列処理自体をコーディングすることなく，プログラムを並列化することができます。

要素並列化を行うためには以下のオプションを指定してコンパイルします。また，オブジェクトモジュールをリンクする際にもリンケージオプション（–parallel）の指定が必要です。

–parallel=要素並列化レベル	*要素並列化オプション*
% f90 -parallel program.f	要素並列化（レベル２）する
% f90 -parallel=4 program.f	要素並列化（レベル４）する
% f90 -noparallel program.f	要素並列化を抑止
% f90 -parallel main.o sub.o	要素並列オブジェクトをリンクする場合

–parallel オプションにレベル指定がない場合は –parallel=2（レベル2）を仮定します。
–parallel リンケージオプションにはレベル指定は不要です。

要素並列化オプションの詳細は次のとおりです。

-noparallel -parallel=0	要素並列化をしない
-parallel=1	SECTION 型要素並列化，強制ループ要素並列化
-parallel -parallel=2	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
-parallel=3	ループ分配，ループ分割，ループの一重化，サイクリック分割
-parallel=4	パイプライン要素並列化，インダクション要素並列化

高いレベルの要素並列化は，それより低いレベルの機能を含んでいます。

要素並列化変換にはオプション指示でDOループを自動的に変換するループ要素並列化とパラメーター指示（POPTION）で文の集まりをプロセッサーに分配するSECTION型要素並列化があります。ループ要素並列化が適用される条件は以下のとおりです。

- SECTION型要素並列化指示された範囲内にあるDOループでない

- 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
  - 関数，手続き呼び出し
  - 入出力文
  - 終了・停止文（STOP文，PAUSE文，RETURN文など）
- DOループにNOPARALLEL指示がない
- DOループ内に同期制御または排他制御指示がない
- DOループの実行性能向上が見込める
- DOループ内に現れる変数または配列に，ループ繰り返しにわたる依存関係がない。

–pardiag ***要素並列化診断メッセージ***

```
% f90 -parallel -pardiag program.f          program.f　を要素並列化してコンパイルし
f90: compile start : program.f             たときの診断メッセージを出力する

*OFORT90 V01-04 entered.
*program name = MAIN
*end of compilation : MAIN
*end of compilation : _parallel_func_1_MAIN
(diagnosis for loop structure)
   KCHF2000K
     the do 10 loop is parallelized.line=7     (DO10 ループを要素並列化)
   KCHF2011K
     the variable(s) or array(s) in do 10 loop   (DO10 ループ内の変数を TLOCAL 化)
is applied to tlocal transformation.name=I
line=7
   KCHF2013K
     the final value is guaranteed for the     (DO10 ループ内の TLOCAL 化する変数の終
variable(s) or array(s) in do 10 loop subject   値保証を行う)
to tlocal transformation.name=I line=7
*program units = 0001, no diagnostics
generated.
%
```

オプションの詳細および個別の要素並列化オプションにつきましてはマニュアル「最適化 FORTRAN90 使用の手引（3000-3-C22)」または「最適化 FORTRAN77 使用の手引（3000-3-C24)」を御覧下さい。

要素並列化されたプログラムは，ノード内の複数のプロセッサー（通常は 8 台）を使用して実行します。このため，本センターではノードを専有できる並列ジョブクラス（P001～P008 等）でプログラムを実行する必要があります。

OpenMP

最適化 FORTRAN90 では OpenMP（OpenMP Fortran Application Program Interface Version 2.0, November 2000 ）を用いて要素並列化を行うことができます。OpenMP とは共有メモリー型の並列計算機における並列化の指示文やライブラリー，環境変数等を規定した規格です。

–omp ***OpenMP オプション***

```
% cat omp.f
      real a(1000)
```

```
!$omp parallel                          OMP 指示文の例
!$omp do
      do i=1,1000
       a(i)=i
      enddo
!$omp enddo
!$omp end parallel
      end
% f90 -parallel -omp omp.f              OpenMP を使用したプログラムのコンパイル
                                        -parallel オプションの指定が必要
```

OMP 指示文や OpenMP オプションにつきましてはマニュアル「最適化 FORTRAN90 使用の手引（3000-3-C22)」を御覧下さい。

OpenMP は最適化 FORTRAN77 ではコンパイルできません（–o mp オプションとして処理するため，エラーにはならず，mp という名の実行ファイルが生成されてしまいます)。

64 ビットアドレッシングモード

プログラムで使用するメモリーサイズが 2GB 以上の場合には，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必要があります。以下のオプションを指定してコンパイルして下さい。オブジェクトモジュールをリンクする際にも指定が必要です。

–64 ***64 ビットアドレッシングモードオプション***

```
% f90  -64  program.f                  64 ビットモードでコンパイルする
% f90  -64  main.o  sub.o              64 ビットモードのオブジェクトをリンクする
% f90       program.f                  32 ビットモードでコンパイルする
% f90  -32  program.f                  32 ビットモードでコンパイルする
```

```
% f90 program.f                        32 ビットモードでコンパイル
f90: compile start : program.f

*OFORT90 V01-04 entered.
   KCHF640K  16                        使用するメモリーサイズが 2GB を超える場合
             program size is too large. コンパイル時にエラーとなる
*program name = MAIN
*program units = 0001, 0001  diagnostics
generated, highest severity code is  16
% f90 -64 program.f                    64 ビットモードでコンパイル
f90: compile start : program.f

*OFORT90 V01-04 entered.
*program name = MAIN
*end of compilation : MAIN
*program  units  =  0001,  no  diagnostics
generated.                              コンパイル終了
%
```

実行時にメモリー不足となる場合にはメモリーの大きさを拡張して下さい。ジョブスクリプトの `#@$-lM` で設定します。以下は–64 オプション指定時の主な注意事項です。

- 名前付き定数および初期化項目には，2,147,483,647 バイトを超えるデーターは使用できない。
- ASSIGN 文，割り当て型 GOTO 文，および FMT 指定子の変数は，整数型 8 バイトでなければならない。
  - 型変更は利用者自身で行うか，オプション–intexp=full で全ての整数型の変数，配列，定数を整数型 8 バイトへ拡張（引数の整合性には注意）する必要がある。
- サービスサブルーチンの引数には，整数型 8 バイトは使用できない。

- サービスサブルーチンを使用している場合には引数を整数型 8 バイトを整数型 4 バイトに変換するオプション–exsrvc を指定する必要がある。

- オプション–hugeary が仮定され，以下の組込み関数は整数型 8 バイトを返す。(f90 のみ)
  - LBOUND，SHAPE，SIZE，UBOUND，COUNT，MAXLOC，MINLOC

ログメッセージ

ログメッセージ出力オプションを指定することで，コンパイル診断メッセージをファイルに出力することができます。ログメッセージファイルはソースプログラム毎に「ソースプログラム名.log 」というファイル名で出力されます。要素並列化等を適用した様子がソースプログラムに併記されるのでプログラムのチューニングが容易になります。

–loglist *ログメッセージ出力オプション*

```
% f90 -Oss -loglist program.f                    ログメッセージファイルを出力する
f90: compile start : program.f

*OFORT90 V01-04 entered.
*program name = MAIN
*end of compilation : MAIN
*end of compilation : _parallel_func_1_MAIN
*program  units  =  0001,  no  diagnostics
generated.

% ls                                              program.log が作成される
a.out        program.f    program.log

% cat program.log                                ログメッセージファイル program.log の内
      program main                                容を表示
      parameter (n=10000)
      real a(n), b(n), c(n)

**
** Parallel processing starting at loop entry    (並列処理開始)
** Parallel function: _parallel_func_1_MAIN
** Parallel loop                                 (並列ループ)
** --- 2 loops (DO10,DO20) fused (depth 1) ---   (DO10，DO20 ループ融合)
** Parallel processing finishing at loop exit
**
**    [DO 10]
**    Innermost loop unrolled (10 times).        (最内側ループ展開)
**    SWPL applied.                              (SWPL 適用)
**
 (省略)
```

ログメッセージは日本語で出力することができます。このときコンパイルメッセージも日本語になります（デフォルトは環境変数 LANG に設定された文字コード種別であり，LANG=C [英語]です)。

–listlang=文字コード種別

```
% f90 -Oss -loglist program.f -listlang=c      英語（ASCII）で出力
% f90 -Oss -loglist program.f -listlang=sjis   日本語（SJIS）で出力
% f90 -Oss -loglist program.f -listlang=euc    日本語（EUC）で出力
```

```
% f90 -Os -loglist program.f -listlang=sjis    ログメッセージファイルを出力する
f90: compile start : program.f                  （オプション –listlang=sjis　を指定）

*OFORT90 V01-04 開始                           コンパイルメッセージが日本語で表示され
```

```
*プログラム名 = MAIN                                  る（端末の文字コードを SJIS に設定すること）
*end of compilation : MAIN
*end of compilation : _parallel_func_1_MAIN
*プログラム数 = 0001 , エラーはありません。

% cat program.log                                    ログメッセージファイル program.log が
      program main                                   日本語で作成される
      parameter (n=10000)
      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
**  --- 2 個のループ(D010,D020)にループ融合(1
重)を行った ---
**  ループ出口で並列処理 終了
**
**    [DO 10]
**    最内側ループ展開(10 倍)を行った。
**    SWPL を適用した。
**
 （省略）
```

性能モニター

性能モニター情報出力オプションを指定することで実行時に性能，負荷等の情報を得ることができます。本オプションはリンク時にも指定が必要です。本オプションを指定してコンパイルしたプログラムを実行すると性能モニター情報ファイルが生成されます（”pm_実行ファイル名_日付_時間_ノード番号_プロセス番号”というファイル名で出力されます)。このファイルはバイナリー形式のため，**pmpr** コマンドを使用して参照します。日本語による出力も可能です（デフォルトは環境変数 LANG に設定された文字コード種別であり，LANG=C [英語]です)。

–pmfunc　性能モニター情報（プロセス単位，関数／手続き単位）出力オプション

```
% f90 -parallel program.f -pmfunc                   性能モニター情報を出力する実行ファイル
f90: compile start : program.f                      を作成

*OFORT90 V01-04 entered.
*program name = MAIN
*program name = SUB
*end of compilation : MAIN
*end of compilation : _parallel_func_1_MAIN
*end of compilation : SUB
*end of compilation : _parallel_func_2_SUB          このファイルはバイナリー形式のため，
*program units = 0002, no diagnostics              性能モニター情報出力コマンド pmpr
generated.                                          を使用してテキスト形式で表示する。
%
 （バッチジョブで a.out を実行）                   実行ファイル a.out を実行する
% ls
…
a.out
pm_a.out_Oct24_1139_n75_696542                      性能モニター情報ファイルが作成されてい
program.f                                           ることを確認

% pmpr pm_a.out_Nov02_1154_n12_659518               pmpr コマンドで情報をテキスト形式で表示
pm_a.out_Nov02_1154_n12_659518:                     （pmpr –j ファイル名とすると日本語 SJIS
 （省略）                                           で表示される）
```

```
############################################################
## Function/Procedure                                     ##
############################################################
========================================
== Function/Procedure Ranking          ==
========================================
       CPU time[%]        Times  Func(File+Line)
------------------------------------------------
[  1]  0.000065[ 88.24]        1  MAIN(program.f+5)
[  2]  0.000009[ 11.76]        1  SUB(program.f+20)
----------------------
TOTAL  0.000074[100.00]
```

性能モニターの使用法についてはスーパーコンピューティングニュース Vol.7 No.3, 2005.5「ベクトル並列型スーパーコンピューターSR11000 チューニングガイド」を御覧下さい。

要素並列化の単位で性能モニター情報を出力することもできます。同様にコンパイル，実行して以下のように参照します。

–pmpar　　性能モニター情報（プロセス単位，要素並列化単位）出力オプション

```
% f90 -parallel program.f -pmpar                性能モニター情報を出力する実行ファイル
f90: compile start : program.f                  を作成

*OFORT90 V01-04 entered.
*program name = MAIN
*program name = SUB
*end of compilation : MAIN
*end of compilation : _parallel_func_1_MAIN
*end of compilation : SUB
*end of compilation : _parallel_func_2_SUB      このファイルはバイナリー形式のため，
*program  units  =  0002,  no  diagnostics     性能モニター情報出力コマンド pmpr
generated.                                      を使用してテキスト形式で表示する。
%
（バッチジョブで a.out を実行）                 実行ファイル a.out を実行する
% ls
…
a.out
pm_a.out_Nov02_1159_n12_643084                  性能モニター情報ファイルが作成されてい
program.f                                       ることを確認

% pmpr pm_a.out_Nov02_1159_n12_643084           pmpr コマンドで情報をテキスト形式で表示
pm_a.out_Nov02_1159_n12_643084:                 （pmpr –j ファイル名とすると日本語 SJIS
（省略）                                        で表示される）
############################################################
## Element Parallel Region                                ##
############################################################
========================================
== Element Parallel Region Ranking     ==
========================================
       CPU time[%]         MFLOPS      MIPS    Times  Func[Rank](File+Line)
-------------------------------------------------------------------------------
[  1]  0.000005[ 52.76]  2097.188  10865.438        1  SUB[-](program.f+20)
[  2]  0.000004[ 47.24]  2342.337  12134.609        1  MAIN[-](program.f+8)
----------------------
TOTAL  0.000009[100.00]
（省略）
```

性能モニターの使用法についてはスーパーコンピューティングニュース Vol.7 No.3, 2005.5「ベクトル並列型スーパーコンピューターSR11000 チューニングガイド」を御覧下さい。

pmpr コマンドによる文字コード種別の指定例

```
% pmpr  -e    性能モニター情報ファイル名      英語（ASCII）で出力
% pmpr  -j    性能モニター情報ファイル名      日本語（SJIS）で出力
% pmpr  -euc  性能モニター情報ファイル名      日本語（EUC）で出力
```

ソースプログラム差分コンパイル

ソースプログラム差分コンパイル機能はオブジェクトファイル中にソースプログラム情報を保持し，再度そのオブジェクトを出力先としてコンパイルしたときには，ソースプログラムの変更があったプログラム単位のみコンパイルする機能です。これによりコンパイル時間を短縮することができます。

–diffcomp　　ソースプログラム差分コンパイルオプション

```
% f90  -diffcomp  program.f                差分コンパイルオプションを指定する
f90: compile start : program.f            通常と同様にコンパイルを開始

*OFORT90 V01-04 entered.
*program name = MAIN
*program name = SUB
*end of compilation : MAIN                プログラム MAIN コンパイル終了
*end of compilation : SUB                 プログラム SUB コンパイル終了
*program  units  =  0002,  no  diagnostics
generated.
% ls
a.out      program.f    program.o         オブジェクトファイル program.o および実
                                          行ファイル　a.out が作成される
～ プログラム SUB を編集 ～

% f90  -diffcomp  program.f                プログラム SUB の内容を変更後，改めて差
f90: compile start : program.f            分コンパイルする

*OFORT90 V01-04 entered.                  プログラム MAIN に変更がないので構文
*program name = MAIN                      チェックのみ行う
*program name = SUB                       プログラム SUB は変更があったのでコンパ
*end of compilation : SUB                 イルを行う
*program  units  =  0002,  no  diagnostics メッセージ「*end of compilation : MAIN」が
generated.                                ないことに注意
%
```

オブジェクト再コンパイル

オブジェクト再コンパイル機能はオブジェクトファイル中に保持しているソースプログラム情報を使用し，オブジェクトを入力として再コンパイルする機能です。

–recomp　　オブジェクト再コンパイルオプション

```
% f90  -diffcomp  program.f                差分コンパイルオプションを指定する
f90: compile start : program.f            通常と同様にコンパイルを開始

*OFORT90 V01-04 entered.
*program name = MAIN
*program name = SUB
*end of compilation : MAIN                プログラム MAIN コンパイル終了
*end of compilation : SUB                 プログラム SUB コンパイル終了
*program  units  =  0002,  no  diagnostics
generated.                                オブジェクトファイル program.o および実
                                          行ファイル a.out が作成される
% rm  program.f                           試しにソースプログラム program.f を消し
% ls                                      てみる（必要なファイルは消さないこと）
a.out      program.o

% f90  -recomp  -parallel  program.f       再コンパイルオプションを指定してコンパ
f90: compile start : program.f            イル（参考のため–parallel オプション追加）

*OFORT90 V01-04 entered.                  オブジェクトモジュール内のソースプログ
*program name = MAIN                      ラム情報を利用して再コンパイルを行う
```

```
*program name = SUB
*end of compilation : MAIN                    プログラム MAIN コンパイル終了
*end of compilation : SUB                     プログラム SUB コンパイル終了
*program  units  =  0002,  no  diagnostics
generated.
%
```

制限コンパイル

通常コンパイラーは最大実行性能の実行ファイルを作成しようとするために，コンパイル時間やコンパイル時使用メモリー量は無制限に使用して最適化を行いますが，時間がかかり過ぎる場合やコンパイル時のメモリー不足の場合に，これらに制限を設けてコンパイルすることができます。

–limit ***制限コンパイルオプション***

```
% f90  -limit    program.f                    制限コンパイルを行う
% f90  -nolimit  program.f                    制限コンパイルを行わない
```

オプション無指定の場合，SR11000 では –limit がデフォルト，SR8000/MPP では –nolimit がデフォルトです。

デバッグオプション

デバッグ情報を表示します。

–debug ***デバッグオプション***

```
% f90  -debug    program.f                    デバッグ情報を表示する
% f90  -argchk=all  program.f                 引数の個数，型をチェックする
```

–debug

以下に示すオプションを設定し，デバッグ情報を出力します。

–lcheck –dochk –argchk=all –dline –erstmt –agochk –subchk

–lcheck

新 Fortran 規格（JIS X 3001-1:1994，または JIS X 3001-1:1998）から拡張した仕様に対して，エラーメッセージを出力します。

–dochk

DO 文の繰り返し回数が 0 の場合，エラーメッセージを出力します。

–argchk=all

関数，サブルーチンの引数の個数，型，属性の不一致をチェックし，実行時にエラーメッセージを出力します。また，引数並びに同一名称が指定されている場合にコンパイル時にメッセージを出力します。

–subchk

配列参照の添字の値が宣言の範囲内にない場合，実行時にエラーメッセージを出力します。

☞ 関数の引数や配列参照に問題があるとプログラムは正常に動作しないため，デバッグオプションを指定していてもエラーにならないことがあります。（結果不正の可能性があります。）また，以下のエラーメッセージを出力し，プログラムが異常終了することがあります。

```
KCHF446R  segmentation violation occurred.
```

トレースバックマップ

エラーを検出したプログラムのトレース情報を実行時にエラー発生場所［ファイル名：行番号］の形式で出力します。これにより，エラー発生箇所を特定できる場合があります。

–s,TRACE ***トレース情報オプション***

% f90 -s,TRACE program.f	トレースバックマップ出力を行う

実行時にエラーメッセージと共に出力されます。（例 0x100003b4　MAIN+0x34　[a.f:4]）

言語仕様の拡張

他の FORTRAN 処理系でコンパイルできるプログラムが本センターのシステムでコンパイルエラーとなるとき，以下のオプション指定により対応できる場合があります。なお，オプションの詳細についてはマニュアルを参照して下さい。

–e

以下に示すオプションを設定し，コンパイルの制限を緩和，および実行モードの切り替えによる言語仕様の拡張を行います。SR11000 ではこのオプションをデフォルトで設定しています。

コンパイルオプション（f90）

–i,P –i,PL –conti199 –h8000 –intptr –commentinclude=asterisk –excmplx

コンパイルオプション（f77）

–i,U –i,P –i,PL –conti199 –h8000 –intptr –commentinclude=asterisk –typeparam

実行時オプション（f90 および f77 共通。実行時に自動的に設定される。）

–F'port(econv,eofback(0),eofrd,eofrdt,getarg,getenv,iargc,msgout(stderr),
nmlist,nscrach,prcntl,realedt,rewnocl,stdunit,tabsp),runst(damnonl,umask)'

–i,U

外部手続き名称中の_（アンダースコア）および 31 文字までの外部手続き名称を使用できます（f77 のみ）。

–i,P

以下の項目について言語仕様を拡張します。

- デバッグ行（D，？），タブコードの扱い
- ＄型，NL 型，O 型，Q 型編集
- 組込み関数（%VAL，%REF）
- 実定数が上限，下限値を超える場合の仮定処理
- DO WHILE 文
- DOUBLE COMPLEX 型　等

–i,PL

16 進定数（X'xx'または'xx'X）が使用できます。

–i,EU

外部手続き名（モジュール名を除く）の末尾にアンダースコアを付加します。

–i,LT

"（ダブルクォーテーション）で囲まれた文字列を定数とみなします。指定がないとき"以降，行の最後までを注釈とみなします。（f77 のみ）

–h8000

OS7 FORTRAN 言語仕様に合わせてコンパイルします。マスキング式，マスキング代入文，BOOL 組込み関数，論理定数を使用できます。

–cpp

C 言語プリプロセッサー呼び出しを行います。

2.3. 実行時オプション

実行時オプションは実行時環境の変更や他社 FORTRAN との互換性に対応するためのオプションです。ロードモジュール（実行ファイル）を実行するときに，以下のように「-F」に続けてアポストロフィーで囲んで指定します。カンマで区切ることで複数のオプションを指定することもできます。

ロードモジュール名　-F'実行時オプション,実行時オプション,…'

実行時オプションはサブオプションを持っており，括弧で囲んで指定します。カンマで区切ることで複数のサブオプションを指定することができます。例えばロードモジュール名が a.out の場合，以下のように記述します。（実行時オプションは大文字でも，小文字でも構いません）

```
%  ./a.out -F'port(fortuf,host),runst(uflow)'
```

主な実行時オプションを紹介します。

runst(uflow)

標準では浮動小数点演算中にアンダーフローが発生した場合でも処理を継続しますが，本オプション指定時にはエラーメッセージを出力してプログラムを終了します。

runst(oflow)

標準では浮動小数点演算中にオーバーフローが発生した場合でも処理を継続しますが，本オプション指定時にはエラーメッセージを出力してプログラムを終了します。

☞ runst(fpecntl(0)) を指定すると，0 を 0 で割る除算例外の場合でもプログラムを終了します。

port(econv)

E 形，ES 形，EN 形，または G 形編集記述子で指数部を表示する場合，倍精度の実数，拡張精度の実数，または複素数の指数表示文字を「E」とします。このオプションがない場合は倍精度が「D」，拡張精度が「Q」の指数表示文字となります。（デフォルト）

port(realedt)

実数型データー0.0 を E 形，D 形，Q 形，または G 形編集記述子で出力するときに指数部を付加します（例 0.00E+00）。また，小数点以下の数値列が書式の長さに満たない場合，書式の長さに合わせて 0 を挿入します（例 0.200000000　E+01 は 0.200000000000E+00 となる）。（デフォルト）

port(tabsp)

入力データーに TAB コードが含まれる場合，空白として扱います。（デフォルト）

port(prcntl)

画面，またはプリンタに書式付き記録を出力する場合，記録の先頭の一文字目を印刷制御文字として扱いません。（デフォルト）

port(dstduf)

書式なし入出力に対応するファイル形式を業界標準書式なしファイルとします。このファイル形式は他社 FORTRAN との互換形式として使用します。（デフォルト）

port(stduf)

書式なし入出力文に対応するファイル形式を標準書式なしファイルとします。このオプションは BACKSPACE 文の動作を除けば port(dstduf)オプションと同じです。

port(fortuf)

書式なし入出力に対応するファイル形式を FORTRAN 固有ファイルとします。 このファイル形式は運用支援システム（HI-OSF/1-MJ）の標準のファイル形式と互換性があります。ただし，VOS3 の場合はファイル変換が必要です。

☞ 書式なしファイル形式については「5.2. ファイル形式」を参照して下さい。

port(ppc)

書式なし入出力文に対応するデーター形式を PowerPC(SR8000)形式として扱います。 単精度および倍精度浮動小数点数については IEEE 形式に準拠しています。（デフォルト）

port(host)

書式なし入出力文に対するデーター形式を M 形式（VOS3，HI-OSF/1-MJ）として扱います。

☞ 装置番号 n について変換する場合や入力と出力で形式が異なる場合には本オプションは使用せず，以下の実行時オプション `runst(cvin(m(n)))`または `runst(cvout(m(n)))`を指定します。

runst(cvin(m(n)))

装置番号 n の書式なし入出力文に対するデーター形式を M 形式（VOS3，HI-OSF/1-MJ）から PowerPC(SR8000)形式に変換して入力します。（n はカンマで区切って複数指定できます。）

runst(cvout(m(n)))

装置番号 n の書式なし入出力文に対するデーター形式を PowerPC(SR8000)形式から M 形式（VOS3，HI-OSF/1-MJ）に変換して出力します。（n はカンマで区切って複数指定できます。）

2.4. サービスサブルーチン

JIS Fortran の組込み関数以外にサービスサブルーチンと呼ばれる最適化 FORTRAN90 または最適化 FORTRAN77 が用意しているサブルーチンがあります。そのなかでよく使われる CPU 時間や経過時間を計測するサービスサブルーチンと他社 FORTRAN で標準的にサポートされているサービスサブルーチンを使用する方法を以下に紹介します。

プログラムの時間計測

プログラムのある部分の実行に要した時間を計測する場合には以下の clock または xclock サブルーチンを使用します。なお，本サブルーチンの使用についてはコンパイル，リンク時の特別な指定は必要ありません。

CPU 時間計測の例

```
    real t                          変数 t は実数型 4 バイト（単精度）
    call clock                      CPU 時間測定タイマー起動
   -- プログラム --
    call clock(t)                   タイマー起動後の CPU 時間（秒）を変数 t
    write(*,*) t                    に代入
    stop
    end

 （高精度タイマー使用の場合）
    real*8 t                        変数 t は実数型 8 バイト（倍精度）
    call xclock                     CPU 時間測定タイマー起動
   -- プログラム --
    call xclock(t)                  タイマー起動後の CPU 時間（秒）を変数 t
    write(*,*) t                    に代入
    stop
    end

経過時間計測の例
    real t                          変数 t は実数型 4 バイト（単精度）
    call clock(t,7)                 経過時間測定タイマー起動
   -- プログラム --
    call clock(t,8)                 タイマー起動後の経過時間（秒）を変数 t に
    write(*,*) t                    代入
    stop
    end

 （高精度タイマー使用の場合）
    real*8 t                        変数 t は実数型 8 バイト（倍精度）
    call xclock(t,7)                経過時間測定タイマー起動
   -- プログラム --
    call xclock(t,8)                タイマー起動後の経過時間（秒）を変数 t に
    write(*,*) t                    代入
    stop
    end
```

オプションの詳細および個別のサービスサブルーチンにつきましてはマニュアル「最適化 FORTRAN90 言語（3000-3-C21-30）」または「最適化 FORTRAN77 言語（3000-3-C23-20）」を御覧下さい。

他社 FORTRAN で標準的にサポートされているサービスサブルーチン

コンパイル，リンク時のオプションに `-lf90c`（FORTRAN77 では `-lf77c`）を指定することで以下のサービスサブルーチンを使用することができます。

```
ABORT, ACCESS, ALARM, BIC, BIS, BIT, CHDIR, CHMOD, CLOCKM, CTIME,
DTIME, ETIME, FDATE, FGETC, FORK, FPUTC, FREE, FSEEK, FSEEKO64,
FSTAT, FSTAT64, FSYNC, FTELL, FTELLO64, GETC, GETCWD, GETFD, GETGID,
GETLOG, GETPID, GETUID, GMTIME, HOSTNM, IDATE, IERRNO, INMAX, ISATTY,
ITIME, KILL, LINK, LSTAT, LSTAT64, LTIME, MALLOC, PUTC, QSORT, RENAME,
SECOND, SETBIT, SIGNAL, SLEEP, STAT, STAT64, SYMLNK, SYSTEM, TIME,
TTYNAM, UNLINK, WAIT
```

```
% f90  program.f  -lf90c                    サービスサブルーチンを使用（f90）
% f77  program.f  -lf77c                    サービスサブルーチンを使用（f77）
```

オプションの詳細および個別のサービスサブルーチンにつきましてはマニュアル「最適化 FORTRAN90 言語（3000-3-C21)」または「最適化 FORTRAN77 言語（3000-3-C23)」を御覧下さい。

3. 並列アプリケーション

並列アプリケーションには，複数ノードで同一のプログラムをパラメーターを変えて実行する並列実行コマンドと，各プロセッサーで実行されるプログラムが通信を行いながら一つの計算を行う MPI プログラムの 2 種類があります。

3.1. 並列実行（prun 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プロセス番号等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 (sr11000-s)

```
% cat a.f                                  … サンプルプログラム
        read(5,*) a,b
        write(6,*) a+b,a-b
        stop
        end

% f90 a.f                                  … コンパイル
f90: compile start : a.f

*OFORT90 V01-04 entered.
*program name = MAIN
*end of compilation : MAIN
*program  units  =  0001,  no  diagnostics
generated.

% cat sample.def                           … 定義ファイル
*2 ./a.out < data.%n > result.%n              2 ノードを使用して 2 種類のデーター
                                              data.1, data.2 をノード毎に入力し，結
% ls                                          果をそれぞれ result.1, result.2 に出力
a.f      data.1     job.csh                   する。
a.out    data.2  sample.def

% cat job.csh                               … ジョブスクリプト例
#!/bin/csh
```

```
#@$-q personal                                          personal キュー使用
#@$-N 2                                                2 ノード使用
cd  {sample.def のあるディレクトリーを指定}
prun -f sample.def                                     prun コマンドを使用
                                                       （定義ファイル指定）
% qsub job.csh
Request xxxxx.n81 submitted to queue: personal.   … ジョブを投入

～ バッチジョブ終了後 ～

% ls
a.f       data.1      job.csh       result.2
a.out     data.2     result.1     sample.def        … result.1, result.2 が作成される
```

定義ファイルは″*ノード数　プログラム < 入力 > 出力″と記述します。
また，以下の記号を使用できます。

定義ファイル記述用の記号

リダイレクト入出力指定		ファイル名修飾	
<	標準入力	%n	1～並列プロセス数の数値
>	上書き 標準出力	%r	各プロセスが動作している相対ノード番号
>&	〃 標準エラー出力	%a	各プロセスが動作している絶対ノード座標
>>	追記 標準出力	%t	prun が起動した時刻
>>&	〃 標準エラー出力	%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

3.2. 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。なお，本センターのスーパーコンピューターシステムでは MPI-2 をサポートしています。

使用例（sr11000-s）

```
% 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(*,*) 'proc=',rank, 'size=',size

      call MPI_FINALIZE(ierr)
      stop
      end

% mpif90 mpi_sample.f                               … コンパイル
f90: compile start : mpi_sample.f                     MPI コンパイルコマンドを使用
```

```
*OFORT90 V01-04 entered.
*program name = SAMPLE
*end of compilation : SAMPLE
*program  units  =  0001,  no  diagnostics
generated.

% cat job.csh                                    … ジョブスクリプト例
#!/bin/csh
#@$-q personal                                    ジョブクラス personal を使用
#@$-N 2                                          2 ノード使用
#@$-J SS                                         ジョブタイプ SS を設定
cd  {a.out のあるディレクトリーを指定}
mpirun ./a.out                                   MPI 実行コマンドを使用
                                                 (2 ノード 16 プロセスで実行)
% qsub job.csh                                  … ジョブを投入
Request xxxxx.n81 submitted to queue: personal.

～ バッチジョブ終了後 ～

% cat job.csh.oxxxxx                            … 結果を表示
 proc=           0 size=          16
 proc=           1 size=          16
                 :
 proc=          15 size=          16
```

MPI 使用における注意点

(1) MPI を使用したプログラムのコンパイルには **mpif90** または **mpif77**（C 言語は **mpicc**, C++は **mpiCC**）という専用のコマンドが用意されています。これらのコマンドは MPI を使用する場合に必要なヘッダーファイルやリンクするライブラリー, コンパイルオプションを自動的に設定し, f90 または f77 コンパイラーを起動します。

MPI プログラムのコンパイルオプション指定例を以下に示します。

```
% mpif90  -o  program  program.f          実行ファイル program を作成する
% mpif77  program.f                       f77 でコンパイルする
% mpif90  -64  program.f                  64 ビットモードでコンパイルする
```

(2) MPI は同一ノードで複数のプロセスを同時に実行することもできます。例えば, P008 キュー（8 ノード）を使用すると 64 プロセッサーの並列計算が可能となります。使用例にジョブタイプ SS（#@$-J SS）という記述がありますが, MPI プログラムがノード内の全てのプロセッサーを使用するためにはジョブタイプ SS（スカラープログラムによるノード共有属性）の指定が必要です。なお, この設定においては要素並列化されたプログラムの実行はできません。

(3) MPI プログラムの実行には **mpirun** コマンドを使用します。mpirun のプロセス数はジョブスクリプトで指定したノード数とジョブタイプで決定されます。ジョブタイプに SS を指定すると, プロセス数はノード数×8 となります。ジョブタイプを指定しない場合はノード数と同じプロセス数となります。

ジョブスクリプト例 1

```
#@$-q personal
#@$-N 2
mpirun ./a.out  → 2 ノード 2 プロセスで実行します。
```

ジョブスクリプト例 2

```
#@$-q personal
#@$-N 2
```

```
#@$-J SS
mpirun ./a.out  → 2 ノード 16 プロセスで実行します。
```

※ SR8000/MPP ではオプションによる指定が可能です。
例：mpirun -n 2 -np 16 ./a.out → 2 ノード 16 プロセスで実行
参考マニュアル 「MPI・PVM 使用の手引」(6A30-3-026)

(4) 同一ノードで複数のプロセスを実行するとメモリー不足で実行時エラー（System error: Not enough space）となることがあります。メモリー制限値，プログラムサイズを確認して下さい。

(5) MPI プログラムでは送信と受信が一対一に対応していないと受信または送信側のプロセスが待ち続け，処理が進まない「デッドロック」の状態に陥ることがあります。プログラム作成の際には御注意下さい。

4. 数値計算ライブラリー

SR11000 システムには数値計算ライブラリーとして MATRIX/MPP，MSL2，BLAS，LAPACK，ScaLAPACK があります。これらのライブラリーを利用するには f90，f77 コマンドのオプションとして

–L ライブラリー検索パス名
–l ライブラリー名

を指定します。–l オプションはプログラムファイル名の後ろに指定します。このオプションは左から順に処理されるのでライブラリー名を指定する順序には注意して下さい。なお，センター提供の数値計算ライブラリーの検索パスは標準で設定されていますので，–L オプションは省略できます。

4.1. MATRIX/MPP

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/MPP を使用する場合にはコンパイル時にオプションとして以下のライブラリーを指定します。（要素並列版は–parallel オプションも同時に指定して下さい。）

MATRIX/MPP	（要素並列版）	–lmatmpp
	（スカラー版）	–lmatmpp_sc
MATRIX/MPP/SSS（スカイライン法）	（要素並列版）	–lmatmpps
	（スカラー版）	–lmatmpps_sc

参考マニュアル 「行列計算副プログラムライブラリ MATRIX/MPP」(3000-3-C87)
「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」(3000-3-C88)

使用例（単一ノードの例, sr11000-s）

```
% cat hsru1m.f                                  … サンプルプログラム
      parameter ( n=10 )                          逐次処理用インターフェースの例
      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

% f90 hsru1m.f -lmatmpp_sc                     … コンパイル
f90: compile start : hsru1m.f                    スカラー版ライブラリーを使用
                                                 (スカラージョブクラスまたはsr11000-sで
*OFORT90 V01-04 entered.                         実行できます。)
*program name = MAIN
*end of compilation : MAIN
*program  units  =  0001,  no  diagnostics
generated.

% ./a.out                                      … 実行*
           1  0.148270369                         a.out を実行する
           2  0.158839539
           3  0.645628750
                 :
          10  0.629399896

% f90 hsru1m.f -lmatmpp -parallel              … コンパイル
f90: compile start : hsru1m.f                    要素並列版ライブラリーを使用
                                                 (実行はバッチジョブ)
*OFORT90 V01-04 entered.
*program name = MAIN
*end of compilation : MAIN
*program  units  =  0001,  no  diagnostics
generated.

% cat job.csh                                  … ジョブスクリプト例
#!/bin/csh
#@$-q personal
#@$-N 1
cd {a.out のあるディレクトリーを指定}
./a.out

% qsub job.csh                                 … ジョブを投入
Request xxxxx.n81 submitted to queue: personal.

～ バッチジョブ終了後 ～

% cat job.csh.oxxxxx                          … 結果を表示
           1  0.148270369
           2  0.158839539
           3  0.645628750
                 :
          10  0.629399896
```

使用例（複数ノード実行の例, sr11000-s）

```
% cat hdru3mdp.f                               … サンプルプログラム
      parameter ( n=20,npu=2 )
      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)
      call hkxpara(ienv)
      me=ienv(1)
      ix=0
      iopt1=1
      iopt2(1)=1
      iopt2(2)=1
      call  hdru3mdp(n,ix,lstpu,npu,iopt1,        並列処理用インターフェース
    & iopt2,x,ier)                                (2 ノードを使用する例)
      write(6,*) me, n, ier
      do 20 i=1,n
        write(6,*) i,x(i)
  20  continue
      end

% mpif90 hdru3mdp.f -lmatmpp -parallel          … コンパイル
f90: compile start : hdru3mdp.f                   mpif90 コマンドおよび要素並列版ライブ
                                                  ラリーを使用 (実行はバッチジョブ)
*OFORT90 V01-04 entered.
*program name = MAIN
*end of compilation : MAIN
*program  units  =  0001,  no  diagnostics
generated.

% cat job.csh                                   … ジョブスクリプト例
#!/bin/csh
#@$-q personal
#@$-N 2
cd  {a.out のあるディレクトリーを指定}
mpirun ./a.out

% qsub job.csh                                  … ジョブを投入
Request xxxxx.n81 submitted to queue: personal.

～ バッチジョブ終了後 ～

% cat job.csh.oxxxxx                           … 結果を表示
          0          20           0
          1  0.108420217337368285E-005
                     :
```

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

MSL2	(要素並列版)	–lMSL2P
	(スカラー版)	–lMSL2

参考マニュアル　「数値計算副プログラムライブラリ MSL2 操作」(3000-3-C86)
「数値計算副プログラムライブラリ MSL2 行列計算」(3000-3-C83)
「数値計算副プログラムライブラリ MSL2 関数計算」(3000-3-C84)
「数値計算副プログラムライブラリ MSL2 統計計算」(3000-3-C85)

使用例（スカラー処理の例, sr11000-s）

```
% 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)
      do 10 i=1,n
        write(6,*) i,x(i)
   10 continue
      end

% f90 msgu1m.f -lMSL2                           … コンパイル
f90: compile start : msgu1m.f                     スカラー版ライブラリーを使用

*OFORT90 V01-04 entered.
*program name = MAIN
*end of compilation : MAIN
*program  units  =  0001,  no  diagnostics
generated.

% ./a.out                                       … 実行
           1  0.148270369                         a.out を実行する
           2  0.158839539
           3  0.645628750
              :
          10  0.629399896
```

使用例（要素並列処理の例, sr11000-s）

```
% cat msvafm.f                                   … サンプルプログラム
       parameter (n=2,m=2)
       real a(n,m),b(n,m),r(n,m)
       iopt=2
       data ((a(i,j),i=1,n),j=1,m)/1,2,3,4/
       data ((b(i,j),i=1,n),j=1,m)/1,2,3,4/
       call msvafm(a,n,m,n,b,n,iopt,r,n,ier)
       write(*,*) ((r(i,j),i=1,n),j=1,m)
       stop
       end

% f90 msvafm.f -lMSL2P -parallel                … コンパイル
f90: compile start : msvafm.f                     要素並列版ライブラリーを使用

*OFORT90 V01-04 entered.
*program name = MAIN
*end of compilation : MAIN
*program  units  =  0001,  no  diagnostics
generated.

% cat job.csh                                   … ジョブスクリプト例
#!/bin/csh
#@$-q personal
#@$-N 1
cd  {a.out のあるディレクトリーを指定}
./a.out
```

```
% qsub job.csh                                        … ジョブを投入
Request xxxxx.n81 submitted to queue: personal.

～ バッチジョブ終了後 ～

% cat job.csh.oxxxxx                                  … 結果を表示
   2.00000000        4.00000000        6.00000
000        8.00000000
```

4.3. BLAS・LAPACK・ScaLAPACK

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，LAPACK（Linear Algebra PACKage）は連立 1 次方程式，固有値，固有ベクトルなどの線形計算ライブラリー，ScaLAPACK（Scalable Linear Algebra PACKage）は並列版の行列計算ライブラリーです。これらのライブラリーの機能の詳細は以下のウェブページを御覧下さい。

http://www.netlib.org/blas/
http://www.netlib.org/lapack/
http://www.netlib.org/scalapack/

これらのライブラリーを使用する場合にはコンパイル時にオプションとして以下のライブラリーを指定します。（要素並列版は –parallel オプションも同時に指定して下さい。）

	（要素並列版）	（スカラー版）	
BLAS	–lblas	–lblas_sc	
LAPACK	–llapack –lblas	–llapack_sc –lblas_sc	
ScaLAPACK	–lscalapack –lblacsBASE –lblacsF77 –lblas	–lscalapack_sc –lblacsBASE_sc –lblacsF77_sc –lblas_sc	(ScaLAPACK) (BLACS Base) (BLACS Fortran) (BLAS)

ライブラリーオプションの指定順序は使用例を参照して下さい。

なお，ScaLAPACK の通信関数には MPI を使用していますのでコンパイルには MPI ライブラリーの指定も必要です。（mpif90，mpif77 コマンド使用の場合は MPI ライブラリーの指定は省略できます。）

使用例（BLAS，sr11000-s）

```
% cat blas.f                                          … サンプルプログラム
      program blas
      parameter ( n=4 )
      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

% f90 blas.f -L/usr/local/lib -lblas -parallel     … コンパイル
f90: compile start : blas.f                          要素並列版ライブラリーを使用

*OFORT90 V01-04 entered.
*program name = BLAS
*end of compilation : BLAS
*program units = 0001, no diagnostics generated.

% cat job.csh                                       … ジョブスクリプト例
#!/bin/csh
#@$-q personal
#@$-N 1
cd  {a.out のあるディレクトリーを指定}
./a.out

% qsub job.csh                                      … ジョブを投入
Request xxxxx.n81 submitted to queue: personal.

～ バッチジョブ終了後 ～

% cat job.csh.oxxxxx                               … 結果を表示
   1.12500000
   2.00000000
   3.50000000
   4.00000000
```

使用例（LAPACK, sr11000-s）

```
% cat lapack.f                                     … サンプルプログラム
       program lapack
       parameter (n=2,lda=n+1,ldb=n+1,nrhs = 1)
       double precision a(lda,n),b(ldb,nrhs)
       integer ipiv(n),info
       data ((a(i,j),j=1,n),i=1,n)/2d0,4d0,1d0,
     & 1d0/
       data (b(i,1),i=1,n)/14d0,5d0/
       call dgesv(n,nrhs,a,lda,ipiv,b,ldb,info)
       do 10 i=1,n
       write(*,*) b(i,nrhs)
 10    continue
       stop
       end

% f90 lapack.f -L/usr/local/lib -llapack -lblas    … コンパイル
-parallel                                            要素並列版ライブラリーを使用
f90: compile start : lapack.f

*OFORT90 V01-04 entered.
*program name = LAPACK
*end of compilation : LAPACK
*program units = 0001, no diagnostics generated.

% cat job.csh                                       … ジョブスクリプト例
#!/bin/csh
#@$-q personal
#@$-N 1
```

```
cd  {a.out のあるディレクトリーを指定}
./a.out

% qsub job.csh                                          … ジョブを投入
Request xxxxx.n81 submitted to queue: personal.

～ バッチジョブ終了後 ～

% cat job.csh.oxxxxx                                    … 結果を表示
   3.00000000000000000
   2.00000000000000000
```

使用例（ScaLAPACK, sr11000-s）

```
% mpif90 -Oss -noparallel example1.f -i,L                … コンパイル（mpif90 コマンド）
  -L/usr/local/lib -lscalapack_sc -lblacsF77_sc
  -lblacsBASE_sc -lblas_sc                                サンプルプログラムは
f90: compile start : example1.f                           http://www.netlib.org/scalapack/ex
                                                        amples/
*OFORT90 V01-04 entered.                                  にある example1.f を使用
*program name = example1
   KCHF476K  00               DESCB
             the variable is declared, but never
appears in an any executable statement.
*program name = matinit
*program name = sl_init
*end of compilation : example1
*end of compilation : matinit
*end of compilation : sl_init
*program  units  =  0003,  0001    diagnostics
generated, highest severity code is  00

% cat job.csh                                            … ジョブスクリプト例
#!/bin/csh
#@$-q personal
#@$-N 1
#@$-J SS
cd  {a.out のあるディレクトリーを指定}
mpirun ./a.out                                            実行は mpirun コマンドを使用

% qsub job.csh                                          … ジョブを投入
Request xxxxx.n81 submitted to queue: personal.

～ バッチジョブ終了後 ～

% cat job.csh.oxxxxx                                    … 結果を表示

ScaLAPACK Example Program #1 -- May 1, 1997

Solving Ax=b where A is a   9 by   9 matrix with
a block size of   2
Running on   6 processes, where the process grid
is   2 by   3

INFO code returned by PDGESV =   0

According to the normalized residual the solution
is correct.

||A*x   -   b||   /   (   ||x||*||A||*eps*N   )   =
0.00000000E+00
```

＊フリーソフトウェア ATLAS について

SR11000 上にフリーソフトウェアの ATLAS（Automatically Tuned Linear Algebra Software）を公開しています。これは、BLAS ライブラリーおよび LAPACK ライブラリーとして利用できますが、キャッシュを効率的に利用するなど高速化が図られています。詳細につきましては、下記のウェブページをご覧ください。なお、インストールされているバージョンは 3.6.0 です。

http://math-atlas.sourceforge.net/

・ATLAS のインストール先

/usr/local/unsupported/atlas/include/	C 言語用ヘッダファイル
/usr/local/unsupported/atlas/lib/	ライブラリー本体

32bit スカラー版のみの提供となっており、64bit 版及び要素並列版はありません。また、C 言語用の BLAS インターフェースである CBLAS を利用する場合には、プログラム中で下記のように cblas.h ファイルをインクルードしてください。

```
#include "/usr/local/unsupported/atlas/include/cblas.h"
```

本ライブラリーを利用する場合には、コンパイル時にオプションとして以下のように指定してください。

・Fortran での利用の仕方

（BLAS のみを利用）

```
-L/usr/local/unsupported/atlas/lib -lf77blas -latlas
```

（LAPACK を利用）

```
-L/usr/local/unsupported/atlas/lib -llapack -lcblas -lf77blas -latlas
```

・C 言語での利用の仕方

（CBLAS のみを利用）

```
-L/usr/local/unsupported/atlas/lib -lcblas -latlas
```

（LAPACK を利用）

```
-L/usr/local/unsupported/atlas/lib -llapack -lcblas -lf77blas -latlas
```

本ライブラリーの利用に関しての保証・サポートは行っておりません。使用方法、性能、障害等に関する質問等については一切回答できませんので、予めご了承ください。また、予告なくバージョンアップや不具合修正などの更新を行うことがあります。最新の情報に関しては、必ずセンターのウェブページにてご確認ください。

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，表現形式が異なるため，変換の際に誤差が生じる場合があるので御注意下さい。PowerPC(SR8000)形式の単精度，倍精度浮動小数点形式は IEEE 形式に準拠しています。

精度	PowerPC 形式	M 形式
単精度	$\pm 1.175495\times10^{-38}$～ $\pm 3.402823\times10^{38}$	$\pm 5.397606\times10^{-79}$～ $\pm 7.237005\times10^{75}$
倍精度	$\pm 2.225074\times10^{-308}$～ $\pm 1.797693\times10^{308}$	
拡張精度	$\pm 2.225074\times10^{-308}$～ $\pm 1.797693\times10^{308}$	

M 形式は運用支援システム（VOS3，m-unix）で標準のデーター形式です。

例えばワークステーション等で作成した IEEE 形式（ビッグエンディアンのみ）の書式なしファイルは SR11000 および SR8000/MPP システムでそのまま読み込めますが, VOS3（ファイル形式の変換が必要)や m-unix で作成した M 形式の書式なしファイルを SR11000 および SR8000/MPP システムで読み込む場合はデーター形式の変換が必要ですので実行時オプションを指定して

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

とします。実行時オプションについては「2.3. 実行時オプション」を参照して下さい。

5.2. ファイル形式

UNIX システムの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とがあり，標準書式なしファイルはさらにファイル位置付け動作の異なる 2 種類（ファイル形式は同じ）に分けられます。

書式なし	標準書式なしファイル	業界標準書式なし（デフォルト）
		標準書式なし
	FORTRAN 固有ファイル（m-unix のデフォルト）	
書式付き	標準テキストファイル	

標準書式なし（stduf）と業界標準書式なし（dstduf）は同じファイル形式ですが，ファイル終了記録検出後の BACKSPACE 文の動作が以下のように異なります。

`stduf` ファイル終了記録検出後の BACKSPACE 文の実行によって，ファイルポインターがファイル終了記録の直前のデーターの先頭に位置付けられます。

`dstduf` ファイル終了記録検出後の BACKSPACE 文の実行によって，ファイルポインターがファイル終了記録の直前に位置付けられます。

SR11000 および SR8000/MPP システムのデフォルトの形式は上記の表の業界標準書式なしファイルです。

標準書式なしファイルとして入出力を行う場合は

```
%  ./a.out -F'port(stduf)'
```

FORTRAN 固有ファイルの入出力を行う場合は

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

とします。ただし，FORTRAN 固有ファイルの入力についてはファイル形式を自動的に判別して入力するのでオプションを省略できます。(データー形式の変換は必要に応じてオプションを指定して下さい。)

☞ VOS3 で作成した書式なしファイル形式は VOS3 独自の形式なので（業界）標準書式なしファイルか，FORTRAN 固有ファイルに VOS3 上で変換する必要があります。

```
>> FCONVERT 順番探査ファイル,標準書式なしファイル,PS2STDUX
>> FCONVERT 順番探査ファイル,FORTRAN 固有ファイル,PS2HITUX
>> FCONVERT 直接探査ファイル,標準書式なしファイル,DA2STDUX
>> FCONVERT 直接探査ファイル,FORTRAN 固有ファイル,DA2HITUX
```

変換したファイルはバイナリー（binary）形式で FTP 転送して下さい。

5.3. ファイル接続

装置番号を使用して，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。

標準入出力

装置番号 5 が標準入力（キーボード)，装置番号 6 が標準出力（画面）に接続されていますので，例のような端末に対する入出力を行うことができます。

```
端末から入力，端末へ出力する例
% cat program.f
       read(5,*) a,b
       write(6,*) a*b,a/b
       stop
       end
% f90 program.f
（メッセージは省略）
% ./a.out
2 3                                          …  端末（キーボード）から入力。
   6.00000000       0.666666687              …  端末の画面に表示される。
```

標準入出力はシェルの機能を利用してデーターをファイルから入力したり，結果をファイルに出力したりすることができます。

```
ファイル a.data から入力し，標準出力へ出力する例
% cat a.data
2 3
% ./a.out < a.data                           …  a.data から入力。
   6.00000000       0.666666687

ファイル a.data から入力し，ファイル a.result に出力する例
% cat a.data
2 3
% ./a.out < a.data > a.result                …  a.data から入力，  a.result に出力。
% cat a.result
   6.00000000       0.666666687              …  結果
```

OPEN 文

装置番号 n と指定した既存のファイルを接続,または指定したファイル名で新規にファイルを生成して接続することができます。

```
装置番号 10 を既存のファイル data に書式なし入力文として接続する例
      open(10,file=,'data',stauts='old',form='unformatted')
      read(10) a,b
      write(6,*) a*b,a/b
      close(10)
      stop
      end
装置番号 11 を新規に生成したファイル result に書式付き出力文として接続する例
      open(11,file=,'result',stauts='new',form='formatted')
      read(5,*) a,b
      write(11,100) a*b,a/b
  100 format(1h ,2d12.4)
      close(11)
      stop
      end
```

環境変数 FT*nn*F*xxx*

OPEN 文を使用すると上記の例のようにプログラム中で指定したファイルと接続できますが，OPEN 文を使用しない場合は環境変数 FT*nn*F*xxx* を使用してファイル名を指定できます。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，プロブ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なりません。設定方法は以下のとおりです。

使用例（sr11000-s）

```
% cat program.f
      read(10,*) a,b
      write(11,100) a*b,a/b
  100 format(1h ,2d12.4)
      stop
      end
% f90 program.f
 （メッセージは省略）
% cat data
2 3
% setenv FT10F001 data                  …  環境変数の設定（入力ファイル data）
% setenv FT11F001 result                …  環境変数の設定（出力ファイル result）
% ./a.out
% cat result
   0.6000E+01  0.6667E+00

 （NQS スクリプトの場合）
#!/bin/csh
#@$-q single
#@$-lT 00:30:00
#@$-lM 1GB
cd work
setenv FT10F001 data
setenv FT11F001 result
./a.out
```

環境変数 FT*nn*F*xxx* による書式なし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。（ファイル形式は実行時オプションで変換して下さい。）

ホスト名	PowerPC 形式	M 形式
mpp-s/bt sr11000-s, batch	FT*nn*F*xxx* （または FT*nn*S*xxx*）	FT*nn*M*xxx*
m-unix	なし	FT*nn*F*xxx*

例えば SR11000（sr11000-s, batch）で環境変数 FT*nn*M*xxx* を設定した場合，ファイルには M 形式で書き出されます。以下にデーター形式を変換する例を挙げます。なお，データー形式については「5.1. データー形式」を参照して下さい。

```
（SR11000 および SR8000/MPP の場合）
setenv FT10F001 data1                    …  PowerPC 形式で入出力
setenv FT12M001 data3                    …  M 形式で入出力
（m-unix の場合）
setenv FT10F001 data3                    …  M 形式で入出力
```

特定ファイル名 ft.*nn*

OPEN 文も環境変数も設定しない場合には，READ，WRITE 文は特定ファイル名 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。既に存在する場合は，先頭から書き込まれます。

使用例（sr11000-s）

```
標準入力（端末）から入力，ファイル ft.11 へ出力する例
% cat a.f
       read(5,*) a,b
       write(11,*) a*b,a/b
       stop
       end
% f90 a.f
（メッセージは省略）
% ./a.out
2 3
% cat ft.11
   6.00000000       0.666666687                 …  ファイルがなければ作成される。

ファイル ft.10 から入力，ファイル ft.11 へ出力する例
（ファイル ft.10 が存在しないとエラーになる）
% cat a.f
      read(10,*) a,b
      write(11,*) a*b,a/b
      stop
      end
% f90 a.f
（メッセージは省略）
% ./a.out
KCHF348R the read statement for unit id 10 is invalid. the file does not exist.
file name is ft.10.

（ファイル ft.10 を作成し，ft.10 から入力，ファイル ft.11 へ出力する）
% vi ft.10                                   …  入力ファイル
2 3
% ./a.out
% cat ft.11
   6.00000000       0.666666687
```