UNIX システム利用入門 ― SR8000 利用ガイド ―

システム運用掛　宮 嵜　　洋
佐々木 一孝

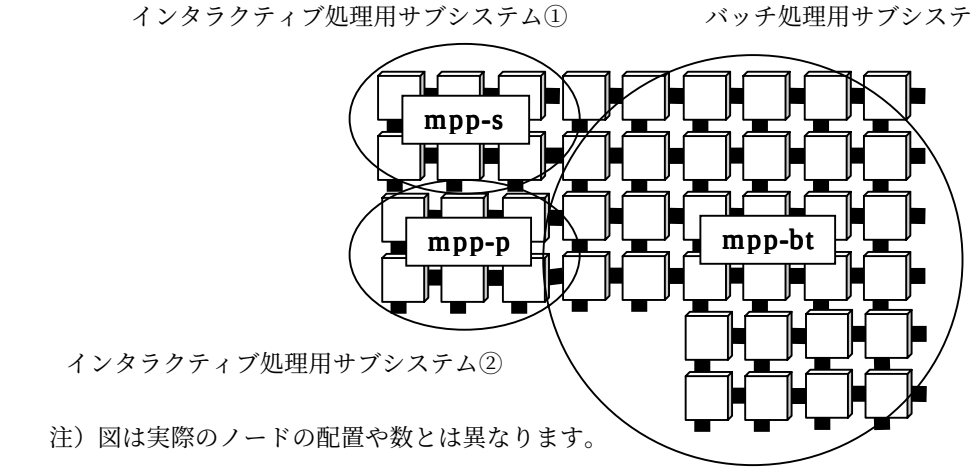
1. SR8000 システム概要

1.1. ノードとサブシステム

本センターでは1999年3月よりSR8000/128(2001年4月からは負担金体系を変更し、バルク利用システムとして運用)、2001 年 4 月より SR8000/MPP の 2 台の並列型スーパーコンピューター（以下 SR8000 システムとします）を運用しています。これらのシステムは1ノード当り8台の演算プロセッサーを備えた演算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。

	SR8000/MPP	SR8000/128（バルク利用）
演算ノード数	144	128
1ノード当りの理論性能	14.4GFLOPS	8GFLOPS
1ノード当りの主記憶容量	16GB	8GB
演算プロセッサー数	1152	1024
1プロセッサー当りの理論性能	1.8GFLOPS	1GFLOPS

各演算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、通常はシステムを以下の図のように3つのサブシステムに論理分割して運用しています。 サブシステムはそれぞれ 1 台の UNIX システムとして機能します。(SR8000/MPP では月に一度､バッチ処理用のサブシステムを拡大して 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。)



注）図は実際のノードの配置や数とは異なります。

本稿では主に最適化 FORTRAN77 を使用したコンパイル、実行例を紹介します。

各サブシステムは以下のように使い分けます。

- インタラクティブ処理（TSS）用サブシステム

  インタラクティブ処理用サブシステムはログインしてコマンド実行するときに使用します。 以下に示す 2 つのサブシステムを用意しています。

  - スカラー処理サブシステム

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

    bulk-s.cc.u-tokyo.ac.jp（SR8000/128）

    主にプログラムの編集、 コンパイル、 バッチジョブの投入に使用します。スカラージョブをインタラクティブに実行できます。

  - 並列処理サブシステム

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

    bulk-p.cc.u-tokyo.ac.jp（SR8000/128）

    主にテストジョブ実行、デバッグに使用します。要素並列ジョブをインタラクティブに実行できます。また、2 ノードまでのノード並列実行を可能とします。

- バッチ処理用サブシステム

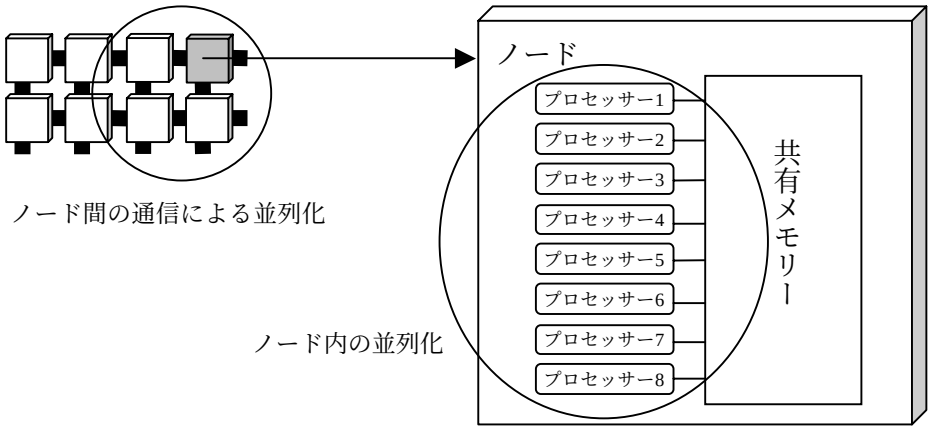
  mpp-bt.cc.u-tokyo.ac.jp（ログイン不可）

  bulk-bt.cc.u-tokyo.ac.jp（ログイン不可）

  バッチ処理用サブシステムはバッチジョブ実行専用のサブシステムです。このサブシステムにログインすることはできません。ジョブのタイプに合わせて各種ジョブクラス（キュー）を用意しており、ジョブはインタラクティブ処理用サブシステムから NQS（Network Queuing System）により投入します。

1.2. 要素並列処理

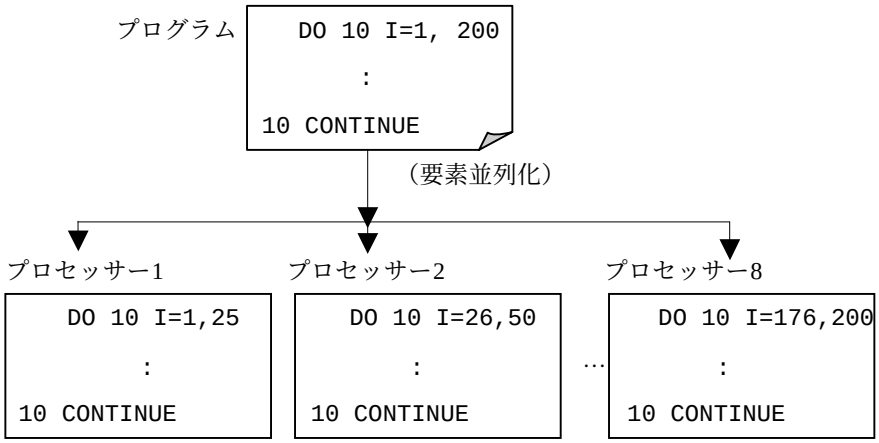
SR8000 システムは複数の演算ノードを搭載した分散メモリー型の並列計算機ですが、各ノードは 8 台のマイクロプロセッサーと共有メモリーで構成されており、ノード単体でも並列実行やスカラージョブの多重処理が可能です。



各演算ノードは、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。

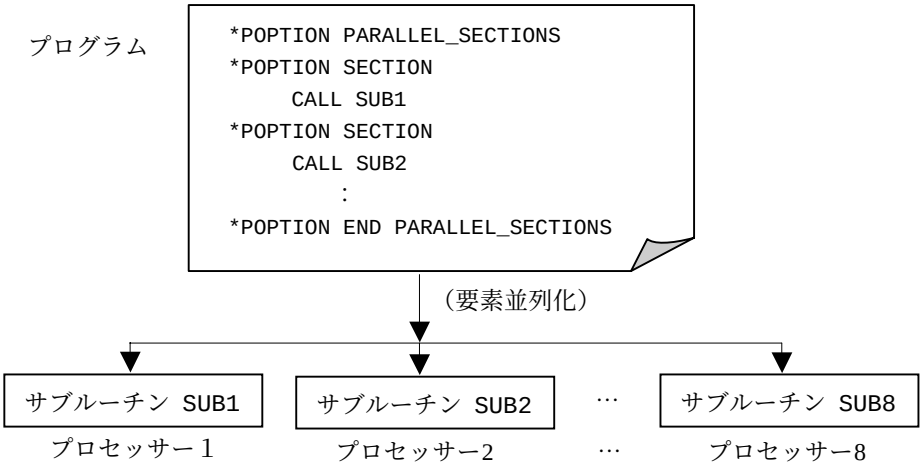
- DO 型要素並列化（ループ要素並列化）

  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。

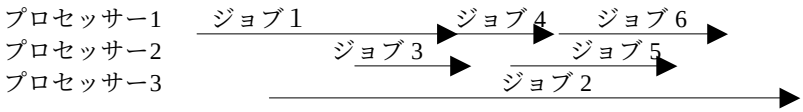


1.3. ジョブの形態

SR8000 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 MPI や PVM、PARALLELWARE などの通信ライブラリー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。

a. スカラージョブ

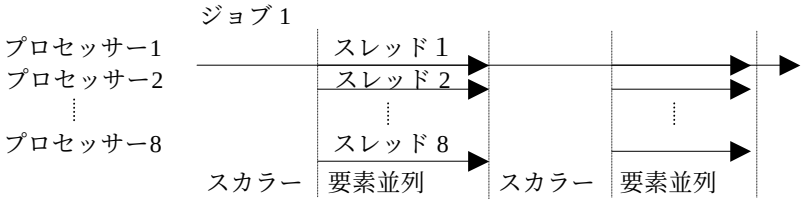
スカラープログラムを実行するジョブです。1 プロセッサーのみ使用するので他のジョブとノードを共有し、ノード内で多重実行します。



☞ このジョブはスカラージョブクラス（A～F）または mpp-s、bulk-s で実行します。

b. 要素並列ジョブ

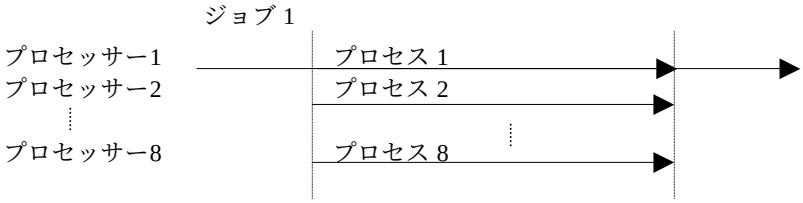
要素並列プログラムを実行するジョブです。要素並列化により分割されたスレッドが各プロセッサーに配置され並列に実行します。



☞ このジョブは並列ジョブクラス（P001～P016、バルク専用キュー）または mpp-p、bulk-p で実行します。ES ジョブクラス（A-ES～F-ES）でも実行できます。

c. ノード内 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。）



☞ このジョブは並列ジョブクラス（P001～P016、バルク専用キュー）または mpp-p、bulk-p で実行します。また、オプションにジョブタイプ SS（#@$-J SS）の指定が必要です。

1.4. 利用負担金

SR8000/MPP 利用については「1.4.1.基本負担金コース」に示す単価に従って利用者ごとに課金します。バルク利用システム SR8000/128 は定額負担金方式のため、基本負担金コースの利用負担金は適用されませんので「1.4.2. バルクコース」を御覧下さい。なお、利用負担金については変更することがありますので必ず最新の内容を御確認下さい。

1.4.1. 基本負担金コース（2001 年 4 月 2 日現在）

CPU 課金

利用者が使用する CPU 時間／月 （または CPU 時間／月に相当する値） について以下の単価を適用します。

月額 1,000 円コース	月額 2,000 円コース	単　価
～　150 時間／月	～　450 時間／月	0　円／秒
150 時間超　～　6,000 時間／月	450 時間超　～　5,000 時間／月	0.02　円／秒
6,000 時間超／月 ～	5,000 時間超／月 ～	0　円／秒

☞ 計算方法（1,000 円定額コースで 500 時間使用した場合）
150 時間×3600×0 円＋(500 時間－150 時間)×3600×0.02 円=25,200 円

CPU 時間の計算方法はログインセッションやバッチジョブがノードを「共有」するタイプか、「占有」するタイプかによって異なります。（例えば要素並列ジョブはノードを占有するタイプです。）

「共有」の場合

ジョブクラス　A　～　F　でバッチジョブを実行する場合と　mpp-s　にログインしたときが対象です。複数のスカラージョブが 1 ノードを共有し、他のジョブと同じノードで動作します。（TSS ではログインしているノードに他の利用者が同時にログインできます。）このため、実際に使用した CPU 時間に対して課金します。

☞ mpp-s に 1 時間ログインしたとき、プログラムやコマンドの実行による実際使用した CPU 時間が 30 秒だった場合の課金は、
30 秒×1×0.02 円 = 0.6 円（切り上げで 1 円）

「占有」の場合

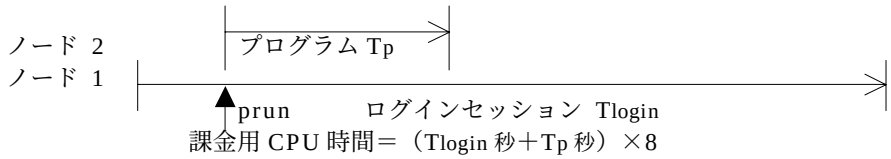
ジョブクラス P001　～　P016　、　A-ES　～　F-ES　でバッチジョブを実行する場合と　mpp-p　にログインしたときが対象です。1 利用者がノード（8 プロセッサー）を占有するため、そのジョブが動作している間、他の利用者はそのノードを使用することができません。（TSS ではログインしているノードに他の利用者が同時にログインすることはできません。）このため、課金はノード当たりの占有時間（経過時間）の 8 倍を CPU 時間とみなして単価を適用します。なお、要素並列ジョブ、複数のノードを使用する並列ジョブはノードを占有して動作します。

☞ mpp-p に 1 時間ログインした場合には、プログラムやコマンドの実行で使用した CPU 時間の多少に係わらず
3600 秒×8×0.02 円= 576 円

ログインして対話的に実行するインタラクティブジョブとバッチジョブでは課金方法が異なります。

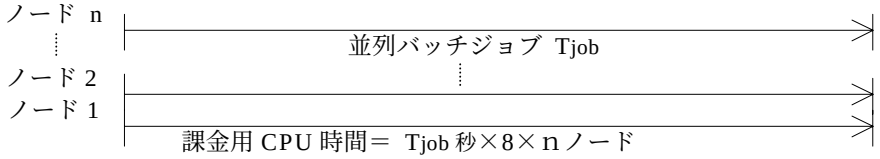
インタラクティブジョブ（mpp-p）

ログインノードについてはセッション時間（＝ノードの占有時間)の 8 倍を CPU 時間とみなして課金します。ここから新たなノードを確保し、プログラムを実行（prun, mpirun 等の並列実行コマンド）した場合は確保したノードの占有時間合計の 8 倍を CPU 時間とみなして別途追加分を課金します。



バッチジョブ

バッチジョブの実行中は利用者が指定した数のノードを確保しますのでジョブの実行時間 （＝ノード当たりの占有時間） の 8 倍をさらにノード数倍したものを CPU 時間とみなして課金します。



ファイル課金

長期保存ファイル　/home

利用者が指定したファイル量の上限値まで使用しているものとして以下の単価を適用します。なお、 上限値変更の手続きは「newuser」により行います。

月額 1,000 円コース	月額 2,000 円コース	単　価
～　200MB	～　400MB	0　円／ (MB　・月)
200MB 超　～　2,560MB	400MB 超 ～　2,048MB	10　円／ (MB　・月)
2,560MB 超　～	2,048MB 超　～	0　円／ (MB　・月)

☞ 計算方法（1,000 円定額コースで 300MB を上限値とした場合）
200MB×0 円＋(300MB－200MB)×10 円=1,000 円／月

短期保存ファイル　/short

利用者が使用しているファイル量に対して以下の単価を適用します。

5 円／(月・MB)

ファイル課金は長期保存ファイル /home が毎月の月始めに１ヶ月分（newuser による利用登録または上限値増加の手続きをした場合は日割り計算）を課金するのに対し、短期保存ファイルはファイルシステム毎に実際のファイル保存量から上記単価で 1 日分を算出（切り上げ）し、毎日課金します。

1.4.2. バルクコース（2001 年 4 月 2 日現在）

定額負担金コース（バルクコース）はプロジェクト（研究室のメンバー等で構成する利用者グループ）単位で一定のシステム資源を一括して利用するという申請形態です。申請したノード数に従って利用登録番号とディスク容量、専用キューがプロジェクトに割り当てられます。現在の設定では SR8000/128 のシステム資源を以下のように割り当てます。

システム資源の割り当て		負担金額
演算装置 利用登録番号 ディスク容量	専用キューとして申請ノード数まで利用可 10 個／ノードまで 20GB／ノードまで	96 万円／ (年 ・ノード)
オプション	利用者番号 1 個につき	24,000 円／年
	ディスク容量 10GB につき	96,000 円／年

☞ 利用期間は原則として 4 月から年度末までの 1 年間ですが、年度途中月からの利用も可能です。年度途中月からの負担金額はスーパーコンピューティングニュースにて御確認下さい。2001 年度については「計算機利用負担金の改正について」（Vol.3 No.2, 2001.3）を御覧下さい。

システム資源の制限はグループ制限値によって管理されますので各個人への資源割り当てはグループ内で調整して下さい。なお、グループ管理者は各利用者に対して CPU 時間、ファイル容量等の制限を行うことができます。（プロジェクト代表者はグループ管理に関する資料を本センターより入手することができます。）

バルクコースでの SR8000/128 利用方法はプロジェクト毎に専用キューが用意されていることを除けば SR8000/MPP とほぼ共通（同様のジョブ実行環境を共用キューとして用意しています。ただし、ジョブクラス制限値は異なりますので `show-info` コマンドで確認して下さい。）ですが、バルクコースで利用できるシステムは SR8000/128 に限られ、基本負担金コースのサービスは利用できませんので御注意下さい。なお、専用キューの属性は以下の通りです。

専用キュー名	プロジェクトコード名
制限時間（CTIME）	なし（CPU 時間制限）
制限時間（ETIME）	24 時間（経過時間制限）
メモリーの大きさ	6.6GB（1 ノード当りの最大値）
ノード数	ノード利用申請時の「希望ノード数」
ジョブ属性	要素並列プログラムの実行が可能です。ただしノードを占有するため、同時に実行可能なジョブは 1 ノードにつき 1 ジョブです。

☞ プロジェクトコード名が a00 の場合、専用キュー名は「a00」となります。専用キューを使用するときはジョブスクリプトに

```
#@$-q  a00
```

と記述します。なお、専用キューの構成や制限時間（CTIME、ETIME）、メモリーの大きさ等は利用状況に応じて変更する場合があります。

2. コンパイルとオプション

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 f77（最適化 FORTRAN77）又は f90（最適化 FORTRAN90）コマンドを使用します。また、リンクにも f77、f90 コマンドを使用します。

```
f77  または f90
    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していきますので同一又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、本稿では －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ご注意下さい。（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。）

f77:　–i,P　–opt=s　–pvec　–noparallel　–symnchk

f90:　–i,P

デフォルトオプションは環境変数 F77OPTS（又は F90OPTS）で利用者ごと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  F77OPTS  " "                    デフォルトオプションを設定しない
% setenv  F77OPTS  "-pvec  -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 又は .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

```
% f77  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
f77: compile start : main.f

*OFORT77 V01-01-A 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:
f77: compile start : sub.f

*OFORT77 V01-01-A 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

```
% f77  -c  main.f   sub.f                      main.f, sub.f  をコンパイルしてオブジェク
（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
%
% f77 -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 FORTRAN77（または最適化 FORTRAN90）で使用される主なオプションとその機能について記述します。

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ない、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ごと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。

最適化レベル

レベル０	原始プログラム通りのコンパイル、文の実行順序に影響を与え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）
レベル３	演算順序の変更はせず、プログラム全体での大域的な最適化 （分岐命令の最適化、命令の実行順序変更、演算子の変更、不変式のループ外への移動、ユーザー関数のインライン展開等）
レベル４	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 （演算順序の変更、除算の乗算化、ループ展開・分配・融合等）
レベル S	実行速度が最も速くなるようなコンパイルオプションの自動設定 （擬似ベクトル化、要素並列化を含む各種最適化オプションの設定）
レベル SS	演算精度を落としても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 （レベル S に加えてライブラリーコード使用、引数チェックの抑止等）

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。

–opt=最適化レベル ***最適化オプション***

```
% f77  program.f                    最適化レベル S でコンパイル
% f77  -opt=3   program.f           最適化レベル３でコンパイル
% f77  -opt=ss  program.f           最適化レベル SS でコンパイル
```

オプション無指定時の最適化レベルは f77 が S（ただし要素並列化を含まない）、f90 が３。

–loopdiag ***最適化ループ診断メッセージ***

```
% f77  -loopdiag  program.f          program.f  をコンパイルしたときのループ
f77: compile start : program.f      構造診断メッセージを出力する
                                    （f77 はデフォルトで最適化レベル S）
*OFORT77 V01-01-A entered.
*program name = MAIN
*end of compilation : MAIN
  KCHF1805C                         （DO10 ループと後続のループを融合）
      the do 10 loop is fuzed with the
following one. line=3
  KCHF1809C                         （DO10 ループを 2 回展開）
      the do 10 loop is unrolled 2 times.
line=3
*program units = 0001, no diagnostics
generated.
```

オプションの詳細及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 FORTRAN77 使用の手引（6A30-3-314）」又は「最適化 FORTRAN90 使用の手引（6A30-3-311）」を御覧下さい。

主な最適化オプションを以下に示します。（–opt=n は最適化レベルｎに含まれる最適化機能であることを意味します。）

–expmove

条件式下にあるループ不変式をループ外に移動する。（–opt=3）

–loopdistribute

ループ分配による最適化をする。（–opt=4）

–approx

除算の逆数による乗算化を許可する。（–opt=s）

–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。(–opt=s)

–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。(–opt=s)

–loopexpand

ループ展開による最適化をする。(–opt=s)

–loopfuse

ループ融合による最適化をする。(–opt=s)

–predicate

IF 文に対して分岐命令を生成しないようにする最適化を行う。(–opt=ss)

演算順序が変更となる最適化(–expmove、–divmove、–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、プレディケート最適化(–predicate)はセグメンテーション例外の発生や性能劣化を引き起こす可能性があります。これらを回避するためには–no*option*(例えば–noapprox、–nopredicate)を指定して原因となる最適化機能を抑止して下さい。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詳細はマニュアル「最適化 FORTRAN77 使用の手引(6A30-3-314)」又は「最適化 FORTRAN90 使用の手引(6A30-3-311)」を参照して下さい。

擬似ベクトル化オプション

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処理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)を隠蔽できるので命令間の依存による性能低下を抑えることができます。擬似ベクトル化が適用される主な条件は以下の通りです。

- 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、計算型GOTO文、SELECT文
  - 関数、手続き呼び出し(擬似ベクトル化数学関数を除く)
  - 入出力文
  - 終了・停止文(STOP文、PAUSE文、RETURN文など)
- DOループ内でIF文による分岐命令を生成しないプレディケート最適化が適用できる
- DOループ内に別名参照(EQUIVALENCE等)を含まない
- DOループの実行性能向上が見込める

–pvec ***擬似ベクトル化オプション***

`% f77 program.f`	擬似ベクトル化する（デフォルト）
`% f77 -pvec program.f`	擬似ベクトル化する
`% f77 -nopvec program.f`	擬似ベクトル化を抑止する

オプション無指定時でもf77では擬似ベクトル化オプションが有効（デフォルト）、f90では指定が必要。

–pvdiag ***擬似ベクトル化診断メッセージ***

```
% f77 -pvdiag program.f                    program.f を擬似ベクトル化してコンパイ
f77: compile start : program.f             ルしたときの診断メッセージを出力する
                                           （f77 はデフォルトで擬似ベクトル化する）
*OFORT77 V01-01-A entered.
*program name = MAIN
*end of compilation : MAIN
   KCHF1700C
       the do 20 loop is pseudo-vectorized.   （DO20 ループを擬似ベクトル化）
line=7
*program units = 0001, no diagnostics
generated.
%
```

オプションの詳細及び個別の擬似ベクトル化オプションにつきましてはマニュアル「最適化 FORTRAN77 使用の手引（6A30-3-314）」又は「最適化 FORTRAN90 使用の手引（6A30-3-311）」を御覧下さい。

擬似ベクトル化は最適化レベル S（–opt=s）に含まれます。本センターの設定では f77 はデフォルトに含まれていますが、f90 の場合は指定（–pvec 又は–opt=s 以上）が必要です。

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。以下に示す要素並列化レベルが設定されています。

要素並列化レベル

レベル 0	要素並列化をしない
レベル 1	SECTION 型要素並列化、強制 DO 型要素並列化
レベル 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
レベル 3	ループ分配、ループ分割、ループの一重化
レベル 4	パイプライン要素並列化、圧縮・拡張ループの要素並列化

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。

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DO型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。DO型要素並列化が適用される主な条件は以下の通りです。

- 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又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

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***

% f77 -parallel program.f	要素並列化（レベル２）する
% f77 -parallel=4 program.f	要素並列化（レベル４）する
% f77 -noparallel program.f	要素並列化を抑止
% f77 -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レベル2（–parallel=2）を仮定。
–parallel リンケージオプションにはレベル指定は不要。

–pardiag ***要素並列化診断メッセージ***

% f77 -parallel -pardiag program.f **f77: compile start : program.f**	program.f を要素並列化してコンパイルしたときの診断メッセージを出力する
***OFORT77 V01-01-A entered.** ***program name = MAIN** ***end of compilation : MAIN** ***end of compilation : _parallel_func_1_MAIN** **(diagnosis for loop structure)**	
KCHF2000C **the do 10 loop is parallelized.line=3**	（DO10 ループを要素並列化）
KCHF2011C **the variable(s) or array(s) in do 10 loop is applied to tlocal transformation.name=I line=3**	（DO10 ループ内の変数を TLOCAL 化）
***program units = 0001, no diagnostics generated.** **%**	

オプションの詳細及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 FORTRAN77 使用の手引（6A30-3-314）」又は「最適化 FORTRAN90 使用の手引（6A30-3-311）」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占有できる並列ジョブクラス（P001～P016、バルク専用キュー）や ES ジョブクラス（A-ES～F-ES）でプログラムを実行する必要があります。また、対話処理なら mpp-p、bulk-p を使用します。

OpenMP

最適化 FORTRAN90 では OpenMP（OpenMP Fortran Application Program Interface）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)
!$omp do                                  OMP 指示文の例
      do i=1,1000
       a(i)=i
      enddo
!$omp enddo
      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（6A30-3-311）」を御覧下さい。

OpenMP は 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、エラーではなく 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 を超える場合には、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ます。また、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***

```
% f77  -64  program.f                     64 ビットモードでコンパイルする
% f77  -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
% f77       program.f                     32 ビットモードでコンパイルする
% f77  -32  program.f                     32 ビットモードでコンパイルする

% f77  program.f                          32 ビットモード
f77: compile start : program.f

*OFORT77 V01-03-/A entered.
*program name = MAIN

   KCHF640C  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 units = 0001, 0001   diagnostics
generated, highest severity code is  16
% f77 -64 program.f                       64 ビットモード
f77: compile start : program.f

*OFORT77 V01-03-/A entered.
*program name = MAIN
*end of compilation : MAIN
*program  units  =  0001,  no  diagnostics
generated.
%                                         コンパイル終了
```

mpp-p/bulk-p では unlimit コマンドでメモリー空間を拡張してから実行して下さい。（mpp-s/bulk-s は実行不可）
バッチジョブの場合はメモリーの大きさ（#@$-lM）を 2GB 以上に設定して下さい。

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。バッチジョブならば#@$-lM、インタラクティブ処理では limit コマンド（又は unlimit コマンド）で設定します。以下は–64 オプション指定時の注意事項です。

- 定数（名前付き定数）及び DATA 文等により配列に初期値を与える場合（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 GOTO 文、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なうか、オプション–intexp=full で全ての整数型 4 バイトの変数、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

なお、2GB を超えるファイルサイズの入出力は非同期入出力文を除いて 32 ビットアドレッシングモードでも可能です。

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。要素並列化や擬似ベクトル化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***

```
% f77  -parallel  -loglist   program.f        ログメッセージファイルを出力する
f77: compile start : program.f
                                               ログメッセージファイルはソースプロ
*OFORT77 V01-01-A entered.                     グラム毎に
*program name = MAIN                             「ソースプログラム名.log 」
*end of compilation : MAIN                     というファイル名で出力される
*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 の内容
       parameter (n=1000)
       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）
** --- loops merged(DO10 and DO20)---          （DO10、DO20 ループ融合）
** Parallel processing finishing at loop exit
**
**    [DO 10]
**    Innermost loop unrolled (2 times).       （最内側ループ展開）
XX  there  is  no  target  for  pattern        （擬似ベクトル化適用せず）
referencing,PVP not applied.
**
（略）
```

ログメッセージは日本語で出力することができます。（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す。）このときコンパイルメッセージも日本語になります。

–listlang=*文字コード種別*

```
% f77  -parallel  -loglist  program.f
       -listlang=sjis
f77: compile start : program.f

*OFORT77 V01-01-A 開始
*プログラム名 = MAIN
*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
       parameter (n=1000)
       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
** --- ループ融合を行った(DO10 と DO20)---
** ループ出口で並列処理 終了
**
**    [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。
**
```

ログメッセージファイルを出力する
（オプション –listlang=sjis　を指定）

コンパイルメッセージが日本語で表示される
（端末の文字コードを SJIS に設定すること）

ログメッセージファイル program.log が
日本語で作成される

文字コード種別の対応

文字コード種別	言語（文字コード）
C	英語（ASCII）
SJIS	日本語（SJIS）
EUC	日本語（EUC）

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。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。

–pmfunc*　　*性能モニター情報（関数、手続き）出力オプション

```
% f77 -parallel  program.f  -pmfunc
f77: compile start : program.f

*OFORT77 V01-01-A 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
*program units = 0002, no diagnostics generated.
% a.out
  3.00000000       3000.00000
% ls
a.out
program.f
pm_a.out_Aug22_1625_node000_149513
```

性能モニター情報を出力する実行ファイルを作成

性能モニター情報ファイルは実行ファイル実行後に
「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る。

このファイルはバイナリー形式のため、
性能モニター情報出力コマンド pmpr
を使用してテキスト形式で表示する。

実行ファイル a.out を実行する

性能モニター情報ファイルが出来ていることを確認

```
% pmpr  pm_a.out_Aug22_1625_node000_149513
pm_a.out_Aug22_1625_node000_149513:
（略）
##########################################
##  Function/Subroutine                 ##
##########################################
======================================
==  Function/Subroutine  Ranking      ==
======================================
    CPU-Time[%]    Times  Func(File+Line)
--------------------------------------
[1] 0.004126[93.08]  1   MAIN(program.f+3)
[2] 0.000011[ 0.25]  1   SUB(program.f+14)
---------------------
TOTAL  0.004137[93.32]
```

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

性能モニター情報の内容については スーパーコンピューティングニュース Vol.2NO.4(2000.7)「SR8000 チューニング支援機能を利用したプログラムチューニング」を参照して下さい。

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（要素並列化）出力オプション

```
% f77  -parallel  program.f  -pmpar
f77: compile start : program.f

*OFORT77 V01-01-A 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
*program  units  =  0002,  no  diagnostics
generated.
% a.out
  3.00000000       3000.00000
% ls
a.out
program.f
pm_a.out_Aug22_1647_node000_149551.
%  pmpr  pm_a.out_Aug22_1647_node000_149551
（略）
##########################################
##  Element  Parallel  Region
##########################################
======================================
==  Element Parallel Region Ranking   ==
======================================
CPU-Time[%] MFLOPS MIPS Times Func[Rank]
--------------------------------------
[1]0.000006[ 0.03] 173.010 963.322 1 SUB[-]
[2] 0.000005[ 0.03] 392.912 1104.777 1 MAIN[-]
--------------------------------------
TOTAL  0.000011[ 0.06]
```

性能モニター情報を出力する実行ファイルを作成

性能モニター情報ファイルは実行ファイル実行後に
「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る。
このファイルはバイナリー形式のため、性能モニター情報出力コマンド pmpr を使用してテキスト形式で表示する。

実行ファイル a.out を実行する

性能モニター情報ファイルが出来ていることを確認

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

性能モニター情報の内容については スーパーコンピューティングニュース Vol.2NO.4(2000.7)「SR8000 チューニング支援機能を利用したプログラムチューニング」を参照して下さい。

ソース差分部分コンパイル

ソース差分部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 ***ソース差分部分コンパイルオプション***

```
% f77  -diffcomp  program.f                差分コンパイルオプションを指定する
f77: compile start : program.f            通常と同様にコンパイルを開始

*OFORT77 V01-01-A 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
                                          ファイル  a.out が作成される
--プログラム SUB を編集--
% f77  -diffcomp  program.f                プログラム SUB の内容を変更後、改めて差分
f77: compile start : program.f            コンパイルする

*OFORT77 V01-01-A entered.                 プログラム MAIN に変更がないので構文チェ
*program name = MAIN                      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イ
*end of compilation : SUB                 ルを行う
*program  units  =  0002,  no  diagnostics メッセージ「*end of compilation : MAIN」がな
generated.                                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***

```
% f77  -diffcomp  program.f                差分コンパイルオプションを指定する
f77: compile start : program.f            通常と同様にコンパイルを開始

*OFORT77 V01-01-A 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
% rm  program.f                           ファイル  a.out が作成される
% ls                                      試しにソースプログラム program.f を消し
a.out      program.o                      てみる（必要なファイルは消さないこと）

% f77  -recomp  -parallel  program.f       再コンパイルオプションを指定してコンパイ
f77: compile start : program.f            ル（参考のため–parallel オプションも追加）

*OFORT77 V01-01-A entered.                 オブジェクトモジュール内のソースプログラ
*program name = MAIN                      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***

% f77 -limit program.f	制限コンパイルを行う
% f77 -nolimit program.f	制限コンパイルを行わない（デフォルト）

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***

% f77 -debug program.f	デバッグ情報を表示する
% f77 -argchk=2 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=2 –dline –erstmt –agochk –subchk

–lcheck

FORTRAN 規格（JISX3001-1:1994 又は: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=2

関数、サブルーチンの引数の個数と型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。–argchk=1 は個数のみチェックします。

–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。

☞ 上記オプション–argchk、–subchk でチェックする項目に問題があると、実行時に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.
```

エラー発生箇所を特定できないときはトレース情報を出力（-s,TRACE オプション）して下さい。

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***

% f77 -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 0x100004c8 SAMPLE+0xc8 [program.f:18]）

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 SR8000 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。

–nosymnchk

8 文字を超えるシンボル名（変数名、外部手続き名等）を許します。（f77 のみ）

–i,U

外部手続き名称中の_（アンダースコア）及び 31 文字までの外部手続き名称を使用できます。（f77 のみ。通常 –nosymnchk オプションも同時に指定します。）

–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、注釈（！）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

☞　本センターではこのオプションをデフォルトで設定しています。

–i,PL

8 進定数（O'xxxx'）、16 進定数（Z'xxxx'）、16 進定数（X'xxxx'又は'xxxx'X）が使用できます。 f77 の場合は –h8000 オプションも同時に指定します。

–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。

–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、実行モードの切り替えによる言語仕様の拡張を行ないます。

コンパイルオプション

–i,P　–i,PL　–cpp　–conti199　–h8000　–intptr　–fixed=119

実行時オプション（実行時に自動的に設定されます）

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

2.3. 実行時オプション

実行時オプションはロードモジュール（実行ファイル）を実行するときに指定し、実行時環境の変更や他社 FORTRAN との互換性に対応するオプションです。指定方法は以下のように「-F」の後、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、又は 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、又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。

port(realedt)

実数型データー0.0 を E 形、D 形、Q 形、又は 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）

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。

port(prcntl)

画面、又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。

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(sr8000)

書式なし入出力文に対応するデーター形式を SR8000 形式として扱います。 単精度及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(ieee)

書式なし入出力文に対応するデーター形式を SR2201 形式として扱います。 単精度及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。

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、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､OSF/1-MJ)から 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 SR8000 形式から M 形式（VOS3、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 の組み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 FORTRAN77 または最適化 FORTRAN90 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
      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 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                     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 に
      write(*,*) t                    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
      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 に代
      write(*,*) t                    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)                CPU 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 CPU 時間（秒）を変数 t に
      write(*,*) t                    代入
      stop
      end
```

オプションの詳細及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 FORTRAN77 言語（6A30-3-313）」又は「最適化 FORTRAN90 言語（6A30-3-310）」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 -lf77c（FORTRAN90 では-lf90c）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, FTELL, FTELLO64, GETC, GETCWD,
GETFD, GETGID, 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
```

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

オプションの詳細及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 FORTRAN77 言語（6A30-3-313）」又は「最適化 FORTRAN90 言語（6A30-3-310）」を御覧下さい。

3. 並列アプリケーション

分散メモリー型の並列計算機でノード間並列実行する場合に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なう必要があります。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アプリケーションが用意されています。

3.1. 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。また、リモート DMA 転送ライブラリーや Parallel FORTRAN を用いた並列プログラムの実行にも使用します。

使用例

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

% f77  a.f  -o  a.out                        … コンパイル
f77: compile start : a.f

*OFORT77 V01-01-A 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
% ls                                            data.? をノード毎に入力し、結果を
a.f        a.out       data.1        data.2      result.? に出す。
sample.def
% prun  -f  sample.def                       … 実行
% ls                                          prun コマンドを使用（定義ファイル指定）
a.f      data.1      result.1    sample.def
a.out    data.2      result.2                 result.1, result.2 が作成される
```

mpp-p/bulk-p を使用したインタラクティブ実行例です。バッチジョブ（並列ジョブクラス）を使用することにより最大 16 ノードまでの並列化が可能です。

prun 使用における注意点

(1) パーティション名は不要です。並列ジョブの実行に使用するパーティション名は ALL ですが、デフォルト（環境変数 DEFPART）で設定されていますので指定は省略できます。

(2) prun コマンドはオプションで指定した数のノードを確保し、取得したノードでプログラムを実行します。ただし、mpp-p、bulk-p では、ログインしているノードとは別のノードを確保しますので、ノード不足の場合にはエラーとなり実行できません。

定義ファイル記述用の記号

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

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

3.2. リモート DMA 転送ライブラリー

リモート DMA（Direct Memory Access） 転送ライブラリーはリモート DMA 転送機能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。リモート DMA 転送は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通信が可能なハードウェアの機能です。本ライブラリーを使用することでノード間通信性能を最大限に引き出すことができます。

使用例

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

      integer rank, size
      integer node(2)

      call $rgetnod(node)

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

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

% f77 -rdma rdma_sample.f                    … コンパイル
f77: compile start : rdma_sample.f            静的データー自動リモート DMA 機能を使用（プログラム中で *COPTION パラメーターを定義しているときには–rdma オプションは不要）

*OFORT77 V01-01-A entered.
*program name = SAMPLE
*end of compilation : SAMPLE
*program units = 0001, no diagnostics
generated.
% prun -n 2 a.out                           … 実行
node=          0 size=          2             prun コマンドを使用
node=          1 size=          2
```

mpp-p/bulk-p を使用したインタラクティブ実行例です。バッチジョブ（並列ジョブクラス）を使用することにより最大 16 ノードまでの並列化が可能です。

参考マニュアル 「リモート DMA 転送使用の手引 -FORTRAN-」(6A30-3-312)
「リモート DMA 転送使用の手引 -FORTRAN77-」(6A30-3-315)

3.3. 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 SR8000 システムでは MPI-2 をサポートしています。

使用例

```
% 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

% mpif77  mpi_sample.f                      … コンパイル
f77: compile start : mpi_sample.f            MPI コンパイルコマンドを使用
                                             ※MPI 機能で提供されているコマンドを
*OFORT77 V01-01-A entered.                     使用する場合には環境変数 path の
*program name = SAMPLE                        設定が必要
*end of compilation : SAMPLE                   % set path=($path  /usr/mpi/bin)
*program  units  =  0001,  no  diagnostics
generated

% mpirun -n 2 -np 4 a.out                   … 実行
proc=          0 size=          4            MPI 実行コマンドを使用
proc=          2 size=          4            ※MPI は 1 ノード複数プロセスで実行可
proc=          3 size=          4               予め、以下の環境変数の設定が必要
proc=          1 size=          4             % setenv JOBTYPE SS
```

mpp-p/bulk-p を使用したインタラクティブ実行例です。バッチジョブ（並列ジョブクラス）を使用することにより最大 16 ノードまでの並列化が可能です。

参考マニュアル　「MPI・PVM 使用の手引」(6A30-3-026)

MPI プログラムをバッチジョブで実行する場合は以下のようなスクリプトファイルを使用します。（以下の例ではジョブタイプ SS の設定が必要です。）

```
#!/bin/csh
#@$-q parallel
#@$-N 4                                    ジョブクラス P004 を使用
#@$-lT 1:00:00
#@$-lM 2GB
#@$-J SS                                   ジョブタイプ SS を設定
cd work
/usr/mpi/bin/mpirun -n 4 -np 32 a.out      4 ノード 32 プロセスで実行
```

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。

% mpif77 -o program program.f	実行ファイル program を作成する
% mpif90 program.f	f90 でコンパイルする
% mpif77 -64 program.f	64 ビットモードでコンパイルする
% mpif77 -mpilibf7 program.f	8 文字ルーチン名を使用している

MPI プログラムの実行時オプション指定例を以下に示します。

% mpirun -n 1 -np 8 a.out	1 ノード取得し、8 プロセス実行
% mpirun -np 8 a.out	取得できたノード数で 8 プロセス実行
% mpiexec -N 1 -n 8 a.out	1 ノード取得し、8 プロセス実行
（バッチ）**mpirun -n 16 -np 128 a.out**	16 ノード取得し、128 プロセス実行
（バッチ）**mpirun -n 4 a.out**	4 ノード取得し、4 プロセス実行

mpp-p/bulk-p では unlimit コマンドによるメモリー空間の拡張が必要な場合があります。

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 **mpif77** または **mpif90**（C 言語のとき 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、f77 または f90 コンパイラーを起動します。例えば mpif77 コマンドの代わりに f77 コマンドを使用して同等のコンパイルをすることが可能です。
% f77 -i,U -nosymnchk -I/usr/mpi/include -L/usr/mpi/lib -lfmpi -lmpi mpi_sample.f

(2) MPI プログラムの実行には **mpirun**（または mpiexec）コマンドを使用します。これらのコマンドはオプションで指定した数のノードを確保し、取得したノードでプログラムを実行します。ただし、mpp-p、bulk-p では、ログインしているノードとは別のノードを確保しますので、ノード不足の場合には実行できないことがあります。

(3) 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ができます。（ノード内 MPI と呼びます。）実行時にプロセス数、ノード数を設定し、プロセス数がノード数より多いとき、一つのノードに複数プロセスが配置されます。例えば、P016 キュー（16 ノード）を使用すると 128 プロセッサーまでの並列計算が可能となります。

(4) パーティション名は不要です。並列ジョブの実行に使用するパーティション名は ALL ですが、デフォルト（環境変数 DEFPART）で設定されていますので指定は省略できます。

(5) 使用例にジョブタイプ SS（setenv JOBTYPE SS 又は#@$-J SS）という記述がありますが、これはノード内 MPI を実行する上で必要な設定です。MPI がノード内の各プロセッサーにプロセスを割り振るためには SS（スカラープログラムによるノード共有属性）を指示しなくてはなりません。この設定がないとプロセスは時分割処理となり性能が劣化します。また、この設定下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。

(6) 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（errno=126））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、リモート DMA 領域の大きさ（マニュアル「MPI・PVM 使用の手引き(6A30-3-026)」参照）を確認して下さい。なお、静的データー自動リモート DMA オプション（-rdma）は使用できませんので外して下さい。

(7) MPI_Send や MPI_Recv などのブロッキング送受信では送信と受信が一対一に対応していないと受信又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のでプログラム作成の際には注意して下さい。

3.4. 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

```
% cat host.f                                  ホストプログラムの例
      program host
      include 'fpvm3.h'
      integer mytid, tids(4), numt
      call pvmfmytid(mytid)
      call  pvmfspawn('node',  PVMDEFAULT,   （2 ノードを使用する例）
'*',
     &              2, tids, numt)
      write(*,*) 'parent =', mytid
      write(*,*) numt, 'tasks:', tids
      call pvmfexit(into)
      stop
      end
% cat node.f                                  ノードプログラムの例
      program node
      include 'fpvm3.h'
      call pvmfmytid(mytid)
      write(*,*) 'task id=',mytid
      stop
      end

% f77  -i,U  -nosymnchk  host.f               … コンパイル
       -I/usr/pvm3/include                     ホストプログラム host を作成
        -L/usr/pvm3/lib/$PVM_ARCH
         -lfpvm3 -lpvm3  -o host  -parallel    ※PVM 機能で提供されているコマンドを
f77: compile start : host.f                    使用する場合には環境変数の設定が必要
                                               % source /usr/pvm3/pvm_config
*OFORT77 V01-01-A entered.
*program name = HOST                           また､以下のディレクトリーがないときは
*end of compilation : HOST                     作成する
*program  units  =  0001,  no  diagnostics     $HOME/pvm3/bin/$PVM_ARCH
generated.                                     （ここに実行ファイルをコピーする）

% f77  -i,U  -nosymnchk  node.f               … コンパイル
       -I/usr/pvm3/include                     ノードプログラム node を作成
        -L/usr/pvm3/lib/$PVM_ARCH
         -lfpvm3 -lpvm3  -o node  -parallel
f77: compile start : node.f

*OFORT77 V01-01-A entered.
*program name = NODE
*end of compilation : NODE
*program  units  =  0001,  no  diagnostics
generated.

% cp node host $HOME/pvm3/bin/$PVM_ARCH      実行ファイルをコピーする
% pvm -batch start                           PVM デーモンの起動
pvmd still running.
% $HOME/pvm3/bin/$PVM_ARCH/host              … 実行
parent =      267266                           ホストプログラムを実行する（ノードプログ
         2 tasks:      332801      365569      ラムはホストプログラムが生成する）
0           0
% pvm -batch end                             PVM デーモンの終了
pvmd already running.
```

```
% ls /tmp                                        実行結果は /tmp に出力される
pvml.10391                 pvml.10391.t51401    （出力先は環境変数 PVM_LOGDIR の設定で
pvml.10391.t59401                               変更できる）
% cat pvml.10391.t51401
 task id=      332801
```

mpp-p/bulk-p を使用したインタラクティブ実行例です。バッチジョブ（並列ジョブクラス）を使用することにより最大 16 ノードまでの並列化が可能です。

参考マニュアル 「MPI・PVM 使用の手引」(6A30-3-026)

3.5. 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（ホスト-ノードモデル）

```
% cat host.f                                     ホストプログラムの例
      program host

      call kxinit()
      pgid=kxrun(2,'node')                      （2 ノードを使用する例）
      call kxclos(pgid)
      stop
      end
% cat node.f                                     ノードプログラムの例
      program node

      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

% srf77 -kXH  host.f  -o host                    … コンパイル
f77: compile start : host.f                       ホストプログラム host を作成

*OFORT77 V01-02 entered.
*program name = HOST
*end of compilation : HOST
*program  units  =  0001,  no  diagnostics
generated.
% srf77 -kXN node.f -o node                      … コンパイル
f77: compile start : node.f                       ノードプログラム node を作成

*OFORT77 V01-02 entered.
*program name = NODE
*end of compilation : NODE
*program  units  =  0001,  no  diagnostics
generated.
% ls
host      host.f   host.o   node      node.f
node.o
```

```
% host                                          … 実行
Using partition: "ALL"                            ホストプログラムを実行する(ノードプログ
Allocated 2 nodes, origin at 0, process id        ラムはホストプログラムが生成する)
378483.
Loading "node" into all processors ...
Loaded, starting ....
node=           1 size=           2
node=           0 size=           2
%
```

使用例(Cubix モデル)

```
% cat cubix.f                                   Cubix プログラムの例
      program node

      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

% srf77 -kcubix cubix.f -o cubix-node          … コンパイル
f77: compile start : cubix_cbx.f                  Cubix モデルとしてコンパイル

*OFORT77 V01-02 entered.
   KCHF233C  00                TIORET
             the  variable  is  defined  but
never referred.
*program name = CBXMAIN
*end of compilation : CBXMAIN
*program units = 0001, 0001   diagnostics
generated, highest severity code is  00

% cubix -n 2 cubix-node                        … 実行
Cubix Version 3.2.5 -- Copyright (C) 1988-1996    cubix コマンドを使用(2 ノードで実行)
ParaSoft Corp.
All Rights Reserved Copyright (C) 1998,1999,
Hitachi Ltd.
HI-UX/MPP for SR8000 PARALLELWARE 03-00
Using partition: "ALL"
Allocated 2 nodes, origin at 0, process id
378501.
Loading "cubix-node" into all processors ...
Loaded, starting ....
Execution Terminated:
  node=  0  size=  2
  node=  1  size=  2
System 0:1   User 0:0
CUBIX: exit status 0
%
```

mpp-p/bulk-p を使用したインタラクティブ実行例です。バッチジョブ(並列ジョブクラス)を使用することにより最大 16 ノードまでの並列化が可能です。

参考マニュアル　「PARALLELWARE ユーザーズガイド -FORTRAN-」(6A30-3-400)
「PARALLELWARE リファレンス -FORTRAN-」(6A30-3-401)

3.6. 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。プログラム中にコメントで並列化指示文を記述することによって通信ライブラリーを使用したプログラムに変換できます。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 program.f                                  サンプルプログラム
      integer a(100)
!hpf$ processors p(2)                             HPF 指示文（2 ノードを使用する例）
!hpf$ distribute a(block) onto p
      do 10 i=1,100
       a(i)=i
 10   continue
      write(*,*)  (a(i),i=1,100)
      end

% pf90 -Wt,'comlib(rdma)' program.f              … トランスレート
pf90: translate start : program.f                 指示文を含むプログラムを並列化ソースプログラムに変換(通信ライブラリーとしてリモート DMA 関数を使用する例)
                                                  ※通信ライブラリーを MPI にする場合
*Parallel FORTRAN V01-01 entered.                   % pf90 -Wt,'comlib(mpi)' program.f
*program name = MAIN
*program units = 0001, no diagnostics
generated.
% ls                                             pf_node ディレクトリー下に変換後のソースプログラムが置かれる
pf_node    program.f
% ls pf_node
program.f
% f90 pf_node/program.f -lhpf
f90: compile start : pf_node/program.f          … コンパイル
                                                  f90 コマンドを使用（HPF ライブラリーを指定する）
*OFORT90 V01-01-A entered.
KCHF656K -I the following file is                 ※通信ライブラリーが MPI の場合
referred.--/usr/include/hpf_var_info.f90            % mpif90 pf_node/sample.f -lhpf
KCHF656K -I the following file is
referred.--/usr/include/hpf_intrinsics.f90        ※MPI 機能で提供されているコマンドを使用する場合には環境変数 path の設定が必要
KCHF656K -I the following file is                   % set path=($path  /usr/mpi/bin)
            :
*program name = MAIN
*end of compilation : MAIN
*program units = 0001, 0002  diagnostics
generated, highest severity code is  00

% prun -n 2 a.out                                … 実行
        1             2             3             prun コマンドを使用
4         5          6                           ※通信ライブラリーが MPI の場合
            :                                       % mpirun -n 2 a.out
```

mpp-p/bulk-p を使用したインタラクティブ実行例です。バッチジョブ（並列ジョブクラス）を使用することにより最大 16 ノードまでの並列化が可能です。

参考マニュアル　「Parallel FORTRAN 言語」(6A30-3-320)
「Parallel FORTRAN 使用の手引」(6A30-3-321)

4. 数値計算ライブラリー

SR8000 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 f77、f90 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）	–l***matmpp***
	（スカラー版）	–l***matmpp_sc***
MATRIX/MPP/SSS（スカイライン法）	（要素並列版）	–l***matmpps***
	（スカラー版）	–l***matmpps_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」(6A30-7-600)
「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」(6A30-7-601)

使用例（単一ノードの例）

```
% 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

% f77 hsru1m.f -lmatmpp_sc              … コンパイル
f77: 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又は mpp-s/bulk-s で
*OFORT77 V01-02 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
% f77 hsru1m.f -lmatmpp -parallel           … コンパイル
f77: compile start : hsru1m.f               要素並列版ライブラリーを使用
                                            （並列ジョブクラス又は mpp-p/bulk-p で実
*OFORT77 V01-02 entered.                    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
          2  0.158839539
          3  0.645628750
                :
         10  0.629399896
```

mpp-p/bulk-p を使用したインタラクティブ実行例です。（スカラー版は mpp-s/bulk-s でも実行可）

使用例（複数ノード実行の例）

```
% 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
% f77 hdru3mdp.f -lmatmpp -parallel         … コンパイル
f77: compile start : hdru3mdp.f             要素並列版ライブラリーを使用

*OFORT77 V01-02 entered.
*program name = MAIN
*end of compilation : MAIN
*program  units  =  0001,  no  diagnostics
generated.
% prun -n 2 a.out                          … 実行
          0          20           0         prun コマンドを使用
                     :
```

mpp-p/bulk-p を使用したインタラクティブ実行例です。バッチジョブ（並列ジョブクラス）を使用することにより最大 16 ノードまでの並列化が可能です。

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）	–l***MSL2P***
	（スカラー版）	–l***MSL2***

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」(6A30-7-613)
「数値計算副プログラムライブラリ MSL2 行列計算」(6A30-7-610)
「数値計算副プログラムライブラリ MSL2 関数計算」(6A30-7-611)
「数値計算副プログラムライブラリ MSL2 統計計算」(6A30-7-612)

使用例（スカラー処理の例）

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

% f77 msgu1m.f -lMSL2                          … コンパイル
f77: compile start : msgu1m.f                    スカラー版ライブラリーを使用

*OFORT77 V01-02 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
```

mpp-s/bulk-s でインタラクティブ実行できます。

使用例（要素並列処理の例）

```
% 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

% f77 msvafm.f -lMSL2P -parallel                    … コンパイル
f77: compile start : msvafm.f                        要素並列版ライブラリーを使用

*OFORT77 V01-02 entered.
*program name = MAIN
*end of compilation : MAIN
*program  units  =  0001,  no  diagnostics
generated.

% a.out                                              …  実行
  2.00000000                  4.00000000             a.out を実行する
6.00000000       8.00000000
```

mpp-p/bulk-p を使用したインタラクティブ実行例です。なお、以下の副プログラムに要素並列化版があります。

MCLF1M	MDLB1M	MDVAFM	MDVF2M	MSLF1M	MSVB2M	MSVMTM
MDFB2M	MDLF1M	MDVAHM	MDVFHM	MSLK1M	MSVF1M	MZFB2M
MDFC2M	MDLF3M	MDVB1M	MDVMFM	MSVAFM	MSVF2M	MZLF1M
MDFI2M	MDLF5M	MDVB2M	MDVMTM	MSVAHM	MSVFHM	
MDFS2M	MDLK1M	MDVF1M	MSLB1M	MSVB1M	MSVMFM	

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）は連立一次方程式、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 URL 又はスーパーコンピューティングニュース Vol.3 No.5(2001.9)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	–l***blas***	–l***blas_sc***	
LAPACK	–l***lapack*** –l***blas***	–l***lapack_sc*** –l***blas_sc***	
ScaLAPACK	–l***scalapack*** –l***pblas*** –l***tools*** –l***redist*** –l***blacsBASE*** –l***blacsF77*** –l***blas***	–l***scalapack_sc*** –l***pblas_sc*** –l***tools_sc*** –l***redist_sc*** –l***blacsBASE_sc*** –l***blacsF77_sc*** –l***blas_sc***	(ScaLAPACK) (PBLAS) (tools) (redistribution) (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。（mpif77、mpif90 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）

```
% 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
% f77 blas.f -lblas -parallel                … コンパイル
f77: compile start : blas.f                    要素並列版ライブラリーを使用

*OFORT77 V01-02 entered.
*program name = BLAS
*end of compilation : BLAS
*program  units  =  0001,  no  diagnostics
generated.
% a.out                                      … 実行
  1.12500000                                   a.out を実行する
  2.00000000
  3.25000000
  4.00000000
```

mpp-p/bulk-p を使用したインタラクティブ実行例です。

使用例（LAPACK）

```
% 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
% f77 lapack.f -llapack -lblas -parallel     … コンパイル
f77: compile start : lapack.f                  要素並列版ライブラリーを使用

*OFORT77 V01-02 entered.
*program name = LAPACK
*end of compilation : LAPACK
```

```
*program  units  =  0001,  no  diagnostics
generated.
% a.out                                          … 実行
  3.00000000000000000                              a.out を実行する
  2.00000000000000000
```

mpp-p/bulk-p を使用したインタラクティブ実行例です。

使用例（ScaLAPACK）

```
% mpif77 fexample.f -parallel -lscalapack     … コンパイル
     -lpblas -ltools -lredist -lblacsBASE        mpif77 コマンドを使用
    -lblacsF77 -lblacsBASE -lblas
f77: compile start : fexample.f                  ※MPI 機能で提供されているコマンドを
                                                   使用する場合には環境変数 path の
*OFORT77 V01-02 entered.                           設定が必要
*program name = DLU                                % set path=($path  /usr/mpi/bin)
*program name = AINITBLK
          :                                      サンプルプログラムは
*end of compilation : PRTMTX                      /usr/examples/ScaLAPACK/fexample.f
*end of compilation : AINITMTX                   をコピーしたもの
*program units = 0007, 0001   diagnostics
generated, highest severity code is  00
% mpirun -n 2 a.out
The input values of matrix A on (0, 0) are :  … 実行
  0.0  1.00000  1.0000   1.00000    1.00000      mpirun コマンドを使用
           :
The Solutions on (0,  0) are :
   5.60000    4.60000    3.60000    2.60000
1.60000      0.60000
  -0.40000   -1.40000   -2.40000   -3.40000
-4.40000
%
```

mpp-p/bulk-p を使用したインタラクティブ実行例です。バッチジョブ（並列ジョブクラス）を使用することにより最大 16 ノードまでの並列化が可能です。

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	SR2201 形式	M 形式
単精度	$\pm 1.175495\times10^{-38}$～$\pm 3.402823\times10^{38}$	$\pm 1.175495\times10^{-38}$～$\pm 3.402823\times10^{38}$	$\pm 5.397606\times10^{-79}$～$\pm 7.237005\times10^{75}$
倍精度	$\pm 2.225074\times10^{-308}$～$\pm 1.797673\times10^{308}$	$\pm 2.225074\times10^{-308}$～$\pm 1.797673\times10^{308}$	
拡張精度	$\pm 2.225074\times10^{-308}$～$\pm 1.797673\times10^{308}$	$\pm 3.37\times10^{-4932}$～$\pm 1.189731\times10^{4932}$	

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 SR8000 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 SR8000 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。

SR8000 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
% f77 program.f
% 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、標準出力へ出力する例
% cat a.data
2 3
% a.out < a.data                             …  ファイル a.data から入力。
  6.00000000     0.666666687

ファイル a.data から入力、ファイル a.result に出力
% cat a.data
2 3
% a.out < a.data > a.result                  …  ファイル a.data から入力、
% cat a.result                                        a.result に出力
  6.00000000     0.66666627                  …  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る。

```
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

```
% cat program.f
      read(10,*) a,b
      write(11,100) a*b,a/b
 100  format(1h ,2d12.4)
      stop
      end
% f77 program.f
% 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
#@$-M 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事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。）

	SR8000 形式	SR2201 形式	M 形式
mpp-s/p/bt bulk-s/p/bt	FT*nn*F*xxx* （又は FT*nn*S*xxx*）	FT*nn*E*xxx*	FT*nn*M*xxx*
m-unix	なし	FT*nn*E*xxx*	FT*nn*F*xxx*

例えば SR8000 システムで環境変数 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。

```
（SR8000 システムの場合）
setenv FT10F001 data1                  …  SR8000 形式で入出力
setenv FT11E001 data2                  …  SR2201 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
setenv FT11E001 data2                  …  SR2201 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

```
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
% f77 a.f
% a.out
?
2 3
% cat ft.11                                        …　ファイルがなければ作成される。
6.00000000     0.66666627

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
% f77 a.f
% a.out
KCHF348R the read statement for unit id 10 is invalid. the file does not exist.

（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     0.66666627
```