

ミニスーパーコンピュータ (vxhost1) における fortran の利用

総合情報処理センター 車 古 正 樹

1. NQS (Network Queuing System) によるバッチジョブの依頼

1-1. バッチジョブの種類

ジョブ・クラスを以下の表に示す。

クラス	cpu時間	種別	サイズ
A	60分		256MB
B	120分		512MB
C	240分		1024MB
D	480分	並列可	2048MB

1-2. バッチジョブの依頼

nqs (Network Queuing System) コマンド

nqs コマンドの主なものについて以下に記す。

《qsubコマンド》

バッチに job を依頼するコマンドである。

qsub -q to_ {A/B/C/D} {ファイル名} {その他}

-q to_ {A/B/C/D} 依頼するジョブキューの指定 (vxhost1からの場合はto_は不要である。)

ファイル名 コマンドの入ったファイル名を指定する。省略した場合は続けてコマンドを入力し、(ctrl)Dで終了する。使用例を参照する。

-a 実行する時間を "11:30pm JST" のように指定する。

-e 標準エラー出力ファイルを prog1.out のように指定する。

-e o 標準エラー出力ファイルを標準出力ファイルに出力する。

-j ジョブ名を指定する。

-lT CPUのタイムリミット時間を 1:25:00 のように指定する。

-o 標準出力ファイルを指定する。

- mi** 統計情報をメールで通知する。なお、開始時刻は **-mb**、終了時刻は **-me** を指定する。
 - mu** メールアドレスを指定する。例: **-mu tarou@kipcgr02**。
ただし、**kipcgr01 ~ kipcgr05** のみ指定できる。
- その他については、**man** コマンドで参照下さい。

q s u b の使用例

qsub コマンドの実行は **kipcgr01 ~ kipcgr05** のいずれかに **login** して行う。

例1. 直接ジョブをクラスAに依頼

- **qsub -q to_A** クラスAへ依頼する。
- **frt prog1.f < prog1.data** プログラム **prog1.f**、データ **prog1.data** を利用して計算する。
- **a.out** プログラムの実行する。
- **(ctrl)D** コマンドの入力を終了する。

例2. コマンドファイル **submit** を指定してクラスBに依頼

- **qsub -q to_B submit** クラスBへ依頼する。
- submit** ファイルの内容は実行するためのコマンドを記述しておく。

例3. シェル **asub** を用いた依頼

- **vi asub** **asub** ファイルに実行コマンドを作成する。
- (**asub** コマンドの例)
- Cクラスにサブミットする。プログラム名は引数で渡す。ジョブ名をプログラム名 **.que** とする。実行モジュールはプログラム名 **.count** を一時的に使用する。
- **#!/bin/csh** シェルコマンドであることを宣言する。
 - **qsub -q to_C-J\${1} .que << end**
\$ {1} はパラメーターであり、**end** は終了文字列宣言である。
 - **frt frt/\${1}.f -Pa -KVX -o\${1}.count**
fortran コマンドを実行する。
 - **./\${1}.count** プログラムを実行する。
 - **rm ./\${1}.count** 実行形式のプログラムを削除する。
 - **end** コマンド列の終了
 - **cmod 700 asub** **asub** を実行形式にする。
 - **asub prog1** **asub** コマンドでジョブを依頼する。

《q s t a t コマンド》

バッチ **job** の状態を調べるコマンドである。

qstat {-a} {-l}@vxhost1

- a 全てのユーザについて表示する。
- l 詳細な形式で表示する。
- @vxhost1 vxhost1 以外から利用している場合に指定する。

バッチジョブの制限

クラス	連続投入数	合計投入数
A	4/16	16
B	2/8	16
C	1/4	8
D	1/2	4

平日/休日前日17時から休日17時まで

ディスクの制限

- ファイル制限値 1.0GB/人
- 制限値を超えて作成されたファイル 3日後に削除
- 1年間アクセスのないファイルは削除

《qdelコマンド》

依頼したバッチ job を削除するコマンドである。

qdel {-k/-シグナル番号}{-r vxhost1 リクエスト番号}

- k 実行中のバッチを削除する。
- シグナル番号 指定されたバッチを削除する。
vxhost1 以外からの場合は -r vxhost1 999のように指定する。

2. f r t (f o r t r a n コンパイラ) コマンドのパラメータ

パラメータについては、man f r t で参照できる。ここでは簡単に紹介する。

パラメータについて

- c: 実行形式プログラムを作成しない。コンパイラのみ^{注意1}が実行される。
注意1: Fortran の文法チェックのみを行う場合は実行形式プログラムを作成しない方が、リンカが起動しない分効率が良い。
- o: 実行形式プログラムを保存^{注意2}するファイル名を指定する。例 -o prog1.out
注意2: 実行形式プログラムの保存を指定しない場合は、a.out という名のファイルに保存される。したがって、バッチ計算をしている時、次のプログラムを翻訳した場合に同一名となり、エラーの原因となる。バッチ計算を依頼する場合は必ず指定するようにする。
- D<a,s,u>: デバッグ機能である。 a: 引数の対応を調べる。 s: 添字式及び部分列式の値を調べる。
u: 未定義データの引用を調べる。これらを組み合わせて指定する。

-Am: 複数ファイルの同時翻訳をする。例: `frc a.f b.f -Am`

-sc: スカラ翻訳を指示する。ベクター長が8以下の場合有効である。

-l<ssl2|ssl2vp|ssl2vpp>: `ssl2`: `sun` で `ssl2` を利用する。`ssl2vp`: `vxhost1` で `ssl2` を利用する。`ssl2vpp`: `vxhost1` で並列化 `ssl2` を利用する。

-P<s,l,a>: `s`: ソースプログラムを出力する。 `l`: 入れ子を表示する。 `a`: ベクトル化情報を表示する。

-Ne: 実行文数が少ない外部手続きをインライン展開する。

-X<7/9/f7>: `-X7`: `fortran77` , `-X9`: `fortran90` , `-Xf7`: 富士通 `fortran77` として解釈する。ただし、コンパイラのデフォルト値はファイル識別子が `.f` の場合 `fortran77` , `.f90` の場合 `fortran90`とみなされ、それぞれ `-X7` , `-X9`となる。

-Ad: 単精度を倍精度に精度を拡張する。例: `frc test.f -Ad`

<-Free/-Fixed>: `-Free` : 自由形式プログラム、`-Fixed` : 固定形式プログラムとして翻訳する。ただし、コンパイラのデフォルト値はファイル識別子が `.f` の場合 `Fixed` , `.f90` の場合 `-Free` とみなされる。

-Of: プログラム単位間の最適化とスカラループに対する最適化も行う。

-Os<,-p,-u,-e,-l>: `-p` : 不変式の先行評価を行う。 `-u` : `do` ループの回転数を減らす。
`-e` : 演算の評価方法を変更する。 `-l` : 多重ループの構成変更を行う。
なお、これらの最適化を行った場合に値が変わることもあるので注意をようする。

3. SUNでのFORTRAN利用の動作環境設定

3-1. SUN上のコンパイラ動作環境

SUN ワークステーション (`kipcgr01` ~ `kipcgr05`) で `fortran` コンパイラを使用しデバッグする場合は、`.cshrc`に以下の行を追加する。

```
#SUN
if ('uname'== "SunOS") then
  set path = ($path /opt/FSUNf90/bin)
  setenv LD __LIBRARY_PATH/usr/lib:/opt/FSUNf90/lib
  setenv MANPATH $MANPATH:/opt/FSUNf90/man
endif
```

3-2. VPP Workbenchの環境設定

SUN から vxhost1 でプログラムを実行し、デバッグをするための環境設定は、.cshrcに以下の行を追加する。

```
#VPP Workbench
if ( `uname` == "SunOS" ) then
    set path = ( $path /opt/FSUNvppwb/bin )
    setenv MANPATH $MANPATH:/opt/FSUNvppwb/man
endif
```

起動コマンドはvppworkbenchである。

4. 入出力文に対する装置番号とファイルの接続

4-1. 割当のない標準入出力

バッチ処理の場合：標準出力（論理機番6）はファイル名STDIN.○受付番号に出力される。

直接実行の場合：標準出力（論理機番6）は端末に出力される。標準入力（論理機番5）がある場合はキーボードとなる。

4-2. OPEN文による割当

例：標準入力ファイルが data/prog1.txt にあり、標準出力を prog1.out にする場合は、以下のようになる。

```
open(5,file='data/prog1.txt')
open(6,file='prog1.out')
```

4-3. OPEN文がない場合の割当

例：標準入力ファイルが data/prog2.txt にあり、標準出力を prog2.o にする場合は、以下のようになる。ただし、実行形式プログラムが a.out にあるものとする。

a.out < data/prog2.txt > prog2.o: <は標準入力を、>は標準出力を指定する。

4-4. 標準入出力以外の割当

例：論理機番1の入力ファイルに a.file を割り当てる。ただし、実行形式プログラムが a.out にあるものとする。

```
setenv fu01 a.out          fu 論理機番2桁を指定する。
a.out < data/prog2.txt > prog2.o  <は標準入力を、>は標準出力を指定する。
```

5. プログラム開発支援のためのWorkbench

5-1. 環境設定と起動方法

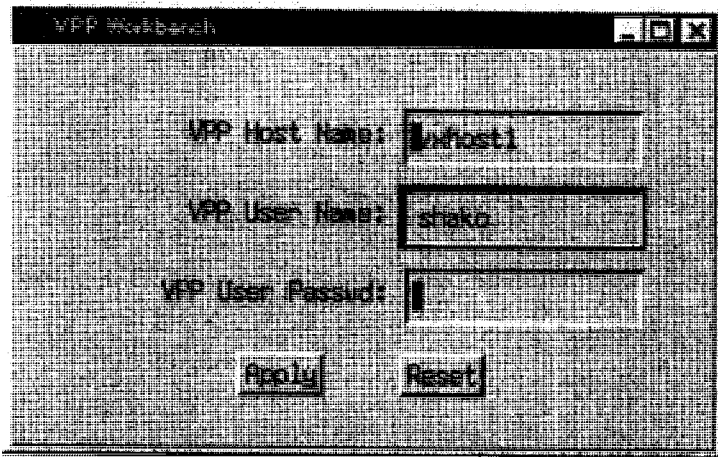


図1

vpp workbench コマンドで起動すると数秒後に図1の画面が表示される。

VPP User Name 欄とVPP User Password 欄に入力して、【Apply】をクリックする。

10数秒すると図2のManager ウィンドウが表示される。

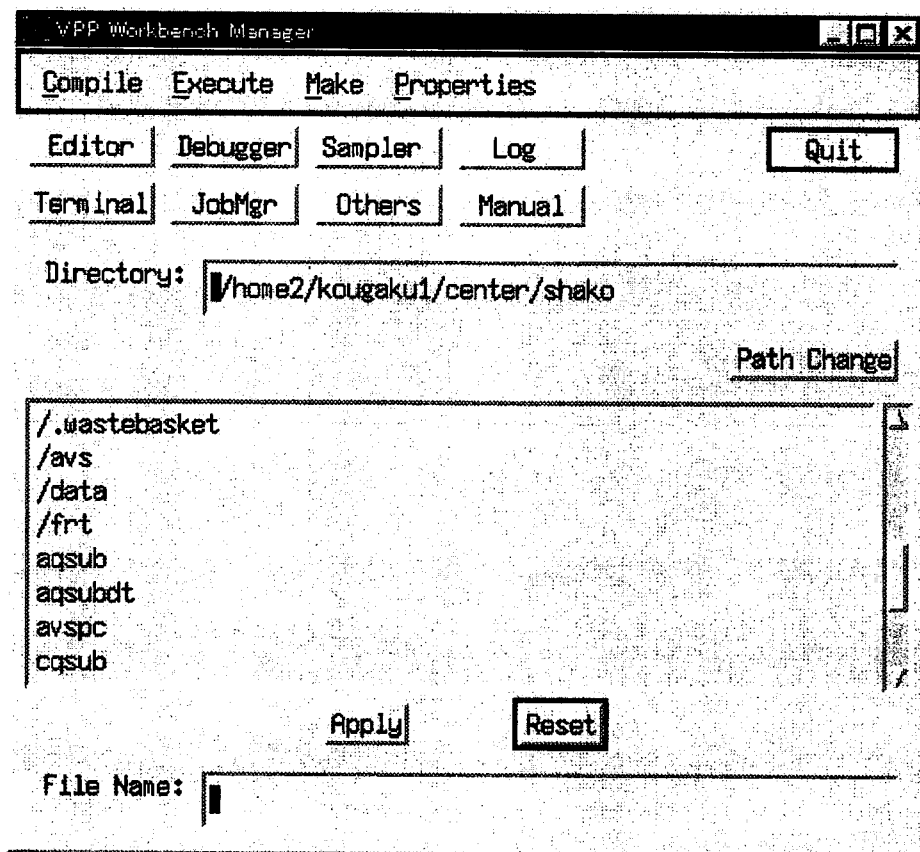


図2

終了は【Quit】をクリックする。

5-2. 翻訳と実行

翻訳の仕方

Maneger ウィンドウの File Name 欄に翻訳するプログラムファイル名を直接入力する。
あるいはプログラムを選択して【Apply】をクリックする。

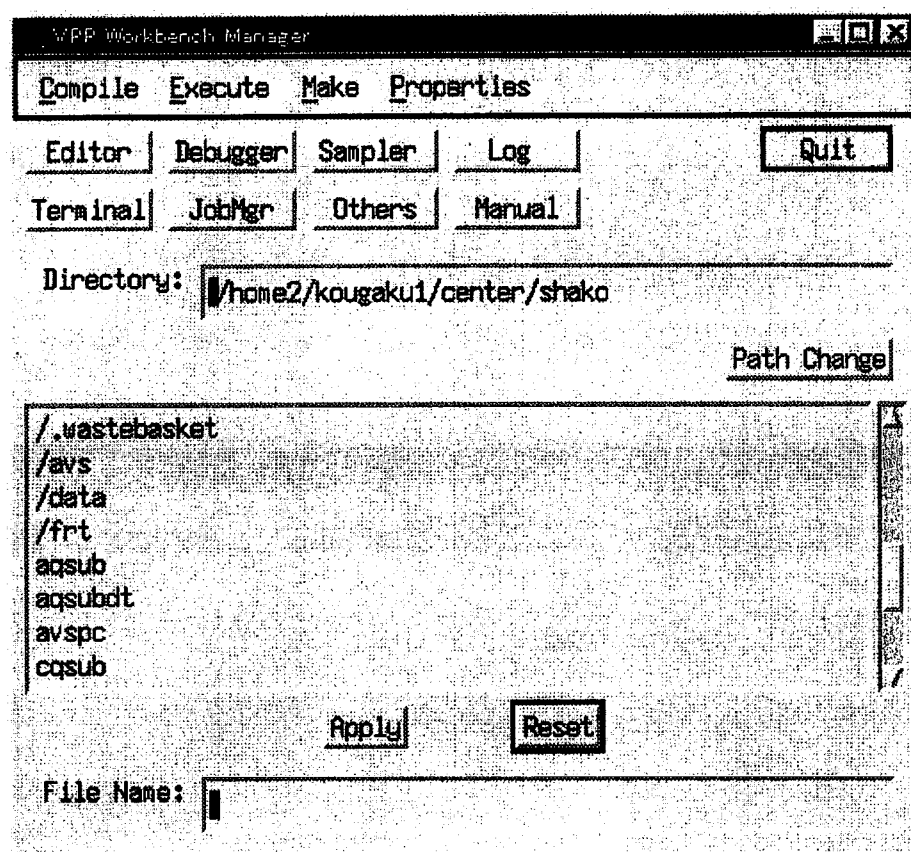
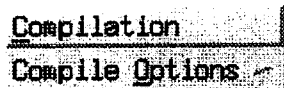
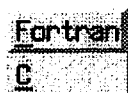


図3



【Compile】をクリックすると【Compilation】と【Compile Options】が表示される。



翻訳する前に必要な条件を設定するために、【Compile Options】をクリックし、次に【Fortran】を選択する。

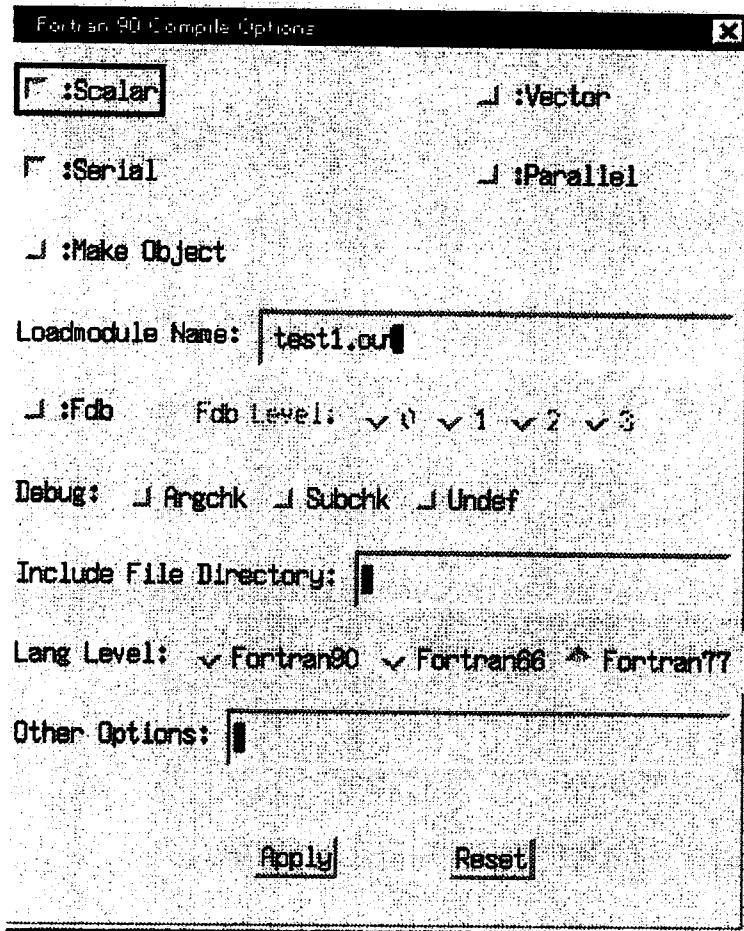


図4

全て指定が終了したならば、【Apply】をクリックする。次にもとのウィンドウで【Compile】をクリックし、次のウィンドウで【Compilation】をクリックする。

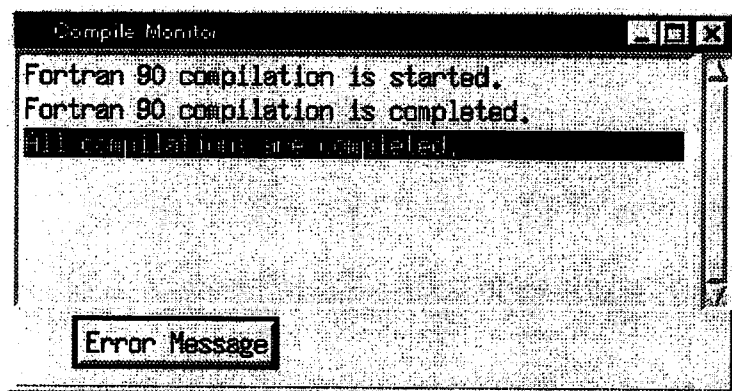


図5

Fortran 90 Compile Options ウィンドウで必要事項を入力する。ベクトル長が8以下の場合には【Scalar】をそれ以上の場合には【Vector】を選択する。バッチ計算のDクラス以外は必ず【Serial】を選択する。Loadmodule Name欄に実行形式のプログラムを格納するファイル名を入力する。省略時はa.outとなる。Debugは必要に応じて指定する。Lang Levelは言語に応じて指定する。Other Options 欄には上記以外で指定したいオプションを指定する。なお、常に指定したい場合は、最初のManeger ウィンドウの Propertiesを選択して指定しておく。ssl2 を利用する場合は、-lssl2vp とこの欄で指定する。並列化の場合は -lssl2vpp と指定する。

Complote Monitor ウィンドウが表示される。終了メッセージが表示されたら、エラーがなかったか確認するため、【ERROR】をクリックする。

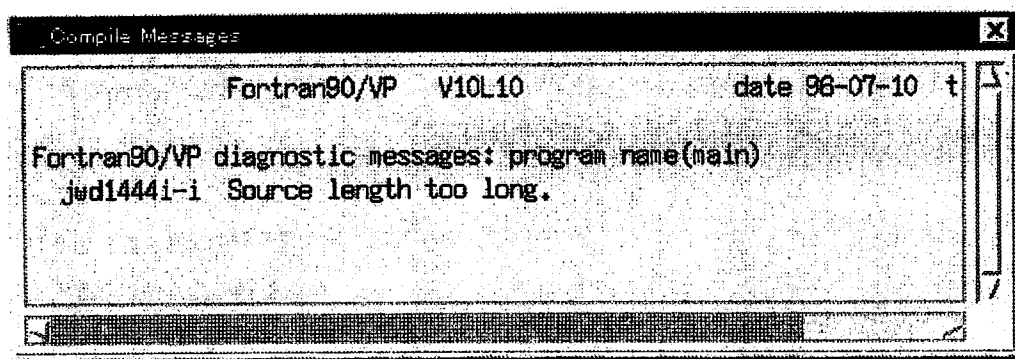
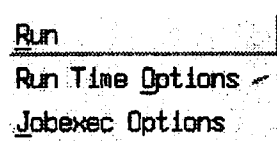


図6

Compile Message ウィンドウが表示され、エラーメッセージが各ウィンドウを終了する。

実行の仕方



Manager ウィンドウで【Execute】をクリックし、次のウィンドウで【Run Time Options】をクリックする。

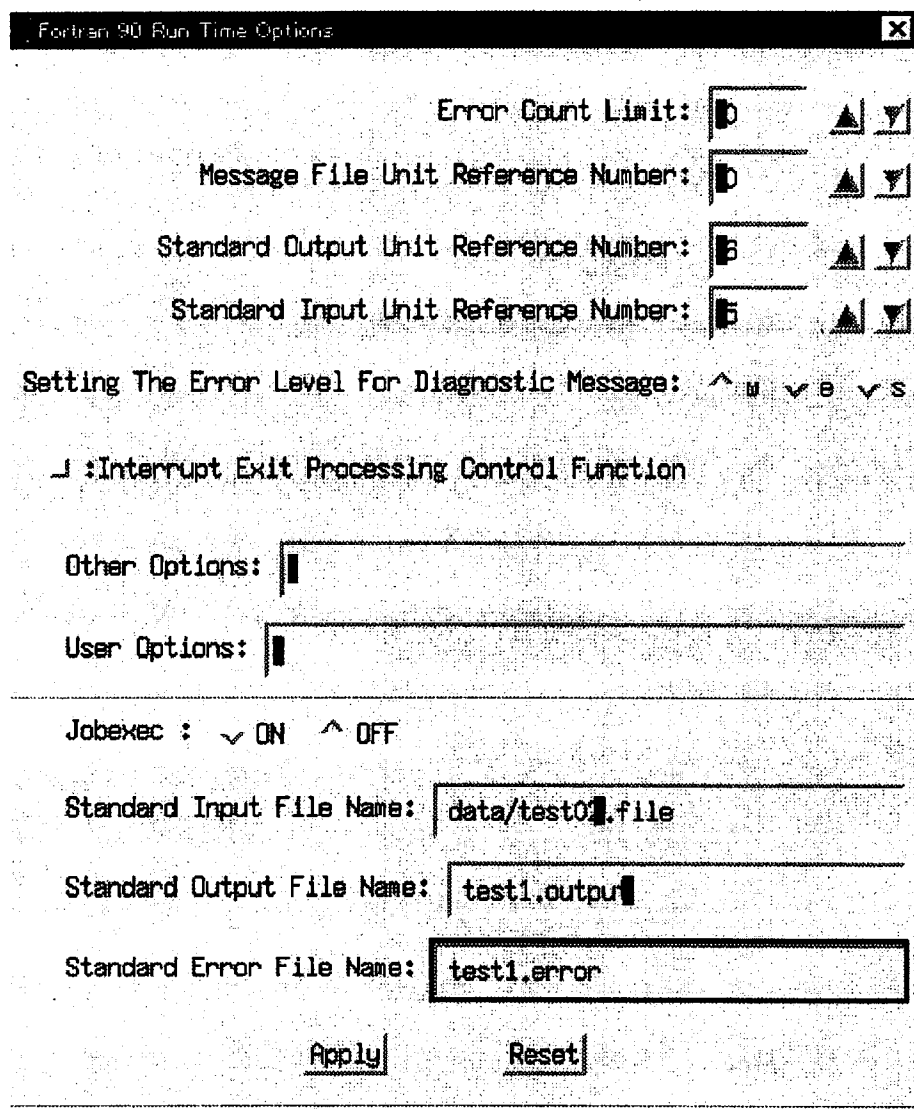


図7

論理機番5にデータファイルを割り当てる必要がある場合は、【Standard Input File Name】欄にファイル名を指定する。

論理機番6の出力を端末以外のファイルに割り当てる必要がある場合は、【Standard Output File Name】欄にファイル名を指定する。

必要な入力が終了したならば、【Apply】をクリックする。なお、論理機番5、6以外を使用している場合は、Maneger ウィンドウで【Properties】をクリックし、次のウィンドウで【Run Time File Allocation】をクリックする。Run Time File Allocation ウィンドウが表示される。Unit Numberを選択し、Allocation File Name欄にファイル名を指定する。Maneger ウィンドウでExecuteをクリックし、次のウィンドウで【Run】をクリックすれば実行が開始され図8のウィンドウが表示される。



図8

5-3. Debuggerの利用

Maneger ウィンドウ の【Debugger】を利用する場合は、以下の手順で行う。

翻訳上の注意

File Name 欄にデバックするプログラムを指定し、Compile Optionsウィンドウで必ず【Fdb】を選択してから翻訳する。

実行の準備

入出力をファイルで行う場合は、ExecuteのRun Time Optionsウィンドウで指定する。

Debuggerの起動

Maneger ウィンドウのFile Name 欄に実行形式のプログラムが入ったファイルを指定して【Debugger】をクリックする。

ブレークポイントの設定と解除

ソース領域の右側をクリックするとブレークポイントが設定され、マークが表示される。

なお、現在の停止行（右矢印のある行）の場合は【Break】をクリックするか【Delete】をクリックする。

プログラムの実行と停止

実行は【Run】をクリックする。停止はブレークポイントの位置である。

強制的に停止させるには、【Program】をクリックし、【Kill】をクリックする。

プログラムの実行の継続

1行毎に実行する場合は【Next】か【Step】をクリックする。

次のブレークポイントまで連続実行する場合は【Continue】をクリックする。

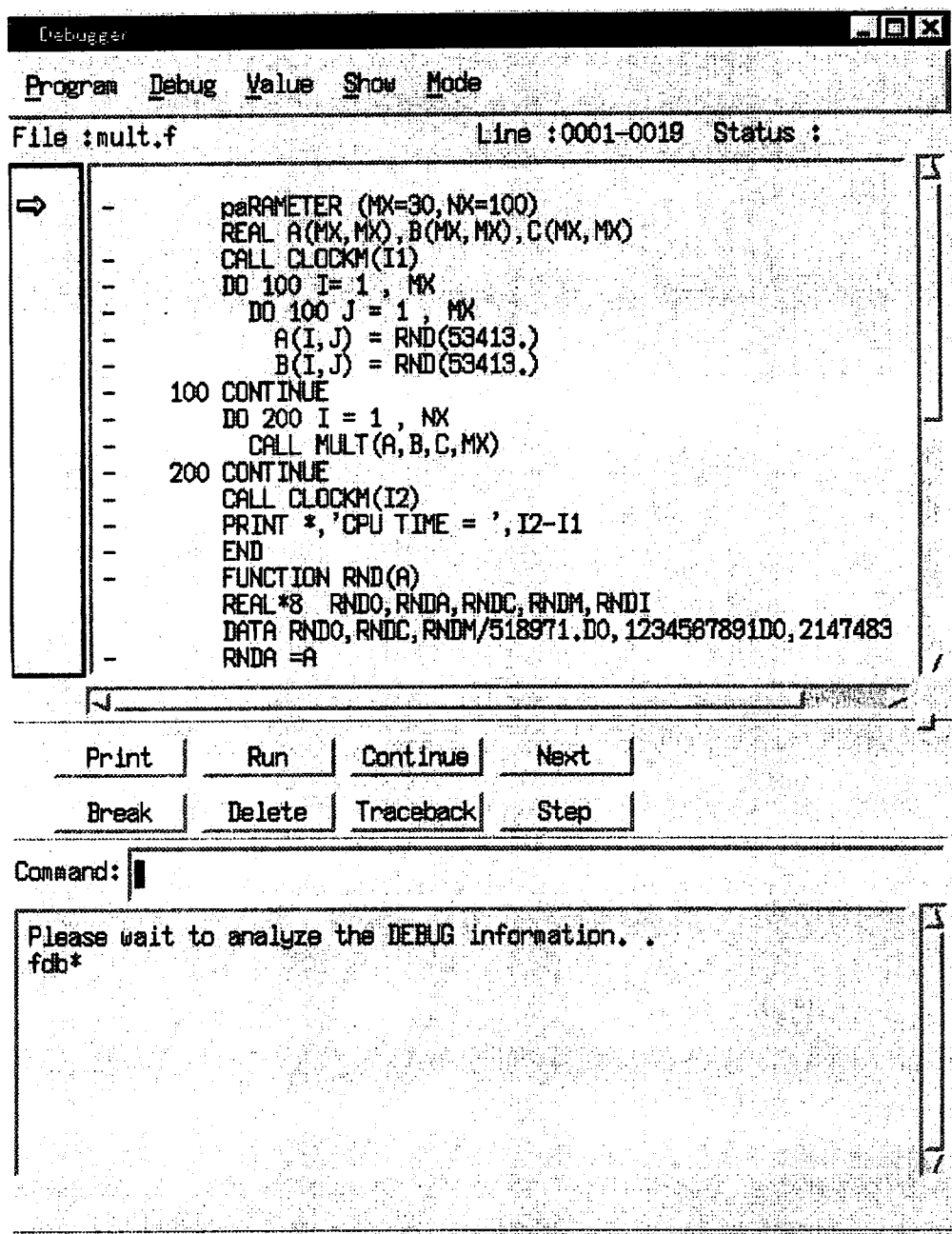


図9

変数の値の表示

ソース領域で変数名をドラッグし、【Print】をクリックする。または、【Value】をクリックし、

【Screen】をクリックする。Value Screen ウィンドウが表示されたら、変数を登録する。
実行が停止するごとに変数の値が見れる。

行の実行回数 (カウントテストカバレッジ)

【Mode】をクリックし【Conver】を選択する。一個以上のブレークポイントを設定し実行する。
ブレークポイント以降の行の実行回数がカウントされる。

Conver が選択されている場合の実行速度はソースを表示しながら実行するために非常に時間がかかる。

回数の参照は、【Debug】をクリックし、【Convergence】をクリックしてソースプログラムを選択する。
結果を図10に示す。

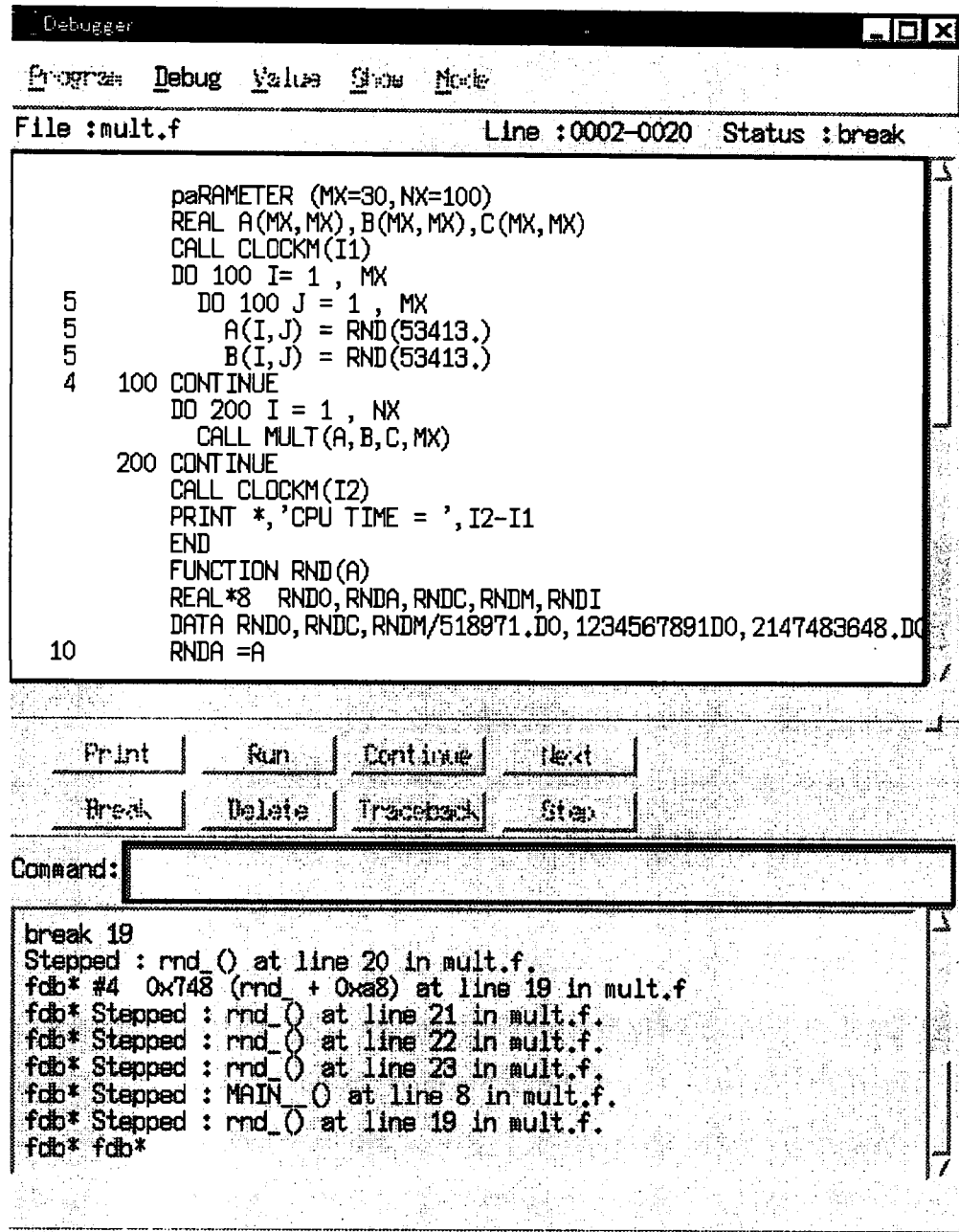


図10

5-4. Samplerによる解析と実行速度

Maneger ウィンドウ の【Sampler】を利用する場合は、以下の手順で行う。

翻訳上の注意

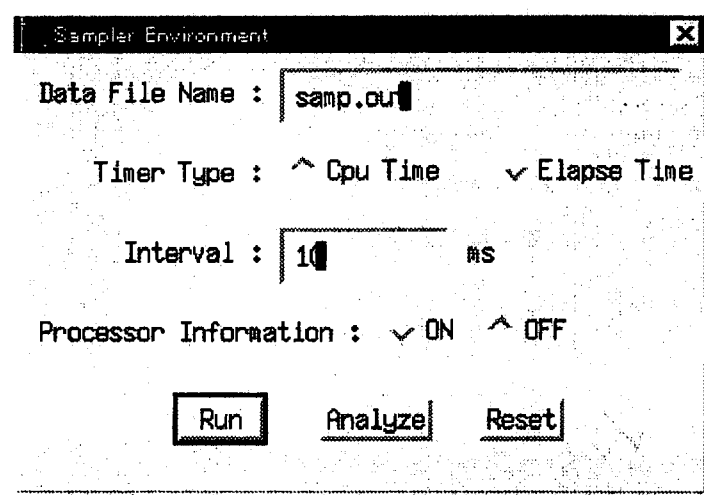
File Name 欄にデバックするプログラムを指定し、Compile Optionsウィンドウで必ず【Vector】を選択してから翻訳する。

実行の準備

入出力をファイルで行う場合は、ExecuteのRun Time Optionsウィンドウで指定する。

Samplerの起動

Maneger ウィンドウ のFile Name欄に実行形式のプログラムが入ったファイルを指定して【Sampler】をクリックする。



Data File Name 欄に解析用データを出力するファイルを指定する。Interval 欄にサンプル時間を指定する。

【Run】をクリックする。

実行が終了したならば、

【Analyze】をクリックすると以下のような結果が表示される。

図11

図12のようにモジュール(手続き)毎のcpu時間の負荷率やベクトル長あるいはループ率等が出力されている。これを基にプログラムをチューニングするとよい。

また、この場合の計算はcpu負荷率の一番高いところのベクトル長が6であるからスカラ計算をした方が早くなる。実際の計算はベクトル化した場合は23.44秒、スカラ計算では19.76秒であり、かつ、インライン指定(-Ne)をした場合は12.41秒と約半分の時間となった。

```

x190run101
Status : Serial
Number of Processors : 1

Type : cpu
Interval (msec) : 10

Synthesis Information
Count| Percent| VLI Name
2074| 90.8| 6| MULT_
106| 4.6| 20| FIELD_
6| 2.7| -| MAIN_
2| 0.9| 3| DETMT_
9| 0.4| 29| MOVE_
4| 0.2| -| FIELD1_
4| 0.2| 3| MOMENT_
3| 0.1| -| MINIV_
1| 0.0| -| POINT2_
1| 0.0| -| INS_

2284| | 8| TOTAL

Program Unit Information(MULT_)-----
Procedure List:
Count| Percent| VLI Name
2074| 100.0( 90.8)| 6| mult_

2074| ( 90.8)| 6| PROCEDURE_TOTAL

Loop & Array Expression List:
Count| Percent| V_Mark| kind | VLI Line
1944| 93.7( 85.1)| V | DO | 6| 00000140-000001
52| 2.5( 2.3)| V | DO | 6| 00000138-000001
50| 2.4( 2.2)| S | DO | -| 00000137-000001

Program Unit Information(FIELD_)-----
Procedure List:
Count| Percent| VLI Name
106| 100.0( 4.6)| 20| field_

106| ( 4.6)| 20| PROCEDURE_TOTAL

Loop & Array Expression List:
Count| Percent| V_Mark| kind | VLI Line
21| 19.8( 0.9)| S | DO | -| 00000230-000002
19| 17.9( 0.8)| S | DO | -| 00000238-000002
12| 11.3( 0.5)| V | DO | 10| 00000201-000002
9| 8.5( 0.4)| S | DO | -| 00000234-000002
7| 6.6( 0.3)| S | DO | -| 00000222-000002

```

図12

6. マトリックス演算の実行時間の目安

下記の表は各行列についてマトリックスの乗算を100回行った結果である。

サイズ	vxhost1	vxhost1	vphost	vphost
	A:スカラー演算 A/B	B:ベクトル演算 cpu時間ms	C:スカラー演算 C/B	D:ベクトル演算 D/B
5行5列	0.5/3	6	0.33/1.98	
8行8列	1.18/13.0	11		
10行10列	1.47/25.0	17	0.82/12.24	0.94
30行30列	2.06/295	143	2.10/300	1.04
100行100列	25.6/41,779	1,632	6.50/10,608	1.68
500行500列	109./5,733*	52,597	22.2/1,167*	3.88
1000行1000列	181.3/50212*	276,958	129/35,930*	7.11
2000行2000列	342./586,726*	1,714,111		

下記の表は配列宣言を2048に固定して各行列についてマトリックスの乗算を1回行った結果である。

サイズ	vxhost1	vxhost1	vphost	vphost
	A:スカラー演算 cpu時間ms	B:ベクトル演算 cpu時間ms	C:スカラー演算 cpu時間ms	D:ベクトル演算 cpu時間ms
8	1.24	0.11	0.088	0.11
16	4.45	0.42	0.49	0.43
32	30.28	1.64	3.17	1.82
64	150.8	6.53	20.1	9.3
128	1,765	27.78	461.	52.0
256	10,090	121.6	4,739	322
512	70,699	585.6	37,709	2,201
1024	547,519	3,146	428,903	34,988
2048	6,364,898	18,326		415,850