UXP/Fortran 利用法

渡部 善隆 *

九州大学大型計算機センターでは 1996 年 1 月に汎用計算機を更新しました。新汎用機の大きな「目玉」のひとつとして、 UNIX 上で対話的なベクトル演算処理が可能となりました。これによって、従来はバックグラウンドでの処理のみであったベクトル計算がぐっと身近になりました。

本稿では、現在のベクトル計算の主力言語である Fortran に的を絞り、汎用計算機の UNIX OS である UXP/M(以下 UXP とします) での Fortran プログラムの実行方法およびベクトル計算機 VP2600/10 へのジョブの投入方法について解説します。

なお、これをお読みの方は、 UNIX のディレクトリの概念にある程度慣れており、 emacs, mule, vi などの UNIX 上のエディタを使ったファイルの編集ができるものとさせていただきます。

目次

*九州大学大型計算機センター・研究開発部　E-mail : f70029a@kyu-cc.cc.kyushu-u.ac.jp

1　基本的な使い方

この章では UXP/Fortran を対話的に使用するための基本的なコマンド、オプションについて説明します。

1.1　プログラム開発の流れ

1.1.1　計算機環境

UXP/Fortran は、汎用計算機 FUJITSU M-1800/20U とベクトル計算機 VP2600/10 上で動作します。プログラムの実行は、 M-1800/20U は対話型処理およびバッチ処理で、 VP2600/10 はバッチ処理のみで行われます。

計算機名	対話型処理	バッチ処理	最大記憶域	最大処理性能
M-1800/20U	○	○ (注)	100MB	1.2GFLOPS
VP2600/10	×	○	400MB	4.9GFLOPS

また、汎用計算機 M-1800/20U の対話型処理でベクトル演算処理が実行可能となりました。ただし、汎用機のバッチ処理ではベクトル演算処理は実行できませんのでご注意下さい。

VP2600/10 にジョブを投入する場合、窓口は M-1800/20U が担当します。利用者は M-1800/20U から VP2600/10 に対して翻訳・実行を依頼 (サブミットといいます) します。 VP2600/10 は処理結果を M-1800/20U にファイルとして返却します。

1.1.2　ベクトル演算処理

新汎用機 M-1800/20U には、 VP2600/10 と同様にベクトル演算処理機能がサポートされました。この機能を使用することで、プログラムによっては従来のスカラー処理に比べて数十倍の高速実行が可能です。さらに、対話形式でベクトル化プログラミングを行うことで、 VP2600/10 向けのチューニングやデバッグが極めて容易になりました。

次の表は、性能評価試験として測定した Fortran プログラムの実行時間 (CPU 時間) の比較です。

計算機	S-4/1000E	M-1800/20U(S)	M-1800/20U(V)	VP2600/10
行列積 512 × 512 (SSL II の DVMGGM 使用)	52 秒 (5MFLOPS)	7 秒 (38MFLOPS)	0.2 秒 (1280MFLOPS)	0.06 秒 (4098MFLOPS)
連立 1 次方程式 1024 元 (SSL II の DVLAX 使用)	42 秒 (16MFLOPS)	20 秒 (34MFLOPS)	0.6 秒 (1064MFLOPS)	0.18 秒 (3831MFLOPS)
Stokes 方程式の有限要素近似解	127 秒	46 秒	2 秒	1 秒
Volterra 型積分方程式の離散近似	2244 秒	509 秒	25 秒	7 秒

M-1800/20U の “(S)” はスカラーモード、 “(V)” はベクトルモードでの実行です。 “S-4/1000E” は新しく導入されたライブラリサーバー (ホスト名 wisdom) での測定結果です。 “FLOPS” とは、 1 秒間に処理される浮動小数点演算回数を表すもので、計算機の性能を計るためによく使われる値です[1]。ただし、どんなプログラムでも数十倍の性能が出るとは限りません。ベクトル化プログラミングについては [2], [5] を参照下さい。

大規模な数値計算を行う場合のプログラム開発の流れは、まず、汎用機 M-1800/20U の対話型処理で十分なデバッグとベクトル化チューニングを行った後、配列規模を拡大し、 VP2600/10 に処理を依頼するという形になります。

[1] VP2600/10 での FLOPS 値が最大処理性能値よりも多少劣っているのは、 512 × 512 や 1024 × 1024 程度の行列では次数が不足して十分な性能が発揮できないためです。次数を増やすに従って最大処理性能に近い値を出すようになります。

1.2 frt, frtex コマンド

Fortran プログラムの翻訳は、以下のコマンドで行います。

言語	コマンド名
FORTRAN77	`frt(/usr/uxp/frt)`
Fortran90	`frtex(/usr/uxp/frtex)`

コマンド (+ オプション) によって起動する Fortran[2]コンパイラは次の名前です。

コマンド	起動システム
`frt`	FORTRAN77 EX V12L10
`frt -J`	FORTRAN77 EX/VP V12L10
`frt -Wv`	FORTRAN77 EX/VP V12L10
`frtex`	Fortran90 V10L10
`frtex -J`	Fortran90/VP V10L10
`frtex -Wv`	Fortran90/VP V10L10

コマンドは M-1800/20U，VP2600/10 共通です。 Fortran プログラムのサフィックスは必ず `.f` として下さい[3]。例えば、 M-1800/20U 上でプログラム `test.f` を翻訳するには、コマンドの後にファイル名を続けて

```
kyu-cc% frt -J test.f  ↵        <--- Fortran77 の場合
kyu-cc% frtex -J test.f  ↵      <--- Fortran90 の場合
```

と入力します。 "-J" はベクトル演算処理を行うことを指示する翻訳オプションです。大規模な配列を宣言し、 DO ループなどの繰り返し処理を多用するプログラムの場合は必ず指定して下さい。

翻訳が正常に終了すると、実行ファイル `a.out` が作成されます。プロンプト (何も設定しなければ `kyu-cc%`) が出た状態で `a.out` を入力することで、プログラムが実行されます。

```
kyu-cc% a.out  ↵              <--- プログラムの実行
          :
     （実行結果）
```

以下、例として専ら `frt` コマンドを用いますが、`frtex` コマンドでも同様です。翻訳オプション、実行時オプションの一覧は付録 A, Bを御覧下さい。また、`man frt` , `man frtex` でも参照できます。

1.3 ベクトル化オプションの指定

ベクトル演算機能を使用するには、翻訳オプションとして `-J` または `-Wv` を指定します。この指定がない場合は、スカラー演算用の Fortran コンパイラが起動されます。`-J` と `-Wv` の違いは、`-Wv` がカンマ (`,`) に続けて細かなベクトルオプションを指定できることに対し、`-J` は標準的なベクトル化だけを指示する点です。

1.4 プログラムが自由形式の場合

Fortran プログラムが自由形式の場合は `-F` オプションを指定します。

[2]新しい規格の Fortran90 から、 "F" が大文字であとは小文字で書き表すことになりましたので、本稿でも真似して "Fortran" と書くことにします。

[3]MSP/Fortran の `.FORT` と異なります。

```
kyu-cc% frt -J -F test.f ↵      <--- Fortran77の場合
kyu-cc% frtex -J -F test.f ↵    <--- Fortran90の場合
```

Fortran90の規格では、新しいプログラム形式が導入され、通常どこから書き始めてもいい"はず"なのですが、富士通のFortran90コンパイラは現在のところ固定プログラム形式[4]が省略値となっています。

1.5 実行ファイル名を変更する

frt, frtexコマンドによって作成される実行ファイルは、省略値で a.outという名前です。実行ファイルにa.out以外の名前を付ける場合は、-oに続いてファイル名を指定します[5]。

```
kyu-cc% frt -J -o test1 test.f ↵      <--- 実行ファイル名の指定
```

例ではFortranプログラムtest.fより実行可能ファイルtest1を作成しました。test1が実行ファイルであるかはfile(/usr/bin/file)コマンドで確認できます。

```
kyu-cc% file test1 ↵
test1:          ELF 32 ビット MSB 実行可能 UXP/M Version 1 use vector unit
                                                  <--- test1に関する情報が表示される
```

1.6 副プログラムを翻訳する

大規模なプログラムを開発する場合には、既にデバッグの完了したサブルーチンや手続き関数などを主プログラムから切り離して管理すると、いちいち翻訳する手間が省けます。例として次を考えます。

ファイル名	中身
main.f	主プログラム
sub.f	サブルーチンプログラム群

sub.fは主プログラムから十分にデバックが完了した副プログラムを取り出したものです。sub.f中にはサブルーチンや関数などが複数個書かれています。

sub.fを翻訳し、main.fから呼ぶことのできる副プログラムのファイル sub.oを作成します。このような、翻訳が終了し主プログラムからの呼び出し・結合を待つだけのファイルをオブジェクトファイルと呼びます。

翻訳オプションは-cを指定します。

```
kyu-cc% frt -J -c sub.f ↵      <--- 副プログラムの翻訳
```

翻訳がうまく終了していれば、オブジェクトファイル sub.oが作成されているはずです。オブジェクトの中身のリストはnm(/usr/ccs/bin/nm)コマンドで出力できます。

```
kyu-cc% nm sub.o ↵          <--- 名前リストの出力

Symbols from sub.o:
```

[4]新JIS Fortran規格では「冗長な旧機能、使わない方がいい機能」に分類されています。

[5]a.outをmvコマンドで名称変更しても構いません。

```
[Index]   Value      Size     Type  Bind  Other Shndx   Name

[1]     |          0|        0|SECT |LOCL |0    |2      |
[2]     |          0|        0|SECT |LOCL |0    |3      |
[3]     |          0|        0|SECT |LOCL |0    |4      |
[4]     |          0|        0|FILE |LOCL |0    |ABS    |sub.f
[5]     |          0|     4200|FUNC |GLOB |0    |2      |setdat_  <--- 副プログラム名
[6]     |       4200|     2672|FUNC |GLOB |0    |2      |mamul_   <--- 副プログラム名
[7]     |       6872|      544|FUNC |GLOB |0    |2      |est_     <--- 副プログラム名
[8]     |          0|        0|NOTY |GLOB |0    |UNDEF  |v_dsin
[9]     |          0|        0|NOTY |GLOB |0    |UNDEF  |f_dsqrt
[10]    |          0|        0|NOTY |GLOB |0    |UNDEF  |f_datan
[11]    |          0|        0|NOTY |GLOB |0    |UNDEF  |f_dsin
[12]    |          0|        0|NOTY |GLOB |0    |UNDEF  |jwe_xalc
[13]    |          0|        0|NOTY |GLOB |0    |UNDEF  |jwe_xdal
[14]    |          0|        0|NOTY |GLOB |0    |UNDEF  |jwe_xvld
```

1.7 分割コンパイル

複数の分割されたファイルを翻訳して、実行ファイルを作成することも可能です。機能ごとにソースファイルを分割して管理するのに便利です。先ほどの例 main.f , sub.f では

```
kyu-cc% frt -J main.f sub.f ↵      <--- 分割したファイルの翻訳
```

と複数のファイルを続けて指定します。sub.o を作成している場合は

```
kyu-cc% frt -J main.f sub.o ↵      <--- 分割したファイルの翻訳
```

と指定します。この様にデバッグの完了した副プログラムをオブジェクトファイルの形で保存することで、翻訳時間が節約できます。

1.8 私用ライブラリの作成

1.8.1 ar コマンド

自分で作成したルーチンで頻繁に使用するものは、私用のライブラリとして管理すると便利です。ライブラリの管理は ar(/usr/ccs/bin/ar) コマンドで行います。ar コマンドで管理されるライブラリ名を「アーカイブファイル」と呼びます[6]。アーカイブファイルの名前は

libexample.a, libEXAMPLE.a

などと、先頭が lib でサフィックスが .a の形式にして下さい[7]。主なオプションは次の通りです。

コマンド	機能
ar rv	アーカイブファイルの作成・ファイルの追加
ar dv	オブジェクトファイルの削除
ar tv	アーカイブファイルの中身を参照

[6] "archive" は文書や情報などを保管する「記録保管所」の意味です。
[7] 他の形式でもできますが、面倒になります。

以下使用例です。まず、新規にアーカイブファイル `example` を作成し、オブジェクトファイル `sub.o` を登録します。

```
kyu-cc% ar rv libexample.a sub.o  ↵      <--- 新規にアーカイブファイル example を作成
a - sub.o                                     sub.o を登録
UX:ar: 情報: libexample.a を作成しています
```

次に、アーカイブファイル `example` にオブジェクトファイル `sub1.o`, `sub2.o` を追加登録します。

```
kyu-cc% ar rv libexample.a sub1.o sub2.o  ↵    <---sub1.o, sub2.o を追加
a - sub1.o
a - sub2.o
```

アーカイブファイルの中を覗いてみます。

```
kyu-cc% ar tv libexample.a  ↵             <--- アーカイブファイルの中身を参照
rw-------   798/   102   4200 May 09 11:13 1996 sub.o
rw-------   798/   102   1456 May 09 11:13 1996 sub1.o
rw-------   798/   102   7080 May 09 11:13 1996 sub2.o
```

オブジェクトファイル `sub1.o` をアーカイブファイル `example` から削除します。

```
kyu-cc% ar dv libexample.a sub1.o  ↵     <---sub1.o を削除
d - sub1.o
```

1.8.2 検索パスの設定

作成したライブラリは、検索パス `LD_LIBRARY_PATH` で参照できるように設定します。そのためには、次のようなステップをふんで下さい。

step1 私用ライブラリを格納するディレクトリ名を決定します。名前は開発プログラムやプロジェクトにあわせて自由に考えて下さい。ホームディレクトリや `tmp` などに置きっぱなしにすると将来必ず後悔します。ここでは例としてホームディレクトリ (`/home/usr9/a79999a`) 下の `MYLIB` だとします。

step2 ディレクトリ `MYLIB` を `mkdir(/usr/bin/mkdir)` コマンドで作成します。

step3 作成したアーカイブファイルを `mv(/usr/bin/mv)` コマンドで`MYLIB` に移動します。

step4 `LD_LIBRARY_PATH` に検索パスを設定します。

```
kyu-cc% setenv LD_LIBRARY_PATH ~/MYLIB  ↵     <-- 検索パスの設定
```

セッションを終了すれば検索パスはキャンセルされます。ただし、この設定では NUMPAC などのライブラリ (`/usr/local/lib`) への検索パスが切れてしまいますので、ホームディレクトリの`.cshrc` に次の一行を追加することをお勧めします。

```
setenv LD_LIBRARY_PATH ".:/home/usr9/a79999a/MYLIB:/usr/lib:/usr/local/lib"
```

“`/home/usr9/a79999a/MYLIB`” の部分が私用ライブラリのあるディレクトリです。

`/usr/lib` と `/usr/local/lib` は標準の Fortran ライブラリが格納されていますので必ず指定下さい。

.cshrc の修正後は

```
kyu-cc% source ~/.cshrc ↵
```

と入力して新しい環境を再定義します。

1.8.3 ライブラリの参照方法

以上の検索パスの設定が済んでいれば、アーカイブライブラリ libexample.a は -l に続けて

```
kyu-cc% frt -J main.f -lexample ↵
```

と指定することで参照され、必要なライブラリが結合されます。

検索パスの設定をしていない場合は、-L に続けてアーカイブライブラリのあるディレクトリをフルパスで記述した後、-l に続けてライブラリ名を指定します。

```
kyu-cc% frt -J main.f -L/home/usr9/a79999a/MYLIB -lexample ↵
```

1.8.4 注意事項

- ベクトル化オプション -J または -Wv を付加して作成したライブラリを結合する場合は、必ず主プログラムも翻訳時に -J または -Wv を指定して下さい。
- 汎用計算機 M-1800/20U とベクトル計算機 VP2600/10 との間では、現在のところライブラリの共有が可能です。しかし、将来汎用機とスーパーコンピュータとの間で浮動小数点形式が異なる可能性が出てくることから、ライブラリを機種によって分けて作成することをおすすめします。

1.9 よく使うオプション集

この節では、frt, frtex コマンドによく付加する翻訳オプションの例をあげます。プログラム名は test.f とします。なお、例はすべてベクトル演算処理を行っています。

1.9.1 ベクトル化のメッセージを出力する

-Wv,-m3 でベクトル化に関する情報が表示できます。何も指定しない場合は情報が端末に表示されますので、-Z オプションで指定したファイルに格納した方がいいでしょう。

```
kyu-cc% frt -Wv,-m3 -Z out test.f ↵      <--- 翻訳結果をファイル out に出力
```

さらに、-Ps オプションによって、プログラムリストに対応したベクトル化情報を得ることができます。これらは、性能解析やチューニングの重要な情報として活用できます。

```
kyu-cc% frt -Ps -Wv,-m3 -Z out test.f ↵
                                    <--- プログラムリストに対応したベクトル化情報を表示
```

1.9.2 最適化オプション

スカラーモードの最適化オプションは基本的な最適化のみの -Ob が省略値です。ベクトルモードのオプション -J または -Wv を指定した場合、自動的に -Oe が省略値になります。-Oe, -Of で翻訳した

場合、極まれにプログラムがエラーを出して終了したり、計算誤差に影響が出たりします。その場合、最適化レベルを調整する必要があります。

```
kyu-cc% frt -Ob -J test.f ↵        <--- 基本的な最適化にとどめる
```

サブオプションで細かい制御も可能です。

```
kyu-cc% frt -Ob,-p -J test.f ↵     <--- 不変式の先行評価の最適化を行う
```

最大限の最適化を行うには -Of を指定します。-Of を指定すると、副プログラムが引用箇所に直接展開され、実行時間の短縮が期待できます。しかし、オブジェクトプログラムの大きさは増大します。また、標準オプションの実行結果とほとんど変わりないこともよくあります。

副作用の生じる可能性のある最適化を行ったことを表示するには、-E オプションを指定します。

```
kyu-cc% frt -J -Eep test.f ↵          <--- -Oe に対応した最適化メッセージを出力
```

```
kyu-cc% frt -Of -J -Eepi test.f ↵     <--- -Of に対応した最適化メッセージを出力
```

1.9.3 デバッグオプション

引数の受渡しや配列と添字がうまく対応しているかなどのチェックは、デバッグオプション -D で行います。思わぬミスが見つかたりして、バグ取りに大変有効です。なお、ベクトルモードでデバッグオプションを指定する場合は、-Wv,-ad でデバッグモードに切替える必要があります。

```
kyu-cc% frt -Daosuv -Wv,-ad test.f ↵     <--- デバッグオプションの指定例
kyu-cc% a.out ↵
```

実行結果にデバッグオプションの指定が反映されます。

実行時の診断メッセージが大量に出る場合は次のように実行時オプションを指定してファイルに書き出します。出力ファイルは out としています。

```
kyu-cc% a.out -Wl,-m6 > out ↵    <--- 実行時の診断メッセージをファイルに出力
```

また、デバッグオプションを指定した場合、最適化やベクトル化は最低限に押えられますので、実行時間は大幅に増加します。そのため、開発中の小規模なプログラムの段階で何度もデバッグオプションを指定し、バグを出しておくことをおすすめします。

1.9.4 サブルーチンライブラリの結合

科学技術計算用のサブルーチンライブラリ SSL II, SSL II/VP, NUMPAC, NUMPAC/VP(4章参照) を利用する場合は、以下のように指定します[8]。

ベクトルモードとスカラーモードでライブラリ名が異なることにご注意下さい。

```
kyu-cc% frt -J test.f -lssl2vp ↵        <--- SSL II/VP を使用
kyu-cc% frt -J test.f -lnumpacvp ↵      <--- NUMPAC/VP を使用
kyu-cc% frt test.f -lssl2 ↵             <--- SSL II を使用
kyu-cc% frt test.f -lnumpac ↵           <--- NUMPAC を使用
```

[8] "/VP" はベクトル版を意味します。

2　ファイル処理

この章では、 UXP/Fortran のファイル入出力の方法について説明します。

2.1　標準入出力

Unix にはリダイレクション (redirection) と呼ばれる便利な機能があります。 Fortran の入出力では装置参照番号[9]5 番が標準入力 (stdin)、 6 番が標準出力 (stdout) に対応しています。

実行ファイル	>	出力ファイル	:	6 番の出力をファイルに上書き
実行ファイル	>>	出力ファイル	:	6 番の出力をファイルに追加書き
実行ファイル	<	入力ファイル	:	5 番の入力をファイルから取り込む

出力ファイルが存在しない場合は、新規に作成されます。以下、具体的な例をあげます。実行ファイルを a.out , 入力データファイルを in.data , 出力データファイルを out.data とします。

2.1.1　ファイル入力

READ(5) に対応するデータが in.data に書かれています。

```
kyu-cc% a.out < in.data  ↵        <---in.data よりデータを取り込む
```

2.1.2　ファイル出力

WRITE(6), PRINT の出力をファイルに書き出します。

```
kyu-cc% a.out > out.data  ↵       <---out.data にデータを上書きする
```

追加書きする場合は >> とします。

```
kyu-cc% a.out >> out.data  ↵      <--- 既存の out.data にデータを追加書きする
```

2.1.3　ファイル入出力

入出力を同時にリダイレクションで指定できます。

```
kyu-cc% a.out < in.data > out.data  ↵
                           <---in.data よりデータを取り込み , 処理結果を out.data に出力
```

このようにリダイレクション機能は使い勝手がとてもいいのですが、方向を間違えると大変なことになりますのでご注意下さい。

2.2　その他の入出力

一般の装置参照番号に対してファイル入出力を行う方法は

[9]論理機番ともいいます。

1. プログラム中に陽にファイルを書き込む
2. 環境変数として設定する

かのどちらかです。

2.2.1 プログラム中にファイルを指定する

open 文中の “file=” に続けてファイル名を ’ ’ で括り指定します。

```
      open(1,file='in.data')
      read(1,100) x,y,z
      close(1)
```

例では、装置参照番号 1 番を in.data に割り当て、データを読み込んでいます。in.data がソースプログラムと同じディレクトリにない場合は、フルパス[10]で指定します。

```
      open(1,file='/home/usr9/a79999a/DATA/in.data')   <--- フルパスでの指定例
```

もちろん、パス名は利用者によって異なります。要はファイルが見えるようにディレクトリが指定できればいいので、指定方法はいろいろあります。

```
      open(1,file='../DATA/in.data')                  <--- ディレクトリの指定例
```

2.2.2 環境変数として設定する

環境変数によって装置参照番号とファイルを対応づけることもできます。環境変数は

fuNN␣ファイル名

です。fu は “fortran unit” の意味です。NN は 2 桁の装置参照番号 (00 ～99) を指定します。ファイル名はパス名込みでも構いません。

装置参照番号 1 番に対してファイル test.data を結合させるには、次のように設定します。

```
kyu-cc% setenv fu01 test.data  ↵       <--- 装置参照番号とファイルとの結合
kyu-cc% a.out  ↵                       <--- プログラムの実行
          :
kyu-cc% unsetenv fu01   ↵              <--- 結合を解除
```

結合の解除は unsetenv で行います。複数のファイル入出力の場合も同様です。なお、バッチリクエストで sh を走らせる場合は設定方法が異なります。 3.2.13節を参照下さい。

2.3 IEEE 形式での入出力

汎用計算機 M-1800/20U とベクトル計算機 VP2600/10 の浮動小数点形式は M 形式と呼ばれるものです。一方、ワークステーションで広く用いられている浮動小数点形式は IEEE 形式と呼ばれるものです。ワークステーションで作成したバイナリーデータを汎用機で処理したり、ベクトル計算機の出力するバイナリーデータをワークステーションで処理する場合には、 M 形式と IEEE 形式の変換をする必要があります。

[10]データファイルがあるディレクトリで pwd とたたけば、フルパス名がわかります。

`frt`, `frtex` で作成された実行ファイルには、バイナリーデータの入出力を IEEE 形式で行う実行時オプションがあります。またライブラリ・サーバー wisdom には、逆に入出力を M 形式で行うオプションがあります。

実行時オプションは `-Wl` のあとにカンマで区切って指定します。

`-C`*uno*	装置参照番号 *uno* からのバイナリーデータの入出力を IEEE 形式で行います。 *uno* の指定がない場合は、すべての装置参照番号を指定したものとします。
`-M`	IEEE ⇔ M 浮動小数点形式の入出力変換後に浮動小数点データの仮数部の一部が損失した場合、診断メッセージを出力します。

以下、例をあげます。

```
kyu-cc% a.out -Wl,-C1,-M ↵          <--- 装置参照番号 1 の入出力を IEEE 形式で行う
         :
```

```
kyu-cc% a.out -Wl,-C,-M ↵           <--- すべての入出力を IEEE 形式で行う
         :
```

なお、浮動小数点形式の違いについての解説としては [6] を御覧下さい。

3　バッチ処理

この章では、記憶領域が 100MB を超える大きなジョブを VP2600/10 に投入するためのバッチ処理の方法について説明します。

3.1　NQS

プログラムの規模がある程度大きくなると、対話型処理ではいつまでたっても結果が返って来なかったり、記憶領域不足で実行ができない場合が出てきます。ベクトル計算機 VP2600/10 では「バッチリクエスト」と呼ばれるシェルスクリプトを M-1800/20U で記述・投入することによって長時間・大規模なプログラムのベクトル処理が可能です。

センターでは、使用できる計算資源に応じてバッチのキュー[11]が設定されています。 1996 年 5 月現在の制限値は以下の通りです。

キュー名	計算機	CPU 時間	記憶域
ss(注意)	M-1800/20U	180 分	100MB
vs	VP2600/10	60 分	100MB
vl	VP2600/10	180 分	100MB
vx	VP2600/10	180 分	400MB

ただし、ss キューは スカラー処理のみ のサポートです。従って、通常のバッチリクエストは VP2600/10 に処理を依頼して下さい。 UXP のバッチ処理システムを総称して NQS(Network Queuing System) と呼びます。 NQS の大きな流れは次のようになります。

処理のコマンドリストをファイルに記述

⇓

qsub コマンドでバッチリクエストを依頼

⇓

qstat, qps コマンドで処理状況を確認

⇓

誤って投入したリクエストは qdel コマンドでキャンセル

⇓

出力結果がファイルとして返送

3.2　バッチリクエストの記述

バッチリクエストはエディターでファイルとして作成します。サフィックスに特別な決まりはありませんが、認識し易いように統一することをお勧めします。ここでは、バッチリクエストファイルはすべて a.vp とします。以下、具体的な例に沿って説明します。

3.2.1　標準的な形

```
#                                  <---csh で記述
cd EXAMPLE                         <--- ディレクトリの移動
frt -Ps -Wv,-m3 test.f             <--- 翻訳
a.out                              <--- 実行
```

[11] queue. 直訳すれば「列」「待ち行列」

先頭の # は、バッチリクエストが csh の文章であることを意味します。# を指定しない場合は sh が走ります。コマンドの羅列のみでしたら、csh と sh に違いはありませんが、それ以上の機能を書き込む場合は文法が異なるので注意が必要です。

次の行の cd は、ディレクトリを ~/EXAMPLE に移動するコマンドです。初期ディレクトリは、バッチリクエストを投入した場所ではなく、利用者の ホームディレクトリ となります。従って、プログラムを翻訳・実行するディレクトリを必ず指定して下さい。

次の行は frt コマンドで test.f を翻訳する処理です。翻訳オプションとして、ベクトル化メッセージとプログラムリストの出力を指示しています。これらはチューニングのための情報として有益なのでできるだけ指定して下さい。最後の行は作成した a.out を実行するコマンドです。

3.2.2 最適化レベルを下げて翻訳・実行

```
#
cd EXAMPLE
frt -Ob -Ps -Wv,-m3 test.f        <--- 翻訳 -Ob の指定
a.out                             <--- 実行
```

最適化を基本的なレベルにとどめます。最適化によって副作用が生じることはありませんので、標準モードと実行結果を比較する場合に有効です。ただし、実行時間が増大するので注意が必要です。

3.2.3 ベクトル化レベルを下げて翻訳・実行

```
#
cd EXAMPLE
frt -Ps -Wv,-an,-m3 test.f     <--- 式の評価順序の変更を抑止
a.out
```

ベクトルモードでは、高速化のために式の評価順序を変更する最適化を行います。その際、丸め誤差の影響から計算誤差が発生することがあります。実行結果の比較は、式の評価順序の変更を抑止するオプションを指定して行います。ただし、こちらも実行時間が増大します。

3.2.4 最大限の最適化を行う

```
#
cd EXAMPLE
frt -Aa -Of -Ps -Eipue -Wv,-p2600,-m3 test.f     <--- 翻訳 . 最大限の最適化
a.out                                            <--- 実行
```

VP2600/10 用の最大限の最適化を指示します。最適化に伴う副作用の可能性を指摘するオプションもあわせて指定します。通常は標準オプション (-J または -Wv) で十分ですが、プログラムによってはかなりの高速化が得られることもあります。

3.2.5 プログラムリストの出力を抑止

```
#
cd EXAMPLE
frt -J test.f                <--- 翻訳
a.out                        <--- 実行
```

十分なデバッグおよびチューニングが済んで、もう翻訳時のベクトル化情報やプログラムリストを見る必要がなくなった時は -Ps , -Wv,-m3 オプションを削除し、標準ベクトル化オプション -J を付加します。

3.2.6 実行ファイルを作成する

```
#                                  <---csh で記述
cd EXAMPLE                         <--- ディレクトリの移動
frt -o b.out -Ps -Wv,-m3 test.f    <--- 翻訳
```

test.f を翻訳して、実行ファイル b.out を作成します。実行はしません。

3.2.7 実行のみ

```
#
cd EXAMPLE
b.out                              <--- 実行のみ
```

既に翻訳が完了している b.out を実行します。このように、翻訳・実行を一つのスクリプトに続けて記述する必要はなく、別個に処理を依頼することもできます。

3.2.8 複数のファイルを分割処理

```
#
cd EXAMPLE
frt -Ps -Wv,-m3 main.f test1.f test2.o       <--- 分割コンパイル
a.out                                        <--- 実行
```

手続きごとに分けて作成したプログラムやオブジェクトファイルも、ファイル名を続けて書くことで実行ファイルが作成できます。

3.2.9 IEEE 形式でファイル処理

```
#
cd EXAMPLE
b.out -Wl,-C,-M                    <---b.out を実行. 入出力は IEEE 形式
```

あらかじめ作成した b.out を実行する際に、実行時のバイナリーデータの入出力を IEEE 形式で行います。ファイル名はプログラムに陽に記述してあるとしています。

3.2.10 標準入力からデータを読み込む

```
#                                  <---csh で記述
cd EXAMPLE                         <--- ディレクトリの移動
frt -Ps -Wv,-m3 test.f             <--- 翻訳
a.out < in.data                    <--- 実行
```

装置参照番号 5 番 (標準入力) からデータを読み込みます。データファイル名は in.data としています。 NQS では対話型のデータ入力ができませんので、必ずファイルを用意して下さい。

3.2.11 標準出力にファイルを指定

```
#
cd EXAMPLE
frt -Ps -Wv,-m3 test.f
a.out > out.data                    <--- 実行
```

装置参照番号 6 番 (標準出力) にデータを書き出す場合も同様です。リダイレクションの方向に注意して下さい。これを指定しない場合は、標準出力ファイルが自動的に作成されます。

3.2.12 標準入出力の例

```
#
cd EXAMPLE
frt -Ps -Wv,-m3 test.f
a.out < in.data > out.data          <--- 実行
```

装置参照番号 5 番にあたるファイル in.data からデータを読み込み、 6 番にあたるファイル out.data にデータを書き出します。

3.2.13 標準入出力以外のファイル処理

```
#                                   <---csh で記述
cd EXAMPLE                          <--- ディレクトリの移動
setenv fu01 ex1.data                <--- 装置参照番号 1 番に ex1.data を割り当てる
frt -Ps -Wv,-m3 test.f              <--- 翻訳
a.out                               <--- 実行
```

ファイル入出力があるプログラムで、ソース中にファイル名指定の open 文がない場合は、上の要領で割り当てます。例では装置参照番号 1 番に ex1.data を対応づけています。unsetenv の必要はありません。

sh の場合は割り当て方法が異なります。バッチリクエストファイルの先頭の “#” がないことに注意して下さい。

```
cd EXAMPLE                          <--- ディレクトリの移動
fu01=ex1.data
export fu01                         <--- 装置参照番号 1 番に ex1.data を割り当てる
frt -Ps -Wv,-m3 test.f              <--- 翻訳
a.out                               <--- 実行
```

3.2.14 複数のファイル処理

複数の装置参照番号を使用する場合は、setenv を続けて指定します。

```
#                                    <---csh で記述
cd EXAMPLE                           <--- ディレクトリの移動
# ------- data allocate ----------
setenv fu01 ex1.data                 <--- 装置参照番号 1 番に ex1.data を割り当てる
setenv fu67 ex2.data                 <--- 装置参照番号 67 番に ex2.data を割り当てる
setenv fu99 ex3.data                 <--- 装置参照番号 99 番に ex3.data を割り当てる
# ------ compile and execute -----
frt -Ps -Wv,-m3 test.f               <--- 翻訳
a.out                                <--- 実行
```

3 行目と 7 行目の # で始まる文はコメントと見なされます。sh の場合も同様です。

3.2.15 作業用ファイルの確保

```
#
cd EXAMPLE
setenv fu01 /tmp/a79999a.work.data <--- 装置参照番号 1 番に a79999a.work.data を割り当てる
frt -Ps -Wv,-m3 test.f
a.out
rm -f /tmp/a79999a.work.data       <--- 作業用ファイルの消去
```

作業用ファイルは、自分の課題のファイルとして確保する他に、/tmp 下に一時的にファイルを作成する方法があります。一般利用者ファイルを参照に行く場合よりも /tmp 下のファイルの参照の方が高速に処理されるため、実行時間の短縮が可能です。ただし、ディスク容量に上限がありますので無制限には確保できません。また、/tmp 下は共用ですので、実行終了後は直ちにファイルを消去して下さい[12]。

3.2.16 SSL II/VP の結合

```
#
cd EXAMPLE
frt -Ps -Wv,-m3 test.f -lssl2vp    <--- 翻訳. SSL II/VP を結合
a.out                              <--- 実行
```

SSL II/VP のサブルーチンを使用している場合は、-lssl2vp を指定します。-lssl2 と指定するとスカラー版のライブラリが呼ばれます。

3.2.17 NUMPAC/VP の結合

```
#
cd EXAMPLE
frt -Ps -Wv,-m3 test.f -lnumpacvp  <--- 翻訳. NUMPAC/VP を結合
a.out                              <--- 実行
```

NUMPAC/VP のサブルーチンを使用している場合は、-lnumpacvp を指定します。-lnumpac と指定するとスカラー版が呼ばれます。

[12] alias として rm を rm -i に再設定している場合は作業用ファイルは消去できません。その場合はジョブが終了した段階で rm コマンドでの消去をお願いします。また、/tmp 下のファイルは定期的に自動消去されます。

3.2.18　私用ライブラリの結合

```
#
cd EXAMPLE
frt -Ps -Wv,-m3 test.f -lmylib     <--- 翻訳．私用ライブラリを結合
a.out                               <--- 実行
```

私用ライブラリ libmylib.a を結合します。検索パス LD_LIBRARY_PATH で参照できるように .cshrc を編集しておく必要があります (1.8.2節参照)。検索パスを通していない場合は、フルパスで例えば

```
#
cd EXAMPLE
frt -Ps -Wv,-m3 test.f -L/home/usr9/a79999a/MYLIB -lmylib
a.out
```

などと指定します。

3.2.19　CPU 時間の計測

timex(/usr/bin/timex) コマンドを用いると、翻訳時間や実行時間の経過時間、 CPU 時間、ベクトルユニット占有時間がそれぞれ計測でき、チューニングに必要な情報が得られます (6章参照)。

```
#
cd EXAMPLE
timex frt -Ps -Wv,-m3 test.f      <--- 翻訳．CPU 時間等をモニター
timex a.out                       <--- 実行．CPU 時間等をモニター
```

timex の情報は標準エラー出力に書き出されます (3.3節参照)。

3.2.20　複数のリクエストの記述

一つのバッチリクエストに複数のプログラム処理を記述することも可能です。ただし、実行ファイル名は a.out 以外に変えた方がいいでしょう。

```
#
cd EXAMPLE
frt -o b.out -Ps -Wv,-m3 test1.f ↵     <--- test1.f を翻訳
b.out ↵                                <--- 実行
frt -o c.out -Ps -Wv,-m3 test2.f ↵     <--- test2.f を翻訳
c.out ↵                                <--- 実行
```

3.3　バッチリクエストの投入

3.3.1　qsub コマンド

前節の要領で作成したバッチリクエストファイルの処理は qsub(/usr/bin/qsub) コマンドで依頼します。形式は

qsub␣*options*␣*scriptfile*

です。 *options* はオプションを空白で区切って指定します。 *scriptfile* はバッチリクエストのスクリプトファイル名です。qsub コマンドには多様なオプションが存在しますが、よく使うオプションは次の通りです。詳細は man qsub を参照下さい。

-q␣*queue*	*queue* はキュー名です。省略すると汎用機の ss キュー (スカラー処理) に投入されます。
-mb	バッチリクエストの実行開始をメールで通知します。
-me	バッチリクエストの実行終了をメールで通知します。
-mi	バッチリクエストの統計情報をメールで通知します。
-e␣*errfile*	標準エラー出力 (翻訳結果，timex の情報など) を指定したファイル *errfile* に出力します。指定がない場合は、リクエストを投入したディレクトリ下に「リクエスト名.e リクエスト番号」という名前のファイルが自動的に作成されます。
-o␣*outfile*	標準出力を指定したファイル *outfile* に出力します。指定がない場合は、リクエストを投入したディレクトリ下に「リクエスト名.o リクエスト番号」という名前のファイルが自動的に作成されます。
-eo	標準エラー出力を標準出力と同じファイルに出力します。

メールでの通知オプションとは別に、利用者課金システムよりバッチリクエストの実行に要した課金情報が実行終了後メールで送付されます。

3.3.2 バッチリクエストの投入例

以下、バッチリクエストの記述されたファイル名を a.vp とします。

```
kyu-cc% qsub -q vs a.vp  ↵                  <--- バッチリクエストの投入
Request 5466.kyu-cc submitted to queue: vs.
```

VP2600/10 の vs キューに投入しました。下線部の “5466” がリクエスト番号、 “kyu-cc” がホスト名です。 “5466.kyu-cc” でリクエスト名を構成します。実行が終了すると、スクリプトファイル名とリクエスト番号に対応した標準エラー出力ファイル a.vp.e5466 と標準出力ファイル a.vp.o5466 が返ってきます。

```
kyu-cc% ls  ↵
test.f    a.out       a.vp        a.vp.e5466  a.vp.o5466
```

```
kyu-cc% qsub -q vl -o out.data -eo a.vp  ↵              <--- バッチリクエストの投入
Request 5467.kyu-cc submitted to queue: vl.
```

次は VP2600/10 の vl キューへの投入例です。標準出力ファイル名を out.data に、また標準エラー出力を標準出力と同じファイル out.data に出力します。実行が終了すると、標準エラー出力と標準出力が同一のファイル out.data に格納されて返ってきます。

```
kyu-cc% ls  ↵
test.f    a.out       a.vp        out.data
```

```
kyu-cc% qsub -q vx -me -mi a.vp  ↵              <--- バッチリクエストの投入
Request 5467.kyu-cc submitted to queue: vx.
```

VP2600/10 の vx キューへの投入しました。更に実行終了と統計情報をメールで知らせることを指定しています。なお、 UXP の電子メールの利用方法は [10] を御覧下さい。

3.4 ジョブの状態表示

qsub コマンドにより依頼したバッチリクエストの処理状況は、qstat(/usr/local/bin/qstat) コマンド、および qps(/usr/local/bin/qps) コマンドによって調べることができます。

3.4.1 qstat コマンド

qstat コマンドはバッチリクエストの処理状況をキューごとに出力します。形式は

qstat␣@*machine*

です。*machine* は計算機のマシン名を指定します。VP2600/10 のマシン名は vpux です。"@*machine*" を指定しないと、M-1800/20U の処理状況が出力されます。VP2600/10 の処理状況が見たい場合は 必ず @vpux オプションを指定下さい。

```
kyu-cc% qstat @vpux  ↵     <--- VP2600/10の処理状況を表示
vs@kyu-vpux;  type=BATCH;  [ENABLED, RUNNING];  pri=63
  0 exit;   2 run;   2 queued;   0 wait;   0 hold;   0 arrive;
  0 freezing;   0 frozen;

         REQUEST NAME        REQUEST ID           USER PRI    STATE BAT-ID  PGRP
    1:           a.vp   5472.kyu-cc           a79999a  31  RUNNING     14  2295
    2:           b.vp   5473.kyu-cc           a79999a  31  RUNNING     15  2338
    3:           c.vp   5474.kyu-cc           a79999a  31   QUEUED
    4:           d.vp   5475.kyu-cc           a79999a  31   QUEUED

vl@kyu-vpux;  type=BATCH;  [ENABLED, INACTIVE];  pri=63
  0 exit;   4 run;   0 queued;   0 wait;   0 hold;   0 arrive;
  0 freezing;   0 frozen;


vx@kyu-vpux;  type=BATCH;  [ENABLED, INACTIVE];  pri=63
  0 exit;   2 run;   1 queued;   0 wait;   0 hold;   0 arrive;
  0 freezing;   0 frozen;
                       :
```

例では、vs キューで 2 つのジョブが実行中であり、2 つが実行を待っていることを示しています。また、他のジョブキューの混み具合もモニターできます。

```
kyu-cc% qstat  ↵     <--- M-1800/20Uの処理状況を表示
ss@kyu-cc;  type=BATCH;  [ENABLED, RUNNING];  pri=63
  0 exit;   1 run;   0 queued;   0 wait;   0 hold;   0 arrive;
  0 freezing;   0 frozen;

         REQUEST NAME        REQUEST ID           USER PRI    STATE BAT-ID  PGRP
    1:           a.vp   5470.kyu-cc           a79999a  31  RUNNING      3  9242
                     :
```

M-1800/20U の処理状況を調べるためには @*machine* を指定しません。

3.4.2 qps コマンド

qps は、実行中のバッチリクエストの進行状況を見るコマンドです。

```
kyu-cc% qps ↵      <--- 進行状況をモニター
    UID    PID   PPID    STIME    REQ-ID  TIME COMMAND
a79999a   2606   2601  16:37:24    5477   0:00 /bin/csh /usr/spool/nqs/scripts/2589
   root   2590   2589  16:37:17    5477   0:00 -utms_sh
a79999a   2601   2590  16:37:24    5477   0:00 -csh
a79999a   2619   2606  16:37:33    5477   1:23 a.out
kyu-cc%
```

例では a.out が 1 分 23 秒走っています。

3.5 リクエストのキャンセル

3.5.1 qdel コマンド

誤って投入したバッチリクエストや、これ以上実行する必要ないと判断したジョブのキャンセルは qdel(/usr/bin/qdel) コマンドで行います。形式は

qdel␣*options*␣*request-id*

です。 *options* はオプションの並び、 *request-id* はリクエスト名です。リクエスト名は、qstat コマンドの REQUEST ID で確認することができます (3.4.1節の例参照)。qdel コマンドのオプションは次の通りです。

-k	実行中のバッチリクエストに対してキャンセルを行います。-k の指定がない場合は、実行待ちのリクエストだけをキャンセルします。
-r kyu-vpux	VP2600/10 に投入したバッチリクエストのキャンセルを行います。指定がない場合は、 M-1800/20U 上のバッチリクエストをキャンセルします。

3.5.2 リクエストのキャンセル例

```
kyu-cc% qdel -r kyu-vpux 5478.kyu-cc ↵    <---VP2600/10 の実行待ちリクエストをキャンセル
Request 5478.kyu-cc has been deleted.
```

実行中の場合は -k オプションの指定が必要です。

```
kyu-cc% qdel -k -r kyu-vpux 5479.kyu-cc ↵ <---VP2600/10 の実行中のリクエストをキャンセル
Request 5479.kyu-cc is running, and has been signalled.
```

3.6 NQS の注意点

バッチリクエストを投入した段階で、スクリプトに記述した Fortran プログラムおよびデータファイルの修正は絶対にしないで下さい。 NQS はバッチリクエストが投入された段階でのファイル群のコピーや排他的な処理をしません。従って、リクエストを投入した後でこれらのファイルを修正すると、修正後のファイルを翻訳・実行することがあり、全く異なる結果や異常終了を起こす可能性があります。

その場合、別なディレクトリにコピーを持つか、同じディレクトリの場合でもバッチリクエストを投入する前の段階で別な名前のコピーを持つなどのファイル管理をお願いします。

また、もしバッチリクエストで指定したディレクトリに以前作成した実行ファイル (省略名 a.out) が残っている場合は、ジョブ投入前に必ず名前を変えるか消去することをお勧めします。もし、翻訳の段階でエラーとなり、実行ファイル (省略名 a.out) が作成できなかった場合、以前の実行ファイルが呼ばれ、全く無駄な計算が行われる危険があるためです。

4 サブルーチンライブラリの利用

4.1 ベクトル計算機用ライブラリ

UXP/Fortran では、汎用サブルーチンライブラリとして SSL II , NUMPAC をサポートしています。さらに、ベクトル計算機向けにチューニングされた SSL II/VP , NUMPAC/VP を利用することで、格段のスピードアップが可能です。例として VP2600/10 で測定した連立 1 次方程式 (Gauss 消去法・密行列) に要する実行時間を自作サブルーチン ([11]) と比べてみると

NUMPAC/VP	2.2 倍
SSL II/VP	6.1 倍

の性能差が出ます。

これらのサブルーチンライブラリを使用する利点は

- ベクトル計算機のハードウェアの能力を最大限に活かすアルゴリズムを用いており、一般の汎用ライブラリに比べて圧倒的に高速である。
- 流体力学・構造解析・分子化学・核融合などの分野の大規模数値シミュレーションでは、連立 1 次方程式・固有値計算・フーリエ変換などの計算に要するコストがプログラム全体のかなりの部分を占め、この部分を高速化することが全体のスピードアップになることが多い。
- 精度についても十分な考慮がなされており、信頼性が高い

などが挙げられます。ただし機能の点では、連立 1 次方程式の反復解法[13]がサポートされてないなど、必ずしも利用者の要求に全面的に応えているとはいえません。

4.2 移植性の心配

また、サブルーチン部分がブラックボックスになるため、移植性などの面から抵抗ある方もいらっしゃると思います。しかし、ライブラリはソースが入手可能な Lapack や他のベクトル計算機の Fortran システムなどで代替可能です。さらに、サブルーチンの入れ換えも引数の若干の修正で済む場合がほとんどです。従って、移植性の心配はしなくていいと思います。

逆に、ワークステーション上で開発したプログラムをベクトル計算機に移植する場合は、SSL II/VP や NUMPAC/VP に該当する機能を積極的に入れ換えることで、かなりの高速化が実現できます[14]。

4.3 マニュアルの参照

4.3.1 オンラインマニュアル

SSL II , NUMPAC のマニュアル [7], [8], [9] はセンター 2 階のプログラム相談室、 4 階の図書室で閲覧できます。

また、 SSL II のマニュアルはオンラインでも参照できます。機能一覧は `man ssl2` です。また個別のサブルーチンについては、`man ssl2` で確認した後、例えば `man dvlax` , `man vmggm` などで機能・引数等が検索できます。

[13] もっとも、反復解法はもともとの離散モデルに大きく依存するので、汎用性のあるパッケージ (要するに売りもの) としては提供しにくい面があります。

[14] さらに進んで、高速化のためにはベクトル計算機用のライブラリが使用できるようにプログラムを書き換えることも必要になるでしょう。

4.3.2 ソースプログラムの閲覧

NUMPACのソースは非公開です。 SSL II のソースはベクトル計算機用の拡張機能版を除き MSP 上のデータセットにコピーすることができます。ただし、コピーしたサブルーチン群を九州大学大型計算機センター以外の計算機システムで使用することは禁止されています。

サブルーチンのコピーは MSP の `LIBCOPY` コマンドで行います。形式は

`LIBCOPY␣`*subroutine*`␣DATASET(`*dataset*`)`

です。 *subroutine* は SSL II のサブルーチン名を、 *dataset* は保存する MSP の区分データセット名を指定します。 *dataset* のメンバー名 *subroutine* として SSL II のサブルーチンがコピーされます。

```
 READY
LIBCOPY DLAX DATASET(SSL2) ↵     <---DLAX を SSL2 のメンバーとしてコピー
```

4.4 ライブラリの結合方法

4.4.1 SSL II の結合

ライブラリの組み込みは翻訳時 (`frt`, `frtex`) に行います。ベクトルモードで利用する場合は `-lssl2vp` ，スカラーモードの場合は `-lssl2` と指定します。バッチリクエストの場合も全く同じです。

```
kyu-cc% frt -J test.f -lssl2vp ↵        <--- SSL II/VP を使用
kyu-cc% frt test.f -lssl2 ↵             <--- SSL II を使用
```

なお、 SSL II の機能一覧は付録 Cを参照下さい。

4.4.2 NUMPAC の結合

ライブラリの組み込みは翻訳時 (`frt`, `frtex`) に行います。

ベクトルモードで利用する場合は `-lnumpacvp` ，スカラーモードの場合は `-lnumpac` と指定します。バッチリクエストの場合も全く同じです。

```
kyu-cc% frt -J test.f -lnumpacvp ↵    <--- NUMPAC/VP を使用
kyu-cc% frt test.f -lnumpac ↵         <--- NUMPAC を使用
```

なお、 NUMPAC の機能一覧は付録 Dを参照下さい。

5　プログラムのデバッグ

この章では、UXP/Fortran におけるデバッグのヒントを簡単に説明します。

5.1　プログラミングの基本

まず、大規模なプログラムを作成する際に気をつける点を挙げます。

5.1.1　副プログラム単位に分割する

たまに、メインプログラムにすべての処理が記述してあるプログラムを見ます。しかし、これは以下の点からよろしくありません。

- 境界条件や初期値を変更したり、一部の計算の有効性を確かめたりする場合、プログラム全体を見渡す必要がある。
- 大量の配列や変数が錯綜し、バグの潜む危険が増大する。
- デバッグ用に `WRITE` 文などを挿入しても、プログラム全体について翻訳・実行しなければならず、効率がよくない。
- アルゴリズムの流れを把握することが難しく、他人に理解してもらいにくい。何よりも、後日自分すら解読できなくなる可能性がある。

機能ごとに副プログラムや関数を作成し、主プログラムは全体を統括する方がプログラムも見易く、デバッグも効率的に行えます。

5.1.2　プログラムサイズが変更可能なように作成する

これはデバッグ、チューニング両面から非常に大事です。プログラム全体の規模が、あるパラメータ (例えば領域の分割数など) をひとつ修正することで自由に変更できるならば、小規模なプログラムでのデバッグ、チューニングが繰り返し可能となります。また、パラメータを変えた場合のデータの比較により、予期しなかったアルゴリズムの問題点が発見できたりします。何よりもプログラムに一般性が備わり、他のプログラムへの使い回しも容易になります。

5.1.3　できるだけ詳しいコメントをつける

最低限、数カ月後の自分が解読できるようにコメントをつけるべきです。配列・変数・定数のひとつひとつに「これは××のための宣言である」とか、DO ループ毎に目的・機能を詳しく記しておくと (プログラム開発中はわかりきっていることでも) 後々非常に助かります。もちろん、デバッグにも有効です。

5.2　異常終了する場合

実行時にシステムの割り込みによって、実行が強制的に打ち切られることがあります。その場合 `jwe00` で始まる診断メッセージが出力されます。

```
jwe0019i-u The program was terminated abnormally with signal number SIGSEGV.
           program name=ex4
           program location=00000230
           signal code=SEGV_SEGERR(10)     psw=071d00008000033e
           instruction=7820d058     location=0000033a
           execution mode=advanced
           VP hardware kind option=VP2200
```

ベクトル演算での実行モードの場合、特にオプションを指定しない時は高速に走行させることを目的とした「高速モード」となっています。強制終了の原因調査は、デバッグのため実行文の出現順序に従って実行する「デバッグモード」で行います。デバッグモードへの切替えオプションは `-Wv,-ad` です。

以下、強制終了の場合の原因と対処方法を述べます。

5.2.1 ゼロ割り

`jwe0013i` のメッセージは、浮動小数点演算でゼロ割りが発生したことを告げています。

```
jwe0013i-e A floating division exception was detected. psw=071d200080000344
           execution mode=noadvanced
           instruction=fe3d40806000      location=0000033e
```

除算の部分を特定してゼロ割りを回避するようにプログラムを修正して下さい。

5.2.2 指数オーバーフロー

`jwe0011i` のメッセージは、浮動小数点演算の結果の絶対値が 10^{63} を超えたことを告げています。

```
jwe0011i-e A floating overflow exception was detected. psw=071d2000800002fe
           instruction=7c08a120          location=000002fa
```

UXP/Fortran システムはこれ以上の数を扱えませんので、正規化などプログラムの再検討が必要です。

5.2.3 指数アンダーフロー

`jwe0012i` のメッセージは、浮動小数点演算の結果の絶対値が 10^{-65} よりも小さくなったことを告げています。

```
jwe0012i-e A floating underflow exception was detected. psw=071d2200800002fe
           instruction=7d08a120          location=000002fa
```

ただしこのメッセージは、実行時オプションとして `-Wl,-u` を指定した時のみ有効です。

```
kyu-cc% a.out -Wl,-u    <--- 指数アンダーフローの検出を指定
```

`-Wl,-u` の指定がない場合は、指数部を少しだけ大きくしたほとんどゼロに近い値として処理を続行します。極めて微妙な収束判定などのプログラムで、納得できない結果となる場合は `-Wl,-u` を指定し、桁あふれを確認してみて下さい。

5.2.4 整数演算オーバーフロー

jwe0015i のメッセージは、整数型演算においてオーバーフローが生じたことを告げています。

```
jwe0015i-e A fixed overflow exception was detected. psw=071d3800800002f6
           instruction=8fe00020          location=000002f2
```

ただし、このメッセージは、翻訳時にデバッグオプション -Do を、さらに実行時オプションとして -Wl,-o を指定した時のみ有効です。

```
kyu-cc% frt -Do -Wv,-ad test.f  ↵   <--- デバッグオプションの指定
kyu-cc& a.out -Wl,-o  ↵             <--- オーバーフローの検出を指定
```

5.2.5 領域外へのアクセス

参照または書き込みが禁止された領域にアクセスが発生した場合、jwe0019i のメッセージを出して実行が打ち切られます。エラーの理由は、宣言した配列の範囲外の記憶領域に対する引用が生じたためです。

調査方法は、添字式の値を検査するデバッグオプション -Ds を指定して翻訳・実行します。

```
kyu-cc% frt -Ds -Wv,-ad test.f  ↵   <--- 添字式の値の検査するオプション
kyu-cc% a.out  ↵                    <--- 調査
            :
```

添字の値がおかしい時は、その旨のメッセージが出力されます。

5.2.6 引数の型の不一致

サブルーチンなどで仮引数と実引数の矛盾があると、jwe0019i または jwe0010i のメッセージが出る場合があります。

調査方法は、仮引数と実引数の矛盾を検査するデバッグオプション -Da を指定して翻訳・実行します。

```
kyu-cc% frt -Da -Wv,-ad test.f  ↵   <--- 引数の矛盾を検査するオプション
kyu-cc% a.out  ↵                    <--- 調査
            :
```

5.2.7 未定義データの引用

未定義の変数をプログラムで引用した時、jwe0019i または jwe0010i のメッセージが出る場合があります。

調査方法は、未定義データの引用を検査するデバッグオプション -Du を指定して翻訳・実行します。

```
kyu-cc% frt -Du -Wv,-ad test.f  ↵   <--- 未定義データの引用を検査するオプション
kyu-cc% a.out  ↵                    <--- 調査
            :
```

5.2.8 最適化による副作用

最適化オプション -Oe(ベクトルモードの標準値)，-Of または -Ob,-p を指定した場合、不変式の先行評価の最適化によってアルゴリズム上実行されないはずの命令が実行されエラーになる場合があります。エラーは

- 配列要素の添字の値が宣言の範囲を飛び越える
- 組み込み関数の引用で引数に定義外の値が入る
- ゼロ割りが発生する

場合がほとんどです。

この最適化の副作用の可能性は、翻訳時に -Ep オプションを付加することで出力できます。

```
kyu-cc% frt -Ep -J test.f  ↵   <--- 不変式の先行評価のメッセージを出力
fortran77 ex/vp   diagnostic messages: program name(test)
jwd8220i-i                Optimization with possibility of side effect.
jwd8201i-i isn:00000008   Invariant expression was moved.
```

対処方法として、不変式の先行評価の最適化を抑止するか、エラーとならないようにプログラムを修正します。

```
kyu-cc% frt -Oe,-P -J test.f  ↵   <--- 不変式の先行評価を抑止
```

5.3 実行結果があやしい場合

エラーメッセージは出ていないが、実行結果が信用できない場合を考えます。デバッグはプログラムに完全に依存しますので、決定的な手段はありません。最後はプログラマーの冷静な観察眼が頼りです。

5.3.1 プログラムリストを見る

節目節目でプログラムリストをプリントアウトして、赤ペン片手に眺めることが、やはりもっとも有力なデバッグ方法です。エラーの発見以外にも、効率的な手法や工夫を発見できることがよくあります。

5.3.2 write 文の挿入

対話型処理では、 write 文を怪しい箇所に埋め込み結果を見ることが容易にできます。行列の表示用や対称性のチェック、誤差ノルムの計算などのサブルーチンを幾つか作っておくのもいいでしょう。

5.3.3 デバッグオプションの指定

デバッグオプションは引数の矛盾、添字の値、未定義データの引用、重複した引数の値などをチェックするデバッグの有力な手段です。ただし、デバッグモードでの実行となるので、通常の高速モードに比べて実行時間が増加します。従って、デバッグオプションはプログラム規模が小さい段階で行って下さい。

```
kyu-cc% frt -Dasuv -Wv,-ad test.f  ↵   <--- デバッグオプションの指定
kyu-cc% a.out  ↵
```

例では、デバッグオプション -Da, -Ds, -Du, -Dv を指定しました。また、実行モードもデバッグモードに変更します。

5.3.4　最適化による副作用

最適化オプション -Oe(ベクトルモードの標準値) ,-Of または -Ob,-e を指定した場合、極まれにですが、演算評価方法の最適化によって計算誤差が生じることがあります。

この最適化の副作用の可能性は、翻訳時に -Ee オプションを付加することで出力できます。

```
kyu-cc% frt -Ee -J test.f ↵    <--- 演算評価方法の最適化に関するメッセージを出力
fortran77 ex/vp   diagnostic messages: program name(setdat)
jwd8220i-i                Optimization with possibility of side effect.
jwd8206i-i isn:00000050   Division was changed to a multiplication.
```

不審の場合、演算評価方法の最適化を抑止するサブオプション -Oe,-E または -Of,-E を指定・実行し、計算誤差を確認して下さい。

```
kyu-cc% frt -Oe,-E -J test.f ↵    <--- 演算評価方法の最適化を抑止
```

特に最適化オプションとして -Of を指定する場合は、一度全ての -E オプションを付加して診断メッセージをチェックすることをお勧めします。

```
kyu-cc% frt -Of -Ecelmipu -J test.f ↵
```

5.3.5　ベクトル用最適化による副作用

ベクトルモードの式の評価順序を変更する最適化によって計算誤差が発生することがあります。その可能性がある場合は、式の評価順序を変更しないオプションを指定して結果を比較します。ただし、実行時間は増大するので注意が必要です。

```
kyu-cc% frt -Wv,-an test.f      <--- 式の評価順序の変更を抑止
```

6　プログラムのチューニング

この章では、ベクトル計算機に適したプログラムへのチューニング方法について説明します。

6.1　ベクトル処理とは

6.1.1　ベクトルデータ

変数や配列の要素などの一つのデータをスカラーデータといいます。これに対して、配列全体や配列の部分など複数のスカラーデータから構成されるデータをベクトルデータといいます。

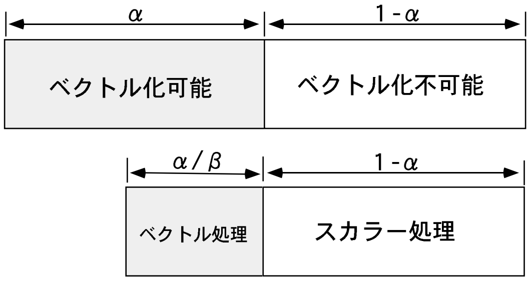
ベクトル処理とは、 DO ループで表現された演算に対し、配列をベクトルデータとして処理する方法です。具体的には、

1. データをベクトルとして連続的にメモリーから取り出し
2. 同一の演算をベクトルデータに対し連続的に実行し
3. 結果を再度連続的にメモリーに書き込む

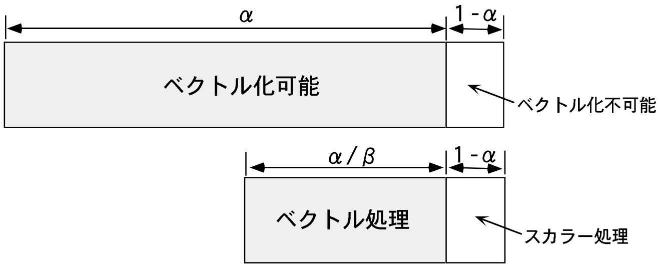
処理のことです。

6.1.2　どれくらい高速化できるか

では、手元のプログラムがどれくらい高速に実行されるかを考えます。あるプログラムをスカラーモードで実行した実行時間を 1 とし、そのうちの α がベクトル処理に置き換え可能だとします。また、α の部分をベクトル処理したとき β 倍の計算速度が得られるとします。 α をベクトル化率、 β を加速率といいます。 α , β の値はプログラムによって異なります。



ベクトル化率と加速率の関係は上図のようになります。図の上段がスカラーモードでの実行時間、下段がベクトルモードでの実行時間です。同じプログラムをベクトルモードで実行した場合、 α の部分は加速率 β によって実行時間が $1/\beta$ となりますが、その他の $1-\alpha$ 部分はスカラー処理のままですので、実行時間は同じです。つまり、いくら β が大きくても、ベクトル化できる部分が少ないと全体の高速化は望めないということです。



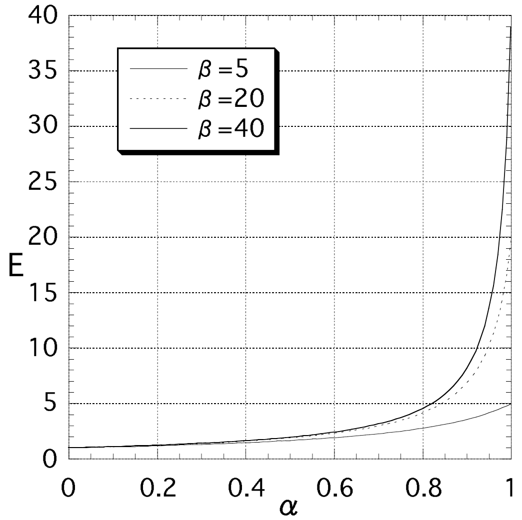
プログラムでベクトル化できる部分が多ければ多いほど β による加速が大きく作用し、全体の高速化が実現できます。

6.1.3 実行性能向上比

スカラーモードでの実行時間とベクトルモードでの実行時間の比を実行性能向上比といいます。これを E とおくと、上の図より実行性能向上比[15]は

$$E = \frac{1}{\alpha/\beta + (1-\alpha)}$$

で与えられます[16]。実行性能向上比 (E) とベクトル化率 (α) および加速率 (β) の関係をグラフにします。



これを見ると、α が 0.9 つまりベクトル化率が 90% を超えるあたりから加速率 β の寄与が大幅に増加します。つまり、ベクトル化率が低いのにいくら頑張ってチューニングしてもあまり効果は期待できませんが、ベクトル化率が 90% を超えたプログラムに対して加速率を数 $\%$ 上げると大幅な実行性能向上が得られることになります。

プログラムの高速化のためには、まずベクトル化できる部分をできるだけ増やし α の値を 1 に近付けた上で、加速率 β を上げることを考えます。

6.1.4 ベクトル化率，加速率の求め方

ベクトル化率 α と加速率 β の値は次の式で概算できます。

$$\alpha \approx \frac{S-(CPU-VPU)}{S}, \qquad \beta \approx \frac{S-(CPU-VPU)}{VPU},$$

S : スカラーモードでの全 CPU 時間
CPU : ベクトルモードでの全 CPU 時間
VPU : ベクトルモードでのベクトルユニットの占有時間

ただし、一般にスカラーモードでは実行時間が大幅に増えるため、わざわざベクトル化率の測定のためだけにスカラーモードで実行するのは現実的ではありません。その場合、ベクトルモードで実行した時の全 CPU 時間に対するベクトルユニット占有時間の比

$$VPU/CPU$$

[15] 相対処理速度ともいいます。
[16] この式をアムダールの法則といいます。

をベクトル化率の目安として下さい[17]。

CPU, VPU の値はプログラム全体ならば tiemx コマンド、サブルーチンや DO ループ単位ならば CLOCKV サブルーチンを用いて計測することができます (6.7, 6.8節参照)。

6.1.5　ベクトル化が期待できるプログラム

ベクトル演算による高速化が期待できるプログラムは

- 大規模な配列を用いて
- DO ループによる演算が多い

ものに限られます。ベクトル化は DO ループに限られ、IF/GOTO 文 , DO WHILE 文 , DO UNTIL 文などはベクトル化されません。

加速率 β は計算機の性能だけでなく、プログラムの性質によって変化します。一般に次のプログラムは β が大きくなることが期待されます。

- ベクトルデータの要素数 (ベクトル長) が多いほど、一括して処理できる配列データの数が増え、処理効率が増加します。ベクトル長は、 DO ループの回転数を超えることはありません。従って DO ループの回転数が多いプログラム[18] 程、加速率のアップが期待できます。

  また、ベクトル長が極端に短いとスカラーモードでの実行の方が速いことがあります。ベクトル化率は高いのにあまり高速化されない場合は、 DO ループの回転数をチェックしてみて下さい。

- 配列データの引用はできるだけ連続した方が効率よく処理されます。配列の処理はできるだけ列方向の計算を行った方が連続的なメモリーアクセスとなります。
- 計算機資源の有効利用の観点からは、 DO ループ中の演算子が多い方が高速化が期待できます。

6.1.6　ベクトル化されるデータの型

ベクトル化の対象となるデータ型は次の通りです。

ベクトル化される	ベクトル化されない
REAL*4 REAL*8	REAL*16
COMPLEX*8 COMPLEX*16 LOGICAL*4	COMPLEX*32 LOGICAL*1
INTEGER*4	INTEGER*2 INTEGER*8

注意していただきたいのは、 4 倍精度データはベクトル化の対象外という点です。

6.1.7　ベクトル化メッセージ

プログラムの DO ループがベクトル化されたかどうかは、-Wv,-m3 オプションで、また、ベクトル化メッセージ付きのソースプログラムを -Ps の指定で見ることができます (1.9.1節, 3.2節参照)。

[17]この比を VU 率といいます。もちろんスカラーモードとの相対比を求めるにこしたことはありません。

[18]わかりやすい言葉を使えば「ぶんまわす」

```
00002301      s        DO 140 K=1,N-1
00002302      v         M=MAX(K)
00002303      m         T=X(M,J)
00002304      s         X(M,J)=X(K,J)
00002305      s         X(K,J)=T
00002306      s         IF(T.EQ.0.D0) GO TO 140
00002307      v         DO 130 I=K+1,MIN0(K+LB-1,N)
00002308      v    130  X(I,J)=X(I,J)+A(LB+K-I,I)*T
00002309      v    140 CONTINUE
```

記号は DO 文に対する記号とそれ以外の文に対する記号があります。

v	DO ループに含まれる文が、すべてベクトル化された。
m	DO ループに含まれる文に、ベクトル化されたものとされなかったものが混在している。
s	DO ループに含まれる文は、ベクトル化されなかった。

DO 文に対する表示記号

v	この文はベクトル化された。
m	この文にはベクトル化されたものとされなかったものが混在している。
s	この文はベクトル化されなかった。
␣	ベクトル化対象外。

DO 文以外に対する表示記号

DO ループがベクトル化可能であっても、計算機の性能上ベクトル化しない方が得策とコンパイラが判断した場合、 DO 文には s を、またループ内の文については診断メッセージをそのまま出力します。

また、表示記号の後に、数字がくっついていることもあります。

```
00002057      s3      DO 300 J=1,3
00002064      v3         LOOEZZ(I,J)=LOOEZZ(I,J)+SEGZZ*GGGG
00002065      v3         LOOERR(I,J)=LOOERR(I,J)+SEGRR*GGGG
00002066      v3         LOOEZR(I,J)=LOOEZR(I,J)+SEGZR*GGGG
00002070      v3  300 CONTINUE
```

この数字はループアンローリング[19]によって展開された数を意味します。例では 3 回のアンローリングが行われています。

6.2 ベクトル化の例

配列に対する DO ループがすべてベクトル化される訳ではありません。ベクトル化される文は限られています。ここでは、どのような DO ループがベクトル化されるのか、具体的な例をあげながら説明します。

6.2.1 代入文

```
00000015      v        DO 10 I=1,N
00000016      v         C(I)=B(4*I+1)+A(I)
00000017      v   10   CONTINUE
```

よくある形の代入文です。配列の添字に DO 変数の演算が入っていてもベクトル化されます。

[19] DO ループの繰り返し回数を減らす最適化のこと。ループ内の実行文を n 重に展開することでループの回転数を $1/n$ に削減する ([1])。

6.2.2 条件つき計算

```
v        DO 10 I=1,N
v         IF(A(I).GT.0.5D0) THEN
v          C(I)=B(I)+A(I)
v         ELSE
v          C(I)=SQRT(B(I))
v         END IF
v   10   CONTINUE
```

これもよくある形の IF 文です。SQRT, SIN など、通常の組み込み関数ならば問題なくベクトル化されます。

6.2.3 総和・内積計算

```
         S=0.0D0
         T=0.0D0
v        DO 10 K=1,N
v         S=S+A(I,K)
v         T=T+A(I,K)*B(K,J)
v   10   CONTINUE
```

総和 S と内積 T を求める計算も、プログラムの形から自動的にベクトル化されます。

6.2.4 最大値・最小値

```
         S= 1.0D+10
         T=-1.0D+10
v        DO 10 I=1,N
v         S=MIN(S,A(I))
v         T=MAX(T,A(I))
v   10   CONTINUE
```

最大値・最小値を求める関数 MAX, MIN もベクトル化されます。下の様に IF 文を用いて書いても大丈夫です。

```
         S= 1.0D+10
         T=-1.0D+10
v        DO 10 I=1,N
v         IF(S.GT.A(I)) S=A(I)
v         IF(T.LT.A(I)) T=A(I)
v   10   CONTINUE
```

6.2.5 リストベクトル

```
v        DO 10 I=1,N
v         B(I)=A(L(I))
v         C(I)=A(I)
v   10   CONTINUE
```

L(I) が添字となるリストベクトルです。

6.2.6 収集・拡散計算

```
                J=1
v               DO 10 I=1,N
v                IF(A(I).LT.0.0D0) THEN
v                 B(J)=A(I)
v                 J=J+1
v                END IF
v    10         CONTINUE
```

収集計算とは、条件を満たす配列要素を集める計算です。

```
                J=1
v               DO 20 I=1,N
v                IF(A(I).LT.0.0D0) THEN
v                 A(I)=B(J)
v                 J=J+1
v                END IF
v    20         CONTINUE
```

拡散計算は、逆に条件を満たした場合、他の値に拡散してしまう計算です。

6.2.7 一次回帰演算

```
v               DO 10 I=2,N
v                A(I)=A(I-1)*B(I)
v    10         CONTINUE
```

一次回帰演算とは、DO ループの 1 回前の実行で定義された値 (ここでは `A(I-1)`) を用いた演算のことをいいます。それ以外の回帰演算はベクトル化されません。

```
s2              DO 20 I=3,N
s2               A(I)=A(I-1)*B(I)+A(I-2)
s2   20         CONTINUE
```

上のプログラムは 2 回前の実行で定義された値 (`A(I-2)`) を用いているため、ベクトル化できません。

6.2.8 多重ループ

DO ループが二重以上のことを多重ループといいます。多重ループは、その中の 1 つだけ がベクトル化の対象になります。どのループを選択するかは、システムが一番よかれと判断したものを選択してくれます。

```
s2               DO 50 K=MM+1,M
v2                DO 40 I=1,L
v2                 C(I,J)=C(I,J)+A(I,K)*B(K,J)
v2   40           CONTINUE
s2   50          CONTINUE
```

これは、内側のループがベクトル化されたことを示しています。

```
00000012      v2       DO 10 J=1,N
00000013      s2        DO 20 I=3,N
00000014      v2         A(I,J)=A(I-2,J)+B(I,J)
00000015      s2  20    CONTINUE
00000016      v2  10   CONTINUE
```

内側のループは二次の回帰参照があるためベクトル化できません。しかし、添字 J についてはベクトル化できるため、外側のループがベクトル化されています。

6.2.9 多重ループの一重化

```
00000014      v        DO 10 J=1,N
00000015      v         DO 20 I=1,N
00000016      v           A(I,J) = SQRT(C(I,J))
00000017      v           B(I,J) = C(I,J)/2.0D0
00000018      v   20    CONTINUE
00000019      v   10   CONTINUE
```

DO ループの繰り返し数が少ない場合、ループを一重化し、一つのベクトルとして取り扱うことがあります。その場合、多重ループのどちらにも v が表示されます。

6.3 ベクトル化されない例

次は、ベクトル化されない DO ループの例を幾つかあげます。

6.3.1 回帰参照

6.2.7節の通り、DO ループの 2 回以上前の実行で定義された値を用いた回帰演算はベクトル化されません。ただし、多重ループの場合は 6.2.8節の例の様なループ交換がされている場合もあります。

6.3.2 利用者関数の組み込み

DO ループ内にサブルーチンや利用者定義関数を引用する文がある場合、その文はベクトル化されません。

```
00000012      s        DO 10 I=1,N
00000013      s         CALL USER(A(I),S)
00000014      s         T=T+S
00000015      s   10   CONTINUE
00000021
00000022               SUBROUTINE USER(X,Y)
00000023               REAL*8 X,Y
00000024                Y=ATAN(X)*COS(X)/2.0D0
00000025               RETURN
00000026               END
```

ベクトル化を促進する方策としては、CALL 文を使わないようにプログラムを修正する方法と、翻訳オプションとして外部手続きをループ内に展開するオプション -N を指定する方法とがあります。翻訳オプション -N には、きめ細かい設定が用意されています (付録 A参照)。組み込み関数の展開は、最適化オプション -Of でも実現できます。

```
00000012      v        DO 10 I=1,N
00000013      vi        CALL USER(A(I),S)
00000014      v         T=T+S
00000015      v   10   CONTINUE
```

メッセージの vi は、その部分に関数が展開され、ベクトル化されたことを意味しています。

6.3.3　配列のベクトル化指示

配列の添字に整数のインデックスベクトルを用いる場合がよくあります。

```
00000013      s8       DO 10 I=1,N
00000014      m8        A(L(I)) = A(L(I)) + 3.0D0
00000015      v8  10   CONTINUE
```

例では、L(I) がインデックスベクトルです。この場合、配列 A の添字が回帰データである可能性があるため、ベクトル化できません。もし L(I) の値の重複がないことがアルゴリズム上わかっているなら、強制的にベクトル化する制御行をソースに埋め込むことができます。

```
00000013         *VOCL LOOP,NOVREC(A)
00000014      v        DO 10 I=1,N
00000015      v         A(L(I)) = A(L(I)) + 3.0D0
00000016      v   10   CONTINUE
```

例では、 DO ループ内の配列 A について回帰参照がないことを指定しています。

ただし、最適化制御行はソースプログラムに指示行を埋め込むという性質から、翻訳システムに大きく依存します。個人的な意見ですが、できるだけ最後の手段としてとっておき、ソースプログラムの修正などによって対処する方が、他の計算機システムにも通用するプログラムに仕上げることができると思います。最適化制御行の詳細は [2] を参照下さい。

6.3.4　4 倍精度

残念なことに 4 倍精度型はベクトル化の対象外です。

```
00000005               REAL*16 A(N),B(N)
                            :
00000011      s9       DO 10 I=1,N
00000012      s9        B(I) = A(I) + 1.0Q0
00000013      s9        A(I) = 1.0Q0
00000014      s9  10   CONTINUE
```

例のようにスカラーモードでの実行となり、実行時間は単精度や倍精度に比べて大きく増加します。4 倍精度演算を行う場合は、欲しい精度と実行時間のバランスをよく考えながらプログラミングして下さい。

6.4　チューニングの方針

経験的に、プログラムのごく一部のサブルーチン、さらにその中のごく一部の DO ループのコストが全体のコストを左右します。チューニングの基本方針は、この部分を特定して、集中的な性能改善を行うことで全体の高速化を狙うのが最も有効です。つまり、次の手順となります。

1. CLOCKV 関数，実行解析ツール (6.6節参照) によって、プログラムの中で実行時間の多くかかるサブルーチン、 DO ループを特定する。

2. 特定した部分がベクトル化されているかどうかを翻訳メッセージで確認する (1.9.1節, 3.2節参照)。

3. ベクトル化されていない場合は、ソースを修正する。または翻訳オプション、最適化制御行の挿入を検討する。

4. ベクトル化されている場合でも、ベクトル長を長くするなどのチューニングを行う。

6.5 プログラミングの注意点

6.5.1 素直にプログラムを書く

ベクトル化によって大幅な速度アップが期待できる箇所は、 6.2節で紹介したパターンのような「大規模な配列に対する DO ループ内の 簡単 な演算」です。従って、あまり技巧に走らない素直なプログラムを書くことが基本です。

6.5.2 SSL II/VP の利用

素直にプログラムを書くと、行列とベクトルの積や連立 1 次方程式、固有値問題など、基本的な線形計算に多くのコストがかかるケースが多くなります。その場合、ハードウェアの性能を最大限に引き出すようにチューニングされたサブルーチンライブラリ、特に本センターでは SSL II/VP を積極的に利用することで、格段の高速化が期待できます。

6.5.3 データアクセスの効率化

主記憶上のとびとびのデータを参照するとき、データの間隔によっては参照の競合を起こし、性能が大きく低下することがあります。性能の劣化を防ぐためには、 2 次元配列を宣言する時、 第一添字 の寸法を次の大きさにとると適切なデータアクセスが得られます。

データ型	REAL*4 INTEGER*4 LOGICAL*4	REAL*8 COMPLEX*8 COMPLEX*16
第一添字の寸法	$4n+2$	$2n+1$

n は正の数

つまり、 1 行分だけ配列メモリーの無駄使いになるのは覚悟の上で次のように配列宣言することをお勧めします。

```
      PROGRAM SAMPLE
      IMPLICIT NONE
      INTEGER NMAX
      PARAMETER(NMAX=1024)
      REAL*8 A(NMAX+1,NMAX)    !  倍精度の場合
            :
```

6.6 Analyzer の利用

チューニングのためにはプログラムの中で実行時間の多くかかるサブルーチン、 DO ループを特定することが大事です。そのためのプログラムの実行解析支援ツールとして Analyzer があります。 Analyzer はプログラムをループ単位まで解析し、各実行コストのデータを収集します。ここでは、 Analyzer の簡単な使用方法とデータの見方を簡単に紹介します。なお、詳細は [3] を御覧下さい。

6.6.1 afrt コマンド

Analyzer の起動コマンドは afrt(/usr/uxp/afrt) です。 M-1800/20U，VP2600/10 上で動作します。主なオプションは次の通りです。

-p2600	対象プロセッサが VP2600/10 であることを指定．
-p2200	対象プロセッサが M-1800/20 であることを指定．
-S	利用者プログラムの各文の実行回数を仮定して解析 (見積り解析) することを指定．-t 及び -e とは排他．
-t	利用者プログラムの実行時間および手続き呼出し実行時間を計測 (実行解析) することを指定．-S とは排他．
-e	利用者プログラムの 文単位の実行回数を計測 (実行解析) することを指定．-S とは排他．
-Wc	翻訳オプションを指定．複数のオプションの場合は' 'で囲む．
-Wg	実行時オプションを指定．複数のオプションの場合は' 'で囲む．
-f␣*file*	解析結果を *file* に出力．

6.6.2 見積り解析

見積り解析は、プログラムを疑似的に実行し、動作を解析する機能です。この機能で各ルーチンの中でコストがかかる部分を (ある程度) 予測することができます。プログラムは実行しませんので、あくまでも「目安」としてデータを利用下さい[20]。

以下は M-1800/20U での見積り解析の例です。解析結果は out に出力します。

```
kyu-cc% afrt -p2200 -S -Wc,'-J' -f out test.f ↵
```

out の中にはサブルーチンごとに予想されるコストのリストと個別のサブルーチンに対する詳細情報が表示されます。

```
vectorize - total list -----------------------------------------------------------

name       ex-count      v-cost     %      s-cost     %      v-leng  v-rate      v-effect
HOBSVW           *1   *37758073  52.4  *.5380E09   46.9       *100   *98.0   *12.1- 16.8
GHBSVW           *1   *10484565  14.5  *.1937E09   16.9       *100   *99.7   *14.9- 23.1
MINVW            *1   *10113413  14.0  *.1903E09   16.6       *100   *99.8   *15.2- 23.6
CHOLFW           *1    *8868295  12.3  *.1492E09   13.0       *100   *99.1   *13.9- 20.5
MULMMW           *1    *3346647   4.6  *62243057    5.4       *100   *99.7   *15.0- 23.3
MUL              *1     *762349   1.1  *12251323    1.1       *100   *98.8   *13.4- 19.4
MAKEG            *1     *289401   0.4    *290634    0.0         *7    *0.4    *1.0-  1.0
MAKEA4           *1     *177091   0.2    *845169    0.1        *33   *80.0    *4.2-  5.0
MAKEA3           *1     *175462   0.2    *306488    0.0        *23   *50.0    *1.7-  1.8
MAKELI           *1      *70695   0.1     *70695    0.0         *9    *0.0    *1.0-  1.0
MAKEGI           *1      *25378   0.0    *427405    0.0       *100  *100.0   *12.1- 20.3
MAKEF            *1      *15898   0.0    *240514    0.0        *99  *100.0    *8.9- 17.6
CLEARM           *1       *6153   0.0    *110512    0.0        *99   *99.5   *14.6- 22.3
MATL             *1       *1040   0.0      *1040    0.0         *1    *0.0    *1.0-  1.0
ELMATE           *1        *604   0.0       *604    0.0         *1    *0.0    *1.0-  1.0
                                   :
```

ex-count は実行回数です。v-cost, s-cost はベクトル、スカラーでの実行コストと全体の比率です。v-leng は平均ベクトル長、v-rate はベクトル化率 (α)、v-effect は加速率 (β) の概算値です。

[20] ループの回転数と IF 文の真である確率を仮定して解析を行いますので、場合によってはかなりの誤差が出ます。

v-rate が 100% に近く、かつ v-effect の値が大きいサブルーチンはベクトル化による高速化が多いに期待できます。

バッチリクエストによる VP2600/10 での解析は次のようなスクリプトを記述してください。

```
#
cd EXAMPLE
afrt -p2600 -S -Wc,'-J' -f out test.f
```

バッチリクエストの投入方法は Fortran の翻訳・実行と全く同じです。

6.6.3 実行解析

実行解析は、プログラムを実際に実行し、実行回数や動作を解析する機能です。ルーチン単位や文単位ごとの実行回数や CPU 時間が計測できます。しかし、解析時間は普通の実行に比べて大幅に増加しますので、ご注意願います。

```
kyu-cc% afrt -p2200 -e -Wc,'-J' -f out test.f ↵
```

例では文単位の実行回数を M-1800/20U で計測しています。 VP2600/10 で解析する場合は、次の様にスクリプトを記述します。

```
#
cd EXAMPLE
afrt -p2600 -S -Wc,'-J' -f out test.f
```

以下は解析結果の一部です。

```
vectorize - total list ----------------------------------------------------------

name       ex-count    v-cost     %     s-cost     %     v-leng  v-rate      v-effect
MINVW             2  41785549  25.3  .1468E10   28.6        346    99.9   27.8- 41.1
MULMMW            3  41040639  24.8  .1457E10   28.4        310    99.9   28.1- 41.7
HOBSVW            2  31375825  19.0  .9762E09   19.0        137    99.9   24.5- 37.3
GHBSVW            2  23566105  14.2  .7156E09   13.9        163    99.8   23.1- 36.8
MUL               4  18217789  11.0  .2846E09    5.5         86    98.7   13.0- 18.7
CHOLFW            2   7934430   4.8  .2053E09    4.0        138    99.8   18.9- 32.6
CLEARM           12    366538   0.2  12715118    0.2        328    99.9   27.5- 40.5
MAKEA4            1    343958   0.2   5608802    0.1        149   100.0   14.0- 19.0
MAKEA3            1    281111   0.2   1808156    0.0        158    90.0    6.1-  6.8
MAKEG             1    259750   0.2    264850    0.0          7     2.0    1.0-  1.0
MAKEGI            1     87365   0.1   2000996    0.0        298   100.0   18.9- 28.0
MAKEF             2     77046   0.0   1372650    0.0        168   100.0   12.8- 20.3
MAKELI            1     74672   0.0     74672    0.0          9     0.0    1.0-  1.0
ELNODE          196     23660   0.0     23660    0.0          1     0.0    1.0-  1.0
ELMATE            2      1212   0.0      1212    0.0          1     0.0    1.0-  1.0
```

こちらは見積り解析と違い、プログラムを実行しながらデータを採取しますので、正確なデータを得ることができます。注目点は、サブルーチン単位のコストの割合とベクトル長、およびベクトル化率と加速率です。

実行解析は -t オプションによってプログラムの実行時間および手続き呼出し、実行時間を計測することもできます。

```
kyu-cc% afrt -p2200 -t -Wc,'-J' -f out test.f ↵
```

以下は解析結果です。

```
vectorize - total list -------------------------------------------------------------

name        ex-count     cpu-avg      vu-avg    cpu-time      %     vu-time      %   overhd
STOKES             1       0.002       0.000       0.002    0.1       0.000    0.0      0.0
MAKEG              1       0.003       0.000       0.003    0.3       0.000    0.0      1.6
CLEARM            12       0.000       0.000       0.004    0.4       0.004    0.4     13.9
GHBSVW             2       0.070       0.068       0.139   12.8       0.136   13.4      0.1
HOBSVW             2       0.100       0.097       0.201   18.4       0.194   19.1      0.1
CHOLFW             2       0.024       0.024       0.049    4.5       0.047    4.6      0.2
MINVW              2       0.131       0.125       0.262   24.1       0.250   24.6      0.0
MUL                4       0.064       0.057       0.256   23.5       0.226   22.3      0.1
MULMMW             3       0.053       0.052       0.158   14.5       0.155   15.2      0.1
(total)   ---------   ---------   ---------      1.090  100.0       1.016  100.0   ------

vectorize - routine list -----------------------------------------------------------
```

cpu-avg, vu-avg は、ルーチンの一回の呼び出しあたりの平均 CPU 時間およびベクトルユニット占有時間です。cpu-time, vu-time は CPU 時間およびベクトルユニット占有時間です。overhd はルーチンの CPU 時間に対して呼び出し (CALL) に要した負荷をパーセントで表しています。

その他、 Analyzer の解析結果にはプログラムリストに対応した詳細な情報が出力されますが、上の記号を知っていればほぼ内容は理解できるでしょう。

6.7 timex コマンド

コマンドレベルの実行時間は timex(/usr/bin/timex) コマンドで簡単に測定することができます。

```
kyu-cc% timex frt -J test.f  ↵     <--- timex コマンドによる翻訳時間の計測
             (翻訳結果)
real            9.20
user            6.94
sys             0.38
```

例として test.f の翻訳に要する時間を測定しました。real は経過時間、user, sys は CPU 時間の利用者およびシステム時間です。user+sys の合計で CPU 時間です。

```
kyu-cc% timex a.out  ↵     <--- timex コマンドによる実行時間の計測
             (実行結果)
real            2.07
user            1.06
sys             0.02
vu-user         1.00
vu-sys          0.01
```

今度は a.out の実行時間を測定しました。vu-user, vu-sys が CPU 時間のうちベクトルユニットで実行された時間を表します。vu-sys+vu-user でベクトルユニット占有時間 (VU 時間) です。一般にベクトルユニットでの実行時間の比率が高ければ高いほどベクトル化による性能向上が期待できます。

timex コマンドは、バッチリクエストの中にも同様に記述できます。詳細は 3.2.19節を御覧下さい。

6.8 CLOCKV サブルーチン

プログラムの任意の場所で時間計測を行うために、 CLOCKV サブルーチンが用意されています。

6.8.1 引用形式

`CALL CLOCKV(G1,G2,I1,I2)`

G1 : プログラム実行開始からのベクトル時間 (利用者とシステム時間の合計) を返す．

G2 : プログラム実行開始からの CPU 時間 (利用者とシステム時間の合計) を返す．

I1 : 返ってくる値の単位を指定．
0= 秒，1= ミリ秒，2= マイクロ秒．

I2 : G1, G2 の型を指定．
0=4 バイト整数，1= 単精度実数，2= 倍精度実数．

6.8.2 使用方法

サブルーチンや DO ループの前後に埋め込み、得られる時間の差をとる方法が最もポピュラーです。ここでは、時間計測に関係する部分だけ抜き出した例をあげます。

```
      REAL*8 VPU(2),CPU(2)                       ! 時間計測用変数の宣言
               :
      CALL CLOCKV(VPU(1),CPU(1),2,2)             ! 計測開始
      CALL SUB(A,NMAX,N,B,10D-16,1,ICON)         ! 副プログラムの CALL
      CALL CLOCKV(VPU(2),CPU(2),2,2)             ! 計測終了
               :
      write(6,1003) (CPU(2)-CPU(1))*1.0D-3       ! CPU 時間の計算・出力
      write(6,1004) (VPU(2)-VPU(1))*1.0D-3       ! ベクトル時間の計算・出力
1003  format(1x,'  CPU TIME AT SUB (MSEC.) = ',F8.2)
1004  format(1x,'  VPU TIME AT SUB (MSEC.) = ',F8.2)
```

A　UXP/Fortran 翻訳時オプション

UXP/Fortran の翻訳コマンド `frt`，`frtex` のオプション一覧です。翻訳オプションは `frt`，`frtex` のあと空白をおいて指定します。複数のオプションを指定する場合は空白で区切って続けます。また、大文字 / 小文字は区別して下さい。

`-J`	ベクトル演算処理を行うことを指定．`-Wv` オプションが指定された場合は必要ない．
`-c`	実行ファイルを作成せず，オブジェクトファイルのみを作成．副プログラムを翻訳する時などに指定．
`-o` *file*	指定したファイル名 *file* に実行ファイルを格納．
`-I` *dir*	INCLUDE 文に指定されているファイル名に対するディレクトリ名 *dir* を指定．
`-F`	プログラムが自由形式であることを指定．
`-C`	コメント文の圧縮処理を抑止する．
`-Y`l	単精度・倍精度・ 4 倍精度の実数型および複素数型のデータが代入文で設定される度に，データの仮数部の下位 l ビットをゼロにする．丸め誤差が計算結果に及ぼす影響を計測する場合などに用いる． l は 0 から 15 までの整数．
`-Z` *file*	指定したファイル名 *file* に翻訳時情報を出力する．

`-A` で始まるオプションは、精度拡張などの機能を指示します。複数の `-A` で始まるオプションを指定する場合は `-Aaf`，`-Aabd` などと記述できます。

`-Aa`	共通ブロック内データを境界調整して割り付ける．
`-Ab`	すべての仮引数を位置取りにする．
`-Ad`	単精度から倍精度に精度拡張する．
`-Aq`	倍精度から 4 倍精度に精度拡張する．
`-A0`	実行時にエラーが発生した場合でも，エラーが発生した行の位置を表示しない．
`-A1`	実行時にエラーが発生した場合，エラーが発生した外部手続きを引用している文の行位置を表示．
`-A2`	`-A1` の機能に加えて，プログラム割込みが発生した文の行位置を表示 (省略値)．
`-An`	`-A0` と同機能．
`-As`	シンボリックダンプ機能 (SDF) に適合するオブジェクトプログラムを生成する．プログラム単位の実行順序，変数または配列の名前・値・型・属性，ファイル情報を出力する．出力は実行時オプション `-Wl,s` で制御．

`-D` で始まるオプションは、デバッグ機能の動作を指示するオプションです。ベクトル演算におけるデバッグ機能は，実行モードがデバッグモードの場合にのみ有効です。従って `-J` または `-Wv,-an` が指定された場合 `-D` オプションは無効となります。複数のデバックオプションを指定する場合は `-Dis`，`-Daouv` などと記述できます。

`-DN`	プログラムに書かれたデバッグのための文を有効にする．
`-Da`	仮引数と実引数の矛盾を検査する．
`-Db`	実行した副プログラムの名前を表示．
`-Di`	変数や配列に割当てた値を表示．
`-Do`	整数演算のオーバフローの検出をする．
`-Ds`	添字式及び部分列式の値を検査．
`-Dt`	実行した文の文番号を表示．
`-Du`	未定義データの引用を検査．
`-Dv`	重複した実引数の値変更を検査．

`-E` で始まるオプションは、プログラムに対するコンパイラの診断メッセージの出力を指示します。複数の診断メッセージオプションを指定する場合は `-Ece`，`-Eupim` などと記述できます。

-Ec	浮動小数点に関する同値比較 (.EQ., .NE.) がある場合，メッセージを出力．
-Ee	演算の評価方法を変更する最適化に関するメッセージを出力．
-El	やや詳しい診断メッセージおよびベクトル化メッセージを出力．
-Em	プログラムリスト付きの診断メッセージを出力．
-Ei	外部手続きの引用箇所展開に関するメッセージを出力．
-Ep	不変式の先行評価の最適化に関するメッセージを出力．
-Eu	DO ループの回転数を減らす最適化に関するメッセージを出力．

-O で始まるオプションは、最適化機能を指示するオプションです。-On, -Ob, -Oe, -Of のどれかを指定します。ただし，-On はスカラーモードのみで、ベクトル化オプション-J，-Wv が指定された場合は無視されます。また、これらのオプションに続けてカンマ (,) で区切ることでサブオプションが指定できます。

-On	局所的な限定最適化を行う (スカラーモードのみ)．
-Ob	プログラム単位内の基本的な最適化を行う．
-Oe	-Ob に加えて，不変式の先行評価および演算評価方法の最適化を行う．
-Of	-Oe に加えて，プログラム単位間の最適化およびスカラーループに対する強い最適化を行う．
,-p	不変式の先行評価の最適化を行う．
,-P	不変式の先行評価の最適化を抑止する．
,-u	DO ループの回転数を減らす最適化を行う．
,-U	DO ループの回転数を減らす最適化を抑止する．
,-e	演算評価方法を変更する最適化を行う．
,-E	演算評価方法を変更する最適化を抑止する．

-N で始まるオプションは、組込み関数および外部手続きの展開を指示するオプションです。

-Ne	組込み関数および実行文の数が 30 以下の外部手続きを引用箇所に展開．
-Ne,*pgm*	組込み関数および *pgm* の外部手続きで実行文の数が 30 以下の外部手続きを引用箇所に展開．
-Ne,*stno*	組込み関数および実行文の数が *stno* 以下の外部手続きを引用箇所に展開．
-Ne,*dsizK*	組込み関数および配列の大きさが *dsiz* キロバイト以下の外部手続きを引用箇所に展開．
-Nf	組込み関数を引用箇所に展開．
-Nf,-i	最内ループの組込み関数だけを引用箇所に展開．
-Nf,-a	全ての組込み関数を引用箇所に展開． ベクトルモードでは省略値．
-NF	組込み関数を引用箇所に展開しない．

-P で始まるオプションは、翻訳時リストの出力機能を指示します。

-Ps	ベクトル化を表示したプログラムリストを出力．
-Pt	翻訳の統計情報を出力．
-Pd	INCLUDE 文で組み込まれたファイルも含めたプログラムリストを出力 (frt のみ)．
-Pi	ブロック IF，DO ループ，DO WHILE ループ，DO UNTIL ループの入れ子を意識して段階付けされたプログラムリストを出力 (frt のみ)．
-Pl	ブロック IF，DO ループ，DO WHILE ループ，DO UNTIL ループの入れ子の深さを表示したプログラムリストを出力 (frt のみ)．
-Px	プログラムで使用した名標および文番号の定義と引用の相互参照リストを出力 (frt のみ)．

-S で始まるオプションは、 翻訳メッセージの出力レベルを指示します。

-Si	すべてのメッセージを出力する (省略値)．
-Sw	i レベルのメッセージを抑止．i レベルとは，エラーではないが注意を促すメッセージ．
-Ss	i,w レベルのメッセージを抑止．w レベルとは，軽度のエラーを示すもので，システムはそのまま処理を実行する．

ベクトルオプションは -Wv に続けてカンマ (,) で区切り指定します。-Wv に続けて指定するオプションは、空白を含んではいけません。また、-Wv のみの指定は -J オプションと同じです。

-Wv,-a で始まるオプションは、実行モードの切替を行います。省略値は式の評価順序を変更した高速モードのオブジェクトプログラムを出力します。デバッグオプション -D を指定する場合は、デバッグモードに切替える必要があります。

-Wv,-an	式の評価順序を変更せず，高速モードで実行。
-Wv,-ae	式の評価順序を変更して，デバッグモードで実行．
-Wv,-ad	式の評価順序を変更せず，デバッグモードで実行．

-Wv,-m で始まるオプションは，ベクトル化に関するメッセージの出力を制御します。

-Wv,-m1	ベクトル化できなかった原因を簡略化して通知する．
-Wv,-m2	-m1 よりも詳しい情報を通知．
-Wv,-m3	-m2 に加えて，ベクトル化できた箇所もあわせて通知．
-Wv,-s	-m2 と同じ．
-Wv,-d	-m3 と同じ．
-Wv,-n	ソースプログラムに挿入した最適化制御行を無効にする．

以下は、その他のベクトルオプションです。

-Wv,-p2600	VP2600/10 向きにチューニングしたコードを出力．
-Wv,-rl	DO ループの繰返し回転の最大値が l であることを指定．l は 1 以上の整数．
-Wv,-sc	ベクトル化を行わないことを指示．
-Wv,-svl	翻訳時に繰返し回数が vl 未満と分かる DO ループのベクトル化を抑止．vl は 1 から 255 の整数．
-Wv,-te	翻訳時にベクトル化のための作業領域が不足したことを示すメッセージ jpc2021i が出力された場合，作業領域を拡張する．
-Wv,-Of	組込み関数 SQRT, DSQRT を高速化する．ただし精度は 10 進数で 1 桁落ちる．
-Wv,-Ov	DO ループの繰り返し回数が実行時にならないと決定できない場合，適切な命令列の選択実行を指定．
-Wv,-Vv	実数型および複素数型ベクトル演算を，倍精度型に変更する．

B 実行時オプション一覧

実行時オプションは、実行ファイル名 (何も指定がないと a.out) のあと空白をおいて -Wl, に続けて指定します。各オプションはカンマ (,) で区切って下さい。

-Wl,-a	プログラムの終了時に強制的に異常終了させ，記憶領域の内容をファイル core に出力する．
-Wl,-d*num1*	直接入出力文を実行する場合，OPEN 文で指定した RECL の値の *num1* 倍の作業領域を使用して入出力を行う．*num1* は 1 から 32767 までの整数．省略値は 20．
-Wl,-e*num2*	プログラムの実行を打ち切るエラー回数を *num2* とする．
-Wl,-g*num3*	順次入出力文の作業領域の大きさを *num3* キロバイトで指定．省略値は 4．書式なし入出力文で大量のデータを入出力する場合に効果がある．
-Wl,-i	実行時にプログラム割込みが発生した場合に Fortran システムで割込みを検出しない．
-Wl,-lw	実行時のすべてのメッセージを出力 (省略値)．
-Wl,-le	i,w レベルのメッセージを抑止．
-Wl,-ls	i,w,e レベルのメッセージを抑止．e レベルとは中度のエラーを示すメッセージで，実行時は 10 回の e レベルエラーを検出すると処理を打ち切る．
-Wl,-m*uno*	診断メッセージを出力するファイルの装置参照番号を *uno* で指定．*uno* は 0 から 99 までの整数．
-Wl,-n	端末に入力促進メッセージを出力する．
-Wl,-o	整数演算でオーバーフローが発生した場合，それを検出することを指定．
-Wl,-p*uno*	標準出力ファイルと結合する装置参照番号を *uno* とする．省略値は 6．
-Wl,-r*uno*	標準入力ファイルと結合する装置参照番号を *uno* とする．省略値は 5．
-Wl,-q	書式付き出力文における E,D,Q,G,I, および Z 編集の出力文および INQUIRE 文における問い合わせ指定子に設定される文字定数を英大文字にする．
-Wl,-s0	異常終了時または SDFDMP サービスサブルーチンにより出力されるシンボリックダンプ情報を制御する．翻訳オプションとして -As が指定されていた場合に有効となる．-s0 は，SDFDMP サービスサブルーチンの第 1 パラメータで指定した値に従う．
-Wl,-s1	主プログラムから現プログラム単位まで，トレースバックチェーンの各プログラム単位ごとにシンボリックダンプ情報を出力．
-Wl,-s2	主プログラムから現プログラム単位までの全てのプログラム単位ごとにシンボリックダンプ情報を出力．
-Wl,-t*sec*	プログラム実行における CPU 時間の上限を *sec* 秒に設定する．
-Wl,-u	指数アンダーフロー (浮動小数点演算の結果の絶対値が 10^{-65} よりも小さい) を検出する．
-Wl,-x	事前に OPEN 文が実行されていない書式付き入力文の数値編集において，入力欄の空白を 0 と解釈する．
-Wl,-C*uno*	*uno* からのバイナリーデータの入出力を IEEE 形式で行う．*uno* を省略した場合は，すべての装置参照番号を指定したものとする．
-Wl,-M	-Wl,-C オプション指定時に，データ変換で情報の欠落が生じた場合，メッセージを出力．

以下は、ベクトルモード固有の実行時オプションです。

-Wl,-Vf	データ例外の診断メッセージの出力を抑止．
-Wl,-Vb	アドレス境界指定例外の診断メッセージの出力を抑止．
-Wl,-Vv	実行時に単精度を倍精度に変更する．
-Wl,-Vp	実行時に必要となった動的作業域に関する情報を出力．

C　SSL II/VP 機能一覧

各サブルーチンは、一部を除き単精度用と倍精度用があります。4 倍精度はサポートしていません。倍精度は単精度サブルーチンの頭に'D' がつきます。なお、[5] より実行性能データを引用します。記号の対応は以下の通りです。

AA	:	ピーク性能に近い最大限のベクトル化効果がある
A	:	極めて高いベクトル化効果がある
B	:	かなりのベクトル化効果がある
C	:	そこそこのベクトル化効果がある
D	:	ベクトル化効果は低いが最適化処理により高速処理される
E	:	ベクトル化効果は低く、ほぼスカラー性能と同じ
–	:	この精度のサポートはない
空白	:	データなし

行列格納モードの変換

		単精度	倍精度
CGSM	行列格納モードの変換 (一般モード → 対称行列)	A	A
CSGM	行列格納モードの変換 (対称行列 → 一般モード)	A	A
CGSBM	行列格納モードの変換 (一般モード → 対称バンド行列)	A	A
CSBGM	行列格納モードの変換 (対称バンド行列 → 一般モード)	A	A
CSSBM	行列格納モードの変換 (対称行列 → 対称バンド行列)	A	A
CSBSM	行列格納モードの変換 (対称バンド行列 → 対称行列)	A	A

行列操作

		単精度	倍精度
AGGM	行列の和 (実行列)	A	A
SGGM	行列の差 (実行列)	A	A
MGGM	行列の積 (実行列)	A	A
VMGGM	行列の積 (実行列) [拡張機能]	AA	AA
MGSM	行列の積 (実行列・実対称行列)	A	A
ASSM	行列の和 (実対称行列)	A	A
SSSM	行列の差 (実対称行列)	A	A
MSSM	行列の積 (実対称行列)	A	A
MSGM	行列の積 (実対称行列・実行列)	A	A
MAV	実行列と実ベクトルの積	A	A
MCV	複素行列と複素ベクトルの積	A	A
MSV	実対称行列と実ベクトルの積	A	A
MSBV	実対称バンド行列と実ベクトルの積	A	A
MBV	実バンド行列と実ベクトルの積	A	A

連立 1 次方程式

		単精度	倍精度
LAX	実行列の連立 1 次方程式 (クラウト法)	A	A
VLAX	実行列の連立 1 次方程式 (ブロッキング LU 分解法) [拡張機能]	AA	AA
LCX	複素行列の連立 1 次方程式 (クラウト法)	A	A
LSX	正値対称行列の連立 1 次方程式 (変形コレスキー法)	B	A
VLSX	正値対称行列の連立 1 次方程式 (変形コレスキー法) [拡張機能]	A	A
LSIX	実対称行列の連立 1 次方程式 (ブロック対角ピボッティング手法)	B	B
LSBX	正値対称バンド行列の連立 1 次方程式 (変形コレスキー法)	C	C
LSBIX	実対称バンド行列の連立 1 次方程式 (ブロック対角ピボッティング手法)	E	E
LBX1	実バンド行列の連立 1 次方程式 (ガウス消去法)	B	B
LSTX	正値対称 3 項行列の連立 1 次方程式 (変形コレスキー法)	E	E
LTX	実 3 項行列の連立 1 次方程式 (ガウス消去法)	E	E
VLTX	実 3 項行列の連立 1 次方程式 (サイクリック・リダクション法) [拡張機能]	AA	AA
VLTX1	定数型実 3 項行列の連立 1 次方程式 [拡張機能]	AA	AA
VLTX2	定数型実 3 項行列の連立 1 次方程式 [拡張機能]	AA	AA
VLTX3	定数型実 3 項行列の連立 1 次方程式 [拡張機能]	C	C
LAXR	実行列の連立 1 次方程式の解の反復改良	A	E
LCXR	複素行列の連立 1 次方程式の解の反復改良	A	E
LSXR	正値対称行列の連立 1 次方程式の解の反復改良	C	E
LSIXR	実対称行列の連立 1 次方程式の解の反復改良	E	E
LSBXR	正値対称バンド行列の連立 1 次方程式の解の反復改良	E	E
LBX1R	実バンド行列の連立 1 次方程式の解の反復改良	B	E

逆行列

		単精度	倍精度
LUIV	LU 分解された実行列の逆行列	A	A
VLUIV	LU 分解された実行列の逆行列 [拡張機能]	AA	AA
CLUIV	LU 分解された複素行列の逆行列	A	A
LDIV	LDL^T 分解された正値対称行列の逆行列	A	A

行列の三角分解

		単精度	倍精度
ALU	実行列の LU 分解 (クラウト法)	A	A
VALU	実行列の LU 分解 (ブロッキング LU 分解法) [拡張機能]	AA	AA
CLU	複素行列の LU 分解 (クラウト法)	A	A
SLDL	正値対称行列の LDL^T 分解 (変形コレスキー法)	B	A
VSLDL	正値対称行列の LDL^T 分解 [拡張機能]	AA	AA
SMDM	実対称行列の MDM^T 分解 (ブロック対角ピボッティング手法)	B	B
SBDL	正値対称バンド行列の LDL^T 分解 (変形コレスキー法)	C	C
SBMDM	実対称バンド行列の MDM^T 分解 (ブロック対角ピボッティング手法)	E	E
BLU1	実バンド行列の LU 分解 (ガウス消去法)	B	B

三角分解された連立 1 次方程式

		単精度	倍精度
LUX	LU 分解された実行列の連立 1 次方程式	A	A
CLUX	LU 分解された複素行列の連立 1 次方程式	A	A
LDLX	LDL^T 分解された正値対称行列の連立 1 次方程式	B	A
VLDLX	LDL^T 分解された正値対称行列の連立 1 次方程式 [拡張機能]	A	A
MDMX	MDM^T 分解された実対称行列の連立 1 次方程式	B	B
BDLX	LDL^T 分解された正値対称バンド行列の連立 1 次方程式	C	C
BMDMX	MDM^T 分解された実対称バンド行列の連立 1 次方程式	E	E
BLUX1	LU 分解された実バンド行列の連立 1 次方程式	B	B

最小二乗解

		単精度	倍精度
LAXL	実行列の最小二乗解 (ハウスホルダー変換)	A	A
LAXLR	実行列の最小二乗解の反復改良	A	C
LAXLM	実行列の最小二乗最小ノルム解 (特異値分解法)	A	A
GINV	実行列の一般逆行列 (特異値分解法)	A	A
ASVD1	実行列の特異値分解 (ハウスホルダー法, QR 法)	A	A

固有値，固有ベクトル

		単精度	倍精度
EIG1	実行列の固有値及び固有ベクトル (2 段 QR 法)	A	A
CEIG2	複素行列の固有値及び固有ベクトル (QR 法)	A	A
SEIG1	実対称行列の固有値及び固有ベクトル (QL 法)	A	A
SEIG2	実対称行列の固有値及び固有ベクトル (バイセクション法, 逆反復法)	C	C
HEIG2	エルミート行列の固有値及び固有ベクトル (バイセクション法, 逆反復法)	A	B
BSEG	実対称バンド行列の固有値及び固有ベクトル	E	E
BSEGJ	実対称バンド行列の固有値及び固有ベクトル (ジェニングス法)	C	B
TEIG1	実対称 3 重対角行列の固有値及び固有ベクトル (QL 法)	B	A
TEIG2	実対称 3 重対角行列の固有値及び固有ベクトル (バイセクション法, 逆反復法)	C	C
GSEG2	実対称行列の一般固有値及び固有ベクトル (バイセクション法, 逆反復法)	B	C
GBSEG	実対称バンド行列の一般固有値及び固有ベクトル (ジェニングス法)	C	C
VSEG2	実対称行列の固有値・固有ベクトル [拡張機能]	A	A
VGSG2	実対称行列の一般固有値・固有ベクトル [拡張機能]	A	A

固有値

		単精度	倍精度
HSQR	実ヘッセンベルグ行列の固有値 (2 段 QR 法)		
CHSQR	複素ヘッセンベルグ行列の固有値 (QR 法)		
TRQL	実対称 3 重対角行列の固有値 (QL 法)		
BSCT1	実対称 3 重対角行列の固有値 (バイセクション法)		

固有ベクトル

		単精度	倍精度
HVEC	実ヘッセンベルグ行列の固有ベクトル (逆反復法)		
CHVEC	複素ヘッセンベルグ行列の固有ベクトル (逆反復法)		
BSVEC	実対称バンド行列の固有ベクトル (逆反復法)		

その他の固有値計算

		単精度	倍精度
BLNC	実行列の平衡化		
CBLNC	複素行列の平衡化		
HES1	実行列の実ヘッセンベルグ行列への変換 (ハウスホルダー法)		
CHES2	複素行列の複素ヘッセンベルグ行列への変換 (安定化基本相似変換)		
TRID1	実対称行列の実対称 3 重対角行列への変換 (ハウスホルダー法)		
TRIDH	エルミート行列の実対称 3 重対角行列への変換 (ハウスホルダー法)		
BTRID	実対称バンド行列の実対称 3 重対角行列への変換		
HBK1	実行列の固有ベクトルへの逆変換と正規化		
CHBK2	複素行列の固有ベクトルへの逆変換		
TRBK	実対称行列の固有ベクトルへの逆変換		
TRBKH	エルミート行列の固有ベクトルへの逆変換		
NRML	実行列の固有ベクトルの正規化		
CNRML	複素行列の固有ベクトルの正規化		
GSCHL	一般形から標準形への変換 (実対称行列の一般固有値問題)		
GSBK	一般形の固有ベクトルへの逆変換 (実対称行列の一般固有値問題)		

非線形計算

		単精度	倍精度
RQDR	実係数 2 次方程式	E	E
CQDR	複素係数 2 次方程式	E	E
LOWP	実係数低次代数方程式 (5 次以下)	C	C
RJETR	実係数高次代数方程式 (ジェンキンス・トラウブの方法)	E	E
CJART	複素係数高次代数方程式 (ヤラット法)	E	E
TSD1	実超越方程式 $f(x) = 0$ (ブレント法)	E	E
TSDM	実超越方程式 $f(x) = 0$(マラー法)	E	E
CTSDM	複素超越方程式 $f(z) = 0$(マラー法)	E	E
NOLBR	連立非線形方程式 (ブレント法)	C	C

極値問題

		単精度	倍精度
LMINF	1 変数関数の極小化 (微係数不要, 2 次補間法)	E	E
LMING	1 変数関数の極小化 (微係数要, 3 次補間法)	E	E
MINF1	多変数関数の極小化 (微係数不要, 改訂準ニュートン法)	C	C
MING1	多変数関数の極小化 (微係数要, 準ニュートン法)	B	B
NOLF1	関数二乗和の極小化 (微係数不要, 改訂マルカート法)	B	B
NOLG1	関数二乗和の極小化 (微係数要, 改訂マルカート法)	B	B
LPRS1	線形計画問題 (改訂シンプレックス法)		
NLPG1	非線形計画問題 (微係数要, パウエル法)		

補間

		単精度	倍精度
AKLAG	エイトケン・ラグランジュ補間	E	E
AKHER	エイトケン・エルミート補間	E	E
SPLV	3 次 spline 補間式による補間, 数値微分	E	E
BIF1	B-spline 補間式 (I) による補間, 数値微分, 数値積分	E	E
BIF2	B-spline 補間式 (II) による補間, 数値微分, 数値積分	E	E
BIF3	B-spline 補間式 (III) による補間, 数値微分, 数値積分	E	E
BIF4	B-spline 補間式 (IV) による補間, 数値微分, 数値積分	E	E
BIFD1	B-spline2 次元補間式 (I-I) による補間, 数値微分, 数値積分	E	E
BIFD3	B-spline2 次元補間式 (III-III) による補間, 数値微分, 数値積分	E	E
AKMID	2 次元準エルミート補間式による補間	E	E
INSPL	3 次 spline 補間式	E	E
AKMIN	準エルミート補間式	A	A
BIC1	B-spline 補間式 (I)	E	E
BIC2	B-spline 補間式 (II)	E	E
BIC3	B-spline 補間式 (III)	E	E
BIC4	B-spline 補間式 (IV)	E	E
BICD1	B-spline2 次元補間式 (I-I)	E	E
BICD3	B-spline2 次元補間式 (III-III)	E	E

近似

		単精度	倍精度
LESQ1	最小二乗近似多項式	A	A

平滑化

		単精度	倍精度
SMLE1	最小二乗近似多項式による平滑化 (等間隔離散点)	A	A
SMLE2	最小二乗近似多項式による平滑化 (不等間隔離散点)	E	E
BSF1	B-spline 平滑化式による平滑化, 数値微分, 数値積分	E	E
BSC1	B-spline 平滑化式 (固定節点)	E	E
BSC2	B-spline 平滑化式 (節点追加方式)	E	E
BSFD1	B-spline2 次元平滑化式による平滑化, 数値微分, 数値積分	E	E
BSCD2	B-spline2 次元平滑化式 (節点追加方式)	E	E

級数

		単精度	倍精度
FCOSF	偶関数の cosine 級数展開 (関数入力, 高速 cosine 変換)	E	E
ECOSP	cosine 級数の求和	E	E
FSINF	奇関数の sine 級数展開 (関数入力, 高速 sine 変換)	E	E
ESINP	sine 級数の求和	E	E
FCHEB	実関数のチェビシェフ級数展開 (関数入力, 高速 cosine 変換)	E	E
ECHEB	チェビシェフ級数の求和	E	E
GCHEB	チェビシェフ級数の導関数, 数値微分	E	E
ICHEB	チェビシェフ級数の不定積分	E	E

変換

		単精度	倍精度
FCOST	離散型 cosine 変換 (台形公式, 2 基底 FFT)	E	E
FCOSM	離散型 cosine 変換 (中点公式, 2 基底 FFT)	E	E
FSINT	離散型 sine 変換 (台形公式, 2 基底 FFT)	E	E
FSINM	離散型 sine 変換 (中点公式, 2 基底 FFT)	E	E
RFT	離散型実フーリエ変換	A	A
CFTM	多次元離散型複素フーリエ変換 (混合基底 FFT)	E	E
CFT	多次元離散型複素フーリエ変換 (8, 2 基底 FFT)	A	B
CFTN	離散型複素フーリエ変換 (8, 2 基底 FFT, 逆順出力)	A	A
CFTR	離散型複素フーリエ変換 (8, 2 基底 FFT, 逆順入力)	B	B
PNR	ビット逆転によるデータの置換	B	C
LAPS1	ラプラス変換 (複素右半平面で正則な有理関数)	E	E
LAPS2	ラプラス変換 (一般の有理関数)	E	E
LAPS3	ラプラス変換 (一般関数)	E	E
HRWIZ	Hurwitz 多項式の判定	E	E
VCOS1	離散型 cosine 変換 (2 基底 FFT) [拡張機能]	A	A
VSIN1	離散型 sine 変換 (2 基底 FFT) [拡張機能]	A	A
VRFT1	離散型実フーリエ変換 (性能優先型, 2 基底 FFT) [拡張機能]	A	A
VRFT2	離散型実フーリエ変換 (メモリー節約型, 2 基底 FFT) [拡張機能]	A	A
VCFT1	離散型複素フーリエ変換 (性能優先型, 2 基底 FFT) [拡張機能]	A	A
VCFT2	離散型複素フーリエ変換 (メモリー節約型, 2 基底 FFT) [拡張機能]	A	A

疑似乱数

		単精度	倍精度
RANU2	一様乱数 $(0,1)$ の生成	A	–
RANU3	一様乱数 $(0,1)$ の生成 (シャフル型)	E	–
RANN1	正規乱数の生成 (高速型)	B	–
RANN2	正規乱数の生成	A	–
RANE2	指数乱数の生成	A	–
RANP2	ポアソン乱数の生成	E	–
RANB2	二項乱数の生成	E	–
RATF1	一様乱数 $(0,1)$ の頻度テスト	C	–
RATR1	一様乱数 $(0,1)$ の上昇・下降連テスト	E	–

数値微分

		単精度	倍精度
SPLV	3 次 spline 補間式による補間, 数値微分		
BIF1	B-spline 補間式 (I) による補間, 数値微分, 数値積分		
BIF2	B-spline 補間式 (II) による補間, 数値微分, 数値積分		
BIF3	B-spline 補間式 (III) による補間, 数値微分, 数値積分		
BSF1	B-spline 平滑化式による平滑化, 数値微分, 数値積分		
BIFD1	B-spline2 次元補間式 (I-I) による補間, 数値微分, 数値積分		
BIFD3	B-spline2 次元補間式 (III-III) による補間, 数値微分, 数値積分		
BSFD1	B-spline2 次元平滑化式による平滑化, 数値微分, 数値積分		
GCHEB	チェビシェフ級数の導関数, 数値微分		

数値積分

		単精度	倍精度
SIMP1	1 次元有限区間積分 (等間隔離散点入力, シンプソン則)	B	B
TRAP	1 次元有限区間積分 (不等間隔離散点入力, 台形則)	A	A
SIMP2	1 次元有限区間積分 (関数入力, 適応型シンプソン則)	E	E
BIF1	B-spline 補間式 (I) による補間, 数値微分, 数値積分		
BIF2	B-spline 補間式 (II) による補間, 数値微分, 数値積分		
BIF3	B-spline 補間式 (III) による補間, 数値微分, 数値積分		
BSF1	B-spline 平滑化式による平滑化, 数値微分, 数値積分		
BIFD1	B-spline2 次元補間式 (I-I) による補間, 数値微分, 数値積分		
BIFD3	B-spline2 次元補間式 (III-III) による補間, 数値微分, 数値積分		
BSFD1	B-spline2 次元平滑化式による平滑化, 数値微分, 数値積分		
AQN9	1 次元有限区間積分 (関数入力, 適応型ニュートン・コーツ 9 点則)	E	E
AQC8	1 次元有限区間積分 (関数入力, クレンショー・カーチス型積分法)	E	E
AQE	1 次元有限区間積分 (関数入力, 二重指数関数型積分公式)	E	E
AQEH	1 次元半無限区間積分 (関数入力, 二重指数関数型積分公式)	E	E
AQEI	1 次元全無限区間積分 (関数入力, 二重指数関数型積分公式)	E	E
AQMC8	多次元有限領域積分 (関数入力, クレンショー・カーチス型積分法)	E	E
AQME	多次元積分 (関数入力, 二重指数関数型積分公式)	E	E

微分方程式

		単精度	倍精度
RKG	連立 1 階常微分方程式 (ルンゲ・クッタ・ギル法)	C	C
HAMNG	連立 1 階常微分方程式 (ハミング法)	C	C
ODRK1	連立 1 階常微分方程式 (ルンゲ・クッタ・ヴァーナー法)	C	C
ODAM	連立 1 階常微分方程式 (アダムス法)	A	A
ODGE	スティフ連立 1 階常微分方程式 (ギア法)	B	A

特殊関数

		単精度	倍精度
CELI	第 1 種完全楕円積分 $K(x)$	D	D
CELI2	第 2 種完全楕円積分 $E(x)$	D	D
EXPI	指数積分 $E_i(x), \bar{E}_i(x)$	D	D
SINI	正弦積分 $S_i(x)$	D	D
COSI	余弦積分 $C_i(x)$	D	D
SFRI	正弦フレネル積分 $S(x)$	D	D
CFRI	余弦フレネル積分 $C(x)$	D	D
IGAM1	第 1 種不完全ガンマ関数 $\gamma(\nu, x)$	D	D
IGAM2	第 2 種不完全ガンマ関数 $\Gamma(\nu, x)$	D	D
IERF	逆誤差関数 $erf^{-1}(x)$	D	D
IERFC	逆余誤差関数 $erfc^{-1}(x)$	D	D
BJ0	第 1 種 0 次ベッセル関数 $J_0(x)$	D	D
BJ1	第 1 種 1 次ベッセル関数 $J_1(x)$	D	D
BY0	第 2 種 0 次ベッセル関数 $Y_0(x)$	D	D
BY1	第 2 種 1 次ベッセル関数 $Y_1(x)$	D	D
BI0	第 1 種 0 次変形ベッセル関数 $I_0(x)$	D	D
BI1	第 1 種 1 次変形ベッセル関数 $I_1(x)$	D	D
BK0	第 2 種 0 次変形ベッセル関数 $K_0(x)$	D	D
BK1	第 2 種 1 次変形ベッセル関数 $K_1(x)$	D	D
BJN	第 1 種整数次ベッセル関数 $J_n(x)$	D	D
BYN	第 2 種整数次ベッセル関数 $Y_n(x)$	D	D
BIN	第 1 種整数次変形ベッセル関数 $I_n(x)$	D	D
BKN	第 2 種整数次変形ベッセル関数 $K_n(x)$	D	D
CBIN	複素変数第 1 種整数次変形ベッセル関数 $I_n(z)$	D	D
CBKN	複素変数第 2 種整数次変形ベッセル関数 $K_n(z)$	D	D
CBJN	複素変数第 1 種整数次ベッセル関数 $J_n(z)$	D	D
CBYN	複素変数第 2 種整数次ベッセル関数 $Y_n(z)$	D	D
BJR	第 1 種実数次ベッセル関数 $J_\nu(x)$	D	D
BYR	第 2 種実数次ベッセル関数 $Y_\nu(x)$	D	D
BIR	第 1 種実数次変形ベッセル関数 $I_\nu(x)$	D	D
BKR	第 2 種実数次変形ベッセル関数 $K_\nu(x)$	D	D
CBJR	複素変数第 1 種実数次ベッセル関数 $J_\nu(z)$	D	D
NDF	正規分布関数 $\phi(x)$	D	D
NDFC	余正規分布関数 $\psi(x)$	D	D
INDF	逆正規分布関数 $\phi^{-1}(x)$	D	D
INDFC	逆余正規分布関数 $\psi^{-1}(x)$	D	D

D　NUMPAC 機能一覧

各サブルーチン名は目的別にまとめ、代表する副プログラム名のみを記述しています。単精度 / 倍精度等の型の区別や、具体的な使用方法は参考文献を参照下さい。

基本行列演算

ADDMMV	行列の加減算
MDETS	行列式の計算
MNORMS	行列の正規化
MNRMBS	バンド行列の正規化
MNRSPS	対称対称行列の正規化
MULMMV	行列の乗算
MULMVV	行列とベクトルの乗算

連立 1 次方程式

BUNCBS	Bunch 法による対称バンド行列係数連立一次方程式の解法
BUNCHS	Bunch 法による対称行列係数連立一次方程式の解法
CGHTCS	共役勾配法による対称正値連立一次方程式の解法（圧縮モード）
CHLBDC	Cholesky 法, 改訂 Cholesky 法による Hermite 対称正値連立一次方程式の解法 (バンド行列)
CHLVBS	Cholesky 法による対称正値連立一次方程式の解法（可変帯幅バンド行列，圧縮表現）
CHOLCS	Cholesky 法, 改訂 Cholesky 法による対称正値連立一次方程式の解法 (密行列, 圧縮表現)
CHOLFC	Cholesky 法, 改訂 Cholesky 法による Hermite 対称正値連立一次方程式の解法（密行列）
CHOLFS	Cholesky 法, 改訂 Cholesky 法による対称正値連立一次方程式の解法（密行列）
CHOLSK	Cholesky 法による対称正値連立一次方程式の解法
GAUELS	LU 分解による連立一次方程式の解法
GSORSS	SOR 分解による疎行列の連立一次方程式の解法（圧縮表現）
LAPLBS	二次元 Laplace 方程式の解法
LEQBDS	Gauss の消去法によるバンド行列係数連立一次方程式の解法
LEQLSS	Householder 変換による一般連立一次方程式の最小二乗解および最小ノルム解
LEQLUS	LU 分解による連立一次方程式解法
LSMNS	特異値分解による一般連立一次方程式の解法最小二乗最小ノルム解
PRCGFS	前処理付き共役勾配法による対称正値連立一次方程式の解法
PRCGSS	前処理付き共役勾配法による対称正値疎行列の連立一次方程式の解法（圧縮表現）
TRDSPS	対称正値三項方程式の解法
TRIDGS	三項方程式の解法

行列の逆転

GINVS	特異値分解による一般化逆行列
MINVS	行列の逆転
MINVSP	対称正値行列の逆転

代数方程式，非線形方程式

BROYDS	Broyden の方法による非線形連立一次方程式の解法
FLPOWS	Davidon-Fletcher-Powell 法による関数の最小化
GJMNKS	Garside-Jarrat-Mack 法による実係数代数方程式の解法
MINSXS	Simplex 法による関数の最小化
NOLEQS	非線形方程式の解法
NOLLS1	Quasi-Newton 法による非線形最小二乗法サブルーチン
POLEQC	複素係数代数方程式の解法
POLESB	静電場モデルによる複素係数代数方程式の解法
QUADRC	複素係数低次数方程式の解法
QUADRS	実係数低次代数方程式の解法
RTFNDS	非線形方程式の解法

線形計画法

LIPS	Criss-Cross 法を用いた線形計画問題解法ルーチン
SIMPLX	Simplex 法を用いた線形計画法

数列と級数の加速

ACCELS	数列または級数の収束の加速

固有値解析

CGHBSS	Householder-Bisection 法による $Ax = \lambda Bx$ 型の固有値解析（Hermite 行列）
CGHQIS	Householder-QR-Inverse Iteration 法による $Ax = \lambda Bx$ 型の固有値解析（Hermite 行列）
CGHQRS	Householder-QR 法による $Ax = \lambda Bx$ 型の固有値解析（Hermite 行列）
CGKLZS	LZ 法による $Ax = \lambda Bx$ 型の固有値解析（複素行列）
CHEQIS	QR 法および逆反復法による複素行列の固有値解析
CHEQRS	QR 法による複素行列の固有値解析
CHOBSS	Householder-Bisection 法による Hermite 行列の固有値解析
CHOQRS	Householder-QR 法による Hermite 行列の固有値解析
CHQRIS	Householder-QR- 逆反復法による Hermite 行列の固有値解析
GHBSVS	Householder-Bisection 法による $Ax = \lambda Bx$ 型の固有値解析
GHQRIS	Householder-QR- 逆反復法による $Ax = \lambda Bx$ 型の固有値解析
GHQRVS	Householder-QR 法による $Ax = \lambda Bx$ 型の固有値解析
HEQRVS	Double QR 法による実非対称行列の固有値解析
HOBSVS	Householder-Bisection 法による実対称行列の固有値解析
HOQRVS	Householder-QR 法による実対称行列の固有値解析
HQRIIS	Householder-QR- 逆反復法による実対称行列の固有値解析
JACOBS	Threshold Jacobi 法による実対称行列の固有値解析
JENNFS	Jennings の同時反復法による実対称行列の固有値解析
NGHOUS	準直接法による $Ax = \lambda Bx$ 型の固有値解析
NGJENS	Jennings 法による $Ax = \lambda Bx$ 型の固有値解析
NSHOUS	準直接法による $Ax = \lambda x$ 型の固有値解析
NSJENS	Jennings 法による $Ax = \lambda x$ 型の固有値解析
RHBSVS	Rutishauser-Bisection 法による対称バンド行列の固有値解析
RHQRVS	Rutishauser-QR 法による実対称バンド行列の固有値解析
SVDS	特異値分解

補間，平滑化

AGFBS	Briggs の方法による不規則分布データの格子化
CFS1A	Spline による曲線のあてはめ
CFS2A	Spline による曲面のあてはめ
DCOMD1	複合多項式による曲線のあてはめ
DSCI1A	１変数スプライン補間
DSCI1D	２変数スプライン補間
HERM31	区分的 Hermite 補間による曲線のあてはめ
HERM32	区分的 Hermite 補間による曲面のあてはめ
LSAICS	直交多項式による最小二乗近似
LSANLS	非線形最小二乗法による曲線のあてはめ
TETPCK	不規則分布 3 変数関数データに対する C^k 級補間法 $(0 \leq k \leq 1)$
TRIPCK	不規則分布 2 変数関数データに対する C^k 級補間法 $(0 \leq k \leq 3)$

表関数

BERNO	Bernoulli 数
BETNO	β 数
EULNO	Euler 数
FCTRL	階数の計算
GAMCO	$1/\Gamma(x)$ の Taylor 級数展開係数
HARMS	$\phi(n) = \sum_{k=1}^{n}(1/k)$
ZETNO	$\zeta(n) = \sum_{k=1}^{\infty}(1/k^n)$

フーリエ解析

BITREV	ビット逆転によるデータの並べ替え
DRCH1S	Chebyshev 級数の導関数，不定積分
FCHB1S	Chebyshev 多項式による関数の Fourier 展開
FCOSCS	開区間 $(0, \pi)$ で与えられた関数の級数展開
FCOSMS	中点公式にもとづく高速 cosine, sine 変換
FCOSTS	台形公式にもとづく高速 cosine, sine 変換
FFT2DC	複素高速 Fourier 変換 (2, 3 次元)
FFT2DR	実高速 Fourier 解析，合成 (2, 3 次元)
FFTC	複素高速 Fourier 解析
FFTR	実高速 Fourier 解析
FFTRI	実高速 Fourier 合成
FFTS	複素高速 Fourier 変換
FT235C	サンプル数が $2^K 3^L 5^M$ の形の高速 Fourier 解析
TRIGQP	２進逆順に並べられた三角関数表
VCHB1S	Chebyshev 級数
VCHB2S	第 2 種 Chebyshev 級数
VCOSS	cosine 級数, sine 級数

数値微積分

AQCHYS	Caucy 主値積分に対する自動積分
AQCOSS	振動する関数の半無限自動積分
AQCPACK	複素数値関数の自動数値積分
AQDCCS	Clenshaw-Curtis 法による自動積分
AQIOSC	複素数値関数の無限振動積分の自動積分
AQIOSS	半無限区間振動積分の自動積分
AQMDS	等差数列的に標本数を増す補間型積分にもとづく自動多重積分
AQNDS	自動多重数値積分
AQNN5S	Newton-Cotes 5,7,9 点則にもとづく適応型自動積分
AQOSCS	有限 Fourier 積分
DEFINS	二重指数関数型公式による有限区間積分
GASNS	Gauss 型数値積分
HINFAS	二重指数関数型公式による半無限区間積分
INFINS	二重指数関数型公式による無限区間積分
MQFSRS	完全対称則による多重数値積分
MQNCDS	Newton-Cotes 則の直積による多重数値積分
MQPRRS	直積型公式による多重数値積分
QDAPBS	等差数列的標本点を増す補間型積分法
ROMBGS	Romberg 積分
TNCOTS	数値積分公式のための重率と分点の値
TRAPZS	台形則による無限区間積分

常微分方程式

ODEBSS	有理補外法による連立１階常微分方程式の初期値問題の解法
RK4S	４次古典的 Runge-Kutta 法による連立１階常微分方程式の初期値問題の解法
RKF4AS	Runge-Kutta-Fehlberg ４次法による連立１階常微分方程式の初期値問題の解法
RKM4AS	Runge-Kutta-Merluzzi ４次法による連立１階常微分方程式の初期値問題の解法

特殊関数

ABRM0	Abramowitz 関数（0 ～ 2 次）
ABRMW	整数次 Abramowitz 関数
ACND	累積正規分布関数と余関数の逆関数
AERF	誤差関数および余関数の逆関数
AICGAM	不完全ガンマ関数
BETIC	不完全ベータ積分
BLAS	Blasius 方程式の解と導関数
CELI1	第 1, 2 種完全楕円積分
CGAMMA	複素変数のガンマ関数
CLASN	Clausen の積分
CND	累積正規分布関数と余関数
DAWSN	Dawson の積分
DEBYE	Debye の関数
DIGAM	Digamma 関数
DILOG	Dilogarithm
ERFC1	余誤差関数の積分
EXI	指数積分
FRESS	Fresnel 正弦，余弦積分
HYPGM	超幾何級数と合流型超幾何級数
ICEILS	第１，２種不完全楕円積分
JACELS	Jacobian 楕円関数
PN	ルジャンドル多項式およびルジャンドル陪多項式
QN	第２種ルジャンドル関数およびルジャンドル陪関数
QNOME	楕円 θ 関数の Nome
RGAMA	ガンマ関数の逆数
SI	正弦積分，余弦積分
SPENC	Spence 関数
TMFRM	Thomas-Fermi 方程式の解とその導関数
ZETA	Riemann Zeta 関数

初等関数

ALANGV	Langevin 関数
ALOG1	$\log(1+x)$
ASINH	逆双曲線関数
CABS1	$\|\|z\|\|_1 = \|x\| + \|y\|, \quad z = x + iy$
COMB	二項係数
EXP1	$e^x - 1$
FASTEE	e の高速高精度計算
FASTPI	π の高速高精度計算
SINHP	引数 $\frac{\pi}{2}x$ に対する三角関数

直交多項式

PLEGE	ルジャンドル多項式 $P_n(X)$
PLEGA	ルジャンドル陪関数 $P_n^m(X)$
PLEGN	規格化ルジャンドル陪関数 $\bar{P_n^m}(X)$
PCHB1	第１種チェビシェフ多項式 $T_n(X)$
PCHB2	第２種チェビシェフ多項式 $U_n(X)$
PLAGU	ラゲール多項式 $L_n(X)$
PLAGG	一般ラゲール多項式 $L_n^{(a)}(X)$
PHERM	エルミート多項式 $H_n(X)$

ベッセル関数

AI	Airy 関数と導関数
BER0	Kelvin 関数（0, 1 次）
BESJFC	複素変数の非整数次第 1 種 Bessel 関数
BESJNC	複素変数の整数次第 1 種 Bessel 関数
BESKNC	複素変数の整数次第 2 種変形 Bessel 関数
BESYNC	複素変数の整数次第 2 種 Bessel 関数
BH0	Struve 関数（0, 1 次）
BI0	変形 Bessel 関数（0, 1 次）
BI0I0	変形 Bessel 関数の積分
BI0ML0	変形 Bessel 関数と変形 Struve 関数の差（0, 1 次）
BIF	非整数次第 1 種変形 Bessel 関数
BIN	整数次変形 Bessel 関数
BJ0	Bessel 関数（0, 1 次）
BJ0I0	Bessel 関数の積分
BJF	非整数次第 1 種 Bessel 関数
BJN	整数次 Bessel 関数
BKF	非整数次第 2 種変形 Bessel 関数
BL0	変形 Struve 関数（0, 1 次）
BYF	非整数次第 2 種 Bessel 関数
J0Y0S	Bessel 関数（0, 1 次）
SI0	変形球 Bessel 関数（0, 1 次）
SIK	整数次変形球 Bessel 関数
SJ0	球 Bessel 関数（0, 1 次）
SJN	整数次球 Bessel 関数
ZBJ0	$J_0 \sim J_{15}$ の正の零点
ZBJ0S	Bessel 関数 J_0, J_1 の零点および導関数
ZBJN	$J_0 \sim J_{15}$ の正の零点

その他

SETPACK	集合演算プログラムパッケージ
SORTPACK	スカラー，ベクトルデータの内部ソーティング
BITLOGIC	4-Byte データ間のビットごとの論理演算
IBITCT	4-Byte データの 2 進表示での 1 のビット数の数え上げ
IBITRV	4-Byte データのビットパターンの逆順並べ換え
IGCD	二つの整数の最大公約数
PRIME	素数表の作成
PRMFAC	整数の素因数分解
RANDOM	一様乱数の生成
ROUND	実数の０捨１入

参考文献

[1] UXP/M FORTRAN77 EX 使用手引書 V12 用, 94SP–5010, 富士通株式会社 (1991).

[2] UXP/M FORTRAN77 EX/VP 使用手引書 V12 用, 94SP–5030, 富士通株式会社 (1991).

[3] UXP/M アナライザ使用手引書 (FORTRAN, VP 用) V10L20 用, 94SP–5083, 富士通株式会社 (1992).

[4] UXP FORTRAN77 EX メッセージ説明書 V12 用, 93SP–5010, 富士通株式会社 (1991).

[5] FORTRAN VP プログラミング手引書, 99SP–0080, 富士通株式会社 (1990).

[6] 佐藤 周行 : コンピュータネットワークのせいで不便になること, 九州大学大型計算機センター広報, Vol.27, No.2, 93–110 (1994).

[7] SSL II 使用手引書 (科学用サブルーチンライブラリ), 99SP–4020, 富士通株式会社 (1987).

[8] SSL II 拡張機能使用手引書 (科学用サブルーチンライブラリ), 99SP–4070, 富士通株式会社 (1991) .

[9] NUMPAC 利用手引書, 富士通株式会社 (1994) .

[10] 電子メール講習会資料, 九州大学大型計算機センター (1996).

[11] 渡部 善隆 : 数値計算と速度の話, 九州大学大型計算機センター広報, Vol.28, No.3, 207–231 (1995).