VPP700/56 利用の手引

渡部 善隆[†]

本書は九州大学大型計算機センターのスーパーコンピュータシステム FUJITSU VPP700/56 の利用方法を解説したものです。 1997 年 2 月発行の『VPP700/56 利用の手引 (第 1 版)』からのもっとも大きな変更点は、VPP700/56 に直接接続しての対話的な処理が可能になったことです。そのため第 2 版では、対話型処理の章を一章追加し、利用例も対話型処理にあわせて修正しました。また、第 1 版発行以降のジョブ制限値の追加・変更などにも対応しています。その他、 Fortran, C コンパイラ, SSL II, Analyzer などの機能追加の説明、(文章の) バグの修正を行ないました。

この本が少しでも皆様の研究に役立つことを願っております。

目次

[†]九州大学大型計算機センター;
e-mail:watanabe@cc.kyushu-u.ac.jp;
http://www.cc.kyushu-u.ac.jp/RD/watanabe/index-j.html

1　VPP700/56 の概要

九州大学大型計算機センターのスーパーコンピュータシステム FUJITSU VPP700/56 は，大規模数値計算を高速に処理するためのベクトル並列型計算機です．この章では，VPP700/56 の概要を，今後の章で使われるキーワードの説明を織り込みながら紹介します．

1.1　スーパーコンピュータ VPP700/56

1.1.1　並列計算機

まず用語の定義です．

並列計算機の個々のプロセッサを **PE**(*Processing Element*) と呼びます．

現在の並列計算機はメモリ (記憶域) の構成によって大きく二つに分類できます[1]．一つはすべての PE からアクセス可能なメモリを備える**共有メモリ** (*shared memory*) 型計算機，もう一つは各 PE がそれぞれメモリを持ち，他の PE のデータが必要になった場合は PE 間を相互に結合したネットワークを介して転送を行う**分散メモリ** (*distributed memory*) 型計算機です．

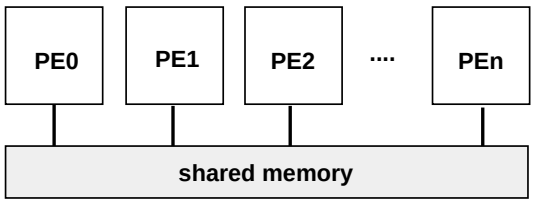


図 1.1: 共有メモリ型

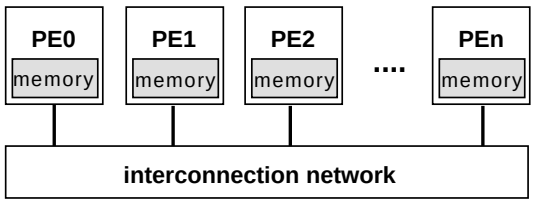


図 1.2: 分散メモリ型

前者の代表的な計算機は NEC の SX4，SGI/Cray の T90，後者の代表的な計算機は SGI/Cray の T3E，日立の SR2201 です．VPP700/56 は後者の分散メモリ型計算機に属します．

1.1.2　自動並列化技術の動向

共有メモリ型計算機の自動並列化技術はかなりの水準まで達しており，プログラムの作成や移行は比較的楽だといわれています．しかし，数百台規模の PE とメモリを効率的に結合するのはハードウェア的に難しいことから，次世代のスーパーコンピュータの主力となるかどうか疑問視されています．

一方の分散メモリ型計算機は，同じ性能を持つ PE をネットワークに接続することで比較的簡単に規模を拡張できることから[2]将来を期待されています．

しかし，肝心のソフトウェアレベルでの並列化技術はまだまだ研究段階で，利用者のプログラムを自動的に並列化する「自動並列化コンパイラ」の提供はしばらく先になります[3]．VPP700/56 で複数の PE を用いた並

[1] 並列計算機の有名な分類方法に，Flynn さんが考えた命令／データの流れによるアーキテクチャの分類があります．ベクトル並列処理の VPP700/56 は，その分類によると MIMD(Multiple Instruction stream, Multiple Data stream) 型に属します．

[2] カタログで宣伝するところの「スケーラビリティ」です．VPP700 も最大 512 台の PE が結合可能です．ただし，ものすごいお金と電気が必要ですので，まず誰も買わないでしょう．

[3] 並列計算機に対応したアプリケーションの提供もまだまだ数えるほどです．

列処理を実行したい場合は，現状，並列処理のための命令 (データ分割方法， PE 間のデータのやりとりなど) を利用者自身で記述する必要があります．つまり，残念なことですが，

VPP700/56 の並列プログラミングは利用者が行う必要があります．

しかし [6] によれば，並列化に関する「部品」は隠されずに公開されているそうです[4]．ということは，チューニングの「腕」が冴えることで理論ピーク性能に迫る “スーパーコンピューティング” の可能性も大いにあるわけです[5]．

1.1.3 VPP700/56 の外観

幅 5.164 メートル，奥行 5.751 メートル，高さ 1.8 メートル，重量 6.6 トンです．付属のディスクなどを合わせると 7 トンを超えます．

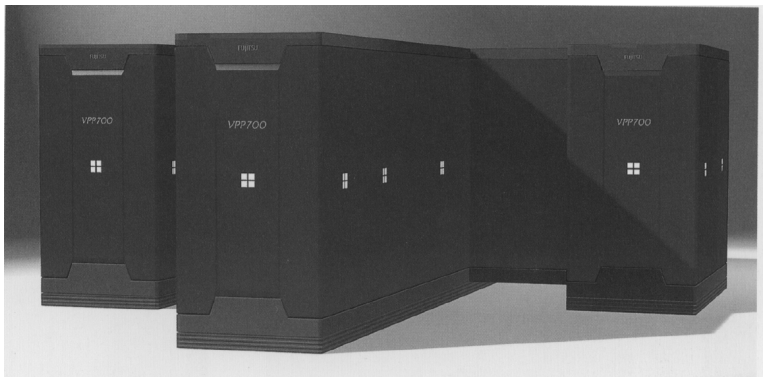


図 1.3: VPP700/56 の外観 (カタログより)

色はダークレッド[6]，発熱量は 69,815Kcal/h，所要電力は 65.6KVA です．

1.1.4 VPP700/56 のシステム構成

VPP700/56 のシステム構成は図 1.4 の通りです．独立したベクトル計算機が 56 台「ずらっ」と並列に並んでいるとイメージすることができます．

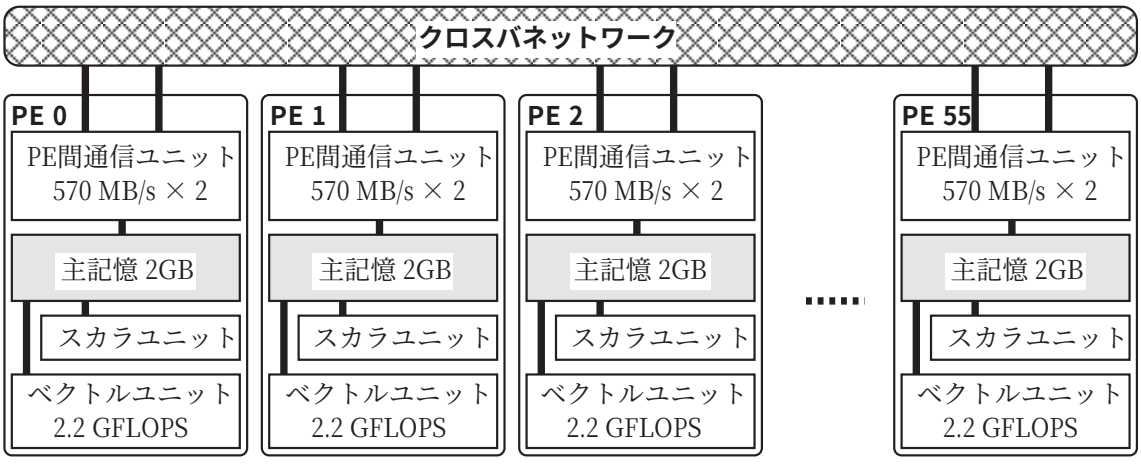


図 1.4: システム構成図

[4] 裏を返せば， HPF(High Performance Fortran) などよりも低水準な仕様ということです．

[5] 現在，並列計算のアルゴリズムやチューニング方法は，(計算機に依存したものであっても) 研究集会での発表のネタや，うまくすれば論文になる時代です．

[6] その他の色にダークブルーとチャコールブラックがあります．

各 PE は 2GB の主記憶と最大処理性能 2.2GFLOPS のベクトルユニットを搭載しています．1PE あたりの処理性能は VP2600/10 の半分程度ですが，VP2600/10(0.5GB) に比べ 4 倍のメモリを搭載しています．さらに，複数 PE(VP2600/10 は 1PE) でのジョブ処理により，スループットは格段に向上するはずです．

図 1.4 に出てくるクロスバネットワーク (crossbar network)とは，PE 間の接続形態の一つ (cf. 図 1.5) で，各 PE はこのネットワークにより相互に接続されています[7]．

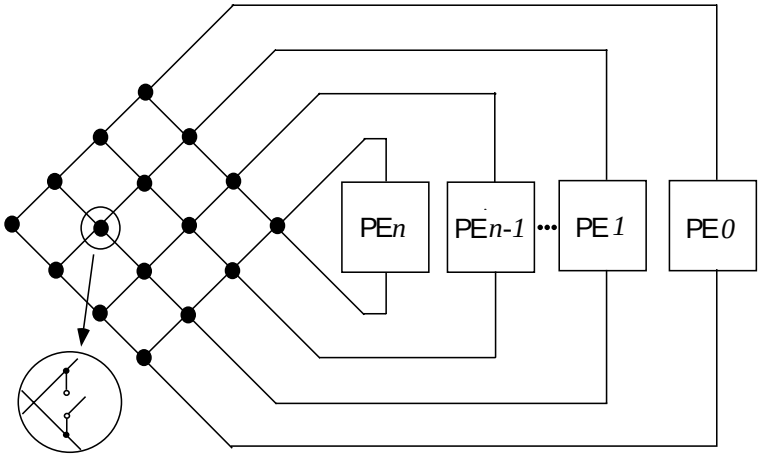


図 1.5: crossbar network 概念図

各 PE は最大 570MB/ 秒の性能を持つ PE 間通信ユニットを 2 台装備しています．

PE の諸元は表 1.1 の通りです．

表 1.1: VPP700/56 の主な諸元 (1PE あたり)

項目	性能
理論ピーク性能	2.2GFLOPS
ベクトルパイプライン数	7 本
浮動小数点レジスタ	64bit×32 個
汎用レジスタ	32bit×32 個
ベクトルレジスタ	128KB
キャッシュ	32KB×2 個
ネットワーク通信速度	570MB/ 秒 ×2
主記憶容量	2GB
記憶素子	SDRAM
主記憶スループット	18.2GB/ 秒

[7] 並列計算機の接続は，たくさんの形態が提案・開発されています．クロスバネットワークの他にも，トーラスメッシュ，多段接続網，階層リングなど並列計算機によってバラバラで決定版がまだありません．

1.2 VPP700/56 のソフトウェア

VPP700/56 で動作するソフトウェアは表 1.2 の通りです.

表 1.2: VPP700/56 のソフトウェア

ソフトウェア名	機能	備考
Fortran 90/VP	ベクトル Fortran コンパイラ	1PE 用
Fortran 90/VPP	ベクトル並列 Fortran コンパイラ	複数 PE 用
C	C コンパイラ	1PE 用
C/VP	ベクトル C コンパイラ	1PE 用
C++	C++ コンパイラ	1PE 用
Analyzer	チューニング・デバッグ支援	
VPP Workbench	プログラム統合開発環境	WS で動作
エンジニアリング開発環境	対話的なプログラム開発支援	WS で動作
SSL II/VPP	ベクトル並列科学技術計算ライブラリ	
SSL II/VP	ベクトル科学技術計算ライブラリ	
NUMPAC	数値計算ライブラリ	
PVM	メッセージパッシングライブラリ	
MPI	メッセージパッシングライブラリ	
PARMACS	メッセージパッシングライブラリ	
α-FLOW	3 次元流体解析システム	
MASPHYC	材料の物性・構造解析システム	
LS-DYNA	非線形動的構造解析システム	
AVS	汎用可視化システム	
MARC	構造解析システム	
Gaussian94	分子軌道計算	

1.3 VPP700/56 の利用方法

直接 VPP700/56 に接続して対話的に利用することができます. 大規模なジョブ, 並列ジョブは「バッチ処理」と呼ばれる処理方式による利用となります. バッチ処理とは, あらかじめ蓄積された必要なデータをひとまとめに処理するデータ処理の方式のことです[8].

また, 汎用計算機 M-1800/20U で作成したソースプログラムを VPP700/56 において (バッチ処理により) 翻訳・実行することもできます.

1.3.1 アカウントの発行

既にセンターの課題をお持ちの方は自動的に VPP700/56 が利用できるようになっています. センター課題をお持ちでない方は課題申請の必要があります. 詳しくは共同利用掛 (e-mail: kyodo@cc.kyushu-u.ac.jp, ダイヤルイン 092-642-2305) まで問い合わせてください.

[8] JIS の定義では『あらかじめ蓄えられたデータの処理またはジョブの実施であり, その進行中は利用者がもはやその処理に影響を及ぼすことができないような手段によるもの』です

1.4 マニュアル

1.4.1 オンラインマニュアル

man(/usr/uxp/man) コマンドで VPP700/56 のコマンド，手続きの詳細を調べることができます.

```
kyu-vpp% man frt ↵                    <---VPP700/56 の frt コマンドの参照

  frt(1)                      (UXP/V Fortran90/VP)                      frt(1)

   【名前】
        frt ― Fortran90/VP コンパイラ

   【形式】
        frt   [ -c ] [ -g [dbg_lvl] ] [ -o exe_file ] [ -I dir_nam ]
              [ -v ] [ -X lan_lvl ] [ -A alignc ] [ -A byname ]
              [ -A double ] [ -A quad ] [ -A ihalf ] [ -A errssn ]
                        :
```

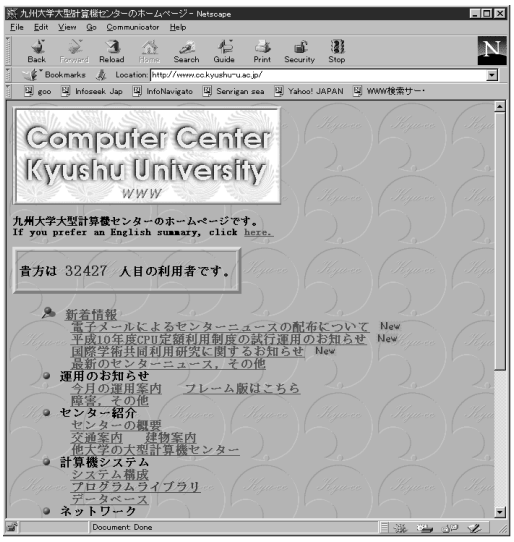
九州大学各キャンパスに設置されたユーザインタフェースワークステーションには，オンラインマニュアル参照用のブラウザ “OLIAS” が提供されています[9]．使用方法は [12] を参照してください.

また，センターホームページ (http://www.cc.kyushu-u.ac.jp/) からオンラインマニュアルを参照することもできます.

1.4.2 マニュアルの購入方法

九大生協の書籍部で富士通株式会社発行のマニュアルを注文することができます．注文の際はマニュアル番号 (例えば『Fortran90/VP 使用手引書』なら “J2U5-0050”) を必ず指定してください.

参考文献にあげたマニュアルは，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.



http://www.cc.kyushu-u.ac.jp/
(1998 年 5 月 1 日現在)

[9] OLIAS のネットワーク経由での利用はできません.

2 対話型処理

1997 年 4 月より VPP700/56 に直接アクセス (login) できるようになりました．この章では VPP700/56 での対話型処理[10]について説明します．

2.1 対話型処理の概要

2.1.1 kyu-vpp

VPP700/56 は PE(Processing Element) が 56 台あります．その中で利用者が直接手元のワークステーションやパーソナルコンピュータからアクセスできる PE は 1 台のみです[11]．ホスト名は “kyu-vpp” です．

表 2.1: 対話型処理の環境

マシン名	FUJITSU VPP700/56
ホスト名	kyu-vpp
IP アドレス	133.5.9.70
OS	UXP/V(UNIX SVR4 準拠)

UXP/Vはコマンドの若干の非互換を除けば汎用計算機の UXP/M(kyu-cc, IP アドレス 133.5.9.1) とほぼ同じ仕様の UNIX OS です． UXP(UNIX) に関する基本的なことがらは [53] を参照してください．

2.1.2 制限値

対話型処理の制限値は表 2.2 の通りです．

表 2.2: 対話型処理の制限値

メモリサイズ	100MB
ファイルサイズ	2GB
CPU 時間	60 分

その他の制限値は `limit` コマンドで調べることができます．

2.1.3 利用できるライブラリ

対話型処理で利用できるライブラリは表 2.3 の通りです． Fortran 90/VP, C/VP などの 1PE で動作するプログラムが実行可能です．並列ライブラリは実行可能ファイルの作成までです．

[10] UNIX の世界で「対話型処理」と言えば “親切なメニュー画面とのやりとり” などのイメージを持つ方もいらっしゃると思います．ここでの「対話型処理」とは，次章で説明するジョブスクリプトを記述し処理を依頼する「バッチ処理」とは異なり，入力に対しリアルタイムで応答を得る (現在ではあたりまえの) 処理形態と定義します． MSP に慣れている方には “TSS(Time Sharing System)” の方が分かりやすいかと思います．

[11] “Primary PE” と呼びます．その名の通り，すべての PE の動きを統括する PE です．

表 2.3: 対話的に利用できるライブラリ

ソフトウェア名	機能	備考	コマンド / リンク方法
Fortran 90/VP	ベクトル Fortran コンパイラ		`frt`
Fortran 90/VPP	ベクトル並列 Fortran コンパイラ	翻訳のみ	`frt -Wx`
C	C コンパイラ		`cc`
C/VP	ベクトル C コンパイラ		`vcc`
C++	C++ コンパイラ (C への変換プロセッサ)		`CC`
Analyzer	チューニング・デバッグ支援	1PE のみ	`fjsamp`
SSL II/VPP	ベクトル並列科学技術計算ライブラリ	結合のみ	`-lssl2vpp`
SSL II/VP	ベクトル科学技術計算ライブラリ		`-lssl2vp`
NUMPAC	数値計算ライブラリ		`-lnumpac`
PVM	メッセージパッシングライブラリ	結合のみ	`-lpvm -lmp` (etc.)
MPI	メッセージパッシングライブラリ	結合のみ	`-lmpi -lmp` (etc.)
PARMACS	メッセージパッシングライブラリ	結合のみ	`-lpm6 -lmp2` (etc.)
Gaussian94	分子軌道計算		`g94`, `subg94`

残念ながら対話的に並列プログラム (Fortran 90/VPP, MPI, etc.) は実行できません．`frt -Wx` などで並列処理の実行ファイルを作成することは可能です．しかし並列実行はバッチ処理となります．また 1PE のプログラムでも 100MB の制限値を超える場合はバッチ処理となります．

VPP700/56 でサポートしているアプリケーションライブラリ α-FLOW, MASPHYC, LS-DYNA, AVS は，フロントエンドのワークステーションからジョブを投入するというもともとの性格から，対話型での利用はできません．また， MARC は実行ファイルが制限値を超えるためバッチ処理となります． Gaussian94 の利用方法は 8.6節を御覧ください．

2.2 kyu-vpp への接続

以降，利用者は a79999a さんとします．

2.2.1 login

kyu-vpp に login します．以下は IP 接続されたワークステーションからの login の例です． ↵ はリターンキーを押し下げることを意味します．

```
user-ws% telnet 133.5.9.70 ↵          <---kyu-vpp に接続
Trying 133.5.9.70 ...
Connected to 133.5.9.70.
Escape character is '^]'.

UXP/V TELNET (kyu-vpp)

login: a79999a ↵                      <--- 課題番号
Password:      ↵                      <--- パスワード
Fujitsu UXP/V (kyu-vpp)
Copyright (c) 1984, 1986, 1987, 1988 AT&T
Copyright (c) 1990, UNIX System Laboratories, Inc.
Copyright (c) 1991, 1992, 1993, 1994, 1995 FUJITSU LIMITED
All Rights Reserved
Last login: Tue Apr 22 16:00:25 on user-ws.cc.kyus
Terminal Type: ↵
```

login した後は passwd(/bin/passwd) コマンドで kyu-vpp 独自のパスワードに変更できます.

2.2.2 シェル

デフォルトのログインシェルは csh(/usr/bin/csh) です. tcsh(/usr/local/bin/tcsh) も利用可能です.

2.2.3 エディタ

vi(/bin/vi)と emacs(/usr/local/bin/emacs)が利用できます. ただし, 現在のところ emacs で日本語入力はできません[12].

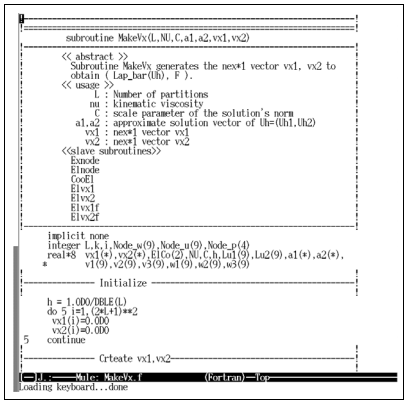


図 2.1: emacs でのプログラム作成画面

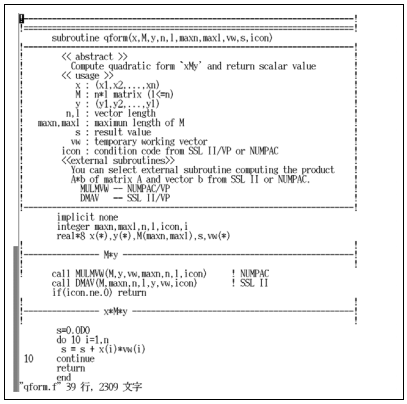


図 2.2: vi でのプログラム作成画面

2.2.4 kyu-cc と kyu-vpp との関係

汎用計算機 M-1800/20U の UXP(ホスト名 kyu-cc) の利用者のホームディレクトリの “VPP” というディレクトリと VPP700/56 の利用者ホームディレクトリとの間にシンボリックリンクが張られています (図 2.3).

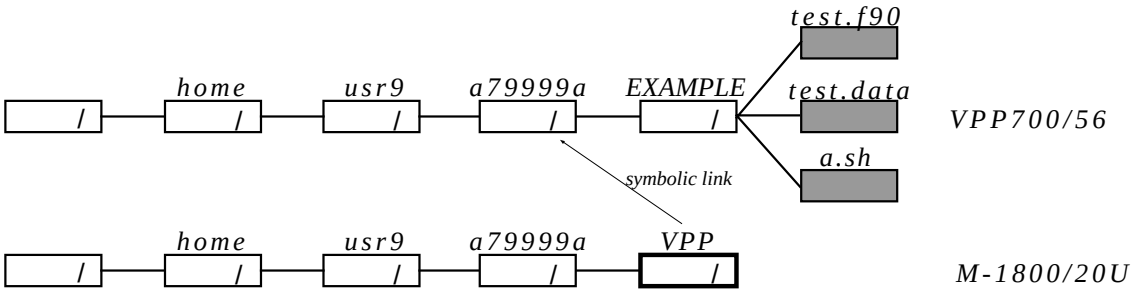


図 2.3: kyu-cc と kyu-vpp のファイルの共有

つまり, kyu-cc の a79999a さんから見える

[12] 端末側の日本語環境, kyu-cc 経由で Wnn を用いての入力は可能です. 日本語の表示は問題ありません. 漢字コードは euc です.

```
kyu-cc% cd ~/ ↵                          <---kyu-cc のホームディレクトリに移動
kyu-cc% ls ↵                             <--- 内容の表示
Interval_Arithmetic  VPP                 manual
MASPHYC              bin                 mspcom
Navier-Stokes        f90                 paper
News                 image               RMAIL
kyu-cc% cd VPP ↵                         <--- ディレクトリ VPP へ移動
kyu-cc% ls ↵                             <--- 内容の表示
Analyzer       DEMO          MPI            PARMACS
C              Fortran       Navier-Stokes  library
```

と， kyu-vpp の a79999a さんのホームディレクトリから見える

```
kyu-vpp% cd ~/ ↵                         <---kyu-cc のホームディレクトリに移動
kyu-vpp% ls ↵                            <--- 内容の表示
Analyzer       DEMO          MPI            PARMACS
C              Fortran       Navier-Stokes  library
```

は同じものです．

kyu-cc から VPP700/56 へバッチジョブを投入する場合，使用するソースプログラムおよびデータは必ず kyu-cc から見たVPP 配下 (即ち kyu-vpp から見える場所) に作成する必要があります．

kyu-vpp のファイルは kyu-cc からすべて見ることができます．従って kyu-vpp のファイルを kyu-cc に転送することなく[13]kyu-cc 側から編集したり， Graphman や gnuplot 等の入力データとして利用することができます．

kyu-vpp から `lp(/bin/lp)` コマンドによりファイルの内容をセンター 2 階のネットワークプリンターに出力することができます[14]．`utoprint` コマンドは利用できません．

2.2.5 オンラインマニュアル

`frt`, `vcc` などのコマンドは `man(/usr/uxp/man)` コマンドで機能，オプションを確認できます．日本語環境[15]の場合，多くのマニュアルは日本語で表示されます．表示を英語にする場合は “`unsetenv LANG`” と入力します．日本語環境に戻す時は “`setenv LANG japan`” と入力します．

2.3 実行ファイルを作成するまで

Fortran, C, C++ のソースプログラムが計算機で実行可能な**実行ファイル**[16]に姿を変えるには，**翻訳**および**結合・編集**のステップを踏みます．通常，`frt` コマンドや `vcc` コマンドなどを使う場合は，ソースプログラムを解釈し実行ファイルを作成するまでの手順は自動的に行われます．

ただし自分専用のライブラリを作ったり C から SSL II/VP を呼び出したりする場合は，翻訳のみが終了し結合・編集処理が行なわれていない**オブジェクトファイル**[17]というものがあることを知っておく必要があります．例えば SSL II/VP は連立 1 次方程式や FFT などの Fortran サブルーチン群を翻訳して出来あがったオブジェクトファイルの集合です．

オブジェクトファイルは編集・結合の際に必要に応じてメインプログラムから呼び出されることで，実行ファイルを構成する一部となります (図 2.4)．

[13]本当は裏で転送しているのですが
[14]kyu-cc 側から kyu-vpp のファイルを出力する場合は `lp -c` と入力します．
[15]`env` と入力することにより，環境の一覧が表示されます．
[16]計算機科学の用語では，実行可能オブジェクト (executable object) と呼びます．
[17]正確には，再配置可能オブジェクト (relocatable object)．

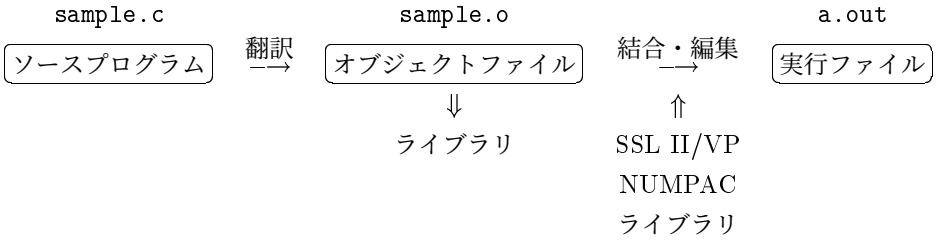


図 2.4: 実行ファイルの作成行程

2.4 Fortran の利用方法

2.4.1 サフィックス

言語仕様が Fortran 90の場合は “.f90”, FORTRAN 77の場合は “.f” として下さい．サフィックスが “.f” のファイルに Fortran 90 のプログラムを記述する場合は，固定長形式と見なされます．翻訳時オプションによって切替えることもできますが，サフィックス[18]で使い分けることをお勧めします．

2.4.2 frt コマンド

kyu-vpp の Fortran のコマンドは `frt(/usr/lang/bin/frt)` です． kyu-cc には 2 つのコマンド (`frt`, `frtex`) がありますが， kyu-vpp では `frt` のみです．

```
kyu-vpp% frt test.f90 ↵          <---Fortran プログラムの翻訳
```

Fortran 90 のプログラム `test.f90` を翻訳しました．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[19]．プロンプト (何も設定しなければ `kyu-vpp%`) が出た状態で `a.out` と入力すると，プログラムが実行されます．

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

2.4.3 Fortran 90/VPP での翻訳

`-Wx` オプションの指定により Fortran 90/VPP が起動され，並列プログラムの翻訳を行います．

```
kyu-vpp% frt -Wx test.f90 ↵      <---Fortran 90/VPP での翻訳
```

`-Wx` オプションはソースプログラムに挿入された拡張最適化制御行 (`!XOCL` 行) を有効にするものです．自動並列化を行うオプションではありません．また，作成された実行ファイルを対話的に実行することもできません．ただし対話型処理により Fortran 90/VPP の文法レベルでのデバッグが容易にできることは利点です．

[18] suffix: 末尾に添えたもの，付加したもの．
[19] 正確には，オブジェクトファイルを結合・編集するプログラム `ld` が起動され，実行ファイルが生成されます．

```
kyu-vpp% frt -Wx test.f90 ↵        <---Fortran 90/VPP での翻訳

              Fortran90/VPP  V10L10                日付 97-04-22  時刻 22:03:41

Fortran90/VPP 診断メッセージ：プログラム名 (dp_vcre_test)
  joo4622i-s  ‘‘test.f90’’: <spread-do-stmt> に対応する <end-spread-do-stmt> が存在しません.
                      :
```

2.4.4 プログラムが FORTRAN 77 の自由形式の場合

Fortran プログラムが FORTRAN 77 の自由形式の場合は，-Free オプションを指定します[20].

```
kyu-vpp% frt -Free job1.f ↵        <---FORTRAN 77 の自由形式プログラムの翻訳
```

2.4.5 実行ファイル名を変更する

frt 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 に続けてファイル名を指定します.

```
kyu-vpp% frt -o b.out test.f90 ↵        <--- 実行ファイル名を b.out に変更
```

2.4.6 ベクトル化メッセージを出力する

オプション -Wv,-m3 でベクトル化に関する翻訳情報が表示できます．省略値で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-vpp% frt -Wv,-m3 -Z out test.f90 ↵      <--- 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なります.

```
kyu-vpp% frt -Ps -Wv,-m3 -Z out test.f90 ↵
                                          <--- プログラムリストに対応したベクトル化情報を表示
```

2.4.7 最適化オプション

省略値の最適化オプション (-Oe) で翻訳した場合，極まれにですがプログラムの実行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 (cf.[1])．最適化の副作用は最適化レベルを下げることで回避できます．実行時間は一般に増大します.

```
kyu-vpp% frt -Ob test.f90 ↵        <--- 基本的な最適化にとどめる
```

サブオプションを使った細かい制御も可能です.

```
kyu-vpp% frt -Oe,-P test.f90 ↵     <--- 不変式の先行評価の最適化を抑止
```

[20] kyu-cc の frt コマンドの -F オプションと異なります.

2.4.8 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．診断メッセージは 実行時 に出力されます．思わぬミスが見つかったりして，バグ取りに大変有効です．

また，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-vpp% frt -Dasux -Wv,-ad test.f90 ↵     <--- デバッグオプションの指定例
kyu-vpp% a.out ↵                           <--- 実行

 jwe0320i-w line 6 配列要素または文字部分列（M）の引用で，添字式または部分列式の
値（334,1001）は，宣言した範囲（1:1000,1:1000）内でなければなりません．
  error occurs at MAIN__   line 6 loc 00000478 offset 000001e8
  MAIN__       at loc 00000290 called from o.s.
 taken to (standard) corrective action, execution continuing.
                    :
```

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

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るため実行時間が増加 (数倍) します．そのため，デバッグオプションは開発中の 小規模なプログラム に対し指定することをおすすめします．

2.4.9 オブジェクトファイルの作成

-c オプションは翻訳のみを行いオブジェクトファイルを出力するオプションです．オブジェクトファイルはサフィックスが “.o” となります．

```
kyu-vpp% frt -c sub.f90 ↵          <--- オブジェクトファイルの作成
```

既にデバッグが完了したサブルーチンなどをファイルとして保存・翻訳しオブジェクトファイルとして管理すると，パラメータを修正する度にいちいち翻訳する手間を省くことができ，効率的です．

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-vpp% nm sub.o ↵          <--- 名前リストの出力
Symbols from sub.o:

[Index]   Value      Size    Type  Bind  Other Shndx   Name

[1]     |         0|       0|SECT |LOCL |0    |1      |
[2]     |         0|       0|OBJT |LOCL |0    |1      |$S0000001
                           :
[117]   |         0|    1832|FUNC |GLOB |0    |2      |setdat_
[118]   |         0|       0|NOTY |GLOB |0    |UNDEF  |g_datan
[119]   |         0|       0|NOTY |GLOB |0    |UNDEF  |g_dsqrt
[120]   |         0|       0|NOTY |GLOB |0    |UNDEF  |w_dsin
[121]   |      1840|    3000|FUNC |GLOB |0    |2      |mamul_
                           :
```

2.4.10 ライブラリの作成

自分で作成した手続き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ファイルを「アーカイブファイル」と呼びます[21]．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．主なオプションは表 2.4 の通りです．

表 2.4: ar コマンドの主なオプション

形式	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

```
kyu-vpp% frt -c sub1.f90 sub2.f90 ↵                <--- 翻訳
kyu-vpp% ar rv libmylib.a sub1.o sub2.o ↵          <--- アーカイブファイルの作成
a - sub1.o
a - sub2.o
UX:ar: 情報: libmylib.a を作成しています
```

手続きプログラムを記述した sub1.f90, sub2.f90 を翻訳し，オブジェクトファイル sub1.o, sub2.o を作成します． n 次に ar コマンドによってアーカイブファイル libmylib.a を作成します．なお，並列化オプション -Wx を指定して作成したライブラリは，指定しないライブラリと独立に管理してください．

2.4.11 自分用のライブラリの結合

-L に続けてアーカイブファイルのあるディレクトリをフルパスで，さらに -l に続けてライブラリ名を指定します．

```
kyu-vpp% frt test.f90 -L/G/kyu-vpp-pe16/usr9/a79999a/MYLIB -lmylib ↵
```

ライブラリ libmylib.a を結合します．ライブラリは kyu-vpp の ~/MYLIB にあるとします．アンダーラインは利用者によって異なります．この部分は以下のように知ることができます．

```
kyu-vpp% pwd ↵
/G/kyu-vpp-pe16/usr9/a79999a/MYLIB        <--- フルパスの表示
```

/G/kyu-vpp-pe*(“*” は利用者によります) は /home にリンクが張られています．
従って /G/kyu-vpp-pe16/usr9/a79999a/MYLIB は /home/usr9/a79999a/MYLIB と見ることもできます．
なお，検索パス LD_LIBRARY_PATH を設定することもできます．

```
kyu-vpp% setenv LD_LIBRARY_PATH /G/kyu-vpp-pe16/usr9/a79999a/MYLIB ↵ <--- 検索パスの指定
kyu-vpp% frt test.f90 -lmylib ↵                                      <--- ライブラリを結合
```

[21] “archive” は文書や情報などを保管する「記録保管所」の意味です．

2.4.12 複数のファイルを分割処理

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.

```
kyu-vpp% frt main.f90 test1.f90 test2.o ↵      <--- 分割処理
```

2.4.13 モジュールの引用

モジュールを USE 文で引用する際に，もしモジュールを引用箇所より後に記述している場合は -Am オプションをつけます．モジュールが引用箇所より前に記述している場合は必要ありません.

```
kyu-vpp% frt -Am main.f90 ↵
```

モジュールが別のファイル・ディレクトリにある場合は注意が必要です． [1] の 4.5 節を参照してください.

2.4.14 リダイレクション機能

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 の入出力に関する装置参照番号 (論理機番) の 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読み込む

出力ファイルが存在しない場合は新規に作成されます.

2.4.15 標準入力からデータを読み込む

```
kyu-vpp% a.out < in.data ↵
```

例では装置参照番号 5 番 (標準入力) からデータを読み込みんでいます．データファイル名は in.data としています．入力ファイルを指定しない場合は端末からの入力となります.

2.4.16 標準出力にファイルを指定

```
kyu-vpp% a.out > out.data ↵                    <---out.data にデータを出力
```

装置参照番号 6 番 (標準出力) にデータを書き出す例です．リダイレクションの方向に注意してください．追加書きする場合は >> とします.

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

2.4.17 標準入出力の例

```
kyu-vpp% a.out < in.data > out.data ↵
```

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．

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切なデータを上書きしてしまう危険性があることにも注意してください．

2.4.18 一般のファイル入出力

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，

1. プログラム中にファイル名を書く
2. 環境変数として設定する

かのどちらかです．

◆プログラム中にファイル名を指定する

open 文中では “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，フルパスなどファイルがきちんと見えるように指定します．

```
      open(1,file='/G/kyu-vpp-pe16/usr9/a79999a/DATA/in.data')
```

◆環境変数としてのファイル処理

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

$$\texttt{fu}nn\texttt{␣}\textit{file-name}$$

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．

```
kyu-vpp% setenv fu01 ex1.data ↵        <--- 装置参照番号 1 番に ex1.data を割り当てる
kyu-vpp% a.out ↵                       <--- 実行
  :
kyu-vpp% unsetenv fu01 ↵               <--- 割り当てを解除
```

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．複数の装置参照番号を使用する場合も同様です．

2.4.19 IEEE 形式でファイル処理

VPP700/56 の浮動小数点形式はワークステーションで広く使われている IEEE 形式と呼ばれるものです．一方，汎用計算機 M-1800/20U の浮動小数点形式は M 形式 (IBM 形式)と呼ばれるものです． M-1800/20U で作成した M 形式のバイナリデータを VPP700/56 で読み込んだり， VPP700/56 の出力する IEEE 形式のバイナリデータを M-1800/20U で読み込んだりする場合には，2 つの形式を実行時オプションによって変換する必要があります．

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．

-Wl,-C*uno*	装置参照番号 *uno* からのバイナリデータの入出力を M 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．
-Wl,-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．

```
kyu-vppo% b.out -Wl,-C,-M  ↵              <---b.out を実行．入出力は M 形式
```

あらかじめ作成した b.out を実行する際に，実行時のバイナリデータの入出力を M 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．

装置番号 88 番の入出力だけを M 形式で行う場合は，次のように指定します．

```
kyu-vpp% b.out -Wl,-C88,-M  ↵             <---b.out を実行．装置番号 88 番の入出力は M 形式
```

2.4.20 SSL II/VP の結合

SSL II/VP のサブルーチンを使用する場合は -lssl2vp を指定します．

```
kyu-vpp% frt test.f90 -lssl2vp  ↵     <--- 翻訳．SSL II/VP を結合
```

2.4.21 NUMPAC の結合

NUMPAC のサブルーチンを使用する場合は -lnumpac を指定します．

```
kyu-vpp% frt test.f90 -lnumpac  ↵      <--- 翻訳．NUMPAC を結合
```

2.4.22 SSL II/VPP の結合

SSL II/VPP のサブルーチンを使用する場合は -lssl2vpp を指定します．並列化指示オプション -Wx も必ず指定します．

```
kyu-vpp% frt -Wx test.f90 -lssl2vpp  ↵    <--- 翻訳．SSL II/VPP を結合
```

ただし並列プログラムの実行はできません．

2.5 C, C++ の利用

C, C++ プログラムも実行ファイル名はデフォルトで “a.out”，オブジェクトファイルのサフィックスは “.o” です．分割コンパイルや自分用のライブラリの作成方法は Fortran と同じです．

2.5.1 コマンド

C/VP は vcc(/usr/lang/bin/vcc)， C は cc(/usr/ccs/bin/cc)， C++ は CC(/usr/lang/bin/CC)です．各オプションは manで検索できます．

2.5.2 C/VP での翻訳

ベクトル版 C の起動は vcc コマンドです．

```
kyu-vpp% vcc test.c ↵                    <--- 翻訳
kyu-vpp% a.out ↵                         <--- 実行
```

```
kyu-vpp% vcc -Wv,-m3,-Ps test.c ↵        <--- 翻訳
kyu-vpp% a.out ↵                         <--- 実行
```

-Wv,-m3 は Fortran と同じくベクトル化に関するメッセージを出力します．-Wv,-Ps は Fortran の -Ps と同じく，ベクトル化表示付きのソースリストを出力します．frt との違いは，ベクトル化オプション-Wv の中に -Ps が組み込まれたことと，プログラムリストの出力先が frt では標準エラー出力であるのに対し，vcc では標準出力となることです．

2.5.3 最適化レベルを上げる

```
kyu-vpp% vcc -O -K4 test.c ↵
```

-O と -K4 の組合せで最大限の最適化となります．実行時間の短縮が期待される反面，翻訳時間の増加と最適化による精度誤差および異常終了の可能性があります (cf.[7])．

2.5.4 翻訳情報の出力

frt と同様に，翻訳情報を指定したファイルに出力する -Z オプションが追加されました．-Z の後空白に続けて任意のファイル名を指定します．

```
kyu-vpp% vcc -Z cout sample.c ↵          <--- 翻訳情報の出力
```

2.5.5 SSL II/VP の結合

C プログラムから SSL II/VP を利用する場合は，vcc コマンドによってオブジェクトファイル test.o を作成した後に，frt コマンドにライブラリをリンクする -lssl2vp -lcvp -lm を付加して実行ファイルを作成します．

```
kyu-vpp% vcc -c test.c ↵                          <--- 翻訳．オブジェクトファイルを出力
kyu-vpp% frt test.o -lssl2vp -lcvp -lm ↵          <--- 実行ファイルの作成
kyu-vpp% a.out
```

SSL II/VP を呼び出す場合の C プログラミングの注意点は [49] を参照してください．

2.5.6 C での翻訳

スカラー版 C の起動は cc コマンドです．オプションはベクトル化オプション -Wv 以外 vcc とほぼ共通です．

```
kyu-vpp% cc -O test.c ↵                           <--- 翻訳
```

最適化レベルを上げた (-O) 翻訳例です．

2.5.7 C++ での翻訳

C++ の起動は CC コマンドです．CC は C++ プログラムを一度 C プログラムに変換します．

```
kyu-vpp% CC test.C -lm ↵        <--- 数学関数を使う場合は -lm を指定
```

通常は自動的にスカラー版の cc が起動されますので，実行はスカラー処理です．

ベクトル処理を行う場合は，一度 C のソースに変換したものを保存し vcc コマンドによって処理することでベクトル処理可能な実行ファイルが作成できます．下線部分は必ず指定します．

```
kyu-vpp% CC -Fc test.C > test.c ↵                 <---C のソースに変換
kyu-vpp% vcc -Wl,-L/usr/uxp/C++/lib test.c -lC ↵  <---vcc コマンドによる翻訳
```

2.6 メッセージパッシングライブラリ

メッセージパッシングライブラリは対話的な結合のみ可能です．並列実行はできません．

2.6.1 PVM の例 (Fortran)

PVM は C 言語または Fortran 言語の関数およびサブルーチンの集合です．従って結合・編集時にデータ通信を制御するライブラリを読み込む必要があります．

```
kyu-vpp% frt test.f90 -Wl, -P -L. -J -dn -lpvm -lmp -lelf -lpx -lc -I. ↵
```

2.6.2 PVM の例 (C)

```
kyu-vpp% cc test.c -Wl, -P -L. -J -dn -lpvm -lmp -lelf -lpx -lc -I. ↵
```

ライブラリの指定順序には細かい規定があります．詳細は [13] を参照下さい．

2.6.3 MPI の例 (Fortran)

```
kyu-vpp% frt test.f90 -Wl,-J,-P -L/usr/lang/mpi/lib -lmpi -lmp -lelf -lpx \ ↵
? -I/usr/lang/mpi/include ↵
```

“\” はコマンドの継続を意味します.

2.6.4 MPI の例 (C)

```
kyu-vpp% cc -K a4 test.c -Wl,-J,-P -L/usr/lang/mpi/lib -lmpi -lmp -lelf -lpx \ ↵
? -I/usr/lang/mpi/include ↵
```

詳細は [14] を参照下さい.

2.6.5 PARMACS の例 (Fortran)

```
kyu-vpp% frt -o host  -Wl,-J,-P,-t host.o node.o other_objects.o other_libs.a \ ↵
? -L.../parmacs/lib/fujitsu-vpp700/VPP700 -lpm6 -lmp2 -lpx -lm -lelf -lc host ↵
```

2.6.6 PARMACS の例 (C)

```
kyu-vpp% cc -o host -Wl,-J,-P,-t host.o node.o other_objects.o other_libs.a \ ↵
? -L.../parmacs/lib/fujitsu-vpp700/VPP700 -lpm6 -lmp2 -lpx -lm -lelf -lc host ↵
```

詳細は [15] を参照下さい.

また, PARMACS の README ファイルを /usr/lang/parmacs/doc/README.VPP700 で公開しています.

2.7 便利なコマンド集

kyu-vpp での対話型処理の際に知っておくと便利なコマンドをいくつか紹介します.

2.7.1 file

file(/bin/file) コマンドはファイルのタイプを調べるコマンドです. 実行ファイルが並列用なのか 1PE 用なのか調べる場合に有効です.

```
kyu-vpp% file b.out  ↵
b.out:          ELF 32 ビット MSB 実行可能 VPP UXP/V Version 1 Vector-EX
kyu-vpp% file a.out  ↵
a.out:          ELF 32 ビット MSB 実行可能 VPP UXP/V Version 1 Vector-EX Parallel
```

2.7.2 size, gsize

size(/usr/ccs/bin/size) コマンドは実行ファイルが必要とする領域をバイト単位で表示するコマンドです.

```
kyu-vpp% size a.out  ↵                    <---size コマンド
1227176 + 146128 + 6446832 = 7820136      <--- 約 7.4MB の領域が必要
```

ただし size コマンドでは Fortran 90/VPP で翻訳された実行ファイルに対してはグローバル変数に対する見積もりができません.

kyu-vpp にはグローバル変数の領域の大きさを表示するコマンドとして gsize(/usr/local/bin/gsize) コマンドがあります.

```
kyu-vpp% gsize b.out  ↵
--------------------------------------------------------------------
  global array information :
    total size        -            50680461KB (   4) on   32 pe's
      partitioned     -            50680461KB (   4)
      non-partitioned -                   0KB (   0)
    max. size / pe    -             1583764KB
--------------------------------------------------------------------
```

実行ファイルは約 48GB のグローバル変数を必要とし, 1PE あたりに必要な領域が約 1.5GB であることがわかります.

2.7.3 timex

timex(/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を得ることができます.

指定方法は通常のコマンドの前に timex と書くだけです.

```
kyu-vpp% timex frt test.f90 ↵    <--- 翻訳．CPU 時間等をモニター
     :
  (翻訳結果)
     :
real           43.03             <--- 経過時間
user           38.55             <--- 利用者の CPU 時間
sys             1.01             <--- システムの CPU 時間
vu-user         0.00
vu-sys          0.00
```

翻訳はすべてスカラーユニットで処理されます．

```
kyu-vpp% timex a.out ↵           <--- 実行．CPU 時間等をモニター
     :
  (実行結果)
     :
real            1.91
user            1.71
sys             0.10
vu-user         1.63             <--- 利用者のベクトルユニット使用時間
vu-sys          0.00             <--- システムのベクトルユニット使用時間
```

2.7.4 kill

何らかの原因で暴走したジョブは kill コマンド[22]でキャンセルします．まず，暴走している端末 (ウィンドウ) 以外から kyu-vpp に login します．次に ps(/bin/ps) コマンドのオプション -u に続けて自分の課題名を入力します．

```
kyu-vpp% ps -u a79999a ↵          <--- プロセス状態の表示
   PID TTY      TIME COMD
   370 pts/3    0:01 tcsh
   345 pts/5    0:00 ps
   257 pts/3    0:06 a.out
   339 pts/3    0:00 csh
   333 pts/5    0:00 csh
```

下線部分が暴走している実行ファイルだとします．このプロセスをキャンセルするため，kill -9 に続けてプロセス ID(PID) を指定します．

```
kyu-vpp% kill -9 257 ↵            <--- プロセスのキャンセル
```

[22] シェルのコマンドです．

2.8 kyu-vpp と kyu-cc

この章の最後に M-1800/20U の UXP/M(ホスト名 kyu-cc) の対話型処理と比べた kyu-vpp の使い勝手を書きます.

● kyu-vpp の長所

◇ native な Fortran 90 コンパイラである

"native" とは (ここでは) 「生粋の」という意味です. kyu-cc の Fortran 90 コンパイラ (`frtex`) は, Fortran 90 の言語を解釈していったん FORTRAN 77 EX に変換し再翻訳するというプリプロセッサ (preprocessor) です. そのため仕様に一部制限があったり翻訳メッセージが不親切だったりと, 完成されたコンパイラとはいえませんでした. kyu-vpp の Fortran 90 は規格を全て包含した純正のコンパイラであり, 文法レベルでの信頼性が高まりました. なお Fortran 90 の文法に関しては [37], [38], [51] を御覧下さい.

◇ベクトルチューニングされたプログラムに対しては十分な性能を発揮する

VPP700/56 のそれぞれの PE は, 理論ピーク性能 2.2GFLOPS の性能を持つベクトル計算機です. FLOPS 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.

十分なベクトルチューニングを行ったプログラムは, ベクトル演算機構のおかげで桁違いの性能を発揮します. 表 2.5 は VU 率 (全 CPU 時間に占めるベクトルユニットが使用した割合) が 90% を超える Fortran プログラムで実測した CPU 時間です. オプションはメーカーの推奨値を採用しています.

表 2.5: VU 率が高いプログラムの性能

計算機	S-4/1000E	M-1800/20U	VPP700/56
行列積 512 × 512 (SSL II/VP の `DVMGGM` 使用)	52 秒 (5MFLOPS)	0.2 秒 (1280MFLOPS)	0.12 秒 (2128MFLOPS)
連立 1 次方程式 1024 元 (SSL II/VP の `DVLAX` 使用)	42 秒 (16MFLOPS)	0.6 秒 (1064MFLOPS)	0.43 秒 (1755MFLOPS)
Stokes 方程式の有限要素近似解	127 秒	2.2 秒	1.7 秒
Volterra 型積分方程式の離散近似	2244 秒	25 秒	12 秒

S-4/1000E はライブラリサーバー (ホスト名 wisdom) です. M-1800/20U の理論ピーク性能は 1.2GFLOPS です. ベクトルチューニングされたプログラムについては理論値に近い性能差が出ています. さらに VPP700/56 の結果は 1PE での値だということを強調しておきます. これらの値は Fortran 90/VPP やメッセージパッシングライブラリによる並列化によってさらに性能が伸びる可能性があります[23].

◇ライブラリを対話型に作成できる

kyu-cc で翻訳したオブジェクトファイル, 実行ファイルは kyu-vpp と非互換です. 従来 (1997 年 4 月以前)VPP700/56 の実行ファイルやライブラリを作成する場合, バッチリクエストを kyu-cc から `qsub` コマンドで投入する必要がありました. しかし kyu-vpp での対話型処理により, オブジェクトファイル, 実行ファイルの作成が簡単になりました.

◇同一コンパイラでデバッグできる

これまでは kyu-cc でプログラム開発, 動作確認, デバッグを行ったあと, VPP700/56 にバッチジョブを依頼していました. 通常はこの方法で問題ないのですが, 浮動小数点の違いに敏感なプログラムや VPP700/56 にしかない機能[24]を使いたい場合は, バッチ処理を余儀なくされていました.

kyu-vpp の対話型処理はバッチ処理と同じコンパイラが起動するため, これらの心配は要らなくなります. もちろんプログラムのデバッグも容易になりました.

[23] もちろんベクトル長や並列化に向いているかなど, 問題はたくさんあります.
[24] 例えば, SSL II/VP の拡張機能 II や GETTOD サービスルーチンなど.

● kyu-vpp の短所

◇スカラー性能

先ほど，ベクトルチューニングされたプログラムに対する速度の比較を表 2.5 にあげました．これらはベクトル計算機の性能をフルに発揮することが期待される理想的なプログラムです．しかし，ベクトル化率があまり高くないプログラムの場合， kyu-cc と kyu-vpp の性能が逆転することがあります．

これは VPP700/56 のスカラー演算性能がそれほど高くないことに原因します．(VPP700/56 にとっては) 最悪のケースとして，先ほどのプログラムを全てスカラー演算で計算させてみます[25]．結果は表 2.6 の通りです．

表 2.6: ベクトル化率の極端に低いプログラムの性能

計算機	M-1800/20U	VPP700/56
Stokes 方程式の有限要素近似解	48 秒	69 秒
Volterra 型積分方程式の離散近似	522 秒	1291 秒

上の表の性能は S-4/1000E と比べてとても魅力的とは言えない数字です[26]．自動ベクトル化技術はもうほとんど完成されたと言われています．しかし逆に言えば，「完成」されたコンパイラが諦めたのですから，そこから先のチューニングは利用者で行なうということでもあります．アムダールの法則 (cf.10章) によれば，ベクトル化率が 90% を超えない限り実行性能はなかなか向上しません．

全くベクトル計算向きのアルゴリズムを意識せずに書いたプログラムをそのまま自動並列コンパイラにかけても， 90% を超えるベクトル化率を達成することはなかなか難しいと思われます．

つまり，ベクトル計算機でのプログラミングでは次のことが大事です．

> ベクトル計算機の性能を引き出すには，利用者がチューニングを行う覚悟が必要．

ベクトル化プログラミングについては 10章, [4], [1], [7] を参考にしてください．

経験的にも，サブルーチンを差し替えたりメモリーアクセスを意識してループを入れ換えただけで数倍の性能向上が得られることがよくあります[27]．

また，スカラー演算のテストでも確認できるように， kyu-vpp は kyu-cc に比べてさらにベクトルチューニングが必要です． Fortran 90/VPP による並列化もベクトル並列計算機という性格から一つ一つの PE のベクトル性能が高くないと全体の性能が発揮できません (cf. [2], [5], [19])．従って，勇んで並列化に挑む前に，事前に 1PE のベクトル化チューニングをきちんと行う必要があります．

◇翻訳時間が増大する

kyu-vpp の Fortran コンパイラは，従来のベクトル化や最適化にプラスしてスカラープロセッサ向けの最適化を行っています．翻訳はスカラーユニットが受け持ちますので，スカラー演算性能の差も翻訳時間に大きく影響します．

そのため， kyu-vpp でのプログラムの翻訳時間は kyu-cc と比較して一般に長くなります．大規模なプログラムでデバッグが済んだ部分はできるだけ手続き化しライブラリとして管理することをお勧めします[28]．

[25] VPP700/56 の SSL II/VP はベクトル化されたライブラリのため，スカラーでは実行できませんでした．

[26] 特に富士通の汎用計算機，ベクトル計算機のスカラー性能は他社と比べても悪いようです．それにしても，ベクトル演算機能を抑止した途端，(極端な話) 個人が趣味で購入するパーソナルコンピュータ並みの性能になってしまうのは困ったものです．

[27] もちろん，本質的にベクトル計算機向きではない問題も存在します．

[28] 翻訳時間はプログラムの長さにほぼ比例します．従ってメインプログラムを可能な限りスリムにすることが肝心です．

3　バッチリクエストの投入

この章では，2章の対話型処理で実行できない大規模なジョブおよび並列ジョブを VPP700/56 で実行する手順である NQS について説明します．

3.1　NQS

UXP のバッチ処理のシステムを総称して NQS(Network Queuing System) と呼びます．NQS はもともと NASA Ames Research Center のために Sterling Software が開発したシステムです．現在では UNIX の標準的なバッチ処理システムとして広く流通しています．

VPP700/56 での計算は「バッチリクエスト」と呼ばれるシェルスクリプト[29]を記述・投入することにより行います．

3.1.1　汎用機との差異

M-1800/20U の UXP システム (kyu-cc) のコンパイラと VPP700/56 のコンパイラとはオプション，言語仕様が異なります．最新のオプションはそれぞれの `man` コマンド (VPP700/56 のオンラインマニュアルの検索方法は 1.4.1節参照) で比較できます．

3.1.2　バッチキュー

VPP700/56 には使用できる計算資源に応じてバッチのキュー[30]が設定されています．1998 年 4 月現在の制限値は表 3.1 の通りです．

表 3.1: バッチキューの制限値

キュー名	CPU 時間	記憶域		処理形態
		最大値	省略値	
c	60 分	0.5GB		翻訳専用
s	60 分	1.7GB	0.5GB	1PE
p1	1200 分	1.7GB	0.5GB	1PE
s8	10 分	1.7GB/PE	0.5GB/PE	最大 8PE
p8	1200 分	1.7GB/PE	0.5GB/PE	最大 8PE
p16	1200 分	1.7GB/PE	0.5GB/PE	最大 16PE
p32	1200 分	1.7GB/PE	0.5GB/PE	最大 32PE

翻訳専用の `c` キューは翻訳しかできない代わりに優先度が高めに設定されています．翻訳レベルのデバッグやベクトル化，並列化の翻訳情報を得るために利用します．並列プログラムを翻訳し実行ファイルを作ることも可能です．

[29] shell script: 単にスクリプトとも呼びます．発する指令を書き下したプログラムのことです．
[30] queue. 直訳すれば「列」「待ち行列」．

s, p1 キューは非並列 (1PE) のジョブの翻訳・実行が可能です．並列でない通常の Fortran プログラムや C, C/VP, C++ プログラムを投入します．使用できる記憶領域は最大 1.7GB です[31]．

s キューは並列処理のデバッグ用のキューです．

p8, p16, p32 キューは大きめの並列処理を行うためのキューです． Fortran 90/VPP プログラム，メッセージパッシングライブラリを埋め込んだ Fortran, C, C++ プログラムの翻訳・実行を行います．使用できる記憶領域は省略値で PE 数 ×0.5GB，最大で PE 数 ×1.7GB です．従って 32PE では最大 54GB 以上の記憶領域が確保できます．利用者が一度に作成できるファイルの上限は 2GB です．

3.1.3 ジョブ投入にあたっての注意事項

- 1997 年 8 月 1 日から 1PE あたりのリージョンサイズの省略値を 0.5GB に設定しました．上限はこれまで通り 1.7GB です．省略値を設定したことにより，qsub コマンド (cf.3.5節) でバッチリクエストを投入する際に -lM オプションを指定しない場合，リージョンサイズは 0.5GB に設定されます． 0.5GB 以上のリージョンサイズを確保したい場合は，-lM オプションによるサイズの設定が必要です．

- 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行ないません．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[32]．

- 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，実行ファイルが作成できなかった場合，以前の実行ファイルが呼ばれ，結果として無駄な計算が行われる危険があるためです．

  参考として，翻訳前に実行ファイル a.out が既に存在する場合無条件に消去するスクリプト (csh) の記述例をあげます．

```
#
cd EXAMPLE
if ( -f a.out) then
 rm -f a.out
endif
frt -Wx -Ps -Wv,-m3 test.f90
a.out
```

- バッチリクエストの記述例にあるディレクトリ名 “EXAMPLE” はあくまでも一つの例です．利用者は自由な名前でディレクトリを作成することができます．

```
kyu-vpp% cd  ↵              <--- ホームディレクトリに移動
kyu-vpp% mkdir FEM  ↵       <--- ディレクトリ FEM を新規に作成
```

- 並列ジョブを実行する場合は必ず翻訳時オプション -Wx を指定して下さい．

- “p” で始まるキューの CPU 時間の上限は 20 時間です． CPU 時間の打ち切り時間を 20 時間以下に設定したい場合は qsub コマンドの -lT オプションで指定します[33]．

[31] 実際には 1.7GB よりもう少し確保できます．
[32] 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．
[33] -lT オプションで制限値を越える CPU 時間の設定はできません．

- 1998 年 5 月現在，VP2600/10 の MSP システムでサポートしていたジョブのリスタート機能はサポートされていません．従って実行中のジョブが定期保守時間などにより実行を中断された場合，最初の処理から (自動的に) 再実行となりますので，運用時間には十分に注意してください．
- 制限値を越えるジョブを投入したい場合は個別対応となります．`request@cc.kyushu-u.ac.jp` まで御相談ください．

3.1.4 NQS の流れ

NQS の大きな流れは図 3.1 のようになります．

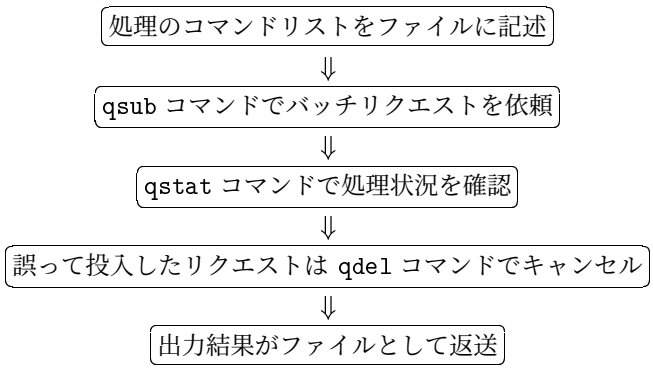


図 3.1: NQS の流れ

以下，具体的な例をあげながら説明します．

3.2 バッチリクエストの記述 (Fortran)

バッチリクエストはエディタで**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．また，ファイル名は 7 文字以下にしてください．

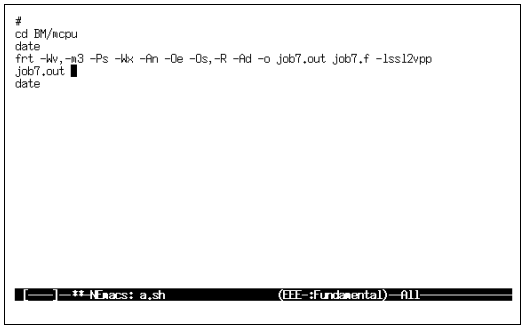


図 3.2: emacs でのバッチリクエスト作成風景

ここではバッチリクエストファイルはすべて `a.sh` とします．

3.2.1 標準的な形 I(Fortran 90/VP)

```
#                                  <---csh で記述
cd EXAMPLE                         <--- ディレクトリの移動
frt -Ps -Wv,-m3 test.f90           <--- 翻訳
a.out                              <--- 実行
```

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 (ファイル入出力など) を書き込む場合は文法が異なるので注意が必要です[34].

次の行の “cd” は，ディレクトリを kyu-vpp の ~/EXAMPLE に移動するコマンドです (kyu-cc から見れば ~/VPP/EXAMPLE に対応します)．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 kyu-vpp 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ください．

次の行は frt コマンドで test.f90 を翻訳する処理です．オプションは 2章の対話型処理と同じものが指定できます．例では翻訳オプションとして，ベクトル化メッセージ (-Wv,-m3) とプログラムリストの出力 (-Ps) 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です．

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 標準的な形 II(Fortran 90/VP)

```
#
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．この場合 frt コマンドは必要ありません．

3.2.3 標準的な形 (Fortran 90/VPP)

```
#                                  <---csh で記述
cd EXAMPLE                         <--- ディレクトリの移動
frt -Wx -Ps -Wv,-m3 test.f90       <--- 翻訳
a.out                              <--- 実行
```

-Wx オプションの指定により Fortran 90/VPP が起動され，並列プログラムの翻訳を行います．実行ファイルが既に存在する場合は frt コマンドは必要ありません．

3.2.4 最適化レベルを下げて翻訳・実行

```
#
cd EXAMPLE
frt -Ob -Ps -Wv,-m3 test.f90       <--- 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 (cf.9.2.7節) 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．

[34] shell(シェル) は，端末またはファイルから読み取ったコマンドを実行するコマンド・プログラミング言語です．sh は，B shell(Bourne shell)，csh は C shell を起動します．詳しい機能については市販の UNIX の本または man sh，man csh を参照してください．

3.2.5 ベクトル化レベルを下げて翻訳・実行

```
#
cd EXAMPLE
frt -Ps -Wv,-an,-m3 test.f90      <--- 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．オプション -Wv,-an の指定により，式の評価順序の変更を抑止した実行結果と比較することで計算誤差の影響をチェックできます．最適化レベルは自動的に -Ob になります．また，実行時間は増大します．

3.2.6 最大限の最適化を行う

```
#
cd EXAMPLE
frt -Of -KVPP700 -Ps -Eipue -Wv,-Of,-Ov,-m3 test.f90  <--- 翻訳．最大限の最適化
a.out                                                 <--- 実行
```

VPP700/56 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 (-E) もあわせて指定します．通常は省略値で十分ですが．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．翻訳時間は増大します．

3.2.7 プログラムリストの出力を抑止

```
#
cd EXAMPLE
frt test.f90           <--- 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プログラムをバッチ処理で翻訳する時は -Ps，-Wv,-m3 オプションを削除しても構いません．

3.2.8 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.f90  <--- 翻訳
```

test.f90 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．-o は，続いて記述する名前で実行ファイルを作成するオプションです．

ソースプログラムが長大な場合，対話的な処理ではシステムの制限値によって翻訳ができない場合があります．その時はバッチリクエストを翻訳専用キューである c キューに投入するといいでしょう．

その他，オブジェクトファイルの作成，自分用のライブラリの作成・結合，複数のファイルの分割処理，モジュールの引用などの処理は対話型処理と同様のオプションを指定してください．

3.2.9 標準入出力の例

リダイレクション (redirection) も対話型と同様に利用できます． Fortran の入出力に関する装置参照番号 (論理機番) の 5 番が標準入力 (stdin)， 6 番が標準出力 (stdout) に対応しています．

```
#
cd EXAMPLE
frt -Ps -Wv,-m3 test.f90
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．

3.2.10 環境変数によるファイル処理

open 文中で “file=” に続けて，ファイル名を ’ ’で括り指定する方法は対話型と同様です．
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バッチリクエストは次のように記述します．

```
#                                 <---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.f90          <--- 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，上記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．
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ください．

```
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.f90          <--- 翻訳
a.out                             <--- 実行
```

3.2.11 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.f90          <--- 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．

3.2.12 IEEE 形式でファイル処理

M 形式 (IBM 形式) と IEEE 形式の変換は実行時オプションで指定します.

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

3.2.13 サブルーチンライブラリの結合

対話型処理と同様です. 以下は SSL II/VPPの結合例です. SSL II/VPP のサブルーチンを使用している場合は `-lssl2vpp` を指定します.

```
#
cd EXAMPLE
frt -Wx -Ps -Wv,-m3 test.f90 -lssl2vpp  <--- 翻訳. SSL II/VPP を結合
a.out                                   <--- 実行
```

3.2.14 CPU 時間の計測

`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.

```
#
cd EXAMPLE
timex frt -Ps -Wv,-m3 test.f90    <--- 翻訳. CPU 時間をモニター
timex a.out                       <--- 実行. CPU 時間をモニター
```

`timex` の情報は標準エラー出力 (cf.3.5.1節) に書き出されます.

3.2.15 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. この場合は異なる実行ファイルとして保存するため, `-o` オプションによって `a.out` 以外のファイル名に変えた方がいいでしょう.

```
#
cd EXAMPLE
frt -o b.out -Ps -Wv,-m3 test1.f90        <--- test1.f90 を翻訳
b.out                                     <--- 実行
frt -o c.out -Ps -Wv,-m3 test2.f90        <--- test2.f90 を翻訳
c.out                                     <--- 実行
```

3.3 バッチリクエストの記述 (C/VP, C, C++)

Fortran と同様, C/VP, C, C++ のバッチリクエストの記述方法もディレクトリの移動と `vcc`, `cc`, `CC` コマンドを指定します. また, 実行ファイルを実行するだけの場合, 翻訳コマンドは省略できます.

以下, 簡単な例をあげます.

3.3.1 標準例 (vcc)

```
#
cd EXAMPLE
vcc -Wv,-m3,-Ps test.c
a.out
```

ベクトル版 C の起動は vcc コマンドです．-Wv,-m3 はベクトル化に関するメッセージを出力します．

3.3.2 最適化レベルを上げる (vcc)

```
#
cd EXAMPLE
vcc -O -K4 -Wv,-m3,-Ps test.c
a.out
```

-O と -K4 の組合せで最大限の最適化となります．

3.3.3 SSL II/VP の結合

```
#
cd EXAMPLE
vcc -c -Wv,-m3,-Ps test.c
frt test.o -lssl2vp -lcvp -lm
a.out
```

3.3.4 C の翻訳・実行

```
#
cd EXAMPLE
cc -O test.c
a.out
```

スカラー版 C の起動は cc コマンドです．最適化レベルを上げた (-O) 翻訳例です．

3.3.5 C++ の翻訳・実行

```
#
cd EXAMPLE
CC -Fc test.C > test.c                  <---C のソースに変換
vcc -Wl,-L/usr/uxp/C++/lib test.c -lC   <---vcc コマンドによる翻訳
a.out
```

C++ の起動は CC コマンドです．ベクトル処理を行う場合は，一度 C のソースに変換したものを保存し，vcc コマンドによって処理することで，ベクトル処理可能な実行ファイルが作成できます．下線部分は必ず指定します．

3.4 メッセージパッシングライブラリの結合

メッセージパッシングライブラリの結合方法も対話型処理と同じです.

3.4.1 PVM の例 (Fortran)

```
#
cd EXAMPLE
frt test.f90 -Wl, -P -L. -J -dn -lpvm -lmp -lelf -lpx -lc -I.
setenv PVM_BUF_SIZE 204800
a.out
```

3.4.2 PVM の例 (C)

```
#
cd EXAMPLE
cc test.c -Wl, -P -L. -J -dn -lpvm -lmp -lelf -lpx -lc -I.
setenv PVM_BUF_SIZE 204800
a.out
```

なお, ライブラリの指定順序には細かい規定があります. 詳細は [13] を参照してください.

3.4.3 MPI の例 (Fortran)

```
#
cd EXAMPLE
frt test.f90 -Wl,-J,-P -L/usr/lang/mpi/lib -lmpi -lmp -lelf -lpx \
-I/usr/lang/mpi/include
a.out
```

“\” はコマンドの継続を意味します.

3.4.4 MPI の例 (C)

```
#
cd EXAMPLE
cc -K a4 test.c -Wl,-J,-P -L/usr/lang/mpi/lib -lmpi -lmp -lelf -lpx \
-I/usr/lang/mpi/include
a.out
```

詳細は [14] を参照してください.

3.4.5 PARMACS の例 (Fortran)

```
#
cd EXAMPLE
frt -o host  -Wl,-J,-P,-t host.o node.o other_objects.o other_libs.a \
-L.../parmacs/lib/fujitsu-vpp700/VPP700 -lpm6 -lmp2 -lpx -lm -lelf -lc
host
```

3.4.6 PARMACS の例 (C)

```
#
cd EXAMPLE
cc -o host -Wl,-J,-P,-t host.o node.o other_objects.o other_libs.a \
-L.../parmacs/lib/fujitsu-vpp700/VPP700 -lpm6 -lmp2 -lpx -lm -lelf -lc
host
```

詳細は [15] を参照してください.

3.5 バッチリクエストの投入

3.5.1 qsub コマンド

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．よく使うオプションは次の通りです．詳細は man qsub を参照してください．

-q␣*queue*	*queue* はキュー名です．省略すると， VPP700/56 の p1 キューに投入されます．
-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．
-mu␣*user*	バッチリクエストに関するメールを *user* に送ります．
-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「スクリプトファイル名.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「スクリプトファイル名.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．
-lM␣整数．小数 gb	リージョンサイズを指定します． “gb” はギガバイト単位を意味します． 1.7GB の場合は “-lM 1.7gb”， 0.8GB の場合は “-lM 0.8gb” となります．
-lT␣*time*	使用するバッチリクエストの CPU 時間の上限を設定します．設定は [[時間:]分:] 秒です． 【制限値の指定例】 -lT 10:35:20　10 時間 30 分 20 秒 -lT 12345　12345 秒 -lT 59:45　59 分 45 秒
-lPv *n*	並列処理で *n* 台の PE を使用することを指定します．

メールでの通知オプションとは別に，利用者の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．送付先は kyu-cc になります．

3.5.2 qsub コマンドオプションの記述

qsub コマンドのオプションは，バッチリクエストファイルに記述することができます．オプションは “# @$” に続けて空白を置かずに指定します．

```
#                                  <---csh で記述
# @$-lM 1.7gb                      <---qsub オプションの指定例 (1)
# @$-eo -me                        <---qsub オプションの指定例 (2)
# @$-lT 1:30:00                    <---qsub オプションの指定例 (3)
cd EXAMPLE                         <--- ディレクトリの移動
frt -Wx -Ps -Wv,-m3 test.f90       <--- 翻訳
a.out                              <--- 実行
```

3.5.3 バッチリクエストの投入例 1

以下バッチリクエストの記述されたファイル名を a.sh とします.

```
kyu-vpp% qsub -q s a.sh ↵                    <--- バッチリクエストの投入
Request 54661.kyu-vpp submitted to queue: s.
```

kyu-vpp から s キューに投入しました．下線部の “54661” がリクエスト番号，“kyu-vpp” がホスト名です．“54661.kyu-vpp” で**リクエスト名**を構成します.

kyu-cc からの投入方法も全く同じです．ただしリクエスト名のホスト名が “kyu-vpp” ではなく，“kyu-cc” となります.

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.sh.e54661 と標準出力ファイル a.sh.o54661 が返ってきます.

```
kyu-vpp% ls ↵
test.f90    a.out       a.sh        a.sh.e54661  a.sh.o54661
```

バッチリクエストを記述したファイルの名前が 8 文字以上だと，標準エラー出力ファイル，標準出力ファイルの最初のファイル名が 7 文字で切られて表示されます.

3.5.4 バッチリクエストの投入例 2

```
kyu-vpp% qsub -lM 1.7gb a.sh ↵               <---p1 キューに 1.7GB で投入
Request 30482.kyu-vpp submitted to queue: p1.
```

リージョンサイズを 1.7GB に設定したバッチリクエストの投入例です.

3.5.5 バッチリクエストの投入例 3

```
kyu-vpp% qsub -q p16 -lM 1.7gb a.sh ↵        <---p16 キューに 1PE あたり 1.7GB で投入
Request 30483.kyu-vpp submitted to queue: p16.
```

リージョンサイズを 1.7GB に設定し p16 キューに投入する例です．並列ジョブの場合も -lM オプションでリージョンサイズを設定します．指定するサイズは合計のサイズではなく 1PE あたりの値を設定します.

●制限値を超えたジョブの運命

省略値の 0.5GB および qsub コマンドの -lM オプションで指定したリージョンサイズを超えてしまったジョブは，システムにより処理を打ち切られます． Fortran プログラムであらかじめ (静的に) 領域を宣言している場合は，実行に入る前に以下の例のようなメッセージを標準エラー出力ファイルに書き出します.

```
Warning: no access to tty; thus no job control in this shell...
Mon Sep  8 11:35:07 JST 1997
UX:timex: ERROR: Unable to exec a.out: Not enough space
```

動的に領域を確保するプログラムの場合は，実行時にメモリーが足りなくなった旨のメッセージを出力し，実行が打ち切られます．以下は Fortran 90 で動的に配列を宣言するプログラムでの出力例です.

```
Warning: no access to tty; thus no job control in this shell...
 jwe0912i-u line 41 The dynamic area cannot be reserved because of insufficient
area. The used size is 00032768KB. The required size is 00032769KB.
```

配列の大きさをすべて宣言してあるプログラムから作成された実行ファイルは size(/usr/ccs/bin/size) コマンドでおおよそのリージョンサイズを見積もることができます．また， Fortran 90/VPP で翻訳された実行ファイルに対しグローバル変数の領域の大きさを表示するコマンドとして gsize(/usr/local/bin/gsize) コマンドがあります．それぞれ 2.7.2節を参照してください．

3.5.6 バッチリクエストの投入例 4

```
kyu-vpp% qsub -q p1 -o out.data -eo a.sh ↵   <--- バッチリクエストの投入
Request 15467.kyu-vpp submitted to queue: p1.
```

p1 キューへ投入しました．-q p1 は省略可能です．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 に格納されます．

```
kyu-vpp% ls ↵
test.f90    a.out       a.sh        out.data
```

3.5.7 バッチリクエストの投入例 5

```
kyu-vpp% qsub -q p16 -me -mi a.sh ↵              <--- バッチリクエストの投入
Request 15467.kyu-vpp submitted to queue: p16.
```

p16 キューへ投入しました．オプションとして，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．

●実行終了のメール

qsub コマンドに -me オプションを付加した場合，実行終了のメールが送られてきます．

```
Return-Path: <root@kyu-vpp.cc.kyushu-u.ac.jp>
Date: Tue, 7 Jan 1997 23:06:03 +0900
From: 0000-Admin(0000) <root@kyu-vpp.cc.kyushu-u.ac.jp>
Subject: NQS request:  19373.kyu-vpp ended.
Apparently-To: a79999a@kyu-vpp.cc.kyushu-u.ac.jp

Request name:   a.sh
Request owner:  a79999a
Mail sent at:   Tue Jan  7 23:06:03 JST 1997

Request exited normally.

_Exit() value was: 0.
```

“Request exited normally.” の連絡があれば (とりあえず) ひと安心です．

●統計情報のメール

qsub コマンドに -mi オプションを付加した場合，統計情報のメールが送られてきます．資源の使用状況の詳しい情報が得られます．

```
Return-Path: <root@kyu-vpp.cc.kyushu-u.ac.jp>
Date: Tue, 7 Jan 1997 23:06:04 +0900
From: 0000-Admin(0000) <root@kyu-vpp.cc.kyushu-u.ac.jp>
Subject: NQS request:  19373.kyu-vpp information.
Apparently-To: a79999a@kyu-vpp.cc.kyushu-u.ac.jp

Request name:   a.sh
Request owner:  a79999a
Mail sent at:   Tue Jan  7 23:06:04 JST 1997

Queue name:  p16
Exit code :  0
Start/End :  Tue Jan  7 22:36:07 JST 1997 / Tue Jan  7 23:05:57 JST 1997
Elaps     :  00:29:50

System CPU time       :    637549 msec   User CPU time         :  26643632 msec

System vector CPU time:         0 msec   User vector CPU time  :  25253540 msec

Max vector memory     :     21760 Mbyte
Processing Element    :        16 PE

PE-ID              CPU                          Vector        Max vector memory
       System time      User time    System time      User time
-------------------------------------------------------------------------------
  28    41806 msec   1668409 msec         0 msec   1578335 msec     1360 Mbyte
  29    39334 msec   1665269 msec         0 msec   1578339 msec     1360 Mbyte
  2a    39163 msec   1665096 msec         0 msec   1578346 msec     1360 Mbyte
  2b    39599 msec   1664443 msec         0 msec   1578340 msec     1360 Mbyte
                  :
```

●課金システムからのメール

課金システムからこのジョブに要した負担金情報が kyu-cc に送られてきます．

```
Return-Path: <root@utms>
Date: Tue, 7 Jan 1997 23:06:46 +0900
From: root@utms (0000-Admin(0000))
To: a79999a@kyu-vpp.cc.kyushu-u.ac.jp
Subject: 145
Content-Type: text

User_ID = 193 Group_ID = 101
Job_ID = 145 Qsub_host = kyu-vpp.cc.kyushu-u.ac.jp
Job_Account =  179 (yen) Total-Account =  19944 (yen)
Your Budget = 250000 (yen)
```

3.6 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) コマンドによって調べることができます．

3.6.1 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．

なお， 1997 年 1 月 7 日から，マシン名の省略値が VPP700/56 のマシン名 “kyu-vpp” に変更されましたので， VPP700/56 の処理状況は qstat のみで調べることができます．また， kyu-cc に投入したスカラージョブの処理状況が知りたい場合は qstat @kyu-cc を指定してください．

```
kyu-vpp% qstat | less ↵          <--- VPP700/56 の処理状況を表示
c@kyu-vpp;  type=BATCH;  [ENABLED, INACTIVE];  pri=31
  0 exit;   2 run;   1 queued;   0 wait;   0 hold;   0 arrive;


s@kyu-vpp;  type=BATCH;  [ENABLED, INACTIVE];  pri=31
  0 exit;   1 run;   0 queued;   0 wait;   0 hold;   0 arrive;


p1@kyu-vpp;  type=BATCH;  [ENABLED, INACTIVE];  pri=31
  0 exit;   5 run;   1 queued;   0 wait;   0 hold;   0 arrive;


p8@kyu-vpp;  type=BATCH;  [ENABLED, RUNNING];  pri=31
  0 exit;   1 run;   0 queued;   0 wait;   0 hold;   1 arrive;

        REQUEST NAME       REQUEST ID          USER PRI    STATE  JOB-ID  PHASE
<2 request RUNNING>
<1 request QUEUED>

p16@kyu-vpp;  type=BATCH;  [ENABLED, RUNNING];  pri=31
  0 exit;   1 run;   0 queued;   0 wait;   0 hold;   0 arrive;

        REQUEST NAME       REQUEST ID          USER PRI    STATE  JOB-ID  PHASE
    1:          a.sh   15467.kyu-vpp        a79999a  31  RUNNING     145  RUN

p32@kyu-vpp;  type=BATCH;  [ENABLED, INACTIVE];  pri=31
  0 exit;   0 run;   2 queued;   0 wait;   0 hold;   0 arrive;
                              :

s8@kyu-vpp;  type=BATCH;  [ENABLED, INACTIVE];  pri=31
  0 exit;   0 run;   0 queued;   0 wait;   0 hold;   0 arrive;
                              :
```

リクエスト “15467.kyu-vpp” が p16 キューで実行中であることがわかります．

3.6.2 qps コマンド

バッチリクエストの実行状況を表示するコマンドとして qps(/usr/local/bin/qps) コマンドがあります. qps コマンドの形式は

qps␣*options*

です. *options* は次が指定できます.

-a	実行中のすべてのジョブを表示する.
-q	実行待ちジョブの一覧を表示する.
-p	並列ジョブの詳細を表示する.

なお, 自分以外の利用者のジョブは user 名が “*******” で表示されます.

●使用例 I

kyu-vpp からqps を入力し, ジョブの状態を表示します.

```
kyu-vpp% qps ↵
que  user    request            cpu      vu  vu/cpu cpu-limit  elapse  v-mem(MB)
 p1 a79999a 35193.kyu-vpp  18:02:31  17:09:26  95%  20:00:00  18:10:29 1240/1600
```

●使用例 II

kyu-vpp からqps -a を入力し, 実行中のジョブの一覧を表示します.

```
kyu-vpp% qps -a ↵                 <--- 実行中ジョブの一覧を表示
que  user    request            cpu      vu  vu/cpu cpu-limit  elapse  v-mem(MB)
 p1 a79999a 35193.kyu-vpp  18:02:31  17:09:26  95%  20:00:00  18:10:29 1240/1600
 p1 ******* 26462.kyu-cc   10:01:41   9:07:15  90%  20:00:00  17:25:08   48/ 512
 p1 ******* 35228.kyu-vpp  16:37:48  16:19:35  98%  20:00:00  16:45:40 1536/1704
 p1 ******* 35236.kyu-vpp   5:28:14   0:02:06   0%  20:00:00   6:10:03  104/ 512
 p8 a79999a 35243.kyu-vpp   waiting for allocation  20:00:00  --------    */1000
p32 ******* 35031.kyu-vpp   1:40:35   1:29:22  88%  20:00:00   2:03:22  936/1704
 mx ******* 35246.kyu-vpp   0:49:27   0:39:04  79%  19:59:46   1:39:33  160/ 512
 mx ******* 35249.kyu-vpp   0:29:31   0:14:44  49%  19:57:33   0:59:24  480/ 752
 d1 ******* 35231.kyu-vpp  15:17:28   5:06:56  33% 160:00:00  15:32:16  160/1792
```

●使用例 III

kyu-vpp からqps -q を入力し, 実行待ちジョブの一覧を表示します.

```
kyu-vpp% qps -q ↵                 <--- 実行待ちジョブの一覧を表示
No  queue user    request                shell
 1     p8 a79999a 24964.kyu-cc             a.sh
 2     p8 ******* 31220.kyu-vpp    ns03_1_1.nqs
 1    p16 ******* 24495.kyu-cc             c.vpp
 2    p16 ******* 30556.kyu-vpp          sc.nqs
 3    p16 ******* 30611.kyu-vpp          move16
 1    p32 ******* 24508.kyu-cc         cuag3_a1
 2    p32 a79999a 24522.kyu-cc             b.sh
```

“No” はそのキュー内での順番を示します.

3.7 バッチリクエストのキャンセル

3.7.1 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(/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 (3.6.1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-cc	kyu-cc へ投入したバッチリクエストをキャンセルする場合に指定します．

3.7.2 リクエストのキャンセル例

```
kyu-vpp% qdel 15478.kyu-vpp ↵     <--- 実行待ちリクエストをキャンセル
Request 5478.kyu-vpp 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-vpp% qdel -k 15479.kyu-vpp ↵     <--- 実行中のリクエストをキャンセル
Request 5479.kyu-vpp is running, and has been signalled.
```

3.8 印刷

ソースリストや処理結果のテキストデータはセンター 2 階のプリンターに出力することができます．もちろん研究室のワークステーションやパーソナルコンピュータに ftp で転送しても構いません．

3.8.1 ネットワークプリンターへの出力

kyu-vpp のコマンドは lp(/bin/lp)です．

```
kyu-vpp% lp a.sh.o29349 ↵          <--- ネットワークプリンターへの出力
```

kyu-cc からVPP ディレクトリにあるファイルをネットワークプリンターへ出力する場合は -c オプションを付加します．それ以外の kyu-cc のファイルには必要ありません．

コマンドは lp(/usr/bin/lp) です．

```
kyu-cc% lp -c a.sh.o29349 ↵          <--- ネットワークプリンターへの出力
```

3.8.2 NLP への出力

kyu-cc からのみ NLP に出力できます．コマンドは utoprint(/usr/local/bin/utoprint) です．

```
kyu-cc% utoprint a.sh.o29349 ↵          <---NLP への出力
```

後は， NLP 横のコンソールから MSP の 課題とパスワードを入力し最後が “0” のジョブに対して出力要求を出します．

3.9 kyu-cc からの利用

VPP700/56 は汎用計算機 M-1800/20U の UNIX OS である UXP/M システム (ホスト名: kyu-cc, IP アドレス: 133.5.9.1) からも利用できます．バッチリクエストの記述方法は kyu-vpp と同様です．ここでは利用の際の注意点を述べます．

3.9.1 処理の流れ

1. 汎用計算機 M-1800/20U の UXP システム (kyu-cc) に login します． kyu-cc の OS(UXP/M) も kyu-vpp と同様 UNIX の SYSTEM V 互換のオペレーティングシステムです．

図 3.3: M-1800/20 の外観

2. kyu-cc の利用者のホームディレクトリ (例えば a79999a さんの場合 /home/usr9/a79999a) に “VPP” というディレクトリがあります． 2.2.4節で説明したように，このディレクトリ VPP は VPP700/56 の利用者ホームディレクトリとシンボリックリンクが張られています．

   VPP700/56 にジョブを投入する際，使用するソースファイルやデータは VPP700/56 から見える場所にある必要があります．従って kyu-cc から VPP700/56 へジョブを投入する場合，使用するソースプログラムおよびデータは必ず VPP 下 (例えば /home/usr9/a79999a/VPP) に作成してください．

3. kyu-cc のエディタ (vi, mule)[35]によりプログラムを作成，または ftp で他のホストからプログラムを転送します．

4. kyu-cc の Fortran, C, C++ コンパイラによるデバッグを行います． kyu-cc のプログラム言語の環境は表 3.2 の通りです．各コマンドは man コマンドで機能が参照できます．

[35] UXP 版の PFD はサポートされていません．

表 3.2: M-1800/20 の UXP で利用可能なプログラム言語

ソフトウェア名	機能	コマンド
FORTRAN77 EX/VP	ベクトルコンパイラ	`frt`
Fortran 90/VP	ベクトルコンパイラ	`frtex`
C	C コンパイラ	`cc`
C/VP	ベクトル C コンパイラ	`vcc`
C++	C++ コンパイラ	`CC`

各プログラム言語は，言語仕様，オプションが VPP700/56 と完全に同じではありませんので注意してください[36].

また，実行ファイルに互換性はありません．従って kyu-cc で作成した実行ファイルを VPP700/56 で実行することはできませんし，逆も不可です．

5. バッチリクエストファイルを kyu-cc のエディタ (vi, mule) により作成します．
6. VPP700/56 にバッチ処理を依頼します．処理結果は kyu-cc に返却されます[37].

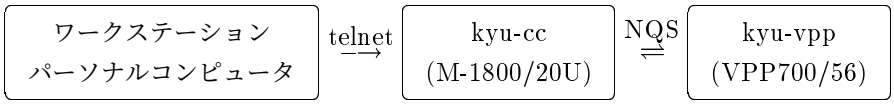


図 3.4: kyu-cc から利用する場合の処理の流れ

[36] kyu-cc の Fortran 90(`frtex`) は， Fortran 90 の言語仕様を FORTRAN77 EX に変換するプリプロセッサです．特に Fortran 90 のモジュール機能を使う場合 `frtex` はメッセージが極めて不親切ですので， kyu-vpp かライブラリ・サーバー wisdom(IP アドレス : 133.5.9.9) の Fortran 90 コンパイラ (コマンド `frt`) でクロスチェックすることをお勧めします．

[37] kyu-cc とリンクされた VPP700/56 の利用者のディレクトリ配下に返却され，それをあたかも kyu-cc のファイルのように取り扱う仕組みとなっています．

4 MSP からのジョブの投入

この章では，汎用計算機 M-1800/20U の OS である MSP システムから VPP700/56 を利用する方法について説明します．なお， MSP のジョブ制御文に関する知識は仮定します．

4.1 M-VPP 連携機能

MSP から VPP700/56 の利用は「M-VPP 連携機能」と呼ばれるソフトウェアを介して行います[38]．

M-VPP 連携機能は，汎用計算機 M-1800/20U の MSP システム (ホスト名 kyu-msp) から VPP700/56 の UXP/V システム (ホスト名 kyu-vpp) へバッチジョブを投入し，処理結果を MSP へ返すという MSP 利用者のためのシステムです．浮動小数点形式の変換および並列ジョブの翻訳は自動的にオプションが設定されますので，利用者が指定する必要はありません．

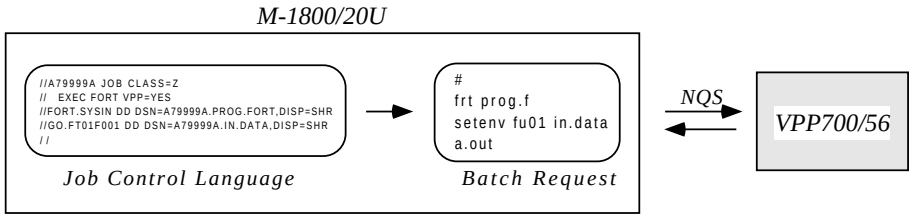


図 4.1: M-VPP 連携機能

従来のスーパーコンピュータ VP2600/10 は UXP と MSP の 2 つの OS が動作していましたが， VPP700/56 では UNIX OS である UXP に一本化されました．このため， MSP で VP2600/10 を利用されていた方は，UNIX OS である UXP に移行するか，または M-VPP 連携機能を利用する必要があります[39]．

4.2 制限事項

汎用機の MSP から VPP700/56 を利用する M-VPP 連携機能には，以下の制限があります．

- 利用可能な言語処理プログラムは Fortran のみです． C, C/VP, C++ は使用できません[40]．また，メッセージパッシングライブラリも利用できません．
- Fortran のプログラミングは Fortran 90 で記述可能ですが，コマンドオプション `-X9` または翻訳指示行 `LANGLVL(90)` の指定が必要です．また，汎用機の MSP 上のコンパイラは FORTRAN 77 仕様の FORTRAN77 EX です．従って， FORTRAN77 EX の仕様を超えたプログラムを汎用機で対話的に作成することはできません[41]．
- 翻訳／結合編集／実行を行う `STEP=CLG`，および翻訳を行う `STEP=C` の形態でのみ利用できます．ロードモジュールを作成して保存することはできません．

[38] M-VPP/CF(MSP and VPP/Control Facility) という名前のソフトです．
[39] アプリケーション，他の計算機との親和性を考えると， UNIX OS である UXP を利用することを今後強くお勧めします．
[40] 無理すれば使えないこともないのですが， UXP でプログラムを作成する方がはるかに便利です．
[41] つまり，モジュールやポインタ機能を使う場合，バッチ処理で動作を確認する必要があります．

- 使用可能な数値計算ライブラリは SSL II/VP, SSL II/VPP と NUMPAC です.
- PKS や Graphman などの図形ライブラリは利用できません. 汎用機での処理をお願いします[42].
- 私用ライブラリの利用, 実行時オプションの指定, Sampler の利用はできません. UXP を利用してください.

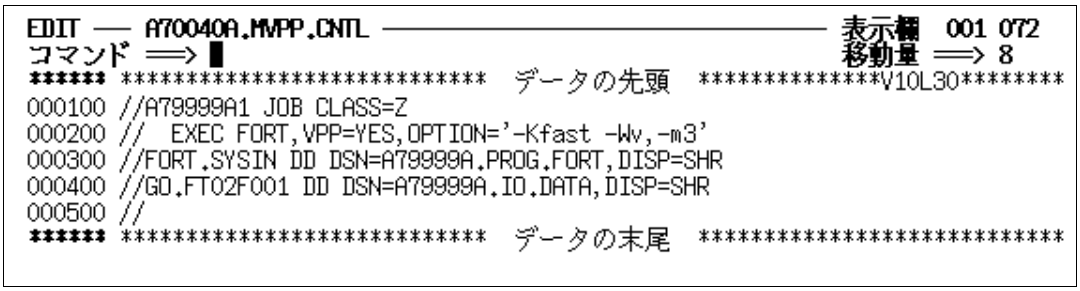


図 4.2: PFD での JCL 編集風景[43]

4.3 ジョブクラス

VPP700/56 のジョブクラスは W, X, Y, Z です. 制限値は表 4.1 の通りです.

表 4.1: ジョブクラスと制限値

ジョブクラス名	CPU 時間	最大記憶域	省略値	処理形態
W	60 分	1.7GB	0.5GB	1PE
X	1200 分	1.7GB	0.5GB	1PE
Y	1200 分	1.7GB/PE	0.5GB/PE	最大 8PE
Z	1200 分	1.7GB/PE	0.5GB/PE	最大 16PE

1997 年 8 月 1 日より記憶領域の省略値を 0.5GB に設定しました. これ以上のリージョンサイズを確保する場合は, カタログドプロシジャ FORT の `VREGION` パラメータを設定します.

16 台を超える PE を利用した並列ジョブを実行する場合は UXP を利用下さい. また, MSP のファイルの上限は 12,500 トラックです[44].

4.4 JCL の記述方法

4.4.1 カタログドプロシジャ FORT

MSP から VPP700/56 を利用するためには, 従来のカタログドプロシジャ`FORT` に `VPP=YES` オプションを付加します. 指定可能なパラメータは以下の通りです. それぞれカンマで区切って指定します.

[42]従来より図形ライブラリはすべて汎用計算機でのサポートです. また Graphman は UXP 版が機能的に優れています [50].
[43]ファイル名と課題名が矛盾していることは気にしないでください.
[44]`SPACE=(5000,500)` で宣言します. 1 トラック 47KB です.

VPP=YES	Fortran ジョブの実行 (翻訳／結合編集／実行) を VPP700/56 で行うことを指定.
STEP={CLG|C}	CLG は翻訳／編集結合／実行を指定する．C はデバッグ用に翻訳のみを行う場合に指定する．省略値は CLG.
OPT={B|E|F}	最適化のレベルを指定．省略値は E．“OPT=E” は “-Oe” に対応.
OPTION='*option-list*'	翻訳時オプションを UXP のコマンドライン形式で指定する.
VREGION=*size*	1PE あたりのリージョンサイズをメガバイト単位で指定.
VPPSRC=PO	FORT.SYSIN で指定した区分データセットのメンバすべてを翻訳する.
UCS=*UCS-name*	プリンタ出力時の UCS 名を指定.
FCB=*FCB-name*	プリンタ出力時の FCB 名を指定.

UCS(Universal Character Set), FCB(Form Control Buffer) は，センター 2 階の NLP に出力する文字のフォント，大きさ，1 ページの行数を変更するオプションです.

4.4.2 翻訳時オプションの指定方法

翻訳時オプションを指定する方法は，ソースプログラムの @FORTRAN 行を利用する方法と，OPTION パラメータに UXP の翻訳時オプションリストを記述する方法があります.

VPP=YES オプションを指定した場合，PARM.FORT= パラメータで翻訳時オプションを指定することはできません.

● UXP のオプションリストでの指定例

```
//A79999AW JOB CLASS=Z
//  EXEC FORT,VPP=YES,OPTION='-Ps -Wv,-m3'
//FORT.SYSIN DD DSN=A79999A.PROG.FORT,DISP=SHR
//
```

UXP のコマンドラインのオプションは，大文字／小文字を区別します．小文字を入力できない端末の場合は，直前に EBCDIC-ASCII の “¥” を記述します．また，Fortran 90/VPP を起動するオプション -Wx は，投入するジョブクラス (Y, Z) に応じて自動的に付加されます.

Fortran 90 の自由形式でプログラムを記述する場合は，-Free -X9 オプションを指定します.

●翻訳指示行での指定例

```
//A79999AV JOB CLASS=Z
//  EXEC FORT,VPP=YES
//FORT.SYSIN DD *
@FORTRAN NOADVANCED(NOEVL),DEBUG(SUBCHK)
//           DD DSN=A79999A.PROG.FORT,DISP=SHR
//
```

@FORTRAN 行は，翻訳されるすべての翻訳単位に対して有効となる翻訳指示行です．ただし上の例は FORT.SYSIN データセット (例では A79999A.PROG.FORT) が RECFM=F または RECFM=FB で，LRECL=80 である必要があります．それ以外の属性の場合は，ソースプログラムに記述してください．

なお，UXP のコマンドラインでのオプションと比べ，翻訳指示行で指定できるオプションはかなり制限されています．翻訳指示行オプション形式とコマンドオプション形式との対比表は付録 A.3を参照してください.

4.4.3 データセットの指定方法

ジョブの実行に必要なデータセット (プログラム，データ) は， DD 文で指定します．インクルードデータセット FORT.SYSINC，および VPPSRC=PO を指定した場合の FORT.SYSIN にのみ区分編成データセットの指定が許されます．その他のデータセットには順編成データセット，または，メンバ名指定の区分編成データセットを指定してください．

DD 文で指定したデータセットは，連携機能の標準的な解釈によりシステム間で自動的に転送されます．M 形式と IEEE 形式のデータ変換も自動です．この解釈が適当でない場合は， DD 文の DATATYPE パラメータでデータ形式および転送方法を指定してください[45]．

● DATATYPE パラメータ

DATATYPE パラメータは DATATYPE= の後で（ ）で括りカンマで続けて指定します．パラメータは 4 つあります．

DATATYPE=(*para1*,*para2*,*para3*,*para4*)

何番目を指定するかはっきりしているなら，パラメータとカンマは省略できます．例えば DATATYPE=(*para1*) は第 1 パラメータの指定， DATATYPE=(,,*para3*) は第 3 パラメータの指定です．

●パラメータの説明

- 第 1 パラメータは指定したデータセットのデータ形式を指定します．

SRC	ソースプログラム
TXT	書式付き順次入出力データ
FBIN	書式なし順次入出力データ
BIN	書式なし直接入出力データ

  FORT.SYSIN または FORT.SYSINC で指定したデータセットはSRC，その他のデータセットは TXT が省略値となります．

- 第 2 パラメータは文字コードの変換 (EBCDIC-ASCII, JEF ⇔ ASCII, EUC) の有無を指定します．

CNV	文字コードを変換する
NOCNV	文字コードを変換しない

  データ形式が SRC または TXT の場合は CNV，FBIN または BIN の場合は NOCNV が省略値となります．

- 第 3 パラメータは MSP から UXP へのデータセットの転送の有無を指定します．

EXP	データセットを転送する
NOEXP	データセットを転送しない

  DISP パラメータに OLD または SHR を指定した場合は EXP，NEW または MOD の場合は NOEXP が省略値となります．

- 第 4 パラメータは UXP から MSP へのデータセットの転送の有無を指定します．

IMP	データセットを転送する
NOIMP	データセットを転送しない

  データセットが UXP で更新された場合は IMP が省略値となります．

[45]特に M-1800/20U, VP2600/10 用のバイナリデータ (IBM 形式) を read/write する場合は，必ず DD 文の DATATYPE パラメータの第一パラメータにFBIN をすることをお勧めします．

●パラメータの指定例

- データセットのデータ形式が書式なし順次入出力データの場合.

```
//GO.FT01F001 DD DSN=A79999A.IN.DATA,DISP=SHR,DATATYPE=(FBIN)
```

- データセットのデータ形式が書式なし直接入出力データの場合.

```
//GO.FT02F001 DD DSN=A79999A.IO.DATA,DISP=SHR,DATATYPE=(BIN)
```

- 既存のデータセットにデータを出力する場合 (MSP から UXP への転送が必要ない場合).

```
//GO.FT03F001 DD DSN=A79999A.OUT.DATA,DISP=SHR,DATATYPE=(,,NOEXP)
```

4.4.4 JCL の記述例

●従来の JCL 例

汎用計算機 M-1800/20U から前スーパーコンピュータ VP2600/10 へ翻訳／結合編集／実行処理を依頼する従来の JCL の例です.

```
//A79999A1 JOB CLASS=V
// EXEC FORT,VP=YES,OPT=F,OPTION='OPTMSG(EVAL)'
//FORT.SYSIN  DD DSN=A79999A.PROJ.FORT,DISP=SHR
//GO.SYSIN    DD *
5  5  3600
//GO.FT01F001 DD DSN=A79999A.ARRAY.DATA,DISP=SHR
//GO.FT02F001 DD DSN=A79999A.REPORT.DATA,DISP=SHR
//GO.FT03F001 DD DSN=A79999A.NEWA.DATA,DISP=(NEW,CATLG),UNIT=PUB,
//             SPACE=(TRK,(100,10),RLSE)
//
```

●連携用 JCL

汎用計算機 M-1800/20U から VPP700/56 へ翻訳／結合編集／実行処理を依頼する連携用 JCL の例です.

```
//A79999A1 JOB CLASS=X                                              (1)
//  EXEC FORT,VPP=YES,OPT=F,OPTION='-Ee -Ps -Wv,-m3'                (2),(3)
//FORT.SYSIN   DD DSN=A79999A.PROJ.FORT,DISP=SHR
//GO.SYSIN     DD *
5  5  3600
//GO.FT01F001  DD  DSN=A79999A.ARRAY.DATA,DISP=SHR,DATATYPE=(FBIN)   (4)
//GO.FT02F001  DD  DSN=A79999A.REPORT.DATA,DISP=SHR
//GO.FT03F001  DD  DSN=A79999A.NEWA.DATA,DISP=(NEW,CATLG),UNIT=PUB,  (4)
//             SPACE=(TRK,(100,10),RLSE),DATATYPE=(FBIN)
//
```

○説明

(1) 連携ジョブはジョブクラス W, X, Y または Z を指定します.

(2) 連携ジョブはカタログドプロシジャ FORT の “VPP=YES” パラメータを指定します.

(3) 翻訳時オプションを指定する “OPTION=” パラメータは UXP のコマンドラインのオプションを大文字／小文字を区別して記述します. @FORTRAN 行を利用して指定することもできます.

(4) テキスト形式以外のデータセットの場合は “DATATYPE” パラメータでデータ形式を指定します.

なお, 最適化オプションの省略値は VP2600/10 では OPT=B でしたが, VPP700/56 では OPT=E(-Oe) に変更されていますので注意してください[46].

●リージョンサイズの設定

```
//A79999AK JOB CLASS=X
//  EXEC FORT,VPP=YES,OPTION='-Ps -Wv,-m3',VREGION=1740
//FORT.SYSIN DD DSN=A79999A.PROG.FORT,DISP=SHR
//
```

1PE あたりの記憶領域を 1.7GB 確保することを指定します. 省略された場合は 1PE あたり 0.5GB が制限値となります.

●区分データセットの全メンバの翻訳

```
//A79999AW JOB CLASS=W
//  EXEC FORT,VPP=YES,OPTION='-Ps -Wv,-m3',VPPSRC=PO
//FORT.SYSIN DD DSN=A79999A.PROG.FORT,DISP=SHR
//
```

区分データセット A79999A.PROG.FORT の全メンバを翻訳します. VPPSRC=PO の指定は, VPP=YES が指定されたときのみ有効です.

従来の 翻訳オプション ELM(*) の代替となります.

4.5 ジョブの投入

バッチジョブの投入は, 従来と同じ SUBMIT コマンドです. PFD 内からの SUBMIT サブコマンド (SUB と省略可) も使用できます.

```
EDIT --- A79999A.MVPP.CNTL(JOB6) ----------------------------- 表示欄  001 072
コマンド ===> SUB  ↵                                          移動量 ===> 8
****** ***************************  データの先頭  **************V10L30********
000100 //A79999A1 JOB CLASS=W
000200 //  EXEC FORT,VPP=YES,OPTION='-Kfast -Wv,-m3 -Ps'
000300 //FORT.SYSIN DD DSN=A79999A.MVPP.FORT(JOB6),DISP=SHR
000900 //
****** ***************************  データの末尾  ****************************
```

[46] -Oe オプションは, 最適化による副作用の可能性がある反面, プログラムによっては数十倍の性能差が出ます.

```
***           A79999A1        : (RECEIVED) ***
        *** A79999A1 (J5502) A79999A : (JOB ACCEPTED) *** FIB   CN(01)
JOB A79999A1(JOB05502) SUBMITTED
```

4.6 実行状況および結果の確認

連携ジョブの状態は STATE コマンド，STATUS コマンドで確認できます．実行結果はこれまで同様 MSO コマンドなどで検索できます．

```
READY
STATE ↵

TIME=22.46.34  DATE=97.01.08
TSS USER 0010
JOBNAME  STEP     CLS REGION E-REGION   CPU        START       ACCEPT    SYSTEM
A79999A1 FORT      W  ******   1700MB 00:02:23  01/08-22:46 01/08-22:46  VPP700

***************** INFORMATION OF WAITING JOB FOR EXECUTION *****************
* : JOBCLASS :  A  :  B  :  F  :     V     :  N  :  W  :  X  :  Y  :  Z  : *
* :   TYPE   :     :     :     :  S  :  L  :     :     :     :     :     : *
* +----------+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+ *
* :  M-1800  :   0 :   0 :   1 :    ---    :   0 : --- : --- : --- : --- : *
* :  VPP700  : --- : --- : --- :    ---    : --- :   0 :   2 :   0 :   1 : *
* :  VP2600  :   0 :   5 : --- :   4 :   4 : --- : --- : --- : --- : --- : *
*********************************************************** 22:46 ON 01/08/97 **
```

4.7 連携ジョブのキャンセル

キャンセル方法は従来のジョブと同じです．CANCEL コマンドを使用してください．

```
 READY
ST ↵                                          <--- ジョブ番号の確認
 JOB A79999A1(JOB05502) IS EXECUTING
 JOB A79999A#(TSU02073) IS EXECUTING
 JOB A79999A#(TSU02131) IS EXECUTING ON THIS TERMINAL
 READY
CANCEL A79999A1 ↵                             <--- ジョブのキャンセル
 READY
```

●連携ジョブのジョブログ例

連携ジョブは，ジョブログ部分に KKA で始まる識別子を持つメッセージが出力されます．また，メッセージ KKA002I で UXP 側で消費した CPU 時間を知ることができます．

```
                    J E S    J O B  L O G  --  S Y S T E M  S P E X  --  N O D E  J P N C C K U

22.46.16 JOB 5499          *** A79999A1 (J5499) A79999A : START    TIME=22.46.16
23.03.48 JOB 5499  CD=0000 *** A79999A1 (J5499) A79999A : END      TIME=23.03.48

                           <<<  JCL STATEMENTS LIST  >>>    DATE  01/08/97    TIME  22:46

   1     //A79999A1 JOB CLASS=W                                                  JOB 5499
   2     //  EXEC FORT,VPP=YES,OPTION='-Kfast -Wv,-m3 -Free -Ps -Po'
   7     //FORT.SYSIN DD DSN=A79999A.MVPP.FORT(JOB1),DISP=SHR
  17     //GO.SYSIN    DD DSN=A79999A.JOB1.DATA,DISP=SHR
  18     //GO.FT07F001 DD DSN=A79999A.OUT1.DATA,DISP=(NEW,CATLG),UNIT=PUB,
         //       SPACE=(TRK,(100,10),RLSE),DATATYPE=(FBIN)
  19     //GO.FT08F001 DD DSN=A79999A.OUT2.DATA,DISP=(NEW,CATLG),UNIT=PUB,
         //       SPACE=(TRK,(100,10),RLSE),DATATYPE=(FBIN)
  20     //GO.FT54F001 DD DSN=A79999A.OUT3.DATA,DISP=(NEW,CATLG),UNIT=PUB,
         //       SPACE=(TRK,(100,10),RLSE),DATATYPE=(FBIN)
  21     //GO.FT55F001 DD DSN=A79999A.OUT4.DATA,DISP=(NEW,CATLG),UNIT=PUB,
         //       SPACE=(TRK,(100,10),RLSE),DATATYPE=(FBIN)

                           <<<  SYSTEM MESSAGES LIST  >>>
KDS40613I USER(A79999A) LAST ACCESS DATE(1997.01.08),TIME(22:43:49)
KKA101I VPID       CPU                VU              VIRT
        VP000      1MIN 12.70SEC      0MIN 00.00SEC    20128K
KKA001I STEP/FORT    / START  97008.2246
KKA002I STEP/FORT    / STOP   97008.2247 CPU  1MIN 12.70SEC ELAPSE  1MIN 15.00SEC VIRT 20128K
JDJ142I A79999A1 FORT - STEP WAS EXECUTED - COND CODE 0000
JDJ373I STEP/FORT    / START  97008.2246
JDJ374I STEP/FORT    / STOP   97008.2247 CPU  0MIN 00.21SEC  SRB 0MIN 00.03SEC VIRT1628K ERGN 2328K
KKA101I VPID       CPU                VU              VIRT
        VP000      0MIN 01.67SEC      0MIN 00.00SEC    52800K
KKA001I STEP/LKED    / START  97008.2247
KKA002I STEP/LKED    / STOP   97008.2247 CPU  0MIN 01.67SEC ELAPSE  0MIN 03.00SEC VIRT 52800K
JDJ142I A79999A1 LKED - STEP WAS EXECUTED - COND CODE 0000
JDJ373I STEP/LKED    / START  97008.2247
JDJ374I STEP/LKED    / STOP   97008.2247 CPU  0MIN 00.08SEC  SRB 0MIN 00.01SEC VIRT1628K ERGN 2328K
KKA101I VPID       CPU                VU              VIRT
        VP000     15MIN 20.44SEC     12MIN 58.42SEC   212992K
KKA001I STEP/GO      / START  97008.2248
KKA002I STEP/GO      / STOP   97008.2303 CPU 15MIN 20.44SEC ELAPSE 15MIN 31.00SEC VIRT 212992K
JDJ142I A79999A1 GO - STEP WAS EXECUTED - COND CODE 0000
JDJ373I STEP/GO      / START  97008.2247
JDJ374I STEP/GO      / STOP   97008.2303 CPU  0MIN 00.48SEC  SRB 0MIN 00.07SEC VIRT 1632K ERGN 2328K
JDJ375I  JOB/A79999A1/ START  97008.2246
JDJ376I  JOB/A79999A1/ STOP   97008.2303 CPU  0MIN 00.77SEC  SRB 0MIN 00.11SEC
```

4.8 NLP への出力

NLP への出力，OUTPUT コマンドによるジョブログのデータセットへの保存方法は M-1800/20U と全く同じです．

5　VP2600/10 からの移行

この章では，従来の VP2600/10 の Fortran 資産を VPP700/56 に移行するための注意事項について述べます．VP2600/10 の Fortran 資産は汎用計算機 M-1800/20U のデータセット・ファイルとして保存されています．

図 5.1：旧スーパーコンピュータ VP2600/10

5.1　VPP700/56 と VP2600/10 の差異

VP2600/10 と VPP700/56 では，以下の点が大きく異なります．

表 5.1: VP2600/10 と VPP700/56 の差異

	VPP700/56	VP2600/10
計算機	ベクトル並列計算機 (1PE での実行も可能)	ベクトル計算機 (1PE)
OS	UXP/V	MSP, UXP/M
浮動小数点形式	IEEE 形式	M 形式 (IBM 形式)
Fortran	Fortran 90 (+ 富士通並列化仕様)	FORTRAN 77 (+ 富士通仕様)
最大記憶域	1PE あたり 1.7GB* (最大 1.7×32GB)	400MB*
理論ピーク性能	1PE あたり 2.2GFLOPS (最大 2.2×32GFLOPS)	4.9GFLOPS

*) 利用者が確保できる最大リージョンサイズです．

5.2　ベクトル並列計算機

VPP700/56 は，合計 56 台のプロセッサで構成される並列計算機です．各プロセッサ (PE) は独立したベクトル計算機として機能しますので，「ベクトル計算機が 56 台並列に並んだものが VPP700/56 なんだな」とイ

メージして構いません.

利用者は従来の VP2600/10 の Fortran プログラムおよび C プログラムを VPP700/56 の 1PE でそのまま実行することができます．また，並列処理のための指示行 (拡張最適化制御行) をソースプログラムに記述することで最大 32PE を用いた並列処理を実行することもできます.

1PE での CPU 時間は理論性能比で計算すれば 2 倍以上になりますが，複数 PE によるジョブ処理により，特に繁忙期のターンアラウンドは格段に向上するはずです．また，記憶領域は 1PE で従来の 4 倍以上が確保できるようになりました．さらに，並列処理では最大 54.4GB(1.7GB×32) の記憶領域が確保できます.

処理性能は (必ずしも PE 台数に比例して向上するとは言えませんが) 理論上は 32PE で 70.4GFLOPS(2.2×32GFLOPS) となります．実際，並列化により理論ピーク性能の 70% 以上を実現するプログラムも報告されていますので，アプリケーションレベルの実行性能でも VP2600/10 の 10 倍以上は実現可能です.

また，主記憶容量の増大にともない， VP2600/10 でサポートしてきた SSU(システム記憶装置) は廃止されました.

5.3 UXP/V

VPP700/56 の OS(オペレーティングシステム) は UNIX System V Release 4 をベースとした分散並列オペレーティングシステム UXP/V です．従来の VP2600/10 では， IBM 型 OS である MSP EX と UNIX OS である UXP/M の 2 つの OS をサポートしていましたが， VPP700/56 では UNIX に一本化されました.

ただし， MSP と UXP/V との連携機能により，従来の VP2600/10 と同様の書式の JCL(ジョブ制御文) を汎用計算機 M-1800/20U の MSP から投入し，結果を再び MSP で検索する機能 (M-VPP 連携機能) がサポートされています． MSP からのジョブの投入については 4章を御覧ください.

5.4 浮動小数点形式

VPP700/56 は浮動小数点を IEEE 形式で表現します．一方， VP2600/10， M-1800/20U は M 形式(IBM 形式)です．単精度および倍精度型データに対する表現方法の違いは表 5.2 の通りです.

表 5.2: 浮動小数点形式の違い

型名		IEEE 形式	M 形式
単精度型	指数部	8 ビット	7 ビット
	仮数部 *	24 ビット	24 ビット
倍精度型	指数部	11 ビット	7 ビット
	仮数部 *	53 ビット	56 ビット

* 隠れビットを含む.

IEEE 形式は仮数部に 1 ビットの “隠れビット” を持っています．表は隠れビットも含んでいます.

指数部で IEEE 形式は M 形式に比べより広い数を表現できます．その代わり仮数部では倍精度型で 3 ビット少なくなります．しかし， 10 進の精度で見ればほぼ M 形式と同じ 15 桁が表現できます.

IEEE 形式は，ほとんどのワークステーションで採用されている浮動小数点形式で，ワークステーション上で書式なし出力文によって作成したデータをそのまま VPP700/56 で読み込むことが可能です.

5.4.1 M 形式と IEEE 形式の変換

実行時オプション `-Wl,-C` (cf. 付録 B) を指定することで，書式なし入出力文 (従ってバイナリデータです) の実行時に READ 文で M 形式から IEEE 形式に， WRITE 文で IEEE 形式から M 形式にデータを変換することができます[47].

[47] 汎用計算機 M-1800/20U(M 形式) で `-Wl,-C` を指定した場合はこの逆の変換になります.

ただし，変換可能な型は，実数型，倍精度実数型，複素数型，倍精度複素数型のみです．それ以外の型がデータの入出力並びに指定されている場合は形式の 変換ができません ので注意してさい．

VPP700/56 上で作成された実行ファイルを a.out とすると，シェルスクリプトの例は

```
#
frt -Ps -Wv,-m3 test.f90
setenv fu01 input.data
setenv fu02 output.data
a.out -Wl,-C              <--- すべての入出力を M 形式で行う
```

となります[48]．

“-Wl,-C” に続けて装置番号を指定すると，その番号のみの入出力が M 形式で行われます． M 形式で作成されたバイナリデータを装置番号 1 番から読み込んで，処理結果を IEEE 形式で装置番号 2 番に書き出したい時は，次のように書きます．

```
#
frt -Ps -Wv,-m3 test.f90
setenv fu01 input.data
setenv fu02 output.data
a.out -Wl,-C1              <--- 装置番号 1 番からの入出力を M 形式で行う
                                2 番の入出力は IEEE 形式
```

また， Fortran 90/VP システムでサポートされている IETOM, MTOIE サービスサブルーチンによる浮動小数点形式の変換も可能です ([1])．引用形式は以下の通りです．

```
CALL IETOM(R1,R2,TYPE,RETCD)
```

R1	:	変換前の IEEE 形式データ．スカラー変数または配列要素名．
R2	:	変換後の M 形式データ．スカラー変数または配列要素名．
TYPE	:	データの型を指定． 0= 実数型，1= 倍精度実数型．
RETCD	:	復帰コード． 4 バイト整数型． 0= 正常終了，4= 仮数部が最大 3 ビット損失． 8= 指数オーバーフロー，または指数アンダーフロー，または非数値データが検出された． 12= 引数 TYPE の指定に誤りがある．

```
CALL MTOIE(R1,R2,TYPE,RETCD)
```

R1	:	変換前の M 形式データ．スカラー変数または配列要素名．
R2	:	変換後の IEEE 形式データ．スカラー変数または配列要素名．
TYPE	:	データの型を指定．IETOM と同じ．
RETCD	:	復帰コード．IETOM と同じ．

5.4.2 精度の損失

実行時 -Wl,-C オプションによる浮動小数点データの変換では，もともとの仮数部と指数部の持つ情報量の差から，表 4.3 のような精度の損失が起きる可能があります．

[48] もちろん setenv ではなく，ソースプログラムの OPEN 文にファイル名を記述しても構いません．

表 5.3: 精度の損失の可能性

型名		READ(M → IEEE)	WRITE(IEEE → M)
実数型	指数部	10^{-37}〜10^{38} の範囲で変換可能	変換可能
	仮数部	変換可能	最大 3 ビットを損失する可能性がある
倍精度実数型	指数部	変換可能	10^{-78}〜10^{75} の範囲で変換可能
	仮数部	最大 3 ビットを損失する可能性がある	変換可能

複素数型は実部と虚部それぞれに上の型に応じた情報の損失が起きる可能性があります．実行時オプション `-Wl,-M` を指定すると，変換処理の過程で生じたビット損失，オーバーフローなどに関するメッセージが出力されます．

```
a.out -Wl,-C,-M  <--- ビット損失の検査
```

5.5 ハードウェアの違いによる影響

VPP700/56 と VP2600/10 とは，ハードウェアの違いから実行結果が異なる可能性があります．例えば，かなり厳しい (マシンイプシロンに近い) 収束条件を IF 文に組み込んでいるような (精度に敏感な) プログラムでは，実行結果が変わる可能性がありますので注意してください．

実行結果に影響を与えるハードウェアの要因は以下の通りです．

- 浮動小数点形式の違いによる精度差 (5.4節参照)．
- 丸めモードの差． VPP700/56 ではスカラー命令とベクトル命令で丸めモードが異なります． VP2600/10 では同じ丸めモードです．
- 総和演算の演算順序が VPP700/56 と VP2600/10 で異なります．
- VPP700/56 ではメモリアクセスは正しい境界になければなりません． VP2600/10 で正常に動作したプログラムでも，境界の不整合がある場合，実行結果異常となる場合があります．

5.6 言語仕様レベルによる差異

VPP700/56 の Fortran システムは， Fortran 90， FORTRAN 77，および FORTRAN IVの言語仕様に対応しています．ただし，並列プログラムを実行する場合は， FORTRAN IV 言語仕様は利用できません．

言語レベルはサフィックスにより自動的に決まります．翻訳時オプションで指定することも可能です．

表 5.4: サフィックスにより決まる言語レベル

言語仕様	解釈条件
Fortran 90	ファイルのサフィックスが `.f90` 翻訳時オプション `-X9` を指定
FORTRAN 77	ファイルのサフィックスが `.f` 翻訳時オプション `-X7` または `-Xf7` を指定
FORTRAN IV	翻訳時オプション `-X6` または `-Xf6` または `-v` を指定

```
frt test.f90       <---Fortran 90 プログラムと解釈
frt -X9 test.f     <---Fortran 90 プログラムと解釈
frt test.f         <---FORTRAN 77 プログラムと解釈
frt -X7 test.f90   <---FORTRAN 77 プログラムと解釈
frt -X6 test.f     <---FORTRAN IV プログラムと解釈
```

言語レベルの差による解釈の違いについての詳細は [1], 4.2.10 節を御覧ください.

5.7 FORTRAN77 EX との互換性

5.7.1 動作を保証するには

VP2600/10 の FORTRAN システムであった FORTRAN77 EX/VP システム[49]の動作を Fortran 90/VP でも保証したい場合は，翻訳時オプションとして `-Xf7` を指定します.

```
kyu-vpp% frt -Xf7 test.f90 <---FORTRAN77 EX/VP との互換性を保証
```

ただし，この互換性は《言語の解釈》での互換性であって，ハードウェアの違いから生じる精度の違いは保証できません[50]. FORTRAN77 EX/VP と Fortran 90/VP の具体的な動作の差異は [1], 4.2.11 節を御覧ください.

5.7.2 サポートされない機能

VP2600/10 と VPP700/56 のシステムの機能の違いによってサポートされなくなった機能で特に重要なものを以下に列挙します. 詳しくは [1] の 12 章を御覧ください.

●一次回帰演算のベクトル化

DO ループの 1 回前の実行で定義された値を用いた一次回帰演算は VP2600/10 ではベクトル化されていましたが, VPP700/56 ではベクトル化されません[51].

●デバッグ機能

デバッグオプションは新システムでは `-Da`, `-Ds`, `-Du` のみがサポートされ，それ以外のデバッグオプション `-Db`, `-Di`, `-Do`, `-Dt`, `-Dv` は使えなくなります. ただし高速デバッグオプション `-Dx` がサポートされました.

●組み込み関数のインライン展開

スカラーの Fortran 組み込み関数 (利用者定義の関数ではありません) を引用箇所に展開する機能 (インライン展開) はサポートされなくなりました. また，翻訳時オプション `-Nf` は指定できなくなりました.

● CLOCK, CLOCKV の引数

実行開始からの CPU 占有時間を返却する CLOCK サブルーチン，および CPU 占有時間とベクトルユニット占有時間を返却する CLOCKV サブルーチン[52]の引数は，これまで省略可能なものがありましたが, Fortran 90/VP では省略できなくなりました. 引数を省略した場合は，実行時に 異常終了 します.

---

[49] 汎用計算機のコンパイラと同じです.
[50] `-Xf7` オプションは主に入出力制御に関する互換性を保証するもので，通常は意識しなくても大丈夫です.
[51] 1997 年 4 月現在ベクトル化されていません. 理由はハードウェア命令がサポートされていないためです. 今後ベクトル化を検討するそうです.
[52] 富士通が提供するサービスサブルーチンで，他のメーカーとの互換性はありません.

●翻訳時間

Fortran 90/VP は，従来のベクトル化や最適化にプラスして，スカラープロセッサ向けの最適化を行っています．そのため，翻訳時間は一般的に VP2600/10 と比較して長くなります．また，実行ファイル (a.out) も多少大きくなります．

5.8 MSP から UXP へのファイル転送

MSP(kyu-msp) から UXP(kyu-cc) へのデータ転送方法および注意点は [54]，[40]，[41] を参照してください．

5.9 印刷制御用コマンド

UXP に移行した MSP の Fortran プログラムで， NLP(Nihongo Line Printer) 用の制御文字がファイルに出力される場合， kyu-vpp の`fot(/usr/lang/bin/fot)` コマンドで UNIX のラインプリンタ規則に従ったファイルに変換することができます．

```
kyu-vpp% fot file1 file2 ↵
```

6　数値計算ライブラリの利用

VPP700/56 では，単一 PE 上でベクトル計算向けにチューニングされた Fortran サブルーチンライブラリ SSL II/VP， NUMPACを，また，複数 PE を使った並列処理向けに設計された SSL II/VPPをサポートします.

表 6.1: 数値計算ライブラリ

ライブラリ名	複数 PE	単一 PE	C からの利用
SSL II/VPP	○	△	×
SSL II/VP	▲	○	○
NUMPAC	▲	○	×

▲ ： 冗長実行として利用可能.

△ ： 利用可能だが使うメリットがない.

これらのサブルーチンライブラリの特徴は， VPP700/56 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，一般のライブラリに比べて (普通は) 圧倒的に高速である点と，精度についても十分な考慮がなされており，信頼性が高い点です.

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ため，これらのライブラリを積極的に利用することで格段の性能向上が期待できます.

6.1　SSL II/VPP

SSL II/VPP(Scientific Subroutine Library II for Vector Parallel Processors) は，オーストラリア国立大学の数値計算グループ[53]と富士通が共同開発した並列数値計算ライブラリです.

6.1.1　主な機能

SSL II/VPP は，複数 PE 向き並列高速アルゴリズムをサブルーチン形式で提供します．使用頻度が高く，計算コストのかかる重要な機能をサポートしています.

- 行列積
- 実・複素行列の連立 1 次方程式
- 正定値対称行列の連立 1 次方程式
- 実バンド・スパース行列 (非対称，不定値) の連立 1 次方程式
- 固有値問題 (3 重対角，実対称， Hermite，実対称スパース，一般化)
- 逆行列
- 実・複素 Fourier 変換 (2, 3, 5 の混合基底)
- 線形最小二乗解
- 一様乱数，正規乱数の生成

[53]有名なところでは Richard Peirce Brent さん, Michael Robert Osborne さん.

特徴は，疎な行列に対する (前処理付き) 共役勾配法 (CG 法， ICCG 法, MICCG 法)，修正一般化共役残差法 (MGCR 法)，非対称または不定値のスパース行列に対する TFQMR 法， QMR 法，および高速な一様乱数・正規乱数生成ルーチンをサポートした点です．機能の一覧は付録 Dを参照してください[54].

SSL II/VPP は，並列化 Fortran である Fortran 90/VPP から CALL 文により利用します．従って， Fortran 90/VPP に関する基本的な知識が必要です． C, C/VP からの利用はできません．また，精度はすべて倍精度です．単一 PE 用の SSL II/VP とは機能範囲や呼び出し方法が大きく異なります．

6.1.2 参考文献

SSL II/VPP のオンラインマニュアルは kyu-vpp から man コマンドで， kyu-cc から vman コマンドで検索できます[55].

```
kyu-vpp% man ssl2vpp ↵        <--- サブルーチンリストの出力
SSL2VPP(3)                (VPP SSL II/VPP)                SSL2VPP(3)

［名称］

      ＳＳＬＩＩ／ＶＰＰ　－　科学用サブルーチンライブラリ

［説明］

   (1) 機能概要

      ＵＸＰ／Ｖ　ＳＳＬＩＩ／ＶＰＰは，線型方程式や離散型フーリエ変換
       などの数学的問題を，並列的に解く，約３６種類のサブルーチンからな
       る汎用数値計算ライブラリです．各サブルーチンは，ＶＰＰ　ＦＯＲＴ
       ＲＡＮで記述されたスプレッド手続きであり，利用者プログラムから，
      ＰＡＲＡＬＬＥＬ　ＲＥＧＩＯＮ内で，ＣＡＬＬ文を用いて使用できます．
                          :
```

個別のサブルーチンは，名前を**小文字**で入力することで機能，引数を検索できます．

```
kyu-vpp% man dp_vmggm ↵        <--- サブルーチン DP_VMGGM の検索
                :
```

また， SSL2/VPP を十分に使いこなすためには，参考文献 [19] および疎行列の格納方法を調べるためには [18] が手元に必要です． SSL II/VP と異なりアルゴリズムの詳細な解説はありませんが，参考文献は明示してあります．並列数値計算の参考資料へのポインタとしての価値もありますので，並列計算に興味のある方は購入されてはいかがでしょうか．

6.1.3 追加機能

第 1 版以降に追加されたサブルーチンは表 6.2 の通りです．

[54]必ずしも十分とは言えませんが，今後の共同研究でさらなる機能アップを行うそうですので，期待しましょう．

[55]英語で見たい場合は環境変数 LANG を変更して下さい．具体的には “unsetenv LANG” と入力します．日本語に戻したい時は “setenv LANG japan” です．

表 6.2: SSL II/VPP の新機能

サブルーチン名	機能
DP_VTFQD	非対称または不定値のスパース行列の連立 1 次方程式 (TFQMR 法，対角形式格納法)
DP_VTFQE	非対称または不定値のスパース行列の連立 1 次方程式 (TFQMR 法， ELLPACK 形式格納法)
DP_VQMRD	非対称または不定値のスパース行列の連立 1 次方程式 (QMR 法，対角形式格納法)
DP_VQMRE	非対称または不定値のスパース行列の連立 1 次方程式 (QMR 法， ELLPACK 形式格納法)
DP_VSEVPH	実対称行列の固有値・固有ベクトル (3 重対角化，マルチセクション法，逆反復法)
DP_VHEVP	Hermite 行列の固有値・固有ベクトル
DP_VTDEVC	実 3 重対角行列の固有値・固有ベクトル
DP_VLAND	実対称スパース行列の固有値・固有ベクトル (Lanczos 法，対角形式格納法)
DP_VRANN3	正規乱数の生成

なお，実 3 重対角行列の固有値・固有ベクトルを求めるサブルーチン DP_VTDEV は DP_VTDEVC に移行しています.

6.2 SSL II/VP

SSL II/VP(Scientific Subroutine Library II for Vector Processor) は，連立 1 次方程式や微分方程式などの数値計算を行う約 230 種類のサブルーチンからなる VPP700 の単一 PE 向け汎用数値計算ライブラリです. 各サブルーチンは Fortran 90/VP プログラムから使用できます．また， C/VP から呼び出すことも可能です (cf.[49]). 使用方法は M-1800/20U の SSL II/VP と変わりませんが， VPP700/56 用の『拡張機能 II』があります[56].

6.2.1 拡張機能 II の一覧

○線形計算

VLSBX	正値対称バンド行列の連立 1 次方程式 (変形 Cholesky 分解)
VBLDL	正値対称バンド行列の LDL^T 分解 (変形 Cholesky 分解)
VBLDX	LDL^T 分解された正値対称バンド行列の連立 1 次方程式
VLBX	実バンド行列の連立 1 次方程式 (Gauss の消去法)
VBLU	実バンド行列の LU 分解 (Gauss の消去法)
VBLUX	LU 分解された実バンド行列の連立 1 次方程式
VCGD	正値対称スパース行列の連立 1 次方程式 (前処理付き CG 法，対角形式格納法)
VCGE	正値対称スパース行列の連立 1 次方程式 (前処理付き CG 法， ELLPACK 形式格納法)
VCRD	非対称または不定値スパース行列の連立 1 次方程式 (MGCR 法，対角形式格納法)
VCRE	非対称または不定値スパース行列の連立 1 次方程式 (MGCR 法， ELLPACK 形式格納法)
VMVSD	スパース実行列と実ベクトルの積 (対角形式格納法)
VMVSE	スパース実行列と実ベクトルの積 (ELLPACK 形式格納法)

“ELLPACK 形式” とは係数行列の各行ベクトルの非ゼロ要素のみを圧縮して格納する方法です． “対角形式” は非ゼロ要素のある対角方向のベクトルを格納する方法です． [18] には楕円型偏微分演算子 (Dirichlet 境界条件) を差分法で離散化した係数行列の具体的な作成方法が示してあります.

[56] 拡張機能 II の部分もオーストラリア国立大学の数値計算グループと富士通が共同開発したサブルーチン群です

○ Fourier 変換

VCPF3	3 次元素因子離散型複素 Fourier 変換
VMCFT	1 次元，多重，多次元離散複素 Fourier 変換 (混合基底)
VRPF3	3 次元素因子離散型実 Fourier 変換
VMRFT	多重・多次元離散型実 Fourier 変換 (2,3 および 5 の混合基底)
VSRFT	1 次元・多重離散型実 Fourier 変換 (2,3 および 5 の混合基底)

○乱数

DVRAN3	正規乱数の生成 (倍精度)
DVRAU4	一様乱数 $[0,1)$ の生成 (倍精度)

短くても 10^{52} 以上の長い周期で統計的特性の優れた乱数を発生します.

6.2.2 参考文献

SSL II/VPP と同様，man コマンド経由で検索下さい．機能一覧は man ssl2 です．付録 Eも御覧下さい．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.

```
kyu-vpp% man dvlax ↵
VLAX(3SSL2)          (Scientific Subroutine Library)          VLAX(3SSL2)

[名称]

       ＶＬＡＸ，ＤＶＬＡＸ　－　実係数の連立１次方程式
                                （ブロッキングＬＵ分解法）

[記述形式]

       CALL VLAX(A,K,N,B,EPSZ,ISW,IS,VW,IP,ICON)

[説明]

   (1) 機能

       実係数連立１次方程式

           Ａｘ＝ｂ
            :
```

新しくサポートされた機能の詳細は [18] を参照してください.

6.3 NUMPAC

NUMPAC(Nagoya University Mathematical PACkage) は名前の通り，もともとは名古屋大学大型計算機センター研究開発部及び名古屋大学工学部数値解析研究グループを主として開発された数値計算パッケージです． 1994 年に富士通に移管されていますが，残念ながら保守のみの管理であり，バージョンアップなどは計画されていません.

使用は単一 PE です．各サブルーチンは Fortran 90/VP から使用できます[57]．

6.3.1 制限事項

機能は従来の VP2600/10 と比べ，システムの違いから以下の制限があります．

1. e の高速高精度計算ルーチン FASTEE，π の高速高精度計算ルーチン FASTPI はサポートされません[58]．
2. 図形表示応用プログラム CONRM, CONTOR, CONT1M, CONT1S, CTL2, SOLMOR, SOLRM, TRIMAP はサポートされません[59]．
3. ビットごとの論理演算ルーチン IAND, IOR, IEOR は Fortran 90/VP の組み込み関数として実現されているため，未サポートです．
4. 浮動小数点形式の違いから，引数値の許容範囲が異なる場合があります．特に単精度ルーチンでは一般に引数の範囲が狭くなります．
5. NUMPAC のプログラムと Fortran 90/VP の組み込み関数が重複し，NUMPAC を使用したい場合は，EXTERNAL 文での宣言が必要です．

NUMPAC の機能一覧は付録 Fを，マニュアルは [20] を参照下さい．またオンラインマニュアルにセンターのホームページからたどることができます．

6.4 サブルーチンライブラリの使用方法

各サブルーチンライブラリは，Fortran プログラムから CALL で呼び出します．SSL II/VP を C/VP プログラムから呼び出す場合は，各パラメータをポインタで渡します．なお，C/VP から SSL II/VP を利用する場合は，配列の格納順序に注意する必要があります．詳しくは [49] を御覧ください．

6.4.1 単一 PE の場合

リンク・エディタ ld に渡すオプション -l に続けて -lssl2vp，または -lnumpac と指定します．
以下は対話型処理の例です．シェルスクリプトの記述も同様です．

○ SSL II/VP の利用

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．

```
kyu-vpp% frt -Ps -Wv,-m3 test.f90 -lssl2vp   <--- 翻訳．SSL II/VP を結合
kyu-vpp% a.out                               <--- 実行
```

○ NUMPAC の利用

NUMPAC のサブルーチンを使用している場合は，-lnumpac を指定します．

```
kyu-vpp% frt -Ps -Wv,-m3 test.f90 -lnumpac    <--- 翻訳．NUMPAC を結合
kyu-vpp% a.out                                <--- 実行
```

[57] C/VP からの利用も SSL II/VP と同様に基本的には可能だと思われますが，正式なサポートがないことから当面センターでは「利用できない」ことにします．
[58] 富士通 M シリーズに依存したアセンブラで書かれているため移植を断念しています．
[59] 汎用機の図形処理パッケージ PSP を使用しているため移植不可能です．

○ C/VPから SSL II/VP の利用

リンクは vcc コマンドではなく frt コマンドで行います．その際 -lcvp，-lm オプションも指定して下さい．

```
kyu-vpp% vcc -c test.c                          <---vcc で翻訳， test.o を作成
kyu-vpp% frt test.o -lssl2vp -lcvp -lm          <---SSL II/VP を結合
kyu-vpp% a.out                                  <--- 実行
```

○ SSL II/VPP の利用

同様に -lssl2vpp と指定します．翻訳時には Fortran 90/VPP を起動するオプション -Wx を必ず指定します．以下は翻訳から実行までを行なうバッチリクエストの例です．

```
#                                       <---csh で記述
cd EXAMPLE                              <--- ディレクトリの移動
frt -Wx -Ps -Wv,-m3 test.f90 -lssl2vpp  <--- 翻訳．SSL II/VPP を結合
a.out                                   <--- 実行
```

6.5 注意事項

SSL II/VP, SSL II/VPP, NUMPAC を利用して得られた結果を公表する場合には，使用プログラム名について言及することをお願いします[60]．

6.6 SSL II/VPP の演算性能

この節では，参考データとして SSL II/VPP の代表的なサブルーチン性能を調べたデータを幾つか紹介します．

6.6.1 56PE でのピーク性能試験

1996 年 12 月 5 日，富士通沼津工場で VPP700/56 の現地テストを行ないました．その際，ピーク性能試験として行列積演算の Fortran プログラムを 56PE で並列実行しました．プログラムは次数 n の倍精度実数データ型の正方行列 A, B の積を計算し，結果を次数 n の正方行列 C に格納するものです．

行列積演算には SSL II/VPP の DP_VMGGM を用います[61]．行列のデータは各要素を A_{ij}, B_{ij} とするとき，

$$A_{ij} = B_{ij} = \left(\frac{2}{n+1}\right)^{\frac{1}{2}} \sin\left(\frac{ij\pi}{n+1}\right)$$

で定義します．なお π は $4\tan^{-1}1$ とします．

このように定義した理由は，行列積演算の精度の確認として理論的に $AB = E(E$ は単位行列) となることを用いて行列 C に対する要素を C_{ij}， E の要素を E_{ij} とするとき最大行和ノルムの意味での誤差 ($||E||_\infty = 1$ より絶対誤差かつ相対誤差となります．)

$$||C - E||_\infty = \max_{1 \le i \le n} \sum_{j=1}^{n} |C_{ij} - E_{ij}|$$

が知りたかったからです．経過時間はウォールクロックにより行列積演算の開始から結果格納までを測定しました．次元 n の大きさと A, B, C に必要な記憶領域および経過時間，誤差は表 1 の通りです．経過時間は小数点第 1 位以下を，誤差は小数点第 4 位以下を切捨てています．

[60] 数値計算の結果を公表する場合には，計算機，プログラム言語，精度，ライブラリのデータを公表するのは最低限のマナーです．

[61] 普通に行列積のプログラムを組むと数行で済むのですが，ソースリストを見たところ， A4 判 29 頁におよぶ長大なものでした．

表 6.3: 56PE でのピーク性能試験結果

n	記憶領域	経過時間	誤差
18000	7.2GB	105sec.	0.138e-09
40000	35.7GB	1135sec.	0.393e-09
55000	67GB	2865sec.	0.713e-09
60000	80GB	3683sec.	0.681e-09
65000	94GB	5197sec.	0.819e-09

個人的な感想としては，演算数の増大によって着々と増え続ける誤差の方が気になります．もちろん，誤差ノルムの計算自体に混入する丸め誤差も考慮に入れる必要がありますが，大規模な数値計算を行なう場合の丸め誤差の評価は今後きちんと考える必要があるのではないでしょうか．図 6.1 は演算数を $2n^3$ とした GFLOPS 値をプロットしたものです．“Theoretical_Peak” は， 56 台での理論ピーク性能値です．

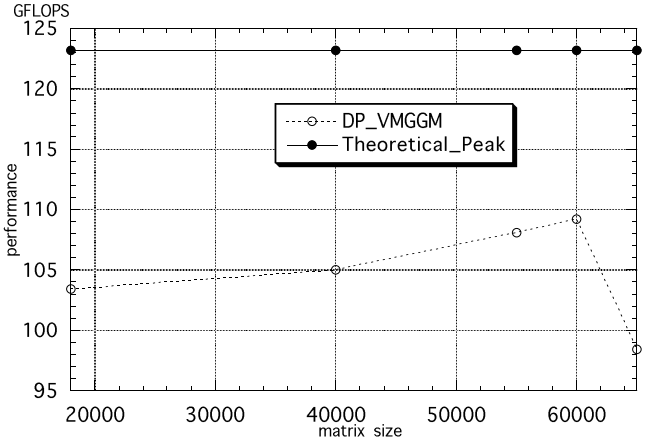


図 6.1: 56PE でのピーク性能試験結果

$N = 65000$ で処理性能が低下した原因は “たまたま” この値に対して性能低下を起こす要因があり，ソースの修正によって $N = 60000$ 以上の性能が出るとの報告をメーカーから受けました．しかし 56PE の性能試験を行なうためにはすべての PE の割り当てを変更する必要があるため，運用を開始した現在ではなかなかテストできません．

6.6.2 台数効果

現在，利用者が使用できる最大 PE 台数は 32 です．以下， 4 つの SSL II/VPP サブルーチンに対し， PE 台数を変化させ，台数効果を調べてみました．測定は 1997 年 4 月〜 6 月に行ないました． FLOPS 値は， Analyzer([11]) の浮動小数点演算数の採取機能 PEPA(PE Performance Analyzer) の値を信用して用いています．

○行列積

図 6.2 は 56PE でのピーク性能試験で使用した同じプログラム (サブルーチン`DP_VMGGM`) の $n = 8500$ に対し PE 台数を変化させたものです． “`Theoretical_Peak`” は，その台数での理論ピーク値です．

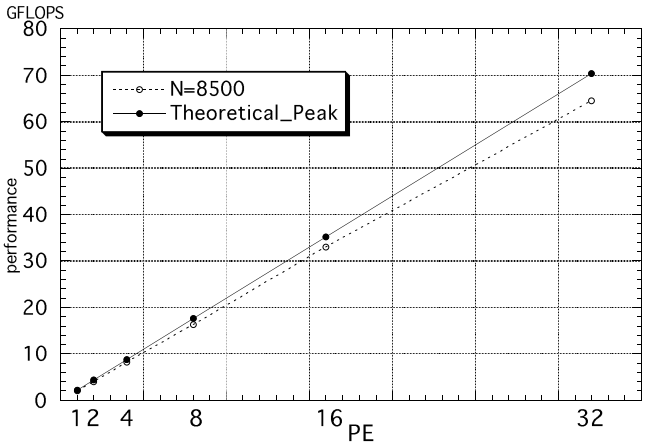


図 6.2: `DP_VMGGM` の台数効果

$N = 8500$ にした理由は，ぎりぎり 1PE で実行可能だからです．各 PE 台数分のメモリーの上限に迫るように N を変化させたデータも採取しましたが，それほど性能は変わりませんでした．

図 6.3 は，実スパース行列と実ベクトルの積を求める DP_VMVSD の性能を示すものです．実スパース行列は，領域 $\Omega = [0,1]^3$ を各方向 $N+1$ 等分割したときの楕円型偏微分演算子を中心差分を用いて離散化した時に得られるものです．[17] による対角形式格納法により格納されています．

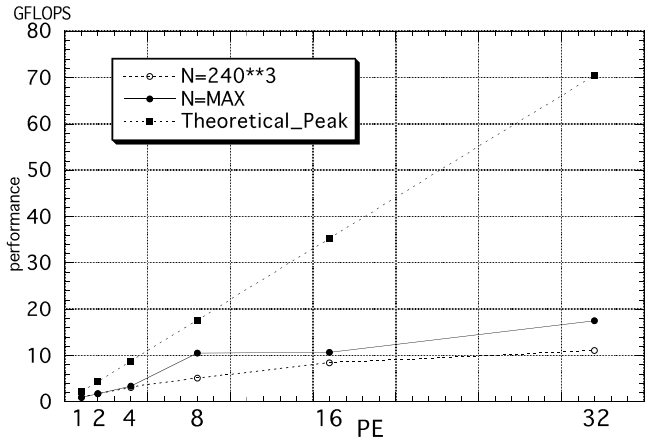


図 6.3: DP_VMVSD の台数効果

“N=MAX” は，各次元の分割数 N を実行可能な記憶領域の上限近くに設定して採取した値をプロットしたものです．

○連立 1 次方程式

図 6.4 は密実行列に対する連立 1 次方程式 DP_VLAX の性能を示すものです．行列は DP_DVMGGM と同じものを用いました．

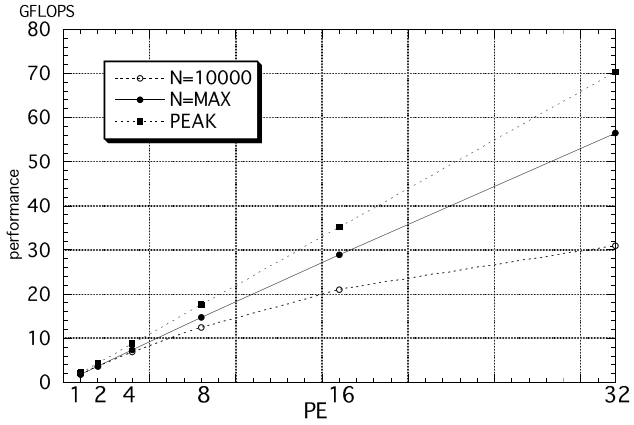


図 6.4: DP_VLAX の台数効果

“N=MAX” の意味は先ほどと同じです．

図 6.5 は非対称または不定値のスパース行列に対する連立 1 次方程式 (MGCR 法) DP_VCRD の性能を示すものです．行列は DP_VMVSD と同じものを用いました．

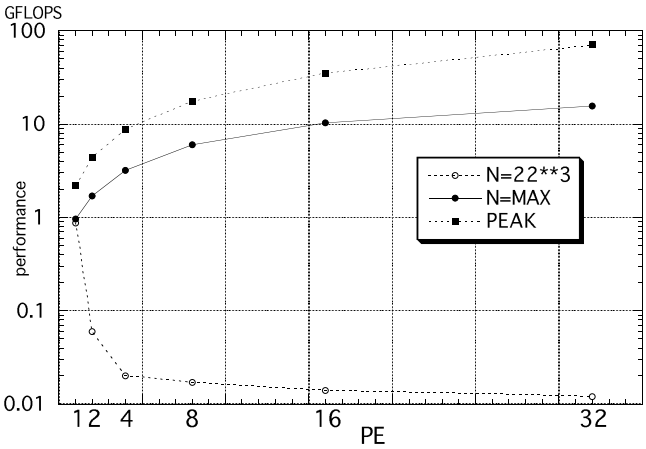


図 6.5: DP_VCRD の台数効果

“N=MAX” の意味は先ほどと同じです．このテストでは，N(ここでは分割数です) の値を小さめに設定してみました．

$N = 22$(従って方程式の次元は 22^3) の場合に方程式を解く時間は数秒です．しかし，PE が増えるに従い，浮動小数点演算に要する時間よりも PE 間のデータ転送にかかるコストが増大するため，性能は悪くなっていきます．このような小規模な問題では PE 台数を増やしてもあまり意味がないことがわかります．

7　プログラムの性能測定

ベクトル化，並列化の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手続きやループを特定することが大事です． VPP700/56 では，ベクトル化，並列化に関する情報を提供するツールとして “Sampler”, “Counter” と呼ばれる実行解析支援ツールが用意されています[62].

7.1　Sampler

Sampler は， Fortran 90/VP, Fortran 90/VPP, C/VP, C++ と C/VP で翻訳した実行ファイルに対し，実行中に一定の周期で割り込みをかけ，プログラム単位，手続き単位，ループや配列式ごとの実行コストのデータを収集します．手続きごとのベクトルヒット率の測定，動的にリンクされたプログラムの解析も可能です．

利用者は実行解析用に特別に再翻訳を行う必要はなく，実行ファイルをそのまま Sampler に渡すことでプログラムのチューニング情報を得ることができます．

ただし，実行中に割り込みをかけながら情報を収集するため実行時間が増えることにご注意下さい．なお，Sampler の利用は UXP のみで MSP からの利用はできません． Sampler についての詳細は [11] を御覧ください．

7.1.1　Sampler を使用する前に

Sampler を起動する前に実行ファイルを用意します．チューニング用の翻訳時オプションの指定は必要ありません．コマンド `frt`(Fortran 90/VP), `frt -Wx`(複数 PE Fortran 90/VPP), `vcc`(C/VP), `CC+vcc` で翻訳します[63]．具体的な手順は 2章を参照してください．

7.1.2　環境変数の設定

Sampler を使用して情報収集，解析を行うために，環境変数名 `FJSAMP` に変数を設定する必要があります．設定できる変数は以下の通りです．

`file:`*filename*	*filename* に Sampler の解析用の情報を出力する．バイナリファイル．
`type:rtime`	経過時間をもとにした情報を出力する．省略した場合は CPU 時間をもとにした情報を出力する．
`interval:`*time*	タイマーの割り込み間隔 ($1 \sim 2^{31}$) をミリ秒単位で指定．省略値は 10．
`pe:on`	各 PE ごとの情報を出力することを指定．並列 Fortran プログラムの場合のみ有効．

環境変数の設定はジョブスクリプトの記述が `sh`(先頭に `#` がない) か `csh`(先頭に `#` がある) かに応じて設定が異なります．使用例を参照してください．

[62] なお， Sampler， Counter に加え，ハードウェア情報採取ツールの PEPA と MPA，デバッグ支援ツール fdb をあわせて “Analyzer” と呼びます．なお，マニュアルに記述されている並列プログラムの対話型並列デバッガ pfdb は運用上の問題から当面利用できません．

[63] 基本的にベクトル化，ベクトル並列化に関する情報を収集しますので，cc で翻訳したスカラー実行ファイルを用意しても無意味です．

7.1.3 解析コマンド

Sampler による解析は fjsamp コマンドによって行います．fjsamp に続けて実行ファイル名を指定します．実行時オプションも指定できます． Fortran 90/VP の経過時間による実行打ち切りを指定する -Wl,-t オプションは指定できません．

7.1.4 対話型処理の例

Analyzer の利用は 1PE のジョブに限って対話的に利用できます．

● Fortran 90/VP の情報収集

```
kyu-vpp% frt test.f90 ↵                          <--- 実行ファイルの作成
kyu-vpp% setenv FJSAMP file:sampler.data ↵       <--- 環境変数の設定
kyu-vpp% a.out ↵                                 <--- サンプラ情報ファイルの作成
kyu-vpp% fjsamp a.out > out ↵                    <--- サンプラによる解析
```

解析結果は標準出力に出力されます．ここではファイル out に結果を書き出しています．sampler.data は情報収集用のファイルです．名前は任意です．

● C/VP の情報収集

C/VP の場合も同様に，作成した実行ファイルを一度実行することでサンプラ情報を作成し fjsamp による解析を行います．

```
kyu-vpp% vcc test.c ↵                            <--- 実行ファイルの作成
kyu-vpp% setenv FJSAMP file:sampler.data ↵       <--- 環境変数の設定
kyu-vpp% a.out ↵                                 <--- サンプラ情報ファイルの作成
kyu-vpp% fjsamp a.out > out ↵                    <--- サンプラによる解析
```

解析結果は標準出力に出力されます．対話的に情報を収集できる実行ファイルは領域が 100MB 以下の 1PE ジョブに限られます．それ以外はバッチ処理になります．

7.1.5 バッチリクエストの記述例

以下は VPP700/56 に投入するバッチリクエストの記述例です．

● Fortran 90/VP(情報収集＋解析)

```
#                                  <---csh で記述
cd EXAMPLE                         <--- ディレクトリの移動
setenv FJSAMP file:sampler.data    <--- 環境変数の設定
a.out                              <---Sampler 情報ファイル sampler.data の作成
fjsamp a.out                       <---Sampler による解析
```

最初の a.out の実行により，sampler.data に情報が収集されます．ファイル名は何でも構いません．次の fjsamp で，収集された情報をもとに解析を行います[64]．

[64]通常，実行時間は情報を収集するための a.out の処理に集中します．

スクリプトを sh で記述する (先頭に # がない) 場合は環境変数の設定が異なります.

```
cd EXAMPLE
FJSAMP=file:sampler.data
export FJSAMP                          <--- 環境変数の設定
a.out
fjsamp a.out
```

● Fortran 90/VP(情報収集と解析を分ける)

情報収集と解析とは分けてスクリプトに記述し，順番に NQS に渡しても構いません．ただし情報ファイル名 (ここでは sampler.data) は同じにします.

```
#
cd EXAMPLE
setenv FJSAMP file:sampler.data        <--- 環境変数の設定
a.out                                  <---Sampler 情報ファイル sampler.data の作成
```

情報ファイル sampler.data を作成した後に，fjsamp により解析します.

```
#                                      <---csh で記述
setenv FJSAMP file=sampler.data,type:rtime <--- 環境変数の設定，経過時間をもとにする
fjsamp a.out                           <---Sampler による解析
```

変数のうち “file” は情報収集，解析とも必ず指定します．“type”，“pe” は情報収集の段階では不要です.

● Fortran 90/VP(翻訳＋情報収集＋解析)

翻訳から解析までを一つのスクリプトで記述することも可能です.

```
#
frt -Ps -Wv,-m3 sample.f90             <--- 翻訳
setenv FJSAMP file:sampler.data        <--- 環境変数の設定
a.out                                  <--- 情報収集
fjsamp a.out                           <--- 解析
```

○単一 PE 版プログラムの解析例

手続きやループ／配列式の全体に占める割合と平均ベクトル長[65]の情報が主な情報となります.

[65] 長いほどベクトル計算機の性能を十分に引き出していると考えて結構です.

```
Status                      : Serial              <--- 逐次実行
Number of Processors        : 1                   <--- PE 数

Type                        : cpu                 <--- CPU 時間情報
Interval (msec)             : 10                  <--- 割り込み間隔

Synthesis Information                             <--- 総合情報
    Count|   Percent|    VL| Name                      Count : 割り込み回数
     7540|      23.7|  1575|  MULMMW_                Percent : 割合 (%)
     6586|      20.7|  1781|  MINVW_                      VL : 平均ベクトル長
     6368|      20.0|  1107|  HOBSVW_                   NAME : 名前
     4702|      14.8|  1273|  GHBSVW_
     3572|      11.2|   742|  MUL_
     3051|       9.6|  1292|  CHOLFW_
        9|       0.0|  2048|  CLEARM_
        9|       0.0|   441|  MAKEA4_
        4|       0.0|     -|  MAKEG_
        3|       0.0|     -|  MAKEA3_
        2|       0.0|     -|  MAKEF_
        2|       0.0|  2048|  MAKEGI_
        1|       0.0|     -|  ELNODE_
        1|       0.0|     -|  MAKELI_

    31850|          |  1389|  TOTAL


Program Unit Information(MULMMW_)---------------------------------
 Procedure List:                                          <--- 手続き情報
    Count|          Percent|    VL| Name
     7540|     100.0( 23.7)|  1575|  mulmmw_

     7540|          ( 23.7)|  1575|  PROCEDURE_TOTAL

 Loop & Array Expression List:                            <--- ループ／配列式情報
    Count|          Percent| V_Mark| kind  |    VL| Line
     7497|      99.4( 23.5)|   V   | DO    |  1575|  00002071-00002072
       41|       0.5(  0.1)|   S   | DO    |     -|  00002070-00002073
        1|       0.0(  0.0)|   S   | DO    |     -|  00002067-00002078
        1|       0.0(  0.0)|   S   | DO    |     -|  00002074-00002077
```

● Fortran 90/VPP(情報収集＋解析)

並列プログラムの解析も同様です．Fortran 90/VPP の場合は変数 “pe:on” により各 PE 毎の情報を得ることができます．

```
#                                       <---csh で記述
setenv FJSAMP file:sampler.data,pe:on   <--- 環境変数の設定，各 PE 毎の情報を出力
a.out                                   <---Sampler 情報ファイル sampler.data の作成
fjsamp a.out                            <---Sampler による解析
```

なお，実行プログラムは `frt -Wx` によって Fortran 90/VPP で翻訳されている必要があります．

○解析結果の見方

詳細は [11] に譲ることにして，重要な箇所だけを表にまとめました．表中の x はその項目の値を変数としたものです．

表 7.1: Fortran 90/VPP の解析項目

項目	意味	範囲	見方
`speedup`	並列効果	$1 \leq x \leq$ PE 数	値が大きいほど並列効果が高い
`ratio`	並列化率	$0 \leq x \leq 1$	値が大きいほど並列化されている
`Load balance`	負荷バランス	$0 \leq x \leq 1$	値が小さいほど PE が効率よく動作
`Asynchronous`	非同期転送待ち発生率	$0 \leq x \leq 1$	値が小さいほどデータ転送待ちが少ない
`ALL`	プログラム全体の割り込み回数	—	—
`AW`	待ち状態割り込み回数	—	回数が少ないほどよい
`AMW`	転送待ち状態割り込み回数	—	回数が少ないほどよい

次ページは並列プログラムの解析例です．

○並列版プログラムの解析例

```
 Status                     : Parallel           <--- 並列実行
 Number of Processors       : 8                  <--- PE 数

 Type                       : cpu
 Interval (msec)            : 10

  Performance Information : Parallel Information :
  Parallel             Parallelization  Parallel to serial  Load balance  Asynchronous    Name
      speedup                   ratio       speed ratio                transfer ratio
   1.05555556            1.00000000        1.08333333    0.01851852      0.00000000  dp_umggmx_
   3.14754098            0.99871589        7.35460993    0.52272727      0.02685950  EST_
   6.50000000            0.97959184        8.00000000    0.07142857      0.00000000  SETDAT_
   1.00000000            1.00000000        1.00000000    1.00000000      0.00000000  MAIN__
   1.85465116            0.99875992        3.37447523    0.19882612      0.00953778  TOTAL

 Synthesis Information (Count)
   PM      PMW     PMMW|      RM      RMW     RMMW|     ALL       AW      AMW|    VL| Name
  108        2        0|     117        3        0|     864       16        0|  1001| dp_umggmx_
   61       56        2|     445      240       13|     484      253       13|     -| EST_
    2        0        0|      14        1        0|      14        1        0|  1006| SETDAT_
    0        0        0|       0        0        0|       1        1        0|     -| MAIN__
  171       58        2|     576      244       13|    1363      271       13|  1001| TOTAL

 Synthesis Information (Percent)
   PM      PMW     PMMW|      RM      RMW     RMMW|     ALL       AW      AMW| Name
 63.2      1.2      0.0|    20.3      0.5      0.0|    63.4      1.2      0.0| dp_umggmx_
 35.7     32.7      1.2|    77.3     41.7      2.3|    35.5     18.6      1.0| EST_
  1.2      0.0      0.0|     2.4      0.2      0.0|     1.0      0.1      0.0| SETDAT_
  0.0      0.0      0.0|     0.0      0.0      0.0|     0.1      0.1      0.0| MAIN__


 Program Unit Information(EST_)-----------------------------------
  Performance Information :      Parallel Information :
  Parallel     Parallelization  Parallel to serial  Load balance  Asynchronous    Name
      speedup             ratio         speed ratio              transfer ratio
    3.14754098       0.99871589          7.35460993    0.52272727   0.02685950  est_
    3.14754098       0.99871589          7.35460993    0.52272727   0.02685950  PROCEDURE_TOTAL

  Procedure List (Count):
   PM      PMW     PMMW|      RM     RMW    RMMW|     ALL      AW     AMW|   VL| Name
   61       56        2|     445     240      13|     484     253      13|    -| est_
   61       56        2|     445     240      13|     484     253      13|    -| PROCEDURE_TOTAL

  Procedure List (Percent):
    PM       PMW      PMMW|       RM       RMW      RMMW|      ALL        AW        AMW| Name
 100.0      91.8       3.3|    100.0      53.9       2.9|    100.0      52.3        2.7| est_
( 35.7)   ( 32.7)   (  1.2)   ( 77.3)   ( 41.7)   (  2.3)   ( 35.5)   ( 18.6)    (  1.0)

  Loop & Array Expression List:
      PM         RM        ALL|          Percent| V_Mark| kind  |    VL| Line
      61        324        363|     100.0( 35.7)|   S   | DO    |     -|  00000083-00000095
       0        121        121|       0.0(  0.0)|   S   | DO    |     -|  00000086-00000092
```

● C/VP(情報収集＋解析)

C プログラムは vcc コマンドにより翻訳された実行ファイルに対して解析を行います．cc コマンドで作成された実行ファイルは使用できません．

```
#                                       <---csh で記述
setenv FJSAMP file:sampler.data         <--- 環境変数の設定
a.out                                   <---Sampler 情報ファイル sampler.data の作成
fjsamp a.out                            <---Sampler による解析
```

○ C プログラムの解析例

```
Status                     : Serial
Number of Processors       : 1

Type                       : cpu
Interval (msec)            : 10

Synthesis Information
    Count|   Percent|    VL| Name
   142623|      99.9|     -|  laplace
      125|       0.1|     -|  strline
        2|       0.0|     -|  main

   142750|          |     -|  TOTAL


Function Information(laplace)-----------------------------------
 Function List:
    Count|          Percent|    VL| Name
   142623|     100.0( 99.9)|     -|  laplace

 Loop List:
    Count|          Percent| V_Mark| kind  |    VL| Line
   142304|      99.8( 99.7)|   S   | for   |     -|  00000035-00000043
      317|       0.2(  0.2)|       | for   |     -|  00000033-00000044


Function Information(strline)-----------------------------------
 Function List:
    Count|          Percent|    VL| Name
      125|     100.0(  0.1)|     -|  strline
```

7.1.6 注意事項

- DO ループを 1 つの端末文で共有している場合[66] 最も内側のループに情報が集中することがあります．
- 1 行に複数の文を書くと，情報が正しく収集されない場合があります．
- include 文に実行文がある場合，ループ，配列に関する情報が正しくないことがあります．

[66] Fortran 90 の廃止予定事項です．新しく Fortran を勉強する方は使わないようにしてください．

7.2 Counter

Counter は，単一 PE で動作する Fortran プログラムに対し，手続き，ループ，文の実行回数，ループの回転数，IF 文の正しかった割合などの詳細な実行状況をプログラムリストと共に表示するツールです．

Counter を使用するためには翻訳時オプション `-Wc` の指定が必要です．ただし，`-Wc` オプションの指定により Counter 用のオブジェクトコードが挿入されるため，通常より実行性能が 劣化 します[67]．

●バッチリクエストの記述例

Fortran プログラム “`example.f90`” に対し，翻訳から Counter による解析まで行なうバッチリクエストを記述すると，次の例ようになります．

```
#                                    <--- csh で記述
cd EXAMPLE                           <--- ディレクトリの移動
frt -Wc example.f90                  <--- -Wc オプションを付け翻訳
setenv FJCNT file:counter.data       <--- 環境変数の設定
a.out                                <--- 実行情報の採取
fjsamp example.ainf                  <--- Counter による解析
```

`frt` コマンドに `-Wc` オプションを付加した翻訳の結果，実行ファイル `a.out` と翻訳情報ファイル `example.ainf` が生成されます[68]．次に `csh` の `setenv` サブコマンドによる環境変数を設定します．環境変数名は “`FJCNT`” です．“`file:`” に続くファイルに実行情報が出力されます．ファイル名は任意です．ここでは “`counter.data`” という名前にしています． Counter による解析は Sampler と同じ `fjsamp` コマンドです．翻訳情報ファイル (例では `example.ainf`) を引数に指定します．

`fjsamp` コマンドの機能は kyu-vpp の `man fjsamp` でも検索できます．

[67] この点が Sampler との大きな違いです． Sampler は通常の実行ファイルに対し解析ができるため，実行性能はほとんど変わりません．ただし， Counter の方がより詳細なチューニング情報を収集しますので，状況に応じて使い分けることをお勧めします．

[68] 拡張子 “`ainf`” が自動的につきます．翻訳を対話的に行なうことも可能です．

●解析結果の出力例

```
Status                     : Serial
Number of Processors       : 1

Synthesis Information
   Exec-cnt| Loop-leng| Name
      24096|       362|  LAPLACE_
          1|       147|  STRLINE_
          1|      1184|  MAIN__
          -|         0|  VAL_

           |       361|  TOTAL

Program Unit Information(LAPLACE_)++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
 Procedure List:
   Exec-cnt| Loop-leng| Name
      48192|       241|  laplace.calc_
      24096|     96641|  laplace_

           |       362|  PROCEDURE_TOTAL

 Loop & Array Expression List:
   Loop-exe| Loop-leng| V_Mark| kind  | Line
   19228608|       241|   V   | DO    |  00000060 - 00000065
      24096|     96641|   V   | ARRAY |  00000053 - 00000053
      48192|       399|   V   | DO    |  00000059 - 00000066

 Vectorize Statement List:

  Line         Exec-cnt  true  v o a
  00000046                          !
  00000047        24096                    SUBROUTINE LAPLACE(DELTA)
  00000048                                 USE VAL
  00000049                                 REAL*8 DELTA
  00000050        24096                    CALL CALC(P1, P2)
  00000051        24096                    DELTA = 0.0
  00000052        24096                    CALL CALC(P2, P1)
  00000053        24096        v   v       D = ABS(P2 - P1)
  00000054        24096                    DELTA = MAXVAL(D)
  00000055                          !
  00000056                                 CONTAINS
  00000057        48192                      SUBROUTINE CALC(PIN, POUT)
  00000058                                   REAL*8 PIN(IXMESH,IYMESH), POUT(IXMESH,IYMESH)
  00000059        48192        v             DO J=2, IYMESH-1
  00000060     19228608        v               DO  I=1, IXMESH
  00000061     4.63E+09  83.1  v                 IF (MBOUND(I, J)) THEN
  00000062     3.85E+09        v                  POUT(I,J)=(PIN(I+1,J)+PIN(I-1,J)+PIN(I,J+1)&
  00000063                                        +PIN(I,J-1))*0.25
  00000064     4.63E+09        v                 ENDIF
  00000065     4.63E+09        v               END DO
  00000066     19228608        v             END DO
  00000067        48192                      END SUBROUTINE CALC
  00000068        24096                    END

Message List:

 vectorization messages:
 Internal subprogram name(laplace.calc_)
    jpc1101i-i  ‘‘sampler-spe.f90’’, line 61 - 65: Vectorized by DO variables J and I.
    jpc1002i-i  ‘‘sampler-spe.f90’’, line 62 - 62: Vectorized with masked operation method.
                                  :
```

Counter の解析結果は標準出力に返却されます．もしバッチリクエストを sh で記述している (即ち先頭の # をつけない) 場合，環境変数の設定は

```
FJCNT=file:counter.data
export FJCNT
```

となります．

7.3 GETTOD サブルーチン

5.7.2節で説明したように， VPP700/56 ではCLOCKV サブルーチンの引数が省略不可になりました． CLOCKV サブルーチンは Fortran 90/VPP でも利用できますが， VU 時間は各 PE ごとに独立に測定された時間となります．

Fortran 90/VPP のSPREAD DO 文などの実行時間計測には GETTOD サブルーチンを用いるとオーバーヘッドが軽くすみます．倍精度の引数にはある時点からの経過時間がミリ秒単位で返却されます．

```
      PROGRAM GETTOD_TEST
      REAL(KIND=8) START,FINISH,LA(10000)  ! START,FINISH は重複ローカル変数
!XOCL PROCESSOR P(4)
!XOCL INDEX PARTITION Q=(P,INDEX=1:10000)
!XOCL LOCAL LA(/Q)
!XOCL PARALLEL REGION
!XOCL BARRIER
      CALL GETTOD(START)                   !----+
!XOCL SPREAD NOBARRIER DO/Q                !    |
      DO I=1,10000                         !    |   SPREAD DO ループ
       LA(I)=1.0                           !    |-- 開始と終了の時刻を
      END DO                               !    |   測定
!XOCL END SPREAD DO                        !    |
      CALL GETTOD(FINISH)                  !----+
      WRITE(6,*) (FINISH-START)            ! バリア同期をとるためほぼ同じ値
      END PROGRAM GETTOD
```

7.4 timex コマンド

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．使用方法はいたって簡単で，frt コマンド，vcc コマンドや実行ファイル名の前に “timex” と書くだけです．

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

timex の情報は標準エラー出力に書き出されます．

```
            (翻訳結果)
real            9.20
user            6.94
sys             0.38
```

翻訳はスカラーユニットで実行されます．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 の合計が CPU 時間です．

```
            (実行結果)
real            2.07
user            1.06
sys             0.02
vu-user         1.00
vu-sys          0.01
```

a.out の実行時間の測定例です．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) を表します．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．

また，並列実行の場合は複数 PE の 合計 の CPU 時間が出力されます．

7.5 浮動小数点演算数の採取

PE の動作に関する情報を採取する機能に PEPA(PE Performance Analyzer) があります． PEPA では，PE の総平均性能や演算回数などが採取できます． PEPA の使用は， Fortran では環境変数の設定またはサブルーチンによって， C では環境変数または関数によって行います．

7.5.1 環境変数の設定例

環境変数は FJPEPA，変数は ON です．プログラム全体の動作に関する情報を採取する場合は，実行時に環境変数を設定します．以下はスクリプトの記述例です．

```
#
setenv FJPEPA ON                        <---PEPA 環境変数の設定
a.out                                   <--- 実行
```

採取された情報は，プログラムの実行が終了した段階で標準エラー出力に書き出されます． PEPA のサブルーチン，関数を用いると，プログラムの特定の部分の情報も採取できます．使用方法は [11] を御覧ください．

表 7.2: PEPA の情報採取項目

番号	採取項目	測定内容	単位
1	VU ビジー率	計測時間のうち，ベクトルパイプラインが動作していた時間の割合．複数のベクトルパイプラインの重複時間は含まない	%
2	PE 総平均性能	スカラー命令とベクトル命令の浮動小数点演算数 (3 の値) を計測時間 (5 の値) で割った値	回 $\times 10^6$/sec.
3	演算回数 (VU+SU)	スカラー命令とベクトル命令の浮動小数点演算数	回 $\times 10^6$
4	演算回数 (VU)	ベクトル命令の浮動小数点演算数	回 $\times 10^6$
5	計測時間	採取開始から採取終了までの経過時間	sec.

●出力例

```
PE performance analyzer
[ VU performance ]
PE No   1
-------------------------------------------------------------------------------
 VU busy ratio                                    [%]          6.105639e+01
 PE average performance                                        1.057640e+03
 Operation count of Scalar UNIT and Vector UNIT   [times*1e6]  2.628549e+02
 Operation count of Vector UNIT                   [times*1e6]  2.623517e+02
 Measurement time                                 [ sec ]      2.485297e-01
-------------------------------------------------------------------------------
```

7.5.2 注意事項

- 演算回数は，プログラムから生成されたオブジェクトの演算だけでなく，組み込み関数や SSL II/VP などのライブラリの演算回数も含めてカウントされます.

- ベクトル命令の演算回数は実際はパイプラインの動作時間より算出しているため，実際の演算回数と必ずしも完全には一致しません.

- 計測時間は経過時間で測定するため，同じプログラムを実行しても実行ごとに値が異なる可能性があります.

7.6 データ転送に関する情報収集

複数の PE を用いた並列プログラムのデータ転送に関する情報を採取する機能に MPA(Mover Performance Analyzer)があります． MPA では，データの送信時間，転送量，転送回数などの情報が採取できます． MPA は， Fortran 90/VPP により作成された並列実行ファイルにのみ使用できます.

7.6.1 環境変数の設定例

環境変数は FJMPA，変数は ON です．プログラム全体の動作に関する情報を採取する場合は，実行時に環境変数を設定します．以下はスクリプトの記述例です.

```
#
setenv FJMPA ON                          <---MPA 環境変数の設定
a.out                                    <--- 実行
```

採取された情報は，プログラムの実行が終了した段階で標準エラー出力に書き出されます． MPA のサブルーチンを用いると，プログラムの特定の部分の情報も採取できます．使用方法は [11] を御覧下さい.

また， 1PE の場合のみ，対話的に PEPA による情報収集ができます.

```
kyu-vpp% setenv FJPEPA ON  ↵                    <--- 環境変数の設定
kyu-vpp% a.out  ↵                               <--- 実行
    :
  (実行結果)
    :
PE performance analyzer
[ VU performance ]
PE No   1
-------------------------------------------------------------------------------
 VU busy ratio                                        [%]          5.566235e+01
 PE average performance                                            1.299649e+01
 Operation count of Scalar UNIT and Vector UNIT       [times*1e6]  1.668771e-02
 Operation count of Vector UNIT                       [times*1e6]  1.566971e-02
 Measurement time                                     [ sec ]      1.284017e-03
-------------------------------------------------------------------------------
```

表 7.3: MPA の情報採取項目

番号	採取項目	測定内容	単位
1	送信時間	パケット送信の開始から完了までの時間の合計	sec.
2	送信待ち時間	送信時間中で，パケット送信先 PE がビジーで自 PE が送信待ち状態になった時間の合計	sec.
3	送信被妨害時間	多数のパケットの送受信によりオーバーフローを起こすなどして，送信が妨害された時間の合計	sec.
4	自 PE 宛送信時間	送信時間中で，自 PE へのパケット送信の開始から完了までの時間の合計	sec.
5	受信時間	パケット受信の開始から完了までの時間の合計	sec.
6	受信パケットなし時間	パケット受信中でなく，かつ受信被妨害状態でない時間の合計	sec.
7	他 PE への送信量	他 PE への送信データ量の合計	Kbyte.
8	他 PE からの受信量	他 PE からの受信データ量の合計	Kbyte.
9	送信パケット数	送信パケット数の合計 (自 PE 宛送信を含む)	回
10	受信パケット数	受信パケット数の合計	回
11	MPA 稼働時間	採取開始から終了までの経過時間	sec.

●出力例

```
PE No   4
-------------------------------------------------------------------------------
 time of packet sending                      [ sec  ]          6.009052e+01
 time of packet sending NET busy             [ sec  ]          3.952059e-01
 time of packet sending obstructed           [ sec  ]          6.988800e-05
 time of packet sending to self-PE           [ sec  ]          0.000000e+00
 time of packet receiving                    [ sec  ]          6.001732e+01
 time of no receiving packet                 [ sec  ]          5.269496e+03
 length of packets sent to other PEs         [kbyte ]          3.307451e+07
 length of packets received from other PEs   [kbyte ]          3.307145e+07
 number of packets sent                      [packet]          3.913500e+05
 number of packets received                  [packet]          3.913500e+05
 time of MPA active status                   [ sec  ]          5.329561e+03
-------------------------------------------------------------------------------
```

8　アプリケーションライブラリの利用

この章では，VPP700/56 で動作するプログラム言語以外のアプリケーションライブラリの概要と参考文献を紹介します．

8.1　α-FLOW

8.1.1　概要

最新の解析技術を導入した汎用 3 次元流体解析システムです．格子生成，AI 支援機能など充実したプリ／ポスト機能を有し，ベクトル計算機向けの最適化技術を採用しています．

8.1.2　利用方法

従来のセンター 2 階ワークステーション室の gws-o1，gws-o2 に加え，ユーザインタフェースワークステーション vhsun でも動作します[69]．また，各地区に新たに設置されるワークステーション vcsun(筑紫)，vbsun(病院)，vrsun(六本松) でも動作します．

α-FLOW は必要なデータをワークステーション側で作成し，大規模な計算を VPP700/56 に依頼し，計算結果の可視化を再びワークステーションで担当する仕組みになっています．

α-FLOW を利用するためには，プリ／ポスト処理を担当するワークステーションへの登録が必要です[70]．詳しい動作環境の設定方法，使用方法は，各ワークステーション横に設置した手引，マニュアル ([21]–[27]) を参照してください．

8.2　MASPHYC

8.2.1　概要

MASPHYC は，材料の物性・構造を原子・分子レベルのミクロな情報から分子動力学の手法を用いて予測するシステムです．材料の性質を決定する原子・分子間相互作用ポテンシャルをライブラリ化することにより，有機化合物から無機化合物まで幅広い材料に適用可能です．

8.2.2　利用方法

従来のセンター 2 階ワークステーション室の gws-o1，gws-o2 に加え，ユーザインタフェースワークステーション vhsun，vcsun，vbsun，vrsun でも動作します．詳しい動作環境の設定方法，使用方法は，各ワークステーション横に設置した手引，マニュアル ([28], [29]) を参照してください．

---

[69] リモートでの処理は未公開です．

[70] 登録は kyu-cc から `touroku` *workstation-name* です．

8.3 LS-DYNA

8.3.1 概要

非線形動的構造解析ソフトウェア LS-DYNA は，衝突安全解析やプレス成形解析の分野でトップレベルの機能を持つ解析プログラムです．時間積分に陽解法を使用し，大変形・弾塑性・動的接触を含む数万要素を短時間で計算できます．さらに構造解析だけでなく，熱や流体との連成解析も標準で装備しており，通常の非線形ソフトウェアでは不可能な数万素以上のモデルや収束性の悪い問題にも適用可能です．

8.3.2 利用方法

ユーザインタフェースワークステーション vhsun, vcsun, vbsun, vrsun で動作します．プリ／ポスト処理は“eta/FEMB”というソフトが担当します．詳しい動作環境の設定方法，使用方法は，各ワークステーション横に設置した操作手順書，マニュアル ([30], [31]) を参照してください．また，以下のマニュアルはワークステーション上のオンラインマニュアルとしてサポートされています．

- LS-DYNA3D User's manual
- LS-DYNA3D Theory manual
- LS-DYNA3D Examples manual

8.4 AVS

8.4.1 概要

AVS(Application Visualization System) は，豊富なツールを用いて多角的なデータ解析ができるように設計されたデータおよびアプリケーション可視化システムです． AVS は，新しく導入される可視化用サーバ，ユーザインタフェースワークステーションで動作します． VPP700/56 には AVS のリモートモジュールが提供されます．

また， VPP700/56 の Fortran, C とユーザインタフェースワークステーションの AVS を連携しながら，リアルタイムで可視化を実現するツール VisLink もサポートされます．利用方法は [32], [43], [44], [45] を参照してください．

8.5 MARC/MENTAT II

8.5.1 概要

MARC プログラムは， NASTRAN, ADINA と同様，今日世界的に広く利用されている汎用有限要素法解析プログラムです．構造解析をはじめとして，熱伝導解析，音響解析，静電場解析などの機能を持ち，豊富なライブラリ群から必要なものを選択して自由度の高い解析処理を行なうことが出来ます．特に非線形解析が精度良く行なえるのが特徴です． VPP700/56 のバージョンは MARC K6.2 です．なお， MARC K6.2 は 1PE での動作です．

MENTAT II は， MARC プログラムのための会話型プリポストプロセッサです．グラフィックス・ディスプレイ上で対話的な入力データの作成・編集および解析結果の表示が可能です． MENTAT II はユーザインタフェースワークステーション vhsun 上で利用できます．バージョンは MENTAT II 2.3 です．

従来 VP2600/10 の MSP で公開してきた MARC K5.2 は， VP2600/10 の撤去にともない 1997 年 2 月末日で運用を停止しています．ただし， MARC K5.2 のプログラムおよび入出力データは汎用計算機 M-1800/20U の MSP システムにそのまま保存されています．また，センターのワークステーション qviss, medics で公開中の旧バージョンの MENTAT は 1997 年 3 月末日で運用を停止し，新バージョンに一本化しています．

8.5.2 MARC の利用方法

◇利用環境

MARC の利用は kyu-vpp または汎用計算機 M-1800/20U の UXP システム (ホスト名 kyu-cc, IP アドレス 133.5.9.1) からバッチリクエスト (NQS) により行ないます． MSP からの利用はできません.

◇ MSP からのデータ転送

従来の MSP の資産を引続き利用する場合は，汎用計算機の MSP システム (ホスト名 kyu-msp, IP アドレス 133.5.9.2) から kyu-vpp, kyu-cc に ftp でデータを転送してください.

```
kyu-vpp% ftp kyu-msp  ↵                          <--- kyu-vpp から ftp
Connected to kyu-msp.
220 Service ready for new user
Name (kyu-msp:a79999a): a79999a  ↵               <--- ID
331 User name okay, need password
Password:          ↵                             <--- MSP のパスワード
230 User logged in, proceed
ftp> get marc.data marc.dat  ↵                   <--- MARC.DATA を marc.dat として転送
200 Command okay
150 File status okay;about to open data connection
226 File transfer complete
local: marc.data remote: marc.dat
1709536 bytes received in 2.9 seconds (5.7e+02 Kbytes/s)
ftp> quit  ↵                                     <--- ftp の終了
221 Service closing CONTROL connection
```

MSP(kyu-msp) から UXP(kyu-vpp, kyu-cc) へのデータ転送では以下の点に注意してください.

- MARC の入力データは行番号を PFD の unn サブコマンドで取り除いて転送します.
- 行番号付きのデータを転送した場合は，転送したファイルの 73–80 桁に行番号がついています．行番号は以下の手順で取り除くことができます.

```
kyu-vpp% cut -c1-72 marc.dat > marc1.dat ↵
```

- UXP 側の入力データファイルのサフィックスは必ず “.dat” とします.

◇ MARC の動作するディレクトリ (kyu-cc の場合)

MARC の入力データファイルは， kyu-cc のホームディレクトリ下の VPP ディレクトリに作成します (Fortran, C プログラムを実行する要領と同じです)． kyu-vpp の場合は特にディレクトリに注意する必要はありません.

例として， MARC 用のディレクトリ MARC を VPP 下に作成します.

```
kyu-cc% cd ~/VPP ↵          <--- VPP700/56 の作業用ディレクトリに移動
kyu-cc% mkdir MARC ↵        <--- MARC 用のディレクトリの作成 (あくまでも例です)
kyu-cc% cd MARC ↵           <--- MARC 用のディレクトリ (例) への移動
```

◇サンプルプログラム

MARC マニュアル《E 編》の入力データ，ユーザーサブルーチン集を kyu-vpp では

/usr/local/MARC/marck62/demo

に， kyu-cc では

/usr/local/doc/demo/marck62

に公開しています．各自コピーして参照することができます.

kyu-vpp のe2x1.dat,e2x14.dat, u2x14.f のコピー例です.

```
kyu-vpp% ls /usr/local/MARC/marck62/demo | more ↵          <--- 一覧表示
kyu-vpp% cp /usr/local/MARC/marck62/demo/e2x1.dat . ↵      <--- サンプルのコピー例
kyu-vpp% cp /usr/local/MARC/marck62/demo/e2x14.dat . ↵
kyu-vpp% cp /usr/local/MARC/marck62/demo/u2x14.f . ↵
```

◇ MARC の動作するキュー

VPP700/56 での MARC の実行は， Fortran, C と同様「バッチリクエスト」と呼ばれるシェルスクリプトに処理手順を記述し，qsub コマンドでジョブを投入します．投入できるキューは表 8.1 の通りです.

表 8.1: MARC の動作するキュー

キュー名	CPU 時間	最大記憶域	省略値	処理形態
s	60 分	1.7GB	0.5GB	1PE
p1	1200 分	1.7GB	0.5GB	1PE

バッチキューの指定を省略すると p1 キューに投入されます.

◇ marck62 コマンド

MARC の処理は marck62 コマンドで行ないます．指定可能なオプションは以下の通りです．なお，これ以外の MARC オプションを指定すると，永久待ちが発生する可能性がありますので注意してください．

-jid␣*jobname*	ジョブファイル名を与えます．通常は “*jobname*.dat” という名前の入力データファイルとなります．
-prog␣*progname*	ユーザーサブルーチン付のジョブを実行したときにセーブした実行ファイル “*progname*.marc” を実行します．
-user␣*username*	ユーザーサブルーチン “*username*.f” を使って “*username*.marc” という名前の新しい実行ファイルを作成し，実行します．
-save␣yes	ユーザーサブルーチンを組み込んで作成した実行ファイル “*username*.marc” を保存することを指示します．
-rid␣*restart-name*	リスタートファイルを出力するように指定して前もって実行したジョブのジョブファイル名を与えます．
-pid␣*postname*	温度情報を持ったポストファイルを作成するために前もって実行したジョブのジョブファイル名を与えます．
-pid␣*substructure*	サブストラクチャを使用するジョブで使用します．サブストラクチャファイル名は “*substructure*.t31” となります．

- MARC の入力データは常に “*jobname*.dat” というファイル名であることが必要です．つまり拡張子は “.dat” となります．
- ユーザーサブルーチンを使って作成した実行ファイルはかなりの容量となります．

◇バッチリクエストの記述例

- ジョブ e2x1 を実行します．入力データは e2x1.dat です．

```
#                          <--- csh を起動
cd MARC                    <--- ディレクトリの移動
marck62 -jid e2x1          <--- MARC の実行
```

- ジョブ e2x14 をユーザーサブルーチン u2x14 を使って実行します．ユーザーサブルーチンファイルは u2x14.f，入力データは e2x14.dat です．また，新規に作成された実行ファイル u2x14.marc をジョブの終了後も保存します．

```
#
cd MARC
marck62 -jid e2x14 -user u2x14 -save yes
```

- 先の例題で作成した実行ファイル u2x14.marc を用いて再びジョブ e2x14 を実行します．

```
#
cd MARC
marck62 -jid e2x14 -prog u2x14
```

- リスタートオプションを記述したジョブ e3x2a を実行します.

```
#
cd MARC
marck62 -jid e3x2a
```

- ジョブ e3x2a の結果を用いてリスタートジョブ e3x2b を実行します.

```
#
cd MARC
marck62 -jid e3x2b -rid e3x2a
```

◇バッチリクエストの投入

先の例の手順で作成したバッチリクエストファイル名を marc.sh とします．バッチリクエストの投入は qsub コマンドです.

```
kyu-vpp% qsub marc.sh ↵                          <--- p1 キューに投入
Request 11261.kyu-vpp submitted to queue: p1.
```

s キューに投入する場合は，-q s オプションをつけます.

```
kyu-vpp% qsub -q s marc.sh ↵                      <--- s キューに投入
Request 11262.kyu-vpp submitted to queue: s.
```

kyu-cc の場合も同様です．バッチリクエストの投入，状態表示，キャンセルの手順の詳細は 3 章を参照してください.

◇ファイルとユニット番号の対応

MARC の使用するファイルとユニット番号は表 8.2 の通りです.

表 8.2: ファイルとユニット番号

サフィックス	番号	説明	形式
.log	0	エラーメッセージなどの出力用	テキスト
.t01	1	フォーマット形式のデータ	通常はメッシュデータ
.t02	2	Out-Of-Core ソルバー用スクラッチ	ランダムアクセス，バイナリ
.t03	3	ELSTO	順次アクセス，バイナリ
.t04	4	neutral-plot	順次アクセス，バイナリ
.dat	5	入力データ	フォーマット形式 (Fortran)
.out	6	MARC の出力	フォーマット形式 (Fortran)
.t08	8	新たに作成されるリスタートファイル	順次アクセス，バイナリ
.t08	9	前のジョブで作成されたリスタートファイル	順次アクセス，バイナリ
.t11	11	Out-Of-Core ソルバー用スクラッチ	順次アクセス，バイナリ
.t12	12	Out-Of-Core ソルバー用スクラッチ	順次アクセス，バイナリ
.t13	13	Out-Of-Core ソルバー用スクラッチ	順次アクセス，バイナリ
.t14	14	Out-Of-Core ソルバー用スクラッチ	ランダムアクセス，バイナリ
.t15	15	Out-Of-Core ソルバー用スクラッチ	順次アクセス，バイナリ
.t16	16	新たに作成されるポストファイル (Fortran)	順次アクセス，バイナリ
.t17	17	前のジョブで作成されたポストファイル (Fortran)	順次アクセス，バイナリ
.t18	18	フォーマット形式のデータ，オプティマイズテーブル	フォーマット形式 (Fortran)
.t19	19	新たに作成されるポストファイル	フォーマット形式 (Fortran)
.t19	20	前のジョブで作成されたポストファイル	フォーマット形式 (Fortran)
.t22	22	サブスペースのためのスクラッチ	順次アクセス，バイナリ
.t23	23	流体ソリッド用スクラッチ	順次アクセス，バイナリ
.t19	24	温度分布データ	フォーマット形式 (Fortran)
.t16	25	温度分布データ (Fortran)	順次アクセス，バイナリ
.t31	31	サブストラクチャ用マスターファイル	ランダムアクセス，バイナリ
.t32	32	セカント法	順次アクセス，バイナリ
.t34	34	ニュートラルプロット	フォーマット形式 (Fortran)
.t35	35	サブストラクチャ	順次アクセス，バイナリ
.t36	36	サブストラクチャ	順次アクセス，バイナリ

8.5.3 MENTAT II の利用方法

◇利用環境

MENTAT II 2.3 は，センター 2 階のユーザーインターフェースワークステーション vhsun で動作します. vhsun への登録方法はセンターニュース No.553 を御覧ください. なお， MENTAT II の利用は当面コンソールからの利用のみとします.

起動コマンドは mentat(/usr/local/bin/mentat) です. 詳しい利用方法は，備えつけのマニュアルを御覧ください.

◇ VPP700/56 とのディスク共有

vhsun のホームディレクトリ下のディレクトリ “VPP” は， kyu-cc と同様， VPP700/56 の利用者のホームディレクトリにマウントされています. 従って，このディレクトリに移動することで ftp でデータの転送をすることなく MARC のプリポスト処理が可能です.

```
vhsun% cd VPP/MARC  ↵              <--- ディレクトリの移動 (MARC はあくまでも例です)
vhsun% ls  ↵                       <--- MARC の作業用ファイルが参照可能
e2x1.dat        e2x1.t19        marc.sh.e11030
e2x1.out        marc.sh         marc.sh.o11030
vhsun% mentat  ↵                   <--- MENTAT II の起動
```

◇ PostScript ファイルへの保存

MENTAT II のグラフィックスは PostScript ファイルに保存することができます．メニューの《UTILS》の《Color file》または《Gray file》を選択してください．

8.5.4 マニュアル

九州大学大型計算機センター 4 階図書室で閲覧可能です． MENTAT II 2.3 の日本語マニュアルはセンター 2 階の vhsun 横に設置しています．

購入する場合は，日本マーク株式会社 (03-3345-0181) に問い合わせください．

8.6 Gaussian94

8.6.1 概要

Gaussian は Carnegie-Mellon 大学の Pople を中心として開発された分子軌道計算プログラムパッケージで，プログラム構造の明解さ，計算の安定性から急速に普及し，広く計算化学の分野で利用されている世界的に著名なアプリケーションソフトウェアです． kyu-vpp, kyu-cc で動作します．

参考文献 [33], [34], [35] はセンター 4 階の図書室で閲覧できます．

8.6.2 Gaussian94 の環境設定

◇ kyu-vpp の設定

kyu-vpp で Gaussian94 を利用するには，ホームディレクトリ上でエディタ (vi, mule) により以下の内容を “.cshrc” というファイル名で作成 (または追加) します．

```
setenv g94root /usr/local/gaussian94
setenv GAUSS_SCRDIR /tmp
source $g94root/g94/bsd/g94.login
```

設定時のみ，.cshrc の編集後 source ~/.cshrc と入力してください．以降の login 時には必要ありません．

.cshrc に設定しない場合は， login するごとにコマンドとして環境設定を行ってください．

◇ kyu-cc の設定

M-1800/20U の UXP/M システム (ホスト名 kyu-cc) で Gaussian94 を利用するためには，同様にホームディレクトリ下の .cshrc に以下を追加します．

```
setenv g94root /usr/local/gaussian94
setenv GAUSS_SCRDIR /tmp
source $g94root/g94/bsd/g94.login
```

8.6.3 対話型での利用

◇ g94 コマンド

起動コマンドは g94(/usr/local/gaussian94/g94/g94) です． Gaussian プログラムを test.com とすると，

```
kyu-vpp% g94 test.com ↵
```

で test.log に結果が出力されます．

なお，“&” を付けることでバックグラウンドジョブ (バッチ処理とは異なります) として走らせることも可能ですが，その場合はあらかじめ stty -tostop を入力し，非同期にバックグラウンドからのメッセージを表示できるように設定してください．もし設定をしない場合は，エラーや CPU 時間オーバーによる打ち切りが発生した場合に端末にメッセージを出力しようとし，プロセスが残ってしまう可能性があります．

```
kyu-vpp% stty -tostop  ↵              <--- 非同期にメッセージを表示
kyu-vpp% g94 test.com &  ↵            <--- バックグラウンド処理をする
```

◇使用ファイルの制限

対話的処理でのファイルの使用上限は kyu-vpp が 2GB, kyu-cc が省略値で 30MB です． kyu-cc で制限を解除したい場合は，`unlimit filesize` と入力します．

また，経過時間，使用 CPU 時間を `timex` コマンド計測できます．

```
kyu-cc% unlimit filesize  ↵           <--- ファイル使用の制限を解除 (kyu-cc)
kyu-cc% timex g94 test.com  ↵         <--- 処理時間を計測

real           21.15                  <--- 経過時間
user           13.71                  <---CPU 時間 (利用者)
sys             1.58                  <---CPU 時間 (システム)
vu-user         4.30                  <--- ベクトルユニット使用時間 (利用者)
vu-sys          0.00                  <--- ベクトルユニット使用時間 (システム)
vuw-user        0.23
vuw-sys         0.00
```

◇オンラインヘルプ

`ghelp(/usr/local/gaussian94/g94/ghelp)` コマンドで簡単な Gaussian の機能を検索することが可能です．`ghelp` とだけ入力すると使用方法が出力されます．

◇サンプルプログラム

Gaussian94 のサンプルプログラムが `/usr/local/gaussian94/g94/tests/com` 下にあります．各自でコピーし利用してください．

8.6.4 バッチ処理

◇利用環境

大規模なジョブは 3 章で説明したバッチ処理になります．バッチ処理は VPP700/56, M-1800/20U 双方で利用できます．

◇ Gaussian94 の動作するディレクトリ

kyu-cc から VPP700/56 のバッチキューに投入する場合についてのみ注意が必要です． kyu-vpp から VPP700/56 のバッチキューへの投入， kyu-cc から M-1800/20U へのバッチキューへの投入の場合は特に注意事項はありません．ただし， kyu-vpp から M-1800/20U へのバッチ処理は依頼できません．

kyu-cc から VPP700/56 のバッチキューに投入する場合， Gaussian94 のプログラムは， kyu-cc のホームディレクトリ下の `VPP` ディレクトリに作成します (Fortran, C プログラムを実行する要領と同じです)．

例として， Gaussian94 用のディレクトリ `Gaussian` を `VPP` 下に作成します．

```
kyu-cc% cd ~/VPP ↵              <--- VPP700/56 の作業用ディレクトリに移動
kyu-cc% mkdir Gaussian ↵        <--- Gaussian 用のディレクトリの作成 (あくまでも例です)
kyu-cc% cd Gaussian ↵           <--- Gaussian 用のディレクトリ (例) への移動
```

◇ Gaussian94 の動作するキュー

Gaussian94 の実行は， Fortran, C と同様「バッチリクエスト」と呼ばれるシェルスクリプトに処理手順を記述し，qsub コマンドでジョブを投入します．subg94 コマンドは NQS の手順を簡略化したものです．

投入できるキューは表 8.3 の通りです．

表 8.3: Gaussian94 の投入できるキュー

キュー名	CPU 時間	最大記憶域	省略値	計算機
s	60 分	1.7GB	0.5GB	VPP700/56
p1	1200 分	1.7GB	0.5GB	VPP700/56
v	1200 分	0.5GB	0.5GB	M-1800/20U

◇汎用計算機のバッチキュー

1998 年 1 月より新設された汎用計算機のバッチキュー “v” は， 0.5GB の記憶領域が確保できるベクトル処理可能なキューです．また，あわせて 1998 年 4 月より汎用計算機のバッチ処理の演算負担金を 大幅に値下げ[71] しています．

Gaussian94 は処理の内容によっては VPP700/56 と M-1800/20U の実行速度がそれほど変わらないもの，あるいはむしろ M-1800/20U の方が速いものもあります．

◇ subg94 コマンド

subg94(/usr/local/gaussian94/g94/bsd/subg94) により，簡単にバッチジョブを投入できます．なお，プログラムファイル名は必ずサフィックスを “.com” としてください．

kyu-vpp のプログラム test.com を s キューに投入する場合は以下のように行います．

```
kyu-vpp% subg94 s test ↵                <---s キューへの投入 (kyu-vpp)
Request 7308.kyu-vpp submitted to queue: s.
kyu-vpp%
```

処理結果は汎用機での対話型処理と同じくtest.log へ，また，バッチ処理のメッセージは test.batch-log へそれぞれ出力されます．

kyu-cc のプログラム test.com を v キューに投入する場合は以下のように行います．

```
kyu-cc% subg94 v test ↵                 <---v キューへの投入 (kyu-cc)
Request 7309.kyu-cc submitted to queue: v.
kyu-cc%
```

[71] 従来の 15 分を超えての 1 円 / 秒を 0.1 円 / 秒に値下げしました．なお 5 分未満は 2 円 / 秒， 5 分以上 15 分未満は 1 円 / 秒です．

◇ qsub コマンド

一般のバッチ処理と同様に，スクリプトファイルを記述し，qsub(/usr/bin/qsub) コマンドでバッチ処理を依頼することも可能です.

以下はスクリプトファイルの記述例です.

```
#
cd Gaussian
g94 test.com
```

1 行目の # は，バッチリクエストが csh であることの宣言です． 2 行目は，test.com のあるディレクトリ (例として Gaussian) に移動します．移動しない場合は，ホームディレクトリにあるとみなされます． 3 行目は g94 による処理のコマンドです.

スクリプトファイルを a.sh とし，qsub コマンドにより p1 キューに投入します.

```
kyu-vpp% qsub a.sh ↵                        <---p1 キューへの投入
Request 7309.kyu-vpp submitted to queue: p1.
kyu-vpp%
```

◇ qstat コマンド

kyu-cc から v キューに投入したバッチジョブの処理状況を知るには qstat @kyu-cc と指定してください.

```
kyu-cc% qstat @kyu-cc | more     <--- バッチ処理のモニター
                  :
v@kyu-cc;  type=BATCH;  [ENABLED, RUNNING];  pri=63
  0 exit;   1 run;   1 queued;   0 wait;   0 hold;   0 arrive;
  0 freezing;   0 frozen;

         REQUEST NAME        REQUEST ID          USER PRI    STATE BAT-ID  PGRP
<1 requests RUNNING>
    2:        test001  36071.kyu-cc           a79999a  31   QUEUED
                  :
```

例では，現在実行待ちであることがわかります.

◇制限値を越えるジョブについて

Gaussian ではジョブが長時間におよぶ場合や，大量のファイルを必要とする場合があります．制限値を越えるジョブの実行を希望される方は,

request@cc.kyushu-u.ac.jp

まで連絡してください．できる限りの対処をします.

参考文献の入手方法など， Gaussian についての情報は info@gaussian.com まで (英語で) 問い合わせ下さい.

8.7 プログラムライブラリ開発分

8.7.1 利用可能なライブラリ

表 8.4 のライブラリが利用可能です.

表 8.4: VPP700/56 で利用可能なライブラリ

プログラム名	機能
CGJQ	Gauss-Jacobi 積分公式の係数
CGLQ	Gauss-Laguerre 積分公式の係数
DIFF1D, DIFF1S	解析関数の数値微分
MINMAX	線形方程式のミニマックス解
PRESNL	一般化されたフレネル積分
CA01	計算修正による関数の極小点発見
DA02	BCS 方程式
DB01–DB03	クレブシュ・ゴルダン, ラカー, 9-J 係数
KNL1	生成座標方式による直交条件模型の積分核
VAR1	微積分方程式または積分方程式変分原理による錯乱境界条件解
TBS1	厳密 3 体理論による量子学的 3 体系束縛状態のエネルギーと波動関数 I [46]
TBS2	厳密 3 体理論による量子学的 3 体系束縛状態のエネルギーと波動関数 II [42]
ww2d	有限差分法を用いた自由表面を持つ気液二相流の 数値シミュレーションプログラム [39]
vpdsm	実対称行列の正定値性判定プログラム [52]
cyc	円分多項式の係数の計算 [47]
AACOUST	建築音響解析ライブラリ

8.7.2 UXP からの利用

サブルーチン，関数型のライブラリは `-lsslq` オプションを付加してリンクします.

```
#
cd EXAMPLE
frt test.f90 -lsslq
a.out
```

なお，`MINMAX` を使用する場合は `-lsslq -lnumpac`，`AACOUST` の場合は `-lsslq -lssl2vp` と指定してください.

8.7.3 MSP からの利用

サブルーチン，関数型のライブラリは Fortran のソースに記述するだけで自動的にライブラリが結合されます.

TBS1 を MSP から利用する場合は，[46] に記述されたカタログドプロシジャ `VPGO` と `EXEC` 文，`DD` 文が異なりますので注意願います.

```
//A79999A1 JOB CLASS=W
//  EXEC PGM=TBS1
//FT06F001 DD SYSOUT=*
//FT05F001 DD *
        (入力データ)
//
```

9　プログラムのデバッグ

この章では，VPP700/56 で Fortran プログラムをデバッグするにあたってのヒントを簡単に説明します．

9.1　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．

●手続き単位に分割する

主プログラムにすべての処理を記述することは以下の点からよくありません．

- 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．

●プログラムの規模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，パラメータを変えて採取した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，他の用途への使い回しも容易になります．

●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．

9.2 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.
            signal identifier = SEGV_MAPERR, address not mapped to object
            execution mode = advanced
jwe_etrc: error occurred during jwercv_i() execution
 error summary (Fortran 90)
 error number  error level  error count
                 :
```

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，強制終了の原因と対処方法を述べます．

9.2.1 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.
            execution mode = advanced, insturction = scalar
  error occurs at MAIN__   line 5 loc 00000338 offset 000000a8
  MAIN__       at loc 00000290 called from o.s.
```

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．

9.2.2 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表現可能な値を超えたことを告げています．

```
 jwe0011i-e A floating overflow exception was detected.
            execution mode = advanced, insturction = scalar
```

正規化などプログラムの再検討が必要です．

9.2.3 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表現可能な値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.
            execution mode = advanced, insturction = scalar
```

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．

```
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 を指定しアンダーフローを確認してみてください．

9.2.4 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 -Dsx を指定して翻訳・実行します．また同時にデバッグモードに切替える -Wv,-ad オプションも指定します．

```
kyu-vpp% frt -Dsx -Wv,-ad test.f90          <--- 添字式の値を検査するオプション
kyu-vpp% 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．

9.2.5 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-vpp% frt -Da -Wv,-ad test.f90           <--- 引数の矛盾を検査するオプション
kyu-vpp% a.out                              <--- 調査
```

9.2.6 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 -Dux を指定して翻訳・実行します．

```
kyu-vpp% frt -Dux -Wv,-ad test.f90          <--- 未定義データの引用を検査するオプション
kyu-vpp% a.out                              <--- 調査
```

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．-Dx オプションは高速なデバッグを促進するオプションです．しかし，やはり実行時間は数倍になります．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．

9.2.7 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-vpp% frt -Ep test.f90          <--- 不変式の先行評価のメッセージを出力
```

不変式の先行評価の最適化を抑止するには，次のように指定します.

```
kyu-vpp% frt -Oe,-P test.f90       <--- 不変式の先行評価を抑止
```

9.3 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.

9.3.1 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[72].

9.3.2 write 文の挿入

write 文，print 文を怪しい箇所に埋め込み，結果を見ます．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作っておくのも良いでしょう.

9.3.3 デバッグオプションの指定

実行時に異常終了しない場合でも，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ください.

```
kyu-vpp% frt -Dasux -Wv,-ad test.f90  ↵    <--- デバッグオプションの指定
kyu-vpp% a.out  ↵
```

例では，デバッグオプション -Da, -Ds, -Du と高速デバッグ機能の -Dx 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.

9.3.4 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.

[72] 人によりますが，完全に煮詰まってしまった時は視点を変えるためにその日はさっさと遊びに行く方が結果的に良かったりします.

```
kyu-vpp% frt -Ee test.f ↵   <--- 演算評価方法の最適化に関するメッセージを出力
fortran90/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ください．

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．

9.3.5 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-vpp% frt -Wv,-an test.f90          <--- 式の評価順序の変更を抑止
kyu-vpp% a.out
```

9.3.6 違うコンパイラで試す

異なる計算機環境で流してみるのもデバッグには有効です．メーカーの違うコンパイラで翻訳するだけでも思いがけないバグを発見することがあります．

9.4 Fortran 90/VPP のデバッグ

1997 年 9 月より， Fortran 90/VPP の機能追加として添字式および部分列式の値を検査するデバッグ機能がサポートされました．

並列プログラムのデバッグを行なうためには，オプション -Wx,-Cf で翻訳して作成した実行ファイルを実行します．以下は翻訳から実行までのバッチリクエストの記述例です．

```
#                                      <---csh で記述
cd EXAMPLE                             <--- ディレクトリの移動
frt -Wx,-Cf -Ps -Wv,-m3 test.f90       <--- デバッグオプションを指定し翻訳
a.out                                  <--- 並列実行
```

例えば，宣言した配列をはみ出したアクセスが起きた場合には，実行時に以下のメッセージが出力されます．

```
 jwe2310i-w The subscript(a) is outside of the specified range. referece value is
 a(130:130:1,641:641:1), specification value is a(1:1024:1,0:0:1)(file=test.f90,
line=38,vpid=6).
 taken to (standard) corrective action, execution continuing.
```

並列デバッグオプションを指定して作成した実行ファイルは，オプションを指定しない場合に比べて大幅に実行時間が増加します．従って，デバッグオプションを指定して作成した実行ファイルと通常の実行ファイルは区別して利用して下さい．また並列デバッグは小さいプログラムの規模で行なうことをお勧めします．

9.5 その他

9.5.1 実行ファイルの判定

特に kyu-cc で作業をしていると，実行ファイルが汎用機用なのか VPP700/56 用なのか，単一 PE 版なのか並列計算版なのかがわからなくなることがあります．そのような場合は file コマンドで調べます．

```
kyu-cc% file a.out  ↵
a.out:          ELF 32 ビット MSB 実行可能 VPP UXP/V Version 1 Vector-EX
kyu-cc% file b.out  ↵
b.out:          ELF 32 ビット MSB 実行可能 VPP UXP/V Version 1 Vector-EX Parallel
kyu-cc% file c.out  ↵
a.out:          ELF 32 ビット MSB 実行可能 UXP/M Version 1 use vector unit
```

9.5.2 実行時の記憶領域の算定

実行ファイルの段階でどれくらいのメモリを使用するのかを size コマンドで見積もることが出来ます．

```
kyu-vpp% size a.out
1238764 + 147744 + 349439296 = 350825804
```

単位はバイトです．例では約 334MB[73]のメモリが必要です．

9.5.3 倍精度の使用

大規模数値計算における丸め誤差の影響を考えれば，倍精度での計算を強くお勧めします．また， 4 倍精度は残念ながらベクトル化の対象外となるため，使用する際は計算時間の増大を覚悟してください．

9.5.4 手続きをオブジェクトファイルにする

既に出来上がった関数やサブルーチンなどの手続きは，オブジェクトファイルとして持っておく方が，デバッグ時に何度も翻訳する手間が節約できます．

9.6 VPP WorkBench

VPP700/56 向けのプログラム開発環境としてプログラムの実行，デバッグ，およびチューニングを Unix ワークステーション vhsun, vcsun, vbsun, vrsun 上で統合的に行うツール “VPP WorkBench” があります．

機能は，すべてワークステーションの GUI を通してコンパイラの起動， VPP700/56 へのジョブ投入と実行管理，チューニングツールの起動， GUI ベースの会話型デバッグ，並列プログラムの各 PE での動作状況のグラフィックス表示などです．使用方法は [12] を参照してください．

[73] 350825804/1024/1024

10　ベクトルプログラミング入門

ベクトル並列計算機 VPP700/56 の性能を引き出すためには，並列化の前に 1PE での十分なベクトルチューニングを行う必要があります．この章では，ベクトル計算機に適した Fortran プログラムのチューニング方法について説明します．

10.1　ベクトル処理とは

10.1.1　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．

10.1.2　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．

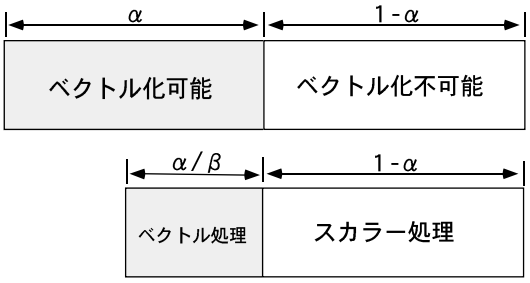


図 10.1: ベクトル化率と加速率の関係 I

ベクトル化率と加速率の関係は図 10.1 のようになります．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．しかし，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．

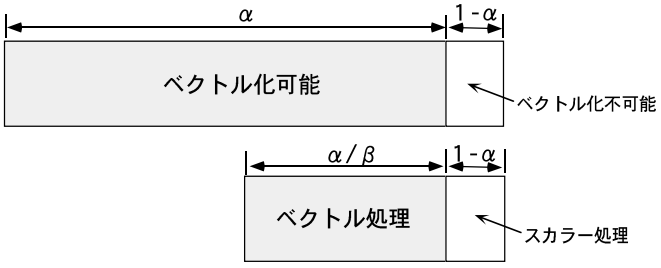


図 10.2: ベクトル化率と加速率の関係 II

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．

10.1.3 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[74]は

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

で与えられます[75]．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．

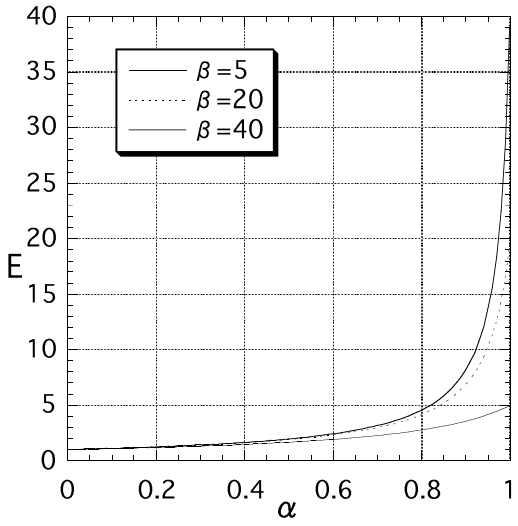


図 10.3: アムダールの法則

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．

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

10.1.4 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 nn ベクトルユニット占有時間の比

$$VPU/CPU$$

をベクトル化率の目安として下さい[76]．

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．

10.1.5 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[77]，`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[78] 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．

10.1.6 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表 10.1 の通りです．

[76] 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．
[77] where 構文，配列名での演算もベクトル化の対象となります．
[78] わかりやすい言葉を使えば「ぶんまわす」

表 10.1: ベクトル化の対象となるデータ型[79]

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

4 倍精度データはベクトル化の対象外であることに注意してください.

10.1.7 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.

```
00000286 s              do i=1,9
00000287 m               if(Node_u(i).ne.0) then
00000288 s4               do j=1,4
00000289 v4                im=Node_u(i)
00000290 v4                jm=Node_p(j)
00000291 m4                G(im,2*n+jm)   = G(im,2*n+jm)   - Local_Ex(j,i)
00000293 v4               end do
00000294 v                end if
00000295 v              end do
```

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.

表 10.2: DO 文に対する表示記号

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.

表 10.3: DO 文以外に対する表示記号

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 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 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      END DO
```

[79] REAL*4 はREAL, REAL(KIND=4) と同じ，また REAL*8 は DOUBLE PRECISION, REAL(KIND=8) と同じです.

この数字はループアンローリング[80]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．

10.2　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．

10.2.1　代入文

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

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．

10.2.2　条件つき計算

```
00000012      v        DO I=1,N
00000013      v         IF(A(I)>0.5D0) THEN
00000014      v          C(I)=B(I)+A(I)
00000015      v         ELSE
00000016      v          C(I)=SQRT(B(I))
00000017      v         END IF
00000018      v        END DO
```

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．

10.2.3　総和・内積計算

```
00000015                S=0.0D0
00000016                T=0.0D0
00000017      v        DO K=1,N
00000018      v         S=S+A(I,K)
00000019      v         T=T+A(I,K)*B(K,J)
00000020      v        END DO
```

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．

10.2.4　最大値・最小値

```
00000011                S= 1.0D+10
00000012                T=-1.0D+10
00000013      v        DO I=1,N
00000014      v         S=MIN(S,A(I))
00000015      v         T=MAX(T,A(I))
00000016      v        END DO
```

[80] 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．

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．

```
00000011              S= 1.0D+10
00000012              T=-1.0D+10
00000013      v       DO I=1,N
00000014      v        IF(S.GT.A(I)) S=A(I)
00000015      v        IF(T.LT.A(I)) T=A(I)
00000016      v       END DO
```

さらに， Fortran 90 の標準組み込み関数 MAXVAL, MINVAL でもベクトル化されます．

```
00000011      v       S= MAXVAL(A(1:N))
00000012      v       T= MINVAL(A(1:N))
```

10.2.5 リストベクトル

```
00000013      v       DO I=1,N
00000014      v        B(I)=A(L(I))
00000015      v        C(I)=A(I)
00000016      v       END DO
```

L(I) が添字となるリスト (インデックス) ベクトルです．

10.2.6 収集・拡散計算

```
00000012              J=1
00000013      v       DO I=1,N
00000014      v        IF(A(I)<0.0D0) THEN
00000015      v         B(J)=A(I)
00000016      v         J=J+1
00000017      v        END IF
00000018      v       END DO
```

収集計算とは，条件を満たす配列要素を集める計算です．

```
00000020              J=1
00000021      v       DO I=1,N
00000022      v        IF(A(I)<0.0D0) THEN
00000023      v         A(I)=B(J)
00000024      v         J=J+1
00000025      v        END IF
00000026      v       END DO
```

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．

10.2.7 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．

```
00000072      s2        DO K=MM+1,M
00000073      v2         DO I=1,L
00000074      v2           C(I,J)=C(I,J)+A(I,K)*B(K,J)
00000075      v2         END DO
00000076      s2        END DO
```

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

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．

10.2.8 多重ループの一重化

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

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．

10.3 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．

10.3.1 回帰参照

DO ループの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 10.2.7節の例の様なループ交換がされている場合もあります．

```
00000018      s7   do i=2,N
00000019      s7    A(i)=A(i-1)/2 + B(i)
00000020      s7  end do
```

M-1800/20U, VP2600/10 では，一次回帰演算まではベクトル化できましたが， VPP700/56 ではできなくなりました．

10.3.2 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 I=1,N
00000013      s         CALL USER(A(I),S)
00000014      s         T=T+S
00000015      s        END DO
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 I=1,N
00000013      vi        CALL USER(A(I),S)
00000014      v         T=T+S
00000015      v        END DO
```

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．

10.3.3 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 I=1,N
00000014      m8        A(L(I)) = A(L(I)) + 3.0D0
00000015      v8       END DO
```

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         !OCL NOVREC(A)
00000014      v        DO I=1,N
00000015      v         A(L(I)) = A(L(I)) + 3.0D0
00000016      v        END DO
```

例では， DO ループ内の配列 A について回帰演算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 [1] を参照してください．

10.3.4　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 I=1,N
00000012      s9        B(I) = A(I) + 1.0Q0
00000013      s9        A(I) = 1.0Q0
00000014      s9       END DO
```

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ください.

10.4　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．つまり，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ことになります．具体的には次の手順となります.

1. CLOCKV 関数，実行解析ツール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，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.
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.

10.5　プログラミングの注意点

10.5.1　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， 10.2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.

10.5.2　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.

10.5.3　データアクセスの効率化 I

Fortran のデータは列方向に参照する方がメモリが連続してアクセスされ，高速に実行できます.

$$A = \begin{pmatrix} a_{11} & \boxed{a_{12}} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{pmatrix}$$ 列方向のアクセス

$$a_{11}\ a_{21} \cdots a_{n1} \boxed{a_{12}\ a_{22} \cdots a_{n2}} \cdots a_{1n}\ a_{2n} \cdots a_{nn}$$

$$A = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ \boxed{a_{21} \quad a_{22} \quad \cdots \quad a_{2n}} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{pmatrix}$$ 行方向のアクセス

$$a_{11}\ \boxed{a_{21}} \cdots a_{n1} a_{12}\ \boxed{a_{22}} \cdots a_{n2} \cdots a_{1n}\ \boxed{a_{2n}} \cdots a_{nn}$$

従って，2 次元配列の添字のそれぞれを二重の DO ループで動かすような場合には，内側の DO ループで第 1 の添字が動き，外側の DO ループで第 2 の添字が動くようにプログラムを書くとよいでしょう[81].

10.5.4 データアクセスの効率化 II

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．VPP700/56 では 2 次元配列 `A(N,N)` の宣言で `N=512` とすると最悪の性能劣化がおきます[82].

これを防ぐためには，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します．

```
      PROGRAM SAMPLE
      IMPLICIT NONE
      INTEGER NMAX
      PARAMETER(NMAX=512)
      REAL*8 A(NMAX+1,NMAX)   !  倍精度の場合
            :
```

10.5.5 リストベクトルの限界

リストベクトルは，メモリ節約の観点からよく使われます．しかし，VPP700/56 でリストベクトルを用いたプログラミングをすると，メモリアクセスの制限からピーク性能が約半分に落ちてしまいます．これに満足できない人は，リストベクトルを使ったアルゴリズムそのものを変更する必要があります．

[81] Fortran 90/VP の場合は，コンパイラが親切にループを入れ換えてくれる場合が多いです．むしろワークステーションでのプログラムに有効なテクニックです．ちなみに C はアクセスが逆になります．

[82] と富士通の開発の人から非公式に聞きました．同一のバンクに対して連続してアクセスを行なうとき，1 回のアクセスの後，次にデータアクセスが可能になる状態になるまでに長い時間を要する状態を「バンクコンフリクト」と呼びます．一般的に，N を 64 の倍数にすることは避けた方がいいでしょう．

11　並列化プログラミング入門

この章では，Fortran 90/VPP による並列プログラミングについて簡単に解説します．詳しくは [5], [2] を御覧ください．

11.1　並列化のイメージ

11.1.1　XOCL 行

Fortran 90/VPP では，ソースプログラムに「拡張最適化制御行」と呼ばれる並列化に関する指示行を埋め込み，翻訳時オプション -Wx を付加し翻訳，編集結合することにより並列化を行います．

```
      SUBROUTINE SETDAT(A,B)
      INTEGER LA,N,NPE,NMAX,NBPE
      PARAMETER (LA=18000,N=LA,NPE=32)
      PARAMETER (NMAX=((LA-1)/NPE+1)*NPE,NBPE=100*NPE,LD=NMAX)
      DOUBLE PRECISION A(LD,NMAX),B(LD,NMAX),PI
      INTEGER I,J
!XOCL PROCESSOR P_MAIN(NPE)                                  ! 拡張最適化制御行 1
!XOCL SUBPROCESSOR P(NPE)=P_MAIN(1:NPE)                      ! 拡張最適化制御行 2
!XOCL INDEX PARTITION IP= (PROC=P,INDEX=1:NMAX,PART=BAND)    ! 拡張最適化制御行 3
!XOCL LOCAL   A(:,/IP),B(:,/IP)                              ! 拡張最適化制御行 4
      PI=4.0D0*ATAN(1.0D0)
!XOCL SPREAD DO /IP                                          ! 拡張最適化制御行 5
      DO 10 J=1,N
      DO 20 I=1,N
        A(I,J)=SQRT(2.0D0/(N+1))*SIN(I*J*PI/(N+1))
        B(I,J)=A(I,J)
  20  CONTINUE
  10  CONTINUE
!XOCL END SPREAD                                             ! 拡張最適化制御行 6
      RETURN
      END
```

拡張最適化制御行は **XOCL 行**ともいいます[83]．XOCL 行の記述方法は，第 1 桁から第 5 桁までが《必ず》“!XOCL” で始まります．固定形式では *XOCL も許します．また，小文字で !xocl と書いても構いません．以下はすべて !XOCL で統一します．

```
!234567890                          ! これは単なるコメント
!XOCL PROCESSOR P_MAIN(NPE)         ! XOCL 行
```

[83] “eXtended Optimization Control Line” の略です．Fortran 90/VPP 独自の用語です．また，ベクトル化に関する指示行である OCL 行もあります ([1]).

!XOCL に続く第 6 桁目は空白です.

第 7 桁以降に並列化に関する構文を記述します[84]. !XOCL の継続は，継続する行の最後に & を記述します.

```
!XOCL INDEX PARTITION IP= (PROC=P,INDEX=1:NMAX, &     ! 次の行へ継続
!XOCL                      PART=BAND)
```

翻訳時オプション -Wx を指定しないと，!XOCL はコメント行と見なされます[85]．従って，単一 PE 版の Fortran プログラム (Fortran 90/VP) や他のワークステーションのコンパイラとの互換性は保たれます[86]．具体的な拡張最適化制御行の機能は次節で紹介します.

11.1.2 変数の配置

並列処理でもっとも重要なのが，配列に代表される**変数**を各 PE にどのように割り振るかです.

Fortran 90/VPP における変数の形態は，**グローバル変数**と**ローカル変数**とに分類されます.

●グローバル変数

1章で紹介したように， VPP700/56 は各 PE がそれぞれ独立したメモリとプロセッサを持つベクトル計算機だと見ることができます.

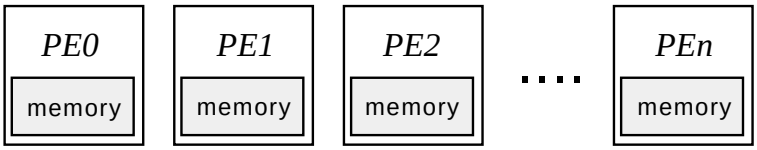


図 11.1: VPP700/56 のイメージ

Fortran 90/VPP では，これら物理的に分散したメモリをあたかも一つのメモリ空間であるかのように見せる機能があります．論理的にメモリが 1 つのように見える空間を**グローバル空間**[87]と呼び，グローバル空間に配置された変数を**グローバル変数** (*global variable*) と呼びます.

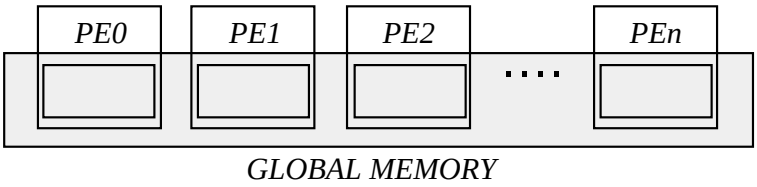


図 11.2: グローバル変数によるメモリの共有 I

図 11.2 はすべての PE のメモリを一つのグローバル空間として見た極端な例で，実際の計算では各 PE のメモリの一部をグローバル変数として宣言します.

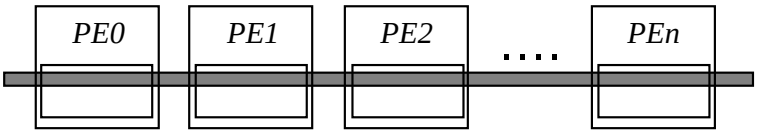


図 11.3: グローバル変数によるメモリの共有 II

[84] 並列化 Fortran は， FORTRAN 77 を強く意識した仕様となっており， Fortran 90 の「自由形式プログラミング」の精神から逸脱しています．しかし，現在のところ，プログラムの見やすさを考えると通常のプログラムも第 7 桁から書き始めた方がいいでしょう.

[85] Fortran 90 では，“!” 以降は注釈と見なされ無視されます．なお，この機能は FORTRAN77 EX/VP でも使用できます.

[86] とはいっても，あまり「見栄え」がいいプログラムとは言えなくなります.

[87] あくまでもソフトウェアの力によって実現された空間ですので，「仮想グローバル空間」ともいいます.

グローバル変数は，すべての PE から共通に参照することができます．つまり， VPP700/56 は物理的には「分散メモリ型計算機」でありながら，論理的にはグローバル変数によって「共有メモリ型計算機」であるかのように見ることもできます[88]．ただし，

> グローバル変数はデータアクセスに時間がかかり，配列演算もベクトル化できません．

PE 間のデータ通信はネットワークを経由して行います．異なる PE(例えば PE1 と PE2) のデータ転送は PE 内の転送 (例えば PE1 の中だけ) に比べて大量の時間を要します．また DO ループなどのベクトル化もできませんので，ベクトル計算機としての性能を発揮することができません．つまり，実際問題としてグローバル配列を用いた直接の演算は使いものになりません．

グローバル変数は，通常分割ローカル変数と `EQUIVALENCE` 文により記憶域を共有し，異なる PE 間のデータ交換を仲介するために利用されます (cf.11.2.9節).

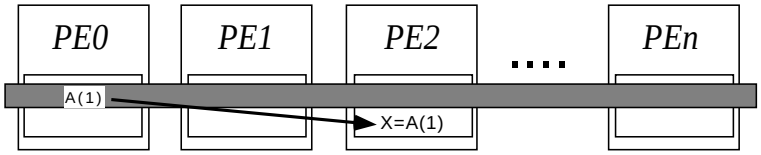


図 11.4: グローバル変数によるデータの参照

図 11.4 は PE0 の `A(1)` をグローバル変数として宣言することで，異なる PE から参照できるようにする例です．

●ローカル変数

各 PE のメモリに配置された変数を**ローカル変数** (*local variable*) と呼びます．ローカル配列[89]は，他の PE から参照できないかわりに， Fortran 90/VP の自動ベクトル化機能がすべて使えます．異なる PE からローカル変数を参照するためには，グローバル空間を介して行います．

ローカル変数は，データの分割方法によって，さらに**重複ローカル変数**と**分割ローカル配列**に分かれます．

○重複ローカル変数

`!XOCL` で何も指定しない変数は， すべての PE に重複して 割り当てられます．このローカル変数を重複ローカル変数 (*duplicate local variable*) と呼びます．

例えば， 8PE を用いたプログラムの中で `DOUBLE PRECISION A(100)` で宣言された配列 `A` は，各 PE で重複して

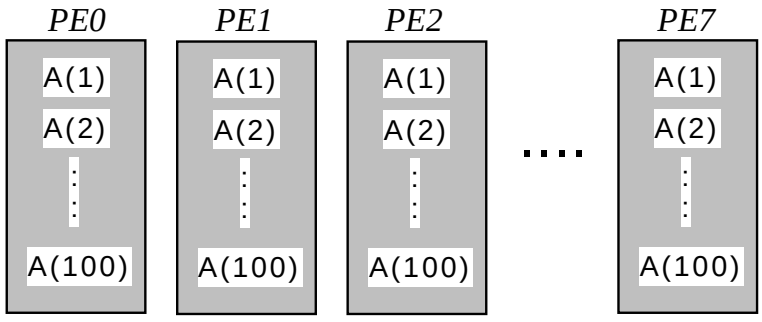


図 11.5: 重複ローカル変数

と配置されます．また，プログラムでこれまた `!XOCL` の指定がない場合は，各 PE でまったく同じ演算が実行されます．

[88]マニュアルにはこの構造を「階層メモリ型」と名付けています．
[89]配列宣言されたローカル変数の集まりのことです．

○分割ローカル配列

1 つの PE のメモリでは収まりきらない大きな配列を使用するプログラムでは，配列を複数の PE に分割して配置する必要があります．この配列を分割ローカル配列 (*partitioned local array*) と呼びます．

例えば，`DOUBLE PRECISION A(1000)` で宣言した `A` を 8 つの PE で均等に分割すると，配列のイメージは

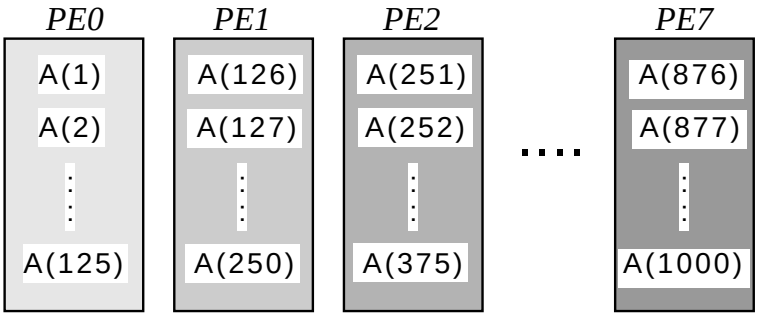


図 11.6: 分割ローカル配列

となります[90]．

11.1.3 実行方式

VPP700/56 の実行方式には，逐次，冗長，並列の 3 つの実行方式があります．

逐次実行	1 台の PE を使ってプログラムを実行します
冗長実行	すべての PE で同一処理を行う方式です
並列実行	複数の PE が処理を分担して並列に実行することです

マルチプロセッサの効果を引き出すためには，並列実行の部分の割合を高める必要があります．たとえば，1PE では A, B, C, D の処理を順番に実行する必要があり，もっともコストのかかる部分は C だとします．この C の部分を手続きの分割 (C1 ～ Cn) によって複数の PE で並列に実行することができれば， C の時間の短縮が全体の処理時間の短縮につながります．

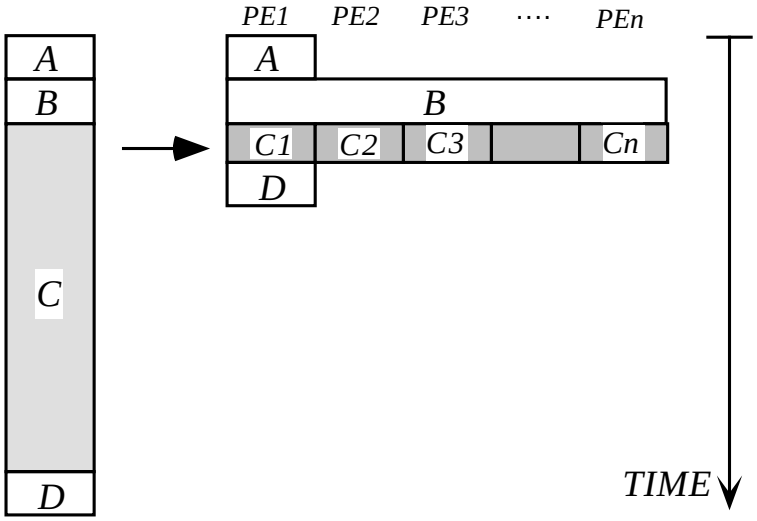


図 11.7: 並列処理による実行時間の短縮イメージ

[90]例では配列が PE 数できれいに割り切れていますが，割り切れなくても後ろのプロセッサで自動的に調整されますので，特に意識する必要はありません．

図 11.7 の A, D は 1 台の PE で実行されている逐次実行です． B はすべての PE で実行される冗長実行です．C の部分は各 PE に処理が分担されて処理されている並列実行です．

冗長実行は並列処理の意味では「無駄」な部分といえます．しかし，実際のプログラムではどうしても出てくるようです[91]．

そのため，センターでは並列に実行される各 PE の CPU 時間の**最長**のものに対して課金する方式を採用しています．

[91]『冗長』は「繁雑でわずらわしく長いこと」ですので，もちろん，いい意味ではありません．「冗長な会議」「冗長な話」「冗長な文章」で無駄な労力を使うのは世の常です．

11.2 拡張最適化制御行の機能

この節では，拡張最適化制御行 (XOCL 行) の機能を紹介します． XOCL 行は宣言構文 (declaration construct) と実行構文 (executable construct) に分類できます.

11.2.1 宣言構文一覧

宣言構文は，INTEGER I,J,K や REAL A(10,10) などの宣言文のある場所に記述します.

表 11.1: 宣言構文一覧

制御文	機能
PROCESSOR	使用する PE の数と名前の宣言
SUBPROCESSOR	サブルーチン内で使用する PE の数と名前の宣言
INDEX PARTITION	分割指定の定義と名前の宣言
LOCAL	分割指定をしたローカル配列の宣言
GLOBAL	グローバル配列の宣言
PROC ALIAS	プロセッサグループの別名を宣言

11.2.2 実行構文一覧

実行構文は DO 文や IF 文などのある場所に記述します.

表 11.2: 実行構文一覧

制御文	機能
PARALLEL REGION	並列実行の開始
END PARALLEL REGION	並列実行の終了
SPREAD REGION	文の並びを分割し，分割部分を該当プロセッサで実行することを指定
REGION	SPREAD REGION の補助として区切りを示す
END SPREAD REGION	SPREAD REGION の終了
SPREAD DO	DO ループを分割し，並列実行する
END SPREAD DO	SPREAD DO の終了
SPREAD MOVE	ローカル変数とグローバル変数の代入を一括転送する
END SPREAD MOVE	SPREAD MOVE の終了
SPREAD ASSIGNMENT	代入文の実行を， PE が分担して実行する
END SPREAD ASSIGNMENT	SPREAD ASSIGNMENT の終了
BROADCAST	指定された PE のデータを転送する
OVERLAPFIX	袖付き分割ローカル配列の袖部分にデータを転送する
UNIFY	各 PE で定義された重複ローカル配列の値を揃える
MOVEWAIT	転送の完了を待つことを指示
BARRIER	PE 間でバリア同期を取ることを指示
LOCKON	リージョン間での排他制御の区間の開始
END LOCKON	リージョン間での排他制御の区間の終了

11.2.3 PROCESSOR 文

PROCESSOR 文は，プログラムで使用する PE の数と名前を宣言します．何はともあれ，この指定がないとプログラムが始まりません.

```
!XOCL PROCESSOR P(16)          ! プロセッサの定義
```

例では，16 個の PE を使用することを宣言しています．PROCESSOR 文によって定義された “P” を**プロセッサグループ**と呼びます．プロセッサグループの名前は任意です．“PRO” であっても “MAIN_P” であっても構いません．しかし，プログラム内の変数名とプロセッサグループ名を 重複させることはできません．この記事では，一番安直な名前として “P” とします．

PE の数は，パラメータを用いて

```
      PARAMETER (NP=8)
!XOCL PROCESSOR P(NP)
```

と書くこともできます． Fortran 90 なら

```
      INTEGER,PARAMETER :: NP=8
!XOCL PROCESSOR P(NP)
```

となります[92]．プロセッサグループを 2 次元 (P(3,3)) や 3 次元 (P(2,3,4)) で定義することもできます[93]が，これは上級者用のテクニックです．

11.2.4 PROC ALIAS 文

既存のプロセッサグループを分割して別の名前のプロセッサグループとして定義することもできます．別名は PROC ALIAS 文で宣言します．

```
!XOCL PROCESSOR P(32)                          ! プロセッサの定義
!XOCL PROC ALIAS P1(8)=P(1:8),P2(24)=P(9:32)   ! P を二つに分け，別名をつける
```

例では，プロセッサグループ P を二つに分割して， 8 つの PE から成る P1 と， 24 の PE から成る P2 に分割しています．PROC ALIAS 文は異なるプロセッサグループを駆使するため，初心者はまず使いません．

11.2.5 SUBPROCESSOR 文

サブルーチンに XOCL 行が出てくる場合は，メインプログラムで宣言したプロセッサグループの情報を引き継ぐ必要があります．

まず，メインプログラムで，プロセッサグループが次のように宣言してあるとします．

```
      PARAMETER (NP=32)
!XOCL PROCESSOR P(NP)
```

サブルーチンでのプロセッサグループの引き継ぎは SUBPROCESSOR 文で行います．

```
      SUBROUTINE SETDAT(A,B)
      PARAMETER (NPE=32)
!XOCL PROCESSOR P(NPE)                         ! プロセッサグループの宣言
!XOCL SUBPROCESSOR PS(NPE)=P(1:NPE)            ! プロセッサグループの引き継ぎ
```

[92] Fortran 90 のパラメータ属性，および暗黙の型宣言追放の観点からは，“INTEGER,PARAMETER::NP=8” と書くべきですが，現状 Fortran 90 がさほど浸透していないことからこの後は FORTRAN 77 の記法を踏襲します．

[93] 上限は 7 次元です．

SUBPROCESSOR 文によって，メインプログラムのプロセッサグループがサブルーチン内のプロセッサグループ PS に引き継がれました．“1:NPE” は，1 から NPE までの PE を意味します．

また，メインプログラムとサブルーチンとは変数が独立ですので，メインプログラムと同じプロセッサグループ名 (ここでは P) を使いたければ，

```
      SUBROUTINE SETDAT(A,B)
      PARAMETER (NPE=32)
!XOCL PROCESSOR P_MAIN(NPE)                      ! プロセッサグループの宣言
!XOCL SUBPROCESSOR P(NPE)=P_MAIN(1:NPE)          ! プロセッサグループの引き継ぎ
```

と記述しても差し支えありません．大事なことは，メインプログラムで宣言したプロセッサグループの情報がサブルーチンに引き継がれることです[94]．

11.2.6 INDEX PARTITION 文

INDEX PARTITION 文は，分割方法を定義し，名前をつけます．指定できるものは，プロセッサグループの名前，次元，インデックスの範囲，分割方法，PE にオーバーラップを持たせるかどうかの 5 つです．具体的な配列の分割のイメージは次の LOCAL 文で紹介します．

```
!XOCL PROCESSOR P(32)                                  ! プロセッサグループの宣言
!XOCL INDEX PARTITION IP=(PROC=P,INDEX=1:100,PART=BAND) ! 分割方法を定義
```

“IP” は分割指定名です．プロセッサグループ名と同じで，名前は何でも構いません．“PROC=P” は，すぐ上で定義したプロセッサグループを分割に使用することを意味します．“INDEX=1:100” は分割する大きさの定義です．“PART=BAND” は，分割を均等に行うことを指定します．

分割方法は PART=BAND の他に循環分割 (PART=CYCLIC) および不均等分割があります[95]．

```
      PARAMETER(NMAX=10000)
!XOCL PROCESSOR P(16)
!XOCL INDEX PARTITION IP=(PROC=P,INDEX=1:NMAX,PART=BAND,OVERLAP=(1,1))
```

分割指定では，分割された左右の部分を重ねて持つことができます．この重なった部分を「袖」と呼びます．

OVERLAP=(1,1) は，左右に 1 個袖を持つことを指定します．

11.2.7 LOCAL 文

LOCAL 文は INDEX PARTITION 文で分割を指定したローカル配列を宣言します．

●分割例 (1 次元)

```
      PARAMETER(NMAX=10000)
      DOUBLE PRECISION A(NMAX)
!XOCL PROCESSOR P(4)                                   ! プロセッサグループの宣言
!XOCL INDEX PARTITION IP=(PROC=P,INDEX=1:NMAX,PART=BAND) ! 分割方法を定義
!XOCL LOCAL A(/IP)                                    ! 分割ローカル配列の宣言
```

[94] SUBPROCESSOR 文によるプロセッサの引き継ぎには他にも方法があります (cf.[2])．また，部分的にプロセッサを引き継ぐこともできます．現在のところ，並列化の機能を引き出すためには，サブルーチン単位で使用する PE 数を記述する必要があります．実は，PE 台数に依存しないプログラムの記述方法があるのですが，メーカーの正式サポートの回答を得ておりませんので，ここでは紹介できません．

[95] 循環分割, 不均等分割とも初級レベルでは使いませんので，説明を省略します．

1 次元の配列 A を 4 つのプロセッサに均等に割り当てます．“INDEX=1:NMAX” は配列の上下限を指定します．LOCAL 文の “/” は，「データ分割をするよ」という意味です．IP は，すぐ上で定義した分割方法の名前です．この分割指定により．配列 A は図 11.8 のように各 PE に配置されます．

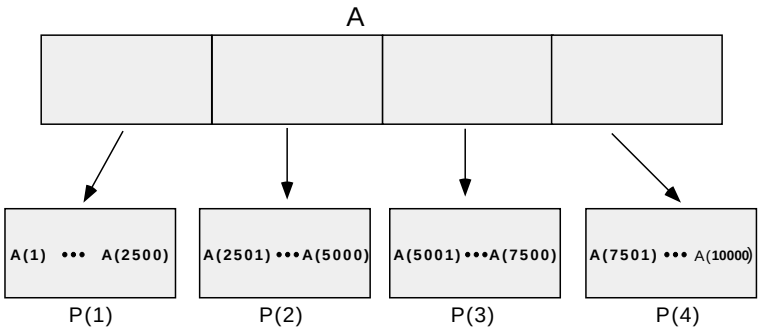


図 11.8: 分割ローカル配列 (1 次元)

PROCESSOR 文で定義された 4 つの PE(P(1) 〜P(4)) にそれぞれ担当となる配列データが割り付けられます．配列 A を例えば A(-500,500) で宣言した場合も上下限を設定すれば大丈夫です．

```
      PARAMETER(NMAX=500)
      DOUBLE PRECISION A(-NMAX,NMAX)
!XOCL PROCESSOR P(4)
!XOCL INDEX PARTITION IP=(PROC=P,INDEX=-NMAX:NMAX,PART=BAND)
!XOCL LOCAL A(/IP)
```

分割方法は直接 LOCAL 文に書き込むことも可能です．

```
      PARAMETER(NMAX=10000)
      DOUBLE PRECISION A(NMAX)
!XOCL PROCESSOR P(4)
!XOCL LOCAL A(/(P,INDEX=-NMAX:NMAX,PART=BAND))
```

例の 2 つの LOCAL 文は同じ意味を持ちます．が，プログラムの見やすさや拡張性を考えると INDEX PARTITION 文で名前を定義した方がいいでしょう．

また，LOCAL 文に “/” がないと，重複ローカル配列と見なされます．

●分割例 (2 次元)

Fortran で多次元のデータを分割する場合，第 1 添字でベクトル化し，最後の添字で並列化をする方法が，メモリアクセスの観点から見て有効です．従って，特別な事情がないかぎり，

2 次元配列の分割は 2 番目の添字を分割する

ようにして下さい．

```
      PARAMETER(NMAX=10000)
      DOUBLE PRECISION A(NMAX,NMAX)                        ! 2 次元配列
!XOCL PROCESSOR P(4)                                       ! プロセッサグループの宣言
!XOCL INDEX PARTITION IP=(PROC=P,INDEX=1:NMAX,PART=BAND) ! 分割方法を定義
!XOCL LOCAL A(:,/IP)                                       ! 分割ローカル配列の宣言
```

10000 × 10000 の正方行列 A を第 2 添字で均等に分割しました．メモリアクセスの良い行方向はそのままです．“:” は「この方向には分割をしないよ」という意味です．この分割指定により，配列 A は図 11.9 のように各 PE に配置されます．

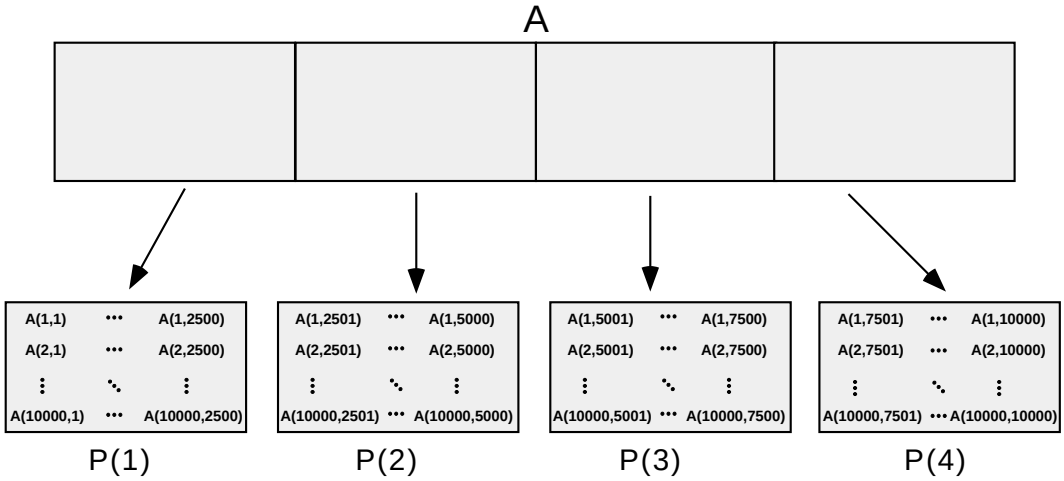


図 11.9: 分割ローカル配列 (2 次元)

なお，各添字を LOCAL A(/IP1,/IP2) などと同時に分割することも可能です[96]．

●分割例 (3 次元)

3 次元の流体の数値シミュレーションでは， X, Y, Z 方向の格子を切りますので， 3 次元配列がよく登場します．

```
      PARAMETER(NX=150,NY=100,NZ=80)
      DOUBLE PRECISION A(NX,NY,NZ)                          ! 3 次元配列
!XOCL PROCESSOR       P(4)                                  ! プロセッサグループの宣言
!XOCL INDEX PARTITION IP=(PROC=P,INDEX=1:NZ,PART=BAND)      ! 分割方法を定義
!XOCL LOCAL           A(:,:,/IP)                            ! 分割ローカル配列の宣言
```

150 × 100 × 80 の 3 次元配列 A を第 3 添字で均等に分割しました．この分割指定により，配列 A は図 11.10 のように各 PE に配置されます．

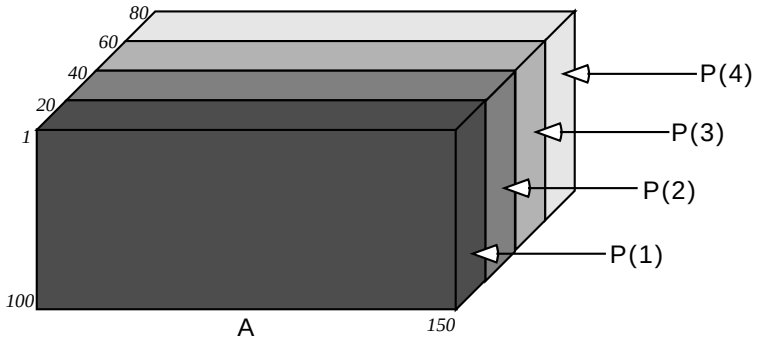


図 11.10: 分割ローカル配列 (3 次元)

2 番目の添字を分割する場合も同様です．

[96] もちろんデータの管理が複雑になります．

```
      PARAMETER(NX=150,NY=100,NZ=80)
      DOUBLE PRECISION A(NX,NY,NZ)                          ! 3 次元配列
!XOCL PROCESSOR       P(4)                                  ! プロセッサグループの宣言
!XOCL INDEX PARTITION IP=(PROC=P,INDEX=1:NY,PART=BAND)      ! 分割方法を定義
!XOCL LOCAL           A(:,/IP,:)                            ! 分割ローカル配列の宣言
```

●分割例 (袖付き 2 次元)

分割指定により各 PE に割り当てられた分割ローカル配列は，割り当てられた当の PE からは高速にアクセスできますが，他の PE からはアクセスできません．しかし，プログラムでは他の PE の値がよく必要になります．差分法のアルゴリズムを持ち出すまでもなく，参照したい他の PE の値の多くは分割されたメモリの「境界」の部分です．

均等分割では，左右の添字の値を重なって持つことができます．具体的には，OVERLAP パラメータを指定し，隣の PE の境界の値を重複して持つことにより，前後の添字の指す配列を利用することができます．この重なった添字のことを**袖** (*overlap*) と呼びます．

袖

OVERLAP(l,r) は，分割後の各添字の範囲に対して，小さい方向に l 個，大きい方向に r 個伸ばすことを意味します[97]．

```
      PARAMETER(NMAX=10000)
      DOUBLE PRECISION A(NMAX,NMAX)
!XOCL PROCESSOR P(4)
!XOCL INDEX PARTITION IP=(PROC=P,INDEX=1:NMAX,PART=BAND,OVERLAP=(1,1))
!XOCL LOCAL A(:,/IP)
```

10000 × 10000 の正方行列 A を第 2 添字で前後 1 個の袖つきで均等に分割しました．この分割指定により，配列 A は図 11.11 ように袖つきで各 PE に配置されます．

[97] 袖の長さは INDEX の範囲を超えなければいくらでも伸ばすことができますが，その分，分割ローカル配列のメモリが増えます．

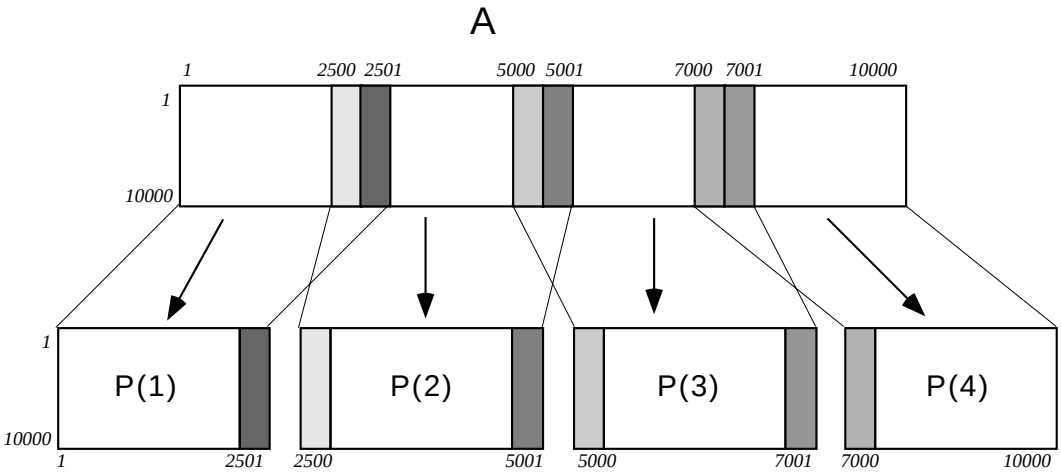


図 11.11: 袖付き分割ローカル配列 (2 次元)

11.2.8 GLOBAL 文

GLOBAL 文は，各 PE で共有したい配列を指定します．次の EQUIVALENCE 文と対でよく使われます．
配列を分割する要領は LOCAL 文と同じです．ただし，グローバル配列は袖を持つことができません．

```
      PARAMETER(NMAX=10000)
      DOUBLE PRECISION A(NMAX,NMAX),B(NMAX)
!XOCL PROCESSOR P(4)                                          ! プロセッサグループの宣言
!XOCL INDEX PARTITION IP=(PROC=P,INDEX=1:NMAX,PART=BAND) ! 分割方法を定義
!XOCL GLOBAL A(:,/IP)                                         ! グローバル配列の宣言
```

11.2.9 EQUIVALENCE 文

GLOBAL 文で宣言したグローバル配列は，各 PE で共有のデータとして扱うことができます．しかし，アクセスに時間がかかるうえにベクトル化もできません．一方，LOCAL 文で宣言した分割ローカル配列は，割り当てられた PE 内ならば (同じプロセッサ内のデータですので) 高速にアクセスできますし，ベクトル化も可能です． Fortran の EQUIVALENCE 文を用いてグローバル配列と分割ローカル配列を結合すると，両方の性格をもったデータを宣言することができます．

EQUIVALENCE 文で結合する場合のグローバル配列は分割指定をしないか，袖以外のすべてがローカル配列と一致した分割でなくてはなりません．プログラムの見やすさからは，分割指定を書かない方がよいでしょう．

```
      PARAMETER(NMAX=10000)
      DOUBLE PRECISION A(NMAX),AG(NMAX)
!XOCL PROCESSOR P(8)                                          ! プロセッサグループの宣言
!XOCL INDEX PARTITION IP=(PROC=P,INDEX=1:NMAX,PART=BAND) ! 分割方法を定義
!XOCL GLOBAL AG                                               ! グローバル配列の宣言
!XOCL LOCAL  A(/IP)                                           ! ローカル配列の宣言
      EQUIVALENCE(AG,A)                                       ! 記憶単位の共有
```

EQUIVALENCE 文による結合によって，例えばプロセッサ P(1) からはグローバル配列 AG の AG(1) 〜AG(10000) すべてが，また，分割ローカル配列 A の A(1) 〜A(1250) が参照可能です．さらに，A(1) 〜A(1250) と AG(1) 〜AG(1250) が結合されています[98].

[98] Fortran 90 では「使わない方がいい機能」 ([38]) に入っている EQUIVALENCE 文が Fortran 90/VPP の大きな特徴となっているのは悲しいことです.

```
      PARAMETER(NMAX=10000)
      DOUBLE PRECISION A(NMAX),AG(NMAX)
!XOCL PROCESSOR P(8)
!XOCL INDEX PARTITION IP=(PROC=P,INDEX=1:NMAX,PART=BAND,OVERLAP=(1,1))
!XOCL GLOBAL AG
!XOCL LOCAL  A(/IP)
      EQUIVALENCE(AG,A)
```

こちらは 1 次元配列の袖付きの例です．A と AG の結合は図 11.12 のようになります．

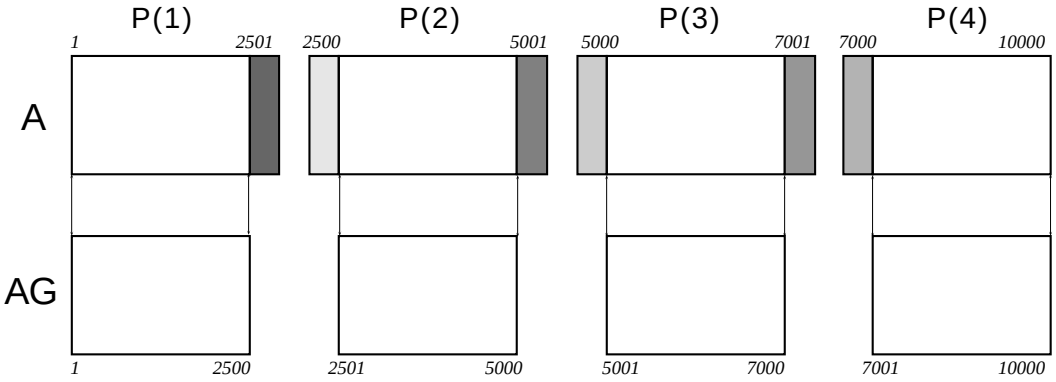


図 11.12: 分割ローカル配列とグローバル配列の結合

```
      PARAMETER(NMAX=10000)
      DOUBLE PRECISION A(NMAX+1,NMAX),AG(NMAX+1,NMAX)
!XOCL PROCESSOR P(32)
!XOCL INDEX PARTITION IP=(PROC=P,INDEX=1:NMAX,PART=BAND)
!XOCL GLOBAL AG
!XOCL LOCAL A(:,/IP)
      EQUIVALENCE(AG,A)
```

2 次元のグローバル配列と分割ローカル配列との結合例です．

11.2.10 分割指定の省略形

これまで見てきた宣言構文で，分割にかかわるパラメータはそれぞれ省略形 (デフォルト) を持っています．最初はきちんと INDEX PARTITION 文で分割指定を定義すべきですが，慣れてくるとだんだん不精な構文が書けるようになります．

- “PROC=” は省略できます．ただし項目の最初にプロセッサグループ名を書く必要があります．
- INDEX の下限の省略値は 1 です．
- 分割指定で INDEX を省略すると，配列の下限・上限が採用されます．
- PART の省略値は “PART=BAND” です．

つまり，

```
      PARAMETER(NMAX=10000)
      DOUBLE PRECISION A(NMAX+1,NMAX)
!XOCL PROCESSOR P(32)
!XOCL INDEX PARTITION IP=(PROC=P,INDEX=1:NMAX,PART=BAND)
!XOCL LOCAL A(:,/IP)
```

は

```
      PARAMETER(NMAX=10000)
      DOUBLE PRECISION A(NMAX+1,NMAX)
!XOCL PROCESSOR P(32)
!XOCL INDEX PARTITION IP=(P,INDEX=NMAX)
!XOCL LOCAL A(:,/IP)
```

と書くことができます．さらに INDEX PARTITION を削ってしまい，プロセッサグループだけを使って，

```
      PARAMETER(NMAX=10000)
      DOUBLE PRECISION A(NMAX+1,NMAX)
!XOCL PROCESSOR P(32)
!XOCL LOCAL A(:,/(P))
```

と書くこともできます． SSL II/VPP のマニュアル ([19]) にはよくこの記述がでてきます．

11.2.11 PARALLEL REGION 構文

並列実行の開始と終了は PARALLEL REGION，END PARALLEL REGION で指示します．この構文は必ずプログラムのどこかに書く必要があります．END PARALLEL REGION は “END PARALLEL” と省略することができ，手引書の例では省略形しか記述してありませんので，本稿でも END PARALLEL と書くことにします．

PARALLEL REGION 文の前までは，1 つの PE による逐次実行が行われています．PARALLEL REGION 文に到達すると，実行ファイルとデータが他の PE に複写され並列実行が開始されます．END PARALLEL により並列実行が終了すると，再び 1 つの PE による逐次処理に戻ります．

```
      PROGRAM TEST
      PARAMETER (LA=18000,N=LA,NPE=32)
      PARAMETER (NMAX=((LA-1)/NPE+1)*NPE,NBPE=100*NPE,LD=NMAX)
      REAL*8  A(LD,NMAX),B(LD,NMAX),C(LD,NMAX),W(LD,NBPE)
      REAL*8  AG(LD,NMAX),BG(LD,NMAX),CG(LD,NMAX)
!XOCL PROCESSOR P(NPE)
!XOCL INDEX PARTITION IP= (PROC=P,INDEX=1:NMAX,PART=BAND)
!XOCL GLOBAL          AG(:,/IP),BG(:,/IP),CG(:,/IP)
!XOCL LOCAL           A(:,/IP),B(:,/IP),C(:,/IP)
      EQUIVALENCE    (AG,A),(BG,B),(CG,C)
!XOCL PARALLEL REGION                          !-----+
        CALL SETDAT(A,B)                      !     |
        CALL MAMUL(AG,BG,CG,LA,LA,LA,N,N,N)   !     |--- 並列実行
        CALL EST(C,ERROR)                     !     |
!XOCL END PARALLEL                             !-----+
      STOP
      END
```

なお，1 回のPARALLEL REGION 構文の実行にはものすごいオーバヘッドがかかるため，プログラムの中に幾つも PARALLEL REGION 構文を記述すると，いつまで経っても計算が終らない事態を招きます．従って，

> PARALLEL REGION 構文はプログラムの中に 1 回しか使ってはいけません．

PARALLEL REGION 構文は，主プログラムまたは副プログラムの中に (全体で)1 回だけ記述するようにして下さい．

11.2.12 SPREAD REGION 構文

SPREAD REGION[99]文は，実行文を分割してプロセッサに処理を割り振ることを指示します．

終了は END SPREAD REGION 文です．END SPREAD REGION は “END SPREAD” と省略可能です．“REGION” は区切りを表します．

```
!XOCL PROCESSOR P(16)                                          ! プロセッサ数の宣言
!XOCL PROC ALIAS P1(8)=P(1:8),P2(4)=P(9:12),P3(4)=P(13:16) ! 別名の定義
        :
!XOCL SPREAD REGION / P1
        :                  ! P(1) ～ P(8) での処理
!XOCL REGION / P2
        :                  ! P(9) ～ P(12) での処理
!XOCL REGION / P3
        :                  ! P(13) ～ P(16) での処理
!XOCL END SPREAD REGION
```

SPREAD REGION 構文では入れ子[100]も可能です．SPREAD REGION 構文は，プログラムを複数の PE に分割して割り振るため，どの PE で何が実行されているかをきちんと管理する必要があります．そのため，初心者は手を出さない方がいいでしょう．

実行文の分割よりも，次に紹介する DO ループの分割の方が簡単かつ並列化効果が期待できます．

11.2.13 SPREAD DO 構文

SPREAD DO 構文は，DO ループを分割し，並列に実行することを指示します．DO ループの分割の終りを意味する END SPREAD DO は “END SPREAD” と省略可能です．また，総和や最大・最小などのよく使われる演算を並列に実行する関数もサポートされています．

●分割ローカルデータの演算例 I

```
      PARAMETER (N=10000)
      DOUBLE PRECISION A(N)
!XOCL PROCESSOR P(4)                                    ! プロセッサ数の宣言
!XOCL INDEX PARTITION IP=(PROC=P,INDEX=1:N,PART=BAND) ! 分割方法の指定
!XOCL LOCAL A(/IP)                                      ! 分割ローカル配列の宣言
            :
!XOCL PARALLEL REGION                                   ! 並列実行の開始
            :
!XOCL SPREAD DO /IP                                     !---+
       DO I=1,N                                         !   |
        A(I)=SQRT(2.0D0*I)                              !   |---DO ループの分割
       END DO                                           !   |
!XOCL END SPREAD DO                                     !---+
            :
!XOCL END PARALLEL                                      ! 並列実行の終了
            :
```

4 つの PE を用いて DO ループを分割し，並列処理します．A は分割ローカルデータとして各 PE に割り当てられているので，それぞれの PE 上で

[99] “SPREAD” は「広める」「ばらまく」という意味です．

[100] SPREAD REGION 構文の中で SPREAD REGION 構文を使うこと．

```
DO I=1,2500
 A(I)=SQRT(2.0D0*I)
END DO
```
P(1)

```
DO I=2501,5000
 A(I)=SQRT(2.0D0*I)
END DO
```
P(2)

```
DO I=5001,7500
 A(I)=SQRT(2.0D0*I)
END DO
```
P(3)

```
DO I=7501,10000
 A(I)=SQRT(2.0D0*I)
END DO
```
P(4)

とベクトル処理されます.

“/IP” は，INDEX PARTITION で指定した分割方法に従って DO ループを分割することを指示しています. 分割方法 (“/IP”) を省略した場合は，「DO ループの添字の範囲をプロセッサ数で均等に分割する」仕様となります. 従って，上の例では “/IP” は省略しても構わないのですが，複数の分割方法を使いこなす将来を見越して，きちんと書いた方が無難です.

●分割ローカルデータの演算例 II

```
      PARAMETER (N=18000)
      REAL*8  A(N+1,N),B(N+1,N)
!XOCL PROCESSOR P(32)
!XOCL INDEX PARTITION IP=(PROC=P,INDEX=1:N,PART=BAND)
!XOCL LOCAL A(:,/IP),B(:,/IP)

!XOCL PARALLEL REGION
           :
       (A,B に関する演算)
           :
!XOCL SPREAD DO /IP
       DO J=1,N
         DO I=1,N
          A(I,J)=A(I,J)+B(I,J)/2.0D0
         END DO
       END DO
!XOCL END SPREAD DO
!XOCL END PARALLEL
      STOP
      END
```

多重ループの一番内側を分割した場合は，ベクトル性能が出ない場合がありますので，分割はできるだけ外側のループで分割します[101].

○グローバル配列の演算は？

グローバル配列が DO ループの中に混入すると，ベクトル性能が著しく低下します. 対策としては，DO ループの実行に先だってローカル配列に SPREAD MOVE 構文を使って転送するか，EQUIVALENCE 文によって結合された分割ローカル配列を使用するかします.

○重複ローカルデータの演算は？

各 PE にデータが重複して存在する重複ローカルデータも SPREAD DO で並列処理することができます. ただし，DO ループを並列化すると，DO ループ終了後の重複ローカルデータは PE の一部に正しい演算結果があり，その他は適当な数値が入っている状態となっています. そのため，PE に散らばったデータをきちんと集め直す必要があります. その機能として UNIFY 文が提供されています (cf.11.2.20節).

[101] また，PART=CYCLIC で循環分割された DO ループはベクトル化されません.

●グローバル関数

SPREAD DO 構文には，数値計算でよく使われる処理を**グローバル関数** (*global function*) として用意しています．グローバル関数は総和 (SUM)，最大 (MAX)，最小 (MIN)，論理積 (AND)，論理和演算 (OR)，最終値 (LAST) です．

```
      PARAMETER (N=10000,NPE=32)
      REAL*8  A(N+1,N),T
      INTEGER I,J,MAXI,MAXJ
!XOCL PROCESSOR P(32)
!XOCL INDEX PARTITION IP=(PROC=P,INDEX=1:NMAX,PART=BAND)
!XOCL LOCAL A(:,/IP)
         :
      (A の計算)
         :
      T=0.0D0
!XOCL SPREAD DO /IP
      DO J=1,N
       DO I=1,N
        IF ( T < ABS(A(I,J) ) ) THEN      !--+
          T = ABS(A(I,J)                  !  |
          MAXI = I                        !  |-- 最大値と添字の捜索
          MAXJ = J                        !  |
        END IF                            !--+
       END DO
      END DO
!XOCL END SPREAD DO MAX(T,MAXI,MAXJ)
```

分割した DO ループで最大値の計算を行うと，それぞれの PE が担当する局所的な最大値は求まりますが，全体の最大値はこれだけでは分かりません．グローバル関数 SUM は，各 PE の最大値を調べ，配列全体の最大値 (例では T) を計算します．また，添字の値 (例では MAXI, MAXJ) も調べることができます．添字を調べる必要がない場合は，(例では)MAX(T) とだけ記述します．

```
      DO I=1,N
       PIVOT(I) = 0
      END DO
!XOCL SPREAD DO /IP
      DO J=1,N
       DO I=1,N
        PIVOT(LIST(I,J)) = PIVOT(LIST(I,J)) + 1
       END DO
      END DO
!XOCL END SPREAD DO SUM(PIVOT)
```

2 次元配列 LIST の情報を PIVOT に格納する例です．SUM グローバル関数に記述する変数，配列 (この場合 PIVOT) は，SPREAD DO の前ですべての PE で同じ値にしておく必要があります．

●グローバル関数の注意事項

グローバル関数に配列を使用する場合，作業領域が不足して処理が遅くなる場合があります．その場合，実行時オプション “-Wl,-Pg” に続けて作業領域を Kbyte 単位で指定すると，処理が高速化されます．省略値は 1Kbyte です．例えば実行ファイルを a.out に対し， 1MB を指定する場合は，

```
a.out -Wl,-Pg1024
```

と指定します.

11.2.14 SPREAD MOVE 構文

SPREAD MOVE 構文は，ローカル変数とグローバル変数の代入を一括して行うことを指定します．分割の方向が異なるような分割ローカルデータ相互の代入をグローバルデータを通して行う場合などに使います.

SPREAD DO を用いても同様の代入が可能ですが，SPREAD MOVE に比べてかなり遅くなります．SPREAD MOVE 構文は，あくまでも「代入文」のみに使える機能です.

一括代入の終了は END SPREAD MOVE です．“END SPREAD” と省略することもできます.
また，END SPREAD MOVE に続けて，データ転送の識別名をつけることで，その後の MOVEWAIT 文で転送の完了を確認することができます.

```
!XOCL PROCESSOR P(4)
!XOCL INDEX PARTITION IP=(PROC=P,INDEX=1:100000,PART=BAND)
      REAL*8  A(10000,10000),B(10000,10000)
!XOCL LOCAL  A(:,/IP)
!XOCL GLOBAL B(/IP,:)
          :
!XOCL SPREAD MOVE /IP,:          !---+
      DO J=1,N                   !   |
        DO I=1,N                 !   |
         A(I,J)=B(I,J)           !   |-- データの一括代入
        END DO                   !   |
      END DO                     !   |
!XOCL END SPREAD MOVE (ID)       !---+ 識別番号 ID を振る
!XOCL MOVEWAIT (ID)              !     転送の完了を確認
```

例では，分割ローカル配列 A にグローバル配列 B の値を全要素一括して代入しています.

SPREAD MOVE /IP,: の “/IP,:” は，DO ループの出現順 に分割方法を指定しています．つまり，第 2 添字を分割すること，第 1 添字は何もしないことを意味します[102]．分割指定は分割ローカル配列に合わせて指定します.

END SPREAD MOVE (ID) の “(ID)” は，転送の識別番号 ID (任意です) を振り当てることを意味します．直後の MOVE WAIT (ID) は，識別番号 ID の転送が完了したことを確認する指令です[103].

11.2.15 SPREAD ASSIGNMENT 構文

SPREAD ASSIGNMENT 構文は，代入文の実行を PE が分担して実行することを指示します．記述できる文は代入文の他に，単純 WHERE 文，WHERE 文です．組み込み代入文の変数 (左辺) はベクトル添字を持たない配列である必要があります.

[102] “,:” は省略できます.
[103] なぜ識別番号や転送確認の指示があるのかというと，実行時間の短縮のためには，データの転送中に他の処理を行ったり，複数の転送をうまく使いこなしたりするチューニングを想定しているためです.

```
!XOCL PROCESSOR P(4)
!XOCL INDEX PARTITION IP=(PROC=P,INDEX=1:100,PART=BAND)
      REAL*8  A(100),B(100),C(100)
           :
!XOCL SPREAD ASSIGNMENT /IP
       A(1:50) = B(11:60) + C(21:70)
!XOCL END SPREAD ASSIGNMENT
```

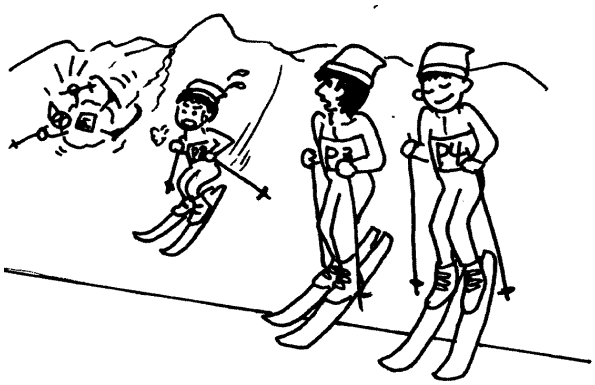
11.2.16 BROADCAST 文

BROADCAST 文は，LOGICAL が true となる重複ローカル変数の値を，他の PE に転送することを指定します.

```
      PARAMETER (N=10000,NPE=32)
!XOCL PROCESSOR P(NPE)
!XOCL INDEX PARTITION IP=(PROC=P,INDEX=1:N,PART=BAND)
      REAL*8  A(N),B(N)
      LOGICAL FLAG                    ! 論理型の定義
          :
      FLAG=.FALSE.                    ! PE すべてを false にする
!XOCL SPREAD DO /IP
      DO J=1,1                        ! P(1) でのみ計算
        FLAG=.TRUE.                   ! P(1) の論理型を true にする
        DO I=1,N                      !--+
           A(I)=B(I)+2.0              !   |--P(1) での計算
        END DO                        !--+
      END DO
!XOCL END SPREAD DO
!XOCL BROADCAST(A) (FLAG)            ! 転送を行う
```

11.2.17 BARRIER 文

各 PE に異なる処理を割り振って「せーの」で実行に移しても，処理の形態によっては，ある PE の処理はすぐに終ってしまったのに，いつまでも処理が終らない PE があったりします. Fortran 90/VPP には，並列実行する PE にあるポイントで実行を停止し，すべての PE の処理がそのポイントまで到達した後，実行を再開するという同期機能があります. この機能を**バリア同期**と呼びます. BARRIER 文は，すべての PE でバリア同期を取ることを指定します.



バリア同期

通常，SPREAD DO，SPREAD MOVE 構文などは，特に指定しなければ BARRIER 文が指定されたと見なされます[104].

[104]逆に BARRIER 文は，バリア同期をプログラマ自ら取り外して「ギンギン」のチューニングに手を染め出した時の歯止めだと言えます.

11.2.18 LOCKON 構文

LOCKON 構文は，SPREAD REGION 構造で分割された複数のリージョン間のプログラムの実行を制御します．
LOCKON で振られた番号と同じものを持つ文は同時には実行されません (排他制御)．

11.2.19 MOVEWAIT 文

MOVEWAIT 文は，SPREAD MOVE 構文，OVERLAPFIX 文，UNIFY 文で開始された PE 間データ転送の完了を確認します．

11.2.20 UNIFY 文

UNIFY 文は，各 PE で定義された重複ローカル配列の値を揃えることを指示します．

```
      PARAMETER (N=1000)
      DOUBLE PRECISION A(N,N),B(N,N)                    ! 重複ローカル配列の宣言
!XOCL PROCESSOR P(4)                                    ! プロセッサ数の宣言
!XOCL INDEX PARTITION IP=(PROC=P,INDEX=1:N,PART=BAND) ! 分割方法の指定
            :
!XOCL PARALLEL REGION                                   ! 並列実行の開始
!XOCL SPREAD DO /IP                                     !---+
       DO J=1,N                                         !   |
         DO I=1,N                                       !   |
           A(I,J)=SIN(B(I,J)+1.0D0)                     !   |---DO ループの分割
         END DO                                         !   |
       END DO                                           !   |
!XOCL END SPREAD DO                                     !---+
!XOCL UNIFY(A(:,/IP)) (ID)                              ! データを均一にする
!XOCL MOVEWAIT(ID)                                      ! データ転送の完了を確認
            :
!XOCL END PARALLEL
```

上の例で宣言された重複ローカル配列 A は， 4 つの PE で全く同じ値を持ちます．SPREAD DO でのループの分割により，処理は各 PE で分担して

```
DO J=1,250
DO I=1,1000
A(I,J)=SIN(B(I,J)+1)
END DO
END DO
```
P(1)

```
DO J=251,500
DO I=1,1000
A(I,J)=SIN(B(I,J)+1)
END DO
END DO
```
P(2)

```
DO J=501,750
DO I=1,1000
A(I,J)=SIN(B(I,J)+1)
END DO
END DO
```
P(3)

```
DO J=751,1000
DO I=1,1000
A(I,J)=SIN(B(I,J)+1)
END DO
END DO
```
P(4)

と実行されます．ところが，重複ローカルデータは，各 PE で同じ値を持っていなければいけないデータですので，各 PE が分担した計算結果を他の PE に知らせる必要があります．
“ID” は識別番号です．直後の MOVEWAIT は，データ転送の完了を確認する文です．

11.2.21 OVERLAPFIX 文

OVERLAPFIX 文は，袖付き分割ローカル配列の袖部分にデータを転送することを指示します．
置き換えは，EQUIVALENCE 文によって結合しているグローバル配列の対応する範囲の値で行われます．
例えば，単一 PE で動く次のプログラムの並列化を企んだとします．

```
! ======================== STEP0 ==================================== !
      PARAMETER(N=1000000)
      DOUBLE PRECISION A(N,N),B(N,N)
           :
      DO J=2,N-1
        DO I=1,N
          B(I,J) = A(I,J-1) -2*A(I,J) +A(I,J+1)
        END DO
      END DO
```

まず，PE32 台で A, B の第 2 添字を均等に分割します．また，A, B を分割ローカル配列として各 PE に割り振ります．さらに，SPREAD DO 構文で DO ループを分割します．

```
! ======================== STEP1 ==================================== !
      PARAMETER(N=1000000,NPE=32)
!XOCL PROCESSOR P(32)
!XOCL INDEX PARTITION IP=(PROC=P,INDEX=1:N,PART=BAND)
      DOUBLE PRECISION A(N,N),B(N,N)
!XOCL LOCAL A(:,/IP),B(:,/IP)
           :
!XOCL PARALLEL REGION
           :
!XOCL SPREAD DO /IP
      DO J=2,N-1
        DO I=1,N
          B(I,J) = A(I,J-1) -2*A(I,J) +A(I,J+1)
        END DO
      END DO
!XOCL END SPREAD DO
!XOCL PARALLEL REGION
```

これで終りではありません．分割された DO ループでは，PE 上にあるデータ以外に隣の PE の境界のデータが必要です．

従って A を袖付きのローカル配列として定義し直します．方法は幾つかありますが，LOCAL 文の A の宣言に OVERLAP パラメータを追加するのがいいでしょう．

```
! ======================== STEP2 ==================================== !
      PARAMETER(N=1000000,NPE=32)
!XOCL PROCESSOR P(32)
!XOCL INDEX PARTITION IP=(PROC=P,INDEX=1:N,PART=BAND)
      DOUBLE PRECISION A(N,N),B(N,N)
!XOCL LOCAL A(:,/IP(OVERLAP=(1,1))),B(:,/IP)
           :
!XOCL PARALLEL REGION
           :
!XOCL SPREAD DO /IP
      DO J=2,N-1
        DO I=1,N
          B(I,J) = A(I,J-1) -2*A(I,J) +A(I,J+1)
        END DO
      END DO
!XOCL END SPREAD DO
!XOCL PARALLEL REGION
```

まだあります．ローカル配列 A の袖へのデータの転送は，EQUIVALENCE 文で結合されたグローバル配列を介して行います．とにかく，グローバル配列を宣言し，結合させてみましょう．

```
! ======================== STEP3 ==================================== !
      PARAMETER(N=1000000,NPE=32)
!XOCL PROCESSOR P(32)
!XOCL INDEX PARTITION IP=(PROC=P,INDEX=1:N,PART=BAND)
      DOUBLE PRECISION A(N,N),B(N,N),AG(N,N)
!XOCL LOCAL  A(:,/IP(OVERLAP=(1,1))),B(:,/IP)
!XOCL GLOBAL AG
      EQUIVALENCE (AG,A)
           :
!XOCL PARALLEL REGION
           :
!XOCL SPREAD DO /IP
      DO J=2,N-1
        DO I=1,N
          B(I,J) = A(I,J-1) -2*A(I,J) +A(I,J+1)
        END DO
      END DO
!XOCL END SPREAD DO
!XOCL PARALLEL REGION
```

これで OVERLAPFIX 文が登場する準備が整いました．OVERLAPFIX 文は，グローバル配列 AG から袖付き分割ローカル配列 A の袖にデータを転送することを指示します．

```
! ======================== STEP4 ==================================== !
      PARAMETER(N=1000000,NPE=32)
!XOCL PROCESSOR P(32)
!XOCL INDEX PARTITION IP=(PROC=P,INDEX=1:N,PART=BAND)
      DOUBLE PRECISION A(N,N),B(N,N),AG(N,N)
!XOCL LOCAL  A(:,/IP(OVERLAP=(1,1))),B(:,/IP)
!XOCL GLOBAL AG
      EQUIVALENCE (AG,A)
           :
!XOCL PARALLEL REGION
           :
!XOCL OVERLAPFIX(A) (ID)    ! A の袖に AG の値を転送
!XOCL MOVEWAIT (ID)         ! 転送の完了を待つ
!XOCL SPREAD DO /IP
      DO J=2,N-1
        DO I=1,N
          B(I,J) = A(I,J-1) -2*A(I,J) +A(I,J+1)
        END DO
      END DO
!XOCL END SPREAD DO
!XOCL PARALLEL REGION
```

転送の識別名 ID は何でも構いません．OVERLAPFIX 文にグローバル配列 AG の姿は見えませんが，袖のデータを提供する大切な役目を果たしています．

11.3 サブルーチンの記述方法

11.3.1 プロセッサグループの受渡し

メインプログラムで宣言したプロセッサグループの情報はSUBPROCESSOR 文でサブルーチンに引き渡します.

```
      PROGRAM MAIN                          !  メインプログラム
      PARAMETER (NPE=4)                     !  プロセッサ数
!XOCL PROCESSOR PE(NPE)                     !  プロセッサグループの宣言
            :
!XOCL PARALLEL REGION                       !  並列実行開始
            :
      CALL SUB(A)                           !  サブルーチンの CALL
            :
!XOCL PARALLEL REGION                       !  並列実行終了
            :
      SUBROUTINE SUB(A)                     !  サブルーチン
      PARAMETER (NPE=4)                     !  プロセッサ数
!XOCL PROCESSOR PE(NPE)                     !  プロセッサグループの宣言
!XOCL SUBPROCESSOR PES(NPE)=PE(1:NPE)       !  プロセッサグループの引き継ぎ
            :
```

現在の Fortran 90/VPP の文法でプロセッサ数に依存しないサブルーチンを書くことは，できないことはないのですが，EQUVALENCE 文や OVERLAPFIX 文が使えないなどの強い制限があるため，通常はいちいちプロセッサ数を宣言することになります[105].

サブルーチンに XOCL 行を記述しない場合は，SUBPROCESSOR 文によるプロセッサグループの引き継ぎは不要です．ただし，そのサブルーチンはすべての PE で同じ処理がされます (冗長実行)．この部分が著しく足を引っ張るようでしたら，並列化を検討する必要があります.

11.3.2 データの受渡し

特に，グローバル，分割ローカル配列をサブルーチンに渡す場合は，実引数と仮引数は同じ分割をしたデータである必要があります.

例えば，メインプログラムが

```
      PROGRAM MAIN                              !  メインプログラム
      PARAMETER (NPE=32,MAXDIM=100001)          !  プロセッサ数，次元数
!XOCL PROCESSOR PE(NPE)                         !  プロセッサグループの宣言
!XOCL INDEX PARTITION IP=(PE,INDEX=1:MAXDIM)    !  分割方法の指定
      DOUBLE PRECISION A(MAXDIM,MAXDIM),AG(MAXDIM,MAXDIM) !  配列の宣言
!XOCL LOCAL  A(:,/IP)                           !  分割ローカル配列の宣言
!XOCL GLOBAL AG                                 !  グローバル配列の宣言
      EQUIVALENCE (A,AG)                        !  記憶領域の共有
            :
!XOCL PARALLEL REGION                           !  並列実行開始
            :
      CALL MAKE_A(L,NU,U1,U2,A)                 !  サブルーチンの CALL
            :
!XOCL PARALLEL REGION                           !  並列実行終了
      END
```

[105]ではSL II/VPP はプロセッサ数に依存しないプログラムをどうやって書いているのだろうという疑問が出てきますが，ソースが公開されていないので調べることができません．なお [5] の 5.7 節で PE 数に依存しない並列手続き方法を公開しています.

とすると，サブルーチンにも同様の分割指定が必要です．

```
      SUBROUTINE MAKE_A(L,NU,U1,U2,A)
      PARAMETER (NPE=32,MAXDIM=100001)       !  プロセッサ数，次元数
!XOCL PROCESSOR MAIN_PE(NPE)                 !  プロセッサグループの宣言
!XOCL SUBPROCESSOR PE(NPE)=MAIN_PE(1:NPE)    !  プロセッサグループの引き継ぎ
!XOCL INDEX PARTITION IP=(PE,INDEX=1:MAXDIM) !  分割方法の指定
      DOUBLE PRECISION A(MAXDIM,MAXDIM)      !  配列の宣言
!XOCL LOCAL  A(:,/IP)                        !  分割ローカル配列の宣言
            :
      RETURN
      END
```

慣れた人は，同様の手続き部分を別のファイルに記述し，include 文で読み込んだりしています．

また，サブルーチン内で，グローバル配列と分割ローカル配列を EQUIVALENCE 文で結合する必要がある場合は，グローバル配列を COMMON 文で渡し，サブルーチン内で分割ローカル配列を宣言し結合します．

```
      PROGRAM MAIN
      PARAMETER (NPE=8,MAXDIM=100001)
!XOCL PROCESSOR PE(NPE)
!XOCL INDEX PARTITION IP=(PE,INDEX=1:MAXDIM)
      DOUBLE PRECISION AG(MAXDIM)
!XOCL GLOBAL AG
      COMMON/BLK/AG
            :
!XOCL PARALLEL REGION
            :
      CALL SUB                               !  サブルーチンの CALL
            :
!XOCL PARALLEL REGION
            :
      SUBROUTINE SUB
      PARAMETER (NPE=8,MAXDIM=100001)
!XOCL PROCESSOR MAIN_PE(NPE)
!XOCL SUBPROCESSOR PE(NPE)=MAIN_PE(1:NPE)
!XOCL INDEX PARTITION IP=(PE,INDEX=1:MAXDIM)
      DOUBLE PRECISION A(MAXDIM),AG(MAXDIM)
!XOCL LOCAL  A(/IP)                          !  分割ローカル配列の宣言
!XOCL GLOBAL AG                              !  グローバル配列の宣言
      COMMON/BLK/AG
      EQUIVALENCE (A,AG)
            :
      RETURN
      END
```

なお，COMMON 文の共通ブロックに属する要素に，重複ローカル変数，分割ローカル配列，グローバル変数を混在させることはできません[106]．

11.4 ファイル入出力

分割ローカル配列の入出力をする場合の注意点です．

[106] Fortran 90 では「使わない方がいい機能」([38]) に入っている COMMON 文が Fortran 90/VPP の大きな特徴となっているのは，実に悲しいことです．

- ファイルは PARALLEL REGION 文の前にオープンする.
- EQUIVALENCE 文により結合したグローバル配列で行う.
- グローバル配列の読み書きは冗長実行部で行い，SPREAD DO 内には書かない.

重複ローカル配列は従来通りのプログラムで問題ありません. READ, WRITE には， DO 型並びも使用できますが，配列名でのアクセスが効率的です.

```
      PROGRAM MAIN
      PARAMETER (NPE=8,MAXDIM=100001)
!XOCL PROCESSOR PE(NPE)
!XOCL INDEX PARTITION IP=(PE,INDEX=1:MAXDIM)
      DOUBLE PRECISION A(MAXDIM,MAXDIM),AG(MAXDIM,MAXDIM)
!XOCL LOCAL  A(:,/IP)
!XOCL GLOBAL AG
      EQUIVALENCE (A,AG)
      OPEN(33,FILE='input.data')            ! ファイルのオープン
      OPEN(15,FILE='output.data')           ! ファイルのオープン
!XOCL PARALLEL REGION
            :
      READ(33) AG
            :
      WRITE(15) AG
            :
!XOCL END PARALLEL REGION
      CLOSE(15,FILE='output.data')          ! ファイルのクローズ
      CLOSE(33,FILE='input.data')           ! ファイルのクローズ
      END
```

11.5 並列化例 (パラメトリックスタディ)

入力データやパラメータを変えることにより，何度も同じ計算を繰り返し実行するようなケースを**パラメトリックスタディ** (*parametric study*) と呼びます.

簡単な例で説明します.

```
      PROGRAM MAIN
      DOUBLE PRECISION A(1000),B(1000)
      INTEGER I
      READ(5) B                         ! データの読み込み
      DO I=1,1000                       !
       CALL SUB(A(I),B(I))              ! 同じサブルーチンを CALL
      END DO                            !
      WRITE(6) A                        ! 処理結果の書き出し
      END
!
      SUBROUTINE SUB(X,Y)
      DOUBLE PRECISION X,Y
       Y=SIN(X)
      RETURN
      END
```

5 番からデータを読み込み，B(I) をパラメータとして，次々に A(I) を求めるプログラムです．この場合，個々の I に対する SUB の処理は全く独立ですので，簡単に並列化することができます.

```
      PROGRAM MAIN
      PARAMETER (NPE=16)                    ! プロセッサ数の宣言
!XOCL PROCESSOR PE(NPE)                     ! プロセッサグループの宣言
!XOCL INDEX PARTITION IP=(PE,INDEX=1:1000) ! 分割方法の指定
      DOUBLE PRECISION A(1000),B(1000)
      INTEGER I
      READ(5) B                             ! データの読み込み
!XOCL PARALLEL REGION                       ! 並列実行の開始
!XOCL SPREAD DO /IP                         ! DO ループの分割
      DO I=1,1000
       CALL SUB(A(I),B(I))                  ! 同じサブルーチンを CALL
      END DO
!XOCL END SPREAD DO                         ! DO ループの分割
!XOCL UNIFY (A(/IP)) (ID)                   ! データを揃える
!XOCL MOVEWAIT(ID)                          ! データ転送の完了
!XOCL END PARALLEL REGION                   ! 並列実行の終了
      WRITE(6) A                            ! 処理結果の書き出し
      END
!
      SUBROUTINE SUB(X,Y)
      DOUBLE PRECISION X,Y
       Y=SIN(X)
      RETURN
      END
```

なお，パラメトリックスタディによる並列化プログラムのサンプル[107]を kyu-cc の /usr/local/doc 下に公開します．各自コピーして参照してください．

```
kyu-cc% copy /usr/local/doc/parasta.txt .  ↵    <--- 使用方法のテキスト
kyu-cc% copy /usr/local/doc/parasta.tar .  ↵    <--- プログラム，データ (tar 形式)
```

11.6 並列化例 (SSL II/VPP)

数値計算でコストのかかる部分は，多くが連立 1 次方程式や固有値計算などの線形計算に集中するといってもいいでしょう． Fortran 90/VPP の賢い利用方法としては，並列計算用にチューニングされたサブルーチンライブラリ SSL II/VPP を積極的に利用することです．これらのライブラリは VPP700/56 の性能を最大限に発揮するようにチューニングされていますので，自分で組んだプログラムの方が速いということはまずありません[108]．

ここでは，需要が最もあると思われる連立 1 次方程式サブルーチン DP_VLAX の使用方法について，具体的な例をあげて説明します．

11.6.1 サブルーチンの引数の説明

man dp_vlax で見ればわかるように，DP_VLAX の引数は 15 個もあります[109]．ただし， SSL II/VPP, SSL II/VP の引数の形式は一つ憶えてしまえばあとは似たりよったりなので，面倒でもすべて説明します．

[107] 著作権は富士通に帰属しています．
[108] もし勝ったら是非センターまで御一報下さい．
[109] はっきり言って，富士通の日本語オンラインマニュアルの全角英数字は大いに間抜けです．

●呼び出し形式

```
CALL DP_VLAX(AG,KA,NA,N,B,IBLKS,EPSZ,ISW,IP,IS,WG,WA,WU,IWU,ICON)
```

●機能

実係数連立 1 次方程式 $Ax = b$ を，ブロック化した外積形の LU 分解法で解きます． A は $n \times n (n \geq 1)$ の正則な実行列， b は n 次元の実定数ベクトル， x は n 次元の解ベクトルです．

●引数の説明

整数は 4 バイトです．定数として入力できる引数もありますが，パラメータとしてきちんと宣言した方がデバッグに有利です．

表 11.3: DP_VLAX の引数一覧

引数	型	属性	入出力	内容
AG	倍精度実数	2 次元配列	入出力	入力として行列 A を AG(1:N,1:N) に格納．出力として行列 L と行列 U が AG に返される． AG は AG(KA,NA) で宣言されるグローバル配列で，このルーチンを呼び出すリージョンで 2 次元目を均等に分割されている．AG(1:N,1:N) 以外の AG の値は保証されない．
KA	整数	変数	入力	AG，WG の 1 次元目の大きさ (宣言した値)．KA $\geq$ N．
NA	整数	変数	入力	AG，WG の 2 次元目の大きさ (宣言した値)．NA $\geq$ N．NA はリージョンを構成する PE 数と IBLKS の積の倍数であること．
N	整数	変数	入力	行列 A の次数 n．
B	倍精度実数	1 次元配列	入出力	大きさ N のベクトル b を入力．解ベクトル x が出力される．
IBLKS	整数	変数	入力	ブロック化した外積形 LU 分解を行うブロックの大きさ．偶数でありかつ， 30 $\leq$IBLKS$\leq$ 60 であること．
EPSZ	倍精度実数	変数	入力	ピボットの相対零判定値．EPSZ$\geq$0.0D0．0.0D0 のときは標準値が採用される．
ISW	整数	変数	入力	同一の係数行列 A をもつ方程式 $Ax = b_i$ に対し，ISW=1 のとき 1 組目の方程式を解く．ISW=2 のとき 2 組目以降の方程式を解く．ただし，このとき B の値だけを新しいベクトルに変え，それ以外の引数はそのままで使用する．
IP	整数	1 次元配列	出力	部分ピボット選択による行の入換えの履歴を示す大きさ n のベクトル．
IS	整数	変数	出力	行列 A の行列式を求めるための情報 (1 または -1)，演算後の配列 AG の n 個の対角要素と IS の値を掛け合わせると行列式が得られる．
WG	倍精度実数	2 次元配列	作業域	WG(KA,NA) なるグローバル配列で，このルーチンを呼び出すリージョンで 2 次元目を均等に分割されている．
WA	倍精度実数	2 次元配列	作業域	WA(N,IBLKS) なる配列．
WU	倍精度実数	1 次元配列	作業域	WU(IWU) で宣言される配列．
IWU	整数	変数	入力	作業用配列 WU の大きさ．NA をプロセッサ数で割った値．
ICON	整数	変数	出力	ICON=0 エラーなし． ICON=20000 行列 A のある行の要素がすべて零であったか，又はピボットが相対的に零となった．行列 A は非正則の可能性が強い． ICON=30000 NA がリージョンを構成する PE 数と IBLKS の積の倍数でない．またはNA<N, KA<N, N<1 である．または EPSZ<0.0D0 である．または IBLKS>60, IBLKS<30 である．または，グローバル配列が正しく分割されていない．

●使用上の注意

1. グローバル配列 AG は AG(1:N,1:N) の部分を渡します．通常 AG(KA,NA) と宣言していれば問題ありません．

2. SSL II/VP や NUMPAC のサブルーチンと異なり，2 次元配列の 2 次元目の大きさ NA も引数に必要です．また，NA は (IBLKS×PE 数) で割り切れる必要があります．

3. EPSZ に標準値以外の値を設定したとき，ピボットが[110] 設定した EPSZ 以下なら，そのピボットを零と見なし ICON=20000 として処理を打ち切ります．EPSZ の標準値は，丸め誤差の単位を u としたとき，EPSZ=$16 \times u$ です．なおピボットが小さくなっても計算を続行させる場合には，EPSZ へ極小の値を設定すればよいのですが，その結果は保証されません[111]．

4. 同一の係数行列 A をもつ方程式 $Ax = b_i$ を続けて解く場合は，2 回目以降を ISW=2 として解くと，もっともコストを要する LU 分解の過程を省略するため，実行時間が大幅 ($O(n)$ 違います) に短縮できます．

5. IP と IS は，LU 分解の過程で部分ピボット選択による行の入れ換えが生じた場合の情報を受け持ちます[112]．

6. DP_VLAX には，3 つ (も) の作業用配列が必要です．特に，WG は AG と全く同じ記憶領域が必要です．

11.6.2 単一 PE 版プログラム例

それでは，具体的な例に沿って DP_VLAX を利用するまでの過程を追いかけてみましょう．
まず，単一 PE で動作する Fortran プログラムを用意します．$n \times n$ 行列 $A = (a_{ij})$ を

$$a_{ij} = \sqrt{\frac{2}{n+1}} \sin\left(\frac{ij\pi}{n+1}\right),$$

$n \times 1$ ベクトル $b = (b_i)$ を

$$b_i = \sum_{j=1}^{n} a_{ij}$$

で定義します．すると連立 1 次方程式 $Ax = b$ の解は

$$x \equiv 1$$

と (理論的に) なります．プログラム Linear_Equation1 は，A, b を作成し，SSL II/VP のサブルーチン DVLAX に放りこんで得られた x(数値解) の誤差を

$$\max_{1 \le i \le n} |x_i - 1|$$

で測定するものです．

---

[110] 行列の消去段階で割り算を司る代表元です．ゼロになってしまうと計算が破綻してしまいます．
[111] 数値的に行列が非正則になったということは，もともとの問題に何らかの原因があるのではないかと疑って見るべきです．ただし，プログラムのチェックが先です．
[112] 「部分ピボット選択」と言われてもピンと来ない人は数値計算の本を御覧下さい．

```
      program Linear_Equation1                    ! プログラムの名前
      implicit none                               ! 暗黙の型宣言の抑止
      integer,parameter             :: KA=501     ! 1 次元目の大きさのパラメータ
      integer,parameter             :: NA=500     ! 2 次元目の大きさのパラメータ
      real(kind=8),dimension(KA,NA) :: A          ! A の宣言
      real(kind=8),dimension(NA)    :: B,VW       ! B の宣言，作業領域 vw
      real(kind=8)                  :: error,s,Pi,EPSZ
      integer,dimension(NA)         :: IP
      integer                       :: N,i,j,IS,ICON,ISW
      N=KA-1                                      ! 次数の定義
      Pi=atan(1.0D0)*4.0D0                        ! πの定義
      B=0.0D0                                     ! B の初期化
       do j=1,N
        do i=1,N
         A(i,j)=sqrt(2.0D0/DBLE(N+1))*sin(dble(i)*dble(j)*Pi/dble(N+1))
           B(i)=B(i)+A(i,j)                       ! A,B の作成
        end do
       end do
! ------------------------------------------------------------------------ !
      EPSZ=0.0D0
      ISW=1
      call DVLAX(A,KA,N,B,EPSZ,ISW,IS,VW,IP,ICON)  ! 連立 1 次方程式を解く
! ------------------------------------------------------------------------ !
      error=0.0D0
      do i=1,N
       s=abs(B(i)-1.0D0)
       if(s.ge.error) error=s                     ! 誤差評価
      end do
      write(6,*) 'ICON=',ICON
      write(6,*) 'N=',N,' ERROR=',error
      end program Linear_Equation1
```

KA が N よりも一つだけ多いのは，バンクコンフリクトによる性能低下を避けるためで，それ以外の意味はありません．

11.6.3 SSL II/VPP のための変更

並列化に先だって，DP_VLAX のための変数を準備します．まず，多少メモリの無駄使いになるのは覚悟の上で，任意の IBLKS と PE 数に対応できるように NA の値を変更します．具体的には， Fortran で

$$\text{新 NA} = \left(\frac{\text{旧 NA}}{\text{PE 数} \times \text{IBLKS}} + 1 \right) \times \text{PE 数} \times \text{IBLKS}$$

と宣言します．

次に，作業領域などのこまごまとした変数を宣言し，サブルーチン名を書き直します．グローバル配列となる AG での演算はベクトル化できませんので，A を分割ローカル配列としグローバル配列 AG と EQUIVALENCE 文で結合します．ここまでを Linear_Equation2 とします．

```
      program Linear_Equation2
      implicit none
      integer,parameter               :: NPE=4,IBLKS=50,KA=501,LA=KA-1
      integer,parameter               :: NA=(LA/(NPE*IBLKS)+1)*NPE*IBLKS
      integer,parameter               :: IWU=NA/NPE
      real(kind=8),dimension(KA,NA)   :: A,AG,WG
      real(kind=8),dimension(LA,IBLKS):: WA
      real(kind=8),dimension(LA)      :: B
      real(kind=8),dimension(IWU)     :: WU
      real(kind=8)                    :: error,s,Pi,EPSZ
      integer,dimension(LA)           :: IP
      integer                         :: N,i,j,IS,ICON,ISW
      EQUIVALENCE (A,AG)
      N=KA-1
      Pi=atan(1.0D0)*4.0D0
      B=0.0D0
       do j=1,N
        do i=1,N
         A(i,j)=sqrt(2.0D0/DBLE(N+1))*sin(dble(i)*dble(j)*Pi/dble(N+1))
           B(j)=B(j)+A(i,j)
        end do
       end do
! ------------------------------------------------------------------------ !
      EPSZ=0.0D0
      ISW=1
      call DP_VLAX(AG,KA,NA,N,B,IBLKS,EPSZ,ISW,IP,IS,WG,WA,WU,IWU,ICON)
! ------------------------------------------------------------------------ !
      error=0.0D0
      do i=1,N
       s=abs(B(i)-1.0D0)
       if(s.ge.error) error=s
      end do
      write(6,*) 'ICON=',ICON
      write(6,*) '   N=',N,' ERROR=',error
      end program Linear_Equation2
```

11.6.4 並列化例

準備が整ったところで XOCL 行を挿入します．DP_VLAX の使用方法に従って AG(A), WG の 2 次元目を均等に分割し，グローバル配列を宣言します．二つの DO ループはともに SPREAD DO で分割処理します．最初のループでは，A が対称行列であることを用いて，B の添字を変更しています．B は重複ローカル配列ですので，分割処理の後は各 PE で値を均一にする UNIFY 文を実行します．

二つ目のループでは，最大値を検索するグローバル関数 MAX が必要です．この部分は無理して並列化しなくても冗長実行で計算可能です．

以上でとりあえずの並列化が完了しました．あとは，PE 数と次元のパラメータ NPE, KA だけを変更するだけで，自由に計算規模を変更できます．このプログラムを Linear_Equation3 とします．

現実の研究レベルとなると A, B の作成方法はこんなに楽ではありません．しかし，行列に i, j についての規則性があればそれほど苦労せずに並列化できるのではないでしょうか[113]．

[113] なにしろ，肝心な連立 1 次方程式の計算は SSL II/VPP まかせですから．

```
      program Linear_Equation3
      implicit none
      integer,parameter               :: NPE=4,IBLKS=50,KA=501,LA=KA-1
      integer,parameter               :: NA=(LA/(NPE*IBLKS)+1)*NPE*IBLKS
      integer,parameter               :: IWU=NA/NPE
      real(kind=8),dimension(KA,NA)   :: A,AG,WG
      real(kind=8),dimension(LA,IBLKS):: WA
      real(kind=8),dimension(LA)      :: B
      real(kind=8),dimension(IWU)     :: WU
      real(kind=8)                    :: error,s,Pi,EPSZ
      integer,dimension(LA)           :: IP
      integer                         :: N,i,j,IS,ICON,ISW
!XOCL PROCESSOR P(NPE)
!XOCL INDEX PARTITION PX=(PROC=P,INDEX=1:NA,PART=BAND)
!XOCL LOCAL  A(:,/PX)
!XOCL GLOBAL AG, WG(:,/PX)
      EQUIVALENCE (A,AG)
!XOCL PARALLEL REGION
      N=KA-1
      Pi=atan(1.0D0)*4.0D0
      B=0.0D0
!XOCL SPREAD DO /PX
       do j=1,N
        do i=1,N
         A(i,j)=sqrt(2.0D0/DBLE(N+1))*sin(dble(i)*dble(j)*Pi/dble(N+1))
           B(j)=B(j)+A(i,j)
        end do
       end do
!XOCL END SPREAD DO
!XOCL UNIFY(B(/PX)) (ID)
!XOCL MOVEWAIT (ID)
! ------------------------------------------------------------------------ !
      EPSZ=0.0D0
      ISW=1
      call DP_VLAX(AG,KA,NA,N,B,IBLKS,EPSZ,ISW,IP,IS,WG,WA,WU,IWU,ICON)
! ------------------------------------------------------------------------ !
      error=0.0D0
!XOCL SPREAD DO /PX
      do i=1,N
       s=abs(B(i)-1.0D0)
       if(s.ge.error) error=s
      end do
!XOCL END SPREAD DO MAX(error)
!XOCL END PARALLEL
      write(6,*) 'ICON=',ICON
      write(6,*) '   N=',N,' ERROR=',error
      end program Linear_Equation3
```

11.6.5 測定データ

計算の大部分は DP_VLAX のコストですので，この部分のみ PEPA のサブルーチンで性能を測定しました．

●並列化性能

1PE でメモリーに入りきるように次数を N=10000 に固定して並列化効率を測定しました．計算部分の AG には約 763MB のメモリが必要です．

表 11.4: N=10000 の処理性能

PE 数	GFLOPS	ピーク比
1(DVLAX)	2.1	95%
1	1.8	81%
2	3.6	81%
4	6.9	78%
8	12.5	71%
16	21.0	59%
32	31.0	44%

PE 数が増えるに従ってだんだんピーク性能比が落ちていきます．これは 1 台の PE が担当する配列が小さくなったため，ベクトル計算機の性能が十分発揮できなくなったためと思われます．

●最大性能

次は，複数の PE で確保できる最大の記憶領域近くまで N を大きくして測定してみました．

表 11.5: 最大記憶領域近くの N の処理性能

PE 数	N	AG のメモリ	GFLOPS	ピーク比
1(DVLAX)	10000	0.7GB	2.1	95%
1	10000	0.7GB	1.8	81%
2	14500	1.5GB	3.7	84%
4	19500	2.8GB	7.4	84%
8	29000	6.2GB	14.7	83%
16	40000	11.9GB	28.9	82%
32	58000	25.0GB	56.5	80%

表 11.4, 11.5 を図にしたものが図 6.4, 6.5 です．次数を大きくすると 32PE でもピーク性能の 80% をなんとか達成しました[114]．

[114]同じ容量の作業領域をとっているのだから，このくらいの性能は出てくれないと困るという気もします．また，最後の N=58000 の連立 1 次方程式を解くのに約 25 分要します．

11.7 追加機能の紹介

レベルアップにともない Fortran 90/VPP に新しい機能が追加になりました.

11.7.1 RESIDENT 機能

SPREAD DO 文にグローバル変数をローカル変数と同様の方法で高速にアクセスする RESIDENT 機能が追加されました. この機能によって, グローバル変数に対応するローカル変数の宣言や EQUIVALENCE 文の記述が不要になります. 例えば

```
      PARAMETER(N=10000)
!XOCL PRECESSOR P(10)
!XOCL INDEX PARTITION IP=(PROC=P,INDEX=1:N,PART=BAND)
      DOUBLE PRECISION G(N),L(N)
!XOCL GLOBAL G
!XOCL LOCAL L(/IP)
      EXQUIVALENCE (G,L)
!XOCL PARALLEL REGION
!XOCL SPREAD DO /IP
      DO I=1,N
       L(I)=SQRT(2.0D0*I)
      END DO
!XOCL EDN SPREAD DO
        :
```

というプログラムはグローバル変数を直接用いて

```
      PARAMETER(N=10000)
!XOCL PRECESSOR P(10)
!XOCL INDEX PARTITION IP=(PROC=P,INDEX=1:N,PART=BAND)
      DOUBLE PRECISION G(N)
!XOCL GLOBAL G(/IP)
!XOCL PARALLEL REGION
!XOCL SPREAD DO RESIDENT(G) /IP     ! RESIDENT 指定
      DO I=1,N
       G(I)=SQRT(2.0D0*I)
      END DO
!XOCL EDN SPREAD DO
        :
```

と記述できるようになりました.

RESIDENT はグローバル配列へのアクセスが PE 内に割り付けられた範囲に閉じていることを指定するものです. RESIDENT 指定されたグローバル配列は, その変数のアクセスを実行する PE 上に割り当てられているとみなされ, ローカル変数と同様の方法でアクセスすることができます. 詳細は [5]5.5 節を参照してください.

11.7.2 モジュール内の XOCL 行記述

モジュールの宣言部に XOCL 行を記述できるようになりました. 従って PROCESSOR 文や GLOBAL 文などをモジュールにまとめて記述し, サブルーチンなどで Fortran 90 の USE 文を用いて参照することができます.

11.8 メッセージパッシングライブラリの利用

Fortran 90/VPP 以外の並列化として，メッセージパッシングライブラリ (MPL) を用いた方法がプログラミングが可能です.

以下， MPL の概要と参考文献を紹介します.

11.8.1 PVM

PVM(Parallel Virtual Machine) は，アメリカオークリッジ国立研究所で開発されたメッセージパッシングライブラリです.

通常はネットワークを介した分散環境上で各ノード間の通信のために利用されますが， VPP700 上で PE をノードとしたデータ変換を行います.

サポート機能としては，プロセス生成，各種問い合わせなどのプロセス管理，同期／非同期送受信，広報送信などのメッセージ通信，複数メッセージの一括送受信，複数プロセス間にまたがる演算機能を提供します.

●マニュアル

解説記事 [48] および富士通のマニュアル [13] を参照してください[115].

11.8.2 MPI

MPI Forum において業界標準を目指して仕様検討されているメッセージパッシングライブラリです．分散されたプロセス間の集配信機能，集積演算およびグルーピングなどの強力な機能をサポートしています．なお，Fortran の自由形式はサポートされていません.

●マニュアル

富士通のマニュアル [14] を参照してください[116].

11.8.3 PARMACS

プロセス管理，メッセージ送信，同期処理，各種問い合わせ，論理ネットワーク定義を C 言語関数または Fortran サブルーチンとして制御するメッセージパッシングライブラリです． VPP700 シリーズは分散メモリ並列コンピュータですが，高速クロスバ通信により共有メモリ型並列コンピュータ並の性能を実現します (とカタログに書いてあります).

●マニュアル

[15] を御覧下さい． PostScript ファイルが kyu-cc の`/usr/local/doc/PM61-UserGuide.ps.gz` にあります. また，`README` ファイルを次のように参照することができます.

```
kyu-cc% rsh kyu-vpp cat /usr/lang/parmacs/doc/README.VPP700 | more ↵
```

[115] PVM の日本語マニュアルは `ftp://etlport.etl.go.jp/pub/pvm/new-jdoc/jpvmug-950205.ps.gz` にあります.
[116] MPI の日本語マニュアルは `ftp://et7alpha.et.u-tokai.ac.jp/pub/mpi/mpiprimj.ps.gz` にあります.

A Fortran 翻訳時オプション

VPP700/56 の UNIX オペレーティングシステム UXP/V 上で動作する Fortran 90/VP システム (単一 PE 版) および Fortran 90/VPP システム (複数 PE 版) の翻訳コマンド `frt` の翻訳時オプション一覧です．デフォルトとなるオプションは原則として載せていません．翻訳オプションは `frt` のあと空白をおき，ハイフン (-) に続けて英数字を指定します[117]．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．

翻訳時オプションには，ソースプログラムに直接翻訳指示行を埋め込む方法や，環境変数として設定する方法もあります (cf.[1])．

●汎用オプション

`-Wx`	Fortran 90/VPP コンパイラを起動する．富士通仕様の拡張最適化制御行が有効となり，複数 PE での並列実行可能なファイルが作成される．
`-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．
`-Cpp`	プリプロセッサを呼び出すことを指示．
`-Z` *file*	指定したファイル名 *file* に翻訳時情報を出力する．
`-Free`	プログラムが自由形式であることを指定．ファイルのサフィックス (拡張子) が `.f90` の場合は標準で有効となる．
`-Fixed`	プログラムが固定形式であることを指定．ファイルのサフィックス (拡張子) が `.f` または `.F` の場合は標準で有効となる．
`-w`	プログラムが固定形式であり，1 行の長さを 255 文字に拡張することを指定．
`-L` *dir*	ld がライブラリを検索するディレクトリのリストに *dir* を追加．
`-l` *name*	ライブラリ `lib`*name*`.a` を検索する．ライブラリはコマンド行に他のライブラリやオブジェクトファイルが現れる順に検索されるため，オプションの位置に注意．
`-o` *file*	指定したファイル名 *file* で実行ファイルを作成．省略した場合のファイル名は `a.out`．
`-I` *dir*	INCLUDE ファイルまたはモジュールの翻訳情報を検索するディレクトリ名 *dir* を指定．
`-sc`	スカラー翻訳を指示する．スカラー翻訳され，ベクトルプロセスで実行可能な実行ファイルが生成される．`-g` オプションの標準値に影響を与える．
`-Wc`	Analyzer のカウンタ機能のための翻訳を行う．カウンタ用の実行可能ファイルと翻訳情報ファイルを生成する．
`-g`*level*	シンボリック・デバッガ (fdb) の使用に必要なデバッグ情報を生成． *level* を省略すると標準値が有効となる．ただし，`-sc` の指定によって標準値が異なる (*level* の詳細は [1] 参照)．fdb は Fortran 90 プログラム， C， C++ 言語プログラムのシンボリック・デバッガ．機能は kyu-vpp の `man fdb` 参照．
`-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．
`-Dl`	Fortran 手続きごとに利用者定義のサブルーチンを呼び出すことを指示．

●言語仕様

Fortran の言語仕様およびバージョンに関するオプションです[118]．

[117]「コマンドオプション形式」といいます．

[118]FORTRAN IV および富士通仕様は今後の動作が保証できませんので，早急な移行をお勧めします．

-v	プログラムの言語仕様が富士通 FORTRAN IV であることを指定．-Xf6 と同じ．
-X6	プログラムの言語仕様が FORTRAN IV であることを指定．
-X7	プログラムの言語仕様が FORTRAN 77 であることを指定．ファイルのサフィックス (拡張子) が .f または .F の場合は標準で有効となる．
-X9	プログラムの言語仕様が Fortran 90 であることを指定．ファイルのサフィックス (拡張子) が .f90 の場合は標準で有効となる．
-Xf6	プログラムの言語仕様が富士通 FORTRAN IV であることを指定．-v と同じ．
-Xf7	プログラムの言語仕様が富士通 FORTRAN 77 であることを指定．
-v9	Fortran 90 規格に準拠しているか検査する．-v と同時に指定できない．
-vd	Fortran 90 規格の廃止事項，廃止予定事項を検査する．-v と同時に指定できない．
-V	バージョン情報を標準エラー出力 (stderr) に出力する．

●デバッグ機能

-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．実行モードを「デバッグモード」に切替える -Wv,-ad とあわせて指定します．検査は実行時に行われます．また，デバッグ用のオブジェクトを出力するため，DO ループがベクトル化されなくなる場合があります．従ってデバッグオプションを指定する場合は，プログラムの規模を小さくしておく[119]ことをお勧めします．

複数のデバックオプションを指定する場合は -Das，-Dasux などと記述できます．

-Da	仮引数と実引数の矛盾を検査する．
-Ds	添字式及び部分列式の値を検査．翻訳時にもチェックを行う．
-Du	未定義データの引用を検査．
-Dx	-Ds, -Du 指定時にベクトル化をできるだけ促進する．

●翻訳時リストの出力

-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．
-Pl	DO ループ，IF 文，CASE 文の入れ子の深さを表示したプログラムリストを出力．
-Pi	DO ループ，IF 文，CASE 文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．
-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．
-Pf	ベクトル化情報，最適化情報を含めたプログラムリストを出力．
-Pa	配列演算に対するベクトル化情報を含めたプログラムリストを出力．
-Po	翻訳時に有効となったオプション一覧を出力する．
-Pt	翻訳の統計情報を出力．
-Px*num*	以下の *num* に応じてリストを出力する．相互参照プログラム (指定なし)，ヘッダ (1)，名前の情報 (2)，共通ブロックのマップ (3)，文番号および構文名のマップ (4)，ライブラリ呼び出しのマップ (5)，モジュール情報のマップ (6)，構造型のマップ (7)．

●翻訳メッセージの出力

-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．s レベルのエラー (重度のエラー) は無条件で出力されます．

-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．

●診断メッセージ

-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．

-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．
-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．
-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．
-Ei	外部手続きの引用箇所への展開 (インライン展開) 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 (ループアンローリング) に関するメッセージを出力．

[119]分割幅を大きめに取るなどして，実行時間が少なくて済むように工夫することです．

●精度拡張，記憶域合わせ

-A で始まるオプションは，精度拡張，記憶域合わせなどの機能を指示します．

-AA	共通ブロック内データを境界調整しないで割り付ける．
-AB	すべての仮引数を位置取りにする機能を抑止する．
-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．
-Ap	単精度を倍精度に，倍精度を 4 倍精度に精度拡張し，データの記憶域合わせを行う．
-Ap4	単精度を倍精度に精度拡張し，データの記憶域合わせを行う．
-Ap8	倍精度を 4 倍精度に精度拡張し，データの記憶域合わせを行う．
-Ai	4 バイト整数型の長さを 2 バイトとする．-sc オプションが指定されたときのみ有効．
-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．
-AE	文字列中に特殊文字が存在しても，特殊文字として扱わない．
-AR	定数やデバッグ機能で生成される名前情報を書き換えできない領域に割り付ける．
-AT	暗黙の型宣言を抑止する． IMPLICIT 文がある場合はこちらが優先する．
-AU	英語の大文字／小文字を区別することを指示．ただし， JIS Fortran 90 規格違反．
-Aw	関数名 IACHAR, ACHAR, IBITS, ISHFTC を組み込み手続きとすることを指示．
-Ay	代入文の右辺の符合付き実定数を，左辺の変数の型に拡張することを指示．
-Az	外部手続き呼び出しの実引数に現れた文字定数の末尾に\0(ヌル文字) を追加することを指示．
-Ao	!FORTRAN 行および !OPTIONS 行を翻訳指示行とすることを指示．指定がない場合は注釈行として解釈．

●最適化機能

-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。翻訳時間は一般に

$$\texttt{-On} \leq \texttt{-Ob} \leq \texttt{-Oe} \leq \texttt{-Of}$$

となりますので，プログラムの開発状況に応じてうまく使い分けることをお勧めします．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	最適化を行わない．翻訳時間は最小なので，エラーのチェックなどのプログラミング初期に有効．
-Ob	基本的な最適化を行う．
-Oe	-Ob に加えて，ループアンローリングの最適化，不変式の先行評価，演算評価方法の最適化ををを行う (省略値)．
-Of	-Oe に加えて，多重ループの構成変更，ループアンローリングの強化，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．
,-x	演算評価方法を変更する最適化機能を更に強化する．
,-X	演算評価方法を変更する最適化機能を更に強化しない．
,-r	LIW 型 RISC 命令の最適化を行う．
,-R	LIW 型 RISC 命令の最適化を抑止する．
,-l	多重ループの構成変更の最適化を行う．
,-L	多重ループの構成変更の最適化を抑止する．
,-g	広域命令スケジューリングの最適化を行う．
,-G	広域命令スケジューリングの最適化を抑止する．

●インライン展開

-N で始まるオプションは，利用者定義の外部手続きの引用箇所への展開 (インライン展開) を指示するオプションです．ただし，-On が指定された場合は無視されます．翻訳時間は大幅に増加しますが，(プログラムによっては) 実行時間の大幅な短縮が期待できます．

-Ne	実行文の数が 30 以下の利用者定義の外部手続きを引用箇所に展開.
-Ne,*pgm*	手続き名 *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*	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*	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.
-No,*files*	ファイル *files* にある利用者定義の外部手続きをその引用箇所にインライン展開する. *files* はサフィックスが .f または .f90 の Fortran プログラム. 複数指定する場合はカンマで区切って続ける.

●実行性能向上

-K で始まるオプションは，オブジェクトプログラムの実行性能向上を追求するオプションです．ただし，-On が指定された場合は無視されます.

-Kauto	局所変数を，手続きの終了時に不定になるものとして扱う.
-Kfast	翻訳しているマシンに適したオブジェクトファイルを出力する.
-KVPP700	VPP700 に適したオブジェクトファイルを出力する.
-Karraystack1	自動割付け配列による動的作業領域をスタックに割付ける.
-Karraystack2	自動割付け配列以外の動的作業領域をスタックに割付ける.
-Karraystack3	-Karraystack1, -Karraystack2 の機能を同時に行う.

●型の変換

-C で始まるオプションは，個別の型を変換するオプションです．組み合わせて指定する場合は，カンマで区切ります.

-CcL2L1	2 バイトの論理型を 1 バイトの論理型とすることを指示. -CcL2L4 と排他.
-CcL2L4	2 バイトの論理型を 4 バイトの論理型とすることを指示. -CcL2L1 と排他.
-CcdII8	基本整数型を 8 バイトの整数型とすることを指示. 変数，定数，名前付き定数および関数に適用される.
-CcI4I8	すべての 4 バイトの整数型を 8 バイトの整数型とすることを指示. 変数，定数，名前付き定数および関数に適用される.
-CcdLL8	基本論理型を 8 バイトの論理型とすることを指示. 変数，定数，名前付き定数および関数に適用される.
-CcL4L8	すべての 4 バイトの論理型を 8 バイトの論理型とすることを指示. 変数，定数，名前付き定数および関数に適用される.
-CcdRR8	基本実数型および基本複素数型を倍精度実数型および倍精度複素数型とすることを指示. 変数，定数，名前付き定数および関数に適用される.
-CcR4R8	すべての実数型および複素数型を倍精度実数型および倍精度複素数型とすることを指示. 変数，定数，名前付き定数および関数に適用される. -Ad と同等の機能.
-Ccd4d8	-CcdII8,cdLL8,cdRR8 と等価.
-Cca4a8	-CcI4I8,cL4L8,cR4R8 と等価.
-CcdDR16	基本倍精度実数型および基本倍精度複素数型を 4 倍精度実数型および 4 倍精度複素数型とすることを指示. 変数，定数，名前付き定数および関数に適用される.
-CcR8R16	すべての倍精度実数型および倍精度複素数型を 4 倍精度実数型および 4 倍精度複素数型とすることを指示. 変数，定数，名前付き定数および関数に適用される. -Aq と同等の機能.

●モジュール処理

以下は，モジュール処理に関するオプションです.

-Am	プログラム中のモジュールの翻訳情報をカレントディレクトリに登録する. プログラム中に存在しないモジュールをカレントディレクトリから引用したい場合にも指定. -Am が指定された場合，-M *directory* および -As は無効となる.
-M *directory*	プログラム中のモジュールの翻訳情報を *directory* に登録する. プログラム中に存在しないモジュールを *directory* から引用したい場合にも指定.
-As	モジュールの翻訳情報を格納したファイルを英大文字で作成する.

●ベクトルオプション

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.

○実行モードの切替

-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．省略した場合，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．

-Wv,-an	式の評価順序の変更を抑止した高速モードのオブジェクトを出力．最適化レベルは -Ob となる．
-Wv,-ae	式の評価順序を変更したデバッグモードのオブジェクトを出力．最適化レベルは -Oe となる．
-Wv,-ad	式の評価順序の変更を抑止したデバッグモードのオブジェクトを出力．最適化レベルは -Ob となる．

○ベクトル化に関するメッセージの出力

-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,-ms	-m2 と同じ．
-Wv,-md	-m3 と同じ．

○ベクトル化と最適化制御行

以下は，ベクトル化と最適化制御行[120]についてのオプションです．

-Wv,-sc	ベクトル化を行わないことを指示．すべての DO 構文のベクトル化が抑止される．
-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,-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．

○ DO ループのベクトル化の指示

以下は， DO ループにベクトル化できない部分がある場合の指示と IF 文を含む DO ループのベクトル化方式を選択するオプションです．

-Wv,-qs	DO ループ中にベクトル化できない部分がある場合ループ全体をベクトル化しない．
-Wv,-qm	DO ループ中にベクトル化できない部分がある場合コンパイラの判断にまかせる．
-Wv,-qv	DO ループ中にベクトル化可能な部分をすべてベクトル化する．翻訳時間は増加する．また，実行時間が増加する可能性もある．
-Wv,-Im	IF 文をマスク方式でベクトル化する．
-Wv,-Il	IF 文をリスト方式でベクトル化する．
-Wv,-Ic	IF 文を収集拡散方式でベクトル化する．
-Wv,-Ge	ベクトル化された DO ループ内の配列要素のアクセスが外側の DO ループの繰返しによっても不変な場合，コンパイラの判断によるベクトルレジスタのグローバル化を行う．
-Wv,-Gf	ベクトル化された DO ループ内の配列要素のアクセスが外側の DO ループの繰返しによっても不変な場合，ベクトルレジスタのグローバル化可能な要素をすべてベクトル化する．

○組み込み関数に関するオプション

-Wv,-Of	高速な組込み関数の使用を指示する．ただし標準の組み込み関数に比べ，精度が劣化するので注意が必要．
-Wv,-Op	引数の範囲を限定した組込み関数を使用する．
-Wv,-ilfunc	ベクトル化された組込み関数のインライン展開を指示する．オブジェクトプログラム，翻訳時間は増大する．また，引数のチェックも行わないため，デバッグが完了したプログラムに対し指定すること．

[120]実行効率をあげるための情報を注釈行の形でプログラムに埋め込んだ行．

○その他のベクトルオプション

-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 〜 2147483 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,-tE	ベクトル化のための作業領域を縮小する．翻訳時間は短縮するが，ベクトル化が低下することがある.
-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,-v	すべてのループに関しベクトル化を阻害するデータ依存関係がないことを指示．誤って指定すると実行結果は保証されない.
-Wv,-Pn	命令スケジューリングの範囲を指示する． nは 1 から 255 の整数.
-Wv,-align7	配列データの先頭の境界調整値を UXP/M VPP FORTRAN77 EX/VP に合わせる.
-Wv,-noalias	ポインタ変数のベクトル化を促進することを指示．ポインタ変数が他のポインタ変数やターゲット変数と同じ領域を参照している場合誤った結果となる可能性がある.
-Wv,-fusion	繰り返し回数が同じループの融合を指示する．実行性能が低下する場合もある.
-Wv,-preload	ベクトル化されたループの外側にループがある場合，ベクトルデータの参照を先行実行することを指示．実行結果に副作用が生じることがある.

●その他の翻訳時オプション

-Wl,-dy	結合編集時，動的なリンクを行うことを指示.
-Wp,-D $name$[=$tokens$]	$name$を #define 前処理と同様に指定された $tokens$と関連づける.
-Wp,-U $name$	#undef 前処理と同様に $name$の定義を無効にする.
-W$tool$,$arguments$	各ツール別にオプションを指定する (詳細は ([1])).

A.1 排他的なオプション

以下のオプションは互いに排他的な関係です．排他的なオプションが指定された場合，右のオプションは無効となります.

有効となるオプション	無効となるオプション
-Wv,-an	-Da, -Ds, -Du
-Wv,-n	-Wv,-s -Wv,-rl -Wv,-Ov -Wv,-v
-sc	-Dl

A.2 Fortran 90/VPP で指定できないオプション

-Wx オプションを指定して Fortran 90/VPP コンパイラを起動した場合，以下のオプションは同時に指定できません.

指定できないオプション	機能の概要
-g	デバッガ用の情報出力
-v, -Xf6, -X6	FORTRAN IV の言語仕様
-Ai, -Ap, -Ap4, -Ap8	精度拡張
-AU	英大文字／英小文字を区別
-Da, -Ds, -Du	デバッグ用のオブジェクト出力
-v9, -vd	Fortran 90 規格の検査
-Free (注意)	自由形式プログラムの翻訳

(注意) ファイルのサフィックスが .f または .F の場合もしくは -X7 または -Xf7 オプションを指定した場合.

A.3 翻訳指示行として指定できるオプション

MSP から VPP700/56 にジョブを投入する場合には，翻訳指示行にオプションを記述する形式が可能です (cf.4章)．翻訳指示行として指定できるオプションは以下の通りです．機能は翻訳オプションの項を参照下さい．なお，これ以外の翻訳時オプション，および実行時オプションは指定できません．

●言語レベル，精度拡張

コマンドオプション	翻訳指示行
-v	LANGLVL(66)
-X6	LANGLVL(66)
-X7	LANGLVL(77)
-X9	LANGLVL(90)
-Xf6	LANGLVL(66)
-Xf7	LANGLVL(77)
-AA	NOALIGNC
-AB	NOBYNAME
-Ad	DOUBLE
-Aq	QUAD
-Ai	IHALF
-A0	ERRSSN(0)
-A1	ERRSSN(1)
-An	NOERRSSN
-Ad	AUTODBL(DBL4)
-Aq	AUTODBL(DBL8)
-Ap	AUTODBL(DBLPAD)
-Ap4	AUTODBL(DBLPAD4)
-Ap8	AUTODBL(DBLPAD8)

●デバッグ，最適化

コマンドオプション	翻訳指示行
-Da	DEBUG(ARGCHK)
-Ds	DEBUG(SUBCHK)
-Du	DEBUG(UNDEF)
-Ec	CONDINF
-Eeipu	OPTMSG
-On	NOOPT
-Ob	OPT(B)
-Oe	OPT(E)
-Of	OPT(F)
,-p	XOPT(PREEX)
,-P	XOPT(NOPREEX)
,-u	XOPT(UNROLL)
,-U	XOPT(NOUNROLL)
,-e	XOPT(EVAL)
,-E	XOPT(NOEVAL)

●インライン展開，出力制御

コマンドオプション	翻訳指示行
-Ne	INLINE(EXT)
-Ne,*pgm*	INLINE(EXT(*pgm*))
-Ne,*stno*	INLINE(EXT(*stno*))
-Ne,*dsizK*	INLINE(EXT(*dsizK*))
-Ps	SOURCE または S
-Pl	SOURCE(NEST)
-Pi	SOURCE(INDENT)
-Pd	SOURCE(INCLUDE)
-Pf	SOURCE(FORM)
-Pa	SOURCE(AMARK)
-Pt	STATIS
-Px	XREF
-Px1	XREF(HEADER)
-Px2	XREF(NAME)
-Px3	XREF(COMMON)
-Px4	XREF(LABEL)
-Px5	XREF(LIB)
-Px6	XREF(MODULE)
-Px7	XREF(DERIVED)
-Sw	FLAG(W)
-Ss	FLAG(S)
-Y*n*	PR(*n*)

●ベクトル化に関するオプション

コマンドオプション	翻訳指示行
-Wv,-an	ADVANCED(NOEVL)
-Wv,-ae	NOADVANCED(EVL)
-Wv,-ad	NOADVANCED(NOEVL)
-Wv,-n	NOOCL
-Wv,-sc	SCALAR
-Wv,s*vl*	SCALAR(*vl*)
-Wv,te	VPXTBL
-Wv,-r*l*	REPEAT(*l*)
-Wv,-m1	VMSG(1)
-Wv,-m2	VMSG(2)
-Wv,-m3	VMSG(3)
-Wv,-ms	VMSG
-Wv,-md	VMSG(DETAIL)
-Wv,-Of	XOPT(HSFUN)
-Wv,-Ov	VDOPT
-Wv,-v	NOVREC

B Fortran 実行時オプション

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．ただし， Fortran 90/VPP では `core` は出力しない.
`-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． *num2* は 1 から 32767 までの整数． Fortran 90/VPP ではプロセッサのどれかがこの回数に達すると実行を打ち切る.
`-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 32．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,-li`	実行時のすべてのメッセージを出力 (省略値).
`-Wl,-lw`	i レベルのメッセージを抑止.
`-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.
`-Wl,-m`*uno*	診断メッセージを出力するファイル (標準エラー出力) 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`	READ 文で標準入力からデータを入力するときに入力を促すメッセージを出力.
`-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,-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． *sec* は 1 から 9999 までの整数.
`-Wl,-u`	指数アンダーフローを検出する．`-i` と同時に指定できない． 実行時間は増大する.
`-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,-q`	書式付き出力文における E,EN,ES,D,Q,G,L, および Z 編集の出力文字および 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,-C`*uno*	装置番号 *uno* からのバイナリーデータの入出力を M 形式 (IBM 形式) で行う．指定がない場合は IEEE 形式のデータとして処理される． *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.
`-Wl,-G`*uno*	装置番号 *uno* からの書式なし入出力文において， M 形式の文字データ (EBCDIC 文字コード) を ASCII 文字コードに変換して読み込む．指定がない場合は ASCII 文字コードのまま入出力が行われる． *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,-Lb`	サービスルーチンの 4 バイトの論理型または論理型の引数および復帰値を 8 バイトの論理型として扱う.
`-Wl,-Li`	サービスルーチンの 4 バイトの整数型または整数型の引数および復帰値を 8 バイトの整数型として扱う.
`-Wl,-Lr`	サービスルーチンの 4 バイトの実数型または実数型の引数および復帰値を 8 バイトの実数型として扱う.
`-Wl,-Ls`	DATE, SIGNAL, TIME サービスサブルーチンの呼び出し形式を制御.
`-Wl,-Oi`	入出力統計情報を出力する.
`-Wl,-Op`	実行時に有効となったオプションの一覧を出力する.

実行時オプション (続き).

`-Wl,-Q`	書式付き入力文の一つの Fortran 記録を構成する欄の幅が入力される Fortran 記録の長さより長い場合, Fortran 記録の後ろに入力欄の幅が必要とする空白を詰めない.
`-Wl,-Re`	診断メッセージ, トレースバックマップ, エラー集計情報の出力, ERRSET/ERRSTR サービスサブルーチンによるエラー項目の修正, 利用者修正出口処理ルーチンの実行を抑止する. エラーが発生した場合, Fortran システムの標準修正が行なわれ, 実行が継続される.
`-Wl,-Vp`	実行時に必要となった動的作業領域に関する情報を出力する.
`-Wl,-Vs`*dvt*	動的ベクトル作業領域を確保する場合, 初期値および増分値の大きさを指定する. *dvt* は 1 キロバイト単位で, 省略値は 50 キロバイト.

以下は, Fortran 90/VPP でのみ有効な実行時オプションです.

`-Wl,-Pm` *num1*	*num1* 要素以下のグローバル関数をバタフライ演算方式で実行. 省略値は 8.
`-Wl,-Pg` *num2*	グローバル関数が使用する作業領域を *num2* キロバイトに設定. 省略値は 1. グローバル関数で要素数が大きい場合に効果がある.

C　C 翻訳時オプション

VPP700/56 の UNIX オペレーティングシステム UXP/V 上で動作する C 言語システム (スカラー版)， C/VP システム (ベクトル版) および C++ システムの翻訳時オプション一覧です．翻訳コマンドはそれぞれ cc, vcc, CC です．オプションはコマンドのあと空白をおき，ハイフン (-) または + に続けて英数字を指定します．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．大文字／小文字は区別してください．

C.1　共通オプション

cc，vcc，CC 共通のオプションです．

●処理制御

-c	リンク・エディットを行わないことを指定.
-o *pathname*	省略時の a.out の代わりに， *pathmane* の名前でオブジェクトファイルを出力.
-Kfconst	接尾辞が指定されていない浮動小数点定数を float 型として扱う.
-Knoconst	const 型修飾子が指定されたデータを定数として取り扱わない.
-W*tool*,*arg*	指定された *tool* に *arg* をそれぞれ別の引数として渡す．引数はカンマで区切って複数指定できる．カンマ自身は直前にバックスラッシュを置く． *tool* に指定できる文字は p(プリプロセッサ), 0(コンパイラ), v(ベクトライザ (C/VP)), a(アセンブラ), l(リンク・エディタ).

●前処理機能

-A *name*[(*tokens*)]	*name* を #assert 前処理と同様に指定された *tokens* と関連づける．既定のアサーションは machine(fujitsu)， system(unix)， system(uxppx)， cpu(fujitsu)， system(uxpv)， system(uxp).
-A -	既定のマクロ (__ で始まるものを除く) と既定のアサーションを無効にする.
-D *name*[=*tokens*]	*name* を #define 前処理と同様に指定された *tokens* と関連づける．[=*tokens*] の指定がない場合はトークン 1 が与えられる．既定のマクロは __uxp__, __uxppx__, __uxpv__, unix (vcc の場合は更に__uxpvp__).
-U *name*	#undef 前処理と同様に *name* の定義を無効にする．-U と -D オプションに同一の *name* が指定された場合は，その順序によらず *name* は定義されない.
-C	前処理の段階で通常削除される注釈を，前処理の行にあるもの以外削除しない.
-E	C ソースファイルの前処理のみを実行し，結果を標準出力に送る．出力は次の過程で使用される前処理命令を含む.
-P	C ソースファイルの前処理のみを実行し，結果をサフィックス .i に出力．出力は前処理命令を含まない.

●言語仕様，出力制御

-Xa	C 言語の仕様が ANSI C の新しい仕様をすべて含む (省略値).
-Xc	C 言語の仕様と関連するヘッダファイルが，より厳密に ANSI C 準拠のものとなる.
-Xt	C 言語仕様が， ANSI C 以前の従来仕様に互換性のある ANSI C 仕様を追加したものとして翻訳を行なう.
-w	警告メッセージの出力を抑止する.
-H	翻訳中にインクルードされるファイルのパス名を， 1 行ごとに標準エラー出力に送る.
-g	fdb([11]) によるデバッグ情報を出力する．vcc の場合更にレベル -g0--g3 が指定できる．省略値は -g3，ただし -K novp が指定された場合は -g0．cc の場合レベルの指定はできない．また -O と排他.

●最適化機能

最適化機能を指示するオプションです．-K オプションは，カンマで区切ることで -Klib,NOARG,popt など，複数の引数の指定が可能です．

-O	オブジェクトプログラムに対する最適化を行う．最適化レベルは K1 〜K4 で指定 (省略値は K2)．この最適化は .s の付いたファイルには無効.
-K1	プログラムの流れを詳細に解析して最適化を行なう．-O と同時に指定されたときのみ有効.
-K2	-K1 に加え，ループアンローリング，さらに vcc コマンドでは演算順序の変更，除算を乗算化する最適化を行なう．オブジェクトが増加する場合がある．-O と同時に指定されたときのみ有効.
-K3	-K2 に加え，最適化機能を繰り返し実施する．翻訳時間は増加する．-O と同時に指定されたときのみ有効.
-K4	-K3 に加え，演算の評価方法を変更 (-Keopt と等価)，lib 関数の展開／変更の最適化 (-Klib と等価) を行なう．これらの最適化は，精度誤差または実行時の異常終了の副作用を生じる可能性があるため注意が必要．-O と同時に指定されたときのみ有効.
-Keopt	演算の評価方法を変更する最適化を行う．精度誤差または実行時の異常終了の副作用を生じる可能性があるため注意が必要．-O と同時に指定されたときのみ有効.
-Klib	標準関数の動作を認識し，インライン展開および，より高速な関数への置換えを行なう．-O と同時に指定されたときのみ有効.
-Kan	外部変数および静的変数の最低の割り付け境界を指定． n は 1,2,4 のいずれか．省略値は 1.
-KJMP	すべての関数呼び出しが setjmp 関数であると想定して翻訳する.
-Knounroll	ループアンローリングの最適化を抑止する.
-Kpopt	異なるポインタが同一の領域を指していないと解釈し，ポインタにより示されるデータの最適化を行う．-O と同時に指定されたときのみ有効.
-KNOARG	関数呼び出しのための引数を，レジスタあるいはスタックに設定する．関数原型宣言のある可変個引数関数呼び出しを含むプログラムを翻訳する場合に指定.
-Kdiv	算術変換後の型が int 型または long int 型であるオペランドの除算を浮動小数点型に変換する． -O と同時に指定されたときのみ有効.
-xn	関数内の初期化のある宣言と，実行文の数が n 以下の関数をインライン展開する． n は 4 バイトで表現可能な正の整数．-O と同時に指定されたときのみ有効.
-x *func*	C プログラムで定義された関数 *func* に対しインライン展開を指示．関数はカンマで区切り複数指定可能．再帰的呼び出し関数のインライン展開は行わない．-O と同時に指定されたときのみ有効.
-x -	C プログラムで定義されたすべての関数に対しインライン展開を指示．マクロ展開後の初期化のある宣言，実行文の数が 30 を越える関数，再帰的呼び出し関数のインライン展開は行わない．-O と同時に指定されたときのみ有効.

●ディレクトリの指定

-L *dir*	ld がライブラリを検索するディレクトリのリストに *dir* を追加.
-l *name*	ライブラリ lib*name*.a を (cc コマンドの場合は lib*name*.so も) 検索する．ライブラリはコマンド行に他のライブラリやオブジェクトファイルが現れる順に検索されるため，オプションの位置に注意.
-I *dir*	名前が / 以外で始まるインクルードファイルの検索を， *dir* で指定されたディレクトリから先に検索する．'' で囲まれたファイルは，#inlude 前処理を含む現ディレクトリ，-I で指定されたディレクトリ，標準ディレクトリの順で検索する．< > で囲まれたファイルは，-I で指定されたディレクトリ，標準ディレクトリの順で検索する．インクルードファイルが絶対パス名で指定された場合は，そのパス名のみを検索する.
-Yp,*dir*	プリプロセッサのパス名を vcc では *dir*/acompvp に，cc では *dir*/acpp 変更.
-YO,*dir*	コンパイラのパス名を vcc では *dir*/acompvp に，cc では *dir*/ccom に変更.
-Ya,*dir*	アセンブラのパス名を *dir*/as に変更.
-Yl,*dir*	リンク・エディタのパス名を *dir*/ld に変更.
-YI,*dir*	ヘッダファイル検索のディレクトリを *dir* に変更.
-YP,*dir*	結合するライブラリのディレクトリを *dir* に変更． *dir* にはセミコロンで区切ったパスリストを指定.
-YS,*dir*	結合するスタートアップ・オブジェクトのディレクトリを *dir* に変更.

C.2 cc コマンドのみのオプション

以下は cc コマンドでのみ有効なオプションです.

-S	C プログラムを翻訳し，アセンブラ言語出力をサフィックス .s の付いたファイルに出力.
-K0	オブジェクトの最適化を行わない．-O オプションを指定しない場合と等価．-O と同時に指定されたときのみ有効.
-Bstatic	-l*name* に指定したライブラリを lib*name*.a だけ検索する． 省略した場合 lib*name*.so， lib*name*.a の順に検索する.
-dn	ライブラリを静的にリンクする．省略時は動的にリンクする.
-G	共用オブジェクトを作成する．-dn とは排他.
-KPIC	position-independent code を生成する．詳細は [9].
-Kpic	position-independent code を生成する．詳細は [9].
-p	関数の呼出し回数をカウントするコードを出力.
-Qn	翻訳ツールの識別情報を出力ファイルに追加しない.

C.3 vcc コマンドのみのオプション

vcc コマンドのみで使用できるベクトルオプションは通常 -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.

●実行モードの切替

-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．省略した場合，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.

-Wv,-an	式の評価順序の変更を抑止した高速モードのオブジェクトを出力.
-Wv,-ae	式の評価順序を変更したデバッグモードのオブジェクトを出力.
-Wv,-ad	式の評価順序の変更を抑止したデバッグモードのオブジェクトを出力.

●ベクトル化に関するメッセージの出力

-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,-Ps	ベクトル化表示ソースリストを出力.
-Wv,-Pt	翻訳に関する統計情報を出力.
-Z *file*	翻訳情報を指定されたファイル名 *file* に出力.

● for, while ループの指示

-Wv,-Bs	継続条件が常に真となるループおよび逆向きループがないことを指示.
-Wv,-Bf	継続条件が常に真となるループがないこと，かつ，逆向きループの可能性があることを指示．省略値.
-Wv,-Bn	継続条件が常に真となるループおよび逆向きループがあることを指示.
-Wv,-eloop	while ループの入口で回転数が確定しない場合もベクトル化する．実行時に異常終了する可能性があるので注意.
-Wv,-sloop	while ループのベクトル化を抑止.
-Wv,-vloop	while ループの入口で回転数が確定する場合のみベクトル化する (省略値).
-Wv,-fusion	ループ融合を行う.

●ベクトル化と最適化制御

ベクトル化と最適化制御行についてのオプションです.

-Wv,-sc	ベクトル化を行わないことを指示．すべての for ループのベクトル化が抑止される．
-Wv,-s*vl*	翻訳時に繰返し回数が *vl* 未満と分かる for ループのベクトル化を抑止． *vl* は 1 〜 255 の整数．
-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．
-Wv,-Im	if 文をマスク方式でベクトル化する．
-Wv,-Il	if 文をリスト方式でベクトル化する．
-Wv,-Ic	if 文を収集拡散方式でベクトル化する．
-Wv,-Mi	すべての多重ループのベクトル化を行う．
-Wv,-MI	for 文と for 文の間に実行文がない多重ループのベクトル化だけを行う (省略値)．
-Wv,-Mn	多重ループのベクトル化を抑止する．
-Wv,-preload	ベクトルデータのプレロードを行う．
-Wv,-qs	ループ中にベクトル化できない部分があった場合，そのループをベクトル化しない．
-Wv,-qm	ループ中にベクトル化できない部分があった場合，そのループをベクトル化するかどうかの判断をコンパイラに任せる (省略値)．
-Wv,-qv	ループ中のベクトル化できる部分をすべてベクトル化する．
-K preex	不変式の先行評価の最適化を行なう．
-K novp	ベクトル翻訳を抑止する．cc コマンドと同じ効果．

●その他のベクトルオプション

-Wv,-ilfunc	ベクトル関数をインライン展開する．
-Wv,-r*l*	for 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 〜 2147483647 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が出力された場合，作業領域を拡張する．
-Wv,-f	ベクトル数学関数の使用を抑止．
-Wv,-Of	高速数学関数を使用する．演算精度が低くなるので注意が必要．
-Wv,-Om	文字関数 (memset, memcpy) をより高速な関数で実行する．
-Wv,-Op	引数範囲を限定した数学関数を使用する．
-Wv,-Ov	スカラー実行とベクトル実行を動的に切り換えるオブジェクトを生成する．
-Knoline	チューニングの際にサンプラ ([11]) に渡す追加情報を生成しない．
-Knoenv	環境変数およびプロフィルファイルに指定された翻訳時オプションを無効にする．
-Knoeval	演算の評価方法を変更する最適化を抑止する．-K2 以上で有効．
-Knorecipr	除算を乗算化する最適化を抑止する．-K2 以上で有効．

C.4 排他的なオプション

以下のオプションは互いに排他的な関係です．排他的なオプションが指定された場合，右のオプションは無効となります．

有効となるオプション	無効となるオプション
-Wv,-n	-Wv,-sc -Wv,-s*vl* -Wv,-r*l*

C.5　CC コマンドのみのオプション

CC コマンドは，C++ ソースプログラムを C ソースに変換し，cc によるオブジェクト生成が行われます．従って，cc コマンドのオプションはそのまま指定できます．

以下のオプションは，翻訳処理の段階で C++ プログラムに適用されるオプションです．

`-E`	C++ ソースファイルに対し，前処理 (`cpp`) のみを行う．結果は標準出力へ返す．
`-F`	C++ ソースファイルに対し，前処理 (`cpp`) と C++ の処理 (`cfront`) を行う．結果は標準出力へ返す．出力には `#line` 指令が含まれる．
`-.`*suffix*	`-E` または `-F` の出力先を添字 *suffix* を持つファイルに出力．
`+a1`	ANSI C に従う宣言が生成される．省略した場合は K&R C スタイル．
`+d`	インライン関数を展開しない．
`+e0`	仮想関数のテーブルを `external` 宣言し，定義を行う．
`+e1`	仮想関数のテーブルを `external` 宣言するが，定義は行わない．
`+i`	翻訳処理後，中間結果の C ファイルをサフィックス `..c` に残す．前処理指令は入っていない．
`+p`	古い構文の使用を禁止する．
`+w`	誤りと思われる，移植性が損なわれる，効率が悪い構造について警告を出力する．
`+xfile`	サイズと alignment のファイルを読み込む．

なお，マニュアル [10] の以下の機能は使用できません．

- タスク・ライブラリ．
- シンボリックデバッガ sdb+．
- 以下の関数のシステムコール．
  `getrlimit(2)`, `setrlimit(2)`, `msgctl(2)`, `msgget(2)`, `msgsnd(2)`, `priocntl(2)`, `priocntlset(2)`, `semctl(2)`, `semget(2)`, `semop(2)`, `sysinfo(2)`, `uname(2)`, `ustat(2)`, `ftok(3C)`, `xdr_hyper(3N)`, `xdr_longlong_t(3N)`, `xdr_u_hyper(3N)`, `xdr_u_longlong_t(3N)`, `xdr_sprayarr(3N)`, `xdr_spraycumul(3N)`

D SSL II/VPP 機能一覧

SSL II/VPP は分散記憶型ベクトル並列計算機上で並列動作する数値計算ライブラリです．単一 PE で処理できない大規模な問題を並列処理するサブルーチン群を提供しています．精度はすべて倍精度です．

●行列演算

DP_VMGGM	実行列の積
DP_VMVSD	実スパース行列と実ベクトルの積 (対角形式格納法)
DP_VMVSE	実スパース行列と実ベクトルの積 (ELLPACK 形式格納法)

●連立 1 次方程式

DP_VLAX	実行列の連立 1 次方程式 (ブロック化された LU 分解法)
DP_VLCX	複素行列の連立 1 次方程式 (ブロック化された LU 分解法)
DP_VLSX	正値対称行列の連立 1 次方程式 (ブロック化された変形 Cholesky 分解法)
DP_VLBX	実バンド行列の連立 1 次方程式 (Gauss の消去法)
DP_VLSBX	正値対称バンド行列の連立 1 次方程式 (変形 Cholesky 分解)
DP_VCGD	正値対称スパース行列の連立 1 次方程式 (前処理付き CG 法，対角形式格納法)
DP_VCGE	正値対称スパース行列の連立 1 次方程式 (前処理付き CG 法， ELLPACK 形式格納法)
DP_VCRD	非対称または不定値スパース行列の連立 1 次方程式 (MGCR 法，対角形式格納法)
DP_VCRE	非対称または不定値スパース行列の連立 1 次方程式 (MGCR 法， ELLPACK 形式格納法)
DP_VTFQD	非対称または不定値のスパース行列の連立 1 次方程式 (TFQMR 法，対角形式格納法)
DP_VTFQE	非対称または不定値のスパース行列の連立 1 次方程式 (TFQMR 法， ELLPACK 形式格納法)
DP_VQMRD	非対称または不定値のスパース行列の連立 1 次方程式 (QMR 法，対角形式格納法)
DP_VQMRE	非対称または不定値のスパース行列の連立 1 次方程式 (QMR 法， ELLPACK 形式格納法)

●行列の三角分解

DP_VALU	実行列の LU 分解 (ブロック化された LU 分解法)
DP_VCLU	複素行列の LU 分解 (ブロック化された LU 分解法)
DP_VSLDL	正値対称行列の LDL^T 分解 (ブロック化された変形 Cholesky 分解法)
DP_VBLU	実バンド行列の LU 分解 (Gauss の消去法)
DP_VBLDL	正値対称バンド行列の LDL^T 分解 (変形 Cholesky 分解)

●三角分解された行列の連立 1 次方程式

DP_VLUX	LU 分解された実行列の連立 1 次方程式
DP_VCLUX	LU 分解された複素行列の連立 1 次方程式
DP_VLDLX	LDL^T 分解された正値対称行列の連立 1 次方程式
DP_VBLUX	LU 分解された実バンド行列の連立 1 次方程式
DP_VBLDX	LDL^T 分解された正値対称バンド行列の連立 1 次方程式

●逆行列

DP_VMINV	実行列の逆行列 (ブロック化された Gauss-Jordan 法)

●特異値分解

DP_VSVD	実行列の特異値分解 (One sided Jacobi 法)

●最小二乗解

DP_VLSQ	最小二乗解 (修正 Gram-Schmidt 法)

●固有値問題

DP_VGEVP	実対称行列の一般化固有値および固有ベクトル (One sided Jacobi 法)
DP_VSEVP	実対称行列の固有値および固有ベクトル (One sided Jacobi 法)
DP_VSEVPH	実対称行列の固有値・固有ベクトル (3 重対角化，マルチセクション法，逆反復法)
DP_VHEVP	Hermite 行列の固有値・固有ベクトル
DP_VTDEVC	実 3 重対角行列の固有値・固有ベクトル
DP_VLAND	実対称スパース行列の固有値・固有ベクトル (Lanczos 法，対角形式格納法)

● Fourier 変換

DP_V1DCFT	1 次元離散型複素 Fourier 変換 (2, 3, 5 の混合基底)
DP_V2DCFT	2 次元離散型複素 Fourier 変換 (2, 3, 5 の混合基底)
DP_V3DCFT	3 次元離散型複素 Fourier 変換 (2, 3, 5 の混合基底)
DP_V1DRFT	1 次元離散型実 Fourier 変換 (2, 3, 5 の混合基底)
DP_V1DRCF	1 次元離散型実 Fourier 変換 (2, 3, 5 の混合基底)
DP_V2DRCF	2 次元離散型実 Fourier 変換 (2, 3, 5 の混合基底)
DP_V3DRCF	3 次元離散型実 Fourier 変換 (2, 3, 5 の混合基底)

●乱数

DP_VRANU4	一様乱数 $[0, 1)$ の生成
DP_VRANN3	正規乱数の生成

●補助ルーチン

DMACH	丸め誤差の単位 (unit round off)

E　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．

●行列格納モードの変換

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

●行列操作

AGGM	行列の和 (実行列)
SGGM	行列の差 (実行列)
MGGM	行列の積 (実行列)
VMGGM	行列の積 (実行列) ［拡張機能］
MGSM	行列の積 (実行列・実対称行列)
ASSM	行列の和 (実対称行列)
SSSM	行列の差 (実対称行列)
MSSM	行列の積 (実対称行列)
MSGM	行列の積 (実対称行列・実行列)
MAV	実行列と実ベクトルの積
MCV	複素行列と複素ベクトルの積
MSV	実対称行列と実ベクトルの積
MSBV	実対称バンド行列と実ベクトルの積
MBV	実バンド行列と実ベクトルの積
VMVSD	スパース実行列と実ベクトルの積 (対角形式格納法) ［拡張機能 II］
VMVSE	スパース実行列と実ベクトルの積 (ELLPACK 形式格納法) ［拡張機能 II］

●行列の三角分解

ALU	実行列の LU 分解 (Crout 法)
VALU	実行列の LU 分解 (ブロッキング LU 分解法) ［拡張機能］
CLU	複素行列の LU 分解 (Crout 法)
SLDL	正値対称行列の LDL^T 分解 (変形 Cholesky 法)
VSLDL	正値対称行列の LDL^T 分解［拡張機能］
VBLDL	正値対称バンド行列の LDL^T 分解 (変形 Cholesky 分解) ［拡張機能 II］
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)
SBDL	正値対称バンド行列の LDL^T 分解 (変形 Cholesky 法)
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)
BLU1	実バンド行列の LU 分解 (Gauss 消去法)
VBLU	実バンド行列の LU 分解 (Gauss の消去法) ［拡張機能 II］

●三角分解された連立 1 次方程式

LUX	LU 分解された実行列の連立 1 次方程式
CLUX	LU 分解された複素行列の連立 1 次方程式
LDLX	LDL^T 分解された正値対称行列の連立 1 次方程式
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］
VBLDX	LDL^T 分解された正値対称バンド行列の連立 1 次方程式［拡張機能 II］
MDMX	MDM^T 分解された実対称行列の連立 1 次方程式
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
BMDMX	MDM^T 分解された実対称バンド行列の連立 1 次方程式
BLUX1	LU 分解された実バンド行列の連立 1 次方程式
VBLUX	LU 分解された実バンド行列の連立 1 次方程式［拡張機能 II］

●連立 1 次方程式

LAX	実行列の連立 1 次方程式 (Crout 法)
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］
LCX	複素行列の連立 1 次方程式 (Crout 法)
LSX	正値対称行列の連立 1 次方程式 (変形 Cholesky 法)
VLSX	正値対称行列の連立 1 次方程式 (変形 Cholesky 法)［拡張機能］
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)
LSBX	正値対称バンド行列の連立 1 次方程式 (変形 Cholesky 法)
VLSBX	正値対称バンド行列の連立 1 次方程式 (変形 Cholesky 分解)［拡張機能 II］
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)
LBX1	実バンド行列の連立 1 次方程式 (Gauss 消去法)
VLBX	実バンド行列の連立 1 次方程式 (Gauss の消去法)［拡張機能 II］
LSTX	正値対称 3 項行列の連立 1 次方程式 (変形 Cholesky 法)
LTX	実 3 項行列の連立 1 次方程式 (Gauss 消去法)
VLTX	実 3 項行列の連立 1 次方程式 (Cyclic reduction 法)［拡張機能］
VLTX1	定数型実 3 項行列の連立 1 次方程式［拡張機能］
VLTX2	定数型実 3 項行列の連立 1 次方程式［拡張機能］
VLTX3	定数型実 3 項行列の連立 1 次方程式［拡張機能］
VCGD	正値対称スパース行列の連立 1 次方程式 (前処理付き CG 法，対角形式格納法)［拡張機能 II］
VCGE	正値対称スパース行列の連立 1 次方程式 (前処理付き CG 法， ELLPACK 形式格納法)［拡張機能 II］
VCRD	非対称または不定値スパース実行列の連立 1 次方程式 (MGCR 法，対角形式格納法)［拡張機能 II］
VCRE	非対称または不定値スパース実行列の連立 1 次方程式 (MGCR 法， ELLPACK 形式格納法)［拡張機能 II］

●連立 1 次方程式の解の反復改良

LAXR	実行列の連立 1 次方程式の解の反復改良
LCXR	複素行列の連立 1 次方程式の解の反復改良
LSXR	正値対称行列の連立 1 次方程式の解の反復改良
LSIXR	実対称行列の連立 1 次方程式の解の反復改良
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
LBX1R	実バンド行列の連立 1 次方程式の解の反復改良

●逆行列

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

●最小二乗解

LAXL	実行列の最小二乗解 (Householder 変換)
LAXLR	実行列の最小二乗解の反復改良
LAXLM	実行列の最小二乗最小ノルム解 (特異値分解法)
GINV	実行列の一般逆行列 (特異値分解法)
ASVD1	実行列の特異値分解 (Householder 法, QR 法)

●固有値，固有ベクトル

EIG1	実行列の固有値及び固有ベクトル (2 段 QR 法)
CEIG2	複素行列の固有値及び固有ベクトル (QR 法)
SEIG1	実対称行列の固有値及び固有ベクトル (QL 法)
SEIG2	実対称行列の固有値及び固有ベクトル (Bisection 法, 逆反復法)
HEIG2	Hermite 行列の固有値及び固有ベクトル (Bisection 法, 逆反復法)
BSEG	実対称バンド行列の固有値及び固有ベクトル (Lutishauser-Schwarz 法， Bisection 法，逆反復法)
BSEGJ	実対称バンド行列の固有値及び固有ベクトル (Jennings 法)
TEIG1	実対称 3 重対角行列の固有値及び固有ベクトル (QL 法)
TEIG2	実対称 3 重対角行列の固有値及び固有ベクトル (Bisection 法, 逆反復法)
GSEG2	実対称行列の一般固有値及び固有ベクトル (Bisection 法, 逆反復法)
GBSEG	実対称バンド行列の一般固有値及び固有ベクトル (Jennings 法)
VSEG2	実対称行列の固有値・固有ベクトル (並列 Bisection 法，逆反復法) ［拡張機能］
VGSG2	実対称行列の一般固有値・固有ベクトル (並列 Bisection 法，逆反復法) ［拡張機能］

●固有値

HSQR	実 Hessenberg 行列の固有値 (2 段 QR 法)
CHSQR	複素 Hessenberg 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 (Bisection 法)

●固有ベクトル

HVEC	実 Hessenberg 行列の固有ベクトル (逆反復法)
CHVEC	複素 Hessenberg 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

BLNC	実行列の平衡化
CBLNC	複素行列の平衡化
HES1	実行列の実 Hessenberg 行列への変換 (Householder 法)
CHES2	複素行列の複素 Hessenberg 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 (Householder 法)
TRIDH	Hermite 行列の実対称 3 重対角行列への変換 (Householder 法)
BTRID	実対称バンド行列の実対称 3 重対角行列への変換 (Lutishauser-Schwarz 法)
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	Hermite 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

RQDR	実係数 2 次方程式
CQDR	複素係数 2 次方程式
LOWP	実係数低次代数方程式 (5 次以下)
RJETR	実係数高次代数方程式 (Jenkins-Traub の方法)
CJART	複素係数高次代数方程式 (Jarratt 法)
TSD1	実超越方程式 $f(x) = 0$ (Brent 法)
TSDM	実超越方程式 $f(x) = 0$(Muller 法)
CTSDM	複素超越方程式 $f(z) = 0$(Muller 法)
NOLBR	連立非線形方程式 (Brent 法)

●補間

AKLAG	Aitken-Lagrange 補間
AKHER	Aitken-Hermite 補間
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
BIF4	B-spline 補間式 (IV) による補間, 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
AKMID	2 次元準 Hermite 補間式による補間
INSPL	3 次 spline 補間式
AKMIN	準 Hermite 補間式
BIC1	B-spline 補間式 (I)
BIC2	B-spline 補間式 (II)
BIC3	B-spline 補間式 (III)
BIC4	B-spline 補間式 (IV)
BICD1	B-spline2 次元補間式 (I-I)
BICD3	B-spline2 次元補間式 (III-III)

●極値問題

LMINF	1 変数関数の極小化 (微係数不要, 2 次補間法)
LMING	1 変数関数の極小化 (微係数要, 3 次補間法)
MINF1	多変数関数の極小化 (微係数不要, 改訂準 Newton 法)
MING1	多変数関数の極小化 (微係数要, 準 Newton 法)
NOLF1	関数二乗和の極小化 (微係数不要, 改訂 Marquardt 法)
NOLG1	関数二乗和の極小化 (微係数要, 改訂 Marquardt 法)
LPRS1	線形計画問題 (改訂 simplex 法)
NLPG1	非線形計画問題 (微係数要, Powell 法)

●平滑化

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

●級数

FCOSF	偶関数の cosine 級数展開 (関数入力, 高速 cosine 変換)
ECOSP	cosine 級数の求和
FSINF	奇関数の sine 級数展開 (関数入力, 高速 sine 変換)
ESINP	sine 級数の求和
FCHEB	実関数の Chebyshev 級数展開 (関数入力, 高速 cosine 変換)
ECHEB	Chebyshev 級数の求和
GCHEB	Chebyshev 級数の導関数, 数値微分
ICHEB	Chebyshev 級数の不定積分

●変換

FCOST	離散型 cosine 変換 (台形公式, 2 基底 FFT)
FCOSM	離散型 cosine 変換 (中点公式, 2 基底 FFT)
FSINT	離散型 sine 変換 (台形公式, 2 基底 FFT)
FSINM	離散型 sine 変換 (中点公式, 2 基底 FFT)
RFT	離散型実 Fourier 変換
CFTM	多次元離散型複素 Fourier 変換 (混合基底 FFT)
CFT	多次元離散型複素 Fourier 変換 (8, 2 基底 FFT)
CFTN	離散型複素 Fourier 変換 (8, 2 基底 FFT, 逆順出力)
CFTR	離散型複素 Fourier 変換 (8, 2 基底 FFT, 逆順入力)
PNR	ビット逆転によるデータの置換
LAPS1	Laplace 変換 (複素右半平面で正則な有理関数)
LAPS2	Laplace 変換 (一般の有理関数)
LAPS3	Laplace 変換 (一般関数)
HRWIZ	Hurwitz 多項式の判定
VCOS1	離散型 cosine 変換 (2 基底 FFT) ［拡張機能］
VSIN1	離散型 sine 変換 (2 基底 FFT) ［拡張機能］
VRFT1	離散型実 Fourier 変換 (性能優先型, 2 基底 FFT) ［拡張機能］
VRFT2	離散型実 Fourier 変換 (メモリー節約型, 2 基底 FFT) ［拡張機能］
VCFT1	離散型複素 Fourier 変換 (性能優先型, 2 基底 FFT) ［拡張機能］
VCFT2	離散型複素 Fourier 変換 (メモリー節約型, 2 基底 FFT) ［拡張機能］
VCPF3	3 次元素因子離散型複素 Fourier 変換［拡張機能 II］
VMCFT	1 次元，多重，多次元離散複素 Fourier 変換 (混合基底) ［拡張機能 II］
VRPF3	3 次元素因子離散型実 Fourier 変換［拡張機能 II］
VMRFT	多重・多次元離散型実 Fourier 変換 (2,3 および 5 の混合基底) ［拡張機能 II］
VSRFT	1 次元・多重離散型実 Fourier 変換 (2,3 および 5 の混合基底) ［拡張機能 II］

●疑似乱数

RANU2	一様乱数 $(0, 1)$ の生成
RANU3	一様乱数 $(0, 1)$ の生成 (シャフル型)
DVRAU4	一様乱数 $[0, 1)$ の生成 (倍精度) ［拡張機能 II］
RANN1	正規乱数の生成 (高速型)
RANN2	正規乱数の生成
DVRAN3	正規乱数の生成 (倍精度) ［拡張機能 II］
RANE2	指数乱数の生成
RANP2	Poisson 乱数の生成
RANB2	二項乱数の生成
RATF1	一様乱数 $(0, 1)$ の頻度テスト
RATR1	一様乱数 $(0, 1)$ の上昇・下降連テスト

●特殊関数

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

●近似

LESQ1	最小二乗近似多項式

●微分方程式

RKG	連立 1 階常微分方程式 (Runge-Kutta-Gill 法)
HAMNG	連立 1 階常微分方程式 (Hamming 法)
ODRK1	連立 1 階常微分方程式 (Runge-Kutta-Verner 法)
ODAM	連立 1 階常微分方程式 (Adams 法)
ODGE	stiff 連立 1 階常微分方程式 (Gear 法)

●数値積分

SIMP1	1 次元有限区間積分 (等間隔離散点入力, Simpson 則)
TRAP	1 次元有限区間積分 (不等間隔離散点入力, 台形則)
SIMP2	1 次元有限区間積分 (関数入力, 適応型 Simpson 則)
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
BIF4	B-spline 補間式 (IV) 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 Newton-Cotes9 点則)
AQC8	1 次元有限区間積分 (関数入力, Clenshaw-Curtis 型積分法)
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)
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)
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)
AQMC8	多次元有限領域積分 (関数入力, Clenshaw-Curtis 型積分法)
AQME	多次元積分 (関数入力, 二重指数関数型積分公式)

●数値微分

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
BIF4	B-spline 補間式 (IV) 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	Chebyshev 級数の導関数, 数値微分

F　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／ 4 倍精度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	正値対称行列の正規化
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)
CHLBDV	Cholesky 法, 改訂 Cholesky 法による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)
CHOLFV	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
LEQBDV	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
LEQLUV	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
MINVV	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

BROYDV	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	複素係数低次代数方程式の解法
QUADRS	実係数低次代数方程式の解法
RTFNDS	非線形方程式の解法

●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- 逆反復法による $Ax = \lambda Bx$ 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
GHBSVV	Householder-Bisection 法による $Ax = \lambda Bx$ 型の固有値解析
GHQRIV	Householder-QR- 逆反復法による $Ax = \lambda Bx$ 型の固有値解析
GHQRVV	Householder-QR 法による $Ax = \lambda Bx$ 型の固有値解析
HEQRVV	Double QR 法による実非対称行列の固有値解析
HOBSVV	Householder-Bisection 法による実対称行列の固有値解析
HOQRVV	Householder-QR 法による実対称行列の固有値解析
HQRIIV	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	同時反復法による $Ax = \lambda Bx$ 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	同時反復法による $Ax = \lambda x$ 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 Spline 補間
DSCI1D	2 変数 Spline 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 $(0 \leq k \leq 3)$

● Fourier 解析

BITREV	ビット逆転によるデータの並べ換え
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

●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-Fehlberg4 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-Merluzzi4 次法による連立 1 階常微分方程式の初期値問題の解法

●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

●数列と級数の加速

ACCELS	数列または級数の収束の加速

●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	$e^x - 1$
SINHP	引数 $\frac{\pi}{2}x$ に対する三角関数

●特殊関数

ABRMO	Abramowitz 関数 ($0 \sim 2$ 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 Gamma 関数
BETIC	不完全 Beta 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 Gamma 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	Legendre 多項式および Legendre 陪多項式
QN	第 2 種 Legendre 関数および Legendre 陪関数
QNOME	楕円 θ 関数の Nome
RGAMA	Gamma 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

●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

●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	$\zeta(n) = \sum_{k=1}^{\infty}(1/k^n)$

●直交多項式

PLEGE	Legendre 多項式 $P_n(X)$
PLEGA	Legendre 陪関数 $P_n^m(X)$
PLEGN	規格化 Legendre 陪関数 $\bar{P}_n^m(X)$
PCHB1	第 1 種 Chebychev 多項式 $T_n(X)$
PCHB2	第 2 種 Chebychev 多項式 $U_n(X)$
PLAGU	Laguerre 多項式 $L_n(X)$
PLAGG	一般 Laguerre 多項式 $L_n^{(a)}(X)$
PHERM	Hermite 多項式 $H_n(X)$

● Bessel 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
JOY0S	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/V Fortran 90/VP 使用手引書 V10 用, J2U5-0050, 富士通株式会社, 1996.

[2] UXP/V Fortran 90/VPP 使用手引書 V10 用, J2U5-0080, 富士通株式会社, 1996.

[3] UXP/V Fortran 90 メッセージ説明書 V10 用, J2U5-0060, 富士通株式会社, 1996.

[4] UXP/V VP プログラミングハンドブック V10 用, J2U5-0070, 富士通株式会社, 1997.

[5] UXP/V VPP プログラミングハンドブック V10 用, J2U5-0090, 富士通株式会社, 1997.

[6] 岩下 英俊, 進藤 達也, 岡田 信: VPP Fortran: 分散メモリ型並列計算機言語, 情報処理学会論文誌, Vol.36, No.7, pp.1542–1550 (1995).

[7] UXP/V C/VP 使用手引書 V10 用, J2U5-0120, 富士通株式会社, 1997.

[8] UXP/V C 言語使用手引書 V10 用, J2U5-0110, 富士通株式会社, 1995.

[9] UXP/V ANSI C プログラミング支援ツール使用手引書, J2U5-0160, 富士通株式会社, 1995.

[10] FUJITSU C++ 言語システム説明書, J2X0-0670, 富士通株式会社, 1994.

[11] UXP/V アナライザ使用手引書 V12 用, J2U5-0131, 富士通株式会社, 1997.

[12] FUJITSU S ファミリー VPP WorkBench 使用手引書 1.0 用, J2S1-0660, 富士通株式会社, 1995.

[13] UXP/V PVM 使用手引書 V10 用, J2U5-0140, 富士通株式会社, 1997.

[14] UXP/V MPI 使用手引書 V10 用, J2U5-0270, 富士通株式会社, 1996.

[15] PARMACS reference manual, user's manual, PALLAS GmbH, 1995.

[16] 富士通 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.

[17] FIJITSU 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.

[18] FUJITSU SSL II 拡張機能使用手引書 II(科学用サブルーチンライブラリ), J2X0-1360, 富士通株式会社, 1997.

[19] FUJITSU SSL II/VPP 使用手引書 (科学用サブルーチンライブラリ)V12 用, J2X0-1370, 富士通株式会社, 1997.

[20] NUMPAC 利用手引書, 富士通株式会社, 1994. (`http://cronos.fuis.fukui-u.ac.jp/numpac/` から検索可能)

[21] FUJITSU α-FLOW 初級講習会テキスト, 富士通株式会社, 1996.

[22] FUJITSU S ファミリー α-FLOW 入力モジュール解説書 解析条件入力データ作成編 第 1 分冊, J1S1-5050, 富士通株式会社, 1996.

[23] FUJITSU S ファミリー α-FLOW 入力モジュール解説書 解析条件入力データ作成編 第 2 分冊, J1S1-5060, 富士通株式会社, 1996.

[24] FUJITSU S ファミリー α-FLOW 入力モジュール解説書 形状定義・格子生成編, 53AL-5020, 富士通株式会社, 1992.

[25] FUJITSU S ファミリー α-FLOW 出力モジュール解説書 解析条件入力データ作成編 第 2 分冊, J1S1-5070, 富士通株式会社, 1996.

[26] FUJITSU α-FLOW 理論説明書, J1X0-5030, 富士通株式会社, 1996.

[27] FUJITSU α-FLOW 入門書, 99AL-5010, 富士通株式会社, 1996.

[28] FUJITSU S ファミリー MASPHYC/WB 使用手引書, J1S1-1060, 富士通株式会社, 1994.

[29] 計算材料設計システム MASPHYC 登録ポテンシャル関数説明書, 富士通株式会社, 1997.

[30] LS-DYNA3D 初級講習テキスト, 富士通株式会社, 1997.

[31] FEMB User's Manual Version 26.5 — Finite Element Model Builder —, Engineering Technology Associates, Inc., 1996.

[32] FUJITSU VisLink User's Guide(Real-time visualization software, J1X0-1090, 富士通株式会社, 1996.

[33] Gaussian94 User's Reference, Gaussian, Inc., 1995.

[34] Gaussian94 Programmer's Reference, Gaussian, Inc., 1995.

[35] Exploring Chemistry with Electronic Structure Method, Second Edition, Gaussian, Inc., 1996.

[36] JIS プログラミング言語 Fortran, JIS X3001-1994, 日本規格協会, 1994.

[37] 東田 幸樹, 山本 芳人, 熊沢 友信: 入門 Fortran90 実践プログラミング, ソフトバンク, 1994.

[38] M. Metcalf, J. Reid (西村 恕彦, 和田 英穂, 西村 和夫, 高田 正之 訳): 詳解 Fortran 90, bit 別冊, 共立出版, 1993.

[39] 馬場 紀彰, 小森 悟: 有限差分法を用いた自由表面を持つ気液二相流の数値シミュレーションプログラム, 九州大学大型計算機センター広報, Vol.30, No.4, pp.317–322 (1997).

[40] 肥田木 直子: まあ, お茶でも飲みながら I ～始めてのファイル転送～, 九州大学大型計算機センター広報, Vol.28, No.3, pp.232–243 (1995).

[41] 肥田木 直子: まあ, お茶でも飲みながら II ～前回の続き～, 九州大学大型計算機センター広報, Vol.28, No.4, pp.350–355 (1995).

[42] 肥山 詠美子, 上村 正康, 木野 康志, Jan Wallenius: 厳密 3 体理論による量子学的 3 体系束縛状態のエネルギーと波動関数 (その II): 任意中心力ポテンシャル～新登録プログラムの紹介～九州大学大型計算機センター広報, Vol.30, No.2, pp.117–123 (1997).

[43] 伊東 栄典: AVS 入門 (1), 九州大学大型計算機センター広報, Vol.30, No.3, pp.225–241 (1997).

[44] 伊東 栄典: AVS 入門 (2), 九州大学大型計算機センター広報, Vol.30, No.4, pp.323–338 (1997).

[45] 伊東 栄典: AVS 入門 (3), 九州大学大型計算機センター広報, Vol.31, No.1, pp.11–18 (1998).

[46] 上村 正康, 肥山 詠美子, 木野 康志, Jan Wallenius: 厳密 3 体理論による量子学的 3 体系束縛状態のエネルギーと波動関数 (その I): ミューオン分子～新登録プログラムの紹介～九州大学大型計算機センター広報, Vol.29, No.2, pp.78–81 (1996).

[47] 小柴 洋一: 円分多項式 (Cyclotomic Polynomial) の係数の計算, 九州大学大型計算機センター広報, Vol.30, No.2, pp.141–145 (1997).

[48] 南里 豪志: VPP 上の PVM 利用法, 九州大学大型計算機センター広報, Vol.30, No.4, pp.339–362 (1998).

[49] 渡部 善隆, 山元 規靖: C プログラムから SSL II を利用するには, 九州大学大型計算機センター広報, Vol.29, No.3, pp.234–242 (1996).

[50] 渡部 善隆: Graphman/UXP の紹介, 九州大学大型計算機センター広報, Vol.29, No.1, pp.1–7 (1996).

[51] 渡部 善隆: Fortran 90 の紹介, 九州大学大型計算機センター広報, Vol.29, No.4, pp.301–313 (1996).

[52] 渡部 善隆, 山本 野人, 中尾 充宏: 実対称行列の正定値性判定プログラム, 九州大学大型計算機センター広報, Vol.31, No.1, pp.1–10 (1998).

[53] 南里 豪志: 大型計算機センターの UNIX 入門, 九州大学大型計算機センター広報, Vol.31, No.2, pp.61–102 (1998).

[54] 利用の手引・ MSP:TCP/IP 編, 九州大学大型計算機センター (1996).

索引