VPP700/56 での対話型処理

渡部 善隆 *

1997 年 4 月より，スーパーコンピュータ VPP700/56 に直接アクセス (login) できるようになりました．本稿は，VPP700/56 での対話型処理[1] (アクセス方法，処理コマンドなど) についての簡単な解説です．

全国の研究者に計算機資源を提供するというセンターの性格およびシステムの制限から，一利用者が対話型処理で使用できる資源はそれほど潤沢とはいえません．しかし，プログラム開発段階でのデバッグや並列プログラミングにおける翻訳レベルでのバグ出し，およびオブジェクトファイル，アーカイブライブラリの作成など，大いに役に立つことと思います．

なお，対話型処理以外の VPP700/56 の詳しい利用方法については，『VPP700/56 利用の手引 (第 1.0 版)』([1]) を御覧下さい[2]．

1 対話型処理の概要

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[3]．ホスト名は "kyu-vpp" です[4]．

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 (kyu-cc, IP アドレス 133.5.9.1) の UXP/M 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 [2] を参照下さい．

1.2 制限値

対話型処理の制限値は以下の通りです．

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．

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

[1] 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，従来の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) 処理形態と定義させていただきます．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．

[2] 『VPP700/56 利用の手引 (第 1.0 版)』の入手を希望される方は，連絡所経由で共同利用掛までお申し込みください．また，最新のバージョンは kyu-cc の `/usr/local/doc/VPP700guide.ps` で PostScript ファイルとして公開しています．現在この原稿を含めた新版を作成中です．

[3] "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．

[4] ネーミングのセンスについては私に責任はありません．

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以下の通りです．実行は Fortran 90/VP, C/VP などの 1PE で動作するプログラムで可能です．並列ライブラリは実行可能ファイルの作成までです．

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-DYNA3D, 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．

1.4 kyu-vpp と kyu-cc

ここでは，M-1800/20U の UXP/M(ホスト名 kyu-cc) の対話型処理と比べた kyu-vpp の使い勝手について書かせていただきます[5]．

• 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，文法のレベルにおいては信用できるものだと思います．

なお，Fortran 90 の文法に関しては [3], [4], [5] を御覧下さい．

[5] あくまでも個人の意見です．できるだけ正直に書いたつもりですが...

◇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．以下の表は，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．

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．1996 年 1 月の段階では最新のワークステーションでした．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[6]．

◇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．従って従来 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行えるようになりました．

◇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 (cf.[1]) に敏感なプログラムや，VPP700/56 にしかない機能[7]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 の対話型処理では，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．

• 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表をあげました．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[8]．結果は次の通りです．

---

[6]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．

[7]例えば，SSL II/VP の拡張機能 II や GETTOD サービスルーチンなど．

[8]翻訳は kyu-cc は `frt` コマンドで (`-J` オプションをつけずに)，kyu-vpp では `frt -Wv,-sc` で行いました．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．

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[9]．

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「手前でやれ」ということでもあります．アムダールの法則 (cf. [1]) によれば，ベクトル化率[10]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 [1], [7], [8], [9] 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[11]．

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[12]．また，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. [10], [11], [12])．従って，勇んで並列化を行う前に，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．また，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[13]．

従って，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[14]．

◇並列ジョブが対話的に実行できない

並列プログラム (Fortran 90/VPP, MPI, etc.) を kyu-vpp で対話的に翻訳・結合することは可能です．従って，翻訳レベルでのデバッグは可能です．しかし，対話的な実行はできません．実行はバッチ処理になります[15]．

[9] 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．

[10] 定義は省略しますが，VU 率でよくおきかえます．

[11] 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．

[12] なぜかといえば，スカラー演算性能がそれほど高くないからです．

[13] 一概に性能の比較はできませんが，スーパーコンピュータ性能測定用に用意した 6 本の利用者プログラムでの測定結果では，平均で kyu-cc の 3.6 倍の翻訳時間がかかりました (1997 年 4 月現在)．

[14] 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．

[15] 並列ジョブの実行は kyu-cc でもできないので比較の項目として不適当ですが，悲しいので載せました．

2　対話型処理の流れ

この章では，具体的な例に沿って対話型処理の手順を説明します．なお，利用者は a79999a さんとします．

2.1　kyu-vpp へのアクセス

2.1.1　login

kyu-vpp に login します．IP 接続されたワークステーションから login します．

```
user-ws% telnet 133.5.9.70
Trying 133.5.9.70 ...
Connected to 133.5.9.70.
Escape character is '^]'.


UXP/V TELNET (kyu-vpp)

login: a79999a
Password:
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:
kyu-vpp%
```

kyu-vpp のパスワードは 1997 年 4 月 1 日 現在の kyu-cc のパスワードです[16]．login した後は `passwd` コマンドで kyu-vpp 独自のパスワードに変更できます．

2.1.2　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．

2.1.3　エディタ

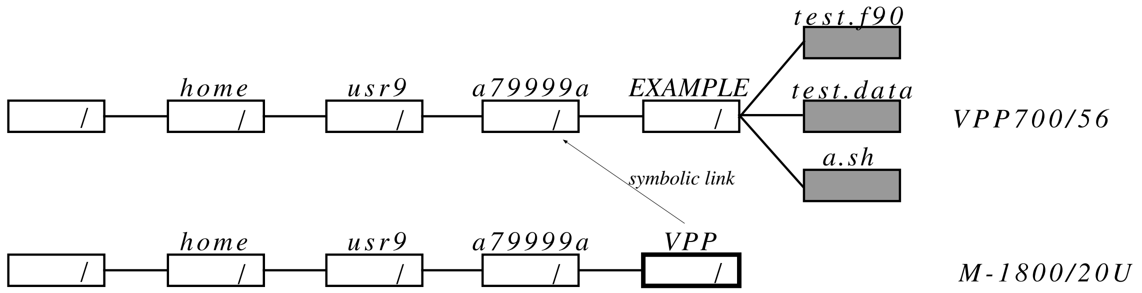
`vi(/bin/vi)` と `emacs(/usr/local/bin/emacs)` が利用できます．ただし，現在のところ `emacs` で日本語入力はできません[17]．

2.1.4　kyu-cc と kyu-vpp との関係

汎用機 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．

[16] 1997 年 4 月 1 日以降センター課題を取得された場合は，初期パスワードです．また，毎年度 4 月に kyu-cc のパスワードに書き変わることもありません．1997 年 4 月 1 日以降 kyu-cc のパスワードを変更したりなどして kyu-vpp のパスワードが分からなくなってしまった場合は，共同利用掛まで連絡願います．

[17] 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．また，日本語の表示は問題ありません．



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

```
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
```

は全く同じものです．

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 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．

つまり，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[18]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[19]．`utoprint` コマンドは利用できません．

2.1.5 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[20]の場合，マニュアルは大抵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．

[18]本当は裏で転送しているのですが

[19]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．

[20]`env` と入力することにより，環境の一覧が表示されます．

2.2 Fortran

2.2.1 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．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[21]で使い分けることをお勧めします．

2.2.2 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．また，kyu-cc でベクトル演算処理を指示する場合 (kyu-cc の frt コマンドに) 必要な翻訳時オプション -J は必要ありません．

```
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[22]．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）
```

2.2.3 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，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，対話型処理による Fortran 90/VPP の文法レベルでのデバッグが容易にできます．

```
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．
                    :
```

2.2.4 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[23]．

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

---

[21] suffix：末尾に添えたもの，付加したもの

[22] 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．

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

2.2.5 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
```

2.2.6 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
```

2.2.7 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], [8])．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．しかし，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
```

2.2.8 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 -Dasu -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
```

また，デバッグオプションを指定した場合，最適化やベクトル化は最低限に押えられますので、実行時間は大幅に増加します．そのため，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．

2.2.9　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_
                               :
```

2.2.10　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ライブラリ名を「アーカイブファイル」と呼びます[24]．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．主なオプションは次の通りです．

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

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

```
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．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下さい．

2.2.11 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
```

2.2.12 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 ↵     <--- 分割コンパイル
```

2.2.13 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．[8] の 4.5 節を参照下さい．

2.2.14 標準入力からデータを読み込む

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取り込む

出力ファイルが存在しない場合は，新規に作成されます．

```
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．

2.2.15 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下さい．

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
```

2.2.16 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変なことになりますのでご注意下さい．

2.2.17 プログラム中にファイルを指定する

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を書き込む
2. 環境変数として設定する

かのどちらかです．

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')
```

2.2.18 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

fu*nn*␣*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．

2.2.19 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
```

2.2.20 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
```

2.2.21 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
```

2.2.22 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．

2.3 C, C++

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なども同じです．

2.3.1 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．

2.3.2 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．

2.3.3 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. [9])．

2.3.4 SSL II/VP の結合

```
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
a.out
```

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．

SSL II/VP を呼び出す場合の C プログラミングの注意点は [6] を参照下さい．

2.3.5 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．

2.3.6 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
```

2.4 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，結合のみ可能で，並列実行はできません．

2.4.1 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. ↵
```

2.4.2 PVM の例 (C)

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

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] を参照下さい．

2.4.3 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．

2.4.4 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．

2.4.5 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
```

2.4.6 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．

2.5 Analyzer

Analyzer の利用は，1PE のジョブに限って対話的に利用できます．機能の詳細は [1] を参照下さい．

2.5.1 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，利用者が任意の名前で指定します．

2.5.2 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．ここでは，ファイル out に結果を書き出しています．

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．

2.5.3 PEPA による情報収集

1PE の場合のみ，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
-----------------------------------------------------------------------------
```

2.6 Gaussian94

Gaussian94 の利用方法は，kyu-vpp のホームディレクトリの “.cshrc” の環境設定を行うことで，kyu-cc と全く同じように利用できます．詳しくは [1] を参照下さい．

2.7 バッチリクエストの投入

kyu-vpp からのバッチリクエストの投入方法は，従来の kyu-cc からの手順と同様，ファイルに記述したバッチリクエストを qsub コマンドで投入します．

kyu-cc との違いは，リクエスト名を構成するホスト名が “kyu-cc” ではなく，“kyu-vpp” だという点だけです．リクエスト名は，リクエストをキャンセルする際に指定が必要になります．

バッチリクエストの記述方法，リクエスト投入の方法などは [1] をご覧下さい．参考までにバッチキューの制限値をあげておきます．

キュー名	CPU 時間	記憶域	処理形態
c	60 分	100MB	翻訳専用
s	60 分	1.7GB	1PE
p1	1200 分	1.7GB	1PE
p8	1200 分	1.7GB/PE	最大 8PE
p16	1200 分	1.7GB/PE	最大 16PE
p32	1200 分	1.7GB/PE	最大 32PE

3 便利なコマンド集

最後に，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．

3.1 qps

qps(/usr/local/bin/qps) コマンドは，qsub コマンドで投入したバッチリクエストの処理状況をモニターするコマンドです．

qps コマンドは kyu-vpp からのみ利用できます．kyu-cc から利用する場合は，rsh kyu-vpp qps と入力します．

並列ジョブの各 PE 毎の処理状況を知りたい場合は，-p オプションを付加します．

```
kyu-vpp% qps ↵
que  user    request           cpu      vu  vu/cpu cpu-limit  elapse  v-mem(MB)
 p1 a79999a  16803.kyu-cc   0:31:04   0:00:01   0%  20:00:00  0:31:24   48/1792
```

1PE のジョブ (p1 キュー) の実行状況のモニター例です．

```
kyu-vpp% qps -p ↵
que  user    request           cpu      vu  vu/cpu cpu-limit  elapse  v-mem(MB)
 p8 a79999a  16758.kyu-cc  12:57:14  11:34:11  89%  20:00:00  18:50:03 1512/1704
             parallel--->  12:24:59  11:34:40  93%
             parallel--->  12:24:42  11:34:37  93%
             parallel--->  12:24:44  11:34:39  93%
             parallel--->  12:24:40  11:34:39  93%
```

p8 キューに投入した並列ジョブの実行状況の表示です．

3.2 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
```

3.3 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．

3.4 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が得られます．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
```

3.5 kill

何らかの原因で暴走したジョブは kill コマンド[25]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．

[25] シェルのコマンドです．

```
kyu-vpp% ps -u a70033d ⏎              <--- 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
```

参考文献

[1] VPP700/56 利用の手引 (第 1.0 版), 九州大学大型計算機センター (1997).

[2] 佐藤 周行, 山元 規靖, 南里 豪志：UXP 初級講習会資料, 九州大学大型計算機センター (1997).
(講習会資料はセンターの共同利用掛で残部を配布しています)

[3] 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).

[4] 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).

[5] 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).

[6] 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).

[7] UXP/V VP プログラミングハンドブック V10, 富士通株式会社 (to appear).
(現在校正段階です．1997 年の夏前には発行と聞いています)

[8] UXP/V Fortran 90/VP 使用手引書 V10 用, J2U5-0050, 富士通株式会社 (1995).

[9] UXP/V C/VP 使用手引書 V10 用, J2U5-0120, 富士通株式会社 (1995).

[10] UXP/V Fortran 90/VPP 使用手引書 V10 用, J2U5-0080, 富士通株式会社 (1995).

[11] UXP/V VPP プログラミングハンドブック V10, 富士通株式会社 (to appear).
(現在校正段階です．1997 年の夏前には発行と聞いています)

[12] FUJITSU SSL II/VPP 使用手引書 (科学用サブルーチンライブラリ)V11 用, J2X0-1370, 富士通株式会社 (1996)．

[13] UXP/V PVM 使用手引書 V10 用, J2U5–0140, 富士通株式会社 (1995).

[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).