Unix環境でスーパーコンピューティングをする人のためのガイド

佐藤, 周行
九州大学大型計算機センター研究開発部

https://doi.org/10.15017/1470236

出版情報：九州大学大型計算機センター広報. 26 (4), pp.452-484, 1993-07-26. 九州大学大型計算機センター
バージョン：
権利関係：

解　　説

Unix 環境で スーパーコンピューティングをする人のためのガイド

佐藤周行*

目次

平成 5 年 5 月 25 日 受理
*九州大学大型計算機センター研究開発部

九大センターでは 1993 年 1 月から VP2600/10 で UXP/M のサービスを開始し、Unix 環境でも本格的にスーパーコンピューティングができるようになりました。本稿はそのための利用の手引です。

1 キーワードは「ケタ違い」 – Unix 環境でスーパーコンピューティングをする人のためのガイド

処理能力がアップすると一般的に言っていろいろ大きな仕事ができるようになります。そのアップ率がたとえば 2 倍とか 3 倍ならば「速くなったな」とか「大きくなったな」で済みますが、それが 10 倍以上になると仕事の質そのものが変質します。スーパーコンピューティングやビジュアリゼーションはその典型的な例ですが、そこまでいかなくても手計算からパーソナルコンピュータへの移行、さらにワークステーションへの移行によって「変質」を体験している人もたくさんいるでしょう。

ポイント 1 (仕事の質変質の法則) 処理能力が 1 桁あがると仕事の質そのものが変質する。

1.1 センターのケタ違いの処理能力

具体的な数字をあげましょう。科学技術計算において重要な数字は浮動小数点演算の性能です。

ポイント 2 浮動小数点演算の性能を表すのに MFLOPS(Millions of *or* Mega **F**Loating point **O**perations **P**er Secons: 意味は読んで字の如し) が主に用いられる。性能測定のためにいろいろなプログラムが使われるが、その中で最も良く使われるもののひとつに Linpack と呼ばれるパッケージがある。

実は Linpack を用いた倍精度での MFLOPS 値を UXP 講習会資料 “UXP/M for the Working Scientists” を書く時に計測しています。以下の表はその時点での数字ですが...

マシン (OS)	MFLOPS
S-4/2(Sun-OS 4.1.1)	3.3
M1800/20(UXP/M)	26.0

S-4/2 はいわゆるエントリタイプではそこそこの性能を持っています[1]。これから

ポイント 3 (M1800/20 におけるケタ違いの法則) エントリタイプレベルのワークステーションと比較して M1800/20 UXP/M の処理能力はケタ違いである。

[1]これ以上をもとめようとすると ALPHA, HP や IRIS などに行くしかありません。

もうひとつプリンタの能力についても述べておきましょう。最近は高速高性能のページプリンタが出回っていますが、単純にプログラムリストを出す、または計算結果を出すのならばセンター設置の A4 高速プリンタが便利です。NLP[2]へは UXP のコマンド utoprint で出力が可能です。

プリンタ	処理能力
B4 NLP(O)	2000 行 / 分　(約 30 枚 / 分)
A4 NLP(H)	10000 行 / 分　(約 150 枚 / 分)

手持ちのプリンタと比べて下さい。次のことがわかるでしょう。

ポイント 4 (リスト出力におけるケタ違いの法則) 単純なリスト出力程度ならばセンター設置のプリンタは手持ちのプリンタとケタ違いの処理能力を持つ。

1.2 さて VP

さて今回 VP2600 が Unix 環境で利用可能になりました。VP は大規模配列に対してループをがんがん回すプログラムを処理するのに特に適しています。VP2600/10 のカタログ性能は 5.0GFLOPS[3]です。M1800/20 の 200 倍弱です。

ただしこれだけの性能を出そうと思ったらプログラミングに特別の注意を払う必要があります。実はそれだけの労力を払う価値のあるプログラムは世の中にたくさんあることがわかっています。そういうプログラムを抱えている人には次のエールを送りましょう。

ポイント 5 小柳義夫先生曰く、「プログラミングはアートです。」

ただし、特別な注意を払わなくてもちょっと注意するだけで M1800/20 で作成した Fortran プログラムは 10 倍くらい速くなります。つまり

ポイント 6 (VP 利用におけるケタ違いの法則) ちょっと注意するだけで M1800/20 UXP/M で作成した Fortran プログラムは VP2600/10 でケタ違いに速くなる。

ただし、VP 向きでないプログラムを漫然と流すだけではその効率アップは数十 % にとどまります(考えてみればこれだけでもありがたいのですが)。この手のプログラムを VP でより速く実行させるには性能解析とチューニングをきちんとやらなければなりません。

やはりプログラミングはアートであるという自覚を持つことが必要です。

1.3 VP を使う前に

一般的な注意です。

1. センター M1800 の UXP への登録が必要です。登録は MSP から SINSEI コマンドで行なって下さい。
2. VP で実行させる時、負担金は VP の負担金体系が適用されます。具体的に次の通りです。

マシン	～1	～5	～15	15～	(分)
M1800		5	3	1	(円 / 秒)
VP2600	7	6	3	1	(円 / 秒)

[2]Nihongo Line Printer

[3]1 GFLOPS = 1,000 MFLOPS

3. M1800 の UXP へログインした時に出る課金情報は前日までの情報です。VP の負担金は翌日まとめて UXP の負担金に足し込まれます。この点で MSP と異なるので予算管理には注意して下さい。

1.4 本稿の構成

これ以後、VP の使い方について具体的に解説します。バッチシステムの利用方法、プログラミング言語 Fortran と C の使い方、性能解析、チューニング、さらに後処理用のグラフィックスについてできるだけ具体的に述べることにします。

2 VP におけるバッチシステムの運用

現在 UXP/M における VP2600 の運用方式はバッチのみです。TSS のサポートは当面ありません。

2.1 用語

バッチシステムの先輩としては IBM MVS とその仲間 (富士通ならば MSP) が有名で、用語もだいたいそれに従っています。ただ、微妙に用語の使い回しが異なります (本質は変わらないんですけどね)。

MSP を使ったことのない人は以下の説明をそのまま理解して下さい。

2.1.1 バッチシステムの運用 – オリジナル編

1. リクエストキューが用意されています。それぞれについて CPU 時間、メモリなどの計算資源に対する制限値が定められています。
2. ユーザはその制限値をにらみながら、その範囲内で一群の仕事を計算機に実行させることができます。この「仕事」の単位を「バッチリクエスト」と言います。
3. 実行させる時にはバッチリクエストの実行依頼をします。これを「バッチリクエストをサブミットする」と言います。
4. サブミットされたバッチリクエストはユーザの管理をはなれ、計算機側の管理の下で制御、実行されます。計算結果はファイルなどの形で返却されます。

2.1.2 バッチシステムの運用 – MSP ユーザへの翻訳

ここでは主に MSP ユーザに対して今後出てくる用語の意味を MSP に翻訳します。

- リクエストキューとはジョブクラスのことです。
- バッチリクエストとはジョブのことです。
- サブミットはそのままサブミットです。

さて、以後具体的に九大センターでの運用を説明しましょう。

2.2 コマンドの種類

VP2600 で実行できるコマンドは以下の通りです。

種類	ソフトウエア	コマンド
ベクトル化コンパイラ		
	Fortran	frt
	C	vcc
性能評価ツール (Fortran のみ)	アナライザ	afrt

その他に CPU 時間を計測するコマンド timex があります。

M1800 側の /usr/local/bin にあるコマンドは実行できません。VP2600 でのログインシェルは M1800 側での設定にかかわらずすべて csh です。

2.3 リクエストキュー

リクエストキューとしてデバッグ、小規模用と標準の 2 つが利用可能です。利用制限対象になる計算資源は

- CPU 時間
- 使用メモリ

の 2 つです。

また途中でシステムがシャットダウンした場合、計算は一からやり直しになります。Checkpoint-Restart 機能 (MSP 側の SaveHalt 機能に対応するもの) はありません。

本運用開始時点でのリクエストキューとその制限値を下にあげます。

キュー	CPU 時間	使用メモリ
vs	10 分	10MB
vl	60 分	50MB

ここで vl の l は ℓ(ell) であって 1(one) ではありません。フォントの微妙な違いに注目！

2.4 バッチリクエストとは

バッチリクエストはテキストファイルの形で書き、コマンドの並びで記述します。例えば次のように。

```
frt -Ps -Wv,-m3 a.f
a.out
```

これは最初に frt コマンドを実行し、次に a.out コマンドを実行するバッチリクエストです。

さて以下は物の正体を知りたい人向け。わからない人は飛ばしてかまいません。

バッチリクエストは csh または sh のシェルスクリプトで記述します。スクリプトの先頭が # であれば csh、それ以外は sh がインタプリタになります。2 つは (残念なことに) 文法が微妙に異なります。コマンドを羅列したバッチリクエストの場合はシェルの種類を意識することは必要ありません。それ以上のことをやろうとする時には問題になります (例えばバッチリクエストの中に if 文を書く場合) のでその場合だけ注意して下さい。

なお、上のことは自分で UXP/M 側のログインシェルを csh 以外にしている人にもあてはまります。

例 1 (csh で解釈させる場合のリクエスト) リクエストは普通のファイルとして用意すればいいのですが、ここでは整理のために.vp というサフィックスをつけることにします。

先頭の # に注目。

```
kyu-cc:21] cat a.vp
#
frt -Ps -Wv,-m3 a.f
timex -H a.out
kyu-cc:22]
```

後々のために vi などで編集しておきましょう。ファイル名は何でも良いですが、とりあえず a.vp にしておいてください。

面倒ならば cat ででも作って下さい。

例 2 (a.vp の安直な作成法) この方法が役に立つ場合が実はかなりあります。

ただし、cat でファイルの変更はできませんからその場合には ex を覚えて下さい。

```
kyu-cc:21] cat > a.vp
#
frt -Ps -Wv,-m3 a.f
timex -H a.out
^D
kyu-cc:22]
```

2.5 バッチリクエストを投入するまで − 環境の整備

VP 上でバッチを実行させる場合の実行環境は次のようになっています。

1. ディレクトリは VP と M とでは共有。
2. バッチリクエストの実行前にホームディレクトリの.cshrc と.login に書いてあるものが設定されます。VP 側でのログインシェルは M 側のログインシェルの如何にかかわらず csh です。

自分で.profile、.tcshrc に色々な情報を書き込んでいる人がいますが、それらの情報は.cshrc または.login 中で自分で読み込むことを陽に指定しない限り無視されます。特にパスの設定に注意。

デフォルトの.login では端末タイプを対話的に入力するようになっていますが、これはバッチシステムでは拒否され何もせずに終ってしまいます。次は典型的な失敗例です。

例 3 (.login が悪くて実行されない例) 次のような結果が返ってきたらその理由は間違いなく上記のことです。

```
kyu-cc:25] cat a.vp.o780
Warning: no access to tty; thus no job control in this shell...
Terminal Type: logout
kyu-cc:26]
```

つまり VP を使うためにまずすべきことは.login と.cshrc を書き換えることなのです。バッチリクエストがどうしても動かない場合、このステップを省略しているかどうかの確認をした方が良いでしょう。

ポイント 7 バッチリクエストがどうしても実行されない場合、初心に戻って.login と.cshrc のチェックをすること。

書き換えは以下のようにします。

例 4 (センター用意のものをコピーしてくる) センターではデフォルトの.login と.cshrc を /usr/lib/model に用意しています。以下のようにして下さい。以下のようにすれば VP 対応版デフォルトの.login、.cshrc を再設定してくれます。

```
kyu-cc:33] cd
kyu-cc:34] cp /usr/lib/model/login .login
kyu-cc:35] cp /usr/lib/model/cshrc .cshrc
```

デフォルトの.login、.cshrc を使っている人はこれでバッチリクエスト投入の準備ができました。2.6までお進み下さい。なお、1992 年 12 月 22 日以降に UXP への登録を済ませた人はこれをする必要はありません。

例 5 (自分で書き換える) 自分で.login や.cshrc を書き換えている人は自分で書き換えて下さい。書き換えの対象になるのは

- 端末が割り当てられることを想定して対話的になっている部分や端末属性を設定している部分。

です。自分がバッチの中にいるのかどうかは環境変数 ENVIRONMENT が設定されているかどうかでチェックできます。センター用意の login ファイルを参考にして下さい。該当部分を下に載せます。

```
...
if ($?ENVIRONMENT != 0) then
        exit
endif

# 端末設定その他
#
echo -n "Terminal Type: "
...
```

2.6 qsub – バッチリクエストをサブミットする

前述のようにリクエスト a.vp を用意します。それをサブミットするには以下のようにします。

例 6 (バッチリクエストをサブミットする) リクエストキューは -q に続けて指定します。これが指定されないと vs に投入されます。

```
kyu-cc:35] qsub -q vs a.vp
Request 300.kyu-cc submitted to queue: vs.
kyu-cc:36]
```

投入されるとリクエスト番号が返ってきます。上の例でいえば 300 がリクエスト番号です。このリクエストは (VP が落ちていない限り) 直ちに VP2600 に転送され、リクエスト 300.ccux となります。「リクエスト番号. サブミットしたホスト名」でリクエスト名を構成します。

2.7 バッチリクエストの書き方

すでに説明しましたが、バッチリクエストは通常のシェルスクリプトです。リクエストを書く時に特に問題になる部分をあげておきます。

2.7.1 シェルの種類

スクリプトの1行めが # の場合、csh が、それ以外の場合 sh が走ります。

2.7.2 ディレクトリの指定

リクエストを実行する場合、最初はユーザのホームディレクトリにいるつもりになっています。リクエストをサブミットしたディレクトリではありません。

従って、例えばディレクトリ ~/vpjob で仕事をする場合には最初に cd をする必要があります。

例 7 (ディレクトリ vpjob のファイルを展開する場合) スクリプトは以下のようになります。

```
kyu-cc:33] cat a.vp
cd vpjob
frt -Ps -Wv,-m3 a.f
a.out
kyu-cc:34]
```

2.7.3 I/O

I/O の扱いは通常のスクリプトと微妙に違います。

1. stdin は端末から切り離されます。従って対話的に作ったロードモジュールはその実行に失敗します。その場合はヒアコマンドまたはリダイレクトで stdin を調整して下さい。

   例 8 (stdin を調整する) まずみなさん見なれた redirection から。この例ではファイル infile を使用します。

   ```
   kyu-cc:35] cat a.dir.vp
   cd vpjob
   frt -Ps -Wv,-m3 a.f
   a.out < infile
   kyu-cc:36]
   ```

   面倒だったらヒアコマンドを使うのも一つの手です。例えば infile の中身が

   ```
   kyu-cc:36] cat infile
   1
   2 4
   8
   kyu-cc:37]
   ```

   の場合、以下のようにしても上のスクリプトと同じ結果が得られます。

```
kyu-cc:37] cat a.hear.vp
cd vpjob
frt -Ps -Wv,-m3 a.f
a.out << EOF
1
2 4
8
EOF
kyu-cc:38]
```

2. リクエストをサブミットすると、サブミットした時点のディレクトリに「スクリプト名.o リクエスト番号」と「スクリプト名.e リクエスト番号」というファイルが自動的にできます。stdout と stderr の出力はそこに書き出されます。stdin からリクエストをサブミットした場合はスクリプト名が「STDIN」になります。例として一連の作業を連続して実行してみましょう。

   例 9 (stdout と stderr の出力先) 下の例の場合 stdout の出力先は a.vp.o307、stderr の出力先は a.vp.e307 になります。

```
kyu-cc:35] ls
a.vp          a.f
kyu-cc:36] qsub -q vs a.vp
Request 307.kyu-cc submitted to queue:vs

      ...

kyu-cc:42] ls
a.vp          a.vp.e307     a.vp.o307     a.f           a.out
kyu-cc:43]
```

   もちろん結果を redirection によって普通のファイルに書き出すことは可能です。どちらが良いかは各々判断して下さい。

2.7.4 資源の使用状況のモニタ

さすが Unix と思わせる点が NQS のそっけなさです。使用 CPU 時間、使用メモリその他デフォルトでは一切出力されません (MSP の懇切丁寧さを見習ってほしいものだ...)。CPU 時間がオーバーしたら「CPU time exceeded」のメッセージを出してちょんぎるだけです。メモリ使用量は自分で計算して下さい。CPU 時間の方はコマンド timex で計測することができます。結局

ポイント 8 (冷たいけれど)Unix においては資源管理は自分の責任。

例 10 (CPU 時間を計測する) 次のようなスクリプトを書くと a.out の実行時間が CPU、VPU それぞれにわかれて標準エラーに出てきます。ベクトル化率はそれを基に自分で計算しましょう。

```
kyu-cc:41] cat a.vp
cd vpjob
frt -Ps -Wv,-m3 a.f
timex -H a.out
kyu-cc:42] qsub -q vs a.vp
Request 308.kyu-cc submitted to queue:vs
kyu-cc:43]
     ...
kyu-cc:44] cat a.vp.e308
     ...
real            0.44
user            0.14
sys             0.16
vu-user         0.04
vu-sys          0.00
vuw-user        0.00
vuw-sys         0.00


kyu-cc:45]
```

この場合の VU 率は ((VPU = vu-user + vu-sys) / (CPU = user+sys)) = 0.04/(0.14 + 0.16) = 13% になります。なお、vuw-user と vuw-sys は九大センターのシステムでは 0 が返ってきます。

ポイント 9 VU 率とは全体にかかった CPU 時間のうち、ベクトルユニットで動いていた時間の割合を言う。これが高いほど VP を使いこなしていることになる。

2.7.5 エラーリカバリ – 涙のリクエスト

リクエストは普通のスクリプトですからエラー処理はユーザが責任を持ってする必要があります。最悪の場合を考えましょう。問題: 次のスクリプトはどこがいけないか?

```
cd vpjob
frt -Ps -Wv,-m3 a.f
timex -H a.out
```

正解: いけないところはどこもありません。ただし、ディレクトリ ~/vpjob がきれいに掃除されていればの話です。もし a.out がジョブのリクエスト前にすでに存在し、コンパイルが何らかの原因で失敗した場合、実は古い a.out が実行されてしまいます。これでは泣くに泣けません。

用心深い人は次のようにするに違いありません[4]。ええと、ここでの教訓。

ポイント 10 フェイルセーフは基本だけれど、そんなことをする必要がないくらい常日頃からディレクトリの中身をきれいにしておこう。

[4] やりすぎという感じもしますが。

```
kyu-cc:52] cat a.safe.vp
cd vpjob
if test -x a.out
then
        mv a.out a.out.old
fi
frt -Ps -Wv,-m3 a.f
timex -H a.out
kyu-cc:53]
```

2.8 バッチリクエストの結果をもらう

さて、結果はファイルの形で返ってきました。それを出力するのにはとりあえずプリンタです。

例 11 (結果をプリンタに出力する) プログラムリストをセンターのNLPに出力します。

```
kyu-cc:48] utoprint a.vp.e307
utoprint : output to NLP
kyu-cc:49]
```

最近、無味乾燥な数字のリストだけではいけない！ということが理解されてきて、きれいなグラフの形にして出すことがはやっています。7節でここらへんを解説しています。参考にして下さい。

2.9 qstat と qps – バッチリクエストの処理状況をみる

qstat はリクエストの処理状況をキュー毎に出力します。使い方は下を見て下さい。

例 12 (qstat の使い方) 引数なしだとローカルなマシンのキューの状況を、@ に続いてマシン名を指定するとそのマシンのキューの状況を報告します。

```
kyu-cc:43] qstat

vs@kyu-cc;  type=PIPE;  [ENABLED, RUNNING];  pri=20
  0 depart;   0 route;   0 queued;   0 wait;   0 hold;   0 arrive;


vl@kyu-cc;  type=PIPE;  [ENABLED, RUNNING];  pri=10
  0 depart;   0 route;   0 queued;   0 wait;   0 hold;   0 arrive;

kyu-cc:44] qstat @vpux

vs@vpux;  type=BATCH;  [ENABLED, RUNNING];  PIPEONLY;  pri=20
  0 exit;   1 run;   4 queued;   0 wait;   0 hold;   0 arrive;

        REQUEST NAME        REQUEST ID            USER  PRI    STATE     PGRP
   1:           a.vp        308.ccux            a79999a  31  RUNNING    10769
<2 requests QUEUED>
   4:          STDIN        313.ccux            a79999a  31   QUEUED
   5:          STDIN        314.ccux            a79999a  31   QUEUED

vl@vpux;  type=BATCH;  [ENABLED, RUNNING];  PIPEONLY;  pri=10
  0 exit;   0 run;   0 queued;   0 wait;   0 hold;   0 arrive;

        REQUEST NAME        REQUEST ID            USER  PRI    STATE     PGRP
<1 request RUNNING>
kyu-cc:45]
```

ここで意味のあるのは qstat @vpux の方です。上の例では VP 側でリクエスト 308.ccux が実行中であり、その他に 2 件たまっていることを示しています。なお、出力メッセージ中の <2 requests QUEUED> は誰か他人がジョブを流し、それが 2 件、313.ccux と 314.ccux の前にたまっていることを示しています。

一方 qps は実行中のリクエストの進行状況をレポートするコマンドです。使い方は以下の通りです。

例 13 (qps の使い方) 実行中のリクエスト 308.ccux の進行状況をみるには以下のようにします。

```
kyu-cc:45] qps 308.ccux
     UID   PID  PPID  C    STIME TTY      TIME COMD
 a79999a 10769 10768  0 16:24:47 ?        0:00 -csh
 a79999a 10776 10775 22 16:24:47 ?        0:09 a.out
 a79999a 10774 10769  0 16:24:47 ?        0:00 /bin/sh /usr/spool/nqs/scripts/2
 a79999a 10775 10774  0 16:24:47 ?        0:00 timex -H a.out
kyu-cc:46]
```

9 秒走っていることがわかります。

2.10 qdel – バッチリクエストをキャンセルする

バッチリクエストのキャンセルは qdel を用いて次のようにします。

例 14 (リクエストをキャンセルする) キューにたまっている状態ならば次のようにすればキャンセルできます。

```
kyu-cc:55] qstat @vpux
vs@vpux;  type=BATCH;  [ENABLED, RUNNING];  PIPEONLY  pri=20
  0 exit;   1 run;   1 queued;   0 wait;   0 hold;   0 arrive;

        REQUEST NAME        REQUEST ID            USER  PRI    STATE     PGRP
   1:        two.csh       309.ccux            e70019a  31  RUNNING    11362
   2:        two.csh       310.ccux            e70019a  31   QUEUED

vl@vpux;  type=BATCH;  [ENABLED, INACTIVE];  PIPEONLY  pri=10
  0 exit;   0 run;   0 queued;   0 wait;   0 hold;   0 arrive;


kyu-cc:56] qdel 310.ccux
Request 310.ccux has been deleted
kyu-cc:57]
```

問題なのは実行途中のリクエストのキャンセルです。オプションとして -k をつけなければなりません。

例 15 (実行途中のリクエストをキャンセルする) 次のようにします。

```
kyu-cc:55] qstat @vpux
vs@vpux;  type=BATCH;  [ENABLED, RUNNING];  PIPEONLY;  pri=20
  0 exit;   1 run;   1 queued;   0 wait;   0 hold;   0 arrive;

        REQUEST NAME        REQUEST ID            USER  PRI    STATE     PGRP
   1:        two.csh       309.ccux            e70019a  31  RUNNING    11362

vl@vpux;  type=BATCH;  [ENABLED, INACTIVE];  PIPEONLY;  pri=10
  0 exit;   0 run;   0 queued;   0 wait;   0 hold;   0 arrive;

kyu-cc:56] qdel -k 309.ccux
Request 309.ccux is running, and has been signalled.
kyu-cc:57]
```

2.11 まとめ

冗長な説明でしたが、まとめると次のようになります。

1. リクエストキューは vs と vl。
2. サブミットするバッチリクエストは csh または sh のスクリプトとして記述する。

3. ディレクトリは VP と M とで共有。
4. 正しくバッチリクエストがサブミットされるためには.loginと.cshrcを書き換える必要がある。
5. サブミットするコマンドは qsub。
6. サブミットするとリクエスト番号が返ってくる。バッチリクエストが終了すると「スクリプト名.o リクエスト番号」と「スクリプト名.e リクエスト番号」に結果が返ってくる。
7. 資源管理は自分の責任。
8. 途中でエラーが起こった場合の処理も自分の責任。
9. リクエスト処理状況は qstat で見ることができる。
10. リクエストをキャンセルする場合は qdel を用いる。

3 Fortran 77/EX VP コンパイラ

3.1 やっぱり Fortran

Unix だろうがなんだろうが、こと科学技術計算の分野においては Fortran が記述言語の王様です。主に Pascal を使う教条主義者から Fortran に浴びせられた罵声は数限りなくあります。曰く構造化プログラミングができない、近代的なデータ構造を持っていない、等々。しかし真の科学技術計算野郎はこんないいがかりをまともにとることはありません。VP 上で大規模配列を相手に「ぶんまわす」計算をする時に性能と汎用性、普及度を考えると Fortran にかなう言語は (現在) 存在しません。

ポイント 11 Fortran はスーパーコンピューティングにおける現在の主力言語である。

VP 上の Fortran77EX/VP コンパイラは特に DO ループ (だけ) を対象にしてベクトル化を行ないます。

しかし「UNIX なら C だろう」という (マニアかミーハーかはともかく) その手の声も実際多く出るようになってきました。それに応えて VP の上でもベクトル化 C コンパイラをサポートしています。しかしこれも言語に本質的な制約があって、本当に性能を出そうとすれば現在のところ自分で手を加える必要があります。つまり C に関してはコンパイラ技術が成熟しているとはいえません。

ポイント 12 C はスーパーコンピューティングにおいてはまだまだ改善の余地を残す言語である。

というわけで「VP を使うには Fortran!」ということで Fortran をまず説明し、ついでに C ということにします。

3.2 コンパイルの仕方

VP 用のコードを生成するにはリクエストの中に以下のように書きます。ここでディレクトリ vpjob 下でソースプログラム a.f をコンパイルして実行するものとします。リクエストにはとりあえずこのように書いておけば大丈夫です。

例 16 (Fortran ソースプログラムをコンパイルし実行させるリクエスト)

```
cd vpjob
if test a.out
then
   rm -f a.out
fi
#
frt -Ps -Aa -Wv,-m3 a.f
timex -H a.out
```

ベクトル化表示プログラムリストが標準エラーに出力されます。なお、-Wv を省略するとベクトル化されたコードが出力されません。十分注意して下さい。

3.3 ライブラリの組込 – SSL II

現時点では富士通提供の科学技術計算ライブラリ SSL II がサポートされています。このルーチンを使ったプログラムを書いている場合は以下のように指定します。

例 17 (SSL2 を使った Fortran ソースプログラムをコンパイルし実行させるリクエスト)

```
cd vpjob
if test a.out
then
   rm -f a.out
fi
#
frt -Ps -Aa -Wv,-m3 -lssl2vp -lssl2 a.f
timex -H a.out
```

ここで -lssl2vp を指定しないと VP 対応版が正しくリンクされませんから注意して下さい。

3.4 ライブラリの組込 – NUMPAC

SSL II 以外の数値計算ライブラリとして NUMPAC が利用できます。

例 18 (NUMPAC のルーチンを使う) NUMPAC のルーチンを使う場合、以下の num.vp のようなバッチリクエストを書けば OK です。ただし、これはディレクトリ ~/vpjob のファイル num.f をコンパイルする場合です。

```
kyu-cc:33] cat num.vp
cd vpjob
frt -Ps -Wv,-m3 num.f -L/usr/local/lib -lnumpacvp -lnumpac
timex -H a.out
kyu-cc:34]
```

3.5 I/O の扱い

I/O の機番と実際のファイル名の対応のさせ方は通常の Fortran プログラムと同じです。デフォルトでは以下の対応がついています。

ファイル	機番
stderr	0
stdin	5
stdout	6

ただし、標準入力 (機番 5) から読み込む場合は redirection 機能を使って端末以外から入力を読み込むようにしないと異常終了します。

3.6 最適化に関するオプションを自分で指定する

最適化によって副作用が出て、さらにそれが異常終了など好ましくない結果につながる場合や、突っ込んだ解析をしてくれず (コンパイラは大体安全サイドに立ちます) 思ったようなコードが出ない場合などは最適化のレベルを調節する必要があります。

3.6.1 用語の解説

最適化ではコンパイラ屋さんの用語 (つまり実際のユーザには馴染みのない言葉) がでてきます。でも中身は実は簡単。その中で本文の内容に関係ある基本的な用語を解説しましょう。

ポイント 13 コンパイラ屋は単純な内容のことを難しく言い直したり、カタカナでいったりしてなんとか自分が高尚なことをやっているんだと思わせたがる。

不変式の先行評価 例えば以下のプログラムを考えましょう。

```
      do 100 i=1,10000
       param=1.0
       array(i) = param+i
  100 continue
```

これをまともに実行すると変数 parm に 1.0 を代入する作業が 10000 回実行されます。ところがこれは以下のプログラムと意味的に変わりません。

```
      param=1.0
      do 100 i=1,10000
       array(i) = param+i
  100 continue
```

このようにループの中で毎回同じ値が計算されるものをループの外に出す変換をすることを「不変式の先行評価」といいます。実際、プログラムの大きさが大きくなった場合など気づかないままに不変式を書いてしまっていることがあります。コンパイラは不変式を自動的に検出して下の形式に自動的に変換してコンパイルをしてくれる能力を持っています。

ループアンローリング 以下のループを考えます (ループアンローリング前)。

```
      do 100 i=1,100
       a(i) = i
  100 continue
```

これを以下のように変換して

ループの分岐条件のチェック回数を減らして高速実行をするのがループアンローリングです。これもコンパイラは自動的に展開してくれます。

```
      do 100 i=1,100,2
       a(i) = i
       a(i+1) = i+1
  100 continue
```

演算評価方法の変更 評価方法の変更には複数の方法がありますが、ここでは strength reduction だけを説明しましょう。一般に計算機においては加減算より乗算、さらに除算となるにしたがって計算コストが増大します。これを利用して乗算を加算に置き換えたり、除算を乗算に置き換えたりすると一般に計算コスト (この場合は時間) が減ります。この最適化を strength reduction といいます。

3.6.2 最適化を指定するオプション

最適化は以下のように指定します。なお、-Wv を指定すると -Oe がデフォルトになります。

-Ob	基本的な最適化レベル
-Oe	上に加えて不変式の先行評価と演算評価方法の最適化を行なう
-Of	上に加えてプログラム単位間の最適化も行なう
,-p	不変式の先行評価の最適化をする。
,-P	不変式の先行評価の最適化をしない。
,-u	ループアンローリングをする。
,-U	ループアンローリングをしない。
,-e	演算の評価方法を変更する最適化をする。
,-E	演算の評価方法を変更する最適化をしない。

例 19 最適化レベルを基本的なものにとどめ、ただし不変式の先行評価をする場合は以下のようにします。

```
frt -Ob,-p -Ps -Wv,-m3 a.f
```

また、関数のインライン展開に関するオプションが以下のように指定できます。

-Ne	組込関数と実行文の数が 30 以下の外部手続きをインライン展開する。

さらにベクトル命令が最大限利用可能なようにオブジェクトの配置を調整するオプションがあります。

-Aa	オブジェクトの配置を調整する。

どこまで効くかわかりませんが次のことも指定できます。

-Wv,-p2600	VP2600 に最適なコードを出力する。

できる限りのことを全部すると次のようになるでしょう。

例 20 (できる限りの最適化を試みる) できる限りの最適化を試みます。ただし、マニュアルを見ればもう少し細かいことができます。

```
frt -Aa -Ne -Of -Ps -Wv,-p2600,-m3 a.f
```

3.6.3　最適化に関するメッセージを出力するオプション

最適化のレベルが適切かどうかということを知るには、最低リストと実行結果、必要ならばアナライザ afrt の出力結果を眺める必要があります。やみくもに上記オプションを指定してもうまくいきません。最適化に関するメッセージは次のオプションで出力が可能です。他人に聞く前にコンパイラが出してくれるメッセージをにらみましょう。

まずベクトル化できなかった箇所とその原因を出力してくれるありがたいオプション。

-Wv,-m3	ベクトル化できなかった原因とベクトル化の状況を出力する
-Wv,-m2	ベクトル化できなかった原因にしぼって出力する

次のオプションは個々の最適化に関する情報を出力するものです。

-Ei	インライン展開に関するメッセージを出力する
-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する
-Eu	ループアンローリングに関するメッセージを出力する
-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する

最適化、ベクトル化情報を最大限に得たい場合、frt のオプションは次のようにします。

例 21 (メッセージの嵐にする)

```
frt -Oe -Ne -Eipue -Ps -Wv,-m3 a.f
```

と、ここまで脅かしておいて、と。

コンパイラはあまりうるさいことをいわなくとも -Oe [5]とだけ指定すれば適切な処理をするように構成されています。不安な人は -Ob と指定してもよろしい。ただし、不本意ながら最適化の細かい設定をしなければならない場合がたまにあります。

3.6.4　最適化のレベルを調節する必要に迫られる場合

最適化のレベルを調節する必要に迫られる場合は最適化のレベルを上げたり下げたりすると実行結果が変わり、極端な場合異常終了する場合がある場合です。これはコンパイラの虫ではありません。最適化レベル -Oe 以上は実行に際して副作用のある場合でもかまわずやってしまうからです。たとえば不変式の先行評価は最悪の場合異常終了を引き起こす場合がありますし、式の評価順序の変更は精度に微妙な影響を与えます。

この時にはレベルを下げれば良い (-Ob や -Wv,-ad) のですが、すると当然ベクトル化されるべき部分がベクトル化されていない場合が存在します。その時は上で説明したように微調整が必要です。頑張って下さい。あ、そうそう微調整と言うくらいですから対象プログラムへの深い理解が当然必要です。

ポイント 14 最適化のレベルを微調整するのはプログラムリスト、afrt の結果を見た上で自分で責任を持つことのできる人がやること。

[5] -Wv オプションを指定した場合、デフォルトでこうなります。ただし、陽に指定することをお勧めします。

3.6.5 ソースプログラムに書き込む

さらにぎりぎりまで最適化する必要のある人はマニュアルを見て下さい。ここでは詳細については説明しません。

ソースプログラムに *VOCL ではじまるコンパイラ制御行を挟み込めばサブルーチン / ループ単位での最適化が可能です。ただし、以下のことが経験的に知られています。

ポイント 15 *VOCL 制御行を挟み込む場合は後述の afrt や tuner の助けを借りること。また、挟み込まなければ最適化できないようなプログラムはどこか変なことをしている場合が多いから考え直した方が保守その他の点で結局得であることが多い。

3.7 マニュアルとまとめ

以下のマニュアルにはここで説明されていないことも含めていろいろ詳しく説明されています。

- UXP/M FORTRAN77EX/VP 使用手引書 V12 用, 富士通,1991. 94SP-5030

数値計算ライブラリのマニュアルは次の通りです。

- (SSL II) FACOM FORTRAN SSL II 使用手引書, 富士通,1989. 99SP-0050.
- (SSL II VP) FUJITSU 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), 富士通,1991. 99SP-4070.
- (NUMPAC) ライブラリ・プログラム利用の手引 (NUMPAC Vol. 1,2,3) 名古屋大学大型計算機センター (MSP の MANUAL コマンドで出力可能)

まとめです。

1. Fortran は VP の主力言語である。利用者は Fortran を使うことに対して何ら恥じることはない。
2. 自動ベクトル化 Fortran コンパイラとして frt が用意されている。コマンドとして frt -Ps -Aa -Wv,-m3 をおぼえておけば良い。ここで -Wv を省略してはいけない。
3. ライブラリとして SSL II, NUMPAC, LAPACK が提供されている。
4. 最適化の程度は自分で調整できる。調整するのはリストやアナライザの出力を見て自分の責任ですること。
5. ソースプログラムに *VOCL 制御行で書き込むことでも最適化のレベルを調節できる。

4 デバッグ

VP は当面 TSS のサービスをせず、従って対話的デバッガが使えません。そこでデバッグというプロセスはちょっと特殊になります。でもデバッグの基本は変わりません。

4.1 一般的な注意

ポイント 16 デバッグは自分でするのが基本。

そうはいってもどうにもつまって他人の助けを借りたい場合があります。その時の注意を以下に述べます。センターまたはプログラム相談員に対して、長〜いリストを放り出して一言「動かないんですけど...」という人がごく一部存在することを聞いています。デバッグに当たっての一般的な注意で重要なものを 2 つあげておきましょう。

まず最初。

ポイント 17 自分の書いたプログラムを他人に理解してもらうためにはそれなりの努力が必要である。

DO ループの制御変数以外で II とか KKK などというわけのわからない変数名を用いている人は意味のある名前に書き換えてから相談するのが礼儀です。そしてもうひとつ。

ポイント 18 自分の理解の範囲を越えたプログラムを書いてはいけない。

他人の質問に対して答えられないプログラムを持ち込むのは反則です。

4.2 デバッグのプロセス

さて、以下は VP でのデバッグのプロセスです。

4.2.1 M1800 でのデバッグはお済みですか?

M1800 の UXP/M 上でサービスしている frt はベクトル命令に関連する部分以外は VP の frt と同じです。つまらない文法エラーや論理面でのエラーはまず M1800 上でデバッグして下さい。

4.2.2 最適化のレベルは適切ですか?

M1800 上では正しく動くが VP2600 上では正しく動かない時、まず疑うのは最適化が悪さをしているのではないかと言うことです。最適化のレベルを調節して下さい。

4.2.3 デバッグ用にオプションを調節しましょう

プログラムをコンパイルする時にいろいろなチェックをするコードを挟み込むことができます。ただし、当然のことながら遅くなります。

-Da	実引数と仮引数の対応チェック
-Db	サブルーチンをトレースする
-Di	値の割当を表示する
-Do	整数演算のオーバーフローのチェック
-Ds	配列の添字チェック
-Dt	文番号を持つ文をトレース
-Du	未定義データの引用をチェック
-Dv	重複実引数の値変更をチェック

例 22 (デバッグ用) 実引数と仮引数の対応チェック、サブルーチンのトレース、配列の添字チェックをする (-Wv,-ad は必須)。

```
frt -Dabs -Wv,-ad a.f
```

4.2.4 コンパイラの虫では?

ごくまれにですが (?) コンパイラが正しい動作をしないで誤ったコードを出すことがあります。その場合はセンター備えつけの調査依頼票でセンターまでお知らせ下さい。ただし、

ポイント 19 人間は往々にして間違いを他人のせいにする。

コンパイラは抗弁しませんから人間側はとかく図にのりやすい。

4.3 計算誤差との戦い

数値計算では誤差の評価はいつも問題になります。演算の順序を変更しただけで結果に差が出てくることがあります。(例えば $1.0\mathrm{E}^7 + \underbrace{1.0\mathrm{E}(-8) + \cdots 1.0\mathrm{E}(-8)}_{10^7\mathrm{times}}$ と $\Sigma_{i=1}^{10^7} 1.0\mathrm{E}(-8) + 1.0\mathrm{E}(7)$ は数学的には等価でも計算機の世界では一般的には等価になりません)。

ポイント 20 浮動小数点演算では誤差にも注目すること。

例 23 上の発言が嘘でない証拠をお見せしましょう。プログラムは *C* で書いています。

```
kyu-cc:22] cat de.c
main()
{
        double x = 10000000.0;
        double y = 0.0;
        int i;

        for (i = 0; i < 10000000; i++) {
                x += 0.00000001;
                y += 0.00000001;
        }
        y += 10000000.0;
        printf("%f <> %f\n", x, y);
}
kyu-cc:23] cat a.vp
vcc -Wv,-m3,-Ps,-ad dd.c
a.out
kyu-cc:24] qsub -q vs a.vp
...
kyu-cc:25] cat a.vp.o765
Warning: no access to tty; thus no job control in this shell...
line     vo *** source list ***
...
10000000.097789 <> 10000000.100000
kyu-cc:26]
```

ところが VP の世界では最適化のために演算順序を変更することがよくおこります。それが原因となって計算結果が異なってくる場合があります。

これも証拠をお見せしましょう。

```
kyu-cc:26] cat b.vp
vcc -Wv,-m3,-Ps dd.c
a.out
kyu-cc:27] qsub -q vs b.vp
...
kyu-cc:28] cat b.vp.o766
Warning: no access to tty; thus no job control in this shell...
line     vo *** source list ***
...
10000000.100000 <> 10000000.100000
kyu-cc:29]
```

この差は -ad というオプションです。これは何をするものか? [6]

もし、(1) それが原因と信じるに足る理由があって、しかも (2) プログラムは計算誤差に注意して書かれたものならば次のオプションを試して下さい。

例 24 (式の評価順序を変更しないオプションの指定) -Wv,-ad オプションの意味は「文の評価順序を変更する最適化と式の評価順序を変更する最適化を両方抑止する」です。

```
kyu-cc:35] cat a.d.vp
cd vpjob
frt -Ps -Wv,-ad a.f
timex -H a.out
kyu-cc:36]
```

4.4 まとめ

さて、まとめです。

1. デバッグは自分でするのが基本。
2. M1800 でデバッグを済ませておこう。
3. 最適化のレベルを調整する必要のある場合がある。
4. デバッグ用のコードを生成させて調べることもできる。

5 性能改善とチューニングへの旅

スーパーコンピューティングにおけるプログラミングと一般的なプログラミングとの違いは

ポイント 21 スーパーコンピューティングにおいては正しいだけでは意味がない。速くなければ価値がない。

に尽きるでしょう。性能改善やチューニングは避けて通れない道です。VP 側ではこれらをサポートするためにアナライザ afrt と呼ばれる性能評価ツールとチューナ tuner と呼ばれるチューニングツールを提供しています。

でもその前にプログラムがどんな風にコンパイルされているかを知っておくことが重要です。プログラムリストをながめることはスーパーコンピューティングでは一定の意義を持っています。

[6] 答えはすぐ後。

5.1 リストをながめる

-Ps -Wv,-m3 オプションをつけてコンパイルするとベクトル化の状況を示すリストが標準エラーに出力されます。まずはそれをじっくりながめて下さい。ごく簡単ななめ方を述べましょう。

例 25 (リストをながめる) リストは標準エラーに出力されます。ここでは a.vp.e56 に結果が返って来ています。

```
kyu-cc:21] more a.vp.e56
         fortran77 ex/vp    v12l10              date 92-12-10   time 06:27:27

 isn
00000001
00000002         ************************************************************
                    .
                    .
                    .
00000044      v        DO 1 I=1,M
00000045      v         T=(DINT(DBLE(I-1)/DBLE((L-1)**2))+1.0D0)*H
00000046      v         R1=MOD(I-1,(L-1)**2)+1
00000047      v         X=(DINT(DBLE(R1-1)/DBLE(L-1))+1.0D0)*H
00000048      v         Y= 1.0D0-(DMOD(DBLE(R1-1),DBLE(L-1))+1.0D0 )*H
00000049
00000050      v         INIT(I)=CONST*SIN(PI*X)*SIN(PI*Y)*T
00000051
00000052         *      PRINT'(2X,I3,1X,F10.5,F10.5,F10.5,F13.8)',I,X,Y,T
00000053      v  1     CONTINUE
```

ここでプログラムリストの頭の方に記号 v が出てきました。これは対象となる DO ループがベクトル化されたことを意味します。プログラム単位の終りにこれらの情報がまとめて出力されます。ちょっとのぞいてみましょう。

```
                  ...
fortran77 ex/vp   diagnostic messages: program name(para)
jwd8220i-i                Optimization with possibility of side effect.
fortran77 ex/vp   vectorization messages: program name(para)
jpc1001i-i isn:00000045 - 00000053   Vectorized by DO variable I.
jpc1001i-i isn:00000063 - 00000063   Vectorized by DO variable I.
jpc1001i-i isn:00000065 - 00000065   Vectorized by DO variable I.
jpc2303i-i isn:00000067              DO variable may be defined.
                  ...
```

記号としてはその他に s と m があります。それぞれについて説明します。

v	DO ループに対してベクトル化されたコードを出力した
m	DO ループに対してベクトル化されたものとされないものが混在したコードを出力した
s	DO ループに対してベクトル化されないコードを出力した (ベクトル化をあきらめた)

その他にこういうのもあります。

例 26 (ループアンローリング) v や s のあとについている数字 n(この場合 2) はループアンローリングを n 重やる最適化を実行したことを意味します。

```
00000061      v2        DO 2 I=1,M
00000062      s2         DO 3 J=1,M
00000063      v2          Q(I,J)=0.0D0
00000064      s2 3      CONTINUE
00000065      v2 2      CONTINUE
```

ここまではうまくいった例です。問題はうまく行かなかった場合です。次の例を見ましょう。

例 27 (ベクトル化されなかった DO ループの解析) リストが

```
          ...
00000162      s         DO 9 I=1,M
00000163      s         S1=Q(I,I)
00000164      s         Q(I,I)=1.0D0
00000165      v         DO 10 J=1,M
00000166      v         Q(I,J)=Q(I,J)/S1
00000167      v  10     CONTINUE
00000168      s         DO 11 K=1,M
          ...
```

でそれに対するメッセージが

```
          ...
jpc2310i-i isn:00000163              S1 used in inner DO loop.
jpc2310i-i isn:00000163              Q used in inner DO loop.
jpc2310i-i isn:00000164              Q used in inner DO loop.
          ...
```

と出てきました。これにはベクトル化をあきらめた言い訳が書いてあります。当然ベクトル化されると思っていたのに s がついて返ってきた場合、「どうしてだろう」と思うところから性能改善の旅が始まります。

メッセージの意味がわからないという人もいるでしょう。そういう人のためにここでは標準的な教科書をあげましょう。

- 島崎眞昭, スーパーコンピュータとプログラミング, 計算機科学 / ソフトウエア技術講座, 共立出版

これを読めばメッセージの意味は大抵わかるようになります。またベクトル化を推進するためのテクニックがたくさん書いてありますから一読の価値はあります。あ、そうそう

ポイント 22 著者はセンターの研究開発部長である:-)

5.2 一般的な性能改善方法

さて、プログラムがどんな風にコンパイルされているかはわかりました。次のプロセスはスピードアップを主とした性能改善です。そのプログラムが1回流して終りというものならばこのプロセスは実はあまり重要ではありません。しかし、多くのプログラムはパラメタを変えて複数回流すように作ります。その場合、徹底的な性能改善は結果としてプラスになることが多いのです。

その時に一度性能評価をしておくとどこにエネルギーを注ぐと最も効率よくスピードアップが図れるかということがわかります。

一般的なスピードアップの手順として次のようにすれば良いでしょう。

1. プログラム中で最も時間のかかる部分を見つける。
2. その部分のベクトル化の度合を見る。
3. NUMPAC や SSL II のルーチンを積極的に利用する。
4. ベクトル化の度合が低い場合は、アルゴリズムの改善または *VOCL 制御行の挿入でベクトル化する。
5. ベクトル化の度合が高くてもベクトル長がもっと長くできないかなどを検討する。
6. どうしてもだめだったら M1800 で流す。特に繁忙期には待時間を考えると M1800 の方が早く結果が得られることがある。

5.3 アナライザ

アナライザ afrt は上記を支援することを目的とするツールです。プログラムのループ単位までをみて各部分がどのくらいのコストで実行されるかを計測します。

アナライザを実行するには以下のようなスクリプトを書きます。

例 28 (afrt を実行させるスクリプト) ディレクトリ vpjob 中でプログラム a.f をアナライザにかける場合。以下の場合は見積り解析が出力されます。

```
kyu-cc:63] cat a.afrt.vp
cd vpjob
afrt -S -p2600 -Po -Wc,-Wv a.f
kyu-cc:64]
```

5.4 アナライザの出力の解析

出力の内、見て損はないものは v-leng、 v-rate と v-effect です。プログラム、サブルーチン、ループ毎の情報が出てきます。あとの情報は人間が見るものと言うよりはチューニング用ツールが見るものですからとりあえず無視してよろしい。以下に afrt の出力例をあげます。

例 29 (afrt の出力例) 出力は一般に長いですから utoprint を利用してセンター2階の NLP に出力することをお勧めします。

```
kyu-cc:22] utoprint a.vp.o322
utoprint : output to NLP
```

右端が切れているのでちょっと見にくいですが...

解　　説

```
Warning: no access to tty; thus no job control in this shell...

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

name        ex-count     v-cost     %     s-cost     %     v-leng  v-rate      v-effect
MAIN            *1    *2206017  54.6   *2206017   4.3        *5    *0.0    *1.0-  1.0
SETTBL          *1    *1210793  30.0  *33097172  65.1      *100   *99.8   *22.2- 34.0
SETLST          *1     *526458  13.0  *13082014  25.7      *100   *99.4   *20.5- 30.2
CF22M           *1      *42546   1.1   *1177955   2.3       *99   *99.8   *22.4- 34.6
CF23M           *1      *39020   1.0   *1114065   2.2       *99   *99.9   *23.0- 35.9
SFOOM           *1       *5460   0.1    *131016   0.3       *99   *99.2   *19.9- 29.0
CF2             *1       *3030   0.1     *16358   0.0       *84   *84.4    *5.2-  5.6
FFT2            *1       *2845   0.1      *4583   0.0       *67   *39.3    *1.6-  1.6
CF24M           *1        *527   0.0     *14435   0.0       *67   *99.8   *22.2- 34.1
INIT            *1        *505   0.0     *12384   0.0       *51   *99.3   *20.3- 29.8
EREVL           *1        *278   0.0      *6363   0.0       *51   *99.0   *19.2- 27.3
FFTSUB          *1        *206   0.0      *5615   0.0       *51   *99.7   *22.1- 33.8
SETDAT          *1         *40   0.0        *909   0.0       *51   *99.0   *19.2- 27.3
(total)   ---------    *4037726 100.0  *50868886 100.0  ---------   *95.3   *11.4- 13.8

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

routine     ex-count     v-cost     %     s-cost     %     v-leng  v-rate      v-effect
MAIN            *1    *2206017  54.6   *2206017   4.3        *5    *0.0    *1.0-  1.0

vectorize - loop list ------- MAIN ------------------------------------------------------------------------------

isn               do-id  kind   v   ex-count     v-cost     %     s-cost     %     v-leng  v-rate      v-effec
00000016-00000069 1000   do            *1000   *1720000  78.0   *1720000  78.0      *10    *0.0    *1.0-  1.
00000064-00000065 200    do     S     *10000    *480000  21.8    *480000  21.8       *4    *0.0    *1.0-  1.
00000014-00000070 3000   do               *1      *6000   0.3      *6000   0.3    *1000    *0.0    *1.0-  1.
00000001-00000081 -----  total            *1        *17   0.0        *17   0.0       *1    *0.0    *1.0-  1.

vectorize - statement list -- MAIN ------------------------------------------------------------------------------

isn       ex-count   true    v-cost    s-cost v o c--------1---------2---------3---------4---------5---------6-
00000001                                                 IMPLICIT REAL*8 (A-H,O-Z)
00000002                                                 PARAMETER (NMAX=5000,MMAX=12)
00000003                                                 COMPLEX*16 C(NMAX),OMEGK(NMAX,MMAX,2),X(NMAX)
00000004                                                 COMPLEX*16 C1(NMAX),C2(NMAX)
00000005                                                 REAL*8 ER(NMAX)
00000006                                                 INTEGER LIST(NMAX*MMAX)
```

5.4.1　v-effect が低い場合

v-effect は (ベクトルユニットを使わない場合の実行時間 ≈ M1800 での実行時間)/ (VP2600 での実行時間) のことです。これが 2 倍もいかないようだったら真剣にチューニングに取り組む必要があります。10 倍以上だったらまずは合格といったところでしょう。

5.4.2 v-rate

5.4.3 v-leng

v-rate が高くても v-length が小さいとベクトル化の効果は薄れます。

5.5 チューニング

具体的なチューニングのために M1800 側にツール tuner を用意しています。

例 30 (チューナ使用のための準備) tuner を実行する場合、あらかじめアナライザ afrt でチューナ用の情報ファイルを作っておく必要があります。afrt の -Po オプションはこのファイルを作成することを指定します。

```
kyu-cc:24] cat a.afrt.vp
cd vpjob
afrt -p2600 -S -Po a.f
kyu-cc:25] qsub -q vs a.afrt.vp
Request 56.kyu-cc submitted to queue: vs.
     ...

kyu-cc:29] ls -l a.pinf
-rw-------   1 a79999a  other     336021 Dec 15 18:46 a.pinf
kyu-cc:30] tuner a.pinf
>
```

ソースファイル a.f に対しては情報ファイル a.pinf が作成されます。それを tuner に喰わせてやればチューニングがはじまります。

詳細は (筆者も) マニュアルをみないとわかりません。マニュアルを参考に自分でチューニングの道を極めて下さい。

5.6 マニュアルとまとめ

まずマニュアルです。

- UXP/M アナライザ使用手引書 (FORTRAN,VP 用) V10L20 用, 富士通,1992. 94SP-5080.
- UXP/M チューナ使用手引書,V10L10 用, 富士通,1992. 94SP-5020.

次はまとめ。

1. チューニングのためにアナライザ afrt が用意されている。afrt -S -p2600 -Po -Wc,-Wv としておぼえておけば良い。
2. アナライザの出力で注目すべきは v-leng、v-rate と v-effect である。
3. チューニングのためのツールとして tuner がある。

6 C/VP コンパイラ

6.1 それでもまだ C ですか?

VP 用の記述言語はもともと Fortran が主流でした。それはいくつか歴史的な理由がありますが、今になって思えば Fortran 自身ベクトル計算機用の最適化がしやすい言語だったことが大きな理由の一つ

です。それに比べてCはちょっと面倒です。ベクトル計算機のコンパイラを作る人にとっては不倶戴天の敵ともいえるポインタがあるのでどうしても性能的に問題が出やすい言語です[7]。ところがUnixでのスーパーコンピューティング環境が整備されるに従いUnixでの主力言語であるCでプログラムを書きたい人が出てきました。ということで仕方なく(と思うんですがねえ)ベクトル化Cコンパイラが出てきました。

ベクトル化Cコンパイラの守備範囲はそんなに広くありません。普通のユーティリティのソースをコンパイルしてもいきなり10倍の速度が得られるわけではありません。やはり数値計算用のプログラムを念頭に作られているようです。

というわけで、みなさん、できるだけFortranを利用して下さい。でなければモルモットになることを覚悟で使って下さい。何年か後にFortranなみのコードを出力するコンパイラが出現するかもしれません。

6.2 ベクトル化される範囲

ベクトル化されるループはforループのうち、以下の条件をみたすものです。

1. 一般に変数N、MとIに対して以下のように書けるもの、およびそのバリエーションがベクトル化可能です。Iは定数であることが望ましい。

```
for(i = N; i < M; i += I) {
    ....
}
```

   変数Mは途中で値が変わってはいけません。Iが変数の場合は後述しますが-Bsオプションを使って無理矢理ベクトル化することも可能です。その場合も途中で値が変わってはいけません。

2. 実行開始時に繰り返し回数が確定することが必要です(ただし途中での飛び出しは1回まで可)。

3. ループの途中への飛び込みがあってはいけません。

4. 蛇足ですがdoループ、whileループはベクトル化の対象外です。

もうおわかりでしょう。Fortranに無理矢理直した時にDOループに直せるものがベクトル化可能範囲です。さてあなたはそのようなプログラムを持っていますか?

6.3 コンパイルの仕方

コマンド名はvccです。リクエスト用のスクリプトは以下のようにします。なお、-Wvオプションは Fortranと共通です。

例 31 (Cソースプログラムをコンパイルし実行させるリクエスト) プログラムリストを出力するオプション(Fortranでは-Ps)が-Wvオプションに続けて指定するようになっていることがFortranとの違いです。

[7] Cの名誉のためにちょっといっておくと、注意深くプログラミングされたコードならばCでもちゃんとした性能が出ます。

```
cd dir
if test a.out
then
rm -f a.out
fi
#
vcc -Wv,-m3,-Ps a.c
timex -H a.out
```

標準エラーにリストがベクトル化に関するメッセージとともに出力されます。

6.4 デバッグと解析用のツール

Fortran の afrt、tuner に対応するツールはありません。

6.5 C で良いコードを出すために

以下はよいベクトル化コードを出すために気をつけることのうち、C に特有のことです (もちろんすべてをつくしているわけではありません)。

- for ループは「素直に」書く。増分値を変更したりするとベクトル化されない。
- 配列の添字もできるだけ単純に書く。

  そう、ここまでは Fortran の DO ループのつもりで for ループを書くことを心がければ大丈夫ということです。
- ポインタの使用は必要最小限にする。あっちこっちのオブジェクトを飛び回るようなポインタはベクトル化 C コンパイラにとって不倶戴天の敵です。もっともこれもトリッキーなことをしなければ良いのですが。

6.6 ループの制御をするオプション

C の for 文は Fortran の DO 文と異なりかなり勝手なことが書けるので人間がコンパイラの手助けをしないとベクトル化できるのにできないとあきらめることがあります。特に変数でループ変数の増分を決めている場合に注意して下さい。

-Wv,	-Bs	for 文の条件文が常に真というループがなく、 さらにループ変数の増分が負にならない (逆向きループがない) としてベクトル化する。
	-Bf	for 文の条件文が常に真というループがなく、 さらにループ変数の増分が負になる可能性がある (逆向きループがある)。 としてベクトル化する。
	-Bn	for 文の条件文が常に真というループがあり、 さらにループ変数の増分が負になる可能性がある としてベクトル化する。
	-s*n*	繰り返しの回数が *n* 回未満のループのベクトル化をやめる。

6.7　さらなる最適化のために

Fortran の *VOCL 制御行に対応するものは C では #pragma ディレクティブです。詳細はマニュアルを参照して下さい。マニュアルでこれらディレクティブを挿入することは勧められません。誤った実行結果になる可能性があるからです。しかしその危険性をおかしても最適化しなければならないものはあることは承知しています。そういう方は、自らの責任においておやり下さい。

6.8　マニュアルとまとめ

- C/VP 使用手引書 V11 用, 富士通,1992. 94SP-5070.

まとめです。

1. C は注意深くプログラミングする必要がある。

7　後処理

スーパーコンピューティングにおいて後処理の重要性は年々認識されるようになってきています。数字の羅列よりは具体的に絵、アニメーションの形にした方が説得力も増すというものです。ここでは VP2600 でバッチ処理をした後、M1800 の UXP で後処理することを考えます。

7.1　グラフィックスのためのプラットフォーム

センターの Unix 環境ではプラットフォームとして次にあげるグラフィックス環境があります。

- X Window システム。
- PostScript プリンタ。
- Tektronix 端末。
- Sunview。

7.2　センター整備のビジュアライザ

グラフィックスとビジュアリゼーションは現在では区別して考えるのが普通です。

ポイント 23 プログラミングを必要とせずにパラメタと数値を入力して科学的に意味のある絵を出力することをビジュアリゼーションと言い、そのためのパッケージソフトウエアを一般にビジュアライザと言う。

現在センターのビジュアリゼーション環境は整備が (残念ながら) 進んでいません。
上の定義からいくとかろうじてビジュアライザと呼べなくもないものには次のものがあります。

7.2.1　gnuplot

gnuplot は多くの関数、コマンドを備え対話的にプロットができるソフトウエアです。詳細はドキュメントを眺めて下さい。

例 32 (gnuplot のドキュメントを出力する) ドキュメントは `/usr/local/doc/gnuplot.dvi` です。出力は以下のようにします。

```
kyu-cc:21] dvi2ps /usr/local/doc/gnuplot.dvi|lp -dps -Tps
 [/usr/local/lib/dvi2ps/tex.ps]
Prescanning ...............................
Reading font info ................
[1] [2] [3] [1] [2] [3] [4] [5] [6] [7]
[8] [9] [10] [11] [12] [13] [14] [15] [16] [17]
[18] [19] [20] [21] [22] [23] [24] [25] [26] [27]
[28] [29]
request id is ps-3872 (standard input)
kyu-cc:22]
```

7.2.2 S

S はもともと汎用の対話的統計パッケージですが、グラフィックス環境も充実しています (統計に関係するグラフの描画 (3 次元を含む) が得意です)。参考書がありますからそれを参考にして下さい。

7.2.3 MENTAT

MENTAT はセンターの qviss 上で動くパッケージであり、構造解析用パッケージ MARC(これは MSP) の前処理と後処理を担当します。これは使いこなせばもしかしたらビジュアライザとして使えるかも知れません (無責任)。

7.3 グラフィックス環境を利用してビジュアリゼーションをする

ある程度整備されたグラフィックス環境があればそれを利用して自分で簡単なビジュアライザを構築することができます。これからは Tektronix 環境より X ウインドウや PostScript を対象にした方が良いでしょう。

現在センターで提供しているグラフィックスに関するライブラリは以下の通りです。

ライブラリ	性格	ドキュメント
X ライブラリ	一番低レベルなルーチン群	参考書山のようにあり
VOGLE	3D グラフィックスルーチン群	/usr/local/doc/VOGL
VOPL	VOGLE を利用したプロッタルーチン群	/usr/local/doc/VOGL
VORT	VOGLE を利用したレイトレーサ、画像処理ルーチン	/usr/local/doc/VOGL

7.3.1 X ライブラリ

X11R4 のライブラリが提供されています。コンパイル時にはオプションとして -lsocket -lnsl をつけて下さい。

例 33 (X ライブラリをリンクする) 例として libX11 のみを使用する場合をあげます。その他 Xaw、Xau、Xdmcp、Xext、Xinput、Xmu、Xt がありますから必要に応じてリンクして下さい。

```
kyu-cc:24] cc xtest.c -lX11 -lsocket -lnsl
kyu-cc:25]
```

7.3.2 VOGL

VOGL は IRIS の GL に良く似たライブラリ (らしい) です。業界ではかなり有名らしいのですが... 3D 用のプロッタルーチンで現在 X Window、PostScript、Tektronix に対応しています。

例 34 (VOGLE をリンクする) Fortran から呼び出す場合。

```
kyu-cc:26] frt vogltest.f -L/usr/local/lib -lvogle -lX11 -lc -lm -lsocket -lnsl
kyu-cc:27]
```

C の場合 -lc は不要です。

例を /usr/local/doc/VOGLE/examples.vogle においておきます。参考にして下さい。

7.3.3 VOPL

VOPL は VOGLE をベースにしたプロッタルーチンです。

例 35 (VOPL をリンクする) Fortran から呼び出す場合。

```
kyu-cc:26] frt vopltest.f -L/usr/local/lib -lvogle -lvopl -lX11 \
-lc -lm -lsocket -lnsl
kyu-cc:27]
```

C の場合 -lc は不要です。

例を /usr/local/doc/VOGL/examples.vopl においておきます。参考にして下さい。

7.3.4 VORT

VORT は VOGLE をベースにしたレイトレーサ、画像処理ルーチンです。ライブラリの他にいろいろツールが用意されていてそれを用いれば動画も作成できます。ただしこれは現在 X のカラーにしか対応していません。つまり白黒のディスプレイ上では動きません。

ツールは /usr/local/bin/vogle の下、ライブラリは /usr/local/lib/libvort.a、ドキュメントは /usr/local/doc/VOGL にあります。参考にして下さい。

7.3.5 番外 – qviss 上の PHIGS+

qviss は Sunview 用のワークステーションです。ここに PHIGS+ がインストールされているのでこれを用いて 3D グラフィックスができます。

例 36 (qviss で PHIGS を使う) コンパイル、リンクの仕方は次の通りです。

```
qviss:2] cc test.c -lphigs -lpixrect
qviss:3]
```

7.4 参考文献

- S システム (I)(II)、共立出版