VPP5000/64におけるベクトル化について

永井　亨

Ⅰ．はじめに

当センターのスーパーコンピュータシステムFujitsu VPP5000/64は64台のPE（prosessing element）で構成される分散メモリ型のベクトル並列計算機です。すなわち，各PEはベクトル演算機能を備えたプロセッサ（ベクトル計算機）です。このベクトル演算機能を使うことをベクトル化とよびます。VPP5000でうまく並列化できたとしても各PEでのベクトル化効率がよくないと全体としての並列効果が低下してしまいます。VPP5000にはFortran，C及びC++の自動ベクトル化コンパイラが備わっています。本稿ではFortranを例にとってVPP5000のベクトル化について解説します。

Ⅱ．演算パイプライン方式

高速処理を必要とする科学技術計算は最終的には行列演算に帰着する場合が多く，対象となるデータはFortranでは配列で表現されます。１次元配列で表現されるデータをベクトルデータまたはベクトルとよびますが，ベクトル計算機の特長はこのベクトルデータを高速処理する（ベクトル処理する）ところにあります。

まず逐次処理について説明します。１つの命令の実行が，命令の読み出し及び解読（a），アドレスの計算（b），オペランドの読み出し（c），演算の実行（d）の４つの段階に分けられるとすると，処理の流れは $a \rightarrow b \rightarrow c \rightarrow d$ を１サイクルとして，これを命令の数だけ繰り返すことになります。したがって，処理の時間的な流れは，

命令１　$a \rightarrow b \rightarrow c \rightarrow d$
命令２　　$a \rightarrow b \rightarrow c \rightarrow d$
命令３　　　　$a \rightarrow b \rightarrow c \rightarrow d$
⋮　　⋮

のようになります。演算処理を高速にするためにそれぞれの段階を時間的にオーバーラップさせて処理するパイプライン処理方式があります。これは，例えば，自動車の生産をいくつかの工程に分けて流れ作業で行うことを思い浮かべるとよいでしょう。パイプライン処理の場合には，

命令１　$a \rightarrow b \rightarrow c \rightarrow d$
命令２　　$a \rightarrow b \rightarrow c \rightarrow d$
命令３　　　　$a \rightarrow b \rightarrow c \rightarrow d$
⋮　　⋮

のような流れになり，それぞれの段階が時間的にオーバーラップして処理されるため演算実行（d）の部分が連続的に処理されることになります。

演算の実行部分をもう少し詳しく見てみましょう。浮動小数点数（Fortranでいう実数）は符号部 s，指数部 e，仮数部 f を用いて表現され，倍精度の場合，データの値は $(-1)^s \times 2^{e-1023} \times (1.f)$ となります。２つの浮動小数点数の加算や乗算などの演算，すなわち浮動小数点演算は，例えば加算を考えると，指数部の比較（　），仮数部の桁合わせ（　），仮数部の加算（　），加算後の正規化（　）の４つの動作に分解することができます［1］。　～　の動作が１つの演算実行を構成するとすると，パイプライン処理で浮動小数点加算を連続実行させたときの流れは，

加算１　→　→　→
加算２　　→　→　→
加算３　　　→　→　→
⋮　　⋮

となります。

演算の実行自身をオーバーラップさせて高速演算処理を図るのが演算パイプライン方式です。演算パイプライン方式ベクトル計算機での浮動小数点加算の流れは，

加算１　→　→　→
加算２　　→　→　→
加算３　　→　→　→
⋮　　⋮

となります。演算パイプライン処理を行うハードウェアをパイプライン演算器または単にパイプラインとよびますが，ベクトル計算機ではベクトルデータをパイプラインに流し込むことによって，同一演算を高速に実行することができるのです。

Ⅲ．ベクトル計算機の基本構成

ベクトル計算機の概念図を図１に示します。ベクトル計算機はベクトル処理ユニットとスカラ処理ユニットで構成されます。ベクトルデータを用いて演算パイプライン処理を行うための命令をベクトル命令と呼び，ベクトル命令を処理するハードウェアがベクトル処理ユニットです。ベクトル処理ユニットでは一般に，

１．必要なデータを主記憶からベクトルレジスタにロードする。
２．ベクトルレジスタ上のベクトルデータをパイプライン演算器に流し込み，演算結果をベクトルレジスタに書き込む。
３．ベクトルレジスタ上の最終結果を主記憶にストアする。

のような流れの処理が行われます。主記憶とベクトルレジスタの間のデータ転送にはロードパイプライン，ストアパイプラインが用いられ，これもベクトル命令により処理されます。

パイプライン演算器はステージと呼ばれるいくつかのセグメントに分割されており，データが各ステージを通過していくうちに先ほど述べたような演算処理が行われます。通常，ベクトル処

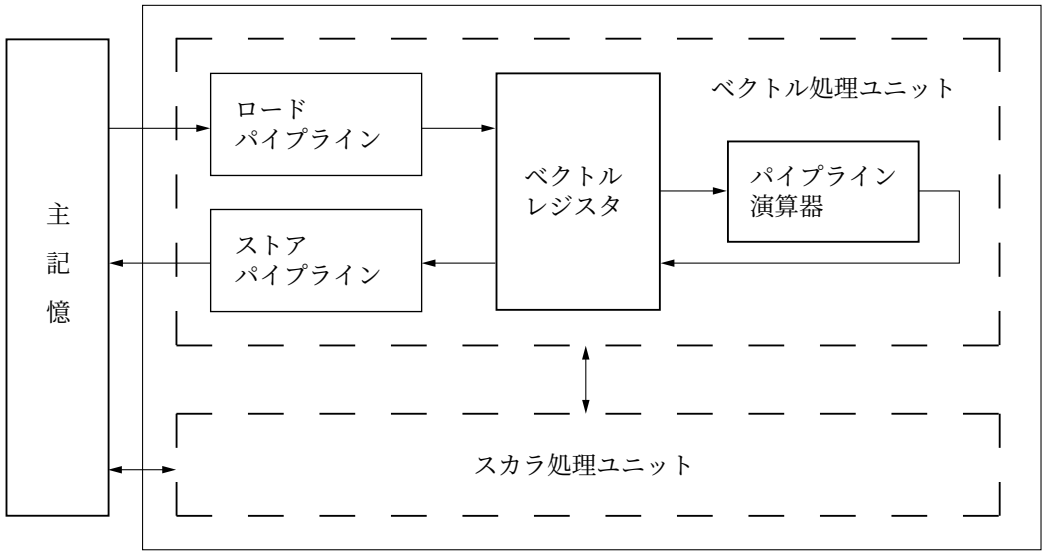


図１　ベクトル計算機の概念図

理能力を高めるため，異なる演算に対して独立に並列動作ができるパイプライン演算器（例えば，パイプライン加算器，パイプライン乗算器，パイプライン除算器など）を設けたり，複数本の同一演算用パイプラインを設けて多重化したりする方式がとられます。

利用者が作成するプログラムのすべての部分がベクトル命令により処理されることはまずありません。ベクトル処理に関係しない命令（スカラ命令）を処理する部分がスカラ処理ユニットです。プログラムのコンパイル，リンケージ等もスカラ処理ユニットで行われます。また，多くの場合，スカラ処理ユニットはスカラ処理のほか，ベクトル処理ユニットを含めたシステム全体の制御も行います。システム全体の処理能力を高めるため，スカラ処理の高速化も重要です。

Ⅳ．ベクトル計算機の性能

ベクトル計算機の性能を文献［２］を参考にしてモデル化してみることにします。例えば，ベクトル長 n のベクトルデータ $\boldsymbol{a}$，$\boldsymbol{b}$，$\boldsymbol{c}$ の要素間で

$$c_i = a_i + b_i \quad (i = 1, \cdots, n) \qquad (1)$$

のような加算を考えます。このベクトル処理では最初のデータがパイプライン演算器に達した後，パイプラインの全ステージを通過するまでの時間は演算結果は得られませんが，それ以後は１ステージ当たりの通過時間（これをパイプラインのピッチとよびます）ごとに演算結果が１つ（m 多重のパイプラインならば最大m個）得られます。したがって，ベクトル長 n の演算に要する時間 t は，最初のデータがパイプライン演算器に達するまでの時間を t_i，パイプラインのステージ数を l，パイプラインのピッチを t_p とすると，１多重の場合には，

$$t = t_i + l t_p + (n-1) t_p \qquad (2)$$

となります（１）式は n 回の浮動小数点加算を行いますから，そのflops値は，

$$\text{flops値} = \frac{n}{t}$$

$$= \frac{1}{t_p}\left(1 - \frac{1}{n} + \frac{l}{n} + \frac{1}{n}\frac{t_i}{t_p}\right)^{-1} \qquad (3)$$

となります。$n \to \infty$としたときのflops値はピーク性能または理論最大性能などと呼ばれ，（3）式より$1/t_p$すなわちパイプラインの1ピッチの逆数に等しくなります。また，通常マシンサイクルはt_pと等しくなります。

ここでVPP5000のピーク性能を計算してみましょう。VPP5000のマシンサイクルは3.33ナノ秒（ナノ秒は10^{-9}秒）です。ただし，VPP5000のパイプライン演算器は16多重で，乗算と加算を同時に行うことができますから，この場合には演算数は通常の16×2倍になります。したがって，ピーク性能は，

$$\frac{1}{3.33\times10^{-9}} \times 16 \times 2 = 9.6\times10^{9}\ (\text{flops})$$

すなわち，9.6Gflopsとなります。ただし，これはあくまでも$n \to \infty$としたときの理論上の最大性能です。

つぎに，ベクトル長 n に対する演算時間 t の変化をVPP5000で実測してみましょう。例えば（1）式であらわされる加算はFortranでは配列を用いて，

```
do i=1,n
  c(i)=a(i)+b(i)
enddo
```
（4）

のように記述されます。変数 n の値を1から1200まで変えていき，それぞれのループ長（ベクトル長）に対して（4）のdoループを実行するのに要するCPU時間をプロットしたのが図2です。ただし，配列 a，b，c は倍精度実数型としたので演算は64ビット浮動小数点加算です。個々の測定データは×印で示されていますが，重なっているため太くて短い水平な線が階段状に並んでいるようにみえます。このように $n-t$ のグラフは実際のベクトル計算機では（2）式であらわされるような単純な直線にはなりません。その理由としては，複数のパイプラインが並列動作可能なこと，パイプライン演算器が多重化されていること，ベクトルレジスタの容量が有限であるためベクトル長が大きい場合にはデータを区切って処理されることなどが考えられます。図2中の直線は測定データを最小二乗近似して得られたものです。データにばらつきはあるものの，全体としては直線的に変化しているとみてよいでしょう。

Hockneyら［3］は，ベクトル長 n のベクトルデータに対する単一の演算に要する時間 t を

$$t = \frac{1}{r_\infty}(n + n_{1/2}) \qquad (5)$$

で近似することを提案しています。ここで，r_∞と $n_{1/2}$は定数で，r_∞は$n \to \infty$としたときの処理速度をあらわし，$n_{1/2}$は r_∞の1/2の速度を達成するのに必要なベクトル長をあらわします。ただし，処理速度 r は，

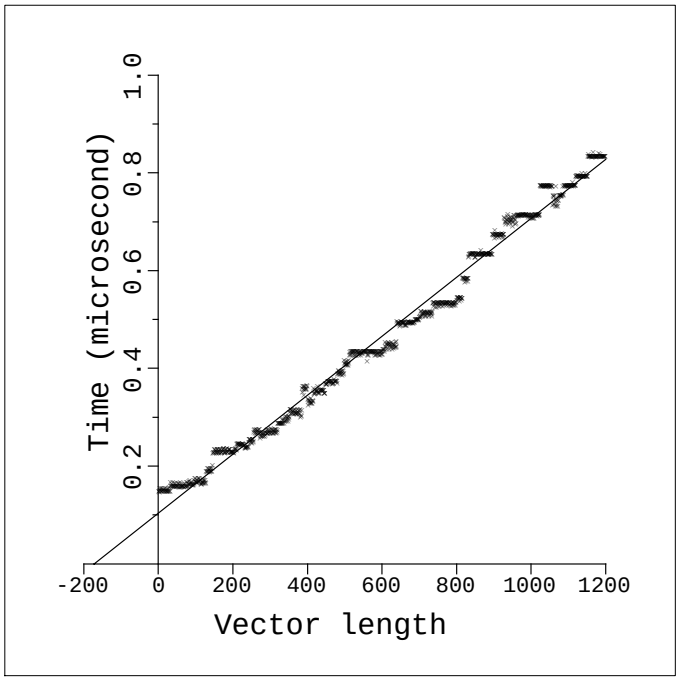


図２　加算に要するCPU時間

$$r=\frac{n}{t} \qquad (6)$$

で，通常flops値であらわします。いくつかの n に対して t を実測し，その測定データを最小二乗近似して得られる直線の傾きの逆数が r_∞を与え，直線と n 軸との交点が$-n_{1/2}$を与えます。

（４）のdoループであらわされる加算に関するVPP5000の r_∞と $n_{1/2}$を求めると，図２の直線の傾きと n 軸との交点から，r_∞=1.66（Gflops），$n_{1/2}$=172となります。（３）式と（６）式を比較するとわかるように，r_∞はピーク性能をあらわしています。先程VPP5000のピーク性能は9.6Gflopsと計算されました。ただし，（１）式のように加算しか行わない場合にはその１/２の4.8Gflopsが理論上のピーク性能となります。よって，実測によるピーク性能はその約１/３になります。また，$n_{1/2}$は r_∞の１/２の速度を達成するのに要するベクトル長をあらわしますから，（６）式で n の値が172（$n_{1/2}$の値と同じくらい）のときピーク性能（1.66Gflops）の50%，n が700程度（$n_{1/2}$の４倍）のときピーク性能の80%の速度が得られることになります。一般に，ベクトル長（ループ長）をより長くした方がベクトル計算機の性能を引き出すことができます。

いくつかの簡単な演算について実測したVPP5000の r_∞と $n_{1/2}$の値を表１に示します。表１中の計算式は（４）の形のdoループ中にあらわれる代入文を指しています。また，ループ長は１から1200まで変化させています。

表１　VPP5000の r_∞ と $n_{1/2}$

項番	計算式	r_∞ (Gflops)	$n_{1/2}$
1	`c(i)=a(i)+b(i)`	1.66	172
2	`c(i)=a(i)*b(i)`	1.65	165
3	`c(i)=a(i)*b(i)+d(i)`	2.79	222
4	`c(i)=e*a(i)+b(i)`	3.27	159
5	`c(i)=1.0d0/a(i)`	1.20	180

Ⅴ．ベクトル化率

コンパイラは自動ベクトル化機能を備えているため，プログラムの中でベクトル処理可能な部分は自動的にベクトル命令に置き換えられます（ベクトル化されます）。このため，我々はベクトル演算を意識しないでプログラムを作成し，ベクトル計算機を利用することができます。しかし，ベクトル計算機はベクトルデータを対象にした演算を高速に処理する計算機ですから，そのハードウェア性能をいかすにはベクトル化される部分の割合が大きいプログラムを作成することが重要です。

ここで，ベクトル化率 v を

$$v = \frac{\text{ベクトル化可能な部分をスカラ処理するのに要する時間}}{\text{全体をスカラ処理するのに要する時間}}$$

で定義します。ベクトル化可能な部分をベクトル処理することにより，スカラ処理に対して速度がα倍になるとすると，ベクトル計算機ではプログラムの中で（１－v）の部分はスカラ処理され，v の部分は処理時間が１/αになります。よって，このプログラム全体をスカラ処理した場合に対する処理速度の比 E は，

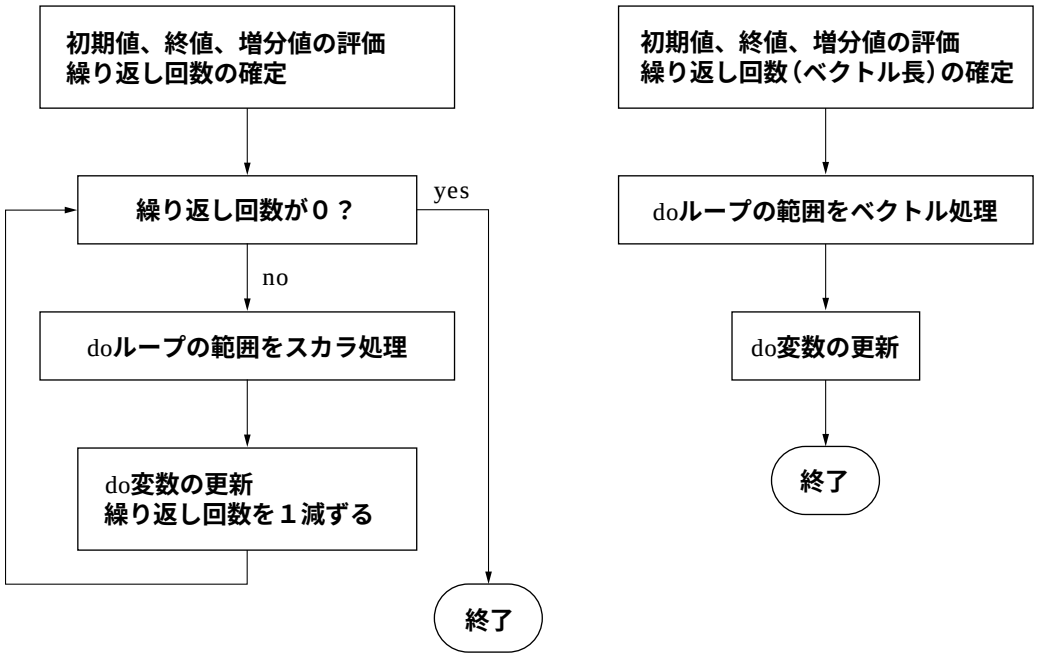
$$E = \frac{1}{1 - v(1 - 1/\alpha)} \qquad (7)$$

となります。ここで，α≫１とすると $E \approx 1/(1-v)$ ですから，v =0.5，0.75，0.9のとき，それぞれ $E \approx 2$，４，10となります。したがって，例えベクトル処理能力が優れていてα≫１であっても，ベクトル化率が１に近くなければベクトル計算機の性能をいかすことができないということになります。

．doループとベクトル処理

Fortranでは1つの配列全体あるいはその一部がベクトルデータとしてベクトル演算の対象になります。doループの場合には，ループ内にあらわれる配列要素に対するスカラ演算をベクトル演算に置き換えることでベクトル化を実現しています。

ベクトル化するときとしないときのdoループの処理の違いを図３に示します。スカラ処理では



（ａ）スカラ処理　　（ｂ）ベクトル処理

図３　doループ処理

図３の（ａ）のように必要な回数だけdoループの範囲が繰り返し実行されます。ベクトル化した場合には図３の（ｂ）のように繰り返し回数をベクトル長としてdoループの範囲にあるそれぞれの文がベクトル処理されます。スカラ処理と比べて決定的に異なるのはベクトル処理では図３の（ａ）のようなループが形成されないことです。

つぎのdoループを例にして実行の順序をみてみましょう。

```
do i=1,n
  c(i)=a(i)+b(i)
  z(i)=x(i)+y(i)
enddo
```
（8）

スカラ処理ではこのdoループの実行順序は,

```
c(1)=a(1)+b(1)
z(1)=x(1)+y(1)
c(2)=a(2)+b(2)
z(2)=x(2)+y(2)
      ⋮
c(n)=a(n)+b(n)
z(n)=x(n)+y(n)
```

となります。一方，ベクトル処理では,

```
c(1)=a(1)+b(1)
c(2)=a(2)+b(2)
      ⋮
c(n)=a(n)+b(n)
z(1)=x(1)+y(1)
z(2)=x(2)+y(2)
      ⋮
z(n)=x(n)+y(n)
```

のように実行されます。すなわち，doループ内の文があたかもそれぞれ単独のdoループを形成したかのように（最終的な演算結果が変わらない範囲で）順序が変更されて実行されます。このようにして（8）のdoループは長さが n のベクトル $\boldsymbol{a}$，$\boldsymbol{b}$，$\boldsymbol{c}$ 及び $\boldsymbol{x}$，$\boldsymbol{y}$，$\boldsymbol{z}$ に関する演算

$$\boldsymbol{c}=\boldsymbol{a}+\boldsymbol{b} \tag{9}$$

$$\boldsymbol{z}=\boldsymbol{x}+\boldsymbol{y} \tag{10}$$

に置き換えられます。

もう1つの例をつぎに示します。

```
do i=1,n
  c(i)=a(i)+b(i)
  a(i+1)=c(i)+d(i)
enddo
```
(11)

このループをスカラ処理すると，

```
c(1)=a(1)+b(1)
a(2)=c(1)+d(1)
c(2)=a(2)+b(2)
a(3)=c(2)+d(2)
      ⋮
```

となります。一方，ベクトル処理すると，

```
c(1)=a(1)+b(1)
c(2)=a(2)+b(2)
      ⋮
a(2)=c(1)+d(1)
a(3)=c(2)+d(2)
      ⋮
```

となりますが，実行順序を変更したことによって正しい結果が得られなくなります。なぜならば，(11) のdoループは $k \geq 2$ なる k に対して，do変数iの値が $k-1$ のときの`a(i+1)`の値が得られるまではiの値が k のときの`a(i)`の値が決まらないために`c(i)`の計算ができない構造になっている

からです。このような`a(i)`と`a(i+1)`の関係を回帰参照であるといいます。回帰参照が存在する演算を回帰演算と呼びますが，一般に回帰演算ではデータの参照関係が不適切になるのでベクトル化できません。Fortranコンパイラはdoループ中にあらわれるデータの参照関係の適・不適を認識してベクトル化するかしないかを自動的に判断するので，（11）のdoループはベクトル化されません。

．ベクトル化の条件

Fortranプログラムがベクトル化されるための条件は以下のようにまとめられます。

１．ベクトル化対象範囲内にあること
　doループ
　配列式
２．ベクトル化可能な文であること
　算術代入文（例：`c(i)=a(i)+b(i)`）
　論理代入文（例：`lc(i)=la(i).and.lb(i)`）
　算術 if 文（例：`if(m(i)) 10,20,30`）
　論理 if 文（例：`if(a(i).gt.0.0) c(i)=a(i)+b(i)`）
　ブロック if 文，elseif 文，else 文，endif 文
　単純 go to 文（例：`go to 10`）
　continue 文
　単純 where 文（例：`where(a > 0.0) a=1.0/a`）
　where 構文
　case 構文
　exit 文
　cycle 文
３．ベクトル化してもデータの定義引用の順序に変更がないこと

したがって，入出力文やcall文などはベクトル化されません。これらベクトル化可能な文にあらわれるデータは，４バイトまたは８バイトの整数型，単精度または倍精度実数型，単精度または倍精度複素数型，４バイトの論理型のいずれかでなければなりません。

以下にベクトル化される例を示します。

［例１］

```
v  do i=1,n
v    c1(i)=a1(i)*a2(i)+b(i)
v    c2(i)=a1(i*istrid)/b(i*istrid)
v    c3(i)=a1(n+1-i)*e-a2(n+1-i)
v    c4(i)=a1(list(i))*a2(list(i))
v  enddo
```

1つの配列は記憶域に連続して割り付けられます。これを参照するパターンとして，

連続アクセス（上のdoループ中の1番目の文にあらわれる配列a1，a2，bなど）

一定間隔アクセス（2番目の文のa1，b）

逆方向アクセス（3番目の文のa1，a2）

間接アクセス（4番目の文のa1，a2）

などがありますが，いずれの場合もベクトル化されます。配列listはリストベクトルと呼ばれています。このdoループの各文の先頭にあるvはコンパイラが出力するベクトル化の表示記号で，その文がベクトル化されたことをあらわします。一般に，データを参照する速度（単位時間当たりに主記憶からベクトルレジスタにロードまたはベクトルレジスタから主記憶にストアするデータ量）は連続的な参照が最も速く，間接的な参照が最も遅くなります。一定間隔の参照では間隔のとり方（変数 istrid の値）によって参照する速度が大きく変動します（後述）。

［例2］

```
v  do i=1,n
v    d=d+a1(i)
v    e=e+a1(i)*a2(i)
v    f=min(f,b(i))
v    c1(i)=sin(a1(i))
v  enddo
```

1番目の文は総和，2番目は内積，3番目は最小値（最大値も可）をそれぞれ求める演算ですが，これらはベクトルマクロ命令と呼ばれる特別なベクトル命令により実行されます。この場合もスカラ処理とは異なる演算順序で実行され，最終的な結果がそれぞれの左辺の変数に代入されます。4番目の例は組込み関数がベクトル化された例です。組込み関数の実引数がベクトルデータの場合にベクトル化されます。

［例3］

```
v  do i=1,n
v    if(b(i).gt.d) then
v      c1(i)=sqrt(b(i)-d)
v    elseif(b(i).lt.e) then
v      c1(i)=b(i)**2
v    else
v      c1(i)=0.0
v    endif
v  enddo
```

これはブロック if 文がベクトル化された例です。

［例４］

```
m  do i=1,n
v    c1(i)=a1(i)*a2(i)/b(i)
v    c2(i)=log(a1(i))
v    c3(i)=a1(i)*d+a2(i)*e
s    write(1,600) c1(i),c2(i),c3(i)
v  enddo
```

入出力文のようなベクトル化されない文（記号sで表示）がdoループ中にあらわれると，本来ベクトル化可能な文は部分的にベクトル化させます（記号mで表示）。だたし一般に，スカラ処理する場合に比べて大幅な処理時間の短縮は望めません。この例のように１つのdoループ中にベクトル化可能な文とそうでない文を混在させることはできる限り避けた方がよいでしょう。

［例５］

```
s   do j=1,m
s2    do k=1,n
v2      do i=1,l
v2        c(i,j)=c(i,j)+a(i,k)*b(k,j)
v2      enddo
s2    enddo
s   enddo
```

これは多重doループのベクトル化の例（行列積の計算）です。コンパイラはベクトル長やデータの参照のパターンなどを認識し，場合によってはdoループを入れ換えてより効率的なdo変数でベクトル化することもあります。この例ではコンパイラはdo変数iでベクトル化したことを示しています。記号vやsに続く数字の２はその文がループアンローリングによって展開され，その展開数が２であることを示しています。ループアンローリングの手法は並列動作可能なパイプラインをできるだけ使用するようにベクトル演算の演算密度を高めるために用いられます。このdoループはつぎのように展開されたものと思われます。（変数l，nの値が２で割り切れるかどうかで実際の処理は異なりますが，核になる部分は基本的には下のようになると思われます）

```
  do j=1,m
    do k=1,n,2
      do i=1,l,2
        c(i,j)=c(i,j)+a(i,k)*b(k,j)+a(i,k+1)*b(k+1,j)
        c(i+1,j)=c(i+1,j)+a(i+1,k)*b(k,j)+a(i+1,k+1)*b(k+1,j)
      enddo
    enddo
  enddo
```

変数l，m，nの値がそれぞれ2000のとき，このdoループの処理時間を実測すると9.3Gflopsという

ピーク性能に近い値が得られました。

［例６］

```
v  do i=1,n
v    e=a1(i)+a2(i)
v    c1(i)=e*b(i)
v  enddo
```

このdoループは変数（スカラデータ）への代入文を含んでいて本来ベクトル化に適さないのですが，変数の代わりに一時的な配列をコンパイラが用意してベクトル化を促進しています。

．ベクトル演算の高速化

ここではベクトル演算をより高速に実行するためのポイントをいくつか述べることにします。

１．ベクトル長

すでに述べたように，ベクトル長 n のベクトルデータに対する単一の演算に要する時間 t は，

$$t=\frac{1}{r_\infty}\left(n+n_{1/2}\right)$$

で近似できます。ただし，r_∞と $n_{1/2}$は定数で，r_∞は$n\to\infty$としたときの処理速度をあらわし，$n_{1/2}$は r_∞の１/２の速度を達成するのに必要なベクトル長をあらわします。処理速度 r は，

$$r=\frac{n}{t}$$

$$=\frac{r_\infty}{1+\frac{n_{1/2}}{n}}$$

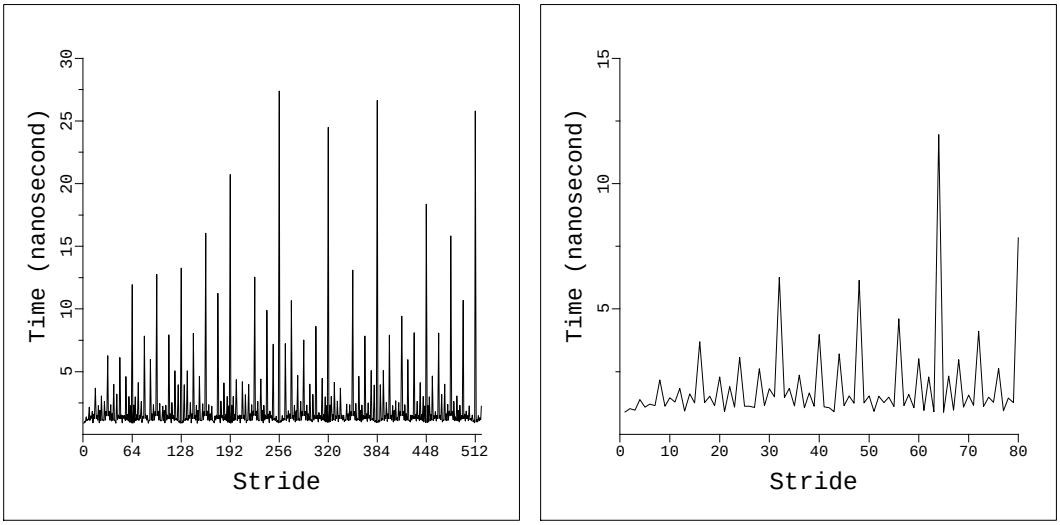
となりますから，一般にnが大きいほど処理速度は増します。表１に示したように，VPP5000上での実測例では$n_{1/2}$は200前後ですから，nが800くらい（$n_{1/2}$の約４倍）のとき，ピーク性能 r_∞の80%の速度が得られます。

２．doループ内の演算数

doループ内の演算数を増加させることは，パイプラインの並列動作の可能性を高めるので高速化の有効な方法です。よって，ループアンローリングは効果的な手法です。ループアンローリングによらなくても，ループ長の等しい複数個のdoループを１つのdoループにまとめたり，doループ内にできるだけたくさんの演算を詰め込んだりすることによってもパイプラインの並列動作性を高めることができます。また，doループが1つになれば図３の（b）に示した初期処理は１回で済みます。

３．一定間隔のデータ参照

配列を一定間隔に参照する場合，その間隔の大きさによって処理時間は著しく変動します。図



（a）　　（b）

図４　データ参照の間隔とCPU時間

４の（ａ）は以下のようなdoループをベクトル処理するとき，変数istridの値を１から520まで１ずつ増加させて，istridの値，すなわち，配列を参照する間隔に対するdoループ１回当たりの処理時間を示したものです。ただし，doループ中の配列a，cは倍精度実数型で，変数nの値は1000としました。

```
v  do i=1,n
v    c(i)=a(i*istrid)
v  enddo
```

図４の（ａ）では細かいところはわかりにくいですが，少なくとも間隔が64，128，192，…といった2^mの倍数のところで非常に処理時間が大きくなる（遅くなる）のはおわかり頂けるでしょう。間隔が80までの部分を表示したのが図４の（ｂ）です。間隔が１，すなわち連続的な参照のとき最も速く，１を除く奇数のときがこれに次ぎます。また，間隔が偶数のところで遅くなり，特に2^mの倍数でこれが顕著になります。

主記憶装置はバンクと呼ばれる2^l個の独立して動作するモジュールで構成されていて，倍精度実数型配列の１番目の要素はバンク０に，２番目の要素はバンク１に，…，2^l番目の要素はバンク2^l-1に，2^l+1番目の要素は元に戻ってバンク０に，…という形で配置されます。したがって，配列を連続的に参照すると2^l個のバンクが並列に動作することができます。しかし，間隔が２で参照するときには並列動作可能なのは2^{l-1}個になりますから，主記憶・ベクトルレジスタ間のデータ転送能力は連続参照に比べて低下し，間隔が2^lのときもっとも低下します。同じバンク上にある複数個のデータを参照するとき，１つ前のデータの参照が終了するまでそのつぎのデータの参照が待たされるという現象はメモリ競合と呼ばれますが，この発生が図４にみられる特徴的な変化の主な原因です。

メモリ競合を回避するには間隔が2^mの倍数となるようなデータ参照をしないことが本質的に重要です。２次元配列a(i,j)の場合には，連続的な参照になるようにできるだけiを変化させる方向に参照するのが望ましいです。jを変化させる方向の参照は一定間隔の参照になりますから，メモリの発生を防ぐために宣言文中の配列aの第１添字の寸法を奇数にとるようにします。単精度実数型や４バイト整数型の配列では，１番目と２番目の要素はバンク０に，３番目と４番目の要素はバンク１に，…のように配置されるので，２次元配列の第1添字の値は奇数の２倍（$4k+2$，$k \geq 0$）にします。

４．最適化制御行の挿入

最適化制御行とは，コンパイラが認識できない情報やベクトル化を促進する情報を含む行で，利用者が作成するFortranプログラム中に挿入します。最適化制御行の詳細については文献［４，５］をご覧ください。

［使用例１］

```
   !ocl k.lt.0
v       do i=2,n
v         a1(i+k)=a1(i)*a2(i)+b(i)
v       enddo
```

第１～第４カラムが!oclではじまる行が最適化制御行です。この例ではdoループ中の配列a1の添字式にあらわれるkが０より小さい（回帰参照ではない）ことを指示してベクトル化しています。

［使用例２］

```
   !ocl novrec(a1)
v     do i=1,n-1
v       a1(i+k2)=a1(i+k1)*a2(i)+b(i)
v     enddo
```

このdoループ中の配列a1の添字式i+k1とi+k2の大小関係がコンパイラには認識できないので，何も指定しないとa1は回帰参照であるとみなされベクトル化されません。回帰参照ではないと分かっている場合には上のように指定することによってベクトル化できます。

．おわりに

CPU時間を大量に費やすプログラムでは，実際にはそのプログラムのある限られた部分だけで集中的にCPU時間を消費するといわれます。したがって，このような部分を見つけて最適なベクトル化を図ることが重要です。VPP5000の並列化に関しては文献［６，７，８，９］等をご覧ください。文献［４，５，６，７，８，９］の最新版はセンターのホームページから参照できます。

参考文献

［1］小高俊彦・河辺峻（1980）超高速演算の動向. 情報処理, Vol. 21, No. 9, 927－937

［2］島崎眞昭（1985）スーパーコンピュータ入門. コンピュータソフトウェア, Vol. 2, No. 3, 484--508

［3］R. W. Hockney and C. R. Jesshope (1988) Parallel Computers 2 ― Architecture, Programming and Algorithms. Adam Hilger, 625pp

［4］富士通（1999）UXP/V Fortran使用手引書 V20用. J2U5-0410-02

［5］富士通（1997）UXP/V VPプログラミングハンドブック V10用. J2U5-0070-01

［6］富士通（1999）UXP/V Fortran/VPP使用手引書 V20用. J2U5-0430-02

［7］富士通（1997）UXP/V VPPプログラミングハンドブック V10用. J2U5-0090-01

［8］永井 亨（2000）VPP Fortran入門（改訂版）名古屋大学大型計算機センターニュース, Vol.31, No.３, 230－270

［9］富士通（1998）UXP/V MPI使用手引書 V11用. J2U5-0271-02

（ながい　とおる：名古屋大学情報連携基盤センター大規模計算支援環境研究部門）