SXシステムにおける有効なプログラミング例

日本電気㈱ 情報処理システム技術本部
萬　　淳一，花村光泰，宮平知博

1. はじめに

一般に、ベクトルマシンの性能を最大限に発揮させようとすれば、ハードウェアの特徴とコンパイラの機能を理解し、それに適合した数値計算法および高速化技法を用いてプログラムを作る必要がある。ここでは、SXシステムを有効に利用するために、利用者自身がどのようなことを考えてプログラムを作れば良いかを実例を挙げて説明する(注)。

ベクトルマシンに関して、そのハードウェア性能を引き出すために、多くの高速化技法が報告されている。ここでは、高速化技法を、アルゴリズムレベルとブロック・ステートメントレベルに分けて説明をする。アルゴリズムレベルの高速化は、数値計算アルゴリズムの並列性を見い出すことによってベクトルマシン向きのアルゴリズムを選択することである。一方、ここで言うブロック・ステートメントレベルの高速化とは、アルゴリズムは、かえないが、ベクトル化を阻害する定義/参照関係を解消してベクトル化可能としたり、あるいは、ベクトル化可能なDOループでもそのDOループ内の演算密度を上げるべく、プログラムを部分的に書き換えることである。一般に、前者の高速化の方が、後者の高速化より、効果が大きい。以下に、上記の2つのレベルの高速化について、実例を挙げて述べることにする。

2. アルゴリズムレベルの高速化

2.1 ベクトルマシン向きのアルゴリズム選択

ベクトルマシン向きのアルゴリズム選択において重要である計算速度の評価について考えてみる。従来の数値解析法における計算速度の評価は、コンピュータのハードウェアやシステムの詳細に立ち入らぬという暗黙の了解があり、その計算法に現れる浮動小数点演算回数だけにより行なっていた。しかし、ベクトルマシンにおける実践的な計算速度の評価では、算術演算回数以外に、ベクトル化の可否、並列実行の度合、更には、システムに依存する非算術演算回数による評価が必要となる。ここで述べた非算術演算とは扱うデータの参照/格納、DOループの設定、判定・分岐といったものであり、プログラムの書き方に依存するところも多いので、マシンの特性を良く理解した上で評価する必要がある。

注）本稿は大阪大学および東北大学からの執筆依頼に基づき、計算時間等の実際例を中心にまとめたものである。

2.2 高速フーリエ変換を例にして

2.2.1 1次元高速フーリエ変換の例

高速フーリエ変換（FFT）は、N^2回の演算を必要とする離散型フーリエ変換を、$N\log_2 N$回の演算で実行してしまうアルゴリズムのことで、1965年にCooleyとTukeyにより開発された。それ以来、多くの研究がなされ、種々のアルゴリズムが提案されている。離散型フーリエ変換およびFFTの原理については、参考文献[4]を参照されたい。

(1) FFTの3つのアルゴリズム

ここでは、FFTの代表的な3つのアルゴリズム（Cooley-Tukey, Peaseおよび Stockham）について、SXシステムへの適応性を評価する。図1のプログラムリストは、各々のアルゴリズム

```
      DO 130 M=1,ISTEP
         LE=2**M
         LE1=LE/2
         LK=N/LE
         DO 110 J=1,LE1
            K=(J-1)*LK+1
            DO 100 I=J,N,LE
               IP=I+LE1
      A2R(I)=AR(I)+(WR(K)*AR(IP)-WI(K)*AI(IP))
      A2I(I)=AI(I)+(WR(K)*AI(IP)+WI(K)*AR(IP))
      A2R(IP)=AR(I)-(WR(K)*AR(IP)-WI(K)*AI(IP))
      A2I(IP)=AI(I)-(WR(K)*AI(IP)+WI(K)*AR(IP))
  100       CONTINUE
  110    CONTINUE
         DO 120 I=1,N                                  リスト1
            AR(I)=A2R(I)
            AI(I)=A2I(I)
  120    CONTINUE
  130 CONTINUE
C-----------------------------------------------------------
      N2=N/2
      DO 300 J=1,L
         N2J=N/(2**J)
         NR=N2J
         NI=(2**J)/2
         DO 250 I=1,NI
            IJ=(I-1)*N2J
            IJ2=IJ*2
            IJN=IJ2+N2J
            IJN2=IJ+N2
            DO 200 IR=1,NR
               A2R(IR+IJ)=AR(IR+IJ2)
     &                   +WR(IJ+1)*AR(IR+IJN)-WI(IJ+1)*AI(IR+IJN)
               A2I(IR+IJ)=AI(IR+IJ2)
     &                   +WR(IJ+1)*AI(IR+IJN)+WI(IJ+1)*AR(IR+IJN)
               A2R(IR+IJN2)=AR(IR+IJ2)
     &                     -WR(IJ+1)*AR(IR+IJN)+WI(IJ+1)*AI(IR+IJN)
               A2I(IR+IJN2)=AI(IR+IJ2)
     &                     -WR(IJ+1)*AI(IR+IJN)-WI(IJ+1)*AR(IR+IJN)
  200       CONTINUE
  250    CONTINUE
         DO 270 IR=1,N                                 リスト2
            AR(IR)=A2R(IR)
            AI(IR)=A2I(IR)
  270    CONTINUE
  300 CONTINUE
C-----------------------------------------------------------
      DO 101 IS=IS2,ISTEP
         LA=N/2**IS
         IL=0
         DO 120 L=1,LA
            DO 110 K=1,NN,LA
               IL=K+L-1
               W2R(IL)=WR(K)
               W2I(IL)=WI(K)
  110       CONTINUE
  120    CONTINUE
         DO 180 J=1,NN
            J2=J*2
            J2M1=J2-1
            A2R(J)=AR(J2M1)+(W2R(J)*AR(J2)-W2I(J)*AI(J2))
            A2I(J)=AI(J2M1)+(W2R(J)*AI(J2)+W2I(J)*AR(J2))
            A2R(J+NN)=AR(J2M1)-(W2R(J)*AR(J2)-W2I(J)*AI(J2))
*           A2I(J+NN)=AI(J2M1)-(W2R(J)*AI(J2)+W2I(J)*AR(J2))
  180    CONTINUE
         DO 172 I=1,N                                  リスト3
            AR(I)=A2R(I)
  172       AI(I)=A2I(I)
  101 CONTINUE
```

図1 1次元高速フーリエ変換の3つのアルゴリズムのプログラム例
（上から、Cooley-Tukey, Stockham, Pease）

の中心部分のプログラムである。FFTのアルゴリズム選択で注目すべきことは、DOループ長、ベクトルデータの間隔、およびビット反転処理の有無である。表1に、3つのプログラムの特徴について示す。

表1　FFTプログラムの比較

アルゴリズム	ループ長	ベクトルデータの間隔		ビット反転
		入力側	出力側	
Cooley-Tukey	増　加	2のべき乗で増加	2のべき乗で増加	有　り
Pease	一　定	一定(2のべき乗)	連　続	有　り
Stockham	減　少	連　続	連　続	無　し

一般にベクトルマシンで有利な条件は、DOループ長が一定で長いこと、ベクトルデータの間隔が連続または奇数間隔であること、およびビット反転がないことである。ビット反転処理は、ベクトルマシンになじまず、スカラマシン向きであると言われていたが、SXシステムでは、ビット反転処理のマシン命令を持っているので、不利な条件とは言えない。そこで、SXシステムに適合したアルゴリズムは、PeaseとStockhamのアルゴリズムであると予想される。どちらが優位であるかは、両方のプログラムに高速化を施して実際に測定する必要がある。

(2)　高速化技法と測定結果

3つのアルゴリズムに以下の4つの高速化技法を施し、SX-1（256バンク；以下同じ）を使用して実行時間を測定した。

①　ベクトルビット反転命令の使用（Cooley-TukeyおよびPeaseのアルゴリズムに適用）

SXシステムに特有なベクトルビット反転命令を用いることにより、ビット反転処理の高速化を図る。図2に、プログラムリストを示す。ベクトル組込み関数IBRVが、反転命令であり、これにより生成されるビット逆順テーブル IBR をリストベクトルとして、ビット反転処理を行っている。実際にFFTプログラムの中で使用する際は、リストベクトルIBRによるバンクコンフリクト(注)を低減させるためにループ変数Iに間隔を設ける。

(注)　SX-1の主記憶は512あるいは256個のバンクに分かれており、異なるバンクに対するアクセスは最高速で行なうことができる。ところが、メモリのサイクルタイムと転送速度の関係から、1サイクルタイム内に同一バンクへ複数回のアクセスが発生するとデータ供給能力が著しく低下する。このようにデータアクセスが一部のバンクに集中する状態をバンクコンフリクト（bank conflict）という。

```
      DO  10  I=1, N
          IT(I)=I-1
          IBR(I)=IBRV(IT(I), 32-ISTEP, ISTEP)
          A2R(I)=AR(IBR(I)+1)
          A2I(I)=AI(IBR(I)+1)
   10 CONTINUE
```

図2　ベクトルビット反転命令の使用例

② 三角関数テーブルの改良（Peaseのアルゴリズムに適用）

Peaseのアルゴリズムの場合、回転因子行列の参照が複雑なので、$N\log_2 N$の大きさの配列に回転因子行列を初期化時に格納しておく。

③ 中間データの転送の減少（Cooley-Tukey, PeaseおよびStockhamのアルゴリズムに適用）

FFTの演算では、各ステージの出力が次のステージの入力となるため、中間データの転送が必要となる（図1参照）。これを避けるために、2つのステージをDOループ中で連続して実行するようにする。

④ 折り返し処理（Stockhamのアルゴリズムに適用）

Stockhamのアルゴリズムの場合、DOループの入れ換えにより、最深DOループのループ長が減少するプログラムと増加するプログラムの2通りのプログラムを書くことができる。この2つのプログラムを結合することにより、できるだけループ長を長くすることが可能となる。プログラム例を図3に示す。

```
      NN =N/2
      IS1=ISTEP/2
      IS2=IS1+1
      DO 101  IS=1,IS1
       LA=2**(IS-1)
       JUMP=LA*2
       DO 110  L=1,LA
        IA=L
        IB=L+NN
        JA=L
        JB=L+LA
        I=0
        J=0
        DO 120 K=1,NN,LA
         A2R(JA+J) =AR(IA+I) +AR(IB+I)
         A2I(JA+J) =AI(IA+I) +AI(IB+I)
         A2R(JB+J) =WR(K)*(AR(IA+I)-AR(IB+I))
     &             -WI(K)*(AI(IA+I)-AI(IB+I))
         A2I(JB+J) =WI(K)*(AR(IA+I)-AR(IB+I))
     &             +WR(K)*(AI(IA+I)-AI(IB+I))
         I=I+LA
         J=J+JUMP
  120   CONTINUE
  110  CONTINUE
       DO 172 I=1,N
        AR(I)=A2R(I)
  172   AI(I)=A2I(I)
  101 CONTINUE
      IA=1
      IB=1+NN
      JA=1
      DO 1  IS=IS2,ISTEP
       LA=2**(IS-1)
       JB=1+LA
       I=0
       J=0
       DO 20 K=1,NN,LA
        DO 10  L=1,LA
         A2R(JA+J) =AR(IA+I) +AR(IB+I)
         A2I(JA+J) =AI(IA+I) +AI(IB+I)
         A2R(JB+J) =WR(K)*(AR(IA+I)-AR(IB+I))
     &             -WI(K)*(AI(IA+I)-AI(IB+I))
         A2I(JB+J) =WI(K)*(AR(IA+I)-AR(IB+I))
     &             +WR(K)*(AI(IA+I)-AI(IB+I))
         I=I+1
         J=J+1
   10    CONTINUE
         J=J+LA
   20  CONTINUE
       DO 72 I=1,N
        AR(I)=A2R(I)
   72   AI(I)=A2I(I)
    1 CONTINUE
```

図3　折り返し処理（Stockham）

図4に、これらの高速化を施した後の測定結果を示す。測定は、SX−1で行い、精度は単精度である。横軸はデータ点数、縦軸は、Cooley−Tukeyの実行時間をPeaseおよびStockhamの実行時間で割った比である。即ちそれぞれのアルゴリズムがCooley−Tukeyのアルゴリズムの何倍速いかを示す。Cooley−Tukeyのアルゴリズムは、予想通り、すべてのデータ点数で最も遅く、これらのデータ点数の範囲ではPeaseのアルゴリズムが、SXシステムに適している。

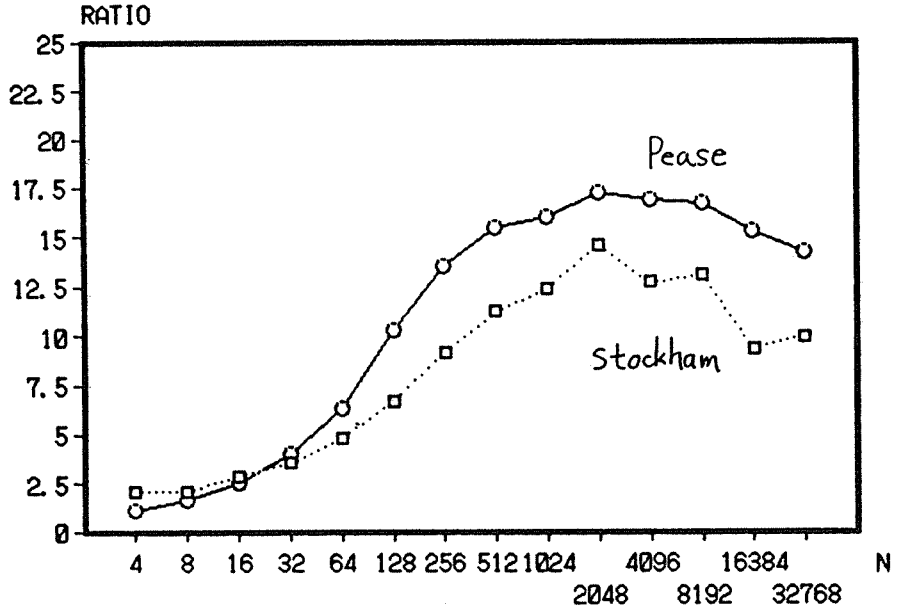

図4 Cooley−Tukeyに対するPease及びStockhamのアルゴリズムの実行時間比

表2に、各々のアルゴリズムで、高速化技法を施こす前後の高速化率（前と後の実行時間を各々T_0, T_1と表わすとき、$(T_0-T_1)/T_0$で定義する）を示す。PeaseおよびStockhamのアルゴリズムで90%近い高速化の効果が得られた。また、表3に、それぞれの高速化技法がその改善に寄与した割合を示す。この表より、ベクトルビット反転命令の使用および折り返し処理の効果が大きいことがわかる。

表2 高速化の効果（1024点）

アルゴリズム	高速化率（%）	
	単精度	倍精度
Cooley−Tukey	23	21
Pease	88	87
Stockham	91	90

表3 高速化手法の効果（1024点、単精度）

アルゴリズム	高速化手法	割合(%)
Cooley−Tukey	ベクトルビット反転命令使用	86
	中間データ転送の減少	14
Pease	ベクトルビット反転命令使用	87
	中間データ転送の減少	4
	三角関数テーブルの改良	9
Stockham	折り返し処理	97
	中間データ転送の減少	3

2.2.2　2次元高速フーリエ変換の例

2次元 FFT では、同一のデータ点数の1次元変換が複数回行われるので、1次元 FFT の場合とは異った観点からベクトルマシン向きの高速化アルゴリズムを考える必要がある。今M×Nの2次元データに対し2次元 FFT を実行するとして、次の3つのアルゴリズムを考える。

(1)　アルゴリズムA

2次元 FFT 演算における、1次元 FFT の多重性に注目し、入力データの格納されている2次元配列に対し1次元 FFT を各々の次元について実行することにより結果を得る方法である。即ち次の2つの処理ステップで行う。

①　各列にM点の1次元 FFT を施す（M点の FFT をN回実行）。

②　上記の変換結果の各行にN点の1次元 FFT を施す（N点の FFT をM回実行）。

図5（a）は②の処理に相当する部分のプログラムの中心部である。このアルゴリズムでは、1次元 FFT 演算を行うループが内側に設定されるので、使用した1次元 FFT アルゴリズムのベクトルマシンに対する特性がそのまま性能に反映される。

(2)　アルゴリズムB

これは同時に行う複数の1次元 FFT の処理に共通する部分に注目し、それらについてベクトル処理を行う方法である[5]。プログラム的には図5（b）に示すように FFT を行う方向（ここではデータの格納されている配列の2次元目の添字に対応する方向）と直交する方向にベクトル化のループを走らせる形となる。アルゴリズムBにおける最深 DO ループのループ長は各々の次元のデータ点数となり、さらに一定となる（アルゴリズムAでは最大でもアルゴリズムBの1/2である）。

(3)　アルゴリズムC

アルゴリズムBの最深 DO ループのループ長を、さらに長くするために、Aのアルゴリズムとの融合を行う。即ち、アルゴリズムBの DO 10 と DO 5 の2重 DO ループを1つのループにまとめる。これによりループ長は、各々の次元のデータ点数に FFT 演算の（最深）ループ長を乗じたものとなる。プログラムは図5（c）である。

```
      NN = N/2
      DO 5 IX = 1,M
         IA = 1
         IB = 1+NN
         JA = 1
         DO 1 IS = 1,ISTEP
            LA = 2**(IS-1)
            JB = 1+LA
            I = 0
            J = 0
            DO 20 K = 1,NN,LA
               DO 10 L = 1,LA
                  A2R(IX,JA+J) = AR(IX,IA+I) + AR(IX,IB+I)
                  A2I(IX,JA+J) = AI(IX,IA+I) + AI(IX,IB+I)
                  A2R(IX,JB+J) = WR(K)*(AR(IX,IA+I) - AR(IX,IB+I))
     &                           -WI(K)*(AI(IX,IA+I) - AI(IX,IB+I))
                  A2I(IX,JB+J) = WI(K)*(AR(IX,IA+I) - AR(IX,IB+I))
     &                           +WR(K)*(AI(IX,IA+I) - AI(IX,IB+I))
                  I = I+1
                  J = J+1
   10          CONTINUE
               J = J+LA
   20       CONTINUE
            .
            .
            .
    1    CONTINUE
    5 CONTINUE
```

(a) アルゴリズムA

```
      NN = N/2
      IA = 1
      IB = 1+NN
      JA = 1
      DO 1 IS = 1,ISTEP
         LA = 2**(IS-1)
         JB = 1+LA
         I = 0
         J = 0
         DO 20 K = 1,NN,LA
            DO 10 L = 1,LA
               DO 5 IX = 1,M
                  A2R(IX,JA+J) = AR(IX,IA+I) + AR(IX,IB+I)
                  A2I(IX,JA+J) = AI(IX,IA+I) + AI(IX,IB+I)
                  A2R(IX,JB+J) = WR(K)*(AR(IX,IA+I) - AR(IX,IB+I))
     &                           -WI(K)*(AI(IX,IA+I) - AI(IX,IB+I))
                  A2I(IX,JB+J) = WI(K)*(AR(IX,IA+I) - AR(IX,IB+I))
     &                           +WR(K)*(AI(IX,IA+I) - AI(IX,IB+I))
    5          CONTINUE
               I = I+1
               J = J+1
   10       CONTINUE
            J = J+LA
   20    CONTINUE
         .
         .
         .
    1 CONTINUE
```

(b) アルゴリズムB

```
      NN = N/2
      DO 1 IS = 1,ISTEP
         LA = 2**(IS-1)
         JB = 1+LA
         LAM = LA*M
         J = 0
         DO 20 K = 1,NN,LA
            JAI = J*LAM
            JBI = JAI+M*NN
            JAJ = 2*JAI
            JBJ = LAM+JAJ
            J   = J+1
            DO 5 IR = 1,LAM
               A2R(IR+JAJ) = AR(IR+JAI) + AR(IR+JBI)
               A2I(IR+JAJ) = AI(IR+JAI) + AI(IR+JBI)
               A2R(IR+JBJ) = WR(K)*(AR(IR+JAI) - AR(IR+JBI))
     &                       -WI(K)*(AI(IR+JAI) - AI(IR+JBI))
               A2I(IR+JBJ) = WI(K)*(AR(IR+JAI) - AR(IR+JBI))
     &                       +WR(K)*(AI(IR+JAI) - AI(IR+JBI))
    5       CONTINUE
   20    CONTINUE
         .
         .
         .
    1 CONTINUE
```

(c) アルゴリズムC

図5　2次元高速フーリエ変換の3つのアルゴリズムのプログラム例

表4に以上3つのアルゴリズムのSX-1での実測結果を示す。

なお、本測定においてアルゴリズムAで使用した1次元FFTはPeaseのアルゴリズムである。この表より、アルゴリズムBあるいはCがSXシステムに適しており、さらにデータ点数が小さい時は、アルゴリズムCが最適であることがわかる。また今回は示さなかったが、各次元のデータ点数M、Nが大きく異る場合はアルゴリズムAと、BあるいはCを組み合わせることにより、効率よく処理ができる。

表4　2次元FFTの3つのアルゴリズムの実測結果（単精度、単位msec）

データ点数（N＝M）	アルゴリズムA	アルゴリズムB	アルゴリズムC
16	0.851	0.320	0.183
32	1.92	0.743	0.491
64	4.79	1.91	1.49
128	12.5	5.87	6.19
256	41.4	25.4	25.8
512	162	110	120
1024	705	479	498

3. ブロック・ステートメントレベルの高速化

3.1　高速化技法

ここでは、プログラムをSXシステムに適合する形に高速化する場合のポイント、特に、既にベクトル化されているループをさらに効率よく実行させるための技法について述べる。

スーパーコンピュータにおけるプログラムの高速化の最重要点はベクトル化できる部分を増やすことであるが、ベクトル化されているDOループでも、わずかな書き換えによってさらに高速になる場合がある。そのためのSXシステムにおけるポイントは、主に次の3つである。

1）メモリアクセス時のバンクコンフリクトの防止

2）内積型演算の外積型演算化

3）外積型演算のアンローリング

以下に、実際例に則して説明する。

3.2　LU分解を例として

図6は、連立方程式の解法に多用される実行列のLU分解を行なう簡単なサブルーチン（ピボッティング等は行なっていない）である。まず、メモリアクセス時のバンクコンフリクトを防ぐため

に、メインプログラム中のLU分解される行列の配列宣言で第1添字寸法を奇数（一般にはn次元配列の初めのn－1個の添字寸法を奇数）にとる。これだけでも、かなりの効果が出る（表5）。

```
      SUBROUTINE LU1 (A,LNA,N)
      DOUBLE PRECISION A(LNA,N),SUM
      DO 10 I=1,N
         DO 20 J=I,N
            SUM=0.0D0
①           DO 30 K=1,I-1
               SUM=SUM+A(J,K)*A(K,I)
30          CONTINUE
            A(J,I)=A(J,I)-SUM
20       CONTINUE
            DO 40 J=I+1,N
               SUM=0.0D0
②           DO 50 K=1,I-1
               SUM=SUM+A(I,K)*A(K,J)
50          CONTINUE
            A(I,J)=(A(I,J)-SUM)/A(I,I)
40       CONTINUE
10    CONTINUE
      RETURN
      END
```

図6　サブルーチン LU1

このサブルーチン中の2つの最内側DOループは共にベクトル化される内積型演算であるが、この内、総和の計算部分はベクトル計算の効果が弱いので、図6の①のDOループを次のような外積型(注)の計算に書き換えた方がSXシステムでは有利である。この際、変数の定義・参照関係をコンパイラが判断できなくなるので、ベクトル化指示行を次のように入れる。

```
            DO 20 K=1,I-1
*VDIR NODEP
            DO 20 J=1,N
               A(J,I)=A(J,I)-A(J,K)*A(K,I)
   20       CONTINUE
```

さらに、このDOループはアンローリングをすることができる。ここで、アンローリング（unrolling）とは、文字どおりDOループをほどいて展開する（unroll）ことであり、一般には、DOループを飛びにして、その分の処理をDOループ中に書き下す技法のことである。一般にアンローリングは複数の演算器を並列に動作させるために行なうが、SXシステムではハードウェア的に複数の演算器が並列動作するので、そのようなアンローリングは行なう必要がない。SXシステムにおいてアンローリングの効果があらわれるのは、1つの式の演算密度が上がる場合だけであり、従って、外積型演算の外側DOループをアンローリングする場合に限られる。上記のループは次のように10段のアンローリングをすることで、さらに高速に実行される。

(注)　この例は厳密には中間積型の演算であるが、広い意味で外積型という名称を用いることとする。

```
          I1=(I-1)/10*10
          DO 20 K=1,I1,10
*VDIR NODEP
             DO 20 J=I,N
                A(J,I)=A(J,I)-(A(J,K  )*A(K  ,I)+A(J,K+1)*A(K+1,I)
     &                        +A(J,K+2)*A(K+2,I)+A(J,K+3)*A(K+3,I)
     &                        +A(J,K+4)*A(K+4,I)+A(J,K+5)*A(K+5,I)
     &                        +A(J,K+6)*A(K+6,I)+A(J,K+7)*A(K+7,I)
     &                        +A(J,K+8)*A(K+8,I)+A(J,K+9)*A(K+9,I))
  20     CONTINUE
          DO 30 K=I1+1,I-1
*VDIR NODEP
             DO 30 J=I,N
                A(J,I)=A(J,I)-A(J,K)*A(K,I)
  30     CONTINUE
```

同様に②のDOループも外積型演算化及びアンローリングを行なうことができ、最終的に図7のLU2 のように書き換えることができる。これらの書き換えによる効果を表5にまとめて示しておく。

```
C
      SUBROUTINE LU2 (A,LNA,N)
      DOUBLE PRECISION A(LNA,N)
       DO 10 I=1,N
          I1=(I-1)/10*10
          DO 20 K=1,I1,10
*VDIR NODEP
             DO 20 J=I,N
                A(J,I)=A(J,I)-(A(J,K  )*A(K  ,I)+A(J,K+1)*A(K+1,I)
     &                        +A(J,K+2)*A(K+2,I)+A(J,K+3)*A(K+3,I)
     &                        +A(J,K+4)*A(K+4,I)+A(J,K+5)*A(K+5,I)
     &                        +A(J,K+6)*A(K+6,I)+A(J,K+7)*A(K+7,I)
     &                        +A(J,K+8)*A(K+8,I)+A(J,K+9)*A(K+9,I))
  20     CONTINUE
          DO 30 K=I1+1,I-1
*VDIR NODEP
             DO 30 J=I,N
                A(J,I)=A(J,I)-A(J,K)*A(K,I)
  30     CONTINUE
          DO 40 K=1,I1,10
*VDIR NODEP
             DO 40 J=I+1,N
                A(I,J)=A(I,J)-(A(I,K  )*A(K  ,J)+A(I,K+1)*A(K+1,J)
     &                        +A(I,K+2)*A(K+2,J)+A(I,K+3)*A(K+3,J)
     &                        +A(I,K+4)*A(K+4,J)+A(I,K+5)*A(K+5,J)
     &                        +A(I,K+6)*A(K+6,J)+A(I,K+7)*A(K+7,J)
     &                        +A(I,K+8)*A(K+8,J)+A(I,K+9)*A(K+9,J))
  40     CONTINUE
          DO 50 K=I1+1,I-1
*VDIR NODEP
             DO 50 J=I+1,N
                A(I,J)=A(I,J)-A(I,K)*A(K,J)
  50     CONTINUE
          DO 60 J=I+1,N
             A(I,J)=A(I,J)/A(I,I)
  60     CONTINUE
  10  CONTINUE
      RETURN
      END
```

図7　サブルーチン LU2

表5　高速化の効果（上段 sec 単位 / 下段 MFLOPS 単位）　（SX−1で測定）

次元	LU1 (配列 2000×2000)	LU1 (配列 2001×2000)	LU1 (DOループ①を外積演算化)	LU1 (さらにDOループ①をアンローリング)	LU2
500	1.354 61	0.7634 109	0.6086 136	0.5208 160	0.2597 320
1000	8.677 76	4.023 165	3.620 184	2.974 224	1.848 360
1500	26.62 84	10.97 205	10.65 211	8.528 263	5.922 379
2000	59.83 89	22.81 233	23.31 228	18.36 290	13.65 390

4. おわりに

コンピュータが与えられていれば、計算時間を短くする最強の決め手はアルゴリズムの選択である。これは従来から言われていることであり、今後も変らないであろう。しかし、ベクトルマシンにおけるアルゴリズム選択の評価基準としては、少ない浮動小数点演算回数で結果が求められるというだけでは不十分であり、DOループ長、IF文の現われ方、データ転送などの非演算的な要素を加味させねばならない。つまり、従来の汎用コンピュータではいずれの計算手法を取ろうとも同程度の計算時間であったものが、ベクトルマシンでは数倍の差が出る場合がありうるということである。そこで、SXを有効に利用するには、この観点から再度プログラムを見直してみる価値があると思われる。

一方、高速化のためのプログラミング技法をいくつか示した。これらは、使用するマシンとプログラムに依存し、効果の少ない場合もあろう。また、このような技法は近い将来コンパイラの機能に含まれるものと期待している。そこで、現時点では計算時間を最も消費する部分についてのみ適用を試みることを勧めたい。

なお、連立方程式の解や固有値を求めるなどの一般的計算の場合は、ここに述べた高速化のための煩わしい作業を避ける手段がある。それはSX用に最適化された科学技術計算ライブラリASL/SXを利用することである。

最後に、本稿に示した測定データは、1986年6月までに測定したものである。現在も高速化作業を継続しており、測定データがかわることもあり得るので注意されたい。

5. 参考文献

[1] Hockney, R. W., Jesshope, C. R., (奥川 史, 黒住祥祐共訳):"並列計算機", 共立出版 (1984).

[2] 村田健郎, 小国 力, 唐木幸比古:"スーパーコンピュータ", 丸善 (1985).

[3] Dongarra, J. J., Gustavson, F. G. and Karp, A.; "Implementing Linear Algebra Algorithms for Dense Matrices on a Vector Pipeline Machine", SIAM Rev., 26, 91-112, (1984).

[4] Brigham, E. O., (宮川 洋, 今井秀樹共訳):"高速フーリエ変換", 科学技術出版社 (1978).

[5] Peterson, W. P.; "Vector Fortran for Numerical Problems on CRAY-1", Commun. ACM. 26, 1008 (1983).