

S X システムのアーキテクチャの特徴と

科学技術計算ライブラリ A S L / S X

日本電気株式会社 基本ソフトウェア開発本部 片山 博 (Hiroshi Katayama)

同 システム技術本部 花村 光泰 (Mitsuyasu Hanamura)

1. はじめに

S X システムは、大規模科学技術計算の高速実行を目的に開発したスーパーコンピュータであり、ベクトル計算を超高速度で実行するだけでなく、スカラ計算や入出力処理、システム制御などにおいても高速化を図り、総合的な処理能力を向上させている。また、使い易さも特徴の一つであり、汎用超大型機と同等の豊富な機能をもつオペレーティングシステム、高度な自動ベクトル化や最適化能力を持つ FORTRAN77/SX コンパイラ、ANALYZER/SX や VECTORIZER/SX などの使い易い性能向上支援ツールなどによりこれを実現している。

本稿では、始めに主としてプログラミングの立場から見た S X システムのアーキテクチャの特徴について述べ、続いて S X システムの科学技術計算ライブラリである A S L / S X で採用している手法を例に、S X システムにおける代表的な

高速化技法について述べる。

2. SXシステムのアーキテクチャの特徴

SXシステムには、SX-2、SX-1およびSX-1Eの3つのモデルがあり、それぞれ1300MFLOPS、570MFLOPS、285MFLOPSの最大性能をもつ。以下では、最大のモデルであるSX-2を中心に、そのアーキテクチャの特徴を述べる。

2.1 SXシステムの構成

SXシステムは、演算プロセッサと制御プロセッサの2つの独立したプロセッサから構成されている。演算プロセッサは、超高速のベクトルユニットと高速のスカラーユニットを備え、利用者のプログラムの高速実行を受け持つ。一方、制御プロセッサは、ジョブ制御、入出力処理、資源管理やTSS処理など、システム制御機能を受け持つ。コンパイルやリンク処理などもこの上で実行される。SXシステムは、このような機能分散構成をとることにより、全体の処理能力を高めている。図1にSXシステムの構成を示す。

2.2 演算プロセッサ

次に、演算プロセッサのアーキテクチャの特徴を述べる。

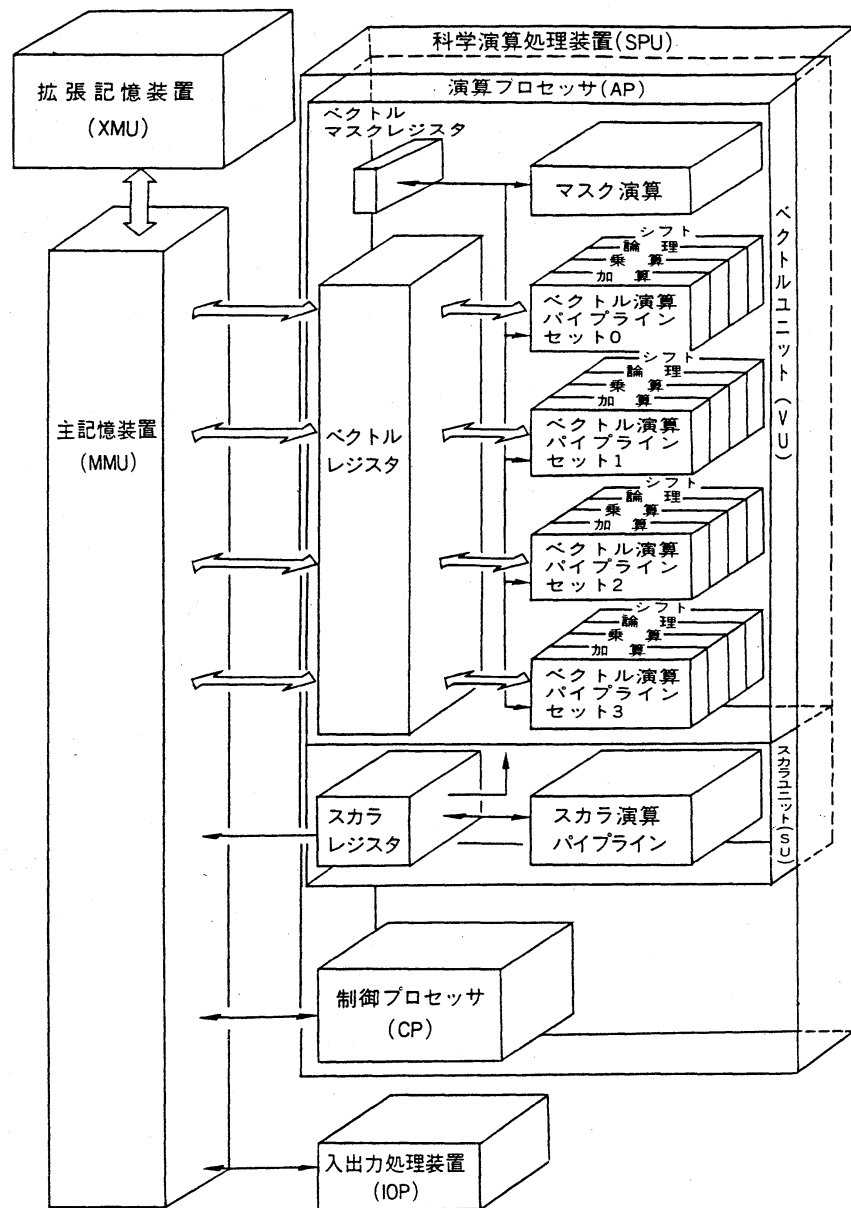


図1 SXシステムの構成

(1) RISCアーキテクチャの採用

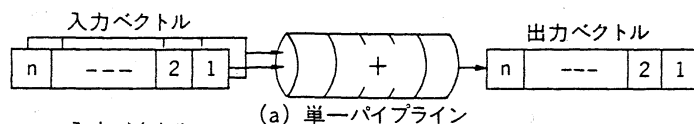
RISC (Reduced Instruction Set Computer) は、命令セットの単純化によりハードウェアの基本性能を上げ、それによりシステム全体の性能を高くするという思想である。演

算プロセッサは，この R I S C の考え方に基づいて設計されている．6ナノ秒という高速マシンサイクルやスカラ性能の高速化の実現は，R I S C の採用によるところが大きい．

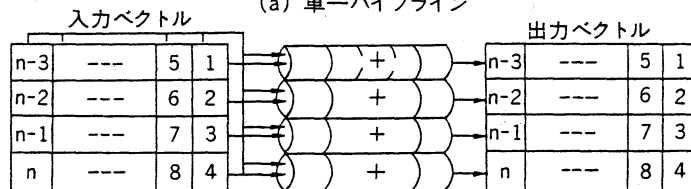
(2) 多重並列パイプラインと自動チェイニング

演算プロセッサは，ベクトル計算の高速化のために，多重並列パイプラインと呼ぶ方式を採用している（図2参照）．

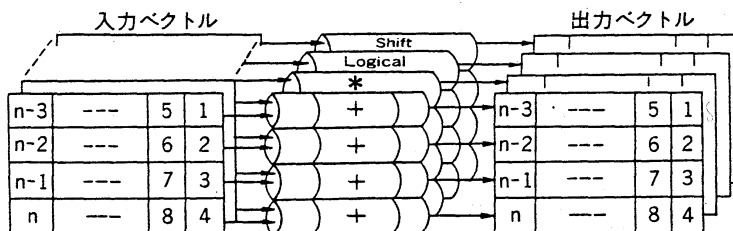
図2(a)は，通常の単一パイプライン方式を示したもので，1マシンサイクルに1個の結果しか得る事ができない．そこで，S X システムでは，(b)に示すよう



(a) 単一パイプライン



(b) 並列パイプライン



(c) 多重並列パイプライン

に同一構成のパイプ

図2 種々のパイプライン方式

ラインを4セット（S X - 2の場合）並列に配置し，単一演算に対して，1マシンサイクルに4個の結果が得られるようにしている．しかも，各パイプラインへのデータの配分は，ハードウェアにより自動的に行われる．さらに，この4セットのパイプラインを加算，乗算，論理，シフトのそれぞれに対して多重に設け，全体で16本のパイプライン構成としてい

る。また、演算結果を後続の演算で使用するような場合に、パイプラインを自動的に連結して一方の出力を待ち時間なしに他方に入力し、これらを並列に動作させる、自動チェイニング機能を備えている。

(3) 高速の主記憶装置と大容量ベクトルレジスタ

ベクトルデータは、演算の前に主記憶からベクトルレジスタに転送され、演算される。結果もベクトルレジスタ中に出力され、その後、必要に応じて再び主記憶に戻される。したがって、主記憶とベクトルレジスタとの間の転送は、ベクトル演算パイプラインの性能に見合うだけ高速でなければならない。SXシステムの主記憶装置は、最大256Mバイトの容量をもつ。また、最大512ウェイのインタレースを採用することにより、1マシンサイクルに8ワード（1ワードは64ビット）、すなわち、11Gバイト/秒の高速転送を実現している。また、ベクトルレジスタは、最大80Kバイトの容量をもつ。このレジスタは、演算の中間結果の保持にも使われる。

(4) ベクトル命令とスカラ命令の並列実行

ベクトル命令とスカラ命令は、並列に実行可能である。ベクトル命令と並列に実行されたスカラ命令の実行時間は、実効的にゼロとなる。

(5) 幅広いベクトル化適応性

ベクトル演算に対するマスク制御機能を始め、圧縮/伸長、収集/拡散、総和、漸化式などの豊富なベクトル命令セットを用意することにより、幅広いベクトル化に適応している。

(6) スカラ演算の高速化

スーパーコンピュータでは、ベクトル性能に劣らずスカラ性能も極めて重要である。実際、ベクトル化率が80%~90%であっても、実行時間を支配しているのは、ベクトル性能ではなくスカラ性能である。そこで、SXシステムでは、次に示すような工夫により、スカラ演算の高速化を図っている。

① 演算パイプラインの採用

ベクトル演算ばかりでなく、スカラ演算部もパイプライン化することにより、複数のスカラ演算の並列実行を可能としている。なお、FORTRAN77/SXコンパイラは命令の並べ替えを行い、パイプラインの空き時間を最小にすることにより、その効果を最大限に発揮させている。表1に、リバモアループのスカラでの実行結果をもとにした、命令並べ替えの効果を示す。

② 128個のスカラレジスタ

SXシステムでは、高速化のために、128個という多数のスカラレジスタを用意している。これにより、主記憶の参照回数が大幅に減少するばかりでなく、命令の並べ替え

表 1 命令並べ替えの効果

| ループ 番号 | 並べ替えなし (MFLOPS) | 並べ替えあり (MFLOPS) | 効果 (%) |
|-----------|--------------------|--------------------|-----------|
| 1 | 17.3 | 18.1 | 4.6% |
| 2 | 17.9 | 31.4 | 75.4% |
| 3 | 13.3 | 15.9 | 19.5% |
| 4 | 11.5 | 11.5 | 0.0% |
| 5 | 12.0 | 15.9 | 32.5% |
| 6 | 12.0 | 16.7 | 39.2% |
| 7 | 20.0 | 32.2 | 61.0% |
| 8 | 24.7 | 46.2 | 87.0% |
| 9 | 20.5 | 32.0 | 56.1% |
| 10 | 13.7 | 18.5 | 35.0% |
| 11 | 8.3 | 8.3 | 0.0% |
| 12 | 8.3 | 8.3 | 0.0% |
| 13 | 6.3 | 9.3 | 47.6% |
| 14 | 11.6 | 14.3 | 23.3% |
| 平均 | 14.1 | 19.9 | 41.1% |

の可能性が増大し、パイプラインの効果を一層向上させることができる。

(7) 拡張記憶装置

拡張記憶装置は、入出力の高速化のために用意された半導体の記憶装置で、磁気ディスク装置の数百倍の転送速度をもつ。この装置を使用するためには、プログラムの変更は不要であり、単にジョブ制御言語 (JCL) あるいは実行時のオプションによる指定だけでよい。

(8) その他

SXシステムでは、高速フーリエ変換 (FFT) の高速化

のために，ビット反転処理を高速で実行するベクトルビット反転命令を用意している．この命令は，後述の A S L / S X において使用されている．図 3 に命令動作を示す．

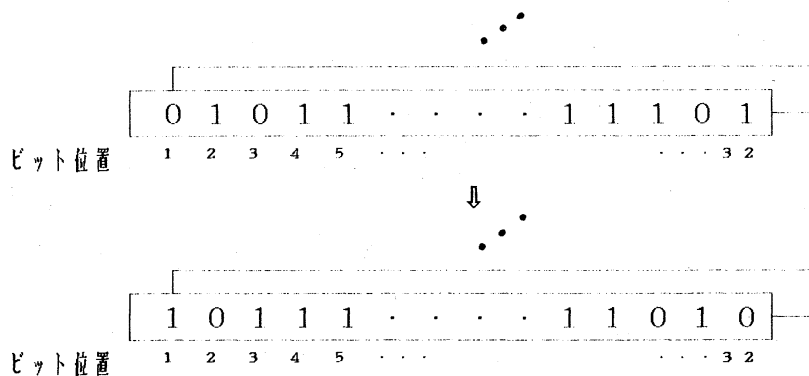


図 3 ベクトルビット反転命令の動作

3. A S L / S X における高速化技法

A S L / S X (Advanced Scientific Library SX) は，S X システム用に最適化された科学技術計算ライブラリで，① 基本行列演算 ② 連立一次方程式 ③ 固有値・固有ベクトル ④ 高速フーリエ変換 (F F T) ⑤ スプライン関数 ⑥ 数値積分などを適用領域としている．A S L / S X で採用している高速化技法は，アルゴリズムレベルの高速化と，ブロックステートメントレベルの高速化に大別される．以下に，前者の例として次元 F F T を，後者の例として L U 分解を取り上げ，S X システムにおける代表的な高速化技法を示す．

3.1 一次元 F F T の高速化

F F T は、データ点数が N の離散型フーリエ変換の演算数を N^2 から $N \log_2 N$ に減少させる画期的なアルゴリズムで、1965年に Cooley と Tukey により開発されて以来、多くの研究が成されている。ここでは、一次元 F F T の代表的なアルゴリズムである Cooley-Tukey, Pease および Stockham の 3 つのアルゴリズムについて、S X システムへの適合性を評価する。

(1) 各アルゴリズムの特徴

S X システムにおける高速化の重要なポイントとして、次の 2 つが挙げられる。

① ベクトルの長さを可能な限り長くすること。これは、ベクトル化されたループの実行時における立ち上がり時間の影響を可能な限り小さくするためである。

② 参照するベクトルを、可能な限り連続あるいは奇数間隔のベクトルにすること(ただし、偶数でも 2 ならよい。)

これは、主記憶参照時の、少数のバンクへの参照の集中化(バンクコンフリクト)を避けるためである。

また、F F T プログラムには、ビット反転処理を必要とするものとしなないものがあり、前者はベクトル化に適さず、スーパーコンピュータ向きではないとされてきた。表 2 は、これらの観点から 3 つのアルゴリズムを比較したものである。

表 2 3つのアルゴリズムの比較

| アルゴリズム | ベクトル長 | ベクトルの間隔 | | ビット反転 |
|--------------|---------|---------|-------|-------|
| | | 入力側 | 出力側 | |
| Cooley-Tukey | 変化(減少) | 2のべき乗 | 2のべき乗 | 有り |
| Pease | 一定(N/2) | 一定(2) | 連続 | 有り |
| Stockham | 変化(減少) | 連続 | 連続 | なし |

また、図4に、それぞれのアルゴリズムの中心部、すなわち
 バタフライ演算部分のプログラムを示す。

```

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
      AZR(I)=AR(I)+(WR(K)*AR(IP)-WI(K)*AI(IP))
      AZI(I)=AI(I)+(WR(K)*AI(IP)+WI(K)*AR(IP))
      AZR(IP)=AR(IP)-(WR(K)*AR(IP)-WI(K)*AI(IP))
      AZI(IP)=AI(IP)-(WR(K)*AI(IP)+WI(K)*AR(IP))
    100 CONTINUE
    DO 120 I=1, N
      AR(I)=AZR(I)
      AI(I)=AZI(I)
    120 CONTINUE
  130 CONTINUE
  N2=N/2
  DO 300 J=1, L
    N2J=N/(2**J)
    NR=N2J
    N1=(2**J)/2
    DO 250 I=1, N1
      IJ=(I-1)*N2J
      IJN=IJ+NR
      IJN2=IJ+2*NR
      DO 200 IR=1, NR
        AZR(IR+IJ)=AR(IR+IJ)
          +WR(IJ+1)*AR(IR+IJN)-WI(IJ+1)*AI(IR+IJN)
        &
        AZI(IR+IJ)=AI(IR+IJ)
          +WR(IJ+1)*AI(IR+IJN)+WI(IJ+1)*AR(IR+IJN)
        &
        AZR(IR+IJN2)=AR(IR+IJ2)
          -WR(IJ+1)*AR(IR+IJN)+WI(IJ+1)*AI(IR+IJN)
        &
        AZI(IR+IJN2)=AI(IR+IJ2)
          -WR(IJ+1)*AI(IR+IJN)-WI(IJ+1)*AR(IR+IJN)
        &
      200 CONTINUE
      DO 270 IR=1, N
        AR(IR)=AZR(IR)
        AI(IR)=AZI(IR)
      270 CONTINUE
    300 CONTINUE
  DO 101 IS=1S2, 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
      DO 180 J=1, NN
        J2=J+NR
        J2M1=J2-1
        AZR(J)=AR(J2M1)+(W2R(J)*AR(J2)-W2I(J)*AI(J2))
        AZI(J)=AI(J2M1)+(W2R(J)*AI(J2)+W2I(J)*AR(J2))
        AZR(J+NN)=AR(J2M1)-(W2R(J)*AR(J2)+W2I(J)*AI(J2))
        AZI(J+NN)=AI(J2M1)-(W2R(J)*AI(J2)-W2I(J)*AR(J2))
      180 CONTINUE
      DO 172 I=1, N
        AR(I)=AZR(I)
        AI(I)=AZI(I)
      172 CONTINUE
    101 CONTINUE
  
```

図 4 一次元 FFT の 3 つのアルゴリズム
 (上から, Cooley-Tukey, Stockham, Pease)

(2) 適用した高速化技法

それぞれに対し，次の4つの高速化技法を施した。

① ベクトルビット反転命令の使用

Cookey-TukeyおよびPeaseの両アルゴリズムに対して，前述のベクトルビット反転命令を適用した。コーディング例を次に示す。

```

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

```

なお，実際のプログラムでは，バンクコンフリクトの影響を小さくするための工夫を行っている。

② 三角関数テーブルの改良

Peaseのアルゴリズムの場合，回転因子行列参照が複雑なので，あらかじめ $N \log_2 N$ の大きさの配列に回転因子行列を格納するようにした。

③ 中間データの転送回数の減少

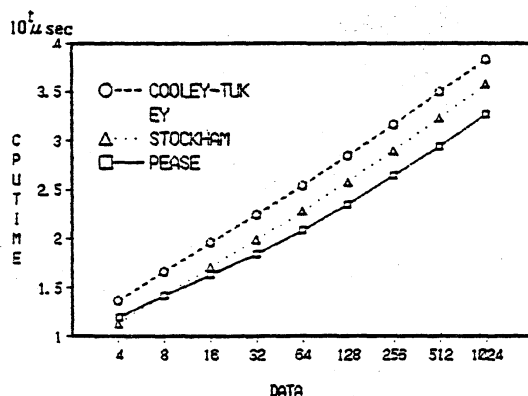
FFTの演算では，各ステージの出力が次のステージへの入力となるため，出力を入力側に転送する必要がある。これを避けるため，D O ループ中に2つのステージを並べるように改良した。この高速化は，3つのアルゴリズム全てに対して適用した。

④ 折り返し処理

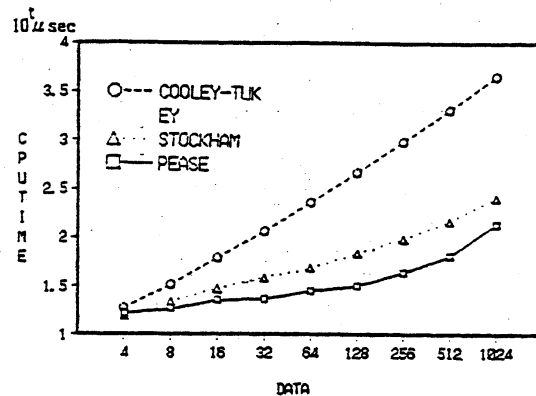
Stockhamのアルゴリズムの場合，ベクトル長が増加するようにも減少するようにもプログラミングできるので，この2つを結合して実行中に切り替えることにより，ベクトル長をできるだけ長く保つようにした。

(3) 実測結果

図5に，SX-2による実測結果を，高速化前と高速化後について示す。なお，それぞれの高速化技法が性能改善にどの程度寄与しているかを，データ点数1024，単精度の場合について見てみると，Cooley-TukeyおよびPeaseのアルゴリズムでは，ベクトルビット反転命令の使用が，それぞれ86%，87%と期待どおり大きな効果を上げており，またStockhamのアルゴリズムでは，改善値は小さいものの，折り返し処理が97%と極めて大きな比率を占めているのが目につく。



高速化前



高速化後

図5 実測結果

3.2 LU分解の高速化

次に，ASL/SXで採用しているブロックステートメントレベルの高速化技法について，連立一次方程式の解法において多用されるLU分解の簡単なサブルーチン（ピボットティングなし）を例に説明する．プログラムを図6に示す．

```

SUBROUTINE LU1 (A,LNA,N)
DOUBLE PRECISION A(LNA,N),SUM
DO 10 I=1,N
  DO 20 J=1,N
    SUM=0.000
    DO 30 K=1,I-1
      SUM=SUM+A(J,K)*A(K,I)
    CONTINUE
    A(J,I)=A(J,I)-SUM
  CONTINUE
  DO 40 J=I+1,N
    SUM=0.000
    DO 50 K=1,I-1
      SUM=SUM+A(I,K)*A(K,J)
    CONTINUE
    A(I,J)=(A(I,J)-SUM)/A(I,I)
  CONTINUE
CONTINUE
10 CONTINUE
RETURN
END

```

① { 30
20

② { 50
40

図6 LU分解のプログラム

(1) 適用した高速化技法

適用した高速化技法を，以下に示す．

① ベクトルデータの参照パターンの改善

呼び出し側における配列宣言の一次元目の寸法を2000から2001にすることにより，主記憶参照時のバンクコンフリクトを避ける．

② 内積演算の積和演算化

SXシステムでは，内積も積和も高速であるが，ループ

長が1000以下では積和のほうが高速である。それは、内積は、積和に比べて主記憶参照が少なく効率がよいが、ループ長が短いと、ベクトル値をスカラ値に集約するための操作に要する時間の影響が無視できなくなるからである。

```

DO 20 J=I, N
  SUM=0.0D0
  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 20 K=1, I-1
  DO 20 J=I, N
    A(J, I)=A(J, I)-A(J, K)*A(K, I)
20  CONTINUE

```

③ アンローリング

ループ本体を n 倍にし、代わりにループの回転数を $1/n$ にする高速化技法を、アンローリングと呼ぶ。SXシステムでは、多重並列パイプライン構成を採用しているため、一般にアンローリングは不要である。逆に、最深ループについてのアンローリングは、かえってベクトル化の効率を低下させるため、避けたほうがよい。しかし、多重ループの外側ループについてのアンローリングで、かつベクトルの参照（特にストア）回数が減少する場合には、大きな効果が得られることがある。②の変形後の積和演算は、この例である。次に、2段のアンローリングのプログラム例を

示す。ただし、端数処理は省いてある。

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

↓

```
I1=(I-1)/2*2
DO 20 K=1, I1, 2
  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)
20 CONTINUE
```

なお、今回は、10段のアンローリングを施した。

(2) 実測結果

表3に高速化の効果の実測結果を示す。表が示すように、この例では、ブロックステートメントレベルの書き替えだけで、4～5倍の高速化が達成されている。

表3 高速化の効果

(上段：msec, 下段：MFLOPS. 倍精度. SX-2で測定)

| 次元 | 配列宣言 2000*2000 | 配列宣言 2001*2000 | DOループ①を 積和演算化 | さらにDOループ① をアンローリング | ②についても 適用 |
|------|-------------------|-------------------|------------------|-----------------------|---------------|
| 500 | 0.8096 103 | 0.5533 150 | 0.4025 207 | 0.3556 234 | 0.1421 585 |
| 1000 | 4.792 139 | 2.773 240 | 2.189 304 | 1.881 354 | 0.9218 722 |
| 1500 | 13.96 161 | 7.178 313 | 6.050 371 | 5.075 443 | 2.830 794 |
| 2000 | 30.32 175 | 14.28 373 | 12.68 420 | 10.45 510 | 6.367 837 |

4. おわりに

以上、主としてプログラミングの立場から見たSXシステムのアーキテクチャの特徴と、ASL/SXで採用されている高速化技法を例に、SXにおける有効なプログラミング技法について述べた。

最後に、本稿をまとめるにあたり、数々の貴重な助言をいただいた、当社システム技術本部応用システム部の、津和義昭氏、萬淳一氏ならびに宮平知博氏に、この紙面を借りて感謝いたします。

参考文献

- 1) 古勝, 渡辺, 近藤; 「最大性能1.3GFLOPS, マシンサイクル6nsのスーパーコンピュータ SXシステム」, 日経エレクトロニクス, 1984.11.19, pp.237-272
- 2) 片山, 河原, 大中; 「FORTRAN77/SXにおける高速化技法」, 大阪大学大型計算機センター・ニュース, Vol.16, No.1 (1986.5)
- 3) 花村, 宮平; 「SXを有効に利用するための実際例」, 東北大学大型計算機センター・ニュース (SENAC), Vol.19, No.2 (1986.4)
- 4) H.Katayama, M.Tsukagoshi; "FORTRAN and tuning utilities aiming at ease of use of a supercomputer", Proceedings of FJCC'86, 1986.11, pp.1034-