# JSS V システムの効率的利用について

長嶺　七海，　百瀬　真太郎
日本電気株式会社

# Effective use of JSS V System

by
Nanaumi Nagamine, Shintaro Momose, NEC Corporation

**ABSTRACT**

An SX-9 system has been installed as JSS V system to make good use of the program property written for the vector supercomputers. But the performance of the components of the supercomputer is not equally improved because of the technology trends and restrictions of hardware implementation. This paper introduces the performance characteristics of the SX-9 by comparing with those of the SX-6 and the knack of coding from the users' point of view especially focusing on the memory bandwidth and the performance of the processor (B/F ratio) and ADB (Assignable Data Buffer), a new feature of the SX-9. In some cases, the application programs run three times as fast as before by adding some directives and modifying a few loops. It is fruitful for the owner and the other users as well.

## １．はじめに

JAXA JSS V システムは，ベクトル型スーパーコンピュータ向けに書かれたプログラム資産を有効活用するため，NEC 製 SX-9[1)]が採用された．SX-9 は，単一コアの倍精度浮動小数点演算性能 100GFLOPS 超を世界で初めて実現，単一ノードは 16CPU の SMP 構成により，演算性能 1.6TFLOPS，共有メモリ容量 1TB を提供するベクトル型スーパーコンピュータである．本稿では，利用者の観点から SX-9 のハードウェア性能特性，及びその性能をより引き出すためのチューニング技法について，実際のアプリケーションを例に述べる．

## ２．SX-9 のハードウェア特性

### 2.1. SX-9 の特徴

SX-9 には大きく以下の 3 点の特徴がある．

①1CPU 当たり 102.4GFLOPS のパワフルプロセッサ
②1CPU 当たり 256GB/s の超広帯域メモリバンド幅
③16CPU/1TB 共有メモリの巨大 SMP ノード

SX-9 の CPU は単一コア構成であり，単一チップ当たりの倍精度浮動小数点演算性能において世界初となる 100GFLOPS 超を実現し，同時に演算性能にバランスした 256GB/s という超広帯域メモリバンド幅を実現している．図 1 にノード構成を示す．ノード構成は 16 個の CPU，及び 1T バイトのメモリを，均一に接続した SMP であり，総演算性能 1.6TFLOPS，総メモリバンド幅 4TB/s である．

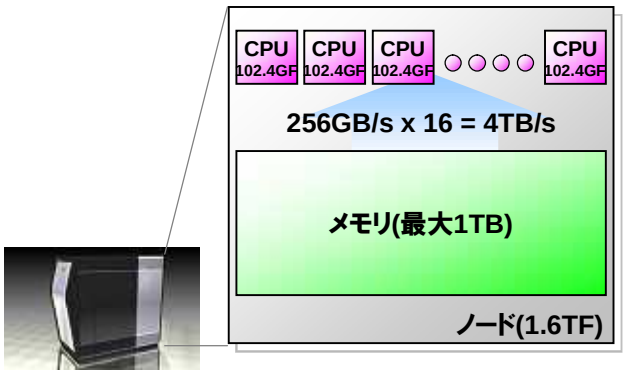


図 1　SX-9 のノード構成

表 1 に，角田 NSE，相模原 SSS で採用された SX-6 システムと JSS V システムで採用された SX-9 システムの性能比較を示す．表 1 に示されるように，各コンポーネントにより 3.2 倍から 12.8 倍の性能向上を実現した．特に，ベクトル演算性能は，12.8 倍の高速化を実現している．

表 1　SX-6 と SX-9 の性能比較

| | SX-6 | SX-9 | 倍率 |
|---|---|---|---|
| クロック | 1GHz | 3.2GHz | 3.2x |
| ベクトル演算性能 | 8GF | 102.4GF | 12.8x |
| スカラ演算性能 | 1GF | 3.2GF | 3.2x |
| ロード性能/CPU | 32GB/S | 256GB/S | 8.0x |
| ストア性能/CPU | 32GB/S | 128GB/S | 4.0x |

(注：相模原 SSS の SX-6 はすべて表 1 の 1.125 倍の性能)

### 2.2. SX シリーズとスカラプロセッサの性能トレンド

図 2 に，SX シリーズとインテル x86 プロセッサファミリの性能向上トレンドを示す．

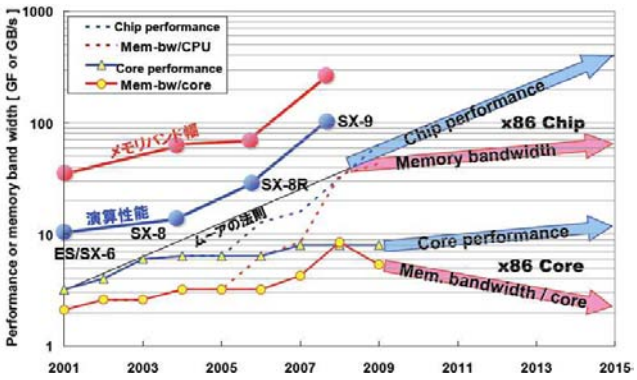


図 2 SX と x86 の演算性能/メモリバンド幅の変遷

x86 プロセッサは，2003 年頃まではムーアの法則に従う形で演算性能が向上している．その後，単一コアの演算性能向上は鈍化し，マルチコア化することにより CPU 演算性能向上率を維持している．しかし，メモリバンド幅の向上は，CPU の演算性能向上と比較して小さく，また，CPU がマルチコア化されるために，今後単一コア当たりのメモリバンド幅は一定，または低下するものと思われる．

SX は，SX-6(2001 年)から SX-9(2008 年)まで，単一コアの演算性能向上による CPU 演算性能向上を維持し，7 年間で 12.8 倍の CPU 高速化を実現した．さらに，高実効性能を実現するために，超広帯域メモリバンド幅向上を維持し，SX-9 は SX-6 の 8 倍となる，単一 CPU 当たり 256GB/s を実現している．

### 2.3. ADB の実装

実際のアプリケーションにおいて，高実効性能を実現するためには，演算性能とバランスしたメモリバンド幅が重要な要素の一つとなる．SX は高実効性能，及び高実行効率を実現するために，CPU 当たり 256GB/s の超広帯域なメモリバンド幅を提供し，演算性能に対するデータ供給能力を示す指標である byte/FLOP(B/F)は，SX-6 が 4.0，SX-9 が 2.5 という高い B/F を実現している．一方，スカラプロセッサは B/F が 1 以下であり，今後更に低下することが予想されている．SX-9 は高 B/F であるが，科学技術計算において更なる高実効性能を実現するために，B/F の強化，メモリレイテンシの短縮，及びメモリバンク競合の回避を目的として，新たに Assignable Data Buffer(ADB)を導入した．

ADB はレジスタとメモリ間に実装される HPC 専用キャッシュ機構であり，従来型のキャッシュと異なり，再利用性の高いデータを選択的に ADB に格納することにより，再利用性の低いデータによるキャッシュ汚染を防ぎつつ，高いキャッシュ利用効率を実現し，実際のアプリケーションにおける実効性能を更に高めるものである．

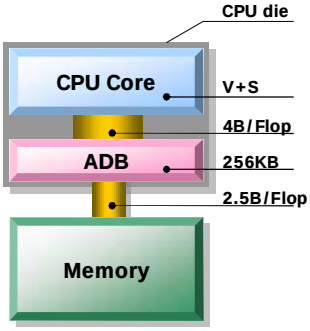


図 3　ADB

### 2.4. メモリレイテンシとバンクサイクル時間

メモリアクセス性能を支配する要因として，メモリバンド幅の他に，1 つのメモリアクセス命令を開始してから，終了するまでにかかる時間を示すメモリレイテンシがある．一般に，メモリレイテンシが長いと，メモリアクセス開始から終了までに時間を要することとなり，実効性能低下の要因となる．SX は，ベクトルロード命令，ベクトルストア命令などにより，単一命令で大量のデータを扱い，かつ，それら命令を図 4a に示すように，連続実行するために，メモリレイテンシの隠蔽が可能であり，高実効性能を実現する．

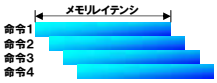


図 4a　メモリアクセス連続実行


図 4b　メモリアクセス連続実行不可による性能低下

一方，メモリアクセスの連続実行ができない場合を図 4b に示す．メモリアクセスにおいて，リストベクトル（間接ベクトル）等，先行メモリアクセス命令と後続メモリアクセス命令間に依存関係がある場合(または依存関係が不明な場合)，実行順序保証のために後続メモリアクセス命令は先行命令の終了待って，実行開始する必要があり，図 4a のようなメモリアクセスの連続実行ができない．このため，メモリアクセス性能は CPU 当たりの総メモリバンド幅ではなく，メモリレイテンシに支配されることになる．

メモリレイテンシと SMP サイズはトレードオフの関係にある．SX-9 は，高実効性能実現やプログラミングのしやすさを実現するために 1TB のメモリを 16CPU により完全対称形で共有する SMP アーキテクチャを採用している．一方，このような大規模 SMP を実現するために各 CPU と共有メモリ間の接続が複雑なものとなり，CPU とメモリ間の接続部がメモリレイテンシの 40%以上を占める．

また，図 5 に示すようにメモリアクセス時に複数のメモリアクセス命令が同一バンクにアクセスすることによりバンク競合が発生した場合，一方のメモリアクセスが他方のメモリアクセス終了を待つ必要があり，メモリアクセスレイテンシが伸びる．

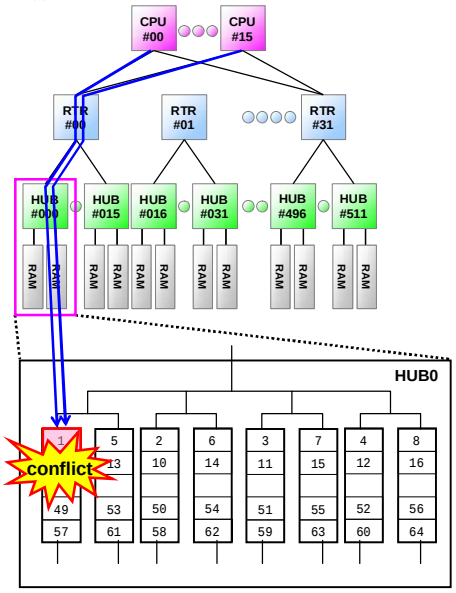


図 5　バンク競合イメージ

1 つのバンクがアクセスされてから，次にアクセス可能になるまでの時間をバンクサイクル時間と呼ぶ．SX-6 以降の SX シリーズのバンクサイクル時間を図 6 に示す．バンクサイクル時間は RAM 素子により決定されるものであり，SX の各世代において，バンクサイクル時間はほぼ一定であると言える．SX-8 では，超短レイテンシメモリとして FC-RAM が準備されたが，現在 FC-RAM は妥当なコストでの採用は不可である．

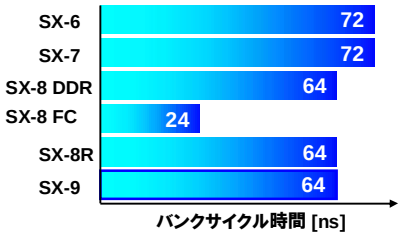


図 6　バンクサイクル時間

SX は SX-6 から，SX-9 まで，CPU のクロックサイクルとしては，3 倍以上高速化されている．一方，バンクサイクル時間はほぼ一定であるため，バンク競合時に伸長されるメモリアクセスレイテンシは一定であるが，SX-9 はクロックサイクルが高速化されている分，バンク競合時に待たされるクロック数が多く，バンク競合が実効性能に与える影響が大きい．このため，SX-9 のメモリは 32,768 という多数のバンクから構成されている．多バンク構成とすることにより，メモリアクセス時のバンク競合を最小限とし，高実効バンド幅の実現が可能となる．

以上より，SX-9 は，100GFLOPS 超の演算性能，及び 256GB/s のメモリバンド幅を有するパワフル CPU16 個により，演算性能 1.6TFLOPS，メモリバンド幅 4TB/s，メモリ容量 1TB の大規模ノード構成を採用することにより，容易な高実効性能の実現を可能とした．更に，多バンク構成，及び ADB の採用により，メモリバンド幅の強化，メモリレイテンシの短縮，及びバンク競合の回避を可能としている．一方で，メモリレイテンシが隠蔽できない場合や，バンク競合によるメモリアクセス速度の低下が，実効性能の低下を引き起こす場合があるため，メモリアクセスの連続実行，及びバンク競合回避のチューニングを行うことにより，高いメモリアクセス性能を引き出すことが必要である．

## ３．チューニング事例

SX-9 のハードウェア性能を引き出すためのチューニングとして，メモリアクセスを軽減する手法 4 点(外側ループのアンローリング，ループ交換によるリストベクトル削除，ADB の利用，外側ループのストリップマイニング)，またソフトウェア処理からハードウェア処理になった MAX/MIN 関数の活用例を挙げる．

### 3.1. 外側ループのアンローリング

アンローリングとは，例 1 のようにループ長を短くして，ループ内の演算数を増やす最適化技法である．

```
   DO I = 1, 1024
      C(I) = A(I) + B(I)
   ENDDO

    ↓  4 段アンローリング

   DO I = 1, 1021, 4
      C(I)   = A(I)   + B(I)
      C(I+1) = A(I+1) + B(I+1)
      C(I+2) = A(I+2) + B(I+2)
      C(I+3) = A(I+3) + B(I+3)
   ENDDO
```

例1　単純なアンローリングの例

しかし，通常ベクトル化される最内側ループに対して適用すると，メモリアクセスが連続から飛びになるなどの理由により性能が悪化する場合がある．一方，通常ベクトル化されない外側ループに対して適用すると，メモリアクセス回数が減り高速化される場合が多い．例えば，例 2 にあるようなコード変形を行うと，最内側ループにおいて，X(I)のロード，ストア，及び A(I,K)のロードがいずれも半分となる．

```
DO J=1,100
  DO I=1,N
     X(I)=X(I)+A(I,K)*B(K,J)
  END DO
END DO

       ↓ 外側ループのアンローリング

DO J=1,99,2
  DO I=1,N
    X(I)=X(I)+A(I,K)*B(K,J)
&             +A(I,K)*B(K,J+1)
  END DO
END DO
```

例 2 外側ループのアンローリング例

例 3 では，より実際のアプリケーションに近い HimenoBMT (http://accc.riken.jp/HPC/HimenoBMT/program2.html)のサイズ L を用いて外側ループアンローリングの効果測定を行った例を示す．なお，ソース改変は行わず，指示行を 1 行追加したのみである．

```
!CDIR OUTERUNROLL=n   OUTERUNROLL 指示行の追加
 DO J=2,jmax-1
  DO I=2,imax-1
   S0=   a(I,J,K,1)*p(I+1,J,K)+a(I,J,K,2)*p(I,J+1,K)
&       +a(I,J,K,3)*p(I,J,K+1)
&       +b(I,J,K,1)*(p(I+1,J+1,K)-p(I+1,J-1,K)
&       -p(I-1,J+1,K)+p(I-1,J-1,K))
&       +b(I,J,K,2)*(p(I,J+1,K+1)-p(I,J-1,K+1)
&       -p(I,J+1,K-1)+p(I,J-1,K-1))
&       +b(I,J,K,3)*(p(I+1,J,K+1)-p(I-1,J,K+1)
&       -p(I+1,J,K-1)+p(I-1,J,K-1))
&       +c(I,J,K,1)*p(I-1,J,K)+c(I,J,K,2)*p(I,J-1,K)
&       +c(I,J,K,3)*p(I,J,K-1)+wrk1(I,J,K)
        SS=(S0*a(I,J,K,4)-p(I,J,K))*bnd(I,J,K)
        WGOSA=WGOSA+SS*SS
        wrk2(I,J,K)=p(I,J,K)+OMEGA *SS
    enddo
  enddo
```

例 3　HimenoBMT による効果測定

例 3 において，斜体で示す配列 p の J 次元方向アクセスについて，それぞれ J-1, J, J+1 のアクセスがあるため，外側のループアンローリングにより，メモリアクセス数の削減が期待できる．この効果はアンローリング段数(例 3 の 1 行目 n に相当)を増やすほど大きくなるが，過度の増加は最内側のコードが過大となり，ベクトルレジスタ溢れを引き起こすため逆効果となる．一般に最適なアンローリング段数はアプリケーションに依存し，完全に理論的に予測するのは困難であるため実験的に求めることになる．この例の場合，表 2 のようにアンロール 16 段の場合に最大実効性能が得られた．

表 2 Himeno-BMT による効果測定

| 段数 | なし | 2 | 4 | 8 | 16 | 32 | 64 |
|---|---|---|---|---|---|---|---|
| 性能値 | 16.5GF | 20.9GF | 25.8GF | 28.7GF | 29.0GF | 22.9GF | 12.4GF |

最大75%高速化

### 3.2. ループ交換によるリストアクセスの回避 [2)]

次に最内側のループを入れ替えることでリストアクセスを避ける例について述べる．

```
(オリジナル)
DO K=1,KF
  DO M=1,MF
    DO J=1,JF
      DO I=2,IF
        IL2=ilir(I,1)
        IL1=ilir(I,2)
        IR1=ilir(I,3)
        IR2=ilir(I,4)
        STBC=wstbc(I)
        DQR0=ain(I,J,K)
        DQR1=ain(IR1,J,K)
        DQR2=ain(IR2,J,K)
        DQL0=ain(I-1,J,K)
        DQL1=ain(IL1,J,K)
        DQL2=ain(IL2,J,K)
             :

(チューニング後)
DO M=1,MF
  DO I=2,IF
    IL2=ilir(I,1)
    IL1=ilir(I,2)
    IR1=ilir(I,3)
    IR2=ilir(I,4)
    STBC=wstbc(I)
   DO JK=1,JF*KF
     DQR0=ain(JK,1,I)
     DQR1=ain(JK,1,IR1)
     DQR2=ain(JK,1,IR2)
     DQL0=ain(JK,1,I-1)
     DQL1=ain(JK,1,IL1)
     DQL2=ain(JK,1,IL2)
          :
```

例 4　リストアクセス回避例(1)

例 4 におけるオリジナルでは，最内側の I 方向のアクセスがリストになっているため，高実効性能実現を阻害する要因となっている．J 方向と K 方向のループを一重化し，さらに連続アクセスとなるように配列の入れ替えを行うことにより，図 7 に示すように約 2.4 倍の性能向上が得られた．


図 7　リストアクセス回避例(1)

この手法は，配列の入れ替えを行っているため，一般にソース修正が広範囲に及ぶ可能性がある．I と J のループを入れ替えるだけでも一定程度の効果が期待できる．

例 5 に，IF 文による条件判定をループ外に出すことによるリストアクセス回避法を示す．

```
(オリジナル)
  DO  IS=1,ITE
    DO  JJ=0,CL-1
      IX=IBASE(I)+LL*JJ
!CDIR NODEP
      DO KK=0,LL -1
        IF(LOC(KK,IS).NE.KK)THEN
          IVAL=LOC(KK,IS)
          VOL(IX+KK)=VOL(IX+KK)
 &              +ELM(KK,IS)
 &              *VT(IX+IVAL)
          VOL(IX+IVAL)=VOL(IX+IVAL)
 &              +ELM(KK,IS)
 &              *VT(IX+ KK)
        ENDIF
      ENDDO
    ENDDO
  ENDDO

(チューニング後)
  DO  IS=1,ITE
    DO KK=0,LL -1
      IF(LOC(KK,IS).NE.KK) THEN
        IVAL=LOC(KK,IS)
!CDIR NODEP
        DO  JJ=0,CL-1
          IX=IBASE(I)+LL*JJ
          VOL(IX+KK)=VOL(IX+KK)
 &              +ELM(KK,IS)
 &              *VT(IX+IVAL)
          VOL(IX+IVAL)=VOL(IX+IVAL)
 &              +ELM(KK,IS)
 &              *VT(IX+ KK)
        ENDDO
      ENDIF
    ENDDO
  ENDDO

---
!CDIR NODEP は直後のループ中の依存関係を否定しベクトル化を指示する指示行.
```

例5　リストアクセス回避例(2)

例 5 のオリジナルでは，IF 文が最内側ループの内部にあり，それに続く VT,VOL の参照，代入がリストアクセスとなっている．チューニング例では，最内側ループに IF 文の条件判定後に置くことにより，VT,VOL のリストアクセスを回避している．

表 3　リストアクセス回避例(2)

| | オリジナル | チューニング後 |
|---|---|---|
| 実行時間(秒) | 11 | 3 |
| MFLOPS | 345 | 695 |
| ベクトル長 | 128 | 99 |
| ベクトル演算率(%) | 98.6 | 95.1 |

IF 文がベクトル化される場合，文中の処理は真偽にかかわらず全て実行され，マスク処理により真の場合だけが有効化される．本例では，チューニングにより IF 文がベクトル対象外となったことから，真の場合のみ演算が実行されるようになり，ベクトル長とベクトル演算率は低下しているものの，実行時間は 3 分の 1 以下に高速化した．

### 3.3. ADBの利用

SX-9 の新機能である ADB は，汎用スカラプロセッサと異なり，ADB に格納するデータを指示行で指定する．これにより，必要なデータだけを ADB に格納し，容量が 256KB の ADB の利用効率を最大限上げることが可能となる．また，ADB はユーザアプリケーションが使用しない

場合，自動的にスカラキャッシュ，及びベクトルレジスタの一時退避領域に割り当てられる．ADB をユーザアプリケーションが使用する場合(ADB 使用の指示行記述時)は，指定されたデータ以外は ADB に格納されなくなるため，高実効性能のための ADB の最適利用については実験的に最適値を得る必要がある．

ADB は同一ループ中で同一配列を複数回参照する場合に効果を発揮する．例 5 では，配列 W は J 方向にのみ次元を持ち，外側ループの繰り返し毎に同じ内容がロードされる．このため，W を ADB に格納することにより，I=2 以降の配列 W のロード時間を短縮できる．この例では約 1.7 倍の高速化を実現した．

```
      DO I=1,NMAX
!CDIR ON_ADB(W)
      DO J=1,NMAX
        WD(I)=WD(I) + GD(J,I) * W(J)
      ENDDO
     ENDDO
```

例 6　ADB の利用例(1)

例 7 では，NN と KK の値が異なるため，ループ 40 とループ 50 は融合できないが，配列 NU の再利用性があるため， ADB に格納することにより高速な処理が可能となる．本例では図 8 の通り，約 1.5 倍の高速化を実現した．

```
   DO 60 JJ = 1, NN
!CDIR ON_ADB(NU)
     DO 40 II = 1, NN
       MU = .... B(JJ) * A(L(II))
       IF (MU.GT.SC)  NU(II) = MU
           :::
  40  CONTINUE
!CDIR ON_ADB(NU)
     DO 50 II = 1, KK
       MU = NU(II) * S
         :::
  50 CONTINUE
         :::
   60 CONTINUE
```

例 7　ADB の利用例(2)

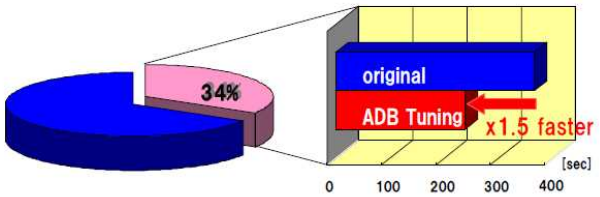


図 8　ADB の利用例(2)

### 3.4. 外側ループのストリップマイニング [2)]

例 8 にストリップマイニングの例を示す．ストリップマイニングとは，ベクトルレジスタ長(SX-9 の場合 256)以上のループをベクトルレジスタ長以下に分割する手法である．

```
      DO I=1,1000
           A(I)=B(I)+C(I)
      ENDDO

                ↓      Strip-mining

      DO i=1,1000,maxvl
           l=MIN(1000-i,maxvl-1)
           A(i:i+l)=B(i:i+l)+C(i:i+l)
      ENDDO
---
maxvl はベクトルレジスタ長
```

例 8　単純なストリップマイニングの例

本手法を外側ループに適用することにより，最内側ループ内の演算がベクトルレジスタ上で実行され，アプリケーションにより程度の差はあるが，メモリアクセス回数が低減される．

```
(オリジナル)
  DO K=1,L
    DO J=1,M
      DO I=1,N
        SUM(I,K,1) = SUM(I,K,1)
 &        + A(J,K,1)*B(I,J,1)
 &        - A(J,K,2)*B(I,J,2)
        SUM(I,K,2) = SUM(I,K,2)
 &        + A(J,K,1)*B(I,J,2)
 &        + A(J,K,2)*B(I,J,1)
      ENDDO
    ENDDO
  ENDDO

(チューニング後)
  DO K=1,L
    DO I2=1,N,256
      DO J=1,M
!CDIR SHORTLOOP
        DO I=I2,MIN(N,I2+255)
          SUM(I,K,1) = SUM(I,K,1)
 &          + A(J,K,1)*B(I,J,1)
 &          - A(J,K,2)*B(I,J,2)
          SUM(I,K,1) = SUM(I,K,1)
 &          + A(J,K,1)*B(I,J,2)
 &          + A(J,K,2)*B(I,J,1)
        ENDDO
      ENDDO
    ENDDO
  ENDDO
---
!CDIR SHORTLOOP は直後のループ長がベクトルレジスタ長以下であることを指示する指示行
```

例 9　外側ストリップマイニングの例

例 9 も前例同様にストリップマイニングの例を示す．この例では，配列 SUM が J 方向の次元を持たないことから，I 方向でストリップマイニングしている．これにより I 方向，J 方向でループが実行されている間はメモリアクセスが行われなくなり，図 9 に示す通り約 37%の高速化が図られた．

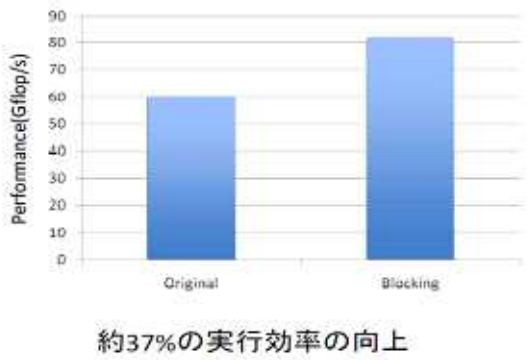


図 9　外側ストリップマイニングの例

3.5. IF 文代替としての MIN 関数の利用 [2)]

SX-9 では，これまでソフトウェア処理であった MAX/MIN 関数をハードウェア処理することにより，高速化を図っている．例 10 では，配列 QV の値を見て，負数があれば，FLAG_EXECUTE を.TRUE.にしている．作業変数と MIN 関数を使い，条件判定をループ外に出すことで，図 10 に示す通り，約 3.3 倍の高速化を実現した．

```
(オリジナル)
DO KZ = 1, NZ
  DO JY = JYSTLC, JYENLC
    DO IX = IXSTLC, IXENLC
      IF (QV(IX, JY, KZ) < 0.0) THEN
        FLAG_EXECUTE = .TRUE.
      ENDIF
    END DO
  END DO
END DO

(チューニング後)
DO KZ = 1, NZ
  DO JY = JYSTLC, JYENLC
    DO IX = IXSTLC, IXENLC
      WRK = MIN(QV(IX, JY, KZ),WRK)
    END DO
  END DO
END DO
IF (WRK .LT. 0.0) FLAG_EXECUTE = .TRUE.
```

例 10　MIN 関数の利用

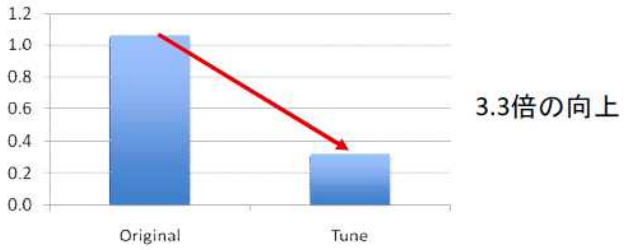


図 10 MN 関数の利用

## ４．結論

以上より SX-9 は高実効性能を実現するスーパーコンピュータであるが，ハードウェア特性を意識したチューニングを実施することにより，更に高い実効性能を実現可能であることを示した．特に，ADB の利用，メモリアクセスレイテンシの隠蔽，及びバンク競合回避チューニングが有効である．また，本稿で述べたチューニング手法の多くは，ベクトル型スーパーコンピュータ特有のものではなく，スカラ型システムにも効果のある手法である．

個々のアプリケーションの性能が向上することにより，そのアプリケーションの利用者はもちろんのこと，計算資源をより多くの利用者に提供でき，システム全体の有効利用に寄与できるため，我々はメーカーの立場として，より簡便でより効果のあるチューニング手法を継続して紹介していきたい．

## 参考文献

1) S.Nakazato, S.Tagaya, N.Nakagomi, T.Watai and A.Sawamura. “*Hardware technology of the SX-9 (1) main system*”, NEC TECHNICAL JOUNAL, 2008

2) T.Soga, A.Musa, Y.shimomura, K.Itakura, K.Okabe, R.Egawa, H.Takizawa and H.Kobayashi. “*Performance Evaluation of NEC SX-9 using Real Science and Engineering Applications*”, In proceedings of the ACM/ IEEE SC2009 conference, November 2009