社団法人 情報処理学会 研究報告
IPSJ SIG Technical Report

2005－EVA－14 (1)
2005／8／5

SMPにおけるスレッド並列の台数効果と高速化手法について

杉崎 由典[1] 青木 正樹[1] 義久 智樹[2] 金澤 正憲[2]
[1]富士通（株） [2]京都大学

京都大学学術情報メディアセンターでは、スーパーコンピュータをベクトル機からスカラ並列機にリプレースした。プログラムを作成する研究者の関心事は、自動ベクトル化機能に比べて、自動並列化機能がどの程度であるかということである。この論文では、ベクトル機用に用いていたベンチマークプログラム4本を用いて、スカラ並列機であるSMPの台数効果に関する評価を行う。ベンチマークプログラムそのままでは、あまり台数効果がでないプログラムがあったが、並列化チューニングでよく知られている、ループ分配、ループ交換、ループ融合という技法を用いれば、台数効果が画期的に改善されることが判った。また、ベンチマークをMPIで書き直したが、その台数効果は、自動並列の場合と変わらないものと判断できた。

On Thread-Parallel Performances of the SMP Machine and Tuning

Yoshinori SUGISAKI[1], Masaki AOKI[1], Tomoki YOSHIHISA[2], Masanori KANAZAWA[2]
[1]Fujitsu Limited, [2]Kyoto University

Academic Center for Computing and Media Studies in Kyoto University replaced its super computers from vector machines to scalar machines. The researchers are interested in how effectively scalar machines automatically parallelize programs comparing with vector machines. In this paper, by using 4 benchmark programs for vector machines, we evaluate parallel performances under the SMP(Symmetric Multi-Processor). Although parallel performances in two of original programs are relatively low, they are dramatically improved by tuning techniques for parallelization, such as loop-division, loop-exchange, and loop-integration. Moreover, we manually modify programs for MPI and evaluate them. It is shown that the performances under manual MPI are equivalent to those under automatically parallelization.

1. はじめに

京都大学学術情報メディアセンターでは、スーパーコンピュータを、2004年3月に、ベクトル並列機（富士通 VPP800/63）から、SMP クラスタ（富士通 PRIMEPOWER HPC2500/12 ノード）に置き換えた。ベクトル計算機は約20年間に5機種が導入され、ハードウェアに関する改善、自動ベクトル化技術の発展により、スーパーコンピューティング分野で大きな役割を演じてきた。

しかし、マイクロプロセッサの長足の進歩により、ベクトル機は必ずしも絶対的な存在ではなくなってきたと考えられる。ベクトル演算器、スカラ演算器、ベクトルレジスタといったハードウェアは複雑であり、製品開発の期間とコストの問題が考えられる。

一方、スカラプロセッサの進歩は留まるところを知らないばかりか、その開発周期も短く、次々と高速化された製品が出現している。また、共有メモリ型コンピュータにおいて、CPU 台数の増加、コンパイラによる自動並列化などの技術進歩もみられた。このような観点から、スカラ並列機は将来有望であると考えられるため、京大センターでは、SMP クラスタの導入に至った。

本論文では、京大センターでベクトル計算機の性能評価に用いられていたベンチマーク4本を取り上げ、SMP クラスタで実行し、その並列台数効果を測定した。台数効果の少ないベンチマークに対して、プログラムの書き換えを行って実行させ、台数効果が著しく改善されることを確認した。また、MPI で書き換えたベンチマークを実行し、MPI による並列処理との比較を行った。

2. ベンチマークについて

ここで用いたベンチマークプログラムは、4本で、affine、 grad、 product5、 shift3 と呼んでいる。概要を表 1 に示す。

表1：ベンチマーク概要

名称	プログラムの概要
affine	配列の間接参照
grad	近傍との差分
product5	行列積
shift3	値の伝播（他の影響を含む）

それぞれのプログラムの主要部分を附録に記した。どれも2次元配列を扱うプログラムで、配列の大きさを増減することにより、いろいろな演算速度のコンピュータで実行できるようになっている。

3. SMP クラスタでの測定結果

affine, grad, product5, shift3 の 4 種類のプログラムについて、オリジナルソースのまま並列化した場合、高速化のためのチューニングを行った場合、MPI で記述した場合について実測し、３つの場合の結果をまとめて図示する。測定結果については、特に断らない限り、1CPU 向きの最適化を施したプログラムの実行時間を 1 として表示している。また、実行時間とともにモニタ機能から収集された性能情報をもとに、結果を分析する。

ここで使用した測定マシンと条件（パラメータなど）を表 2 に示す。

表 2. 測定マシンと条件

実行環境	富士通 PRIMEPOWER HPC2500 (SPARC64V 1.82GHz) Parallelnavi2.4
並列数	128CPU, 512GB の 1 ノード上にて 1～120CPU を使用
翻訳時オプション	-Kfast_GP2=3, prefetch=4, parallel=3, V9, largepage=2, hardbarrier -w

3.1 オリジナルソースプログラムの場合

各プログラムに手を加えることなく実行させて得られた結果から判明した性能上の問題を以下に示す。

(1) affine

aintを含むループコストが高い。aint数学ライブラリが多数呼び出されている。そのため、関数呼出しのオーバヘッドが大きくなっている。

(2) grad

TLB ミス、L2 キャッシュミスが多発している。

同一ループ内に配列の次元アクセス順序が異なる文が存在するため、連続アクセスとストライドアクセスが混在している。混在によりコンパイラの最適化が効かず、ストライドアクセスによる TLB ミスにより性能が劣化している。

(3) product5

matmul 並列ライブラリ呼出しに展開されている。

(4) shift3

L2 キャッシュミスが多発している。

10 個の do 文があるが、1 つ目の do 文を実行した後の do 文実行ではアクセスするデータがキャッシュに残っていない。そのためキャッシュミスが多発し、性能が劣化している。

3.2 最適化を行った場合

オリジナルソースコードの性能検証から、性能影響要因への対処方法を検討・実施し、性能向上を図った。実施した性能向上策と効果を以下に示す。

(1) affine

関数呼出しのオーバヘッドを抑止するため、aint の手動による展開を行った。図 a 参照。aint の展開方法による差を表 3 に示す。(2) と (3) の違いは、AINT の引数が int の範囲を超えた際の考慮がある (2) かないか (3) であり、引数の取りうる値の範囲によっては (3) が有効となる。結果を図 1 に示す。

```
C(I,J)=A(INT(FI),INT(FJ))*(FI-AINT(FI)+1)*
    (FJ-AINT(FJ)+1)
                ↓ 変更
C(I,J)=A(INT(FI),INT(FJ))*(FI-real(int(FI))+1)*
    (FJ-real(int(FJ))+1)
```

図a. チューニング例(aint)

表３：チューニング効果

チューニング方法	CPU 時間（秒）
(1) オリジナル	437.52
(2) コンパイラによるインライン展開	344.37
(3) 手動によるインライン展開	275.19

注 (N=100000, 4CPU)

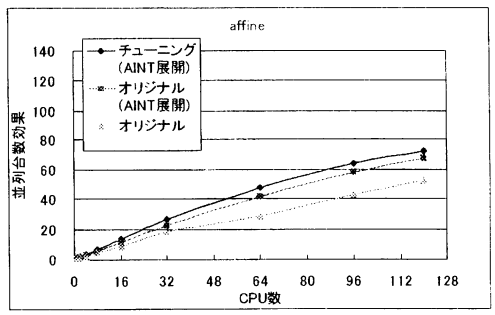

図１. affine チューニング効果

(2) grad

メモリアクセスを全て連続アクセスにするため、ループ分配及びループ交換を行った。図b参照。これにより、データの局所性が高まり、モニタ機能による解析情報から、TLBミス及びL2キャッシュミスが削減されたことを確認した。

```
      DO 300 J=1,N-1
        DO 300 I=1,N
          B(I,J)=A(I,J+1)-A(I,J)
300       C(J,I)=A(J+1,I)-A(J,I)
                  ↓ 変更
      DO 300 J=1,N-1
        DO 300 I=1,N
300       B(I,J)=A(I,J+1)-A(I,J)
      DO 310 I=1,N
        DO 310 J=1,N-1
310       C(J,I)=A(J+1,I)-A(J,I)
```

図b. チューニング例 (grad)

チューニング前後のメモリ性能を表4に示す。図2に台数効果を示す。この2つの結果からループ分解と交換が非常に効果があったことが判る。

表4. メモリ性能(grad)

チューニング	L2キャッシュミス(%)	TLB ミス(%)	メモリアクセス(%)
前	32.9950	0.5789	98.39
後	3.4604	0.0000	41.98

(N=50000, CPU=4)

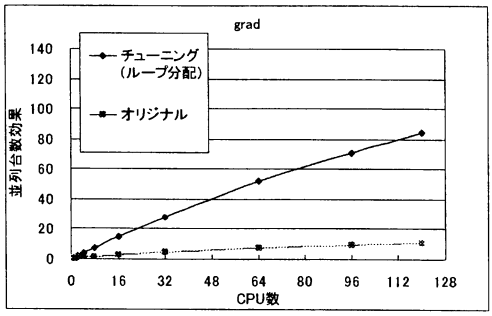

図2. grad チューニング効果

(3) product5

matmul 並列ライブラリ呼出しに展開されているため、図3に示すとおり、十分高速な結果が得られているが、ループ分配最適化を行うことで、配列のゼロクリア処理を並列化する余地があった。図c参照。余り大きな効果はないが、多並列の際に効果が現れている。

```
      DO 300 I=1,N
        DO 300 J=1,N
          C(I,J)=0
          DO 300 K=1,N
300         C(I,J)=C(I,J)+A(I,K)*B(K,J)
                  ↓ 変更
      do j=1,n
      do i=1,n
        C(I,J)=0
      enddo
      enddo
      DO 300 I=1,N
        DO 300 J=1,N
          DO 300 K=1,N
300         C(I,J)=C(I,J)+A(I,K)*B(K,J)
```

図c. チューニング例 (product5)

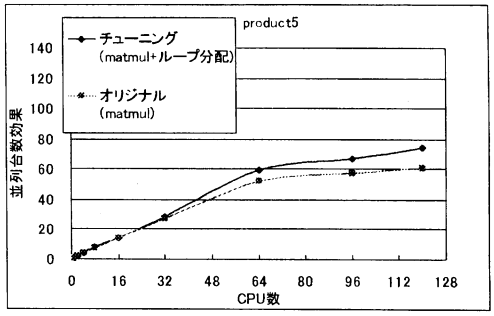

図3. product5 チューニング効果

(4) shift3

ループ融合を行うことで、内側のループでアクセスする次元方向の配列領域が局所化され、次のループ処理の際にはその次元方向のデータは全てオンキャッシュとなる。チューニングについては、図dを参照。図4に台数効果を示す。

チューニング前後のメモリ性能を表5に示す。明らかに、非常に大きな効果があったことが判る。

表5. メモリ性能(shift3)

チューニング	L2キャッシュミス(%)	TLB ミス(%)	メモリアクセス(%)
前	0.9596	0.0000	42.89
後	0.0998	0.0000	10.07

(N=50000, CPU=4)

```
      DO 10 J=1,NN
      DO 10 I=1,NN
        A(I,J)=A(I+1,J)+0.48*B(I,J)+0.122*(B(I-1,
        J)+B(I,J-1)+B(I+1,J)+B(I,J+1))
10    CONTINUE
       ...
      DO 100 J=1,NN
      DO 100 I=1,NN
        A(I,J)=A(I+2,J)+0.46*B(I,J)+0.118*(B(I-1,
        J)+B(I,J-1)+B(I+1,J)+B(I,J+1))
100   CONTINUE
                ↓ 変更
      DO J=1,NN
        DO 10 I=1,NN
          A(I,J)=A(I+1,J)+0.48*B(I,J)+0.122*(B(I-
          1,J)+B(I,J-1)+B(I+1,J)+B(I,J+1))
10      CONTINUE
        ...
        DO 100 I=1,NN
          A(I,J)=A(I+2,J)+0.46*B(I,J)+0.118*(B(I-
          1,J)+ B(I,J-1)+B(I+1,J)+B(I,J+1))
100     CONTINUE
      ENDDO
```

図 d. チューニング例 (shift3)

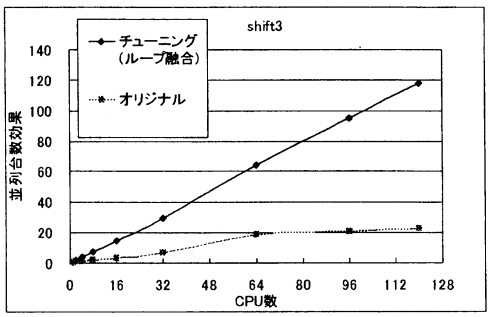

図４. shift3 チューニング効果

3.3 MPI で書き直した場合

チューニングを施したソースコードを用い、これを MPI へ書き換えを行った。書き換えに際しては、主要ループ内での通信が発生しないような方法にて行っている。MPI への書き換え方法と性能結果を以下に示す。

(1) grad

スレッド並列と同様に、配列の 2 次元目を並列化した。

主要ループ中には通信処理がないこともあり、並列台数効果もスレッド並列と同等の性能を示している。図５参照。

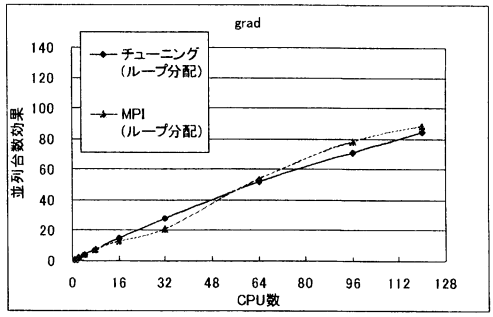

図５. grad 自動並列と MPI の性能比較

(2) product5

一部の配列を各プロセスで重複して保持することにより、主要ループ中に通信処理が入らないようにしている。その影響で matmul 関数ライブラリ呼出しに展開されなくなった。そのため、性能がスレッド並列と比較して大きく低下している。図６参照。

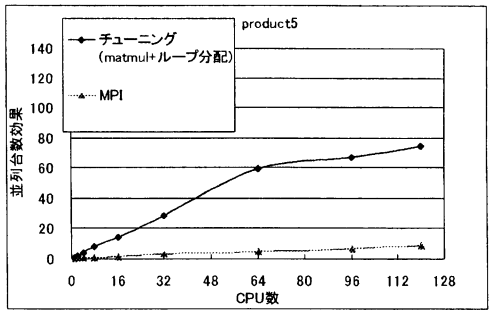

図６. product5 自動並列と MPI の性能比較

(3) shift3

スレッド並列と同様に、配列の 2 次元目を並列化した。

主要ループ中には通信処理がないこともあり、並列台数効果もスレッド並列と同等の性能を示している。図７参照。

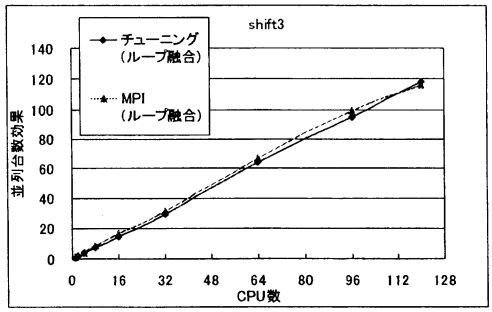

図７. shift3 自動並列と MPI の性能比較

4. VPP との性能比較

Fujitsu VPP5000 の 1CPU で同じプログラムを実測し、HPC2500 の実測値と比較した。さらに、スレッド並列で 32CPU の場合も測定した。表 6 に測定値及び比較結果を示す。効果の欄については HPC2500 の 1CPU を 1 としたた際の性能効果を現している。

VPP5000 はピーク性能 9.6GFlops、HPC2500 はピーク性能 7.28GFlops である。 LINPACK によると、SMP の性能はピークの 50%と換算するのが、妥当と考えられ、VPP5000 と HPC2500 は、約 2.6 倍と推測される。shift3 では、同程度の性能が出ているといえる。なお、affine に関しては、適当な処理時間になるような大きさにすると、主記憶の不足のため、測定できなかった。

表 6 : VPP5000 と HPC2500 の性能比較

Bench mark	Size (N)		VPP5000	HPC2500	
		CPU 台数	1	1	32
grad	500	時間	2,864	25,611	967
		効果	8.94	1.00	26.49
	5000	時間	393,423	2,679,202	101,226
		効果	6.81	1.00	26.47
product5	12000	時間	2,147	903	33
		効果	0.42	1.00	27.36
shift3	1000	時間	14,766	46,520	1,411
		効果	3.15	1.00	32.97
	10000	時間	1,344,125	4,765,837	154,533
		効果	3.55	1.00	30.84

注 : 時間は grad, shift3 が μ sec, product5 が sec。

5. おわりに

京都大学学術情報メディアセンターで用いてきたベンチマークプログラムを用いて、SMP による並列処理の効果を実測し、評価した。共有メモリ型システムにおいて、自動並列化による効果が、十分多数の台数まで (今回は 120CPU まで)、ほぼ線形的に向上することがわかった。

高速処理をするためには、よく知られた並列化手法であるループ分配、ループ交換、ループ融合が大いに有効であることが実証できた。このことから、コンパイラがループ分配、交換、融合などの典型的な並列化技法を自動的に取り入れる、または、プログラマに候補の技法と場所を推定して、助言を行う必要がある。さらに、会話的に並列化機能とその効果を直ちに判定できる機能が不可欠であろう。

自動ベクトル化コンパイラが種々の機能を順次、取り入れたように、自動並列化コンパイラも今後絶えず新しい機能を盛り込むことが重要である。

product5 の行列積のように、並列化、最適化を駆使するよりも、既存の並列ライブラリを用いたほうが、圧倒的に性能が高いことがわかった。その上、プログラムも

```
c=matmul(b,c)
```

と簡潔な形で記述できるので便利である。

したがって、まず、第 1 の解決策として、並列ライブラリの普及にセンターとしては努める必要があると自覚した。一方、コンパイラがソースを読みきって並列ライブラリを組み込むようにすることも非常に有効である。導入当初 (昨年) のコンパイラは、行列積の計算であると読み切る能力が弱かったが、現在は改善されている。

上述したように、自動並列化コンパイラは、ソース解析技術の向上が必要であると考えられる。

SMP マシンは、巨大な主記憶を共有するため、CPU 数が多くなると、メモリアクセスで競合が起こるという問題が指摘されていた。そこで、ベンチマークプログラムを MPI で書き直して、その実行時間を実測したところ、メモリアクセスの競合に関しては、ほとんどないと推測される。

さらに、VPP5000 の 1CPU で実行した結果と比較すると、同じ 1CPU なら VPP5000 が 3 倍から 8 倍程度速いことが確認できた。

これらの結果は、ハードおよびソフトの並列技術の発展に有意義なデータを与えるものと考えている。今後も利用者プログラムを収集して、ベンチマークを整備していきたい。

最後に、助言をいただいた京都大学学術情報メディアセンター岩下武史助教授、富士通㈱荒川征己氏、並びに、関係者各位に深く感謝します。

参考文献

1) 草野義博, 新庄直樹 ; ハイパフォーマンスコンピュータ : PRIMEPOWER HPC, Fujitsu, Vol. 53, No. 6, pp. 444-449, 2002.
2) 富士通 ; Parallelnavi Fortran 使用手引書, マニュアル
3) 富士通 ; Parallelnavi Fortran 文法書, マニュアル

附録１：ベンチマークプログラムの主要ループ部

各プログラムの主要ループ部を以下に示す。

附表１：affine

```
      DO 300 J=1,N
        DO 300 I=1,N
          FI=REAL(I-N/2)/2-REAL(J-N/2)/SQRT(12.0)+N/2
          FJ=REAL(I-N/2)/SQRT(12.0)+REAL(J-N/2)/2+N/2

C(I,J)=A(INT(FI),INT(FJ))*(FI-AINT(FI)+1)*(FJ-AINT(FJ)+1)
     &
+A(INT(FI)+1,INT(FJ))*(AINT(FI)-FI)*(FJ-AINT(FJ)+1)
     &
+A(INT(FI),INT(FJ)+1)*(FI-AINT(FI)+1)*(AINT(FJ)-FJ)
     &
+A(INT(FI)+1,INT(FJ)+1)*(AINT(FI)-FI)*(AINT(FJ)-FJ)
  300 CONTINUE
```

affineでは、配列の間接参照を行っている。

附表２：grad

```
      DO 300 J=1,N-1
        DO 300 I=1,N
          B(I,J)=A(I,J+1)-A(I,J)
  300     C(J,I)=A(J+1,I)-A(J,I)
      DO 400 I=1,N
        B(I,N)=A(I,1)-A(I,N)
  400   C(N,I)=A(1,I)-A(N,I)
      DO 500 J=1,N
        DO 500 I=1,N
  500     A(I,J)=ATAN2(B(I,J),C(I,J))
```

gradでは、配列のｘ軸方向及びｙ軸方向の差分を求め、結果に対してatan2演算を行っている。

附表３：product5

```
      DO 300 I=1,N
        DO 300 J=1,N
          C(I,J)=0
          DO 300 K=1,N
  300       C(I,J)=C(I,J)+A(I,K)*B(K,J)
```

product5では行列積を行っている。

附表４：shift3

```
      DO 10 J=1,NN
      DO 10 I=1,NN

A(I,J)=A(I+1,J)+0.48*B(I,J)+0.122*(B(I-1,J)+B(I,J-1)+B(I+1
,J)+B(I,J+1))
10    CONTINUE
      DO 20 J=1,NN
      DO 20 I=1,NN

A(I,J)=A(I+3,J)+0.44*B(I,J)+0.114*(B(I-1,J)+B(I,J-1)+B(I+1
,J)+B(I,J+1))
20    CONTINUE
      DO 30 J=1,NN
      DO 30 I=1,NN

A(I,J)=A(I+5,J)+0.40*B(I,J)+0.106*(B(I-1,J)+B(I,J-1)+B(I+1
,J)+B(I,J+1))
30    CONTINUE
      DO 40 J=1,NN
      DO 40 I=1,NN

A(I,J)=A(I+7,J)+0.36*B(I,J)+0.098*(B(I-1,J)+B(I,J-1)+B(I+1
,J)+B(I,J+1))
40    CONTINUE
      DO 50 J=1,NN
      DO 50 I=1,NN

A(I,J)=A(I+9,J)+0.32*B(I,J)+0.090*(B(I-1,J)+B(I,J-1)+B(I+1
,J)+B(I,J+1))
50    CONTINUE
      DO 60 J=1,NN
      DO 60 I=1,NN

A(I,J)=A(I+10,J)+0.30*B(I,J)+0.086*(B(I-1,J)+B(I,J-1)+B(I+
1,J)+B(I,J+1))
60    CONTINUE
      DO 70 J=1,NN
      DO 70 I=1,NN

A(I,J)=A(I+8,J)+0.34*B(I,J)+0.094*(B(I-1,J)+B(I,J-1)+B(I+1
,J)+B(I,J+1))
70    CONTINUE
      DO 80 J=1,NN
      DO 80 I=1,NN

A(I,J)=A(I+6,J)+0.38*B(I,J)+0.102*(B(I-1,J)+B(I,J-1)+B(I+1
,J)+B(I,J+1))
80    CONTINUE
      DO 90 J=1,NN
      DO 90 I=1,NN

A(I,J)=A(I+4,J)+0.42*B(I,J)+0.110*(B(I-1,J)+B(I,J-1)+B(I+1
,J)+B(I,J+1))
90    CONTINUE
      DO 100 J=1,NN
      DO 100 I=1,NN

A(I,J)=A(I+2,J)+0.46*B(I,J)+0.118*(B(I-1,J)+B(I,J-1)+B(I+1
,J)+B(I,J+1))
100   CONTINUE
```

shift3では、配列Aのx軸方向に1～10要素離れた要素に対して、配列B及び配列Bの４近傍の係数付きの加算結果を加えている。