計算機アーキテクチャ　113−30
（１９９５．　８．　２５）

スーパースカラ方式とベクトル処理方式の比較
—主記憶アクセス特性に着目して—

保田淑子　田中輝雄(*)　稲上泰弘
yoshikoy@crl.hitachi.co.jp

日立製作所　中央研究所
〒185　東京都国分寺市東恋ヶ窪1-280
(*)日立製作所　汎用コンピュータ事業部
〒259-13　神奈川県秦野市堀山下1

本稿では、演算器構成、主記憶構成が同一と仮定して、積和型および内積型にループ展開等の最適化を施した行列乗算コードを用いてベクトルプロセッサと主記憶をパイプライン化したスーパースカラ型プロセッサの主記憶アクセス特性を比較評価した。評価結果から、ベクトルプロセッサでは、積和型コードのように主記憶アドレスを連続参照する場合には、非常に高い主記憶アクセス性能が得られスーパースカラに比べて相当優位であるが、内積型コードのように主記憶アドレスを非連続参照する場合には、性能がより低下する。一方、スーパースカラはベクトルプロセッサに対して欠点はあるものの、積和型および内積型のどちらのコードに対しても主記憶アドレスを非連続に参照し同様な主記憶アクセス性能が得られ、汎用的であることを確認した。

Comparison of Memory Access between Superscalar and Vector Processing

Yoshiko Yasuda　Teruo Tanaka(*)　Yasuhiro Inagami

Central Research Lab. Hitachi, Ltd.
1-280, Higashi-koigakubo, Kokubunji-shi, Tokyo 185, Japan
(*)General Purpose Computer Division, Hitachi, Ltd.
1, Horiyamashita, Hadano-shi, Kanagawa 259-13, Japan

We make a comparative evaluation of the memory accesses of vector and superscalar processor whose memory accesses are pipelined. In this evaluation, we assume that the configuration of arithmetic units and main memory are the same, and used the most suitable code for each architecture by applying optimizations like loop unrolling. We have found that vector processor achieves higher performance in multiply-add accumulation because of sequential memory access. On the other hand, for inner product, both processors perform stride accesses. Despite the disadvantage of the superscalar processor on multiply-add accumulation, given that both architectures yield similar performances, we conclude that the superscalar processor is general purpose.

1. はじめに

80年代のスーパーコンピュータ隆盛をもたらしたベクトルプロセッサは、複数の演算パイプラインによる高度演算並列と主記憶—ベクトルレジスタ間データ転送のパイプライン化および多バンク化による高スループット主記憶をベースとしている。90年代に入り、RISCスカラプロセッサにおいてスーパースカラ方式が発展し、ベクトルプロセッサに迫る演算並列を実現しつつある。一般に、スカラプロセッサでは主記憶アクセスがパイプライン化されていないため、高い演算能力を維持する手段として、キャッシュを用いている。しかしながら、ベクトルプロセッサで扱うような大規模科学技術計算では、プログラムで扱うデータ量がキャッシュ容量より大きく、キャッシュミスが頻発し、主記憶アクセス時間の全体処理時間に占める割合が増大し、性能が低下する。スカラプロセッサが大規模科学技術計算においても高性能を維持するためには、ベクトル処理のアプローチのように、主記憶データ供給能力の向上及び主記憶アクセスレイテンシの影響削減等、主記憶アクセス性能を強化することが重要である。

主記憶データ供給能力を向上させる手段としては、主記憶アクセスのパイプライン化、主記憶アクセスピッチの短縮、主記憶の多バンク化があり、主記憶アクセスレイテンシの影響を削減する手段としては、データ先読み処理がある。データ先読みは、ある演算命令が必要とするデータを先行的に読み出す処理である。データ先読み処理では、先行する読み出し命令から得たデータをレジスタに格納する。同様な処理として先行して読み出したデータをキャッシュに格納する方法もあるが、本研究ではキャッシュを使用しないベクトル処理方式との比較を行うため、主記憶からレジスタにデータを読み出すデータ先読み処理を対象とする。

スカラプロセッサにおいて上述の主記憶のパイプライン化、データ先読み処理、スーパースカラ方式を用いてベクトルプロセッサと同様な処理を実現するためには、ソフトウェア的にベクトルパイプライニング（ソフトウェアパイプライニング[1]）を行う必要がある。スーパースカラを意識したソフトウェアパイプライニングの一つにモジュロスケジューリングがある[2-3]。モジュロスケジューリングは、対象とするプロセッサで同時動作可能な処理の種類と数、それぞれの処理のレイテンシ等を意識して、命令コードのスケジューリングを行い、プログラムのループ処理の性能向上を達成する。このソフトウェアパイプライニングにより、従来スカラプロセッサが苦手としていたキャッシュの効果が小さいプログラムにおいてもベクトルプロセッサと同等な処理を実現できる。

しかしながら、実際はベクトルプロセッサおよびスカラプロセッサの処理方式の差異により、ループ展開等の最適化レベルのソースコードが同じでも多バンク主記憶上の配列データのアクセスパターンが変わる。従来のベクトル処理方式でも明らかになっているように、アクセスパターンによってはバンク競合が発生し性能が低下してしまう[4]。

そこで、本稿では大規模科学技術計算において、ループ展開等の最適化を施した場合の、ベクトルプロセッサとスカラプロセッサの処理方式の違いによる主記憶上の配列データのアクセスパターンの差異を明らかにすることを目的とする。さらに、演算器構成および主記憶構成が同一であると仮定して、この2つの処理方式の主記憶アクセス性能を評価することにより、それらの特性を明らかにすることを目的とする。評価にあたり、スカラプロセッサとして主記憶アクセスがパイプライン化されているスーパースカラ型プロセッサを想定し、評価ベンチマークとして、LINPACK等の各科学技術計算で重要になる行列乗算コードを使用した。

2. ベクトルプロセッサとスカラプロセッサの処理方式と主記憶アクセス方式

2.1 ベクトル処理

基本的な計算は、(1)主記憶からの配列データ読み出し；(2)読み出したデータを用いた演算；(3)演算結果の主記憶への書き込み；という3つの段階からなる。例えば、

```
      DO 10 I = 1,N
        A(I) = B(I) + C(I)          …式(2.1)
   10 CONTINUE
```

という加算プログラム部分を、ベクトルプロセッサでは、

```
10  VLD B(I)        ；ベクトルBの読み出し
20  VLD C(I)        ；ベクトルCの読み出し
30  VADD            ；ベクトル演算B＋C→A
40  VST A(I)        ；ベクトルAへ格納
```

という形で処理する。ベクトル処理では、A[I]、B[I]、C[I]（I = 1 .. N）のデータをそれぞれまとめてベクトルA、B、Cとし、このベクトルを単位として演算を行う。ここで、DOループ内にそれぞれ10、20、30、40の文があり、ループの制御変数IはI_1, I_2, ..., I_nの値をとると仮定する。ベクトル命令は一連のベクトルデータをまとめて1命令で処理することから、図1(1)に示すように、まず文10に対しI_1, I_2, ..., I_nで実行し、以下同様に実行する。ロード／ストア命令に着目すると、ベクトル処理では、配列Bの1番目要素からN番目要素のロード→配列Cの1番目要素か

らN番目要素のロード→配列Aの1番目要素からN番目要素のストアのように、各配列毎にデータを連続してベクトルレジスタに読み出す。また、ベクトルチェイニング制御により、主記憶からベクトルレジスタへのベクトルデータ読み出しが完了しなくても読み出し直後にベクトル演算を開始することや、ある演算結果を用いて次のベクトル演算を開始できるため、ロード／ストア命令と演算の並列実行が可能である。

2.2 スカラ処理

スカラプロセッサでは、式(2.1)に示す加算プログラムを

```
10  LD B(I)          ；要素B(I)のロード
20  LD C(I)          ；要素C(I)のロード
30  ADD              ；演算B(I)＋C(I)→A(I)
40  ST A(I)          ；要素A(I)へのストア
50  ADDIBF           ；Iの更新とI、Nの比較分岐
```

という形で処理する。図1(2)に示すように、まずループI＝I1に対して文10から50までを実行し、次にI＝I2に対し文を実行する。ベクトル処理では各配列が連続的に読み出されるのに対して、スカラ処理ではB(I)のロード→C(I)のロード→A(I)のストア→B(I+1)のロード→C(I+1)のロード→A(I+1)のストアのように複数の配列が交互に読み出される。

2.3 主記憶アクセスをパイプライン化した場合のスーパースカラ処理

主記憶アクセスをパイプライン化したスーパースカラ型プロセッサでは、データ先読み処理、浮動小数点レジスタ拡張によりベクトルプロセッサと同様な処理を実現できる。スーパースカラ処理は、データ先読み処理により演算命令と後続の別の演算命令に必要なデータのロード命令を並列に実行することができ、主記憶アクセスレイテンシの影響で待ち状態になるプロセッサを常に稼動させることができる。データ先読み処理では演算命令を連続実行させるために、データ先読み命令も連続実行する必要があり、結果として主記憶の連続アクセスを可能にするパイプライン化や先読みデータを格納するための多数のレジスタが必要になる。

一方、数値計算プログラムは、式(2.1)に示すように、データのレジスタへの読み出し／演算／書き込み処理の繰り返しである。この繰り返し処理を利用して多数のレジスタを効率よく使用するためにさまざまなレジスタ管理機構が提案されている[2-3][5-8]。一般に、レジスタ番号リネーム方式と呼ばれるこれらのレジスタ管理機構では、レジスタの指定をハードウェアポインタが指示する物理番号からの相対値によって行う。ハードウェアポインタの値は、ループの繰り返し処理ごとに更新し、繰り返し処理ごとのレジスタの使い分けを保障する。

このようなスーパースカラ型プロセッサでは、式(2.1)に示す加算プログラムにおいて

```
10  PLD (I+1)  ；ループ(I+1)の要素のロード
20  ADD (I)    ；ループ(I)の演算B(I)＋C(I)→A(I)
30  PST (I-1)  ；ループ(I-1)の要素へのストア
40  ICP        ；ハードウェアポインタの更新
```

という形の処理を繰りかえす。各演算で使用するロード／ストア命令は主記憶アクセスのパイプライン化によりあらかじめ連続してノンブロックに実行できるため、N個のベクトルデータを連続して発行するベクトル命令と同様な効果が得られる。また、スーパースカラ処理によりロード／ストア命令と演算命令の並列実行が可能であり、ベクトルチェイニングと同様な効果が得られる。しかしながら、根幹はスカラ処理であるため、B(I+1)のロード→C(I+1)のロード→A(I-1)のストア→B(I+2)のロード→C(I+2)のロード→A(I)のストアのように複数の配列が交互に読み出される。

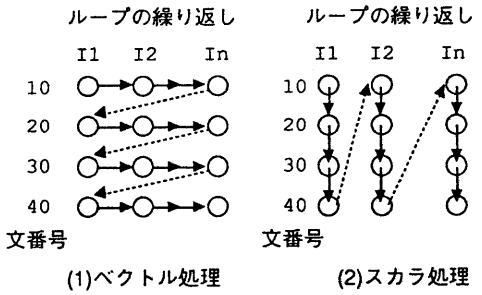

図1 ループの演算順序

2.4 主記憶上の配列データ参照方式の違い

ベクトルプロセッサでは、主記憶データ供給能力を向上させるために、主記憶を多バンク構成にしている。さらに、主記憶アクセスサイクルピッチを短縮するためにインターリーブ方式でアドレス付けし、一次元配列データを順に格納する。科学技術計算で現れるA[I,J]のような多次元配列データは、連続した一次元配列に格納される。

式(2.1)のケースでは、DOループカウンタがIであり、Iが1ずつ増加するため、アドレス連続で格納されている配列データを順次参照することになる。このとき、主記憶アクセス時間に対してバンク数が多ければ、主記憶アクセス時間内に同じバンクをアクセスすることにより発生するバンク競合はない。

一方、主記憶アクセスをパイプライン化したスー

バースカラプロセッサが多バンク主記憶をアクセスする場合、まずループカウンタIにおいて、種類の異なる配列データを順次参照し、続いてループカウンタI+1の異なる種類の配列データを順次参照する必要があり、主記憶アドレス非連続アクセスとなり、バンク競合が発生する可能性がある。

3. 性能評価方法

3.1 性能評価の前提

本性能評価における要素プロセッサの前提事項を以下に示す。ベクトルプロセッサ、スーパースカラプロセッサとも演算器構成および主記憶構成が同一であると仮定した。

(1)命令処理方式：2命令並列実行可能
　(浮動小数点乗加算命令有り)
(2)ロード／ストアパイプ：1本
(3)加算パイプ／乗算パイプ：各1本、計2本
(3)浮動小数点演算レイテンシ：2マシンサイクル
(4)ロード／ストア命令：ベースレジスタ値更新可
(5)浮動小数点レジスタ数：制限なし
(6)主記憶構成：16バンクインターリーブ方式
(7)主記憶アクセス時間：16マシンサイクル
(8)主記憶アクセスレイテンシ：40マシンサイクル
(9)ロード／ストアデータサイズ：8バイト
(10)ロード／ストア命令：常に主記憶アクセス

ベクトルプロセッサについてはさらに、

(11)ベクトルレジスタ数：32

スーパースカラプロセッサについてはさらに、

(12)レジスタ番号リネーム方式

1ループ内演算でアクセス可能なレジスタ数：32を仮定した。

3.2 性能評価ベンチマーク

性能評価ベンチマークとして、図2に示す行列乗算コードを用いた。行列乗算コードは、LINPACK[9]等の各種数値計算で重要になる。

```
DO 10 K=1,N
 DO 20 J=1,N
  DO 30 I=1,N
   C(I,J)=C(I,J)+A(I,K)*B(K,J)
CONTINUE
```

図2：行列乗算コード

評価では、この行列乗算コードの3つのDOループを展開する。展開方法は、主記憶アドレスアクセスパターンが異なる積和型、内積型の2つを考えた。

3.3 性能評価手法

性能評価ベンチマークについて、ハンドコーディングによりオブジェクトコード生成および最適化を行い、これを3.1の前提の元でプログラム実行マシンサイクル数を計算する主記憶シミュレータに入力し、性能を見積もった。

オブジェクトコードの最適化は、スーパースカラ方式およびソフトウェアパイプライニングを用いて行った。以下に最適化手順を示す。

step1：演算単位（計算フェーズ）を決定する。演算単位は、基本的にソースコード中の一つまたは複数の演算式、あるいはループ展開を施した最適化レベルの最内側ループとする。演算単位は演算命令列が必要とするレジスタ数と定義できる。この評価では、演算単位に使用可能なレジスタ数は32本に制限する。演算単位は可能な限り大きくして、浮動小数点演算レイテンシを隠すように個々の演算を行わせる。演算単位を大きくしすぎてレジスタ数が不足する場合には、演算単位を分割する。

step2：1つの計算フェーズの命令実行マシンサイクル数と主記憶アクセスレイテンシの関係から、1つの演算単位のロード命令列を演算命令列に対していくつ前の計算フェーズで実行させるかを決定する。

step3：以上の方法によっても主記憶アクセスレイテンシを隠すことができない場合は、主に演算命令列にデータ待ちを考慮した変更を加える。

4. 性能評価

4.1 評価用ループ展開コード

評価では、まずハンドコーディングによりオブジェクトコードを作成した。オブジェクトコード作成にあたり、1つの計算フェーズ内の演算で使用するレジスタ数が制限を越えないようにループ展開を施した。次に、そのコードからループ実行に要するマシンサイクル数を計算し性能を算出し、その中で最も高い性能が得られた積和型6段2列コードおよび

```
DO 10 K=1,N,6
DO 20 J=1,N,2
DO 30 I=1,N
  C(I,J)   =C(I,J)+A(I,K)*B(K,J)+A(I,K+1)*B(K+1,J)
            +A(I,K+2)*B(K+2,J)+A(I,K+3)*B(K+3,J)
            +A(I,K+4)*B(K+4,J)+A(I,K+5)*B(K+5,J)
  C(I,J+1)=C(I,J+1)+A(I,K)*B(K,J+1)+A(I,K+1)*B(K+1,J+1)
            +A(I,K+2)*B(K+2,J+1)+A(I,K+3)*B(K+3,J+1)
            +A(I,K+4)*B(K+4,J+1)+A(I,K+5)*B(K+5,J+1)
30 CONTINUE
20 CONTINUE
10 CONTINUE
```

図3：積和型6段2列展開コード

```
DO 10 J=1,N,5
DO 20 I=1,N,2
DO 30 K=1,N,2
  S1=S1+B(J,K)*A(K,I)+B(J,K+1)*A(K+1,I)
  S2=S2+B(J+1,K)*A(K,I)+B(J+1,K+1)*A(K+1,I)
  S3=S3+B(J+2,K)*A(K,I)+B(J+2,K+1)*A(K+1,I)
  S4=S4+B(J+3,K)*A(K,I)+B(J+3,K+1)*A(K+1,I)
  S5=S5+B(J+4,K)*A(K,I)+B(J+4,K+1)*A(K+1,I)
  S6=S6+B(J,K)*A(K,I+1)+B(J,K+1)*A(K+1,I+1)
  S7=S7+B(J+1,K)*A(K,I+1)+B(J+1,K+1)*A(K+1,I+1)
  S8=S8+B(J+2,K)*A(K,I+1)+B(J+2,K+1)*A(K+1,I+1)
  S9=S9+B(J+3,K)*A(K,I+1)+B(J+3,K+1)*A(K+1,I+1)
  S10=S10+B(J+4,K)*A(K,I+1)+B(J+4,K+1)*A(K+1,I+1)
30 CONTINUE
C(J,I)=S1;C(J,I+1)=S6
C(J+1,I)=S2;C(J+1,I+1)=S7
C(J+2,I)=S3;C(J+2,I+1)=S8
C(J+3,I)=S4;C(J+3,I+1)=S9
C(J+4,I)=S5;C(J+4,I+1)=S10
20 CONTINUE
10 CONTINUE
```

図４：内積型5段2列2行展開コード

内積型5段2列2行展開コードの2つを評価用ループ展開コードとして用いた。最内側ループのコードを図３および図４に示す。

4.2　主記憶競合の性能に及ぼす影響

(1)積和型6段2列展開コードの評価

スーパースカラ方式による評価コードの最内側ループの繰り返し処理部分を図５に示す。また、ベクトル処理方式の最内側ループのコード例を図６に示す。

図５は、計算フェーズIの時刻Tにおける処理内容を表している。例えば、この計算フェーズIでは、9つ先のループで使用する配列データAおよび配列データCを先読みし(PLD)、１つ前の計算結果を主記憶に書き込んでいる(PST)。最内側ループの処理が終了するとハードウェアポインタ(ICP)を更新して次計算フェーズ(ループカウンタIをインクリメント)に移る。

主記憶アクセスに関係するロード／ストア部分に

T	ロード／ストア	浮動小数点乗算	浮動小数点加算
1	PLD A(I+9,K)→FR	B(K,J)*A(I,K)→FR	
2	PST FR→C(I-1,J)	B(K,J+1)*A(I,K)→FR	
3	PLD A(I+9,K+1)→FR	B(K+1,J)*A(I,K+1)→FR	C(I,J)+FR→FR
4	PST FR→C(I-1,J+1)	B(K+1,J+1)*A(I,K+1)→FR	C(I,J+1)+FR→FR
5	PLD A(I+9,K+2)→FR	B(K+2,J)*A(I,K+2)→FR	C(I,J)+FR→FR
6	PLD A(I+9,K+3)→FR	B(K+2,J+1)*A(I,K+2)→FR	C(I,J+1)+FR→FR
7	PLD A(I+9,K+4)→FR	B(K+3,J)*A(I,K+3)→FR	C(I,J)+FR→FR
8	PLD A(I+9,K+5)→FR	B(K+3,J+1)*A(I,K+3)→FR	C(I,J+1)+FR→FR
9	PLD C(I+9,J)→FR	B(K+4,J)*A(I,K+4)→FR	C(I,J)+FR→FR
10	PLD C(I+9,J+1)→FR	B(K+4,J+1)*A(I,K+4)→FR	C(I,J+1)+FR→FR
11		B(K+5,J)*A(I,K+5)→FR	C(I,J)+FR→FR
12		B(K+5,J+1)*A(I,K+5)→FR	C(I,J+1)+FR→FR
13			C(I,J)+FR→FR
14			C(I,J+1)+FR→FR
15	ICP		

図５　最内側ループ繰返し処理部（積和型6段2列展開コード）

```
VLD C(I,J)
VLD A(I,K)
VMAD
VLD A(I,K+1)
VMAD
VLD A(I,K+2)
VMAD
VLD A(I,K+3)
VMAD
VLD A(I,K+4)
VMAD
VLD A(I,K+5)
VMAD
VSTD  C(I,J)
VLD C(I,J+1)
VMAD
VMAD
VMAD
VMAD
VMAD
VMAD
VSTD  C(I,J+1)
```

図６：ベクトル処理用コード

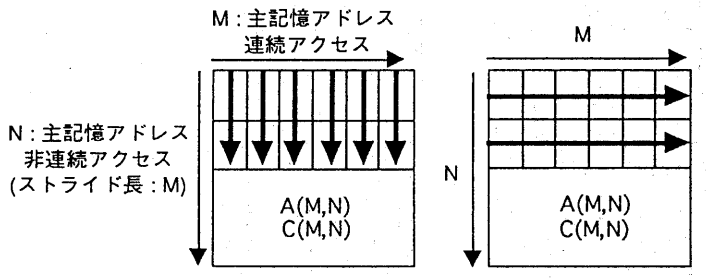

図７：配列データアクセスパターン

着目すると、繰り返し処理部分では図7(a)に示すように、配列Aのロード命令は常に主記憶上の配列データをアドレス非連続（ストライド長M）に参照し、同様に配列Cのロード／ストア命令も主記憶上の配列データをストライド長Mで参照していることがわかる。ストライド長Mを変更するには、配列サイズを変更すればよい。そこで、配列サイズを変更し、主記憶アドレスアクセスパターンが性能に及ぼす影響を評価した。

図9にスーパースカラ方式において、配列サイズを変更して主記憶アドレスアクセスパターンを変化させた場合の相対性能を示す。最良ケースの性能を1とした。

図9から、配列サイズが16の倍数の時に性能が大きく低下することがわかる。これは、主記憶を16バンクインターリーブ構成であると仮定していることに起因する。配列サイズが16の倍数の場合、ストライド長Mが16の倍数になり、図5に示すループカウンタIの繰り返し処理部分において、配列Aの6個のロード命令はバンク0、バンク0、バンク0、のように常に主記憶の同バンクに格納されているデータを参照する。そのため、バンク競合が多発する。配列Cについても同様である。しかしながら、異なる複数の配列データ（A、C）参照をループ毎に繰り返すため、ループ単位で参照するバンクが変わり、図10(a)に示すように主記憶アドレス参照がある程度ばらつき、性能低下は20%程度である。

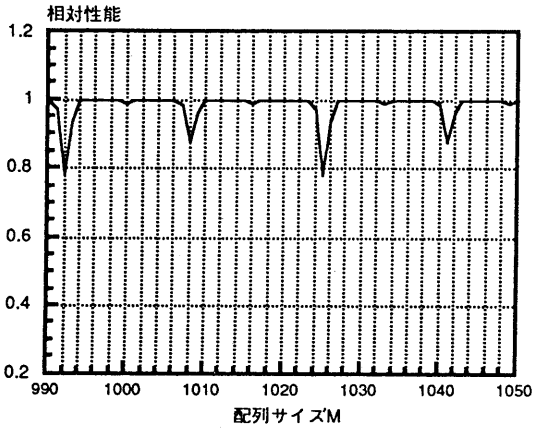

図9：スーパースカラにおける配列サイズと性能の関係
（積和型6段2列展開コード）

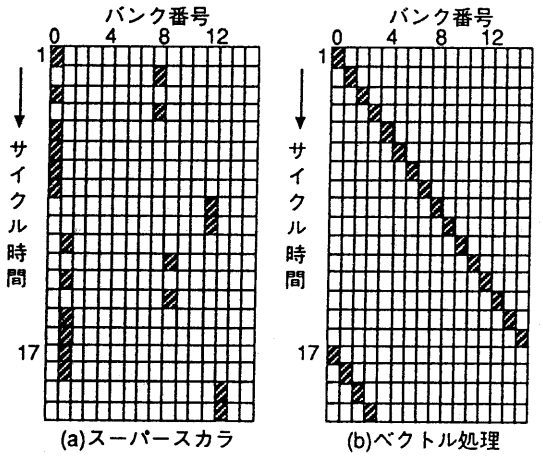

図10　ストライド長M=16の場合の
配列データアクセスパターン

一方、ベクトル処理の最内側ループはIをループ制御変数として実行される。この場合、VLD命令で読み出される配列C(I,J)および配列A(I,K)はどちらも図7(b)のように主記憶アドレス連続参照になる。従って、図10(b)に示すようにバンク競合は発生せず、配列サイズに関係なく高い性能を得られる。

(2)内積型5段2列2行展開コードの評価

(1)と同様に配列サイズを変更して主記憶アドレスアクセスパターンを変化させた場合の相対性能を示す。最良ケースの性能を1とした。図11にスーパースカラの評価結果、図12にベクトル処理の評価結果を示す。

評価結果は以下の通り。

(a)スーパースカラでは、配列サイズが16の倍数の時に性能が大きく低下する。

(b)ベクトル処理では、2の冪乗で性能が著しく低下する。

(c)ベクトル処理では、配列サイズが奇数の場合ピーク性能を得られるが、スーパースカラでは配列サイズが16の倍数近辺の奇数配列では性能があまり向上しない。

(d)ベクトル処理では、配列サイズが2の冪乗である場合性能がピークの10%～60%まで低下するが、スーパースカラでは、配列サイズが2の冪乗であってもピークと同等の性能が得られ、16の倍数であってもピークの半分の性能が得られる。

次に評価結果について考察する。(a)..(d)は評価結果に対応する。

(a)配列サイズが16の倍数の時に性能が低下するのは、主記憶が16バンクインターリーブ構成であり、配列A,Bのロード／ストアが常に主記憶上の同じバンク（例えば、バンク0）を参照するためである。この傾向は(1)のケースと同様である。

(b)積和型コード（図3）では主記憶上の配列データを連続参照する。しかしながら、内積型コード

（図４）では、配列A、Bとも非連続（ストライド長M）参照になる。ベクトル処理では、各配列が最内側ループKのループ回数分のデータをストライド長Mで主記憶から読み出す（例えば、配列サイズが16の倍数の場合、バンク0を連続してK/2回参照する）ため、配列サイズが2の冪乗である時、バンク競合が多発し性能が著しく低下する。

(c)ベクトル処理では、配列サイズ（ストライド長M）が奇数である場合、図13(b)に示すようにバンク競合が発生しないため、高い性能を得られる。一方、スーパースカラでは、１つの計算フェーズにおいて複数の配列が交互にロード／ストアされ、図13(a)のように異なる配列の要素が互いに悪影響を及ぼしバンク競合が発生し、ベクトル処理のように高い性能を得られない。

(d)ベクトル処理では、同じ配列を連続して読み出すため、配列サイズが2の冪乗である時バンク競合が多発し性能が著しく低下する。一方、スーパースカラでは、同じ配列を連続して読み出すのではなく、複数の配列を交互に読み出すため、バンク競合が緩和され、配列サイズが2の冪乗であってもベクトル処理ほど性能は低下しない。

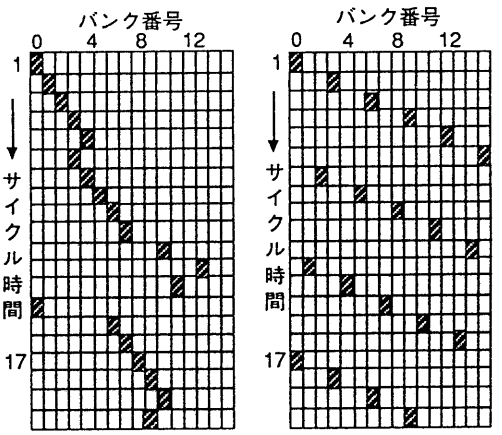

図13　ストライド長M=3の場合の配列データアクセスパターン

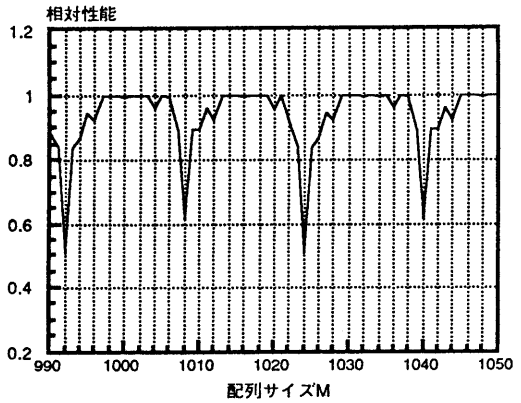

図11：スーパースカラにおける配列サイズと性能の関係
（内積型5段2列2行展開コード）

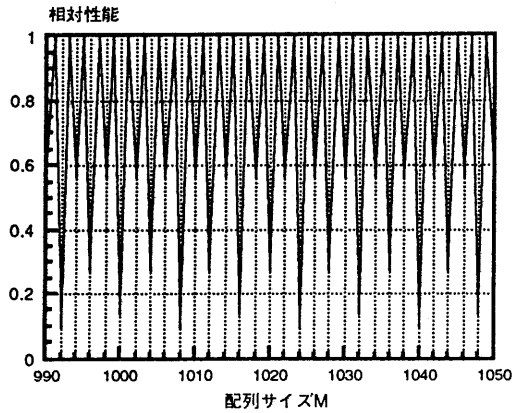

図12：ベクトル処理における配列サイズと性能の関係
（内積型5段2列2行展開コード）

4.3　ベクトル処理とスーパスカラの比較

(1)ベクトル処理のスーパースカラに対する優位点

・ベクトル処理では、ベクトル命令で主記憶からロードあるいはストアされる配列データがアドレス連続参照である場合、バンク競合が発生せず、主記憶アクセス性能は低下しない。一方、スーパースカラでは、主記憶上の複数配列データを交互に参照するため、配列サイズが主記憶バンク数の倍数であるときに性能が急激に低下する。

・ベクトル処理では、配列サイズが奇数であれば主記憶アクセス性能は低下しないが、スーパースカラでは、複数配列が主記憶上の配列データを交互にアクセスするため、それらのアクセスが互いに悪影響を及ぼし性能があまり向上しない。

(2)スーパースカラのベクトル処理に対する優位点

・スーパースカラでは、ループ展開アルゴリズムの差異によらず主記憶アクセスは非連続であり、主記憶アクセス性能は同じ傾向を示すため、アルゴリズムの変更に強く汎用的である。

一方、ベクトル処理では、主記憶を連続アクセスする場合は高性能であるが、非連続アクセスする場合、配列サイズが2の冪乗であるとバンク競合が多発し、性能が著しく低下する。

・主記憶を非連続にアクセスするスーパースカラの主記憶アクセス性能が著しく低下するのは、配列サイズがバンク数の倍数である場合のみである。

ベクトル処理における性能低下アクセスパターン（2の冪乗）であっても性能はあまり低下しない。また、ベクトル処理が各配列を連続して読み出すのに対して、スーパースカラでは複数配列を交互に読み出すため、バンク競合が緩和され主記憶アクセス性能のばらつきは小さい。

5. まとめ

本稿では、大規模科学技術計算において、演算器を効率よく使用するためにループ展開等の最適化を施した場合、スーパースカラ型プロセッサとベクトルプロセッサの処理方式の差異により主記憶上の配列データのアクセスパターンが変わることを示した。さらに、演算器構成および主記憶構成が同じであると仮定して、これら2つの処理方式の主記憶アクセス性能を評価することにより、スーパースカラとベクトル処理の特性の差異を明確にした。評価にあたり、主記憶アクセスがパイプライン化されたスーパースカラ型プロセッサを想定し、大規模科学技術計算ではキャッシュの効果が小さいこと、キャッシュを使用しないベクトルプロセッサとの比較を行うことから、想定したプロセッサではデータの読み出し／書き込みにはキャッシュを使用しないと仮定した。また、主記憶のバンク数を16とし、評価用ベンチマークとして各種数値計算で重要な行列積演算の積和型および内積型コードについてループ展開を施した。

評価結果から、表1に示すように、演算器構成、主記憶構成が同一であると仮定した場合、積和型コードでは、ベクトル処理は配列サイズによらず主記憶アドレスを連続参照するため非常に高い主記憶アクセス性能がえられ、スーパースカラに比べて相当優位である。

一方、スーパースカラは、ベクトル処理に対して欠点はあるものの、積和型／内積型のどちらのアルゴリズムに対しても主記憶アドレスを非連続に参照し、同様な主記憶アクセス性能が得られること、また、アドレスを非連続参照するケースにおいて、配列サイズをベクトル処理程意識しなくとも安定した主記憶アクセス性能が得られることから汎用的であるといえる。

今後は、将来的にハードウェアコストや命令並列度を考慮してスーパースカラ型プロセッサの有効性を検討する。

表1　ベクトル処理方式とスーパースカラ方式比較

処理方式	アルゴリズム			
	積和型		内積型	
ベクトル	アドレス連続	◎	アドレス非連続	×
スーパースカラ	アドレス非連続	○	アドレス非連続	○

謝辞

本研究を進めるにあたり有益な助言および協力を頂きました日立製作所ソフトウェア開発本部　後保範主任技師、同中央研究所　マシエルフレデリコ博士に深く感謝いたします。

参考文献

[1] M. Lam：Software Pipelining: An Effective Scheduling Technique for VLIW Machines：Proceedings of the SIGPLAN'88 (June, 1988)

[2] B. R. Rau, M. Lee, P. P. Tirumalai, and M. S. Schlansker：Register Allocation for Software Pipelined Loops：Proceedings of the SIGPLAN'92 (June, 1992)

[3] P. Tirumalai, M. Lee, and M. S. Schlansker：Parallelization of Loops with Exits on Pipelined Architectures：Proceedings of Supercomputing'90 (Nov., 1990)

[4] 長島, 田中（義）：スーパーコンピュータ：オーム社（平成4年11月）

[5] B. R. Rau, D. W. L. Yen, and R. A. Towle：The Cydra 5 Departmental Supercomputer：IEEE Conputer, vol. 22, no. 1 (Jan., 1989)

[6] 中村, 位守, 伊藤, 中澤：レジスタウインドウとスーパースカラ方式による擬似ベクトルプロセッサの提案：JSPP'92予稿集（平成4年6月）

[7] 位守, 中村, 朴, 中澤：スライドウインドウ方式による擬似ベクトルプロセッサ：情報処理学会 論文誌 Vol. 34, No. 12（平成5年12月）

[8] 廣野, 上野, 中村, 朴, 中澤：マルチバンクメモリ上における擬似ベクトルプロセッサPVP-SWの性能評価：情処研報 Vol. 95, No. 29（平成7年3月）

[9] Dongarra J. J.：Performance of Various Computers Using Standard Linear Equations software：Comp. Science Dept., Univ. of Tennessee (Dec., 1994)