計算機アーキテクチャ 86－5
（1991 1．24）

記号処理における命令レベル並列性の評価

丸山 勉
日本電気(株) C&Cシステム研究所

近年マイクロプロセッサの高速化技術として、命令レベル並列処理が注目を集めている。記号処理においては、メモリアクセス命令の頻度が約50%程度を占めるため、より多くの命令を並列に実行することによって処理の高速化を行なうためには、キャッシュメモリのマルチポート化、キャッシュアクセス遅延の低減化等が不可欠であるが、このようなハードウェアの複雑化は実現可能なクロックスピードを低下させる可能性がある。定理証明プログラムBOYER等を用いたシミュレーション評価によれば、演算器数を2個程度に限定し、キャッシュアクセス遅延を大きくすることによって高速なクロックスピードを実現する方法においても、命令レベル並列性を重視して処理の高速化を狙う方法と同程度の性能が実現できることがわかった。ハードウェアの簡単さを考えると、クロックスピードを優先する方法の方が有利であると考えられる。この場合には、キャッシュのマルチポート化等は不要であり、実現される命令レベル並列処理は、演算器2個、キャッシュアクセス遅延2の場合で約1.3~1.4である。

Evaluation of Instruction Level Parallelism in Symbolic Processing

Tsutomu Maruyama

C&C Systems Research Lab., NEC Corporation

1-1, Miyazaki 4-Chome Miyamae-ku Kawasaki Kanagawa 213 JAPAN

In order to make full use of instruction level parallelism in symbolic processing, it is necessary to realize multi-port cache memory with low access delay, because memory access instructions amount to 50% in symbolic processing. However, such complex hardware may decrease clock rate. There are two ways in order to attain high performance as follows. One is to make full use of instruction level parallelism with multi-port cache memory with low access delay. The other is to realize faster clock rate with fewer processing units by allowing more cache access delay slots and making pipeline deeper. The simulation results show that both approaches achieve almost same performance. However, the second approach is more advantageous because it requires less complex hardware. The instruction level parallelism with 2 processing units and two cache access delay slots is about 1.3 ~1.4.

1 はじめに

最近のデバイス技術の進歩によって、複数の演算器を1チップ上に実装することが可能となり、マイクロプロセッサの高速化技術として命令レベル並列処理が注目を集めている。

命令レベルの並列処理において、より多くの命令を並列に実行するためには、より多くの演算器を並置することが望ましいが、多くの演算器を並置すると、ハードウェアおよびデータパスの複雑化によるクロックスピードの低下を引き起こし、かえって性能の低下を招くことになりかねない。

本稿では、記号処理向き言語において命令レベル並列性を最も有効に活用し、より高い処理性能を実現するためには、どのようなハードウェア構成が最も適しているか評価する。評価においてはタグ分岐命令を持つRISC型の命令セットを仮定し、定理証明プログラムBOYER等を用いて、スーパースカラー方式によってシミュレーションを行なった。

第2章では、どのような命令セットが記号処理に適しているかについて述べる。第3章では、第2章で述べた命令セットを用いた場合のLISPのベンチマークプログラムの命令実行頻度を示す。第4章では、記号処理において実行頻度の高いメモリアクセス命令と分岐命令の並列処理について検討する。第5章では、今回行なったシミュレーションの方式及び評価項目について述べる。第6章では、シミュレーション結果について述べる。

2 記号処理向き命令セット

従来、記号処理にはマイクロプログラムマシンが用いられてきたが、近年のハードウェア技術の進歩により、記号処理においてもRISCアーキテクチャの方が優位になってきている[1][2]。以下にその理由を述べる。

従来の記号処理向きプロセッサは、以下に示す機能を実現することによって汎用マイクロプロセッサに対して、より高速な処理を実現したきた。

- 大容量かつ高速なメモリシステム
- マイクロプログラムによる命令レベル並列処理
- タグ分岐

しかし、RISCプロセッサの出現により、汎用マイクロプロセッサにおいても、大容量かつ高速なキャッシュメモリシステムが用いられるようになり、また近年汎用マイクロプロセッサの高速化技術として、命令レベルの並列処理が注目を集めており、近い将来においては多くの汎用マイクロプロセッサが命令レベルの並列処理を行なうものと考えられる。

マイクロプログラムマシンの特徴は、マイクロ命令を用いることにあるが、一般に、マイクロ命令の各サブ命令はRISC命令と類似している。従って、RISCプロセッサにおいて命令レベル並列処理が導入された場合、同時に実行される幾つかの命令をひとまとめとしてみると、これはマイクロ命令とほぼ同じものであると考えることができる。即ち、命令レベル並列処理を行なうRISCプロセッサは、LISP等の言語を直接マイクロプログラムにコンパイルするコンパイラをサポートしたマイクロプログラムマシンとほぼ等価であるといえる。

このように考えると、命令レベル並列を行なうRISCプロセッサの方が以下の点で有利である。

- ハードウェアが小規模
  中間コードの命令フェッチ、デコードが不要
- パイプラインの乱れが少ない
  LISP,PROLOGにおいて、1中間コードあたり実行される命令数は平均数個であり、マイクロプログラムマシンでは、数命令に一回中間コードのフェッチのためにパイプラインが乱れる可能性がある。
- グローバルな最適化が可能
  中間コード間に跨る最適化が可能

しばしば、RISCプロセッサを用いた場合にはコードサイズの増加が指摘されるが、それは全てのコードをインライン展開した場合であり、ある程度以上の長さを持ち頻繁に実行される命令列は関数呼び出しにより実現することによって、コードサイズはそれほど大きくならない。関数呼び出しの手間も、グローバルなレジスタ割り当てなどの最適化をコンパイル時に行なえば十分小さくすることができる。

3 命令実行頻度

RISC命令セットを用いた場合の定理証明プログラムBOYERと、QUICKSORTプログラム(リスト長50)における命令実行頻度を表1に示す。命令セットとしては、MIPS[4]の命令セットにタグ分岐命令として、下位2ビットの即値比較による分岐命令を付け加えたものを用いた。これらの例題においては、下位2ビットの即値比較による分岐命令で、タグ分岐命令としては十分である(下位2ビットに限定したことによって、処理速度の低下は見られない)。下位2ビットによるタグ分岐の例を図1に示す。

評価に用いたコードは、基本的にはLISPマシンLIME[3]のコードを用いているが分岐命令の実行回数等が最小になるように、最大限のオプティマイズを行なった。比較のために示したDHRYSTONE(Cプログラム)の実行結果では、MIPSのオブジェクトを逆アセンブルしたものを用いた。

表1からわかるように、メモリアクセス命令と分岐命令の頻度が高い。

```
andi Rtemp, Rsrc, 0x3
addi Rtemp, Rtemp,-CONS
beq Rtemp, Rzero, _Label
        ↓
bteq Rsrc, CONS, _Label
```

図1: タグ分岐命令

表 1: 命令実行頻度

	boyer	qsort	dhrystone
メモリアクセス	0.491	0.519	0.369
Read	0.298	0.259	0.254
Write	0.193	0.259	0.115
分岐	0.327	0.213	0.252
条件分岐	0.088	0.069	0.209
タグ分岐	0.125	0.063	0.000
無条件分岐	0.026	0.006	0.006
関数 call/ret	0.088	0.074	0.036
ALU 演算	0.181	0.268	0.379
総実行命令数	10339048	8796	488(1loop)

4 記号処理における命令レベル並列処理

前章で述べたように、記号処理においてはメモリアクセス命令と分岐命令の頻度が高い。従って、最大限の並列性を実現するためには、これらの命令をできるだけ並列に実行することが望ましい。本章では、より高い命令レベル並列処理を実現するための方式とその問題点について検討する。

4.1 メモリアクセス命令

メモリアクセス命令は、全命令の約 50% を占める。このようにメモリアクセス頻度が高い場合には、より高い命令レベル並列処理を実現するためには、複数語の同時読み出し / 書き込み及びキャッシュメモリアクセス遅延時間の低減を実現することが必要である。

4.1.1 複数語の同時読み出し / 書き込み

メモリアクセス命令の実行頻度は約 50% であり、命令レベルの並列性を有効に活かすためには、演算器の約半数においてメモリアクセスが可能であることが望ましい。従って、4 つの演算器を配置し効率的に稼働させるためには、データキャッシュメモリを 2 ポート化する必要があることになる。しかし、キャッシュメモリのマルチポート化は、以下のような問題をもたらす。

- チップ上に搭載可能なキャッシュサイズの低下
- メモリアクセス時間の増大

例題 BOYER では、約 1KB のスタック領域と約 2MB のヒープ領域が用いられる。従って、キャッシュサイズの低下は総合性能に大きな影響を与える危険性があり、キャッシュをマルチポート化したことによる処理の高速化とキャッシュサイズの減少によるキャッシュ更新の手間とのトレードオフとして考えることが必要である。

4.1.2 メモリアクセスの遅延

また、このようにメモリアクセス頻度が高いと、より多くの命令を同時に実行するためにはメモリアクセスの遅延時間をできるだけ小さくすることが重要となる。

しかし、キャッシュメモリアクセスは、最も重い処理のひとつであり、整数演算等に較べてより多くの処理が必要である。このため、現在の多くの汎用マイクロプロセッサにおいても、整数演算等の実行結果は、次命令で直ちに参照可能であるのに対して、キャッシュアクセスの結果は、次命令では参照することができずに、1 サイクル待たされる。

以下、キャッシュメモリのアクセス遅延とクロックスピードの高速化についての簡単な検討を行なう。キャッシュアクセスには、おおよそ以下の手順が必要である。

1. アドレスの供給及びアドレスデコード
2. TLB の読み出し
3. タグメモリの読み出し,
   TLB ヒット / ミスヒットの判定
4. データメモリの読み出し / 書き込み、
   タグメモリヒット / ミスヒットの判定
5. 読み出した値の選択及びフォーワーディング

これらの処理の幾つかはオーバーラップ可能であり (どの処理がオーバーラップ可能であるかは、キャッシュの構成によって異なる)、ダイレクトマップのキャッシュの場合にはデータメモリの読み出し結果の選択が不要である。

説明を簡単にするために論理アドレスキャッシュを仮定し (論理アドレスキャッシュの場合には、TLB のサイクルが不要となる)、各処理の処理時間は同一であるものとし、更に各処理はオーバーラップされないものとして非常に単純なキャッシュアクセスパイプラインを考えることにする。

キャッシュメモリをアクセス遅延 0 で読み出すためには、1 サイクル内で、アドレスのデコード、タグメモリの読み出し、データメモリアクセスの 3 つの処理を行うことが必要である。1 遅延スロットの場合には、アドレスデコードとタグメモリ読み出し、またはタグメモリ読み出しとデータメモリアクセスのいずれか 2 つの処理を行なえばよい。また、2 遅延スロットが許される場合には、アドレスデコード、タグメモリ読み出し、データメモリアクセスをそれぞれ 1 サイクルで行なえばよいことになる。

従って、メモリアクセス遅延が 0,1,2 の場合に実現可能なクロックスピード比は、1 対 1.5 対 3 となる。実際には、各処理がオーバーラップ可能であり、各処理の処理時間も一律ではないので、これほど単純ではないが、この数値は、メモリアクセス遅延を 0,1,2 とした場合に実現されうるクロックスピード比の上限と考えることができる。

このように、メモリアクセス遅延時間の減少による命令レベル並列性の向上と、クロックスピードの高速化は相反する要因であるため、どちらを優先するかは非常に重要な選択となる。

4.2 分岐命令

分岐命令は全命令数の約 1/4~1/3 を占め、命令レベル並列処理を考えるにあたっては重要な要素である。命令レベルの並列処理において分岐命令により処

理速度の低下を防ぐためには、以下のような方法が考えられる。

- 分岐によるパイプラインの乱れの防止
  - 分岐遅延なしの分岐
  - squashing 分岐
- 複数の分岐の同時実行

4.2.1 分岐によるパイプラインの乱れの防止

現在のマイクロプロセッサでは、分岐が生じたことによる命令フェッチのオーバーヘッドを隠すために遅延分岐方式が用いられている。しかし、命令レベル並列処理を行なう場合には、遅延分岐方式では不十分である。これは、命令レベルの並列処理を行なう場合には、同時に複数の命令を実行するため、実質的な遅延スロットの数が増え、遅延スロットを埋めることが非常に難しくなるからである(現状でも2つの遅延スロットを埋めることは一般に困難である)。

(1) 分岐遅延なしの分岐命令

分岐命令が3~4命令に1命令ある場合に、命令レベル並列処理を行なうと半数以上のサイクルにおいて分岐命令が実行されることになる。従って、分岐遅延0の分岐命令を実現することはより実行ステップ数の削減には非常に大きな影響を与える。

しかし、分岐遅延0の分岐命令を実現するためには、命令読み出しのサイクル内において、

- 命令読み出し
- 命令のデコード(分岐部分のみ)
- 条件判定

を行なわねばならず、クロックスピードを低下させる要因となる。

(2) squashing 分岐

squashing 分岐と呼ばれる手法もパイプラインの乱れを防止するのに有効である。squashing 分岐では、分岐命令より前の命令で遅延スロットを埋めるのではなく、分岐予測を行ない分岐するであろうと予測される分岐先の命令で遅延スロットを予め埋める。従って、分岐予測が正しかった場合には、分岐のオーバーヘッドを隠すことができる。分岐予測が外れた場合には、遅延スロットの命令の実行のキャンセルおよび、命令フェッチのやり直しが必要となる。

分岐予測が当たる確率が性能に大きな影響を与えるが、一般には静的な予測のみによってもある程度の予測が可能である(BOYER等では、静的な予測によって約75%の的中率が可能)。

また、squashing 分岐を命令レベル並列処理に導入するにあたって、分岐予測が外れた場合の遅延スロットのキャンセルのみではなく、同一スロット内の多の命令もキャンセルすることによってより高い並列性を実現することができる。

4.2.2 複数の分岐命令の同時実行

BOYERでは、分岐命令の実行頻度が全命令数の約1/3を占めるが、このような状況では1サイクルにおいて実行可能な分岐命令数が1であれば、実現できる最大命令レベル並列性は3であり、それ以上は期待できないことになる。

これに対して、同一サイクルにおいて複数の分岐命令が実行できれば、より大きな命令レベル並列性を実現することが期待できる。しかし、同一サイクルにおいて複数の分岐命令を実行するためには、

- 複数の分岐先アドレスの計算
- 分岐命令間での優先順位の制御

等の処理が必要となり、クロックスピードを低下させる危険があるため、実際にどの程度の効果が得られるのかを評価する必要がある。

5 シミュレーション方式

5.1 評価項目

前章で行った検討をもとに、以下の項目について評価を行った。

- 命令レベル並列性
- クロックスピードを考慮した場合の性能比
- キャッシュメモリサイズの影響
- タグ分岐命令の効果
- squashing 分岐の効果
- 複数の分岐命令の同時実行の効果

また、上記の評価において、演算器数、データキャッシュのアクセス遅延及び分岐遅延、データキャッシュのアクセスポート数を変化させてその影響を調べた。

演算器数については1~4まで変化させた。これは、演算器を4個以上にしてもほとんど性能の向上が見られないからである。

データキャッシュのアクセス遅延については、0,1,2の場合についてシミュレーションを行なった。分岐遅延については、データキャッシュアクセスと同じ遅延スロット数を仮定した。これは、両者において行なわれる処理がほぼ同様のものだからである。

データアクセスポート数は、メモリアクセス頻度が約50%であることから、演算器数を1~4と変化せたことに対応して1~2として評価を行った。

5.2 実行コードの最適化

評価を行うに当たっては、実行時における分岐命令間の命令の実行順序の最適化のみではなく、以下に示したようなソフトウェア最適化を実行前に行っている。

(1) レジスタ番号の付けかえ

レジスタ番号をできる限り重複しないように付けかえることによって、命令間での不必要な待ちがなくなり最大限の並列性を引き出すことが可能となる。

(2) 分岐命令を越えた命令実行順の変更

分岐命令後のある命令の実行結果が、その分岐命令の他方向のパスにおいて参照されていない場合には、その命令を分岐命令より先に実行することが可能である。このような分岐命令を越えた命令実行順の変更を

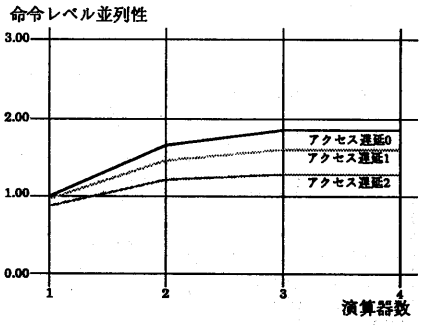

図 2: 命令レベル並列性 (BOYER, キャッシュアクセス 1 ポート)

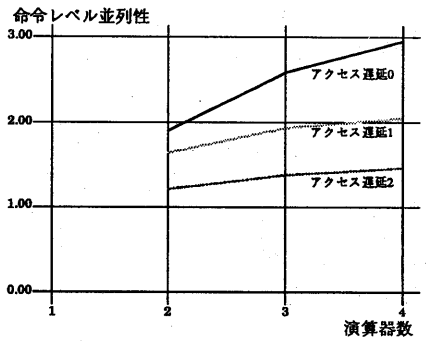

図 3: 命令レベル並列性 (BOYER, キャッシュアクセス 2 ポート)

動的に行なうことは困難であるが、実行前に演算器数を考慮にいれて、分岐命令の分岐先予測を静的に行いながらステップ数が最小になるように命令順の入れ替えを行うことによって実現できる。このような最適化により分岐予測が的中した場合の処理は、より高速となる。

(3) メモリアクセス命令の実行順の変更

一般に、メモリアクセス命令の順番を変更するためには、それらの命令の間でアドレスが衝突しないことを保証する必要があるが、LISP,PROLOG におけるスタックアクセスとヒープアクセス等の間ではアドレスの衝突が生じないこと自明であるので、これらのメモリアクセス命令の実行順は容易に入れ替えることができる。

6 シミュレーション結果

6.1 命令レベル並列性

図 2,3に BOYER における命令レベル並列性を示す。図 2はキャッシュメモリが 1 ポートである場合の命令レベル並列性であり、図 3は 2 ポートである場合のものである。比較のために DHRYSTONE における命令レベル並列性を図 4,5に示す。BOYER の場合と同様に、図 4は 1 ポートの場合、図 5は 2 ポートのある場合である

図 2からわかるように、キャッシュアクセスポート数が 1 の場合には、演算器数を 2 以上に増やしても、キャッシュアクセス遅延時間にかかわらず、命令レベル並列性はあまり向上しない。演算器数の増加によるクロックスピードの低下を考えればこの程度の性能向上は容易に打ち消されてしまうと考えられる。従って、メモリアクセスポート数が 1 の場合には演算器は 2 個で十分である。これは、メモリアクセス頻度が約 50% であることを考えれば当然の結果である。

キャッシュアクセスが 2 ポートの場合には、キャッシュアクセス遅延が 1,2 サイクルならば、演算器数を 3 以上に増やしてもあまり性能の向上は見られない。キャッシュアクセス遅延が 0 の場合のみ、演算器数が 4 の場合まで性能の向上が見られる。

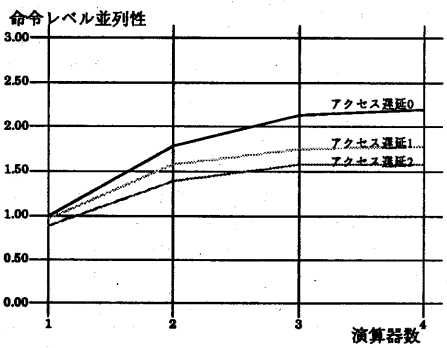

図 4: 命令レベル並列性 (DHRYSTONE, キャッシュアクセス 1 ポート)

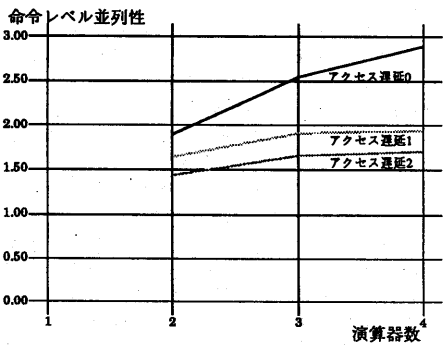

図 5: 命令レベル並列性 (DHRYSTONE, キャッシュアクセス 2 ポート)

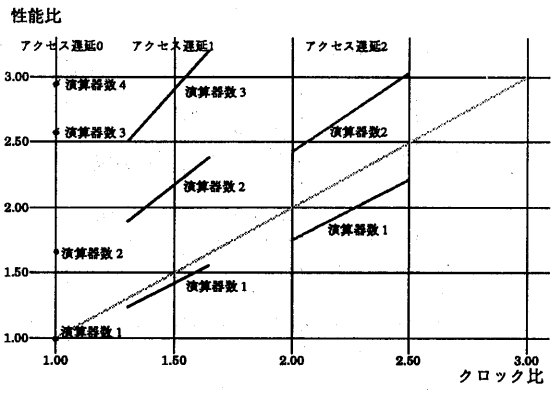

図 6: 相対性能 (BOYER)

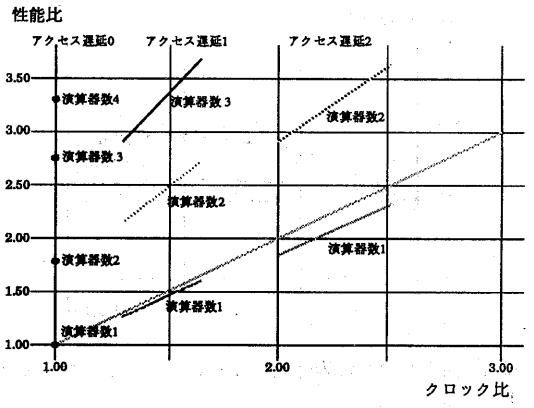

図 7: 相対性能 (QUICKSORT)

図4, 図5に示したDHRYSTONEの場合も、ほぼ同様の傾向が見られるが、キャッシュアクセスが1ポートの場合に(図4)演算器数が3の場合まで性能の向上が見られることが異なる。これはつHRYSTONEの場合には、メモリアクセス頻度は約37%であることによるものと考えられる。

6.2 クロックスピードの影響

キャッシュアクセス遅延が1,2の場合にアクセス遅延が0の場合に対してどれだけ高速なクロックスピードを実現できるかはデバイス技術によってきまるが、4章での検討をもとに、キャッシュアクセス遅延1の場合には1.25~1.6、2の場合には2~2.5倍程度の高速化が可能としたときのBOYERにおける性能の相対値を図6に示す(アクセス遅延0, 演算器数1の場合の性能を1とする)。また、例題QUICKSORTにける同様の性能比を図7に示す。

前節の結果より、アクセス遅延が1の場合には演算数を3以上にしても実現可能な命令レベル並列性は増えず、また、アクセス遅延が2の場合には、演算器数を2以上にしても命令レベル並列性はそれほど増えないため、図6, 図7には、アクセス遅延が0のときには演算器数が4の場合まで、アクセス遅延が1のときには演算器数が3の場合まで、アクセス遅延が2のときには演算器数が2の場合までの結果を示した。

演算器数が3以上の場合には、キャッシュアクセスが2ポートの場合の結果を用いたが、これは前節で述べたようにキャッシュアクセスが1ポートの場合には、演算器数を3以上にしてもそれほどの命令レベル並列性の向上がみられないからである。

図6, 図7からわかるように、アクセス遅延が0の場合には演算器数4、アクセス遅延が1の場合には演算器数3、アクセス遅延が2の場合には演算器数2のときにほぼ同等の性能が実現されていることがわかる。

比較のために、DHRYSTONEについて性能の相対値を図8に示す。DHRYSTONEの場合には、クロックスピードを重視した方がより高い性能が得られることがわかる。これは、DHRYSTONEではBOYERの場合ほどメモリアクセス命令, 分岐命令の実行頻度が

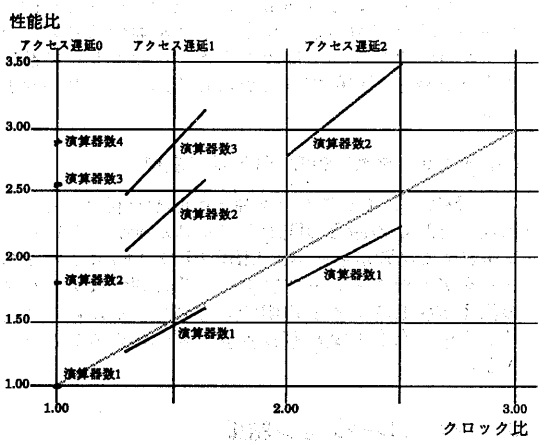

図 8: 相対性能 (DHRYSTONE)

高くないため、キャッシュアクセス遅延, 分岐遅延を減少させたことによる速度向上率がそれほど大きくないためである。

従って、プロセッサの汎用性を考えて、C言語等の処理においても高速な処理速度を実現するためには、演算器数を2個程度に抑え、キャッシュアクセス遅延を2スロットと大きくし、クロックスピードの高速化を狙う手法の方が優れているといえる。

6.3 キャッシュサイズの影響

キャッシュアクセス遅延が0,1の場合には、命令レベルの並列性を利用して性能を向上させるためにはキャッシュのマルチポート化が必要である。キャッシュアクセス遅延0の場合に、キャッシュサイズによってどのように性能が変化するかを図9に示す。図9においては、キャッシュが100%ヒットの場合の性能を1とした。図9に示した結果は、2セット, ブロックサイズ8ワードの場合のものであり、ブロックの更新に

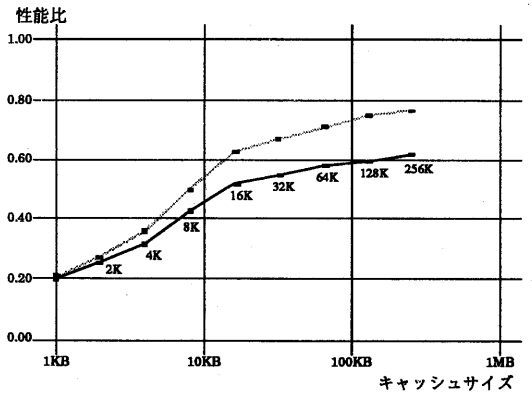

図 9: 相対性能 (BOYER)

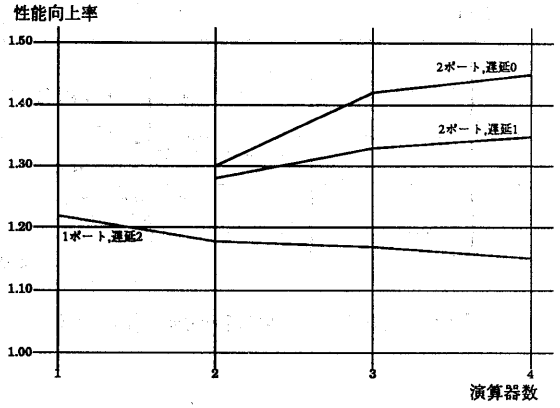

図 10: タグ分岐命令の効果 (BOYER)

は、プロセッサのクロックスピードの 20 倍がかかるものとした (キャッシュアクセス遅延 0 にするためには、プロセッサ内のクロックスピードをそれ程高速化することはできない)。図 9中の破線は、LISP 等ではスタック等のアクセスへの書き込みにおいては、常に破壊的に書き込みを行なうことを利用して、スタック、ヒープトップへの書き込み時にキャッシュがミスヒットした場合に、メインメモリからのブロックリードを行なわない場合のものである。

実現可能なキャッシュサイズが 16KB 以下の場合には、キャッシュサイズの低下により処理速度の低下が著しく、キャッシュアクセスを 2 ポート化する高速化手法は有効ではないことがわかる。

6.4 タグ分岐

タグ分岐命令の効果を表 10に示す。図 10からわかるようにキャッシュメモリが 1 ポートである場合には、演算器数を増やすに従って、タグ分岐の効果が減少しているのに対し、キャッシュメモリが 2 ポートである場合には演算器数を増やすに従ってタグ分岐命令の効果は大きくなる。特に、キャッシュアクセス遅延が 0 の場合には、最大 45% もの効果がみられる。

これは以下の理由によるものであると考えられる。タグ分岐命令がない場合には、タグ部を切り出すために 2 命令が必要であるが、この 2 命令は他の命令とは全く独立であるため、分岐命令を越えて演算器に空きがある場合に自由に先行して実行することが可能である。キャッシュメモリが 1 ポートの場合には、実現される命令レベル並列性が比較的低く、演算器に空きがあることが多い。このため、タグ部を切り出す命令を分岐命令前に実行することが可能となる。演算器数を増やすに従って、演算器の空きが増えるため、タグ分岐命令の効果が小さくなる。

これに対して、キャッシュメモリが 2 ポートである場合には、実現される命令レベル並列性が比較的高いため、演算器の空きができにくく、タグ部を切り出す 2 命令は並列に実行できないため、そのの影響が大きく現れるからである。

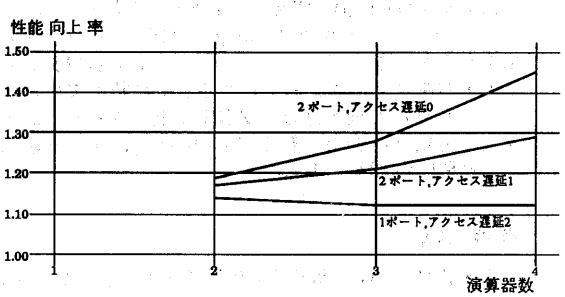

図 11: squashing 分岐の効果 (BOYER)

6.5 squashing 分岐

同一サイクル中の他の命令の実行を抑止する squashing 分岐による効果を図 11に示す。効果は演算器数によって異なるが、最低でも約 10% 以上の効果がみられ有効な方法であることがわかる。

6.6 複数の分岐命令の同時実行

2 つの分岐命令を同時に実行した場合の速度向上率を図 12に示す。この場合には、キャッシュアクセス遅延が 0 の場合を除いてそれほど有効ではない。特にキャッシュアクセス遅延が 2 の場合にはほとんど効果がない。

7 おわりに

LISP の定理証明プログラム BOYER 等を用いて、記号処理における命令レベル並列処理についてのシミュレーション評価を行なった。その結果、クロックスピードを低速に抑えることによってキャッシュアクセス遅延を小さくし、より多くの命令の並列実行を狙う方法と、キャッシュアクセス遅延を大きくとりパイプラインを深くすることによって高速なクロックスピードを実現し、少数の演算器で処理の高速化を狙う

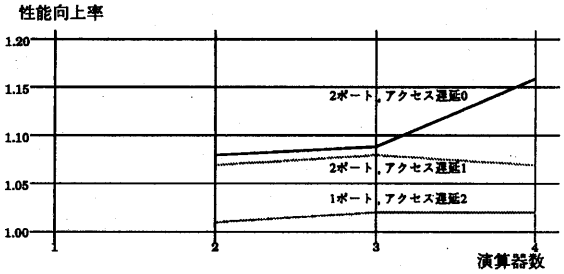

図 12: 2 分岐命令の同時実行の効果 (BOYER)

方法の両者において、ほぼ同等の性能が実現されることがわかった。

一方、比較のために行なったC言語のベンチマーク結果によれば、クロックスピードを重視する方法の方がより高速な処理が実現できると考えられる。

従って、ハードウェアの簡単さ及びプロセッサの汎用性を考えるならば、クロックスピードを重視する方法の方が優れていると考えられる。この時に実現される命令レベル並列処理は、演算器2個、キャッシュアクセス遅延2の場合で約1.3~1.4であり、タグ分岐命令の効果は約18%である。

参考文献

[1] 丸山他, "AI 言語向き RISC アーキテクチャ", 第 39 回情処全国大会
[2] 幅田他, "記号処理言語プロセッサ OLIVE の性能評価", 第 41 回情処全国大会
[3] 横田他, "LISP マシン LIME", *IPSJ, アーキテクチャ研究会 74-6*, 1989
[4] Kane, G., "MIPS R2000 RISC Architecture", Prentice Hall
[5] Taylor, G.S., Hilfinger, P.H., Larus, J.R., Patterson, R.A. and Zorn, B.G., "Design Decisions in SPUR", in *Computer*, Nov. 1986