計算機アーキテクチャ 134－19
（１９９９．８．３）

分岐予測と条件付実行

小沢 年弘† 新井 正樹† 細井 聡‡ 木村 康則†
†新情報富士通研 ‡(株)富士通研究所

条件付実行命令を用い、分岐命令を削除することにより、動的分岐予測率が向上することは、よく知られている。しかし、それが全体の実行スピード向上につながるかどうかは、条件付実行命令生成のためのオーバーヘッドがどの程度であるかに依存する。我々は、コンパイラにより生成した条件付実行命令を含むコードを、動的分岐予測を行う場合と、行わない場合で性能測定を行い、分岐予測と条件付実行命令の関係を調査した。

Branch Prediction and Predicated Execution

Toshihiro Ozawa†, Masaki Arai†, Akira Hosoi‡ and Yasunori Kimura†
† RWCP Multi-Processor Computing Fujitsu Laboratory ‡Fujitsu Laboratories LTD.
Email: ozawa@flab.fujitsu.co.jp

It is well known that predicated execution increases accuracy of dynamic branch prediction by reducing the number of branch instructions. Total execution time, however, depends on the overhead of generation of predicated instructions and the amount of the reduction of branch overhead. We measured execution time of codes which include predicated instructions on the model with and without dynamic branch.

１．はじめに

プロセッサ高性能化の手法として、命令レベルの並列性を利用して、一度に複数の命令を実行する方式が広く用いられている。このような構成のプロセッサでは、プロセッサに複数の命令を供給し続ける必要がある。このため、命令の並列度を上げること、分岐による命令供給のストール（分岐オーバーヘッド）を減らすことが求められる。

命令並列度向上のためには、out-of-order 実行やコンパイラによる命令スケジューリングなどの方法がある。分岐オーバーヘッド削減のためには、分岐予測の手法が広く用いられている。分岐予測には、ハードウエアに予測器を組み込んだ動的予測と、コンパイル時に予測する静的予測がある[1) 2)]。

さらに、命令並列度向上と分岐オーバーヘッド削減との両方に効果が期待できる方式として、条件付実行命令が上げられる[3) 4)]。本稿では、メディア処理のプログラム・セットである Mediabench ベンチマーク[5)] の一部を題材に、VLIW プロセッサにおいて、条件付実行命令による性能向上について述べる。特に、動的予測も行うプロセッサにおいてはどの程度積極的に条件付実行命令を用いるべきであるかについて考察する。

２．動的分岐予測

動的分岐予測は、ある分岐命令が以前の実行において分岐したかどうか、および分岐先アドレスをテーブル（分岐予測テーブル）に登録しておき、それに基づいて、次回の分岐方向を予測する方法である。予測に基づいて、命令フェッチを通常よりも早期に行うことで、予測が合えば、分岐オーバーヘッドを軽減することができる。逆に予測が外れると、余分なオーバーヘッドが生じる。分岐履歴から、どのように予測するかは、いろいろな方式が提案されている。分岐予測テーブルの大きさも、予測率向上のために重要な要因となる。

3．条件付実効命令

条件付実行命令とは、通常の計算に用いるデータのほかに、プレディケートと呼ぶ入力を取り、プレディケートの値により、その命令の実行を無効化する方式である。

本稿では、以下のような形式の条件付命令を想定する。

<OP>　<REG>,..., <PRED>, <VALUE>

<OP>：命令の操作

<REG>：入力、出力に使われるレジスタ

<PRED>：プレディケート・レジスタ
（汎用レジスタではない）

<VALUE>：True, False を表す値

命令の意味は、<PRED>に指定されたプレディケート・レジスタの値が<VALUE>と等しければ、<REG>を入力、もしくは出力として、<OP>の操作を行うことを意味する。

プレディケート・レジスタに値を設定するには、以下の命令を使用する。

check　<cond>, <CC>, <PRED>

<cond>：条件

<CC>：コンディション・コード・レジスタ

<PRED>：プレディケート・レジスタ

これは、<CC>の値が条件<cond>を満たしていれば、<PRED>に True を設定し、満たしていなければ、False を設定する。

条件付実行命令を用いた簡単な例を図1に示す。図1(a)に示したCプログラムを、2並列のVLIW用のコードに変換した例が、図1(b)である。このように、分岐を含むプログラムを、分岐命令を含まないコードに変換できる。さらに、図1(b)の最後の行に見られるように、命令を並列に実行できるようになる。

4．条件付実行命令の効果

条件付実行命令の利点として、次のことが言われている。

(1) 分岐命令削除による分岐命令実行コスト削減

(2) 分岐命令削除による動的分岐予測率の向上[6) 7)]

```
if  (i == j)
    k += i;
else k -= j;
```

(a) ソース・プログラム

```
subcc   i, j, 0, cc1;    nop;
check   ne, cc1, p1;     nop;
cadd k, i, k, p1, False; csub k, j, k, p1, True;
```

(b)条件付実行命令を使った命令列

図1．条件付実行命令の例

Fig. 1 Sample of predicated execution

分岐命令削減が、分岐予測テーブルへの登録量削減になり、分岐予測テーブル・ミスを減らすことができる。それが分岐予測の向上につながると言われている。

(3) 命令並列度の向上

分岐条件の両方の命令を含んだ命令列になり、命令の並列性を抽出しやすくなる。

しかし、条件付実行命令には、以下のような欠点、限界が存在する。

(1) 条件付実行命令で削除できる分岐命令は、アドレス上で順方向の分岐（前方分岐）に限られる。ループのバックエッジなどの後方分岐は削除できない。

(2) 条件付実効命令は、背反する条件の命令を同時に実行しようとする方式であり、実際に有効となる命令は、どれか一つの条件を満たす命令だけである。命令並列度が上がったように見えても、それは見かけ上に過ぎない場合もある。

例えば、if 文の then 部に含まれる命令数と、else 部に含まれる命令数がほぼ等しいならば、条件付実行命令に変換して、then 部、else 部を並列に実行することにより、どちらが実際に有効であっても、分岐命令が削除された効果が得られる。

then 部、else 部の命令数差が大きい場合には、短い方の実行は、長い方の実行に引きずられて、

遅くなってしまう。

しかし、この場合も分岐命令は削除されるので、分岐予測の向上が期待できる。問題は、どの程度までならば命令数を増やしても分岐命令を削除する価値があるのかということである。これには、分岐テーブルの大きさや、アプリケーションに現れる分岐命令数、分岐確率などが影響する。

我々は、動的分岐予測と条件付実行命令の関係を調べるために、条件付実行命令を生成するVLIWプロセッサ用コンパイラを開発した。

以下、コンパイラに実装した条件付実効命令の生成法について述べた後、実験方法、結果について述べる。

5．条件付実行命令の生成法

今回コンパイラに実装した条件付実行命令生成のアルゴリズムを図2に示す。条件付実行命令への変換を行うのは、以下の条件を満たすハンモック型と呼ばれる制御構造の部分Hに限っている。

ハンモック型制御構造H：

·Hはループを含まない。

·Hは、入り口と、出口は、それぞれただ一つの基本ブロックを持つ。

プレディケート付きの分岐命令を用いれば、ハンモック型に限らず、条件付実効命令へ変換することができる[8) 9)]。今回、ハンモック型に限ったのは、分岐命令が削除できるのはハンモック型の場合に限られ、分岐予測への影響を調べるには十分であるためである。

条件付実行命令への変換前には、H は複数の基本ブロックからなり、実行時の分岐の仕方によってかかる実行サイクル数は変化する。このアルゴリズムでは、条件付実行命令への変換で一つになった基本ブロックの実行サイクル数が、変換前の平均の実行サイクル数よりも小さければ、条件付実行命令へ変換している。実際には、分岐方向に偏りがあるので、変換した方が、実行サイクル数が増えてしまう可能性が残っている。

CFG :=プログラムの制御フロー
NG_LIST:= 空
while(H := CFG から NG_LIST に入っていないハンモック型の制御を選ぶ) {
　H に対して、条件付実行命令を使用しないコード N を生成し、平均の実行サイクル数 C_n を求める。
　H に対して、条件付実行命令を使用するコード P を生成し、平均の実行サイクル数 C_p を求める。
　if (Cp < Cn){
　　コードPを採用する。
　} else {
　　H を NG_LIST に入れる。
　}
}
CFG の残り部分のコードを生成する。

図2　条件付実行命令の生成

Fig. 2　Algorithm of predication code generatoin

5．実験環境

今回開発したコンパイラを用いて、動的分岐予測と条件付実行命令の関係をシミュレーションにより調べた。つまり、動的分岐予測を行わない場合（ベース·モデル）と、行う場合（BTB モデル）のそれぞれのマシン・モデルに対して、条件付実行命令を生成した場合（pred）と、しない場合(no-pred)の性能を測定し、その比較を行った。

5．1 マシンモデル

実験に用いたベースのマシンモデル（ベース·モデル）は、以下の通りである。

·4命令発行のVLIW：整数系2、浮動小数系2、分岐1（分岐命令があるときは、浮動小数1）

·演算器：Load/Store 2, ALU 2, FADD 2, FMUL 2, Branch 1

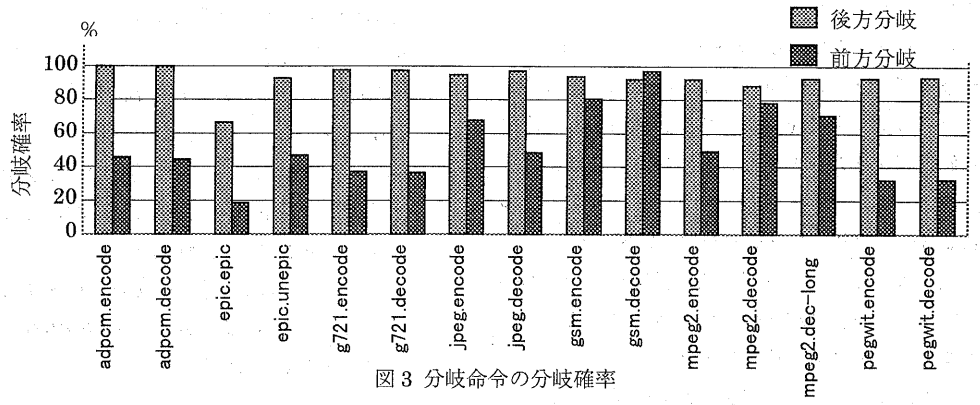

図 3 分岐命令の分岐確率

Fig. 3 Possibility of branch taken

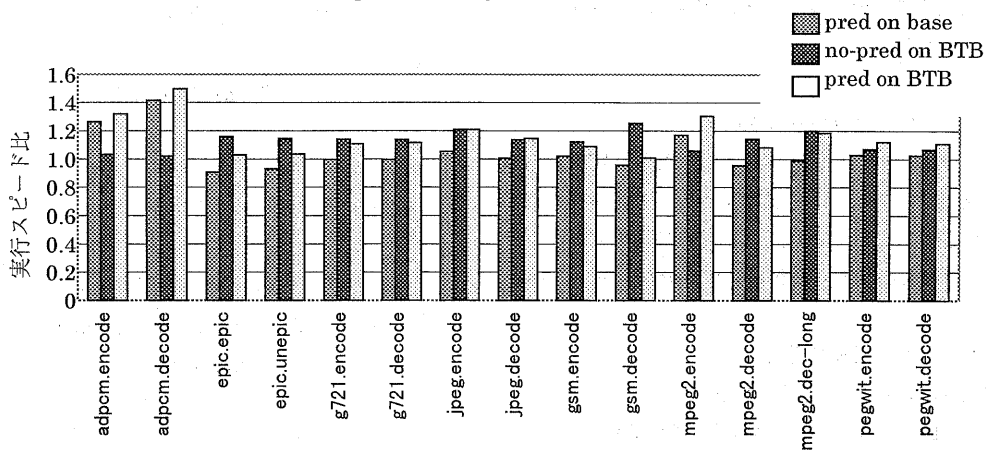

図 4 ベースと BTB モデルにおける条件付実行命令あり/なしの性能比

Fig. 4 Performance ratio between base and BTB models with/without predicated execution

・整数レジスタ、浮動小数レジスタ数は、それぞれ 64 本、プレディケート・レジスタは 4 本
・キャッシュはすべてヒット。ロードのレイテンシは 4 サイクル
・分岐命令は、分岐時に 2 サイクルの空き
・動的分岐予測なし
・基本的な命令セットは、SPARC に準拠
ただし、レジスタ・ウインドウと、ディレイド分岐はなし。

ベース・モデルに、以下の動的分岐予測器を加えたものを、BTB モデルと呼ぶ。

◇動的分岐予測器
・予測方式：2 ビット予測
・128 エントリ、2 way
・予測が成功した場合、空きサイクルなし。
・予測が失敗した場合、4 サイクルの空き。

5．2 ベンチマーク

ベンチマークとして、Mediabench の一部を用いた。これは、アプリケーションの動向として、メディア系アプリケーションの重要性が、今後、高まっていくことが予想されることから選定した。

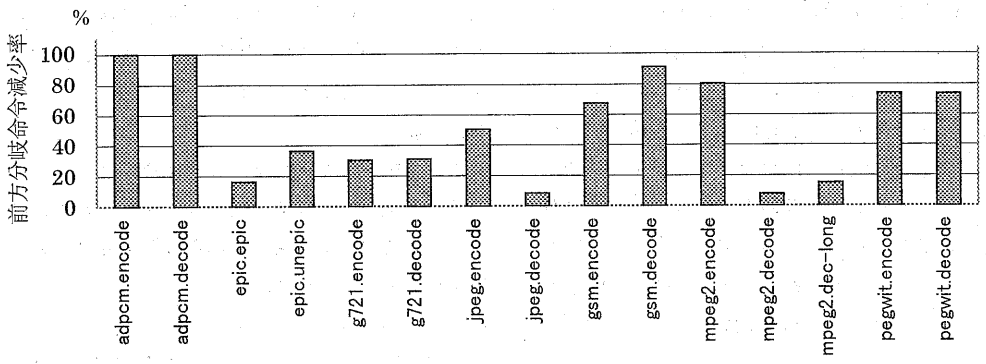

図5 predにおける動的前方分岐命令減少率

Fig. 5 Reduction of the number of executed forward branch

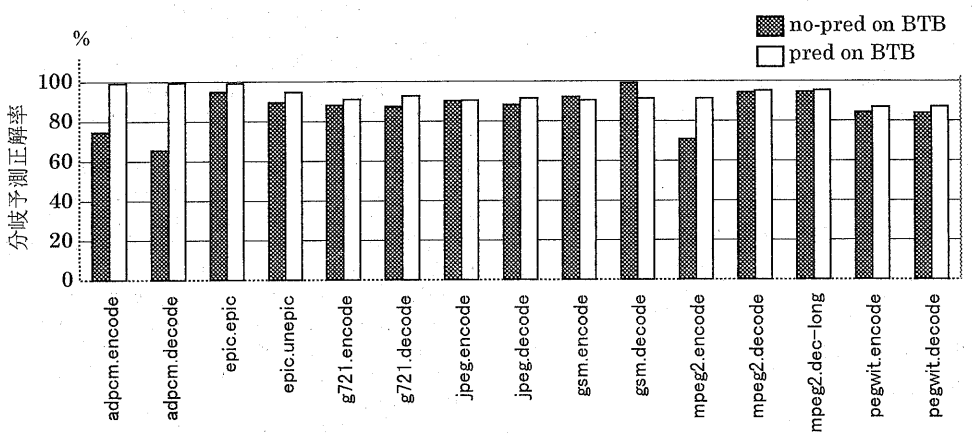

図6 前方分岐命令の予測正解率

Fig. 6 Accuracy of forward branch prediction

5．3 ベンチマークの基本的性質

図3にベンチマークの分岐確率を示す。ベースモデルを用い、条件付実行命令を生成しない場合（**no-pred**）のデータである。図では、後方分岐の確率と先方分岐の確率を分けて示している。また、無条件分岐は含んでいない。これから、通常のプログラム同様、後方分岐の分岐確率が高いこと、前方分岐は、分岐する場合としない場合が半々程度であることが分かる。

5．4 条件付実行命令と動的分岐予測

条件付実行命令を生成しない場合のベース・モデル上での実行スピード（**no-pred on base**）に対する、以下の場合の実行スピードを図4に示す。

- ·条件付実行命令を生成した、ベース・モデル上での実行スピード（**pred on base**）
- ·条件付実行命令を生成せず、BTB モデル上での実行スピード（**no-pred on BTB**）
- ·条件付実行命令を生成した、BTB モデル上での実行スピード（**pred on BTB**）

ベース・モデルにおいて、条件付実行命令によ

り、40%以上の実行スピードの向上があるプログラムもある。しかし、多くは、2%程度の向上に留まり、一部、低下しているプログラムもある。これは、各分岐方向の実行時間の平均値と、条件付実行命令での実行時間の比較で条件付実行命令を生成するか決める今回の方式では、実行時の分岐方向の偏りのために、実際には命令実行時間が増えてしまったためである。また、動的分岐予測を行うと、pred, no-pred に関わらず、実行スピードが向上している。

条件付実行命令で、先方分岐の分岐命令をどの程度減らすことができたかを動的に測定した結果を図5に示す。これは、no-pred における先方分岐の実行数を $FBEX_{no\text{-}pred}$ とし、pred における先方分岐の実行数を $FBEX_{pred}$ とした時、

$$(1 - (FBEX_{pred}/FBEX_{no\text{-}pred})) * 100$$

の値である。つまり、この値が 100 であるとは、条件付実行命令への変換によりすべての前方分岐命令が削除されたことを意味する。

図6に、BTB モデルにおける no-pred と pred の前方分岐の予測正解率を示す。pred の場合、前方分岐命令が削減されているので、多くのプログラム分岐予測率は向上している。

6．考察

図5に示したように、条件付実行命令を用いると、多くの場合、半数以上の前方分岐命令が除去でき、図6のように、分岐予測率が向上する。

しかし、ベース・モデルにおいて条件付実行命令が実行スピードの向上に結びつかなかったプログラムにおいては、BTB モデルにおいても、条件付実行命令を用いない方が実行スピードが良かった。つまり、条件付実行命令により命令実行自体の処理時間が増加してしまうと、その量は、分岐予測率の向上による分岐処理のオーバーヘッドの削減量よりも多くなってしまった。これは、条件付実行命令を用いると、分岐命令自体が少なくなるので、分岐予測によるスピード向上も小さくなってしまうためである。

逆に、ベース・モデルで条件付実行命令により実行スピードの向上があったプログラムでは、分岐予測率の向上により、BTB モデルでも、より高速な処理が実現できている。

7．今後

ここで得られた結果は、分岐命令のペナルティや分岐予測方法および予測ミス・ペナルティなどにより変化する。今後、これらの値を変化させた場合を測定する予定である。

1) J. Lee, and A. J. Smith, "Branch prediction strategies and branch target buffer design", IEEE Computer, pp. 6-22, Jan. 1984

2) J. E. Smith, "A study of branch prediction strategies", in Proceedings of the 8th International Symposium on Computer Architecture, pp. 135-148, May 1981

3) B. R. Rau, D. W. L. Yen, W. Yen, and R. A. Towle, "The Cydra 5 departmental supercomputer", IEEE Computer, vol. 22, pp. 12-35, Jan. 1989

4) J. R. Allen, K. Kennedy, C. Porterfield, and J. Warren, "Conversion of control dependence to data dependence" in Proceedings of the 10th ACM Symposium on Principles of Programming Languages, pp. 177-189, Jan. 1983

5) Chunho Lee, Iodrag Potkonjak, and William H. Mangione-Smith, "Mediabench: A Tool for Evaluating and Synthesizing Multimedia and Communications Systems", in Proceedings of the 30th International Symposium on Microarchitecture, , Dec. 1997

6) S. A. Mahlke, R. E. Hank, R. A. Bringmann, J. C. Gyllenhaal, D. M. Gallagher, and W. W. Hwu, "Characterizing the impact of predicated execution on branch prediction", in Proceedings of the 27th International Symposium on Microarchitecture, pp. 217-227, Dec. 1994

7) G. S. Tyson, "The effects of predicated execution on branch prediction", in Proceedings of the 27th International Symposium on Microarchitecture, pp. 196-206, Dec. 1994

8) S. A. Mahlke, D. C. Lin, W. Y. Chen, R. E. Hank, and R. A. Bringmann, "Effective compiler support for predicated execution using the hyperblock", in Proceedings of the 25th International Symposium on Microarchitecture, pp. 45-54, Dec. 1992

9) S. A. Mahlke, R. E. Hank, J. McCormik, D. I. August, and W. W. Hwu, "A comparison of full and partial predicated execution support for ILP processors", in Proceedings of the 22th International Symposium on Computer Architecutre, pp. 138-150, June, 1995