計算機アーキテクチャ 110－9
設 計 自 動 化 73－9
（1995. 1. 19）

分岐確率と広域命令スケジューリング

林 正和　　松山 学　　堀田耕一郎

富士通株式会社

本稿は、分岐を越えて命令を移動するトレーススケジューリングと分岐確率の関係について述べる。トレーススケジューリングは、分岐が存在すると、より多く通るパスをスケジューリング範囲に加えていくことによって、そのスケジューリングの範囲を大きくしてく手法であるため、分岐する／しないという情報が重要になってくる。そこで、広域命令スケジューリングの効果を高めるために、プロファイル機能を利用して、分岐情報をコンパイラに与える方法とコンパイラの分岐予測手段を拡張した手法のそれぞれを実現した。両者それぞれの効果について述べる。

Branch Probability and Global Instruction Scheduling

Masakazu HAYASHI, Manabu MATSUYAMA, KohIchiro HOTTA

FUJITSU LIMITED

This paper describes the relation between the branch probability information and the performance of Trace Scheduling. If there is a branch instruction, Trace Scheduling selects more frequent pass and extends its scheduling area. Therefore, it is important for Trace Scheduling to know whether each branch instruction is taken or not. This paper shows the way to tell the branch probability information to the compiler with using branch profile information and some ways for static branch prediction in a compiler. Also we present the effect of the performance by the both ways.

1 はじめに

1サイクルで複数命令実行できるRISCプロセッサが数多く発表されている。これらのプロセッサの性能向上には、最適化コンパイラの役割りが非常に大きい。特に、命令スケジューリングは、演算パイプラインのストールを避け、演算器を有効に利用するための最適化技術であり、アーキテクチャの性能を引き出すためには必須の技術である。また、非科学計算系プログラムでは、分岐命令の実行頻度が非常に高いことが知られている([1])。このようなプログラムに対しては、分岐命令を越えて命令を移動させる広域命令スケジューリングを行なうことによって、基本ブロック内の命令スケジューリング以上の効果を期待できると考えられる。

我々は、トレーススケジューリングといわれる広域命令スケジューリングを採用したコンパイラによってプログラムの実行性能向上を図ろうと試みている。

トレーススケジューリングは、分岐命令が存在したら、より多く実行される経路を選択することによって、スケジューリング範囲を拡張していく手法である。そのため、コンパイラの分岐予測の正確さが、スケジューリングの効果に大きく影響すると考えられる。

今回我々は、分岐する確率（以下、分岐真率と呼ぶ）の情報とトレーススケジューリングの性能の関係を調べるために、トレーススケジューリング機能をもったコンパイラに対し、分岐真率の情報を与える手段を構築した。この手段には、プロファイル機能を利用し、コンパイラの外部から与えられるものと、コンパイラ自身の分岐予測技術の改良による方法によって与えられるものの2つが考えられる。本論文では、これらの2つの手段と、それらの効果について考察する。

2 トレーススケジューリング

基本ブロック内の命令スケジューリングでは、命令列中に分岐命令が現れると、そこでスケジューリングの範囲が切れてしまう。そのため分岐命令が頻繁に出現するようなプログラムでは、命令を移動する範囲が狭くなり、その結果命令の並列性を十分に得ることができないことがある。

トレーススケジューリング([2],[3])は、以下のようなステップで行なわれる広域命令スケジューリングであり、分岐を越えて命令を移動できるために、基本ブロック内スケジューリングに比べて、命令の並列度を高めることができる。

1. まず、コンパイラは未スケジューリングの命令の中で、1番実行されると予想される部分の命令を探す(Seedという)。そして、Seedから命令を前後をサーチし、分岐が現れると、より多く分岐するパスを予想し、そのパスをスケジューリング範囲に加えていく。この範囲をトレースと言う。トレースの選択処理は、トレースがループ構造を越えたり、予め定められた命令数を越えたりすると終了する。

2. 上記で得たトレースを1つの基本ブロックのように考え、この範囲に対して、命令スケジューリングを行なう。

3. このままでは、実際にプログラムが分岐予測と異なった経路を実行した場合、実行されるべき命令が実行されない場合がありうる。従って、このような命令を分岐先／分岐の合流先にコピーする処理も行なわれ、実行結果の誤りを防ぐ。また、レジスタ割り付けの矛盾を避けるためのレジスタ転送命令やスピル命令

も出力されることがある。この時に生成されるコードを補正コードと呼ぶことにする。

4. 未スケジューリングの命令部分がある間、上記ステップを繰り返す。

3 分岐真率とトレーススケジューリング

2. で述べたように、トレーススケジューリングでは、補正コードを実行してしまうことによる実行性能のロスが生じることがある。スピルコードの例を用いて、この実行性能のロスについて説明をする。例えば、図1のようなPROGRAM 1の場合を考える。

PROGRAM1のforループ内のブロック構成を表したものが、図2(A)である。図2 (A)において、if文で分岐する確率Pに対して、P ≥ 1-Pであったとすると、コンパイラは図2(B)のようにトレースを選択する。この時コンパイラは、Trace No.1に対して、優先的に資源を与えるため、y(yはTrace No.1に現れないデータ)に対して割り付けられていたレジスタを他の用途に使ってしまうことがありうる。このような場合には、図2(B)のように、TRACE No.2とTrace No.1の間のエッジには、yをメモリからLOADする命令及びyの値をメモリにStoreするというSPILL命令が挿入されることになる。ここで実際には、P < 1-Pであったとすると、上記のSPILL命令が、より頻繁に実行される経路に存在することになり、結果的には実行性能の低下を招いてしまう可能性もある。

```
x=...
y=...
for(  ){
  a[i]=a[i]+10
  if (a[i]!=0){
     :
    x=x+1;
     : /* y dose't
  }else{
     :
    y=y+1;
     : /* x dose't
  }
}
   =x;
   =y;
```

図1: PROGRAM 1

従って、トレーススケジューリングの実行性能向上には、分岐命令の分岐する／しないという情報を正しく把握することが必要であると言える。

4 正確な分岐情報を得る手段

コンパイラが正確な分岐情報を得る方法としては、大きく分けて以下の2つの方法がある。

1. プロファイラなど、コンパイラ外部からの情報を利用する方法。
2. ヒューリスティックな技法を用いて、コンパイラ自身の分岐予測の正確性を高めていく方法。

以下、それぞれの方法について説明する。

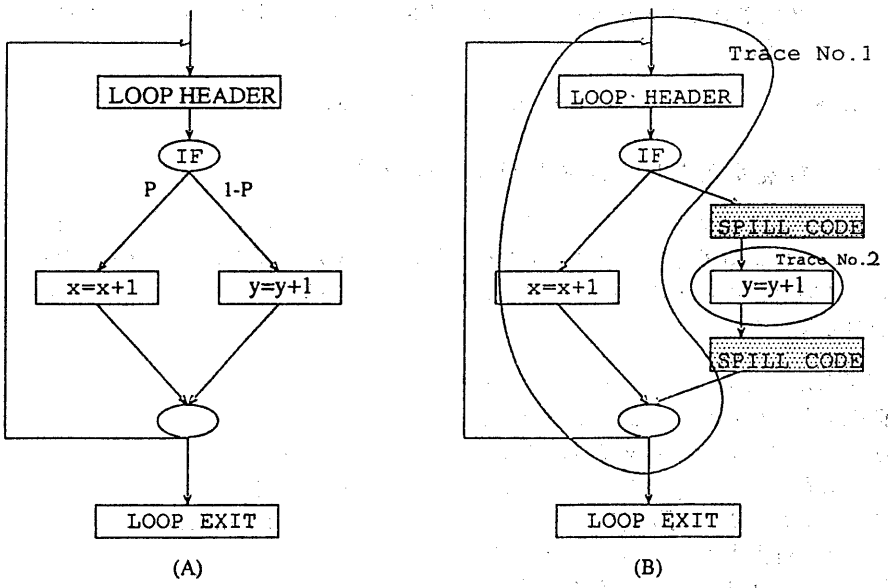

図 2: トレーススケジューリングによる補正コード出力の例

4.1 コンパイラ外部から分岐真率を指定する方法

まず、コンパイラ外部から分岐真率を指定する方法について述べる。この方法は、1度プログラムを実行させて各分岐命令に対して、そこに何回到達し、何回分岐したかの回数をカウントすることにより、分岐真率を求め、それを利用するという方法である。

我々のコンパイラでは、次のステップで個々の分岐真率を利用することができる。

1. プロファイル情報収集オプションで翻訳されたロードモジュールを実行することによって、個々の条件分岐に対し、taken/not taken をカウントし、その結果を情報ファイルに格納する。

2. プロファイル情報を利用するオプションが指定されたら、コンパイラは、上記で得られた情報ファイルを元に、各分岐命令の真率を計算し、スケジューラはその情報を利用して、トレースの選択を行なう。

この手法は、情報収集するためにプログラムを実行させることが必要であるが、全ての分岐命令に関しての正確な情報を得ることができる。

4.2 コンパイラの分岐予測技術の向上を図る方法

次に、コンパイラの分岐予測技術を向上させる手段について述べる。

分岐予想を全く行なわないということは、全ての分岐命令に対して、分岐する／しないという判断を固定してしまうことである。そして、分岐予測の手段として有名なものは、「前方向分岐は分岐せず、後方向分岐は分岐する」というものである。この予想手段でもループバックエッジに対する予想や、ループから下方向への飛び

出しの予想などは、ループの実行回数が多い場合などは、正解することになるので、ある程度の効果を期待できる。さらに最近では、ループを構成する分岐以外の分岐に対しても、ソース情報や実行回数に関する統計情報を用いることによって、前述した分岐予測よりも精度の高い分岐予想が可能であることが知られている ([4],[5])。

今回我々は、現状のコンパイラが生成するコードといくつかのプログラムのソース情報、実行回数に関する統計情報を参考にし、次に挙げる分岐予想手段をコンパイラに実装した。以下、それぞれについて簡単に説明する。

1. ループバックエッジは、ループに戻る方の確率を高く、ループから出る分岐はループから出る方の確率を低く設定する。

2. ループの回転数がコンパイル時に不明な場合に、ループを実行するか回避するかの分岐をコンパイラが生成する。この分岐に対しては、ループを実行する方を高い確率に設定する。

3. 'abort','exit','error','print','write' などを文字列に持つ関数は、例外処理的な用途が多いと考え、これらの文字列を含む処理への分岐確率を低くする。

4. ループを構成しない分岐命令を命令の種類別及び入力ソース言語別に、分岐する／しないの調査を行なった。我々の調査によれば、例えば、整数型の bgt は、表 1 に示したように多くのプログラムで、分岐しないというデータが得られた。従って、整数型 bgt の分岐は分岐する方を低い確率に設定した。このような調査を全ての分岐命令に対して行ない、言語別に表 2 のように定めた。表 2 において、'Taken' は分岐する方を高い確率に設定することを意味し、'Fallt' は分岐しない方を高い確率に設定することを意味する。

5 実行結果

実行結果は、コンパイラにとってどのようなヒューリスティックが有効なのかということを確かめるために以下の 9 パターンを比較する。

Default コンパイラは、全ての分岐命令は「分岐しないもの」と判断する。

Loop 4.2で述べたヒューリスティック手法のうち、1, 2を実施する。

Function 4.2で述べたヒューリスティック手法のうち、3を実施する。

Branch 4.2で述べたヒューリスティック手法のうち、4を実施する。

Loop+Function Loop ヒューリスティックと Function ヒューリスティックを実施する。優先順位は、Loop > Function とする。

Loop+Branch Loop ヒューリスティックと Branch ヒューリスティックを実施する。優先順位は、Loop > Branch とする。

Function+Branch Function ヒューリスティックと Branch ヒューリスティックを実施する。優先順位は、Function > Branch とする。

ALL Loop, Function, Branch の各ヒューリスティックを実施する。優先順位は、Loop > Function > Branch とする。

表 1: 整数型 BGT 命令の分岐情報

Program	言語	Trip	Taken	Prob
espresso	C	14380747	4724624	0.329
eqntott	C	1083281	96632	0.089
li	C	36523695	36385372	0.996
gcc	C	13002796	5156399	0.397
mdljsp2	F	956085	0	0.000
spice2g6	F	39138330	567077	0.014
wave5	F	15645	13677	0.874
fpppp	F	158633	496	0.003

表 2: 今回設定した Default 分岐情報

C 言語の時

	BNE	BEQ	BLE	BLT	BGE	BGT
INT	Taken	Fallt	Taken	Taken	Fallt	Fallt
FLOAT	Taken	Taken	Taken	Fallt	Taken	Fallt

Fortran 言語の時

	BNE	BEQ	BLE	BLT	BGE	BGT
INT	Taken	Fallt	Taken	Fallt	Taken	Fallt
FLOAT	Fallt	Taken	Taken	Fallt	Taken	Fallt

Profile 4.1で述べたプロファイルデータによる情報を利用する。

表 3 に実行結果を示す。なお、実行マシンは、S-4/10 Model40(SuperSPARC 40MHz) である。それぞれ、Default の時を 1.0 とした時の相対値である。

6 考察

まず最初に、我々の実験から、正確な分岐情報はトレーススケジューリングの実行性能向上にとって、重要な情報であるということが言える。

次に、静的分岐予測手段に関してであるが、ループに関するヒューリスティックは多くのプログラムに対して有効であることがわかる。これは、多くのプログラムにおいて、特に科学技術系プログラムにおけるループの実行頻度が大きいことを示している。また、関数に関するヒューリスティックと分岐命令のヒューリスティックは、悪影響を及ぼす場合があることがわかった。この 2 つのヒューリスティック手法に関しては、個々のプログラム固有の特徴が大きく影響している。例えば、分岐のヒューリスティックにおいて、我々は整数型 bgt 命令を 'Not Taken' と予測させるようにした。しかし、この決定は、プログラム li だけを考えた場合には悪い方向の決定となり、実際の実行性能もこのヒューリスティックによって、かえって悪くなっている。

最後に、profile 情報を利用した場合について考える。例えば、tomcatv や swm256

表 3: 実行結果

	Default	Loop	Function	Branch	L + F	L + B	F + B	L+F+B	Profile
espresso	1.00	1.05	1.01	1.02	1.05	1.04	1.02	1.05	1.06
li	1.00	1.01	1.01	0.98	0.98	1.0	0.98	0.98	1.03
gcc	1.00	1.03	0.98	1.04	1.04	1.04	1.04	1.05	1.10
tomcatv	1.00	1.11	1.00	1.10	1.11	1.11	1.11	1.11	1.11
swm256	1.00	1.13	1.00	1.13	1.13	1.13	1.13	1.13	1.13

というベンチマークプログラムは、プログラムのループ部分が実行頻度全体の 80% 以上を占めている。このようなループ主体のプログラムにおいては、ループヒューリスティックを行なえば、 profile を利用した場合と同様の効果をあげることができる。しかし、 gcc のように実行頻度を特定の場所に絞ることができないようなプログラムにおいては、今回のヒューリスティック技法は、分岐全体の情報を収集することができる profile を利用した手法に及ばなかった。

7 今後の課題

コンパイラの静的分岐予測技術のうち、ループに関するヒューリスティックは一般的に有効であるが、他の手法は、まだプログラム固有の特徴によって左右されている。この点について、我々の分岐予測が、実際のプログラムの実行経路と異なっていた点などを検討し直し、より正確な分岐判定技術にしていくことが、必要である。

また、今回は、分岐真率をトレーススケジューリングのトレース選択に利用した場合に主眼をおいてその効果を見たが、分岐真率の情報を、その他の最適化において利用する技術なども向上させる所存である。

参考文献

[1] ヘネシー他, 「コンピュータアーキテクチャ — 設計・実現・評価の定量的アプローチ」, 日経BP社, 1992

[2] John R. Ellis, *'Bulldog : A Compiler for VLIW Architecture'*, The MIT Press 1986

[3] 松山 他, 「*RISC* プロセッサ向け広域命令スケジューリング」, 情報処理学会第 47 回全国大会, 1993

[4] THOMAS BALL and JAMES R. LARUS, *'Branch Prediction For Free'*, ACM SIGPLAN Conference on PLDI 1993 pp.300-313

[5] 細井 他, 「コンパイラによる静的分岐予測」, 情報処理学会第 49 回全国大会, 1994