計算機アーキテクチャ 113－6
（１９９５．８．２３）

拡張VLIWプロセッサGIFTの命令供給機構

古関 聰† 小松 秀昭‡ 鈴木 英俊† 深澤 良彰†
†早稲田大学理工学部 ‡日本IBM(株)東京基礎研究所

近年、プロセッサのパイプライン速度の向上が目覚しい。このようなアーキテクチャでは、条件分岐に際する分岐予測や命令供給が非常に重大な問題である。我々は、これまでに、非数値計算プログラムを含む汎用アプリケーションの高速実行を目指して、VLIWの改良提案を行なってきた。しかしながら、非数値計算プログラムを効率良く実行するためには、これまでの分岐予測や命令供給方法では解決できない問題点が存在する。本論文では、分岐動作特性を二つに分類し、これらに合わせた二つの分岐処理機構をVLIWに追加することで、この問題点を解決することを試みた。また、これらの機構を追加したことによる性能改善の評価を行なった。

Instruction Issue Mechanisms in the Extended VLIW Processor GIFT

Akira Koseki† Hideaki Komatsu‡ Eishun Suzuki† Yoshiaki Fukazawa†
† School of Science & Engineering, Waseda University
‡ Tokyo Research Laboratory, IBM Japan, Ltd.

The pipelining techniques of modern processors have improved, and the issue rate of instructions also has been increasing. In these architectures, the importance of the branch prediction and the instruction issue becomes much larger. We proposed an extension of VLIW processors to achieve high performance in executing ordinary applications including non-numerical programs. However, there exist some serious problems with the branch prediction and the instruction issue to execute non-numerical programs. In this paper, we categorize behaviors of branches into two classes and propose the two extensions of VLIW to individually solve the problems. At last, we evaluate the performance of our processor with the new extensions.

1 はじめに

アプリケーションの大規模化にともない、計算機の高速化が求められている。高速化を実現する手段の一つとして、並列処理の研究が、分散処理レベルから命令レベルまで、さまざまな粒度において行なわれている。その中で我々は、命令レベル並列処理アーキテクチャであるVLIWに注目し、研究の対象としている。

VLIWは、これまで、主に科学技術計算を対象として研究されてきたアーキテクチャであり、それを背景としてトレーススケジューリングなどのコンパイラ技術が開発されてきた[1]。我々は、VLIWアーキテクチャの問題点を克服して、科学技術計算も含めてOSやコンパイラなど全てのソフトウェアに対して、並列・高速化を図ることのできるアーキテクチャとコンパイラフレームワークを確立することを目的として研究を行なっており、これまでの研究成果として、汎用アプリケーション等を効率良く処理できる並列処理機構[2]とコンパイラフレームワーク[3]の提案を行なっている。

本稿では、VLIWのさらなる改良として、ジャンプ発生時のパイプラインの分断による実行効率低下を防ぐ独自の機構について述べ、分岐処理等の効率に関する評価を行なう。

2 本研究の背景

我々は、論文[2]において、非数値計算等のプログラムをVLIW上で実行する場合に発生する問題点を明らかにし、これを解決するためのアーキテクチャGIFT(Guarded Instruction architecture for Fine-grain Technique)を提案した。具体的なアプローチとしては、コンパイラによる並列化をサポートするハードウェアを導入することで、VLIWの並列処理能力を十分に活かすことを試みた。しかしながら、非数値計算等のプログラムを効率良く実行するためには、依然として命令供給に関する非効率が問題点として残っていると筆者らは考えている。

近年、プロセッサのパイプライン速度の向上が目覚しい。しかしながら、このような高パイプラインピッチ、深いパイプライン段数を前提とすると、非数値計算等のプログラムの命令供給には難しい問題点が存在する。

効率の良い命令供給をするためには少なくとも以下の三点を考えることが重要である。

- 分岐予測の正確さ
- 予測が成功したときの命令供給の効率
- 予測が失敗したときの命令供給の効率

しかしながら、上記の前提において、これら全てを同時に向上させることは難しい。

分岐予測を正確にするためには様々な方法があるが、その中でも、分岐の履歴を数種類の状態であらわし、その状態間を遷移しながらダイナミックに予測を行なう方法[4]が代表的である。しかしながら、非数値計算等のプログラムでは、この方式を用いた場合でも分岐予測の正確さの向上は芳しくない。一般的に、非数値計算等のプログラムにおいて、分岐予測を正確に行なうためには複雑な予測機構を用いなければならない。このため、分岐履歴を表す状態を増やす方法[4]や、ブランチコリレーションを利用した方法[5]が提案されている。しかし、我々は、無規則に変化する非数値計算における分岐を、このような方法で包括するのは難しいと考える。また、処理を複雑にすることで、予測アドレスを得るまでの時間が伸び、予測が成功したときの命令供給の時間を長くしてしまうことは問題である。

予測が成功したときの命令供給については、予測アドレスを得るまでのターンアラウンドタイムの向上がその効率を決定する重要なファクタである。予測アドレスを得るまでには、まず、キャッシュからの命令読み出しを行なって、次に分岐予測を行なうプロセスを経るのが一般的である。筆者らは、論文[6]において、VLIWの命令キャッシュに予測を行なうための情報と予測アドレスを格納し、ジャンプ時にも連続的な命令供給を行なう試みを行なってその効果を確認している。この方法は、非常に効果的であるため、UltraSPARC[7]等のプロセッサでも同様の方式が実装されている。このような高速な命令供給は非常に重要であるが、分岐予測機構が複雑になってしまっては、近年の非常に短いパイプラインピッチに合わせてアドレスを予測し、命令供給を行なうことは難しい。

また、最近では、予測が失敗したときのペナルティが大きくなっている。分岐予測機構は、非常に投機的であり、分岐予測が成功したかどうかが実際に判明するのは数サイクル後のことである。このため、パイプラインの段数が深くなるにつれ、予測が外れたときのペナルティが増大している。非数値計算等のプログラムでは、予測の失敗は少なからぬ割合で発生することが考えられるので、予測失敗時のペナルティは大きく、予測が失敗したときの命令供給の効率が悪くなる。

3 命令供給機構の設計方針

前章において、高パイプラインピッチ、深いパイプライン段数、非数値計算等のプログラムの命令供給を考えた場合、難しい問題点が存在することを指摘した。

我々は、このジレンマを以下のように考えている。

プログラムはさまざまな制御動作によって処理が行なわれ、それぞれの動作にはそれぞれの特性がある。これまでの分岐予測方法は、全ての制御動作を対象と

しており、その予測のアルゴリズムは、「分岐とはこのような動きをするものである」と仮定されて設計される。しかしながら、プログラムの制御の流れは入力データにより千差万別であり、どのように複雑なアルゴリズムを用意したとしても、意図した動きをとらないデータの存在は明らかである。一方、ループを構成する分岐や例外処理の分岐等は、入力されるデータに拘らずその動作が規則的であり、その規則をアルゴリズムに反映することで、十分正確な分岐予測を行なうことができる。

すなわち、全ての分岐を

・規則的な分岐

・不規則な分岐

に分類することができ、非数値計算等のプログラムにおいては、この違いが顕著であると思われる。

これまでの方法は、この違いを考慮することなく分岐予測を行なってきたので、第2章で述べた問題に直面せざるをえない。そこで、我々は、VLIWにおける命令供給の問題を以下のように解決する。

1) 不規則な分岐は、ジャンプの方向を予測することを断念し、むしろジャンプをせずに条件処理を行なうような機構を導入する。具体的には、分岐の両方向の命令を供給し、無効な命令を破棄するような機構を導入する。

2) 規則的な分岐は、分岐予測が有効であるので、ジャンプ履歴を基にした予測機構を活用する。

3) プログラム中の分岐の分類を、最適化コンパイラによって行なう。

以上のようなアプローチにより、以下のような利点が得られる。

1) 動作が規則的な分岐だけを予測すればよいので、簡単な機構で正確な予測を行なうことができる。

2) 予測機構が簡単であるので、予測アドレスを得るまでの時間が早くなる。

3) 分岐の両方向の命令を供給するので、予測が失敗したときのペナルティがない。

VLIWは最適化コンパイラの助けを借りながら、シンプルなハードウェアで高速な処理を行なうことが特徴である。我々の拡張は、この特徴を継承したものであり、非数値計算プログラムの効率の良い命令供給のためには最適なアプローチであると考える。

我々は、上記の機構を実現するため、不規則な分岐を処理するために、条件分岐をジャンプなしで実行できる条件実行機構を導入した。また、規則的な分岐を処理するために、キャッシュエントリに予測情報を格納し、ジャンプ時にも連続してアドレスを供給するような命令供給機構を導入した。分岐の分類を行なう最適化コンパイラについては、紙面の都合上割愛した。

4 GIFTのアーキテクチャについて

GIFTは機能非均質型VLIWで、固定小数点演算ユニット、浮動小数点演算ユニット、ジャンプ処理ユニット、メモリ処理ユニットそれぞれを複数個持っている。

GIFTの基本的パイプライン構成は、図1のようになっている。固定小数点パイプラインと、ジャンプパイプラインは、フェッチステージ、デコードステージ、実行ステージから構成され、浮動小数点パイプラインとメモリパイプラインは、フェッチステージ、デコードステージ、第一実行ステージ、第二実行ステージから構成されている。各命令は、この各ステージをメジャーサイクルとして命令の投入が行なわれ、さらに、各ステージは四つのマイナサイクルに分割されている

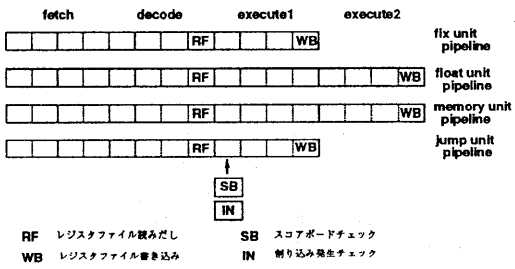


図1: パイプライン構成

図1はGIFTのパイプライン構成である。各命令は、デコードステージの最後のマイナサイクルで、レジスタファイルの読み込みを行ない、実行ステージの最後のマイナサイクルでレジスタへの書き込みを行なう。また、全ての命令のスコアボード、割り込みのチェックは実行ステージの最初のマイナサイクルで演算と並行して行なわれる。

5 GIFTの条件実行機構

5.1 不規則な分岐と条件実行

図2は、ある配列の要素について、奇数のものと偶数のものの個数をカウントするプログラムである。入力データに偏りがなければ、奇数かどうかを判定する分岐の動作に規則性はない。

これを、分岐が失敗した時のペナルティがcサイクルであるプロセッサで実行することを考える。分岐予測の正確さを$p(<1)$、if文の実行回数をMAXとすると、全体で、$c \times (1-p) \times MAX$サイクルが無駄になる計算になる。図2のようなプログラムにおいて、高い分岐予測の正確さを得ることは難しく、また、最近のプロセッサでは分岐が失敗した時のペナルティは小さくはないことを考えると、この$c \times (1-p) \times MAX$サイクルの無駄は深刻である。

```
for (i= 1; i < MAX; i++){
   tmp = ARRAY[i];
   if (ODDP(tmp)){
      odd++;
   }else{
      even++;
   }
}
```

図 2: サンプルプログラム

条件実行では、thenブロックの命令とelseブロックの命令が同時に供給される。実行条件はなんらかの方法で記憶され、この情報をもとに条件に合わない命令が無効化される。このような方式をもとに命令供給を行なうことで、分岐の動作の規則性にかかわらず、パイプラインの分断が発生することなく命令を処理することが可能である。

5.2 条件実行機構の実装

GIFTの条件実行機構は、VLIWの一命令語 (a Very Long Instrucion Word) の各命令に付加された、実行時に満足していなければならない条件部 (ガード) と、条件フラグレジスタ (cf register) と、条件に合わない命令を無効化する仕組みから成っている。条件部は、各条件フラグレジスタと一対一に対応したフィールドで構成されており、この条件部を使って、各命令の実行条件を記述することができる。

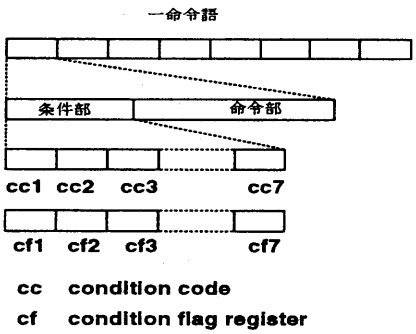


図 3: ガード付き命令と条件フラグレジスタ

図3において、条件部の cc_j と条件フラグレジスタ cf_j が対応している。cf_j はプログラム上の比較命令その他によって真、偽がセットされる。各命令は、条件フラグレジスタのそれぞれをどのように参照するかを決定するため、cfに対応したフィールドに次の三つのうちの一つを記述することができる。

T 対応するcfが真のときに真
F 対応するcfが偽のときに真
* 対応するcfの真偽にかかわらず真

条件部が記述された各命令は、実行時に全ての cc_j と cf_j が比較され、cf_j の状態が完全に cc_j の記述にマッチしたもののみが実行される。また、その他の命令は無効化される。

条件実行により、図2のサンプルプログラムは図4のように実行される (ここからさらにループアンローリング、ソフトウェアパイプラインで最適化することもできる)。

```
L1: cc1=r1 < MAX
    (cc1)r2=Load [A+r1]; (cc1)r1=r1+sizeof(int); (!cc1)jmp L2
    cc2 = r2 & 1
    (cc2)r3=r3+1; (!cc2)r4=r4+1; jmp L1
L2:
```

図 4: 条件実行の実際

ただし、図4において、条件レジスタを*で参照する部分は省略した。例えば、(!cc1)jmp L2 は、(!cc1,*cc2,*cc3,...)jmp L2 を表している。

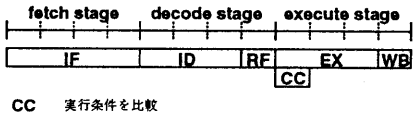


図 5: 条件実行機構のタイミング

GIFTでの計算タイミングを図5に示す。実行条件は、デコードステージで決定され、条件フラグレジスタの内容はデコードステージの終りまでに読み出される。これらの比較は、演算と並行して実行ステージの第1マイナサイクルで行なわれる。実行条件に適合しなかった命令に関しては、演算結果をレジスタに書き込むことを禁止するなど、命令の副作用が生じる書き込みを取り消すことによって無効化する。

5.3 条件実行機構の導入

条件実行機構を導入することで、効率の良い分岐処理を行なうことができる。しかし、これを実現するためにハードウェアが複雑になってしまい、パイプラインピッチを圧迫してしまっては意味がない。

具体的に追加されるハードウェアは、条件フラグレジスタ、実行条件 (ガード) を保持するラッチ、比較器などである。命令無効化までの動作は前節で示した通りであるが、図5に示す通り、レジスタの読み込み、比較など、全て他の動作と並行して行なわれている。また、それぞれの動作は単純であり、各マイナサイクルの間で十分計算を完了することが可能である。

したがって、条件実行機構の導入は、何らパイプライン速度の向上を妨げるものではない。

また、条件実行を十分に活用するためには、最適化コンパイラのサポートが必要である。

条件実行では、後向き (プログラムカウンタが増える方向と反対の方向) のジャンプを実行することができない。不規則な分岐で後向きのものを、最適化コン

パイラが条件実行可能なプログラム構造に変換することが必要である。また、その他の分岐に関しても、その全てを条件実行で行なうことはできない。ガードに用いるビット数は有限であるため、if文の入れ子構造が幾重にもなると実行条件を表現できなくなるためである。入れ子構造が深いプログラムに対しては、プロファイルなどを利用してコンパイラが最適な条件レジスタの割り当てを行なうことが重要である。

6 GIFTの命令供給機構

6.1 規則的な分岐と分岐予測

分岐によるペナルティを小さくするためには、正確な分岐予測による命令供給が必要である。分岐予測機構には様々なものがあるが、分岐の履歴に基づいたダイナミックな予測 [4] が代表的である。

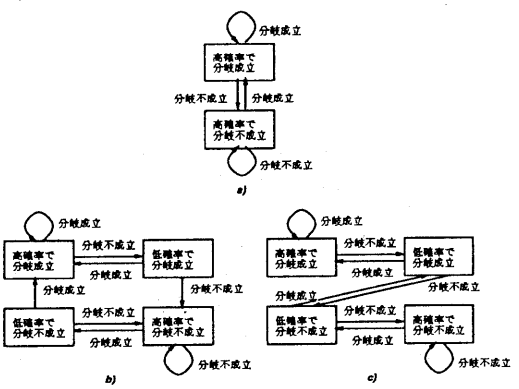


図 6: 分岐の履歴に基づいた分岐予測

図 6-a は一履歴に基づいた分岐予測、図 6-b,c は二履歴に基づいた分岐予測を示す状態遷移図である。図 6-a は一度分岐方向が変われば、以降その方向の分岐が続くことを想定した設計であり、図 6-b,c は二度連続して分岐方向が変われば、その方向が続くことを想定した設計になっている。

二履歴に基づいた方法は一履歴のものより若干複雑であるが、数値計算を主体としたアプリケーションでは、二履歴に基づく予測でかなり高い予測成功率を得ることができる。しかしながら、非数値計算等のプログラムではこの方法を用いても予測成功率はあまり高くならない。このようなプログラムの分岐予測の成功率を上げるため、2章で挙げたようなアプローチが試されている。しかし、非数値計算等のプログラムにおける様々な分岐を全て正確に予測するのは難しいと考えられる、また、なによりも、複雑な方法を用いたことで、パイプライン実行の高速化を妨げてしまうことは問題である。

我々の命令供給機構では、分岐予測を規則的な分岐に限定している。このことによって、単純なハードウェアで十分正確な分岐予測を行なうことが可能である。また、単純なハードウェアを用いることで、非常に高速な命令供給が可能である。

6.2 命令供給機構の動作

6.2.1 次候補アドレスによる命令供給

GIFTでは、命令用キャッシュメモリに格納されている命令語に、分岐予測のための情報と次に供給すべきアドレス (次候補アドレス) を付加することで、方向予測が成功したときには、分岐時にもパイプラインを分断しない高速な命令供給機構を導入した。

次候補アドレスとは、ある命令語が実行された後に実行すべき命令語のアドレスである。命令語の供給は全て次候補アドレスを参照して行なわれるので、通常のようなプログラムカウンタを用いた命令フェッチは行なっていない。命令用キャッシュメモリから供給された命令語と次候補アドレスは、プロセッサに送られる。次候補アドレスは同時に再びキャッシュメモリに送られ、次の命令語が連続的に供給される。また、命令のデコードが終ると、実際に実行されるべきアドレスが判明するので、このアドレスと次候補アドレスが比較される。これらが一致した場合はそのまま処理が続けられるが、もし、一致しなかった場合は、このアドレスから新たに命令供給を行なうよう命令供給機構に通知するとともに、パイプラインを無効化する。

命令語がキャッシュメモリに格納される際、次候補アドレスには、その命令語のアドレスと、含んでいる命令数から計算されたアドレス (引き続いたアドレス) が格納される。従って、次候補アドレスの書き換えが行なわれていない状態では、通常のプロセッサのように引き続くアドレスの命令語が供給される。ジャンプが発生した場合は、予測的にパイプラインに投入した命令語を無効にして、新たにジャンプアドレスの命令語から供給を行なわなければならい。この損失を抑えるため、ジャンプが発生した場合、その命令語の次候補アドレスを、以下に述べるアルゴリズムでジャンプ先アドレスに書き換える。この書き換えにより、次候補アドレスが正しいアドレスを示した場合には、再び連続的な命令供給が続けられる。

6.2.2 次候補アドレスの書き換え

次候補アドレスの書き換えは、予測情報の状態と、次候補アドレスが正しいがどうかの信号 (アドレス供給ミス通知) によって、以下のように行なわれる。書き換えのアルゴリズムは、図 6-b の方式を VLIW 用に拡張した。

予測情報は、命令語の各ジャンプフィールドに対応したものが、キャッシュの各エントリに用意されている。それぞれの予測情報は 2 ビットで以下のような 3 状態を保存する。

N: 供給しない

L: 低確率で供給

H: 高確率で供給

予測情報の初期状態は、引き続くアドレスに対応したものをH状態、その他をN状態とする。

アドレス供給ミス通知が偽である場合、今まで供給していたアドレスに対応する予測情報によって以下のように操作が行なわれる。

1: 予測情報がH状態であった場合、何もしない。

2: 予測情報がL状態であった場合、これをH状態にする。

アドレス供給ミス通知が真である場合、今まで供給していたアドレスに対応する予測情報によって以下のように操作が行なわれる。

1: 予測情報がH状態であった場合、これをL状態にする。次候補アドレスは書き換えない。

2: 予測情報がL状態であった場合、これをN状態にする。さらに、条件レジスタとジャンプフィールドのガードの比較で、実際に供給が行なわれたものがわかるので、これに対応した予測情報をH状態にする。また、次候補アドレスを実際に供給を行なったアドレスに書き換える。

以上のアルゴリズムは、図6-bで示した状態遷移に対応するものである。ここで、仮にジャンプフィールドが一つであると仮定すると、その動きは図6-bのものと全く一致する(図7)。

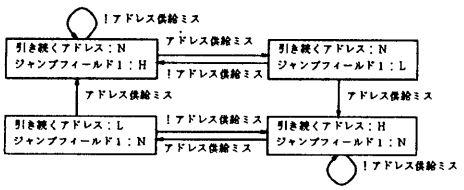


図 7: 分岐予測の状態遷移

6.2.3 本機構の実装

図8,9にこの命令供給と分岐予測機構の実装、図10に計算タイミングを示す。

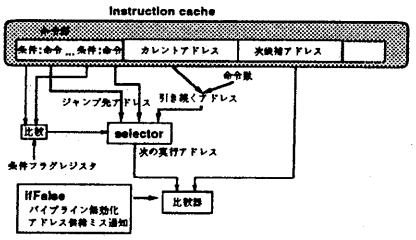


図 8: 命令供給機構

図10において、命令語の各ジャンプフィールドに置かれた命令から、ジャンプ先アドレスが計算される。

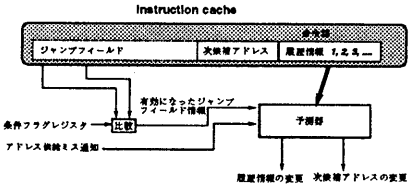
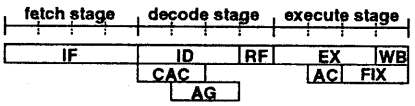


図 9: 分岐予測機構



CAC 引き続くアドレスを計算
AG プログラムカウンタ相対ジャンプアドレスを計算
AC 次候補アドレスとジャンプアドレスを比較
FIX 次候補アドレスと予測情報の更新

図 10: タイミング

GIFTでは、レジスタによる直接ジャンプと、プログラムカウンタ相対ジャンプをサポートしており、これらは実行サイクルの直前までに決定される。実行サイクルに入ると、条件フラグレジスタとガードが比較され、どのジャンプ命令が有効になるかが決まり、次に実行する命令のアドレスが決まる。どのジャンプ命令も有効にならなかった場合は、引き続くアドレスが次に実行する命令のアドレスとなる。次に実行されるアドレスと次候補アドレスの比較は、実行ステージの2番目のマイナサイクルで行なわれる。

6.3 命令供給機構の導入のコスト

本命令供給機構を導入することで追加されるハードウェアは、次候補アドレス、予測情報、予測器である。次候補アドレス等をキャッシュエントリに追加することで、1ラインのビット数は増えるが、現在の集積技術ではすでに数百万トランジスタを1チップに集積可能であり、この問題は集積技術の向上で解決可能である。

予測器のアルゴリズムは、簡単な状態遷移アルゴリズムであり、非常に単純である。また、次候補アドレス等の書き換えも他の演算と並行して行なうことができ、全体のパイプラインピッチを圧迫するようなことはない。

7 評価

表1,2に示す二つのプログラムに対し、プログラム中の各ジャンプの分岐状態及び予測成功率等を計測した。ここでは、固定小数点演算、ロードストア、浮動小数点演算、分岐をそれぞれ最大4個実行可能で、条件フラグレジスタを8個持つGIFTを仮定した。ま

た、それぞれのプログラムに対し、前向き(プログラムカウンタが増えるのと同じ方向)の分岐を不規則な分岐、後向きの分岐を規則的な分岐とした。これは、後向きの分岐はループを構成するものが多く動作が規則的であることから判断した。

評価に用いたプログラムは、チェッカーボードパズル(ある決った形状を持つブロックを、すき間なく長方形に並べる問題)、8クイーン問題の二つである。

表1,2に、各プログラムの分岐と分岐方向、ジャンプ回数、非ジャンプ回数条件実行による吸収の可否を示した。条件実行による吸収の欄は、その分岐がGIFTの条件実行機構によって、ジャンプなしで実行できるかどうかを示すものとした。

表3,4に、それぞれの分岐に対する分岐方向の予測成功率を示した。ここで、takenは必ずジャンプすることを予測、not takenは必ずジャンプしないことを予測するものとした。また、一履歴は各分岐に対して、過去の分岐方向を記録しておき、その方向で予測するものとし、二履歴は過去の2回の分岐方向を記録し、6章で述べたような方法で予測を行なうものとした。さらに、二履歴b,cは、それぞれ6章で述べた二つの実装方法を表すものとした。

表5に、それぞれの予測方法における分岐予測の平均成功率を示した。□+条件実行となっている欄は、表1,2において条件実行が可能である分岐をこの機構で行ない、その他の分岐を□の方法で分岐予測することを示すものとした。

表1,2から、どの分岐が実際に条件実行されたがわかる。ここでは、数個の前方向への分岐を条件実行で行なうことができなかった。これは条件分岐のネストが深いので8個の条件フラグレジスタで分岐構造を表現することができなかったことによるものである。

表1,2と表3,4より、分岐方向が後向きのジャンプに関しては、高い成功率で分岐予測ができることがわかる。また、一履歴より二履歴で分岐予測した方が成功率が高く、二履歴cの方式よりも二履歴bの方が若干成功率が高いことがわかる。これは、後向きのジャンプは、ループを継続する等の処理であることが多く、分岐方向が規則的で安定していることを示している。従って、このような分岐だけを考えると、これまでの予測方法で十分なパフォーマンスを得ることができるといえる。

これに比べて、前向きのジャンプは、どの方法でも成功率が高くないものが存在する。また、一履歴、二履歴のどちらがよいとは一概に言えず、ある分岐では一履歴による予測のパフォーマンスがよく、ある分岐ではその逆である。二履歴cとbについてもどちらがよいかの判断は難しい。これは、前向きのジャンプは、分岐の方向が不規則で安定していないことを示している。これらの分岐は、データにより様々挙動をすることが予想され、固定したアルゴリズムで予測を成功させるのが非常に難しいことを示している。

我々のアプローチは、分岐方向の安定しない分岐は、条件実行機構によりジャンプを行なわずに処理を行なうことでパイプラインの分断を防ぐものであった。表5は、GIFTの条件実行機構を考慮して分岐予測を行なったものを加えた、平均の分岐予測成功率である。チェッカーボードでは、後方向への分岐予測で高い成功率を示した二履歴の方式は、前方向への分岐も合わせた全体では一履歴の場合よりもパフォーマンスが悪くなっている。これは、プログラムやデータの性質によって分岐の特性はまちまちであり、必ずしも複雑な予測機構がよいパフォーマンスを示すとは限らないことを示している。条件実行を取り入れたものでは、分岐予測の難しい前方向への分岐をジャンプせずに命令供給することで、安定して高い予測成功率を得ることに成功していることがわかる。条件実行を行なわない分岐に対しては、分岐方向の予測の正確さが重要であるが、この方式として、二履歴bを採用することで全体として非常に高い予測成功率を得ることに成功した。

結論として、GIFTでは、8クイーンやチェッカーボードパズルのような、非数値計算のプログラムに対して、高い分岐予測成功率を得ることができ、非常に効率のよい命令供給が行なわれていることが実証された。

8 終りに

本研究では、VLIWの命令供給に関する問題を分析し、これを解決すべき機構を提案した。また、いくつかのプログラム中のジャンプについて、本機構の評価を行なった。結果、本手法がVLIWに適した命令供給を実現していることが確認できた。

これからの研究としては、条件実行を十分に活かすためのコンパイラ技法を確立することが必要である。また、次候補アドレスを積極的に利用し、プログラムの実行中にこのアドレスを自由に書き換えることにより、さらなる命令パイプライン分断の緩和手法が構成できるものと思われる。

参考文献

[1] J.R. Ellis,'Bulldog: A Compiler for VLIW Architectures',The MIT Press (1985).

[2] 小松, 古関, 鈴木, 深澤, '拡張VLIWプロセッサGIFTの命令レベル並列処理機構', 情報処理学会論文誌, Vol.34, No.12, pp.2599-2610 (1993).

[3] 古関, 小松, 深澤, '命令レベル並列アーキテクチャのための大域的コードスケジューリング技法とその評価', 並列処理シンポジウムJSPP'94論文集, pp.1-8 (1994).

表 1: チェッカーボードパズルの分岐状態

分岐番号	方向	分岐回数	ジャンプ回数	非ジャンプ回数	条件実行による吸収
0	backward	128	112	16	×
1	backward	16	14	2	×
2	backward	10	9	1	×
3	backward	64	56	8	×
4	backward	8	7	1	×
5	backward	13	12	1	×
6	forward	1878245	1688505	189740	×
7	forward	5764752	3793060	1971692	×
8	forward	1971692	973154	998538	×
9	forward	998538	547360	451178	○
10	forward	451178	190932	260246	○
11	forward	260246	91963	168283	○
12	forward	168283	34218	134065	○
13	forward	5764752	134065	5630687	×
14	forward	134065	2	134063	×
15	forward	407024	43952	363072	○
16	backward	407024	272961	134063	×
17	backward	5764752	5630686	134064	×
total	-	23970790	13401070	10569720	-

表 2: 8 クイーンパズルの分岐状態

分岐番号	方向	分岐回数	ジャンプ回数	非ジャンプ回数	条件実行による吸収
0	forward	5888	736	5152	○
1	backward	5888	5152	736	×
2	backward	736	644	92	×
3	forward	46752	7196	39556	○
4	forward	39556	6468	33088	○
5	backward	33088	31040	2048	×
6	forward	15720	2056	13664	×
7	forward	2056	92	1964	×
8	backward	15720	13755	1965	×
total	-	168404	67139	98265	-

表 3: チェッカーボードパズルのパイプライン連続率

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
taken	88%	88%	90%	88%	88%	92%	90%	66%	49%	55%	42%	38%	20%	2%	0%	11%	67%	98%
not taken	12%	12%	10%	12%	12%	8%	10%	44%	51%	45%	58%	62%	80%	98%	100%	91%	33%	2%
一履歴	75%	75%	80%	75%	75%	85%	84%	84%	61%	58%	58%	56%	69%	95%	100%	79%	55%	95%
二履歴 c	86%	75%	70%	86%	75%	77%	10%	58%	59%	59%	56%	60%	74%	98%	100%	89%	67%	98%
二履歴 b	86%	75%	70%	86%	75%	85%	83%	56%	56%	50%	49%	42%	30%	87%	100%	89%	67%	98%

表 4: 8 クイーン問題のパイプライン連続率

	0	1	2	3	4	5	6	7	8
taken	13%	88%	88%	15%	16%	94%	13%	4%	88%
not taken	87%	12%	13%	85%	84%	6%	87%	96%	12%
一履歴	75%	75%	71%	73%	70%	88%	77%	91%	75%
二履歴 c	87%	87%	75%	70%	81%	94%	87%	91%	83%
二履歴 b	87%	87%	75%	85%	84%	95%	87%	91%	86%

表 5: 分岐予測の成功率

プログラム	taken	not taken	一履歴	二履歴 c	二履歴 b	一履歴 + 条件実行	二履歴 c+ 条件実行	二履歴 b+ 条件実行 (GIFT)
8queen	56%	44%	76%	81%	87%	84%	89%	90%
checker	41%	59%	85%	74%	76%	90%	94%	96%

[4] J.K. Lee, and A.J. Smith, 'Branch Prediction Strategies and Branch Target Buffer Design', IEEE Computer, Vol.17, No.1, pp.6-22 (1984).

[5] S. Pan, K. So, and J. Rahmeh, 'Improving the Accuracy of Dynamic Branch Prediction Using Branch Correlation', Proc. of 5th Annual Intl. Conf. on Architectural Support for Prog. Lang. and Operating Systems (1992).

[6] 鈴木, 小松, 深澤, 門倉, '条件実行アーキテクチャ GIFT のメモリインターフェース', SWoPP'90, CPSY90-53, pp.91-96 (1990).

[7] 'UltraSPARC-I Data Sheet', Sun Microsystems, Inc., (1994).