計算機アーキテクチャ 87－2
（1991. 3. 11）

パイプライン計算機における分岐仮実行アーキテクチャ

丸島 敏一　　西 直樹　　大沢 謙二†　　中崎 良成

日本電気(株) C&Cシステム研究所　　†日本電気技術情報システム開発(株)

分岐命令の分岐先確定を待たずに予測先命令を実行する、分岐仮実行アーキテクチャを提案する。本アーキテクチャでは、基本ブロック内のin-order実行、基本ブロック間のout-of-order実行という、準動的命令スケジューリングを行なう。これにより、従来複雑な制御を必要とした動的命令スケジューリングを行なうReservation Stationに比べ、より単純な制御論理の下で、より広い型のプログラムにおいて高い性能を見込むことができる。本稿では、このアーキテクチャの構成および動作を説明する。また、シミュレーションによる性能評価により本アーキテクチャの有用性を示す。

A Speculative Branch Architecture for Pipelined Computers

Toshikazu MARUSHIMA, Naoki NISHI, Kenji OHSAWA†, Ryosei NAKAZAKI

C&C Systems Research Labs., NEC Corporation　　†NSIS Corporation

1-1, Miyazaki 4-Chome, Miyamae-ku Kawasaki, Kanagawa 216, JAPAN

We propose a speculative branch architecture in place of the reservation station which provides a dynamic instruction scheduling. The speculative branch architecture introduces semi-dynamic instruction scheduling; programs are executed in-order intra-basic-block, out-of-order inter-basic-block. The architecture has several advantages in its simple logic and high-performance as compared with the complicated reservation station. In this report, we describe a structure and algorithms of this pipeline architecture. Moreover, performance estimation by software simulator is described.

1 はじめに

LSIチップの高集積化に伴い、プロセッサ内部アーキテクチャは高性能化を目指して種々の技法を取り入れるようになっている。現在のプロセッサのほとんどは命令制御部でパイプライン構成をとっており、さらに演算器のパイプライン化、多重化（MFU; Multiple Functional Unit）構成を採るものも少なくない。このようにプロセッサが高機能になるにつれ、演算資源を有効に使いこなすために、命令コード配置の善し悪しが性能上重要な役割を果たすようになる。通常のプロセッサは、コンパイラによる静的命令スケジューリングにより得られた命令コードを、順序通り（in-order）に実行する。このため、演算資源を有効に利用できるかどうかは実行前に固定されてしまい、実行時に依存する情報は利用することができない。これに対し、動的命令スケジューリングを行なうプロセッサでは、実行時に各命令間の依存関係を検出し、命令コードを動的に入換えて実行する（out-of-order実行）[10]。これにより、既存の命令コードとの互換性を保ちながらプログラムの高速化が可能となる。

動的命令スケジューリングを行なう代表的なものに、Tomasuloにより提案されたReservation Station [13]がある。このReservation Stationは、データ駆動計算機のマッチングメモリにあたるもので、これにより局所データフロー実行を実現する。このReservation Stationは、高機能であり広範囲の高速化が期待できるが、本来はコンパイラによる最適化によって対処すべき機能も含んでいる。また、必要となるハードウェア制御論理が複雑になり、ゲート遅延が増してクロック速度に悪影響を及ぼす恐れがある[14]。このことは、MFU構成でさらに複数命令同時発行を行おうとする場合、重大な問題となってくる[5][7]。

そこで本稿では、Reservation Stationに比較して単純な制御論理を持つ新しい命令発行方式として分岐仮実行アーキテクチャを提案する。本アーキテクチャでは、Reservation Stationの持つ機能の内コンパイラで対処すべき機能を省き、さらに実行時の情報に依存すべき機能についてはより広範囲に対処することができる。

2 Reservation Stationによる動的命令スケジューリング

パイプラインプロセッサでは複数の命令が同時に異なるステージで処理されているため、処理されている命令間に以下のような依存関係があると、それによるハザードを回避するためにパイプラインに乱れを生じる恐れがある[1]。

- フロー依存
  先行命令による結果を後続命令が参照する
- 逆依存
  先行命令の参照元と後続命令の書込み先が同一（参照前の更新不可）
- 出力依存
  先行命令と後続命令が同一の書込み先を持つ（終値保証が必要）
- 制御依存
  分岐命令の後続命令の実行要否は分岐先結果に依存する

Tomasuloが提案したReservation Stationによる動的命令スケジューリングによれば、上述の依存関係に起因する性能低下に対し、以下のような効果を得ることができる。

(1) 分岐命令の先行実行による基本ブロック間並列処理

分岐命令を先行して実行することにより、後続命令に対する制御依存をできるだけ早く解決して、基本ブロック間にまたがった並列性を抽出することができる。これを静的に行なうものに、例えば遅延分岐（delayed branch）があるが、命令セットレベルで機能が固定されており、一般的に遅延スロットは短い（ことを狙っている）ため、性能ゲインという面では小さい。また、コンパイラによる同様の効果を狙った技法に、Loop Unrolling（ループ展開）やTrace Scheduling [2]等があるが、この場合、命令コード量が増加する、使用レジスタ数が増加する等の問題がある。

(2) メモリアドレス競合の動的検出

ロード命令は、後続命令がオペランドとして参照するデータを生成するため、なるべく前方にコード配置しておくことが望ましい。この際、ストア命令を追越して配置するためにはメモリアドレスが競合していないことを保証しておかなければならない（memory disambiguation [2]）。アドレス競合の有無が実行時にしか判断できない場合には、コンパイラによる並び替えは不可能になる。

(3) 出力依存／逆依存による影響の回避

実行しようとする命令の書込み先レジスタ番号が、現在実行中の命令の書込み先レジスタ番号と一致していると、出力依存のため終値保証ができるまではその命令の実行を開始できない。Reservation Stationではレジスタ番号を、プロセッサ内部で一意に識別できるタグ番号に付換える（register renaming）ため、タグ番号に余裕がある限り出力依存による性能低下は起きない。同様の理由で、逆依存がある場合にも対処が可能である。コ

ンパイラで対処するためには、レジスタ番号が競合しないように付換えておく必要がある。但し、使用レジスタ数が増加するという問題がある。

(4) フロー依存による影響の回避

フロー依存がある場合、参照するオペランドデータが生成されるまで、その命令は実行を開始できない。このためコンパイラは、オペランドデータを生成する命令をなるべく前方に移動して、フロー依存による影響を極力少なくするのが望ましい。もし、コンパイラがこの最適化に失敗しても、動的命令スケジューリングによれば、後続命令が先行命令を追越して実行開始できるので、即座にパイプラインをストールしないで済み、性能低下を避けることができる。

このように、Reservation Station による動的命令スケジューリングにより期待される効果は多岐にわたっている。しかし、この中には本来コンパイラで対処すべき項目も含んでいる。例えば、最後にあげた命令の入換えは現在の最適化コンパイラでは当然のごとく行なわれている。また、3番目にあげたレジスタの付換えも、レジスタ数の多い近年のプロセッサでは比較的対処し易くなっている。

一方、ここであげた動的命令スケジューリングの機能全てを実現するためには以下のようなハードウェア機構が必要になる。

(a) タグ化機構
ソースレジスタ番号とディスティネーションレジスタ番号とをタグ番号に付換える。

(b) オペランド待合せ機構
生成されたデータのタグ番号とソースレジスタのタグ番号とを比較して、一致した時にそのデータをオペランドデータとして取込む。

(c) オペランド保持機構
実行開始できない命令のオペランドデータを保持する。

これらのハードウェア機構は、プロセッサの制御論理を複雑にし、ゲート遅延が増してクロック速度に悪影響を及ぼす恐れがある。このため、性能上の効果があるにもかかわらず、これまでに動的命令スケジューリングを採用したプロセッサは少ない。

3 分岐仮実行方式

3.1 準動的命令スケジューリング

ここまでに述べたように、Reservation Station のような全機能を持った動的命令スケジューリングは制御論理が複雑になり過ぎ、ハードウェアに対する負担が重い。これは、近年のコンパイラ最適化技術の向上により、静的最適化のカバレージ範囲が広がっているという背景にうまく合致する。すなわち、静的に可能な最適化は済ませておき、実行時の動作に依存する部分のみをハードウェアで対処することが重要である。

そこで本稿では、動的命令スケジューリングの持つ「分岐命令の先行実行」を実現する手段として、分岐命令を疑似的に先行実行する分岐仮実行方式を提案する。この分岐仮実行方式では、前述の「分岐命令の先行実行」と「アドレス競合検出」に機能を限定し、その他の機能はコンパイラでの最適化を前提とする。これにより、次のような準動的命令スケジューリングを行なう。

- 基本ブロック内は静的に配置された命令コードの順序に従って (in-order) 実行する。
- 分岐予測による予測先基本ブロックをオーバラップさせて実行する。この時、基本ブロックをまたがった命令同士は out-of-order に実行する。

この機能変更により、ハードウェアに必要な機構は以下のようになる。

(1) 命令発行機構

従来の Reservation Station にあたる部分で、前述のタグ化機構、オペランド待合せ機構、オペランド保持機構は不要となる。分岐仮実行方式では、基本ブロック内は順序通りに実行するため、単純なインタロック機構ですむ。但し、後述するように、後続の基本ブロックをオーバラップ実行させるために命令をバッファリングするキューが必要となる。

(2) 状態保存機構

分岐仮実行方式では、分岐先が確定する前に分岐予測先を実行してしまうため、分岐予測失敗に備えて状態の保存をしておく必要がある。これを実現する機構としては、以下のものが提案されている。

- バッファ方式

レジスタの前段にバッファを配置してレジスタを復元可能にする。

reorder buffer [9], DCAF [6], - 晴 - [15]

- 多重レジスタ方式

物理的レジスタを複数用意して論理的レジスタを復元可能にする。

checkpoint repair [4], TORCH [11] , DSNS [3]

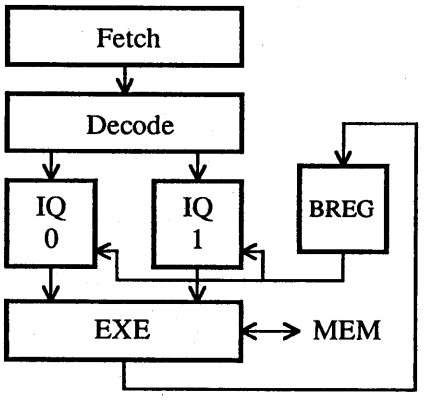


図 1: パイプライン構成

3.2 構成

提案方式の構成図を図1に示す。本構成では、フェッチ（Fetch）、デコード（Decode）の各ステージまでは処理の流れは一本であるが、その後段である命令キュー（Instruction Queue）で二つに分れる。この二つの命令キューの内どちらに命令が投入されるかは基本ブロックを単位にして交互に切替わる。すなわち、ある片方の命令キューへの投入は分岐命令で一旦停止し、以降の命令はもう一方の命令キューに投入する。また、この際フェッチ部では分岐予測を行ない、予測先命令を後続命令として命令フェッチを続行する。

二つの命令キューはそれぞれFIFO動作し、各命令キューにおいて先頭命令がデータ依存関係を満たしていればその命令は実行フェーズ（EXE）に移る。この命令キューはTomasuloのアルゴリズムにおけるReservation Stationとは異なり、任意のエントリの命令発火（Dispatch）機能は持たないため、もし、この先頭命令が発火不可能であれば同一命令キューからの発火はホールドされる。この命令キューは二つの内、片方は制御依存のない基本ブロックの実行（本実行）に使用され、もう片方は分岐予測先で制御依存が解決していない基本ブロックの実行（仮実行）に使用される。このとき、二つの命令キューは実行ユニットでの競合がない限り独立に動作可能で、これにより基本ブロック同士の並列処理を行なう。

実行を終了した命令は実行結果があればそれを状態保存機能付きレジスタ（BREG）へ書込む。本実行として実行された命令は、レジスタを更新しても以前のデータを保存しておく必要はない。しかし、仮実行として実行されてレジスタ更新時にまだ制御依存が解決していない命令は、分岐予測失敗に備えてレジスタ上の元のデータは復元可能にしておかなければならない。このために、前述の状態保存機構が必要となる。バッファ方式、多重レジスタ方式のいずれを選択するかは、例えば命令セット上のレジスタ数といった、設計時のトレードオフに依存する。

メモリアクセスに関しては、ロード命令については仮実行を行なうが、ストア命令は仮実行を許さない。これは以下の理由による。ロード命令は基本ブロックの先頭部分に配置されていることが多く、ロード命令を仮実行しないと性能が上がらない。また、ロード命令の実行結果の行き先はレジスタであるため、状態保存機能付きレジスタによれば他の演算の場合と同様に復元が可能である。一方、ストア命令は性能的にメリットが少ない上、実行結果がメモリに反映されるため、状態保存をするためには（レジスタアドレスより桁違いに広い）メモリアドレスによる管理が必要になり、処理が繁雑になる。以上の理由によりメモリアクセスに関しては、ロード命令のみ仮実行することとする。但し、ページフォールトや記憶アクセス例外といった事象が仮実行によっても起こり得るため、これらの例外を一時的に保留しておくための機構が必要となる。

3.3 動作

ここまでに述べた分岐仮実行方式の簡単な動作例を図2に示す。BLK1～3はおのおの基本ブロックで、分岐予測によってこのパスが選択されたものとする。BLK1はIQの0側に進み、先頭命令から順番に実行する。次に、BLK1が全てIQ0に進んだ後、BLK2はIQ1に進み、先頭命令から順番に実行する。この時、BLK1の一部はまだIQ0から命令を実行している可能性があり、BLK1とBLK2は並行して命令実行される。同様に、BLK3はIQ0に進み、先頭命令から順番に実行して、BLK2とBLK3が並行して命令実行される。

Reservation Stationによる動的命令スケジューリングと分岐仮実行方式との比較をするために、簡単なプログラムチャートを図3に示す。例としているプログラムは、一次元配列の定数倍を別の一次元配列に書込む（$Y(i) \leftarrow X(i) \times C$）というループ構造を持つものである。また、チャート上で実線は実行ユニット、点線はReservation Stationもしくは命令キューに命令があることを示している。本チャートから、次のことがわかる。Reservation Stationでは分岐命令を先行実行することによって、後続ブロックを早く実行開始して実行時間を短縮する。一方、分岐仮実行では、先行ブロックの分岐確定を待たずに後続ブロックを実行開始して実行時間を短縮する。この種のループでは、分岐命令を前倒しして実行可能なため、両方式はほぼ同等の処理能力があると考えられる。しかし、浮動小数点演算のような実行時間の長い命令の結果を分岐命令が参照するような場合は、分岐命令を前倒し出来ないため、分岐仮実行によってしか性能向上を望めない。

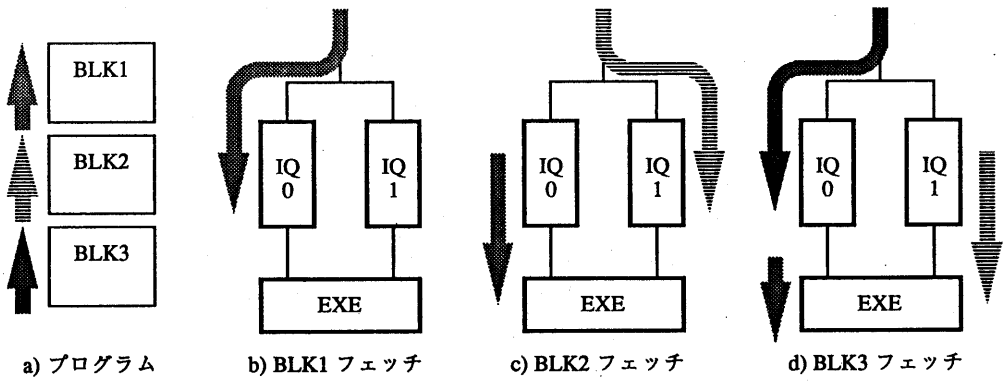


図 2: 命令キュー概略動作

3.4 命令制御

本アーキテクチャでは命令間の依存関係を検出するための機構として、レジスタの各エントリに対応した以下の制御ビットを備える。各ビットは 1 の時以下の意味を持つ。

- PB0 [i], PB1 [i] (Pending Bit)
  レジスタ i を書込み先とする命令がそれぞれ命令キュー IQ0, IQ1 中に存在する
- RB [i] (Reserve Bit)
  レジスタ i を書込み先とする命令が本実行として実行中である
- CB [i] (Conditional Bit)
  レジスタ i を書込み先とする命令が仮実行として実行中である

また、二つの命令キューの内いずれが先行する基本ブロックを保持しているかを管理しておく必要がある。以下に、レジスタ i を書込み先とする命令が処理されていく過程を説明する。

1. 命令キュー・イン

   入るべき IQ が 0 側であるとすると、PB0[i] をチェックして 0 であれば 1 にセットしてキューイン。1 であれば 0 になるまで待機する。また、ある基本ブロックが命令キュー・インを開始した場合、その基本ブロックの全命令が発火するまでは、そちら側の命令キューには新たな基本ブロックはキューインできない。

2. 命令発火

   命令キューの先頭にきた時、以下の二つの条件を満たすまで待機する。命令発火時には、本実行であれば RB[i] を、仮実行であれば CB[i] をそれぞれ 1 にセットし、IQ0 を抜けるため PB0[i] を 0 にリセットする。

   (a) 書込み先レジスタがビジーでない
   - i. 本実行の時、RB[i]=0
   - ii. 仮実行で先行側 IQ の時、RB[i]=0 & CB[i]=0
   - iii. 仮実行で後続側 IQ の時、RB[i]=0 & CB[i]=0 & PB1[i]=0

   (b) 参照レジスタがビジーでない
   書込み先レジスタの場合と同様

   (c) 機能ユニット (演算器) 等で競合がない

   また、命令キューから発火した分岐命令の分岐予測が成功するまでは、そちら側の命令キューからは新たな基本ブロックは命令発火開始できない (仮実行は 1 ブロックまで)。

3. 結果書込み

   RB[i] もしくは CB[i] をリセット。状態保存機能付きレジスタ側では、本実行による結果であればレジスタを更新し、仮実行による結果であれば復元可能な状態にしておく。

4. 分岐先確定

   (a) 分岐予測成功時
   仮実行中の命令を本実行に切換える。
   (RB[*] ← RB[*] | CB[*])

   (b) 分岐予測失敗時
   パイプラインをフラッシュして、仮実行中の命令を無効化する。(CB[*] ← 0)

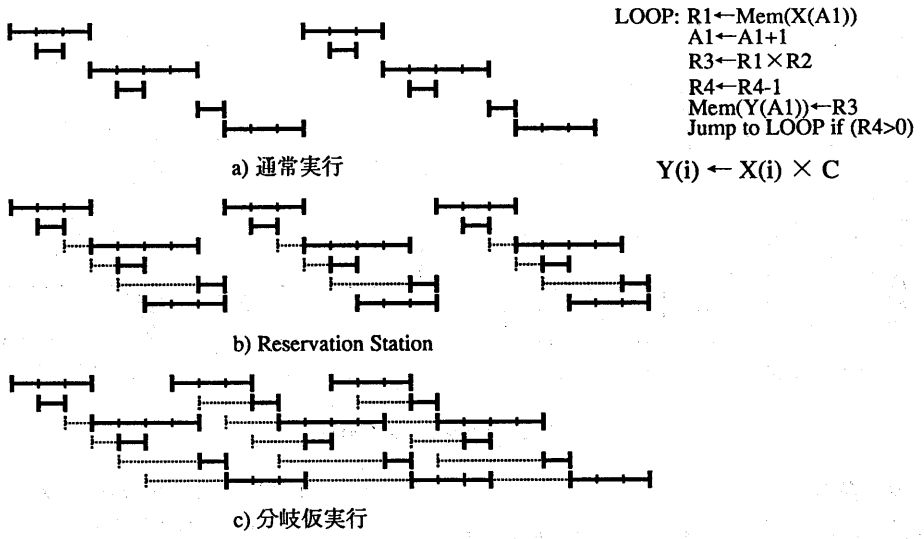


図 3: 各方式のタイムチャート

4 評価

4.1 評価条件

分岐仮実行の有用性を確認するために、シミュレータによる性能評価を行なった。評価に用いたモデルは、図1 のパイプライン構成で、命令供給能力としては最大1 命令／クロックで命令フェッチするものとした。また、実行部の機能ユニットとしては固定小数点加算器系、固定小数点乗算器系、浮動小数点加算器系、浮動小数点乗算器系とロードストアユニットを持ち、各々はパイプライン動作する。命令キューはこれらの機能ユニットを共有しており、競合しない限り両命令キューから同時実行ができるが、競合時には先行側命令キューの命令が優先的に発行される。また、これと比較するために、命令キュー部分を Reservation Station に置き換えたモデルの評価も行なった。このモデルでは、命令キューとの対比が容易であるように、命令キューの場合と同様に機能ユニットを共有する、共有 Reservation Station 方式 [12] とした。

評価に用いたコードは、分岐判定にループカウンタを用いるものとしてリバモア 14 ループ、分岐判定に浮動小数点演算結果を用いるものとしてニュートン法による求根プログラムを用い、それぞれをロードストア型の命令セットにハンドコンパイルして使用した。また、各々の命令処理時間は CRAY-1 のスカラプロセッサ [8] に準ずるものとした。これらの組合せを選んだ理由は、Weiss 等が行なったシミュレーション結果（CRAY-1 のスカラプロセッサに out-of-order 実行を施した場合のリバモア 14 ループの性能評価 [14]）との比較が行なえるからである。

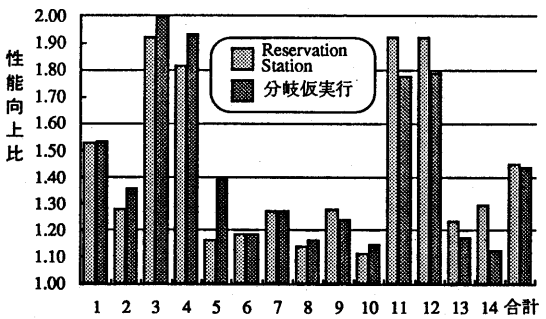
4.2 結果

4.2.1 リバモア 14 ループ

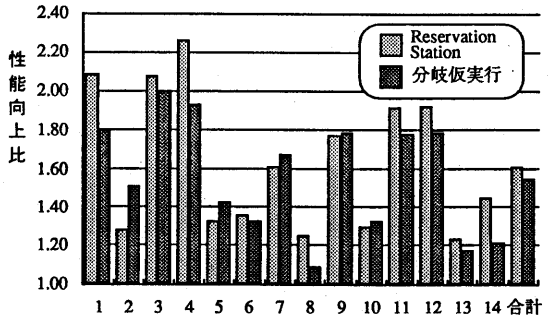
リバモア 14 ループは、14 のカーネルループから成る科学技術計算用ベンチマークプログラムである。リバモア 14 ループでは、分岐命令の判定条件としてループカウンタを用いており、out-of-order 実行でループカウンタのインクリメントやチェックを前倒しすることにより、「分岐命令の先行実行」の効果を得ることができる。また、分岐仮実行によっても同様の効果が期待できる。

リバモア 14 ループにおける評価結果を図4 に示す。a) は命令キュー 1 本当りのキュー長もしくは Reservation Station エントリ数が 4、b) は 8 である。図において、縦軸の性能向上比は通常の分岐予測における性能によって正規化している。また、横軸はループ番号である。この図から、ループによってばらつきはあるが、Reservation Station と命令キューはほぼ同等の性能比を示していることがわかる。また、Weiss 等は、CRAY-1 のアーキテクチャに対して out-of-order 実行を施したシミュレーション結果により、リバモア 14 ループ合計で 1.58 倍の性能向上が見込まれると報告しており [14]、これは本結果とほぼ一致する。

命令キュー長もしくはエントリ数を変化させた場合の、リバモア 14 ループの合計性能比（荷重平均）を図5 に示す。命令キュー長やエントリ数が増加するに従



a）命令キュー長＝4



b）命令キュー長＝8

図 4: リバモア 14 ループの各ループにおける性能向上比

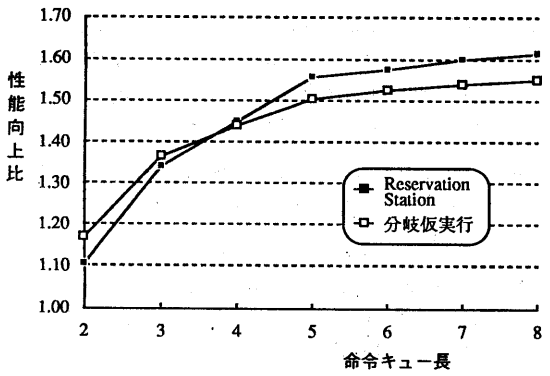


図 5: リバモア 14 ループ合計における性能向上比

い、両者の性能は同様な傾向で向上する。但し、命令を保持するだけの簡単な構造である命令キューに比較して、Reservation Station は複雑な制御論理を持つため、一般にエントリ数を多くとることができない。

本評価結果から、ループカウンタを分岐判定に用いるタイプのループプログラムにおいて、複雑な制御論理を必要とする Reservation Station に比較して、本アーキテクチャでは単純な制御の下でほぼ同等の性能が見込まれることがわかる。

4.2.2 ニュートン法による求根プログラム

評価に用いたニュートン法による求根プログラムのフローチャートを図 6 に示す。本プログラムではループの繰返し毎に浮動小数点データの収束チェックを必要としており、out-of-order 実行による「分岐命令の先行実行」の効果は期待できない。一方、分岐仮実行では分岐確定を待たずに後続ブロックを実行開始するため、基本ブロック同士の並列実行による性能向上が期待できる。

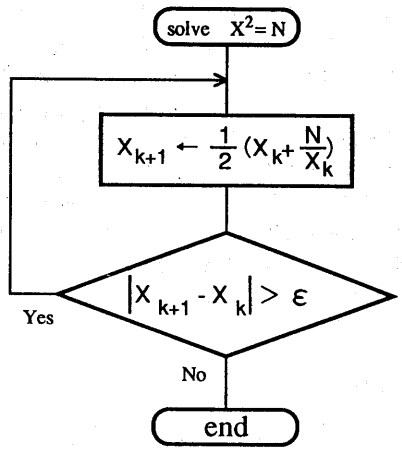


図 6: 求根プログラムのフローチャート

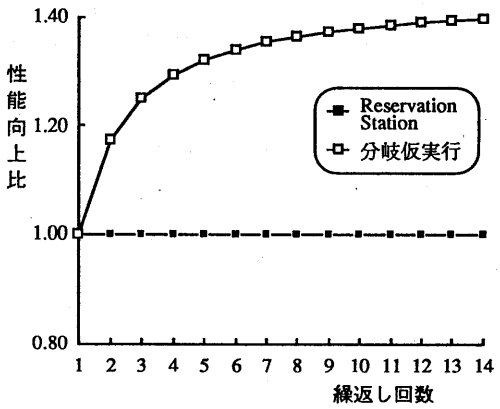


図 7: 求根プログラムにおける性能向上比

ニュートン法による求根プログラムにおける評価結果を図7 に示す。図において、縦軸は図4 と同様で、横軸は収束するまでに要するループ繰り返し回数である。本結果から、分岐仮実行による性能向上比は数回の繰返しにおいて1.3～1.4倍であることがわかる。ところが、本プログラムではループの繰返し毎に浮動小数点データの収束チェックを必要とするため、Reservation Stationでは制御依存がネックとなり性能が向上しない。この評価結果から、Reservation Stationでは高速化不可能な、制御依存が強いプログラムにおいても、本アーキテクチャが有効であることがわかる。

5 おわりに

本稿では、従来コードの高速化を単純な制御論理の下で実現する分岐仮実行アーキテクチャを提案した。また、本アーキテクチャのソフトウェアシミュレーションによる評価を行ない、分岐命令を前倒しできるプログラムにおいてはReservation Stationと同等の効果を得ることができ、さらに、分岐命令を前倒しできずReservation Stationでは高速化不可能なプログラムに対しても高速化が可能であることを示した。

今回の評価では、キャッシュヒット率や分岐予測成功率の影響を取り除いて考えるために、カーネルループを対象コードとした。今後は実アプリケーションへ適用していくことにより、本アーキテクチャの可能性を検証していく必要がある。

参考文献

[1] U.Banerjee, S.Chen, D.Kuck, R.Towle : "Time and Parallel Processor Bounds for Fortran-Like Loops", IEEE Trans. on Computers, Vol.28, No.9, pp.660-670 (1979)

[2] J.A.Fisher : "Trace Sceduling: A Technique for Global Microcode Compaction", IEEE Trans. on Computers, Vol.30, No.7, pp.478-490 (1981)

[3] 原, 久我, 村上, 富田 : "DSN型スーパスカラ・プロセッサ・プロトタイプの分岐パイプライン", 情処研資, 86-3 (1991)

[4] W.W.Hwu, Y.N.Patt : "Checkpoint Repair for Out-of-Order Execution Machines", Proc. of 14th ISCA, pp.18-26 (1987)

[5] W.M.Johnson : "Super-Scalar Processor Design", Stanford University, CSL-TR-89-383 (1989)

[6] B.D.Lightner : "The SPARC Lightning Processor", Hot Chips Symposium II (1990)

[7] 丸島, 西 : "スーパスカラプロセッサにおけるレジスタ分割方式", 情処全大39, 7X-1 (1989)

[8] R.Russell:"The CRAY-1 Computer System", CACM, Vol.21, No.1, pp.63-72 (1978)

[9] J.E.Smith, A.R.Pleszkun : "Implementing of Precise Interrupts in Pipelined Processors", IEEE Trans. on Computers, Vol.37, No.5, pp.562-573 (1988)

[10] J.E.Smith : "Dynamic Instruction Scheduling and the Astronautics ZS-1", IEEE COMPUTER, Vol.22 , No.7, pp.21-35 (1989)

[11] M.D.Smith, M.S.Lam, M.A.Horowitz : "Boosting Beyond Static Scheduling in a Superscalar Processor", Proc. of 17th ISCA, pp.344-354 (1990)

[12] G.S.Sohi : "Instruction Issue Logic for High-Performance, Interruptible, Multiple Functional Unit, Pipelined Computers", IEEE Trans. on Computers, Vol.39, No.3, pp.349-359 (1990)

[13] R.Tomasulo:"An Efficient Algorithm for Exploiting Multiple Arithmetic Units", IBM J. of R&D, pp.25-33 (1967)

[14] S.Weiss, J.E.Smith : "Instruction Issue Logic for Pipelined Supercomputers", Proc. of 11th ISCA, pp.110-118 (1984)

[15] H.Yamana, T.Hagiwara, J.Kohdate, Y.Muraoka : "A Preceding Activation Scheme with Graph Unfolding for the Parallel Processing System -Harray-", Proc. of Supercomputing '89, pp.675-684 (1989)