計算機アーキテクチャ　113－8
（１９９５．　８．　２３）

命令再構成型 VLIW プロセッサ V++における 2 つの再構成機能の評価

金岡弘記 †　　高木浩光 †　　有田隆也 ‡　　川口喜三男 †

名古屋工業大学 †　　名古屋大学 ‡

細粒度並列性を効率的に抽出する VLIW プロセッサに対し、規定型再構成機能と適応型再構成機能を付加することにより、コード量の増大化と実行時間変動に対する弱さを解決する細粒度並列プロセッサアーキテクチャである V++プロセッサを研究中である。本稿では、V++プロセッサにおける 2 つの再構成機能を活用するアーキテクチャの詳細を検討するとともに、ベンチマークプログラムを使ったシミュレーションをさまざまな角度から行ない、その結果に基づいて議論する。

Evaluation of two Instruction-Restructuring Mechanisms Adopted in V++ Processor Architecture

Hiroki KANEOKA †　　Hiromitsu TAKAGI †

Takaya ARITA ‡　　Kimio KAWAGUCHI †

Nagoya Institute of Technology †　　Nagoya University ‡

An extended model of VLIW (Very Long Instruction Word) architecture called V++ has been proposed. V++ processor has facilities for predetermined restructuring and adaptive restructuring. Therefore, V++ processer can reduce code size remarkably and become robust against dynamic variation in instruction latency. This paper describes the branch architecture utilizing both predetermined restructuring function and adaptive restructuring function, and evaluates the architechture based on the results of simulations using some benchmark programs.

1 はじめに

細粒度並列性の抽出はより重要になってきている。VLIWプロセッサでは、コンパイル時に並列性が抽出されるため、実行時には並列性抽出にともなうオーバヘッドがなく、また、ハードウェアを単純なものとすることができる。しかし、VLIWプロセッサの問題点として、コード量が増加することとオペレーションの実行時間変動に対して弱いということが挙げられる。これらを解決する手段として、オペレーションの遅延実行をもちいてVLIWの再構成によりコード量の削減を図るアーキテクチャ[4]を基に、さらに重複可能バリア同期機構[6][7]を導入することにより、VLIWにおけるコンパイラの役割の一部をハードウェアで処理する(図1参照)アーキテクチャV++を提案した[1][2]。

コンパイラの規定する再構成(規定型再構成)と重複可能バリア同期による再構成(適応型再構成)は、機能的、概念的には独立であるがハードウェアを共有しているところに特徴がある。規定型再構成機能により、ソフトウェア最適化によるコード量の増加を防ぎ、適応型再構成により、オペレーション実行の不要な待ちを削減すると同時に不要なNOPを削減する。すでに、重複可能バリア同期のスケジューリングの基本方針を示している[3]。また、V++アーキテクチャの再構成機能を活用した分岐処理方式なども検討してきた[5]。一方、規定型再構成は(RISC)^2[8]でも採用されており、また、マルチスレッド化の際にVLIW単位で遅延させる手法も提案されている[9]。

本稿では、未だ明確に定められていなかった部分も含めてV++アーキテクチャを定義し直すとともに、特に、規定型再構成機能を活用した分岐予測方式と適応型再構成機能を活用した投機的実行方式の詳細を示す。さらに、それらを含めた両再構成機能に関して、さまざまな検討事項を念頭においたシミュレーションを行ない、その結果に基づいて議論する。

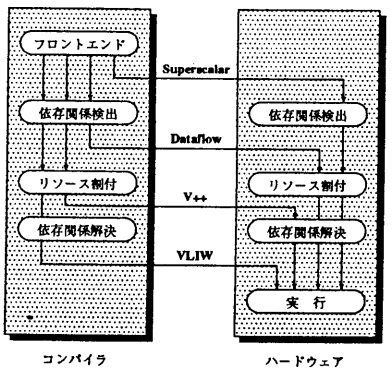

図1: コンパイラとハードウェアの役割分担

2 V++プロセッサの命令再構成

2.1 規定型再構成

規定型再構成とは、コンパイラ等によりあらかじめオペレーションに付加された実行タイミングの情報(遅延タグ)に基づいて、実行時に命令の再構成を行なうものであり、各オペレーションは、フェッチ後指定されたクロックだけ遅らされた後、実行される。これにより、フェッチされた1つのInstruction Word(命令語)を構成するオペレーションの組と、実行される1つの命令語を構成するオペレーションの組とが異なることが許され、基本ブロック間でのオーバラップが基本ブロック間の静的なコード移動なしに可能となる。その結果、コード量を減らしコードの命令密度を高くすることができる[4]。

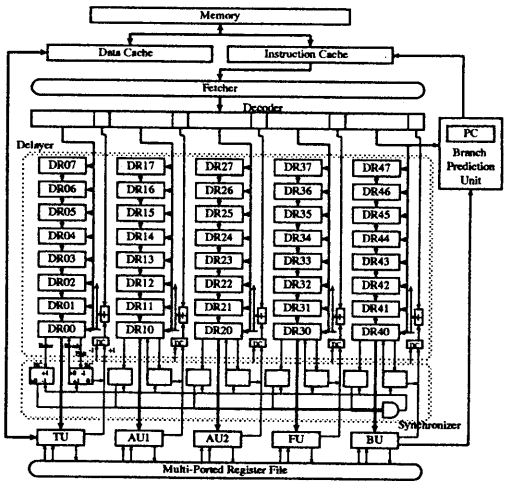

図2: V++プロセッサの基本構成

2.2 適応型再構成

適応型再構成とは、実行時に種々の要因により発生するオペレーション実行時間の変動のために生じる不要な待ちを、ハードウェアにより削減し、全体の処理時間の増大を抑えることを可能としたものである。具体的には、高速な同期機構(重複可能バリア同期機構[6][7])を用いることにより、ユニット間の実行タイミングのずれを吸収し、無駄な待ちを削減する。また、動的にオペレーション間の先行制約を保証するので、従来のVLIWで必要とされていた、タイミングを保証するためだけの多くのNOPは不要となる。

3 V++プロセッサ構成

3.1 プロセッサの基本構成

V++プロセッサの基本構成を図2に示す。V++プロセッサには、規定型再構成を行なうために遅延レジスタ(Delay Register)と、適応型再構成を実現するために同期ユニット(Synchronization Unit)および同期によって生ずるユニット間の相対的な実行の遅れを示す遅延カウンタ(DelayCounter,DC)を、従来のVLIWプロセッサに付加している。本研究では、同期ユニットとして重複可能バリア同期機構を採用しており遅延レジスタの段数は8段とし、実行ユニットは、転送ユニット(TU)×1・演

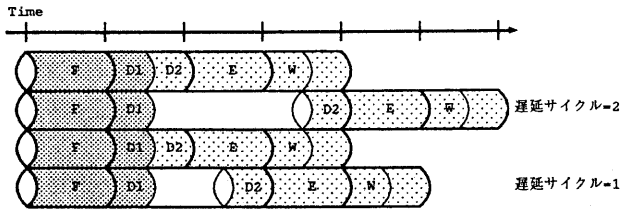

図 3: 実行遅延の例

算ユニット (AU)×2・浮動小数点演算ユニット (FU)×1・分岐ユニット (BU)×1 の構成としている。

3.2 命令パイプライン構成

命令パイプラインは、

1. F : 命令フェッチ
2. D : 命令デコード・レジスタ読み出し
3. E : 実行
4. W : レジスタ書き込み

の4ステージ構成とし、D ステージは次の2つのフェーズからなる。

1. D1 : 命令デコード
   命令コード (レジスタ番号を含む) のデコードを行なう。それと同時に、規定型再構成の遅延タグを遅延カウンタの値 (3 ビット) と加算し、これをデコードすることによって、オペレーションの遅延レジスタ (8 段) への格納位置を決め、フェーズの最終段階で、命令デコードの結果を指定の遅延レジスタにラッチする。ただし、オペレーションが NOP である場合は、遅延レジスタに上書きするのではなく、どの遅延レジスタにもラッチしないものとする。
2. D2 : レジスタ読み出し
   遅延レジスタの最終段 (DRx0) に格納されているレジスタ番号のデコード結果に基づいて、その時点での整数レジスタ/浮動小数レジスタの出力をラッチする。

D1 フェーズから D2 フェーズにかけて、規定型再構成により指定されたステップ数、また適応型再構成の同期操作のために実行が遅延される。このオペレーションの実行遅延は遅延レジスタにより行なわれる。遅延レジスタはクロックに同期して、保持しているデコード結果をシフトする。例えば、D1 フェーズで図 2の DR12 にデコード結果がラッチされた場合には、それは次のクロックで DR11 へ, また次のクロックで DR10 へとシフトする。これは 2 サイクルの実行遅延を意味する。ただし、遅延レジスタの最上段 (DRx7) には NOP をシフトインするものとする。

命令パイプライン構成を踏まえた実行遅延の例を図 3 に示す。上から 2 番目のオペレーションは、規定型再構成と適応型再構成で遅延サイクルが計 2 サイクルであり、下端のオペレーションには遅延サイクルが計 1 サイクルであることを表している。

3.3 プロセッサの基本動作制御

1. **オペレーション発行の制御**
   以下の 2 条件が満たされている場合に、最下段の遅延レジスタ内のオペレーションを実行ユニットに発行する。オペレーション間の先行関係を保証する同期回路からの Ready 信号が 1、現サイクルで該当実行ユニットのオペレーションが終了を表す実行ユニットの Read 信号が 1。
2. **遅延ユニットの制御**
   オペレーションは適応型再構成の遅延タグと遅延カウンタの値を加算して決められる遅延レジスタの位置にラッチする。しかし、その値が遅延レジスタの段数を越えてしまった場合には遅延レジスタへのラッチは行なえない。

   オペレーションが発行できない場合に、遅延カウンタを 1 増加させる。そして、すべての実行ユニットに対応する遅延カウンタの値が正になった時、すべての遅延カウンタを 1 減少させ、遅延レジスタへのラッチをしない。

   遅延レジスタのシフトできる条件は、最下段オペレーションを発行でき、かつ、デコード結果を遅延レジスタにラッチできる場合である。ただし、デコード結果を遅延レジスタにラッチできない場合でも、実行ユニットに対応する遅延カウンタが正の時には遅延レジスタのシフトができる。その時、オペレーション間の相対的な進み具合を保つために、遅延レジスタのシフトと同時に遅延カウンタを 1 減少させる必要がある。このことは、シフトしたユニットの相対的な進み具合の遅れが 1 減少したことを意味し、個別ユニットで変動を吸収できることになる。

4 2 つの再構成機能の活用

4.1 規定型再構成による分岐予測先決定方式

VLIW の特長を保持するという意味で、最小限度のハードウェア増に抑えるために、静的分岐予測方式をとる。コンパイラによって分岐命令に付加された分岐予測情報 (taken または not-taken の予測を示す 1 ビット) によって、デコードステージで分岐予測先アドレスを生成し、そのアドレスにしたがって次の命令語フェッチを行なう。通常ではこのとき、デコードステージが終了するまで分岐先アドレスを得られないために、次にフェッチされる命令語はコード上の次の命令語となる。分岐予測が taken のときはその命令は無効化されることとなる。しかし V++プロセッサでは、規定型再構成機能を活用す

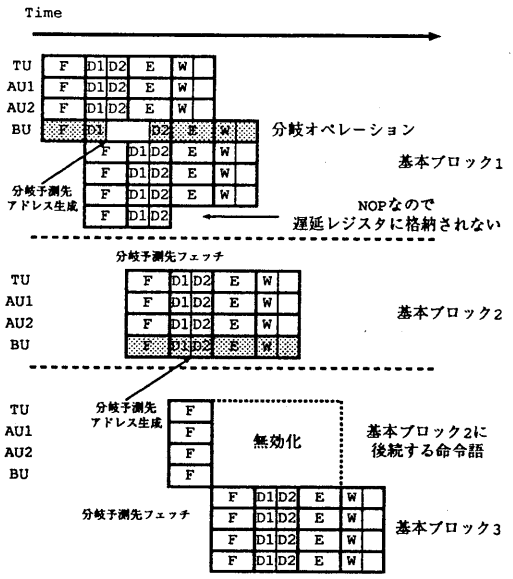

図 4: 分岐予測ペナルティの有無を表すタイミング

ることで、分岐オペレーションを本来分岐オペレーションが実行されるべき位置より早くフェッチすることができ、これにより分岐予測先アドレスを早く生成することができる。

分岐予測ペナルティがある場合とない場合を表すタイミングを図 4に示す。基本ブロック 1 から基本ブロック 2 に、また基本ブロック 2 から基本ブロック 3 に分岐予測をしていることを表している。基本ブロック 1 は分岐オペレーションのフェッチを早められる場合であり、デコードステージで生成した分岐予測先アドレスを使って即座に分岐先 (基本ブロック 2) の命令語をフェッチすることができる。これに対し、基本ブロック 2 のようにサイズが 1 命令語しかない場合には、分岐オペレーションのフェッチを早めることができず、デコードステージで分岐予測先アドレスが生成される前に基本ブロック 2 に後続する命令語をフェッチしてしまう。このとき、この分岐オペレーションの分岐予測が taken の場合には、そのフェッチした命令語を無効化することになる。

V++ の分岐方式は、分岐オペレーションのフェッチを早められない場合で taken 予測となる場合に、分岐予測ペナルティを削除できず、分岐ペナルティの期待値が大きくなる。

4.2 適応型再構成による部分投機的実行

分岐オペレーションがフェッチされると、次にフェッチされる基本ブロックが分岐予測によって決定される。分岐予測に基づいてフェッチされた命令は、通常、分岐先が確定するまで実行は遅らされるが、この遅らせる操作をしないでそのまま実行することを投機的実行と呼ぶ。通常このような投機的実行では、分岐先が確定したときに分岐予測が失敗していたならば、すでに実行してしまった分岐予測先の命令の実行を無効化せねばならない。そのためにしばしばレジスタファイルの多重化などのハードウェアが導入されるが、V++ ではこれらのハードウェアを付加せず、適応方再構成の機能を活用することで、部分的に投機的実行を実現する。具体的には、分岐予測が失敗していたときに実行してもプログラムの実行結果に影響しない部分 (副作用のない領域) を、コンパイラの最適化により最大限に確保したうえで、適応型再構成のための同期機構を用いて、副作用のない領域を越えて投機的実行が進まないことを保証する。

図 5の例は、基本ブロック B1 と基本ブロック B2 との境界における実行タイミングを示している。図 5 (a) は、動的変動のないときの実行タイミングであり、これはコンパイラが予測したオペレーション実行時間通りに実行が進んだ場合である。この場合は投機的実行は行なわれない 。図 5 (b) は、図 5 (a) に比べて TU と BU に含まれるオペレーションの実行時間がコンパイラが予測したオペレーション実行時間より長くなった場合で、投機的実行を行なわないとした場合である。基本ブロック B1 の分岐オペレーションに依存するオペレーションの終了時刻が遅れたなどの理由により、基本ブロック B2 の AU1 と AU2 と FU の実行開始までもが、制御依存が解消されるまで遅らされている。図 5 (c) は、(b) のタイミングのときに投機的実行を行なった場合であり、基本ブロック B1 の BU の制御依存が解消される前に基本ブロック B2 の AU1 と AU2 は、副作用のない範囲まで実行を進める。基本ブロック B2 の AU2 は、副作用のない範囲のオペレーションをすべて実行してもまだ制御依存が解消されていない場合であり、この場合は同期処理によって制御依存が解消されるまでそれ以降のオペレーションの実行が待たされる。基本ブロック B2 の FU は、副作用のない範囲がない場合であり、この場合は (b) の FU のタイミングと同じになる。

分岐予測が失敗したときに、投機的実行がなされたユニットがある場合は、同期機構の状態を正しく保つための特別な対処が必要である。その方法として次の 2 つを検討した。

- 同期機構の状態を初期化する方法
  投機的実行がなされているときに、分岐予測がはずれた場合、間違った同期機構の状態 (重複可能バリア同期機構の同期処理用の 2 つのカウンタ (NC,SC) 値,DC 値) を初期化することによって、同期機構の状態を正しく保つ。この方法では、BU 以外のユニットが基本ブロックの境界に達しているものと達していないものとがある場合は、すぐには同期機構の状態を初期化できないので、境界に達していないユニット全てが境界に達した後、同期機構の状態を初期化して次の基本ブロックを実行する。一方、BU 以外の全てのユニットが基本ブロックの境界に達していなかった場合は、同期機構の状態に矛盾が生じないので初期化する必要がない。

  この方法は、基本ブロック毎に実行が再開可能になるように同期タグの付加に関する制約が付く。また、正しい分岐先をすぐに実行出来ない場合がある。
- 同期機構の状態を保存する方法
  投機的実行がなされているときに、分岐予測がはずれた場合、間違った同期機構の状態を正しい同期機構の状態に置き換えることによって、同期機構の状態を正しく保つ。この方法では、正しい同期機構の

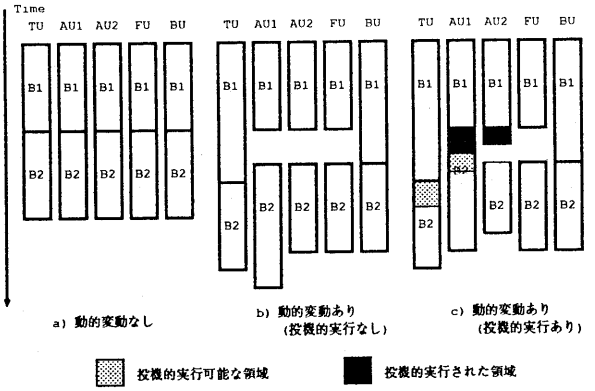

図 5: 適応型再構成を用いた投機的実行の概念

状態を保存しておかなくてはならない。投機的実行できる投機的実行レベルをnまで許す時、それぞれの同期機構は投機的実行レベルを0まで評価するものからレベルnまで評価するもののn+1個の同期機構が必要となる。ここで投機的実行レベルとは、投機的実行される基本ブロックの数であり、投機的実行レベル1というのは、1つ先の基本ブロックだけ投機的実行可能であるということである。

この方法は、投機的実行が間違いとわかった時すぐに正しい基本ブロックを実行できる。しかし、同期状態を保存するためのハードウェアが必要となる。

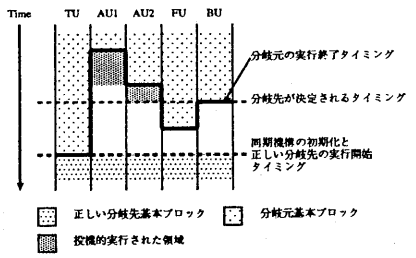

図 6: 同期機構状態の初期化による保証方法

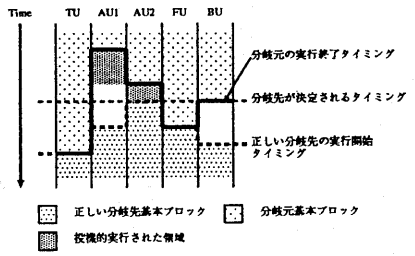

図 7: 同期機構状態の保存による保証方法

以上のような投機的実行方式には、

1. 投機的に実行したオペレーションを無効化する、またはレジスタファイルの多重化するなどのためのハードウェアコストがかからない。
2. 機能ユニットに余裕が生じた場合にのみ投機的実行を行なうので、予測が外れた場合にかえって性能が低下してしまうといった現象が起きない。

などの特長がある。ただし、コンパイラの最適化によっても、副作用のない範囲を十分に大きくできない場合には、投機的実行の効果は限定されたものとなる。

5 2つの再構成機能の評価

5.1 評価方法

ベンチマークプログラムとしてLivermore kernelのkernel1とkernel24を用い、さまざまな条件のもとで両機能をシミュレータを用いて検討した。kernel1は、配列(大きさ100)同士の計算を定数回(1,100)行なう単純な2重ループのプログラムである。kernel24は、配列(大きさ100)に格納されているデータの最小値を求めるのを定数回(1,100)行なう。これは、データの並び方でループ内に分岐先が変化するプログラムである。

VLIWは、ソフトウェアパイプライニングによる最適化を行なっている。規定型再構成による分岐予測先決定方式の評価は、遅延分岐を分岐予測ペナルティの削減に適用させたVLIWとの比較で行なった。

今回の仕様で、個別ユニットで変動を吸収可能条件(3.3参照)のデコード結果を遅延レジスタにラッチできないのは、遅延タグと遅延カウンタの値の和が遅延レジスタの段数を越えた場合, すべての遅延カウンタの値が正の場合, 分岐予測ペナルティが削減できない場合である。

プログラムのデータ配列の大きさを100としたのでキャッシュの大きさを無限大とし、キャッシングはブロック単位で行なうものとした。また、初期状態ではデータがキャッシュされていないものとする。キャッシュミスが起こった時には、Load/Storeオペレーションの実行サイクル数が1サイクルから8サイクルになる。

5.2 コード量圧縮効果

表1と表3にコード量圧縮を示す。表1のE:規定型再構成(分岐オペレーションの移動なし)は、B:VLIW(遅延分岐なし)と比較して、9命令少ない命令数で同じ実行サイクル数になっている。また、表1のN:規定型再構成(分岐オペレーションの移動なし)は、K:VLIW(遅延分岐なし)と比較して、4命令少ない命令数でほぼ同じの実行サイクル数になっている。この結果から、規定型再構成を用いることによりソフトウェアパイプライニングとほぼ同等の性能を、約62%～70

5.3 分岐予測先アドレス早期生成の効果

表1と表3に、4.1で示した分岐予測先アドレス早期生成に関する結果を示している。表3のQ:規定型再構成(分岐オペレーションの移動あり)は、R:規定型再構成(分岐オペレーションの移動なし)と比較して、約84%に実行サイクル数が短縮されている。これは、規定型再構成による分岐予測ペナルティの削減による効果の表れであると考えられる．なお、F:規定型再構成(分岐オペレーションの移動あり)が、C:VLIW(遅延分岐あり)に比べて、実行サイクル数が悪いのは、表2に示している実行タイミング上での分岐オペレーションの移動可能率が低いためである。F:規定型再構成(分岐オペレーションの移動あり)のコードでループの内部で有効な分岐オペレーションの移動ができなかったからである。

表1: 実行時間とコード量(kernel1)

	実行時間(cache miss 0%)	プログラムワード数
A	60513	-
B	70413	24
C	60613	26
D	60513	-
E	70413	15
F	70413	15
G	60613	-
H	70413	15
I	70413	15

表2: 分岐予測ペナルティ削減率(kernel1)

	遅延スロット充填率/分岐命令移動可能率(%)	
	コード上	実行タイミング上
C	66.67	99.50
F	75.00	50.50
I	75.00	50.50

A VLIW(分岐予測ペナルティなしの理想状態)
B VLIW(遅延分岐なし)
C VLIW(遅延分岐あり)
D 規定型再構成(分岐予測ペナルティなしの理想状態)
E 規定型再構成(分岐オペレーションの移動なし)
F 規定型再構成(分岐オペレーションの移動あり)
G 規定型再構成 + 適応型再構成(分岐予測ペナルティなしの理想状態)
H 規定型再構成 + 適応型再構成(分岐オペレーションの移動なし)
I 規定型再構成 + 適応型再構成(分岐オペレーションの移動あり)

表3: 実行時間とコード量(kernel24)

	実行時間(cache miss 0%)	プログラムワード数
J	51714	-
K	61615	18
L	61614	20
M	51714	-
N	62114	14
O	52114	14
P	51814	-
Q	62214	15
R	52314	15

表4: 分岐予測ペナルティ削減率(kernel24)

	遅延スロット充填率/分岐命令移動可能率(%)	
	コード上	実行タイミング上
L	60.00	51.71
O	75.00	97.63
R	75.00	97.63

J VLIW(分岐予測ペナルティなしの理想状態)
K VLIW(遅延分岐なし)
L VLIW(遅延分岐あり)
M 規定型再構成(分岐予測ペナルティなしの理想状態)
N 規定型再構成(分岐オペレーションの移動なし)
O 規定型再構成(分岐オペレーションの移動あり)
P 規定型再構成 + 適応型再構成(分岐予測ペナルティなしの理想状態)
Q 規定型再構成 + 適応型再構成(分岐オペレーションの移動なし)
R 規定型再構成 + 適応型再構成(分岐オペレーションの移動あり)

5.4 投機的実行における同期機構状態の復元法の比較

表5と表6に初期化による方法と保存による方法の比較の結果を示す。表中のCBSとはキャッシュのブロックサイズであり、cache missの項目は、上にキャッシュミス数とメモリアクセス数を示し、その下にキャッシュミス率を示している。また、増加量とはキャッシュミスによる実行サイクル数の増加量を表しており、括弧内は、キャッシュミス1回当たりの増加量を示している。kernel1(表5)の場合も、kernel24(表6)の場合も、初期化による方法と保存による方法に差がなかった。kernel1,kernel24の両方とも実行可能な投機的実行レベルが1しかなかったためである。性能に差がないのであれば，ハードウェア量の小さい初期化による方法が優れていると考えられる．

5.5 コード作成方針に関する検討

基本ブロック内で移動可能なオペレーションは投機的実行可能範囲を広げるように配置する。実行サイクル数に影響のない範囲でオペレーション配置に自由度がある場合がある。この時、オペレーションを基本ブロック中の前(プログラムの先頭の方向)に配置する場合に比較して、オペレーションを基本ブロック中の後に配置することにより投機的実行可能範囲を大きくとることが可能となる。なぜなら、オペレーションを基本ブロック中の後に配置することにより相対的に投機的実行の可能なNOPを基本ブロック中の前に配置することになるからである。

表7にkernel1のオペレーション配置による効果を示す。kernel24では、このような配置の自由度がなかった。同表より約2%程度実行サイクル数の短縮が行なわれた。この結果から、移動可能なオペレーションを後ろに配置することにより、投機的実行可能な範囲が広がり、実行サイクル数の短縮がプログラムによっては可能であることがわかる。

表 5: 初期化と保存による方法の比較 (kernel1)

LP=1 (外側ループ回数 1)				
CBS	16byte	32byte	64byte	
cache miss	153/307	78/307	40/307	
	50.16%	25.41%	13.03%	0%
I	1697	1165	923	717
S	1697	1165	923	717
T	1697	1165	923	717
I 増加量	980(6.36)	448(5.74)	206(5.15)	
S 増加量	980(6.36)	448(5.74)	206(5.15)	
T 増加量	980(6.36)	448(5.74)	206(5.15)	
LP=100 (外側ループ回数 100)				
CBS	16byte	32byte	64byte	
cache miss	153/30106	78/30106	40/30106	
	0.51%	0.26%	0.13%	0%
I	71393	70861	70619	70413
S	71395	70861	70619	70413
T	71395	70861	70619	70413
I 増加量	980(6.36)	448(5.74)	206(5.15)	
F 増加量	982(6.37)	448(5.74)	206(5.15)	
C 増加量	982(6.37)	448(5.74)	206(5.15)	

I 初期化
S 保存 (投機的実行レベル 1)
T 保存 (投機的実行レベル 2)

表 6: 初期化と保存による方法の比較 (kernel24)

LP=1 (外側ループ回数 1)				
CBS	16byte	32byte	64byte	
cache miss	53/106	27/106	14/106	
	50.00%	25.47%	13.21%	0%
R	892	710	619	537
U	892	710	619	537
V	892	710	619	537
R 増加量	355(6.70)	173(6.41)	82(5.86)	
U 増加量	355(6.70)	173(6.41)	82(5.86)	
V 増加量	355(6.70)	173(6.41)	82(5.86)	
LP=100 (外側ループ回数 100)				
CBS	16byte	32byte	64byte	
cache miss	53/10204	27/10204	14/10204	
	0.52%	0.26%	0.14%	0%
R	52665	52483	52396	52314
U	52665	52483	52396	52314
V	52665	52483	52396	52314
R 増加量	351(6.62)	169(6.26)	82(5.86)	
U 増加量	351(6.62)	169(6.26)	82(5.86)	
V 増加量	351(6.62)	169(6.26)	82(5.86)	

R 初期化
U 保存 (投機的実行レベル 1)
V 保存 (投機的実行レベル 2)

表 7: オペレーション配置による効果 (kernel1)

LP=100 (外側ループ回数 100)				
cache miss	20%	14%	10%	0%
W	104997	93452	86252	70413
X	106944	95417	88030	70413
W/X	98.18%	97.94%	97.98%	100.00%

W 移動可能なオペレーションを後
X 移動可能なオペレーションを前

5.6 各アーキテクチャの総合評価

表 8と表 9に各アーキテクチャの総合的な評価を示す。適応型再構成よる効果として、I:規定型再構成 + 適応型再構成の 1 回のキャッシュミス当たりの実行サイクル数の増加量は、kernel1(表 8) で 5.1～6.4 程度、kernel24(表 9) で 5.8～6.7 程度になっている。また、F:規定型再構成と C:VLIW では 7 となっている。また、外側ループ回数が 1 の場合には、kernel1,kernel24 の両方とも規定型再構成+適応型再構成は、規定型再構成または VLIW と比較して、ほぼ同等かそれ以上によい実行サイクル数になっている。このことから、キャッシュミスがある場合には、適応型再構成により、規定型再構成や VLIW と比較して実行サイクル数の増加が押えられていることがわかる。ただし、外側ループ回数が 100 の場合には、2 回目移行のループからキャッシュミスが起こらないので、適応型再構成による効果が小さくなり、ほぼキャッシュミスが 0%の時と同じ結果となる。また、本稿では、実行時間の変動があるオペレーションを Load/Store オペレーションだけに限り、しかも、Load/Store オペレーションを実行できる実行ユニットが TU1 つであるため、実行サイクルの延長に偏りがあり、適応型再構成に関する性能は、限られたものとなっていると考えられる。

なお、kernel1 の場合の方が kernel24 の場合より、1 回のキャッシュミス当たりの実行サイクル数の増加量が小さいのは、kernel1 の分岐ペナルティの削減率が約 50%(表 2) であり、kernel24 の分岐ペナルティの削減率が約 98%(表 2) であるのと比べて小さかったため、個別ユニットで動的変動を吸収する場面が多かったためであると考えられる。また、表 8より、I は F と比較して、外側ループ回数が 1 の場合に約 2%程度実行サイクル数の短縮がされている。しかし、外側ループ回数が 100 の場合では、約 0.2%程度実行サイクル数が大きくなっている。これは、F の方が I より内側ループで実行サイクル数が 1 サイクル短い。そのため、外側ループ回数の 100 サイクル差がつく。適応型再構成により、I は F より 16 サイクル実行サイクルの増加を押えることができている。よって、最終的に F の方が I より 84 サイクル短くなっている。

表 8: 各方式の実行サイクル数 (kernel1)

LP=1 (外側ループ回数 1)				
CBS	16byte	32byte	64byte	
cache miss	153/307	78/307	40/307	
	50.16%	25.41%	13.03%	0%
I	1697	1165	923	717
F	1795	1263	997	717
C	1697	1165	899	619
I/F	94.54%	92.24%	92.58%	100.00%
I/C	100.00%	100.00%	102.67%	115.83%
I の差	980(6.36)	448(5.74)	206(5.15)	
F の差	1078(7)	546(7)	280(7)	
C の差	1078(7)	546(7)	280(7)	
LP=100 (外側ループ回数 100)				
CBS	16byte	32byte	64byte	
cache miss	153/30106	78/30106	40/30106	
	0.51%	0.26%	0.13%	0%
I	71393	70861	70619	70413
F	71491	70959	70693	70413
C	61691	61159	60893	60613
I/F	99.86%	99.86%	99.90%	100.00%
I/C	115.74%	115.86%	115.97%	116.17%
I の差	980(6.36)	448(5.74)	206(5.15)	
F の差	1078(7)	546(7)	280(7)	
C の差	1078(7)	546(7)	280(7)	

I 規定型再構成+適応型再構成
F 規定型再構成
C VLIW

表 9: 各方式の実行サイクル数 (kernel24)

LP=1 (外側ループ回数 1)				
CBS	16byte	32byte	64byte	
cache miss	53/106	27/106	14/106	
	50.00%	25.47%	13.21%	0%
R	892	710	619	537
O	907	725	634	536
L	1001	819	728	630
R/O	98.35%	97.93%	97.63%	100.19%
R/L	89.11%	86.69%	85.03%	85.24%
R 増加量	355(6.70)	173(6.41)	82(5.86)	
O 増加量	371(7)	189(7)	98(7)	
L 増加量	371(7)	189(7)	98(7)	
LP=100 (外側ループ回数 100)				
CBS	16byte	32byte	64byte	
cache miss	53/10204	27/10204	14/10204	
	0.52%	0.26%	0.14%	0%
R	52665	52483	52396	52314
O	52585	52403	52312	52214
L	60500	60318	60227	61614
R/O	100.15%	100.15%	100.16%	100.19%
R/L	87.05%	87.01%	87.00%	84.91%
R 増加量	351(6.62)	169(6.26)	82(5.86)	
O 増加量	371(7)	189(7)	98(7)	
L 増加量	371(7)	189(7)	98(7)	

I 規定型再構成+適応型再構成
F 規定型再構成
C VLIW

6 まとめ

V++プロセッサの規定型再構成機能を活用した分岐予測方式と、適応型再構成機能を活用した投機的実行方式とを示すとともに、それらを含めた2つの再構成機能に関して、シミュレーションを行ない、その結果に基づいて議論した。まとめると次の通りである。1) 規定型再構成を用いることにより、ソフトウェアパイプライニングと同等の実行サイクルをより少ない命令数で実現可能。2) 規定型再構成を活用することにより分岐予測先アドレスを早期に生成でき、分岐予測ペナルティを削減可能。3) 投機的実行における同期機構状態の復元方法の比較により、初期化による方法がハードウェア量が小さくて保存による方法と同等の性能を出せる。4) コードを基本ブロック中の後法に配置することにより約2%程度の性能のよいコードを作成可能。5)kernel24 ではキャッシュミス率が25%程度で V++は VLIW より約 13%程度性能向上が可能。

本稿で示した手法は、新たに大きなハードウェアを付加することなく、V++の基本アーキテクチャに備わっていたハードウェアを活かすことにより、効率的に実現されている。ただし、今回の実行ユニット構成では適応型再構成の効果を十分に発揮できない面もあることがわかった。現在、評価プログラム数を増やすことにより、さらに正確な評価を目指している。引続き、機能ユニットのハードウェアの詳細の決定、規定型再構成用のタグと適応型再構成用のタグの詳細の決定、などを行なう予定である。

参考文献

[1] 有田隆也, 曽和将容, "動的再構成型 VLIW プロセッサアーキテクチャ V++" , 並列処理シンポジウム JSPP '92 論文集, pp. 265–272 (1992).

[2] 有田隆也, 曽和将容, "命令語再構成型プロセッサアーキテクチャ" , 電子情報通信学会論文誌, Vol. J76-D-I, No. 4, pp.184–186 (1993).

[3] T. Arita, H. Takagi and M. Sowa, "V++: An Instruction-Restructurable Processor Architecture", Proceedings of the 27th Hawaii International Conference on System Sciences, Vol. I , pp.398–407 (1994).

[4] 加藤工明, 有田隆也, 曽和将容, "長命令語 (LIW) コンピュータにおける命令実行遅延方式", 電子情報通信学会論文誌, Vol. J74-D-I, No. 9, pp.613–622 (1991).

[5] 金岡弘記, 高木浩光, 有田隆也, 川口喜三男, "V++ プロセッサにおける適応型再構成機能を利用した分岐処理方式", 並列／分散／協調処理に関するサマー・ワークショップ, 情報処理学会研究報告, Vol. 94, No. 66, pp. 113-120 (1994).

[6] 高木浩光, 有田隆也, 曽和将容, "細粒度並列実行を支援する種々の静的順序制御方式の定量的評価" , 並列処理シンポジウム JSPP '91 論文集, pp. 269–276 (1991).

[7] 高木浩光, 有田隆也, 曽和将容, "重複可能なバリア型同期のためのスケジューリングアルゴリズムとその性能", 並列／分散／協調処理に関する「大沼」サマー・ワークショップ, CPSY91-15, pp. 92-98 (1991).

[8] 石井吉彦, 野上忍, 小野寺毅, 三浦敏孝, 村岡洋一, "- Restructured Instruction and Switched Context RISC Processor - (RISC)2 の提案", 情報処理学会研究報告 ARC, Vol. 93, No. 71, pp. 145-152 (1993).

[9] 國領琢也, 富田眞治, "マルチスレッドをサポートする VLIW プロセッサ・アーキテクチャ", 情報処理学会研究報告 ARC, Vol. 93, No. 91, pp. 17-24 (1993).