計算機アーキテクチャ　89−13
（１９９１．　７．　１９）

実行遅延に基づく再構成VLIW型計算機の基本構成

有田 隆也*　　加藤 工明**　　曽和 将容*

*名古屋工業大学 工学部
**ナムコ　開発技術部

VLIW型計算機において、VLIWを構成する各オペレーションの実行を遅延することにより、VLIWを構成し直して実行する方式を提案する。この方式の導入により、ループの各インスタンス間やサブルーチンのコール／リターン部分でのオーバラップ実行により速度が向上されるとともに、未使用命令フィールドの減少が可能となる。本方式を採用した再構成VLIW型計算機の基本構成を示すとともに、シミュレーションにより従来のVLIW型計算機が8から58%程度高速化されることや命令語数が削減されることを示す。

Reconstruction of Instruction Words by Delayed Execution in VLIW machines

Takaya ARITA*, Komei KATO** and Masahiro SOWA*

*Nagoya Institute of Technology
**NAMCO Co., Ltd.
E-mail: ari@debris.elcom.nitech.ac.jp

This paper describes a new mechanism to speed up VLIW machines by introducing a delay parameter into the individual operations which compose a single VLIW instruction. This parameter indicates the number of the steps by which execution of the operation is delayed. This mechanism makes it possible to overlap the execution of loop and subroutine instances and to improve the efficiency of code usage. The simulation shows that this method results in 8-58% speedup.

1．はじめに

VLIW (Very Long Instruction Word) 型計算機[1]では、長い語長を持つ命令を多数のフィールドに分け、それぞれのフィールドに格納されたオペレーションにより複数の実行ユニットを同時に制御することによって細粒度並列処理を行う。VLIW型計算機の大きな特長はコンパイラが基本ブロックを越えたスケジューリングを行うことによって並列性を抽出することであり、代表的なスケジュール手法にトレース・スケジューリング[2]がある。

しかし、トレーススケジューリングに基づくVLIW型計算機に対して次のような欠点が指摘されてきた。

(1) トレース・スケジューリングにおける、副作用をもつ命令状態の退避と回復に伴うオーバヘッド、及び、多数の条件分岐を持つ非数値演算プログラムに対するトレース探索の困難さ[3]。

(2) 並列度の低いプログラム部分における命令フィールドの空きによるビット使用効率の低下。

(1)に対するアプローチのひとつに、ループ実行の高速化に着目したソフトウェア・パイプライニング[4]がある。ソフトウェア・パイプライニングは、もとのプログラムのループにおける異なる繰り返し（インスタンス）からコードを抽出してループを再構成し、パイプライン的実行によりトレース・スケジューリングとは別種の並列性を抽出する手法である。(2)に対するアプローチとして、命令を圧縮してメモリに可変長語で格納し、実行時にそれをキャッシュメモリでVLIWに拡張する方式[5]がある。この方式では、フェッチしたオペレーションをどのフィールドに格納するかという情報を各ブロックごとに付け加えて主メモリ上に格納しておき、実行時にキャッシュ詰替エンジン(cache refill engine)が、その情報に基づいてVLIWに展開するというものである。

しかしながら、ソフトウェア・パイプライニングでは、コードの移動によって生じる矛盾を解決するためにループの前後に調整用コード（pre-codeとpost-code）の追加が必要となる。また、キャッシュ詰替エンジンは、その計算機システム中で最も複雑なハードウェアであろうと言われている[5]。

これらの問題点を解決するために本論文では各オペレーションを指定したサイクル数だけ遅延させることによりVLIWを実行時に構成し直す方式[6][7]を提案する。トレース・スケジューリングで何十もの並列度が抽出できる[8]ような数値演算プログラムではなく、それほど並列度が内在していない非数値演算プログラムを対象とする。さらにコストパーフォーマンスなども考慮し、並列動作する実行ユニットが数個から10個程度である非均質型の VLIW型計算機を本論文では対象とする。

本方式では、ループ部などでソフトウェア・パイプライニングで得られるような並列性が抽出できると同時に、問題点(2)に関する効率化もはかれる。本方式によるループの高速実行の形態はソフトウェア・パイプライニングと同様である。ただし、本方式では命令フェッチのタイミングと命令実行のタイミングを変更させることが可能であるので、コンパイラにおいてループを構成する命令群を別のインスタンスに属するものから再構成する必要がない点が異なる。このため、ソフトウェア・パイプライニングでは必要となっていた、ループ前後の調整用コードを除去することが可能となった。このことは、本方式がループの高速化を実現し、しかも、問題点(2)のような未使用フィールドの除去だけでなく、別種のコード量圧縮が可能であるということを示している。

2．実行遅延の原理

本論文では、図1(a)に示すように、VLIW 1語が、メモリとレジスタ間のデータ転送、レジスタ操作、整数演算、分岐の4種類のフィールドからなるVLIW型計算機を対象とする。同図のプログラムは配列の各要素を2倍して1加えるループを表している。R1、R2などはレジスタを表し、[R1]などはレジスタ間接アドレッシングを表す。LOADLはロングワードデータのメモリからレジスタへのロード、STORELはロングワードデータのレジスタからメモ

リへのストア、SHLはレジスタ値の左論理シフトを表す。実行可能なオペレーションがみつけられなかったフィールドはNOPで埋められている。

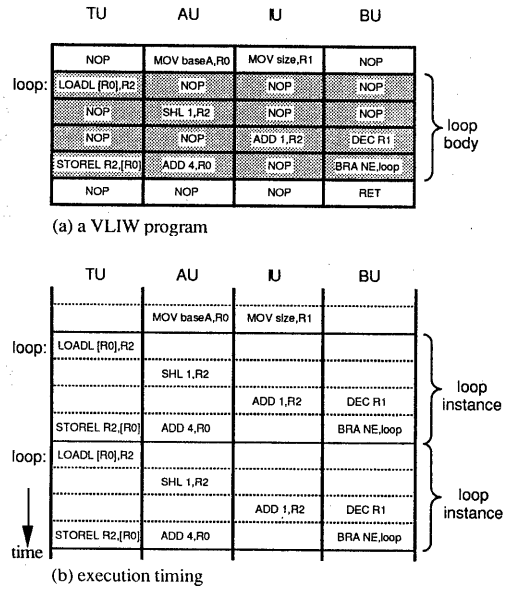

図1　VLIWプログラムとその実行

各オペレーションは、メモリからフェッチされた後、それぞれのオペレーション専用の実行ユニット、すなわち、転送ユニット(TU)、レジスタ操作ユニット(AU)、整数計算ユニット(IU)、分岐ユニット(BU)によってそれぞれ実行される。同図のプログラムは4つのVLIWからなるループを含んでおり、同図(b)に示すように実行される。NOPの実行は空白としてある。この空白部分が計算パワーを無駄にしているといえる。この例では、最大並列度は3であり、平均並列度は1.75となっている。

図2は本論文で提案するオペレーションの実行遅延の概念を表している。同図(a)は実行遅延を行わないVLIWの実行形態を表しており、同図(b)は実行遅延に基づいた実行形態を表している。実行遅延を行う場合は、指定されたオペレーションが、フェッチ、デコード後、指定されたサイクル（同図では2サイクル）だけ遅延されてから実行される。このような実行遅延の導入によって、プログラム（フェッチ）形態と実行形態を異なるものにすることが可能となる。

図3はVLIWフェッチのタイミング（同図(a)）及び実行のタイミング（同図(b)）を示している。プログラムのA,Bなどはオペレーションを表し、その後につけられた(+1)が、そのオペレーションの実行を1サイクルだけ遅延させることを示している。オペレーションBはオペレーションAと同時にフェッチされたが、1サイクル実行遅延されたために、次サイクルでフェッチされたオペレーションC,Dと同時に実行される。オペレーションFも同様である。

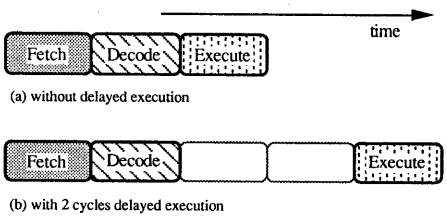

図2　実行遅延の概念

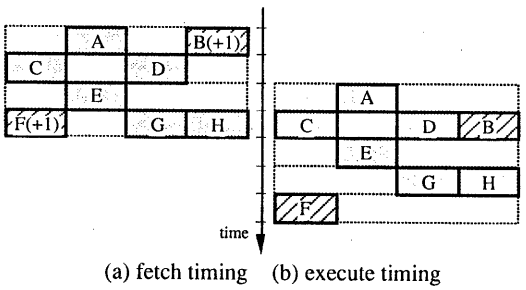

図3　VLIWの再構成

3．実行遅延の効果

3.1 ループのオーバラップ実行

図4(a)は、従来のVLIW型計算機用のループを含むサブルーチンである。LOADB/STOREBはバイト

データのロード／ストアを表す。ループ本体は3つのVLIWから構成されている。このプログラムを、実行遅延に基づいて書き直すと同図(b)のようになる。同図(a)の上から4語めの ADD 1、R1 というオペレーションが、図4(b)では上から2語めに移動させられ、2サイクルの実行遅延の指定が付加されている。これにより遅延を指定する前はVLIW 3語より構成されていたループ本体が、2語で構成されるようになっている。

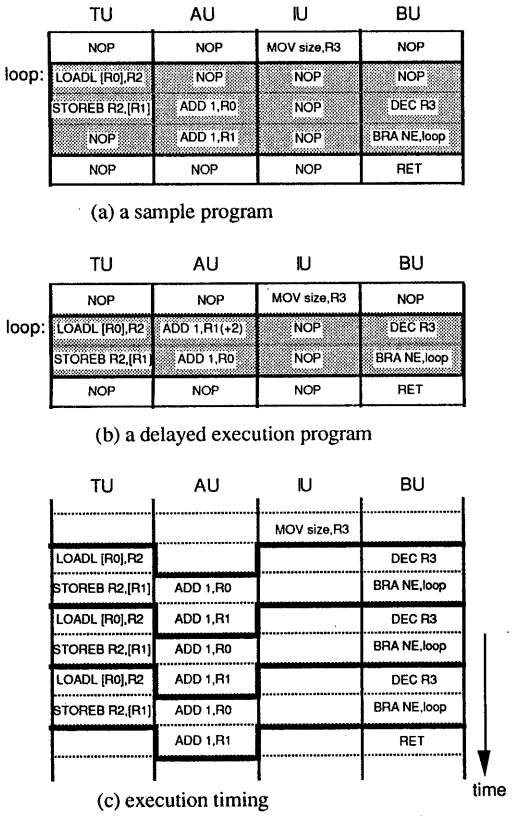

図4　ループのオーバラップ実行

このプログラムを実行したときのタイミングを同図(c)に示す。太線で囲まれた部分がループの各インスタンスである。各ループインスタンスはその前後のインスタンスとオーバラップされており、しかも2サイクルで実行されている。この例では、本方式の導入により、ループ本体の実行時間が従来の2/3になり、また、プログラムの長さも5から4に減少していることが示されている。

3.2 サブルーチンのオーバラップ実行

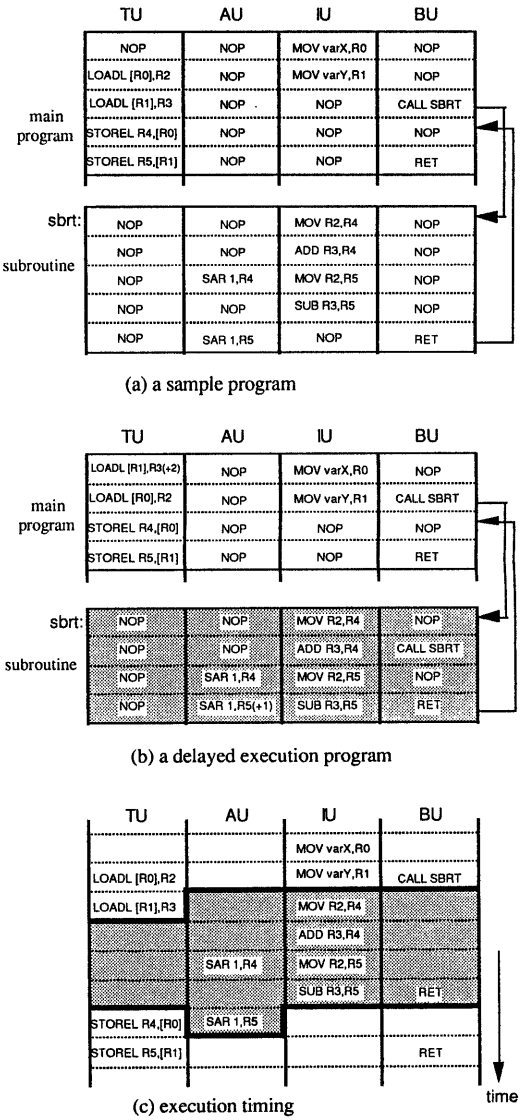

図5　サブルーチンのオーバラップ実行

サブルーチン本体とコール側を静的に解析することにより、サブルーチンコール／リターン時にもオーバラップ実行が可能となる。図5は、

R4 = (R2+R3) / 2

R5 = (R2-R3) / 2

を計算するサブルーチンを含むプログラムである。SARは右算術シフトを表す。同図(a)が実行遅延を導入する前のプログラム、同図(b)が実行遅延を導入してオーバラップ実行を可能としたプログラムである。同図(a)ではメインプログラムの3語めのVLIWがサブルーチンをコールしているが、(b)では2語めのVLIWがサブルーチンコールを行っている。メインプログラムとサブルーチンプログラムで1個ずつ実行遅延が指定されている。プログラム長は10語が8語に減少している。実行遅延に基づいたプログラムは同図(c)のように実行される。実行サイクル数が10から8に減少していることがわかる。

4．ハードウェアの基本構成

オペレーションの実行遅延を可能にするVLIW型計算機の基本構成を図6に示す。同図には最高3サイクルの実行遅延が可能な計算機を示している。T0-T2、A0-A2、I0-I2、B0-B2がオペレーション実行を遅延するために新たに導入された遅延用のレジスタであり、それぞれ、データ転送オペレーション用、レジスタ操作オペレーション用、整数演算オペレーション用、分岐オペレーション用である。TUやAUなどは各処理に対応する実行ユニットでありマルチポート・レジスタファイルを共有する。

VLIWはフェッチされた後、デコーダに送られてオペレーションの種類と実行遅延サイクル数が識別される。そして、実行遅延の指定がある場合には該当する遅延レジスタへ送られ、指定がない場合には実行ユニットに直接送られる。遅延レジスタは各実行ユニットごとにシフトレジスタを構成しており、クロックパルスに同期して内容が実行ユニットに向かって送られる。たとえば、

LOADL [R0]、 R2 (+2)

というオペレーションの場合には、デコードされた後、遅延レジスタT1に送られ、T0を経由して2サイクル経過してからTUで実行される。

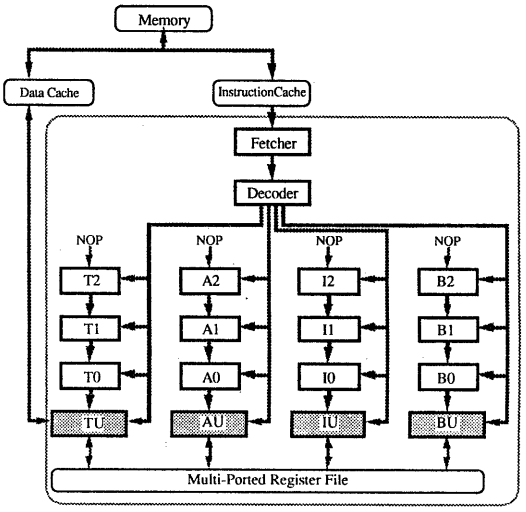

図6　再構成VLIW型計算機の基本構成

5．シミュレーション評価

本方式に基づくVLIW型計算機をシミュレーションにより評価した。対象とした計算機は図6に示したように構成されており、1命令1サイクルで実行するものとしている。サンプルプログラムは、バブルソート、フィボナッチ数計算、行列計算、階乗計算、最大値最小値計算、2進10進変換、マージ(併合)の7個である。

表1にサンプルプログラムを実行した結果を示す。同表には、左から順に、サンプルプログラム名、プログラムを構成するオペレーション数、実行したオペレーション数、実行遅延を行わない場合の実行時間、実行遅延を行った場合の実行時間、速度向上率、オーバラップされたプログラム部分を表している。Lはループでオーバラップがなされたことを表し、Sはサブルーチンのコール／リターンでオーバラップがなされたことを示している。最適化は基本ブロックの境界部でレジスタの依存関係を見ながら可能なところまで1ステップずつオーバラップさせていくという、比較的シンプルな方針に基づいて行った。

表1　シミュレーション結果

Program name	Number of operations	Number of executed operations	Execution time - Not Overlapped	Execution time Overlapped	Speed up	Loop or Subroutine
bubble	17	569	344	254	1.35	L
fibonacci	20	412	236	149	1.58	L
matrix	37	1874	1362	1266	1.08	L
factorial	10	164	123	114	1.08	S
minmax	13	165	104	84	1.24	L
decimal	28	212	158	144	1.10	L
merge	55	43997	27371	22255	1.23	S+L

速度向上率が最も高いのはフィボナッチ数計算で58%で、最も低いのは行列計算と階乗計算の8%である。フィボナッチ数計算では、実行遅延を行わなかったときはループ部の実行に8サイクル必要だったのが、オーバラップにより5サイクルに減少したため、このような高速化が達成された。行列演算では、共有部を持つ3つのループのうち2つのループでそれぞれ1サイクルと2サイクル減少している。また、ループインスタンスのオーバラップ実行のほうが、サブルーチンのコール／リターンのオーバラップ実行より効果が大きいことが同表から推測される。

さらに、表1の中でループについてのみオーバラップ化をしたプログラムに関して、従来のソフトウェア・パイプライニングを用いた場合の命令数との比較結果を表2に示す。同表には、5個のプログラムについてオペレーション数と命令（VLIW）数が示されている。最適化はソフトウェア・パイプラインニングについても同じように行っているのでループの実行形態は同様であるが、本方式ではループ前後の調整用コードを除去できるので、オペレーション数、VLIW数とも、20%から76%程度に減少していることが同表よりわかる。matrixではループが3重になっているので特にその効果が大きい。実行遅延サイクル数を指定するための各フィールドの数ビット（本評価では2ビット）の増加を考慮してもメモリやバスの使用効率は大きく向上すると考えられる。

表2　ソフトウェア・パイプライニングとの比較

	Number of instruction words			Number of operations		
Program name	Software pipelining[a]	Delyed execution[b]	Rate ([b]/[a])	Software pipelining[c]	Delyed execution[d]	Rate ([d]/[c])
bubble	15	10	0.67	28	17	0.61
fibonacci	16	8	0.5	33	20	0.61
matrix	113	23	0.20	189	37	0.20
minmax	12	8	0.67	21	13	0.62
decimal	21	16	0.76	37	28	0.76

6．おわりに

従来のVLIW型計算機に対して、シフトレジスタを各実行ユニットの前段に付加し、オペレーションに数ビットの遅延サイクル数を指定するフィールドを付加する程度の拡張を施すことにより、実行時にVLIWを構成し直す方式を提案した。また、本方式の導入によりループやサブルーチンの実行が高速化できることをシミュレーションにより示した。

本論文では詳細には触れなかったが、本方式はNOPの直接的な除去に応用することも可能である[6]。たとえば、各VLIWを2つのオペレーションから構成されるとしてメモリに格納しておき遅延レジスタで4オペレーションに展開することができる。このようなNOPの除去や本論文で述べたループのオーバラップ実行時におけるコード量の減少は、コード量が大きくなりがちなVLIW型計算機にとって、メモリなどのコストを削減するだけでなく、キャッシュ搭載時にはヒット率の向上につながるという点で、特に実用的な見地において重要な意義があると思われる。

本論文で提案した方式は、コンパイラが完全に実行タイミングを制御するようなピュアなVLIW型計算機モデルを維持しているところに特長がある。た

だし、オペレーション間の動的な衝突の解消やフェッチしたコードの再利用などのために、遅延レジスタを動的に使用することを想定することも可能である。このことに関しては別稿で報告する予定である。

参 考 文 献

(1) J.A. Fisher、"Very Long Instruction Word Architecture and the ELI-512"、Proc. 10th International Symposium on Computer Architecture、pp. 140-150、1983.

(2) J.R. Ellis、"Bulldog: A Compiler for VLIW Architectures"、The MIT Press、1986.

(3) M.D. Smith、M.S. Lam and M.A. Horowitz、"Boosting Beyond Static Scheduling in a Superscalar Processor"、Proc. 17th Annual Symposium on Computer Architecture、pp. 344-354、1990.

(4) M. Lam、"Software Pipelining: An Effective Scheduling Technique for VLIW Machines"、Proc. ACM SIGPLAN '88 Conf. on Programming Language Design and Implementation"、pp. 318-128、1988.

(5) R.P. Colwell、R.P. Nix、J.J. O'Donnell、D.B. Papworth and P.K. Rodman、"A VLIW Architecture for a Trace Scheduling Compiler"、IEEE Trans. on Computers、Vol. 37、No. 8、pp. 967-979、1988.

(6) 加藤工明、" VLIWプロセッサの高効率化に関する研究" 、平成元年度名古屋工業大学電気情報工学科卒業論文、1990.

(7) 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、1991 (採録決定)。

(8) A. Nicaolau and J.A. Fisher、"Mesuring the Parallelism Available for Very Long Instruction Word Architectures"、IEEE Trans. on Computers、Vol. C-33、pp. 968-976、1984.