計算機アーキテクチャ　102－3
（1993．10．21）

マルチスレッド処理をサポートする VLIWプロセッサ・アーキテクチャ

國領 琢也†　富田 眞治‡
†: 富士通 (株)　‡: 京都大学工学部

VLIW 方式による命令レベル並列処理と複数命令ストリーム実行 (マルチスレッド処理) による粗粒度並列処理を行うプロセッサ・アーキテクチャを提案する。本プロセッサでは並列性の抽出 (VLIW) と完全なパイプライン・スケジューリングをコンパイラが行うことにより、ハードウェアのオーバヘッドを小さく抑えている。VLIW 命令には命令発行遅延サイクル数を指定するフィールドが付加されており、各スレッドにおいて命令間の依存関係を示す情報として用いられる。実行スレッドにおいて依存関係を検出した場合は遅延時間なしに実行スレッドを切り替え、パイプライン・ハザードを回避する。また、内蔵キャッシュのミスヒットにおいても動的に実行スレッドを切り替え、メモリアクセス待ちによる実行遅延を軽減する。

本稿ではプロセッサ・アーキテクチャの概要と動作について説明した後、シミュレーションによりマルチスレッド処理の効果を測定する。

A Multi-Threaded VLIW Processor Architecture

Takuya Kokuryo†　and　Shinji Tomita‡

†: Fujitsu Limited
‡: Kyoto University

Email: kokuryo@prd.cs.fujitsu.co.jp

A VLIW processor architecture which supports the processing of multiple instruction threads is proposed. In this processor architecture, the compiler extracts instruction-level parallelism and applies complete software pipeline-scheduling technique, therefore the hardware overhead is reduced. In this processor, each instruction has a field that specifies instruction issue delay and is used as information for dependencies between instructions. Pipeline hazards due to instruction dependencies are prevented by switching to another thread. In case of cache miss, execution thread is dynamically switched to another thread. This paper describes the outline and the operation of the processor architecture, and the effect of multi-threading estimated by simulation.

1 はじめに

近年の高性能マイクロプロセッサは、動作周波数が飛躍的に高速になると共に、命令レベルの並列処理を行うことにより性能向上を図っている。これらは、パイプラインピッチを短縮するためのスーパパイプライン方式に加えて、命令の並列実行のためにスーパスカラ方式やVLIW方式を導入することにより実現している。

しかしながら、命令間の依存関係や階層メモリのアクセスにより命令の並列実行が阻害され、プロセッサが持つ本来の並列度を引き出すことができず、現状のアーキテクチャでは大幅な性能向上を達成することができない[4][7]。

この問題を解消する手段の一つにマルチスレッド処理方式がある。これは、単一プロセッサにおいて複数の命令ストリーム(スレッド)を実行することにより、パイプライン・ハザードを回避し、パイプラインのスループットを向上する方法である。

本稿では命令レベル並列処理と共にマルチスレッド処理を行うプロセッサ・アーキテクチャについて提案を行う。まず、次章において命令レベル並列処理とマルチスレッド処理を融合したアーキテクチャについてまとめた後、今回提案するプロセッサ・アーキテクチャについて説明する。

2 従来のアーキテクチャ

2.1 マルチスレッド処理の概要

現在のスーパスカラ方式やVLIW方式のプロセッサは単一命令ストリーム(シングルスレッド)から命令の並列性抽出を行っている。このため、命令間の依存関係の存在やメモリアクセスのレイテンシにより、引き出せる並列度には限界がある。近年、これらの問題点を解消するために、マルチスレッド処理方式を導入したアーキテクチャの研究が盛んに行われている[1][2][5][6][8]。これらのプロセッサは、スレッドごとに独立した命令制御を可能にするため、各スレッドに対応したプログラム・カウンタと各種状態レジスタを持ち、内蔵の機能ユニットを複数のスレッドにより共有する形態をとる。これにより、命令発行の抑止されたスレッドに代わって、別のスレッドから命令発行を行うことにより、パイプラインのスループット向上を図ることが可能である。

マルチスレッド処理では、粗粒度並列を活用をしたスレッド単位の並列実行を行うことが可能である。したがって、命令レベルの並列処理とマルチスレッド処理を組み合わせることにより、従来の命令レベルの並列処理だけでは得られなかった大きな並列度を引き出すことが可能である。

命令レベルの並列処理とマルチスレッド処理を融合させたプロセッサには以下の方式がある。

- マルチスレッド型スーパスカラ方式[5][6]
- マルチスレッド型VLIW方式[1][2]
- マルチスレッド型スーパパイプライン方式[3]

以下にそれぞれの方式についてまとめる。

2.2 マルチスレッド型スーパスカラ方式

スーパスカラ方式のプロセッサは多数の独立した機能ユニットを搭載し、1サイクル当たりに複数の機能ユニットへ命令を発行することにより並列実行を行う。一般的に複数の機能ユニットは、常時100%の稼働状態ではないため、機能ユニットを共有した複数スレッドの並列実行により、機能ユニットの稼働率を高めることが可能である。

しかしながら、実行時に命令の並列性を抽出するために、命令間の依存関係を全て検出する必要がある。さらに、スーパスカラ方式においてマルチスレッド処理を行う場合には、各スレッドごとにこれら依存関係の検出を行った上で、複数のスレッドに対して利用資源(機能ユニット)の調停を行う必要がある。

このように、スーパスカラ方式ではマルチスレッド処理に必要なハードウェアは増大するため、クロック速度の向上を妨げる可能性がある。また、複数スレッドの中から並列実行可能な複数命令を同時発行する場合、命令フェッチバンド幅は、命令長×スーパスカラ多重度×スレッド数となるため、スレッドごとに独立した命令キャッシュを用意する必要がある。

2.3 マルチスレッド型VLIW方式

VLIW方式では、並列実行可能なオペレーションの集まりである長形式の命令(VLIW命令)を対

応する機能ユニットにおいて並列実行する。

VLIW プロセッサのマルチスレッド処理では、あるスレッドの VLIW 命令中のオペレーションから使用しない (NOP 指示の) 演算器を他のスレッドの VLIW 命令のオペレーション実行に割り当てる。これにより、スーパスカラ方式と同様に、機能ユニットの稼働率を高めることが可能である。

VLIW 方式においてマルチスレッド処理を行うためには複数のスレッドに対して利用資源 (機能ユニット) の調停を行う必要があるが、VLIW 命令の各オペレーションは対応する機能ユニットに直結しているため、これらの制御は容易である。新たなハードウェアとして、VLIW 命令をオペレーションごとに分離して命令発行・命令保持する機構がスレッドごとに必要となるが、スーパスカラ・プロセッサの命令ウィンドウに比べれば単純な構造である。

以上のことから、スーパスカラ方式に比べて命令発行に関するハードウェアが簡略化されることがわかる。しかしながら、複数スレッドの VLIW 命令を同時発行する場合、スーパスカラ方式と同様にスレッドごとに独立した命令キャッシュを用意する必要がある。

2.4 マルチスレッド型スーパパイプライン方式

スーパパイプライン方式は、パイプラインの処理単位を細分化し、処理速度と時間方向の並列度を高める方式である。パイプラインを多段化するため、一般のパイプラインよりも命令間の依存関係に対するペナルティが大きくなるが、マルチスレッド処理を導入することにより、パイプラインのスループットを大幅に向上することが可能である。しかしながら、パイプライン処理段数の増加に伴って、パイプライン処理遅延を隠蔽するために多くのスレッドを必要とする。

この方式は 1 サイクル当たりの発行命令数が 1 命令であるため、各スレッドの命令フェッチ処理において命令キャッシュから一度に複数命令をプリフェッチしておくことができる。このため、複数のスレッドにより命令キャッシュを共有しても全てのスレッドに対して均一な命令供給を実現することが可能である。

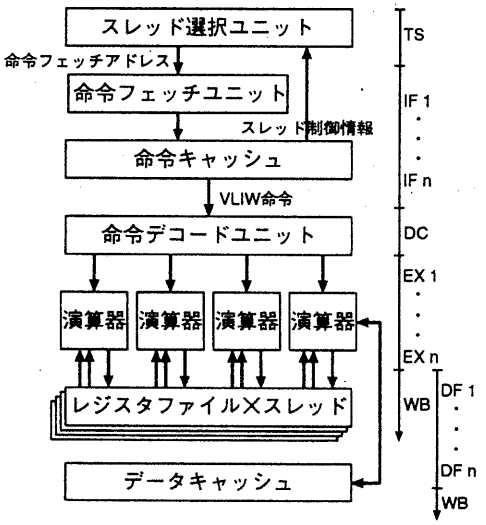

図 1: プロセッサのハードウェア構成

3 アーキテクチャの概要

本章では前章にて述べたマルチスレッド型 VLIW 方式よりもオーバヘッドが少なく、スーパパイプライン構造に適したマルチスレッド処理方式について述べる。本アーキテクチャの主な特徴は次の通りであり、以降の節において詳細を述べる。

1. VLIW 命令パイプライン
   - スーパパイプライン構造 (3.1節)
   - 非インタロック・パイプライン (3.2節)
2. VLIW 命令フォーマット (3.2節)
   - 命令発行遅延サイクル数の導入
   - VLIW 命令単位の NOP 命令の圧縮
3. マルチスレッド処理 (3.3節)
   - 命令情報による静的スレッド切り替え
   - 実行要因による動的スレッド切り替え

3.1 ハードウェア構成

プロセッサのハードウェア構成を図 1に示す。命令フェッチユニット、命令デコードユニット、および各種の機能ユニットは複数のスレッドにより共有されており、前章にて述べたマルチスレッド型スーパスカラ方式・VLIW 方式とは異なり、スレッドごとに命令フェッチユニットや命令デコードユ

ニットを搭載しない。また、レジスタファイル、プログラムカウンタ、および各種制御レジスタはスレッドごとに持つ。プロセッサはスレッドに対応するプログラムカウンタに従って、VLIW 方式により単一スレッドの並列実行を行う。命令間の依存関係により、ある実行スレッドの命令発行を停止する場合には、クロックサイクル単位で異なるスレッドの VLIW 命令を発行し、複数のスレッドの命令を同一パイプラインにおいて並列実行を行うことができる。

命令パイプラインは、図 1の右側に示すように、スレッド選択 (TS) →命令フェッチ (IFn) →命令デコード (DC) →命令実行 (EXn,DFn,WB) という処理ステージから構成される。また、命令フェッチ (命令キャッシュ) 、命令実行 (演算器) 、オペランドフェッチ (データキャッシュ) は細分化され、それぞれスーパパイプライン構造となっている。

スレッド選択ユニットは全てのスレッドの中から命令発行抑止状態でないスレッドを選択して、命令フェッチユニットに対して命令フェッチ要求を出力する。命令発行可能なスレッドが複数ある場合は、あらかじめ設定した優先度に従ってスレッドを選択する。このユニットは全てのスレッドのプログラムカウンタ (PC) を管理しており、命令フェッチ要求を出力するごとに対応するスレッドの PC を更新する。また、分岐命令が実行される場合は、分岐ユニットから PC の更新値がスレッド選択ユニットに通知される。命令フェッチユニットは 1 サイクルに 1 つのスレッドの VLIW 命令をフェッチし、命令デコードユニットに直接転送される。命令デコードユニットは VLIW 命令をデコードし、スレッドに対応するレジスタファイルからオペランドを読み込んだ後、全オペレーションが対応する機能ユニットに対して発行される。機能ユニットの演算結果はスレッドに対応するレジスタファイルに書き込まれる。

3.2 パイプライン・スケジューリング

プロセッサは VLIW 方式のため、命令の並列性抽出をコンパイル時に行う。それに加えて、ハードウェアはパイプライン・インタロック機構を持たないため、コンパイラがパイプラインのスケジューリングを完全に行う必要がある。VLIW 命令の各

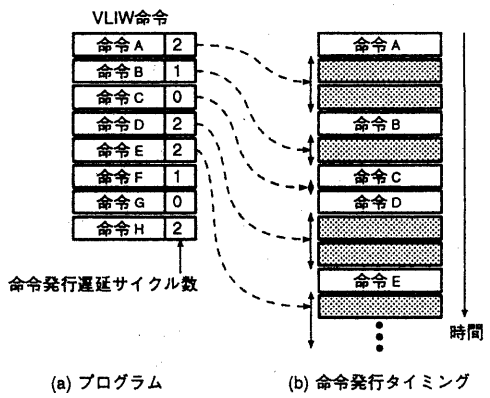

図 2: プログラムの命令発行タイミング

オペレーションは、パイプライン段数の異なる機能ユニットにおいて実行されるが、オペレーション間の実行完了の同期はとらない。つまり、VLIW 命令の各オペレーションは同時 (in-order) に命令実行が開始されるが、命令実行の完了は out-of-order となる。したがって、コンパイラがこれらの処理を考慮してスケジューリングを行う。

一般的に、このような静的スケジューリングのプロセッサでは、依存関係を保証するために命令間に NOP 命令を挿入する必要がある。これが、スーパパイプライン方式になれば NOP 命令の増加は顕著になる。しかしながら、本アーキテクチャでは VLIW 命令単位での命令圧縮を行うことができる。これは、コンパイル時に命令間の依存関係を検出した場合、次命令の発行を遅らせるサイクル数 (命令発行遅延サイクル) を VLIW 命令のフィールド (4bit) に格納しておき、実行時にその値に従って命令発行を制御することにより不要な NOP 命令を削除できる。図 2に本アーキテクチャにおける命令の発行タイミング例を示す。この例では単純化のため命令フェッチの遅延は考慮していない。したがって、VLIW 命令中に指定した遅延サイクルは次命令に対する値を示している。

3.3 スレッド切り替え制御方式

各スレッドの命令発行制御と実行スレッドの切り替えは、前述のスレッド選択ユニットにおいて行う。図 3にスレッド選択ユニットの構成を示す。こ

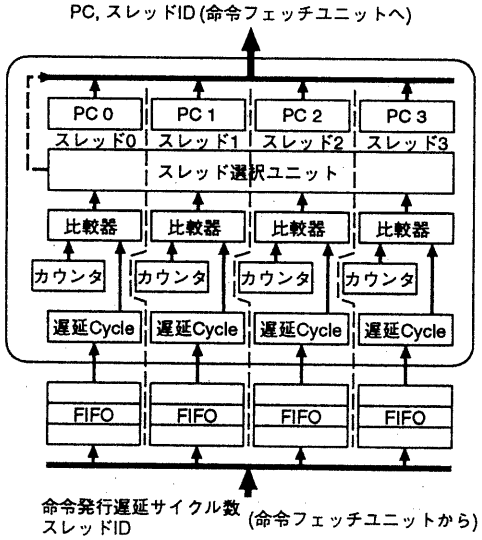

図 3: スレッド選択ユニットの構成

の構成図は、命令フェッチに4サイクルの遅延[1]があることを考慮している。命令発行遅延サイクルを遅延することなくスレッド選択ユニットに供給するためには、対象命令の4サイクル前 (4命令前) の命令に遅延サイクルを指定しておく必要がある。したがって、先見された遅延サイクルを格納するために、各スレッドに対して4命令分のFIFOバッファを用意している。

スレッド選択ユニットは命令フェッチユニットから受け取った遅延サイクルをスレッドに対応したFIFOバッファに格納する。格納された遅延サイクルはさらに3命令後には比較器と接続された遅延サイクル・バッファに到達し、サイクルカウント (カウンタ) 値と比較される。カウント値は1サイクルごとにカウントアップし、カウント値と遅延サイクルの値が一致した時点でそのスレッドの命令発行抑止が解除され、対応するスレッドの命令発行 (パイプラインへの投入) と共にカウント値はリセットされる。同時に、FIFOバッファから新たな遅延サイクルを遅延サイクル・バッファに格納して、新たに遅延サイクルのカウントを行う。

スレッド選択ユニットでは以下の要因に対してスレッドの命令発行が抑止状態となり、命令発行可能な別のスレッドの命令フェッチが開始される。

1. VLIW命令の命令発行遅延サイクル数 (>0) が指示されている場合
2. ロード／ストア命令において内蔵キャッシュのミスヒットが発生した場合

1の要因は、コンパイル時にデータ依存や制御依存を検出した結果であり、スレッド選択ユニットは命令発行を規定サイクル期間抑止する。

2の要因は、ロード／ストア命令のキャッシュミスヒットをプロセッサが実行時に検出するため、実行スレッドも動的に変更される。ロード／ストア命令に依存する後続命令はコンパイル時にパイプライン・スケジューリングされているので、パイプライン処理中のデコードステージ以降の命令はディレイスロットとしてキャンセルすることなく実行することができる。また、命令フェッチ中の命令は依存関係を含んでいる可能性があるため、命令デコードユニットへは供給されずに命令キューに一時的に退避され、キャッシュミスヒット処理後に実行される。

スレッド選択ユニットがサポートするスレッドの選択アルゴリズムは2種類ある。1つは、あるスレッドの命令発行が命令発行遅延サイクル数 (>0) の指定により抑止された時点から次のスレッドの命令発行に切り替えられる。したがって、実行スレッドは順番に切り替えられ、全てのスレッドの間をほぼ均一に巡回する。もう一方は優先度の高いスレッドがパイプライン・スケジューリング通りに実行され、他のスレッドの命令発行は優先度の高いスレッドの命令発行抑止期間だけ行われる。

4 命令スケジューリングと動作

4.1 VLIW命令のスケジューリング例

リバモア・カーネル No.3(内積) におけるループ部分のVLIW命令スケジューリング例を図4に示す。1VLIW命令当たり4つのオペレーション・フィールドの他に命令発行遅延サイクル・フィールドが指定される。この例では、次命令を遅延なしで発行できる場合に'0'が指定され、データ依存

[1]図1においてスレッド選択 (TS) とキャッシュ・アクセス3ステージ (IFn) の計4ステージ分の遅延を仮定している。

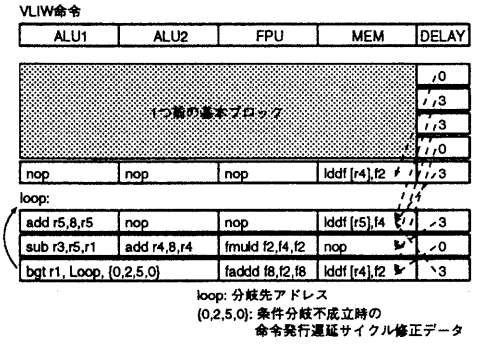

図 4: カーネル No.3 のスケジューリング例

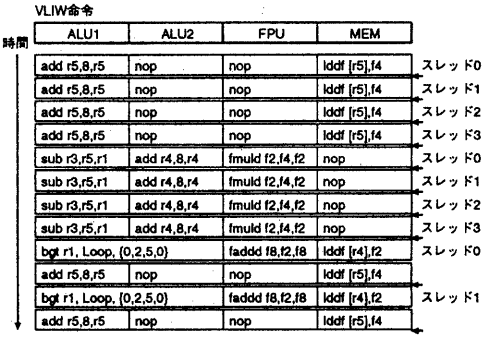

図 5: カーネル No.3 の動作タイミング

により 4 サイクル後に命令を発行できる場合に'3'が指定されている。このフィールドは命令フェッチの遅延[2]を考慮して、4 命令後の命令に対する遅延サイクルを先見して指定している。図中の点線矢印は、指定された遅延サイクルの対象命令を示している。プログラム中に条件分岐命令がある場合は、分岐が成立すると仮定して分岐先の命令に対する遅延サイクルを指定している。このため、分岐が成立しなかった場合は、条件分岐命令中に指定された非分岐側 4 命令分の遅延サイクルを用いて先見された遅延サイクル (図 3の FIFO バッファのデータ) を修正する。

4.2 マルチスレッド処理の動作

図 5は図 4のプログラムを 4 スレッドにより実行した場合の動作タイミングを示している。プログラムは、4 スレッド間において共有されている。また、データ領域については、ループの直前にスレッドごとに異なったアドレス計算を行い、お互いに異なったデータ領域をアクセスすると仮定する。

スレッドの選択アルゴリズムはラウンドロビン方式により命令発行が抑止された時点において順番にスレッドを切り替えている。図の右側の矢印は実行スレッドが切り替わるポイントを示している。この例では命令発行遅延サイクルの先見指定により命令フェッチの遅延を意識することなく実行スレッドの切り替えを可能にし、パイプラインの高スループットを実現できていることがわかる。

[2]図 1においてスレッド選択 (TS) とキャッシュ・アクセス 3 ステージ (IFn) の計 4 ステージ分の遅延を仮定している。

機能ユニット	実行／反復サイクル数	個数
整数演算	1 ／ 1	2
ロード/ストア	4 ／ 1	1
浮動小数点加算／乗算	4 ／ 1	1

表 1: 機能ユニットの実行サイクル数

5 性能評価

5.1 シミュレーションモデル

ソフトウェア・シミュレーションにより、本プロセッサの性能評価を行った。命令パイプラインは図 1において、命令／データキャッシュ・アクセスがそれぞれ 3 ステージ構成[3](n = 3) の合計 10 ステージから構成される。機能ユニットの実行サイクル数は表 1に示す通りであり、毎サイクル新しい命令を実行できる構成となっている。今回の評価では、表 1に示す機能ユニット構成について、マルチスレッド処理 (4 スレッド) を行った場合と行わなかった場合の性能比較を行った。いずれも、VLIW 命令は 1 サイクル当たり 4 つのオペレーションを発行可能としている。命令キャッシュのヒット率は 100%とし、データキャッシュのヒット率は特に断わりのない場合は 100%としている。

評価プログラムには、倍精度のリバモア・カーネルから No.2(不完全コレスキー分解) 、No.3(内積) の 2 つとクイックソートを使用した。リバモア・カーネルはコンパイル段階にてループのイタレーションを分割し、各スレッドに割り当てた。ま

[3]ロード／ストア命令はアドレス計算 (EX1) とキャッシュ・アクセス 3 ステージ (DFn) の計 4 ステージを必要とする。

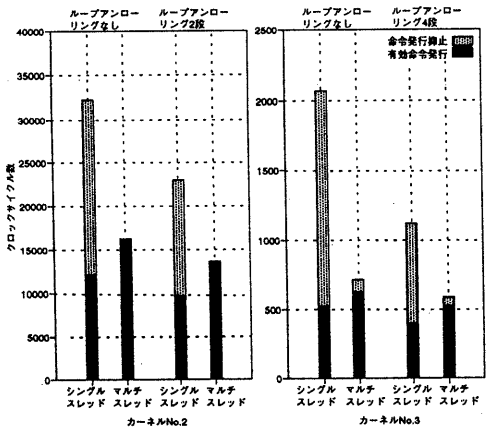

図 6: リバモア・カーネルの実行サイクル数

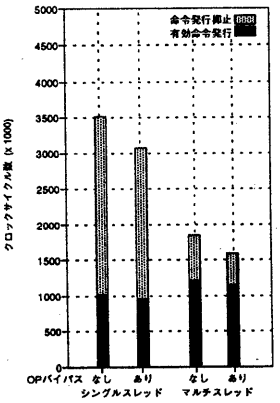

図 7: クイックソートの実行サイクル数

た、スレッド間のバリア同期はメモリデータ (データキャッシュ) を用いて行った。ループ・アンローリングについては、プログラムの性質を考慮して、カーネルNo.2が2イタレーション、カーネルNo.3が4イタレーションの展開を行っている。クイックソートは、400個以上からなるストリングに対して比較・複写の処理を行うプログラムであり、再帰呼び出し時に新規スレッドの生成を行いながら、複数スレッドの実行により並列にソーティングを行う。生成するスレッドの数は、最大4つまでとしている。

5.2 シミュレーション結果

図6〜図8は、シングルスレッドの場合と4スレッドの場合の性能比較を行っている。

リバモア・カーネルNo.2(図6左) のマルチスレッド処理では、プログラムの内側ループの終了ごとに同期操作を行っているため、全体のプログラムの実行量 (有効命令発行数) は増加している。しかしながら、ループ・アンローリングを行ったシングルスレッドの場合よりもマルチスレッド処理を行った方が性能向上率が高いことを示している。アンローリングなしのシングルスレッドと比較すると、アンローリングなしのマルチスレッドでは約2.0倍の性能向上を、アンローリングありのマルチスレッドでは約2.4倍の性能向上を達成している。

リバモア・カーネルNo.3(図6右) のマルチスレッド処理では、プログラムの最後にスレッド間の同期操作と部分和の加算処理を行っているため、シングルスレッドの場合と比較してプログラムの実行量は若干増加している。しかしながら、アンローリングなしのシングルスレッドと比較すると、アンローリングなしのマルチスレッドでは約2.9倍の性能向上を、アンローリングありのマルチスレッドでは約3.5倍の性能向上を達成している。

並列クイックソート (図7) のマルチスレッド処理では、再帰呼び出し (新規スレッド生成) を行うまではただ1つのスレッドしか動作していないため、パイプラインの使用効率をリバモアカーネル並に高めることが困難である。しかしながら、最終的に4つのスレッドが並列に動作し、約1.9倍の性能を達成している。整数演算ユニットのリザルト・オペランド・バイパスを行っても、約2.2倍と性能向上の割合が少ないのはプログラムのストリング処理 (ロード／ストア) に費やす時間が多いためであった。

図8はリバモア・カーネルNo.3のデータ・キャッシュのミスヒットにおける実行時間の影響について示す。ミスヒット時のペナルティサイクル数は外部メモリアクセスに4サイクル要した時に、内部キャッシュへの書き込みに必要なサイクルを含めて7サイクルとしている。また、外部メモリアクセスに12サイクル要した時に、ペナルティサイクル数は15サイクルとしている。いずれも、デー

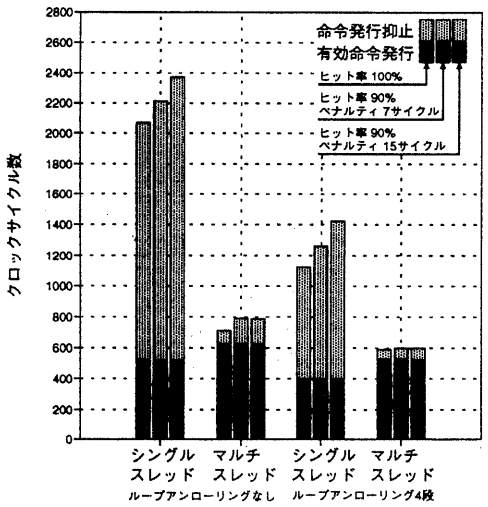

図 8: 内蔵キャッシュミスヒットの効果

タ・キャッシュのヒット率が約 90%の場合についてシミュレーションしている。シングルスレッドではミスヒットのペナルティに比例して総実行サイクル数が増加しているが、マルチスレッドではミスヒット時にスレッドを切り替え、性能低下を軽減している。さらに、ループ・アンローリングを行う場合は、4 つのスレッドによりメモリアクセスの遅延を完全に隠蔽でき、性能の低下はほとんど見られない。

6 おわりに

従来の VLIW 方式による命令レベル並列処理に加えて、マルチスレッド処理によりクロックサイクル単位に実行スレッドを変更できるプロセッサ・アーキテクチャを提案した。本アーキテクチャでは、本来の簡略な VLIW 方式の利点を損なわずにパイプラインのスループットを向上するマルチスレッド処理を組み合わせた。

今後は、VLIW 命令フォーマットの詳細な検討と共に、本アーキテクチャに最適なコンパイル手法を確立する必要がある。

謝辞

本研究に関して日頃より御討論頂く京都大学工学部・富田研究室の諸氏に感謝致します。

参考文献

[1] R. Guru Prasadh and Chuan-lin Wu: A Benchmark Evaluation of a Multi-Threaded RISC Processor Architecture, In *Proc. of the 20th Int'l Conf. on Parallel Processing Vol.1*, pp.84-91, 1991.

[2] Stephen W. Keckler and Willian J. Dally: Processor Coupling: Integrating Compile Time and Runtime Scheduling for Parallelism, In *Proc. of the 19th Annual Int'l Symp. on Computer Architecture*, pp.202-213, 1992.

[3] Daniel C. McCrackin: Eliminating Interlocks in Deeply Pipelined Processors by Delay Enforced Multistreaming, *IEEE Transactions on Computers*, Vol.40, No.10, pp.1125-1132, Oct 1991.

[4] Norman P. Jouppi and David W. Wall: Available Instruction-Level Parallelism for Superscalar and Superpipelined Machines, In *Proc. of the Third Int'l Conf. on ASPLOS*, pp.272-282, 1989.

[5] George E. Daddis Jr. and H.C. Torng: The Concurrent Execution of Multiple Instruction Streams on Superscaler Processors, In *Proc. of the 20th Int'l Conf. on Parallel Processing Vol.1*, pp.76-83, 1991.

[6] Won Woo Park, Donald S. Fussell and Roy M. Jenevein: Performance Advantages of Multithreaded Processors, In *Proc. of the 20th Int'l Conf. on Parallel Processing Vol.1*, pp.97-101, 1991.

[7] Gurindar S. Sohi and Sriram Vajapeyam: Tradeoffs in Instruction Format Design for Horizontal Architectures, In *Proc. of the Third Int'l Conf. on ASPLOS*, pp.15-25, 1989.

[8] Hiroaki Hirata, Kozo Kimura, Satoshi Nagamine, Yoshiyuki Mochizuki, Akio Nishimura, Yoshimori Nakase, and Teiji Nishizawa: An Elementary Processor Architecture with Simultaneous Instruction Issuing from Multiple Threads, In *Proc. of the 19th Annual Int'l Symp. on Computer Architecture*, pp.136-145, 1992.