計算機アーキテクチャ　94－2
（１９９２．６．１２）

多重制御フロー機構を備えた資源共有型プロセッサ・アーキテクチャ

平田博章、木村浩三、永峰聡、望月義幸、
西村明夫、中瀬義盛、西澤貞次

松下電器産業(株)　メディア研究所

本稿では、プロセッサの機能ユニットの使用率を改善するために多重スレッド技術を用いる要素プロセッサ・アーキテクチャについて述べる。本プロセッサは複数の命令流（スレッド）を並列実行するための多重制御フロー機構を有し、複数のヘテロジニアスな機能ユニットに対して、複数の命令流からの命令を並列発行する。機能ユニットに対する命令間のコンフリクトは動的に調停する。異なる命令流間では命令の依存関係が存在しないため、プロセッサのスループットを飛躍的に向上させることができる。シミュレーションによる評価では、11個の機能ユニットを備えたプロセッサで、並列実行スレッド数に比例した性能向上が確認できた。なお、ここで述べた多重スレッド技術をスーパスカラ方式と組み合わせて使用した場合についても評価し、価格性能比の観点からその効果について議論する。

A Resource-Shared Processor Architecture with a Multiple Control-Flow Mechanism

Hiroaki Hirata, Kozo Kimura, Satoshi Nagamine, Yoshiyuki Mochizuki,
Akio Nishimura, Yoshimori Nakase and Teiji Nishizawa

Media Research Laboratory,
Matsushita Electric Industrial Co., Ltd.
1006 Kadoma Osaka.　571 Japan

In this paper, we present a multithreaded processor architecture oriented for use as a base processor of multiprocessor systems. A processor implementing our architecture is facilitated with a multiple control-flow mechanism which enables instructions from different threads to be issued simultaneously to multiple heterogeneous functional units. These issued instructions begin execution unless there are functional unit conflicts. This parallel execution scheme significantly increases the utilization of functional units, and, therefore, greatly improves machine throughput. Simulation results show that our n-threaded eleven-functional-unit processor ($n \leq 8$) can achieve a speed-up to nearly a factor of n, over a single-threaded processor. We also discuss the combination of our multithreading technique with a superscalar design.

1 はじめに

現在、我々は現実感のある高品位画像を高速生成する分散共有メモリ型並列計算機システムの開発を行なっている。本システムの第一の目的はコンピュータ・グラフィックスのアルゴリズムを効率よく実行することにあるが、アニメーションなどにおいて自然な動きを表現するためには物理モデルに従ったシミュレーションが必要であり、従って、大規模な数値計算をも高速実行する能力が要求される。本稿では、その並列計算機システムで使用する要素プロセッサのアーキテクチャ[1, 2] について述べる。

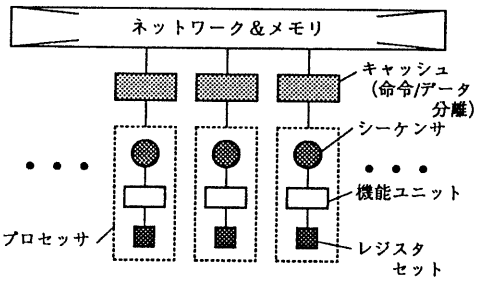
2 アーキテクチャ

2.1 並列スレッド命令発行方式

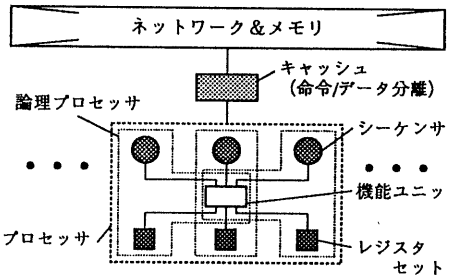
我々が対象とするアプリケーション・プログラムは粗粒度（プロセスまたはスレッド・レベル）および細粒度（命令レベル）の両方の並列性を備えている。従って、目的の並列計算機システムを構築するために、その各要素プロセッサにスーパスカラ方式やVLIW 方式のプロセッサを採用することも考えられる。しかし、命令レベルの並列性を活用するこれらのアーキテクチャでは、プログラムのもつデータ依存や制御依存などの命令間の依存関係によって、充分な並列性を得るには限界がある。

並列計算機システムの場合、システム内で並列実行する粗粒度レベルの処理単位のおのおのを高速実行することは問題そのものを高速実行することにつながるが、重要なのは問題そのものの高速実行であって、元の問題から分割されたある特定の部分問題のみの処理時間ではない。これは、コンピュータ・グラフィックスなどのアプリケーションにおいて特に当てはまる。そこで、我々は、各要素プロセッサ内において粗粒度並列性を積極的に利用し、これを多重命令発行と組み合わせることによってプロセッサのスループットを飛躍的に向上させることを狙った。

図 1(a) は並列計算機の構成を一般化して示したものである。図 1(a) に示された各要素プロセッサにおいて、前述の命令レベルの依存性が原因で機能ユニットの稼働率が30%であったと仮定しよう。この場合、機能ユニットを共有する形で３つの要素プロセッサを１つに結合した構成方式（図 1(b)）をとる。図 1(b) において、論理プロセッサが元の物理的な要素プロセッサに対応する。各論理プロセッサ上で実行される命令間に依存関係は存在しないので、単純には機能ユニットの稼働率が 90%まで改善されることが期待できる。



(a) 一般的な構成



(b) 我々のアーキテクチャを使用した構成

図 1: マルチプロセッサ・システムの構成

このように、１つのプロセッサ内において異なる命令流（スレッド）からの命令を同時に発行することにより、１つの命令の実行に伴う演算遅延を別のスレッドの命令の実行によって隠蔽する。我々はこの方式を「並列スレッド命令発行方式」と名付けた。

2.2 ハードウェア構成

プロセッサのハードウェア構成の概略を図２に示す。プロセッサは複数組の命令キュー・命令解読ユニット対（これをスレッド・スロットと呼ぶ）を持ち、それぞれが各自のプログラム・カウンタに基づいて各自の命令流（スレッド）の制御を行なう。

命令キューはフェッチした命令を蓄えるためのものであり、実際の命令フェッチ操作を行なう命令

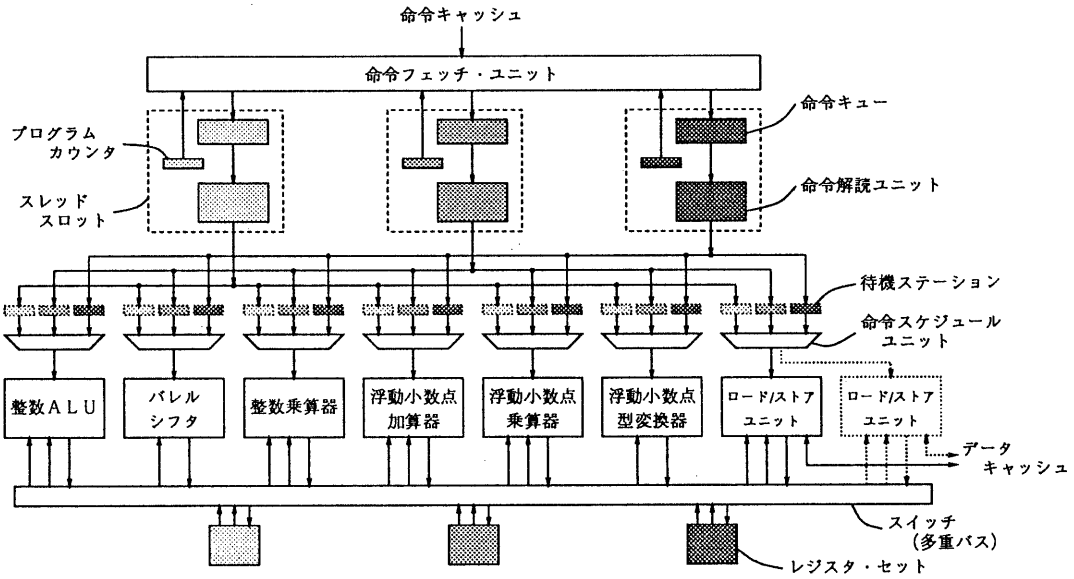


図 2: ハードウェア構成

フェッチ・ユニットはスレッド・スロット間で共有される。命令フェッチ・ユニットは命令キャッシュからの命令読み出しをインタリーブ方式で行なう。すなわち、あるスレッド・スロットからの命令読み出し要求に対して、命令フェッチ・ユニットはアドレスの連続する複数の命令を読み出し、命令キューに格納する。そして、次のフェッチ・サイクルでは、別のスレッド・スロットに対して同様の処理を行なう。ただし、分岐命令に起因する命令読み出し要求に対しては、これを優先して処理する。複数のスレッド・スロットが同時に分岐命令を実行した場合には待ちが生じるが、複数スレッドの実行によりその遅延は隠蔽することができる。

命令解読ユニットは命令キューから 1 命令ずつ取り出し、その命令が実行されるべき機能ユニットに対応してそれぞれ設けられている命令スケジュール・ユニットに対して、解読結果を発行する。また、分岐命令は命令解読ユニット内で処理する。自命令流内の命令間の依存関係に起因するハザードは、パイプライン・インタロックで対処する。各命令流間では依存関係が存在しないため、各命令解読ユニット間に渡る依存関係解析機構は不要である。

命令解読ユニットから発行された命令は、機能ユニットにおける競合を起こさない限り、並列に実行が開始される。競合を起こした場合には、命令スケジュール・ユニットが動的に調停を行う。この調停は、各スレッド・スロットに割り当てられた優先度に基づいて行なわれる。公平な命令選択を実現するため、この優先度は一定サイクル（巡回間隔）を経るごとに巡回シフトして再割り当てする（巡回多重優先度方式）。この様子を図 3に示す。

レジスタ・セットはスレッド・スロット毎に設ける。従って、おのおののレジスタ・セットは 3 つのポート（読み出し用 2、書き込み用 1）を有し、細粒度並列処理アーキテクチャにおいてみられるポート数増大の問題を免れている。レジスタ・セット群と機能ユニット群とは、多重バス構成のスイッチを介して接続する。図 2の例では 3 個のスレッド・スロットを備えているので、6 本のソース・バスと 3 本のデスティネーション・バス（それぞれ整数用と浮動小数点用とで分離）が必要である。

当初の設計 [1, 2] では、機能ユニットに対する命令発行の競合に対処する目的で、命令スケジュール・ユニットの入力部分に待機ステーションを設けていた。これは命令解読ユニットの出力ラッチとみなせるもので、例えば Tomasulo の予約ステーション [3] のような複雑な機能は持たない。発行された命令が命令スケジュール・ユニットに選択されなかった場

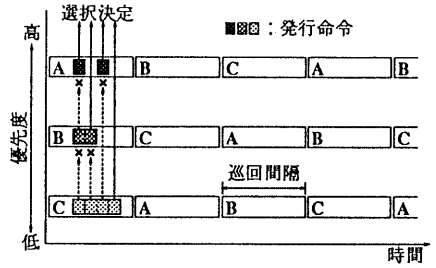


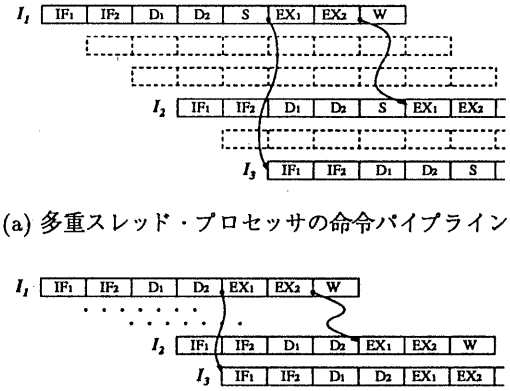
図3: 巡回多重優先度方式（A、B、Cはそれぞれスレッド・スロットを表す）

合、その命令は待機ステーション中で選択されるのを待つ。これにより、命令間で競合が生じた場合でも命令解読ユニットは命令の発行を続行できる。しかし、3.2節で議論するように、その効果は少ないため、待機ステーションの廃止を決定した。

2.3 命令パイプラインの構成

本多重スレッド・プロセッサにおける命令パイプラインのステージ構成を図4(a)に示す。大きくは、命令フェッチ(IF)、命令解読(D)、命令選択(S)、実行(EX)、書き戻し(W)の各ステージから成る。通常のプロセッサの命令パイプライン構成との違いは、命令スケジュール・ユニットでの命令選択のためにSステージが挿入されている点にある。また、IFステージは便宜上示したものであって、実際には1サイクルで命令キューから命令が読み出される。命令、データいずれの場合も1回のキャッシュ・アクセスには最低2サイクルを要するものと仮定しているので、IFステージに2ステージ設けた。参考のため、図4(a)の命令パイプラインを検討するにあたってベースとした従来の単一スレッド・プロセッサの命令パイプラインを図4(b)に示す。この図4(b)の命令パイプラインは、1命令1サイクル実行を基本とするRISCプロセッサからみた場合には、パイプライン・ピッチを半分にしたスーパパイプライン構造と捉えることができる。

単純にSステージを挿入した場合、命令解読ユニットから見た各命令の実行パイプラインの段数は1段増加することになる。これは単一スレッドの実行性能を低下させることになるため、各ステージでの担当機能を注意深く検討する必要がある。



(a) 多重スレッド・プロセッサの命令パイプライン

(b) 単一スレッド・プロセッサの命令パイプライン

図4: 命令パイプラインのステージ構成

まず、データ依存を有する命令を実行する場合について考える。図4(a)において、命令I_1の演算結果を命令I_2がソースとして使用するものと仮定する。D_1ステージでは命令のフォーマットや種類について検査する。つづくD_2ステージでスコアボーディング・ビットを読み出し、命令の依存関係をチェックする。その結果、発行可能と判定されればその命令は命令スケジュール・ユニットに送られるが、ソース・データの読み出し、およびデスティネーション・レジスタに対するスコアボードへの使用予約はSステージで行なう。また、スコアボードにおける使用予約の解除はEXステージの最後のステージで行なう。従って、命令I_2は命令I_1の開始から3サイクル遅れて開始することが可能であり、この遅延は図4(b)の場合と同じである。

次に、制御依存の場合について考える。命令I_1が条件分岐命令であり、命令I_3が分岐先の命令であるものと仮定する。分岐先のアドレスがレジスタで与えられる場合には、命令I_1の処理がSステージに達するまで分岐先命令のフェッチを待たなければならない。結果として、分岐する場合には5サイクルの遅れが生じ、図4(b)の場合（4サイクル）よりも1サイクル多い。また、命令フェッチ・ユニットを複数のスレッド・スロット間で共有していることから、さらに数サイクルの遅延が加わることもあり得る。単一スレッドの実行性能は低下することになるが、分岐命令に起因するこのような遅延は複数

表 1: 命令の実行パイプラインの段数

機能ユニット	カテゴリ	段数
整数 ALU	加算/減算	2
	論理演算	2
バレル・シフタ	シフト	2
整数乗算器	乗算	6
	除算	6
浮動小数点加算器	加算/減算	4
	比較	4
	符号操作	2
	移動	2
浮動小数点乗算器	乗算	8
	除算	24
型変換器	型変換	4
ロード/ストア †	ロード	4
	ストア	(4)

† ロード/ストア命令の発行遅延は 2 サイクル（他の命令については、毎サイクル発行可能）。

のスレッドを並列実行することで隠蔽することが可能であり、プロセッサのスループットにはそれほど大きな悪影響を与えない。なお、分岐しない場合には命令キュー内の命令を使用できるので、3 サイクルの遅延で次の命令を開始できる。

以上のように、レジスタ読み出しなどの機能を S ステージで実現することによってデータ依存関係が存在する場合の実行遅延の増大を防いだが、制御依存関係の場合には一部で実行遅延の増大を招く。逆に、例えば、レジスタ読み出しなどの機能を D_2 ステージで実現することにすれば、データ依存関係が存在する場合に実行遅延の増大を招き、制御依存関係の場合にはこれを回避できる。我々が前者の選択を行なった理由は、実際のアプリケーション実行において、制御依存関係を有する命令の実行遅延を隠蔽する方がデータ依存関係を有する命令の実行遅延を隠蔽するよりも効果的であると判断したことによる。

3 評価

3.1 シミュレーション・モデル

並列スレッド命令発行方式を評価するためにシミュレータをソフトウェアによって実現した。本シミュレータでは、スレッド・スロット数や機能ユニット構成、命令パイプライン構成などをパラメータとして与え、処理時間や各ユニットの使用状況を採取することができる。また、スレッド・スロット内で命令流内の同時発行可能な命令を複数発行する、すなわち、スーパスカラ方式を採用することが可能である。

今回評価に用いた機能ユニットの構成は、図 2 に示したように、整数 ALU、バレル・シフタ、整数乗算器、浮動小数点加算器、浮動小数点乗算器、型変換器、ロード/ストア・ユニットの計 7 ユニットから成る構成を基本構成（構成 A）とし、この基本構成にロード/ストア・ユニットを 1 つ追加した構成（構成 B）、および基本構成に 3 つのロード/ストア・ユニットと 1 つの整数 ALU を追加した構成（構成 C）について評価した。各命令の実行パイプラインの段数を表 1 に示す。なお、命令スケジュール・ユニットにおける巡回間隔は 8 サイクルとした。

キャッシュ・アクセスに関してはヒット率 100%を仮定した。また、複数のロード/ストア・ユニットを備えている場合でも、データ・キャッシュを並列にアクセスできるものとした。

命令パイプライン構成は図 4(a) に示したものを用いた。ただし、スレッド・スロット数が 1 の場合には図 4(b) のパイプラインを採用した。また、以降の節で用いる性能向上率は、スレッド・スロット数が 1 で、機能ユニット構成が基本構成の場合を 1 として正規化している。

アプリケーション・プログラムにはレイ・トレーシング法を用いた画像生成プログラム、およびリバモア・ループの No.1、No.7、No.15（いずれも倍精度）のカーネルを使用した。これらのアプリケーション・プログラムは、ループの各イタレーションをインタリーブしてそれぞれのスレッドに分割・並列化した。

3.2 処理性能の評価

それぞれのアプリケーションに対する並列スレッド命令発行方式の評価結果を図 5〜8 に示す。

リバモア・ループのカーネル No.1 と No.7 の場合、基本構成ではスレッド数を増やしても性能向上は 2 倍程度である。これは命令コード中に多くのロード/ストア命令を含んでおり（シミュレーションに用いたマシン・モデルが倍語長のロード/ストア命令を持たないことも原因の 1 つである）、命令実行の要求がロード/ストア・ユニットの処理能力を越えているためである。

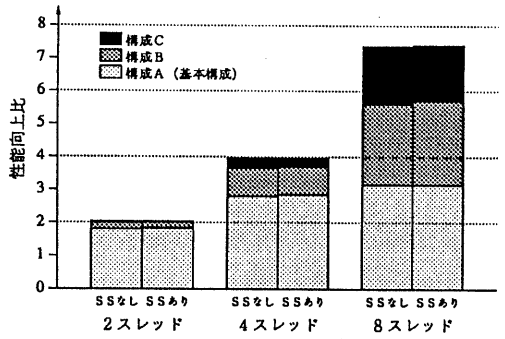


図5: 並列スレッド命令発行方式による性能向上比（レイ・トレーシング）

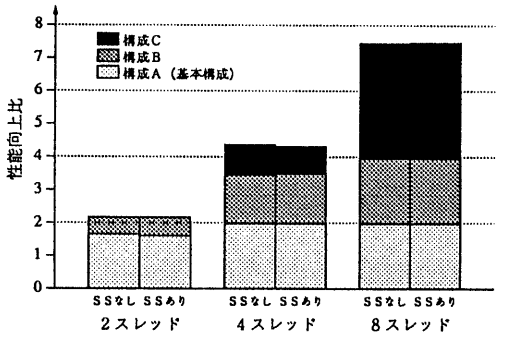


図6: 並列スレッド命令発行方式による性能向上比（リバモア・カーネルNo.1）

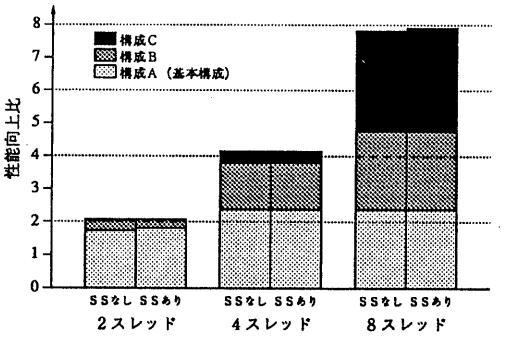


図7: 並列スレッド命令発行方式による性能向上比（リバモア・カーネルNo.7）

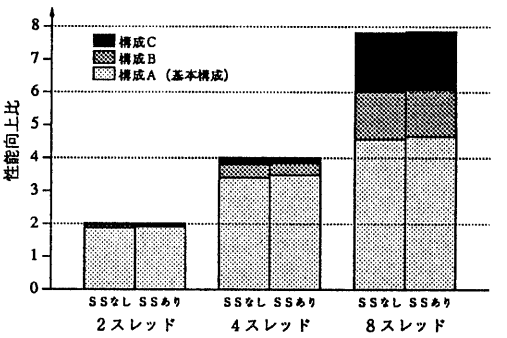


図8: 並列スレッド命令発行方式による性能向上比（リバモア・カーネルNo.15）

使用率の高い機能ユニットを順次追加していくことで、より大きな性能向上を達成することができる。構成Cでは、いずれのプログラムにおいても、スレッド数にほぼ比例した性能向上が得られている。なお、nスレッドでn倍以上の性能向上を達成している場合がみられる。これは1回のキャッシュ・アクセスに2サイクル分の時間（ブロック・サイクル）を必要とするものと仮定したことに起因するもので、複数のロード/ストア・ユニットを備えることによって、並列スレッド命令発行方式の効果の上にさらに命令発行の機会が増大したからである。

さて、待機ステーション（SS）を設けた場合とそうでない場合とを比較すると、待機ステーションを設けることの効果は僅かであり、当初期待した結果が得られていない。また、カーネルNo.1の場合には、待機ステーションの存在がかえって不利な命令実行状態を引き起こし、僅かながら性能を低下させている（単純なループの場合には、静的コード・スケジューリングによってこのような状況を回避できる[2]）。待機ステーションの有効性はスレッド内における命令レベルの並列性に依存するが、およそ次のように考察できる。

- スレッド数が少なく、従って命令実行要求がプロセッサの処理能力に満たない場合には、機能ユニットでの競合があまり生じない。従って、待機ステーションの効果が発揮できる機会が少ない。
- スレッド数が多く、命令実行要求がプロセッサの処理能力を上回り、命令実行要求のキューイング状態に陥ると、待機ステーションが存在してもその状況を緩和することはできない。

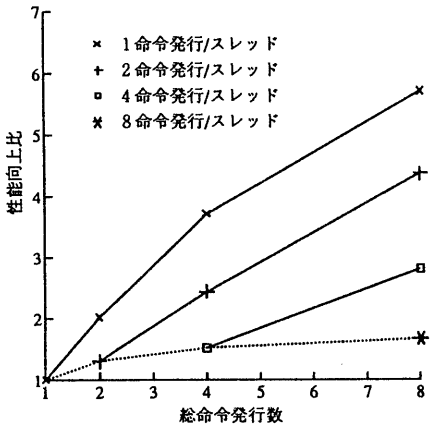


図9: 細粒度並列処理との組み合わせによる性能向上比（レイ・トレーシング）

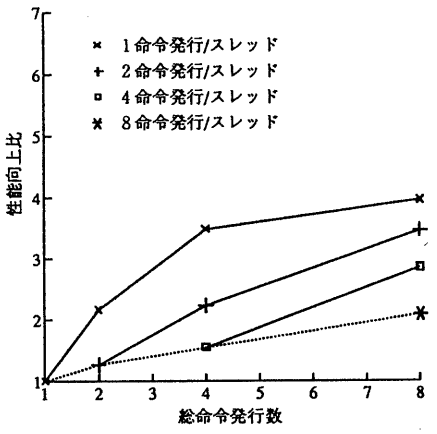


図10: 細粒度並列処理との組み合わせによる性能向上比（リバモア・カーネルNo.1）

- 命令流内で同じ機能ユニットを使用する命令が連続すれば、待機ステーションの効果は発揮できない。

待機ステーションは解読後の命令の他にレジスタから読み出したオペランド・データも記憶するので多くのハードウェアが必要である。従って、今回のシミュレーション結果から、待機ステーションの廃止を決定した。

3.3 細粒度並列処理との組み合わせ

ここでは、各スレッド・スロット内で、スーパスカラ・プロセッサにみられるような単一命令流内での多重命令発行方式を採用する場合について検討する。

各命令解読ユニットは、命令キューの先頭のs個の命令を同時に解読し、発行可能な命令を並列発行する。スレッド・スロット数をtとすると、1サイクルに発行できる命令数（総命令発行数）は$d = t \times s$である。sの値に応じて、レジスタ・セットのポート数も拡張した。また、命令のフェッチ幅など、同じdの値をもつプロセッサ同士でハードウェア量がほぼ同一となるように考慮した（プログラム・カウンタの数や命令解読/発行可能性検査の複雑さなどについては無視する）。純粋なスーパスカラ・プロセッサの場合には、文献[4, 5]などに提案されている方式を採用することでさらに性能向上を図ることが可能であるが、ハードウェア量をほぼ一定にして評価を行なうという観点から、ここではそれらの方式の採用は考えないものとする。

構成Bを用いた評価結果を図9〜12に示す。いずれのアプリケーションにおいても、スレッド数を増やす方がスレッド当たりの同時命令発行数を増やすよりも効果的であることが示されている。

Prasadh等[6]は、VLIW命令のNOP部分を他のスレッドのNOPでない命令フィールドで動的に置き換えることにより、機能ユニットの使用率向上を図っている。しかし、図9〜12に示した結果から判断すると、多重スレッド処理方式と細粒度（命令レベル）並列処理方式との組み合わせは、価格性能比の観点からあまり得策であるとは言えない。

単一命令流の高速実行を目指すスーパスカラ方式（またはVLIW方式）とスループットの向上を目指す並列スレッド命令発行方式とは、その目的の違いから単純に比較することは不可能である。しかし、複数のスレッドを同時実行する方式を導入するのであれば、細粒度並列処理方式の採用は避けるべきであろう。Prasadh等は、スレッド数を増やすことによって、スレッド毎に設けたレジスタ・セットのポート数が少なくできると報告している[6]が、これは、暗に、我々の結論を支持するものであると考えられる。

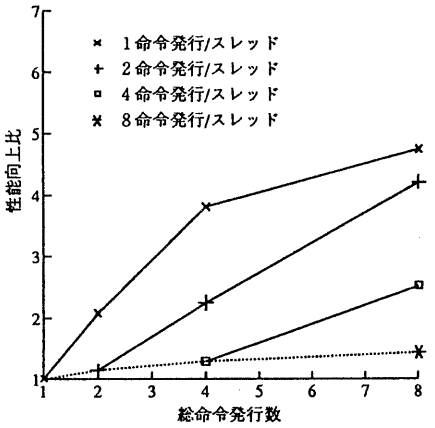


図 11: 細粒度並列処理との組み合わせによる性能向上比（リバモア・カーネル No.7）

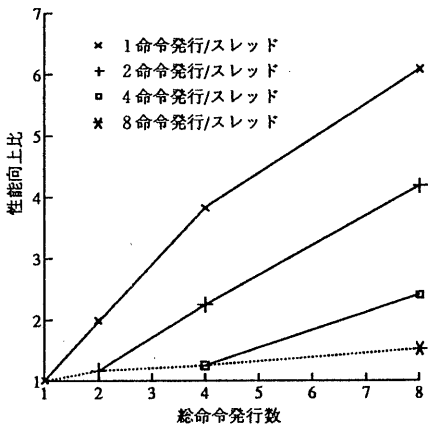


図 12: 細粒度並列処理との組み合わせによる性能向上比（リバモア・カーネル No.15）

4 おわりに

複数スレッドを同時実行することによってスループット向上を図る要素プロセッサ・アーキテクチャについて述べ、シミュレーションによる評価を通して、我々のアーキテクチャの有効性を示した。今後は、キャッシュの効果も含めたシミュレーションにより、アーキテクチャの評価を行なっていく。

現在、プロセッサの詳細設計を進めており、また、遠隔メモリ・アクセスの際に実行スレッドを高速に切り替えることによってアクセス遅延を隠蔽する多重スレッド並行処理機構についても検討している。

参考文献

[1] H. Hirata, Y. Mochizuki, A. Nishimura, Y. Nakase and T. Nishizawa: "A Multithreaded Processor Architecture with Simultaneous Instruction Issuing." *Proc. of Intl. Symp. on Supercomputing, Fukuoka, Japan*, pp. 87-96, (November 1991).

[2] H. Hirata, K. Kimura, S. Nagamine, Y. Mochizuki, A. Nishimura, Y. Nakase and T. Nishizawa: "An Elementary Processor Architecture with Simultaneous Instruction Issuing from Multiple Threads." *Proc. of the 19th Annual Intl. Symp. on Computer Architecture*, (May 1992).

[3] R.M. Tomasulo: "An Efficient Algorithm for Exploiting Multiple Arithmetic Units." *IBM Journal of Research and Development*, vol. 11, no. 1, pp. 25-33, (January 1967).

[4] 久我守弘, 入江直彦, 弘中哲夫, 村上和彰, 富田眞治: "SIMP(単一命令流/多重命令パイプライン) 方式に基づく『新風』プロセッサの低レベル並列処理アルゴリズム." 情報処理学会論文誌, vol. 30, No. 12, pp. 1603-1611, (1989 年 12 月).

[5] M.D. Smith, M.S. Lam and M.A. Horowitz: "Boosting Beyond Static Scheduling in a Superscalar Processor." *Proc. of the 17th Annual Intl. Symp. on Computer Architecture*, pp. 344-354, (May 1990).

[6] R.G. Prasadh and C. Wu: "A Benchmark Evaluation of a Multi-Threaded RISC Processor Architecture." *Proc. of the 20th Intl. Conf. on Parallel Processing*, pp. I:84-91, (August 1991).