多重スレッド・多重命令発行を用いる要素プロセッサ・アーキテクチャ

平田博章† 木村浩三† 永峰聡†
西澤貞次† 鷺島敬之†

我々は，アプリケーションが有する粗粒度並列性を多重命令発行と組み合わせることによって，スループットの向上を狙うプロセッサ・アーキテクチャの開発を行っている．本稿では，そのアーキテクチャの特徴である「並列スレッド命令発行方式」について述べ，キャッシュ効果を含めたシミュレーションによって評価した結果を報告する．アプリケーション・プログラムをデータ分割によって並列化し，本プロセッサ上で同時実行されるスレッドが命令テキストを共有する場合，命令キャッシュのヒット率は単一スレッド実行時よりも向上することがわかった．一方，データ・キャッシュについては複数スレッドの同時実行によって単一スレッドの場合よりもヒット率が低下する傾向がある．しかし，適切なデータ・キャッシュ構成のもとではヒット率低下の傾向を抑えることができ，2, 4, 8スレッド並列実行時でそれぞれ単一スレッド・プロセッサの2.0倍，3.6倍，5.4倍の性能向上が得られることを確認した．

An Elementary Processor Architecture with Parallel Instruction Issuing from Multiple Threads

Hiroaki Hirata,† Kozo Kimura,† Satoshi Nagamine,†
Teiji Nishizawa† and Takayuki Sagishima†

In this paper, we present a multithreaded processor architecture which improves machine throughput. In this architecture, instructions from different threads are issued simultaneously to multiple functional units, and these instructions begin execution unless there are functional unit conflicts. Through simulation, including finite cache effects, we demonstrate the effectiveness of our architecture. When multiple threads are created from a program by means of data partitioning and they share the same code text, the hit ratio of the instruction cache is higher than the single-thread execution. On the other hand, the parallel-thread execution generally damages the hit ratio of the data cache. With an appropriate data cache organization in our multithreaded processor, however, two, four and eight threads executing in parallel can achieve a 2.0, a 3.6 and a 5.4 factor speed-up, respectively, over single-thread execution.

1. はじめに

現在，我々はコンピュータ・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を目的とした並列計算機システムの開発を行っている．現実感のある高品位画像を生成するアルゴリズムとしてはレイ・トレーシング法[1]やラジオシティ法[1]が有名であるが，同時に，これらのアルゴリズムは多大な計算時間を要することでも知られている．そこで，我々は，これらのアルゴリズムに内在する粗粒度並列性を活かして，複数命令流を同時実行する要素プロセッサ・アーキテクチャを提案してきた[2)~5)]．本稿では，キャッシュ効果を含めたシミュレーションによって本アーキテクチャを評価した結果を報告する．

† 松下電器産業(株)メディア研究所
Media Research Laboratory, Matsushita Electric Industrial Co., Ltd.

2. アーキテクチャ

2.1 並列スレッド命令発行方式

レイ・トレーシング法やラジオシティ法による画像生成時間の大部分は，交差判定と呼ばれる処理によって占められる．従って，交差判定処理自体を高速化することは，全体の画像生成時間を短縮する上で非常に重要である．交差判定とは，ある視線が物体と交わるかどうかを調べるものであり，視線ごとに全く独立に処理することができる．このようにアルゴリズムそのものに内在する多大の粗粒度並列性を利用することに

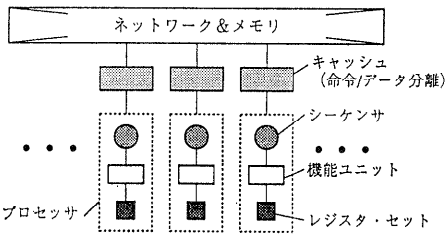
よって，マルチプロセッサを用いて高速化を図ることができる．

しかし，並列化されたそれぞれの処理単位（以後，スレッドと呼ぶ）に内在する細粒度並列性は乏しく，1つのプログラム・カウンタを備えて時間的に1つのスレッドの実行に占有される従来方式のプロセッサを使用する場合には，プロセッサがパイプライン化などの高速化技術を用いて設計されていても，そのメリットを十分に活かすことは難しい．例えば，生成する画像データに依存する条件分岐，サブルーチン呼び出し，他の演算と比べて比較的演算遅延の大きい浮動小数点演算，複雑な形状データを扱うことによって必要となる間接データ・アクセスなどが頻繁に現れ，これらが命令レベルの依存関係を生み出し，プロセッサの命令発行率を低下させる原因となっている．

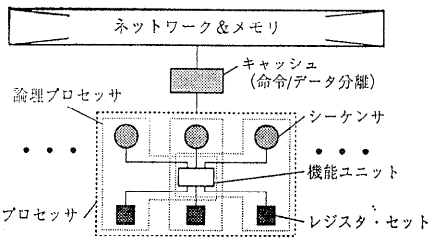
そこで，我々は，プロセッサ内において粗粒度並列性を積極的に利用し，これを多重命令発行と組み合わせることによってプロセッサのスループットを飛躍的に向上させることを狙う．**図1**(a)は並列計算機の構成を一般化して示したものである．図1(a)に示された各要素プロセッサにおいて，命令間に存在するデータ依存や制御依存などの依存関係が原因で，機能ユニットの稼働率が30%であったと仮定☆しよう．この場合，機能ユニットを共有する形で3つの要素プロセッサを1つに結合した構成方式（図1(b)）をとる．図1(b)において，論理プロセッサが元の物理的な要素プロセッサに対応する．このように，1つのプロセッサ内において異なる命令流（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実行によって隠蔽する．異なる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30×3=90%まで改善されることが期待できる．

ところで，1つのプロセッサ内で複数の命令流を並列（並行）実行するマルチスレッド・プロセッサ（パイプライン化 MIMD，資源共有 MIMD などとも呼ばれる）の提案は古くから行われてきた．**図2**(a)はその先駆けである HEP[8] の命令パイプライン・モデルを示したものである．各命令流からの命令はサイクル単位でインタリーブされ，順次命令パイプラインに投入される．パイプライン中のすべてのステージが命令流間で共有される．

これに対し，我々が提案するアーキテクチャの命令パイプライン・モデルを図2(b)に示す．デコード・ステージを命令流ごとに設けている点が図2(a)と異なる．単に演算遅延を隠蔽するだけでなく，ヘテロジニアスな多重機能ユニットに対して複数命令を同時発

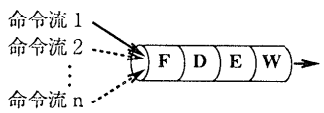


(a) 一般的な構成
(a) Conventional organization.

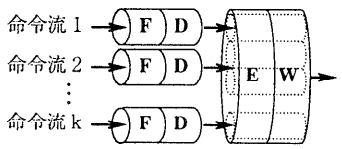


(b) 本論文のアーキテクチャを使用した構成
(b) Organization using our multithreaded processor.

図1 マルチプロセッサ・システムの構成
Fig. 1 Multiprocessor system organizations.



(a) 従来(HEP)のモデル
(a) Conventional model (Model of HEP).



(b) 本論文で提案するモデル
(b) Our model.

F: 命令フェッチ D: 命令解読
E: 実行 W: 結果書き戻し

図2 マルチスレッド・プロセッサの命令パイプライン・モデル
Fig. 2 Instruction pipeline models for multithreaded processor architectures.

☆ アプリケーション・プログラムの性質やプロセッサの機能ユニット構成に依存すると考えられるが，本論文の評価環境の下で行った単一スレッド実行の評価結果によれば，ロード/ストア・ユニットの稼働率が最も高く，その値は30%強であった．

行することにより，図2(a)の場合よりもさらに機能ユニットの利用率を向上させる．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3**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）の制御を行う．

図2(b)では命令フェッチ・ステージもスレッド・スロットごとに設けられていたが，命令キャッシュに対して任意の異なるアドレスの命令を並列にアクセスするための機構はハードウェア的に高価となるので，各スレッド・スロットごとに命令キューを配し，命令フェッチ・ユニットをスレッド・スロット間で共有することにした．命令フェッチ・ユニットは命令読み出しをインタリーブ方式で行う．すなわち，ある1つのスレッド・スロットに対して，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命令キャッシュから同時に読み出し，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う．ただし，分岐命令に起因する命令読み出しについては，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，通常は，複数スレッドの実行によりその遅延を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する命令スケジュール・ユニットに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．発行命令が命令スケジュール・ユニットで受け付けられない場合も，同様に，以降の命令発行をインタロックする．

命令解読ユニットから発行された命令は，機能ユニットの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た優先度に基づいて行う．公平な命令選択を実現するために，一定サイクルを経るごとに優先度を巡回シフトして再割り当てする．

各レジスタ・セットはスレッド・スロットと1対1に対応する．実行中のスレッドは，自分に割り当てられているレジスタ・セット以外のレジスタをアクセスすることはできない．スレッド・スロットから発行される命令は1サイクルに最大1命令であるので，レジスタ・セットのポート数は3（読み出し：2，書き込み：1）で十分である．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．バスの本数は，機能ユニット数ではなく，スレッド・スロット数に比例する．図3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

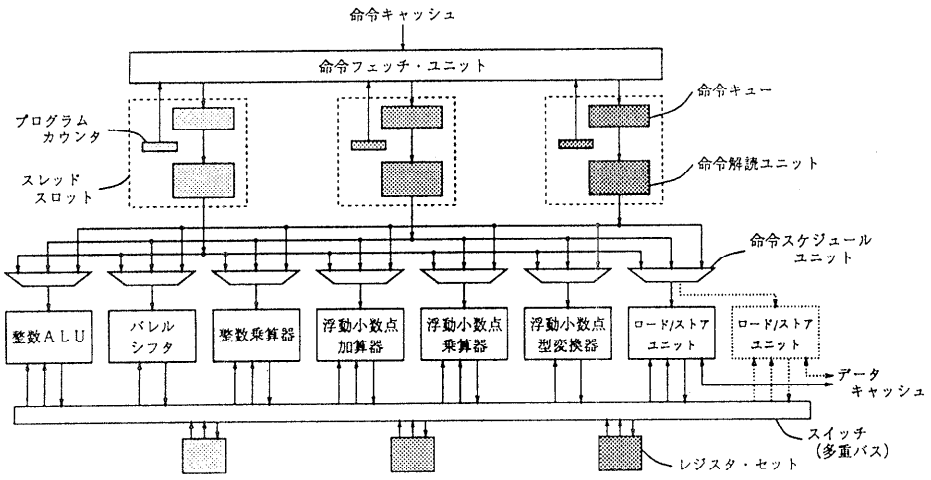


図3 ハードウェア構成
Fig. 3 Hardware organization.

数用と浮動小数点用とで分離）が必要である．現在の半導体技術では，このスイッチに要するハードウェア量は大きくなることが予想されるので，機能ユニット共有化によるハードウェア・コスト削減とのトレードオフも考慮して，スレッド・スロット数を選択する必要がある．

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論理プロセッサの命令パイプラインのステージ構成を**図4**(a)に示す．また，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．

以下，図4(a)のパイプラインの各ステージの機能について説明する．

①命令フェッチ (IF) ステージ　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．命令とデータのいずれの場合も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たので，図4では IF ステージを2ステージ設けて示した．

②命令解読 (D) ステージ　D_1 ステージで命令のフォーマットや種類を検査し，つづく D_2 ステージで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を命令スケジュール・ユニットに送る．

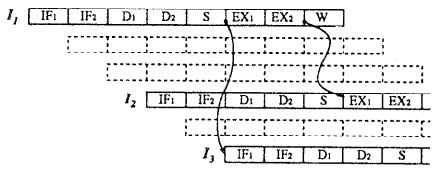
③命令選択 (S) ステージ　命令スケジュール・ユニットで命令選択を行う．また，命令解読ユニットから発行された命令のソース・データ読み出しやデスティネーション・レジスタへの書き込み予約はこのSステーションで行う．

④実行 (EX) ステージ　それぞれの機能ユニットで命令の実行を行う．各機能ユニットはその操作の種類に応じてパイプライン化されている．デスティネーション・レジスタに対する書き込み予約の解除は EX ステージの最後のステージで行う．

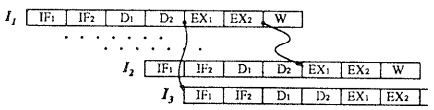
⑤結果書き戻し (W) ステージ　演算結果をレジスタに書き戻す．書き戻されるデータは同じサイクルで読み出すことができる．

レジスタの読み出しを D_2 ステージではなく S ステージで行い，書き込み予約の解除をWステージではなく EX ステージで行うのは，命令パイプラインにSステージが挿入されることによって演算遅延が増大するのを防ぐためである．例えば，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_2 ステージ，Wステージでそれぞれレジスタ読み出し，書き込み予約の解除を行ったならば，命令 I_2 は命令 I_1 の開始から少なくとも4サイクル待たなければ開始できない．これに対して，各ステージに対する機能割当を前述のように行えば，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となる．この遅延は図4(b)の場合と同じである．

しかし，レジスタ読み出しを D_2 ステージからSステージへと遅らせたことにより，分岐命令の実行性能の低下を招く場合がある．命令 I_1 が分岐命令であり，命令 I_3 が分岐先の命令であるものと仮定する．また，簡単のため，複数のスレッドで分岐が同時に発生しないものと仮定する．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の命令フェッチを待たなければならない．結果として5サイクルの遅れが生じ，これは図4(b)の場合よりも1サイクル多い．従って，単一スレッドの実行性能は低下することになる．しかし，レジスタ間接の分岐命令の出現頻度を考慮すると，このような遅延を複数スレッドの並列実行によって隠蔽することは十分可能であり，プロセッサのスループットに



(a) 多重スレッド・プロセッサの命令パイプライン
(a) Instruction pipeline of multithreaded processor.



(b) 単一スレッド・プロセッサの命令パイプライン
(b) Instruction pipeline of single-threaded processor.

IF: 命令フェッチ　D: 命令解読
S: 命令選択　EX: 実行
W: 結果書き戻し

図4 命令パイプラインのステージ構成
Fig. 4 Stage configuration of instruction pipelines.

はそれほど大きな悪影響を与えない.

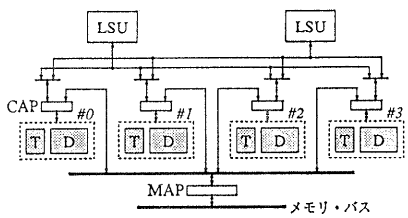
3. 性能評価

3.1 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，並列スレッド命令発行方式の評価を行った．今回，評価に用いた機能ユニットの構成は，図3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，これにロード／ストア・ユニットを1つ追加した構成（構成B）についても評価した．この構成Bでは，複数スレッドの同時実行に対して十分なデータ供給能力を確保するため，**図5**に示すようにデータ・キャッシュを4つのバンクで構成した．キャッシュを検索する際のインデックスの下位2ビットをバンクの識別に用いることにより，データ・アレイもタグ・アレイもバンクごとに完全に分割して構成できる．従って，いずれかのバンクでキャッシュ・ミスを起こしても，他のキャッシュ・ミスを起こしていないバンクをアクセスすることが可能である．

命令／データ・キャッシュともに論理キャッシュを想定し，ライン・サイズは32Bに設定した．マッピング方式にセット・アソシアティブ方式を採用した場合の置き換えアルゴリズムには完全なLRUを仮定した．また，キャッシュ・ミス時のペナルティとして，1ライン分のデータをメモリとの間で転送するのに要するサイクル数を与えた．なお，書き込み方式にコピー・バック方式を採ったので，ダーティ状態のラインを置換する場合には，パラメータとして与えられた値の2倍のペナルティを課した．

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．各命令の実行ステージにおけるパイプライン段数を**表1**に示す．



LSU: ロード/ストア・ユニット
CAP: キャッシュ・アクセス・ポート
MAP: メモリ・アクセス・ポート
T: タグ・アレイ
D: データ・アレイ

図5 データ・キャッシュの多重バンク構成方式
Fig. 5 Multibanked organization of data cache.

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（C言語で記述）を使用した．並列化（スレッド生成）はデータ分割（より具体的には画面分割）によって行った．各スレッドは単一のアドレス空間を共有し，スタック・フレームとして使用する領域のみ前記アドレス空間から切り出して個別に割り付けた．

3.2 命令アクセス評価

2.1節で述べたように，我々は，アプリケーション・プログラム実行時における単一スレッド・プロセッサの機能ユニットの稼働率が低く，機能ユニットを複数のスレッドで共有できることに着眼して並列スレッド命令発行方式を提案するに至った．しかし，nスレッドを並列に実行することでn倍のキャッシュ容量が必要となるのであれば，本方式はキャッシュに関してはコストの軽減に寄与できないことになる．ここでは，機能ユニットと同様，キャッシュに関してもスレッド間での共有が可能であることを示す．

まず，命令アクセスに関して，アプリケーション・プログラムの振る舞いを調べた．オペレーティング・システムの分野で用いられるワーキング・セット[6]の概念を導入し，評価した結果を**図6**，**表2**に示す．仮想アドレス空間をキャッシュのライン・サイズである32Bのブロックに分割し，プロセッサがアクセスした

表1 命令の実行パイプラインの段数
Table 1 Execution latencies of instructions.

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)

† キャッシュ・アクセス・ステージでは2サイクルのブロックを必要とする．

ブロックに足跡マークを付ける．ある時間単位 Δ の間に足跡マークが付けられたブロックの数がワーキング・セット・サイズであり，その時間的変化を示したのが図6である．Δ には単一スレッド実行において平均 50 命令を実行するのに要する時間を設定し，n スレッド並列実行によって n 倍の性能向上が得られるものと仮定して測定した．従って，例えば2スレッド並列実行の場合は Δ の間に100命令のアクセスが行われることになる．処理を開始した直後の時点では，すべてのスレッドが同一のテキスト領域をアクセスしているが，しばらくすると，データ依存による分岐などによってそれぞれ時間的に異なる領域をアクセスするようになり，並列実行スレッド数が多いほどワーキング・セット・サイズも大きくなる．これを集計して最大値，平均値などを示したものが表2である（表2では，データ・アクセスの場合についても併記している）．表2において，総数として提示されている値は Δ に ∞ を設定した場合のワーキング・セット・サイズであり，合計1,298ブロックがアクセスされたことを表している．平均値でみると，並列実行スレッド数が8の場合を除いて，ワーキング・セット・サイズが並列実行スレッド数に比例して増加している．これは，命令アクセスにおいて並列実行スレッド間で時間的にコード共有を行うことが不可能であることを意味する．しかし，並列実行スレッド数が8の場合のワーキング・セット・サイズの増加は 47.6/7.5=6.3倍（<8）に留まっている．並列実行されるスレッドの数が少ない場合には，小さい時間間隔における各スレッドのワーキング・セットに重なりは生じないが，並列実行スレッド数が多くなると，全体のワーキング・セット・サイズは増加するものの，異なるスレッドが同一の領域をアクセスする場合も生じ得ることがわかる．

次に，命令キャッシュのヒット率の測定結果を**図7**に示す．グラフの横軸には命令キャッシュの容量（1KB～8KB）およびスレッド・スロット数（1～8）をとっている．1KBのキャッシュでは，スレッド・スロット数が多いほどヒット率は低下している．しかし，キャッシュ容量を増加する，あるいは，キャッシュのウェイ数を増加して連想度を上げると，複数スレッドの同時実行によってヒット率が向上する場合がみられる．例えば，容量4KBの2ウェイ・セット・アソシアティブ方式のキャッシュにおいては，単一スレッド実行時よりも2スレッド実行時のほうが，また，

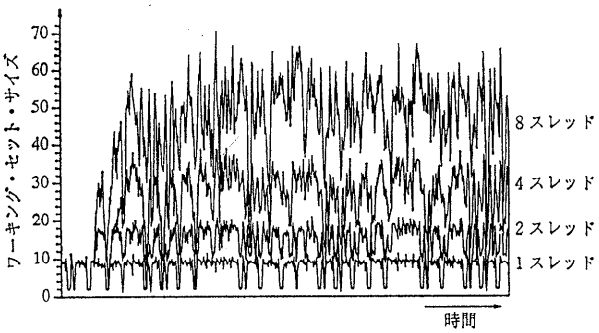


図6 命令アクセスにおけるワーキング・セット・サイズの時間的変化
Fig. 6 Working set sizes of instructions with respect to time.

表2 ワーキング・セット・サイズの評価結果
Table 2 Evaluation of working set sizes.

		並列実行スレッド数				参考
		1	2	4	8	
命令	最大	19	29	53	85	Δ=50 命令時間
	平均	7.5	14.6	27.2	47.6	
	総数	1,298				
データ	最大	20	36	57	99	Δ=100 命令時間
	平均	8.7	17.2	33.8	64.0	
	総数	4,958	5,085	5,338	5,918	

（単位：ブロック）

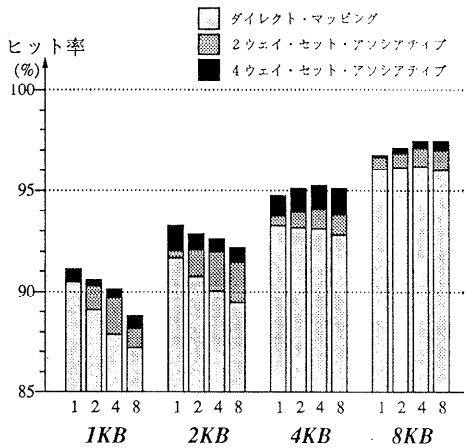


図7 命令キャッシュのヒット率
Fig. 7 Hit ratios of instruction caches.

2スレッド実行時よりも4スレッド実行時のほうが高いヒット率を示している.

複数のスレッドでキャッシュを共有する場合，次の2種類のキャッシュ干渉[7]が生じる.

- あるスレッドがアクセスしようとする命令（データ）が，他のスレッドによって既にキャッシュ上にロードされている（正のキャッシュ干渉）.
- あるスレッドによって以降もアクセスされる命令（データ）が，他のスレッドによってキャッシュから追い出される（負のキャッシュ干渉）.

図6や表2によれば，複数スレッドの並列実行によってワーキング・セット・サイズが増加する．従って，キャッシュ容量が小さく，また，連想度も低い場合には，負のキャッシュ干渉の影響が大きく作用し，並列実行スレッド数が多いほどキャッシュのヒット率は低下する．しかし，キャッシュ容量を増加し，また，連想度を増やすと，正のキャッシュ干渉の効果が負の干渉効果を上回って現れる．小さな時間間隔の範囲で異なるスレッドが同一の領域をアクセスすることは希であるが，より大きな時間間隔では各スレッドのワーキング・セットに重なりが生じ，結果としてキャッシュのヒット率が増加することが確かめられた.

本例のように，データ分割によって並列化された各処理単位が命令テキストを共有して並列実行される場合には，単一スレッド実行の場合よりも命令キャッシュのヒット率を向上させることができる．従って，少なくともこのような並列処理方式に限れば，命令キャッシュのヒット率の点からも，並列スレッド命令発行方式は要素プロセッサのアーキテクチャとしての適性を有するものと言える.

3.3 データ・アクセス評価

命令アクセスの場合と同様，データ・アクセスに関してワーキング・セット・サイズを評価した結果を**図8**，表2に示す．Δ には単一スレッド実行において平均100命令を実行するのに要する時間を設定して測定した．アプリケーション・プログラムにおけるロード／ストア命令の出現頻度は全命令の約 40% を占めるため，データ・アクセスを 40 回行うのに要する時間を Δ として設定したものと考えることもできる．命令の場合と異なり，ワーキング・セット・サイズの変動は少ない.

表2において，総数の値が並列実行スレッド数によって異なるのは，並列実行するスレッドごとにスタック・フレームの領域を用意しているからである．1スレッドが使用するスタック・フレーム領域の大きさは約130ブロックであり，全体の使用量に比べてかなり小さい．しかし，より詳細に調べてみると，スタック・フレーム領域におけるワーキング・セット・サイズが全ワーキング・セット・サイズの 75% 以上を占

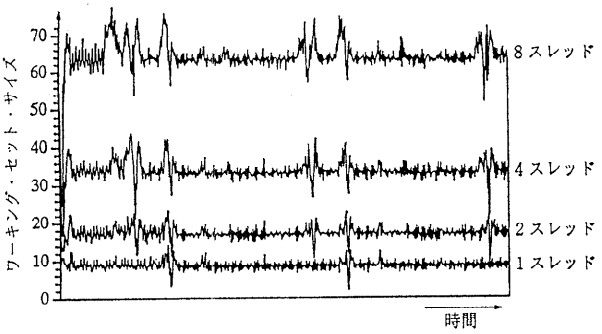


図8 データ・アクセスにおけるワーキング・セット・サイズの時間的変化
Fig. 8 Working set sizes of data with respect to time.

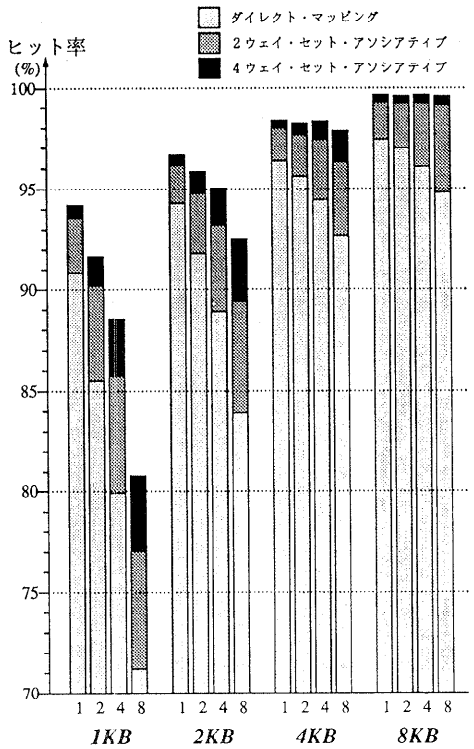


図9 データ・キャッシュのヒット率
Fig. 9 Hit ratios of data caches.

めている．表2ではワーキング・セット・サイズがスレッド・スロット数に比例して増加しているが，これにはスタック・フレーム領域に対するアクセスが大きく関与しているものと考えられる．

データ・キャッシュのヒット率の測定結果を**図9**に示す．8KB の 2/4 ウェイのキャッシュにおけるヒット率はスレッド・スロット数と無関係にほぼ同じ値となっているが，命令キャッシュの場合とは異なり，全体的にスレッド・スロット数の増加に伴ってヒット率が低下している．スタック・フレーム領域は並列実行されるスレッド間で共有されないので，正のキャッシュ干渉を期待することはできない．

3.4 処理性能の評価

これまでのキャッシュ・ヒット率に関する評価から，適切なキャッシュ構成を選択すれば，並列スレッド命令発行方式によって命令キャッシュのヒット率を向上させることができることがわかった．しかし，データ・キャッシュにおいては，ヒット率は低下する傾向にある．データ・キャッシュのヒット率が低ければ，複数スレッドの並列実行による高速化がキャッシュ・ミスによって相殺される危険性がある．そこで，ここでは，データ・キャッシュのヒット率と性能向上との関係について評価した結果を報告する．条件を揃えるため，命令キャッシュについては100% のヒット率を仮定した．

まず，図9に示した中でヒット率低下が最も著しいダイレクト・マッピング方式の1KB データ・キャッシュを用いた場合について，その評価結果を**図 10** に示す．機能ユニット構成が構成Aおよび構成Bのそれぞれの場合について，ミス・ペナルティを変化させて処理時間を測定した．図 10 では，スレッド・スロット数が1，機能ユニット構成が基本構成，ミス・ペナルティが0サイクル（キャッシュのヒット率が 100% であると仮定した場合と同じ効果が得られる）と仮定した場合の性能向上率を1として正規化している．ミス・ペナルティが0サイクルの場合には，スレッド・スロット数を増やすことによって性能は向上するが，ミス・ペナルティを課すことによって性能向上率は小さくなる．特に，スレッド・スロット数が8の場合には，スレッド・スロット数が4の場合よりも性能が低く，単一スレッド実行時の性能と比べても性能向上はわずかである．

構成Aを採用した場合について，メモリ・バス（キャッシュ-主記憶間）およびプロセッサ・バス（プロセッサ-キャッシュ間）におけるバス使用率を調べた結果を**図 11** に示す．ここで，バス使用率とは，データ転送に費やされた時間の総和が全処理時間に占める割合であり，キャッシュ・ミスの場合にはメモリ・バス上での転送時間もプロセッサ・バスの使用時間に含め

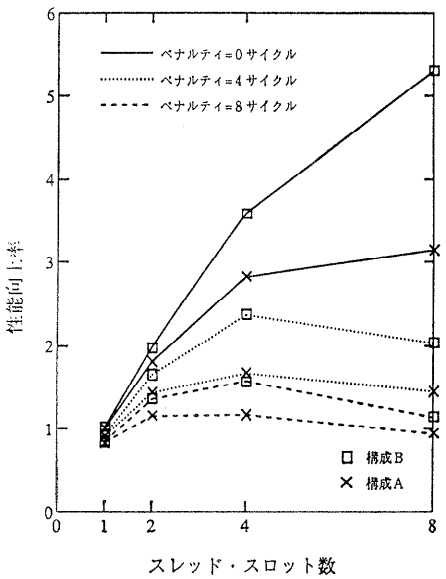


図 10 性能向上率（容量 1KB，ダイレクト・マッピング方式のキャッシュ使用時）
Fig. 10 Speedups with 1KB, direct mapped data cache.

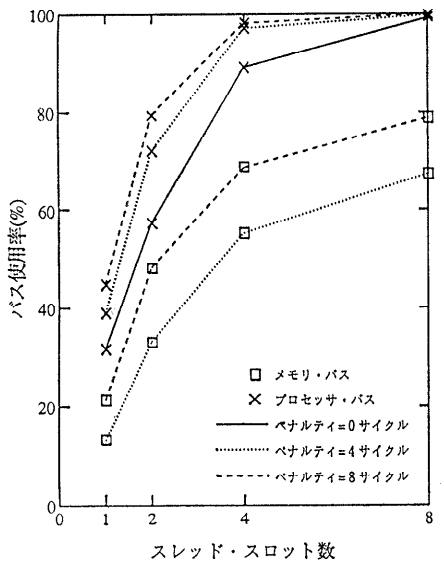


図 11 バス使用率（構成A）
Fig. 11 Data bus utilizations (with configuration A).

て計算した．なお，ミス・ペナルティを0サイクルとした場合のメモリ・バスのバス使用率は0%であるので図には示していない．さて，キャッシュ・ミスが発生するにつれてメモリ・バス，プロセッサ・バス共にデータ転送に費やされる時間が増大する．図11によれば，スレッド・スロット数が4のときに，プロセッサ・バスのバス使用率がほぼ100%に達し，飽和状態となっている．この状態でヒット率低下につながる要因をもちこめばそれに伴って性能が低下することは明らかである．スレッド・スロット数が8の場合の性能低下はこのような理由による．

また，構成Bの場合のバス使用率を**図12**に示す．プロセッサ・バスに関しては，4つのバンクのうちで最も高いバス使用率☆の値を示した．キャッシュをバンク分けしたことにより，プロセッサ・バスが飽和する状態はみられない．しかし，スレッド・スロット数が多いほどメモリ・バスのバス使用率が大きくなり，処理時間に大きな影響を与えている．スレッド・スロット数を4から8に増やしてもプロセッサ・バスのバス使用率が増大しないのは，メモリ・バス側の飽和によってキャッシュ・ミス処理の要求に待ちが生じているからである（従って，各スレッドの実行も中断されることが多くなる）．

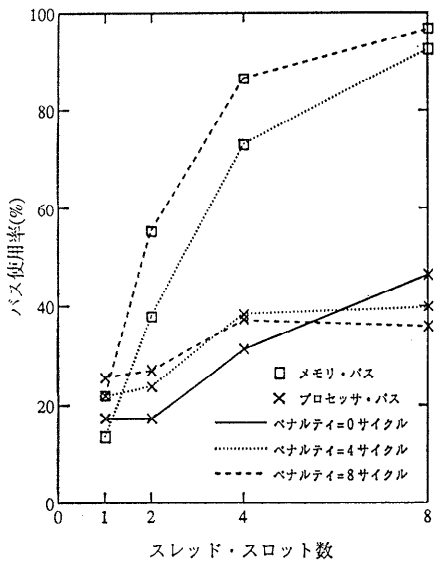


図12 バス使用率（構成B）
Fig. 12 Data bus utilizations (with configuration B).

☆ ただし，4つのバンク間でバス使用率に大きな差はない．

次に，4ウェイ・セット・アソシアティブ方式の8KBデータ・キャッシュを用いた場合の評価結果を**図13**に示す．いずれのスレッド・スロット数においてもヒット率はほぼ99.6%に達しており，ミス・ペナルティを課しても性能低下はわずかである．

現在のVLSI技術からみれば8KB程度のデータ・キャッシュを実装することは十分可能であり，キャッシュ・ヒット率の低下による性能低下を抑えることができる．構成Aでは，スレッド・スロット数を2, 4, 8とすることによって，それぞれ1.8倍，2.8倍，3.1倍の性能向上が得られる．スレッド・スロット数が4以上の場合の性能の伸び悩みはプロセッサ・バスの転送能力による制限が原因である．構成Bではその制限が緩和され，2スレッド同時実行でほぼ2倍，また，4，8スレッド同時実行でそれぞれ3.6倍，5.4倍の性能向上が得られる．各機能ユニットの使用率を調べたところ，8スレッド同時実行でさらに性能向上を達成するためには，整数ALUの追加が必要であることがわかった．

4. 関連研究

HEP[8]と同様に，サイクル単位で命令をインタリーブする方式は，MASA[9]や文献12)でも採用されている．これらのアーキテクチャでは，命令パイプライン

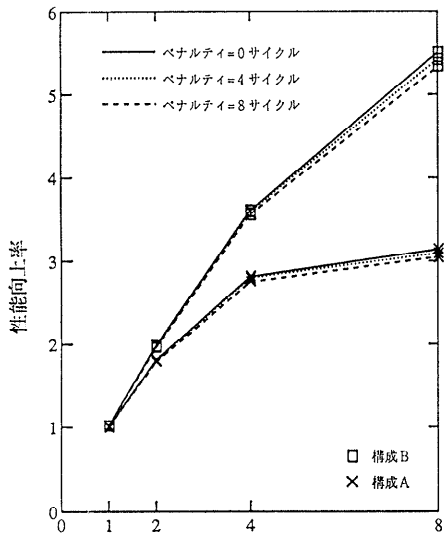


図13 性能向上率（容量8KB，4ウェイ・セット・アソシアティブ方式のキャッシュ使用時）
Fig. 13 Speedups with 8KB, 4-way set associative data cache.

中に同一命令流からの命令が複数存在することを禁止して命令間の動的なハザード検出の必要性を免れている．従って，パイプライン制御を簡単化できるという利点があるが，パイプラインの深さやメモリ・アクセス遅延に見合うだけの並列性が得られなければ十分なスループットを得ることはできない．FLATS 2[11] や DISC[14] では図2(a)と同様の命令パイプラインを採用しながらも同一命令流に対してパイプラインの先行制御を行い，より少ない命令流数でパイプラインを稼働状態に保つことを狙っている．また，Horizon[10]/Tera Computer[13] も HEP と同様の方式を採用しているが，インタリーブされる命令が VLIW 形式をとる点が異なる．

Farrens らが提案する方式[15]は図2(b)と考えられるが，単一の演算パイプラインを採用しており，1サイクル当たりの最大命令発行数は1である．

Prasadh ら[17]は図2(b)の構成のマルチスレッド方式と VLIW 方式とを組み合わせている．VLIW 命令の NOP フィールドの部分を他の命令流からの有用な操作で置き換える．各々の命令流が VLIW 形式であるために，命令フェッチに多大のバンド幅が要求される．

Daddis ら[16]や Keckler ら[18]は，マルチスレッド方式とスーパスカラ（または VLIW）方式とを組み合わせている．命令パイプライン構成は，基本的には図2(a)とみなせるが，Horizon のように完全にサイクル・インタリーブで各命令流の実行を行うのではなく，スーパスカラ方式における動的命令スケジュール機構を利用して，異なる命令流の命令を同時発行できる構成としている．

5. 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プロセッサ・アーキテクチャについて述べ，キャッシュの効果も含めたシミュレーションにより評価を行った．レイ・トレーシング・プログラムを用いた評価では，2，4，8 スレッド並列実行時でそれぞれ単一スレッド・プロセッサの 2.0 倍，3.6 倍，5.4 倍の性能向上が得られることを確認した．今後は，他の多くのアプリケーションを用いてアーキテクチャの評価を行う必要がある．

また，命令発行率（1サイクル当たりの命令発行数）の観点からアーキテクチャを再検討する必要もある．機能ユニット構成にも大きく依存するが，現状では，必ずしもスレッド・スロット数に十分見合うほどの命令発行率が得られていない．例えば，4スレッド同時実行時の命令発行率は 2.2 以下（シミュレーションによる）である．従って，図2(b)の構成に図2(a)の要素を組み合わせることにより，性能を低下させることなく（並列実行スレッド数はそのままで）ハードウェア量を軽減することができるであろう．特に，機能ユニット-レジスタ・セット間のスイッチに要するハードウェア量の削減は，アーキテクチャ実現の上で非常に重要である．

現在，プロセッサの詳細設計を進めるとともに，マルチプロセッサ・システムで用いる要素プロセッサとして，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．

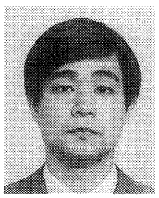
謝辞 本研究の初期の段階でご検討をいただいた松下電器産業(株)メディア研究所の西村明夫氏，中瀬義盛氏，日頃有益なコメントをいただく日高教行氏，浅原重夫氏の各氏に厚く感謝の意を表します．

参考文献

1) 鷲島ほか：並列図形処理，第3章，コロナ社 (1991).
2) Hirata, H. et al.: A Multithreaded Processor Architecture with Simultaneous Instruction Issuing, *Proc. of Intl. Symp. on Supercomputing, Fukuoka, Japan*, pp. 87-96 (1991).
3) Hirata, H. et al.: An Elementary Processor Architecture with Simultaneous Instruction Issuing from Multiple Threads, *Proc. of the 19th Annual Intl. Symp. on Computer Architecture*, pp. 136-145 (1992).
4) 平田ほか：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，情報処理学会研究会報告，92-ARC-94-2 (1992).
5) 平田ほか：多重スレッド・多重命令発行を用いる要素プロセッサ・アーキテクチャ，並列処理シンポジウム JSPP '92 論文集，pp. 257-264 (1992).
6) Denning, P. J.: The Working Set Model for Program Behavior, *Comm. ACM*, Vol. 11, No. 5, pp. 323-333 (1968).
7) Weber, W. D. and Gupta, A.: Exploring the Benefits of Multiple Hardware Contexts in a Multiprocessor Architecture: Preliminary Results, *Proc. of the 16th Annual Intl. Symp. on Computer Architecture*, pp. 273-280 (1989).
8) Jordan, H. F.: Performance Measurements on HEP—A Pipelined MIMD Computer, *Proc. of the 10th Annual Intl. Symp. on Computer*

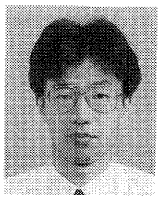
Architecture, pp. 207-212 (1983).
9) Halstead, R. H. and Fujita, T.: MASA: A Multithreaded Processor Architecture for Parallel Symbolic Computing, *Proc. of the 15th Annual Intl. Symp. on Computer Architecture*, pp. 443-451 (1988).
10) Thistle, M. R. and Smith, B. J.: A Processor Architecture for Horizon, *Proc. of 1988 Intl. Conf. on Supercomputing*, pp. 35-41 (1988).
11) 市川ほか：循環パイプライン計算機 FLATS 2, 情報処理学会研究会報告, 88-ARC-72-1 (1988).
12) 森下 巖：多段結合ネットワークを用いる超並列マシンのためのパイプライン化 MIMD プロセッサ, 情報処理学会論文誌, Vol. 31, No. 4, pp. 523-531 (1990).
13) Alverson, R. et al.: The Tera Computer System, *Proc. of 1990 Intl. Conf. on Supercomputing*, pp. 1-6 (1990).
14) Nemirovsky, M. D. et al.: DISC: Dynamic Instruction Stream Computer, *Proc. of the 24th Annual Intl. Symp. on Microarchitecture*, pp. 163-171 (1991).
15) Farrens, M. K. and Pleszkun, A. R.: Strategies for Achieving Improved Processor Throughput, *Proc. of the 18th Annual Intl. Symp. on Computer Architecture*, pp. 362-369 (1991).
16) Daddis, G. E. Jr. and Torng, H. C.: The Concurrent Execution of Multiple Instruction Streams on Superscalar Processors, *Proc. of the 20th Intl. Conf. on Parallel Processing*, pp. I: 76-83 (1991).
17) Prasadh, R. G. and Wu, C.: A Benchmark Evaluation of a Multi-Threaded RISC Processor Architecture, *Proc. of the 20th Intl. Conf. on Parallel Processing*, pp. I: 84-91 (1991).
18) Keckler, S. W. and Dally, W. J.: Processor Coupling: Integrating Compile Time and Runtime Scheduling for Parallelism, *Proc. of the 19th Annual Intl. Symp. on Computer Architecture*, pp. 202-213 (1992).

(平成4年9月2日受付)
(平成5年1月18日採録)



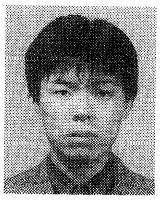
平田 博章（正会員）

昭和38年生．昭和62年京都大学工学部情報工学科卒業．平成元年同大学院修士課程情報工学専攻修了．同年松下電器産業(株)に入社，現在に至る．計算機アーキテクチャ，並列処理，コンピュータ・グラフィックス等の研究に従事．電子情報通信学会，ACM 各会員．



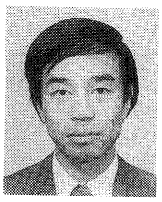
木村 浩三（正会員）

昭和36年生．昭和59年早稲田大学理工学部電気工学科卒業．同年松下電器産業(株)入社，現在に至る．マイクロプロセッサの研究開発に従事．電子情報通信学会会員．



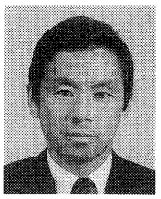
永峰 聡

昭和40年生．昭和63年京都大学工学部情報工学科卒業．平成2年同大学院修士課程情報工学専攻修了．同年松下電器産業(株)に入社，現在に至る．コンピュータ・グラフィックス，計算機アーキテクチャ等の研究に従事．



西澤 貞次（正会員）

昭和24年生．昭和47年大阪大学工学部通信工学科卒業．昭和49年同大学院修士課程通信工学専攻修了．同年松下電器産業(株)入社．現在同社メディア研究所画像第2開発室室長．昭和58～60年カーネギーメロン大学客員研究員．グラフィックスプロセッサ，画像生成装置の開発に従事．著書「並列図形処理」(共著，コロナ社)．電子情報通信学会，IEEE-CS 各会員．



鷲島 敬之（正会員）

昭和19年生．昭和41年大阪大学工学部電子工学科卒業．昭和43年同大学院修士課程電子工学専攻修了．同年松下電器産業(株)入社．現在同社メディア研究所所長．コンピュータ・グラフィックス，計算機アーキテクチャ等に興味を持つ．著書「並列図形処理」(共著，コロナ社)．電子情報通信学会，ACM, IEEE-CS 各会員．