計算機アーキテクチャ　104－13
システムソフトウェアと
オペレーティング・システム　62－13
（１９９４．　１．　２８）

画像生成用マルチスレッド・プロセッサのマイクロ・アーキテクチャ

木村浩三†　平田博章†　清原督三†　浅原重夫†
鷲島敬之†　尾上孝雄‡　白川功‡

†松下電器産業(株) メディア研究所　‡大阪大学 工学部 情報システム工学科
〒 571 大阪府門真市門真 1006　〒 565 大阪府吹田市山田丘 2-1

本稿では，高品位ＣＧ画像の高速生成を目的とした計算機システムの要素プロセッサとして，画像生成に特化したマルチスレッド・プロセッサのマイクロ・アーキテクチャとその開発手法について述べる．本開発手法では，ラジオシティ法のプログラムの命令実行トレース結果を分析して，動的な命令出現頻度，命令間の依存関係，分岐実行頻度などのプログラムの振舞いを詳細に解析し，それをもとにマイクロ・アーキテクチャを決定した．命令実行シミュレーションによりそのマイクロ・アーキテクチャを性能評価した結果，３スレッドを同時実行した場合には１スレッド実行に比べて 2.73 倍もの処理速度の向上を得ることができた．

Micro-architecture of Multi-threaded Processor for Image Generation

Kozo KIMURA†　Hiroaki HIRATA†　Tokuzo KIYOHARA†　Shigeo ASAHARA†
Takayuki SAGISHIMA†　Takao ONOYE‡　Isao SHIRAKAWA‡

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

‡Department of Information Systems Engineering,
Faculty of Engineering, Osaka University
Yamada-Oka, Suita, Osaka, 565 Japan

This paper describes a micro-architecture and a design approach of a multithreaded processor specialized for a base processor of parallel systems generating high quality CG image. The approach consists of 3 procedures, first, the detail characteristics of the program of radiosity method, (like instruction frequency, data- and control-dependency, and branch distance) are analyzed. After that, the micro-architecture is decided using the detail analysis, and then the performance of the multithreaded processor is evaluated. The result of the performance evaluation shows that the high utilization of function units brings the high performance for image generation.

1 はじめに

プロセッサの高性能化を実現するアプローチとしては，動作周波数の高速化や複数命令の同時発行がある．前者を利用するためにパイプラインを多段化して高性能化を図ったのがSuperpipeline方式であり，後者を利用するために機能ユニットを多重化して一度に複数の命令を実行するのがSuperscalarやVLIW方式である[1]．これらのアプローチはいずれも複数命令を同時に実行するためプログラムに内在する細粒度（命令）レベルの並列性を利用している．

しかしながら，命令間の依存関係，分岐実行やメモリアクセスによって細粒度レベルの並列性は抑えられるため，Superscalar，VLIWおよびSuperpipeline方式を利用したプロセッサでは飛躍的な性能向上は望めない[2, 3]．

この問題を解決する手法の一つとして，粗粒度レベルの並列性を利用して，複数のスレッド（並列実行可能な処理単位）を同時に実行するプロセッサ（以後，マルチスレッド・プロセッサと呼ぶ）の提案がなされている[4, 5, 6, 7, 8, 9, 10, 11]．マルチスレッド・プロセッサは複数のスレッドを同時に実行することにより，パイプライン実行時に発生するハザードを隠蔽して機能ユニットの稼働率を上げ，性能向上を図る．既に，我々は，画像生成や科学技術計算においてよく見られるように，独立に実行可能な複数のスレッドを発生するアプリケーションに対して，このマルチスレッド・プロセッサが有効であることは，命令実行シミュレーションにより明らかにしている[4, 5, 6, 7]．

現在、我々は，高品位ＣＧ(Computer Graphics)画像を高速に生成する計算機システムの要素プロセッサを開発している．ラジオシティ法[12]やレイ・トレーシング法[12]に代表される高品位ＣＧ画像の生成アルゴリズムには，粗粒度レベルの並列性は存在するが，細粒度レベルの並列性は低く抑えられている（詳細は第2章に後述）．本稿では，この要素プロセッサをマルチスレッド方式で実現する際のマイクロ・アーキテクチャとその開発手法について報告する．まず，ラジオシティ法のプログラムの命令実行トレース結果をもとに，動的な命令出現頻度，命令間の依存関係，分岐出現頻度や分岐先までの距離などを詳細に解析する．その解析結果をもとにマルチスレッド・プロセッサのマイクロ・アーキテクチャを決定し，命令実行シミュレーションにより評価を行なう．この開発手法により得られたマイクロ・アーキテクチャが3スレッド同時実行時に1スレッドと比較して2.73倍という優れたパフォーマンスを達成することを明らかにする．

2 マルチスレッド・アーキテクチャ

画像生成のアルゴリズムであるラジオシティ法やレイ・トレーシング法は，プログラムに内在する細粒度レベルの並列性が低く，複数の命令を同時に実行できる機会が比較的少ない．なぜならば，これらのアルゴリズムでは，描画される物体に対し交差判定を行なう部分が頻繁に現れるが，この交差判定処理は実行時に分岐方向が決定される条件分岐，再帰呼び出しされるサブルーチン呼び出し，および物体の形状データを扱うために必要となる間接データ・アクセスなどに占められているからである．一方，これらのアルゴリズムは，プログラムに内在する粗粒度レベルの並列性が高く，独立に実行可能なスレッドを発生することができる．ラジオシティ法ではパッチ（物体表面の一定の単位）ごとに受ける光のエネルギーを独立に計算することができ，レイ・トレーシング法では，画面のピクセルごとに輝度を独立に計算することが可能である．

ラジオシティ法やレイ・トレーシング法が持つ，細粒度レベルの並列性が乏しいという性質のため，通常用いられる複数命令の同時発行機構やパイプラインの多段化などのプロセッサの高速化技術では，機能ユニットの稼働率を上げることができず，プロセッサの持つ能力を最大限発揮することは難しかった．

これに対し，マルチスレッド・プロセッサは，スレッドごとの命令解読部を用意し，各機能ユニットにそれぞれ異なるスレッドの命令を供給する構成をとることにより，機能ユニットの稼働率を上げて処理性能を向上させる．すなわち，ラジオシティ法やレイ・トレーシング法のプログラムに内在する粗粒度レベルの並列性を生かして高いスループットを得ることができる．

図1は，機能ユニットを共有する形で3つの異なるスレッドの命令を同時に実行可能なマルチスレッド・プロセッサの構成例である．プロセッサ内において異なるスレッドの解読部からそれぞれ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

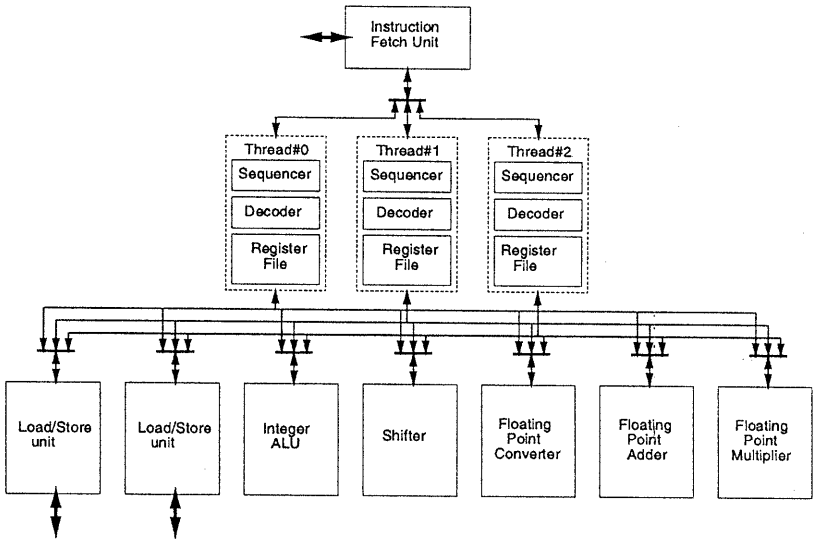

図 1: マルチスレッド・プロセッサの構成例

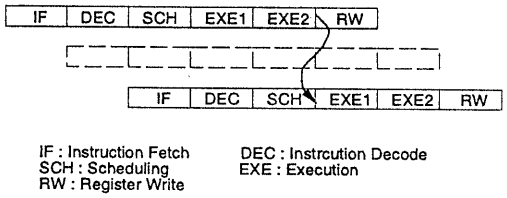

図 2: パイプラインのステージ構成

演算遅延を別のスレッドの命令実行によって隠蔽できる．なぜならば，異なるスレッドの命令間に依存関係は存在しないので，それぞれ異なる機能ユニットに命令が発行される場合には同時に実行できるからである．マルチスレッド・プロセッサのパイプラインのステージ構成の概略を図 2 に示す．

3 アプローチ

マルチスレッド・プロセッサをＣＧ画像生成専用のアーキテクチャに特化するためには，画像生成アプリケーション・プログラムの振舞いの解析と，命令実行シミュレーションによるプロセッサモデルの評価が重要である．なぜならば，アプリケーション・プログラムの振舞いを詳細に解析した結果にもとづいて，プロセッサ内部の機能ユニットの種類やその個数，各機能ユニットのパイプライン段数，命令ごとの演算遅延，分岐機構そしてキャッシュの構成を決定することができ，さらにそのプロセッサ・モデルが実際のアプリケーション・プログラムで有効かどうかを，命令実行シミュレーションで性能評価することにより確認できるからである．アプリケーションの解析とプロセッサ・モデルの命令実行シミュレーションによる評価を基本とする，筆者らが採用した開発手順を以下に示す．

Step1　Ｃ言語で書かれたラジオシティ法のプログラムについて，SPARC Ver.7 [1][13] の命令セットで実行命令のトレースを行ない，その結果から動的な命令出現頻度，命令間の依存関係，分岐命令の出現頻度や分岐先までの距離などの解析を行なう．

Step2　解析結果に基づいて，プロセッサ内部の機能ユニットの種類やその個数，機能ユニットのパイプライン段数，命令ごとの演算遅延，そして分岐機構を決定する．

Step3　このマイクロ・アーキテクチャが有効かどうかを調べるために，命令実行シミュレータ

[1]SPARC は SPARC International,Inc. の登録商標である

上で，ラジオシティ法のプログラムを実行し，性能の評価を行なう．有効性が満足できなければ，Step2 へ戻り，十分であれば終了する．

4 アプリケーションの解析

本章では，動的命令トレースにもとづき，ラジオシティ法のプログラムの振舞いを解析した結果について述べる．

4.1 命令の出現頻度

ラジオシティ法のプログラムを実行した時の命令の出現頻度を表 1 に示す．命令の分類については，同じ演算器の構成で処理可能な命令を同じ範疇とした．ロード／ストア命令と整数演算命令は動的出現頻度が他の命令と比べて高く，浮動小数点命令は，比較的出現頻度が低い．なお，浮動小数点除算命令は，乗算器を利用するアルゴリズムを用いることとして，浮動小数点乗算命令と同じ範疇とした．

通常，性能向上を図るには，動的出現頻度の高い命令群に着目し，頻度の高い命令を処理する機能ユニットの数を増加させたり，あるいはその機能ユニットのパイプラインを多段化させることにより実現できると思われている．しかし，実際には浮動小数点命令のような動的出現頻度の比較的低い命令群についても，その振舞いに注意し機能ユニットの構成を十分に考慮しなければならない．例えば，演算の遅延が非常に大きい命令の場合には，一つの命令に機能ユニットが長時間占有され他の命令がストールしてしまうため，性能を向上するには機能ユニットのパイプライン化が必要である．同じ機能ユニットを使用する命令の出現頻度が局所的に高くなる（命令出現に偏りがある）場合には，機能ユニットでの競合が頻繁に発生する可能性が高いため，機能ユニットの多重化が必須である．また，依存関係が発生する命令間の距離が平均的に短い場合には，後続する命令が発行できずにストールする機会が増加するため，機能ユニットの演算の高速化を図らなければならない．

命令の出現頻度の偏りを解析するために命令の出現頻度の動的変化を調べる．プログラムの実行時間に伴う命令群ごとの出現頻度の変化を図 3 に示す．横軸は，30 個のパッチから構成された 1 ブロック分のラジオシティ計算にかかる時間を 100 としてあらわし，縦軸はそれぞれの命令群の出現頻度を表している．処理の最初の部分ではデータの初期化が行なわれるため，ロード／ストア命令が多いが，光のエネルギー計算や交差判定で出現頻度が高くなる浮動小数点命令や分岐命令の出現頻度は低い．一方，ラジオシティ計算に費やされる時間のほとんどの部分では，それぞれの命令群の出現頻度の変化は少なくほぼ一定であり，この部分の出現頻度は，表 1 の数値とほぼ一致する．従って，表 1 の命令出現頻度が定常保たれていると考えて，最適なアーキテクチャを考案すればよい．

表 1: 命令の出現頻度

category of instructions	ratio(%)
load/store	34.83
integer	27.46
floating point (add,sub,cmp,abs,neg,cnv)	4.25
floating point (mul,div)	13.16
branch	12.18
others	8.12

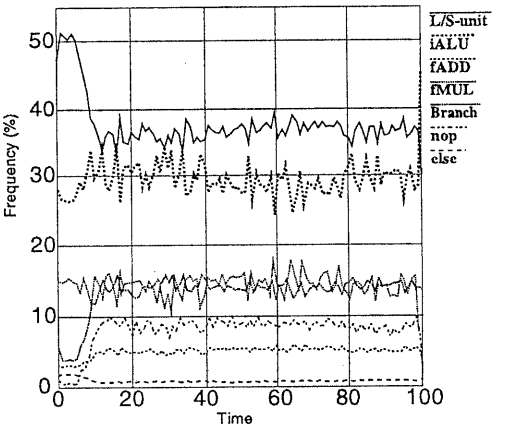

図 3: 命令出現頻度の変化

4.2 命令間の依存関係

依存関係が存在する命令間の距離が遠く離れている場合には，プロセッサの性能低下を引き起こさないだけでなく，複数の命令を同時に実行する機会が多いため，機能ユニットの細分化や多重化，パイプラインの多段化による性能向上が期待できる．反対

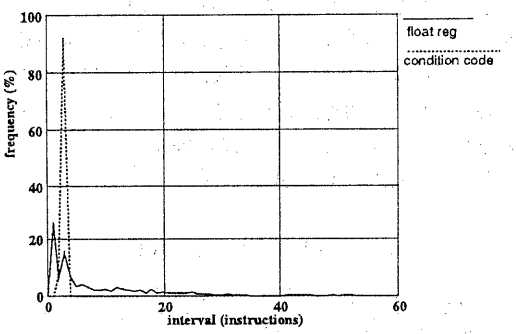

図 4: 依存を発生する命令間の距離

に命令間の距離が短い場合には，後続する命令がストールし性能を劣化させる．

浮動小数点演算命令は動的出現頻度が比較的低いものの，一般には演算遅延が大きいため依存関係が存在する命令間の距離には注意を払う必要がある．図 4 に，浮動小数点演算命令について，依存関係を発生する命令間の距離の頻度を示す．浮動小数点演算のコンディションコードに関して，依存関係を発生する命令間の距離は常に短いことがわかる．また，浮動小数点命令間のレジスタの依存関係においては，直前の命令の実行結果に依存しているものが 26.2% と最も多く，次いで，4 命令以内で依存しているものが 50% 以上，10 命令以内で依存しているものが 70% 以上を占めている．

4.3 分岐処理

分岐実行は，命令実行が非連続となるためパイプラインが乱れ，先行して読みだした命令が無効になるなど性能上大きな問題となる．従って，分岐命令の振舞いは，命令キャッシュの構成，命令フェッチや分岐処理の制御，およびパイプラインの構成を決定するのに大きな影響を及ぼす．まず、アプリケーション実行時における分岐実行／非実行（branch taken/not taken）の比率を表 2 に示す．分岐実行の方が非実行を上まわるが圧倒的に多いほどではない．

分岐命令が短い間隔で出現し，しかも分岐実行の頻度が高い場合には，多くの命令を先読みしても破棄する場合が多く命令先読みの効果が小さいが，反対に，出現間隔が長い場合には，多くの命令を先読

表 2: 分岐発生の頻度

taken/not taken	ratio(%)
taken	59.53
not taken	40.47

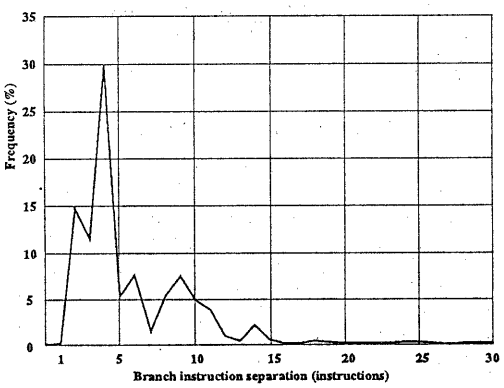

図 5: 分岐発生間隔

みし，さらに機能ユニットのパイプライン段数を深くすることにより，処理性能の向上が図れる．分岐命令がアプリケーション内でどのような間隔で出現するかを図 5 に示す．横軸は，分岐命令の出現した間隔を表し，縦軸には，その間隔で出現した分岐命令の比率を示す．これより 60% 以上の分岐命令が 5 命令以内の間隔で出現していることがわかる．

5 マイクロ・アーキテクチャ

本章では，ラジオシティ法のプログラムの振舞いの解析結果に基づき，ＣＧ画像生成に適したマルチスレッド・プロセッサのアーキテクチャを提案する．

5.1 機能ユニットの構成

第 4.1 節の結果より，ロード／ストア命令と整数演算命令の出現率はそれぞれ 34.83%，27.46% と平均して高いため，両方の命令群については，それぞれ独立に専用の機能ユニットを設けるべきと判断できる．浮動小数点演算命令は，17.41% と他の命令群と比べて比較的出現頻度が低いものの，浮動小数点演算の演算遅延は他の演算器と比べて一般的に大きく，また，命令の出現頻度にも偏りがあるため，

表 3: 機能ユニットの種類

function unit	category of instructions
load/store unit	load, store
integer unit	integer add/sub, logical, compare
floating point add unit	floating point add/sub, compare, absolute/negate, convert
floating point multiply unit	floating point multiply, divide

表 4: 機能ユニットの構成

function unit	the number of threads					
	1	2	3	4	5	6~
load/store unit	1	1	1	1~2	1~2	1~2
integer unit	1	1	1	1	1	1
floating point add unit	1	1	1	1	1	1
floating point multiply unit	1	1	1	1	1	1

浮動小数点加算ユニットと浮動小数点乗算ユニットの2種類の機能ユニットを用意すべきと考えられる．浮動小数点演算命令の出現頻度は，CG画像生成の場合には3次元データを扱うためX軸，Y軸，Z軸それぞれについて対象的に浮動小数点演算が必要となり，局所的に変化する．この考察に基づいて提案する機能ユニットの構成を表 3 に示す．ロード／ストア命令の出現比率は全命令のほぼ3分の1であるから，同時に実行するスレッド数が3の時までは，表 3 の機能ユニットを1つづつ備えた構成で，十分な性能が得られるが，スレッド数が4を越えた場合には，ロード／ストアユニットの稼働率が100% に近づき，それ以上スレッド数を増加させても性能の向上が得られなくなる事態が発生すると考えられる．その場合には，ロード／ストアユニットを多重化する必要を生じる．同時に実行するスレッド数を考慮に入れた機能ユニットの構成を表 4 に示す．

5.2 パイプライン

各機能ユニットのパイプライン段数は，第 4.2 節で解析した依存関係が存在する命令間の距離に大きく影響される．図 4 によると，浮動小数点命令で依存関係が存在する命令間の距離は2～4命令が最も多いので，通常はパイプライン段数が4を越えると急激に性能低下が引き起こされるはずである．マルチスレッド・プロセッサの場合には同時に複数のスレッドを実行するため，6段以内のパイプライン段数であれば，演算遅延を隠蔽することが可能と考えられる．提案するそれぞれの機能ユニットのパイプライン段数について表 5 に示す．

表 5: 実行パイプライン段数

function unit	category	latency
load/store unit	load	2
	store	3
integer unit	add/subtract	1
	logical	1
	compare	1
floating point add unit	add/subtract	5
	compare	5
	absolute/negate	1
	convert	5
floating point multiply unit	multiply	6
	divide	37

5.3 分岐処理

分岐実行は，パイプラインを乱し性能向上を阻害する大きな要因のため，分岐方向の予測による命令の先読みや投機的な命令の実行などの性能低下を抑止する手段が考えられている [14]．しかし，ラジオシティ法のプログラムでは，分岐方向が実行時にデータにより動的に決まるため分岐予測が極めて難しいことから，分岐の高速化のために投機的に分岐命令を実行したり，実行が不確かな命令を先読みしても性能向上は極めて少ないと思われる．むしろ，マルチスレッド・プロセッサの場合には，分岐による遅延が発生しても異なるスレッドの実行により遅延を隠蔽できることを利用して，予測が失敗して命令を廃棄する可能性のある先読みや機能ユニットの使用を避け，結果が確定するまで分岐処理は待たせる方が望ましいと考えられる．命令バッファは，図 5 の分岐命令の出現間隔から考慮して5命令～10 命令ぐらいが妥当と考えられる．

6 性能評価

6.1 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，提案したモデルについて性能評価を行なった．今回は，表

3 に示した，ロード／ストアユニット，整数演算ユニット，浮動小数点加算ユニットと浮動小数点乗算ユニットをそれぞれ1つづつそろえた基本構成と，表4に示した，ロード／ストアユニットの数を拡張した構成について評価した．それぞれの機能ユニットのパイプライン段数については，表5に示す．

アプリケーション・プログラムには，第4章で解析の対象とした，ＣＧ画像生成アプリケーションのラジオシティ計算の部分を使用し，それぞれのスレッドでは複数個のパッチに対するラジオシティ計算を行なう．一般に，ラジオシティ計算をする物体の大きさは不均一なため，構成するパッチの数も異なる．このため，スレッドごとに物体を割り付けたのでは，それぞれのスレッドの負荷が異なる．今回はスレッドの負荷を均一にするため，1スレッドについてパッチの数は30個とし，30個以上のパッチからなる物体については，複数のスレッドに分割してラジオシティ値を計算することにした．

6.2 評価結果

まず，表3に示した基本構成に対して，同時に実行させるスレッド数を変化させて命令実行シミュレーションを行なった．結果を図6に示す．横軸はスレッド数を示し，縦軸には，スレッド数が1の時に対する性能向上比を示す．この結果，基本構成のマルチスレッド・プロセッサにおいては，スレッド数の増加にともない性能向上が得られることが判明した．ただし，スレッド数が3までは，ほぼリニアに性能が向上しているが，スレッド数が4以上になると向上率が鈍ってしまう．これは，異なるスレッドのそれぞれの命令を機能ユニットに発行する場合，機能ユニットの競合が発生する頻度が増加してくるためと考えられる．続いて，表4を考慮してロード／ストアユニットの数を2，3と拡張した構成に対して，スレッド数を変化させて命令実行シミュレーションを行なった．結果を図7に示す．横軸は，ロード／ストアユニットの数を表し，縦軸は，基本構成でかつスレッド数が1の時に対する性能向上比を示す．同時に実行させるスレッド数が4以上の場合には，ロード／ストアユニットを増加させたことによる性能向上が得られる．これは，ロード／ストアユニットに対する機能ユニットの競合が緩和されたことによると，考えられる．

命令実行シミュレーション結果より，提案した基本構成において，1スレッドのみを実行させるプロセッサに対して，2スレッドで1.95倍，3スレッドで2.73倍，4スレッド以降ではロード／ストアユニットを2個に拡張した構成において，4スレッドで3.31倍，5スレッドで4.08倍の性能向上が得られることがわかった．

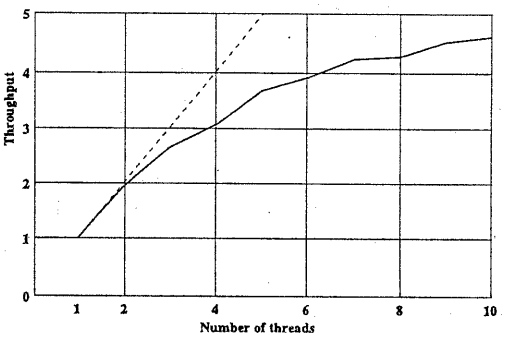

図6: スレッド数に対する性能向上比

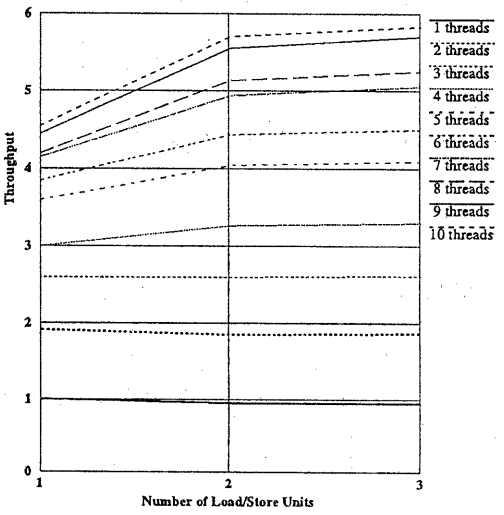

図7: 機能ユニット数に対する性能向上比

7 まとめ

今回、マルチスレッド・プロセッサをＣＧ画像生成専用に特化するために，ＣＧ画像生成アプリケーション・プログラムの振舞いを解析，その結果をもとにプロセッサ・アーキテクチャを決定，それが実際のアプリケーション・プログラムで有効であるか

どうかを命令実行シミュレーションによって確認する，という開発手法を用いた．

具体的には，まず，ラジオシティ法のCプログラムにおける動的な命令頻度，命令間の依存関係，そして分岐命令の出現頻度や分岐先までの距離などを解析し，プロセッサ内部の機能ユニットの種類やその個数，機能ユニットのパイプライン段数，命令ごとの演算遅延，分岐機構，そしてキャッシュの構成などを提案した．その後，提案したアーキテクチャのモデルをシミュレータでアプリケーション・プログラムを実行し性能を評価した．

命令実行シミュレーションの結果，今回決定したアーキテクチャは，3スレッドを同時実行させた場合には1スレッドに比べて2.73倍もの性能向上が得られることがわかり，高品位の画像を生成するラジオシティ法やレイ・トレーシング法に対して，極めて有効であることが明らかになった．

謝辞

本研究を行なうにあたり御助力いただきました，大阪大学 工学部 情報システム工学科 古賀拓也 氏（現在 日本電気（株）），松下電器産業（株）メディア研究所 望月義幸 氏，ならびに，日頃から貴重な御助言をいただきました松下電器産業（株）メディア研究所 日高教行 氏 の各氏に感謝いたします．

参考文献

[1] Hennessy, J. et al.: Computer Architecture a Quantitative Approach, Morgan Kaufmann Publishers (1990).

[2] Jouppi, N.P. and D.W.Wall: Available Instruction-level Parallelism for Superscalar and Superpipelined Machines, *Proc. of Third Int'l Conf. on ASPLOS*, pp. 272–282 (1989)

[3] Sohi, G.S. and S.Vajapeyam: Tradeoffs in Instruction Format Design for Horizontal Architectures, *Proc. of Third Int'l Conf. on ASPLOS*, pp. 15–25 (1989)

[4] Hirata, H. et al.: A Multithreaded Processor Architecture with Simultaneous Instruction Issuing, *Proc. of Int'l Symp. on Supercomputing, Fukuoka, Japan*, pp. 87–96 (1991).

[5] Hirata, H. et al.: An Elementary Processor Architecture with Simultaneous Instruction Issuing from Multiple Threads, *Proc. of the 19th Annual Int'l Symp. on Computer Architecture*, pp. 136–145 (1992).

[6] 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).

[7] Hirata, H. et al.: An Elementary Processor Architecture with Parallel Instruction Issuing from Multiple Threads, *Proc. of the Joint Symposium on Parallel Processing 1992*, pp. 257–264 (1992).

[8] Thistle, M.R. and Smith, B.J.: A Processor Architecture for Horizon, *Proc. of 1988 Int'l Conf. on Supercomputing* pp. 35–41 (1988).

[9] Daddis, G.E.Jr. and Torng, H.C.: The Concurrent Execution of Multiple Instruction Streams on Superscalar Processors, *Proc. of the 20th Int'l Conf. on Parallel Processing*, pp. I:76–83 (1991).

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

[11] Keckler, S.W. and Dally, W.J.: Processor Coupling: Integrating Compile Time and Runtime Scheduling for Parallelism, *Proc. of the 19th Annual Int'l Symp. on Computer Architecture*, pp. 202–213 (1992).

[12] Foley, J. et al.: Computer Graphics principles and practice, Addison-Wesley.

[13] SPARC RISC User's Guide, Cypress Semiconductor (1990).

[14] Smith, A. et al.: Branch Prediction Strategies and Branch Target Buffer Design, *Computer*, pp. 6–12 (Jan. 1984)