The University of Osaka
Institutional Knowledge Archive

Title	多重スレッド方式のプロセッサ・アーキテクチャおよびそのVLSI化設計に関する研究
Author(s)	木村, 浩三
Citation	大阪大学, 2001, 博士論文
Version Type	VoR
URL	https://doi.org/10.11501/3184510
rights	
Note	

The University of Osaka Institutional Knowledge Archive : OUKA

https://ir.library.osaka-u.ac.jp/

The University of Osaka

多重スレッド方式の
プロセッサ・アーキテクチャおよび
その VLSI 化設計に関する研究

2001年

木 村 浩 三

①

多重スレッド方式の
プロセッサ・アーキテクチャおよび
そのVLSI化設計に関する研究

2001年

木 村 浩 三

内容梗概

本論文は，著者が平成 4 年から平成 12 年にかけて，松下電器産業株式会社マルチメディア開発センターにおいて行った，画像生成に適した多重スレッド方式のプロセッサ・アーキテクチャとその VLSI 化設計に関する研究成果，ならびに平成 7 年から平成 10 年にかけて，大阪大学大学院工学研究科博士後期課程在学中に行った，多重スレッド方式プロセッサの高性能化のためのマイクロアーキテクチャに関する研究成果をまとめたものである．

画像生成 (コンピュータグラフィックス) の分野においては，膨大なデータを扱うとともに複雑な生成アルゴリズムを適用するために，極めて高い演算性能が求められている．プロセッサの高性能化については，アーキテクチャの改良や動作周波数の高速化の研究開発に，多くの研究者や技術者が携わっている．このうち，プロセッサにおける複数命令の同時実行は，高性能化を図る有効な手法としてよく知られており，特に，近年汎用プロセッサに採用されているスーパースカラおよび VLIW 方式は，命令レベルの並列性の利用により，同時実行可能な命令数を増やして性能向上を実現している．画像生成のアルゴリズムには，命令レベルの並列性および粗粒度レベルの並列性が内在し，これらはアプリケーションの広い範囲で容易に抽出可能であるため，画像生成の高速化に対してはスーパースカラや VLIW 方式は有効であるが，予測不可能な分岐や不要なコードによる実行時のオーバーヘッドのため，実効的に得られる性能の向上は抑えられている．

一方，粗粒度レベルの並列性を利用して高性能化を図る場合には，マルチプロセッサシステムの構成が適しており，プロセッサ数を増大させることによって，有効に性能向上が可能である．このような理由で，多くの高性能な画像生成スシステムではマルチプロセッサシステムの構成を採用しているが，マルチプロセッサシステムのシステム開発コストは高くなる．

これらの問題を解決するために提案された多重スレッドプロセッサは，機能ユニットやキャッシュなどのハードウェア資源を複数のスレッド (命令流) で共有し，単一プロセッサ内で複数のスレッドを同時並列に実行する．すなわち，アルゴリズムに内在する粗粒度レベルの並列性を利用して，複数命令の同時実行を可能にし，機能ユニットの稼働率を高めることにより，システム性能を向上させるものである．

本研究では，まず，画像生成に適した多重スレッドアーキテクチャおよびそのハードウェア構

成について，ハードウェアコストと性能の観点から考察する．次に，アプリケーションプログラムの解析と性能評価シミュレーションからなる，マイクロアーキテクチャの構築手法を提案し，それに基づいた多重スレッドプロセッサのマイクロアーキテクチャについて考察する．さらに，キャッシュミスヒットによる性能への影響に対して，ノンブロッキング制御方式をスレッド間に限定して適用する手法を考察する．

本論文は，全 6 章から構成される．

第 1 章では，序論として画像生成アルゴリズム処理の特徴と従来のプロセッサやシステムで実行する場合の課題について述べ，本研究の背景と目的を明らかにするとともに研究内容と成果について概説する．

第 2 章では，第 1 章の課題を解決するために提案した多重スレッドアーキテクチャのハードウェア構成について述べ，シミュレーションと実装設計から性能とハードウェアコストについて述べる．

第 3 章では，画像生成のアプリケーションプログラムの解析と性能評価シミュレーションからなる，マイクロアーキテクチャの構築手法を提案し，それに基づいたマイクロアーキテクチャについて考察する．

第 4 章では，キャッシュの容量や連想度が十分でない場合にも，ミスヒットペナルティによる性能劣化を抑止できる，スレッド間ノンブロッキングキャッシュ制御方式の提案と評価を行い，その有用性について述べる．

第 5 章では，本研究で得られた成果を要約し，今後に残された課題について述べ，結論とする．

関連発表論文

I. 学会論文誌発表論文

(1) 木村浩三, 奥畑宏之, 尾上孝雄, 清原督三, 鷺島敬之, 白川功: “マルチスレッドプロセッサのデータキャッシュ制御方式,” 映像情報メディア学会誌, Vol. 52, No. 5, pp. 742–749 (1998 年 5 月).

(2) T. Onoye, T. Masaki, I. Shirakawa, H. Hirata, K. Kimura, S. Asahara, and T. Sagishima: “High-level synthesis of a multithreaded processor for image generation,” *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, Vol. E78-A, No. 3, pp. 322–330 (March 1995).

(3) 平田博章, 木村浩三, 永峰聡, 西澤貞次, 鷺島敬之: “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,” 情報処理学会論文誌, Vol. 34, No. 4, pp. 595–605 (1993 年 4 月).

II. 研究会等発表論文（査読付）

(1) K. Kimura, K. Yoshioka, T. Kiyohara, T. Sagishima, and I. Shirakawa: “Microarchitecture of multithreaded processor for computer graphics,” in *Proc. IEEE Int'l Conference on Neural Networks and Signal Processing*, pp. 1586–1589 (December 1995).

(2) T. Onoye, T. Masaki, H. Hirata, K. Kimura, S. Asahara, T. Sagishima, I. Shirakawa, S. Tsukiyama, and S. Shinoda: “High-level synthesis of a multithreaded processor for image generation: design and simulation,” in *Proc. European Simulation Multiconference*, pp. 948–953 (June 1994).

(3) T. Sagishima, K. Kimura, H. Hirata, T. Kiyohara, S. Asahara, T. Onoye, and I. Shirakawa: “Multithreaded processor for image generation,” in *Proc. IEEE Int'l Symposium on Circuits and Systems*, pp. 4.231–4.234 (June 1994).

(4) K. Kimura, H. Hirata, T. Kiyohara, S. Asahara, T. Sagishima, T. Onoye, and I. Shirakawa: "Evaluation method of micro-architecture for multi-threaded processor," in *Proc. IEEE Int'l Symposium on Industrial Electronics* pp. 53–58 (May 1994).

(5) T. Onoye, T. Masaki, H. Hirata, K. Kimura, S. Asahara, T. Sagishima, I. Shirakawa, S. Tsukiyama, and S. Shinoda: "High-level synthesis of multithreaded processor based image generator," in *Proc. IEEE Int'l Symposium on Industrial Electronics*, pp. 47–52 (May 1994).

(6) 正城敏博, 尾上孝雄, 平田博章, 木村浩三, 浅原重夫, 鷺島敬之, 白川功: "画像生成用多重スレッドプロセッサの高位合成手法による設計," 電子情報通信学会回路とシステムワークショップ, pp. 61–66 (1994 年 4 月).

(7) H. Hirata, K. Kimura, S. Nagamine, A. Nishimura, Y. Nakase, and T. Nishizawa: "An elementary processor architecture with parallel instruction issuing from multiple threads," in *Proc. of the Joint Symposium on Parallel Processing 1992*, pp. 257–264 (June 1992).

(8) 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," in *Proc. of the 19th Annual Int'l Symp. on Computer Architecture*, pp. 136–145 (May 1992).

III. その他研究会等発表論文

(1) 木村浩三, 平田博章, 清原督三, 浅原重夫, 鷺島敬之, 尾上孝雄, 白川功: "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," 情報処理学会研究会報告, ARC 104–13, OS 62–13, pp. 97–104 (1994 年 1 月).

(2) 正城敏博, 尾上孝雄, 平田博章, 木村浩三, 浅原重夫, 鷺島敬之, 白川功: "多重スレッドアーキテクチャを採用した画像生成用プロセッサの設計 ～高位論理合成システムによる実現～," 電子情報通信学会技術研究報告, CAS 93–99, Vol. 93, No. 432, pp. 67–74 (1994 年 1 月).

(3) 尾上孝雄, 正城敏博, 平田博章, 木村浩三, 浅原重夫, 鷺島敬之, 白川功: "多重スレッドアーキテクチャを採用した画像生成用プロセッサの設計 ～詳細アーキテクチャの設計～," 電子情報通信学会技術研究報告, CAS 93–98, Vol. 93, No. 432, pp. 61–66 (1994 年 1 月).

(4) 尾上孝雄, 正城敏博, 古賀拓也, 藤井貴晴, 平田博章, 木村浩三, 白川功: "高位論理合成システムによる画像生成用プロセッサの設計," 電子情報通信学会秋季大会, A-65 (1993 年 9 月).

(5) 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: "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, ARC 94–2, pp. 9–16 (1992 年 6 月).

目 次

第1章 序論 1

1.1 背景 1

1.1.1 3次元画像の生成手法 1

1.1.2 画像生成における並列処理 5

1.2 本論文の概要 9

第2章 多重スレッドアーキテクチャ 11

2.1 緒言 11

2.2 多重スレッドアーキテクチャ 11

2.2.1 高性能化のアプローチ 11

2.2.2 ハードウェア構成と動作 14

2.3 性能評価 18

2.3.1 ベンチマークプログラム 18

2.3.2 シミュレーション 19

2.3.3 実装設計 20

2.4 評価結果 25

2.5 結言 27

第3章 多重スレッドプロセッサのマイクロアーキテクチャ 29

3.1 緒言 29

3.2 マイクロアーキテクチャの決定手法 30

3.2.1 決定手法の流れ 30

3.2.2 シミュレーション環境 31

3.3 アプリケーションの解析 33

3.4 評価対象のモデル 38

3.5 評価結果 40

3.6 結言 42

第4章 多重スレッドプロセッサのキャッシュ構成 43
4.1 緒言 43
4.2 キャッシュミスによるペナルティの削減 43
4.2.1 多重スレッドプロセッサにおける課題 43
4.2.2 従来手法によるキャッシュ制御方式 44
4.2.3 スレッド間ノンブロッキングキャッシュ制御方式 45
4.3 シミュレーション 46
4.3.1 評価環境およびシミュレーションモデル 46
4.3.2 ベンチマークプログラム 48
4.4 評価結果 50
4.4.1 キャッシュ容量と連想度の増加によるヒット率の向上 50
4.4.2 ブロッキングキャッシュ制御方式による性能劣化 50
4.4.3 スレッド間ノンブロッキングキャッシュ制御方式による性能向上 51
4.5 結言 55

第5章 結論 57

謝辞 59

参考文献 61

第1章

序論

1.1 背景

画像生成 (コンピュータグラフィックス) の分野においては，より現実に近い画像を生成することを目標として様々な手法が構築されているが，膨大な量のデータを扱うとともに複雑な生成アルゴリズムを適用するために，極めて高い演算性能が要求される．このため，多数の研究者や技術者は高品位な画像を実時間で生成することを究極の目的として，高品位画像生成アルゴリズム，高速化アルゴリズム，および，それらを実現するアーキテクチャについて長年にわたって研究開発を行なっており，現在も留まることなく続けられている．

本節では，画像生成の主流となっている 3 次元画像の生成法ならびにマルチプロセッサシステムなどに代表される並列化ハードウェアによる高速な実現方法について述べる[1,2]．

1.1.1 3 次元画像の生成手法

3 次元画像の生成に関する研究の歴史は古く，1960 年代の Roberts の隠線消去アルゴリズム[3]などのワイヤフレーム・メッシュモデルの研究にさかのぼる．その後，1970 年前後から物体の “線” ではなく “面” を対象としたいわゆるサーフェスモデルに関する研究がなされ始めた．Warnock による隠れ面処理[4]，Gouraud や Phong によるシェーディングモデル[5,6]，および，Catmull によるテクスチャマッピング[7]などがその代表例であり，これらの開発により CG 画像の現実感が大幅に改善された．さらに，1980 年前後からは物体の体積情報やその他の物理情報などを駆使した，ソリッドモデルによる高品位な 3 次元画像生成に関して研究開発が行われた．特に，物体に対する光線の反射・屈折・透過を扱う Whitted のレイ・トレーシング法 (Ray-tracing)[8]や拡散反射を扱う ラジオシティ法 (Radiosity)[9]などの光学的性質を活用した画像生成はその高い表現性から，現在でも主幹技術として用いられている．表 1.1 に 3 次元画像生成の歴史とその形状モデルについて示す[2]．

表 1.1: 3 次元画像の生成の歴史

年代	モデル名	構成要素
1960	Wireframe model	点，線
1970	Surface model	点，線，面 (表面のみ)
1980	Solid model	点，線，面 (中身の情報を持つ)

次に，3 次元画像の一般的な生成過程を以下に示す．

1. モデリング

モデリングは，3 次元空間内に存在する物体を計算機で扱える表現で記述する作業である．ここでは，代表的なモデルとしてのソリッドモデルを構築する手法の一つである，CSG (constructive solid geometory) モデル[10] について説明する．複雑な形状を持つ物体を定義するために，CSG モデルでは，形状プリミティブの集合演算を繰り返し用いる．この形状プリミティブは，直方体，三角柱，円柱，球などのように式による表現が単純な形状を持つものを用いる．このような形状プリミティブに対する集合演算を木構造で表現することにより，モデリング処理を行なう (図 1.1)．CSG モデル法は，Octree 法や Voxel 法との併用により計算機環境で容易に利用することができる．

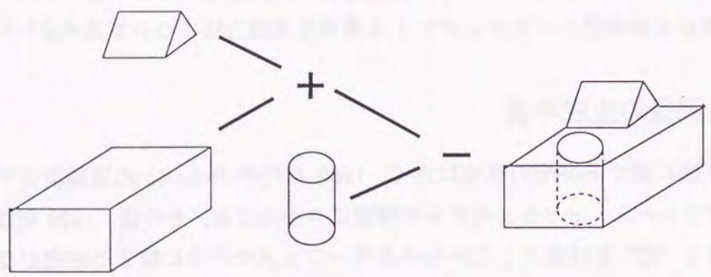


図 1.1: CSG(Constructive solid geometry) モデルで表現した物体の一例

2. 幾何学処理

3 次元図形に対する操作として，平行移動，回転，スケーリングなどが挙げられるが，これらの操作はそれぞれ式 (1.1)，(1.2)，(1.3) で表現できる．

$$T(d_x, d_y, d_z) = \begin{bmatrix} 1 & 0 & 0 & d_x \\ 0 & 1 & 0 & d_y \\ 0 & 0 & 1 & d_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{1.1}$$

$$S(s_x, s_y, s_z) = \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{1.2}$$

$$R_z(\theta) = \begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{1.3}$$

これまで考えてきた物体は，モデルの記述にローカル座標系 (モデリング座標) を用いており，これらの物体を実際に 3 次元空間に配置する作業を行なう必要がある．この配置はローカル座標からワールド座標系に変換することにより実行ができるが，この変換をモデリング変換という．また，ワールド座標系から視点を原点とした座標系 (視点座標系) に変換することにより，3 次元空間を 2 次元画面に投影することが可能になる．この座標変換を視野変換と呼ぶ．

幾何学変換処理では複数の変換が行われるが，これらは全て平行移動，回転マトリックスの乗算などの組合せにより行なうことができる．

3. レンダリング

3 次元空間中に定義された物体に光源から光を照射して，それらを特定の視点から眺めた場合の現実感のある画像を生成する処理がレンダリング処理である．形状モデルに加えて物体表面の光学的特性や光源を記述するシェーディングモデルを利用して，視点から可視となる物体面を決定する隠れ面処理，影をつける影処理，可視点の色や明るさを算出する輝度計算，画像の質を改善するためのアンチエイリアシングなどの処理を行なう．

以下に，代表的なレンダリングのアルゴリズムであるレイ・トレーシング法およびラジオシティ法について説明する．

- レイ・トレーシング法

レイ・トレーシング法は，スクリーン上のピクセルごとにレンダリング処理を行う手法であり，ピクセルを通過して視点に入る光線を逆追跡し，その光線を放出する物体表面を同定することにより隠れ面処理を行い，さらにその輝度を計算するという手法である．本手法は，光線と物体との交差判定演算により，光線と交点を持つ物体の中でもっとも視点に近い物体を求める．本手法の最大の特徴は，反射，透過や影処理を含む輝度計算を，交差判定という極めて単純な演算の繰り返しにより統一的に扱っていることである．レイ・トレーシング法の概要を図 1.2 に示す．

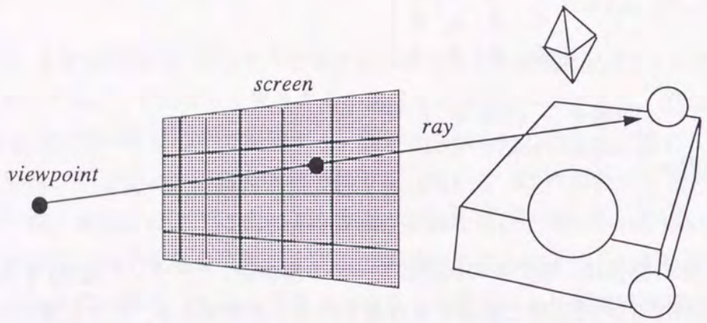


図 1.2: レイ・トレーシング法の概要

- ラジオシティ法

従来の輝度計算モデルでは，間接光を一定値として取り扱ってきたが，散乱光による影響が考慮されていないため，特に室内空間を対象とした画像生成を行う際には，現実感に欠ける．これに対し，ラジオシティ法は，相互乱反射を取扱うことにより，現実感の高い画像を生成するアルゴリズムである．本アルゴリズムでは，まず物体表面をメッシュに区切ったパッチに分割した後，各々のパッチが受けた光エネルギーの一部を拡散反射により放出し，その 2 次反射光の一部が他のパッチを照らす．このような拡散反射の繰返しによるエネルギーの平衡状態は次のような連立方程式 (1.4) で表される．

$$B_i = E_i + \sigma_i \sum_{j=1}^{N} B_j F_{ji} \frac{A_j}{A_i} \tag{1.4}$$

B_i, B_j:パッチ$_i$, パッチ$_j$の各々のラジオシティ

E_i:パッチ$_i$が放出するエネルギー

σ_i:パッチ$_i$の反射率

A_i, A_j:パッチ$_i$, パッチ$_j$の各々の面積

N:パッチの総数

ここで，パッチ i がパッチ j から受け取ったエネルギーであるフォームファクタ F_{ji} は，式 (1.5) で与えられる．この式を用いて，すべてのパッチのラジオシティを計算し輝度を求めることにより，相互反射を考慮した画像を得ることができる．ラジオシティ法の概要を図 1.3 に示す．

$$F_{ji} = \frac{1}{A_j} \iint_{A_j, A_i} \frac{\cos\phi_j \cos\phi_i dA_j dA_i}{\pi r^2} \tag{1.5}$$

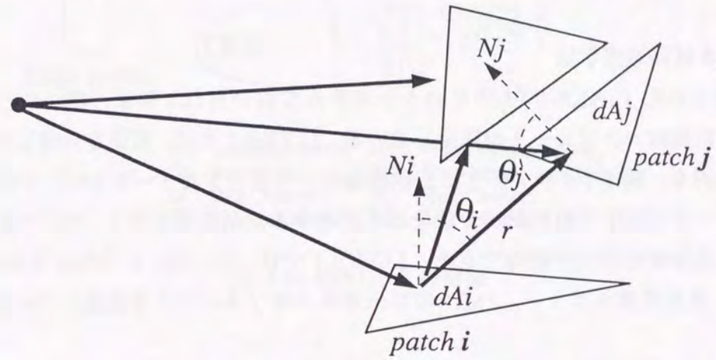


図 1.3: ラジオシティ法の概要

4. ディスプレイ

3 次元空間を投影した 2 次元画面を画面上に表示する処理である．フレームメモリの読みだし制御を行なうことにより，メモリ内の各ピクセルを水平方向にスキャンする．CLUT (color lookup table) を用いて色変換を行ない，必要な場合にはマルチウインドウ制御を行なうことにより，それぞれの画素に対し信号を合成し，ディスプレイに表示する．

1.1.2 画像生成における並列処理

上記に述べた 3 次元画像生成の処理は，処理とデータの流れが単一方向であることが多く，各処理過程を実行するための個々のハードウェアを用意し，パイプライン処理することにより，

高速化を図ることができる[11,12,13,14,15,16,17]．このなかで，圧倒的に演算量が大きいのはレンダリング処理であり，画像の生成速度はこのレンダリングの処理速度に大きく依存する．

本節では，レンダリング処理，特に上記のレイ・トレーシング法およびラジオシティ法の並列処理による高速化手法について考察する．

1. レイ・トレーシング法の並列処理

前述した通り，レイ・トレーシング法の処理のほとんどが物体と光線の交差判定処理である．このことから，処理の高速化のために，1. 交差判定の回数を減らすこと，2. データ構造の工夫などにより交差判定処理自体を高速化すること，が必要である．また，レイ・トレーシング法は本質的に演算量が多いため，並列処理による高速化が実用上不可欠である．並列処理アーキテクチャとして以下のアプローチが提案されている．

a) 画面分割による並列処理手法

阪大で開発された LINKS-1 [11,12]というシステムで用いられて以来，幾つかのレイ・トレーシング用並列コンピュータが採用している．LINKS-1 では，画面を分割し各ピクセルに関する処理を，図 1.4に示すように星状構造のノードコンピュータ (NC) で行い，ルートコンピュータ (RC) で動的負荷分散をはじめとする統括処理を行う．NC の台数に比例した画像生成速度の向上が期待できる．LINKS-1 では，NC/RC を Z8000 制御ユニット，i8086/8087 算術演算ユニット，1MB のローカルメモリユニットで構成している．

b) 専用プロセッシングエレメントによる高速化手法

同じく阪大で開発された LINKS-2 は，レイ・トレーシングに特化したプロセッシングエレメントを採用しており，以下のようなアーキテクチャ上な特徴を有する．

- 32 ビット固定長データフォーマットによるアドレス演算の簡素化
- 116 ビットの固定長命令による命令コードの簡単化
- 3 アドレス演算ユニットによる入出力アドレスの並列演算
- インタリーブ方式の 4 バンク構成のデータメモリと 3 組の内部バスをクロスバスイッチで接続
- プロセッサ間通信ユニットの採用による通信オーバヘッドの削減

この他にも，3 次元空間の分割による並列化手法などが提案されている．

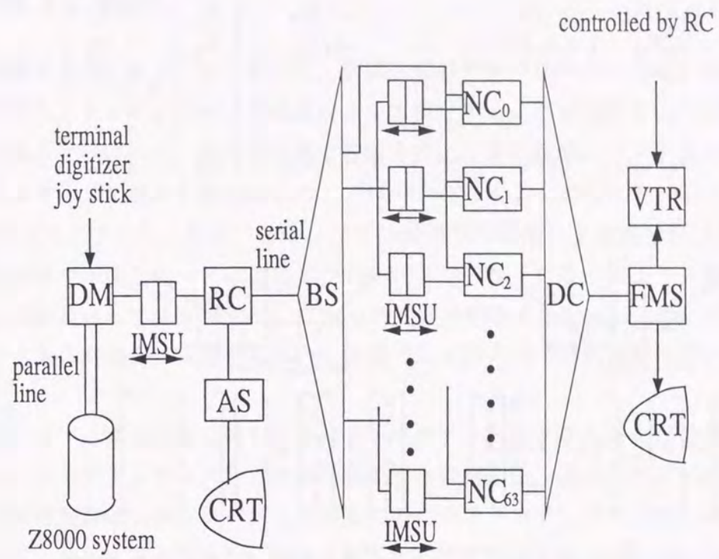


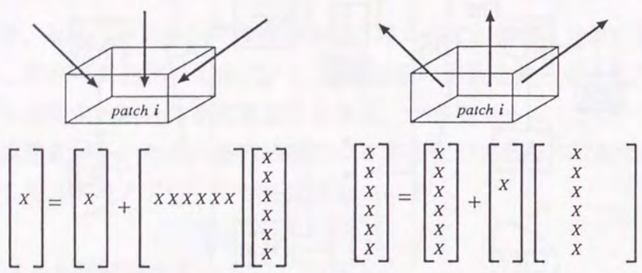
図 1.4: LINKS-1 の構成

2. ラジオシティ法の並列処理

ラジオシティ法では相互反射光の照度計算を行なう際に，各パッチでのラジオシティの収集 (gather) と放出 (shoot) を並列に処理することができる．実際には，式 (1.6) に示すように，ガウスザイデル法の反復処理でパッチ$_i$の放出するラジオシティを他のパッチに与え，各パッチで他のパッチから放出されたラジオシティを収集し，近似解を求める (図 1.5)[18]．

各パッチの単位反復処理は，各々独立に行なうことができ，したがって，マルチプロセッサ構成のシステムがラジオシティ法適用可能である．ラジオシティ法を取扱うシステムである，松下電器の SIG2 [13,14]は PE (Processing Element) を 36 台バス結合したマルチプロセッサシステムである (図 1.6)．

$$\begin{bmatrix} 1-\sigma_1 F_{11} & -\sigma_1 F_{12} & \cdots & \cdots & -\sigma_1 F_{1n} \\ -\sigma_2 F_{21} & 1-\sigma_2 F_{22} & \cdots & \cdots & -\sigma_2 F_{2n} \\ \vdots & \vdots & \cdots & \cdots & \vdots \\ \vdots & \vdots & \cdots & \cdots & \vdots \\ -\sigma_n F_{n1} & -\sigma_n F_{n2} & \cdots & \cdots & 1-\sigma_n F_{nn} \end{bmatrix} \begin{bmatrix} B_1 \\ B_2 \\ \vdots \\ \vdots \\ B_n \end{bmatrix} = \begin{bmatrix} E_1 \\ E_2 \\ \vdots \\ \vdots \\ E_n \end{bmatrix} \tag{1.6}$$



$B_i = E_i + \sum_{j=1}^{N} (\sigma_i F_{ij}) B_j$ $\qquad$ $B_j = B_j + B_i(\sigma_j F_{ji}),\ F_{ji} = F_{ij}\ A_i/A_j$ *for all j*

(a) gather $\qquad$ ***(b) shoot***

図 1.5: ラジオシティの収集と放出

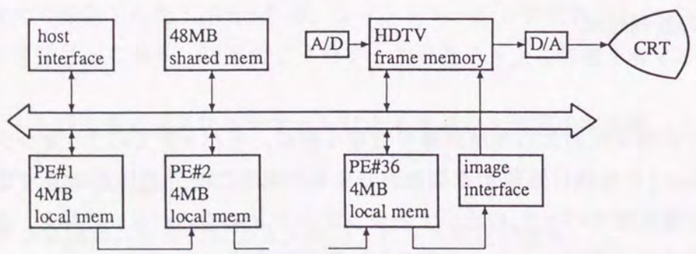


図 1.6: SIG2 の構成

1.2 本論文の概要

高品位な画像を高速に生成するためには，画像生成のアルゴリズムに内在する粗粒度レベルの並列性を利用し，マルチプロセッサ構成によって実行することは前節で述べた．プロセッサ数を増加させることによって，極めて有効に性能を向上できる反面，システムコストは高くなる．一方，プロセッサにおける高性能化のアプローチとしてよく知られているスーパースカラおよび VLIW 方式のプロセッサは，命令レベルの並列性の利用による高速化は有効であるが，予測不可能な分岐や不要なコードによる実行時のオーバーヘッドなどにより，実効的に得られる性能の向上は抑えられている[19,20,21]．これらの問題を解決するために，単一プロセッサ内で複数のスレッドを同時並列に実行するという多重スレッド方式プロセッサ・アーキテクチャを提案する．

本研究では，まず，画像生成に適した多重スレッドアーキテクチャおよびそのハードウェア構成について，ハードウェアコストと性能の観点から考察する．次に，アプリケーションプログラムの解析と性能評価シミュレーションに基づいたマイクロアーキテクチャの構築手法を提案する．さらに，キャッシュミスヒットによる性能劣化に対処するため，ノンブロッキング制御方式をスレッド間に限定して適用する手法を考察する．

第 2 章では，提案した多重スレッドアーキテクチャのハードウェア構成について述べ，シミュレーションと実装設計から性能とハードウェアコストにおける有効性について述べる．

第 3 章では，画像生成のアプリケーションプログラムの解析と性能評価シミュレーションに基づいて，マイクロアーキテクチャの構築手法を考察する．

第 4 章では，キャッシュの容量や連想度が十分でない場合にも，ミスヒットペナルティによる性能劣化を抑止できる，スレッド間ノンブロッキングキャッシュ制御方式の提案を行い，その有用性について評価する．

第 5 章では，本研究で得られた成果を要約し，今後に残された課題について述べ，結論とする．

第2章

多重スレッドアーキテクチャ

2.1 緒言

本章では，まず，多重スレッドアーキテクチャによる3次元画像生成の高速化手法と，そのハードウェア構成について述べる．次に，性能評価シミュレーションと実装設計を行ない，性能向上とハードウェアコストの観点から多重スレッドアーキテクチャの有効性について考察する．

2.2 多重スレッドアーキテクチャ

2.2.1 高性能化のアプローチ

一般的に，画像生成アルゴリズムには，“命令レベルの並列性” および “粗粒度レベルの並列性” が内在し，アプリケーションの広い範囲においてこれらは容易に抽出可能である[1,2]．このため，スーパースカラやVLIW方式のプロセッサを用いた高速化は容易であるが，予測不可能な分岐や不要なコードによる実行時のオーバーヘッドなどのため，実効的に得られる性能向上は抑えられている[19,20,21]．例えば，ラジオシティ法やレイ・トレーシング法などのアルゴリズムでは，描画される物体に対し交差判定が頻繁に出現し，しかもこの交差判定処理実行時に分岐方向を決定しなければならないため，複数命令の同時発行機構やパイプラインの多段化(スーパーパイプライン)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が上がらず，プロセッサの持つ能力を最大限発揮することは難しい．

一方，粗粒度レベルの並列性をアプリケーションから容易に抽出できる場合には，性能向上を図るための手段として，通常マルチプロセッサシステムが利用される．これは，プロセッサの数を増やすことによって，比較的容易に性能向上を図れるからであり，現在，高性能画像生成システムの多くはマルチプロセッサシステムを採用している[11,12,13,14,2,15,16,17]．例えば，3次元コンピュータグラフィックスのアプリケーションからは，粗粒度レベルの並列性は容易に

抽出でき，ポリゴンごとの座標変換 (geometrical transformation) や輝度計算 (lighting) の処理は同時並列に実行できる．ラジオシティ法ではパッチ (物体表面の一定の単位) ごとに独立に受光エネルギーの計算ができ，レイ・トレーシング法では，画面のピクセルごとに独立に輝度計算が可能である．マルチプロセッサシステムの場合には，これらのポリゴン，パッチ，および，ピクセル単位の処理を要素プロセッサに順次割り付けて並列実行することにより，処理の高速化が実現できる．

このような理由で，多くの高性能画像生成システムではマルチプロセッサシステムを採用しているが，ハードウェア規模が巨大化するとともに，システムコストが高くなるという問題点を抱えている．

この問題を解決するために，単一プロセッサ内で複数のスレッドを同時に実行可能な多重スレッドプロセッサが提案されている[22,23,24,25]．多重スレッドプロセッサは，機能ユニットやキャッシュなどのハードウェア資源を複数のスレッド (命令流) で共有し，単一プロセッサ内で複数のスレッドを同時並列に実行する．即ち，アルゴリズムに内在する粗粒度レベルの並列性を利用して，複数命令の同時実行を可能にし，機能ユニットの稼働率を高めることにより，システム性能を向上させるものである．

図2.1および図2.2に，マルチプロセッサシステムと多重スレッドプロセッサの概念図を示す．マルチプロセッサシステムでは，上述したように，各要素プロセッサごとにポリゴン，パッチ，および，ピクセル単位の処理を実行する．通常はこれらの処理をさらに高速化するために，要素プロセッサが同時実行可能な複数の機能ユニットを備えているが，実際には予測不可能な分岐や不要なコードなどにより，機能ユニットの稼働率は飛躍的には上がらない場合が多い．一方，多重スレッドプロセッサの場合には，プロセッサ内で共有している機能ユニットを使用して同時実行する．これにより，機能ユニットの稼働率を向上させ，単一プロセッサで性能向上が実現できる．[1]

[1]図中の網かけの部分が稼働している機能ユニットを表している．マルチプロセッサシステムにおいて，各要素プロセッサ内で機能ユニットA，B，Cのみが稼働している状態とすると，多重スレッドプロセッサにおいて，すべての機能ユニットA，B，Cが稼働している状態と処理性能は変わらないことになる

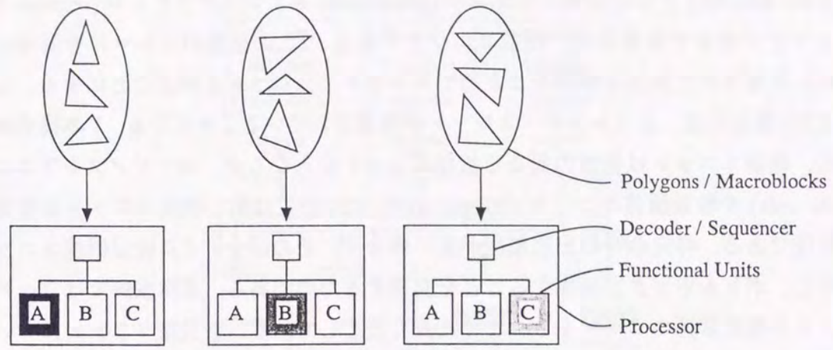


図 2.1: マルチプロセッサシステムの概念図

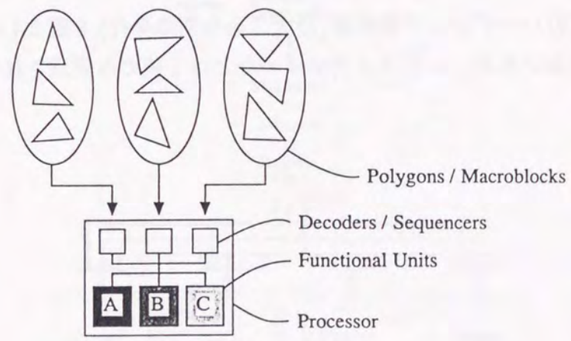


図 2.2: 多重スレッドプロセッサの概念図

2.2.2 ハードウェア構成と動作

図 2.3 に，3つの異なるスレッドを同時に実行可能な多重スレッドプロセッサの構成例を示す．スレッドごとに独立に用意する資源としては，命令解読部 (decoder)，およびプログラムカウンタ (program counter) を含む命令シーケンサ (sequencer) とレジスタファイル (register file) である．スレッドで共有する資源は，機能ユニットである．(この複数のスレッドを同時に実行できるために用意された資源や機構をまとめてスレッド・スロットと呼ぶことにする．したがって，図 2.3の場合には，3スレッド・スロットが用意されていることになる．) 本構成例では，基本的に，機能ユニットは複数の異なる機能ユニットからなるが，ロード／ストアユニット (load/store unit) や整数演算ユニット (integer unit) については同じ機能ユニットを複数設けることも可能である．特定命令の出現頻度が高い場合に，その命令による特定機能ユニットの処理が集中し，ボトルネックが発生することを回避するためである．本構成例ではロード／ストアユニットと整数演算ユニットを2つまで実装可能としている．各機能ユニットについては，スループットを上げるためにパイプライン化しており，ユニット間では独立に命令を実行できる構成をとる．命令キャッシュについては，全く異なるアドレスに位置する複数の命令を同時にアクセスすることはハードウェア上高コストになる．このため，各スレッド・スロットごとに命令フェッチバッファを備え，各スレッド・スロットが命令キャッシュから順番に複数の命令を読みだすことにより，命令フェッチユニット (instruction fetch unit) を複数のスレッドで共有する．

多重スレッドプロセッサに対し，複数命令の同時発行という観点から類似している，スーパースカラ方式のプロセッサのハードウェア構成例 (最大3命令同時発行) を図 2.4 に示す．同時に単一のスレッドしか扱わないため，レジスタファイルなどは1組のみ用意されている．

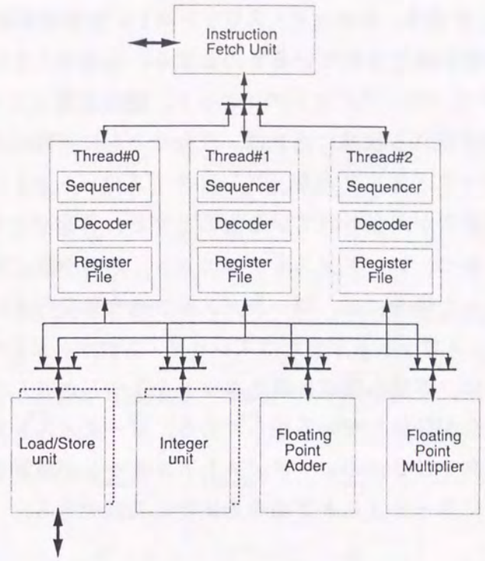


図 2.3: 多重スレッドプロセッサのハードウェアの構成例

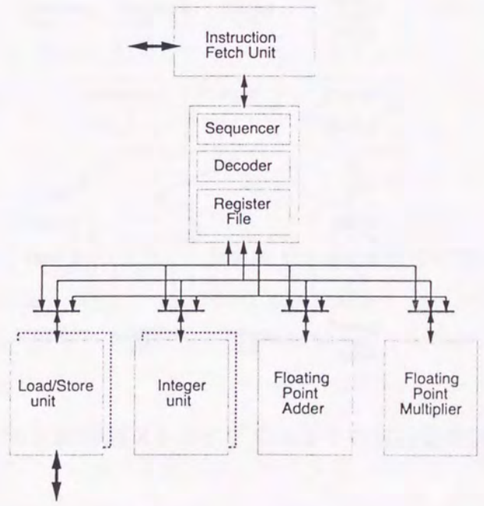


図 2.4: スーパースカラプロセッサのハードウェアの構成例

本構成例で，図2.5を用いて多重スレッドプロセッサでの命令実行動作を説明する．いま，スレッド・スロット＃0にロード命令，スレッド・スロット＃1に整数演算命令，スレッド・スロット＃2に浮動小数点加算命令が用意されているものとする．各命令ともにスレッド・スロット内での依存関係がなく，かつ，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がともに命令を受け取れる状態にあれば，3命令ともに同時に発行される．次に，スレッド・スロット＃0にロード／ストア命令，スレッド・スロット＃1，スレッド・スロット＃2ともに浮動小数点加算命令が用意されているものとする．各命令ともにスレッド・スロット内での依存関係がなく，かつ，ロード／ストアユニット，浮動小数点加算ユニットがともに命令を受け取れる状態であった場合には，ロード／ストア命令は発行されるが，2つの浮動小数点加算命令は，スレッド・スロット＃1またはスレッド・スロット＃2のどちらかの命令が選択されて発行される．これは，浮動小数点加算ユニットを1つしか有していないため，2つのスレッド・スロット間で競合が発生しているからである．ロード／ストア命令を同時に2命令発行する場合には，本構成例では2つのロード／ストアユニットが用意されているため，2つのスレッド・スロットともにロード／ストア命令を同時に発行できる．

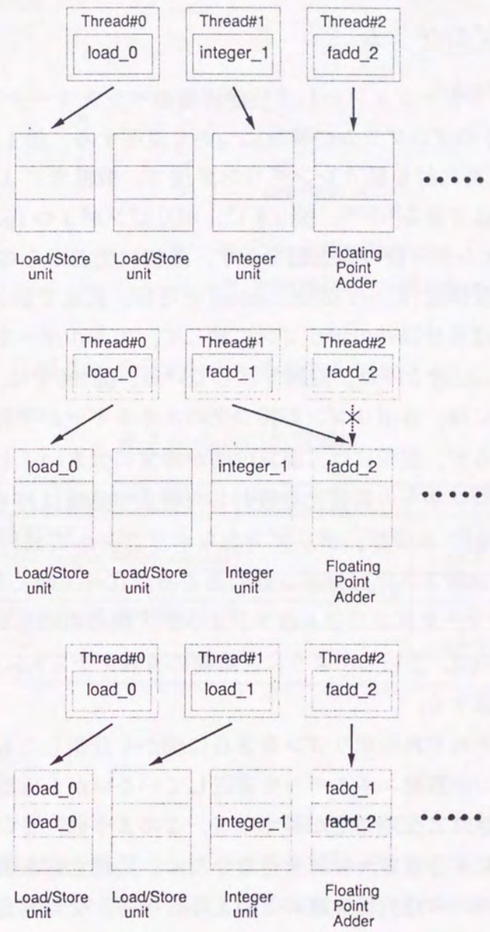


図 2.5: 多重スレッドプロセッサの実行動作例

2.3 性能評価

本節では，多重スレッドプロセッサならびにスーパースカラプロセッサの性能について，アプリケーションプログラムを用いた性能評価結果について述べる．

2.3.1 ベンチマークプログラム

まず，ターゲット・アプリケーションとして性能評価のベンチマークプログラムに用いた，ラジオシティ法 (Radiosity) のプログラムの特徴について説明する．第 1.1.1小節で述べたように，ラジオシティ法は相互乱反射を扱うレンダリング法で，散乱光による影響も考慮するため，現実間の高い画像を生成できる[1,2,26]．図 2.6 に，ポリゴンが 3 つ (polygon A, polygon B, polygon C) の場合のラジオシティ法の概念図を示す．最初に光源から各ポリゴンに光のエネルギーを放射する．それを直接光 (direct illuminance) と呼び，実線で示している．続いて，エネルギーを受けたポリゴンは自分以外のポリゴンに対して，エネルギーを放射する．それを間接光 (interreflection illuminance) と呼び，点線で示している．図 2.6では，ポリゴン A からの間接光を表している．正確には，各ポリゴンが持つ光のエネルギーが平衡状態になるまで，エネルギーの放射は行なわれるが，実際には，演算時間が非常に大きいことから，比較的エネルギーの大きい幾つかのポリゴンからの間接光を放射した時点で処理は打ち切られる．

光源からポリゴンへの放射，および，ポリゴンからポリゴンへの放射において，他のポリゴンによってエネルギーが遮蔽された部分は，他のところに比べて暗くなる (polygon B の上部)．本研究で用いたベンチマークプログラムはラジオシティ法の中でもレイキャスティング法 (ray-casting) と呼ばれるもので，上述したように遮蔽物の有無をチェック (intersection test) しながら放射エネルギーを計算する．

実際のプログラムでは，それぞれのポリゴンをさらに細かく分割したもの（エレメント）に対して，自分以外のポリゴンが放射エネルギーを遮蔽していないかを確認しながらエネルギー計算をするため，必然的に演算と交差判定が多くなる．このように，ラジオシティ法のプログラムでは，エレメントごとにエネルギー計算を行なうため，処理全体を多数のスレッドに分割することができ，粗粒度レベルの並列性は高いといえる．一方，交差判定処理による条件分岐の発生頻度が高く，かつその分岐方向はポリゴンやエレメントに依存するため，命令レベルの並列性は実効的に低く抑えられる．

ベンチマークプログラムとして用いたラジオシティプログラムの特性を把握するために，実行時の命令出現頻度を表 2.1に示す[27]．命令セットは Sun Microsystems 社の SPARC[2]アーキテクチャに準拠している[28]．

[2]Scalable Processor ARChitecture の略

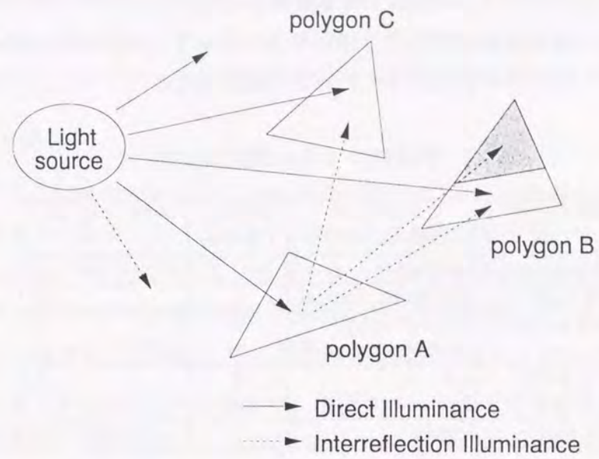


図 2.6: ラジオシティ法のアルゴリズム

表 2.1: 命令の出現頻度

category of instructions	ratio(%)
load/store	33.2
integer (arithmetic/logical)	33.9
floating point (add/sub/cmp/abs/neg/cnv)	8.8
floating point (mul/div)	6.7
branch	12.6
others	4.9

2.3.2 シミュレーション

命令実行のパイプライン動作をシミュレーションするとともに，LSI 実装設計を行い，多重スレッドプロセッサの性能評価とハードウェアコストを含めた有効性を示す．同時に，複数の命令を実行できる点とハードウェア構成が類似している点より，スーパースカラ方式のプロセッサについても性能評価を行なう．

性能評価を行なうモデルの各機能ユニットの構成 (種類，担当する命令，パイプライン段数，実装数) について，表 2.2 に示す．上記構成において，同時発行命令数の増加にともなう性能向上を評価する．多重スレッドプロセッサにおける発行命令数は，同時に扱うスレッド数を意味する．

表 2.2 に示すモデルは，性能評価に使用するラジオシティ法のプログラムの動的な命令出現

頻度より決定した (表 2.1)．また，発行命令数を増加した場合には一部の機能ユニットの使用効率が 100% に近づき，性能向上の劣化が予想される．そこで，出現頻度の高い命令を実行する機能ユニットについては複数個実装するモデルも用意する．

表 2.2: 機能ユニットの構成と役割

function unit	category of instructions	latency (cycles)	number of unit
load/store unit	load/store	3	1-2
integer unit	arithmetic/logical	1	1-2
floating point adder	add/sub/cmp/ abs/neg/cnv	3	1-2
floating point multiplier	mul div	3 28	1
branch unit	branch	2	1/instruction

2.3.3 実装設計

Verilog-HDL を使用して，3スレッドプロセッサを RTL レベルで記述し，論理合成およびレイアウト設計を行ない，LSI 実装を試みた．論理合成ツールは，Synopsys 社製 *Design Complier* を用い，VLSI Technology 社の供給する 0.6μm 3 層メタル CMOS スタンダードセル (*VSC450*) にテクノロジーマッピングを行ない，その後，Soliton Systems 社製のレイアウト設計ツール *Compass Design Navigator* を用いて，レイアウト設計を行なった．

以下に，3スレッドプロセッサに実装した機能ユニットの詳細を述べ，その構成を表 2.3に示す[29,30]．

- 命令フェッチ

  プロセッサは3組の命令キュー・命令解読ユニットを有し，独自のプログラム・カウンタに基づいたスレッドの制御を行なう．

  各スレッド・スロット毎に命令キューを配し，命令フェッチ・ユニットを複数のスレッド・スロットで共有する．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なう．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なう．

  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/命令発行

  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，それ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なう．公平な命令選択を実現するために，一定サイクル（巡回間隔）を経るごとに優先度を巡回シフトして再割り当てする．

- 実行演算

  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，ほとんどの場合，レジスタ・セットのポート数は3（読み出し:2，書き込み:1）で充分である．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．本 LSI 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に分離）が必要である．

  また，整数演算ユニットについては，命令の出現頻度が高いが，ロード／ストアユニットに比べて規模が小さくかつ論理的な複雑さが低いことから各スレッド・スロットに1個づつ実装した．

図2.7に3スレッドプロセッサのフロアプランを，表2.4に3スレッドプロセッサの諸元を示す．

表 2.3: 実装した機能ユニットの構成

unit	number of unit
load/store unit	1
integer unit	1/thread
floating point adder	1
floating point multiplier	1
decoder sequencer(branch unit)	1/thread
register file	1/thread
instruction fetch	1

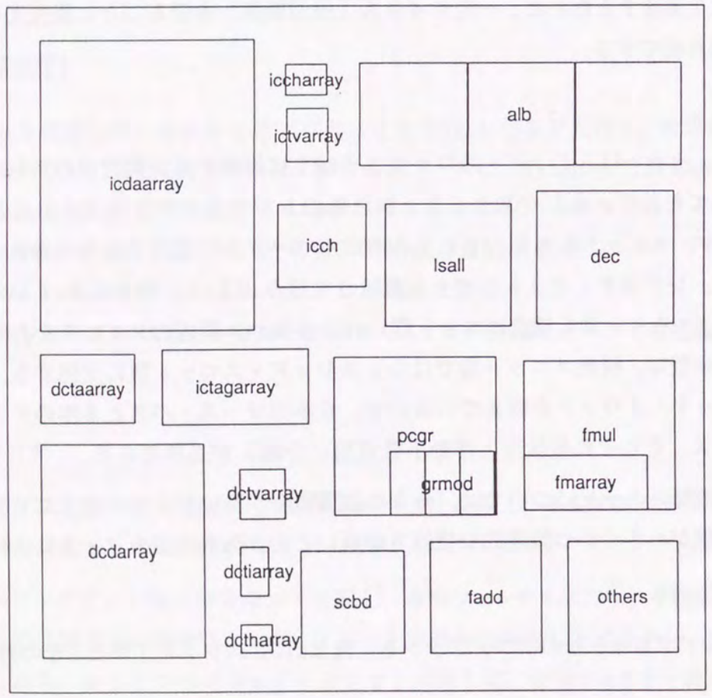


図 2.7: 3スレッドプロセッサのフロアプラン

表 2.4: 3スレッドプロセッサの諸元

テクノロジ	0.6 μm 3層メタル CMOS
サイズ (I/O パッドを含まず)	12.2 mm x 12.1 mm
ハードウェア規模	Logic 130K gates + Memory 12K bytes
動作周波数	66 MHz
性能	MAX 200 MIPS (3 threads)

LSI 実装において，レイアウト設計結果より算出したコア部の各ユニットの相対面積比を表2.5に示す[31]．さらに，この面積比を用いて，多重スレッドプロセッサにおけるスレッド・スロット数や機能ユニットの構成を変化させたモデル，および，スーパースカラプロセッサにおける同時発行命令数や機能ユニットの構成を変化させたモデルについて，チップの面積比 (プロセッサコア部) を概算した結果を図2.8に示す．命令解読部とレジスタファイルについてはスレッド・スロットごとに実装するため，同時実行スレッド数が M のモデルのチップ面積は，単一スレッドプロセッサの面積＋ (M-1) × (命令解読部とレジスタファイルの面積) で近似する．縦軸は，各機能ユニットを1個づつ実装し，同時に実行できるスレッドもしくは命令が1であるモデル (基本モデルと呼ぶ) に対する面積比を表しており，横軸は，同時に実行できるスレッドもしくは命令の数を表している．モデルBおよびCは，1つの整数演算ユニットと1つのロード／ストアユニットを実装したモデルAに対して，各々整数演算ユニットとロード／ストアユニットを追加実装したものであり，モデルDは，両ユニットともに追加実装したものである．ここで，“multi-” および “super-” は，各々多重スレッドプロセッサおよびスーパースカラプロセッサを意味している．

表 2.5: ユニットの面積比 (プロセッサコア部)

unit	size ratio [%]
load/store unit	10
integer unit	7
floating point adder	11
floating point multiplier	16
decoder sequencer(branch unit)	19
register file	6
instruction fetch	8
others	23
Total	100

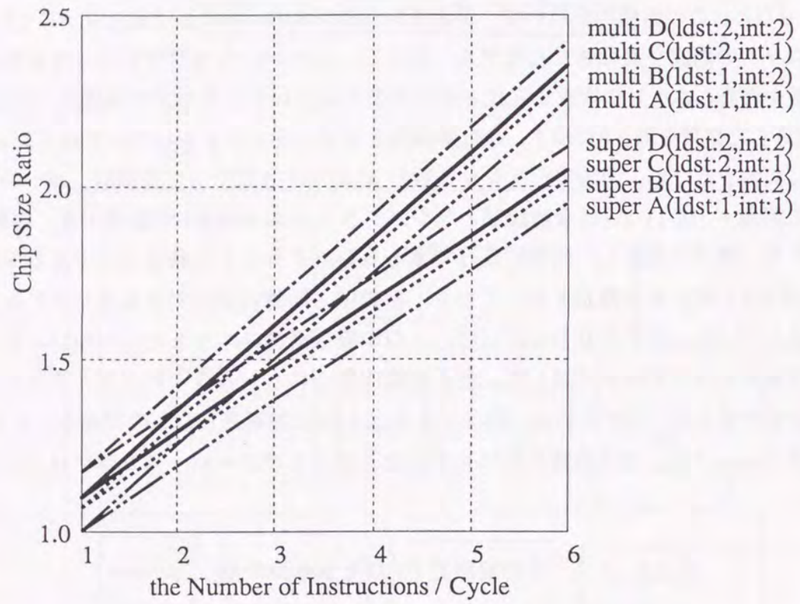


図 2.8: 同時発行命令数とチップ面積比 (プロセッサコア部)

2.4 評価結果

ラジオシティ法のプログラムを使用して，多重スレッド方式とスーパースカラ方式について，機能ユニットの種類と個数を変化させたモデルの命令レベルでのシミュレーションを行なった．同時発行命令数に伴う性能向上比を図 2.9に示す[31]．データキャッシュのヒット率は理想状態 (100%) と仮定し，縦軸の性能向上比は，命令発行数 1 の単一スレッドプロセッサ (同時実行スレッド数が 1) の性能を 1 として正規化した．表 2.2 のモデルの場合，多重スレッド方式では，スレッド数の増加による性能向上は 3〜5 倍に達するが，スーパースカラ方式では，命令数の増加による性能向上は最大 1.8 倍で留まる．このように，多重スレッド方式は，高スループットを達成することより，粗粒度レベルの並列性を十分に活用できる効率的なアーキテクチャといえる．

多重スレッド方式においては，3 スレッド・スロットまで，すべてのモデルがほぼ線形に増加していくが，3 スレッドを越えた時点で，ロード／ストアユニットと整数演算ユニットがともに 1 個のモデルが，他のモデルと比べて向上が鈍くなる．一方，ロード／ストアユニットと整数演算ユニットがともに 2 個実装しているモデルは，6 スレッドを同時に実行する時にも性能向上は飽和しない．これは，単一スレッドでロード／ストア命令と整数演算命令がそれぞれ全体の 1/3 近くを占めているため，ロード／ストアユニットと整数演算ユニットの使用効率が限界に達したことによるものと考えられる．一方，スーパースカラ方式では，同時発行可能な命令数が低く抑えられていると考えられ，機能ユニットの追加による性能向上はほとんど見られない．これより，多重スレッド方式は，プロセッサの持つハードウェアの能力を 100%近くまで発揮できる，効率の良いアーキテクチャといえる．

図 2.10に，プロセッサコア部 (図 2.8) の単位面積当たりの性能比を示す．多重スレッドプロセッサは，面積当たりの性能比の観点から，スーパースカラ方式などの単一スレッドプロセッサと比べ高い効率を実現している．一般に，性能向上実現のためのハードウェアの追加や変更は効率的でなく，“N%のハードウェア面積の増加により N%の性能向上を達成すること”，すなわち 1.0 倍の単位面積当たりの性能比を維持すること，は難しいとされている．しかしながら，同時実行スレッド数 3 の多重スレッド方式のプロセッサは，1.5 倍以上の単位面積当たりの性能比を達成している．

また，図 2.10のグラフの頂点は，面積当たりの性能比の観点より，最もハードウェアコスト効率の良い同時実行スレッド数を表わしており，3 次元 CG アプリケーションの場合，機能ユニットの構成にも依存するが同時実行スレッド数が 3〜5 の多重スレッドプロセッサが最適といえる．

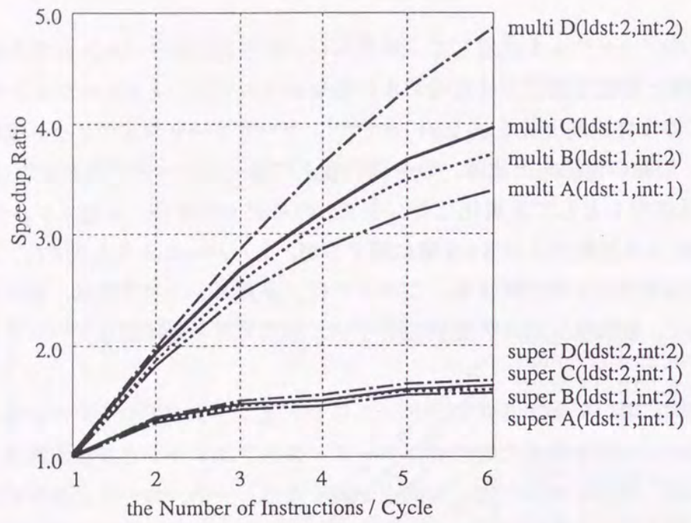


図 2.9: 同時発行命令数と性能向上比

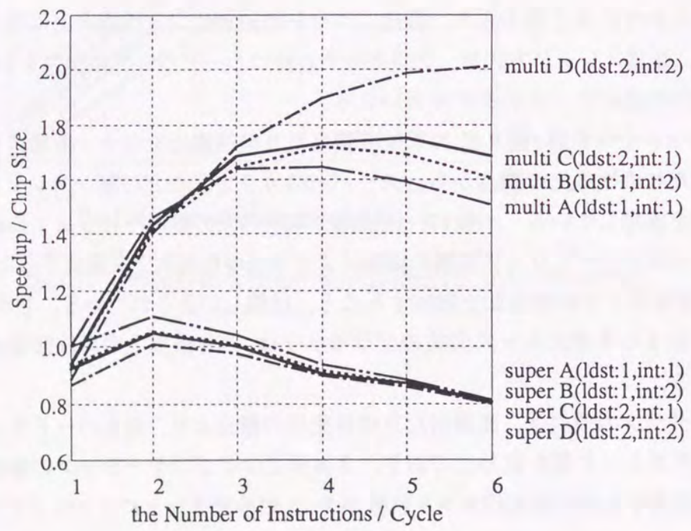


図 2.10: 同時発行命令数と単位面積あたりの性能向上 (性能向上/チップ面積)

2.5 結言

本章では，多重スレッドアーキテクチャの概念，ハードウェア構成の概要について述べ，さらに，性能評価とLSI実装設計を通して，ハードウェアコストの観点から多重スレッドアーキテクチャの有効性について報告した．特に，機能ユニットの構成が類似しているとともに，高速化の手法としてよく知られているスーパースカラプロセッサと比較することにより，高品位ＣＧ画像生成システムの要素プロセッサに多重スレッド方式が有効であることを示すことができた．

性能評価については，ＣＧアプリケーションの代表例の一つであるラジオシティ法のプログラムを用いて，多重スレッド方式とスーパースカラ方式に対して命令レベルのシミュレーションを行なった．スレッド数の増加による性能向上が５倍に近づき，高いスループットを達成することより，多重スレッド方式は粗粒度レベルの並列性を十分に活用できる効率的なアーキテクチャといえる．また，多重スレッド方式については，動的な出現頻度の高い命令を実行する機能ユニットを追加することにより，大幅な性能向上が得られることも示されている．これより，多重スレッド方式は，プロセッサの持つハードウェアの能力を100%近くまで発揮できる，ハードウェア効率の良いアーキテクチャといえる．

LSI実装設計を通して評価した対コスト性能比については，単位面積当たりの性能比が同時実行スレッド数の増加に伴って向上することから，単一スレッド実行のプロセッサに比べ，極めて効率の良いアーキテクチャであることを示すことができた．さらに，動作周波数向上による性能向上を考えた場合，パイプライン段数の増加が引き起こす実行サイクル数の増加 (例えば，分岐やデータ依存時に発生するハザードと呼ばれるオーバーヘッドなど) が比較的少ないため，多重スレッド方式は周波数向上に適しているといえる．また，多重スレッド方式のプロセッサに必要な命令間の依存検出用ハードウェアは，スーパースカラ方式プロセッサのそれと比較して簡単であるため，多重スレッド方式はクリティカルパス発生の観点からも周波数高速化による性能向上を実現しやすいアーキテクチャである．

第3章

多重スレッドプロセッサのマイクロアーキテクチャ

3.1 緒言

多重スレッド方式のプロセッサは，複数のスレッドの命令を同時に実行し，依存関係によるアイドル・スロットを隠蔽することにより，機能ユニットの稼働率を向上させ性能向上を図る．このように，多重スレッドプロセッサのアーキテクチャは性能向上に有効であるが，複数のスレッドが相互の依存関係なく実行されるため，内部動作が複雑である．このため，多重スレッドプロセッサの詳細な性能の予測は，その他のプロセッサと比べて非常に難しい．多重スレッドプロセッサの詳細な性能評価には，実際のプロセッサに近い内部動作をシミュレーションできる環境が不可欠である．しかしながら，このシミュレーション環境が整っていたとしても，考えられるマイクロアーキテクチャのバリエーションをシミュレーションしたり，また，思考錯誤を繰り返しながらマイクロアーキテクチャを変更していく手段を取る場合には，性能向上を抑止する原因の特定が難しく，ターゲットに適するマイクロアーキテクチャの決定に膨大な時間がかかる．

本章では，上述した課題を解決するために提案した多重スレッドプロセッサのマイクロアーキテクチャを構築する手法を述べ，それに基づいたマイクロアーキテクチャとその有効性について記述する．

3.2 マイクロアーキテクチャの決定手法

3.2.1 決定手法の流れ

今回提案した多重スレッドプロセッサのマイクロアーキテクチャを構築する手法について，そのステップごとの処理を示し、同時にその流れを図 3.1 に示す[27].

Step1 ターゲットプログラムの実行をトレースし，動的な命令列を作成する．命令列を動作解析し，シングルスレッドでの動的な命令の出現頻度，命令間の依存関係の距離などプログラムの振る舞いを解析する．

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などのマイクロアーキテクチャのバリエーションを絞り込む．

Step3 このマイクロアーキテクチャの有効性を判断するために，命令レベルのシミュレータ上で，ターゲットプログラムを実行し，性能評価を行なう．

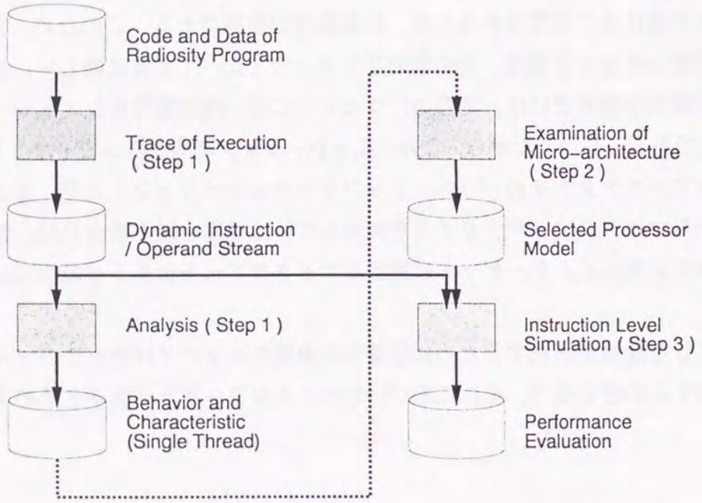


図 3.1: マイクロアーキテクチャ決定手法の流れ

多重スレッド方式の命令レベルのシミュレーションは，動的な命令トレースを用いる．この命令レベルシミュレーションを，次節で説明する．

3.2.2 シミュレーション環境

多重スレッドプロセッサの有効性を示すために，命令発行や実行時のパイプライン動作をソフトウェアでシミュレーションし性能評価を行った．図 3.2に性能評価シミュレーションの説明図を示す．まず最初に，対象のベンチマークプログラムを単一スレッドの環境 (同時に一つのスレッドしか実行できない環境) で実行し，アセンブラレベルで命令の動的な実行トレースをスレッドごとに作成する．この命令の動的トレースをシミュレータのスレッドごとの入力として性能評価をする．多重スレッド方式については，機能ユニットへの命令発行のスケジューリングや多重命令発行機能を実現し，パイプライン実行については，命令間のデータ依存や制御依存 (分岐)，ならびにキャッシュミスによるパイプラインの乱れについても考慮している．一方，オペランドについても，ベンチマークプログラムを単一スレッドの環境で実行し，オペランドアドレスの動的な実行トレースをスレッドごとに作成する．ロード／ストアユニットは，命令実行時にスレッドに対応するオペランドアドレスを順番に参照する．

なお，オペランドの場合，スレッドごとに管理しているローカルデータの領域 (スタック領域など) については，アドレスの再割り付けが必要である (図 3.3)．単一スレッドの環境で動的トレースを作成するため，実際にはスレッドごとに異なる領域に置かれるべきローカルデータについても，同一領域に割り付けられるからである．

プログラミングモデルは，整数，浮動小数点それぞれ 32 本のレジスタを持つ 32 ビット RISC の命令セットと仮定する．

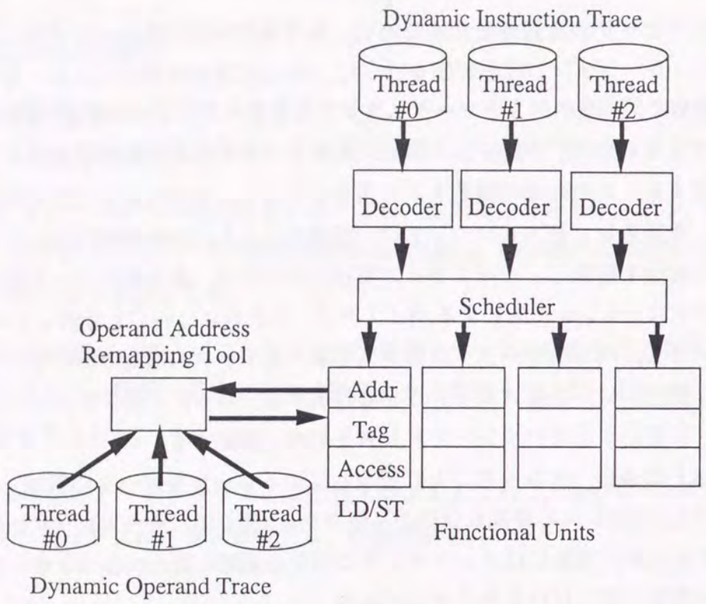


図 3.2: 性能評価シミュレータ

図 3.3: オペランドアドレスの再配置

3.3 アプリケーションの解析

本節では，動的トレースに基づいたアプリケーションの特徴や振る舞いについて述べる．第2.3.1 節で用いた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 3.1 に示す[27]．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，相対的に出現頻度が低い．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考えられる．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，演算遅延が極めて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．

表 3.1: 命令の出現頻度

category of instructions	ratio(%)
load/store	33.15
integer	33.89
floating point (add,sub,cmp,abs,neg,cnv)	8.83
floating point (mul,div)	6.67
branch	12.62
others	4.85

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推移を調べる．プログラムの実行時間に伴う命令群ごとの出現頻度の推移を図 3.4 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全体を 100 として表し，縦軸は各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時に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，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 3.1 の数値とほぼ一致する．従って，この命令出現頻度が定常的に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ことになる．

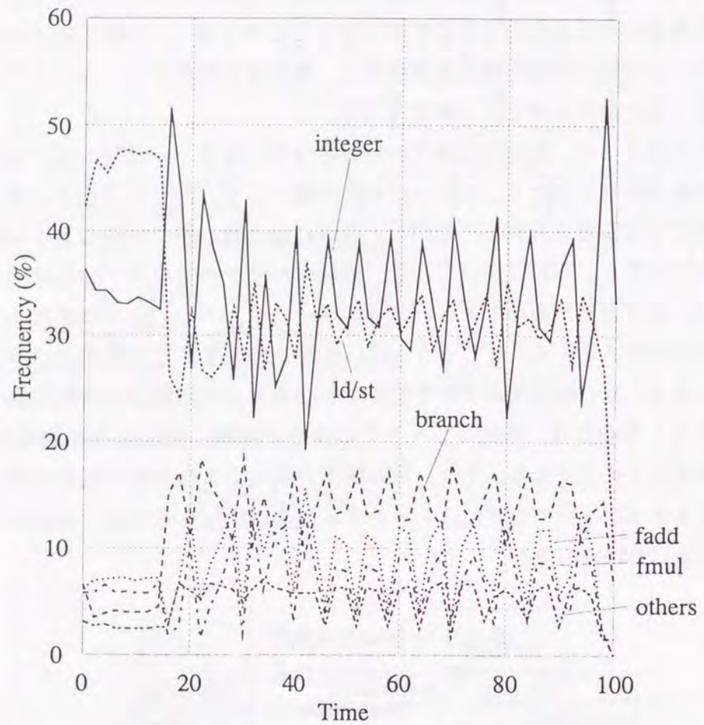


図 3.4: 命令出現頻度の動的推移

依存関係が存在する命令間の距離が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，および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相対的に低いものの，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 3.5 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出現頻度を示す．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．

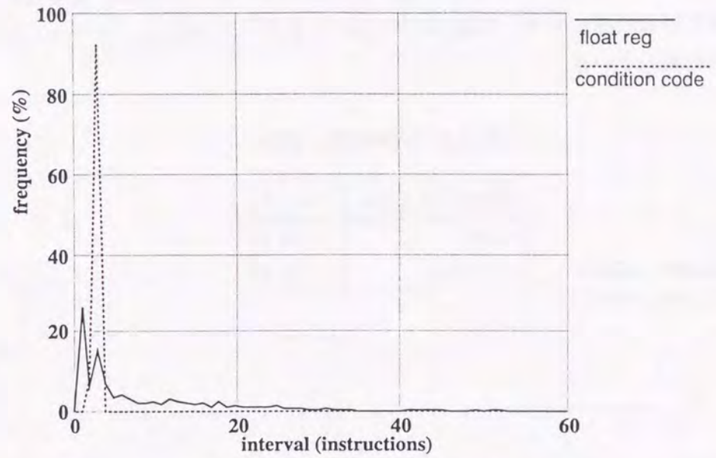


図 3.5: 依存を発生する命令間の距離

また，分岐実行は，命令実行が非連続となるため，パイプラインの乱れによるバブルの発生や，先行して読みだした命令の無効化により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与える．表 3.2 に，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，さらに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無効化され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，処理性能の向上を図れる．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 3.6 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．

分岐実行時の分岐先までの距離は，命令セットにおけるアドレス指定方法や命令キャッシュの構成に影響を与える．条件付き分岐命令（conditional branch），無条件分岐命令（unconditional branch）のそれぞれの場合について，分岐先までの距離を図 3.7に示す．横軸は，分岐命令から

分岐先までの距離を表し，縦軸は，条件付き分岐命令，無条件分岐命令それぞれの出現頻度を表している．この結果，条件付き分岐命令は，アドレスの増加する方向に比較的短い距離の分岐が圧倒的に多いが，これは，if-then 型の選択実行に用いられていることが影響している．一方，無条件分岐命令は，後方へ 4 命令程度先への分岐が多いが，これは，短いループが多いことを示していると思われる．また，無条件分岐命令の前方への分岐は，サブルーチンへの分岐およびリターンを表している．

表 3.2: 分岐発生の頻度

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

図 3.6: 分岐命令の出現した間隔

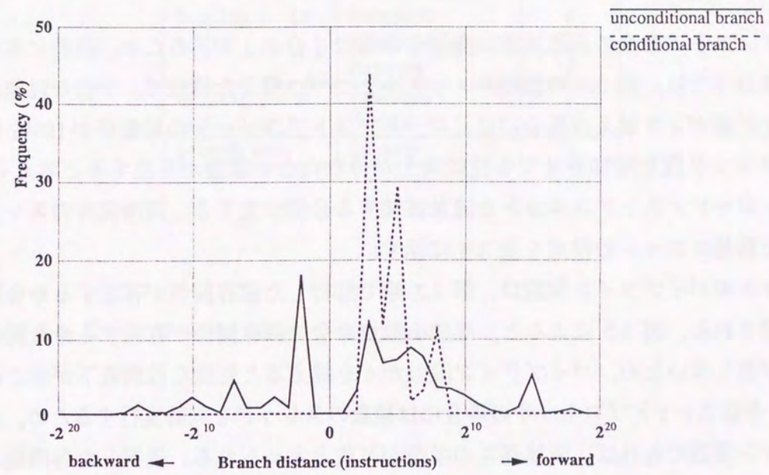


図 3.7: 分岐実行時の分岐先までの距離

3.4 評価対象のモデル

第3.3節の解析結果より，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する．浮動小数点演算命令は，17.41%と他の命令群と比べて相対的に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，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3.3に示す．

また，ロード／ストア命令の出現比率は全命令のほぼ3分の1であるため，同時に実行するスレッド数が3以下では，表3.3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向上が得られない事態が発生すると考えられる．その場合には，ロード／ストアユニットを追加実装する必要が生じる．同時実行のスレッド数を考慮に入れた機能ユニットの構成を表3.4に示す．

各機能ユニットのパイプライン段数は，第3.3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3.5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命令が最も多いため，パイプライン段数が4を越えると急激な性能低下が起こるはずである．一方，多重スレッドプロセッサの場合には複数のスレッドを同時実行するため，6段以内のパイプライン段数であれば，演算遅延の隠蔽が可能と考えられる．提案した各機能ユニットのパイプライン段数について表3.3に示す．

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命令の投機実行などの性能劣化抑止手段が考えられている[32]．しかしながら，ラジオシティ法のプログラムでは，分岐方向が実行時のデータにより決定され分岐予測が極めて難しいことから，投機的な分岐命令の実行や，実行が不確定な命令の先読みによる分岐高速化の効果は極めて少ないと思われる．むしろ，多重スレッドプロセッサの場合には，分岐による遅延(アイドル・スロット)が発生するとしても，異なるスレッドの実行による遅延の隠蔽を利用して，予測が失敗して命令を無効化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．

表3.3: ユニットでの実行命令と実行レイテンシ

function unit	category of instructions	latency (cycles)
load/store unit	load	3
	store	3
integer unit	add/subtract	1
	logical	1
	compare	1
floating point adder	add/subtract	3
	compare	3
	absolute/negate	3
	convert	3
floating point multiplier	multiply	3
	divide	28
branch unit	branch	2

表3.4: 評価モデルの実装ユニット数

model	A	B	C	D
load/store unit	1	1	2	2
integer unit	1	2	1	2
floating point adder	1	1	1	1
floating point multiplier	1	1	1	1
branch unit	1/each*	1/each*	1/each*	1/each*

(*: Each thread has a branch unit.)

3.5 評価結果

まず，第 3.4 節で決定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 3.8 (model A) に示す[27]．横軸はスレッド数を表し，縦軸には，スレッド数が 1 の構成に対する性能向上比を示す．この結果，基本構成の多重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を図れることが判明した．ただし，スレッド数が 3 以下では，ほぼ線形に性能が向上するが，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，異なるスレッドの各命令を機能ユニットに発行する場合に，機能ユニット間の競合発生の頻度が増加するためと考えられる．続いて，表 3.4 を考慮してロード／ストアユニットと整数演算ユニットを追加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 3.8 (model B〜D) に示す．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／ストアユニットと整数演算ユニットを増加させたモデルの性能向上比が大きくなる．これは，ロード／ストアユニットと整数演算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効果と考えられる．

命令実行シミュレーションの結果より，提案した構成において，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 1.9 倍，3 スレッドで 2.5 倍，4 スレッド以上ではロード／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 3.7 倍，5 スレッドで 4.3 倍の性能向上が得られることがわかった．

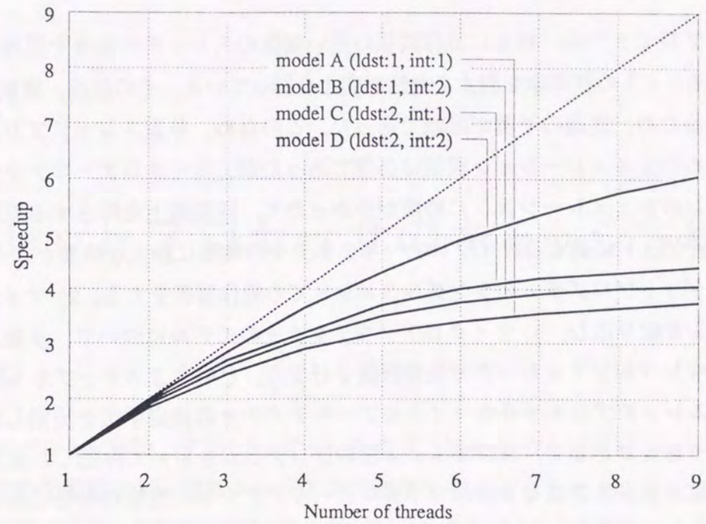


図 3.8: 同時実行スレッド数と性能向上比

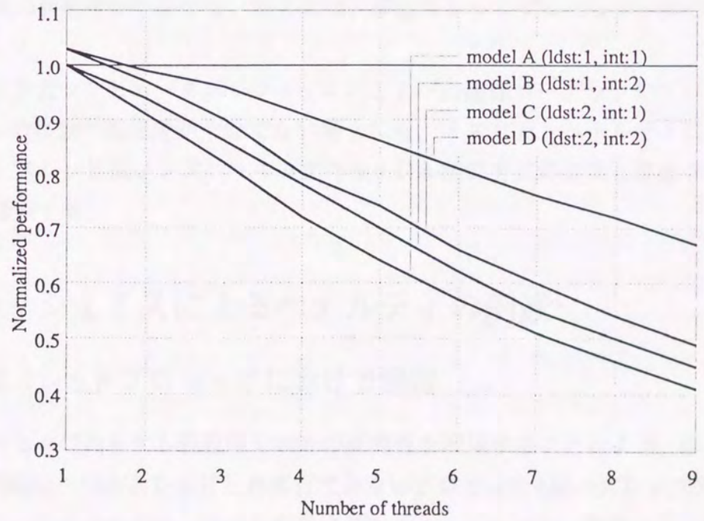


図 3.9: 理想状態に対する性能比

3.6 結言

多重スレッドプロセッサは，相互に依存関係の無い複数のスレッドの命令を同時実行することにより，機能ユニットの稼働率を向上させ性能向上を図っている．その反面，複数のスレッドが独立に動作するため，性能の予測が困難であった．このため，多重スレッドプロセッサの性能評価をするためのシミュレーション環境は必須であったが，マイクロアーキテクチャの多様なバリエーションのシミュレーションに時間がかかったり，性能向上を抑止する原因の特定が難しいため，ターゲットに適するマイクロアーキテクチャの構築に膨大な時間がかかる．今回，我々は，1) ターゲットプログラムのシングルスレッドでの動作解析をする，2) マイクロアーキテクチャのモデルを絞り込む，3) マイクロアーキテクチャのモデルについて，多重スレッド方式を実現した命令レベルシミュレータで性能評価を行なう，という3ステップからなる手順にもとづく，多重スレッドプロセッサのマイクロアーキテクチャの決定手法を提案した．

ＣＧアプリケーションとして，ラジオシティ法のプログラムを用いて評価した結果，今回提案した手法が多重スレッドプロセッサのマイクロアーキテクチャの構築に有効であることを明らかにする，とともに粗粒度レベルの並列性を十分に活用した多重スレッドプロセッサは，高い処理性能を達成することを示した．

第4章 多重スレッドプロセッサのキャッシュ構成

4.1 緒言

多重スレッドプロセッサは，アルゴリズムに内在する粗粒度レベルの並列性を利用して，複数命令の同時実行を可能にし，機能ユニットの稼働率を高めることにより，システム性能を向上させる．この観点より，3次元CGアプリケーションなどにおいては，スーパースカラ方式に比べて効率良く性能向上が図れる．一方，同時に実行するスレッド数の増加に伴い，オペランドデータとしてアクセスするメモリ空間，いわゆるワーキングセットが拡大するため，共有しているデータキャッシュのヒット率が低下する．この結果，キャッシュの容量やセット・アソシアティブ方式の連想度が十分でない場合には，多重スレッドプロセッサが持つ能力を発揮できなかった[33]．

本章では，3次元コンピュータグラフィックスおよび動画像コーデックのプログラムを用いて，キャッシュの容量や連想度が十分でない場合にも，ミスヒットペナルティによる性能劣化を抑止できる，スレッド間ノンブロッキングキャッシュ制御方式の提案と評価を行い，その有用性について考察する．

4.2 キャッシュミスによるペナルティの削減

4.2.1 多重スレッドプロセッサにおける課題

アプリケーションに内在する粗粒度レベルの並列性を利用することにより，多重スレッドプロセッサは，同時に一つのスレッドしか実行できないプロセッサ(単一スレッドプロセッサ)に比べて，3次元CGなどでは高い性能を実現できる．しかしながら，多重スレッドプロセッサの場合，各スレッドは共通のデータ領域とは別に各々独立な作業領域を確保するために，同時に実行するスレッド数の増加に伴い，オペランドデータのワーキングセットが拡大してしまう．

このため，複数のスレッドが共有のデータキャッシュをアクセスする場合，ヒット率の低下を起こし，多重スレッドプロセッサの有効性を低く抑えてしまうという問題を有している．

一方，命令キャッシュについては，ある程度以上の容量を備え，かつ，複数のスレッドが同一のプログラムを実行する場合，実行スレッド数が増加してもキャッシュのヒット率はほとんど低下しないことが知られている．これは，頻繁に参照される関数やループ部分については，あるスレッドが初めて参照する以前に，他のスレッドによって既にキャッシングされていることなどが理由と考えられる．

4.2.2 従来手法によるキャッシュ制御方式

キャッシュのヒット率を向上させるためによく用いられる方法として，キャッシュの容量やセット・アソシアティブ方式の連想度 (ウェイ数) を増やすことが挙げられる[34,35]．一方，キャッシュのミスヒット時の制御方式として簡単で一般的な方式が，ブロッキング (キャッシュ) 制御方式である．この方式は，キャッシュにミスヒットした場合には，後続のキャッシュアクセスを停止させ，ミスヒットしたアクセスに関する処理が完了した後に，後続のアクセスを受け付ける．このため，キャッシュの容量やセット・アソシアティブ方式の連想度を増やしたとしても，ブロッキング制御方式では，キャッシュミス時に発生するペナルティにより処理性能の向上が十分に得られない場合がある．

ブロッキング制御方式のキャッシュを採用した多重スレッドプロセッサの場合には，ある一つのスレッドによって引き起こされたキャッシュミスによって，全スレッドのキャッシュアクセスが停止させられる．図 4.1に，ブロッキングキャッシュ制御方式におけるキャッシュアクセス時の命令実行の動作例を示す．スレッド #0 によるキャッシュアクセス #0-0 がミスヒットした場合 (Cycle C+1)，すべてのスレッドの後続するキャッシュアクセス #0-1，#1-0，#2-0 は停止し，それ以降のアクセスは発行禁止となる．

したがって，粗粒度並列性を応用しようとしている多重スレッドプロセッサでは，単一スレッドのプロセッサに比べてキャッシュミスによる性能への影響は深刻であり，キャッシュの容量や連想度を増やすだけの方法では不十分と予想される．

一方，キャッシュミスに伴うデータの更新中にも，後続するキャッシュへのアクセスを受け付け可能な制御方式をノンブロッキングキャッシュ制御[36] と呼ぶが，この方式は，ブロッキング制御方式に比べキャッシュミスペナルティによる性能劣化を抑止することができる．しかしながら，この方式を採用した場合には，ミスヒットした命令に後続する命令が先に完了し，同一スレッド内で命令完了の順序が入れ替わるため，命令実行結果の整合性を維持するリオーダバッファやリザベーション・ステーション[21] などの複雑な機構を必要としている．このため，ノンブロッキングキャッシュ制御方式は，一般的には，極めて高い性能を実現する一部の単一

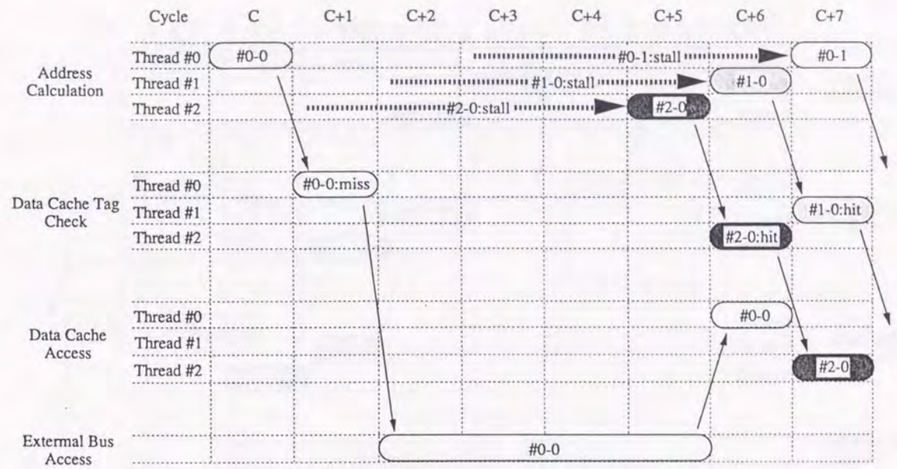


図 4.1: ブロッキングキャッシュ制御方式でのパイプライン動作

スレッドプロセッサに限って実装されている．

4.2.3 スレッド間ノンブロッキングキャッシュ制御方式

多重スレッドプロセッサにおけるキャッシュミス時のペナルティを低減するために，異なるスレッド間で相互にブロッキングしないキャッシュ制御方式 (スレッド間ノンブロッキング (キャッシュ) 制御方式) を提案する[37]．本方式は，4.2.2で述べたノンブロッキング制御方式を，同一スレッド内のキャッシュアクセスではなく，異なるスレッド間のキャッシュアクセスに限定して適用する．すなわち，あるスレッドのアクセスがキャッシュミスを起こし，キャッシュのデータを更新している期間にも，異なるスレッドに限っては後続するキャッシュアクセスを受け付け，同一のスレッドについては受け付けない．スレッド間には本来依存関係はないため，同一アドレスへのアクセスをしない限り，他のスレッドのミスヒットにかかわらずキャッシュアクセス可能である．後続するアクセスがミスヒットしたアドレスと一致する場合には，データが更新されるまで待たされる．スレッド間ノンブロッキング制御方式におけるキャッシュアクセス時の命令実行の動作例を図 4.2に示す．スレッド #0 によるキャッシュアクセス #0-0 がミスヒットした場合 (Cycle C+1)，それに後続するスレッド #0 からのキャッシュアクセス #0-1 は停止し，それ以降のアクセスは発行禁止となるが，スレッド #1 およびスレッド #2 からのキャッシュアクセス #1-0，#2-0 は受け付けられる．

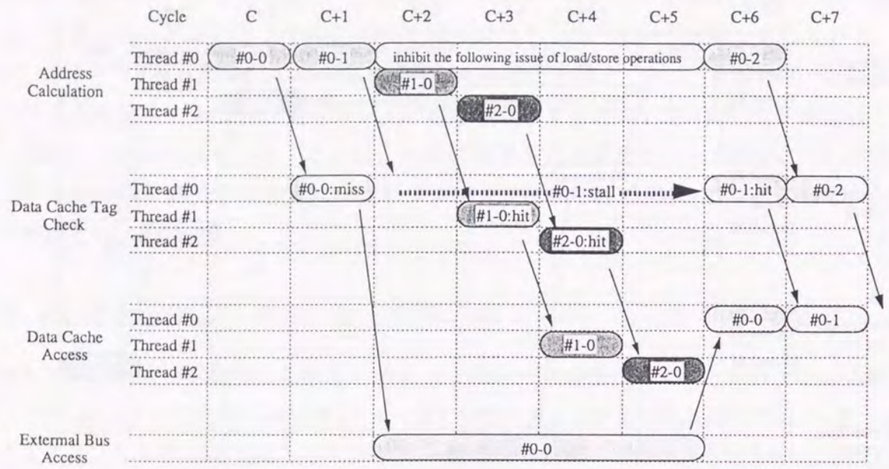


図 4.2: スレッド間に限定したノンブロッキングキャッシュ制御方式でのパイプライン動作

本方式では，同一スレッド内の命令の実行および完了順序は入れ替わらないため，整合性維持のための特別なハードウェア機構は不要であり，キャッシュミスに後続するアクセスのアドレスが，ミスをしたアドレスと一致しないことをチェックするためのアドレス比較器の追加だけですむ．これより，多重スレッドプロセッサに本方式を適用した場合には，ハードウェアコストの著しい増加と機構の複雑化をともなわずにキャッシュミスペナルティの低減を期待できる．ブロッキング制御方式，スレッド間に適用を限定したノンブロッキング制御方式，および，スレッド間およびスレッド内の両方に適用したノンブロッキング制御方式について，実現に必要なハードウェア機構を表 4.1にまとめる．

4.3 シミュレーション

4.3.1 評価環境およびシミュレーションモデル

スレッド間ノンブロッキングキャッシュ制御方式の有効性を示すために，命令発行や実行時のパイプライン動作およびデータキャッシュの振る舞いをソフトウェアでシミュレーションし性能評価を行った．評価環境は，第 3.2.2 小節と同じであるが，データキャッシュについては，アドレスを管理しているタグ部をソフトウェアで実現し，ミスヒット時には，指定した外部アクセスサイクル数の間パイプライン実行を停止するようにした．

3.3).

表 4.1: キャッシュ制御方式によるハードウェア機構の違い

	ブロッキング	ノンブロッキング	
		スレッド間	スレッド間およびスレッド内
単一命令流での実行モデル(必要なハードウェア機構)	in-order	in-order	out-of-order (リオーダバッファ，リザベーション・ステーションなど)
ロード／ストアユニットへの追加ハードウェア機構		アドレス比較器	

表 4.2: 機能ユニットの役割と構成

機能ユニット	命令の種類	レイテンシ(サイクル)	実装ユニット数
ロード／ストアユニット	load/store miss penalty	3 10	1
整数演算ユニット	arithmetic/ logical	1 1	1/thread
浮動小数点加算ユニット	add/sub/ cmp/convert	3 3	1
浮動小数点乗算ユニット	multiply divide	3 16	1
分岐ユニット	branch	2	1/thread

データキャッシュのラインサイズは 16 byte に設定し，マッピング方式にセット・アソシアティブを採用した場合の置き換えアルゴリズムには LRU(Least-Recently Used) 方式[35]を仮定した．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外部メモリとの間で転送するのに必要なサイクル数を付加した (本評価では 10 サイクルと仮定 (表 4.2))．なお，書き込みについては，書き込み後のデータの参照は比較的少ないことからライトスルー方式 (キャッシュに書き込むと同時に外部メモリにも書き込むこと) とし，ライトのミスヒット時にはノンアロケート方式 (キャッシュにはデータをロードせずに外部メモリへの書き込みだけを行うこと) とした．また，条件を揃えるために命令キャッシュは 100%ヒットと仮定した．

4.3.2 ベンチマークプログラム

ベンチマークプログラムには3次元CGのアプリケーション(C言語で記述)と動画像の符号化と復号化のアプリケーション(H.263)[38]を用いた．前者のプログラムには，ポリゴンに対して行う，座標変換，輝度計算，クリッピング，ポリゴンの細分割，テキスチャ・マッピング用のDDAパラメータ計算，と遠近法を考慮したグーロ・シェーディングが含まれている[1]．後者については，文献[38]に記載されているC言語のプログラムのうち，(逆)離散コサイン変換と(逆)量子化部分を用いた．スレッドの生成(並列化)については，前者はポリゴン単位で，後者はブロック(圧縮伸長で扱う画像の最小グループで，通常縦8画素×横8画素)単位で行った．また，出現する命令の頻度を表4.3に示す．

表4.3: 命令の出現頻度

命令の種類	出現頻度(%)	
	3次元CG	画像codec
ロード／ストア	36.40	32.21
整数演算	33.10	56.13
浮動小数点加減算	11.18	2.44
浮動小数点乗除算	11.82	2.00
分岐	7.50	7.22

この二つのベンチマークプログラムについて，データキャッシュのヒット率が理想状態の100%と仮定した時の性能向上比を図4.3に示す．同時実行スレッド数は1〜4とし，縦軸の性能向上比は，単一スレッドプロセッサ(同時実行スレッド数が1)の性能を1としている．

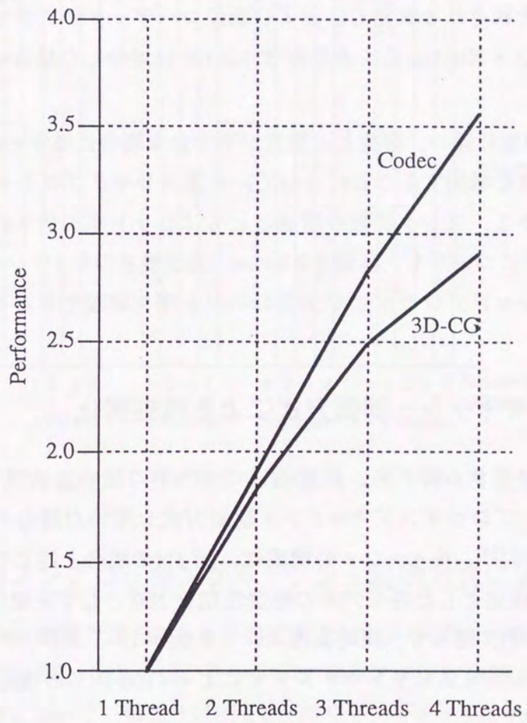


図4.3: キャッシュ100%ヒット時の性能比

4.4 評価結果

4.4.1 キャッシュ容量と連想度の増加によるヒット率の向上

まず最初に，キャッシュ容量とセット・アソシアティブ方式の連想度を増加することの有効性を評価するために，データキャッシュの構成を変えながらヒット率および性能向上についてシミュレーションを行った．

図4.4および図4.5に，それぞれ3次元CGおよび画像コーデックのアプリケーションを用いて，キャッシュ容量を1,2,4,8 Kbytesに，連想度を1,2,4,8に増やした場合のキャッシュヒット率を示す[37]．

同時実行スレッド数の増加に伴い，容量と連想度が不十分な場合にはキャッシュのヒット率は低下するが，容量や連想度を増加することによって，多重スレッドプロセッサでも高いヒット率を達成できることがわかる．スレッド数の増加とともにヒット率の低下の大きい3次元CGのアプリケーションの場合についても，容量8 Kbytes，連想度8のキャッシュ構成時には，4スレッド同時実行の多重スレッドプロセッサで97%のヒット率を実現できる(図4.4)．

4.4.2 ブロッキングキャッシュ制御方式による性能劣化

二つのベンチマークプログラムのうち，相対的にヒット率の低い3次元CGのアプリケーション(図4.4)に対して，ブロッキングキャッシュ制御方式を用いた時のキャッシュの構成に対する性能比を図4.6に示す[37]．キャッシュの構成は，図4.4の場合と同じで，縦軸は，キャッシュのヒット率が100%と仮定とした各モデルの理想性能を100として正規化している(各モデルの理想性能は図4.3を参照)．従って，同時実行スレッド数が1〜4スレッドであるモデルごとのP%の性能比は，キャッシュミスヒットペナルティによる(100-P)%の性能劣化をあらわしている．

容量8 Kbytes，連想度8のキャッシュ構成を用意すれば，同時実行のスレッド数が3または4の場合でも97%のヒット率を達成するが(図4.4)，得られる性能は17%も劣化してしまう．また，連想度を増加させることは，容量を増やすことに比べて著しく実装時のハードウェアコストと論理的な複雑さを増す．連想度8のキャッシュの実装は極めて大きな負荷であり，これらを考慮して連想度を4程度に抑えて実装した場合には，さらに性能は低下してしまう．

これより，ブロッキングキャッシュ制御方式を実装した多重スレッドプロセッサの場合，キャッシュの容量やセット・アソシアティブ方式の連想度を増やすことは，ヒット率の向上には有効であるが，性能向上には十分ではなく，低く抑えられてしまうことが判明した．

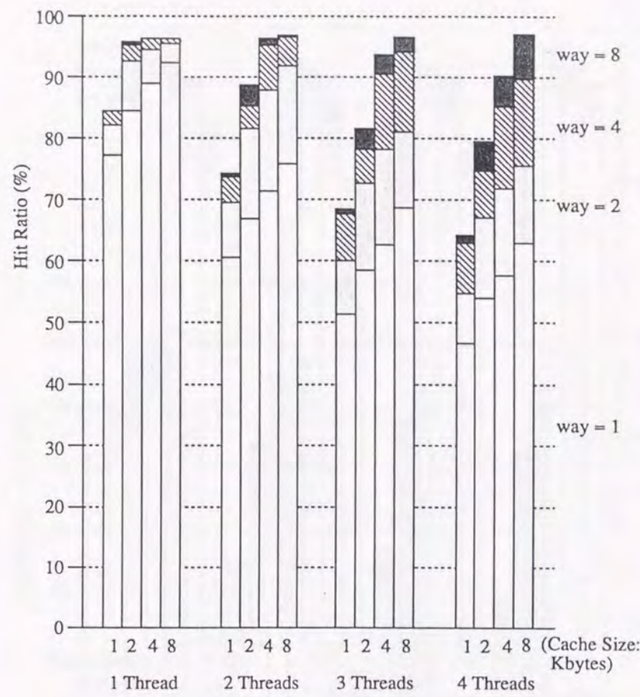


図4.4: 同時実行スレッド数とキャッシュヒット率(3次元CG)

4.4.3 スレッド間ノンブロッキングキャッシュ制御方式による性能向上

ブロッキングキャッシュ制御方式とスレッド間に限定したノンブロッキングキャッシュ制御方式の性能比を図4.7に示す[37]．ここでは，セット・アソシアティブ方式の連想度を4に固定して評価を行った．図4.7において，性能比は図4.6と同じく，キャッシュのヒット率が100%と仮定とした各モデルの理想性能を100として正規化している．

この結果より，スレッド間ノンブロッキングキャッシュ制御方式は，ブロッキングキャッシュ制御方式と比較して，ミスヒット時に発生するペナルティによる性能劣化を大幅に抑止できることがわかる．同時実行スレッド数が3または4で，4〜8Kbytesの容量を持つ場合には，性能劣化を約半分に抑え，理想性能の80〜90%を達成している．

また，スレッド間ノンブロッキングキャッシュ制御方式の性能は，2倍の容量または2倍の連想度を持つブロッキング制御方式の性能を上回ることがある．特に，同時実行スレッド数が2または3のモデルでは，ブロッキングキャッシュ制御方式に比べて，本方式はさらに効果が

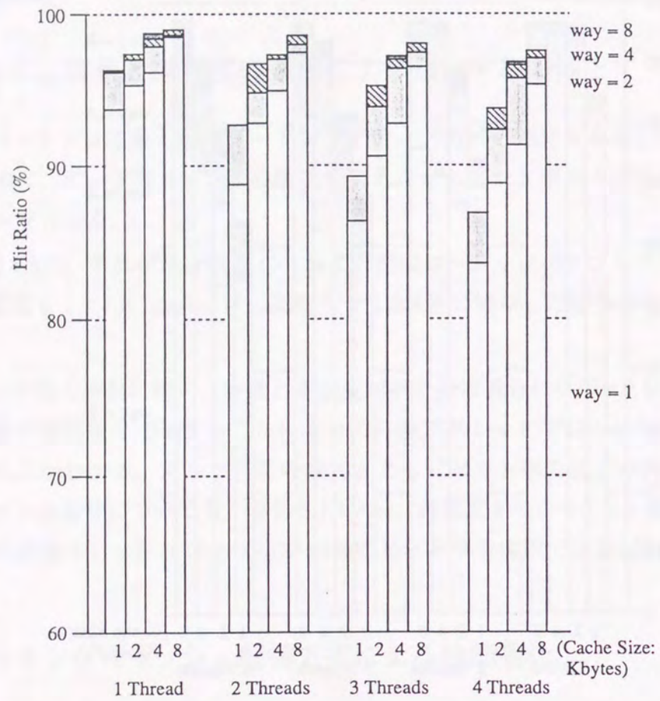


図 4.5: 同時実行スレッド数とキャッシュヒット率 (画像コーデック)

大きい．例えば，同時実行スレッド数が 4 の場合には，容量 4 Kbytes，連想度 4 のスレッド間ノンブロッキング制御方式の性能比 (図 4.7) は，容量 4 Kbytes，連想度 8，または，容量 8 Kbytes，連想度 4 のブロッキング制御方式の性能比 (図 4.6) を越えている．また，スレッド数が 2，3 の場合には，容量 4 Kbytes，連想度 4 のスレッド間ノンブロッキング制御方式の性能比 (図 4.7) は，容量 8 Kbytes，連想度 8 のブロッキング制御方式の性能比 (図 4.6) を上回っている．これは，ブロッキング制御方式のキャッシュで容量や連想度を増やすよりも，スレッド間ノンブロッキング制御方式を実装するほうが，小さいハードウェアコスト (チップ面積) でより高い性能を得られることを意味している．

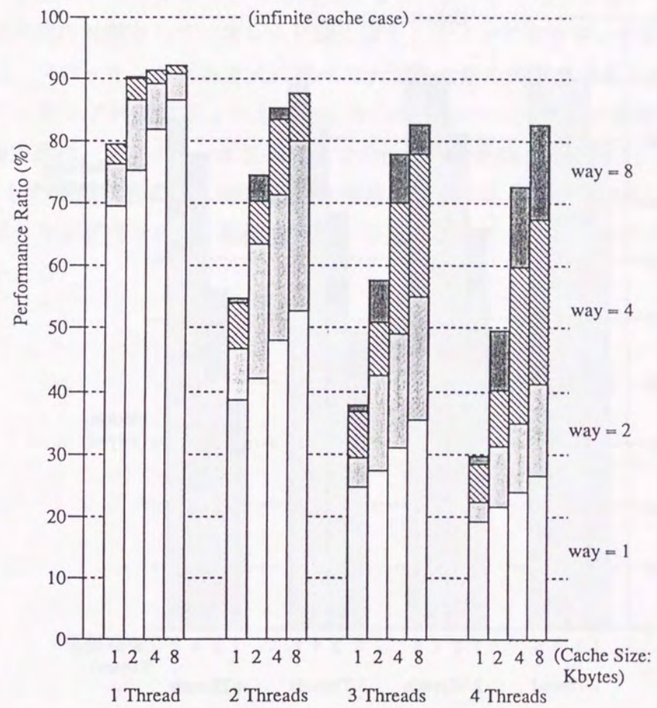


図 4.6: ブロッキングキャッシュ制御方式におけるキャッシュの構成による性能比

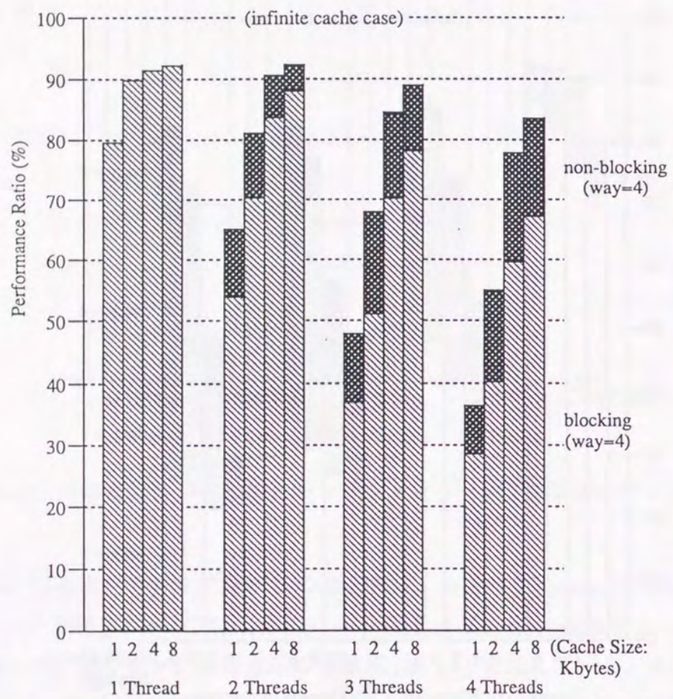


図 4.7: ブロッキングキャッシュ制御方式に対するスレッド間ノンブロッキングキャッシュ制御方式の性能向上

4.5 結言

多重スレッドプロセッサは，単一スレッドプロセッサに比べて，3 次元 CG などでは高い性能を実現できるが．オペランドデータのワーキングセットの拡大により，データキャッシュのヒット率低下を引き起こすという問題を有していた．さらに，一つのスレッドで発生したミスヒットが全スレッドのキャッシュアクセスを停止させるため，キャッシュミスによるペナルティが，多重スレッドプロセッサの有効性を低く抑えてしまうという問題を有していた．

本章では，多重スレッドプロセッサで問題となる，データキャッシュアクセス時のミスヒットによる性能劣化の対策として，スレッド間に限定したノンブロッキング制御方式を提案した．本制御方式は，ブロッキング制御方式に比べて大幅に性能を向上させることができ，さらに従来のノンブロッキング制御方式よりも小さくかつ簡単なハードウェア機構で実現可能である．提案した制御方式は，ブロッキング制御方式での性能劣化を約半分に抑え，キャッシュのヒット率が 100%とした理想状態の 80–90% と高い性能を達成することができた．これより，スレッド間ノンブロッキングキャッシュ制御方式は，多重スレッドプロセッサの性能向上に極めて有効であるといえる．

第 5 章

結論

本論文では，画像生成の高速化を実現する多重スレッド方式のプロセッサについて，ハードウェアコスト，アプリケーションプログラムに対する性能向上，マイクロアーキテクチャと内部構成，および，多重スレッド方式のプロセッサに適したキャッシュ構成について考察した．本研究で得られた主要な成果を以下に要約する．

1. 多重スレッド方式のプロセッサについて，LSI 実装設計と，3 次元画像生成プログラムを用いた性能評価を行った．実装した 3 スレッド同時実行のプロセッサの場合，性能向上比は 2.5〜3 倍を達成していること，機能ユニットの追加によりさらに大幅な性能向上が得られること，面積当たりの性能比は 1.5 倍以上に達していることより，本方式は粗粒度レベルの並列性を十分に活用し，ハードウェアの能力を 100%近くまで発揮できる，効率的なアーキテクチャであることを示した．

2. 性能の予測が難しかった多重スレッドプロセッサについて，アプリケーションプログラムの解析と性能評価シミュレーションからなる，マイクロアーキテクチャの構築手法の提案ならびにその環境を構築した．提案した手法と環境を用いて，3 次元画像生成アプリケーションの一つであるラジオシティ法に適した，多重スレッドプロセッサのマイクロアーキテクチャを構築し，プログラムを用いて評価した結果，本プロセッサは高い処理性能を達成した．

3. キャッシュの容量や連想度が十分でない場合にも，ミスヒットペナルティによる性能劣化を抑止できる，スレッド間ノンブロッキングキャッシュ制御方式の提案と性能評価を行った．その結果，ブロッキング制御方式での性能劣化を約半分に抑え，キャッシュのヒット率が 100%とした理想状態の 80–90% の高い性能を達成することができ，従来のノンブロッキング制御方式よりも小さくかつ簡単なハードウェア機構で実現可能であることを示した．

今後に残された課題としては，以下のものが挙げられる。

1. アーキテクチャおよびハードウェアに関しては，より広い応用や柔軟性確保のために，スレッド単位で異なるプログラムを実行する場合に，スレッドごとの負荷に応じて性能の割り振りを変更できる仕組みや，スレッド間通信のためのハードウェア機構の検討が必要である．

2. ソフトウェアに関しては，プロセッサのスレッド・スロットへの負荷バランスを考慮したスケジューラや，複数スレッド同時実行状態を考慮して各スレッドごとの命令実行順を最適化するコンパイラを検討しなければならない．

謝辞

本研究の全過程を通じて，終始御懇切な御指導と御鞭撻を賜わりました，大阪大学大学院工学研究科情報システム工学専攻白川功教授に謹んで深謝の意を捧げます．

大阪大学大学院工学研究科情報システム工学専攻石浦菜岐佐助教授には適切な御指導，ご助言をいただき，心から感謝申し上げます．

本論文を執筆するにあたり有益な御教示，御助言を賜わりました，大阪大学大学院工学研究科情報システム工学専攻藤岡弘教授，村上孝三教授に厚く感謝申し上げます．

大阪大学大学院博士後期課程に在学中に多くの御指導，御教示を賜わりました大阪大学大学院工学研究科情報システム工学専攻西尾章治郎教授，薦田憲久教授，赤澤堅造教授，下條真司教授，寺田浩詔名誉教授 (現在，高知工科大学) ならびに鈴木胖名誉教授 (現在，姫路工業大学) に厚く御礼申し上げます．

本研究を進めるにあたり，直接の御指導，御助言，御討論頂き，また様々な面でお世話になりました京都大学大学院情報学研究科通信情報システム専攻尾上孝雄助教授に感謝申し上げます．

大阪大学において社会人学生として本研究を進める機会を与えていただいた松下電器産業株式会社三木弼一常務取締役，櫛木好明取締役，同社九州マルチメディアシステム研究所坂尾隆所長，同社 DTV ネットワークソリューションセンター中島不二雄所長，同社マルチメディア開発センター津賀一宏所次長，同社中尾研究所鷺島敬之主幹技師に深く感謝の意を表します．

松下電器産業株式会社マルチメディア開発センター西澤貞次主担当，浅原重夫主担当には本研究の遂行にあたって御指導と御鞭撻をいただきました．心より感謝申し上げます．

本研究の全過程を進めるにあたり，直接の御指導，御助言頂き，また様々な面でお世話になりました松下電器産業株式会社マルチメディア開発センター清原督三リーダに衷心より感謝申し上げます．

本研究を行うにあたり，松下電器産業株式会社在職中に，共同研究者として常に議論を行ない，御討論をいただきました京都工芸繊維大学工芸学部電子情報工学科平田博章助教授に深甚なる謝意を表します．

第 2 章について，アーキテクチャの検討や LSI 化に御協力いただいた，松下電器産業株式会社マルチメディア開発センター吉岡康介主任技師に深く感謝いたします．また，同社在職中に

LSI 化に御協力いただいた，白岩基紹氏 (現在，松下通信工業株式会社)，中川健児氏，山下智郎氏に御礼申し上げます．

第 2 章，第 3 章における研究を行なうにあたり，性能評価用ベンチマークプログラムを提供していただいた松下電器産業株式会社マルチメディア開発センター中俊弥主席技師，西村健二主任技師，望月義幸主任技師，に感謝の意を表します．

第 3 章について，大阪大学在学中にともに研究を進め御討論御協力をいただきました正城敏博氏 (現在，大阪大学大学院工学研究科講師)，古賀拓也氏 (現在，日本電気株式会社)，藤井貴晴氏に深く感謝致します．

第 4 章において，ベンチマークプログラムとソフトウェア開発環境を提供していただいた松下電器産業株式会社マルチメディア開発センター平井誠主席技師，コアソフト開発センター藤井茂樹主任技師に深く感謝いたします．ハードウェア構成について議論し，シミュレータのプログラム化に御協力いただいた同社半導体開発本部甲斐康司主任技師に御礼申し上げます．また，シミュレーションに御協力いただきました奥畑宏之氏 (現在，株式会社シンセシス) に深く感謝致します．

本研究を進めるにあたり，暖かい御助言，御協力をいただき，また，様々な面でお世話になりました，大阪工業大学工学部電気工学科 重弘裕二講師，大阪大学先導的研究オープンセンター藤田玄助手，ならびに，社会人学生として後期課程に在学しておられた岡田圭介氏 (現在，三菱電機株式会社)，宇野裕史氏 (現在，シャープ株式会社)，長尾明氏 (現在，シャープ株式会社)，矢野政顕氏 (現在，高知工科大学教授)，近藤仁志氏 (現在，古野電気株式会社)，山口雅之氏 (現在，シャープ株式会社)，藤嶋秀幸氏 (現在，九州松下電器株式会社)，に深く感謝の意を表します．

また，本研究を行うにあたり，様々な面でお世話になりました，松下電器産業株式会社九州マルチメディアシステム研究所東幸哉主担当，同社マルチメディア開発センター日高教行主席技師，西村明夫主席技師，同社 DTV ネットワークソリューションセンター上原宏敏主担当，同社半導体開発本部本原章リーダ，豊永昌彦主席技師，落合利之主任技師，一宮敬弘主任技師に厚くお礼申し上げます．

研究室での楽しい時を一緒に過ごし，種々の面でお世話になった，秘書須賀エリサ嬢，長岡智嗣氏 (現在，任天堂株式会社)，石田真樹氏 (現在，ソニー株式会社)，井堀雅淳氏，松村謙次氏 (現在，株式会社ケー・シー・エス)，大下勝氏，吉田幸弘氏 (現在，株式会社シリコンメディア)，宮野鼻晃士氏 (現在，大阪大学大学院工学研究科後期博士課程在学)，山本哲三朗氏 (現在，三菱電機株式会社)，宗宝玉嬢 (現在，大阪大学大学院工学研究科後期博士課程在学) をはじめとする白川研究室の諸氏に厚く御礼申し上げます．

最後に，本研究の全過程を通じて，常に励まし支援してくれた妻章代と長男琢人に感謝する．

参考文献

[1] J. Foley, A. Dam, S. Feiner, and J. Hughes: *Computer Graphics: Principles and Practice (2nd Ed.)*, Addison-Wesley Publishing Company (1990).

[2] 鷺島敬之, 西澤貞次, 浅原重夫 共著: 並列図形処理, コロナ社 (1991 年).

[3] L.G. Roberts: "Machine Peerception of Three Dimensional Solids", *Optical and Electro-Optical Information Processing*, MIT Press, pp. 159–197 (1964).

[4] J. Warnock: "A hidden-surface algorithm for computer generated half-tone pictures", *University of Utah, Computer Sciende Department Technical Report*, TR4–15 (June 1969).

[5] H. Gouraud: "Continuous shading of curved surfaces", *IEEE Trans, Computers*, vol. C–20, no. 6, pp. 623–629 (June 1971).

[6] B.T. Phong: "Illumination for computer generated pictures", in *Comm. ACM*, vol. 18, no. 6, pp. 311–317 (June 1975).

[7] E. Catmull: "ComPuter display of curved surfaces", in *Proc. IEEE Conf. Computer Graphics, Pattern Recog., and Data Structures*, pp. 11–17 (May 1975).

[8] T. Whiffed: "An improved illumination model for shaded display", in *Comm. ACM*, vol. 23, no. 6, pp. 343–349 (June 1980).

[9] C.M. Goral, K.E. Torrance, D.P. Greenberg, and B. Battaile: "Modeling the interaction of light between diffuse surfaces", in *Proc. SIGGRAPH'84*, pp. 213–222 (July 1984).

[10] A.A.G Requicha: "Representations for rigid solids: Theory, methods, and systems", in *ACM Computing Surveys*, vol. 12, no. 4, pp. 437–464 (December 1980).

[11] H. Nishimura, H. Ohno, T. Kawata, I. Shirakawa, and K. Omura: "LINKS-1: A parallel pipelined multimicrocomputer system for image generation", in *Proc. IEEE 10th Annual Int'l Symp. Comput. Architecture*, pp. 387–394 (June 1983).

[12] 西村仁志, 出口弘, 辰己敏一, 河田亨, 白川功, 大村皓一: "コンピュータグラフィックスシステム LINKS-1 における並列処理性能の評価", 電子情報通信学会論文誌, vol. J68-D, no. 4, pp. 733–740 (1985 年).

[13] M. Abe, K. Nishimura, K. Takabatake, M. Hirai, and Y. Nakase: "Photorealistic image generation system for realtime animation", in *IPSJ SIG Notes*, CG 37–9 (February 1989).

[14] M. Hirai, K. Nishimura, M. Abe, and K. Takabatake: "Photo-realistic rendering system and its evaluation", in *IPSJ SIG Notes*, CG 41–5 (October 1989).

[15] S. Teller, C. Fowler, T. Funkhouser, and P. Hanrahan: "Partitioning and ordering large radiosity computations", in *Proc. SIGGRAPH'94*, pp. 443–450 (July 1994).

[16] T.A. Funkhouser: "Coarse-grained parallelism for hierarchical radiosity using group interactive methods", in *Proc. SIGGRAPH'96*, pp. 343–352 (August 1996).

[17] J. Torborg and J.T. Kajiya: "Talisman: Commodity realtime 3D graphics for the PC", in *Proc. SIGGRAPH'96*, pp. 353–363 (August 1996).

[18] M.F. Cohen, S.E. Chen, J.R. Wallace, and D.P. Greenberg: "A progressive refinement approach to fast radiosity image generation", in *Proc, SIGGRAPH'88*, pp. 75–84 (July 1988).

[19] J. L. Hennessy and D. A.Patterson: *Computer Architecture a Quantitative Approach*, Morgan Kaufmann Publishers (1990).

[20] N.-P. Jouppi and D. W. Wall: "Available Instruction-level Parallelism for Superscalar and Superpipelined Machines", in *Proc. of Third Int'l Conf. on Architectural Support for Programing Languages and Operating Systems*, pp. 272–282 (1989)

[21] M. Johnson: *Superscalar Microprocessor Design*, Prentice Hall Series in Innovative Technology (1991).

[22] H. Hirata, Y. Mochizuki, A. Nishimura, Y. Nakase, and T. Nishizawa: "A Multithreaded Processor Architecture with Simultaneous Instruction Issuing", in *Proc. of Int'l Symp. on Supercomputing, Fukuoka, Japan*, pp. 87–96 (1991).

[23] 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," in *Proc. of the 19th Annual Int'l Symp. on Computer Architecture*, pp. 136–145 (May 1992).

[24] 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: "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, ARC 94–2, pp. 9–16 (1992 年 6 月).

[25] H. Hirata, K. Kimura, S. Nagamine, A. Nishimura, Y. Nakase, and T. Nishizawa: "An Elementary Processor Architecture with Parallel Instruction Issuing from Multiple Threads ", in *Proc. of the Joint Symposium on Parallel Processing 1992*, pp. 257–264 (June 1992).

[26] 西尾一孝, 西村健二, 峰久次郎, 平井誠, 中瀬義盛: "画像生成におけるマルチプロセッサ管理の一手法", 電子情報通信学会研究会資料, CPSY90-63, pp. 151–155 (1985 年 7 月).

[27] K. Kimura, H. Hirata, T. Kiyohara, S. Asahara, T. Sagishima, T. Onoye, and I. Shirakawa: "Evaluation method of micro-architecture for multi-threaded processor", in *Proc. IEEE Int'l Symposium on Industrial Electronics* pp. 53–58 (May 1994).

[28] Cypress Semiconductor, *SPARC RISC User's Guide*, (February 1990).

[29] T. Onoye, T. Masaki, H. Hirata, K. Kimura, S. Asahara, T. Sagishima, I. Shirakawa, S. Tsukiyama, and S. Shinoda: "High-level synthesis of multithreaded processor based image generator," in *Proc. IEEE Int'l Symposium on Industrial Electronics*, pp. 47–52 (May 1994).

[30] T. Onoye, T. Masaki, I. Shirakawa, H. Hirata, K. Kimura, S. Asahara, and T. Sagishima: "High-level synthesis of a multithreaded processor for image generation", *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, Vol. E78-A, No. 3, pp. 322–330 (March 1995).

[31] K. Kimura, K. Yoshioka, T. Kiyohara, T. Sagishima, and I. Shirakawa: "Microarchitecture of multithreaded processor for computer graphics", in *Proc. IEEE Int'l Conference on Neural Networks and Signal Processing*, pp. 1586–1589 (December 1995).

[32] Lee, J.K.F. and Smith, A.J.: "Branch Prediction Strategies and Branch Target Buffer Design", in *IEEE Computer*, vol. 17, pp. 6–12 (January 1984).

[33] 平田博章, 木村浩三, 永峰聡, 西澤貞次, 鷺島敬之: “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”, 情報処理学会論文誌, Vol. 34, No. 4, pp. 595–605 (1993年4月).

[34] Stone, S.H.: *High-Performance Computer Architecture*, Addison-Wesley.

[35] 馬場敬信: コンピュータアーキテクチャ, オーム社.

[36] Kroft, D.: “Lookup-Free Instruction Fetch / Prefetch Cache Organization”, in *Proc. of the 8th Annual Int'l Symp. on Computer Architecture*, pp. 81–87 (May 1981).

[37] 木村浩三, 奥畑宏之, 尾上孝雄, 清原督三, 鷺島敬之, 白川功: “マルチスレッドプロセッサのデータキャッシュ制御方式”, 映像情報メディア学会誌, Vol. 52, No. 5, pp. 742–749 (1998年5月).

[38] ITU-T Rec. H.263: *Video coding for low bitrate communication*, Draft International Standard (May 1996).