第５章　ジェットパイプラインを用いた動画像生成用並列処理システム

§5.1　緒言

これまで述べてきた並列処理システムは、命令レベルの細粒度並列性を利用したものであるが、より大きな粒度の並列性を利用するシステムもある。また、細粒度並列性を利用したプロセッサを複数使用して、マルチプロセッサシステムを構成することによりさらなる速度向上を図ることも考えられる。

一般に大粒度のシステムは、独立したプロセッサユニットを複数個有し、それらの間をバスもしくはネットワークを介して接続したものが多い[Tomita 86][Hwang 85]。プロセッサユニット数が比較的少数の場合は主記憶をバスを介して共有した、共有メモリ型の密結合形態をとる場合が多いが、プロセッサ数が多くなるとバスがボトルネックになるため、各プロセッサにローカルメモリを分散配置しプロセッサ間は通信ネットワークによってメッセージ通信を行い並列処理する方式が用いられる。このような各種の並列処理システムについて、密結合の場合には共有メモリに対する各プロセッサのキャッシュのコヒーレンスの制御プロトコルに関する問題など[Suzuki 88,93]、粗結合形態の場合には結合ネットワークの形態および制御方法等に関する問題などに対する研究が幅広く行われている。

このようなマルチプロセッサシステムは汎用のものもあるが、それらのシステムにおいて有効な並列処理を行うためには通常の逐次処理プログラムと大きくプログラム手法を転換する必要があり、広く一般に使用されるためには解決すべき課題が多く残っている。そこで、対象となる目的を限定し、それらの問題を高速処理できるように特化した専用並列処理システムが数多く研究・開発されている[Tomita 86]。たとえば通常の計算機ではオーバヘッドが大きく処理に多くの時間を要する、論理型や関数型言語の処理システム等の研究が進められている[Shen 94][Inaba 95]。また、計算機による画像生成も並列処理が有効な分野である[Ishihata 87][Murakami 87a,87b][Yoshida 85][Nishimura 85][Kobayashi 87,88,94]。

本章では、第３章で提案した命令レベル並列処理を行うアーキテクチャであるジェットパイプライン・アーキテクチャが高性能を発揮できる応用例として、高速な画像生成能力が要求される光線追跡法による動画像生成のための並列処理システム

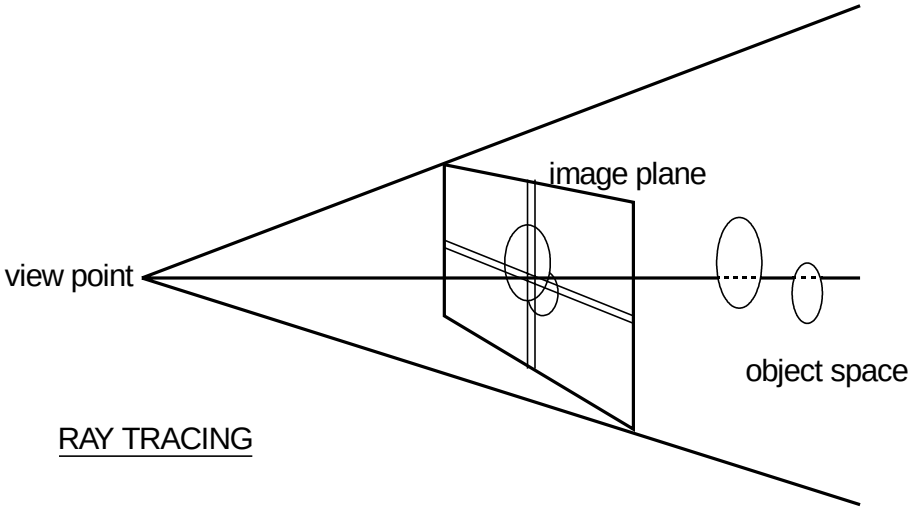
と、そのシミュレーションによる性能評価について述べる。まず、アルゴリズムの改良について部分更新法[Katahira 88;Katahira 90b]を提案する。そして、CPUレベルおよびシステムレベルの並列化について、ハードウエアシステムおよびソフトウエアの両面から検討しシミュレーションにより性能評価を行う[Katahira 88][Katahira 90a][Horiguchi 93]。アルゴリズムの改良、CPUレベルでの並列処理、およびシステムレベルでの並列処理の組み合わせによって、従来よりも非常に高速な動画像生成が行えることが期待できる。

§5.2　部分更新光線追跡法による動画像生成

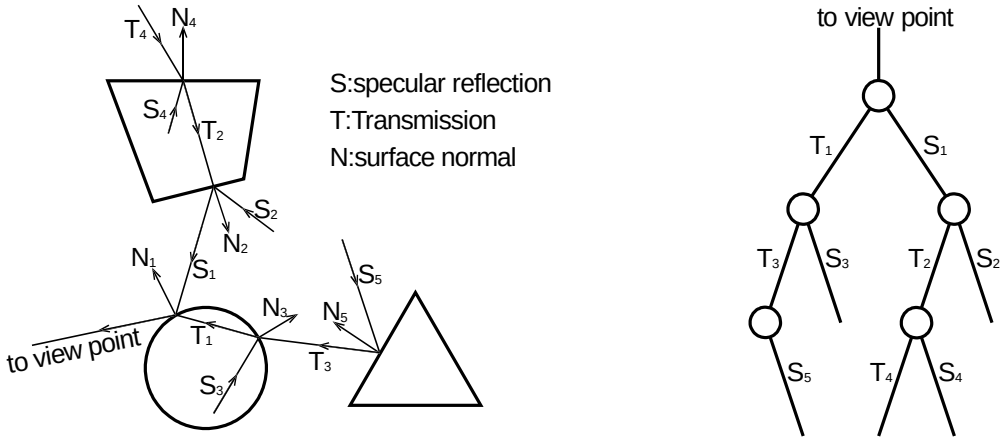
コンピュータ・グラフィックスの分野で現在用いられている画像生成アルゴリズムには様々なアルゴリズムがあるが、その中でも最も高品質な画像を生成できるものの一つに光線追跡法(ray tracing)がある[Whitted 80]。これは、図5-1に示すように、光源から発した光が、物体に当たって反射・屈折し、視点に至るまでの過程を視点から光源へと逆にたどって、各画素の輝度を計算する方法である。図5-1(b)に示すように、光線追跡の状況は木の形で表現でき、これを光線追跡木と呼ぶ。光線追跡法は、比較的単純なアルゴリズムにもかかわらず、反射、屈折などの現象を表現することができるため、非常に現実感のある画像を生成することが可能である。しかし、その一方で、膨大な計算時間を要するという欠点を持っている。

光線追跡法を用いて動画を生成する場合、各フレームごとにすべての画素を生成する方法では、空間分割法などの高速化手法を用いても膨大な時間を要する。そこで各フレーム間のコヒーレンスを考慮にいれた部分更新レイトレーシング法が提案され、高速な動画生成ができることが報告されている[Murakami 86]。部分更新レイトレーシング法は、動画の連続するフレーム間では画像の変化量が少ないことに着目して前フレームから変化した部分のみを再計算する手法である。この方法によって、物体配置の変更時には７～２１倍、質感パラメータの変更時には７０～１１０倍もの高速化が達成されることが報告されている[Ishihata 87][Murakami 87a,87b][Hirota 87]。

部分更新レイトレーシング法は、全ての画素の計算結果の保存、すなわち光線追跡情報の保存のために膨大な量のメモリを必要とする。そこでここでは動画の１シーンの間の各フレームで変化することが予想される画素の光線追跡木のみ保存する



(a) **光線追跡法**



(b) **光線追跡木** (binary shade tree)

図5-1　**光線追跡法による画像生成**

方法を提案する[Katahira 88,90b; Horiguchi 93]。 これは、移動するオブジェクトの軌跡から判定することができ、光線追跡情報の記憶に必要とされる記憶容量の低減が可能になる。レイトレーシングでは、光線と物体の交差判定を高速にするため物体の存在するオブジェクト空間を多数の小空間（セル、cell）に分割し、前処理においてそれぞれのセルが含む物体のリストが生成される[Fujimoto 85,86]。このとき、移動するオブジェクトの軌跡にあるセルを記録しておく。このオブジェクトの移動軌跡を含んでいるセルのことを、軌跡セル（locus cell）と定義する。

各画素の光線追跡木を保存するか否かの判定は、その画素から追跡した光線が軌跡セルを通過しているかどうかを調べることにより行う。その様子を図5-2に示す。光線が１本でも軌跡セルを通過していれば、その画素は移動物体の影響を受ける可能性があるので光線追跡木を保存する。追跡したどの光線も軌跡セルを通過しなかった場合は、画素は動画の1シーンの中で移動物体の影響を受けない。この場合は、光線追跡木を保存せず前に計算したフレームの計算結果を用いる。さらに、光線追跡木の各ノードにおいて下位レベルで再計算の必要性が無ければ、下位の全てのノードを光線追跡木から削除する。これらの方法により、従来の部分更新レイトレーシング法よりメモリ要求量を少なくし、現有のコンピュータで効率よく計算できるアルゴリズムに改良した。

２フレーム目以降において再計算を実行する際に、光線が移動物体の影響を受けたか否かを判定するためには、1フレーム目において保存された光線追跡木をルートから順にたどり、通過セルリスト(traverse list)に登録されているセルを一つづつしらべなければならない。部分更新法をさらに高速に実行するためには、この処理を高速化する必要がある。本手法では、各部分空間がどのフレームで移動物体を含むかを示す移動物体フラグを用いる新しい方法を用いることにより、移動物体の影響を受ける光線の判定を高速化している。

移動物体フラグを用いる手法では、まずオブジェクト空間を分割する際にどのフレームでそのセルを移動物体が通過するかを表す情報を記録しておく。これを移動物体フラグと呼ぶ。現在、移動物体フラグは4バイト=32ビットで表現しており、1シーンのフレーム数が32以下の場合は１フレームにつき1ビット、33以上64以下の時には2フレームを1ビットで表すようにしている。この移動物体フラグが0以外の場合、そのセルは軌跡セルとなる。

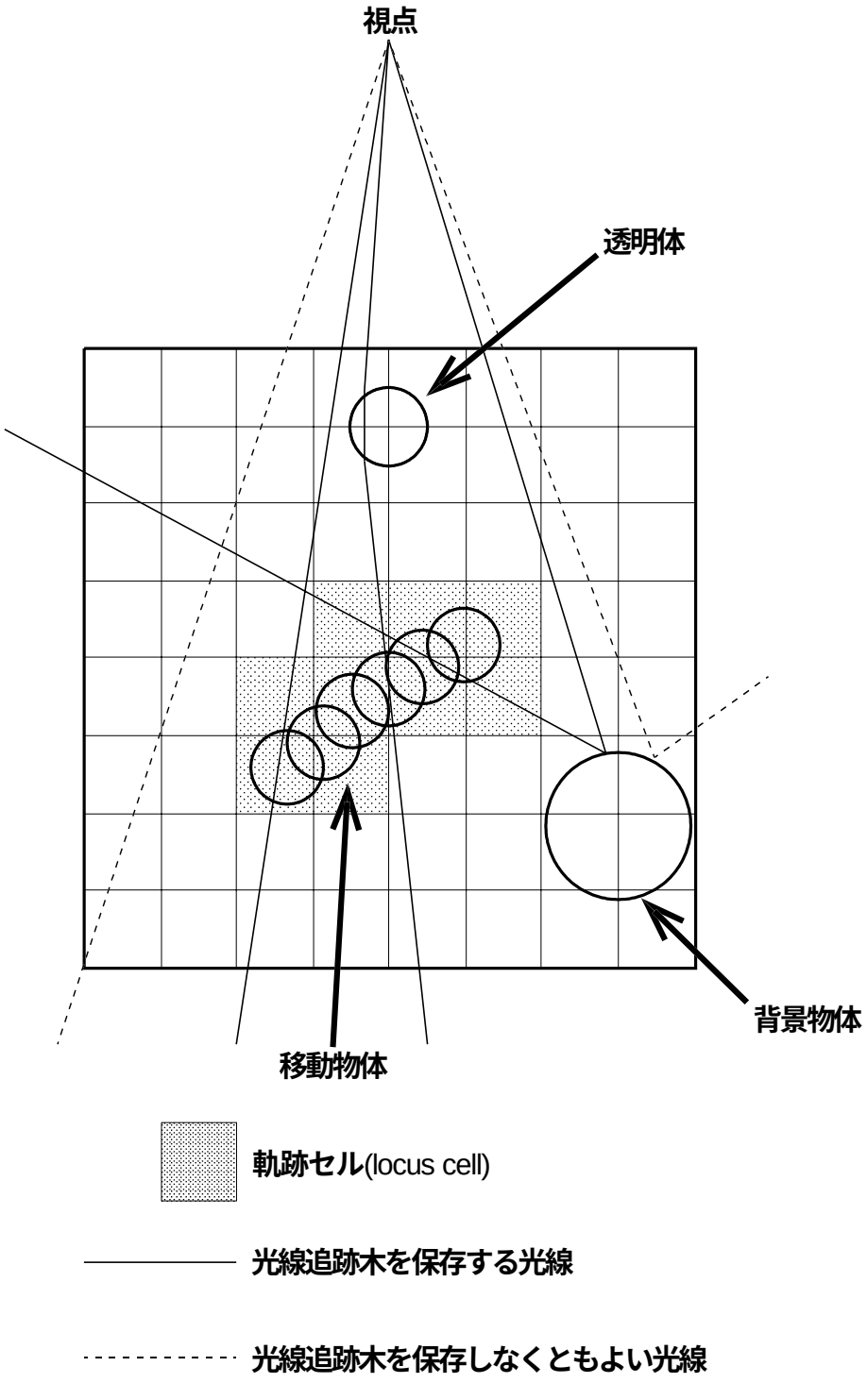


図5-2　部分更新光線追跡法

1フレーム目における処理はほぼ従来の手法と同様に行われるが、光線を追跡する際に光線が通過した各セルの移動物体フラグの論理和をとりながら進むようにして、追跡が終わったときにそれを光線の移動物体フラグとして保存する。もし光線が物体と交差した場合には、光線追跡木上に新たなノードが生成されることになるが、各ノードにおけるノードの移動物体フラグは図5-3に示すように、そのノードに関する光線および自分より下位のノードにおける移動物体すべてのビットごとの論理和をとることにより定義される。また、光線追跡木を保存するか否かの判定は、この各ノードや光線についての移動物体フラグが0であるか否かによって判定される。

次に、2フレーム目以降の再計算時における処理について述べる。再計算の必要性の判定は、まず、各ノードにおける移動物体フラグと、前フレームと現フレームを表すビットを立てた判定用フラグとの、ビットごとの論理和をとることにより行われる。その結果が0の場合、そのノードより下位のレベルでは移動物体の影響を受けないことがわかり、再計算は不要となる。0以外の結果となった場合についてのみ、保存してある光線追跡木をたどることによる従来のアルゴリズムを用いた判定を行えばよい。簡単な例を図5-4、図5-5および表5-1に示す。1−2フレーム間で移動物体が図5-4のように移動する場合、各セルにおける移動物体フラグは図5-5のようになる。ここで各光線に対する移動物体フラグは表5-1に示した値となる。後は上記のアルゴリズムに従って移動物体の影響の有無が判定され、表5-1最下段のような結果が得られる。

以上の処理において、従来に加えて新たに実行される演算は、ビットごとの論理演算が主であり、これは浮動小数点演算等に比べればはるかに簡単な処理であるので、実行速度に対する悪影響は低く抑えることが可能である。一方、各セルに関する移動物体フラグや、ノードや光線に関する移動物体フラグを保存するために従来の手法よりもややメモリ使用量が多くなるが、移動物体フラグは１つあたり1ワード(=4バイト)しか占めないため、メモリ使用量に対するオーバーヘッドも少なく抑えられる。

従来の部分更新レイトレーシングの場合は、全画素についての情報を保存するため、物体のどのような変化にも対応することが可能である。一方、本手法の場合、動画の１シーン中の移動物体の軌跡を用いて、そのシーンの間で変化することが予

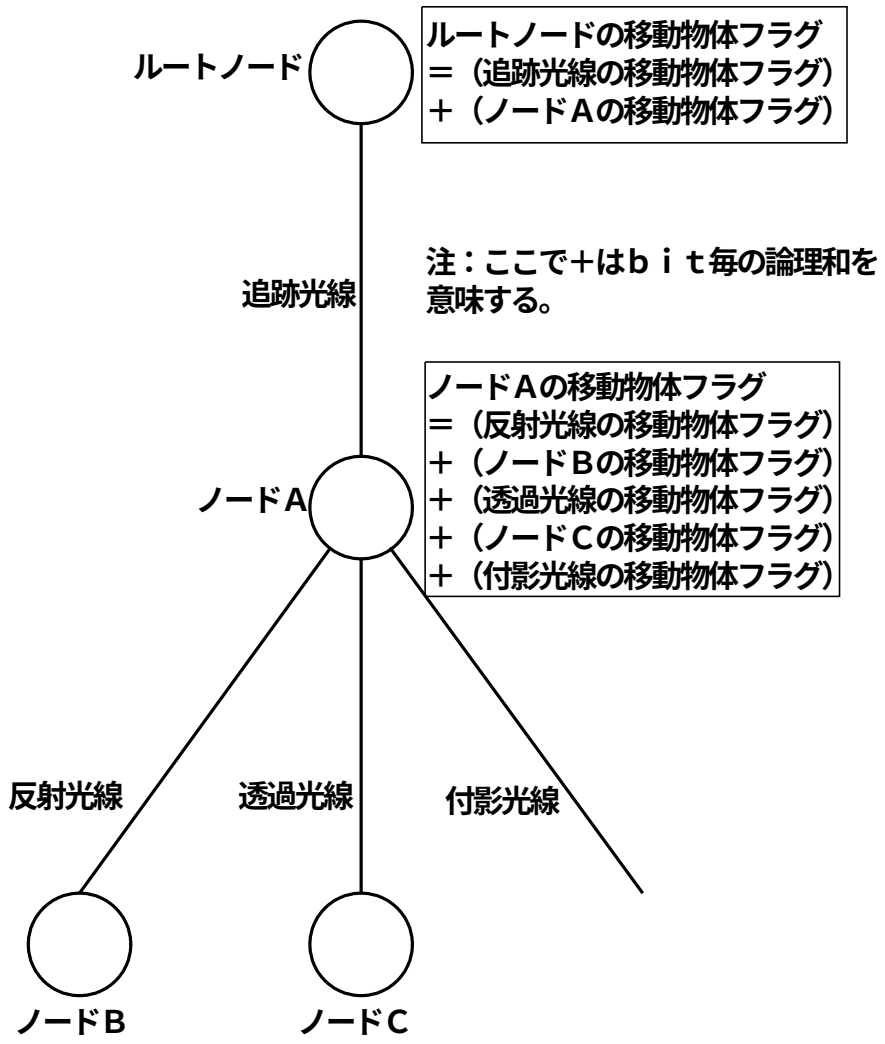


図5-3　**各ノードにおける移動物体フラグの定義**

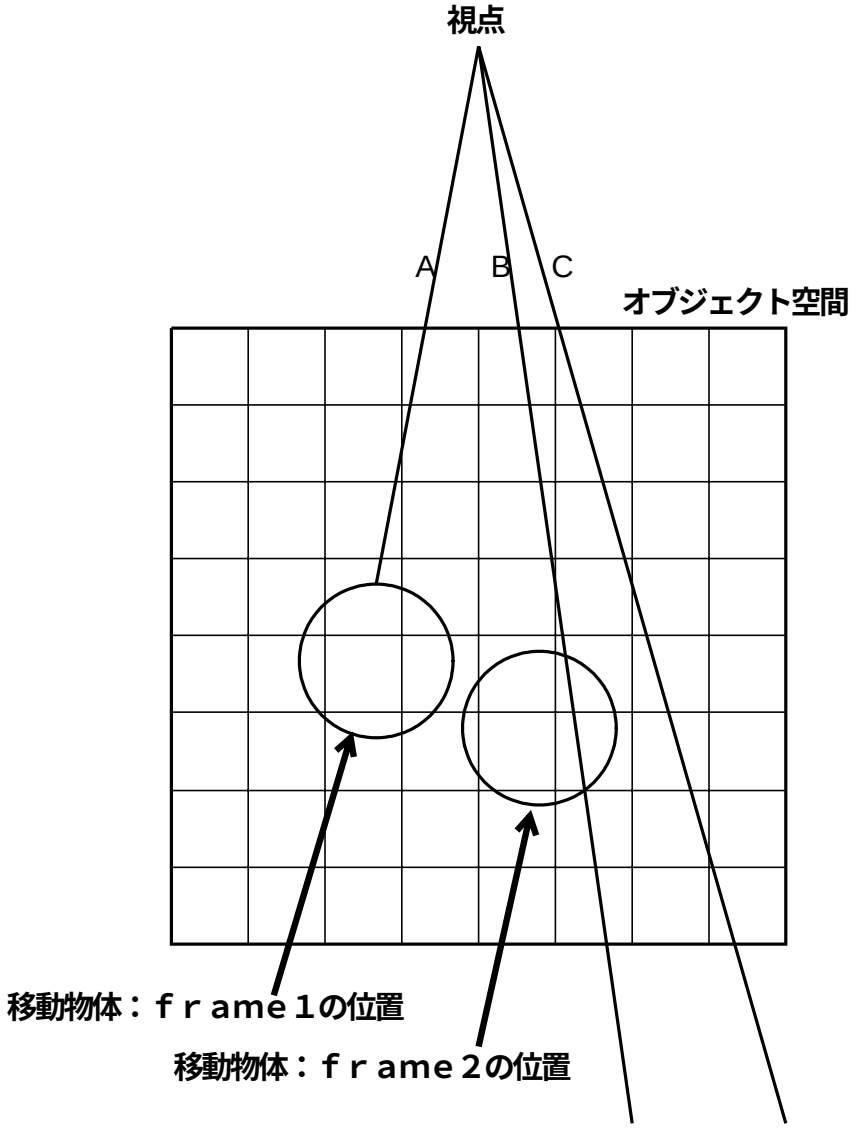


図5-4　移動物体フラグを用いた再計算の例
(移動物体の配置)

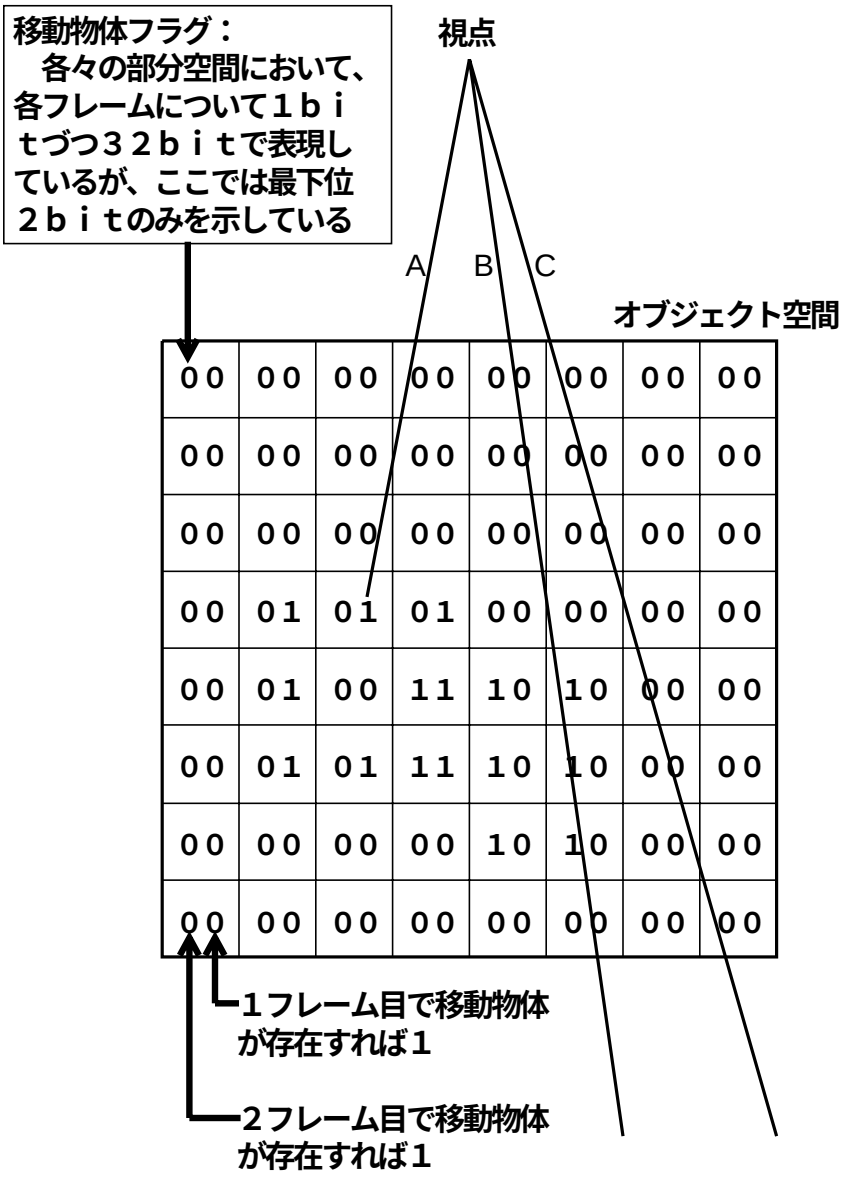


図5-5　移動物体フラグを用いた再計算の例
(各cellにおける移動物体フラグ)

表5-1　移動物体フラグによる再計算必要性の判定

追跡光線	A	B	C
通過した部分空間の移動物体フラグの論理和…（a）	ＸＸＸ０１	ＸＸＸ１０	ＸＸＸ００
前フレームと現フレームを表わすｂｉｔ…（b）	００００１１	００００１１	００００１１
（a）と（b）の論理積…（c）	００００１	００００１０	００００００
再計算の必要性	必要あり（c）<>0	必要あり（c）<>0	必要なし（c）＝0

想される画素の情報のみを選択して保存する。そのため、空間分割処理を行う以前に計画した物体の移動等の変更にしか対応できない。しかし、例えば移動しない背景の中を少数の移動物体が移動するようなアニメーションの画像生成をするような場合には、全画素を保存するよりもメモリ使用量を少なくすることが可能である。

部分更新レイトレーシング法のアルゴリズムの性能評価を行うため、各種の動画を用いて実際に画像生成を行い、計算時間、使用メモリ量などについて調べた。表5-2にその結果を示す。1枚目のフレームの場合、保存する光線追跡木の作成のために、従来の方法よりも1.3～1.4倍の処理時間が必要になる。2枚目以降のフレームでは、画像によって大きく異なるが、従来の方法に比較して1/5から1/13の時間で計算可能である。部分更新レイトレーシング法が動画生成において高速処理が可能であることが分かる。

次に部分更新レイトレーシング法に必要なメモリ量について検討する。サンプル画像の場合、最も少ない場合でも100×100画素の場合で約386Kバイトを要し、最大の場合では約1.3Mバイトも必要である。これは従来のレイトレーシングと比較して11～200倍のメモリ量である。軌跡セルの概念を用いてメモリ使用量を改善した手法でも、光線追跡木を保存するために莫大な量のメモリを必要とすることが分かる。

表5-2　通常の光線追跡法と部分更新光線追跡法の比較
（実行時間比は通常の光線追跡法を１とした値）

画　像　名	mvtst	mirror	orbit	orbit3	glass	rasen
総フレーム数	11	11	20	20	20	40
１フレーム目 (実行時間比)	1.27	1.26	1.34	1.41	1.34	1.36
２枚目以降 (実行時間比)	0.109	0.09	0.077	0.16	0.147	0.19
全画素に対する再計算画素数の割合	7.6%	7.5%	1.3%	4.0%	5.8%	10.9%
メモリ使用量 (KBytes)	806.7	998.9	385.7	809.7	1281.6	1786.8

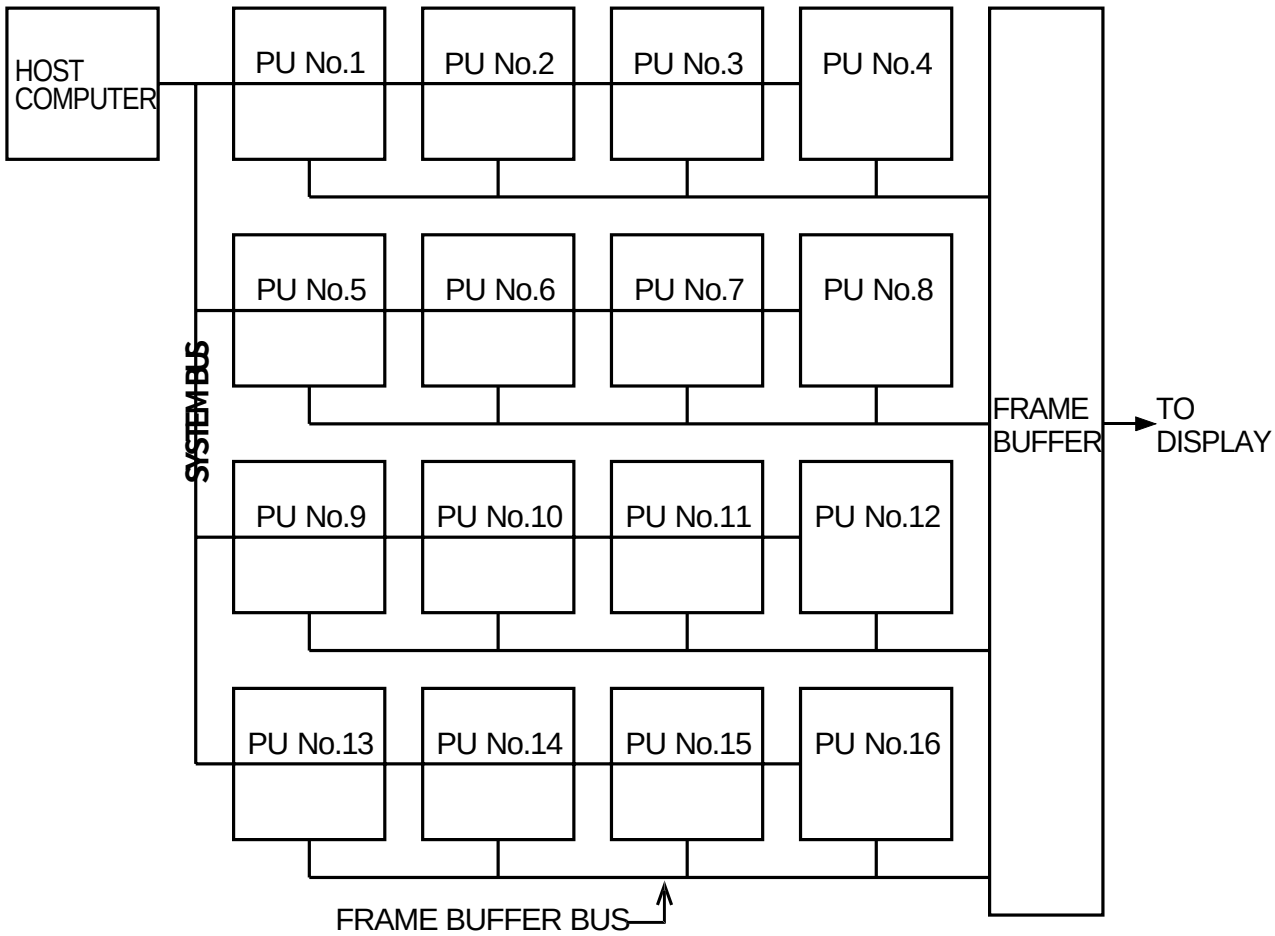
§5.3　画素分割型並列処理システム

光線追跡法は、各画素の輝度をそれぞれ独立に計算できるため、並列処理を行うことにより画像生成の高速化を図ることが容易である。並列処理を行う場合、各プロセッサにどの様に処理を割り当てるかによって処理速度が大きく異なる。今まで、種々の計算負荷割り当て法が提案されている。その中で最も簡単な方法として、画面の分割を行い分割した画素を各プロセッサに割り当てる方法がある。このような方式は画素分割型並列処理方式と呼ばれる[Nishimura 85]。 画素分割型の並列処理方式は、各プロセッサにオブジェクト空間内の全オブジェクト情報を持つ必要がある。そのためにデータメモリ量がプロセッサ台数に比例して増加するという欠点がある。 しかし、プロセッサ数を増すとそれに応じた処理能力の向上を期待することができる。

部分更新レイトレーシング法にこの画素分割型並列処理方式を導入する場合、各画素の光線追跡木のデータはそれぞれ独立であるためプロセッサ間で重複して持つ必要はない。そのため、プロセッサ数の増加によって光線追跡木の保存のための使用メモリ量が増加することはあまりない。現実には、各プロセッサ間で保存する画素数に片寄りが生じるために、全プロセッサのトータルメモリ使用量は単一プロセッサの場合よりも若干多くなる。

一般に、光線追跡木の保存のために必要なメモリ量はオブジェクト情報のためのメモリ使用量に較べて数十から数百倍に達する。したがって、数十台程度のマルチプロセッサシステムを考えると、オブジェクト情報と光線追跡木のためのメモリ使用量とを合計した全体としてのメモリ使用量は、単一プロセッサと比較して数倍程度以内であると考えられる。これは、フレーム間の相関を利用しない場合の画素分割型並列処理方式がプロセッサ台数に比例してメモリ使用量が増加するのに対し、本方式の大きな特長である。

以上の検討に基づき、部分更新レイトレーシング法のための並列処理システムとして本論文では図5-6に示すような画素分割型並列処理システムを提案する。全体のシステムはホストコンピュータ、それとシステムバスにより結合された各処理ユニット、フレームバッファバス、フレームバッファから構成される。ホストコンピュータはシステム全体の管理、オブジェクト情報の管理、各処理ユニットへの画素の割付情報の管理などを行う。各処理ユニットは、ホストコンピュータより各フレー



(PU: PROCESSING UNIT)

図5-6　画素分割型並列処理システムの構成

ム毎に移動物体のオブジェクト情報を受け取り、割り当てられた画素に関してそれぞれ独立に計算を行う。全処理ユニットが計算を終了したときそのフレームに関する処理が終了したことになる。光線追跡木を保存するためのローカルメモリは各処理ユニットに分散配置されるため、全体としては単一の計算機よりも記憶容量を大きくとることができ、光線追跡木の記憶のために膨大な記憶容量を必要とする部分更新法には適している。

各処理ユニットが計算した輝度はフレームバッファバスを通してフレームバッファに書き込まれる。なお各処理ユニットに割り当てられた画素の輝度計算は独立に行われるため、処理ユニット間の通信は行われないものとした。また、画素の処理ユニットへの割り当ては１シーン内では固定とする。これは、画素割り当てを可変にした場合、ユニット間での割り当て画素の移動が生じる可能性があるが、移動する画素に対する光線追跡木が保存されていた場合にはそれを移動するためのコストが大きくなることが予想されるためである。同様の理由により、動的な処理の割り当ても困難である。

このような並列処理システムにおいて、各処理ユニットのCPUの性能をより向上させることにより、システムのさらなる性能向上をはかることができる。すなわち、第３章で提案したジェットパイプラインを用いることによりシステム全体の処理性能をさらに向上させることが可能となる。具体的には、光線追跡法の処理ではベクトル化は一般に困難であるが、座標計算(x,y,z)および輝度(R,G,B)のように対称的な３組の計算を同時に行うことがしばしば見られる[Yoshida 85]。これらの処理において、最大４命令を並列実行可能なジェットパイプライン・アーキテクチャにおけるVLIW的な並列化が非常に有効になる。たとえば、図5-7に示したプログラム例は実際に使用した画像生成プログラムの一部であるが、明らかにX,Y,ZおよびR,G,Bに対して同様な計算を行っていることがわかる。そのため、図中の下部に示したように各命令パイプラインに対して処理を割り付けることにより、高速化が可能である。また、各パイプラインに割り付けられた処理の間でデータのやりとりが必要になったときには、共有されているレジスタを用いて行えばよい。

§5.4　シミュレーションによる性能評価

本節では、前節で提案したジェットパイプラインを用いた画素分割型並列処理方

・実際の光線追跡法のプログラムの一部

```
localeye[X] = eye[X] - (p[X] - MAXCELL/2 + 0.5) * CSIZE;
localeye[Y] = eye[Y] - (p[Y] - MAXCELL/2 + 0.5) * CSIZE;
localeye[Z] = eye[Z] - (p[Z] - MAXCELL/2 + 0.5) * CSIZE;
```

X,Y,Zの各軸に対して同様な計算を行っている

```
rr = (float) bright[R] * light.lr / ((double)(255*255));
gg = (float) bright[G] * light.lg / ((double)(255*255));
bb = (float) bright[B] * light.lb / ((double)(255*255));
c = rr*(costh*obj->dr*(1.0-light.shadow) + cosa*obj->hr);
newbr[R] = BRLIM(c);
c = gg*(costh*obj->dg*(1.0-light.shadow) + cosa*obj->hg);
newbr[G] = BRLIM(c);
c = bb*(costh*obj->db*(1.0-light.shadow) + cosa*obj->hb);
newbr[B] = BRLIM(c);
```

R,G,Bの各輝度に対して同様な計算を行っている

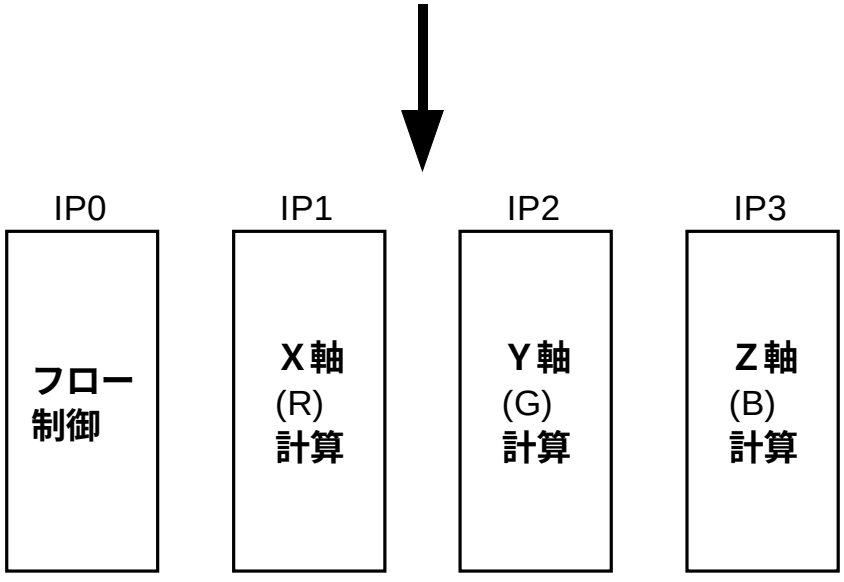


図5-7　光線追跡法のVLIW式並列化

式の計算機アーキテクチャについて、シミュレーションによる評価を行う。シミュレーションにおいては、おもに各種の負荷分散を考慮した画素割付法について評価する。また､実際の共有メモリ型並列計算機[Suzuki 88,93]上で実行した場合の結果との比較についても検討する[Katahira 90a][Horiguchi 93]。

シミュレーションは実際の計算機における実行時間に基づいた方法を採っており、また実機による実験においてはジェットパイプライン・アーキテクチャが現実にまだ存在していない以上通常型のCPUを用いているシステムを使用したが、処理ユニットのプロセッサがジェットパイプラインとなったとしても、全体の速度は向上するであろうが、負荷分散等には影響しないと考えられるため、相対的な性能としては、本節の結果とは大きく異なることはないと予想される。

まずはじめに、シミュレーションによる性能評価の結果について述べる[Katahira 88]。

各処理ユニットに画素をどのように割り当てるかによって、負荷の分散の度合が変化することが考えられる。できるだけ光線追跡木を保存する画素を各処理ユニットに分散して配置する方が、メモリ使用量、稼働率両方の改善をはかることができる。ここでは、図5-8に示した3種の割当方法(シーケンシャル型、ドット型､ブロック型)と､乱数によってランダムに処理ユニットに画素を割り当てた場合についてシミュレーションを実行した。シミュレーションは、すべて空間分割数をX,Y,Zの各座標軸当り16、画素数を160×160として行った。

図5-9に、サンプル画像"rasen"のプロセッサ数と速度向上比の関係について示した。シーケンシャル型割当と、ドット型割当ではほとんど差はなく、ほぼ台数の増加とともにリニアな速度の向上がみられる。ランダム型割当の場合は、台数の増加につれて速度向上比がわずかに低下している。ブロック型割当の場合、他の割当方法に比較してかなり悪い結果となった。このことより、負荷分散の効果はドット　シーケンシャル＞ランダム＞＞ブロックの順であることがわかる。これは、負荷の大きい部分が局在しているために、画面のある部分をまとめて一つのプロセッサに割り当てるブロック型では、負荷に片寄りが生じるためである。それに対して画面のとびとびの部分を一つのプロセッサに割り当てる方法は、平均的に負荷が分散される。

図5-10にサンプル画像"rasen"のプロセッサ数と稼働率の関係について示した。最

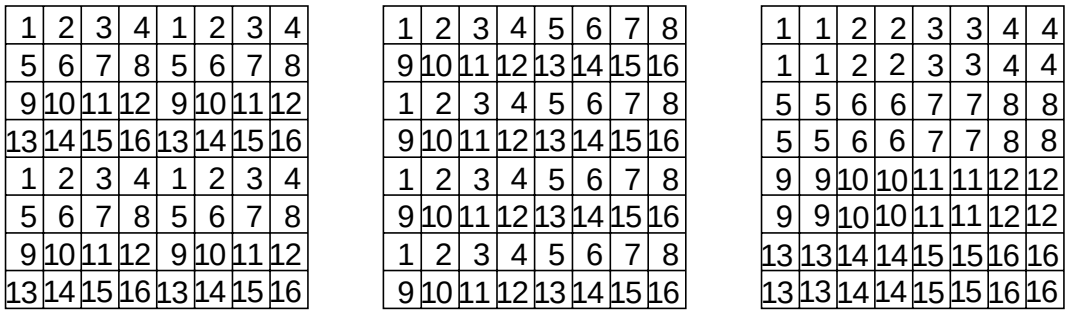


図5-8　処理ユニットへの画素割り当て法

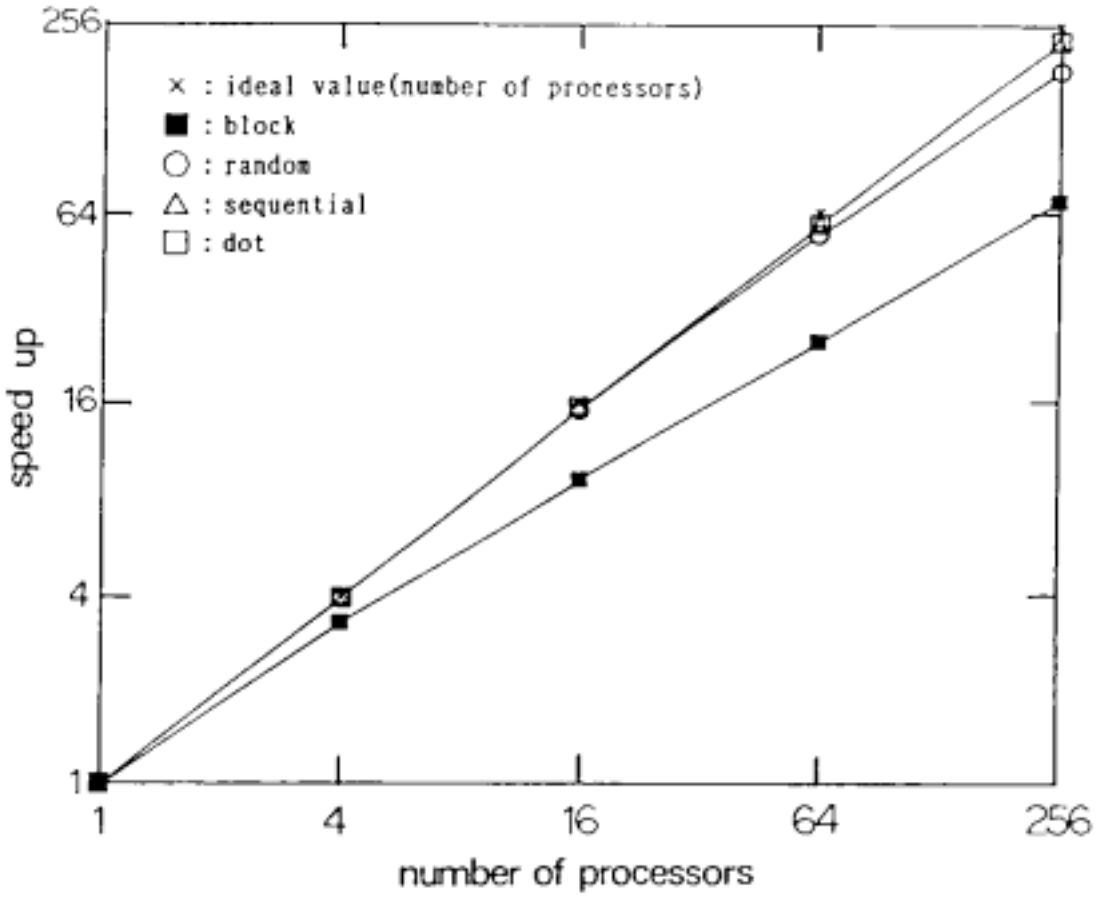


図5-9　**プロセッサ数と速度向上比の関係**(RASEN)

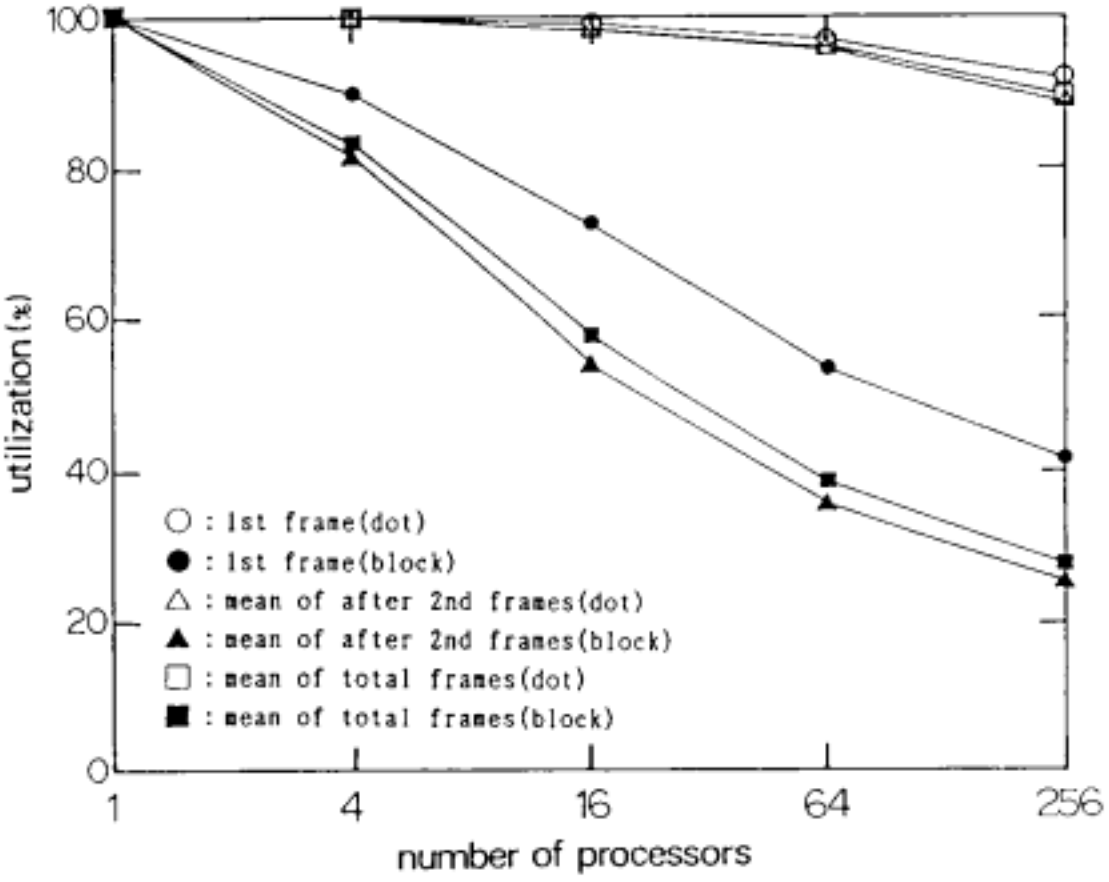


図5-10　**プロセッサ数と稼働率の関係**(RASEN)

も負荷分散の効果が高いと思われるドット型と、最も低いブロック型の結果を示した。なお、稼働率は各プロセッサの稼働時間の総和を、画像生成時間とプロセッサ数の積で割った値である。ドット型の場合、プロセッサ数256個の場合で稼働率は90%程度を維持している。ブロック型の場合では、プロセッサ数の増加により稼働率は低下する。また、ブロック型においては、1枚目と比較して2枚目以降の稼働率がかなり下がっている。これは、光線追跡木を保存した画素数がプロセッサによって大きくばらついているためであると考えられる。

次に、これらのシミュレーション結果に基づき実際の並列計算機上に部分更新法をインプリメントし、実験した結果について述べる[Katahira 90a][Horiguchi 93]。実験には、日本アイ・ビー・エム(株)により試作された共有メモリ型の密結合マルチプロセッサシステムであるTOP-1を使用した[Suzuki 88,93]。そのシステム構成図を図5-11に示す。実際に使用したのは10プロセッサ構成のものであり、全体で30MIPS程度の処理能力を有している。

シュミレーションの際に仮定したシステムは疎結合型であり、全てのプロセッサはローカルメモリのみを持ち、共有メモリは持たないものとしていた。一方、実験に使用した並列計算機は共有メモリ型であり、共有メモリ上にオブジェクト等のプロセス間で共通なデータを共有できる点が異なる。また、専用のフレームバッファもないため、計算された各画素の輝度は共有メモリ上に配置された配列に一旦書き込まれ、計算終了後にファイルに書き出される。

実験は、6種のサンプル画像について画素数160×160、オブジェクト空間の分割数は1辺当り16で行い、画素割当法はブロック、シーケンシャル、ドットの3種類を用いた。プログラムの並列化はUNIXオペレーティングシステムのforkシステムコールにより、複数のプロセスを生成することにより行った。生成された各プロセスは、自分のプロセス生成順を考慮して自分がどの画素を担当すればよいかを判断することにより、処理する画素の割り当てが実現されている。また、生成されたプロセスは、TOP-1オペレーティングシステムにより各プロセッサに割り当てられる。なお、図5-12に複数プロセスの並列動作の様子を示す。

並列処理を行うプログラムは、それぞれ並列動作させるプロセス数を2,4,8として実行した。一部の画像では、16プロセスについても実行してみた。ただし、実際にユーザ・プロセスが利用できるプロセッサは最大8台であるので、8プロセスまでは

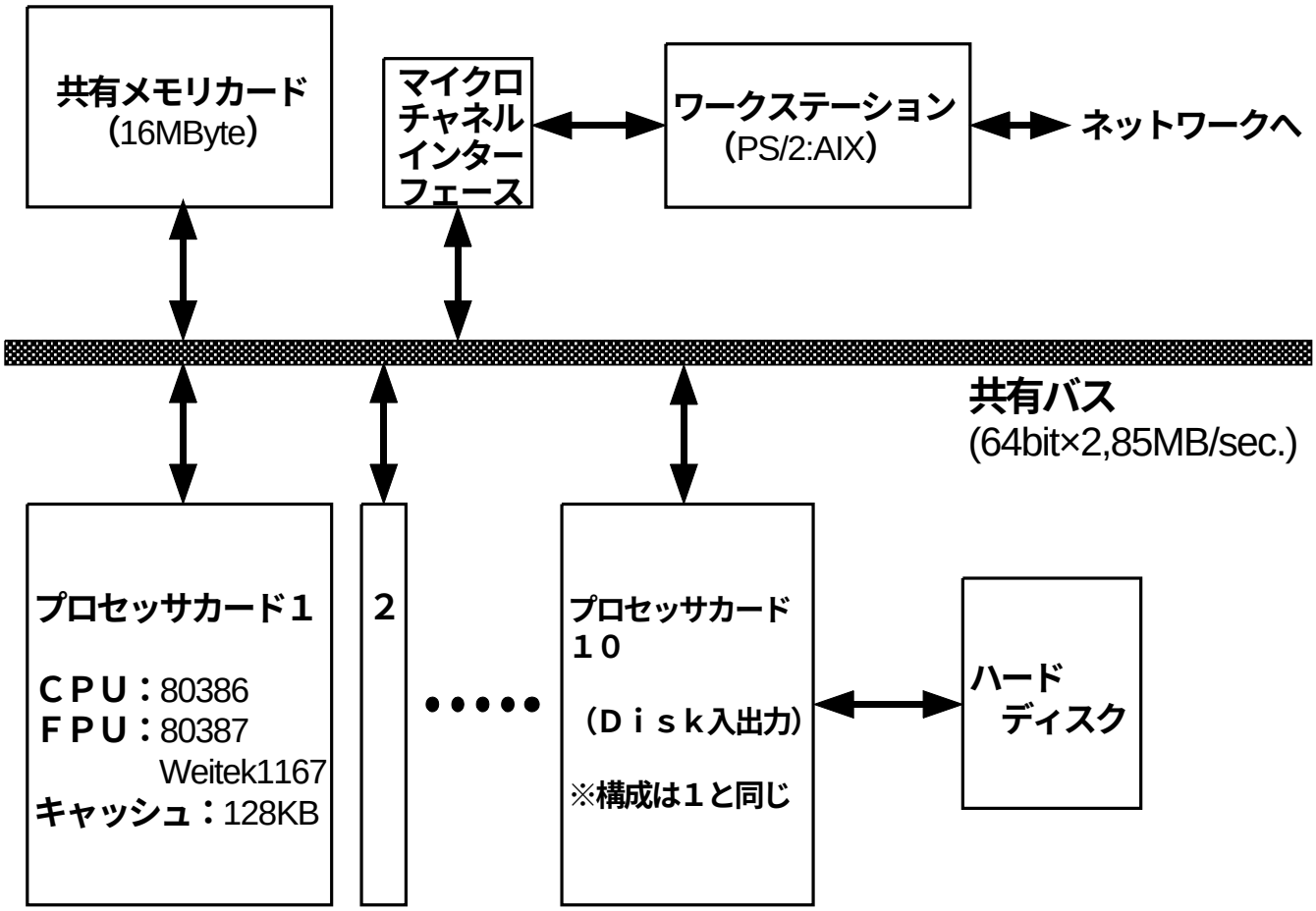


図5-11　**共有メモリ型並列計算機システム**(TOP-1)**の構成**

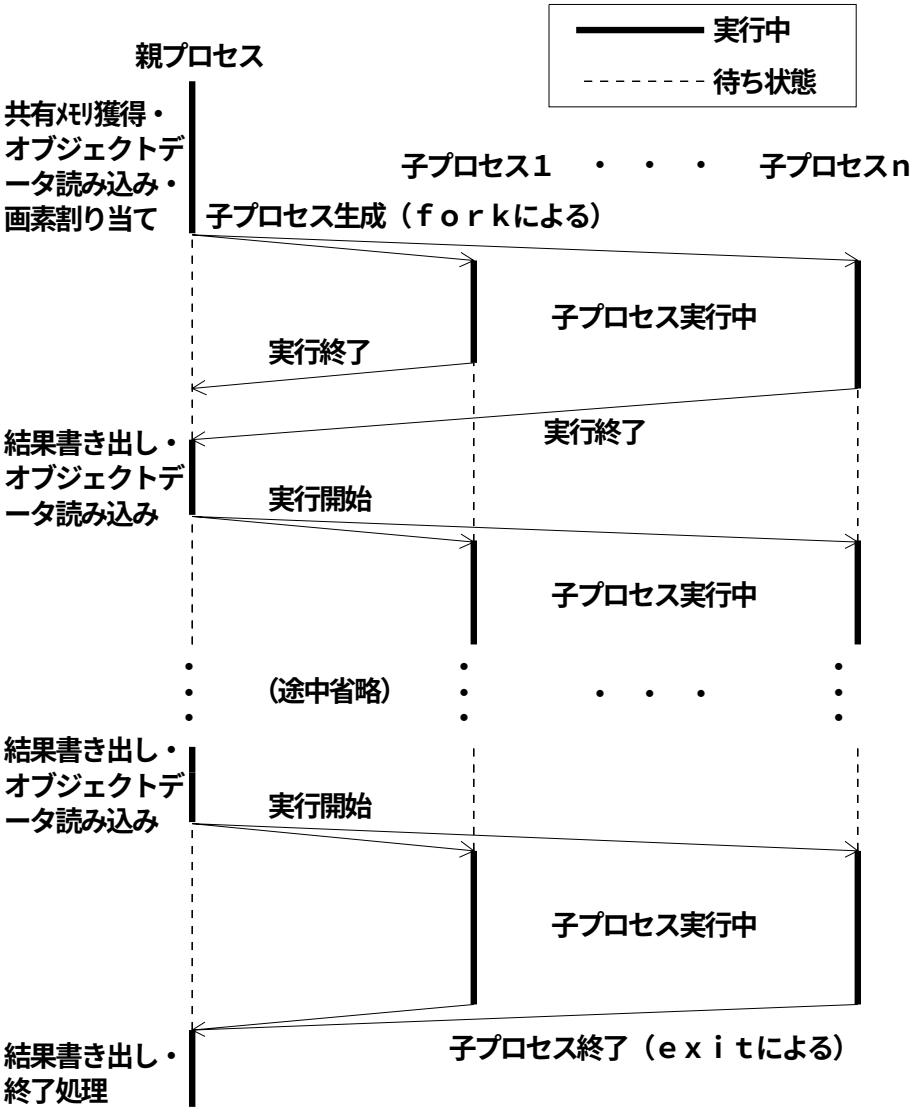


図5-12　共有メモリ型並列計算機システム上での並列実行

並列動作が可能であるが､16プロセスの場合にはあふれたプロセスがキューに入れられて実行待ちとなり、すべてのプロセスが並列に動作するわけではない。

図5-13～図5-15に､サンプル画像"MVTST"についてプロセス数を変えて実行した結果を示す。なお各プロセスに対する画素割当法はシーケンシャル型を用いた。それぞれ、全フレーム、1フレーム目、2フレーム目以降の平均の実行時間である｡また、比較のため、他のシステム（NEWS830,SUN-4/260,M-360 UTS）上で実行した結果も示した。なお､+Oが付いているものはプログラムのコンパイル時に最適化を行ったことを示す。

理想的には、プロセス１つの場合の実行時間と比較して並列動作させるプロセス数倍のスピードアップが達成できることが望ましいが、実際には負荷の偏りや並列動作のためのオーバーヘッドなどのためにそこまで達することはないのが普通である。1フレーム目の実行結果では、2プロセスで1.9倍､4プロセスで3.7倍とかなり理想値に近い結果が得られているが、8プロセスでは6.75倍とやや低下する。また､16プロセスでは、システムが持つプロセッサ数を越えているため、かえって遅くなっている。一方、2フレーム目以降の平均の実行結果には、プロセス数よりも大きなスピードアップが2,4プロセスの時に得られている。これは､膨大なメモリを使用する部分更新光線追跡法が並列化されることにより、1プロセス当りの使用メモリ量が減少し、キャッシュメモリの効果が増したことによるものと考えられる。

次に、画素割当法による負荷分散の効果について示す。実験ではブロック型(B)、シーケンシャル型(S)、ドット形(D)の3種類の画素割当法について調べた｡サンプル画像"ORBIT3"についての実験結果を、図5-16～図5-18に示す。1フレーム目では全ての画素が計算されるため､2フレーム目以降の平均と比較して3種の割当法の間の差は小さくなっている。しかし、2フレーム目以降の計算においては、再計算の行われる画素が画面上に偏って存在するようになるため、負荷分散の効果の高いシーケンシャル形、ドット形とブロック形の差が大きく開くようになる。これはシミュレーション結果と同様であることが確かめられた。また、実験の結果、一般にシーケンシャル形の方がわずかにドット形よりも速度向上度が上回っていることが判明した。

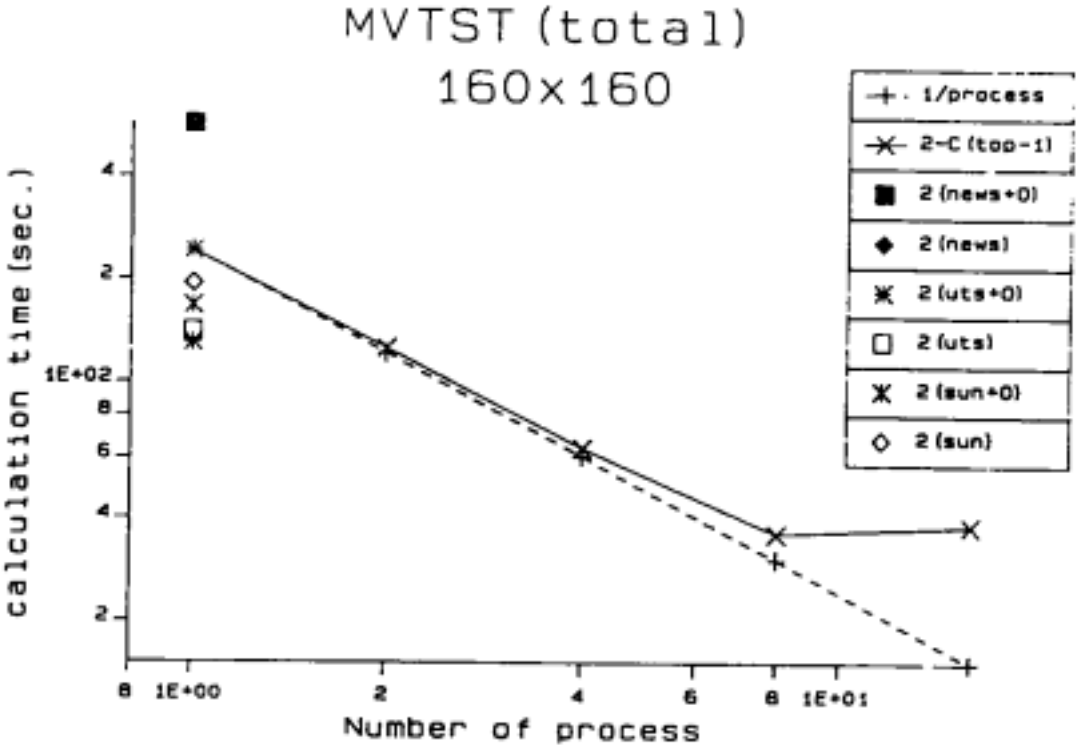


図5-13　**プロセス数と実行時間の関係**(MVTST,**全フレーム**)

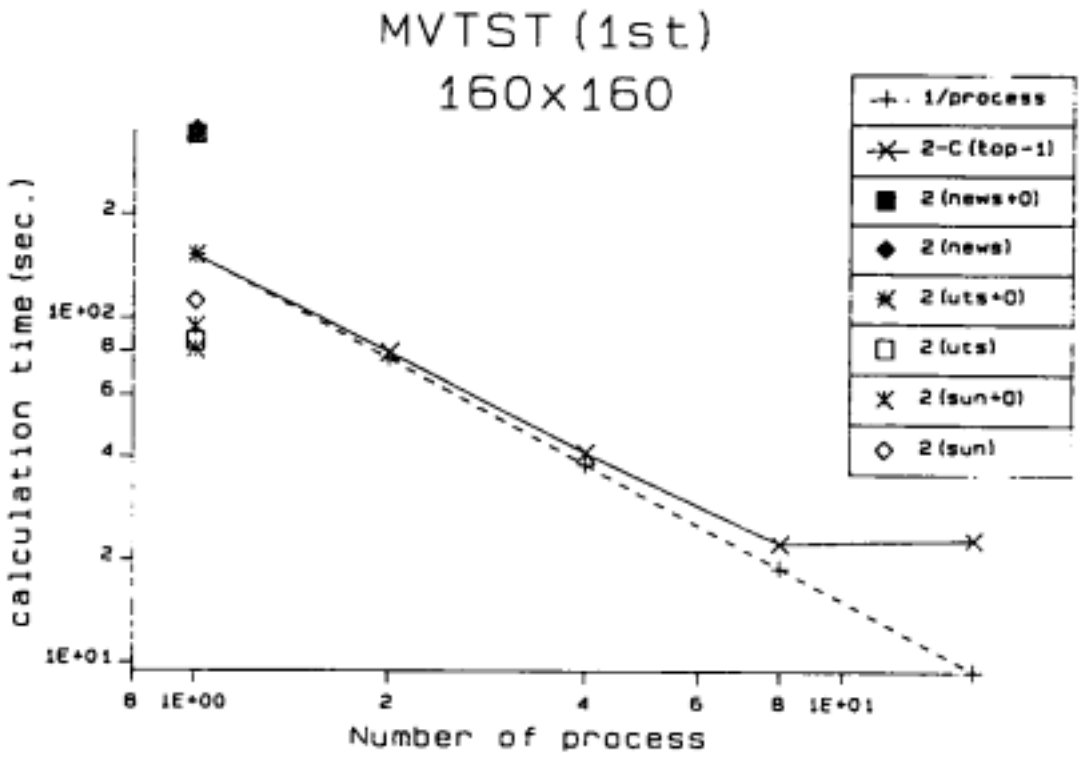


図5-14　**プロセス数と実行時間の関係**(MVTST,1**フレーム目**)

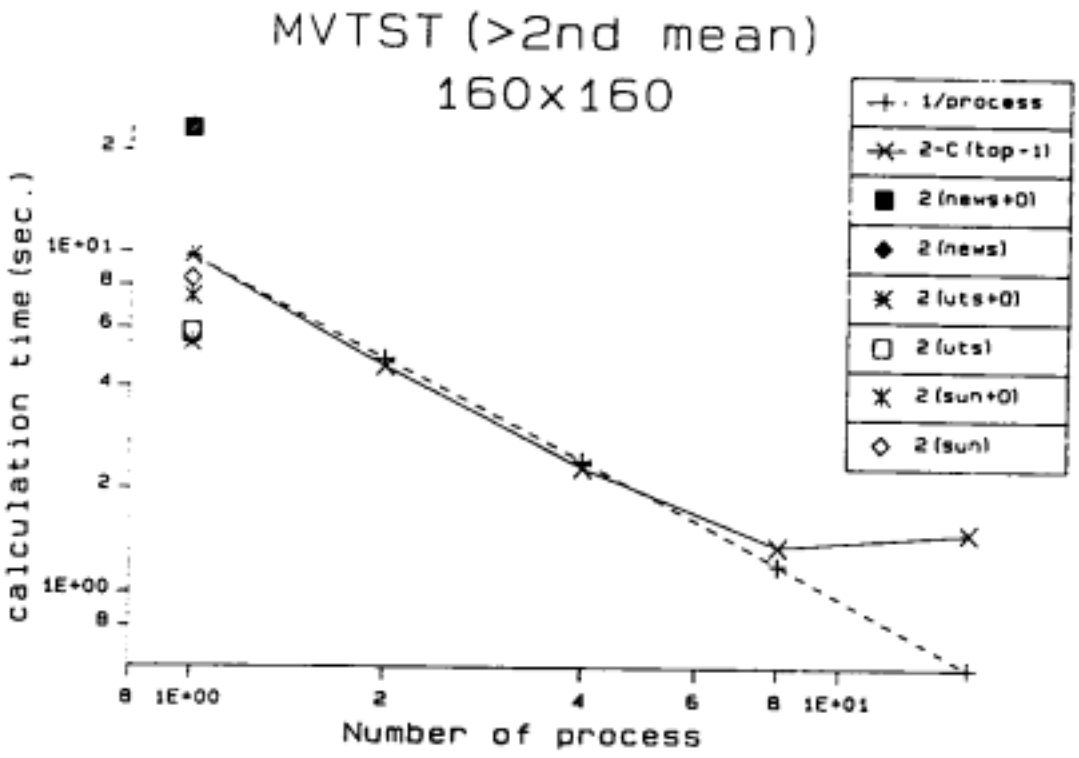


図5-15　**プロセス数と実行時間の関係**(MVTST,2**フレーム目以降**)

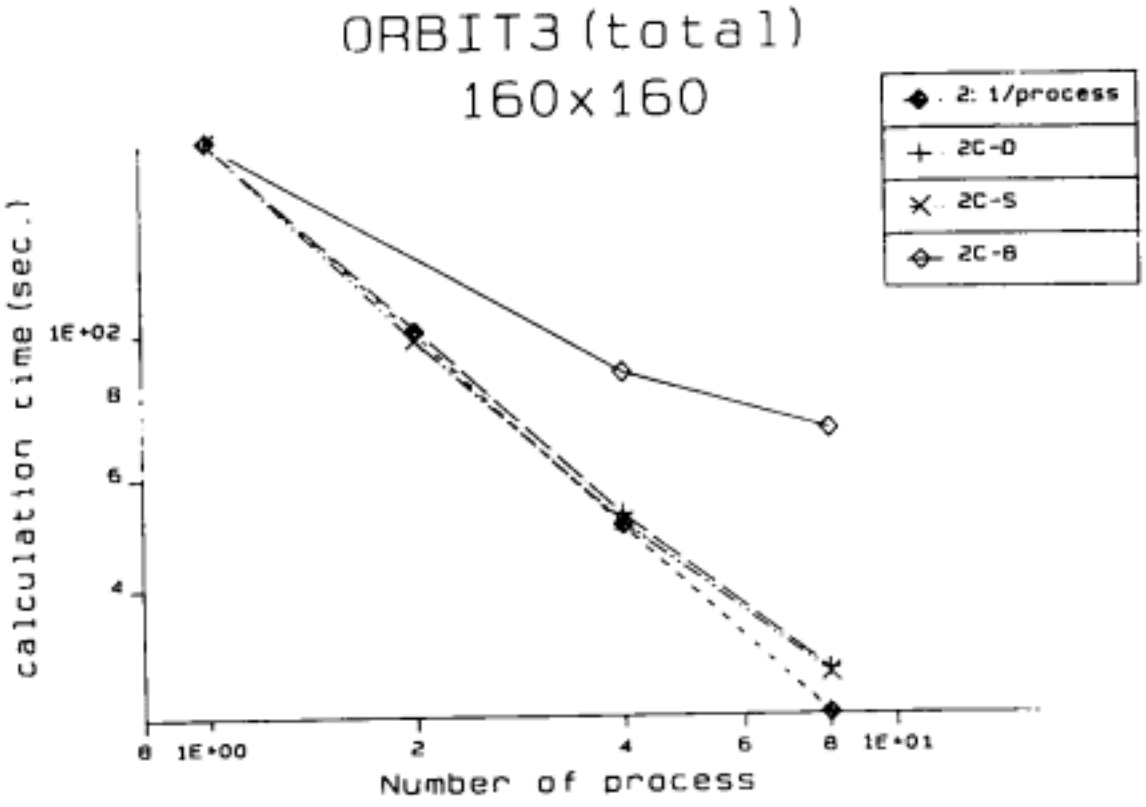


図5-16　**画素割り当て法と実行時間の関係**(ORBIT3,**全フレーム**)

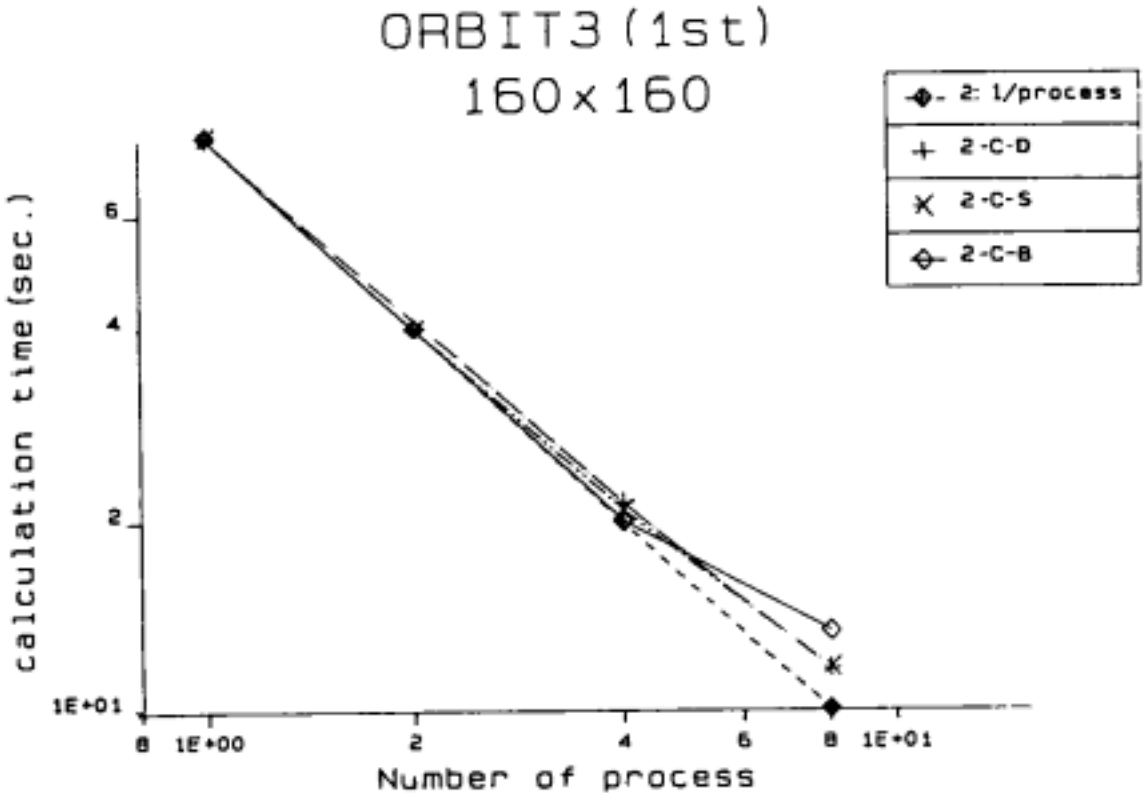


図5-17　**画素割り当て法と実行時間の関係**(ORBIT3,1**フレーム目**)

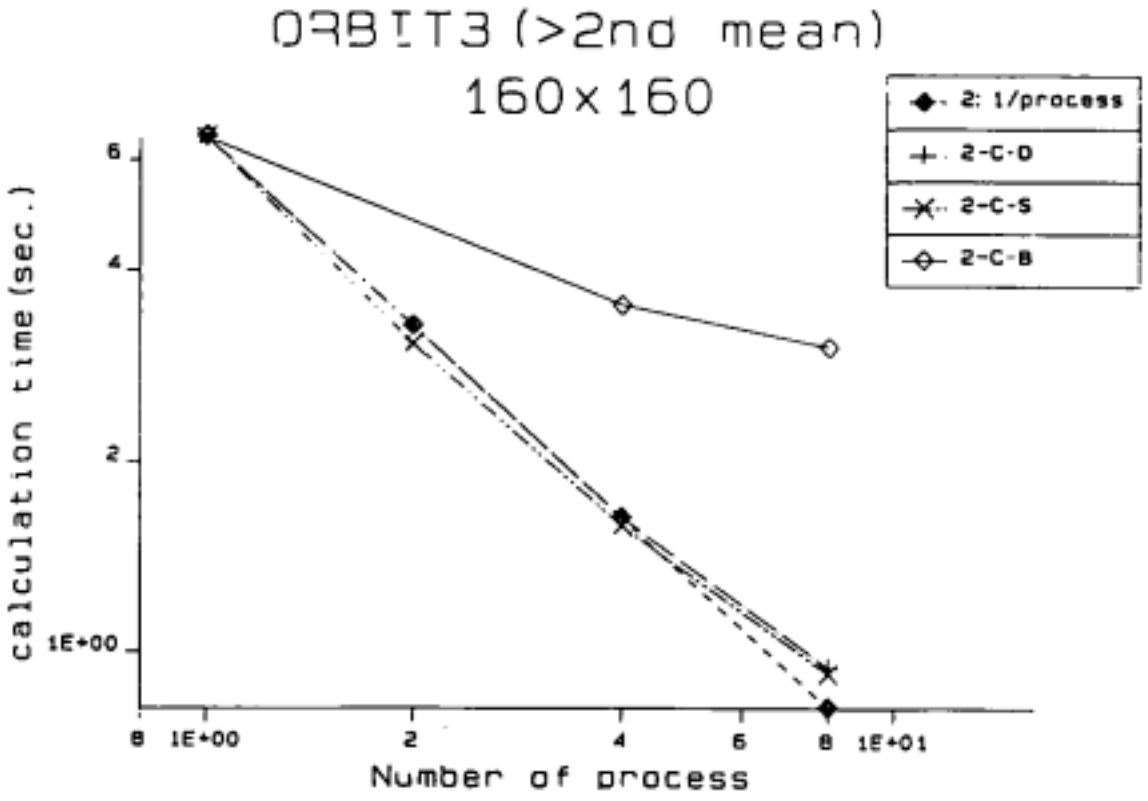


図5-18　**画素割り当て法と実行時間の関係**(ORBIT3,2**フレーム目以降**)

§5.5　結言

本章では、前章までに提案したジェットパイプラインの一応用例として、動画像生成用並列処理システムについて考察した。まず、光線追跡法による動画像生成アルゴリズムとして、動画像の１シーン中の各フレーム間で変化している部分のみを再計算することにより高速化を図る部分更新光線追跡法について、その記憶容量の低減化と高速化を目的とした新たなアルゴリズムを提案し、実際に２フレーム以降の再計算においてはかなり高速化可能なことを確認した。つぎに、この部分光線追跡法を用いてさらに高速に画像を生成するための並列処理システムを提案し、シミュレーションと実際の共有メモリ型密結合並列計算機上での実行実験により性能評価を行った。その結果、負荷分散を考慮した処理の割り当て法を用いることによって、高速な画像生成の可能性が明らかとなった。また、光線追跡法における画像生成アルゴリズム中に含まれる対称性に基づいた並列性を利用することにより、ジェットパイプライン・アーキテクチャCPUを並列処理システムの各処理ユニットに用いれば、さらに高速化可能であることが示唆された。