

3次元立体集合演算のための並列プロセッサ・システム

新實 治男 柴山 潔 萩原 宏

京都大学・工学部・情報工学教室

3次元形状定義処理において最も多くの計算量を占める立体集合演算について、その処理を高速化するための並列処理アルゴリズム、およびその効率の実行のための並列プロセッサ・システムの構成方式を提案する。本方式では、立体構成面どうしの交差計算と交線生成に要する処理時間が全処理時間の大部分を占めることに注目し、この部分の処理を重点的に並列化することによって集合演算全体的高速化を図っている。そのために、 2^N 個のプロセッサを2分木状に接続し、プロセッサ間で最大 N 個のメモリを共有するとともに、すべての処理段階において、プロセッサ間でのメモリ・アクセス競合が起り得ないようにした結合ネットワーク構造を考案した。

A MULTIPROCESSOR SYSTEM FOR SET OPERATIONS OF 3-DIMENSIONAL SOLID OBJECTS

Haruo Niimi Kiyoshi Shibayama Hiroshi Hagiwara

Department of Information Science, Faculty of Engineering, Kyoto University

Yoshida Hon-machi, Sakyo-ku, Kyoto 606, Japan

This paper describes a parallel algorithm and a multiprocessor system organization for high-speed set operations of 3-dimensional solid objects. We focused that most of the processing time is spent to evaluate the intersections of faces and to create new edges. To increase the system's throughput, we propose a novel multiprocessor system organization provided with 2^N processing modules which are mutually connected in the form of a binary tree network. These processing modules share at most N memory planes which exclude different processing modules the simultaneous access. The efficiency of the proposed system is confirmed by software simulations.

1 はじめに

3次元形状のモデリング作業の効率化を図るためには、図形定義機能の高度化を図る一方で、処理の高速化によってシステムの応答時間を短縮し、利用者の試行錯誤の過程を円滑化することが重要である。本報告では、形状定義処理のうち、最も計算量の多い立体集合演算について、その高速化のための並列処理アルゴリズム、およびこれを効率よく実行する並列プロセッサ・システムの構成方式を提案する。

形状の表現方法は、CSG (constructive solid geometry) と B-Reps (boundary representations) の二つに大きく分類できる^[1]。CSG は目的形状を基本的な形状(プリミティブ)の集合演算(和、積、差)により記述するもので、人間が直感的に理解しやすいという長所がある。これに対し、B-Reps は面・稜線・頂点およびこれらの接続関係による表現であり、計算機による処理には適している反面、人間が形状を直感的に把握することが難しい。そこで我々のシステムでは、形状定義処理への入力となる形状記述には CSG を用い^[2]、その形状の図形表示は、CSG 記述を B-Reps に変換した後に実行するという方式を採用している。この方法は、B-Reps に変換する際の計算で誤差が混入したり^[3]、曲面を複数の平面で近似したりするので、正確な表示ができないという欠点がある。しかし、Zバッファ法やスキャンライン法などによる高速表示処理に適しており、我々の目的に合致している。

本報告では、この CSG 記述から最終的な B-Reps を得るための「立体集合演算」を高速化するための並列プロセッサ・システムについて述べる。特に、全処理時間の大部分を占める、「立体構成面どうしの交差計算」および「交線生成」に注目し、この処理を重点的に並列化することによって、全体の高速化を図っている。

2 立体集合演算の並列処理

2.1 並列処理方式の概要

形状のプリミティブを「葉」、集合演算子を中間の「節」として、目的形状を表現する2分木を「CSG 木」と呼ぶ。CSG から目的形状の B-Reps を得る手法には次の二つがある。

- i) CSG 木の全てのプリミティブ間での交差計算を行った後、目的形状の B-Reps を構成する方法。
- ii) CSG 木を帰りがけ順にたどりながら二つの B-Reps 間の集合演算を繰り返すことにより目的形状を得る方法。

i) はプリミティブを方程式の形で与える場合に適しているものの、プリミティブ数が増加するにつれて、計算すべき交線数が急速に増加する欠点がある。ii) はプリミティブを B-Reps で与える場合に適しており、交線数が爆発的に増加することはない。

また、立体集合演算を並列処理する際の負荷分散単位としては、

- i) 二つの B-Reps 間の集合演算。
- ii) 二つの構成面間の交差計算。

という、2とおりが考えられる。i) の場合、各プロセッサに CSG 木中の各中間節を割当てることになる。ところが、集合演算可能な節は CSG 木の構造に大きく依存し、また、節以下の部分木の構造により処理量が大きく異なる。したがって、i) では、常に効率のよい並列処理を行えるとは限らない。ii) の場合は、一つの間節に対応する集合演算を複数のプロセッサで行うため、i) よりも負荷の偏りが少ない。もちろん、簡単なプリミティブどうしの集合演算では、プロセッサ台数に対して十分な数の構成面が得られない場合もある。しかし、一般には、CSG 木の根に近づくにつれて立体の構成面数が増加するので、十分並列処理効果を発揮できると考えてよい。

以上のことから、本システムでは「CSG 木を帰りがけ順にたどりながら各節ごとに、構成面を処理単位として並列処理を行う」方式を採用している(図1)。以下、B-Reps を「立体」と呼ぶ。

2.2 立体集合演算のアルゴリズム

立体集合演算は、以下の3段階から成る(図2)。

- i) 交差計算: 一方の立体 A の稜線 E を他方の立体 B の構成面 F で切り取る。
- ii) 交線生成: i) の結果、二つの立体間の構成面と稜線との交点が求まったので、次に、交点の組から二つの構成面 F_a 、 F_b の交線を求める。
- iii) 立体の再構成: i) ii) により、集合演算後の立体に含まれる全ての構成要素が得られている。この段階では、構成面ごとに面の形状を再構成した後、不要な頂点と法線ベクトルを除き、新しい立体を構成する。

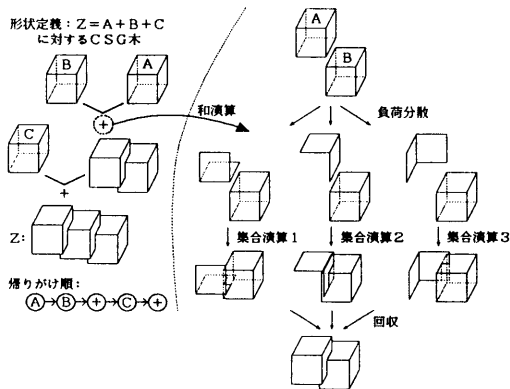


図 1: 集合演算の並列処理

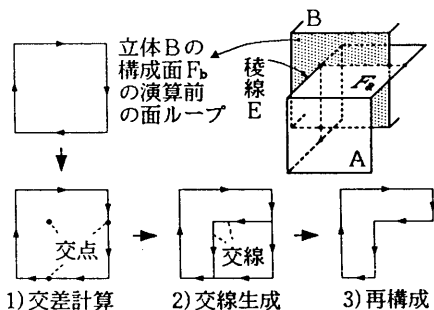


図 2: 集合演算の基本処理過程

2.3 立体集合演算の並列処理

立体集合演算に要する各処理段階の計算時間は以下のように見積ることができる。立体 A 、 B それぞれの構成面数を m 、 n 、二つの構成面間の交差計算および交線生成に要する平均計算時間を C 、稜線の再結合により一つの新しい構成面を求めるのに要する平均計算時間を M とすると、

- 交差計算、交線生成に要する総計算時間: $C \cdot m \cdot n$,

- 立体の再構成に要する総計算時間: $M \cdot (m+n)$, となる。経験的に $C > M$ であることを考慮すると、交差計算および交線生成の処理時間が全処理時間の大部分を占めると判断してよい。したがって、この処理の並列化による集合演算の高速化が非常に有効であると言える。

目的立体を得るには、一般に集合演算を複数回行う必要があり、次のような過程で並列処理を実行す

ることになる。

- まず、CSG 木から、実行すべき集合演算 $A \text{ op } B$ を選ぶ。これで目的立体が得られれば終了。
- A の各構成面を担当するプロセッサを決める。
- A 、 B を全プロセッサにブロードキャストする。
- 各プロセッサで A の担当構成面と B との交差計算および交線生成を行う。
- 計算結果を一つのプロセッサに集める。
- 立体の再構成を行い、さらに集合演算を行う場合には、新たに得た立体を再び複数のプロセッサに対してブロードキャストして i) へ戻る。

iv) のうち、交線生成についてはプロセッサ間で重複することはない。しかし、構成面の境界稜線については、二つのプロセッサでそれぞれ構成面との交差計算を行うことになり、処理が重複する。v) は、二つのプロセッサの計算結果を一つに合成する処理を繰返すことにより行う。具体的には、稜線・交点・交線の合成、稜線と立体の包含関係の合成からなる。

以上の処理の流れは、 k 段のパイプライン処理を行う $2^k - 1$ 個のプロセッサ・システムにそのまま写像することができる (図 3 参照)。しかし、パイプラインの 1 段目に相当する iv) の処理は、第 2 ~ k 段目に相当する v) vi) の処理に比べて非常に処理量が多い。したがって、このような単純なシステム構成では第 2 ~ k 段目のプロセッサの使用効率が低くなる。

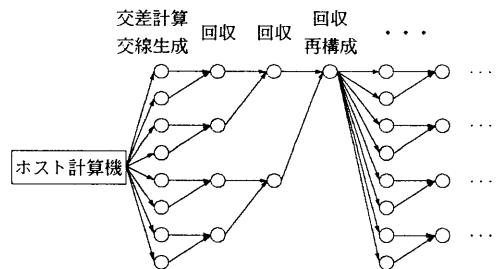


図 3: $2^k - 1$ 個のプロセッサの k 段縦続接続

そこで、空間的なデータの流れに着目し、プロセッサ台数を約半分に減らすことを考えた (図 4)。この場合には、全てのプロセッサで iv) の処理を並列に実行するため、ハードウェアの使用効率が非常に高い。ただし、処理結果の合成が進むにつれて待機プロセッサが増加し、vi) の処理を行う際にはただ一つのプロセッサしか動作しない。しかし、プロセッサの待機時間は、全体の処理時間に比べて非常に少ないと考えられる。

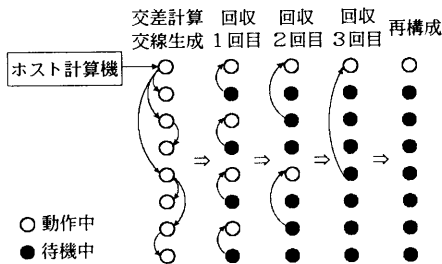


図 4: 2^{k-1} 個のプロセッサ間でのデータの流れ

3 専用並列プロセッサ・システム

3.1 ハードウェア構成

立体データを全てメモリ上に構築すると仮定した場合、ハードウェア構成に対する要求を列挙すると、以下ようになる。

- i) 立体データが全てのプロセッサに素早く行き渡ること。
- ii) 計算結果の回収を、メモリ間の異なる内容のみを一方へ複写するという処理だけでできること。
- iii) 全ての交差計算、交線生成結果を最終的に一つのプロセッサから参照でき、新たに得た立体を再びブロードキャストできること。

逆にハードウェア構成に対する譲歩を列挙すると、次のようになる。

- i) 立体をブロードキャストする際および計算結果を回収する際のデータ・バスは、あらかじめ固定でよい。したがって任意のプロセッサとメモリ間を結合できる結合網は不要である。
- ii) ブロードキャストにより、交差計算と交線生成時にはプロセッサ間でメモリを共有する必要がない。

以上のことから、本システムでは、プロセッサとメモリを対にした「モジュール」を 2^N 個 2 分木状に配置し、プロセッサ間で最大 N 個のメモリを共有する構成を採用した。

各モジュールは、プロセッサ、局所メモリ、共有メモリ、および二つのバス・スイッチからなる (図 5)。モジュール内のバスを「内部バス」、モジュール間をつなぐバスを「外部バス」と呼ぶ。また、内部バスのうち、プロセッサ側バス・スイッチと外部バスをつなぐものを「能動バス」、メモリ側バス・スイッチと N 本の外部バスをつなぐものをそれぞれ「受動バス

1 ~ N 」と呼ぶ。プロセッサからは三つのメモリ・バスが出ている。一つは局所メモリに接続しており、他の二つはそれぞれバス・スイッチを介して自モジュールあるいは他モジュール内の共有メモリに接続している。局所メモリは作業用に用いる。共有メモリには立体データを格納し、異なるプロセッサによって段階的に処理を加えていく形で集合演算を実行する。共有メモリの切り換えは、各プロセッサが二つのバス・スイッチにより、アクセス対象メモリを順に選択することにより実現する。各プロセッサは、アクセスする共有メモリがどのモジュール内にあるかを意識する必要はない。

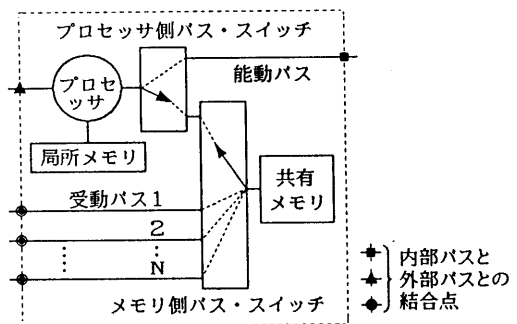


図 5: モジュールの構成

2^N 個のモジュールは、外部バスにより 2 分木状に結合する。図 6 に $N = 3$ の場合の例を示す。図 6 において、あるモジュールよりもより左にあるモジュールを「上位モジュール」、より右にあるモジュールを「下位モジュール」と呼んでいる。また、葉の部分にあたる $2^N - 1$ 個のモジュールを「末端モジュール」と呼び、最上位モジュールを特に「ルート・モジュール」と呼ぶ。プロセッサについても同様の呼称とし、かつ、左から右、上から下の順に番号 $1, 2, \dots, 2^N$ をつける。メモリ側バス・スイッチの受動バスのポートには、上から $1, 2, \dots, N$ の順にネットワーク番号をつける。ネットワーク番号はバス・スイッチの切り換えに用い (3.2 節で詳述)、プロセッサ番号は処理担当プロセッサ番号として各構成面に付加するものである (3.3 節で詳述)。

各プロセッサはバス・スイッチが固定されている期間内は独立に処理を行う。この際、全ての動作状態において複数のプロセッサが一つのメモリを同時にアクセスすることはないため、アービトレーション

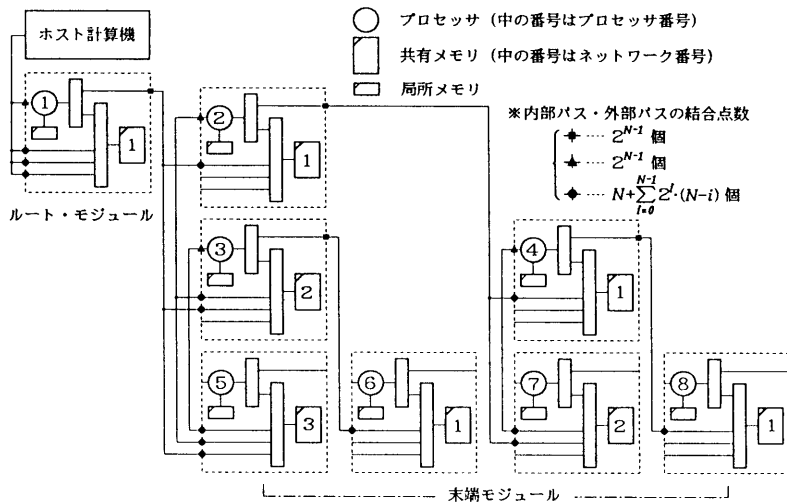


図 6: モジュール間結合方式

機構は不要である。したがって、各プロセッサは単一プロセッサとしての処理能力を最大限に発揮できる。また、プロセッサ・共有メモリ間のバス・スイッチ数は常に2であることから、システムを拡張しても素子遅延に関する限りメモリ・アクセス速度は低下しない。

本システムの諸元を以下に示す。

- プロセッサ数、外部バス数 $\dots 2^N$
- 外部と内部のバス結合点数 $\dots 3 \cdot 2^N - 2$
- バス・スイッチの最大接点数 $\dots N + 1$
- ブロードキャスト時のメモリ $\dots N$
間データ転送段数、および計算結果回収の所要処理段数

3.2 モジュールの動作

各モジュールは次に示す i) から iv) の動作モードを順に遷移する。各モジュールの動作モードは、上位モジュールからの命令、および下位モジュールの状態により決定される。バス・スイッチの切り換えは、ネットワーク番号 k と動作モードにより決定することができ、複雑なルーティング・アルゴリズムは必要ない。

i) ブロードキャスト・スレーブ (BS) モード (ブロードキャスト時): ルート・モジュールからのブロードキャストは N 段階に分けて行う。プロ

セッサ側バス・スイッチを能動バス側へ、メモリ側バス・スイッチをネットワーク番号 k と同じ受動バス k 側へ切り換える。そして自モジュール内の共有メモリに立体データが転送されてくるのを待つ。転送終了後、直ちに ii) の BM モードへ遷移する。

ii) ブロードキャスト・マスタ (BM) モード (ブロードキャスト時): ブロードキャストの段階によらず、末端モジュールの場合、直ちに上位モジュールへ処理終了を通知する。その他のモジュールの場合、能動バスとプロセッサ、受動バス 1 と共有メモリをそれぞれ接続するようにバス・スイッチを切り換え、下位モジュールに対して立体データをブロードキャストする。そして、全ての下位モジュールのブロードキャスト処理の終了を確認した後、上位モジュールへ処理終了を通知する。このとき、ルート・プロセッサを除くプロセッサは、ブロードキャストすべきデータの一つ下のモジュール内の共有メモリから読み出すことになる。そしてこれを右側の下位モジュールに対して、ブロードキャストする。

iii) スタンドアロン (S) モード (交差計算および交線生成時): モジュール内のプロセッサと共有メモリを接続するようにバス・スイッチを切り換え、モジュール内で処理を行う。

iv) マージ (M) モード (計算結果の回収および立体の再構成時): 回収は N 段階に分けて行う。 j 段

目では、能動バスとプロセッサ、受動バス j と共有メモリをそれぞれ接続するようにバス・スイッチを切り換える。各モジュールは、全ての下位モジュールの $j-1$ 段目の回収処理終了を確認すると、下位モジュールに対してバス切り換え命令を出す。そして j 段目の回収処理を行った後、上位モジュールへ処理終了を通知する。 N 段目の回収処理が終了した時点で、全ての計算結果はルート・モジュール内の共有メモリに集積されている。最後にルート・プロセッサで立体の再構成を行い、集合演算を終了する。さらに集合演算を続ける場合は再び i へ戻り、以上の処理を繰り返す。

3.3 プロセッサのメモリ参照

立体 A 、 B 間の演算により立体 C を求める場合について、各プロセッサのメモリ参照が前節で述べた各動作モードごとにどのように変化するかを図 7 に示す。

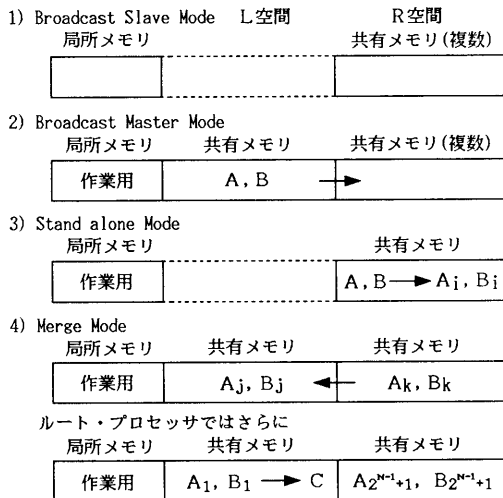


図 7: プロセッサのメモリ参照

各プロセッサのアドレス空間を三つの連続領域に分割し、一つを局所メモリ空間、残る二つを共有メモリ空間としている。そして、プロセッサに接続している共有メモリ用バスのうち、直接外部バスに接続するものをLバス、バス・スイッチを介するものをRバスとし、それぞれを通して参照するメモリ空間をL空間、R空間と呼んでいる。なお、モジュール

総数は 2^N とする。また、図中点線部は、メモリが接続されていないことを表す。

Sモードでは、番号 i のプロセッサで、 A の担当構成面と B の全構成面との交点計算および交線生成を行い、R空間上に A_i と B_i を得る。

Mモードでは、回収段数を m ($1 \leq m \leq N$) とすると、プロセッサ i ($1 \leq i \leq 2^{N-m}$) は、 A_j と A_k 、 B_j と B_k をそれぞれ合成し、L空間上に改めて A'_j 、 B'_j を得る。このとき j, k はそれぞれ $j = (i-1) \cdot 2^m + 1$ 、 $k = j + 2^{m-1}$ と表せる。 N 段目の回収が終わった段階で、ルート・プロセッサからは、L空間に A_1 と B_1 を格納した共有メモリが見える。最後に A_1 、 B_1 に対して立体の再構成を行うことにより、L空間上に立体 C を得る。

4 シミュレーションによる性能評価

4.1 シミュレーションの方法

ここでは、並列プロセッサ・システムの動作を単一プロセッサ・システム上でのソフトウェア・シミュレーションで近似した。本システムの各モジュール内のプロセッサには、「汎用プロセッサおよび本システム特有の付加演算器」という構成のものを仮定した。また、立体データを格納する各共有メモリは、単一プロセッサ・システムのメモリ上にそれぞれ連続領域の形で用意した。本並列プロセッサ・システムは、

- プロセッサ間通信は動作モード遷移時のみに限られ、また、全ての動作モードにおいてプロセッサ間のメモリ・アクセス競合は皆無である。
- メモリ参照に関しては、バス・スイッチはトランスバレントである。

という特徴を有している。したがって、仮に、シミュレーションに用いた単一プロセッサと同様のハードウェアを要素プロセッサとして用いた場合、シミュレーション時の処理特性（たとえば、キャッシュのヒット率など）が、そのまま並列プロセッサ下でも現れると予想される。

集合演算処理プログラムは、EWS:HP9000/350CH (CPU:MC68020/25MHz、FPU:MC68881/20MHz、キャッシュ:32Kバイト、主記憶:8Mバイト) 上でC言語により作成 (180 関数、計 9,100 行) し、各部の処理時間を OS のプロファイリング機能を利用して測定した。これにより、サンプリング間隔 20msec.

でCPU消費時間を測定することができる。なお、この時間には、キャッシュ・ミスによるオーバーヘッドも含まれる。

4.2 並列処理性能の評価

負荷分散戦略としては、次の二つをそれぞれ用い、その効果を測定した。すなわち、

- i) 各プロセッサの担当構成面の数を均等にする。
- ii) 各プロセッサの担当構成面に含まれる稜線の数の合計を均等にする。

なお、集合演算の種類による処理時間の差はほとんどないことから、以下では、和演算の場合についてのみ示す。

まず、6角ナットと円柱との和演算について、プロセッサ数をそれぞれ1、2、4、8とした場合の各段階の処理時間を測定した(図8、表1、表2)。負荷分散の対象は6角ナットである。なお、各処理段階の処理時間は、最も時間のかかったプロセッサの処理時間である。第2節での考察通り、交差計算および交線生成に要する処理時間が全処理時間の大部分を占めており、立体のブロードキャストおよび計算結果の回収に要する処理時間はわずかであることがわかる。

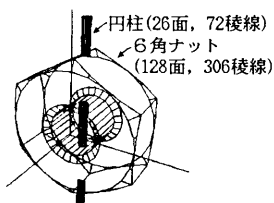


図8: 6角ナットと円柱の和演算

負荷分散戦略i)をとった場合(表1)、プロセッサ数の増加につれてある程度の並列処理効果が得られてはいるものの、十分とは言えない。これに対してii)の方法(表2)では、かなりの効果が得られている。ただ、プロセッサ数8で並列処理効果が頭打ちとなっているのは、2.3で述べた「プロセッサ間で重複する交差計算」が増加するためである。したがって、負荷分散対象となる立体の構成面数が増加するほど、この頭打ち傾向は解消していくものと考えられる。

そこで次に、球と円柱との和演算について測定した(図9、表3)。この場合には、図8の場合に比べ

表1: 図8の場合の処理時間(構成面数による負荷分散)

処理段階	プロセッサ数			
	1	2	4	8
ブロードキャスト	—	0.02	0.04	0.06
交差計算、交線生成	107.42	58.42	40.56	32.58
回収	—	0.36	0.56	0.78
再構成	1.26	1.28	1.28	1.28
合計 [秒]	108.68	60.08	42.44	34.70
(処理速度比)	(1)	(1.81)	(2.56)	(3.13)

表2: 図8の場合の処理時間(稜線数による負荷分散)

処理段階	プロセッサ数			
	1	2	4	8
ブロードキャスト	—	0.02	0.04	0.06
交差計算、交線生成	107.42	49.44	27.02	16.62
回収	—	0.36	0.60	0.86
再構成	1.26	1.28	1.28	1.28
合計 [秒]	108.68	51.10	28.94	18.82
(処理速度比)	(1)	(2.13)	(3.76)	(5.77)

て大きな並列処理効果が得られていることがわかる。球は、448個の等脚台形と64個の二等辺三角形により近似しており、構成面数が6角ナットの4倍あることから、プロセッサ間の重複計算が相対的に減少するためと考えられる。また、各構成面の稜線数が4または3と平均しているため、負荷分散戦略i)、ii)による相違はほとんど現れなかった。したがって、わずかに優れていたii)による結果だけを表3に示した。

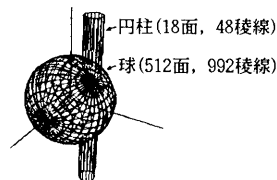


図9: 球と円柱の和演算

最後に、機械部品と円柱との和演算について測定した(図10、表4)。この機械部品は6角ナットに比べて構成面の空間的な分布範囲が広く、円柱との交差部分が非常に偏る性質がある。実際、6角ナットの

表 3: 図 9 の場合の処理時間 (稜線数による負荷分散)

処理段階	プロセッサ数			
	1	2	4	8
ブロードキャスト	—	0.04	0.08	0.12
交差計算、交線生成	199.06	100.58	51.98	27.14
回収	—	0.36	0.60	0.80
再構成	2.12	2.14	2.14	2.14
合計 [秒]	201.18	103.12	54.80	30.20
(処理速度比)	(1)	(1.95)	(3.67)	(6.66)

場合に比べて並列処理効果が上がっていないことがわかる。交差部分が局所的であり、また担当範囲が局所的であるほど、交差部分を担当するプロセッサが集中して影響を受け、全体として並列処理効果が向上しないと言える。

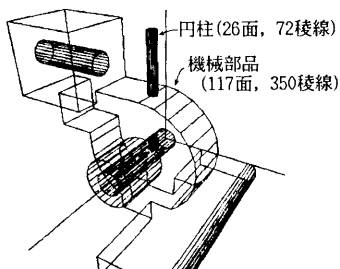


図 10: 機械部品と円柱の和演算

表 4: 図 10 の場合の処理時間 (稜線数による負荷分散)

処理段階	プロセッサ数			
	1	2	4	8
ブロードキャスト	—	0.02	0.04	0.06
交差計算、交線生成	86.98	42.96	27.16	16.94
回収	—	0.24	0.42	0.64
再構成	1.00	1.02	1.02	1.02
合計 [秒]	87.98	44.24	28.64	18.66
(処理速度比)	(1)	(1.99)	(3.07)	(4.71)

ところで、全ての和演算において、交差計算および交線生成時間が、プロセッサ数 1 の場合よりもプロセッサ数 2 の場合の計算合計時間のほうが少なくなる現象が見られた。特に、表 2 の結果、すなわち戦略 ii) による 6 角ナットと円柱の和演算では、プロセッサ数 2 による並列処理効果が 2 倍以上となっている。

これは、負荷分散にともなう立体データ参照の局所化と、これによるキャッシュ・メモリのヒット率向上が原因である。

5 おわりに

本報告では、立体の集合演算を高速化する手法について考察を行い、具体的な並列プロセッサ・システムの構成方式を提案した。そして、シミュレーション結果から、その並列処理方式の有効性を示した。

まず、集合演算処理の中で、交差計算および交線生成処理に要する処理量が非常に多いことに注目し、この処理機能部分に集中的に並列処理による高速化が適用できるようなハードウェア構成方式を提案した。また、本システム特有のプロセッサ・メモリ間結合方式により、並列処理のオーバーヘッドを極めて小さくできることを示した。

負荷分散については、各プロセッサ間で構成面数を均等にする方法と、構成面に含まれる稜線数を均等にする方法の二つについて検討し、後者の方が比較的良好な結果を生むことを示した。

謝辞

在学中、本研究に多大な貢献をして頂いた中島康彦氏（現在富士通）に感謝致します。

参考文献

- [1] 沖野 教郎. 形状モデリング. 精密機械, 47(11):20-28, 1981.
- [2] 中島 康彦, 新実 治男, 富田 眞治, 萩原 宏. 実時間 3 次元動画システムにおける動画記述. 情報処理学会第 33 回全国大会講演論文集, (2Q-2):2073-2074, September 1986.
- [3] 杉原 厚吉, 伊理 正夫. 計算誤差による暴走の心配のないソリッドモデラの提案. 情報処理, 28(9):962-974, September 1987.
- [4] 伊藤 宏一, 水町 肇, 稲垣 充廣. GKS-3D のビューイングパイプライン縮退による高速化. 情報処理学会第 33 回全国大会講演論文集, (3Q-8):2099-2100, 1986.
- [5] 川島 泰正, 太田 吉美, 徳増 真司. CAD 対話インタフェースのためのソリッドモデル隠面処理法. 情報処理, 28(2):147-155, 1987.
- [6] G. Kedem and J.L. Ellis. The ray casting machine. Proc. IEEE Int'l Conf. Comput. Design '84 (VLSI in Comput.), 533-558, 1984.