修士論文

体感型シミュレーションシステム Scubeの構築と可視化性能の評価

指導教員　　富田眞治

京都大学大学院情報学研究科
修士課程通信情報システム専攻

吉村 知晋

平成18年2月3日

体感型シミュレーションシステム Scube の構築と可視化性能の評価

吉村 知晋

内容梗概

近年，汎用計算機の性能向上は著しく，これらを並列に連携させた PC クラスタの構築が盛んである．PC クラスタの用途として，大規模かつ高精度な数値シミュレーションが挙げられる．そうした中，次世代のシミュレーション技術として，従来の実験の代替手段となり得る，実時間数値シミュレーションとシミュレーション結果の可視化処理の両方を実現するシステムの構築が望まれている．

本研究では，人間の視覚・触覚を通した実時間インタラクションを考慮にいれた超高速体感型シミュレーションシステムのプロトタイプとして，Sensable Simulation System Scube の実時間可視化クラスタの開発を行っている．Scube は 64 ノード構成の PC クラスタで，各ノードに可視化処理のための汎用グラフィクスプロセッサを搭載する．Scube のノード間ネットワークは，比較的小規模なギガビットイーサーネット・スイッチの組合せで高いコストパフォーマスを実現する汎用ネットワーク構成に特徴がある．

本論文では，Scube を構築するにあたり，中規模コモディティクラスタ向け相互結合網 Three Quads を提案した．Three Quads は，多様な網の埋め込みが可能であり，ランダムな通信要求が発生した場合など，柔軟なネットワークリソースの提供が可能であることを示した．3 次元の隣接間通信と共に，数値計算で多用される行列の転置処理のための全対全通信，並列画像合成処理における 3 次元 Reduction 型の通信等に対して優れた特性をもつ相互結合網であることを示した．

また，並列可視化という観点では，各ノードでのレンダリング処理と各ノードで生成された部分画像の合成処理の両方の高速化が必須である．Scube では各ノードに GPU を搭載することでレンダリング処理自体の高速化が実現されている．また，本論文においては，部分画像の重複関係を考慮した並列画像合成アルゴリズムの提案を行い，評価を行った．その結果，従来の合成手法と比較して，およそ 2 倍の描画速度の向上が期待できることを示した．

Development of Sensable Simulation System Scube and Evaluations for its Visualization Performance

Tomoyuki Yoshimura

Abstract

In recent years, there is a remarkable improvement in PC performance and the construction of PC cluster cooperating PCs in parallel become popular. One of the application of PC cluster is a large-scale numerical simulation. Under this circumstance, the construction of a system, which makes it possible to execute both a real-time numerical simulation and the visualization of the simulation data is desired as simulation technology for the next generation, which may serve as an alternative means of the conventional experiment.

We have been developing the Visualization Subsystem for the Sensable Simulation System(Scube). Scube is a 64-nodes PC-based cluster system in which a commodity GPU as the visualization accelerator is configured with each node. There is no dedicated special purpose network for the visualization, however, the high cost-performance inter-connection network is originally designed for Scube. All the hardware components for this network is essentially the small-scale and commodity hardware designed for Giga-bit Ethernet.

In this paper, we have proposed an inter-connection network Three Quads for Medium Scale Commodity Cluster such as Scube. Thee Quads can embed various initer-connections and distribute the adaptable network resource in the random communication request and so on. This inter-connection has a superior characteristics in various communication patterns such as in the communication between neighboring nodes in 3-D, between all nodes for the transpose process of matrix frequently executed in numerical caluculation, and so on.

This paper have also designed the parallel image composition algorithm which utilizes the degree of overlapping between sub-images, in order to reduce the amount of unnecessary communication and computation, and evaluated its algorithm. The result shows our algorithm's speedup to the ordinary composition algorithms reaches about 2.

体感型シミュレーションシステム Scubeの構築と可視化性能の評価

目次

第1章 はじめに 1

第2章 背景 3

2.1 実時間インタラクティブシミュレーション 3

2.2 PCクラスタ環境の設計 4

2.2.1 Beowulf型クラスタ 4

2.2.2 相互結合網 5

2.2.3 SAN 7

2.3 並列ボリュームレンダリング 8

2.3.1 ボリュームレンダリング 8

2.3.2 汎用グラフィクスカードを用いたボリュームレンダリング 10

2.3.3 並列化 11

2.3.4 関連研究 13

第3章 インタラクティブシミュレーションシステム構築に向けての検討 17

3.1 スイッチ多段結合の通信遅延 17

3.2 並列ボリュームレンダリングにおけるボトルネック 18

3.2.1 実験環境 19

3.2.2 実験結果 21

3.3 本章のまとめ 23

第4章 Scubeの構築 24

4.1 Scubeの概要 24

4.2 ノード配置及びスイッチの物理的配置 25

4.3 相互結合網 Three Quadの提案 25

4.3.1 トポロジ 27

4.3.2 ノード間通信 28

4.3.3 耐故障性 (fault-tolerance) 29

4.3.4 スケーラビリティ 29

4.3.5 埋め込み性 (Embedability) 30
4.3.6 性能比較 32
4.3.7 転置行列演算への適用 33
4.4 本章のまとめ 35

第 5 章 主軸優先木構造合成 37
5.1 提案合成アルゴリズム 37
5.2 理論的考察 39
5.3 実装評価 42
5.3.1 実験環境 43
5.3.2 主軸優先合成の効果 43
5.4 高速化に向けた検討 45
5.5 本章のまとめ 45

第 6 章 おわりに 47

謝辞 49

参考文献 50

第1章　はじめに

近年，汎用 PC の性能向上はめざましいものがあり，かつてはスーパーコンピュータを必要としたような演算も安価な PC による高速処理が可能になってきている．こうした汎用 PC を複数台ネットワークで結ぶことにより，大きな計算能力を得ようという PC クラスタの構築が盛んである．PC クラスタの用途として，代表的なものにシミュレーションがある．従来のスーパコンピュータは，演算処理に関しては非常に高速で実行可能であるが，シミュレーション結果のインタラクティブな可視化には適していない．シミュレーション結果を，人間に理解しやすいよう画像にして提示する可視化技術は，数値シミュレーション結果の意味を理解し，また次のシミュレーションを方向づけるための一助となる役割を持つという点で，非常に重要である [1, 2, 3, 4]．このような用途の例として，医療画像のような大規模なボリュームデータや実空間モデルに基づく大規模科学技術演算の実時間可視化が挙げられる．

我々は人間の視覚・触覚を通した実時間インタラクションを考慮にいれた，超高速体感型シミュレーションシステム [5] の研究を行っている．そのプロトタイプとして Sensable Simulation System(以下 Scube) の実時間可視化クラスタ [6] を開発している．本システムでは，オペレータによるシミュレーション対象へのインタラクティブな操作に対応して，実時間でシミュレーションを行うとともに，即時にその結果を可視化などの手段により提示することが求められている．Scube の提案は，システム全体をコモディティ部品のみで構成し，複数系統設けたネットワークの冗長性を利用した通信のスケジューリングにより，シミュレーションと可視化処理の同時実行を目指すとともに，シミュレーションに必要となる通信の高速化も目指している．

このような実時間インタラクティブシミュレーション環境の実現にむけて，PC クラスタの構築と可視化処理に対してどれだけの投資が可能かの検討が必要である．Scube では，これら実現のため，比較的小規模なギガビットイーサーネット・スイッチ (以下 GbE SW) の組合せで構成する，中規模コモディティクラスタ向け相互結合網 Three Quads を開発した [6]．また，高速な並列画像処理は，処理対象に応じて処理内容を柔軟に変更でき，処理の高機能化も容易であるという理由からソフトウェアによる実現を図る．この実現に向けて，Scube 上への汎用グラフィクスカードを用いた並列ボリュームレンダリングの実装におい

て，Scube と親和性の高い合成アルゴリズムの提案を行った．

以下，第 2 章において，実時間インタラクティブシミュレーション環境の構想，構成方式，並びに，本システムに実装する汎用グラフィクスカードを用いた並列ボリュームレンダリングについて述べる．第 3 章において，インタラクティブシミュレーションシステム構築に向けての検討課題について，予備評価を行う．第 4 章では，Scube を構築するにあたり，提案する相互結合網 Three Quads の説明と定性的な評価を行う．第 5 章で通信時間を削減するための部分画像の重複関係を考慮した合成アルゴリズムを提案し，Scube 上でアルゴリズムの評価を行う．最後に第 6 章において結論を述べる．

第2章　背景

本項では，まず実時間インタラクティブシミュレーションについて説明を行う．次に，シミュレーション環境の設計指針について述べる．

最後に，汎用グラフィクスカードを用いた並列ボリュームレンダリングについて述べ，合成アルゴリズムの関連研究およびその問題を指摘する．

2.1　実時間インタラクティブシミュレーション

実時間インタラクティブシミュレーションとは，オペレータによるシミュレーション対象へのインタラクティブな操作に対応して実時間でシミュレーションを行うとともに，即刻その結果を視覚その他の手段により提示することのできる仮想実験型/仮想体験型のシミュレーションのことである．

通常の数値シミュレーションでは，初期状態と幾つかのパラメータをシミュレーションを開始する前に設定して，大型計算機等で計算を行い，後にシミュレーションの最終結果の可視化を行うというものが一般的であり，そのパラメータを計算の途中で変更することは極めて稀である．しかし，医療分野で必要とされている手術シミュレーションや触診シミュレーションといった対話性の要求されるシミュレーションでは，計算途中にオペレータの判断によって，パラメータを変更することがしばしば起こり，それに伴ってそのような時々刻々と変わる状況を実時間でシミュレーションしていかなければならない (インタラクティブ性)．このような対話性の要求されるシミュレーション環境では高精度なシミュレーションはもちろんのこと，従来の数値シミュレーションではあまり重視されていなかったリアルタイムでの応答が必要とされる．

我々は，計算負荷の高いシミュレーションと多次元データの可視化までを行うPCクラスタベースの計算サーバと，オペレータの操作や結果の呈示を行うユーザインタフェース端末としてのクライアントを用いたクライアントサーバモデルの環境を構築する (図2.1)．これにより，計算サーバを煩雑な入出力処理から解放するとともに，柔軟なユーザインタフェース環境の実現を目指す．シミュレーション結果のデータをどこで可視化し，どのようにしてオペレータに提示するのが望ましいかというのは扱う問題，構成するシステムの仕様によって様々である．シミュレーション系から可視化系へのデータの受け渡しが頻繁に起こる場合，各計算機ごとに可視化系を設け，画像のような中間データを生

成し，これをクライアントに集めて最終結果を可視化するという方式 (図 2.2) が採用される．

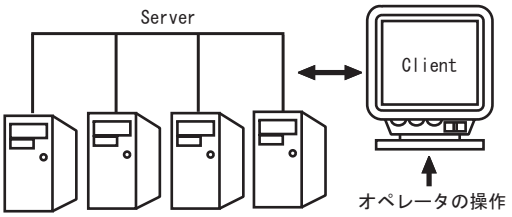


図 2.1: クライアントサーバモデル

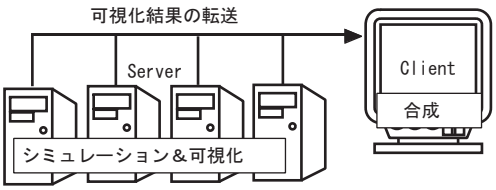


図 2.2: 可視化サーバ

我々と同様に，数値シミュレーションと実時間可視化の同時処理を目指したシステムとして，緒方らは VGCluster[1, 7] を提案している．VGCluster では可視化処理を高速化する画像合成処理専用のハードウェアを設けることで，画像合成処理のための通信が数値シミュレーションのための通信を妨げない手法を提案している．これに対して Scube の提案は，システム全体をコモディティ部品のみで構成し，複数系統設けたネットワークの冗長性を利用した通信のスケジューリングにより，シミュレーションと可視化処理の同時実行を目指すとともに，シミュレーション自体で必要となる通信の高速化も目指している．

2.2 PC クラスタ環境の設計

PC クラスタによるシミュレーション環境を設計するにあたり，クライアント·サーバ間，クライアント間での通信が頻繁に生じる場合，ネットワーク設計がシステム性能を決定付ける．本項では PC クラスタの設計技術について述べる．

2.2.1 Beowulf 型クラスタ

クラスタとは，英語でブドウの房または星団のような集団を意味する．クラスタをコンピュータシステムに当てはめた PC クラスタとは，一般的に使用されている PC をネットワーク結合して構築した並列計算機の 1 種である．Pfister によるとクラスタは単一で稼働するコンピュータの集まりで，一つの計算機資源として使用可能な並列もしくは分散システムであると定義されている．

1990 年前後に，TCP/IP ベースのネットワークで接続された複数のコンピュータの集合体を，仮想的に 1 台の大きな並列計算機として見せるためのソフトウエアである PVM (Parallel Virtual Machine) が開発された．これにより「PC ク

ラスタ」と呼ぶべき物理的な計算機群が登場した．1995 年前後には，イーサネットスイッチや 100Mbit Ethernet などの高速ネットワーク技術が一般にも普及し，比較的安価に高性能なネットワークの構築が可能となった．さらには計算機クラスタを指向した専用の高速ネットワークが登場し，一方では PC の高性能化と低価格化により，コストパフォーマンスに優れた PC クラスタ (Beowulf 型クラスタ [8]) の実現が可能となった．また，複数のベンダーから提供される部品を最適に組み合わせることが可能であり，システム規模の拡張性にも優れ，技術動向に柔軟な対応が可能であることなどが挙げられる．

しかし，次の 2 点に問題があり，その利用範囲は限られたものである．第 1 に，専用の高速スイッチで利用されている Myrinet[9] 等を使用すると，システム構築が高価になるため，基本的に使用しない．一般的に，通信性能 100Mbps や 1Gbps 程度のネットワークを使い，インターネットで使用されている TCP/IP ネットワークプロトコルを使用しているために，専用並列コンピュータに比べて通信性能が低下する．第 2 に，逐次的に実行していくプログラムでは，通信量を考慮する必要はないが，並列処理を行う場合，PC 間の通信が頻繁に生じるプログラムを実行すると，処理速度が著しく低下する．そのため，通信ボトルネックを解消するために，アルゴリズムの最適化を行う必要がある．

2.2.2 相互結合網

相互結合網は並列計算機を構成する複数のプロセッサやメモリを相互に結合し，その間における交信路を提供する．目標は通信遅延 (レイテンシ) を最小にし，スループット (処理能力) を最大にすることである．並列計算機の相互結合網には，高い通信性能とともに，実装性，拡張性および耐故障性などに優れていることが求められる．

既存の相互結合網は主として専用並列計算機のためのものである．すなわち，各相互結合網にはそれぞれの長所と短所があり，その専用並列計算機の利用目的に応じて相互結合網を採用する，もしくはクラスタ用に設計された汎用のネットワークハードウエアを使用する．

高い通信性能とともに，拡張性，耐故障性，他の網の埋め込み性に優れたハイパーキューブ網系の代表的な相互結合網を以下に示す．

3 次元トーラス網

IBM 社の BlueGene/L におけるノード間通信に採用されている結合網である．BlueGene/L では 8×8×8 ノード数の 3 次元トーラス網が再帰的，階層的に構築

され，スケーラビリティを保証している．図 2.3 は，4×4×4 ノードの例である隣接ノード間のデータ転送だけを行うようなローカリティの高い計算に適している．遠方のノード間とのデータ転送を頻繁に行う場合，中継ノードが転送処理に時間を要し，ボトルネックとなる．隣接ノード間通信が主となるアルゴリズムの開発が必要となる．

ハイパーキューブ網

ノードを n 桁の 2 進数で表現した場合，ノード番号 a はそれとハミング距離が 1 だけ異なるノードに直接結合している．図 2.4 は，16 ノードの例である．ノード 0101 は 1101,0001,0111,0100 の 4 つのノードとのリンクを持つ．次数 n は 2 進数で表した時の桁数に等しいので，全体のノード番号を N とすると，$n = \log_2 N$ となる．ノード $a(a_n, a_{n-1}, \ldots, a_1)$ から $b(b_n, b_{n-1}, \cdots, b_1)$ へのメッセージ転送のルーティングは次のようになる．最下位ビットから調べ，最初に $a_i \oplus b_i = 1$ となるノード $(a_n, a_{n-1}, \cdots, a_{i+1}, b_i, b_{i-1}, \cdots, b_1)$ にデータを送信する．最上位ビットまでこの過程を繰り返すと，ノード a と b の通信が可能である．

汎用並列計算処理向きの相互結合として注目されたが，次数が他の網と比較して大きく，しかも物理的に距離の長いリンクを必要とするため，実装が煩雑である [10].

ベース-m n-キューブ

様々なところから同時発生的に提案されたが，1987 年に東芝からの報告が最初であり，後に並列計算機 Prodigy[11] で用いられた．各ノード m 進 n 桁で表現した場合，1 桁値が異なるノード m 台をクロスバスイッチで結合する．図 2.5 は，ベース-m 3-キューブの例である．ノード番号は 4 進 3 桁で表現され，最下位桁のみ異なるノード，中央桁のみ異なるノード，最上位桁のみ異なるノードがそれぞれクロスバ網で結合されいる．Scube で想定されるアプリケーションの基本的な並列化手法は，実空間離散化に基づく並列処理である．絶対的な総計算量が増加する傾向にある問題に対して，多次元メッシュ化されたノードに直接マッピングすることにより，実空間での相互作用は隣接通信に帰着され，ネットワークへの負荷が大幅に削減される．Massively Parallel Processor のスタイルを踏襲する計算手法では，広範囲な通信を適度なバンド幅で支援する一般的なクラスタ向け SAN は適さず，ベース-m 3-キューブのような 3 次元のハイパークロスバ結合のようなネットワークが望ましい [12]．単一の NIC ではなく，3 次元方向に対応した 3 つの NIC によって外部スイッチに接続される．理論上，こ

れは，ノード当りに必要な総ネットワークバンド幅を 3 つの NIC に分散させることに相当し，NIC を結合するバスと NIC そのものに要求されるバンド幅を低減するという効果がある．

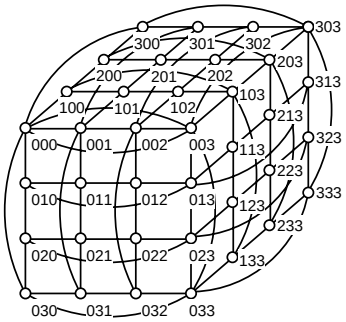


図 2.3: 3 次元トーラス網

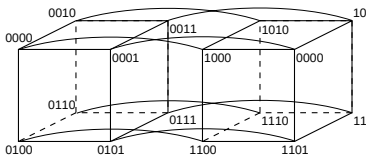


図 2.4: ハイパーキューブ網

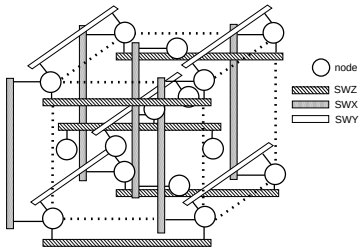


図 2.5: ベース-m 3-キューブ

2.2.3 SAN

高性能 PC クラスタにおいて PC 間を接続するシステムエリアネットワーク (SAN) は，システムの性能に大きな影響を与える [9]．SAN はダイレクトメモリ通信を高速に行うために，高バンド幅，低レイテンシであることが求められる．SAN では，複数のスイッチ群と大容量の point-to-point リンクを用いて構成されるため，パケットは複数のスイッチを経由して目的地に到達することになる．そのため，スイッチ群のトポロジが高バンド幅及び低レイテンシを実現するための 1 つの鍵となる．現在，多くの SAN は，拡張性，耐故障性を重視して，様々なトポロジを取ることができる．例えば，Myricom 社は Myrinet のトポロジとして，完全結合に近い Myrinet-Clos 網 (図 2.6) を推奨しているが，他のトポロジを取ることも可能である [9]．そのため，SAN の性能を向上させるためには，適切なトポロジを採用することが重要であるとされている．

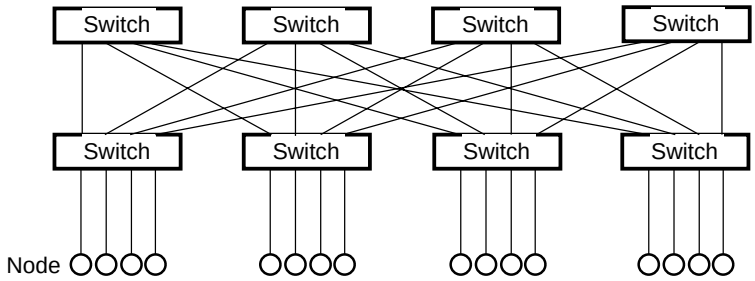


図 2.6: Myrinet-Clos 網

2.3 並列ボリュームレンダリング

仮想現実感を提供する実時間インタラクティブシミュレーション環境が扱うシミュレーションは主に 3 次元現象である．したがって，ここでは 3 次元現象を可視化する方法についてのみ述べる．

2.3.1 ボリュームレンダリング

3 次元現象の数値データ (ボリュームデータ) を可視化する方法の一つとして，ボリュームレンダリングがある．ボリュームレンダリングとは，与えられたボリュームデータを別形状のデータに変換することなく，そのままの形状で数値から透明度を伴う色に変換を行い，半透明状の画像を得る手法である．データを等値面に変換して可視化を行うサーフェイスレンダリングは，物体の表面だけを可視化対象としているのに対し，ボリュームレンダリングは物体の中が詰まったまま可視化処理を行う．そのため，具体的な形をとらないガス状に分布するシミュレーション結果の可視化や，物体の内側を観察することが必要となる医療シミュレーション結果の可視化などにおいて，有効な可視化手法といえる [13].

ボリュームデータは空間の最小単位であるボクセルの集合で表されている．ボクセルは 3 次元座標上の点からサンプリングされたもので，もとの対象物について，その地点での 1 つもしくは複数の観測値や計算値（例えば，傾向強度，密度，電荷，湿度など）を持たせることにより，空間全体をデータとして保持している．ボリュームレンダリングでは，格子点以外の地点のデータは，その地点近傍の格子点から線形補間を用いて求める．これにより，空間内の全ての地点におけるデータを得ることが出来る．数値シミュレーションにより得られた 3 次元空間上の数値データを，RGB 値と不透明度に対応付けて可視化することで，3 次元空間内部のデータの分布状況を可視化する．具体的には描画面の各画素から視線方向に沿ってボクセル値の持つ色情報を積分していく．これを離散化すると，スクリーン上の各ピクセルごとに発生する視線に沿って，視線と交差するボクセル値のサンプリングを視線上のボクセルがなくなるまで繰り返し，ピクセル値を求めることになる (図 2.7)．この方法は視点から近い順にサンプリングする方法 (front to back) と，視点から遠い順にサンプリングする方法 (back to front) に分けられる．back to front の場合，ピクセル値 P はボクセルの値を視点に対して遠い順から，v_0，v_1，$\cdots$，v_n とし，RGB 値の各色情報 c_k

と不透明度 α_k がボクセル値 v_k の関数で表されるとすると，

$$P = \sum_{i=0}^{n} \alpha(v_i)c(v_i) \prod_{j=0}^{i-1}(1-\alpha(v_j)) \tag{2.1}$$

と表される．このピクセル値計算式は累積値 C_k と累積不透明度 A_k を用いて式 (2.2),(2.3) のような漸化式に変形される．

$$C_k = \alpha(v_i)c(v_k) + (1-\alpha(v_i))C_{k-1} \tag{2.2}$$

$$A_k = \alpha(v_i)A_{k-1} \tag{2.3}$$

ここで $P = C_n$ である．このピクセル値計算は，演算区間をいくつかの部分区間に分割し，それぞれの区間に対する計算結果に対して，再度ピクセル値計算を行うことが可能であるという性質がある．

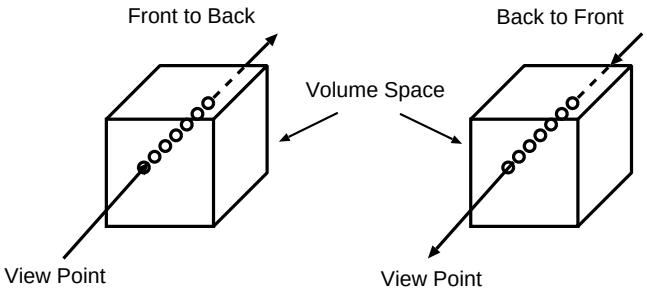


図 2.7: ボリュームレンダリング

ボリュームレンダリングは膨大な計算時間と記憶容量を必要とし，大型計算機や特殊ハードウェアを用いる場合に利用が限られており，リアルタイムに可視化することは一般に困難であった．この問題に対して，しかし，近年，専用ハードウェアや汎用 GPU の採用によるレンダリング時間の大幅な短縮が報告されている．しかし，大規模データの実時間可視化を実現するにあたり，演算性能ならびにメモリ容量の不足を補うための並列化が必要である．このような並列可視化環境においては，レンダリング処理と部分画像の並列合成処理の両方の並列処理に最適化することが必須である．

並列ボリュームレンダリングにおける並列画像合成処理の高速化を目指した研究としては，Binary-Swap-Comoposition[14](以下 BSC)，SLIC[15]，VG クラスタ [1, 16]，Sepia[17]，VisA[18] などの研究がある．BSC および SLIC は PC クラスタ上でのソフトウェアベースの画像合成アルゴリズムであり，VG クラス

タ，Sepia，VisA は専用ハードウェアを用いた実現である．本論文では，ソフトウェアベースの画像合成アルゴリズムを提案し，Scube 上に汎用グラフィクスカードを用いた並列ボリュームレンダリングシステムの実装を行う．

2.3.2 汎用グラフィクスカードを用いたボリュームレンダリング

汎用グラフィクスカードでサポートされているテクスチャマッピングと α ブレンディング機能を用い，Scube の各ノードはボリュームレンダリングを行う．

テクスチャマッピングとは，物体の表面の質感を表現するために貼り付けるテクスチャと呼ばれる画像を物体表面に貼り付ける方法である．ポリゴンの各頂点がテクスチャ画像上のどの点に対応しているのかをあらかじめ関数として持っており，ポリゴンを描画するときに，それを元にポリゴン上の 1 点 1 点が，テクスチャ上のどの位置を参照しているのかを求めて，その位置に対応している色でポリゴンにマッピングしていく．

一方，α ブレンディングとは，2 枚の画像を持つ RGB 値を α 値の示す比率で線形内挿し，合成画像の RGB 値を算出する方法である．前後関係にある 2 枚の画像の RGB 値に乗じる α 値を混合係数として設定する．色情報 (source) が，バッファに保存されている色情報 (destination) と α ブレンドされて処理されるとする．source と destination の RGB 値を C_s,C_d，α 値を A_s,A_d とすると，α ブレンド後の RGB 値 C，α 値 A はそれぞれ式 (2.4),(2.5) のように表される．

$$C = A_s C_s + (1 - A_s) C_d \tag{2.4}$$

$$A = A_s + (1 - A_s) A_d \tag{2.5}$$

式 (2.3) はボリュームのサンプリングを視線方向に従って一定のサンプリングで行い，描画面に遠い方から順に RGB 値を α ブレンディングすることでボリュームレンダリングを行うことが可能であることを示している．ボリュームをある軸に対して垂直なスライスの重ね合わせで表現する．そのスライスをテクスチャとしてポリゴンにマッピングし，それらを視点から遠い順に順次 α ブレンディングすることでボリュームレンダリングを行う (図 2.8)．この手法を用いることで，汎用グラフィクスカードの機能を利用した高速処理が可能である．グラフィクスハードウェアが 3 次元テクスチャ(図 2.9) をサポートしている場合，視線に対して垂直な面を用意し，ボリュームデータを 3 次元テクスチャとして扱う方法が可能である．3 次元テクスチャが利用できない場合は，ボリューム

データを各座標軸に対して垂直なスライスとして3つ用意し，視線方向とスライスの法線のなす角が一番小さい軸のスライスに対して，2次元テクスチャとしてマップする方法を用いる．

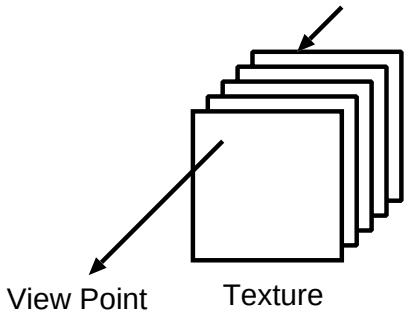


図 2.8: テクスチャベース

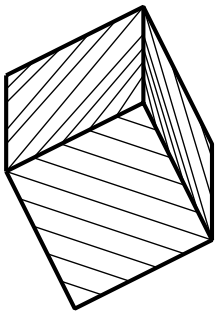


図 2.9: 3次元テクスチャ

2.3.3 並列化

ボリュームレンダリングは光線に沿ったボリューム空間内の標本点の透明度の累積をスクリーン上のピクセル毎に行うものである．したがって，その処理はピクセル毎に独立であり，その計算は空間内を通過する光線の積分計算である．よって，空間をいくつかに分割し，それぞれの区間に対する演算結果を足し合わせることで最終結果を得ることができる．つまり，シミュレーションサーバの各ノードが，そのノード内で生成されたデータに対する部分3次元空間 (以下サブボリュームと呼ぶ) に対してピクセル値計算を行い，そこで得られた画像と画素毎の透明度を視点からの距離の順番に従って，合成を行っていくという手法で並列処理が可能である．サーブノード数が8台の場合，ボリュームデータを x, y, z 全ての軸方向に2等分し，図 2.10 のように8つのデータおよびサブボリュームに分割する．

その分割したデータそれぞれをクラスタの各サーブノードが分担してテクスチャベースでボリュームレンダリングを行い，各サブボリュームごとのレンダリング結果（中間画像）を生成する．そして，フレームバッファ上でレンダリング結果が存在する部分だけを読み出し，それを一つの Master ノードに集める．データは最初に分配されるだけで回転等のパラメータの変更による再描画において，各サーブノードは現在保持しているサブボリュームに関してのみレンダリングを行う．各サブボリュームからそれに対応するオブジェクト座標で

の代表点をとり，サブボリュームのスクリーンからの距離を得る．Master ノードは，各サブノードで生成されたレンダリング結果の画像データを受信し，視点からの距離を考慮してスクリーン上の適切な位置に，式 (2.4) により α 値で合成 (α ブレンディング) することにより最終画像合成結果を得る．このような並列画像合成プロセスを sort-last 並列レンダリング [19] とも呼ぶ．ノード数が 8 台の場合の並列ボリュームレンダリングを図 2.11 に示す．

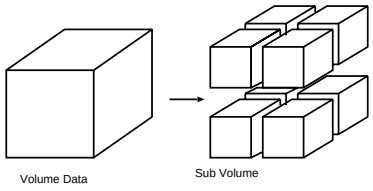


図 2.10: ボリューム分割

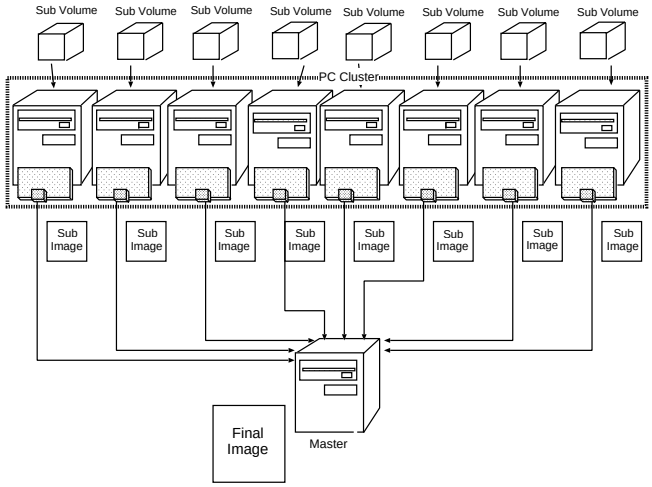


図 2.11: 並列ボリュームレンダリング

逐次的合成処理

各サブノードが，自身の担当するサブボリュームのレンダリングを終了し，フレームバッファ上のレンダリング結果を読み出した後，その画像データを一斉に Master ノードへ送信する手法である．Master ノードは，視点からの距離が最も遠いサブボリュームを担当しているノードから順に，中間画像データを逐次的にブロッキング受信する．ノード間合成にかかる時間は視点の位置に依らず，安定した速度が得られる．しかし，全てのサブノードが Master ノードへ同時送信を行う場合，送受信 MAC が CSMA/CD 方式であるため，Master ノードは複数のサブノードから同時に受信することは不可能である．そのため，サブノード群において，Master ノードへの送信待機を行うノードが存在し，送信待機中は合成処理に関する処理は行わない．Master ノードへの送信順序が後になるにつれ，演算リソースの空き時間が長くなる．

高速化手法の 1 つとして，中間画像を圧縮し，送信するデータ量を軽減する

手法としてランレングス符号化が挙げられる．ランレングス符号化は繰り返しの情報を圧縮することによって，データのサイズを減少する方式である．同じ色の連続している部分を破棄して，データを圧縮する．これを各ノードの保持する画像中の透明部分に関してのみ行う．一般に断層撮影などから得られるボリュームデータは全体の 70% から 95% が透明領域である．透明 $(A = 0)$ であるピクセルは RGB 値に関わらず可視化結果に影響はないので，RGB 値を Master ノードに送信する必要はない．そこで，送信する画像の中に含まれる連続する $A = 0$ となるピクセルの個数を RGB の領域を用いて表し，データ量を減らす (図 2.12)．これにより，通信データ量を減らすことができるので，画像データの転送時間を減らすことが可能となる．

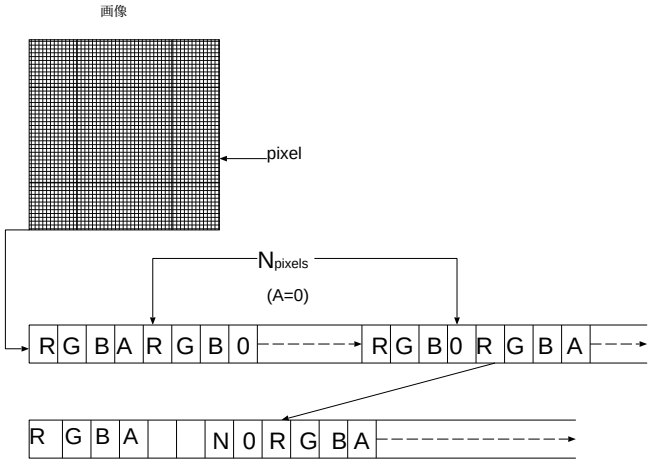


図 2.12: ランレングス符号化

2.3.4 関連研究

並列ボリュームレンダリングの合成プロセスにおいて，種々の高速化アルゴリズムが提案されている．並列画像合成では，ノード間の大量のデータ通信を必要とする．通信すべきデータ量はノード数と画像の大きさに比例する．インタラクティブな可視化シミュレーションにおけるシステム性能を考える場合，通信コストが重要な問題である．通信コストを軽減することに主眼が置かれた関連研究について以下に述べる．

Binary Swap Composition

BSC[14] は，オブジェクト分割法 (図 2.13) により，各ノードがサブボリュームの描画を行った後，投影面分割法 (図 2.14) により，通信量と演算量を抑え，

高い並列性を抽出可能な分割統治型の並列画像合成アルゴリズムである．

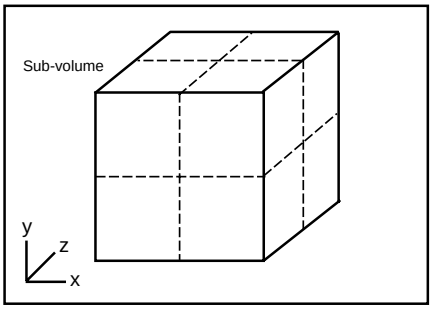


図 2.13: オブジェクト分割法

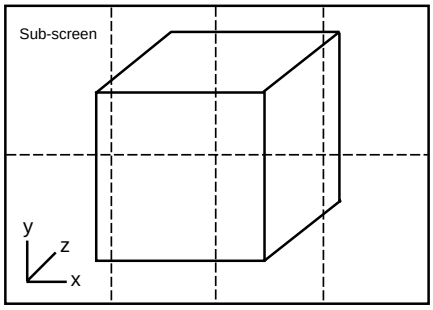


図 2.14: 投影分割法

N 台のプロセッサで最終合成を行う場合を考える．BSC は交換 (Swap)，合成 (Composite)，収集 (Gather) という 3 つのフェイズから構成される．ただし，図 2.15 は $N = 4$ の場合である．

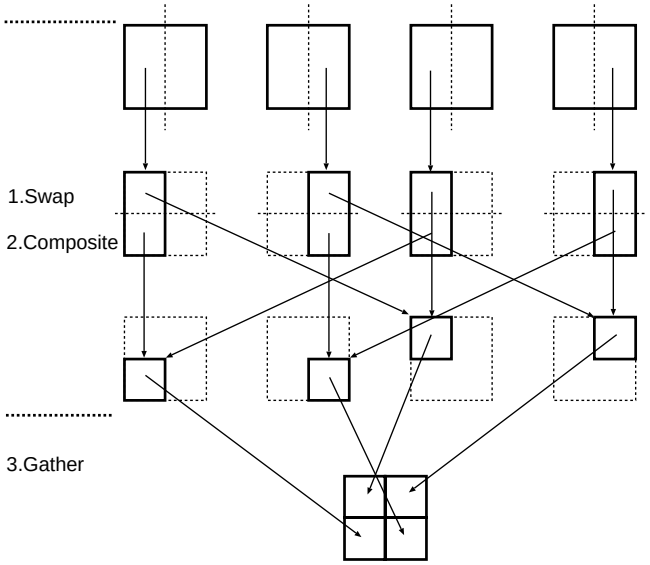


図 2.15: Binary Swap Composition Tree

1. Swap
   各プロセッサは，合成すべきスクリーンを 2 分割する．自身と同じ領域を持つプロセッサと，互いが異なる領域を担当するように，分割したスクリーンを交換する．この交換を $log_2 N$ 段繰り返す．
2. Composite

$log_2 N$ 段の交換後には，各プロセッサは自身が担当するスクリーン領域（全体の N 分の 1）の，全てのプロセッサによるサブスクリーンを持つことになる．即ち，最終合成を行うことが可能な条件を満たしている．したがって，各プロセッサが自身の担当領域を最終合成することにより，並列最終合成を行うことが出来る．

3. Gather
   並列最終合成が終わったあとでは，単純に領域を繋ぎ合わせるだけで最終画像を得ることが出来る．この繋ぎ合わせは，N 枚の最終合成画像を通常の木による合成を行うことで，1 台のプロセッサに集めることで完了する．

この手法は，通信量の削減と合成処理における負荷の均衡をとる点において優れている [16] 手法である．しかし，実用規模の並列度 (例えば 64 並列) をもった可視化システムにおいて，初期状態として各ノードがもつ部分画像間の重複が比較的少なく，かつ，当該部分画像のサイズ (1/並列数) が最終画像サイズに比べて小さい場合には必ずしも最適なアルゴリズムであるとは限らない．この原因を以下に述べる．並列ボリュームレンダリングは，ボリュームデータを N 等分したうちの 1 つを各ノードが担当する．このため，各ノードの描画領域はまとまっていて，スクリーンのどこにあたるかを予め求めることができる．また，その領域は全体の 4 分の 1 ほどである．つまり，何も描画されていない領域が全体の 4 分の 3 を占める．そのような条件下で BSC を行うと，全く画像が描画されていない領域が Swap に多く含まれることになる．そのため，BSC にランレングス符号化を導入し，通信データ量を削減し，合成処理負荷を軽減する手法 [20] も提案されている．

SLIC

SLIC[15] は，各ノードが生成した中間画像同士の重複関係を視線方向に基づいて解析し，合成の必要がない背景領域や他の中間画像と重なりのない領域を並列画像合成の対象から省くことで合成処理の演算量を削減する．合成処理の対象から省かれた領域に対応する中間画像は，必要に応じて最終画像表示ノードへ直接転送を行う．負荷の均等化に際しては，各ノードにおいてスキャンライン内での中間画像の重複回数と重複状態を求め，その重複状態が同じ範囲 (スパン) を負荷分散の単位として，スパン単位の合成処理を各ノードに静的に割当てる負荷分散方式を採用している optimized direct send といえる．重複状態を考慮することによる演算量削減の効果は大きいと考えられるが，SLIC で提案

されている負荷分散方式では，アルゴリズムの実装に際し，ノード間の通信パターンの不規則性が増加し，ノード間のネットワークに高いランダム通信性能が要求されることになる．Myrinet 等の PC クラスタベースのシステムが多く採用しているネットワーク構成では，１つのスイッチ内に閉じる場合は高いランダム通信性能が期待できるが，複数のスイッチに跨った場合，SLIC の負荷分散アルゴリズムの要求に応え得る十分なランダム通信性能を得るのは困難となる．

第3章　インタラクティブシミュレーションシステム構築に向けての検討

PC クラスタ上で大規模な数値シミュレーションおよび可視化処理を行うにあたり，以下の検討項目について，予備評価を行う．

3.1　スイッチ多段結合の通信遅延

中規模クラスタにおいて，スイッチを多段結合し，ネットワークを構成している場合が多い．システム全体をコモデティ部品の組み合わせにより構築し，比較的小規模なスイッチを多段結合することにより実現する．このような複数のスイッチを多段結合して構築したPC クラスタにおいて，全ノードが同時に 1 対 1 通信を一斉に行った場合 (図 3.1)，ネットワークの競合が発生する可能性がある．つまり，同一スイッチ内のノード同士が同時に一斉通信を行う場合，各送信要求に対して通信経路が保証される．一方，スイッチを多段結合して通信経路が確保されているノード間で複数の送信要求が発生した場合 (図 3.2)，ネットワーク上でのリソース競合が発生すると考えられる．そのため，このような場合，データ送信は逐次的に行われ，全ノードが送信完了する時間において，大きな通信遅延を招くと考えられる．

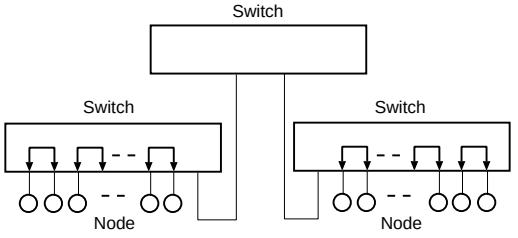


図 3.1: 同一スイッチ内のノード間通信

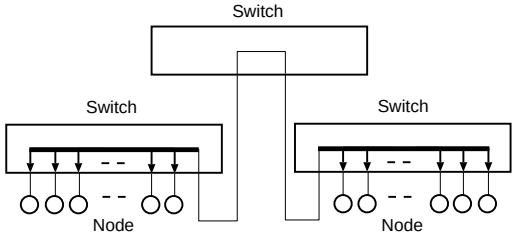


図 3.2: スイッチの多段結合によりリンクを有するノード間通信

そこで，同一スイッチ内の 16 台のノード同士が通信を行う場合，また 1 段のスイッチを跨いでリンクを持つ 16 台のノード間が通信を行う場合のそれぞれにおいて，同時に送信要求を発生させ，全ノードが受信を完了するまでの時間を測定した．

ノードおよびネットワーク仕様を表 3.1 に示す．ただし，送信データのサイ

ズは 86436Bytes とした．

表 3.1: ノードおよびネットワーク仕様

CPU	Pentium4 3.4GHz
バス	PCI-Express
NIC	Gigabit Ethernet
実測性能	およそ 700Mbps
通信ライブラリ	lam7.1.1/MPI

送受信完了時間の測定結果を表 3.2 に示す．但し，測定時間は数百回の通信を行った結果の平均値である．

表 3.2: 送受信完了時間の比較

	同一スイッチ内ノード間通信	スイッチを跨いだノード間通信
送受信完了時間 [ms]	1.12	11.54

サイズ 86436Bytes のデータの 2 ノード間通信時間は実測性能より，0.98[ms] である．同一スイッチ内のノード間同士で通信を行った場合の通信時間は 1.12[ms] であり，多少の遅延が確認できるが，スイッチ内のネットワークの競合が生じることなく，通信が行われたと考えられる．一方，1 段のスイッチを跨いだノード間で通信を行う場合の通信時間は 11.54[ms] であり，スイッチ間通信の際に，ネットワークの競合が生じ，Ethernet 上で CSMA/CD 方式によってスイッチ内でコリジョンの衝突を回避されるため，通信遅延が生じたと考えられる．理論的な通信時間は 1.16[ms] であり，ほぼ逐次的に 16 ノードの通信が行われたと考えられる．多数の 2 ノード間通信が同時に行われる場合，通信遅延を回避するために，通信パターンが同一スイッチ内通信となるよう，送信ノードは受信ノードを選択するべきである．

3.2 並列ボリュームレンダリングにおけるボトルネック

PC クラスタ上で可視化処理を行うにあたり，アプリケーションとして並列ボリュームレンダリングを考えている．本節では，比較的小規模なスイッチの

組み合わせによって構成されたPCクラスタにおける並列ボリュームレンダリングの描画処理についての検討を行う．並列ボリュームレンダリングにおけるMasterノードが最終画像表示するまでに要する時間T_{total}は式(3.1)で表される[21]．なお，式(3.1)では，各項の時間はノードの処理を表し，順番は処理の流れに沿っている．

$$T_{total} = T_{render} + T_{read} + T_{comm} + T_{compose} + T_{collect} + T_{draw} \tag{3.1}$$

但し，T_{render}はGPUによるサブボリュームのレンダリング時間，T_{read}はレンダリング結果のグラフィクスメモリのフレームバッファからの読み込み時間を表す．T_{comm}は2台のサブノード間の転送時間表す．逐次合成の場合，レンダリング後直ちに最終画像表示ノードへの転送を行うため，この時間は省略される．$T_{compose}$は受信画像の合成時間を表す．ただし，逐次合成の場合，この時間は省略される．合成処理はCPUまたはGPUによって行われる．$T_{collect}$は合成結果のMasterノードへの転送時間を表し，T_{draw}はMasterノードが最終画像表示に要する時間を表す．画像表示処理はGPUにより行われる．

想定しているアプリケーションの要求に応えるスクリーンサイズは，1024×1024である．このスクリーンサイズにおいて並列ボリュームレンダリングを行い，描画速度の高速化に向けてボトルネックとなる処理についての検討を行う．

3.2.1 実験環境

64台のPCで構成されるクラスタにおいて，OpenGLグラフィックスライブラリ，lam7.1.1/MPIを用いて並列ボリュームレンダリングを行う．64台のサブノードにおいてサブボリュームのレンダリング処理を行い，Masterノードが最終合成画像を得るまでの描画速度および各処理時間を測定する．合成アルゴリズムは，2.3.3および2.3.4項で述べた逐次合成，ランレングス符号化を適用した逐次合成，BSCを用いる．ただし，本環境では，BSCのサブノード間合成処理はCPUを用いる．Swap処理のデータ受信終了後直ちにComposite処理をCPUで行うことになるが，Composite処理をCPUではなくGPUで行った場合，CPUとGPUの並列処理により，処理時間が短縮される可能性がある．ただし，フレームバッファからの読み込み，書き込み時間と処理時間のロードバランスを考慮する必要がある．

シミュレーションに用いたノード構成を表3.3に示す．サブボリュームの分割方法は，可視化の対象となるボリュームデータを各座標軸に対して均等に分割

し，各分割データをそれぞれの座標に対応する担当のノードに配置する．64 台のノードのトポロジーは，図 3.3 に示すように，独立した 4 つの系統のネットワーク間を接続する上位の GbE SW を設け，スイッチを多段結合することにより．Master ノードと全サブノード間での通信路を確保している．

表 3.3: ノード仕様

CPU	Pentium4 3.4GHz
OS	Linux 2.6.9
Memory	1.0 GB
バス	PCI-Express
コンパイラ	gcc 3.4.2
オプション	-O3 -mfpmath=sse -march=pentium4
Motherboard	ASUSTeK P5AD2-E Premium
Gfx Card	GeForce6800GT PCI-Express
Gfx Memory	256 MB
通信ライブラリ	lam7.1.1/MPI
実測性能	700 Mbps

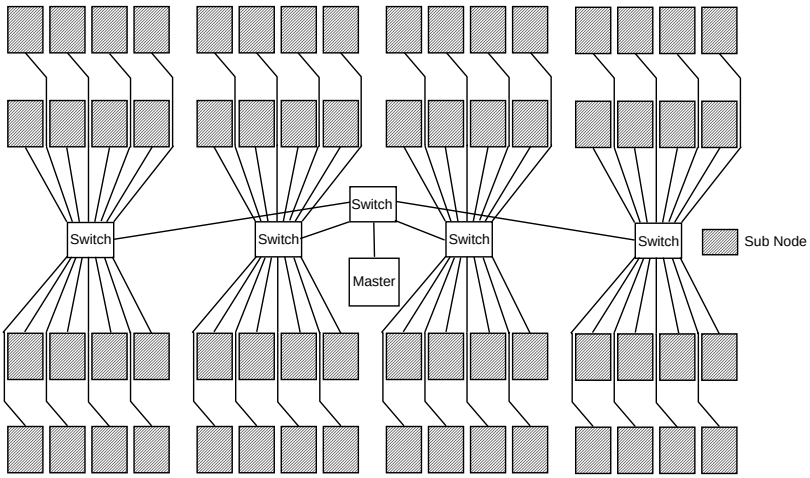
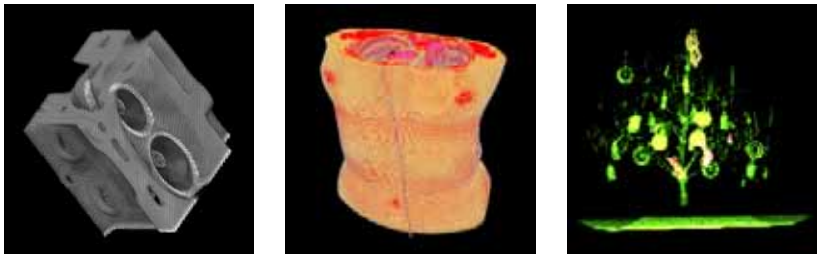


図 3.3: ノードおよびスイッチのトポロジー

評価に使用したボリュームデータは Engine(エンジンのボリュームデータで,

サイズは 256 × 256 × 128 である (図 3.4-(a)), Chest(人間の胸部のボリュームデータで，サイズは 512 × 512 × 512 である (図 3.4-(b)), Tree(クリスマスツリーのボリュームデータで，サイズは 512 × 512 × 1024 である (図 3.4-(c)) である．



(a) Engine (b) Chest (c) Tree

図 3.4: 使用したボリュームデータ

Master で表示するスクリーンのサイズは 1024×1024 とし，各サブノードが生成するサブスクリーンのサイズは 256×256 となる．スライスの数はボリュームデータの各軸方向のサイズの最大値とし，奥行き方向の解像度は縦，横方向の解像度と等しく設定した．不透明度については各サブボリュームデータともにボクセル値と等しい値をとっている．

測定方法は以下のとおりである．オブジェクトを固定し，連続して 360 回のレンダリング命令を発行させ，それぞれの平均時間を求めた．

BSC の場合，各ノードはほぼ同時に Swap 処理と Composite 処理を行うが，Composite 処理後の時間は完全に同じではない．そのため，Swap 処理の直前と Composite 処理の直後に全ノードの同期をとり，これらの処理時間を測定した．同期をとる場合，正確な総描画時間を測定することは不可能であり，これら各処理時間の和を総描画時間とした．また，Master ノードでの描画処理時間にはサブノードから送信されるデータをグラフィクスメモリへ受け渡すための配列データにコピーを行う時間を含む．

3.2.2 実験結果

各条件における描画速度および各処理時間の測定結果を表 3.4 に示す．また，測定結果より求められる総描画時間を表 3.5 に示す．

表 3.4: 各処理時間の測定結果

	Engine			Chest			Tree		
	逐次	逐次+RL	BSC	逐次	逐次+RL	BSC	逐次	逐次+RL	BSC
T_{render}[ms]	12.0	12.0	12.3	12.4	12.5	12.6	12.6	12.6	12.6
T_{read}[ms]	0.7	0.6	0.7	0.5	0.5	0.5	0.6	0.6	0.6
T_{comp}[ms]	-	-	57.7	-	-	56.5	-	-	56.4
T_{comm}[ms]	-	-	112.0	-	-	121.4	-	-	123.5
$T_{collect}$[ms]	221.3	183.5	38.9	250.5	226.5	37.8	293.5	183.0	35.9
T_{draw}[ms]	54.1	52.4	36.8	59.0	55.7	37.9	59.0	57.3	37.7

表 3.5: 総描画時間の比較

	Engine			Chest			Tree		
	逐次	逐次+RL	BSC	逐次	逐次+RL	BSC	逐次	逐次+RL	BSC
T_{total}[ms]	288.1	248.5	258.4	322.4	295.2	266.7	365.7	253.5	266.7
描画速度 [fps]	3.47	4.02	3.87	3.10	3.39	3.74	2.73	3.94	3.74

データサイズの違いによって生じる時間の差は，サブボリュームレンダリング時間である．しかし，総描画時間の内，この時間が占める割合は数% に過ぎないため，描画速度に影響を与えない．総描画時間の内占める割合が最も大きい処理はサブノード間とサブノード・Master ノード間通信であり，逐次合成の場合，ランレングス圧縮の有無に関わらず，70% ~80% である．また，BSC では，40% ~50% がノード間の通信時間である．BSC のサブノード間画像合成の時間が 20% である．全サブノードが 6 ステップの合成処理および画像データの送受信を行っている．第 1 ステップから第 4 ステップは 4MB から 500KB の比較的大きいデータを Swap するが，同一スイッチ内のノード間通信であるため，ネットワークの競合は生じない．しかし，第 5,6 ステップは，250kB,125kB の画像データをスイッチ外のノード間で Swap を行うため，ネットワークの競合が発生するためにネットワークの負荷が大きくなり，遅延を招いている．このため，逐次合成と比較した場合の BSC の性能向上が著しくない．

ランレングス符号化を用いることで最も描画速度が速くなった．これは，データ転送量が 40~60% 圧縮されたためである．通信時間を短縮するためには，通信データ量の削減が最も効果的であると考える．

以上より，重畳処理 (合成処理とデータ通信) が描画速度のボトルネックとなっ

ている．重畳処理時間の短縮のためには，合成処理の高速化および通信時間の高速化が必要とされる．

3.3 本章のまとめ

本章では，PC クラスタによるインタラクティブシミュレーションシステムを構築するにあたり，スイッチの多段結合，および並列可視化処理における問題について述べ，その検討を行った．まず，スイッチの多段結合におけるノード間通信に関しては，スイッチの多段結合によって接続されるノード間同士が同時に一斉送信を行うと，ノード台数に比例してネットワークの競合が生じる．このような競合を回避するためには，送信ノード数を削減するなど，ネットワークの要求仕様に応えうる送受信ノード選択が必要である．PC クラスタ上で並列可視化処理を行うにあたり，ボトルネックとなる要素を明らかにした．スクリーンサイズが 1024×1024 規模の 64 台の並列ボリュームレンダリングでは，総描画時間に重畳処理時間が占める割合が 60% ～80% 以上であることが分かった．並列ボリュームレンダリングの高速化を図るためには，通信データ量の削減可能な合成アルゴリズムを提案し，通信時間および合成時間を削減することが必要である．

第4章　Scubeの構築

本章では，実時間インタラクティブシミュレーション環境の実現にむけて，PC クラスタにより Scube の構築を行う．本システムでは，通信のスケジューリングにより，計算負荷の高い数値シミュレーションとその結果の実時間可視化の同時処理を目指す．また，シミュレーションと可視化処理における通信の高速化を目指す．これらのシステム要件に対し，システム全体をコモデティ部品の組み合わせにより実現し，高いコストパフォーマンスを保証する．特にネットワークにおいて，中規模コモディティクラスタ向け相互結合網 Three Quads を提案し，Three Quads の定性的な評価を行う．

4.1　Scubeの概要

Scube は，64 ノード構成の PC クラスタで，各ノードに可視化処理のための GPU を搭載する．計算負荷の高いシミュレーションと多次元データの可視化までを PC クラスタベースの計算サーバで行い，オペレータの操作や結果の呈示を行うユーザインタフェース端末としてのクライアントを用いたクライアントサーバモデルである．Scube のノード間ネットワークは，実時間数値シミュレーションとシミュレーション結果の可視化処理の両方で共有する形態をとる．Scube を構成するクラスタノードおよびネットワークの仕様を表 4.1 に示す．

表 4.1: Scube のノード仕様

CPU	Pentium4 3.4GHz
OS	Linux 2.6.9
Memory	1.0GB
Motherboard	ASUSTeK P5AD2-E Premium
Gfx Card	GeForce6800GT PCI-Express
Gfx Memory	256MB
Network	Gigabit Ethernet×3

4.2 ノード配置及びスイッチの物理的配置

Scube の各ノードは，論理的には図 4.1 に示すように，3 次元に配置されているものと考え，各ノードの 3 次元座標（x, y, z）に対応した 3 次元のノード番号を付与する．各座標系のネットワーク構成を図 4.2~4.4 に示す．3 次元ノード番号において x の値が一定のノード群 (x,∗,∗) に対して図 4.2 のように，4 つの GbE SW を用いて独立した 4 組のネットワークを構成する．y,z の値が一定のノード群についても x と同様に，それぞれ図 4.3,4.4 のようにネットワークを構成する．基本的には，それぞれの座標系に対して 4 つのノード群が存在し，各群ごとに 16 ポートの GbE SW を割当て，ノード数は常に 16 台である．なお，Z 座標系のネットワークでは，システム全体の管理を目的として，独立した 4 つの Z 座標系ネットワーク間を接続する上位の GbE SW を設け，Host 計算機やファイルサーバと全ノード間での通信路を確保している．しかし，以下の説明では，議論の混乱を避けるため，これらはないものとして考える．

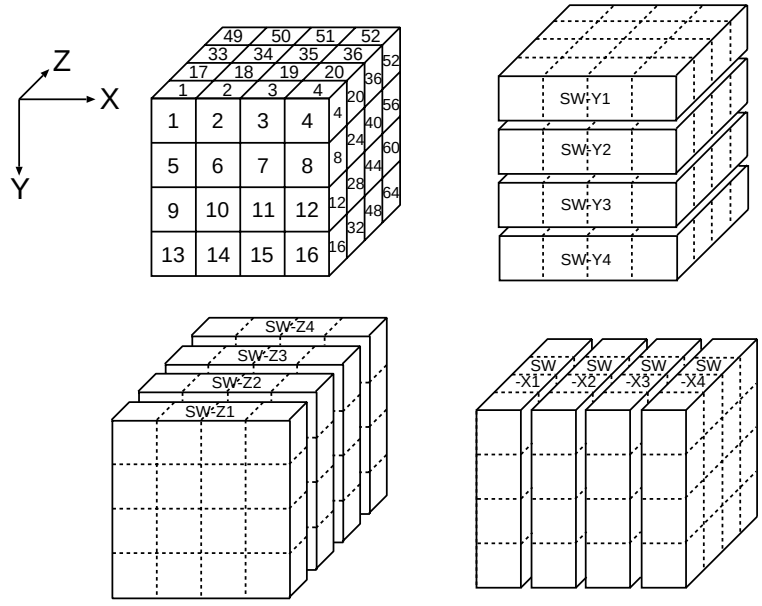


図 4.1: ノード配置および各スイッチへのノード割当て

4.3 相互結合網 Three Quad の提案

Three Quads は，数値シミュレーションと実時間可視化の同時並列実行を可能にすることを目的として設計した相互結合網であり，3 次元空間をプロセッサ

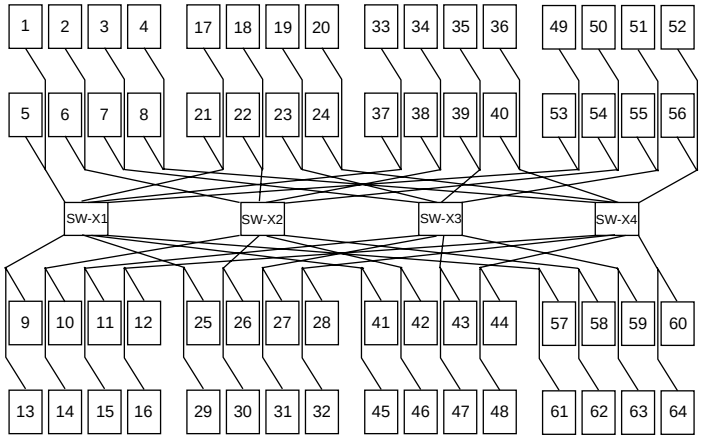


図 4.2: X 座標系のネットワーク構成

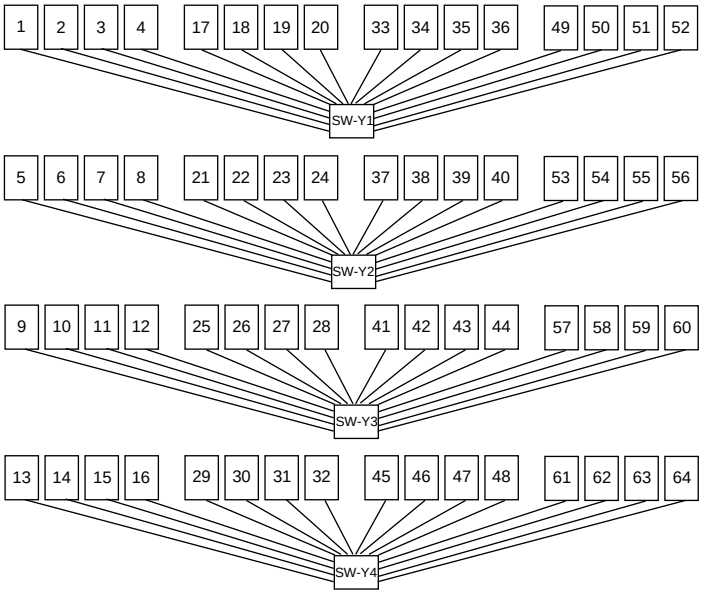


図 4.3: Y 座標系のネットワーク構成

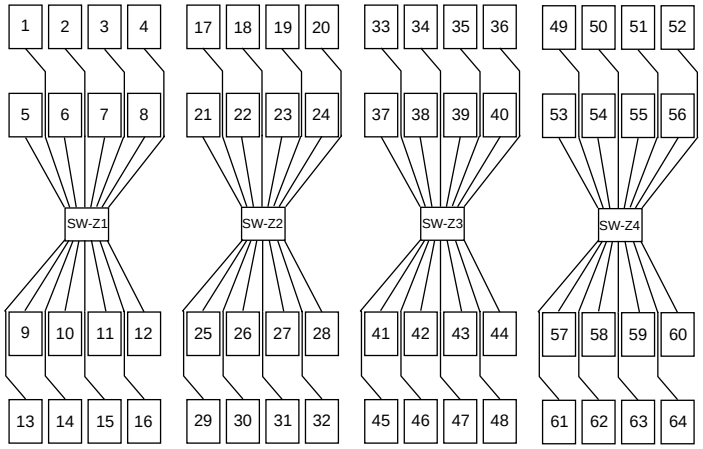


図 4.4: Z 座標系のネットワーク構成

空間に直接写像する数値シミュレーションにおける 3 次元隣接間通信 [12, 22]，3 次元画像合成処理における 3 次元 Reduction 型の通信 [1] を 1 つのネットワーク上で効率よく実現することを目的とした相互結合網である．またその実装に際しては中規模コモディティクラスタにターゲットを絞り，比較的小規模な GbE SW のみを用いて構成するとともに，複数系統設けたネットワークの冗長性を利用した通信のスケジューリングにより，シミュレーションと可視化処理の同時実行を目指している．

以下では，Three Quads の性質を説明した後，各種相互結合網の埋め込みについて述べる．

4.3.1 トポロジ

以下簡単のため, Three Quads の各次元に対して各々2^n 台のノードが配置されているものとする．任意のノード番号を N_i とし，この時，ノード番号は 2 進 $3n$ 桁表記でそれぞれ $(a_{3n}, a_{3n-1}, \cdots, a_2, a_1)$，$2^n$ 進 3 桁表記でそれぞれ (A_3, A_2, A_1) と表記できるものとする．

Three Quads は，X 系統，Y 系統，Z 系統の 3 系等のネットワーク群で構成する．ノード数が X,Y,Z 方向それぞれ l, m, n 台のノードで構成されるシステムでは，X 系統のスイッチ群は $m \times n$ ポートのスイッチ l 台，Y 系統のスイッチ群は $l \times n$ ポートのスイッチ m 台，Z 系統のスイッチ群は $l \times m$ ポートのスイッチ n 台で構成し，各系統のスイッチは 3 次元座標系における当該系統の座標値に対応する．例えば，Z 系統の i 番目のスイッチは $Z = i$ 平面に対応しており，同一平面上のノードは 1 ホップで直接通信が可能となる．

各ノードが持つ 3 本のリンクは，X 系統，Y 系統，Z 系統のネットワークに 1 本づつ接続する．この際，ノード番号が (A_3, A_2, A_1) の場合，X 系統の A_1 番目のスイッチ，Y 系統の A_2 番目のスイッチ，Z 系統の A_3 番目のスイッチに接続する．ノード数が 64 台の場合，Z 系統のネットワークのノード配置は図 4.5 のようになされる．

このように **Three** は，ネットワーク全体において，3 系統 (X,Y,Z 座標系統) のネットワークが存在するということを表している．ベース-m n-キューブ [11] は，その定義に従うと各次元のスイッチのサイズを一定に定めているが，PACS-CS[12] 等において採用されているハイパークロスバにはこの制限がない [10]. Three Quads も同様，3 系統のネットワークの各スイッチのポート数を等しく設定する必要はない．

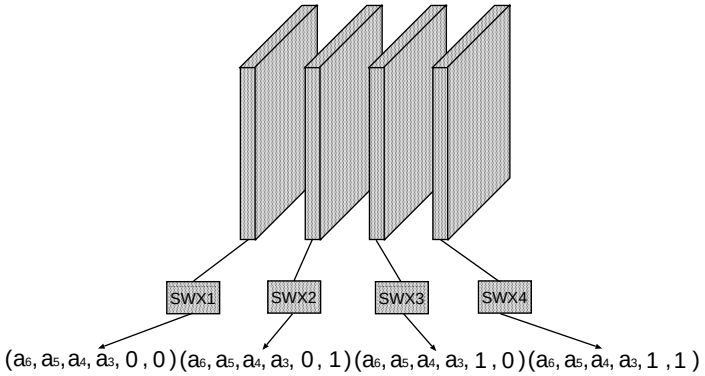


図 4.5: Z 系統のネットワークへのノード配置 (64 台の場合)

一方，**Quads** は，各スイッチに結合しているノード群が，X-Y,Y-Z,Z-X 平面を構成しているということを表す.

今，各次元方向のノード数が m であるとすると，ベース-m 3-キューブは 1 次元方向 (直線上) の m 台のノードがスイッチにより結合しているのに対し，Three Quads は 2 次元方向 (平面上) の m^2 台のノードがスイッチにより結合している．この時，Three Quads はベース-m 3-キューブの拡張と考えると非常に理解しやすく，ベース-m^2 3-キューブと呼ぶことができる．

4.3.2 ノード間通信

任意の 2 ノード A, B 間の 1 対 1 通信が最大 2 ホップで実現可能であることを示す．簡単のため, Three Quads の各次元に対して各々2^n 台のノードが配置されているものとする．この時，ノード A および B のノード番号は 2 進 $3n$ 桁表記でそれぞれ $(a_{3n}, a_{3n-1}, \cdots, a_2, a_1)$ および $(b_{3n}, b_{3n-1}, \cdots, b_4, b_3, b_2, b_1)$，$2^n$ 進 3 桁表記でそれぞれ (A_3, A_2, A_1) および (B_3, B_2, B_1) と表記できるものとする．

Three Quads では，上記 2^n 進 3 桁表記において最大 2 桁までが異なるノード間の直接通信が可能である．今，2^n 進 3 桁表記においてノード A と 1 桁が等しく，それ以外の桁がノード B と一致するノード C (C_3, C_2, C_1) を仮定する．この時，Three Quads 上では，ノード A は 1 ホップでノード C と通信が可能であると同時に，ノード C はノード B と 1 ホップで通信が可能である．したがって，ノード C を経由することで，ノード A とノード B の通信が可能であり，任意の 2 ノード間が 2 ホップで通信可能であることが示せる．

この仮定ではノード C の選び方，すなわち，ルーティング経路が 3 通り存在することが分かる．さらに，より細かく 2 進 $3n$ 桁表記で考えると，一致させ

るビット数の選択，また，一致させるビット位置の選び方にも複数の自由度があり．2 ホップという制約を課しても非常に多くのルーティング経路を設定することができる．例えば n=2 の場合，ノード C として $(a_6, a_5, a_4, b_3, b_2, b_1)$ や $(a_6, a_5, a_4, a_3, b_2, b_1)$，さらには $(a_6, a_5, c_4, c_3, b_2, b_1)$ 等も選択可能である．

4.3.3 耐故障性 (fault-tolerance)

任意の 2 ノード間通信は，特定の 2 系統のスイッチのみを用いて実現が可能であり，ある特定の 1 系統のスイッチが故障した場合でも直径 2 を維持できる．冗長パス (リンク) が多数存在するため，ある隣接ノード間の 1 つのリンクが切断された場合であっても，最大 2 ホップ，最小 1 ホップで通信が可能となる．例えば，ノード N_1 と N_{64} が通信を行う場合ルーティングにおいて，Z 系統のネットワークを用い，SWZ1 内でノード N_4 を中継し，X 系統のネットワークの SWX4 内で N_4 と N_{64} が通信を行うと設定されていたとする．この場合，N_1 と N_{64} は 2 ホップで通信を行うことになる．ここで，N_1 の Z リンクが故障した場合，Y リンクを用い，SWY1 内で N_4 を中継し，X 系統のネットワークの SWX4 内で N_4 と N_{64} が通信を行うことで，2 ホップで N_1 と N_{64} が通信が完了する (図 4.6).

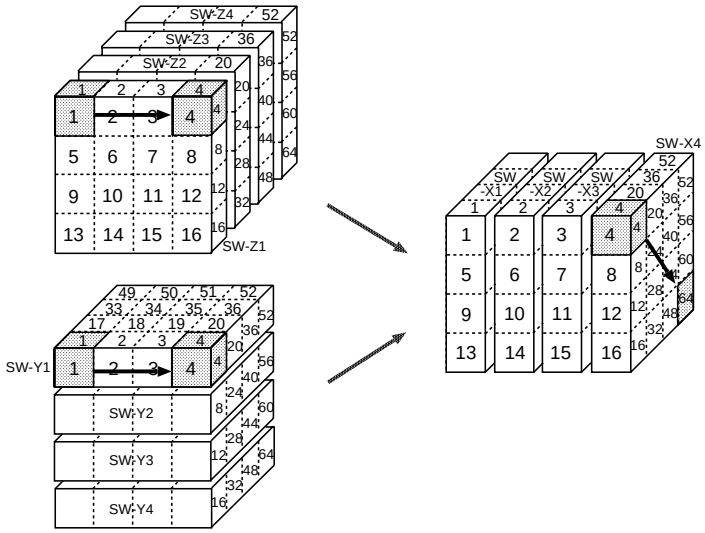


図 4.6: ノード N_1 と N_{64} 間の代替経路

4.3.4 スケーラビリティ

m ポートのスイッチを使用してシステムを構築する場合，ベース-m n-キューブでは m^3 のノード構成が可能であるのに対して，Three-Quads では，1 つのスイッチで平面状のノードを収容する必要があるため，最大ノード数は $m^{3/2}$ とな

る．これが中規模クラスタ向けと呼んでいる所以である．なお，m=64 とすれば，512 ノードのクラスタまでは構築可能である．

4.3.5 埋め込み性 (Embedability)

多数の他の網の特徴を包含し，必要に応じて各々の網を再現可能である必要がある [23]．64 台のノードが構成する Three Quads への主なネットワークの埋め込み能力について，トポロジーの観点から述べる．

- 1/2/3 次元トーラス網

  3 次元トーラスの埋め込みはほぼ自明であるため，2 次元トーラスの埋め込みについて説明する．Z 系統のネットワークを図 4.7 のように 2 次元に平面展開し，2 次元トーラスを考える．このとき，SWZ1 と SWZ2 ならびに SWZ3 と SWZ4 を接続する横方向のリンクは，Y 系統のネットワークを用いることによって接続可能である．一方，SWZ1 と SWZ3 ならびに SWZ2 と SWZ4 を接続する縦方向のリンクは，X 系統のネットワークを用いることによって接続可能である．これにより 2 次元トーラスの埋め込みが完了する．

  1 次元トーラス (リング) に関しても，Z 系統のネットワークを 1 次元展開することで同様に考えることが可能である．

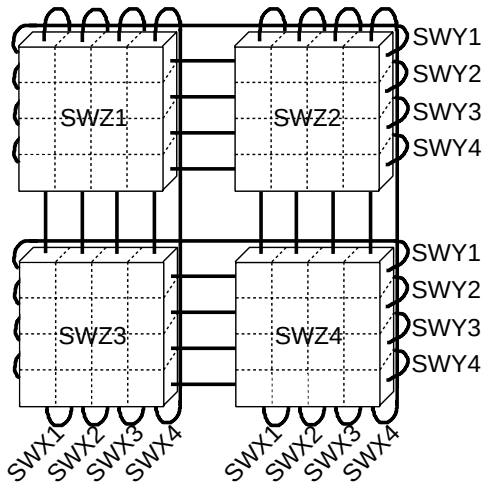


図 4.7: 2 次元トーラス網の埋め込み

- 単純トリー網

  Z 系統のネットワークで，16 ノードが直接結合している各 SW において段

数が 4 段のトリーが成立する．各スイッチ内におけるトリー網の親ノードのノードを仮に N_{16},N_{32},N_{48},N_{64} とする．これら親ノードは X 系統または Y 系統のネットワーク内の 1 つのスイッチにより結合しており，このスイッチにおいて 2 段のトリーが成立する．先の 16 ノードによる 4 段のトリーと親ノードによる 2 段のトリーによって，最終的に 6 段の単純トリーが完成する．

- Fat Tree

  Scube への Fat Tree の埋め込みを示す．Fat Tree の要素 (上向きリンク数, 下向きリンク数，階層数) を (2,4,3) とし，多重化されたツリー網を考える (図 4.8)．任意のノード $(a_6, a_5, a_4, a_3, a_2, a_1)$ は，まず Z 系統のネットワークにおいて 4 ノード $(a_6, a_5, a4, a_3, *, *)$ と結合し，4 分木が成立する．次に X 系統のネットワークにおいて 16 ノード $(a_6, a_5, *, *, *, *)$ と結合し，16 分木が成立する．最後に Y 系統のネットワークにおいて 64 ノードと結合し，64 ノードの Fat Tree が完成する．但し，各階層における Scube の物理的なリンク数は 1 なので，Fat Tree(図 4.8) のような 2 本の上向きリンクを同時に使用した通信をエミュレートすることはできない．しかし，1 本の物理リンクを時分割で 2 本のリンクとしてシミュレートすることで同時通信をシミュレートすることは可能である．

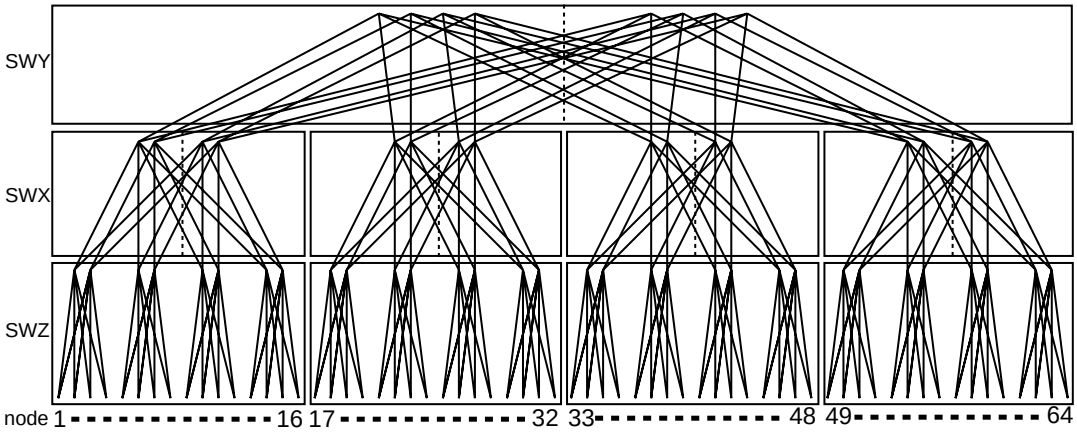


図 4.8: ファットトリーの埋め込み

- ハイパーキューブ (2 進 6-キューブ) 網

  任意の隣接ノード (ハミング距離が 1 となるノード) 間が結合していることを

示せばよい．任意のノード $(a_6, a_5, a_4, a_3, a_2, a_1)$ はZ系統のネットワークにおいて，ノード $(a_6, a_5, a_4, a_3, a_2, \bar{a_1})$,$(a_6, a_5, a_4, a_3, \bar{a_2}, a_1)$ と結合している．X系統のネットワークにおいて，ノード $(a_6, a_5, a_4, \bar{a_3}, a_2, a_1)$,$(a_6, a_5, \bar{a_4}, a_3, a_2, a_1)$ と結合している．Y系統のネットワークにおいて，ノード $(a_6, \bar{a_5}, a_4, a_3, a_2, a_1)$, $(\bar{a_6}, a_5, a_4, a_3, a_2, a_1)$ と結合している．よってX,Y,Z3系統のネットワークを用いることで，ハイパーキューブの埋め込みが可能である．但し，Fat Treeと同様，Scubeでは物理的なリンク数は3なので，2進6-キューブの6本のリンクを同時に使用した通信をエミュレートすることはできない．しかし，1本の物理リンクを時分割で2本のリンクとしてシミュレートすることで同時通信をシミュレートすることは可能である．

- ベース-m n-キューブ (ベース-4 3-キューブ)
  ベース-4 3-キューブの埋め込みはThree Quadsにおいて各スイッチを独立した4個のスイッチとみなすことで，埋め込みが可能である (図2.5参照)．例えばノード N_2(0,0,1) の場合，図4.9に示すように，SWX2またはSWY1において，ノード群 (i,0,1)(i=1,2,3) と結合している．SWX2またはSWZ1において，ノード群 (0,j,1)(j=1,2,3) と結合している．SWY1またはSWZ1において，ノード群 (0,0,k)(k=0,2,3) と結合している．

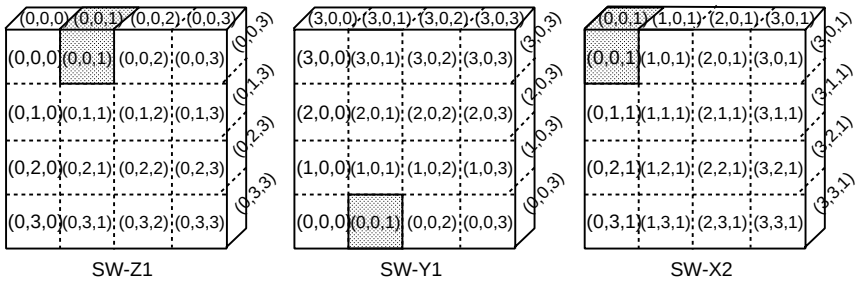


図 4.9: ベース-4 3-キューブの埋め込み

4.3.6 性能比較

ハイパーキューブ網，ベース-4 3-キューブ，ハイパークロス網，ベース-4^2 3-キューブについての主な性能について比較した．ノード数が64台の場合の性能比較を表4.2に示す．隣接ノード数は3次元シミュレーション空間上の隣接ノード数を表す．また，スイッチ数は物理的なクロスバスイッチの数であり，ベース-4 3-キューブ，ベース-4^2 3-キューブのスイッチ数において $(n \times n)$ は n ポー

トのクロスバスイッチを表す.

表 4.2: 各種ネットワークの性能比較 (ノード数 64 台の場合)

	Hypercube	base-4 3-cube	Hypercross	base-4^2 3-cube
次数	6	3	2	3
直径	6	3	2	2
隣接ノード数	6	6	-	18
平均ホップ数	3	2.25	1.75	1.41
スイッチ数	0	(4×4)×64	8×8×8	(16×16)×12

ベース-4^2 3-キューブは他網と比較すると，スイッチ数は多くなるが，平均ホップ数が少なく，隣接ノード間も含めた任意のノード間の通信経路が多く存在するため，対故障性に優れている．特に隣接ノード数が多く存在し，3 次元シミュレーションモデルにおいて非常に優位性を有する．1 ホップ通信が可能なノード数は，ベース-m 3-キューブの場合は 9 台 (内，隣接ノード数は 6 台) であるのに対し，Three Quads の場合は 36 台 (内，隣接ノード数は 18 台) である (図 4.10). そのため，Three Quads は，ベース-m 3-キューブと比較するとランダムな通信要求が発生した場合など，柔軟なネットワークリソースの提供が可能である.

ハイパークロス網 [24] は Three Quads と同様，直径が 2 であり，数値シミュレーションにおける行列転置の重要性に注目して，転置を効率良く実現するネットワークとして提案されたネットワークであるが，次項で示すように，Three Quads も高い行列転置能力を有している.

4.3.7 転置行列演算への適用

科学技術計算において多用される転置行列演算 [25, 26, 27] が，Three Quads を用いることで，効率良く実現できることを示す.

いま，各ノードに $N \times N$ の配列を割り当て，システム全体 (64 ノード) で $8N \times 8N$ の行列 $\mathbf{M}$ を転置する場合を考える．この時，Scube のノードを図 4.11 のような 2 次元平面展開した時の i 行 j 列のノードがもつ $N \times N$ の配列を $\mathbf{M}_{ij}$ と表すこととする.

第 1 ステップでは，$\mathbf{M}$ をサイズ $4N \times 4N$ の 4 つの部分行列 $\mathbf{A}$,$\mathbf{B}$,$\mathbf{C}$,$\mathbf{D}$ を用いて，$\mathbf{M} = \begin{pmatrix} \mathbf{AB} \\ \mathbf{CD} \end{pmatrix}$ によって表した時の $\mathbf{A}^t$,$\mathbf{B}^t$,$\mathbf{C}^t$,$\mathbf{D}^t$ を求める．なお，$\mathbf{A}$,$\mathbf{B}$,$\mathbf{C}$,$\mathbf{D}$ の

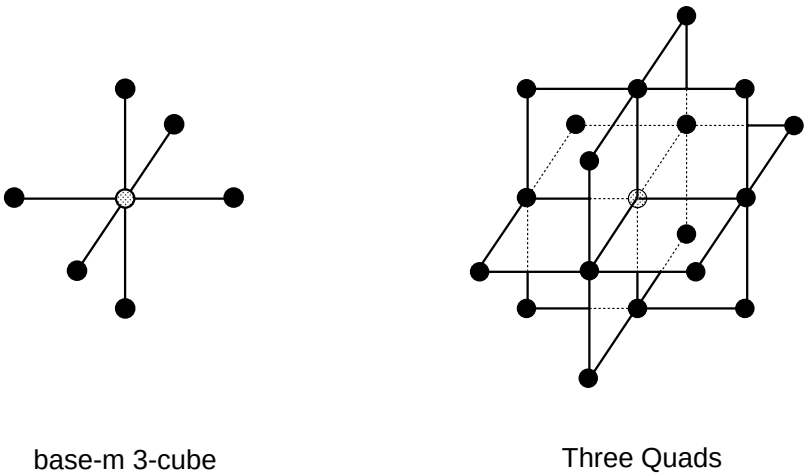


図 4.10: 3 次元モデル上のベース-m 3-キューブと Three Quads の隣接ノード数比較

各要素を a_{ij},b_{ij},c_{ij},$d_{ij}(0{\leq}i,j{\leq}3)$ と表現するものとする．このステップでは各部分行列の転置に必要な通信がそれぞれ，SWZ1,SWZ2,SWZ3,SWZ4 内に閉じた 1 対 1 通信であるため，ネットワーク上でのリソース競合が発生せず，全ノードが並列に同時処理することが可能である．なお，通信に先立ち，各ノード内では $\mathbf{M}'_{ij}$ を求めておくものとする[1)]．これにより，$\mathbf{M}' = \left(\begin{smallmatrix}\mathbf{A}^t\mathbf{B}^t\\ \mathbf{C}^t\mathbf{D}^t\end{smallmatrix}\right)$ が得られる．

第 2 ステップでは，$\mathbf{M}'$ における $\mathbf{C}^t$,$\mathbf{B}^t$ を入れ換えることで，$\mathbf{M}^t = \left(\begin{smallmatrix}\mathbf{A}^t\mathbf{C}^t\\ \mathbf{B}^t\mathbf{D}^t\end{smallmatrix}\right)$ を求める．$\mathbf{M}'$ と $\mathbf{M}^t$ の関係をみると $\mathbf{A}^t$,$\mathbf{D}^t$ は既に転置は完了しており，通信を行う必要はない．いま，c^t_{ij},b^t_{ij} が置かれているノードの位置関係を考えると，これらは X 系統あるいは Y 系統のスイッチにおいて直接通信可能な位置にあり，すべての i,j に対して同時に通信を行っても，一切のネットワークの競合は発生しない．よって，このステップも全ノードが並列に同時実行可能である．これにより，2 ステップで行列の転置が可能であることを示せる．

さらに第 1 ステップと第 2 ステップで使用するネットワークが全く独立であることに注目すると，通信のタイリングなどの技術を用いることで，これらの 2 つのステップが融合可能であり，実質的には 1 ステップで転置が可能であることが分かる．

また，このアルゴリズムで使用したネットワークは 2 系統のみで他の 1 系統は全く使用していない．したがって，他の 1 系統のネットワークが他の用途で

[1)] これは最後のステップに行ってもよい．

使用されていても，理論上このアルゴリズムの実装には影響を与えない．

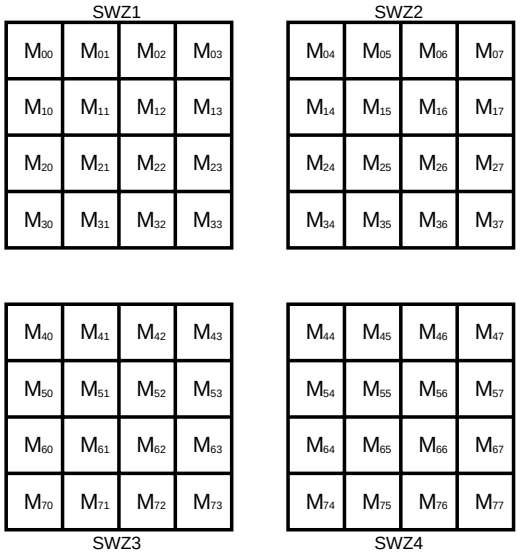


図 4.11: 初期設定

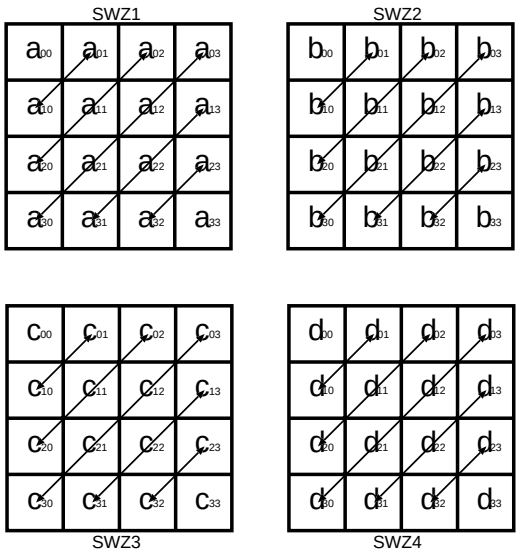


図 4.12: 第 1 ステップ

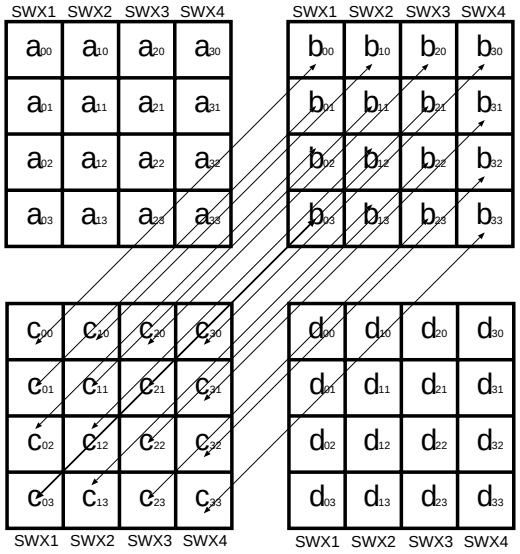


図 4.13: 第 2 ステップ

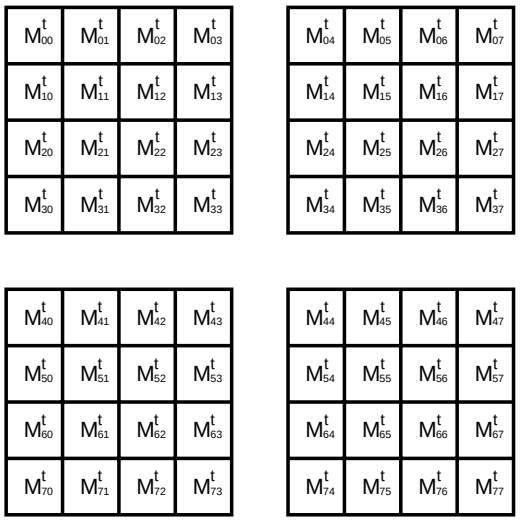


図 4.14: 最終状態

4.4 本章のまとめ

実時間インタラクティブシミュレーションシステムの実現に向けて，コモデティ技術を使用した Scube の構築を行った．Scube 上にコモデティ技術を用いた中規模 PC クラスタ向けの相互結合網 Three Quads の提案を行い，主な特徴，

定性的な評価について述べた．Three Quads は，ベース-m 3-キューブの拡張と考えられ，ベース-m^2 3-キューブと呼ぶことができる．また，任意の 2 ノード間通信が 2 ホップで通信可能であり，通信経路が多く存在し，耐故障性，多様な網の埋め込み能力に優れていることを示した．Three Quads は数値シミュレーションに対して科学技術計算に広く使用されている行列の転置演算処理の全対全通信において，ネットワークの競合が生じることなく，高速に実現が可能である．

第5章　主軸優先木構造合成

第3章ではsort-last並列ボリュームレンダリングにおいて，システム性能を考えた場合，重畳処理の高速化が重要な問題であることを示した．本章では，通信データ量を削減し，通信時間を短縮するアルゴリズム (以下主軸優先合成) を提案する．

5.1　提案合成アルゴリズム

単純な木構造合成では，合成アルゴリズムの終盤で通信量と演算量が大幅に増加するが，提案するシステムでは重複度を考慮すると実質的な通信量と演算量の増加を防げるという性質を利用する．また，ノード間の木構造合成の順番を適切にスケジュールすることで，アルゴリズムの実行ステップにおける通信パターンを規則化することで，ネットワークに対する要求仕様を緩和する．

具体的には，2分木構造に基づく合成処理 (図5.1) において，合成処理に参加する各ノードは，各々のステージにおいて，視線ベクトルの絶対値が最も大きい成分 (主軸) 方向の隣接ノード間との合成を優先して行う．これにより，各ノードの合成処理において最も重複の多い中間画像間での合成が可能となり，合成結果として得られる画像サイズの不必要な増加を抑制する．また，各々のステージでは，ノード間の通信が x,y あるいは z 軸の何れか一つのみに平行な極めて規則的な通信パターンとなり，ネットワークに対するランダム通信性能の要求を軽減することが可能である．

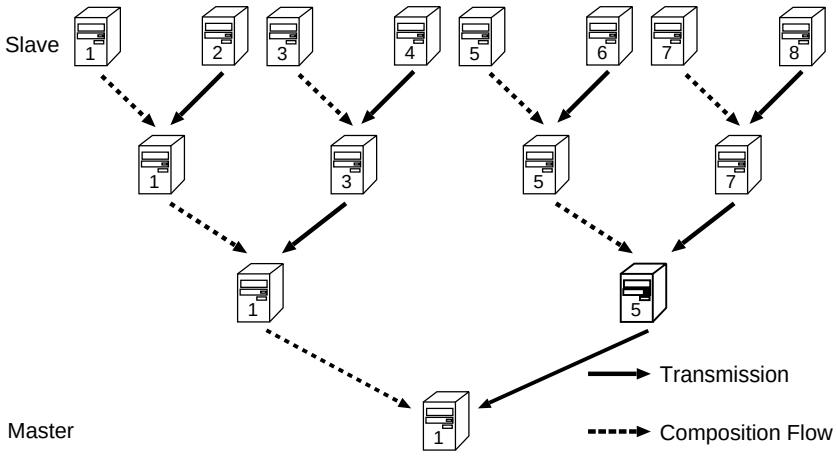


図5.1: 2分木構造の合成ステップ (サブノード8台の場合)

例えば，ボリュームデータを 8 個のサブボリュームに分割している場合，任意のサブボリュームに隣接するサブボリュームは 3 つである．その 3 つのサブボリュームはそれぞれボリュームデータ固有の座標系の X,Y,Z 方向と対応させることができる (図 5.2)．視線ベクトルを (x, y, z) と表すとする．

あるサブボリューム V_0 について，Z 方向に隣接するサブボリューム V_z の投影される領域と，V_0 の投影される領域が重なる部分領域の大きさは，視線ベクトルの z 成分の絶対値の大きさに依存する．例えば，視線ベクトルが (0,0,-1) のとき，どのサブボリュームの投影領域も，x, y 方向に隣接するサブボリュームの投影領域とは全く重ならない．しかし，Z 方向に隣接するサブボリュームの投影領域を考えると，互いに完全に重なっている (図 5.3)．視線ベクトルの各成分の絶対値が大きいほど，その方向に隣接するサブボリュームの投影領域の重複部分が大きくなる．

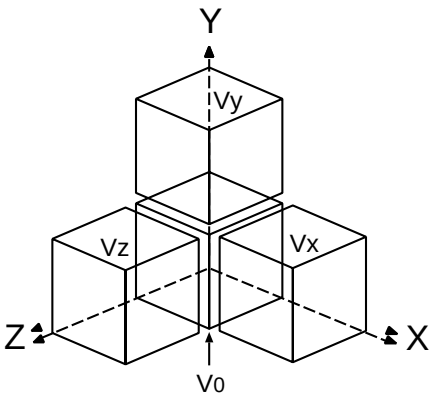


図 5.2: 隣接するサブボリュームと軸の対応

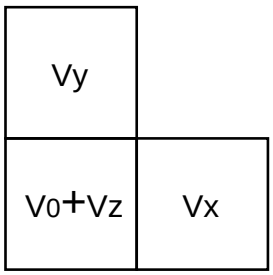


図 5.3: 視線ベクトル (0,0,-1) の場合のサブボリュームの重複関係

主軸に基いて合成処理を行うことにより，合成処理するために選ばれた 2 つのサブボリュームの中間画像の重複部分が最も大きくなる．このような 2 台のサブノード間で 2 分木の枝を構成し，合成処理を行っていくことで，合成結果のデータサイズを最も小さくすることができ，通信コストの削減に繋がる．

本合成アルゴリズムは第 4 章で提案した Three Quads 上に実装することで，いずれか 1 つのスイッチ内で X（あるいは Y,Z）方向の合成処理を完結することが可能である．また，各々のステージでは，ノード間の通信パターンが X,Y あるいは Z 軸の何れか一つのみに平行な極めて規則的な通信パターンとなるよ

う静的に合成スケジューリングを行うことで，視線情報の頻繁な変更などネットワークに対するランダム通信性能の要求に対応可能である．本合成アルゴリズムは Three Quads と非常に親和性の高い合成アルゴリズムあると言える．

5.2 理論的考察

第 5.1 節で提案する合成アルゴリズムの理論的な考察を行うにあたり，Scube 上のノード間 1 対 1 通信の通信レイテンシとバンド幅の測定を行う．MPI 通信ライブラリ lam7.1.1/MPI を用い，1 対 1 通信における通信レイテンシおよびバンド幅を pingpong プログラムにより測定した．

- 通信レイテンシ (各プロセッサ間の通信に対する処理時間)
  ソフトウェアオーバヘッド，インターフェース遅延を含み，実質的にデータのないメッセージの送信に要する時間である．ルーティング時間，発信元でメッセージをパックする時間と宛先でアンパックする時間が含まれる．一般的にこの時間は一定であると仮定される．
  通信レイテンシは，通信データ量や通信回数によって総処理時間のボトルネックとなる可能性がある．0Bytes のデータを 1000 回送信，受信を行い，送信開始から受信完了までに要した平均時間 $T_{send-recieve}$ を計測し，1 回当たりの送信に要する時間 $T_{send}(T_{send} = \frac{T_{send-recieve}}{2})$ を求めた．
- バンド幅
  一定サイズのメッセージを指定回数，送信 · 受信を行い，要した時間を測定した．バンド幅の計算には，式 (5.1) を用いた．

$$\text{バンド幅} = \frac{\text{メッセージサイズ} \times \text{指定回数} \times 2}{\text{実行時間}} \tag{5.1}$$

X,Y,Z 系統それぞれのネットワークの測定結果を表 5.1 に示す．

表 5.1: 通信性能

	レイテンシ [ms]	バンド幅 [MB/s]
X,Y 系統	0.162	87.41
Z 系統	0.057	47.03

ノード間合成において，想定される最小の通信データ量は，65536Bytes である．通信性能の測定結果 (表 5.1) より，各系統において，データ量が 65536Bytes

の場合の通信時間は，X,Y 系統の場合，0.749[ms]，Z 系統の場合 1.39[ms] と求められる．Z 系統の場合の通信レイテンシが通信時間に占める割合は 4% であり，通信レイテンシは通信時間に隠蔽されるとみなすことが可能である．X,Y 系統の場合の通信レイテンシが通信時間に占める割合は 18% となるが，Z 系統同様，通信レイテンシは通信時間に隠蔽されるとみなす．以下に述べる理論的考察において，通信データ量と通信時間は比例関係にあるとする．

通信コストは Master あるいは表示ノードが最終画像を表示するまでの総通信時間と考えることができ，総通信時間は最終画像を得るまでの時間から主にレンダリング時間，リードバッファ時間，合成時間を除いた時間である．通信コストおよび総通信時間は，Master あるいは表示ノードが受信する通信データ量 T は比例関係にある．そこで，この Master が受信する通信データ量に着目する．$N_x \times N_y \times N_z (N_x{=}N_y{=}N_z{=}N)$ サイズのボリュームデータを，P 台のサブノードの並列ボリュームレンダリングにより，N^2 サイズのスクリーンに描画する場合を考える．

まず，主軸優先合成との比較のため，逐次的合成における通信量 T_{seq} について考える．各ノードは $N^2/P^{\frac{2}{3}}$ のスクリーンサイズの範囲を描画する．描画結果を画像表示ノードに順次送信する．したがって，T_{seq} は，

$$T_{seq} = (P-1)\frac{N^2}{P^{\frac{2}{3}}} \tag{5.2}$$

と導くことができる．

次に，BSC の通信量 T_{bsc} を導出する．BSC の項で述べたように，BSC の合成処理が完了するまでの通信ステップは $\log_2 P$ 回であり，通信データ量は $\frac{1}{2}N^2, \frac{1}{4}N^2, \frac{1}{8}N^2$ と $\frac{1}{2}$ ずつ減少する．さらに，全ノードが担当領域の合成処理完了後，最終画像を表示するために Master ノードに一斉に送信する必要がある．よって，BSC の合成処理に要する通信量と Master へ送信される通信量は，T_{bsc} によって表される．

$$\begin{aligned} T_{bsc} &= \Big((\underbrace{\frac{1}{2} + \frac{1}{2^2} + \cdots + \frac{1}{2^{\log_2 P}}}_{\text{サブノード間通信 } (\log_2 P \text{ ステップ})}) + \underbrace{\frac{P-1}{P}}_{Master \text{ への一斉逐次送信}} \Big) N^2 \\ &= \frac{2(P-1)}{P} N^2 \end{aligned} \tag{5.3}$$

主軸優先合成における通信量 T_{tree} について考える．逐次合成，ＢＳＣの通信量は視点情報に関らず，それぞれ式 (5.2)(5.3) で表され，一定である．一方，主

軸優先合成の通信量は視点情報によって主軸が変化し，バウンディングボックスの大きさが変化するため，通信量は一定ではない．

ボリュームデータに対し，視線ベクトル (-1,1,1)（または (1,1,1)，(-1,-1,1) など）を設定した場合のデータ量を $T_{tree-diag}$ とする．ノード数が 8 台の場合の合成処理フローおよびデータ量は図 5.4 のようになる．逐次合成と同様に，各ノードが $N^2/P^{\frac{2}{3}}$ のスクリーンサイズの範囲を描画すると仮定すると，$T_{tree-diag}$ は，以下の式 (5.4) のように導かれる．但し，$n = \log P^{\frac{1}{3}}$ である．

$$
\begin{aligned}
T_{tree-diag} &= \underbrace{\left(2^{n+2} - 2^2 + 2n\right)\frac{\sqrt{3}}{16}(\frac{N}{P^{\frac{1}{3}}})^2}_{\log P^{\frac{1}{3}}\text{ステップ}} \\
&+ \underbrace{\left(2^{2n+2} + n2^{n+1} - 2^{2n+1} - 2\right)\frac{\sqrt{3}}{16}(\frac{N}{P^{\frac{1}{3}}})^2}_{\log P^{\frac{1}{3}}\text{ステップ}} \\
&+ \underbrace{\left(2^{2n+2} - 2^{n+2} + n2^{2n+1}\right)\frac{\sqrt{3}}{16}(\frac{N}{P^{\frac{1}{3}}})^2}_{\log P^{\frac{1}{3}}\text{ステップ}} \\
&= \left(2^{2n+3} + (n-1)2^{2n+1} + n2^{n+1}\right. \\
&+ \left. 2n - 6\right)\frac{\sqrt{3}}{16}(\frac{N}{P^{\frac{1}{3}}})^2 \qquad (5.4)
\end{aligned}
$$

また，ボリュームデータに対し，視線ベクトル (0,0,1)（または (0,1,0)，(-1,0,0)）を設定し，視線ベクトルと主軸が平行関係にある場合の Master ノードが受信するデータ量を $T_{tree-para}$ とする．ノード数が 8 台の場合の合成処理フローおよびデータ量は図 5.5 のようになる．$T_{tree-para}$ は，以下の式 (5.5) のように，

$$
\begin{aligned}
T_{tree-para} &= \underbrace{\log P^{\frac{1}{3}} \cdot \frac{N^2}{P^{\frac{2}{3}}}}_{\text{主軸方向への通信 } (\log P^{\frac{1}{3}}\text{ステップ})} + \underbrace{\frac{N^2}{P^{\frac{2}{3}}} + \frac{N^2}{P^{\frac{1}{3}}} + \cdots + \frac{N^2}{2}}_{\text{主軸方向以外の通信 } (\log P^{\frac{2}{3}}\text{ステップ})} \\
&= \left((\frac{1}{3P^{\frac{2}{3}}}\log P) + (1 - (\frac{1}{2})^{\log P^{\frac{2}{3}}})\right)N^2 \qquad (5.5)
\end{aligned}
$$

と導くことができる．

ここで，P=8，64，512 の場合のそれぞれの合成アルゴリズムによる通信量は式（5.2）～（5.5）より，表 5.2 のように表される．

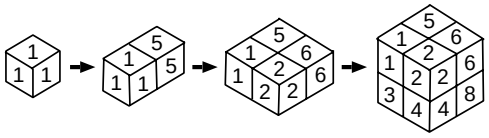


図 5.4: 視線ベクトル (-1,1,1) の場合の合成処理 ($T_{tree-diag}$)

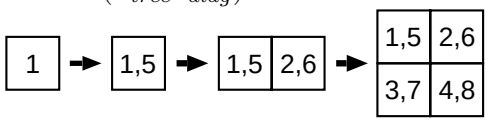


図 5.5: 視線ベクトル (0,0,1) の場合の合成処理 $T_{tree-para}$)

表 5.2: 理論的通信量の比較

	T_{seq}	T_{bsc}	$T_{tree-diag}$	$T_{tree-para}$
8 台	$1.75N^2$	$1.75N^2$	$0.87N^2$	N^2
64 台	$3.94N^2$	$1.97N^2$	$1.18N^2$	$1.06N^2$
512 台	$7.98N^2$	$2.00N^2$	$1.38N^2$	$1.03N^2$

T_{tree} は視点位置によらず，何れの場合も T_{seq} より小さいので，最終画像を得るまでの通信時間の短縮が図ることが可能であることが分かる．また，表 5.2 では $T_{tree-diag}$ と $T_{tree-para}$ は，表示スケールが異なるが，表示スケールを同じにした場合は，常に $T_{tree-para}$ が最も小さい値となる．$T_{tree-diag}$ の理論式 5.4 は，合成ステップは静的に決定した場合であり，合成ステップの最適化を行うことで，より一層の通信コストの削減が可能であると考えられる．ただし，$T_{tree-para}$ の値以下になることはない．主軸優先合成アルゴリズムを採用する効果は，サブノード台数が少ない場合は逐次合成，BSC と比較すると 43% の通信データ量の削減が可能であり，また，サブノード台数が増加すると，逐次合成と比較しておよそ 80% ，BSC と比較して 50% の通信データ量の削減が可能となる．

5.3 実装評価

Scube 上に並列ボリュームレンダリングの合成処理に主軸優先合成を採用し，描画速度の測定を行った．

5.3.1 実験環境

ノードおよびネットワークの構成は 3.2.1 項と同じである．スクリーンサイズは 1024×1024 と設定し，使用したボリュームデータは Tree(図 3.4-(c)) を用いた．描画測定方法は，主軸が Y 軸となるようにオブジェクトを回転させる．主軸優先合成では，各 Slave は分割されたサブボリュームデータを描画し，視線情報により静的に決定される順序に従って，2 分木による画像合成を行う．合成結果を転送する画像サイズは 2 枚の Bounding Box(図 5.6) とする．測定方法は第 3.2.2 項の実験環境における BSC の測定方法と同じとする．

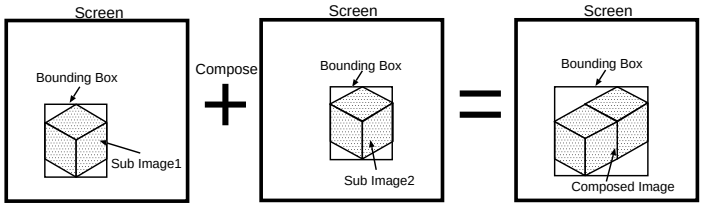


図 5.6: バウンディングボックス

5.3.2 主軸優先合成の効果

ネットワーク X 系統を用いた主軸優先合成の各処理時間の測定結果を表 5.3 に示す．ただし，比較対象として，逐次合成と BSC の各処理時間の評価を併記する．

表 5.3: 主軸優先合成の各処理時間の測定結果

	主軸優先	逐次	BSC
T_{render}[ms]	12.6	12.6	12.6
T_{read}[ms]	0.7	0.6	0.6
T_{comp}[ms]	26.7	-	56.4
T_{comm}[ms]	42.3	-	123.5
$T_{collect}$[ms]	33.9	293.5	35.9
T_{draw}[ms]	28.5	59.0	37.7

また，測定結果より求められる総描画時間を表 5.4 に示す．

- BSC と主軸優先合成の比較

表 5.4: 総描画時間の比較

	主軸優先	逐次	BSC
T_{total}[ms]	144.7	365.7	266.7
描画速度 [fps]	6.91	2.73	3.74

合成時間，通信時間ともに 53% の短縮が可能となった．理論的には，表 5.2 の通信量は合成データ量と等しく，理論的な合成時間，通信時間は 50% 程度の短縮が可能である．合成時間は，理論値とほぼ等しい時間の短縮率であった．要因として，視線ベクトルを (0,0,1) に設定したことが挙げられるが，理論的な通信時間はネットワークの競合を考慮していない．しかし，実際は，ネットワークの競合が生じる．つまり，主軸優先合成の場合，同一スイッチ内通信は最初の 4 ステップで行われる．残りの 2 ステップは，スイッチを跨いだ通信が行われるが，通信を行うノード数は 4 ノード，2 ノードである．BSC の場合，主軸優先合成と同様，同一スイッチ内通信は最初の 4 ステップで行われる．また，残りの 2 ステップも，スイッチを跨いだ通信が行われるが，通信を行うノード数は全 64 ノードであり，各スイッチにおいて 16 ノードの送信要求に対する逐次処理が行われる．このため，通信遅延が生じている．それぞれのスイッチを跨ぐ通信の送信データサイズは，主軸優先合成の場合はスクリーンサイズの約 1/4,1/2 と大きいが，BSC の場合は，スクリーンサイズの 1/32,1/64 と小さい．これらトータルの通信遅延時間を考慮すると，主軸優先合成の方が遅延はほぼ同じであったと言える．

- 逐次合成と主軸優先合成の比較
  主軸優先合成の通信時間は逐次合成のおよそ 1/4 となった．理論的な通信量の比較から，主軸優先合成の通信時間は，逐次合成のおよそ 1/4 である．やはり，逐次合成の場合，全サブノードが Master に一斉にサブボリュームレンダリング結果の送信を行うため，ネットワークの競合が各スイッチ内において頻繁に発生している．一方，主軸優先合成では，BSC との比較で述べたとおり，第 5,6 ステップにおいてネットワークの競合が生じるが，その影響は逐次合成と比較すると軽微であるといえる．

以上の比較より，通信コストにおける理論的な考察は正しかったといえる．

5.4 高速化に向けた検討

主軸優先合成を採用することで，通信コストが削減され，理論的な描画速度の向上は可能となった．本節では，今後更なる高速化に向けて，検討を行う．

- 通信と合成のパイプライン処理
  合成処理に必要な BoundingBox の領域のデータを通信後，合成処理を行う．全てのデータの合成終了後にブロッキング通信方式を採用し，通信を行っている．合成データをある大きさのブロックに分割し，合成処理が終了したブロックから順次送信を行う．合成処理と通信のパイプライン化を行うことにより，通信時間に対して合成処理に要する時間の隠蔽が行えると考えられる．
- バックグラウンド処理
  各合成フェーズにおいて，合成処理に必要のないデータを送信しないという最適化は考慮していない．この最適化を考慮すると，主軸優先合成の通信量は，より一層削減されると期待できる．合成処理に不要なデータの回収を主軸優先合成処理のバックグラウンド処理として行うことにより，更なる高速化が可能であると考えられる．サブノード間で通信されているデータに着目すると，その後のサブノード間合成処理の際に必要のない画像データも送受信されている．合成処理に必要のない画像データとは，そのデータを他ノードに送信しても，合成に用いられないデータのことを指す．このような画像データを有するノードが一斉に 1 つのノード (例えば Master ノード) に送信した場合，この処理自体がボトルネックとなる可能性が生じる．主軸優先合成では，合成結果を送信し終えたノードは，最終画像合成が完了するまでの間，CPU はアイドル状態にある．このようなノード間で木構造などによって合成を行い，最終的にネットワークの競合が起こり得にくい段階で Master ノードへの送信を行う．

5.5 本章のまとめ

中間画像の通信量に着目し，最終画像を得るまでに要する描画時間における，データ通信に要する時間を短縮するために，サブボリューム間の重複関係を考慮した主軸優先木構造による合成手法を提案した．主軸優先木構造合成において，従来の視線に基いた合成処理を行うのではなく，主軸に基いた合成処理を

行うことにより，理論上，ノード台数が少ない場合は逐次合成，BSC と比較すると 43% の通信データ量の削減が可能であり，また，ノード台数が増加すると，逐次合成と比較しておよそ 80% ，BSC と比較して 50% の通信データ量の削減が可能である．Scube 上に主軸優先合成を実装し，BSC と逐次合成の各処理時間の比較を行った．BSC の場合，合成時間，通信時間はともに 53% 短縮され，描画速度も 1.8 倍の高速化が実現できた．逐次合成の場合，通信時間は 75% 短縮され，描画速度も 2.5 倍の高速化が実現できた．

第6章　おわりに

本研究では実時間数値シミュレーションとシミュレーション結果の可視化処理の両方を実現する，超高速体感型シミュレーションシステムのプロトタイプとして，Scube の開発を行った．Scube の提案は，システム全体をコモディティ部品のみで構成し，複数系統設けたネットワークの冗長性を利用した通信のスケジューリングにより，シミュレーションと可視化処理の同時実行を目指すとともに，シミュレーションに必要となる通信の高速化を図っている．

第 3 章では，中規模コモデティクラスタによる実時間インタラクティブシミュレーションシステムの構築に向けて，2 つの検討項目についての予備評価を行った．まず，数値シミュレーション，並列可視化処理では，基本的な通信パターンは隣接ノード間通信であるが，同一スイッチ内のノード間通信とスイッチを多段結合してスイッチを跨いで行うノード間通信において生じる問題を明らかにした．ノード間通信に関しては，スイッチの多段結合によって接続されるノード間同士が同時に一斉送信を行うと，ノード台数に比例してネットワークの競合が生じる．このような競合を回避するためには，送信ノード数を削減する，ネットワークの要求仕様に応えうる送受信ノード選択が必要である．次に，並列可視化処理を行うにあたり，ボトルネックとなる要素を明らかにした．スクリーンサイズが 1024×1024 規模の 64 台の並列ボリュームレンダリングでは，総描画時間に重畳処理時間が占める割合が 60% ∼80% 以上であることが分かった．並列ボリュームレンダリングの高速化を図るためには，通信データ量の削減可能な合成アルゴリズムを提案し，通信時間および合成時間を削減することが必要である．

第 4 章では，Scube 上にコモデティ技術を用いた中規模 PC クラスタ向けの相互結合網 Three Quads の提案を行い，その主な特徴，定性的な評価について述べた．任意の 2 ノード間通信が 2 ホップで通信可能であり，通信経路が多く存在し，耐故障性，多様な網の埋め込み能力に優れていることを示した．また，数値シミュレーションに対して科学技術計算に広く使用されている行列の転置演算処理の全対全通信において，ネットワークの競合が生じることなく，高速に実現が可能であることを示した．

第 5 章では，第 3 章で明らかにした並列ボリュームレンダリングにおける重畳処理の高速化のために，通信データ量を削減し，通信時間を短縮する主軸優先

合成アルゴリズムを提案した．中間画像の通信量に着目し，最終画像を得るまでに要する描画時間における，データ通信に要する時間を短縮するために，サブボリューム間の重複関係を考慮した．従来の合成アルゴリズムと比較して，通信時間および合成処理時間の短縮が可能となり，描画速度の向上も可能となった．

謝辞

本研究を進めるにあたり，御多忙の中，丁寧な御指導と御助言を頂き，絶えず暖かく見守って下さった 富田 眞治 教授に深く感謝致します.

本研究を進めるにあたり，適切且つ心暖かな御助言，御指導を頂いた 森 眞一郎 助教授に深く感謝致します.

本研究を進めるにあたり，鋭く且つ心暖かな御助言，御指導を頂いた 中島 康彦 助教授に深く感謝致します.

本研究を進めるにあたり，異なる研究グループにも関わらず多くの御助言，御指導を頂いた 嶋田 創 助手， 三輪 忍 助手に深く感謝致します.

本研究を進めるにあたり，同じ Revolver グループとして有益なアイデアと御助言を頂い た本学修士課程 2 回生 岡村 大 氏，小松原 誠 氏，篠本 雄基 氏，本学学士課程 4 回生 野田 祐介 氏，吉田 智一 氏に深く感謝致します.

上記の皆様方をはじめ，常日頃の研究室生活にあたり，様々な面でお世話になった本学富田研究室の皆様に深く感謝致します.

参考文献

[1] Ogata, M., Kajihara, K., Kukukawa, T. and Terada, T.: An Evaluation of Communications Cost for Simultaneous Processing with Simulation and Visualization using an Image-Composition Device, ハイパフォーマンスコンピューティングと計算科学シンポジウム (HPCS), pp. 63–70 (2006).

[2] 助村俊一ほか: バーチャルマイクロスコープの制御手法の開発, 情報処理学会研究報告 99-HPC-66(HOKKE'97), Vol. 97, No. 37, pp. 25–30 (1997).

[3] S., D. et al.: RVSLIB:A Library for Concurrent Network Visualization of Large Scale Unsteady Network Visualization of Large Scale Unsteady Simulation, *SPEEDUP journal 11*, pp. 55–65 (1997).

[4] Groner, J., M.Lee, Martin, J., Moorhed, R. and Newman, J.: A Concurrent Visualization System for High-Performance Computational Simulations, *IEEE Visualization Poster Abstracts*, pp. 139–140 (2005).

[5] 富田眞治: 超高速体感型シミュレーションシステムの研究, 科学研究費補助金平成 16 年度基盤研究 (S) 新規採択課題の概要について[1)], 日本学術振興会, pp. 10–12 (2004).

[6] 吉村知晋, 吉良祐司, 森眞一郎, 中島康彦, 富田眞治: 汎用 GPU を用いた大規模可視化クラスタの構築, 第 33 回可視化情報シンポジウム講演論文集, Vol. 25-Suppl, No. 1, pp. 277–280 (2005).

[7] 緒方正人, 菊川孝明, 梶原景範: イメージ合成装置を用いたシミュレーションと可視化の並列処理における通信コストの評価, 情報処理学会研究報告, Vol. 2005-CG-120, pp. 7–12 (2005).

[8] トーマス L スターリング: PC クラスタ構築法, 産業図書 (2001).

[9] Boden, N. J. and et al.: Myrinet A Gigabit-per-Second Local Area Network, *IEEE Micro*, Vol. 15, No. 1, pp. 29–36 (1995).

[10] 天野英晴: 並列コンピュータ, 昭晃堂 (1996).

[11] 鈴岡節, 田邊昇, 中村定雄, 藤田純一, 小柳滋: 並列 AI マシン Prodigy の相互結合網の評価, 信学論, Vol. J71-D, No. 8, pp. 1496–1501 (1988).

[12] 朴泰祐, 佐藤三久, 宇川彰: 計算科学のための超並列クラスタ PACS-CS の概要, 情報処理学会研究報告, Vol. 2005-HPC-103, pp. 133–138 (2005).

[1)] http://www.jsps.go.jp/j-grantsinaid/

[13] 中嶋正之, 藤代一成: コンピュータヴィジュアリゼーション, 共立出版 (2000).

[14] K.Ma, J.Painter, C.Hansen and M.Krogh: Pallael Volume Rendering Using Binary-Swap Compositing, *IEEE Computer Graphics and Application*, pp. 59–68 (1994).

[15] Stompel, A., Ma, K.-L., B.Lum, E., Ahrens and Patchett, J.: SLIC:Scheduled Linear Image Compositing for Parallel Volume Renderin, *IEEE Symposium on Parallel and Large-Data Visualization and Graphics 2003*, pp. 33–40 (2003).

[16] Nonaka, J. and et al.: Hybrid Hardware-Accelerated Image Composition for Sort-Last Parallel Rendering on Graphics Clusters with Commodity image Comositor, *IEEE Symposium on Volume Visualization and Graphics 2004*, pp. 17–24 (2004).

[17] Moll, L. and et al.: Sepia:scalabel 3D compositing using PCI Pammete, *Proc. of the 7th IEEE Symp. on Field-Programmable Custom Computing Machines*, pp. 55–118 (1999).

[18] 森 眞一郎他: 大規模データの並列可視化を支援する FPGA 搭載 PCI カード, 第 1 回リコンフィギャラブルシステム研究会論文集, pp. 15–20 (2003).

[19] Molnar, S., Cox, M., Ellsworth, D. and Fuchs, H.: A Sorting Classification of Parallel Rendering, *IEEE Computer Graphics and Applications*, Vol. 14, No. 4, pp. 23–32 (1994).

[20] Takeuchi, A., Ino, F. and Hagihara, K.: An improved binary-swap compositing for sort-last parallel redering on distributed memory multiprocessors, *Parallel Computing*, Vol. 29, No. 11/12, pp. 1745–1762 (2003).

[21] Cavin, X., Mion, C. and Filbois, A.: COTS Cluster-based Sort-last Rendering:Performance Evaluation and Pipelined Implemetation, *IEEE Visualization 2005*, pp. 111–118 (2005).

[22] 住元真司, 久門耕一, 朴泰祐, 佐藤三久, 宇川彰: PACS-CS のための Ethernet を用いた高性能通信機構の設計, 情報処理学会研究報告, Vol. 2005-HPC-103, pp. 139–144 (2005).

[23] 富田眞治: 並列コンピュータ工学, 昭晃堂 (1996).

[24] Kadota, H. et al.: Parallel Computer ADENART-Its Architecture and Application-, Vol. ICS-91, pp. 1–8 (1991).

[25] 小国力, 村田健郎, 三吉俊郎, J.J. ドンガラ, 長谷川秀彦: 行列計算ソフトウェア, 丸善 (1991).

[26] Agarwal, R. C., Gustavson, F. G. and Zubair, M.: A High Performance Parallel Algorithm for 1-D FFT, *Proc. Supercomputing '94*, pp. 34–40 (1994).

[27] 高橋大介, 朴泰祐, 佐藤三久: PC クラスタにおける並列一次元 FFT のブロックアルゴリズム, 情報処理学会論文誌, Vol. 43, No. SIG 6, pp. 134–142 (2002).