修士論文

ボリューム可視化用並列アクセラレータ
VoViAc のアーキテクチャの検討と実現

指導教官：山崎 勝弘 教授

氏名：山崎 恭啓
学籍番号 6114940095–7

提出日：1996 年 2 月 15 日

立命館大学大学院理工学研究科情報システム学専攻

内容梗概

本論文では，ボリューム・グラフィクスを可視化するレイ・キャスティング処理を高速処理する，3 次元トーラス型の接続網を持つ VoViAc アーキテクチャについて述べる．スケイラビリティの高い接続形態を採ることで，大型計算機上で 3 次元シミュレーションされた大型ボリュームの高速な可視化からワークステーションレベルでの小型ボリュームの可視化まで広範な応用に対応することができる．

各 PE にはプロセッサ間に発生するデータ依存性を解消するため，独自のルータ回路 (VoViRo) を用いる．VoViRo は FPGA で構成したルータで，データを配信するプロセッサの周囲の 26 のプロセッサへ，高速に通信することを目的としている．データ移動する方向を X, Y, Z 軸の移動要素に分割し，パイプライン処理することによって高スループットを実現している．処理全般が浮動小数点演算であるため，プロセッサには TI 社の DSP を用いる．DSP は多くの特徴を持ち，VoViAc を実現する上でそれらが大いに貢献している．

並列性にはボクセル並列とピクセル並列と呼ばれる 2 つの並列性を組み合せることで，3 次元のデータ並列性を全て利用する．このため，ボリューム・グラフィクスを 3 次元インターリーブ分割し，各 PE のボリューム・メモリへ格納する．そして，ボリューム間で発生するレイ方向の依存関係を解消するため，3 次元トーラス上をリング状のデータ流が複数発生する．このデータ流を斜めに曲げることによって，任意方向からの可視化を可能にしている．

また，ボリューム可視化に求められる，補間処理，遠近法といった，これまで計算性能の点から後回しにされてきた技術についても，粗粒度インターリーブ分割と冗長法，対称 4 点法といった独自の方法を用いてオーバーヘッドなく処理可能であることを確認した．

目 次

1　はじめに

ボリューム・グラフィクスとは3次元空間をメッシュ状に区切り，その個々に色値や透明度の属性を配置したデータのことである．それゆえ情報量が多く，2次元の動画と同様に冗長であるが，ボリューム・グラフィクスには次のような魅力がある．

まず，幾何記述でしか表現されていない3次元画像に自由な物体を持ち込むことが可能になる．物体記述でしか記述できない3次元画像はどうしても冷たく単純なものになり，一時的に人間の目をごまかせてもすぐに飽きられてしまう．ボリューム・グラフィクス用の描画ツールが揃えば，グラフィクス・デザイナーはよりリアルな3次元の彫刻物を作り出すことができる．第2に，画像の複雑さによらない処理ができるようになる．これによって，リアルタイム性を保証しやすくなり，ディスプレイのような秒あたりのフレーム数を保証することができ，ボリューム・グラフィクスを通してインタラクティブにやりとりできるようになる．第3に，ボクセルに適当な属性値を付加することによって，全ての3次元系をシミュレーションすることが可能である．ボリューム・データはボクセル値にスカラ値を割り当てればスカラ場を表現し，ベクトルを割り当てればベクトル場になる．しかし，ベクトル場は直接は可視化できないので，ボクセル値は黒のスカラ値で書かれた矢印や色で表現される．また，物理法則では物質まで場で表現されるので，直接場として扱うのは効率的かどうかは別として，いかなる3次元系もシミュレーションできるようになる．とりわけ，流体計算は3次元シミュレーションとして少しでも大きく少しでも高い精度で計算したい分野で，装置の熱対流や気象予測，ヘリコプターのプロペラ周囲の気流，船の船頭の水流といったように多くの分野で応用されている．それゆえ，これを実現する意義は大きく，ボリューム・グラフィクス・アクセラレータはポリゴングラフィクス・アクセレータの次のスタンダードになると考えられる．

ボリューム・グラフィクスには専用のアーキテクチャを構築しなければならない．通常の逐次型計算機ではデータを1次元のメモリに格納しているので，折りたたんで格納したデータでは弱点となる視線方向が必ず生じてしまうからである．

ボリューム・グラフィクスはこれまで医用グラフィクスとして手術のシミュレーションに用いられてきた．また，計算能力が低かったため色情報として2値のものを対象としたものが多い．それに対して，最近登場したアーキテクチャは科学技術計算の結果を表示するものを対象にするものが多く，計算能力が高くなってきたこともあり，研究がさかんになっている．

ボリューム可視化専用並列計算機としては，PARCUM-II[9]，ReVolver[3][4][5]，CUBE-4[6][7][8]等がある．PARCUM-IIは投射面に対する並列性(ピクセル並列性)を用いたもので，ボリュームの2^3のデータを同時に取り出すことができるメモリ構造を使用している．ReVolverはボクセル並列性を用いて並列化したもので，依存関係のあるレイ・キャスティング処理をパイプライン状のプロセッサ結合で処理することによって並列実行する．Cube-4はスキュースキーム構造を持つボリューム・メモリに対して同時にレイ上のデータをアクセスし，調停機構を用いて一番手前のボリュームを探し出す．どの方式でも長所と短所が存在し，決定的なものは存在しない．

また，これからの数年間はリーズナブルなコストパーフォーマンスを出すために，画像の冗長性を利用して省略するといった計算の省略を行なわなければ，ボリューム処理にかかる計算量の多さに対処することは難しい．例えば，MPEGのような動画も画質は完全ではないながらもそれ

なりに実用的である．しかし，新たなハードウェアを入れるとなれば，ソフトウェアを導入するだけで表示が可能になるソフト MPEG のような付加機能のようなわけにはいかず，完全ではないにしても相当完全に近くなければ受け入れられないだろう．本研究でも計算量削減のため不透明体もしくはあるしきい値以下の物体より奥の画像計算を省略することを試みた．

本論文では以上のような条件を満すものとして，3 次元トーラス型のアーキテクチャを提案する．このトポロジは 2 次元トーラス以上ハイパーキューブ以下で，6 次元以上の 2 進ハイパーキューブは全て 3 次元トーラスを内包している．2 次元トーラスでも 1 次元拡張を行えば 3 次元トーラスになることを考えると，本アーキテクチャで行う並列計算をシミュレーションできる並列計算機は多いと考えられる．

本アーキテクチャではボリュームにおける 3 次元のデータ並列性のうち，ボクセル並列性とピクセル並列性の両方を併用する．前者はこのトポロジー上をレイがパイプライン状に回転することによって行ない，後者は投射面に対して複数のピクセルを放射することによって実現している．ピクセル並列性は偏った画像の場合に負荷均衡が悪くなる可能性があるので，各プロセッサには 3 次元インターリーブした画像を配置する．これによって同時処理する単位がキューブ状になるため，任意方向からの可視化が可能となる．

PE(Processing Element) は，プロセッサとルータを 1 セットにして構成する．高速な浮動小数点演算器が必須なため，プロセッサには DSP を用いることにした．各 PE を接続するルータ内部にはリング状のデータ流があり，リング上のコンポーネントには X, Y, Z の軸ごとに別々に動作するスイッチングボックスを用意して，現在データを持っているプロセッサから周囲の 26 のどの PE にもデータを送れるようにしている．また，動作速度を速くするために，これらのコンポーネントをパイプライン動作するように設計した．これによって，スループットを 3 倍近くにすることが可能である．各 PE 同士を接続するルータには FPGA を用いる．これによってルータの細かい部分での問題点をフィードバックしやすくなる．また，レイ・キャスティングと 3DDDA のような独立した動作は最初はプロセッサで行う予定だが，最終的にルータに転送されることになるので，FPGA の容量が許せば FPGA で高速処理することも可能であり，FPGA を ASIC 化する意義も高くなるだろう．

本論文は 2 章でボリューム・グラフィクス・アーキテクチャに必要とされる処理をグラフィクスの視点から見ていく．3 章では 2 章で述べられた必要条件を，私が提案するアーキテクチャ VoViAc ではどのように解決しているのかを説明していく．4 章では DSP 内でのソフトウェア処理について説明する．5 章では VoViAc の処理概念を実現する上で行った設計・製作上の工夫や，工程，及び処理能力の初期評価を行う．

2 ボリューム可視化アルゴリズム

ボリューム可視化は，ボリュームデータが多値の場合には，レイ・キャスティング法を用いてレイをボリューム中でスキャンすることによって行う．この結果はX線写真のようになるので，一般的には視点から不透明体までのZ値をもとにシェーディングを行う．本論文ではシェーディングは別のハードウェアで行うことを前提として話を進めていく．

2.1 レイ・キャスティング

レイ・キャスティング[1][2]は半透明体を光が通り抜ける時に起こる現象を真似た手法で，ボリューム・グラフィクスからX線写真のような画像を得ることができる．図1にレイ・キャスティング処理した画像の例を示す[1]．任意の面でクリッピングすることによって，切断面を表示することが可能で，このような演算は，従来，手術時のシミュレーションを目的として多く用いられてきた．コンピュータの科学技術計算能力が向上し，3次元計算に耐えうるものになってきたため，その表示に対しての必要性が今後出て来るものと思われる．

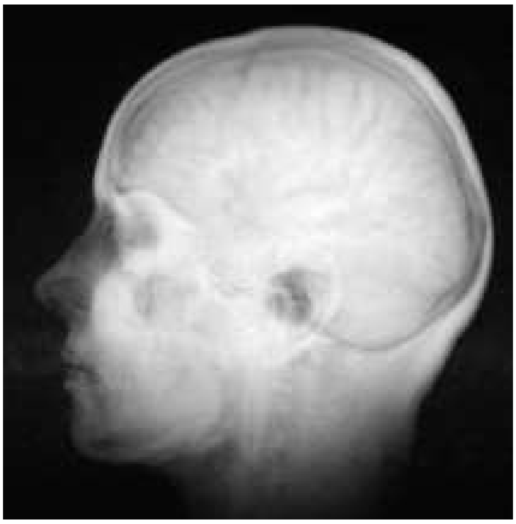


図1: レイ・キャスティングの例

この手法ではボクセル個々を光に対するフィルタと考える．図2に示すように，各フィルタは不透明な割合だけその点のカラーシェーディング値にし，透明な割合だけ後ろからの光量にするように働く．

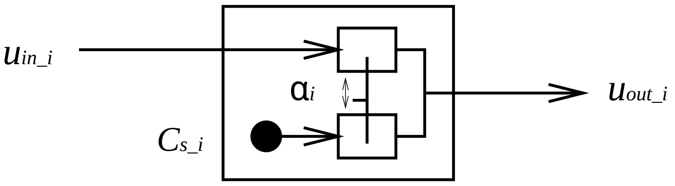
個々のフィルタの作用を式にすると，

$$u_{out\_i} = (1-\alpha_i)\,u_{in\_i} + \alpha_i\,C_{s\_i} \tag{1}$$

となる．このフィルタを図3のように視線上にならべて作用させる．

---

[1] ftp://omicron.cs.unc.edu:/pub/projects/softlab.v/CHVRTDに所在する"Chapel Hill Volume Rendering Test Data Set Volume I"に含まれる3dheadを用いてSPARC STATION 20上でレイ・キャスティング計算を行なった．



α_i ： ブロック i の不透明度
C_{s_i} ： ブロック i のカラーシェーディング値
u_{in_i} ： ブロック i の入力値
u_{out_i} ： ブロック i の出力値

図 2: 各ボクセルの働き

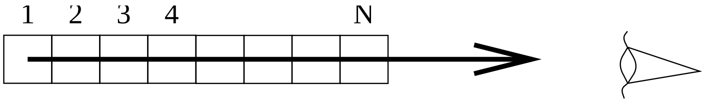


図 3: フィルタの連結

この作用を式にすると，

$$u_{out\_0} = 0 \tag{2}$$

$$u_{out\_i} = (1-\alpha_i)\, u_{out\_i-1} + \alpha_i\, C_{s\_i} \qquad (1 \leq i \leq N) \tag{3}$$

この演算は，$i = n+1$ の演算は $i = n$ の演算が終了してからでないと行なえない．つまり，u_{i+1} は u_i に依存している．この演算順序だと，奥から手前に計算が進んでいくことになるため，たとえ一番手前のボクセルが不透明なボクセルであったとしても，奥から全てのボクセルに対して計算しなければならない．

そこで，最終的に得たい値である u_{out_N} に着目してみる．

$$\begin{aligned}
u_{out\_N} &= (1-\alpha_N)\, u_{out\_N-1} + \alpha_N\, C_{s\_N} \\
&= (1-\alpha_N)((1-\alpha_{N-1})\, u_{out\_N-2} + \alpha_{N-1}\, C_{s\_N-1}) + \alpha_N\, C_{s\_N} \\
&= (1-\alpha_N)(1-\alpha_{N-1})\ .....\ (1-\alpha_3)(1-\alpha_2)\,\alpha_1\, C_{s\_1}\ + \\
&\quad\ (1-\alpha_N)(1-\alpha_{N-1})\ .....\ (1-\alpha_3)\,\alpha_2\, C_{s\_2}\ + \\
&\quad\ .... \\
&\quad\ (1-\alpha_N)\,\alpha_{N-1}\, C_{s\_N-1}\ + \\
&\quad\ \alpha_N\, C_{s\_N}
\end{aligned} \tag{4}$$

こうすると，(4) はそれぞれのカラーシェーディング値が観測者に対してどれだけの影響を与えるのかという値を加算していると解釈することができる．つまり，C_{s_N-1} は観測者に対して $(1-\alpha_N)\,\alpha_{N-1}\,C_{s\_N-1}$ だけ影響を与えていると解釈できる．

そこで，式 (4) を次のような値を用いて表すと，

$$\begin{aligned}
\alpha_{mul\_N} &= 1 && && (5)\\
\alpha_{mul\_i-1} &= \alpha_{mul\_i}\,(1-\alpha_i) && (2 \leq i \leq N) && (6)\\
u_{sum\_N} &= \alpha_N\,C_{s\_N} && && (7)\\
u_{sum\_i-1} &= u_{sum\_i} + \alpha_{mul\_i}\,\alpha_i\,C_{s\_i} && (2 \leq i \leq N) && (8)
\end{aligned}$$

この方法の場合，後ろの項から必要な項までを計算し，不透明体より後ろの加算値は全て 0 になるので計算しないで済む (図 4)．ただし，この方法では u_{sum_i}だけではなくα_{mul_i}も通信しなければならない．

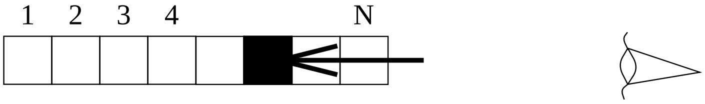


図 4: 逆方向の演算

2.2　ボリューム分割の分類

ボリューム・グラフィクスはデータ量が多く，個々に対する処理はそれほど複雑ではないので，オブジェクト並列性は少なく，データ並列性を多く含んでいると考えられる．ボリューム・グラフィクスの並列性はボクセル単位まで及んでいるので，ボリューム・グラフィクスの 3 次元の並列性のうち，どこまでを並列処理に利用するかによって，ボクセル分割，ピクセル分割，ボリューム分割 (図 5参照) に分類することができ，それぞれ，1 次元，2 次元，3 次元の分割であると考えることができる．

2.2.1　ボクセル分割

ボリューム・グラフィクス中を通るレイが交差するボクセル個々に対する並列処理で，ReVolver [3][4][5] や CUBE-4 [6][7][8] がこれに相当する．

ReVolver ではプロセッサをパイプライン状に配置し，レイを時間間隔を空けて注入することによって奥行き方向の処理を並列に行う．データ並列性を 1 次元しか利用しないため，補間処理や遠近法が高速処理可能であるという特長がある反面，ボリュームが大きくなった時に 1PE にかかる負荷が大きくなることや，任意方向からの可視化を実現するためにメモリの 3 重化を行わなければならないといった難点もある．

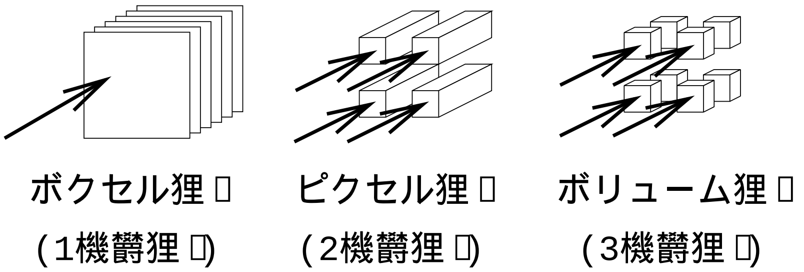


図 5: ボリューム分割法

CUBE-4 は，図 6に示すスキュースキーム構造を持つボリュームメモリ CFB(Cube Frame Buffer) が特徴的で，図で示される斜め方向の面が特定のプロセッサに割り振られる構造をしている．計算する前に，投影する画像の横方向と奥行き方向を辺とする面を取り出して，Template と呼ばれる lookup-table に入れる．このことによって，1 ボクセルより小さい間隔のレイが同じボクセルを複数回アクセスしたり，8 点補間によって複数回アクセスされるのを防ぐ．また，Template に入った面データは，視線上の全ボクセルを一度に取り出すことができ，観測者に最も近い不透明ボクセルを探し出す機構に通される．このアーキテクチャの弱点は，図 6 に描かれている面と，Template を構成する面が一致した場合に起るアクセスの集中である．しかし，このような事態は視点に制約を与えることで逃れることができ，それができない場合には，2 次元画像を 90 度回転するハードウェアを用意することで逃れられる．

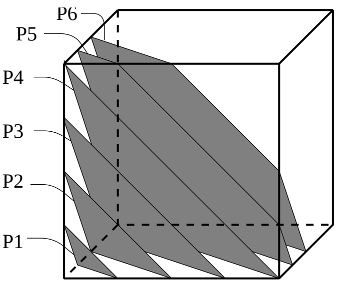


図 6: CUBE-4 の CFB

この分割方法は，3 次元ある並列度を 1 次元しか用いていない．これにより，両アーキテクチャは残った 2 次元の自由度を任意方向からの可視化を高速に実行するために用いている．

2.2.2 ピクセル分割

投射面のピクセル個々が独立に処理できることを利用した分割法で，レイ・トレーシングにおけるレイの発生方法に似ているので，ボリュームレイ・トレーシングとも呼ばれる．

PARCUM-II [9] はこの並列性を利用したアーキテクチャで，2^3の小ボリュームを XYZ方向に積み重ねた構造のボリュームバッファを持ち，どの小ボリュームも 8 ボクセル同時に取り出すこ

とができる．取り出す単位が 2^3 のため，任意方向からの可視化は完全ではない．

この分割では1次元分割より多くの並列度を抽出することができるが，2つの問題が出てくる．1つは任意方向からの可視化が難しいことである．どの方向から見ても同じように並列度を出すことができ，しかもハードウェアで用意したメモリ並列度がそのままピクセル並列度になる分割方法は考えられていない．2つめは物体の存在位置によっては負荷バランスが悪くなるという問題がある．この並列性を用いるため，当研究室でのレイ・トレーシング処理における画面分割法で最も理想に近い負荷均衡性を出したインターリーブ分割[11]を3次元に拡張した3次元インターリーブを用いることにした．

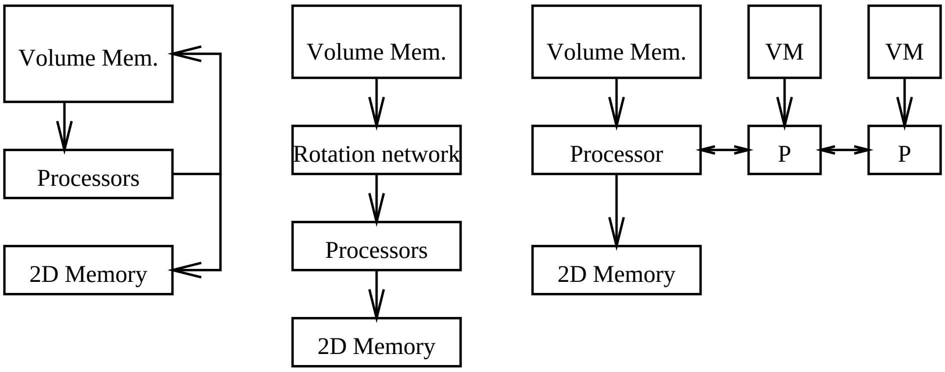
2.2.3 ボリューム分割

3次元のボリュームを小さな立方体のサブ・ボリュームに分割し，各サブ・ボリュームで画像を投影し，得られた画像を視点に近い画像のZ値の優先度を上げてマージする．不透明ボクセルのみを対象とする場合のように，容易にマージできる場合には高い並列性を発揮するが，半透明ボクセルを扱う場合には実現が難しく，複雑な処理を必要とする．

2.2.4 本アーキテクチャでの方法

本研究ではボクセル分割とピクセル並列性を並用する．ボクセル並列性を用いることができない部分でピクセル並列性を用いる．これにより3次元のデータ並列性全てを用いることが可能で，高いスケーラビリティを提供する．これを実現するために3次元インターリーブが不可欠で，ボクセル並列性に対してはどの方向からも視線上のボクセル個々が別のPEに存在するように働き，ピクセル並列性に対しては負荷均衡性を提供するため，両者とも密接な関係がある．

2.3 任意方向からの可視化



(a)酌機妾な□松　(b)ネットワークによる□松　(c)縮軒による□燦□恵

図7: 任意方向からの可視化

任意方向からボリュームグラフィクスを可視化するために，初期のハードウェアでは逐次的に

1 ボクセルずつ回転する方法が採られている (図 7(a))．リアルタイムに実行するには程遠かったため，あくまで付加機能として用いられてきた．これをハードウェアで高速実行できるようにするためには，どの方向からも同じようにメモリアクセスできる構造がなくてはならない．そのため，XYZの立場を同じにした対称的なメモリ構造を用いる必要がある．本研究では初期の段階でネットワークによるボリュームの回転を行うことを考えた (図 7(b))．この時，クロスバー網から必要とされる網だけを残し，分割できる部分を分割するという方法で回転網を形成した．必要とされる残ったネットワークから回転変換向けのトポロジーを見つけることも目的としていたが，このアプローチでは回転網は集中的で非常に複雑になるということが分かった．

そこで，ボリュームを回転するのではなく，逆に観測者が移動するという方法を考えた．これはレイ・キャスティング計算において，依存するデータを通信することによって並列実行する方法 (図 7(c)) である．この方法では 3 次元メッシュ状に並べられた PE が周囲の 26PE へ通信できれば，ボリュームが任意方向から可視化できることが分かった．周囲の 26PE へ直接データを送れるリンクを用意するのは困難なので，ルーティングしてリンクを削除することにした．このようにして 3 次元トーラスが有効であると考えて，各種の処理が実行できるように模索してきた．

本研究では依存関係の解消で処理する方向で問題解決に向かったが，実際はボリュームを回転するのと観測者が移動するのは表裏一体の関係にあって，3 次元トーラス結合網で観測者は移動しないで，ボリュームを必要とされる PE へ転送するという方法でも可能であると考えられる．ただし，この方法では必要とする PE へそのデータを必要としない PE が送らなければならないので，アルゴリズムが複雑になると考え，依存関係の通信による解消を用いることにした．

任意方向からの可視化を実現するもう 1 つの方法として ReVolver で行っているボリューム・メモリの多重化がある．通常，任意方向からの可視化は主軸に対して 45 度以下の任意角度の可視化と 3 主軸からの可視化を組合せて行う．そこで，3 主軸からの可視化が高速実行できるようにボリュームメモリを 3 重化している．ReVolver では 3 次元の並列性のうち，1 次元しか並列処理に用いていない．残りの 2 次元の並列性は 45 度以下の任意角度での可視化に用いている．この方法ではボリューム表示のために 3 倍のボリューム・メモリを必要とする．そのため，このことは超高速処理用途に限定されてしまうという欠点となる．

2.4 シェーディング

ボリューム・グラフィクスを可視化する時，レイ・キャスティング計算をしたままだと，X 線写真のように物体を投射した場合の陰を見ているように見える．これは，物体の表面に対する考慮がなされていないからで，シェーディングは表面に対する計算を行う．

ボリューム・グラフィクスをシェーディングする際，濃度変化の激しい部分を表面として扱うために，グラディエント・ベクトルを求める．そして，このベクトル値に対して光源との内積をとることによって (レイ・トレーシングの Phong シェーディング [10])，光が物体の表面に対して示す効果を再現することができる．グラディエント値を求めるにはそれなりのコストを必要とし，ベクトルの求め方によって，Z-buffer gradient 法 [1] や graylevel gradient 法 [1] がある．graylevel gradient 法によるシェーディングの例を図 8に示す．この図は，図 1と同じ 3 次元画像を用いて，6 点近傍によるグラディエント計算による結果である．

図 8: graylevel gradient 法によるシェーディング例

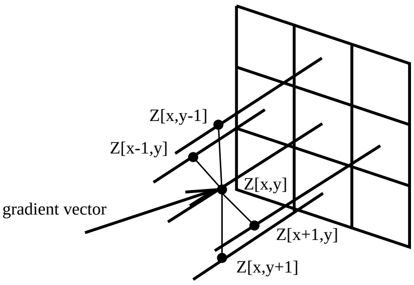


図 9: Z-buffer gradient 法

Z-buffer gradient 法は，レイが物体にヒットした Z 値を隣りの Z 値と比較することによってグラディエント・ベクトルを求める (図 9)．比較的コストが小さいので，実装は難しくないが，半透明体の中に物体がある場合のように，複数の表面が存在する画像では，表面のうちの 1 つしか表示できない．

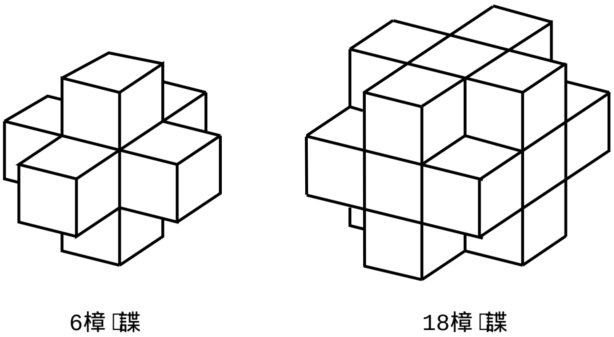


図 10: graylevel gradient 法

一方，graylevel gradient 法はリアルシェーディングとも呼ばれ，グラディエント値を求めるのに近傍のボクセルの濃度差から求める (図 10)．したがって，レイ・キャスティング計算時に graylevel gradient 演算も行い，それらの値を重み付け加算して，光量とする．この方法は複数の半透明な面に対しても対処できることが特長で，リアルな画像が得られる．反面，計算に必要なボクセルが非常に多いため，スキャンする際に近傍のボクセルを再利用しないと速度の点で不利になる．

2.5 補間処理

ボリューム・グラフィクスは小さな画像でも大量のメモリと計算能力を必要とするため，画素のない部分での値を補間して，少ないボクセルを十分に利用しなければならない．

補間処理には，8 点 (Trilinear) 補間 [1][2] と 4 点補間がある．ボクセルが格納されている空間の座標軸に対して斜め方向に視線がある場合，サンプリングポイントがボクセルの存在する点にちょうどこなくなる．この時，サンプリングポイントに最も近い 8 点を用いて補間するのが 8 点補間で，主軸 (3 座標軸の中で，視線との角度が最も鋭角になる軸) に対して垂直な平面上で，最も近い 4 点を用いるのが 4 点補間である．8 点補間の方が結果が良いが，レイ・キャスティングを行う場合は，奥行きの情報がそれほど画像に影響してこないので，4 点補間でもそれなりの結果は得られる [3]．いずれも線形補間であり，非線形な方法も補間式を置き換えることで実現可能であるが，それに対するコストが大きくなるため速度を求める場合には不利である．

また，補間処理は隣接点間の値を求める処理なので，本アーキテクチャにおける 3 次元インターリーブと概念が相反するものになってしまう．そのための対処方法として，粗粒度インターリーブ分割と冗長法，及び投射面補間法について説明する．

2.5.1　8 点 (Trilinear) 補間法

ボクセル間での値 (C) を図 11に示す近傍の 8 点 (P1 ~ P8) を用いて補間を行う．

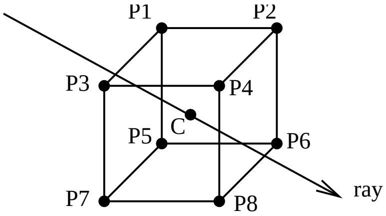


図 11: 8 点 (Trilinear) 補間法

式

$$f(x,y,z) = a_0 + a_1\,x + a_2\,y + a_3\,z + a_4\,xy + a_5\,yz + a_6\,zx + a_7\,xyz \tag{9}$$

に対して，図 11における P1 ~ P8 の座標値を代入すると，a_0 ~ a_7に関する 8 個の連立方程式ができる．この方程式を解き，a_0 ~ a_7を求め，元の式に求めたい点である C の座標値を代入する．

この方式の利点は正確さである．しかし，この事に対する代償は大きく，1 点のボクセル値を得るのに周囲の 8 点のデータを得る必要があり，さらに連立方程式を解かなければならない．この 8 点のデータが別々のプロセッサに存在する場合，通信にかかる時間は致命的なものになる可能性が高い．つまり，8 点補間のために必要な通信の量が非常に少ないか，同一データを複数のプロセッサに持たせる冗長なデータ配置にしなければならない．本アーキテクチャでは，粗粒度インターリーブ分割と冗長なデータ配置を組み合わせることで，8 点補間を実行可能にしている (2.5.3章 参照)．

2.5.2　4 点補間法

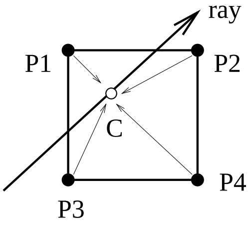


図 12: 4 点補間

この方法の特徴は，奥行き方向での補間を行わないことである．つまり，奥行き方向のサンプリング点の間隔は，主軸に対して角度がつけばつくほど広くなり，観測者からは奥行きが縮まったように見えることになる．しかし，この事は実際に投射してみるとそれほど大きな問題になら

ない事が確かめられている [3]．投射という処理自体が投射面に対して横方向は画像に大きく影響するのに対して，垂直方向は情報が失われてしまうからである．

また，主軸に対して横方向のデータアクセスが一定になることは，アーキテクチャに対して自由度を与える．ReVolver ではプロセッサをパイプライン状に配置してピクセル並列性を用いて高速処理を行う．パイプライン上の各プロセッサは，視線に対して垂直方向の面データを保持しているので，4 点補間で同時に必要な 4 点のデータを高いバンド幅で収集することが可能である．

2.5.3 粗粒度インターリーブ分割と冗長法

補間処理は近隣のピクセルを同時に必要とし，インターリーブ分割は近隣のピクセルを別プロセッサに割り当てることによって高速化する方法である．これらは相反する概念で，共存は難しい．そこで，インターリーブ分割の効果を弱くする粗粒度インターリーブ分割と冗長法を組み合せる方法を考えた．

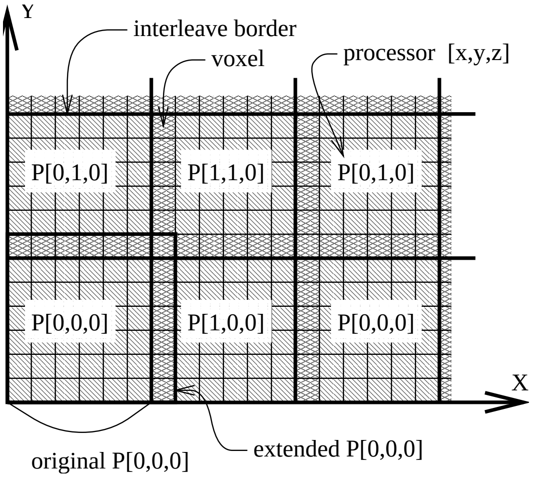


図 13: 粗粒度インターリーブ分割と冗長

図 13に示すように，粒度を高めたインターリーブの境界にあたる面が冗長になるように配置することによって，補間のターゲットとなるボクセルが複数のプロセッサにまたがるのを防ぐ．この冗長性によってどのくらいのボクセルが必要なのかを表 1に示す．

この表の増加率が冗長性を持たせない場合を 1 としたボリュームメモリの増加率である．これから分かるように，インターリーブブロックの幅が小さい場合，メモリが大量に必要となる．実用的なのは 4 から 16 の範囲と考えられる．また，注意しなければならないのは，平行レイで間隔が 1 ボクセル幅の場合，あるブロックから出て行くレイがあれば，必ず入って来るレイがあるということである．つまり，粗粒度インターリーブ分割でもプロセッサの負荷は均衡になる．

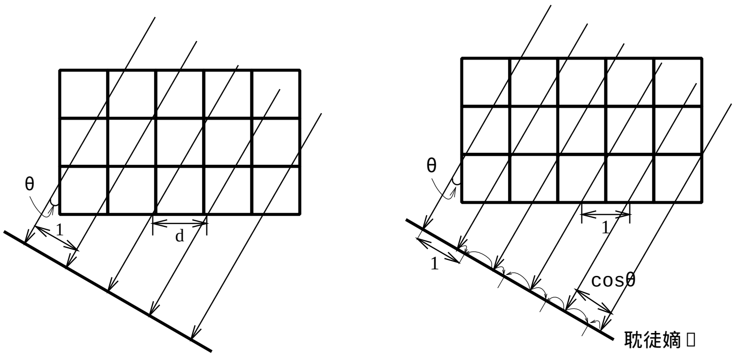
2.5.4 投射面補間

投射面補間とはボリュームグラフィクスを斜め方向から見る場合，サンプリング点で補間処理を行なわずに，投射面において補間する方法である．

表 1: インターリーブ度と冗長度

インターリーブ幅	非冗長時ボクセル数	冗長時ボクセル数	増加率
n	n^3	$(n+1)^3$	$\frac{(n+1)^3}{n^3}$
1	1	8	8.00
2	8	27	3.38
4	64	125	1.95
8	512	729	1.42
16	4096	4913	1.20
32	32768	35937	1.10

ターゲットとなるレイの主軸に対して斜め方向に見る場合，図 14のようなレイの放射が起き，d の間隔は$\frac{1}{cos\theta}$となる．



(a)レイ□□が１の血牙　　(b)レイ□□がｃｏｓθの血牙

図 14: 投射面補間

つまり，特に細粒度インターリーブ分割を行った場合に負荷バランスが悪くなる可能性がある．そこで，図 14(a) のように d の間隔を 1 にすると，全てのプロセッサの負荷が均等になる代りに画像に伸長作用が働いてしまうので，図 14(b) のように面で補間処理を行い圧縮する．この時の最大圧縮率はボリュームの角から見る場合で$\frac{1}{\sqrt{3}}$である．これは，一辺の長さが 1 の立方体の頂点と反対の頂点との距離が$\sqrt{3}$であることから来ている．また，斜めから見れば見るほどレイの数を増さなければ画像自体が小さくなってしまうということや，2 次元補間用のハードウェアを用意しなければならないという欠点もある．しかし，この事を考えてもなおボクセル個々で補間処理を行わなくても良いので，全体としての速度はボリュームに対する補間法よりも高速になると考えられる．

2.6　遠近処理

これまで，本論文ではレイは全て平行レイであるとの前提で高速化手法や対処法を述べてきた．しかし，画像が大きくなってくると遠近法を取り入れ，よりリアルな画像が求められてくる．アーキテクチャが平行レイの特性を利用した同期を行っているため，遠近法は対処するのが本来難しい．その中で，VoViAc に対して良い結果が得られる可能性の高い対称 4 点法を紹介する．ボリューム面に垂直な線に対して，等角度の 4 本のレイを同時処理する方法で，軸に対称になるように選ばれた 4 本のレイは，プロセッサから見ると平行レイの場合と同じ振舞いをすることを利用したものである．計算機の台数が増すと効果が薄くなってしまうという欠点はあるものの，プロセッサ間でコンフリクトが生じないので円滑な並列処理が可能である．

2.6.1　対称 4 点法

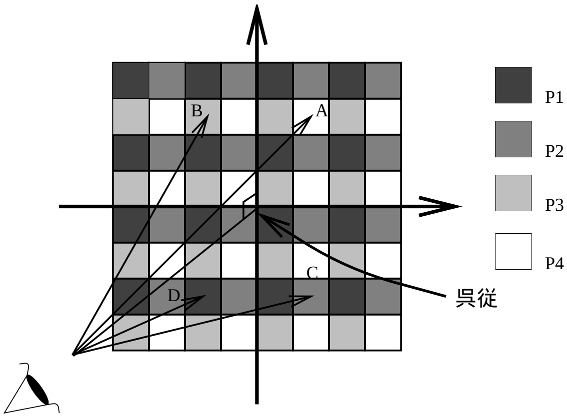


図 15: 対称 4 点法

この方法では，図 15に示す，軸に対して対称な 4 点を同時処理する．

これらの 4 本のレイを A,B,C,D とすると，B がアクセスするプロセッサは A を X 方向に-1 だけ並行移動したレイがアクセスするプロセッサと同じプロセッサをアクセスしながら進む．同様に C,D も A を並行移動したレイと同じアクセスをし，結局 4 本の並行レイを扱っているのと同じになる．本来，このアーキテクチャは並行レイに対してはプロセッサ間でのコンフリクトなく処理を進めることができるので，高速処理が可能になる．このことを以下に数学的に証明しておく．

プロセッサ間でのコンフリクトが生じないことの証明

図 16は図 15の座標軸を$+X$方向，$+Y$方向に 0.5 ボクセルずつずらしたものである．この時，レイ A のボリューム面での座標を (x_0, y_0) とすると，レイ A の Xおよび Y座標 (X_A, Y_A) は Z_Aによって次のように表される．

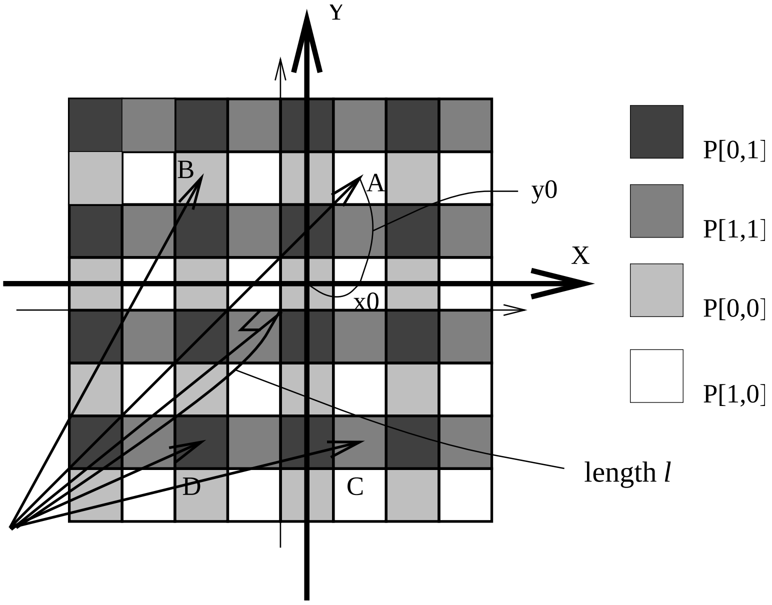


図 16: 対称 4 点法

$$X_A = round\left(\frac{x_0\,(Z_A+l)}{l}\right) \tag{10}$$

$$Y_A = round\left(\frac{y_0\,(Z_A+l)}{l}\right) \tag{11}$$

実はこのこと自体は Z の関数であることの確認で，対称なので B,C,D の座標 (X_B, Y_B),(X_C, Y_C), (X_D, Y_D) は (X_A, Y_A) を用いて次のように表される．

$$\begin{aligned}
X_B &= -X_A - 1\\
Y_B &= Y_A\\
X_C &= X_A\\
Y_C &= -Y_A - 1\\
X_D &= -X_A - 1\\
Y_D &= -Y_A - 1
\end{aligned}$$

このことからアクセスされるプロセッサ番号は，$(x+1)mod2 = (-x-1)mod2$ であることを用いて，

$$\begin{array}{rrrrrrl} A & : & P[& X_A & mod\,2\,, & Y_A & mod\,2\,] \\ B & : & P[& X_B & mod\,2\,, & Y_B & mod\,2\,] \\ & & =P[& (X_A+1) & mod\,2\,, & Y_A & mod\,2\,] \\ C & : & P[& X_C & mod\,2\,, & Y_C & mod\,2\,] \\ & & =P[& X_A & mod\,2\,, & (Y_A+1) & mod\,2\,] \\ D & : & P[& X_D & mod\,2\,, & Y_D & mod\,2\,] \\ & & =P[& (X_A+1) & mod\,2\,, & (Y_A+1) & mod\,2\,] \end{array} \tag{12}$$

となることが分かる．ここでのプロセッサ番号は 2 次の matrix で示している．これから，Z が同じ場合，同じプロセッサに重複しないことが分かる

しかし，これだけではプロセッサ間で通信できるのか保証されないので，この点についてレイ A を $(+1,+1)$ だけ平行移動したレイについて考えてみる．これは式 (12) の A に関する式の X_A，Y_Aに$+1$ すれば良い，

$$A(+1,+1) \;\;:\;\; P[\,(X_A+1)\;\;mod\,2,\;(Y_A+1)\;\;mod\,2\,] \tag{13}$$

これはまさしく P_Dのアクセスするプロセッサ番号と同じである．同様に，

$$\begin{array}{rcrrrrl} A(+1,0) & : & P[& (X_A+1) & mod\,2, & Y_A & mod\,2\,] \\ A(0,+1) & : & P[& X_A & mod\,2, & (Y_A+1) & mod\,2\,] \end{array} \tag{14}$$

これらは B,Cのものと同じである．(証明終り)

よって，4 本までの同時処理であれば，平行レイと同じように扱うことができる．

3　ハードウェア構成

3.1　全体構成

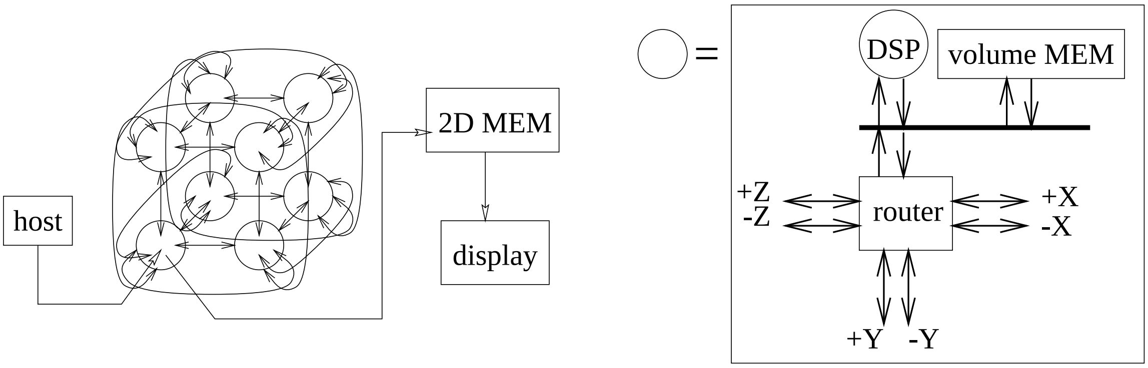


図17: VoViAcの全体構成

8台構成のVoViAcの全体構成を図17に示す[12]．VoViAcは3次元トーラス型の接続網を持っていて，このことは64台や512台に台数が変化しても変わらない．各PEはプロセッサとボリューム・メモリ，そしてルータで構成される．計算に必要とされる処理全般が数値演算処理であることから，プロセッサにはDSP(Digital Signal Processor)を用いた．本研究で用いたTI社のTMS320C31[13]は，浮動小数点演算用のプロセッサで，特徴は，

- 積算器と加算器を同時に使用可能なパラレル・インストラクション
- 1K word × 2(2K byte × 2)の内部メモリ
- サーキュラー・アドレッシング
- 2つの独立したタイマ
- 多様な分岐命令
- 条件代入命令
- プログラマブル・ウェイト・コントローラ
- ブート・ローダ機能

である．パラレル・インストラクションは命令フィールドの中に2命令を持つ命令で，積算器と加算器を同時に使用可能である．インクリメント・アドレッシングが使用できるので，ディジタル信号処理に求められることの多い積和演算を単一命令で実現することができる．内部に1K wordの1サイクルに2回ずつアクセス可能なメモリ2つ持っていて，うまく制御することで，2回のデータ・フェッチと命令フェッチ，DMAアクセスをウェイトなしにアクセスできる．少なくとも2K wordのメモリはほとんどのアルゴリズムで命令フェッチを見えなくしてしまうことができる．

サーキュラー・アドレッシングは，ディジタル信号処理で必要とされることの多いFIFOを高速に実現することが可能で，パラレル・インストラクションと組み合せて用いることで，単一命令でFIFOのリードやライトを行うことができる．VoViAcではリンクからのデータ入力を一時保持するのに用いる．2つの独立したタイマはソフトウェアでは使用しないが，その1つをFPGAの動作クロックを作るのに用いる．また，TMS320C31は遅延分岐，ブロック・リピート命令，単一リピート命令，条件付きデクリメント分岐，コール命令といったほとんどの分岐構造を持っている．命令実行がパイプライン化されているため，単純な分岐命令は4命令サイクル必要とするが，遅延分岐，ブロック・リピート，単一リピート命令を用いることで，分岐ペナルティをなくすことが可能である．DSPの場合，リアルタイム制御に使用されることが多く，単純な繰り返し演算が多いため，分岐にはさまざまなものを用意している．しかし，このことは，高級言語にとってはあまり良いことではない．プログラム制御命令を最適化するオプティマイザを作るのは難しく，単純なものしか使用できない．そのため，高級言語を用いていてもクリティカルな部分はアセンブリ言語で書かれる．条件代入命令も分岐構造を最適化するのに用いられる．通常，実行される全体の分岐命令数は最適化してもほとんど変らない．この命令を用いることで，分岐命令を削減することが可能である．プログラマブル・ウェイト・コントローラは，各PEの外付け回路を少なくするのに非常に役立っている．DSPから見て，EEPROMは非常に遅く，5ウェイト必要で，S-RAMは0ウェイト，FPGAは回路設計によってウェイト数が変わる．これらのウェイト数はプログラムから変更することも可能だが，変更には時間がかかるので，外部のアクノリッジ信号によっても変更可能である．EEPROMは起動時にのみ使用するので，初期段階では5ウェイトでプログラムをロードする．外部のアクノリッジをアドレスの23番ピンに接続し，ゴーストを出してプログラムから0ウェイトか，プログラムしたウェイト数にするかを選択する．つまり，アドレスの前半部分でアクセスすれば，負論理信号であるアクノリッジがかかり0ウェイトになり，後半でアクセスすれば，アクノリッジがかからないのでプログラムしたウェイト数になる．この方法でウェイト・コントロール回路をなくした．また，TMS320C31にはブート・ローダ機能があるので，これを利用してEEPROMから内部RAMへプログラムを転送する．本研究では命令実行サイクルが20MHzのDSPを用いる．パラレル・インストラクションを用いて加算器と積算器を同時に使用した時に発生する性能のピーク値は40M flopsである．

ルータにはFPGA(Field-Programmable Gate Array)を用いる．FPGAにはXilinx社のXC4010Dを用いた[14]．このFPGAは，10,000ゲート相当のもので，ゲート遅延が5nsである．ピン数が160ピンあり，そのほとんどが電源とコンフィギュレーション可能なピンに割り振られている．DSPと16bitで接続し，他PEと他プロセッサと11本(8bit+3制御信号)で接続するリンクが6本，内部状態を表示する表示装置を接続したり，ホスト・コンピュータとの接続のためにリンクと同様の11本のピンを使用している．10,000ゲートのものを用いているが，これはFPGAが，ゲート数の使用率が高くなると動作速度が低下することから余裕をもたせたるためで，実際はその数分の1しか使用しない．そもそも，ルータを用意するきっかけとなったのは，細粒度インターリーブを用いた時，リンクから入って来るデータの$\frac{2}{3}$がプロセッサに入らず，別のリンクへ抜けていくデータだからである．TI社にはTMS320C4xシリーズがあり，これは，6本のパラレルリンクを持つプロセッサである．6つのDMAによってリンクからのデータをメモリへ転送することが可能だが，リンクの全てが使用されるとなると，プロセッサのメモリ競合が多くなる．ソフ

トウェアでルーティングもさせなければならないので，そのオーバーヘッドを考えてルータを用意することにした．

ボリューム・メモリには1byte/voxelのボリュームデータや3DDDAテーブル，2次元投射データを格納する．1PEあたり128K word(512K byte)でno waitの高速S-RAMを用いた．入出力バンド幅は20M word/sec(80M byte/sec)である．

3.2 データの配置

各プロセッサへのデータ配置は図18に示す3次元拡張されたインターリーブ分割(interleaved partitioning[11])を用いる．これは，元々レイ・トレーシング法を並列実行する際に分割する方法の1つで，できるだけ近隣のピクセルが別のプロセッサに割り当てられるように分割される．通常，人間が見ることを目的とした画像は，近隣ピクセルの画像が似たものになるため，各プロセッサには同じ負荷が割り振られる可能性が高くなり，非常に高い負荷均衡性を得ることができる．本研究室でもレイ・トレーシングの並列処理で最も高い負荷均衡性を確認している[15][16]．2次元画像における近隣画像の類似性は，3次元のボリューム・グラフィクスでも同じように得られると考えた．我々が暮している3次元空間は，同じ種類の物質同士がクーロン力によって結合した物体が形成している世界だからである．

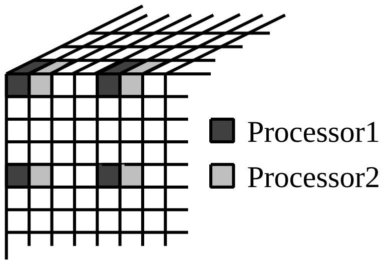


図18: 3次元インターリーブ分割

並列実行する上では非常に高い負荷均衡性を示すインターリーブ分割だが，他の手法との組み合せが難しいという問題がある．これは原理的に回避の方法がない場合があり，インターリーブ分割の効果を薄めるしかない．例えば，ボリュームグラフィクスで求められることが多い補間処理がこれにあたる．インターリーブ分割の効果を薄めるために，3次元インターリーブ分割の分割単位を大きくする．このインターリーブブロック内では，グラフィクスは通常のラスタ配置されたボリュームグラフィクスと同じ並び方をしている．

これを粗粒度インターリーブ分割と呼ぶことにする．粗粒度インターリーブ分割はインターリーブの効果が薄まったために，ピクセル並列における負荷均衡性が悪くなるというデメリットがあるが，いくつかのメリットもある．まず，通信速度とプロセッサ計算速度のバランスを取ることができる．1ボクセル計算に対して1依存データの通信というのは通信がボトルネックになる可能性がある．粗粒度インターリーブ分割を用いると，連続した複数のボクセルを計算することになるために通信量が減少する．2つめにさきほどの補間処理がある．足りないボクセルを隣りのプロ

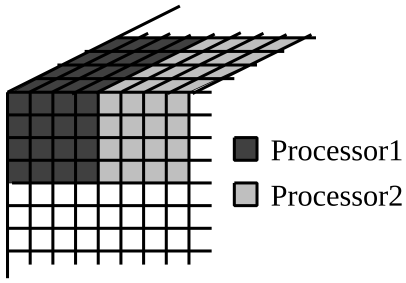


図 19: 粗粒度インターリーブ分割

セッサへ要求する場合，粒度が細いと通信量が劇的に増加すると考えられる．本研究では補間処理を行うために，インターリーブ境界にあたる面を各プロセッサ間で共有する方法を用いる．この場合にも粗粒度インターリーブ分割を行うことで共有するデータの量が劇的に減少する (2.5.3 章参照)．

3.3 通信経路

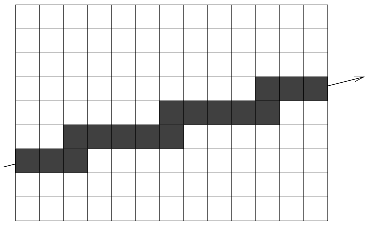


図 20: 増分ラインアルゴリズム

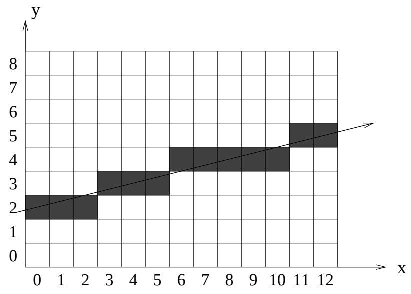


図 21: 中点ラインアルゴリズム (Bresenham's algorithm)

3 次元トーラス結合上で任意方向からの可視化を行うために，3DDDA(3 Dimension Digital Differential Analyzer) によってレイの進行方向を決定する．3DDDA はラスタ・グラフィックス・ディスプレイ上に，直線を描くために考案された方法で，増分ラインアルゴリズム (Incremental

line algorithm)(図 20) と中点ラインアルゴリズム (Midpoint line algorithm)(図 21) がある．前者はわずかでも通過するボクセルは全て対象とする方法である．後者は Bresenham's algorithm とも呼ばれ，レイの主軸方向成分を一定のステッピングで進めて，残りの成分を最も近い値に丸める方法である．

細粒度インターリーブを用いる場合，中点ラインアルゴリズムを用いる．中点ラインアルゴリズムを用いると，主軸方向のステッピングが 1 になるので，観測者から見て奥行方向の並列性の負荷が一定になる．このことによって 3 次元トーラス上をスムーズに回転させることが可能になる．逆に細粒度インターリーブに対して増分ラインアルゴリズムを用いた場合に，コンフリクトが生じる様子を図 22に示す．

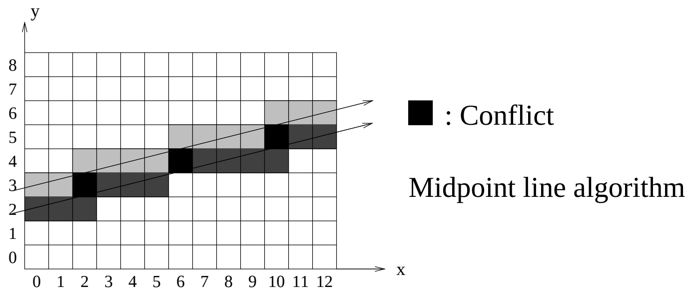


図 22: 増分ラインアルゴリズムにおける衝突

また，粗粒度インターリーブを用いる場合は，どちらのアルゴリズムを用いても大きな違いはない．インターリーブブロックが大きいため，どちらにしても横方向通信は逃れられないからである．しかし，細粒度との一貫性を考えると，増分ラインアルゴリズムの優位性はない．

実際に 3 次元トーラス上を移動するデータの様子を図 23に示す．説明を容易にするため，$4\times4\times4$ の 3 次元トーラス中の $Z = 0$ のみを示している．また，主軸は X で，$+X$ 方向に向かうレイが $+Y$ 方向にシフトして少し $+Y$ 方向へ斜めに見ている様子である．データ A,B,C,D が図 23の Step 1 に示すように配置しているが，1 ステップ後には Step 2 のように移動する．

また，データ通信に関しては次のような問題がある．

1. 通信データの識別
2. レイ発射のタイミング
3. 横方向通信の通信時間

3.3.1 通信データの識別

各 PE では，通信されてきた依存データが，どこから放射されたどの位置まで計算されたデータであるかが分かっていなければ，レイの進行方向を計算することができない．粒度の細い並列計算機では，そのデータを通信する方が良いだろう．しかし，粒度が粗いマシンに対しては，各 PE が視線方向のようなあらかじめブロードキャストしたデータから最初に来るデータがどの位

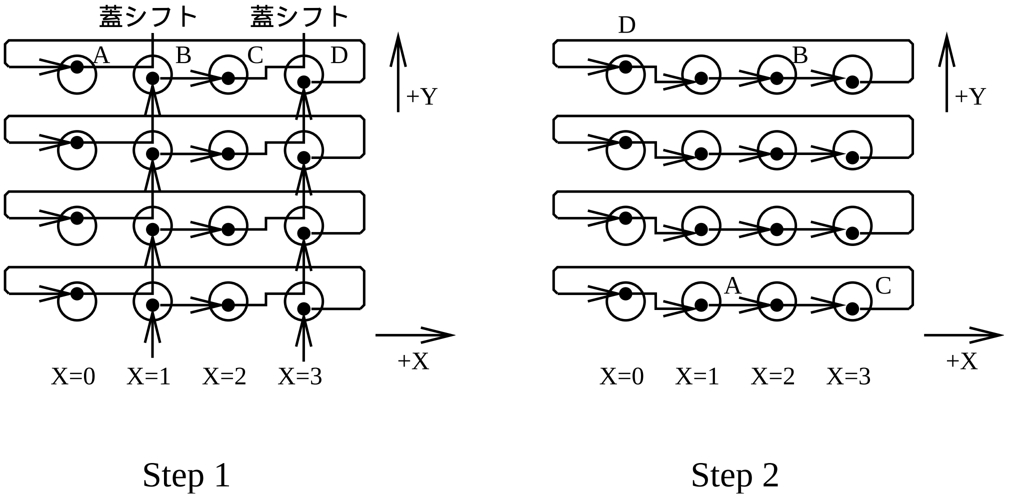


図 23: $4 \times 4 \times 4$ の場合の通信

置まで計算されたデータなのかを計算し，作り出す方が速い可能性がある．平行レイの場合，最初のレイは遅くなるかもしれないが，2 本目以降はテーブルによって高速に実行できる．そのため，VoViAc では両方を検討する必要がある．VoViRo では，通信にかかる時間は 5(スタートアップ・タイム)+(通信バイト数) のステップだけかかる．通信サイクルを 20MHz まで上げられたとしても，その差は 600ns で，命令サイクルが 50ns で動作する DSP は 12 命令をこなすことができる．つまり，この 12 命令以内で後者を実行できるかどうかがターニングポイントで，これは可能な値だと考えられる．

3.3.2 レイ発射のタイミング

レイをどういう出来事をきっかけにして発射するかという問題である．座標軸に平行なレイの場合は，レイが発射した PE へ戻って来るため，その時点で次のレイを発射すればよい．しかし，座標軸に平行でない場合，発射する PE とたどり着く PE とは異なる．この場合にも，平行レイを扱えば，発射する PE とたどり着く PE とは 1 対 1 の関係になるので，たどり着いた PE が発射した PE でなくても，たどり着くレイがあれば，新たなレイを発射すれば良い (図 24)．

遠近法の場合も，対称 4 点法ではレイの進み方が平行レイと同じに扱えるので，同じように行う．

3.3.3 横方向通信の通信時間

粗粒度インターリーブを用いた場合，視線方向に対して縦横方向へ転送するデータが生じるようになる．3.4章で述べるルータの制約上，一度データがルータに入ってしまうと，出てくるまでに必ず 3 ステップかかってしまう．このため，横方向通信が起こるたびに，リング中の PE を全てストールさせてしまう．これに対するペナルティは大きく，全体の速度が大きく低下してしまう．

この状態を最小限にするために，処理順序の入れ換えを行う．しかし，入れ変えられた本来後続のデータの処理が終わってもすぐには出力せず，出力は本来の順序どおりに行う．このように

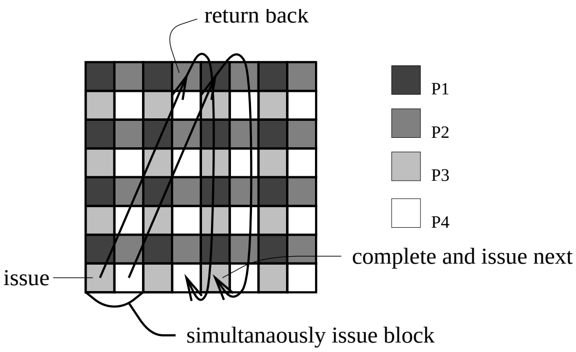


図 24: 発射した PE と異なる PE へ戻る場合

順序入れ替えをすることで，横方向通信に対するペナルティは，パイプライン全体で最も多くストールした箇所のステップ数になる．

この条件を満したソフトウェアの設計については 4 章で説明する．

3.4 ルータの構成

本アーキテクチャにおけるルータに求められることは，

- PE から周囲の 26PE 中の指示した PE へ通信できる．
- 3.3章における通信で，コンフリクトを起こさない．

の 2 点である．この事を満たす VoViAc のルータ VoViRo – Volume Visualization Router として，図 25に示す構造を提案する．図から分かるように，XYZ のそれぞれの軸に対して，パス・スイッチとセレクタがあり，パケットが周囲の 26 のどのプロセッサへ送られるかはこれらの状態によって決定される．

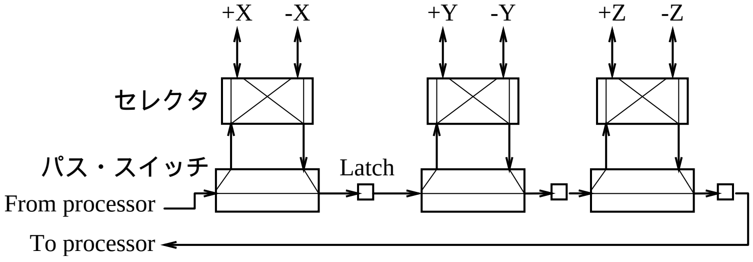


図 25: VoViRo の構造

パス・スイッチは，図 26に示すような 2 状態を持ち，データが+方向と−方向のどちらに出力/入力されるのかを決定する．3.3章のデータフローでは，等間隔の平行レイの場合に+方向にデー

タが出力される時は−方向からデータが入力され，逆も成り立つことを利用するためのものである．このことからも，リンクは単方向の切り替え式のものとする．

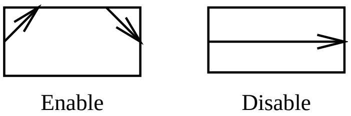


図 26: パス・スィッチ

セレクタは，図 27に示す 2 状態を持つ．このブロックを通るデータがその軸方向に通信を行うかどうかを決定する．

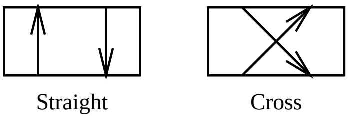


図 27: セレクタ

例えば，あるパケットが現在の PE に対して (+1,0,−1) の位置にある PE へ送られる場合，このパケットに対して，図 28のような組み合せで目的の PE へ着く．

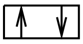
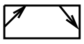
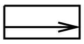
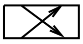
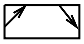
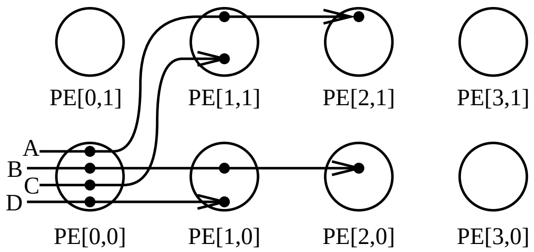
	パス・スイッチ	セレクタ
X		
Y		
Z		

図 28: 選択の例

この情報は，ルーティングヘッダに 6bit の情報として付加され，一緒に送られる．また，各軸間およびプロセッサとの間にラッチを用意してある．これがない場合には 3 本のリンク，パス・スイッチおよびセレクタを回線交換のように通ることになり，レイテンシが大きくなり，1 つのデータが長い間これらのリソースを占有してしまう．このために，ラッチを入れる事によってスループットが 3 倍近くになると考えられる．

この方法で，ルータを含めてどのようにデータが流れるのかを確かめておく．

図 30にルータ内部も含めたデータの流れの 1 例を挙げる．ここでも説明を容易にするために，Z 軸を除いた 2 次元で説明する．まず，主軸を$+X$とする．データ A,B,C,D は図 29のように移動するデータとする．データ E はデータ A が左シフトする時に右から入ってくるデータである．まず Step 1 でデータ A は X 軸移動をする．この時点でのデータ A の移動変位は (+1,+1) であるから，$+X$方向へ移動する．Step 2 で，データ A は Y 軸移動をする．X 軸移動と同様に$+Y$方向



A : PE[0,0] → PE[1,0] 経由→ PE[1,1] ⟶ PE[2,1]
B : PE[0,0] ⟶ PE[1,0] ⟶ PE[2,0]
C : PE[0,0] → PE[1,0] 経由→ PE[1,1]
D : PE[0,0] ⟶ PE[1,0]

図 29: データ A,B,C,D の動き

へ移動する．データ A が移動するのと同時に，データ B も$+X$軸方向へ移動する．同様の手順で Step 2，Step 3... と進んでいく．

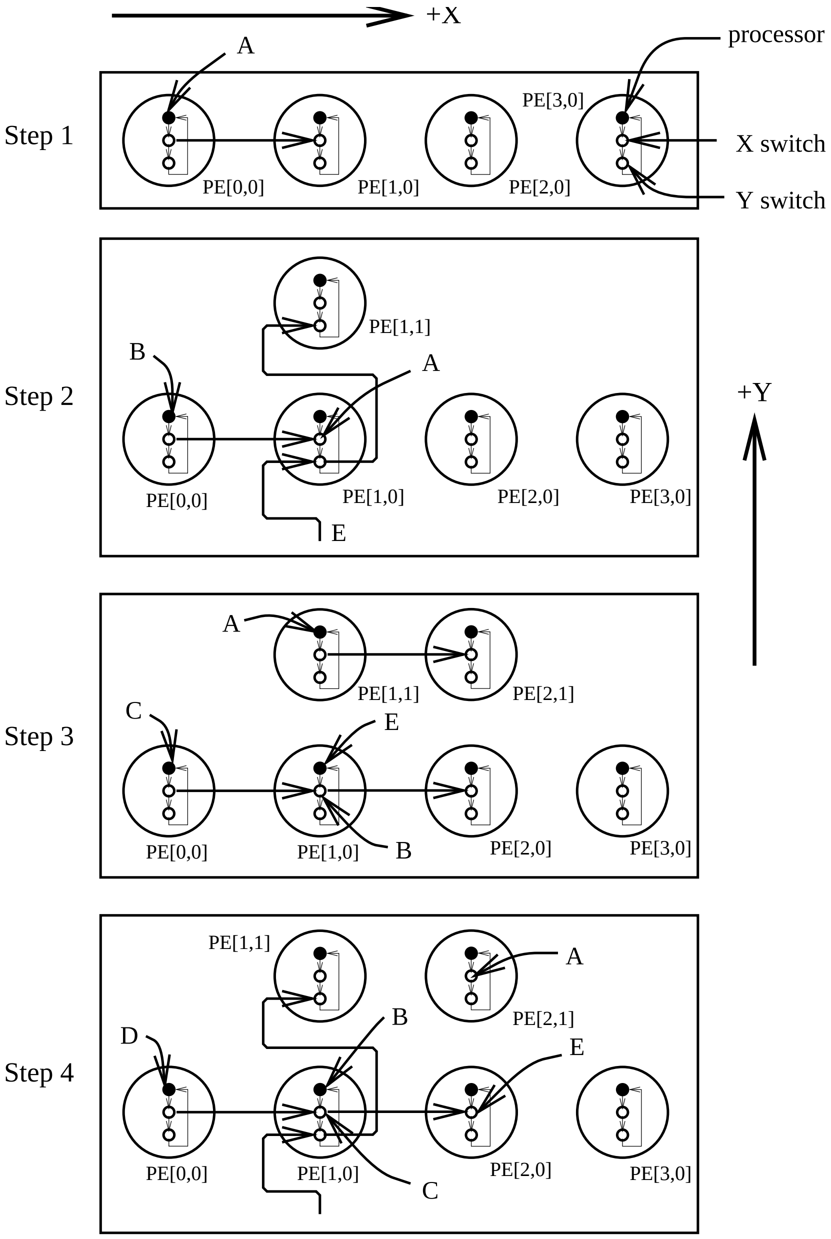


図 30: ルータを含めた全体のデータ流

4　ソフトウェア設計

4.1　通過レイの順序決定計算

VoViAcの制御ソフトウェアには，デッドロックを回避するために，各プロセッサの通信順序に一貫性を持たせ，シングルタスクで実行する．各PEは複数のリンクを持つため，受け順序には送り主も含まれる．そのため，送られてきたデータが望みどおりの相手のものでなければ，望みのものが来るまで受信データはストックしておく．そして，各プロセッサはこのレイがどこから来るのか，どこへ送るのかをはっきりさせながら，実行していく．

通信順序の一貫性はレイを管理することによって保たれる．

まず，主軸方向をzとする．そして，レイの方向を主軸方向の座標値を1として，$(R_h, R_v, 1)$とする．この時，主軸がzなので，$R_h < 1, R_v < 1$である．各PEのインターリーブ幅をIとし，3次元トーラスの1辺に相当するPE数をN，プロセッサ番号をPE$[X, Y, Z]$とする．

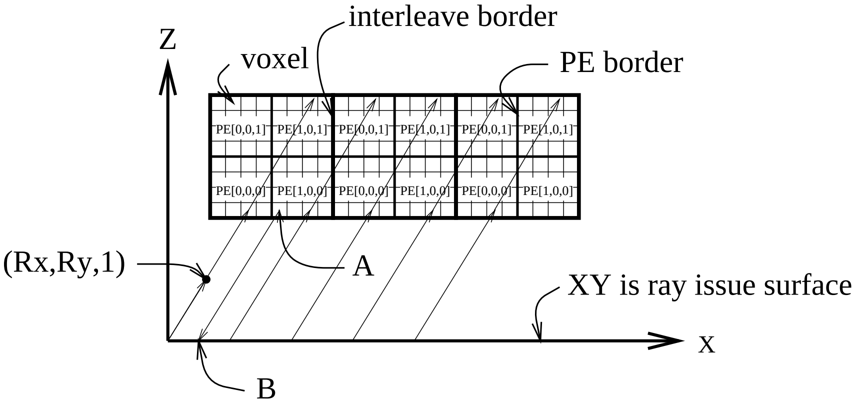


図31: レイ順序決定のための計算

図31にこの様子を示す．このとき，ターゲットとなるPEの点Aの座標値は，整数値p, q, rを用いて，

$$(NIp + IX \, , \, NIq + IY \, , \, NIr + IZ \,) \tag{15}$$

となる．よって，この点からレイと逆方向にレイの発射面へたどると，発射面には，

$$(NIp + IX - R_x(NIr + IZ), \; NIq + IY - R_y(NIr + IZ) \,) \tag{16}$$

にヒットする(点B)．これを(X_{hit}, Y_{hit})としておく．レイはインターリーブして発射するので，インターリーブブロックからは同時に1本のレイのみしか飛び出さない．a, bをインターリーブブロック内のオフセット，m, nを整数値とすると，レイが発射する座標は，

$$(NIm + a, NIn + b) \tag{17}$$

となる．計算する際は，これを，m,n,a,b の 4 重 loop としてスキャンする．(内から m,n,a,b の順) ここで，PE[X,Y,Z] に入るレイは，図 32から，

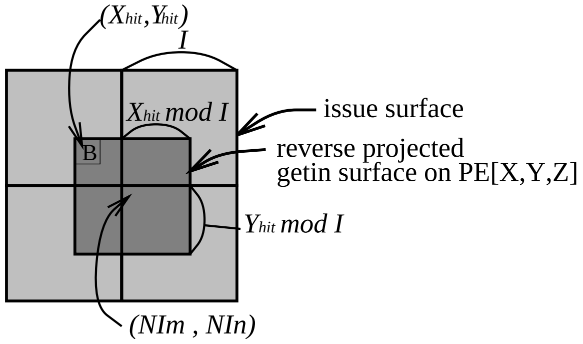


図 32: PE[X,Y,Z] への入力面の逆投射

$$\begin{aligned}
m &= X_{hit}\ div\ NI + 1 && (a \le X_{hit}\ mod\ I) \\
m &= X_{hit}\ div\ NI + 0 && (a > X_{hit}\ mod\ I) \\
n &= Y_{hit}\ div\ NI + 1 && (b \le Y_{hit}\ mod\ I) \\
n &= Y_{hit}\ div\ NI + 0 && (b < Y_{hit}\ mod\ I)
\end{aligned}$$

となる (div は除算結果の整数部分)．つまり，m, n, a, b および r が決まれば p, q が決定することを意味している．

そして，プロセッサがパイプライン実行することを考えると，m, n, a, b のスキャン順序が決まれば，p, q, r のスキャン順序が決まる．(m, n, a, b) を，ボリュームの一辺を V として，

```
for(b=0;b < I;b++)
  for(a=0;a < I;a++)
    for(n=0;n < V/NI; n += N)
      for m=0;n < V/NI; m += N){
        < キャスティング演算 >
        < (m,n),(m+1,n),(m,n+1),(m+1,n+1) を並列処理 >
      }
```

とスキャンした場合，同時にパイプライン実行するレイの数を pipeline_len とすると，p,q は，

```
for(b=0;b < I;b++)
  for(a=0;a < I;a++)
    for(r=0;n < V/NI; r++){
      for(count2 = 0;count2 < pipeline_len;count2++){
```

```
  n = (count+count2) div V/NI;
  m = (count+count2) \% V/NI;

  p = f(m,n);
  q = g(m,n);

  < キャスティング演算 >
 }
 count += pipeline_length;
}
```

のようにスキャンされる．当然，このまま実行するのではなく，できるかぎり差分計算を行うことで，高速化する．

4.2 通信レイテンシ削減のためのスケジューラ

横方向へレイを送り出す場合，後続のレイをストールさせてしまっては全体の性能を損ねてしまう．これに対処するには全体のデータフローをそのまま複数のプロセスにして，マルチタスキングによって実行することによって，ストールを防ぐ方法や，全体をスケジューリングすることによって，実行順序を入れ換える方法がある．本研究では後者を選択した．これを実現するために，ソフトウェアで 図 33に示されるパイプライン構造のスケジューラを用意する．

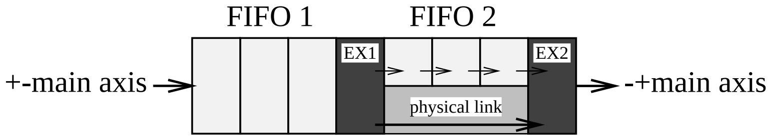


図 33: スケジューラ

リンクからのデータが FIFO 1 の左から入力される．EX1 ステージを実行中に，レイが横方向へ移動するものであると分った時点で，横方向へ通信し，図 33では下のリンク側へ送られる．もし，データが横方向通信しなければ上の FIFO 2 を通る．このように，パイプライン全体を右へシフトしながら，FIFO 2 を通ったデータもリンクを通ったデータも同じステップ後に EX2 ステージへ入る．EX2 ステージでは，もし，そのデータがリンクを通って来たデータなら，それを EX1 ステージより優先して実行し，奥行き方向へデータを送る．それから EX1 ステージを実行する．EX2,EX1 の両方が終ってからパイプラインを 1 ステップ右へ進める．このようなスケジューリングを行うことで，横方向通信のレイテンシを削減することが可能である．

残念ながら，これでもなおペナルティが残ることになる．横方向通信の中で，EX1 ステージで実行される時間が短く，EX2 ステージで実行される時間が長いものの割合が多い場合に，20～30%程度の速度低下は避けられないと考えられる．そのため，粗粒度インターリーブを全面的に

利用する場合，ルータを隣りとの通信だけの単純なものに換え，レイテンシを抑えた方がうまくいく可能性が高い．

4.3 PE の回路構成とコンフィギュレーション

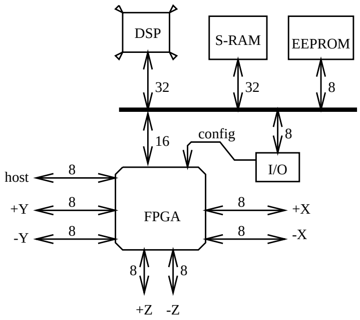


図 34: PE の回路構成

FPGA にはシリアル ROM からブートアップする機能が付いている．しかし，DSP と FPGA を両方共，ホストコンピュータからコンフィギュレーションできた方が，開発が容易になる．

まず，電源が投入されると，DSP は自身のブートアップ機能を用いて，EEPROM に格納されたデータを RAM に転送してブートアップする．次に，DSP は EEPROM の別エリアに格納した FPGA のコンフィギュレーションデータを FPGA へ転送して，FPGA をブートアップする．この FPGA データは，ホストプロセッサからのデータを各 PE へブロードキャストするだけのデータで，この時に本来の FPGA データ (VoViRo) と DSP のプログラムをロードする．そして，DSP は FPGA を再コンフィギュレーションしてから，目的プログラムを実行する．

また，FPGA の動作クロックは，DSP の分周サイクルか，DSP の timer 出力を用いることで，20MHz,10MHz,5MHz,.... が選択可能である．

5 実現と初期評価

5.1 PARTHENON および XACT によるルータの設計

PARTHENON は NTT(株) が開発したハードウェア記述言語 SFL を処理する処理系である [17]．SFL は C に似た論理記述で高レベルのハードウェア記述を行うことができ，スケマッティックや VHDL に比べ修得時間，開発期間ともに大幅に短縮することが可能である．制御信号を記述する instrin,instrout や順序論理回路を記述するタスクやステージ，プライオリティ付き条件文を記述する alt が特徴的である．

XACT は Xilinx 社が開発した自社の FPGA 用の開発環境である [18][19][20]．FPGA 用に論理をオプティマイズしたり，ピン・スワップしたりできる．

PARTHENON で設計した SFL をシミュレータである SECONDS でシミュレーションし，設計にミスがなければ，PARTHENON のオプティマイザである OPT_MAP でオプティマイズ後，XACT 用の xnf フォーマットに変換する．XACT で，ピン・スワップ後，FPGA 用にオプティマイズし，FPGA へデータを転送する．

SFL を用いる場合も，基本的には通常の回路設計と同じで，オートマトンを設計する必要がある．そこで，図 25のルータについて考えると，この概念図のままだと実際に回路設計する場合，各 PE をどうやって同期させるのかという問題が出てくる．

これには全プロセッサのプログラムを同じものにし，完全にクロック同期させてしまうことによって通信を同期させる方法や，プログラム側には時間のばらつきを許容して，調停回路によって早すぎるものを待たせることによって同期させる方法がある．前者は実行が単純明解に行える点では良いが，全てのプログラム実行が一致しているので，SIMD 型並列計算機における弱点と同様の問題が出てくる．例えば，データの値によって処理が場合分けされる場合に弱くなる点が挙げられる．また，本アーキテクチャの場合は，PE 数が多くなった場合にそれらの同期をとるのが難しくなるという問題もある．そのため，後者を用いることにした．

また，通信を同期させる場合，調停 (arbitration) はどうするのかという問題が出てくる．3.4章で説明したセレクタやパス・スィッチを用いた方法では，全ての調停を集中管理しなければならないことから複雑になるため，各部位で調停する．

図に X 方向のみのブロック図を示す．それぞれの丸がシーケンサを示している．

±inout は単方向のリンクで双方が送ろうとした時に排他的に動作する．±inout を制御するためのオートマトンを図 36に示す．

emitter はパケットヘッダによってどの方向にデータを送るのかを振り分ける．

collector は±inout と異なり，3 本のリンクを調停するので，競合した場合は，round robin で調停を行う．図 37に示すオートマトンで制御を行う．図から分かるように，リクエストやアクノリッジは±inout や emitter といった FPGA 内部の信号なので，信号が来たらすぐに返事を返す．

FIFO は 1 パケットを保持する 5 バイトの FIFO である．アルゴリズム上，FIFO が必要になった場合は DSP のサーキュラ・アドレッシングを用いて，ソフトウェアで高速に行う．そのため，本来 1 バイトのラッチで足りるが，プロセッサと router の間での受渡し時間がかかったとしても全体の性能をある程度保つことができると考え用意した．

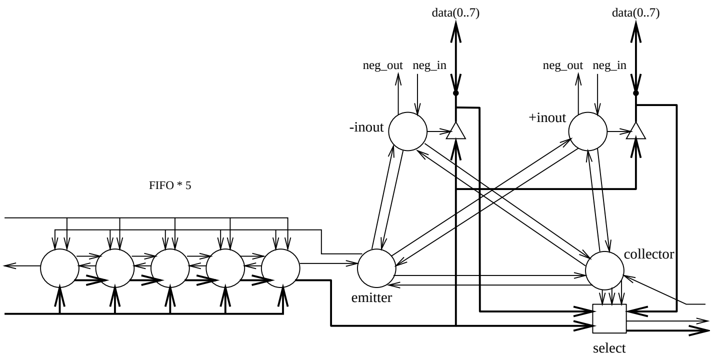


図 35: ルータのブロック図

5.2　初期評価

VoViAc では，どの程度の性能が見込まれるかを評価してみる．ルータは，FPGA で実行し，5～10MHz 程度で実行する．この時，データは 1～2M word/sec 程度の速度で通信される．プロセッサは細粒度インターリーブの場合，レイ・キャスティング計算と 3DDDA 計算，入力データ認識計算を行う．1 ボクセルあたり，5,5,8 クロック程度必要であると考えられ，これには命令実行速度が 20MHz の DSP を用いているので，900ns かかる．よって，一辺が 128 のボリューム (2M byte) を用いた時，4fps 程度の処理速度になると考えられる．

補間処理を必要とする場合は，粗粒度インターリーブを用いる必要があり，プロセッサ性能よりもリンク性能の方が高くなるので，通信されてきたレイの位置を各プロセッサが自身で計算するのではなく，通信することによって行う．

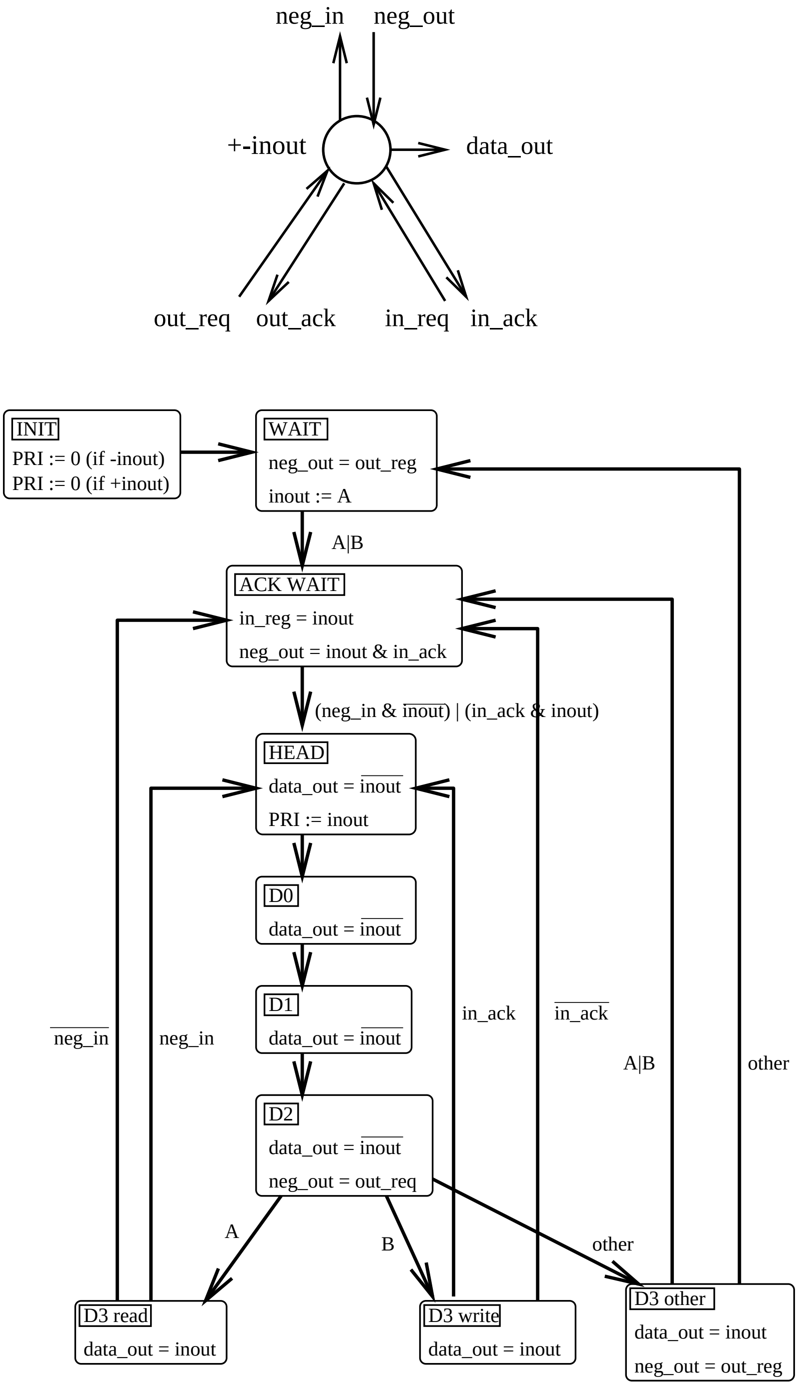


図 36: ±inout 制御オートマトン

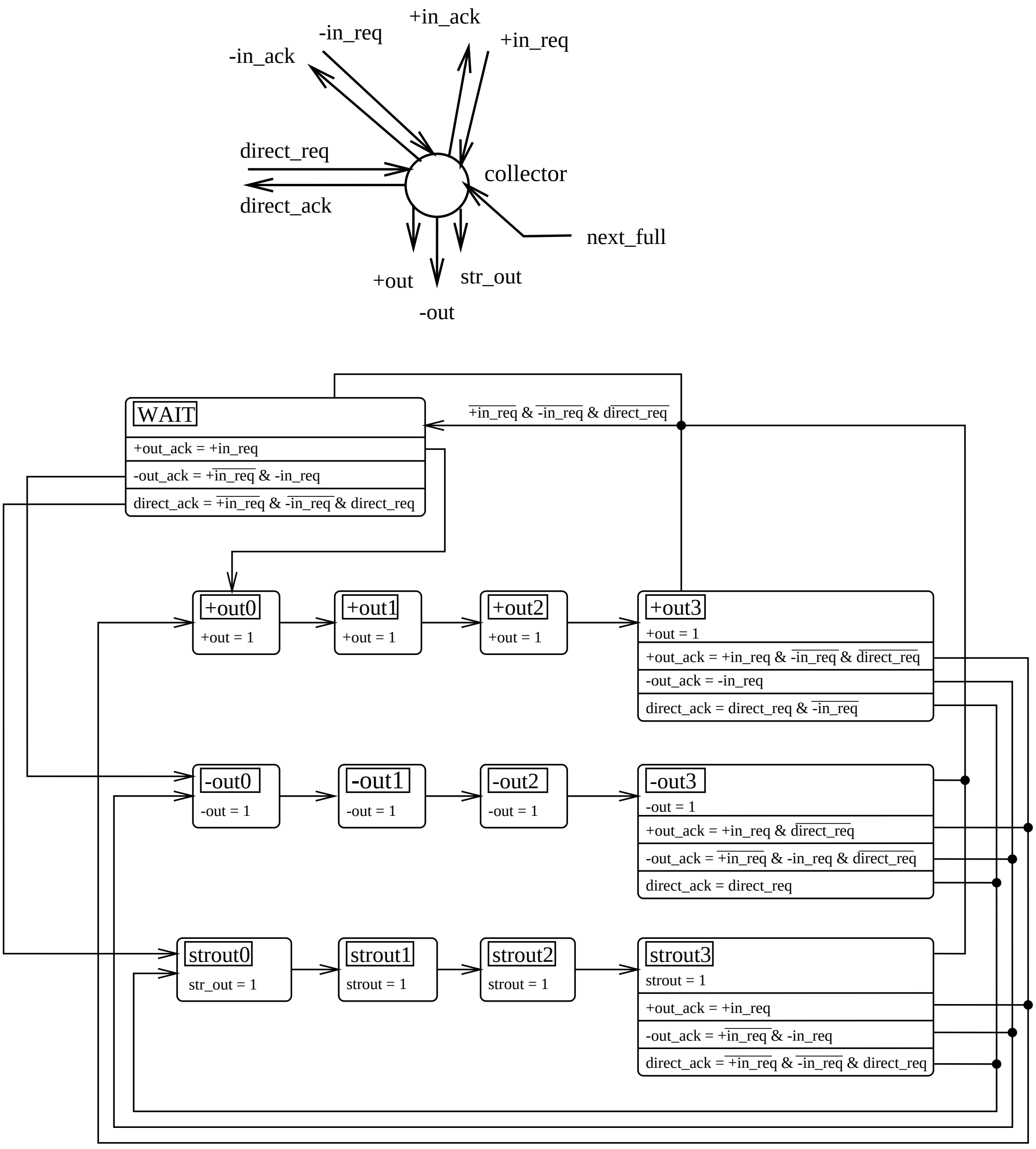


図 37: collector 制御オートマトン

6　おわりに

研究を進める過程で，3 次元ボリューム・グラフィクス処理はボリュームレベルで並列性を持っているにもかかわらず，任意方向からの可視化，補間処理，遠近法といった求められる条件の難しさのために，並列性を活用するのが難しいことが分った．汎用計算機のように実現手段が多く，その中からいかに良いものを選ぶかが難しいのとは正反対である．そのため，ボリュームグラフィクスを処理するハードウェアでは，画質と高速化手法とをうまく折衝することが良く行われる．4 点補間法や Z-buffer gradient 法はそのような過程で生まれて来たものである．こういったことは，計算能力が追いつくまでボリュームグラフィクスにおいては今後も行われていくだろう．

本論文では，3 次元トーラス結合上で，3 次元インターリーブ分割を用いる VoViAc のハードウェアについて述べた．3 次元の並列性を用いても，補間処理や遠近法を粗粒度インターリーブや冗長法，対称 4 点法といった方法を用いることで，処理可能なことが分った．現時点では対称 4 点法のように任意方向からの可視化との組み合せが難しいものもあり，改善の必要がある．その代り，3 次元の並列性を用いているので，スケイラビリティが高く，2 次元画像の収集やレイ方向のブロードキャストにかかる時間が大きくならない間は，必要とする画像のサイズとフレーム・レートに合わせてプロセッサ台数を拡張すればよい．

また，VoViAc に用いている並列アルゴリズムは，2^{3n} 台の 2 進ハイパーキューブ結合の並列計算機であれば，実行可能である．細粒度インターリーブはプロセッサ性能に合わせてルータも性能を出さなければならないが，粗粒度インターリーブを用いる場合は粒度を変えることができるので，計算機に依存せずにプロセッサ性能を出すことができると考えられる．VoViAc のアルゴリズムで粗粒度インターリーブを使う理由は，補間処理にあるため，ルータの性能よりはプロセッサ性能が必要となるだろう．

また，現時点では，VoViAc は最小構成の 8 台のうち 2PE が完成している．残りの 7 台も現在作成中である．計算に細粒度インターリーブを用いる場合，ある程度の性能予測が可能だが，粗粒度インターリーブでは，ルータを作成して通信速度が分からなければ，どの程度まで粒度を変えられるのかはっきりしない．また，ホスト間との 2 次元データの収集やブート・アッププログラムのブロードキャストといった，ソフトウェア面での開発も必要で，こういった通信が，計算に対してどういった影響を与えるのかも予測は難しい．つまり，計算機上でやってみなければ分からないことも多く，最小構成で動作させ，パラメータを変更し，データを収集する必要がある．つまり，VoViAc に関しては，ハードウェアを完成させるとともに，ソフトウェアの開発も必要である．

2 次元グラフィクスがボリューム・グラフィクスに置き換わるには，8 台の VoViAc の 250 倍程度の計算性能が必要となる．この 10 年間の計算機の性能向上を保てれば，これは遠い将来ではない．ボリュームグラフィクスではオンチップ・マルチプロセッサのようなアプローチでもかまわないので，逐次型計算機よりは可能性がずっと高い．

また，ハードウェアは支援ソフトウェアがなければユーザは使うことができない．そのため，2 次元グラフィクスがボリューム・グラフィクスに置き換わるそう遠くない将来がやって来るまでにボリューム・グラフィクスだから可能な魅力的な GUI やユーザが使う上で便利な応用を開拓しておく必要があると思う．

謝辞

本研究の機会を与えて下さいました，山崎勝弘教授に深く感謝の意を表します．また，これまで数々の御助言を頂きました研究室の皆様に感謝いたします．

PCB CAD を使用させていただくため，パソコンを使用させていただきソフトウェアのアップデートにもすぐに応じていただきました電気工学科の寺井秀一先生，並びに研究室の皆様に感謝いたします．

また，ハードウェア記述言語処理系 PARTHENON を使用させていただきました NTT 株式会社と，PCB CAD の MY-PCB III を使用させていただきました日立ソフト株式会社に感謝いたします．

参考文献

[1] 周藤安造: 医学における3次元画像処理 —基礎から応用まで—, コロナ社, 1995.

[2] 中島正之 監修: 3次元CG, オーム社, Feb. 1994.

[3] 明石英也: ボリューム・レンダリング向き並列計算機のアーキテクチャ, 京都大学工学部情報工学教室修士論文, 1992.

[4] 中山明則, 對馬雄次, 荻野友隆, 金喜都, 森眞一郎, 中島浩, 富田眞治: ボリュームレンダリング専用並列計算機 ReVolver/C40の試作, 情報処理学会第50回全国大会, 4B-1, 1995.

[5] 中山明則, 對馬雄次, 荻野友隆, 金喜都, 森眞一郎, 中島浩, 富田眞治: ボリュームレンダリング専用並列計算機 — ReVolver/C40 —, 並列処理シンポジウム JSPP'1995, pp.11–18.

[6] A.Kaufman and R.Bakalash: Memory and Processing Architecture for 3D Based Imagery, IEEE Computer Graphics & Applications, Vol.8, No.6, pp.10-23, Nov. 1988.

[7] Hanspeter Pfister, Arie Kaufman, Frank Wessels: Towards a Scalable Architecture for Real-Time Volume Rendering, Eurographics Workshop on Graphics Hardware, pp.123-130, August 1995.

[8] Jürgen Hesser and Reinhard Männer, Günter Knittel and Wolfgang Straßer, Hanspeter Pfister and Arie Kaufman: Three Architectures for Volume Rendering, EUROGRAPHICS '95, Vol.14, No.3, pp.C111-C122, 1995.

[9] D.Jackèl and W.Strasser: Reconstructiong Solids from Tomographic Scans - The PARCUM II System–, Advances in Computer Graphics Hardware II, pp.209-227, 1988.

[10] 高桑昌男: CGレイトレ物語, アスキー, Aug. 1990.

[11] J.D.Foley, A.VanDam, S.K.Feiner and J.F.Hughes: Computer Graphics – PRINCIPLES AND PRACTICE –, ADDISON-WESLEY, Nov. 1992.

[12] 山崎恭啓，山崎勝弘: ボリューム可視化用アクセラレータの検討, 情報処理学会第51回全国大会，4S-11，Sep. 1995.

[13] 日本テキサス・インスツルメンツ: TMS320C3x ユーザーズ・マニュアル 第3版, Apr. 1994.

[14] Xilinx: The Programmable Logic Data Book, 1994/1995.

[15] 上嶋 明, 山崎 勝弘, 渡部 透, 得丸 英勝: マルチトランスピュータシステム上でのラジオシティ法の並列化, 並列処理シンポジウム JSPP'1995, pp. 19–26, 1995.

[16] 吉谷崇史，原田博之，山崎 勝弘: 画面分割によるレイトレーシング法の並列化，システム制御情報学会第39回研究発表講演会, pp. 169–170, 1995.

[17] 中島 彰: ハードウェア記述言語によるASIC設計講座1–9, トランジスタ技術, CQ出版社, Jun. 1994 – Feb. 1995.

[18] Xilinx: XACT HARDWARE & PERIPHERALS GUIDE, Apr. 1994.

[19] Xilinx: XACT USER GUIDE, Apr. 1994.

[20] Xilinx: XACT REFERENCE GUIDE, VOL.I II III, Apr. 1994.