Vol. 34 No. 4 情報処理学会論文誌 Apr. 1993

PHIGS の構造体を処理するジオメトリ演算のマルチプロセッサ上での実行効率評価

松木 尚[†] 川瀬 桂[††] 森山孝男[††]

三次元グラフィックスのジオメトリ演算処理はマルチプロセッサの応用分野として有望なものの一つである．グラフィックスインタフェースとしてPHIGSを採用する場合には，そのモデリングデータの構造から派生する制約が並列化による高速化の問題点になる．筆者らは，実装が容易で汎用性の高い共有メモリ共有バス型マルチプロセッサ上で，この問題点を克服し効率良くPHIGSのジオメトリ処理を実行する二つの方式を提案してきた．これらの方式の有効性の確認および負荷分散やバス競合等のオーバヘッドを考慮に入れた場合の問題点や性能の上限を探るために，現実のモデリングデータを使用して，実行駆動型のマルチプロセッサシミュレータ上で様々な条件でシミュレーションを行った．一台のプロセッサがタスクのディスパッチを行う方法では，プロセッサ台数が増加するとディスパッチを行うプロセッサがボトルネックとなり，性能の向上を阻害する原因となることが判明した．この阻害要因を克服するために新たに簡単なディスパッチ支援機構を考案し，この機構を付加することによって性能が大幅に改善できることをシミュレーションにより確認した．また，すべてのプロセッサが対等にタスクの分配を行う方法ではバスのトラフィックの増大が問題となる．この場合には，all-read キャッシュプロトコルを用いることでバストラフィックを押え性能を改善できることが確認された．

Evaluation of PHIGS Geometry Processing on Multiprocessor Systems

Takashi Matsumoto,[†] Kei Kawase[††] and Takao Moriyama[††]

In present high-end graphics systems, geometric calculations for 3D graphic images are bottlenecks in system performance. To cope with this problem, parallel processing techniques are employed. However, in PHIGS, there are some obstacles to efficient parallel processing. We therefore proposed two mechanisms that enable shared-memory/shared-bus multiprocessors to perform efficient geometric calculations in PHIGS. To evaluate the quantitative effects of the mechanisms and estimate the influence of the overhead caused by job-dispatching and bus-contentions, we have performed various simulations on an execution-driven multiprocessor simulator. In a master-slave dispatching model, as the number of slave processors increases, task dispatches of the master processor becomes the bottleneck. We therefore devised a simple architectural support for task dispatch, and confirmed that it significantly improves the performance. In a no-master dispatching model where all processors work symmetrically, the increase in the bus traffic is the problem. We confirm that the all-read protocol reduces the bus traffic and improves the performance.

1. はじめに

マルチプロセッサの応用分野として三次元グラフィックスのジオメトリ演算処理は最も有望なものの一つである．現に高速なグラフィックスワークステーションではジオメトリ演算部をマルチプロセッサ構成にしたものが登場し始めている．従来からレイトレーシング等のグラフィックスアルゴリズムは並列度が高く粒度が大きいためにマルチプロセッサ上での処理に向いていることが知られている．筆者らはポリゴンレンダリングベースのグラフィックスのジオメトリ演算（座標変換，ライティング計算，クリッピング，データの形式変換）においても，負荷分散やスケーラビリティの観点から均質型のマルチプロセッサの優位性を示し，演算が基本的にコンピュテーションインテンシブでメモリアクセスが比較的少ない処理であるため，実装の容易な共有バス型マルチプロセッサが適用できることを示した[1]．それと同時に，グラフィックスインタフェースの標準になりつつある PHIGS (Programmer's Hierarchical Interactive Graphics System)[2] を採用する場合，モデリングデータの構造からくる制約が性能上の問題点となることを指摘し，この問題を

† 東京大学理学部情報科学科
Department of Information Science, Faculty of Science, The University of Tokyo
†† 日本アイ・ビー・エム(株)東京基礎研究所
Tokyo Research Laboratory, IBM Japan

解決する二種類の方式（ハードウェアサポート機構を含む）の提案を行った．

提案したアーキテクチャ（共有メモリ共有バス型マルチプロセッサ＋提案した機構）上で，負荷分散やバス競合等のオーバヘッドを考慮に入れた場合の問題点や性能の上限（スケールアップのメリットを享受できる限界）を探るために，現実のモデリングデータを使用して，実行駆動型のマルチプロセッサシミュレータの上で様々な条件でシミュレーションを行った．本論文ではその結果を報告する．なお，グラフィックスワークステーション等ではジオメトリ演算部のマルチプロセッサとは独立にホストプロセッサを設け，プロセッサへのジョブのディスパッチや負荷分散をホスト上で行うことがある．しかし，本論文では議論の単純化のためにマルチプロセッサ上でジオメトリ演算の他にディスパッチ等の処理も行われると仮定する．また，画像生成時のジオメトリ演算に必要となるデータはマルチプロセッサの共有メモリ内に存在するものとする．

2. 前提知識

まず，本論文の前提とする知識について PHIGS のデータ構造とトラバーサル，属性設定，順次振り分けによる負荷分散とその問題点の順に簡単に述べる．

2.1 PHIGS の階層データ構造とトラバーサル

PHIGS ではモデリング・データを階層状のデータ構造として記憶域に格納している．今後，この格納場所を CSS (Central Structure Store) と呼ぶ．このデータ構造はさらに構造体 (Structure) と呼ばれる下部構造を持つ．構造体はプログラミング言語のサブルーチン・コールのように他の構造体を呼び出すことができる．この呼び出しによりモデリング・データが階層状のデータ構造を形成する．各構造体は構造体要素から構成され，構造体要素にはこの構造体呼び出しの他に，各種の図形要素（プリミティブ：点，直線，ポリゴン等）の描画を指示するものと属性（アトリビュート：プリミティブの色，光の拡散率，変換マトリックス等）の設定を指示するものがある（**図1**）．

モデリング・データを表示する際は，CSS 内の階層状に構築されたデータを構造体呼び出しのチェインをたどって順次列に展開する（これをトラバーサルと呼ぶ）．トラバーサルの結果は，図形要素の出力と属性設定を指示する構造体要素の列になる（**図2**）．この要素を順次処理することで画像を生成する．

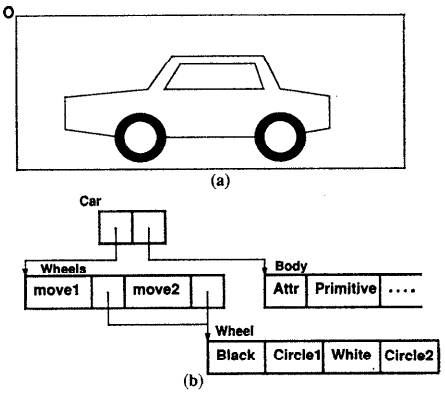


図 1 階層状データ構造の例
Fig. 1 An example of hierarchical data structure

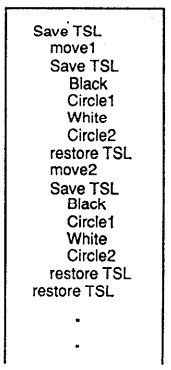


図 2 トラバーサル結果の順次列
Fig. 2 An example of display list generated by structure traversal.

通常 PHIGS は CSS 内の構造体の編集とトラバーサルによる表示を交互に行う．対話性を重んじるため，表示の速度と共に構造体の編集の速度も要求される．画像表示の高速化のみに焦点を絞れば，モデリング・データのデータ構造を表示時の前（編集時）にあらかじめデータ構造変換（コンパイル）しておくことにより，トラバーサル中の順次処理を無くし，並列処理可能にする方式も考えられる．しかし，この方式は構造体の編集時の速度低下につながり，また表示データのために必要とされるメモリ量も大幅に増大する．このため，本論文ではデータ構造を PHIGS 本来の物から大幅に変更して高速化する方式は議論の対象としない．

2.2 属性設定

属性（プリミティブの色，光の拡散率，変換マトリックス等）の設定に関して説明する．

プリミティブ描画処理中に必要とされる属性の種類はプリミティブの種類によって異なるが，多くのものは後続するプリミティブ間で共通している場合が多い．例えば，一つの物体をいくつかのポリゴンで表現する場合，変換マトリックスは共通であるし，均一な物質からできていれば色や光の拡散率は変わらない．PHIGS では，一度設定した属性値は同じ属性が再設定されない限りその構造体と下位構造体に現れるすべてのプリミティブの描画時に有効である．トラバーサル時には属性設定の指令にしたがってある領域に属性値を保存しておき，プリミティブの描画指示を処理する間にはその領域から属性値を読み出して使用する．この属性を保存する領域を今後 **TSL** (Traversal State List) と呼ぶことにする．PHIGS の定義[2]では TSL の大きさは可変長 (1 K byte 以上) で制限はないが，実装時に大きさを制限することも許されており，通常は 1 K から 2 K byte 程度の固定長で実装する．

構造体の呼び出し時に下位構造体は上位構造体の TSL の値を相続する．下位構造体の処理が終わった時点で呼び出し前の TSL の値を復帰する．図2において，最初の Circle 1 を描画する場合，move 1 と Black の属性が有効で，2個目の Circle 2 については move 2 と White が有効となる．

2.3 順次振り分けによる負荷分散

ジオメトリ演算のマルチプロセッサ上での負荷分散の方法は一つのプリミティブ（ポリゴンやライン）の表示のための処理をパイプライン状に細かく分割する**パイプライン方式**と，手の空いた一つのプロセッサに一つのプリミティブの表示のためのジオメトリ計算をひとまとめにして順次割り当てる**順次振り分け方式**[4),5)]が考えられる．

パイプライン方式は，最初のパイプラインステージ（プロセッサ）は座標変換，二番目はライティング計算，三番目はクリッピング計算，四番目は透視変換のための割算，五番目はデプスキューイングやドローイング部へのデータの加工（浮動小数点数から固定小数点数への変換等）を行うといったように役割分担を行うことによって負荷分散を行う．このパイプライン方式は以下のような欠点を持っている．

1. ステージ間の負荷分散が難しい．
2. ステージ間のデータの移動量が多く，局所性が利用できない．
3. ハードウェアの構造とプログラム（マイクロコード）が密に関連し，スケーラビリティやフレキシビリティが乏しい．

逆に順次振り分け方式の場合，後述する TSL 管理の問題が存在するが，上記の問題点は存在しない．本論文では順次振り分け方式の負荷分散を採用し，TSL 管理の問題を解決する方式を提案するアプローチを選択している[1)]．

通常ポリゴンベースのグラフィックスシステムではデプスバッファによる隠面消去がジオメトリ演算部の次工程のドローイング部内でハードウェアにより実現されるので，基本的にジオメトリ演算部における各プリミティブを独立に処理することができる．そこで，異なるプリミティブを同時に複数の要素プロセッサで並列処理することによりジオメトリ演算が高速化できる．順次振り分け方式の場合，動的に負荷を分散することができるので，要素プロセッサへのデータ供給がボトルネックにならなければ，要素プロセッサの使用効率を 100% 近くにすることができる．

2.4 順次振り分け方式の問題点

順次振り分け方式で並列処理する場合，プロセッサがプリミティブの処理をしている間，TSL のスナップショットをプロセッサごとに個別に保持しなければならない．

何故ならば，共有メモリ上に全プロセッサで共通の TSL を設けたとすると，あるプロセッサがプリミティブの描画を行っている間は TSL をそのプリミティブのためのスナップショットに固定しておく必要があるので，そのプリミティブの描画の指示に続く属性設定の指示を実行することができない．このため，トラバーサルによって展開された構造体要素の順次列の処理を先に進めることができず，プリミティブ単位の並列実行ができない．

この制約の最もナイーブな解決法は，共有メモリ上に共通の TSL を設け，各プロセッサがプリミティブの処理開始前に毎回共通 TSL をローカルなメモリエリアにコピーすることでローカルな TSL のスナップショットを確保し，一個のプリミティブの描画中はそのコピーを参照するという方法である．しかし，この方式では，プリミティブの描画のたびに TSL (gra PHIGS[3)]では約 1 K Byte) のコピーが必要となり，コピー自体がオーバヘッドであり，また，共有バスの競合を招き性能が低下する．

3. TSL の管理を支援する方式

前章で述べた問題点を解決するためにトラバーサル

を行う主体によって異なる二種類のトラバーサル法（DP 方式とシンメトリ方式）を考案した[1]．それらにおける TSL の管理法を説明する．

3.1 DP 方式と支援機構

構造体トラバーサルとプリミティブのディスパッチを特定の1台のプロセッサ DP (Dispatch Processor) が行う場合（以下 **DP 方式**と呼ぶ）の機構について最初に述べる．**図3**のように属性設定の指示を全プロセッサにブロードキャストするバス（ABUS: Attribute BUS）を設け，各要素プロセッサに TSL のメンテナンスのためにデュアルポートメモリ三組からなる **TTS** (Triple Tsl Store) を用意する．ただし，共有バスに ABUS の役割を兼ねさせることもできる．

DP は CSS 内のデータのトラバーサルを行い，構造体要素が属性設定の指示であれば，その情報を ABUS に出力する．プリミティブ描画の指示の場合は，処理を行っていない要素プロセッサの一つを選択し，その TTS の一部を凍結させるとともにプリミティブの処理を依頼する（タスクのディスパッチ）．

各 TTS の三組のメモリの内の一組がマスタ（master）であり，マスタは常に ABUS の内容を反映した最新の TSL の値を保持している．残りの二つのメモリはスレーブ（slave 1, slave 2）で交互にマスタのある時点のスナップショットとして要素プロセッサから参照される（**図4**）．

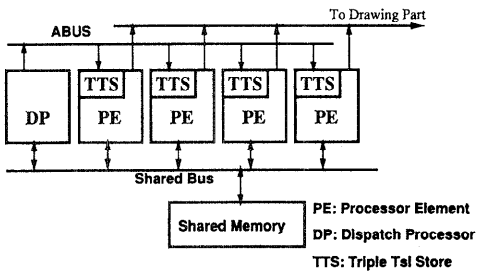


図 3 DP 方式に対応する機構を含む構成
Fig. 3 A system with the mechanism for DP method.

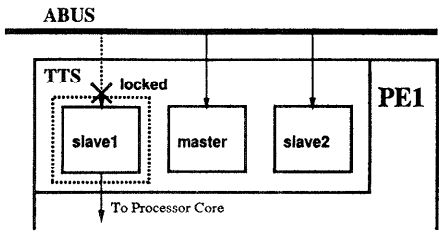


図 4 要素プロセッサ内の TTS の構造
Fig. 4 Structure of TTS in a processor element.

この機構の動作の概略を述べる．要素プロセッサ PE1 がスレーブメモリの一方である slave1 を参照しながらあるプリミティブを処理していると仮定する．slave1 は ABUS から切り離され内容は不変に保たれている（ロックされている）．PE1 がこの処理を継続中に ABUS 上に放送された属性設定の指示は master ともう一方のスレーブ slave 2 に反映される．

処理が終了すると PE1 は DP にタスクの処理の終了を通知し，新しいプリミティブの割当を要求する．これと同時に slave 1 はロックが解除され，TSL の最新の値をキャッチアップするために，master から slave 1 へのコピーが行われる．このコピーの最中も ABUS 上の属性設定の指示が master や slave 1 に反映されるように，デュアルポートメモリが TTS の構成要素に用いられる．slave1 上で master からの書き込みと ABUS からの書き込みが同じアドレスで競合した場合は，ABUS からの書き込みが優先され，master からの書き込みデータは捨てられる．

master から slave1 へのコピーの間は PE 1 に対するプリミティブの新規の割当を禁止することにすれば，slave 2 は必要がない．しかし，処理の終了した要素プロセッサにすぐさま次のプリミティブをディスパッチできるようにスレーブを二重化している．DP は PE 1 に新しいプリミティブを割り当てる直前に，ABUS を用いて PE 1 に対するスレーブメモリのロック指令を放送する．PE 1 以外の要素プロセッサはこの指令を無視するが，PE 1 は直前に使用されていたスレーブと異なる側のスレーブメモリ（この場合 slave 2）にロックをかける（ABUS からの書き込みを停止する）．この直後に DP が PE 1 に新しいプリミティブの処理を開始させ，PE 1 は slave 2 内の属性値を参照しながら処理を行う．PE 1 が新しい処理を行っている間も，master から slave 1 へのコピーを継続できる．そして，このコピーに要する時間よりプリミティブの処理時間が長ければ，コピーの時間的オーバヘッドは完全に隠される．

3.2 シンメトリ方式と支援機構

次に，専用ハードウェアである TTS を用いずに，汎用アーキテクチャで解決する方法について考える．基本的には特定のディスパッチ用のプロセッサを持たず，各要素プロセッサが独立にトラバーサルを行う方式（以下**シンメトリ方式**と呼ぶ）である．各プロセッサはそれぞれ TSL を持ち，トラバースの進行にそって個別に管理する．

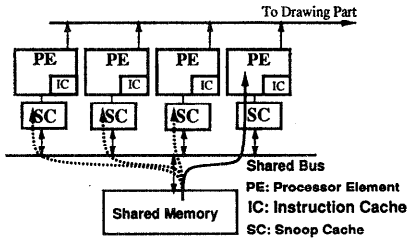


図 5 all-read プロトコルの動作
Fig. 5 Behavior of 'all-read' protocol.

プリミティブのディスパッチの方法を概説する．共有メモリ上に排他的にアクセスされるカウンタを設け，各要素プロセッサはカウンタの値を読み出し（チケットを獲得し），1だけインクリメントして更新する．要素プロセッサは通し番号がチケットの値と同じプリミティブに出会うまでトラバーサルを進める．その間属性設定の指示に従って TSL の内容は更新するが，通し番号がチケットの値より小さいプリミティブは無視する．その後チケットの値と同じ通し番号のプリミティブの処理を行う．処理が済むと，再び共有カウンタをアクセスし，次のチケットを獲得する．

この方式で，問題となるのは複数のプロセッサが共有メモリ内の構造体を使ってトラバーサルを行うことに起因するバストラフィックの増加とそれに伴うバス競合である．これを解決する方法として筆者らの考案した**スヌープキャッシュ制御機構**[6),7)]を用いることができる．CSS 領域内のすべてのプロセッサが読み込む必要のある部分（構造体ヘッダや属性設定指示など）に対して，スヌーププロトコルを **all-read**[7),8)]に設定しておき，ある要素プロセッサが共有バスを用いてこれらのデータを読みだした場合にバスをスヌープしている他のプロセッサのキャッシュもそのデータをできる限りキャッシュ内に取り込むようにする（**図 5** 参照，右端のプロセッサがバスアクセスを行っている）．これにより，他の要素プロセッサがトラバーサルのために同じデータをその後に読もうとした場合，そのデータはすでにキャッシュされていることが期待され，共有バスへのアクセスの必要もなく，また高速にデータを参照できる．もちろん，あるプリミティブの処理を終えた後，新しいチケットの値に対応するプリミティブを獲得するまでのトラバーサルによる空読みは時間的オーバヘッドになる．

4. シミュレーションの方式と仮定

4.1 シミュレータの構成

筆者らが作成したマルチプロセッサシミュレータの構成について簡単に説明する．オーバヘッドの厳密な評価が可能なように，クロックレベルの実行駆動方式，つまりシミュレータのプロセッサモデルが実際に命令をフェッチ解釈実行する方式を取っている．プロセッサモデルの命令セットは R 3000 の命令セット[9]を一部拡張（fetch&inc, fetch&dec, exchange 不可分命令の追加）して使用している．プロセッサのパイプラインは2段で1段目が命令フェッチ，2段目が解釈実行となっており，2段目の時間コストは命令ごとに設定可能である．プロセッサはレジスタ 32 本の他に命令キャッシュを内蔵している．

スヌープキャッシュ制御機構のためメモリをセグメントによって管理しており，セグメントごとにキャッシュプロトコルを指定できる．スヌープキャッシュは命令・データ混在型で，キャッシュのブロック幅が共有バスのデータ幅よりも広い場合は，バースト（ブロック）転送によって共有バス上のデータの転送を行う．また，スヌープキャッシュ制御機構に対応するために，複数のプロトコルを実装しており，プロセッサからのプロトコルタイプの出力によりプロトコルを使い分ける．スヌープキャッシュとプロセッサ内蔵のインストラクションキャッシュは共にキャッシュサイズ，ブロックサイズ，ウェイ数，キャッシュメモリのアクセスタイムが設定可能である．バスアービタのバス調停は先着順で，同一クロックサイクル内に到着したバス要求に対しては，バス使用ごとに優先度をプロセッサ間でローテーションすることで公平化を図っている．共有バスはアドレス出力から所要データ転送まで1バスサイクルで行うインタロック方式で，1バスサイクルのクロック数は共有メモリまたはキャッシュメモリのアクセスタイムで決定される．

4.2 シミュレーションの方式

シミュレーションの方式の特徴を以下に列挙する．

- オーバヘッドを完全に反映させるために，ディスパッチとトラバーサルの処理をアセンブラで記述し，シミュレータに実行させた．
- 要求される画像の品質によって，出力三角形当たりの処理量も変化する．ここでは，以前の研究[1)]の見積りに基づいて，低品位，中品位，高品位に対応して，200, 400, 600 クロックの3種の場合

を想定し，ウエイトループを回すことでそれぞれのシミュレーションを行った．なお，n 角形のポリゴンは $n-2$ 個の三角形として扱った．また，プリミティブを処理するためのデータフェッチのレイテンシとバス競合を反映させるために，上記のクロック数のウエイトを行わせる前に三角形単位で必要なデータをフェッチさせた．

- PHIGS の構造体のサンプルとして，現実のモデリングデータで入手可能であった GPCmark[10] の cyl_head を採用した．ただし，テストループの繰り返し回数は 255 回のところ 1 回とした．ループ 1 回あたりのプリミティブ数は 3653 個，平均 5.2 角形，出力三角形に換算すると 11692 個である．構造体は 2 ワードの構造体要素ヘッダの配列から成り，要素データ自身は他の領域に置かれヘッダ内のポインタによって参照される．要素データは要素データのサイズを表す 1 ワードを先頭に構造体要素固有のデータが続く．
- DP 方式のプリミティブのディスパッチは共有メモリ上の共有変数を介して行った．ディスパッチのための空きプロセッサの探索は各プロセッサに対応するビジーフラグをポーリングで調べる方式を用いた．
- シンメトリ方式の場合の共有カウンタへのアクセスには fetch&inc 命令を使用する方式で行った．
- DP 方式の ABUS は抽象化すると共有バスが ABUS の分だけ増強されていると見做すことができる．また，実際に TTS を共有バス上に設けることで共有バスに ABUS の役割を兼ねさせることができる．このため，シミュレーションでは共有バスに ABUS の役割を兼務させた．

4.3 パラメータの設定

シミュレータの設定可能なパラメータのうち，シミュレーションにおいて固定したパラメータの設定内容を以下に列挙する．

- 共有バスの転送能力はバースト転送時 266 MB/sec，非バースト転送時 133 MB/sec (64 bit 幅，50 MHz，バースト転送は 4 転送 6 クロック，非バースト転送は 1 転送 3 クロック) と仮定した．また，バス幅 128 bit で非バースト転送としたもの (266 MB/sec) も試した．
- キャッシュの構成はプロセッサ内蔵の命令キャッシュが 4 KB，スヌープキャッシュが 16 KB，共に 2 ウェイ．データの load/store はバスアクセスを伴わない場合が 2 クロックで，バスアクセスを伴う場合は 2 クロックに加えてバスを獲得してからバースト転送 (キャッシュブロックサイズ 32 Byte) 時で 7 クロック，非バースト (キャッシュブロックサイズ 8 Byte) 時で 4 クロックのコストがかかる．ただし，バス幅 128 bit の時はキャッシュブロックサイズは 16 Byte．
- スヌープキャッシュのプロトコルは基本的に writeback・invalidate とし，シンメトリ方式ではメモリの領域ごとにプロトコル切替を利用した．トラバーサルですべてのプロセッサから参照されるデータには all-read プロトコルの一種 allread-up (read 時は all-read で write 時は update) を使用し，共有カウンタには update (broadcast) プロトコルを使用した．
- プロセッサの命令コストはレジスタ演算命令が 1 クロック，分岐命令が 2 クロック，load/store はデータのアクセスが終了するまでサスペンドと設定した．

5. シミュレーションの結果と考察

5.1 DP 方式

DP 方式においてバースト転送時と 128 bit 幅非バースト転送時の結果を対比したグラフを**図 6** に示す．横軸はプロセッサ台数 (DP を含む) を表し，縦軸は出力三角形換算の処理性能 (M triangles/sec) を表す．1 画面作成し終わるのに掛かったクロック数の測定を行い，このクロック数から処理性能を計算した．ただし，クロック周波数は 50 MHz として換算した．図中の burst，wbus がそれぞれバースト，128 bit バス幅転送の区別を，(200)，(400)，(600) がそれぞれ低品位，中品位，高品位の画像生成の場合を表す．

図 6 からわかるように，DP 方式はプロセッサ台数が比較的少ない範囲では非常に線形性のよいスピードアップを示すが，ある時点で処理能力が飽和してしまう．また，飽和時点の処理能力は画像の品位 (つまりジオメトリ演算の粒度) にはほとんど無関係である．しかも，品位別の処理能力のわずかの差の大部分が，プリミティブ単位で負荷分散することで生じるプロセッサの処理終了時間のばらつきで説明可能である．このことから，共有バスが飽和しているのではなく，プリミティブのトラバースとディスパッチのためのコストで DP の処理が飽和していると考えられる．実際，

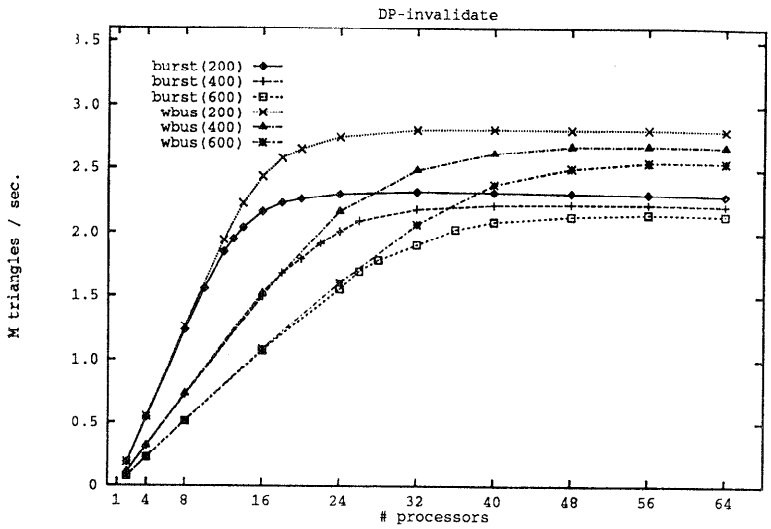


図 6 DP 方式のシミュレーション結果
Fig. 6 Simulation result for DP method.

プロセッサがプリミティブのディスパッチ待ちの状態のクロック数とプリミティブの処理中のクロック数を分けて集計したところ，プリミティブの処理のための時間の割合は低品位バースト転送 32 プロセッサ時で 40% 台であった．また，その時のバスの占有率を集計したところバースト転送時で 90%，128 bit 幅転送時で 77% であった．バス転送能力が同じであるにも拘らず，両者の間に性能差やバス占有率の差が生じるのは，バースト転送時のキャッシュブロックサイズが 32 Byte，128 bit バス幅転送時のブロックサイズが 16 Byte であり，128 bit 幅転送時の方が不必要なデータがバスに流れることが少ないためである．また，このときメモリアクセスのレイテンシも 128 bit 幅転送の方が小さいためディスパッチ能力も向上する．

また，図には示していないが，バースト転送時と同じ 64 bit 幅のバスで非バースト転送をした場合の性能は，常にバースト転送のグラフを若干下回るものとなった．ただし，この場合の性能の飽和は DP のディスパッチ能力の飽和が主原因ではなく，バスの飽和によるものである．

5.2 シンメトリ方式

図 7，**図 8** にそれぞれバースト転送時と非バースト転送時のシンメトリ方式のシミュレーション結果を示す．この二図にはスヌープキャッシュ制御機構の効果を明らかにするため，キャッシュプロトコルを invalidate に固定した場合のグラフ（図中で inv と表示）も描かれている．グラフの横軸，縦軸および画像の品位の指定は図 6 と同じで，all がスヌープキャッシュ制御機構を利用した提案方式を表している．

図 7 および図 8 より，バースト転送時ではプロセッサ数が 16 台を越えるあたりから，また，非バースト転送時では 8 台を越えるあたりから，スヌープキャッシュ制御機構による all-read プロトコルの効果が顕著になっている．

通常の invalidate プロトコルのみの場合，バスアクセスの競合によって台数以上では処理能力が低下し始めるが，all-read プロトコルを利用してバスアクセスの最小化を図ることでこの現象は防げることがわかる．

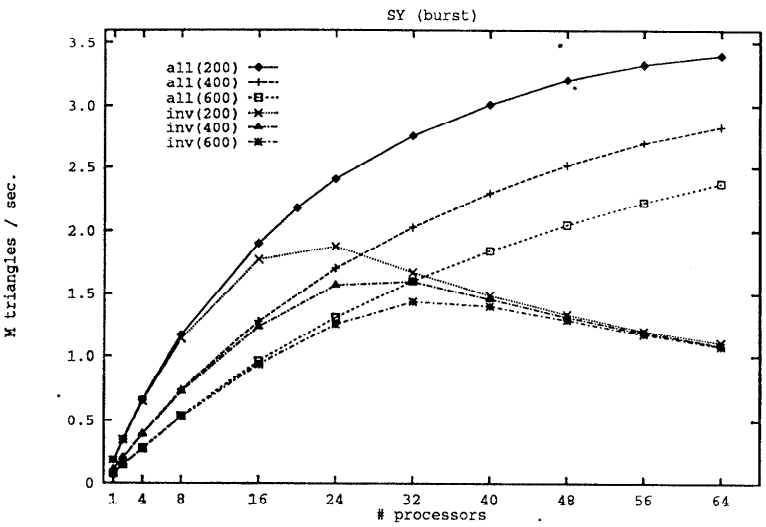


図 7 バースト転送時のシンメトリ方式のシミュレーション結果
Fig. 7 Symmetry method with burst bus transfer.

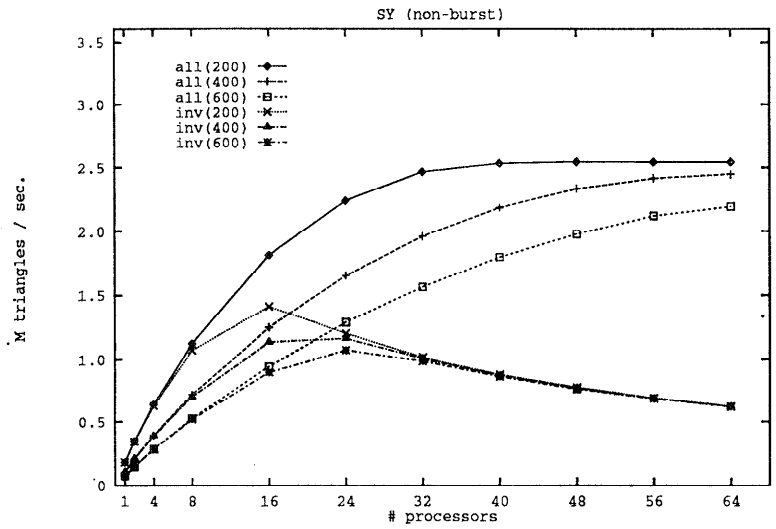


図 8 非バースト転送時のシンメトリ方式のシミュレーション結果
Fig. 8 Symmetry method with non-burst transfer.

今回用いた cyl_head のポリゴンは平均 5.2 角形であるため，ポリゴンあたりの必要なデータ量は 8+4+5.2*24=136.8 バイトである．シンメトリ方式の場合すべての構造体要素のヘッダの第一ワード目をすべてのプロセッサが読み込む必要があるため，n プロセッサで処理する場合，各プロセッサが 1 プリミティブあたりに読み込むデータは $4*n+4+4+5.2*24$ となる．1 プロセッサの場合と比べると 32 プロセッサの場合 1.9 倍，64 プロセッサの場合 2.8 倍である．このためプロセッサ数が増加するにつれてプロセッサあたりの実効性能（処理のうち，ジオメトリ演算に使用される割合）が落ちていく．属性設定指示もヘッダ同様にすべてのプロセッサが読んで処理する必要があり，モデリングデータ内の属性設定指示が多くなると，この実行性能低下傾向はさらに顕著になる．

図 8 の all-read (200) の曲線がプロセッサ台数 40 以上のあたりで飽和しているが，これは共有バスが飽和したものである（バス占有率 99.6%）．

5.3 両方式の性能の比較

DP 方式とシンメトリ方式の性能を比較するために，バースト転送の場合に関して，invalidate プロトコルを用いた DP 方式と all-read プロトコルを用いたシンメトリ方式を対比したグラフを**図 9** に示す．

図より，プロセッサ台数が比較的少ない範囲（8 台以下程度）では，ディスパッチとトラバースに 1 台のプロセッサを占有されないために，シンメトリ方式の方が有利である．それ以上の台数になると，DP のディスパッチ能力が飽和する地点（品位に依存）までは，直線的な台数効果が得られる DP 方式の方が有利である．DP 方式ではプリミティブごとのディスパッチのコストが DP に集中し，しかもこのコストの大きさがトラバースのコストとほぼ同じかそれ以上かかっている．このことがシンメトリ方式よりも低い性能で飽和する原因につながっている．

これに対して，シンメトリ方式は DP 方式のディスパッチ能力のようなボトルネックが無いので，バスが飽和しない限り台数効果が期待できる．ただし，前節で述べたようにシンメトリ方式では属性設定指示が多くなると各プロセッサが読み込むデータ量が増加する

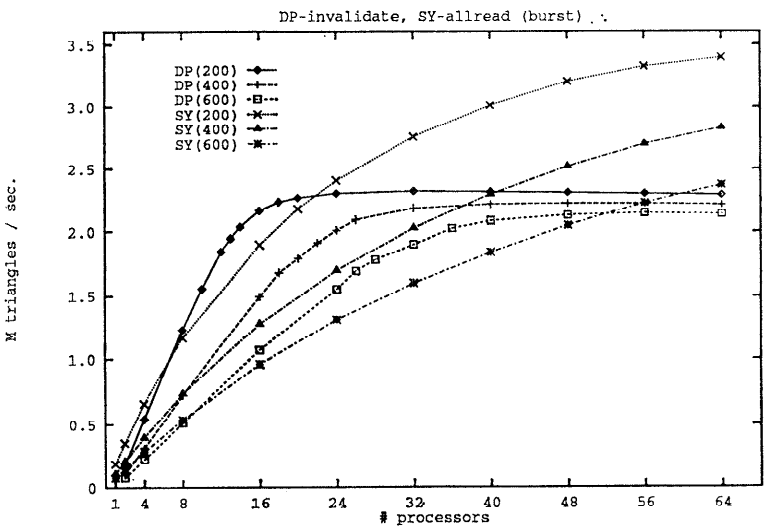


図 9 DP 方式とシンメトリ方式の結果（バースト転送）
Fig. 9 Comparison of DP method and Symmetry method (both burst transfer).

ため，プロセッサあたりの実効処理性能が低下する．

属性指示が頻繁に現れるモデリングデータに対応した評価を行うために，今回用いた cyl_head のデータの各プリミティブに7ワードの属性設定指示を付加した人工的なモデリングデータを用意して，シミュレーションを行った（**図10**）．なお，属性指示の頻度の高いケースは高品位画像が求められていると見做して，高品位画像に対するシミュレーションのみを対象とした．図の w/attr で示される曲線が属性設定指示を付加したものである．属性設定指示が追加された場合，シンメトリ方式の性能低下が顕著であり，プロセッサ台数が多くなると DP 方式が常にシンメトリ方式よりも高い性能を示す．

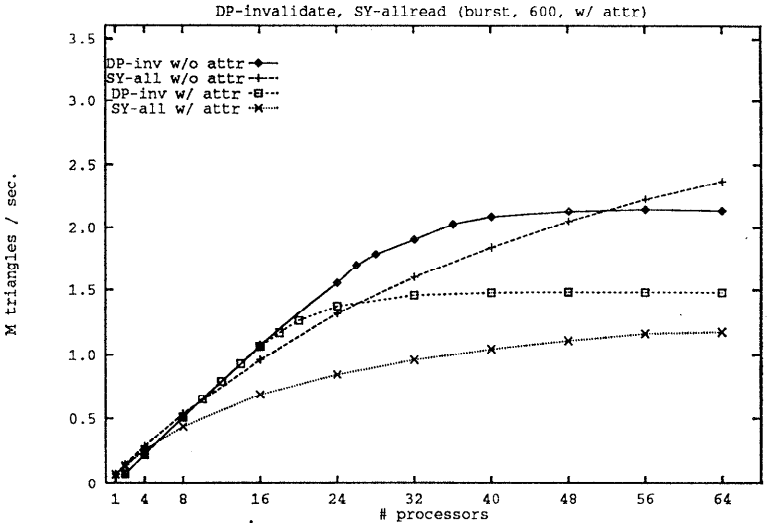


図 10 属性設定指示を付加した場合
Fig. 10 Effect of attribute processing overhead.

これは，シンメトリ方式の場合，属性設定指示の処理に対しては台数効果が現れないため，台数効果が得られるプリミティブ処理の時間が相対的に低下して，全体の性能向上比が上がらなくなるためである．

これに対して DP 方式ではディスパッチ能力が飽和しやすくなるが，飽和地点までの台数効果は本質的に変わらない．従ってディスパッチ能力を高めることにより，属性設定指示が増えた場合での性能低下を押える余地はある．

5.4 DP 方式の改善

前節で述べたように，DP 方式ではプリミティブごとのディスパッチのコストが DP に集中し，しかもこのコストの大きさがトラバースのコストとほぼ同じかそれ以上かかっている．

そこで，ジオメトリ演算部専用のマルチプロセッサハードウェアを作製する場合は，このディスパッチのコストを減らすために，プロセッサからバスを使わずに直接 1-to-1 で演算終了を DP に知らせる信号線，その信号を DP 側で到着順（FIFOオーダ）に整理するハードウェア，さらに DP から演算開始を 1-to-1 で知らせる信号線（演算開始の通知は ABUS が存在する場合，流用できるため必要ない）を実装すると，性能が大幅に改善し投資効果があると考えられる．

このようなハードウェアによるディスパッチ支援機構を仮定し，DP 方式を改良した方式（**HD 方式**）でのシミュレーションを行った．

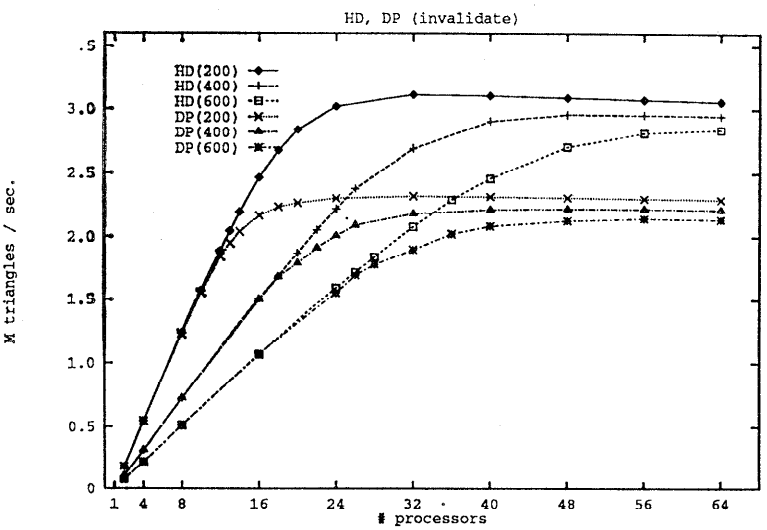


図 11 ディスパッチ支援機構を付加した場合
Fig. 11 Effect of hardware dispatch support.

ここでは，DP とそれ以外のプロセッサの間を1ビットの双方向信号線で繋ぎ，DP は，それぞれ2クロックずつで，到着順に整理するハードウェアから先頭の空きプロセッサの ID を得ることと，プロセッサへの選択を示す信号を送ることができるものとする．ディスパッチ支援機構を使った場合のシミュレーション結果と DP 方式の比較を示す（**図11**）．ディスパッチ支援機構により，ディスパッチのコストが大幅に減少し性能が向上することがわかる．

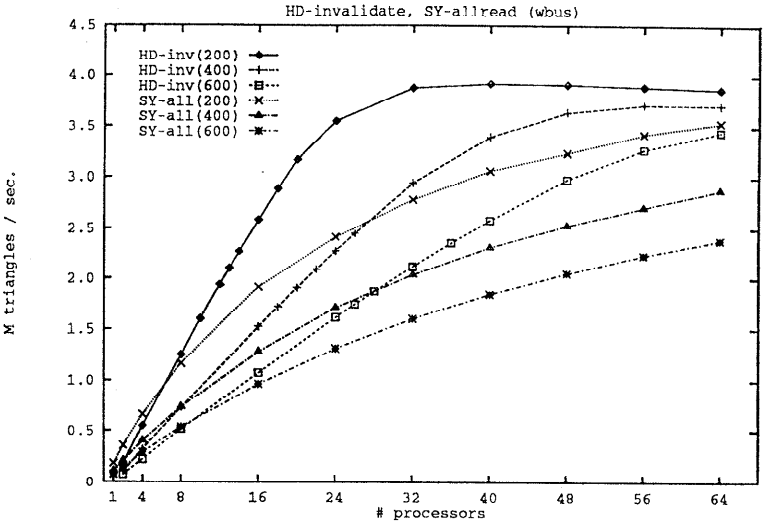


図 12 128 bit 幅バスでディスパッチ支援機構を付加した場合
Fig. 12 Effect of hardware dispatch support and 128 bit width bus.

DP 方式においては 5.1 節で示したように 128 bit 幅バス転送を用いることで処理速度が向上する．そこで，ディスパッチ支援機構と 128 bit 幅バス転送を組み合わせた場合の結果が**図 12** である．比較のために，シンメトリ方式で 128 bit 幅バス転送を用いた場合も図示してある．シンメトリ方式の場合，バスが飽和しているわけではないので 128 bit 幅バス転送を用いても大きな性能向上は無いが，HD 方式の場合メモリアクセスのレイテンシが減少するためにディスパッチ性能が向上し，全体としてシンメトリ方式よりも高い性能を示すようになる．

ただし，HD 方式（および DP 方式）の場合 TTS 等の専用ハードウェアを必要とすることに加え，マルチタスク（マルチウィンドウ）を行う場合，コンテクストスイッチの際に TTS の管理等の付加的な処理が発生する．

6. おわりに

PHIGS を採用した高性能グラフィックスシステム特にそのジオメトリ演算の並列処理について，これまでに提案した処理方式および機構の定量的効果を実行駆動型のかなり精密なシミュレーションによって測定を行った．

本論文で使用した cyl-head のような単純なモデリングデータでも，従来のマルチプロセッサを利用した並列処理の場合は，様々な制約で実効性能が低く抑えられていた．しかし，提案した機構を用いることで，実装が容易で汎用性の高い共有メモリ共有バス型マルチプロセッサにおいて，従来（2 M triangles/sec 程度の性能）を大きく上回る性能までスケールアップが可能になることが示せた．

グラフィックス専用のハードウェアの付加を回避した汎用指向のシンメトリ方式でも，all-read キャッシュプロトコルを使用することでプロセッサ台数に対するスケーラビリティが確保できることがわかった．ただし，属性設定指示が増加してくると台数効果が効かない処理が増加してくるので性能向上比が上がらなくなる．

これに対して，簡単なグラフィックス専用ハードウェアを付加してトラバースとタスクのディスパッチを1台のプロセッサに行わせる DP 方式の場合，属性設定指示の多いモデリングデータに対しても対応できることを示した．また，プロセッサ台数が増加するとディスパッチの処理を行うプロセッサが全体性能のボトルネックとなることが判明した．しかし，簡単なディスパッチ支援機構を付加することによって性能を改善できることが明らかになった．

本論文で述べた方式は PHIGS 固有の問題を解決することを目的としたが，属性設定指示と描画指示の処理がベースであるグラフィックスインタフェースではスナップショットの処理は共通の問題であり，PHIGS 以外のインタフェースでも本方式が適用可能である．

今後は，属性設定指示が頻繁に構造体内に出現するより一般的で複雑なモデリングデータを使った場合の

性能評価をさらに進めていく予定である.

謝辞 PHIGS に関して御指導頂いた IBM 東京基礎研究所の清水和哉氏に感謝いたします.

参考文献

1) 松本, 川瀬, 森山: PHIGS の構造体を処理するジオメトリ演算部の並列アーキテクチャについて, グラフィックスと CAD シンポジウム論文集, 情報処理学会, pp. 191-200 (Nov. 1991).
2) ISO/IEC 9592-1 : 1989 (E), Information processing systems—Computer Graphics—Programmers Hierarchical Interactive Graphics System (PHIGS), Part 1—functional description (1989).
3) The graPHIGS Programming Interface Understanding Concepts Version 2, Release 1.0, IBM Corp. (1989).
4) Kirk, D. and Voorhies, D.: The Rendering Architecture of the DN 10000 VS, *ACM Comput. Gr.*, Vol. 24, No. 4, pp. 299-308 (1990).
5) Horning, R. et al.: System Design for a Low Cost PA-RISC Desktop Workstation, *Proc. of COMPCON 91 SPRING, IEEE*, pp. 208-213 (Feb. 1991).
6) 松本 尚: 細粒度並列実行支援機構, 情報処理学会計算機アーキテクチャ研究会報告 No. 77-12, pp. 91-98 (1989).
7) 松本 尚: 細粒度並列実行支援マルチプロセッサの検討, 情報処理学会論文誌, Vol. 31, No. 12, pp. 1840-1851 (1990).
8) Matsumoto, T. et al.: MISC: A Mechanism for Integrated Synchronization and Communication Using Snoop Caches, *Proc. of the 1991 Int. Conf. on Parallel Processing*, Vol. 1, pp. 161-170 (Aug. 1991).
9) 日本電気: μPD 30300 (VR 3000) 32 ビット・マイクロプロセッサ ユーザーズ・マニュアル アーキテクチャ編, 日本電気 (1989).
10) Graphics Performance Characterization Picture Level Benchmark Program—Benchmark Interchange Format—Release 1.1, GPC Committee (1990).

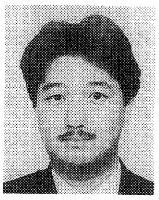
(平成4年9月16日受付)
(平成5年2月12日採録)

松本 尚 (正会員)

1962 年生. 1985 年東京大学工学部計数工学科卒業. 1987 年大阪市立大学大学院理学研究科物理学専攻修士課程修了. 日本アイ・ビー・エム(株)東京基礎研究所研究員を経て, 1991 年 11 月より東京大学理学部情報科学科助手. 並列計算機アーキテクチャ, オペレーティングシステム, 最適化コンパイラに関する研究に従事. 他にニューラルネットワーク, 学習, 力の超統一理論等に興味を持つ. 1990 年本学会学術奨励賞受賞. 電子情報通信学会, 日本ソフトウェア科学会, ACM 各会員.

川瀬 桂

1961 年生. 1985 年早稲田大学理工学部機械工学科卒業. 1987 年同大学院前期課程修了. 同年日本アイ・ビー・エム(株)に入社. 現在同社東京基礎研究所に勤務. グラフィックスの並列処理の研究に従事.



森山 孝男 (正会員)

1962 年生. 1985 年東京工業大学工学部情報工学科卒業. 1987 年同大学院修士課程修了. 同年日本アイ・ビー・エム(株)に入社. 現在同社東京基礎研究所に勤務. 並列マシンのオペレーティングシステム, グラフィックスの並列処理の研究に従事.