M-015

オープンソースグリッドによる CG の並列計算
Parallel Computing of CG using Open Source Grid

田中堅一† 上原 稔‡ 森 秀樹‡
Kenichi Tanaka Minoru Uehara Hideki Mori

1. はじめに

グリッド・コンピューティングは近年商用製品が出現し、またその製品を使用したグリッド・システムの開発も行われるようになった(文献[1])。

製品によるグリッドは、安定して動作しメーカーのサポートも受けられるといった利点があるが、導入にかかるコストも低くはない。コストの面ではオープンソースのソフトウェアを用いることで抑えることができる。

本研究ではオープンソースソフトウェアを用いて 3DCG レンダリングを対象としたグリッド・システムを構築し、その性能特性や問題点を明らかにすることを目的とする。

2. 概要

システムは各ノードへのジョブの振り分けや画像の結合を行うマスターノードと、実際にレンダリング処理を行うスレーブノードから構成されている (図 1)。

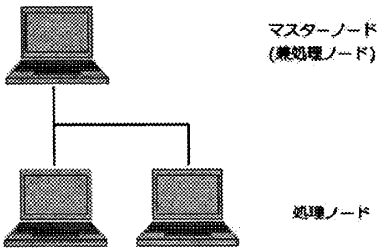

図 1 システム構成図

表 1 システムのスペック

OS	Windows XP Pro x64	Windows Server 2000 x64
CPU	AMD Athlon 64 X2 Dual Core、2.00 GHz	
メモリ	1.93 GB	
台数	2 台	1 台

実験時には同性能の PC が 3 台に限定されていたため、マスターノードは処理ノードを兼ねるようにした。

このシステムに対しクライアントは、モデルの記述がされたファイルとレンダリング結果の画像サイズをマスターノードに渡す。マスターはモデルを接続済みのスレーブに対し転送した後に処理範囲を分割し、各スレーブに範囲を割り当て、レンダリングの実行を指示する。各スレーブが処理を終えた時点でその結果をマスターが受け取り、画像の結合を行う。最後に、すべての分割された領域を処理し終えた時点で結合した画像をクライアントへと渡す。

† 東洋大学 大学院 工学研究科 情報システム専攻
‡ 東洋大学

3. 実験環境

本実験ではグリッド構築のためのミドルウェアとして Globus Toolkit 4、CG をレンダリングするためのソフトウェアとして Blender 2.43 と YafRay 0.0.9 を使用した。

3.1 Globus Toolkit

Globus Toolkit (GT) は Globus Alliance によって開発されているオープンソースのミドルウェアである。GT のコア機能は Java によって実装されおり、Windows の上でも使用できる。今回はこのコア部分を用いてグリッド・サービスを構築した。

3.2 Blender

Blender はオープンソースの 3DCG ソフトウェアである。もともとはオランダのアニメーションスタジオ NetGeo 社によって開発された商用製品である。その後 Not a Number (NaN)社に引き継がれたが、NaN 社の倒産に伴い、有志によって GNU GPL に基づくオープンソースソフトウェアとなった。

Blender の持つ特徴として、Python によるスクリプトの実行がある。Blender 自体に Python インタプリタが内蔵されており、Python によって Blender を操作することや、Python プログラムを Blender から実行することもできる。

3.3 YafRay

YafRay は LGPL 準拠のオープンソースのレンダラーである。Blender に統合されており、Blender 内部から使用することが可能である。

YafRay 自体はレイトレーシングを用いたレンダラーで、コマンドラインプログラムである。モデルは XML 形式で渡される。Blender は YafRay の設定に対するユーザーインターフェイスを提供し、一時ファイルとして YafRay の XML ファイルを生成する。これを YafRay に渡すことでレンダリングを行っている。

4. 実験結果

本実験では画素数が 640×480、800×600、1024×768、1280×960、1600×1200 である画像を、処理ノード数 N=1、2、3 台のそれぞれの場合でレンダリングを行い、その応答時間を測定した。レンダリング時の画像の領域分割は分割しない場合から 16 分割までの場合を測定した。

図 2 に処理ノード数が 3 の場合の測定結果を示す。この図は横軸に画素数、縦軸に応答時間をとり、分割数ごとに散布図として表したものである。この図から応答時間はほぼ画素数に比例することがわかる。ここで、一つの処理ノードで処理する範囲を縮小、すなわち処理対象の分割数を増やした場合に応答時間が増加している。これは処理範囲縮小による各ノードにおけるレンダリング処理時間の減少よりも、処理範囲の分配や画像の結合といった並列処理が

できない部分の処理時間が増大しているためと考えられる。特に部分ごとに処理された画像を結合する処理は、分割された画像の枚数と画素数に比例して時間がかかる。この部分が全体の応答時間に影響を及ぼしていると考えられる。

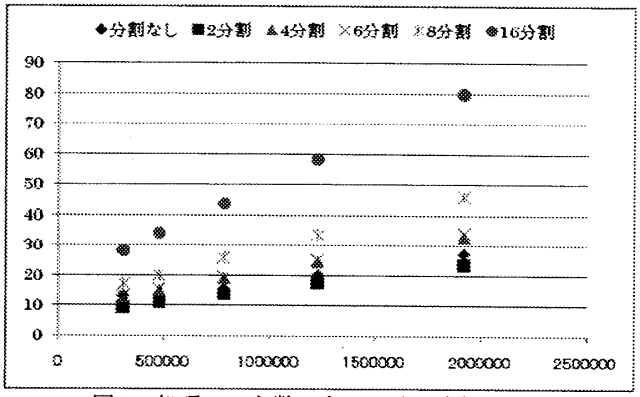

図 2　処理ノード数 3 台における応答時間

表 2　画素数と分割数に対する応答時間 (単位は秒)

	画素数			
分割数	307200	480000	1228800	1920000
1	13.073	11.630	19.984	27.083
2	9.143	10.872	17.374	23.515
4	12.545	14.864	24.451	32.919
6	12.841	15.223	24.976	34.039
8	16.861	19.958	33.434	46.106
16	28.140	34.057	58.390	79.932

次に、画素数ごとに処理ノード数による応答時間を比較した図を示す。

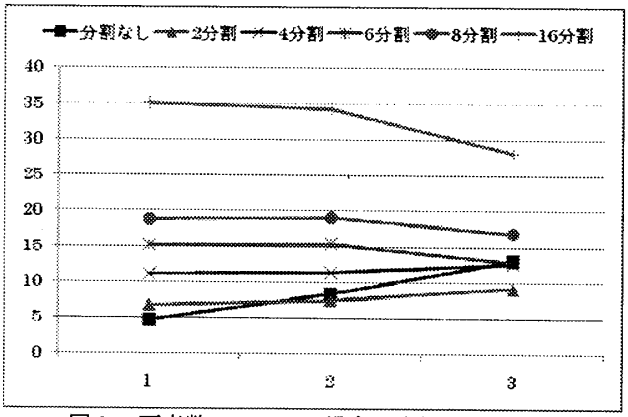

図 3　画素数 640x480 の場合の応答時間の変化

この結果から画素数が少ない場合はグリッド化してもあまり応答時間に変化がないことがわかる。つまり、ある画素数以上の画像を処理しない限り、グリッド化による効率化は見込めないということである。また、分割数による影響もあり、処理ノード数が増加しても線形に応答時間が減少していない。

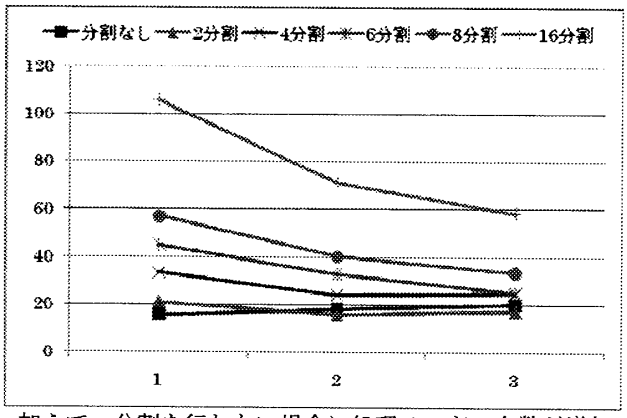

図 4　画素数 1280x960 の場合の応答時間の変化

加えて、分割を行わない場合に処理ノードの台数が増加すると、応答時間がわずかながら増加するという結果が得られた。これはシステムの構成上、モデルを記述したファイルを処理ノードすべてに転送しており、この処理がオーバーヘッドとなっていると考えられる。

5. まとめ

本実験では画素数が多い画像ほどグリッド化による効率化が図れることがわかった。しかし、応答時間は処理範囲を狭めることによる処理時間の短縮よりも、分割された画像の結合処理による処理時間増加のほうが大きく、分割すればするほど応答時間が増加するという結果だった。

処理ノードに対しレンダリングを指示する回数 C は、処理ノード数を N、領域分割数を D として天井関数 ceiling(x) を用いると、

$$C = ceiling(D / N)$$

と表すことができる。この式から、もっとも最適な状態は、処理ノード数と同じ数に領域を分割した場合であり、その場合の C は 1 である。分割数が処理ノード数を上回ると C が 2 以上となり、何度もレンダリングを行わせる必要がある。

これらのことから、処理ノード数が十分であれば、応答時間の減少が期待できる。特に、画素数が多い画像ほど応答時間の減少分が大きく、グリッド化による性能向上を実感できる。また、画像の結合処理部分を改良することで、さらなる性能向上も期待できる。今後は性能の違う PC をグリッドに組み込み、ヘテロジニアス環境での性能評価を行う予定である。

参考文献

[1] 草薙 信照、“グリッド技術を用いた GIS 処理の制御と効率化”、情報処理学会第 69 回全国大会
[2] Globus Toolkit、http://www.globus.org/
[3] Blender、http://www.blender.org/
[4] YafRay、http://www.yafray.org/