オープンソースWindowsグリッドによるCGの並列計算

田中堅一　　上原 稔　　森 秀樹
東洋大学大学院工学研究科情報システム専攻

近年、コンピュータが扱う情報は増加し続けており、それを処理する技術に対するニーズが高まっている。グリッドとは主にHTC (High Throughput Computing) をねらう分散コンピューティングの一形態であり、HPCをねらうクラスターに対し低コストで構築できる可能性を持つ。加えてグリッドにはオープンソースによるミドルウェアも存在する。しかしながら従来のグリッドはWindowsマシンでは十分に活用することは困難であった。そこで、我々はWindowsマシンによる安価なグリッドの利用法を研究している。本論文では、最近増えつつあるマルチコアプロセッサを用いたWindowsグリッドについて評価を行う。

Parallel Computing of CG using Open Source Windows Grid

Kenichi Tanaka　　Minoru Uehara　　Hideki Mori
Toyo University Graduate School, Department of Open Information Systems

Recently, the information which the computer handles continues to increase, the needs for the technology which processes that have increased. The grid HTC (High Throughput Computing) is one form of the distributed computing which mainly aims for, it has the possibility where it can construct low at cost vis-a-vis the cluster which aims for HPC (High Performance Computing). In addition there are existed open source middleware in grid. But as for the former grid in the Windows machine as for utilizing sufficiently it was difficult. Then, we have researched the application of the cheap grid by the Windows machine. In this paper, we describe the Windows grid which uses the multiple core processor which recently is increasing.

1. はじめに

グリッド・コンピューティングというものはネットワークを通じて計算資源を共有し、利用者から見れば一つの巨大な仮想コンピュータのように利用するものである。このとき、共有されるものはディスク資源や物理メモリ、そしてCPUなどである。

近年ではグリッド関連の研究も進み、富士通のSystemwalker CyberGRIP (文献[1]) のような商用のグリッド・ミドルウェアや、IBMのアプリケーションサーバであるWebSphere Application Server (文献[2]) のようにグリッド技術をWebサービス・アプリケーションの資源融通に利用した製品も登場してきている。さらに、このような商用製品を使用してのグリッドの研究を行った論文も発表されている(文献[3])。

しかしながら、グリッド・コンピューティングのプロジェクトは主にUnix / Linuxプラットフォームにおいて行われている。そのためWindowsプラットフォームにいてグリッド・システムを構築する方法は限られている。前述のCyberGRIPは計算サーバとクライアントとしてWindowsをサポートしているが、マスターサーバはSolaris / Linuxのみである。

さらに商用のミドルウェアでは導入するためのコストも考慮する必要がある。メーカーによるサポートがあるメリットはあるが、それでもかなりのコストである。CyberGRIPを例にとれば、計算サーバで100,000円/台程度、マスターサーバとなると1台あたり1,600,000円ほどとなる。これらのコストを削減する方法として、オープンソースのミドルウェアを使用するという選択肢がある。管理コストや初期導入コストは多少掛かるが、商用製品を導入するよりははるかに低コストでグリッドを構築することができる。

また近年では、映画やアニメーションの世界においてCGを使用する作品も多く、中にはほぼすべてをCGによって製作された映画もある。これらのCGを作成するためには高速なレンダリング処理が必要不可欠である。このような理由から本研究では、CGレンダリングを検証用アプリケーションとして使用する。

ところで、今日ではデュアルコアやクアッドコアといった、マルチコアのCPUが登場してきている。これらは一つのCPU内部に複数のCPUコアを内蔵し、一つのCPUでありながらシステムからは複数のCPUとして認識される。そこで、グリッド環境においてこのマルチコアCPUを複数のCPUのように用いれば、1台のホストに複数台分の計算能力を持たせることができる可能性がある。

本研究ではオープンソースのグリッド・ミドルウェアを用いたWindows Gridの構築及び運用を対象としているものである。文献[4]においては、Windowsプラットフォームにおいてオープンソースソフトウェアによるグリッドを構築し、その性能特性の評価を行った。今回は3DCGの並列計算を行うシステムを一部改造し、デュアルコアプロセッサが2台分の計算能力を発揮しうるかどうかを検証する。

2章では本実験において使用したソフトウェアについてその概要と特徴、及び本研究における位置付けを示す。3章では、それらのソフトウェアによって構築されたCG並列計算システムについて示す。

4 章にシステムの評価環境及び評価結果を示す。5 章で評価結果に対する考察を行う。そして、総合的なまとめを 6 章で行う。

2. 関連研究

本実験において使用したソフトウェアは、すべてオープンソースソフトウェアである。グリッド・ミドルウェアとして Globus Toolkit 4.0.4、3DCG モデリングとレンダリングエンジンに Blender 2.43 と Python 2.4.3 を使用した。また、Globus のコンテナに必要な Java は 1.6.0 を使用した。

2.1 Globus Toolkit

Globus Alliance によって開発されているオープンソースのグリッド・ミドルウェアが Globus Toolkit である。最新は 4.0.5 である。WS-Resource Framework を利用することにより、Web サービス開発の技法を利用したサービス開発が可能である。

ツールキットとしてのコア機能は Java で記述されており、Java の動作する環境ならグリッド・サービスを動作させることができる。

2.2 Blender

Blender はオープンソースの 3DCG モデリングソフトウェアである。もとは Not a Number Technologies (NaN) 社によって販売されていた製品であるが、NaN 社が倒産した後に、トン・ローセンダール氏によって設立された Blender Foundation によって現在開発されている。

特徴は独自のレイトレーサーを内蔵している他に外部のレイトレーサーである YafRay との親和性が高いことと、Python によるスクリプトの実行がある。前者は YafRay を直接 Blender から呼び出せることを意味する。後者は Python の知識があれば、Blender のプラグインの作成や、Blender 自体の制御が行える。

また、複数のフォーマットのインポート／エクスポートをスクリプトで拡張可能であり、他のモデリングソフトやレンダリングソフトとの連携が可能である。

2.3 Python

Python とは欧米で広く普及しているオブジェクト指向のスクリプト言語である。Guido van Rossum 氏を中心にオープンソースでの開発がされている。インデントによるブロック構造を持ち可読性に優れる一方、C/C++による拡張も可能で幅広い分野のアプリケーションを構築することができる。

本研究では、Blender を制御する Python スクリプトを生成し、それを Blender に渡すことでレンダリングを行った。

2.4 YafRay

YafRay は Blender の外部レイトレーサーとして利用できる、オープンソースのレイトレーサーである。Blender 側からは統合されていると見ることもでき、YafRay 専用のインターフェイスが用意されている。

特徴は、モンテカルロ近似法・準モンテカルロ近似法によるグローバルイルミネーション（周囲の拡散反射光も考慮し、より実世界に近い描画をする技術）を使用することが可能という点である。これにより非常に高画質な画像を生成することが可能である。

3. システム概要

本システムの概念図を図 1 に示す。クライアントがマスターに 3DCG モデルとともにジョブを投入すると、マスターがレンダリング領域を区切り、小さなジョブとしてスレーブノードへと投入する。なお、マスターは各ホストに 1 つのみ配備することができるという制約があるが、スレーブの配備数には特に制限は設けてはいない。

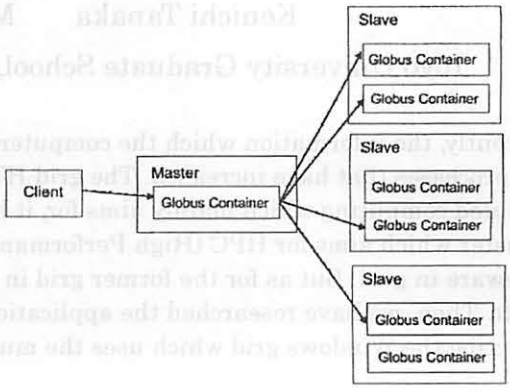

図 1 システム概念図

本システムではデュアルコアプロセッサ等のマルチコアプロセッサが、複数の処理ノードと同等の処理を行えるかを検証するため、同一ホスト内で複数のコンテナを動作させる。しかし、実験に使用可能な同性能のマシンの台数に制限があり、実際に構築されたシステムは、純粋な処理ノードとして 2 台を用い、1 台をマスターノードと処理ノードを兼用し、なおかつクライアントも兼ねるという変則的な構成となった。この構成によるマシン間の関係は図 2 の通りである。

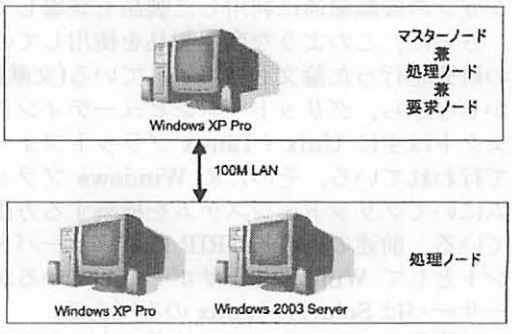

図 2 システム構成図

図 2 におけるマスターノードには、マスター用のコンテナとスレーブ用のコンテナを 2 つ動作させている。

本システムは Globus Container 上で動作する Web

サービス技術によるグリッド・サービスとして開発した。実装言語はJavaを使用した。

スレーブノードに配置するサービスはBlenderService、マスターノードに配置するサービスはBlenderRootServiceとして開発した。ただし、BlenderServiceは文献[4]において開発したシステムで使用したものと同一であり、本研究で開発した部分はBlenderRootServiceである。

BlenderRootServiceの構成は次のようになっている。

・grid.blender.master、grid.blender.master.implパッケージ
・grid.blender.master.clientパッケージ
・grid.blender.master.stubs以下のパッケージ

grid.blender.masterにはサービスインターフェイスが含まれ、その実装がgrid.blender.master.implパッケージに含まれる。また、grid.blender.master.stubs以下は、グリッド・サービスのビルド時に自動的に生成されるパッケージであり、このパッケージによってサービスのパラメータなどをSOAPによって通信することができるようにシリアライズしている。

grid.blender.master.clientはサービスに処理を依頼するクライアントクラスのパッケージであり、今回はBlenderRootClientクラスのみを定義してある。

このシステムにおける処理は次の順に行われる。

1) 処理可能なスレーブのURIをマスターに送信する。
2) クライアントからマスターに処理する3DCGモデルを転送する。また、画素数と領域分割数を縦方向と横方向に指定する。
3) スレーブのサービス・インスタンスを生成する。このとき画素数やレイ・トレーシング処理の共通なプロパティもセットする。
4) 各サービス・インスタンスに3DCGモデルを転送。
5) レンダリング領域の分割を行う。分割された範囲はキューに入れておく。
6) レンダリング領域をキューから取り出し、処理可能なサービス・インスタンスへ処理の実行を依頼する。なお、処理可能なノードがない場合は、ノードの処理終了まで待機する。キューに領域がなくなればレンダリングは終了する。ただし、各サービス・インスタンスの処理終了は、レンダリング後の部分画像を受け取るまでとしている。これは、連続して同一ノードがレンダリングを行った場合に、前回の結果を上書きする可能性があるためである。
7) 部分画像を分割された範囲に基づいて結合する。
8) 結合された画像をクライアントに送信し、ジョブを終了する。

4. 評価

4.1 評価環境

実験に使用したマシンの性能を表1に示す。

表1 実験マシンのスペック

OS	Windows XP Pro x64	Windows Server 2003 x64
CPU	AMD Athlon 64 X2 Dual Core、2.00 GHz	
メモリ	1.93 GB	
台数	2台	1台

これらのマシンのうち、Windows XP Professional x64の1台をマスターとスレーブを兼用した。また、各マシン間をつなぐネットワークは100Mbps LANを使用した。

4.2 評価

評価として図3に示す画像のレンダリングを行う。この評価画像ではジョブに割り当てられた領域によって単純な部分と複雑な部分で偏りが発生するが、今回の評価では単純に処理の終わったノードに次のジョブを割り当てるため、この領域による偏りは考慮されていない。

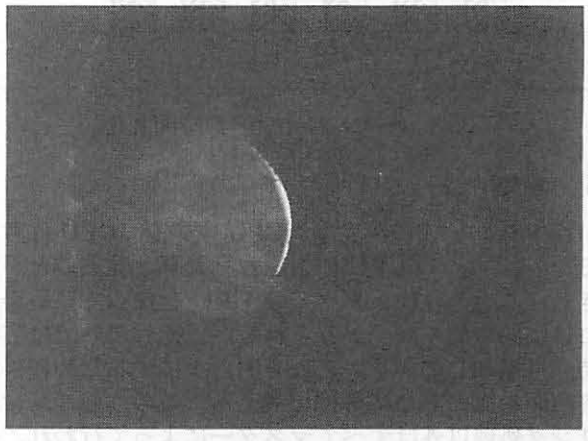

図3 評価画像

全体的な応答時間の変化を図4, 5に示す。図4は1台のマシン上で処理させたものであり、図5は2台のマシンを用いたものである。それぞれレンダリングに使用した画像の画素数は640×480である。また、縦軸が応答時間（ミリ秒）で、横軸がレンダリング領域の分割数である。

図4, 5の応答時間に注目すると、1台の場合よりも2台の場合のほうが短くなっており、概ねデュアルコアによる性能向上が認められる。各応答時間の向上率を求めると、およそ1.20から1.75の範囲である。したがって、デュアルコアによる応答時間の向上はあるといえる。

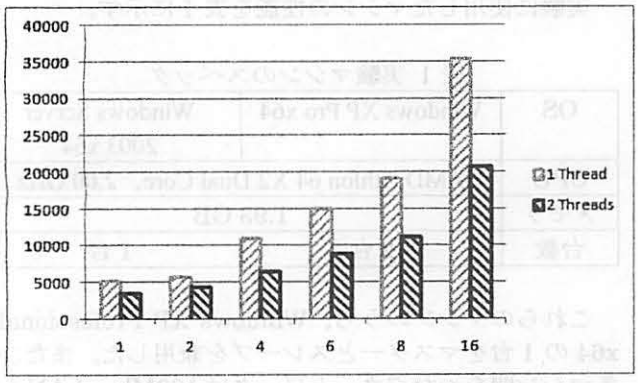

図 4　マシン 1 台の応答時間変化

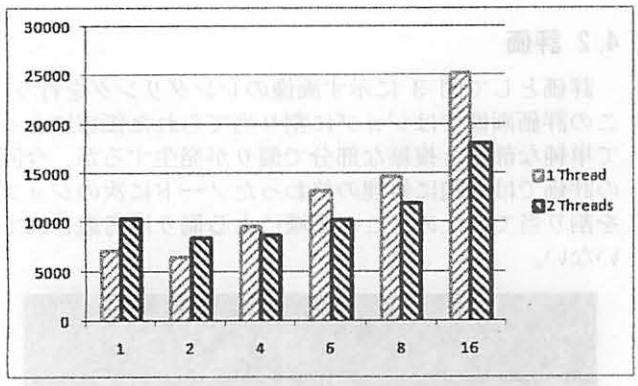

図 5 マシン 2 台の応答時間変化

5. 考察

図 5 においての環境は、CPU コアを 1CPU と考えるならば 4 つの CPU による並列計算と見ることができる。このときに、すべての CPU を使っている場合には応答時間の減少がみられるが、それ以外では応答時間は増加している。これを明らかにするために、応答時間を次の 5 つに分けて考える。このうち、レンダリング処理以外はすべてマスターによって行われ、それらは逐次処理である。

・ノード初期化時間
・3DCG モデル転送時間
・レンダリング領域分割時間
・レンダリング処理時間
・画像結合処理時間

5.1　初期化時間

初期化時間は、システム内におけるスレーブとの接続を確立し、共通のレンダリング設定を適用する処理である。この処理時間を計測したものを表 4 に示す。なお、単位はミリ秒である。さらに、比較のために画素数が 800×600 と 1024×768 で、マシン 3 台の場合を含めた初期化時間を表 5 に、表 6 にマシン 1 台での場合を示す。

これらの表をみると、レンダリング領域の分割数が無関係であることがわかる。また、表 5 から画素数にも無関係であることがわかる。事実、この処理では領域分割や画素数にかかわる処理は行っていないので、このようになっていない方が不自然である。

表 4　初期化時間（640×480）

分割数	2 Hosts, 1 Container	2 Hosts, 2 Containers	3 Hosts, 2 Containers
2	1765	3741	6872
4	1906	3597	6883
8	1760	3619	6830
16	1786	3673	6749

表 5 初期化時間（マシン 3 台、コンテナ各 2 つ）

	3 Hosts, 2 Containers		
分割数	640x480	800x600	1024x768
2	6872	6676	6961
4	6883	6744	6866
8	6830	7126	6687
16	6749	6822	6791

表 6 初期化時間（マシン 1 台、コンテナ各 2 つ）

	1 Host, 2 Containers	
分割数	640x480	800x600
2	184	244
4	242	286
8	170	253
16	229	176

また、マシンの台数が同一であっても各マシンで動作させるコンテナの数が異なると初期化時間が大幅に異なる。したがって、初期化時間は起動しているコンテナの数に比例することが予想される。また、マシン 3 台でコンテナが 2 つの場合の初期化時間は、640×480 をひとつのマシンで処理させた場合の応答時間を上回っている。このことから、コンテナを増加させた場合には初期化時間が問題となることが予測される。

5.2　3DCG モデル転送時間

図 6 はホスト数によるモデル転送時間の変化である。ただし、各ホストマシンは 2 つのコンテナを動作させているので、正確には、コンテナの数が 2, 4, 6 の場合である。横軸は画像の分割数である。

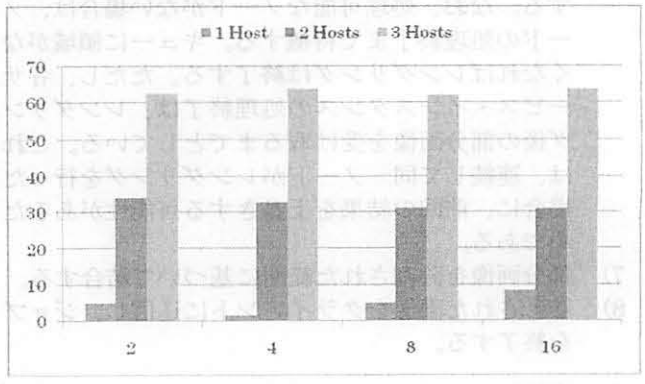

図 6　モデル転送時間（ミリ秒）

図6のようにモデル転送時間は分割数にかかわらずほぼ一定の値を示す。初期化時間とモデル転送時間は、現在の実装上すべてのコンテナに対して行われる処理であるため、コンテナの数によって処理時間が決定されるのは明らかである。また、モデルは描画される物体の形状を定義するものであり画素数は含まれないので、画素数に対しても無関係である。それは、次の図7からもわかる。

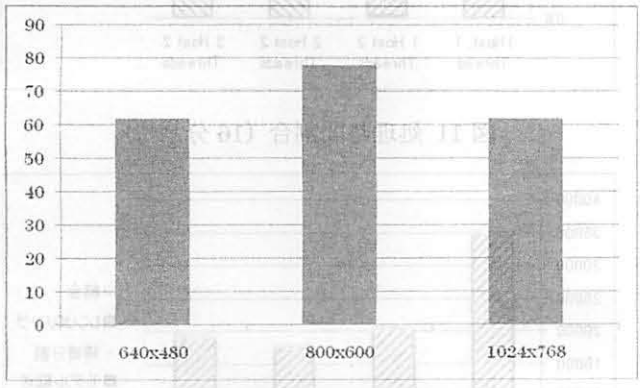

図7 画素数ごとのモデル転送時間

図7では画素数が800x600の場合に最も時間がかかっている。しかし、それよりの画素が多い1024x768の場合と、少ない640x480の場合の転送時間がほぼ同じである。したがって、画素数には無関係に転送時間が決定されていると結論付けることができる。

5.3 領域分割時間

領域分割時間について測定した結果、すべてのパターンにおいて0ミリ秒のみが結果として得られた。したがって、2〜16分割においてはほとんど時間のかからない処理であり、これよりも分割数を多くしても応答時間に影響をほとんど与えないと考えられる。

システム内部ではこの分割は倍精度浮動小数点数によって計算されている。実行されているのは、分割された領域のX、Y方向の比率の計算と、それによる領域の座標をキューに入れることである。これらの処理は分割数に比例するはずであるが、計測結果がすべて0ミリ秒であったことから、ほんのわずかの時間で終了しているとしか考えることはできない。

したがって、領域分割時間はホストマシンのCPU性能と、コンテナを動作させている Java 仮想マシンの浮動小数点演算速度によって変動する。

5.4 レンダリング処理時間

図8は、レンダリング処理時間の変化を離散図でプロットしたものである。横軸はマシンの台数、縦軸が処理時間（ミリ秒）である。なお、各マシン上ではコンテナを2つ動作させているので、最大で領域6つまで並列計算することができる。

図8の2 Divisionsと4 Divisionsを見ると、マシン台数が2台と3台ではほぼ同じ処理時間であることがわかる。これはすなわち、一度に実行できるレンダリング処理数以内の領域分割数であれば、ほぼ同じ処理時間で済む、ということを示している。表8の計測データ上でも、2台以上使用する場合2分割と4分割は近い結果を示す。

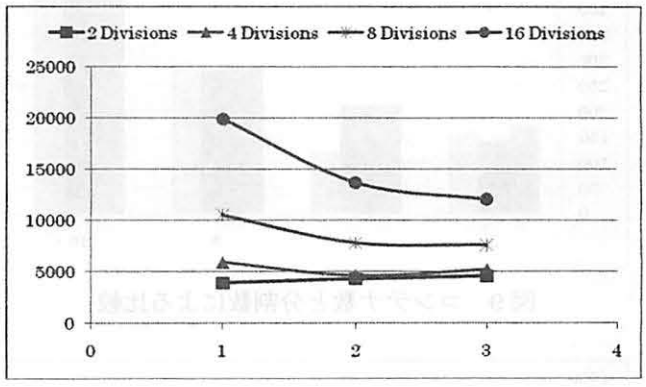

図8 レンダリング処理時間（640x480）

また、分かりやすい結果となっているのは8分割や16分割といった、分割数が多いものである。同時に処理すべきジョブの数が多いため、スレーブノードの増加の恩恵を受けやすいためである。すべてのスレーブノードが平均的にジョブを処理すると仮定すれば、この処理はジョブ数をD、スレーブノード数をNとすればCeiling(D/N)に比例する。

ただし、分割数が増加するとレンダリング処理時間が増加する傾向にある。分割数が増加すると単位ジョブあたりの仕事量は減少し、処理時間は減少する。しかし、同時に実行すべきジョブ数が増加する。そのため、単位ジョブあたりの仕事量減少による処理時間短縮よりも、ジョブ数増加によるオーバーヘッド増加の方が大きいために、分割数の増加に伴ってレンダリング処理時間が増加していると考えられる。

5.5 画像結合時間

画像結合時間は、領域分割数と画素数に比例し、コンテナの数には無関係である。図9にホスト（コンテナ）数と分割数によってプロットした結合時間のグラフを示す。また、画素数との関連性を示すグラフを図10に示す。

まず、図9を見る限りでは、領域分割数による変化は認められるが、ホスト数、すなわちコンテナ数に対する関連性は認められない。画像結合は部分的にレンダリングされた画像をつなぎ合わせ、本来の画像にする処理である。ゆえに部分画像の枚数に比例することは明らかである。したがって、結合時間は領域分割数によって決定されると結論付けられる。

図10では、画素数に比例する形で結合処理時間が増加している様子がみられる。また、同一画素数ではレンダリング領域分割数が多いほど結合処理に時間がかかることがわかる。

画素数が多いということは、処理しなければならない画素数が多いということと同義であり、画素数に比例して画像結合に時間がかかることは明らかである。

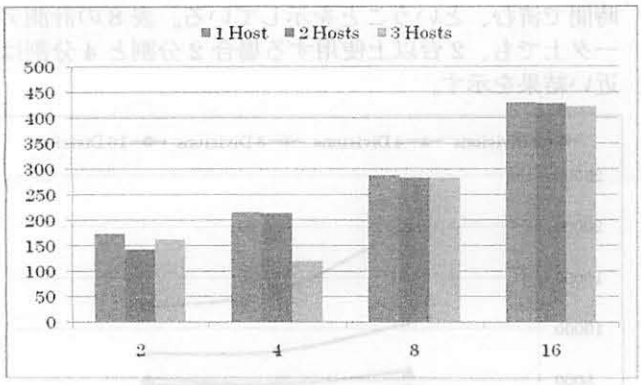

図 9　コンテナ数と分割数による比較

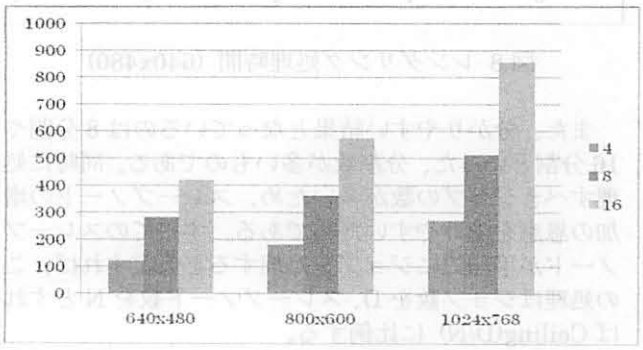

図 10　画素数による比較

領域分割数が多いということは、同一画素数の画像であれば一度に処理すべき画素数が少なくなっているということである。したがって、各部分画像に対する処理時間が減少し、それを増加した部分画像の枚数分行うので、結果としては、同一画素数であれば画像結合時間はそう変わらないということが予想される。しかし、計測データ上ではそのようにならず、分割数に比例する形をとっているということは、画像結合時におけるデータの読み取りと出力のオーバーヘッドが大きいか、部分画像のコピー処理時間の減少幅が期待よりも小さいと推測される。

5.6 スレッド数による処理時間割合

これまでは処理単位ごとに考察を行った。そこでスレッド数によってそれぞれの処理の割合がどのように変化しているか考察する。図 11 は各処理がどの程度の割合を占めているかを示す図である。

スレッド数の増加に伴い初期化処理の占める割合が増加していることが確認できる。初期化処理はマスターノードで行われるために並列化による効果はない。そのため、初期化処理の割合が増加するということは全体の応答時間に対する影響が大きく出るということである。このことは図 12 で確認できる。

図 12 の右 2 本の棒グラフに注目すると、レンダリング時間はあまり変化がないが全体の応答時間は 2 Hosts の場合よりも 3 Hosts の場合のほうが長い。これは初期化処理時間が 3 Hosts の場合のほうが長いからである。

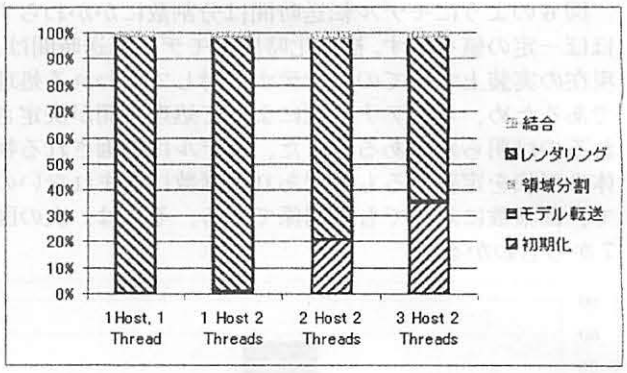

図 11 処理時間割合（16 分割時）

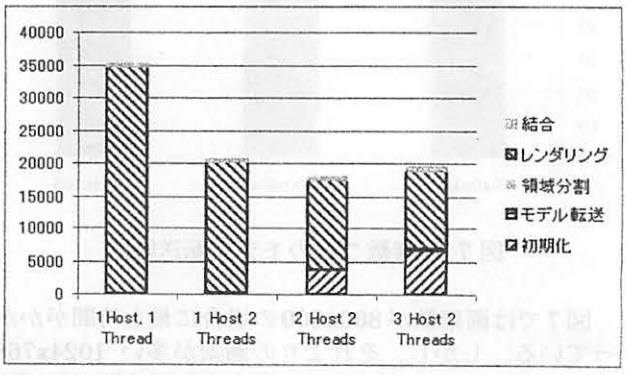

図 12 各処理時間の相対関係

6. おわりに

本研究はグリッド環境におけるデュアルコアマシンが、2 台のマシンのような性能を持つかどうかを検証するためのものであった。その結果は、2 台分までは届かないが、1 台分以上の性能は発揮する、というものである。

しかし、完全に 2 台分の処理をデュアルコアマシンが行うということは、物理的には不可能であるのではないかとも考えられる。CPU コアが 2 つであっても、使用するストレージやメモリ空間は OS によって管理され、その OS は常に一つのマシン上では一種類のみ稼働しているからである。

参考文献

[1] “Systemwalker CyberGRIP : 富士通”, http://systemwalker.fujitsu.com/jp/cybergrip/
[2] “IBM アプリケーション・サーバー”, http://www-06.ibm.com/jp/software/websphere/ft/was/
[3] 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、3-47
[4] 田中堅一、“オープンソースグリッドによる CG の並列計算”、FIT2007 第 6 回情報科学技術フォーラム
[5] “The Globus Alliance”, http://www.globus.org/
[6] “blender.org – Home”, http://www.blender.org/
[7] “Python Programming Language”, http://www.python.org/
[8] “Free Raytracing for masses – YAFRAY.ORG”, http://www.yafray.org/