Bulletin of Networking, Computing, Systems, and Software – www.bncss.org, ISSN 2186–5140
Volume 1, Number 1, pages 31–37, January 2012

A Study of Performances Considering Communications on a GPU Cluster

Takahito Kawakami and Shinichi Yamagiwa
School of Information
Kochi University of Technology/JST PRESTO
Kami, Kochi 782-8502 Japan

Abstract

Multiple processing nodes with GPUs connected by a high speed network organize a GPU cluster environment. It is expected to achieve very higher performance than the one of CPU-based cluster due to the drastic performance growth of the GPU. This paper focuses on the performance impact when the tasks are distributed by different assignments using MPI framework. Due to the difference between the local communication in a processing node and the remote one among the nodes, the performance is very likely to be changed Applying famous benchmarks available in the internet, this paper evaluates the performance impacts of different task assignments among the nodes on the 32-node Tesla-based GPU cluster.

1. はじめに

近年、PC(パーソナルコンピュータ) の低価格化・高性能化が顕著であり、PC を複数台ネットワークで接続し、並列計算による高速化を図るクラスタ [1] が用いられるようになっている。特に、本来グラフィックの計算を担当する GPU(Graphics Processing Unit) によって、CPU(Central Processing Unit) が行う計算を肩代わりする GPGPU[2] を応用した GPU クラスタ の利用が進んでいる。

一方で、GPU クラスタでは従来からの CPU クラスタ環境と同様に、複数の GPU にタスクを分割した場合、そのタスク間で通信が必要である。従って、GPU クラスタにも通信のボトルネックが存在している [3][4]。これは、ノード内部の CPU や GPU のデータ転送の速度に比べて、GPU クラスタのノード間を接続するネットワークの速度が圧倒的に遅いことに起因する。そのため、ノード間の通信システムが全体の計算能力を左右する要因となってしまい、CPU や GPU に潜在する計算能力を引き出すことが難しい。そこで、ノード間の通信性能が計算能力に与える影響を一般的なベンチマークソフトによる性能評価を行うことで、通信性能と CPU・GPU の持つ潜在能力の関係を明らかにする指標となる可能性がある。

本論文では、GPU クラスタにおいて、フリーで入手可能なベンチマークソフトを使い、性能を評価することを目的とする。ベンチマークソフトとして、姫野ベンチマーク・NPB(NAS Parallel Benchmarks)・NAMD を使用し、計算の並列性の度合いや計算を並列化する際のタスクの割り当て方を変化させ、ノード間の通信速度が与える影響を調査する。

2. GPU クラスタ

2.1. 構成

本論文で対象とする GPU クラスタを Figure 1 に示す。このノードの構成を Figure 2 に示す。

GPU クラスタは全 32 ノードからなっており、1 つのノードには Intel 社製の CPU である Intel XeonR Processor E5645 2.4GHz と、NVIDIA 社製の GPU である Tesla M2050[5] が 2 台ずつ搭載されている。その他、メモリとして 12GB の DDR3、記憶装置として 80GB の SSD を有している。ノード間の通信には、Infiniband QDR 40Gbps と Gbit Ethernet の 2 種類を使用することができる。GPU クラスタを利用した計算を行う場合は、Gbit Ethernet で接続された Headnode から各ノードに対してタスクをディスパッチすることによって実行する。

2.2. GPU クラスタで使われるソフトウェア

2.2.1. MPI での並列化プログラム. 複数のノードからなる GPU クラスタにおいて並列計算を行う場合、複数のノードで実行されるプロセス間においてネットワークを利用した通信を行う必要がある。このプロセス間の通信に利用されるのが MPI(Message-Passing Interface)[6][7] である。

MPI とは、プロセス間のメッセージの送受信を定義したものである。Message-Passing とあることからわかるように、Message-Passing モデルを採用しており、送信側・受信側の両方のプロセスを操作すること

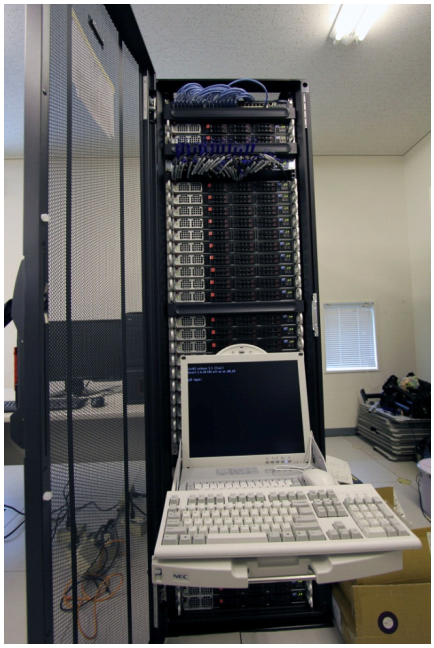

Figure 1. 本論文が使用する GPU クラスタ

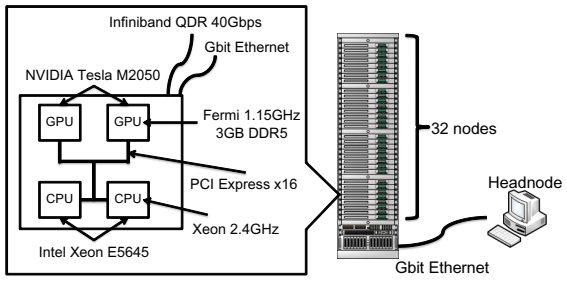

Figure 2. GPU クラスタの構成

によって通信を行う。MPI そのものは、言語ではなく仕様を定めたものとなっており、C・C++・Fortran-77・Fortran-95 などの言語によって実装されている。

MPI を利用することで、プロセス間の通信に必要なルーチンを手に入れることが可能になり、複雑な通信操作に必要とされる手間を軽減できる。

この MPI を利用した並列計算を行う際の、ノードに対するタスクの割り当て方は machinefile で定義される。machinefile の記入例を Figure 3 で示す。

通常は Figure 3 の (A) のように、使用するノード名を 1 つずつ記述する。これにより、各ノードのプロセッサコアに対して 1 個ずつタスクがラウンドロビン (RR) によって割り当てられる。この方法を、1 コアずつのノード間 RR と呼ぶ。

また、(B) のように同じノード名を連続して記述することで、同じノードのプロセッサコアに対する連続したタスクの割り当てが RR によって行われる。

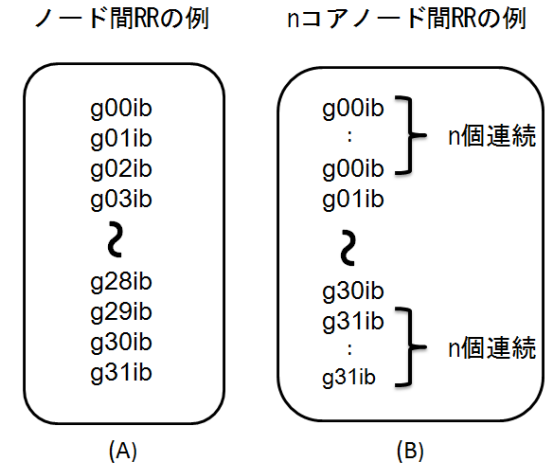

Figure 3. machinefile の記述例

この方法を、n コアずつのノード間 RR と呼ぶ。例えばノード名を 6 回連続で記述すると、各ノードのプロセッサコアに対して、連続して 6 個のタスクが割り当てられる 6 コアずつのノード間 RR となる。

今回使用する CPU には、6 つのプロセッサコアが存在する。そのため、6 コアずつのノード間 RR を行うことで、連続した 6 個のタスクが同じプロセッサ内に割り当てられ、それらのタスクの通信は CPU 内で行われる。これをプロセッサ内通信という。さらに、1 つのノードには 2 つの CPU が搭載されているため、12 コアずつのノード間 RR を行うことで、12 個のタスク間の通信はノード内で行われる。これをノード内通信という。プロセッサ内通信・ノード内通信は、ノード間のネットワーク通信に比べて高速である。

2.2.2. GPGPU(General-Purpose Computation on GPU). 本来、GPU や Video RAM(VRAM) を含むグラフィックアクセラレータは CPU によってコントロールされ、グラフィック処理を補助する目的で使用される [8]。この GPU を CPU が行うような汎用計算に応用するのが GPGPU である。

GPU を GPGPU として利用する場合、CPU は GPU による計算の実行を補助する目的で使用される。CPU は GPU の持つメモリに対してプログラムのダウンロードを行う必要がある。また、GPU はプログラムの読み出しや結果の書き込みを VRAM に対して行うので、CPU は VRAM とメインメモリ間でデータのコピーを行うことで実行を補助する。

Figure 4 に示すような現在利用されている GPU は、Stream Processor(SP) と呼ばれるプロセッサによって構成されている。この SP が汎用計算用のプロセッサとして使用される。しかし、SP によってプログラムが実行されるため、プログラムをストリームベース

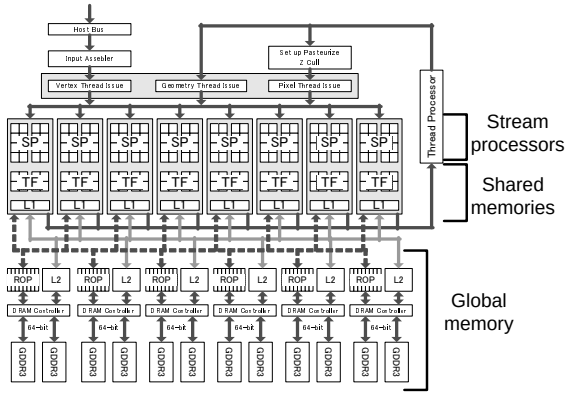

Figure 4. 近年の GPU の構成

に変更しなければならない。このストリームベースのプログラミングインタフェースは、NVIDIA 社によって提案されている、GPGPU ためのランタイムである CUDA(Compute Unified Device Architecture)[9] によってサポートされる。CUDA 用のプログラミングツールや API は、現在同社によって提供されている。

2.2.3. GPU プログラムの並列化. GPU によって実行されるプログラムを並列化する場合、先に述べた MPI による並列化と GPGPU のためのランタイムである CUDA を利用して行う。CUDA によって GPU を汎用計算に利用し、必要となるノード間のネットワーク通信を MPI によって実現する。

この GPU プログラムの並列化を行う際に問題となるのが、ノード間のネットワーク通信速度である。GPU の処理速度に対してノード間の通信速度が圧倒的に遅いため、GPU の処理能力を生かし切ることができない可能性がある。つまり、通信速度に全体の実行時間が支配されてしまうのである。この問題は、CPU プログラムの並列化においても同様である。

2.3. 議論

先述したとおり GPU クラスタによる並列計算において問題となるのが、ノード間のネットワーク通信速度である。複数のノードによって実行される並列計算において、避けることができない点であると共に、計算速度を左右する大きな問題でもある。

この問題を解決する 1 つのアプローチとして、machinefile による並列計算プログラムの分配方法を検討することが挙げられる。例えば、並列計算を行うプログラム中で連続したタスク間の通信が多発する場合、1 コアずつのノード間 RR を行うと、通信を必要とするタスクは異なるノードに割り振られてしまう。そのため、低速なノード間通信が必要となり、ノード間の通信速度に実行時間が左右される。逆に、同じプロセッサ内・ノード内にタスクを割り振れば、

Table 1. 姫野ベンチマークの問題サイズ

S	$128 \times 64 \times 64$
M	$256 \times 128 \times 128$
L	$512 \times 256 \times 256$
XL	$1024 \times 512 \times 512$

高速なプロセッサ内通信・ノード内通信を利用することによる実行時間の短縮が見込まれる。

以上より、本論文では使用するプログラムや CPU・GPU のプロセッサコア数を考慮した n コアずつのノード間 RR を行うことで、高速なプロセス内通信・ノード内通信を促進し、低速なノード間ネットワーク通信を減少させることで、インターネットなどで入手可能なベンチマークを用いて、その性能に与えるインパクトを調査し報告する。

3. ベンチマークソフト

本論文で対象とするベンチマークを以下に示す。

3.1. 姫野ベンチマーク

姫野ベンチマークとは、ポアッソン方程式解法をヤコビの反復法で解く場合の主要ループによって計算速度を測定するのものである [10]。実行時間が短いため測定結果を直ちに求めることができる他、Table 1 に示すように複数の問題サイズが用意されている点や、様々な環境下で測定された結果が公開されているなどの特徴がある。

3.2. NPB

NPB(NAS Parallel Benchmark) とは、NASA(National Aeronautics and Space Administration) によって開発されたベンチマークソフトである [11]。数値流体力学 (CFD : Computational Fluid Dynamics) アプリケーションから派生した、5 つのカーネルプログラムと 3 つの疑似アプリケーションから構成されている。問題のクラス (サイズ) を変更するオプションが用意されており (A・B・C・D・S・W)、使用する環境に合わせた性能の測定が可能である。

問題として、8 つの中から CG・FT・IS・LU を使用する。それぞれの詳しい説明が Table 2 である [12]。

3.3. NAMD

NAMD[13] とは、大規模な生体分子システムのシミュレーションを行うアプリケーションである。イリノイ大学によって開発されており、CUDA を利用した GPU アクセラレーションが可能になっている。本来はベンチマークとして開発された物では無いが、CPU と GPU の計算能力の測定・比較を行うために使用する。

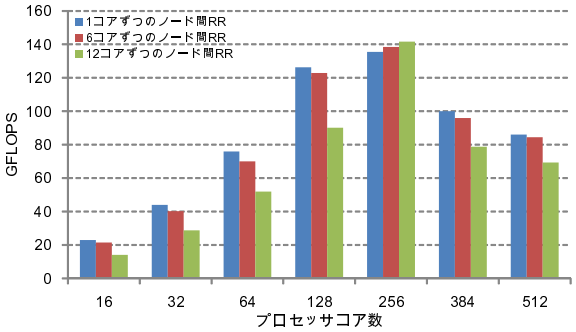

Figure 5. 姫野ベンチマーク サイズ XL GCC

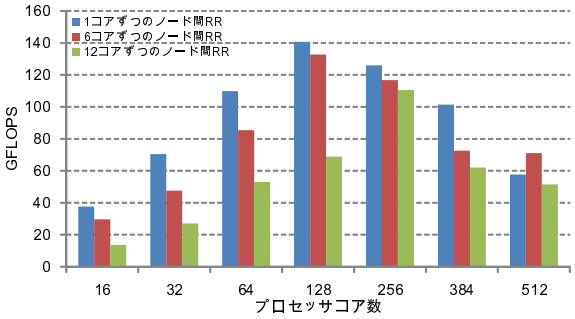

Figure 6. 姫野ベンチマーク サイズ XL Intel

Table 2. 使用する NPB 問題の詳細

CG	大規模な正値対称な疎行列の最少固有値の近似値を、共役勾配法によって計算する
FT	FFT(Fast Fourie Transform) によって、3 次元偏微分方程式の解を計算する
IS	大規模な整数のソートを行う
LU	LU 分解を行うのではなく、5 × 5 の上下三角行列システムを SSOR(Symmetric Successive Over-Relaxation) 法で計算する

Table 3. 3 種類の machinefile

1 コアずつのノード間 RR	ノード 0 から 31 まで 1 個ずつ順番にタスクを割り当てる
6 コアずつのノード間 RR	ノード 0 から 31 まで 6 個ずつ順番にタスクを割り当てる
12 コアずつのノード間 RR	ノード 0 から 31 まで 12 個ずつ順番にタスクを割り当てる

Table 4. 2 種類の machinefile

1 コアずつのノード間 RR	ノード 0 から 31 まで 1 個ずつ順番にタスクを割り当てる
2 コアずつのノード間 RR	ノード 0 から 31 まで 2 個ずつ順番にタスクを割り当てる

4. 性能評価

姫野ベンチマーク・NPB・NAMD を使用した性能評価を行う。

姫野ベンチマークは、[10] からソースコードをダウンロードし、GCC[14] と Intel Compiler[15] によるコンパイルを行う。この際、問題サイズは XL を選択する。そして、姫野ベンチマークを実行するにあたって、ノードへのタスクの割り当て方法として Table 3 に示す 3 種類の machinefile を利用する。

NPB は、8 つのベンチマークの中から CG・FT・IS・LU を使用し、問題のクラスは C とする。ノードへのタスク割り当て方法として、姫野ベンチマークと同様に、Table 3 に示す 3 種類の machinefile を利用する。

NAMD は、実行環境に合わせた実行形式ファイルが用意されており、コンパイル環境に左右されない普遍的な測定が可能になっている。今回は、CPU のみの実行を行うものと、CUDA による GPU での実行をサポートしたものの 2 種類を使用する。さらに、実行する問題サイズによる計算能力の差を測定するため、[13] で公開されているサンプルの中から 2 種類 (ATPase benchmark, STMV (virus) benchmark) を使用する。

NAMD を GPU で実行するにあたって、CPU 同様にタスクの割り当て方が計算能力に与える影響を調べるために Table 4 に示す 2 種類の machinefile を使用する。

4.1. 姫野ベンチマーク

Figure 5 と Figure 6 は、姫野ベンチマークの問題サイズ XL を、GCC と Intel コンパイラによってコンパイルしたものを実行し、FLOPS 値を求めたものである。

4.1.1. コンパイラによる比較. Figure 5 と Figure 6 に関して、使用するコンパイラの違いという点で比較すると、ピーク性能を発揮するプロセッサコア数に違いがある。GCC の場合 256 個のプロセッサコアで実行した際にピークとなっているのに対して、Intel コンパイラの場合 128 個のプロセッサコアで実行した際にピークとなっている。さらに、両者がピークとして示す値には、差がほとんど存在しない。つまり、Intel コンパイラの方が少ないプロセッサコア数で高い性能を引き出しており、プロセッサコア数がある一定を超えてしまうとコンパイラの最適化によって高速化されたタスクの処理時間以上に、ノード間のネットワーク通信時間が長くなり、それがボトルネックとなってプロセッサコア数が増えても性能が下がる。

4.1.2. machinefile による比較. Figure 5 と Figure 6 を Table 4 に示した machinefile の違いという点で比

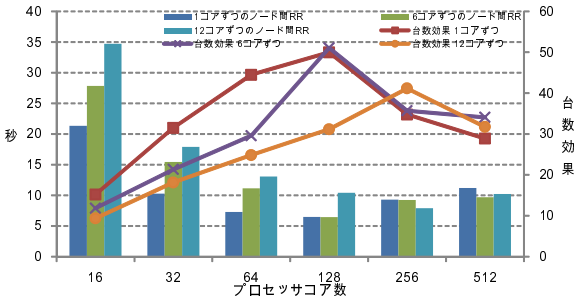

Figure 7. NPB CG クラス C

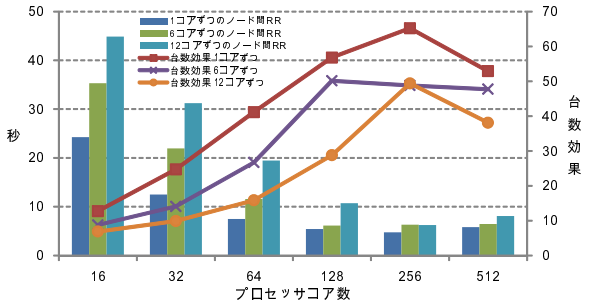

Figure 8. NPB FT クラス C

較すると、1 コアずつのノード間 RR によるタスクの割り当てを行った際の結果が優れている場合が多い。これは、姫野ベンチマークがメモリバンド幅の性能に左右されやすいベンチマークであることに起因する。12 コアずつのノード間 RR のように 1 つのノードに集中的にタスクを割り振ると、プロセッサが持つプロセスあたりのメモリバンド幅が低下してしまう。そのため、各プロセスのメモリアクセス時間の合計がノード間のネットワーク通信時間よりも長くなり、複数のタスクが同じメモリを参照することによるメモリバンド幅の圧迫によって全体性能が低下する。

4.2. NPB

NPB に含まれるものの中から、クラス C の CG・FT・IS・LU ベンチマークを選択して実行する。実行結果として、実行時間と台数効果を求めている。

4.2.1. CG. Figure 7 は CG の実行結果である。CG ベンチマークに関しては、一部を除き 1 コアずつのノード間 RR による実行の際に結果が優れている。これは、選択的なプロセス間のリデュース操作が行われているためだと考えられる。同じノード内に連続してタスクを配置してしまうと、リデュース操作の送信者と受信者が、異なるノードに配置されている可能性が高くなる。そのため、外部のノードへの通信回数が増加してしまい、ネットワークでの通信時間が増加するため全体性能が低下する。

4.2.2. FT. Figure 8 は FT ベンチマークの実行結果である。FT ベンチマークに関しては、全体を通して 1 コアずつのノード間 RR による実行の際に結果が優れている。特に、使用するプロセッサコア数が 16 個のときは、最長で約 20 秒ほどの開きがある。これは、FT ベンチマークで使用されている FFT おいて、バタフライ通信を行っているためだと考えられる。バタフライ通信では、一定間隔ごとのプロセス同士が交互に通信を繰り返している。そのため、同じノード内に連続してプロセスを割り当ててしまうと、ネットワーク通信に時間を必要とする。よって、1 コアずつのノード間 RR を使用した割り当ての方が、適度にタスク間の間隔が取られ、良い結果となっている。

4.2.3. IS. Figure 9 は IS の実行結果である。IS ベンチマークに関しては、使用するプロセッサコア数に対して問題サイズが小さいため、タスクの割り当てによる実行時間の差は誤差と呼べる程度のものである。IS ベンチマーク中では、それぞれのプロセスの計算結果を全体で送受信する全対全の通信を行っている。そのため、タスクの割り当て方に関係なく通信が発生してしまうため、実行結果に差が生まれにくい。

4.2.4. LU. Figure 10 は LU の実行結果である。LU ベンチマークに関しては CG ベンチマークと同様に、1 コアずつのノード間 RR での実行結果が優れており、16 個のプロセッサコアで実行した場合には約 20 秒ほどの差が確認される。ただし、こちらは全対全通信を行っている分だけ、CG ベンチマークに比べると実行方法の違いによる差が少ない。

4.3. NAMD

4.3.1. CPU と CPU+GPU の比較. Figure 11 と Figure 12 は、NAMD において ATPase benchmark(327,506 atoms) と NAMD STMV (virus) benchmark(1,066,628 atoms) を実行した結果である。それぞれの問題において CPU のみによる実行時間と、CPU+GPU による実行時間を比較している。

ATPase benchmark においては、CPU による実行と CPU+GPU による実行に大きな差は存在しない。これは、問題サイズが大きくないため、両者ともに問題に対する十分な計算能力を有しているためだと考えられる。これに対して、NAMD STMV (virus) benchmark では両者の差は明確である。特に、計算に使用するプロセッサコア数が少ない場合は、最高で約 5.5 倍ほど CPU+GPU による実行の方が計算速

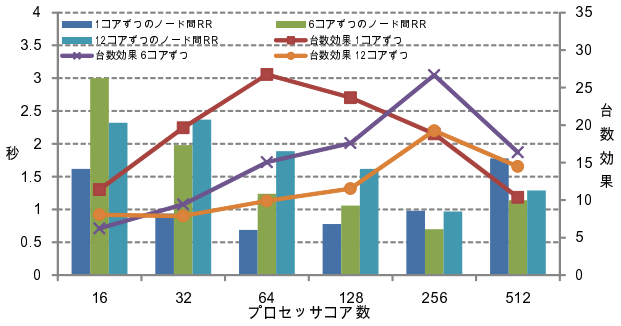

Figure 9. NPB IS クラス C

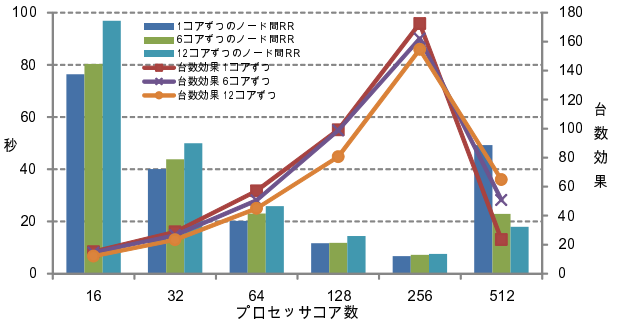

Figure 10. NPB LU クラス C

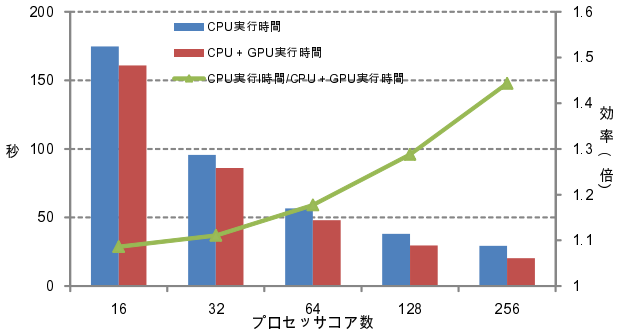

Figure 11. NAMD ATPase benchmark

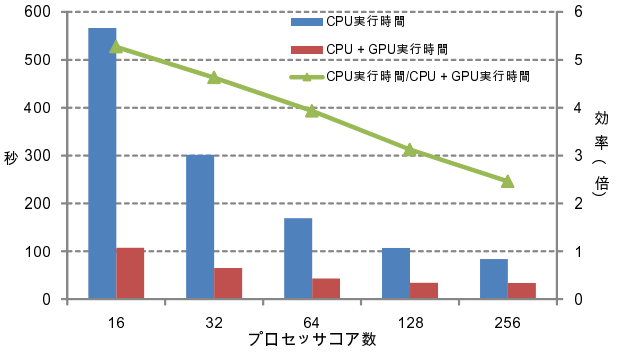

Figure 12. NAMD STMV (virus) benchmark

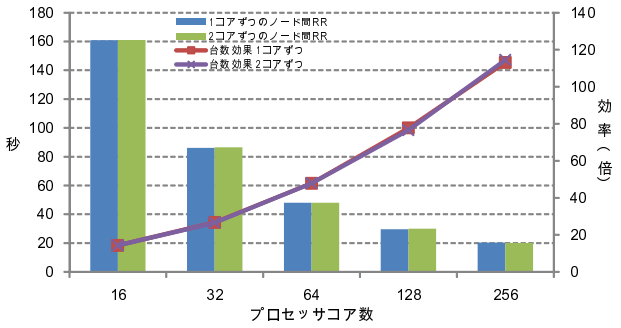

Figure 13. NAMD ATPase benchmark GPU 比較

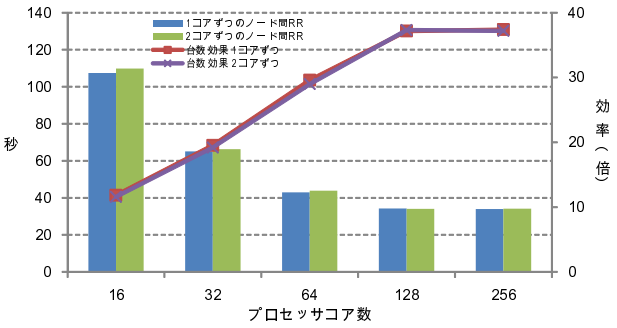

Figure 14. NAMD STMV (virus) benchmark GPU 比較

度が速いことがわかる。使用するプロセッサコア数が増加するに伴って両者の差は減少していくが、これは問題サイズが小さい場合の理由と同様で、問題サイズが CPU の計算能力に対して小さすぎてしまい、ネットワークの通信性能が全体性能に対して顕著になるからである。

以上のことから、問題サイズが十分に大きいまたは、使用するプロセッサコア数が少ない場合は CPU+GPU による計算の実行に優位性があることがわかる。これは、GPU による計算の高速性を示すものである。

4.3.2. GPU のタスク割り当てによる比較. Figure 13 と Figure 14 は、Table 4 に示した 2 種類の machinefile を使用して、CPU+GPU によって ATPase benchmark と NAMD STMV (virus) benchmark を実行し、実行時間を比較している。ATPase benchmark･NAMD STMV (virus) benchmark 共に、実行時間に差は見られないという結果になった。これは、これらのベンチマークを実行する際に、ノード間の通信を必要としていないことを示している。よって、ノード間のネットワーク通信速度が計算能力に与える影響が少なく、並列計算に適していることがわかる。

5. まとめ

本論文では、GPU クラスタにおいてインターネットで入手可能なベンチマークソフトによる並列計算を行い、特にノード間のネットワーク通信に注目し、複数の計算能力の測定を行った。

並列計算を行う場合は、ノードに対するタスクの割り当て方が、計算速度に大きな影響を与えることを示すことができた。特に FT ベンチマークのようなバタフライ通信を使用している場合、連続的なタスクの割り当てが悪影響となり、計算速度が低下してしまう。つまり、必ずしも同じノードに連続的にタスクを割り当てることが効果的とはいえず、使用するプログラムのアクセスパターンを考慮した、タスクの割り当て方を行うことが必要であることを示した。

謝辞

本実験を行うにあたり、高知工科大学 IACP の GPU クラスタを使用させていただいた。

References

[1] Rajkumar Buyya, “High Performance Cluster Computing: Architectures and Systems,” *Prentice Hall*, 1999.

[2] John D. Owens and David Luebke and Naga Govindaraju and Mark Harris, Jens Kruger and Aaron E. Lefohn and Timothy J. Purcell, “A Survey of General-Purpose Computation on Graphics Hardware,” *In Eurographics 2005, State of the Art Reports, pages 21-51*, August 2005.

[3] Vijay Karamcheti and Andrew A. Chien, “Software Overhead in Messaging Layers: Where Does the Time Go?” *In ASPLOS-VI: Proceedings of the sixth international conference on Architectural conference on Architectural support for programming languages and operating systems, pages 51-60*, 1994.

[4] Shinichi Yamagiwa and Kevin Ferreira and Keiichi Aoki and Masaaki Ono and Koichi Wada and Leonel Sousa, “Maestro2: Experimental Evaluation of Communication Performance Improvement Techniques in the Link Layer,” *Journal of Interconnection Networks (JOIN), World Scientific Publishing, vol.7 no.2, pp. 295-318*, June 2006.

[5] David B. Kirk and Wen-mei W. Hwu, “Programming Massively Parallel Processors,” *MORGAN KAUFMANN PUBLISHERS*, 2010.

[6] N. Doss William Gropp, E. Lusk and A. Skjellum, “A High-Performance, Portable Implementation of the MPI Message Passing Interface Standard,” *In MPI Developers Conference*, 1995.

[7] Message Passing Interface Forum, “MPI : A Message-Passing Interface Standard,” September 4, 2009.

[8] Pablo Lamilla Alvarez and Shinchi Yamagiwa and Masahiro Arai and Koichi Wada, “A Uniform Platform to Support Multigenerational GPUs for High Performance Stream-based Computing,” *International Journal of Networking and Computing Vol 1, No 2*, 2011.

[9] “NVIDIA Developer Zone,” http://developer.nvidia.com/category/zone/cuda-zone.

[10] “姫野ベンチマーク Homepage,” http://accc.riken.jp/HPC/HimenoBMT/.

[11] “NAS Parallel Benchmarks Changes,” http://www.nas.nasa.gov/Resources/Software/npb.html.

[12] D. H. Bailey and E. Barszcz and J. T. Barton and D. S. Browning and R. L. Carter and L. Dagum and R. A. Fatoohi and P. O. Frederickson and T. A. Lasinski and R. S. Schreiber and H. D. Simon and V. Venkatakrishnan and S. K. Weeratunga, “THE NAS PARALLEL BENCHMARKS,” *Intl. Journal of Supercomputer Applications, vol. 5, no. 3, pp.66-73*, Fall.1991.

[13] “NAMD - Scalable Molecular Dynamics,” http://www.ks.uiuc.edu/Research/namd/.

[14] “GCC, the GNU Compiler Collection,” http://gcc.gnu.org/.

[15] “Intel Compilers from Intel,” http://software.intel.com/en-us/articles/intel-compilers/.