複数 GPU を用いた DS-CUDA による P2P 機能使用時の性能評価及び最適化

電気通信大学 大学院 情報・通信工学専攻 成見研究室 1531009 伊藤 一輝

1.はじめに

近年、GPU の演算性能は飛躍的な向上を見せており並列演算処理性能が高いことから、GPU を従来のグラフィック処理以外に数値流体計算や Deep Learning などの分野に応用する GPGPU が広く普及してきた。しかし、大規模なシミュレーションや数値流体計算問題をアプリケーションプログラムとして実行するには単体の GPU ではメモリなどの計算資源が不足する。通常は並列コンピューティングで用いられる MPI[1]と GPGPU[2]で用いられる CUDA[3]を利用することで GPU 間あるいはノード間での通信を行い不足する計算資源を補う。また当研究室で扱っている DS-CUDA[4]はネットワークに接続されたサーバ上の GPU を仮想化するミドルウェアであり、クライアント側でソフトを書き換えることなくリモートの GPU の計算資源を用いた GPGPU によって同様の問題を解消することが可能である。しかし、大規模なデータに対し高並列化のプログラムを実装するとレイテンシが大きくなり通信速度が向上しないという問題が新たに発生する。対策としては DS-CUDA API の dscudaMemcopies()を利用することで、サーバ上の GPU 間の通信を Peer to Peer(P2P)で並列に処理することで高速化が可能になっている。

本研究では、まず単一 GPU 向けの数値流体計算用のコードを複数 GPU に対応する。更に DS-CUDA に搭載されている P2P 機能を用いてノード間の通信をサーバ側だけで行うシステムを構築し、複数の GPU を用いた計算性能を評価し並列化効率を示す。

2.複数 GPU 間通信に関連する技術

2.1 GPUDirect[5]

GPU Direct は NVIDIA 社が 2010 年 6 月に導入した異なる GPU 間のデータ転送を高速化させる機能である。CPU 上の不要なオーバーヘッドを取り除くことにより、PCI-Express における GPU 間の高速通信を可能とする。GPU Direct ver.1 では InfiniBand によって CUDA ドライバによるノード間転送、ver.2 では P2P 通信によってノード内のホストメモリを経由しない GPU 間転送, ver.3 では InfiniBand 間での DMA 転送(RDMA)を実装している。しかし、上記の機能を使用し処理性能を向上させるにはデータ転送時に MPI(CUDA-Aware MPI)の導入が不可欠となる.

本研究では MPI を使用せずに DS-CUDA API を使用することで複数のデータ転送を並列に行うという点で異なる.

2.2 A High-productivity Framework for Multi-GPU computation of Mesh-based applications[6]

青木らは複数 GPU による格子に基づいたシミュレーションを簡便に高い生産性で開発することを可能にするマルチ GPU コンピューティング・フレームワークを提案した。フレームワークはユーザコードをノード間は MPI、ノード内の複数 GPU は OpenMP で並列化している。GPU 間通信は CUDA API の cudaMemcpy()、もしくは GPU Direct による P2P によって配列のポインタを指定しデータ通信を行っている。

評価実験においては圧縮性流体計算のシミュレーションを 2 基の NVIDIA Tesla K20X GPU を使用してフレームワークを適用した結果、約 1.4 倍の高速化に成功している。

本研究とは複数 GPU 計算でもユーザコードを簡単に記述できるという点は相似しているが、本研究では CUDA に 1 つの API を追加するだけで簡単に記述できるという点で異なる。

3. システム構成

3.1 システムの概要

DS-CUDA[4]システムは、ユーザがアプリケーションを実行するクライアントと GPU を搭載した複数のサーバで構成される。お互いは InfiniBand ネットワークで接続されている。ユーザの CUDA プログラムは nvcc ではなく DS-CUDA コンパイラ(dscudacpp)でコンパイルする。サーバ側では DS-CUDA サーバプログラムを常時稼働させておく。クライアントで呼び出された CUDA API の引数はサーバ側に転送され、本当の API はサーバ側の GPU で実行される。

GPU1～3 のデータを GPU0 に転送したい場合を考える。P2P 機能を使用しない場合（図 1 左）は、一旦クライアントにデータを転送してから GPU0 に送るので、クライアント-サーバ間の転送量が多くなる。P2P を使用する場合（図 1 右）は、データ転送自体はサーバ間で行われるため、クライアント-サーバ間転送量を最小限に抑えることが出来る。

3.2 dscudaMemcopies() API

DS-CUDA は、異なるノード間であっても cudaMemcpy(..., DeviceToDevice) API を使う事で見かけ上 GPU 間通信を行うことが出来る。しかし大量の GPU 間通信を行う場合は通信レイテンシが問題となるため、複数 GPU 間通信をまとめた以下の dscudaMemcopies()API を提供している。

void dscudaMemcopies(void***dbufs*, void***sbufs*, int**counts*, *int ncopies*)

dbufs: 転送元アドレスのリスト

sbufs: 転送先アドレスのリスト

counts: データ転送量のリスト

ncopies: データ転送個数

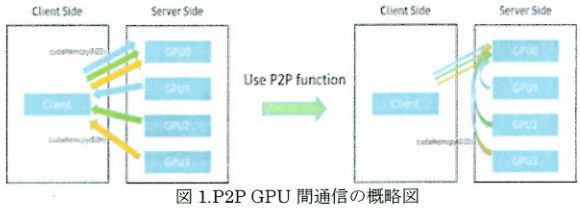

図 1.P2P GPU 間通信の概略図

転送元および転送先がどの GPU かは、アドレス(UVA)から自動的に判定される。同時実行可能なデータ転送は複数のスレッド上で並列に実行されるため、通信レイテンシをある程度隠ぺいすることが出来る。DS-CUDA を用いた場合は通信レイテンシがボトルネックになり易いことが分かっている。

3.3 数値流体計算用コードについて

本研究で扱う数値流体計算用のコードは圧縮性流体計算として 3 次元 Euler 方程式から Rayleigh-Taylor 不安定性の成長シミュレーションを次の方程式で解く。

$$\frac{\partial U}{\partial t}+\frac{\partial E}{\partial x}+\frac{\partial F}{\partial y}+\frac{\partial G}{\partial z}=S \quad (1)$$

$$U=\begin{bmatrix}\rho\\ \rho u\\ \rho v\\ \rho w\\ \rho e\end{bmatrix},\ E=\begin{bmatrix}\rho u\\ \rho uu+p\\ \rho vu\\ \rho wu\\ (\rho e+p)u\end{bmatrix},\ F=\begin{bmatrix}\rho v\\ \rho uv\\ \rho vv+p\\ \rho wv\\ (\rho e+p)v\end{bmatrix},$$

$$G=\begin{bmatrix}\rho w\\ \rho uw\\ \rho vw\\ \rho ww+p\\ (\rho e+p)w\end{bmatrix},\ S=\begin{bmatrix}0\\ 0\\ 0\\ \rho g\\ \rho wg\end{bmatrix}$$

ここで、ρは密度、(u,v,w)は速度、pは圧力、eはエネルギーを表している。gは重力加速度である。移流計算は 3 次精度風上手法で解き、時間積分は低メモリ消費型の 3 段 3 次精度の TVD Runge-Kutta 法を用いる。

拡散計算では扱う変数がfのみであるが、本研究では$\rho,\rho u,\rho v,\rho w,\rho e$ における 5 変数の時間発展を解く。ステンシル計算は 1 方向に 5 点、3 次元計算では 13 点の格子点を参照する。

4 評価

本研究では以下の 3 つの方法を表 1 のノード構成を使用して評価を行った。

・Native:DS-CUDA を使用しない場合(1GPU, 2GPU)の性能評価(GPU 数に対する計算性能のみ)

・InfiniBand:DS-CUDA を利用した InfiniBand ネットワークによる(1GPU~8GPU)性能評価

・P2P:DS-CUDA を利用した P2P 機能による(2GPU~8GPU)性能評価

本研究で使用した数値流体計算用のコードは 1GPU から 4GPU までは16^3から256^3のグリッドサイズで 8GPU は32^3から512^3のグリッドサイズで処理性能、通信時間の評価を行った。1 回あたりの実行におけるステップ数は 1000 回とし、1 回目の実行結果は GPU の処理に急な負荷をかけることから予期せぬ性能評価が出る可能性があるため無視し、2 回目以降から計測を 5 回行った。性能評価より InfiniBand の性能を 1.0 とした時の P2P の処理性能を図 2 に示す。並列数とグリッド数が増加するほど処理性能比は増加しており、特に 8GPU 時のグリッドサイズ256^3において最大 2.75 倍の処理性能を出すことに成功した。

表 1. ノードの構成

OS	Ubuntu 14.04.3 LTS / Fedora 14
CPU	Intel Core i7 920 2.67GHz
CPU Memory	8GB (4GB×2)
GPU	GeForce GTX 780
GPU Memory	3GB (3072MB)
Compiler	dscudapkg2.4.0 and CUDAToolkit 6.0

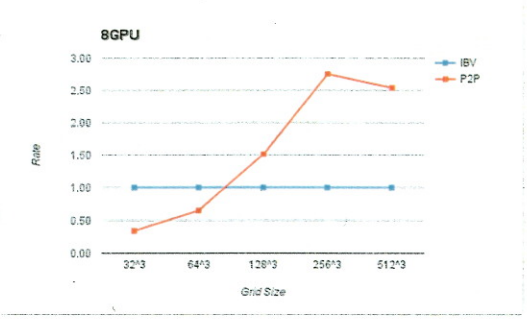

図 2. InfiniBand に対する性能比(8GPU)

表 2.P2P による通信時間削減率

P2P	128^3	256^3	512^3
2GPU	-25.31%	11.79%	-
4GPU	27.35%	14.10%	-
8GPU	42.27%	71.90%	72.51%

最後に P2P 機能使用時に InfiniBand 使用時と比較してどれだけ通信時間を削減できたのか表 2 に示す。負の値は InfiniBand 使用時と比較して通信時間の増加を、正の値は通信時間の削減を表している。P2P 機能はグリッドサイズが大きく高並列であるほど有効であり、8GPU における P2P 機能使用時には512^3グリッドサイズにおいて最大 72.51%の通信時間の削減に成功した。

5 まとめと今後の課題について

本研究では、単一 GPU 向けの数値流体計算用のコードを複数 GPU に対応した。更に DS-CUDA に搭載されている P2P 機能を用いてノード間の通信をサーバ側だけで行うシステムを構築した。その結果、P2P 機能を使うことにより通信時間を削減して複数 GPU 使用時の性能を向上することが出来た。

評価では DS-CUDA を利用した P2P 機能使用時に処理性能の点において 8GPU、256^3グリッドサイズにおいて InfiniBand ネットワーク使用時と比較して約 2.75 倍の高速化を達成できた。また、通信時間においては P2P 機能使用時512^3グリッドサイズにおいて InfiniBand ネットワーク使用時の通信時間より約 72.51%の削減を達成できた。このため、DSCUDA API の P2P 機能である dscudaMemcopies()はサーバ側のノード間 GPU 通信において転送データ量を大きくし高並列化することが通信時間削減に有効であるといえる。

今後の課題として、本研究では数値流体計算を題材として 1 つの問題に対し P2P 機能を利用し通信時間の削減を図ることで高速化したが数値流体計算分野においては解析が困難とされている問題も無数に存在し、それらに対しても高速化が図れる汎用性が求められる。

参考文献

[1]"MPI Solutions for GPUs", https://developer.nvidia.com/mpi-solutions-gpus (最終アクセス日 2017 年 1 月 5 日)

[2]大島 聡史, "これからの並列計算のための GPGPU 連載講座" http://www.cc.u-tokyo.ac.jp/support/press/news/VOL12/No1/201001gpgpu.pdf (最終アクセス日 2017 年 1 月 5 日)

[3]"開発者向けの CUDA 並列コンピューティングプラットフォーム" http://www.nvidia.co.jp/object/cuda-parallel-computing-platform-jp.html (最終アクセス日 2017 年 1 月 5 日)

[4]Atsushi Kawai, Kenji Yasuoka, Kazuyuki Yoshikawa, Tetsu Narumi,Distributed-Shared CUDA: Virtualization of Large-Scale GPU Systems for Programmability and Reliability FUTURE COMPUTING 2012 : The Fourth International Conference on Future Computational Technologies and Applications, Nice, 2012

[5]"NVIDIA GPUDirect™ Technology",http://developer.download.nvidia.com/devzone/devcenter/cuda/docs/GPUDirect_Technology_Overview.pdf (最終アクセス日 2017 年 1 月 5 日)

[6]Takashi Shimokawabe, Takayuki Aoki, Naoyuki Onodera, A High-productivity Framework for Multi-GPU computation of Mesh-based applications , ハイパフォーマンスコンピューティングと計算科学シンポジウム論文集 2013-12-31, 78 – 86, 2014