5A-02

nbody シミュレーションのマルチ GPU 環境における最適化

亀ノ上　剛†

日本アイ・ビー・エム株式会社†

1. はじめに

GPU の高い演算能力を引き出すためには，GPU の持つ多数の浮動小数演算ユニットを飽和させるだけの高速なデータ供給が求められる．

複数の GPU で共有される大量のデータを持つプログラムの場合，プログラミングを容易にするために，そのようなデータを CPU のメモリに置いて，GPU から参照する方法がある．多体問題の nbody シミュレーションのプログラムがその一例である．このプログラムでは，時間ステップ毎に更新される N 個の質点の位置情報を，CPU メモリに置いている．このプログラムでは，N が大きい場合は CPU メモリ-GPU 計算ユニット間で大量のデータ転送が発生する．この間のデータ転送速度が，性能に影響を与えることが予想される．既存研究[1]では，CPU-GPU 間のデータ転送能力と性能の関係を計算で予測している．

本論文では，シングル GPU 実行の場合に，CPU-GPU 間のデータ転送バンド幅が nbody シミュレーションの性能に影響していることを実機を用いて示し，CPU-GPU 接続を PCI Express（以下 PCIe）から高速な NVLink に変更することで 3.3 倍の性能向上が得られることを示した．

さらに，マルチ GPU 実行の場合に，CPU 側でのデータ転送がボトルネックとなり nbody シミュレーションの性能がスケールしないことを実機を用いて示し，メモリ・インターリーブを使うことで 2.0 倍の性能向上が得られることを示した．

2. nbody とその計測環境

nbody は，N 個の質点（宇宙空間の星など）がそれぞれ自分以外の N-1 個の質点から受ける重力から加速度を求め，Δt 秒後の速度と位置座標を計算，これを 1 イテレーションとする．このようなイテレーションを繰り返すことで，N 個の質点の位置座標の時間発展をシミュレートする．

今回は，CUDA Toolkit に含まれるサンプルプログラムを用いた[2][3]．最初に 1 回ダミーのイテレーションを実行した後，本計測用に 10 回のイテレーションを実行し，プログラム自身が定める理論的な浮動小数演算回数を実行経過時間で割って GFLOPS 値を算出，これを結果として表示する．32bit モードと 64bit モードの測定が可能であるが，今回はデフォルトの 32bit モードを使用した．問題サイズは N=2048000 を用いた．

実行環境としては，x86 アーキテクチャ (Intel Xeon CPU E5-2640 v4 @2.40GHz 20core, 256GB memory) に 4 基の NVIDIA 社製 GPU P100 を PCIe で接続したシステム (SUPERMICRO 1028GQ-TXRT)，POWER アーキテクチャ (POWER8 @2.86GHz 20core, 256GB memory) に 4 基の GPU P100 を NVLink で接続したシステム (IBM Minsky)，の 2 種類のシステムを使用した．

3. プログラムの解析と性能改善

3.1. シングル GPU 実行

まず，CPU-GPU 間のデータ転送速度が性能に与える影響を知るために，N 個の質点の位置座標データを，GPU メモリに配置した場合，CPU メモリに配置した場合，の 2 通りの性能を 1028GQ-TXRT システムで測定した．結果は下記のとおりである．

・GPU メモリに配置：6300 GFLOPS

・CPU メモリに配置：1670 GFLOPS

CPU 側のメモリにデータが配置された場合は，図 1 の右の図が示すように，ゼロコピー[4]によって CPU-GPU 間転送を行った後，GPU の計算ユニットに直接転送される．

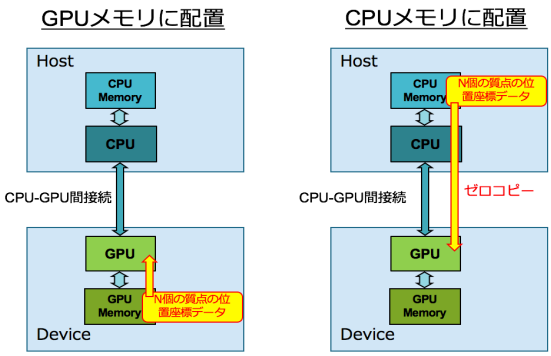

図 1：N 個の質点の位置座標データの配置

CPU メモリにデータが配置された場合に性能が低下している，ということは CPU-GPU 間のデータ転送バンド幅がボトルネックになっていることが考えられる．1028GQ-TXRT システムでは，理論ピークバンド幅が 32GB/s である．

データ転送バンド幅がボトルネックであるか

The optimization of nbody simulation on Multi-GPU environment

†Tsuyoshi Kamenoue, IBM Japan

を確かめるために，CPU-GPU 間のデータ転送の理論ピークバンド幅が 76.8GB/s の NVLink を使用した Minsky システムで性能を計測した．結果は 5519 GFLOPS であった．これは PCIe 接続の結果の 3.3 倍の性能改善である．PCIe の単方向データ転送実測値 10.3 GB/s と NVLink の単方向データ転送実測値 36.8 GB/s という性能差[5]が反映されたものである．

3.2. マルチ GPU 実行

次に Minsky システムを用いてマルチ GPU 実行のベンチマークを実施した．当ベンチマークは，CPU 側では起動するプロセスは 1 個であり，1 コアを使用する．結果は以下のとおりで，4 基の GPU を使っても 1 基の GPU を使用したケースの 1.5 倍弱の性能しか得られていない．

・1GPU：5519 GFLOPS
・4GPU：8005 GFLOPS

この原因を追究するため，4 基の GPU で実行する際に，NVIDIA 社が提供するツールでプロファイルを取得して可視化した．すると各時間ステップ毎に全ての GPU が同期を取るために，GPU0, GPU1 が GPU2, GPU3 での処理が終わるのを待っており，演算処理を行っていない状態が発生していることが確認できた．これはメモリアクセスの経路が図 2 に示すように，全て片方の CPU に接続するメモリに集中しており，CPU0 側のメモリバンド幅と，CPU0 と CPU1 をつなぐ SMP バスのバンド幅がボトルネックになっていることが原因であると考えられる．

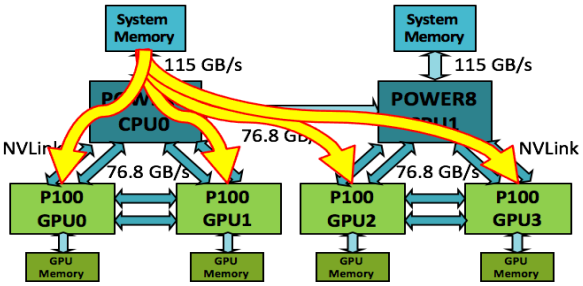

図 2：インターリーブなしの場合のデータ経路

そこで CPU0 のメモリへのアクセスの集中を解消させる目的で，OS の機能であるメモリ・インターリーブ[6]を使用し，CPU0 のメモリと CPU1 のメモリをページ単位で交互にアクセスした．結果は以下のとおりで，4 基の GPU を使用すると，1 基の GPU の 3.0 倍の性能が得られた．

・1GPU：5348 GFLOPS
・4GPU：16019 GFLOPS

4GPU 実行の結果をインターリーブなしの結果と比べると 2.0 倍の性能向上が得られた．このケースでのプロファイラの出力を見ると，4GPU 実行における GPU0, GPU1 の待ち状態は解消されていた．これはメモリアクセスの経路が図 3 に示すように分散されたことで，CPU0 側のメモリバンド幅と，CPU0 と CPU1 をつなぐ SMP バスのバンド幅のボトルネックが緩和されたためである．

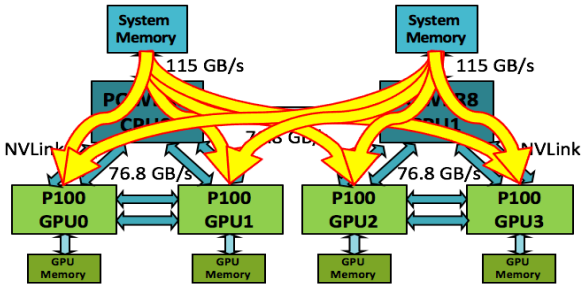

図 3：インターリーブありの場合のデータ経路

4. おわりに

本論文の貢献は下記の 2 つである．

(1) nbody シミュレーションのように大きなデータを CPU 側に配置しながら GPU 演算を行うプログラムにおいて高い性能を得るために，CPU-GPU 間のデータ転送バンド幅の高さが重要であることを実測によって示した．

(2) マルチ GPU でのスケーラビリティを確保するために，CPU 側の各種バス幅のボトルネックを緩和する手段として，メモリ・インターリーブが効果的であることを実測によって示した．

今後の課題として，プログラム内部でのブロックサイズを変化したときの性能変化，GPU キャッシュの利用状況の詳細，の検証がある．

参考文献

[1] Sergio M. Martin, N-Body Simulation Using GP-GPU: Evaluating Host/Device Memory Transference Overhead, XIX Argentine Congress on Computer Sciences, 2013

[2] Lars Nyland, Mark Harris, Jan Prins, Fast N-Body Simulation with CUDA, https://github.com/marwan-abdellah/GPU-Computing-SDK-4.2.9/blob/master/C/src/nbody/doc/nbody_gems3_ch31.pdf

[3] Mark Harris, Multi-GPU programming, Melbourne HPC GPU Computing Workshop and Introduction to MASSIVE, 2011

[4] NVIDIA Advanced CUDA Webinar, Memory Optimizations, http://developer.download.nvidia.com/CUDA/training/NVIDIA_GPU_Computing_Webinars_CUDA_Memory_Optimization.pdf

[5] 成瀬彰，PASCAL:最新 GPU アーキテクチャ，https://www.gputechconf.jp/assets/files/1011.pdf

[6] 吉廣保，実アプリケーションの最適化のテクニック，https://www.cc.u-tokyo.ac.jp/support/press/news/VOL10/No6/200811tuning.pdf