2013 年 4 月 2 日 FS 専攻修士論文中間発表会発表資料

CPU と GPU 間のデータ転送時間の短縮に関する研究

高性能コンピューティング学講座　本多・近藤研究室
1253011 沈　峻
主任指導教員：本多弘樹

1　はじめに

GPU(Graphics Processing Unit) は、高速の VRAM（ビデオメモリ；グラフィックボード上の GPU 専用メモリ）を有し、グラフィクスシェーディングに特化したプロセッサが多く集まった構造を持っていて、一つ一つのプロセッサの構造は単純なためその機能は CPU に比べて限定されたものだが、大量のデータを複数のプロセッサで同時かつ並列処理することができる。近年、NVIDIA 社の CUDA に代表される GPU 向けの開発環境が整備され、手軽に CUDA プログラミングが行えるようになった。しかし、GPU で演算するためには、CPU と GPU の間でのデータ転送が必要で、計算時間よりもデータの転送時間が遙かに大きいという事態になる。

CUDA 2.2 で追加された Zero Copy ではデータを CPU 上の Mapped Memory に保存して、メモリのアドレス空間を CPU と GPU 間でマッピングすることにより、GPU が CPU のメモリ上のデータを直接アクセスすることができる。

本研究では、CPU と GPU のデータ転送時間の短縮を着目し、Zero Copy 機能を応用し、NVIDIA の CUDA API を拡張する。API の機能として、データのアクセス回数に応じて、自動的に転送方式を選択する。アクセス回数の少ないデータについては Zero Copy により転送し、回数の多いデータだけ GPU 側のメモリに Memcpy してアクセスすることにより、CPU と GPU のデータ転送時間を短縮することを目的としている。

2　関連研究

GPU コンピューティングにかかる時間は CPU と GPU 間でのデータ転送時間とカーネル実行時間二つの部分である。

カーネル実行時間の短縮を着目して、Shuai Che らは Dymaxion[1] を提案した。二次元配列を転置して、まとめてアクセスできるようになる。これにより実行時間を短縮することができた。研究している中で、Zero Copy 機能を使って、Zero Copy を使う有用性を示している。しかし、場合によって、Zero Copy を使うと、転送時間が増加してしまう場合もある。

1 つのプログラム内に、カーネル関数が二つ以上ある場合、複数回のデータ転送が必要となる。そこで、Michela Becchi, らは、Data ー Aware スケジューリング手法 [2] を提案した。データ転送する必要な時間を見積もり、カーネル実行時間を含めて比較し、全体時間の短い方法で実行する。しかし、こうすると、実行時間の遅い側で選択せざるを得ない。データ転送時間を短縮できれば、高速側で実行できるようになれる。

3　研究へのアプローチ

通常の GPU コンピューティングでは次のステップが必要となる。

(1)CPU 側のメモリ内にデータを作成する。
(2)CPU から GPU 側のメモリにデータを転送する。
(3) カーネル関数を呼び出して、計算を行う。
(4)GPU から CPU 側のメモリにデータを転送する。

本研究では、(2) と (4) の段階の時間を減らすのがことを目的としている。

3.1　Memcpy と Zero Copy

Memcpy と Zero Copy を使用する場合のデータ転送を図 1 に示す。

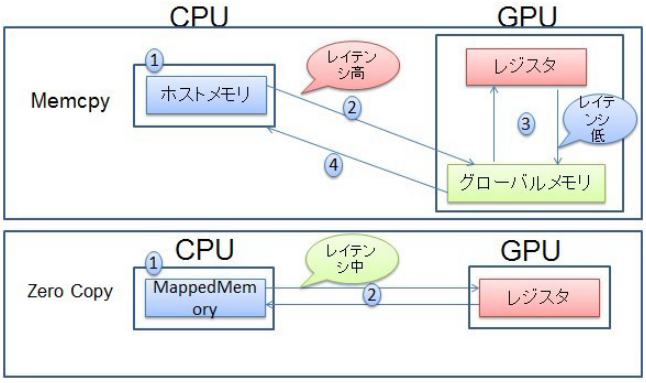

図 1: Memcpy と Zero Copy を使用する場合のデータ転送

Memcpy を使う際に、CPU から GPU のグローバルメモリにデータを転送して、カーネルを起動したら、レジスタがグローバルメモリ側のデータにアクセスする。Zero Copy を使う際に、レジスタが直接 CPU 側の MappedMemory にアクセスできる。1 回アクセスする場合、Zero Copy の方が速いが、2 回以上アクセスする場合、GPU 側のメモリ内にデータが保存していないのため、再度 CPU にアクセスしなければならない。CPU と GPU のデータ転送が

GPU 内のメモリ間の転送より遥かに大きいのため、Zero Copy の方が遅くなってしまう。

時間を減らすために、Zero Copy 機能に注目している。Zero Copy を使う際に、通常のメモリ領域が使わずに、Mapped Memory を使用している。

Mapped Memory 使用する際の手順は以下のようになる。

(1) デバイスに Mapped Memory を使用するためのフラグをセットする。

(2)Host Memory を Mapped Memory 専用に確保する。

(3)2 で確保した Host Memory を Device Memory にマップし、デバイスから Mapped Memory へアクセスするためのポインタを取得する。

(4)3 で取得したポインタを使用して Kernel 内部から Mapped Memory へアクセスし、適宜データをロード、保存する 。

(5) 計算が終了したら、Mapped Memory を開放する。

通常の Device Memory と Mapped Memory を使用する際に異なる部分は以下の通りである。

(1) フラグを設定しているか否か

(2)cudaMemcpy による転送を明示的に行っているか否か

(3)Device Memory の領域の開放を行っているか否か

ポイントは、Kernel 側の変更は一切行わずに使用できるということである。 Kernel 内部でのアクセス先が通常の Global Memory である場合は、Device Memory へのアクセスとして処理されるが、 Mapped Memory である場合には自動的に PCI を介してデータ転送が実行される。つまり、Kernel 内部の計算と、PCI バスを介したデータ転送を非同期に実行することができるようになる。 これが、Mapped Memory が効率よくデータ転送を行うことができる理由である。

3.2 提案手法

CPU から GPU へのデータ転送する際に、CUDA の Zeco Copy 機能を使って、GPU が直接 CPU 内のデータをアクセスすることによって、Memcpy を減少することができるが、同じデータを頻繁的にアクセスすることには、逆にデータを GPU に Memcpy するより時間がかかってしまう場合もある。そこで、データのアクセス回数に応じて、読む回数の少ないデータが CPU 側の Mapped Memory に保存し、回数の多いデータが GPU 側に Memcpy する。

続いて、データが GPU から CPU への転送する時の量を減らす手法について述べる。書き込み回数の少ないデータが GPU 側に記憶せず、直接 CPU 側の MappedMemory に保存する。また、こちらのデータ転送はカーネル実行時間とオーバーラップできるのため、全体の実行時間の短縮も目的としている。

ここで、cudaMemcpy の代りに、新しいデータ転送用の API を提案する。この API がカーネル実行前に、データアクセス回数を検出して、検出した結果により、アクセス回数に応じて、各データの転送方式を自動的に選択する。コードの変更を最小限にするために、カーネル側が変更なして実現するのが目的としている。

4 進捗状況

実験環境については、GPU が TeslaS2050 を使用し、OS は Ubuntu11.10 である。提案手法に対して、実用性を見るために、以下の実験を行った。カーネルは 512MB の Float 型ベクトル c[n]=a[n] + b[n] の計算であり、本来計算する時データが普通の領域内に保存しているが、メモリの性質によって転送速度が違うのため、より正確な結果を得るために、GPU に転送するデータと転送しないデータとも Mapped Memory に保存している。Memcpy と Zero Copy 両方でベクトルの足し算 1 回と 2 回で実験した。実験の結果を表 1 に示す。示しているのは、データ転送時間を含めて、全体の実行時間である。

表 1: Memcpy と Zero Copy の比較

アクセス回数/転送方式	Memcpy	Zero Copy
1 回	339ms	252ms
2 回	345ms	505ms

実験結果により、アクセス回数が 1 回の時、レジスタが直接 CPU 側のメモリにアクセスしての Zero Copy 方法の方が速いだが、2 回アクセスするの場合、毎回 CPU 側にデータをアクセスしなければならないので、逆にデータを全部 GPU 側に転送して、レイテンシの短いグローバルメモリにアクセスした方が効率がいいである。

5 今後の方針

今後は、データアクセス回数の検出手法を検討して、出力できるようになる。その後、検出したデータアクセス回数に応じて、自動的に転送方式を選択できる API を作成する。最後に、作成した API を実装し、ベンチマークを実行して、実装する前との実行時間の比較によって、本研究の有用性を評価する。

参考文献

[1] Shuai Che, Jeremy Sheaffer, Kevin Skadron, “ Dymaxion: Optimizing Memory Access Patterns for Heterogeneous Systems ”,in Proceedings of the 2011 ACM/IEEE International Conferencefor High Performance Computing, Networking, Storageand Analysis, SC ’ 11.2011.

[2] Michela Becchi, Surendra Byna, Srihari Cadambi, Srimat Chakradhar.Data-aware scheduling of legacy kernels on heterogeneous platforms with distributed memory.In SPAA ’10,June 2010.