2012 年 11 月 19 日 情報システム基盤学合同輪講発表資料

Dymaxion: Optimizing Memory Access Patterns for Heterogeneous Systems

著者： Shuai Che, Jeremy W. Sheaffer and Kevin Skadron
出典： *SC '11 2011 International Conference for HPC, Networking,Storage and Analysis Article No. 13.*
発表者： 本多・近藤研究室 1253011 沈　峻

1　概要

本論文での Heterogeneous System とは CPU ＋ GPU で計算するシステムである。

GPU ではメモリアクセスバターンによって、実行時間が大きく変わるので、配列変数の Remapping が必要となる場合がある。

本研究では、Remapping を CPU-GPU データ転送とオーバーラップさせることにより、Remapping にかかる時間を隠蔽する手法とそのための API を提案する。

2　Remapping

CUDA（Compute Unified Device　Architecture) とは「NVIDIA GPU 製品において汎用的なプログムを動すためのプラットフォーム・統合開発環境」である。

Cuda プログラムを実行する際に、32 スレッドの Warp ごとが単位で実行され、メモリアクセスは半ワープ単位 (16 スレッド) で行われ、連続したメモリにアクセスしていると、効率良くアクセスできる。連続ではない場合に連続領域になるように変数の配置を変える Remapping により効率いいアクセスが可能となる。

本論文では、3 つのメモリ配置 (図 1〜図 3) に対しての Remapping を対象としている。Remapping する時に、1 スレッドあたりのデータ処理量は [2] の結果で決める。Remapping は図 1 から図 3 に示す。

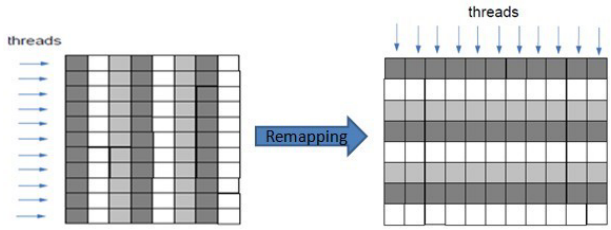

図 1: Row-major to column-major transformation.

図 1 は Row-major から column-major の変換、図 2 は対角線のアクセスの Remapping、図 3 は配列変数の間接参照の Remapping である。

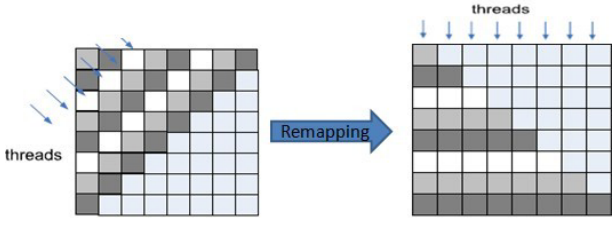

図 2: Diagonal strip matrix transposition

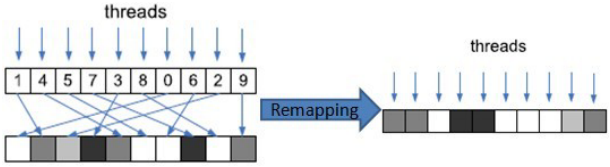

図 3: Indirect remapping for gather.

3　提案手法

3.1　Remapping とデータ転送のオーバーラップ

Remapping の単純な実装は、データを CPU のメモリから GPU にデータを一括してそのまま転送して、GPU がメモリ内で Remapping を行う方法である。しかし、この方法ではメモリを Remapping するには時間がかかるので、オーバーヘッドが発生してしまい、計算時間が短くなっても、全体の性能が向上できない (図 4)。

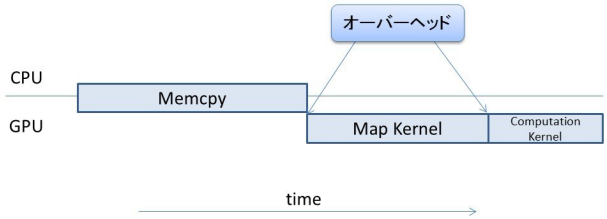

図 4: Memory layout reorganization

そこで、本論文では、CPU-GPU 間のデータ転送と Remapping をオーバーラップさせる。CPU-GPU 間でのデータではデータを分割転送して、GPU は受け取った一部分のデータに対して Remapping カーネルを起動して、メモリ配置を変える。データ転送と Remappping をオーバーラップするイメージは図**??**に示す。

本論文の実験により、分割数が 16 以上であれば、より多いのメモリコピーと Remap カーネルの起動はオーバーヘッドになって、スループットが大幅に低下するので、分割は 16 にしている。

また、データ転送方法は同期の cudaMemcpy() と streaming ＋非同期の cudaMemcpyAsync() があるが、実験により、非同期の方法のほうが平均 16.7 %速くなるため、本論文は二つ目のデータ転送を行っている。

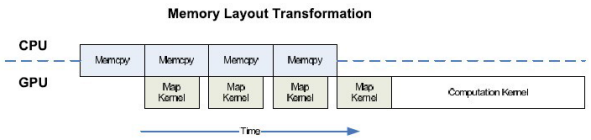

図 5: Memory layout reorganization

3.2 API

本論文は、Cuda の API を提供しているので、プログラマは、メモリを Remapping したい際に、自分で Remap Kernel のコードを書く必要がない。プログラマはメモリコピーを cudaMemcpyAsync() に変更した後、メモリコピーのコードの下に Remapping カーネルの API を導入することによって、メモリを Remapping することができる。イメージとしては、図 1 の左側のメモリ配置を右側に変更するための API である。

また、メモリ配置が変わっているので、計算カーネルを実行する際に、スレッドのアクセス方向を変換しないと誤データにアクセスしてしまうので (メモリ Remapping する前の第一列が Remapping した後の第一行になっている)、正しいデータにアクセスために Index も変換しなければならない。プログラマが自分で変換できるが、本論文では、Index 変換の API も用意している。計算カーネルにその API をそのまま導入と、Index の変換ができる。

4 実験評価

4.1 実験環境

GPU は NVIDIA GeForce GTX480（480 コア,1.4GHz）で、CPU は Intel Core2 Quad CPU(2.66GHz) のシステムを使用した。ソフトウェア環境は CUDA3.1 と GCC4.2.4 である。

4.2 実験結果

Row-Major Order to Column-Major Order(K-means)

図 1 の場合の実験結果は図**??**に示す。

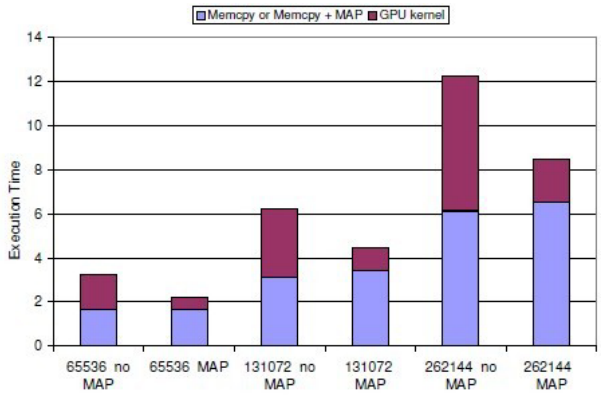

図 6: Row to col の実行時間

紫色部分はデータ転送時間で、赤い部分はカーネルの実行時間である。横軸は二つずつ一組として、同じデータ量 (65536B,131072B,262144B) の Remapping していない (左側 No Map) の実行時間と Remapping している (右側 Map) の実行時間で、縦軸は全体の実行時間である。

最後の一個分割を Remapping する必要なので、転送時間が 5.8 %のオーバヘッドが発生してしまったが、カーネルが 3.11 倍速くなって、全体 30.6 %の性能向上が実現した。

Diagonal-Strip(Needleman-Wunsch)

図 2 の場合の実験結果は図**??**に示す。

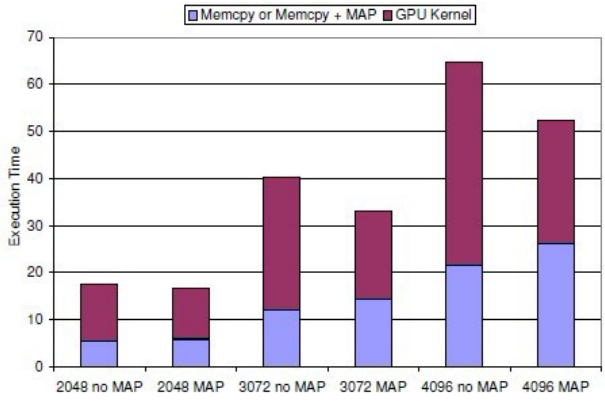

図 7: Diagonal の実行時間

転送時間が 16.1 %のオーバヘッドが発生してしまったが、カーネルが 42.4 %のスピードアップがあって、全体 14.0 %の性能向上が実現した。

Indirect(Sparse Matrix-Vector Multiplication)

図 3 の場合の実験結果は図**??**に示す。

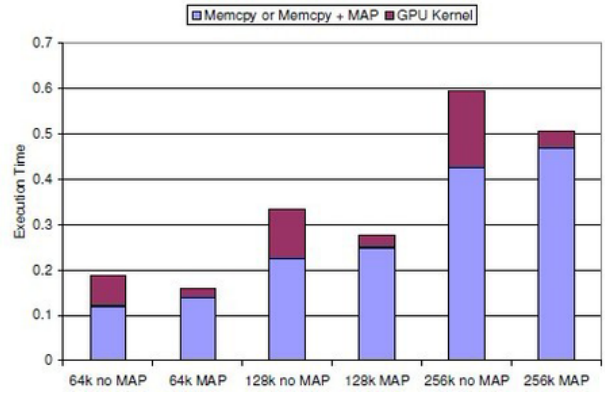

図 8: Indirect の実行時間

転送時間が 10.2 %のオーバヘッドが発生してしまったが、カーネルが 4.1 倍速くなって、全体 15.6 %の性能向上が実現した。

5 まとめ

Dymaxion を提案した。メモリを Remapping して、データ転送とオーバーラップことによって、計算時間が短くなり、データ転送時間のオーバーラップが最小限になり、結果として全体の性能が向上した。

参考文献

[1] NVIDIA CUDA Programming Guide. Web resource. http://developer.nvidia.com/object/gpucomputing.html.

[2] D. Merrill and A. Grimshaw. Parallel scan for stream architectures. Technical Report CS2009-14, Department of Computer Science, University of Virginia, Dec 2009.