2012 年 4 月 3 日 FS 専攻修士論文中間発表会発表資料

GPU 向け OpenMP 処理系におけるチューニングパラメータに関する研究

高性能コンピューティング学講座　本多・近藤研究室
1153004　岩下　光弘
主任指導教員：本多　弘樹

1　はじめに

高性能計算のためのアクセラレータとして GPU が注目されている。GPU はシンプルなアーキテクチャの演算コアを多数搭載し、並列処理を行うことが可能である。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、CUDA プログラミングを行うためには、GPU の複雑な実行モデルやメモリの特性について理解を要するため、プログラマにとって知識習得の負担が大きい。

我々の研究室ではこの負担を軽減する方法として、CPU 向け並列化プログラミング環境 OpenMP を用いて記述したプログラムから CUDA プログラムを生成する処理系 OMPCUDA を提案してきた [1]。一方で OMPCUDA には、計算性能向上のためのチューニングについての課題が残っている。そのため、OMPCUDA で生成した CUDA プログラムは、ハンドコーディングした CUDA プログラムに比べ計算性能を発揮できない場合がある。

そこで本研究では、CUDA プログラムにおけるチューニングパラメータの検証を行い、OMPCUDA へのチューニング機構の実装を提案する。

2　OMPCUDA の課題

最適なスレッドブロック数のチューニング

CUDA ではグリッドとブロックでスレッドを管理している (図 1)。グリッドは GPU に対してスレッドを割り当てる単位である。また、ブロックはプロセッサユニットに対してスレッドを割り当てる単位である。スレッドはスカラプロセッサに対して割り当てる命令実行の単位である。

ブロック当たりのスレッド数とグリッド当たりのブロック数（スレッドブロック数）は、CUDA プログラム毎に異なる。

そのため、生成する CUDA プログラムに合わせて最適なスレッドブロック数のチューニングをどのように行うかが課題となる。

シェアードメモリの利用に関するチューニング

シェアードメモリは GPU コアに実装された高速アクセス可能なメモリである。各シェアードメモリの容量は 16kByte と小容量なため、格納したいデータが収まるよう上手くデータサイズを調整する必要がある。

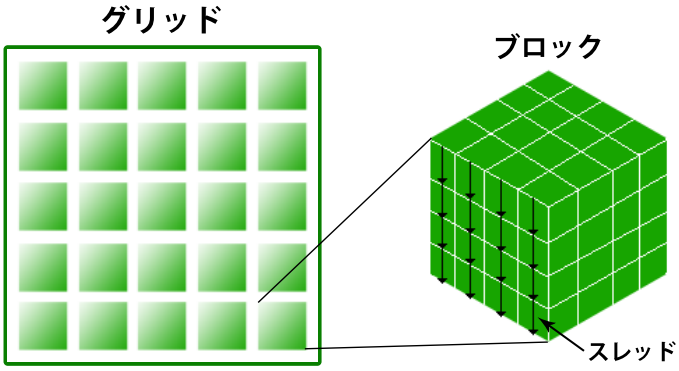

図 1: グリッドとブロックによるスレッド管理

また、シェアードメモリの同一メモリバンクに複数のスレッドが同時アクセスするとバンクコンフリクトが発生する。アクセス競合を起こしたスレッドは、同一メモリバンクに順番にアクセスすることでバンクコンフリクトを解消するため、計算性能低下の原因になる。

そのため、格納したいデータのデータサイズ調整とバンクコンフリクト回避のためのチューニングをどのように行うかが課題となる。

3　関連研究

OMPCUDA と同様のアプローチの研究に、S. Lee・R. Eigenmann らによる CUDA プログラム生成のための OpenMP 処理系の提案があげられる [3]。この研究では、コード変換コンパイラ Cetus Compiler をベースに実装した CUDA プログラム生成のための OpenMP 処理系を提案している。この処理系はコアレスアクセスなどのチューニング機構を実装している。

シェアードメモリの利用とスレッドブロック数のチューニングに関する研究として、額田・松岡らによる GPU 向け自動最適化 FFT ライブラリの提案があげられる [2]。この研究では、メモリアクセスによる遅延の発生が主な計算性能のボトルネックとなる 3 次元 FFT ライブラリに対して、シェアードメモリの利用に関するチューニングを施している。また、バンクコンフリクトを回避する方法としてパディングの有効

性を唱えている。

4 課題解決へのアプローチ

4.1 最適なスレッドブロック数のチューニング

最適なスレッドブロック数を決定するためには、計算に参加する総スレッド数に対してどの程度のブロック数を設定するかが重要である。

ブロック当たりのスレッド数を多くすれば並列性は高まるが、スレッド間の同期によるオーバヘッドが増大する。また、スレッドに対して割り当てるレジスタが不足する恐れもある。CUDA は、レジスタが不足した場合、プログラム中で指定したスレッド数を減らすことで、全てのスレッドにレジスタを割り当てるよう調整する。そのため、計算性能の低下につながる場合がある。

一方、ブロック当たりのスレッド数を少なくすることで並列性は低くなるが、スレッドに対して割り当てるレジスタ数が増加するため、計算性能の向上が期待できる場合がある。

そのため、プログラムで扱うデータサイズやスレッド当たりの計算量を考慮した総スレッド数からブロック数を調整することで最適なスレッドブロック数を決定できると考えられる。

このスレッド数とブロック数の関係をヒントに、入力するプログラムの内容を解析すれば、最適なスレッドブロック数をある程度割り出せるのではないかと考えられる。

4.2 シェアードメモリの利用に関するチューニング

格納するデータサイズの調整

データサイズを調整する方法の 1 つにブロッキングがあげられる。ブロッキングは、プログラム中の 2 重ループ構文を 3 重ループ構文などに書き換えることでデータサイズを調整する方法である。

そこで、シェアードメモリへ格納するデータに対してもこの方法が有効であると考えられる。

バンクコンフリクトの回避

バンクコンフリクトを回避する方法の一つにパディングがあげられる。パディングは、利用しないメモリバンクを含めてメモリバンクサイズを大きめに確保することで、スレッドのアクセスパターンを変更する方法である。

そこで、メモリバンクを確保する際にこの方法を適用することでバンクコンフリクトの回避が可能であると考えられる。

5 進捗状況

これまで、CUDA や GPU アーキテクチャについて理解を深めた。また OMPCUDA の CUDA プログラム生成過程を検証した。更に、行列積などの CUDA プログラムについて、スレッドブロック数毎の計算性能を比較しながらその傾向について検証した。

例えば図 2 は、ベクトルの内積計算を行う CUDA プログラムにおけるブロック数毎の実行時間を比較した図である。この CUDA プログラムのベンチマークテストでは、ブロック数（スレッド数は 1 つで固定）を一定まで増やすことで実行時間が短くなることを確認した。

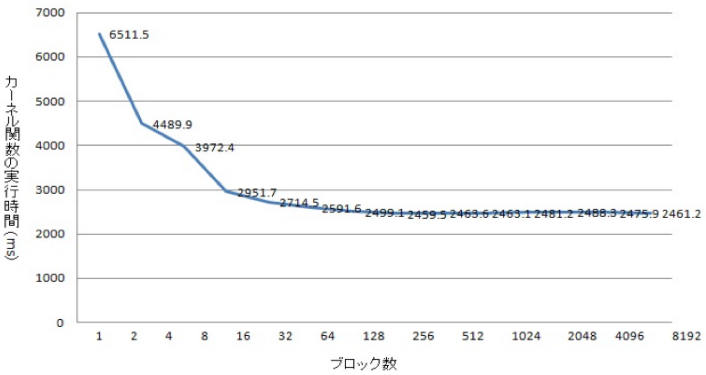

図 2: ベクトルの内積計算におけるブロック数毎の実行時間

6 今後の方針

今後は、最適なスレッドブロック数とシェアードメモリ利用のためのチューニングについての観点からベンチマークプログラムを実行して、計算性能にどのような傾向があるのか検証する。また、その検証結果などを元に OMPCUDA にチューニング機構を実装する。

最後に、チューニング機構を実装した OMPCUDA と既存の OMPCUDA、あるいはハンドチューニングした CUDA プログラムとで計算性能を比較することにより本研究の有用性を評価する。

参考文献

[1] 大島聡史, 平澤将一, 本多弘樹, ”OMPCUDA：GPU 向け OpenMP の実装”, 情報処理学会研究報告, pp.121-126(2008).

[2] A. Nukada and S. Matsuoka. Auto-tuning 3-D FFT library for CUDA GPUs. In Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis, pages 1—10. ACM, 2009.

[3] S. Lee, S.-J. Min, and R. Eigenmann, “OpenMP to GPGPU: A compiler framework for automatic translation and optimization,” in ACM SIG- PLAN Symposium on Principles and Practice of Parallel Programming(PPoPP). New York, NY, USA: ACM, Feb. 2009, pp. 101—110.