2G-04

CUDAによる並行処理のためのデータ転送のスケジューリング手法の提案

Scheduling Data Transfers in CUDA for Efficient Concurrency

甲田亮太　杉野栄二　成田匡輝　猪股俊光

所属:岩手県立大学ソフトウェア情報学部

1. はじめに

近年，情報技術の発展とともに大きな計算能力が要求される中で，大量のプロセッサが搭載されたGPUを汎用計算に用いるGPGPUが注目を集めている．NVIDIA社のGPUのアーキテクチャCUDAでは，CとC++に対してAPIを提供しており，従来に比べ容易なGPGPUプログラミングを可能にしている．GPUにおける複数のタスク実行においてCPU - GPU間のデータ転送とGPUでの計算を並列実行することで高速化を図ることができる．しかし，その効果は命令の依存関係と発行順序に依存し，効率的なGPGPUアプリケーション開発を難しくしている．

本研究ではCUDAでの複数タスクの並列実行におけるデータ転送のスケジューラの実装と評価を行い，GPUアプリケーション開発の効率化を図る．

2. 背景

本節では，本研究における関連技術，先行している関連研究について述べる．

2.1. CUDA 非同期 API

CUDAではデータ転送とカーネル実行において同期APIと非同期APIの2種類のAPIが提供されている．同期APIは，処理の実行が終了されるまでホスト(CPU)側プログラムの実行がロックされ，処理が終了した後に次の処理を実行できる．同期APIを用いた場合，複数種類のタスクについて，GPUは逐次的に処理を行う．これに対して，非同期APIはGPUへの命令の発行のみがなされて即時ホスト側に制御が戻る．非同期APIを連続して用いた場合，リソースが許す限りは複数のタスクが並列に実行される．

後述する本研究の実験に用いるGPUでは，データ転送ラインが2本あるため，方向が違う2つのデータ転送を並列実行できる．またカーネル実行はタスクに割り当てられるコアが競合しない限り並列に実行できる．データ転送・カーネル実行時間はタスクによって異なるため，並列実行するタスクのスケジュールにより並列実行全体の実行時間は変化する．

2.2. 関連研究

Teng Li氏らはCUDAのプロファイラから得られる情報を用いて，発行順序を決定するアルゴリズムを提案した[1]．こちらはプロファイラによる事前実行で得た情報を使った最適化であるのに対し，本研究は実行時に得られる情報を用いた手法を提案する．また，F.Wende氏らはカーネル関数実行が同一ストリームに連続して発行されないようにするKernel Reorderingを提案した[2]．本研究ではカーネル関数ではなくデータ転送に着目して命令の発行を行うことを目的とする．

3. 提案手法

本研究で提案する手法を示す．データ転送の発行順序を自動的に決定するためにデータ転送量に着目する．転送するデータサイズが大きいほど，転送時間が長くなる．そのため，カーネル実行前のCPU → GPU方向のデータ転送量が小さいものを優先して実行した方が早くカーネル実行を開始することができ，資源の遊休時間の削減ができる可能性がある．

4. スケジューラの設計

本研究の提案手法に基づくGPU向けタスクスケジューラの設計を行った．スケジューラの構成を図1に示す．

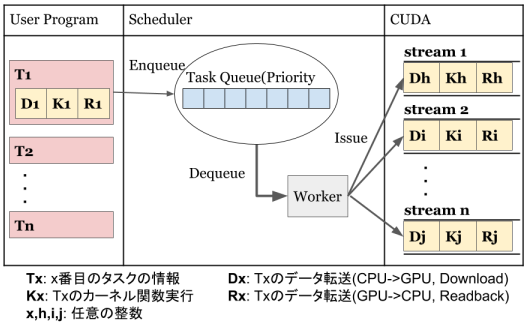

図1 提案スケジューラの構成

スケジューラはタスクキューとワーカーから構成される．ユーザプログラムは節5.で示すAPIを用い

て GPU で実行するタスクをタスクキューへ格納する．タスクキューは転送データサイズを優先度とする優先度付きキューとする．ワーカーはタスクキューから優先度の最も高いタスクを取り出し CUDA ストリームへ処理を発行する．

5. スケジューラの API

スケジューラへのタスク情報の発行のためにソースコード 1 に示す API を実装した．

ソースコード 1 タスクスケジューラの提供する API

```
1 task_schedule(kernel, "Iff|f", DIM3(1,1,1),
      DIM3(1,1,1), N, h1, h2, h3);
2 task_schedule(kernel, "Ii|i", DIM3(1,1,1),
      DIM3(1,1,1), M, h4, h5);
```

task_schedule() 関数は，提案スケジューラにタスクを登録する．引数に実行するカーネル関数ポインタ，カーネル関数引数のフォーマット文字列，ブロック数，スレッド数，カーネルに渡す実引数を指定する．第 2 引数の引数フォーマット指定子は表 1 の通りである．

表 1 引数フォーマット指定子と型の対応

指定子	意味	指定子	意味
i	int*	d	double*
f	float*	I	int
|	右側の指定子が戻り領域		

6. 評価

本研究の提案手法の有用性の確認のために以下の実験及び評価を行う．

6.1. 評価環境

評価環境は GPU は NVIDIA 社の Maxwell アーキテクチャを採用した GeForce GTX 970 を選定した．

6.2. 評価方法

実験は異なる大きさの正方行列積を計算するカーネルを 4 つ実行する処理を CUDA の逐次実行と非同期 API，提案スケジューラの 3 種類の方法で 10 回実行し，平均の実行時間を比較して評価する．4 つのカーネルで用いる正方行列の大きさは 3 通り用意した．

・Test1 縦横 256, 384, 512, 1024 の順に命令発行
・Test2 縦横 512, 512, 512, 512 の順に命令発行
・Test3 縦横 1024, 512, 384, 256 の順に命令発行

実験の結果を図 2 に示す．

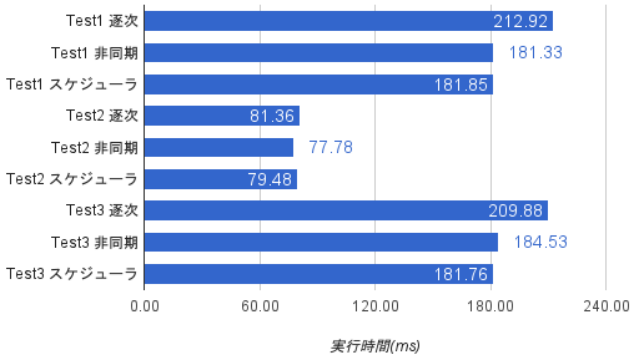

図 2 実験結果

Test1 では非同期 API とスケジューラは同じ命令発行順になる．スケジューラの処理コストの分，スケジューラの実行時間は約 0.5ms 増加している．Test3 では，非同期 API ではデータ転送量の最も大きいサイズが先に転送され，計算資源の利用が遅れるため，スケジューラでの実行時間が約 3ms の高速であった．スケジューラによる高速化の効果は最小転送量と最大の転送量の差が大きいほど大きくなると期待できる．

7. おわりに

本研究では GPGPU アプリケーションの開発の簡易化のためにデータ転送のスケジューリング手法の提案を行い，この提案に基づくスケジューラを実装した．

参考文献

1) Teng Li, Vikram K. Narayana and Tarek El-Ghazawi: Reordering GPU Kernel Launches to Enable Efficient Concurrent Execution, arXiv(2015)
2) F. Wende, F. Cordes and T. Steinke: On Improving the Performance of Multi-threaded CUDA Applications with Concurrent Kernel Execution by Kernel Reordering, IEEE(2012)