GPUを用いた高スループット計算システムの実装

谷　和也
広島大学大学院工学研究科

概 要

近年，膨大なデータの処理は現代のコンピュータ・システムの課題として広く知られている．本研究では，膨大なデータに対して処理を行う際の高スループットな計算システムの実装を提案する．実装にはGPUとNVIDIA社が提供するGPU向けの統合開発環境であるCUDA[1]を使用する．GPU(Graphics Processing Units)とはグラフィクス処理用に開発されたハードウェアであり，多数の演算コアを搭載している．これを汎用演算に利用し，システムを実現する．本システムでは膨大なデータに対しては分割して処理を行う．性能の評価として膨大な数の32点FFTを計算するプログラムを作成し，本システム上で実行したときの結果を考察する．GPUとしてGeforce GTX TITANを使用して構築した本システムでは50ギガバイトの膨大な32点FFT入力データに対して，32MBごとに分割して処理したときが総処理時間12799.53ms，スループットは3.91GB/secとなり，処理時間とスループットのバランスに優れていた．

1. はじめに

GPU(Graphics Processing Units)とは多数のプロセッサコアが搭載されたハードウェアである．多数のコアで並列計算が可能で，非常に高い演算能力を有している．もとはグラフィクス処理用に開発されたハードウェアであったが，近年ではその演算能力を汎用演算に用いるGPGPU(General-Purpose computing on Graphics Processing Units)と呼ばれる試みが盛んに行われており，GPGPU向けのGPUが多く開発されている．

本研究で用いたCUDAとはNVIDIA社が提供するGPGPU向けの統合開発環境である．GPUに対応したソフトウェアアーキテクチャであり，GPU上の多数のコアに対してスレッドを割り当て並列計算を行う．複数のスレッドの集合はブロックと呼ばれ，GPUのストリーミングマルチプロセッサに割り当てられる．GPUには全てのスレッドが共有するグローバルメモリと，同じブロック内でのみ共有するシェアードメモリが存在する．本研究では，膨大なデータ処理のため，比較的大容量であるグローバルメモリを使用して実装を行う．

CUDAを使用して開発したプログラムはCPUによる処理部分を記述したコードと，GPUによる処理部分を記述したコードからなる．GPU上で実行される関数はカーネルと呼ばれ，CPU側のコードから呼び出される．GPUのメモリ領域はCPUから独立しているため，カーネルが処理するデータはGPUメモリ上へと転送してやる必要がある．当然，処理終了後に結果を得るためには，同様にGPUメモリからCPUメモリ上へと転送しなければならない．以上のことから，GPUを用いた処理は以下の3ステップで行われる．

- データ転送 (HostToDevice)
- カーネル実行 (Kernel)
- データ転送 (DeviceToHost)

ここではCPUはホストと呼ばれ，GPUはデバイスと呼ばれている．すなわち，HostToDeviceとはCPUからGPUへのデータを転送を意味する．全体的な手順としては，まず入力データがCPU側で作成されGPUへと転送される．その後CPU側からカーネルが呼び出され，GPUのグローバルメモリにあるデータに対して処理が行われる．そして得られた結果は再びCPUへと送り返される．

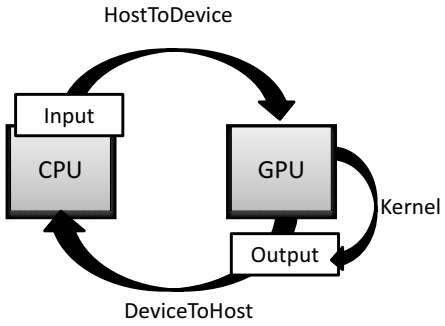

図 1: CPU-GPU プログラムの概要

2. CUDA アーキテクチャ

図2にGPUのハードウェアアーキテクチャを示す．GPUにはSM(Streaming Multiprocessor)が複数あり，SMには多数のプロセッサコアが存在している．各SM内にあるシェアードメモリは同一のSM内でのみ共有可能なメモリ領域で，アクセスが非常に速い代わりに容量が小さいという特徴を持っている．一方で各SMが接続されているグローバルメモリは全てのプロセッサコアで共有が可能なメモリである．ただし，グローバルメモリはシェアードメモリと比べてアクセスのオーバーヘッドが大きいため，効率的なアク

セスをするのが重要である．

GPU のハードウェアアーキテクチャに対応するソフトウェアアーキテクチャが CUDA である．図 3 に CUDA アーキテクチャを示す．CUDA はスレッド，ブロック，グリッドが GPU と対応して階層構造になっている．スレッドが処理の最小単位であり，GPU におけるプロセッサコアに対応する．同様にスレッドの集合であるブロックは SM に対応し，グリッドは GPU 全体と対応している．

グローバルメモリアクセス

本研究では膨大な数のデータを扱うため，大容量のグローバルメモリを使用している．グローバルメモリアクセスでは各スレッドが連続したメモリ領域へアクセスする場合は一度でアクセスすることができる．これはコアレスドアクセスと呼ばれている (図 4)．したがって CPU から GPU へと入力データが渡される際に，メモリアクセスがコアレスドアクセスとなるようなデータに並べ替えてから転送している．この並び替えの方法が図 10 にあるが，これについてはまた第 3. 章で詳しく説明する．

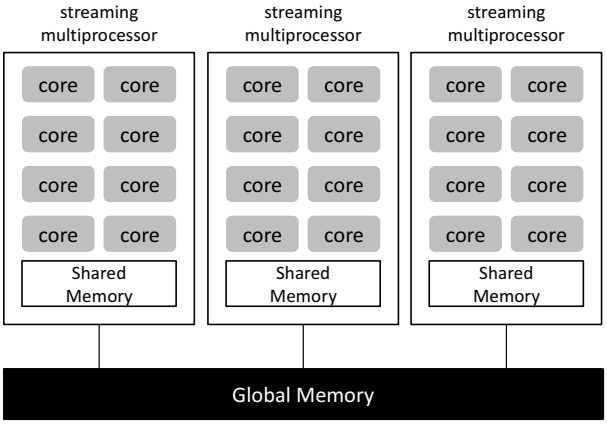

図 2: GPU アーキテクチャ

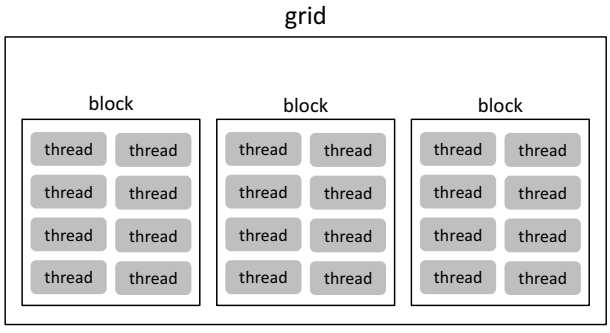

図 3: CUDA アーキテクチャ

ストリーム

CUDA におけるストリームには GPU の処理を管理するキューとしての役割がある．デフォルトではストリームは 1 つのみ用意されており，GPU へ発行した命令は発行された順に従って逐次的に実行されることが保証されている．また，ストリームは複数生成することができ，プログラマが任意のストリームに命令を投入していくことが可能になっている．ストリーム間で異なる命令を処理することができるため，分割した各データに対するメモリコピー及びカーネル実行を各ストリームに割り当てれば別々に結果を得ることも可能である．また，今回の研究には使用していないが，ストリームにはもっと発展的な機能が備わっている．それは，ストリーム同士には依存関係がなく独立に処理されるため，非同期に実行が可能でストリーム同士の並列処理が可能であるということである．これにより，処理をオーバーラップすることで全体の処理時間をさらに短縮できる．しかしオーバーラップの可能性はハードウェアに依存し，オーバーラップした場合は処理の順序が複雑化するため，これについては今後の課題として考えている．

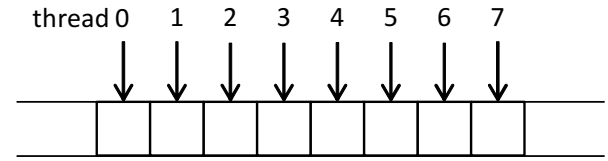

図 4: コアレスドアクセス

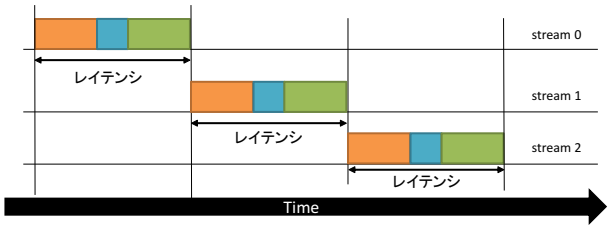

図 5: ストリームを利用したデータの分割処理

3. 実装方法

本研究では膨大な数のデータが外部のソースから計算を行うデバイスに送信されてくる状況を想定し，そのような状況下で高いスループットを保ったまま，効率よく出力を得るためのシステムを提案する．外部のデータソースを CPU，計算を行うデバイスを GPU で構築する．このシステムでは通常の GPU を用いた演算手法ではデータの数が多いほど転送時間と処理時間が長くなるため効率が良いとはいえない．膨大なデータの転送と処理を一括で行うのは結果を得るまでに時間がかかることになる．ここで，GPU への転送開始から計算結果の CPU 転送が終了するまでの時間をレイテンシと呼ぶものとする．

そこで，分割した入力データを一定の周期で GPU に転送するようなプログラムを作成することで，CPU から GPU へデータが流れてくる状況を実現した．図

6 は一定の周期でデータが転送されるプログラムの概要を表している．左の時間軸は CPU のタイムラインであり，このタイムラインにおいて 10ms ごとにコールバック関数が呼び出されている．コールバック関数は第 1. 章で紹介した通り，GPU の処理の基本である 3 ステップを含んでいる．この 3 ステップの GPU 処理が完了したらコールバック関数を終了し，次の周期が来るまでは CPU はスリープさせておく．データの到着周期よりも GPU の処理にかかる時間が長くなると，現実のシステムにおいて入力されてくるデータに対して処理が間に合わなくなり，エラーが発生する可能性がある．したがって，本研究のシステムでは GPU 処理にかかる時間が周期よりも長くならないように，GPU の処理の実行時間を予め計測しておき，その時間を例外なくカバーできるような周期を設定する必要があった．図 7 および図 8 に分割サイズが 32MB と 128MB の 2 つにおいて全レイテンシをグラフ化したものを示す．このグラフからレイテンシに大きく外れる値はほとんどないとわかる．また，32MB よりも 128MB のほうが値の振れ幅が小さいため，データを分割するサイズが大きい場合，レイテンシは安定した値を計測できる．このようにして全ての分割サイズについて求めた周期は表 1 にある．

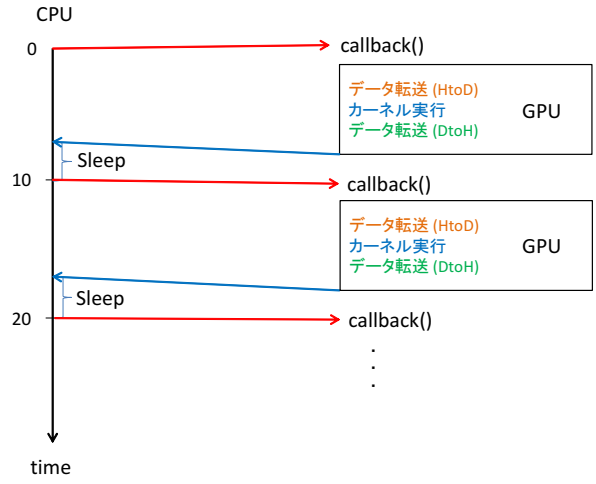

図 6: 周期 10ms でデータが GPU へ転送されるプログラム

分割サイズ (MB)	1	2	4	8	16	32	64	128	256	512	1024
周期 (ms)	2	2	2	3	5	8	16	30	60	120	240

表 1: 分割サイズごとのコールバック関数を呼ぶ周期

このシステムでは分割されたデータごとに処理が行われるため，計算結果を細かく得ることが可能となる．しかし，分割するサイズによっては良い結果をもたらさない可能性が考えられる．例えば，小さく分割し過ぎた場合は実行時間の短い多数のカーネルを実行することになり，カーネル実行のオーバーヘッドにより全体の実行時間が長くなることがある．全体の実行時間やスループットのパフォーマンスの低下を抑えつつ，レイテンシを短縮できるような効率的な分割サイズを発見するため，FFT のプログラムを用いて実験を行った．

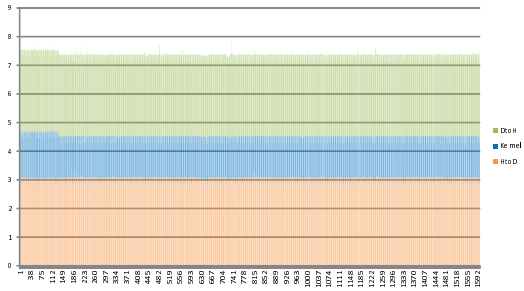

図 7: 分割サイズ 32MB のときのレイテンシ

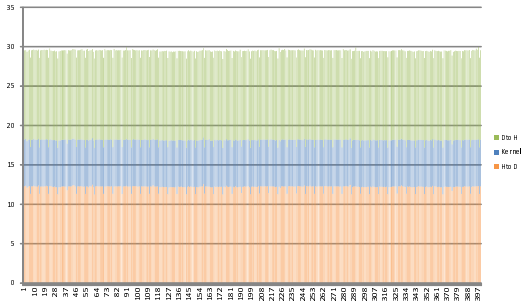

図 8: 分割サイズ 128MB のときのレイテンシ

32 点の FFT を CUDA の各スレッドで実行するプログラムを作成し，このプログラムに対して膨大な数の入力データを与えて実験を行った．具体的には $4*10^6$ 個の FFT の入力データ (1GB) を 1 つの配列内に用意し，これを 50 回繰り返し利用することで $200*10^6$ 個の FFT 入力データとして使用している．この入力データ配列 (1GB) を CUDA のストリームという機能を用いて分割して処理することでレイテンシの短縮を試みる．様々な分割サイズで時間の計測を行い，パフォーマンスを確認した結果，データを適切に分割すれば全体の実行時間をほとんど延ばすことなくスループットについても高い値を保つことができた．

CUDA による FFT の実装

高速フーリエ変換 (FFT) は離散フーリエ変換を高速に計算するアルゴリズムである [3]．本研究では 1 次元 FFT である Cooley-Tukey 型 FFT と呼ばれるアルゴリズム [2] の実装を行った．ここでは FFT の実装方法ではなく，FFT の実装に CUDA アーキテクチャがどう使用されたかということについて説明する．本研究では膨大な数の FFT データを処理するために，各スレッドが FFT を逐次的に計算している．FFT の入力データは全てまとめて 1 つの配列に格納されており，各スレッドはその配列の中の自分の担当する領域にアクセスする．このスレッドの振る舞いを

示しているのが図 9 である．

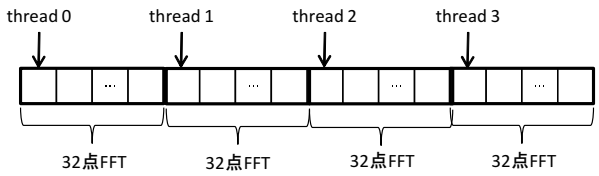

図 9: FFT 入力データに対するスレッド割り当て

このアクセス方法では各スレッドが連続した領域にアクセスしていないのでコアレスドアクセスにはならない．コアレスドアクセスを行うようにするためには，配列をブロックごとに転置してから GPU のメモリへとコピーすればよい (図 10)．FFT では 32 点へのアクセス順は一定になる特徴があるため，ブロック内の全スレッドのアクセスが連続した領域に対して行われコアレスドアクセスとなる．

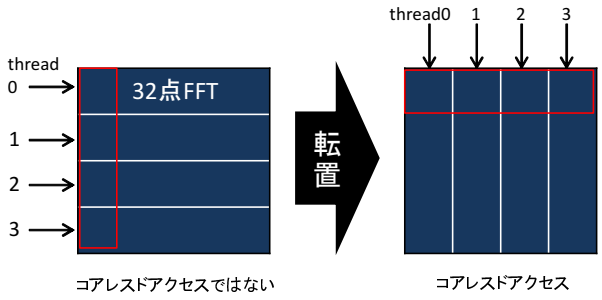

図 10: GPU に転送する前の転置処理

4. 実験結果・考察

実験に使用した GPU は NVIDIA Geforce GTX TITAN でストリーミングマルチプロセッサが 14 基，プロセッサコアが 2688 個，グローバルメモリは 6GB 搭載されている．使用した CPU は Intel Core i7 である．GPU を用いて 32 点 FFT を実行するプログラムを実装し，レイテンシ，総実行時間，及びスループットを計測した．$200*10^6$ 個の FFT データ (50GB) を入力とするプログラムを 11 通りの分割サイズで計測して結果を比較した．分割サイズは 1MB, 2MB, 4MB, 8MB, 16MB, 32MB, 64MB, 128MB, 256MB, 512MB, 1024MB の 11 通りである．

表 2 は分割サイズごとにスループット，レイテンシ，総実行時間を比較したものである．ただし，表 2 のスループットは分割されたデータ内で計測されたレイテンシの最大値である．より小さく分割する 1MB, 2MB, 4MB ではレイテンシが短くなる代わりに総実行時間がかなり長くなっている．一方で 1024MB ではレイテンシが最長となっている．計測結果についてのグラフを図 11 に示す．図 11 は総実行時間とレイテンシを表すグラフである．横軸には分割サイズとスループットが表示されている．グラフから，総実行時間とレイテンシのバランスが優れているのは 32MB 付近であることが見て取れる．このときスループットについても高い値を示している．

サイズ (MB)	スループット (GB/sec)	レイテンシ (ms)	総実行時間 (ms)
1	0.49	0.56	102396.28
2	0.98	0.73	51197.86
4	1.95	1.22	25598.51
8	2.60	2.58	19198.26
16	3.13	4.39	15998.85
32	3.91	7.85	12799.53
64	3.91	15.37	12798.81
128	4.17	29.95	11999.54
256	4.17	59.80	11998.84
512	4.17	119.97	12000.03
1024	4.17	239.73	11999.85

表 2: 分割サイズごとの計測値

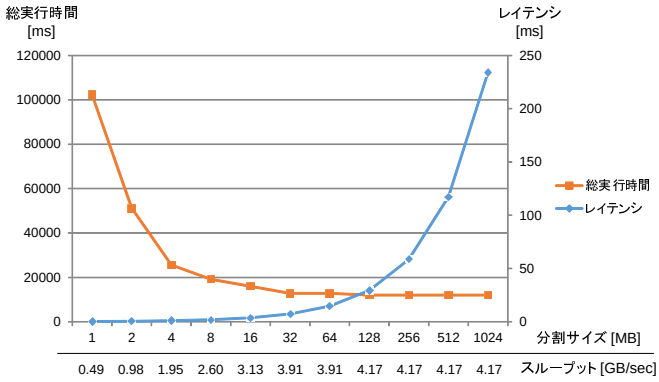

図 11: 計測結果のグラフ

5. まとめ

本研究では GPU を用いて膨大な数の入力データを分割し効率的に処理する高スループット計算システムを実装した．各スレッドが 32 点の FFT を実行するプログラムを分割サイズを変えて実行時間を計測し，計 11 通りのサイズで比較を行った．その結果，32MB のデータごとに処理を行うとき，高スループットを維持しながら実行時間とレイテンシのバランスが優れていることがわかった．

参考文献

[1] NVIDIA Corp., ”CUDA ZONE”, https://developer.nvidia.com/cuda-zone

[2] Takuya Ooura, ”FFT の概略と設計法”, http://www.kurims.kyoto-u.ac.jp/ ooura/fftman/

[3] 佐川雅彦・貴家仁志, ”高速フーリエ変換とその応用”, 1992