マルチプラットフォーム対応
並列プログラミングフレームワーク

平成26年度 学位論文(修士)

琉球大学大学院　理工学研究科
情報工学専攻

渡真利 勇飛

要 旨

Abstract

目 次

図 目 次

表 目 次

第1章　研究目的と背景

PC やタブレットの一般的な利用方法として動画の編集や再生、ゲームといったアプリケーションの利用が挙げられる。これらのアプリケーションはグラフィックや音質といった点から要求する処理性能が上がってきている。

しかし消費電力や発熱、クロックの限界から CPU の性能自体を上げることによる処理性能の向上は不可能となっている。その事からプロセッサメーカーはマルチコアやヘテロジニアス構成の路線を打ち出している。そういったアーキテクチャ上でリソースを有効活用するにはプログラムの並列化は必須と言える。

しかしプログラムを並列化するのみではリソースの有効活用としては不充分であり、実行の順番やタスクをどのリソース上で実行するかといった Scheduling も行わなければならない。こういった部分をサポートするプログラミングフレームワークが必要である。

当研究室では Cerium というプログラミングフレームワークを開発している。Cerium をマルチプラットフォームに対応させ、高い並列度を維持したプログラミングを可能にする。本研究ではマルチプラットフォーム上でのプログラムの実行における最適化について、Sort、Word Count、FFT を例題に考察していく。

1.1　本論文の構成

第2章　既存のマルチプラットフォームフレームワーク

2.1　OpenCL

OpenCL とは、マルチコア CPU と GPU のようなヘテロジニアスな環境を利用した並列計算を支援するフレームワークである。

OpenCL には主に 2 つの仕様がある。

- OpenCL C 言語
- OpenCL Runtime API

OpenCL C は演算用プロセッサ上で動作する、 C 言語を拡張したプログラミング言語である。一方で OpenCL Runtime API は OpenCL C で記述したプログラムを演算用プロセッサ上で実行させるため、制御用のプロセッサが利用する API である。

OpenCL では演算用プロセッサ側を device 、制御用デバイス側を host として定義する。また、 Device 上で動作するプログラムの事を kernel と呼ぶ。

2.1.1　Command Queue

OpenCL では、デバイスの操作に Command Queue を使用する。 Command Queue は Device に命令を送るための仕組みである。Command Queue は clCreateCommandQueue という OpenCL API で作成され、Command Queue が所属するコンテキストや実行対象となるデバイスを指定する。

kernel の実行、input data への書き込み、 output data の読み込みといったメモリ操作はこの Command Queue を通して行われる。

2.1.2　メモリアクセス

host では主に data を input/output するメモリ資源の確保を行う。GPU のメモリ空間 (図:2.1) はマルチコア CPU (図:2.2) と違い、共有メモリでないため host と kernel(Task) 間で data の共有ができない。

アクセスするにはメモリ空間間でコピーしなければならない。

OpenCL は host 側で memory buffer を作成してメモリのコピーを行う。これらの処理や Task は Command Queue に enqueue することで実行される。

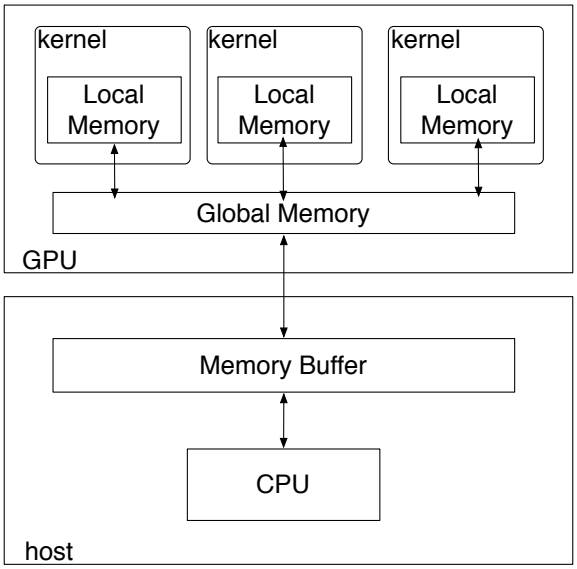

図 2.1: Gpu Architecture

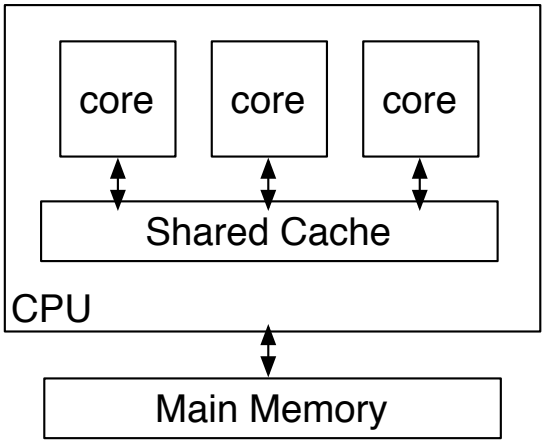

図 2.2: Cpu Architecture

2.1.3　データ並列

多次元のデータ構造を扱う計算において高い並列度を保つには、それを分割して並列に実行する機能が必要である。データ並列実行という。OpenCL はデータ並列実行もサポートしている。OpenCL は次元数に対応する index があり、OpenCL は一つの記述から異なる index を持つ複数の kernel を自動生成する。その添字を global_id と呼ぶ。この時入力されたデータはワークアイテムという処理単位に分割される。

OpenCL はワークアイテムに対してそれぞれを識別する ID (global_id) を割り当てる。kernel は get_global_id API によって ID を取得し、取得した ID に対応するデータに対して処理を行い、データ並列を実現する。この ID によって取得してきたワークアイテムをグローバルワークアイテムという。また、ワークアイテムは 3 次元までのデータを渡すことができる。

データ並列による kernel 実行の場合は clEnqueueNDRangeKernel API を使用するが、この関数の引数としてワークアイテムのサイズと次元数を指定することでデータ並列で実行できる。

2.1.4　ワークグループ

前節でワークアイテムという処理単位について述べたが、さらに複数個のグローバルワークアイテムを work_group という単位にまとめることができる。work_group 内では同期やローカルメモリの共有が可能となる。

グローバルワークアイテム (ワークアイテム全体) の個数と、ローカルワークアイテム (グループ一つ辺りのアイテム) の個数を指定することでワークアイテムを分割する。なお、このときグローバルワークアイテム数はローカルアイテム数の整数倍でなければ clEnqueueNDRangeKernel API 呼び出しは失敗する。

ローカルアイテム数は 0 を指定することで、コンパイル時に最適化させることができる。したがってローカルアイテムのサイズは 0 を指定するのが一般的である。

なお、 work_group を設定した場合は global_id の他に work_group_id 、local_id がそれぞれの kernel に割り当てられる (図:2.3)。

なお、work_group を設定した場合は global_id の他に work_group_id、local_id がそれぞれの kernel に割り当てられる (図:2.3)。

kernel 側からそれぞれ ID に対応した API を使用して、各 ID を取得する。取得した ID から自分が担当する index を計算して導く。表:2.1 は kernel 側で使用できる、 ID を取得するための API となる。 なお、 local_id 、global_id を取得する API は引数に 0、

get_group_id	work_group_id を取得
get_local_id	local_id を取得
get_global_id	global_id を取得

表 2.1: kernel で使用する ID 取得の API

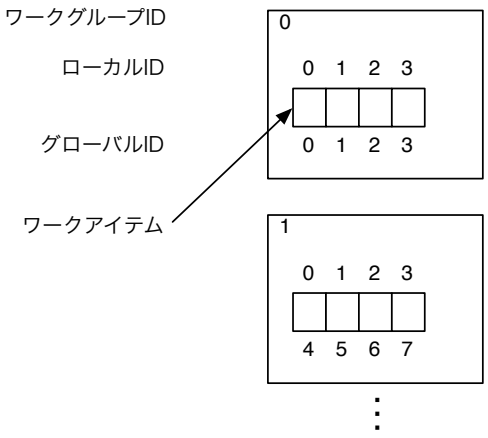

図 2.3: WorkItem ID

1、2 の値を set することができる。id は x, y, z 座標があり、それぞれが 0, 1, 2 に対応している。例えば get_global_id(1) と呼び出した場合は y 座標の、get_global_id(1) と呼び出した場合は z 座標の global_id を取得する。

2.2　CUDA

CUDA とは、半導体メーカー NVIDIA 社が提供する GPU コンピューティング向けの総合開発環境である。

CUDA には主に 3 つの仕様がある。

- CUDA C
- CUDA Runtime API
- CUDA Driver API

CUDA C は GPU 上で動作する、C 言語を拡張したプログラミング言語である。CUDA Runtime API も CUDA Driver API も CUDA C で記述したプログラムを GPU 上で実行させるために制御用プロセッサが利用する API である。Driver API は Runtime API に比べ、プログラマが管理しなければならないリソースが多くなる代わり、より柔軟な処理を行う事ができる。

CUDA も OpenCL と同様、演算用プロセッサ (GPU) を Device 、制御用デバイス側を Host として定義する。また、 Device 上で動作するプログラムの事も kernel と呼ぶ。

2.2.1　Stream

OpenCL における Command 、 CommandQueue に対応するものとして、 CUDA には Operation と Stream がある。Stream は Host 側で発行された Operation を一連の動作として Device で実行する。Stream に発行された Operation は発行された順序で実行されることが保証されている。更に、異なる Stream に発行された Operation も依存関係が存在しない場合、Operation は並列に実行される。

Stream は cuStreamCreate という Driver API で生成される。引数に Stream を指定しない API は全て host 側をブロックする同期的な処理となる。複数の Stream を同時に走らせ、 Operation を並列に実行するためには非同期的な処理を行う API を利用する必要がある。

2.2.2　データ並列

CUDA では OpenCL の WorkItem に相当する単位を thread として定義している。この thread をまとめた単位として block がある。

CUDA でデータ並列による kernel 実行を行う場合、cuLaunchKernelAPI を使用する。この関数は引数として各座標の block 数、各座標の block 1 つ辺りの thread 数を指定することによりデータ並列実行を行う。

cuLaunchKernel で kernel を実行すると各 thread に対して blockID と threadID が割り当てられる。CUDA には OpenCL と異なり、ID を取得する API が存在しない。それに代わり、 kernel に組み込み変数が準備されている。その組み込み変数を参照し、対応するデータに対し処理を行うことでデータ並列実行を実現する。組み込み変数は以下の 3 つである。

- uint3 blockDim
- uint3 blockIdx
- uint3 threadIdx

3 つの組み込み変数はベクター型で、 blockDim.x とすると x 座標の thread 数を参照することができる。同じように blockID 、 threadID の x 座標を参照することができる。blockDim.x * blockIdx.x + threadIdx.x とする事で OpenCL における get_global_id(0) で取得できる ID に相当する値を算出する事ができる。

例としてある kernel で get_global_id(0) の値が 8 の時、CUDA では 図 2.4 のように算出する。

2.3　StarPU

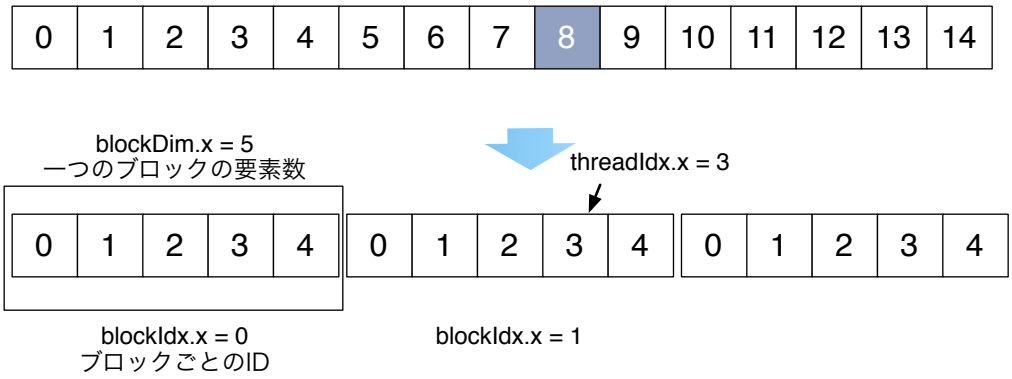

図 2.4: Calculate Index example

第3章　Cerium

3.1　Cerium における Task

3.2　Task の Scheduling

第4章　Ceriumを用いた例題

4.1　WordCount

4.2　Sort

4.3　FFT

第5章　OpenCLによるGPGPUへの対応

5.1　GPU上での実行の機構

5.2　ベンチマーク

第6章　CudaによるGPGPUへの対応

6.1　GPU上での実行の機構

6.2　ベンチマーク

第7章　データ並列

7.1　データ並列実行の機構

7.2　iterate API

7.3　ベンチマーク

第8章　Memory Allocator

8.1　現状のMemory Allocator

8.2　新しいMemory Allocator

8.3　ベンチマーク

第9章　結論

9.1　まとめ

9.2　今後の課題

謝辞

本研究を行うにあたりご多忙にも関わらず日頃より多くの助言, ご指導をいただきました河野真治准教授に心より感謝いたします.
研究を行うにあたり, 並列計算環境の調整, 意見, 実装に協力いただいた谷成 雄さん, 杉本優さん, 並びに並列信頼研究室の全てのメンバーに感謝いたします.
また, 本研究は, JST/CREST 研究領域「実用化を目指した組み込みシステム用ディペンダブル・オペレーティングシステム」D-ADD 研究チームとして実施された. 様々な研究や勉強の機会を与えてくださった, 株式会社 Symphony の永山辰巳さん, 同じく様々な助言を頂いた森田育宏さんに感謝いたします. 様々な研究に関わることで自身の研究にも役立てることが出来ました.
最後に, 大学の修士まで支えてくれた家族に深く感謝します.

参考文献

[1] Messagepack. http://msgpack.org/.

[2] 大城信康, 河野真治. Data segment の分散データベースへの応用. 日本ソフトウェア科学会, September 2013.

[3] 玉城将士, 河野真治. Cassandra を使ったスケーラビリティのある cms の設計. 情報処理学会, March 2011.

[4] 玉城将士, 河野真治. Cassandra と非破壊的構造を用いた cms のスケーラビリティ検証環境の構築. 日本ソフトウェア科学会, August 2011.

[5] Avinash Lakshman and Prashant Malik. Cassandra - a decentralized structured storage system. *LADIS*, Mar 2003.

[6] Fay Changand Jeffrey Dean, Sanjay Ghemawat, Wilson C. Hsieh, Deborah A. Wallach, Mike Burrows, Tushar Chandra, Andrew Fikes, and Robert E. Gruber. Bigtable : A distributed storege system for structured data.

[7] Nancy Lynch and Seth Gilbert. Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services. *ACM SIGACT News*, 2002.

[8] Giuseppe DeCandia, Deniz Hastorun, Madan Jampani, Gunavardhan Kakulapati, Avinash Lakshman, Alex Pilchin, Swaminathan Sivasubramanian, Peter Vosshall, and Werner Vogels. Dynamo: Amazon's highly avaliable key-value store.

[9] 所眞理雄. DEOS プロジェクト研究成果集 Dependability Engineering for Open Systems, 2013.

[10] 永山 辰巳, 横手 靖彦. オープンシステムディペンダビリティと D-Case を繋ぐリポジトリ, 2013.

発表履歴

- Java による授業向け画面共有システムの設計と実装, 大城信康, 谷成雄 (琉球大学), 河野真治 (琉球大学), オープンソースカンファレンス 2011 Okinawa, Sep, 2011
- Continuation based C の GCC 4.6 上の実装について,
  大城信康, 河野真治 (琉球大学),
  第 53 回プログラミング・シンポジウム, Jan, 2012
- GraphDB 入門 TinkerPop の使い方,
  大城信康, 玉城将士 (琉球大学),
  第 15 回 Java Kuche, Sep, 2012
- ディペンダブルシステムのための木構造を用いた合意形成データベースの提案と実装,
  大城信康, 河野真治 (琉球大学), 玉城将士 (琉球大学), 永山 辰巳 (株式会社 Symphony),
  情報処理学会システムソフトウェアとオペレーティング・システム研究会 (OS), May, 2013
- Data Segment の分散データベースへの応用,
  大城信康, 杉本優 (琉球大学), 河野真治 (琉球大学),
  日本ソフトウェア科学会 30 回大会 (2013 年度) 講演論文集, Sep, 2013