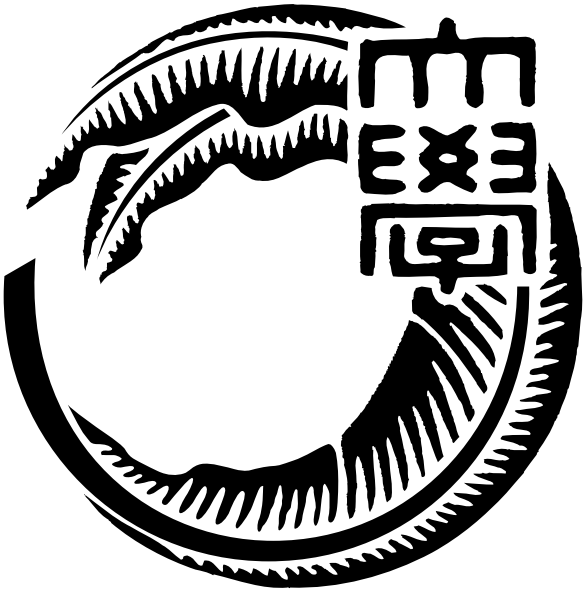
平成27年度 卒業論文

Code Gear Data Gear によるGPGPU処理実装



琉球大学工学部情報工学科

135704C 東恩納 琢偉

指導教員 河野 真治

目 次

図 目 次

表 目 次

第1章　GearsOSでのGPU実行

CPU の処理速度の向上のためクロック周波数の増加は発熱や消費電力の増大により難しくなっている。
そのため、クロック周波数を上げる代わりに CPU のコア数を増やす傾向にある。マルチコア CPU の性能を発揮するには、処理をできるだけ並列化しなければならない。また、PC の処理性能を上げるためにマルチコア CPU 以外にも GPU や CPU と GPU を複合したヘテロジニアスなプロセッサが登場している。並列処理を行う上でこれらのリソースを無視することができない。
しかし、これらのプロセッサで性能を出すためにはこれらのアーキテクチャに合わせた並列プログラミングが必要になる。
並列プログラミングフレームワークではこれらのプロセッサを抽象化し、CPU と同等に扱えるようにすることも求められる。
そのためには並列処理を行うプログラムの依存関係を解決することや、本研究室で開発している Gears OS は Code Gear と Data Gear によって構成される。Code Gear は処理の単位、Data Gear はデータの単位となる。Gears OS では Code/Data Gear を用いて記述することでプログラム全体の並列度を高めて、効率的に並列処理することが可能になることを目的とする。
また、Gears OS の実装自体が Code/Data Gear を用いたプログラミングの指針となるように実装する。
Gears OS における Task は実行する Code Gear と実行に必要な Input Data Gear, 出力される Output Data Gear の組で表現される。Input/Output Data Gear によって依存関係が決定し、それに沿って並列実行する。
依存関係の解決などの Meta Computation の実行は Meta Code Gear で行われる。Meta Code Gear は Code Gear に対応しており、 Code Gear が実行した後にそれに対応した Meta Code Gear が実行される。これらのことから、並列プログラミングで必要とされる GPU やマルチコア CPU を CPU と同等に扱うためには依存関係を明確に出来る Gears OS で CUDA による GPU 実装を行い、CbC による GPU 処理での有用性を発見することが本研究の目的である。

第2章　GearsOS

2.1　Continuation Baseed C

Gears OS の実装は本研究室で開発している CbC(Continuation based C)[?] を用いて行われている。。

CbC は処理を Code Segment を用いて分割して記述することを基本としている。、 Gears OS の Code Gear は CbC を元に記述されている。CbC のプログラムでは C の関数の単位で Code Segment を用いて分割し、処理を記述している。Code Segment は C の関数と異なり戻り値を持たない。Code Segment の宣言は C の関数の構文と同様に行い、 型に __code を使い宣言している。Code Segment から Code Segment への移動は goto の後に移動先の Code Segment 名と引数を並べた記述する構文を用いて行う。この goto による処理の遷移を継続と呼び、C での関数呼び出しにあたり、C では関数の引数の値がスタックに積まれていくが、Code Segment の goto では戻り値を持たないため、スタックに値を積んでいく必要がなくスタックを変更する必要がない。このようなスタックに積まない継続を軽量継続と呼び、呼び出し元の環境を持たない。図 2.1 は Code Segment 間の接続関係を表している。

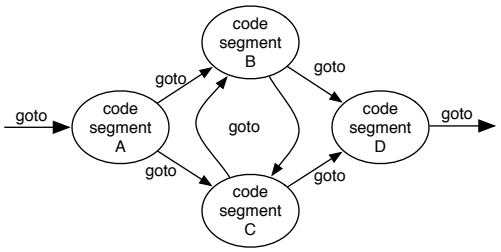


図 2.1: goto による Code Segment 間の接続

2.2 Code Gear と Data Gear

Code Gear はプログラムの実行コードそのものであり、OpenCL[?]/CUDA[?] の kernel に相当する。

Code Gear は処理の基本として、Input Data Gear を参照し、一つまたは複数の Output Data Gear に書き込む。また、接続された Data Gear 以外には参照を行わない。Input Data Gear と Output Data Gear の 2 つによって、Code Gear の Data に対する依存関係を解決し、 Code Gear の並列実行を可能とする。

Code Gear は CbC を元に記述されており、処理の移行は function call ではないので、呼び出し元に戻る概念はない。その代わりに、次に実行する Code Gear を軽量継続の goto で指定する。

Data Gear は、int や文字列などの Primitive Data Type の組み合わせ（struct）である。Data Gear は様々な型を持つ union として定義される。

Gear の特徴の一つはその処理が Code Gear, Data Gear に閉じていることにある。これにより、Code Gear の実行時間、メモリ使用量を予測可能なものにする。

2.3 並列性

Code Gear が処理するのに必要な Input Data Gear と処理の実行後に出力される Output Data Gear の組を Task と呼び、Code Gear は接続する Task 以外とは依存関係がなく、また Input Data Gear があれば Task に従って並列処理を行う事が出来る。

2.4 柔軟性

Code Gear Data Gear は CbC を用いて記述されており、処理の遷移には goto を用いた軽量継続によって移行している。また Code Gear は C の関数と同じ単位で分割されており、処理の遷移に用いられている goto の遷移先を変更することで、容易に処理の順番を変更や新しい処理を追加することが出来る

2.5 Gears でのMeta computation の実行

Gears OS では通常の処理を Computation、 Computation のための Computation を Meta Computation として扱う。例として、 Code Gear が次に実行する Code Gear を goto で名前指定する。この継続処理に対して Meta Code Gear が名前を解釈して、処理を対応する Code Gear に引き渡す。これらは、従来の OS の Dynamic Loading Library や Command 呼び出しに対応する。名前と Code Gear へのポインタの対応は Meta Data Gear に格納される。この Meta Data Gear を Context と呼ぶことにする。これは従来の OS の Process や Thread を表す構造体に対応する。

第3章　Gears OSの構成

3.1　TaskManager

Gears OS の TaskManager は WaitTaskQueue に入っている Task の依存関係を解決する。Task には Input/Output Data Gear の情報が格納されている。Input Data Gear は Task に必要な Data Gear で揃ったら Task は実行可能な状態になる。Output Data Gear は Task が Persistent Data Tree に書き出す Data Gear である。この Input と Output の関係が依存関係となる。TaskManager は Persistent Data Tree を監視しており、WaitTaskQueue に入っている Task の Input Data Gear が揃っているのを確認したら実行可能な Task として AcitiveTaskQueue へ移動させる。

3.2　Worker

Worker は TaskQueue から Task を取得し、実行する。Task には実行する Code Gear と実行に必要な Code Gear の key が格納されている。実行に必要な Code Gear は Persistent Data Tree から key を使って取得する。

各 Worker は個別の Context を参照しており、 メモリ空間も独立しているのでメモリを確保する処理で他の Thread を止めることはない。ただし、Persistent Data Tree への書き出しは競合する可能性があるので CAS を利用してデータの一貫性を保証する必要がある。

Worker が TaskQueue から Task の取得を行う Code Gear を Code **??** に示す。Task Queue から取得した Task から実行する Code Gear と必要な Data Gear の key を Worker Context に書き込むことで実行される。

Worker から取得された Task の Code Gear は並列実行される。並列実行される Code Gear と言っても他の Code Gear と同じである。これは Gears OS 自体が Code Gear によって構成されていることに起因する。つまり、 Gears OS を利用して書かれたプログラムで定義されている Code Gear に依存関係がないとき、全て並列に実行することができる。

第4章　GPGPU

4.1　GPGPUとは

GPGPU とは、元々は画像出力や画像編集などの画像処理に用いられる GPU を画像処理以外に利用する技術の事である。
画像の編集はピクセル毎に行われるため多大な数の処理を行う必要があるが、 GPU は CPU に比べコア数が多数あり、多数のコアで同時に計算することによって CPU よりも多数の並列な処理を行う事が出来る。
これによって GPU は画像処理のような多大な処理を並列処理することで、 CPU で処理するよりも高速に並列処理することが出来る。しかし、GPU のコアは CPU のコアに比べ複雑な計算は出来ない構造であるため単純計算しか出来ない、また一般的にユーザーから GPU 単体に直接命令を書き込むことも出来ないなどの問題点も存在する。GPGPU は CPU によって単純計算の Task を GPU に振り分ける事によって、 GPU の問題点を解決しつつ、高速な並列処理を行うことである。また Data Gear へのアクセスは接続された Code Gear からのみであるから、処理中に変数が書き変わる事がない。図 4.1 では以下の流れで処理が行われる。

- Data Gear を Persistent Data Tree に挿入。
- TasMannager で実行する Code Gear と実行に必要な Data Gear への Key を持つ Tas k を生成。
- 生成した Task を TaskQueue に挿入。
- Worker の起動。
- Worker が TskQueue から Task を取得。
- 取得した Task を元に必要な Data Gear を Persistent Data Tree から取得。
- 並列処理される Code Gear を実行。

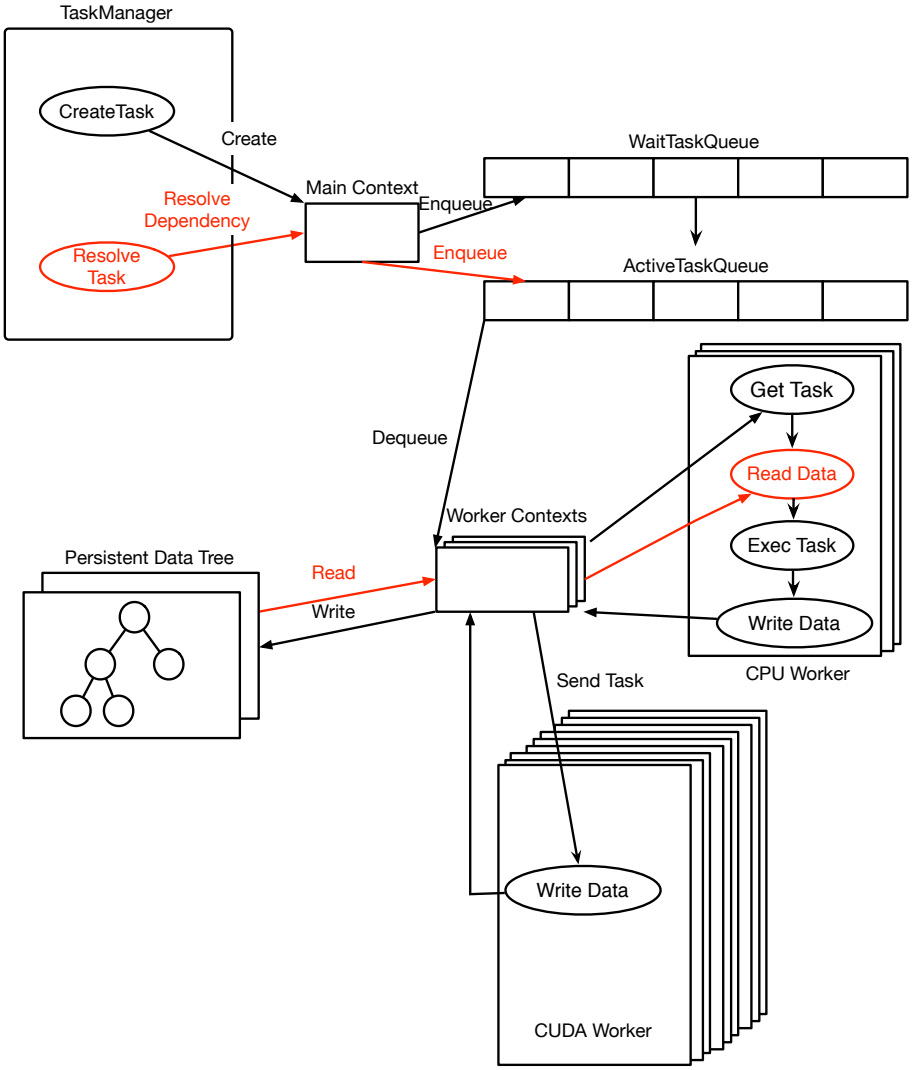


図 4.1: Gears OS による GPGPU

4.2 CUDA とは

CUDA とは NVIDIA 社が提供している並列コンピューティング用の統合開発環境で、並列プログラムの記述や、コンパイラ、ライブラリなど、また GPU といった並列コンピューティングを行うのに必要なサポートを提供しており、一般的にも広く使われている GPU の開発環境です。

第5章　GPU 実装

5.1　GPUWoker の実装

5.2　CUDATwiceの実装

5.3　CMake

5.4　性能評価

5.5　比較

第6章　今後の課題

6.1　GPU 並列実行

第7章　結論

7.1　まとめ

7.2　今後の課題

謝辞

2017年 3月