# 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

渡真利 勇飛 (並列信頼研究室)

## 研究概要

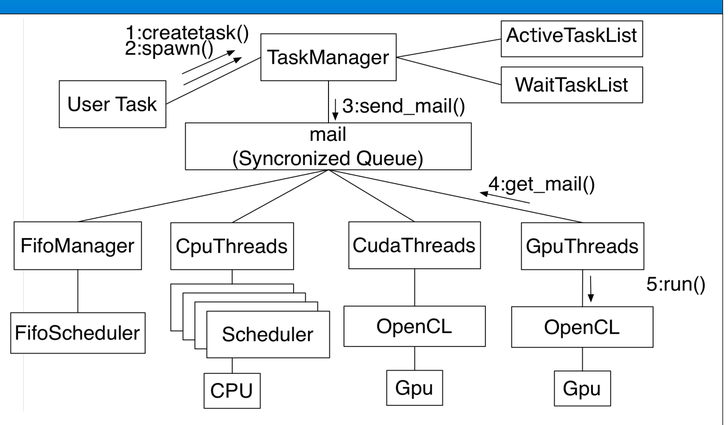
・消費電力や発熱、クロックの限界といった問題からCPUの性能を上げることによる処理性能の向上は難しい
・マルチコアCPUやGPUを含んだヘテロジニアス構成が主流
・そういったマルチプラットフォームに対応したフレームワークが必要
・並列プログラミングフレームワークCeriumを開発
・プラットフォームに最適な形でプログラムを並列に動作させる
・Ceriumの並列実行機構の測定・評価を行った

## 並列プログラミングフレームワークCerium

・CeriumはLinux、MacOSX上で動作する汎用計算用のフレームワーク
・CeriumはマルチコアCPUとGPUにおける並列プログラミングを可能にする

## TaskManager

・Ceriumはユーザーが定義したTaskを受け取り、管理する機構TaskManagerを持つ
・TaskManagerとThreadsの間にはSynchronisedなMailQueueがある




・依存関係の解決したTaskはTaskManagerからMailQueueに送られる
・ThreadsがMailQueueからTaskを取得し、並列実行していく

## パイプラインによる並列実行


```
do{
        task1->read();
        task2->exec();
        task3->write();
        delete task3;
        task3 = task2;
        task2 = task1;
        task1->next();
} while(task1);
```

・マルチコアCPU上での並列実行はSynchronizedQueueとパイプラインにより構成されている
・TaskManagerで依存関係を解決されたTaskはSchedulerに送信され、パイプラインに沿って並列実行される

## GPGPUへの対応

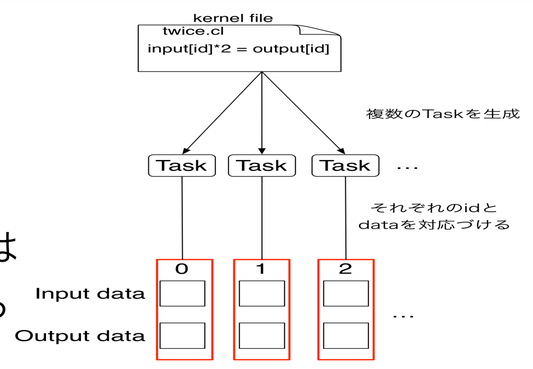
・OpenCL、CUDAを用いてCeriumをGPGPUへ対応
・TaskManagerから受け取ったTaskやデータをOpenCL、CUDAのAPIを介してGPUに転送する機構を実装

## GPGPUへの対応

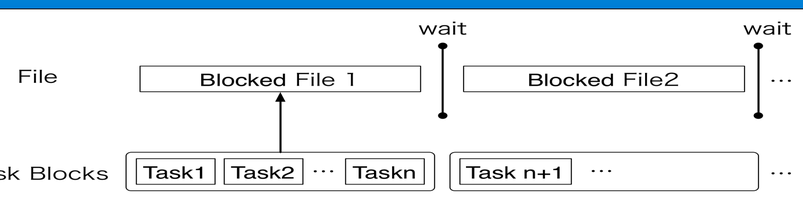
・OpenCL、CUDAを用いてCeriumをGPGPUへ対応
・TaskManagerから受け取ったTaskやデータをOpenCL、CUDAのAPIを介してGPUに転送する機構を実装

## データ並列実行

・データ並列用のAPI、iterateを実装
・1つの記述から復数のTaskを生成する
・生成した復数のTaskにIDとデータを割り当てる
・Taskの持つIDと割り当てられるデータは対応があり、IDから担当範囲を算出できる




## 並列処理向けI/O BlockedRead

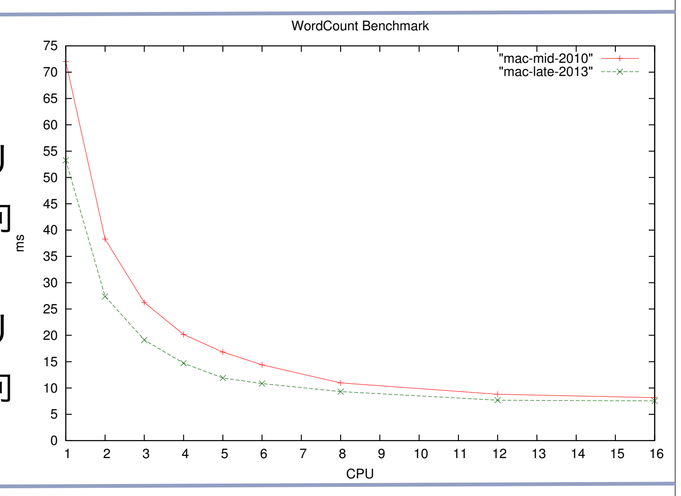


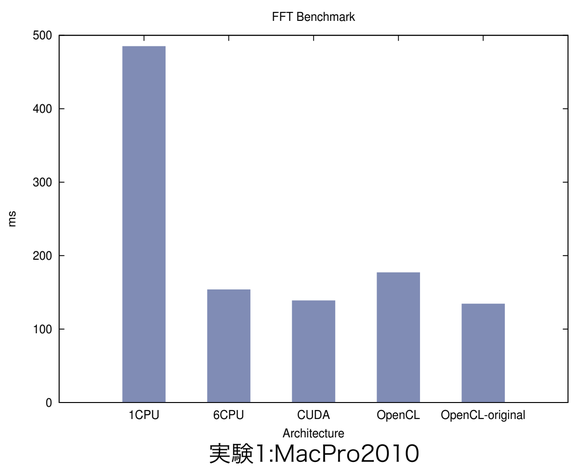

・I/Oを含むプログラムは読み込み部分がネックになることが多い
・並列I/O BlockedReadを実装
・ファイルを分割して読み込み、読み込んだ部分にTaskを割り当てる
・読み込み専用のThreadを実装し、優先度を設定

## Ceriumの性能評価

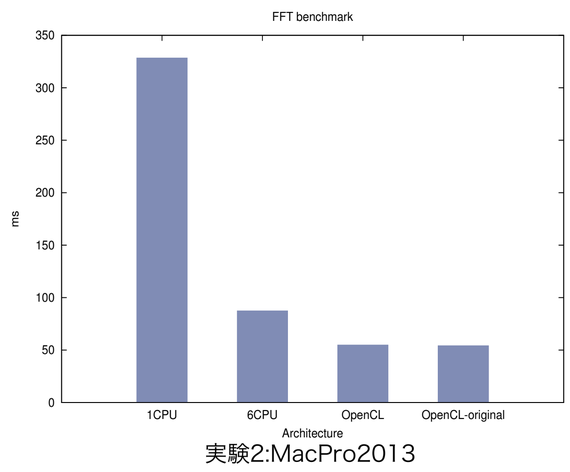
・WordCount、Sort、FFTの例題を用いて性能評価を行った
・評価対象はマルチコアCPU、GPGPU(CUDA,OpenCL)、BlockedRead

・WordCountによるマルチコアCPUのベンチマーク
・MacPro2010において、6CPUの場合1CPUと比べ5.0倍の性能向上が見られた
・MacPro2013において、6CPUの場合1CPUと比べ5.2倍の性能向上が見られた




実験1:MacPro2010

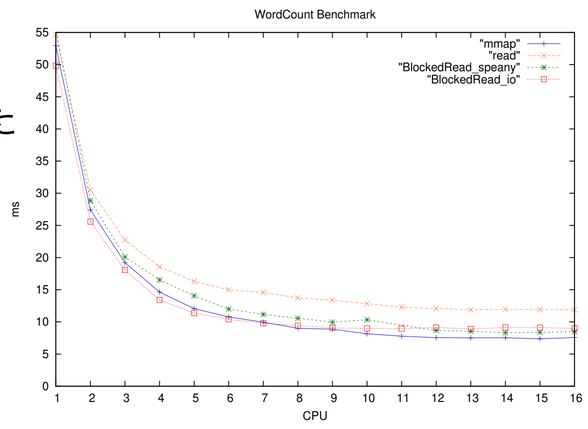



実験2:MacPro2013

・FFTによるCUDA、OpenCLによるGPGPUのベンチマーク
・【実験1】CUDAは6CPUと比べて1.1倍の性能向上
・【実験1】しかしOpenCLは6CPUと比べ0.76倍の性能低下
・【実験1】OpenCLのみのFFTと比べても0.76倍の性能低下
・【実験2】より高性能な計算機で測定
・【実験2】OpenCLは6CPUと比べ1.6倍の性能向上
・【実験2】OpenCLのみの場合と比べても同等の性能

GPUの性能を活用できている事がわかる。マルチコアCPUの処理性能も上がっていることから、GPUよりも上のレイヤでの最適化(パイプライニング)が有効に働いていると考えられる。

・WordCountによるmmap,read, BlockedReadのベンチマーク
・BlockedReadはIO Threadを用いた場合(io)と用いていない場合(speany)がある
・6CPUにおいてBlockedRead_ioはmmapに比べて1.1倍、readに比べて1.58倍、BlockedRead_speanyに比べて1.34倍の性能向上




## 今後の課題

・依存関係の記述方式にDataDependencyを追加
・GpuSchedulerのパイプラインの改良

確認することリスト
・載せるベンチマークはこれでいいか
・ベンチマークはラインで区切るのではなく別の項目にわけるべきか
・今後の課題、ポスターではこれくらいにして口頭で説明すべきか