CUDAを触ろう

GitHub: SuperHotDogCat

注意点

・これは大学の研究室発表用の資料なので簡潔な説明にとどめています。

・わかりにくいところは後ろに参考文献をつけておいたので参考にしてください

CUDAってなんだい？

・NVIDIAのGPUを動かすためのプログラミング言語

・C/C++と互換があり，近い書き方ができる

GPUってなんだい?

Graphics Processing Unitの略で, 本来はCG処理などのグラフィック処理のためのもの

画像処理を高速にリアルタイムに行うためのハードウェアのために, 並列計算性能が高かった。

最近はGPGPU(General Purpose computing on GPU)の分野でこの演算資源を科学技術計算に使ったりする。

今日のトピック

・CUDAでプログラミングしてみよう

・行列積を高速化しよう

・ソートアルゴリズムを高速化しよう

用語整理

Host: CPUのこと

Device: GPUのこと

Kernel: Hostで呼ばれてDevice上で実行される関数

CUDAで
プログラミングしてみよう

GitHub: SuperHotDogCat

関数識別子

CUDAでの関数はCPUで呼び出される関数とGPUで呼び出される関数を区別して書かなければならない。

__global__: Hostで呼び出されてDevice上で実行される関数, 必ずvoid型

__device__: Deviceで呼び出されてDevice上で実行される関数, Hostからは呼び出せない。呼び出す時は__global__ -> __device__の順番

__host__: Hostで呼び出されてHost上で実行される関数

global識別子には制約あり
https://http.download.nvidia.com/developer/cuda/jp/CUDA_Programming_Basics_PartII_jp.pdf

__global__関数の呼び出し方

__global__ kernel(something);

kernel<<<grid, block>>>(something);

という呼び出し方をする。この時, grid×block個のスレッドが同時に立ち上がる

Handson

https://github.com/SuperHotDogCat/cuda-introduction/tree/cudasource のtutorial directoryにあるコードを実行してみましょう。

cudaSetDevice(int device): 実行するDeviceの番号を指定する

cudaMalloc(void **devPtr, size_t size): Device上にsize byte分だけメモリを確保する

cudaMemcpy(void *dst, const void *src, size_t size, cudaMemcpyKind kind): dstにsrcの内容をsize byte分だけメモリを確保する。kindはHostからDeviceにデータを送るかなどを指定するenum型

cudaFree(void *devPtr): *devPtrのメモリ領域を解放する

cudaDeviceSynchronize(): Deviceで実行される関数は非同期実行のためこれを実行するとDeviceでの実行が完了するまで待ってくれる

Compute Capability

GPUの機能を使うために, GPUのarchitectureに応じてCompute Capabilityという数値があり, 指定することで対応するGPUに応じたcompileがなされる。

https://developer.nvidia.com/cuda-gpus

makefileには発表者のGeForce RTX 4060 laptopのCompute Capabilityが反映されているので実行環境に応じて変更すること

行列積を高速化しよう

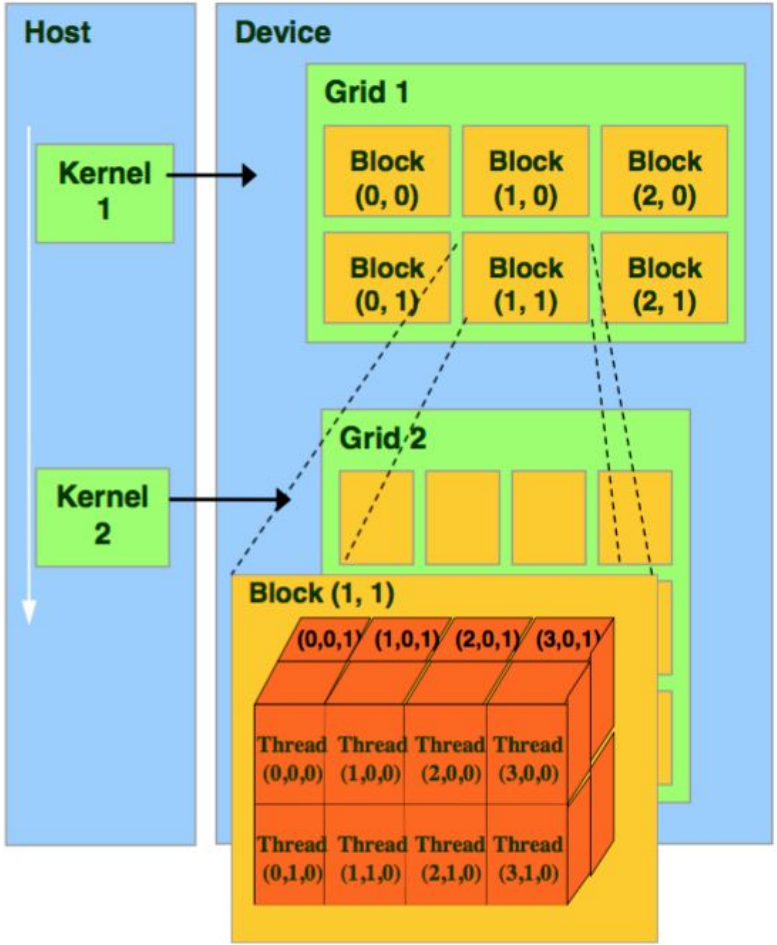
GitHub: SuperHotDogCat

CUDAの並列実行モデル

CUDAのDevice内ではスレッドが右のように生成される。

CUDAのスレッドはGrid, Block, Threadという3つの単位で構成される。

Gridの中にBlockがあり, Blockの中にThreadがあるという構成である。



grid, blockの指定

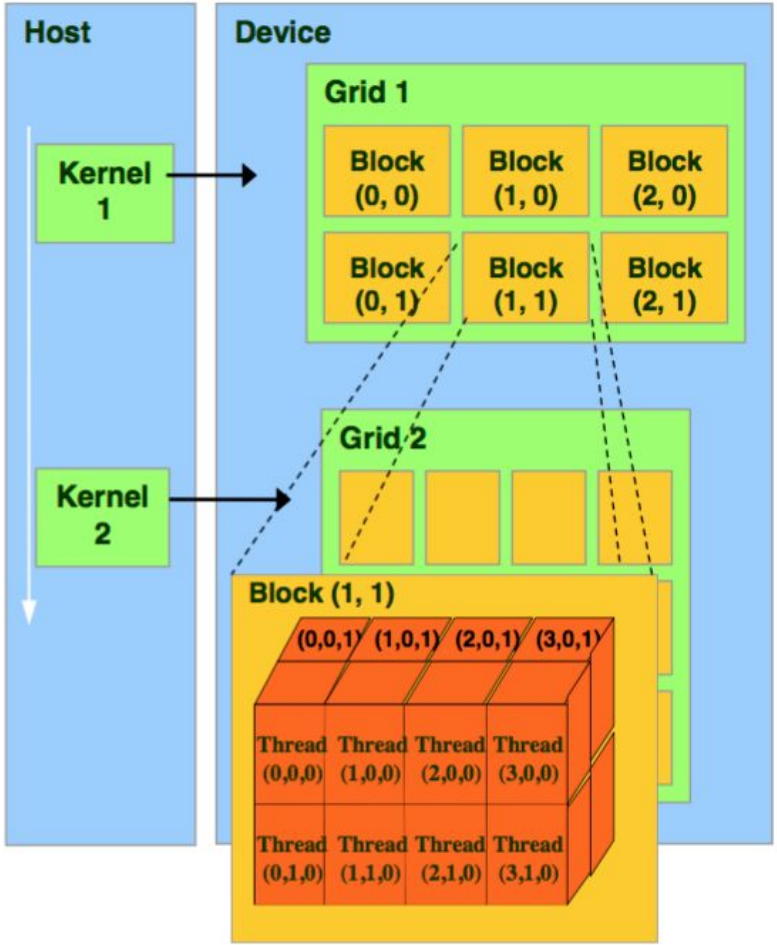
__global__関数を実行するときに,

kernel<<<grid, block>>>と指定する本が多い。ここが少し紛らわしいが,

最初の変数(grid)は右図のgridの中のblockの個数

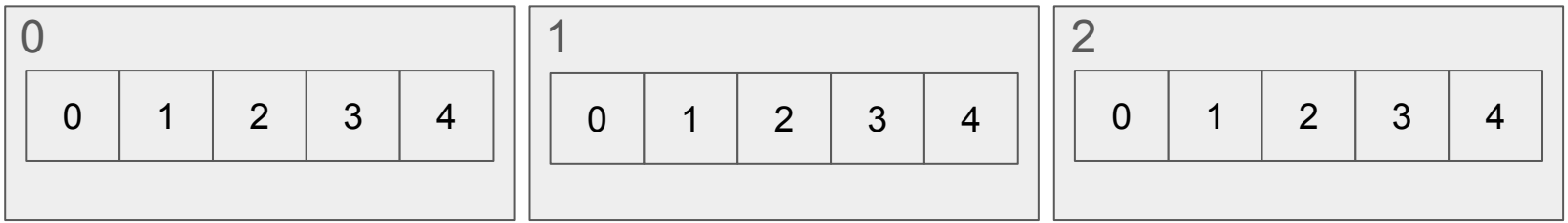
二つ目の変数(block)は右図のblockの中のthreadの個数を指定している

まじでこの言葉の使い分けが紛らわしいのでここは強調します。



grid, blockの指定の図

grid(3), block(5)の場合

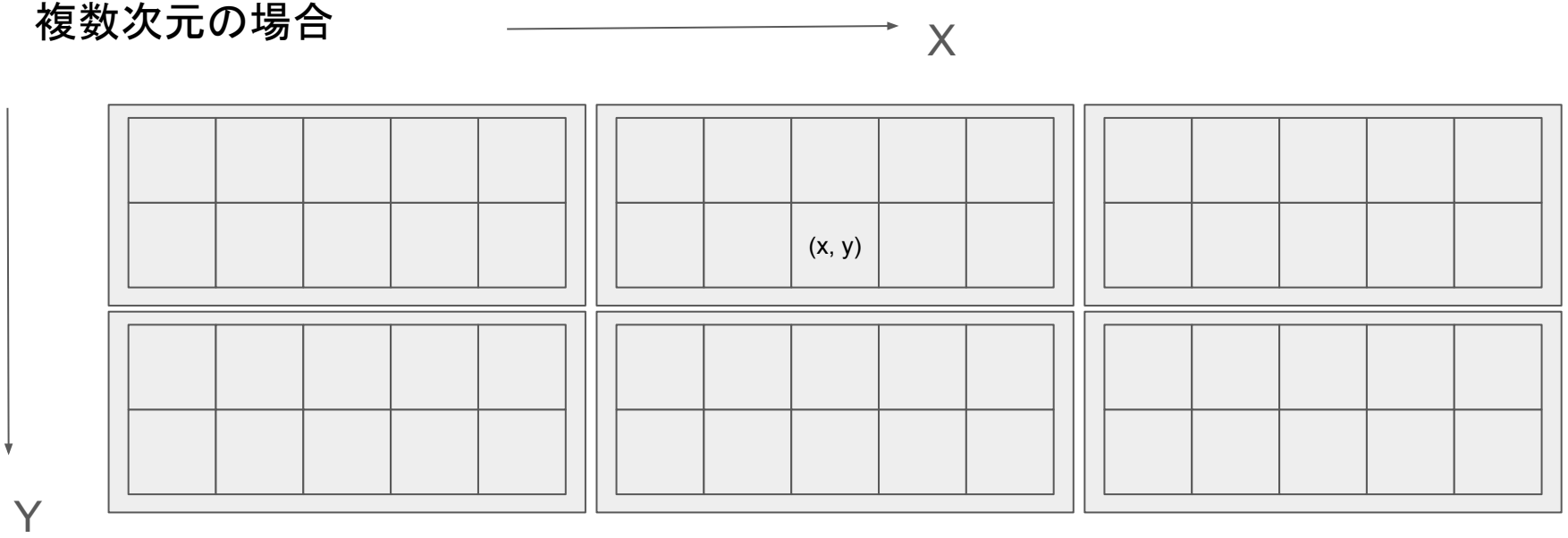


x番目は何番目のBlockの何番目のThreadかを指定する。

x=threadIdx.x+blockDim.x*blockIdx.x;

grid, blockの指定の図

複数次元の場合



x=threadIdx.x+blockDim.x*blockIdx.x;
y=threadIdx.y+blockDim.y*blockIdx.y;

grid, blockの指定

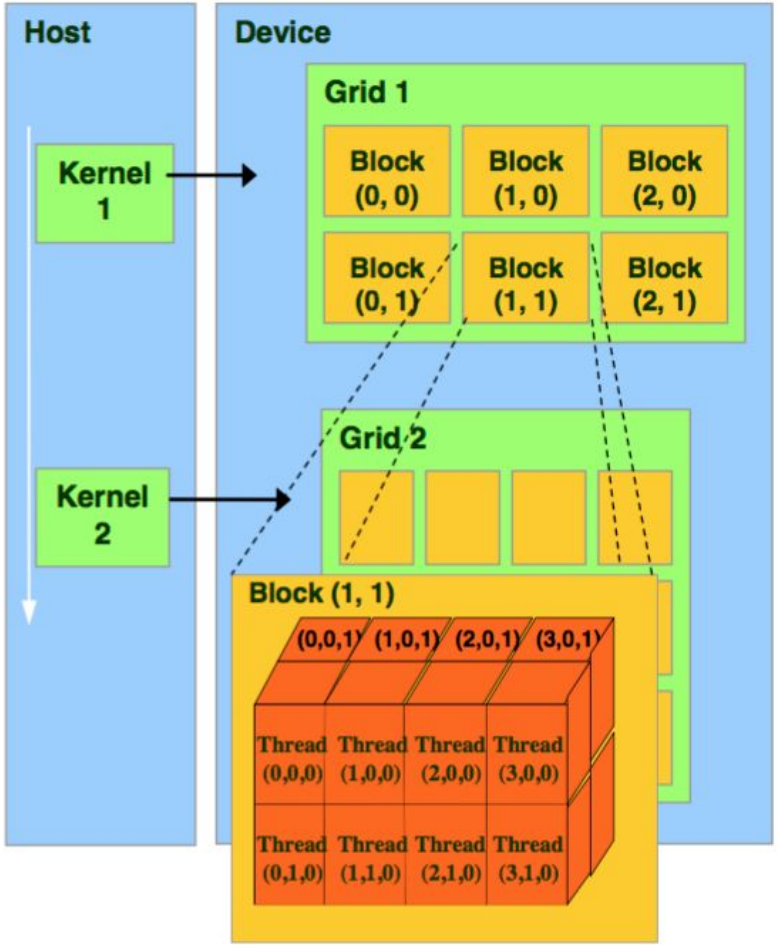
__global__関数を実行するときに,

kernel<<<grid, block>>>と指定すると

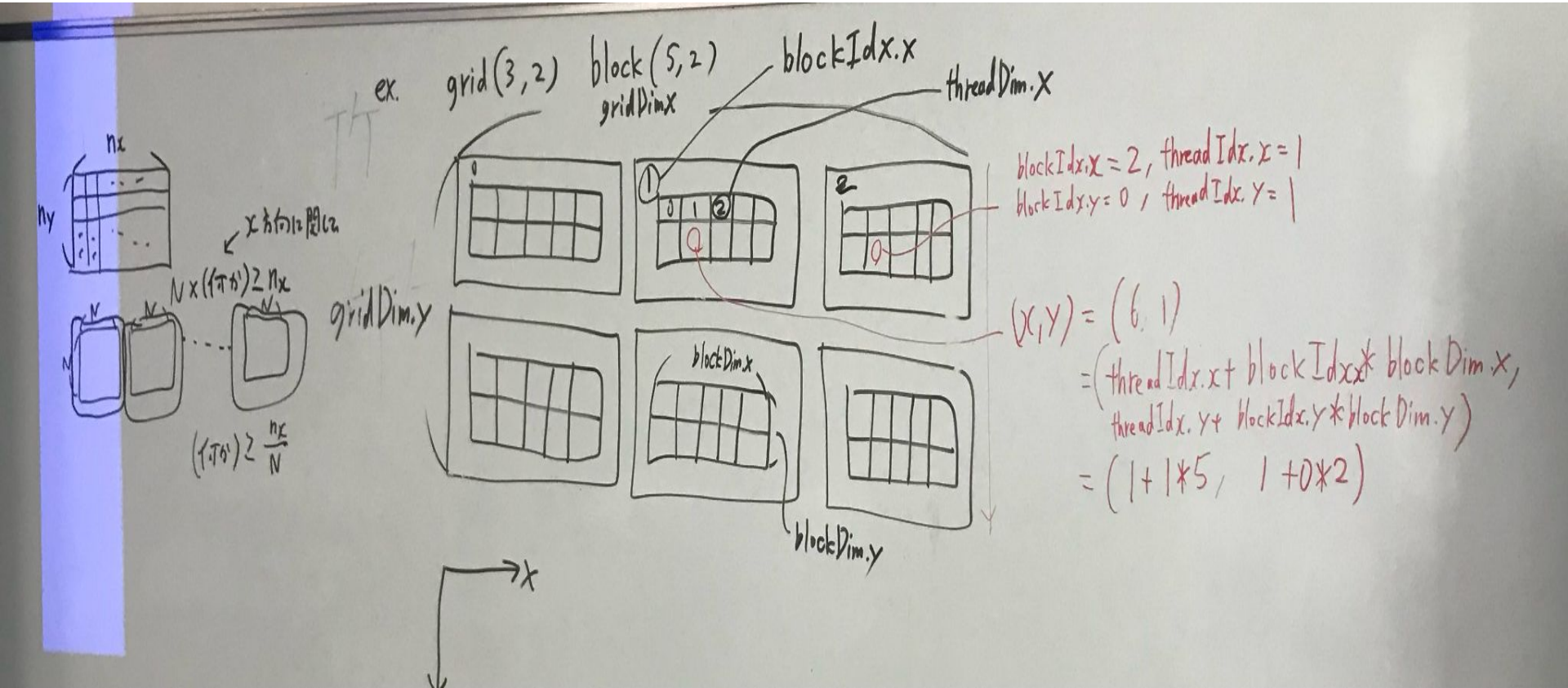
grid × block個のthreadが同時に実行される。

並列化可能なところを見極めることで高速化が可能に

ただし, gridとblockの組み合わせで計算時間は大きく変わる。最良の結果が得たいのなら考えて設定すること



発表当日の素晴らしいスライド



Handson

https://github.com/SuperHotDogCat/cuda-introduction/tree/cudasource のcudamodel directoryにあるコードを実行してみましょう。

高速化 Template

```
for (int i = 0; i < nx; ++i){
    for (int j = 0; j < ny; ++j){
        // iとjを用いた処理
    }
}
```

CUDA化

```
int i=threadIdx.y+blockDim.y*blockIdx.y;

int j=threadIdx.x+blockDim.x*blockIdx.x;

// iとjを用いた処理
```

呼び出す時は行列のサイズをDとして
int num_threads = N
block(num_threads, num_threads)
grid((nx+block.x-1)/block.x,
(ny+block.y-1)/block.y);
として指定

Handson

https://github.com/SuperHotDogCat/cuda-introduction/tree/cudasource のmatmul directoryにあるコードを実行してみましょう。

cpuコードとも時間を計測して比較してみましょう

ソートアルゴリズムを高速化しよう

GitHub: SuperHotDogCat

ソートアルゴリズム

比較を元にしたソートアルゴリズムでは O(NlogN)が最小であることが知られており

計数ソートでもO(N)のアルゴリズムが知られている。これらのアルゴリズムは CPU環境で最速である。

GPU環境ではどうなるのかを見ていく

並列可能なソートアルゴリズム

・調べた限りではRadix SortとBitonic Sortがある。

・今回のWork ShopではBitonic Sortの方を扱う(こっちの方がアルゴリズムが複雑なので)

・宿題はRadix Sortの実装です(要素はint型を仮定してください)

並列可能なソートアルゴリズム

Bitonic Sort: 計算量O(N(logN)^2)

要素数がN = 2^nの時

メインブロック数: logN = n

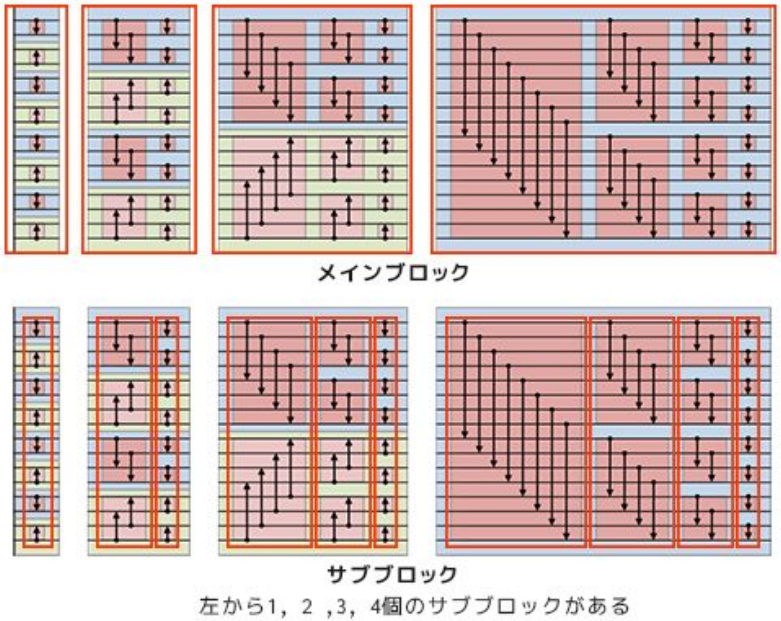
サブブロック数: m番目のメインブロック内ではm個

あとは矢印の向きとかの決め方とか細かいのは省略

並列化する場所はサブブロックの赤い四角のところです。(データ依存がない)

メインブロックは並列化できません。

並列化する場所について ->
https://edom18.hateblo.jp/entry/2020/09/21/150416



Handson

https://github.com/SuperHotDogCat/cuda-introduction/tree/cudasource のsort directoryにあるコードを実行してみましょう。

cpuコードとも時間を計測して比較してみましょう

宿題

1. Matmulを実装してきましょう(高速化もできればしましょう)
2. (Advanced) Radix Sortを実装してきましょう。
3. threadの数を色々変えてmatmulやsortのプログラムの実行時間を計測しましょう。
4. 配列のサイズNを大きくしてCPUとのmatmulやsortの実行時間の比較をしてみましょう。
5. (Advanced) OpenMPでCPUの方も並列化してGPUが勝てるようにしましょう。

Warp

内部ではスレッドが32個ずつまとめられたWarpと呼ばれるグループにまとめられて同一命令が実行される。したがって,以下のようなプログラムは性能低下を引き起こす可能性がある。http://www.butsuri.it-chiba.ac.jp/~yasutake/matter/tominaga.pdf

```
if (条件1){
        hoge();
} else if (条件2){
        fuga();
} (最近はスレッドごとにプログラムカウンタがあるので大丈夫かも?)
```

Warp

必ずしもif文を含むプログラムがwarpによる性能低下を引き起こすわけではない

分岐の粒度をwarp sizeの倍数にすることでワープダイバージェンスを避けるようにすることができる。

http://www.sim.gsic.titech.ac.jp/Japanese/GPU/pdf/20131226717065-474-0-30.pdf

上記に加えて, Threadはやはり32の倍数で設定した方が性能が良い。

他にも

・ワープの特性を考えた高速化

・メモリアクセス, (Array of StructuresかStructures of Arrayかでメモリレイアウトが異なるなど)https://proc-cpuinfo.fixstars.com/2017/07/acceleration-of-reduce-banding-filter-by-cuda-02/

etc….

・shared memory https://yusuke-ujitoko.hatenablog.com/entry/2016/02/05/012618

Reference

CUDA C++ Programming Guide

Technical Training

CUDA Programming Basics I

CUDA Programming Basics II

CUDA Programming Guide

GPUプログラミング・基礎編 東京科学大(旧 東工大)

CUDAの導入，CUDAの基礎 茨城大学

CUDA Primer

Reference

CUDAを用いた数値解析の高速化

thread の実行方式とWarp

CUDAによるバンディング低減フィルタの高速化（2）

【CUDA】Shared memoryの動的な確保