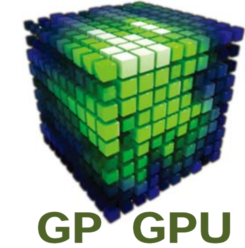


GPUコンピューティング No.4

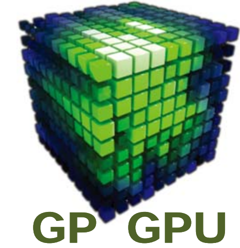
CUDA プログラミング基礎

― メモリバンド幅の測定 ―

東京工業大学　学術国際情報センター

青木　尊之

DATA転送テスト



例題

CPU側の配列 a_h[n] の内容を GPU の global memory の a_d[n] に転送し、次に GPU カーネルで a_d[n] → b_d[n] のコピーを行い、最後に b_d[n] から CPU側の配列 b_h[n] に転送する。

a_h → a_d → b_d → b_h
(CPU) (GPU) (GPU) (CPU)

% tar xvf copy_test.tar

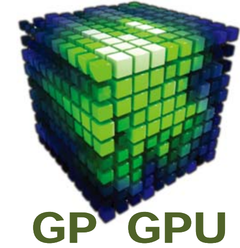
カレント・ディレクトリに copy_test というディレクトリが生成

% cd copy_test
% ls　　（ディレクトリの中身を見る）

Source Code #01

Copyright © Takayuki Aoki , Global Scientific Information and Computing Center, Tokyo Institute of Technology

Source Code #01（メモリ確保と転送）



#define NUM (1024*1024*1) 1～16 などに変更してみる

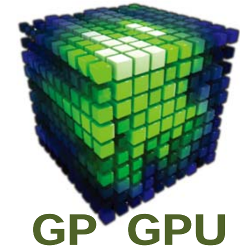
```
int n = NUM;

double *a_h, *b_h, *a_d, *b_d;

a_h = (double *) malloc(n*sizeof(double)); for(i = 0; i < n; i++) a_h[i] = 9.3;
b_h = (double *) malloc(n*sizeof(double)); for(i = 0; i < n; i++) b_h[i] = 0.0;

cudaMalloc( (void**) &a_d, n*sizeof(double) );
cudaMalloc( (void**) &b_d, n*sizeof(double) );

cudaMemcpy( Ad, A, n*sizeof(double), cudaMemcpyHostToDevice );
cudaMemcpy( Bd, B, n*sizeof(double), cudaMemcpyHostToDevice );
```

host側にメモリ確保と代入

device側にメモリ確保

hostからdeviceにデータ転送

global → global memory copy のkernel関数



dim3 grid(n/256), block(256); ← block size を 256 としている。

g2g_copy<<< grid, block>>>(a_d, b_d); ← kernel 関数の呼び出し実行

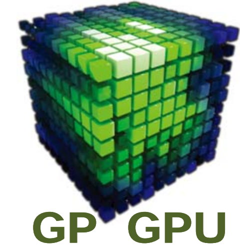
または

g2g_copy<<< n/256, 256>>>(a_d, b_d);

```
__global__ void g2g_copy
// ==========================================
(
   double   *A,      // array pointer of the global memory
   double   *B       // array pointer of the global memory
)
// ----------------------------------------------------------------------
{
    int   i = blockDim.x*blockIdx.x + threadIdx.x;


    B[i] = A[i];
}
```

Copyright © Takayuki Aoki , Global Scientific Information and Computing Center, Tokyo Institute of Technology

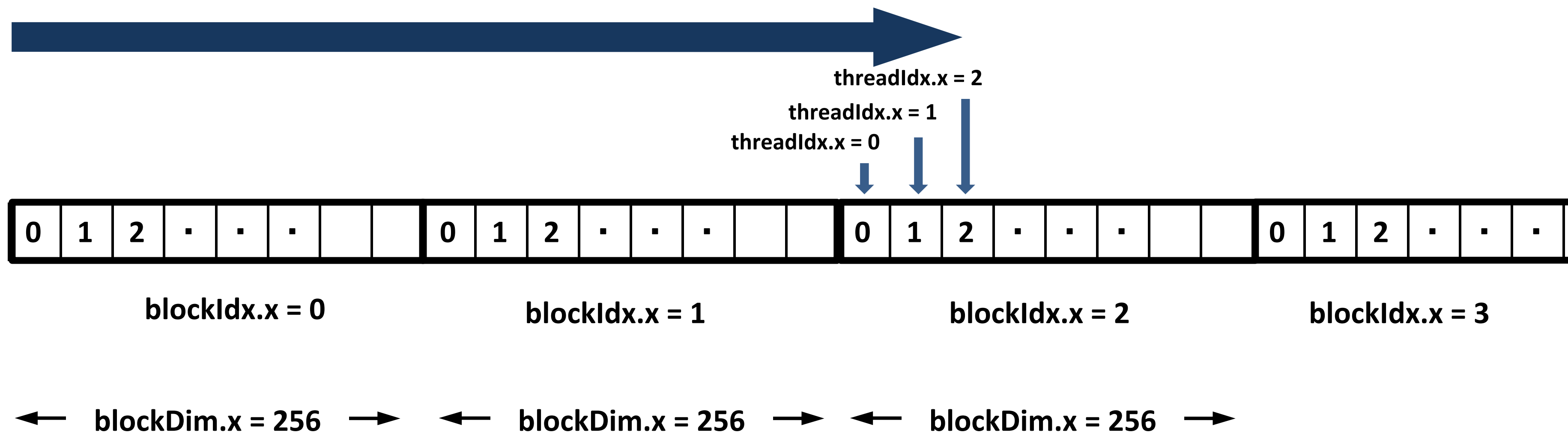
並列データ·アクセス



■ 1次元配列データへのthreadからのアクセス

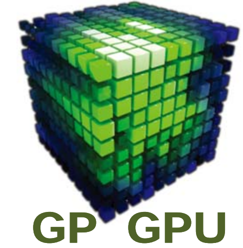
■ N個のthread を発生させ、1 thread が配列の1要素にアクセス

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



Copyright © Takayuki Aoki , Global Scientific Information and Computing Center, Tokyo Institute of Technology

Built-in 変数

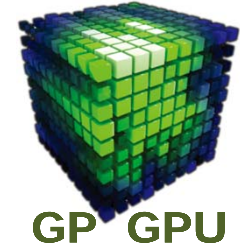


Device code の中で宣言せずに引用でき、
書き換え不可

gridDim	**gridDim.x, gridDim.y, gridDim.z** **grid の各方向のサイズ**
blockIdx	**blockIdx.x, blockIdx.y, blockIdx.z** **block の各方向のindex**
blockDim	**blockDim.x, blockDim.y, blockDim.z** **block の各方向のサイズ**
threadIdx	**threadIdx.x, threadIdx.y, threadIdx.z** **thread の各方向のindex**

Copyright © Takayuki Aoki , Global Scientific Information and Computing Center, Tokyo Institute of Technology

C言語の拡張

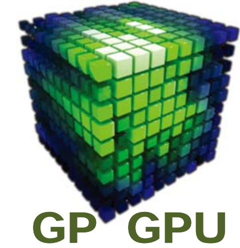


関数型の Qualifier

__global__	device 上でのみ実行される host 側からのみ call される
__device__	device 上でのみ実行される device からのみ call される
__host__	host 上でのみ実行される host 側からのみ call される （普通の CPU 上のプログラムの関数で、特に宣言する必要はない。）

Copyright © Takayuki Aoki , Global Scientific Information and Computing Center, Tokyo Institute of Technology

計算結果の回収



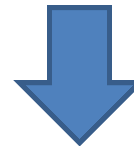

```
cudaMemcpy(b_h, h_d, n*sizeof(double), cudaMemcpyDeviceToHost );

double  sum = 0.0;
for(i = 0; i < n; i++) sum += b_h[i];

printf("Value  = %8.6f¥n",sum/(double)n);
```

DeviceからHostにデータ転送

計算結果の確認

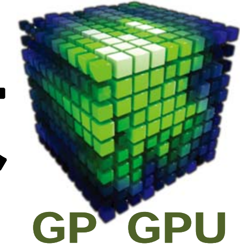
CPUでの実行：

```
g2g_copy<<< grid, block>>>(a_d, b_d);
```



```
for(int  j = 0; j < N; j++)  b_h[j] = a_h[j];
```

配列データの加算のkernel関数



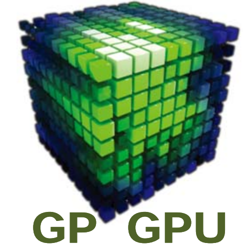

```
dim3  grid(N/256),  block(256);
```
← block size を 256 としている。

```
add<<< grid, block>>>(a_d, b_d, c_d);
```
← kernel 関数の呼び出し実行

```
__global__  void  add
// ===========================================
(
    double    *A,       // array pointer of the global memory
    double    *B,       // array pointer of the global memory
    double    *C        // array pointer of the global memory
)
// -----------------------------------------------------------------------
{
    int    i = blockDim.x*blockIdx.x + threadIdx.x;

    C[i] = A[i] + B[i];
}
```

Copyright © Takayuki Aoki , Global Scientific Information and Computing Center, Tokyo Institute of Technology

例：算数の練習問題



A[i]		B[i]			A[i]		B[i]			A[i]		B[i]	
14	+	34	=		10	+	23	=		29	-	31	=
8	+	50	=		12	+	55	=		- 2	-	20	=
22	+	92	=		42	-	23	=		51	-	44	=
33	-	23	=		13	-	7	=		- 39	+	28	=
- 5	+	40	=		31	+	12	=		- 27	+	10	=
- 1	+	70	=		15	+	77	=		- 9	-	62	=
17	+	3	=		3	+	12	=		30	-	38	=
100	-	40	=		- 7	+	80	=		78	+	22	=
6	+	25	=		- 23	-	2	=		- 54	+	70	=

Copyright © Takayuki Aoki , Global Scientific Information and Computing Center, Tokyo Institute of Technology

算数の練習問題(1)

生徒48人のクラスで算数ドリルを使って計算練習を行う。48題の問題の中から、生徒がそれぞれ違う問題を解いて、先生に提出することにする。

先生が行うこと:

- 事前に算数ドリルを生徒に配布。
- クラスの生徒を班に分ける。
- 問題の解き方の解説プリントの配布。
- 採点と平均点の算出。

Copyright © Takayuki Aoki / Global Scientific Information and Computing Center, Tokyo Institute of Technology

算数の練習問題（2）

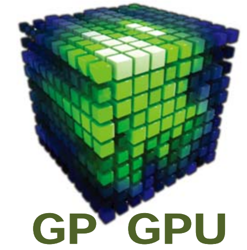
生徒が行うこと:

・自分はどの問題をやればよいか指示を聞く。
・問題を解いて計算する。
・何ページの問題を解いたかと、その答えを解答用紙に記入し、名前を書いて提出。

教師はクラス全員に同じ指示しか出せないとする。

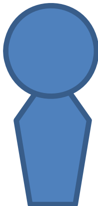
Copyright © Takayuki Aoki / Global Scientific Information and Computing Center, Tokyo Institute of Technology

クラスの班分け

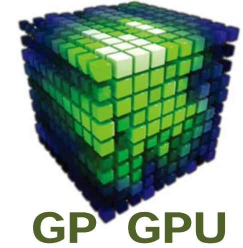


Copyright © Takayuki Aoki / Global Scientific Information and Computing Center, Tokyo Institute of Technology

クラスの班分け



アナロジー



host code: 先生の立場になって、問題を多くの生徒にならせる指令を出す。
班分けを決める。

device code: 個々の生徒の立場になって何を実行するかを記述する。

クラスに何班あるか、自分は何班に所属しているか、班にはメンバーが何人いるか、自分はその班の何番目か。

Copyright © Takayuki Aoki / Global Scientific Information and Computing Center, Tokyo Institute of Technology

所要時間計測（1）

経過時間を計測することで、GPU Computing のパフォーマンスを確認でき、ハードウェア実行のようすを想像することができる。また、チューニングのためには必須。

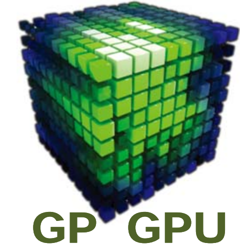
```
cudaEvent_t start, stop;
cudaEventCreate(&start);
cudaEventCreate(&stop);
float elapsedTime;

cudaEventRecord(start,0);


g2g_copy<<< grid, block>>>(a_d, b_d);


cudaEventRecord(stop,0);
cudaEventSynchronize(stop);
cudaEventElapsedTime(&elapsedTime,start,stop);
```

計測範囲

elapsedTime に
経過時間（msec)

Copyright © Takayuki Aoki , Global Scientific Information and Computing Center, Tokyo Institute of Technology

所要時間計測（2）



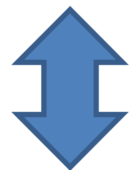
CUDA Utility を使った時間計測

CUDA_SDK_PATH = /usr/apps/free/NVIDIA_GPU_Computing_SDK/4.0/C

コンパイル・オプション　-I $CUDA_SDK_PATH /common/inc

リンク・オプション　–L $CUDA_SDK_PATH /lib -lcutil

```
#include <cutil.h>
unsigned  int   timer;

cutCreateTimer(&timer);

cudaThreadSynchronize();        （これ以前の非同期実行の終了を待つ）
cutStartTimer(timer);           （gettimeofday() を使った時刻測定開始）
```

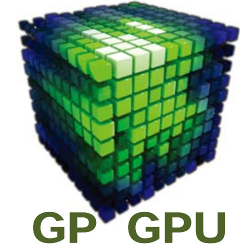


計測範囲

```
cudaThreadSynchronize();        （この間の非同期実行の終了を待つ）
cutStopTimer(timer);            （gettimeofday() を使った時刻測定終了）

double  elapsed_time = cutGetTimerValue(timer);     経過時間（msec)
```

Copyright © Takayuki Aoki , Global Scientific Information and Computing Center, Tokyo Institute of Technology

データ転送レートの測定



時間計測の方法を用いて
g2g_copy<<< grid, block>>>(a_d, b_d) の所要時間を測定する。

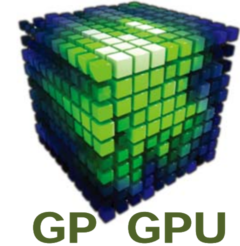
Exercise 1

転送したデータ量を元に、データ転送レート（global memory のメモリバンド幅）を測定する。

Exercise 2

Copyright © Takayuki Aoki , Global Scientific Information and Computing Center, Tokyo Institute of Technology

エラー処理(API)



#define N (1024*1024*17) 等で実行すると、結果が滅茶苦茶になる。

CUDA の API は全て return 値が cudaError_t 型の error の status を返すようになっている。

```
cudaError_t err = cudaMemcpy(…);
if (err != cudaSuccess) {
    fprintf(stderr, “Memcopy failed: %s.¥n”,
            cudaGetErrorString(err));
}
```

もし、cudaMalloc しないて、cudaMemcpy(); を実行してしまった場合などは、invalid device pointer が帰ってくる。

Copyright © Takayuki Aoki , Global Scientific Information and Computing Center, Tokyo Institute of Technology

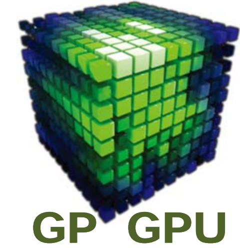
エラー処理（ kernel関数）

kernel 関数には return 値はない。 cudaGetLastError() で直後のエラーを拾い、cudaGetErrorString() でメッセージを表示させる。

```
vec_add<<< , , , >>>(. . . );
cudaError_t err = cudaGetLastError();
if (err != cudaSuccess) {
    fprintf(stderr, "kernel launch failed: %s¥n",
            cudaGetErrorString(err));
    exit(-1);
}
```

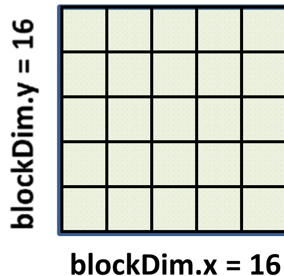
invalid configuration argument が表示される。
grid.x = 65536 となっていて、最大値 65535 を超えている。

2次元データ·アクセス



NX*NYの1次元配列データであるが、2次元的にアクセス

dim3 grid(NX/16, NY/16), block(16, 16);



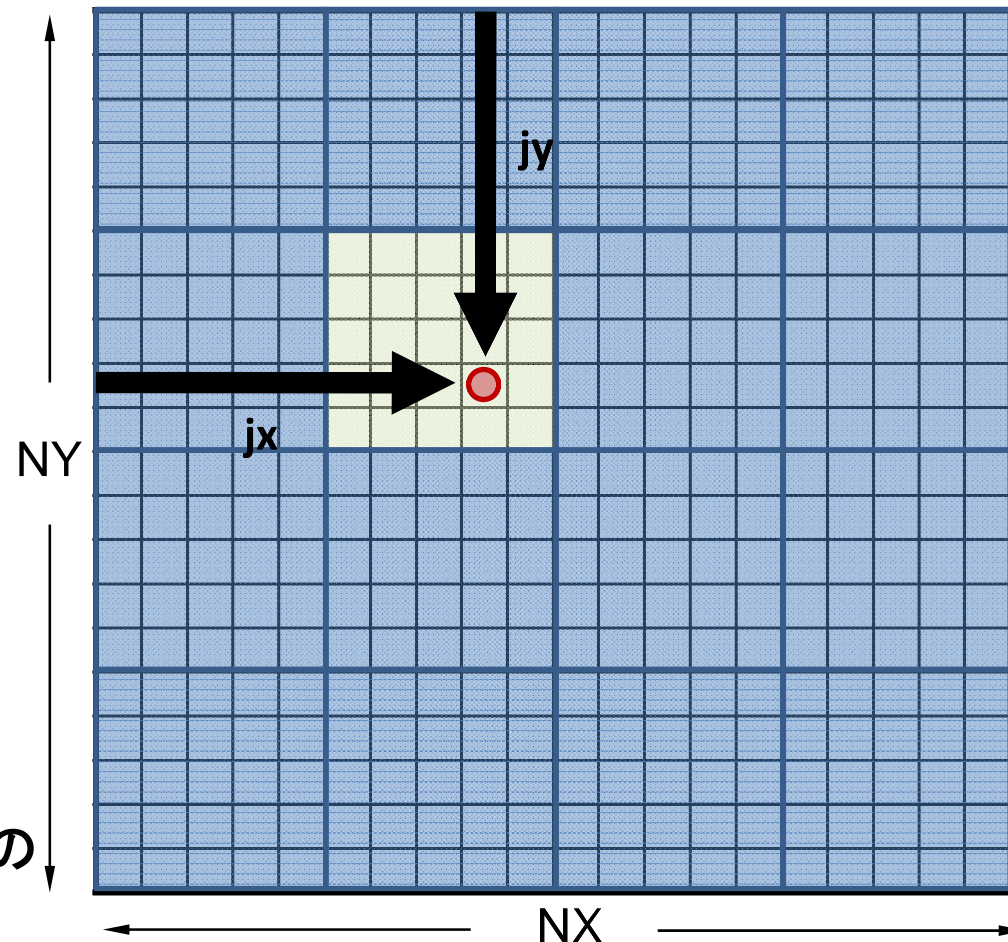
jx = . . .

jy = . . .

j = . . .

gridサイズの最大値の制限から開放

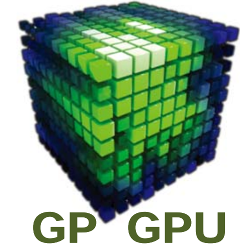
Exercise 3



#define N (1024*1024*20) のメモリバンド幅を測定する

Copyright © Takayuki Aoki , Global Scientific Information and Computing Center, Tokyo Institute of Technology

C[i] = A[i] + B[i] のFLOPS測定



時間計測の方法を用いて
add<<< grid, block>>>(a_d, b_d, c_d) の所要時間を測定し、
各thread の演算回数が 1 であることから、FLOPSを計算する。

```
__global__ void add
// ==========================================
(
   double    *A,      // array pointer of the global memory
   double    *B,      // array pointer of the global memory
   double    *C       // array pointer of the global memory
)
// ------------------------------------------------------------------------
{
    int    i = blockDim.x*blockIdx.x + threadIdx.x;

    C[i] = A[i] + B[i];
}
```

Exercise 4

Copyright © Takayuki Aoki , Global Scientific Information and Computing Center, Tokyo Institute of Technology

レポート課題2

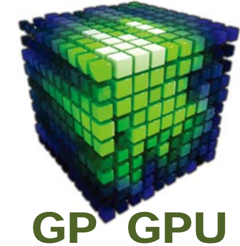
『CUDAプログラミング基礎』

Exercise 1　Exercise 2　Exercise 3　Exercise 4

をレポートにまとめ、提出すること。考察や試行、工夫を入れると点数が良くなる。

Copyright © Takayuki Aoki , Global Scientific Information and Computing Center, Tokyo Institute of Technology

レポート課題2

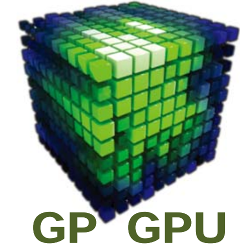


『CUDAプログラミング基礎』

期限：2013年 5月23日（木） 17:00
場所：学術国際情報センター・国際棟 1F のI7-3メールボックスに提出。または Subject:「CUDAプログラミング基礎」とし、メールの宛先
gpu_report2013@sim.gsic.titech.ac.jp
に上記の内容を pdf ファイルとして提出すること。

氏名、学籍番号、日付とタイトル (CUDAプログラミング基礎) は紙に印刷する場合でも pdf 提出でも書くこと。

Device マネージメントAPI



Device の情報を取得する API が準備されている。

cudaGetDeviceCount(int *count)
　　CUDAの動作するGPUの個数を返す。

cudaSetDevice(int device_no)
　　それ以降の実行を device_no の GPU に向ける。

cudaGetDevice(int *current_device)
　　現在指定されている GPU の device 番号を返す。

cudaGetDeviceProperties(int *device,
　　　　cudaDeviceProp *prop)
　　deviceQueryのような情報を prop のメンバーとして取得

Tips: これらは、同一ノード内に複数GPUがある場合は必須。

Copyright © Takayuki Aoki , Global Scientific Information and Computing Center, Tokyo Institute of Technology