GPUコンピューティング No.5

カーネル関数内での多次元配列

東京工業大学　学術国際情報センター

青木　尊之

カーネル関数内で多次元配列

GPUカーネル関数の中で、多次元配列を使いたい場合がある。

```
g2g_copy2d<<< Db,  Dt >>>(   );


__global__  void    g2g_copy2d( )
{
   B[blockIdx.x] [threadIdx.x] [threadIdx.y]
         = A[blockIdx.x] [threadIdx.x] [threadIdx.y] ;
}
```


線形メモリ

cudaMallocPitch(), cudaMemcpy2D() を使っても、GPUカーネル関数の中で簡単に多次元配列を使えない。プログラミングとしては薦められないが、以下のようにすれば可能。

```
__device__  double  A[NY][NX],  B[NY][NX];


__global__  void    g2g_copy2d( )
{
   B[blockIdx.x] [threadIdx.x] [threadIdx.y]
         = A[blockIdx.x] [threadIdx.x] [threadIdx.y] ;
}
```


多次元配列のデータ転送


```
__device__  double   a_d[NY][NX],   b_d[NY][NX];
static      double   a_h[NY][NX],   b_h[NY][NX];
```

host からdevice へのデータ転送

```
cudaMemcpyToSymbol(a_d, a_h, sizeof(double)*NX*NY, 0);
```

device から host へのデータ転送

```
cudaMemcpyFromSymbol(b_h, b_d, sizeof(double)*NX*NY, 0);
```

Source Code #02a

間接参照による2次元配列確保

CPUの場合には、動的メモリ確保で **a_h を2次元配列 a_h[NY][NX] のサイズで使えるようにする。

```
double  **a_h;

a_h = (double **) malloc(sizeof(double *)*NY);

for(j = 0; j < NY; j++) {
    a_h[j] = (double *) malloc(sizeof(double)*NX);
}
```


ホストでの2次元配列への代入

2次元配列　a_h[NY][NX] の要素に乱数を代入する。

```
for(j = 0; j < NY; j++) {
    for(i = 0; i < NX; i++) {
        a_h[j][i] = (double)rand() / RAND_MAX;
    }
}
```


GPUで2次元配列の確保と代入

GPU側で2次元配列　a_d[NY][NX] を確保するにはどうしたらよいか。

CPU側で値を代入した　a_h[NY][NX] を、GPU 上の配列 a_d[NY][NX] にデータコピーするにはどうしたらよいか。

カーネル関数の実行


```
g2g_copy2d<<< NY,  NX >>>(a_d, b_d);

__global__  void    g2g_copy2d(double **A, double **B)
{
   B[blockIdx.x] [threadIdx.x] = A[blockIdx.x] [threadIdx.x];
}
```

GPU側で値をコピーした　b_d[NY][NX] を、CPU 上の配列 b_h[NY][NX] にコピーするにはどうしたらよいか。

Source Code #03

レポート課題3

『CUDAの2次元配列』

期限：2013年 6月6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の2次元配列」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。

ソースコード sample03.tar.gz の copy_test2d.cu の空欄3箇所を埋め、計算結果が 0.5近傍になることを確認する。（今回はレポート中にソースコード添付すること。）ホスト側にポインター等を追加するのは自由。

GPUコンピューティング No.6

ZERO Copy の実行性能

東京工業大学　学術国際情報センター

青木　尊之

それぞれのDATA転送の測定

例題

CPU側の配列 a_h[n] のそれぞれの要素を GPU で一定数を加算すし、CPU側に戻す。

a_h → a_d
(CPU) (GPU)

```
cudaMemcpy( a_d, a_h, n*sizeof(double),
            cudaMemcpyHostToDevice );
```

a_d[i] += 1.3;
(GPU Kernel 関数で実行)

```
add<<grid, block>>( a_h, 0.111);
```

a_d → a_h
(GPU) (CPU)

```
cudaMemcpy( a_h, a_d, n*sizeof(double),
            cudaMemcpyDeviceToHost );
```

それぞれの転送レートを測定する。

Source Code #04

Page-locked Memory

（Page-locked Memory = Pinned Memory）

確保したCPUメモリの物理アドレスが固定していて、ページアウトされないことが保証されている。システム（OS側）が配列のアドレスを記憶している。

CPUを介さないDMA転送が可能。

CUDA 2.3

```
cudaHostAlloc( (void**) &a_h, n*sizeof(double), cudaHostAllocDefault );
```

CUDA 2.3

```
cudaMallocHost((void**) &a_h, n*sizeof(double));
```

cudaHostAlloc と、ほぼ同じに使える

CUDA 4.0

malloc() でメモリをpageable memory として確保したあと、
cudaHostRegister(a_h, n*sizeof(double), flags); で
pinned memory とする。

Pageable と Page-locked の場合のデータ転送速度を比較する。

Source Code #05

Zero Copy の方法

GPU から CPU のメモリを直接読み書きできるように、アドレスをマッピングする。

CUDA 3.2 cudaHostAlloc((void**) &a_h, n*sizeof(double),
cudaHostAllocMapped | cudaHostAllocWriteCombined);

cudaSetDeviceFlags(cudaDeviceMapHost);
GPUにHost memory へのMapping の許可

cudaHostGetDevicePointer((void**) &a_d, a_h, 0);

GPU が host memory の Mapping をサポートしているかどうか。

Utility: device Query
cudaGetDeviceProperties(&prop, Device_No);

Support host page-locked memory mapping: Yes
prop.canMapHostMemory = 1

Zero Copy の性能

■ cudaMemcpy によるデータの往復が不要

cudaThreadSynchronize(); は必須。Zero copy 中に CPU による非同期の書き込みを防ぐ。

host メモリをGPUのメモリとして共有できれば、Zero copy は性能を大きく向上させる。

Utility: device Query
cudaGetDeviceProperties(&prop, Device_No);

Integrated: Yes

PCI Express Bus を介した通信に対しても、Warp 単位の処理で Latency を隠蔽するなど、Performance が向上することが期待できる。

Zero Copy の実行性能を Source Code #05 と比較する。

Source Code #06