GPUコンピューティング No.11

CUDA Streamと複数GPU計算

東京工業大学　学術国際情報センター

青木　尊之

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

CUDA Stream

Device での thread の実行(タスク)は、ストリーム単位で管理されている。

- GPU の kernel 関数の実行も 1つの stream
- cudaMemcpy も1つの stream

Default で全ての stream の番号は 0 に設定されている。

同じ stream の番号は、投入順に実行される。

GPU kernel の同時実行が可能になったのは CUDA 3.0 から。

Utility: deviceQuery
Concurrent kernel execution: **Yes**

cudaGetDeviceProperties(&prop, Device_No);
prop.deviceOverlap = **1**

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

Stream の指定

```
cudaStream_t  stream;
cudaStreamCreate( &stream );
```

```
cudaMemcpyAsync(void *dst, const void *src, size_t count,
                enum cudaMemcpyKind kind,
                cudaStream_t stream);

           kind:  cudaMemcpyHostToDevice
                  cudaMemcpyDeviceToHost
```

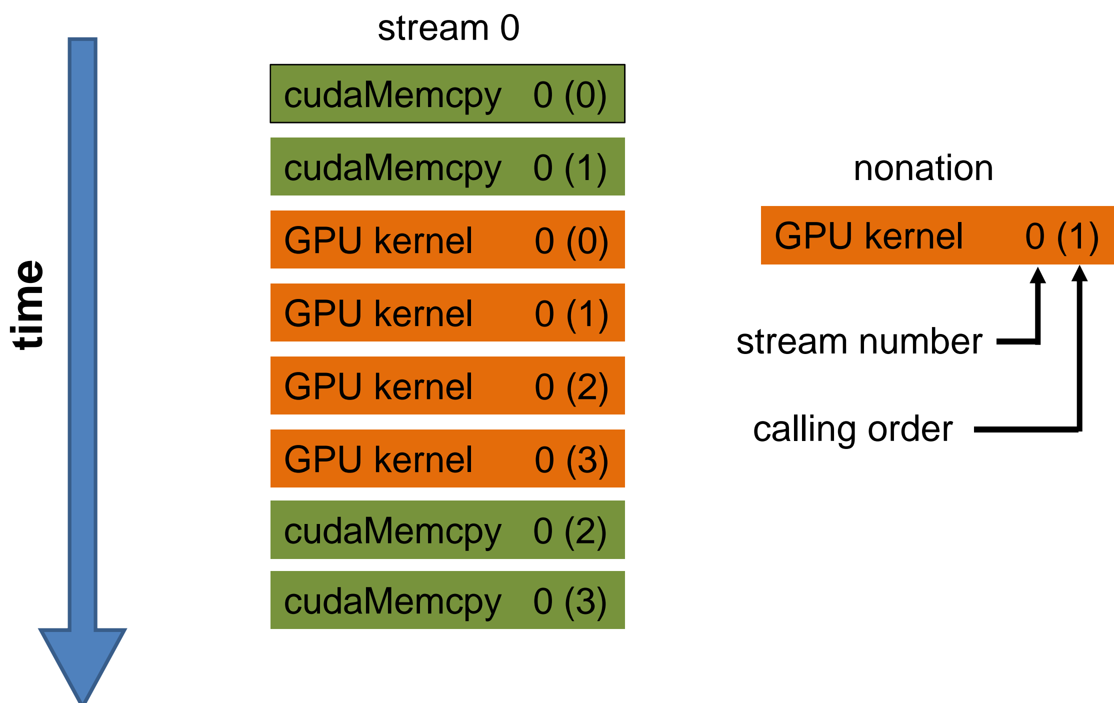
```
gpu_kernel_fucntion<<< grid, thread, 0, stream>>(a, b, c);

                 grid:  block_per_grid,
                 thread: threads_per_block,
```

```
cudaStreamDestroy( stream );
```

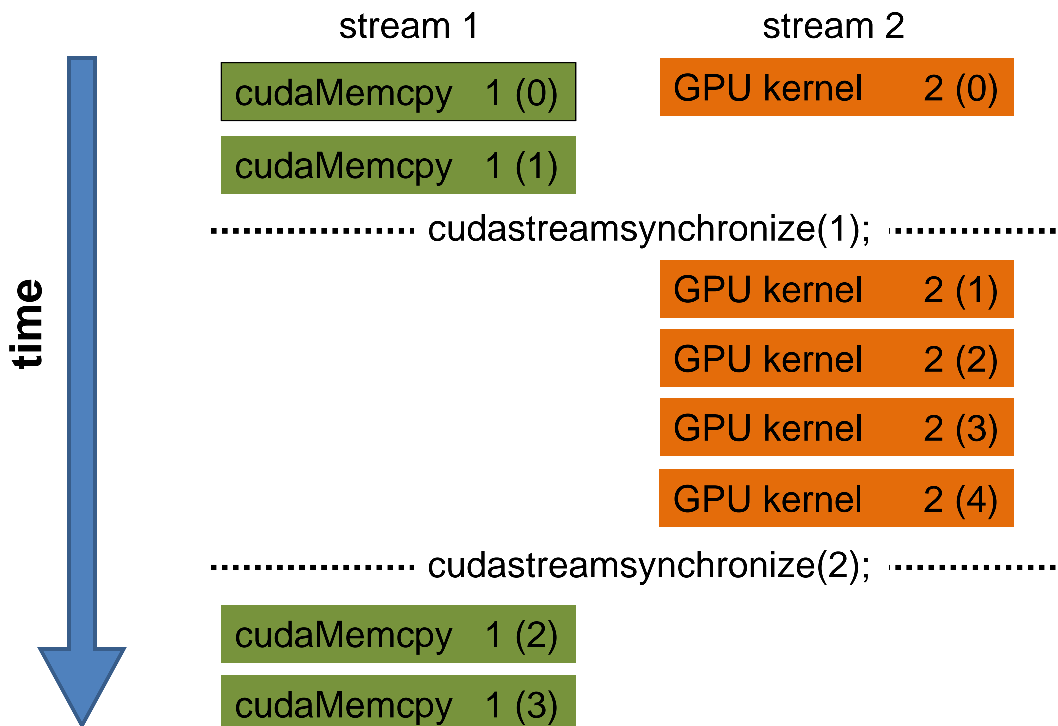
Copyright © Takayuki Aoki , Global Scientific Information and Computing Center, Tokyo Institute of Technology

Stream Scheduling (1)



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

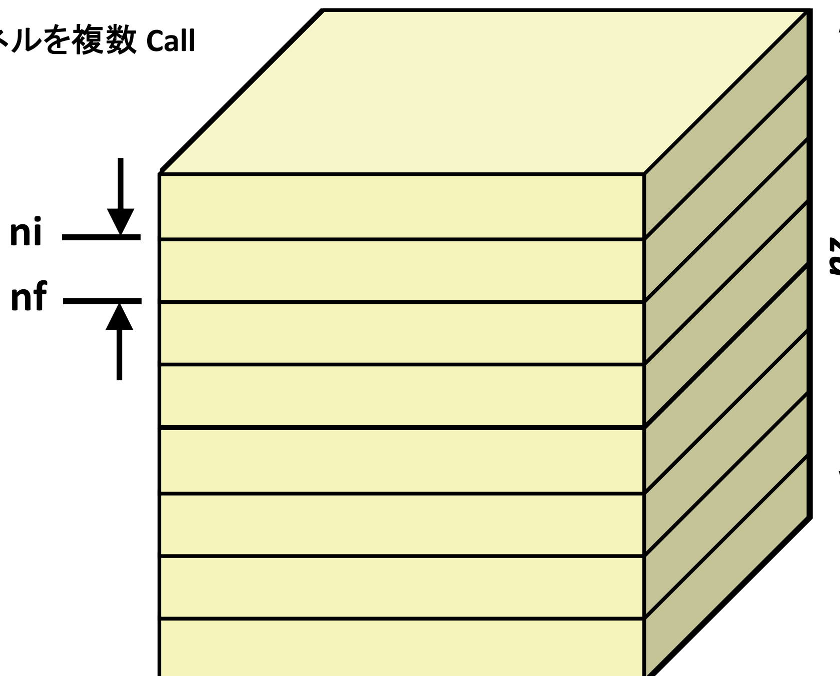
Stream Scheduling (2)



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

計算領域を分割

分割に応じて、GPU カーネルを複数 Call



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

Modified GPU kernel (1)

```
__global__  void  gpu_diffusion3d
// ====================================================================
//
// program    :  CUDA device code for 3D diffusion equation
//
(
   FLOAT     *f,             /* dependent variable               */
   FLOAT     *fn,            /* dependent variable               */
   int       nx,             /* grid number in the x-direction   */
   int       ny,             /* grid number in the y-direction   */
   int       nz,             /* grid number in the z-direction   */
   int       zi,             /* z-dimensional starting grid number */
   int       zf,             /* z-dimensional ending grid number   */
   FLOAT     ce,             /* coefficient no.0                 */
   FLOAT     cw,             /* coefficient no.1                 */
   FLOAT     cn,             /* coefficient no.2                 */
   FLOAT     cs,             /* coefficient no.3                 */
   FLOAT     ct,             /* coefficient no.4                 */
   FLOAT     cb,             /* coefficient no.5                 */
   FLOAT     cc              /* coefficient no.6                 */
)
// --------------------------------------------------------------------
```

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

Modified GPU kernel (2)

```
for(jz = zi; jz < zf; jz++) {

       j = nx*ny*jz + nx*jy + jx;

       je = j+1; jw = j-1;  jn = j+nx; js =j-nx;
       jt = j + nx*ny;  jb = j - nx*ny;

       if(jx == nx-1) je = j;
       if(jx == 0   ) jw = j;
       if(jy == ny-1) jn = j;
       if(jy == 0   ) js = j;
       if(jz == nz-1) jt = j;
       if(jz == 0   ) jb = j;

       fn[j] = cc*f[j]
             + ce*f[je] + cw*f[jw]
             + cn*f[jn] + cs*f[js]
             + ct*f[jt] + cb*f[jb];
   }
```

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

Serial kernel execution

Without specifying the stream number, the default stream 0 is set.

```
int     zi = 0,  num_stream = 64,  mz = nz/num_stream,
        zf = mz;

    if(on_GPU > 0) {
        dim3  grid(nx/64, ny/2, 1), threads(64, 2, 1);

        for(int k = 0; k < num_stream; k++) {

            gpu_diffusion3d<<< grid, threads >>>
                  (f,fn,nx,ny,nz,zi,zf,ce,cw,cn,cs,ct,cb,cc);

            zi = zf;  zf += mz;
        }
    }
```

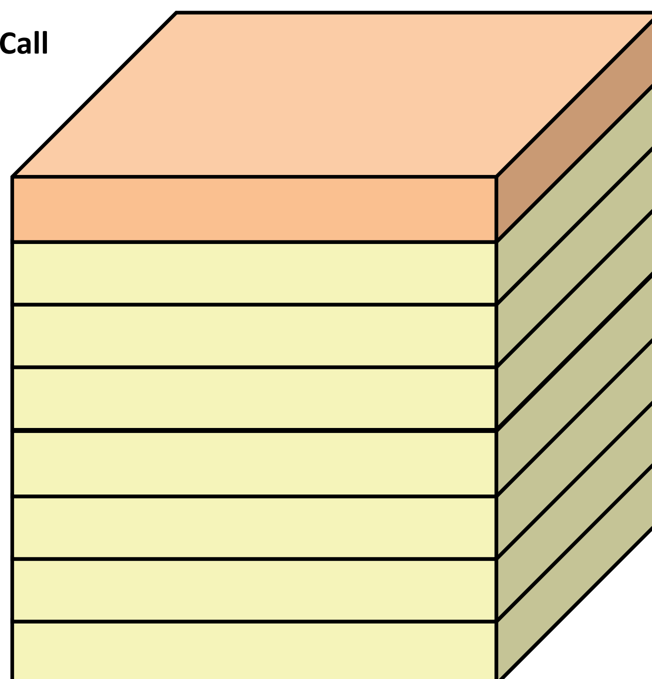
Source Code #22

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

計算領域を分割

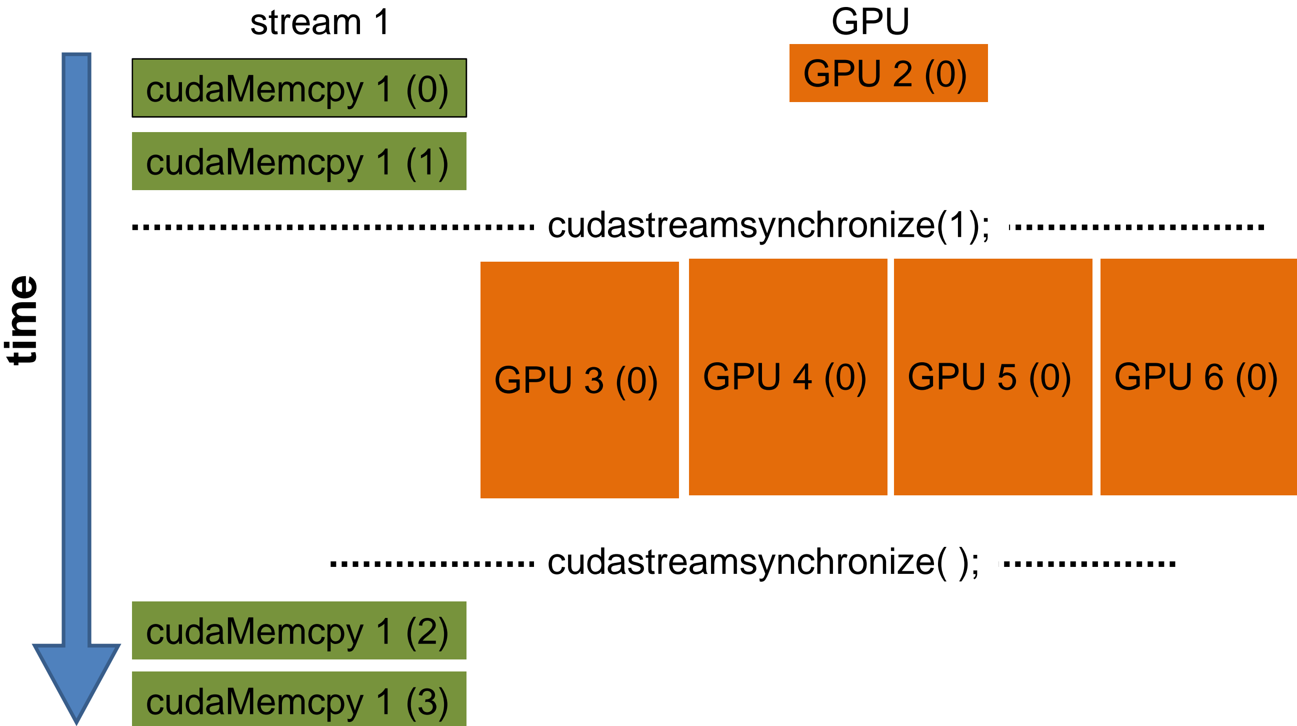
分割に応じて、GPU カーネルを複数 Call

GPU Kernel 0 : execution (call)



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

Stream Scheduling (3)



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

Concurrent Execution

GPU Kernel : stream 1

GPU Kernel : stream 2

GPU Kernel : stream 3

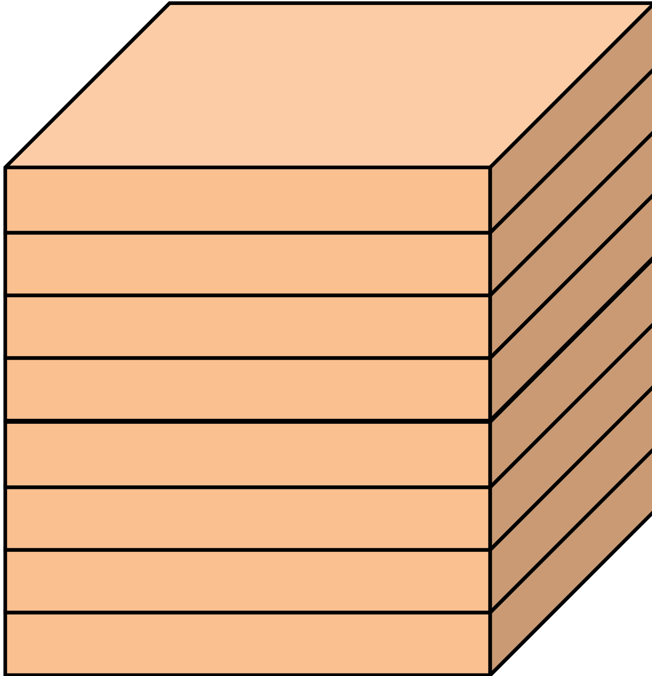
GPU Kernel : stream 4

GPU Kernel : stream 5

GPU Kernel : stream 6

GPU Kernel : stream 7

GPU Kernel : stream 8



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

Concurrent Kernal Execution

```
 int    zi = 0,  num_stream = 32,  mz = nz/num_stream,
        zf = mz;
if(on_GPU > 0) {
   cudaStream_t  streams[1000];
   for(k = 0; k < num_stream; k++)
                 cudaStreamCreate( &(streams[k]) );
        .  .  .  .  .  .  .
    for(int k = 0; k < num_stream; k++) {
        gpu_diffusion3d<<< grid, threads, 0, streams[k] >>>
                  (f,fn,nx,ny,nz,zi,zf,ce,cw,cn,cs,ct,cb,cc);
       zi = zf;  zf += mz;
     }
     cudaThreadSynchronize();
     for(k = 0; k < num_stream; k++)
                 cudaStreamDestroy( streams[k] );
}
```

Source Code #22

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

複数 GPU による計算

複数 GPU を使う目的:

- 高い実行性能(演算性能・メモリバンド幅)を期待する。
- 1枚のボードに搭載されるメモリを超えた大容量のメモリを使う計算が可能になる(C2050: 3GB, C2070: 6GB)

複数 GPU マシンの構成の分類:

- 1ノードに複数の GPU が PCI-e に接続されている。(GeForce GTX 590, Tesla S2050, S2070 等も同じ)
- GPU を搭載したノードが複数インターコネクトされている。

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

CUDA の複数 GPU の取組み

CUDA の複数 GPU への対応は、これから。

GPU Direct　ver. 1 （CUDA 3.0）
　Pinned メモリのcudaMemcpy への利用と Infiniband により
　MPI 通信の RDMA モードのバッファ共有の問題点の改善
　（Mellanox, Voltaire ）

GPU Direct ver. 2 （CUDA 4.0）
　同一チップセットに接続された GPU 同士で、直接 GPU間の
　通信が可能。device メモリが unified メモリとして見える。
　MPI Library は、まだサポートされていない。

※ CUDAにはマルチGPUのための支援もなければ（本質的な）制約もなし

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

複数 CPU による計算

ソフトウェアに関して：

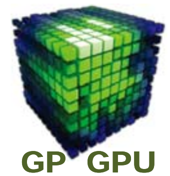
標準的に利用可能なコンパイラ・ライブラリの利用
- CUDA（1GPUのときと同じ）
- CPU側並列化のためのコンパイラ・ライブラリ
- OpenMP、MPI ライブラリなど

ハードウェアに関して：

GPU のメモリは、GPU ボード上に分散して配置。
- 分散メモリの計算機と認識した方が良い。

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

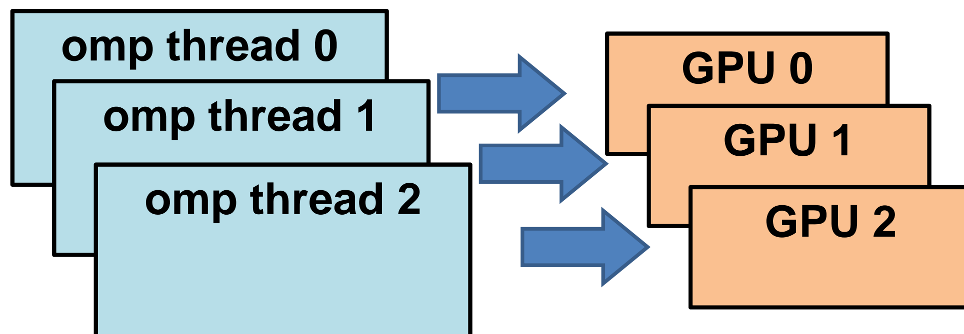
複数 GPU による計算



GPU は単独では動作しない。host から kernel code を各 GPUに launch して実行する。

host 側も GPU 数と同じ数の thread を用意し、各 CPU の thread から kernel code を各 GPUに launch するのが簡単。

Open MP による GPU 数と同じ数の thread 計算。



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

THE OPENMP® API SPECIFICATION FOR PARALLEL PROGRAMMING

共有メモリ環境での指示文(#pragma)ベースのマルチスレッド・プログラミング

C/C++ とFortranを標準的にサポート

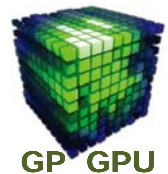
```
Int main(int  argc,  char* argv[])
{
#pragma omp parallel
{
     /* この部分を並列にスレッド
        実行 */
}
Return 0;
}
```

```
Int main(int  argc,  char* argv[])
{
#pragma omp parallel for
for(int i = 0;  i < 1024; i++)
{
     /* この部分を並列にスレッド
        実行 */
}
Return 0;
}
```

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

OpenMP

THE OPENMP® API SPECIFICATION FOR PARALLEL PROGRAMMING



並列リージョン指示文 同期に関する指示文

#pragma omp parallel

処理分散指示文

#pragma omp for
#pragma omp sections

同期に関する指示文

#pragma omp single
#pragma omp master
#pragma omp critical
#pragma omp atomic
#pragma omp barrier

スコープ指示節

private
ordered
firstprivate
lastprivate if
shared
num_threads
default nowait
reduction

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

OpenMP

THE OPENMP® API SPECIFICATION FOR PARALLEL PROGRAMMING

実行環境

omp_set_num_threads(int)
omp_get_num_threads()
omp_get_max_threads()
omp_get_thread_num()
omp_get_num_procs()
omp_set_dynamic(int)
omp_get_dynamic()
omp_get_nested()

環境変数

OMP_NUM_THREADS
OMP_SCHEDULE
OMP_DYNAMIC
OMP_NESTED

時間計測ルーチン

omp_get_wtime()
omp_get_wtick()

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

Open MPによるマルチCPU計算

```
int    j,  jx,  jy,  jz,  je,  jw,  jn,  js,  jt,  jb;

#pragma omp parallel for private (jy, jx, j, je, jw, jn, js, jt, jb)
num_threads(8)

//  printf("omp_get_thread_num=%d¥n", omp_get_thread_num());

for(jz=0; jz < nz; jz++) {
    for(jy=0; jy < ny; jy++) {
        for(jx=0; jx < nx; jx++) {
            j = nx*ny*jz + nx*jy + jx;
            je = j+1; jw = j-1;  jn = j+nx; js =j-nx;
            jt = j + nx*ny;  jb = j - nx*ny;

            fn[j] = cc*f[j] + ce*f[je] + cw*f[jw]
                  + cn*f[jn] + cs*f[js] + ct*f[jt] + cb*f[jb];
        }
    }
}
```

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

Open MPによるマルチCPU計算

<u>コンパイルの方法</u>:

OpenMP部分のコンパイル方法
gccでは: gcc –c –fopenmp foo.c

CUDAとOpenMPが同一ソースファイルに共存する場合
nvcc –c –Xcompiler –fopenmp foo.cu

<u>リンク方法</u>:

nvcc もしくはgcc を用いてリンク
OpenMPのライブラリを指定する (libgomp)
gcc host.o gpu.o –lgomp

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

マルチCPUによる実行速度の検証

`num_threads(int)` ※int には thread 数を指定

`omp_set_num_threads(int);`
※ int には thread 数を指定

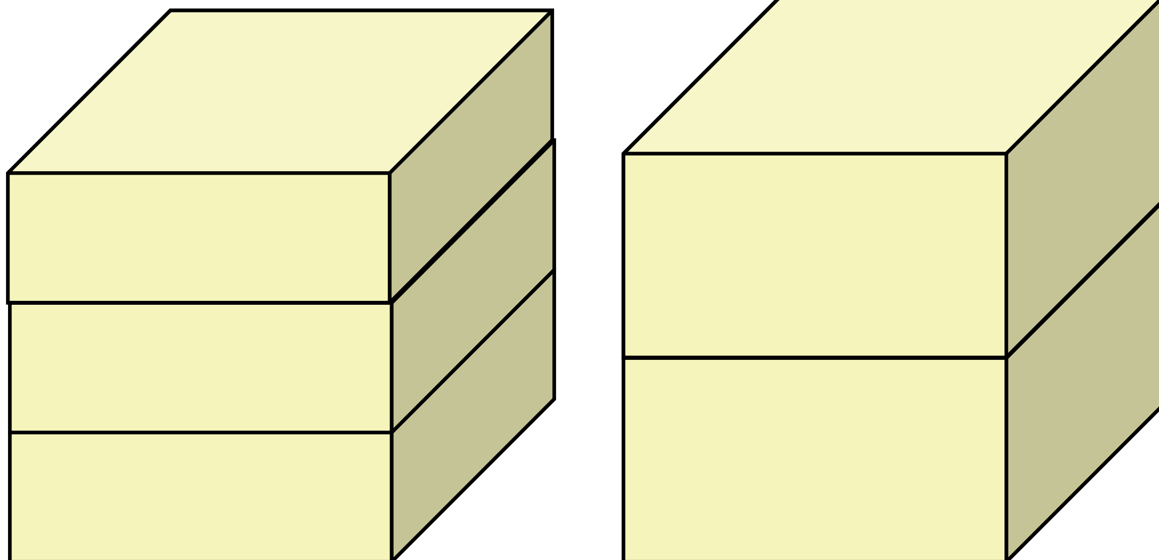
どちらかの方法で thread 数を 12 まで増やし、実行速度を検証する。

Source Code #23

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

マルチGPUによる実行（1）

CPU のマルチプロセスによる並列計算と同様に、計算領域を分割して実行する。

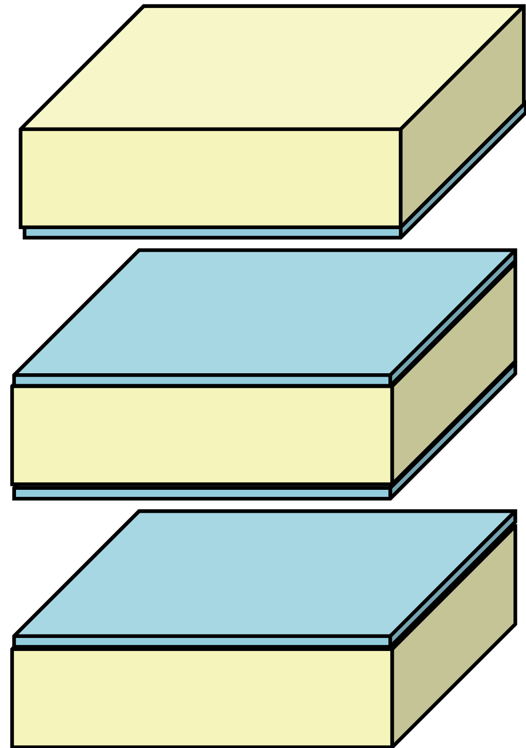


Copyright © Takayuki Aoki , Global Scientific Information and Computing Center, Tokyo Institute of Technology

マルチGPUによる実行(2)

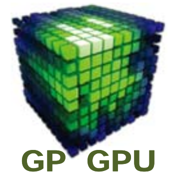
分割されたそれぞれの領域を
1つのGPUが計算を担当する。

z方向に領域分割した場合、
真ん中の領域は上下1層分、
計算領域の外へのアクセスが
発生する。

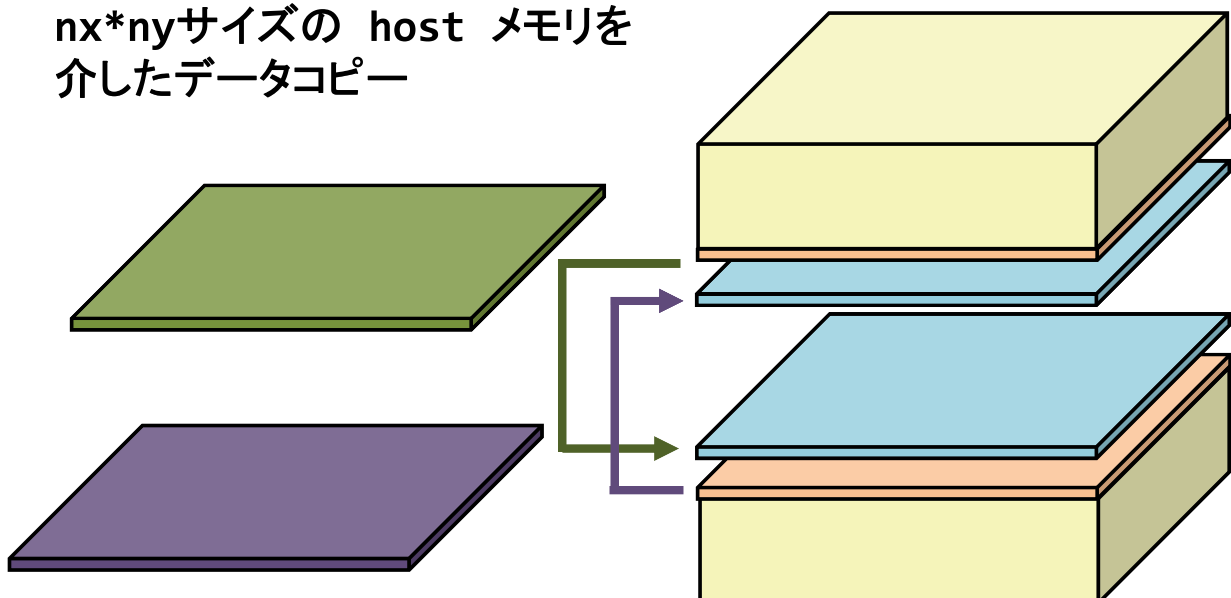


Copyright © Takayuki Aoki , Global Scientific Information and Computing Center, Tokyo Institute of Technology

マルチGPUによる実行(3)



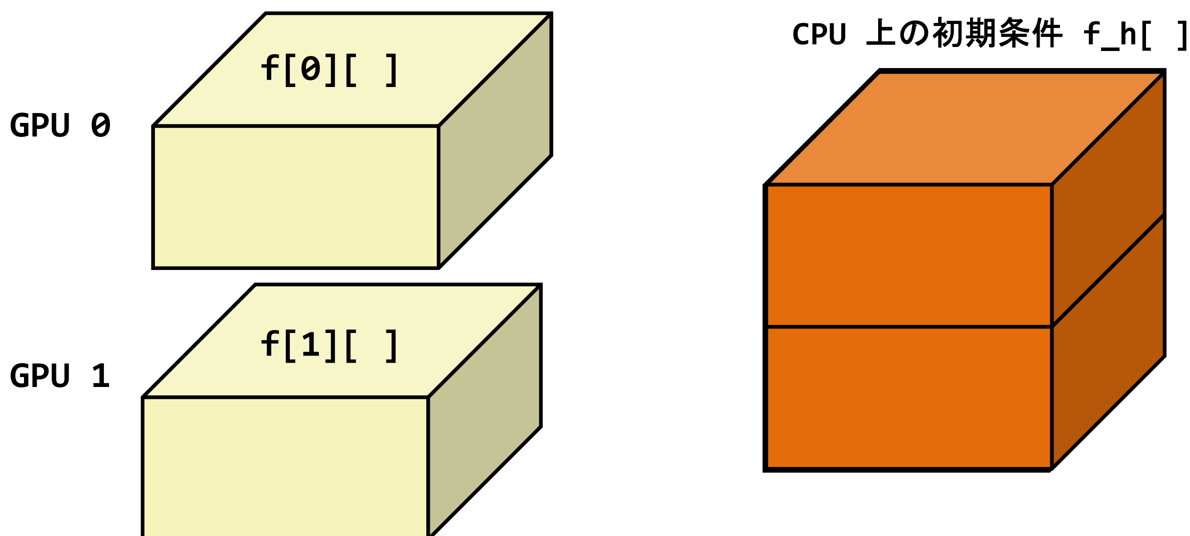
nx*nyサイズの host メモリを
介したデータコピー



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

マルチGPU用の初期条件

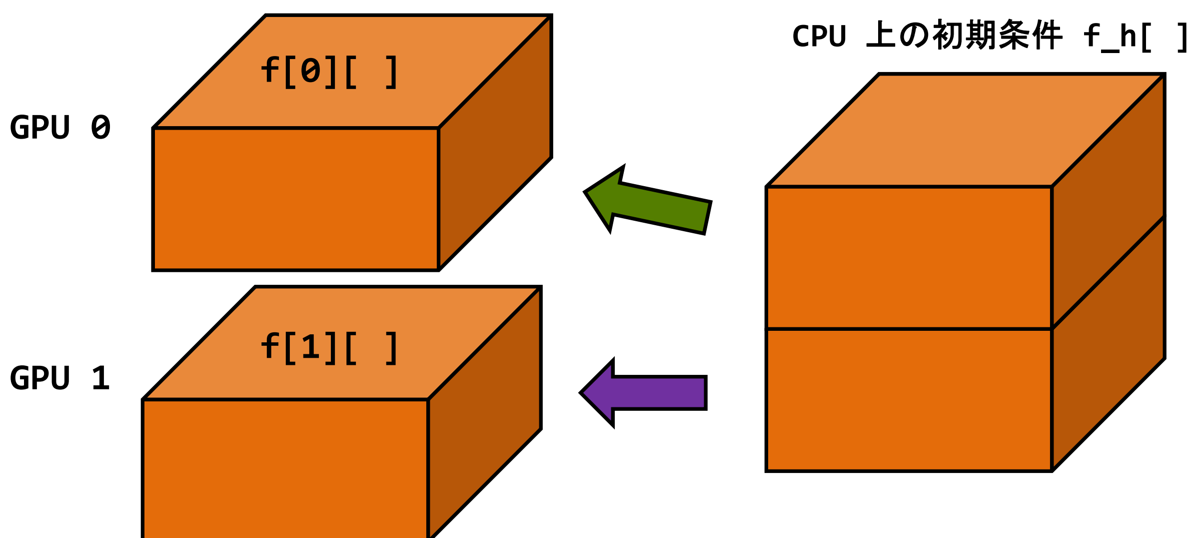
CPU で作成した初期条件を分割して GPU の global memory に転送する。



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

マルチGPU用の初期条件

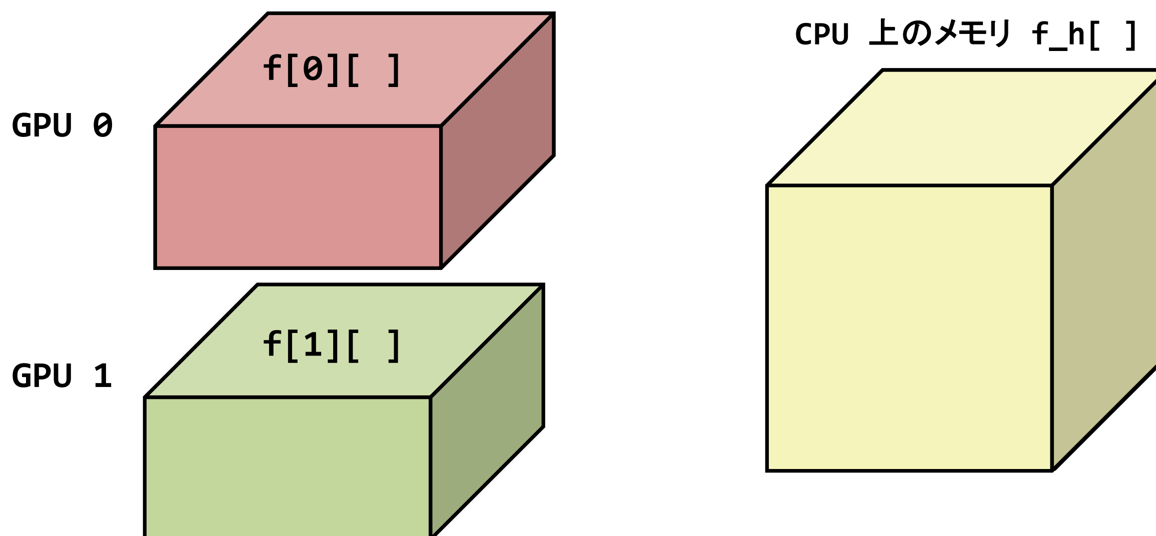
CPU で作成した初期条件を分割して GPU の global memory に転送する。



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

マルチGPUの精度検証

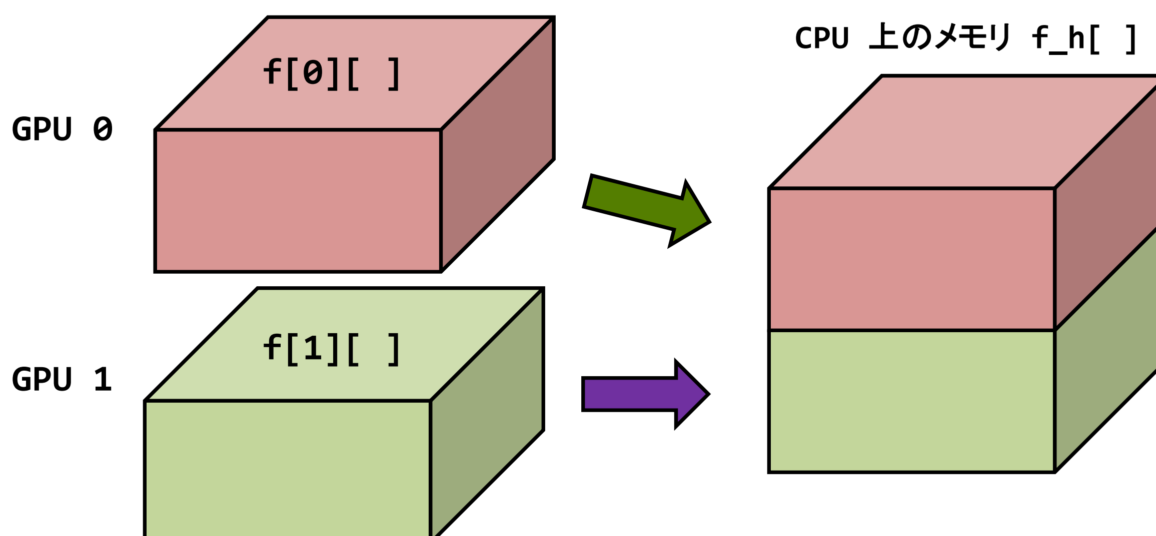
GPU の計算結果を CPU のメモリに転送し、そこで計算精度の検証を行う。



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

マルチGPUの精度検証

GPU の計算結果を CPU のメモリに転送し、そこで計算精度の検証を行う。



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