CUDA ON ARM PLATFORM—线程层次

NVIDIA企业级开发者社区 何琨

AGENDA

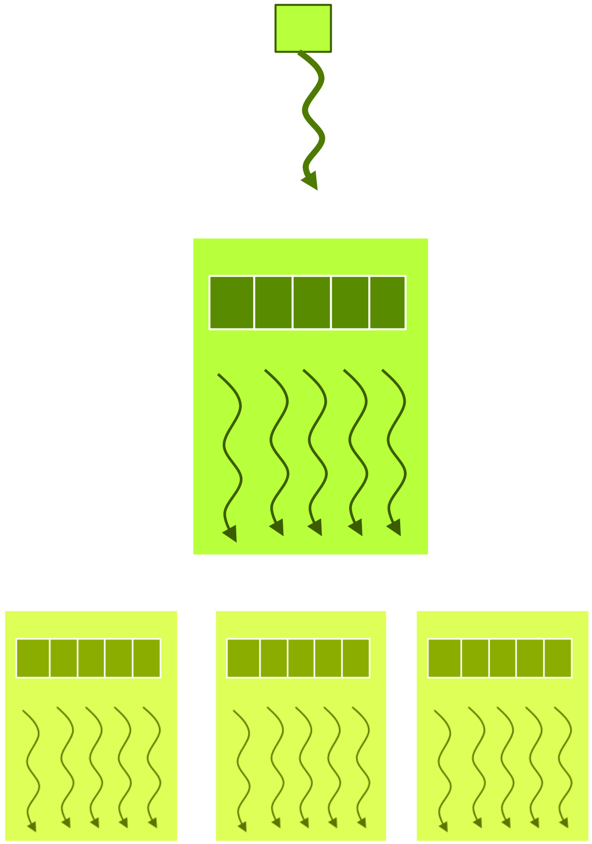
CUDA并行计算基础

- CUDA线程层次
- CUDA线程索引
- CUDA线程分配

CUDA线程层次

`HelloFromGPU <<<?, ?>>>();`

- Thread: sequential execution unit
  - 所有线程执行相同的核函数
  - 并行执行
- Thread Block: a group of threads
  - 执行在一个Streaming Multiprocessor (SM)
  - 同一个Block中的线程可以协作
- Thread Grid: a collection of thread blocks
  - 一个Grid当中的Block可以在多个SM中执行



线程层次

- 执行设置:

dim3 grid(3,2,1), block(5,3,1)

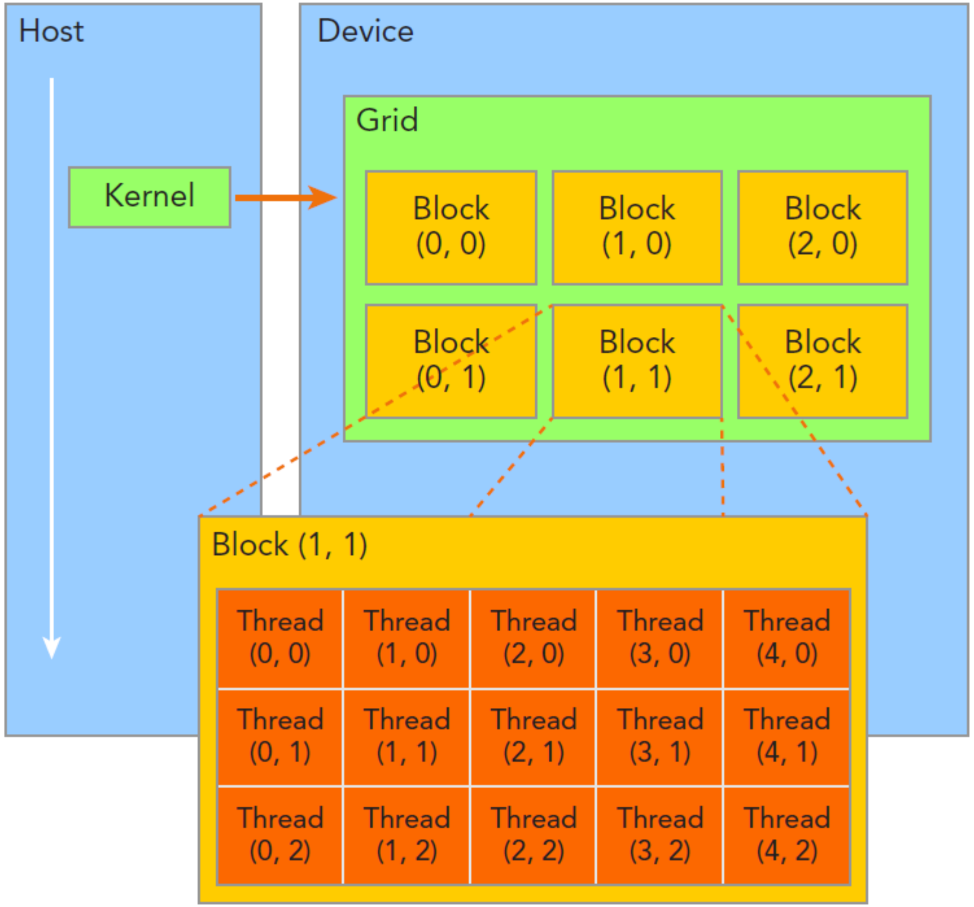
- Built-in variables:
  - threadIdx.[x y z]

  是执行当前kernel函数的线程在block中的索引值
  - blockIdx.[x y z]

  是指执行当前kernel函数的线程所在block，在grid中的索引值
  - blockDim.[x y z]

  表示一个grid中包含多少个block
  - gridDim.[x y z]

  表示一个block中包含多少个线程



线程层次

- 我们写的程序:

```
__global__ void add( int *a, int *b, int *c ) {
    c[threadIdx.x] = a[threadIdx.x] + b[threadIdx.x];
}
add<<<1,4>>>( a, b, c);
```

- 实际上在设备上运行的样子:

Thread 0

```
c[0] = a[0] + b[0];
```

Thread 1

```
c[1] = a[1] + b[1];
```

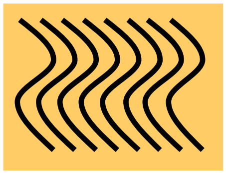
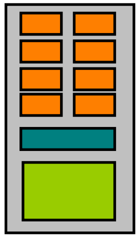
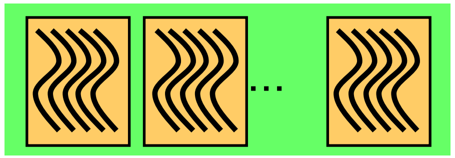
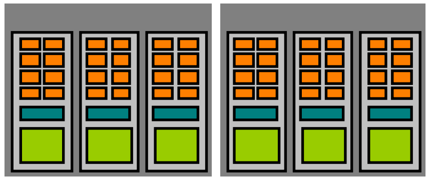
Thread 2

```
c[2] = a[2] + b[2];
```

Thread 3

```
c[3] = a[3] + b[3];
```

线程层次

Software	GPU	
 Thread	 CUDA Core	Threads are executed by cuda core
	 SM	Thread blocks are executed on SM
 Grid	 Device	A kernel is launched as a grid of thread blocks

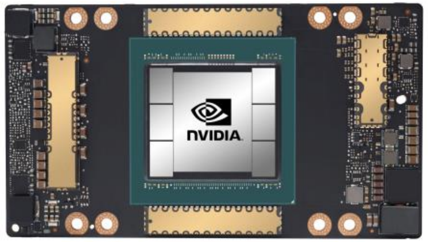
线程层次

WHY BOTH BLOCK AND THREAD?

为什么不是只有线程？这样我们不就可以使用的更方便了吗？

Blocks 好像也不是必须的

- 增加了一个层级的抽象，也增加了复杂度
- 使用Blocks或者Grid我们收获了什么？



This is related to the GPU Architecture!

CUDA的执行流程

1. 加载核函数
2. 将Grid分配到一个Device
3. 根据<<<..>>>内的执行设置的第一个参数，Giga threads engine将block分配到SM中。一个Block内的线程一定会在同一个SM内，一个SM可以有很多个Block。
4. 根据<<<..>>>内的执行设置的第二个参数，Warp调度器会将调用线程。
5. Warp调度器为了提高运行效率，会将每32个线程分为一组，称作一个warp。
6. 每个Warp会被分配到32个core上运行。

线程层次

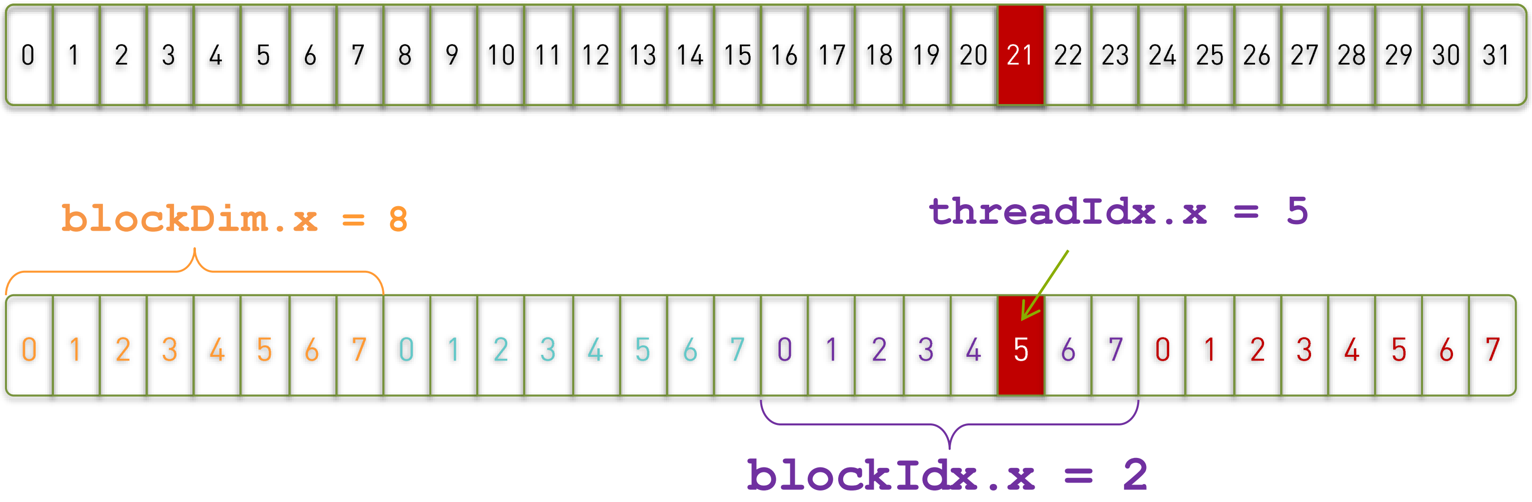
- 硬件调度：
  - Grid： GPU(GPC)级别的调度单位
  - Block(CTA)：SM级别的调度单位
  - Threads/Warp：CUDA core级别的调度单位
- 资源和通信：
  - Grid：共享同样的kernel 和 Context
  - Block(CTA)：同一个SM(Streaming Multiprocessor)，同一个SM(Shared Memory)
  - Threads/Warp: 允许同一个warp中的thread读取其他thread的值

线程层次

- 硬件调度：
  - Grid： GPU(GPC)级别的调度单位
  - Block(CTA)：SM级别的调度单位
  - Threads/Warp：CUDA core级别的调度单位
- 资源和通信：
  - Grid：共享同样的kernel 和 Context
  - Block(CTA)：同一个SM(Streaming Multiprocessor)，同一个SM(Shared Memory)
  - Threads/Warp: 允许同一个warp中的thread读取其他thread的值

CUDA的线程索引

- 如何确定线程执行地数据




```
int index = threadIdx.x + blockIdx.x * blockDim.x;
          =         5   +       2    * 8;
          = 21;
```

CUDA的线程索引

```
__global__  void add(const double *x, const double
*y, double *z)
{
    const int n = blockDim.x * blockIdx.x + threadIdx.x;
    z[n] = x[n] + y[n];
}
```

每个线程都执行相同的命令

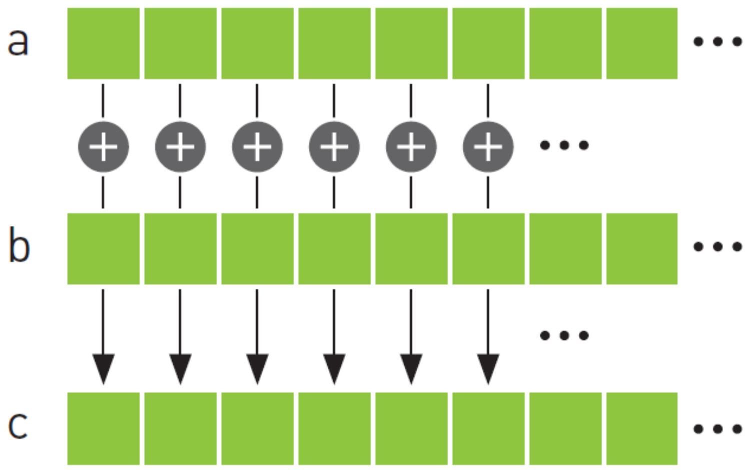
CUDA PROGRAMMING BY EXAMPLE

Case: Vector Add

- Parallelizable problem:
  - c = a + b
  - a, b, c are vectors of length N
- CPU implementation:

```
void main(){
   int size = N * sizeof(int);
   int *a, *b, *c;
   a = (int *)malloc(size);
   b = (int *)malloc(size);
   c = (int *)malloc(size);
   memset(c, 0, size);
   init_rand_f(a, N);
   init_rand_f(b, N);

   vecAdd(N, a, b, c);
}
```

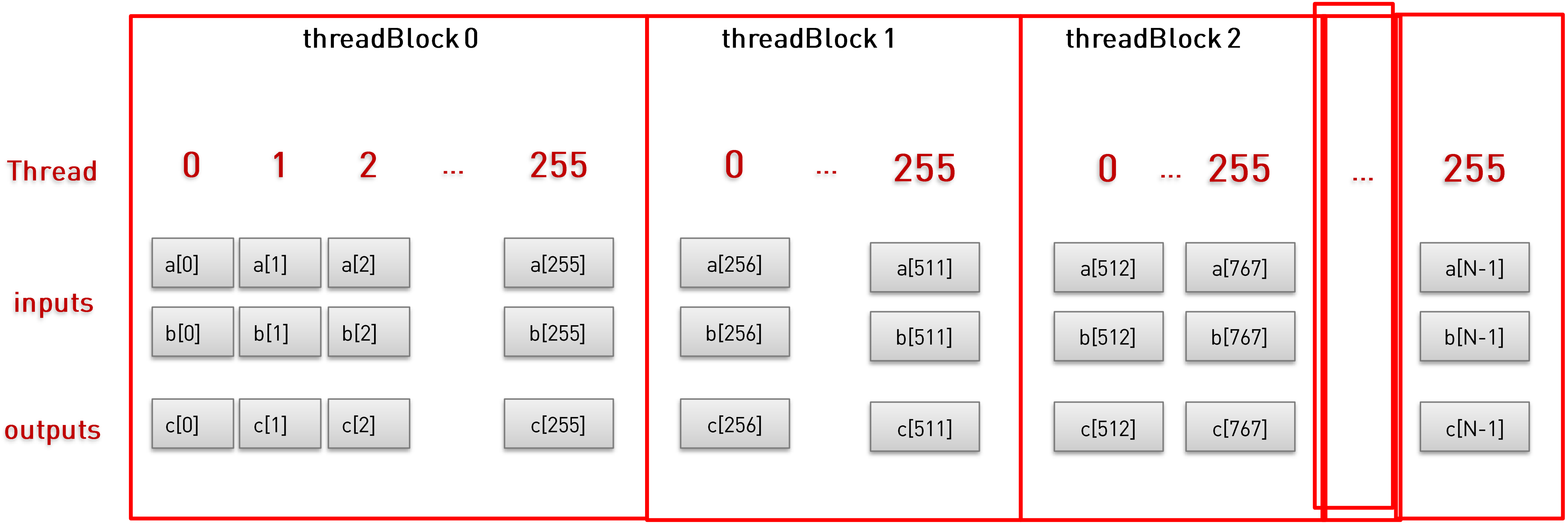



```
void vecAdd (int n, int *a,
             int *b, int *c)
{
   for(int i = 0; i < n; i++)
   {
      c[i] = a[i] + b[i];
   }
}
```

PARALLELIZATION OF VECTORADD

Thread	0	1	2	...	255	256	...	511	512	...	767	...	N-1
inputs	a[0]	a[1]	a[2]		a[255]	a[256]		a[511]	a[512]		a[767]		a[N-1]
	b[0]	b[1]	b[2]		b[255]	b[256]		b[511]	b[512]		b[767]		b[N-1]
outputs	c[0]	c[1]	c[2]		c[255]	c[256]		c[511]	c[512]		c[767]		c[N-1]

PARALLELIZATION OF VECTORADD



work index i = threadIdx.x + blockIdx.x * blockDim.x;

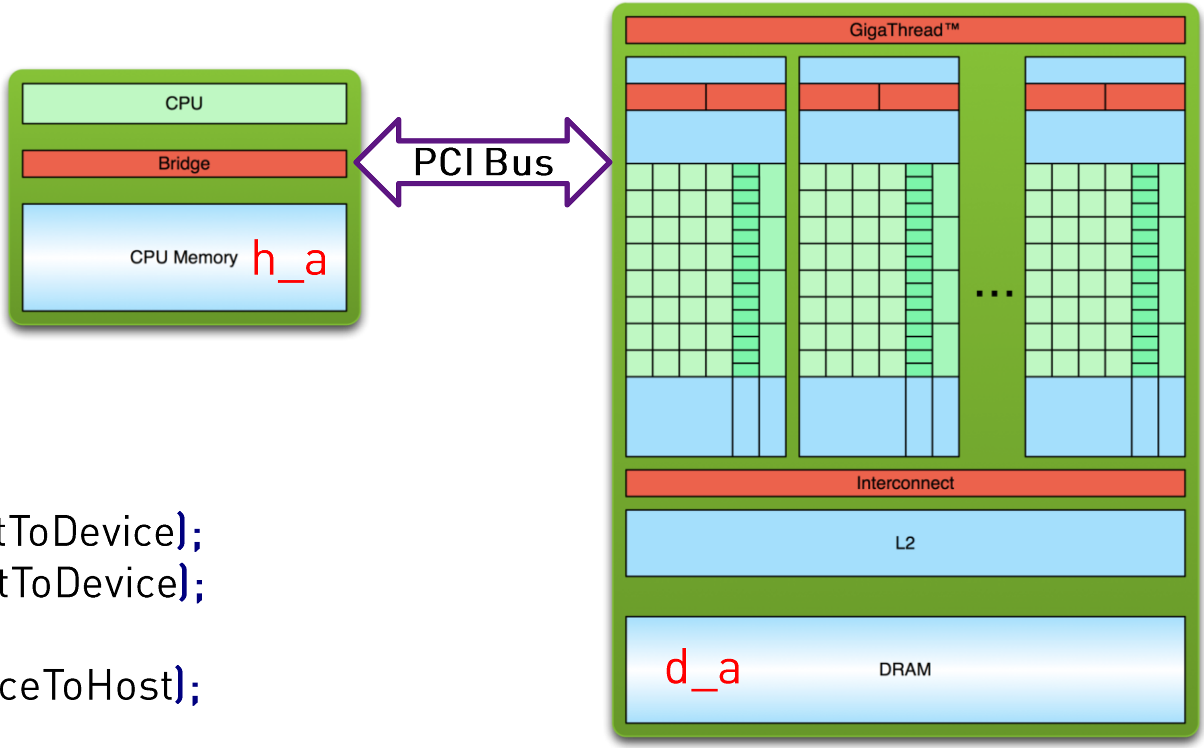
GPU CODE WORKFLOW

Allocate GPU Memories

```
int main(void) {
    size_t size = N * sizeof(int);
    int *h_a, *h_b; int *d_a, *d_b, *d_c;
    h_a = (int *)malloc(size);
    h_b = (int *)malloc(size);
    ...
    cudaMalloc((void **)&d_a, size);
    cudaMalloc((void **)&d_b, size);
    cudaMalloc((void **)&d_c, size);

    cudaMemcpy(d_a, h_a, size, cudaMemcpyHostToDevice);
    cudaMemcpy(d_b, h_b, size, cudaMemcpyHostToDevice);
    vectorAdd<<<grid, block>>>(d_a, d_b, d_c, N);
    cudaMemcpy(h_c, d_c, size, cudaMemcpyDeviceToHost);

    cudaFree(d_a); cudaFree(d_b); cudaFree(d_c);
    free(h_a); free(h_b);
    return 0;}
```



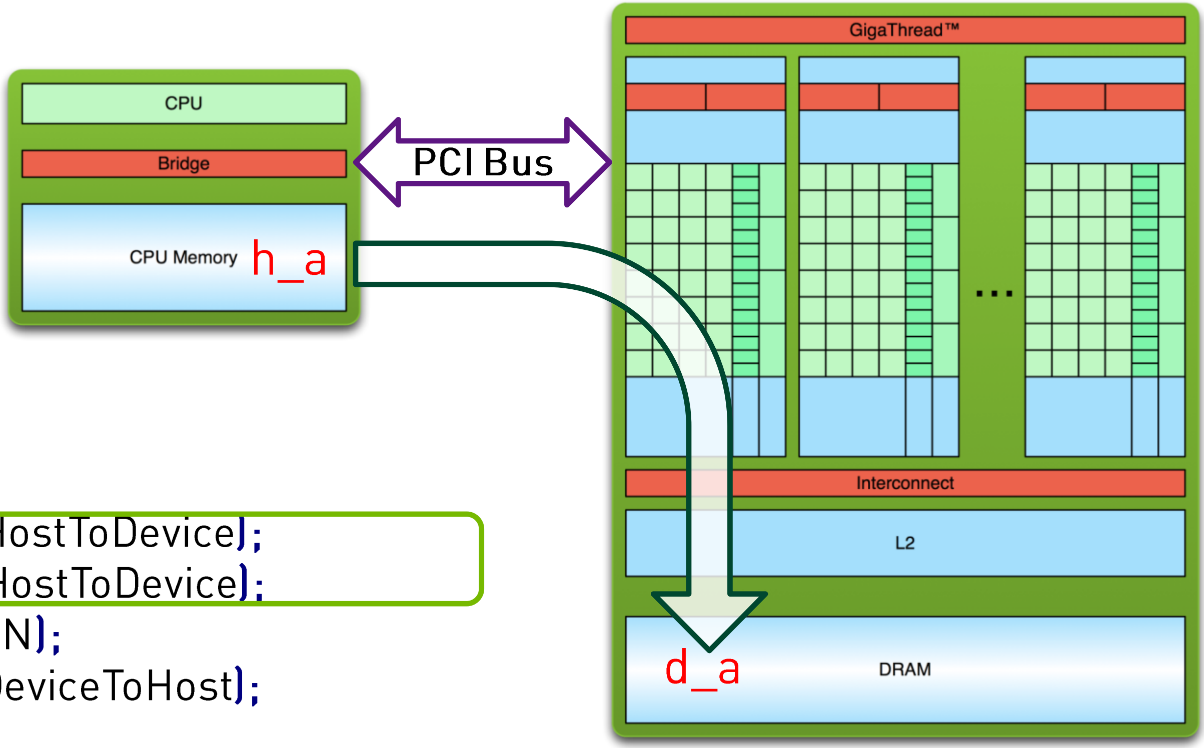
GPU CODE WORKFLOW

Copy data from CPU to GPU

```
int main(void) {
    size_t size = N * sizeof(int);
    int *h_a, *h_b; int *d_a, *d_b, *d_c;
    h_a = (int *)malloc(size);
    h_b = (int *)malloc(size);
    ...
    cudaMalloc((void **)&d_a, size);
    cudaMalloc((void **)&d_b, size);
    cudaMalloc((void **)&d_c, size);

    cudaMemcpy(d_a, h_a, size, cudaMemcpyHostToDevice);
    cudaMemcpy(d_b, h_b, size, cudaMemcpyHostToDevice);
    vectorAdd<<<grid, block>>>(d_a, d_b, d_c, N);
    cudaMemcpy(h_c, d_c, size, cudaMemcpyDeviceToHost);

    cudaFree(d_a); cudaFree(d_b); cudaFree(d_c);
    free(h_a); free(h_b);
    return 0;}
```



GPU CODE WORKFLOW

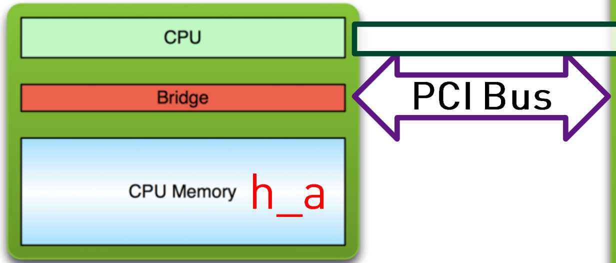
Invoke the CUDA Kernel

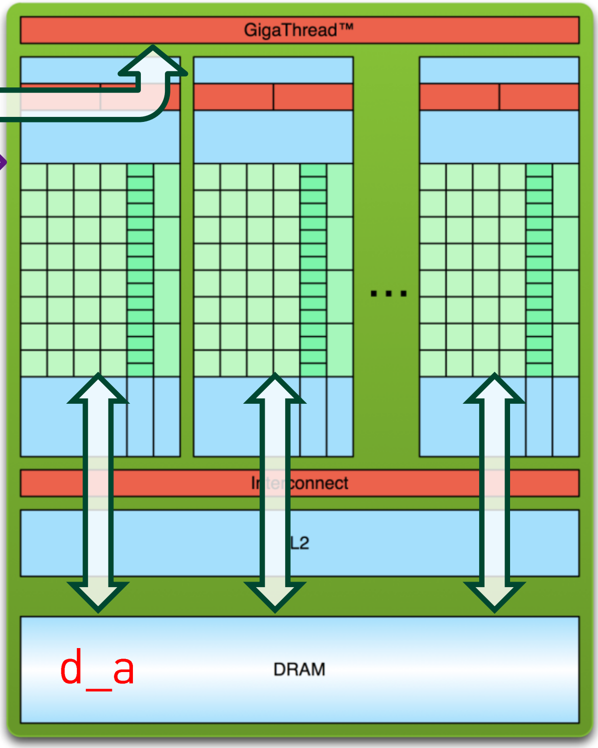
```
int main(void) {
    size_t size = N * sizeof(int);
    int *h_a, *h_b; int *d_a, *d_b, *d_c;
    h_a = (int *)malloc(size);
    h_b = (int *)malloc(size);
    ...
    cudaMalloc((void **)&d_a, size);
    cudaMalloc((void **)&d_b, size);
    cudaMalloc((void **)&d_c, size);

    cudaMemcpy(d_a, h_a, size, cudaMemcpyHostToDevice);
    cudaMemcpy(d_b, h_b, size, cudaMemcpyHostToDevice);
    vectorAdd<<<grid, block>>>(d_a, d_b, d_c, N);
    cudaMemcpy(h_c, d_c, size, cudaMemcpyDeviceToHost);

    cudaFree(d_a); cudaFree(d_b); cudaFree(d_c);
    free(h_a); free(h_b);
    return 0;}
```

Device Pointers





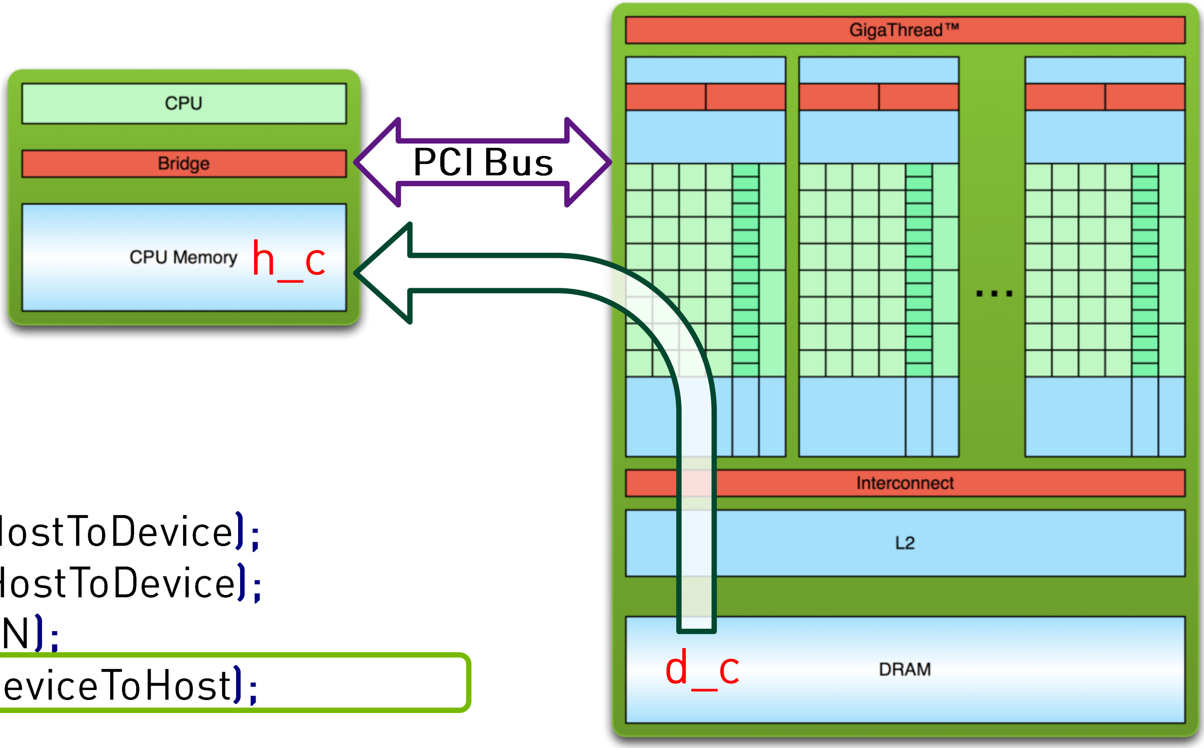
GPU CODE WORKFLOW

Copy result from GPU to CPU

```
int main(void) {
    size_t size = N * sizeof(int);
    int *h_a, *h_b; int *d_a, *d_b, *d_c;
    h_a = (int *)malloc(size);
    h_b = (int *)malloc(size);
    ...
    cudaMalloc((void **)&d_a, size);
    cudaMalloc((void **)&d_b, size);
    cudaMalloc((void **)&d_c, size);

    cudaMemcpy(d_a, h_a, size, cudaMemcpyHostToDevice);
    cudaMemcpy(d_b, h_b, size, cudaMemcpyHostToDevice);
    vectorAdd<<<grid, block>>>(d_a, d_b, d_c, N);
    cudaMemcpy(h_c, d_c, size, cudaMemcpyDeviceToHost);

    cudaFree(d_a); cudaFree(d_b); cudaFree(d_c);
    free(h_a); free(h_b);
    return 0;}
```



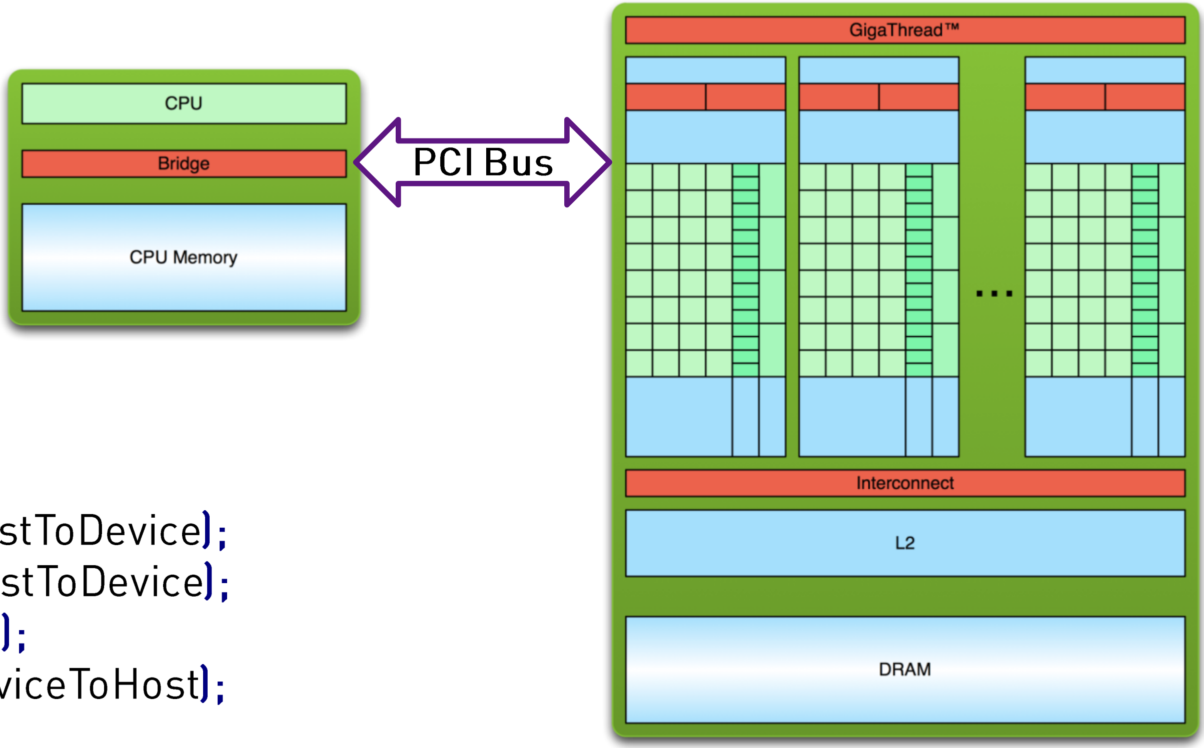
GPU CODE WORKFLOW

Release GPU Memories

```
int main(void) {
    size_t size = N * sizeof(int);
    int *h_a, *h_b; int *d_a, *d_b, *d_c;
    h_a = (int *)malloc(size);
    h_b = (int *)malloc(size);
    ...
    cudaMalloc((void **)&d_a, size);
    cudaMalloc((void **)&d_b, size);
    cudaMalloc((void **)&d_c, size);

    cudaMemcpy(d_a, h_a, size, cudaMemcpyHostToDevice);
    cudaMemcpy(d_b, h_b, size, cudaMemcpyHostToDevice);
    vectorAdd<<<grid, block>>>(d_a, d_b, d_c, N);
    cudaMemcpy(h_c, d_c, size, cudaMemcpyDeviceToHost);

    cudaFree(d_a); cudaFree(d_b); cudaFree(d_c);
    free(h_a); free(h_b);
    return 0;}
```



CUDA的线程索引

如何设置Gridsize & Blocksize：

```
block_size = 128;
grid_size = (N + block_size - 1) / block_size;
```

CUDA的线程分配

那么，我们的每个BLOCK可以申请多少个线程?

```
Total amount of shared memory per block:       49152 bytes
Total number of registers available per block: 65536
Warp size:                                     32
Maximum number of threads per multiprocessor:  2048
Maximum number of threads per block:           1024
Max dimension size of a thread block (x,y,z): (1024, 1024, 64)
Max dimension size of a grid size    (x,y,z): (2147483647, 65535, 65535)
```

CUDA的线程分配

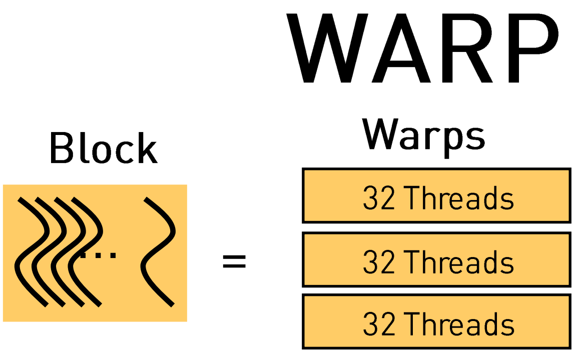
那么，我们的每个BLOCK可以申请多少个线程？

CUDA的线程

那么，我们的每个BLOCK应该申请多少个线程?

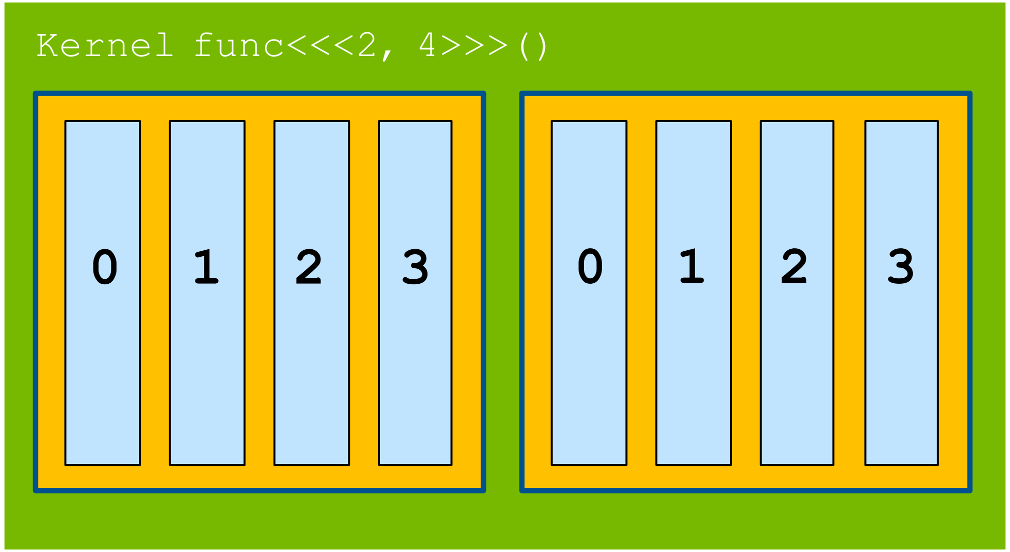
CUDA的线程分配

- Thread blocks can be 1D, 2D, 3D, it's just for convenient. The hardware 'looks' at threads in 1D
- Warp is successive 32 threads in a block
- E.g. blockDim = 160
  - Automatically divided to 5 warps by GPU
- E.g. blockDim = 161
  - If the blockDim is not the Multiple of 32
    The rest of thread will occupy one more warp



CUDA的线程

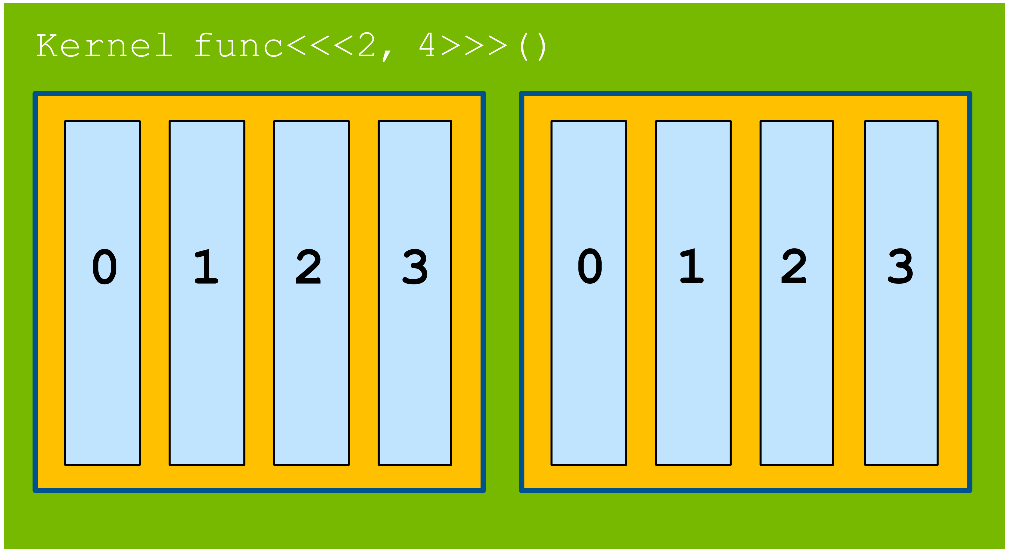
那么，如果我们的数据过大，线程不够用怎么办？

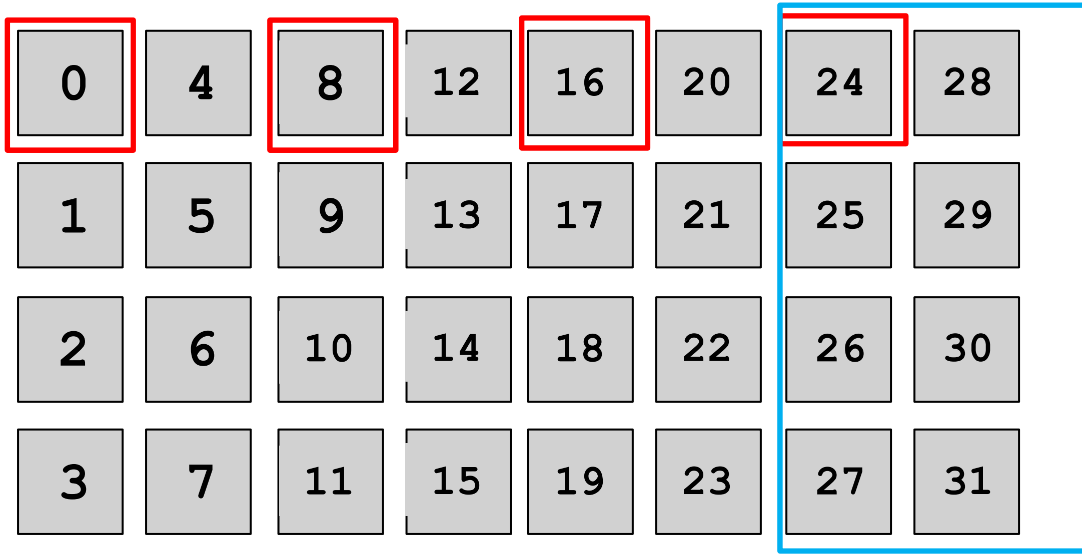


0	4	8	12	16	20	24	28
1	5	9	13	17	21	25	29
2	6	10	14	18	22	26	30
3	7	11	15	19	23	27	31

CUDA的线程

那么，如果我们的数据过大，线程不够用怎么办?





红色框代表索引值为0的线程处理的数据?

CUDA的线程

那么，如果我们的数据过大，线程不够用怎么办?

```
__global__ add(const double *x, const double *y, double *z, int n)
{
    int index = blockDim.x * blockIdx.x + threadIdx.x;
    int stride = blockDim.x * gridDim.x;
    for(; index <n; index +=stride)
        z[index] = x[index] + y[index];
}
```

更多资源:

https://developer.nvidia-china.com

何琨-Ken

北京 密云

扫一扫上面的二维码图案，加我微信

https://www.nvidia.cn/developer/community-training/

NVIDIA®