Harness the Power of GPUs: An Introduction to GPGPU Programming
Lecture 2: Kernels, Threads, Blocks, and Grids

Guido Juckeland
Visiting Scholar
Technische Universität Dresden, Germany

June 16, 2014
Recap: GPU system setup

CPU

Main Memory

System Bus

Accelerator

Local Memory

Application calls Accelerator Library invokes Subprogram

DMA Transfers
Kernels

A kernel is a piece of a program that will be compiled for being executed on the GPU. Kernels are invoked by the host on the device:

```c
int main(int argc, char *argv[]) {
    kernel1<<<..., ...>>>(...);
    kernel2<<<..., ...>>>(...);
    kernel3<<<..., ...>>>(...);
    return 0;
}
```

- Kernel launches are asynchronous on the host
- Kernel order is sequential on the device
Synchronous behavior

- Synchronous operations wait until the activity is finished
Asynchronous behavior

- Asynchronous Operations launch their activity and return immediately to the calling context
Kemel declaration

- Kernels are declared like “normal” functions of return type `void` prepended by the key word `__global__`
  
  Example:
  ```c
  __global__ void do_nothing(float *data) { ... }
  ```

- Since kernels are launched asynchronously they cannot return a value
- Kernels can invoke device functions
  ```c
  __device__ float help_do_nothing() { ... }
  ```
Where is the parallelism?

- A CUDA kernel is executed as a parallel program

- One instance of such an executed kernel is called **thread**

  ```c
  int main( int argc, char *argv[] ) {
    kernel1<<<...,...>>>(...);  // generates many parallel threads
    kernel2<<<...,...>>>(...);  // generates many parallel threads
    kernel3<<<...,...>>>(...);  // generates many parallel threads
    return 0;
  }
  ```

- Threads can execute anything, but best performance when running in SIMD fashion
SIMD stands for Single Instruction, Multiple Data. Parallel threads work on different pieces of the data. Assignment is up to the programmer.

```c
__global__ void settozero( float *elem ) {
    *elem = 0.0f; // bad example
}
...
settozero<<<...,...>>>( devicemem );
```

Thread 1
*elem = 0.0f;

Thread 2
*elem = 0.0f;

Thread 3
*elem = 0.0f;

Thread 4
*elem = 0.0f;

http://s09.idav.ucdavis.edu/talks/02_kayvonf_gpuArchTalk09.pdf
Thread divergence

When threads do different things, the runtime of the threads can vary.

```c
__global__ void diverge( void *data ) {
    if ( data[mythread] > random_number )
        do_a_whole_lot();
    else
        do_nothing;
}
```

Kernel `diverge` runs until the last thread is finished.
Kepler GK110 architecture
Streaming Multiprocessor

Threads are grouped in **blocks** which are executed on one Streaming Multiprocessor (SMX).

They can cooperate using a (small) shared memory (covered later). Threads from different blocks cannot cooperate.
Creating thread blocks

• Order of thread execution is not fixed and can vary
• Threads are executed in batches of 32 threads (warp) in SIMD fashion
• Thread blocks can have arbitrary sizes (within limits, but up to 3D)
• Arrangement of threads is called block

```c
int nx; int ny; int nz; ...
dim3 block(nx, ny, nz); // nx, ny, nz = describes the block in 3D
t kernel<<1,block>>>(...); // creates nx*ny*nz threads in 1 block

Alternative:
k kernel<<1,512>>>(...); // blocksize can also be a number, then 1D
```
The dim3 data structure

```c
struct dim3
{
    unsigned int x, y, z;
};
```

- Create with just assigning a variable, unused dimensions are set to 1
Block size restrictions

- Total number of threads in a block is the product of the number of threads in each dimension
- Total number of threads and threads per dimension have limits

<table>
<thead>
<tr>
<th>Compute Capability</th>
<th>1.0</th>
<th>1.1</th>
<th>1.2</th>
<th>1.3</th>
<th>2.x</th>
<th>3.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Max. block size in x,y</td>
<td>512</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1024</td>
</tr>
<tr>
<td>Max. block size in z</td>
<td></td>
<td></td>
<td></td>
<td>64</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Max. threads per block</td>
<td>512</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1024</td>
</tr>
</tbody>
</table>
Multiple thread blocks (a.k.a. grid of blocks)

- Order of block execution is not fixed and can vary
- Multiple blocks can be put onto one multiprocessor (as long as resources are available)
- Blocks are spread over all multiprocessors
- Arrangement of blocks is called grid

```cpp
int nx; int ny; int nz; ...
dim3 grid(nx, ny, nz); // nx, ny, nz = describes the grid in 3D
kernel<<<grid,block>>>(...); // creates nx*ny*nz blocks
Alternative:
kernl<<1024,512>>>(...); // gridsize can also be a number, then 1D
```
Thread + block mapping
Grid size restrictions

- Total number of blocks in a grid is the product of the number of blocks in each dimension
- Total number of blocks and blocks per dimension have limits

<table>
<thead>
<tr>
<th>Compute Capability</th>
<th>1.0</th>
<th>1.1</th>
<th>1.2</th>
<th>1.3</th>
<th>2.x</th>
<th>3.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Grid dimension</td>
<td>2D (grid(nx,ny,1))</td>
<td></td>
<td></td>
<td></td>
<td>3D (grid(nx,ny,nz))</td>
<td></td>
</tr>
<tr>
<td>Max. grid size in x,y,z</td>
<td>65535</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>$2^{31}-1$</td>
</tr>
<tr>
<td>Max. total grid size</td>
<td>65535</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>$2^{31}-1$</td>
</tr>
</tbody>
</table>
Who am I?

- Threads need to decide on which data they need to work
- Requires ID and size queries

<table>
<thead>
<tr>
<th>Type</th>
<th>ID</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Thread</td>
<td>threadIdx</td>
<td>-</td>
</tr>
<tr>
<td>Block</td>
<td>blockIdx</td>
<td>blockDim</td>
</tr>
<tr>
<td>Grid</td>
<td>-</td>
<td>gridDim</td>
</tr>
</tbody>
</table>

- All variables available in all three dimensions
- Example:
  ```c
  int idx = threadIdx.x + blockIdx.x * blockDim.x;
  ```
Rule of thumb

- Every thread creates one output element

- Example: Vector-Addition

```c
__global__ void vecadd(int *a, int *b, int *c, int N) {
    // who am I?
    int idx = threadIdx.x + blockIdx.x * blockDim.x;

    // if I am inside the vector, work on my data
    if (idx < N) c[idx] = a[idx] + b[idx];
}
```
QUESTIONS?