KU LEUVEN

Geavanceerde Computerarchitectuur
3rd session

Optimization techniques – grid dimensioning and data formatting

Alex Marinsek, alexander.marinsek@kueluven.be

DRAMCO – WaveCore – ESAT

Academic year 2023/24

0 Outline

1 Outline

KU LEUVEN

1 A warp

- A warp is a group of 32 threads
- number of warps $= \left\lceil \frac{\text{Threads per block}}{32} \right\rceil \cdot \text{number of blocks}$
- For example
  - `foo<<<2,64>>>()`
  - `foo<<<4,33>>>()`

1 A warp

- A warp is a group of 32 threads
- number of warps $= \left\lceil \frac{\text{Threads per block}}{32} \right\rceil \cdot$ number of blocks
- For example
  - `foo<<<2,64>>>()` $\Rightarrow$ 4 (128 threads ... 128 cores)
  - `foo<<<4,33>>>()`

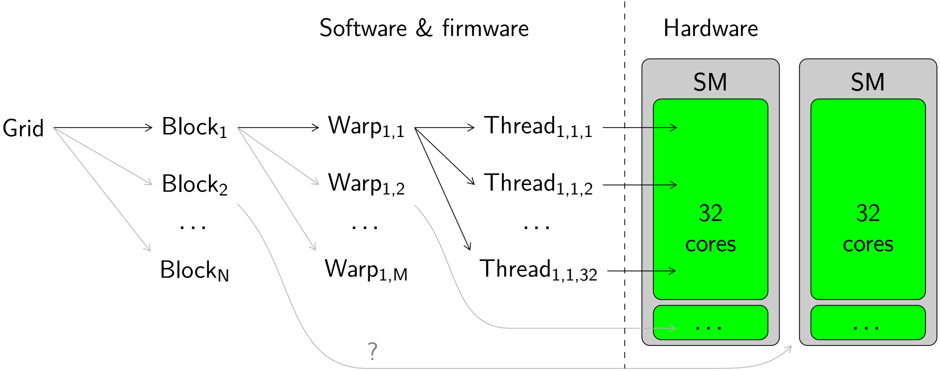
1 A warp

- A warp is a group of 32 threads
- number of warps $= \left\lceil \frac{\text{Threads per block}}{32} \right\rceil \cdot$ number of blocks
- For example
  - `foo<<<2,64>>>()` $\Rightarrow$ 4 (128 threads ... 128 cores)
  - `foo<<<4,33>>>()` $\Rightarrow$ 8 (132 threads ... **256 cores**)

1 A warp

- A warp is a group of 32 threads
- number of warps $= \left\lceil \frac{\text{Threads per block}}{32} \right\rceil \cdot$ number of blocks
- For example
  - `foo<<<2,64>>>()` $\Rightarrow$ 4 (128 threads ... 128 cores)
  - `foo<<<4,33>>>()` $\Rightarrow$ 8 (132 threads ... **256 cores**)
- Also called *a sub-group* in OpenCL, a *wavefront* by AMD, ...

1 Threads are grouped into warps and mapped to cores



A block is divided into warps. The warps of a block are associated with a single stream multiprocessor (SM) – the GT 740 has 2 SMs, each with 192 cores.

1 ”Warping” can over provision GPU resources

Assume an input array with 384 elements and a GPU with 384 CUDA cores
How many warps are spawned?

- `foo<<<2, 192>>>(...)`
- `foo<<<384, 1>>>(...)`

KU LEUVEN

1 "Warping" can over provision GPU resources

Assume an input array with 384 elements and a GPU with 384 CUDA cores
How many warps are spawned?

- `foo<<<2, 192>>>(...)` $\Rightarrow$ 12 (requires 384 cores)
- `foo<<<384, 1>>>(...)`

1 "Warping" can over provision GPU resources

Assume an input array with 384 elements and a GPU with 384 CUDA cores
How many warps are spawned?

- `foo<<<2, 192>>>(...)` $\Rightarrow$ 12 (requires 384 cores)
- `foo<<<384, 1>>>(...)` $\Rightarrow$ 384 (requires 12288 cores)

1 ”Warping” can over provision GPU resources

Assume an input array with 384 elements and a GPU with 384 CUDA cores
How many warps are spawned?

- `foo<<<2, 192>>>(...)` $\Rightarrow$ 12 (requires 384 cores)
- `foo<<<384, 1>>>(...)` $\Rightarrow$ 384 (requires 12288 cores)
- `foo<<<1, 192>>>(...)` $\Rightarrow$ 6 (requires 192 cores, 2 thread iterations)

1 "Warping" can over provision GPU resources

Assume an input array with 384 elements and a GPU with 384 CUDA cores
How many warps are spawned?

- `foo<<<2, 192>>>(...)` ⇒ 12 (requires 384 cores)
- `foo<<<384, 1>>>(...)` ⇒ 384 (requires 12288 cores)
- `foo<<<1, 192>>>(...)` ⇒ 6 (requires 192 cores, 2 thread iterations)
- `foo<<<1, 384>>>(...)` ⇒ 12 (requires 192 cores, 2 scheduling iterations)

1 Thread divergence within a warp

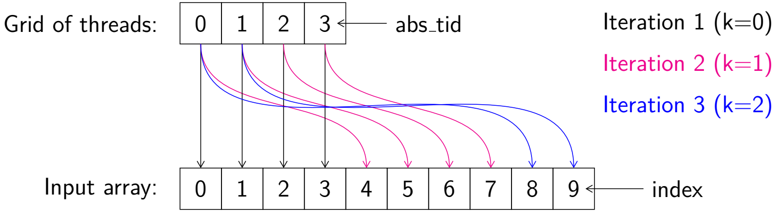
- Part of the warp has higher complexity, taking longer to execute.
- The other part of the warp becomes idle (has finished its job).
- Idle threads occupy cores and do not contribute to the processing of data.

```
__global__ void addOrMultiply(int *a, int *b, int *c) {
    int tid = blockIdx.x * blockDim.x + threadIdx.x;

    if (tid % 2 == 0) {       // If tid is even, add elements
        c[tid] = a[tid] + b[tid];
    } else {                  // If tid is odd, multiply elements
        c[tid] = a[tid] * b[tid];
    }
}
```

1 Striding

- When the matching between threads and elements is not 1:1
- A thread may compute multiple elements
- `if(abs_tid < input_size`... ⇒ `..while(...)`.or `..for(...)`.



`index = abs_tid + k * grid_size`, where `index < input_size`

1 Striding, divergence, and warps on an example

Assume we're using the GT 740 (2 SMs, each with 192 cores) and:

- We spawn a kernel with a single block of 256 threads
- To process an input array with 260 elements
- Striding is implemented

KU LEUVEN

1 Striding, divergence, and warps on an example

Assume we're using the GT 740 (2 SMs, each with 192 cores) and:

- We spawn a kernel with a single block of 256 threads
- To process an input array with 260 elements
- Striding is implemented

- Is there thread divergence?

1 Striding, divergence, and warps on an example

Assume we're using the GT 740 (2 SMs, each with 192 cores) and:

- We spawn a kernel with a single block of 256 threads
- To process an input array with 260 elements
- Striding is implemented

- Is there thread divergence? Yes, at least one warp of 32 threads has to run twice, while only 4 threads are needed. Divergence can be avoided by using inputs with a size that is a multiple of 32.

1 Striding, divergence, and warps on an example

Assume we're using the GT 740 (2 SMs, each with 192 cores) and:

- We spawn a kernel with a single block of 256 threads
- To process an input array with 260 elements
- Striding is implemented
- Is there thread divergence? Yes, at least one warp of 32 threads has to run twice, while only 4 threads are needed. Divergence can be avoided by using inputs with a size that is a multiple of 32.
- How many cores are utilized?

KU LEUVEN

1 Striding, divergence, and warps on an example

Assume we're using the GT 740 (2 SMs, each with 192 cores) and:

- We spawn a kernel with a single block of 256 threads
- To process an input array with 260 elements
- Striding is implemented

- Is there thread divergence? Yes, at least one warp of 32 threads has to run twice, while only 4 threads are needed. Divergence can be avoided by using inputs with a size that is a multiple of 32.
- How many cores are utilized? Only 192 cores are utilized since a block (all of its warps) gets mapped to one SM.

2 Outline

2 Goal

Part 1 – warps

- Implement a kernel that inverts an image (calculate $x = 255 - x$, where $x \in (0, 255)$, for each color component – RGB).
- Assess how the number of threads (full/partial warps) influences the computation time.

Part 2 – thread divergence

- Augment the inversion kernel, so that the R color component is calculated as $x = (x \,\%\, 25) \cdot 10$, $x \in (100, 200)$
- Assess how thread divergence impacts the computation time.

2 Some pointers - nonobligatory

- Implement the inverse kernel and use the provided image load/save function (Toledo $\rightarrow$ Docs) to invert an image. You can use as many threads as there are input elements.

- Implement striding in the above kernel $\rightarrow$ if difficult, first implement striding in a simpler kernel that handles a short 1D sequence, for example, using 12 threads, 16 elements (0, 1, . . . , 15), and adding 1 to each element.

- Check that the inversion kernel, with the added striding, generates a fully-inverted image even when using less threads than there are input data elements. For example, consider a ratio of 1:16 (threads:elements).

2 Some pointers - nonobligatory

- Invoke the inverse kernel from a loop, where you progressively use more threads. Time the kernel on every iteration (see cheat sheet).
- Get the average execution time, as the time may vary. For example, due to preemption by the OS.
- Note that on some setups, the GPU may take more time to finish when first invoked – you can avoid this by calling the function once before actually timing it.

2 Some pointers - nonobligatory

- Make a second kernel, where the R color component is processed according to the provided equation.
- Check that the returned image is indeed different.
- Now make a second kernel, which instead of processing `RGBRGB...RGB` operates on an input format of `RR...RGGG...GBB...B` (re-format the data correspondingly).
- Evaluate how the execution time changes with / without thread divergence.