Programming Parallel Computers

Jukka Suomela · Aalto University · ppc.cs.aalto.fi

Part 5C: GPU programming — conclusions

What if the threads of a warp try to do different things?

- How are these two claims compatible:
  - all threads of a warp work in a synchronous manner
  - kernel is arbitrary C++ code written from the perspective of a thread
- ***Then what happens if different threads of a warp try to do different things?***
- For example, what if different threads have different values of x:
  - `if (x < 123) { ... } else { ... }`
  - `for (int i = 0; i < x; ++i) { ... }`

What if the threads of a warp try to do different things?

```
...
if (x < 123) {
    ...
} else {
    ...
}
...
```

True for threads 0...15

Entire warp takes these steps, but threads 16...31 are disabled

Entire warp takes also these steps, but threads 0...15 are disabled

What if the threads of a warp try to do different things?

- You *can* write arbitrary C++ code in which different threads do completely different things, it will be executed correctly!

- But it may be very inefficient, e.g.:
  1. the warp follows what **thread 0** does (threads 1, 2, 3, …, 31 disabled)
  2. the warp follows what **thread 1** does (threads 0, 2, 3, …, 31 disabled)
  3. …

- ***You can lose in performance by a factor of 32 if you don’t keep in mind that the entire warp is executed synchronously***

Compilation process and GPU assembly language

- C++ → PTX → SASS
  - PTX: platform independent intermediate language
  - SASS: what the GPU runs
- You can use `cuobjdump --dump-sass` to show the SASS code

Block-wide vs. warp-wide communication

- **Communication between the threads of a block:**
  - allocate shared memory with `__shared__`
  - read/write shared memory
  - synchronize with `__syncthreads()`
- **Communication between the threads of a warp:**
  - call e.g. functions `__shfl_sync()` or `__shfl()`
  - see *CUDA C++ Programming Guide*

GPU programming recap

- **You need to explicitly say what the GPU should run**
  - write a ***kernel***, specify how many ***blocks*** of threads you want, specify how many ***threads*** there are per block, launch the kernel
- **All threads will run the same kernel code**
  - in the kernel you can use the ***thread index*** and ***block index*** to decide what to do
- **GPU-side code accesses only *GPU memory***
  - you need to use CUDA functions to move data between CPU memory and GPU memory

GPU programming recap

- **Threads are organized in *warps* of 32 threads**
  - all threads of a warp are always synchronized
  - pay attention to memory access pattern
- **Threads are organized in *blocks* of *x* threads**
  - threads of a block can use shared memory for communication
- **GPU executes instructions in a linear order**
  - only looks at the next instruction in each active warp
  - ***good to have lots of active warps***
  - number of active warps limited by register & shared memory usage

What you learned earlier still applies

- It is always a good idea to try to ***minimize memory reads*** by reusing data in registers
- The same idea works both on CPUs and on GPUs
- See the course material for examples!