Programming Parallel Computers

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

Part 4A: GPU programming

GPU — graphics processing unit

- All modern computers have at least two processors:
  - **CPU**: what we have been using so far
  - **GPU**: lots of more computing power available and we can use it, too!
- **CPU** in our classroom computers [Intel Xeon E3-1230v5, “Skylake”]:
  - **460 billion** single-precision floating-point operations / second
  - **230 billion** double-precision floating-point operations / second
- **GPU** in our classroom computers [NVIDIA Quadro K2200, “Maxwell”]:
  - **1400 billion** single-precision floating-point operations / second
  - **45 billion** double-precision floating-point operations / second

Common feature: lots of wide, pipelined arithmetic units

CPU

- Pipelined arithmetic units:
  1 new operation per cycle
- ***Vector*** operations:
  8 similar operations
- Lots of arithmetic units:
  4 × 2 vector operations in parallel e.g. for FMA

GPU

- Pipelined arithmetic units:
  1 new operation per cycle
- "***Warp***" of "***threads***":
  32 similar operations
- Lots of arithmetic units:
  5 × 4 warps executed in parallel e.g. for FMA

Different tradeoffs between parallelism and clock frequency

CPU

64 single-precision arithmetic operations per cycle

3.4–3.8 GHz

FMA = **2** operations

460 billion arithmetic operations per second

GPU

640 single-precision arithmetic operations per cycle

1.0–1.1 GHz

FMA = **2** operations

1400 billion arithmetic operations per second

Differences in programming models

CPU

- ***“SIMD”*** — single instruction, multiple data
- ***One thread per core***

GPU

- ***“SIMT”*** — single instruction, multiple threads
- ***Very many threads***
  - threads organized in “blocks”
  - blocks consist of “warps”
  - **warp = 32 threads always works together**

Differences in programming models

CPU

- ***“SIMD”*** — single instruction, multiple data
- ***One thread per core***
  - each thread refers to ***vector registers***
  - each thread performs ***vector operations***

GPU

- ***“SIMT”*** — single instruction, multiple threads
- ***Very many threads***
  - each thread refers to ***scalar registers***
  - each thread performs ***scalar operations***

Differences in programming models

CPU

- *“SIMD”* — single instruction, multiple data
- ***One thread*** says e.g.:
  - compute vector sum
    z[i] = x[i] + y[i]
    for all i = 0…7

8 similar operations in parallel

GPU

- *“SIMT”* — single instruction, multiple threads
- ***All threads of a warp*** say e.g.:
  - compute scalar sum
    z = x + y

32 similar operations in parallel

Differences in programming models

CPU

- ***"SIMD"*** — single instruction, multiple data
- ***One thread*** says e.g.:
    - read one vector from memory and store it in vector variable x
    - can read 8 scalars in one step, ***have to be in consecutive memory locations***

GPU

- ***"SIMT"*** — single instruction, multiple threads
- ***All threads of a warp*** say e.g.:
    - read one scalar from memory and store it in scalar variable x
    - can read 32 scalars in one step, ***do not need to be consecutive memory locations***

Differences in programming models

CPU

- ***“SIMD”*** — single instruction, multiple data
- Some SIMT solutions are hard to implement here

GPU

- ***“SIMT”*** — single instruction, multiple thread
- Any SIMD solution easy to implement here (?)

Forms of parallelism

CPU

- Threads
- Vector operations

GPU

- **Threads — *lots of them!***
- ~~Vector operations~~

Forms of parallelism

CPU

- Threads
- Vector operations
- Instruction-level parallelism

GPU

- **Threads — *lots of them!***
- ~~Vector operations~~
- ***Instruction-level parallelism??***

How the hardware tries to keep pipelines busy

CPU

- Looks at the instruction stream ***far into the future***
  - tries to find some ***instruction*** that is ready for execution
  - ***instruction-level parallelism important***
  - small number of threads enough

GPU

- Only looks at the ***first instruction*** of each warp
  - tries to find some ***warp*** that is ready for execution
  - instruction-level parallelism not that important
  - ***large number of warps needed***

Forms of parallelism

CPU

- Threads
- Vector operations
- Instruction-level parallelism

GPU

- **Threads — *lots of them!***
- ~~Vector operations~~
- ~~Instruction-level parallelism~~

Different history, different hardware design

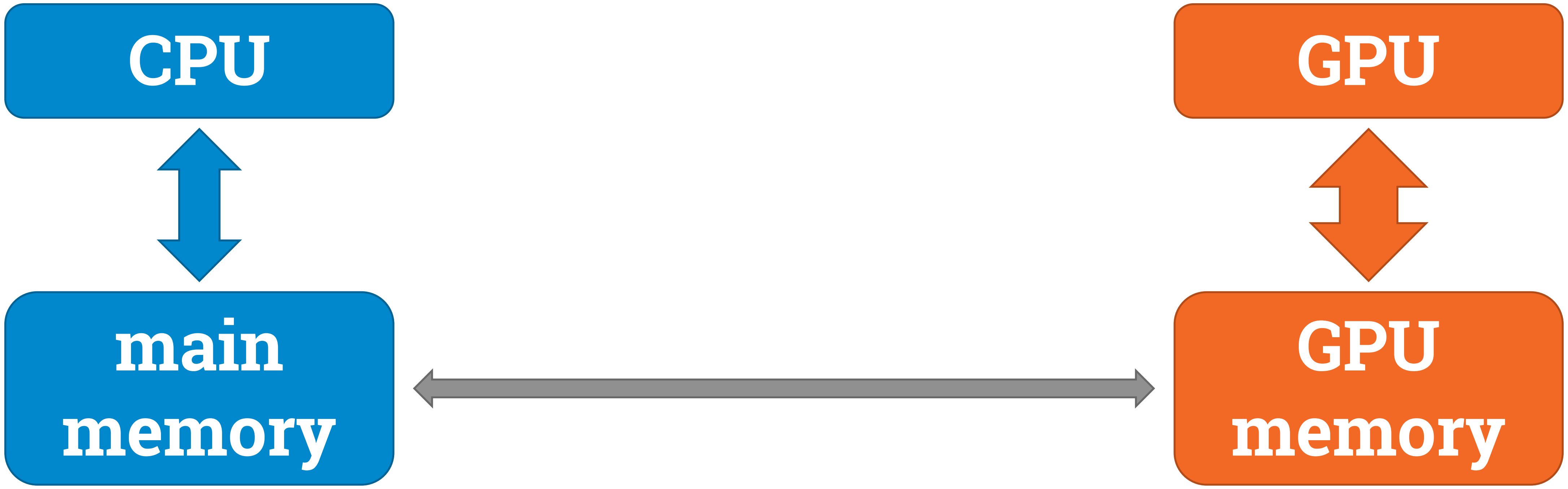
CPU

- Tries to run old sequential code reasonably well
- Lots of complicated hardware to support that
  - out-of-order execution
  - high clock frequency
  - many layers of cache

GPU

- Does not care anything about running old sequential code
- Simpler hardware
  - transistors used for arithmetic, not for control logic
  - easier to add more parallel units than increase clock speed

Hardware overview


CPU
GPU
main memory
GPU memory