Programming Parallel Computers

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

Part 6D: Conclusions

Computers are massively parallel

- **Huge amounts of computing power available**
  - CPUs: ***hundreds of billions*** of operations per second
  - GPUs: even more
- **All new performance comes from parallelism**
  - > ***factor 100 difference*** between sequential and parallel performance
- **Memory is slow**
  - > ***factor 50 difference*** between memory bandwidth and arithmetic throughput

Parallel computing resources

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

Programmer’s view

CPU

- ***Instruction-level parallelism important***
- Everything else is sequential unless explicitly parallelized

```
#pragma omp
float8_t
```

GPU

- Instruction-level parallelism not so important
- The ***only*** primitive that we can use is inherently parallel

```
f<<<blocks, threads>>>()
```

Key ideas

- Design algorithms so that there are lots of ***independent operations***
  - needed for **any kind of parallelism**
- Preferably lots of ***similar*** independent operations
  - needed for SIMD (vectors on CPUs)
  - needed for SIMT (warps on GPUs)
- Try to do ***lots of arithmetic operations per memory access***
  - otherwise the CPU will be mostly idle, **waiting for some data to process**

What is happening to hardware

- ***Wider vector units***
  - Intel CPUs with **AVX-512** already available
- ***GPU-like auxiliary processors***
  - Google's "Tensor Processing Unit":
    special hardware for **matrix multiplications**
- ***Low-precision floating-point numbers***
  - NVIDIA's "Tensor cores":
    4×4 matrix multiplication of **16-bit floats**

What is happening to hardware

- ***Transactional memory***
  - you can use memory a bit like transactional databases:
    - **begin transaction**
    - read and write memory (without any coordination)
    - try to **commit**
    - **rollback** if conflicts
  - some hardware support available in recent Intel CPUs

What next?

- **Practical path:**
  - computer architecture, computer hardware, compilers, programming languages, ***distributed computing, cloud computing, computer networks, internet protocols, mobile computing*** …
- **Theory path:**
  - algorithm design & analysis, computational complexity, parallel algorithms, concurrency theory, formal verification & synthesis, ***distributed algorithms*** …