Computational Model

Algorithms

Modeling Algorithms

- Algorithms can be implemented, but are not equal to an implementation
- Performance is always concrete
- We can only measure what is there
  - A **given** implementation of an algorithm
  - On a **given** platform
  - Under **given circumstances**

Modeling Algorithms

- Goal of algorithm design is not to invent well performing algorithms
  - Such a thing does not exist
  - But to develop algorithms that work well under a large variety of circumstances

RAM Model

- Classic Model
  - RAM Model
    - A machine consists of a CPU and RAM
      - CPU has a large number of registers
  - Unit costs for:
    - Moving data between RAM and CPU
    - Calculating between registers

RAM Model

- RAM Model is not accurate
  - Operations do not cost the same
    - Moving data from RAM to Cache (cache miss) can take 200 nsec
  - Simple operations take 20 nsec

RAM Model

- Operations are not sequential:
    - Intel 486DX: 0.336 instructions per clock cycle at 33 MHz = 11.1 Million Instructions per Second (MIPS)
    - AMD Ryzen 7 1800X: 84.6 instructions per clock cycle at 3.6 GHz = 304,510 MIPS
- Now: many instructions run in parallel and execution overlaps

RAM Model

- Data and instructions are cached in several cache levels
  - Caches belong exclusively to a chip
  - Core has own L1 / L2 caches
- Up till now:
  - Caches are coherent through invalidation
    - If one thread changes a cache content, other threads will not see the old content
    - Cache lines are invalidated and a read results in a cache miss

RAM Model

- Effectiveness of caches depends on the instructions and data
- Modern algorithm design:
  - Find cache aware / cache oblivious algorithms
    - Cache aware: Algorithm optimized depending on cache parameters
    - Cache oblivious: Algorithm does not need cache parameters in order to make efficient use of caches

RAM Model

- Threading
  - Many tasks can be performed in parallel
    - Processes can be broken into threads
      - Algorithms need to be thread-safe
        - Correct even when execution is split over several threads
- Usual tool is locking
  - But locking can be detrimental to performance
- Modern algorithms can be lock-free **and** threadsafe

RAM Model

- Branch prediction and speculative execution
  - Because cache misses are long
    - Processor will executes statements after a conditional statement
    - At the danger of these statements not being usable

Branch Prediction

Code

Block
if X go to A else go to B
Block A
Block B

Branch Prediction

Block
if X go to A else go to B

Block B

Execute B if X is predicted to be false

Speculative Execution

Block
if X go to A else go to B
Block A
Block B

Create two streams executing A and B in parallel, knowing that one stream’s result are thrown out

RAM Model

- Too many if statements and branch prediction and speculative execution become ineffective
- Good algorithms can be designed that minimize branches

RAM Model

- Large Data Sets
  - RAM is limited and expensive
    - This might change soon with Phase Change Memories as RAM substitutes
  - Some data does not fit into RAM
    - Performance becomes dominated by moving data from storage into RAM and back
  - Modern algorithms can be designed to work well with certain storage systems

RAM Model

- Distributed Computing
  - Many tasks are to massive to work on a single machine
  - Distribute computation over many nodes
  - Performance can now be dominated by the costs of moving data between machines and / or coordinating between them
  - **Distributed Algorithms**

RAM Model

- Parallel Computation
  - GPU have millions of simple processing elements
  - Modern CUDA algorithms will make use of parallelization
  - Successors to earlier parallel algorithms

RAM Model

- Despite it all:
  - RAM model has allowed us to develop a set of efficient algorithms
    - To which we still add
  - However: Software engineers and algorithm designers need to be aware of architecture

RAM Model

- Calculating timings
  - Can depend on data
    - Example: Sorting algorithm can run much faster on almost sorted data (or much worse)
  - Can calculate maximum time (pessimistic)
  - Can calculate expected time
    - Needs to make assumption on probabilities
  - Can calculate minimum time (optimistic)
    - Usually a useless measure

RAM Model

- Probabilistic algorithms
  - Algorithms can make decisions based on probabilities
    - Useful in case there is an "**adversary**" who gets to select data
  - Example:
    - Cryptography:
      - Can always break cryptography by guessing keys
      - But the probability of breaking cryptography with reasonable high probability in a limited amount of time should be very small