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 <u>thread-safe</u>
      - Correct even when execution is split over several threads
  - Usual tool is <u>locking</u>
    - But locking can be detrimental to performance
  - Modern algorithms can be <u>lock-free **and** threadsafe</u>

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

Algorithm Evaluation

- Program solve **instances** of a problem
  - Good algorithms scale well as instances become large
- Clients are only interested how fast a given instance of a given size is solved
- Algorithm designers are interested in designing algorithms that work well independent of the size of the instance

Algorithm Evaluation

- Evaluate performance by giving maximum or expected run time of a program on an instance size n
  - Gives a function $\phi(n)$
  - Interested in asymptotic behavior

Algorithm Evaluation

- Example: Compare $n^2,\ 0.1n^3,\ 0.01\cdot 2^n$ for $n = 0, 100, 200, \ldots, 1000$

n	n**2	0.1n**3	0.01 2**n
0	0.000000e+00	0.000000e+00	1.000000e-02
100	1.000000e+04	1.000000e+05	1.267651e+28
200	4.000000e+04	8.000000e+05	1.606938e+58
300	9.000000e+04	2.700000e+06	2.037036e+88
400	1.600000e+05	6.400000e+06	2.582250e+118
500	2.500000e+05	1.250000e+07	3.273391e+148
600	3.600000e+05	2.160000e+07	4.149516e+178
700	4.900000e+05	3.430000e+07	5.260136e+208
800	6.400000e+05	5.120000e+07	6.668014e+238
900	8.100000e+05	7.290000e+07	8.452712e+268
1000	1.000000e+06	1.000000e+08	1.071509e+299

Asymptotic Growth

- To compare the growth use Landau's notation
  - Informally
    - **Big O:** $f(n) = O(g(n))$ means f grows slower or equally fast than g
    - **Little O:** $f(n) = o(g(n))$ means f grows slower or than g
    - **Theta:** $f(n) = \Theta(g(n))$ means f and g grow equally fast
    - **Omega:** $f(n) = \Omega(g(n))$ means f grows faster than g

Landau Notation

- Exact definitions
  - Little o:

$$f(n) = o(g(n)) \Leftrightarrow \lim_{n \to \infty} \frac{f(n)}{g(n)} = 0$$

Landau Notation

- Exact definitions
  - Big O:

$$f(n) = O(g(n)) \Leftrightarrow \exists c > 0\ \exists n_0 > 0\ \forall n \in \mathbb{N}, n > n_0 : |f(n)| \leq cg(n)$$

Landau Notation

- Exact definitions
  - Θ:

$$f(n) = O(g(n)) \Leftrightarrow \exists c_0 > 0\ \exists c_1 > 0\ \exists n_0 > 0\ \forall n \in \mathbb{N}, n > n_0 : c_0 g(n) < f(n) \leq c_1 g(n)$$

Landau Notation

- Exact definitions
  - Ω:

$$f(n) = \Omega(g(n)) \Leftrightarrow \exists c_1 > 0 \ \exists n_0 > 0 \ \forall n \in \mathbb{N}, n > n_0 : |f(n)| \geq c_1 g(n)$$

Landau Notation

- In general, we only look at positive functions
- For analytic functions (complex differentiable), there are easier ways to determine the relationship between functions

Example

- Use the definition to show that
  $2n^2 + 4n + 5 = O(n^2)$ for $n \to \infty$

Example

- $2n^2 + 4n + 5 \leq 2n^2 + 4n^2 + 5n^2$ if $n \geq 1$
- $2n^2 + 4n + 5 \leq 11n^2$ if $n \geq 1$
- Pick $c_0 = 12$ and $n_0 = 1$ and find that
  - $\forall n > n_0 2n^2 + 4n + 5 < 12 \cdot n^2$
- Therefore $2n^2 + 4n + 5 = O(n^2)$ for $n \rightarrow \infty$

- Notice that we did not care about the exact constants

Some Useful Theorems

- Assume from now on that all functions f are positive
  - $\forall n \in \mathbb{N} : f(n) > 0$
- We also assume that the functions are analytic
  - Differentiable as complex functions (almost everywhere)
  - This includes all major functions used in engineering
  - Implies that they are infinitely often differentiable (almost everywhere)

Some Useful Theorems

- Assume $\lim_{n\to\infty} \frac{f(n)}{g(n)} = a > 0$
  - (this means that we also assume that the limit exists)
- Then: $f(n) = \Theta(g(n))$ for $n \to \infty$

Some Useful Theorems

- Proof:
  - $\lim_{n\to\infty} \frac{f(n)}{g(n)} = a > 0$
  - $\Rightarrow \forall \epsilon > 0 \; \exists \delta > 0 \forall n > 1/\delta \;:\; |\frac{f(n)}{g(n)} - a| < \epsilon$
    - Definition of the limit
  - $\Rightarrow \forall \epsilon > 0 \; \exists \delta > 0 \forall n > 1/\delta \;:\; a - \epsilon < \frac{f(n)}{g(n)} < a + \epsilon$

Some Useful Theorems

- Now we select one particular $\epsilon > 0$, namely $\epsilon = a/2$.
- For this selection, we have
- $$\exists \delta > 0 \forall n > 1/\delta \;:\; a/2 < \frac{f(n)}{g(n)} < (3/2)a$$
- We also set $n_0 = \lceil 1/\delta \rceil$
  - $$\forall n > n_0 \;:\; a/2 < \frac{f(n)}{g(n)} < (3/2)a$$
- Now we have
  - $$\forall n > n_0 : \frac{a}{2} g(n) < f(n) < \frac{3a}{2} g(n)$$
- Thus by definition: $f(n) = \Theta(g(n))$

Some Useful Theorems

- $f(n) = o(g(n))$ implies $f(n) = O(g(n))$

Proof:

$f(n) = o(g(n))$ implies

$$\lim_{n \to \infty} \frac{f(n)}{g(n)} = 0,$$

which implies $\forall \epsilon > 0 \; \exists \delta > 0 \; \forall n > \frac{1}{\delta} \; : \; \frac{f(n)}{g(n)} < \epsilon$

Some Useful Theorems

We select $\epsilon = 1$, which implies

$$\exists \delta > 0 \ \forall n > \frac{1}{\delta} \ : \frac{f(n)}{g(n)} < 1$$

We select $n_0 = \lceil \frac{1}{\delta} \rceil$ and obtain

$$\forall n > n_0 \ : \frac{f(n)}{g(n)} < 1$$

which implies

$$\forall n > n_0 : f(n) < g(n)$$, i.e.

$$f(n) = O(g(n)$$

Some Useful Theorems

- $\lim_{n \to \infty} \frac{f(n)}{g(n)} = \infty$ implies $f(n) = \Omega(g(n))$

- Proof is homework

Examples

- Relationship between $\log(n)$ and n?
- Evaluate the asymptotic behavior of $\frac{\log n}{n}$.
- The limit is of type $\frac{\infty}{\infty}$, so we use the theorem of L'Hôpital
- Take the derivatives of denominator and numerator
- Obtain $\frac{\frac{1}{n}}{1} = \frac{1}{n}$.
- Because $\lim_{n\to\infty} \frac{1}{n} = 0$, we have $\lim_{n\to\infty} \frac{\log n}{n} = 0$ and $\log(n) = o(n)$

Examples

- Relationship between 2^n and 3^n?
- $\lim_{n\to\infty} \frac{2^n}{3^n} = \lim_{n\to\infty} (\frac{2}{3})^n = 0$
- Therefore $2^n = o(3^n)$.