

Lecture 1

Background and Motivation

- Parallel computing is a part of HPC.
 - HPC also includes everything else that makes the computation fast.
 - No point parallelizing without increasing performance.
 - You might want to optimize for the architecture.
 - Sometimes overhead outweighs benefits from parallelization.
- Focusing on parallel algorithms.
 - Different version of parallel algorithms suits different architecture or models.
- Many application yo.
- People made super computers throughout the 1900s
- Super computers rely on carefully designed interconnects.
- Cloud computers are just AWS instances.
- Many aspects

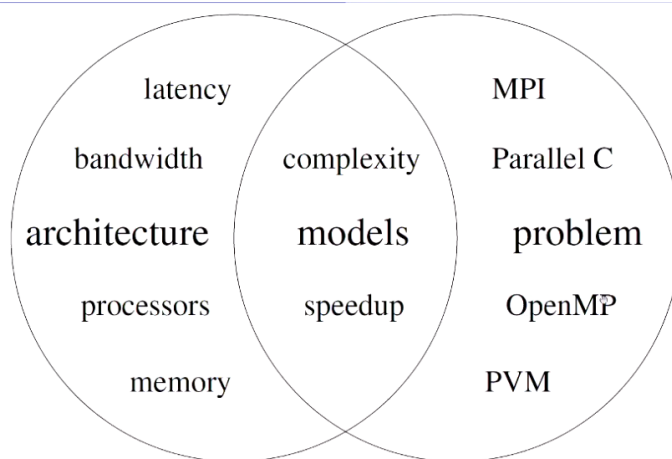


Figure: Overlapping aspects of parallel computing.

Complexity

- $f(n) = O(g(n)) \Rightarrow f$ grows no faster than g
- $f(n) = \Omega(g(n)) \Rightarrow f$ grows no slower than g
- $f(n) = o(g(n)) \Rightarrow f$ grows slower than g
- $f(n) = \omega(g(n)) \Rightarrow f$ grows faster than g
- $f(n) = \Omega(g(n)) \wedge f(n) = O(g(n)) \Rightarrow f(n) = \Theta(g(n))$
- Strictly speaking we should really use \in instead of $=$
- Some common name for complexities:
 - Constant
 - Logarithmic
 - Polylog: $(\log(n))^c$
 - Linearithmic: $n \log n$
 - Quadratic: n^2
 - Polynomial or geometric
 - Exponential
 - Factorial
- Log factor are often ignored.

Model

- RAM model: *random access machine*
 - Common model when we talk about sequential time complexity.
- Multiplying the number of computers by a constant factor doesn't change the complexity.
 - Solution: allow p , the number of processors to increase with problem size and hence reduces the complexity.

PRAM

- Parallel Random Access Machine
- p number of RAM processors, each have private memory and share a large shared memory, all memory access takes the same amount of time.
- Does things synchronously, AKA in lock steps.
- PRAM pseudo code looks like regular pseudo code but there's this
for $i \leftarrow 0$ to $n - 1$ do in parallel
processor i does thingy

Many different PRAM model

- EREW: exclusive read, exclusive write
- CREW: concurrent read, exclusive write
- CRCW: concurrent read, concurrent write
 - Concurrent write have different types
 - COMMON: Error when two processor tries to write to the same location with different value.
 - ARBITRARY: Pick a arbitrary processor if many processor writes the same time.
 - PRIORITY: Processor with lowest ID writes.
 - COMBINING: Runs a function whenever multiple processors tries to write at the same time.
 - Too powerful.
- ERCW: exclusive read, concurrent write (never used)

Power of model: expresses the set of all problems that can be solved within a certain complexity.

- A is more powerful than B if A can solve a larger set of problems within any complexities.
- A is equally powerful as B if they can solve the same set problems within any complexities.
- Partial ordering.
- COMMON, ARBITRARY, PRIORITY and COMBINING are in increasing order of power.
- Any CRCW PRIORITY PRAM can be simulated by a EREW PRAM with a complexity increase of $\mathcal{O}(\log p)$

- *Parallel Computation Thesis*: any thing can be solved with a Turing Machine with polynomially bounded space can be solved in polynomially bounded space with unlimited processors.
 - Unbounded *word sizes* are not useful, so we limit word counts to $\mathcal{O}(\log p)$
- *Nick's Class* (NC): Solvable in polylog time with poly number of processors.
- Widely believed that $\mathbf{NP} \neq \mathbf{P}$

Definitions (need to remember)

- $w(n) = t(n) \times p(n)$ where $w(n)$ is the work / cost, $t(n)$ is the time and $p(n)$ is the number of processors.
- Optimal processor allocation means: $t(n) \times p(n) = \Theta(T(n))$ where $T(n)$ is the time taking by a sequential algorithm.
 - Equivalent to $t(n) \times p(n) = O(T(n))$ because $t(n) \times p(n) = \Omega(T(n))$ always.
- $\text{Speedup}(n) = \frac{T(n)}{t(n)}$
 - Speedup optimal = processor optimal.
- Optimal: processor optimal AND $t(n) = \mathcal{O}(\log^k n)$
 - Processor optimal and polylog in time.
- Efficient: Assume $T(n) = \Omega(n)$ $w(n) = \mathcal{O}(T(n) \log^\alpha n)$ AND polylog in time
- **size**: $\text{Size}(n)$ is the total number of operations it does.
- **efficiency**: $\eta(n)$ speedup per processor
 - $\eta(n) = \frac{T(n)}{w(n)} = \frac{\text{Speedup}(n)}{p(n)}$
- You can decrease p and increase t by a factor of $O\left(\frac{p_1}{p_2}\right)$, $w(n)$ doesn't increase its complexity.
 - Can't do it the other way around.

Brent's Principle (important)

- If something can be done with size x and t time with infinite processors, then it can be done in $t + \frac{x-t}{p}$ time with p processors

Amdahl's Law

- Maximum speedup: if f is the fraction of time that can't be parallelized, then $\text{Speedup}(p) \rightarrow \frac{1}{f}$ as $p \rightarrow \infty$
 - Honestly very obvious.

Gustafson's Law

- s is fraction time of serial part, r is fraction time of parallel part, then $\text{Speedup}(p) = \Omega(p)$
 - Very obvious again...