Super Algorithms for Supercomputers

Lecture 3 : Parallel Performance

08/27/2021

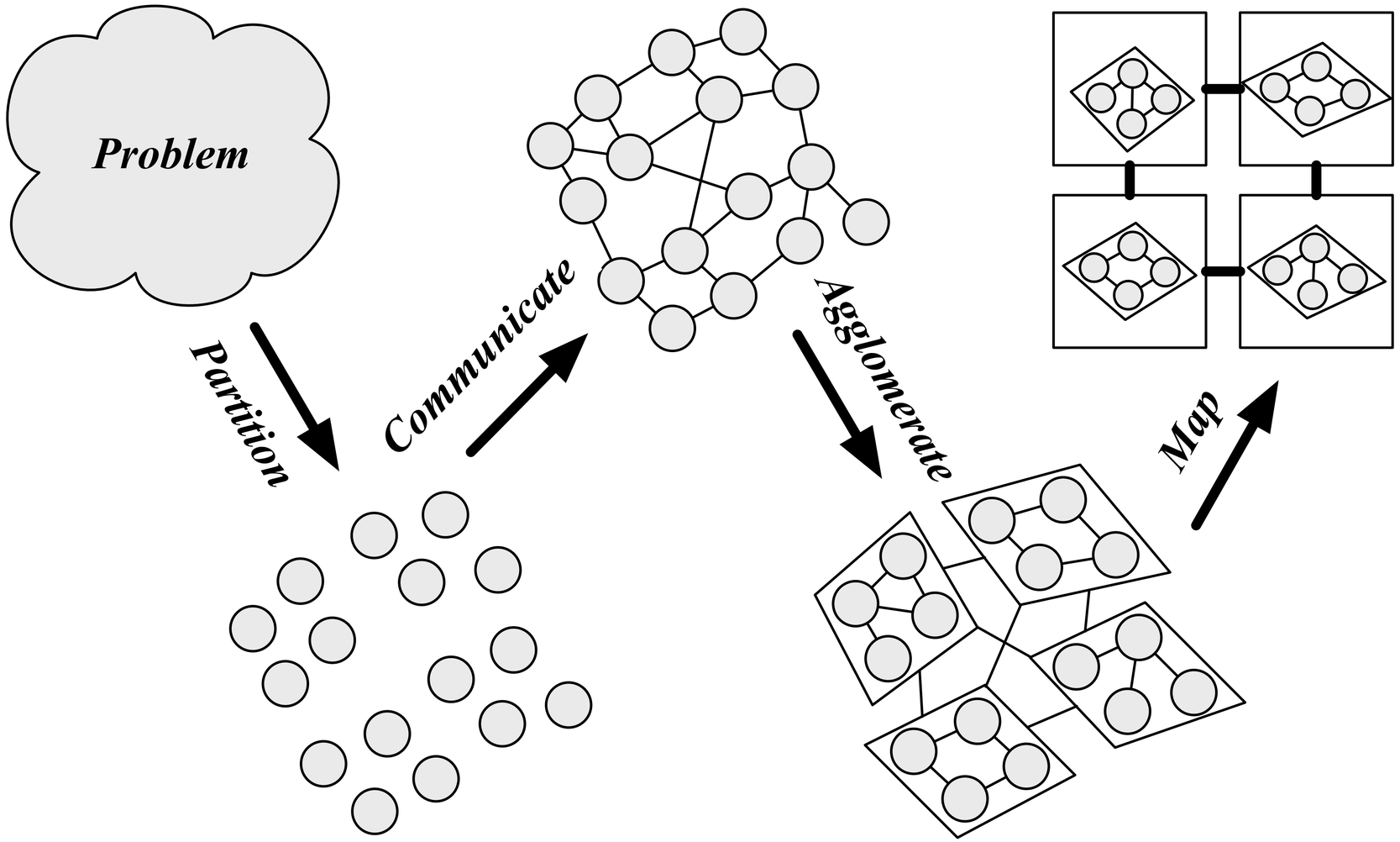
Professor Amanda Bienz

https://courses.grainger.illinois.edu/cs554/fa2015/notes/04_performance.pdf

Today's Learning Objectives

- What is efficiency? Scalability?
- What makes an algorithm scalable?
- What is the difference between strong scaling and weak scaling?

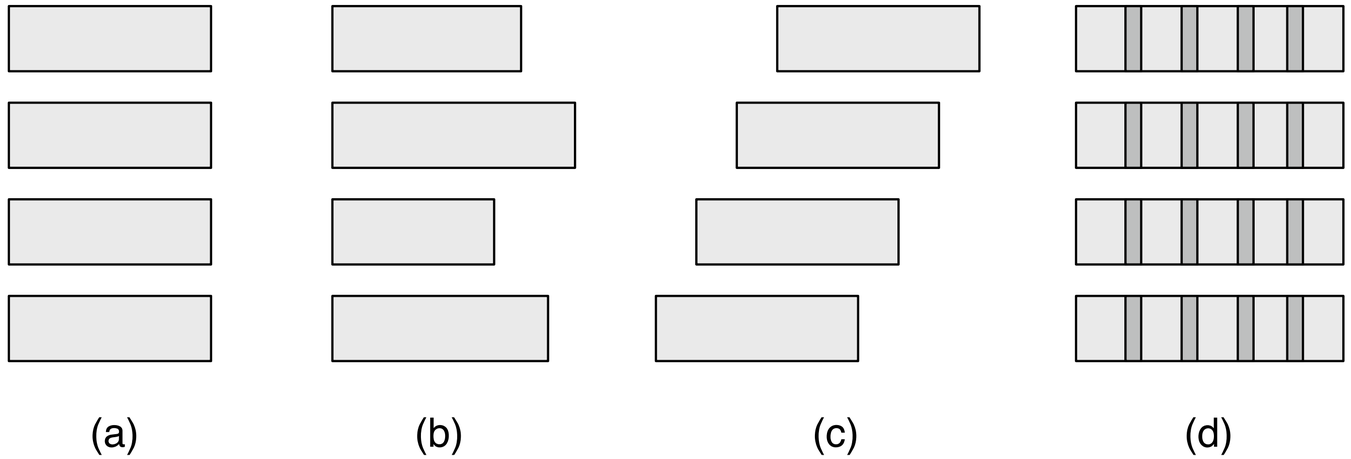
Review : Parallel Algorithm Design



Parallel Efficiency

- **Efficiency :** effectiveness of parallel algorithm relative to serial counterpart
  - **Load balance :** distribution of work among processors
  - **Concurrency :** processors working simultaneously
  - **Overhead :** additional work not present in serial computation

Parallel Efficiency



- What is load balance, concurrency, and overhead for each?

Definitions

- **Memory (M):** amount of storage required
- **Work (W):** number of operations (flops, loads, stores) required
- **Velocity (V) :** number of operations per unit time performed by one processor
- **Time (T) :** time (e.g. seconds) required from beginning to end of computation
- **Cost (C) :** number of processors times execution time

Definitions

- Can assume memory requirement of application across p processes is equivalent to memory on one process ($M_p = M_1$)… can just refer to this as M

- **Parallel Overhead of P processes (O_p) :** difference between work required by p processes - work by one process ($W_p - W_1$)

Definitions

- Can say processor speed is dependent on amount of memory used
- V(M) : processor speed for some memory requirement M
- Can assume $V_p(M) = V_1(M)$
- However, $V(M) \neq V(N)$ if $M \neq N$
- $p \cdot V\left(\frac{M}{p}\right)$: aggregate speed of p processors

Parallel Cost

- Execution time : total work / aggregate speed
- Cost : number of processors / execution time
- Efficiency : serial cost / parallel cost
- Speedup : serial time / parallel time

Speedup : Superlinear

- Mathematically, when can speedup be superlinear?

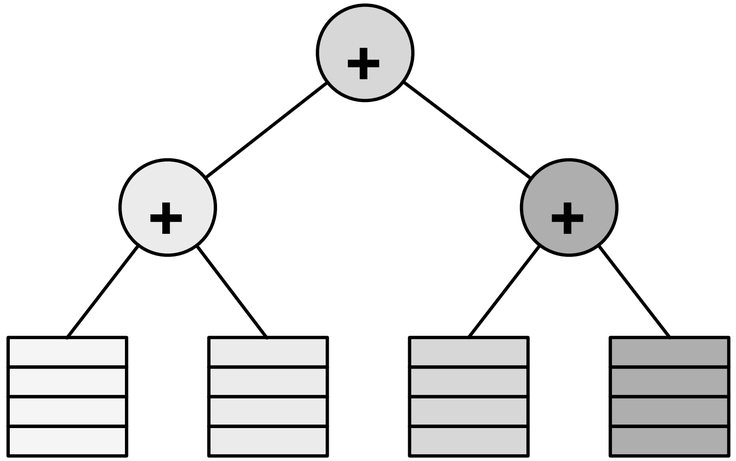
Speedup : Superlinear

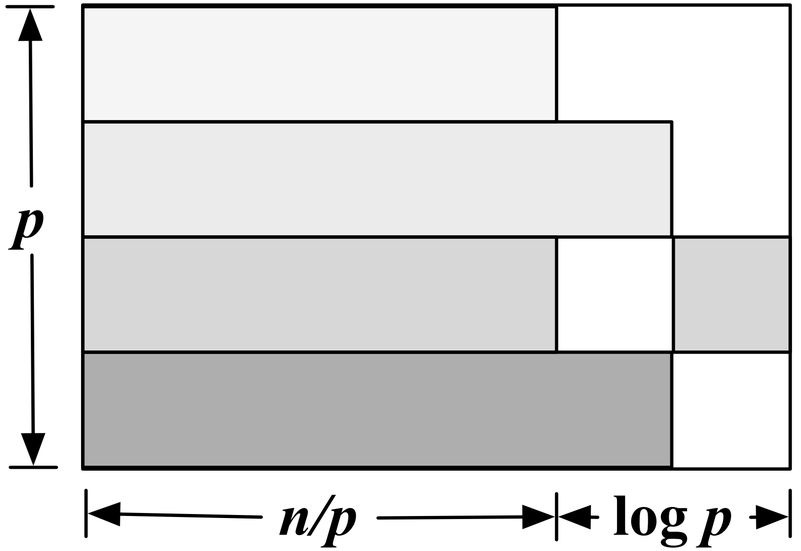
- Mathematically, when can speedup be superlinear?
- This happens in practice is M does not fit in cache, while M/p does
- For simplicity, we will assume processor speed is constant

Example : Summing n numbers

- Using p processes, each process computes the first n/p summations

- Can use a reduction to gather final sum





- What is the efficiency? Speedup?

Parallel Scalability

- **Scalability :** how effectively can parallel algorithm utilize additional processors?
  - If I add more processes, does my parallel algorithm take less time?
- Algorithm is considered *scalable* if efficiency is bounded away from zero as number of processors grows without bound ($E_p = \mathcal{O}(1)$ as $p \to \infty$)
- Algorithmic scalability may be impractical if growth rate of problem size causes total execution time to grow unacceptably

Parallel Scalability

- Could use more processes to :
  - Solve given problem in less time (strong scaling)
  - Solve larger problem in (ideally) same time (weak scaling)
  - Obtain sufficient memory to solve given or larger problem
- Large problems require more memory and total work (W_1)

Problem Scaling

- Problem size : what is the growth rate required to keep some quantity constant as number of processors increases:
  - Serial work : $W_1 = \mathcal{O}(1)$ **(strong scaling)**
  - Serial work per processor : $W_1 = \mathcal{O}(p)$ **(weak scaling)**
  - Execution time : $T_p = \mathcal{O}(1)$
  - Memory per processor : $M = \mathcal{O}(p)$

Strong Scaling

- Fixed serial work requirement
- Goal : use more processors to reduce execution time
  - **Strong scaling limit :** limit to number of processors you can use
- By definition of *scalable*, are algorithms scalable for fixed problem?

Amdahl's Law

- Theoretical speed up is limited by serial portion of algorithm
- Some fraction $0 \leq s \leq 1$ is serial
- Remaining $1 - s$ is $p-$fold parallel
- $E_p \to 0$ and $S_p \to \frac{1}{s}$ as $p \to \infty$

Weak Scaling

- Fixed serial work per processor
- Not natural for applications, but useful for studying scalability
- Ideally, would like to see constant execution time when increasing problem size p times
- When is an algorithm weakly scalable (mathematically)?

Fixed Execution Time

- Maybe there is a strict time limit (such as real world constraints)
- With fixed execution time, algorithm cannot be scalable unless both memory M and work W_p grow at most linearly with p

Fixed Memory Per Processor

- Some problems may use all available memory
  - Grows linearly with p in distributed memory computers
- If work grows linearly with memory : same as fixed serial work per processor example
- If work grows faster than linearly with memory, execution time grows superlinearly with p : algorithm may be impractical (too much work)

Fixed Efficiency

- Determine minimum growth rate in problem size required to maintain constant efficiency
- If fixed efficiency is possible, algorithm is scalable
  - May be impractical (large growth in problem size required)
- Growth rate in problem size determines degree to which an algorithm is scalable

Isoefficiency Function

- **Isoefficiency function :** $W_1(n(p))$
  - $W_1(n)$: Express W_1 in terms of n (work is dependent on problem size)
  - $W_p(n,p)$: Express W_p in terms of both n and p (parallel work dependent on both problem size and number of processes)
  - For constant efficiency, $W_1(n) - EW_p(n,p) = 0$
- For given number of processors, minimum problem size (order of magnitude) required to maintain constant efficiency E

Isoefficiency and Scalability

- Execution time T_p is constant if isoefficiency function is $\mathcal{O}(p)$, but otherwise grows with p
- Growth rate of T_p may be unacceptable
- Isoefficiency $\mathcal{O}(p)$ is desirable but may be unattainable for many problems
- More achievable isoefficiency $\mathcal{O}(p \log p)$ or $\mathcal{O}(p\sqrt{p})$
- Algorithm with isoefficiency $\mathcal{O}(p^2)$ or higher has poor scalability since T_p would grow at least linearly with p

Modeling Parallel Work

- Categories of parallel work : computation, communication, and idle time
  - $W_p = W_{comp} + W_{comm} + W_{idle}$
  - W_{comp}: serial work W_1 plus any additional computational work required for parallel execution
  - W_{comm}: time spent sending and receiving messages
  - W_{idle} : time spent waiting due to lake of work or lack of data

Reducing Idle Time

- Idle time due to lack of work : improve load balancing
- Idle time due to lack of data : overlap communication and computation
- Could have more than one process (or thread) per processor… processor will stay active even when one process has no work or data (**multithreading**)

Today's Learning Objectives

- What is efficiency? Scalability?
- What makes an algorithm scalable?
- What is the difference between strong scaling and weak scaling?