CS-251: Parallel and Distributed Computing

Lecture 03 – Speedup and Amdahl's Law

Dr. Zeeshan Rafi

PhD MIS, MPhil IT, BS Software Engineering
Former Software Engineer, Database Administrator
System Analyst, Big Data Analyst
Member Turkish Scientific & Technological Research Council

Department of Computing and Information Technology

Istanbul University, TR

KHAS University, TR

University of Gujrat, PK

GCF University, Pk

Speedup and Amdahl's Law

Concurrency/Granularity

- One key to efficient parallel programming is *concurrency.*
- For parallel tasks we talk about the *granularity* – size of the computation between *synchronization points*
  - Coarse – heavyweight processes + IPC (interprocess communication (PVM, MPI, …)
  - Fine – Instruction level (eg. SIMD)
  - Medium – Threads + [message passing + shared memory synch]

One measurement of granularity

- Computation to Communication Ratio
  - (Computation time)/(Communication time)
  - *Increasing* this ratio is often a key to good efficiency
  - How does this measure granularity?
    - ↑ CCR = ? _____ Grain
    - ↓ CCR = ? _____ Grain

Communication Overhead

- Another important metric is *communication overhead* – time (measured in instructions) a zero-byte message consumes in a process
  - Measure time spent on communication that *cannot* be spent on computation
- *Overlapped Messages* – portion of message lifetime that can occur concurrently with computation
  - time bits are on wire
  - time bits are in the switch or NIC

Many little things add up …

- Lots of little things add up that ***add overhead*** to a parallel program
  - Efficient implementations demand
    - Overlapping (aka hiding) the overheads as much as possible
    - Keeping non-overlapping overheads as small as possible

Speed-Up

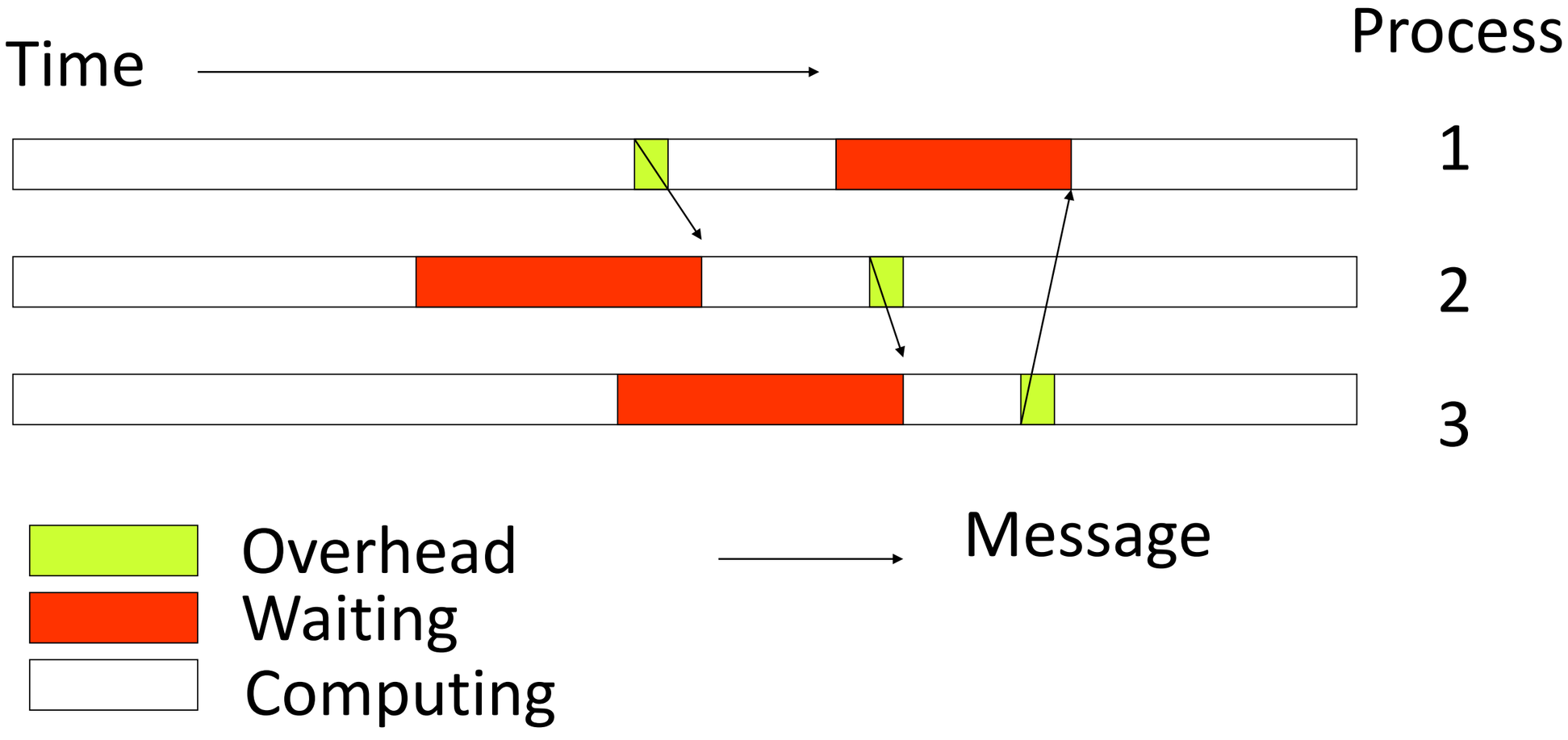
- S(n) =
  - (Execution time on Single CPU)/(Execution on N parallel processors)
  - t_s / t_p
  - Serial time is for ***best*** *serial algorithm*
    - This may be a different algorithm than a parallel version
      - Divide-and-conquer Quicksort O(NlogN) vs. Mergesort

Linear and Superlinear Speedup

- **Linear speedup = N,** for N processors
  - Parallel program is perfectly scalable
  - Rarely achieved in practice
- *Superlinear* Speedup
  - S(N) > N for N processors
    - Theoretically not possible
    - How is this achievable on real machines?
      - Think about physical resources of N processors

Space-Time Diagrams

- Shows comm. patterns/dependencies
- XPVM has a nice view.



What is the Maximum Speedup?

- *f* = fraction of computation (algorithm) that is serial and *cannot be parallelized*
  - Data setup
  - Reading/writing to a single disk file
- $t_s = f t_s + (1-f)\, t_s$
  = serial portion + parallelizable portion
- $t_p = f t_s + ((1-f)\, t_s)/n$
- $S(n) = t_s/(f t_s + ((1-f)\, t_s)/n)$
  $= n/(1 + (n-1)f)$ <- Amdahl's Law

Limit as n -> ∞ = 1/f

Amdahl’s law

- Amdahl’s Law implies that the overall performance improvement is limited by the range of impact when an optimization is applied. Consider the following equations:
- Improvement rate N = (original execution time) / (new execution time)
- New execution time = timeunaffected + timeaffected / (Improvement Factor)
- Time = (instruction count) × CPI × CCT

Amdahl’s law

- If a program currently takes 100 seconds to execute and loads and stores account for 20% of the execution times, how long will the program take if loads and stores are made 30% faster? For this, you can use Amdahl's law or you can reason it out step by step. Doing it step by step gives (1) Before the improvement loads take 20 seconds (2) If loads and stores are made 30 percent faster they will take 20/1.3 = 15.385 seconds, which corresponds to 4.615 seconds less. (3) Thus, the final program will take 100 - 4.615 = 95.38

- CPI = (execution_time x clock_rate)/instructions
- (cpu_ex_time_B/cpu_ex_time_A) – formula to know which is faster

Amdahl’s law

- **Amdahl's law**, named after computer architect Gene Amdahl, is used to find the maximum expected improvement to an overall system when only part of the system is improved.
- Amdhal's law can be interpreted more technically, but in simplest terms it means that it is the algorithm that decides the speedup not the number of processors.

Wikipedia

Amdahl’s law

- Amdahl's law is a demonstration of the law of diminishing returns: while one could speed up part of a computer a hundred-fold or more, if the improvement only affects 12% of the overall task, the best the speedup could possibly be is

$$\frac{1}{1 - 0.12} = 1.136 \text{ times faster.}$$

Amdahl's law

- More technically, the law is concerned with the speedup achievable from an improvement to a computation that affects a proportion *P* of that computation where the improvement has a speedup of *S*.

Amdahl’s law

- For example, if an improvement can speedup 30% of the computation, *P* will be 0.3; if the improvement makes the portion affected twice as fast, *S* will be 2.
- Amdahl's law states that the overall speedup of applying the improvement will be

$$\frac{1}{(1-P)+\frac{P}{S}}$$

Amdahl’s law

- To see how this formula was derived, assume that the running time of the old computation was 1, for some unit of time. The running time of the new computation will be the length of time the unimproved fraction takes (which is 1 − *P*) plus the length of time the improved fraction takes. The length of time for the improved part of the computation is the length of the improved part's former running time divided by the speedup, making the length of time of the improved part *P*/*S*.

Amdahl's law

$$\frac{1}{(1-P) + \frac{P}{S}}$$

- The final speedup is computed by dividing the old running time by the new running time, which is what the above formula does.

Amdahl’s law

Here's another example. We are given a task which is split up into four parts: P1 = .11 or 11%, P2 = .18 or 18%, P3 = .23 or 23%, P4 = .48 or 48%, which add up to 100%. Then we say P1 is not sped up, so S1 = 1 or 100%, P2 is sped up 5x, so S2 = 5 or 500%, P3 is sped up 20x, so S3 = 20 or 2000%, and P4 is sped up 1.6x, so S4 = 1.6 or 160%.

Amdahl's law

- By using the formula = $\frac{P1}{S1} + \frac{P2}{S2} + \frac{P3}{S3} + \frac{P4}{S4}$

- We find the running time is = $\frac{.11}{1} + \frac{.18}{5} + \frac{.23}{20} + \frac{.48}{1.6} = .4575$

or a little less than 1/2 the original running time which we know is 1.

- Therefore the overall speed boost is = $\frac{1}{.4575} = 2.186$

  or a little more than double the original speed using the formula

$$\frac{1}{\frac{P1}{S1}+\frac{P2}{S2}+\frac{P3}{S3}+\frac{P4}{S4}}$$

Amdahl’s law

- Parallelization

In the special case of parallelization, Amdahl's law states that if *F* is the fraction of a calculation that is sequential (i.e. cannot benefit from parallelization), and (1 − *F*) is the fraction that can be parallelized, then the maximum speedup that can be achieved by using *N* processors is =

$$\frac{1}{F + (1 - F)/N}$$

Amdahl's law

- As an example, if *F* is only 10%, the problem can be sped up by only a maximum of a factor of 10, no matter how large the value of *N* used. For this reason, parallel computing is only useful for either small numbers of processors, or problems with very low values of *F*: so-called embarrassingly parallel problems. A great part of the craft of parallel programming consists of attempting to reduce *F* to the smallest possible value.

Amdahl’s law

- Where f is the fraction of the problem that must be computed sequentially and N is the number of processors.

$$S \leq \frac{1}{f + (1 - f)/N}$$

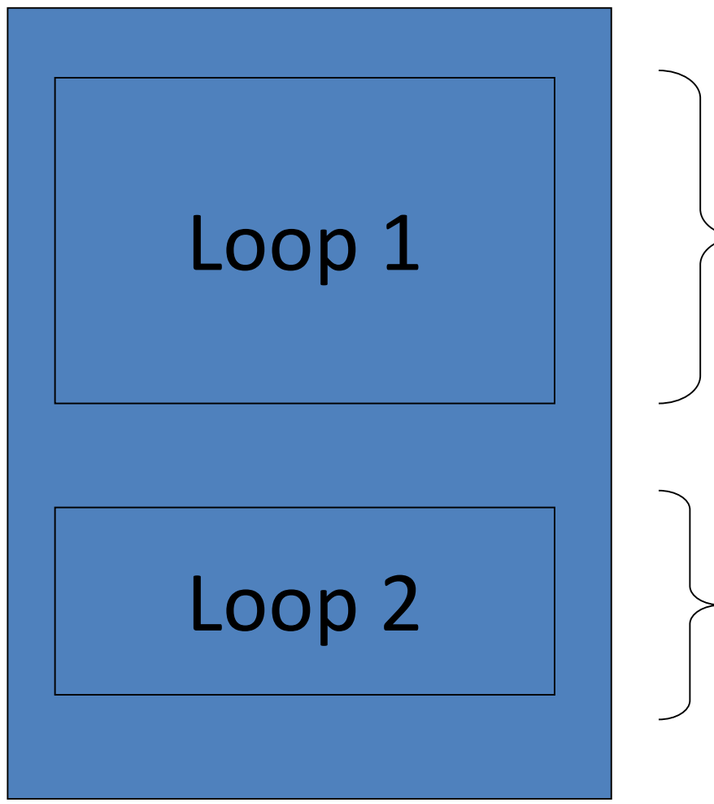
Amdahl's Law Recap

- *Amdahl's law states that the performance improvement to be gained from using some faster mode of execution is limited by the fraction of the time the faster mode can be used.*

Amdahl's Law

- *Consider the following example:*

Program



200 lines — 10% of total execution time

10 lines — 90% of total execution time

Amdahl's Law

- Overall speedup = $$S_{overall} = \frac{1}{(1-f) + \frac{f}{S_{part}}}$$

Where:

•S- overall is the max of how many times that parallel computing could be faster than single-thread performance.

•*f*- is the fraction of the algorithm that can be parallelized.

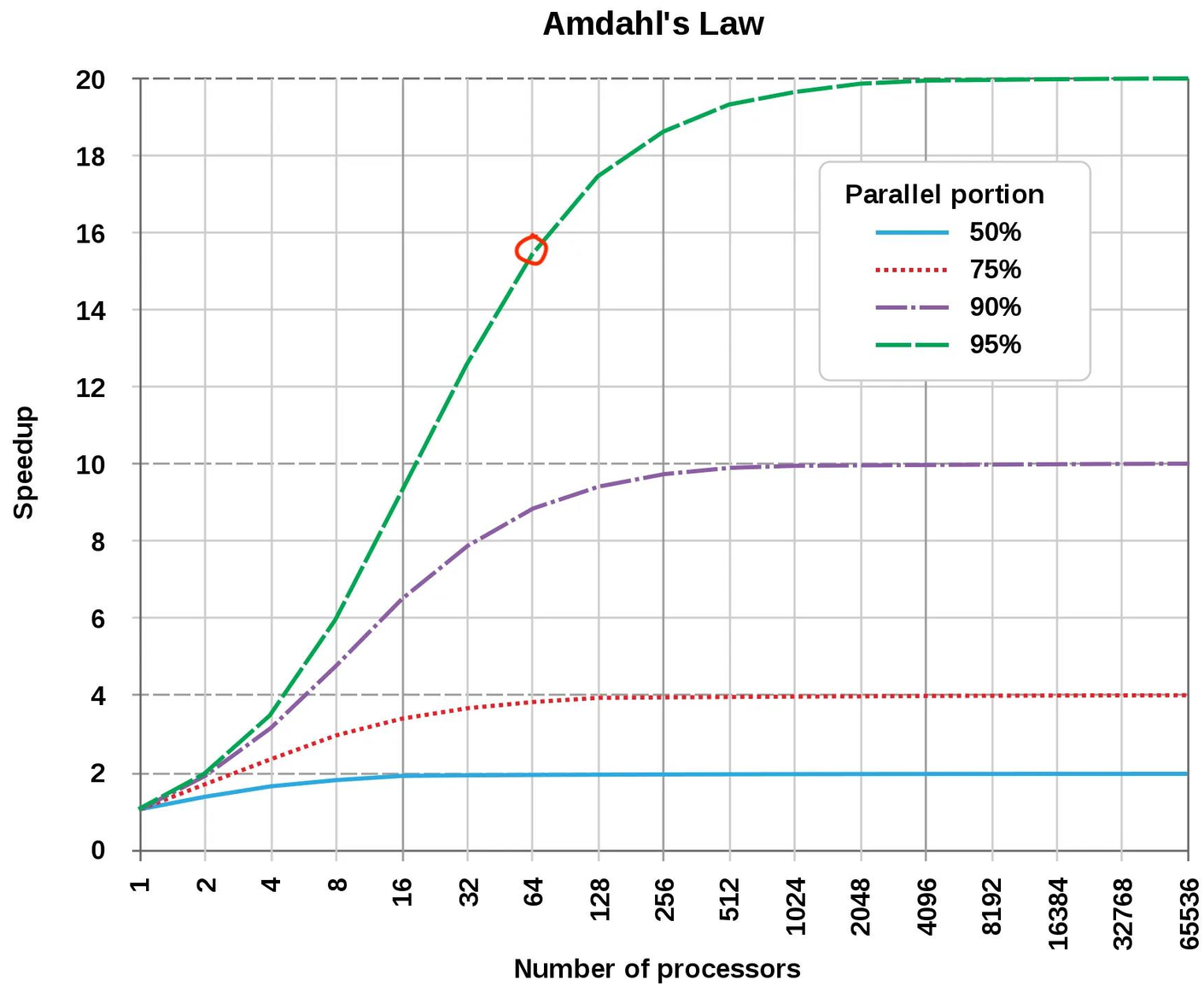
•S-part is the corresponding speedup.

For example: task A needs 20 hours to complete using a single thread. There is a 1-hour portion of the task that cannot be parallelized and the remaining 19 hours of execution time can be parallelized (*f* = 19/20 = 0.95).

When increasing the number of processors to S-part = 64 for parallel processing, the maximum speed up is **1/((1–0.95) + 0.95/64) = 15.42** times, compared to single processing.

More to think about …

- Amdahl's law works on a *fixed* problem size
  - This is reasonable if your only goal is to solve a problem faster.
  - What if you also want to solve a larger problem?
    - Gustafson's Law (Scaled Speedup)



Example of Amdahl's Law

- Suppose that a calculation has a 4% serial portion, what is the limit of speedup on 16 processors?
  - 16/(1 + (16 – 1)*.04) = 10
  - What is the maximum speedup?
    - 1/0.04 = 25

Gustafson's Law

- Fix execution of on a single processor as
  - $s + p$ = serial part + parallelizable part **= 1**
- S(n) = (s +p)/(s + p/n)

  = 1/(s + (1 – s)/n) = Amdahl's law
- Now let, 1 = s + π = execution time on a parallel computer, with π = parallel part.
  - $S_s(n) = (s + \pi\, n)/(s + \pi) = n + (1\text{-}n)s$

More on Gustafson's Law

- Derived by fixing the parallel execution time (Amdahl fixed the problem size -> fixed serial execution time)
  - For many practical situations, Gustafson's law makes more sense
    - Have a bigger computer, solve a bigger problem.
  - Amdahl's law turns out to be too conservative for high-performance computing.

Efficiency

- E(n) = S(n)/n * 100%
- A program with linear speedup is 100% efficient.

Example questions

- Given a (scaled) speed up of 20 on 32 processors, what is the serial fraction from Amdahl's law?, From Gustafson's Law?
- A program attains 89% efficiency with a serial fraction of 2%. Approximately how many processors are being used according to Amdahl's law?

Evaluation of Parallel Programs

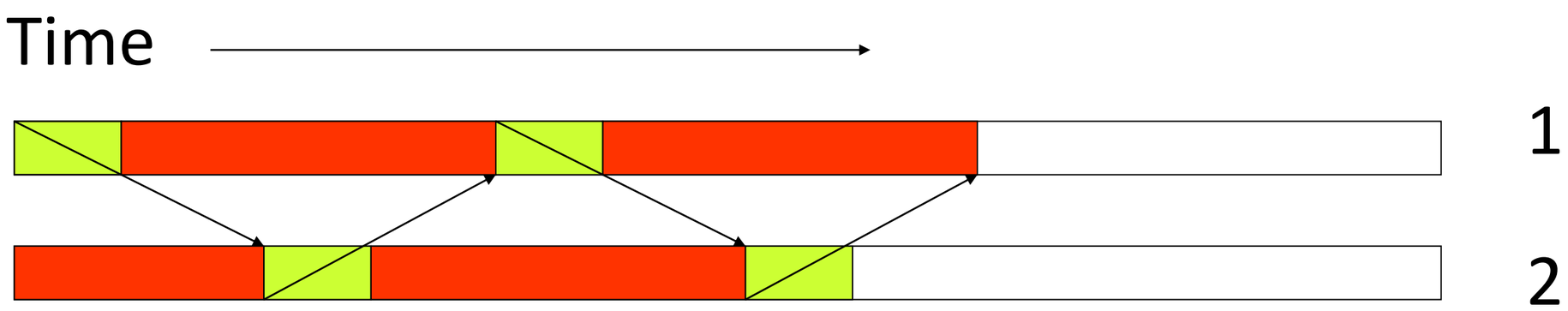
- Basic metrics
  - Bandwith
  - Latency
- Parallel metrics
  - Barrier speed
  - Broadcast/Multicast
  - Reductions (eg. Global sum, average, …)
  - Scatter speed

Bandwidth

- Various methods of measuring bandwidth
  - Ping-pong
    - Measure multiple roundtrips of message length L
    - BW = 2*L*<#trials>/t
  - Send + ACK
    - Send #trials messages of length L, wait for single ACKnowledgement
    - BW = L*<#trials>/t
- Is there a difference in what you are measuring?
- Simple model: $t_{comm} = t_{startup} + nt_{data}$

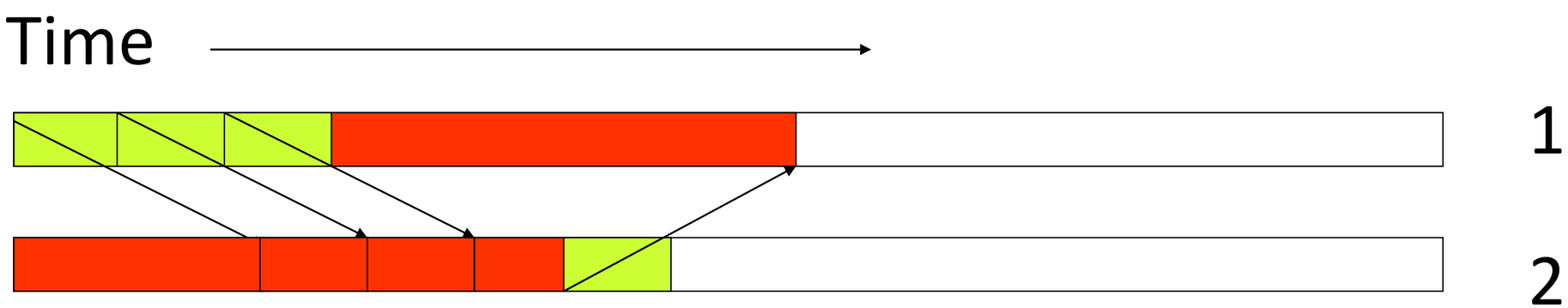
Ping-Pong

- All the overhead (including startup) are included in every message
- When message is very long, you get an accurate indication of bandwidth



Send + Ack

- Overheads of messages are masked
- Has the effect of decoupling *startup latency* from bandwidth (concurrency)
- More accurate with a large # of trials



In the limit …

- As messages get larger, both methods converge to the same number
- How does one measure latency?
  - Ping-pong over multiple trials
  - Latency = 2*<#ntrials>/t
- What things aren't being measured (or are being smeared by these two methods)?
  - Will talk about cost models and the start of LogP analysis next time.

How you measure performance

- It is important to understand exactly what you are measuring and how you are measuring it.

Exam …

- What topics/questions do you want reviewed?
- Are there specific questions on the homework/programming assignment that need talking about?

Questions

- Go over programming assignment #1
- Go over programming assignment #2