Unit 2

Speed-up, Amdahl's Law, Gustafson's Law, efficiency, basic performance metrics

Outline

- Speedup
- Superlinearity Issues
- Speedup Analysis
- Cost
- Efficiency
- Amdahl's Law
- Gustafson's Law (not the Gustafson-Baris's Law)
- Amdahl Effect

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

Speedup

- Speedup measures increase in running time due to parallelism. The number of PEs is given by n.
- Based on running times, $S(n) = t_s/t_p$, where
  - t_s is the execution time on a single processor, using the fastest known sequential algorithm
  - t_p is the execution time using a parallel processor.
- For theoretical analysis, $S(n) = t_s/t_p$ where
  - t_s is the worst case running time for of the fastest known sequential algorithm for the problem
  - t_p is the worst case running time of the parallel algorithm using n PEs.

- 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

Speedup in Simplest Terms

$$\text{Speedup} = \frac{\text{Sequential execution time}}{\text{Parallel execution time}}$$

- Quinn's notation for speedup is

  $\Psi(n,p)$

  for data size n and p processors.

Linear Speedup Usually Optimal

- Speedup is linear if $S(n) = \Theta(n)$
- Theorem: The maximum possible speedup for parallel computers with n PEs for "traditional problems" is *n*.
- Proof:
  - Assume a computation is partitioned perfectly into *n* processes of equal duration.
  - Assume no overhead is incurred as a result of this partitioning of the computation – (e.g., partitioning process, information passing, coordination of processes, etc),
  - Under these ideal conditions, the parallel computation will execute *n* times faster than the sequential computation.
  - The parallel running time is t_s/n.
  - Then the **parallel speedup of this computation** is

$$\boldsymbol{S(n) = t_s/(t_s/n) = n}$$

Linear Speedup Usually Optimal (cont)

- **We shall later see that this "proof" is not valid for certain types of nontraditional problems.**
- Unfortunately, the best speedup possible for most applications is much smaller than *n*
  - The optimal performance assumed in last proof is unattainable.
  - Usually some parts of programs are sequential and allow only one PE to be active.
  - Sometimes a large number of processors are idle for certain portions of the program.
    - During parts of the execution, many PEs may be waiting to receive or to send data.
    - E.g., recall blocking can occur in message passing

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

Superlinear Speedup

- Superlinear speedup occurs when $S(n) > n$
- Most texts besides Akl's and Quinn's argue that
  - Linear speedup is the maximum speedup obtainable.
    - The preceding "proof" is used to argue that superlinearity is always impossible.
  - Occasionally speedup that appears to be superlinear may occur, but can be explained by other reasons such as
    - the extra memory in parallel system.
    - a sub-optimal sequential algorithm used.
    - luck, in case of algorithm that has a random aspect in its design (e.g., random selection)

Superlinearity (cont)

- Selim Akl has given a multitude of examples that establish that superlinear algorithms are required for many nonstandad problems
  - If a problem either cannot be solved or cannot be solved in the required time without the use of parallel computation, it seems fair to say that $t_s=\infty$.

    Since for a fixed $t_p>0$, $S(n) = t_s/t_p$ is greater than 1 for all sufficiently large values of t_s, it seems reasonable to consider these solutions to be "superlinear".
  - Examples include "nonstandard" problems involving
    - Real-Time requirements where meeting deadlines is part of the problem requirements.
    - ***Problems where all data is not initially available, but has to be processed after it arrives.***
    - Real life situations such as a "person who can only keep a driveway open during a severe snowstorm with the help of friends".
  - Some problems are natural to solve using parallelism and sequential solutions are inefficient.

Linear Speedup

- Practically, there is a little chance to get linear speedup because the multiple processes / threads invariably introduces some overhead.
- Example: Shared memory programs will always have critical sections, which will require that, use some mutual exclusion mechanism such as a mutex.
- The calls to the mutex functions are overhead that's not present in the serial program, and the use of the mutex forces the parallel program to serialize the execution of the critical section.
- Distributed memory programs will almost always need to transmit data across the network, which is usually much slower than local memory access. Serial programs, on the other hand, won't have these overheads.

Linear Speedup

- So getting linear speedup for parallel programs is unusual.
- Overheads will increase when we increase the number of processes or threads
  - More threads will probably mean more threads need to access the critical section
  - More processes will probably mean more data needs to be transmitted across the network.
- So if we define speedup of a parallel program to be

$$S(n) = T_{serial} / T_{parallel}$$

- Then linear speedup has S = p, which is unusual.
- Furthermore, as p increases we expect S to become smallr and smaller fraction of the ideal, linear speedup p.
- In another way, S/p will probably get smaller and smaller as p increases.

Linear Speedup

- This value, S/p sometimes called as **efficiency of the parallel program**.

$$E = \frac{S}{p}$$

$$= \frac{\left(\frac{T_{serial}}{T_{parallel}} \right)}{p} = \frac{T_{serial}}{p \cdot T_{parallel}}$$

P	1	2	4	8	16
S	1.0	1.9	3.6	6.5	10.8
$E = \frac{S}{p}$	1.0	0.95	0.90	0.81	0.68

Table 2.1 : Speedups and Efficiencies of parallel program

Linear Speedup

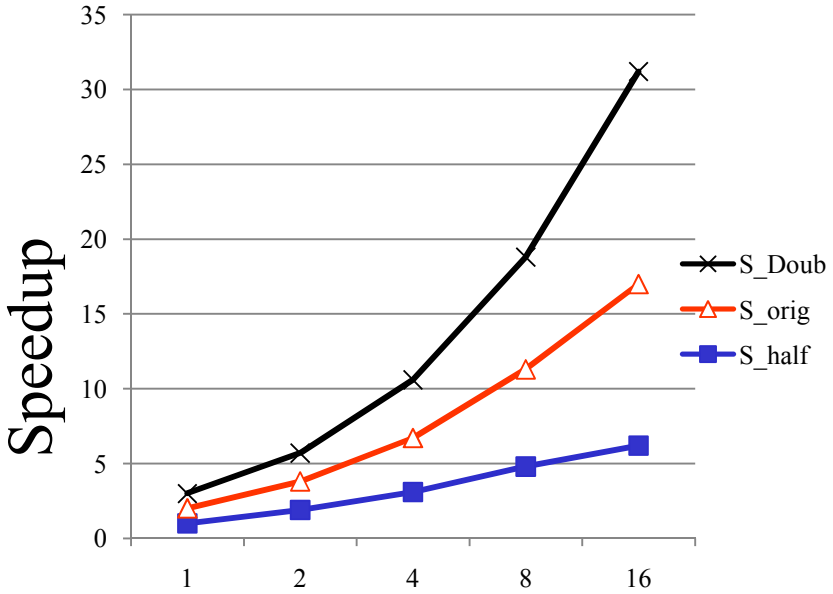
Table 2.2

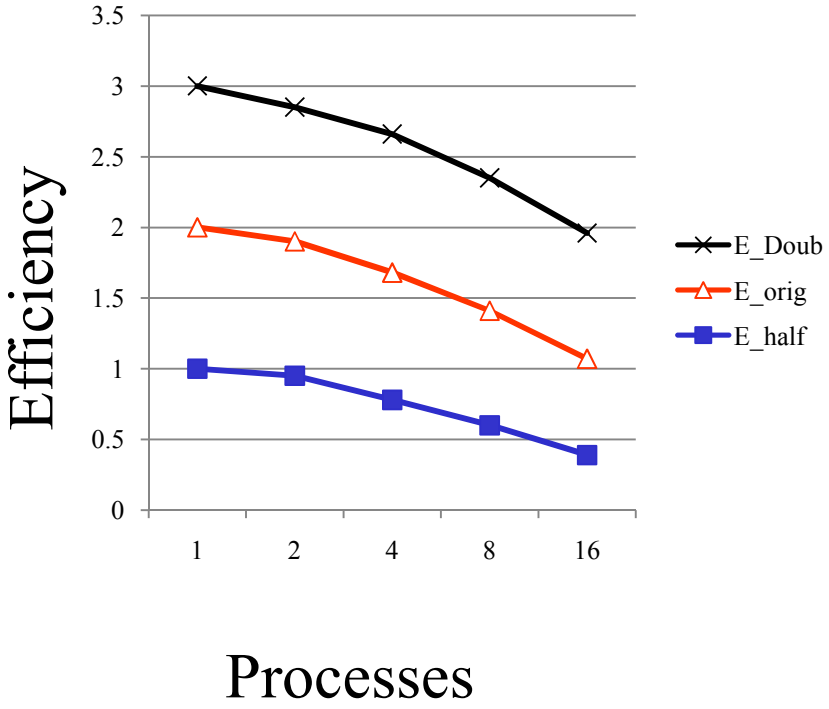
	P	1	2	4	8	16
Half	S	1.0	1.9	3.1	4.8	**6.2**
	E	1.0	0.95	0.78	0.60	**0.39**
Original	S	1.0	1.9	3.6	6.5	**10.8**
	E	1.0	0.95	0.90	0.81	**0.68**
Double	S	1.0	1.9	3.9	7.5	**14.2**
	E	1.0	0.95	0.98	0.94	**0.89**

- It's clear that $T_{parallel}$, S and E depend on p. the number of processes or threads.
- Also, $T_{parallel}$, S, E and T_{serial} all depend on the problem size.

Linear Speedup

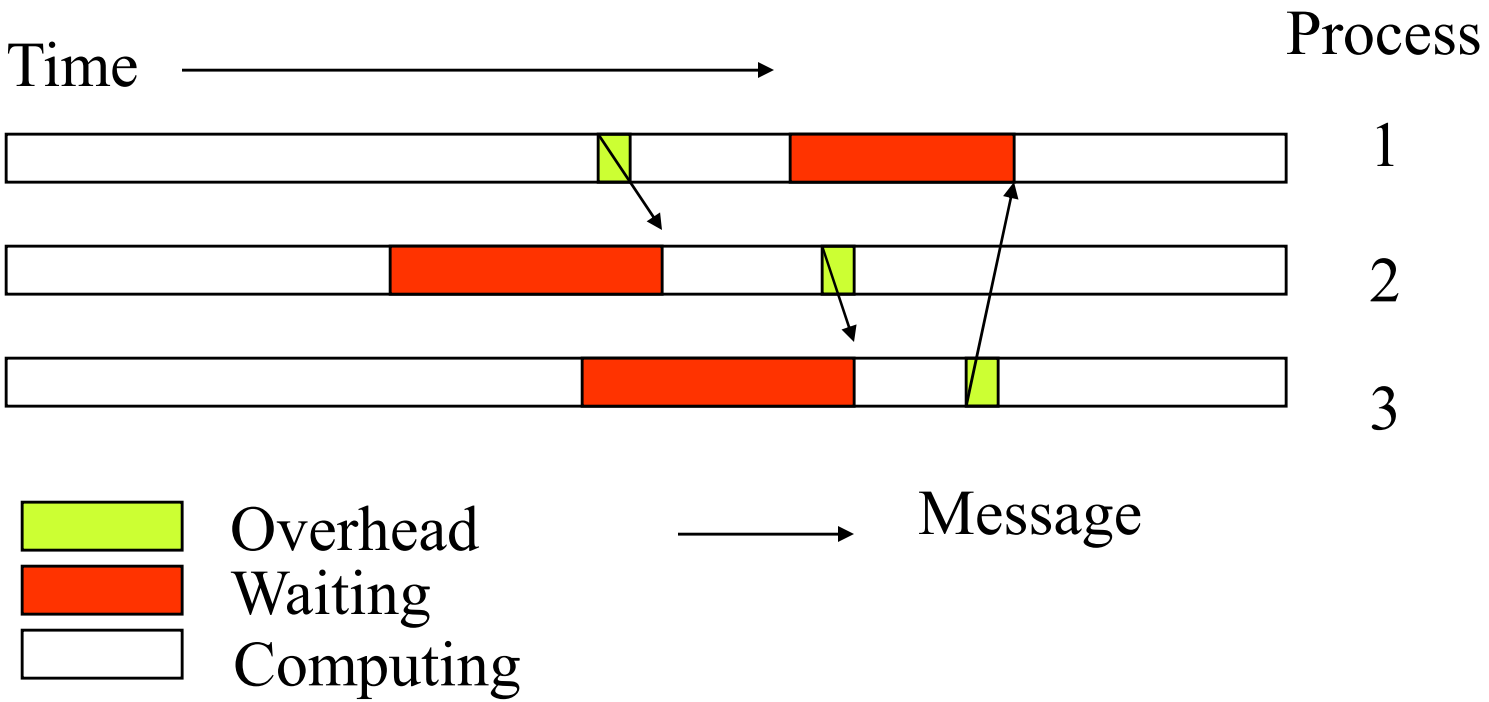
- For example, if halve and double the problem size of the program (whose speedups are shown in Table 2.2) get the speedups and efficiencies shown in Table 2.2.





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

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

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)

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, $1 = s + \pi$ = 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

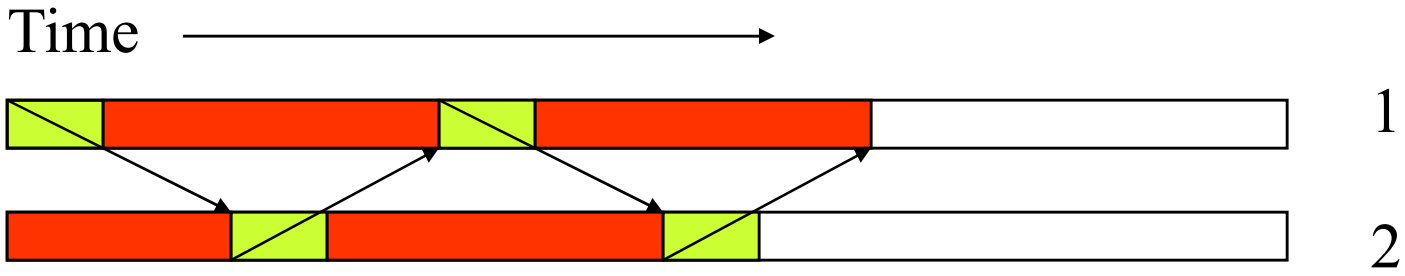
- Basic metrics
  - Bandwith
  - Latency
- Parallel metrics
  - Barrier speed
  - Broadcast/Multicast
  - Reductions (eg. Global sum, average, …)
  - Scatter speed

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: $t_{comm} = t_{startup} + nt_{data}$

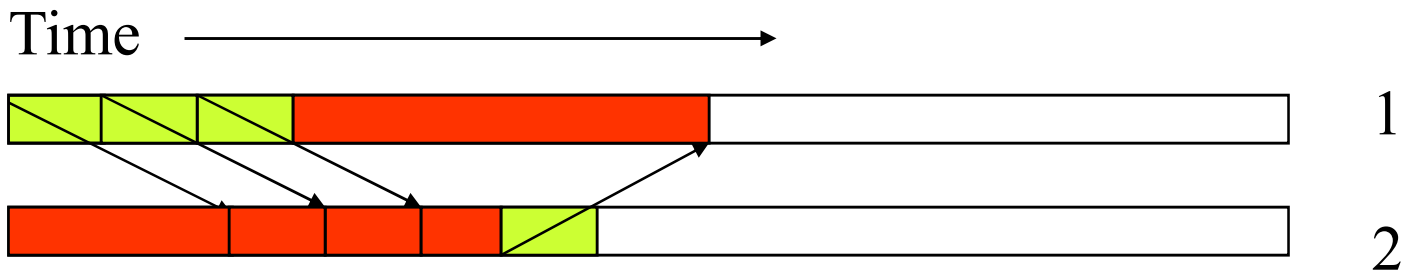
Ping-Pong

- 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.