18-742 Fall 2012
Parallel Computer Architecture
Lecture 2: Basics

Prof. Onur Mutlu
Carnegie Mellon University
9/10/2012

Reminder: Assignments for This Week

1. Review two papers from ISCA 2012 – due September 11, 11:59pm.

2. Attend NVIDIA talk on September 10 – write an online review of the talk; due September 11, 11:59pm.

3. Think hard about
   - Literature survey topics
   - Research project topics

4. Examine survey and project topics from Spring 2011

5. Find your literature survey and project partner

Reminder: Reviews for This Week

- Due: Tuesday September 11, 11:59pm

- Required – Enter reviews in the online system

- Pick **two** papers from ISCA 2012 proceedings
  - At least one of them should be new to you
  - Cannot be a paper you have co-authored
  - http://isca2012.ittc.ku.edu/index.php?option=com_content&view=article&id=53&Itemid=57
- Read them thoroughly
- Write a "critical" review for each paper online
- Bonus: pick three papers instead of two

Reminder: NVIDIA Talk Today

- NVIDIA Tech Talk by ECE Alumnus, Philip Cuadra
- Monday, September 10, 2012 7-9 pm, HH-1107
- Refreshments will be provided.

- Inside the Kepler GPU Architecture and Dynamic Parallelism
- This talk will dive into the features of the compute architecture for "Kepler" – NVIDIA's new 7-billion transistor GPU. From the reorganized processing cores with new instructions and processing capabilities, to an improved memory system with faster atomic processing and low-overhead ECC, we will explore how the Kepler GPU achieves world leading performance and efficiency, and how it enables wholly new types of parallel problems to be solved. The software improvements to expose the new dynamic parallelism features of the architecture will also be discussed.

- Your job: Attend the talk and write a review of the talk.

Reminder: How to Do the Reviews

- Brief summary
  - What is the problem the paper is trying to solve?
  - What are the key ideas of the paper? Key insights?
  - What is the key contribution to literature at the time it was written?
  - What are the most important things you take out from it?
- Strengths (most important ones)
  - Does the paper solve the problem well?
- Weaknesses (most important ones)
  - This is where you should think critically. Every paper/idea has a weakness. This does not mean the paper is necessarily bad. It means there is room for improvement and future research can accomplish this.
- Can you do (much) better? Present your thoughts/ideas.
- What have you learned/enjoyed most in the paper? Why?

- Review should be short and concise (~half a page or shorter)

Reminder: Advice on Paper/Talk Reviews

- When doing the reviews, be very critical

- Always think about better ways of solving the problem or related problems

- Do background reading
  - Reviewing a paper/talk is the best way of learning about a research problem/topic

- Think about forming a literature survey topic or a research proposal

Reminder: Literature Survey

- More information to come

- Read a lot of papers; find focused problem areas to survey papers on

Supplementary Readings on Research, Writing, Reviews

- Hamming, "You and Your Research," Bell Communications Research Colloquium Seminar, 7 March 1986.
    - http://www.cs.virginia.edu/~robins/YouAndYourResearch.html

- Levin and Redell, "How (and how not) to write a good systems paper," OSR 1983.

- Smith, "The Task of the Referee," IEEE Computer 1990.
- Read this to get an idea of the publication process

- SP Jones, "How to Write a Great Research Paper"

- Fong, "How to Write a CS Research Paper: A Bibliography"

Quiz 0 (Student Info Sheet)

- Was due Sep 7 (this Friday)
- Our way of getting to know about you fast
- All grading predicated on passing Quiz 0
  - But, you are not in this room for grades anyway

Today's Lecture

- Basics of Parallel Processing

- Outline of required readings (no reviews required yet):
  - Hill, Jouppi, Sohi, "Multiprocessors and Multicomputers," pp. 551-560 in Readings in Computer Architecture.
  - Hill, Jouppi, Sohi, "Dataflow and Multithreading," pp. 309-314 in Readings in Computer Architecture.

  - Suleman et al., "Accelerating Critical Section Execution with Asymmetric Multi-Core Architectures," ASPLOS 2009.
  - Culler & Singh, Chapter 1

Parallel Computer Architecture: Basics

What is a Parallel Computer?

- Definition of a "parallel computer" not really precise
- "A 'parallel computer' is a "collection of processing elements that communicate and cooperate to solve large problems fast"
  - Almasi and Gottlieb, "Highly Parallel Computing," 1989
- Is a superscalar processor a parallel computer?

- A processor that gives the illusion of executing a sequential ISA on a single thread at a time is a sequential machine
- Almost anything else is a parallel machine

- Examples of parallel machines:
  - Multiple program counters (PCs)
  - Multiple data being operated on simultaneously
  - Some combination

Remember: Flynn's Taxonomy of Computers

- Mike Flynn, "Very High-Speed Computing Systems," Proc. of IEEE, 1966

- SISD: Single instruction operates on single data element
- SIMD: Single instruction operates on multiple data elements
  - Array processor
  - Vector processor
- MISD: Multiple instructions operate on single data element
  - Closest form: systolic array processor, streaming processor
- MIMD: Multiple instructions operate on multiple data elements (multiple instruction streams)
  - Multiprocessor
  - Multithreaded processor

Why Parallel Computers?

- Parallelism: Doing multiple things at a time
- Things: instructions, operations, tasks

- Main Goal
  - Improve performance (Execution time or task throughput)
    - Execution time of a program governed by Amdahl's Law

- Other Goals
  - Reduce power consumption
    - (4N units at freq F/4) consume less power than (N units at freq F)
    - Why?
  - Improve cost efficiency and scalability, reduce complexity
    - Harder to design a single unit that performs as well as N simpler units
  - Improve dependability: Redundant execution in space

Types of Parallelism and How to Exploit Them

- Instruction Level Parallelism
  - Different instructions within a stream can be executed in parallel
  - Pipelining, out-of-order execution, speculative execution, VLIW
  - Dataflow

- Data Parallelism
  - Different pieces of data can be operated on in parallel
  - SIMD: Vector processing, array processing
  - Systolic arrays, streaming processors

- Task Level Parallelism
  - Different “tasks/threads” can be executed in parallel
  - Multithreading
  - Multiprocessing (multi-core)

Task-Level Parallelism: Creating Tasks

- Partition a single problem into multiple related tasks (threads)
  - Explicitly: Parallel programming
    - Easy when tasks are natural in the problem
      - Web/database queries
    - Difficult when natural task boundaries are unclear
  - Transparently/implicitly: Thread level speculation
    - Partition a single thread speculatively
- Run many independent tasks (processes) together
  - Easy when there are many processes
    - Batch simulations, different users, cloud computing workloads
  - Does not improve the performance of a single task

MIMD Processing

- Loosely coupled multiprocessors
  - No shared global memory address space
  - Multicomputer network
    - Network-based multiprocessors
  - Usually programmed via message passing
    - Explicit calls (send, receive) for communication

- Tightly coupled multiprocessors
  - Shared global memory address space
  - Traditional multiprocessing: symmetric multiprocessing (SMP)
  - Existing multi-core processors, multithreaded processors
  - Programming model similar to uniprocessors (i.e., multitasking uniprocessor) except
    - Operations on shared data require synchronization

Main Issues in Tightly-Coupled MP

- Shared memory synchronization
  - Locks, atomic operations
- Cache consistency
  - More commonly called cache coherence
- Ordering of memory operations
  - What should the programmer expect the hardware to provide?
- Resource sharing, contention, partitioning
- Communication: Interconnection networks
- Load imbalance

Aside: Hardware-based Multithreading

- Coarse grained
  - Quantum based
  - Event based (switch-on-event multithreading)
- Fine grained
  - Cycle by cycle
  - Thornton, "CDC 6600: Design of a Computer," 1970.
  - Burton Smith, "A pipelined, shared resource MIMD computer," ICPP 1978.
- Simultaneous
  - Can dispatch instructions from multiple threads at the same time
  - Good for improving execution unit utilization

Caveats of Parallelism

- Amdahl' s Law
  - p: Parallelizable fraction of a program
  - N: Number of processors

$$\text{Speedup} = \frac{1}{1 - p + \frac{p}{N}}$$

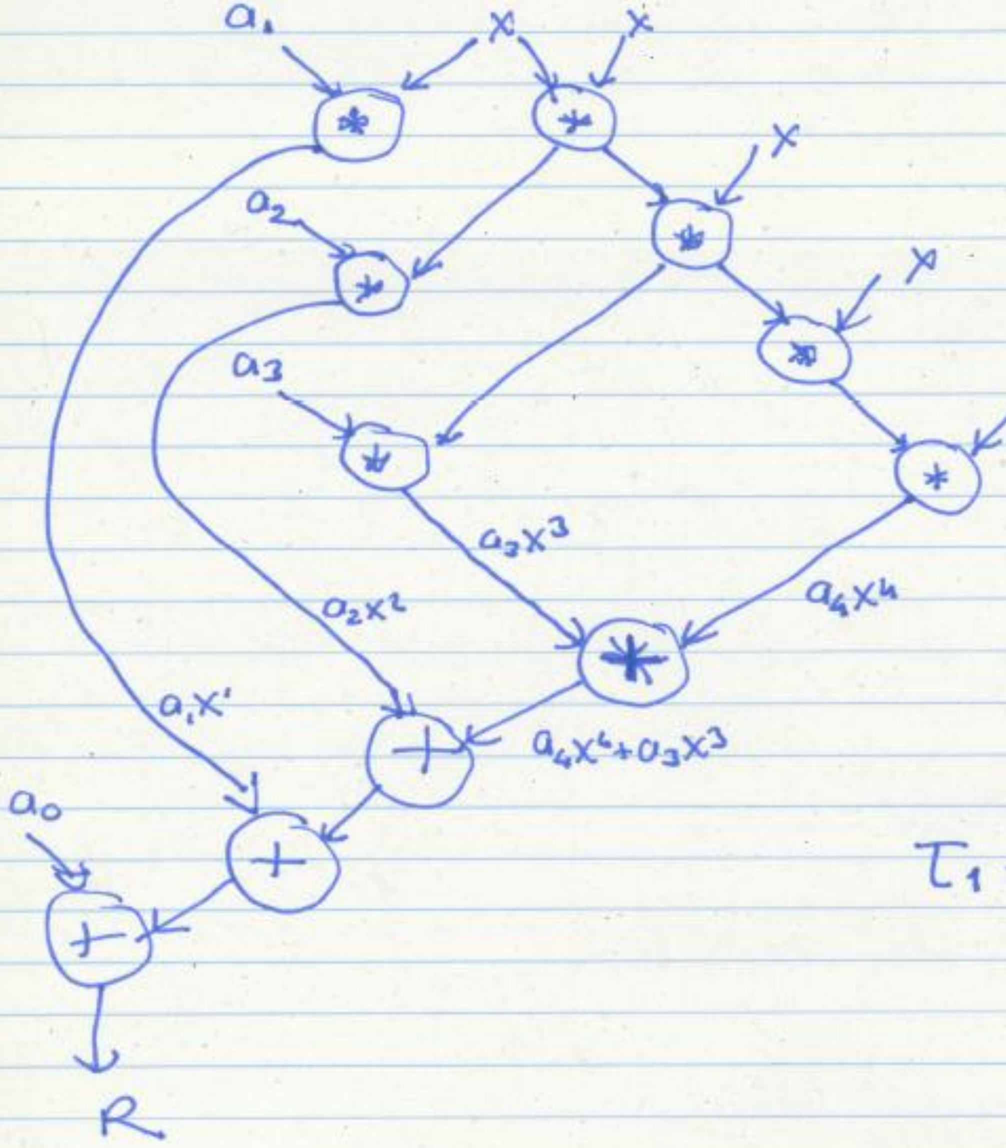
  - Amdahl, "Validity of the single processor approach to achieving large scale computing capabilities," AFIPS 1967.
- Maximum speedup limited by serial portion: Serial bottleneck
- Parallel portion is usually not perfectly parallel
  - Synchronization overhead (e.g., updates to shared data)
  - Load imbalance overhead (imperfect parallelization)
  - Resource sharing overhead (contention among N processors)

Parallel Speedup Example

- $a4x^4 + a3x^3 + a2x^2 + a1x + a0$
- Assume each operation 1 cycle, no communication cost, each op can be executed in a different processor
- How fast is this with a single processor?
- How fast is this with 3 processors?

$R = a_4x^4 + a_3x^3 + a_2x^2 + a_1x + a_0$

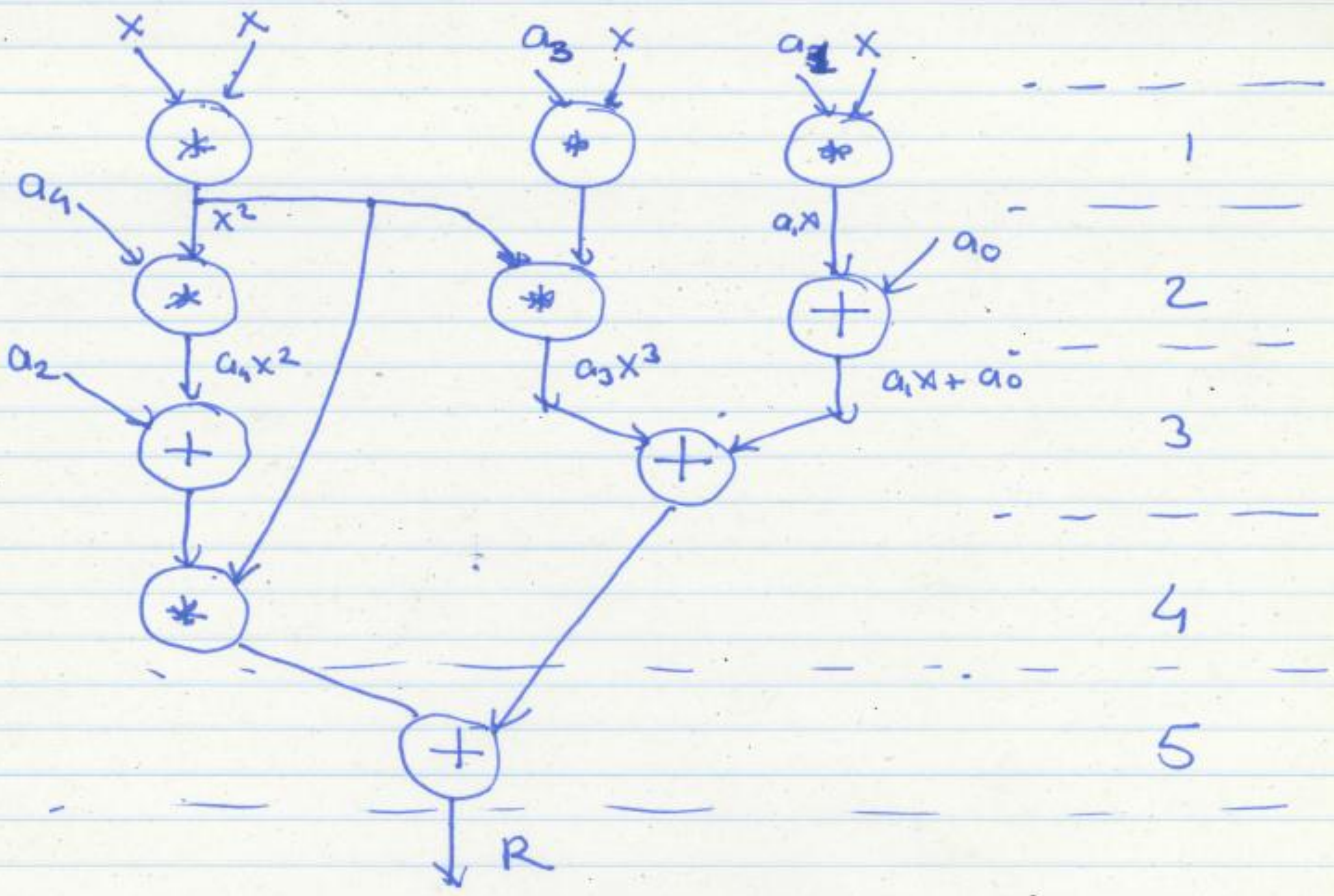
Single processor : 11 operations (DRAW the data flow graph)



$T_1 = 11$ cycles

$R = a_4x^4 + a_3x^3 + a_2x^2 + a_1x + a_0$

Three processors: τ_3 (exec. time with 3 proc.)



$\tau_3 = \underline{5 \text{ cycles}}$

Speedup with 3 Processors

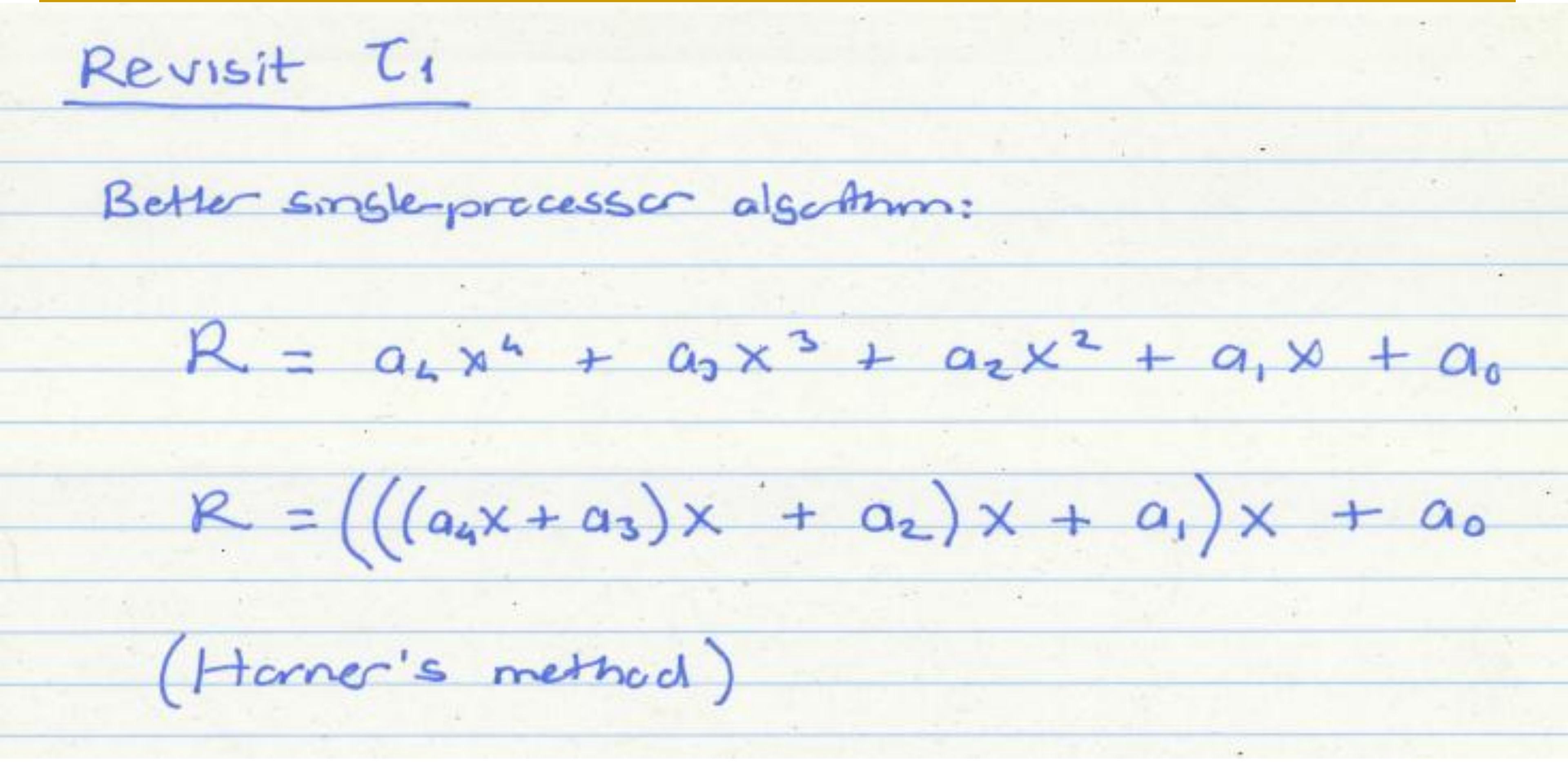
$\tau_3 = \underline{5 \text{ cycles}}$

Speedup with 3 processors $= \dfrac{11}{5} = 2.2$

$$\left(\frac{\tau_1}{\tau_3}\right)$$

<u>Is this a fair comparison?</u>

Revisiting the Single-Processor Algorithm



Revisit τ_1

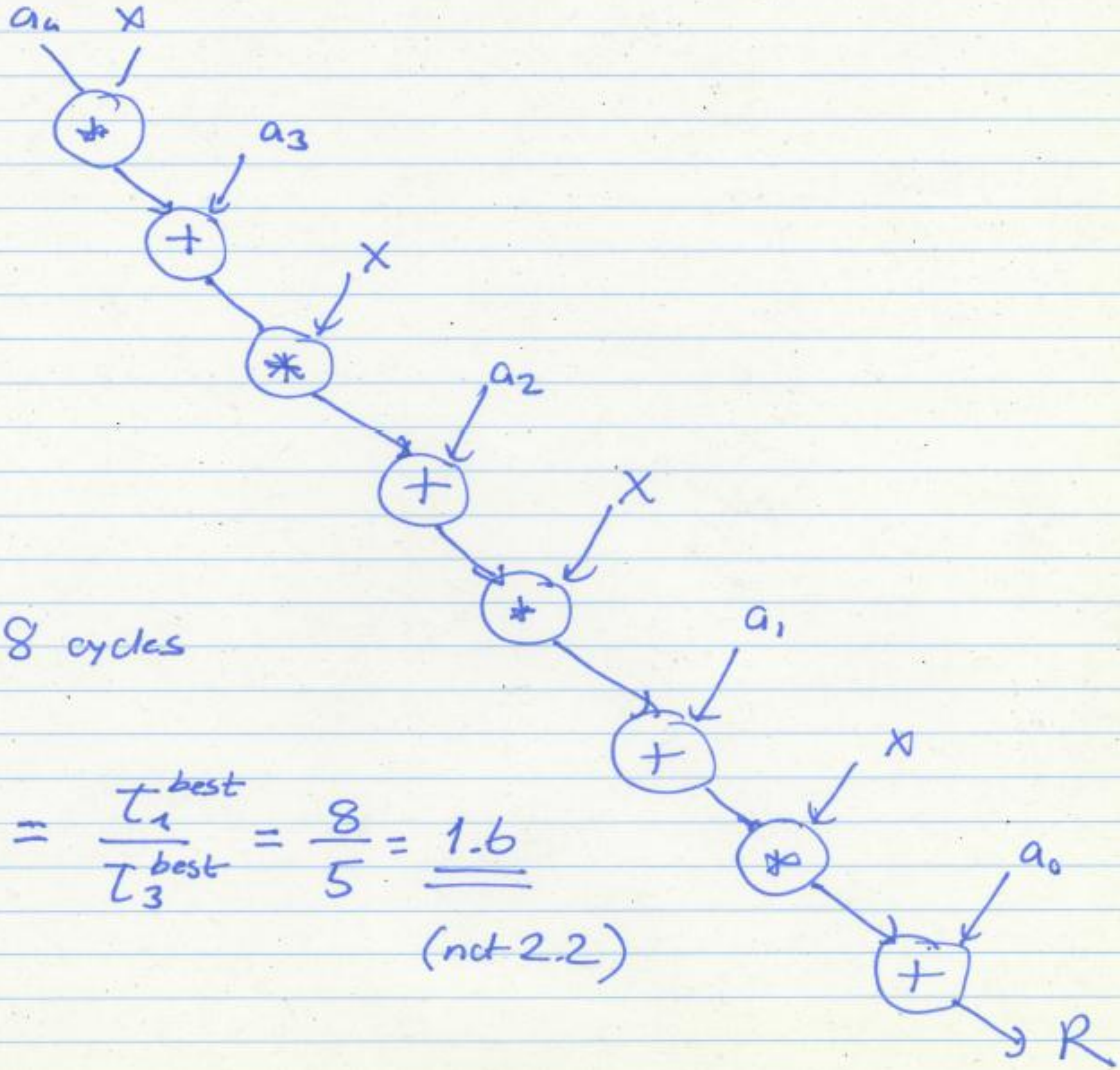
Better single-processor algorithm:

$$R = a_4 x^4 + a_3 x^3 + a_2 x^2 + a_1 x + a_0$$

$$R = (((a_4 x + a_3) x + a_2) x + a_1) x + a_0$$

(Horner's method)

Horner, "A new method of solving numerical equations of all orders, by continuous approximation," Philosophical Transactions of the Royal Society, 1819.



$\tau_1 = 8$ cycles

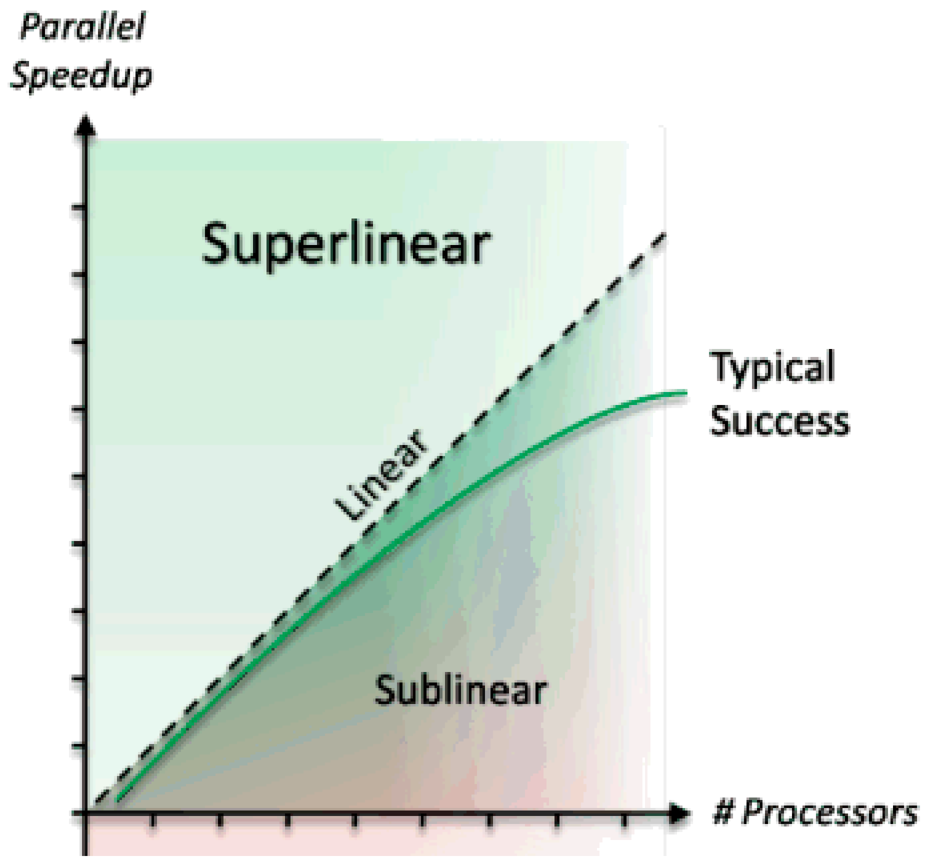
Speedup with 3 procs. $= \dfrac{\tau_1^{best}}{\tau_3^{best}} = \dfrac{8}{5} = \underline{\underline{1.6}}$

(not 2.2)

Superlinear Speedup

- Can speedup be greater than P with P processing elements?

- Happens in two ways:
  - Unfair comparisons
  - Memory/cache effects



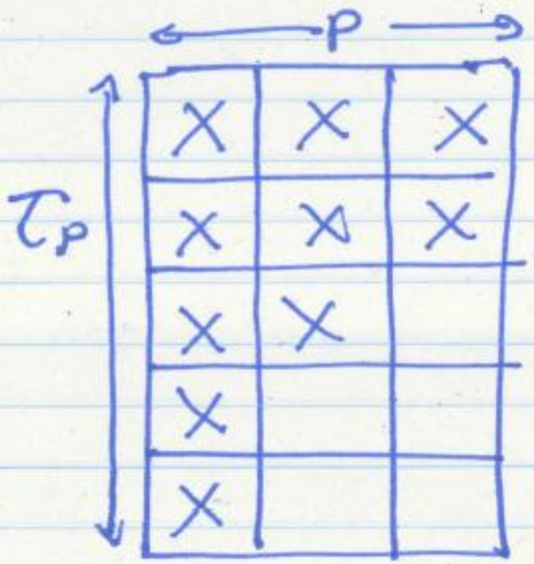
Utilization, Redundancy, Efficiency

- Traditional metrics
  - Assume all P processors are tied up for parallel computation
- Utilization: How much processing capability is used
  - U = (# Operations in parallel version) / (processors x Time)
- Redundancy: how much extra work is done with parallel processing
  - R = (# of operations in parallel version) / (# operations in best single processor algorithm version)
- Efficiency
  - E = (Time with 1 processor) / (processors x Time with P processors)
  - E = U/R

Utilization of Multiprocessor

Multiprocessor metrics

Utilization: How much processing capability we use



$$U = \frac{\text{10 operations (in parallel version)}}{\text{3 processors} \times \text{5 time units}}$$

$$= \frac{10}{15}$$

$$U = \frac{\text{Ops with } p \text{ proc.}}{p \times \tau_p}$$

Redundancy: How much extra work due to multiprocessing

$$R = \frac{\text{Ops with } p \text{ proc.}^{\text{best}}}{\text{Ops with 1 proc.}^{\text{best}}} = \frac{10}{8}$$

R is always $\geqslant 1$

Efficiency: How much resource we use compared to how much resource we can get away with

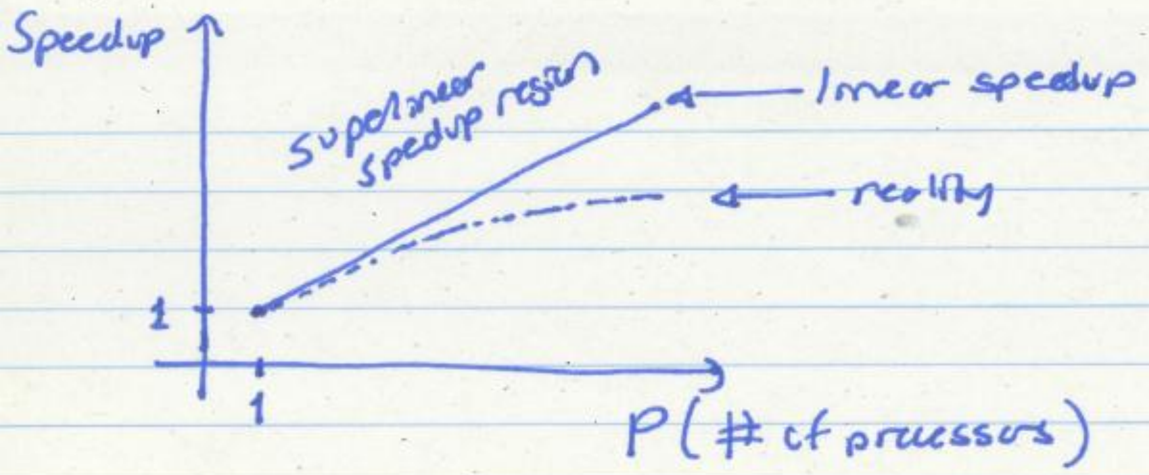
$$E = \frac{1 \cdot T_1^{\text{best}}}{p \cdot T_p^{\text{best}}}$$

(tying up 1 proc for T_p time units)

(tying up p proc. for T_p time units)

$$= \frac{8}{15} \qquad \left(E = \frac{U}{R} \right)$$

Caveats of Parallelism (I)



Why the reality? (diminishing returns)

$$\tau_P = \underbrace{\alpha \cdot \frac{\tau_1}{P}}_{\text{parallelizable part/fraction of the single-processor program}} + \underbrace{(1-\alpha) \cdot \tau_1}_{\text{non-parallelizable part}}$$

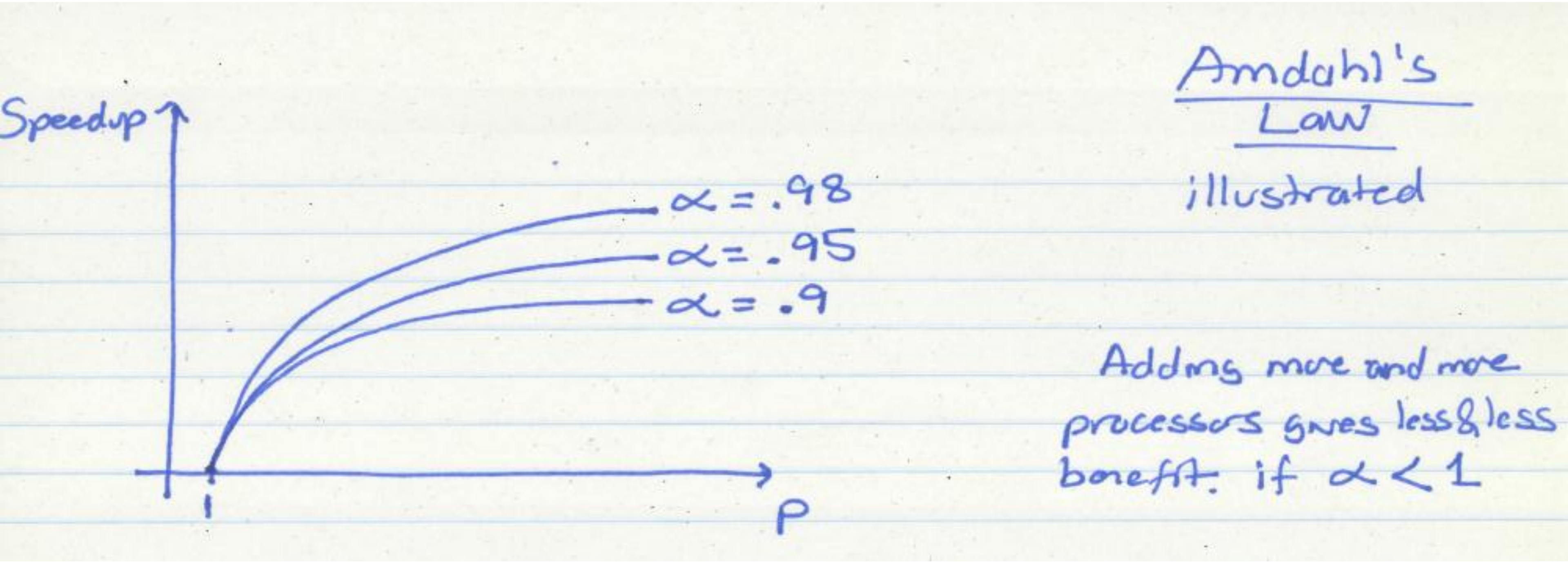
Amdahl's Law

$$\underset{\text{with } p \text{ proc.}}{\text{Speedup}} = \frac{t_1}{t_p} = \frac{1}{\frac{\alpha}{p} + (1-\alpha)}$$

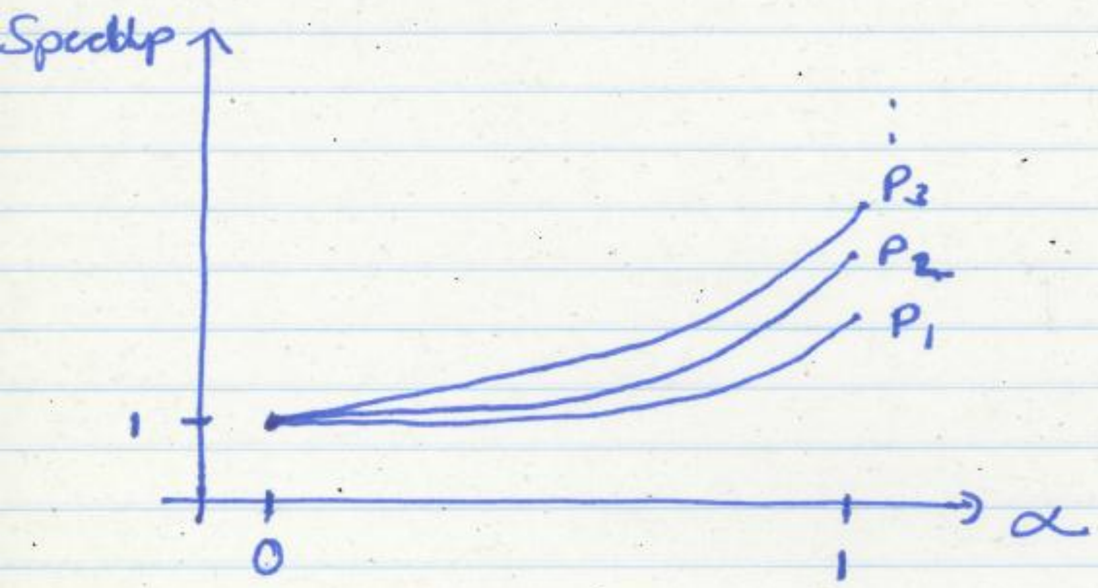
$$\underset{\text{as } p\to\infty}{\text{Speedup}} = \frac{1}{1-\boxed{\alpha}} \longrightarrow \text{bottleneck for parallel speedup}$$

Amdahl, "Validity of the single processor approach to achieving large scale computing capabilities," AFIPS 1967.

Amdahl's Law Implication 1



Amdahl's Law Implication 2



The benefit (speedup) is small until $\cancel{\alpha}\alpha \approx 1$

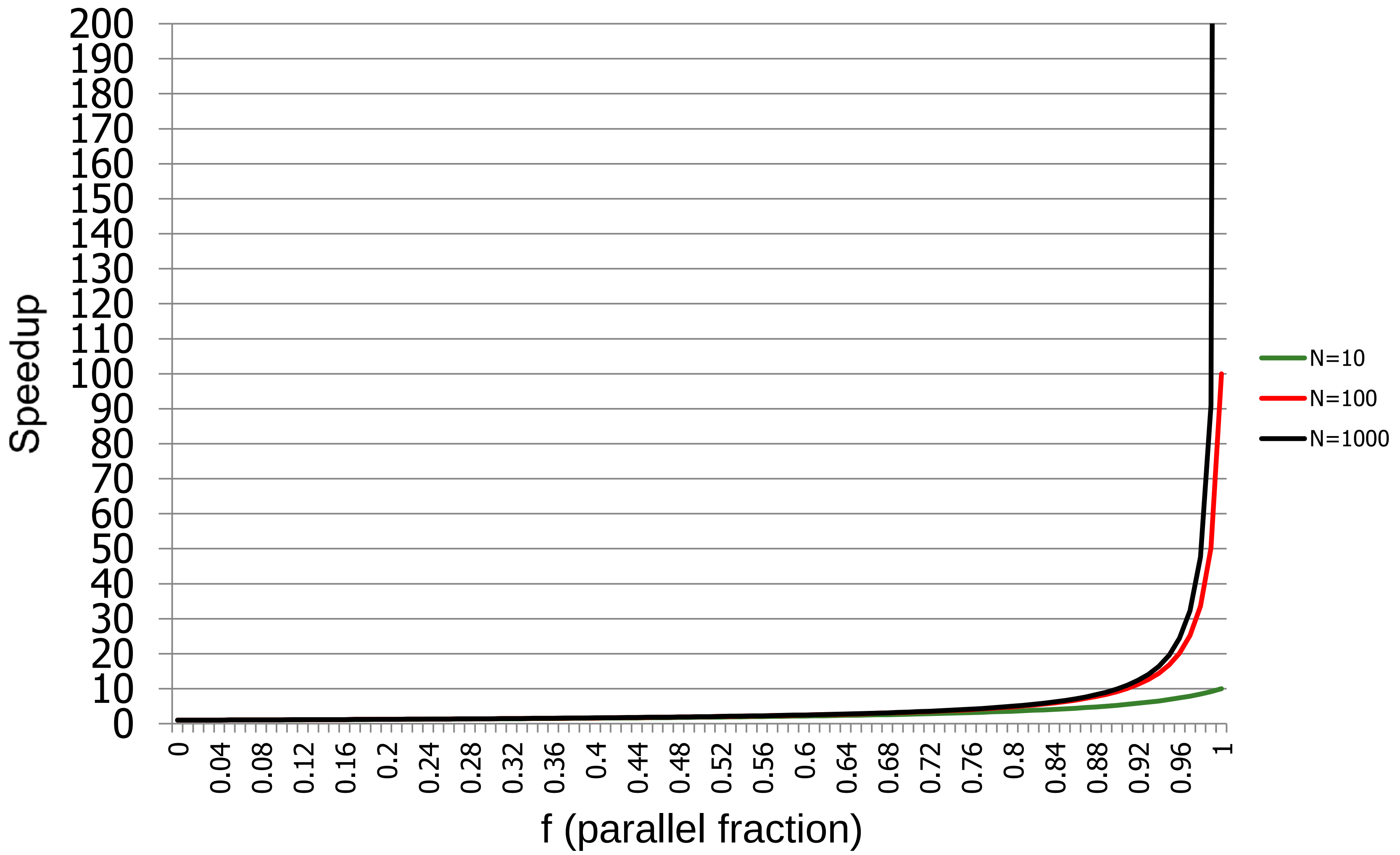
Caveats of Parallelism (Again)

- Amdahl's Law
  - f: Parallelizable fraction of a program
  - N: Number of processors

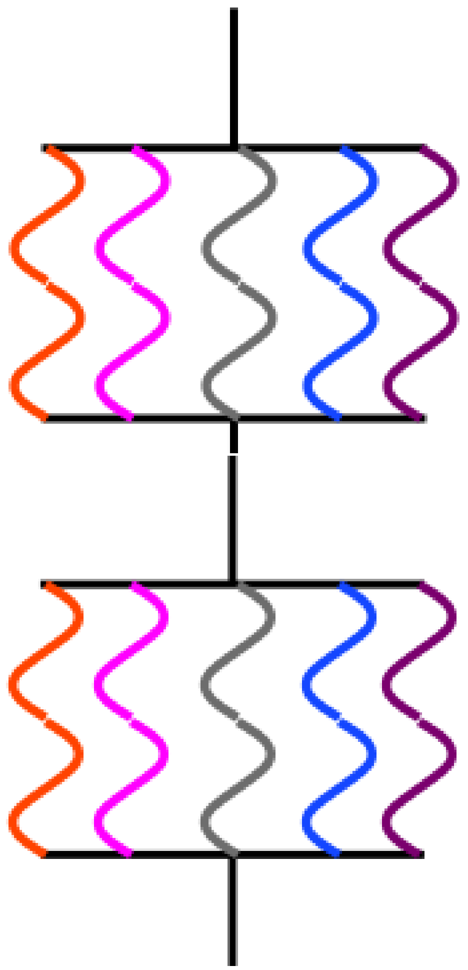
$$\text{Speedup} = \frac{1}{1 - f + \frac{f}{N}}$$

  - Amdahl, "Validity of the single processor approach to achieving large scale computing capabilities," AFIPS 1967.
- Maximum speedup limited by serial portion: Serial bottleneck
- Parallel portion is usually not perfectly parallel
  - Synchronization overhead (e.g., updates to shared data)
  - Load imbalance overhead (imperfect parallelization)
  - Resource sharing overhead (contention among N processors)

Sequential Bottleneck



Why the Sequential Bottleneck?

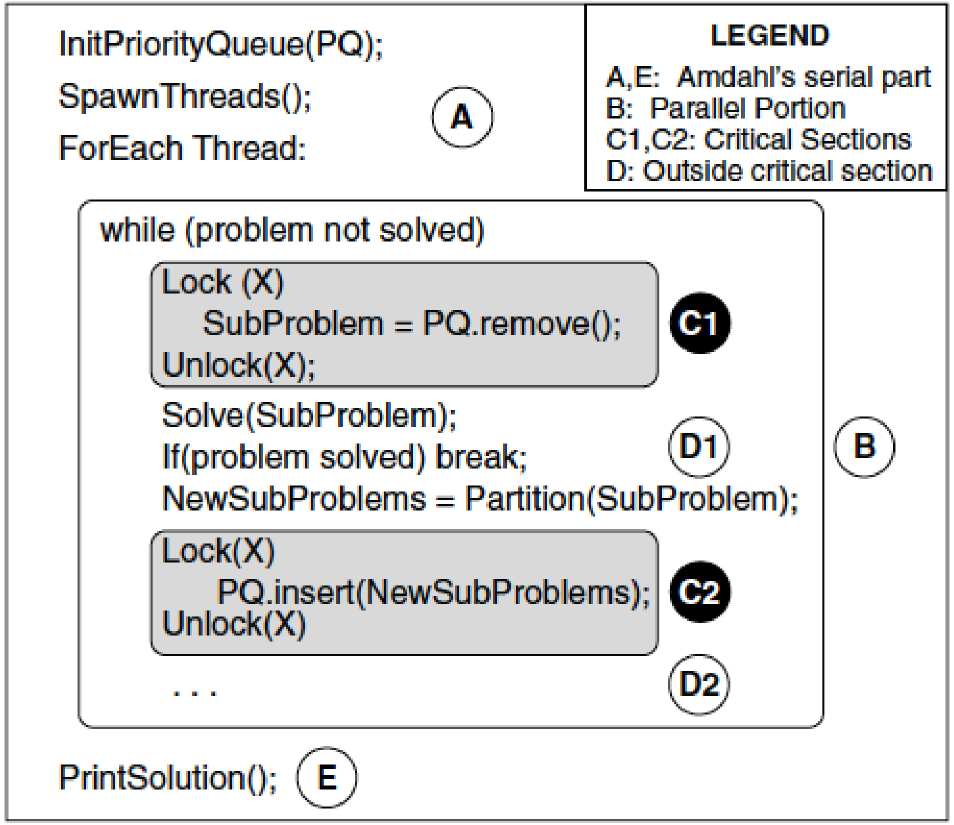


- Parallel machines have the sequential bottleneck
- Main cause: Non-parallelizable operations on data (e.g. non-parallelizable loops)

```
for ( i = 0 ; i < N; i++)
    A[i] = (A[i] + A[i-1]) / 2
```

- Single thread prepares data and spawns parallel tasks (usually sequential)

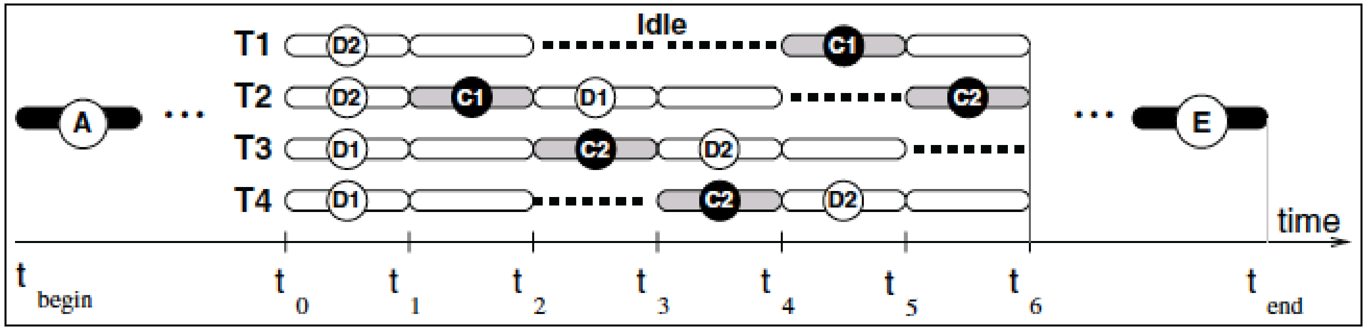
Another Example of Sequential Bottleneck



```
InitPriorityQueue(PQ);
SpawnThreads();
ForEach Thread:
    while (problem not solved)
        Lock (X)
            SubProblem = PQ.remove();
        Unlock(X);
        Solve(SubProblem);
        If(problem solved) break;
        NewSubProblems = Partition(SubProblem);
        Lock(X)
            PQ.insert(NewSubProblems);
        Unlock(X)
        . . .
PrintSolution();
```

LEGEND
A,E: Amdahl's serial part
B: Parallel Portion
C1,C2: Critical Sections
D: Outside critical section

(a)



Bottlenecks in Parallel Portion

- Synchronization: Operations manipulating shared data cannot be parallelized
  - Locks, mutual exclusion, barrier synchronization
  - Communication: Tasks may need values from each other

  - Causes thread serialization when shared data is contended

- Load Imbalance: Parallel tasks may have different lengths
  - Due to imperfect parallelization or microarchitectural effects

  - Reduces speedup in parallel portion

- Resource Contention: Parallel tasks can share hardware resources, delaying each other
  - Replicating all resources (e.g., memory) expensive

  - Additional latency not present when each task runs alone

Difficulty in Parallel Programming

- Little difficulty if parallelism is natural
  - "Embarrassingly parallel" applications
  - Multimedia, physical simulation, graphics
  - Large web servers, databases?

- Difficulty is in
  - Getting parallel programs to work correctly
  - Optimizing performance in the presence of bottlenecks

- Much of **parallel computer architecture** is about
  - Designing machines that overcome the sequential and parallel bottlenecks to achieve higher performance and efficiency
  - Making programmer's job easier in writing correct and high-performance parallel programs

Parallel and Serial Bottlenecks

- How do you alleviate some of the serial and parallel bottlenecks in a multi-core processor?
- We will return to this question in the next few lectures
- Reading list:
  - Hill and Marty, "Amdahl's Law in the Multi-Core Era," IEEE Computer 2008.
  - Annavaram et al., "Mitigating Amdahl's Law Through EPI Throttling," ISCA 2005.
  - Suleman et al., "Accelerating Critical Section Execution with Asymmetric Multi-Core Architectures," ASPLOS 2009.
  - Joao et al., "Bottleneck Identification and Scheduling in Multithreaded Applications," ASPLOS 2012.
  - Ipek et al., "Core Fusion: Accommodating Software Diversity in Chip Multiprocessors," ISCA 2007.

Bottlenecks in the Parallel Portion

- Amdahl's Law does not consider these
- How do synchronization (e.g., critical sections), and load imbalance, resource contention affect parallel speedup?

- Can we develop an intuitive model (like Amdahl's Law) to reason about these?
  - A research topic
- Example papers:
  - Eyerman and Eeckhout, "Modeling critical sections in Amdahl's law and its implications for multicore design," ISCA 2010.
  - Suleman et al., "Feedback-driven threading: power-efficient and high-performance execution of multi-threaded workloads on CMPs," ASPLOS 2008.
- Need better analysis of critical sections in real programs

Reviews Due Sunday

- Sunday, September 16, 11:59pm.
- Suleman et al., "Accelerating Critical Section Execution with Asymmetric Multi-Core Architectures," ASPLOS 2009.
- Suleman et al., "Data Marshaling for Multi-core Architectures," ISCA 2010.
- Joao et al., "Bottleneck Identification and Scheduling in Multithreaded Applications," ASPLOS 2012.