Computer Architecture

Lecture 19a: Multiprocessors

Dr. Mohammad Sadrosadati

Prof. Onur Mutlu

ETH Zürich

Fall 2023

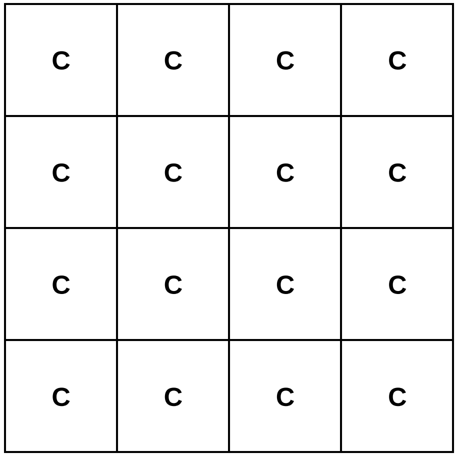
30 November 2023

Heterogeneity Wrap Up

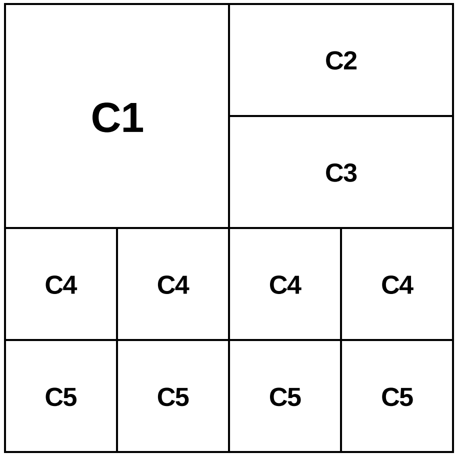
A Case for Asymmetry Everywhere

Onur Mutlu,
"Asymmetry Everywhere (with Automatic Resource Management)"
CRA Workshop on Advancing Computer Architecture Research: Popular Parallel Programming, San Diego, CA, February 2010.
Position paper

Asymmetry Enables Customization



Symmetric

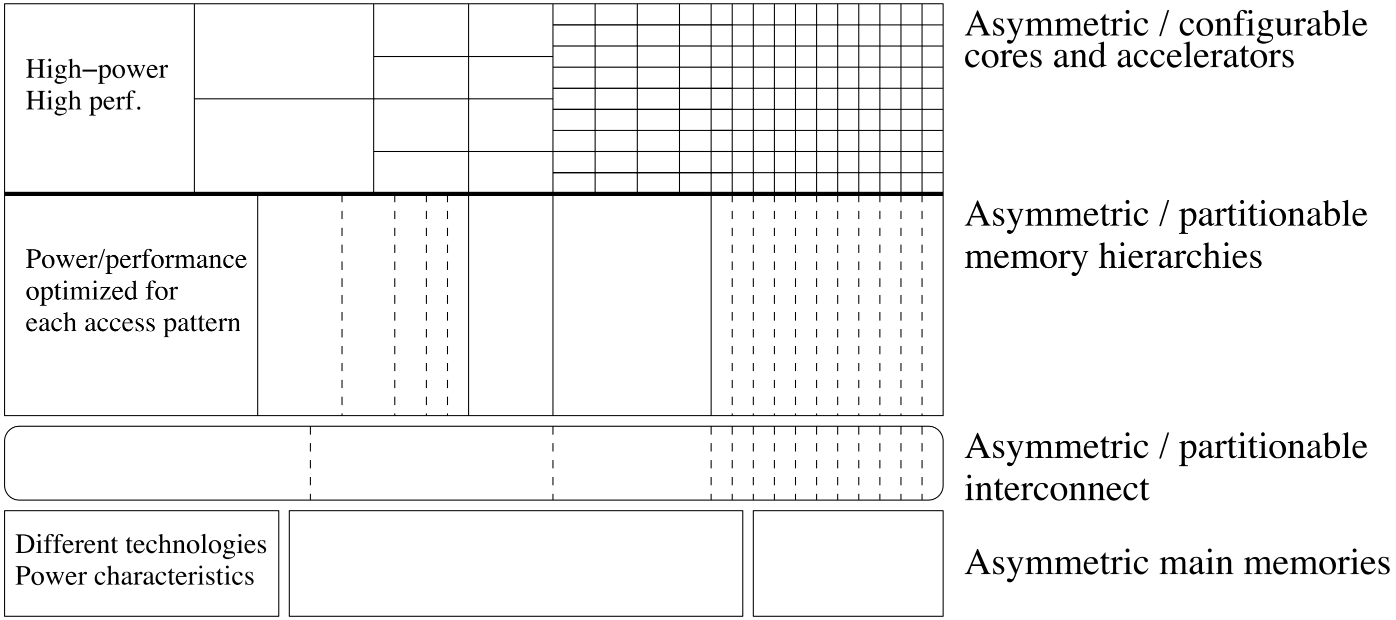


Asymmetric

- Symmetric: One size fits all
  - Energy and performance suboptimal for different phase behaviors
- Asymmetric: Enables tradeoffs and customization
  - Processing requirements vary across applications and phases
  - Execute code on best-fit resources (minimal energy, adequate perf.)

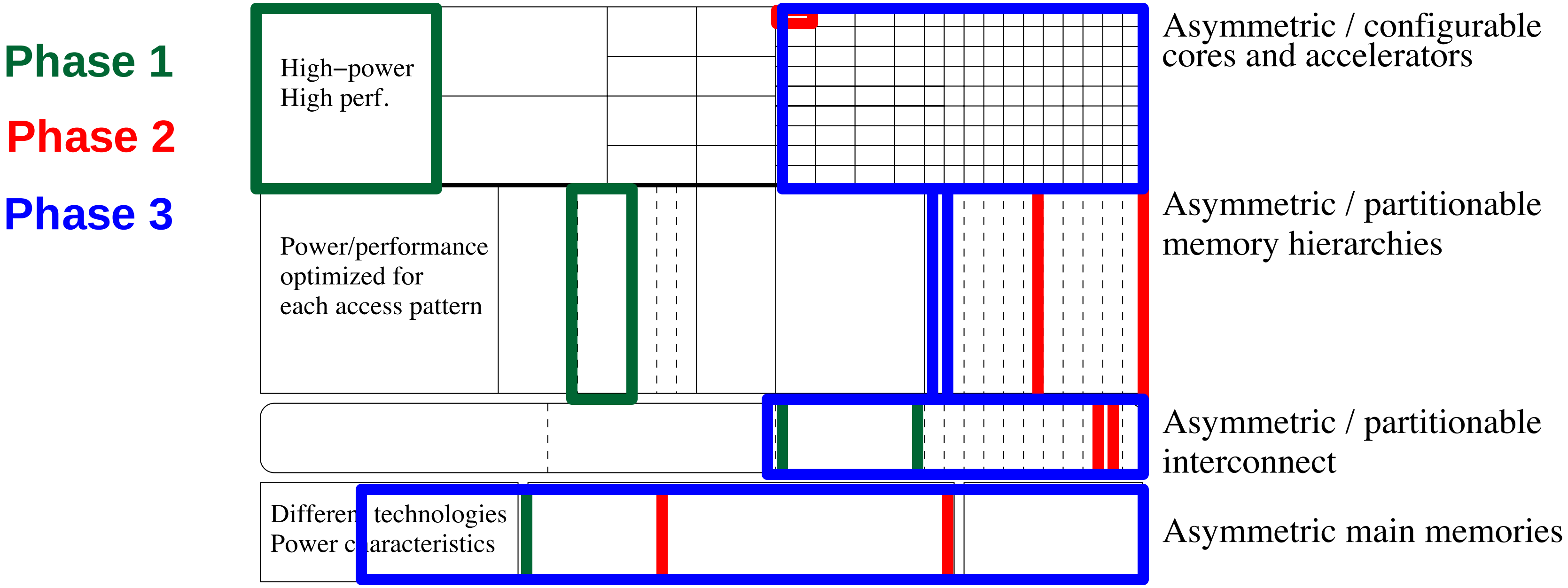
Thought Experiment: Asymmetry Everywhere

- Design each hardware resource with asymmetric, (re-)configurable, partitionable components
  - Different power/performance/reliability characteristics
  - To fit different computation/access/communication patterns



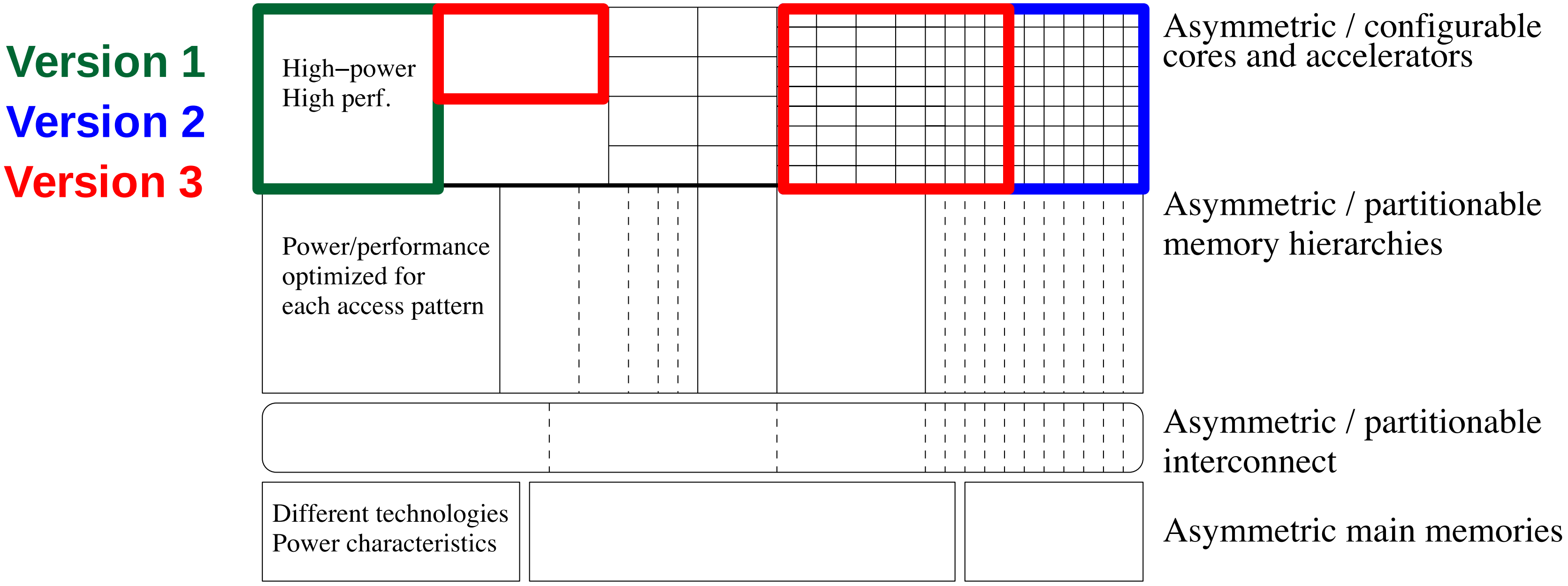
Thought Experiment: Asymmetry Everywhere

- Design the runtime system (HW & SW) to automatically choose the best-fit components for each phase
  - Satisfy performance/SLA with minimal energy
  - Dynamically stitch together the “best-fit” chip for each phase



Thought Experiment: Asymmetry Everywhere

- Morph software components to match asymmetric HW components
  - Multiple versions for different resource characteristics



Many Research and Design Questions

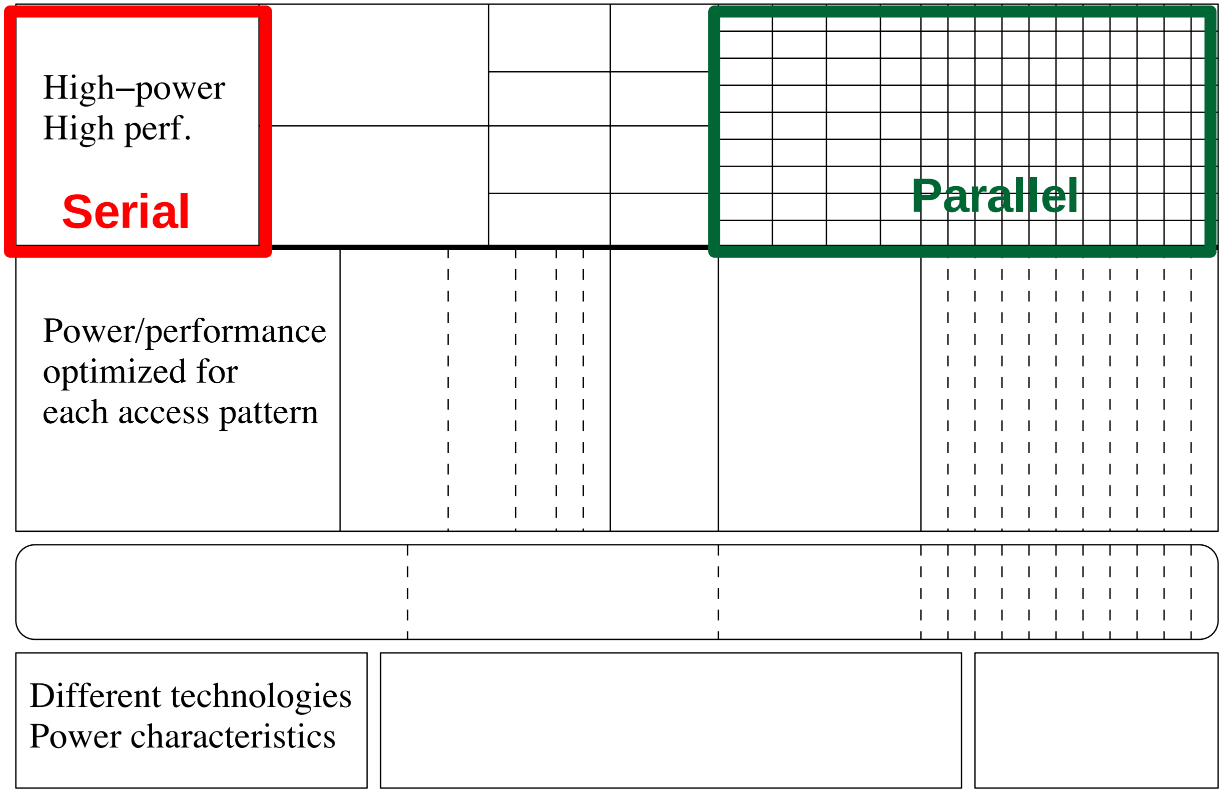
- How to design asymmetric components?
  - Fixed, partitionable, reconfigurable components?
  - What types of asymmetry? Access patterns, technologies?

- What monitoring to perform cooperatively in HW/SW?
  - Automatically discover phase/task requirements

- How to design feedback/control loop between components and runtime system software?

- How to design the runtime to automatically manage resources?
  - Track task behavior, pick "best-fit" components for the entire workload

Exploiting Asymmetry: Simple Examples



Asymmetric / configurable cores and accelerators

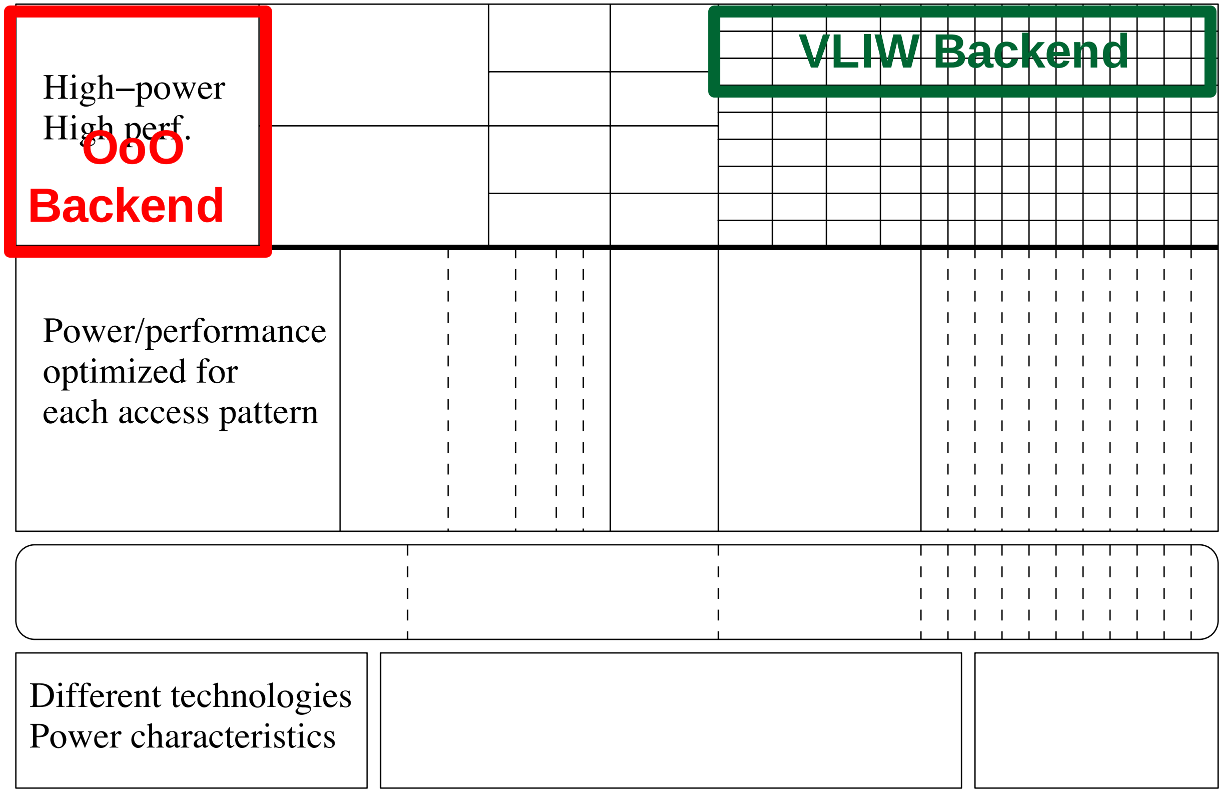
Asymmetric / partitionable memory hierarchies

Asymmetric / partitionable interconnect

Asymmetric main memories

- Execute critical/serial sections on high-power, high-performance cores/resources [Suleman+ ASPLOS'09, ISCA'10, Top Picks'10'11, Joao+ ASPLOS'12,ISCA'13]
  - Programmer can write less optimized, but more likely correct programs

Exploiting Asymmetry: Simple Examples



Asymmetric / configurable cores and accelerators

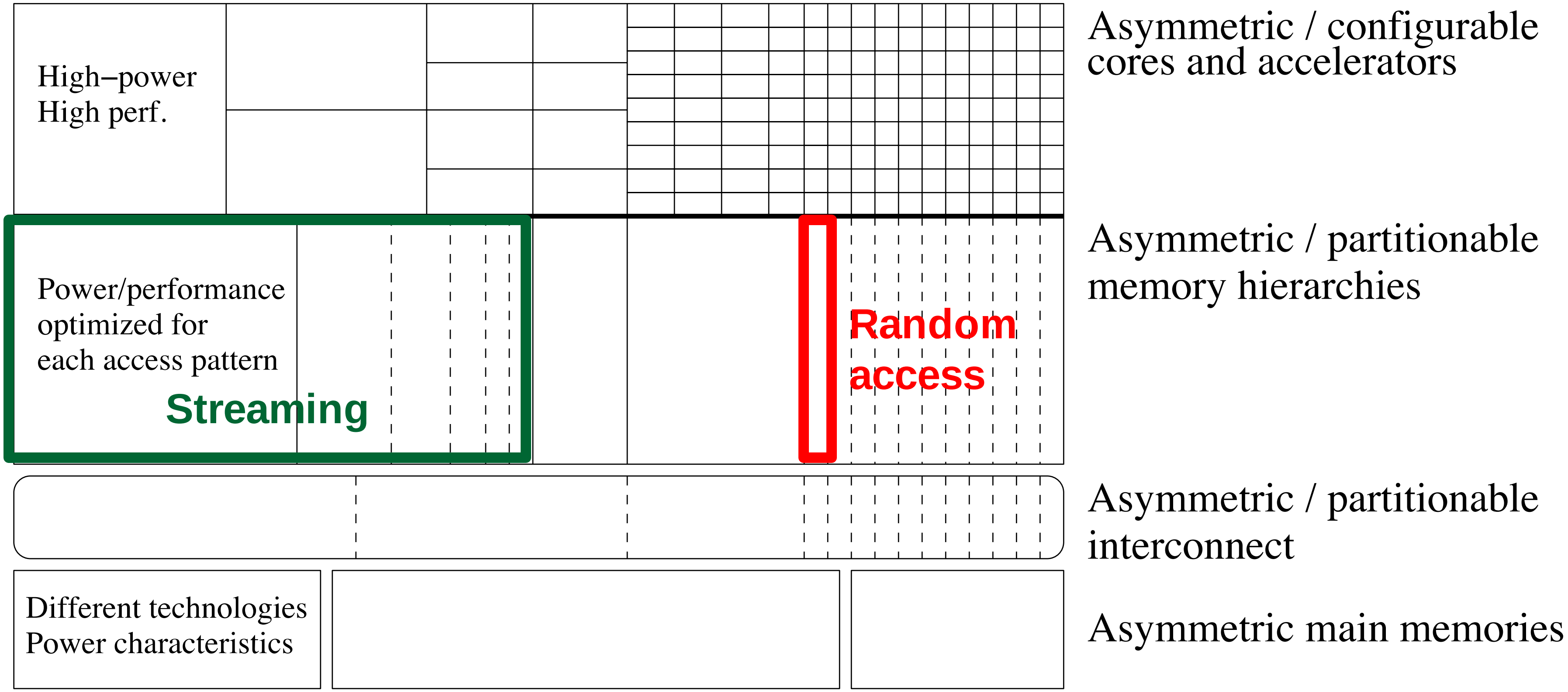
Asymmetric / partitionable memory hierarchies

Asymmetric / partitionable interconnect

Asymmetric main memories

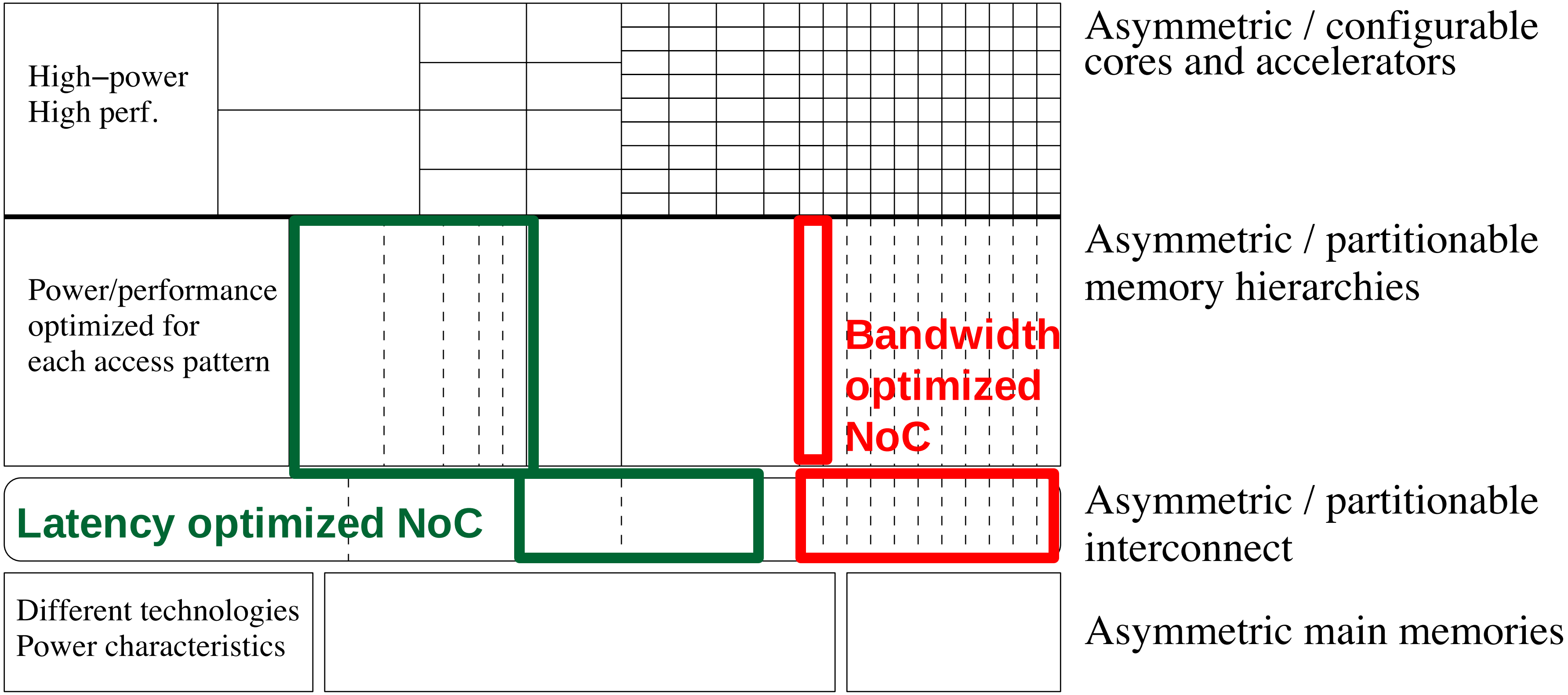
- Execute each code block on the most efficient execution backend for that block [Fallin+ ICCD'14]
  - Enables a much more efficient and still high performance core design

Exploiting Asymmetry: Simple Examples



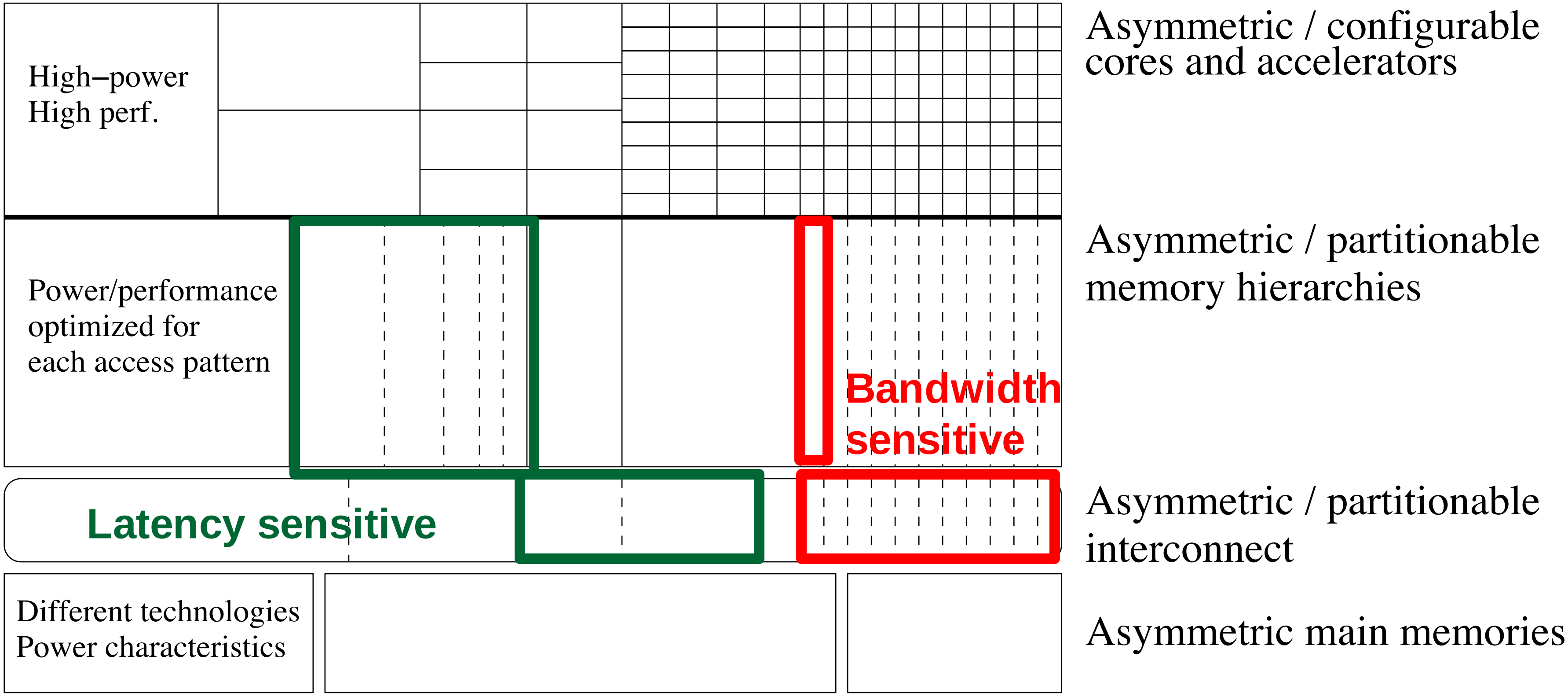
- Execute streaming "memory phases" on streaming-optimized cores and memory hierarchies
    - More efficient and higher performance than general purpose hierarchy

Exploiting Asymmetry: Simple Examples


High-power
High perf.
Asymmetric / configurable cores and accelerators
Power/performance optimized for each access pattern
Bandwidth optimized NoC
Asymmetric / partitionable memory hierarchies
Latency optimized NoC
Asymmetric / partitionable interconnect
Different technologies
Power characteristics
Asymmetric main memories

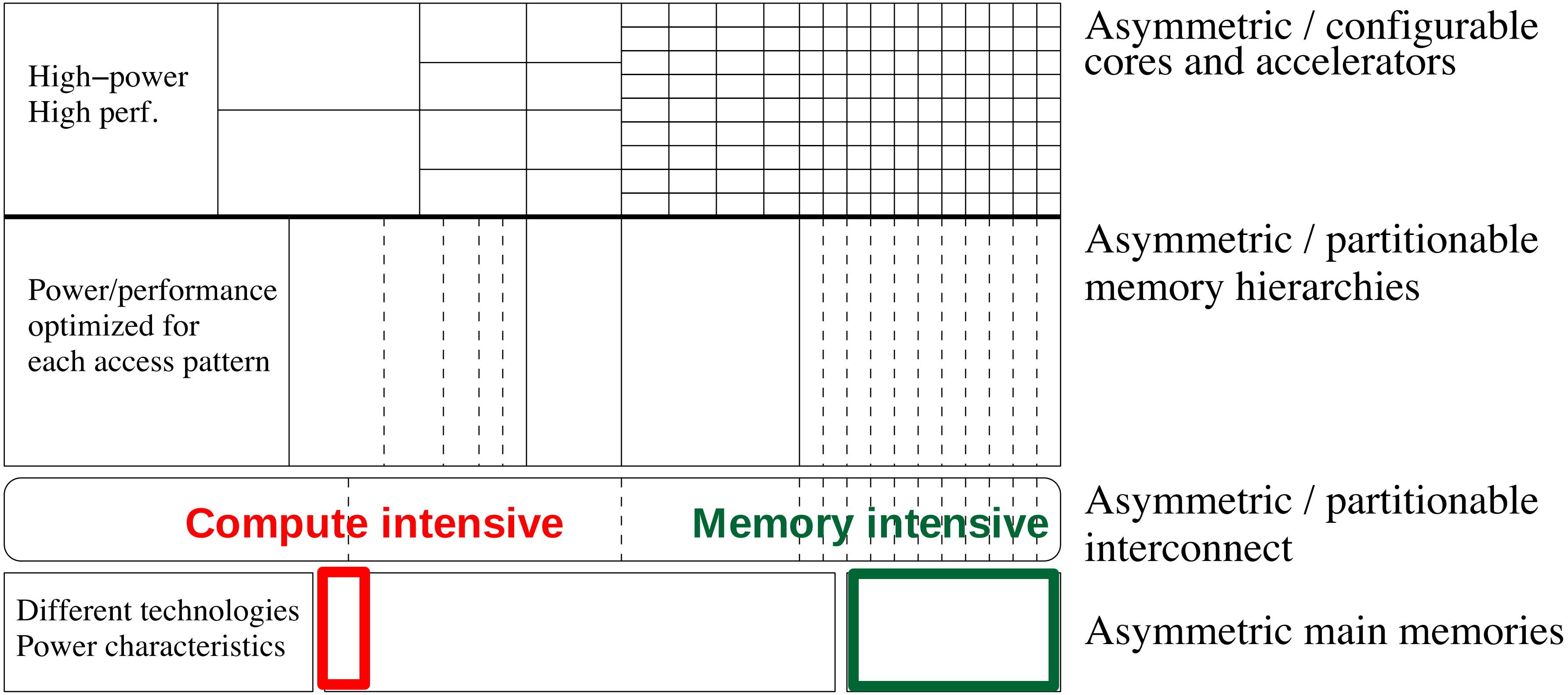
- Execute bandwidth-sensitive threads on a bandwidth-optimized network, latency-sensitive ones on a latency-optimized network [Das+ DAC'13]
  - Higher performance and energy-efficiency than a single network

Exploiting Asymmetry: Simple Examples


High–power
High perf.
Asymmetric / configurable cores and accelerators
Power/performance optimized for each access pattern
Bandwidth sensitive
Asymmetric / partitionable memory hierarchies
Latency sensitive
Asymmetric / partitionable interconnect
Different technologies
Power characteristics
Asymmetric main memories

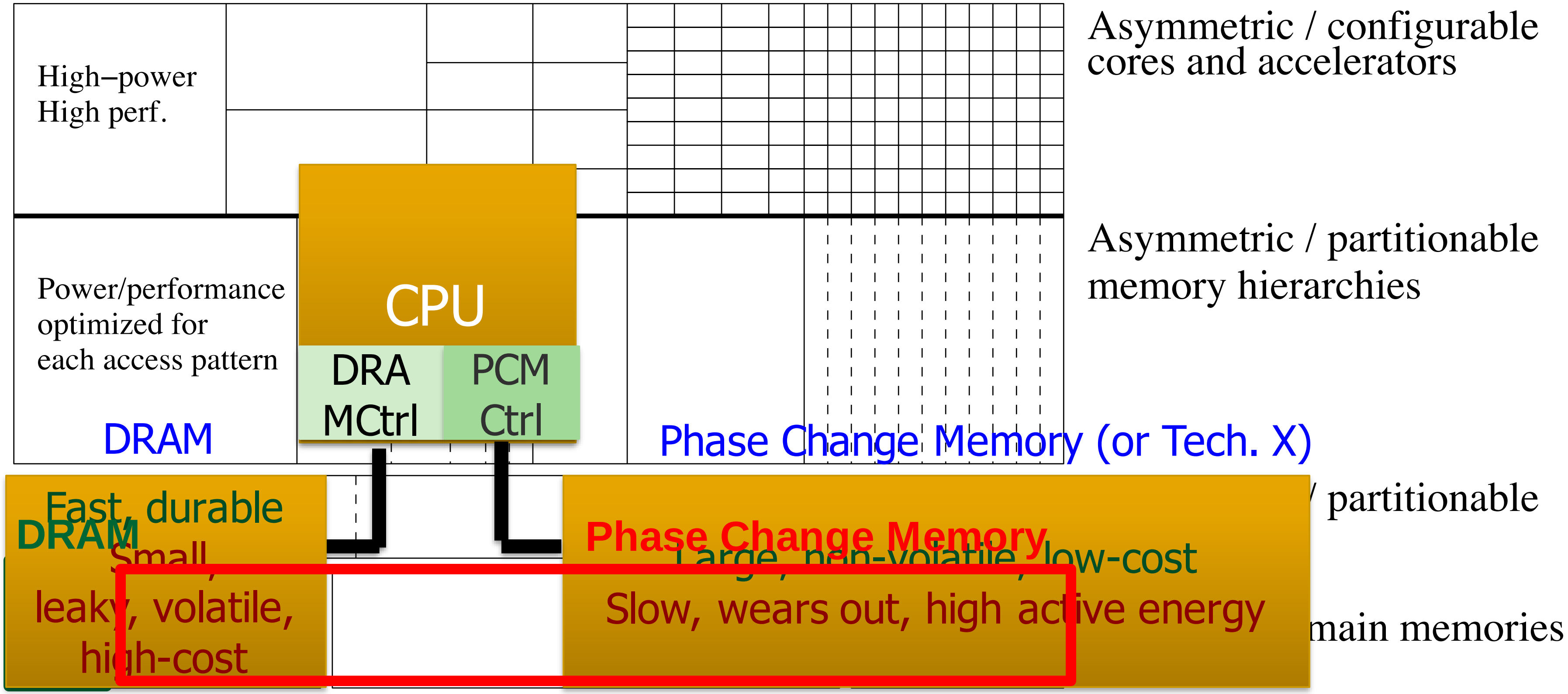
- Partition memory controller and on-chip network bandwidth asymmetrically among threads [Kim+ HPCA 2010, MICRO 2010, Top Picks 2011] [Nychis+ HotNets 2010] [Das+ MICRO 2009, ISCA 2010, Top Picks 2011]
  - Higher performance and energy-efficiency than symmetric/free-for-all

Exploiting Asymmetry: Simple Examples


High–power
High perf.
Asymmetric / configurable cores and accelerators
Power/performance optimized for each access pattern
Asymmetric / partitionable memory hierarchies
Compute intensive
Memory intensive
Asymmetric / partitionable interconnect
Different technologies
Power characteristics
Asymmetric main memories

- Have multiple different memory scheduling policies apply them to different sets of threads based on thread behavior **[Kim+ MICRO 2010, Top Picks 2011] [Ausavarungnirun+ ISCA 2012]**
  - Higher performance and fairness than a homogeneous policy

Exploiting Asymmetry: Simple Examples



High–power
High perf.

Asymmetric / configurable cores and accelerators

Power/performance optimized for each access pattern

Asymmetric / partitionable memory hierarchies

CPU

DRAMCtrl

PCM Ctrl

DRAM

Phase Change Memory (or Tech. X)

/ partitionable

main memories

DRAM

Fast, durable

Small,

leaky, volatile,

high-cost

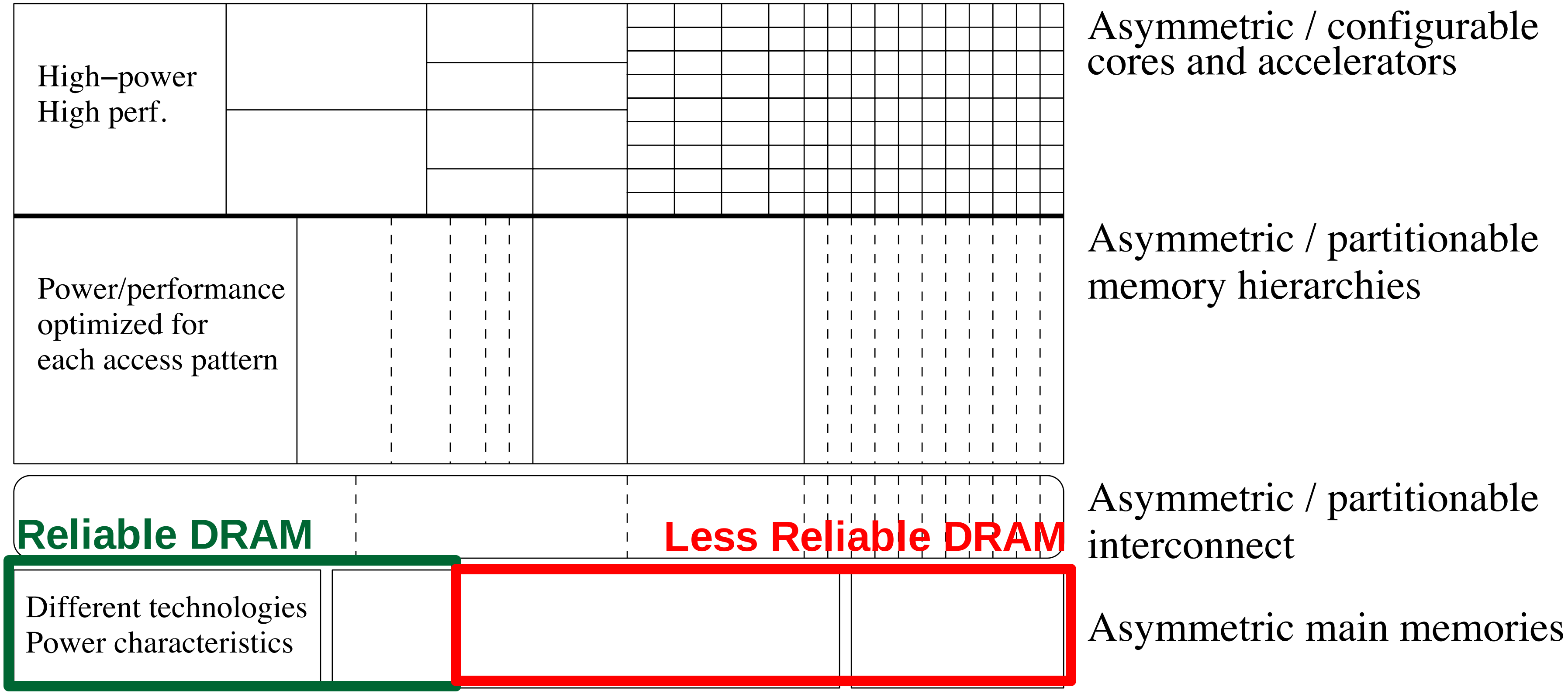
Phase Change Memory

Large, non-volatile, low-cost

Slow, wears out, high active energy

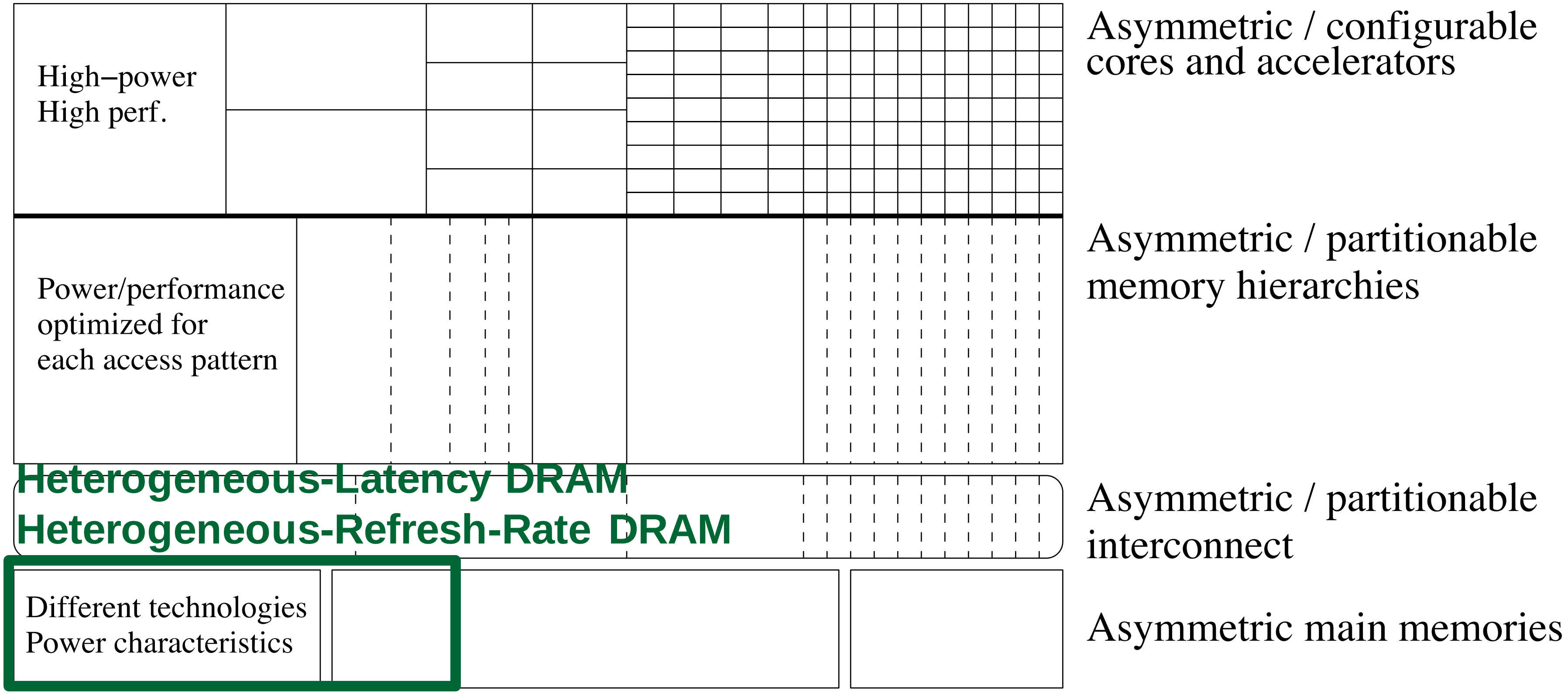
- Build main memory with different technologies with different characteristics (e.g., latency, bandwidth, cost, energy, reliability)
  [Meza+ IEEE CAL'12, Yoon+ ICCD'12, Luo+ DSN'14]
  - Higher performance and energy-efficiency than homogeneous memory

Exploiting Asymmetry: Simple Examples


High–power
High perf.
Asymmetric / configurable cores and accelerators
Power/performance optimized for each access pattern
Asymmetric / partitionable memory hierarchies
Reliable DRAM
Less Reliable DRAM
Asymmetric / partitionable interconnect
Different technologies Power characteristics
Asymmetric main memories

- Build main memory with different technologies with different characteristics (e.g., latency, bandwidth, cost, energy, reliability) **[Meza+ IEEE CAL'12, Yoon+ ICCD'12, Luo+ DSN'14]**
  - Lower-cost than homogeneous-reliability memory at same availability

Exploiting Asymmetry: Simple Examples



High–power
High perf.

Asymmetric / configurable cores and accelerators

Power/performance
optimized for
each access pattern

Asymmetric / partitionable memory hierarchies

Heterogeneous-Latency DRAM

Heterogeneous-Refresh-Rate DRAM

Asymmetric / partitionable interconnect

Different technologies
Power characteristics

Asymmetric main memories

- Design each memory chip to be heterogeneous to achieve low latency and low energy at reasonably low cost **[Lee+ HPCA'13, Liu+ ISCA'12]**
  - Higher performance and energy-efficiency than single-level memory

Some Readings

- Suleman et al., "Accelerating Critical Section Execution with Asymmetric Multi-Core Architectures," ASPLOS 2009, IEEE Micro Top Picks 2010.
- Joao et al., "Bottleneck Identification and Scheduling in Multithreaded Applications," ASPLOS 2012.
- Joao et al., "Utility-Based Acceleration of Multithreaded Applications on Asymmetric CMPs," ISCA 2013.
- Suleman et al., "Data Marshaling for Multi-Core Architectures," ISCA 2010, IEEE Micro Top Picks 2011.
- Grochowski et al., "Best of Both Latency and Throughput," ICCD 2004.

Multiprocessors

Readings: Multiprocessing

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

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

Memory Consistency

- Required
  - Lamport, "How to Make a Multiprocessor Computer That Correctly Executes Multiprocess Programs," IEEE Transactions on Computers, 1979

Readings: Cache Coherence

- Required
  - Papamarcos and Patel, "A low-overhead coherence solution for multiprocessors with private cache memories," ISCA 1984.

- Recommended:
  - Culler and Singh, *Parallel Computer Architecture*
    - Chapter 5.1 (pp 269 – 283), Chapter 5.3 (pp 291 – 305)
  - P&H, *Computer Organization and Design*
    - Chapter 5.8 (pp 534 – 538 in 4th and 4th revised eds.)

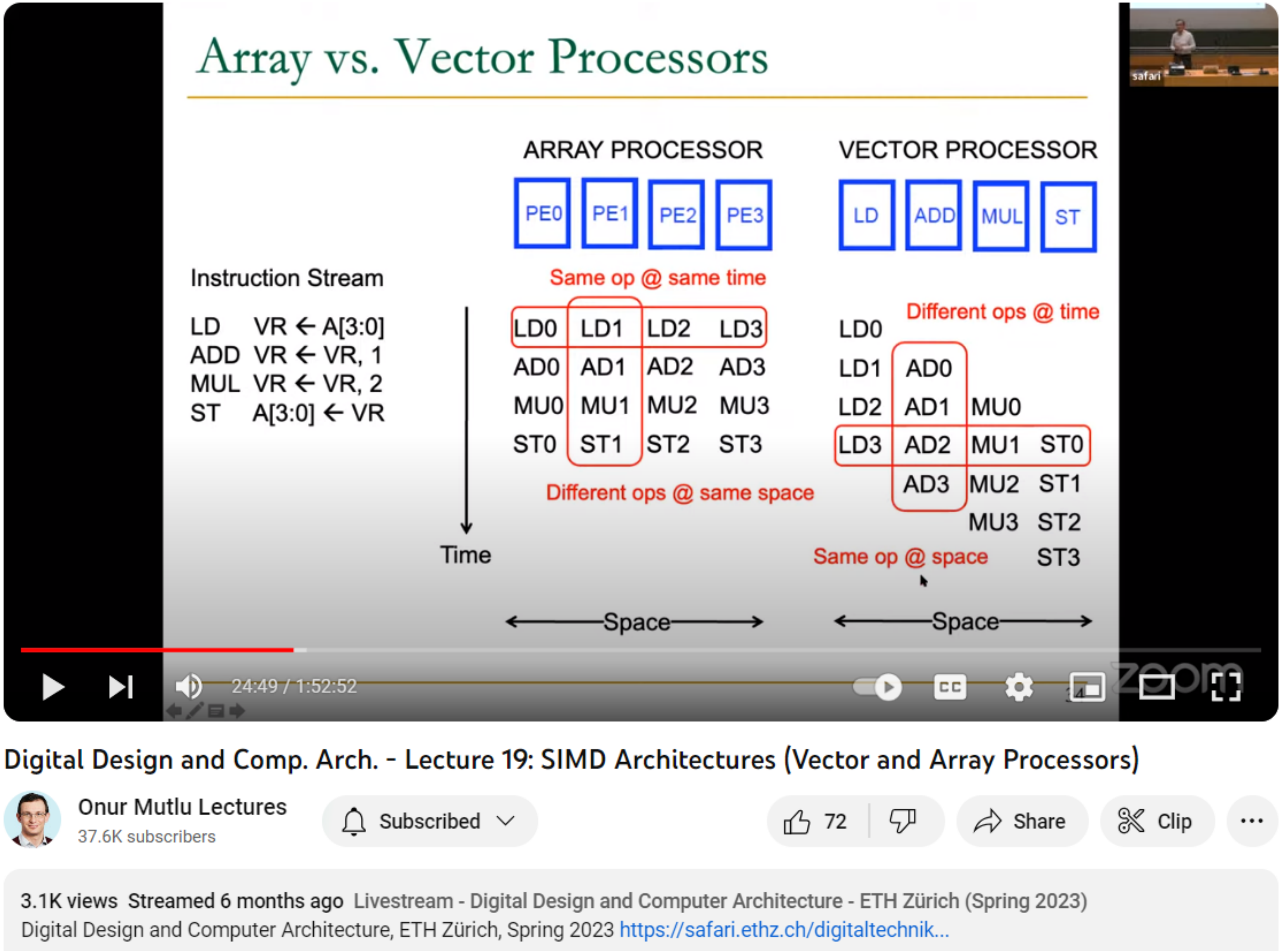
Multiprocessors and Issues in Multiprocessing

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

SIMD Example: Vector & Array Processors



Array vs. Vector Processors

Instruction Stream

```
LD   VR ← A[3:0]
ADD  VR ← VR, 1
MUL  VR ← VR, 2
ST   A[3:0] ← VR
```

ARRAY PROCESSOR

Same op @ same time

PE0	PE1	PE2	PE3
LD0	LD1	LD2	LD3
AD0	AD1	AD2	AD3
MU0	MU1	MU2	MU3
ST0	ST1	ST2	ST3

Different ops @ same space

VECTOR PROCESSOR

Different ops @ time

LD	ADD	MUL	ST
LD0			
LD1	AD0		
LD2	AD1	MU0	
LD3	AD2	MU1	ST0
	AD3	MU2	ST1
		MU3	ST2
			ST3

Same op @ space

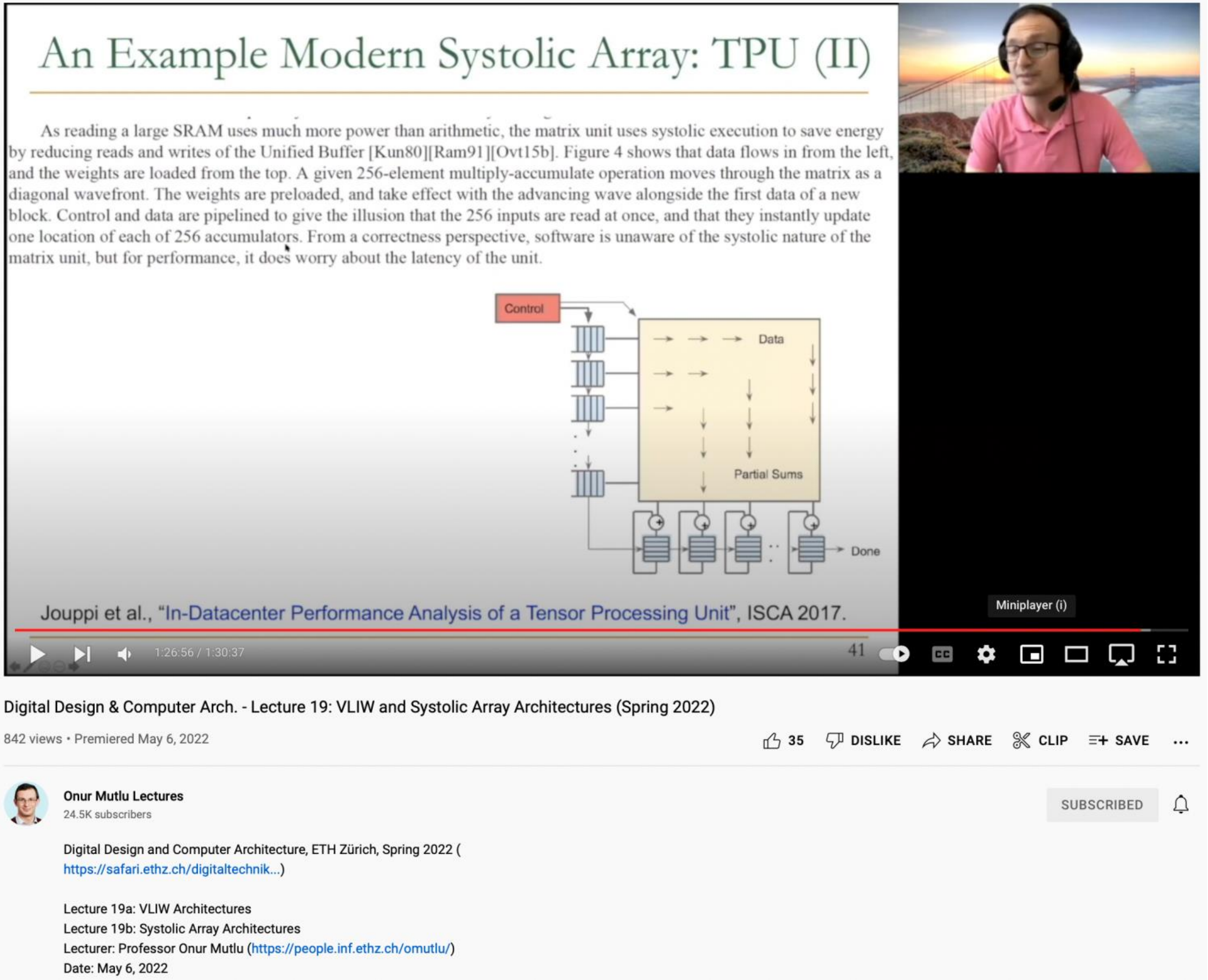
Digital Design and Comp. Arch. - Lecture 19: SIMD Architectures (Vector and Array Processors)

Onur Mutlu Lectures
37.6K subscribers

3.1K views Streamed 6 months ago Livestream - Digital Design and Computer Architecture - ETH Zürich (Spring 2023)
Digital Design and Computer Architecture, ETH Zürich, Spring 2023 https://safari.ethz.ch/digitaltechnik...

https://www.youtube.com/watch?v=gkMaO3yJMz0

MISD Example: Systolic Arrays



An Example Modern Systolic Array: TPU (II)

As reading a large SRAM uses much more power than arithmetic, the matrix unit uses systolic execution to save energy by reducing reads and writes of the Unified Buffer [Kun80][Ram91][Ovt15b]. Figure 4 shows that data flows in from the left, and the weights are loaded from the top. A given 256-element multiply-accumulate operation moves through the matrix as a diagonal wavefront. The weights are preloaded, and take effect with the advancing wave alongside the first data of a new block. Control and data are pipelined to give the illusion that the 256 inputs are read at once, and that they instantly update one location of each of 256 accumulators. From a correctness perspective, software is unaware of the systolic nature of the matrix unit, but for performance, it does worry about the latency of the unit.

Jouppi et al., "In-Datacenter Performance Analysis of a Tensor Processing Unit", ISCA 2017.

Digital Design & Computer Arch. - Lecture 19: VLIW and Systolic Array Architectures (Spring 2022)

842 views • Premiered May 6, 2022

Onur Mutlu Lectures
24.5K subscribers

Digital Design and Computer Architecture, ETH Zürich, Spring 2022 (
https://safari.ethz.ch/digitaltechnik...)

Lecture 19a: VLIW Architectures
Lecture 19b: Systolic Array Architectures
Lecturer: Professor Onur Mutlu (https://people.inf.ethz.ch/omutlu/)
Date: May 6, 2022

https://youtu.be/1SSqV7Y75oU?t=2316

Why Parallel Computers?

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

- Main (or Original) 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

Multiprocessing Fundamentals

Multiprocessor Types

- 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 Design Issues in Tightly-Coupled MP

- Shared memory synchronization
  - How to handle synchronization: locks, atomic operations, barriers

- Cache coherence
  - How to ensure correct operation in the presence of private caches keeping the same memory address cached

- Memory consistency: Ordering of all memory operations
  - What should the programmer expect the hardware to provide?

- Shared resource management

- Communication: Interconnects

Main Programming Issues in Tightly-Coupled MP

- Load imbalance
  - How to partition a single task into multiple tasks

- Synchronization
  - How to synchronize (efficiently) between tasks
  - How to communicate between tasks
  - Locks, barriers, pipeline stages, condition variables, semaphores, atomic operations, …

- Contention (avoidance & management)
- Maximizing parallelism
- Ensuring correct operation while optimizing for performance

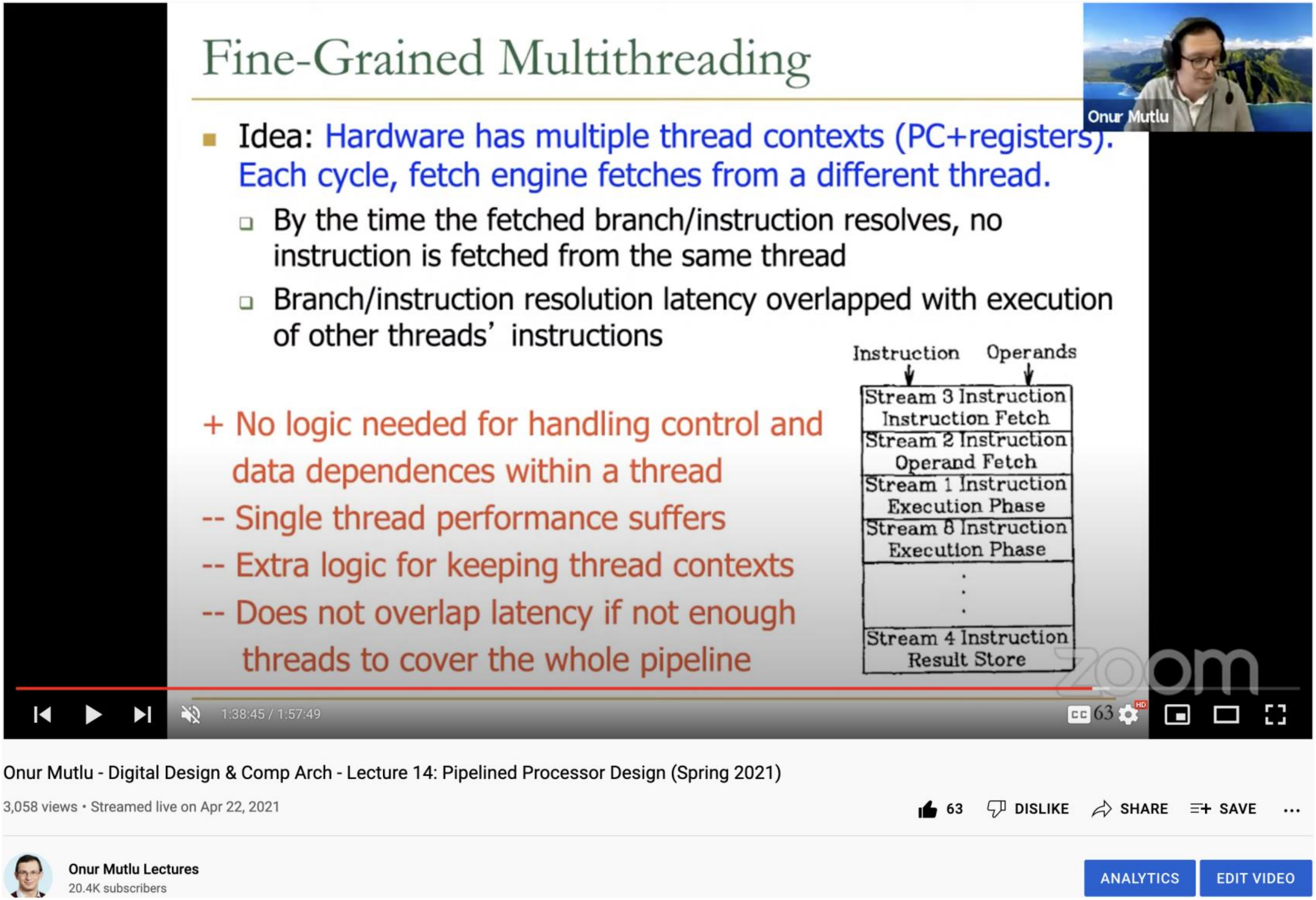
Aside: Hardware-based Multithreading

- Coarse grained
  - Quantum based
  - Event based (switch-on-event multithreading), e.g., switch on L3 miss

- 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

Lecture on Fine-Grained Multithreading



Fine-Grained Multithreading

- Idea: Hardware has multiple thread contexts (PC+registers). Each cycle, fetch engine fetches from a different thread.
  - By the time the fetched branch/instruction resolves, no instruction is fetched from the same thread
  - Branch/instruction resolution latency overlapped with execution of other threads' instructions

+ No logic needed for handling control and data dependences within a thread

-- Single thread performance suffers

-- Extra logic for keeping thread contexts

-- Does not overlap latency if not enough threads to cover the whole pipeline

Instruction Operands

Stream 3 Instruction Instruction Fetch

Stream 2 Instruction Operand Fetch

Stream 1 Instruction Execution Phase

Stream 8 Instruction Execution Phase

.
.
.

Stream 4 Instruction Result Store

Onur Mutlu - Digital Design & Comp Arch - Lecture 14: Pipelined Processor Design (Spring 2021)

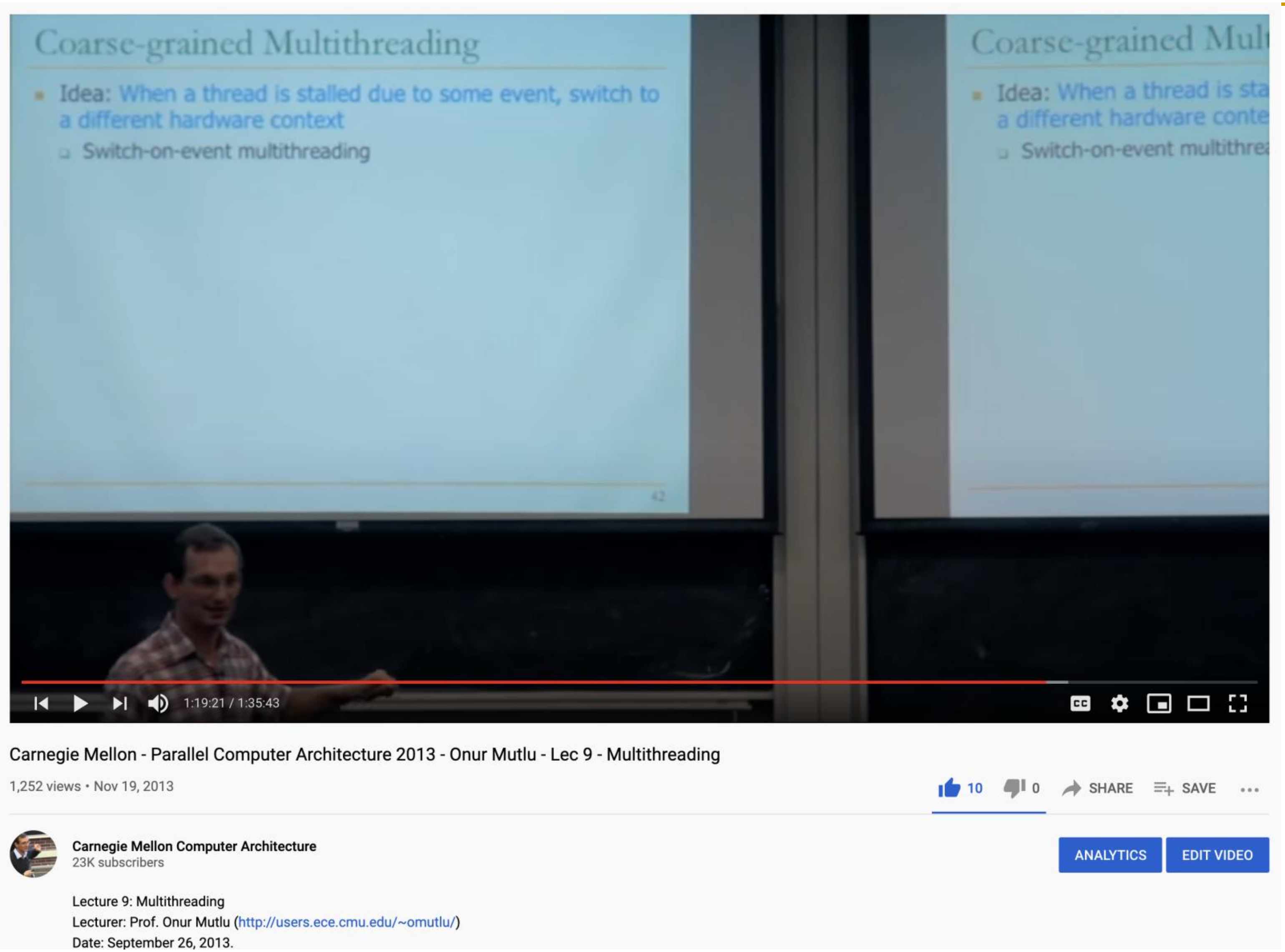
3,058 views • Streamed live on Apr 22, 2021

Onur Mutlu Lectures

20.4K subscribers

https://www.youtube.com/watch?v=6e5KZcCGBYw&list=PL5Q2soXY2Zi_uej3aY39YB5pfW4SJ7LlN&index=16

More on Multithreading (I)



Coarse-grained Multithreading

- Idea: When a thread is stalled due to some event, switch to a different hardware context
  - Switch-on-event multithreading

Carnegie Mellon - Parallel Computer Architecture 2013 - Onur Mutlu - Lec 9 - Multithreading

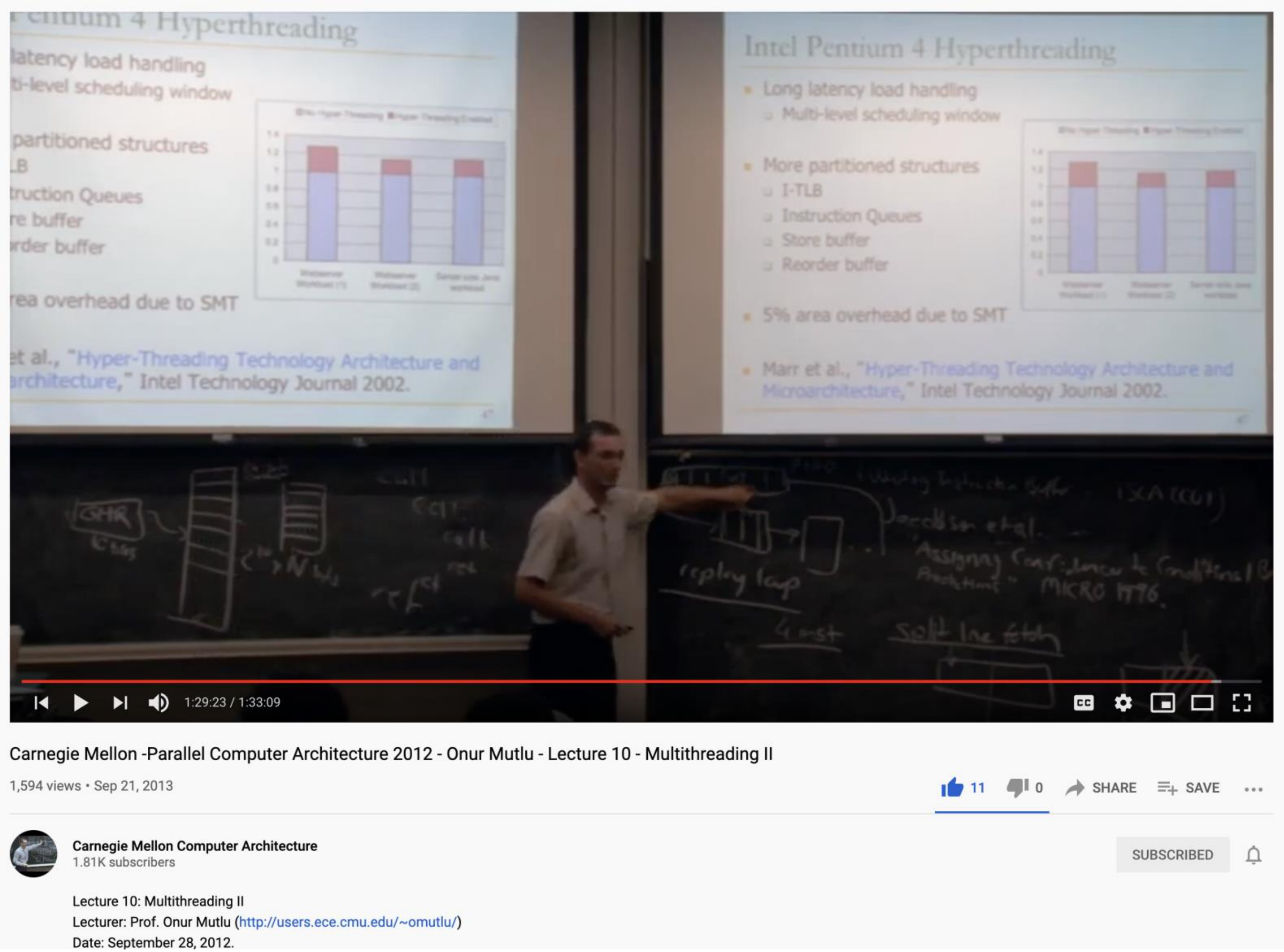
1,252 views • Nov 19, 2013

Carnegie Mellon Computer Architecture
23K subscribers

Lecture 9: Multithreading
Lecturer: Prof. Onur Mutlu (http://users.ece.cmu.edu/~omutlu/)
Date: September 26, 2013.

https://www.youtube.com/onurmutlulectures

More on Multithreading (II)


Intel Pentium 4 Hyperthreading
Long latency load handling
Multi-level scheduling window
More partitioned structures
I-TLB
Instruction Queues
Store buffer
Reorder buffer
5% area overhead due to SMT
Marr et al., "Hyper-Threading Technology Architecture and Microarchitecture," Intel Technology Journal 2002.
replay loop
MICRO 1996
1:29:23 / 1:33:09
11
0
SHARE
SAVE
SUBSCRIBED

Carnegie Mellon -Parallel Computer Architecture 2012 - Onur Mutlu - Lecture 10 - Multithreading II

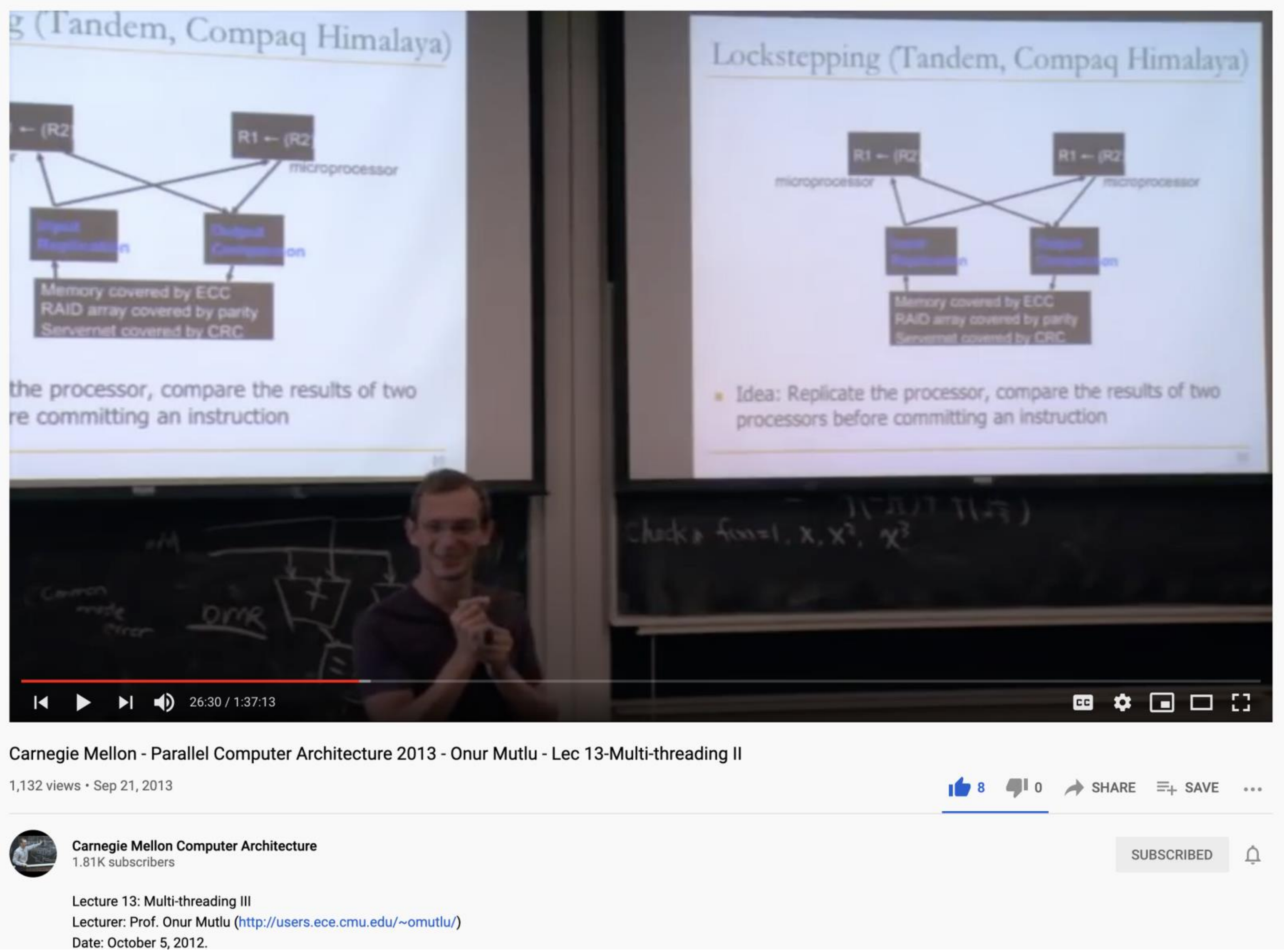
1,594 views • Sep 21, 2013

Carnegie Mellon Computer Architecture
1.81K subscribers

Lecture 10: Multithreading II
Lecturer: Prof. Onur Mutlu (http://users.ece.cmu.edu/~omutlu/)
Date: September 28, 2012.

https://www.youtube.com/onurmutlulectures

More on Multithreading (III)


Lockstepping (Tandem, Compaq Himalaya)
R1 ← (R2
microprocessor
Input Replication
Output Comparison
Memory covered by ECC
RAID array covered by parity
Servernet covered by CRC
Idea: Replicate the processor, compare the results of two processors before committing an instruction
26:30 / 1:37:13
8
0
SHARE
SAVE
SUBSCRIBED

Carnegie Mellon - Parallel Computer Architecture 2013 - Onur Mutlu - Lec 13-Multi-threading II

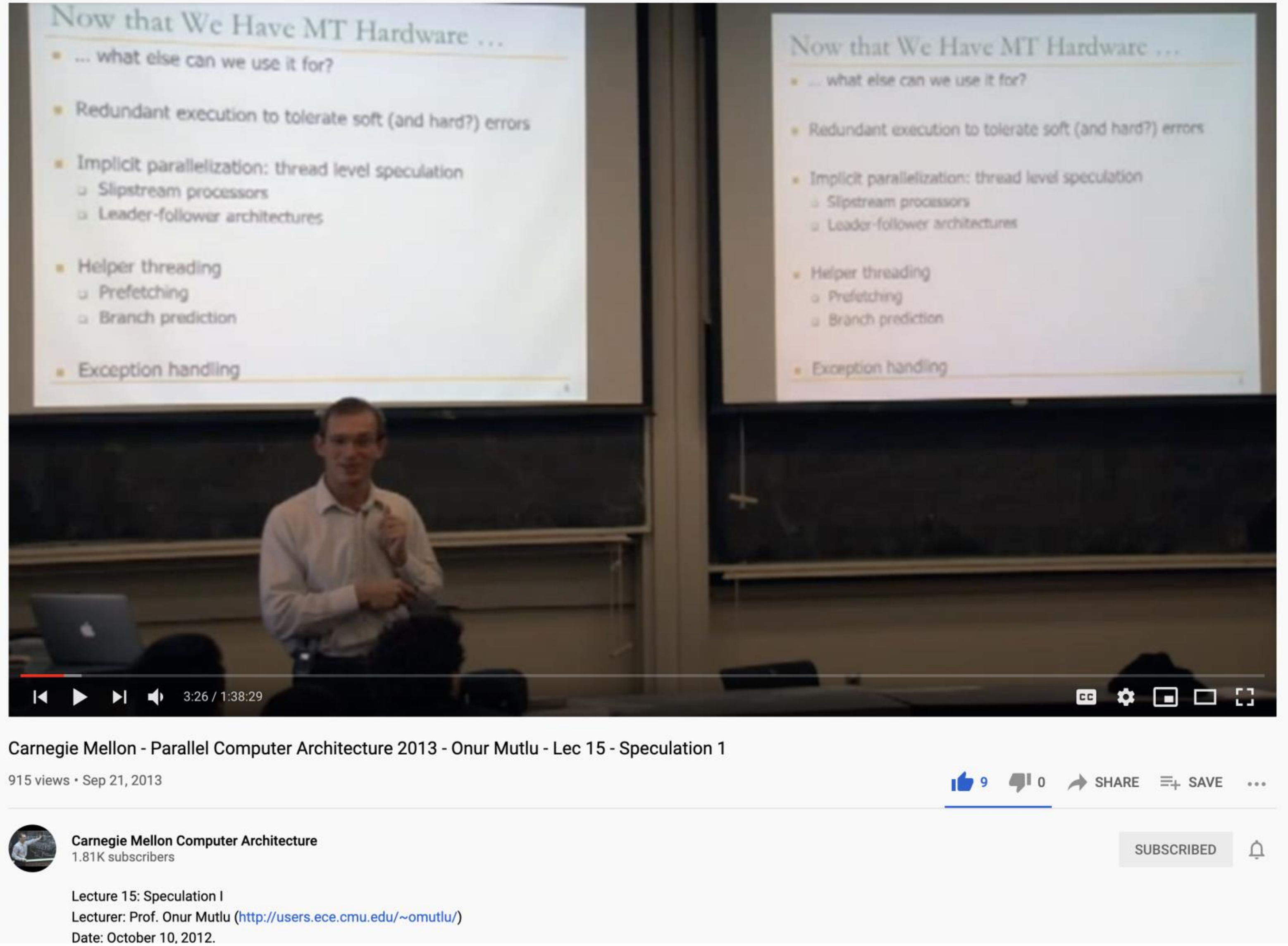
1,132 views • Sep 21, 2013

Carnegie Mellon Computer Architecture
1.81K subscribers

Lecture 13: Multi-threading III
Lecturer: Prof. Onur Mutlu (http://users.ece.cmu.edu/~omutlu/)
Date: October 5, 2012.

https://www.youtube.com/onurmutlulectures

More on Multithreading (IV)



Now that We Have MT Hardware …

- … what else can we use it for?
- Redundant execution to tolerate soft (and hard?) errors
- Implicit parallelization: thread level speculation
  - Slipstream processors
  - Leader-follower architectures
- Helper threading
  - Prefetching
  - Branch prediction
- Exception handling

Now that We Have MT Hardware …

- … what else can we use it for?
- Redundant execution to tolerate soft (and hard?) errors
- Implicit parallelization: thread level speculation
  - Slipstream processors
  - Leader-follower architectures
- Helper threading
  - Prefetching
  - Branch prediction
- Exception handling

Carnegie Mellon - Parallel Computer Architecture 2013 - Onur Mutlu - Lec 15 - Speculation 1

915 views • Sep 21, 2013

Carnegie Mellon Computer Architecture
1.81K subscribers

Lecture 15: Speculation I
Lecturer: Prof. Onur Mutlu (http://users.ece.cmu.edu/~omutlu/)
Date: October 10, 2012.

https://www.youtube.com/onurmutlulectures

Lectures on Multithreading

- Parallel Computer Architecture, Fall 2012, Lecture 9
  - Multithreading I (CMU, Fall 2012)
  - https://www.youtube.com/watch?v=iqi9wFqFiNU&list=PL5PHm2jkkXmgDN1PLwOY_tGtUlynnyV6D&index=51
- Parallel Computer Architecture, Fall 2012, Lecture 10
  - Multithreading II (CMU, Fall 2012)
  - https://www.youtube.com/watch?v=e8lfl6MbILg&list=PL5PHm2jkkXmgDN1PLwOY_tGtUlynnyV6D&index=52
- Parallel Computer Architecture, Fall 2012, Lecture 13
  - Multithreading III (CMU, Fall 2012)
  - https://www.youtube.com/watch?v=7vkDpZ1-hHM&list=PL5PHm2jkkXmgDN1PLwOY_tGtUlynnyV6D&index=53
- Parallel Computer Architecture, Fall 2012, Lecture 15
  - Speculation I (CMU, Fall 2012)
  - https://www.youtube.com/watch?v=-hbmzIDe0sA&list=PL5PHm2jkkXmgDN1PLwOY_tGtUlynnyV6D&index=54

Limits of Parallel Speedup

Parallel Speedup Example

- $a_4x^4 + a_3x^3 + a_2x^2 + a_1x + a_0$

- Assume given inputs: x and each a_i

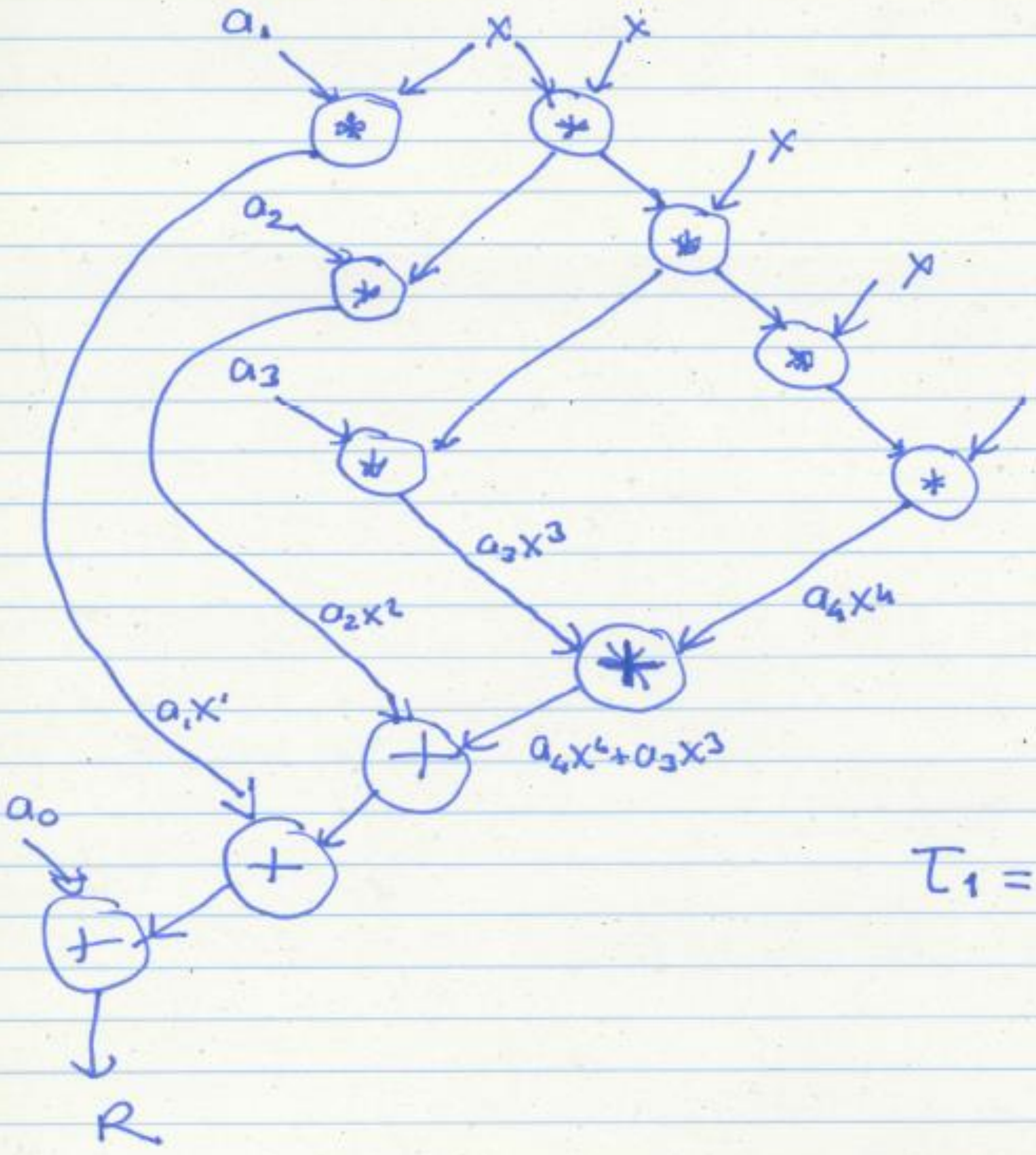
- Assume each operation 1 cycle, no communication cost, each op can be executed in a different processor

- How fast is this with a single processor?
  - Assume no pipelining or concurrent execution of instructions

- How fast is this with 3 processors?

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

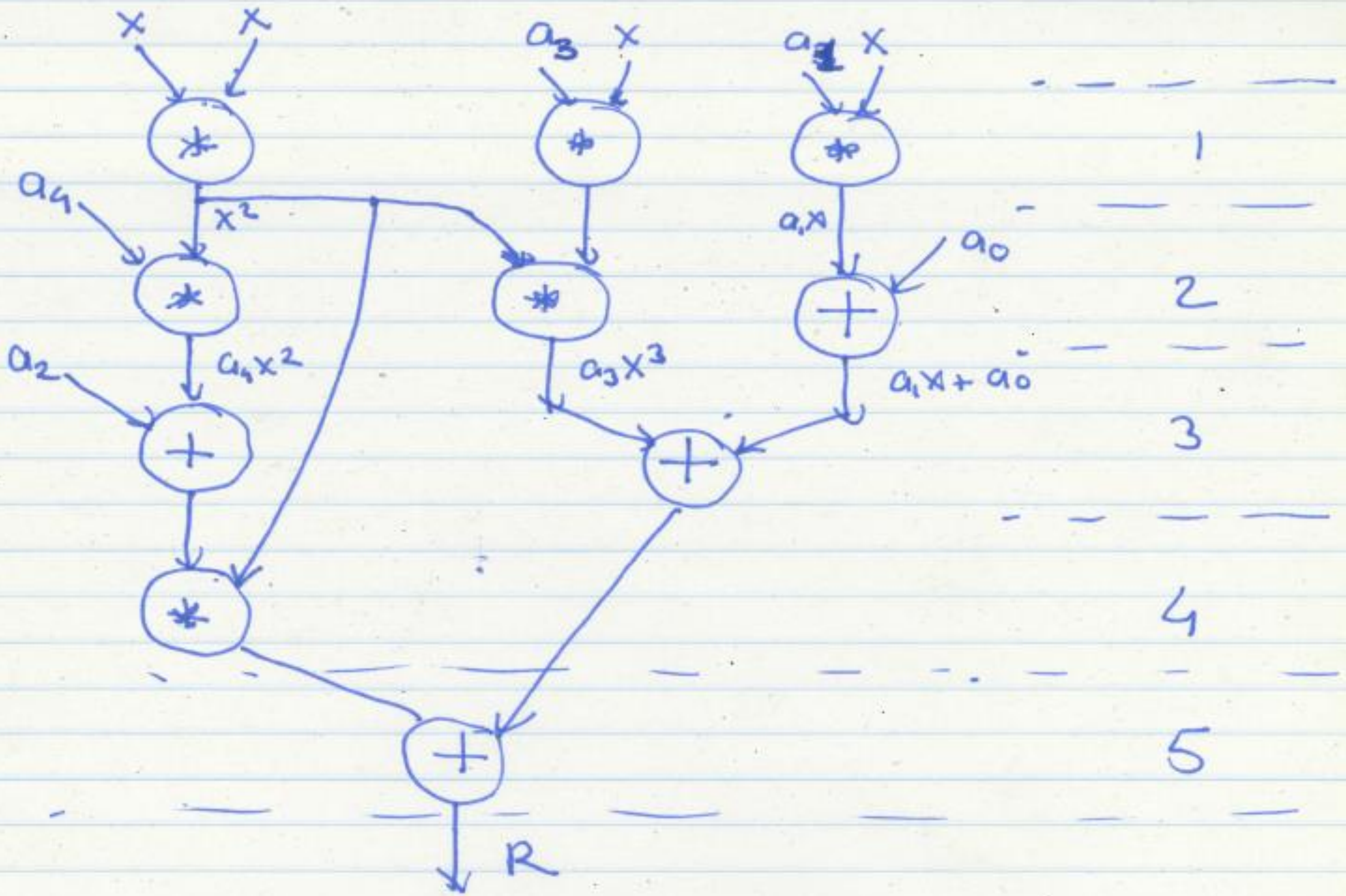
Single processor : 11 operations (DRAW the data flow graph)



$T_1 = 11$ cycles

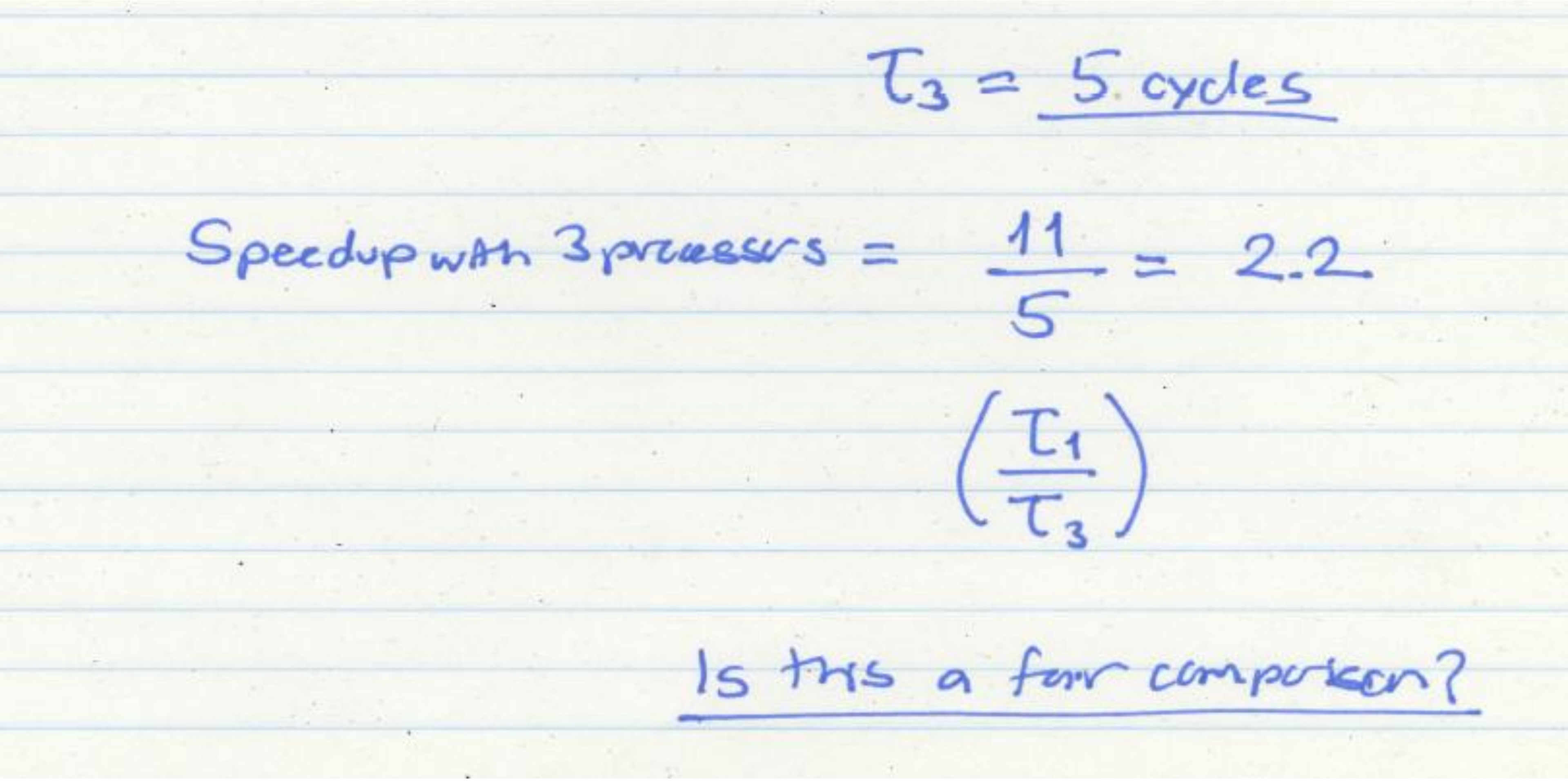
$R = a_4 x^4 + a_3 x^3 + a_2 x^2 + a_1 x + a_0$

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



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

Speedup with 3 Processors



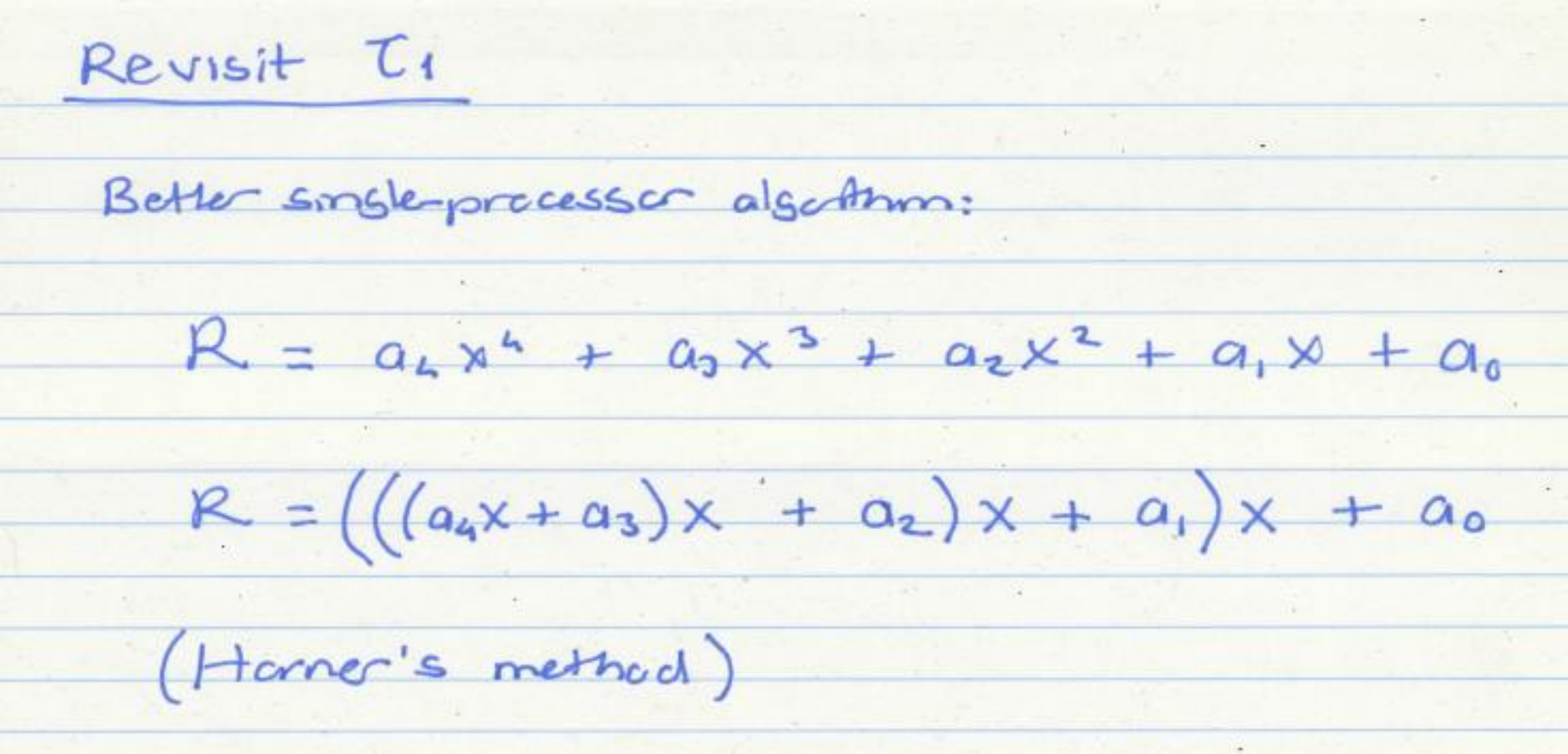
$\tau_3 = \underline{5 \text{ cycles}}$

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

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

Is this a fair comparison?

Revisiting the Single-Processor Algorithm



Revisit τ_1

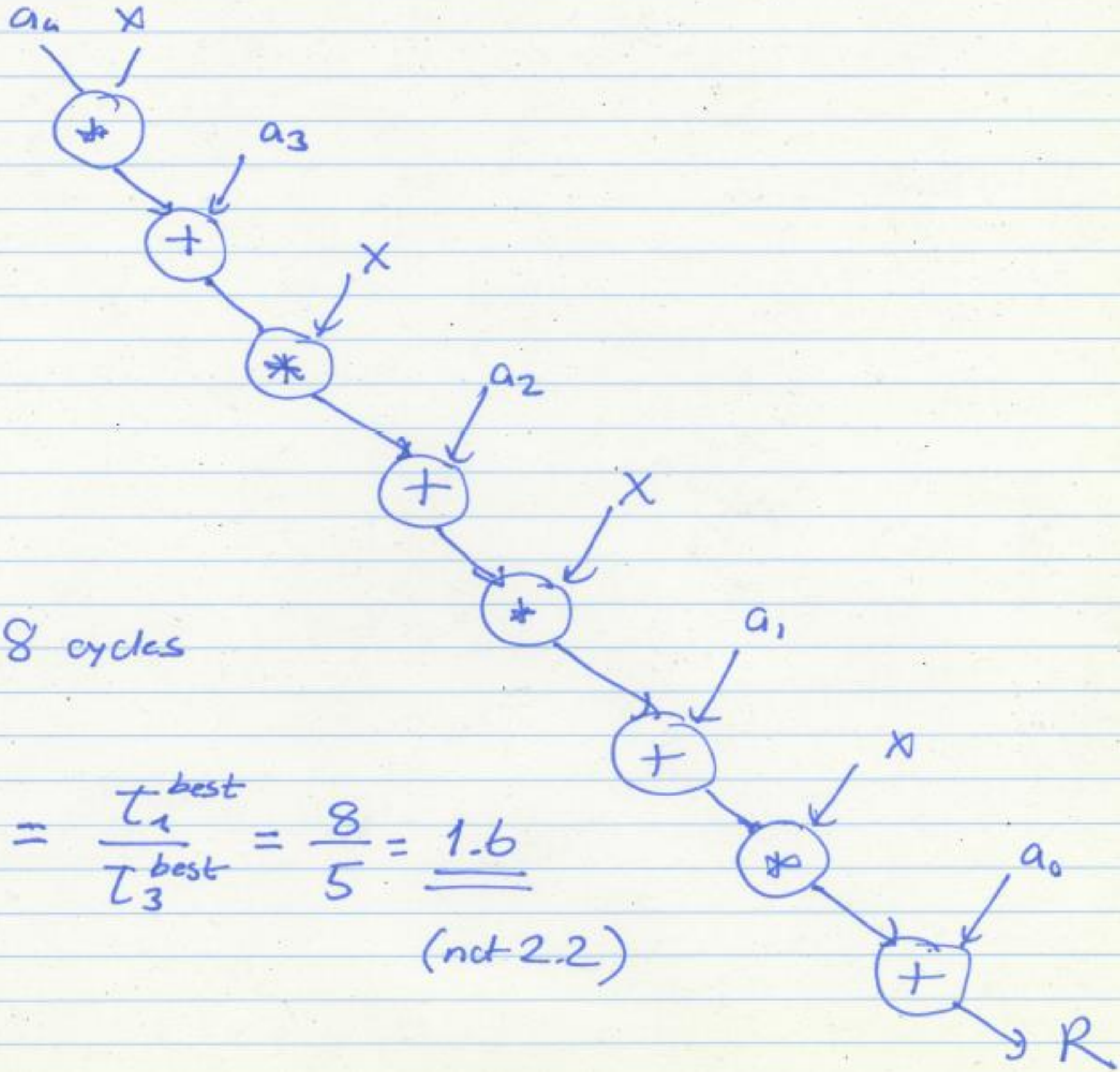
Better single-processor algorithm:

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

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

(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. $= \frac{\tau_1^{best}}{\tau_3^{best}} = \frac{8}{5} = \underline{\underline{1.6}}$

(not 2.2)

Superlinear Speedup

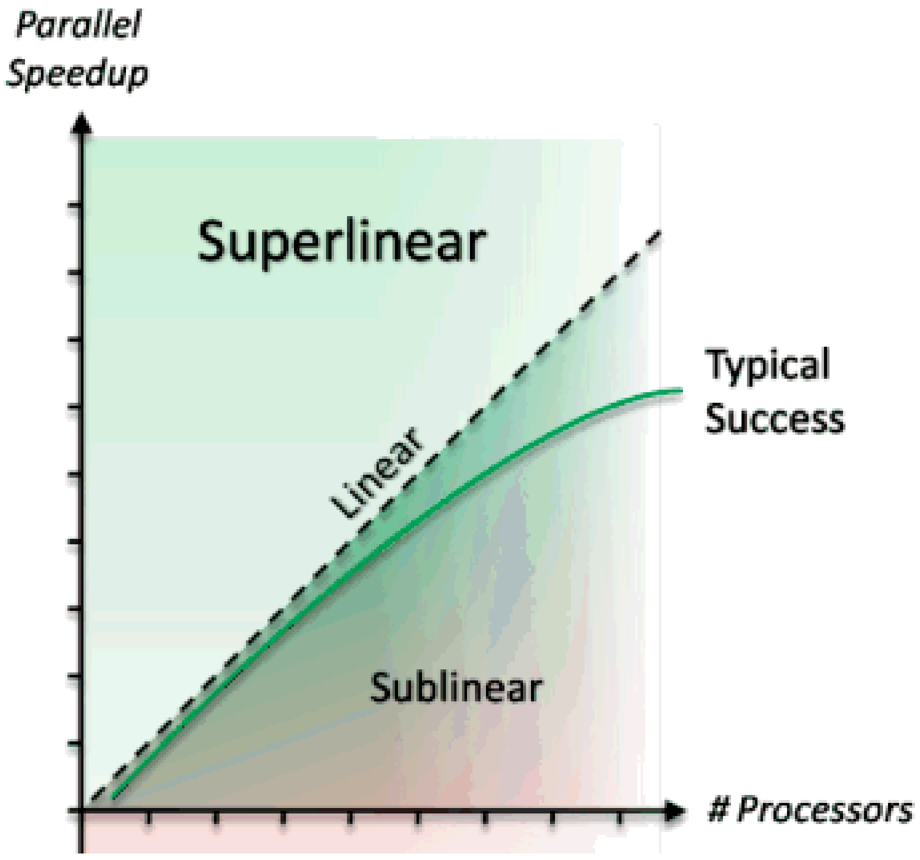
- Can speedup be greater than P with P processing elements?

- Unfair comparisons

  Compare best parallel algorithm to wimpy serial algorithm → unfair

- Cache/memory effects

  More processors → more cache or memory → fewer misses in cache/mem



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)

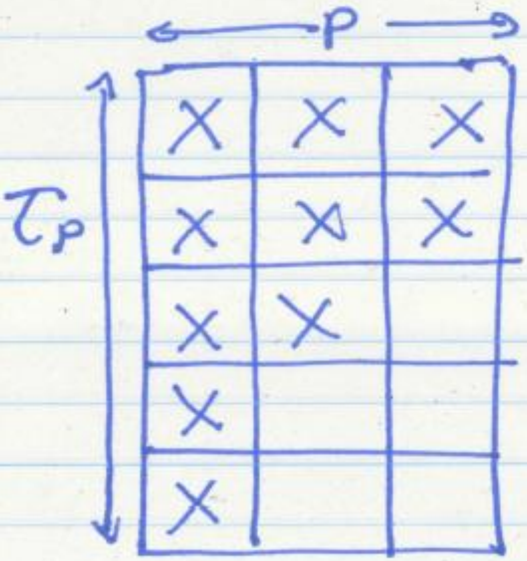
- Redundancy: how much extra work is done with parallel processing
  - R = (# of operations in parallel version) / (# operations in best single processor algorithm version)

- Efficiency
  - E = (Time with 1 processor) / (processors x Time with P processors)
  - E = U/R

Utilization of a 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

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

R is always ≥ 1

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

$$E = \frac{1 \cdot \tau_1^{best}}{p \cdot \tau_p^{best}}$$

(tying up 1 proc for T_1 time units)

(tying up p proc. for T_p time units)

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

Amdahl's Law and Caveats of Parallelism

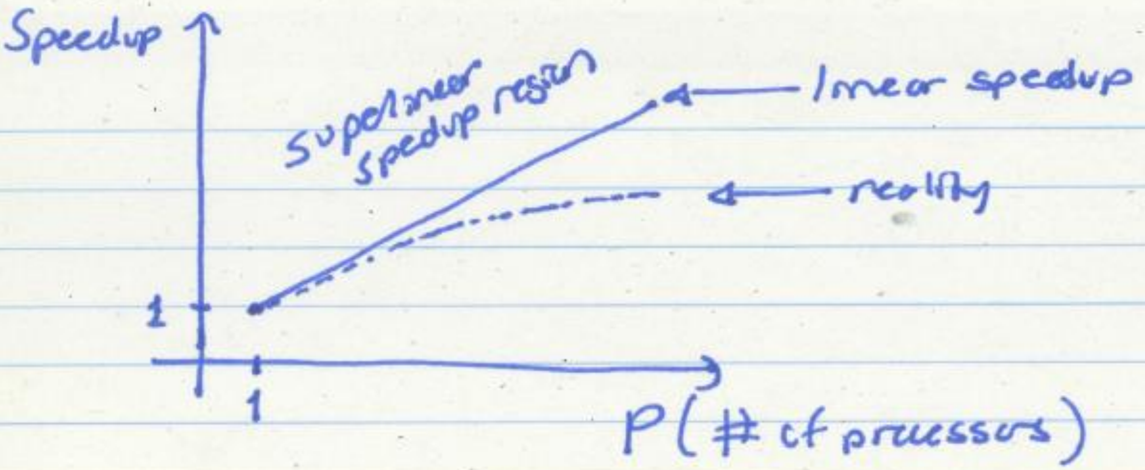
Amdahl's Law

- 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

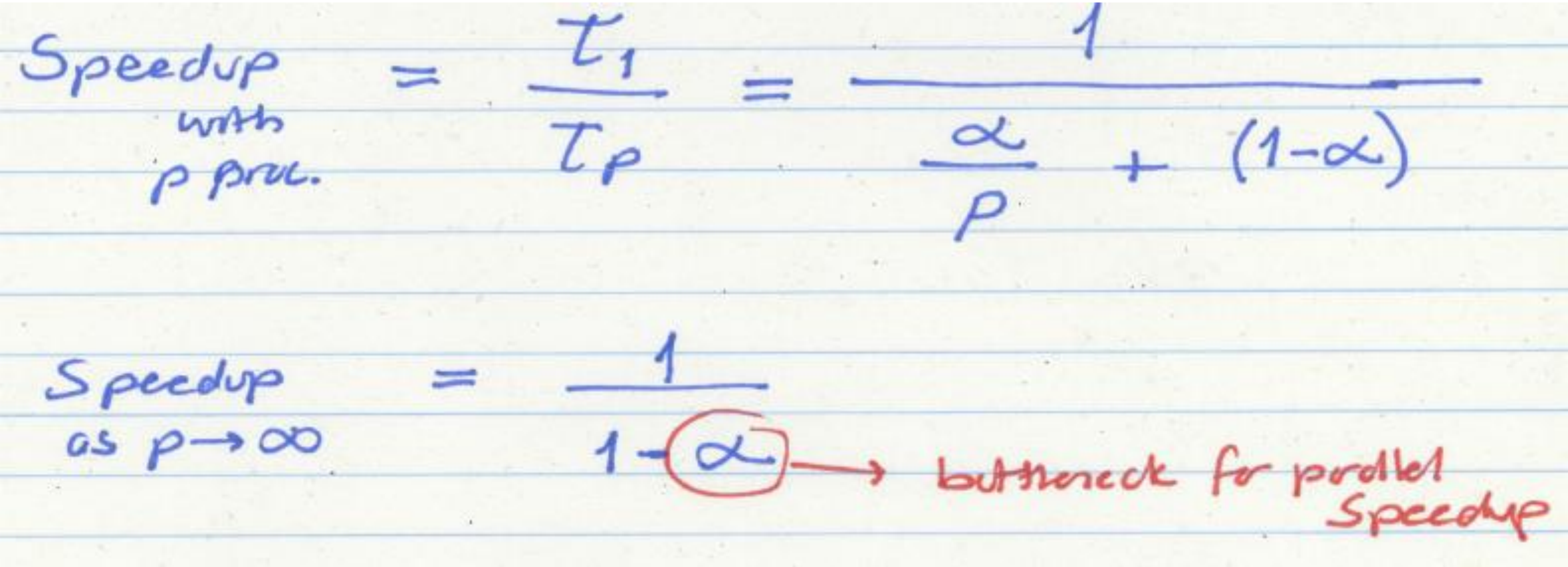
Caveats of Parallelism (I)



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



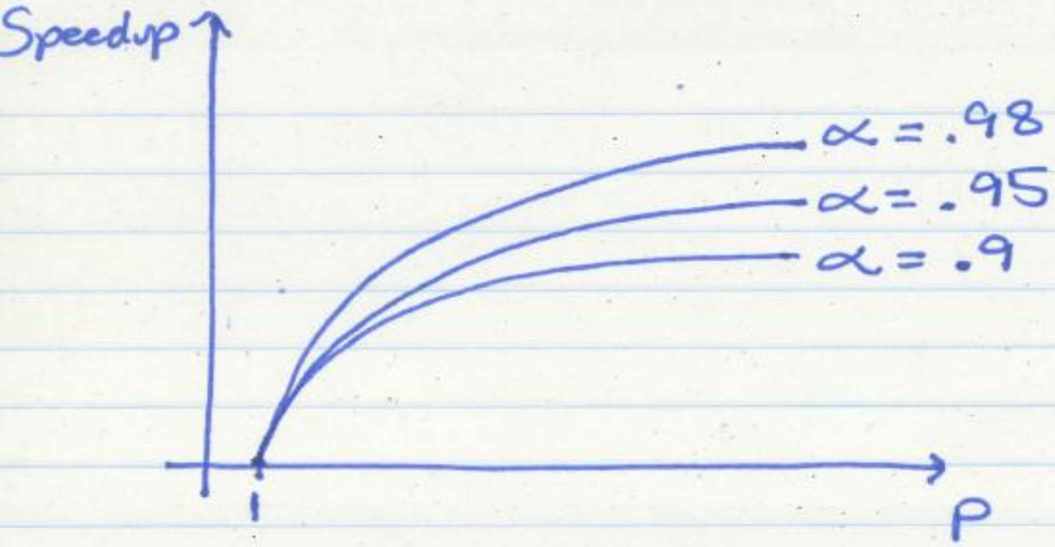
$$\underset{\text{with } p \text{ proc.}}{\text{Speedup}} = \frac{t_1}{T_p} = \frac{1}{\frac{\alpha}{p} + (1-\alpha)}$$

$$\underset{\text{as } p\to\infty}{\text{Speedup}} = \frac{1}{1-\boxed{\alpha}}$$

→ bottleneck for parallel speedup

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

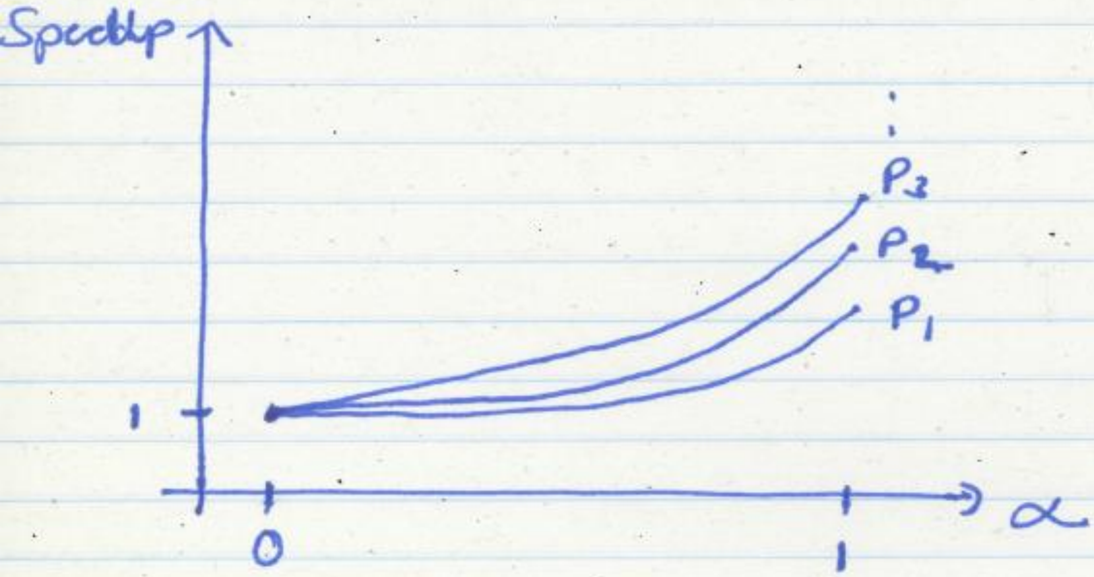
Amdahl's Law Implication 1



Amdahl's Law illustrated

Adding more and more processors gives less & less benefit if $\alpha < 1$

Amdahl’s Law Implication 2



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

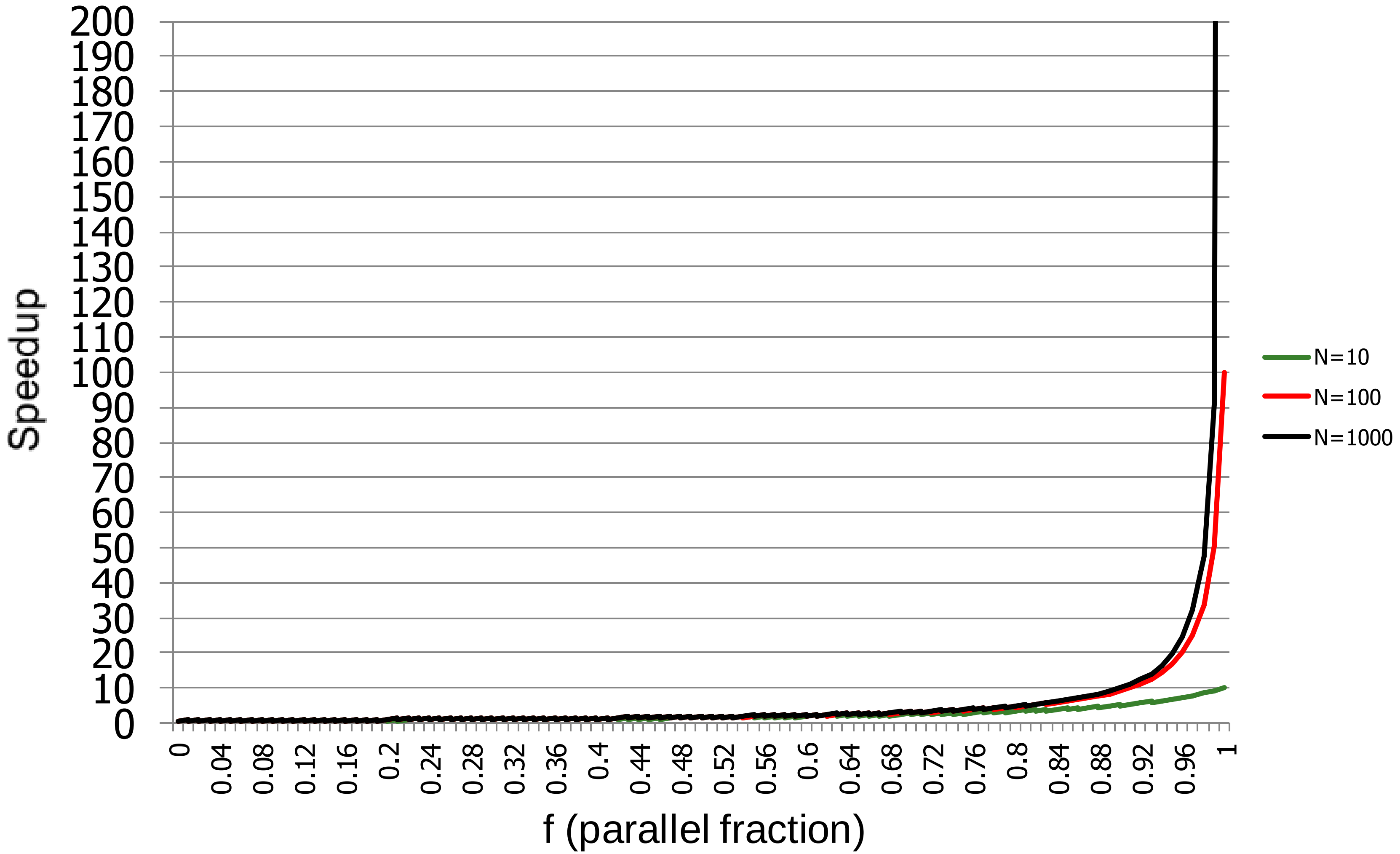
Caveats of Parallelism (II)

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

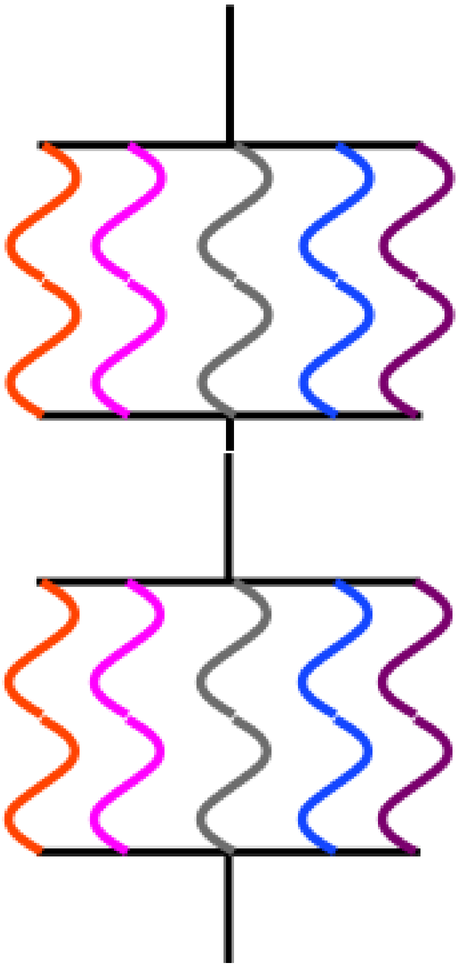
$$\text{Speedup} = \frac{1}{1 - f + \frac{f}{N}}$$

  - 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?



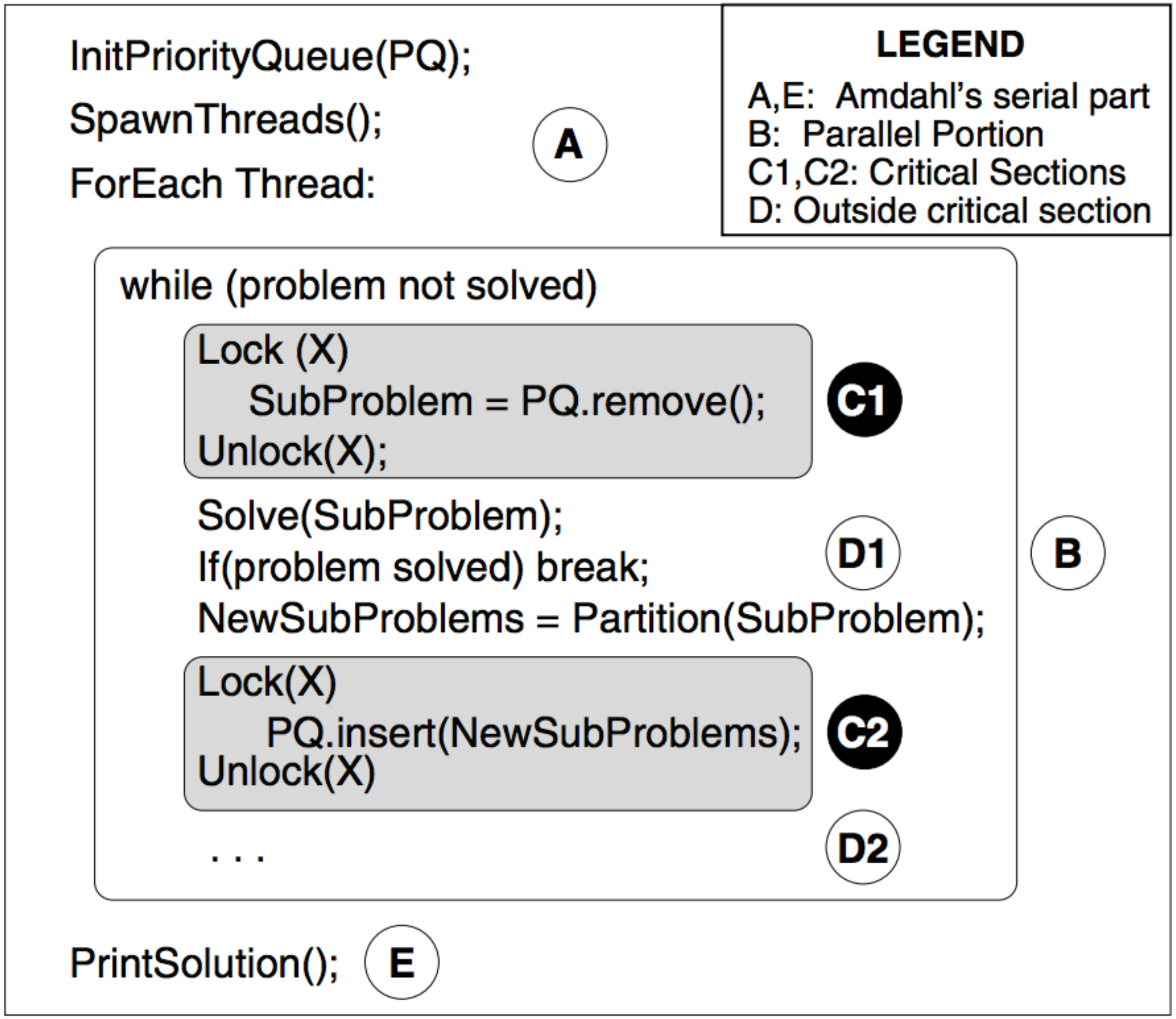
- Parallel machines have the sequential bottleneck

- 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
```

- There are other causes as well:
  - Single thread prepares data and spawns parallel tasks (usually sequential)

Another Example of Sequential Bottleneck (I)



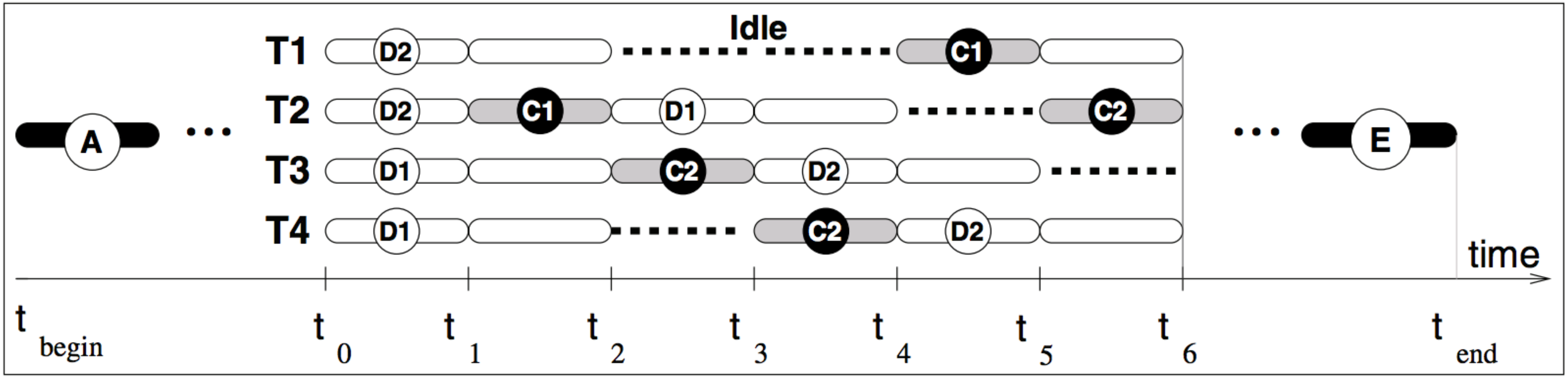
```
InitPriorityQueue(PQ);
SpawnThreads();
ForEach Thread:

    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

Suleman+, "Accelerating Critical Section Execution with Asymmetric Multi-Core Architectures," ASPLOS 2009.

Another Example of Sequential Bottleneck (II)



Suleman+, "Accelerating Critical Section Execution with Asymmetric Multi-Core Architectures," ASPLOS 2009.

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

Bottlenecks in Parallel Portion: Another View

- Threads in a multi-threaded application can be inter-dependent
  - As opposed to threads from different applications

- Such threads can synchronize with each other
  - Locks, barriers, pipeline stages, condition variables, semaphores, …

- Some threads can be on the critical path of execution due to synchronization; some threads are not

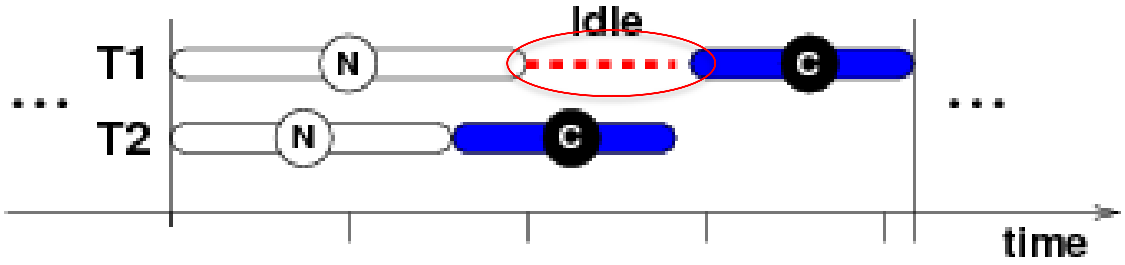
- Within a thread, some "code segments" may be on the critical path of execution; some are not

Remember: Critical Sections

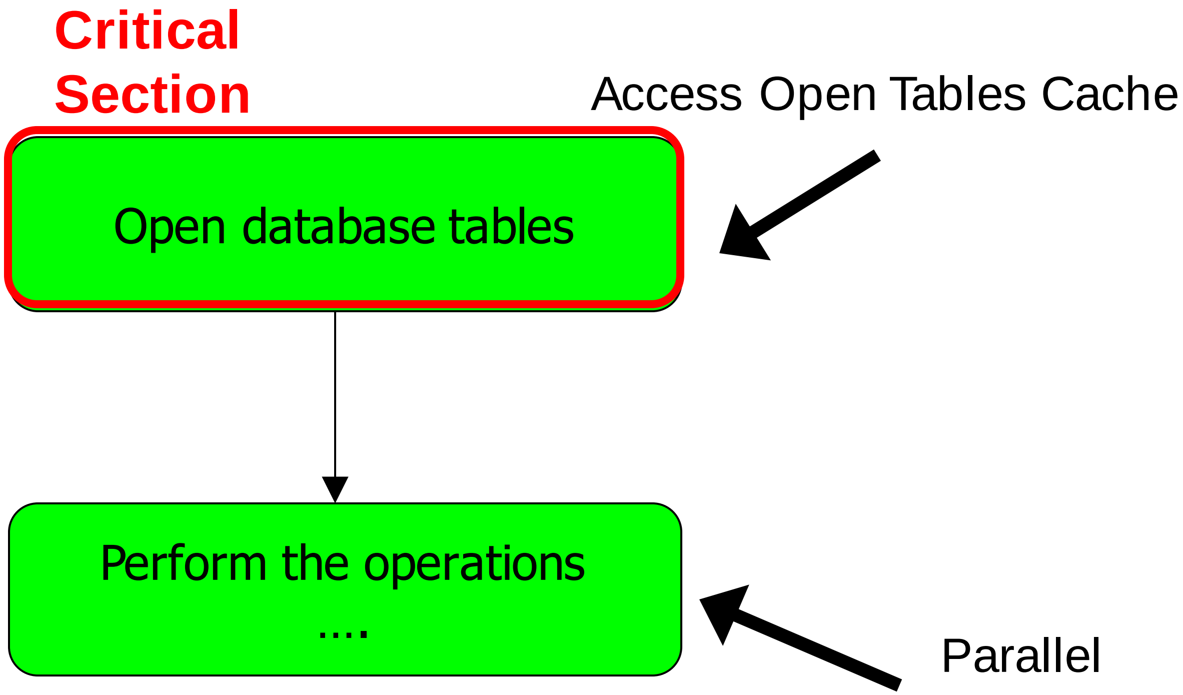
- Enforce mutually exclusive access to shared data
- Only one thread can be executing it at a time
- Contended critical sections make threads wait → threads causing serialization can be on the critical path

Each thread:

```
loop {
   Compute                      N
   lock(A)
      Update shared data
   unlock(A)                    C
}
```



Critical Section Example from MySQL



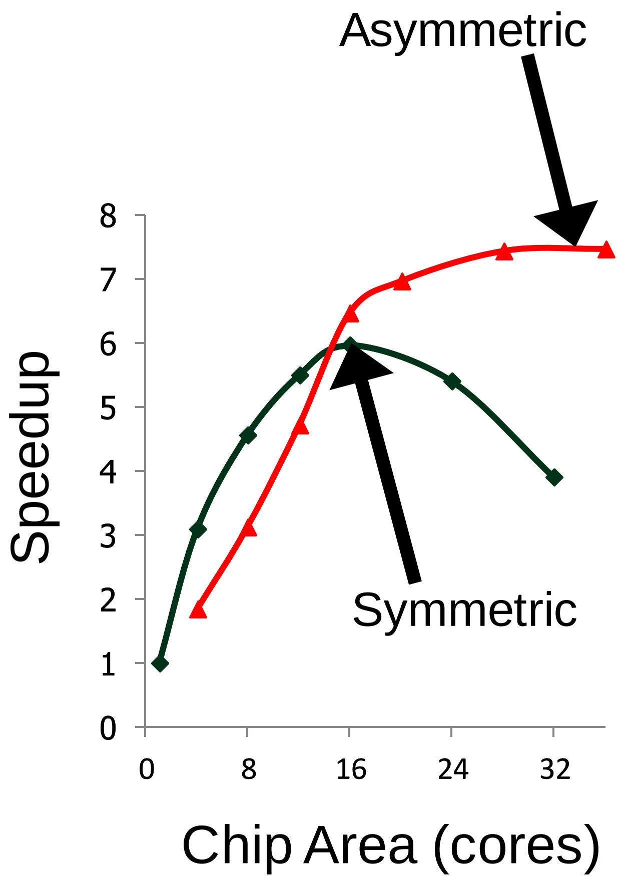
Critical Section

Access Open Tables Cache

Open database tables

Perform the operations ....

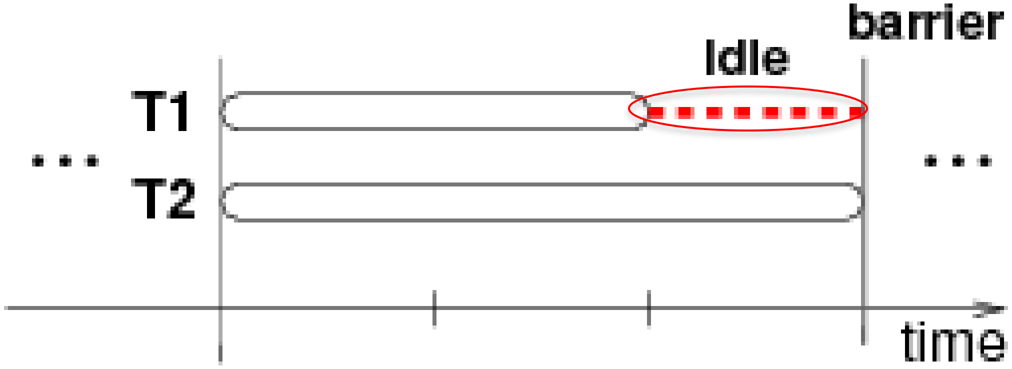
Parallel



Remember: Barriers

- Synchronization point
- Threads have to wait until all threads reach the barrier
- Last thread arriving to the barrier is on the critical path

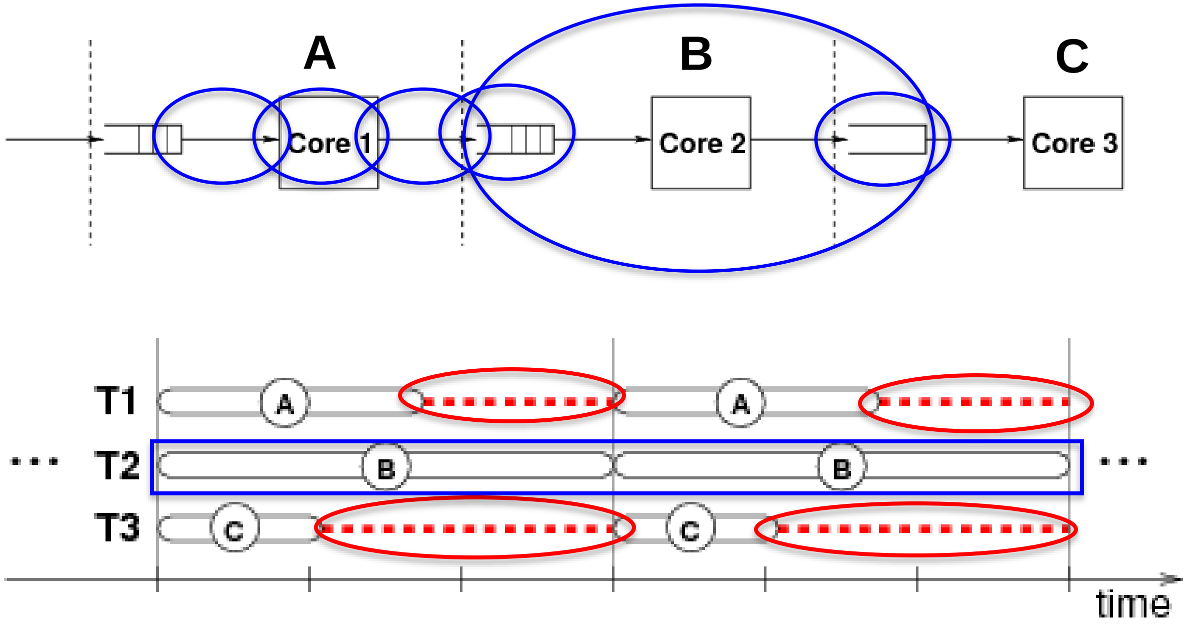
```
Each thread:
    loop1 {
        Compute
    }
    barrier
    loop2 {
        Compute
    }
```



Remember: Stages of Pipelined Programs

- Loop iterations are statically divided into code segments called *stages*
- Threads execute stages on different cores
- Thread executing the slowest stage is on the critical path

```
loop {
  Compute1      A

  Compute2      B

  Compute3      C
}
```



Difficulty in Parallel Programming

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

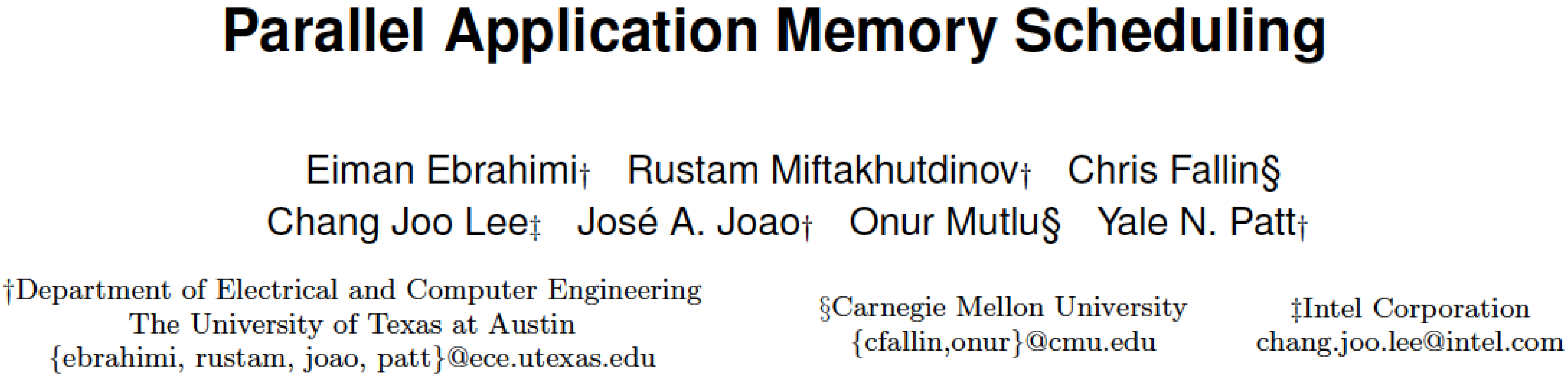
- Difficulty is in
  - Getting parallel programs to work correctly
  - Optimizing performance in the presence of bottlenecks

- 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

Some Readings on Bottlenecks & Bottleneck Acceleration

Parallel Application Memory Scheduling

- Eiman Ebrahimi, Rustam Miftakhutdinov, Chris Fallin, Chang Joo Lee, Onur Mutlu, and Yale N. Patt, **"Parallel Application Memory Scheduling"** *Proceedings of the 44th International Symposium on Microarchitecture* (***MICRO***), Porto Alegre, Brazil, December 2011. Slides (pptx)



Parallel Application Memory Scheduling

Eiman Ebrahimi† Rustam Miftakhutdinov† Chris Fallin§
Chang Joo Lee‡ José A. Joao† Onur Mutlu§ Yale N. Patt†

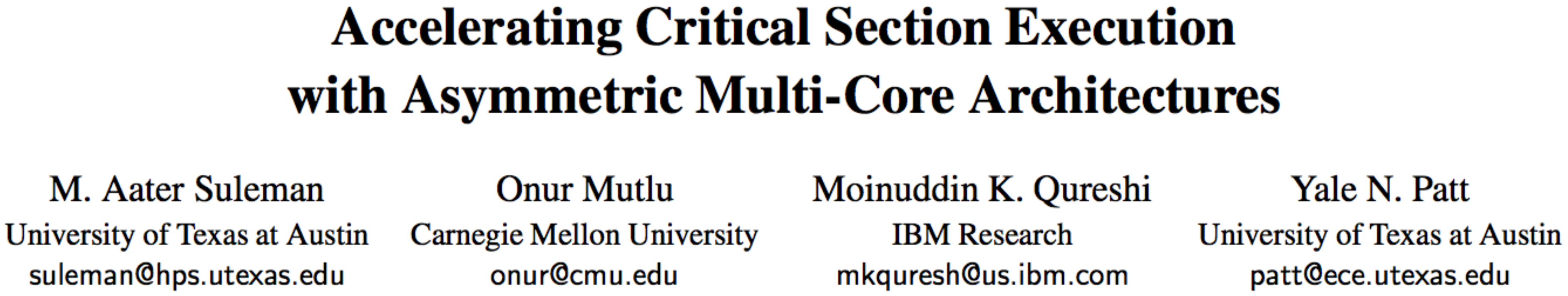
†Department of Electrical and Computer Engineering
The University of Texas at Austin
{ebrahimi, rustam, joao, patt}@ece.utexas.edu

§Carnegie Mellon University
{cfallin,onur}@cmu.edu

‡Intel Corporation
chang.joo.lee@intel.com

Accelerated Critical Sections

- M. Aater Suleman, Onur Mutlu, Moinuddin K. Qureshi, and Yale N. Patt, **"Accelerating Critical Section Execution with Asymmetric Multi-Core Architectures"**
Proceedings of the 14th International Conference on Architectural Support for Programming Languages and Operating Systems (***ASPLOS***), pages 253-264, Washington, DC, March 2009. Slides (ppt)
One of the 13 computer architecture papers of 2009 selected as Top Picks by IEEE Micro.



Accelerating Critical Section Execution with Asymmetric Multi-Core Architectures

M. Aater Suleman
University of Texas at Austin
suleman@hps.utexas.edu

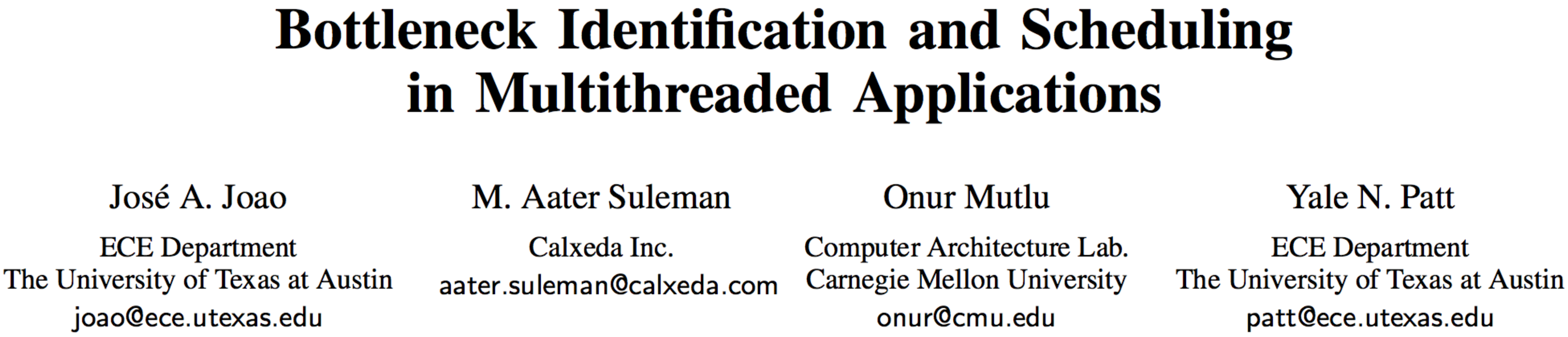
Onur Mutlu
Carnegie Mellon University
onur@cmu.edu

Moinuddin K. Qureshi
IBM Research
mkquresh@us.ibm.com

Yale N. Patt
University of Texas at Austin
patt@ece.utexas.edu

Bottleneck Identification & Scheduling

- Jose A. Joao, M. Aater Suleman, Onur Mutlu, and Yale N. Patt, **"Bottleneck Identification and Scheduling in Multithreaded Applications"** *Proceedings of the 17th International Conference on Architectural Support for Programming Languages and Operating Systems* (***ASPLOS***), London, UK, March 2012. Slides (ppt) (pdf)



Bottleneck Identification and Scheduling in Multithreaded Applications

José A. Joao
ECE Department
The University of Texas at Austin
joao@ece.utexas.edu

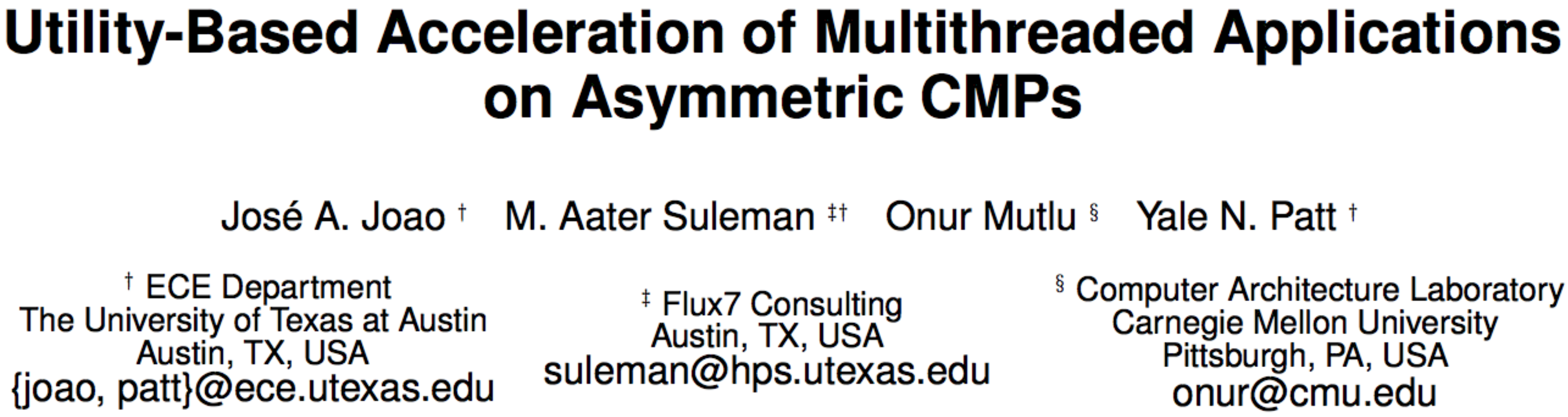
M. Aater Suleman
Calxeda Inc.
aater.suleman@calxeda.com

Onur Mutlu
Computer Architecture Lab.
Carnegie Mellon University
onur@cmu.edu

Yale N. Patt
ECE Department
The University of Texas at Austin
patt@ece.utexas.edu

Utility-Based Acceleration

- Jose A. Joao, M. Aater Suleman, Onur Mutlu, and Yale N. Patt, **"Utility-Based Acceleration of Multithreaded Applications on Asymmetric CMPs"** *Proceedings of the 40th International Symposium on Computer Architecture* (***ISCA***), Tel-Aviv, Israel, June 2013. Slides (ppt) Slides (pdf)



Utility-Based Acceleration of Multithreaded Applications on Asymmetric CMPs

José A. Joao [†] M. Aater Suleman [‡†] Onur Mutlu [§] Yale N. Patt [†]

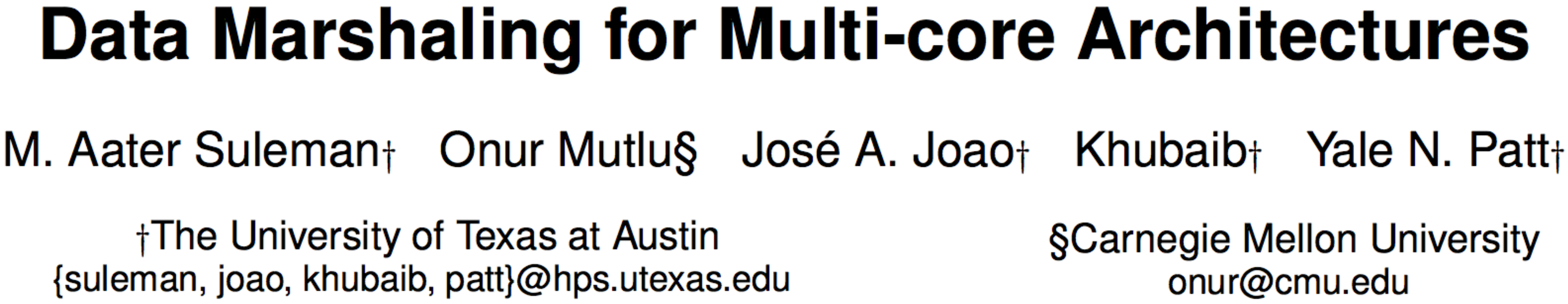
[†] ECE Department
The University of Texas at Austin
Austin, TX, USA
{joao, patt}@ece.utexas.edu

[‡] Flux7 Consulting
Austin, TX, USA
suleman@hps.utexas.edu

[§] Computer Architecture Laboratory
Carnegie Mellon University
Pittsburgh, PA, USA
onur@cmu.edu

Data Marshaling

- M. Aater Suleman, Onur Mutlu, Jose A. Joao, Khubaib, and Yale N. Patt, **"Data Marshaling for Multi-core Architectures"** *Proceedings of the 37th International Symposium on Computer Architecture (**ISCA**)*, pages 441-450, Saint-Malo, France, June 2010. Slides (ppt)
  One of the 11 computer architecture papers of 2010 selected as Top Picks by IEEE Micro.



Data Marshaling for Multi-core Architectures

M. Aater Suleman[†] Onur Mutlu[§] José A. Joao[†] Khubaib[†] Yale N. Patt[†]

[†]The University of Texas at Austin
{suleman, joao, khubaib, patt}@hps.utexas.edu

[§]Carnegie Mellon University
onur@cmu.edu

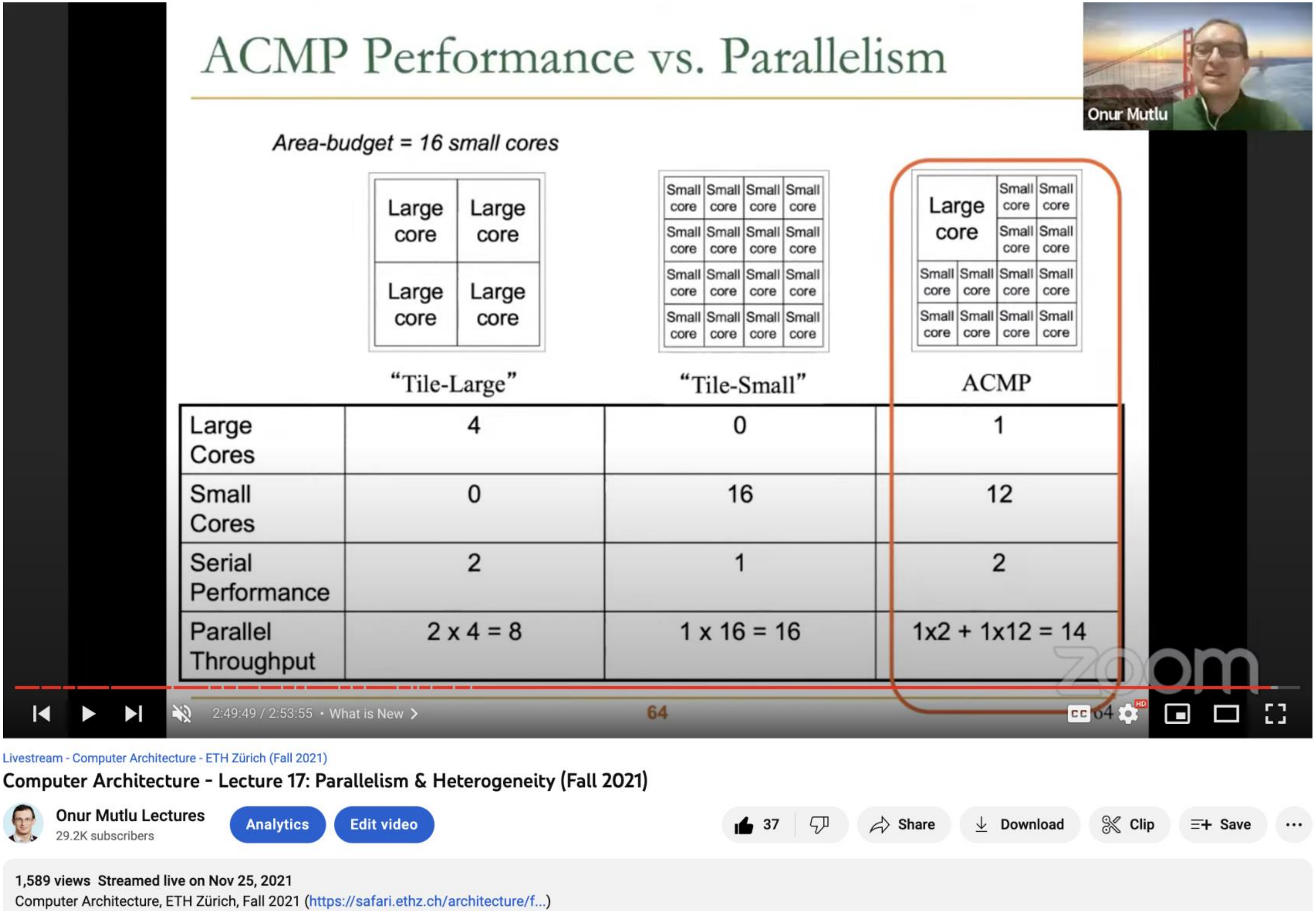
Lectures on Bottleneck Acceleration

- Lecture 18: Parallelism and Heterogeneity
  - Comp Arch, ETH Zurich, Fall 2023
  - https://www.youtube.com/live/r1GmCoVQ_hA?si=5t0kRN2l5sG8YhSa

- Lecture 17a: Parallelism and Heterogeneity
  - Comp Arch, ETH Zurich, Fall 2021
  - https://www.youtube.com/watch?v=GLzG_rEDn9A&list=PL5Q2soXY2Zi-Mnk1PxjEIG32HAGILkTOF&index=18

- Lecture 18a: Bottleneck Acceleration
  - Comp Arch, ETH Zurich, Fall 2021
  - https://www.youtube.com/watch?v=P8l3SMAbyYw&list=PL5Q2soXY2Zi-Mnk1PxjEIG32HAGILkTOF&index=19

Lecture on Parallelism & Heterogeneity



ACMP Performance vs. Parallelism

Area-budget = 16 small cores

	"Tile-Large"	"Tile-Small"	ACMP
Large Cores	4	0	1
Small Cores	0	16	12
Serial Performance	2	1	2
Parallel Throughput	2 x 4 = 8	1 x 16 = 16	1x2 + 1x12 = 14

Livestream - Computer Architecture - ETH Zürich (Fall 2021)

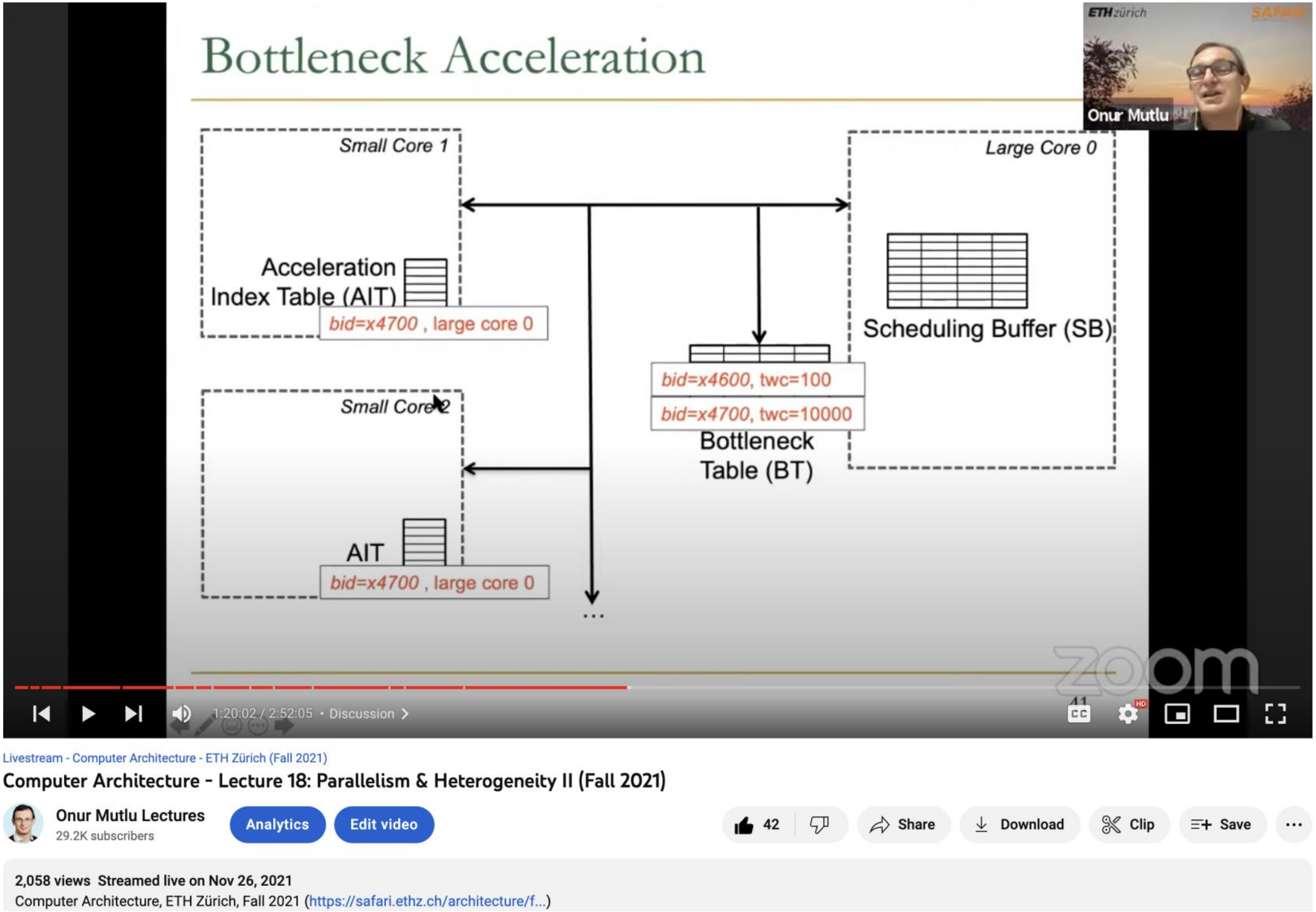
Computer Architecture - Lecture 17: Parallelism & Heterogeneity (Fall 2021)

Onur Mutlu Lectures
29.2K subscribers

1,589 views Streamed live on Nov 25, 2021
Computer Architecture, ETH Zürich, Fall 2021 (https://safari.ethz.ch/architecture/f...)

https://www.youtube.com/watch?v=GLzG_rEDn9A&list=PL5Q2soXY2Zi-Mnk1PxjEIG32HAGILkTOF&index=19

Lecture on Bottleneck Acceleration



Livestream - Computer Architecture - ETH Zürich (Fall 2021)

Computer Architecture - Lecture 18: Parallelism & Heterogeneity II (Fall 2021)

Onur Mutlu Lectures
29.2K subscribers

2,058 views Streamed live on Nov 26, 2021
Computer Architecture, ETH Zürich, Fall 2021 (https://safari.ethz.ch/architecture/f...)

https://www.youtube.com/watch?v=P8l3SMAbyYw&list=PL5Q2soXY2Zi-Mnk1PxjEIG32HAGILkTOF&index=19

Computer Architecture

Lecture 19a: Multiprocessors

Dr. Mohammad Sadrosadati

Prof. Onur Mutlu

ETH Zürich

Fall 2023

30 November 2023

An Example Parallel Problem: Task Assignment to Processors

Static versus Dynamic Scheduling

- Static: Done at compile time or parallel task creation time
  - Schedule does not change based on runtime information

- Dynamic: Done at run time (e.g., after tasks are created)
  - Schedule changes based on runtime information

- Example: Instruction scheduling
  - Why would you like to do dynamic scheduling?
  - What pieces of information are not available to the static scheduler?

Parallel Task Assignment: Tradeoffs

- Problem: N tasks, P processors, N>P. Do we assign tasks to processors statically (fixed) or dynamically (adaptive)?

- Static assignment

  + Simpler: No movement of tasks.

  - Inefficient: Underutilizes resources when load is not balanced

  When can load not be balanced?

- Dynamic assignment

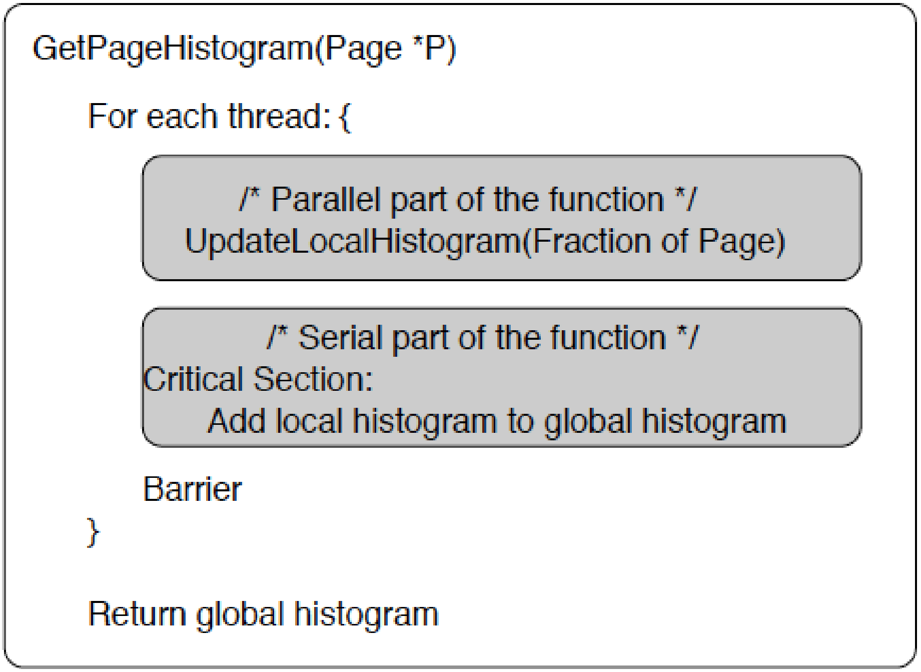
  + Efficient: Better utilizes processors when load is not balanced

  - More complex: Need to move tasks to balance processor load

  - Higher overhead: Task movement takes time, can disrupt locality

Parallel Task Assignment: Example

- Compute histogram of a large set of values
- Parallelization:
  - Divide the values across T tasks
  - Each task computes a local histogram for its value set
  - Local histograms merged with global histograms in the end

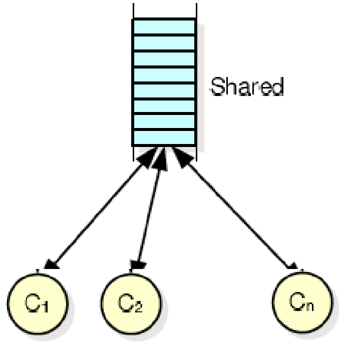


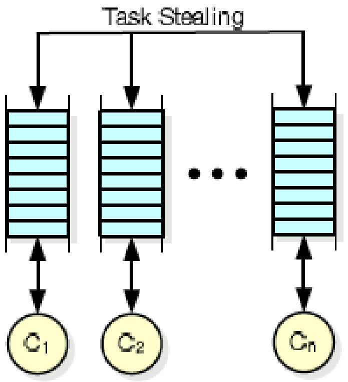
```
GetPageHistogram(Page *P)
    For each thread: {
            /* Parallel part of the function */
            UpdateLocalHistogram(Fraction of Page)

            /* Serial part of the function */
            Critical Section:
                Add local histogram to global histogram

        Barrier
    }

    Return global histogram
```

Parallel Task Assignment: Example (II)

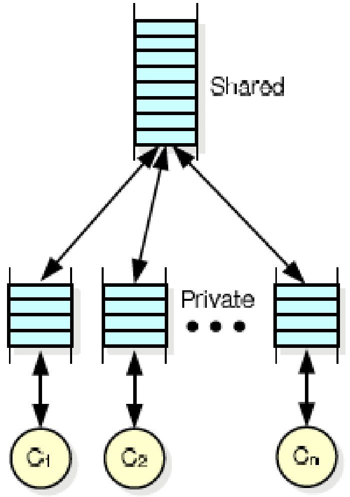
- How to schedule tasks updating local histograms?
  - Static: Assign equal number of tasks to each processor
  - Dynamic: Assign tasks to a processor that is available
  - When does static work as well as dynamic?

- Implementation of Dynamic Assignment with Task Queues





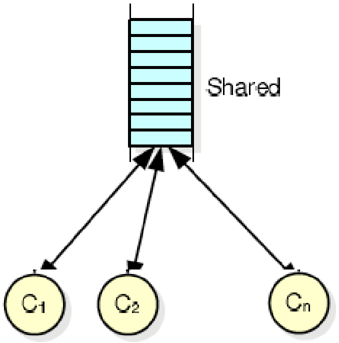
(a) Distributed Task Stealing

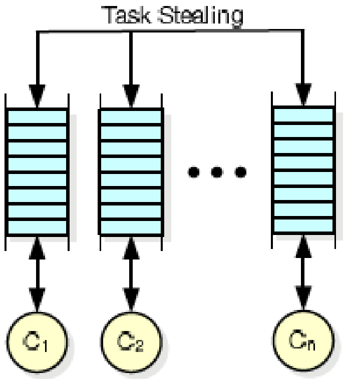


(b) Hierarchical Task Queuing

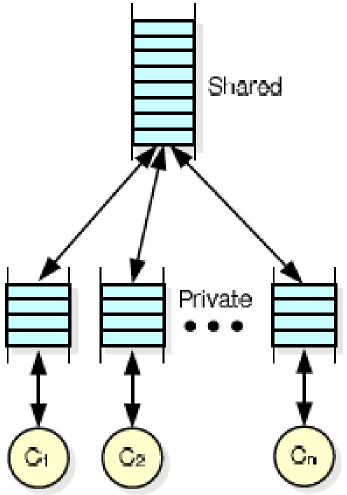
Software Task Queues

- What are the advantages and disadvantages of each?
  - Centralized
  - Distributed
  - Hierarchical





(a) Distributed Task Stealing



(b) Hierarchical Task Queuing

Task Stealing

- Idea: When a processor' s task queue is empty it steals a task from another processor' s task queue
  - Whom to steal from? (Randomized stealing works well)
  - How many tasks to steal?

+ Dynamic balancing of computation load

- Additional communication/synchronization overhead between processors

- Need to stop stealing if no tasks to steal

Parallel Task Assignment: Tradeoffs

- Who does the assignment? Hardware versus software?

- Software

  + Better scope

  - More time overhead

  - Slow to adapt to dynamic events (e.g., a processor becoming idle)

- Hardware

  + Low time overhead

  + Can adjust to dynamic events faster

  - Requires hardware changes (area and possibly energy overhead)

How Can the Hardware Help?

- Managing task queues in software has overhead
  - Especially high when task sizes are small

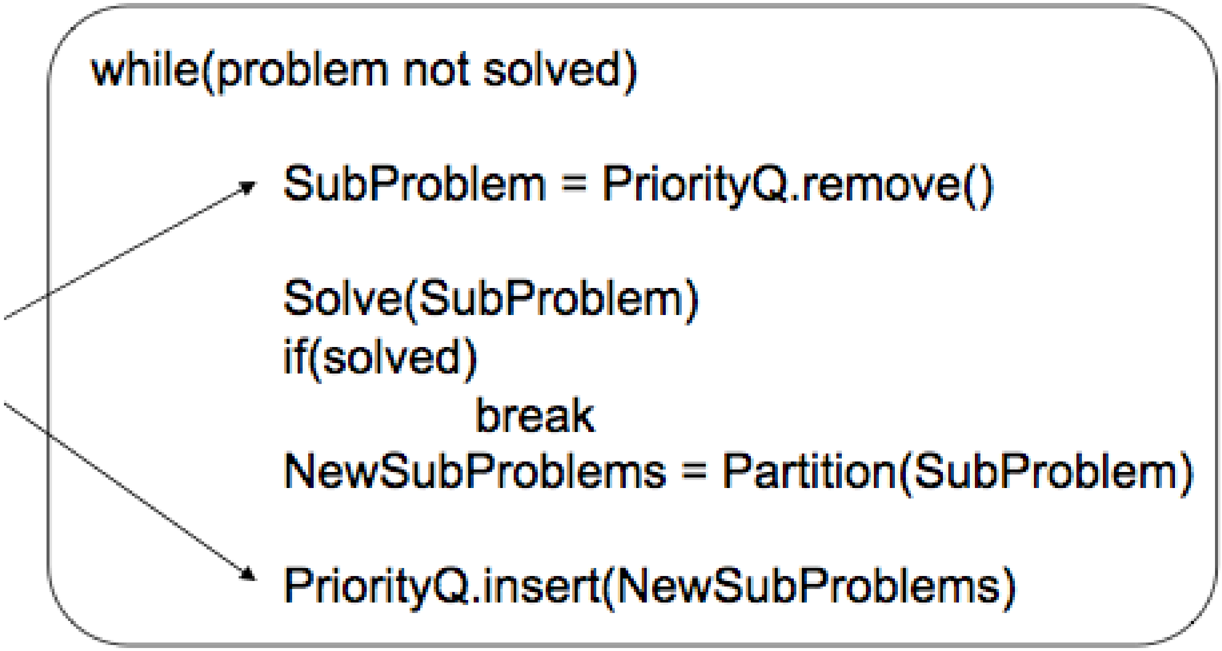
- An idea: Hardware Task Queues
  - Each processor has a dedicated task queue
  - Software fills the task queues (on demand)
  - Hardware manages movement of tasks from queue to queue
  - There can be a global task queue as well → hierarchical tasking in hardware

  - Kumar et al., "Carbon: Architectural Support for Fine-Grained Parallelism on Chip Multiprocessors," ISCA 2007.
    - Optional reading

Dynamic Task Generation

- Does static task assignment work in this case?

- Problem: Searching the exit of a maze



```
while(problem not solved)

          SubProblem = PriorityQ.remove()

          Solve(SubProblem)
          if(solved)
                    break
          NewSubProblems = Partition(SubProblem)

          PriorityQ.insert(NewSubProblems)
```

Programming Model vs. Hardware Execution Model

Programming Models vs. Architectures

- Five major models
  - (Sequential)
  - Shared memory
  - Message passing
  - Data parallel (SIMD)
  - Dataflow
  - Systolic

- Hybrid models?

Shared Memory vs. Message Passing

- Are these programming models or execution models supported by the hardware architecture?

- Does a multiprocessor that is programmed by “shared memory programming model” have to support a shared address space processors?

- Does a multiprocessor that is programmed by “message passing programming model” have to have no shared address space between processors?

Programming Models: Message Passing vs. Shared Memory

- Difference: how communication is achieved between tasks
- Message passing programming model
  - Explicit communication via messages
  - Loose coupling of program components
  - Analogy: telephone call or letter, no shared location accessible to all
- Shared memory programming model
  - Implicit communication via memory operations (load/store)
  - Tight coupling of program components
  - Analogy: bulletin board, post information at a shared space

- Suitability of the programming model depends on the problem to be solved. Issues affected by the model include:
  - Overhead, scalability, ease of programming, bugs, match to underlying hardware, …

Message Passing vs. Shared Memory Hardware

- Difference: how task communication is supported in hardware
- Shared memory hardware (or machine model)
  - All processors see a global shared address space
    - Ability to access all memory from each processor
  - A write to a location is visible to the reads of other processors
- Message passing hardware (machine model)
  - No global shared address space
  - Send and receive variants are the only method of communication between processors (much like networks of workstations today, i.e. clusters)
- Suitability of the hardware depends on the problem to be solved as well as the programming model.

Programming Model vs. Hardware

- Most of parallel computing history, there was no separation between programming model and hardware
  - Message passing: Caltech Cosmic Cube, Intel Hypercube, Intel Paragon
  - Shared memory: CMU C.mmp, Sequent Balance, SGI Origin.
  - SIMD: ILLIAC IV, CM-1

- However, any hardware can really support any programming model
- Why?
  - Application → compiler/library → OS services → hardware