

# Computer Architecture

## Lecture 17a: Multiprocessors

Prof. Onur Mutlu

ETH Zürich

Fall 2022

24 November 2022

# Prefetching Wrap Up

# Runahead as an Execution-Based Prefetcher

# Runahead as an Execution-based Prefetcher

---

- Idea of an Execution-Based Prefetcher: Pre-execute a piece of the (pruned) program solely for prefetching data
- Idea of Runahead: Pre-execute the main program solely for prefetching data
- Advantages and disadvantages of runahead vs. other execution-based prefetchers?
- Can you make runahead even better by pruning the program portion executed in runahead mode?
  - Yes → Continuous Runahead is an example of this

# Taking Advantage of Pure Speculation

---

- Runahead mode is purely speculative
- The goal is to find and generate cache misses that would otherwise stall execution later on
- How do we achieve this goal most efficiently and with the highest benefit?
- Idea: Find and execute only those instructions that will lead to cache misses (that cannot already be captured by the instruction window)
- How? → Continuous Runahead is an example of this

# Continuous Runahead: Much More Efficient

---

- Milad Hashemi, Onur Mutlu, and Yale N. Patt,  
**"Continuous Runahead: Transparent Hardware Acceleration for Memory Intensive Workloads"**  
*Proceedings of the 49th International Symposium on Microarchitecture (MICRO), Taipei, Taiwan, October 2016.*  
*[Slides (pptx) (pdf)] [Lightning Session Slides (pdf)] [Poster (pptx) (pdf)]*  
***Best paper session.***

## Continuous Runahead: Transparent Hardware Acceleration for Memory Intensive Workloads

Milad Hashemi\*, Onur Mutlu<sup>§</sup>, Yale N. Patt\*

\**The University of Texas at Austin*    <sup>§</sup>*ETH Zürich*

# Execution-based Prefetchers: Pros and Cons

---

- + Can prefetch pretty much **any access pattern**
- + **Can be very low cost** (e.g., runahead execution)
  - + Especially if it uses the same hardware context
  - + Why? The processor is equipped to execute the program anyway
- + **Can be bandwidth-efficient** (e.g., runahead execution)
- Depend on **branch prediction and possibly value prediction accuracy**
  - Mispredicted branches dependent on missing data throw the thread off the correct execution path
- Can be **wasteful**
  - speculatively execute many instructions
  - can occupy a separate thread context
- Complexity in deciding when and what to pre-execute

# More on Runahead Execution

---

- Onur Mutlu, Jared Stark, Chris Wilkerson, and Yale N. Patt,  
**"Runahead Execution: An Alternative to Very Large Instruction Windows for Out-of-order Processors"**

*Proceedings of the 9th International Symposium on High-Performance Computer Architecture (HPCA)*, pages 129-140, Anaheim, CA, February 2003. [Slides \(pdf\)](#)

***One of the 15 computer arch. papers of 2003 selected as Top Picks by IEEE Micro. HPCA Test of Time Award (awarded in 2021).***

[\[Lecture Slides \(pptx\) \(pdf\)\]](#)

[\[Lecture Video \(1 hr 54 mins\)\]](#)

[\[Retrospective HPCA Test of Time Award Talk Slides \(pptx\) \(pdf\)\]](#)

[\[Retrospective HPCA Test of Time Award Talk Video \(14 minutes\)\]](#)

## **Runahead Execution: An Alternative to Very Large Instruction Windows for Out-of-order Processors**

Onur Mutlu §    Jared Stark †    Chris Wilkerson ‡    Yale N. Patt §

§ECE Department

The University of Texas at Austin

{onur,patt}@ece.utexas.edu

†Microprocessor Research

Intel Labs

jared.w.stark@intel.com

‡Desktop Platforms Group

Intel Corporation

chris.wilkerson@intel.com



# More on Efficient Runahead Execution

---

- Onur Mutlu, Hyesoon Kim, and Yale N. Patt,  
**"Techniques for Efficient Processing in Runahead Execution Engines"**  
*Proceedings of the 32nd International Symposium on Computer Architecture (ISCA)*, pages 370-381, Madison, WI, June 2005. Slides (ppt) Slides (pdf)  
***One of the 13 computer architecture papers of 2005 selected as Top Picks by IEEE Micro.***

## Techniques for Efficient Processing in Runahead Execution Engines

Onur Mutlu   Hyesoon Kim   Yale N. Patt

Department of Electrical and Computer Engineering  
University of Texas at Austin  
{onur,hyesoon,patt}@ece.utexas.edu

# More Effective Runahead Execution

---

- Onur Mutlu, Hyesoon Kim, and Yale N. Patt,  
**"Address-Value Delta (AVD) Prediction: Increasing the Effectiveness of Runahead Execution by Exploiting Regular Memory Allocation Patterns"**  
*Proceedings of the 38th International Symposium on Microarchitecture (MICRO)*,  
pages 233-244, Barcelona, Spain, November 2005. [Slides \(ppt\)](#) [Slides \(pdf\)](#)  
***One of the five papers nominated for the Best Paper Award by the Program Committee.***

## **Address-Value Delta (AVD) Prediction: Increasing the Effectiveness of Runahead Execution by Exploiting Regular Memory Allocation Patterns**

Onur Mutlu   Hyesoon Kim   Yale N. Patt

Department of Electrical and Computer Engineering  
University of Texas at Austin  
{onur,hyesoon,patt}@ece.utexas.edu

# More on Runahead Execution

---

- Lecture video from Fall 2020, Computer Architecture:
  - [https://www.youtube.com/watch?v=zPewo6IaJ\\_8](https://www.youtube.com/watch?v=zPewo6IaJ_8)
- Lecture video from Fall 2017, Computer Architecture:
  - <https://www.youtube.com/watch?v=Kj3relihGF4>
- Onur Mutlu,  
**"Efficient Runahead Execution Processors"**  
Ph.D. Dissertation, HPS Technical Report, TR-HPS-2006-007, July 2006. [Slides \(ppt\)](#)  
***Nominated for the ACM Doctoral Dissertation Award by the University of Texas at Austin.***

# More on Continuous Runahead

---

- Milad Hashemi, Onur Mutlu, and Yale N. Patt,  
**"Continuous Runahead: Transparent Hardware Acceleration for Memory Intensive Workloads"**  
*Proceedings of the 49th International Symposium on Microarchitecture (MICRO), Taipei, Taiwan, October 2016.*  
[\[Slides \(pptx\) \(pdf\)\]](#) [\[Lightning Session Slides \(pdf\)\]](#) [\[Poster \(pptx\) \(pdf\)\]](#)  
***Best paper session.***

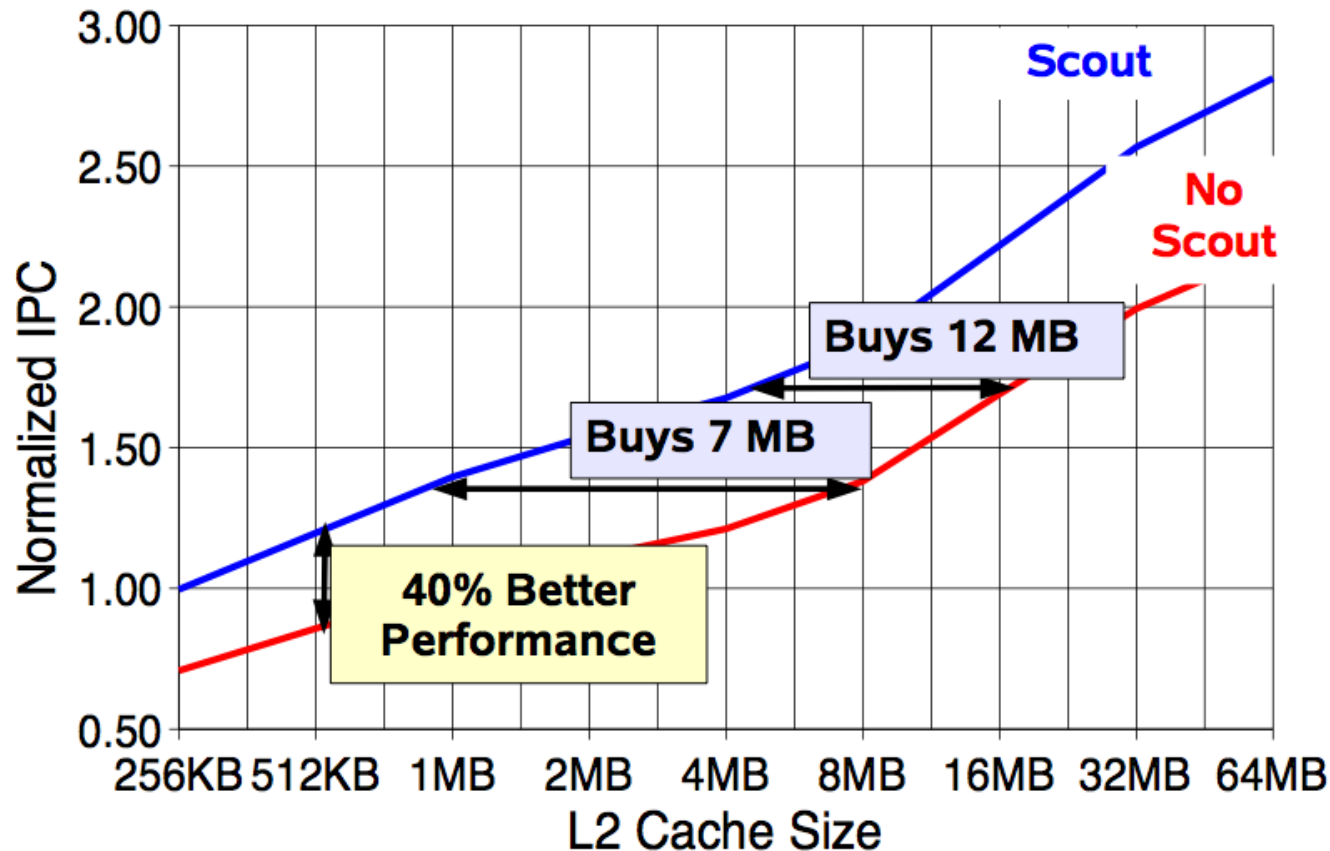
## Continuous Runahead: Transparent Hardware Acceleration for Memory Intensive Workloads

Milad Hashemi\*, Onur Mutlu<sup>§</sup>, Yale N. Patt\*

\**The University of Texas at Austin*    <sup>§</sup>*ETH Zürich*

# Effect of Runahead in Sun ROCK

- Shailender Chaudhry talk, Aug 2008.



Effective prefetching can both improve performance and reduce hardware cost

# HIGH-PERFORMANCE THROUGHPUT COMPUTING

---

THROUGHPUT COMPUTING, ACHIEVED THROUGH MULTITHREADING AND MULTICORE TECHNOLOGY, CAN LEAD TO PERFORMANCE IMPROVEMENTS THAT ARE 10 TO 30× THOSE OF CONVENTIONAL PROCESSORS AND SYSTEMS. HOWEVER, SUCH SYSTEMS SHOULD ALSO OFFER GOOD SINGLE-THREAD PERFORMANCE. HERE, THE AUTHORS SHOW THAT HARDWARE SCOUTING INCREASES THE PERFORMANCE OF AN ALREADY ROBUST CORE BY UP TO 40 PERCENT FOR COMMERCIAL BENCHMARKS.

# More on Runahead in Sun ROCK

---

## **Simultaneous Speculative Threading: A Novel Pipeline Architecture Implemented in Sun's ROCK Processor**

Shailender Chaudhry, Robert Cypher, Magnus Ekman, Martin Karlsson,  
Anders Landin, Sherman Yip, Håkan Zeffer, and Marc Tremblay  
Sun Microsystems, Inc.  
4180 Network Circle, Mailstop SCA18-211  
Santa Clara, CA 95054, USA  
{shailender.chaudhry, robert.cypher, magnus.ekman, martin.karlsson,  
anders.landin, sherman.yip, haakan.zeffer, marc.tremblay}@sun.com

# Runahead Execution in IBM POWER6

---

## **Runahead Execution vs. Conventional Data Prefetching in the IBM POWER6 Microprocessor**

Harold W. Cain

Priya Nagpurkar

IBM T.J. Watson Research Center  
Yorktown Heights, NY  
{tcain, pnagpurkar}@us.ibm.com

Cain+, “[Runahead Execution vs. Conventional Data Prefetching  
in the IBM POWER6 Microprocessor](#),” ISPASS 2010.



## DENVER: NVIDIA'S FIRST 64-BIT ARM PROCESSOR

NVIDIA'S FIRST 64-BIT ARM PROCESSOR, CODE-NAMED DENVER, LEVERAGES A HOST OF NEW TECHNOLOGIES, SUCH AS DYNAMIC CODE OPTIMIZATION, TO ENABLE HIGH-PERFORMANCE MOBILE COMPUTING. IMPLEMENTED IN A 28-NM PROCESS, THE DENVER CPU CAN ATTAIN CLOCK SPEEDS OF UP TO 2.5 GHZ. THIS ARTICLE OUTLINES THE DENVER ARCHITECTURE, DESCRIBES ITS TECHNOLOGICAL INNOVATIONS, AND PROVIDES RELEVANT COMPARISONS AGAINST COMPETING MOBILE PROCESSORS.

Boggs+, "[Denver: NVIDIA's First 64-Bit ARM Processor](#)," IEEE Micro 2015.

# Runahead Execution in NVIDIA Denver

Reducing the effects of long cache-miss penalties has been a major focus of the micro-architecture, using techniques like prefetching and run-ahead. An aggressive hardware prefetcher implementation detects L2 cache requests and tracks up to 32 streams, each with complex stride patterns.

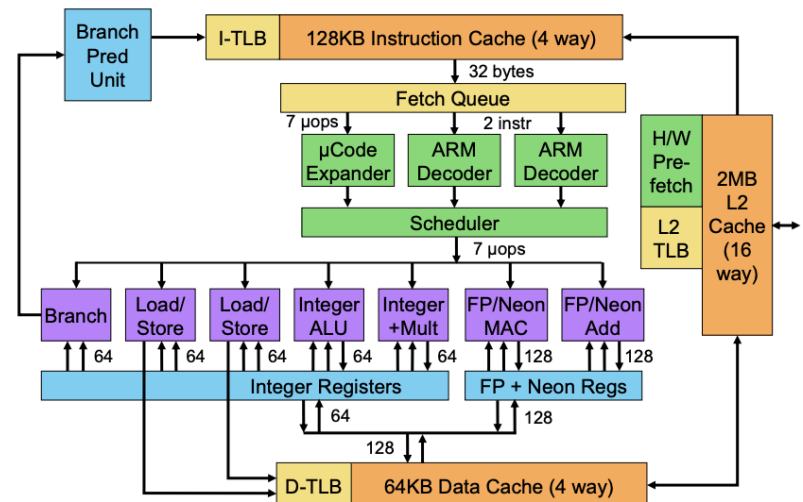
Run-ahead uses the idle time that a CPU spends waiting on a long latency operation to discover cache and DTLB misses further down the instruction stream and generates prefetch requests for these misses.<sup>1</sup> These prefetch requests warm up the data cache and DTLB well before the actual execution of

the instructions that require the data. Run-ahead complements the hardware prefetcher because it's better at prefetching nonstrided streams, and it trains the hardware prefetcher faster than normal execution to yield a combined benefit of 13 percent on SPECint2000 and up to 60 percent on SPECfp2000.

Boggs+, "Denver: NVIDIA's First 64-Bit ARM Processor,"  
IEEE Micro 2015.

Gwennap, "NVIDIA's First CPU is a Winner," MPR 2014.

The core includes a hardware prefetch unit that Boggs describes as “aggressive” in preloading the data cache but less aggressive in preloading the instruction cache. It also implements a “run-ahead” feature that continues to execute microcode speculatively after a data-cache miss; this execution can trigger additional cache misses that resolve in the shadow of the first miss. Once the data from the original miss returns, the results of this speculative execution are discarded and execution restarts with the bundle containing the original miss, but run-ahead can preload subsequent data into the cache, thus avoiding a string of time-wasting cache misses. These and other features help Denver outscore Cortex-A15 by more than 2.6x on a memory-read test even when both use the same SoC framework (Tegra K1).



**Figure 3. Denver CPU microarchitecture.** This design combines a fairly

# Looking to the Past

# At the Time... Early 2000s...

---

- Large focus on increasing the size of the window...
  - And, designing bigger, more complicated machines
- Runahead was a different way of thinking
  - Keep the OoO core simple and small
  - At the expense of some benefits (e.g., non-memory-related)
  - Use aggressive “automatic speculative execution” solely for prefetching
  - Synergistic with prefetching and branch prediction methods
- A lot of interesting and innovative ideas ensued...

# Important Precedent [Dundas & Mudge, ICS 1997]

---

## Improving Data Cache Performance by Pre-executing Instructions Under a Cache Miss

James Dundas and Trevor Mudge

Department of Electrical Engineering and Computer Science

The University of Michigan

Ann Arbor, Michigan 48109-2122

{dundas, tnm}@eecs.umich.edu

### Abstract

In this paper we propose and evaluate a technique that improves first level data cache performance by pre-executing future instructions under a data cache miss. We show that these pre-executed instructions can generate highly accurate data prefetches, particularly when the first level cache is small. The technique is referred to as *runahead* processing. The hardware required to implement runahead is modest, because, when a miss occurs, it makes use of an otherwise idle resource, the execution logic. The principal hardware cost is an extra register file. To measure the impact of runahead, we simulated a processor executing five integer Spec95 benchmarks. Our results show that runahead was able to significantly reduce data cache CPI for four of the five benchmarks. We also compared runahead to a simple form of prefetching, sequential prefetching, which would seem to be suitable for scientific benchmarks. We confirm this by enlarging the scope of our experiments to include a scientific benchmark. However, we show that runahead was also able to outperform sequential prefetching on the scientific benchmark. We also conduct studies that demonstrate that runahead can generate many useful prefetches for lines that show little spatial locality with the misses that initiate runahead episodes. Finally, we discuss some further enhancements of our baseline runahead prefetching scheme.

are allocated by the software. This hybrid hardware-software technique was presented in [8]. Their instruction stride table (IST) selectively generates cache miss initiated prefetches for accesses chosen beforehand by the compiler. This resulted in multiprocessor performance for scientific benchmarks comparable in some cases to software prefetching, with an instruction stride table as small as 4 entries. The IST concept was subsequently combined with the prefetch predicates of [2] in [9]. Another hardware prefetching scheme that avoids the need for significant amounts of hardware is the “wrong path” prefetching described in [10]. This actually prefetches instructions from the not-taken path, in the expectation that they will be executed during a later iteration.

Most prefetching techniques, software- or hardware-based, tend to perform poorly on an important class of applications having recursive data structures such as linked-lists. A software technique that overcomes this limitation was presented recently in [11], in which software prefetches were inserted at subroutine call sites that passed pointers as arguments. Another pointer-based approach was described in [12]. This approach uses pointers stored within the data structures to generate software prefetches.

The runahead prefetching approach presented in this paper is a hardware approach, that requires only a modest amount of hardware, because, when a miss occurs, it makes use of an otherwise

# An Inspiration [Glew, ASPLOS-WACI 1998]

## MLP yes! ILP no!

*Memory Level Parallelism, or why I no longer care about Instruction Level Parallelism*

Andrew Glew

Intel Microcomputer Research Labs and University of Wisconsin, Madison

**Problem Description:** It should be well known that processors are outstripping memory performance: specifically that memory latencies are not improving as fast as processor cycle time or IPC or memory bandwidth.

Thought experiment: imagine that a cache miss takes 10000 cycles to execute. For such a processor instruction level parallelism is useless, because most of the time is spent waiting for memory. Branch prediction is also less effective, since most branches can be determined with data already in registers or in the cache; branch prediction only helps for branches which depend on outstanding cache misses.

At the same time, pressures for reduced power consumption mount.

Given such trends, some computer architects in industry (although not Intel EPIC) are talking seriously about retreating from out-of-order superscalar processor architecture, and instead building simpler, faster, dumber, 1-wide in-order processors with high degrees of speculation. Sometimes this is proposed in combination with multiprocessing and multithreading: tolerate long memory latencies by switching to other processes or threads.

I propose something different: build narrow fast machines but use intelligent logic inside the CPU to increase the number of outstanding cache misses that can be generated from a single program.

**Solution:** First, change the mindset: MLP, Memory Level Parallelism, is what matters, not ILP, Instruction Level Parallelism.

By MLP I mean simply the number of outstanding cache misses that can be generated (by a single thread, task, or program) and executed in an overlapped manner. It does not matter what sort of execution engine generates the multiple outstanding cache misses. An out-of-order superscalar ILP CPU may generate multiple outstanding cache misses, but 1-wide processors can be just as effective.

Change the metrics: total execution time remains the overall goal, but instead of reporting IPC as an approximation to this, we must report MLP. Limit studies should be in terms of total number of non-overlapped cache misses on critical path.

Now do the research: Many present-day hot topics in computer architecture help ILP, but do not help MLP. As mentioned above, predicting branch directions for branches that can be determined from data already in the cache or in registers does not help MLP for extremely long latencies. Similarly, prefetching of data cache misses for array processing codes does not help MLP – it just

Instead, investigate microarchitectures that help MLP:

- (0) Trivial case – explicit multithreading, like SMT.
- (1) Slightly less trivial case – implicitly multithread single programs, either by compiler software on an MT machine, or by a hybrid, such as Wisconsin Multiscalar, or entirely in hardware, as in Intel's Dynamic Multi-Threading.
- (2) Build 1-wide processors that are as fast as possible: use circuit tricks, as well as logic tricks such as redundant encoding for numeric computation and memory addressing.
- (3) Allow the hardware dynamic scheduling mechanisms to use sequential algorithms implemented by this narrow, fast, processor, rather than limiting it to parallel algorithms implementable in associative logic.
- (4) Build very large instruction windows allowing speculation tens of thousands of instructions ahead. Avoid circuit speed issues by caching the instruction window. Remove small arbitrary limits on the number of cache misses outstanding allowed.
- (5) Further reduce the cost of very large instruction windows by throwing away anything that can be recomputed based on data in registers or cache.
- (6) Don't stall speculation because the oldest instruction in the machine is a cache miss. Let the front of the machine continue executing branches, forgetting data dependent on cache misses.
- (7) Parallelize linked data structure traversals by building skip lists in hardware – converting sequential data structures into parallel ones. Store these extra skip pointers in main memory.

Call such a processor microarchitecture a "super-non-blocking" microarchitecture.

**Justification:** The processor/memory trend is well known. Theoretically optimal cache studies show only limited headroom. Barring a revolution in memory technology, the Memory Wall is real, and getting closer. Multithreading and multiprocessing have some hope of tolerating memory latency, but only if there are parallel workloads. If single thread performance is still an issue, the only potentially MLP enhancing technologies are what I describe here, or data value prediction – and data value prediction seems to only do well for stuff that fits in the cache.

"Super-non-blocking" processors extends dynamic, out-of-order, execution to maximize MLP, but simplifies it by discarding superscalar ILP as unnecessary.

# Looking to the Future



# A Look into the Future...

---

- Microarchitecture (especially memory) is critically important
  - And, fun...
  - And, impactful...
- Runahead is a great example of harmonious industry-academia collaboration
- Fundamental problems will remain fundamental
  - And will require fundamental (and creative) solutions



# Citation for the Test of Time Award

---

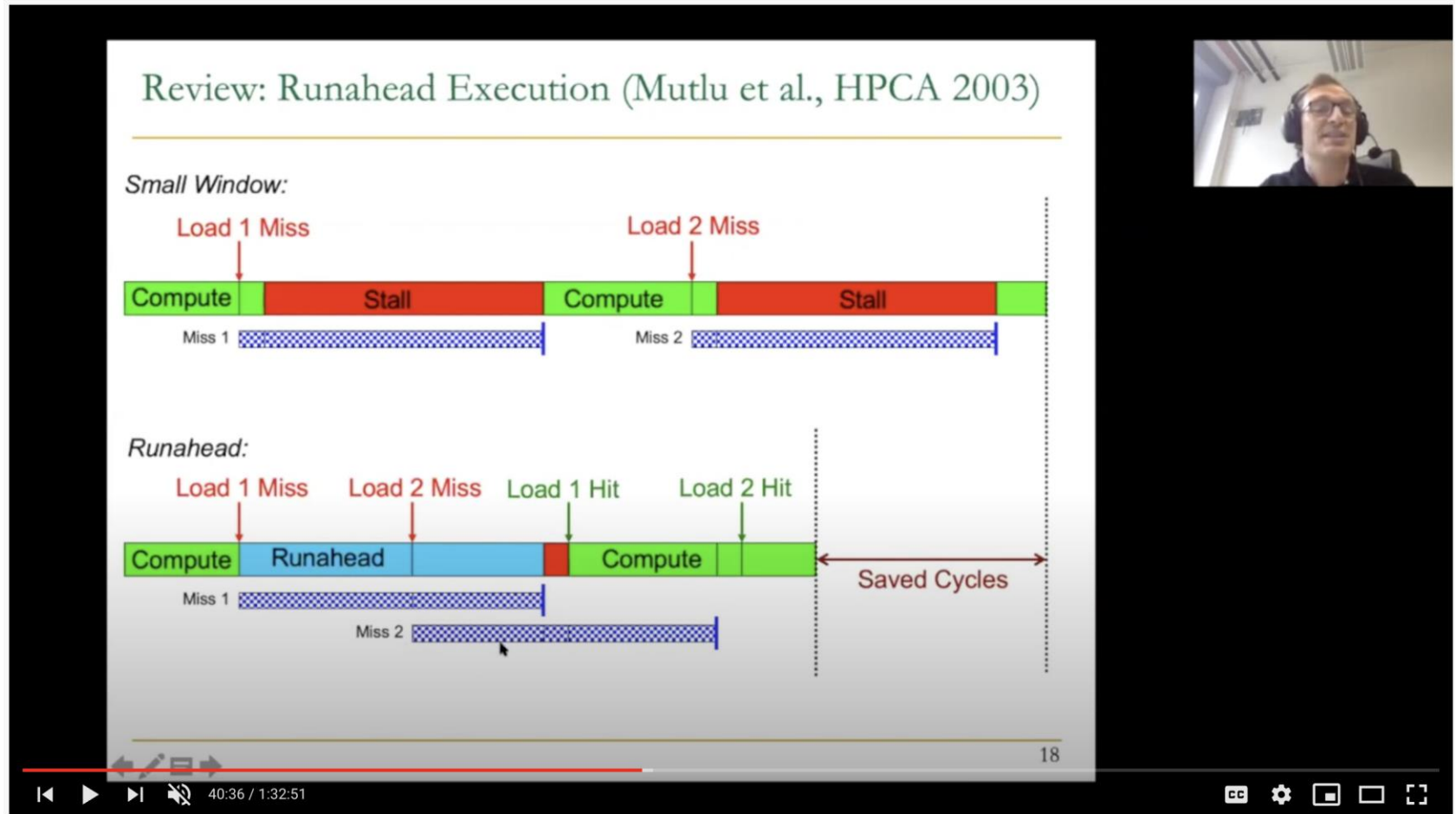
- Runahead Execution is a pioneering paper that opened up new avenues in dynamic prefetching.
- The basic idea of runahead execution effectively increases the instruction window very significantly, without having to increase physical resource size (e.g. the issue queue).
- This seminal paper spawned off a new area of ILP-enhancing microarchitecture research.
- This work has had strong industry impact as evidenced by IBM's POWER6 - Load Lookahead, NVIDIA Denver, and Sun ROCK's hardware scouting.

# More on Runahead Execution

---

- Lecture video from Fall 2020, Computer Architecture:
  - [https://www.youtube.com/watch?v=zPewo6IaJ\\_8](https://www.youtube.com/watch?v=zPewo6IaJ_8)
- Lecture video from Fall 2017, Computer Architecture:
  - <https://www.youtube.com/watch?v=Kj3relihGF4>
- Onur Mutlu,  
**"Efficient Runahead Execution Processors"**  
Ph.D. Dissertation, HPS Technical Report, TR-HPS-2006-007, July 2006. [Slides \(ppt\)](#)  
***Nominated for the ACM Doctoral Dissertation Award by the University of Texas at Austin.***

# More on Runahead Execution (I)



Computer Architecture - Lecture 19a: Execution-Based Prefetching (ETH Zürich, Fall 2020)

395 views • Nov 29, 2020

14 0 SHARE SAVE ...



Onur Mutlu Lectures  
16.5K subscribers

ANALYTICS

EDIT VIDEO

[https://www.youtube.com/watch?v=zPewo6laJ\\_8&list=PL5Q2soXY2Zi9xidyIqBxUz7xRPS-wisBN&index=34](https://www.youtube.com/watch?v=zPewo6laJ_8&list=PL5Q2soXY2Zi9xidyIqBxUz7xRPS-wisBN&index=34)

# More on Runahead Execution (II)

## Runahead Execution in NVIDIA Denver

Reducing the effects of long cache-miss penalties has been a major focus of the micro-architecture, using techniques like prefetching and run-ahead. An aggressive hardware prefetcher implementation detects L2 cache requests and tracks up to 32 streams, each with complex stride patterns.

Run-ahead uses the idle time that a CPU spends waiting on a long latency operation to discover cache and DTLB misses further down the instruction stream and generates prefetch requests for these misses.<sup>1</sup> These prefetch requests warm up the data cache and DTLB well before the actual execution of the instructions that require the data. Run-ahead complements the hardware prefetcher because it's better at prefetching nonstrided streams, and it trains the hardware prefetcher faster than normal execution to yield a combined benefit of 13 percent on SPECint2000 and up to 60 percent on SPECfp2000.

The core includes a hardware prefetch unit that Boggs describes as "aggressive" in preloading the data cache but less aggressive in preloading the instruction cache. It also implements a "run-ahead" feature that continues to execute microcode speculatively after a data-cache miss; this execution can trigger additional cache misses that resolve in the shadow of the first miss. Once the data from the original miss returns, the results of this speculative execution are discarded and execution restarts with the bundle containing the original miss, but run-ahead can preload subsequent data into the cache, thus avoiding a string of time-wasting cache misses. These and other features help Denver outscore Cortex-A15 by more than 2.6x on a memory-read test even when both use the same SoC framework (Tegra K1).

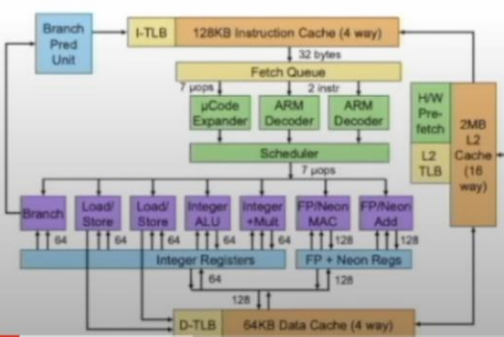


Figure 3. Denver CPU microarchitecture. This design combines a fairly

Boggs+, "Denver: NVIDIA's First 64-Bit ARM Processor," IEEE Micro 2015.

Gwennap, "NVIDIA's First CPU is a Winner," MPR 2014.

Onur Mutlu - Runahead Execution: A Short Retrospective (HPCA Test of Time Award Talk @ HPCA 2021)

1,162 views • Premiered Mar 6, 2021



Onur Mutlu Lectures  
16.5K subscribers

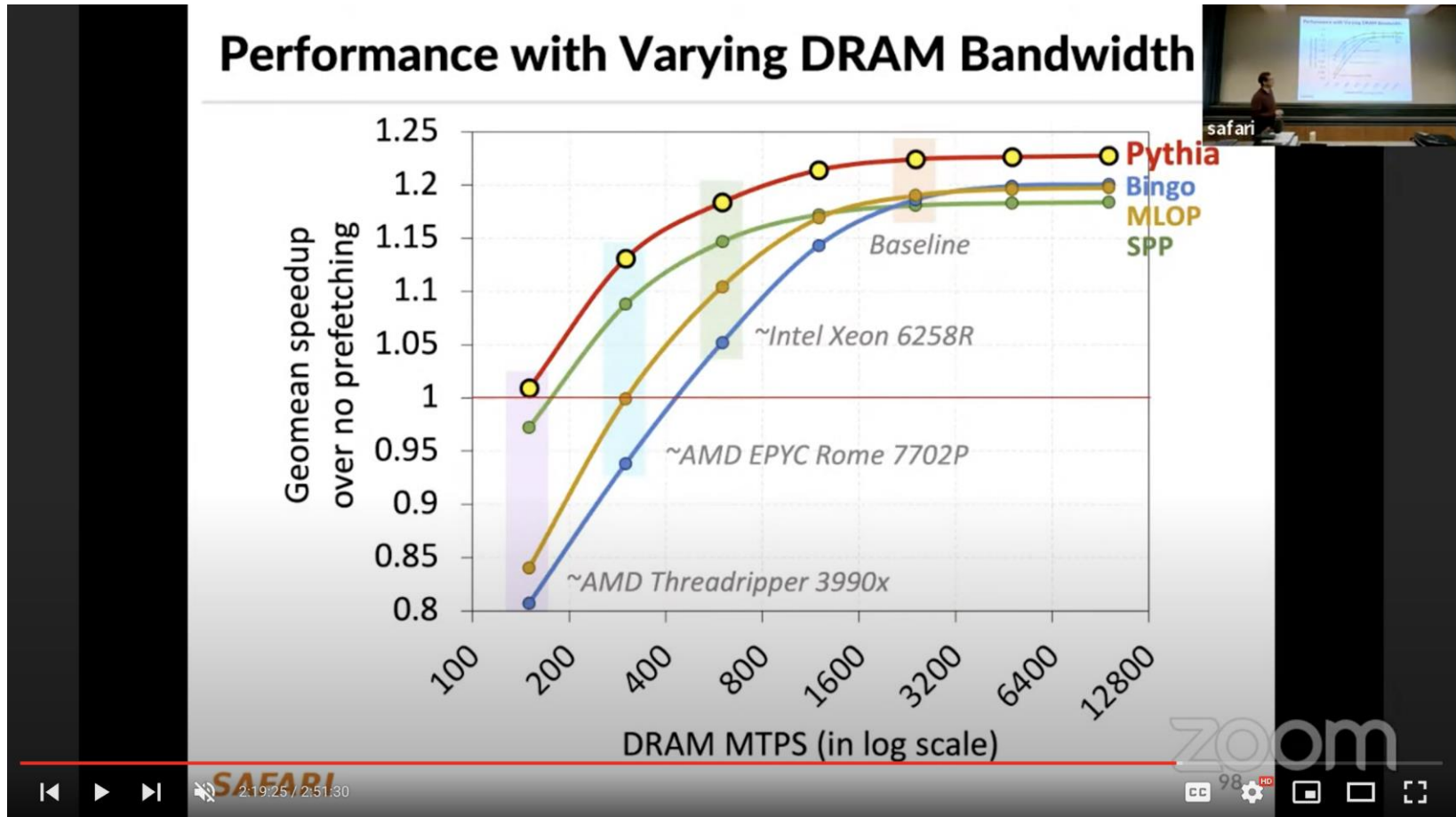
50 0 SHARE SAVE ...

ANALYTICS

EDIT VIDEO

# More Recommended Material on Prefetching

# Lecture on Prefetching: Fall 2022



Livestream - Computer Architecture - ETH Zürich (Fall 2022)

## Computer Architecture - Lecture 16: Prefetching (Fall 2022)



Onur Mutlu Lectures

29.2K subscribers

Analytics

Edit video

32

Share

Download

Clip

Save

...

6.2K views Streamed 5 days ago

Computer Architecture, ETH Zürich, Fall 2022 (<https://safari.ethz.ch/architecture/f...>)

# Lectures on Prefetching (I)

## X86 PREFETCH Instruction

### PREFETCHh—Prefetch Data Into Caches

Opcode	Instruction	64-Bit Mode	Compat/ Leg Mode	Description
OF 18 /1	PREFETCH0 m8	Valid	Valid	Move data from m8 closer to the processor using T0 hint.
OF 18 /2	PREFETCH1 m8	Valid	Valid	Move data from m8 closer to the processor using T1 hint.
OF 18 /3	PREFETCH2 m8	Valid	Valid	Move data from m8 closer to the processor using T2 hint.
OF 18 /0	PREFETCHNTA m8	Valid	Valid	Move data from m8 closer to the processor using NTA hint.

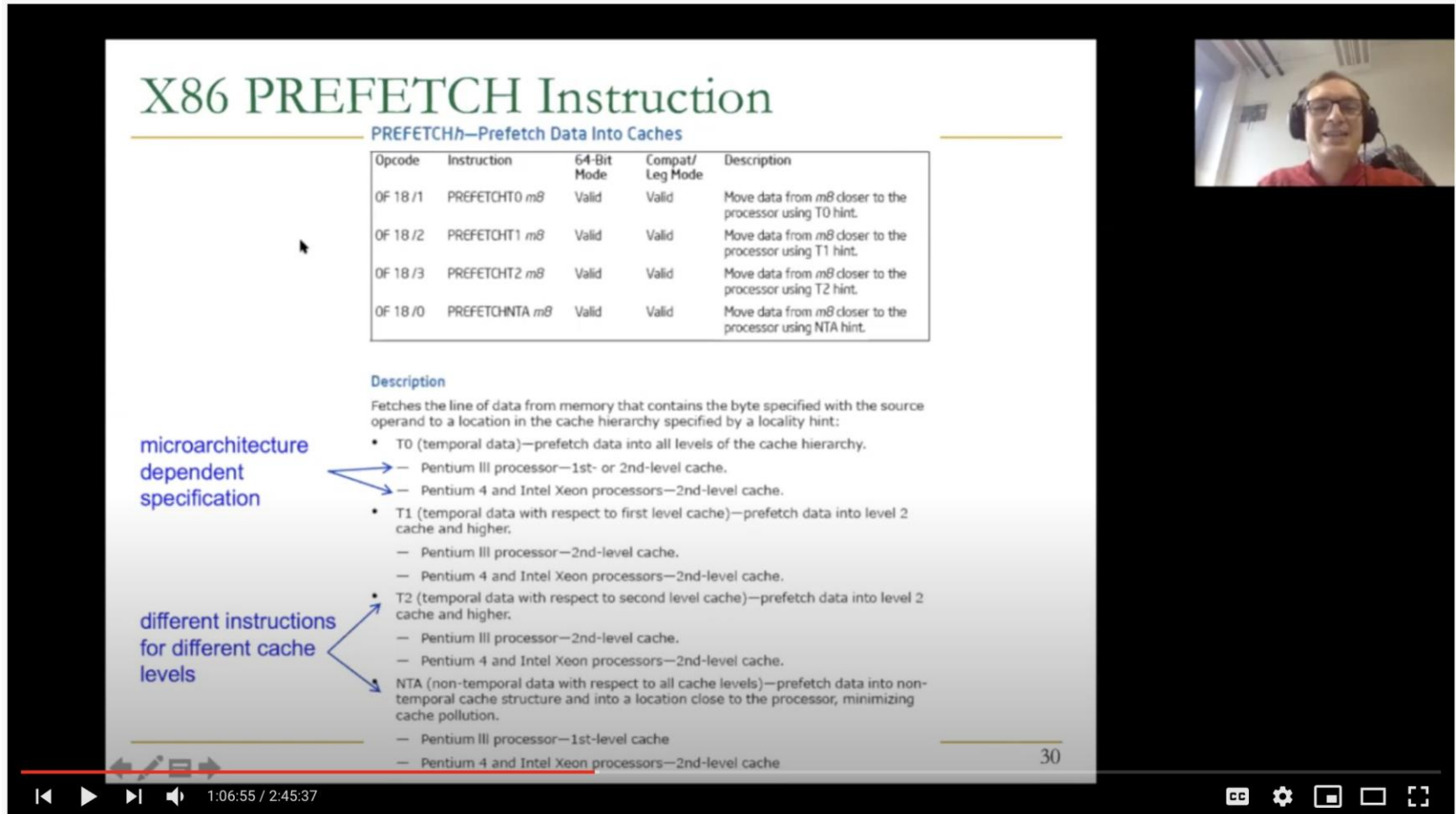
#### Description

Fetches the line of data from memory that contains the byte specified with the source operand to a location in the cache hierarchy specified by a locality hint:

- T0 (temporal data)—prefetch data into all levels of the cache hierarchy.
  - Pentium III processor—1st- or 2nd-level cache.
  - Pentium 4 and Intel Xeon processors—2nd-level cache.
- T1 (temporal data with respect to first level cache)—prefetch data into level 2 cache and higher.
  - Pentium III processor—2nd-level cache.
  - Pentium 4 and Intel Xeon processors—2nd-level cache.
- T2 (temporal data with respect to second level cache)—prefetch data into level 2 cache and higher.
  - Pentium III processor—2nd-level cache.
  - Pentium 4 and Intel Xeon processors—2nd-level cache.
- NTA (non-temporal data with respect to all cache levels)—prefetch data into non-temporal cache structure and into a location close to the processor, minimizing cache pollution.
  - Pentium III processor—1st-level cache
  - Pentium 4 and Intel Xeon processors—2nd-level cache

microarchitecture dependent specification

different instructions for different cache levels



Computer Architecture - Lecture 18: Prefetching (ETH Zürich, Fall 2020)

1,203 views • Nov 29, 2020

👍 26    💬 0    ➦ SHARE    ⚙️ SAVE    ⋮



**Onur Mutlu Lectures**  
16.5K subscribers

ANALYTICS

EDIT VIDEO

<https://www.youtube.com/watch?v=xZmDyj0g3Pw&list=PL5Q2soXY2Zi9xidylgBxUz7xRPS-wisBN&index=33>



# Lectures on Prefetching (II)

## Thread-Based Pre-Execution

The diagram illustrates thread-based pre-execution. A main thread (grey bar) contains a 'BRANCH' instruction (blue box) followed by a 'LOAD' instruction (blue box). A 'fork' operation creates a subordinate thread (grey bar) that starts at the 'LOAD' instruction. A 'prediction' arrow points from the 'BRANCH' instruction to the subordinate thread. A 'cache miss' box points to the 'LOAD' instruction, and a 'cache hit' box points to the subordinate thread. A bracket labeled 'speedup' indicates the time saved by the subordinate thread. The video player interface shows a progress bar at 12:23 / 1:32:51 and a video number 7.

- Dubois and Song, “**Assisted Execution**,” USC Tech Report 1998.
- Chappell et al., “**Simultaneous Subordinate Microthreading (SSMT)**,” ISCA 1999.
- Zilles and Sohi, “**Execution-based Prediction Using Speculative Slices**”, ISCA 2001.

Computer Architecture - Lecture 19a: Execution-Based Prefetching (ETH Zürich, Fall 2020)

424 views • Nov 29, 2020

16 0 SHARE SAVE ...



Onur Mutlu Lectures  
16.7K subscribers

ANALYTICS

EDIT VIDEO



# Lectures on Prefetching (III)

# Runahead Execution in NVIDIA Denver

Reducing the effects of long cache-miss penalties has been a major focus of the microarchitecture, using techniques like prefetching and run-ahead. An aggressive hardware prefetcher implementation detects L2 cache requests and tracks up to 32 streams, each with complex stride patterns.

Run-ahead uses the idle time that a CPU spends waiting on a long latency operation to discover cache and DTLB misses further down the instruction stream and generates prefetch requests for these misses.<sup>1</sup> These prefetch requests warm up the data cache and DTLB well before the actual execution of the instructions that require the data. Run-ahead complements the hardware prefetcher because it's better at prefetching nonstrided streams, and it trains the hardware prefetcher faster than normal execution to yield a combined benefit of 13 percent on SPECint2000 and up to 60 percent on SPECfp2000.

The core includes a hardware prefetch unit that Bogg describes as “aggressive” in preloading the data cache but less aggressive in preloading the instruction cache. It also implements a “run-ahead” feature that continues to execute microcode speculatively after a data-cache miss; this execution can trigger additional cache misses that resolve in the shadow of the first miss. Once the data from the original miss returns, the results of this speculative execution are discarded and execution restarts with the bundle containing the original miss, but run-ahead can preload subsequent data into the cache, thus avoiding a string of time-wasting cache misses. These and other features help Denver outscore Cortex-A15 by more than 2.6x on a memory-read test even when both use the same SoC framework (Tegra K1).

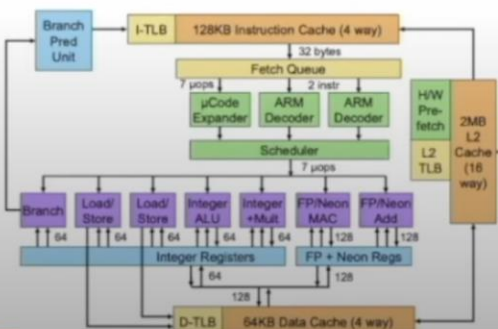


Figure 3. Denver CPU microarchitecture. This design combines a fairly

Boggs+, "Denver: NVIDIA's First 64-Bit ARM Processor," IEEE Micro 2015.

Gwennap, "NVIDIA's First CPU is a Winner," MPR 2014.



**Onur Mutlu Lectures**  
16.5K subscribers

Onur Mutlu - Runahead Execution: A Short Retrospective (HPCA Test of Time Award Talk @ HPCA 2021)

1,162 views • Premiered Mar 6, 2021

50 0 SHARE SAVE ...

ANALYTICS

[EDIT VIDEO](#)

# Lectures on Prefetching (IV)

## Software Prefetching (II)

```
for (i=0; i<N; i++) {  
    __prefetch(a[i+8]);  
    __prefetch(b[i+8]);  
    sum += a[i]*b[i];  
}
```

```
while (p) {  
    __prefetch(p->next);  
    work(p->data);  
    p = p->next;  
}
```

```
while (p) {  
    __prefetch(p->next->next->next);  
    work(p->data);  
    p = p->next;  
}
```

Which one is better?

- Can work for very regular array-based access patterns. Issues:
  - Prefetch instructions take up processing/execution bandwidth
  - How early to prefetch? Determining this is difficult
    - Prefetch distance depends on hardware implementation (memory latency, cache size, time between loop iterations) → portability?
    - Going too far back in code reduces accuracy (branches in between)
  - Need “special” prefetch instructions in ISA?
    - Alpha load into register 31 treated as prefetch (r31==0)
    - PowerPC *dcbt* (data cache block touch) instruction
  - Not easy to do for pointer-based data structures

40

Lecture 25: Prefetching - Carnegie Mellon - Computer Architecture 2015 - Onur Mutlu

5,216 views • Apr 3, 2015

39 0 SHARE SAVE ...



Carnegie Mellon Computer Architecture  
23.3K subscribers

SUBSCRIBED



<https://www.youtube.com/watch?v=ibPL7T9iEwY&list=PL5PHm2jkkXmi5Cxxl7b3JCL1TWybTDtKq&index=29>

# Lectures on Prefetching (V)

**Address Correlation Based Prefetching (II)**

The diagram illustrates the Address Correlation Based Prefetching (II) mechanism. It shows a 'Cache Block Addr' pointing to a table with 'Cache Block Addr (tag)' and a list of addresses. To the right, there are two tables: 'Prefetch Candidate 1' and 'Confidence', and another table with 'Prefetch Candidate N' and 'Confidence'. The tables are connected by arrows, indicating the flow of data and the correlation between addresses.

- Idea: Record the likely-next addresses (B, C, D) after seeing an address A
  - Next time A is accessed, prefetch B, C, D
  - A is said to be correlated with B, C, D
- Prefetch up to N next addresses to increase *coverage*
- Prefetch accuracy can be improved by using multiple addresses as key for the next address: (A, B) → (C)  
(A,B) correlated with C
- Joseph and Grunwald, "Prefetching using Markov Predictors," ISCA 1997.

10

Lecture 26. More Prefetching and Emerging Memory Technologies - CMU - Comp. Arch. 2015 - Onur Mutlu

3,642 views • Apr 6, 2015

26 0 SHARE SAVE ...



Carnegie Mellon Computer Architecture  
23.3K subscribers

SUBSCRIBED



<https://www.youtube.com/watch?v=TUFins4z6o4&list=PL5PHm2jkkXmi5CxxI7b3JCL1TWybTDtKq&index=30>

# Lectures on Prefetching

---

- **Computer Architecture, Fall 2020, Lecture 18**
  - ❑ Prefetching (ETH, Fall 2020)
  - ❑ <https://www.youtube.com/watch?v=xZmDyj0g3Pw&list=PL5Q2soXY2Zi9xidyIgBxUz7xRPS-wisBN&index=33>
- **Computer Architecture, Fall 2020, Lecture 19a**
  - ❑ Execution-Based Prefetching (ETH, Fall 2020)
  - ❑ [https://www.youtube.com/watch?v=zPewo6IaJ\\_8&list=PL5Q2soXY2Zi9xidyIgBxUz7xRPS-wisBN&index=34](https://www.youtube.com/watch?v=zPewo6IaJ_8&list=PL5Q2soXY2Zi9xidyIgBxUz7xRPS-wisBN&index=34)
- **Computer Architecture, Spring 2015, Lecture 25**
  - ❑ Prefetching (CMU, Spring 2015)
  - ❑ <https://www.youtube.com/watch?v=ibPL7T9iEwY&list=PL5PHm2jkkXmi5CxxI7b3JCL1TWybTDtKq&index=29>
- **Computer Architecture, Spring 2015, Lecture 26**
  - ❑ More Prefetching (CMU, Spring 2015)
  - ❑ <https://www.youtube.com/watch?v=TUFins4z6o4&list=PL5PHm2jkkXmi5CxxI7b3JCL1TWybTDtKq&index=30>

# Research Opportunities

# Computer Architecture Research

---

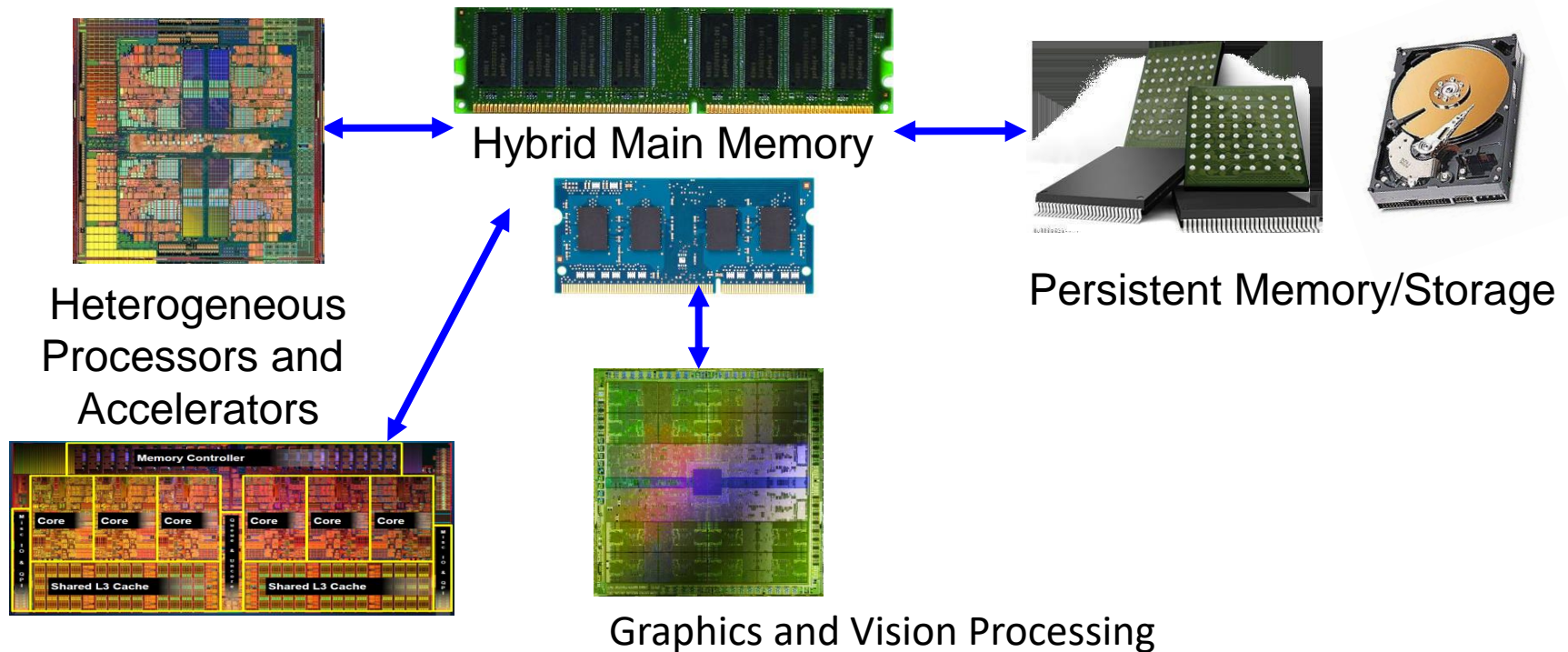
- If you want to do research in any of the covered topics or any topic in Comp Arch, HW/SW Interaction & related areas
  - We have many projects and a great environment to perform top-notch research, bachelor's/master's/semester projects
  - Talk with me (email, whatsapp, etc.) & apply online
- Many research topics and projects
  - Memory (DRAM, NVM, Flash, SW/HW issues, emerging tech)
  - Processing in Memory
  - Hardware Security
  - New Computing Paradigms
  - Machine Learning for System Design
  - System Design for AI/ML, Health, Genomics, Medicine
  - ...



# Current Research Mission

---

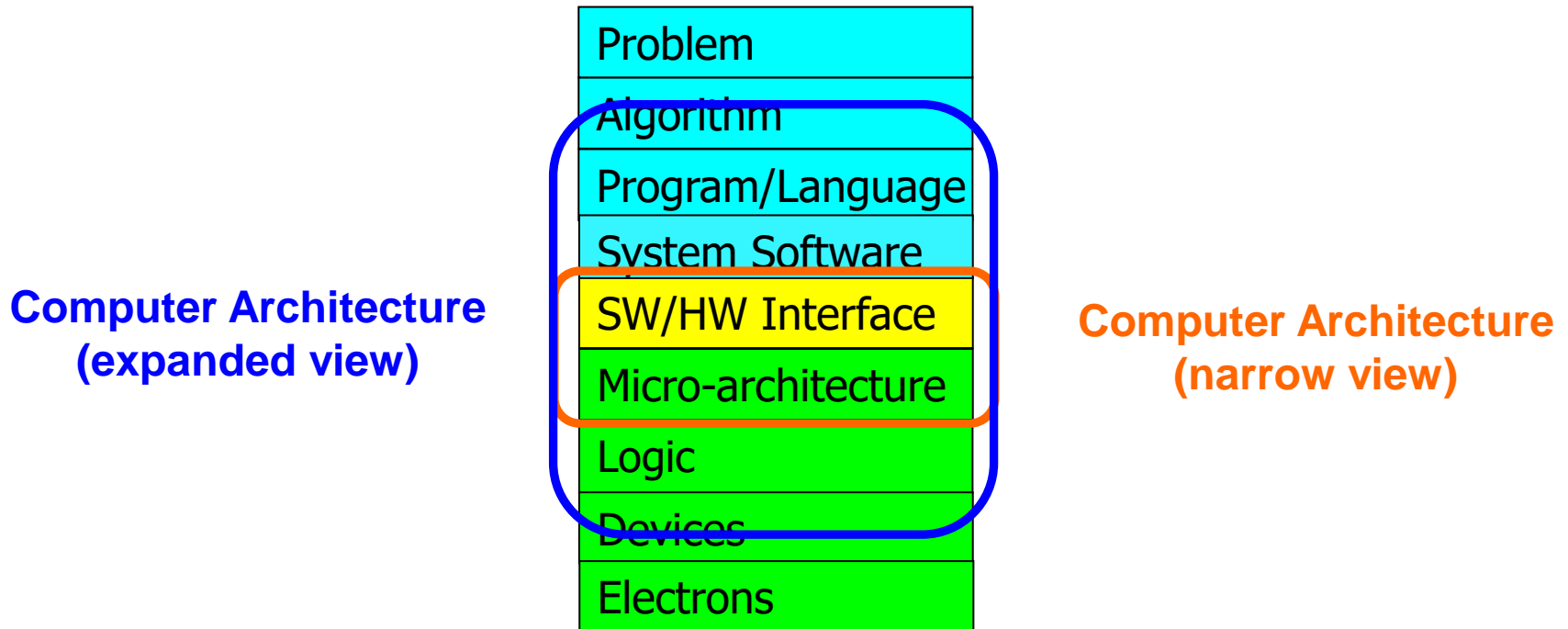
*Computer architecture, HW/SW, systems, bioinformatics, security*



**Build fundamentally better architectures**

# The Transformation Hierarchy

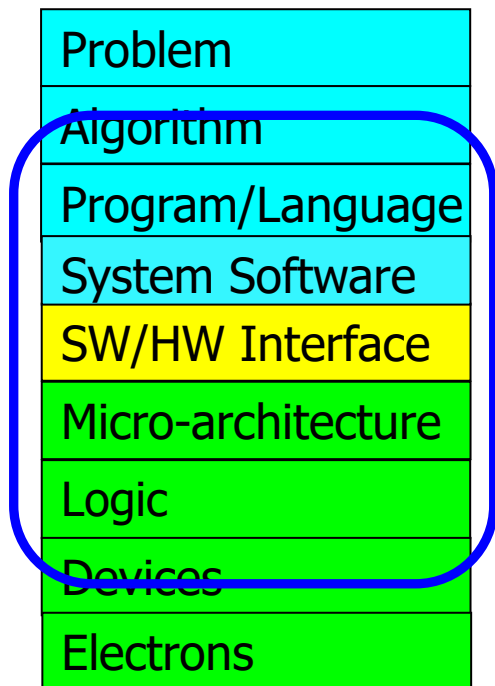
---





# SAFARI Research Mission & Major Topics

## Build fundamentally better architectures



**Broad research**

**spanning apps, systems, logic  
with architecture at the center**



- Data-centric systems: memory/storage systems
  - **Proc. in Memory/Storage**, emerging tech, DRAM
- Fundamentally secure/reliable/safe architectures
  - **RowHammer**; patchable HW; **secure memory**
- Low-latency & predictable architectures
  - **Low-latency, low-energy** yet low-cost memory
  - **QoS-aware** and predictable memory systems
- Systems for ML/AI/Genomics/Health/Graphs
  - Algorithm/architecture co-design; accelerators
- Data-driven and data-aware architectures
  - **ML/AI for architectural control and design**
  - **Expressive memory** and expressive systems
- Ultra-fast & efficient genome analysis

# Open Source Tools: SAFARI GitHub



## SAFARI Research Group at ETH Zurich and Carnegie Mellon University

Site for source code and tools distribution from SAFARI Research Group at ETH Zurich and Carnegie Mellon University.

📍 ETH Zurich and Carnegie Mellon U... 🔗 <https://safari.ethz.ch/> ✉ [omutlu@gmail.com](mailto:omutlu@gmail.com)

🏠 Overview 📁 Repositories 71 📁 Projects 📁 Packages 👤 Teams 1 👤 People 44 ⚙ Settings

### Pinned

Customize pins

#### 📁 ramulator Public

A Fast and Extensible DRAM Simulator, with built-in support for modeling many different DRAM technologies including DDRx, LPDDRx, GDDRx, WIOx, HBMx, and various academic proposals. Described in the...

🔴 C++ ☆ 311 🍷 161

#### 📁 prim-benchmarks Public

PRIM (Processing-In-Memory benchmarks) is the first benchmark suite for a real-world processing-in-memory (PIM) architecture. PRIM is developed to evaluate, analyze, and characterize the first publ...

⬛ C ☆ 53 🍷 21

#### 📁 DAMOV Public

DAMOV is a benchmark suite and a methodical framework targeting the study of data movement bottlenecks in modern applications. It is intended to study new architectures, such as near-data processin...

🔴 C++ ☆ 26 🍷 4

#### 📁 SneakySnake Public

SneakySnake🐍 is the first and the only pre-alignment filtering algorithm that works efficiently and fast on modern CPU, FPGA, and GPU architectures. It greatly (by more than two orders of magnitude...

🔵 VHDL ☆ 41 🍷 8

#### 📁 MQSim Public

MQSim is a fast and accurate simulator modeling the performance of modern multi-queue (MQ) SSDs as well as traditional SATA based SSDs. MQSim faithfully models new high-bandwidth protocol implement...

🔴 C++ ☆ 146 🍷 93

#### 📁 rowhammer Public

Source code for testing the Row Hammer error mechanism in DRAM devices. Described in the ISCA 2014 paper by Kim et al. at [http://users.ece.cmu.edu/~omutlu/pub/dram-row-hammer\\_isca14.pdf](http://users.ece.cmu.edu/~omutlu/pub/dram-row-hammer_isca14.pdf).

⬛ C ☆ 189 🍷 41

<https://github.com/CMU-SAFARI/>

# Onur Mutlu's SAFARI Research Group

*Computer architecture, HW/SW, systems, bioinformatics, security, memory*

<https://safari.ethz.ch/safari-newsletter-april-2020/>



## Think BIG, Aim HIGH!

**SAFARI**

<https://safari.ethz.ch>



# SAFARI Newsletter January 2021 Edition

- <https://safari.ethz.ch/safari-newsletter-january-2021/>



**SAFARI**  
SAFARI Research Group

Newsletter  
January 2021

*Think Big, Aim High, and  
Have a Wonderful 2021!*



Dear SAFARI friends,

Happy New Year! We are excited to share our group highlights with you in this second edition of the SAFARI newsletter (You can find the first edition from April 2020 [here](#)). 2020 has

# SAFARI Newsletter December 2021 Edition

- <https://safari.ethz.ch/safari-newsletter-december-2021/>

**SAFARI**  
SAFARI Research Group

*Think Big, Aim High*

**ETH** zürich



View in your browser  
December 2021



# Acknowledgments

---



Think BIG, Aim HIGH!

<https://safari.ethz.ch>

---

# A Talk on Our Research & Teaching



The video player shows a presentation slide with the title "Applying to Grad School & Doing Impactful Research" in a green serif font, enclosed in a thin gold border. Below the title, the speaker's name "Onur Mutlu" is listed, followed by his email "omutlu@gmail.com" and his website "https://people.inf.ethz.ch/omutlu". The date "13 June 2020" and the event "Undergraduate Architecture Mentoring Workshop @ ISCA 2021" are also displayed. At the bottom of the slide, the logos for "SAFARI", "ETH zürich", and "Carnegie Mellon" are shown. The video player interface includes a progress bar at 0:27 / 50:31, a small video feed of the speaker in the top right corner, and a YouTube interface at the bottom with the video title, view count (1,563 views), premiere date (Jun 16, 2021), like/dislike counts (74/1), and share/save options. The channel name "Onur Mutlu Lectures" with 17.2K subscribers is also visible.

Applying to Grad School  
& Doing Impactful Research

Onur Mutlu  
[omutlu@gmail.com](mailto:omutlu@gmail.com)  
<https://people.inf.ethz.ch/omutlu>  
13 June 2020  
Undergraduate Architecture Mentoring Workshop @ ISCA 2021

SAFARI ETH zürich Carnegie Mellon

Arch. Mentoring Workshop @ISCA'21 - Applying to Grad School & Doing Impactful Research - Onur Mutlu  
1,563 views • Premiered Jun 16, 2021

Onur Mutlu Lectures  
17.2K subscribers

Panel talk at Undergraduate Architecture Mentoring Workshop at ISCA 2021  
(<https://sites.google.com/wisc.edu/uar...>)



# An Interview on Computing Futures



Interview with Onur Mutlu @ ISCA 2019 on computing research & education (after Maurice Wilkes Award)

6,749 views • Oct 19, 2019

👍 195 🗨️ 0 ➦ SHARE ➦ SAVE ...



**Onur Mutlu Lectures**  
19.1K subscribers

ANALYTICS

EDIT VIDEO



# Computer Architecture Research

---

- If you want to do research in any of the covered topics or any topic in Comp Arch, HW/SW Interaction & related areas
  - We have many projects and a great environment to perform top-notch research, bachelor's/master's/semester projects
  - Talk with me (email, whatsapp, etc.) & apply online
  
- Many research topics and projects
  - Memory (DRAM, NVM, Flash, SW/HW issues, emerging tech)
  - Processing in Memory
  - Hardware Security
  - New Computing Paradigms
  - Machine Learning for System Design
  - System Design for AI/ML, Health, Genomics, Medicine
  - ...

# 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 4<sup>th</sup> and 4<sup>th</sup> revised eds.)

# Multiprocessors and Issues in Multiprocessing

# Flynn's Taxonomy of Computers

---

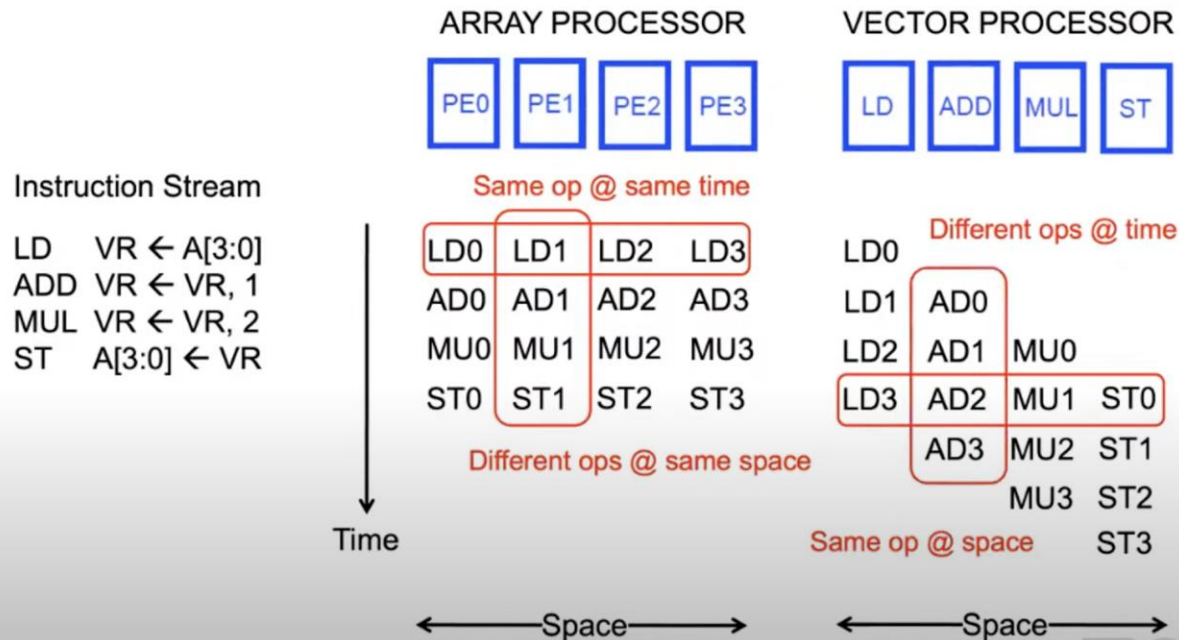
- Mike Flynn, “[Very High-Speed Computing Systems](#),” Proc. of IEEE, 1966
  
- **SISD**: Single instruction operates on single data element
- **SIMD**: Single instruction operates on multiple data elements
  - Array processor
  - Vector processor
- **MISD**: Multiple instructions operate on single 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



Juan Gomez L...



Livestream - Digital Design and Computer Architecture - ETH Zürich (Spring 2022)

Digital Design & Computer Arch. - Lecture 20: SIMD Processing (Vector and Array Processors) (S 2022)



Onur Mutlu Lectures

29.3K subscribers

Analytics

Edit video

28

Share

Download

Clip

Save

1,124 views Streamed live on May 12, 2022

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



# 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.

Minplayer (I)

1:26:56 / 1:30:37

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

842 views • Premiered May 6, 2022

35 DISLIKE SHARE CLIP SAVE ...



Onur Mutlu Lectures

24.5K subscribers

SUBSCRIBED



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

---

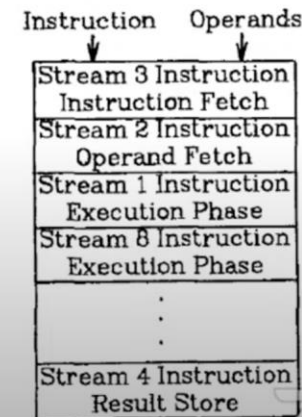
- Coarse grained
  - Quantum based
  - Event based (switch-on-event multithreading), e.g., switch on L3 miss
  
- Fine grained
  - Cycle by cycle
  - Thornton, “[CDC 6600: Design of a Computer](#),” 1970.
  - Burton Smith, “[A pipelined, shared resource MIMD computer](#),” ICPP 1978.
  
- Simultaneous
  - 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



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

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

63 DISLIKE SHARE SAVE ...

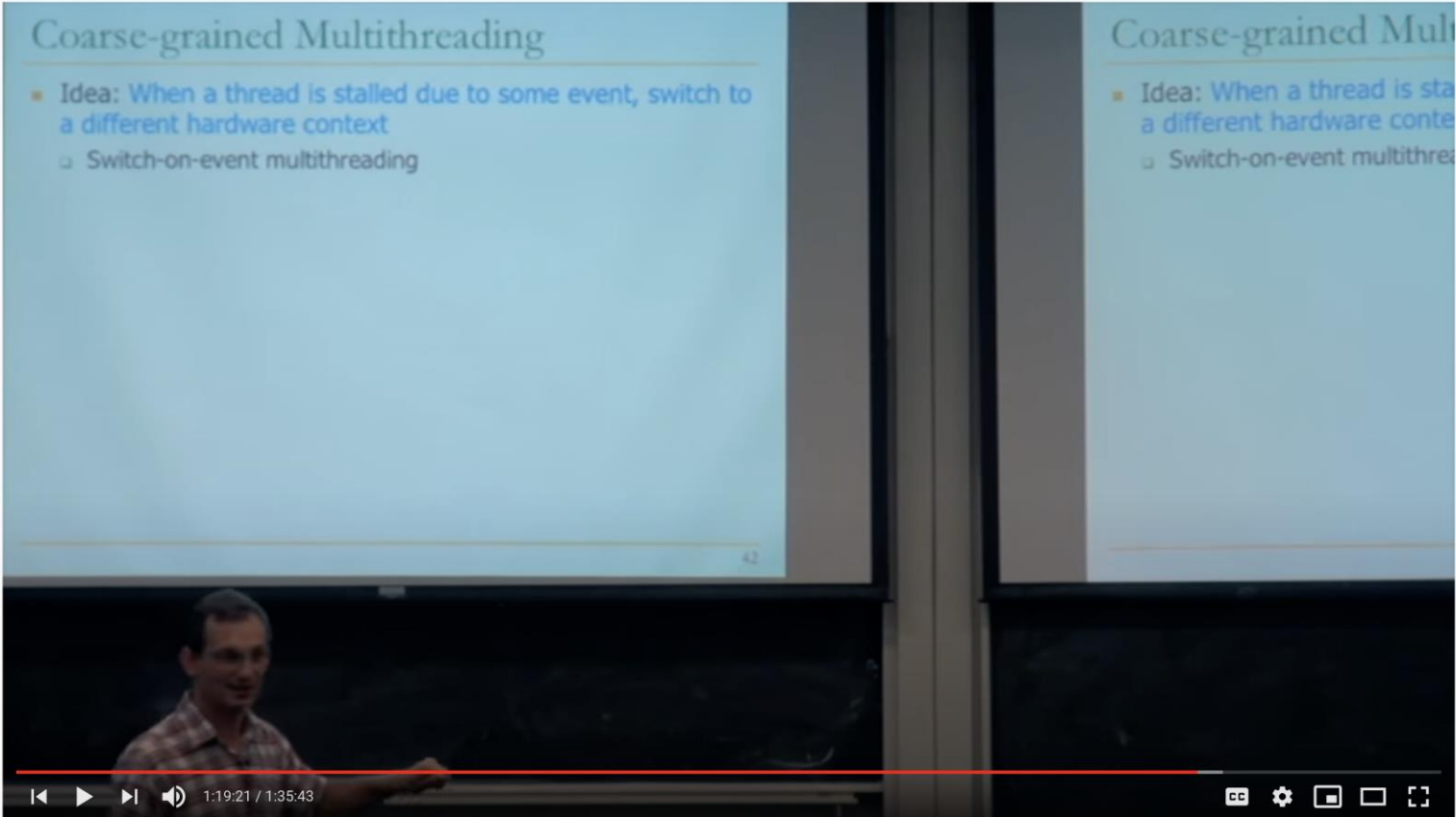


Onur Mutlu Lectures  
20.4K subscribers

ANALYTICS

EDIT VIDEO

# More on Multithreading (I)



The video player shows a lecture slide titled "Coarse-grained Multithreading". The slide content is as follows:


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

The video player interface includes a progress bar at 1:19:21 / 1:35:43, a like count of 10, and a comment count of 0. The video is titled "Carnegie Mellon - Parallel Computer Architecture 2013 - Onur Mutlu - Lec 9 - Multithreading" and has 1,252 views as of Nov 19, 2013. The channel is "Carnegie Mellon Computer Architecture" with 23K subscribers. The video description includes "Lecture 9: Multithreading", "Lecturer: Prof. Onur Mutlu (<http://users.ece.cmu.edu/~omutlu/>)", and "Date: September 26, 2013.".

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

1,252 views • Nov 19, 2013

10 0 SHARE SAVE ...

 Carnegie Mellon Computer Architecture  
23K subscribers

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

ANALYTICS EDIT VIDEO

# More on Multithreading (II)

Intel Pentium 4 Hyperthreading

- latency load handling
- multi-level scheduling window
- partitioned structures
- I-B
- 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.

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

4-est

soft line fetch

Assigning Confidence to Conditional Branches

MICRO HT6

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

1,594 views • Sep 21, 2013

11 0 SHARE SAVE ...



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.

SUBSCRIBED



# More on Multithreading (III)

g (Tandem, Compaq Himalaya)

Microprocessor R1 ← (R2) and R2 ← (R1)

Input Register and Output Register

Memory covered by ECC  
RAID array covered by parity  
Servernet covered by CRC

the processor, compare the results of two  
re committing an instruction

Lockstepping (Tandem, Compaq Himalaya)

Microprocessor R1 ← (R2) and R2 ← (R1)

Input Register and Output Register

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

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

1,132 views • Sep 21, 2013

8 0 SHARE SAVE ...



**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.

SUBSCRIBED



<https://www.youtube.com/onurmutlulectures>

# More on Multithreading (IV)



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

915 views • Sep 21, 2013

9 0 SHARE SAVE ...



**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.

SUBSCRIBED



<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\\_tGtUlynnnyV6D&index=51](https://www.youtube.com/watch?v=iqi9wFqFiNU&list=PL5PHm2jkkXmgDN1PLwOY_tGtUlynnnyV6D&index=51)

## ■ Parallel Computer Architecture, Fall 2012, Lecture 10

- ❑ Multithreading II (CMU, Fall 2012)
- ❑ [https://www.youtube.com/watch?v=e8lfl6MbILg&list=PL5PHm2jkkXmgDN1PLwOY\\_tGtUlynnnyV6D&index=52](https://www.youtube.com/watch?v=e8lfl6MbILg&list=PL5PHm2jkkXmgDN1PLwOY_tGtUlynnnyV6D&index=52)

## ■ Parallel Computer Architecture, Fall 2012, Lecture 13

- ❑ Multithreading III (CMU, Fall 2012)
- ❑ [https://www.youtube.com/watch?v=7vkDpZ1-hHM&list=PL5PHm2jkkXmgDN1PLwOY\\_tGtUlynnnyV6D&index=53](https://www.youtube.com/watch?v=7vkDpZ1-hHM&list=PL5PHm2jkkXmgDN1PLwOY_tGtUlynnnyV6D&index=53)

## ■ Parallel Computer Architecture, Fall 2012, Lecture 15

- ❑ Speculation I (CMU, Fall 2012)
- ❑ [https://www.youtube.com/watch?v=-hbmzIDe0sA&list=PL5PHm2jkkXmgDN1PLwOY\\_tGtUlynnnyV6D&index=54](https://www.youtube.com/watch?v=-hbmzIDe0sA&list=PL5PHm2jkkXmgDN1PLwOY_tGtUlynnnyV6D&index=54)

# Limits of Parallel Speedup



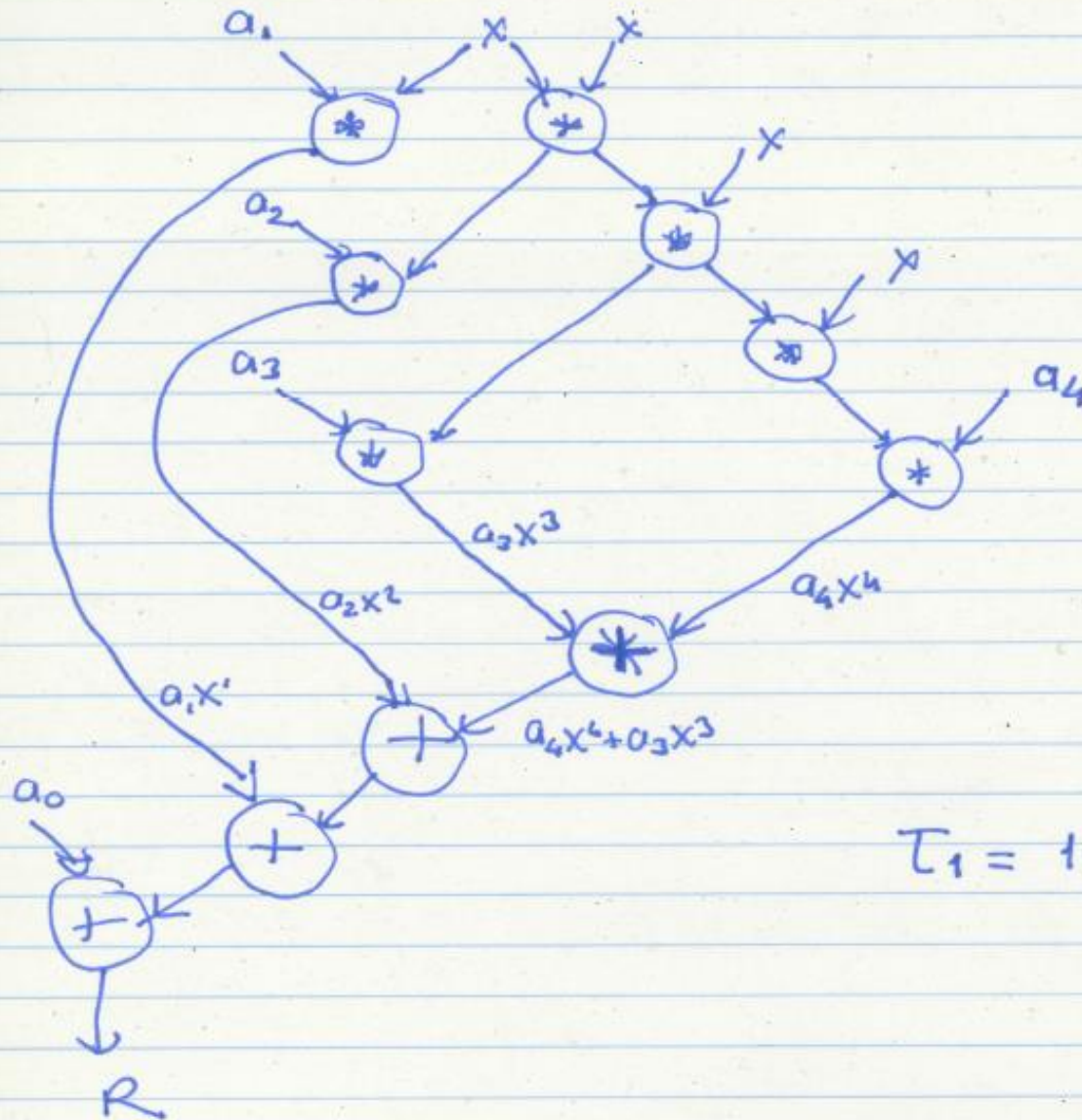
# Parallel Speedup Example

---

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

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

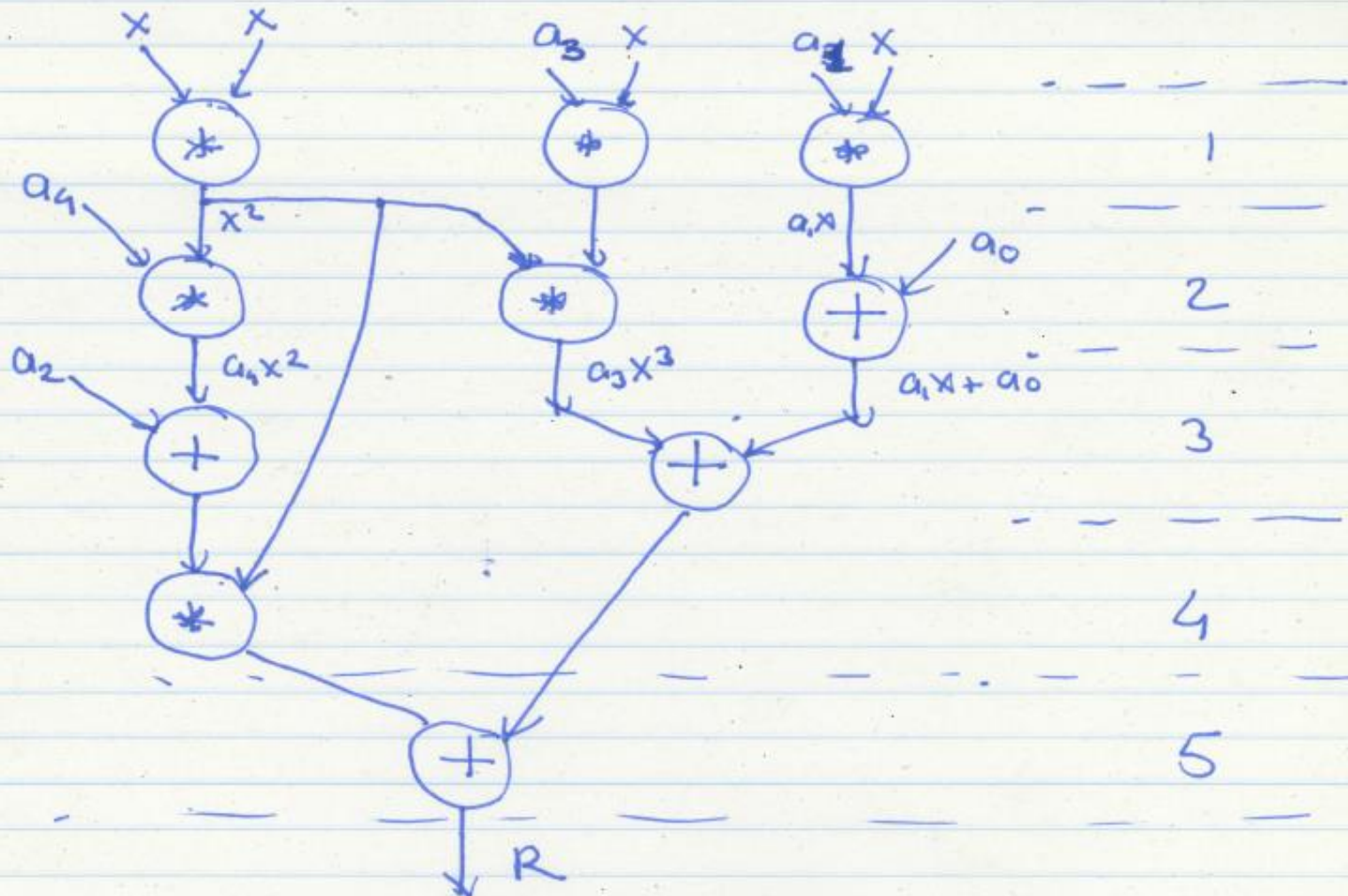
Single processor : 11 operations (data flow graph) <sup>draw the</sup>



$T_1 = 11$  cycles

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

Three processors :  $T_3$  (excc.time with 3 proc.)



$$T_3 = \underline{5 \text{ cycles}}$$

# Speedup with 3 Processors

---

$$T_3 = \underline{5 \text{ cycles}}$$

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

$$\left( \frac{T_1}{T_3} \right)$$

Is this a fair comparison?



# Revisiting the Single-Processor Algorithm

---

Revisit  $T_1$

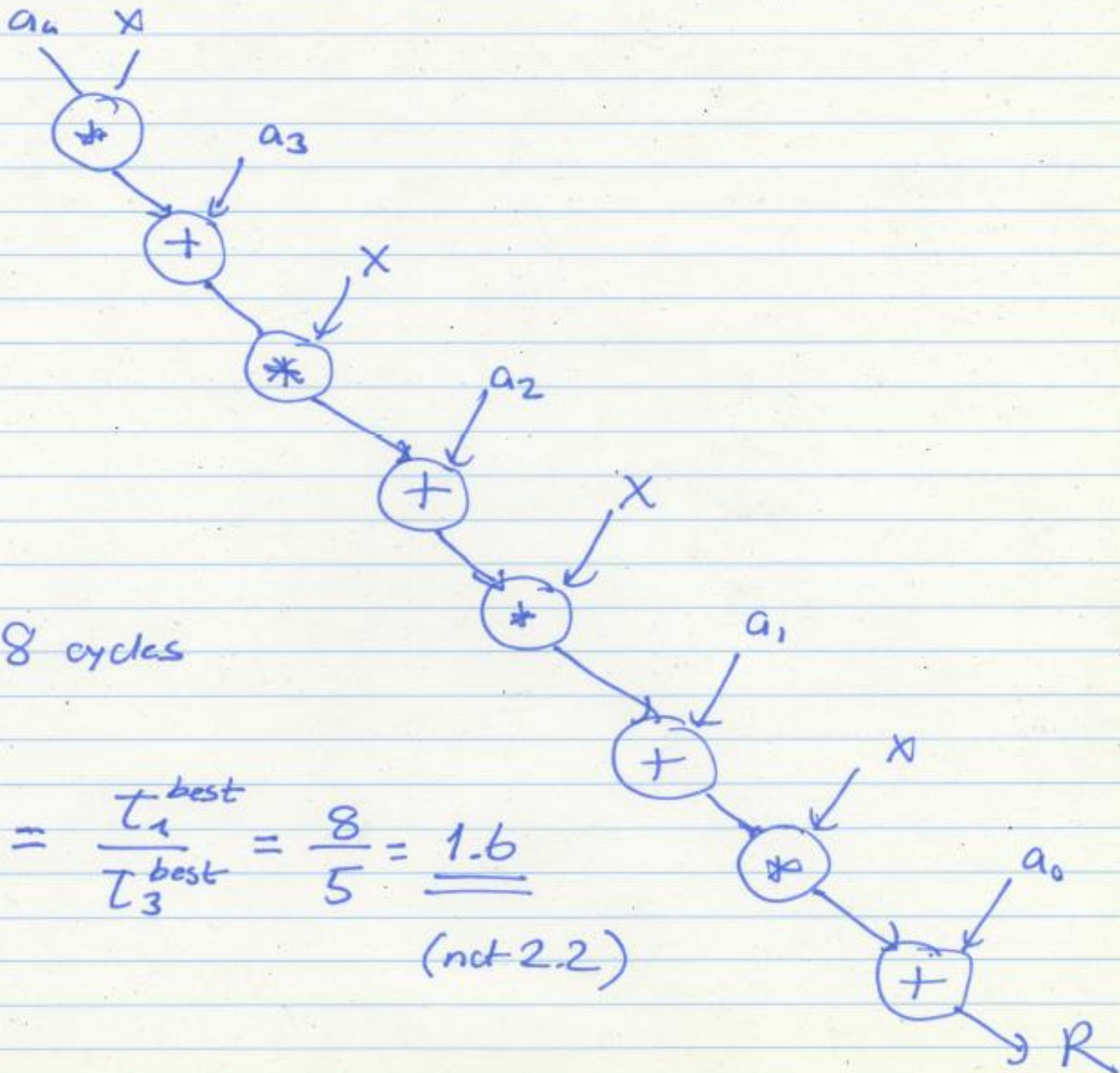
Better single-processor algorithm:

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

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

(Horner's method)

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

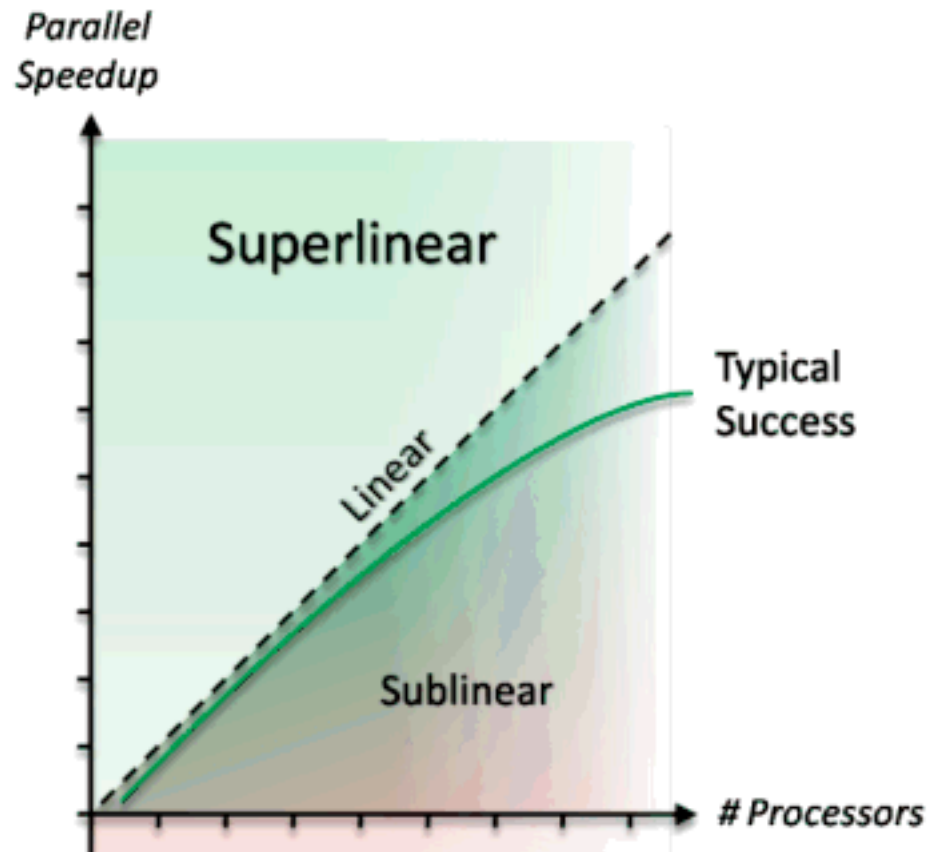


$T_1 = 8$  cycles

Speedup  
with  
3 procs.  $= \frac{T_1^{\text{best}}}{T_3^{\text{best}}} = \frac{8}{5} = \underline{\underline{1.6}}$   
(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  $\rightarrow$  unfair
- **Cache/memory effects**  
More processors  $\rightarrow$   
more cache or memory  $\rightarrow$   
fewer misses in cache/mem



# Utilization, Redundancy, Efficiency

---

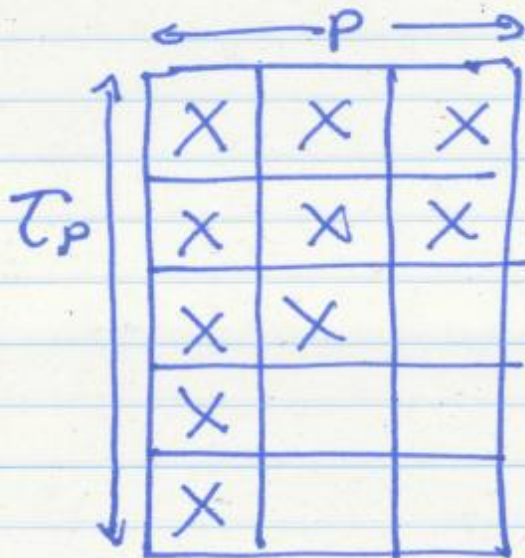
- Traditional metrics
  - Assume all P processors are tied up for parallel computation
- Utilization: How much processing capability is used
  - $U = (\# \text{ Operations in parallel version}) / (\text{processors} \times \text{Time})$
- Redundancy: how much extra work is done with parallel processing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



# Utilization of a Multiprocessor

## Multiprocessor metrics

Utilization : How much processing capability we use



$$U = \frac{10 \text{ operations (in parallel version)}}{3 \text{ processors} \times 5 \text{ time units}}$$
$$= \frac{10}{15}$$

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

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  $\geq 1$

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

$$E = \frac{1 \cdot T_1^{\text{best}}}{p \cdot T_p^{\text{best}}} \quad \begin{array}{l} \text{(tying up 1 proc for } T_p \text{ time units)} \\ \text{(tying up } p \text{ proc. for } T_p \text{ time units)} \end{array}$$

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

# Amdahl's Law and Caveats of Parallelism

# Amdahl's Law

---

- Amdahl's Law

- f: Parallelizable fraction of a program
- N: Number of processors

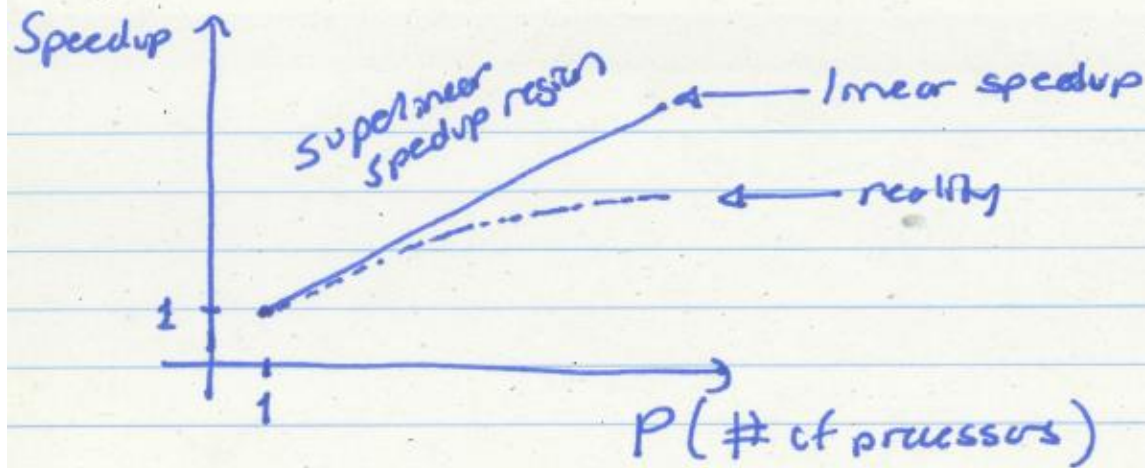
$$\text{Speedup} = \frac{1}{1 - f + \frac{f}{N}}$$

- 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)

$$T_p = \alpha \cdot \frac{T_1}{p} + (1-\alpha) \cdot T_1$$

┌  
└  
↓  
parallelizable part/fraction  
of the single-processor  
program

┌  
└  
└  
non-parallelizable part

# Amdahl's Law

---

$$\text{Speedup}_{\text{with } p \text{ proc.}} = \frac{T_1}{T_p} = \frac{1}{\frac{\alpha}{p} + (1-\alpha)}$$

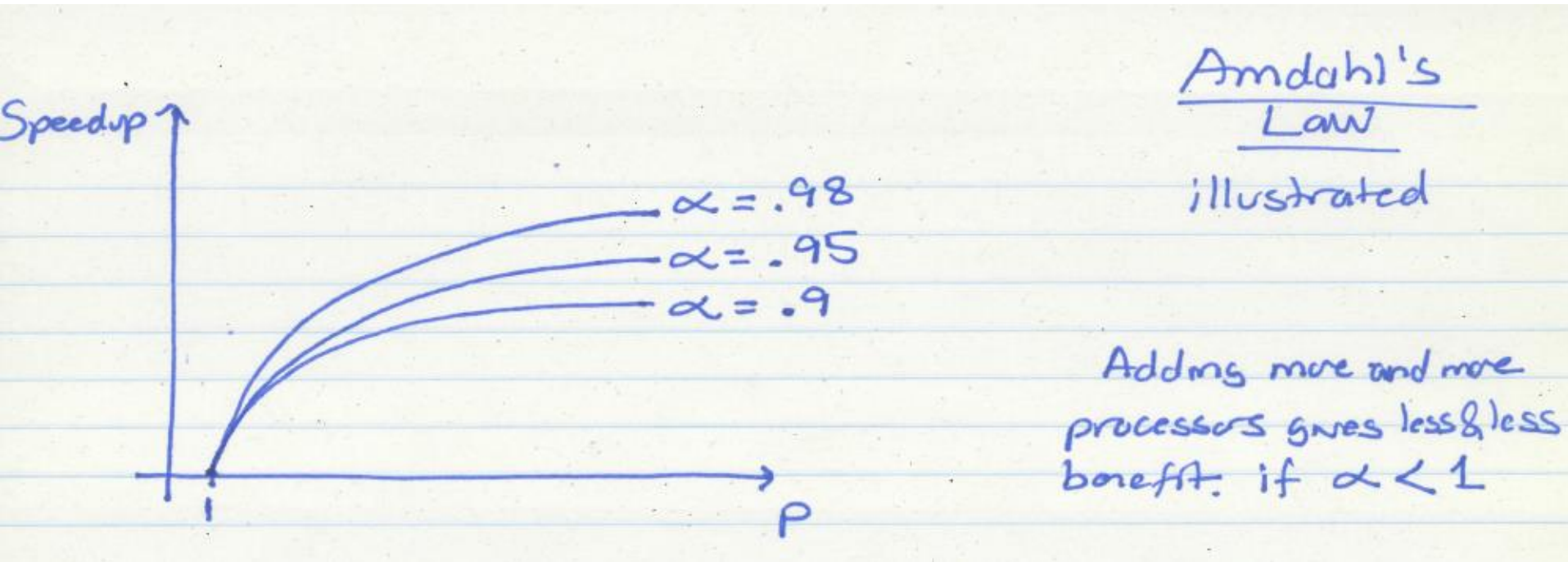
$$\text{Speedup}_{\text{as } p \rightarrow \infty} = \frac{1}{1 - \alpha}$$

$\alpha$  → bottleneck for parallel Speedup

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

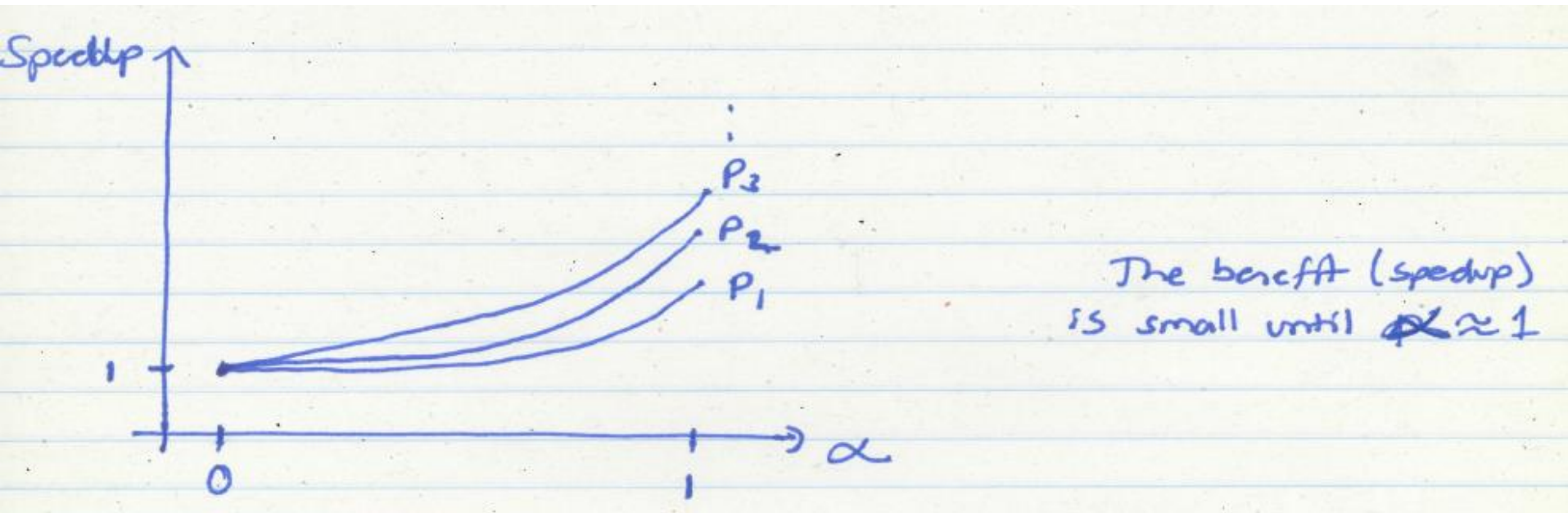
# Amdahl's Law Implication 1

---



# Amdahl's Law Implication 2

---





# Caveats of Parallelism (II)

---

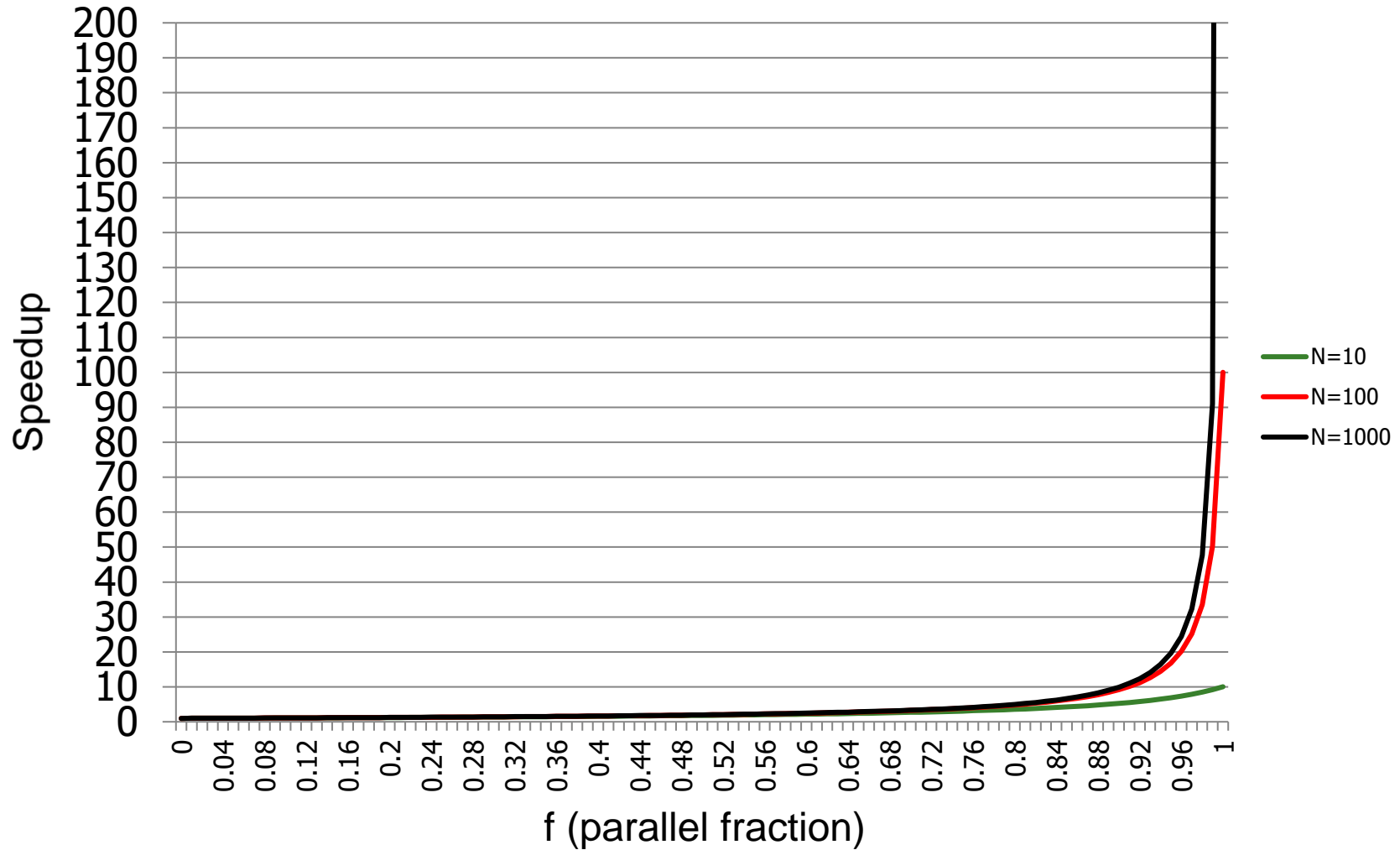
## ■ Amdahl's Law

- f: Parallelizable fraction of a program
- N: Number of processors

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

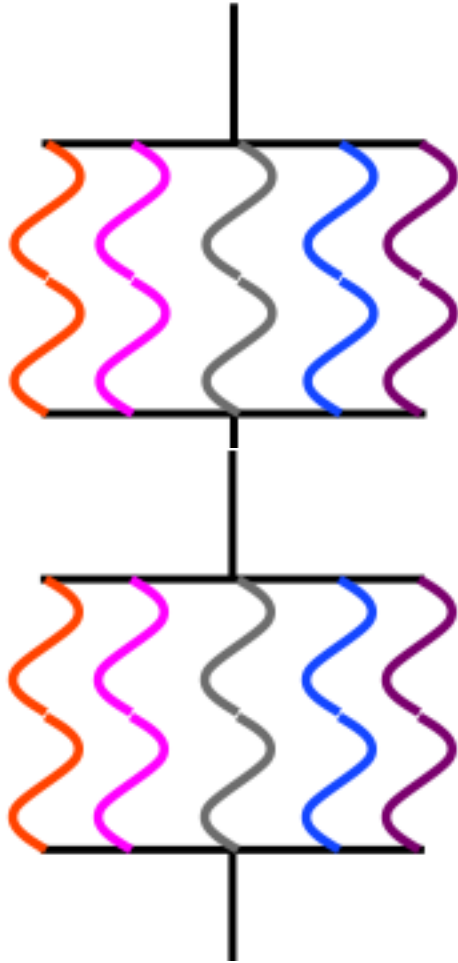
- Amdahl, “Validity of the single processor approach to achieving large scale computing capabilities,” AFIPS 1967.
- **Maximum speedup limited by serial portion: Serial bottleneck**
- **Parallel portion is usually not perfectly parallel**
  - Synchronization overhead (e.g., updates to shared data)
 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

# Sequential Bottleneck



# Why the Sequential Bottleneck?

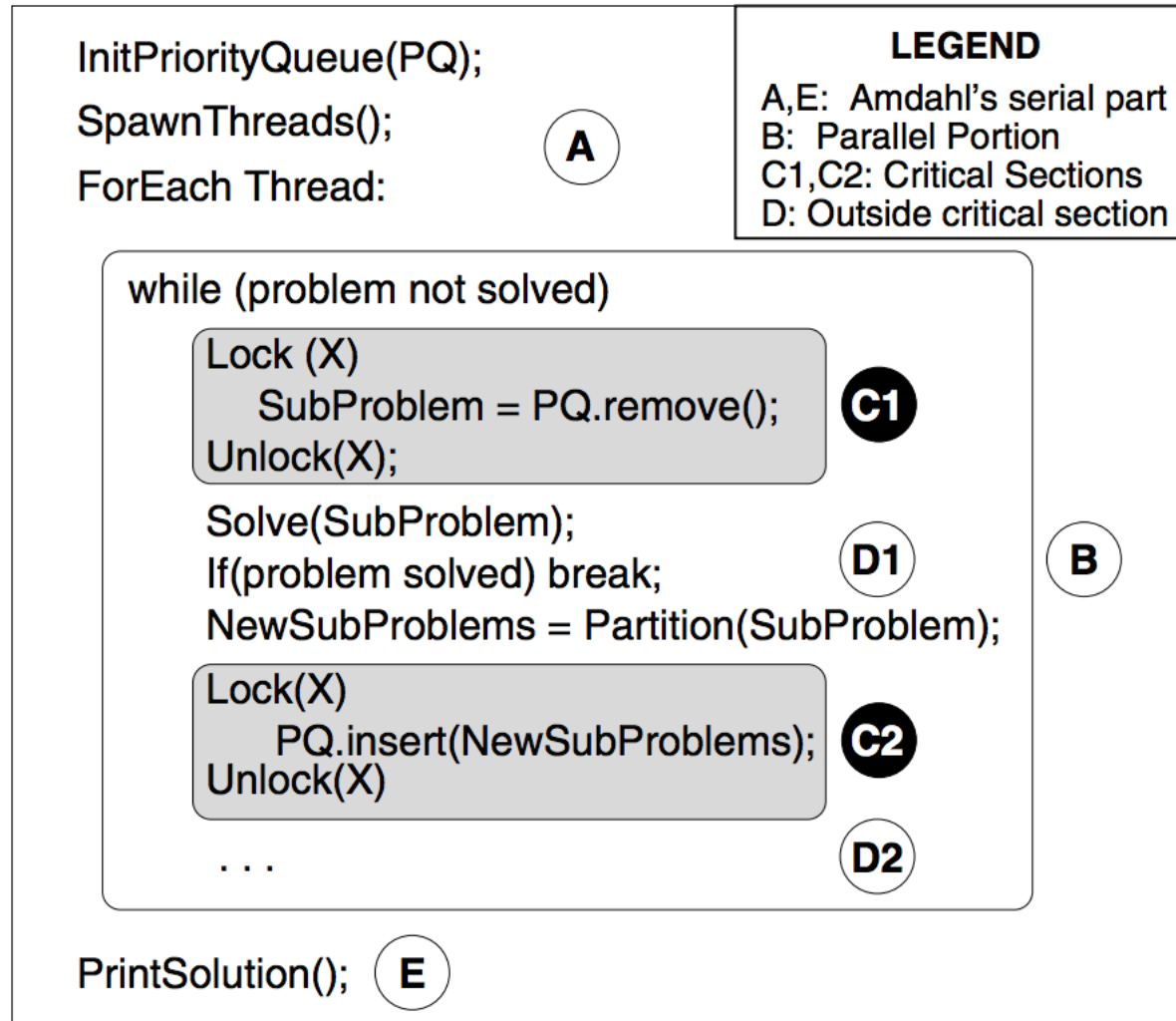
---



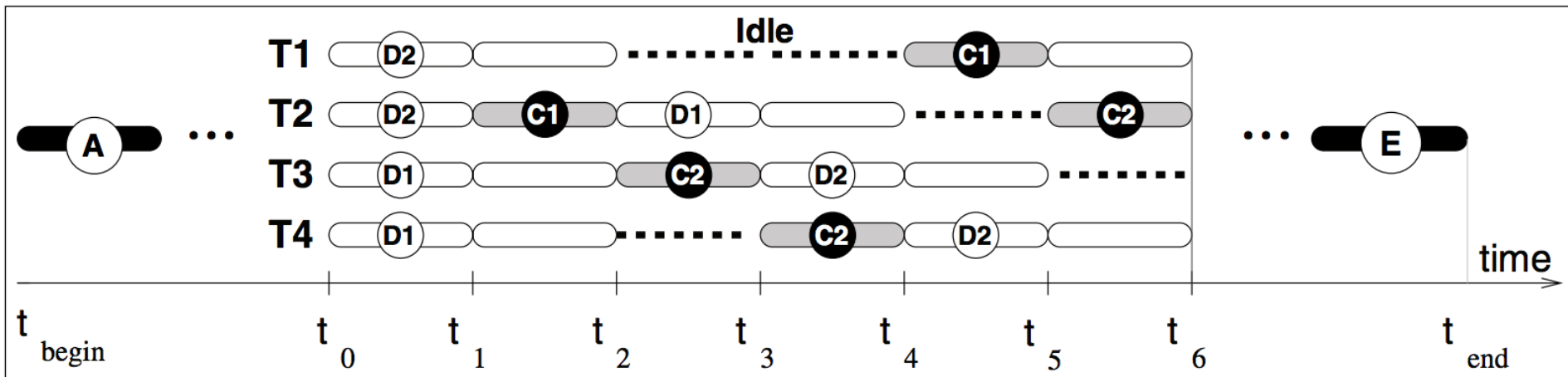
- Parallel machines have the sequential bottleneck
- Main cause: **Non-parallelizable operations on data** (e.g. non-parallelizable loops)  

```
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)



# Another Example of Sequential Bottleneck (II)



# 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

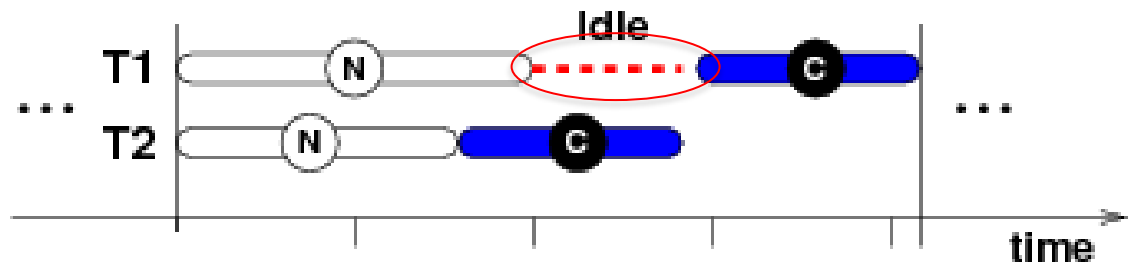
```
  lock(A)
```

```
    Update shared data
```

```
  unlock(A)
```

C

```
}
```





# Critical Section Example from MySQL

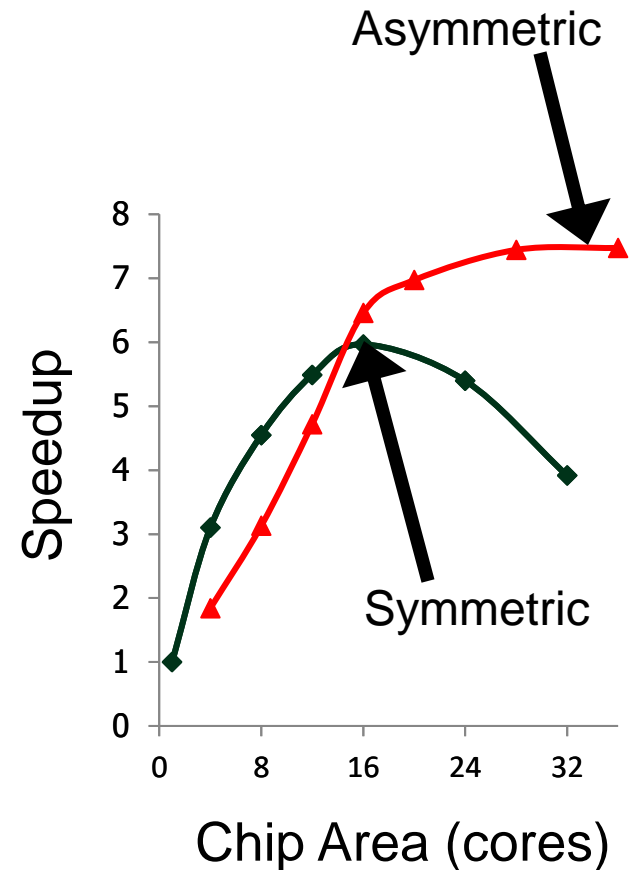
**Critical  
Section**

Access Open Tables Cache

Open database tables

Perform the operations  
....

Parallel

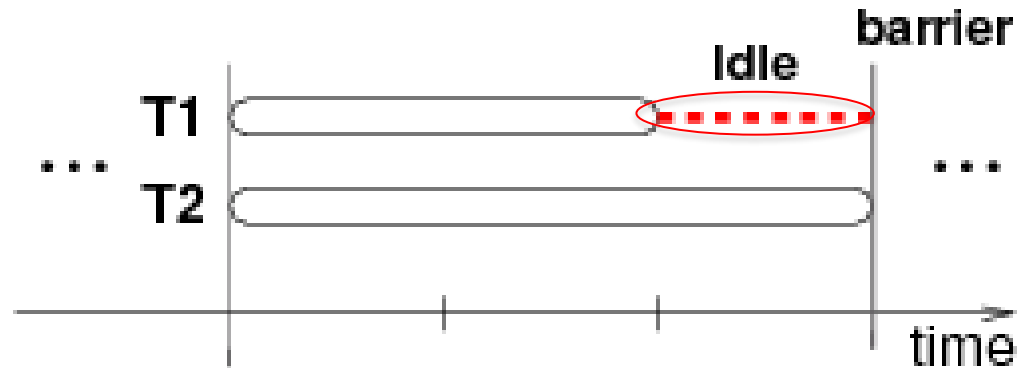


# Remember: Barriers

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

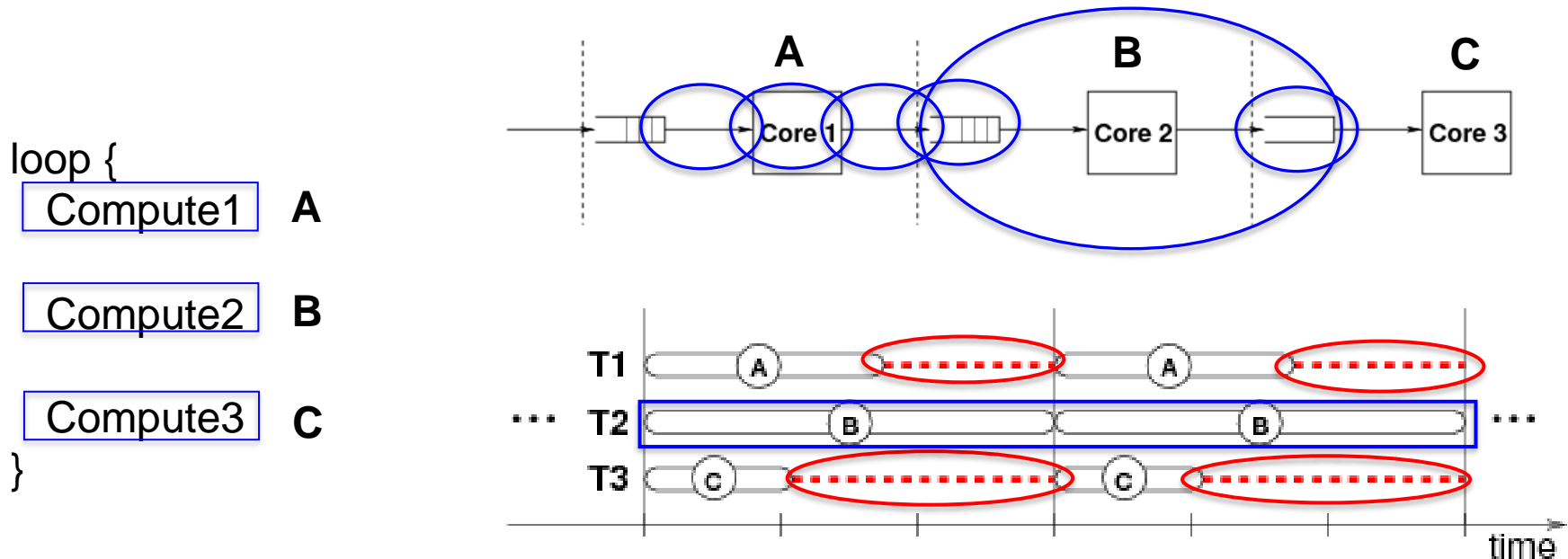
Each thread:

```
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



# 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<sup>†</sup> Rustam Miftakhutdinov<sup>†</sup> Chris Fallin<sup>§</sup>  
Chang Joo Lee<sup>‡</sup> José A. Joao<sup>†</sup> Onur Mutlu<sup>§</sup> Yale N. Patt<sup>†</sup>

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

<sup>§</sup>Carnegie Mellon University  
{cfallin,onur}@cmu.edu

<sup>‡</sup>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 <sup>†</sup> M. Aater Suleman <sup>‡†</sup> Onur Mutlu <sup>§</sup> Yale N. Patt <sup>†</sup>

<sup>†</sup> ECE Department  
The University of Texas at Austin  
Austin, TX, USA  
{joao, patt}@ece.utexas.edu

<sup>‡</sup> Flux7 Consulting  
Austin, TX, USA  
suleman@hps.utexas.edu

<sup>§</sup> 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<sup>†</sup>   Onur Mutlu<sup>§</sup>   José A. Joao<sup>†</sup>   Khubaib<sup>†</sup>   Yale N. Patt<sup>†</sup>

<sup>†</sup>The University of Texas at Austin  
{suleman, joao, khubaib, patt}@hps.utexas.edu

<sup>§</sup>Carnegie Mellon University  
onur@cmu.edu

# Lectures on Bottleneck Acceleration

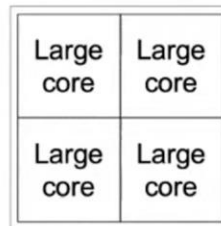
---

- Lecture 17b: Parallelism and Heterogeneity
  - Comp Arch, ETH Zurich, Fall 2021
  - [https://www.youtube.com/watch?v=GLzG\\_rEDn9A&list=PL5Q2soXY2Zi-Mnk1PxjEIG32HAGILkTOF&index=18](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=P8l3SMABYw&list=PL5Q2soXY2Zi-Mnk1PxjEIG32HAGILkTOF&index=19>

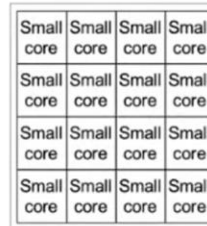
# Lecture on Parallelism & Heterogeneity

## ACMP Performance vs. Parallelism

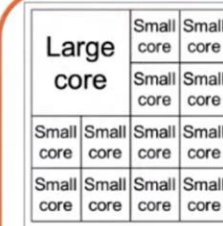
Area-budget = 16 small cores



"Tile-Large"



"Tile-Small"



ACMP

Large Cores	4	0	1
Small Cores	0	16	12
Serial Performance	2	1	2
Parallel Throughput	$2 \times 4 = 8$	$1 \times 16 = 16$	$1 \times 2 + 1 \times 12 = 14$



Livestream - Computer Architecture - ETH Zürich (Fall 2021)

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



Onur Mutlu Lectures

29.2K subscribers

Analytics

Edit video

37

Share

Download

Clip

Save

...

1,589 views Streamed live on Nov 25, 2021

Computer Architecture, ETH Zürich, Fall 2021 (<https://safari.ethz.ch/architecture/f...>)

# Lecture on Bottleneck Acceleration

**Bottleneck Acceleration**

The diagram illustrates a system architecture for bottleneck acceleration. It features three cores: Small Core 1, Small Core 2, and Large Core 0. Small Core 1 and Small Core 2 each contain an Acceleration Index Table (AIT). Large Core 0 contains a Bottleneck Table (BT) and a Scheduling Buffer (SB). A central vertical line with arrows indicates data flow between the AITs and the BT. The BT contains two entries: `bid=x4600, twc=100` and `bid=x4700, twc=10000`. The AITs also reference `bid=x4700, large core 0`. The SB is a grid structure. The video player interface shows a progress bar at 1:20:02 / 2:52:05 and a Zoom logo.

Livestream - Computer Architecture - ETH Zürich (Fall 2021)

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



Onur Mutlu Lectures

29.2K subscribers

Analytics

Edit video

42



Share

Download

Clip

Save



2,058 views Streamed live on Nov 26, 2021

Computer Architecture, ETH Zürich, Fall 2021 (<https://safari.ethz.ch/architecture/f...>)

# Computer Architecture

## Lecture 17a: Multiprocessors

Prof. Onur Mutlu

ETH Zürich

Fall 2022

24 November 2022

# 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
```

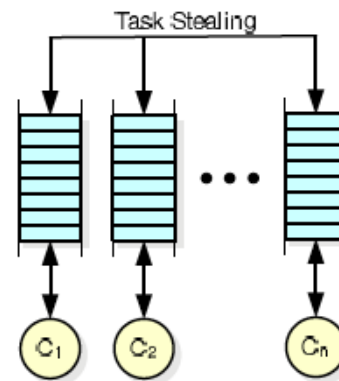
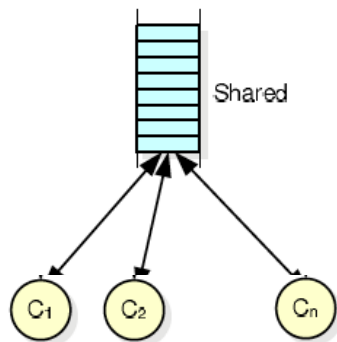
```
  Barrier
```

```
}
```

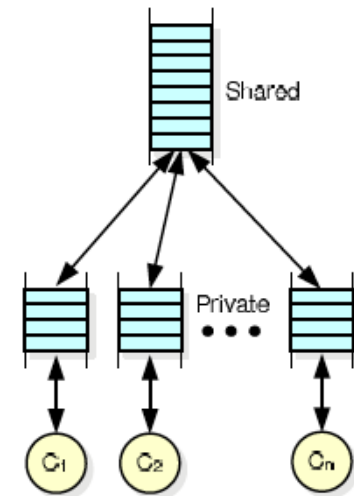
```
Return global histogram
```

# 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



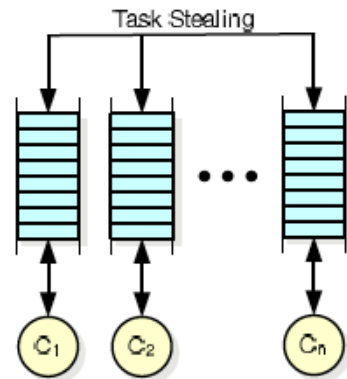
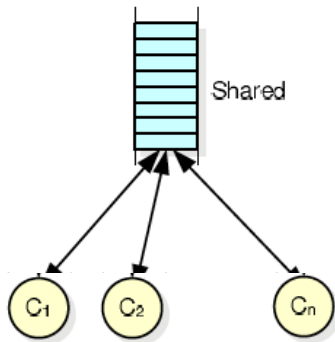
(a) Distributed Task Stealing



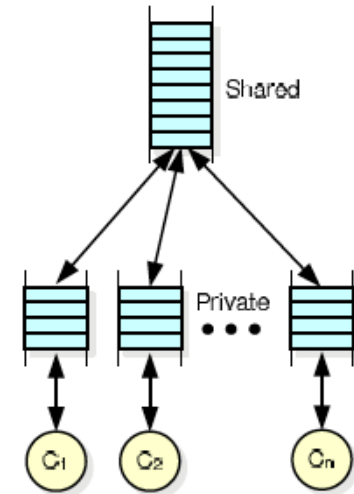
(b) Hierarchical Task Queuing

# 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?

---

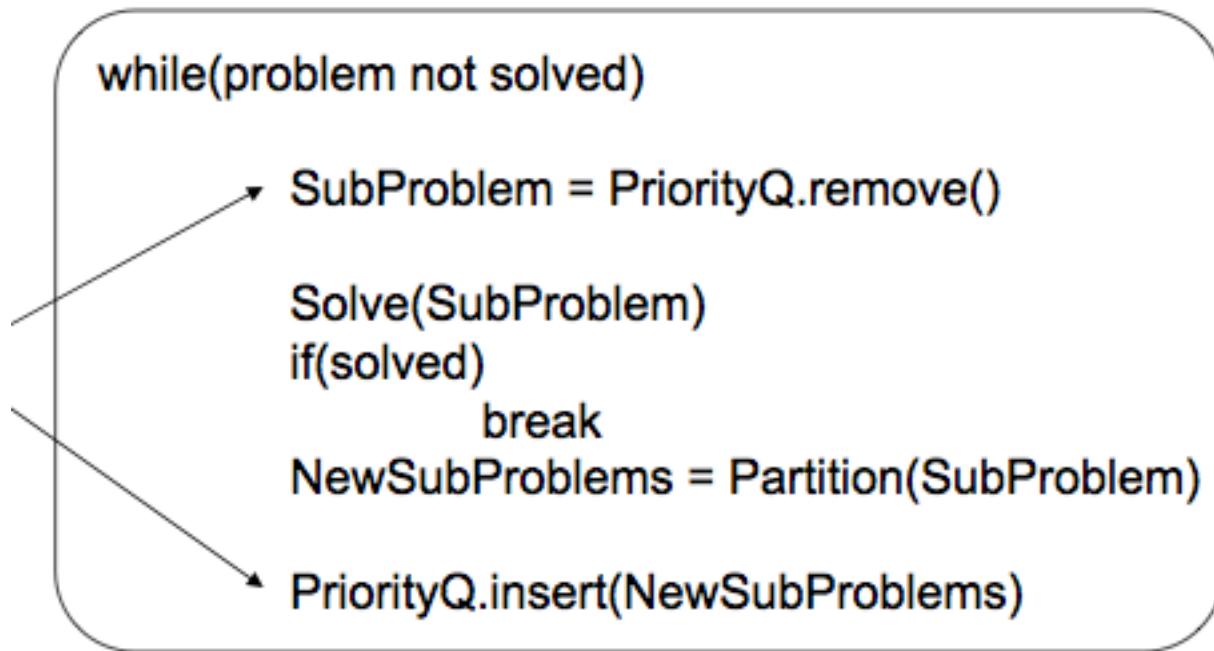
- Managing task queues in software has overhead
  - Especially high when task sizes are small
- An idea: Hardware Task Queues
  - Each processor has a dedicated task queue
  - Software fills the task queues (on demand)
  - Hardware manages movement of tasks from queue to queue
  - There can be a global 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



# 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