18-742:
Computer Architecture & Systems

The Case for a Single-Chip Multiprocessor

Prof. Phillip Gibbons

Spring 2025, Lecture 3

Parallelism Is The Key

- **Recall Amdahl's law**

$$\text{Speedup}_{\text{enhanced}}(f, S) = \frac{1}{(1-f) + \frac{f}{S}}$$

- **Parallel processing to achieve speedup**
  - More processors → more parallelism → higher speedup

- **This lecture: different forms of parallelism**

Instruction-Level Parallelism (ILP)

- Parallelism across instructions

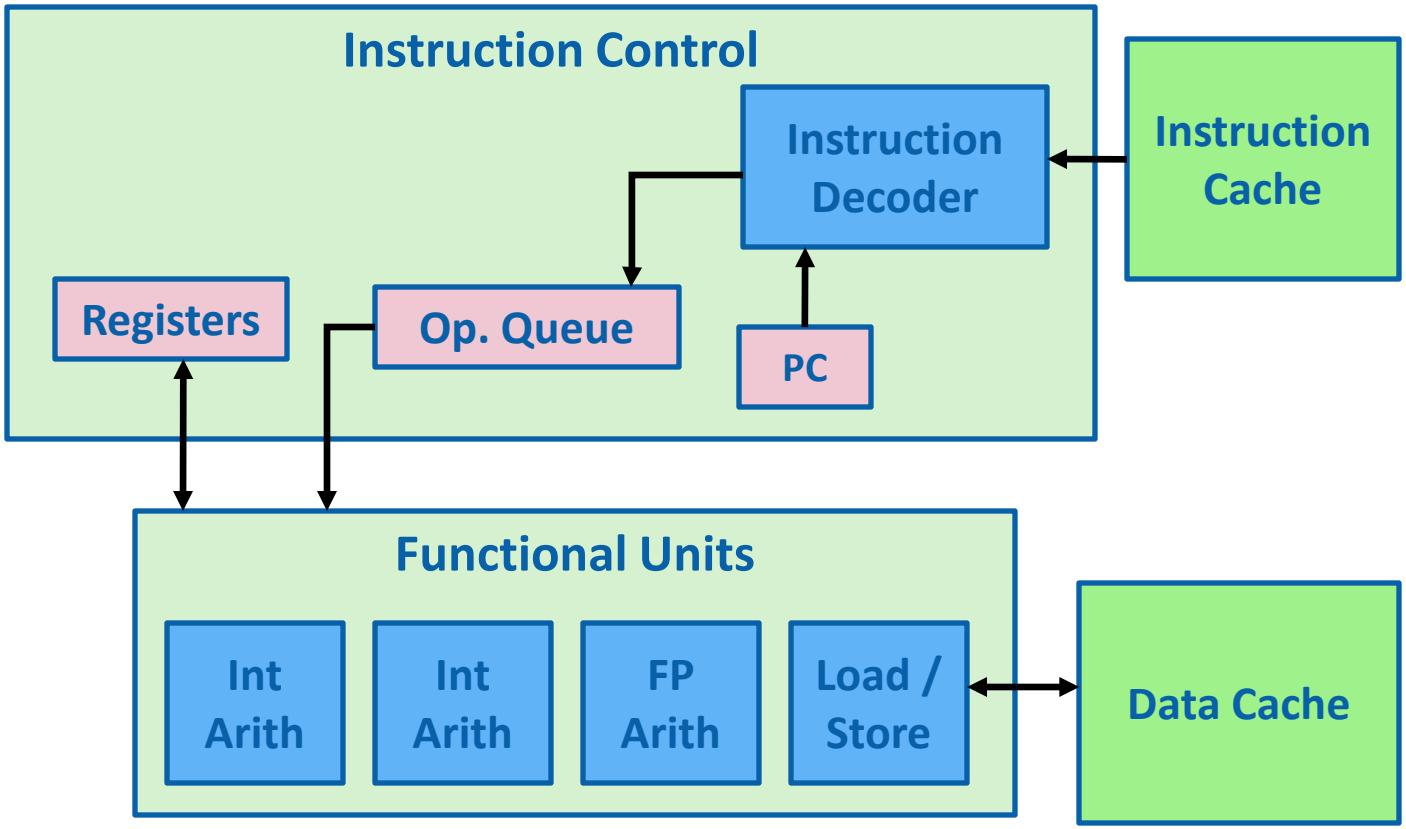
```
# Two independent instructions
x = a + b
y = c * d
```

(Assuming a, b, c, d, x, y cannot alias one another)

Instruction-Level Parallelism (ILP)

- **Parallelism across instructions**
- **How to achieve?**
  - Pipelining
  - Superscalar execution
  - Out-of-order execution
- **Examples**
  - High-performance CPUs
- **Scaling: Build “beefier” cores**

Out-of-Order Processor Structure



- Instruction Control dynamically converts program into stream of ops
- Operations mapped onto functional units to execute in parallel

ILP Limitations

- **Diminishing return from making the processor "beefier"**
  - Significant complexity
  - High port requirements in register files & caches
  - Need extremely good branch prediction

- **Not all applications expose a good level of ILP**
  - Due to inter-instruction dependencies

```
# Two dependent instructions
x = a + b
y = c * x
```

Thread-Level Parallelism (TLP)

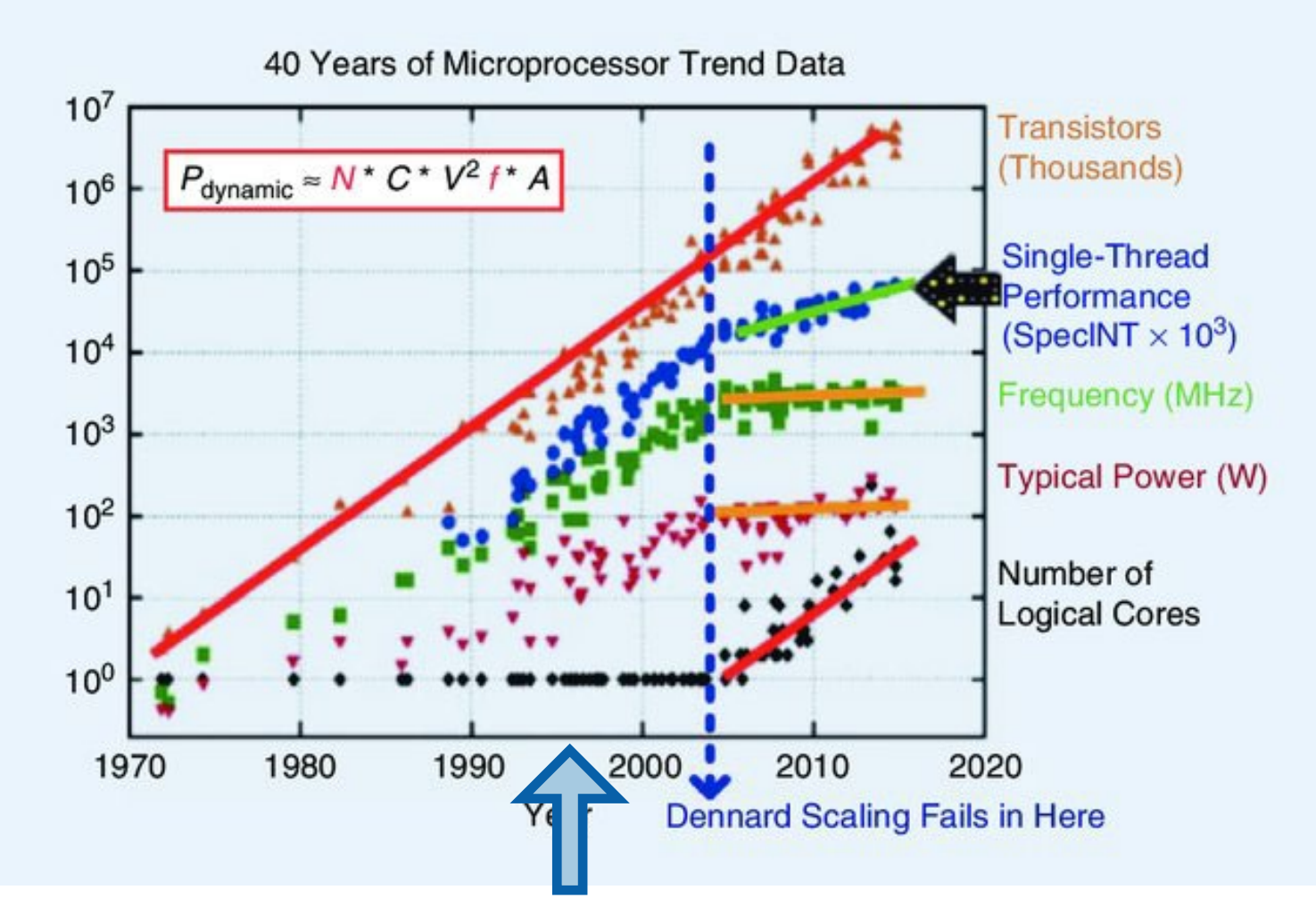
- **Parallelism across different threads**

```
def sum_array_part(arr, start, end, result, index):
    result[index] = sum(arr[start:end])

arr = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
n = len(arr)
results = [0, 0]

t1 = threading.Thread(target=sum_array_part, args=(arr, 0, n//2, results, 0))
t2 = threading.Thread(target=sum_array_part, args=(arr, n//2, n, results, 1))

total_sum = results[0] + results[1]
```

Thread-Level Parallelism (TLP)

- **Parallelism across different threads**
- **How to achieve?**
  - Multiprocessors
  - Hyperthreading
- **Examples**
  - Chip multiprocessors
  - Multi-socket systems
  - GPUs
  - FPGAs

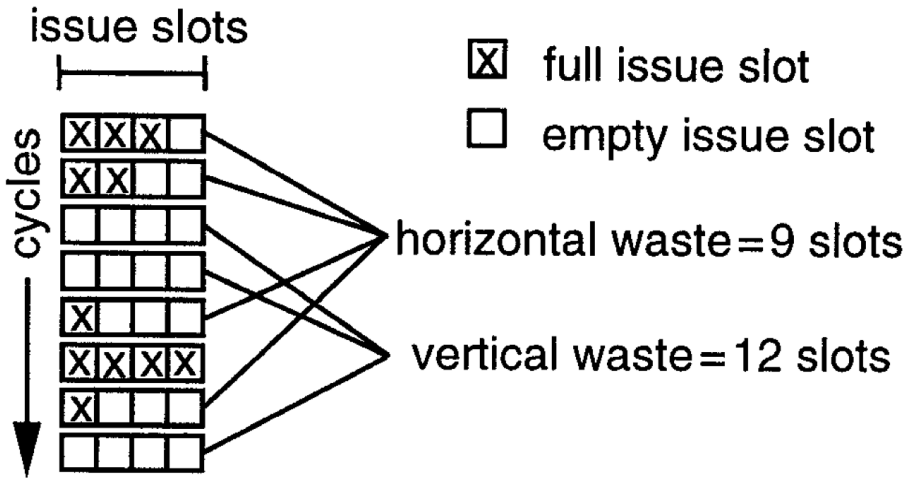
Moore's Law w/o Dennard Scaling



"Simultaneous Multithreading: Maximizing On-chip Parallelism"

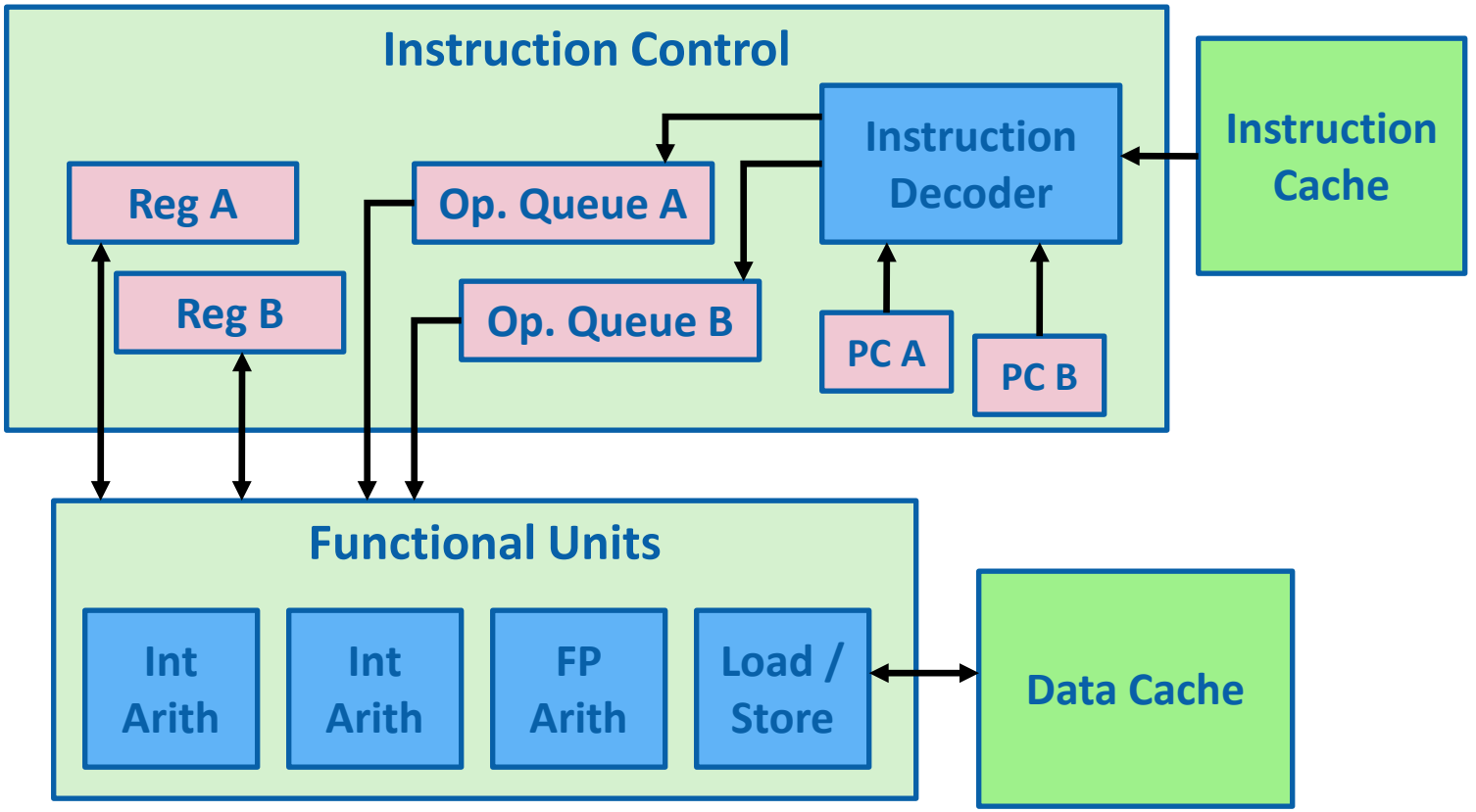
Dean Tullsen, Susan Eggers, Henry Levy 1995

- **Applications can't fully utilize beefy cores**



- **Solution: SMT (a.k.a., Hyperthreading)**
  - Run multiple threads on the same core, simultaneously

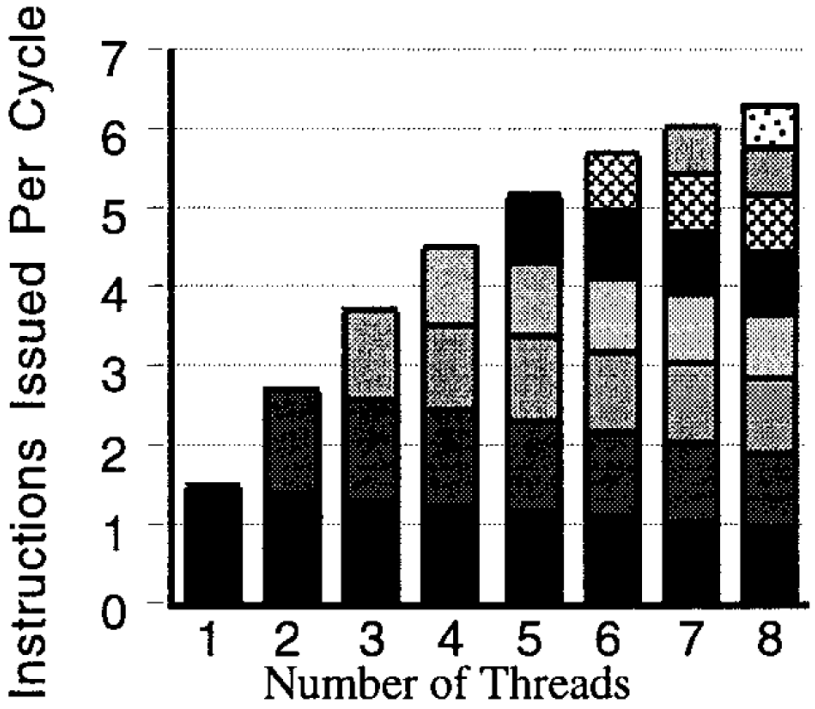
K-way Hyperthreading



- Replicate Instruction Control to process K instruction streams
- K copies of all registers
- Share functional units

How Many Hyperthreads?

- Hardware complexity vs. diminishing returns



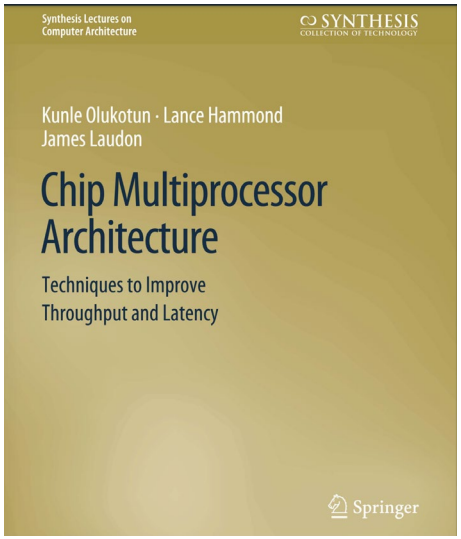
Real CPUs use 2 hyperthreads

But: Intel is discontinuing hyperthreading on its mobile chip designs

"The Case for a Single-Chip Multiprocessor"

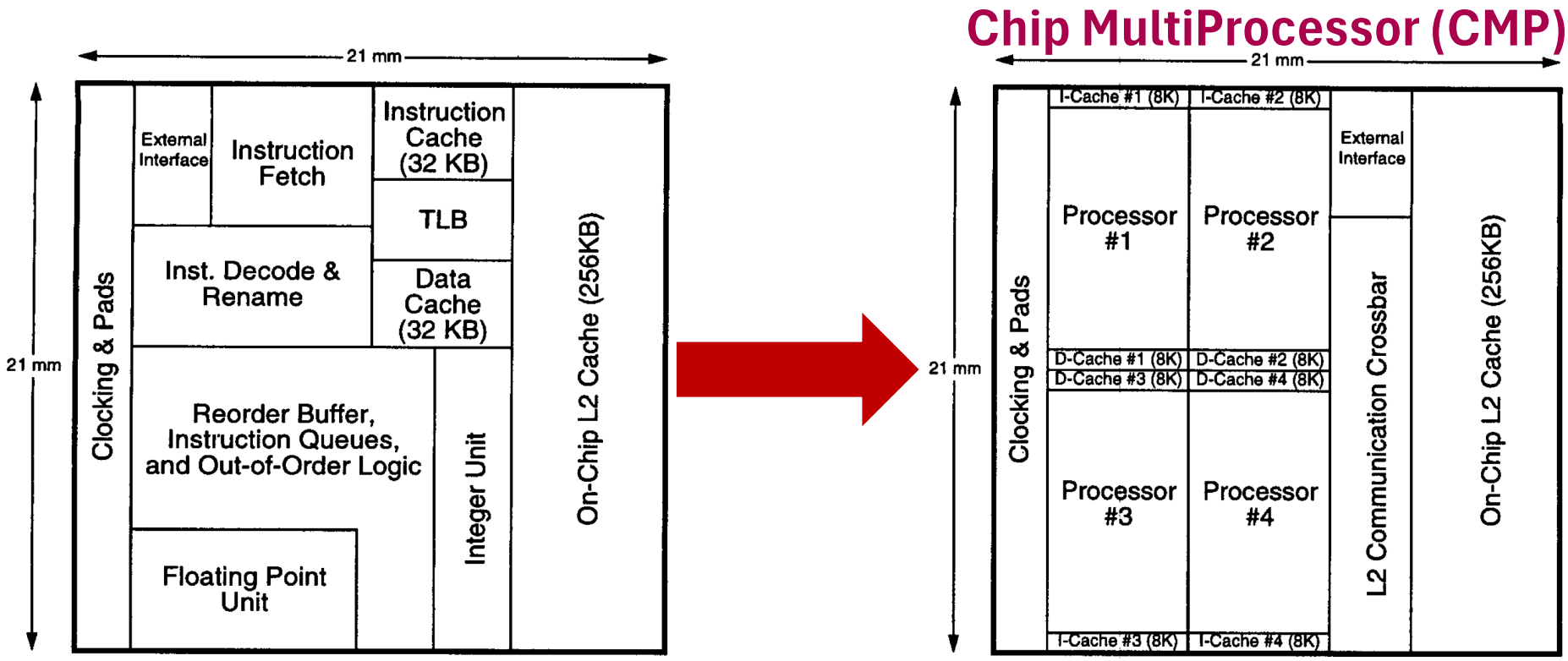
Kunle Olukotun, Basem A. Nayfeh, Lance Hammond, Kenneth M. Wilson, Kunyung Chang 1996

- **Kunle:** Stanford prof
  - ACM Fellow, IEEE Fellow, NAE
  - Eckert-Mauchly Award 2023
- **Basem:** Stanford PhD, CTO Cisco
- **Lance:** Stanford PhD, now Apple
- **Ken:** Stanford PhD, now Apple
- **Kun-Yung:** Stanford PhD, now Rambus



The Case for a Single-Chip Multiprocessor

- Focus on TLP instead of ILP
- Multiple “less beefy” processors instead of one huge processor



Application Requirements

- **Low-ILP programs (e.g., SPEC-Int) benefit little from wide-issue superscalar machines**
  - 1-wide R5000 is within 30% of 4-wide R10000

- **High-ILP programs (e.g., SPEC-FP) benefit from large windows – typically, loop-level parallelism that might be easy to extract**

The Argument

- **Build many small CPU cores**
- **The small cores are enough to optimize low-ILP programs (high thruput with multiprogramming)**
- **For high-ILP programs, the compiler parallelizes the application into multiple threads**
  - since the cores are on a single die, cost of communication is affordable
- **Low communication cost → even integer programs with moderate ILP could be parallelized**

Discussion: Summary Question #1

What Did the Paper Get Right?

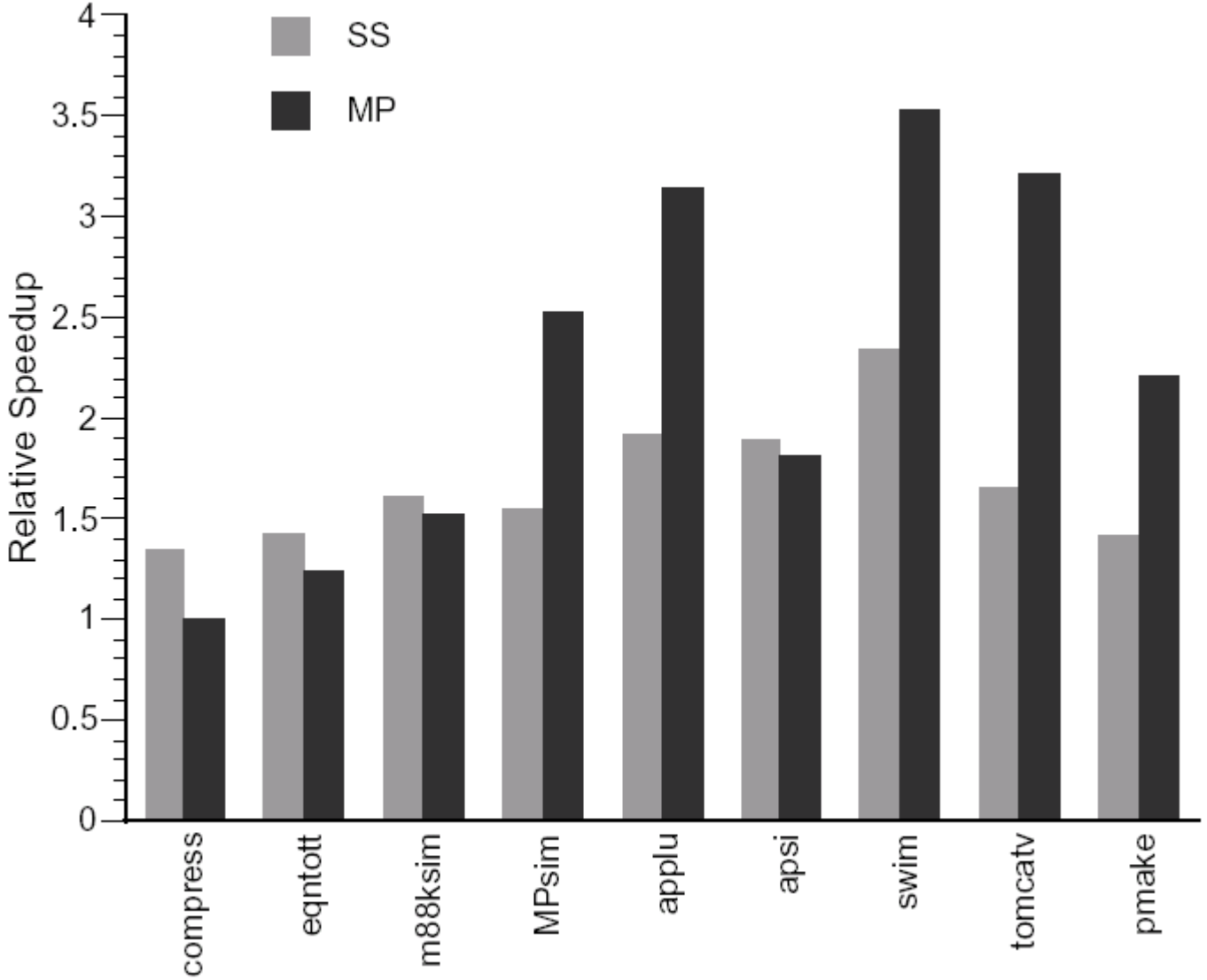
State the 3 most important things the paper says.

These could be some combination of the motivations, observations, interesting parts of the design, or clever parts of the implementation.

Processors' Parameters

	6-way SS	4x2-way MP
# of CPUs	1	4
Degree superscalar	6	4 x 2
# of architectural registers	32int / 32fp	4 x 32int / 32fp
# of physical registers	160int / 160fp	4 x 40int / 40fp
# of integer functional units	3	4 x 1
# of floating pt. functional units	3	4 x 1
# of load/store ports	8 (one per bank)	4 x 1
BTB size	2048 entries	4 x 512 entries
Return stack size	32 entries	4 x 8 entries
Instruction issue queue size	128 entries	4 x 8 entries
I cache	32 KB, 2-way S. A.	4 x 8 KB, 2-way S. A.
D cache	32 KB, 2-way S. A.	4 x 8 KB, 2-way S. A.
L1 hit time	2 cycles (4 ns)	1 cycle (2 ns)
L1 cache interleaving	8 banks	N/A
Unified L2 cache	256 KB, 2-way S. A.	256 KB, 2-way S. A.
L2 hit time / L1 penalty	4 cycles (8 ns)	5 cycles (10 ns)
Memory latency / L2 penalty	50 cycles (100 ns)	50 cycles (100 ns)

Results


SS
MP
Relative Speedup
4
3.5
3
2.5
2
1.5
1
0.5
0
compress
eqntott
m88ksim
MPsim
applu
apsi
swim
tomcatv
pmake

Similar area for 6-wide SS and 4x2 CMP

	4-wide	6-wide SS	4x2-way CMP	Comments
32KB DL1	13	17	4x3	Banking/muxing
32KB IL1	14	18	4x3.5	Banking/muxing
TLB	5	15	4x5	
Bpred	9	28	4x7	
Decode	11	38	4x5	Quadratic effect
Queues	14	50	4x4	Quadratic effect
ROB/Regs	9	34	4x2	Quadratic effect
Int FUs	10	31	4x10	More FUs in CMP
FP FUs	12	37	4x12	More FUs in CMP
Crossbar			50	Multi-L1s → L2
L2, clock, ext. interface	163	163	163	Remains unchanged

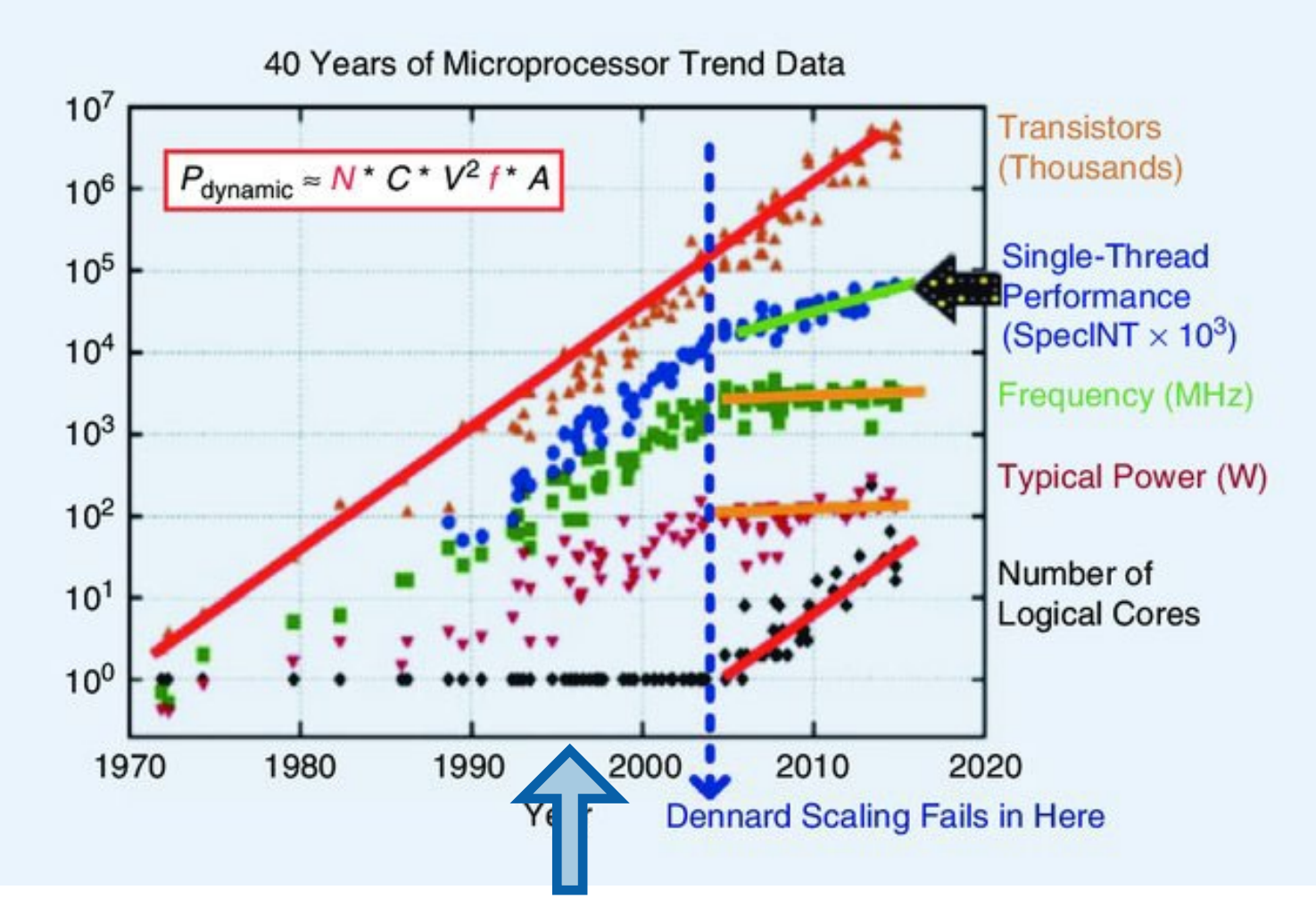
Discussion: Summary Question #2

What Did the Paper Get Wrong?

Describe the paper's single most glaring deficiency.

Every paper has some fault. Perhaps an experiment was poorly designed or the main idea had a narrow scope or applicability.

Why Did Multicore Only Emerge in 2004?



"Single-chip Heterogeneous Computing: Does the Future Include Custom Logic, FPGAs, and GPGPUs?"

Eric Chung, Peter Milder, James Hoe, Ken Mai 2010

- Chips with heterogeneous compute units

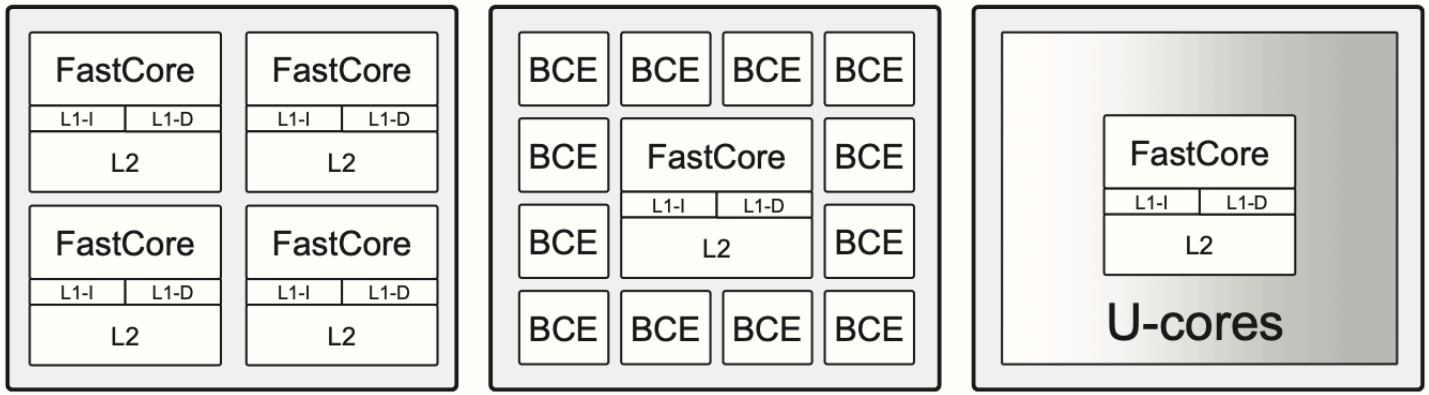


Figure 1: Chip models:
(a) Symmetric, (b) Asymmetric, (c) Heterogeneous.

- Creates/studies scaling model of Hill2008 extended to U-cores

BCE: Base-Core Equivalent U-cores: Unconventional cores

Key Findings

- >90% parallelism required for big U-core performance gains
- If application is BW-limited, flexible U-cores (FPGAs, GPGPUs) can keep up with fixed U-cores (ASICs)
- When parallelism is 90-99%, flexible U-cores are competitive with fixed U-cores
- U-cores are more broadly useful if goal is same performance with reduced energy/power

To Read for Wednesday

"Why On-chip Cache Coherence is Here To Stay"

Milo M.K. Martin, Mark D. Hill, Daniel J. Sorin 2012

Optional Further Reading:

"CoNDA: Efficient Cache Coherence Support for Near-Data Accelerators"

Amirali Boroumand, Saugata Ghose, Minesh Patel, Hasan Hassan, Brandon Lucia, Rachata Ausavarungnirun, Kevin Hsieh, Nastaran Hajinazar, Krishna T. Malladi, Hongzhong Zheng, Onur Mutlu 2019