Lecture 2
ILP, DLP and TLP in Modern Multicores

Daniel Sanchez and Joel Emer

6.888 Parallel and Heterogeneous Computer Architecture
Spring 2013

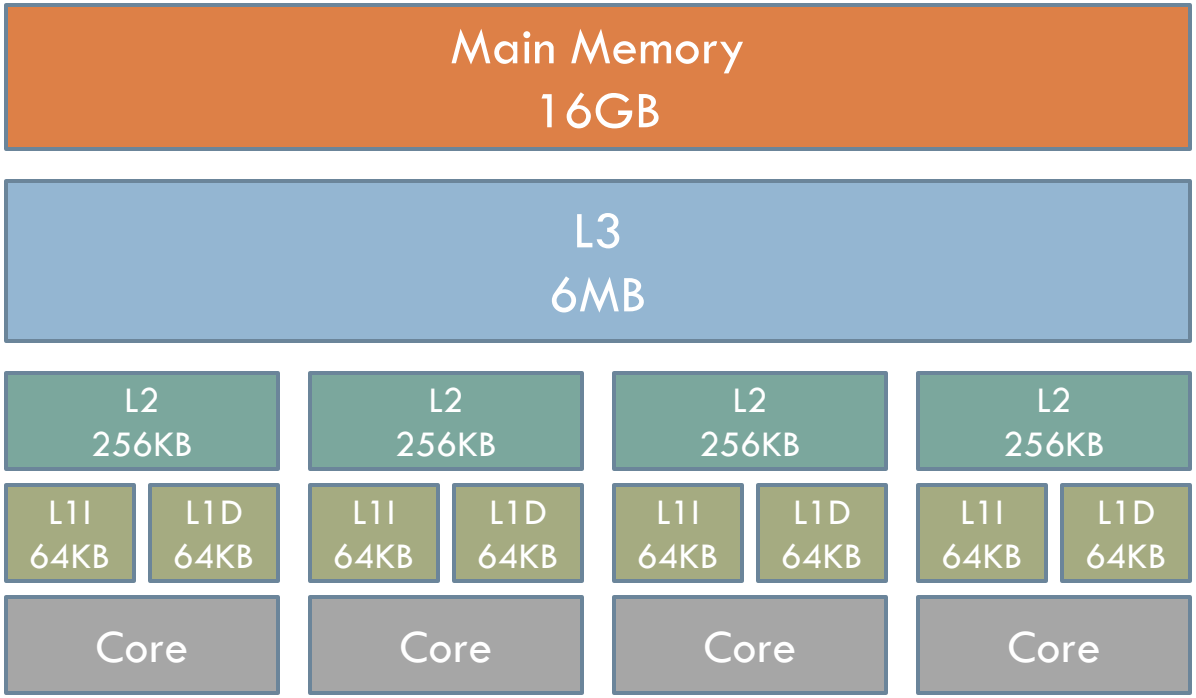
Massachusetts Institute of Technology

Review: ILP Challenges

- Clock frequency: getting close to pipelining limits
  - Clocking overheads, CPI degradation
- Branch prediction & memory latency limit the practical benefits of out-of-order execution
- Power grows superlinearly with higher clock & more OOO logic
- Design complexity grows exponentially with issue width

- Limited ILP → Must exploit TLP and DLP
  - Thead-Level Parallelism: Multithreading and multicore
  - Data-Level Parallelism: SIMD

Review: Memory Hierarchy



150 cycles, 21GB/s

26-31 cycles, 96GB/s
(24GB/s/core)

8 cycles, 48GB/s

4 cycles, 72GB/s

- Caching: Reduce latency, energy, BW of memory accesses
  - Why multilevel?
  - Why not just on-chip memories?
  - How does parallelism impact latency/BW constraints?
- Prefetching: Trade-off latency for bandwidth, energy, capacity (pollution)

Flynn's Taxonomy

	Single instruction	Multiple instruction
Single data	SISD	MISD (?)
Multiple data	SIMD	MIMD

SIMD Processing

- Same instruction sequence applies to multiple elements
  - Vector processing → Amortize instruction costs (fetch, decode, …) across multiple operations
  - Requires regular data parallelism (no or minimal divergence)

- Exploiting SIMD:
  - Explicit & low-level, using vector intrinsics
  - Explicit & high-level, convey parallel semantics (e.g., foreach)
  - Implicitly: Parallelizing compiler infers loop dependencies
    - How easy is this in C++? Java?

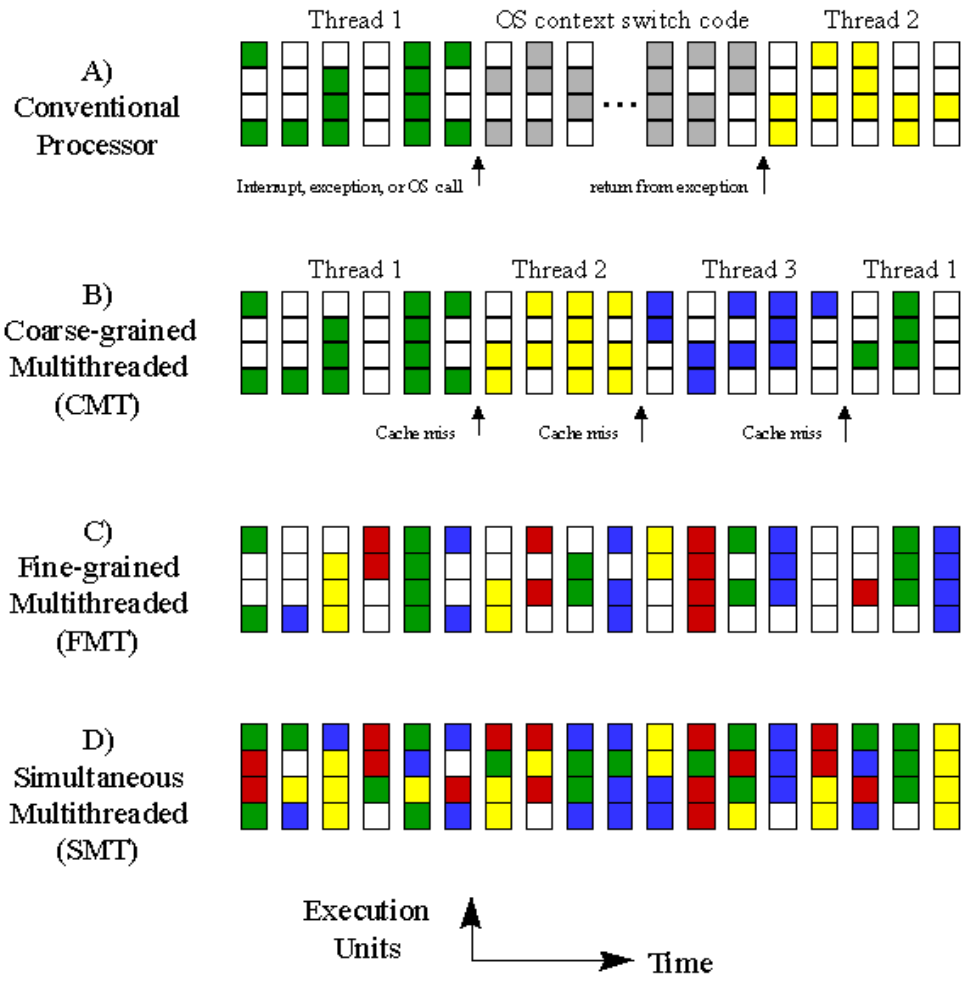
SIMD Implementations

- Modern CPUs: SIMD extensions & wider regs
  - SSE: 128-bit operands (4x32-bit or 2x64-bit)
  - AVX (2011): 256-bit operands (8x32-bit or 4x64-bit)
  - LRB (upcoming): 512-bit operands
  - Explicit SIMD: Parallelization performed at compile time

- GPUs: Architected for SIMD from the ground up
  - 32 to 64 32-bit floats
  - Implicit SIMD: Scalar binary, multiple instances always run in lockstep
    - How to handle divergence?

Multithreading: Options

- Motivation: Hardware underutilized on stalls → TLP to increase utilization
- CGMT, SMT typically increase throughput with moderate cost, maintain single-thread performance
- FGMT typically trades throughput and simplicity at the expense of single-thread performance



Example 1: SMT (Nehalem)

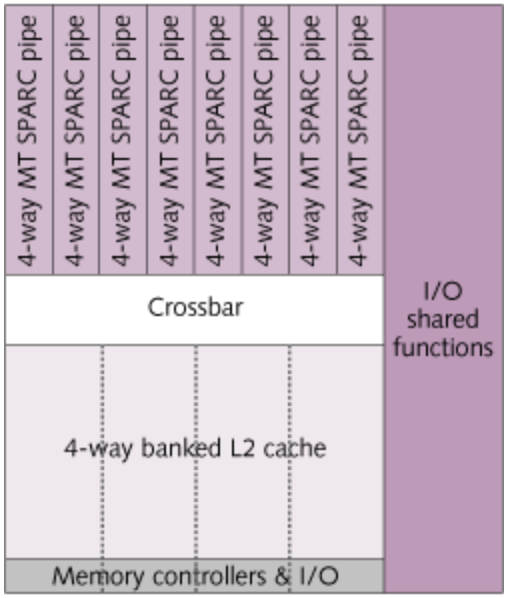
- SMT design choices: For each component,
  - Replicate, partition statically, or share
  - Tradeoffs? Complexity, utilization, interference & fairness

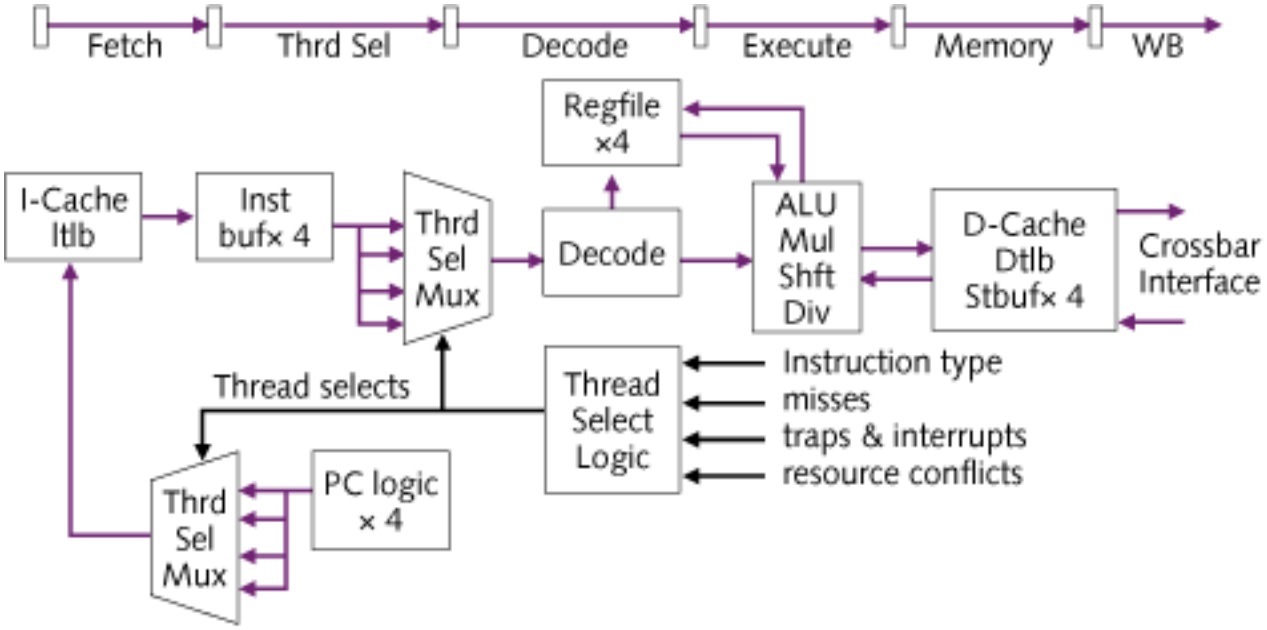
- Example: Intel Nehalem
  - 4-wide superscalar, 2-way SMT
  - Replicated: Register file, RAS predictor, large-page ITLB
  - Partitioned: Load buffer, store buffer, ROB, small-page ITLB
  - Shared: Instruction window, execution units, predictors, caches, DTLBs

- SMT policies:
  - Fetch policies: Utilization vs fairness
  - Long-latency stall tolerance: Flushing vs stalling

[See: "Exploiting Choice: Instruction Fetch and Issue on an Implementable Simultaneous Multithreading Processor", Tullsen et al, ISCA 96]

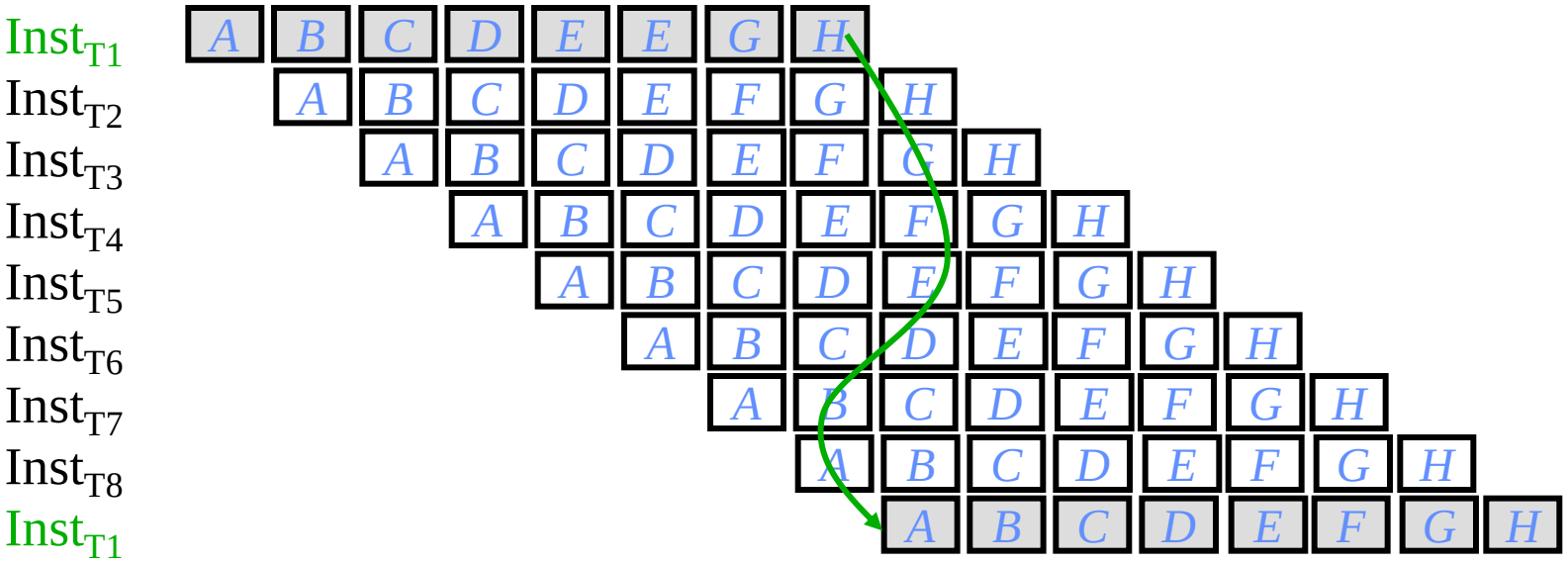
Example 2: FGMT (Niagara)





- 4 threads/core, round-robin scheduling
  - No branch prediction, minimal bypasses → more stalls
  - Small L1 caches (can tolerate higher L1 miss rates)
    - But L2 is still large… performance with long-latency stalls?

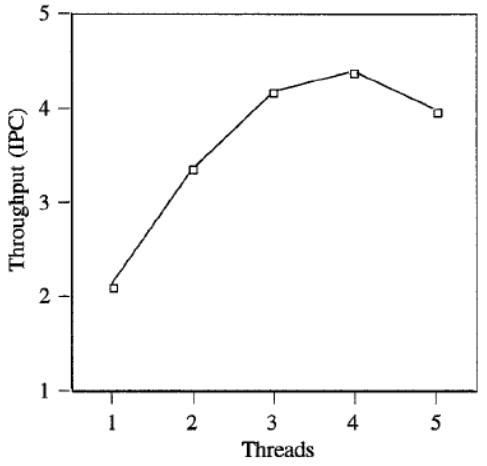
Example 3: Extreme FGMT (Tera MTA)



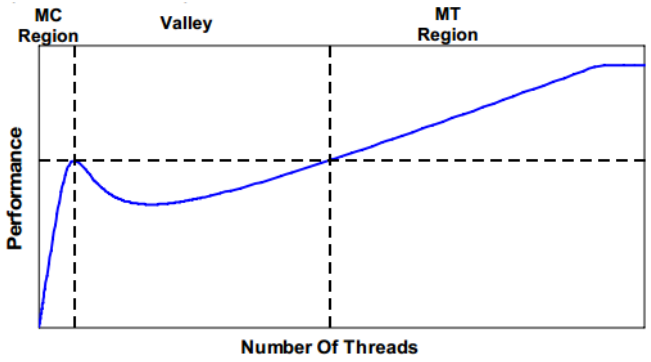
- Use FGMT to hide all instruction latencies
  - Worst case instruction latency is 128 cycles → 128 threads
- Benefits: no interlocks, no bypass, and no cache
- Problem: single-thread performance
- GPUs also exploit high FGMT for latency tolerance (e.g., Fermi, 48-way MT)
  - Throughput-oriented functional units: Longer latency, deeply pipelined
  - Throughput-oriented memory system: Small caches, aggressive memory scheduler

Multithreading: How Many Threads?

- With more HW threads:
  - Larger/multiple register files
  - Replicated & partitioned resources → Lower utilization, lower single-thread performance
  - Shared resources → Utilization vs interference and thrashing



- Impact of MT/MC on memory hierarchy?



["Many-Core vs. Many-Thread Machines: Stay Away From the Valley", Guz et al, CAL 09]

Amdahl's Law

- Amdahl's Law: If a change improves a fraction *f* of the workload by a factor *K*, the total speedup is:

$$\text{Speedup} = \frac{\text{Time}_{\text{before}}}{\text{Time}_{\text{after}}} = \frac{1}{f / K + (1 - f)}$$

- Not only valid for performance!
  - Energy, complexity, …
- I/D/TLP techniques make different tradeoffs between *K* and *f*
  - SIMD vs MIMD f and K?

Amdahls' Law in the Multicore Era [Hill & Marty, CACM 08]

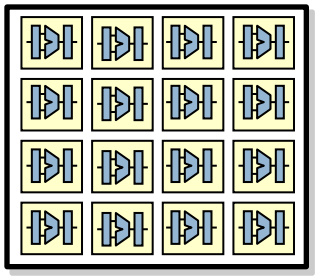
- Should we focus on a single approach to extract parallelism?
- At what point should we trade ILP for TLP?

- Assume a resource-limited multi-core
  - N base core equivalent (BCEs) due to area or power constraints
  - A 1-BCE core leads to performance of 1
  - A R-BCE core leads to performance of perf(R)
    - Assuming perf(R) = sqrt(R) in following drawings (Pollack's rule)

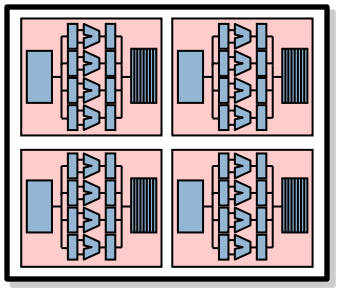
- How should we design the multi-core?
  - Select type & number of cores
  - Assume caches & interconnect are rather constant
  - Assume no application scaling (or equal scaling for seq/par portions)

Three Multicore Approaches

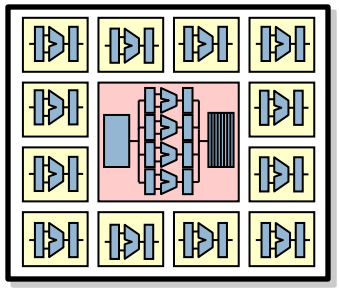
	Large Cores (R BCEs/core)		Simple Cores (1 BCE/core)	
	Number	**Performance**	**Number**	**Performance**
Symmetric CMP	N/R	Seq: Perf(R) Par: N/R*Perf(R)	-	-
Asymmetric CMP	1	Seq: Perf(R) Par: Perf(R)	N-R	Seq: - Par: N-R
Dynamic CMP	1	Seq: Perf(R) Par: -	N	Seq: - Par: N



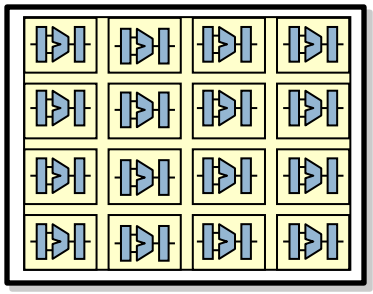
16 1-BCE cores



Symmetric:
4 4-BCE cores



Asymmetric:
1 4-BCE core
& 12 1-BCE cores



Dynamic:
Adapt between
16 1-BCEs and 1 16-BCE

Amdahl's Law x3

- Symmetric CMP

$$\text{Symmetric Speedup} = \frac{1}{\frac{1 - F}{\text{Perf}(R)} + \frac{F * R}{\text{Perf}(R) * N}}$$

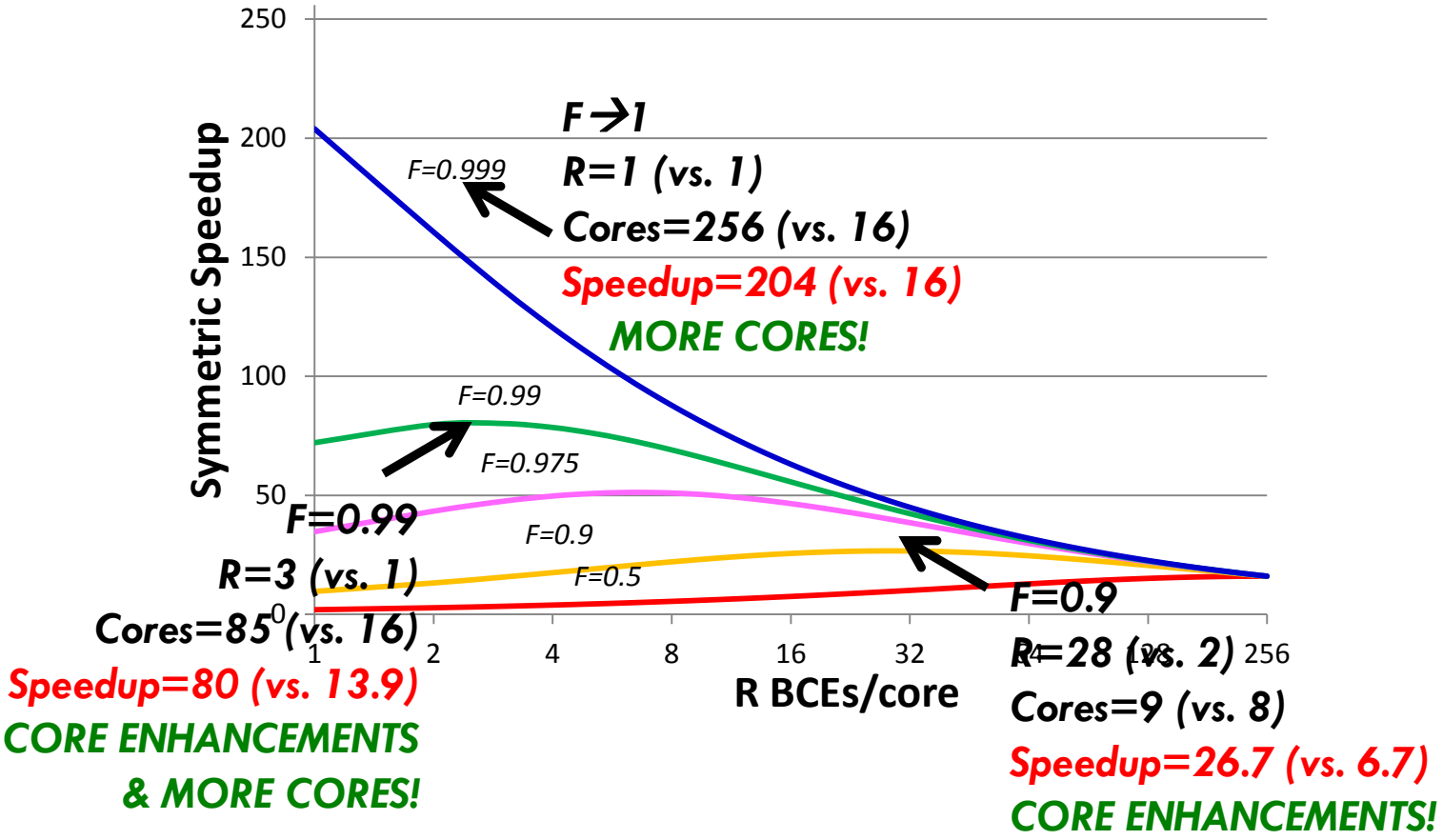
- Asymmetric CMP

$$\text{Asymmetric Speedup} = \frac{1}{\frac{1 - F}{\text{Perf}(R)} + \frac{F}{\text{Perf}(R) + N - R}}$$

- Dynamic CMP

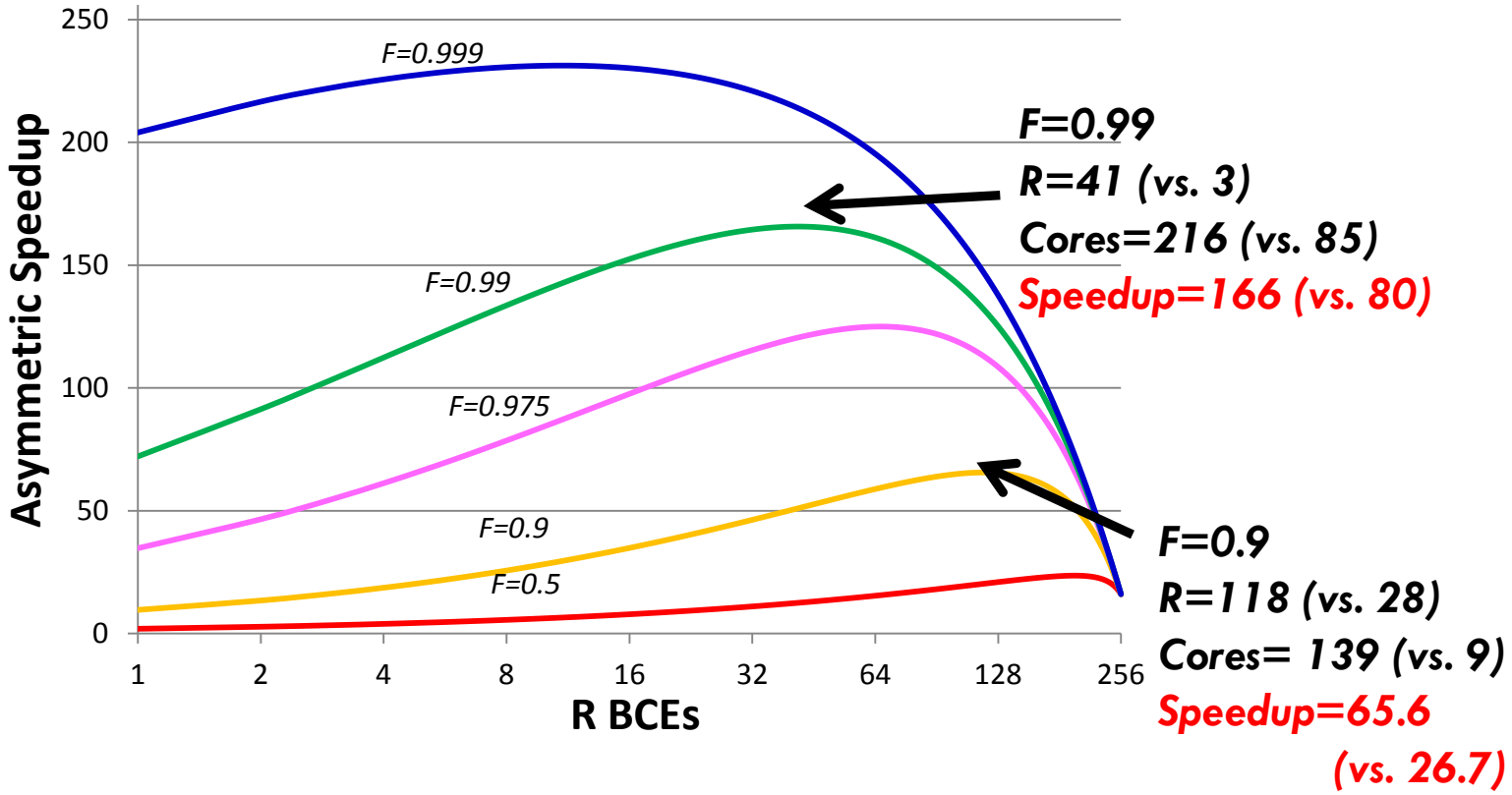
$$\text{Dynamic Speedup} = \frac{1}{\frac{1 - F}{\text{Perf}(R)} + \frac{F}{N}}$$

Symmetric Multicore Chip
N = 256 BCEs



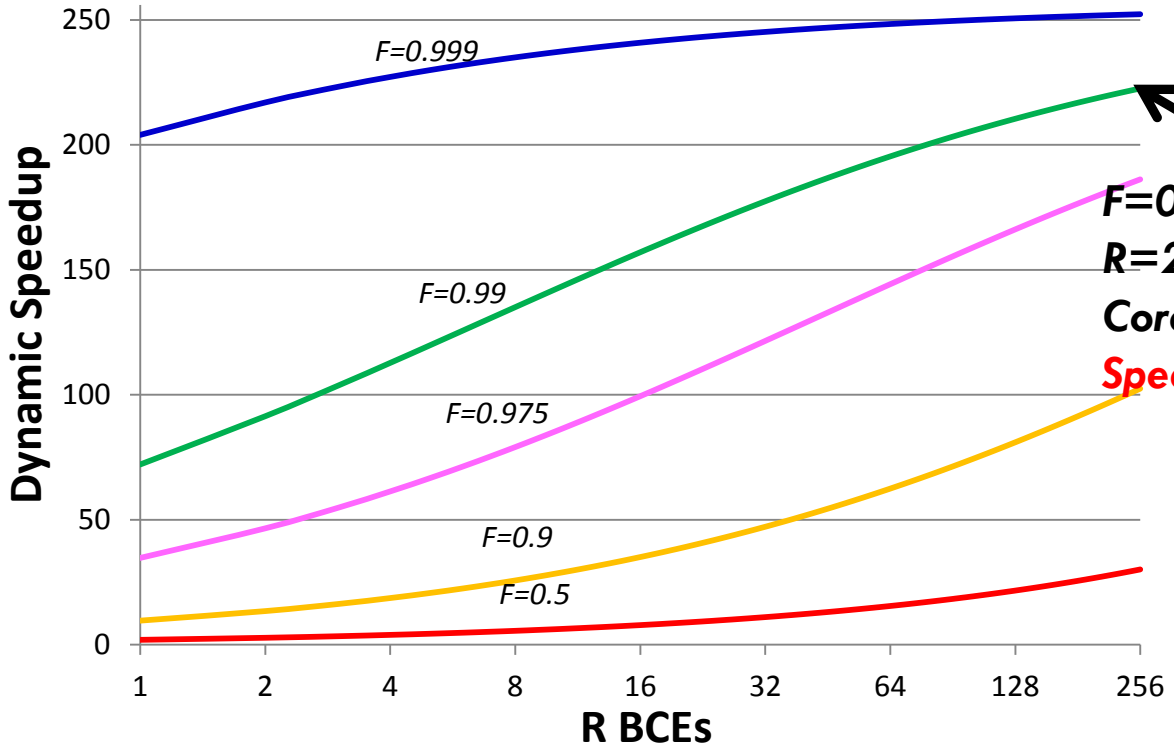
- Results in parentheses → N= 16
- Higher N → Higher R (more ILP) for fixed f

Asymmetric Multicore Chip
N = 256 BCEs



- Results in parentheses → N= 16
- Better speedups than symmetric
  - Software complexity?

Dynamic Multicore Chip N = 256 BCEs



- Results in parenthesis refer to N=16
- Dynamic offers even higher speedups than asymmetric
  - SW and HW complexity?

Readings for Wednesday

1. Is Dark Silicon Useful?
2. Dark Silicon and the End of Multicore Scaling
3. Single-Chip Heterogeneous Computing