18-640 Foundations of Computer Architecture

Lecture 13: "Multicore Processor Architecture"

John Paul Shen
October 21, 2014

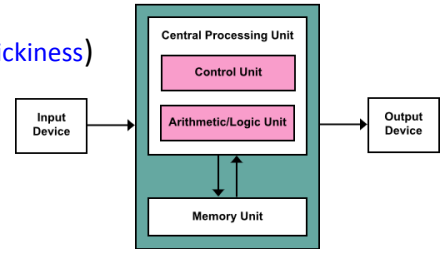
- Required Reading Assignment:
  - "Chip Multiprocessor Architecture: Techniques to Improve Throughput and Latency" by Kunle Olukotun, Lance Hammond, and James Laudon in Synthesis Lectures on Computer Architecture, Morgan & Claypool, 2007.

Electrical & Computer ENGINEERING

18-640 Course Coverage: Processor Designs

Persistence of Von Neumann Model (Legacy SW Stickiness)

1. One CPU
2. Monolithic Memory
3. Sequential Execution Semantics



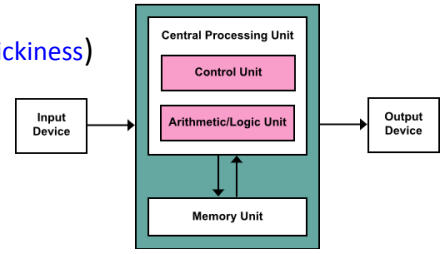
Evolution of Von Neumann Implementations:

- PP: Pipelined Processors (overlapped execution of in-order instructions)
- SSP: Superscalar Processors (out-of-order execution of multiple instructions
- MTP: Multithreaded Processors (concurrent execution of multiple threads)
- MCP: Multicore Processors == CMP: Chip Multiprocessors (concurrent multi-threads)
- SMP: Symmetric Multiprocessors (concurrent multi-threads and multi-programs)

18-640 Course Coverage: Parallelism Exploited

Persistence of Von Neumann Model (Legacy SW Stickiness)

1. One CPU
2. Monolithic Memory
3. Sequential Execution Semantics



Parallelisms for **Performance** (→ for **Power** Reduction → for **Energy** Efficiency)

➢ **ILP**:	Basic Block	(exploit ILP in PP, SSP)
➢ **ILP**:	Loop Iteration	(exploit ILP in SSP, VLIW)
➢ **TLP**:	Task/Thread	(exploit TLP in MTP, MCP)
➢ **DLP**:	Data Set	(exploit DLP in VP/SIMD, GPU)
➢ **PLP**:	Process/Program	(exploit PLP in MCP/CMP-single chip, SMP-multi chips)

Processor Performance

$$\text{Processor Performance} = \frac{\text{Time}}{\text{Program}}$$

$$= \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Cycles}}{\text{Instruction}} \times \frac{\text{Time}}{\text{Cycle}}$$

(path length) (CPI) (cycle time)

- In the 1980's (decade of pipelining):
  - CPI: 5.0 => 1.15 (also reduces cycle time)
- In the 1990's (decade of superscalar):
  - CPI: 1.15 => 0.5 (best case) (exploits ILP)
- In the 2000's (decade of multicore):
  - Core CPI unchanged; chip CPI scales with #cores (exploits TLP on chip)

18-640 Foundations of Computer Architecture

Lecture 13:
"Multicore Processor Architecture"

A. Case for Multicore Processors
B. Throughput Performance
C. Design Space Exploration
D. Niagara Case Study

Electrical & Computer ENGINEERING

Four Foundational "Laws" of Computer Architecture

- **Application Demand** (PROGRAM)
  - **Amdahl's Law** (1967) ➔ Importance of Latency Performance
    - Speedup through parallelism is limited by the sequential bottleneck
  - **Gustafson's Law** (1988) ➔ Importance of Throughput Performance
    - With unlimited data set size, parallelism speedup can be unlimited
- **Technology Supply** (MACHINE)
  - **Moore's Law** (1965) ➔ Exploitation of Chip-Level Integration
    - (Transistors/Die) increases by 2x every 18 months
  - **Bell's Law** (1971) ➔ Exploitation of System-Level Cost Saving
    - (Cost/Computer) decreases by 2x every 36 months

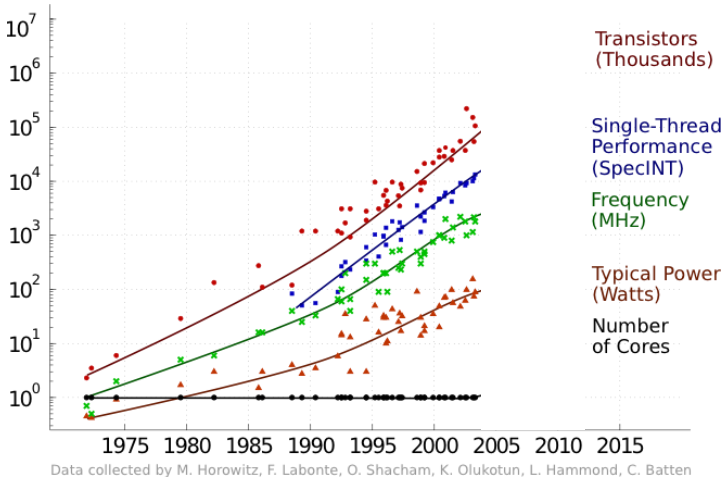
Historical Trends Leading to Chip Multiprocessors

- **Application Demand** (PROGRAM)
  - **Large Mainframe Computers**
    - Time-shared multi-programming for scientific applications
  - **Large Multiprocessor Servers** (Symmetric Multiprocessors, SMP)
    - Multi-tasking transaction processing for business applications
- **Technology Supply** (MACHINE)
  - **Continuation of Moore's Law**
    - Increasing degrees of integration and granularities of parallelism
  - **Continuation of Bell's Law**
    - Migration of high-end system designs to high-volume chip designs

A. The Case for Multicore Processors

- Stalled Scaling of Single-Core Performance
- Expected continuation of Moore's Law
- Throughput Performance for Server Workloads

Processor Scaling Until ~2004



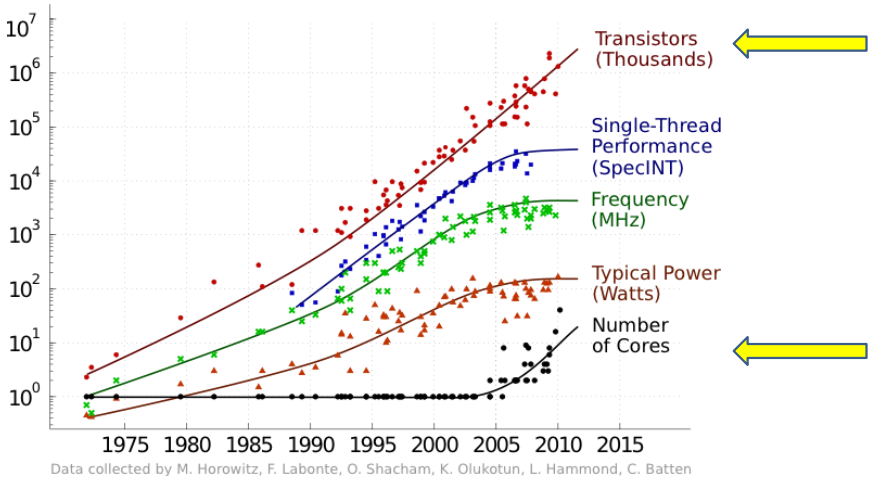
Processor Development Until ~2004

- Moore's Law: transistor count doubles every 18 months
  - Used to improve processor performance by 2x every 18 months
  - Single core, binary compatible to previous generations
- Contributors to performance improvements
  - More ILP through OOO techniques
    - Wider issue, better prediction, better scheduling, …
    - Better memory hierarchies
  - Clock frequency improvements

Problems with Single Core Performance

- Moore's Law is still doing well (for now…)
- The Power Wall
  - Power ≈ C * Vdd^2 * Freq
  - Cannot scale transistor count and frequency without reducing Vdd
  - Unfortunately, voltage scaling has essentially stalled
- The Complexity Wall
  - Designing and verifying increasingly large OOO cores is very expensive
    - 100s of engineers for 3-5 years
  - Caches are easier to design but can only help that much…

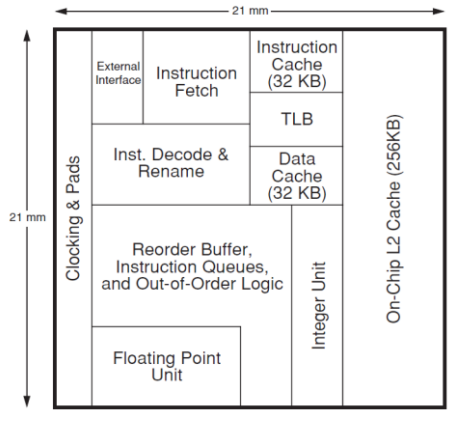
Processor Scaling Since ~2005

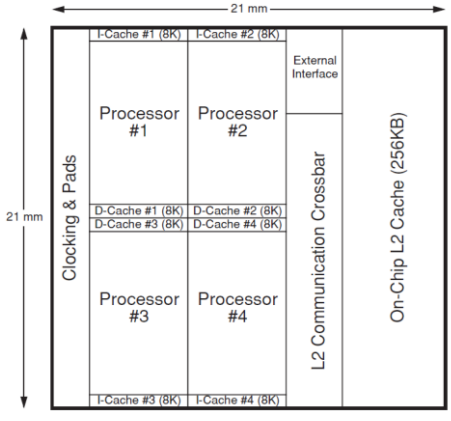


The Multicore Alternative

- Use Moore's law to place more cores per chip
  - Potentially 2x cores/chip with each CMOS generation
  - Without significantly compromising clock frequency
  - Known as Multi-Core Processors (MCP) or Chip Multiprocessors (CMP)
- The good news
  - Continued scaling of chip-level peak (throughput) performance
  - Mitigate the undesirable superscalar power scaling ("wrong side of the square law")
  - Facilitate design and verification, and product differentiation
- The bad news
  - Require multithreaded workloads: multiple programs or parallel programs
  - Require parallelizing single applications into parallel programs
  - Power is still an issue as transistors shrink due to leakage current

OOO Superscalar vs. Chip Multiprocessor





IPC Scaling of Wide-Issue OOO Superscalar

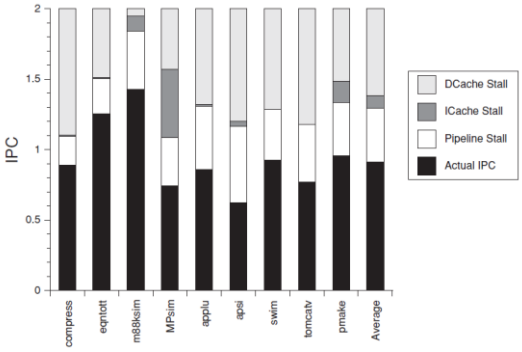


FIGURE 1.7: IPC breakdown for a single 2-issue processor.

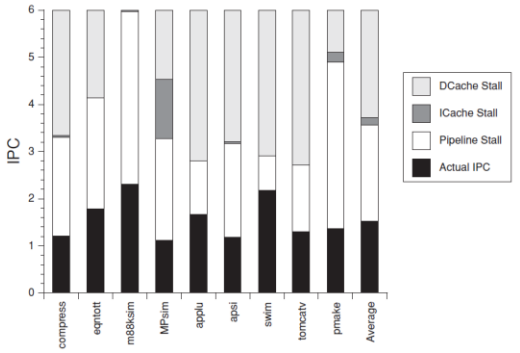
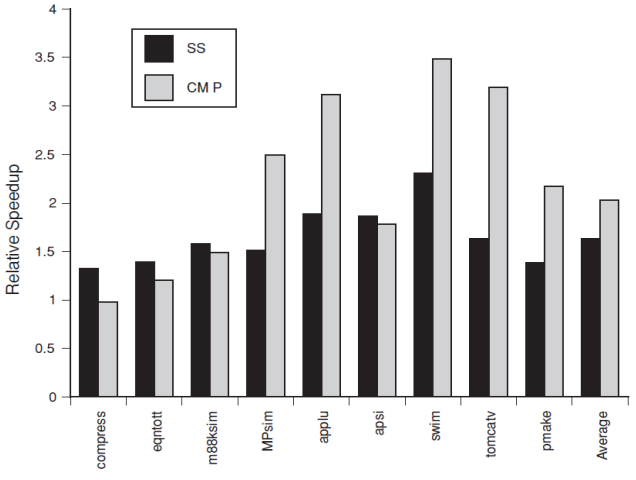
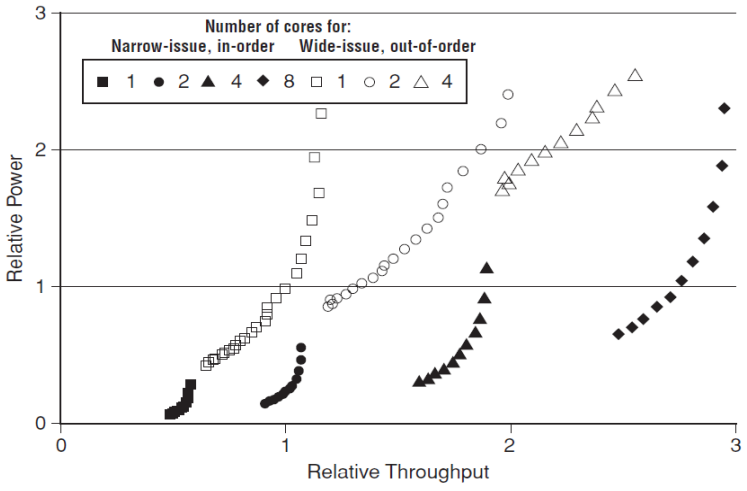


FIGURE 1.8: IPC breakdown for the 6-issue processor.

OOO Superscalar vs. Chip Multiprocessor Speedups



Power Scaling for In-Order vs. OOO Cores



B. Throughput Performance

- Law #1 - CPU (Latency) Performance
- Law #2 – CMP (Throughput) Performance

Law #1 – CPU (Latency) Performance

- Time to execute a program: T (latency)

$$T = \frac{instructions}{program} \times \frac{cycles}{instruction} \times \frac{time}{cycle}$$

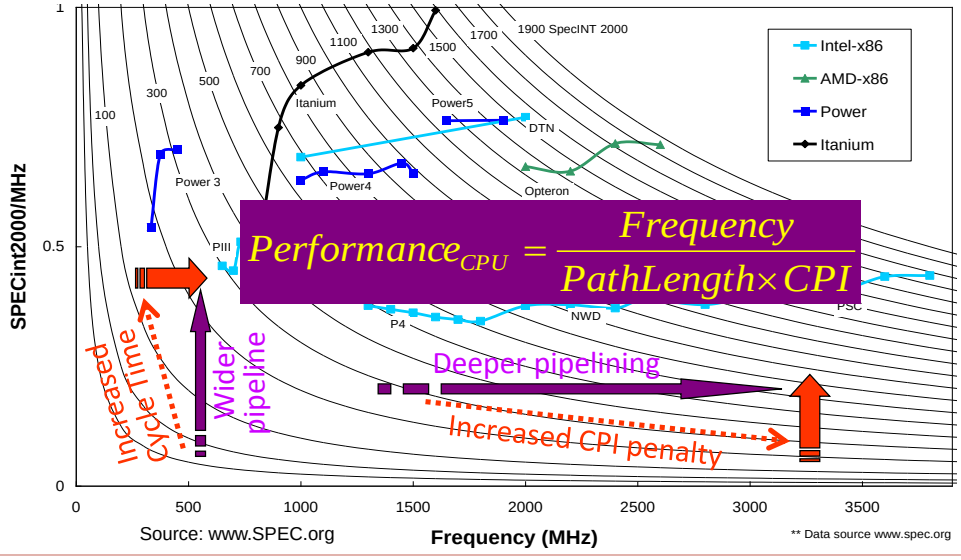
$$T = PathLength \times CPI \times CycleTime$$

- Processor performance: Perf = 1/T

$$Perf_{CPU} = \frac{1}{PathLength \times CPI \times CycleTime} = \frac{Frequency}{PathLength \times CPI}$$

[John DeVale & Bryan Black, 2005]

SPECint (Latency) Performance of Processors



Latency vs. Throughput Performance

- Reduce Latency of Application
  - Uni-processor, Single Program
  - Target Single-Thread Performance
  - Examples: SPEC, PC and Workstations
- Increase Throughput of System
  - Multi-processors, Many Threads/Tasks
  - Target Multi-threaded/Multi-tasking Throughput
  - Example: Database Transaction Processing

Law #2 – CMP (Throughput) Performance

- Time to process a thread/task: T (latency)

$$T = \frac{instructions}{thread} \times \frac{cycles}{instruction} \times \frac{time}{cycle}$$

$$T = PathLength \times CPI \times CycleTime$$

- MP (n processors) performance: Perf = 1/T

$$Perf_{MP} = \frac{n \times 1}{PathLength \times CPI \times CycleTime} = \frac{n \times Frequency}{PathLength \times CPI}$$

Iron Law of Multithread/Multicore Performance

- Multi-Core Performance:

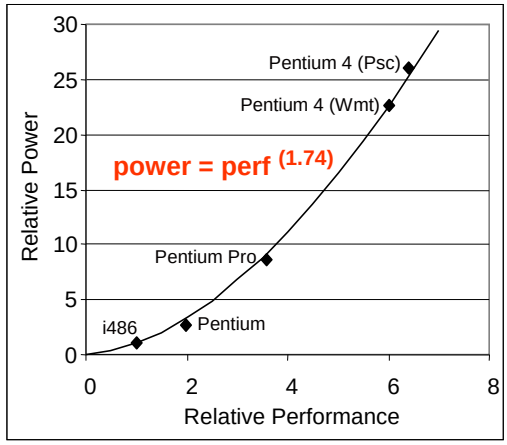
$$Perf_{MC} = \frac{n \times Frequency}{PL(n) \times CPI(n)}$$

- Can Improve $Perf_{MC}$ by:
  - Increasing: n (no. of CPUs or cores)
  - Increasing: $Frequency$ (CPU clock frequency)
  - Decreasing: PL (dynamic instruction count)
  - Decreasing: CPI (cycles/instruction)

Power and Scalar (Latency) Performance

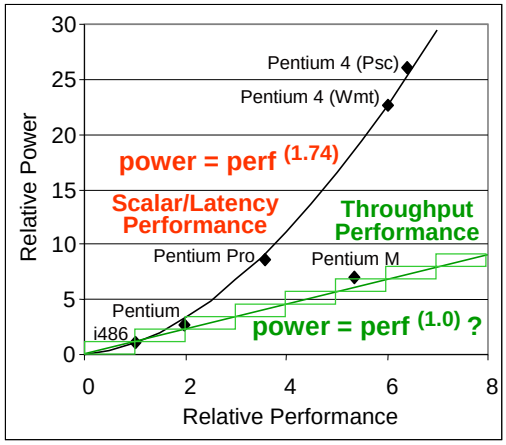
[Ed Grochowski, 2004]

- For comparison
  - Factor out contributions due to process technology
  - Keep contributions due to microarchitecture design
  - Normalize to i486™ processor
- Relative to i486™ Pentium® 4 (Wmt) processor is
  - 6x faster (2X IPC at 3X frequency)
  - 23x higher power
  - Spending 4 units of power for every 1 unit of scalar performance

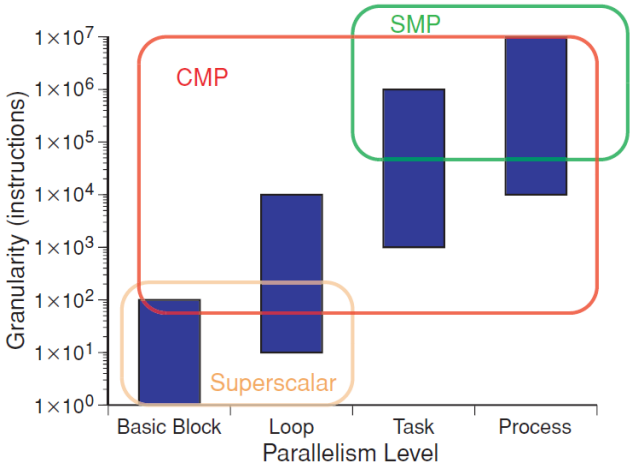


Power and Throughput Performance

- Assume a large-scale CMP with potentially many cores
- Replication of cores results in nearly proportional increases to both throughput performance and power (hopefully).



Program Parallelism and Processor Parallelism



- SMP: "Symmetric Multiprocessors" – large system with multiple processors on multiple chips.
- CMP: "Chip Multiprocessors" – single chip with multiple processors or multiple "cores".

Chip Multiprocessors (CMP)

- CMPs HAVE SEVERAL DESIRABLE PROPERTIES
  - DESIGN SIMPLICITY
  - IMPROVED POWER SCALABILITY
  - LOW-LATENCY COMMUNICATION
  - MODULARITY AND CUSTOMIZATION
- CMPs CAN BE HOMOGENEOUS OR HETEROGENEOUS
  - DEPENDS ON WHETHER THE CORES ARE IDENTICAL OR NOT

C. Design Space Exploration

- Core Types and Organization
- Memory Hierarchy and Caches
- On-Chip Interconnects

Multicore Questions

- Type of cores
  - E.g. few OOO cores Vs many simple cores
  - More on this at a later lecture
- Memory hierarchy
  - Which caching levels are shared and which are private
  - How to effectively share a cache
  - More on this in the next lecture
- On-chip interconnect
  - Bus Vs ring Vs scalable interconnect (e.g., mesh)
  - Flat Vs hierarchical
- HW assists for parallel programming
  - HW support for fine-grain scheduling, transactions, etc

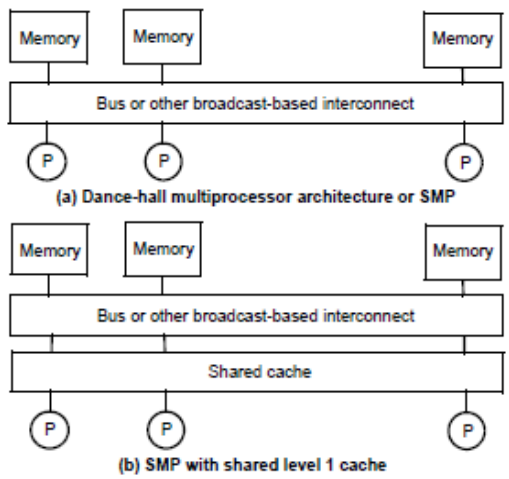
CMP Cache Bandwidth

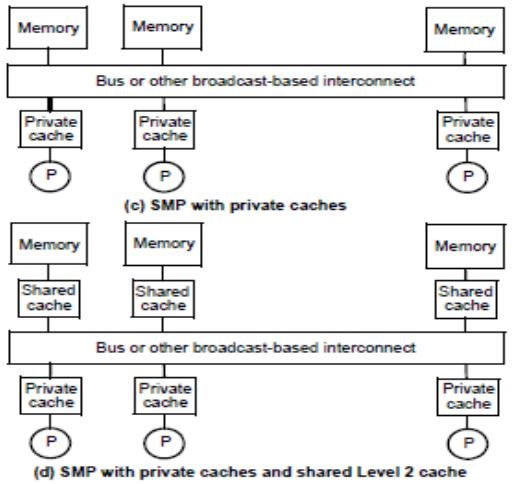
- OFF-CHIP BANDWIDTH IS A CRITICAL RESOURCE IN CMP
  - PIN COUNT, FREQUENCY
- MEMORY WALL PROBLEM REPLACED BY BANDWIDTH PROBLEM
  - REDUCE BW REQUIREMENT OF THREADS
  - AUGMENT AVAILABLE BANDWIDTH
- SHARED CACHE PREFERABLE
  - NO COHERENCE PROBLEM
  - COOPERATIVE SHARING (CODE AND DATA SHARING): GOOD
    - EFFECTIVE SIZE IS GREATER BECAUSE OF SHARING
    - PREFETCHING EFFECTS (ON SHARED CODE AND DATA)
  - DESTRUCTIVE INTERFERRENCE: BAD
    - THREADS DO NOT HELP EACH OTHERS
    - EACH THREAD SHARING THE CACHE COMPETES WITH OTHER THREADS
    - AS THE THREAD COUNT INCREASES, THE AMOUNT OF CACHE PER THREAD DECREASES
    - MISSES AND CONTENTION BOTH INCREASE

High-Level Design Issues

- Share caches?
  - yes: all designs that connect at L2 or L3
  - no: initial designs that connected at "bus"
- Coherence?
  - Private caches? Reuse existing MP/socket coherence
    - Optimize for on-chip sharing?
  - Shared caches?
    - Need new coherence protocol for on-chip caches
    - Often write-through L1 with back-invalidates for other caches

Cache Organization for Parallel Systems





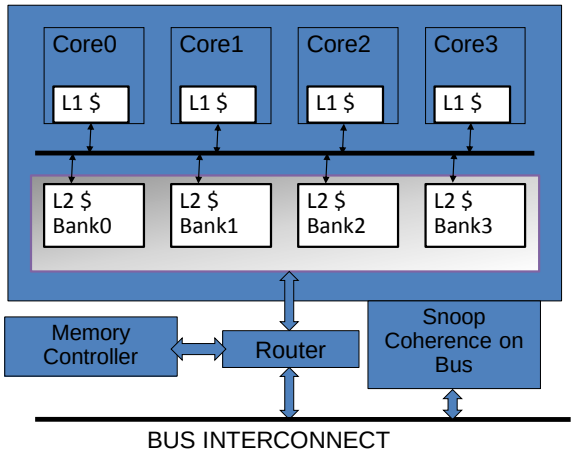
High-Level Design Issues

- How to connect?
  - Off-chip bus? Time-to-market hack, not scalable
  - Existing pt-to-pt coherence interconnect
    - e.g. AMD's hypertransport
  - Shared L2/L3:
    - Crossbar, up to 3-4 cores
    - 1D "dancehall" organization with crossbar
  - On-chip bus? Not very scalable
  - Interconnection network
    - scalable, but high overhead, design complexity
    - E.g. ring, 2D tiled organization, mesh interconnect

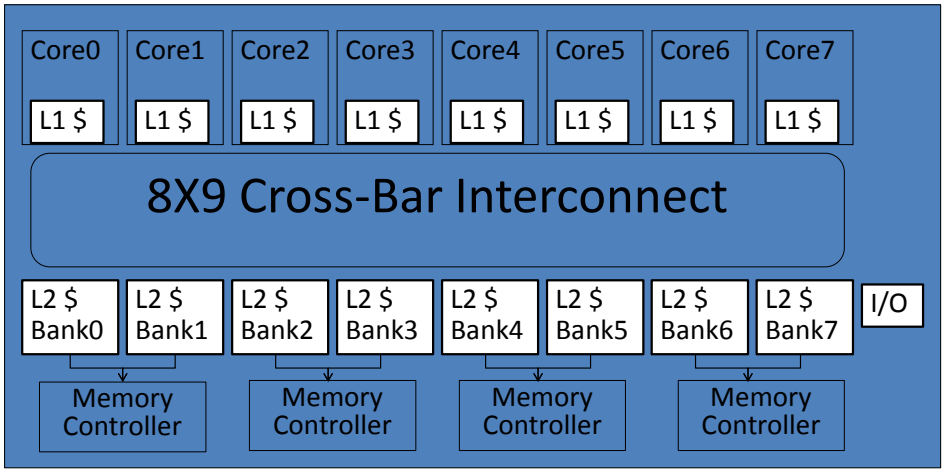
Multicore Interconnects

- Bus/Crossbar
- Point-to-point links, many possible topographies
  - 2D (suitable for planar realization)
    - Ring
    - Mesh
    - 2D torus
  - 3D - may become more interesting with 3D packaging (chip stacks)
    - Hypercube
    - 3D Mesh
    - 3D torus
- More detail in subsequent lecture

Bus-Based CMPs (e.g. Pentium 4 Dual Core)



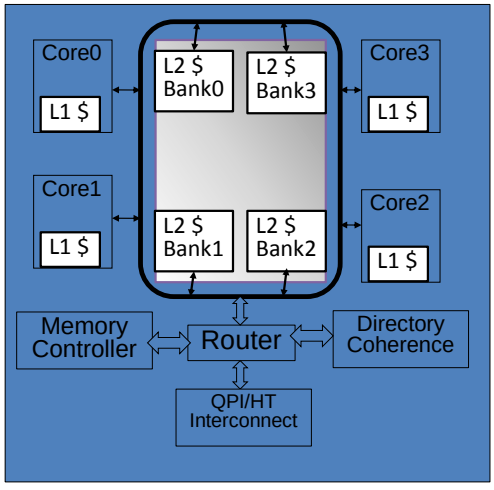
Crossbar-Based CMPs (e.g. IBM Power4/5/6/7)



On-Chip Bus/Crossbar Interconnects

- Used widely (Power4/5/6/7 Piranha, Niagara, etc.)
  - Assumed not scalable
  - Is this really true, given on-chip characteristics?
  - May scale "far enough": watch out for arguments at the limit
  - e.g. swizzle-switch makes x-bar scalable enough [UMich]
- Simple, straightforward, nice ordering properties
  - Wiring can be a nightmare (for crossbar)
  - Bus bandwidth is weak (even multiple busses)
  - Compare DEC Piranha 8-lane bus (32GB/s) to Power4 crossbar (100+GB/s)
  - Workload demands: commercial vs. scientific

Ring-Based CMPs



On-Chip Ring Interconnect

- Point-to-point ring interconnect
  - Simple, easy
  - Nice ordering properties (unidirectional)
  - Every request a broadcast (all nodes can snoop)
  - Scales poorly: *O(n)* latency, fixed bandwidth
- Optical ring (nanophotonic)
  - HP Labs Corona project [Vantrease review]
  - Latency is arguably *O(sqrt(n))*
    - Covert switching – broadcast not easy any more
  - Still fixed bandwidth (but lots of it)

On-Chip Mesh

- Widely assumed in academic literature
- Tilera (Wentzlaff reading), Intel 80-core prototype
- Not symmetric, so have to watch out for load imbalance on inner nodes/links
  - 2D torus: wraparound links to create symmetry
    - Not obviously planar
    - Can be laid out in 2D but longer wires, more intersecting links
- Latency, bandwidth scale well
- Lots of existing literature

CMPs with Heterogeneous Cores

- WORKLOADS HAVE DIFFERENT CHARACTERISTICS
  - LARGE NUMBER OF SMALL CORES (APPLICATIONS WITH HIGH THREAD COUNT)
  - SMALL NUMBER OF LARGE CORES (APPLICATIONS WITH SINGLE THREAD OR LIMITED THREAD COUNT)
  - MIX OF WORKLOADS
  - MOST PARALLEL APPLICATIONS HAVE PARALLEL AND SERIAL SECTIONS (AMDAHL LAW)
- HENCE, HETEROGENEITY
  - TEMPORAL: e.g., EPI THROTTLING
  - SPATIAL: EACH CORE CAN DIFFER EITHER IN PERFORMANCE OR FUNCTIONALITY
- PERFORMANCE ASYMMETRY
  - USING HOMOGENEOUS COREs AND DVFS, OR PROCESSOR WITH MIXED CORES
  - VARIABLE RESOURCES: e.g., ADAPT SIZE OF CACHE BY GATING OFF POWER TO CACHE BANKS (UP TO 50%)
  - SPECULATION CONTROL (LOW BRANCH PREDICTION CODE): THROTTLE THE NUMBER OF IN-FLIGHT INSTRUCTIONS (REDUCES ACTIVITY FACTOR)

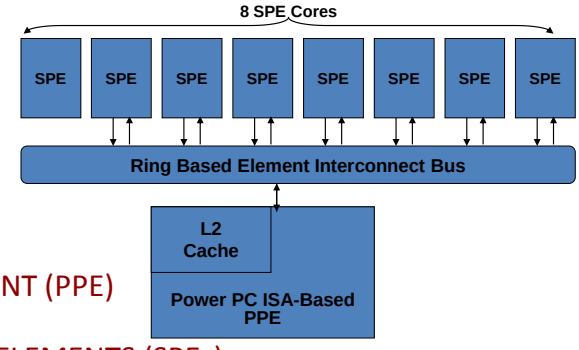
CMPs with Heterogeneous Cores

- FUNCTIONAL ASYMMETRY
  - USE HETEROGENEOUS CORES
    - E.G., GP CORES, GRAPHICS PROCESSORS, CRYTOGRAPHY, VECTOR CORES, FLOATING-POINT CO-PROCESSORS
    - HETEROGENEOUS CORES MAY BE PROGRAMMED DIFFERENTLY
    - MECHANISMS MUST EXIST TO TRANSFER ACTIVITY FOR ONE CORE TO ANOTHER
    - FINE-GRAIN: IN THE CASE OF FLOATING POINT CO-PROCESSOR, USE ISA
    - COARSE GRAIN: TRANSFER THE COMPUTATION FROM ONE CORE TO ANOTHER USING APIs
  - EXAMPLES:
    - CORES WITH DIFFERENT ISAs
    - CORES WITH DIFFERENT CACHE SIZES, DIFFERENT ISSUE WIDTH, DIFFERENT BRANCH PREDICTORS
    - CORES WITH DIFFERENT MICRO-ARCHITECTURES (E.G., STATIC AND DYNAMIC)
    - DIFFERENT TYPES OF CORES (E.G., GP AND SIMD)
  - GOALS:
    - SAVE AREA (MORE CORES!)
    - SAVE POWER BY USING CORES WITH DIFFERENT POWER/PERFORMANCE CHARACTERISTICS FOR DIFFERENT PHASES OF EXECUTION

CMPs with Heterogeneous Cores

- DIFFERENT APPLICATIONS MAY HAVE BETTER PERFORMANCE/POWER CHARACTERISTICS ON SOME TYPES OF CORE (STATIC)
- SAME APPLICATION GOES THROUGH DIFFERENT PHASES THAT CAN USE DIFFERENT CORES MORE EFFICIENTLY (DYNAMIC)
  - EXECUTION MOVES FROM CORE TO CORE DYNAMICALLY
  - MOST INTERESTING CASE (DYNAMIC)
  - COST OF SWITCHING CORES (MUST BE INFREQUENT: SUCH AS O/S TIME-SLICE)
- ASSUME CORES WITH SAME ISA BUT DIFFERENT PERFORMANCE/ENERGY RATIO
  - NEED ABILITY TO TRACK PERFORMANCE AND ENERGY TO MAKE DECISIONS
  - GOAL: MINIMIZE DELAY-ENERGY PRODUCT
  - SAMPLE PERFORMANCE AND ENERGY SPENT PERIODICALLY
  - TO SAMPLE, RUN APPLICATION ON ONE OR MULTIPLE CORES IN SMALL INTERVALS

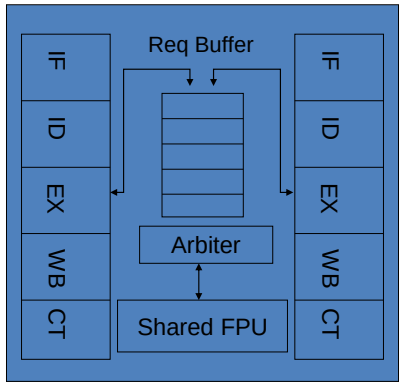
IBM CELL Processor



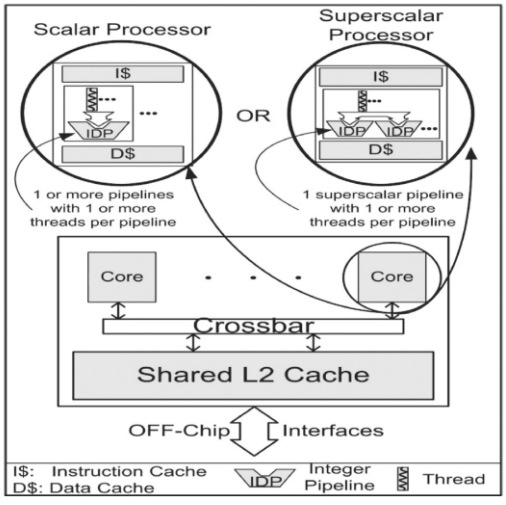
- ONE PowerPC PROCESSING ELEMENT (PPE)
  - 2-WAY SMT Power CORE
- PLUS 8 SYNERGISTIC PROCESSING ELEMENTS (SPEs)
  - SPE IS 2-ISSUE IN-ORDER PROCESSOR
  - TWO SIMD INSTRUCTIONS CAN BE ISSUED IN EACH CYCLE (VECTORS)
  - NO COHERENCE SUPPORT BETWEEN SPE AND PPE (DATA TRANSFERS ARE EXPLICITELY PROGRAMMED)

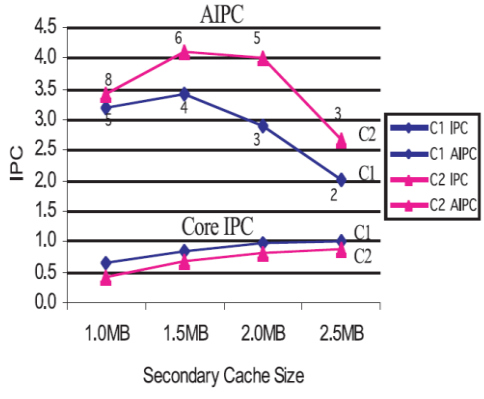
Conjoined Cores

- IN SIMULTANEOUS MULTITHREADING ALL THREADS SHARE ALL THE RESOURCES OF THE CORE (CACHES, FUs, NETWORK INTERFACE...)
- IN CMPs THREADS RUN ON DIFFERENT, DISJOINT CORES
- AN INTERMEDIATE SOLUTION IS CONJOINED CORES, IN WHICH CORES SHARE SOME (BUT NOT ALL) PHYSICAL RESOURCES SUCH AS
  - L1 I-CACHE PORTS
  - L1 D-CACHE PORTS
  - INTERCONNECT INTERFACE
  - CO-PROCESSORS
- EXAMPLE: SUN SPARC T1
  - 8 CORES ON A CHIP
  - SHARING A SINGLE FPU
  - CORE SENDS OPCODE, OPS, CORE ID
  - FPU SENDS RESULT
  - THREAD REQUESTING FPU IS SUSPENDED

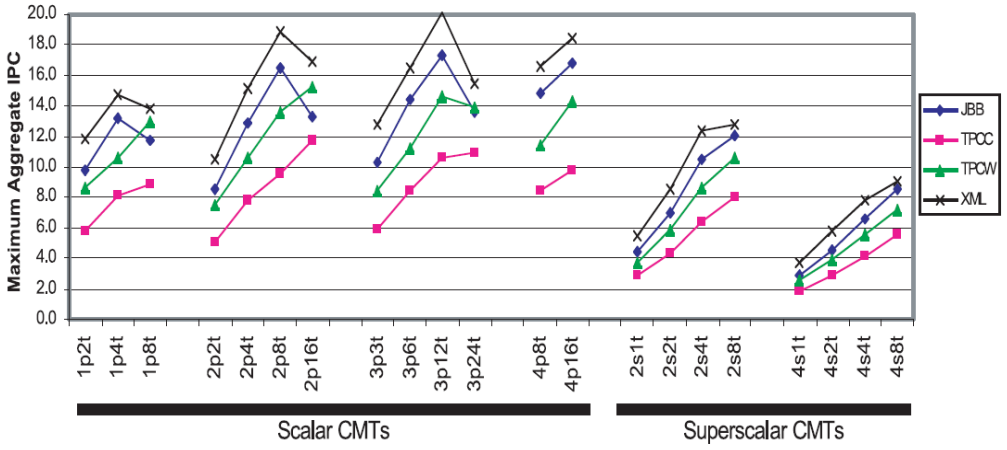


Explore MT/MC Processor Design Space





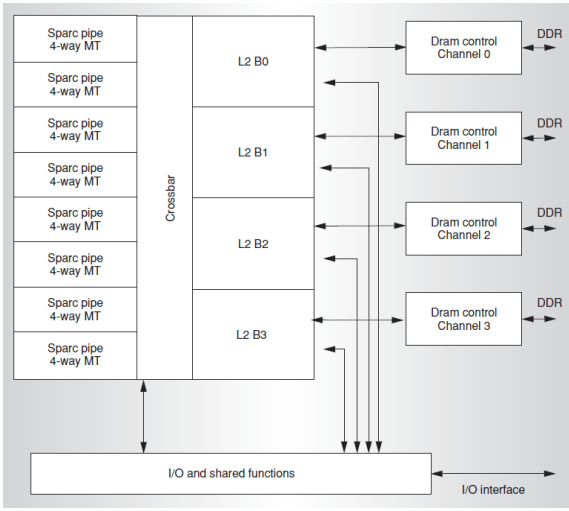
Explore MT/MC Processor Design Space

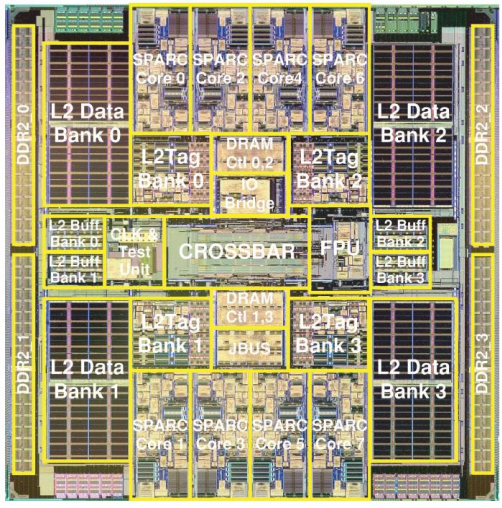


D. Niagara Case Study

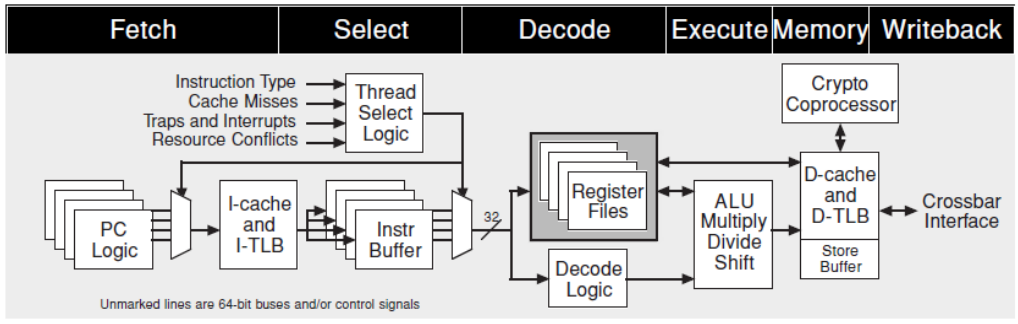
- Niagara 1
- Niagara 2

Sun Niagara 1

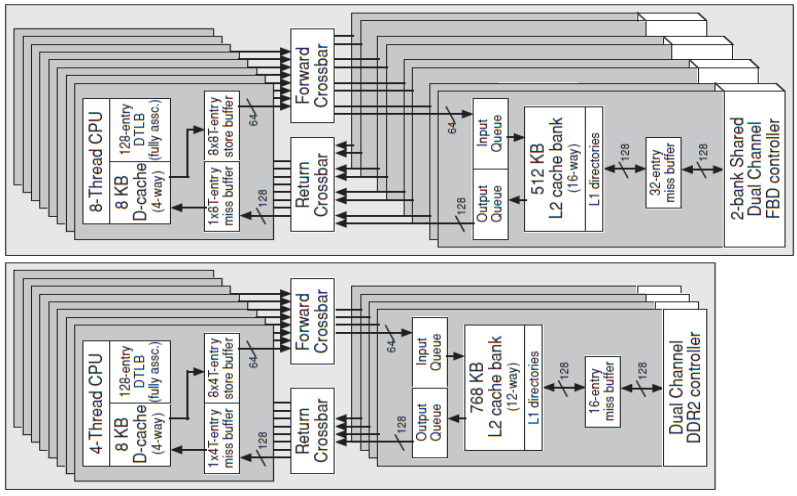




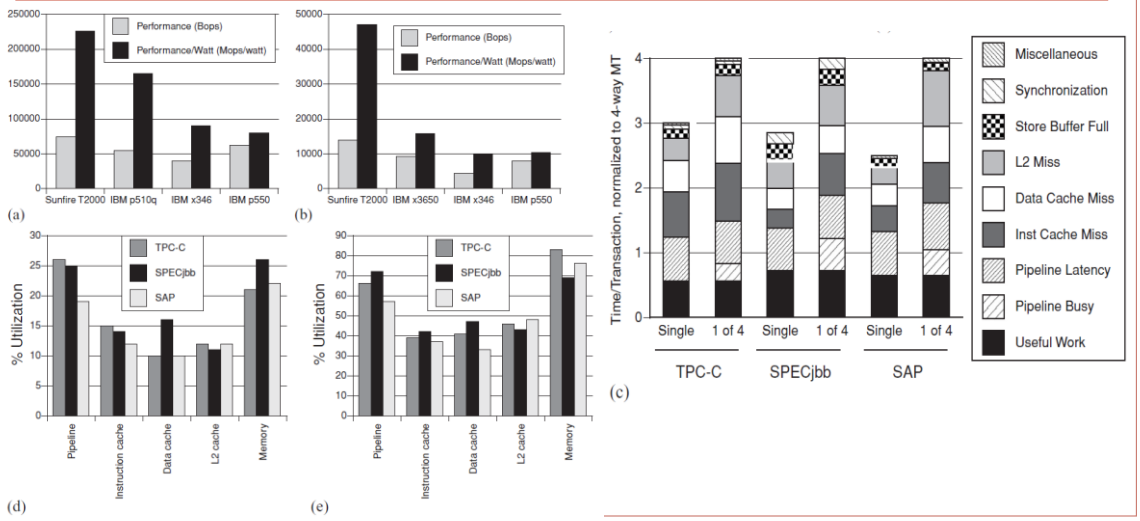
Sun Niagara Pipeline Stages



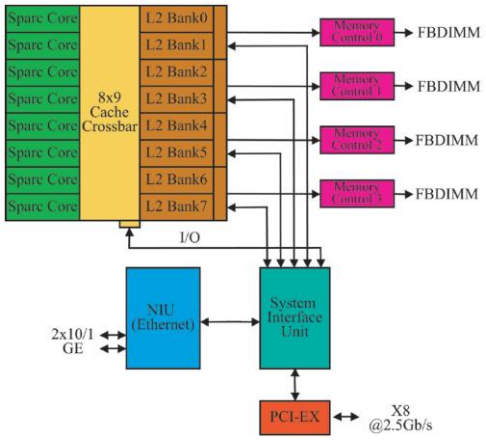
Niagara 1 and Niagara 2 Memory Hierarchies

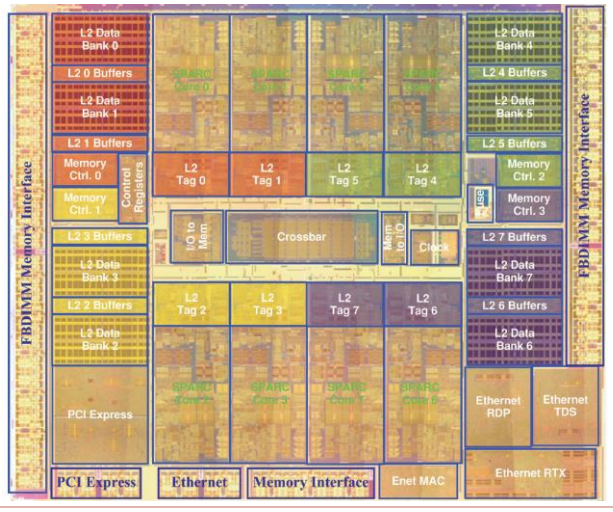


Performance Comparison on Server Workloads



Sun Niagara 2





Shared Memory Multiprocessors

- All processor cores have access to unified physical memory
  - The can communicate using loads and stores
- Advantages
  - Looks like a better multithreaded processor (multitasking)
  - Requires evolutionary changes the OS
  - Threads within an app communicate implicitly without using OS
    - Simpler to code for and low overhead
  - App development: first focus on correctness, then on performance
- Disadvantages
  - Implicit communication is hard to optimize
  - Synchronization can get tricky
  - Higher hardware complexity for cache management