Module 8: Memory Consistency Models and Case Studies of Multi-core

Lecture 15: Memory Consistency Models and Case Studies of Multi-core

The Lecture Contains:

Previous Next

Memory Consistency Models

Memory Consistency

- Coherence protocol is not enough to completely specify the output(s) of a parallel program
  - Coherence protocol only provides the foundation to reason about legal outcome of accesses to the same memory location
  - Consistency model tells us the possible outcomes arising from legal ordering of accesses to all memory locations
  - A shared memory machine advertises the supported consistency model; it is a “contract” with the writers of parallel software and the writers of parallelizing compilers
  - Implementing memory consistency model is really a hardware-software tradeoff: a strict sequential model (SC) offers execution that is intuitive, but may suffer in terms of performance; relaxed models (RC) make program reasoning difficult, but may offer better performance

SC

- Recall that an execution is SC if the memory operations form a valid total order i.e. it is an interleaving of the partial program orders
  - Sufficient conditions require that a new memory operation cannot issue until the previous one is completed
  - This is too restrictive and essentially disallows compiler as well as hardware re-ordering of instructions
  - No microprocessor that supports SC implements sufficient conditions
  - Instead, all out-of-order execution is allowed, and a proper recovery mechanism is implemented in case of a memory order violation
  - Let's discuss the MIPS R10000 implementation

Previous Next

SC in MIPS R10000

- Issues instructions out of program order, but commits in order
  - The problem is with speculatively executed loads: a load may execute and use a value long before it finally commits
  - In the meantime, some other processor may modify that value through a store and the store may commit (i.e. become globally visible) before the load commits: may violate SC (why?)
  - How do you detect such a violation?
  - How do you recover and guarantee an SC execution?
  - Any special consideration for prefetches ? Binding and non-binding prefetches
- In MIPS R10000 a store remains at the head of the active list until it is completed in cache
  - Can we just remove it as soon as it issues and let the other instructions commit (the store can complete from store buffer at a later point)? How far can we go and still guarantee SC?
- The Stanford DASH multiprocessor, on receiving a read reply that is already invalidated, forces the processor to retry that load
  - Why can't it use the value in the cache line and then discard the line?
- Does the cache controller need to take any special action when a line is replaced from the cache?

Relaxed Models

- Implementing SC requires complex hardware
  - Is there an example that clearly shows the disaster of not implementing all these?
- Observe that cache coherence protocol is orthogonal
  - But such violations are rare
  - Does it make sense to invest so much time (for verification) and hardware (associative lookup logic in load queue)?
  - Many processors today relax the consistency model to get rid of complex hardware and achieve some extra performance at the cost of making program reasoning complex
  - P0: A=1; B=1; flag=1; P1: while (!flag); print A; print B;
  - SC is too restrictive; relaxing it does not always violate programmers' intuition

Previous Next

Case Studies

CMP

- CMP is the mantra of today's microprocessor industry
  - Intel's dual-core Pentium 4: each core is still hyperthreaded (just uses existing cores)
  - Intel's quad-core Whitefield is coming up in a year or so
  - For the server market Intel has announced a dual-core Itanium 2 (code named Montecito); again each core is 2-way threaded
  - AMD has released dual-core Opteron in 2005
  - IBM released their first dual-core processor POWER4 circa 2001; next-generation POWER5 also uses two cores but each core is also 2-way threaded
  - Sun's UltraSPARC IV (released in early 2004) is a dual-core processor and integrates two UltraSPARC III cores

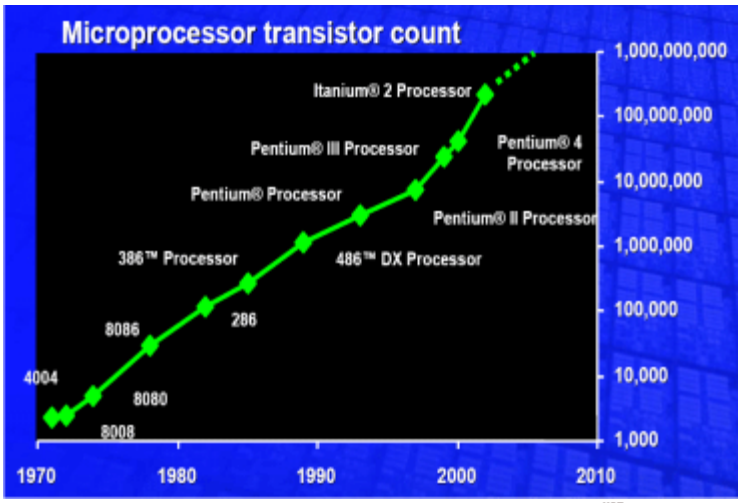
Why CMP?

- Today microprocessor designers can afford to have a lot of transistors on the die
  - Ever-shrinking feature size leads to dense packing
  - What would you do with so many transistors?
  - Can invest some to cache, but beyond a certain point it doesn't help
  - Natural choice was to think about greater level of integration
  - Few chip designers decided to bring the memory and coherence controllers along with the router on the die
  - The next obvious choice was to replicate the entire core; it is fairly simple: just use the existing cores and connect them through a coherent interconnect

Previous Next

Moore's Law

- The number of transistors on a die doubles every 18-24 months
  - Exponential growth in available transistor count
  - If transistor utilization is constant, this would lead to exponential performance growth; but life is slightly more complicated
  - Wires don't scale with transistor technology: wire delay becomes the bottleneck
  - Short wires are good: dictates localized logic design
  - But superscalar processors exercise a “centralized” control requiring long wires (or pipelined long wires)
  - However, to utilize the transistors well, we need to overcome the memory wall problem
  - To hide memory latency we need to extract more independent instructions i.e. more ILP
- Extracting more ILP directly requires more available in-flight instructions
  - But for that we need bigger ROB which in turn requires a bigger register file
  - Also we need to have bigger issue queues to be able to find more parallelism
  - None of these structures scale well: main problem is wiring
  - So the best solution to utilize these transistors effectively with a low cost must not require long wires and must be able to leverage existing technology: CMP satisfies these goals exactly (use existing processors and invest transistors to have more of these on-chip instead of trying to scale the existing processor for more ILP)



Previous Next

Power Consumption?

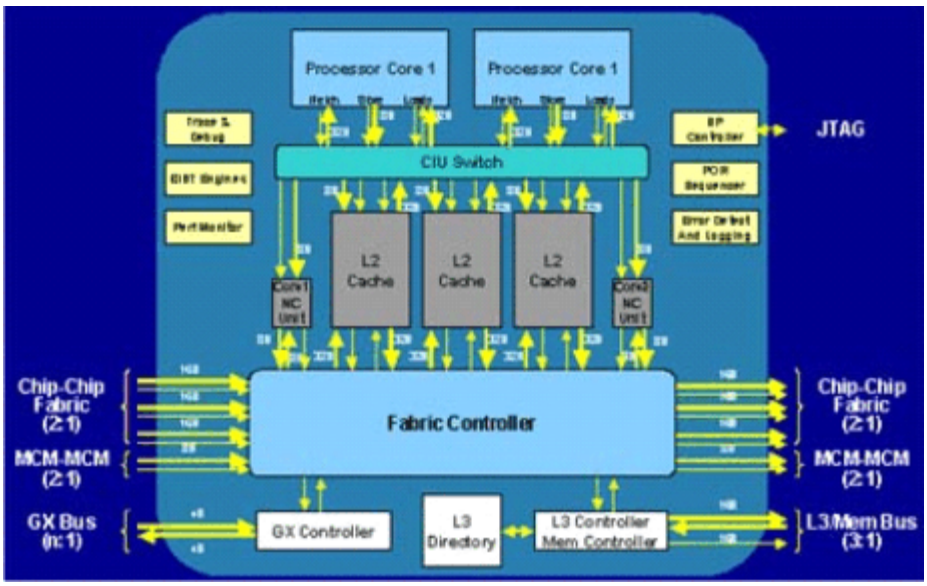
- Hey, didn't I just make my power consumption roughly N-fold by putting N cores on the die?
  - Yes, if you do not scale down voltage or frequency
  - Usually CMPs are clocked at a lower frequency
- Oops! My games run slower!
  - Voltage scaling happens due to smaller process technology
  - Overall, roughly cubic dependence of power on voltage or frequency
  - Need to talk about different metrics
- Performance/Watt (same as reciprocal of energy)
- More general, Performance k+1 /Watt (k > 0)
  - Need smarter techniques to further improve these metrics Online voltage/frequency scaling

ABCs of CMP

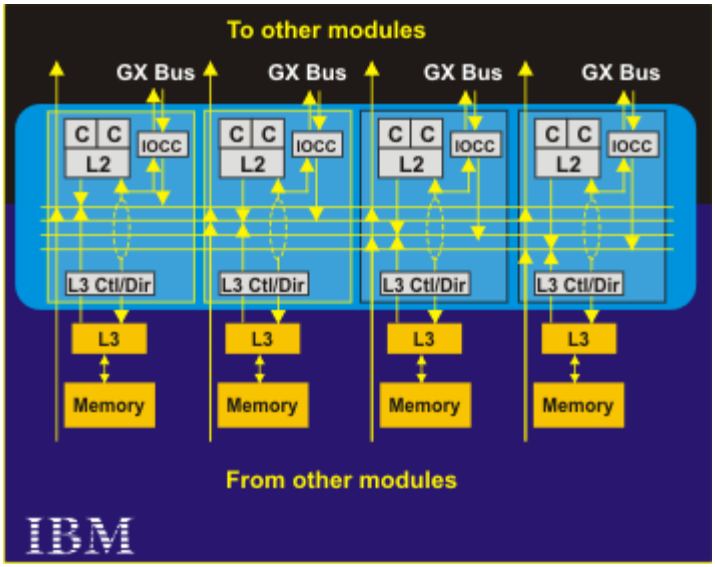
- Where to put the interconnect?
  - Do not want to access the interconnect too frequently because these wires are slow
  - It probably does not make much sense to have the L1 cache shared among the cores: requires very high bandwidth and may necessitate a redesign of the L1 cache and surrounding load/store unit which we do not want to do; so settle for private L1 caches, one per core
  - Makes more sense to share the L2 or L3 caches
  - Need a coherence protocol at L2 interface to keep private L1 caches coherent: may use a high-speed custom designed snoopy bus connecting the L1 controllers or may use a simple directory protocol
  - An entirely different design choice is not to share the cache hierarchy at all (dual-core AMD and Intel): rids you of the on-chip coherence protocol, but no gain in communication latency

Previous Next

IBM POWER4

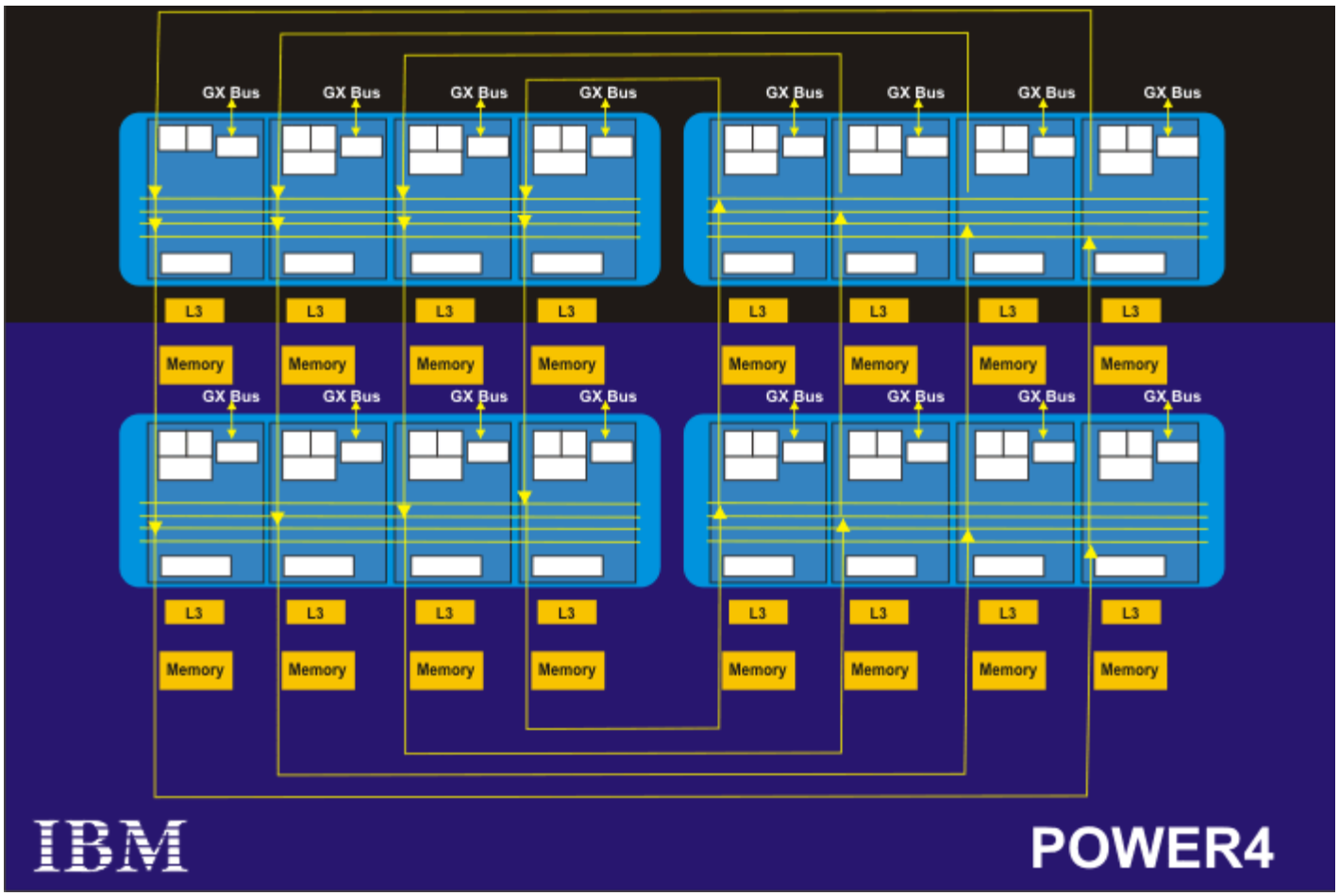


4-chip 8-way NUMA



Previous Next

32-way: Ring Bus

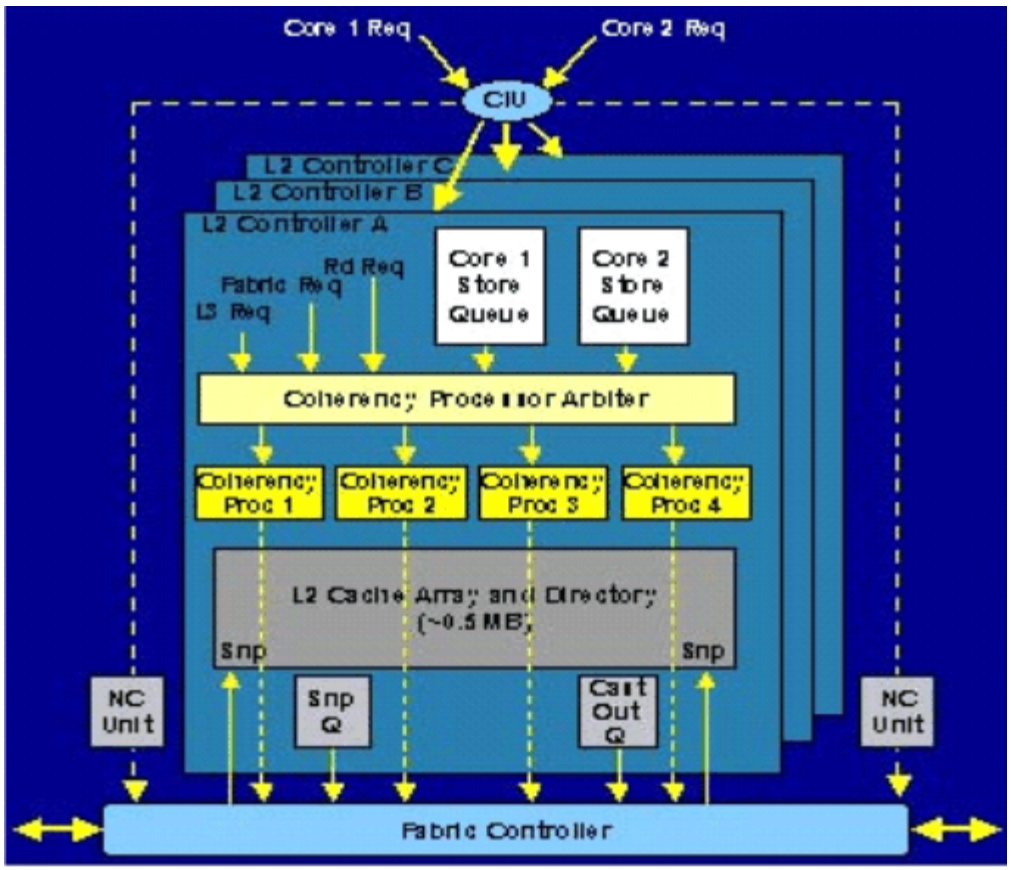


POWER4 Caches

- Private L1 instruction and data caches (on chip)
  - L1 icache : 64 KB/direct mapped/128 bytes line
  - L1 dcache : 32 KB/2-way associative/128 bytes line/LRU
  - No M state in L1 data cache (write through)
- On-chip shared L2 (on-chip coherence point)
  - 1.5 MB/8-way associative/128 bytes line/pseudo LRU
  - For on-chip coherence, L2 tag is augmented with a two-bit sharer vector; used to invalidate L1 on other core's write
  - Three L2 controllers and each L2 controller has four local coherence units; each L2 controller handles roughly 512 KB of data divided into four SRAM partitions
  - For off-chip coherence, each L2 controller has four snoop engines; executes enhanced MESI with seven states

Previous Next

POWER4 L2 Cache



Previous Next