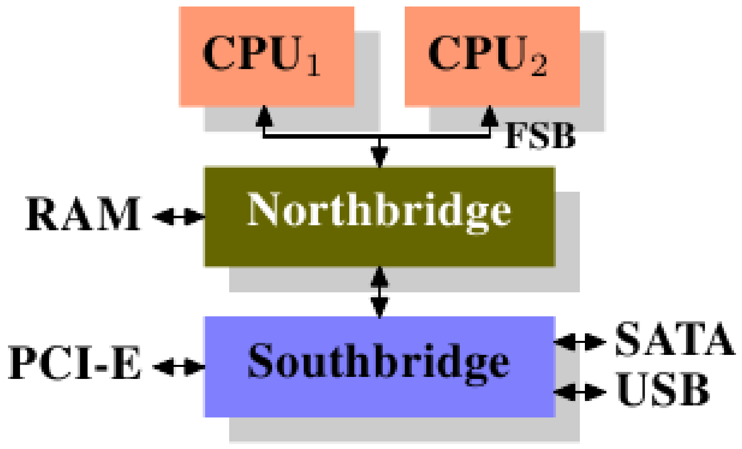
Memory and Memory Caches

Joe Sventek

Commodity Hardware Today

- All CPUs connected via a common bus (the Front Side Bus) to the Northbridge
- Northbridge contains the memory controller
- To reach other system devices, Northbridge must communicate with Southbridge
- Southbridge (aka I/O bridge) provides access to a variety of different buses



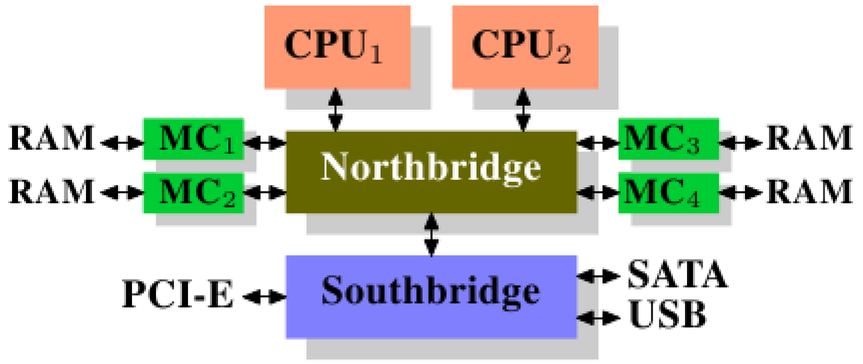
Consequences of this structure

- All interCPU communication must travel over the same bus used to communicate with Northbridge
- All communication with RAM must pass through Northbridge
- The RAM has only a single port
- Communications between a CPU and a device attached to Southbridge is routed through Northbridge

Bottlenecks

- DMA access creates contention for the bandwidth of Northbridge, as DMA requests compete with RAM access from the CPUs
- The single port to the memory (which may be one or more channels) is another source of contention; with limited bandwidth available, Northbridge must schedule memory access in ways that minimize delays

External controllers



- On more expensive systems, Northbridge does not contain the memory controller; instead, it can be connected to a number of external memory controllers
- This provides enhanced memory bus bandwidth
- System still limited by the internal bandwidth of the Northbridge

Types of RAM

Static RAM (SRAM)

- Maintaining the state of a memory cell requires constant power
- The memory cell state is available for reading almost immediately once the word access line is raised
- The memory cell state is stable, no refresh cycles are needed.

Dynamic RAM (DRAM)

- Due to leakage, each cell must be constantly refreshed (~ every 64ms); during refresh, no access to the memory is possible
- The memory cell state is not immediately available for reading
- The chip real estate needed for a DRAM cell is many times smaller than for SRAM; thus can pack many more DRAM cells on a chip, and DRAM is cheaper

Addressing Northbridge contention

- Use SRAM – not economical, SRAM as fast as current CPU cores is orders of magnitude more expensive than DRAM
- Introduce a small amount of high-speed SRAM to complement large amounts of DRAM
- The SRAM could be part of the addressable memory of the machine, forcing the OS to optimally distribute instructions/data to make best use of the SRAM
  - Logistically a nightmare, as requires each process to administer in software the allocation of the SRAM memory region.

Caches

- Make the SRAM a resource that is transparently used and administered by the processors
- SRAM is used to make temporary copies of (i.e. to cache) data in main memory which is likely to be used soon by the processor
- Possible because program code and data exhibit spatial and temporal locality
  - Spatial – loops in code, sequential access to data
  - Temporal – likely that the same data/instructions will be reused before long

Example speedup

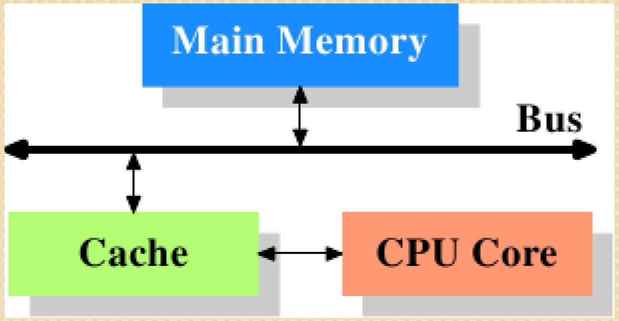
- Assume access to main memory takes 200 cycles and access to cache memory takes 15 cycles.
- If code uses 100 data elements 100 times:
  - If no cache, then code will spend 2,000,000 cycles on memory operations
  - If all data is in the cache, code will spend 168,500 cycles
  - This is an improvement of 91.5%

Can the working set fit in the cache?

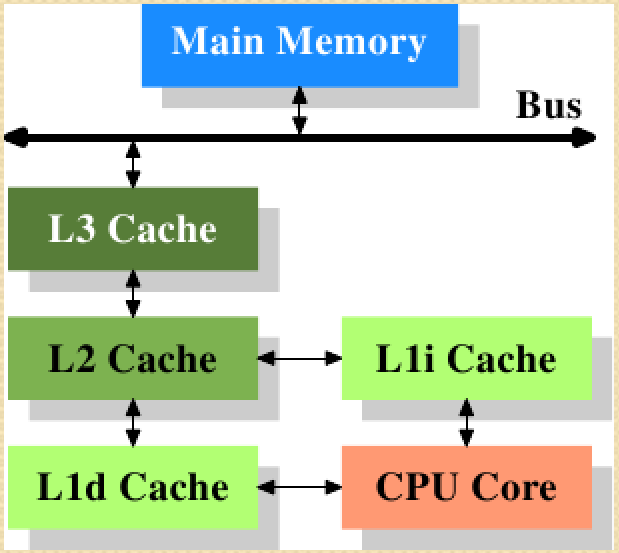
- Cache size ~ 10^{-3} x DRAM size (e.g. 4MB cache for a system with 4 GB of DRAM)
- Working set for all processes is bound to be larger than the cache
- Need a set of good strategies to determine what should be cached at any given time
- Since not all data of the working set is used at exactly the same time, can use techniques to temporarily replace some data in the cache with other data

Where do caches fit in the architecture?

Minimum	More Typical

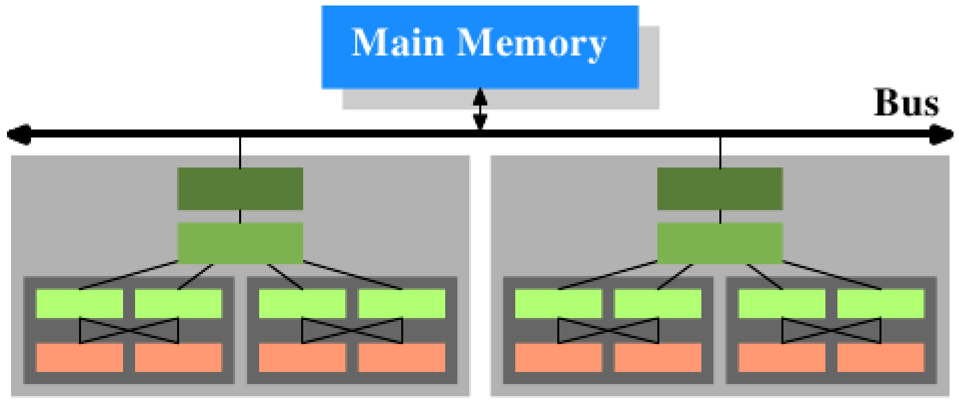


- All loads and stores go through the Cache
- The Bus is the FSB
- Assume Northbridge is there to facilitate CPU-mem communications



- Separate instruction and data caches
- Each level is larger, but slower, than the previous level.

What about multicore?



- Each core has its own level 1 cache
- All cores on a processor share level 2 and level 3 caches
- The different processors do not share any caches

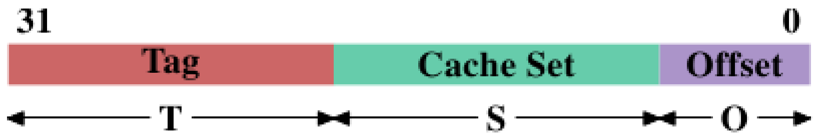
Cache operations

- All data read or written by a CPU core is stored in the cache
- When a CPU needs a data word, the caches are searched first
- The cache cannot contain the contents of all of main memory, and all memory addresses are cacheable
- Each cache entry is *tagged* using the address of the data word in main memory
- The address can be physical or virtual, depending upon the cache implementation

Cache operations (cont)

- Since the tag requires space in addition to the actual contents of the data word, it is inefficient to choose a word (e.g. 32-bits) as the granularity of the cache.
- Thus, entries stored in the cache are lines of several contiguous words
- Typical line size today is 64 bytes
- If the memory bus is 64bits wide, this means 8 transfers per cache line.
- This transport mode is supported efficiently by modern memories (DDR)

Cache operations (cont)



- When memory content is needed by the CPU, the entire cache line is loaded into the L1d
- In the figure above,
  - The cache line size is 2^O bits; the low O bits are the offset into the cache line
  - The next S bits select the Cache Set (see section 3.3)
  - The remaining T bits are used to distinguish all addresses with the same S part of the address
  - The S bits do not have to be stored in the cache since they are the same for all cache lines in the same set

Cache operations (cont)

- When an instruction modifies memory, the CPU has to load a cache line first because no instruction modifies an entire cache line at once
- It is not possible for a cache to hold partial cache lines
- A cache line that has been written to and which has not been written back to main memory is "dirty"
- Once it is written to main memory, the dirty flag is cleared

Interactions between caches

- To load new data in a cache, it is almost always first necessary to make room in the cache
- An eviction from L1d pushes the cache line down into L2 (uses the same cache size)
- This means that room must be made in L2, evicting the content into L3.
- Eviction from L3 means that the cache line must be written to main memory
- Exclusive cache – cache line is only in a single cache
- Inclusive cache – each cache line in L1d is also in L2

Cache management

- CPUs may manage the caches in any way they wish as long as the processor architecture memory model is not violated
- For symmetric multi-processor (SMP) systems, the CPU caches cannot work independently from each other.
  - All processors are supposed to see the same memory content at all times
  - Without collusion, a processor would not see the content of dirty cache lines in other processors

Cache coherency protocols

- Each CPU must be able to detect when another CPU wants to read or write a certain cache line (snooping)
- If write access is detected and the CPU has a clean copy of the cache line in its cache, it must mark the cache line as invalid; thus, future references by the detecting CPU will require reloading the cache line
- If read access is detected and the CPU has a clean copy, no action is needed

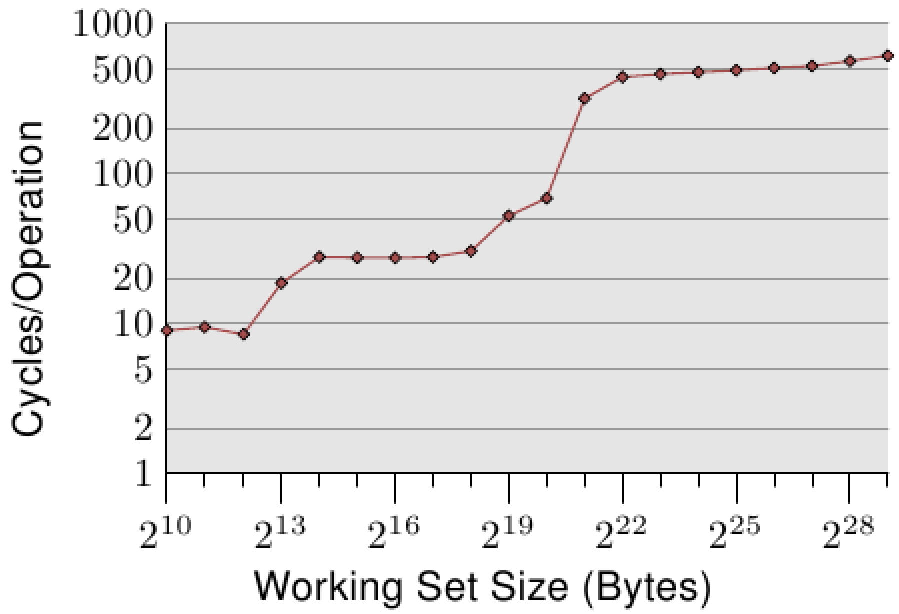
Cache coherency protocols (cont)

- If read or write access is detected and the CPU has a dirty copy of the cache line in its cache
  - Detecting CPU sends the dirty cache line directly to the requesting CPU
  - Detecting CPU writes the cache line to main memory (or the memory controller is supposed to notice the direct transfer and store the updated cache line in memory)
  - If write access was detected, the detecting CPU then invalidates its local cache line
- See 3.3.4 on the MESI protocol in the handout.

Relative costs

To Where	Cycles
Register	<= 1
L1d	~3
L2	~14
Main Memory	~240

- Costs provided by Intel for a Pentium M



- L1d is 2^{13} bytes
- L2 is 2^{20} bytes
- No L3 cache