APRIL 30, 2015

SMART MEMORY

OPERATING SYSTEM SEMINAR

ADITYA PARIKH
ID - 121001

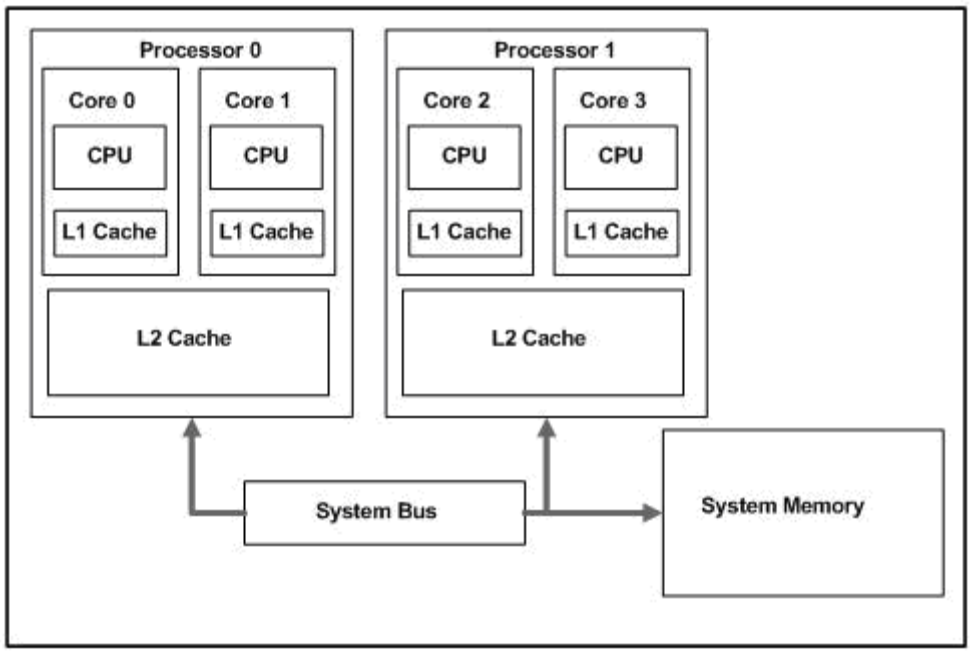
What is Smart Memory?

A Smart Memories chip is made up of many processing tiles, each containing local memory, local interconnect, and a processor core. In Smart Memories, the user can program the wires and the memory, as well as the processors. This allows the user to configure the computing substrate to better match the structure of the applications, which greatly increases the efficiency of the resulting solution. Smart Memories combines to create a partitioned, explicitly parallel, reconfigurable architecture for use as a future universal computing element. Since different application spaces naturally have different communication patterns and memory needs, finding a single topology that fits well with all applications is very difficult. Rather than trying to find a general solution for all applications, we tailor the appearance of the on-chip memory, interconnection network, and processing elements to better match the application requirements. We leverage the fact that long wires in current (and future) VLSI chips require active repeater insertion for minimum delay. The presence of repeaters means that adding some reconfigurable logic to these wires will only modestly impact their performance. Reconfiguration at this level leads to coarser-grained configurability than previous reconfigurable architectures, most of which were at least in part based on FPGA implementations. Compared to these systems, Smart Memories trades away some flexibility for lower overheads, more familiar programming models, and higher efficiency. At the highest level, a Smart Memories chip is a modular computer. It contains an array of processor tiles and on-die DRAM memories connected by a packet-based, dynamically-routed network. The network also connects to high-speed links on the pins of the chip to allow for the construction of multi-chip systems. Most of the initial hardware design works in the Smart Memories project has been on the processor tile design and evaluation, so this paper focuses on these aspects.

Aspects of Smart Memory:

A.) Smart Cache:

Smart Cache is a level 2 or level 3 caching method for multiple-execution cores. Smart Cache shares the actual cache memory among cores; both CPU cores and integrated GPU are sharing the same cache. In comparison to a dedicated per-core cache, the overall cache miss rate decreases in times where not all cores need equally much of the cache space. Consequently, a single core can use the full level 2 cache or level 3 cache, if the other cores are inactive. Furthermore, the shared cache makes it faster to share memory among different execution cores. As the trend moves from single-core to multi-core processors for the next computing performance leap, system architectures have several options for the organization of one of the most important system resources-the cache. Some architectures choose to keep the last-level cache private to each core for simplicity, while other architectures explore sharing the last-level cache among different cores for better performance/cost ratio and improved resource allocation. The shared cache architecture brings exciting new opportunities to software and system designers. Figure shows two processors (processor 0 and processor 1) sharing the same system bus and system memory. Inside each processor there are two CPU cores; each has its own private L1 cache while sharing the L2 cache.



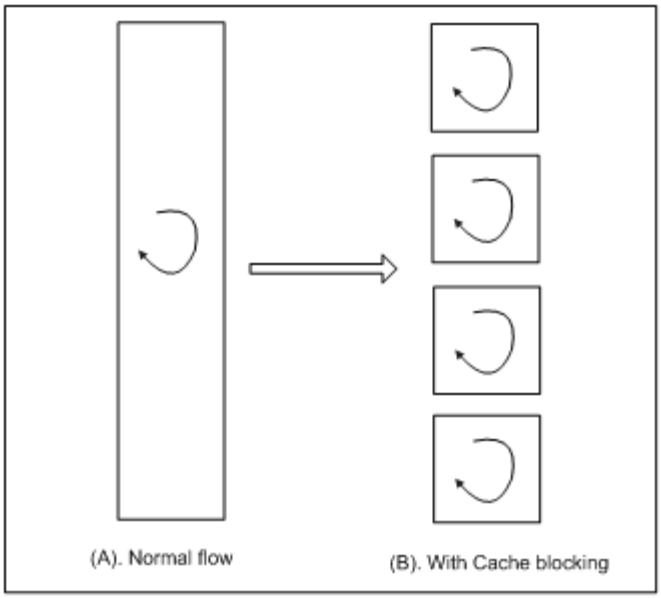
Software techniques for shared-cache multi-core systems

1.) Processor Affinity:

- In a multi-core environment, the control over which core to run a specific thread or application on is essential. Without this control, the threads/applications may get assigned to the wrong processors or cores and cause unnecessary performance degradation.
  It may be worth it to consider separating the contention-only threads (threads that are not sharing any data but need to use cache) from each other, and assigning them to cores that are not sharing the cache. On the other hand, it is also important to consider assigning data-sharing threads to cores that are sharing the cache, or even assigning them to the same core if they are closely tied to each other.

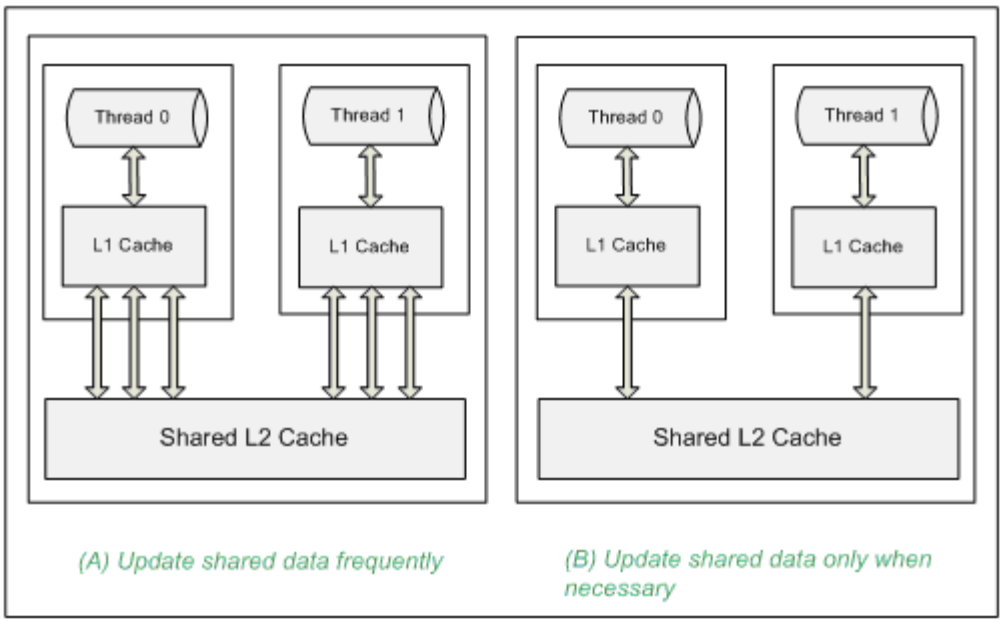
2.) Cache blocking (data tiling):

- The cache blocking technique (also called data tiling) tries to allow data to stay in the cache while being processed by data loops. By reducing the unnecessary cache traffic (fetching and evicting the same data throughout the loops), a better cache hit rate is achieved. Possible candidates for the cache blocking technique include large data sets that get operated on many times. By going through the entire data set and performing one operation, followed by another pass for the second operation, and so on, if the entire data set does not fit into the cache, the first elements in the cache will be evicted to fit the last elements in. Then, on subsequent iterations through the loop, loading new data will cause the older data to be evicted, possibly creating a domino effect where the entire data set needs to be loaded for each pass through the loop. By sub-dividing the large data set into smaller blocks and running all of the operations on each block before moving on to the next block, there is a likelihood that the entire block will remain in cache through all the operations.



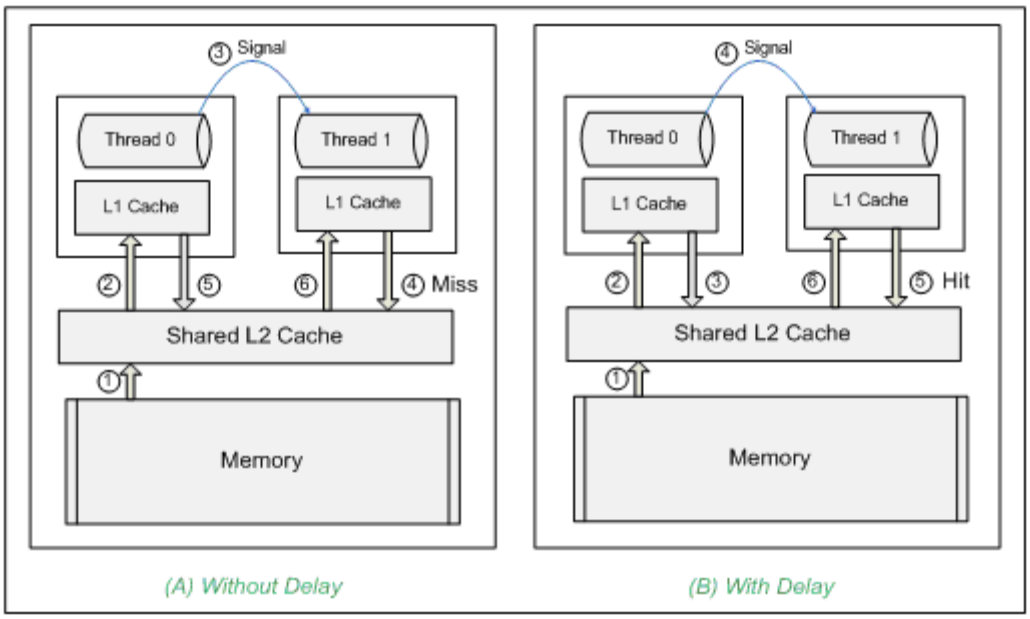
3.) Hold approach:

- The hold approach involves reducing the frequency of access to the shared data by letting each thread maintain its own private copy of data, only updating the shared copy when it is necessary.



4.) Delayed approach:

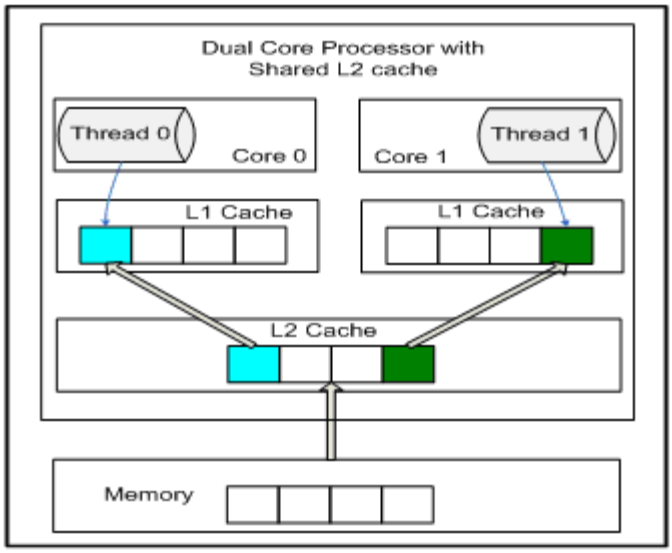
- The delayed approach takes advantage of the natural L1 cache eviction by inserting a wait until data gets pushed to the shared L2 cache – before the other core requests access to it. When two threads are running on two cores that are sharing the L2 cache and the threads are sharing the data, it is possible to implement such a scheme to fine-tune the performance.



In reality, the environment is noisy and there may be multiple contenders for the caches. It may be difficult to get a crisp control of the delay needed. This technique is recommended for the applications so that designers have control on what threads are running, and is used to fine-tune the applications and squeeze out some possible cycles.

5.) Avoid false sharing:

- False sharing is a well-known problem in multiprocessor environments. It happens when threads on different processors write to a shared cache line, but not at the same location. Since processors write to different locations, there is no real coherency problem, however, the cache-coherency protocol marks the cache line dirty, and when the other location gets a read/write request the hardware logic will force a reload of a cache-line update from memory. Frequent updates of the data in the shared-cache line could cause severe performance degradation. As a result, false sharing should be avoided, particularly in areas of high processing. Although the shared-cache architecture has reduced cache-coherency issues at the shared-cache level, the multi-core system overall may still have false sharing issues. For example, to maintain cache coherency in the private L1 cache, false sharing could happen. As illustrated in Figure 5, thread 0 in core 0 brings a cache line to L1 and modifies a portion of it. Thread 1 in core 1 wants to modify a different portion of the same cache line, as well. The hardware coherency logic detects that the cache line has been modified by core 0. As a result, the L1 cache line from core 0 must be evicted to L2 cache, and then loaded into core 1's L1 cache before the update can be completed.



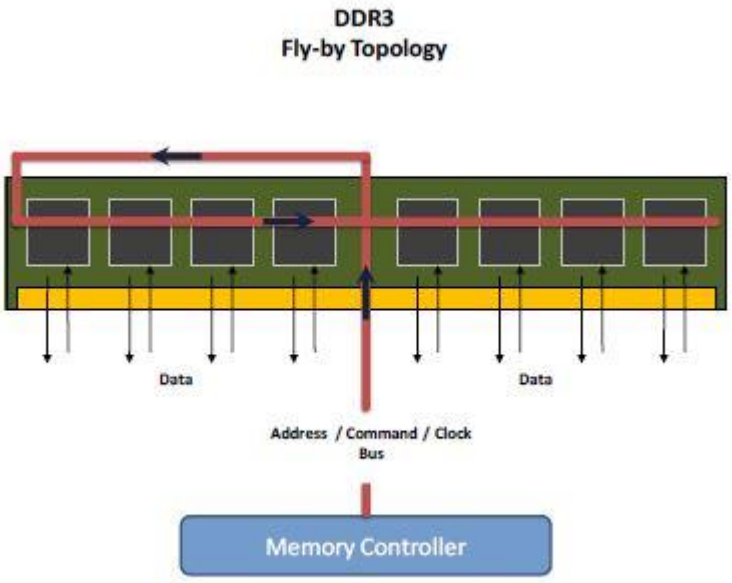
Moreover, in multiprocessor systems, there could be false sharing between cores that are not sharing the cache. For example, in Figure 1, threads running on core 0 will possibly run into false sharing issues with threads running on core 2.

B.) DDR3 architectures (Hewlett Packard):

DDR3 uses the same basic DRAM configuration and architecture as previous DDR implementations. Each DIMM consists of ranks of 9 or 18 DRAMs that deliver 72 bits—64 bits of data and 8 bits of ECC (error correction code)—in parallel to the memory bus to the CPU.

Fly-by topology:

- Fly-by topology solves the problem of the shrinking data-eye by eliminating the need to deliver the data signals simultaneously to each DRAM. With Fly-by topology, each command and address signal flows along a single path that goes from DRAM 0 to DRAM 8. This simpler topology helps increase signal integrity. But it guarantees that the command and address signals won't arrive at each DRAM at the same time. If the signals arrive at DRAM 0 at time N, then they should arrive at DRAM 1 at time N+1, DRAM 2 at time N+2, and so forth. The result is that, on a read, each DRAM presents its data to the memory controller at a slightly different time. Fly-by topology solves the problem of the shrinking data-eye by eliminating the need to deliver the data signals simultaneously to each DRAM. With Fly-by topology, each command and address signal flows along a single path that goes from DRAM 0 to DRAM 8.



On-die Termination:

- Electrical circuits that carry signals need to be terminated with resistors to damp electrical reflections and to improve overall signal integrity. Earlier memory standards had memory termination on the system board. On-die termination puts the resistors on the DRAMs themselves, increasing their effectiveness by placing them at the end of the memory bus circuits. With DDR3 memory, the number of possible termination values is significantly greater than for DDR2. Just as important, the memory controller now empirically sets the termination values during POST based on the configuration of the DIMM module itself (number of ranks) and its position on the memory channel. Both of these refinements contribute to the signal integrity improvements necessary to support the faster DDR3 speeds.

Address parity checking:

- In DDR2, address parity detection was an optional feature. With DDR3, it's now standard. On DDR3 RDIMMs, the register chip performs a parity check on the DRAM address lines and compares it to the parity bit from the memory controller. This process detects potential addressing errors. Although address parity checking cannot correct addressing errors, it does stop the controller from writing data to an incorrect DRAM address, preventing silent data corruption. Unbuffered DIMMs do not support address parity checking because they do not have a register.

System memory bandwidth:

- By removing the front side bus and moving the memory controllers onto the processors, the newer system architectures eliminate some of the previous memory bottlenecks. The maximum theoretical memory bandwidth is unattainable in practice, because it represents an idealized scenario in which all memory channels operate at full throughput all the time. Using NUMA architectures, 2P ProLiant servers can achieve improved measured memory throughput relative to their theoretical maximums.

DDR3 latency:

Memory latency is a measure of the time required for the CPU to receive data from the memory controller once it has requested it. It is an important measurement of memory subsystem responsiveness. Retrieving data from the memory subsystem consists of several steps, each of which consumes time. Taken together, these times comprise the overall latency:

- Time memory request spends in the processor I/O queue and being sent to the memory controller
- Time in the memory controller queue
- Issuing of the Row Address Select (RAS) and Column Address Select (CAS) commands on the memory address bus
- Retrieving data from the memory data bus
- Time through the memory controller and I/O bus back to the requesting processor Arithmetic Logic Unit (ALU)

Maximizing system throughput:

- The key to maximizing system throughput is to populate as many of the system memory channels as possible. This helps to ensure that the memory bandwidth of all channels is available to the system. With 2P ProLiant G6 servers based on the Intel Xeon 5500 series processors, this means installing a minimum of six DIMM modules, one in each memory channel.

Minimizing memory latency:

- You can optimize memory latency, particularly loaded memory latency, by running at the highest data rate. For systems that are capable of supporting the higher data rates, achieving this memory speed depends on the number of and the rank of the DIMMs installed in each channel.

Using balanced memory configurations:

- For nearly all application environments, the optimal configuration for DDR3 memory is to balance installed memory across memory channels and across processors. Balancing installed memory across memory channels on a processor optimizes channel and rank interleaving, ensuring maximum memory throughput.

C.) SmartTrim - The intelligent RAM Optimizer

SmartTrim acts conservatively and with sophisticated selectivity. It doesn't force everything out of memory all at once. It politely asks memory hogging background processes to release their working sets. It has many other criteria built in, such as never trimming the application you're actively engaged with. It is the first truly intelligent RAM optimization algorithm ever conceived.

- Conservative

- Selective
- Throttled paging
- User-controlled

D.) S.M.A.R.T:

S.M.A.R.T. (Self-Monitoring, Analysis and Reporting Technology; often written as SMART) is a monitoring system included in computer hard disk drives (HDDs) and solid-state drives (SSDs) that detects and reports on various indicators of drive reliability, with the intent of enabling the anticipation of hardware failures. Many motherboards display a warning message when a disk drive is approaching failure. Although an industry standard exists among most major hard drive manufacturers, there are some remaining issues and much proprietary "secret knowledge" held by individual manufacturers as to their specific approach. As a result, S.M.A.R.T. is not always implemented correctly on many computer platforms, due to the absence of industry-wide software and hardware standards for S.M.A.R.T. data interchange. Some of the attributes of S.M.A.R.T as follows:

ID	Hex	Attribute name	Better	Description
01	0x01	**Read Error Rate**	▼	(Vendor specific raw value.) Stores data related to the rate of hardware read errors that occurred when reading data from a disk surface. The raw value has different structure for different vendors and is often not meaningful as a decimal number.
02	0x02	**Throughput Performance**	▲	Overall (general) throughput performance of a hard disk drive. If the value of this attribute is decreasing there is a high probability that there is a problem with the disk.
03	0x03	**Spin-Up Time**	▼	Average time of spindle spin up (from zero RPM to fully operational [milliseconds]).
04	0x04	**Start/Stop Count**		A tally of spindle start/stop cycles. The spindle turns on, and hence the count is increased, both when the hard disk is turned on after having before been turned entirely off (disconnected from power source) and when the hard disk returns from having previously been put to sleep mode.[16]
05	0x05	**Reallocated Sectors Count**	▼	Count of reallocated sectors. When the hard drive finds a read/write/verification error, it marks that sector as "reallocated" and transfers data to a special reserved area (spare area). This process is also known as remapping, and reallocated sectors are called "remaps". The raw value normally represents a count of the bad sectors that have been found and remapped. Thus, the higher the attribute value, the more sectors the drive has had to reallocate. This allows a drive with bad sectors to continue operation; however, a drive which has had any reallocations at all is significantly more likely to fail in the near future.[3] While primarily used as a metric of the life expectancy of the drive, this number also affects performance. As the count of reallocated sectors increases, the read/write speed tends to become worse because the drive head is forced to seek to the reserved area whenever a remap is accessed. If sequential access speed is critical, the remapped sectors can be manually marked as bad blocks in the file system in order to prevent their use.
06	0x06	**Read Channel Margin**		Margin of a channel while reading data. The function of this attribute is not specified.

References:

- http://www.seminarsonly.com/computer%20science/Smart-Memories.php
- http://www.pcper.com/reviews/Processors/Detailed-Look-Intels-New-Core-Architecture/Smart-Memory-Access-and-Smart-Cache
- http://www.simmtester.com/page/news/showpubnews.asp?num=149
- http://www.simmtester.com/page/news/showpubnews.asp?num=145
- https://www.google.co.in/url?sa=t&rct=j&q=&esrc=s&source=web&cd=3&cad=rja&uact=8&ved=0CC4QFjAC&url=http%3A%2F%2Fwww.simmtester.com%2Fpage%2Fnews%2Fshowpubnews.asp%3Fnum%3D145&ei=lBNCVf2pEpWUuQTrtYCAAw&usg=AFQjCNGagrif5CYt679Do9ZjWCABMNw0Vg&sig2=gwjgAZy8X3rGmJjJgxTrzg&bvm=bv.92189499,d.c2E
- http://en.wikipedia.org/wiki/Smart_Cache
- http://uk.hardware.info/reviews/850/5/intel-core-micro-architecture-advanced-smart-cache