History of Cache Evolution and Future Trends

Aniket Kumar [ECE], Gurram Mahidhar [CSE],Ishaan Yadav [CSE],K.Vivek Veman [CSE],Rohit Kumar [ECE],Sourabh Singh [ECE],Vishal Babbal [ECE]

Abstract:
During the the early days of microcomputer technology, memory access was only slightly slower than register access. But since the 1980s the performance gap between processor and memory has been growing. Over time Microprocessors have advanced much faster than memory, especially in terms of their operating frequency, so memory became a performance bottleneck. While it was technically possible to have all the main memory as SRAM which could be as fast as the CPU,but a more economically viable path use taken: use plenty of low-speed memory, but also introduce a small high-speed cache memory to close the performance gap.
As CPUs become faster compared to main memory, stalls due to cache misses displace more potential computation; modern CPUs can execute hundreds of instructions in the time taken to fetch a single cache line from main memory.
Cache performance has become important in recent times where the speed gap between the memory performance and the processor performance is increasing exponentially. The cache was introduced to reduce this speed gap. Thus knowing how well the cache is able to bridge the gap in the speed of processor and memory becomes important, especially in high-performance systems. The cache hit rate and the cache miss rate play an important role in determining this performance. It is also important to see how these developments were made and what factors influence the design of caches in current computers.
Early cache designs focused entirely on the direct cost of using various cache designs on execution speed and efficiency in reducing the gap between processor and RAM . More recent cache designs also consider energy efficiency, fault tolerance, and goals.
We plan to attempt a thorough study of various trends and developments made since early days of introduction of cache memory to what we have today and what we might expect in the future based on current research and areas where room for improvement is present.

Keywords: Cache Performance, Cache Design,Cache enhancements,Cache optimizations,Cache Hierarchy.

History of Cache Memory

Aniket kumar

October 2019

1 Definition of Cache Memory

The cache is a very high speed, expensive piece of memory, which is used to speed up the memory retrieval process. Due to it's higher cost, the CPU comes with a relatively small amount of cache compared with the main memory.Without cache memory, every time the CPU requests for data, it would send the request to main memory which would then be sent back across the system bus to the CPU. This is a slow process. The idea of introducing cache is that this extremely fast memory would store data that is frequently accessed and if possible, the data that is around it. This is to achieve the quickest possible response time to the CPU.

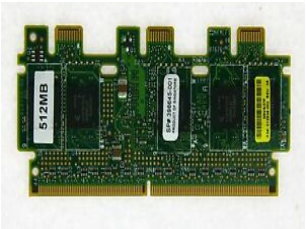


Figure 1: HP 512 DDR2 Cache Memory

2 Earlier Cache memory

Memory cache was first used on PCs at the 386DX timeframe. Even though the CPU itself didn't have memory cache inside, its support circuitry – i.e., the chipset – had a memory cache controller. Thus, the memory cache at this time was external to the CPU and thus was optional, i.e., the motherboard manufacturer could add it or not. If you had a motherboard without memory cache your PC would be far slower than a PC with this circuit. The amount of available memory cache varied as well depending on the motherboard model and typical values for that time were 64 KB and 128 KB. At this time the memory cache controller used an architecture known as "write-through," where

for write operations – i.e., when the CPU wants to store data in memory – the memory cache controller updates the RAM memory immediately. With the 486DX processor Intel added a small amount (8 KB) of memory cache inside the CPU. This internal memory cache was called L1 (level 1) or "internal," while the external memory cache was called L2 (level 2) or "external." The amount and existence of the external memory cache depended on the motherboard model. Typical amounts for that time were 128 KB and 256 KB. Later 486 models added the "write back" cache architecture, which is used until today, where for write operations the RAM memory isn't updated immediately, the CPU stores the data on the cache memory and the memory controller updates the RAM memory only when a cache miss occurs. Then with the first Pentium processor Intel created two separated internal memory caches, one for instructions and another for data (at the time with 8 KB each). This architecture is still used to date, and that is why you sometimes see the L1 memory cache being referred as 64 KB + 64 KB, for example – this is because there are one 64 KB instruction L1 cache and one 64 KB data L1 cache. Of course we will explain later what is the difference between the two. At that time the L2 memory cache continued to be located on the motherboard, so its amount and existence depended on the motherboard model. Of course having a system without memory cache was insane. Typical amounts for that time were 256 KB and 512 KB. On AMD side K5, K6 and K6-2 processors used this same architecture, with K6-III having a third memory cache (L3, level 3). The problem with the L2 memory cache being

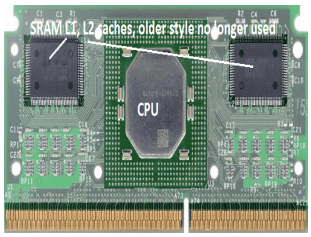


Figure 2: SRAM L1 and L2 caches

external is that it is accessed with a lower clock rate, because since 486DX2 the CPU internal clock rate is different from the CPU external clock rate. While a Pentium-200 worked internally at 200 MHz, it accessed its L2 memory cache at 66 MHz, for example. Then with P6 architecture Intel moved the memory cache from the motherboard to inside the CPU – what allowed the CPU to access it with its internal clock rate –, except on Pentium II, where the memory cache was not located inside the CPU but on the same printed circuit board where the CPU was soldered to (this printed circuit board was located inside a cartridge), running at half the CPU internal clock rate, and on Celeron-266 and Celeron-300, which had no memory cache at all (and thus they were the worst-performing CPUs in PC history). This same architecture is used until today: both L1 and L2 memory caches are located inside the CPU running at the CPU internal clock rate.

3 Conclusion

"The amount of memory cache you can have on your system will depend on the CPU model you have; there is no way to increase the amount of memory cache without replacing the CPU."

EARLY CACHE IMPROVEMENTS

A PREPRINT

Ishaan Yadav*
18114032
Department of Computer Science
Indian Institute of Technology, Roorkee

October 31, 2019

1 Background

In the history of computer and electronic chip development, there was a period when increases in CPU speed outpaced the improvements in memory access speed. The gap between the speed of CPUs and memory meant that the CPU would often be idle. CPUs were increasingly capable of running and executing larger amounts of instructions in a given time, but the time needed to access data from main memory prevented programs from fully benefiting from this capability. This issue motivated the creation of memory models with higher access rates in order to realize the potential of faster processors.

This resulted in the concept of cache memory, first proposed by Maurice Wilkes, a British computer scientist at the University of Cambridge in 1965. He called such memory models "slave memory".Between roughly 1970 and 1990, papers and articles by Anant Agarwal, Alan Jay Smith, Mark D. Hill, Thomas R. Puzak, and others discussed better cache memory designs. The first cache memory models were implemented at the time, but even as researchers were investigating and proposing better designs, the need for faster memory models continued. This need resulted from the fact that although early cache models improved data access latency, with respect to cost and technical limitations it was not feasible for a computer system's cache to approach the size of main memory. From 1990 onward, ideas such as adding another cache level (second-level), as a backup for the first-level cache were proposed. Jean-Loup Baer, Wen-Hann Wang, Andrew W. Wilson, and others have conducted research on this model. When several simulations and implementations demonstrated the advantages of two-level cache models, the concept of multi-level caches caught on as a new and generally better model of cache memories.Apart from this the concept of separate data and instruction L1 cache was introduced.

2 The i386 , i486 and Pentium P5 microprocessor caches

2.1 i386

The 80386 was introduced in October 1985. It was the first 32bit CPU introduced by Intel and it also marked the first time intel supported cache memory on any of its chips. The i386 processor although didn't have any on-chip cache it did contained support for an external cache of 16 to 64 KB. As all other on-board caches this wasn't a much faster implementation as on-board caches were only slightly faster than DRAMs' and their limited sizes didn't bring much performance to the table.

2.2 i486

The Intel 80486, also known as the i486 or 486, is the successor model of 32-bit x86 microprocessor to the Intel 80386. Introduced in 1989, the 80486 improved on the performance of the i386 thanks to on-die L1 cache and floating-point unit, as well as an improved, five-stage tightly-coupled pipelined design. It was the first x86 chip to use more than a

*Use footnote for providing further information about author (webpage, alternative address)—*not* for acknowledging funding agencies.

million transistors From a performance point of view, the architecture of the i486 is a vast improvement over the 80386. It has an on-chip unified instruction and data cache, an on-chip floating-point unit (FPU) and an enhanced bus interface unit. Due to the tight pipelining, sequences of simple instructions (such as ALU reg,reg and ALU reg,im) could sustain a single clock cycle throughput (one instruction completed every clock). These improvements yielded a rough doubling in integer ALU performance over the 386 at the same clock rate.

2.2.1 Differences between the i386 and i486 caches

- An 8 KB on-chip (level 1) SRAM cache stores the most recently used instructions and data (16 KB and/or write-back on some later models). The 386 had no such internal cache but supported a slower off-chip cache (which was not a level 2 cache because there was no internal level 1 cache on the 80386).
- An enhanced external bus protocol to enable cache coherency and a new burst mode for memory accesses to fill a cache of 16 bytes within 5 bus cycles. The 386 needed 8 bus cycles to transfer the same amount of data

2.2.2 Models

	i486DX (P4)	20, 25 MHz 33 MHz 50 MHz	5 V	8 KB WT	April 1989 May 1990 June 1991
	i486SL	20, 25, 33 MHz	5 V or 3.3 V	8 KB WT	November 1992
	i486SX (P23)	16, 20, 25 MHz 33 MHz	5 V	8 KB WT	September 1991 September 1992
	i486DX2 (P24)	40/20, 50/25 MHz 66/33 MHz	5 V	8 KB WT	March 1992 August 1992
	i486DX-S (P4S)	33 MHz; 50 MHz	5 V or 3.3 V	8 KB WT	June 1993

*WT = write-through cache strategy, WB = write-back cache strategy

2.3 P5 (micro architecture)

The first Pentium microprocessor was introduced by Intel on March 22, 1993.[2][3] Its P5 microarchitecture was the fifth generation for Intel, and the first superscalar IA-32 microarchitecture. As a direct extension of the 80486 architecture, it included dual integer pipelines, a faster floating-point unit, wider data bus, separate code and data caches and features for further reduced address calculation latency. In October 1996, the Pentium with MMX Technology (often simply referred to as Pentium MMX) was introduced, complementing the same basic microarchitecture with the MMX instruction set, larger caches, and some other enhancements. The P5 microarchitecture was designed by the same Santa Clara team which designed the 386 and 486.[6] Design work started in 1989;[7] the team decided to use a superscalar architecture, with on-chip cache, floating-point, and branch prediction

2.3.1 Major improvements over the 80486 microarchitecture

- Superscalar architecture — The Pentium has two data paths (pipelines) that allow it to complete two instructions per clock cycle in many cases. The main pipe (U) can handle any instruction, while the other (V) can handle the most common simple instructions. Some[who?] RISC proponents had argued that the "complicated" x86 instruction set would probably never be implemented by a tightly pipe-lined micro architecture, much less by a dual-pipeline design. The 486 and the Pentium demonstrated that this was indeed possible and feasible.
- 64-bit external databus doubles the amount of information possible to read or write on each memory access and therefore allows the Pentium to load its code cache faster than the 80486; it also allows faster access and storage of 64-bit and 80-bit x87 FPU data.
- Separation of code and data caches lessens the fetch and operand read/write conflicts compared to the 486. To reduce access time and implementation cost, both of them are 2-way associative, instead of the single 4-way cache of the 486. A related enhancement in the Pentium is the ability to read a contiguous block from the code cache even when it is split between two cache lines
- Enhanced self-test features like the L1 cache parity check
- The later Pentium MMX also added the MMX instruction set, a basic integer SIMD instruction set extension marketed for use in multimedia applications. MMX could not be used simultaneously with the x87 FPU instructions because the registers were reused (to allow fast context switches).
- More important enhancements were the doubling of the instruction and data cache sizes and a few microarchitectural changes for better performance.

References

[1] Intel (July 1997) Embedded Intel486 Processor Hardware Reference Manual (273025-001)

[2] Pryce, Dave 486 32-bit CPU breaks new ground in chip density and operating performance. (Intel Corp.) (product announcement) EDN | May 11, 1989 |

[3] https://en.wikichip.org/wiki/intel/microarchitectures/p5

[4] https://en.wikipedia.org/wiki/Intel$_8$0386

[5] https://en.wikipedia.org/wiki/Intel$_8$0486

[6] "Product Change Notification 777" (PDF). Intel. February 9, 1999. Archived from the original (PDF) on January 27, 2000. Retrieved October 14, 2019.

[7] View Processors Chronologically by Date of Introduction, Intel, retrieved August 14, 2007

[8] Intel Pentium Processor Family, Intel, retrieved August 14, 2007

[9] D. Alpert and D. Avnon p. 21, "Architecture of the Pentium microprocessor", , IEEE Micro, 13, 3 (June 1993), pp. 11–21, doi:10.1109/40.216745.

Cache Evolution

Sourabh Singh

October 2019

1 Current Cache Design

In computing, a cache is a hardware or software component that stores data so that future requests for that data can be served faster; the data stored in a cache might be the result of an earlier computation or a copy of data stored elsewhere. A cache hit occurs when the requested data can be found in a cache, while a cache miss occurs when it cannot. Cache hits are served by reading data from the cache, which is faster than recomputing a result or reading from a slower data store; thus, the more requests that can be served from the cache, the faster the system performs.

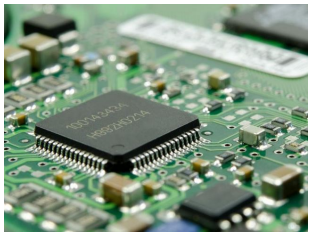


Figure 1: Cache Memory

To be cost-effective and to enable efficient use of data, caches must be relatively small. Nevertheless, caches have proven themselves in many areas of computing, because typical computer applications access data with a high degree of locality of reference. Such access patterns exhibit temporal locality, where data is requested that has been recently requested already, and spatial locality, where data is requested that is stored physically close to data that has already been requested.

Caching configurations continue to evolve, but cache memory traditionally works under three different configurations:

Direct mapped cache has each block mapped to exactly one cache memory location. Conceptually, direct mapped cache is like rows in a table with three columns: the data block or cache line that contains the actual data fetched and stored, a tag with all or part of the address of the data that was fetched,

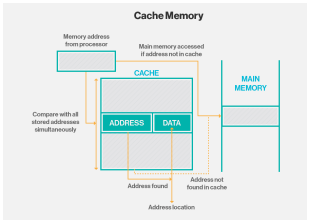


Figure 2: Working of Cache Memory

and a flag bit that shows the presence in the row entry of a valid bit of data. Fully associative cache mapping is similar to direct mapping in structure but allows a block to be mapped to any cache location rather than to a prespecified cache memory location as is the case with direct mapping. Set associative cache mapping can be viewed as a compromise between direct mapping and fully associative mapping in which each block is mapped to a subset of cache locations. It is sometimes called N-way set associative mapping, which provides for a location in main memory to be cached to any of "N" locations in the L1 cache.

The cache stores the frequently used data by the CPU. The CPU first checks the cache for the required data. Even though the RAM is fast, it is not as fast as the cache. Therefore, storing the commonly required data in the cache is beneficial to increase the computation speed.

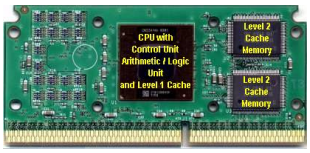


Figure 3: Levels of Cache Memory

There are three types of cache. The level 1 cache is the smallest. It is located inside the CPU or the processor. So, it runs at the same speed as the CPU. Level 2 and level 3 caches are external. Level 2 cache is larger than level 1 cache. If the required data is not available in level 1 cache, the CPU checks the level 2 cache. If the required data is not available in both level 1 and level 2 caches, the CPU checks the level 3 cache. If the required data is not available in any of these caches, the CPU will access the RAM. Level 1 cache is the fastest cache of all. A CPU can have multiple cores. A core is the execution unit of the CPU. Each core can have separate level 1 and level 2 caches. The level 3 cache is shared among all cores.In the early days of microcomputer technology, memory access was only slightly slower than register access. But since the 1980s the performance gap between processor and memory has been growing. Microprocessors have advanced much faster than memory, especially in terms of their operating frequency, so memory became a performance bottleneck.

While it was technically possible to have all the main memory as fast as the CPU, a more economically viable path has been taken: use plenty of low-speed memory, but also introduce a small high-speed cache memory to alleviate the performance gap. This provided an order of magnitude more capacity—for the same price—with only a slightly reduced combined performance. Early cache designs focused entirely on the direct cost of cache and RAM and average execution speed. More recent cache designs also consider energy efficiency,fault tolerance, and other goals. Researchers have also explored use of emerging memory technologies such as eDRAM (embedded DRAM) and NVRAM (non-volatile RAM) for designing caches.There are several tools available to computer architects to help explore tradeoffs between the cache cycle time, energy, and area. These tools include the open-source CACTI cache simulator and the open-source SimpleScalar instruction set simulator. Modeling of 2D and 3D SRAM,eDRAM, STT-RAM, ReRAM and PCM caches can be done using the DESTINY tool.

A more modern cache might be 16 KB, 4-way set-associative, virtually indexed, virtually hinted, and physically tagged, with 32 B lines, 32-bit read width and 36-bit physical addresses. The read path recurrence for such a cache looks very similar to the path above. Instead of tags, vhints are read, and matched against a subset of the virtual address. Later on in the pipeline, the virtual address is translated into a physical address by the TLB, and the physical tag is read (just one, as the vhint supplies which way of the cache to read). Finally the physical address is compared to the physical tag to determine if a hit has occurred.Current Cache designs are susceptible to cache based attacks.Caches should have low miss rates and short access times and should be power efficient at the same time.

2 Reference

https://searchstorage.techtarget.com
https://www.cbronline.com
https://www.slideshare.net
https://pediaa.com

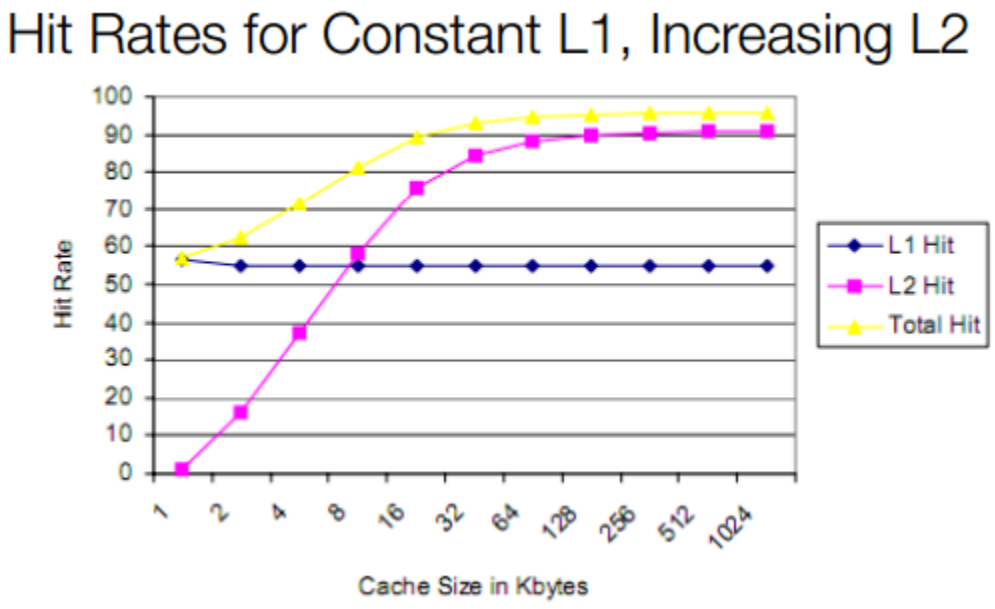
Cache Evolution

G. MAHIDHAR
October 31, 2019

1 CURRENT CACHE DESIGN :

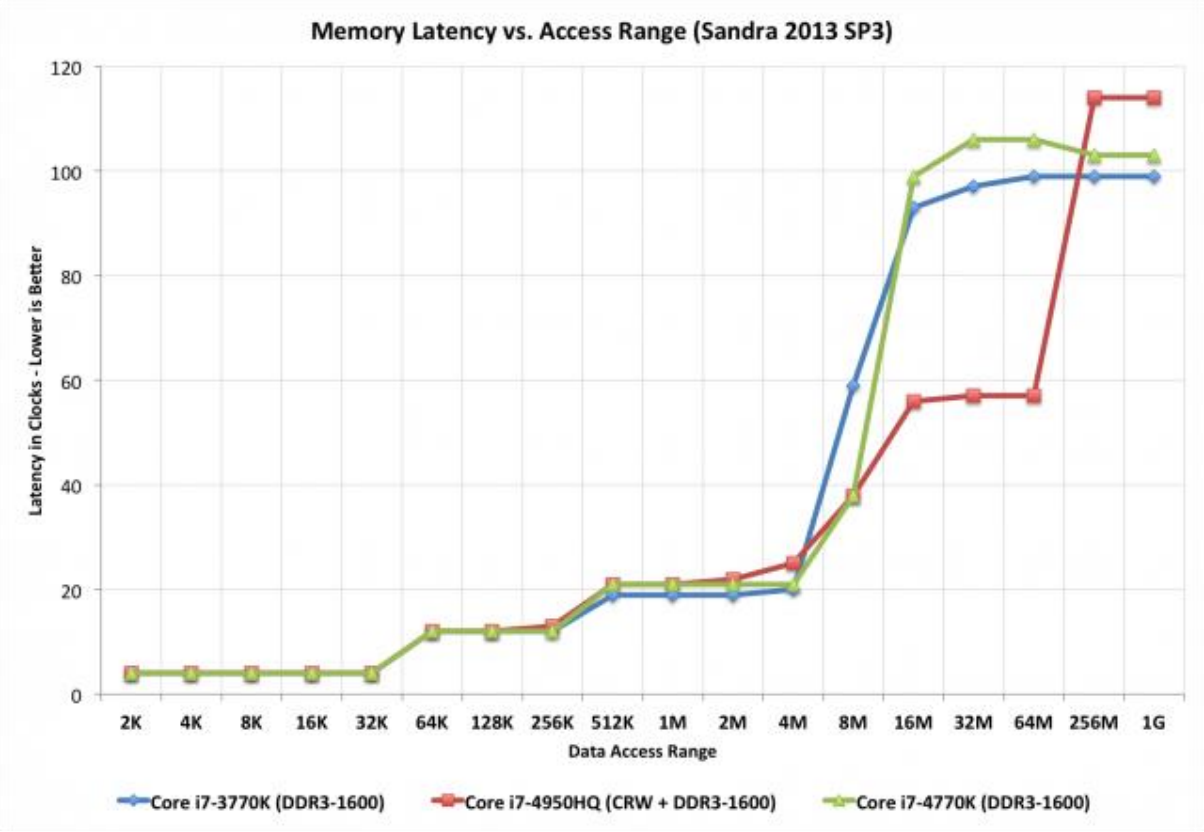
Almost all current CPUs with caches have a split L1 cache. They also have L2 caches and, for larger processors, L3 caches as well. The L2 cache is usually not split and acts as a common repository for the already split L1 cache. Every core of a multi-core processor has a dedicated L1 cache and is usually not shared between the cores. The L2 cache, and higher-level caches, may be shared between the cores. L4 cache is currently uncommon, and is generally on (a form of) dynamic random-access memory (DRAM), rather than on static random-access memory (SRAM), on a separate die or chip (exceptionally, the form, eDRAM is used for all levels of cache, down to L1).

The goal of the cache system is to ensure that the CPU has the next bit of data it will need already loaded into cache by the time it goes looking for it (also called a cache hit).A cache miss, on the other hand, means the CPU has to go scampering off to find the data elsewhere. This is where the L2 cache comes into play — while it's slower, it's also much larger. Some processors use an inclusive cache design (meaning data stored in the L1 cache is also duplicated in the L2 cache) while others are exclusive (meaning the two caches never share data). If data can't be found in the L2 cache, the CPU continues down the chain to L3 (typically still on-die), then L4 (if it exists) and main memory (DRAM).



This chart shows the relationship between an L1 cache with a constant hit rate, but a larger L2 cache. Note that the total hit rate goes up sharply as the size of the L2 increases. A larger, slower, cheaper L2 can provide all the benefits of a large L1 — but without the die size and power consumption penalty. Most modern L1 cache rates have hit rates far above the theoretical 50 percent shown here — Intel and AMD both typically field cache hit rates of 95 percent or higher.

The CPU caches keep getting bigger ,this is because each additional memory pool pushes back the need to access main memory and can improve performance in specific cases.



This chart from Anandtech's Haswell review is useful because it actually illustrates the performance impact of adding a huge (128MB) L4 cache as well as the conventional L1/L2/L3 structures. Each stair step represents a new level of cache. The red line is the chip with an L4 — note that for large file sizes, it's still almost twice as fast as the other two Intel chips.

It might seem logical, then, to devote huge amounts of on-die resources to cache — but it turns out there's a diminishing marginal return to doing so. Larger caches are both slower and more expensive. At six transistors per bit of SRAM (6T), cache is also expensive (in terms of die size, and therefore dollar cost). Past a certain point, it makes more sense to spend the chip's power budget and transistor count on more execution units, better branch prediction, or additional cores. At the top of the story you can see an image of the Pentium M (Centrino/Dothan) chip; the entire left side of the die is dedicated to a massive L2 cache.

As the latency difference between main memory and the fastest cache has become larger, some processors have begun to utilize as many as three levels of on-chip cache. Price-sensitive designs used this to pull the entire cache hierarchy on-chip, but by the 2010s some of the highest-performance designs returned to having large off-chip caches, which is often implemented in eDRAM and mounted on a multi-chip module, as a fourth cache level. In rare cases, as in latest IBM mainframe CPU, IBM z15 from 2019, all levels down to L1 are implemented by eDRAM, replacing SRAM entirely (for caches, i.e. it's still used for registers) for 128 KiB L1 for instructions and for data, or combined 256 KiB.

The benefits of L3 and L4 caches depend on the application's access patterns. Examples of products incorporating L3 and L4 caches include the following:

- Alpha 21164 (1995) has 1 to 64 MB off-chip L3 cache.
- IBM POWER4 (2001) has off-chip L3 caches of 32 MB per processor, shared among several processors.
- Itanium 2 (2003) has a 6 MB unified level 3 (L3) cache on-die; the Itanium 2 (2003) MX 2 module incorporates two Itanium 2 processors along with a shared 64 MB L4 cache on a multi-chip module that was pin compatible with a Madison processor.
- Intel's Xeon MP product codenamed "Tulsa" (2006) features 16 MB of on-die L3 cache shared between two processor cores.
- AMD Phenom II (2008) has up to 6 MB on-die unified L3 cache.
- Intel Core i7 (2008) has an 8 MB on-die unified L3 cache that is inclusive, shared by all cores.
- Intel Haswell CPUs with integrated Intel Iris Pro Graphics have 128 MB of eDRAM acting essentially as an L4 cache.[38]

Finally, at the other end of the memory hierarchy, the CPU register file itself can be considered the smallest, fastest cache in the system, with the special characteristic that it is scheduled in software—typically by a compiler, as it allocates registers to hold values retrieved from main memory for, as an example, loop nest optimization. However, with register renaming most compiler register assignments are reallocated dynamically by hardware at runtime into a register bank, allowing the CPU to break false data dependencies and thus easing pipeline hazards.

Register files sometimes also have hierarchy: The Cray-1 (circa 1976) had eight address "A" and eight scalar data "S" registers that were generally usable. There was also a set of 64 address "B" and 64 scalar data "T" registers that took longer to access, but were faster than main memory. The "B" and "T" registers were provided because the Cray-1 did not have a data cache. (The Cray-1 did, however, have an instruction cache.)

2 Reference:

http://highscalability.com/blog/

https://www.extremetech.com

future cache design:STT MRAMs

k.vivek veman

October 2019

Spin-transfer torque magnetic RAM (STT MRAM) has emerged as a promising candidate for on-chip memory in future computing platforms. We present a cross-layer (device-circuit-architecture) approach to energy-efficient cache design using STT MRAM

1 INTRODUCTION

• The ever-increasing gap between processor speed and main memory latency has driven the demand for larger on-chip caches in processors. • Traditionally, on-chip caches in modern processors are implemented using static random access memories (SRAM). • However, limited scalability, susceptibility to soft errors and high leakage power of SRAM pose challenges to high-density on-chip cache implementation. In order to address the limited scalability of SRAMs, several recent processors have adopted embedded dynamic RAM (EDRAM) in lower level caches. However, vulnerability to soft errors and significant standby power of EDRAM caches due to high cell leakage are still major bottlenecks in on-chip cache design • STT MRAMs compatibility with CMOS processes makes it an attractive vehicle to realize high-density low-power embedded memories in scaled technologies

STT MRAM CACHE DESIGN

1.1 STT MRAM Preliminaries

• A conventional STT MRAM cell comprises of a magnetic tunnel junction (MTJ) and an access transistor in series (Figure 1 (a-b)). The MTJ contains a pinned layer and a free layer separated by a dielectric layer (e.g. MgO). The pinned layer has a fixed magnetization, and the free layer is programmable by changing its magnetic orientation. The resistance of the MTJ depends on the relative magnetization of the free layer with respect to the pinned layer. Parallel magnetization of the free layer with respect to the pinned layer leads to a lower resistance (RP) compared to the resistance in the anti-parallel state (RAP). The two resistances of the MTJ define the binary states of the memory cell. A read operation is performed by sensing resistance difference of the two binary states. A write operation is performed by passing a current

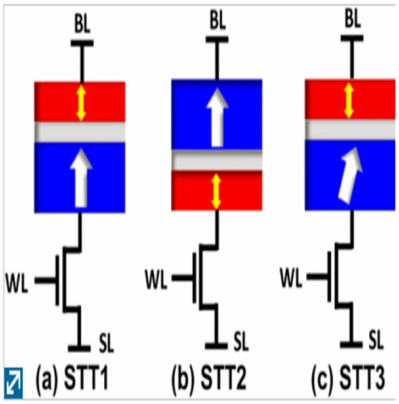


Figure 1: Schematics of an STT MRAM bitcell (a) in the standard-connected configuration (b) in the reverse-connected configuration and (c) with tilted magnetic anisotropy

(IW) through the bitcell that exceeds a critical current (IC). The direction of (IW) determines the final magnetization of the free layer (i.e., parallel or anti-parallel states of the MTJ)

1.2 STT MRAM Bitcell Design: Devices and Circuits

• Different types of MTJ stacks , and bitcell configurations provide several design choices, and can result in substantially different bitcell characteristics. Before exploring these choices, we first discuss design considerations of STT MRAM bitcells. A conventional MTJ has a large switching current density requirement, and the requirement increases dramatically with lower switching delay . The large switching current requirement for fast write operation is one of the major challenges for energy-efficient STT MRAM design. In order to address the excessive switching current requirement, an MTJ with tilted magnetic anisotropy (TMA) has been proposed in . Tilting the direction of the pinned layer, by a larger angle than what stochastic thermal noise can provide, leads to a thermal-noise-independent non-zero initial angle for precessional switching. As a result, the switching current overdrive and switching delay can be reduced significantly

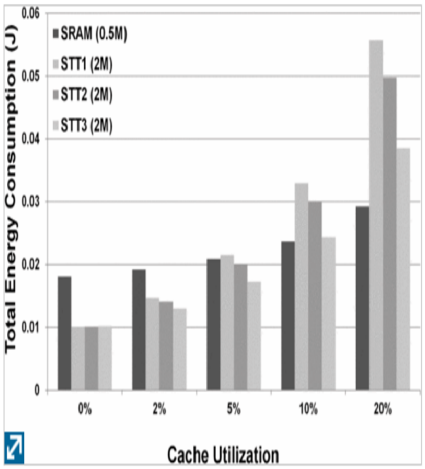


Figure 3: Total energy consumption of L2 caches at iso-area (0.5MB SRAM vs. 2MB STT MRAM)

1.3 Cache Utilization and Energy Consumption

• The contribution of active and leakage energy to total energy consumption is different for SRAM- and STT MRAM-based caches. The leakage energy in an STT MRAM cache is smaller than an SRAM cache even with 4 times larger capacity (at iso-area). On the other hand, the dynamic energy for a write operation is higher in an STT MRAM cache compared to an SRAM cache. It is important to note that the total energy dissipation in a cache depends on factors such as cache access patterns (number of read and write operations) and cache utilization (number of times a processor accesses the cache per unit cycle). The cache utilization is lower than 30 in today's processors . Moreover, for lower levels of the cache hierarchy, the cache utilization is significantly lower than 30 percentage . We have measured L2 cache utilizations for various SPEC2000 benchmarks based on the Simplescalar framework with a 32KB L1 cache configuration. Our simulation results also confirm low L2 cache utilization. For a majority of the benchmarks, L2 cache utilization is lower than 3. The highest utilization, observed for the AMMP benchmark, is about 13, and the average utilization across 16 benchmarks is only 2.2.

2 ENERGY-EFFICIENT STT MRAM CACHE DESIGN

2.1 Column Selection: SRAM vs. STT MRAM

• Since set associativity is common in modern caches, column selection in SRAM arrays is imperative. Furthermore, bit-interleaving can only be achieved by employing column selection. Bit-interleaving is a commonly adopted technique in SRAM arrays to mitigate soft errors to increase array density by bitline multiplexing . In the column selection operation of an SRAM array, all unselected bitcells in the accessed row have to be under read mode to prevent unexpected bit flips, when a wordline is asserted. This phenomenon is commonly known as pseudo read or half selection . Note that, in an STT MRAM array, the nonvolatility of bitcells can eliminate the half selection problem.

2.2 Read Energy Reduction in STT MRAM Cache

• One challenge to enable energy-efficient column selection can be to identify the selected column address during cache read operation with minimal performance penalty. We observed that the proposed technique can be easily adopted in a cache implementing sequential tag-data access. Sequential tag-data access is often employed in large, lower-level caches to improve energy-efficiency during operation . In sequential tag-data access, a cache probes the tag array first, and identifies a hit or miss. Access to the data array occurs only when there is a cache hit, and only the sub-array storing the corresponding cache line in the data array is accessed. As a result, significant energy savings can be achieved.

2.3 Write Energy Reduction in STT MRAM Cache

• Similar to the read energy reduction technique described above, improvement in write energy efficiency of STT MRAM cache can also be achieved by exploiting half-selection-free column selection. We propose partial cache line update (PLU) to reduce cache writeback energy consumption. This technique exploits data redundancy in a multi-level cache hierarchy as well as non-volatility of STT MRAM bitcells. In a writeback cache, writeback is performed when a dirty cache line in the L1 cache needs to be replaced by a new cache line. Hence, the dirty line has to be written into the L2 cache.
• Figure 6 presents the proposed PLU STT MRAM cache architecture. Each cache line is partitioned into n partial lines in order to utilize the energy-efficient column selection of STT MRAM arrays (n=4 in the given example). During writeback from the SRAM L1 cache, only the partitions in the cache line that have been updated by the processor (1 out of 4 partitions in the example) are written to the STT MRAM L2 data array. The data in the remaining partitions are identical to the data already stored in the L2 cache.

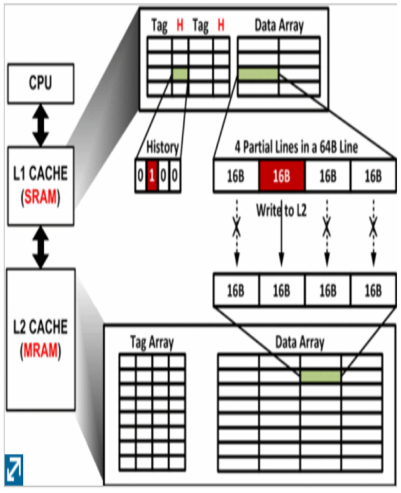


Figure 6: Partial cache line update

Therefore, writing the unchanged partitions into the L2 cache is unnecessary. The change of partitions can be tracked by using a history bit per partition. In the given example, 4 history bits to support 4 partitions are added into each tag in the L1 SRAM cache. The history data is used and reset whenever the corresponding cache line is written back into the L2 STT MRAM cache.

2.4 Total L2 Energy Consumption

• In order to analyze the energy efficiency of STT MRAM caches in comparison to SRAM caches, we measured the total energy consumption of L2 cache including leakage, read and write energy over 1 billion cycles of processor execution. The results presented in Figure 8 are obtained by averaging L2 cache energy consumption across 8 integer and 8 floating point benchmarks. SRAM-based L2 cache shows the largest energy consumption compared to STT MRAM caches with the same capacity, due to the significant leakage energy of SRAM bitcells. The energy difference is further improved for larger cache capacities. Moreover, under iso-area comparison (e.g., 0.5MB SRAM and 2MB STT MRAM caches), STT MRAM caches show significant energy benefit along with larger cache capacity (note that larger capacity improves processor performance by lowering cache misses). Our results show that a processor with

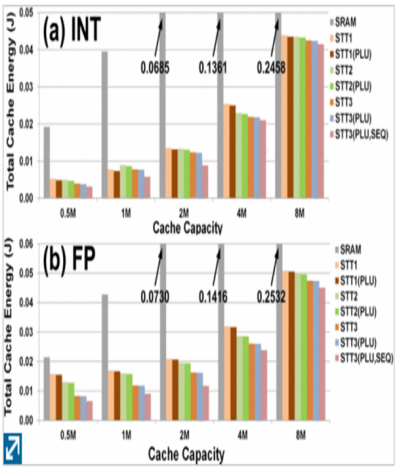


Figure 8: Total energy consumption of SRAM and STT MRAM L2 caches

2MB STT MRAM L2 outperforms one with 0.5MB SRAM L2 by 10 in IPC.

3 Conclusion

•In this work, they performed a comprehensive analysis of the performance, energy consumption and integration density of STT MRAM caches in comparison to conventional SRAM cache. they considered different genres of MTJ stacks and STT MRAM bitcell configurations in this study. Based on the detailed analysis of various bitcell characteristics including accurate area estimation from physical layout, they showed that, for large cache capacity, STT MRAM caches can have lower dynamic energy consumption and read latency compared to SRAM caches with the same capacity. Moreover, the low leakage energy consumption and high integration density of STT MRAM are highly beneficial for lower level caches (due to low utilization), and improve energy efficiency and processor performance. they also proposed read and write energy reduction techniques, namely sequential tag-data access in reads and partial cache line update in writes, which exploit the non-volatility of STT MRAM bitcells. The results show that the proposed techniques further improve the energy efficiency of STT MRAM caches.
Reference: source:ieee authors: sang phill park and sumeet gupta and niladri