Contents

1 Description

In recent years, increases in memory subsystem speed have not kept pace with the increase in processor speed, causing processor execution rates to become increasingly limited by the latency of accessing instructions and data. On-chip caches are a popular technique to combat this speed mismatch. As integrated circuits become denser, designers have more chip area that can be devoted to on-chip caches. Straight-forward scaling of cache sizes as the available area in- creases, however, may not be the best solution, since the larger a cache, the larger its access time. Using cache hierarchies (two or more levels) is a potential solution. This paper explores the tradeoffs in the design of on-chip microprocessor caches for a range of available on-chip cache areas.

There are a number of potential advantages of two-level on-chip caching with a mixed (in- struction and data) second-level cache over single-level on-chip caching. First, the primary cache (also referred to as the L1 cache) usually needs to be split into separate instruction and data caches to support the instruction and data fetch bandwidths of modern processors. By having a two-level hierarchy on-chip where the majority of the cache capacity is in a mixed second-level cache (L2 cache), cache lines are dynamically al- located to contain data or instructions depending on the program's requirements, as opposed to living with a static partition given by single-level on-chip cache sizes chosen at design time.

A second and more important potential advantage of two-level on-chip caching is an improve- ment in cache access time. As existing processors with single-level on-chip caching are shrunk to smaller lithographic feature sizes. If the additional area available due to a process shrink is used to simply extend the first-level cache sizes, the caches will get slower relative to the processor datapath. Instead, if the extra area is used to hold a second-level cache, the primary caches can scale in access time along with the datapath, while additional cache capacity is still added on-chip.

A third potential advantage of two-level cache structures is that the second-level cache can be made set-associative while keeping the primary caches direct-mapped. This keeps the fast primary access time of direct-mapped caches, but reduces the penalty of first-level conflict misses since many of these can be satisfied from an on-chip set-associative cache instead of requiring an off-chip access.

When primary cache sizes are less than or equal to the page size, address translation can easily occur in parallel with a cache access. However, most modern machines have minimum page sizes of between 4KB and 8KB. This is smaller than most on-chip caches. By using two-level on-chip caching, the primary caches can be made less than or equal to the page size, with the remaining on-chip memory capacity being devoted to the second-level cache. This allows the address translation and first-level cache access to occur in parallel. This is a fourth potential advantage of two-level cache structures.

A fifth advantage of two-level cache structures is that a chip with a two-level cache will usually use less power that one with a single-level organization (assuming the area devoted to the cache is the same). In a single-level configuration, wordlines and bitlines are longer, meaning there is a larger capacitance that needs to be charged or discharged with every cache access. In a two-level configuration, most accesses only require an access to a small first-level cache.

2 Why topic is interesting and important ?

Several trends in current technology have increased the vi- ability of a two-level cache, for example:

- Technology is yielding impressive reductions in processor speed, but memory speed has not kept pace. Therefore, the disparity between processor and memory speed is in- creasing.
- The appetite for memory is rapidly increasing and pri- mary memories of 100s to 1000s of megabytes will be common. Larger memories typically have larger access times because of packaging and physical constraints.
- As on-chip densities increase, it becomes possible to in- clude small on-chip instruction and/or data caches. These caches will need to be backed up by larger second-level caches. As one example, the MicroVAX 3500 has a 1K on-chip cache and a 64K second-level cache.
- Shared memory multiprocessors require local caches to reduce bus contention. A second-level global cache can help to further reduce access time on cache misses.

Therefore, from the above points it sums up that processors have more than one level cache in order to increase the capacity of the processor cache without also dramatically increasing the price of the processor. This careful mixture allows for processors that are faster and cheaper.

Cache memory is very useful because it saves the computer user a lot of time in opening common data and giving common commands. Cache memory is very convinient because opening data in cache is dramatically faster than opening data in the main hard disk. Because of this utter importance of cache it is really necessary to discuss on this topic and bring out some revolutionary improvement in this area.

3 Description of Related Work

3.1 Reference

http://www.bitsavers.org/pdf/dec/tech$_reports/WRL-93-3.pdf$
https://dl.acm.org/citation.cfm?doid=633625.52410

4 Description of method used to evaluate the proposed topic

For this study we chose to examine several system organizations,including

- a baseline system with a single cache
- a system with a two-level cache, and
- a two-level system with separate local instruction and data caches.

We refer to the processor-local cache as the L1 cache and the second-level cache as the L2 cache. At a high level, the components of this system operates in the following way;

- The instruction fetch unit contains an instruction register and a prefetch buffer into which the next instruction is read while the current instruction is executing. When the prefetch buffer is empty, the instruction fetch unit sends an address to the cache and waits for the instruction to return. This provides a simple model of instruction prefetch.
- The instruction execution unit fetches operands and stores results. The instruction execution unit waits until an instruction is ready in the instruction register, begins fetching operands from the cache, executes the instruction,and then stores results.

- The L1 cache receives addresses from the prefetch and returns instructions either from the cache or from the next level of the memory hierarchy. The cache also receives addresses from the execution unit and reads or writes operands, again from the cache or from the next level of the hierarchy. The handling of writes varies with different write algorithms. If separate Ll instruction and data caches are present, they respond to the instruction fetch and instruction execution units, respectively.
- The L2 cache receives addresses from the Ll cache (or caches) and reads or writes operands from its storage or from the primary memory system. The handling of writes varies with different write algorithms.
- The bus is a half-duplex datapath connecting the caches to the memory system. Devices on the bus must arbitrate for bus ownership before commands or data can be sent.
- The primary memory consists of a number of interleaved memories. Simulation parameters include the interleaving factor, access time, and cycle time of main memory.

Implemented a two-level (L1 and L2) cache simulator in C++ with round robin eviction policy The cache simulator takes several parameters describing the cache (block size, associativity, etc) along with a memory access trace file for an input program.

Cache Design

- Read Miss:
  - on a read miss, the cache issues a read request for the data 1 from the lower level of the cache. Once the data is returned, it is placed in an empty way, if one exists, or data in one of the ways is evicted to create room for the new data.
  - The ways of the cache are numbered from 0,1,2..W-1 for a W-way cache. If an empty way exists, data is placed in lowest numbered empty way.
  - Eviction is performed based on a round-robin policy. Each way has a counter that is initialized to 0, counts up to W-1 and loops back to zero. The current value of the counter indicates the way from which data is to be evicted. The counter is incremented by 1 after an eviction.
- Write Hit: both the L1 and L2 caches are write-back caches.
- Write Miss: both the L1 and L2 caches are write no-allocate caches. On a write miss, the write request is forwarded to the lower level of the cache.
- Non-Inclusive: the L1 and L2 caches are non-inclusive.

Configuration File(config.txt)

The parameters of the L1 and L2 caches are specified in a configuration file. The format of the configuration file is as follows.

- Block size: Specifies the block size for the cache in bytes. This should always be a non-negative power of 2 (i.e., 1, 2, 4, 8, etc).
- Associativity: Specifies the associativity of the cache. A value of "1" implies a direct-mapped cache, while a "0" value implies fully-associative. Should always be a non-negative power of 2.
- Cache size: Specifies the total size of the cache data array in KB.

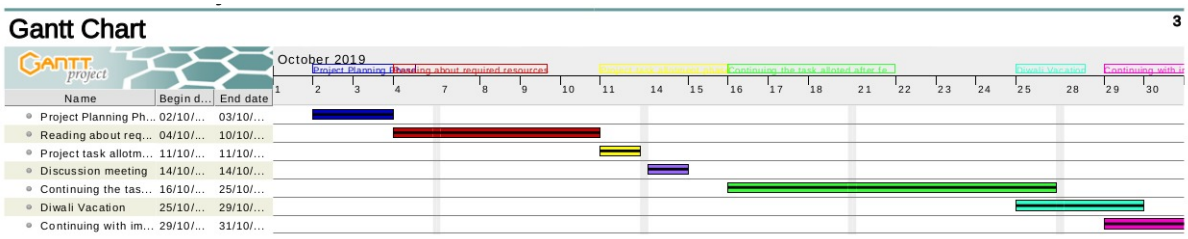
An example config file is: a 16 KB direct-mapped L1 cache with 8 byte blocks, and a 32KB 4-way set associative L2 cache with 16 byte blocks

Trace File (trace.txt): Simulator needs to take as input a trace file that is used to compute the output statistics. The trace file specifies all the data memory accesses that occur in the sample program. Each line in the trace file specifies a new memory reference. Each line in the trace cache therefore have the following two fields:

- Access Type: A single character indicating whether the access is a read ('R') or a write ('W').
- Address: A 32-bit integer (in unsigned hexidecimal format) specifying the memory address that is being accessed. Fields on the same line are separated by a single space.

The code reads the trace file one line at a time in order. After each access, code emulates the impact of the access on the cache hierarchy.

5 Gantt chart with proposed timeline



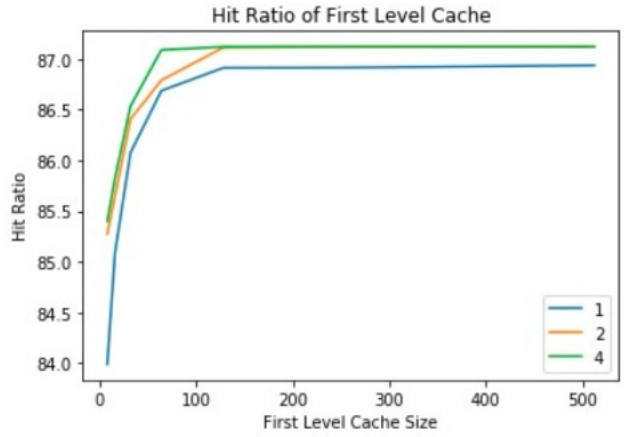
6 Result and Discussion

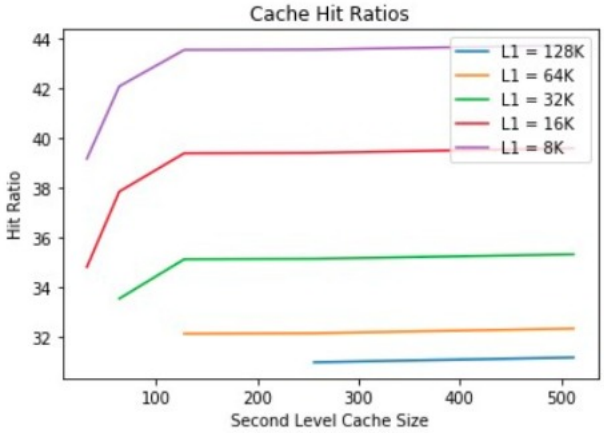
Our simulation model is fairly complex and permits many parameters to be varied. We have chosen to fix a number of the parameters and to concentrate primarily on those which we consider most relevant, namely the sizes of the L1 and L2 caches. In general, we have examined systems with primary memory access times that are large compared to cycle time. We have also made a number of simplifying assumptions. First, the line sizes of the Ll and L2 caches are identical (4 bytes). We examined variations in fill size (i.e., the number of cache lines read on each miss) but don't report here results for all combinations of sizes . Second, we assume only direct mapped organization. Third, we assume no write allocate on write misses, and only limited write buffering.

6.1 Hit Ratio

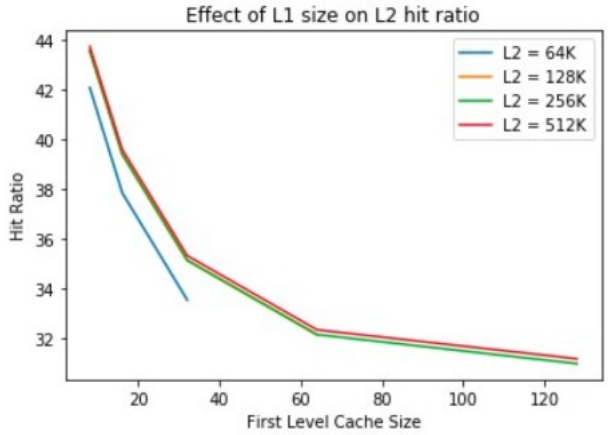
Figure 3 shows hit ratios for a single-level cache of various sizes. These hit ratios are consistent with those reported in a study of VAX cache behavior 171. Of course, the hit ratio for a single level cache will be the same as the hit ratio for the L1 cache in our two-level model. The figure also shows the effect of using l-, 2-, and 4-line cache fills. The total variation in cache hit ratio when increasing cache size from SIC bytes to 512Ii bytes is less than 10 percent. For a memory access time of 15 cycles, this difference in hit ratio accounts for almost a 50 percent increase in performance. The change in fill size can also be significant, particularly with small cache sizes; increasing fill size from 1 to 2 lines produces a 10 percent performance increase for an SK byte cache.

Figure 4 shows the hit ratio of the L2 cache in our two-level cache system. As the figure shows, varying the size of the L2 cache has a significant effect on hit ratio, and our simulation shows L2 hit ratios ranging from 20 to 90 percent. Fill size can also affect the hit ratio of the L2 cache. For example, using an 8K Ll cache, increasing the L2 fill size from 2 to 4 lines increases its hit ratio by 7 to 20 percent, depending on its size.





As would be expected, hit ratios of L2 caches are relatively low because the L2 cache sees a reference stream consisting of only misses to the Ll cache. This is true even for Iarge L2 caches; with our trace data, an L2 cache of 512K bytes has a hit ratio of less than 75 percent when placed behind a lFI¡ Ll cache, and less than 65 percent when placed behind a 321¡ Ll cache. As the Ll cache size increases, its hit ratio increases, and the hit ratio of the L2 cache decreases. This is shown more diagramatically in Figure 5. Here we see that a G4K L2 cache suffers a one third decrease in hit ratio when its Ll cache is increased from 8K to 16K bytes. A 128K L2 caches suffers a 23 percent decrease for the same Ll size change.



6.2 Performance

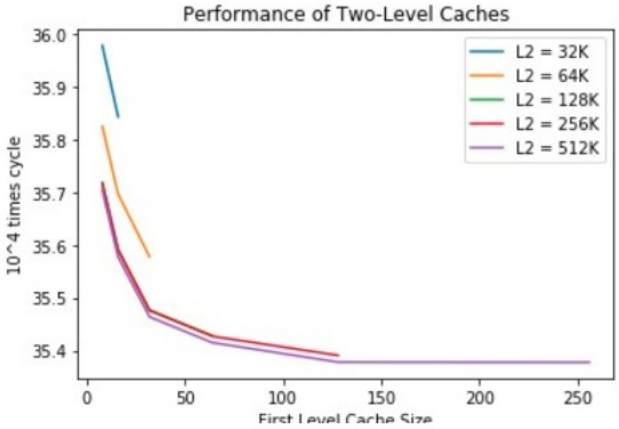
Figure Ga shows the performance effect of increasing L2 cache size for various Ll cache sizes. It is interesting to note that while increasing the size of the L2 cache has a substantial impact on its hit ratio, the hit ratio change may have little effect on the real system-level performance. For example, consider the hit ratio and performance of an 8K Ll cache with line fill and different L2 cache sizes. Increasing the size of the L2 cache from 32K to 512I¡ increases its hit ratio from G3 to SS percent - a 25 percent improvement. This improved hit ratio decreases the runtime from 30 million cycles to 28 million cycles, a performance improvement of only 7 percent. The reason for the disparity is straightforward; most memory requests are satisfied at the Ll cache, and only the small percentage that miss are sent to the second level. For example, if the Ll cache has a 90 percent hit ratio, only 10 percent of the requests are passed on to the L2 cache. Even if the L2 cache has a hit ratio as low as 50 percent, only 5 percent of memory requests will be passed to primary memory. In this case, improving the 2 hit ratio can at best improve on this 5 percent of requests going to primary memory. Of course, for primary memories with large access times, the improvement may be worthwhi

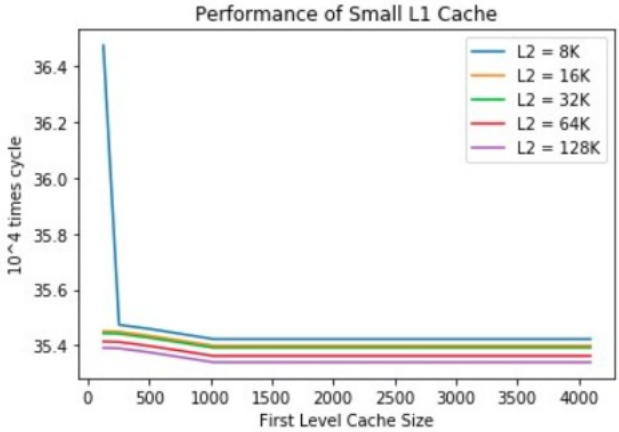
It is clear from Figure Ga that a larger L2 cache will always outperform a smaller L2 cache. On the other hand, an L2 cache is not always beneficial. This can be seen in Figure which shows the same two-level data plotted as a function of Ll cache size. Compared to a system with a one-level 8Ii cache (the top line in Figure adding a 32K L2 cache improves performance by almost 12 percent, while adding

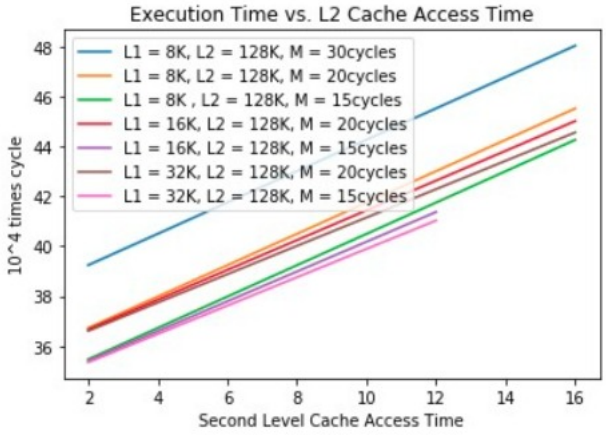
a 64K L2 cache shows a 15 percent improvement. However, adding 128K L2 cache to a system with 641¡ L1 cache has no impact, and adding a 25GK L2 cache to a system with 128Ii Ll actually degrades performance

The existence of an L2 cache can degrade overall system performance because it increases the total memory access time for accesses that miss. This becomes more of a problem as the Ll cache size increases and the L2 hit ratio decreases. If the L2 hit ratio is less than 50 percent, more than half of the accesses to the L2 cache will take an additional penalty in getting to primary memory. Starting the L2 lookup and the primary memory access in parallel will alleviate this problem but at the cost of wasted bus and memory bandwidth. Figure 7 illustrates the effect of increasing the L2 cache access time for an L2 size of 128K. Ll sizes of SK, lGI¡, and 321¡ are shown with various primary memory access times. An increase in L2 cache access time causes a linear increase in execution cycles with a greater slope for smaller Ll sizes. Obviously the smaller the Ll cache, the lower the Ll hit ratio, and the greater the effect of a slower L2 cache.

AS previously stated, high on-chip densities make it possible to include small on-chip caches on current microprocessors. lt is interesting to know whether even tiny on-chip caches are worthwhile, particularly when backed up by second-level caches. T modelled small Ll caches with sizes varying from 128 bytes to 4096 bytes, backed up by L2 caches of SK to 128Ii bytes, to cheek the utility of such configurations. These measurements assume a l-cycle Ll cache and a cycle L2 cache. For comparison, we also show the performance of a single l-cycle Ll cache and a single 4-cycle "L2" cache. The results are summarized in Figure First, we see the significant impact of adding a second level; adding an SK L2 cache to the 128 byte Ll cache nearly doubles the performance. Adding an on-chip cache is also significant; compared with no Ll cache, a 12 percent byte Ll cache gives a 3.5 percent performance over a single 81¡ L2 cache. Given a two-level structure, the size of the Ll cache is clearly the most crucial parameter. the 128 byte Ll cache, changing the size of the L2 cache from SK to 128K produces less than a 12 percent improvement. This is because the four cycles needed to process Ll cache misses are large enough that eliminating misses from the second level is relatively insignificant. The effect is similar on larger Ll caches but is magnified by the higher miss ratio in small local caches.


Performance of Two-Level Caches
L2 = 32K
L2 = 64K
L2 = 128K
L2 = 256K
L2 = 512K
10^4 times cycle
First Level Cache Size


Performance of Small L1 Cache
L2 = 8K
L2 = 16K
L2 = 32K
L2 = 64K
L2 = 128K
10^4 times cycle
First Level Cache Size


Execution Time vs. L2 Cache Access Time
L1 = 8K, L2 = 128K, M = 30cycles
L1 = 8K, L2 = 128K, M = 20cycles
L1 = 8K , L2 = 128K, M = 15cycles
L1 = 16K, L2 = 128K, M = 20cycles
L1 = 16K, L2 = 128K, M = 15cycles
L1 = 32K, L2 = 128K, M = 20cycles
L1 = 32K, L2 = 128K, M = 15cycles
10^4 times cycle
Second Level Cache Access Time

6.3 Sequential Cache Misses

In a high speed system it may be possible to process more than one memory request simultaneously, i.e., when a read misses in the cache and a request for the data is sent to memory the next sequential read can be started before the first has completed. With a long memory access time and several independent memory banks, the first reference will require the entire memory access time to complete. But, if a following reference can be started before the first finishes, there will be a significant performance advantage. smaller caches with low hit ratios would benefit greatly from the ability to be able to process two reads simultaneously (10 - 15 percent fewer cycles). The incremental improvement made possible by increasing the number of simultaneous reads to three is extremely small, on the order of one percent. However, as the cache size is increased there is a much smaller performance improvement. With a 32Kbyte local cache and a 30 cycle memory access time

6.4 Separate Instruction and Data Caches

There is little performance difference between a separate I and D cache and a shared cache with the same total cache size and access time. Separating the caches provided a performance increase in the range of five to ten percent. The extra complexity of adding a complete set of cache control logic and an extra cache directory would not seem to be cost effective. This result is an effect of the complex instruction set architecture. In a RISC-like architecture with a small number of cycles per instruction, separate I and D caches may be required because the I fetches will saturate the cache.

On the other hand, separate I and D caches may have an advantage in our system. As previously stated, larger memories typically have longer access times due to physical constraints; the shared cache model, with the same total amount of memory, might require a slower clock frequency than would separate caches. In'effect, adding an extra cache may allow us to double the effective size of the cache without decreasing the system cycle time. This performance increase would be in the range of ten to almost twenty percent and may justify the extra expense and complexity.

7 Conclusion

We have studied several aspects of two-leve cache memories in uniprocessors.We have modeled the miss rate, cache area, and cache access time to achieve a solid basis to study on-chip memory system tradeoffs.

An extra level of cache memory can provide a worthwhile performance gain when used with proper combinations of small first-level caches and large main memory access times. Two-level on-chip cache hierarchies perform even better in low-cost systems without a board-level cache.

Combining this with set-associative second-level caches improves performance even further. This is because the increase in capacity provided by two-level exclusive caching increases as the second level of caching is made more associative.

Write-back strategies were studied for both cache levels. The best performance is provided by write-back at both levels The principal effect of write-back was the reduction of write bandwidth required at the next higher level of memory.

If only one of the caches were to use write-back,then write-back should be included in the L1 cache because

- The L1 cache typically has a faster access time, and writes could be processed more quickly without buffering, and
- The L1 cache is smaller than the L2 cache.

Lastely, multilevel caches provide a fruitful area for study, and much more needs to be done.

Certainly an important area for further evaluation is multi-level caches in multiprocessors. In this study, we concentrated on only 2-level Cache.