Performance Evaluation of Split Temporal/Spatial Caches: Paving the Way to New Solutions

Miloš Prvulović, Darko Marinov, Veljko Milutinović
Department of Computer Engineering
School of Electrical Engineering
University of Belgrade
P.O BOX 35-54
11120 Belgrade, Serbia, Yugoslavia

prvul@sezampro.yu, marinov@galeb.etf.bg.ac.yu, vm@etf.bg.ac.yu

Abstract

The purpose of this paper is to describe methods used to evaluate performance of split temporal/spatial caches. Previous work in this area mostly describes the architecture of the caches themselves and presents only the performance results, without saying much to explain the method used to obtain those results. Therefore, we first briefly survey the methods used in previous papers on split caches, then explain the particular process used to evaluate one of the designs, and finally present the results of that evaluation. We conclude with possible improvements to that cache design pointed to by our evaluation results.

Introduction

In recent years, the growth in microprocessor clock rates has by far outpaced the speedup in dynamic memory elements. At the same time, the cost of each cycle spent while waiting for memory to respond has increased due to superpipelining and superscalar microprocessor architectures. Caching is the first line of defense against long delays associated with main memory access. For these reasons, further improvement of cache designs is of the uttermost importance for the performance of microprocessors.

Recently, several research groups have come up with proposals to split the data cache horizontally in order to gain improved performance. A horizontal split, as opposed to a vertical split (i.e., hierarchy), means that the cache is divided into two or more subcaches that reside at the same level in the memory hierarchy. A widely known example of a horizontal split is the separation into instruction and data caches, found in most modern microprocessors. A logical next step would be to further split the data cache, with each subcache exploiting a different locality pattern.

Data cache performance has traditionally been evaluated by measuring cache hit rate or average memory latency during execution of representative benchmarks, such as the SPEC benchmark. The purpose of this paper is to describe the process used to evaluate the performance of split caches, the problems encountered and the solutions applied.

Split cache architectures

This section briefly describes several split data cache architectures, in order to familiarize the reader with the idea and mechanisms behind the split data cache designs. Although not explicitly underlined, in all these cases, in essence, different types of locality are treated using different mechanisms. Two types of locality are considered: spatial and temporal. In the first four designs, the splitting is done according to the spatial locality, and in the last two, the splitting is done according to the temporal locality of the data.

Proceedings of the IEEE/ACM ISCA98 Workshop on Performance analysis and its Impact to Design (PAID98), Barcelona, Spain, June 27-28 1998.

Caches split by spatial locality

The Split temporal/spatial cache [Milutinovic95] was introduced by a research team including members from University of Belgrade (Belgrade, Yugoslavia), University of Montenegro (Podgorica, Yugoslavia), and Sun Microsystems (Mountain View, USA). In this design, the cache is split into a temporal subcache (with line size of one 4-byte word) and a spatial subcache (with a line size of four 4-byte words). Three possibilities are explored for the ratio between capacities of spatial and temporal primary subcaches: 1:1, 1:2, and 1:4. Also, there is a secondary cache only for the temporal subcache. The decision which subcache to use is done at runtime, profile time or both using the same counter-based heuristic. Two counters per cache line are present in the spatial subcache. One of those counters, the Y counter, is incremented at each access to cache line. The other (X counter) is incremented when the upper half of cache line is accessed and decremented at access to the lower half. When Y counter saturates, if the absolute value of X counter is larger than a specified threshold, cache line is marked as temporal and removed from the spatial subcache. Sequential prefetching is also done in this design, but only in the spatial subcache.

The Dual data cache [Gonzalez95] was introduced by a research group at Polytechnic University of Catalonia (Barcelona, Spain). In this design, the primary data cache is split into smaller (33% of total cache capacity) temporal subcache and a larger (66% of the total cache capacity) spatial subcache. Temporal subcache has a line size of eight bytes, while the spatial subcache has a larger line size (16 or 32 bytes). The decision which subcache to use is done at runtime, using a variant of stride directed prefetch predictor. The predictor, in essence, tries to detect if a load/store instruction accesses memory at addresses that differ by a constant stride. Using this information, at each cache miss the cache controller decides if data should be cached in temporal subcache, spatial subcache, or not cached at all. Sequential prefetching is also done in this design, but only in the spatial subcache.

In [Sanchez97], a group from the same university describes a substantially modified design of the dual data cache. The temporal subcache in this new design is only a small fully associative buffer (up to 16 single word 8-byte entries) while the spatial subcache remains large and with the line size of 32 bytes. The decision which subcache to use for each data access is done at compile time in this new design, by performing data locality analysis. In this design it is also possible that, on a cache miss, the data is fetched into temporal subcache, spatial subcache, or not cached at all.

The Array cache [Tomasko97a] was introduced by a research group at Colorado State University (Fort Collins, USA). In this design, the primary data cache is split into a smaller (25% of the total cache capacity) scalar subcache and a larger (75% of the total cache capacity) array cache. Line size in scalar subcache is 32 bytes while in the array cache it is larger (experimentally varied from 64 to 512 bytes). The decision which subcache to use by a particular reference is done at compile time, by marking the scalar variable accesses to use the scalar subcache, while array accesses go to the array subcache.

Caches split by temporal locality

The assist cache [Chan96] is incorporated into the HP PA7200 CPU. The first level cache consists of a large, external, direct-mapped main cache and a smaller, on-chip, fully associative assist cache. Main cache and the assist cache have equal line sizes of 32 bytes. Besides reducing conflicts in the main cache, the assist cache also serves as a “non-temporal” subcache, while main cache serves as a “temporal cache.” While the previous four approaches split the cache according to spatial locality of data, in the assist cache the splitting is done according to temporal locality. At compile time, some memory access instructions are marked as “spatial locality only.” Data accessed by these instructions is not cached in the main cache, but only in the assist cache. Other data items are considered as having both temporal and spatial locality and can be cached by the main cache.

The non-temporal streaming (NTS) cache [Rivers96] was proposed by a research team from University of Michigan (Ann Arbor, USA). This design is similar to the assist cache. The cache is split into a larger, direct-mapped, temporal subcache, and a smaller, fully associative, non-temporal subcache. Both subcaches have equal line sizes of 32 bytes. As in the assist cache, in NTS cache the splitting is done according to temporal locality. The decision which subcache should be used is done at runtime. Each entry in temporal subcache is associated with a bit-vector containing one bit for each word used. If no reuse is detected while a particular cache line was in the temporal subcache, it is marked as non-temporal (on eviction from temporal subcache). Future accesses to this memory block would cause a fetch into a non-temporal subcache.

Previous work

The performance evaluation of Split Temporal/Spatial (STS) cache in [Milutinovic96a] was done using the ATUM traces [Agarwal86] collected on a DEC VAX by altering its microcode. These traces were fed into software simulators of the STS cache and the conventional cache (for comparison). Performance results are reported in [Milutinovic96a] by giving the percentual performance improvement of STS over the classical cache design, using the average memory latency as the performance indicator. The ATUM traces have the advantage of including not only the memory accesses of a single application. These traces contain all the data accesses of a processor, including OS activities. However, these traces are quite old and contain fewer data accesses when compared to modern benchmarks. Therefore, we felt that evaluation of STS using the newer IBS traces (which also include OS references) and SPEC benchmarks would increase confidence in the STS design and perhaps direct us toward improved split cache designs.

Evaluation of hardware-managed Dual data cache in [Gonzalez95] was done using synthetic benchmarks, kernels for several important applications (FFT and matrix multiplication variants), 3 applications from the SPECint92, and 5 applications from the SPECfp92 application benchmark suite. The data access traces were collected on a DEC Alpha-based workstation using the Atom instrumentation tool. The traces were then fed into software simulators of dual data cache and conventional cache (for comparison). As the SPEC92 benchmarks applications produced too many references for efficient simulation, for each application the first 10 million references were discarded and the following 5 million were simulated. The performance results reported in [Gonzalez95] are expressed by average memory latency (observed by the processor) and are compared to corresponding results for the conventional cache.

Performance of software-managed Dual data cache was evaluated in [Sanchez97] using seven applications from the SPECfp95 benchmark suite. These FORTRAN applications were instrumented using the ICTINEO toolkit and fed into software simulators. Performance results are expressed in [Sanchez97] by hit rates, average memory latencies and the amount of fetched data. These are then compared with corresponding results for the conventional cache.

The performance evaluation of Array cache design in [Tomasko97a] was done using eight of SPECfp95 and four NAS benchmark applications. As reported in [Tomasko97b], the data access traces were collected using the QPT2 instrumentation tool. It is unclear from the reports whether only a part of all data references or all of them were simulated for each application. However, from the reported number of simulated references for each SPECfp95 application, it can be assumed that all references were simulated, but using either test or training input data sets (not the reference data sets). The results are reported using miss ratio of a cache as a performance indicator. The results for split array/scalar cache are compared with corresponding results for the conventional cache.

The performance evaluation of the assist cache is not described in [Chan96]. However, as this scheme is included in a high-volume commercial CPU, one can safely assume that during the design phase at HP this scheme produced significant improvements over a classical cache design.

The performance of non-temporal streaming cache in [Rivers96] was evaluated using traces of several different applications (three from NAS and one each from of the Perfect and SPEC benchmark suites, LINPACK benchmark as well as three other applications). The traces were collected using the ATRACE package on an IBM RS/6000 machine running under AIX. Performance results are reported in [Rivers96] by presenting the miss ratios and average memory latencies. The NTS cache performance is compared to the performance of a conventional direct-mapped cache and with an assist cache.

Measuring the performance of STS cache

To evaluate performance of the split temporal/spatial cache, we developed a functional cache simulator. Then we fed the simulator with the data memory access streams extracted from IBS traces, as well as the data memory access streams obtained by running applications from the SPEC 95 benchmark suite.

Development of the cache simulator proved to be the easiest part. The simulator was developed in C++. The major components are a generic trace reader and functional simulator of the classical cache design. Inheritance mechanisms were then used to derive several variants of STS cache from the classical cache design and to derive the IBS and Pixie trace readers from a generic one.

The IBS traces were readily available. However, to get the data access streams of SPEC95 applications we had to employ the Pixie instrumentation tool on an SGI MIPS-based workstation.

The purpose of the SPEC95 benchmark suite is to evaluate high-performance computers. Each application in SPEC95 has three different input data sets: test, train, and reference. Test data sets should be used to test the correctness of program execution. Training data sets should be used for profiling in optimizing compilers. Reference data sets are intended for use in actual evaluation of computer systems. A single SPEC95 application using its reference input data set takes on the order of hours to execute on an actual SGI MIPS R10000-based workstation. It is obvious that it is neither possible to simulate the entire reference stream of a reference run, nor it is possible to store such trace for the purpose of simulation.

Not storing traces is not a problem. The trace stream generated by the instrumented application can be immediately fed to the cache simulator through a pipe. However, reduction in the number of data accesses to be simulated in each application should maintain the overall data access pattern of that particular application. A sampling technique, in which evenly spaced parts of the trace are simulated while the remaining parts are skipped, is a good candidate. However, this approach leads to many cache misses at the beginning of each simulated portion, where the working set of one simulated portion is discontinuously exchanged with the working set of another.

The approach used in this paper is to execute a statistics-gathering run using the reference data set of an application. Statistics such as the percentage of writes and reads and several data locality indicators such as conventional cache hit ratios are collected during this run. Then we run the same application using its training, test and reference input data set and search for a portion of trace that has statistics close enough to those of the entire reference run.

When evaluating a particular cache design, we simulate the entire data access stream from the beginning of the trace, but collect performance statistics only during execution of the “significant” portion we found. It is unnecessary to simulate past the end of the “significant” portion. It is obvious that it is desirable to find a “significant” portion as close to the beginning of the trace as possible, in order to reduce the number of data accesses simulated before the start of the “significant” portion.

Let S be the number of data memory accesses before “significant” portion begins, and let P be the number of accesses in the “significant” portion. We need to simulate a total of S+P accesses, although statistics are gathered only for P accesses of a “significant” portion. For our evaluation, for each application we seek a significant portion that meets the following conditions:

1) “Significant” portion contains at least 10 million accesses ($P \geq 10,000,000$)
2) If the percentage of writes in the entire reference run is W_p, the percentage of writes in the “significant” portion must be between $0.99W_p$ and $1.01W_p$.
3) Let HR_1 and HR_2 be the hit rates of 2kB primary and 16kB secondary caches in the conventional cache hierarchy for an entire reference run, and let HP_1 and HP_2 be the corresponding hit rates for the “significant” portion. Then the following must hold: $0.99HR_1 \leq HP_1 \leq 1.01HR_1$ and $0.99HR_2 \leq HP_2 \leq 1.01HR_2$.
4) We examine other statistics for the “significant” portion and an entire reference run and reject portions that satisfy 1), 2), and 3) but are dissimilar to the entire reference run according to some other statistic.

Of all the portions satisfying the above conditions, we take the one that has the smallest S+P.

Experimental results

The IBS traces are readily available and contain few enough accesses to be simulated in entirety. Therefore, our initial evaluation of STS cache design was done using the IBS traces. We assumed the same STS cache design as in [Milutinovic96a]. All caches are 4-way set associative. Secondary cache in the conventional cache system is eight times larger than the primary cache, and secondary temporal cache is eight times the size of the primary temporal subcache. Spatial subcache has no secondary cache. Access times differ from those in [Milutinovic96a] because the technology has progressed since then. We assume that each primary cache hit takes one cycle. A secondary cache hit takes four cycles for the first four-byte word and a cycle for each additional word. A cache miss takes 16 cycles for the first word and a cycle for each additional word (i.e., the data busses are all 32 bits wide). Conventional cache and spatial subcache of STS cache have line sizes of four words, while the temporal subcache of STS has the line size of one word. To avoid additional complexity in the bus controller and the bus protocol, a miss in the temporal subcache causes an entire four-word line to be fetched. However, only the required word is actually cached in the temporal subcache and the CPU request is satisfied as soon as the cache line is filled.

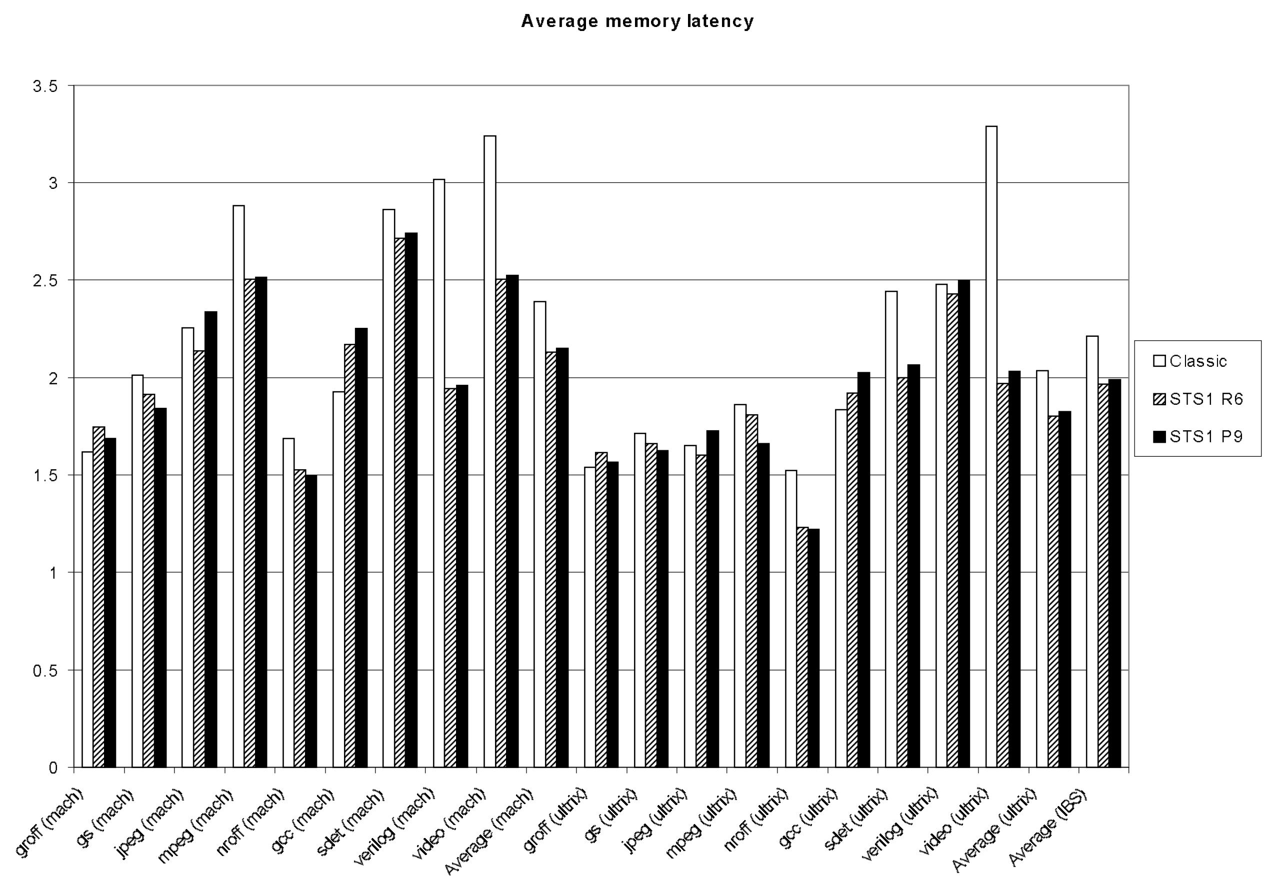

Figure 1:
STS1 R6 is an STS cache that has equal spatial and temporal primary subcaches, locality decision is made at runtime and threshold for X and Y counters are both set at six. STS1 P9 makes locality decisions by profiling using nine as the threshold for both counters. Classic is a conventional cache with a primary cache equal to the sum of capacities of STS subcaches and a secondary cache twice the secondary temporal cache of STS.

It is obvious from Figure 1 that, on the average, both runtime and profiling-based STS designs outperform a conventional cache of similar complexity (see [Milutinovic96b] for complexity evaluation of STS cache). Such results on IBS benchmarks justify the effort needed to evaluate the STS cache design using the SPEC95 benchmark suite.

Deadlines did not permit us to complete the evaluation using SPEC95 benchmarks before submitting this paper.

The road ahead

The STS cache shows improved overall performance over the conventional cache. However, it can be seen that for some applications STS has slightly lower performance while for some other applications it shows huge improvements. As the first step in improving the STS design, we want to explain this variation in performance.

Our experiments with varying X and Y counter thresholds (XMax and YMax) indicate optimum values for these thresholds vary greatly between the applications. For example, in runtime-based STS most IBS traces show best results when the XMax and YMax are both six or seven, but JPEG under Ultrix shows best performance when the thresholds are 17. The purpose of counters is to determine if only one half of a spatial cache line is used most of the time. If a single half of particular cache line is accessed Xmax times in a row, then the line is marked temporal. Consider a program that accesses a particular word several times and then moves on to the next word. An STS cache having small Xmax and Ymax may declare the line temporal before the program accesses gets to access the other half, thus moving an essentially spatial line into the temporal subcache. On the other hand, if the thresholds are too high, an actually temporal line gets accessed a few times, but Y counter never saturates and the line is never declared temporal.

To overcome this problem, a different heuristic for detecting temporal locality is needed. We are currently evaluating a scheme in which a simple flag is kept for each part of the spatial cache line. This flag is initially reset to zero and is set to one when that part of the cache line is first accessed. On eviction, if only one of the flags is set, the line is temporal (because only one part of it was accessed). For example, two flags per line are enough to detect if only a single half of cache line was accessed.

Another problem in STS design is that, once the data is declared temporal, there is no way for it to be declared spatial again. If a line that is actually spatial gets marked as temporal, it will always be cached in the temporal part. Caching lines that do exhibit spatial locality in the temporal part is costly - instead of incurring one miss per four words in the spatial part, one miss for each of the four words is incurred in the temporal part.

To overcome this problem, we need a way to detect spatial locality in the temporal subcache. We are currently investigating a scheme in which, at each temporal cache miss, a neighboring line is sought in the temporal cache. If a neighboring line is present, it is evicted, and a larger line is fetched into the spatial part (and marked spatial, of course).

Finally, different applications access different mixtures of spatial and non-spatial data. Consequently, optimum ratio of spatial to temporal subcache capacity varies. This problem can only be attacked by dynamic (runtime) allocation of cache capacity between spatial and temporal subcache. We are currently not considering such designs.

Conclusion

Split caches offer performance improvements over conventional (non-split) caches because they are better suited to exploitation of different locality patterns commonly found in applications. However, the choice of representative access traces is more difficult when evaluating split caches, because more locality parameters must be fairly represented. In this work, we have proposed a way to reduce the number of simulated data accesses and still keep a high confidence in the results obtained by simulation. We have also shown how the simulation results, when correlated with known properties of data access pattern, may lead to improvements in split cache design. We hope that our work on both cache designs and methods of data trace selection can serve as a guideline for researchers and designers of future cache architectures.

References

[Agarwal86] Agarwal, A., Sites, R., Horowitz, M., "ATUM: A New Technique for Capturing Address Traces Using Microcode," *Proceedings of the 13th Annual Symposium on Computer Architecture,* Tokyo, Japan, June 1986, pp. 119-127.

[Chan96] Chan, K. K., Hay, C. C., Keller, J. R., Kurpanek, G. P. Schumacher, F. X. Zheng, J., "Design of the HP PA7200 CPU," *Hewlett-Packard Journal*, February 1996, pp. 1-12.

[Gonzalez95] Gonzalez, A., Aliagas, C. and Valero, M., "A Data Cache with Multiple Caching Strategies Tuned to Different Types of Locality,“ *Proceedings of the International Conference on Supercomputing (ICS '95)*, Barcelona, Spain, 1995, pp. 338-347.

[Milutinovic95] Milutinovic, V., "The STS Cache," University of Belgrade Technical Report #35/95, Belgrade, Serbia, Yugoslavia, 1995.

[Milutinovic96a] Milutinovic, V., Markovic, B., Tomasevic, M., Tremblay, M., "The Split Temporal/Spatial Cache: Initial Performance Analysis," *Proceedings of SCIzzL-5*, Santa Clara, California, USA, March 1996, pp. 63-69.

[Milutinovic96b] Milutinovic, V., Markovic, B., Tomasevic, M., Tremblay, M., "The Split Temporal/Spatial Cache: A Complexity Analysis," *Proceedings of SCIzzL-6*, Santa Clara, California, USA, September 1996, pp. 89-96.

[Sanchez97] Sanchez, F. J., Gonzalez, A., Valero, M., "Software Management of Selective and Dual Data Caches,“ *IEEE TCCA NEWSLETTERS*, March 97, pp. 3-10.

[Rivers96] Rivers, J. A., Davidson, E. S., "Reducing Conflicts in Direct-mapped Caches with a Temporality Based Design," *Proceedings of International Conference on Parallel Processing*, 1996.

[Tomasko97a] Tomasko, M., Hadjiyiannis, S. and Najjar, W. A., "Experimental Evaluation of Array Caches,“ *IEEE TCCA NEWSLETTERS*, March 97, pp. 11-16.

[Tomasko97b] Tomasko, M., Hadjiyiannis, S. and Najjar, W. A., "Evaluation of a Split Scalar/Array Cache," Technical report CS-TR-97-105, Colorado State University, Fort Collins, Colorado, USA, January 1997.