The Split Spatial/Non-Spatial Cache: A Performance and Complexity Evaluation

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

*From August 98, at the University of Illinois at Urbana-Champaign
**From August 98, at the Massachusetts Institute of Technology

prvul@computer.org, darko@mit.edu, zorand@galeb.etf.bg.ac.yu, vm@etf.bg.ac.yu

Abstract

A simple new method of detecting useful spatial locality is proposed in this paper. The new method is tested by incorporating it into a new split cache design. Complexity estimation and performance evaluation of the new split cache design is done in order to compare it to the conventional cache architecture and the split temporal/spatial cache design.

Introduction

In recent years, the speed gap between dynamic memories and microprocessors has been steadily increasing. For this reason, a lot of effort is invested into finding ways to reduce or hide memory latency. One of the oldest and most powerful ways of reducing the memory latency is through use of cache memories.

Caches exploit the locality of data access. A small (but fast) memory is able to satisfy most memory access requests issued by the processor, so in most cases there is no need to wait for slow (but large) main memory to respond. Conventional cache designs non-selectively cache all data. If the memory request is not satisfied from the cache, the main memory response has to be waited for. However, at the same time memory contents are brought into the cache in hope that future processor accesses will reuse this data. The property that the same data items tend to be accessed again in the near future is called temporal locality. Neighboring data items tend to be accessed in the near future, so spatial locality also exists. Spatial locality is exploited by bringing in an entire cache block (with several data words in it).

Today, it is widely recognized that not all data exhibit both types of locality, and some data exhibits no locality at all. For example, clearing a large vector involves spatial locality only, because each data item is accessed only once. Frequent accesses to a single global counter are in most cases temporal only, because neighboring data items are often not used so heavily. Most accesses to large hash tables exhibit neither type of locality. If a data item exhibits no temporal locality, bringing it into the cache is useless. If no spatial locality is exhibited by a data item, bringing an entire cache block is even more wasteful.

Several ways to detect differing localities and exploit each in a manner suited to that particular locality type are found in the open literature. Here we will concentrate on an increasingly popular way of exploiting different data localities – splitting the cache into several subcaches, with each subcache designed with a particular type of locality in mind. First, we will survey the existing split cache designs. Then we will provide quantitative definitions for both locality types. Finally, we will provide a way to represent each solution as a line in the temporal-spatial locality plane.

The Split Spatial/Non-Spatial Cache

Variations in temporal locality is easier to efficiently detect and exploit, so the design space in this are is narrower and better explored than in the are of spatial locality exploitation. Because almost any technique for exploitation of varying temporal locality is compatible with any technique for exploiting varying spatial locality, we will concentrate on devising a new design for spatial locality exploitation.

This new design will utilize cache splitting and prefetching in order to exploit any degree of spatial locality in the most efficient manner. The primary cache is split into two subcaches, according to the spatial locality of data. The non-spatial subcache consists of blocks that are one word (eight bytes) wide, and is intended to cache data words exhibiting no useful spatial locality. The spatial subcache consists of blocks that are 32 bytes (four words) wide, and is intended to cache the data words exhibiting useful spatial locality. The number of blocks in the non-spatial and in the spatial part is the same. Therefore, the capacity of the spatial part is four times the capacity of the non-spatial part. Because the block sizes in the spatial and the non-spatial subcaches are different, we will hereafter refer to a spatial block as "block," and use "sub-block" to refer to a non-spatial block. For example, one block consists of four sub-blocks.

The flag-based method of spatial locality detection is used to detect if the data exhibits useful spatial locality. Each block in the spatial subcache is associated with four bits (one for each word in the block). These bits are initialized to zero when the block is fetched into the cache, and each bit is set when the corresponding word has been accessed. When that block is evicted from the cache, the four bits associated with it are examined. If less than two of these four bits are set, the block exhibits no useful spatial locality (since none or only one of its words has been accessed during its lifetime in the cache). This block is marked as non-spatial, and the next access to any sub-block of that block will cause (only) that sub-block to be fetched into the non-spatial subcache.

In the non-spatial subcache, a method of detecting spatiality is also necessary. If two or more sub-blocks that are parts of the same block are present in the non-spatial subcache, then this block should be marked as spatial and cached in the spatial subcache.

Since caching spatial blocks in the non-spatial subcache causes a miss whenever a new sub-block of that block is accessed, we want to stop caching such blocks in non-spatial cache as soon as they are detected to be spatial. Therefore, each miss in the non-spatial subcache invokes a search for the other subblocks that belong to the same block as the missed sub-block. If any such sub-block is found, fetching another sub-block into the non-spatial subcache would cause two sub-blocks of the same blocks to reside in the cache, which indicates that the block in question is spatial. Instead, the block is marked as spatial and a fetch of the entire block into the spatial subcache is initiated. The one sub-block that was found in the non-spatial cache is transferred into the newly allocated block in the spatial cache and invalidated in the non-spatial cache. Since fetching the missed sub-block into the non-spatial subcache would cause a miss anyway, the hit rate does not suffer from this kind of transfer between the subcaches. Another question is how to determine if sub-blocks belonging to the same block as the missing one are present in the non-spatial subcache. Figure 1 shows a design that detects this situation as soon as hit/miss of the sub-block being accessed is detected. This is important because otherwise the tag array access of the non-spatial subcache could become a bottleneck.

When spatial locality is too large to be exploited by multiple-worded cache blocks, prefetching is used. The spatial locality of data that resides in the non-spatial subcache does not even justify caching multiple words. Prefetching in the non-spatial subcache would therefore be useless. For this reason, prefetching should only be done for data that resides in the spatial subcache.

Various prefetching methods may be implemented, but "always," "flag" and "bi-directional" are particularly well suited for implementation in this design.

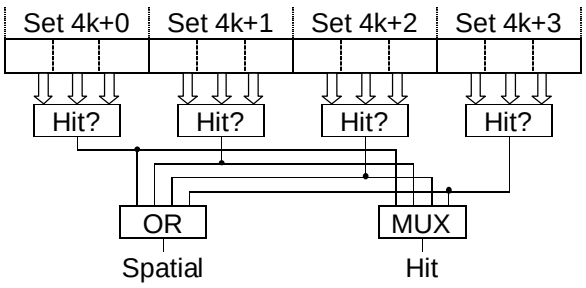

Figure 1: Detecting spatiality in non-spatial part of SS/NS cache.

Whenever a cache block is accessed, “always” initiates a fetch of the next cache block. As spatial blocks are accessed multiple times during their lifetime in the cache, only the first access is likely to initiate a useful prefetch. Other accesses initiate a prefetch of a block that has already been prefetched, thus degrading performance by making cache tag ports busy (to check presence of the block for which a prefetch has been initiated).

“Flag” is similar to “always,” but a prefetch is initiated only at first access to a particular block. Implementation of “flag” in a conventional cache design requires one bit per block to mark whether the block has already been accessed or not (thus the name “flag”). In the spatial subcache, no additional bit is required. The block has been accessed if any of its words is accessed, so if none of the four bits associated with the block is set, a prefetch is initiated.

“Bi-directional” is a bit more sophisticated and intends to exploit an array walk-through in either forward or backward direction. When block B is accessed, if block B-1 is present in the cache a prefetch of B+1 is initiated, while presence of block B+1 initiates a prefetch of block B-1. In a conventional cache, this technique causes the cache tag array to be busy with three accesses (for block B, as well as for blocks B+1 and B-1) instead of one. In the spatial subcache, the four bits can be used to detect forward or backward array walk-through without accessing neighboring blocks. When block B is accessed, block B+1 should be prefetched if the lower half of the block was used before and this is the first time the upper half is accessed. Block B-1 should be prefetched if the upper half was accessed before and this is the first access to the lower half of the block.

In [Prvulovic98] it was indicated that in caches split according to the spatial locality, a secondary cache is useful for both spatial and non-spatial data. The question remains whether to the secondary cache should also be split according to spatial locality or not. In this design, we have decided not to split the secondary cache, in order to keep the design relatively simple. Any prefetching will always be done into the primary cache in this design. There will be no prefetching into the secondary cache only. We have described a way to determine if useful spatial locality exists in a particular block while it is in the cache. It remains to be shown how the locality information about each block is to be kept while the block is not in any of the two subcaches.

The shared secondary cache can store locality information for each block that is present in secondary cache. For data that are not present in the secondary cache, one-bit spatial locality predictor can be used. In this design, we assumed that the predictor contains 512k one-bit entries, but we will examine the performance implications of varying the number of entries in this predictor. Locality information provided by this predictor is read and updated only on secondary cache misses and writebacks, so slow and cheap dynamic memory elements can be used for this predictor. No tags are necessary for the locality predictor because inaccurate prediction does not lead to inconsistencies in program execution.

For related readings see the following papers [Chan96, Gonzales95, Johnson97a, Johnson97b, Milutinovic96a, Milutinovic96b, Sahuquillo99, Sanchez97, Reilly95, Rivers96, Tomasko97, Tse98, and Tyson95].

Complexity Estimation

In the following section, we will evaluate the performance of the Split Spatial/Non-Spatial (SS/NS) cache and compare it to the conventional cache, as well as with the Split Temporal/Spatial cache of similar complexity. We assume that the caches are physically tagged, with the physical address being 36 bits wide, which is common in today’s 64-bit microprocessors. We will use the total number of bits required to implement each level of the hierarchy as a measure of complexity. Primary caches are faster and usually consume precious on-chip space, while secondary caches may be slower and are usually off-chip. Therefore, each bit in the primary cache is more costly than a bit in the secondary cache. The locality predictor in the SS/NS cache (and in STS) is not included in this evaluation since it requires only the cheap and slow dynamic memory elements.

Each cache block in a cache contains data bits, tag bits, LRU bits and a validity bit. Write-back caches include a dirty bit, the SS/NS spatial primary subcache includes four bits (F-bits) per block to detect spatial locality and the SS/NS secondary cache includes one bit (L-bit) to mark spatial locality of the block. The STS spatial primary subcache includes two four-bit counters (C-bits) to determine data locality. Since all caches are three or four-way set associative, two bits are enough to encode LRU state information. Since number of sets is a power of two in all cases in Table 1, bits of the address that encode the set number need not be stored in the tag.

	Primary								Secondary			
	Size (kB)		Associativity		Bytes per Block		Write policy		Size	Assoc	Block	WP
	NS, T	S	NS, T	S	NS, T	S	NS, T	S				
CNV	8kB		4-way		32		WT		64kB	4-way	32	WB
STS	4kB	4kB	4-way	4-way	8	32	WT	WB	32kB*	4-way	8	WB
SS/NS	1.5kB	6kB	3-way	3-way	8	32	WT	WT	64kB	4-way	32	WB

Table 1: Cache parameters for complexity and performance evaluation.
Legend: CNV–Conventional cache; STS–Split Temporal/Spatial cache; SS/NS–Split Spatial/Non-Spatial cache; NS–Non-Spatial; T–Temporal; S–Spatial; WT–Write Allocate, Write Through; WB–Write Allocate, Write Back; K-way–K-way set associative; Assoc–Associativity; Block–Bytes per block; WP–Write policy; Size–Cache size in kB.
* Only temporal data is cached in STS secondary cache.

	Number of Blocks		Bits per Block								Total Number Of Bits
			LRU+V+D+F+C+L		Tag		Data		Total		
	NS, T	S	NS, T	S	NS, T	S	NS, T	S	NS, T	S	
Primary Cache of an "8K" design											
CNV	64*4		2+1+0+0+0+0		25		256		284		72704
STS	128*4	32*4	2+1+0+0+0+0	2+1+1+0+8+0	26	26	64	256	93	294	85248
SS/NS	64*3	64*3	2+1+0+0+0+0	2+1+0+4+0+0	27	25	64	256	94	288	73344

Table 2: Cache complexity evaluation.
Legend: CNV–Conventional cache; STS–Split Temporal/Spatial cache; SS/NS–Split Spatial/Non-Spatial cache; NS–Non-Spatial; T–Temporal; S–Spatial; LRU–Replacement policy; V–Validity bit; D–Dirty bit; F–Bits to detect spatial locality; C–Counter bits; L–Bit to mark spatial.
* Only temporal data is cached in STS secondary cache.

Performance Evaluation

For simplicity, we will refer to cache hierarchies described in Table 1 as "8K" caches. A "32K" cache has all "Size" parameters form Table 1 multiplied by four, and other parameters remain the same. For example, an "SS/NS 8K" cache has a total capacity of 7.5kB in the primary cache and 64kB in the secondary cache. Similarly, a "SS/NS 32K" cache has a total capacity of 28kB in the primary cache and 256kB in the secondary cache. If "always" prefetching is included in the design, PfA is added to the mnemonic for that design, and PfB is added if "bi-directional" prefetching is used. Therefore, a "CNV 32K PfA" is a conventional cache hierarchy "always" prefetching into the 32kB primary cache, with the secondary cache being 256kB and all other parameters same as in Table 2. It will also be interesting to compare a prefetching SS/NS design with a non-prefetching conventional design with a larger block size. Therefore, "CNV 8K 64B" is the same as "CNV 8K" except both primary and secondary caches contain blocks of 64 bytes.

The STS cache as originally described in [Milutinovc95] incorporates "always" prefetching, "STS 8K" actually means "STS 8K PfA".

The original STS design proposes runtime-based and profile-based locality detection, and involves parameters – the X and Y counter limits. Fortunately, optimum performance is most often achieved when X and Y limits are equal. For each set of benchmarks, STS performance will be reported only for the counter values that result in optimum average performance on that set of benchmarks. The type (runtime or profiling) and counter limits for STS cache are signified in the mnemonic by a letter followed by the number. Therefore, "STS 8K P11" means profile-based "STS 8K" with both counter limits set to 11 and "STS 8K R7" is a runtime-based "STS 8K" with both counter limits set to seven.

Performance will be expressed using the average memory latency and bus traffic as performance measures. In our evaluation, each primary cache hit has a latency of one cycle. A secondary cache hit has a latency of four cycles for the first word and one cycle per word thereafter. Latency of a miss is 16 cycles for the first word and one cycle per word thereafter. Bus traffic is expressed in bus activity cycles per data access issued by the CPU. Bus activity includes only data transfer cycles of data fetches and prefetches (writebacks are ignored).

We used the IBS [Uhlig95] traces as the first set of workloads. These traces contain both application and OS data references collected during execution of nine applications under Ultrix and Mach operating systems.

It is obvious from Table 3 and Table 4 that the SS/NS cache improves the performance significantly over the conventional cache when no prefetching is used. Performance of the STS cache (which incorporates PfA) lags far behind the performance of prefetching SS/NS. However, the most stunning observation can be made from Table 5 and Table 6. Prefetching in the conventional cache hierarchy significantly increases bus traffic, but reduces average latency. The effect of prefetching on memory latency in SS/NS cache is about the same as in conventional hierarchy. However, prefetching causes a negligible bus traffic increase in SS/NS. To achieve this effect, it is not necessary to split the cache.

The bus traffic in a conventional hierarchy should not increase much when prefetching is added if flag-based spatial locality detection mechanism is incorporated and only accesses to spatial blocks are allowed to initiate prefetches. This is very important for prefetching in shared memory multiprocessing environments, where the bus is usually the bottleneck.

Second set of workloads is several SPEC95 applications. Results are shown in the following tables. We did not present simulation results for STS caches because results vary substantially for different values of X and Y counter thresholds.

Results obtained from simulations using SPEC traces completely confirm conclusions obtained analyzing results from simulations using IBS traces.

Summaries of average memory latency and average bus traffic are presented on Figure 2 and 3.

	groff	gs	jpeg	mpeg	nroff	gcc	sdet	verilog	video	Mach (avg)
CNV 8K	1.224	1.394	1.456	1.748	1.194	1.312	1.734	1.931	1.919	1.546
CNV 8K PfA	1.164	1.219	1.281	1.354	1.097	1.222	1.484	1.289	1.467	1.286
CNV 8K 64B	1.299	1.440	1.511	1.726	1.212	1.410	1.844	1.783	1.872	1.566
STS 8K R11	1.242	1.343	1.535	1.706	1.211	1.362	1.785	1.400	1.736	1.480
STS 8K P11	1.233	1.370	1.589	1.712	1.176	1.417	1.754	1.382	1.726	1.484
SS/NS 8K	1.190	1.347	1.400	1.709	1.181	1.265	1.652	1.874	1.852	1.497
SS/NS 8K PfA	1.144	1.198	1.272	1.385	1.097	1.211	1.460	1.303	1.451	1.280
SS/NS 8K PfB	1.149	1.205	1.285	1.398	1.100	1.221	1.476	1.343	1.469	1.294

Table 3: Average memory latency on IBS Mach traces

	groff	gs	jpeg	mpeg	nroff	gcc	sdet	verilog	video	Ultrix (avg)
CNV 8K	1.195	1.248	1.262	1.344	1.135	1.295	1.530	1.651	1.907	1.396
CNV 8K PfA	1.133	1.137	1.084	1.144	1.046	1.188	1.230	1.420	1.313	1.188
CNV 8K 64B	1.255	1.293	1.236	1.311	1.133	1.363	1.512	1.722	1.771	1.400
STS 8K R11	1.188	1.187	1.250	1.297	1.062	1.300	1.384	1.577	1.480	1.303
STS 8K P11	1.180	1.211	1.322	1.308	1.058	1.348	1.387	1.551	1.461	1.314
SS/NS 8K	1.166	1.215	1.236	1.317	1.127	1.255	1.482	1.574	1.836	1.356
SS/NS 8K PfA	1.116	1.111	1.075	1.141	1.047	1.184	1.221	1.453	1.288	1.182
SS/NS 8K PfB	1.119	1.107	1.070	1.132	1.049	1.193	1.230	1.512	1.246	1.184

Table 4: Average memory latency on IBS Ultrix traces

	groff	gs	jpeg	mpeg	nroff	gcc	sdet	verilog	video	Mach (avg)
CNV 8K	0.022	0.049	0.054	0.115	0.023	0.031	0.092	0.172	0.141	0.078
CNV 8K PfA	0.048	0.086	0.124	0.206	0.039	0.062	0.183	0.226	0.238	0.134
CNV 8K 64B	0.035	0.067	0.087	0.162	0.032	0.049	0.146	0.201	0.190	0.108
STS 8K R11	0.079	0.147	0.291	0.439	0.126	0.085	0.378	0.316	0.485	0.261
STS 8K P11	0.063	0.111	0.237	0.423	0.099	0.056	0.336	0.294	0.466	0.232
SS/NS 8K	0.021	0.049	0.054	0.114	0.023	0.031	0.092	0.171	0.140	0.077
SS/NS 8K PfA	0.023	0.051	0.060	0.119	0.024	0.033	0.098	0.177	0.143	0.081
SS/NS 8K PfB	0.022	0.050	0.059	0.118	0.024	0.032	0.095	0.174	0.142	0.079

Table 5: Average bus traffic on IBS Mach traces

	groff	gs	jpeg	mpeg	nroff	gcc	sdet	verilog	video	Ultrix (avg)
CNV 8K	0.021	0.032	0.041	0.056	0.017	0.037	0.071	0.097	0.144	0.057
CNV 8K PfA	0.044	0.053	0.058	0.091	0.026	0.064	0.113	0.177	0.200	0.092
CNV 8K 64B	0.031	0.040	0.045	0.067	0.021	0.052	0.086	0.142	0.170	0.073
STS 8K R11	0.066	0.081	0.119	0.159	0.052	0.082	0.252	0.300	0.341	0.161
STS 8K P11	0.051	0.062	0.099	0.128	0.045	0.055	0.213	0.272	0.323	0.139
SS/NS 8K	0.020	0.032	0.041	0.055	0.017	0.036	0.071	0.097	0.142	0.057
SS/NS 8K PfA	0.022	0.033	0.042	0.056	0.017	0.038	0.077	0.104	0.144	0.059
SS/NS 8K PfB	0.021	0.033	0.041	0.056	0.017	0.037	0.076	0.101	0.143	0.058

Table 6: Average bus traffic on IBS Ultrix traces

	applu	apsi	fpppp	hydro2	mgrid	su2cor	swim	tomcat	turb3d	wave5	SPECfp(avg)
CNV 8K	2.038	2.225	1.124	4.261	1.860	2.804	5.085	3.523	1.532	3.875	2.833
CNV 8K PfA	1.251	1.983	1.111	1.054	1.040	1.112	4.680	2.569	1.409	2.680	1.889
CNV 8K 64B	1.654	2.310	1.189	3.056	1.541	2.147	7.731	4.279	1.602	4.191	2.970
SS/NS 8K	2.041	2.234	1.193	4.274	1.888	2.803	4.006	3.442	1.554	3.365	2.680
SS/NS 8K PfA	1.310	2.114	1.176	1.109	1.187	1.174	3.693	2.875	1.550	2.501	1.869
SS/NS 8K PfB	1.774	2.102	1.155	1.073	1.072	1.148	3.993	2.886	1.474	2.612	1.929

Table 7: Average memory latency on SPECfp traces

	compr	gcc	go	ijpeg	li	m88ks	perl	SPECint(avg)
CNV 8K	2.063	1.261	1.364	1.158	1.250	1.010	1.034	1.306
CNV 8K PfA	2.030	1.189	1.370	1.058	1.126	1.012	1.020	1.258
CNV 8K 64B	2.370	1.372	1.791	1.172	1.239	1.025	1.123	1.442
SS/NS 8K	1.974	1.216	1.254	1.191	1.253	1.007	1.000	1.271
SS/NS 8K PfA	1.950	1.171	1.238	1.106	1.195	1.007	1.000	1.238
SS/NS 8K PfB	1.948	1.179	1.237	1.107	1.195	1.007	1.000	1.239

Table 8: Average memory latency on SPECint traces

	applu	apsi	fpppp	hydro2	mgrid	su2cor	swim	tomcat	turb3d	wave5	SPECfp(avg)
CNV 8K	0.226	0.222	0.001	0.669	0.186	0.380	0.274	0.308	0.098	0.272	0.264
CNV 8K PfA	0.234	0.514	0.002	0.673	0.192	0.405	0.856	0.463	0.270	0.573	0.418
CNV 8K 64B	0.229	0.372	0.002	0.670	0.190	0.386	0.276	0.308	0.156	0.279	0.287
SS/NS 8K	0.226	0.221	0.001	0.669	0.186	0.378	0.274	0.308	0.092	0.272	0.263
SS/NS 8K PfA	0.227	0.244	0.001	0.670	0.190	0.384	0.305	0.337	0.117	0.290	0.276
SS/NS 8K PfB	0.227	0.237	0.001	0.669	0.188	0.381	0.284	0.322	0.095	0.273	0.268

Table 9: Average bus traffic on SPECfp traces

	compr	gcc	go	ijpeg	li	m88ks	perl	SPECint(avg)
CNV 8K	0.202	0.026	0.018	0.021	0.041	0.0002	0.000	0.044
CNV 8K PfA	0.414	0.048	0.101	0.023	0.056	0.0023	0.000	0.092
CNV 8K 64B	0.399	0.038	0.034	0.021	0.048	0.0004	0.000	0.077
SS/NS 8K	0.200	0.026	0.018	0.021	0.041	0.0002	0.000	0.044
SS/NS 8K PfA	0.202	0.028	0.019	0.021	0.046	0.0004	0.000	0.045
SS/NS 8K PfB	0.200	0.027	0.019	0.021	0.044	0.0003	0.000	0.044

Table 10: Average bus traffic on SPECint traces

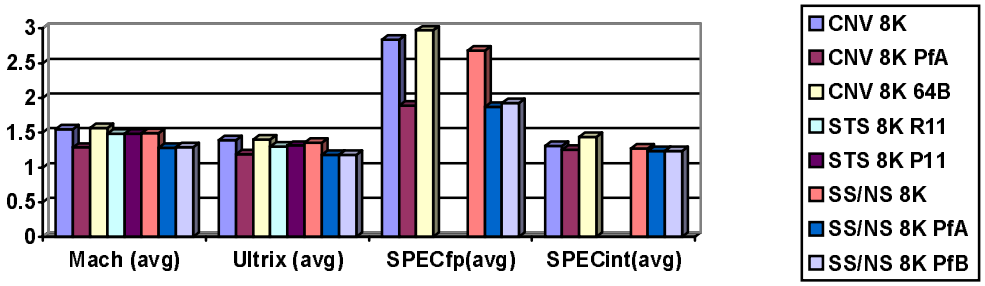

Figure 2: Average memory latency**.** Prefetching technique significantly reduces memory latency.

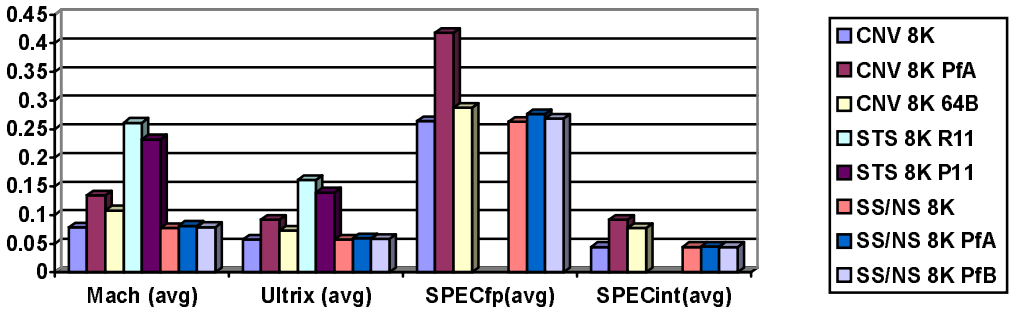

Figure 3: Average bus traffic. Prefetching in the conventional cache hierarchy significantly increases bus traffic, but causes only a negligible increase in bus traffic of the SS/NS cache.

Conclusions

A simple method of detecting useful spatial locality is proposed in this paper. This new method is tested by incorporating it into a new split cache design, called the Split Spatial/Non-Spatial cache (SS/NS). The average memory latency of this design was found to be better than that of a conventional cache of similar complexity, and better than that of a previous Split Temporal/Spatial split cache design. The new flag-based spatial locality detection mechanism overdid problem of determining counter thresholds present in STS cache. It was also shown that prefetching does not significantly increase bus traffic if prefetches are initiated only on access to blocks that are marked as spatial by the flag-based spatial locality detection mechanism. This finding may have important applications in all situations where memory or bus bandwidths are bottlenecks, particularly in shared memory multiprocessors.

Bus traffic in SS/NS cache may be reduced even more if secondary cache misses of non-spatial blocks fetch only the required sub-blocks. Besides, caching only small subblocks should significantly reduce false sharing in shared memory systems. This may be achieved by splitting the secondary cache, so the non-spatial secondary subcache fetches smaller sub-blocks. Another way is for the secondary cache to remain unified, but with each sub-block having its own state bits, while the tag is shared by the entire spatial block. In this way, the secondary cache becomes sectored with one-word blocks and four-word sectors. Exploration of these possibilities remains the topic for future research.

As was mentioned before, temporal locality detection similar to that in the NTS cache can be incorporated into SS/NS using the same four bits that are used for spatial locality detection, with only one additional bit per spatial block. In that way, the cache should be able to adapt to variations in both spatial and temporal locality and splitting according to both temporal and spatial locality can be done. This may lead to a primary cache being split into three or even four subcaches and it would be interesting to evaluate such caches.

Acknowledgments

The authors are thankful to their colleagues at the University of Belgrade: Jelica Protić, Aleksandar Milenković, and Igor Ikodinović for the help and numerous sugestions during the work on this paper.

References

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

[Gonzalez95] A. Gonzalez, C. Aliagas, and M. Valero, "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.

[Johnson97a] T. L. Johnson and W. W. Hwu, "Run-time Adaptive Cache Hierarchy Management via Reference Analysis," *Proceedings of the 24th International Symposium on Computer Architecture*, Denver, Colorado, June 1997.

[Johnson97b] T. L. Johnson, M. C. Merten, and W. W. Hwu, "Run-time Spatial Locality Detection and Optimization," *Proceedings of Micro-30*, Research Triangle Park, North Carolina, USA, December 1997.

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

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

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

[Prvulovic98] Prvulovic, M., Marinov D., Milutinovic V., "A Performance Reevaluation of the Split Temporal/Spatial Cache," *Workshop Digest of the PAID/ISCA-98*, Barcelona, Spain, June 1998.

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

[Sahuquillo99] Sahuquillo, J., Pont, A., "The Split Data Cache in Multiprocessors Systems: An Initial Hit Ratio Analysis," *Proceedings of the 7th Euromicro Workshop on Parallel and Distributed Prcessing*, Madeira, Portugal, February 1999.

[Reilly95] J. Reilly, "SPEC Describes SPEC95 Products And Benchmarks," *SPEC newsletter*, September 1995.

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

[Tomasko97] M. Tomasko, S. Hadjiyiannis, and W. A. Najjar, "Experimental Evaluation of Array Caches," *IEEE TCCA NEWSLETTERS*, March 97, pp. 11-16.

[Tse98] J. Tse and A. J. Smith, "CPU Cache Prefetching: Timing Evaluation of Hardware Implementations," *IEEE Transactions on Computers*, Vol. 47, No. 5, May 1998.

[Tyson95] G. Tyson, M. Farrens, J. Matthews, and A. R. Pleszkun, ``A modified approach to data cache management," *Proceedings of the 28th Annual International Symposium on Microarchitecture*, December 1995, pp. 93--103.

[Uhlig95] R. Uhlig, D. Nagle, T. Mudge, S. Sechrest, and J. Emer, "Instruction Fetching: Coping with Code Bloat," *Proceedings of the 22nd International Symposium on Computer Architecture*, Santa Margherita Ligure, Italy, June 1995, pp. 345-356.

