**Project CSN 221**

*Evaluating Caches through Multiple Caching Techniques*

Chinmaya Chawla

(18116025)

**Related Work**

**Overview**

Current data cache organizations fail to deliver high performance in scalar processors for many vector applications. There are two main reasons for this loss of performance: the use of the same organization for caching both spatial and temporal locality and the “eager” caching policy used by caches.

Many real programs, in particular numerical applications, make use of vector data. Vector processors, are aware of this data type and include some instructions that manipulate vectors. However, this is not the case of scalar processors, which usually do not have any specific instruction for vectors. In such processors, operations on vectors are performed by means of a sequence of scalar operations that manipulate individual elements of vectors. Although the computation power of current superscalar processors may be very effective for performing such operations, the processor performance is usually degraded by the low efficiency of the data cache memory.

The poor performance of current data cache organizations for dealing with vector data is mainly due to the following reasons:

1. Large working sets.
2. Pollution due to non-unit strides.
3. Interferences when the stride and the number of sets are not coprime.
4. Prefetching.

**DESCRIPTION**

As described in the previous section, there are four main issues to be considered when designing a data cache for a (super)scalar processor in order to achieve high performance when the processor is using vector data:

* Vector data usually require very large working sets which result in a sweep of the cache without any chance to exploit temporal locality.
* In some cases, vectors are accessed with a stride not equal to one. In those cases, the ability of a cache to exploit spatial locality is significantly reduced. Because conventional cache memories deal with lines of consecutive data, these strides cause a significant cache pollution. This negative effect increases as the stride is higher, and at the worst, it could end up with just one useful data element in every cache line.
* Finally, when the stride measured in cache lines and the number of sets in the cache are not coprime, not all the sets are used to store the elements of the vector, which in turn aggravates the problem caused by large working sets. For instance, suppose a direct mapped cache with 512 lines and a line size equal to the size of a vector element. Then, it can store a vector with stride 3 and length 512. However, if the stride is 2, the length of the vector can not be greater than 256. In other words, some strides make the vector accesses to interfere with themselves, even if the vector is smaller than the cache size.
* On the other hand, if the stride is constant, accesses to vector elements are quite easy to predict and therefore it is possible to implement a prefetch mechanism to take advantage of this fact.

***Solutions :***  In this section we summarize the most important solutions to deal with individual or a subset of these four issues.

* Block algorithms is a technique that tries to reduce the negative effect of large working sets. Block algorithms can be developed by the programmer or, as claimed by several authors.
* The pollution caused by non-unit strides can be avoided in some cases by means of copying . The main idea of this technique is to copy the elements of non-unit stride vectors into auxiliary vectors with stride one and then performing all the operations using the latter vectors.
* Many proposals to reduce the negative effect of interferences have recently appeared. Some of the most representative are: the victim cache , the Skewed-associative cache , the Column-associative cache , the Half-and-Half cache.
* Software prefetching techniques are managed by the compiler and usually make use of non-blocking caches in order to issue memory requests in advance. Hardware techniques for prefetching vector data require some mechanism to recognize references to vector data. A combination of both has also been proposed in few literatures.

**REFERENCES**

1. M.S. Lam, E.E. Rothberg and M.E. Wolf, The Cache Performance and Optimization of Blocked Algorithms in Proc. of ASPLOS 1991, pp. 67-74, 1991.
2. O. Temam, E.D. Granston, W. Jalby, To Copy or not to Copy: A Compile-time Technique for Assessing when Data Copying Should be Used to Eliminate Cache Conflicts, in Proc. of Supercomputing’93 Conference, pp. 410-419, 1993.
3. T-F. Chen and J-L. Baer, A Performance Study of Software and Hardware Data Prefetching Schemes, in Proc of the 21th. Int. Symp. Comp. Architecture, pp. 223-232, 1994.
4. G. Kurpanek et al. PA7200: A PA-RISC Processor with Integrated High Performance MP Bus Interface in Proc. of CompCon94, pp. 375-382, 1994.

**RESULTS AND CONCLUSIONS**

* The performance figures obtained for a set of benchmarks show that the dual data cache outperforms in many cases a conventional cache. It is even more remarkable the good performance of the selective cache, which consists of an ordinary cache plus a locality prediction table with a very few entries.
* We have also shown that for those cases where software techniques can be applied to improve the locality of a given algorithm (blocking and copying), the performance of the dual datacache is not degraded.For instance, in the case of matrix multiply with blocking and copying, the performance of the dual cache is about the same as that of a conventional cache .
* Then we moved to a newly proposed caching technique which is Dual locality caching we identify a serious weakness in spatial locality exploitation in I/O caching and propose a new and effective memory management scheme, DULO, which can significantly improve I/O performance by exploiting both temporal and spatial localities. Our experiment results show that DULO can effectively reorganize applications’ I/O request streams mixed with random and sequential accesses in order to provide a more disk-friendly request stream with high sequentiality of block accesses .
* We analysed an effective DULO algorithm to carefully trade off random accesses with sequential accesses. The results of performance evaluation on both buffer cache and virtual memory systems show that DULO can significantly improve a system’s I/O performance.