Evaluating caches through multiple caching techniques

Submitted to: Satish Kumar Peddoju

Submitted by: Maninder Singh (18114047)

Nishant Jain (18114052)

Rhythm Gothwal (18114065)

Ujjwal Bagrania (18114078)

Chinmaya Chawla (18114025)

Prakarsh Singh (18114061)

DESCRIPTION:

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. The first issue has led to the well-known trade-off of designing caches with a line size of a few tens of bytes. However, for memory reference patterns with low spatial locality a significant pollution is introduced. On the other hand, when the spatial locality is very high, larger lines could be more convenient. The eager caching policy refers to the fact that data that miss in the cache and is required by the processor is always cached (excepting writes in a no write allocate cache). However, it is common in numerical applications to have large working sets (large vectors, larger than the cache size), that result on a swept of the cache without any opportunity to exploit temporal locality. In addition, they replace some other data that may be required later. In this project we look at and compare all the caching types and discuss their benefits and losses and also discuss alternative caching strategies overcoming aforementioned disadvantages.

MOTIVATION :

We studied about caches in our course structure and came to know about their working and their importance in reducing the access time to memory . We were willing to find out what happens when we use or work with very large data sets when we learnt about how cache exploits spatial and temporal locality . It is interesting to know how different kinds of caching techniques solve the problem for performance in vector applications , how different data caching techniques exploit spatial as well as temporal locality to reduce access time as well as to increase hit/miss ratio .

The topic is important as it helps in spatial locality detection and optimisation and it also throws some light on working of locality prediction table . we have also worked on evaluating their performances on different benchmarks which are currently being used .
Also currently there is no proposal to solve the four major issues - large
working sets, strides different to one, interferences and prefetching
. In this paper, we present a novel cache organization that incorporates some features to exploit the characteristics of vector as well as scalar data. In particular, our cache organization tries to avoid the swept effect of large vectors, reduces the pollution caused by non unit strides, decreases the penalty caused by self-interferences due to strides coprime with the
number of lines, and can take advantage of prefetching of vector elements

Related Work

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.

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.

Runtime spatial locality detection :

To facilitate spatial locality tracking, a spatial counter, or sctr, is included in each MAT entry. The role of the sctr is to track the medium to long-term spatial locality of the corresponding macroblock, and to make fetch size decisions .

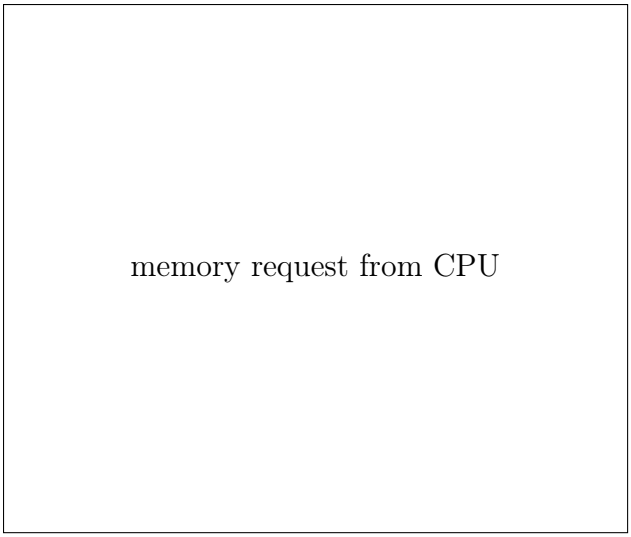
If the cache is not fully-associative, the tags for different blocks residing in the same larger fetch size block will lie in consecutive sets. Searching for other cache blocks in the same larger fetch size block of data will re-
quire access to the tags in these consecutive sets, and thus either additional cycles to access, or additional hardware
support.

a separate structure can be used to detect this information, which is the approach investigated in this work.

This structure is called the Spatial Locality Detection Table (SLDT), and is designed for e_cient detection of spatial
reuses with low hardware overhead. The role of the SLDT is to detect spatial locality of data while it is in the cache,
for recording in the MAT when the data is displaced. The SLDT is basically a tag array for blocks of the larger fetch
size, allowing single-cycle access to the necessary information.

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



2

Methods :

Dual data cache

The proposed data cache, which is called *dual data cache*, consists of two independent memories, or subcaches. One is called the *spatial cache* because it is designed to exploit spatial locality, in addition to temporal locality. The other is called *temporal cache* since it is targeted to exploit just temporal locality. Both subcaches

work independently and in parallel.

When the processor issues a memory reference, both caches are looked up at the same time and, depending on the result, one of the following actions is taken:

• If the required data is only in one of the subcaches, the data

is read or written in that subcache. This is a cache hit .

If the required data is found in both subcaches, it is read from the temporal cache or written into both two in parallel. This is again a cache hit.

• If the required data is not in any subcache, a cache miss occurs. In this case, the processor is stalled and the required data is brought from the next level of the memory hierarchy.

This data may be placed in just one of the two subcaches or may be not cached anywhere, depending on predicted type of locality for this memory access. The predicted locality for a memory reference is based on guessing whether the accessed data is an scalar, or an element of a vector, and in the latter case, it also depends on the stride and the size of the vector. These attributes are estimated by means of the locality prediction table. For every cache miss, the locality prediction table decides where the missed data is cached. The locality prediction table is accessed at the same time as the cache memory.

The locality prediction table is managed as a cache. It may be direct mapped, set associative or fully associative. Every time a load/store instruction is executed, the locality prediction table is looked up.

Selective caching

a selective cache behaves as a conventional cache with a selective caching policy .the caching scheme used is cache bypassing . The locality prediction table behaves in the same way with the difference that now, there is not distinction between spatial and temporal caches. In this profiling is done in order to identify heavily used basic blocks.

The basic block usage frequencies are classified to high, medium and low usage. Based on input compiler marks HU instructions cacheable to level 1,MU to level; 2 and LU to level 3.this is done so that heavily used code is cached and less used is bypassed to processor .

Based on the profile input, the compiler marks HU instructions as cacheable to level 1, MU instructions as cacheable to level 2, LU instructions cacheable to level 3 etc. The essence of the technique is to allow only heavy usage sections of the code to be cached and to bypass rarely used code directly to the instruction register or data registers.

The used technique emphasises temporal locality, however it does not ignore spatial locality. Use of large block sizes can

help to exploit spatial locality in frequently used sections of code or data. Non-caching of rarely used items will mean that spatial locality in them cannot be exploited.

Consider a case where two elements A and B within a single loop compete for space in cache. If the loop is executed ten times the memory access pattern may be represented as (AB)10, where 10 denotes frequency of usage of particular instruction. If we allow both elements to enter the cache, each instruction will knock each other out of the cache and neither hits. Hence behaviour of conventional cache is

$$(A_m B_m)^{10}$$

where m is a miss and h is a hit. Hence miss rate for conventional cache is 100% . Instead of allowing every element to enter the cache, let us keep one element say A. The behaviour of bypassing or selective cache is

$$A_m B_m (A_h B_m)^9$$

and the miss rate is 55% .

Performance evaluation for dual data and selective caching

Three different kinds of benchmarks have been used :

1. synthetic benchmarks :

The synthetic benchmarks consist of a set of loops that deal with vectors of different size and stride. Each loop makes use of either one or two vectors and traverse it/them ten times incrementing every element at each iteration.

Bench	Length	Stride	S-Loc	T-Loc	SS-Loc	TS-Loc	TT-Loc
v1	2000	1	3/4	100	3/4	100	-
v2	4000	1	3/4	100	3/4	2	-
v3	15000	1	3/4	0	3/4	0	-
v4	1000	5	0	100	-	-	100
v5	1000	15	0	13	-	-	100
v6	2500	6	0	0	-	-	0
v7	1000	12	0	100	-	-	0

For 32 kbyte cache

2. kernels

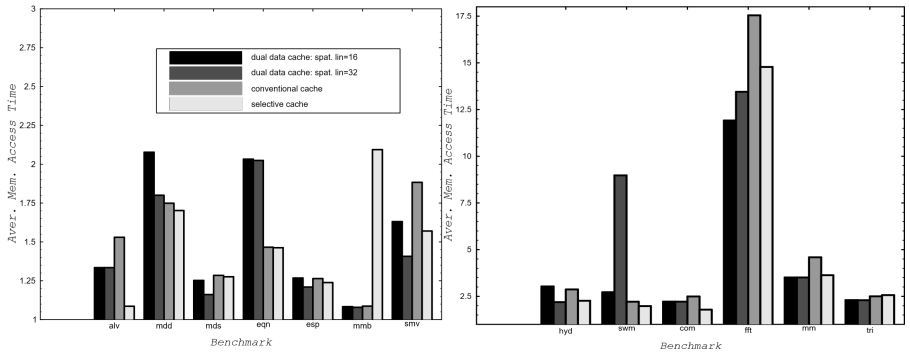
The kernels are a set of routines frequently found in numerical applications:

- *fft*: A Fast Fourier Transform of a 2^{19} element input vector. This is a vectorizable code with strides power of 2.
- *mm*: A matrix by matrix multiplication. A vectorizable code with unit and non-unit strides
- *mmb*: A block algorithm for matrix by matrix multiplication, optimized to make an optimal use of a 16 Kbyte conventional cache.
- *smv*: A sparse matrix by vector multiplication. A non-vectorizable code.

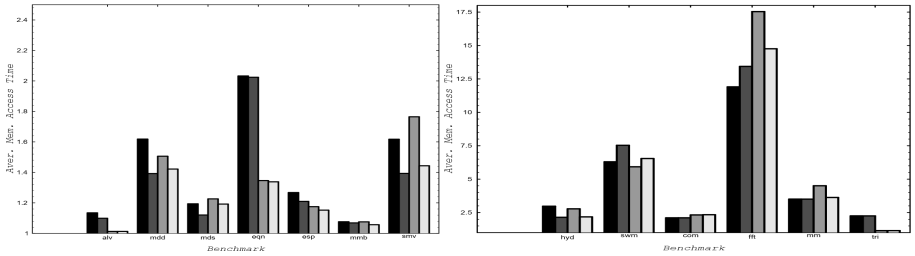
tri: A triangular matrix multiplication

3. spec benchmarks

The following benchmarks from the SPEC 92 benchmark suite have also been used: *compress, eqntott* and *espresso* (from SPECint92) and *alvinn, hydro2d, mdljdp2, mdljsp2* and *swm256* (from SPECfp92). We will refer to them as *com, eqn, esp, alv, hyd, mdd, mds* and *swm* respectively



For 16 kbyte cache



For 32 Kbyte cache

For the *FFT* benchmark, the dual data cache has much better performance than a conventional cache. This benchmark makes vector accesses with strides that are powers of 2

On the other hand, the dual data cache do not cache those vectors that are going to interfere and then it benefits from the shorter miss penalty due to the shorter line size.

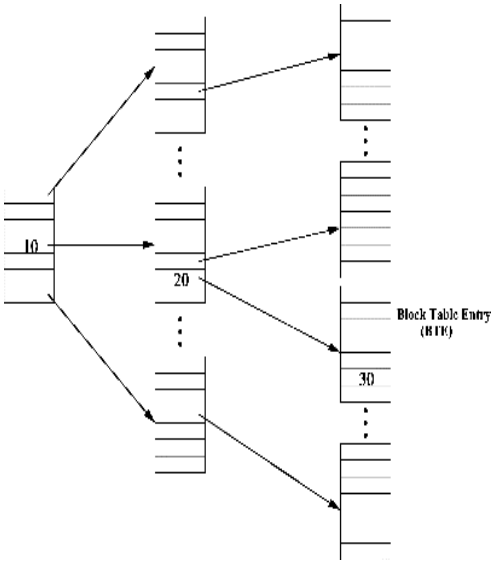
The dual data cache may have a worse performance than a conventional cache for some memory reference patterns. This happens when almost all memory references exhibit spatial locality. In this case, almost all the data is cached into the spatial cache.

In some other cases, the dual data cache is a little bit worse than the conventional cache (*eqntott* always; *alv, mdd* and *esp* for some configurations). However, it is remarkable the good performance of the selective cache, which is always better than the conventional cache with a very few exceptions.

Dual locality caching

we build the DULO scheme by
using theLRUalgorithm and its data structure—theLRUstack—as a reference

point.



In LRU, newly fetched blocks enter into its stack top, and replaced blocks
leave from its stack bottom. There are two key operations in the DULO scheme.
(1) Forming sequences is one of the key operations. A *sequence* is defined as a
number of blocks whose disk locations are close to each other and have been

accessed continuously without an interruption during a limited time period.

2 Sorting sequences in an LRU stack according
to their recency (temporal locality) and size (spatial locality) with the objective

that sequences of large recency and size are close to the LRU stack bottom.

Block table is the data structure for implementing dual locality.
The block table is analogous in structure to the multilevel page

table used to process address translation.
There are three levels in the example block table: two directory levels and one
leaf level. The table entries at different levels are fit into different memory pages. An entry at the

leaf level is called Block Table Entry (BTE).
the block table covers disk space in the unit of block

where a logical block number (LBN) of a block is the index into the table.
In the system, we set a global variable called a *disk access clock*, which ticks
each time a block is fetched into memory and stamps the block being fetched
with the current clock time . We then record the timestamp in an entry at the

leaf-level of the block table, which is determined by the LBN of the block.
When the sequencing bank is full, it is time to examine blocks in the bank to

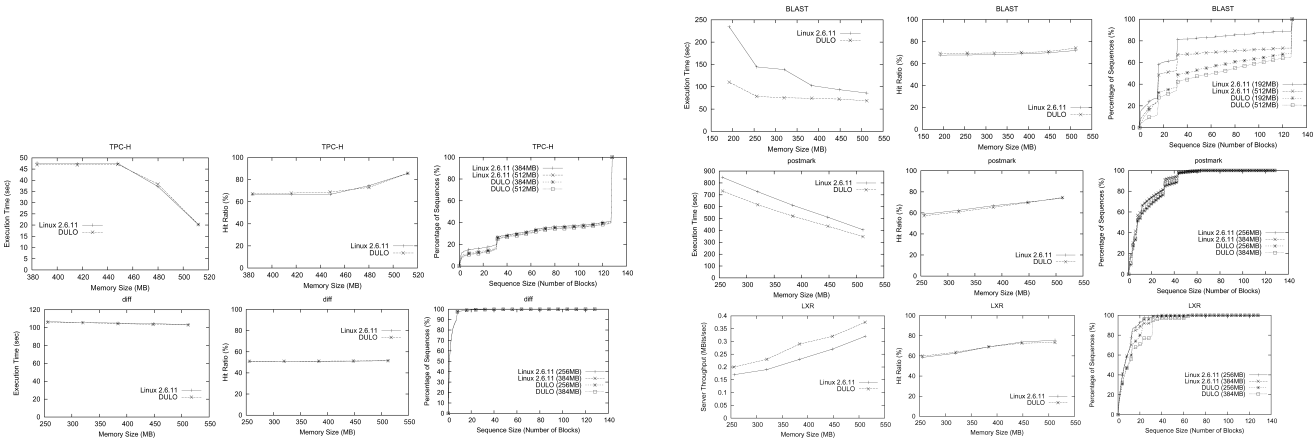
aggregate them into sequences.
The DULO algorithm associates each sequence with an attribute H, where a relatively small H value indicates its associated sequence should be evicted earlier. The algorithm has a global inflation value L, which is initiated as 0. When a new sequence s is admitted into the eviction section, its H value is set as $H(s) = L + 1/size(s)$, where $size(s)$ is the number of the blocks contained in
s. When a sequence is evicted, we assign the H value of the sequence to L. So L records the H value of the most recently evicted sequence. The sequences in the eviction section are sorted by their H values with sequences of small H values at the LRU stack bottom. In the algorithm, a sequence of large size tends to stay at the stack bottom and to be evicted earlier. However, if a sequence of small size is not accessed for a relatively long time, it would be replaced. This is because a newly admitted long sequence could have a larger H value due to the L value, which keeps being inflated by evicted blocks. When all sequences
are random blocks (i.e., their sizes are 1), the algorithm degenerates into the LRU replacement algorithm.

Evaluation

To demonstrate the performance improvements of DULO on a modern operating

system, we implement it in the recent Linux kernel 2.6.11

Benchmarks
1. *TPC-H* is a decision support benchmark that runs business-oriented queries against a database system
2. diff is a tool that compares two files in a character-by-character fashion
3. BLAST (basic local alignment search tool) is software from the National Centre for Biotechnology Information
4. *Postmark* is a benchmark designed by Network Appliance to test performance of systems,
5. *LXR* (Linux cross-reference) is a widely used source code indexer and crossreferencer [LXR].



In Figure 4, the CDF curves show that in workload TPC-H more than 85% of the sequences are longer than 16 blocks. For this almost-all-sequential workload, DULO has limited influence on the performance. It can slightly increase the sizes of short sequences, and accordingly reduce execution time by 2.1% with a memory size of 384MB. However, for the almost-all-random workload *diff*, more than 80% of the sequences are shorter than 4 blocks. Unsurprisingly, DULO cannot create sequential disk requests from application requests consisting of purely random blocks. As expected, we see almost no improvements of execution times by DULO. The other three benchmarks have a considerable amount of both short sequences and long sequences

Results and conclusion :

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 data cache 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