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.

**Methods :**

**Dual data cache**

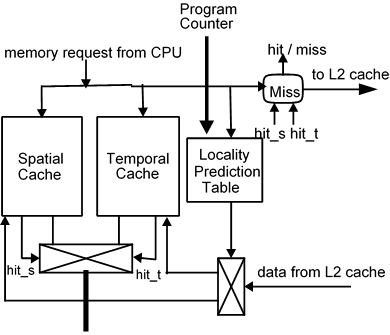
The proposed data cache, which is called *dual data cache*,

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.

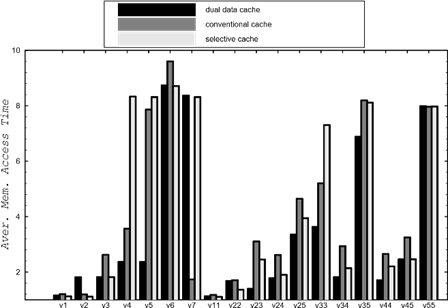
Consider a case where two elements A and B within a single loop compete for space in cache. If the loop is executed ten times the memory access pattern may be represented as (AB)10, where 10 denotes frequency of usage of particular instruction. If we allow both elements to enter the cache, each instruction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

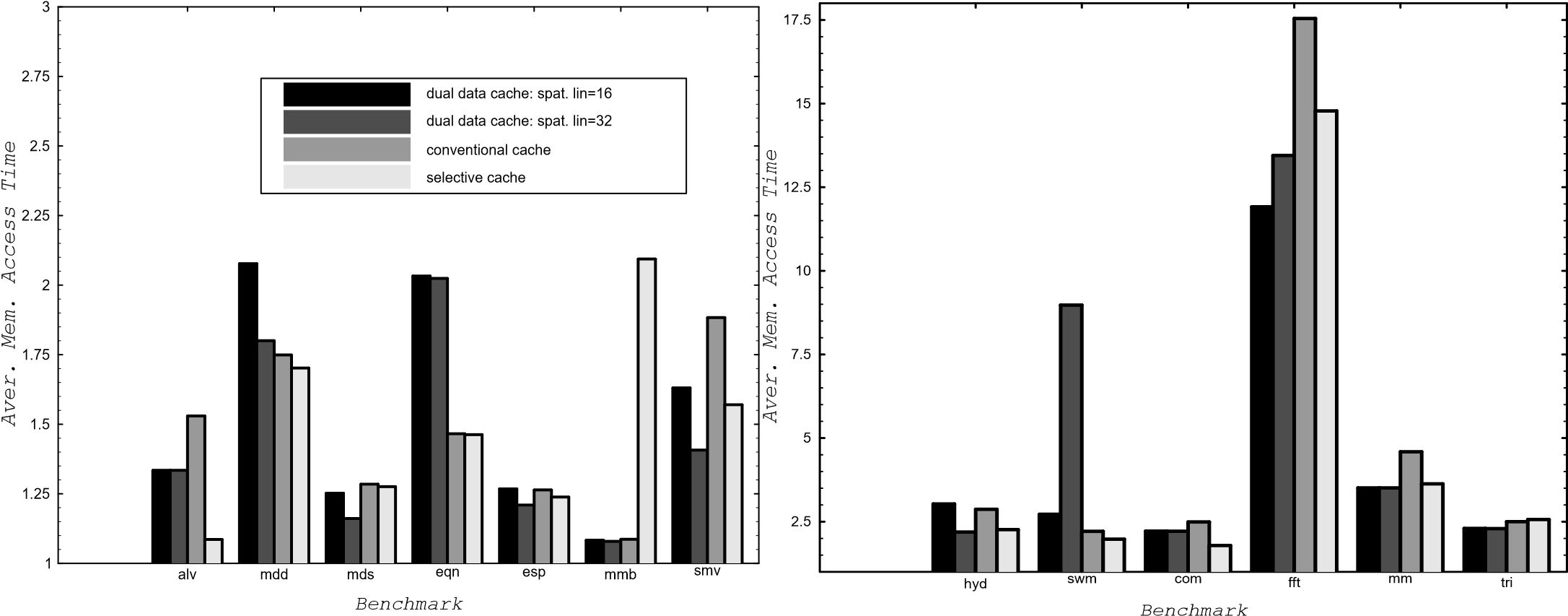
(AmBm)10

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

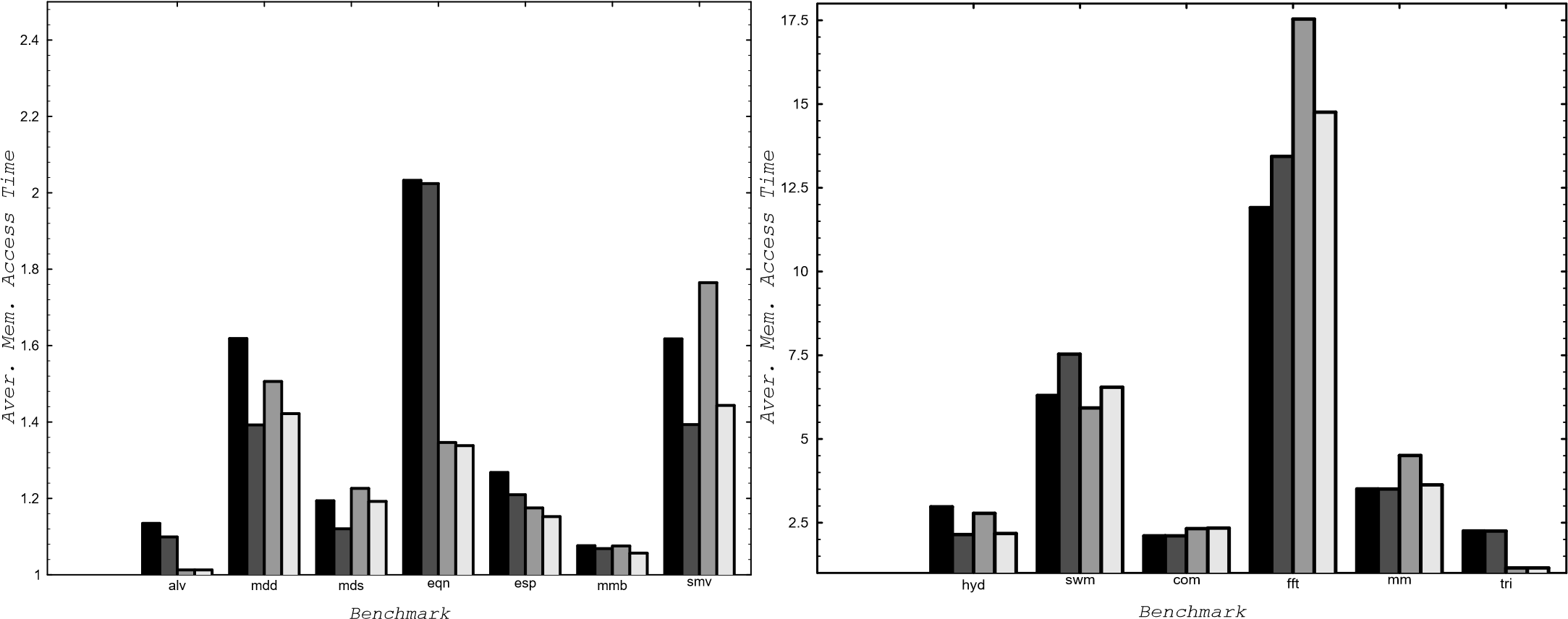
AmBm(AhBm)9

and the miss rate is 55%.





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.