HIGH PERFORMANCE COMPUTING -2005
POSTER PRESENTATION
SUMMARY OF WORK

DYNAMIC SET ASSOCIATIVE MAPPING POLICY

Submitted by,

A.S.Saddique
M.Selvam
S.Sudharsan

B.E. C.S.E. IV year
College of Engineering, Guindy
Anna University
Chennai
Tamil Nadu

TITLE: Dynamic Set Associative Mapping Policy

INTRODUCTION:

The Direct mapping policy in caches is the simplest idea to map the memory (of higher capacity) in a lower hierarchy to a memory (of lower capacity) in a higher hierarchy. It uses modulo technique to map every j^{th} block to a same block. This however increases conflict misses. To reduce it, a more flexible method of associative mapping was introduced. In this method any block can map on to any block in the cache. So there would not be much conflict misses. The tradeoff here is the time it takes to search the tag fields of the entire cache. The tag entry size is more and for every access, all the entries have to be compared. It is a costly approach to achieve flexibility. So a hybrid method was found that would combine the flexibility of the associative mapping and the simplicity of the direct mapping. It is set associative mapping. The entire cache is divided into sets each consisting of same number of blocks. The mapping between a block in secondary memory and the set in the cache is direct whereas the mapping inside a set is associative. A block destined to be placed in a set can be placed anywhere in the set.

We propose a new policy to overcome the problems with the set associative policy. Set associative mapping works good if the degree of association is already known. But different programs need different degree of association. Our idea is to vary the degree of the association according to the cycle of references in the program.

We have organized the remainder text as follows. The idea is explained in the section 2. The issues and it implementation are explained in section 3. The performance chart is depicted in the section 4. We also prove the effectiveness of our idea in achieving the performance of the cache with higher degree of association.

IDEA:

In the case of static associative mapping, if a particular set (we call them beneficiary set hereafter) gets accessed repeatedly than any other then it may happen that the blocks are overwritten and they are needed again. Thus it would increase the number of set misses. If the degree of association is increased then it would reduce the number of misses. But on the other hand it would increase the access time for it takes more time to search within all the sets as degree of association increases. Moreover other sets may not be accessed that frequently and the lower degree of association may suffice for them.

So we identify such beneficiary sets and extend them to accommodate more number of blocks rather than increasing the degree of association uniformly for all the sets. Our idea instead of fixing the degree of association, allows that to be modified according to the program in execution. So it adds more flexibility to the set associative mapping policy.

When more number of misses occurs in a set we extend it and that set alone would have a higher degree of association. The extension is provided by a victim set which would accommodate the additional blocks. When a block in the victim set is accessed, the extension is rolled back and it is treated as a normal cache miss. The beneficiary set will have to find another victim set upon a miss occurring in it.

IMPLEMENTATION:

For the above idea to be implemented, following issues are to be considered.

- Deciding on when to extend a set.
- Identifying the victim set.
- Linking the sets.
- When to unlink the sets.

Deciding on when to extend a set:

A threshold value is arrived based on the associativity of the cache and various bench mark programs. The number of misses in each set is kept track of as the program is executed. When the misses in a set exceed the threshold value the set can be treated as a beneficiary set and extended to accommodate more number of blocks.

Identifying the victim set:

A victim set has to be chosen to accommodate for the extra blocks of the beneficiary set. The victim set should neither be a beneficiary set by itself nor be a victim set already. The set with minimum number of hits is chosen as the victim set. The history information of number of hits in a set is used to identify the victim set.

Linking the sets:

The victim set is chosen as above. Then the flags meant for identifying the sets as victim set and beneficiary set are set appropriately. Then the head block of the victim set is linked to tail block of the extending set. The tail of the extending set is changed to tail of the victim set in order. These head & tail manipulations are done to maintain the LRU information to choose the replacement block.

When to unlink the set:

If the victim set is accessed for its original blocks then the beneficiary set is unlinked from the victim set. Unlinking involves undoing the linked list manipulations. Then the flags are also reset to represent that they are not linked anymore.

DESCRIPTION:

We implemented the proposed idea using the SimpleSim3.0 for the dl1 cache. We added our modules to the file cache.c and the corresponding macros, prototypes in the file cache.h. We had to include modules to find a victim set and roll back the linking done while extension. We had to add the following counters and flags

SetMisscount - to keep track of no. of misses in each set to decide when to extend a set
SetHitcount - to keep track of no. of hits in each set to choose the victim set
Threshold value -to decide upon whether to extend a set or not
Isextended flag - to identify whether a set is already extended or not
Isvalid flag - to identify whether a set is valid or not
extensionof -to identify the extended set of a victim set.

As the program is executed using the simulator, the set miss count and the set hit count are updated accordingly. When a miss occurs, the number of set misses is compared with the threshold value. If it had exceeded it is treated as a beneficiary set and findvictimset module is invoked. The set with the least number of setHitCount is chosen as the victim set.

The flags are then set as victim set to be invalid and the extension of variable is set to the set number of the extended set. Also the extended flag is set for the extended set. Then the sets are linked by manipulating the head & tail pointers of the sets appropriately. Now, the associativity of the beneficiary set alone has been increased.

Then for every miss that occurs it is checked whether it is an invalid set. If so then the set is unlinked from the extended set by invoking the rearrange module. Then all the flags are reset. So by unlinking the associativity of the beneficiary set has again come to original value.

Thus the degree of association of the sets is altered dynamically and the performance of higher associativity is obtained without increasing the associativity overall but for those sets which are accessed more frequently. Thus if by the property of the program, it happens to be a block being accessed at frequent intervals with some blocks of same set getting accessed in between then it will not raise unnecessary misses as some blocks will be already present in the set.

BENCHMARK PROGRAMS USED:

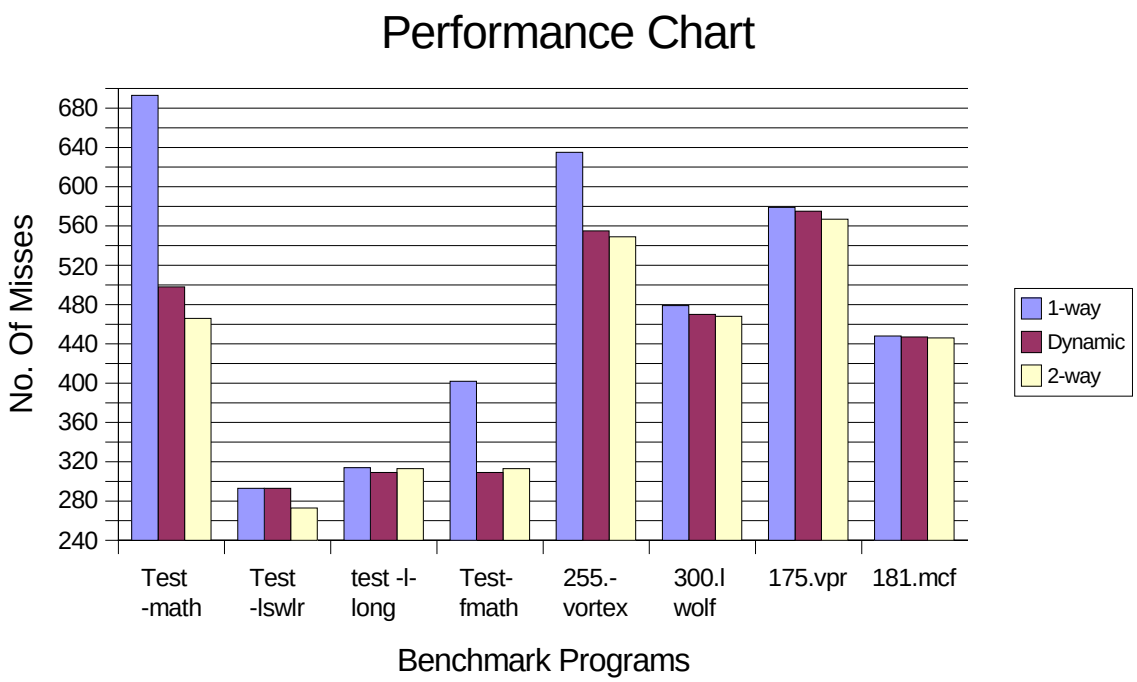
ALPHA:

Test-math
Test-lswlr
Test-fmath
Test-llong

PISA:

SPEC - 2000:

255.vortex
300.twolf
175.vpr
181.mcf



INFERENCE:

The above chart depicts the number of misses that occur in a static 1-way associative cache (direct) and dynamic 1-way associative cache and static 2-way associative cache. From the above graph it is clear that our idea of Dynamic Set Association in 1-way associative cache almost simulates the 2-way associative cache for the afore-mentioned benchmark programs.

We know by 2:1 thumb rule the performance of the m-way cache of cache size 'n' is same as the performance of 2m-way cache of cache size 'n/2'. The number of sets in our implementation was

1-way cache: 256
2-way cache: 128(so that cache size remains the same).

We obtained the following result.

Performance achieved by 1-way associative cache: **99.133752 %** of 2-way cache
Performance achieved by dynamic 1-way associative cache: **99.585879 %** of 2-way cache

Thus our idea provides more flexibility and it achieves the performance of a cache with the higher degree of association varying the degree of association dynamically according to the program executed.