RC23332 (W0409-045) September 8, 2004
Computer Science

IBM Research Report

Generalized Tree-LRU Replacement

John T. Robinson
IBM Research Division
Thomas J. Watson Research Center
P.O. Box 218
Yorktown Heights, NY 10598

Generalized Tree-LRU Replacement

John T. Robinson
<robnson@us.ibm.com>
IBM Research Division
Thomas J. Watson Research Center
PO Box 218
Yorktown Heights, NY 10598

August 23, 2004

Abstract. A generalization of the tree-LRU (pseudo-LRU, PLRU) method for cache line replacement selection in an N-way set-associative cache is described. In this method, a line is selected for replacement within a set using a logical multi-way tree, where the leaf nodes of the tree represent lines in a set, and where if the branching factor of a given node in the logical tree at a given level is k, then the state of the node is represented by one of k! (k factorial) logical states representing an access order of the nodes below the given node in the logical tree. Results showing the degree to which these methods approximate LRU replacement are presented.

This material is based upon work supported by the Defense Advanced Research Projects Agency (DARPA) under Contract No. NBCH3039004.

1. Introduction

Consider an N-way set-associative cache. For small N (e.g. N=4), it is feasible to select the LRU (least recently used) line for replacement within a set since maintaining the access order does not require a large number of states (for N=4 there are 4! = 24 states, and a state machine can be used, with 5 bit state numbers; there are alternative implementations as well for small N). However, in newer processor architectures, it may become desirable for a variety of reasons (for example multi-threaded processor designs) to increase the cache associativity at various levels of the cache hierarchy. For larger values of N (e.g., N=8, 16), maintaining the full access ordering becomes impractical since the number of states increases exponentially (for example, 8! = 40320; 16! = 20922789888000). A well-known solution is to approximate LRU using for example the tree-LRU replacement selection method (also known as pseudo-LRU or PLRU). In this method, one bit indicates whether the most recent reference is to a line in either the first half or the second half of the lines in a set; then, this technique is logically applied recursively, resulting in a logical binary tree with N-1 nodes (thus N-1 bits are required to represent a state in tree-LRU).

Tree-LRU does not approximate LRU particularly well, as shown in Figure 1.1 for the case N=16. These results were generated as follows: 100,000 random permuations of 0,1,2,3,...,15 were generated; for each permutation, the lines in a set were accessed in this order, generating a particular tree-LRU state; finally, the line selected for replacement using tree-LRU was found, its position in the full LRU ordering of the set was found (where 0 was the LRU line, 1 was the 2nd LRU line, etc.), and the statistics were updated.

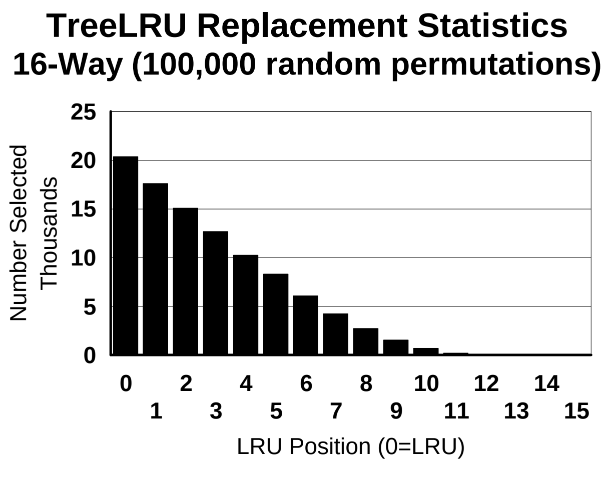

Figure 1.1

For comparison, if the LRU line had always been selected, the graph shown in Figure 1.1 would have had a single bar at LRU position 0 of height 100,000.

Although it is true that the probability of re-reference of a line in a set (where the probability is computed over all sets in the cache) decreases sharply with increasing age of the line within the set, it is also the case that a certain small fraction of sets are typically "hot" and generate a proportionately large fraction of all cache misses. Therefore one would expect that a method which approximates LRU more closely might give non-trivial performance improvements (in terms of decreasing cache misses for the "hot" sets). However an improved method is only practical if it can be implemented efficiently. Generalized tree-LRU methods are described here that can approximate LRU more closely than tree-LRU, with typically only small increases in complexity, and that have (in most cases) efficient implementations.

2. Generalized Tree-LRU

For the sake of a concrete example, including a precise comparison (in terms of complexity) with tree-LRU, first a special case of the more general method will be described for N=16 (i.e. a 16-way set-associativie cache), with a branching factor of four for the logical tree. The 16 ways in a set are divided into 4 subsets with four distinct ways in each subset. Note that the 4! orderings of 4 logical entities can be represented using 5 bits (since there are 24 states), and the state transitions can be implemented using a state machine (again, there are alternative implementations). The root node of the logical tree is in one of 24 states, representing the order in which the four subsets have been accessed. In an implementation, the subset number could be determined using the first two bits of the way number (where the way number is in the range 0, 1, 2, 3, ..., 15). For each such subset, a state is maintained representing the order in which the four lines in the subset have been accessed; since there are four such lines, this state also requires 5 bits, and state transitions among the states can be implemented using a state machine. In an implementation, the line number (0, 1, 2, 3) of a line in a subset could be given by the last two bits of the way number. The total bits required to represent a state are 5 + 4x5 = 25 bits, as compared to 15 bits for tree-LRU. Furthermore, state transitions are easy to implement, for example using logic to compute state transitions for two identical state machines with 24 states (one for the logical root node, and one for computing the state transistion for the state representing the access order of the lines in a subset). Repeating the experiment described above for tree-LRU using this hybrid tree-LRU / LRU method, results are obtained as shown in Figure 2.1.

As can be seen by comparison with Figure 1.1, an approximation to LRU replacement is obtained that is much closer than that using tree-LRU. Furthermore, the additional complexity is relatively small (25 bit states instead of 15 bit states; state transition logic using two state machines for 24 states, each of which implements 4*24 = 96 state transitions, where unchanged state transitions have been included).

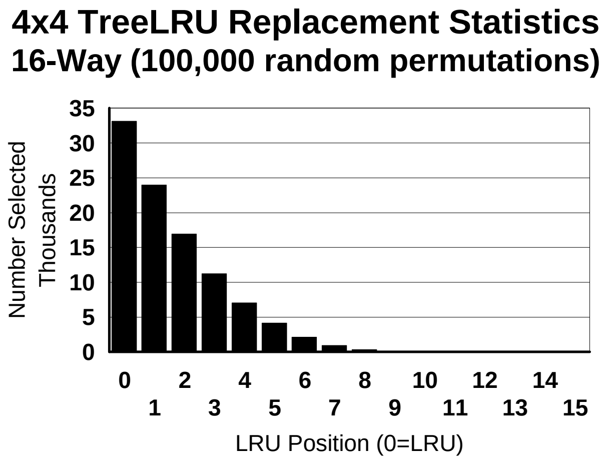

Figure 2.1

It is easy to see how to generalize the above to the general method. Tree-LRU for the case N=16 can be considered a special case of generalized tree-LRU in which the tree is logically a 2x2x2x2 tree; the results shown in Figure 2.1 are for a 4x4 tree. Limiting the branching factor to 4 or less, other possible trees for the N=16 case are 2x2x4, 2x4x2, and 4x2x2. Similarly, for a 12-way set associative cache for example (i.e. for the case N=12), possible generalized tree-LRU organizations include 4x3, 3x4, 2x2x3, 2x3x2, and 3x2x2 trees, and similarly for other values of the degree of set associativity N. In the next section results are presented showing how closely generalized tree-LRU approximates LRU for all possible logical trees for the cases N = 8, 12, and 16.

3. Degrees of LRU Approximation for Generalized Tree-LRU

In this section graphs are shown illustrating the degree to which generalized tree-LRU approximates LRU for all possible logical trees, for several cases (N = 8, 12, 16). First consider the case in which the degree of set-associativity is 8. There are three possible logical trees: 2x4, 4x2, and 2x2x2. The results for these three cases are shown in Figures 3.1a-3.1c (in these and all other graphs in this section, the units on the Y-axis are in thousands, showing the number of lines selected at that LRU position out of 100,000 random permutations, or alternatively percent selected).

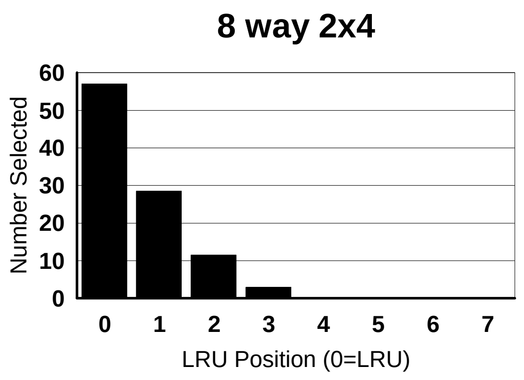

Figure 3.1a

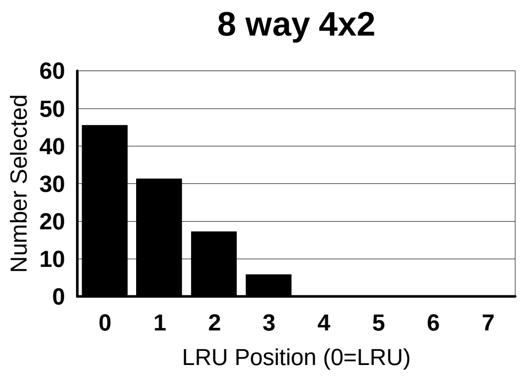

Figure 3.1b

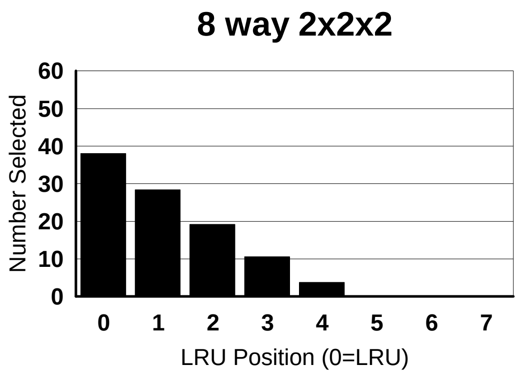

Figure 3.1c

Next consider the case in which the degree of set-associativity is 12. There are seven possible logical trees: 2x6, 6x2, 3x4, 4x3, 2x2x3, 2x3x2, and 3x2x2. The results for these seven cases are shown in Figures 3.2a-3.2g. Although 2x6 and 6x2 trees are probably not practical (for implementation), they are included for completeness.

Finally consider the case in which the degree of set-associativity is 16. There are again seven possible logical trees: 2x8, 8x2, 4x4, 2x2x4, 2x4x2, 4x2x2, and 2x2x2x2. The results for these seven cases are shown in Figures 3.3a-3.3g. Similarly to the 2x6 and 6x2 trees for N=12, the 2x8 and 8x2 trees are impractical for implementation, but are shown for completeness.

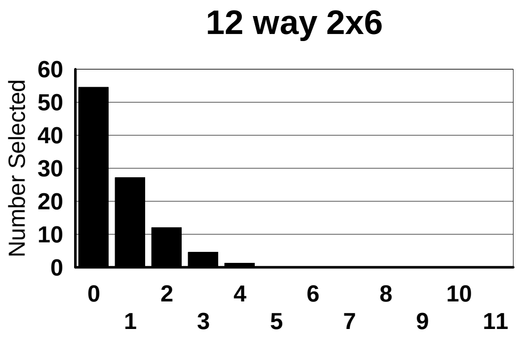

Figure 3.2a

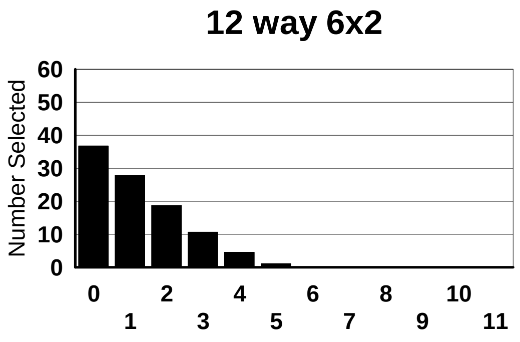

Figure 3.2b

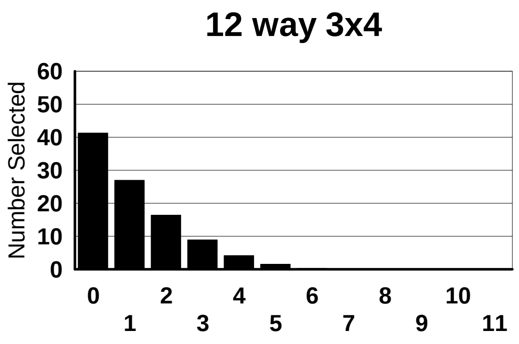

Figure 3.2c

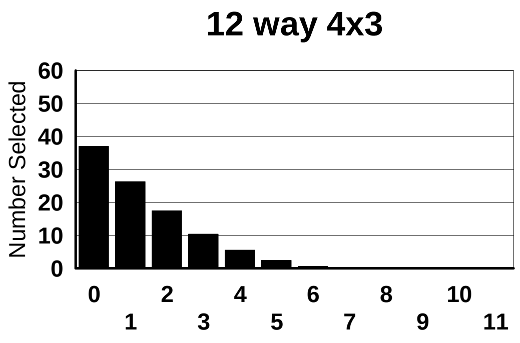

Figure 3.2d

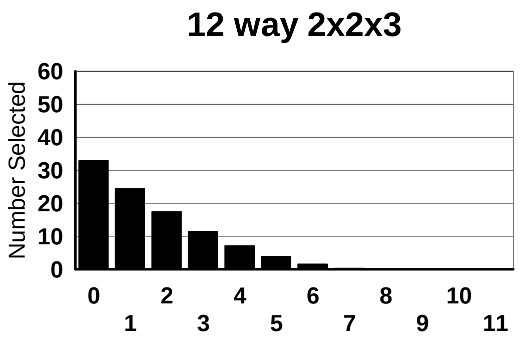

Figure 3.2e

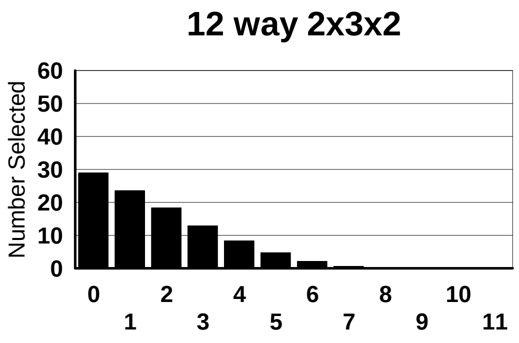

Figure 3.2f

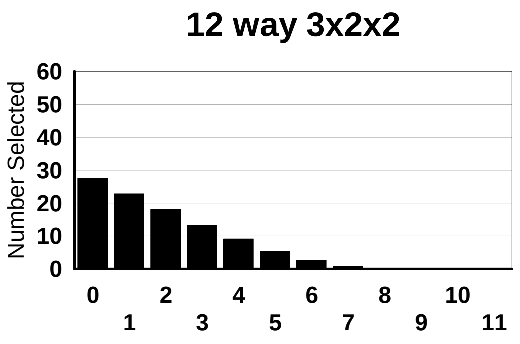

Figure 3.2g

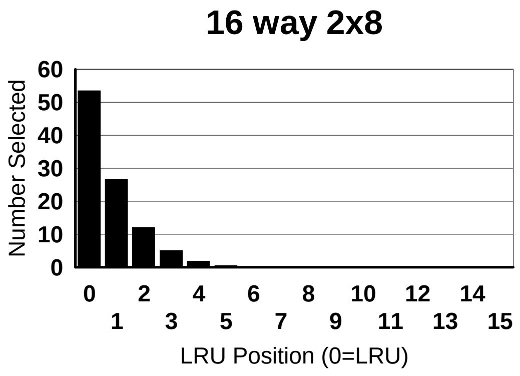

Figure 3.3a

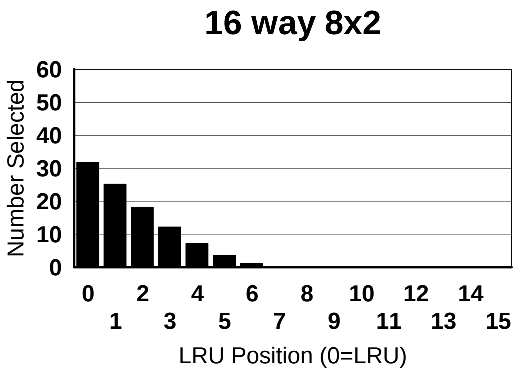

Figure 3.3b

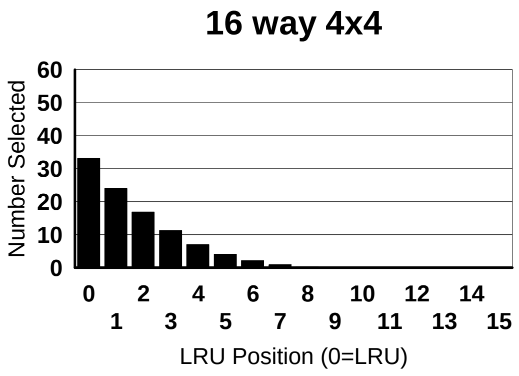

Figure 3.3c

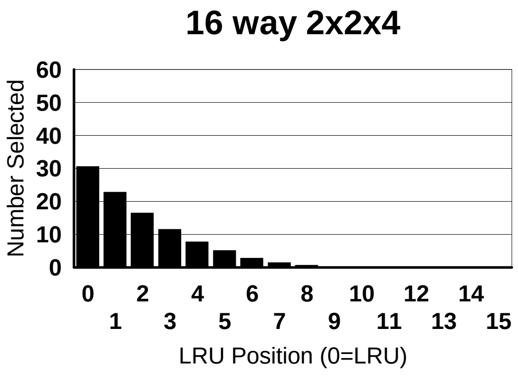

Figure 3.3d

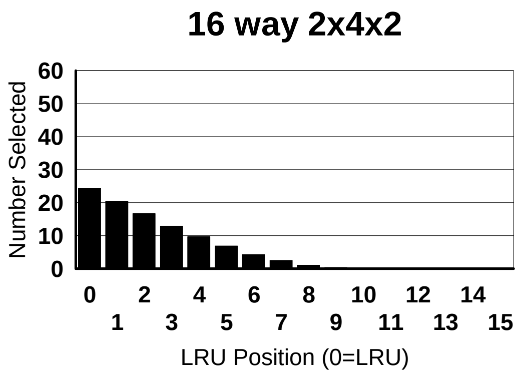

Figure 3.3e

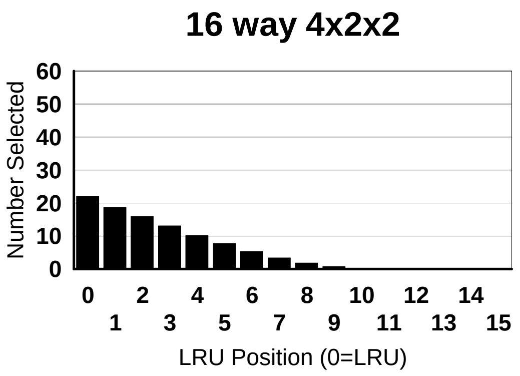

Figure 3.3f

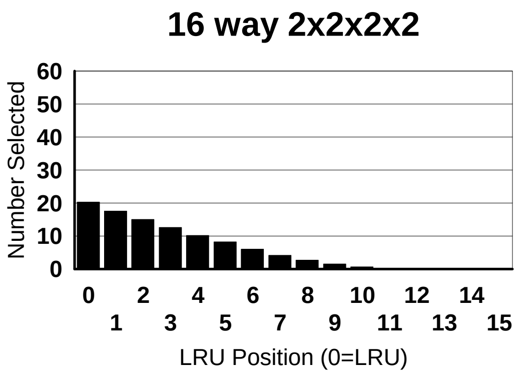

Figure 3.3g

4. Conclusion

A generalization of tree-LRU has been described, in which the branching factor at each level of the logical tree is not limited to two. Results have been shown illustrating to what degree these generalized tree-LRU methods approximate LRU replacement, assuming all permutations of access orderings of the lines in a set are equally likely, for all logical trees for 8, 12, and 16 way set-associative caches. These results are preliminary since in practice all permutations are not equally likely; in particular, the probability of re-reference of a line in a set decreases sharply with increasing age. Nevertheless the results shown give a good indication of the degree to which each logical tree approximates LRU replacement. In practice these methods would have to be evaluated using the usual approaches (detailed studies of design tradeoffs, with projected miss ratios found using traces, execution driven simulation, and so on).

These generalized tree-LRU methods have been known by practitioners for some time [1]. Nevertheless, in the past, tree-LRU has usually been chosen as the standard method to approximate LRU replacement (of course alternative design approaches have been used as well that do not approximate LRU replacement, for example "random" replacement and other approaches). However there currently seems to be a trend towards increasing degrees of set-associativity in cache design. If conflict misses contribute significantly to the overall miss ratio, even with a relatively high degree of set associativity, then closer approximations to LRU replacement could prove useful in increasing cache performance.

References

[1] William Starke, personal communication, July 10, 2004.