A Systolic Array for Implementing LRU Replacement

J.P. Grossman

Abstract

Increasing the associativity of a cache reduces both the miss rate and the power consumption. It also makes LRU replacement more difficult to implement. We present a simple systolic array that can be used to implement LRU replacement in arbitrarily associative caches.

1 Introduction

One of the important design parameters of a hardware cache is its degree of associativity. Increasing a cache's associativity improves performance by reducing the miss rate [Hennessy96] and leads to a lower power implementation [Zhang00]. However, it also becomes more difficult to implement a least recently used (LRU) replacement policy. As a result, hardware designers opt for simpler replacement strategies such as round-robin [Clark01], even though the LRU policy is known to provide better performance [Smith82].

In this paper we present a simple systolic array that can keep track of LRU information for a set of cache lines. Since the length of the critical path is constant, the approach can be used for *N*-way associative caches with *N* arbitrarily large. We begin by constructing a systolic array that can handle one cache access on every other cycle. In section 3 we modify the design to allow a cache access on every cycle. Finally in section 4 we show how to accommodate multiple accesses per cycle.

2 Implementation

The central idea is to maintain a list of cache line indices sorted from LRU to MRU (most recently used). When a cache line is accessed its line index L is presented to the list, and that index is rotated to the MRU position at the end (Figure 1a). We can implement this list as a systolic array by advancing L one node per clock cycle, along with a single-bit "matched" signal M, indicating whether or not the index has found a match within the array. Until a match is found, L is advanced without any changes being made. Once a match is found, nodes begin copying values from their neighbours to the right. Finally, L is deposited in the last node. This is illustrated in Figure 1b. We can use the same design for all nodes by wiring together the last node's inputs, as shown in Figure 1b. This ensures that L will be deposited because by the end of the array we are guaranteed that M=1, so the last node will attempt to copy a value from the right, and with the inputs wired together this value is L. Note that we can only present indices to the array on every other cycle. For example, if in Figure 1b '2' were presented on the cycle immediately following '1', then the value '1' would erroneously be copied into the first node instead of the correct '3'.

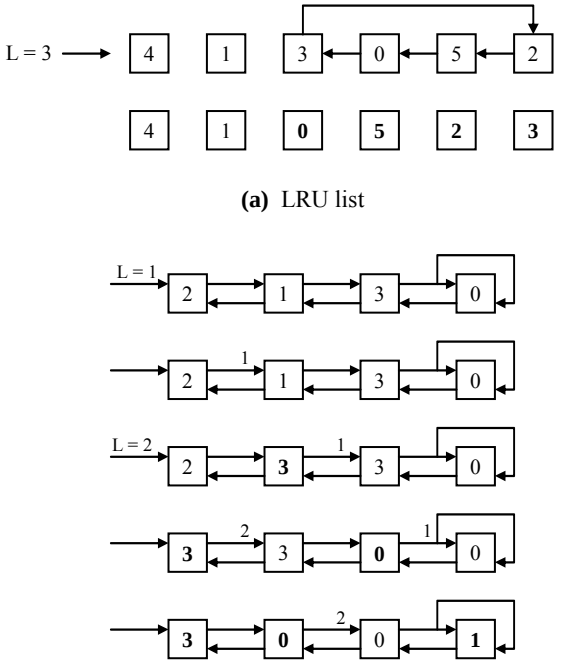

Figure 1: Keeping track of LRU information using **(a)** an atomically updated list **(b)** a systolic array.

Figure 2 shows a hardware implementation of the systolic array node. The forward signals are the line index L and the match bit M; the backward signal is the current index which is used to shift values when M=1. The node contains two log*N* bit registers, one single-bit register, a log*N* bit multiplexer, a log*N* bit comparator, and an OR gate. No extra hardware is required to set up the array as it can be initialized simply by setting M=1 and presenting all *N* line indices in *N* consecutive cycles

Project Aries Technical Memo ARIES-TM-18
Artificial Intelligence Laboratory
Department of Electrical Engineering and Computer Science
Massachusetts Institute of Technology
Cambridge, MA, USA

Sponsored by DARPA/AFOSR Contract Number F306029810172

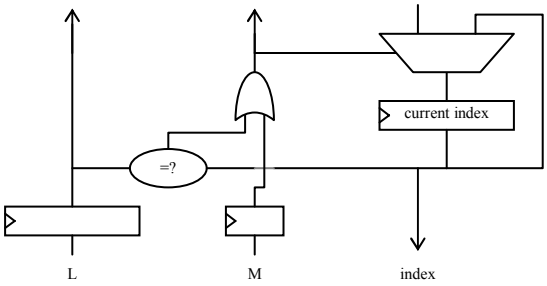

Figure 2: Systolic array node

followed by N copies of the last index ($N - 1$) in the next N consecutive cycles.

In normal operation the input M to the first node is always 0. On a cache hit, the line index L is presented to the array. On a cache miss, the output of the first node gives the LRU line index; this line is replaced and the index is fed back into the array. On a cycle with no cache activity, the index of the most recently accessed line is presented, which does not change the state of the array (this technique avoids the need for a separate "valid" bit).

3 One Access per Cycle

To accommodate one cache line access per cycle, the systolic array must be modified to behave as though it were being clocked twice as fast. We can accomplish this by simply removing every other set of forward registers and modifiying the backward 'index' signal slightly to obtain the new node implementation shown in Figure 3. The index signal is taken from the input rather than the output of the bottom register to ensure that when the previous node attempts to copy the index value, it obtains the value that would be stored in this register *after* the node has finished processing its current inputs. This new systolic array, which contains $N/2$ nodes, can be initialized by presenting all N line indices in N consecutive cycles with M=1.

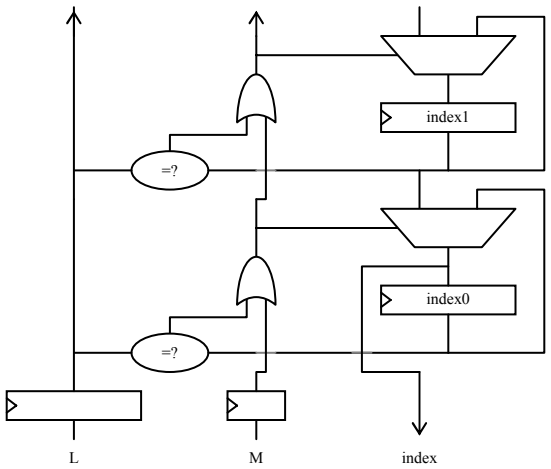

Figure 3: Modified systolic array node

4 Multiple Accesses per Cycle

Some modern processors allow up to four cache accesses on a single cycle [Bradley02]. A systolic array capable of maintaining LRU information with $k > 1$ accesses per cycle is naturally more complicated, but can be implemented as follows: Each node holds $2k$ indicies. There are k forward line index signals $L_1, \ldots, L_k$, and we replace the match bit M with a $\lceil \log(k+1) \rceil$ bit match *counter* indicating how many of the index signals have found their match. The comparator and OR gate used to update M are replaced by k comparators, a k-input OR gate, and a $\lceil \log(k+1) \rceil$ bit incrementer. Each index register is fed by a $k+1$ input multiplexer taking its inputs from that register as well as the k above it; the multiplexer is controlled by M. Finally, a given multiplexer input connects to an index register's *output* if the register is in the same node, otherwise it connects to the index register's *input*. This generalizes the node design presented in Section 3; Figure 4 shows the resulting design for $k = 2$.

References

[Bradley02] David Bradley, Patrick Mahoney, Blaine Stackhouse, "The 16kB Single-Cycle Read Access Cache on a Next-Generation 64b Itanium Microprocessor", Proc. ISSCC 2002, pp. 110-111.

[Clark01] Lawrence T. Clark, Eric J. Hoffman, Jay Miller, Manish Biyani, Yuyun Liao, Stephen Strazdus, Michael Morrow, Kimberley E. Velarde, Mark A. Yarc, "An Embedded 32b Microprocessor Core for Low-Power and

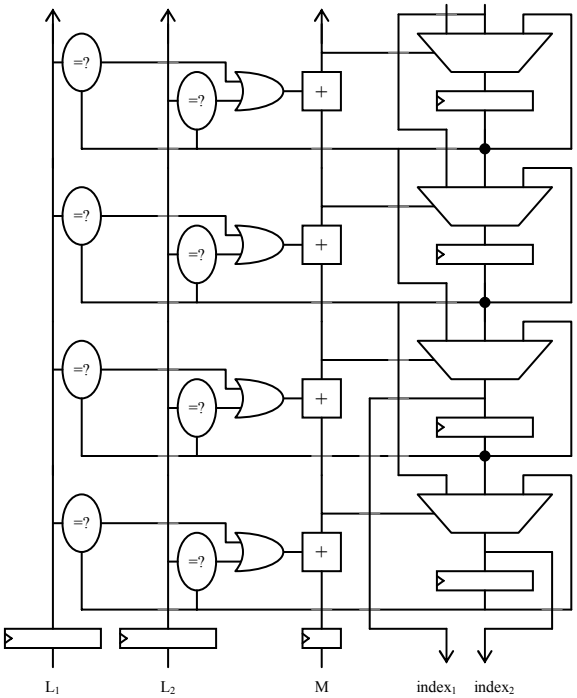

Figure 4: Systolic array node for 2 accesses per cycle

High-Performance Applications", IEEE Journal of Solid-State Circuits, Vol. 36, No. 11, November 2001, pp. 1599-1608.

[Hennessy96] J. Hennessy, D. Patterson, Computer Architecture: A Quantitative Approach, 2nd ed., Morgan Kaufman, San Mateo, CA, 1996.

[Smith82] A. J. Smith, "Cache Memories", ACM Computing Surveys, Vol. 14, No. 3, September 1982, pp. 473-530.

[Zhang00] Michael Zhang, Krste Asanović, "Highly-Associative Caches for Low-Power Processors", Proc. Kool Chips Workshop, 33rd International Symposium on Microarchitecture, Monterey, CA, Dec. 2000.