MRAC: A Memristor-based Reconfigurable Framework for Adaptive Cache Replacement

Ping Zhou†, Bo Zhao†, Youtao Zhang‡, Jun Yang†, Yiran Chen†
†Electrical and Computer Engineering Department, ‡Department of Computer Science
University of Pittsburgh, Pittsburgh, PA 15261
†{piz7, boz6, juy9, yic52}@pitt.edu, ‡zhangyt@cs.pitt.edu

***Abstract*—Memristor, a long postulated yet missing circuit element, has recently emerged as a promising device in non-volatile memory technologies. However, beyond its use as memory cell, it is challenging to integrate memristor in modern architectures for general purpose computation.**

In this paper we propose a non-conventional use of memristor and demonstrate its applicability to enhancing cache replacement policy. We design a memristor-based saturation counter which can track cache access history at low cost. Based on our counter design, we develop a cache replacement framework that is both reconfigurable and adaptive (MRAC). Our evaluation demonstrates MRAC's reconfigurability and adaptivity, which result in better performance (up to 57.9% more cache miss reduction) and more robust performance improvement.

I. Introduction

Memristor, a long postulated yet missing circuit element, has recently emerged as a promising device in non-volatile memory technologies. [8]. Memristor has unique accumulative characteristic that allows it to memorize access history. However, beyond its use as memory cell, it is challenging to integrate memristor in modern architectures for general purpose computation.

In this paper, we propose a novel, non-conventional use of memristor and demonstrate its applicability to enhancing cache replacement policy. By leveraging memristor's unique ability of memorizing history, our design is the first work that integrates memristor in modern architectures for general purpose computation. We design a memristor-based saturation counter, which can be used to track cache access history at low cost. Instead of simply using memristor to store multi-bit data, we use calibrated pulses to alter memristor's state directly, and use matrix scanner to find largest or smallest counter within a set. We then construct a reconfigurable and adaptive cache replacement framework (MRAC) based on our counter design. Our evaluation on various workloads demonstrates MRAC's reconfigurability and adaptivity, which result in not only better performance but also more robust performance improvement.

II. Background of Memristor

Memristor was theoretically predicted by L. Chua in 1971 [1] as the fourth circuit element in addition to resistor, capacitor and inductor. Memristor shares many advantages with other new memory technologies such as non-volatility, high density and zero leakage [8]. What makes memristor unique is its ability to memorize history of inputs. The memristor's state (i.e., resistance) depends on the amount of flux flowing through the device, i.e., the integral of voltage over time. Therefore, by controlling voltage pulses, one can control a memristor's resistance continuously between the smallest R_{on} and the largest R_{off} values. In addition, changes caused by multiple voltage pulses on a memristor cell are accumulated, meaning that access history is memorized as the accumulated change in resistance.

III. Memristor-based Saturation Counter

Memristor's large off-to-on resistance ratio (10,000) [6] allows one cell to store multi-bit data. Since a memristor can accumulate the effect of multiple voltage pulses, and its resistance saturates at two ends, it is natural to treat a memristor as a saturation counter.

In this paper we use memristors as saturation counter to track cache access history. A simple use of memristors as *conventional* multi-bit counter would always perform analog-digital conversion between a memristor's resistance and its represented value before any operation. This would require sophisticated sensing circuit and precise control of voltage pulse. For example, finding the largest value from 32 counters requires a multi-stage comparator that tends to be slow and costly.

In our design, instead of translating a memristor's resistance to digital value back and forth, we apply calibrated pulses to alter the cell's state directly. For comparison operations, we design a semi-analog comparator based on matrix scanner (Fig. 1 through Fig. 3) to support sorting out the largest (or smallest) one among a set of counters. The inaccuracy of our semi-analog implementation is tolerable as cache access counter does not have to be very accurate (e.g. picking a counter that is close to the largest one does not usually make much difference on miss rate).

IV. MRAC Framework

Using our memristor-based saturation counter, we construct a cache replacement framework (MRAC) that supports both *reconfigurability* and *adaptivity*. Fig. 4 illustrates an overview of MRAC.

MRAC has a configurable "policy pool", in which each policy candidate is described by a parameter vector `[InitPos, HitPromo, Aging, AFlag, BIP]`. Each field of the parameter vector describes one aspect of replacement policy: `InitPos` indicates the initial location (counter value) when a new cache line is inserted,

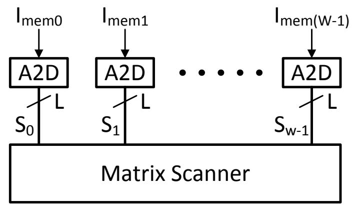

Figure 1. Block diagram of semi-analog comparator.

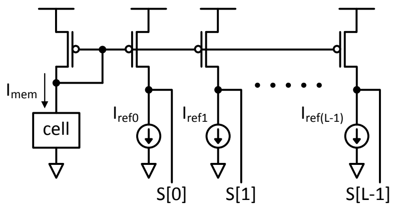

Figure 2. Analog to digital translation.

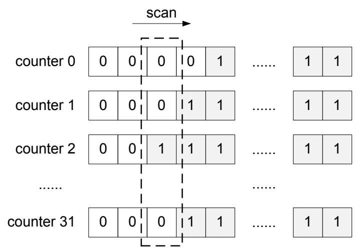

Figure 3. Matrix scanner.

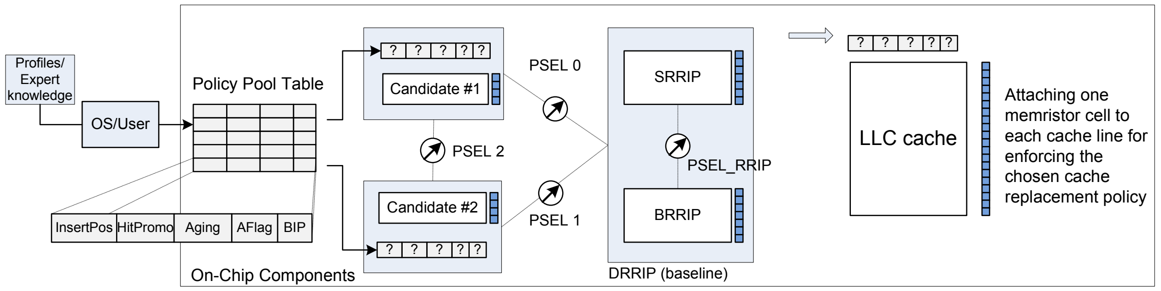

Figure 4. MRAC overview.

HitPromo indicates how to promote a cache line when it gets a hit, AFlag and Aging describe how counters age (LRU-style or RRIP-style), and BIP indicates whether to enable Bimodal Insertion [5] to defeat thrashing.

MRAC leverages the idea of set dueling [5] and constructs a three-entity tournament to choose favorable policy at run-time. The tournament includes two dynamic candidates which are picked from the policy pool, and a baseline policy (RRIP) to ensure bottom line performance. The two dynamic policies are changed over time based on the results of tournament, and the final winner of the tournament is selected and applied to the follower sets of the cache. MRAC can also be extended to support share cache on multi-core systems (TA-MRAC) in a similar way as in TA-RRIP [3].

V. Evaluation

We evaluated MRAC using a PIN-based [4] trace-driven simulator. L1 and L2 caches are 32KB and 256KB respectively, and MRAC is implemented in L3 cache (4MB). We experimented with various workloads from SPEC CPU2006 [2]. The policy pool is configured based on profiling information. For shared cache studies, we modeled a 4-core system with private L1/L2 caches and shared L3 cache, and generated 500 mixes of different programs to form multiprogrammed workloads.

Fig. 5 and Fig. 6 show MRAC performance for single- and multi-core cases respectively. As we can see MRAC not only achieves better performance than baseline (up to 57.9% more cache miss reduction), but also delivers more robust performance improvement.

VI. Conclusions

This paper presents a non-conventional use of memristor. We design a memristor-based saturation counter to track cache access history, and used the counter design to construct MRAC, a low-cost *reconfigurable* and *adaptive* cache replacement framework. Evaluation of MRAC demonstrated its benefits in accommodating different workloads and delivering more stable performance improvement.

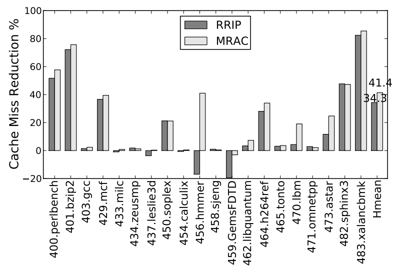

Figure 5. Cache miss reduction % comparing to LRU.

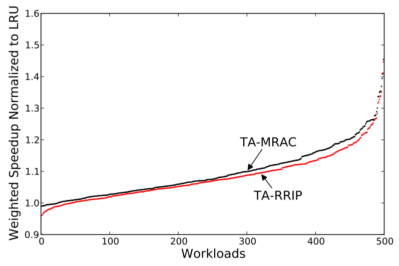

Figure 6. S-curve of Weighted Speedup [7] on shared cache.

References

[1] L. O. Chua. Memristor - The Missing Circuit Element. *IEEE Transactions on Circuit Theory*, 18(5):507–519, 1971.
[2] T. S. P. E. Corporation. SPEC CPU2006. http://www.spec.org/cpu2006/.
[3] A. Jaleel and et al. High Performance Cache Replacement Using Re-Reference Interval Prediction (RRIP). In ISCA, 2010.
[4] C. Luk and et al. Pin: building customized program analysis tools with dynamic instrumentation. In PLDI, pages 190–200, 2005.
[5] M. K. Qureshi and et al. Adaptive Insertion Policies for High Performance Caching. In ISCA, 2007.
[6] T. Raja and et al. Digital Logic Implementation in Memristor-based Crossbars A Tutorial. In DELTA, pages 303–309, 2010.
[7] A. Snavely and D. Tullsen. Symbiotic Jobscheduling for a Simultaneous Multithreading Processor. In ASPLOS, 2000.
[8] R. S. Williams. How We Found the Missing Memristor. *IEEE Spectrum*, 2008.