

Improved Energy Efficiency with Runtime Reconfigurable Caches on FPGAs

Tianqi Wang*, Ang Li[†], Tong Geng*, Martin C. Herbordt*

* Boston University [†]Pacific Northwest National Laboratory

I. INTRODUCTION

The most basic way to improve performance and energy efficiency of high-performance computing (HPC) clusters is to improve the same in the underlying processors. In modern processors, the cache subsystem occupies more than half of the die area and power budget. Improving the cache subsystem's energy efficiency is therefore an inviting research target.

The performance of a cache architecture is often application dependent. However, in fixed logic processors, such as CPUs and GPUs, a single compromise design must be selected, even though this design will be sub-optimal for most applications. Reconfigurable caches can adjust the cache configuration dynamically to the get the best-suited cache parameters at runtime and so significantly reduce energy consumption [1]. For example, sometimes it is possible to deploy a low capacity, low associativity, design without increasing the miss rate substantially. A complication is that for modern multi-core processors, each core's memory access behavior can be influenced by other cores.

In this extended abstract we present a design for reconfigurable caches on FPGAs that can run in modes with different capacities and associativities. We demonstrate that improved performance and energy efficiency can be achieved by tuning these cache parameters at runtime.

II. RELATED WORK

Much previous work has been devoted to configurable caches. Albonesi [2] disabled portions of a set associative cache and proposed a threshold to control the tolerated performance degradation brought by this way-concatenation. Zhang, et al. [3], [4] designed a simple configuration circuit to manage the availability of the address string (tag and set index) to control the associativity. Way concatenation has also been deployed for capacity change. A heuristic is provided to search for the most power efficient configuration for a particular application. The heuristic is based on a cache power equation based on observations on the relative impact on energy consumption and performance of the primary cache parameters: capacity, block size, and associativity [5].

Most of these studies have been theoretical analyses verified on software simulators such as SimpleScalar. Few have included tests on real hardware. Some have proposed that the configuration circuits would effect only minor performance degradation because only a few gates are added to the critical data path. While this is true, it does not account for a number of other factors. These include the added complexity of the cache controller due to configurability and how configurability interacts with the victim selection when no free slot is available. These shortcomings provide motivation to develop a more realistic and reliable implementation.

III. SYSTEM DESIGN

We use the MicroBlaze embedded soft processor IP to build the system. As shown in Fig. 1(A), there are four MicroBlaze cores (MB core 1 - 4) in the system. Each MicroBlaze core has private L1 instruction (I-Cache) and data (D-Cache) caches. All four cores share the L2 cache through a switch.

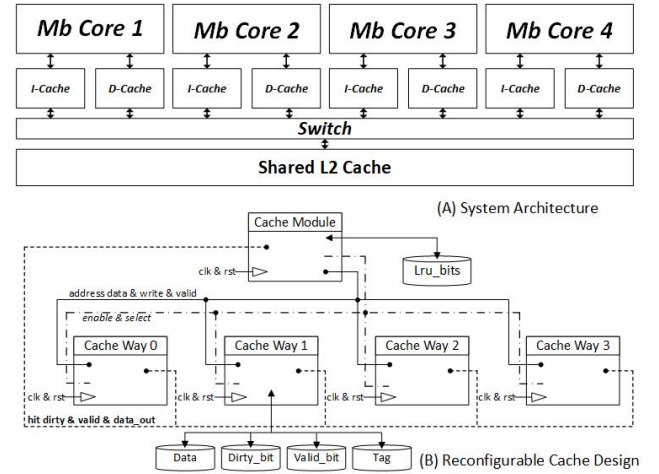


Fig. 1. (A) System architecture and (B) Cache design

A. Cache Design

Figure 1 (B) shows the design of the reconfigurable cache module. It deploys the way-concatenation method of sub-array partitioning to change the associativity and capacity. This implementation, however, has much more complex logic. Three types of signals appear in this figure: address, data, and control. The control signal has enable and select. A particular way (SRAM storage) is pre-charged only when the enable signal is set. The select signal picks the specific way for store under the set associative option.

There are two accessing modes for this module: try access and compulsory replacement.

Try access only happens during the first cycle after a fresh request arrives. The cache module passes the entry address to all the enabled way components. Each way reads from or writes to the target entry if the tag extracted from the sending address coincides with the locally saved tag (a hit). It returns data from that entry on reads and 0 on writes. Other related information, dirty_bit, valid_bit, and hit/miss, are also returned. This information is then transmitted to the higher memory system module within the current cycle. One thing to notice is that this access type only modifies the data if applicable. Other storage (tag, dirty_bit, valid_bit) is read-only.

Compulsory replacement is performed while processing misses. When the write signal is set, the way chosen by the select signal will replace the data inside the target entry with

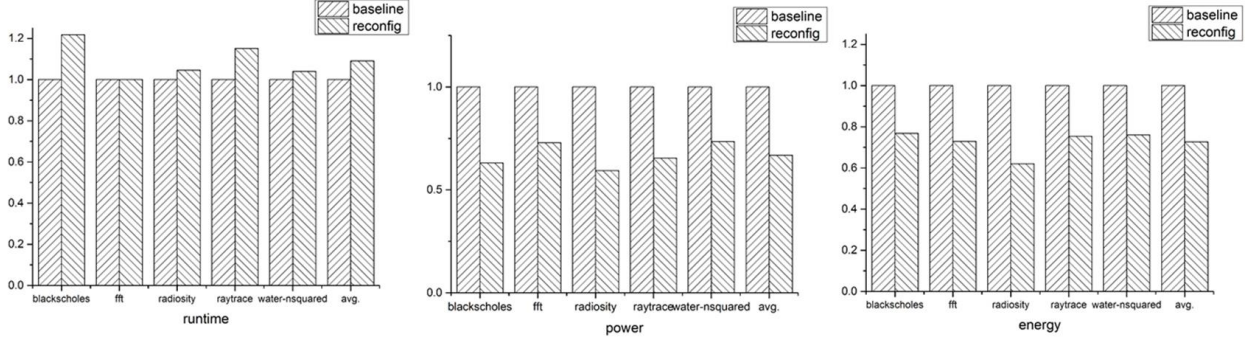


Fig. 2. Experimental results for performance, power, and energy

the one from the data bus. It also modifies the tag storage as well as the dirty_bit and valid_bit correlated with that entry. The processing time depends on DRAM read latency (if data is from there). The input configuration signals are assoc and wayshut. Both are 2-bit signals.

B. Reconfiguration policy

We use performance counters to profile the cache subsystem for T_{sample} , which is the period of the sample time. Cache parameters are adjusted at runtime. Fig. 3 shows the reconfiguration policy of the cache subsystem.

For Level 1 cache, we change the set associativity dynamically. A decision tree is used to generate a suitable configuration. We choose Core-Critical and Hit Ratio to measure the L1 cache behavior. The definition of Core-Critical is shown in Eq. 2. As shown in Fig. 3A, Config 1 is 8KB 2-way associative and Config 2 is 8KB 4-way associative.

For Level 2 cache, we change the capacity of the shared cache dynamically. In order to get the adjustment polarity, we record the cache misses of Level 2 cache and compare this period's result M_t with last three periods' average $(M_{t-1} + M_{t-2} + M_{t-3}) / 3$. As Fig. 3B shows, a saturation counter is used to avoid executing too many reconfigurations. The size of shared L2 cache can be 32KB, 64KB, or 128KB.

$$StallTime_i = L1_{miss} * L1_{penalty} + L2_{miss} * L2_{penalty} \quad (1)$$

$$Core - Critical_i = \frac{StallTime_i}{StallTime_{max}} \quad (2)$$

IV. EXPERIMENT RESULTS

We implement our design on a Xilinx VC707 development kit which hosts a Xilinx Virtex-7 XC7VX485T-2FFG1761C FPGA and 1GB DDR3 memory. The whole FPGA-based design has an operating frequency of 100MHz. We choose several benchmarks (blackscholes, fft, radiosity, raytrace and water-nsquared) from PARSEC-2.0 and SPLASH-2. The toolchain is based on Xilinx Vivado 2016.4.

The baseline result was collected from the four-core system with fixed private 8KB 4-way associative L1 cache and shared 128KB L2 cache. In Fig. 2, we compare the reconfigurable result with the baseline. All the results have been normalized.

As Fig. 2A shows, because the reconfigurable design often uses a smaller cache, there is an 8.3% performance reduction in average. However, Fig. 2B shows the power of reconfigurable

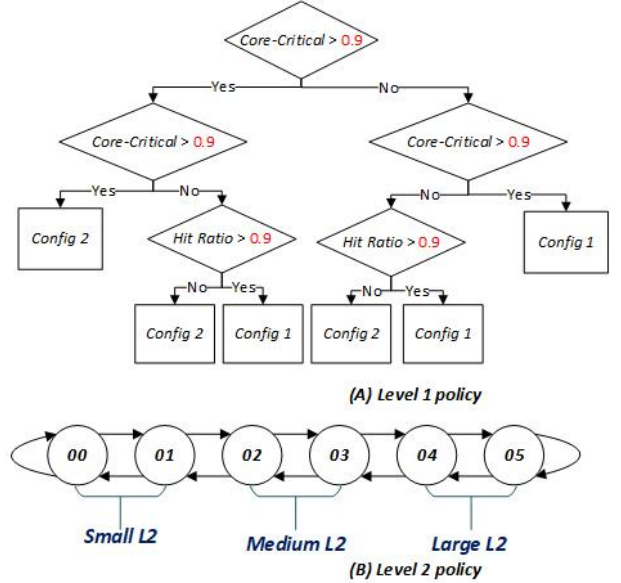


Fig. 3. Cache system reconfigurable policy

system is 27.4% lower than baseline. Fig. 2C shows that the reconfigurable system has 37.1% better energy efficiency than the baseline.

V. ACKNOWLEDGMENTS

This research is partially supported by U.S. DOE Office of Science, Office of Advanced Scientific Computing Research, under the CENATE project (award No. 66150). The Pacific Northwest National Laboratory is operated by Battelle for the U.S. Department of Energy under contract DE-AC05-76RL01830.

REFERENCES

- [1] T. VanCourt and M. Herbordt, "Application-dependent memory interleaving enables high performance in FPGA-based grid computations," in *Proc. Field Programmable Logic and Applications*, 2006, pp. 395–401.
- [2] D. Albonesi, "Selective cache ways: On-demand cache resource allocation," in *Proc Int Symp Microarchitecture*, pp. 248–259.
- [3] C. Zhang, F. Vahid, and W. Najjar, "A highly configurable cache architecture for embedded systems," in *Proc Int Symp Computer Architecture*, 2003, pp. 136–146.
- [4] C. Zhang, F. Vahid, and R. Lysecky, "A self-tuning cache architecture for embedded systems," *ACM Trans Embedded Computing Systems*, vol. 3, no. 2, pp. 407–425, 2004.
- [5] L. Chen, X. Zou, J. Lei, and Z. Liu, "Dynamically reconfigurable cache for low-power embedded system," in *Int Conf Natural Computation*, vol. 5, 2007, pp. 180–184.