Physical implementation of a reconfigurable cache. Optimization possibilities.

Santana Gil, A. D.
General Physics Dept.
Faculty of Physics
University of Havana
Havana, Cuba
adavid@fisica.uh.cu

Benavides Benitez, J.I.
Computer Architecture Dept.
Polytechnic School,
University of Cordoba
Cordoba, Spain
el1bebej@uco.es

Hernandez Calviño, M.
General Physics Dept.
Faculty of Physics
University of Havana
Havana, Cuba
mhernan@fisica.uh.cu

Herruzo Gómez, E
Computer Architecture Dept.
Polytechnic School,
University of Cordoba
Cordoba, Spain
el1hegoe@uco.es

***Abstract*—The microprocessor performance is highly dependent on cache size and structure. This work present two alternatives of a reconfigurable cache implemented on FPGA, with twelve cache modes and a test platform based on the MicroBlaze soft processor. Some experimental results are also presented. In addition, two methods for dynamic reconfiguration of the implemented cache are introduced. Finally, interesting conclusions about the work carried out are presented.**

Keywords-component; Reconfiguration, Cache Memory, FPGA, MicroBlaze;

I. INTRODUCTION

The cache memory has an important role in computer system performance. The design of a cache is an optimization problem that relates with the maximization of the hit ratio and the minimization of the access time. The cache performance has an important dependency on cache size, associativity level, number of words per block and cache latency.

The best cache configuration depends on the application characteristics and design constraints, like performance, power consumption or area, which has promoted the diversity of cache architectures found in different processors. Since no cache organization satisfies the requirements of all applications, a promising approach to this problem is to add reconfiguration capabilities to the cache memory.

Taking in account the constraints involved in the problem and the desired features, cache designers proposed three well-known cache organizations: direct mapped cache, fully associative cache and set associative cache. Each organization can be better for a specific workload.

The main research lines, on cache memories, mainly focused on cache memories architecture simulation for performance analysis, low power cache systems, implementation on FPGAs of fixed and reconfigurable cache system for testing theoretical designs purpose and analysis of cache implemented for high end or embedded processors.

In this paper, we propose two methods for dynamic reconfiguration of a properly designed cache memory. The reconfigurable cache memory, implemented on FPGA, bases its functionality on our previous paper [1] but the new design has 12 different cache modes (4 additional modes than previous implementation) and 6 auxiliary modes for testing and basic performance analysis and enhances the previous one for obtaining resource usage and speed improvement.

The paper is organized as follows: In Section II we review previous related work in reconfigurable cache. Section III describes the proposed design, implementation, and testing platform. In Section IV we present tests and results. In Section V we propose two methods for dynamic reconfiguration. Finally, Section VI presents the conclusions and future work.

II. PREVIOUS WORK

Ranganathan et al. [2] propose a cache memory capable of, dynamically, divide the cache SRAM arrays into multiple partitions used for different processor activities. These activities can benefit applications that otherwise would not use the storage allocated to large conventional caches.

They used a modification of the CACTI [3] analytical model to analyze the design. The system shows a small impact on cache access time. They also evaluate one representative use of reconfigurable caches instruction, reuse for media processing and obtain IPC improvements ranging from 1.04X to 1.20X in simulation across eight media processing benchmarks.

Cheng L. et al.[4] propose a reconfigurable cache that improve not only the overall performance but also the energy consumption for embedded system. They introduce a novel reconfigurable algorithm that dynamically detects the phase changes and automatically select the optimal cache configuration. It comes as no surprise that such a mechanism exhibits some form of performance loss, but the expectation is that the loss is small, compared to the energy saving improvement. They use as starting point for they work the proposition of Albanesi et al [5].

Only a small percent of paper published about cache memory attempts a physical implementation. Most of them are only simulated theoretical models.

Based on the above referenced papers, we presented in a previous work [1] an implementation of a reconfigurable cache on FPGA with eight cache modes including the analysis of basic test results. Now, in this paper, we present a

cache memory/controller with four additional cache modes compare with the previous work [1]. The new added modes correspond to 4-way set associative cache organization with 1, 2, 4 or 8 words per block. This means that new design can work as Direct Mapped, 2-Way Set Associative or 4-Way Set Associative cache with 1, 2, 4 or 8 word per block in each case.

III. RECONFIGURABLE CACHE

We implemented the new design using a Spartan3E-MicroBlaze Development Board. A general structure of the test platform is shown in Fig. 1.

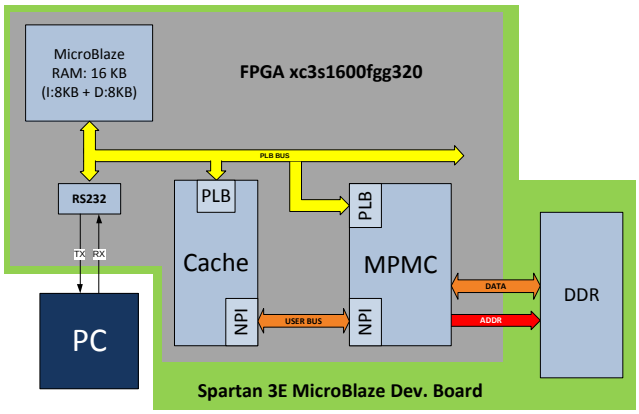

Figure 1. New test platform.

We use the capabilities of the Multi Port Memory Controller (MPMC) from Xilinx, to optimize access to the external RAM (DDR on board) depending on the number of words (1, 2, 4 or 8) per block in cache line. The cache memory/controller connects directly to the Processor Local Bus (PLB) of the MicroBlaze instead of using parallel ports like in the previous paper [1].

It is well known that the MicroBlaze processor has its own cache memory. This cache is a fixed direct mapped type with one word per block and can be enabled at design time if desired. In our design, the MicroBlaze cache remains disabled.

The main part of the implemented reconfigurable cache memory/controller is a group of BRAMs properly connected to allow different logical arrangements. The minimum block size is a 1K. The BRAMs width for data, tag, valid bit and LRU bit is 32, 14, 1 and 1 in turn.

If we use 2 address bits to select appropriately four1K BRAM blocks, combined with its 10-bit address, we can build 4 K module suitable for a direct mapped cache. However, selecting the same four blocks in a parallel fashion, we can build a 4-way associative cache of 1 K size. An intermediate configuration, consistent with a 2-way 2 K size associative cache is also possible, using1address bit to select one of two modules of 2 K size each. Four modes; 1, 2, 4, and 8 words per cache line are also available for each associativity level.

The desired cache mode value is stored on the cache mode register (CMR). Based on CMR value, the control block (CB) arranges properly the BRAMs group mentioned above. At each memory access, the CB triggers the corresponding state machine located on the execution block (EB) to start performing the selected cache mode. The resulting hit or miss event increments the corresponding 16-bit wide counter. A simplified block diagram of the cache memory/controller is presented in Fig. 2.

The implemented cache has data/instruction capability. In this paper, all tests reported are for the cache connected to function as a data cache as observed in Fig. 1.

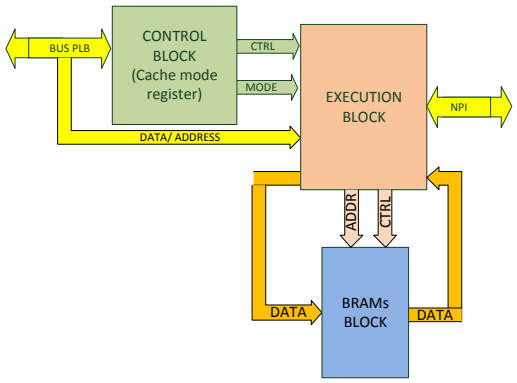

Figure 2. Simplified block diagram of the cache memory/controller.

We implemented two alternatives of the cache memory/controller. The first, option (A), has a maximum size of 4K, achieved for modes with direct mapped structure. Due to the limited amount of BRAMs on the FPGA, it was not possible to maintain the same memory size for modes with larger associativity. In this variant, an increase of the associativity level reduces the cache size when we move to 2-way (2K size) and 4-way (1K size) set associative modes.

For a particular test analysis and a better comparison with cache simulator results [6] we implemented a second (B) variant. This is a1K size limited variant, to make the size constant with the increase of associativity.

We also developed an embedded application for the MicroBlaze processor to allow the control of the test platform from a PC and facilitate the debugging process. Both cache variants were exhaustive tested to detect and eliminate bugs.

For more elaborated tests, we use the memory traces accompanying the SMPCache [6] simulator. We developed an application on LabView to control the test platform, run selected traces, and process statistical data collected from the reconfigurable cache.

IV. TEST AND RESULTS

We ran the memory traces for EAR, COMP, HYDRO and CEXP algorithms, from the SMPCache [6] package, on the reconfigurable cache for the 12 modes in both variants.

In Fig. 3 we present the results of the tests for the EAR algorithm for all modes of both variants. For A variant, (Fig. 3a), we observe that a reduction in the cache size increase the miss rate, as expected. However, the increase of the associativity has no noticeable effect on the miss rate. This

suggests that, as far as concerning the miss rate, the cache size is more important than the associativity.

For B variant, (Fig. 3b), the direct mapped and 4-way associative modes have the same performance. In 2-way associative modes, the miss rate is higher than similar modes on variant A. In this case, increase associativity is not helpful to compensate the low cache size. A similar trend is observed for the algorithms CEXP and HYDRO.

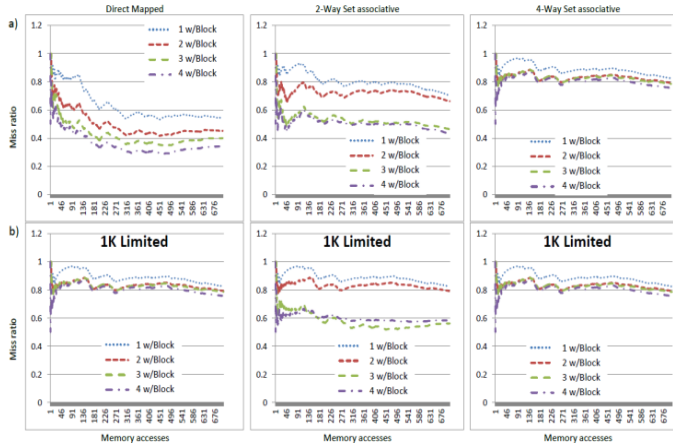

Figure 3. Tests result for EAR algorithm for the two variants of cache memory/controller: a) Size decreases with associativity variant, b) 1K fixed size variant.

Nevertheless, a different behavior is observed for COMP algorithm tested on variant B cache, (Fig. 4b). In this case, the low cache size does not affect significantly the direct mapped modes. For option (A) cache, the behavior is similar to that observed when running the above-cited algorithms.

We configured the SMPCache [6] to replicate each mode of the reconfigurable cache and ran the memory traces of the same algorithms. We confirmed the behavior observed in each test and validated once again our design.

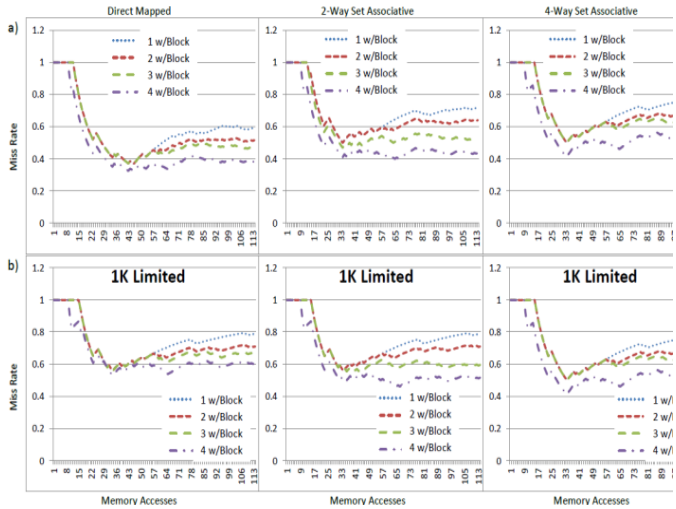

Figure 4. Tests result for COMP algorithm for the two variants of cache memory/controller: a) Size decreases with associativity variant, b) 1K fixed size variant.

V. Methods for Dynamic Reconfiguration

Studies have demonstrated that the optimization of the cache performance is a difficult task. Numerous papers [7,8,9,10,11,12,13] propose methods for a dynamic adjustment of selected parameters to achieve low miss rate values. Motivated by some of these ideas and the fact that we implemented and successfully tested a cache with reconfigurable capability, we propose two methods for cache optimization. These methods should be as simple as possible to allow them be implemented on hardware without excessive FPGA resource usage. Then, the module can be attached to the control port of the cache, allowing automatic reconfiguration.

The first method bases on the detection of program phases as described in [14] and the second one on the dynamic adjustment of the cacheable address boundaries of the external memory.

The two methods have a common initial feature. During execution, we start defining a time window of width, N_{ACC}, characterized by a predefined number of memory accesses, e.g. 500. Beyond this point, both methods differ as follows:

A. *Method I: Detection of program phases.*

We split the external RAM in a number of address segments, N_{SEC}, e.g. 256, and define a bit vector of this length. Every bit in the vector represents a segment of RAM.

When an access takes place, we calculate the segment accessed and set to one the corresponding bit vector. Once the number of accesses, N_{ACC}, is completed, we compute the sum, B, of all the bits set to one in the past time window. B roughly indicates how localized or spread the accesses to RAM have been. The vector is cleared after each time window. In parallel, we also compute the coefficient:

$$D = |B_n - B_{n-1}| \qquad (1)$$

where B_n and B_{n-1} are actual and preceding values of B.

If D is greater than a pre-defined threshold value, we assume that the algorithm has entered in a new different phase and the change must be analyzed in order to decide if reconfiguration is needed.

We base the next analysis step on the miss rate. If the miss rate between two consecutive windows is over a desired value then the reconfiguration process is applied, selecting configurations with higher associativity. If the miss rate maintains below the established value for a given number of consecutive time windows, the system moves to 1 way lower associativity mode. We developed an application in LabView to simulate the algorithm, but the method is still under evaluation to analyze the real impact on system performance.

Method II: Automatic selection of memory range to cache.

The cache memory of the MicroBlaze processor is configured at design time. The user can set the address range of the cached external memory. Therefore, it is possible to

optimize the system performance if the range is accurately selected but it is difficult to have a previous knowledge of the algorithm behavior. This method is an effective solution for small cache systems because the effect of restricting the range to be cache is equivalent to increase the cache size.

Although we do not use the MicroBlaze own cache system [15], this idea motivates us to develop the second method we are proposing.

For this method, we split the external RAM in a number of address segments, N_{SEC}, e.g. 256 and define a vector that stores the total number of accesses for each segment in a certain time windows.

Next, we compute the position of the "center of mass", MC, of the resulting histogram and locate the center of the cache window at that point.

Now, moving up and down from the MC position and following an iterative process, we find a region with symmetrical boundaries from the MC position, that contains a predefined percent (e.g. 90%) of total memory accesses occurred in the past time window. These boundaries define the new cache region that will be used in the next time window. The process repeats itself during all program execution and the result is a cache window automatically adjusted to cache the memory region with majority accesses in each time interval. The method can be implemented for both a reconfigurable and a fixed cache memory.

VI. Conclusion and Recomendation

We successfully implemented and tested a reconfigurable cache memory/controller. The module has 12 cache modes and 6 complementary modes for simplify testing and allow performance analysis. In the reconfiguration process only lasts one clock cycle is controlled by the microprocessor, writing a command to the cache memory/controller. Status parameters like current hit and miss count, and actual running mode can be read from the controller at any time.

Two variants for the reconfigurable cache/controller were presented. In addition, we analyzed the test results for different algorithms in both variants.

We also proposed two methods, still under development, for dynamic reconfiguration. The first method bases on detecting program phases changes. The second one bases on continuously adjusting the external memory region to cache. Taking into account the obtained simulation results, these two methods will be hardware implemented in the future and added as a smart module, to control, dynamically, the reconfiguration process.

In a future work, the cache memory/controller also will be capable to differentiate among kind of miss events (capacity or conflict misses). This could help to carry out an optimal reconfiguration process choosing between, to increase the associativity or the cache size.

References

[1] Gil, A.D.S. Benitez, J.I.B. Calviño, M.H. Gómez, E.H.," Reconfigurable Cache Implemented on an FPGA", 2010 International Conference on ReConFigurable Computing and FPGAs, 2010, 250 – 255, DOI: 10.1109/ReConFig.2010.26.

[2] Ranganathan, P., Adve, S. and Jouppi, N. P., "Reconfigurable caches and their applicat ion to media processing". ACM SIGARCH Computer Architecture News, 2000, 28(2), 214-224. doi: 10.1145/342001.339685.

[3] CACTI 4.0 Tarjan, David; Thoziyoor, Shyamkumar; Jouppi, Norman P. HPL-2006-86 20060606, 2006.

[4] Chen, L., Zou, X., Lei, J., & Liu, Z. (2007). Dynamically Reconfigurable Cache for Low-Power Embedded System. *Third International Conference on Natural Computation (ICNC 2007) Vol V*, (Icnc), 180-184. Ieee. doi: 10.1109/ICNC.2007.346.

[5] Albonesi, D. H. "Select ive cache ways: on demand cache resource allocat ion". Journal of Instruct ion Level Parallelism, May. 2002.

[6] Miguel A. Vega, Raúl Martín, Francisco A. Zarallo, Juan M. Sánchez, Juan A. Gómez. "SMPCache: Simulador de Sistemas de Memoria Caché en Multiprocesadores Simétricos". XI Jornadas de paralelismo, Granada, Spain, Sep. 2000.

[7] H. Kim, A.K. Somani, and A. Tyagi, "A Reconfigurable Mult ifunct ion Computing Cache Archit ect ure", IEEE Transact ions on VLSI, Vol. 9, No. 4, pp. 509-523, Aug., 2001.

[8] Zhang, C., & Vahid, F. A self-tuning cache architecture for embedded systems. ACM Transactions on Embedded Computing System, Vol.3, May. 2004.

[9] Ting, Y., & Chen, B.. Combining select ive cache line replacement and active management for data caching. Thesis. 2005.

[10] Peng, M., Sun, J., & Wang, Y. A Phase-Based Self-Tuning Algorithm for Reconfigurable Cache. First Internat ional Conference on the Digital Society (ICDS'07), 27-27. Ieee. doi: 10.1109/ICDS.2007.2, 2007.

[11] Zhang, C., Vahid, F., & Najjar, W., A highly configurable cache architecture for embedded systems. Proceedings of the 30th Annual International Symposium on Computer Architecture (ISCA'03), June, 2003.

[12] Balasubramonian, R., & Albonesi, D. Memory Hierarchy Reconfigurat ion for Energy and Performance in General Purpose Architecture. Proc of 33 rd Intl Sym on Microarchiterture, 245-257, Dec., 2000.

[13] Cout inho L. M., Mendes J. L., Martins C. A., Dynamically Reconfigurable Split Cache Architecture. 2008 International Conference on Reconfigurable Computing and FPGAs, 163-168. Ieee. doi: 10.1109/ReConFig. 2008.46.

[14] Peng, M., Sun, J., & Wang, Y. A Phase-Based Self-Tuning Algorithm for Reconfigurable Cache. First International Conference on the Digital Society (ICDS'07), 27-27. Ieee. doi: 10.1109/ICDS.2007.2, 2007.

[15] MicroBlaze Processor Reference Guide, Embedded Development Kit.