Optimizing Performance of L1 Cache Memory for Embedded Systems driven by Differential Evolution

Josefa Díaz Álvarez[1] mjdiaz@unex.es
J. Manuel Colmenar[2] josemanuel.colmenar@urjc.es
José L. Risco-Martín[3] jlrisco@dacya.ucm.es
Juan Lanchares[3] julandan@dacya.ucm.es
Oscar Garnica[3] ogarnica@dacya.ucm.es
[1]Centro Univ. Mérida, Univ. de Extremadura, 06800 Mérida, Spain
[2]Dept. of Computer Science and Statistics, Univ. Rey Juan Carlos, 28933 Móstoles, Spain
[3]Department of Computer Architecture and Automation, Univ. Complutense de Madrid, 28040 Madrid, Spain

ABSTRACT

Embedded systems, mainly battery-powered, must guarantee low power consumption and good performance. The cache memory design is crucial in embedded systems. Given the high number of parameters and their ranges of values, the exhaustive exploration of resulting solution space is an unaffordable task.

1. INTRODUCTION

Microprocessors of current portable devices include cache memory, which is one of the most energy-consuming components (20%-30% of the total power of the chip). In fact, the design of the cache memory considerably influences on the performance and the energy consumption and it is determined by the values of its parameters. Moreover, applications present different memory access patterns and require customized cache configurations to meet good performance and low energy consumption needs.

We have designed an optimization process based on Differential Evolution (DE) [1] to identify the best cache configuration for a set of applications.

We optimize both instructions and data cache, together without adding extra hardware complexity. The design space involves cache parameters such as cache size, block size and associativity for both caches; replacement algorithm and prefetch algorithm for instructions cache and write policy for data cache.

2. PERFORMANCE & ENERGY MODELS

We choose the performance and energy model detailed in [2] to measure cache behavior and the ARM9 processor family as our target platform. We consider our system connected to an embedded DRAM memory and with only one cache level where the instructions cache is read-only.

The cache performance and energy models are driven by Eq. (1) and Eq. 2, respectively. We addressed memory cache operations and CPU power is not taking into account.

$$T = (I_a \times I_{at}) + (I_m \times R_{at}) + (I_m \times I_{ls} \times \frac{1}{R_b}) + (D_a \times D_{at}) + (D_m \times R_{at}) + (D_m \times D_{ls} \times \frac{1}{R_b}) \quad (1)$$

$$E = (I_a \times I_{ae}) + (D_a \times D_{ae}) + (I_m \times I_{ae} \times I_{ls}) + (D_m \times D_{ae} \times D_{ls} + I_m \times R_{ap}) + (R_{at} + I_{ls} \times \frac{1}{R_b}) + (D_m \times R_{a_p} + R_{at} + D_{ls} \times \frac{1}{R_b}) \quad (2)$$

3. OPTIMIZATION METHODOLOGY

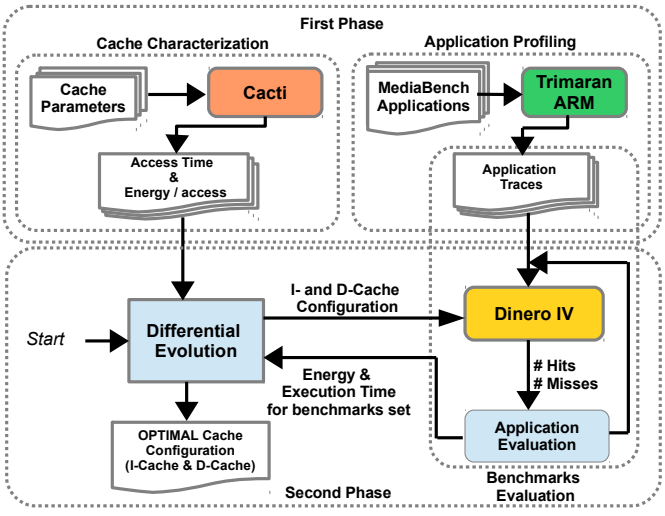

Figure 1: Processes involved in the cache configuration optimization.

The optimization process in Figure 1 consists of two phases, the first phase is in charge of obtaining cache characterization and program traces and must be run just once. Cache characterization uses the analytical model Cacti (32nm), to

GECCO '15 July 11-15, 2015, Madrid, Spain

ACM ISBN 978-1-4503-3488-4/15/07.
DOI: http://dx.doi.org/10.1145/2739482.2764635

compute DRAM and cache access time and dynamic energy consumed; Program traces is carried out by Trimaran and SimpleScalar to compile all memory access of target applications. The second phase driven by the DE algorithm (DE/rand), which works well in continuous spaces. DE has been customized with the following parameters: No. of executions: 30, No. of generations (g_M): 100, Population size (NP): 25, Recombination Factor (RF): 0.3 and Mutation Factor (F): 0.5.

DE explores the search space of cache configurations and for each individual, the cache simulator is called to obtain the number of hits and misses. Then, execution time and energy consumption is computed for the current cache configuration and all the applications. Then, the fitness value is computed according to Eq. (3). After g_M generations, DE obtains an optimized cache configuration, the one that minimizes the fitness for the target applications.

$$f(c_i) = 0.5 \times \frac{T(c_i)}{T(Baseline)} + 0.5 \times \frac{E(c_i)}{E(Baseline)} \quad (3)$$

In this way, $T(c_i)$, $E(c_i)$ are the sum of the execution times and the sum of the energy consumptions over the whole set of applications obtained under the current configuration, c_i.

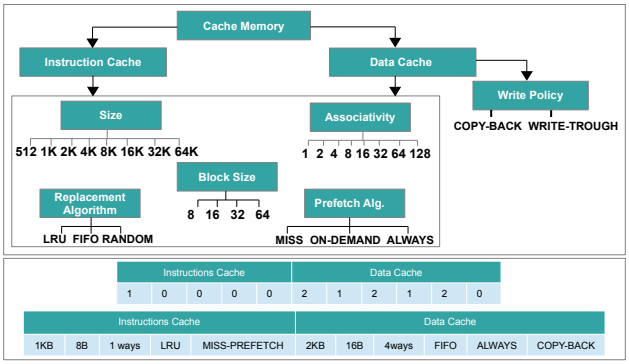

Figure 2: Upper: cache subsystem description and cache parameters values. Bottom: cache configuration, the upper table is the integer vector (genotype) codified to DE algorithm. The bottom table represents the actual cache parameters once decoded.

Thus, according to the search space in Figure 2 (upper), an individual will be decoded as the cache configuration as shown in Figure 2 (bottom).

4. EXPERIMENTAL RESULTS

We applied our heuristic approach to a subset of Mediabench benchmarks (JPEG, MPEG, GSM, PEGWIT, EPIC and ADPCM). Every one has been simulated a maximum of 7.5×10^7 instructions to prevent the capture of a partial or phase behavior. On the other hand, our algorithm has been run 30 times, in order to reduce the probability of falling into a local optimum.

We selected a baseline configuration that is similar to the cache of the first core in the GP2X portable game console.

For each run, DE algorithm evaluates all target applications for each individual and returns the best one, with represents a cache configuration that improves energy consumption and performance for the whole set of applications. Figure 3 shows the experimental results.

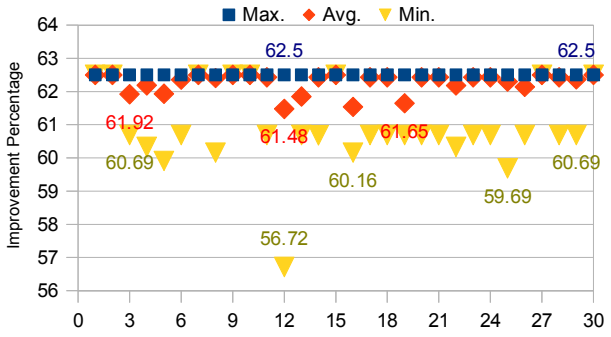

Figure 3: Minimum, maximum and average improvement value for 100 generations and 30 runs obtained by DE for the set of benchmarks.

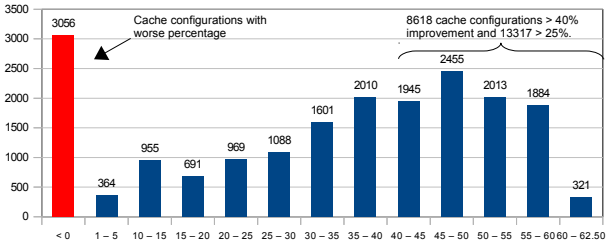

Figure 4: Improvement percentages of fitness values for 30 runs of DE algorithm for Mediabench benchmarks over to baseline configuration. The x-axis represents improvement percentage ranges, while the y-axis is the number of different solutions for each interval. The red bar represents solutions that get worse fitness values than the baseline.

5. CONCLUSIONS AND FUTURE WORK

The algorithm is able to find a cache configuration that improves more than 62.5% the chosen baseline for applications selected. Moreover, different cache configurations share this improvement percentage and an improvement percentage higher than 50% has been got by 22% of cache configurations evaluated.

Acknowledgment

This work has been supported by Ministry of economy and competitiveness under project uex:ephemech: Bio-inspired algorithms in ephemeral environments, TIN2014-56494-C4-2-P.

6. REFERENCES

[1] R. Storn, K. Price, Differential evolution-a simple and efficient adaptive scheme for global optimization over continuous spaces, ICSI Berkeley, 1995.

[2] A. Janapsatya, A. Ignjatovic, S. Parameswaran, Finding optimal L1 cache configuration for embedded systems, in: Design Automation, 2006. Asia and South Pacific Conference on, 2006, p. 6 pp. doi:10.1109/ASPDAC.2006.1594783.