Dynamic Power Reduction in Self-Adaptive Embedded Systems through Benchmark Analysis

Alberto Scionti[1], Stamatis Kavvadias[2], Roberto Giorgi[1]
[1]Dept. of Information Engineering and Mathematics, University of Siena, Siena, Italy
{scionti, giorgi}@dii.unisi.it
[2]Technological Educational Institute of Crete, Dept. of Computer Science, Iraklio, Crete Greece
kavadias@cs.teicrete.gr

Abstract—**Discovering the most appropriate reconfiguration instants for improving performance and lowering power consumption is not a trivial problem. In this paper we show the benefit in terms of performance gain and power reduction of the dynamic adaptation (e.g., cache size, clock frequency, and core issue-width) of an embedded platform, through a design space exploration campaign, and focusing on a relevant case study. To this end, we analyze a set of benchmarks belonging to the embedded application domain with the aim of illustrating how the appropriate selection of reconfiguration instants can positively influence system performance and power consumption. Experimental results using the *cjpeg* benchmark show that power consumption can be reduced by an average of 22%. Our methodology can be used to create a set of run-time management policies for driving the adaptation process.**

Keywords - reconfigurable architectures, low-power electronics, embedded software

I. INTRODUCTION

Nowadays, high-performance embedded systems integrate an ever larger number of cores and functionalities into a single silicon die. In this context, designers must take into account specific constraints, such as power consumption, area occupation, security, resiliency, and clock frequency (e.g., [4][6]). The Embedded Reconfigurable Architectures (ERA) project [1][13][14][15] aims at providing a high-performance energy-efficient embedded architecture, leveraging on the adaptability of the platform to the specific application features and requests. Run-time adaptation to the application specifics requires the capability of discovering points in the execution, where the platform may try to change its configuration improving performance and energy efficiency. Benchmark suites have been proposed to verify the performance of a given architecture with respect to a specific domain (e.g., [2][3]). Further, benchmark suites and work-loads analysis represents a starting point from which discovering the best set of reconfiguration points, and developing run-time approaches for platform adaptation.

Prior research works characterized benchmark suites using both microarchitecture– dependent (MD) metrics and microarchitecture– independent (MI) metrics. In [5], the authors have analyzed benchmark applications using different techniques (e.g., trace simulation, statistical analysis, etc.) with the aim of capturing inherent program characteristics. Although, less accurate considerations can be obtained by considering only MD metrics, these results can provide useful information for dynamically reconfiguring the system, by correlating the application behavior with energy/power consumption. Synthetic benchmarks are much easier to develop, maintain, and use with respect to real applications. While they replicate some workload characteristics into a synthetic trace, they are not able to capture the complexity of a real world application and its correlation with the energy/power consumption [5].

By selecting proper metrics to characterize applications, it is possible to determine and predict program phases. Several studies [7][8] showed that changes in the application phases (i.e., a portion of the executed code that repeats itself in time and that is characterized by a similar behavior on the target architecture) represent the best points where the platform may try to adapt to the changed application behavior. Several approaches have been proposed to discover program phases. Sembrant et al. [8] proposed an efficient on-line phase detection and classification approach, based on common features exhibited by modern microprocessors, such as performance counters. Other works attempt to automatize, speed up, and to extend the analysis process to a parallel architecture [7]. Proposed methodologies are quite general, but rely on hardware features which are not always available on embedded systems, and lack of correlation among the discovered phases and the energy/power consumption measured during application execution.

The main contributions of this paper are as follows. First, we provide the designers with a set of benchmarks (ERA Benchmark Suite – EBS [14][15]) and ranges where it is possible to observe the variation of relevant architecture parameters (issue-width, cache-size, frequency) useful for exploring the design space of run-time reconfiguration of the system. Then, we illustrate our methodology for analyzing the benchmark applications, determining the best appropriate reconfiguration points, and evaluating the impact on the platform adaptation. To this end, we give a detailed analysis of a representative case study to show how to apply the above knowledge in order to achieve a reduction of dynamic power. Finally, we show the main results of an extensive analysis of the reconfiguration potentials that are available in the EBS benchmark applications.

II. THE ERA BENCHMARK SUITE

In the context of the ERA project, we have put together a set of interesting benchmarks for the embedded domain (i.e., the ERA Benchmark Suite – EBS [14][15]). To this purpose, we selected representative applications, by choosing a new generation mobile device as the main scenario. Specifically, the benchmarks include image (*cjpeg*, *djpeg*, *susan*), video (*x.264*, *mpeg2*), and audio (*ac3*) processing applications, as well as a security (*EC-DS*) and a text recognition (*tesseract*) application. All the selected applications are written in C/C++ language. We compiled all the EBS applications for the X86_64 ISA (matching the architecture of the host simulation machine), with no specific compiler optimizations (we used the open source *gcc* compiler). It is worth to observe that although X86_64 is widely adopted in high-performance architectures, low-power versions specifically designed for embedded systems are available, such as in current smartphones and tablet computers (e.g., Intel ATOM architecture [12]).

III. METHODOLOGY

Our methodology aims at discovering the most appropriate points where the system can dynamically change its configuration, thus gaining in terms of performance or power reduction. To this purpose, we deeply analyze the behavior of the target benchmark applications. This analysis requires the ability of measuring several metrics, such as overall performance (instructions per cycle – IPC), power consumption, etc. Different metrics can be classified into two categories: microarchitecture– dependent (MD) and microarchitecture– independent (MI) measurements. For this work we chose to deeply analyze applications behavior using MD metrics, which allow us to simply correlate application behavior to the power consumption and performance of the platform. Specifically, we considered caches, TLBs and branch predictor miss rates, overall performance (IPC), energy and power consumption. For our experiments we used the *COTSon/SimNow* simulator, which have been considerably developed in the TERAFLUX project [9][11]. Our workloads present only integer and control instructions, including memory read and write operations.

Our methodology for the identification of most suitable reconfiguration points exploits an on-line analysis approach. The output of the *objdump* tool allows us to associate a compile-time identifier (ID) of functions, loops, and library code structures (we statically linked all the required libraries with the benchmark binary) with their virtual program counter (PC). During simulations all the selected MD metrics are tracked, allowing the construction of a code-structure vector (CSV) containing the frequency of each specific metric.

Providing these frequency vectors to *Simpoint* tool, we are able to detect application phases. By comparing phases with other metrics, we can identify potential reconfiguration points for the target architecture. This methodology can be further extended in order to detect reconfiguration points at run-time. Finally, to extract detailed power measurements per interval, we modified *McPAT* tool, so that it can run repeatedly with different inputs, but calculate the targeted architecture only the first time, since this is the most time consuming part of its execution. This analysis may guide the design of a set of policies that allows the programmer to select among quality-of-service (QoS) classes for different parts of the code, through a specific API.

TABLE 1 – Architecture configuration parameters and their related ranges.

Feature	Description	Configuration
Core	X86_64 ISA, in-order	2-4-8 wide
L1 caches	I/D 16KB each, 2-way, 64B lines, allocate, WB, single-port	–
L2 cache	Unified, 16-way, allocate, WB, single-port	64KB, 128KB, 256KB, 512KB, 1MB
TLBs	I/D 40-entry each, 4KB pages	–
Main Mem.	70ns latency	–
Clock Freq.	Fixed for each run	800MHz, 1.1GHz, 1.4GHz, 1.7GHz, 2.0GHz

A. Configuration Space Exploration

We considered a large experimental campaign using different architecture configurations. Table 1 shows the range of configurations we considered.

Although in this work we focus on the adaptation of a single core embedded system, all the considerations can be further extended to include heterogeneous multi-cores, specialized accelerators, and a network-on-chip (NoC) communication infrastructure.

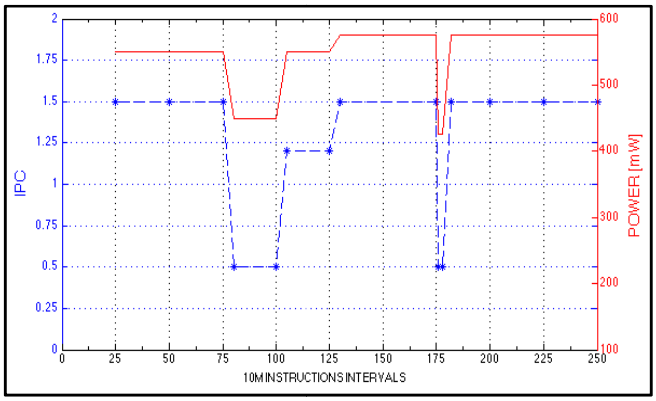

Fig. 1 – Simulation of the *cjpeg* application. The blue dashed line shows the IPC, while the red line shows the corresponding power consumption.

To reduce simulation time we only modeled processor width and memory access stalls, but not instruction dependences and resource conflicts. This overstates both performance increase with processor width, and with frequency, as the model assumes infinite parallelism in the application, unless

interrupted by cache misses. Although this leads to a non-perfectly cycle-accurate characterization of the applications, it gives us a fast and robust way for the understanding of reconfiguration choices.

IV. SIMULATION RESULTS

A. Analysis of a case study

To demonstrate the effectiveness of our methodology we considered the following case study. We analyzed the simulation results of the *cjpeg* application over the full range of system configurations (see table 1). Figure 1 shows the general behavior of the test case application running on a system equipped with a 8-wide superscalar processor, a L2 cache with a size of 1024kB, and running at full speed (i.e., 2GHz). Blue dashed line represents the IPC values sampled over the simulation intervals, while the red line shows the power consumption in each interval, expressed in mW. Detailed analysis of the performance curve (blue dashed line) shows the presence of three regions where the IPC value falls down from an initial value of 1.5 to 0.5, and 1.2. These three regions can be observed respectively between 75M and 100M of instructions, between 100M and 125M of instructions, and between 175M and 180M of instructions. Red line shows three inverse power consumption peaks in correspondence of these low performance regions.

Similarly, figure 2 shows the application behavior in terms of cache misses. The figure is composed of several graphs presenting the cache misses split in read and write misses over the application execution. The analysis of the plots clearly shows that read misses are dominant in this case. Blue peaks represent the regions in the benchmark execution where the performance (IPC) decreases due to the negative impact of the cache misses of read operations. Since the peaks are not affected by changes in the cache configuration, we can advocate that these regions represent good candidates for dynamic system reconfiguration.

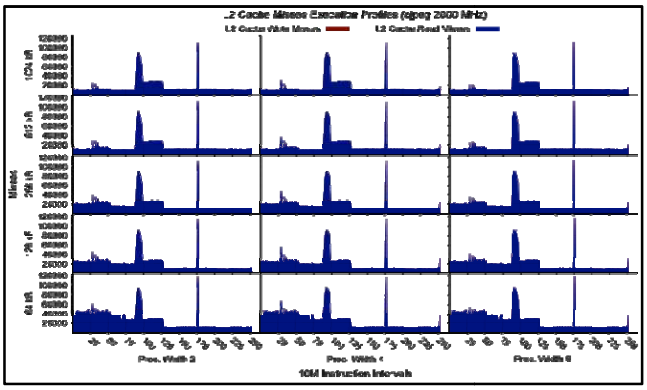

Fig. 2 – L2 cache misses for the *cjpeg* application running on a processor with different issue-width and cache size (clock speed constant at 2GHz).

Change in the system configuration, by decreasing the clock speed, reducing the L2 cache size, and lowering the issue-width of the processor, results in a performance curve with similar poor performance regions, located in the same instruction intervals. Simpoint simulation confirms that the three selected instruction intervals are good candidate points for the system reconfiguration. Since these points are characterized by a similar performance reduction, irrespective of the system configuration, we decided to reconfigure the architecture with the aim of lowering its overall power consumption.

Figure 3 depicts the test case application behavior in terms of system performance (IPC) and power consumption, when running on the adapted system. Whenever the system enters in one of the three reconfiguration regions, it automatically reduces the clock speed to 800MHz, reduces the processor issue-width to 2, and disables part of the L2 cache (i.e., the total size of the L2 cache becomes equal to 64kB). As expected, the IPC curve still presents low performance peaks, but power consumption is greatly reduced. More precisely, the power consumption in the candidate reconfiguration regions decrease from 425mW to 175mW. The global power consumption reduction is confirmed by computing the average value for the overall simulation. In the first case (i.e., fixed configuration with a system equipped with a 8-wide processor, 1024kB L2 cache, and running at 2GHz), the average performance value is IPC_{avg} = 1.19, while the average power consumption is P_{avg} = 530mW.

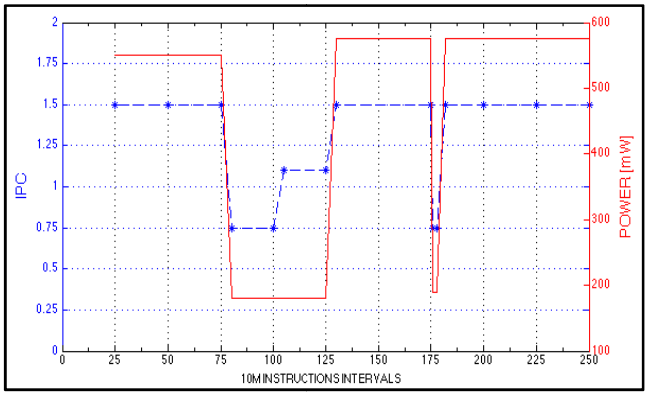

Fig. 3 – A demonstration of power reduction for the *cjpeg* application. The blue dashed line shows the IPC, while the red line shows the corresponding power consumption.

In the second case, we are able to reduce the power consumption to the average value P_{avg}= 413mW, or about a 22% reduction, while the performance remain in line with an average value of IPC_{avg} = 1.24. Of course, this analysis neglects side effects of the reconfiguration, such as in the case of the L2 cache resize. In the real system, resizing the cache may imply the need of the system to reload part of the memory regions that have been evicted after the cache size reduction. Similarly, we neglected possible impacts of the operating system in both performance and power consumption after run-time reconfiguration. From this viewpoint, the previous analysis allows us to determine a limit in the performance gain or in the power reduction associated to the run-time reconfiguration.

B. Extension to other EBS benchmarks

Benchmark analysis can be further extended to the other EBS applications. Due to limited space, we present simulation results for the image processing (*susan*), video processing (*mpeg2*), audio processing (*ac3*), and text recognition (*tesseract*) applications. Figure 5 presents four graphs, each of them depicts the average energy dissipation (measured in mJ) over the entire benchmark execution for different system configurations. Each graph shows a general trend: energy dissipation and execution time (measured in ms) decrease when the configuration moves from low speed to high speed clock signals, and when the cache size and the issue-width increase (i.e., power consumption increases). As stated for the *cjpeg* application, there are regions in the applications execution that are not affected by configuration changes (not shown due to space limitations). Correlating this information with the power/energy execution profiles we can select appropriate reconfiguration points.

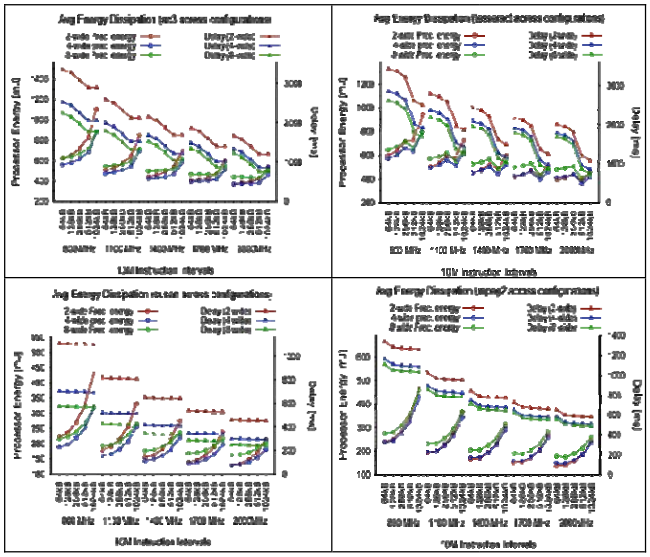

Fig. 4 – Average energy dissipation (mJ) and execution time (ms) over instruction intervals for 4 representative EBS benchmark applications.

Our analysis, confirmed by Simpoint simulations, shows that for the regions in the application execution interval in which the performance are not affected by configuration changes, is possible to dynamically reduce power dissipation by forcing the platform to assume minimal configurations. The observed trends confirm the expected behavior, but also validate the advantages showed in the dynamic changes of figure 1 and figure 3. For these cases we also expect that the execution time will be not significantly affected.

V. Conclusions

Benchmark characterization and phase classification have been largely used to statically optimize the configuration of the embedded systems. Further, they can be exploited to dynamically change the system configuration. Monitoring specific metrics over the application execution, it is possible to discover potential points where a change in the system configuration is beneficial in terms of performance gain or power consumption reduction. In this paper we have shown that these points are present in representative embedded applications. Analyzing real applications on different system configurations is possible to determine a limit in the performance gain and power reduction, helping the designer to optimize the system from a dynamic perspective.

Acknowledgements

We would like to thank Dr. Nikola Puzovic for his propaedeutic work for this paper. This work is partly supported by the European Commission in the context of the ERA (Embedded Reconfigurable Architectures) collaborative project no.249059 (FP7) and the TERAFLUX project no.249013.

References

[1] Wong S., Carro L., Kavvadias S., Keramidas G., Papariello F., Scordino C., Giorgi R., Kaxiras S., Embedded Reconfigurable Architecture, Proc. of the ACM International Conf. on Compilers, Architectures and Synthesis for Embedded Systems (CASES), 2012.

[2] D. Guthaus, et al.; MiBench: A free, commercially representative embedded benchmark suite, Workload Characterization, IEEE International Workshop on, 2001.

[3] Poovey, J.A., et al.; A Benchmark Characterization of the EEMBC Benchmark Suite, Micro, IEEE, vol.29, no.5, Sept.-Oct. 2009.

[4] Basile C., Di Carlo S., Scionti A., FPGA-based remote-code integrity verification of programs in distributed embedded systems, IEEE Transactions on Systems, Man and Cybernetics Part C: Applications and Reviews, 2012.

[5] Hoste, K.; Eeckhout, L.; Microarchitecture-Independent Workload Characterization, Micro, IEEE, vol.27, no.3, May-June 2007.

[6] Di Carlo S., Prinetto P., Scionti A., A FPGA-based reconfigurable software architecture for highly dependable systems, Proc. of the Asian Test Symposium (ATS), 2009.

[7] Perelman E., et al., Detecting phases in parallel applications on shared memory architectures, In Int. Parallel and Distributed Processing Symposium, 2006.

[8] Sembrant, A.; Eklov, D.; Hagersten, E.; Efficient software-based online phase classification, Workload Characterization (IISWC), 2011 IEEE International Symposium on, 6-8 Nov. 2011.

[9] Portero A., Simulating the future kilo-x86-64 core processors and their infrastructure, Simulation Series 44 (2 BOOK).

[10] M. B. C. Alioto, et al.; Exploiting Locality to Improve Leakage Reduction in Embedded Drowsy I-Caches at Same Area/Speed, IEEE Int. Symp. On Circuits and Systems (ISCAS), May 2010.

[11] Giorgi R., et al., TERAFLUX: Harnessing dataflow in next generation teradevices, Microprocessors and Microsystems, Available online Apr. 2014, http://dx.doi.org/10.1016/j.micpro.2014.04.001

[12] Zahir, R.; Ewert, M.; Seshadri, H., "The Medfield Smartphone: Intel Architecture in a Handheld Form Factor," *Micro, IEEE*, 2013.

[13] Wong S., Carro L., Rutzig M., Matos D. M., Giorgi R., Puzovic N., Kaxiras S., Cintra M., Desoli G., Gai P., Mckee S. A., Zaks A.; ERA: Embedded Reconfigurable Architectures; *Springer New York*, Aug 2011.

[14] Puzovic N., et al., A Multi-Pronged Approach to Benchmark Characterization, IEEE Int. Conf. on Cluster Computing, Heraklion, Greece, Sept. 2010.

[15] Wong S., et al., Early Results from ERA – Embedded Reconfigurable Architectures, IEEE INDIN, Lisbon, Portugal, 2011.