Characterizing Phase Behavior for Dynamically Reconfigurable Architectures

Zhibin Yu, Nikola Puzovic, Antoni Portero, Roberto Giorgi

Dept. Ingegneria dell' Informazione, University of Siena, Via Roma 56, 53100 Siean, Italy

ABSTRACT

Previous researches have shown most programs have phase behavior. We would like to take advantage of the phase behavior of applications to dynamically reconfigure an embedded platform in order to achieve more energy efficiency and performance. In this study, we first outline some aspects of the phase behavior with emphasis also on the influence of the Operating System (OS). We developed a tool chain to characterize the phase behavior of workloads (benchmarks) for embedded reconfigurable systems. Eight typical workloads are analyzed by using our tool chain. The results show that i) phase behavior can be relevant in workloads for embedded reconfigurable systems; ii) phase behavior (including the number of phases and the time spent in each phase) is reflected in several types of system metrics; iii) there are more phases when OS activities are involved for the same program; iv) the time spent by each phase of a program varies in a wide range. We believe the outcomes of this work can be used to guide the dynamic reconfiguration of components in embedded reconfigurable systems.

KEYWORDS: Workload Characterization; Program Phase; Reconfigurable Computing

1 Introduction

Recently, embedded reconfigurable architectures have gained a strong momentum. However, it is still a challenge to know when and how to reconfigure the hardware resources of these systems dynamically. One of the main reasons for the problem is that applications for embedded reconfigurable systems have a wide range of behaviors. Therefore, it is vital to accurately characterizing the workload behavior.

Previous researches have demonstrated most programs for general purpose CPUs such as SPEC CPU benchmarks have phase behavior [1][2][3]. However, it is not clear yet whether workloads for embedded reconfigurable systems also have the similar behavior, especially when operating system activities are involved. Although there are some studies have been done on characterizing phase behavior for embedded systems, they still did their experiments by using SPEC CPU benchmarks such as SPEC CPU 2000 [4][5].

A program phase is a set of intervals within the program's execution that have similar behavior (e.g. in terms of required performance) regardless of temporal adjacency. The phase behavior provides an indispensable criterion for the dynamic reconfiguration of embedded reconfigurable systems. For example, the architecture may be dynamically reconfigured at runtime to provide higher performance or to save power based on phase behavior. Therefore, it

is crucial to characterize the phase behavior of the workloads for embedded reconfigurable systems.
In this work, we develop a tool chain to characterize the phase behavior of workloads for embedded systems. Eight workloads [8] such as *AC3, MPEG2* are characterized. Since our target embedded reconfigurable system is a multi-core architecture which employs VLIW processors, the effects of OS activities must be taken into account as well. We compare the phase behavior of the same program with and without OS activities. In addition, we employ the *xSTSim* which is a VLIW processor simulator to measure the time ratio spent in each phase. This information can be used to guide the dynamic reconfiguration of embedded reconfigurable systems.

2 Related Work

There is a large amount of researches have been done on characterizing program behavior for general purpose processors. However, quite few works have been done for embedded dynamically reconfigurable systems.
The most recent benchmark suite for reconfigurable architectures is GroundHog [6]. P. Jamieson et al. designed these benchmarks to help measure and optimize power consumption in mobile domain. J. Poovey et al. characterized the EEMBC benchmark suite for MIPS, PowerPC, x86, and PISA architectures [7]. They used micro-architecture independent metrics such as dynamic instruction percentages (integer, floating-point, load, store, branch, e.g.) and micro-architecture dependent properties such as cache miss rate and branch mis-prediction ratios to characterize the benchmarks. Their primary contribution is the analysis of the similarity of performance behavior among the programs. They did not analyze the phase behavior while our work does.
F. Vandeputte et al. exploited program phase behavior for energy reduction on multi-configuration processors [4]. P. Nagpurkar et al. employed the phase-aware sampling techniques to efficiently profile remote resource-constrained devices [5]. Although these researches studied the program phase behavior for reconfigurable systems, they did their experiments by using SPEC CPU 2000 benchmarks which is targeting for general purpose processors. Our work focuses not only on the general phase behavior characterization but also on more specifically the workloads for embedded reconfigurable systems.

3. Metrics and Algorithms

Since our target platform is a reconfigurable system in which the micro-architectures may vary when programs are running, we mainly choose micro-architecture independent metrics to characterize phase behavior. However, the time information is also very important for the reconfiguration of our embedded reconfigurable system, we use one micro-architecture dependent metric in our characterization.
For the phase characterization of memory access, we employ global stride, local stride, working set size, and stack distance as our metrics. These metrics are micro-architecture independent. The micro-architecture dependent metric is cycles per bundle (CPB). As mentioned before, our target platform employs VLIW cores and the VLIW cores issue instructions as bundles. We, therefore, use CPB as the metric to study whether there is still phase behavior on VLIW processors and the time ratio spent in each phase if available.
Generally, the two primary clustering approaches are partitioning and hierarchical. Partitioning algorithms choose an initial solution randomly and then use iterative updates to find a better solution. K-means and Gaussian Expectation Maximization are popular algorithms in this

family. The run time of these algorithms tends to be linear in the size of the dataset. Hierarchical algorithms either combine together similar points or recursively divides the dataset into more groups. The runtime of them tends to be quadratic in the size of the dataset. Since k-means is a very fast and simple algorithm that yields good results [1], we choose k-means as our algorithm to detect phase behavior.

4. Experiments and Results

To quantify how much time spends in each phase, we use a simulator named xSTSim from ST Microelectronics. The processor of the xSTSim is configured as ST231 which is a VLIW processor from the same company. As a beneficial side effect, we also observed the program phase behavior on VLIW processors. We also modified an emulator QEMU to characterize phase behavior using the metrics mentioned above. Since space is limited, we do not show the results in this paper. Only the time information of phases and the impact of OS on them are reported.

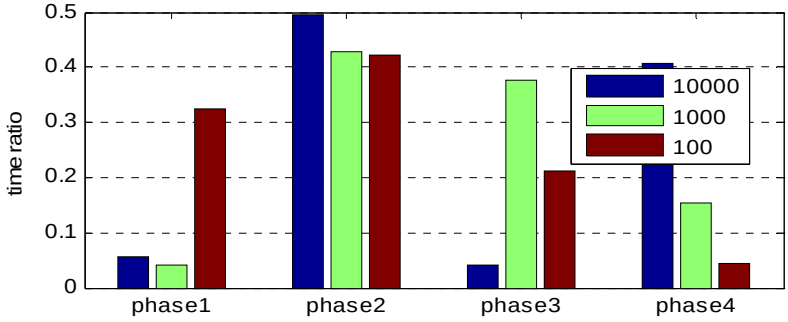

Figure 1: The time ratio of each phase in program AC3 when different measurement granularities are used. The CPB (Cycles Per Bundle) is measured in three granularities which are 10000, 1000, and 100 bundles.

Figure 1 shows the time ratio spent in phases of AC3. A larger time-ratio means that a program spends more time in the corresponding phase (ratio varies between 0 and 1). Although the number of phases is the same for the three different measurement granularities, the time spent in the same phase is much different. The time ratio of phase 1 is less than 5% when the CPB is measured per 10,000 bundles while it is higher than 30% when the CPB is measured per 100 bundles. On the contrary, the time ratio of phase 4 is higher than 40% when the CPB is measured per 10000 bundles while it is lower than 5% when the CPB is measured per 100 bundles. This is because the length of phases is not a multiple of the measurement granularity. Each phase has its natural boundary in a program. Therefore, it is very important to select a proper measurement granularity when we reconfigure the system resources based on phase behavior. We leave this issue as a future work.

Figure 2 shows the number of phases with OS activities of DJPEG. There are 10 distinct phases in the program and the instructions belong to the same phase may not be adjacent. Figure 3 shows the phases of DJPEG detected by the user-mode of our tool chain. Comparing the two figures, we found there are more phases when OS activities are involved. The other benchmarks show the similar results. This finding indicates that we should take into account the OS activities when the resources are dynamically reconfigured based on phase behavior.

5. Conclusions

To help reconfigurable systems reconfigure their resources dynamically, we study the phase

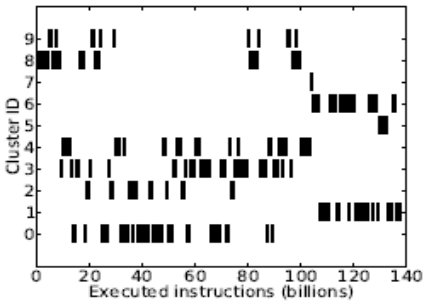

Figure 2 The detected phases of djpeg by using global stride with OS activities

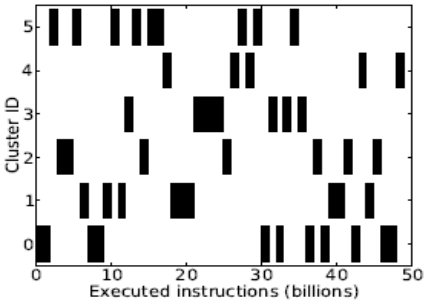

Figure 3 The detected phases of djpeg by using global stride without OS activities

behavior of workloads for them. First the OS activities have a significant impact on phase behavior of our experimented benchmarks, indicating we must take into account the OS activities when reconfiguring the system resources. Second, the time spent in each phase varies wildly, suggesting that it is not necessary to reconfigure resources for the short phases. Third, different number of phases may be detected for the same program by using variant metrics. This implies we should choose the metric carefully when we setup the reconfiguration strategies. Finally, we find the measurement granularities used in program phase detection significantly affects the results, including the number of phases and the time spent in each phase of a program.

Acknowledgements

This work is partially funded by the European FP7 project ERA, ID: 249059, http://www.era-project.eu/ and HiPEAC IST-217068.

References

[1] T. Sherwood, E. Perelman, and B. Calder, "Basic block distribution analysis to find periodic behavior and simulation points in applications," in Proc.Int. Conf. Parallel Architectures and Compilation Techniques., Sept. 2001, pp.3-14.

[2] T. Sherwood, E. Perelman, G. Hamerly, and B. Carder, "Automatically characterizing large scale program behavior," in Proc. Int. Conf. Archiecture Support for Programming Languages and Operasting Systems., Oct 2002, pp.45-57.

[3] A. S. Dhodapkar, and J. E. Smith, "Comparing program phase detection techniques," in Proc. Int. Symp. Micro-architecture., Dec 2003, pp.217-227.

[4] F. Vandeputte, L. Eeckhout, and K. D. Bosschere, "Exploiting program phase behavior for energy reduction on multi-configuration processors," Journal of Systems Architecture., vol.53, no.8, Aug. 2007, pp.489-500.

[5] P. Nagpurkar, H. Mousa, C. Krintz, and T. Sherwood, "Efficient remote profiling for resource-constrained devices," ACM Trans. Architecture and Code Optimization., vol. 3, no.1, Mar. 2006, pp. 35–66.

[6] P. Jamieson, T. Becker, P. Cheung, W. Luk, T. Rissa, and T. Pitkanen, "Benchmarking and evaluating reconfigurable architectures targeting the mobile domain," ACM Trans. Design Automation. Elect. Syst., vol.15, no. 2, Feb. 2010, pp.1-24.

[7] J. Poovey, M. Levy, S. Gal-On, and T. Conte, "A Benchmark Characterization of the EEMBC Benchmark Suite," IEEE Micro., vol.29, no.5, Sept. 2009, pp.18-29.

[8] N. Puzovic S. McKee R. Eres A. Zaks P. Gai S. Wong R. Giorgi, "A Multi-Pronged Approach to Benchmark Characterization", IEEE Int.l Conf. on Cluster Computing (CLUSTER2010), ISBN:978-1-4244-8396-9, Heraklion, Greece, Sept. 2010, pp. 1-4