2014 17th Euromicro Conference on Digital System Design

Fast System Level Benchmarks for Multicore Architectures

Alper Sen*, Gokcehan Kara* Etem Deniz*, Smail Niar†
*Department of Computer Engineering, Bogazici University, Istanbul, Turkey 34342
†ISTV2 UVHC, Campus Mont Houy - 59313, Valenciennes Cedex 9, France
alper.sen@boun.edu.tr, gokcehan.kara@boun.edu.tr, etem.deniz@boun.edu.tr, smail.niar@univ-valenciennes.fr

***Abstract*—We present a framework that automatically generates system level synthetic benchmarks from traditional benchmarks. Synthetic benchmarks have similar performance behavior as the original benchmarks that they are generated from and they can run faster. Synthetics can also be used as proxies where original applications are not available in source form. In experiments we observe that not only are our system level benchmarks much smaller than the real benchmarks that they are generated from but they are also much faster. For example, when we generate synthetic benchmarks from the well-known multicore benchmark suite, PARSEC, our benchmarks have an average speedup of 149x over PARSEC benchmarks. We also observe that the performance behavior of synthetics have more than 85% similarity to the real benchmarks.**

***Index Terms*—Synthetic benchmarks, SystemC, Parallel patterns, Performance evaluation, Multicore architecture**

I. Introduction

The cost of performance evaluation has been gradually increasing with the increasing complexities of the systems and the benchmarks that run on them. System level approaches, such as virtual platforms, are commonly used to tackle the complexity problem in design automation. There is also a need to develop system level performance evaluation techniques to speed-up the design process and avoid costly redesigns. In system level performance evaluation, abstract system level models of hardware and software components are used with added timing details to provide relative figures for comparing different design options in early design phases. During later design phases accurate techniques are used for more precise estimations, however these techniques require a lot of implementation details. In this paper, we develop techniques that allow us to perform system-level performance evaluation using synthetic benchmarks.

A synthetic benchmark is a small, simple and accurate program that mimics the performance characteristics of an original benchmark that it is derived from. Synthetic benchmarks have multiple advantages over traditional benchmarks in performance evaluation. First, synthetic benchmarks can be used for speeding up early architectural exploration and performance analysis. Although Transaction Level Models (TLM) [1] of hardware are fast, these models are slower when timing details are added for performance studies. Therefore, benchmarks that run on these hardware models need to be fast and synthetic benchmarks can provide this speed by abstracting the functional behavior of the original benchmark. Second, since developing new benchmarks from scratch in a new domain is a labor-intensive process, automatic synthetic benchmark generation can help relieve this burden. For example, it is desirable to automatically generate system level synthetics for applications in well-known multicore benchmark suites such as PARSEC [2]. This process can essentially create an "equivalent" of these benchmark suites at the system level. Third, synthetic benchmarks can act as a proxy, hence allowing the sharing of proprietary IPs. For example, our synthetics are generated using the binary of the original application (not the source code) and will not have any resemblance to the source code of the original application. Therefore, the use of synthetic benchmarks also avoids disclosing the source code of the application. Design space exploration by a third party may be implemented using only the synthetic code.

The main motivation to use synthetic benchmarks comes from the slow-downs resulting from the co-simulation of SystemC and Pthreads programs since different simulators need to be accurately synchronized in co-simulation. In this scenario, the original application uses the Pthreads library and the synthetic application uses the SystemC library. We obtain a system level synthetic version of an original multithreaded application, say from PARSEC benchmark suite, that can be plugged into an existing SystemC design. This capability is important because multithreaded applications are commonly used in performance evaluation of newly developed architectures including virtual platforms that contain hardware and software components. In these virtual platforms, SystemC is commonly used to describe hardware components. Furthermore, PARSEC benchmarks are large and slow even in a non-co-simulation environment, and this has led to the development of non-system-level synthetic benchmarks from PARSEC suite as described in related work. Hence, rather than doing architectural exploration or performance evaluation of an existing SystemC design using a potentially large and slow Pthreads application in a co-simulation environment, we can accomplish the same performance evaluation using a small and fast synthetic SystemC application that is similar to the Pthreads application and without the need for co-simulation.

We perform experiments on original benchmarks from PARSEC suite. The average speedup of our SystemC synthetics is 149x compared to the original PARSEC benchmarks. All our benchmarks are similar to the original benchmarks on average 85%, where the similarity is defined as the average of

978-1-4799-5793-4/14 $31.00 © 2014 IEEE
DOI 10.1109/DSD.2014.16

the error percentage between the synthetic and the original using instructions-per-cycle (IPC), cache-miss-rate (CMR), and branch-misprediction-rate (BMR).

Note that our work is orthogonal to the more precise performance estimations that require a lot of implementation details. One can use synthetic benchmarks during early design phases to compare different design options. Precise estimations still need to be performed when the implementation details are available.

To the best of our knowledge, this is the first work on automatically generating synthetic benchmarks from real benchmarks at the system level.

II. Related Work

There are numerous studies on benchmark cloning and synthetic benchmark generation using workload characterization. Hoste et al. [3] compare micro-architecture dependent and micro-architecture independent characteristics for commonly used benchmarks. Joshi et al. [4] propose a method to analyze workloads of proprietary applications. Our work is similar to Deniz et al. [5] who generate synthetic benchmarks based on software architectural patterns for Pthreads programs and use binary instrumentation. However, they do not target system level benchmarks like us. That is, we can generate SystemC synthetics from Pthreads programs using a new system-level back-end.

Several studies estimate the performance and workload characteristics of SystemC and TLM models. SESAME [6] uses Kahn process networks for multimedia application modeling. TAPES [7] is a performance evaluation tool using transaction level simulation in SystemC. In TAPES, the applications are modeled as traces whose specification is a manual process. Similarly, Streubuhr et al. [8] propose a methodology for heterogeneous multiprocessor systems-on-chip specified at ESL level. Kreku et al. [9], [10] developed a framework which uses UML and SystemC for system-level performance evaluation. They have an abstract workload model of applications obtained using several approaches such as analytical modeling, simulation traces, and measurements. The workloads are modeled at a low level, that is, at the basic block level and contain read, write, execute instructions; whereas, we model workloads at a high level and capture the parallel pattern of the original application. They manually add counters, timers, and probes for measuring performance, whereas we obtain program behavior through automatic instrumentation and hardware counters. Their processor models have a cycles per instruction (CPI) parameter similar to us, which is used in estimating the execution time of the workloads. Grammatikakis et al. [11] developed a method to estimate the power usage of cycle and bit accurate TLM models. Number of transactions and bit transitions at each clock cycle is used to compute the relative power dissipation. Greaves et al. [12] describe a power estimation add-on to SystemC TLM modeling.

Our work differs from above mentioned SystemC/TLM works in that our goal is to generate synthetic versions of real applications with similar performance characteristics. We do not develop a system level platform model rather we develop synthetic system level applications. Our synthetics do not work at the basic block level instead we generate a synthetic based on high level parallel pattern of the application. Also, we do not depend on the source code of the application but just the binary of it. Our synthetics are regular C programs which make them portable, unlike low level assembly programs used for synthetics previously [13]. Similar to above approaches, we abstract the functionality of the application as synthetic benchmarks do not perform any useful function. None of the above SystemC/TLM works generate synthetics from real benchmarks.

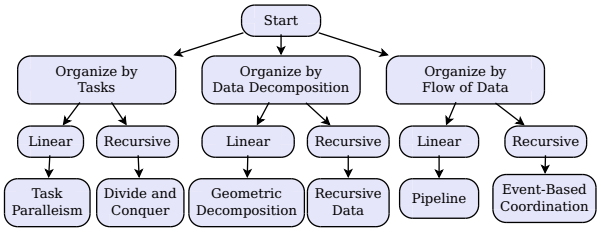

Fig. 1. Classification of Parallel Patterns [14]

III. Workload Characteristics

In this section, we describe high-level (parallel patterns) and low-level workload characteristics used in our framework.

Patterns are high quality solutions to commonly encountered problems. Parallel software patterns are simply those that are applicable to problems related to concurrent programs. Different problems require different parallel patterns for their implementations. Mattson et al. [14] present a set of parallel patterns that are commonly used in the literature that are shown in Fig. 1.

A parallel pattern is a high level characteristic of a parallel program, hence these characteristics allow us to improve portability of our synthetic benchmarks on different platforms while preserving thread communication and data sharing behaviors as demonstrated by our experiments. Whereas, low level characteristics are dependent on the particular platform that they are obtained from. Different parallel programming patterns can be implemented in SystemC or Pthreads.

We also use low-level characteristics in determining whether the synthetic is similar to the original benchmark. The most commonly used low level characteristics for hardware performance evaluation are; Instruction-Per-Cycle (IPC), which denotes the average number of instructions per CPU cycle, Cache-Miss-Rate (CMR), which denotes the average number of cache misses per cache reference, and Branch-Misprediction-Rate (BMR), which denotes the average number of branch misses per branch instruction.

IV. System Level Synthetic Benchmark Generation

We developed a synthetic benchmark generation framework that takes as input a binary program and outputs a synthetic

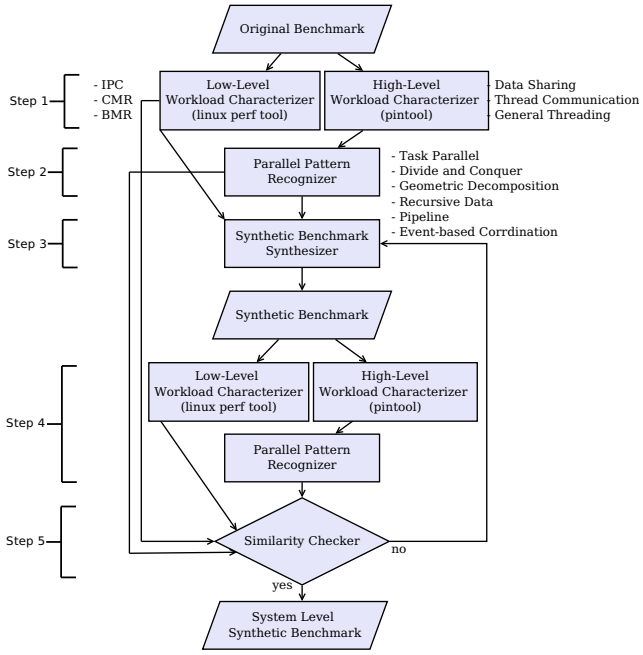

Fig. 2. System Level Synthetic Benchmark Generation Flow

program that mimics the behaviors of the input program. Our algorithm is an adaptation of the algorithm in [5] to SystemC. The steps in our synthesis process are namely:

Step 1: Original Benchmark Characterization
Step 2: Original Benchmark Pattern Recognition
Step 3: Synthetic Benchmark Synthesis
Step 4: Synthetic Benchmark Characterization and Pattern Recognition
Step 5: Original-Synthetic Similarity Comparison

Our workflow begins with collecting the characteristics of the original benchmark. Afterwards, we use this information to generate an initial version of the synthetic benchmark with the same recognized parallel pattern as the original one. We then instrument the synthetic benchmark so that we can converge to similar performance values as the original by iteratively adding necessary code blocks to the synthetic benchmark. This evolutionary algorithm continues until the user specified similarity thresholds are satisfied.

V. Experiments

We implemented our system level synthetic benchmark generation techniques in a tool. Note that none of the works in the literature generate system level benchmarks hence we cannot use those works for comparison purposes. We generated SystemC synthetics from Pthreads originals, which were taken from the well-known PARSEC multicore benchmark suite [2]. We generated the synthetics on HW1, which has Intel Xeon e5520 processors (8 cores and 8MB cache) with 32GB RAM running Ubuntu 12.04. After a successful synthetic generation on HW1, we used two different machines to test the portability of these synthetic programs. HW2 has Intel Core i5 processor (2 cores and 3MB cache) with 4GB RAM running Ubuntu 12.04 and HW3 has Intel Xeon e5-2680 processors (16 cores and 20MB cache) with 32GB RAM running OpenSuse 12.03. The maximum number of iterations for synthetic generation was set to 100 and a timeout of 1 hour was used. Also, the minimum overall and individual IPC, BMR, and CMR similarity scores were set to 80% and 70%, respectively.

Table I shows the parallel patterns and lines of code of the original and synthetic benchmarks as well as the speedup on HW1 and the number of iterations to obtain the synthetic. For PARSEC benchmarks we use medium inputs. Note that the penalty due to synthetic generation time is small compared to the number of times a typical benchmark is run during the design process. Hence, the resulting fast synthetic leads to speedup in the overall design time.

Similarity and portability of synthetics: Our experimental results are shown in Figure 3. All our benchmarks are similar to the original benchmarks on average 85% and above. It can be seen that the synthetic is following the behavior of the original on all three hardware configurations. That is, when the IPC, CMR, or BMR of the original increases or decreases, the synthetic changes similarly on a new configuration. The average error in IPC is less than 30%, in the branch prediction rate is less than 1% and in the cache hit rate is less than 25% for all three architectures as expected. Note that the cache size of HW3 is much larger than the other two architectures resulting in larger errors. Overall the experiments show that similarity of our benchmarks is high, hence they can be used for performance evaluation. We observe that the similarity is higher for original benchmarks that are large. This is because both our high-level and low-level performance metrics work better when the sample size that is proportional to the execution time of the benchmark is large.

Speedup of synthetics: The average speedup of our synthetics is 149x for HW1, 145x for HW2, and 129x for HW3. We observe that task parallel applications have the highest speedup. This is because communication is minimal in task parallel applications hence it is possible to completely take advantage of parallelism in this case.

VI. Conclusions and Future Work

We presented a framework to generate system level synthetic benchmarks from multithreaded applications. To the best of our knowledge, this is the first work on generating synthetic benchmarks from real benchmarks at the system level. Our synthetic benchmarks are smaller and faster than the original benchmarks, while maintaining similar performance characteristics as the original benchmarks. Specifically, we developed system level synthetics of multicore benchmarks from the well-known PARSEC suite and obtained an average speedup of 149x. In the future, we plan to generate Pthreads synthetics from SystemC original models that will speed up the execution of SystemC models.

	ORIGINAL (Pthreads)		SYNTHETIC (SystemC)			
Benchmark	**Loc**	**Parallel Pattern**	**Loc**	**Parallel Pattern**	**Speedup**	**#iterations**
Facesim	20275	Task Parallel	414	Task Parallel	1115	6
Fluidanimate	2784	Geometric Decomposition	415	Geometric Decomposition	29	7
Bodytrack	7696	Geometric Decomposition	397	Geometric Decomposition	59	84
Blackscholes	1262	Task Parallel	223	Task Parallel	47.5	13
Swaptions	1095	Task Parallel	788	Task Parallel	230.6	11
Ferret	10765	Pipeline	386	Pipeline	44.5	29

TABLE I
PARSEC (PTHREADS) ORIGINAL TO SYSTEMC SYNTHETIC RESULTS

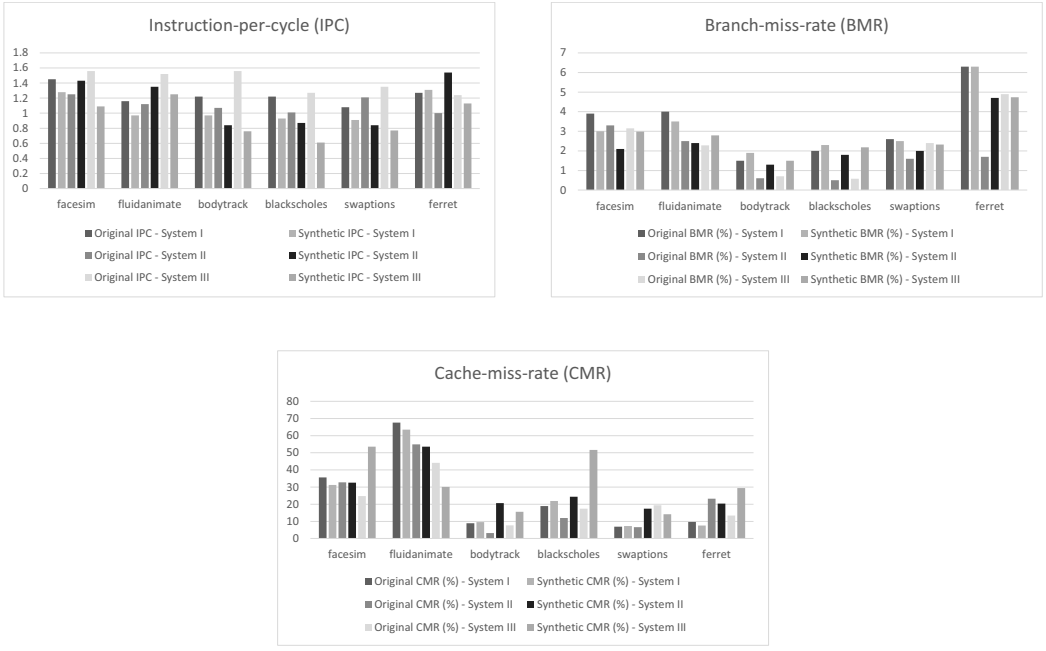

Fig. 3. Performance metrics of original (Pthreads, PARSEC) and synthetic (SystemC) benchmarks on 3 different hardware configurations

VII. ACKNOWLEDGEMENTS

This research was supported by Bogazici University Research Fund 7223 and the Turkish Academy of Sciences.

REFERENCES

[1] "Accelera Systems Initiative, http://www.accelera.org/," 2012.

[2] C. Bienia, S. Kumar, J. P. Singh, and K. Li, "The PARSEC Benchmark Suite: Characterization and Architectural Implications," in *International Conference on Parallel Architectures and Compilation Techniques*, 2008.

[3] K. Hoste and L. Eeckhout, "Microarchitecture-independent workload characterization," *IEEE Micro*, vol. 27, no. 3, pp. 63–72, 2007.

[4] A. Joshi, L. Eeckhout, R. H. Bell,Jr., and L. K. John, "Distilling the essence of proprietary workloads into miniature benchmarks," *ACM Trans. Archit. Code Optim.*, vol. 5, no. 2, pp. 10:1 – 10:33, 2008.

[5] E. Deniz, A. Sen, J. Holt, and B. Kahne, "Using software architectural patterns for synthetic embedded multicore benchmark development," in *IEEE International Symposium on Workload Characterization (IISWC)*, 2012.

[6] M. Thompson and A. D. Pimentel, "Towards multi-application workload modeling in sesame for system-level design space exploration," in *International conference on Embedded computer systems: architectures, modeling, and simulation (SAMOS)*, 2007.

[7] T. Wild, A. Herkersdorf, and G.-Y. Lee, "Tapes-trace-based architecture performance evaluation with systemc," *Design Automation for Embedded Systems*, vol. 10, no. 2-3, pp. 157–179, Sep. 2005.

[8] M. Streubuhr, R. Rosales, R. Hasholzner, C. Haubelt, and J. Teich, "ESL power and performance estimation for heterogeneous MPSOCS using SystemC," in *Forum on Specification and Design Languages (FDL)*, 2011.

[9] J. Kreku, M. Hoppari, T. Kestila, Y. Qu, J.-P. Soininen, P. Andersson, and K. Tiensyrja, "Combining uml2 application and systemc platform modelling for performance evaluation of real-time embedded systems," *EURASIP J. Embedded Syst.*, vol. 2008, pp. 6:1–6:18, Jan. 2008.

[10] J. Kreku, K. Tiensyrjä, and G. Vanmeerbeeck, "Automatic workload generation for system-level exploration based on modified gcc compiler," in *Proceedings of the Conference on Design, Automation and Test in Europe*, 2010.

[11] M. Grammatikakis, S. Politis, J.-P. Schoellkopf, and C. Papadas, "System-level power estimation methodology using cycle- and bit-accurate tlm," in *Design, Automation Test in Europe Conference Exhibition (DATE)*, 2011.

[12] D. Greaves and M. Yasin, "Tlm power3: Power estimation methodology for systemc tlm 2.0," in *Forum on Specification and Design Languages (FDL)*, 2012.

[13] K. Ganesan and L. K. John, "Automatic generation of miniaturized synthetic proxies for target applications to efficiently design multicore processors," *IEEE Transactions on Computers*, vol. 99, no. PrePrints, p. 1, 2013.

[14] T. Mattson, B. Sanders, and B. Massingill, *Patterns for parallel programming*. Addison-Wesley Professional, 2004.