Advances in Computer Science Research, volume 80
International Conference on Computer Science, Electronics and Communication Engineering (CSECE 2018)

Effect of Workload Characteristics on Similarity Analysis

Jiang Sha[1,*] and Wenjuan Xu[2]
[1]National ASIC System Engineering Technology Research Center, Southeast University, Nanjing, China
[2]Institute of Integrated Circuits Technology, Southeast University, Wuxi, China
*Corresponding author

***Abstract*—Workload characterization is the basis for similarity analysis, which is the core idea behind benchmark subsetting to pick up the most representative programs or program slices. The set of characteristics is crucial to the result of similarity analysis. Current studies typically use microarchitecture-independent characteristics (MICs) which reveal the inherent program behaviors to evaluate the similarities. In this paper, we propose a novel MICs: serializing instruction distance (SID). SID can describe the serializing instructions behavior that causes a significant performance loss of system-intensive mobile applications. The distribution of critical path length is also used as a MICs because it can reflect the inherent instruction level parallelism (ILP). Furthermore, we employ the comprehensive set of MICs to pick a representative set of program slices for each program of a mobile benchmark suites: Moby. The instructions per cycle (IPC) of each program slice is used to predict the whole program performance. The coefficient of variation of IPCs is under 6% and weighted average IPC prediction error is only 7%.**

Keywords—microarchitecture-independent characteristics; similarity analysis; mobile applications; serializing instruction

I. INTRODUCTION

Benchmarking is the foundation of processor performance evaluation and prediction [1] [2]. However, due to quantities of dynamic instructions of running benchmarks and the low speed of cycle-accurate simulator, the tedious simulation cost too much time to estimate performance. So it is necessary to subset benchmarks by reducing the total number of dynamic instructions. The current methodology for this purpose mainly involves sampling simulation, statistical simulation and workload synthesis. The core idea behind these techniques is to pick up the most representative programs or program slices through similarity analysis.

Workload characterization is the basis for similarity analysis of program behaviors. There are two approaches to extract workload characteristics. One is microarchitecture-dependent characteristics (MDCs) and the other is microarchitecture-independent characteristics (MICs). MDCs define a set of metrics dependent on microarchitecture, mainly including instructions per cycle (IPC), cache miss rate and branch misprediction rate. Unfortunately, MDCs are often inconsistent on various platforms with different microarchitectures, the conclusions drawn from MDCs are only applicable for the specific microarchitecture. In some cases, even applications have the same performance on the same platform, their inherent behaviors are still different [3] [4]. Inherent behavior is the behavior of program itself, not affected by the platform. Consequently, due to independency on microarchitectures, MICs are widely regarded as workload characteristics to represent program inherent behaviors. Traditionally, MICs are categorized into several groups consisting of instruction mix, inherent instruction level parallelism (ILP), register dependency distance, memory access locality, instruction locality and branch behaviors [5] [6] [7] [8] [9] [10].

MICs are essential to workload characterization based on inherent behaviors. For example, clustering based on shape or color of the boxes in Figure I. , we will get two different results (subset A and subset B). Thinking of these boxes as benchmarks, MICs directly decide that the subsetting framework will export to different results of subsets, which have different representativeness. So the definition of MICs is crucial. The workload characteristics should cover all important program behaviors which influence the performance to the maximum extent. However, mobile applications express some different behaviors from traditional benchmarks due to frequent system calls and libraries invocations [11] [12] [13].

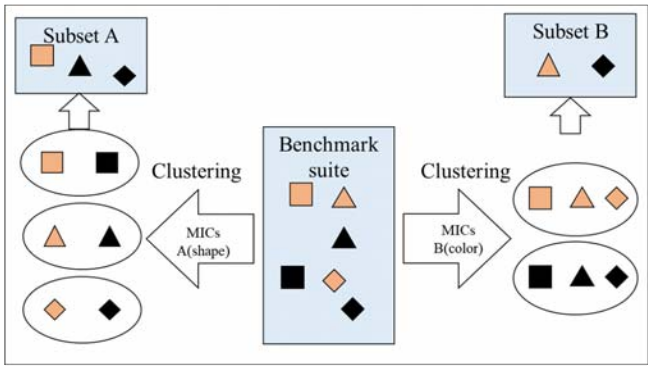

FIGURE I. EFFECT OF MICS ON BENCHMARK SUBSETTING.

Therefore, the purpose of this paper is to find the inherent program behaviors of Android applications and research on the effect of new characteristics on similarity analysis. In a summary, our contributions are as the followings:

- We propose serializing instruction distance (SID) as a new MIC to represent the serializing instruction behavior.
- We firstly employ the critical path length distribution as the corresponding MICs of inherent ILP.

- We firstly use coefficient of variation to validate the representativeness of the set of program slices.

II. METHODOLOGY

A. *Microarchitecture-independent Workload Characterization*

1) Serializing instruction distance:

As an important behavior of Android applications, the serializing instructions (SIs) have a significant impact on performance of system-intensive workloads. SIs can cause a 3–17% overall performance loss in some benchmarks running on Linux OS [14]. In our cases of three Android applications and three SPEC benchmarks, we measure the amount of SIs and stalled cycles due to SIs on a full-system simulator (details are mentioned in section III).

As shown in Figure II. , SIs of the most left three Android applications occupy only 1-2% of total dynamic instructions but contribute 16-22% of total cycles. While there is almost no SIs executed in the three SPEC benchmarks on the right. Frequent SIs have a significant impact on performance. Therefore, we propose a new MIC, SID to represent the serializing instruction intensity statistically. SID is defined as the number of instructions between two adjacent serializing instructions. There is a large amount of serializing instructions to generate too many SID. Hence, to distinguish the differences of variable SI behaviors, we statistically calculate the distribution of SID (P_{SID}) ranging from 2^{i-1} to 2^i-1 (i is in {0, 1, …, 10}). Equation (1) gives its definition.

$$P_{SID} = \begin{cases} \frac{T_i}{Inst_{SI}}, & (i = 0) \\ \sum_{d=2^{i-1}}^{2^i-1} \frac{T_d}{Inst_{SI}}, & (0 < i \leq 10) \end{cases} \quad (1)$$

Where, T_d is the times of SID equal to d, $Inst_{SI}$ is the total numbers of SI.

FIGURE II. INSTRUCTION MIX AND % CYCLES SPENT STALLED DUE TO SIS.

2) Critical path length:

A serial of instructions with strong dependence cannot be out-of-order execution, especially for long dependency chains, which will limit the ILP. Critical path length is defined as the number of instructions in the longest dependent chain. We use the distribution of critical path length to reflect the dependence between instructions. For the convenience of measurement, we actually collect the critical paths in the range of every fixed amount of instructions (refer to instruction window size, we use 40 in this paper). So we statistically calculate these characteristics to get a probability distribution from 1 to 40.

Besides, we measure spatial and temporal locality of memory and instruction. Because they are particularly important with respect to cache behaviors. We also use the proportion of register read and write and register dependency distance. To express the complexity of control flow behavior, we measure basic block size, branch direction (forward or backward), average branch taken rate, average branch transition rate and branch spatial locality. All the MICs employed in this paper are shown in TABLE I.

TABLE I. MICROARCHITECTURE-INDEPENDENT CHARACTERISTICS

Group	No.	Characteristic/Metric
Instruction mix	1-6	integer, floating-point, branch, read, write and serializing instruction
Critical path length	7-46	cpath1~40
Register traffic	47 48	register read, register write
	49-68	register_dependent_distance1~20
Instruction locality	69-89	instAddr0~20
	90-100	instReuse0~10
Memory locality	101-112	loadGlobalAddr0~11
	113-124	loadLocalAddr0~11
	125-136	storeGlobalAddr0~11
	137-148	storeLocalAddr0~11
	149-159	loadGlobalReuse0~10
	160-170	loadLocalReuse0~10
	171-181	storeGlobalReuse0~10
	182-192	storeLocalReuse0~10
Control flow complexity	193-201	basicBlockSize0~8
	202-205	fwBranch, bwBranch, takenBranch, transitionRate
	206-216	branchAddr0~11
Serializing instruction distance	217-227	serialInstDist0~10

B. *Verification Standard*

We group program slices into clusters according to their MICs and choose the most representative slice in each group using K-means.

Usually, weighted average IPC is used to evaluate the representativeness of those selected slices in a result that each slice's contribution to the cluster is ignored. So we exploit coefficient of variation (CV) and weighted average IPC to achieve more accurate similarity analysis. The verification algorithm is as follows:

1) Calculating the average CV of IPCs of all slices in each cluster.

2) Calculating the prediction error of weighted average IPC of representative slices for each cluster.

The CV is a well-known statistical metric to consider the uncertainty of a random variable. CV is generally defined as the standard deviation (square root of variance) divided by the mean [15]. The smaller CV is better. Therefore, we use the CV to evaluate the uniformity of IPC.

Furthermore, the slices closest to the centers of the clusters are chosen to be the representatives, and the proportions of dynamic instructions in the clusters become their weights (w_r). Then we use the weighted average IPC of representative slices (IPC_r) to predict the program's IPC as shown in (2).

$$IPC_{Predicted} = \sum_{r=1}^{K} w_r \cdot IPC_r \quad (2)$$

III. RESULTS

A. Experiment Setup

1) Platforms and tools

We modify the gem5 simulator [16] to collect MICs and MDCs of program slices for each program. Gem5 is a cycle-accuracy architecture simulator. It supports ARM ISA which is a prevalent mobile platform and other ISAs. We conduct our experiments on *arm_detailed* mode of gem5 by setting default configuration which similar to ARM Cortex-A15.

2) Benchmarks

Moby is a mobile benchmark suite which contains a diverse set of Android applications [13]. It also includes another benchmark named BBench [11]. They are all real-world Android applications and have been ported to gem5 simulator. We use five of them and also choose three benchmarks from SPEC CPU 2006 [20] as a comparison. The benchmarks used in this paper are listed in TABLE II.

TABLE II. BENCHMARKS

Suite	Program	Operation/Input
Moby	BaiduMap	Load an area's map
	BBench	Load web pages
	JingDong	Load information
	K9Mail	Load/Show emails
	KingsoftOffice	Open a doc file
SPEC CPU 2006	bzip2	Train inputs
	gcc	
	mcf	

B. Validation of Representativeness of Program Slices

We separate the five Android applications into 97,824 slices by context switches. Putting all the program slices into the same workload space, we use our proposed methods to extract 227 dimensions of MICs, and group them into K distinct clusters by K-means to pick a representative set of program slices. Based on the achieved representative program slices, we carry out the effect of SID and critical path length on similar analysis.

1) CV of IPCs inside clusters:

It is necessary to validate whether the IPC of slices in the same cluster are basically consistent. Figure III. gives average CVs of IPCs at different reduction ratios in cases of all MICs, MICs except SID and MICs except SID and critical path length respectively. The reduction ratio refers to the total number of program slices divided by K. In a general way, we hope to use as few slices as possible to represent the entire program.

It is obvious that CV is the smallest when SID and critical path length are included in MICs. And with the increase in reduction ratio, this advantage is more obvious, which is clearly beneficial to performance evaluation.

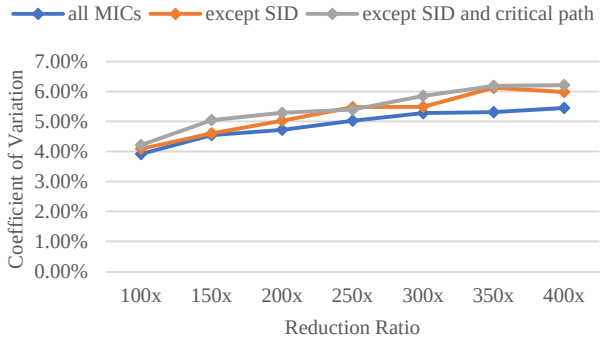

FIGURE III. COEFFICIENT OF VARIATION OF IPCS.

2) Weighted average IPC:

Using the representative slices' IPCs along with their weights referenced in equation (2), we can predict the IPCs of the programs respectively. Figure IV. shows when reduction ratio equals to 400, the IPC prediction errors of the five Android applications range from 0.5% to 15%. The average error can reach only 7%. As a comparison, we give the IPCs predicted based on all MICs and MICs except SID and critical path length. Overall, the differences between the two results are very small, but the former results slightly better than the latter in some programs.

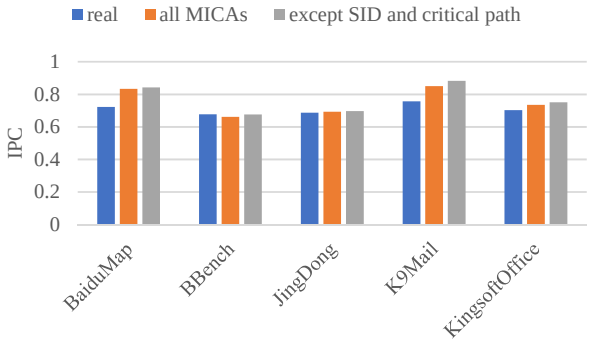

FIGURE IV. PREDICTION ERROR OF IPC (400X REDUCTION RATIO).

IV. RELATED WORK

The researches on program similarity analysis using MICs have fruitful achievements [3] [4] [5] [6] [7] [8]. Eeckhout et al. [5] found similarities of samples in SPEC2000 using 47 MICs and then selected 148 simulation points of 32,964 ones in SPEC2000. The error could be 1.11% by comparing the CPI between the selected simulation points and the completed benchmark simulation. Joshi et al. [6] selected a representative subset of 5 programs from 22 benchmarks in SPEC2000 using 29 MICs to further reduce the number of benchmarks need to be simulated. Hoste et al. [8] extended this methodology from general purpose applications to domain specific applications: BioPerf (bioinformatics) [17], BioMetricsWorkload (biometrics) [18] and MediaBench II (multimedia) [19].

However, the above benchmark suites are all traditional benchmark suites. Like the most widely used general purpose

benchmarks SPEC CPU [20], by design, stress primarily the CPU and are meant to be portable, avoiding extensive use of system calls and shared libraries. While real-world smartphone applications, in contrast, suffer more performance penalties incurred due to use of these [11]. Huang et al. [13] presented a mobile benchmark suite including 10 popular Android applications, named Moby, and captured some MICs: instruction mix, working sets, reuse distance distributions and instruction execution flow. Besides, Wells and Sohi [14] illustrated that frequent SIs have a significant impact on performance of system-intensive workloads. SIs can cause a 3–17% overall performance loss in some benchmarks running on Linux OS.

Obviously, considering the characteristics of mobile applications on Android OS, it is necessary to find new MICs that can extract program SI behavior features. Furthermore, previous set of MICs [4] define inherent ILP as IPC of an idealized out-of-order processor (with perfect caches and branch predictor). From the measurability point of view, the prime necessity is to build an ideal CPU, which is a very difficult task. So it is also necessary to define new metrics of inherent ILP in a better way. And then achieve a more comprehensive set of MICs for representative program slices choice during workload characterization.

V. CONCLUSION

In this paper, we propose the serializing instruction distance as a novel MIC and involve the critical path length to make similarity analysis. Using the comprehensive set of MICs, we pick a representative set of program slices and validate their representativeness. We find that the CV of IPCs inside clusters decreases when SID and critical path length are included in the set of MICs. And the average IPC prediction error of Android applications in Moby is 7% with the reduction ratio of 400.

ACKNOWLEDGMENT

This research was supported by the National Science and Technology Major Project (Grant No. 2014ZX01030-101).

REFERENCES

[1] L. Eeckhout, “Computer Architecture Performance Evaluation Methods,” Synthesis Lectures on Computer Architecture, vol. 5. pp. 1–145, 2010.

[2] J. L. Hennessy and D. A. Patterson, Computer architecture: a quantitative approach. 2012.

[3] K. Hoste and L. Eeckhout, "Comparing Benchmarks Using Key Microarchitecture-Independent Characteristics," 2006 IEEE International Symposium on Workload Characterization, San Jose, CA, 2006, pp. 83-92.

[4] K. Hoste and L. Eeckhout, "Microarchitecture-Independent Workload Characterization," in IEEE Micro, vol. 27, no. 3, pp. 63-72, May-June 2007.

[5] L. Eeckhout, J. Sampson and B. Calder, "Exploiting program microarchitecture independent characteristics and phase behavior for reduced benchmark suite simulation," IEEE International. 2005 Proceedings of the IEEE Workload Characterization Symposium, 2005., 2005, pp. 2-12.

[6] Ajay Joshi, Aashish Phansalkar, L. Eeckhout and L. K. John, "Measuring benchmark similarity using inherent program characteristics," in IEEE Transactions on Computers, vol. 55, no. 6, pp. 769-782, June 2006.

[7] A. Phansalkar, A. Joshi, L. Eeckhout and L. K. John, "Measuring Program Similarity: Experiments with SPEC CPU Benchmark Suites," IEEE International Symposium on Performance Analysis of Systems and Software, 2005. ISPASS 2005., Austin, TX, 2005, pp. 10-20.

[8] K. Hoste and L. Eeckhout, "Characterizing the Unique and Diverse Behaviors in Existing and Emerging General-Purpose and Domain-Specific Benchmark Suites," ISPASS 2008 - IEEE International Symposium on Performance Analysis of Systems and software, Austin, TX, 2008, pp. 157-168.

[9] W. Alkohlani, J. Cook, and N. Siddique, “Insight into Application Performance Using Application-Dependent Characteristics,” in Performance Modeling, Benchmarking and Simulation of High Performance Computer Systems - PMBS ’14, vol. 8966, S. A. Jarvis, S. A. Wright, and S. D. Hammond, Eds. Cham: Springer International Publishing, 2015, pp. 107–128.

[10] W. Alkohlani and J. Cook, "Towards Performance Predictive Application-Dependent Workload Characterization," 2012 SC Companion: High Performance Computing, Networking Storage and Analysis, Salt Lake City, UT, 2012, pp. 426-436.

[11] A. Gutierrez et al., "Full-system analysis and characterization of interactive smartphone applications," 2011 IEEE International Symposium on Workload Characterization (IISWC), Austin, TX, 2011, pp. 81-90.

[12] D. Pandiyan, S. Y. Lee and C. J. Wu, "Performance, energy characterizations and architectural implications of an emerging mobile platform benchmark suite - MobileBench," 2013 IEEE International Symposium on Workload Characterization (IISWC), Portland, OR, 2013, pp. 133-142.

[13] Y. Huang, Z. Zha, M. Chen and L. Zhang, "Moby: A mobile benchmark suite for architectural simulators," 2014 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS), Monterey, CA, 2014, pp. 45-54.

[14] P. M. Wells and G. S. Sohi, "Serializing instructions in system-intensive workloads: Amdahl’s Law strikes again," 2008 IEEE 14th International Symposium on High Performance Computer Architecture, Salt Lake City, UT, 2008, pp. 264-275.

[15] Wencheko, E., and P. Wijekoon. "Improved estimation of the mean in one-parameter exponential families with known coefficient of variation." Statistical Papers 46.1 (2005): 101-115.

[16] N. Binkert, S. Sardashti, R. Sen, K. Sewell, M. Shoaib, N. Vaish, M. D. Hill, D. A. Wood, B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. R. Hower, and T. Krishna, “The gem5 simulator,” ACM SIGARCH Comput. Archit. News, vol. 39, no. 2, p. 1, 2011.

[17] D. A. Bader, Yue Li, Tao Li and V. Sachdeva, "BioPerf: a benchmark suite to evaluate high-performance computer architecture on bioinformatics applications," IEEE International. 2005 Proceedings of the IEEE Workload Characterization Symposium, 2005., 2005, pp. 163-173.

[18] Chang-Burm Cho, A. V. Chande, Yue Li and Tao Li, "Workload characterization of biometric applications on Pentium 4 microarchitecture," IEEE International. 2005 Proceedings of the IEEE Workload Characterization Symposium, 2005., 2005, pp. 76-86.

[19] Chunho Lee, M. Potkonjak, and W. H. Mangione-Smith, “MediaBench: a tool for evaluating and synthesizing multimedia and communications systems,” in Proceedings of 30th Annual International Symposium on Microarchitecture, 1997, pp. 330–335.

[20] SPEC CPU2006 benchmark suite. http://www.spec.org/cpu2006/