JCSE Journal of Computing Science and Engineering

Regular Paper

Journal of Computing Science and Engineering,
Vol. 8, No. 1, March 2014, pp. 34-42

Exploiting Standard Deviation of CPI to Evaluate Architectural Time-Predictability

Wei Zhang* **and Yiqiang Ding**
Department of Electrical and Computer Engineering, Virginia Commonwealth University, Richmond, VA, USA
wzhang4@vcu.edu, dingy4@vcu.edu

Abstract
Time-predictability of computing is critical for hard real-time and safety-critical systems. However, currently there is no metric available to quantitatively evaluate time-predictability, a feature crucial to the design of time-predictable processors. This paper first proposes the concept of architectural time-predictability, which separates the time variation due to hardware architectural/microarchitectural design from that due to software. We then propose the standard deviation of clock cycles per instruction (CPI), a new metric, to measure architectural time-predictability. Our experiments confirm that the standard deviation of CPI is an effective metric to evaluate and compare architectural time-predictability for different processors.

Category: Embedded computing

Keywords: Performance; Reliability; Hard real-time systems; WCET analysis; Time-predictability

I. INTRODUCTION

Processor architectural design has traditionally focused on improving average-case performance or energy efficiency. However, for real-time systems, time-predictability is the most important design consideration. Unfortunately, many performance-enhancing architectural features of today's microprocessors, such as caches and branch predictions, are detrimental to time-predictability [1, 2]. As a result, an estimation of worst-case execution time (WCET) on modern microprocessors, which is crucial for hard real-time and safety-critical systems, is very difficult, if not impossible, to make [1, 2]. On the other hand, a time-predictable processor with low performance may be ineffective or even useless. To achieve both time-predictability and performance, researchers have proposed a time-predictable design for microprocessors [2] with the goal to achieve time-predictability (or WCET analyzeability) while minimizing the impact on average-case performance.

In the last two decades, researchers have proposed several designs of time-predictable processors for real-time systems. Colnaric and Halang [3] proposed a simple asymmetrical multiprocessor architecture without dynamic architectural features, such as pipelines and caches, for hard real-time applications. Delvai et al. [4] designed a scalable processor for embedded real-time applications, which employed a simple 3-stage pipeline without cache memory, and Edwards and Lee [5] proposed a precision timed (PRET) machine. Yamasaki et al. [6] studied instructions per cycle (IPC) control and prioritization of multithreaded processors. Paolieri et al. [7] examined time-predictable multicore architectures to support WCET analyzability. Finally, Schoeberl [8] proposed time-pre-

Open Access http://dx.doi.org/10.5626/JCSE.2014.8.1.34 http://jcse.kiise.org

Received 26 July 2013; **Revised** 18 December 2013; **Accepted** 10 February 2014
*Corresponding Author

pISSN: 1976-4677 eISSN: 2093-8020

dictable Java processor.

However, to the best of our knowledge, none of the prior work has quantitatively evaluated time-predictability. While (average-case) performance can be easily assessed, most prior work on time-predictable design either simply reported the worst-case performance through measurement or analysis or just qualitatively explained that the design was time-predictable by removing undesirable architectural features. This is because, to the best of our knowledge, there is no well-defined metric to evaluate time-predictability of microprocessors, and this has fundamentally limited the advance of time-predictable architectural design. The lack of a metric does not only prevent designers from performing a quantitative comparison of different time-predictable designs in order to select the better one, but also makes it impossible to quantitatively analyze the trade-off between time-predictability and performance since these two goals often conflict with each other.

To address this problem, this paper proposes to use the standard deviation of cycles per instruction (CPI) to measure the time-predictability of different architectures. Such a metric can quantitatively indicate the variation in execution time of different architectural/microarchitectural designs, and thus can provide useful insights for engineers to make better design tradeoffs between performance and time-predictability.

II. ARCHITECTURAL TIME-PREDICTABILITY

The execution time of a real-time task (or generally a program) can be calculated by using Eq. (1), where T is the execution time, N is the number of instructions, CPI represents the clock cycles per instruction, and τ stands for the clock cycle time. The computer architecture community has used this equation to evaluate the average-case performance of microprocessors for decades, and we can also use it explain the variation of the execution time. In this equation, τ is fixed for each instruction, which is a predictable quantity after the processor is designed and implemented. In contrast, both N and CPI can vary significantly, thus making it hard to achieve time-predictability. Specifically, the number of instructions (N) can be affected by both software and instruction set architecture (ISA). Given an ISA, the number of instructions is mainly determined by the software and its inputs, i.e., which paths are executed at runtime. The CPI is affected by the architectural/microarchitectural design. Usually in a pipelined processor, different types of instructions take various clock cycles, and even the same type of instructions may take different clock cycles. For example, the latencies of load instructions depend on whether they hit in the cache or not.

$$T = N \times CPI \times \tau \tag{1}$$

Since our goal is to design time-predictable processors (to support running real-time software), we should separate the time variation caused by software and hardware. For example, the inputs that are given, or the program paths that are executed, should not be a concern for hardware design. Prior work [2] seems to consider time-predictability of both the software and hardware as a single problem, making it overly complicated, if not impossible, to define a useful metric to specifically guide hardware design for time-predictability. In this paper, we propose a concept called architectural time-predictability (ATP) using the definition below.

Definition 1 (Architectural time-predictability). *Given a number of instructions, architectural time-predictability indicates the degree that the architecture under study can provide predictable execution time.*

In the above definition, the number of instructions of the ISA is assumed to be known, which we believe is a reasonable assumption to separate the time-predictability of software and hardware. Once the ISA is defined, the number of instructions executed is mostly affected by software characteristics, features, and processes, including the algorithm design, inputs, and compilation. Moreover, the number of instructions can be accurately obtained. For hard real-time systems, the worst-case number of instructions executed is needed to compute the WCET. The worst-case number of instructions can be calculated by state-of-the-art timing analysis techniques, such as the implicit path enumeration technique (IPET) [9], although the worst-case inputs may not be known without exhaustive measurements, which are prohibitive.

III. STANDARD DEVIATION OF CPI

We propose to use the standard deviation of CPI to quantitatively evaluate architectural time-predictability. The averaged CPI has been used by the computer architecture community as a key measure of microarchitecture effectiveness. However, as an aggregate metric, it cannot indicate the variation of clock cycles for different instructions. As mentioned above, different instructions usually vary in the amount of clock cycles they take on modern processors, which is the fundamental reason why it is so difficult to predict microarchitectural timing during WCET analysis [1]. Therefore, we use the standard deviation of CPI, which can clearly indicate the amount of variation of microarchitectural timing, to measure architectural time-predictability. In the ideal scenarion, there is 100% time-predictability of the architecture (i.e., the standard deviation of CPI is 0), and it becomes trivial to compute the WCET. When we apply Eq. (1) to this ideal case, both CPI and τ are known, and the worst-case number of instructions is independent of hardware design and can be calculated by using today's timing analysis techniques,

Table 1. Real-time benchmarks used in our experiments

Name	Description	Multi-path	Input
fibcall	Simple iterative Fibonacci calculation	No	Single
sqrt	Square root function	No	Single
bsort100	Bubble sort program	Yes	Three
insertsort	Insertion sort	Yes	Three
qsort-exam	Non-recursive quick sort	Yes	Three
select	Select the *N*-th largest number of an array	Yes	Three

Table 2. Basic configuration of the simulated processor

Parameter	Value
Pipeline	2-IFQ, 32-RUU, 32-LSQ Fetch speed, 2; decode width, 8; issue width, 8; commit width, 8
L1 I-cache	16K bytes, direct-map, 32 bytes block, 1 cycle latency
L1 D-cache	16K bytes, 4-associativity, 32 bytes block, 1 cycle latency
Unified L2 cache	256K bytes, 4-associativity, 64 bytes block, 6 cycles latency
Memory	Unlimited, 100 cycles latency

such as IPET [9].

In modern processors, different types of instructions usually have different latencies. For example, an integer addition instruction is usually faster than a load instruction. Thus the standard deviation of CPI needs to be classified into the standard deviation of CPI for each type of instruction, rather than the standard deviation of CPI for all instructions. For instance, in a time-predictable processor, it is desirable for all the loads to take the same number of clock cycles. However, it would be unwise or unnecessary to require all loads to take the same amount of clock cycles as additions and subtractions. Thus in our evaluation, we classify the standard deviation for three types of instructions, including loads, stores, and other regular instructions.

IV. EVALUATION METHODOLOGY

We simulate a superscalar processor with different architectural features to evaluate architectural time-predictability quantitatively based on the standard deviation of CPI, by using SimpleScalar, a cycle-accurate simulator [10]. We select six real-time benchmarks (Table 1) from the Malardalen WCET benchmark suite [11] for our experiments. Two of them are single-pathed, the other four have multiple paths, and each of them is executed with various inputs. We select three representative inputs to represent the best case, the normal case, and the worst case behavior observed (though it may not be the theoretical best/worst case).

The basic configuration of the simulated processors is listed in Table 2, including the parameters of the pipelines, the caches, and the memory.

A. Architectures Evaluated

Caches and pipelines are two important architectural features that can boost performance and are widely employed in modern processors. Due to space limitations, our experiments in this paper focus on evaluating architectural time-predictability of caches and pipelines. More specifically, we quantitatively compare the architectural time-predictability of four different architectures with features as those given in Table 2.

- With caches, with pipelines: a processor with L1 instruction cache, L1 data cache, L2 cache, and pipelines.
- With caches, without pipelines: a processor with L1 instruction cache, L1 data cache, and L2 cache, but no pipeline.
- Without caches, with pipelines: a processor with no cache and with pipelines.
- Without caches, without pipelines: a processor with no cache and no pipeline.

V. EXPERIMENTAL RESULTS

A. Variation of CPI

Our first experiment quantifies the variation of CPI for

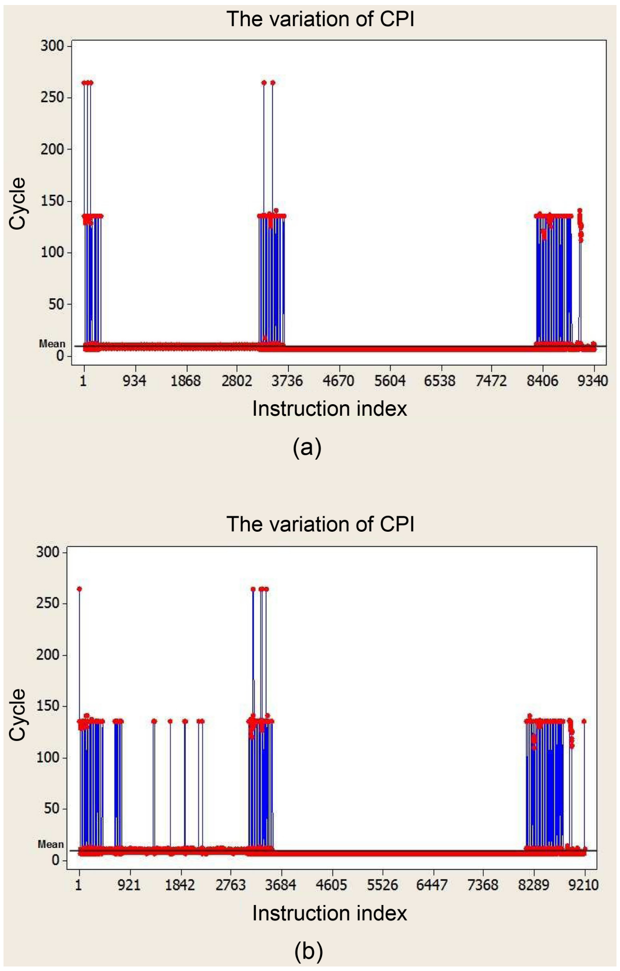


Fig. 1. The variation of cycles per instruction (CPI) of all the instructions for the benchmarks (a) *insertsort* and (b) *qsort-exam* both with the worst-case input.

different benchmarks on a regular processor with caches and pipelines. Fig. 1 shows the variation of CPI of all the instructions for the benchmarks *insertsort* and *qsort-exam*, both with the worst-case inputs. In the rest of the paper, all the experimental results are based on the worst-case inputs, whose timing behaviors are particularly important for hard real-time systems. The black line in each figure shows the mean CPI of all instructions of the benchmark. As we can see for both benchmarks, the CPIs vary significantly, indicating bad time-predictability of the default architecture (i.e., with caches and pipelines).

B. Time-Predictability Results

Our second experiment uses the standard deviation of CPI to quantitatively study the effects of caches on the architectural time-predictability of the processors with pipelines, and the results are given in Table 3. As mentioned above, we classify the CPI for different types of instructions, including loads, stores, and other regular instructions. “All” in Table 3 basically represents all the instructions, regardless of their types. As we can see in Table 3, the averaged standard deviation of CPI for load instructions, store instructions, and other regular instructions are 41.16, 16.58, and 29.79, respectively, with caches, while it decreases to 0, 0.02, and 0.4, respectively without caches. This quantitatively demonstrates the time unpredictability caused by cache memories due to the latencies of memory accesses depending on whether or not there is a hit in the caches and which cache is hit. By disabling all caches, the load instructions become perfectly predictable (i.e., the standard deviation of CPI is 0), as there is neither a cache hit nor miss. The store instructions and other regular instructions also become much more time-predictable without caches. However, their standard deviation of CPI is not exactly 0. This is due to the time variation caused by the pipelines, including delays due to control and data hazards as well as various queuing delays. For example, an instruction may need to stay in the instruction window for longer time because it has to wait for the preceding instruction to commit.

It should be noted that the standard deviation of CPI of all instructions does not reveal architectural time-predictability accurately as mentioned earlier. As can be seen in Table 3, the averaged standard deviation of CPI of all instructions with caches is actually smaller than that without caches. This is because the majority of memory accesses with caches result in cache hits, making the latencies of most loads/stores close to those of other regular instructions. This leads to lower standard deviations when compared to that without caches, in which case the loads/stores are guaranteed to miss and thus have longer latencies than other regular instructions, resulting in a higher standard deviation.

Table 4 compares the mean and the standard deviation of CPI for non-pipelined processors with and without caches. A similar trend can be observed where disabling caches leads to better time-predictability. Moreover, we observe that the standard deviations of CPI of both stores and other regular instructions become 0 without pipelines, indicating that disabling pipelines can further improve time-predictability.

Although both caches and pipelines are detrimental to time-predictability, due to the quantitative nature of our evaluation, we observe that caches can compromise the architectural time-predictability much more than pipelines. This is due to a higher variation of the standard deviation of CPI for each type of instructions between the architectures with caches and without caches when compared to that between the architectures with pipelines and without pipelines (Tables 3 and 4). This may also quantitatively explain why the timing analysis of caches is generally more complex and harder to perform than that of pipelines [1].

Table 3. The mean and standard deviation of cycles per instruction for all benchmarks on architectures with the pipelines, with and without caches

Benchmark	all		ld		st		reg	
	Mean	SD	Mean	SD	Mean	SD	Mean	SD
With the caches (with the pipelines)								
fibcall	12.36	27.27	27.55	49.37	7.57	13.57	15.09	31.54
sqrt	11.82	25.77	21.71	42.12	8.05	15.75	12.93	27.28
bsort100-best	13.01	27.18	19.26	37.47	7.91	14.67	14.61	29.38
bsort100-normal	25.24	39.55	37.24	46.62	13.07	26.42	28.79	41.79
bsort100-worst	25.24	39.55	37.23	46.62	13.07	26.42	28.78	41.78
insertsort-best	12.69	28.05	34.73	55.67	7.691	14.31	15.52	32.30
insertsort-normal	12.11	26.15	24.32	45.37	7.72	14.13	14.08	29.07
insertsort-worst	11.41	23.78	18.49	37.13	7.75	13.88	12.63	25.41
qsort-exam-best	12.01	25.39	20.86	40.29	7.72	13.88	13.45	27.36
qsort-exam-normal	11.98	25.16	20.09	39.04	7.72	13.82	13.35	27.00
hline *qsort-exam-worst*	13.09	24.73	13.63	26.63	11.17	19.89	13.13	24.50
select-best	11.53	24.07	17.67	35.01	7.99	15.04	12.64	25.71
select-normal	11.61	24.30	18.05	35.61	7.99	15.06	12.78	26.05
select-worst	12.03	25.56	20.5	39.25	8.03	15.36	13.59	27.96
Average	14.01	27.61	23.67	41.16	8.82	16.58	15.81	29.79
Without the caches (with the pipelines)								
fibcall	113.60	27.98	205	0	105	0.02	105.07	0.26
sqrt	116.06	31.31	205	0	105	0.02	105.08	0.27
bsort100-best	120.76	36.27	205	0	105	0.02	105.25	0.43
bsort100-normal	123.12	38.29	205	0	105	0.02	105.35	0.66
bsort100-worst	123.12	38.30	205	0	105	0.02	105.35	0.66
insertsort-best	111.34	24.33	205	0	105	0.02	105.05	0.22
insertsort-normal	113.88	28.38	205	0	105	0.02	105.09	0.28
insertsort-worst	116.46	31.76	205	0	105	0.02	105.12	0.33
qsort-exam-best	116.93	32.31	205	0	105	0.02	105.15	0.41
qsort-exam-normal	117.48	32.94	205	0	105	0.02	105.16	0.42
qsort-exam-worst	131.81	44.09	205	0	105	0	105.30	0.54
select-best	117.99	33.51	205	0	105	0.02	105.16	0.40
select-normal	117.74	33.23	205	0	105	0.02	105.15	0.40
select-worst	116.17	31.40	205	0	105	0.02	105.13	0.37
Average	118.32	33.15	205	0	105	0.02	105.17	0.40

C. Performance Results

Although architectural time-predictability is degraded by using caches and pipelines, the average-case performance improves with the use of both. As shown in Table 5, the performance of all the benchmarks is best on the architecture with both caches and pipelines and is worst on the architecture without both caches and pipelines. The former is high-performance but not time-predictable, whereas the latter is time-predictable but not high-performance. Among these four architectures, the one with pipelines but without caches seems to provide high time-predictability with adequate performance, better than that without both caches and pipelines. By com-

Table 4. The mean and standard deviation of cycles per instruction for all benchmarks on architectures without the pipelines, with and without caches

Benchmark	all		ld		st		reg	
	Mean	SD	Mean	SD	Mean	SD	Mean	SD
With the caches (without the pipelines)								
fibcall	14.15	30.31	22.33	44.65	15.44	31.46	10.84	24.11
sqrt	13.61	29.39	18.38	38.12	15.82	32.39	10.45	23.14
bsort100-best	11.27	24.28	11.31	24.34	15.4	31.38	8.58	17.75
bsort100-normal	11.67	25.24	12.83	27.99	15.22	31.07	8.82	18.54
bsort100-worst	11.27	24.28	11.31	24.34	15.4	31.39	8.58	17.75
insertsort-best	14.58	31.12	27.29	50.57	15.82	32.11	11.15	24.83
insertsort-normal	13.46	29.06	19.38	40.7	15.6	31.73	10.21	22.54
insertsort-worst	12.2	26.52	14.93	33.1	15.29	31.2	9.18	19.66
qsort-exam-best	13.09	28.35	17.29	36.89	15.42	31.42	10.09	22.2
qsort-exam-normal	12.95	28.05	16.69	35.8	15.35	31.29	10.05	22.09
qsort-exam-worst	11.11	24.47	12.72	27.19	12.56	28.42	10.26	22.63
select-best	12.57	27.21	14.53	31.23	15.43	31.58	9.85	21.61
select-normal	12.69	27.47	14.83	31.78	15.46	31.64	9.97	21.93
select-worst	13.39	28.85	16.62	35.03	15.67	31.99	10.6	23.56
Average	12.71	27.47	16.46	34.41	15.28	31.36	9.9	21.59
Without the caches (without the pipelines)								
fibcall	163.98	49.19	205	0	205	0	105	0
sqrt	158.52	49.88	205	0	205	0	105	0
bsort100-best	155.13	50	205	0	205	0	105	0
bsort100-normal	155.35	50	205	0	205	0	105	0
bsort100-worst	155.12	50	205	0	205	0	105	0
insertsort-best	162.89	49.38	205	0	205	0	105	0
insertsort-normal	159	49.84	205	0	205	0	105	0
insertsort-worst	155.06	50	205	0	205	0	105	0
qsort-exam-best	157.31	49.95	205	0	205	0	105	0
qsort-exam-normal	156.58	49.98	205	0	205	0	105	0
qsort-exam-worst	157.29	49.95	205	0	205	0	105	0
select-best	155.78	50	205	0	205	0	105	0
select-normal	156.04	49.99	205	0	205	0	105	0
select-worst	157.87	49.92	205	0	205	0	105	0
Average	157.57	49.86	205	0	105	0	105	0

parison, the architecture with caches but without pipelines can generally provide good performance, but time-predictability is inadequate. Moreover, architectural innovations are needed to keep high-performance features that do not affect time-predictability and to design new features that are time-predictable and can also boost performance.

VI. RELATED WORK

Recently, the real-time and embedded computing community has shown interest in defining a metric of time-predictability due to the importance of studying time-predictability quantitatively. Thiele and Wilhelm [2] defined time-predictability as a pessimistic WCET analysis and

Table 5. The total execution cycles of all the benchmarks on all the four architectures studied

Benchmark	With the pipeline		Without the pipeline	
	With the cache	Without the cache	With the cache	Without the cache
fibcall	27680	692501	97952	1135420
sqrt	35055	846101	115142	1341100
bsort100-best	31809	1059701	123627	1646080
bsort100-normal	35649	1142501	128719	1772220
bsort100-worst	35657	1142901	128749	1772740
insertsort-best	27877	658901	96048	1073140
insertsort-normal	29132	771901	103852	1227190
insertsort-worst	29831	934701	114002	1449230
qsort-exam-best	34356	885201	115936	1392355
qsort-exam-normal	35649	921701	119337	1395045
qsort-exam-worst	36617	977301	130802	1443080
select-best	34790	846701	113333	1336530
select-normal	34850	959501	119225	1465870
select-worst	35371	955801	120122	1488785

best-case execution time (BCET) analysis, and Grund [12] defined time-predictability as the relation between BCET and WCET and argued that it should be an inherent system property. Grund et al. [13] then proposed a template for predictability definitions and refined the definition into state-induced time-predictability (SIP) and input-induced time-predictability. Kirner and Puschner [14] formalized a universal definition of time-predictability by combining WCET analyzeability and stability of the system. However, in all the above work, except for that of Grund [12] and Grund et al. [13], the calculation for time-predictability is still dependent on the computation of WCET. Since WCET estimation is usually pessimistic and there is no standard way to compute it (though different methods to derive WCET, such as abstract interpretation and static cache simulation do exist), any time-predictability metric relying on WCET analysis is likely to be inaccurate and would be difficult to be standardized in practice.

Moreover, in all the above work, except again for that of Grund [12] and Grund et al. [13], the definition for time-predictability does not separate the time variation caused by software and hardware, making it overly complicated to derive a time-predictable metric that can effectively guide the architectural design. While Grund [12] and Grund et al. [13] proposed SIP to separate the timing uncertainty between hardware and software, the metric they proposed to evaluate SIP needs to exhaustively find the maximum and minimum execution time of all different states, and this may not be computationally feasible. Also, no quantitative results are given in their study. In contrast, this paper proposes a metric to efficiently assess architectural time-predictability with a quantitative evaluation of architectural time-predictability on statically-scheduled processors with different architectural features.

Recently, Ding and Zhang [15] proposed to use a quantitative architectural time-predictability factor (ATF). Compared to ATF, which can only quantitatively show the ATP of the whole processor, the standard deviation of CPI can provide the ATP for various microarchitectural components, such as caches and pipelines. Thus, we can quantitatively understand the impact of different microarchitectural components on ATP by using the standard deviation of CPI for different types of instructions, a feature which is not available in ATF. Therefore, the standard deviation of CPI can provide useful insights for engineers to design and implement new architectures with better time-predictability.

VII. CONCLUSION

In this paper, we present the concept of architectural time-predictability to separate that of hardware design from that of software design. We then propose a new metric to evaluate the architectural time-predictability that uses the standard deviation of the clock CPI. Our preliminary evaluation demonstrates that the proposed metric can quantitatively measure time-predictability of processors with different architectural features, such as caches and pipelines.

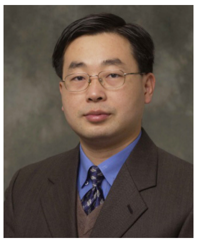
The standard deviation of CPI can be used to quantitatively guide the design of time-predictable processors. In our future work, we plan to use the standard deviation of CPI to evaluate time-predictability of different architectures, for example branch prediction, multicore designs, etc. Moreover, we intend to use this new metric to quantitatively trade off time-predictability with performance, energy, and cost for real-time systems.

ACKNOWLEDGMENTS

This work was funded in part by the United States National Science Foundation (NSF) grant (No. CCF 1063645).

REFERENCES

1. R. Wilhelm, J. Engblom, A. Ermedahl, N. Holsti, S. Thesing, D. Whalley, … P. Stenstrom, “The worst-case execution time problem: overview of methods and survey of tools,” *ACM Transactions on Embedded Computing Systems*, vol. 7, no. 3, article no. 36, 2008.
2. L. Thiele and R. Wilhelm, “Design for time-predictability,” in *Design of Systems with Predictable Behaviour*, Dagstuhl, Germany: Internationales Begegnungs- und Forschungszentrum für Informatik, 2004.
3. M. Colnaric and W. A. Halang, “Architectural support for predictability in hard real time systems,” *Control Engineering Practice*, vol. 1, no. 1, pp. 51-57, 1993.
4. M. Delvai, W. Huber, P. Puschner, and A. Steininger, “Processor support for temporal predictability: the SPEAR design example,” in *Proceedings of the 15th Euromicro Conference on Real-Time Systems*, Porto, Portugal, 2003, pp. 169-176.
5. S. A. Edwards and E. A. Lee, “The case for the precision timed (PRET) machine,” in *Proceedings of the 44th annual Design Automation Conference*, San Diego, CA, 2007, pp. 264-265.
6. N. Yamasaki, I. Magaki, and T. Itou, “Prioritized SMT architecture with IPC control method for real-time processing,” in *Proceedings of the 13th IEEE Real Time and Embedded Technology and Applications Symposium*, Bellevue, WA, 2007, pp. 12-21.
7. M. Paolieri, E. Quinones, F. J. Cazorla, G. Bernat, and M. Valero, “Hardware support for WCET analysis of hard real-time multicore systems,” in *Proceedings of the 36th Annual International Symposium on Computer Architecture*, Austin, TX, 2009, pp. 57-68.
8. M. Schoeberl, “Time-predictable computer architecture,” *EURASIP Journal on Embedded Systems*, vol. 2009, article no. 2, 2009.
9. Y. T. S. Li and S. Malik, “Performance analysis of embedded software using implicit path enumeration,” in *Proceedings of the ACM SIGPLAN Workshop on Languages, Compilers, and Tools for Real-Time Systems*, Montreal, Canada, 1995, pp. 88-98.
10. SimpleScalar, http://www.simplescalar.com.
11. J. Gustafsson, A. Betts, A. Ermedahl, and B. Lisper, “The Malardalen WCET benchmarks: past, present and future,” in *Proceedings of the 10th International Workshop on Worst-Case Execution Time Analysis*, Brussels, Belgium, 2010, pp. 136-146.
12. D. Grund, “Towards a formal definition of timing predictability,” in *Proceedings of Workshop on Reconciling Performance with Predictability*, Grenoble, France, 2009.
13. D. Grund, J. Reineke, and R. Wilhelm, “A template for predictability definitions with supporting evidence,” in *Bringing Theory to Practice: Predictability and Performance in Embedded Systems*, Wadern, Germany: Schloss Dagstuhl - Leibniz-Zentrum für Informatik, pp. 22-31, 2011.
14. R. Kirner and P. Puschner, “Time-predictable computing,” in *Proceedings of the 8th IFIP WG 10.2 International Workshop on Software Technologies for Embedded and Ubiquitous Systems*, Waidhofen/Ybbs, Austria, 2010, pp. 23-24.
15. Y. Ding and W. Zhang, “Architectural time-predictability factor (ATF): a metric to evaluate time predictability of processors,” *ACM SIGBED Review*, vol. 9, no. 4, pp. 6-15, 2012.



Wei Zhang

Dr. Wei Zhang is a tenured associate professor in Electrical and Computer Engineering at Virginia Commonwealth University. Dr. Wei Zhang received his Ph.D. from Pennsylvania State University in 2003. From August 2003 to July 2010, Dr. Zhang worked as an assistant professor and then as a tenured-associate professor at Southern Illinois University Carbondale. His research interests are in embedded and real-time computing systems, computer architectures, compilers, and low-power systems. Dr. Zhang has received the 2009 SIUC Excellence through Commitment Outstanding Scholar Award for the College of Engineering, and the 2007 IBM Real-time Innovation Award. Dr. Zhang has received 5 research grants from the National Science Foundation. In addition, his research and educational efforts have been supported by industry, including leading IT companies such as IBM, Intel, Motorola, and Altera. Dr. Zhang has published more than 100 papers in refereed journals and conference proceedings. He is a senior member of IEEE, and an associate editor of the Journal of Computing Science and Engineering. He has served as a member of the organizing and program committees for several IEEE/ACM international conferences and workshops.

Yiqiang Ding

Yiqiang Ding is currently a Ph.D. student in Electrical and Computer Engineering at Virginia Commonwealth University. He received the B.S. degree of computer science in 2002 and the M.S. degree of computer engineering in 2005 from the Beijing University of Posts and Telecommunications in China. He worked in the Motorola China Design Center as a system engineer from 2005 to 2007. His research interests are in embedded and real-time computing systems, computer architectures, and compilers.