JCSE Journal of Computing Science and Engineering

Regular Paper

Journal of Computing Science and Engineering,
Vol. 7, No. 2, June 2013, pp. 89-98

Overview of Real-Time Java Computing

Yu Sun
Department of Electrical and Computer Engineering, Southern Illinois University Carbondale, Carbondale, IL, USA
sunyu@engr.siu.edu

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

Abstract
This paper presents a complete survey of recent techniques that are applied in the field of real-time Java computing. It focuses on the issues that are especially important for hard real-time applications, which include time predictable garbage collection, worst-case execution time analysis of Java programs, real-time Java threads scheduling and compiler techniques designed for real-time purpose. It also evaluates experimental frameworks that can be used for researching real-time Java. This overview is expected to help researchers understand the state-of-the-art and advance the research in real-time Java computing.

Category: Embedded computing

Keywords: Performance; Reliability; Java; WCET analysis; Compiler; Hard real-time systems

I. INTRODUCTION

Real-time systems ranging from aircraft and nuclear power plant controllers to video games and speech recognition have become an increasingly important part of our society. The small stand-alone real-time applications of the past are giving way to a new type of networked real-time systems that are running on heterogeneous computing and networking environments; for example, integrated battleship management, VoIP-based telecommunications, stock arbitrage, and automotive systems [1]. The software development for real-time systems has also become more and more complex, due to the increase in the size, longevity, and required features of these systems. Following its successful use in Web applications, middleware, and enterprise software development, Java can offer significant advantages such as productivity, reliability, and portability for developing large and complex real-time software, and thus becomes an attractive option for real-time software design. A study by Nortel Networks indicated that using real-time Java doubled their productivity in developing telecom equipment and base stations [2].

For real-time systems, especially hard real-time and safety-critical systems, it is crucial to know the worst-case execution time (WCET) [3] to ensure that each task can meet its deadline. WCET should be computed based on static timing analysis, rather than measurement alone, because in general it is impossible to exhaustively measure all program paths in order to locate the longest execution time. The WCET of a real-time task however, is heavily dependent on the programming language used to implement this task, as well as the target processor and

Open Access http://dx.doi.org/10.5626/JCSE.2013.7.2.89 http://jcse.kiise.org

Received 21 April 2013, **Accepted** 12 May 2013
*Corresponding Author

 pISSN: 1976-4677 eISSN: 2093-8020

the compiler optimizations. Java, designed to run its byte code on Java Virtual Machine (JVM) for portability across different computing platforms, is a serious challenge for predictable execution and WCET analysis. Without solving this problem, Java cannot be safely used in hard real-time and safety-critical systems. Even for firm or soft real-time systems, unpredictable and varied execution time of Java computing may significantly compromise the quality of service.

Traditional JVM designs have mainly focused on improving the average-case performance, which may result in too much unpredictability with the use of multiple threads, dynamic compilation, automatic garbage collection, and so on. Since the first discussion of issues in design and implementation of real-time Java [4] was proposed in the 1990s, a considerable amount of remarkable research work has been done to adapt Java to real-time purpose. The Real-Time for Java Experts Group began to develop Real-Time Specification for Java (RTSJ) in March 1999 [5]. The RTSJ has been evaluated for uses in avionics and space systems by Boeing and Jet Propulsion Laboratory (JPL) [6, 7]. There are also a number of commercial implementations such as Sun Microsystems Mackinac, IBM WebSphere Real Time, TimeSys jTime, and Aicas JamaicaVM, as well as several open source implementations including OVM and jRate. A more recent effort regarding safety-critical Java software is the Safety-Critical Java (SCJ) Specification [8].

Due to the widespread use of real-time applications and the increasing use of Java in developing real-time software with hard deadlines, it becomes important for real-time researchers and developers to understand how to achieve time predictability in Java based computing. This paper presents an overview of the most recent and important studies in the area of real-time Java computing. More specifically, this paper will present related studies on real-time Java in the following topics:

- Real-time threads and scheduling
- Time predictability of Java code
- Bounded garbage collection
- Suitable compilation and optimization

The rest of this paper is organized by these four main issues. Sections II to V present the newest work on one of the above topics. Section VI describes a special way to implement real-time Java by hardware, that is, Java processor. Section VII discusses available tools and frameworks that may be used for real-time Java research and the last section gives the summary and conclusion.

II. REAL-TIME THREADS AND SCHEDULING

The RTSJ [5] by Java Expert Group firstly gives an abstract definition of real-time threads and scheduling. Two kinds of real-time threads are defined in addition to normal Java threads. One is real-time thread and another is no-heap real-time thread. NHRTs are not permitted to access the heap so that it can avoid the possible delay caused by garbage collection. On the other hand, RTs can access heap so that they have more flexibility in coding and are suitable for code with a higher tolerance for longer delays, such as soft real-time application.

The RTSJ also provided minimum requirements for real-time Java scheduling. All implementation of RTSJ must provide a fixed-priority preemptive scheduler with no fewer than 28 unique priorities. RTSJ is open for extension of other scheduling algorithms, and the implementation relies on the support of real-time operating systems.

Similarly to RTSJ definition, the Ravenscar-Java [9] defines its real-time threads and scheduler. Additionally, it derives two specific types of threads from RTSJ: periodic real-time threads and sporadic event handlers. This design is based on the behavior of real-time applications whereby most threads wake up and do their job after fixed time intervals.

III. TIME PREDICTABILITY OF JAVA CODE

It is necessary for real-time Java applications to know the WCET of Java programs. Since Java code is firstly compiled into Java byte code (JBC) and then executed on JVMs, it is quite natural to analyze the WCET of a Java program in two steps: JBC level and lower platform-dependent level.

A. WCET Analysis at JBC Level

There have been many research efforts to conduct WCET analysis for C programs. Lundqvist and Stenstrom [10] discovered timing anomalies in out-of-order superscalar processors. Li and Malik [11] and Li et al. [12] proposed the implicit path enumeration technique (IPET) to compute the worst-case path for deriving the WCET accurately. Recently, the timing analysis has been extended from single-core processors [13-19] to multicore processors [20-28]. A good summary of contributions in the area of WCET analysis can be found in [3].

Java itself is very friendly to JBC level WCET static analysis with its well-formed object-oriented structure and features. JBC stored in Java class files is easily read and analyzed by the WCET analyzer, which generates control flow graph (CFG) and basic block (BB) of given Java programs. With Java annotations which are supported by most modern Java compilers, additional information such as loop bounds can also be provided.

A series of works have been done in this area. Puschner and Bernat [29] described a general method that can be applied for the Java program using the integer linear pro-

gramming (ILP) technique. Bernat et al. [30] provided an implementation of the WCET analyzer based on Java annotations. Bate et al. [31] modified the Kaffe [32] and Komodo [33] to support WCET of Java applications running on these two JVMs. Control flow and data flow are both considered in these studies. The works in [34, 35] designed an extension to bring loop bounds, timing modes, and dynamic dispatch semantics into the WCET analyzer. Harmon and Klefstad [36] then attempted to construct a standard of Java annotations for WCET analysis, based on all previous works. Hepp and Schoeberl [37] explored WCET-based optimizations for Java programs.

B. Platform-Dependent WCET Analysis for Java

The only gap between JBC WCET and reality is the low level, platform-dependent timing model of JVMs. The exact execution time of each JBC and combination of JBCs is needed to compute the final WCET of any Java program. Hu et al. [38] attempted to solve this problem by two methods: profiling based and benchmark based. The profiling method inserts instrument codes into the Java program and collects execution times of JBCs. As another way, by running specially designed benchmarks, the same job can be done on all kinds of platforms without changing the instrument codes. Bate et al. [39] studied the JBC execution overhead due to the JVMs and processor pipelines, as well as the effects on the WCET of each JBC. However, these works are all measurement based. There is still no way to statically analyze the low level WCET for a particular JVM and platform.

IV. REAL-TIME JAVA MEMORY MANAGEMENT

Java's automatic memory management, garbage collection (GC), is a very good feature and brings great benefit to software development. However, most current garbage collectors are not time predictable. As a result, GC actually prevents Java programs from being adapted in the real-time area. It is totally unacceptable that a real-time task is interrupted by GC thread and does not know when it can be up and running again.

A. Bypassing Garbage Collection

The first idea is to remove the unpredictable GC from the real-time Java system. That is why the scoped memory and immortal memory section are defined in RTSJ [5]. In this case, the objects in real-time threads are managed by programmers instead of JVMs. The developers take care of the memory areas that are used for real-time tasks and leave the other part to GC. The GC thread holds lower priority than real-time threads and thus cannot interfere with them. Time predictability is guaranteed but the flexibility is lost in development. Most of the implementations of RTSJ support this kind of technique bypassing GC. Besides, researchers made an effort to improve the efficiency of this scoped memory. Beebee and Rinard [40] implemented the scoped memory model and evaluated its efficiency. Corsaro and Schmidt [41] presented another implementation of real-time JRate. Pizlo et al. [42] presented an informal introduction to the semantics of the scoped management rules of the RTSJ, and it also provided a group of design patterns which can be applied to achieve a higher efficiency of scoped memory. Andreae [43] provided a complete programming model of the scoped memory model to simplify the memory analysis that has to be done by developers. Bypassing GC is proved to be easy to apply to existing JVMs and brings small overhead to runtime memory management. However, it is limited because of the great difficulty it brought to the developers. Without the GC feature, programmers have to pay much more attention to the objects in real-time tasks to avoid memory error. Although scoped memory may be a good solution of small short-term projects, real-time GC is definitely necessary for future real-time Java systems.

B. Time-Predictable Garbage Collection

Baker's incremental copying collector [44] is the first idea of real-time GC. In this work, the memory mutator operation leads to the GC operation. Hence, GC is predictable and the worst case is that every read or allocation invokes a certain amount of collection operations. It is not very efficient yet provides a way to implement real-time GC. This so called work-based GC is then developed by many other following researchers, such as [45] and [46] in JamaicaVM. The basic idea is kept while overhead of allocation detection and unnecessary collection is significantly reduced. Even hardware can be used to assist GC efficiency and reduce WCET bounds, as presented in [47].

On the other hand, Bacon et al. [48] developed a time-based approach that invokes a collector at regular intervals. This kind of GC requires read/write barriers in order to maintain consistency, and the memory allocation must be time predictable. The quota of CPU time for GC can be dynamically computed and set at runtime based on the memory usage of real-time threads. This technique is then applied in IBM's real-time JVM [49] named as Metronome [50, 51]. Henriksson [52] also proposed a time-based GC strategy which set the collector to active only when the processor is idle. The idea then is improved in [53]. The key issue of time-based real-time GC is how much time should be given to the collector. Schoeberl and Vitek [54] presented the algorithm to compute the GC quota and interval in their study. Cho et al. [55] used

statistical tools to guarantee the algorithm's effectiveness and superiority.

C. Concurrent Garbage Collection

Both time-based and work-based real-time GC have the same problem, that is, the processor is fully occupied by the collector thread at the GC stage. Thus thread switching has to be performed when a real-time task is presented. Fortunately, the improvement of multiprocessor technique brings a new approach that can avoid this problem. The concurrent GC runs on a different processor other than real-time tasks. As a result, the overhead of switching threads is reduced to a minimum. However, there are several serious challenges: synchronization between processors, access lock to memory objects, and so on.

The first multiprocessor concurrent GC implementation was presented by [56], which applied the algorithm in [57] on a 64-processor machine. It greatly reduced the pause time of GC to the millisecond level. After that, various kinds of concurrent GC algorithms and implementations were presented by researchers to further improve it. Xian and Xiong [58] showed that their technique can effectively reduce the memory amount used by concurrent real-time GC, which is relatively huge. Sapphire [59] implemented a copying collector for Java with low overhead and short pause time. Pizlo et al. [60] compared three lock-free concurrent GC algorithms: STOPLESS [61], CHICKEN, and CLOVER, which have the pause time as microsecond level. Although these algorithms are designed for and implemented by C#, it is easy to adapt them to Java since they are similar VM based language.

V. COMPILATION FOR REAL-TIME JAVA

Compilation and optimization are also critical issues of real-time Java. Just-in-time (JIT) compilation is already widely used in modern JVMs because of the performance boost it provides. Speculative optimizations are also an essential part now in many advanced JVMs. However, the JIT compiler and some optimizations bring unpredictability into Java applications, so that they cannot be simply used in real-time Java systems, where an old interpreter and ahead-of-time (AOT) compiler take their chances.

A. Interpretation

Interpretation is the most original way to execute JBCs. It reads the JBCs from classes and translates them into native code, and then executes them. It is slow but has excellent time predictability as the interpretation of each JBC can be easily measured. For this reason, most real-time JVMs, such as IBM WebSphere VM [49], Sun Java Real-Time System [62], and Open VM [63], provide the interpreter mode. However, the performance limits this mode to be used in modern real-time systems.

B. JIT Compilation

To improve interpreter performance, JVMs use the JIT compiler to compile the code sequence into native code on the fly before it executes. However, the JIT-only strategy introduces compilation overhead at runtime. To address this problem, a JVM can focus on optimizing only "hot-spots", while the rest of the code can be either interpreted or compiled by a basic compiler without optimization. Examples of adaptive optimization systems include HotSpot virtual machine [64] from Sun (now Oracle), Jikes RVM [65] from IBM, and Open Runtime Platform (ORP) [66] from Intel Corporation. To identify hot-spots, researchers have proposed to use online hardware profiling mechanisms such as counters and samplings [67-71], or to use program instrumentation [72-78], combined instrumentation and sampling [79-81], or coupled offline and online profiling [82]. To further improve adaptive optimization, a number of techniques have been developed; for example, recompilation [83], deferred and partial compilation [84-86], and dynamic deoptimization [87]. The idea of dynamic and adaptive compilation has also been extended and studied in other contexts, among them hybrid JIT compilation [88], trace-based parallelization [89-93], and adaptive garbage collection [94].

While the JIT compiler is useful for improving the average-case performance of non-real-time Java applications, for real-time systems, the JIT compiler has two main drawbacks. One is that it interrupts other threads from execution and the time it takes is unpredictable. Secondly, the speculative optimizations are not suitable for real-time applications. [95] compared AOT and JIT compilers and [96] introduced their implementation and evaluation in IBM WebSphere VM. However, the JIT compiler is still capable of soft real-time systems, as reported in [1]. A carefully managed priority is necessary and some optimizations must be turned off. In addition, Sun and Zhang [97, 98] explored multicore processors to improve time predictability of dynamic compilation.

C. AOT Compilation

The AOT compiler does most of the work before execution, with only the dynamic part left to runtime. The AOT compilation can provide higher performance than interpretation, while keeping good time predictability. However, the part that the AOT compiler cannot complete before execution is very crucial to the performance. For example, the AOT compiler has no idea of class references nor dynamically generated classes. It has to use a resolution thread to patch such information during runtime. Furthermore, none of the aggressive inlining can be performed by the AOT compiler due to lack of class references, which loses a great chance to improve perfor-

mance. In any case, the AOT compiler is still the best choice for hard real-time systems, and is included in most modern real-time JVMs [49, 62, 63].

VI. JAVA PROCESSORS FOR REAL-TIME COMPUTING

Beside all the above software approaches, hardware implementation of JVM which is called Java processor is also presented as a solution of real-time systems. Basically, a Java processor is a stack based processor and executes JBC directly. Method cache and stack cache take the places of instruction cache and data cache, separately, inside of Java processors. It is possible that Java processors are designed to be deterministic in terms of execution time. Komodo [33] is an early implementation of Java processors that provide the main Java features and support real-time tasks. [99] continued working on Komodo processor with advanced scheduling and event-handling algorithms. SHAP [100] is another Java processor that is designed specifically for real-time systems. It implements fast context switching and concurrent GC. JOP [101, 102] is a well-developed Java processor which is WCET analyzable. Method cache in JOP simplifies the analysis of WCET in control flow, because only thread switching can introduce cache misses. Tools for performing WCET analysis on JOP is provided in [103]. The high level WCET analysis is based on ILP and a low level timing model is provided by JOP properties. Similar works have also been done in [104, 105]. Besides, Harmon and Klefstad [106] adapted their work of WCET annotation to Java processors, and made it interactive to developers in order to offer various feedbacks.

VII. EXPERIMENTAL FRAMEWORKS

The research resources of real-time Java are quite limited. Only a few real-time JVMs are completed and fewer of them are under an open-source license.

IBM [49] and Sun Microsystem (now Oracle) [62] are two main companies who provide well-developed commercial real-time Java products. Evaluation or academic version of their real-time JVM can be obtained from the Internet. However, the source code is unavailable.

OVM [63] is a good choice. It is open-sourced, supports most of RTSJ's features, and is still an active research project [6, 107] with some documents. It has been tested in our lab and appears to work well. There are two main problems with OVM: first, its JIT compiler is quite simple and incomplete (lack of dynamic class loading); second, OVM is obviously designed for a single processor. Extra work is needed if we want to do some research on multicore systems.

An alternative may be jRate [108], whose source code is also available. jRate is an extension to the GNU GCJ compiler and a group of runtime libraries. It implemented most features needed by RTSJ.

Another candidate is Jikes RVM [65]. Although Jikes RVM is not designed for real time purpose, it is the most completed open-source JVM. Actually the prototype of the Metronome garbage collector is implemented on Jikes RVM. So it is possible to develop real-time extension for Jikes RVM.

A Java library named Javolution [109] may be helpful. It is an extended library on Sun Java implementing RTSJ. It is open-sourced.

Aside from these software solutions, the JOP [101] Java processor is also open-sourced, in VHDL format. So it is possible to combine it with some simulators such as SimpleScalar or Trimaran so as to build a multiprocessor system. There is some research work [110] focusing on multiprocessor, and I think this is a very promising research field.

VIII. CONCLUSIONS

Real-time embedded systems have increasingly become integral to our society. Real-time applications range from safety-critical systems such as aircraft and nuclear power plant controllers, to entertainment software such as video games and graphics animation. Recently, there have been growing interests in using Java for a wide variety of both soft- and hard-real-time systems, primarily due to Java's inherent features such as platform independence, scalability and safety. However, to enable real-time Java computing, the computation time of Java applications must be predictable, which is especially important for hard real-time and safety-critical systems.

This paper surveys this relatively new research area, which is expected to help researchers understand the state-of-the-art and to advance the real-time Java computing. In this work, we have reviewed the RTSJ and the WCET analysis of Java applications at both the byte code level and the architectural level. Since garbage collection can disrupt the time predictability, we have surveyed the state-of-the-art solutions of real-time Java GC for both uniprocessors and multiple processors. Due to the importance of JIT compilation on the performance of Java programs, we have also discussed the compiler issues for real-time Java applications. In addition to the software-based solutions to achieve time-predictable Java computing, we have also briefly explained the current work in designing real-time Java processors. To assist new researchers in selecting a suitable real-time Java experimental framework, this paper also listed a number of current open-source and private real-time JVMs, libraries and soft Java processors.

As can be seen in this overview, real-time Java computing is an active and promising research field. There

are many research challenges and opportunities as well. Based on this survey, further investigation is still needed in the following directions:

- Time-predictable dynamic compilation for real-time Java applications
- Real-time Java on multiprocessor (multiple uniprocessor **OR** uniprocessor+Java processor)
- Low level Java WCET analysis with architectural timing information (cache, branch prediction, etc.)
- Multiprocessor/multicore real-time GC algorithms.

REFERENCES

1. J. Auerbach, D. F. Bacon, B. Blainey, P. Cheng, M. Dawson, M. Fulton, D. Grove, D. Hart, and M. Stoodley, "Design and implementation of a comprehensive real-time Java virtual machine," in *Proceedings of the 7th ACM & IEEE International Conference on Embedded Software*, Salzburg, Austria, 2007, pp. 249-258.
2. D. Lammers (2005, Mar 28), "Real-time Java: reliability quest fuels RT Java projects," *Electronic Engineering Times*, http://www.eetimes.com/electronics-news/4052151/REAL-TIME-JAVA-Reliability-quest-fuels-RT-Java-projects.
3. R. Wilhelm, J. Engblom, A. Ermedahl, N. Holsti, S. Thesing, D. B. Whalley, G. Bernat, C. Ferdinand, R. Heckmann, T. Mitra, et al., "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.
4. K. Nilsen, "Issues in the design and implementation of real-time Java," http://www.cs.cornell.edu/courses/cs614/1999sp/papers/rtji.pdf.
5. J. Gosling and G. Bollella, The Real-Time Specification for Java, Boston, MA: Addison-Wesley, 2000.
6. J. Baker, A. Cunei, C. Flack, F. Pizlo, M. Prochazka, J. Vitek, A. Armbruster, E. Pla, and D. Holmes, "A real-time Java virtual machine for avionics: an experience report," in *Proceedings of the 12th IEEE Real-Time and Embedded Technology and Applications Symposium*, San Jose, CA, 2006, pp. 384-396.
7. D. Sharp, "Real-time distributed object computing: ready for mission-critical embedded system applications," in *Proceedings of the 3rd International Symposium on Distributed Objects and Applications*, Rome, Italy, 2001, pp. 3-4.
8. T. Henties, J. Hunt, D. Locke, K. Nilsen, M. Schoeberl, and J. Vitek, "Java for safety-critical applications," in *2nd International Workshop on the Certification of Safety-Critical Software Controlled Systems*, York, UK, 2009.
9. J. Kwon, A. Wellings, and S. King, "Ravenscar-Java: a high integrity profile for real-time Java," in *Proceedings of the 2002 Joint ACM-ISCOPE Conference on Java Grande*, Seattle, WA, 2002, pp. 131-140.
10. T. Lundqvist and P. Stenstrom, "Timing anomalies in dynamically scheduled microprocessors," in *Proceedings of the 20th IEEE Real-Time Systems Symposium*, Phoenix, AZ, 1999, pp. 12-21.
11. Y. 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*, San Francisco, CA, 1995, pp. 456-461.
12. Y. S. Li, S. Malik, and A. Wolfe, "Efficient microarchitecture modeling and path analysis for real-time software," in *Proceedings of the 16th Real-Time Systems Symposium*, Pisa, Italy, 1995, p. 298.
13. R. Arnold, F. Mueller, D. Whalley, and M. Harmon, "Bounding worst-case instruction cache performance," in *Proceedings of the 15th IEEE Real-Time Systems Symposium*, San Juan, Puerto Rico, 1994, pp. 172-181.
14. Y. S. Li, S. Malik, and A. Wolfe, "Cache modeling for real-time software: beyond direct mapped instruction cache," in *Proceedings of the 17th Real-Time Systems Symposium*, Washington, DC, 1996, pp. 254-263.
15. M. Alt, C. Ferdinand, F. Martin, and R. Wilhelm, "Cache behavior prediction by abstract interpretation," *Static Analysis, Lecture Notes in Computer Science vol. 1145*, R. Cousot and D. Schmidt, editors, Heidelberg: Springer, pp. 52-66, 1996.
16. J. Yan and W. Zhang, "WCET analysis of instruction caches with prefetching," in *Proceedings of the ACM SIGPLAN/SIGBED Conference on Languages, Compilers, and Tools for Embedded Systems*, San Diego, CA, 2007, pp. 175-184.
17. B. Lesage, D. Hardy, and I. Puaut, "WCET analysis of multi-level set-associative data caches," in *Proceedings of the 9th International Workshop on Worst-Case Execution Time Analysis*, Dublin, Ireland, 2009.
18. B. Huynh, L, Ju, and A. Roychoudhury, "Scope-aware data cache analysis for WCET estimation," in *Proceedings of the 17th IEEE Real-Time and Embedded Technology and Applications Symposium*, Chicago, IL, 2011, pp. 203-212.
19. X. Li, A. Roychoudhury, and T. Mitra, "Modeling out-of-order processors for software timing analysis," in *Proceedings of the 25th IEEE International Real-Time Systems Symposium,* Lisbon, Portugal, 2004, pp. 92-103.
20. J. Yan and W. Zhang, "WCET analysis for multi-core processors with shared instruction caches," in *Proceedings of 14th IEEE Real-Time and Embedded Technology and Applications Symposium,* St. Louis, MO, 2008, pp. 80-89.
21. W. Zhang and J. Yan, "Accurately estimating worst-case execution time for multi-core processors with shared instruction caches," in *Proceedings of the 15th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*, Beijing, China, 2009, pp. 455-463.
22. Y. Li, V. Suhendra, Y. Liang, T. Mitra, and A. Roychoudhury, "Timing analysis of concurrent programs running on shared cache multi-cores," in *Proceedings of the IEEE Real-time System Symposium,* Washington, DC, 2009, pp. 57-67.
23. W. Zhang and J. Yan, "Static timing analysis of shared caches for multicore processors," *Journal of Computing Science and Engineering*, vol. 6, no. 4, pp. 267-278, 2012.
24. L. Wu and W. Zhang, "A model checking based approach to bounding worst-case execution time for multicore processors," *ACM Transactions on Embedded Computer Systems,* vol. 11, no. S2, article no. 56, 2012.

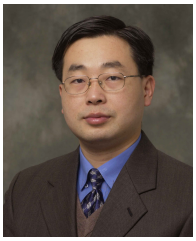
25. Y. Ding and W. Zhang, "Multicore real-time scheduling to reduce inter-thread cache interferences," *Journal of Computing Science and Engineering,* vol. 7, no. 1, pp. 67-80, 2013.
26. M. Lv, W. Yi, N. Guan, and G. Yu, "Combining abstract interpretation with model checking for timing analysis of multicore software," in *Proceedings of the 31th IEEE International Real-time System Symposium*, San Diego, CA, 2010, pp. 339-349.
27. T. Kelter, H. Falk, P. Marwedel, S. Chattopadhyay, and A. Roychoudhury, "Bus-aware multicore WCET analysis through TDMA offset bounds," in *Proceedings of the 23rd Euromicro Conference on Real-time Systems*, Porto, Portugal, 2011, pp. 3-12.
28. S. Chattopadhyay, A. Roychoudhury, and T. Mitra, "Modeling shared cache and bus in multi-cores for timing analysis," in *Proceedings of the 13th International Workshop on Software and Compilers for Embedded Systems*, St. Goar, Germany, 2010, article no. 6.
29. P. Puschner and G. Bernat, "Wcet analysis of reusable portable code," in *Proceedings of the 13th Euromicro Conference on Real-Time Systems*, Delft, the Netherlands, 2001, pp. 45-52.
30. G. Bernat, A. Burns, and A. Wellings, "Portable worst-case execution time analysis using Java byte code," in *Proceedings of the 12th Euromicro conference on Real-time systems,* Stockholm, Sweden, 2000, pp. 81-88.
31. L. Bate, G. Bernat, and P. Puschner, "Java virtual-machine support for portable worst-case execution-time analysis," in *Proceedings of the 5th IEEE International Symposium on Object-Oriented Real-Time Distributed Computing*, Washington, DC, 2002, pp. 83-90.
32. Kaffe Java Virtual Machine, http://www.kaffe.org/.
33. U. Brinkschulte, C. Krakowski, J. Kreuzinger, and T. Ungerer, "A multithreaded Java microcontroller for thread-oriented real-time event handling," in *Proceedings of the International Conference on Parallel Architectures and Compilation Techniques*, Newport Beach, CA, 1999, pp. 34-39.
34. E. Y. Hu, G. Bernat, and A. Wellings, "Addressing dynamic dispatching issues in WCET analysis for object-oriented hard real-time systems," in *Proceedings of the 5th IEEE International Symposium on Object-Oriented Real-Time Distributed Computing*, Washington, DC, 2002, pp. 109-116.
35. E. Y. Hu, A. Wellings, and G. Bernat, "XRTJ: an extensible distributed high-integrity real-time Java environment," *Real-Time and Embedded Computing Systems and Applications*, *Lecture Notes in Computer Science vol. 2968*, J. Chen and S. Hong, editors, Heidelberg: Springer, pp. 208-228, 2004
36. T. Harmon and R. Klefstad, "Toward a unified standard for worst-case execution time annotations in real-time Java," in *Proceedings of 21th International Parallel and Distributed Processing Symposium*, Long Beach, CA, 2007, pp. 1-8.
37. S. Hepp and M. Schoeberl, "Worst-case execution time based optimization of real-time Java programs," in *Proceedings of IEEE 15th International Symposium on Object/Component/Service-Oriented Real-Time,* Guangdong, China, 2012, pp. 64-70.
38. E. Y. Hu, A. Wellings, and G. Bernat, "Deriving Java virtual machine timing models for portable worst-case execution time analysis," *On The Move to Meaningful Internet Systems 2003: OTM 2003 Workshops, Lecture Notes in Computer Science vol. 2889*, R. Meersman and Z. Tari, editors, Heidelberg: Springer, pp 411-424, 2003.
39. I. Bate, G. Bernat, G. Murphy, and P. Puschner, "Low-level analysis of a portable Java byte code WCET analysis framework," in *Proceedings of Seventh International Conference on Real-Time Computing Systems and Applications,* Cheju, Korea, 2000, pp. 39-48.
40. W. S. Beebee and M. C. Rinard, "An implementation of scoped memory for real-time Java," in *Proceedings of the 1st International Workshop on Embedded Software*, London, UK, 2001, pp. 289-305.
41. A. Corsaro and D. C. Schmidt, "The design and performance of the jRate real-time Java implementation," *On the Move to Meaningful Internet Systems 2002: CoopIS, DOA, and ODBASE, Lecture Notes in Computer Science vol. 2519*, R. Meersman and Z. Tari, editors, Heidelberg: Springer, pp. 900-921, 2002.
42. F. Pizlo, J. Fox, D. Holmes, and J. Vitek, "Real-time Java scoped memory: design patterns and semantics," in *Proceeding of the 7th IEEE International Symposium on Object-Oriented Real-Time Distributed Computing*, Vienna, Austrai, 2004, pp. 101-110.
43. C. Andreae, Y. Coady, C. Gibbs, J. Noble, J. Vitek, and T. Zhao, "Scoped types and aspects for real-time Java memory management," *Real-Time Systems*, vol. 37, no. 1, pp. 1-44, 2007.
44. J. Henry and G. Baker, "List processing in real time on a serial computer," *Communications of the ACM*, vol. 21, no. 4, pp. 280-294, 1978.
45. M. Kero, J. Nordlander, and P. Lindgren, "A correct and useful incremental copying garbage collector," in *Proceedings of the 6th International Symposium on Memory Management*, Montreal, Canada, 2007, pp. 129-140.
46. F. Siebert, "Hard real-time garbage-collection in the Jamaica virtual machine," in *Proceedings of the 6th International Conference on Real-Time Computing Systems and Applications*, Hong Kong, 1999, pp. 96-102.
47. W. J. Schmidt and K. D. Nilsen, "Performance of a hardware-assisted real-time garbage collector," in *Proceedings of the 6th International Conference on Architectural Support for Programming Languages and Operating Systems*, San Jose, CA, 1994, pp. 76-85.
48. D. F. Bacon, P. Cheng, and V. T. Rajan, "A real-time garbage collector with low overhead and consistent utilization," *ACM SIGPLAN Notices*, vol. 38, no. 1, pp. 285-298, 2003.
49. IBM WebSphere Virtual Machine, http://www-306.ibm.com/software/webservers/realtime/.
50. D. Bacon and P. Cheng, "*The metronome: an simpler approach to garbage collection in real-time systems*," *On The Move to Meaningful Internet Systems 2003: OTM 2003 Workshops, Lecture Notes in Computer Science vol. 2889*, R. Meersman and Z. Tari, editors, Heidelberg: Springer, pp 466-478, 2003.
51. D. F. Bacon, P. Cheng, and V. T. Rajan, "Controlling frag-

mentation and space consumption in the metronome, a real-time garbage collector for Java," *ACM SIGPLAN Notices*, vol. 38, no. 7, pp. 81-92, 2003.
52. R. Henriksson, "Scheduling garbage collection in embedded systems," Ph.D. dissertation, Department of Computer Science, Lund University, Lund, Sweden, 1998.
53. S. G. Robertz and R. Henriksson, "Time-triggered garbage collection: robust and adaptive real-time GC scheduling for embedded systems," *ACM SIGPLAN Notices*, vol. 38, no. 7, pp. 93-102, 2003.
54. M. Schoeberl and J. Vitek, "Garbage collection for safety critical Java," in *Proceedings of the 5th International Workshop on Java Technologies for Real-Time and Embedded Systems,* Vienna, Austria, 2007, pp. 85-93.
55. H. Cho, C. Na, B. Ravindran, and E. D. Jensen, "On scheduling garbage collector in dynamic real-time systems with statistical timing assurances," *Real-Time Systems*, vol. 36, no. 1-2, pp. 23-46, 2007.
56. P. Cheng and G. E. Blelloch, "A parallel, real-time garbage collector," in *Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation,* Snowbird, UT, 2001, pp. 125-136.
57. G. E. Blelloch and P. Cheng, "On bounding time and space for multiprocessor garbage collection," *ACM SIGPLAN Notices*, vol. 34, no. 5, pp. 104-117, 1999.
58. Y. Xian and G. Xiong, "Minimizing memory requirement of real-time systems with concurrent garbage collector," *ACM SIGPLAN Notices*, vol. 40, no. 3, pp. 40-48, 2005.
59. R. L. Hudson and J. E. B. Moss, "Sapphire: copying GC without stopping the world," in *Proceedings of the Joint ACM-ISCOPE Conference on Java Grande,* Palo Alto, CA, 2001, pp. 48-57.
60. F. Pizlo, E. Petrank, and B. Steensgaard, "A study of concurrent real-time garbage collectors," in *Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation*, Tucson, AZ, 2008, pp. 33-44.
61. F. Pizlo, D. Frampton, E. Petrank, and B. Steensgaard, "Stopless: a real-time garbage collector for multiprocessors," in *Proceedings of the 6th International Symposium on Memory Management*, Montreal, Canada, 2007, pp. 159-172.
62. Sun Microsystem Real-Time Java System, http://Java.sun.com/Javase/technologies/realtime/.
63. Open Virtual Machine, Purdue University, http://www.cs.purdue.edu/homes/jv/soft/ovm/.
64. Java SE HotSpot at a Glance, http: //www.oracle.com/technetwork/Java/Javase/tech/index-jsp-136373.html.
65. Jikes RVM, http://www.jikesrvm.org/.
66. B. Alpern, C. R. Attanasio, J. J. Barton, M. G. Burke, P. Cheng, J. D. Choi, A. Cocchi, S. J. Fink, D. Grove, M. Hind, et al., "The Jalapeno virtual machine," *IBM Systems Journal*, vol. 39, no. 1, pp. 211-221, 2000.
67. U. Hlzle and D. Ungar. "Reconciling responsiveness with performance in pure object-oriented languages," *ACM Transactions on Programming Languages and Systems*, vol. 18, no. 4, pp. 355-400, 1996.
68. M. P. Plezbert and R. K. Cytron, "Does 'just in time' = 'better late than never'?," in *Proceedings of the 24th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, Paris, France, 1997, pp. 120-131.
69. R. M. Karp, "On-line algorithms versus off-line algorithms: How much is it worth to know the future?," in *Proceedings of the IFIP 12th World Computer Congress on Algorithms, Software, Architecture*, Madrid, Spain, 1992, pp. 416-429.
70. T. P. Kistler, "Continuous program optimization," Ph.D. dissertation, University of California, Irvine, CA, 1999.
71. T. Kistler and M. Franz, "Continuous program optimization: a case study," *ACM Transactions on Programming Languages and Systems,* vol. 25, no. 4, pp. 500-548, 2003.
72. K. Pettis and R. C. Hansen, "Profile guided code positioning," *ACM SIGPLAN Notices*, vol. 25, no. 6, pp. 16-27, 1990.
73. P. P. Chang, S. A. Mahlke, and W. W. Hwu, "Using profile information to assist classic code optimizations," *Software-Practice & Experience*, vol. 21, no. 12, pp. 1301-1321, 1991.
74. W. Hwu, S. A. Mahlke, W. Y. Chen, P. P. Chang, N. J. Warter, R. A. Bringmann, R. G. Ouellette, R. E. Hank, T. Kiyohara, G. E. Haab, J. G. Holm, and D. M. Lavery, "The superblock: an effective technique for VLIW and superscalar compilation," *Journal of Supercomputing*, vol. 7, no. 1, pp. 229-248, 1993.
75. R. Cohn and P. G. Lowney, "Design and analysis of profile-based optimization in Compaq's compilation tools for alpha," *Journal of Instruction-Level Parallelism*, vol. 3, pp. 1-25, 2000.
76. M. Mock, C. Chambers, and S. Eggers, "Calpa: a tool for automating selective dynamic compilation," in *Proceedings of the 33rd annual ACM/IEEE international symposium on Microarchitecture*, Monterey, CA, 2000, pp. 291-302.
77. M. Arnold, M. Hind, and B. G. Ryder, "Online feedback-directed optimization of Java," *ACM SIGPLAN Notices*, vol. 37, no. 11, pp. 111-129, 2002.
78. J. Whaley, "Partial method compilation using dynamic profile information," *ACM SIGPLAN Notices*, vol. 36, no. 11, pp. 166-179, 2001.
79. M. Arnold and B. G. Ryder, "A framework for reducing the cost of instrumented code," *ACM SIGPLAN Notices*, vol. 36, no. 5, pp. 168-179, 2001.
80. M. Hirzel and T. Chilimbi, "Bursty tracing: a framework for low-overhead temporal profiling," in *Proceeding of the 4th ACM Workshop on Feedback-Directed and Dynamic Optimization*, Austin, TX, 2001, pp. 117-126.
81. T. M. Chilimbi and M. Hirzel, "Dynamic hot data stream prefetching for general-purpose programs," *ACM SIGPLAN Notices*, vol. 37, no. 5, pp. 199-209, 2002.
82. C. Krintz, "Coupling on-line and off-line profile information to improve program performance," in *Proceeding of the International Symposium on Code Generation and Optimization*, San Francisco, CA, 2003, pp. 69-78.
83. S. J. Fink and F. Qian, "Design, implementation and evaluation of adaptive recompilation with on-stack replacement," in *Proceedings of the International Symposium on Code Generation and Optimization*, San Francisco, CA, 2003, pp. 241-252.
84. G. J. Hansen, "Adaptive systems for the dynamic run-time optimization of programs," Ph.D. dissertation, Carnegie-

Mellon University, Pittsburgh, PA, 1974.
85. C. Chambers and D. Ungar, "Making pure object-oriented languages practical," in *Proceeding of ACM Conference Object-Oriented Programming Systems, Languages, and Applications*, Phoenix, AZ, 1991, pp. 1-15.
86. T. Suganuma, T. Yasue, and T. Nakatani, "A region-based compilation technique for a Java just-in-time compiler," *ACM SIGPLAN Notices*, vol. 38, no. 5, pp. 312-323, 2003.
87. U. Holzle, C. Chambers, and D. Ungar, "Debugging optimized code with dynamic deoptimization," *ACM SIGPLAN Notices*, vol. 27, no. 7, pp. 32-43, 1992.
88. H. S. Oh, S. M. Moon, and D. H. Jung, "Hybrid Java compilation of just-in-time and ahead-of-time for embedded systems," *Journal of Circuits, Systems and Computers*, vol. 21, no. 2, 2012.
89. B. J. Bradel and T. S. Abdelrahman, "Automatic trace-based parallelization of Java programs," in *Proceedings of the International Conference on Parallel Processing*, Xi'an, China, 2007, p. 26.
90. B. J. Bradel and T. S. Abdelrahman, "The potential of trace-level parallelism in Java programs," in *Proceedings of the 5th International Symposium on Principles and Practice of Programming in Java*, Lisboa, Portugal, 2007, pp. 167-174.
91. S. Guo and J. Palsberg, "The essence of compiling with traces," in *Proceedings of the 38th annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, Austin, TX, 2011, pp. 563-574.
92. H. Inoue, H. Hayashizaki, P. Wu, and T. Nakatani, "A trace-based Java JIT compiler retrofitted from a method-based compiler," in *Proceedings of the 8th Annual IEEE/ACM International Symposium on Code Generation and Optimization*, Chamonix, France, 2011, pp. 246-256.
93. Y. Sun and W. Zhang, "On-line trace based automatic parallelization of Java programs on multicore platforms," *Journal of Computing Science and Engineering*, vol. 6, no. 2, pp. 105-118, 2012.
94. Q. Zhu and D. Vergerov, "Adaptive optimization of the sun Java real-time system garbage collector," Sun Microsystems, Mountain View, CA, Technical Report, 2009.
95. M. Stoodley, K. Ma, and M. Lut, "Real-time Java, part 2: comparing compilation techniques," IBM, Armonk, NY, Technical Report, 2007.
96. M. Fulton and M. Stoodley, "Compilation techniques for real-time Java programs," in *Proceedings of the International Symposium on Code Generation and Optimization,* San Jose, CA, 2007, pp. 221-231.
97. W. Zhang and Y. Sun, "Time-predictable Java dynamic compilation on multicore processors," *Journal of Computing Science and Engineering,* vol. 6, no. 1, pp. 26-38, 2012.
98. Y. Sun and W. Zhang, "Exploiting multi-core processors to improve time predictability for real-time Java computing," in *Proceedings of the 15th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*, Beijing, China, 2009, pp. 447-454.
99. J. Kreuzinger, U. Brinkschulte, M. Pfeffer, S. Uhrig, and T. Ungerer, "Real-time event-handling and scheduling on a multithreaded Java microcontroller," *Microprocessors and Microsystems*, vol. 27, no. 1, pp. 19-31, 2003.
100. M. Zabel, T. B. Preuber, P. Reichel, and R. G. Spallek, "Secure, real-time and multi-threaded general-purpose embedded Java microarchitecture," in *Proceedings of the 10th Euromicro Conference on Digital System Design Architectures, Methods and Tools,* Lubeck, Germany, 2007, pp. 59-62.
101. M. Schoeberl, "JOP: a Java optimized processor for embedded real-time systems," Ph.D. dissertation, University of Technology, Vienna, Austria, 2005.
102. M. Schoeberl, "A Java processor architecture for embedded real-time systems," *Journal of Systems Architecture*, vol. 54, no. 1-2, pp. 265-286, 2008.
103. M. Schoeberl and R. Pedersen, "WCET analysis for a Java processor," in *Proceedings of the 4th International Workshop on Java Technologies for Real-Time and Embedded Systems,* Paris, France, 2006, pp. 202-211.
104. C. Z. Lei, T. Z. Qiang, W. L. Ming, and T. S. Liang, "An effective instruction optimization method for embedded real-time Java processor," in *Proceedings of the International Conference on Parallel Processing Workshops*, Oslo, Norway, 2005, pp. 225-231.
105. Z. Chai, W. Zhao, and W. Xu, "Real-time Java processor optimized for RTSJ," in *Proceedings of the ACM Symposium on Applied Computing*, Seoul, Korea, 2007, pp. 1540-1544.
106. T. Harmon and R. Klefstad, "Interactive back-annotation of worst-case execution time analysis for Java microprocessors," in *Proceedings of the 13th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*, Daegu, Korea, 2007, pp. 209-216.
107. A. Armbruster, J. Baker, A. Cunei, C. Flack, D. Holmes, F. Pizlo, E. Pla, M. Prochazka, and J. Vitek, "A real-time Java virtual machine with applications in avionics," *ACM Transactions on Embedded Computing Systems*, vol. 7, no. 1, article no. 5, 2007.
108. jRate, http://jrate.sourceforge.net/.
109. Javolution Library, http://javolution.org/.
110. C. Pitter and M. Schoeberl, "Towards a Java multiprocessor," in *Proceedings of the 5th International Workshop on Java Technologies for Real-Time and Embedded Systems,* Vienna, Austria, 2007, pp. 144-151.

Yu Sun

Yu Sun received his Ph.D. degree in Electrical and Computer Engineering in 2010 at Southern Illinois University Carbondale, Carbondale, IL, USA, and B.E. degree in Computer Science at Tsinghua University, Beijing, China in 2003. His research interests include compiler optimization, parallelization, embedded system and real-time system. He is currently working as a senior software engineer at MathWorks, Inc.



Wei Zhang

Dr. Wei Zhang is an associate professor in Electrical and Computer Engineering of Virginia Commonwealth University. Dr. Wei Zhang received his Ph.D. from the Pennsylvania State University in 2003. From August 2003 to July 2010, Dr. Zhang worked as an assistant professor and then as an associate professor at Southern Illinois University Carbondale. His research interests are in embedded and real-time computing systems, computer architecture, compiler, and low-power systems. Dr. Zhang has received the 2009 SIUC Excellence through Commitment Outstanding Scholar Award for the College of Engineering, and 2007 IBM Real-time Innovation Award. His research has been supported by NSF, IBM, Intel, Motorola and Altera. He is a senior member of the IEEE. He has served as a member of the organizing or program committees for several IEEE/ACM international conferences and workshops.