Available online at www.sciencedirect.com
ScienceDirect
Journal of Systems Architecture 54 (2008) 265–286
JOURNAL OF SYSTEMS ARCHITECTURE
www.elsevier.com/locate/sysarc

A Java processor architecture for embedded real-time systems

Martin Schoeberl

Institute of Computer Engineering, Vienna University of Technology, Austria

Received 11 April 2006; received in revised form 24 May 2007; accepted 11 June 2007
Available online 22 June 2007

Abstract

Architectural advancements in modern processor designs increase average performance with features such as pipelines, caches, branch prediction, and out-of-order execution. However, these features complicate worst-case execution time analysis and lead to very conservative estimates. JOP (Java Optimized Processor) tackles this problem from the architectural perspective – by introducing a processor architecture in which simpler and more accurate WCET analysis is more important than average case performance.

This paper presents a Java processor designed for time-predictable execution of real-time tasks. JOP is the implementation of the Java virtual machine in hardware. JOP is intended for applications in embedded real-time systems and the primary implementation technology is in a field programmable gate array. This paper demonstrates that a hardware implementation of the Java virtual machine results in a small design for resource-constrained devices.

Keywords: Real-time system; Time-predictable architecture; Java processor

1. Introduction

Compared to software development for desktop systems, current software design practice for embedded real-time systems is still archaic. C/C++ and even assembly language are used on top of a small real-time operating system. Many of the benefits of Java, such as safe object references, the notion of concurrency as a first-class language construct, and its portability, have the potential to make embedded systems much safer and simpler to program. However, Java technology is seldom used in embedded real-time systems, due to the lack of acceptable real-time performance.

Traditional implementations of the Java virtual machine (JVM) as interpreter or just-in-time compiler are not practical. An interpreting virtual machine is too slow and therefore waste of processor resources. Just-in-time compilation has several disadvantages for embedded systems, notably that a compiler (with the intrinsic memory overhead) is necessary on the target system. Due to compilation during runtime, execution times are practically not predictable.[1]

E-mail address: mschoebe@mail.tuwien.ac.at

[1] One could add the compilation time of a method to the WCET of that method. However, in that case we need a WCET analyzable compiler and the WCET gets impractical high.

1383-7621/$ - see front matter © 2007 Elsevier B.V. All rights reserved.
doi:10.1016/j.sysarc.2007.06.001

This paper introduces the concept of a Java processor [51] for embedded real-time systems, in particular the design of a small processor for resource-constrained devices with time-predictable execution of Java programs. This Java processor is called JOP – which stands for Java Optimized Processor – based on the assumption that a full native implementation of all Java bytecode instructions [30] is not a useful approach.

Worst-case execution time (WCET) estimates of tasks are essential for designing and verifying real-time systems. Static WCET analysis is necessary for hard real-time systems. In order to obtain a low WCET value, a good processor model is necessary. Traditionally, only simple processors can be analyzed using practical WCET boundaries. Architectural advancements in modern processor designs tend to abide by the rule: '*Make the average case as fast as possible*'. This is orthogonal to '*Minimize the worst-case*' and has the effect of complicating WCET analysis. This paper tackles this problem from the architectural perspective – by introducing a processor architecture in which simpler and more accurate WCET analysis is more important than average case performance.

JOP is designed from ground up with time-predictable execution of Java bytecode as major design goal. All function units, and especially the interaction between them, are carefully designed to avoid any time dependency between bytecodes. The architectural highlights are:

(i) Dynamic translation of the CISC Java bytecodes to a RISC, stack based instruction set (the microcode) that can be executed in a three-stage pipeline.
(ii) The translation takes exactly one cycle per bytecode and is therefore pipelined. Compared to other forms of dynamic code translation the proposed translation does not add any variable latency to the execution time and is therefore time predictable.
(iii) Interrupts are inserted in the translation stage as special bytecodes and are transparent to the microcode pipeline.
(iv) The short pipeline (four stages) results in short conditional branch delays and a hard to analyze branch prediction logic or branch target buffer can be avoided.
(v) Simple execution stage with the two topmost stack elements as discrete registers. No write back stage or forwarding logic is needed.
(vi) Constant execution time (one cycle) for all microcode instructions. No stalls in the microcode pipeline. Loads and stores of object fields are handled explicitly.
(vii) No time dependencies between bytecodes result in a simple processor model for the low-level WCET analysis.
(viii) Time-predictable instruction cache that caches whole methods. Only invoke and return instruction can result in a cache miss. All other instructions are guaranteed cache hits.
(ix) Time-predictable data cache for local variables and the operand stack. Access to local variables is a guaranteed hit and no pipeline stall can happen. Stack cache fill and spill is under microcode control and analyzable.
(x) No prefetch buffers or store buffers that can introduce unbound time dependencies of instructions. Even simple processors can contain an instruction prefetch buffer that prohibits its exact WCET values. The design of the method cache and the translation unit avoids the variable latency of a prefetch buffer.
(xi) Good average case performance compared to other non real-time Java processors.
(xii) Avoidance of hard to analyze architectural features results in a very small design. Therefore an available real estate can be used for a chip multi-processor solution.

In this paper, we will present the architecture of the real-time Java processor and the evaluation results for JOP, with respect to WCET, size and performance. We will show that the execution time of Java bytecodes can be exactly predicted in terms of the number of clock cycles. We will also evaluate the general performance of JOP in relation to other embedded Java systems. Although JOP is intended as a processor with a low WCET for all operations, its general performance is still important. We will see that a real-time processor architecture does not need to be slow.

In the following section, related work on real-time Java, Java processors, and issues with the low-level WCET analysis for standard processors is presented. In Section 3, a brief overview of the architecture of JOP is given, followed by a more detailed description of the microcode. In Section 4 it is shown that our objective of providing an easy target for WCET analysis has been achieved. Section 5 compares JOP's resource usage with other soft-core processors. In the Section 6, a number of