Computers and Electrical Engineering 35 (2009) 904–919

Contents lists available at ScienceDirect

Computers and Electrical Engineering

journal homepage: www.elsevier.com/locate/compeleceng

An accelerator design for speedup of Java execution in consumer mobile devices

Lu Yan[a,*], Zheng Liang[b]

[a] School of Computer Science, University of Hertfordshire, Hatfield, Hertfordshire AL10 9AB, UK
[b] Turku Centre for Computer Science, Joukahaisenkatu 3-5, FIN-20520, Turku, Finland

A R T I C L E I N F O

Article history:
Available online 8 January 2009

Keywords:
Java accelerator
High performance
Low power
Energy efficient
Speedup
Asynchronous

A B S T R A C T

In today's consumer electronics market, Java has become one of the most important programming languages for the rapid development of mobile applications – spanning from home appliances/controllers, mobile and communication devices, to network-centric applets. However, the demand for high-performance low-power Java-based consumer mobile applications puts forward new challenges to the system design and implementation. This paper analyzes the energy consumption, execution efficiency, and speed issues of Java applications in a typical consumer mobile device environment. By adopting a hardware-assisted approach, we introduce a Java accelerator with a companion Java virtual machine. The accelerator is designed in an asynchronous style, and can be integrated with most existing processors and operating systems. The core architecture, design philosophy, and implementation considerations are presented in detail in this paper.

1. Introduction

Java has become one of the most popular and portable languages in the consumer electronics market for its *write once, run any where* promise. It is expected that this enabling technology will make it much easier to develop portable software and standardized interfaces that span a spectrum of hardware platforms. Recently, more and more consumer mobile devices adopt Java as the programming language, especially mobile phones and smart home appliances. Compared to the traditional embedded system development languages like C and Assembly, Java provides a number of advantages including hardware independency (i.e., write once, run anywhere), design reuse, dynamic upgrade of products, and faster time-to-market, which will significantly facilitate product design and resource planning.

However, the Java technology comes with compromises as well, and there are several problems that need to be solved. These include slow execution rates, excessive memory requirements, very high power consumption mainly because of the excessive memory usage, and unpredictable real-time performance. The approaches to increase execution speed, like just-in-time (JIT) compilation, increase the memory consumption and thus also power consumption. On the other hand, optimizing the Java virtual machine for small memory consumption typically decreases the speed.

To solve this dilemma, we developed an advanced self-timed Java accelerator core which has extremely low-power consumption while providing sufficient performance for even the most demanding real-time telecommunication and multimedia applications. The goal is that the accelerator can be directly attached to any general-purpose processor core running Java applications. Asynchronous self-timed circuit technology, where timing is based on local handshakes between circuit blocks

* Corresponding author.
E-mail address: lu.yan@ieee.org (L. Yan).

0045-7906/$ - see front matter © 2008 Elsevier Ltd. All rights reserved.
doi:10.1016/j.compeleceng.2008.11.021

instead of a global clock signal, provides a promising platform for obtaining a high-performance low-power Java accelerator implementation.

The remainder of this paper is organized as follows. In Section 2, we explain the Java execution modes in a Java virtual machine. In Section 3, we review the existing work on hardware-assisted Java technologies. The architecture of our asynchronous Java accelerator is presented with details in Section 4. We discuss our design philosophies for energy efficiency in Section 5, and the system level simulation and implementation considerations in Section 6. We conclude the paper in Section 7.

2. Java application execution

Java applications are first compiled into bytecode streams to execute in the Java virtual machine (JVM). Bytecode representations are portable formats that allow programs, whether small applets in embedded systems or large desktop applications, to run on many platforms. The core of the JVM implementation is the execution engine that executes bytecode instructions. It is important that the JVM provides an efficient execution/runtime environment across diverse hardware platforms.

Each Java application runs inside its own JVM. In the JVM specification, the behavior of a virtual machine instance is described in terms of subsystems, memory areas, data types and instructions. These components describe an abstract inner architecture of the abstract JVM. As shown in Fig. 1, each JVM has a class loader subsystem, which is a mechanism for loading types (classes and interfaces) when given fully qualified names. Each JVM also has an execution engine, which is a mechanism responsible for executing the instructions contained in the methods of loaded classes. The JVM organizes the memory it needs to execute a program into several runtime data areas.

Each thread of a running Java application is a distinct instance of the virtual machine's execution engine. In a thread, bytecode execution can be implemented in either software or hardware:

Interpreter:	it is a software emulation of the virtual machine, in this case, Java interpreter has an additional overhead and more executing cycles are needed.
Just-in-time (JIT) compiler:	it compiles a Java method into native instructions on the fly and caches the native sequence. The JIT compilers' memory requirement is pretty high for embedded systems.
Hardware accelerator:	one mode is to translate the bytecode into binary machine code of native main processor; the other is to work as coprocessor which specially executes the bytecode instruction.
Java native processor:	implements the JVM directly on silicon. It avoids the overhead of translation of the bytecodes to another processor's native language, but it is difficult to upgrade to support new runtimes.

Since our target is for most existing processors, coprocessor mode is the best choice, which means frequently used bytecode instructions are to be executed by the Java accelerator.

Our Java accelerator is designed for consumer mobile devices (e.g. low-end embedded systems), where only one CPU accesses memory and works as master module in the system bus. As shown in Fig. 2, during program running, the Java accelerator works as a filter between main processor and external memory. If the main processor is executing non-Java tasks, or the address of accessed memory is out of current Java stack, the Java accelerator will be transparent.

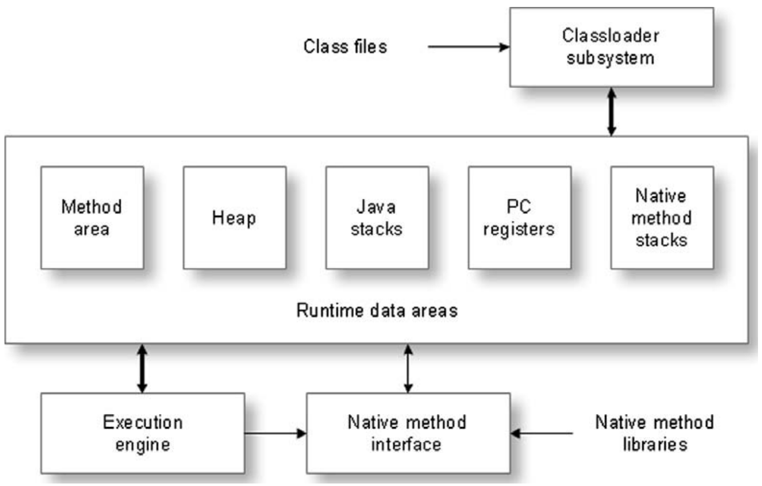

Fig. 1. Typical architecture of a Java virtual machine (JVM).