A Hardware/Software Co-design and Co-verification on a Novel Embedded Object-Oriented Processor

Chi Hang Yau, Yi Yu Tan, Pak Lun Mok, Wing Shing Yu, and Anthony S. Fong

Department of Electronic Engineering, City University of Hong Kong,
Tat Chee Avenue, Hong Kong
Anthony.Fong@cityu.edu.hk

Abstract. In the past, programming language are procedural, the design concept is based on the module and scope which are difficult to manage, but nowadays, the programming trend is Object-Oriented Programming (OOP), where objects are the key elements to build up application and the communications between different objects are through method invocation. A novel object-oriented processor offers an opportunity to enhance the system security, performance and provides a more effective way to manipulate OOP instead of using a software Virtual Machine. jHISC is a novel object-oriented processor which provides a natural way to map the concept of OOP into architectural level through the hardware object data structure. Our solution is to design secure hardware object data structures on a novel processor with Just-In-Time compilation for Java which then makes it possible to implement complex OO related bytecodes at hardware level and access some fields of object in parallel to improve the execution speed. It mainly targets J2ME and implements about 93% bytecodes and 83% OO related bytecodes in hardware directly.

1 Introduction

Computer hardware becomes quite mature, however, software trend is moving to Object-Oriented Programming (OOP) with advantages on reusable design and better security. In traditional systems, processor architecture like Complex Instruction Set Computer (CISC) or Reduced Instruction Set Computer (RISC) does not support object manipulation directly on hardware. To support object technology in nowadays system, there are mainly three categories, which are compiling objects into native (Fig. 1), developing software-based object virtual machine (Fig. 1) and developing an object-oriented processor (Fig. 2).

For the first approach, since traditional computer system does not support object-oriented programming, applications written in object-oriented programming languages like C++ are compiled into native instructions for execution. Application processes will be created and executed in their own spaces, and they are invisible from each other through the use of virtual memory system. Through the specific compiler, objects will be translated into subroutines, which are machine readable for direct execution. Compilation approach can fully optimize the generated codes, but object-oriented features, such as dynamicity behavior, will be removed. Modification of a single class requires the whole application to be re-compiled.

© IFIP International Federation for Information Processing 2005
In the second approach, an object virtual machine application is built on top of the traditional operating system. From the view of operating system, the application is just a usual process like other applications. Protection of different processes is the same as in compilation approach and objects are manipulated on this software virtual machine. The advantages of this method are that without the modification of hardware platform and current system, object technology can be supported through software emulation. But the two layers of software, virtual machine and operating system, also introduce much overhead to the overall computing system and increase the memory footprint of the system.

![Fig. 1. System Architecture of Compilation Approach and Virtual Machine Approach](image1)

In our solution, we will use the third approach where the objects are directly manipulated on an object-oriented processor, and an object-oriented operating system will be built to assist hardware in managing the heap (Fig 2). In this approach, protection of objects is governed by the object-oriented operating system with the protection features offered by the object-oriented processor. This approach can manipulate objects directly and the efficiency of the computing system is increased because no page table updating is required during context switching.

In the following sections, we will firstly discuss our secure hardware object architecture on section 2, their overall security features on section 3. Moreover, we will show the hardware/software co-verification methodology which verifies the Java compatibility on our novel architecture, through the Just-In-time compilation on Java bytecode and demonstrate some co-verification results on section 4. Finally, a conclusion will be presented on Section 5.