Java talks to FPGAs

Philippe Faes

Promoter(s): Dirk Stroobandt

Abstract— **Reconfigurable Computing Systems use a Field Programmable Gate Array (FPGA) to accelerate some computations which execute too slowly on the Central Processing Unit (CPU). The software which runs on the CPU can configure the FPGA so that it can perform a certain computation, according to the needs of the system.**

These systems generally use message-passing for communication between the software (which runs on the CPU) and the hardware (FPGA). This has the disadvantage that the software needs to be written in a specific message-passing style.

We propose a new portable and transparant interface between software (more specifically Java) and reconfigurable hardware. The software side can start complex computations using regular method calls, which can be intercepted and translated to hardware messages. Likewise, the hardware side can start methods in software.

This interface makes it easy to implement new Java/FPGA co-designs, but also to accelerate existing Java applications.

Keywords—**Java, FPGA, message passing interface**

I. Introduction

IN modern computers, the CPU is not the only chip performing calculations. Many complex calculations are performed more efficiently by Application Specific Integrated Cicuits (ASICs). E.g. three dimensional video rendering is not done by the CPU, but by specific hardware on the video card. While application specific hardware outperforms general-purpose processors, it lacks flexibility. An ASIC cannot be reprogrammed and can only be used for one type of calculation. Moreover, the non-recurring engineering cost of ASICs is very high.

FPGAs are chips that can be reconfigured as easily as CPUs can be programmed, but they can process data in a massively parallel manner just like ASICs. Even though FPGAs are a bit slower and less energy-efficient than ASICs, their reconfigurability and their price makes them a viable alternative.

FPGAs have the extra advantage that they can be configured while the system is working. The CPU can reconfigure an FPGA according to the needs of the system. After configuring the FPGA, the CPU sends messages to the FPGA instructing it to start computations [1], [2], [3].

We propose a new interface between Java and FPGAs that allows Java to activate FPGAs without sending explicit messages. Moreover, the FPGA can request exactly the same services form the JVM as Java can. This interface can be used to ease development of Java/hardware co-designs and to easily add hardware support for existing Java applications.

II. Java and the Java Virtual Machine

We choose Java as our software platform for a number of reasons. The Java language is compiled into Java bytecode. This bytecode is then executed by a Java Virtual Machine. Bytecode provides a higher level of abstraction than native machine code. It is rather easy to identify method calls and objects in bytecode,

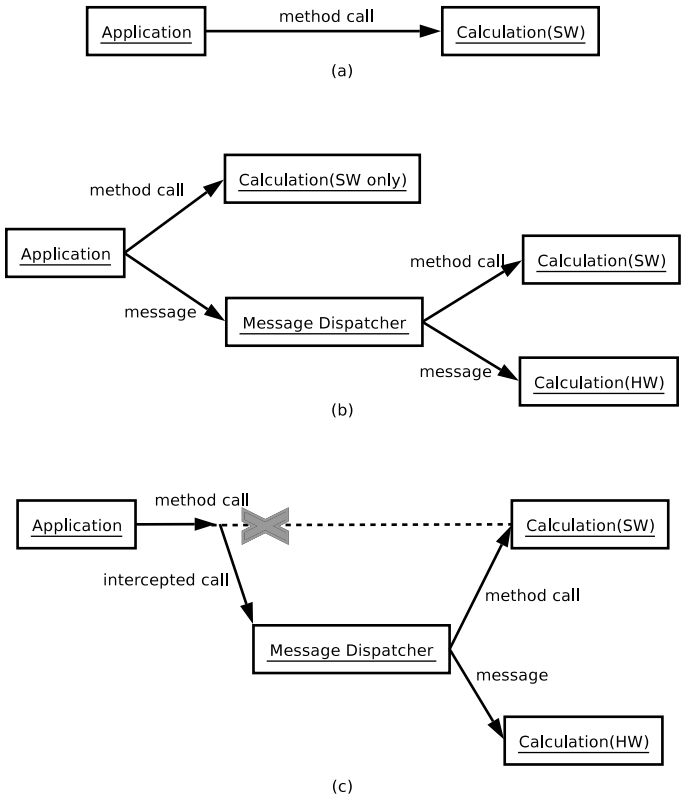

Fig. 1. **Intercepting method calls**

while these things can be very hard or even impossible in machine code.

The JVM can perform certain security checks on bytecode enforcing access rules, locking rules, etc. Java can also guarantee that no arbitrary memory locations can be read or written. This makes Java a lot more predictable than low-level languages like C. Similar advantages can be expected from other languages with an intermediate binary representation, but we choose Java because it's widely spread and well supported.

III. Transparent communication

In most existing systems, communication between software and hardware is done purely by message passing. This means that the software programmer needs to package a message and send his message to hardware.

Some systems allow these messages to be independent of the recipient. The hardware managment system will forward the message to hardware if possible or else to equivalent software (Fig. 1(b)). Note that there is still a difference between a call to a method that is a guaranteed software method and a call to a method that might be executed in hardware (the latter is called a *candidate method* by [4]). Software methods are called by

TABLE I
POSSIBLE REQUESTS

JVM to FPGA	FPGA to JVM
HW method invocation	HW method return
answer to request	SW method invocation
	lock
	unlock

regular Java method calls, while candidate methods are called through a message passing interface.

We don't want to enforce a specific interface for all candidate methods. Rather, we allow hardware acceleration for *every* computation in a Java application. We do this by *intercepting* method calls (Fig. 1(c)). Indeed, the Java programmer starts a computation by calling a method (which is equivalent to a *function* in C or Pascal).

There are two ways to intercept method calls. The first is to adapt the virtual machine. Every time a method which has hardware support gets called, the virtual machine checks the accelerating hardware. If it is available or it can be made available (e.g. by reconfiguring the FPGA) messages are composed and sent to the FPGA. If no hardware is available or the reconfiguration time is too long the software version of the method is executed. Because adapting a virtual machine is rather complicated, it has not been implemented at the time of writing.

The second possible implementation is to change the bytecode while it is loaded into the virtual machine. This method is comparatively easy because many bytecode manipulation libraries exist. We used the JBoss Aspect-Oriented Programming (AOP) library [5] built on top of a bytecode manipulation library. While the AOP library provides an easy interface for manipulating bytecode, it introduces a rather high overhead.

IV. MESSAGE PASSING PROTOCOL

We have discussed how Java can send a *hardware method invocation request* to an FPGA but it is also possible for the FPGA to send messages back to Java (cf. Table I). The most important and most obvious message is the *hardware method return* value. This message is a response to a hardware method invocation and lets the software know that the invocation has finished. The software can then decode the return value and continue its work.

During a hardware method invocation, the FPGA can initiate a new transaction by sending a *software method invocation* to the Java Virtual Machine (JVM). The parameters of the method invocation are decoded by the JVM and the method is executed. After the method finishes, its return value is sent back to the FPGA as an *answer to request* message. The software method invocation allows the hardware to request almost every action from the virtual machine: computations, input/output, creating objects, accessing object fields, throwing exceptions etc. The only thing that is impossible in a Java method is locking an object and keeping it locked when the method returns. Java requires that locks and unlocks are grouped symmetrically in every method. We added two extra request types to the protocol in order to allow *locks* and *unlocks*. The implementation of how these requests are handled is rather technical, and is beyond the scope of this paper.

V. CONCLUSION AND FUTURE WORK

We have designed and implemented both the software and the hardware side of a transparent and portable interface between Java and reconfigurable hardware. This interface can facilitate both the design of new Java/FPGA applications and the FPGA acceleration of existing Java applications. In the near future, we will develop demo applications to prove the acceleration possibilities. We will also enhance the interface so that the FPGA and the JVM can use a shared memory. This way the FPGA will not need to request access to data, but can read the data directly from the computers' main memory.

REFERENCES

[1] Bingfeng Mei, Patrick Schaumont, and Serge Vernalde, "A hardware-software partitioning and scheduling algorithm for dynamically reconfigurable embedded systems," Nov. 2000.

[2] B. Mei, S. Vernalde, H. De Man, and R. Lauwereins, "Design and optimization of dynamically reconfigurable embedded systems," in *Proc. 1st Int. Conf. on Engineering of Reconfigurable Systems and Algorithms (ERSA)*, pp. 78–84.

[3] Antti Pelkonen, Kostas Masselos, and Miroslav Cupák, "System-level modeling of dynamically reconfigurable hardware with SystemC," in *International Parallel and Distributed Processing Symposium (IPDPS'03)*, Apr. 2003, p. 174.

[4] M. Kandemir V. Narayanan L. Benini A. Bogliolo E. Lattanzi, A. Gayasen, "Improving Java performance by dynamic method migration on FPGAs," in *Proceedings of RAW 2004*, 2004.

[5] "JBoss webpage," `http://www.jboss.org`.