Exploiting FPGA Concurrency to Enhance JVM Performance

James Parnis
School of Computer Science & Software Engineering
The University of Western Australia
35 Stirling Highway,
Crawley, W. A. 6009
parnij01@csse.uwa.edu.au

September 13, 2003

Abstract

The Java Programming Language has been praised for its platform independence and portability, but because of its slow execution speed on a software Java Virtual Machine (JVM), some people decide to use faster languages such as C. Building a JVM in hardware is an obvious solution to this problem. Several approaches have been taken to try to achieve the best solution. One approach is by reducing the number of Java instructions a program has to execute along with directly executing instructions in hardware, for example on a Field Programmable Gate Array (FPGA), to increase the execution speed. Another approach is the translation of Java Byte Codes into native code by a FPGA and then executing the native code on a conventional CPU. Others have developed a multi-threaded JVM and exploited the parallelism offered by a FPGA and have specifically designed the JVM for real-time systems. This paper compares and contrasts all these approaches and then concludes that the parallelism of a FPGA should be exploited in the most general way possible by not restricting the threads of execution to a specific task.

Keywords: fpga, cpu, jvm.
CR Classification: B.5.1, B.6.1, B.7.1, C.1.2, C.1.4.

1 Introduction

Field Programmable Gate Arrays (FPGAs) are a relatively new technology. Due to the fact that they haven't been around too long, they haven't been developed as much as general purpose processors. Leaving this aside, FPGAs have the potential to make a big impact on the computer market. FPGAs are already being used in applications today such as satellites, portable electronic devices, and devices that require their circuitry to be changed on a regular basis. FPGAs are ideal for devices that run on batteries due to the low power consumption to work ratio.

There is an abundance of research being done on FPGAs — one of these research areas is building a conventional Central Processor Unit (CPU) on a FPGA. Some CPUs have been specifically designed to act as a Java Virtual Machine (JVM). Researchers have built JVMs that are very similar to software JVMs, but have added a few optimisations such as compressing multiple program instructions into one instruction and built the JVMs on hardware [6] instead of using the conventional software approach. Alliance Core [1] take on a very similar approach, but also add in extra functionality such as support for other programming languages such as C. There has also been research into building a JVM that can run four threads of execution in parallel, but with the limitation that these threads of execution are assigned to specific tasks, such as for interrupt handling [3]. Radhakrishnan et al. [5] build a system similar to a software Just-In-Time (JIT) Compiler [7], but on a FPGA. Ha et al. [2] build a JVM on a FPGA to speed up the execution of web based Java Applications.

This paper discusses how people have taken advantage of FPGAs to build JVMs. It will first discuss the advantages of a FPGA and how these advantages have been used. It will then go on to discuss the limitations and how they have been dealt with. The applications of using a FPGA for a JVM is also looked at. It will then conclude that the parallelism of a FPGA should be used to create multiple threads of execution and that they should not be specialised for a specific task.

2 Java Virtual Machine advantages with FPGAs

A JVM is a complex machine with many security features, and its complex nature makes mapping it to a conventional CPU a hard task. A FPGA is a reconfigurable device such that you can build your own electronic circuit on it with great ease and also change it after it has been built. This makes

a FPGA ideal to test and re-test multiple designs. Because of its flexible nature it is possible to build a very efficient JVM on a FPGA.

Ha et al. [2] gain an advantage by executing expensive Java instructions in hardware, on the FPGA, to speed up Internet applications that use the JVM. This way the advantage of having a platform independent language is still achieved, but because of being executed in hardware the speed of execution is increased. Schoeberl [6] takes advantage of a FPGA in his system where he makes use of the multiple ports of the block RAM devices on a FPGA to compress four instructions into one, thus increasing the efficiency of the machine fourfold. In addition to this he also optimises the instructions that are to be executed, further increasing the speed.

Implementing a JVM on a FPGA has advantages such as being able to build parallel circuits. Kreuzinger et al. [3] take advantage of the parallelism of a FPGA when building their JVM in that they have designed a system capable of running multiple threads in parallel, but with the limitation that these threads are assigned to specific tasks and are left idle when that task is not active, for example a task could be to accept input from a microphone. Radhakrishnan et al. [5] take on an approach that is similar to a Just-In-Time (JIT) compiler [7], but take advantage of using a FPGA to do the translation of Java byte-codes into native code instead of the translation being done in software. They achieve an overall better performance than a regular software JIT compiler. The fact that the JVM executes in hardware instead of software along with the flexibility of a FPGA device makes the JVM do more work in fewer clock cycles. These advantages add strength to the argument that one should build a JVM on a FPGA, but along with these advantages come certain limitations that also must be looked at and will be discussed in the next section.

3 Limitations of using a FPGA for building JVMs

FPGAs are great when flexibility of design is a major issue, which is the case when developing a complex system. A system is never perfect once built, it always needs to be changed over and over again — this makes FPGAs excellent development and building equipment. Yet, FPGAs also have limitations, as any hardware device has, due to the predefined space and limited memory, although the latest FPGAs are overcoming this with increased space and memory. A JVM is a very complex high-level machine, and because of this it is extremely hard to fit a fully fledged JVM on a FPGA. To overcome this

problem, people have limited the amount of the JVM specification [4] they have implemented.

Schoeberl [6] and Kreuzinger et al. [3] only implement certain subsets of the JVM specification, but even with this limitation the devices are well suited to be embedded in predefined systems. Schoeberl [6] overcomes the space limitation by having a supporting host machine that the FPGA communicates with and that does certain pre-processing tasks to take away some of the higher level activities that are required by a JVM. Alliance Core [1] in their CPU boast of an optimised circuit that only takes up to 3% of a top-of-the-line FPGA. Although building a JVM for a FPGA has certain limitations due to the nature of the device, there are also many applications that they can be used for. These applications will be discussed in the next section.

4 Applications of JVMs on FPGAs

We are starting to see an increase in the number of portable electronic devices used. The mobile phone is one of the first on the market and now the latest craze is MP3 players. These electronic devices have circuitry that is designed to perform tasks such as producing sound, manipulating sound, transferring data, and performing algorithms. Because portable devices run on batteries, one of the main design issues is building a circuit that consumes the least amount of power possible in order to gain the most running time on a device. The less power consumed, the better. FPGAs excel in this low power consumption field due to their ability to be designed in such a way that they achieve multiple tasks at the same time, and thus use less power.

Schoeberl [6] takes advantage of this when designing his JVM in that he optimises what would usually be done in four clock cycles to one clock cycle, for example by using the multiple ports on the block RAM to do an addition instruction in one clock cycle instead of four. This achieves the goal of using less power. This is ideal for portable devices that use a JVM to run certain tasks as it would consume less power than a software JVM and also JVMs that don't take advantage of the FPGA technology.

Kreuzinger et al. [3] take advantage of the parallelism of a FPGA by implementing a multi-threaded system, each thread being represented by a separate section of circuitry on the FPGA. The threads are all assigned to a specific task that can be carried out at the same moment in time, instead of the non-multi-threaded solution that would only have one piece of circuitry controlling all the tasks and would be using more power because it would require more clock cycles to complete the tasks. Alliance Core [1] say

that their device is aimed at the following applications: Internet appliances, multimedia controllers, automotive devices, network/switch processors, set-top boxes, networked embedded controllers, terminals, PDAs, TVs, printers, copiers, and smart cards. Ha et al. [2] have aimed their system at Java applications that are web based, taking the place of a regular software JVM in order to increase the speed.

A software JVM is relatively slow compared with executing native code directly on a CPU. JIT compilers try to overcome this by using software to convert the Java Byte Codes into native code just before execution. Radhakrishnan et al. [5] use a FPGA to perform this task. Because of the conversion being done in hardware compared to software there is a major increase in speed. This is an example of how a FPGA can be used in a conventional computer to aid in increasing the speed of a JVM. Although the FPGA is acting as a JVM, the actual execution of the instructions is being done in native code on a conventional CPU. On the other hand, Kreuzinger et al. [3] and Schoeberl [6] designed systems where the Java byte codes are directly executed on the FPGA.

5 Conclusion

Java has an advantage of being platform independent, but because of the conventional software JVM, it has a slower execution speed than a program that does not have to be interpreted before being executed on a conventional CPU. Building a JVM in hardware overcomes the speed problem, but keeps the advantage of having a platform independent language. FPGAs have been used to build JVMs and explore the advantages of building a JVM in hardware. An approach of using multiple read and write ports on the FPGA block RAM and optimising the Java byte codes has been explored by some researchers [6] [1] [5]. Ha et al. [2] have investigated the use of FPGAs for accelerating Java web applications. FPGAs by nature have a great deal of flexibility when it comes to running circuits in parallel. Kreuzinger et al. [3] use this flexibility in designing a four threaded JVM aimed specifically at real-time systems. I believe that the next step for JVMs and FPGAs would be to design a multi-threaded system similar to Kreuzinger et al. [3], but not dedicate threads to a particular task or design the device specifically for a real time system. Instead, to approach it on a more abstract level by having the threads identical in structure, not restricting them to specific tasks, and giving them a more flexible nature.

References

[1] Digital Communcation Technologies. Xilinx alliance core. (Retrieved May 10, 2003 from the World Wide Web: http://www.xilinx.com/products/logicore/alliance/digital_comm_tech/dct_lightfoot_32bit_processor.pdf).

[2] Yajun Ha, Geert Vanmeerbeeck, Patrick Schaumont, Serge Vernalde, Marc Engels, Rudy Lauwereins, and Hugo De Man. Virtual Java/FPGA interface for networked reconfiguration. In *Proceedings of the Conference on Asia South Pacific Design Automation*, pages 427–439. IEEE, January 2001.

[3] J. Kreuzinger, R. Zulauf, A. Schulz, T. Ungerer, M. Pfeffer, U. Brinkschulte, and C. Krakowski. Performance evaluations and chip-space requirements of a multithreaded Java microcontroller. (Retrieved May 8, 2003 from the World Wide Web: http://citeseer.nj.nec.com/384138.html).

[4] Tim Lindholm and Frank Yellin. *The Java Virtual Machine Specification*. Addison-Wesley, April 1999.

[5] Ramesh Radhakrishnan, Ravi Bhargava, and Lizy K. John. Improving Java performance using hardware translation. In *Proceedings of the 15th ACM International Conference on Supercomputing (ICS-01)*, pages 427–439, New York, June 17–21 2001. ACM Press.

[6] Martin Schoeberl. Using JOP at an early design stage in a real world application. In *Proceedings of the workshop on Intelligent Solutions in Embedded Systems*, June 2003.

[7] Sun Microsystems. Java JIT compiler overview. (Retrieved May 8, 2003 from the World Wide Web: http://wwws.sun.com/software/solaris/jit/).