JOP – a Real-time Java Processor

Martin Schoeberl
Institute of Computer Engineering
Vienna University of Technology, Austria
mschoebe@mail.tuwien.ac.at

1 A Real-Time Architecture

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. 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.

JOP, the Java Optimized Processor, is the solution to this issue. The processor architecture is built from ground up to be WCET analyzable. Clever features, such as the real-time stack cache and method cache, provide top performance and are still analyzable. The execution time for Java bytecodes can be exactly predicted in terms of the number of clock cycles. The proposed processor architecture results in a predictable and high-performance execution of real-time tasks in Java, without the resource implications and unpredictability of a JIT-compiler. JOP is the smallest and fastest Java processor available today.

1.1 JOP Architecture

JOP is a stack computer with its own instruction set, called microcode. Java bytecodes are translated into microcode instructions or sequences of microcode in hardware. The difference between the JVM and JOP is best described as the following:

> The JVM is a CISC stack architecture, whereas JOP is a RISC stack architecture.

Figure 1 shows JOP's major function units. A typical configuration of JOP contains the processor core, a memory interface and a number of IO devices. The processor core contains the four microcode pipeline stages: *microcode fetch*, *decode* and *execute* and an additional translation stage *bytecode fetch*. The module called extension provides the link between the processor core, and the memory and IO modules. The memory interface provides a connection between the main memory and the processor core. It also contains the method cache. The extension module controls data read and write. The *busy* signal is used by a microcode instruction to synchronize the processor core with the memory unit. The core executes microcode concurrently to memory access.

1.2 The Processor Pipeline

JOP is a fully pipelined architecture with single cycle execution of microcode instructions and a novel approach to mapping Java bytecode to these instructions.

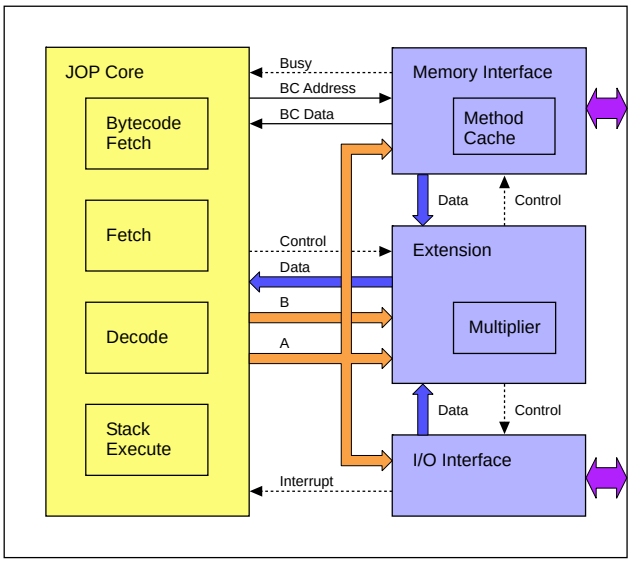

Figure 1. Block diagram of JOP

Three stages form the JOP microcode pipeline, executing microcode instructions. An additional stage in the front of the core pipeline fetches Java bytecodes – the instructions of the JVM – and translates these bytecodes into addresses in microcode. The second pipeline stage fetches JOP instructions from the internal microcode memory. Besides the usual decode function, the third pipeline stage also generates addresses for the stack cache.

The last pipeline stage performs ALU operations, load, store and stack spill or fill. At the execution stage, operations are performed with the two topmost elements of the stack. A stack machine with two explicit registers for the two topmost stack elements and automatic fill/spill needs neither an extra write-back stage nor any data forwarding. The short pipeline results in short branch delays. Therefore, a hard to analyze, with respect to WCET, branch prediction logic can be avoided.

1.3 Cache

In order to fill the gap between processor speed and the memory access time, caches are mandatory, even in embedded systems. However, standard cache organizations improve the average execution time but are difficult to predict for WCET analysis. Two time-predictable caches are proposed for JOP: a *stack cache* as a substitution for the data cache and a *method cache* to cache the instructions.

As the stack is a heavily accessed memory region, the stack – or part of it – is placed in on-chip memory. This part of the

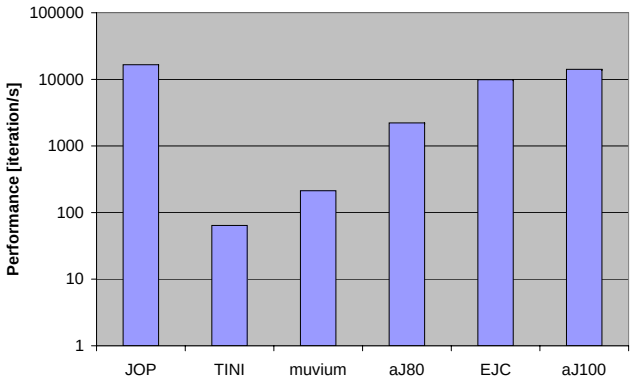

Figure 2. Performance comparison of different Java processors

stack is referred to as the *stack cache*. Fill and spill of the stack cache is subjected to microcode control and therefore time-predictable.

In JOP a novel way to organize an instruction cache, as *method cache*, is implemented. The cache stores complete methods, and cache misses can only occur on method invocation and return. Cache block replacement depends on the call tree, instead of instruction addresses. This *method cache* is easy to analyze with respect to worst-case behavior and still provides substantial performance gain when compared against a solution without an instruction cache.

2 Performance

Although JOP is intended as a processor for embedded real-time systems, whereas accurate worst case execution time analysis is more important than average case performance, its general performance is still important.

To provide a realistic workload for embedded systems, a real-time application (the *Kippfahrleitung*) was adapted to create the benchmark. Figure 2 shows the the performance of several embedded Java systems. The results are in iterations per second - a higher value means higher performance. Note that the vertical axis is logarithmic, in order to obtain useful figures to show the great variation in performance.

When comparing JOP against TINI, and muvium (a Java compiler for PIC micro-controller) we can see that a Java processor is up to 250 times faster than an interpreting or precompiled JVM on a standard processor for an embedded system. The average performance of JOP is even better than a JIT-compiler solution (+68%) on an embedded system, as represented by the EJC system (ARM720T at 74MHz). JOP is about 7 times faster than the aJ80 Java processor and about 17% faster than the aJ100.

3 Size

One major design objective in the development of JOP was to create a small system that could be implemented in a low-cost FPGA. Table 1 shows the resource usage for different configurations of JOP and different soft-core processors implemented in an Altera Cyclone low-cost FPGA. The size is given by the two basic structures in an FPGA: Logic Cells (LC) and embedded memory blocks.

Table 1. FPGA soft-core processors

Processor	Resources [LC]	Memory [KB]	fmax [MHz]
JOP Minimal	1,077	3.25	98
JOP Typical	1,831	3.25	101
Lightfoot	3,400	4	40
LEON3	7,978	10.9	35

Table 2. Gate count estimates for various processors

Processor	Core [gate]	Memory [gate]	Sum. [gate]
JOP	11K	40K	51K
picoJava	128K	314K	442K
aJ80/aj100	25K	590K	615K

The typical configuration also contains some useful I/O devices such as an UART and a timer with interrupt logic for multi-threading. In the minimal configuration shift and multiply are implemented in microcode. Lightfoot is a commercial Java processor available to be implemented in a FPGA. As a reference, LEON3, the open-source implementation if the SPARC V8 architecture, is given in the last row.

Table 2 provides gate count estimates for JOP, picoJava (a Java processor by Sun), and the aJile Java processor. Equivalent gate count for an LC varies between 5.5 and 7.4 – we chose a factor of 6 gates per LC and 1.5 gates per memory bit for the estimated gate count for JOP in the table. JOP is listed in the typical configuration.

4 Applications

Besides usage of JOP in academia as a basis for research and teaching JOP has already been deployed in three real-world applications. Balfour Beatty Austria has developed a *Kippfahrleitung* to speed up loading and unloading of goods wagons. JOP is used to control up to 15 asynchronous motors. The product TAL is a remote control and data logging system used by the lower Austria gas supply company.

Another application of JOP is in a communication device with soft real-time properties – Austrian Railways' (ÖBB) new security system for single-track lines. Each locomotive is equipped with a GPS receiver, a GPRS modem, and a communication device. JOP is the heart of the communication device in the locomotive.

5 Summary

The Java Optimized Processor (JOP) is an implementation of the Java virtual machine (JVM) in hardware. JOP is open-source and freely available for academic research.

JOP is small enough to be implemented in low-cost FPGAs. Due to the efficient implementation of the stack architecture, JOP is the smallest and fastest hardware realization of the JVM available to date. Further information is available at http://www.jopdesign.com.