MicroJava Pushes Bytecode Performance

Sun's MicroJava 701 Based on New Generation of PicoJava Core

by Jim Turley

At last month's Microprocessor Forum, Sun revealed details of its first microprocessor to execute Java bytecodes directly in hardware. The MicroJava 701, which isn't due until 2H98, will run at 200 MHz and deliver what Sun's Harlan McGhan believes is the best Java performance yet seen from any microprocessor. Some new design tweaks should also help the chip's performance on non-Java applications, such as C code.

No cost or price information was available, and Sun's performance figures are just estimates based on simulations, but initial results suggest that MicroJava 701 will be twice as fast as a 266-MHz Pentium II system on Java code. If Sun's initial estimates pan out and production stays on schedule, MicroJava 701 could be among the fastest, most cost-effective ways to execute Java code by late next year.

New PicoJava 2 Core Replaces Original Design

Interestingly, Sun's chip is not based on the PicoJava core it announced at last year's Microprocessor Forum (see MPR 10/28/96, p. 28)—and which all of Sun's licensees are currently using. Instead, Sun quietly developed a newer Java core, which it now calls PicoJava 2. This design improves both Java and non-Java performance with more instruction folding and a longer pipeline that allows higher clock rates.

The new core has not been made available to Sun's Java-chip licensees (see MPR 6/17/96, p. 4). Instead, those six companies are nearing completion of their first chips based on the older PicoJava 1 design. None of the licensees has announced a schedule for these chips, but we expect the first samples to trickle out around 2Q98—the same time as Sun's 701. Even with their head start, this leaves LG, NEC, and the other PicoJava 1 licensees in an awkward position to compete with Sun. They're also about six months behind where they wanted to be when PicoJava was announced.

Sun has not licensed PicoJava 2 because the core has not been "productized." That is, the core design is in a state that only Sun's own designers can use, according to the company. Sun expects a fully portable (or exportable) version of the core to be ready in 2Q98, roughly the time the 701 and most of the original licensees' chips begin sampling.

New Core Faster Through Fab, Pipeline Changes

The new PicoJava 2 core borrows much from its predecessor. Both cores execute about 85% of Java bytecodes directly in hardware, with the remainder trapped and emulated. Like PicoJava 1, the new core augments the bytecode instruction set with a dozen or so "extended " instructions that allow software to manipulate caches, control registers, and absolute memory addresses—all things normally prohibited for Java programs. As before, Sun has not released the list of executed or emulated bytecodes to anyone but its licensees.

The changes between PicoJava 1 and PicoJava 2, as embodied in the MicroJava chip, are designed to improve performance. As Figure 1 shows, the pipeline has been extended to six stages from four. Instruction decoding, which used to take one cycle, is now allotted two. Likewise, the execution phase has been extended by a cycle.

In a planned 0.25-micron process, Sun expects the new core to run at up to 200 MHz, twice the target frequency for the original PicoJava 1 core in 0.35-micron technology. In the same process, we would expect PicoJava 2 to run about 33% faster than PicoJava 1.

Improved Instruction Folding Aids C Code

The other major improvement in the core design involves instruction folding. PicoJava 1 was designed to recognize certain constructs or code pairings common in Java programs (and other stack-based languages). PicoJava 2 improves on this technique by expanding the scope of the comparison. Where PicoJava 1's instruction folding assisted Java applications, PicoJava 2 should improve performance on non-Java applications as well.

Instruction folding works by recognizing certain instruction sequences that occur frequently and quietly replacing them with equivalent, but quicker, operations. In this way, the PicoJava cores address one of the bottlenecks inherent in any stack-based architecture: frequent juggling of operands on the top of the stack. Java code, for example, frequently copies one operand from the interior of the stack to the top, then uses a logical or arithmetic operation to replace the top two operands with their result. PicoJava 1 skips the preliminary copy operation and routes the first operand directly to the ALU.

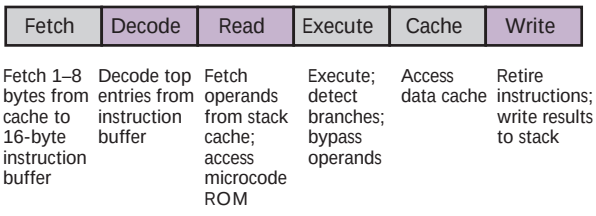

Figure 1. The PicoJava 2 core, which forms the basis of the MicroJava 701 chip, uses a new six-stage pipeline that breaks bytecode decoding and execution into two stages apiece.

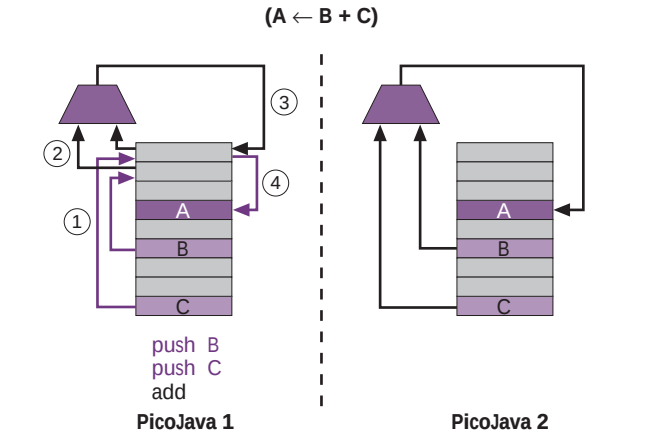

Figure 2. The PicoJava 2 core is more aggressive than the original PicoJava 1 in bypassing stack manipulation. The core's decode logic recognizes common stack operations and converts them to straightforward two-input ALU functions like a conventional CPU.

The PicoJava 2 design goes one step further, routing any two operands from the interior of the stack directly to the ALU. This enhancement helps C code and other conventional languages more than it helps Java, because C compilers (and programmers) frequently use two source operands in their calculations. By fetching operands directly from the stack, PicoJava 2 more closely emulates a conventional register set, which maps much more easily onto the code engines of most compilers.

A normal C-language statement that adds two variables generally translates to a single instruction on most RISC processors. As Figure 2 shows, executing this on a stack-based architecture such as PicoJava takes from one to four instructions, depending on where the source operands happen to be in the stack and where the result will be stored. One or both source operands may have to be copied to the top of the stack before the addition can be performed, costing one or two extra instructions. (All cases assume the operands are already loaded from memory.)

The PicoJava 2 core recognizes the case where two operands are copied to the top of the stack and replaced by their result; it then bypasses both copies, shuttles the two operands to the ALU, and stores the result in the destination register. Thus, PicoJava 2 reduces this otherwise awkward construct to a single operation, just like a RISC chip.

Note that the compiler or programmer doesn't have to be aware of the folding; PicoJava 2 does it automatically, like register renaming or instruction reordering. The object code still includes the intermediate push instructions.

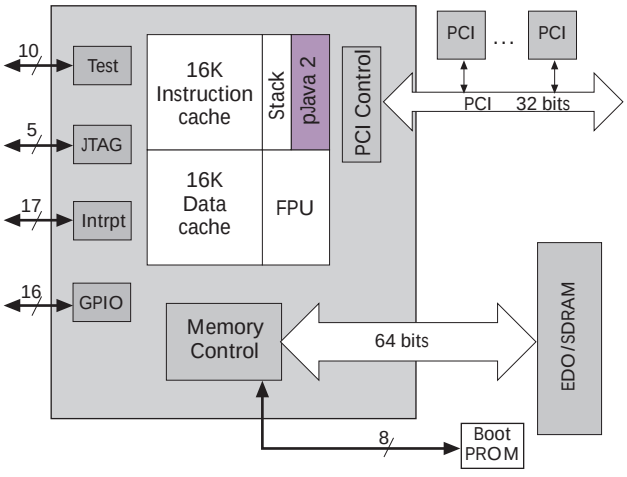

Figure 3. Sun's initial MicroJava 701 implementation will include a pair of 16K caches, a 64-bit memory controller, and a PCI bus.

C Code Uses Back Door Into MicroJava

From a Java programmer's perspective, the PicoJava 2 core creates an infinitely deep stack. In actuality, the first 64 elements are kept in a hardware stack on the chip, while stack elements 65–*n* spill over into external memory (or on-chip cache). From the point of view of a C compiler, PicoJava 2 has 64 general-purpose registers.

Although PicoJava 2's enhancements are designed to help non-Java code, all programs are still compiled to Java bytecodes, regardless of their original source language. Sun is developing a compiler for C and C++ that emits Java bytecode. The compiler uses the "extended" bytecodes on the 701 so programs can reference memory and access I/O devices. In perhaps the ultimate irony, such programs will not be portable because they rely on MicroJava-specific instructions rather than the nominally neutral bytecodes. It's also an incongruous reversal of the current paradigm of writing applications in Java to run on general-purpose processors.

Tape Out Next Year; Production Not As Clear

As the block diagram in Figure 3 shows, the 701 will include dual 16K caches, a memory controller, and a 32-bit PCI interface, making the chip a nearly self-contained Java engine. The 64-bit memory controller handles EDO DRAM, SDRAM, SRAM, flash memory, and ROM. The 701 also has a separate 8-bit bus for a boot PROM. The PCI interface runs at either 33 or 66 MHz and supports both master- and target-mode operation. Overall, the capabilities of the 701 are similar to those of Sun's other embedded processor, the MicroSparc-2ep (see MPR 5/6/96, p. 5), which the company currently uses in its JavaStation 1 and JavaEngine 1 platforms.

The chip has not taped out yet, but Sun's McGhan expects the 701 to reach that milestone sometime in 1Q98, with samples in 2Q98, and full production by 3Q98. These projections may be overly optimistic; past experience has taught us that the long march from tape out to production lasts closer to 12 months, not 6. If the 701 tapes out as planned early next year, it seems likely the chip won't enter production until 1999.

Technically, the company is not behind schedule for its original claim of Java chip availability before the end of

1997—at least, not if one of the licensees can deliver silicon early. Clearly, though, Sun has defaulted on any plans to deliver a chip of its own this year—and possibly next.

Figure 4 shows a die plot of the anticipated design. The device will measure a bit under 50 mm^2 and include about 2.8 million transistors, of which about 2 million go to the caches. Like most 0.25-micron microprocessors, the 701 will need dual power supplies: 2.5 V for the core, 3.3 V for I/O.

The 701 will be built in a 0.25-micron CMOS process, though the company would not identify its foundry partner. Historically, Sun has worked with Texas Instruments for most of its SPARC processors, so TI seems a likely partner.

Sun expects the chips will run at about 166 MHz, with a useful percentage yielding at 200 MHz. At the faster speed, the 701 should consume about 4 W, according to company estimates, and the chip will come packaged in a 316-contact plastic ball-grid array (PBGA). The MDR Cost Model yields an estimated manufacturing cost of $25 for the part.

Java Performance Beats Pentium II by 2×

Sun has been unusually tight-lipped about the performance of its Java chips. More than a year after announcing PicoJava 1, the company still has no verifiable performance metrics.

At the Forum, McGhan revealed that Sun has simulated the 701 running the CaffeineMark and Dhrystone benchmarks. The tests yielded a rating of 200 MIPS on Dhrystone 2.1 and 13,332 on Embedded CaffeineMark 3.0. The Dhrystone score is a little below average for a 200-MHz chip; the Embedded CaffeineMark score, however, is far higher than anything seen before.

Specifically, the highest rating recognized by Pendragon Software (the creator of CaffeineMark; *www.webfayre.com*) is 7,379 for a 266-MHz Pentium II system with 64M of RAM running Windows NT and Internet Explorer. (Embedded CaffeineMark eliminates three of the nine tests from the full CaffeineMark 3.0 suite, for a higher overall rating.)

Thus, Sun's simulations indicate the 701 executes Java almost twice as fast as the high-end Intel system, even though Pentium II has superscalar execution, a one-third faster clock rate, and larger caches. It's also an order of magnitude faster than StrongArm. At 233 MHz, the SA-110 scores just 1,105 on Embedded CaffeineMark 3.0, a disappointing 12× slower than the 701, even at a slightly faster clock speed.

Rockwell Still a Wildcard

MicroJava will also be up against JEM1, Rockwell's surprise entry to the Java field (see MPR 10/27/97, p. 10). The core of this come-from-behind player, which is based on an old avionics processor from the company's archives, is smaller than PicoJava and executes more bytecodes in hardware.

At just 50–60 MHz in 0.5-micron technology, JEM1 is not as fast as the 701. It would speed up considerably, though, if it were built in the same 0.25-micron process. Rockwell and Sun are said to be negotiating a distribution agreement for JEM1; it would be interesting if some of JEM1's design features appear in a future MicroJava processor. Rockwell has no benchmark information whatsoever for JEM1, and no price has been set, so the benefits of this chip are impossible to judge.

Dhrystone Performance Not As Good

To the extent that one trusts Dhrystone, the SA-110 and Pentium II both do much better than the MicroJava chip. The StrongArm chip rates at 268 Dhrystone MIPS, versus 200 MIPS for MicroJava. Pentium II scores range from 300 to 400 on Dhrystone at 233 MHz, putting it 1.5× to 2× ahead of the 701. Both results are measured on real systems.

On the other hand, Sun's results are simulated, and these benchmarks are too small to accurately reflect cache misses or memory latency. When the 701 begins shipping, its actual score may be different. At the same time, we can assume JIT compilers and other microprocessors will only get faster. By 3Q98, Pentium II should be shipping at 400 MHz (at least) in the same 0.25-micron process as the 701.

Still, Sun's results are impressive, even as a first-order approximation. To deliver performance in the same range as a Pentium II—much less beat it by 2×—with a chip that's half the size and (presumably) less expensive is no small feat. Factoring in the memory savings (because the 701 replaces a full Java interpreter or JIT compiler with a small emulation library), MicroJava 701 looks to be a dynamite bargain for customers determined to build Java-execution machines.

Waiting for the Demand

The question, of course, is exactly what kind of machines those might be. The hypothetical Java-based network computer has been slow to appear, perhaps because useful Java applications are not thick on the ground. Corel, for example,

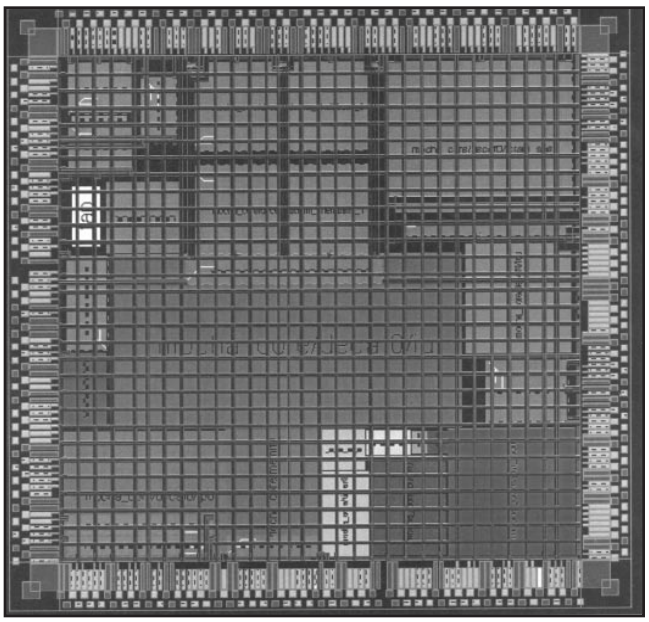

Figure 4. Die plot of proposed MicroJava 701 layout indicates the chip will measure about 50 mm^2 in a 0.25-micron CMOS process.

Price & Availability

Samples of Sun's MicroJava 701 are expected to be available in 2Q98, with production in 2H98. Pricing has not been announced. For more information, contact Sun Microelectronics (Mountain View, Calif.) at 650.960.1300 or visit *www.sun.com/microelectronics/java*.

canceled its high-profile attempt to port its WordPerfect Suite to Java. Without plentiful Java apps, Java systems are superfluous; without the Java systems, the apps may not come, so perhaps Java NCs are just an egg waiting for a chicken.

Even assuming demand for such a system, a Java chip is just one of many options. A general-purpose microprocessor leaves the door open to other languages, APIs, and operating systems besides Java, Java, and Java. By the time the 701 appears, it may be no faster than Pentium II on Java code, but with an infinitely smaller software base. Whereas general-purpose processors can execute anything the 701 can, the reverse condition does not hold true.

Although McGhan would quantify the expected selling price of the initial MicroJava chip only as "two digits," it's safe to assume that the 701 will be much less expensive than Pentium II, thus providing a price/performance advantage to developers for whom software availability is not important. But the same price/performance claim could be made of most other microprocessors as well.

MICHAEL MUSTACCHI

Harlan McGhan of Sun Microelectronics extols the virtues of the MicroJava 701 at the Forum.

Java Bytecode Sets Strategy

With the impending arrival of Java chips, embedded-software developers will soon be faced with three basic alternatives: write in C; write in Java; or write in Java and compile to bytecodes. In at least two of the three cases, Java chips do not make a compelling argument.

For the C-to-native scenario, the 701 makes very little sense. C programs written for the 701 are no more portable than other compiled programs, and the 701's performance (if Dhrystone is any indication) isn't particularly good.

The Java-to-native scenario also favors general-purpose microprocessors. Compiling Java source directly to the native instruction set of the target microprocessor bypasses the bytecode interface, skipping a costly intermediate step. Bytecode was intended for portability; if the software isn't being ported, it serves no purpose. This approach may sacrifice the putative portability of bytecode, but for embedded systems, real-time binary portability is rarely an issue.

Finally, there is the Java-to-bytecode scenario. If bytecode is the preferred delivery mechanism, the 701 will run it quickly and with minimal memory overhead. But in return, the chip exacts a toll in the use of non-Java software, operating systems, tools, and APIs.

Nearly any system can download and run the occasional Java applet. The advantage of the 701 is running those downloaded applets *quickly*. For "casual" use of bytecode, where performance is not all-important, a general-purpose processor can handle the task, and give better overall performance when it's not running bytecode.

In short, the 701 looks better the more bytecode the system has to run. For an all-bytecode system, the 701 is probably faster and cheaper than anything else. As the proportion of bytecode decreases, so does the advantage of a dedicated Java chip. MicroJava 701 and its kind make sense for some small fraction of the market (that does not now exist) that mainly relies on Java code and doesn't already have a microprocessor in it.

Java: Doing Whatever It Takes

It's no secret that Sun has focused its corporate efforts on the success of Java. Java hardware, software, education, and advertising are the company's featured products.

Strategically, Sun is more interested in Java itself than in Java chips specifically. McGhan was careful to point out that Java chips wouldn't and shouldn't replace Java interpreters or JIT compilers, but that they merely bring another option to the table. Java chips are "a complement, not a replacement" for software-only Java environments, he avowed.

At the level of the executive suite, Sun doesn't really care whether Java chips succeed or fail. Sun's ultimate goal is that Java prevail, through whatever means. The company is offering as many different methods of writing, disseminating, and executing Java code as it knows how. Whether customers execute Java applications using interpreters, JIT compilers, or specialized Java chips is irrelevant, as long as they use Java instead of Microsoft APIs.

Like a heavy shovel, Java has left a lasting impression on the minds of designers of both desktop and embedded systems. As companies wrestle with questions about whether they want Java, where to use Java, and how to execute Java, Sun has fanned the flames and encouraged experimentation. For the time being, the experimenters and the tire kickers have been using general-purpose microprocessors with Java interpreters, Java compilers, and Java-aware operating systems. For another 6–9 months, this will probably still be the case. Not until MicroJava 701, JEM1, or one of Sun's licensees' parts starts shipping will Java's early adopters be able to see for themselves whether a dedicated Java processor is valuable for their application. M