



SEARCH

- [Home](#)
- [Research](#)
- [Forums](#)
- [Design](#)
- [New Products](#)
- [Careers](#)
- [Blogs](#)
- [Contact](#)
- [Events](#)
- [Subscribe](#)
- [RSS](#)
- [Most Popular](#)
- [Introducing Trusted Sources
The Web's Independent Voices](#)

[EETimes](#)

Mobile Devices: JVM design key to portable arena

Print | Email | Reprints | RSS | Digital | SHARE

Spencer Horowitz, Consultant, Agile Systems Inc., San Jose, Calif.

[EETimes](#)

(07/23/2001 9:33 AM EDT)

Now that Nokia, Motorola and other leading manufacturers have publicly announced plans to ship more than 100 million Java-enabled mobile phones by 2004, the mass proliferation of Java technology from the desktop to the palmtop is clearly under way.

Java's widely recognized advantages in platform independence, ease of programming and maintenance, security and network compatibility are compelling developers of mobile applications to refine their Java strategies. The race is on to evaluate the best way to deliver Java execution in the next generation of phones, PDAs and other mobile applications.

The key to bringing Java to a mobile device is the Java Virtual Machine (JVM), the black box that translates Java programming instructions, known as byte codes, into the native instruction set of the underlying microprocessor. There are three architectural approaches to implementing a JVM, which can be handily summarized as interpretation, acceleration or dedication. To evaluate the competing benefits of these three approaches, we will review the performance, power consumption and total system implications of these Java architectures.

The first approach, interpretation, refers to conventional software interpretation, where run-anywhere Java byte codes are translated on-the-fly by a compiler to produce native instructions that run on the underlying processor. While perfectly adequate for Java execution on high-performance PC processors, software interpretation is simply too slow, too memory intensive and too power draining for battery-powered mobile applications. Any serious attempt to offer Java in an embedded or mobile application must rely on acceleration or dedication, two methods of implementing a JVM directly in hardware.

By implementing the JVM in hardware, accelerators and dedicated Java processors can increase Java performance five or 10 times over conventional software JVMs. Using this approach, we've been able to demonstrate a PDA that delivers a 320 x 240-pixel, 18-bit full-color display with four separate video-streaming panes running at 45 frames per second. By comparison, conventional PDAs, which use software JVMs running on native processors, are typically capable of only 120 x 160-pixel gray-scale graphics at a sluggish 2 frames/s. The performance advantage of direct hardware execution over software interpretation could not be more clear, so the real choice for mobile-system designers is between Java accelerators and dedicated Java processors. Several vendors currently offer hardware JVMs as either ICs, silicon intellectual property (IP) or both.

Java accelerators are extensions of existing processor designs that offer partial Java byte code execution in hardware, and partial execution in software. As such, accelerators are a compromise between conventional software JVMs and hardwired Java processors. Of the 228 Java byte codes, accelerators typically implement between 50 percent and 70 percent of Java instructions in hardware, relying upon the native processor to implement the

CAREER CENTER

Looking for a new job? [Open](#) | [Close](#)

Your company should be here.

EETimes
RELATED RESOURCE CENTER

Stay a step ahead with the most relevant offerings from
EETimes Related Resource Center

[Click Here](#)

EETimes Marketplace

[Autodesk Maya](#)
Use Autodesk Maya & Create Integrated 3d Models. Download Your Free Trial Today.

[Technical Papers](#)
[All White Papers](#) [»](#)

Education and Learning

Learn Now:

remaining byte codes using traditional software interpretation.

The architectural alternative to a Java accelerator is a dedicated Java processor. Dedicated Java processors are available that implement up to 99 percent of Java byte codes directly in hardware.

Obviously the percentage of Java byte codes that are implemented in hardware has a direct impact on Java performance. Accelerators rely on the statistical assumption that many Java byte codes are infrequently used. While the 80/20 rule holds true for the frequency of individual byte code occurrences, the frequency of an individual byte code must be carefully weighed against the performance penalty that results when these byte codes are encountered. With software interpretation requiring up to 10 times as many instruction cycles as hardwired execution, the 20 percent to 30 percent of Java byte codes that accelerators leave to software execution can cause noticeable performance degradation.

Complicating matters is how accelerator designers have selected which byte codes to execute in hardware, and which in software. Often this decision is based not on a detailed frequency-of-occurrence analysis, but on the complexity of the byte code itself. Important and complex instructions, such as CHECKCAST, MONITORENTER and the various INVOKE byte codes, are rarely implemented in hardware by Java accelerators.

Software interpretation for these critical and advanced byte codes can consume hundreds of extra instruction cycles on the host processor of a Java accelerator. If the Java application uses synchronized methods, the software execution penalty for INVOKE byte codes will further degrade performance. Dedicated Java processors, with their 99 percent Java hardware solution, enjoy clear performance advantages over their part-hardware, part-software Java accelerator cousins.

To enable system designers to further fine-tune system performance, Ajile's Java processor also permits frequently used subroutines to be defined as extended custom byte codes. Ajile was able to achieve its stunningly fast, full-color 320 x 240-pixel PDA demo by adding a special Bit-Blit byte code to the instruction set of its Java processor.

Accelerators

So dedicated Java processors offer better performance than accelerators, but nothing good comes without a price, and accelerators must eat up a lot less silicon than a fully dedicated Java processor.

Since accelerators are extensions of a host RISC processor that requires a minimum of 100,000 gates, a 12,000- or 25,000-gate accelerator is the tail that wags a 100,000-gate dog when it comes to power consumption. Every gate of the host processor-plus-accelerator combination is exercised. Systems with a dedicated Java coprocessor, on the other hand, can place the primary CPU in sleep mode while Java byte codes coolly execute. Power consumption scales linearly with gate count, so battery life can be extended by as much as 75 percent.

With comparable gate counts, the choice of adding an accelerator or a dedicated Java coprocessor should not center on die area if a primary RISC processor is required. Where support for legacy applications is not required, the power and die size burden of a 100,000-gate RISC processor will inevitably push system designers to choose a dedicated Java processor over accelerators. A rapidly growing library of Java applications-browsers, personal information managers, database programs and more-is making the 100 percent Java PDA, based on a single dedicated Java processor, a reality.

Add Your Comment:

spencerh

**MKS, a
TechOnline
partner,
[Solving the
Quality and
Compliance
Challenges for
Embedded
Software in
Medical Devices](#)**

▶ **Fundamental
Courses**
[Fundamentals of
Mixed-Signal
FPGAs](#)

▶ **QNX Tech
Paper**
[Persistent
Publish/Subscribe
for Embedded
Industrial
Applications](#)

▶ **Virtual
Conferences**
[Connected
Devices](#)

▶ **VirtuaLabs**
[Wireless
Communication -
Texas
Instruments
eZ430-RF2500](#)

Sponsored Products

Site Features

**Calendar Events
Conference
Coverage
Forums**

**Column
Archive
Special
Reports**