Why Your Next Embedded CPU Should Provide Hardware Support for Multitasking and Multiple Virtual Machines

David S. Hardin
Chief Technical Officer
aJile Systems, Inc.

http://www.ajile.com
Software Architectural Trends

• **Machine/System Independence**
  – Java Virtual Machine
  – Microsoft .NET Intermediate Language

• **Increased productivity through ‘friendlier’ programming languages**
  – Restrictions on pointer operations
  – Automatic memory management
  – Bounds checking, exception handling, etc.

• **Concurrency**
Software Architectural Trends (cont’d.)

• Late Binding
  – Dynamic class loading/unloading
  – Runtime compilation
• Componentized O-O Development
  – e.g., Java Beans
• Mobile Code Deployment
  – e.g., Java 2 Micro Edition (J2ME) “MIDlets”
• Desire for all of the above in networked embedded environments
Ada Technologies in the New Era

- Embedded, real-time, safety system emphasis
- Concurrency defined in the language
- Many language features to prevent programming errors
- Compilation to virtual machine instruction set possible
  - e.g. JGNAT -- compilation to Java VM
Microprocessor Design Trends

• Focused on extracting parallelism from legacy instruction sets
• Large amount of silicon area devoted to “bookkeeping” associated with out-of-order and/or speculative execution
• High power dissipation (Watts, not milliWatts)
• Average case performance increasing at the expense of predictability
Intel x86 Context Switch Latencies, Cycles

(source: QNX)
Intel x86 Context Switch Latencies, $\mu$sec

(source: QNX)
An alternative:

*Design CPU hardware that supports modern software development and deployment practices!*
Bucking the Hardware Design Trends

• High End
  – Transmeta Crusoe
    • “Code morphing” to VLIW native machine
    • Workload-based power management (LongRun)
  – Sun MAJC
    • Dynamic compilation to dual VLIW’s
    • Java thread-level parallelism with automatic rollback when needed
Bucking the Trends (cont’d.)

• **Low End**
  - Direct Execution Virtual Machine CPU’s
    • Virtual Machine instruction set is the “Actual Machine” instruction set
      • e.g., aJile aJ-100
    - “Hardware JIT’s”
      • e.g. JStar
  - Hardware support for RTOS primitives
    • e.g., aJile aJ-100
aJile’s Software-Friendly Hardware Design Approach

• Implement the Java Virtual Machine directly with simple, low-cost, low-power hardware
  – JVM bytecodes are the native instruction set
  – Low power implementation - < 1mW/MHz

• Provide enhanced security
  – Multiple JVM architecture -- safe execution of multiple Java applications
aJile’s Approach (cont’d.)

- **Aid embedded Java development**
  - Leverage Java APIs and commercial development environments
  - Utilize standard J2ME runtimes from Sun
  - Build tool produces reduced-size ROMable applications from standard Java class files

- **Enable entire system to be written in Java -- no C or assembly required!**
  - Including device drivers, interrupt handlers, trap handlers, runtime system, etc.
aJile’s Approach (cont’d.)

- Optimize key real-time operations via hardware support
  - Fastest real-time Java performance
    - < 500 nsec thread yield() @ 100 MHz
    - No RTOS required
- Provide leadership in the development of Real-Time and Embedded Java standards
Multiple Java Virtual Machine (MJM) Architecture

Features

- N independent JVM contexts
- Programmable deterministic time slicing
- Programmable memory protection
- Full Java threading and selectable GC
- Advanced power saving
aJile Tool Chain

- Symantec Visual Cafe'
- Borland JBuilder
- IBM VisualAge
- aJile Runtime Libraries (Class Files)
- aJile Runtime Libraries (Class Files)
- Application (Class Files)

Linker/ Application Builder (JEMBuilder)

Configuration Data

Load Module

Charade Debugger

Target System
JEMBuilder tool

• Accepts standard Java class files
• Performs transitive closure analysis; eliminates unneeded classes, methods, fields, constants
• Substitutes selected static methods with aJile extended bytecodes
  – No compiler changes needed to introduce extended bytecodes
• Creates ROMable executable
JEMBuilder (cont’d.)

- Hooks interrupts to Java methods
- Hooks traps to Java methods
- Manages multiple JVM builds
  - Allocation of hardware resources to JVM’s, etc.
- Friendly graphical user interface
- Written entirely in Java
aJile Runtime System

- aJile has recently licensed the Java 2 Micro Edition Connected Limited Device Configuration (J2ME CLDC)
- J2ME CLDC ported to aJ-PC104 development system
  - Working on conformance testing
- javax.realtime partially implemented
- Working on Mobile Information Device Profile (MIDP) implementation
Customizable Instruction Set

- **aJile architecture allows user-defined instructions**
- **Example: GPS signal processing**
  - GPS algorithms make extensive use of absolute value operations

```c
int abs(int x) {
  if (a < 0) {return -a;}
  else       {return a;}
}

void gps_sig_proc() {
  short i;
  for (i = 1; i< channels; i++) {
    snip..snip..snip
    if (abs(cr[i].igl) > abs(cr[i].qgl)) {
      spl = abs(cri[i].igl) + abs(cr[i].qgl)/2;
    }
    else {
      spl = abs(cri[i].igl)/2 + abs(cr[i].qgl);
    }
    snip..snip..snip
  }
}
```
Custom Instructions (cont’d.)

- **Java provides abs via a method**
  - java.lang.Math abs() method
  - No abs JVM instruction
  - Significant invoke/return overhead

- **aJile allows abs instruction to be added**
  - Tools substitute abs() method with custom abs instruction
  - aJile CPU: 1T abs

- **No compiler changes needed!**
The aJ-PC104 Single-Board Real-Time Java Computer

- JEM2 Java Processor
- Multiple JVM Unit
- 1 MB Flash
- 1 MB SRAM
- 32-bit Local Bus
- PC/104 Interface
- Interrupt Controller & Timers
- 8-bit GPIO
- UART (16C550)
- 10 Base T Ethernet
aJ-PC104 Evaluation System
aJ-100 Single-Chip JVM Microcontroller
Embedded CaffeineMark 3.0 Benchmark

Note: Longer bars are better

CM/MHz

SA110   R4600   Pentium   aJ-100

(interpreted)
Dual-Thread Benchmarks

Note: Shorter bars are better

Time (μsec)

<table>
<thead>
<tr>
<th>Yield</th>
<th>Wait/Notify</th>
<th>Wait/NotifyAll</th>
</tr>
</thead>
<tbody>
<tr>
<td>aJ-100 100 MHz</td>
<td>Pentium II 300 MHz</td>
<td></td>
</tr>
</tbody>
</table>

Copyright © 2000 aJile Systems, Inc. All rights reserved.
aJile vs. ARM7 Benchmarking

(Shorter bar is better)

- **RAM Size** (K bytes)
- **ROM Size** (K bytes)
- **Act. new thrd (usec)**
- **Res. curr. thrd (usec)**
- **Int. Latency (usec)**
- **Idle Time (10 ms/sec)**

<table>
<thead>
<tr>
<th></th>
<th>ARM7TDMI</th>
<th>aJile</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAM Size (K bytes)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ROM Size (K bytes)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Act. new thrd (usec)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Res. curr. thrd (usec)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Int. Latency (usec)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Idle Time (10 ms/sec)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **Benchmark code**
  - 1 Hz, 10 Hz, and idle threads
  - 100 Hz system tick
- **ARM7 configuration**
  - RTXC RTOS - C and assembly
  - C application source code
  - 16-Bit ARM thumb ISA
- **aJile configuration**
  - Hardware thread support (no RTOS)
  - Java source code
- **Benchmarking performed by Conexant personnel**
Example aJile Technology Applications

- “Gateway” devices
- “Smart Sensor” applications
- Java acceleration for mobile devices
- Real-time control systems
- Java Card with full JVM functionality
Gateway Devices

Robot

Factory floor control network

Control JVM  Monitor JVM

Enterprise network

• Single aJile CPU
  • Secure
  • Low cost
  • Low power
Java Acceleration for Mobile Devices

- Network Interface
- Display
- Input Device
- Legacy CPU core
- Java “Midlets”
- Memory
- aJile Java Acceleration core
aJile Architecture Summary

- Direct execution of JVM instruction set
  - No need for JVM interpreter or JIT compiler
- RTOS on chip
  - Low cost, low memory, high performance
- Multiple JVMs
  - Multiple applications, enhanced security
- System programming accomplished completely in high-level O-O language
  - Includes device drivers, interrupt handlers, etc.