第 33 卷 第 6 期 计 算 机 工 程 2007 年 3 月
Vol.33 No.6 Computer Engineering March 2007
·工程应用技术与实现· 文章编号：1000—3428(2007)06—0233—03 文献标识码：A 中图分类号：TP303

Java 虚拟机的硬件实现研究

刘 丹[1,2]，孟令奎[1]
（1. 武汉大学遥感信息工程学院，武汉 430079；2. 华中师范大学信息技术系，武汉 430079）

摘 要：Java 语言的平台无关性使其正在进入嵌入式系统领域，但 Java 的性能问题一直是 Java 应用关注的焦点。JIT 技术的运用在一定程度上提高了 Java 的执行速度，但在对实时性要求高和存储受限的嵌入式等系统的运用中仍然难以满足其要求。文章提出了一种基于硬件的解决方法——Java 处理器，它能直接执行 Java 字节码，从而提高 Java 的性能，以用于实时及嵌入式系统。

关键词：Java 虚拟机；状态寄存器；缓存；流水线

Research on Hardware Implementation of Java Virtual Machine

LIU Dan[1,2], MENG Lingkui[1]
(1. School of Remote Sensing and Information Engineering, Wuhan University, Wuhan 430079;
2. Department of Information Technology, Huazhong Normal University, Wuhan 430079)

【Abstract】 In embedded system fields, at present, C and compile language which are based on real time operating system still have the high place. As technology's development, Java is stepping into embedded system fields, for example mobile phone, PDA and set-top box etc. However, the problem of Java's performance is always the focus of Java applications because JVM makes Java application execute slowly and need large memory. This paper promotes a Java processor based on hardware which can directly execute Java bytecode and elevate the performance greatly. This method is applicable to real-time and embedded systems.

【Key words】 JVM; PSW; Cache; Pipeline

1 概述

目前在嵌入式系统领域中，基于实时操作系统之上的 C 和汇编语言仍处于主要地位。Java 的平台无关性、安全性、多线程以及垃圾回收机制使其成为互联网通信中的关键技术。随着技术的不断发展，Java 正逐渐进入嵌入式系统领域，如移动电话、PDA、机顶盒及汽车通信系统等。基于 Java 的应用 一般是编译成 Java 字节码然后采用软件方法 JVM 实现，即通过 Java 虚拟机引擎解释 Java 字节码。JVM 使程序的执行缓慢，特别是对于内存容量有限、工作环境苛刻及实时性要求严格的嵌入式系统更是不容忽视的问题。为提高 Java 的性能，目前普遍采用两种方法解决该问题：

(1)软件方法：编译Java字节码为目标处理器的本地码，如SunSoft，Symantec[6]的JIT编译技术。JIT技术将字节码编译成目标机的本地二进制代码，从而在执行时不用解释字节码，提高速度。但JIT技术使原代码量增加 2~3 倍，这对于内存受限的嵌入式系统是不合适的；同时，JIT编译器的移植的工作量非常大，所以JIT编译技术并不适合于嵌入式系统中。

(2)硬件方法：Java 协处理器，将 Java 字节码转换成普通的 CPU 指令从而提高运行速度。Java 协处理器直接在芯片上实现 JVM，这不仅避免解释字节码的开销，而且可以支持 Java 运行时的特征，能比目前通用处理器更好地支持堆栈处理、多线程、垃圾回收和句柄处理，甚至能有效利用目前处理器技术中的缓存、流水线处理、分支预测等优点。

本文主要讨论 Java 虚拟机的硬件实现方法，并提出一种 JVM 的硬件架构——Java 处理器，该处理器能直接执行 Java 字节码，通过消除解释开销时间提高 Java 指令的执行时间。

2 相关工作研究

Java最初的应用领域是在嵌入式系统中，20 世纪 90 年代初源于Sun公司的项目Oak——一种无线PDA的编程工具，后来逐渐变成互联网上重要的语言。在嵌入式应用领域，Sun 定义了J2ME(Java Micro Edition)，该Java版本目前已经在移动设备领域得到广泛的应用。基于硬件的实现可采用Java协处理器方式实现Java独立指令集。1998 年，Sun公司发布 MircroJava、PicoJava[2]和UltraJava[4]芯片，其中PicoJava使用硬件直接执行大部分JVM指令并通过Microcode和Software Traps实现复杂指令，采用监视Cache支持同步和特殊指令实现对象应用的垃圾回收，但仅是原形芯片。ARM公司推出的 Jazelle技术通过增加ARM指令集使ARM处理器为Java虚拟机提供优化的执行能力。相关的研究还包括嵌入式硬件的JVM 加速器，如inSilicon公司的JVXtreme[5]。

3 JVM 结构

JVM 是一种抽象的计算机，规定字节码指令的格式及执行方式。JVM 包括 3 个方面：

(1)JVM虚拟机规范[1,3]；

(2)JVM 的具体实现；

(3)Java 运行实例。

JVM 规范定义了 Java 的指令集、数据类型以及 Java 执行对象即 class 文件的格式，保证了 Java 语言的跨平台性和安全性，但对 JVM 的具体实现没有规定。作为 Java 技术的核心 JVM，其实现方式将直接关系 Java 的执行性能。

作者简介：刘 丹(1978－)，男，博士生、讲师，主研方向：地理信息系统，计算机系统结构；孟令奎，教授、博导

收稿日期： 2006-03-30 **E-mail：**sandyliu_1978@hotmail.com

根据 JVM 规范，JVM 实例的描述包括子系统、内存区域、数据类型和相应指令集，其结构如图 1 所示。JVM 的 class loader subsystem 机制作用是装载类及接口。Execution engine 装载类的指令。当 JVM 实例执行时，内存需存储字节码、对象实例、方法参数、返回值、本地变量及中间计算结果。JVM 实例包括一个 method area 和一个 heap，二者被所有线程共享。新的线程有单独的 pc register 和 Java stack。Java stack 存储线程方法调用的运行状态，本地方法调用的状态采用单独的方法——native method stacks。JVM 没有使用寄存器存储中间计算值，而是采用 Java stack。由于 Java 的执行是通过 JVM 解释 class 文件的字节码，然后调用所在操作系统和硬件的本地方法，其速度和效率受到很大限制，因此不能很好地应用于对实时性要求高和存储空间非常有限的嵌入式等系统中，所以可考虑采用硬件的方式直接实现 JVM 的指令集，从而提高执行效率。

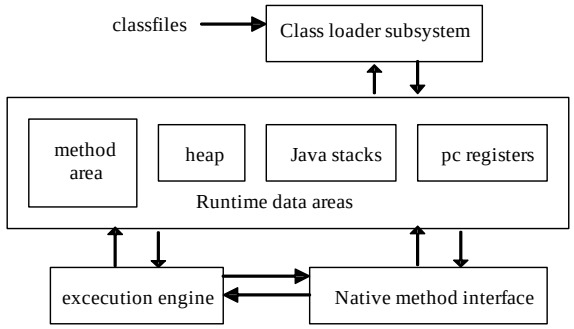

图 1 JVM 结构

4 JVM 硬件结构及实现

对于 JVM 的硬件实现，主要有以下几个问题需要解决：（1）为保持同现有 CPU 结构兼容，考虑主要在现有 CPU 指令集基础上增加 Java 字节码指令，保留原有指令集，从而使 CPU 能同时运行原来的操作系统、应用软件及 Java 应用程序。（2）在状态处理机寄存器(PSW)中设置相应的位以区分 CPU 的普通状态和 Java 运行状态。由于需要对 JVM 的堆、PC 寄存器等作处理，因此考虑同时设置单独的特殊寄存器。（3）Java 指令的执行和普通指令一样可以接受中断、异常并返回，但需要作相应的处理。（4）内存管理对于嵌入式系统是非常重要的，所以要求对 Java 的内存管理部分（如应尽量避免使用垃圾回收机制）作特殊的处理。下面将对以上问题分别进行讨论。

4.1 指令集

JVM 指令基于栈结构，所有指令包括 1B 操作码及 0 或多字节操作数。JVM 包含 10 种类型指令：

(1)Load/Store Instructions
(2)Arithmetic Instructions
(3)Type Conversion Instructions
(4)Object Creation and Manipulation
(5)Operand Stack Management Instructions
(6)Control Transfer Instructions
(7)Method Invocation and Return Instructions
(8)Throwing Exceptions
(9)Finally Instruction
(10)Synchronization Instructions

对于以上类型指令，大部分简单的指令有对应的硬件指令，对于复杂的指令（如指令 new 包含很多步的操作）可以由简单的指令组合实现。同时，增加单独一条指令 JEE（JEE 指令格式如图 2 所示）支持进入/退出 Java 状态，当指令 JEE 的 Condition 条件满足时，程序将选择进入/退出 Java 状态，并修改 PSW(状态寄存器)的 T 状态位。

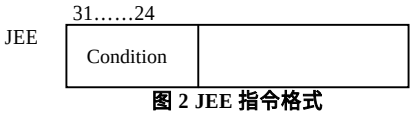

图 2 JEE 指令格式

为区分 CPU 当前的运行状态，在 PSW 中增加一位 T，当 T=1 时，表示系统处于 Java 运行状态，否则 T=0 表示处于普通状态。其指令格式如图 3 所示。

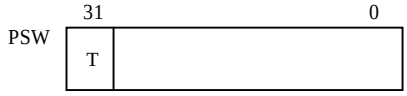

图 3 PSW 增加 T 状态位

以上简单指令，可以直接通过硬件执行如下：

(1)constant loads, (iconst_0, dconst_0, ...)
(2)variable loads/stores, (iload, dstore, ...)
(3)array load/stores (iaload, dastore, ...)
(4)integer data operations (iadd, isub, i2b, ...)
(5)branches (ifeq, icmp_ifeq, ...)
(6)quick constant pool loads (ldc_quick, ...)
(7)quick static/field operations (getfield_quick, ...)

通过其它指令模拟执行的复杂指令如下：

(1)floating point (ddiv, dadd, dmul, ...)
(2)integer division (idiv, irem, ldiv, lrem)
(3)switch (tableswitch, lookupswitch)
(4)invoke (invokevirtual, invokestatic, ...)
(5)return (ireturn, return, ...)
(6)new (new, newarray, ...)
(7)unresolved ldc (ldc, ldc_w, ldc2_w)
(8)unresolved field/static (getstatic, putfield, ...)

以上复杂的指令是通过组合简单的指令完成，从而不需要用硬件实现 JVM 所有的指令集合，这对于降低硬件的开发难度是非常重要的。

4.2 特殊寄存器

在 Java 运行状态下，为其设置特殊寄存器，提高 CPU 执行 Java 的性能，寄存器如下：

(1)R0 变量 0，表示'this'指针；
(2)R1 Java 栈指针；
(3)R2 Java Constant Pool 指针；
(4)R3 机器 Stack 指针；
(5)R4 Java PC；

利用 Java 栈指针和机器 Stack 指针，使堆栈的溢出可通过硬件直接检测。除以上寄存器，可另外设置寄存器如 R5-R8 用于缓存 Java 表达式堆栈，在多数情况下，大部分 Java 应用的堆栈深度比较小，采用寄存器的方式可以在函数调用返回时减少堆栈的开销。特殊寄存器对于提高 JVM 的性能十分重要，同时也降低了程序访问内存的时间。

4.3 中断及异常处理

中断处理通常同设备驱动程序的低级编程联系在一起，在非抢占式操作系统(如 Linux)中，进程调度发生在系统调用返回、中断处理或异常返回到用户空间的前夕，为避免复杂性及保持原应用的兼容性，Java 指令的执行对中断不作特殊处理，即其中断及异常处理机制与原系统相同，具体执行过程如图 4 所示。

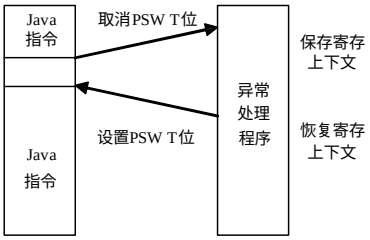

图 4 Java 中断及异常处理过程

与 Java 执行状态相关的信息保存在机器状态寄存器 PSW 的 T 位中，在 Java 中断的入口处保存 PSW 并在中断的出口处恢复 PSW,这种处理方式可以使 JVM 处理器同原 CPU 保持兼容，降低硬件开发难度。

4.4 Cache 及流水线处理

处理器设置指令及数据 Cache 为 32KB。Java 执行采用堆栈，为提高性能及并行执行能力，设置 64 入口寄存器文件的堆栈 Cache。本系统使用 6 段的流水线，即从指令 Cache 中取指令的取指、指令译码、从堆栈 Cache 读取操作数的寄存器、指令执行、从数据 Cache 读取数据及结果写回堆栈 Cache。Cache 部分结构如图 5 所示，Cache 分为指令 Cache 和数据 Cache，指令解码、ALU 等用于下面 JVM 指令流水线的执行，每条 JVM 指令将占用一个指令周期的运行时间，采用流水线的方式可提高指令执行效率。Java 处理器的流水线执行过程如图 6 所示。

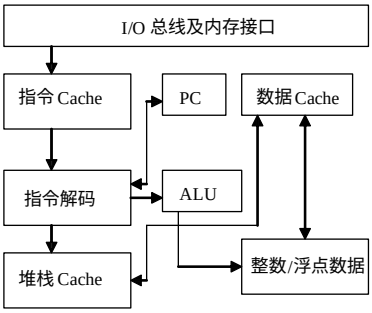

图 5 Java 处理器 Cache

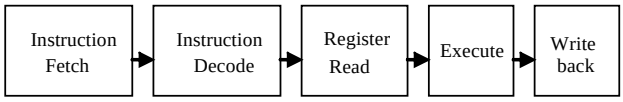

图 6 指令流水线处理过程

在第 1 段流水线中，JVM 的指令被取出，第 2 段流水线进行指令译码和生成地址的操作，第 3 段包括可能的寄存器读操作，然后是指令的执行，最后是指令的写回操作。

5 进程调度

在嵌入式系统中，进程的调度和上下文切换是非常重要的任务，进程的切换主要包括以下两个步骤：调度，进程调度的选择；切换，进程上下文的切换。

(1)调度：在本系统中采用简单的固定优先级抢先式调度方式，进程的优先级是固定不变的，由于系统主要针对嵌入式系统，因此优先级在运行前采用 Deadline-Monotonic 优先级的分配策略。

(2)切换：进程上下文的切换时间依赖与进程堆栈的大小，当堆栈位于内存时，只有少数几个寄存器（如程序计数器和栈指针）需要保存和恢复，但由于栈属于经常被 JVM 操作的内存区域，因此栈应被认为是执行上下文的一部分而在上下文切换时被保存和恢复。本系统利用特殊寄存器可以在很大程度上减少栈操作的开销。

6 结论和以后的工作

本文提出一种 Java 处理器架构，通过直接执行 JVM 字节码，能在很大程度上提高 Java 性能。其实现可用 FPGA 或 ASIC/SoC(System-on-chip)生成专用 Java 处理器，也可为保持兼容在原 CPU 中加入单独的 Java 指令集。

通过 Cache 及流水线方式可以提高 Java 处理器性能，但由于 Java 操作采用不同于普通的 Register-Register/Register-Memory 方式，因此在很大程度上影响其并行执行能力，在后期的研究中，可考虑对其堆栈结构进行优化。

参考文献

1 Lindholm T, Yellin F. The Java Virtual Machine Specification[M]. 2th ed. Addison Wesley, 2001.

2 JOP-Java Optimized Processor[Z]. 2004. http://www.jopdesign.com.

3 JVM[Z]. 2000. http://mail.phys-iasi.ro/Library/Computing/JVM/toc.html.

4 Radhakrishnan R, Rubio J, John L. Characterization of Java Applications at the Bytecode Level and UltraSPARC-II Machine Code Level[C]//Proceedings of International Conference on Computer Design. 2000-10.

5 Berkeley Architecture[Z]. 2001. http://www.eecs.berkeley.edu/Research/Areas/CS/ARC/.

6 Symantec Café[Z]. 2002. www.symantec.com/cafe/index product.html.

(上接第 227 页)

参考文献

1 Aleksic P S, Katsaggelos A K. Speech-to-Video Synthesis Using MPEG-4 Compliant Visual Features[J]. IEEE Trans. on Circuits and Systems for Video Technology, 2004, 14(5).

2 Ostermann J. Animation of Synthetic Faces in MPEG-4[EB/OL]. 1998. http://www.research.att.com/projects/AnimatedHead/pimages/com-panim3.pdf.

3 陈益强, 高　文, 王兆其, 等. 基于机器学习的语音驱动人脸动画方法[J]. 软件学报, 2003, 14(2): 215-221.

4 叶　静, 董兰芳, 王　洵. 用于语音动画合成的语音特征提取和聚类技术[J]. 微型机与应用, 2004, 23(8): 47-49.

5 飞思科技产品研发中心. Matlab 6.5 辅助小波分析与应用[M]. 北京: 电子工业出版社, 2003.

6 胡　航. 语音信号处理[M]. 哈尔滨: 哈尔滨工业大学出版社, 2000.

7 易克初, 付　强. 语音信号处理[M]. 北京: 国防工业出版社, 2000.

8 韩纪庆, 张　磊, 郑铁然, 等. 语音信号处理[M]. 北京: 清华大学出版社, 2004.

9 飞思科技产品研发中心. Matlab 7 辅助信号处理技术与应用[M]. 北京: 电子工业出版社, 2005.

10 Daubechies I. The Wavelet Transform, Time-frequency Location and Signal Analysis[J]. IEEE Trans. on Information Theory, 1990, 36(5): 961-1005.