文章编号:1001 - 9081(2006)05 - 1152 - 03

面向 Linux NC 的 Java 虚拟机的性能优化

杨丽洁
(中国科学技术大学 软件学院,安徽 合肥 230026)
(muyi6081@mail.ustc.edu.cn)

摘 要:针对目前 Linux NC 中 Java 虚拟机运行 Java 程序存在的性能问题。提出了一种优化方案,以直接线索式解释器为基础优化技术,并设计 3 项扩展优化点突出优化效果。旨在兼顾 Linux NC 现有的硬件和软件条件,有效地提升 Java 虚拟机运行效率,同时保证较低的 CPU 和内存成本。

关键词:Kaffe;Linux;网络计算机;Java 虚拟机;性能优化
中图分类号: TP311.5 **文献标识码**:A

Performance optimization of Java virtual machine in Linux NC

YANG Li-jie
(*School of Software Engineering, University of Science and Technology of China, Hefei Anhui* 230026, *China*)

Abstract: At present, the performance of Java virtual machine in Linux Network Computer(NC) is relatively low. An optimization scheme was proposed, which based on the direct threaded interpreter, and three extended techniques were designed to stand out the effect of optimization. In this way, considered the conditions of hardware and software, the executing efficiency of Java VM can be upgraded. Besides, this project can also reduce the cost of CPU and memory.

Key words: Kaffe; Linux; NC; Java virtual machine; performance optimization

0 引言

网络计算机(Network Computer,NC)在金融、保险、教育、电信等行业的市场表现良好,已创造出相当可观的经济效益和社会价值,具有很大的发展潜力。由支持 Java 的三层构架实现的 NC 的软件设计如图 1 所示,这种结构已成为 NC 发展的必然趋势。

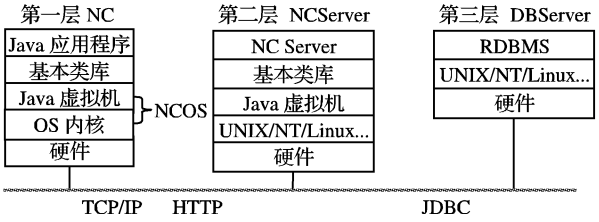

图 1 基于 Java 的三层构架

NC 系统软件包括两大部分:运行在 NC 上的 NCOS 和运行在应用服务器上的 NCServer。由图 1 中可以看出,Java 虚拟机(Java Virtual Machine, JVM)是两大部分的重要组件,它的设计方案和实现技术将影响 Java 应用程序执行的具体效果以及 NC 中其他服务的性能表现。

1 NC 中 JVM 的性能问题

近来,随着多种性能优化技术在 JVM 中的应用,Java 的运行效率有了很大程度的提高,减轻了 Java 技术在桌面操作系统和服务器中的普及压力。但嵌入式领域的 JVM 性能的研发工作却滞留在起步阶段。

Linux NC 应用的是桌面 OS 的 JVM。虽然这种 JVM 的优化技术大大提高了 Java 的执行性能,但嵌入式设备 NC 的计算及存储能力却难以负担这种 JVM 的技术对 CPU、内存等条件的较高要求。目前,国外基于 JVM 解释执行的优化技术取得了较多研究成果,这些成果已应用在移动通讯设备的 JVM 中。而在 NC 的 JVM 中类似的应用很少。

因此,对 NC 的 JVM 的性能优化技术进行研究将具有重要的价值。

2 Linux NC 中 JVM 性能优化方案

Linux 继 Windows CE 之后,成为第二大应用于 NC 的主流操作系统。因此,优化工作将依托 Linux 平台对 NC 中的 JVM 进行优化。

2.1 优化背景

2.1.1 优化对象——Kaffe 虚拟机

源代码开放的软件包 Kaffe 是一个优秀的 Java 语言环境。优化方案选择 Kaffe 作为优化对象,主要基于以下 4 个原因:

1) Kaffe 是基于类 UNIX 系统上开发的,特别是 Linux 和 Free BSD。因此,把 Kaffe 移植到支持 POSIX 原语的体系结构比其他体系结构容易;

2) Kaffe 是一个完整的遵从 Personal Java 1.1 规范的 Java 语言环境,可以应用于各种因特网设备、嵌入式系统;

3) Kaffe 的解释器采用 switch - case 模式,性能相对较低。Kaffe 同时支持即时编译器,但效果不好;

4) Kaffe 基于模块实现,具有伸缩性和高效性。

2.1.2 Linux NC 的硬件条件

为满足一般用户需求,Linux 终端设备通过较低的硬件参数来保证 NC 较 PC 的某些成本优势。Linux NC 现有的硬件配置为:主频较低的 CPU、容量有限的可选的 FlashROM、CF 卡和内存。在这种硬件条件下,Kaffe 虚拟机的优化方案要充分考虑终端机的存储器及 CPU 的条件对 JVM 执行效果的限制。

2.2 优化技术分析

对于 JVM 的优化主要有两大类技术:

1) 编译执行技术及优化

收稿日期:2005 - 11 - 16;**修订日期**:2006 - 02 - 24

作者简介:杨丽洁(1981 -),女(满族),河北围场人,硕士研究生,主要研究方向:嵌入式系统.

主流的执行技术有:即时编译技术、自适应优化技术及提前编译技术。它们各自在 Java 字节码运行前或运行时,不同程度的将字节码编译为本地机器码,从而缩短执行时间。

对应的优化技术有:方法内嵌、消除公共子表达式等。

2) 解释执行技术及优化

该技术的原理是:JVM 经装载、链接及初始化类等步骤,查找指定入口方法,解释器循环取出字节码,并按字节码处理程序执行指令,直到 Java 程序退出。

对应的优化技术有:线索式解释器、直接线索式解释器及内嵌线索式解释器[4]。它们将解释器对相邻字节码的解释执行过程通过一定的方式直接联系在一起。改变解释器中原有的 switch - case 执行模式。

2.3　优化技术选择

基于上述的优化背景及各种优化技术的特点,作者分析得出编译执行技术与 NC 的不适用性:

1) 在 800MHz 工作频率,32 位的总线宽度的 CPU 下,编译执行的 Java 程序会呈现较明显的停滞现象;

2) NC 的 CPU Cache 容量难以容纳全部的核心代码,使得 CPU 访问内存的几率增加,JVM 的执行效率降低;

3) 编译产生的机器代码量是原字节码的几倍到几十倍,NC 的内存和 CPU Cache 很难承受。

相对而言,解释执行技术则更适合终端设备:

1) 解释执行的 JVM 占用较小的 ROM;

2) 字节码占用较小内存空间,减轻了对 Cache 的压力;

3) 解释方式运行的 JVM 的核心代码量较小,增加了在指令 Cache 中的比例。

然而,单纯的解释执行机制内部仍然存在一些缺陷。鉴于目前 Linux 终端在市场上的应用主要集中在金融、保险、电信等行业,对字节码执行的性能要求不是很高。因此,对字节码解释执行的技术加以优化后,虚拟机的表现一定可以达到相对满意的水平。

2.4　优化方案

依据以上对优化背景和优化技术的分析,及原有 Linux NC 应用的编译执行方式的 JVM 在实际应用中表现不佳的现状,另外,解释技术的优化技术在嵌入式移动通讯领域的应用良好。因此,作者拟采用解释执行技术及其优化手段设计优化方案。

2.4.1　基础优化技术

由表 1 可以看出,选择直接线索式解释器(Direct Threaded Interpreter,DTI)优化 switch - case 执行模式的解释器是相对合理的,因为这种选择兼顾考虑了 NC 的内存条件。

表 1　三种优化方案性能提高幅度对照表

	较 switch - case 模式性能提高幅度	特点
线索式解释器	7% ~10%	提高幅度有限
内嵌线索式解释器	150% ~300%	内存代价较大
直接线索式解释器	40% ~100%	内存代价较小

DTI 的工作机制:转化函数对应一个 translated code 数组,函数负责按顺序读入字节码指令,查找其解释程序入口地址,将地址保存在对应的数组中。即原来的操作码对应转化为解释程序的标号地址。如图 2 所示。

为了节约 translated code 数组的空间,方案对数组进行了合理的压缩:

1) 合并操作数

在 32 位 CPU 中,标号地址长度占用 4 个字节的存储器。操作码后的操作数长度从 0 字节到数十字节不等。原来 16 位和 32 位的操作数分别占用 2 个字节,2 个字节码单元和 4 个字节,4 个字节码单元。合并后,它们均占用 1 个 translated code 数组单元。从而节省数组空间。

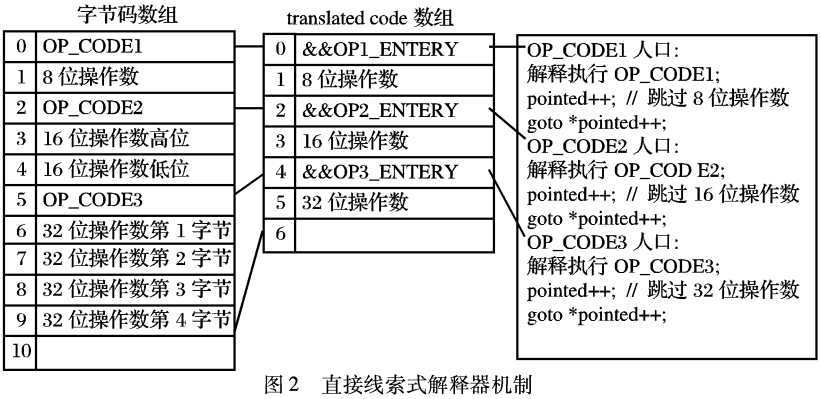

图 2　直接线索式解释器机制

2) 跳转地址的处理

操作数的合并引起了操作码对应的偏移量的变化,因此,需要更新绝对和相对跳转指令的目标地址。方案引入 tcbc_offset 数组,令其保存 translated code 数组相对于原字节码数组的偏移量的差值(带符号)。当修改跳转目标地址时,利用该数组的值分别计算绝对和相对跳转指令的新目标地址。

2.4.2　扩展优化点

为了提高 DTI 在 NC 中提升 JVM 性能的程度,作者额外设计了 3 个优化途径。

1) 引入伪指令

目前《Java 虚拟机规范》只定义了 202 条指令,其余的可由用户自定义,即伪指令[9]。

· 用指定数据类型的伪指令替代不指定操作数类型的指令。省去遍历各种类型常量池的操作。比如:ldc 指令,把常量池中的项压入 Java 栈。方案的伪指令 intldc,floatldc,stringldc 指定具体的数据类型。在转化字节码的过程中,根据常量的数据类型,对应到伪指令,直接指向某个类型的常量池。提高了执行效率,减小了字节码尺寸。

· 将小于等于 4 个字节的常量值放入 translated code 数组中操作数的位置,省去查找常量池的操作。

2) 合并字节码指令序列

在字节码转化过程中,可以跟踪记录每个 Java 方法被调用的次数,虚拟机初始化时,首先设定一个默认的调用频率指标,对调用频率高于默认值,且最高的 Java 方法进行连续的字节码指令合并。将该 Java 方法的字节码指令对应的处理程序合并到一个标号地址下,然后修改 translated code 数组中该 Java 方法的首标号地址,使其指向新的合并后的地址,并依次转移操作数,处理操作数指针,以保证操作码和操作数之间的对应关系,最后释放剩余的 translated code 数组。

该处理方式既提高了解释执行的速度,同时再次获得压缩 translated code 数组的效果。

3) 管理转化码数组空间

字节码转化为 translated code 需要付出原字节码大小的 3 ~4倍的存储空间的代价。这相对于内嵌线索式解释器,内存开销已经减少很多,但是对于内存空间有限的 Linux NC 来

说,对这块内存空间的经济、高效的管理仍是个关键的问题。

方案在方法中设置 accessCount 变量来管理 translated code 内存,其工作机制是:

· 开辟一块 translated code 专用的内存——Translated Code Block;

· 采用"即用即转化"的原则对 Java 方法进行转化;

· 变量 accessCount 记录 Java 方法被调用的次数;

· 若当前 translated code 的内存量不足以存放新的转化码时,查找方法表中已转化的每个 Java 方法的 accessCount 值,选择该值最小的 Java 方法所对应的 translated code 进行内存空间释放。重复该过程直到能容纳新的转化码为止。

"即用即转化"的原则根据访问频率进行空间淘汰的方法满足了 translated code 空间限制的条件。

2.5 优化方案的实现

2.5.1 调整 Kaffe 的模块

将 Kaffe 原运行时环境中的即时编译器替换为字节码转化器,它与字节码解释器联合其他子系统共同完成解释执行任务。

2.5.2 优化后 Kaffe 的执行流程

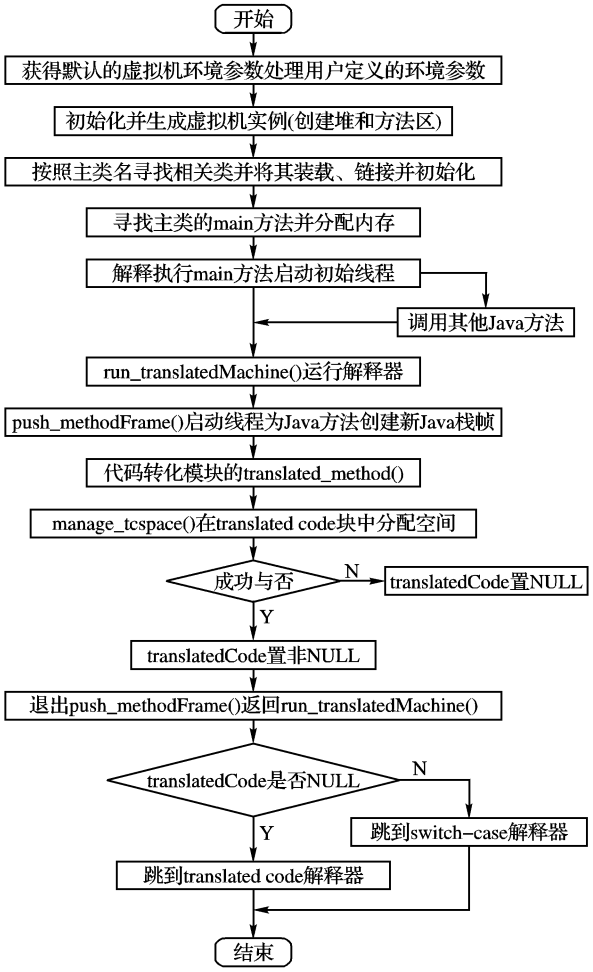

图3 优化后 Kaffe 的执行流程

方案包含两个重要的模块:字节码解释模块和字节码转化模块。解释模块的主体函数同时实现 translated code 解释器和 switch - case 解释器两种执行方式,在字节码转化过程失败的情况下还用原来的 switch - case 模式执行。在转化过程中,转化函数对应的转化码空间是否分配成功是转化工作的制约因素。空间分配成功进入实际的转化过程,此次方案设计 3 个阶段实现基础优化技术和扩展优化途径:第 1 阶段实现 DTI 的转化及操作数的合并;第 2 阶段实现合并字节码序列;第 3 阶段处理前两个阶段引起的操作码偏移量的变化,即跳转指令的更新。如图 3 所示。

3 优化方案的评估

评估工作主要包含两个部分:

· 使用 J2SE 提供的 Benchmark 集合进行测试。应用其中的 specJVM98 和 Java Grande Forum Benchmark suite 部分;

· 自行设计基于 Linux NC 的 Java 评估系统。两个目标:获取 JVM 占用系统资源的数据(CPU 和内存数据);鉴于 NC 产品的需求,测试优化方案对 Java 外设支持效果是否存在干扰因素。

完成评估测试工作后,综合各种测试结果的数据进行比较分析,Benchmark 集合中的 Java 测试程序在执行效率上都有不同程度的提高,同时,方案给系统带来的 CPU 和内存代价处于合理水平,即性能提高的幅度与系统代价成一定比例。且原有的 Java 支持外设的功能没有受到明显干扰,即终端产品的技术更新保证了产品原有功能和特性的完整性。由此可见,本文的 JVM 性能优化方案对于 Linux NC 是合理、有效的。

4 结语

本文的优化方案主要有以下 3 个特点:

1) 显著提高了 Kaffe 在 Linux NC 上的执行效率;

2) 在技术成本方面,系统资源占用率相对较低,并保证 NC 产品在 Java 技术方面原有的支持特性;

3) 方案的设计基于模块化结构,具有清晰、完整的特点,且易于实现和扩展。

该方案虽然已将 NC 的 JVM 性能提升到一定水平,但还存在一些值得深入改进的地方。如:方案的转化字节码的设计本质上属于编译执行方式,因此可应用基于编译方式的优化技术;对代码量较小的高频 Java 方法进行内嵌,以减小方法调用的开销。

参考文献:

[1] VENNERS B. Inside the Java Virtual Machine[M]. 2nd edition. New York: McCraw-Hill, 2000. 27 - 98.

[2] LINDOLM T, YELLIN F. The Java Virual Machine Specification [M]. 2nd edition. Boston, MA: Addison Wesley, 1999. 21 - 104.

[3] DOYLE JK, MOSS JEB, HOSKINGS AL. When are Bytecodes Faster than Direct Execution[DB/OL]. ftp://ftp.cs.umass.edu/pub/osl/papers/oopsla97a.ps.gz, 1997.

[4] IUMARTA I, RICCARDI F. Optimizing direct threaded code by selective inlining[J]. ACM SIGPLAN Notices archive, 1998, 33(5).

[5] ADL-TABATABAI AR, CIERNAK M, LUEH GY, *et al*. Fast, effective code generation in a justin-time Java compiler[J]. Conference on Programming Language Design and Implementation, 1998, 33(6).

[6] GREGG D, ERTL MA, KRALL A. A Fast Java Interpreter[A]. JOSES Workshop at ETAPS'01[C], 2001.

[7] GAGNON EM, HENDREN LJ. SableVM: A Research Framework for the Efficient Execution of Java Bytecode[A]. Proceedings of the Java Virtual Machine Research and Technology Symposium[C]. Monterey, California, 2001. 2740.

[8] 黄广君,普杰信,吴庆涛. 嵌入式 Java 虚拟机实现中的代码优化[J]. 河南科技大学学报(自然科学版), 2003, 24(1): 57 - 60.

[9] 李允,罗蕾,雷昊峰,等. 嵌入式 Java 虚拟机的性能优化技术[J]. 计算机工程, 2004, 30(18): 47 - 49.