Jikes RVM 动态编译技术分析与性能评测

李 旭 卢 凯 李 根
(国防科学技术大学计算机学院 长沙 410072)

摘 要 随着Java语言的广泛应用,Java虚拟机的性能越来越受到人们重视,而虚拟机的动态编译技术是影响其性能的重要因素。Jikes RVM使用Java语言实现了一个Java虚拟机。首先分析了Jikes RVM的3个主要动态编译器的结构及其涉及的关键编译技术,包括基线编译、优化编译和自适应编译,然后利用SPECjvm Client98对Jikes RVM和Sun JVM的动态编译性能进行了测试和比较。测试结果显示,Jikes RVM的性能和Sun JVM性能基本相同。最后针对Jikes RVM的不足提出了改进Jikes RVM编译器的方法。

关键词 Jikes RVM,基线编译器,优化编译器,自适应编译器,SPECjvm Client 98

Analysis and Evaluation about Dynamic Compiling Technology of Jikes RVM

LI Xu LU Kai LI Gen
(School of Computer Science, National University of Defense Technology, Changsha 410072, China)

Abstract As the Java language is widely used, the capacity of Java virtual machine becomes more and more important, because Java virtual machine is the heart of Java technology. It's a truth that the dynamic compiler plays an important role in Java virtual machine, so this paper fully analysed dynamic compiling technologies of Jikes RVM at first, then took a evaluation on the dynamic compiler capacity of Jikes RVM and Sun JVM. After comparing and analyzing the testing results, we found out the advantages and disadvantages of the compiler of Jikes RVM, and put forward some approaches to improve it.

Keywords Jikes RVM, Baseline compiler, Optimized compiler, Adaptive compiler, SPECjvm Client 98

1 引言

Java体系结构包括4个独立但相关的技术:Java程序设计语言、Java class文件格式、Java应用编程接口(API)和Java虚拟机[1]。

Java虚拟机是Java平台的基石,是实现Java语言平台无关性的关键[2]。虚拟机的动态编译引擎是Java虚拟机的重要组成部分,其代码生成与编译开销直接决定了程序的执行性能。通常,虚拟机的动态编译引擎由3部分组成:基线编译、优化编译和自适应优化编译器。其中,基线编译是将字节码不经过任何优化直接翻译为本地机器指令。优化编译是对Java字节码进行一系列优化后生成本地机器指令。自适应优化编译器在开始时解释字节码,但在程序运行过程中,通过扫描程序堆栈统计出活动最频繁的代码。程序再次运行时,虚拟机只把那些活动最频繁的代码编译为本地代码。自适应优化编译器遵循"2-8"假设,即应用程序执行期间,80%的时间用于执行20%的代码。着重优化20%的热点代码,可以大幅度提高程序性能。

Jikes RVM是为研究基本的Java虚拟机设计问题开发的[3,4],其使用Java in Java的方式实现,即用Java语言实现Java虚拟机。本文分析了Jikes RVM的动态编译技术,并用SPECjvm Client 98,对Sun公司的JVM和Jikes RVM的编译性能进行了测试和比较,最后分析了动态编译技术影响Jikes RVM性能的机制。

2 Jikes RVM 动态编译技术分析

Jikes RVM包含基线编译器、优化编译器和自适应编译器,可以通过设置虚拟机参数使用其中之一对程序进行编译[4]。

2.1 基线编译器

基线编译器是Jikes RVM早期为保证正确性而开发的一个简单的编译器[5],是自适应编译器的基础。它不采用解释字节码的方式运行程序[6],而是在class文件被加载后,先把class文件中的字节码以方法为单位直接翻译为对应平台的机器码,同时确定垃圾收集点和线程切换点。翻译完成后程序开始运行。

由于基线编译器在翻译过程中不对程序进行任何优化,因此不能通过指令调度和合理地使用机器的寄存器来提高代码的执行效率,而且所有的操作都要通过虚拟机的栈来执行,所以基线编译器不能生成高性能的目标代码[6]。但基线编译生成代码的时间短,编译开销小,缩短了程序的启动时间。对于执行时间短的程序,基线编译的效率比较高。

到稿日期:2008-05-05 本文受国家973计划项目(2005CB321801),霍英东基金(111072)资助。
李 旭 硕士生,研究方向为计算机软件与理论,E-mail:lixu@nudt.edu.cn;卢 凯 博士,研究员,硕士生导师,研究方向为计算机软件与理论;李 根 博士生,研究方向为计算机软件与理论。

2.2 优化编译器

Jikes RVM 优化编译器首先将 Java 字节码转换为中间表示代码，然后对中间表示代码进行一系列优化，最后将中间表示代码转换为高效的机器码。Jikes RVM 的中间表示代码是基于符号寄存器的，其中一条指令由一个 n 元组来表示，包括一个操作符和零个或多个操作数。操作数可以表示符号寄存器，也可以表示物理寄存器、内存位置、常数、分支目标或类型。操作数不仅带有类型信息，而且含有 define-use 集合等附加信息，以便于编译优化时进行数据流和控制流分析[3]。

如图 1 所示，在编译前端，编译器将程序从字节码形式转换为 HIR(high level intermediate representation)。HIR 指令和 Java 字节码很相似，但是 HIR 指令在符号寄存器操作数上进行操作，而不是在隐式堆栈上。当程序从字节码转换为 HIR 后，在控制流和数据流分析的基础上，对程序进行简单快速的优化，其中包括复写传播、常量传播、局部变量寄存器重命名和冗余代码消除[7]，从而减小了最终生成的 HIR 代码的大小，减少随后的编译时间。

图 1 优化编译工作过程

进入编译后端后，编译器首先将 HIR 转换为 LIR(low level intermediate representation)。一条 HIR 指令一般会被转换为 1～3 条 LIR 指令。程序转换为 LIR 后，由于指令的增加，在 LIR 层再次进行复写传播、常量传播等优化。在 LIR 层进行的另一个重要的优化是公共子表达式清除，在对程序进行可用表达式分析之后，消除公共子表达式以便减少多余的运算。在 LIR 层优化完成后，编译器根据 BURS(bottom-up rewriting system)指令匹配算法将 LIR 转换为 MIR(machine intermediate representation)。

MIR 是最接近机器指令的中间表示代码，在 MIR 层优化的主要任务是完成寄存器分配和确定垃圾收集所需要的信息。首先对程序进行活跃变量分析，以判断符号寄存器和堆栈变量的活动范围，从而确定垃圾收集安全点及进行垃圾收集时所需要的信息。然后使用线性扫描全局寄存器分配算法[6]把物理寄存器指定给 MIR 指令中的符号寄存器。如果物理寄存器不足，还需要进行溢出处理。最后将程序转换为可执行的机器码，将机器码放入指令数组，从而完成优化编译的全过程。

2.3 自适应编译器

Jikes RVM 自适应编译器通过动态统计程序的运行信息将基线编译器和优化编译器结合使用，它由运行时统计子系统、自适应控制子系统和重编译子系统 3 部分组成[8]。自适应编译器 3 个子系统的关系如图 2 所示。首先基线编译器将 Java 字节码翻译为相应的机器码，在程序运行过程中，运行时统计子系统负责收集方法运行时的信息，自适应控制子系统通过分析收集到的信息，获得程序的“热点”方法，然后使用优化编译器对“热点”方法进行重编译，从而提高代码的运行效率。

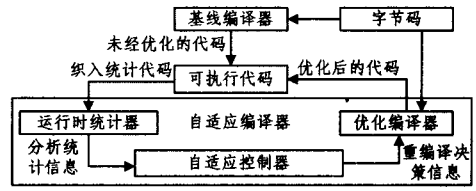

图 2 自适应编译器的组成

Jikes RVM 自适应编译器运行时统计子系统通过扫描程线程栈来获得方法的执行信息，然后将收集到的信息直接交给自适应控制子系统或者写入离线的数据库。

Jikes RVM 自适应编译器的核心是自适应控制子系统，它是编译器的控制中心，负责控制运行时子系统的统计行为，包括发生条件和统计时长，而且会通过分析统计信息决定是否应该改变统计子系统的统计策略。自适应控制子系统通过分析运行时统计子系统收集到的程序运行信息获得程序的“热点”方法，决定是否要对这些方法进行重新编译、进行哪个优化级别的重编译，并将决策信息交给重编译子系统去实施。

控制子系统将基线编译(baseline)和优化编译器的 opt0，opt1 和 opt2 三个优化级别映射为 0，1，2 和 3 四个优化等级。对于一个“热点”方法，控制子系统首先估算方法 f_h 在当前优化级别 i 下可能运行的时间 T_i，对方法 f_h 在优化级别 j 进行重新编译所需要的时间 C_j 和重编译后的方法 f_h 可能运行的时间 T_j，然后通过比较 T_j+C_j 和 T_i 来确定是否需要对方法 f_h 进行重新编译。如果 $T_j+C_j<T_i$，说明对方法 f_h 在优化级别 j 进行重新编译后，编译加运行的时间比在优化级别 i 下的运行时间短，因此需要对方法 f_h 进行重新编译，否则不进行重编译[8]。由此可见，估算方法的编译和运行时间是自适应编译器的关键。下面将介绍 JikesRVM 的时间估算策略。

估算[6,8] T_i：如果程序已经运行的时间为 T_p，编译器会假设程序将要运行的时间 $T_f=T_p$。并根据统计得到的方法 m 的执行频率为方法分配一个权值 ω_m($0<\omega_m\leqslant 1$)，则方法在优化级别 i 将要运行的时间 $T_i=T_f*\omega_m$。

估算[6,8] T_j：编译器根据对离线统计信息的分析得出优化级别 k 相对与级别 0 的加速比 S_k($S_k>1, S_{k+1}>S_k$)。如果方法当前优化级别为 i，将要重编译的优化级别为 j，则 $T_j=T_i*S_i/S_j$。

估算 C_j：现在 Jikes RVM 采用的方法是通过对离线数据进行分析[8]，得出在不同优化级别对方法进行编译所需时间的函数。目前使用的是线性函数，编译时间与方法的大小(代码量)成正比。

重编译子系统根据自适应控制子系统的决策，使用优化编译器对方法进行重新编译。编译完成后替换原来的代码。方法在下次运行时，将执行重编译后的代码。

3 Jikes RVM 编译器性能测试与分析

3.1 测试平台描述

虚拟机：本测试选用 Jikes RVM 2.9.0 和 Sun HotSpot Client1.5.0_05(Sun JVM)为研究对象。Sun JVM 是 Sun 公司提供的针对 J2SE 平台的虚拟机，采用了自适应优化的字节码执行方式。Jikes RVM 是 IBM 公司为研究虚拟机设计问题而开发的开源的虚拟机[3]。在测试中首先对 Jikes RVM 的自适应编译器和 Sun JVM 的性能进行了测试和比较，然后对 Jikes RVM 的 3 种编译器的性能进行了测试、比较和分析。

操作系统：Linux Fedora Core 6。

硬件平台：Pentium(R) 4 CPU 2.40GHz，内存 512MB。

Benchmark：本测试选用 SPECjvm Client 98，它可以用来测试 Java 虚拟机即时编译器的性能。包含 check，compress，jess，db，javac，mpegaudio，mtrt 和 jack 共 8 个测试程序[9]。

测试参数：rvm -verbose：gc -Xms256m -Xmx256m SpecApplication -a -m10 -M10 xxx java -verbose：gc -Xms256m -Xmx256m SpecApplication -a -m10 -M10 xxx。

参数表示虚拟机的堆大小为 256MB，统计垃圾收集时间，每个测试程序连续运行 10 次，xxx 表示测试程序。

3.2 测试结果与分析

每个程序的运行时间包括执行时间和垃圾收集时间，通过比较执行时间就可以发现编译性能的优劣。Jikes RVM 与 Sun JVM 编译性能比较如图 3 所示。图 3 由测试程序每次的运行时间减去垃圾收集时间得到执行时间，然后对 10 次执行时间求平均值得到。check 的运行时间只有几十毫秒，因此图中没有显示出来。从图 3 中可以看出，Jikes RVM 运行 javac，mtrt 和 jack 三个程序的时间比 Sun JVM 的时间长 1s 左右，而运行 compress 和 db 的时间少于 Sun JVM，两个虚拟机运行 jess 和 mpegaudio 的时间基本相同。

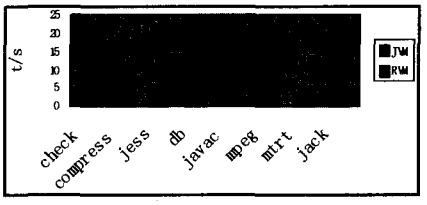

图 3 Jikes RVM 与 Sun JVM 性能比较

图 4 和图 5 显示了 Sun JVM 和 Jikes RVM 运行每个测试程序的执行时间变化曲线(已经除去了垃圾收集的时间)。

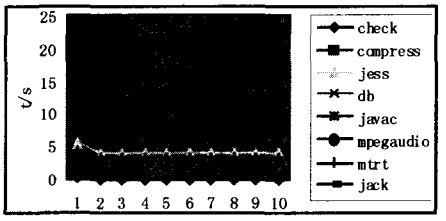

图 4 Sun JVM

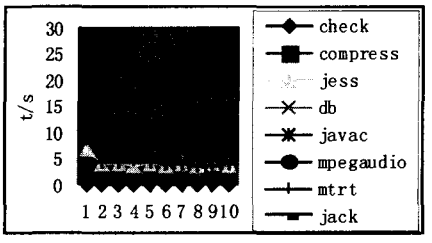

图 5 Jikes RVM

从图 5 看出，测试程序在 Jikes RVM 上第一次的执行时间明显长于后面几次，这是由于 Jikes RVM 自适应编译器在程序最初开始运行时采用的是基线编译器翻译的未经过优化的代码，因此效率不高。随着程序的运行，自适应编译器会对执行频率高的方法进行重新编译，对其进行优化，从而提高执行效率。而图 4 显示测试程序在 Sun JVM 运行的时间变化曲线比较平缓，这是因为程序在运行之前已经进行了一些优化。随着程序的运行，Sun JVM 同样会对程序的执行热点进行优化，这就是 Sun 公司的 Hotspot 技术。Hotspot 技术和 Jikes RVM 的自适应编译器虽然名字不同，实现细节不同，但原理是相同的。

但是，图 5 显示有的测试程序的运行时间并不是一直缩短，反而有的时候会加长，在运行 javac 时特别明显。这可能是因为 Jikes RVM 自适应编译器的控制器会根据运行时统计器收集的数据来改变统计行为和发生时间，从而影响重编译决策，造成程序执行时间暂时变长。从图 4 可以看出，Sun JVM 也存在同样的问题，在运行 compress 和 mpegaudio 时比较明显。

图 6 显示了 Jikes RVM 三种动态编译器的性能，由测试程序每次的运行时间减去垃圾收集时间得到执行时间，然后对 10 次执行时间求平均值得到。从图中可以看出，优化编译器和自适应编译器的性能相当，它们的性能一般比基线编译好两倍；在运行 jess 时，达到了 4 倍。

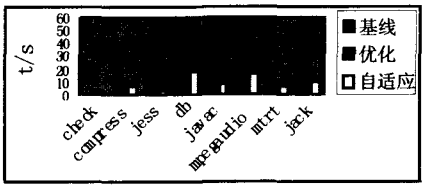

图 6 Jikes RVM 三种编译器性能对比

从以上分析可以看出，Jikes RVM 采用自适应编译器在运行大部分测试程序时性能与 Sun JVM 基本相同，平均运行时间一般比 Sun JVM 多 1s 左右。特别在运行 db 和 compress 时，性能还好于 Sun JVM。这是因为 8 个测试程序的测试重点不同，而且 Jikes RVM 最初是为服务器设计的[3]，因此在运行适合服务器特点的程序(如事务处理、字符串处理)时，Jikes RVM 的性能会更好一些。因为 compress 和 db 的大部分操作是字符串操作[9]，所以在 Jikes RVM 上的运行时间要比 Sun JVM 短。但是，由于 Jikes RVM 在程序最初运行时采用的是未经优化的代码，如果程序只运行一次，那么 Jikes RVM 的性能是比较差的。当程序重复运行多次时，自适应编译器才能真正发挥作用。

通过比较 Jikes RVM 优化编译和自适应编译的性能，可以发现自适应编译通过统计程序热点方法，并对热点方法进行重编译的策略很有效，使自适应编译的性能和优化编译的性能相当。但是，图 5 中通过对测试程序每次的运行时间进行比较发现，虽然 Jikes RVM 自适应编译器会根据运行时统计信息改变统计行为和重编译策略，但是有时策略的改变会引起性能的下降，这说明在对统计信息的分析和策略的选择方面 Jikes RVM 还需进一步改进。

结束语 Java 动态编译技术是编译领域的年轻分支，其区别静态编译的重要特征是可根据程序行为的不同，有针对性地、动态地对程序进行优化，从而提升程序的性能[10]。本文在分析 Jikes RVM 的动态编译技术基础上，对其动态编译

的性能进行了测试和分析，同时将 Jikes RVM 的自适应编译的性能与 Sun JVM 的性能进行了比较和分析。虽然 Jikes RVM 基线编译器的性能较差，但是作为自适应编译器的组成部分，它可以减少程序的启动时间，从而减少整个程序的编译和运行时间。Jikes RVM 的优化编译器按照 HIR→LIR→MIR 的过程将程序进行转换，并在转换的过程中对程序进行优化，最后生成高效的可执行代码。但是，由于要对整个程序进行转换和优化，因此程序的编译时间比较长。Jikes RVM 自适应编译器的关键是确定重编译策略。确定策略时，要估算方法的重编译时间及重编译后的执行时间和当前方法的执行时间。如果前两个时间的和小于方法的当前执行时间，那么就对方法进行重新编译，否则不进行。总之，由于采用了一系列的编译优化技术，Jikes RVM 的动态编译器总体体现了较高的性能，为研究 Java 动态编译技术提供了良好的平台。

参 考 文 献

[1] Bill V. Inside the Java Virtual Machine. Second Edition[Z]. Beijing: China Machine Press, 2005

[2] lindholm T, Yellin F. The Java Virtual Machine Specification. Second Edition[Z]. Addison-Wesley Publishing Co., 1999

[3] Alpern B, Litvinov V, Attanasio C R, et al. The Jalapeno virtual machine[Z]. 2000, 39: 211-238

[4] Donald P. The Jikes Research Virtual Machine User's Guide 2.9.0[Z]. 2007

[5] Suganuma T, Ogasawara T, Takeuchi M, et al. Overview of the IBM Java Just-in-Time Compiler[Z]. 2000, 39(1)

[6] Arnold M, Fink S, Grove D. Adaptive Optimization in the Jalapeno JVM[C]// ACM SIGPLAN Conference on Object Oriented Programming, Systems, Languages, and Applications. ACM Press, 2000

[7] Burke M G, Fink S, Grove D, et al. The Jalapeno Dynamic Optimizing Compiler for Java[C]. 1999

[8] Zhao Jisheng. Jikes RVM Adaptive Optimization System with Intelligent Algorithms[D]. The University of Manchester, 2004

[9] SPECjvm Client 98 Documentation release 1.03 edition [Z]. Standard Performance Evaluation Corporation

[10] 崔慧敏，戴桂兰，王生原，等. 动态编译技术研究[J]. 计算机科学，2004，31(7)

(上接第 83 页)

的结构化 P2P 系统外，常见的还有 Pastry[11] 和 Tapestry[12]。Pastry 同样基于环，每个结点有 $O(\log n)$ 个邻居，平均查找长度为 $O(\log n)$。最近，文献[13-15]分别独立地提出了基于 de Bruijn 图的结构化 P2P 覆盖网络。但由于 de Bruijn 图并不是 Cayley 图，因此在此我们不再讨论。

表 1 几种结构化 P2P 覆盖网络拓扑的性质

结构化 P2P 覆盖网络	对应的静态拓扑	顶点数	度	直径
Chord	Ring	2^d	d	log n
CAN	d-torus	k^d	2d	$d\lfloor k/2 \rfloor$
Ulysses	Improved Butterfly	kr^k	$r + k - 2$	$r + 1$

结束语 Cayley 图在并行分布式互连网络中有着重要的作用，由于 Cayley 图的顶点对称性和传递性，使得我们分析互连网络拓扑结构十分方便，尤其是对于结构化 P2P 覆盖网络。对于几种典型的结构化 P2P 覆盖网络，提出者们事先并不知道它们的静态拓扑就是 Cayley 图，文献[3]指出了这些结构化 P2P 覆盖网络的静态拓扑属于 Cayley 图，但并没有分析它们的构造本质，因此，在本文中，我们在文献[3]的基础之上，使用 Cayley 图的方法，分析了这几种典型的结构化 P2P 覆盖网络的构造本质，由此，为分析和构造更多的 P2P 覆盖网络提供了一种新的方法。

参 考 文 献

[1] Chen G, Xu C, Shen H, et al. P2P overlay networks of constant degree [C]// Proc. of the Int'l Workshop on Grid and Cooperative Computing. 2003: 285-192

[2] Aberer K, Alima L O, Ghodsi A, et al. The essence of p2p: a reference architecture for overlay networks [C]// Fifth IEEE International Conference on Peer-to-Peer Computing. 2005: 11-20

[3] Qu C, Nejdl W, Kriesell M. Cayley DHTs - a group - theoretic framework for analyzing DHTs based on cayley graphs [C]// The Second International Symposium on Parallel and Distributed Processing and Applications. 2004: 89-105

[4] Akers S B, Krishnamurthy B. A group-theoretic model for symmetric interconnection networks [J]. IEEE Trans. Comput, 1989, 38: 555-566

[5] Leighton F T. Introduction to Parallel Algorithms and Architectures: Arrays, Trees, Hypercubes [M]. Morgan Kaufmann, 1992

[6] Parhami B. Introduction to Parallel Processing: Algorithm and Architectures [M]. Plenum, 1999

[7] Stoica I, Morris R, Karger D, et al. Chord: A scalable peer-to-peer lookup service for internet applications [J]. Computer Communication Review, 2001, 31: 149-160

[8] Ratnasamy S, Francis P, Handley M, et al. A scalable content addressable network [J]. Computer Communication Review, 2001, 31: 161-172

[9] Xu J, Kumar A, Yu X. On the fundamental tradeoffs between routing table size and network diameter in peer-to-peer networks [J]. IEEE Journal on Selected Areas in Communications, 2004, 22: 151-163

[10] Kumar A, Merugu S, Xu J, et al. Ulysses: A robust, low-diameter, low-latency peer-to-peer network [J]. European transaction on telecommunications, 2004, 15: 571-587

[11] Rowstron A, Druschel P. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems [C]// IFIP/ACM International Conference on Distributed Systems latforms. 2001: 329-350

[12] Zhao Y, Kubiatowicz J, Joseph A. Tapestry: An infrastructure for fault-tolerant wide-area location and routing [R]. Technical Report UCB/CSD-01-1141. University of California, Berkeley, 2000

[13] Loguinov D, Kumar A, Rai V, et al. Graph-Theoretic Analysis of Structured Peer-to-Peer Systems: Routing Distances and Fault Resilience [J]. IEEE/ACM Transactions on Networking, 2005, 13: 1107-1120

[14] Kaashoek F, Karger D R. Koorde: A Simple Degree - optimal Hash Table [J]// International Conference on Peer to Peer System II [C]. vol. 2735, 2003: 98-107

[15] Fraigniaud P, Gauron P. D2B: a de Bruijn based content-addressable network [J]. Theoretical Computer Science, 2006, 355