小型微型计算机系统
Journal of Chinese Computer Systems

2008年4月 第4期
Vol. 29 No. 4 2008

Java性能优化技术综述

王会进，龙　舜

(暨南大学 计算机科学系，广东 广州 510632)
E-mail:long.shun@gmail.com

摘　要：Java由于其简单、面向对象、独立于硬件体系结构、安全等特点在各种应用领域内获得广泛的应用，但在很多情况下其运行性能仍有待提高.优化Java应用的运行性能成为当前业界迫切要解决的问题和当前研究的热点.本文简要回顾了当前在Java性能优化方面的最新研究成果，对其中的关键技术进行了深入探讨，并结合作者的经验提出对未来发展的一些看法.

关 键 词：Java；高性能；虚拟机；编译；优化；类库
中图分类号：TP314　　**文献标识码**：A　　**文 章 编 号**：1000-1220(2008)04-0720-06

Survey on Java Optimization Techniques

WANG Hui-jin, LONG Shun
(*Department of Computer Science, Jinan University, Guangzhou* 510632, *China*)

Abstract: Java has emerged as a dominant programming language widely used in a large variety of application areas. However, its architectural independent design means that it is frequently unable to deliver high runtime performance. This paper presents a survey on various approaches developed to improve Java performance, including static, just-in-time and dynamic optimization used in Java optimizing compilers, high-performance Java virtual machine, domain-specific optimization techniques and optimization of runtime libraries. It discusses these key techniques in depth and analyzes pros and cons of some successful systems, before outlining future directions.

Key words: Java; high performance; virtual machine; compilation; optimization; class library

1 引　言

Java[17]具有简单、解释执行、动态、安全、健壮、独立于硬件体系结构、可移植等特点，它通过丰富的类库支持包括分布式计算、多线程、网络等各种应用的开发，并支持通过第三方类库进行进一步的扩展，因而被各种应用领域广泛采用.

Java编译程序将Java源代码(.java)编译成与硬件体系结构无关的字节码，并将其存储在.class文件[21]中.Java虚拟机[21]中的解释程序负责解释执行.class文件中的字节码，由于它逐句解释并执行字节码，Java应用的运行性能(以下简称性能)无法达到以C/C++和Fortran等语言开发的应用的性能水平.但是，计算机技术的不断快速发展不仅将计算机的应用领域从单纯的科学计算和数据处理扩展至现代生活的各个方面，并对应用的运行性能也提出更高的要求.Java的上述特点使得它成为这些领域内进行应用开发的热门语言，这使得优化Java应用运行性能成为当前业界迫切要解决的问题和研究的热点.[16,28]分析和论述了确定性能优化的原则、定位性能瓶颈的方法等诸多方面，并提出了一些具体的代码优化措施.

本文简要回顾了当前在Java性能优化方面的最新研究工作成果，对其中的关键技术进行了深入探讨.Java性能取决于以下4个因素[28]：

(1) 应用本身；
(2) 虚拟机执行字节码的速度；
(3) 运行库的运行速度；
(4) 底层操作系统和硬件的速度.

其中(4)由操作系统及硬件设计开发所决定，超出本文讨论的范围，有关的讨论可参见[28].

2 Java编译优化

2.1 静态优化

静态优化是自六七十年代起已在各种语言和应用领域(尤其是高性能计算领域)得到充分发展和广泛应用的成熟优化技术.它主要通过发掘程序中的并行机会来提高其性能.[9]详细地讨论数据依赖性分析、识别归纳和约简变量、数组私有化和运行时刻分析技术(如试探式并行)等自动并行化的核心技术.

Java编译程序可以通过分析Java程序来发掘其中的并行机会，并根据分析结果生成并行代码.Javar[7]和Javab[8]通过分析检测Java源程序中出现在循环和递归方法调用，找出其中的可并行机会，然后通过多线程技术将其并行化.两者在真正支持多线程并行的Java虚拟机上都可以提高程序性能，并能将并行分析成本控制在低水平.类似地，[20]分析了在Java应用的解释运行过程中用多线程化方法进行性能优化

收稿日期：2006-11-23　基金项目：广东省自然科学基金项目(2002-010421)资助. 作者简介：王会进，男，1965年生，硕士，副教授，研究方向为计算机网络及其应用、Java应用技术和数据库系统应用；龙　舜，男，1971年生，博士后，讲师，主要研究方向为优化编译和机器学习.

的技术,并通过直接多线程化提高解释程序性能.JPT[6]分析Java源程序,将其中的可并行循环改写为PVM中的主(master)从(slave)程序,然后在PVM库的支持下并行运行.由于不生成字节码,JPT不具有Java的可移植性.类似地,JavaSpMT[19]在一个多处理器系统上实现Java循环的试探式并行.它将可能存在依赖关系的Java循环并行化,并以多流水线的方式运行.

实现程序自动并行化、尤其是自动检测其中的可并行代码是一项复杂的工作.传统并行编译系统大都针对高性能计算常用的Fortran和C/C++等语言,将这些工作移植到Java上的工作较少.更多的编译程序选择为Java增加指示(directive)成分,程序开发人员通过指示显式地标出Java程序中的应并行部分,编译程序根据这些指示生成并行代码,从而直接利用已有成果避开上述困难.

Titanium[33]是一个基于SIMD模型的针对高性能科学计算的类Java程序设计语言.其编译程序牺牲了Java的可移植性,将Titanium程序编译为C源码,然后即可利用大量现成的C/C++分析和优化技术,并通过显式的全局同步、全局和局部引用等方法实现并行.HPJava[11]是另一个基于SIMD模型的Java扩展,增加了对并行数组、真正的多维数组以及数据并行循环等的分布式程序控制结构的支持.类似的还有JPVM[15]和Java/DSM[36]等.它们共同的特点是通过特殊的数据和控制结构显式地标识程序的并行部分,并通过PVM或Threadmarks等外部系统或运行库支持程序的并行运行.虽然这些改动使得应用开发和运行具有更高的效率,但由于和标准Java存在差异以及需要外部运行系统支持,它们难以被广泛接受成为Java优化的标准做法.

笔者认为通过多线程并行仍能进一步提高Java性能.这是因为系统在创建、同步和回收线程等过程中会和应用程序竞争时间和内存等资源,这必然影响后者的运行,这一影响在并行线程数量较大的情况下尤为明显.在并行技术的传统应用领域如高性能计算中,这些代价与应用的运行时间和并行所能带来的性能提高相比往往可以忽略不计.但在一般情况下,程序并行化必须考虑这些因素的影响,若能根据在实际运行环境中的并行成本(可通过分析运行时间反馈获得)灵活调整并行方案(如使用的线程数),则能进一步提高Java性能.

除并行外,Java编译程序通常进行下述工作:

(1) 清除没有用到的类和方法(例如在子类中已被重载的基类的方法);

(2) 将无需动态绑定的方法调用直接改写为静态绑定,这不仅可以节省进行动态绑定所需的时间,还可以发掘出更多方法内置的机会;

(3) 清除多余的代码或指令,例如在可确定不会发生异常的情况下清除异常处理代码;

(4) 清除类中无用的域;

(5)清除或压缩.class文件中的行号表和局部变量表等仅在编译过程中需要而在编译结束后即可抛弃的辅助数据结构;

(6) 将循环平展;

(7) 将循环无关代码移出循环外;

(8) 进行指令调度等.

实践表明这些方法取得很好的效果.

面向对象程序设计强调封装性和访问控制,通常每个类都包含许多短小的方法用于访问数据域和进行简单的加工处理,方法内置是一项非常有效的优化技术.它将被调用方法的代码直接复制到调用点,因此节省在方法调用过程中保存和恢复现场所需的时间等资源;同时由于扩大了基本代码块,增加使用其它转换的机会,从而进一步提高程序性能.实现方法内置有两个技术难点:其一是判断在何种情况下进行内置,不恰当的判断不仅降低程序性能,而且增加编译时间.[12]用遗传算法来自动调节Jikes优化编译程序中的方法内置决策模块,取得显著的性能提高.另一个难点在于处理方法重载:编译程序必须根据参数个数和类型判断在调用点内置哪一个方法实现.若在编译过程中即可判定,编译程序就能以静态方式直接内置相应的方法代码;否则需要在程序运行到调用点时根据实际调用参数才能判定调用哪个实现,这就需要采用动态优化技术在运行时刻进行方法内置,这项技术被广泛采用于高性能Java虚拟机(如下2.3所述).

准静态(quasi-static)Java编译方法[27]将已完成与运行环境无关的基本优化的代码以准静态映像的方式存储,然后再进一步针对具体运行环境进行编译优化并生成可执行代码.系统仅在运行环境发生变化时才重新快速优化准静态映像并重新生成可执行代码.与重新由Java源代码或字节码开始进行优化编译的做法相比,它能避免重复的程序信息收集工作,从而节省进行程序分析和基本优化的大量时间,并大大提高代码对运行环境改变的响应速度.

Budimlic等[10]根据在应用开发过程的结束阶段Java代码大都已经趋向稳定的特点,将这时的所有代码打成一个包并在源码层对整个包进行全局优化,从中找到更多的优化机会.这一做法在不牺牲Java语言的灵活性的同时能提高应用程序性能.

2.2 即时编译

即时(just-in-time)编译是指在Java虚拟机装入Java字节码后即将投入解释运行的时候,将字节码编译成机器代码并直接运行.由于低速的字节码逐条指令解释执行被快速的机器码直接运行所取代,应用性能得到提高,而且没有牺牲Java的可移植性.即时编译技术已经被当前大多数Java虚拟机广泛采用,并成为其中一项核心功能.

要通过即时编译取得性能的提高,即时编译所花费的时间必须小于应用以解释方式运行所需的时间与它通过即时编译优化技术可以达到的最佳运行时间之间的差额.只有能够在这个时限内完成必需的分析和优化等工作,即时编译才能实际提高应用程序性能.对大多数应用而言,这个时限通常很短而不足以进行复杂耗时的分析和优化,因此即时编译程序大都采用一些简单快速的优化技术.

IBM的Java即时编译程序[18]在开始阶段采用一种基于堆栈的中间格式表达以便进行与平台无关的公共优化;然后改用基于寄存器的中间格式表达进行更深入的优化;最后进行与平台相关的一些优化并生成机器代码.开发组通过事先

选择并测试各种优化技术的组合确定了一组功效比高的优化技术(包括清除异常检测代码、方法内置、清除冗余的表达式计算、简化递归方法调用等),使得它可以在有限时间内尽可能地提高程序性能.

Intel 的 Java 即时编译程序[1]直接用 Java 字节码表示编译过程中使用的表达式和其他数据及控制结构,因此只须一次扫描即可生成 IA32 机器代码.它主要采用清除冗余的表达式计算、快速寄存器分配和清除数组访问越界检测等优化技术.

Jaguar 即时编译程序[30]通过实现高效通讯和 I/O 提高程序性能.它预先将各种系统资源封装成 Java 对象,并将相应的字节码编译成短小快速的机器代码段.通过在即时编译时用这些代码替换程序中的访问系统资源代码,它可以在保持字节码的可移植性和保护机制的同时提高应用性能.类似地,Kaffe[31]也是通过一层预先用机器代码写好的宏将字节码改写为机器代码,并采用快速寄存器分配技术以提高代码生成速度.

AJIT[5]程序用一套标识系统在整个即时编译过程(从前期的翻译阶段到后期的代码优化阶段)中收集并记录程序信息,以实现信息的"一次搜集、反复使用".它分析源程序生成标记,并将字节码与这些标记及其他相关操作相结合,经优化生成机器代码.

即时编译技术的缺陷在于它拖延了应用的启动响应,不适于在一些需要应有能做出快速响应的场合下使用.笔者认为它仍有进一步提高性能的空间,例如:它通常对所有遇到的程序都实施同一组优化,而实验结果表明对不同程序而言,有些优化没有起实际作用却占用了编译时间.若即时编译程序能加入关于程序静态特征和这些优化处理打开或关闭之间的关系判断,即可根据收集到的程序信息判断应如何优化程序,不影响生成代码效率的前提下缩短编译时间,提高响应速度.

2.3 动态优化

动态优化(dynamic optimization)根据应用在运行时刻的信息反馈动态地决定如何对其中的哪些代码段实施何种优化.这要求编译程序能够和运行系统配合以便监视程序运行、收集运行时刻的各种有关信息、加以分析并作出决策.高性能 Java 虚拟机广泛采用这一技术.

Jikes[2]的优化编译程序采用基于寄存器的中间格式,以便采用更多大胆而高效的优化技术来充分发挥硬件的性能.它提供了三个不同优化级别:优化级别 0 主要通过一些简单的编译时间实施的优化(例如简化算术表达式等)缩短中间代码以便缩短编译时间;优化级别 1 包括一些局部优化例如删除数组边界检测代码等;而优化级别 2 则主要提供基于 SSA 的数据与控制流的优化.Jikes 通过其中的 profiler 监视程序的运行,从中找到程序运行的热点(占据大量运行时间的代码段);controller 分析监视结果,若某个热点方法在某一优化级别的预期运行时间大于以更高优化别重新编译所需时间和编译后估计的运行时间之和,Jikes 就将该方法以新级别重新编译.另一方面,它还采用了在线的基于运行时间反馈的方法内置技术,能根据运行时间的方法调用情况决定内置哪个方法实现,解决了多态性给方法内置带来的困难.

笔者认为深入发掘动态优化技术将能进一步提高 Java 应用性能和提高动态编译优化的效率.以上述 Jikes 动态优化系统为例,它将各种优化转换较笼统地划分成不同优化级别,而没有深入考虑各种程序特征和运行时间反馈可以帮助优化决策,例如某个转换对目标方法是否有效,是否能直接删除异常处理代码等.亦即是说在划分优化级别的基础上对不同优化技术再分别处理.另一方面,它对程序运行时间和优化所须时间的估计也较粗,可以考虑以离线方式根据监视运行时间结果动态调整估计公式中的参数.

3 高性能 Java 虚拟机

不同的 Java 虚拟机实现会导致不同的运行特性,各种实现或是着重提供便利的开发环境而对其它方面考虑不足,或是仅考虑了某些影响性能的因素,导致在整体性能可能有很大差异.高性能 Java 虚拟机可以通过充分利用实际硬件平台和操作系统的能力提高 Java 应用的性能.

IBM 的 Jikes Research Virtual Machine[2]的首要目标是充分发挥高性能 SMP 服务器在存储体系、指令级并行和多处理器并行等方面的能力,并提供良好的多线程扩展能力和持续可用性等.它具有以下特点:

(1) 重新根据硬件体系结构的特点设计了高效的对象格式;

(2) 完全放弃解释执行而只采用编译运行;

(3) 几乎全以 Java 实现运行子系统,它提供了异常处理、动态类载入、I/O 等服务;

(4) 将 Java 线程与虚拟处理器间的对应关系与虚拟和物理处理器和操作系统线程间的对应关系分开处理,提高了系统的可移植性和可扩展性;

(5) 采用准抢占方式切换线程,并通过三种锁机制提供低成本的线程同步,而无须操作系统支持;

(6) 支持多种内存管理系统提供高效准确的内存的分配和回收;其中包含并行的对象分配器和垃圾搜集器;

(7) 通过多个优化程序提供不同的动态优化级别,并提供了支持动态优化的运行监控子系统.

Sun 为服务器端应用 Java HotSpot Virtual Machine[29]通过以下方法提高应用性能:

(1) 动态可适应性优化:HotSpot 首先以解释方式启动应用,通过监视运行情况确定其中运行的热点,再将这些热点代码段集中优化编译成机器码并直接运行;

(2) 方法内置:为克服方法重载给方法内置带来的困难,在编译阶段,HotSpot 假设目标类只有一个子类并大胆进行方法内置.在运行过程中当有另一子类被动态载入时,它用动态反优化技术将已优化的机器代码重新转换为未优化的解释代码,并转入解释运行,若新代码段仍为运行热点,则将其重新优化编译后直接运行;

(3) 重新设计了对象表达格式,并用直接的内存引用取代对象句柄,使 HotSpot 对临时数据结构的内存分配效率可以接近 C 的基于堆栈的内存分配的水平;

(4) 内存管理系统能通过快速准确的垃圾收集机制保证所有无法访问的对象所占的内存均被可靠地回收,并通过对象重定位清除内存中因释放对象而产生的碎片;而且它充分利用 Java 应用中大量对象具有较短生命期的特点,将生命期较长的对象分开保存,从而提高垃圾收集的效率;

(5) HotSpot 以一对一方式用操作系统线程实现虚拟机中的 Java 线程,使线程间的互相干扰降到最低,并提供了快速的线程同步机制.

Jessica2[36]通过提供一个基于分布式计算环境的单一虚拟机映象,使得多线程 Java 应用可以在分布式环境中以透明方式运行而无须考虑线程的转移和调度等低层问题.而 Hyperion[4]系统将多线程 Java 应用的字节码编译为机器代码,并在运行库的支持下在一个基于处理器集群的单一 Java 虚拟机上运行.类似的还有 Kaffemik[3]和 Jupiter[14]等,但它们都没有解决如何用多线程分担应用并行负荷的问题.

某些应用场合要求 Java 虚拟机具有很高的响应速度以便快速启动应用.这取决于以下三个因素:

(1) 操作系统启动 Java 虚拟机进程的速度;

(2) 虚拟机初始化所需的时间;

(3) 具体应用的设计和结构.

(1)通常取决与虚拟机进程的代码大小和需要马上装入的运行库的大小,因为它们决定了要将它们装入内存所需的时间.显然,假若有关的内容已被提前装入内存(如在最近刚刚启动过 Java 虚拟机,则相应的内容还驻留在内存中未被取代),这个时间就可以大大缩短.(2)则取决于虚拟机的具体实现,通常以解释方式运行程序不需要额外的即时编译字节码的时间,它的启动速度快于采用即时编译技术运行程序.而对于(3),如果应用需要使用很多的类,则启动时间就越长;开发人员可考虑将一些初始化工作安排以后台方式进行以提高其响应速度.

笔者认为上述 Java 虚拟机尚未充分利用其后台处理能力对应用运行过程中收集到的信息进行深入分析,而且无法逐步积累优化经验,未来应考虑充分利用各次运行之间的空闲时间进上述工作,以便更好地指导动态优化.

4 针对应用的优化

Java 被广泛应用于各种应用领域,不同领域内的应用通常带有不同的特征可被优化编译程序所利用.

高性能计算应用通常含有大量的数组引用.Java 不直接支持多维数组,而是将其看作数组的数组并进行强制数组访问边界检测,这大大影响实现效率.NINJA[25]提供了一个支持真正多维数组的类包,通过编译可以生成无异常的安全的代码段,并对其进行包括各种循环重构和并行化等在内的更全面的优化.实验结果表明在不牺牲安全保障的同时,它可以使应用性能达到接近 Fortran 的水平.但 NINJA 缺乏灵活性,它仅支持三维或以下的基本数值类型的数组,而不支持由用户自行定义更高维的数组或对象的数组,而且数组引用的表达格式采用类似函数调用的格式,与 Java 的风格差别很大,这给开发带来了很大的不便.

高性能计算应用的另一特征是通常包含大量的循环.各种循环重构转换的适当组合可以显著地提高该领域应用的性能.[23]对一个由各种循环重构转换以任意参数和次序组合而成的大而复杂的优化空间进行启发式搜索,它能迅速地找到其中好的点提高程序的性能.其缺点在于对于每一个程序都必须从头开始搜索优化空间,并在搜索过程中不断测试各点并根据反馈决定未来的搜索方向.高性能计算应用大都是长期反复或连续运行,因此可以接受以反复测试为代价换取性能的提高.但其它应用领域则未必如此.[22]根据程序的静态特征直接从过往经验中寻找合适的转换序列,因此无需搜索和反复测试即可找到合适的转换,迅速取得性能提高.

数据库系统是一个重要的 Java 应用领域,[34]提出了在数据库的物理设计、正确选择驱动程序、代码编写等方面优化 Java 数据库访问性能的方法.另外,Java 也被广泛应用于网络和通信领域并开始被嵌入式系统所接受,[26]分析影响手机应用程序的性能指标,并在有限的内存资源的约束下优化了基于 J2ME 的 MIDP 应用,从而提高手机应用的性能.

在应用开发过程中使用适当的数据结构、算法和其它优化技术可更显著地提高应用的性能.例如:由于创建对象需要花费大量的时间,开发人员应避免在频繁调用的方法中创建临时对象,并尽可能使用 Java 基本数据类型取代对象类型;同时,尽可能简化构造方法,并用 clone()方法而非构造方法来创建相似的多个对象等;另一个例子通过尽量使用准确的类型以避免强制类型转换的开销;以及多用局部临时变量而少用实例和静态变量等等.[28,32]深入讨论有关编程技术.

5 优化类库

提高类库的性能也是有效提高 Java 应用性能的一种可行途径[28].类库的优化可以有两种方式:直接用新开发的类取代原有 JDK 中的类;或采用间接方法,在 Java 应用开发时使用新的类库.直接方式的困难在于当应用完成开发而投入使用时,新类库也必须随应用一起发布,在应用开发人员不能完全控制应用使用环境的情况下,这会带来维护的困难.

优化类和类库需要从重新设计开始着手.出于提高通用性和灵活性的考虑,类和类库的设计会以牺牲性能为代价引入一些冗余的类属(generic)关系和层次以便扩展.例如,Java 以 Vector 类提供向量的表述,该类适用于任意类型的对象向量.但若我们只需要表示每个元素是某个特定类型(如 String)的向量,就可牺牲灵活性而直接用数组方式实现,这可以节省访问 Vector 中的元素时将其由 Object 类型强制转换成 String 类型所需时间,从而大大提高该类的性能.

笔者认为类库的优化应结合具体应用领域的特点,在可能的情况下充分结合虚拟机所处的硬件环境将其优化成可执行代码是提高其性能的最好办法.

6 小 结

优化 Java 应用的性能是当前业界迫切要解决的问题和当前研究的热点.综上所述,笔者认为:

(1)高性能 Java 虚拟机是提高 Java 应用性能的最有效方法,因此应成为未来研究开发的重点.一方面开发高性能 Java 虚拟机应充分发挥目标硬件平台的潜能;另一方面,虚拟机有足够时间可以进行较复杂的分析、并行化和动态优化技术;而且,在各次运行之间的间隔,虚拟机能离线地进行更深入的分析处理(如通过机器学习自行提高优化决策水平),指导具体优化的实施[13]或是并行化工作中的运行负荷分配[24]等.

(2)即时编译技术由于有可利用时间的上限,通常仅能通过简单的分析优化技术取得有限的性能提高,而难以充分发挥底层硬件在存储体系、指令级并行和多处理器并行等方面的能力.但由于很多应用场合仍然需要 Java 的硬件无关性,作为在不影响硬件无关性的同时能够提高程序性能的重要手段,即时编译技术仍然有存在的必要.

(3)随着越来越多的嵌入式系统选择 Java 进行应用开发,该领域内的特殊优化需求将成为研究热点.例如,优化目标不仅要考虑代码的运行速度,还因为内存和电池容量的限制对其大小和能耗有严格的要求,如何根据这些应用的具体要求和自身特点进行优化、如何使应用的综合性能达到最优具有广阔的应用前景.

本文回顾了当前在 Java 性能优化方面的最新研究工作和成果,对其中的关键技术进行了深入探讨,并结合作者的经验提出对未来发展的一些看法.

References:

[1] Adl-Tabatabai A, Cierniak M, Lueh G, et al. Fast effective code generation in a just-in-time java compiler[C]. Proceedings of 1998 ACM SIGPLAN Conference on Programming Language Design and Implementation. Canada: 1998, 280-290.

[2] Alpern B, Attanasio C, Barton J, et al. The Jikes research virtual machine[EB/OL]. http://www-106. ibm. com/developerworks/java/library/j-jalapeno/.

[3] Andersson J, Webber S, Cecchet E, et al. Kaffemik-a distributed jvm on a single address space architecture[C]. Proceedings of the Java Virtual Machine Research and Technology Symposium. US: 2001.

[4] Antoniu G L. Bouge L, Hatcher P, et al. Thy hyperion system: compiling multithreaded java bytecode for distributed execution [J]. Parallel Processing, 2001, 27(10): 1279-1291.

[5] Azevedo A, Nicolau A, Sharp O. Java annotation-aware just-in-time (AJIT) compilation system[C]. Proceedings of ACM'99 Java Grande Conference, US, 1999.

[6] Beyls K, Hollander E, Yu Y. JPT: a java parallelization tool [C]. Proceedings of the 6th European PVM/MPI Users' Group Meeting. Lecture Notes of Computer Science 1697, Springer-Verlag, Spain: 1999, 173-180.

[7] Bik A, Gannon D. Javar-a prototype Java reconstructing compiler[J]. Concurrency, Practice and Experience, 1997, 9(11): 1181-1191.

[8] Bik A, Gannon D. A prototype bytecode parallelization tool [TR]. Technical Report TR489, Dept. of Computer Science, Indiana University, US, 1998.

[9] Blume W, Eigenmann R, Hoeflinger J, et al. Automatic detection of parallelism: a grand challenge for high performance computing[J]. IEEE Parallel and Distributed Technology: System and Technology, 1994, 2(3): 37-47.

[10] Budimlic Z, Kennedy K. Almost-whole-program compilation [C]. Proceedings of the Joint ACM Java Grande - ISCOPE Conference, US: 2001, 104-111.

[11] Carpenter B, Zhang G, Fox G, et al. HPJava: data parallel extension to Java[C]. Proceedings of the ACM 1998 Workshop on Java for High Performance Network Computing, UK: 1998, 233-237.

[12] Cavazos J, O'Boyle M. Automatic tuning of Inlining heuristics for java JIT compilation[C]. Proceedings of The 12th Workshop on Compilers for Parallel Computers. Spain, 2006.

[13] Cavazos J, Moss J, O'Boyle M. Hybrid optimizations: which optimization algorithm to use[C]. Proceedings of the 15th International Conference on Compiler Construction. Austria, 2006.

[14] Doyle P, Abdelrahman T. Jupiter: a modular and extensibe JVM[C]. Proceedings of the 3rd Annual Workshop on Java for High Performanece Computing. Italy: 2001, 37-48.

[15] Ferrari A. JPVM: network parallel computing in java[J]. Concurrency: Practice and Experience, 1998, 10(11): 985-992.

[16] Gong J, Guo Q. On performance optimization of java applications[J]. Microcomputer Applications, 2002, 18(3): 23-27.

[17] Gosling J, Joy B, Steele G. The java language specification [M]. US: Addison-Wesley, 1996.

[18] Ishizaki K, Kawahito M, Yasue T, et al. Design implementation and evaluation of optimization in a just-in-time compiler [C]. Proceedings of 1999 ACM Java Grande, US: 1999, 119-128.

[19] Kazi I, Lilja D. JavaSpMT: a speculative thread pipelining parallelization model for Java programs[C]. Proceedings of the International Parallel and Distributed Processing Symposium, Mexico: 2000, 559-568.

[20] Li Y, Luo L, Lei H, et al. Performance optimization technology Java virtual machine for pervasive computing terminals[J]. Application Research of Computers, 2005, 20(3): 55-57.

[21] Lindholm T, Yellin F. The Java virtual machine specification [M]. US: Addison-Wesley, 1999.

[22] Long S, O'Boyle M. Adaptive java optimization using instance-based learning[C]. Proceedings of the 18th ACM International Conference on Supercomputing, France: 2004, 237-246.

[23] Long S, Fursin G. A heuristic search algorithm based on unified transformation framework[C]. Proceedings of the International Workshop on High Performance Scientific and Engineering Computing. Norway: 2005, 137-146.

[24] Long S. Using machine learning to allocate parallel workload [C]. Proceedings of The 1st International Conference on Innovative Computing and Intelligent Control, China, 2006.

[25] Moreira J, Midkiff S, Gupta M, et al. Java programming for high performance computing[J]. IBM System Journal, 1999, 39 (1): 21-56.

[26] Qiao S. Java J2ME MIDP code optimizing memory resource[J]. Information Technology. 2004, 11(10):11-15.

[27] Serrano M, Bordawekar R, Midkiff S, et al. Quicksilver: a quasi static compiler for java[C]. Proceedings of the 2001 ACM Conference on Object-Oriented Programming, Systems, Languages and Applications. US: 2001,66-82.

[28] Shirazi J. Java performance tuning[M]. US: O'Reilly,2002.

[29] Sun Microsystems. The Java HotSpot virtual machine[EB/OL]. http://java.sun.com/products/hotspot/,2002.

[30] Welsh M, Culler D. Jaguar: enabling efficient communication and I/O in java[J]. Concurrency, Practice and Experience, 2000, 12(7):519-538.

[31] Wilkinson T. Kaffe: a JIT and interpreting virtual machine to run Java code[EB/OL]. http://www.transvirtual.com/,1998.

[32] Xie X, Han K, Wang L. Java source code performance optimizing[J]. Computer and Modernization. 2005,12(8): 8-10.

[33] Yelik K, Semenzato L, Pike G, et al. Titanium: a high performance Java dialect[C]. Proceedings of ACM 1998 Workshop on Java for High Performance Network Computing. UK: 1998,1-13.

[34] Yu X. J2EE database JDBC performance optimization[J]. Application Research of Computers, 2005, 22(4):90-92.

[35] Yu W, Cox A. Java/DSM: a platform for heterogeneous computing[J]. Concurrency: Practice and Experience, 1997, 9(11):1213-1224.

[36] Zhu W, Wang C, Lau C. JESSICA2: a distributed Java virtual machine with transparent thread migration support[C]. Proceedings of IEEE 4th International Conference on Cluster Computing. US: 2002,381-388.

附中文参考文献:

[16] 巩　晶,郭庆平. Java 性能优化[J]. 微型电脑应用, 2002,18(3):23-27.

[20] 李　允,罗　蕾,雷昊峰,等. 面向普适计算终端的 Java 虚拟机性能优化技术研究[J]. 计算机应用研究, 2005,20(3):55-57.

[26] 乔少杰. 基于 J2ME 的手机应用程序的性能优化[J]. 信息技术, 2004,11(10):11-15.

[32] 谢晓兰, 韩可轶, 王　林. 提高 Java 程序性能的若干方法[J]. 计算机与现代化, 2005,12(8):8-10.

[34] 于晓慧. J.2EE 架构下数据库访问的性能优化研究[J]. 计算机应用研究, 2005,22(4):90-92.