软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2013,24(11):2558−2570 [doi: 10.3724/SP.J.1001.2013.04482]

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel/Fax: +86-10-62562563

基于统计学习分析多核间性能干扰*

赵家程[1,2], 崔慧敏[1], 冯晓兵[1]

[1](计算机体系结构国家重点实验室(中国科学院 计算技术研究所),北京 100190)

[2](中国科学院大学 计算机控制与工程学院,北京 100049)

通讯作者: 赵家程, E-mail: zhaojiacheng@ict.ac.cn

摘 要: 普遍认为,云计算和多核处理器将会统治计算领域的未来.但是,目前云计算数据中心的计算资源使用率非常低,其主要原因在于多核处理器上存在严重且不可预知的性能干扰.为了保证关键应用程序的 QoS,只能禁止这些关键程序与其他程序共同运行,导致了资源的过度分配.为了提高数据中心的利用率,分析多核间的性能干扰成为一个关键的问题.观察到程序遭受的核间性能干扰可以表示为内存子系统总压力的线性分段函数,而与构成压力的具体应用程序无关.以此观察为基础,提出了一种基于统计学习的多核间性能干扰分析方法,使用主成分线性回归的方法获得干扰模型,可以精确且定量地预测任意程序由于内存子系统资源竞争导致的性能下降.实验结果表明,平均预测误差仅为 1.1%.

关键词: 云计算;多核;核间性能干扰;统计学习;主成分分析;线性回归

中图法分类号: TP312 **文献标识码**: A

中文引用格式: 赵家程,崔慧敏,冯晓兵.基于统计学习分析多核间性能干扰.软件学报,2013,24(11):2558−2570. http://www.jos.org.cn/1000-9825/4482.htm

英文引用格式: Zhao JC, Cui HM, Feng XB. Analyzing cross-core performance interference on multi-core processors based on statistical learning. Ruan Jian Xue Bao/Journal of Software, 2013,24(11):2558−2570 (in Chinese). http://www.jos.org.cn/1000-9825/4482.htm

Analyzing Cross-Core Performance Interference on Multi-Core Processors Based on Statistical Learning

ZHAO Jia-Cheng[1,2], CUI Hui-Min[1], FENG Xiao-Bing[1]

[1](State Key Laboratory of Computer Architecture (Institute of Computing Technology, The Chinese Academy of Sciences), Beijing 100190, China)

[2](School of Computer and Control Engineering, University of Chinese Academy of Sciences, Beijing 100049, China)

Corresponding author: ZHAO Jia-Cheng, E-mail: zhaojiacheng@ict.ac.cn

Abstract: Cloud computing and multi-core processors are emerging to dominate the landscape of computing today. However, in terms of computing resources, the utilization of modern datacenters is rather low because of the potential negative and unpredictable cross-core performance interference. To provide QoS guarantees for some key applications, co-locations of such applications are disabled, causing computing resource overprovisioning. Therefore precise analysis for cross-core interference is a key challenge for improving resource utilization in datacenters. This study is motivated by the observation that the performance degradation of one application suffered from cross-core interference can be represented as a piecewise function of the aggregate pressures on memory subsystem from all cores, regardless of which applications are co-running and what their individual pressures are. The study results in a statistical learning-based method for predicting cross-core performance interference as well as predictor models using PCA linear regression, which can

* 基金项目: 国家自然科学基金(61202055, 60970024, 60925009, 60921002, 61100011); 国家高技术研究发展计划(863)(2012AA010902); 国家重点基础研究发展计划(973)(2011CB302504)

收稿时间: 2013-05-08; 修改时间: 2013-07-17; 定稿时间: 2013-08-27

quantitatively and precisely predict performance degradation caused by memory subsystem contention in any applications. Experimental results show that the average prediction error of the proposed method is 1.1%.

Key words: cloud computing; multi-core; cross-core interference; statistical learning; principle component analysis; linear regression

多核处理器及云计算被认为将会统治计算领域的未来[1].一方面,微处理器厂商在不断地推进多核/众核架构的发展,将越来越多的处理器核集成到同一芯片中.另一方面,随着云计算的高速发展,越来越多的计算任务开始迁移到云计算的数据中心中,数据中心的效率成为一个普遍关注的问题.而在目前的数据中心中,多核处理器被广泛采用.

片上多核处理器(chip multiprocessors,简称 CMPs)在同一芯片上集成多个处理器核心.这些处理器核心共享许多的内存子系统资源,如最后一级缓存、数据预取器、内存控制器等.由于对共享资源的竞争,将导致运行在同一芯片上不同核上的程序遭受严重且不可预知的性能干扰[2−7].数据中心承载着大规模的网络服务,其效率是一个非常关键的问题,但是其计算资源的使用率却非常低,一般不超过 20%[8],其中最主要的一个原因就在于片上多核处理器上存在的核间性能干扰.核间性能干扰的存在,将会为应用程序的服务质量(quality of service,简称 QoS)带来不可预测的影响.为了保证关键应用的 QoS,现有的数据中心调度方法禁止这些应用与其他应用运行在同一片上多核处理器上[3],导致硬件资源使用率较低.为了在保证 QoS 的前提下允许关键应用与其他应用运行在同一处理器上,找到一种合适的方法精确地预测运行在同一处理器上的程序之间的性能干扰,成为一个亟待解决的问题.

尽管在减少由于资源竞争导致的核间性能干扰方面,已经有很多人使用不同的方法展开了相当多的研究[6,7,9−13],比如为了保证性能隔离的缓存管理策略[14,15],但是其中的绝大多数只提供定性的结果[9−12],而不能提供定量的精确的结果,所以不能直接被用来解决数据中心对关键应用程序的 QoS 保证问题.

Mars[3]提出了一种被称为 Bubble-Up 的方法来测量程序对于资源竞争的敏感性,从而可以定量地分析程序由于对内存子系统资源竞争而导致的性能下降,但是该方法局限于只有两个应用程序运行在同一个多核芯片上的场景,对于更多应用混合的数据中心缺乏适用性.Bandit 方法[16]可以用来分析任意程序共同运行时由于对共享带宽竞争导致的性能下降,其刻画了应用程序的性能与系统的总带宽占用的关系,但是该工作只关注了共享带宽的竞争,而且并不能用来预测任意程序遭受的核间性能干扰.MISE[17]关注的是共享带宽和共享内存的竞争,但是需要额外的对硬件的改动.

统计学习是一种运用模型对数据进行分析与预测的方法,其应用领域正处在飞速发展中,如计算机视觉[18,19]、网络流量及协议分析[20,21]等.在计算机体系结构领域,统计学习也在逐渐扮演越来越重要的角色.李胜梅等人[22]使用统计的方法分析应用程序的性能,Tang 等人[4]使用统计学习的方法来分析程序对共享资源的竞争特性,Cavazos 等人[23−25]使用机器学习的方法来优化迭代编译的效果,Michael 等人[26−28]同样将统计学习的方法用在编译优化中.作为一种以数据为对象的方法,统计学习在基于实验数据的研究上具有其独特的优势.

本文将统计学习的方法应用于核间性能干扰的分析上,提出了一种基于统计学习的核间性能干扰分析方法,使用主成分线性回归的方法获得干扰模型,可以用来量化地分析运行在同一处理器上的任意程序由于对共享内存子系统资源竞争导致的性能变化,其主要内容包括:1) 我们观察到,共同运行在同一处理器上的程序的性能下降可以表示为所有共同运行的程序对内存子系统总压力的线性分段函数;2) 上述线性分段函数可以通过主成分线性回归的方法获得;3) 我们通过实验验证了该方法的有效性,平均绝对误差为 1.1%.通过该方法,可以量化地分析程序间的性能干扰,为数据中心的调度程序或者操作系统的调度程序提供调度依据.

本文第 1 节介绍我们的工作的主要动机.第 2 节介绍我们的主成分线性回归的方法.第 3 节是实验部分.第 4 节对全文进行总结.

1 研究动机

我们首先通过部分实验说明核间的性能干扰存在且非常严重;其次介绍我们观察到的重要现象,即核间性

能干扰可以表示为所有共同运行的程序对内存子系统总压力的线性分段函数.

1.1 核间性能干扰

图 1 是部分 SPEC CPU 2006 程序在一个四核处理器上的核间性能干扰,其中,我们使用的实验平台为 Intel 的四核 Xeon E5506(平台详情见第 2.1 节),程序的性能下降由公式(1)计算:

$$PD=(Time_{co\text{-}run}-Time_{solo})/Time_{solo} \quad (1)$$

其中,PD 是指程序的性能下降,$Time_{solo}$ 是指程序单独运行在该四核处理器的一个核上时所需的运行时间,$Time_{co\text{-}run}$ 是指程序和其他程序共同运行在该四核处理器上所需的时间.图 1 中,横轴代表工作负载,每 4 个程序一组为一个工作负载;纵轴代表该工作负载运行在实验平台上时,工作负载中每个程序各自由公式(1)计算得到的性能下降.从图 1 中我们可以看出:在不同的工作负载中,相同的程序遭受的性能下降变化范围非常大,比如 471.omnetpp,在图示的 7 个工作负载中,性能下降得最少的只有 23%,而性能下降得最多的却高达 83%,这种大幅度且波动巨大的性能变化真实地反映了目前片上多核架构中普遍存在的核间性能干扰问题.

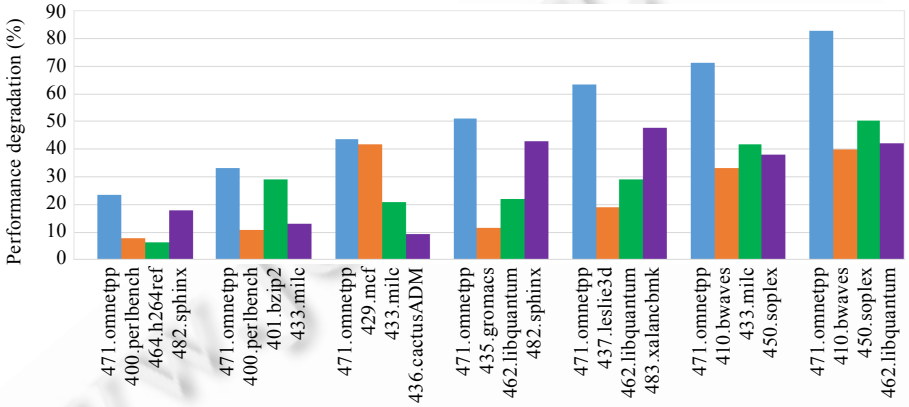

Fig.1 Cross-Core interference when co-running some SPEC 2006 applications on a quad-core Xeon

图 1 在四核 Xeon 平台上使用 SPEC 2006 观察到的核间性能干扰

1.2 线性分段函数

在实验中,我们观察到一个重要的现象:对于任意一个程序,其遭受的核间性能干扰是和它共同运行在同一芯片上的所有程序对内存子系统的压力之和的线性分段函数,与哪些程序和它共同运行无关.这个现象激发了我们采用线性回归的方法来分析核间的性能干扰.该现象包含两个方面:

1) 核间性能干扰与内存子系统的总压力相关,而与构成压力的各个程序无关.在分析核间性能干扰时,程序的特征可以描述为两个方面:对共享资源的压力和对共享资源竞争的敏感性[29].而对内存子系统的总压力是各个共同运行程序对共享资源的压力之和,这种压力代表了内存子系统资源竞争的激烈程度.任意程序遭受的性能干扰都将与共享资源竞争的激烈程度有直接关系,而每个程序对资源竞争的敏感性不同,这意味着每个程序遭受的性能下降不仅与系统总压力有关,而且也与自身的特征有关.这引出了以下的一个方面:

2) 程序遭受的核间性能干扰是内存子系统总压力的线性分段函数.线性函数是我们观察到的实验现象,而分段函数可以用来区分内存子系统资源的竞争程度.由于不同的程序对共享资源竞争的敏感度不同,比如部分程序对缓存竞争更为敏感,而部分程序对带宽的竞争更为敏感,这意味着程序在不同的内存子系统压力下的敏感性也不相同,分段的函数允许我们更精确地捕获这种敏感性的差异.

我们用图 2 来阐明我们观察到的这种现象.

图 2 中,我们以 SPEC 2006 中的 471.omnetpp 为例,从 SPEC 2006 和 Synthetic Benchmarks[30]中随机抽取了不同的程序组合(每组 3 个程序)与 471.omnetpp 组成一个工作负载,一起运行在我们的实验平台(四核处理器)

上,根据公式(1)计算其性能下降.我们选择共享带宽和共享缓存的总压力来代表内存子系统的总压力(详见第1.1 节).图 2 中的星形点代表了我们所有的数据点,每个数据点对应一个工作负载,可以表示为((P_{bw},P_{cache}), $PD_{omnetpp}$).其中,$PD_{omnetpp}$ 是在和该工作负载共同运行在实验平台上时,471.omnetpp 的性能下降,由公式(1)计算;P_{bw} 和 P_{cache} 表示一个包含了 471.omnetpp 的工作负载对共享带宽和共享缓存的总压力,由公式(2)计算:

$$P_{cache}=cache_{omnetpp}+\sum_{i=1}^{3}cache_i,P_{bw}=bw_{omnetpp}+\sum_{i=1}^{3}bw_i \tag{2}$$

其中,P_{cache} 和 P_{bw} 分别代表共享缓存上的总压力和共享带宽上的总压力,$cache_{omnetpp}$ 和 $bw_{omnetpp}$ 分别代表471.omnetpp 对共享缓存和共享带宽的压力,$cache_i$ 和 bw_i 分别代表工作负载中除 471.omnetpp 外的程序对共享缓存和共享带宽的压力.

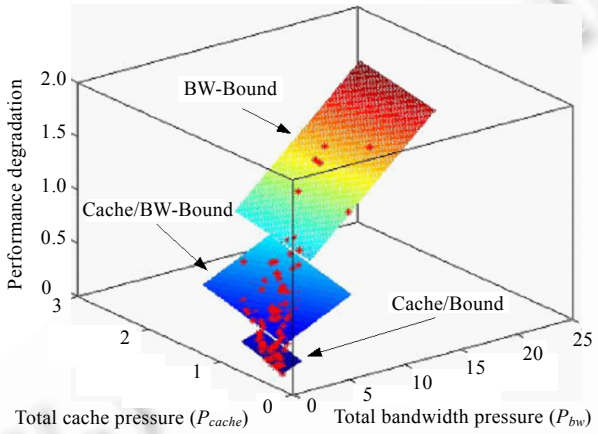

Fig.2 A piecewise function (the three planes corresponding to Eq.(3) and Eq.(4) relates 471.omnetpp's performance degradation of total pressures on shared cache and memory bandwidth

图 2 公式(3)、公式(4)代表的 3 个平面对应了一个线性分段函数,该函数反映了471.omnetpp 的性能下降与共享缓存和共享带宽的总压力的关系

以上述数据点为基础,我们选择总的带宽压力作为分段依据,考虑到我们实验平台最大的内存带宽为12.8GB/s,我们将数据分成了 3 个区间:(0,3.2GB/s),(3.2GB/s,9.6GB/s)和(9.6GB/s,+∞)(详见第 2.2 节),使用简单的分段线性回归的方法,我们获得了如公式(3)所示的线性分段函数:

$$PD_{omnetpp}=\begin{cases}PD_{Cache\text{-}Bound}, & P_{bw}\leqslant 3.2\\ PD_{Cache/BW\text{-}Bound}, & 3.2<P_{bw}\leqslant 9.6\\ PD_{BW\text{-}Bound}, & 9.6<P_{bw}\end{cases} \tag{3}$$

其中,

$$\begin{cases}PD_{Cache\text{-}Bound}=0.0289P_{bw}+0.2277P_{cache}-0.0715\\ PD_{Cache/BW\text{-}Bound}=0.0501P_{bw}+0.1830P_{cache}-0.1093\\ PD_{BW\text{-}Bound}=0.0976P_{bw}+0.1357P_{cache}-0.5056\end{cases} \tag{4}$$

其中,3 个子区间内函数的决定系数(R-square)分别为 0.90,0.92,0.95.由于没有采用标准化等手段,所以 P_{bw} 和 P_{cache} 的数值并不具有太大的含义,但是其相对数值仍旧反映了 471.omnetpp 对不同程度内存压力的不同敏感性.从公式中我们可以看出:在不同的区间,471.omnetpp 的性能下降随带宽压力和缓存压力的变化速率不同;而随着压力区间由 Cache-Bound 进入 Cache/BW-bound,带宽总压力的系数相对变大,而缓存总压力的系数相对减小,这意味着此时共享带宽的竞争对 471.omnetpp 的性能影响更大;而随着由 Cache/BW-Bound 进入 BW-Bound 区间,该系数的变化趋势再次出现,带宽的竞争成为影响 471.omnetpp 的主导因素.

2 主成分线性回归方法分析核间性能干扰

从上述观察现象可以看出,核间性能干扰可以表示为共同运行的所有程序对内存子系统压力的线性分段函数.据此,我们提出了一种基于统计学习的方法,使用主成分线性回归分析核间性能干扰.其主要内容包括两个方面:

1) 刻画应用程序对内存子系统的压力,即选择合适的指标来表示程序单独执行时对内存子系统共享资源的压力;
2) 刻画应用程序对内存子系统压力的敏感性,即通过主成分线性回归的方法来分析应用程序遭受的核间性能干扰与内存子系统总压力的关系.

我们的方法主要包括 3 个步骤:

1) 首先选取代表内存子系统压力的合适指标,刻画每个程序对内存子系统的压力.
2) 构造训练集数据,随机抽取包含目标程序在内的工作负载,将工作负载运行在实验室平台上,收集目标程序在不同的工作负载中的性能下降信息.
3) 使用主成分线性回归的方法构造每个程序的核间性能干扰模型,即目标程序的性能下降与内存子系统总压力的关系.

2.1 程序对内存子系统的压力

程序对内存子系统的压力,即程序单独执行在某一多核芯片上时对内存子系统共享资源的消耗,反映了程序对共享资源的竞争特性,一个程序对内存子系统的压力越大,则与它共同运行的程序遭受的核间性能干扰将会越发严重.在本文的工作中,关注的主要是共享缓存和共享内存带宽的竞争,使用的标准程序为 SPEC CPU2006 和 Synthetic Benchmarks.为了准确地衡量应用程序对共享缓存和共享内存带宽资源的竞争,我们利用处理器上提供的 PMU(performance monitoring unit)资源,借助于 Intel 的 Vtune[31]和 PTU[32]软件,采用如下的方式来刻画程序对这两种共享资源的压力:

- 程序对共享缓存的压力

为了准确地衡量程序对共享缓存的压力,考虑到我们的实验平台的硬件特征,即每个核有私有的 L1 和 L2 缓存,4 个核共享 L3 缓存,我们选择程序单独执行在该多核芯片上时单位时间内从共享 L3 缓存取入私有 L2 缓存的缓存行数[29],即 L2_LinesIn rate.该指标不仅可以代表由于私有缓存缺失引起的对共享缓存的访问,而且可以反映出数据预取对共享缓存的使用.以往的用来刻画程序对共享缓存行为的特征,如最后一级缓存未命中率,并不能很好地代表程序的竞争特性[29],而 L2_LinesIn rate 由于综合了私有缓存未命中和预取器的效果,更能代表程序对共享缓存的压力[29].为了测量程序单独执行时的 L2_LinesIn rate,我们使用了 Intel 的 Vtune 软件,通过对程序单独执行全过程的 profile,可以计算得到程序单独执行时的 L2_LinesIn rate,即程序对共享缓存的压力.

- 程序对共享内存带宽的压力

为了衡量程序对共享内存带宽的压力,我们选择了程序单独执行全过程中的平均带宽,包括读带宽和写带宽.为了测量每个程序单独执行时的带宽,我们使用了 Intel 的 PTU 软件,PTU 是 Intel 提供的一个性能分析软件,借助于处理器提供的 PMU 单元,可以周期性地给出系统的总带宽(system memory throughput).通过统计一个程序单独执行在该多核芯片上时全过程中的所有该采样值,即可计算出程序在单独执行时的平均带宽,此值可以代表程序对共享内存的压力.

- 两种压力的关系

我们需要明确的是,程序对共享资源的压力并不是完全独立的,而是有一定的相关性的.内存子系统资源作为一个整体,程序在使用内存子系统资源时并不是单独地使用某一部分的共享资源,而是将内存子系统作为一个整体在使用,这导致了应用程序对内存子系统的压力不会简单地按照共享资源分开,而是综合地作用在各个不同的共享资源上.具体到我们考虑的共享缓存和共享内存带宽来说,程序对这两种共享资源的压力有一定的相关性,如图 3 所示.图 3 中是我们使用的 SPEC 2006 中的程序对共享缓存的压力和对共享内存带宽压力的关

系.从中我们可以看出,两者具有很强的相关性,这种相关性促使我们使用主成分线性回归的方法.

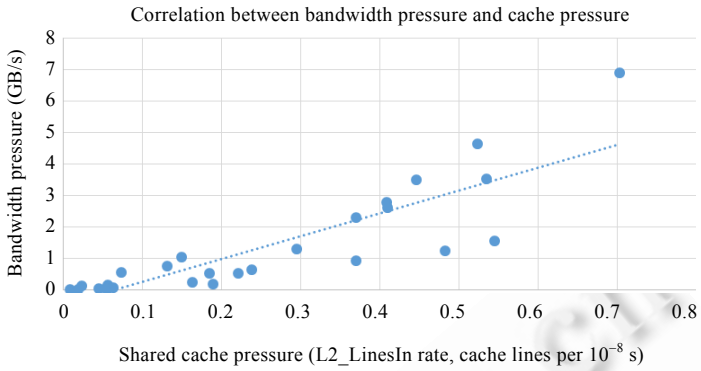

Fig.3 Correlation between bandwidth pressure and cache pressure, each point stands for one application from SPEC 2006

图 3 SPEC 2006 中的程序对共享缓存和共享内存带宽的压力呈现出了很强的相关性

2.2 程序对内存子系统压力的敏感性

为了刻画程序对内存子系统压力的敏感性,我们采用统计学习中的主成分线性回归的方法.即,我们首先通过将包含目标程序的不同的工作负载运行在实验平台上,收集目标程序的性能干扰数据;然后,使用主成分线性回归的方法处理我们收集到的数据,构造出目标程序的性能下降与内存子系统的总压力的线性模型.

2.2.1 性能干扰数据收集

为了使用线性回归的方法,我们需要收集足够的数据点.我们采用的方法如下:对于一个 m 核的处理器,我们从 28 个 SPEC 2006 程序和两个来自 Synthetic Benchmarks 的程序中随机抽取 $m-1$ 个程序,与目标程序组成一个工作负载.在实验平台上运行该工作负载,并且连续运行工作负载中的每一个程序,直至每个程序都至少完成了一次完整的执行[33],记录下目标程序第 1 次完整执行的时间,即为目标程序在该工作负载中的 $Time_{co\text{-}run}$.重复随机抽取和组成工作负载的步骤,直至不同工作负载的数目到达要求的阈值 Q(根据实际情况设立).对于每一个工作负载,我们根据公式(1)计算目标程序的性能下降 PD_{target},根据公式(5)计算该工作负载对内存子系统资源的压力 P_{cache} 和 P_{bw},则该工作负载对应数据点$\langle\langle P_{cache},P_{bw}\rangle,PD_{target}\rangle$.对于目标程序,我们会获得 Q 个这样的数据点,记这些数据点的集合为 D.

$$P_{cache}=\sum_{i=1}^{m}cache_i,\ P_{bw}=\sum_{i=1}^{m}bw_i \tag{5}$$

在我们的四核实验平台中,我们每次随机抽取 3 个程序与目标程序组成一个工作负载,考虑到我们要拟合的线性模型,对于每个目标程序,我们设定 Q 为 120,则对于每一个目标程序,我们都获得了 120 个用来拟合的数据点,将该数据点集合记为 D_{target}.

2.2.2 使用主成分线性回归方法构造核间性能干扰模型

从文章开始部分的观察现象可知,程序遭受的核间性能干扰可以表示为内存子系统总压力的线性分段函数,而与构成这些压力的具体应用程序无关,则使用公式(1)定义的性能下降代表目标程序的性能下降,用共享缓存和共享内存带宽的压力代表内存子系统的压力,目标程序遭受的核间性能干扰可由公式(6)表示:

$$PD_{target}=f(P_{cache},P_{bw})=\begin{cases}f_1(P_{cache},P_{bw}),\ P_{cache}\text{ and }P_{bw}\text{ in subdomian 1}\\ f_2(P_{cache},P_{bw}),\ P_{cache}\text{ and }P_{bw}\text{ in subdomian 2}\\ \vdots\\ f_n(P_{cache},P_{bw}),\ P_{cache}\text{ and }P_{bw}\text{ in subdomian }n\end{cases} \tag{6}$$

其中,PD_{target} 表示目标程序在所处的工作负载中由于核间性能干扰引起的性能下降,是待解释的因变量 Y; P_{cache}

和 P_{bw} 代表目标程序所在的工作负载对内存子系统的压力,由公式(5)计算,是自变量;$f(P_{cache},P_{bw})$代表要求的线性分段函数,可以表示为在 n 个子区间上的线性函数 $f_i(P_{cache},P_{bw})(i=1,\ldots,n)$,该函数代表了目标程序对内存子系统压力的敏感性.

我们在第 1.2.1 节中已经收集了 Q 个观测值,以这 Q 个观测值为基础,采用主成分线性回归的方法分别获取每个程序的 $f_i(P_{cache},P_{bw})$,从而得到每个程序对应的函数 $f(P_{cache},P_{bw})$,主要步骤包括:1) 选择合适的分段条件;2) 数据标准化;3) 主成分分析;4) 线性回归.

1) 分段条件的确定

分段的函数允许我们的模型更为精确地捕捉到程序对不同程度的内存子系统压力的敏感性的差异,从而可以构造出更为精确的模型.所以,确定分段点或者说确定 n 个子区间成为一个关键的问题.最终,我们选择了共享带宽作为我们的分段条件,主要原因在于:1) 对共享带宽的竞争是引起核间性能干扰的主要部分[6],选择带宽作为分段依据可以很好地区分内存子系统资源的竞争程度;2) 共享内存带宽的压力和共享缓存的压力具有很强的相关性,选择共享内存带宽作为分段依据同样可以体现出共享缓存上承受的压力.考虑到我们使用的实验平台的最大内存带宽为 12.8GB/s,我们将目标程序的 Q 个数据点集合 D_{target} 根据 P_{bw} 分入 3 个子区间:(0,3.2GB/s],(3.2GB/s,9.6GB/s]和(9.6GB/s,+∞),分别记为 D_{t1},D_{t2},D_{t3},则目标程序对应的函数 $f(P_{cache},P_{bw})$将由这 3 个区间上的子函数确定.

2) 数据标准化

为了忽略我们选择的程序对内存子系统压力指标的量纲差异,我们需要首先对待拟合数据进行标准化处理.具体到我们的方法来说,对 $D_{ti}(i=1,2,3)$中的自变量数据 P_{cache} 和 P_{bw} 进行标准化处理,变为 Z_{cache} 和 Z_{bw}:

$$\begin{bmatrix} Z_{cache} \\ Z_{bw} \end{bmatrix} = \begin{bmatrix} \dfrac{P_{cache}-\overline{P_{cache}}}{\sigma(P_{cache})} \\ \dfrac{P_{bw}-\overline{P_{bw}}}{\sigma(P_{bw})} \end{bmatrix}, \forall \langle\langle P_{cahe},P_{bw}\rangle,PD_{target}\rangle \in D_{ti} \tag{7}$$

经过标准化处理以后,D_{ti} 中的数据点形式变为$\langle\langle Z_{cache},Z_{bw}\rangle,PD_{target}\rangle$.

3) 主成分分析

在衡量程序对内存子系统的压力时我们已经发现,程序对共享缓存和共享内存带宽的压力具有很强的相关性,而这种相关性同样体现在我们的待拟合数据点上:我们对 471.omnetpp 的 D_{t1} 中的 P_{cache} 和 P_{bw} 的相关性进行了计算,发现两者的相关性高达 0.86,如果直接使用这两种压力指标作为自变量进行线性回归,则会因为“共线性”的问题导致拟合结果对输入自变量数据的高度敏感,影响模型的准确性[34].因此,我们需要对标准化后的变量 Z_{cache} 和 Z_{bw} 进行主成分分析,得到互相独立的新变量 X_1 和 X_2.对 471.omnetpp 的 D_{t1} 中的样本数据,我们进行了主成分分析,结果如下:

$$\begin{bmatrix} X_1 & X_2 \end{bmatrix} = \begin{bmatrix} Z_{cache} & Z_{bw} \end{bmatrix} \begin{bmatrix} 0.7071 & -0.7071 \\ 0.7071 & 0.7071 \end{bmatrix},$$

其中,第 1 个主成分 X_1 解释了 88.36%的方差,第 2 个主成分 X_2 解释了 11.64%的方差.

4) 线性回归

在进行过分段以及数据标准化和主成分分析以后,我们可以建立如下的回归方程:

$$PD_{target} = \begin{cases} b_{10}+b_{11}\times X_1+b_{12}\times X_2, & \text{subdomian 1} \\ b_{20}+b_{20}\times X_1+b_{22}\times X_2, & \text{subdomian 2} \\ \vdots & \\ b_{n0}+b_{n1}\times X_1+b_{n2}\times X_2, & \text{subdomian } n \end{cases} \tag{8}$$

对于每一个子区间,设落入该区间的数据点集合为 D_{ti},数目为 Q_{ti},则该子区间对应的回归方程可以表示为

$$\begin{bmatrix} pd_1 \\ pd_2 \\ \vdots \\ pd_{Q_{ti}} \end{bmatrix} = \begin{bmatrix} 1 & x_{11} & x_{12} \\ 1 & x_{21} & x_{22} \\ \vdots & \vdots & \vdots \\ 1 & x_{Q_{ti}1} & x_{Q_{ti}2} \end{bmatrix} \begin{bmatrix} b_{i0} \\ b_{i1} \\ b_{i2} \end{bmatrix} \tag{9}$$

其中,pd_i是样本数据中的PD_{target}观察值.根据公式(8)和公式(9),确定系数b_{i0},b_{i1},b_{i2}时,为了得到更加精确和简洁的方程,我们会采取如下两种方法:

a) 逐步回归

逐步回归是指对于我们经过主成分分析得到的两个自变量X_1和X_2,拟合结束时,我们会对其对应的系数进行t检验:如果能够通过t检验,则最终的方程会包含该自变量;如果未能通过t检验,说明待解释变量与该自变量在某一显著性水平下不显著线性相关,则从方程中删除该自变量,重新用剩余的变量进行拟合,得到新的方程.也就是说,我们最终得到的方程可能含有两个或者少于两个的自变量.

b) 奇点过滤

奇点过滤是指单次拟合结束以后,对于每一组观测值,根据其残差和置信区间判断该观测值是否显著为奇点,如果其残差的置信区间不包含 0,则说明在给定的显著性水平下,该观测值为明显的奇点,将其从待拟合数据集中删除,重新进行拟合,直到在给定的显著性水平下无明显奇点为止.

采用上述方法,按照我们的分段规则分段并进行数据标准化和主成分分析以后,在 0.05 的显著性水平下,使用逐步归回和奇点过滤方法,471.omnetpp 遭受的核间性能干扰可以表示为

$$PD_{omnetpp} = \begin{cases} 0.139 + 0.034 \times P_{cache} + 0.014 \times P_{bw}, & P_{bw} \leqslant 3.2 \\ 0.362 + 0.039 \times P_{cache} + 0.059 \times P_{bw}, & 3.2 < P_{bw} \leqslant 9.6 \\ 1.067 + 0.023 \times P_{cache} + 0.269 \times P_{bw}, & 9.6 < P_{bw} \end{cases} \tag{10}$$

3 模型验证

在我们的实验平台上,使用我们的基于主成分线性回归的核间性能干扰分析方法,针对 SPEC CPU 2006 中的 20 个程序得到了其遭受的核间性能干扰的模型.本节首先介绍我们的实验平台,然后对我们获得的模型进行验证并分析模型的训练误差和测试误差.

3.1 实验平台简介

我们的实验平台包含一个四核的 Intel Xeon E5506 处理器,主频为 2.13GHz,每个核拥有 32KB 的 L1 数据缓存、32KB 的 L1 指令缓存、256KB 的 L2 缓存以及 4 个核共享的 4MB 的最后一级缓存(L3).该平台的内存带宽为 12.8GB/s(使用了 3 个内存通道中的两个).实验平台参数见表 1.

Table 1 Experimental environment parameters

表 1 实验环境参数

参数名称	参数值
处理器	Intel Xeon E5506
最大内存带宽	12.8GB/s
操作系统	Linux (kernel 2.6.18)
编译器	GCC 4.4
基准程序	SPEC CPU 2006,使用 ref 输入集,编译选项为“-o3 -static”;Synthetic Benchmarks,编译选项同样为“-o3 -static”

3.2 模型验证

我们分析了 SPEC 2006 的所有程序及 Synthetic Benchmarks 对内存子系统的压力,并使用我们的主成分线性回归的方法分析了 SPEC 2006 中 20 个程序对核间性能干扰的敏感性,我们选择的Q为 120,获得的部分结果见表 2.其中,“–”代表在该子区间不存在对应的函数,因为目标程序本身对内存子系统的压力已经不在对应的子区间内;P_{cache}和P_{bw}分别代表标准化后的工作负载对共享缓存和共享内存带宽的压力.

Table 2 Experimental model for some SPEC 2006 benchmarks

表 2 部分 SPEC 2006 程序获得的实验模型

Benchmark	Cross-Core interference predictor
433.milc	–, $P_{bw} \leqslant 3.2$ $0.168+0.018\times P_{cache}+0.062\times P_{bw}$, $3.2<P_{bw} \leqslant 9.6$ $0.780+0.037\times P_{cache}+0.308\times P_{bw}$, $9.6<P_{bw}$
435.gromacs	$0.021+0.004\times P_{cache}+0.007\times P_{bw}$, $P_{bw} \leqslant 3.2$ $0.080+0.010\times P_{cache}+0.018\times P_{bw}$, $3.2<P_{bw} \leqslant 9.6$ $0.231+0.011\times P_{cache}+0.049\times P_{bw}$, $9.6<P_{bw}$
437.leslie3d	$0.027+0.006\times P_{cache}+0.006\times P_{bw}$, $P_{bw} \leqslant 3.2$ $0.086+0.009\times P_{cache}+0.026\times P_{bw}$, $3.2<P_{bw} \leqslant 9.6$ $0.424+0.023\times P_{cache}+0.149\times P_{bw}$, $9.6<P_{bw}$
450.soplex	–, $P_{bw} \leqslant 3.2$ $0.155+0.019\times P_{cache}+0.048\times P_{bw}$, $3.2<P_{bw} \leqslant 9.6$ $0.687+0.054\times P_{cache}+0.236\times P_{bw}$, $9.6<P_{bw}$
470.lbm	–, $P_{bw} \leqslant 3.2$ $0.106+0.017\times P_{cache}+0.031\times P_{bw}$, $3.2<P_{bw} \leqslant 9.6$ $0.600+0.046\times P_{cache}+0.286\times P_{bw}$, $9.6<P_{bw}$
471.omnetpp	$0.139+0.034\times P_{cache}+0.014\times P_{bw}$, $P_{bw} \leqslant 3.2$ $0.363+0.039\times P_{cache}+0.059\times P_{bw}$, $3.2<P_{bw} \leqslant 9.6$ $1.067+0.023\times P_{cache}+0.269\times P_{bw}$, $9.6<P_{bw}$

对于全部的20个被分析程序来说,所有的子区间上的函数都通过了F检验;在采用了逐步回归以后,所有的主成分的系数都通过了t检验,这意味着程序遭受的核间性能干扰确实是与内存子系统的总压力显著线性相关的.在拟合优度方面,我们一共拟合了 60 个函数(每个程序 3 个子区间,每个子区间 1 个函数),其中,53 个函数的拟合优度大于 0.9,而最小的拟合优度也有 0.84,这意味着我们的模型是合理的,与数据契合得非常好.我们以437.leslie3d 为例进行具体说明:在第 1 个子区间,第 2 个主成分 $X_2=0.7071\times P_{bw}-0.7071\times P_{cache}$ 的系数未能通过 t 检验,我们采用逐步回归的办法剔除了第 2 个主成分,仅使用第 1 个主成分 $X_1=0.7071\times P_{bw}+0.7071\times P_{cache}$ 进行拟合,得到了第 1 个子区间对应的函数.该函数通过了 F 检验和 t 检验,拟合优度为 0.99.此时,P_{cache} 和 P_{bw} 的系数都为 0.006 1.在第 2 个子区间,两个主成分都通过了 t 检验,函数也通过了 F 检验,拟合优度为 0.95,此时,P_{cache} 和 P_{bw} 的系数分别为 0.009 9 和 0.026.我们注意到,两者的系数都在上升,而 P_{bw} 的系数明显上升得更快,绝对大小也更大,这意味着此时的内存子系统资源的竞争程度已经由 Cache-Bound 进入 Cache/Bw-Bound 区间.在第 3 个子区间,函数通过了所有的检验,拟合优度为 0.96,此时,P_{cache} 和 P_{bw} 的系数为 0.023 和 0.15,两者的系数仍旧在增大,但是 P_{bw} 的系数相对于 P_{cache} 的增加更为迅速,绝对大小也更大,这意味对带宽资源的竞争成为核间性能干扰的主导部分,内存子系统的资源竞争进入 Bw-Bound 阶段.

3.3 模型误差分析

每个程序遭受的核间性能干扰模型可以用来预测程序在不同的内存子系统压力下的性能下降,所以我们分析两个方面的误差:一方面是模型在训练集上的误差,另一方面是测试的误差.我们定义误差为模型预测的性能下降与实际的性能下降的差的绝对值,称为绝对误差.

- 训练集误差

对于每个程序,我们计算其在 3 个子区间上的总的平均绝对误差,最小的平均绝对误差仅为 0.18%,而最大的平均绝对误差为 3.4%.在 20 个程序中,12 个程序的平均绝对误差都小于 1%,这意味着我们的模型和训练集数据是相符的.

- 测试集误差

对于每个程序,我们随机生成了 10 个新的包含该程序的工作负载,每个工作负载由 4 个程序组成,预测该程

序在这 10 个新的工作负载中的性能下降.对于总共 20 个待分析程序,总共生成了 200 个新的工作负载,我们从中选取了 50 个工作负载,结果如图 4 所示.

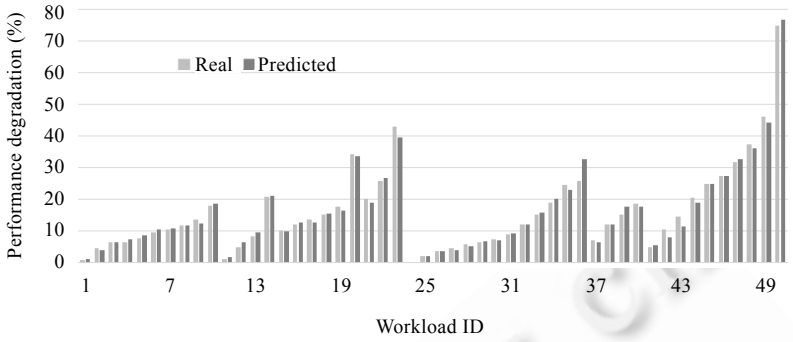

Fig.4 Prediction precision for 50 workloads out of 200 workloads generated from 20 benchmarks, for each workload, target benchmark's performance slowdown is shown

图 4 从 20 个标准程序中随机生成了 200 个工作负载,其中 50 个目标程序的实际性能下降与模型预测的性能下降的对比

图 4 中画出了每个工作负载中目标程序的实际性能下降(浅色)和我们的方法预测的性能下降(深色).从中可以看出:大部分情况下,我们的预测值与实际值非常接近.就全部的 200 个工作负载而言,总的平均预测误差为 1.1%,预测的绝对误差变化区间为 0%~10.7%.表 3 是我们的绝对误差分布.从表 3 中我们可以看出,绝大部分工作负载的绝对误差小于 1%,而大于 3%的误差只有 15 组.

Table 3 Absolute error of our method

表 3 本方法的绝对误差

Absolute error	<1%	1%~3%	>3%
#workloads	143	42	15

3.4 与Bubble-Up方法的对比

在这部分的实验中,将我们方法的结果与 Bubble-Up 方法[3]进行对比.Bubble-Up 可以用来量化分析程序间的性能干扰,但是只适用于两个程序共同运行的情况,因此我们设计了两个程序共同运行的新实验.由于 Bubble-Up 使用的标准程序并不公开,我们使用 SPEC 2006 中的程序予以替代.具体来说,我们从 SPEC 2006 中随机抽取了 7 个测试程序,将这 7 个程序与每个程序组成新的工作负载,并统计工作负载中程序的性能下降,最后用所有程序的预测结果与 Bubble-Up 方法进行对比.对于总共 20 个待分析程序,总共生成了 140 个工作负载,从中随机抽取 50 个,结果如图 5 所示.

从图 5 中可以看出:对于两个程序共同运行的情况,我们的方法仍旧可以取得很好的预测结果,对总共 140 个工作负载的平均预测误差为 1.9%.对于总共 20 个程序,8 个程序的平均误差小于 1%,17 个程序的平均误差小于 3%,而 Bubble-Up 方法总共给出了对 9 个程序的平均预测误差,为 1.4%.也就是说,我们方法的误差略大于 Bubble-Up 方法,但是两者基本处于同一水平.如前所述,Bubble-Up 方法不能预测多于两个程序共同运行时的性能干扰,而我们的方法则可以预测任意个程序之间的影响,因此,我们在保持 Bubble-Up 预测精度的基础上扩大了预测模型的适用面.

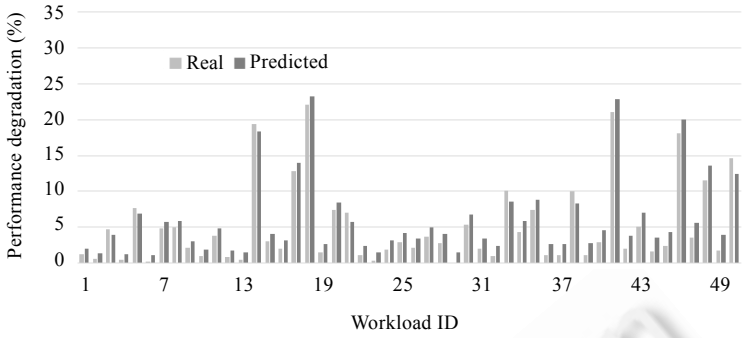

Fig.5 Prediction precision for 50 workloads out of 140 workloads, each of which contains two benchmarks, for each workload, target benchmark's performance slowdown is shown

图 5 在我们生成的包含两个程序的 140 个工作负载中,其中 50 个目标程序的实际性能下降和模型预测的性能下降的对比

3.5 模型的通用性

为了验证我们的方法和模型的通用性,我们在另外的 X-86 平台上针对 SPEC 2006 进行了类似的实验.两个实验平台分别为 AMD 的 Opteron 8374(四核,私有的 L1 和 L2 缓存,共享的 L3 缓存,最大内存带宽为 12.8GB/s)和 Intel 的 Xeon E7-4807(六核,私有的 L1 和 L2 缓存,共享的 L3 缓存,最大内存带宽为 25.6GB/s).在这两个平台上,我们取得了类似的结果,都可以使用线性分段模型来表示程序遭受的核间性能干扰.对 AMD 的四核平台,我们方法的测试误差范围为 0~5.1%,平均为 0.3%;对 Intel 的六核平台,我们方法的测试误差范围为 0~10.2%,平均为 0.1%.实验结果表明,我们的方法及模型对于不同的 X-86 平台具有较好的通用性.

4 总 结

本文提出了一种基于统计学习的核间性能干扰分析方法,使用主成分线性回归的方法获得干扰模型,可以量化和精确地预测程序由于核间性能干扰导致的性能下降.通过将程序对共享缓存和共享带宽的压力视为程序对内存子系统资源的压力,本文将程序遭受的核间性能干扰表示为内存子系统总压力的线性分段函数,并提出了使用主成分线性回归的方法来获得每个程序对应的函数.实验数据表明,本方法的预测平均绝对误差仅为 1.1%.

References:

[1] Sankaralingam K, Arpaci-Dusseau RH. Get the parallelism out of my cloud. In: Proc. of the 2nd USENIX Conf. on Hot Topics in Parallelism. Berkeley: USENIX Association, 2010. 1−6.

[2] Fedorova A, Seltzer M, Smith MD. Improving performance isolation on chip multiprocessors via an operating system scheduler. In: Anon Proc. of the 16th Int'l Conf. on Parallel Architecture and Compilation Techniques (PACT 2007). Piscataway: Institute of Electrical and Electronics Engineers Inc., 2007. 25−36. [doi: 10.1109/PACT.2007.18]

[3] Mars J, Tang LJ, Hundt R, Skadron K, Soffa ML. Bubble-Up: Increasing utilization in modern warehouse scale computers via sensible co-locations. In: Proc. of the 44th Annual IEEE/ACM Symp. on Microarchitecture (MICRO 44). Piscataway: IEEE Computer Society, 2011. 248−259. [doi: 10.1145/2155620.2155650]

[4] Tang LJ, Mars J, Soffa ML. Compiling for niceness: Mitigating contention for QoS in warehouse scale computers. In: Proc. of the 10th Int'l Symp. on Code Generation and Optimization (CGO 2012). Association for Computing Machinery, 2012. 1−12. [doi: 10.1145/2259016.2259018]

[5] Tang LJ, Mars J, Vachharajani N, Hundt R, Soffa ML. The impact of memory subsystem resource sharing on datacenter applications. In: Proc. of the 38th Annual Int'l Symp. on Computer Architecture (ISCA 2011). New York: Institute of Electrical and Electronics Engineers Inc., 2011. 283−294. [doi: 10.1145/2000064.2000099]

[6] Xu D, Wu CG, Yew PC. On mitigating memory bandwidth contention through bandwidth-aware scheduling. In: Proc. of the 19th Int'l Conf. on Parallel Architectures and Compilation Techniques (PACT 2010). Piscataway: Institute of Electrical and Electronics Engineers Inc., 2010. 237−247. [doi: 10.1145/1854273.1854306]

[7] Zhuravlev S, Blagodurov S, Fedorova A. Addressing shared resource contention in multicore processors via scheduling. In: Proc. of the 15th Int'l Conf. on Architectural Support for Programming Languages and Operating Systems (ASPLOS XV). New York: Association for Computing Machinery, 2010. 129−141. [doi: 10.1145/1736020.1736036]

[8] Barroso LA, Holzle U. The case for energy-proportional computing. Computer, 2007,40(12):33−37. [doi: 10.1109/MC.2007.443]

[9] Chandra D, Guo F, Kim S, Solihin Y. Predicting inter-thread cache contention on a chip multi-processor architecture. In: Proc. of the 11th Int'l Symp. on High-Performance Computer Architecture (HPCA-11 2005). New York: Institute of Electrical and Electronics Engineers Computer Society, 2005. 340−351. [doi: 10.1109/HPCA.2005.27]

[10] Jiang YL, Jie C, Shen XP, Tripathi R. Analysis and approximation of optimal co-scheduling on chip multiprocessors. In: Proc. of the 17th Int'l Conf. on Parallel Architectures and Compilation Techniques (PACT 2008). Piscataway: Institute of Electrical and Electronics Engineers Inc., 2008. 220−229. [doi: 10.1145/1454115.1454146]

[11] Knauerhase R, Brett P, Hohlt B, Li T, Hahn S. Using OS observations to improve performance in multicore systems. IEEE Micro, 2008,28(3):54−66. [doi: 10.1109/MM.2008.48]

[12] Machina J, Sodan A. Predicting cache needs and cache sensitivity for applications in cloud computing on cmp servers with configurable caches. In: Proc. of the 23rd IEEE Int'l Parallel and Distributed Processing Symp. (IPDPS 2009). Piscataway: IEEE Computer Society, 2009. 1−8. [doi: 10.1109/IPDPS.2009.5161233]

[13] Xu C, Chen X, Dick RP, Mao ZQM. Cache contention and application performance prediction for multi-core systems. In: Proc. of the 2010 IEEE Int'l Symp. on Performance Analysis of Systems and Software (ISPASS 2010). Piscataway: IEEE Computer Society, 2010. 76−86. [doi: 10.1109/ISPASS.2010.5452065]

[14] Si CX, Meng XX, Xu L. Research on cache optimization technology based on performance insulation. Journal of Computer Research and Development, 2011,48(z1) (in Chinese with English abstract).

[15] Wang Z. Dynamic cache Partitioning under CMP structure [MS. Thesis]. Changchun: Jilin University, 2011 (in Chinese with English abstract).

[16] Eklov D, Nikoleris N, Black-Schaffer D, Hagersten E. Bandwidth bandit: Quantitative characterization of memory contention. In: Proc. of the 11th IEEE/ACM Int'l Symp. on Code Generation and Optimization (CGO 2013). Washington: IEEE Computer Society, 2013. 99−108. [doi: 10.1109/CGO.2013.6494987]

[17] Subramanian L, Seshadri V, Kim YG, Jaiyen B, Mutlu O. MISE: Providing performance predictability and improving fairness in shared main memory systems. In: Proc. of the 19th IEEE Int'l Symp. on High Performance Computer Architecture (HPCA 2013). Washington: IEEE Computer Society, 2013. 639−650. [doi: 10.1109/HPCA.2013.6522356]

[18] Guo LJ, Liu X, Zhao JY, Shi ZZ. Pedestrian detection method of integrated motion information and appearance features. Ruan Jian Xue Bao/Journal of Software, 2012,23(2):299−309 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/4030.htm [doi: 10.3724/SP.J.1001.2012.04030]

[19] Li TZ, Ding XQ, Fang C. A novel wheelchair video detection method based on a combined appearance and motion pattern. Chinese High Technology Letters, 2012,22(2):153−158 (in Chinese with English abstract).

[20] Zhang HL, Lu G. Machine learning algorithms for classifying the imbalanced protocol flows: Evaluation and comparison. Ruan Jian Xue Bao/Journal of Software, 2012,23(6):1500−1516 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/4074.htm [doi: 10.3724/SP.J.1001.2012.04074]

[21] Liu Q, Liu Z, Huang M. Study on Internet traffic classification using machine learning. Computer Science, 2010,37(12):35−40 (in Chinese with English abstract).

[22] Li SM, Chen BQ, Gao XY, Qiao L, Tang ZZ. Principal component linear regression analysis on performance of applications. Journal of Computer Research and Development, 2009,46(11):1949−1955 (in Chinese with English abstract).

[23] Agakov F, Bonilla E, Cavazos J, Franke B, Fursin G, O'Boyle MFP, Thomson J, Toussaint M, Williams CKI. Using machine learning to focus iterative optimization. In: Proc. of the Int'l Symp. on Code Generation and Optimization. Piscataway: IEEE Computer Society, 2006. 295−305. [doi: 10.1109/CGO.2006.37]

[24] Kulkarni S, Cavazos J. Mitigating the compiler optimization phase-ordering problem using machine learning. In: Proc. of the 2012 ACM SIGPLAN Conf. on Object-Oriented Programming, Systems, Languages, and Applications, OOPSLA 2012. New York: Association for Computing Machinery, 2012. 147−162. [doi: 10.1145/2398857.2384628]

[25] Kulkarni S, Cavazos J, Wimmer C, Simon D. Automatic construction of inlining heuristics using machine learning. In: Proc. of the 11th IEEE/ACM Int'l Symp. on Code Generation and Optimization (CGO 2013). Washington: IEEE Computer Society, 2013. [doi: 10.1109/CGO.2013.6495004]

[26] Fursin G, Kashnikov Y, Memon AW, Chamski Z, Temam O, Namolaru M, Yom-Tov E, Mendelson B, Zaks A, Courtois E, Bodin F, Barnard P, Ashton E, Bonilla E, Thomson J, Williams CKI, O'Boyle M. Milepost GCC: Machine learning enabled self-tuning compiler. Int'l Journal of Parallel Programming, 2011,39(3):296−327. [doi: 10.1007/s10766-010-0161-2]

[27] Leather H, Bonilla E, O'Boyle M. Automatic feature generation for machine learning based optimizing compilation. In: Proc. of the 7th Int'l Symp. on Code Generation and Optimization (CGO 2009). New York: Institute of Electrical and Electronics Engineers Computer Society, 2009. 81−91. [doi: 10.1109/CGO.2009.21]

[28] Dubach C, Jones TM, Bonilla EV, Fursin G, O'Boyle MFP. Portable compiler optimisation across embedded programs and microarchitectures using machine learning. In: Proc. of the 42nd Annual IEEE/ACM Int'l Symp. on Microarchitecture (Micro-42). Piscataway: IEEE Computer Society, 2009. 78−88. [doi: 10.1145/1669112.1669124]

[29] Tang LJ, Mars J, Soffa ML. Contentiousness vs. sensitivity: Improving contention aware runtime systems on multicore architectures. In: Proc. of the 1st Int'l Workshop on Adaptive Self-Tuning Computing Systems for the Exaflop Era. New York: Association for Computing Machinery, 2011. 12−21. [doi: 10.1145/2000417.2000419]

[30] Mars J, Soffa ML. Synthesizing contention. In: Proc. of the Workshop on Binary Instrumentation and Applications (WBIA 2009). New York: Association for Computing Machinery, 2009. 17−25. [doi: 10.1145/1791194.1791197]

[31] Intel. Intel vtune amplifier XE. http://software.intel.com/en-us/intel-vtune-amplifier-xe

[32] Intel. Intel performance tuning utility. http://software.intel.com/en-us/articles/intel-performance-tuning-utility

[33] Tian K, Jiang YL, Shen XP. A study on optimally co-scheduling jobs of different lengths on chip multiprocessors. In: Proc. of the 6th ACM Conf. on Computing Frontiers. ACM Press, 2009. 41−50. [doi: 10.1145/1531743.1531752]

[34] Farrar DE, Glauber RR. Multicollinearity in regression analysis: The problem revisited. The Review of Economics and Statistics, 1967,49(1):92−107. [doi: 10.2307/1937887]

附中文参考文献:

[14] 司成祥,孟晓烜,许鲁.基于性能隔离的缓存优化技术研究.计算机研究与发展,2011,48(z1).

[15] 王震.CMP 架构下的共享 Cache 动态划分[硕士学位论文].长春:吉林大学,2011.

[18] 郭立君,刘曦,赵杰煜,史忠植.结合运动信息与表观特征的行人检测方法.软件学报,2012,23(2):299−309. http://www.jos.org.cn/1000-9825/4030.htm [doi: 10.3724/SP.J.1001.2012.04030]

[19] 李同治,丁晓青,方驰.综合表观和运动模式的视频轮椅检测方法.高技术通讯,2012,22(2):153−158.

[20] 张宏莉,鲁刚.分类不平衡协议流的机器学习算法评估与比较.软件学报,2012,23(6):1500−1516. http://www.jos.org.cn/1000-9825/4074.htm [doi: 10.3724/SP.J.1001.2012.04074]

[21] 刘琼,刘珍,黄敏.基于机器学习的 IP 流量分类研究.计算机科学,2010,37(12):35−40.

[22] 李胜梅,程步奇,高兴誉,乔林,汤志忠.主成分线性回归模型分析应用程序性能.计算机研究与发展,2009,46(11):1949−1955.

赵家程(1989－),男,河南新乡人,博士生,CCF 学生会员,主要研究领域为并行计算,并行编译,并行编程环境.
E-mail: zhaojiacheng@ict.ac.cn

冯晓兵(1969－),男,博士,研究员,博士生导师,CCF 高级会员,主要研究领域为先进编译技术及相关工具环境.
E-mail: fxb@ict.ac.cn

崔慧敏(1979－),女,博士,副研究员,CCF 会员,主要研究领域为并行计算,并行编译,并行编程.
E-mail: cuihm@ict.ac.cn