高准确率的应用程序行为分析方法

赵天磊，唐遇星，齐树波，付桂涛，贾小敏，张民选

(国防科技大学计算机学院 长沙 410073)

【摘要】模拟是体系结构研究的重要手段。由于模拟的速度非常慢，有研究提出利用动态二进制翻译技术(DBT)提取程序的代表性模拟点，对代表性模拟点进行详细模拟即可获取程序的准确性能参数，从而缩短模拟时间。然而相关研究并未考虑DBT方法对模拟结果准确度的影响。实验发现，对于某些程序，DBT加速方法会给模拟结果带来近20%的误差。为消除误差，系统分析了引起指令踪迹差异的原因，并提出了一系列消除这些差异的方法。实验结果证实，利用该文提出的方法，可以在不影响DBT方法性能的同时，保证其与传统模拟方法具有完全相同的精确度。

关 键 词 BBV Profile; 二进制翻译; 执行踪迹; 应用行为分析; 可重现模拟; SimPoint; 模拟误差
中图分类号 TP314 文献标识码 A doi:10.3969/j.issn.1001-0548.2011.06.023

Program Behavior Analysis Method with High Accuracy

ZHAO Tian-lei, TANG Yu-xing, QI Shu-bo, FU Gui-tao, JIA Xiao-min, and ZHANG Min-xuan

(School of Computer, National University of Defence Technology Changsha 410073)

Abstract Simulation is a very important way to computer architecture research. To reduce simulation time, the dynamic binary translation (DBT) technique has bee proposed to extract representative simulation points of programs. The performance characteristics of programs can then be obtained by simulating only the extracted representative simulation points. However, the accuracy implications of the DBT method have not been studied. For some programs, the DBT method can incur a nearly 20% error in the simulation results. In this paper, the cause of the execution trace deviation is studied carefully and several methods are proposed to overcome the problem. Experimental results show that with the proposed methods, the accuracy of the DBT method is the same as the simulation method.

Key words BBV Profile; binary translation; execution trace; program behavior analysis; reproducible simulation; SimPoint; simulation error

模拟是体系结构研究的重要手段，主要有两方面的应用：1) 评估体系结构优化技术的效果，2) 分析应用程序的行为特征[1]。然而模拟器的运行速度非常慢，功能模拟的速度通常只有一千万条指令每秒左右，性能模拟的速度只有几十万条指令每秒。为提高模拟速度，研究人员提出使用动态二进制翻译(DBT)技术提取应用程序的BBV (basic block vector) Profile[2-5]，然后再利用SimPoint工具对BBV Profile进行处理，提取出能够代表程序整体行为的代表性模拟点[6-7]。利用DBT技术可以大大加速代表性模拟点的提取过程，减少整个模拟过程所需的时间，但现有研究都没有分析其对模拟结果准确性的影响。

本文实验发现，DBT方法对模拟结果准确度有较大的影响，对部分测试程序会带来近20%的模拟误差。经深入分析，发现误差产生的根源是程序在DBT执行和模拟执行时执行踪迹的巨大差异。进一步分析发现，程序的执行踪迹差异是由两种执行环境中一些参数设置上的细微差异引起的，与文献[8-9]的发现类似。文献[8]分析了体系结构研究中实验的测量偏倚问题，指出设置实验环境时，看似无关的细微差异都会对实验结果产生重大影响，甚至会改变实验的结论。然而文献[8-9]并没有系统地分析哪些因素可能会对实验结果造成较大的影响。

本文根据进程的执行模型，从进程的初始状态和状态更新过程入手，提出了一种能确定影响程序行为所有因素的方法。实验结果证实，文中所提方法可以有效地消除程序在各种执行环境下的执行踪

收稿日期：2011－06－10； 修回日期：2011－09－24
基金项目：国家自然科学基金(60970036)；教育部博士点基金(20094307120007)
作者简介：赵天磊(1982－)，男，博士生，主要从事多核处理器、体系结构模拟技术、二进制翻译等方面的研究.

迹差异，从而保证DBT方法与传统模拟方法具有完全相同的准确性。

1 DBT方法对模拟结果精度的影响

1.1 基于DBT加速的模拟流程

基于DBT加速的模拟是对传统上基于功能模拟器(ISA Simulator)的SimPoint模拟方法的一种加速。图1所示为SimPoint方法的模拟流程。1) 利用基于功能模拟器或基于DBT技术的BBV Profiler将测试程序完整执行一遍，在执行过程中收集测试程序的BBV Profile(程序执行过程的一种表示形式)；2) 利用SimPoint工具对BBV Profile 进行分析，提取能够代表程序整个执行过程的一些模拟点；3) 利用性能模拟器(performance simulator)对提取的代表性模拟点进行详细的性能模拟并得到最终结果。

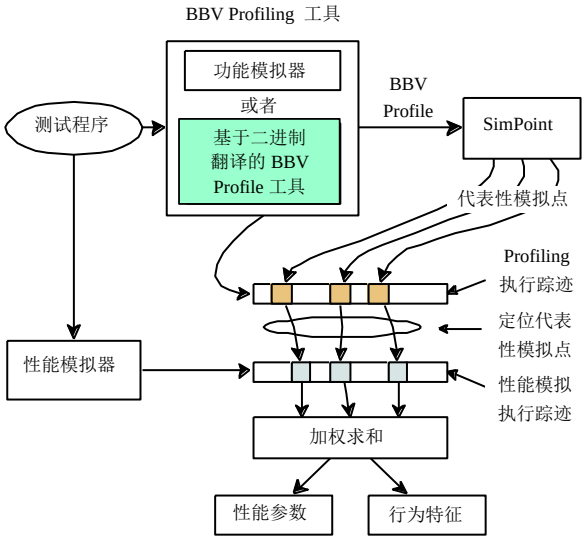

图 1 利用动态二进制翻译技术加速的 SimPoint 模拟流程

由于收集BBV Profile时需要将测试程序完整执行一遍，对于像SPEC2006那样的现代测试程序集，如果使用功能模拟器，可能需要数月时间。而基于DBT的BBV Profiler的速度可以达到功能模拟器的10倍左右[10]，能够大大加速测试过程。

从图1中可以看出，在DBT加速的模拟流程中，测试程序一共被执行了两次。第一次是在基于DBT的BBV Profiler中执行的，用于收集BBV Profile；第二次是在性能模拟器上执行的，用于得到程序的性能或行为特性数据。在第二次执行过程中，需要准确地定位第一次执行时得到的代表性模拟点。定位模拟点有两种方法：1) 根据指令数(icount)定位；2) 根据关键指令PC+执行次数(marker)[6]定位。两种方法都要求两次执行的指令踪迹不能相差太大，否则会引起巨大的定位偏差。一般来说，定位偏差应该远小于模拟点大小(模拟点大小一般在10^7～10^8条指令之间[6])。然而本文实验发现，对于很多程序，基于DBT的执行和基于模拟器的执行会有很大的指令踪迹差异，且这些指令踪迹差异会对模拟结果的准确性造成较大的影响。

表1 所有SPEC2006测试程序分别在QPoint和Simics上执行时的提交指令总数比较

Benchmark	Simics	QPoint	Delta(Δ)
400.perlbench	762 816 940 717	763 456 297 927	639 357 210
401.bzip2	535 110 832 014	535 110 834 361	2 347
403.gcc	83 792 871 309	84 764 827 875	971 956 566
410.bwaves	4 109 187 803 249	4 109 188 163 883	360 634
416.gamess	4 477 015 763 260	4 477 015 771 387	8 127
429.mcf	439 350 891 791	439 351 002 443	110 652
433.milc	1 162 698 380 427	1 162 829 752 454	131 372 027
434.zeusmp	2 315 700 510 347	2 315 700 517 520	7 173
435.gromacs	1 754 307 479 669	1 754 307 606 233	126 564
436.cactusADM	3 692 374 503 694	3 692 376 718 040	2 214 346
437.leslie3d	2 599 825 931 830	2 599 826 865 604	933 774
444.namd	3 800 239 248 763	3 800 239 368 396	119 633
445.gobmk	688 102 677 975	688 102 680 766	2 791
447.dealII	2 069 387 009 051	2 069 306 024 811	80 984 240
450.soplex	448 971 126 111	449 160 697 736	189 571 625
453.povray	1 023 653 968 839	1 023 652 972 016	996 823
454.calculix	8 253 521 155 666	8253 934 643 401	413 487 735
456.hmmer	1 184 883 498 847	1 184 885 134 629	1 635 782
458.sjeng	2 954 760 643 001	2 954 760 645 136	2 135
459.GemsFDTD	2 106 760 402 142	2 106 788 414 210	28 012 068
462.libquantum	3 340 142 177 868	3 340 170 307 670	28 129 802
464.h264ref	382 869 988 950	382 869 983 200	5 750
465.tonto	3 759 189 446 470	3 758 683 629 473	505 816 997
470.lbm	1 678 909 037 991	1 678 909 084 138	46 147
471.omnetpp	631 607 032 909	631 402 465 700	204 567 209
473.astar	882 232 681 082	882 232 682 177	1 095
482.sphinx3	3 539 853 733 851	3 540 026 244 521	172 510 670
483.xalancbmk	1 309 798 089 981	1 306 095 154 135	3 702 935 846

1.2 DBT方法对模拟结果精度的影响

为评估DBT方法对模拟结果精度的影响，本文分别利用Simics[11]模拟器和基于DBT的BBV Profiler——QPoint[4]执行SPEC2006测试程序集，给出了基于DBT方法相对于功能模拟方法的提交指令总数差异、整体性能模拟结果和行为特征分析结果。

表1给出了所有SPEC2006测试程序分别在QPoint和Simics上执行时的提交指令总数。从表中可以看出，有约1/3(9/28)的测试程序在两次执行间的指令数相差超过10^8条指令，超过40%(12/28)的测试程序在两次执行间的指令数差异超过10^7。由于典型的模拟点大小为10^7～10^8条指令，因此该程度的指令偏差会导致模拟点定位的较大误差。

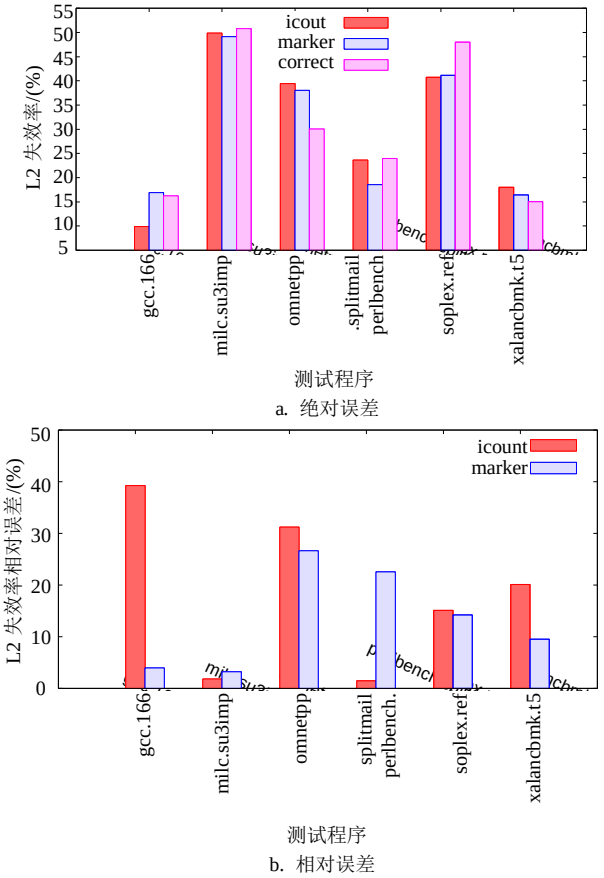

图 2　在 icount 和 marker 两种模拟点定位方法下，DBT 和功能模拟器所得到模拟点对应的 L2 失效率模拟结果

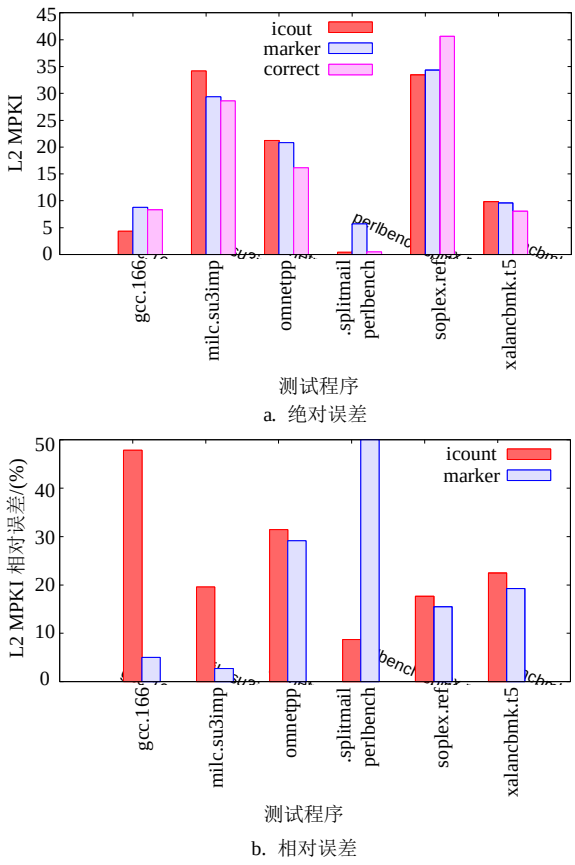

图 3　在 icount 和 marker 两种模拟点定位方法下，DBT 和功能模拟器所得到的模拟点所对应的 L2 MPKI 模拟结果

由于模拟所有测试程序所需的时间过长，本文选取了提交指令数误差大于10^8、且运行时间相对较短的400.perlbench、403.gcc、433.milc、450.soplex、471.omnetpp和483.xalancbmk共6个测试程序进行评估。用于评估的两个参数分别是L2 cache的失效率和MPKI (misses per kilo instructions)。图2给出了L2失效率模拟结果的绝对误差和相对误差，图3给出了L2 MPKI模拟结果的绝对误差和相对误差。图中，icount和marker表示利用DBT方法提取的模拟点得到的模拟结果，icount表示采用icount方法定位模拟点，marker表示采用marker方法定位模拟点，correct表示由Simics提取模拟点时所得到的模拟结果。从实验结果可以看出，不管是采用icount定位方法还是marker定位方法，由DBT方法提取的模拟点所得到的模拟结果都有较大的偏差。对L2失效率和L2 MPKI两个指标，DBT方法(QPoint)相对于传统功能模拟方法(Simics)的平均相对误差，在使用icount模拟点定位方法时分别为18%和24%，使用marker模拟点定位方法时分别为13%和2.01x。DBT方法相对于传统基于功能模拟的代表性模拟点提取方法具有较大的误差。

2　提高DBT方法的精度

导致DBT方法有较大误差的根本原因是代表性模拟点的定位偏差。不管是icount定位方法还是marker定位方法，其定位精度都取决于程序在基于DBT的BBV Profiling时和模拟时的执行踪迹的差异程度。因此提高DBT方法精度的关键是减小程序在两次执行时执行踪迹的差异。下面将针对本文提出的进程执行过程的模型，分析导致程序执行踪迹差异的可能因素，并提出相应的应对方法。

2.1　程序的执行过程模型

程序执行过程是按照程序中的指令对进程的状态进行更新的过程。程序的执行过程可表示为：

$$S_0 \xrightarrow{I_0} S_1 \xrightarrow{I_1} S_2 \xrightarrow{I_2} \cdots S_k \xrightarrow{I_k} \cdots S_n$$

式中，S_k 表示进程的状态；S_0 表示进程被创建后的初始状态；I_k 表示进程所执行的指令序列，是对进程状态的更新过程。每一条指令可以表示为一个映射 $I: S \times E \to S$，E 表示进程的执行环境，如系统时间、处理器速度、操作系统版本等。由于某些指令的行为受执行环境的影响，因此指令的语义中必须包含环境部分。对于行为不受执行环境影响的指令，

可以简单地表示为 $I:S\to S$。分支操作表现为进程状态中PC寄存器的变化。从进程执行过程的上述表示形式中可以看出，只要进程的初始状态 S_0 和执行环境 E 相同，执行踪迹就一定相同。因此，本文从 S_0 和 E 两个方面考虑减小程序在不同环境下的执行踪迹差异。

2.2 进程初始状态

进程的初始状态可以分为寄存器初始状态和内存初始状态，是由执行环境的应用程序加载器(loader)设定的。QPoint实现了一个自己的加载器，而Simics中使用的是Linux内核的加载器。本节分析了SPARC体系结构下两种加载器对进程初始状态的影响，以及带来的差异对进程执行踪迹的影响。

2.2.1 寄存器初始状态

进程的初始状态是由系统ABI规定的。根据SPARC ABI规范[12]，进程被创建后，PC寄存器指向进程的入口地址，SP寄存器指向栈顶，其他寄存器中是预定义的常量。因为可执行程序是不可重定位的，在任何执行环境中都会被加载到固定地址，因此程序的入口地址是固定的，即PC寄存器初值没有差异。

然而，不同的执行环境中，进程的SP寄存器初值并不一定相同，原因有两个：1) ABI中没有规定堆栈区域的位置，所以不同的加载器可能分配不同的存储区域作为堆栈区；2) 为了提高安全性，Linux内核会对堆栈区域的位置进行随机化处理，导致程序每次运行时，堆栈的位置都不相同。

根据以上分析，通过两项操作就可以消除SP寄存器的初值差异，从而保证进程的寄存器初始状态的一致。1) 禁用Linux内核地址空间随机化机制(通过向/proc/sys/kernel/randomize_va_space写0)，保证每次运行时堆栈起始位置都相同；2) 修改QPoint的加载器，使之与Linux内核所分配的堆栈位置相同。

2.2.2 内存初始状态

Linux系统中，一个进程的内存映像如图4所示。由于可执行程序是不可重定位的，所以.text/.data/.bss/heap段的位置和内容都是固定的。另外，使用静态链接程序时，mmap区域初始状态为空。因此内存初始状态中可能存在差异的只有堆栈区域。堆栈区域的内容包括程序执行时的命令行、环境变量、加载器所设置的辅助向量表等参数。在不同环境中执行时，通过将这些参数设置为完全相同，即可消除内存初始状态的差异。

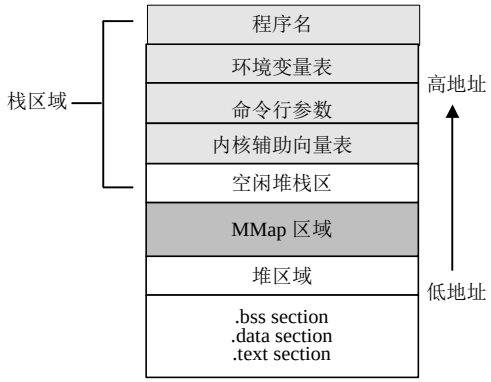

图4 Linux 系统中进程的内存映像

2.3 执行环境的影响

进程所执行的指令可以分为两类：1) 指令的语义不随执行环境变化；2) 指令的语义受执行环境影响。因此，即使进程的初始状态相同，其指令踪迹也可能由于第二类指令的语义不确定性而受到执行环境的影响。第二类指令主要包括系统调用指令、I/O指令、访问硬件计数器指令等。这些指令的语义会受执行环境的影响，从而造成不同执行环境下程序的状态出现差异，进而引起执行踪迹的差异。

执行环境的一个重要部分是操作系统内核，其行为会影响所有的系统调用请求和异常事件的处理。对于运行在QPoint上的进程，QPoint虚拟了一个操作系统内核，用于处理进程的所有系统调用请求和触发的异常事件。而对运行在Simics上的进程，其系统调用请求和异常事件是由真实的Linux内核处理的。QPoint虚拟出的内核和真实的Linux内核在虚拟地址空间分配算法、I/O缓冲块大小等方面有所不同。

地址空间分配算法不同会引起进程的内存状态出现很大差异。虚拟地址空间布局是进程的一个全局持久状态，会影响后续的整个执行过程。因此虚拟地址空间布局不同，会引起进程执行踪迹的极大差异。I/O缓冲块的大小不同也会引起内存状态出现差异。如果不同执行环境下的缓冲块大小不同，一方面会引起I/O缓冲部分执行流程的差异；另一方面，由于缓冲块是从系统堆内存中分配的，不同的块大小会导致堆内存管理系统的状态差异。堆内存状态是一个全局持久状态，因此这种差异会影响到后续的所有堆内存管理操作中，导致指令踪迹出现巨大差异。

除操作系统内核外，运行环境的资源配置、进程运行目录、系统时间，以及SPARC ABI中的寄存

器窗口溢出保存机制等也会对程序的执行踪迹产生一定的影响，不过这些环境因素中的大部分都可以在设置实验环境时轻松避免。

3 评 估

为验证第2节中提出的消除执行踪迹差异方法的有效性，本文将提出的各种方法分别应用于QPoint，然后运行SPEC2006中的所有测试程序，并与这些程序在Simics模拟器上执行时的总提交指令数进行比较，结果如表2所示。

表2　运用本文所提方法后，SPEC2006测试程序在QPoint和Simics上分别执行时的总提交指令数差异

Benchmark	Δ_0	Δ_1	Δ_2	Δ_3
400.perlbench	639 357 210	669 785 193	33 260 879	0
401.bzip2	2 347	47	14	0
403.gcc	971 956 566	841 796 416	22 293	0
410.bwaves	360 634	353 459	14	0
416.gamess	8 127	1 270	2 602	0
429.mcf	110 652	109 552	88 261	0
433.milc	131 372 027	134 305 964	19	5
434.zeusmp	7 173	314	14	0
435.gromacs	126 564	56 223	59 207	83
436.cactusADM	2 214 346	2 220 001	14	0
437.leslie3d	933 774	2 777 321	4 399	12
444.namd	119 633	117 030	117 030	0
445.gobmk	2 791	376	376	0
447.dealII	80 984 240	1 853 802	14	0
450.soplex	189 571 625	178 225 830	14	0
453.povray	996 823	171 847	1 106 492	0
454.calculix	413 487 735	413 190 921	924	0
456.hmmer	1 635 782	1 633 486	1 633 486	0
458.sjeng	2 135	14	14	0
459.GemsFDTD	28 012 068	27 877 521	14	0
462.libquantum	28 129 802	14	14	0
464.h264ref	5 750	332	332	0
465.tonto	505 816 997	821 760 286	402 477 578	345
470.lbm	46 147	43 996	43 996	0
471.omnetpp	204 567 209	17 195	17 195	9 174
473.astar	1 095	14	14	0
482.sphinx3	172 510 670	285 205	285 205	0
483.xalancbmk	3 702 935 846	1 163 328	1 163 330	0

表中，Δ_0表示未采用本文中方法的总提交指令数偏差；Δ_1表示消除进程的初始状态差异情况下的指令偏差；Δ_2表示消除进程的初始状态差异和操作系统内核影响之后的指令偏差；Δ_3表示继消除资源配置、运行目录、系统时间等环境差异之后的指令偏差。可以看出，运用本文所提方法后，对于28个测试程序中的23个，在Qpoint和Simics上执行所得到的提交指令总数完全相同，另外的5个程序只有细微的指令总数差异。相对于10^7～10^8的模拟点大小，这种量级的差异对模拟点定位带来的影响可以忽略不计。

当程序两次执行的指令踪迹完全相同时，由两次执行过程收集的BBV Profile所提取的代表性模拟点也将完全相同，并且在模拟时的模拟点定位也不会有任何偏差。因此，表2中的实验结果说明，运用本文提出的方法，程序在基于DBT加速的模拟流程能够得到与基于功能模拟的模拟流程完全相同的结果精度。

4 结 论

动态二进制翻译技术可以有效地加速SimPoint方法中BBV Profile的提取，然而该加速方法同时也会带来近20%的模拟结果误差。误差产生的根源在于应用程序在DBT环境和模拟环境下执行时具有较大的指令踪迹差异，导致无法准确定位模拟点。本文系统地分析了引起程序指令踪迹差异的可能因素，并提出了相应的方法来消除这些因素的影响。实验结果证实，利用本文提出的方法，可以在不影响DBT方法性能的同时，保证其与传统模拟方法具有完全相同的精确度。

参 考 文 献

[1] YI J J, EECKHOUT L, LILJA D J, et al. The future of simulation: A field of dreams[J]. Computer, 2006, 39(11): 22-29.

[2] PATIL H, COHN R, CHARNEY M, et al. Pinpointing representative portions of large intel itanium programs with dynamic instrumentation[C]//Proceedings of the 37th Annual IEEE/ACM International Symposium on Microarchitecture. Washington, DC, USA: IEEE Computer Society, 2004: 81-92.

[3] WEAVER V M, MCKEE S A. Using dynamic binary instrumentation to generate multi-platform simpoints: methodology and accuracy[C]//Proceedings of the 3rd International Conference on High Performance Embedded Architectures and Compilers. Berlin, Heidelberg: Springer-Verlag, 2008: 305-319.

[4] ZHAO Tian-lei, TANG Yu-xing, QI Shu-bo, et al. Qpoint: Generating simpoint basic block vector profiles efficiently with dynamic binary translation[C]//Proceedings of the 2011 International Conference on Computers, Communications, Control and Automation. Washington, DC, USA: IEEE Computer Society, 2011: 62-65.

[5] ZHAO Tian-lei, JIANG Jiang, FU Gui-tao, et al. Simics-BBV: A comprehensive tool for generating simpoint basic block vector profiles[C]//Proceedings of the 2011 International Conference on Computers, Communications, Control and Automation. Washington, DC, USA: IEEE Computer Society, 2011: 66-70.
[6] HAMERLY G, PERELMAN E, LAU J, et al. Simpoint 3.0: Faster and more flexible program phase analysis[J/OL]. [2011-08-10]. http: //www. jilp. org/volt/indet.html.
[7] YI J J, KODAKARA S V, SENDAG R, et al. Characterizing and comparing prevailing simulation techniques[C]// Proceedings of the 11th International Symposium on High-Performance Computer Architecture. Washington, DC, USA: IEEE Computer Society, 2005: 266-277.
[8] MYTKOWICZ T, DIWAN A, HAUSWIRTH M, et al. Producing wrong data without doing anything obviously wrong![C]//Proceeding of the 14th International Conference on Architectural Support for Programming Languages and Operating Systems. New York, NY, USA: ACM, 2009: 265-276.
[9] MYTKOWICZ T, DIWAN A, HAUSWIRTH M, et al. Evaluating the accuracy of java profilers[C]//Proceedings of the 2010 ACM SIGPLAN Conference on Programming Language Design and Implementation. New York, NY, USA: ACM, 2010: 187-197.
[10] ZHAO Tian-lei, JIANG Jiang, FU Gui-tao, et al. Accelerating the extraction of representative behaviors of programs with dynamic binary translation[C]//Proceedings of the 13th International Conference on High Performance Computing and Communications (to be appeared). Washington, DC, USA: IEEE Computer Society, 2011.
[11] MAGNUSSON P S, CHRISTENSSON M, ESKILSON J, et al. Simics: a. full system simulation platform[J]. Computer, 2002, 35: 50-58.
[12] Sun Microsystems. System V application binary interface, sparc processor supplement[M]. 3rd ed. California, USA: The Santa Cruz Operation, Inc., 1996.

编 辑 张 俊

(上接第914页)

[7] 谭冠军. GM(1,1)模型的背景值构造方法和应用(I)[J]. 系统工程理论与实践, 2000, 20(5): 98-103.
TAN Guan-jun. The structure method and application of background value in grey system GM(1,1) method(I)[J]. System Engineering Theory and Practice, 2000, 20(5): 98-103.
[8] 何文章, 宋国乡. 基于遗传算法估计灰色模型中的参数[J]. 系统工程学报, 2005, 20(4): 432-436.
HE Wen-zhang, SONG Guo-xiang. Estimation of grey model parameter based on genetic algorithm[J]. Journal of System Engineering. 2005, 20(4): 432-436.
[9] 罗党, 刘思峰, 党耀国. 灰色模型GM(1,1)优化[J]. 中国工程科学, 2003, 5(8): 50-53.
LUO Dang, LIU Si-feng, DANG Yao-guo. The optimization of grey model GM(1,1)[J]. Engineering Science, 2003, 5(8): 50-53.
[10] WANG Zheng-xin, DANG Yao-guo, LIU Si-feng. Optimization of background value in GM(1,1) model[J]. Systems Engineering—Theory & Practice, 2008, 28(2): 61-67.

编 辑 漆 蓉