基于代码剖析的定制指令识别

薄 拾 葛 宁 林孝康
(清华大学电子工程系 北京 100084)

摘 要 代码剖析是分析程序行为和发现性能瓶颈的重要手段。根据可重配置处理器的定制指令设计需要,提出了一种基于虚拟机技术的代码剖析方法AID-prof。该方法的优势在于结构无关以及静态\动态分析的无缝契合。基于AID-prof,设计了一种自动化的定制指令识别过程CID。实验显示,AID-prof可有效地发掘程序热点,并且示例程序通过CID产生的定制指令获得明显加速。

关键词 可重配置处理器,定制指令,识别,代码剖析,虚拟机
中图法分类号 TP314 **文献标识码** A

Custom Instructions Identification Based on Code Profiling

BO Shi GE Ning LIN Xiao-kang
(Department of Electronic Engineering, Tsinghua University, Beijing 100084, China)

Abstract Code profiling is a key technique for application behavior analysis and bottlenecks discovery. According to the requirement of designing custom instructions for reconfigurable processors, AID-prof, a novel profiling method based on virtual machine was presented. The benefits of the presented method are architecture-independent and close combination between static and dynamic analysis. Based on AID-prof, an automatic custom instructions identification procedure named CID was proposed. Experiments results show that AID-prof can discover application hot-spots effectively, and custom instructions identified by CID can markedly speed up application execution.

Keywords Reconfigurable processors, Custom instructions, Identification, Code profiling, Virtual machine

代码剖析(Code Profiling)是分析程序行为和特性的重要手段。软件程序员藉此检查设计缺陷和识别关键代码区段以提高代码质量;编译器开发者则借助该手段评估新的指令调度、分支预测等算法的性能。除软件领域以外,代码剖析驱动的优化(Profile-driven Optimization)也得到愈来愈多的应用,如计算机体系结构设计者通过代码剖析来优化结构设计,或评估新的体系结构对目标程序的执行性能。

在嵌入式计算领域,可重配置处理器(Reconfigurable Processor)通过在特定的通用指令集基础上引入定制指令(Custom Instruction,CI),在不牺牲灵活性的前提下加速特定应用的处理。研究者致力于精心设计定制指令,以最大限度地获得性能的提高。为实现该目标,必须首先获取目标应用的行为和特征。

本文对代码剖析技术进行了调查研究,针对可重配置处理器定制指令识别的需要,提出了一种体系结构无关的代码剖析方法AID-prof (Architecture-InDependent Profiling),用于程序结构提取和热点代码发现。以AID-prof为基础,构建了剖析驱动的定制指令识别过程CID (Custom Instructions IDentification)。最后,通过实例对AID-prof的代码剖析能力和CID定制指令识别效果进行测试和评价。

1 代码剖析技术

代码剖析是利用程序执行期间收集的信息来研究程序行为和特征的一种方法。收集的信息通常包括代码区段的执行次数、消耗时间、内存占用以及cache命中等。剖析过程关注的代码层次低至单个指令、基本块,高至循环结构、函数甚至是整个程序模块。代码剖析工具可以利用多种技术获取剖析信息,这些技术包括硬件中断、探针技术、动态翻译、虚拟机和指令集仿真等。

统计型剖析器通常以一定的时间间隔(一般采用系统中断)对目标程序的程序计数器进行采样,以获取动态执行信息。其优点在于分析过程基本不影响目标程序执行,缺点是精度不高且无法指定分析层次,因而现在很少使用。另一类剖析器通过向目标程序插入"探针"来收集需要的特性信息,设计者通过控制探针的插入位置和数量,获得不同层次的分析信息。例如,GNU的gprof[1]通过设置相关编译选项,由GCC编译器在应用程序编译时插入探针。gprof可以获得函数级的代码剖析信息,包括每个函数的调用次数与消耗的处

到稿日期:2009-04-09 返修日期:2009-07-01 本文受国家863高技术研究发展计划项目(2007AA01Z2b3),国家973重点基础研究发展计划前期研究专项项目(2007CB310608)资助。

薄 拾(1982—),男,博士生,主要研究领域为专用指令集设计和可重配置计算,E-mail:boshi99@mails.tsinghua.edu.cn;**葛 宁**(1971—),男,教授,博士生导师,主要研究领域为通信领域片上系统、面向通信系统高速信号处理的芯片设计;**林孝康**(1947—),男,教授,博士生导师,主要研究领域为通信网、数字集群系统、数字对讲机。

理器时间以及调用图。尽管如此，在很多情况下函数级的剖析精度过于粗粒，不能满足特定任务需要。运行时翻译型剖析器，如 Valgrind[2]，则在目标代码执行之前实时插入探针。这类剖析器本质上是一个采用 JIT(Just-In-Time)编译技术实现的虚拟机，目标代码并非直接在主机上运行，而是首先转化为中间表达(Intermediate Representation，IR)，在插入探针后再翻译为主机的机器码进行执行。

除软件工程领域以外，剖析工具也被应用于硬件设计，这类工具通常根据特定系统结构专门定制，并且需要相关辅助工具配套使用。ATOM[3]就是这类专用硬件剖析器的代表。计算机体系结构设计者往往采用指令集仿真器(Instruction-set Simulator，ISS)实现特定目标机器下的程序动态信息的获取。基于 ISS 的代码剖析能够获得细粒度的分析信息。尽管如此，其剖析过程不仅受限于已确定的体系结构，同时还不利于更高层次程序结构信息的获取。另外，德国亚琛工大的 Faruque 等人设计了一种针对微体系结构的性能分析工具 μProfiler[4]，由于 μProfiler 采用的 IR 抽象层次很低，不仅影响了工作效率，也不适用于与体系结构无关的可扩展性研究。

为服务定制指令设计，代码剖析工具需要兼备静态代码分析和动态执行信息收集两种能力。这是因为，指令设计者往往并不是应用程序的设计者，为快速准确地掌握程序主要特征，他们更希望剖析工具具有一定的静态分析能力，并能够与动态信息很好地契合。上述剖析工具在静态分析能力以及静态与动态结合方面均有所欠缺。另外，在设计探索阶段，设计者通常不希望程序剖析信息过多地受到特定体系结构的影响，因此针对特定硬件或目标机器的剖析工具也不能满足本文需要。在定制指令集设计领域，一些研究者采用 SUIF[5]/Mach-SUIF[6]配合实现代码剖析[7]，其中 SUIF 用于静态分析，Mach-SUIF 则通过 SUIFvm 实现基于 SUIF IR 的动态分析。这种方法存在的主要问题是 SUIF 的编译技术来源于轻量级编译器 lcc，代码编译质量较低，从而直接影响后续分析与优化的效果。

2 AID-prof 方法

通过上节分析可知，在定制指令识别中，理想的代码剖析工具应具备静态分析与动态信息收集两种能力，同时分析结果应保证与平台无关。为保证静态与动态分析之间的契合与反馈，剖析工具内部应当采用统一的 IR 表达；为保证分析信息的平台无关性，IR 必须具有合适的抽象层次，同时动态信息需要在一个虚拟环境下通过程序模拟运行来获取。为满足上述要求，我们提出一种新的代码剖析方法 AID-prof。AID-prof 利用 LLVM[8](Low Level Virtual Machine)的 IR 和虚拟机技术实现了平台无关的代码剖析。

LLVM 是 UIUC 开发的编译器框架，其设计目标在于实现编译、链接、运行甚至下线(如软件安装之后)各个阶段的程序优化。LLVM 编译框结构建于一种静态单值表(Static Single Assignment，SSA)形式的 IR 之上，从而保证了各个分析与优化过程的无缝结合。作为开发框架，LLVM 适合于 X86，PowerPC，ARM 等十余类体系结构，因此为保证中间过程的平台无关性，LLVM 设计了一套 RISC 风格的虚拟指令集和与之配套的 JIT 虚拟机 lli。虚拟指令和 IR 的汇编形式如图 1 所示。

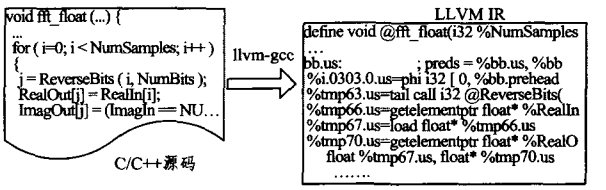

图 1 LLVM IR 的汇编形式

AID-prof 的工作流程如图 2 所示。首先，C/C++编写的应用程序源代码通过 llvm-gcc 前端转化为 IR；然后利用 analysis pass 将诸如程序控制流和数据流的结构信息从 IR 中提取出来；同时根据需要的剖析层次，测量探针被插入 IR 的特定位置；最后将 IR 送入 lli 虚拟机，在模拟执行的同时收集动态执行信息。

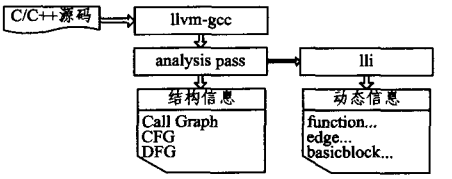

图 2 AID-prof 代码剖析流程

2.1 静态分析

作为编译器框架，LLVM 提供了一系列功能丰富的静态分析方法，统称为 analysis pass。利用 analysis pass，AID-prof 可以从 IR 中提取函数调用图(Call Graph)和控制流图(Control Flow Graph，CFG)。在 LLVM IR 中，控制流和数据流信息是分开组织的。数据流储存于基本块(basic block)中，并构成了 CFG 的节点；控制流信息则构成了 CFG 中的控制转移边，并连接相关的基本块。基本块内包含一组无跳转的顺序运算，由于采用了结构简单的 SSA 形式表达，因此可以方便地获得数据流信息，并以数据流图(Data Flow Graph，DFG)的形式进行表达。

以程序 FFT[10]为例，图 3 显示了一个典型的代码结构分析过程。作为顶层视图，调用图显示了程序中所有函数之间的调用关系；对于值得关注的函数，如图 3 中的 fft_float()，利用控制流图可进一步了解其内部基本块之间的控制和转移关系；如希望对某基本块展开分析，如图 3 中的 bb122，其数据流图将被生成以表达其内的数据流信息。

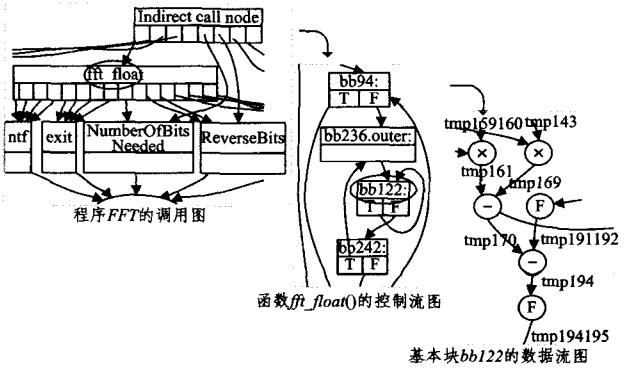

图 3 程序 FFT 的结构分析

2.2 动态分析

AID-prof 通过向 IR 内插入探针来获取动态信息，这一工作通过设计者指定剖析层次并控制 analysis pass 完成。analysis pass能够进行函数、控制转移边和基本块 3 个层次的探针

插入工作，其中基本块级为最细的粒度。含有探针的 IR 字节码在 lli 虚拟机内被实时地翻译成当前主机的二进制代码并执行，在字节码模拟运行的同时，程序的动态信息将被 IR 内部的探针记录下来。

仍以程序 FFT 为例，表 1 列出了通过上述剖析方法得到的基本块级剖析信息。通过 lli 模拟运行，执行频率最高的基本块从程序代码中发掘出来。

表 1　程序 FFT 的基本块级动态信息

基本块	执行频率	指令数	消耗周期
fft_float()-bb. i	491520	10	4915200
main()-bb115	262144	14	3670016
main()-bb111	262144	6	1572864
fft_float()-bb122	245760	48	11796480
fft_float()-bb236. outer	245760	3	737280

通过动态信息，设计者可以准确地发现频繁调度的函数、循环或是基本块。进一步，可以通过函数调用图、控制流图和数据流图来实现不同层次上的结构和行为分析。利用 AID-prof，设计者即使不具备应用程序的背景知识也能快速准确地把握程序的行为和性能瓶颈。

3　定制指令识别 CID

随着嵌入式计算领域（如数码相机、移动电话、便携式媒体播放器等）对功能集成度和处理性能要求的不断提高，可重配置处理器得到日益广泛的应用。可重配置处理器利用专门定制的扩展指令极大地弥补了嵌入式处理器相对于 ASIC（Application-Specific Integrated Circuits，专用集成电路）在数据密集型计算方面的性能缺陷，同时保持了可编程性带来的灵活与适应性优势。根据业务应用特征进行定制指令的行为级设计这一过程称为定制指令识别，在一些研究当中，这个工作主要通过设计者人工完成，然而随着业务应用的日趋复杂，设计自动化是必然的发展趋势。为此我们设计了一种自动化的设计过程 CID，其工作流程如图 4 所示。根据 AID-prof 得到的剖析结果，应用程序的热点被有效地发掘；根据热点计算结构，候选指令模板将被穷举；最后，在设计约束下，一定数量的指令模板将被选择出来作为最终的定制指令行为级模型。CID 的整个工作过程只需要设计者提交应用程序源码和设计约束，定制指令行为级描述和优化评估结果则通过相关算法自动生成。

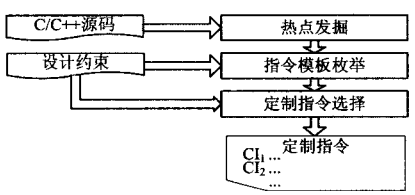

图 4　CID 的工作流程

3.1　热点发掘

根据统计，程序 90%的运行时间往往用于计算其 10%左右的代码，这一规律也被称为 90/10 定律。因此，程序代码中的一小部分热点代码将成为制约程序性能的瓶颈。代码剖析可以准确地发现上述热点代码。从抽象层次上，热点代码的聚类可以采取两种方式，即粗粒（coarse-grained）与细粒（fine-grained）聚类。粗粒聚类一般在循环或函数级上展开，因此需要识别程序中的关键循环或函数；细粒聚类则在数据流的层次上进行。根据不同的聚类层次，代码剖析也将在不同的层次上进行。尽管对于单个目标程序，以粗粒方式设计的定制指令往往能够带来更大的性能增益，但其灵活性与适应性却受到了很大限制，因此本文采用细粒聚类的方式进行定制指令的识别。

由于细粒聚类在基本块内部进行，因此利用 AID-prof 完成基本块级的代码剖析。对于程序的任意一个基本块 b，静态分析得到其内部包含的指令数 I_b；动态分析获得 b 的执行次数 f_b；于是得到基本块 b 消耗的指令周期数 C_b：

$$C_b = I_b \times f_b \tag{1}$$

根据消耗的指令周期数，对程序内的所有基本块进行排序，于是得到：

$$C_{b_1} \geqslant C_{b_2} \geqslant \cdots \geqslant C_{b_i} \geqslant \cdots \geqslant C_{b_m} \tag{2}$$

程序消耗的总指令周期数 C_{sum} 为各基本块消耗的指令周期数之和，即有：

$$C_{sum} = \sum_{i=1}^{m} C_{b_i} \tag{3}$$

为发掘关键基本块，引入影响因子 $\alpha \in (0,1)$，并构造一个简单的优化问题：

$$\min\ p$$

$$\text{s. t.} \sum_{i=1}^{p} C_{b_i} \geqslant \alpha \times C_{sum} \tag{4}$$

优化问题(4)可以通过贪婪算法快速求解。

仍以 FFT 为例，取 $\alpha = 0.9$，如图 5 所示，利用 AID-prof，对程序性能有重大影响的 7 个热点基本块被发掘。这些基本块消耗了整个程序 90%的执行周期。

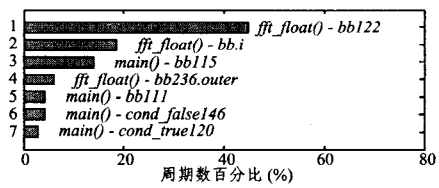

图 5　热点基本块对程序 FFT 总执行周期数的贡献

3.2　指令模板枚举

指令模板是定制指令的行为级描述形式。为实现对热点基本块的加速，一种自然的想法是从热点基本块对应的数据流图中寻找合适的子结构（即子图）作为指令模板。因此我们采用数据流图子图提取的方式来实现指令模板的枚举。根据可重配置处理器所采用的体系结构，模板子图需要满足一些特殊约束。例如，定制指令只能拥有限定数量的数据输入\输出端口，因此模板子图的输入\输出节点数量也将受到限制。

我们知道一个包含 n 个节点的数据流图其可能的子图形式共有 2^n 种，因此最坏情况下子图枚举将具有指数级的时间与空间复杂度。可以证明，在 DFG 中，确定了输入与输出节点也就能够唯一确定子图的拓扑结构。因此子图枚举可以等价转化为子图输入与输出节点的选择。假设模板的输入和输出端口约束分别为 P_I 和 P_O，那么满足约束的子图输入与输出节点组合的数量 N_{comb} 有如下结论：

$$N_{comb} \leqslant \sum_{i=1}^{P_I+P_O} \binom{n}{i} \leqslant n^{P_I+P_O+1} \tag{5}$$

由式(5)可知，将子图提取转化为输入与输出节点的选择便可以构造多项式复杂度的算法。

3.3　定制指令选择

指令模板 t 的性能提升价值可用执行时间的减少量 R_t 进行评价，在指令识别阶段内，R_t 通过估计得到。不考虑体

系结构因素，我们用周期数代替真实的时间。对模板 t，定义其描述的数据运算采用基本指令集执行所消耗的周期数为 sw_t，采用新硬件执行消耗的周期为 hw_t，假设 t 的调用频率为 f_t，则 R_t 为：

$$R_t=(sw_t-hw_t)\times f_t \tag{6}$$

尽管每个指令模板都能带来一定的性能提高，然而实际设计只能允许从中选取有限的个体作为最终的定制指令，被选择的模板组合应该极大程度地获得性能收益。由于指令模板通过子图提取的方式得到，因此来源于同一基本块数据流图的模板之间普遍存在着相互交叠的情况，为避免指令映射冲突的发生，相互交叠的模板不宜被同时选择。指令模板选择问题是一个组合优化问题，在规模较小时，这类问题可以采用分支定界、动态规划等方法求解。然而，很多情况下候选指令模板集合有着很大的规模，上述算法往往不具有可行性，因此 CID 采用贪婪思想的启发式算法实现快速求解。算法首先将模板按收益 R_t 由大到小进行排序；然后依次将与当前组合内的模板相容的新模板加入组合之中；直至组合内部的模板数量达到设计约束，选择过程结束。

假设设计约束允许的定制指令数量为 N_{CI}，最终选定的指令模板组合为 T：

$$T=\{t_1,t_2,\cdots,t_i,\cdots,t_{N_{CI}}\} \tag{7}$$

于是 T 能够节省的执行周期数 R_T 为：

$$R_T=\sum_{i=1}^{N_{CI}}R_{t_i} \tag{8}$$

假设程序总的执行周期为 C_{prog}，则可以估计通过定制指令优化后的执行周期为 $C_{opt}=C_{prog}-S_T$，于是可以计算加速比 S_p：

$$S_p=\frac{C_{prog}}{C_{opt}}=\frac{C_{prog}}{C_{prog}-R_T} \tag{9}$$

CID 采用 S_p 对定制指令带来的性能提高进行评价。

4 实验与分析

以 mpeg2dec，sha，cjpeg 和 gsm_toast 这 4 个具有代表性的应用程序为例，对 AID-prof 和 CID 进行测试。MPEG2 是目前主要的数字视频编码标准之一，其解码程序 mpeg2dec 的应用尤其广泛，因此选择该程序作为媒体处理应用的代表；安全哈希算法 sha(Secure Hash Algorithm)是一种常用的数字签名算法，根据输入信息，sha 会生成一个 160bit 的消息摘要，该摘要可用于数据完整性的验证，sha 算法也在著名的 MD4 和 MD5 算法中得到了应用，可以作为安全加密领域里的典型代表；cjpeg 是最为流行的有损图像压缩技术 JPEG 的压缩编码程序，它大量应用于数字图像处理领域以及数码相机等便携式设备中；gsm_toast 则是 GSM 移动通信标准中的语音压缩算法，它将语音信号压缩为 13kbit/s 的码流，我们选择 gsm_toast 作为通信与语音压缩算法的代表。mepg2dec 与 cjpeg 的源代码来源于媒体计算基准集 MediaBench[9]，sha 和 gsm_ toast 的源代码则来源于嵌入式计算基准集 Mibench[10]。

利用 AID-prof，上述应用程序得到了细致分析。表 2 列举出了各应用程序的函数数量、基本块数量以及总代码量。在这里设置 $\alpha=0.9$，则消耗程序总体 90%执行周期的热点基本块被发掘出来，其数量也列于表 2 之中。通过比较可以发现程序热点代码所占比率很小，基本符合 90/10 定律。更直观地，提取出来的热点基本块对各程序执行周期数的贡献通过图 6 显示。

表 2 目标程序基本信息

目标程序	函数	基本块		代码		
		全部	热点	全部	热点	比率%
mpeg2dec	53	1460	12	9285	145	1.6
sha	8	46	6	803	154	19.2
cjpeg	134	2994	34	23114	595	2.6
gsm_toast	62	1030	26	5920	399	6.7

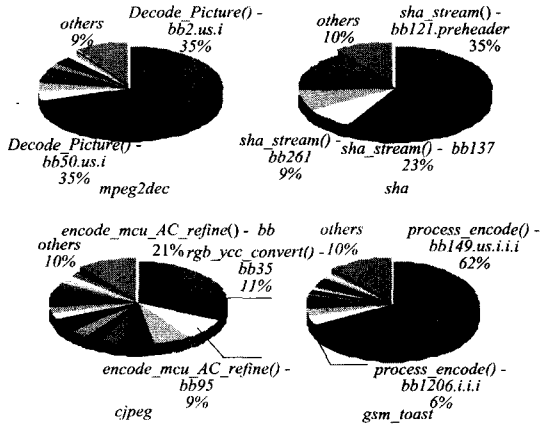

图 6 热点基本块对程序整体执行周期的贡献度

从图 6 可以看出，排在最前列的热点基本块消耗了程序主要的执行时间。相对而言，sha 和 gsm_toast 的热点更为集中，mpeg2dec 次之，cjpeg 的热点分布则最为分散。接下来，为优化热点基本块的执行，从其数据流图中枚举定制指令模板。在实验中，令结构约束分别为 $P_I=6$，$P_O=3$，算法完全地枚举所有输入节点数不超过 6 且输出节点数不超过 3 的模板子图。各程序指令模板枚举情况列于表 3 之中。

表 3 定制指令模板枚举结果

目标程序	mpeg2dec	sha	cjpeg	gsm_toast
模板数量	308	396	2832	1636

表 3 的数据显示，模板枚举过程将产生大量的候选指令模板，然而可增加的定制指令数量是受设计约束严格限制的。利用扩展指令选择过程，限定数量的指令模板得到选择。图 7 反映了定制指令的数量对程序性能提高的影响。很明显，引入的定制指令越多，程序可以获得的加速比越大。然而值得注意的是，当定制指令数目 $N_{CI}<10$ 时，加速比随 N_{CI} 的增大而迅速提高；然而当定制指令数目 $N_{CI}>10$ 时，这个增速将逐渐变缓。因此，实际设计中应在硬件成本与性能提升之间折中考虑，选择合适的定制指令数目以获得理想的费效比。以 $N_{CI}=10$ 为例，4 个目标程序分别实现了 1.41～2.06 倍的加速，性能优化效果显著。比较发现，热点分布相对集中的应用程序(如 gsm_toast)比热点分布分散的程序(如 cjpeg)具有更大的优化潜力。

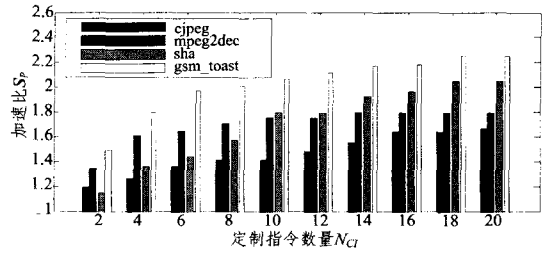

图 7 定制指令数量与加速比的关系

结束语 代码剖析作为一种重要的分析手段被广泛地应用于软件优化、编译器以及计算机体系结构设计等领域。可

重配置处理器定制指令设计是一个典型的剖析驱动的优化过程，根据设计需要，本文提出了一种新的代码剖析方法 AID-prof，实现了剖析结果的平台无关性和静态分析与动态分析的无缝契合，并在该方法的基础上构建了自动化的定制指令识别过程 CID。实验显示，利用代码剖析，CID 识别产生的定制指令可使应用程序获得可观的性能提升。

参 考 文 献

[1] GNU. Profiling a Program: Where Does It Spend Its Time ? [EB/OL]. http://sourceware.org/binutils/docs-2.16/gprof/

[2] Seward J, Nethercote N, Weidendorfer J. Valgrind 3.3-Advanced Debugging and Profiling for GNU/Linux applications [M]. Network Theory Ltd

[3] Srivastava A, Eustace A. ATOM: A System for Building Customized Program Analysis Tools [C]// Proc. of ACM SIGPLAN Conference on Programming Language Design and Implementation. Orlando, Florida, 1994: 196-205

[4] Karuri K, Faruque M A A, et al. Fine-grained application source code profiling for asip design [C]// Proc. of the 42nd Annual Conference on Design Automation. Anaheim, California, USA, 2005: 329-334

[5] Wilson R P, French R S, Wilson C S, et al. SUIF: An Infrastructure for Research on Parallelizing and Optimizing Compilers [J]. SIGPLAN Notices (S0362-1340), 1994, 29(12): 31-37

[6] Smith M D. Machine-SUIF: A Research Infrastructure for Profile-driven and Machine-specific Optimizations [EB/OL]. http://www.eecs.harvard.edu/hube/software/software.html

[7] Atasu K, Pozzi L, Ienne P. Exact and Approximate Algorithms for the Extension of Embedded Processor Instruction Sets [J]. IEEE Transactions on Computer-aided Design of Integrated Circuits and Systems (S0278-0070), 2006, 25(7): 1209-1229

[8] Lattner C, Adve V. LLVM: A Compilation Framework for Lifelong Program Analysis & Transformation [C]// Proc. of the 2004 International Symposium on Code Generation and Optimization. Palo Alto, California, USA, 2004: 75-86

[9] Lee C, Potkonjak M, Mangione-Smith W H. Mediabench: A Tool for Evaluating and Synthesizing Multimedia and Communications Systems [C]// Proc. Int. Symp. Microarch., 1997: 330-337

[10] Guthaus M R, Ringenberg J S, Ernst D, et al. MiBench: A Free, Commercially Representative Embedded Benchmark Suite [C]// Proc. IEEE 4th Ann. Workshop Workload Characterization. 2001: 3-14

(上接第 140 页)

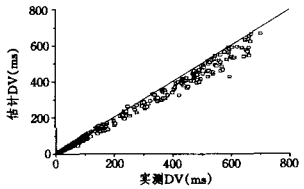

图 5 路径长度为 7 的延迟变化 QQ 图

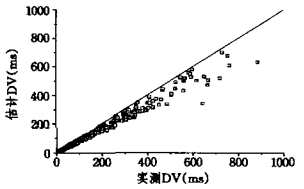

图 6 路径长度为 7～10 的延迟变化 QQ 图

结束语 本文提出了用分段性能参数来估计端到端性能参数的方法，重点解决了由分段延迟变化来准确估计端到端延迟变化的问题。通过在 PlanetLab 上部署生成器和测量机证明了本方法的有效性和准确性。本研究为 Internet 上的服务质量保证提供了一种全新的手段。例如：将 Internet 划分为若干分段，每个分段都维护一系列实时测量参数，以很少的字节描述包流通过本自治域的性能参数。任意两个节点通信时，首先发送信令询问端到端路径上每个分段的当前性能参数，当信令到达通信目的节点时即可计算当前链路所能提供的服务质量，此时接收端可以按照一定的策略选择开始或终止通信。对带宽的讨论超出了本文的讨论范围，调整包长和发送速率可以进一步研究带宽对延迟变化和其他性能参数的影响，这是下一步的研究重点。

最后，感谢普林斯顿大学的 Stephen Soltesz、浙江大学的 PlanetLab 工作组，以及其他对本文的实验工作提供宝贵资源和提出宝贵意见的同行。

参 考 文 献

[1] Internet protocol data communication service-IP packet transfer and availability performance parameters [DB/OL]. ITU-T Recommendation Y. 1540. 2005

[2] Network Performance Objectives for IP-based Services [DB/OL]. ITU-T Recommendation Y. 1541. 2006

[3] Demichelis C, Chimento P. RFC3393: IP Packet Delay Variation Metric for IP Performance Metrics [EB/OL]. RFC Editor United States, 2002

[4] Kapoor R, Chen L J, Lao L, et al. Capprobe: A simple and accurate capacity estimation technique [J]. ACM SIGCOMM Computer Communication Review, 2004, 34(4): 67-78

[5] 刘世栋，张顺颐，邱恭安，等. 一种基于端到端测量的路径性能参数估计算法[J]. 电子与信息学报，2007，29(7)：1617-1621

[6] Ismail M N, Zin A M. Comparing the Accuracy of End-to-End Network Performance Measurement Testbed and Simulation Model for Data Transfers in Heterogeneous Environment [C]// Proceedings of the Second International Conference on Modeling & Simulation. Singapore, 2008: 124-131

[7] Armolavicius R. Simple Approximations of Delay Distributions and Their Application to Network Modeling [J]. Lecture Notes in Computer Science, 2007, 4516(1): 507-518

[8] Ramsey C B. A Note on the Normal Power Approximation [J]. ASTIN Bulletin, 1991, 2(1): 147-150

[9] PlanetLab [EB/OL]. http://www.Planet-Lab.org, [4. 10 2008]

[10] Park K S, Pai V S. CoMon: a mostly-scalable monitoring system for PlanetLab [J]. ACM SIGOPS Operating Systems Review, 2006, 40(1): 65-74