第33卷 第13期 计 算 机 工 程 2007年7月
Vol.33 No.13 Computer Engineering July 2007
·软件技术与数据库· 文章编号：1000—3428(2007)13—0078—03 文献标识码：A 中图分类号：TP311.5

龙芯/ORC编译器中的Edge Profiling技术

梁珊珊，张军超，冯晓兵
(中国科学院计算技术研究所，北京 100080)

摘 要：在程序实际执行中，Profiling 技术能为编译器提供准确的轮廓信息。编译优化借助这种轮廓信息，可在优化时进行取舍，提高生成代码性能。该文介绍了在龙芯/ORC 编译器中 edge profiling 的技术，给出了在 edge profiling 辅助下 CPU2000 性能测试结果。
关键词：轮廓信息；edge profiling；龙芯/ORC 编译器

Edge Profiling Technology in Godson/ORC Compiler

LIANG Shanshan, ZHANG Junchao, FENG Xiaobing
(Institute of Computing Technology, CAS, Beijing 100080)

【Abstract】A compiler can get accurate execution profile information about the program through profiling. It takes advantage of this information to aid trade-off making in various optimizations and improves the performance of the final code. This paper introduces the edge profiling technology in Godson/ORC compiler, gives the performance test report of CPU2000 with edge profiling enabled.
【Key words】profile information; edge profiling; godson/ORC compiler

在计算机科学中有一条著名的经验，即在程序执行过程中，90%以上的执行时间消耗在了比重小于 10%的代码上。这是由程序的局部性原理引起的，即程序在执行过程中的一个较短时期，所执行的指令地址和指令的操作数地址，分别局限于一定区域。比如程序中存在相当多的循环和递归的结构，它们由少量指令组成，却被多次执行。

编译器在做优化时可以利用这种偏向性信息，优先满足经常执行代码的要求(包括执行部件、寄存器要求等)，通过提高热点程序的性能而获得总体性能的提高[1]。早期，编译器获取这种轮廓信息的方法是通过对程序作静态分析，估计得出。比如，在C程序中，1 个if/then/else结构中执行then语句的概率为 90%，执行else语句的概率为 10%。显然这种静态分析方法有很大的局限性和不准确性。编译器若基于这种不准确甚至错误的信息做出优化，优化效果会受到影响。

现代优化编译器的一般做法是采用 Profiling(轮廓)技术(Java 的动态优化技术不在本文的讨论范围内)，即：

(1)在第 1 遍编译时，编译器在应用程序的“中间表示”上插入代码收集所需信息。这些插入的代码被称为插桩代码，此过程被称为插桩(instrumentation)。这样，最终编译所生成的可执行代码中就夹杂了收集信息用的插桩代码。以“典型输入”运行该程序，顺带执行插桩代码，收集所需轮廓信息并以文件形式保存(称为反馈文件)。

(2)在第 2 遍编译时，编译器读取刚刚生成的反馈文件，将收集到的信息标记在“中间表示”上，在相关优化时加以利用。此过程被称为标记(annotation)。

1 Profiling 分类及龙芯/ORC 编译器简介

根据收集信息的不同，Profiling 可分为：

(1)Edge profiling：收集程序控制流图上各个基本块的执行频率及各个控制边的执行概率。图 1 给出了一个标记有 edge profiling 信息的控制流图的例子。其中方框表示基本块，方框旁边的数字表示此基本块的执行频率。边表示控制转移，边上的数字表示控制沿此条边走的概率。编译器后端的多种优化都要用到这类信息。比如在指令调度中，计算关键路径时需要知道控制边的执行概率。假设在图 1 的基本块 D 和 E 中都有一条指令能调度到 C，这时应当优先选择 E 中的指令，因为 E 中的指令执行概率更大，所以本文应尽快地执行它，减少指令延迟。

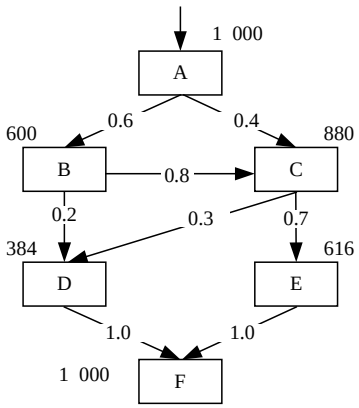

图 1 标记有 edge profiling 信息的控制流

(2)Path profiling：收集程序控制流图上路径的执行频率。直接从 edge profiling 结果推导有些路径的频率是不准确的。仍以图 1 为例，容易知道路径 ABD 的频率为 1 000*0.6*0.2，但路径 ABCD 的频率并不一定等于 1 000*0.6*0.8*0.3，这是因为从 C 到 D，控制流既可以从 BC 过来，也可以从 AC 过

基金项目：国家“973”计划基金资助项目(2005CB321602)
作者简介：梁珊珊(1981－)，女，硕士研究生，主研方向：Profiling，机器模型，软件测试；张军超，助理研究员；冯晓兵，研究员、博士生导师
收稿日期：2006-09-02 **E-mail：**suansuan412@hotmail.com

来。当控制流从 BC 来时，C 到 D 的概率未必是 0.3。Path profiling 可用在编译优化和软件测试中。

(3)Value profiling：收集程序变量的取值信息。利用value profiling的优化包括常数传播、代码特例化等[2]。

(4)Stride profiling：收集程序变量取值的步长信息，即相临两次取值的差值信息。编译器利用这类信息可以实现对不规则数据访问的数据预取[3]。

龙芯/ORC 编译器全面实现了上述各类 profiling 技术。龙芯(Godson)是由中科院计算所设计的一款高性能通用微处理器，其中龙芯 2 号采用超标量超流水技术，与 MIPS R4K 指令集兼容。ORC(open research compiler)原是由中科院计算所与 Intel 公司合作开发一个面向 IA-64/Linux 平台的开放源码编译器。本文把 ORC 移植到了龙芯/Linux 平台，称为龙芯/ORC 编译器。下面将重点介绍 edge profiling 这类应用极为广泛的一类 profiling 技术在龙芯/ORC 编译器中的实现。

2 龙芯/ORC 中的 Edge Profiling 技术

Edge profiling 在龙芯/ORC 编译器的后端(即代码生成部分)实现。它被安排在指令选择之后，此时后端刚刚建立起程序或函数的控制流图。程序、函数在 ORC 中统称为 program unit(PU)。在具体实现中，只需在后端加入对 edge profiling 接口函数的调用。下面介绍如何解决 edge profiling 的几个关键问题，即在什么地方插桩、如何插桩、如何组织反馈信息，和在第 2 遍编译时，如何标记 edge profiling 信息。

2.1 插桩位置

前面提到过 Edge profiling 收集的是控制流图中基本块的执行频率及控制边的执行概率两类信息。其实本文可以通过收集控制边的执行频率来推导到这两类信息。显然，以下 2 个式子成立：

$$\text{基本块BB的频率} = \sum_{e_i \in \text{BB的入边}} e_i\text{的频率} = \sum_{e_j \in \text{BB的出边}} e_j\text{的频率} \quad (1)$$

$$\text{控制边e的概率} = \text{e的频率} \Big/ \text{e的前驱基本块的频率} \quad (2)$$

本文只需在控制边上插入辅助基本块、植入插桩代码、收集执行频率。如图 2(a)所示，在它的每条边上插入基本块，得到如图 2(b)所示的流图，图 2 中椭圆表示新插入的基本块。

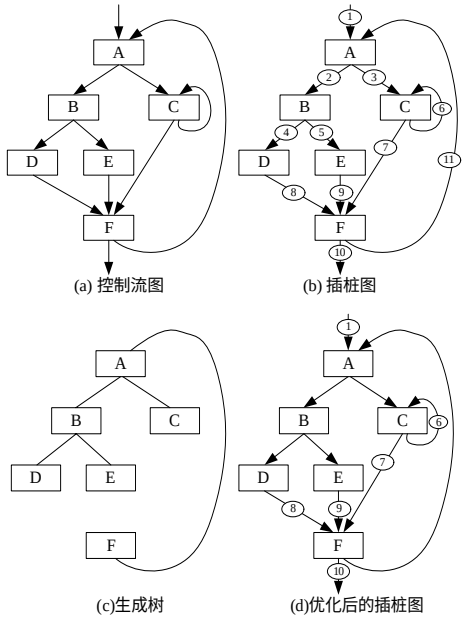

图 2　选择 edge profiling 的插桩位置

由图 2(b)可知，有些插桩是冗余的，它们可以由其它基本块的执行频率推导出来。比如基本块 8 的执行频率必然等于基本块 4 的执行频率。为降低插桩开销，应当有选择性的选择插桩边。一种方法是选择“非生成树边”。对无向图 G(V, E)，若有一个树 T(VT, ET)满足 VT=T，ET⊆T，则称 T 为 G 的生成树。对控制流图，忽略掉边上箭头，把它看成无向图，构造它的生成树。那么插桩时只用在流图中那些不在生成树上的边上插桩即可。生成树可通过对无向图做广度优先搜索或深度优先搜索构造得出。图 2(c)显示了对图 2(a)中以 A 为根节点广度优先搜索而得的生成树，图 2(d)则是化简后的插桩图。

更进一步，还可选择最大生成树。对图 2 中每条边赋一权值，在所有可能的生成树中选择其中边的权值之和最大的一个。此处可令每条边的权值等于静态估计所得到的执行频率。本文只需在较少执行的边上进行插桩，降低profiling开销。求最大生成树，可采用Kruskal算法[4]，其基本思想是将边按权排序，然后在不造成回路的情况下，取权最大的边。

2.2 插桩代码

选择好插桩位置后，就可以在流图中加入插桩基本块了。插桩基本块本质上是一个调用 edge profiling 统计函数的基本块。Edge profiling 统计函数的输入参数有两个：正在编译的函数名(即 PU 名，是个字符串)；正在插桩的控制边的 ID。按照 MIPS N32 应用程序二进制接口(application binary interface, ABI)的规定，第 1 个定点输入参数通过定点寄存器 $4 传入，第 2 个通过$5 传入。先要生成指令把函数名指针和控制边 ID 分别拷贝到$4、$5，然后再生成调用 edge profiling 统计函数的 call 指令(在龙芯中为 JAL 指令)。

插入插桩代码时还应考虑当前控制边前驱基本块的类型。本文把含有 call 指令的基本块称为 CALL_BB，反之称为 NON_CALL_BB。如果控制边前驱基本块为 CALL_BB，则有额外的工作要做。按照 MIPS N32 ABI 的规定，无论定点寄存器还是浮点寄存器都被分为 2 类：由调用者保存(caller-save)；由被调用者保存(callee-save)。对一个 callee-save 的寄存器来说，它的生存区间可穿过 call 指令。相反，对一个 caller-save 的寄存器来说，它不能穿过 call 指令。要想在 call 指令前后保证值不变，必须在 call 指令之前先保存起来，在 call 指令之后再恢复。MIPS N32 ABI 规定，用作函数返回值的寄存器都是 caller-save 的寄存器。这包括传定点返回值的$2、$3 寄存器，传浮点返回值的 f0、f1、f2、f3 寄存器。若控制边前驱基本块是一个 CALL_BB 且它有返回值，那么为了使这些返回值穿过新插入的调用 edge profiling 统计函数的 call 指令，就必须查询它使用了哪些返回值寄存器，并在插桩基本块的开头和末尾分别插入对它们作保存和恢复的指令。edge profiling 插桩算法如下：

For each插桩边e_k

　生成一个新基本块bb，并插在e_k上；

　在 bb 中插入传参指令(即 mov)及 call 指令，调用 edge profiling 统计函数；

　If (e_k的前驱基本块是CALL_BB且有返回值)

　　For each 返回值寄存器 reg

　　在 bb 头尾分别插入对 reg 的 save、restore 指令

Edge profiling 统计函数要做的工作很简单，它根据传入的 PU 名、控制边 ID，找到这个 PU 的 edge profiling 信息的

存储位置，并将指定控制边的执行频率加 1。Edge profiling 统计函数在 ORC 中是一个库函数，程序执行时将调用它。因为这个函数功能很简单，所以为了降低 profiling 开销，还可以进一步对它进行内联(inlining)。另外，除了在选定的控制边上插入基本块调用 edge profiling 统计函数，还需要在每个 PU 的入口连续插入两个基本块，依次调用整个 profiling 的初始化函数和本 PU profiling 的初始化函数，这用来打开反馈文件、申请内存，初始化变量等。

需注意的是，一旦插入了新基本块，需要同步更新控制流图，比如更新新加入的基本块及受它影响的基本块的前驱和后继信息。

2.3 反馈文件

Edge profiling 信息以文件方式保存，称为反馈文件。为了便于将各种 profiling 信息存储在同一个反馈文件中，ORC 的反馈文件参考了 Linux ELF 文件格式，采用了图 3 所示的结构。

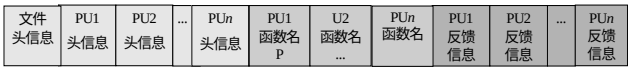

图 3 反馈文件格式

主要的反馈文件格式如下：

(1)文件头信息：包含的内容有：

1)一个幻数，标记这是一个反馈文件；

2)PU 头信息的偏移地址、PU 的个数、PU 头信息的大小；

3)PU 函数名表的偏移地址、PU 函数名表的大小；

4)反馈信息的偏移地址。

(2)PU 头信息：包含的内容有：

1)插桩的基本块个数；

2)该 PU 反馈信息的偏移地址；

3)该 PU 函数名在 PU 函数名表中的索引值。

(3)PU 函数名表：这是一个字符数组，连续存放 PU1~PUn 的函数名，比如“ main ”。

(4)PU 反馈信息：按预定的顺序存放该 PU 的 Profiling 信息，也就是非最大生成树边及其对应的执行次数。一般以数组形式存放，以提高访问速度。

可见，定义的反馈文件格式有易扩展、易操作的特点。

2.4 标记

有了反馈信息文件，在第 2 遍编译时，就可按它的格式进行读取。给出 PU 名和控制边 ID，查找执行频率，并在 PU 的中间表示上进行标记。最初，只能得到非最大生成树边的执行频率。这些信息虽然不完全，但是完备的。根据式(1)、式(2)，很容易推导出所有基本块的执行频率以及所有边的执行概率。

3 实验结果及结论

本文在龙芯/ORC 编译器中实现了上述的 edge profiling 技术并作了测试。测试平台为：405 MHz 龙芯 2 号处理器，1 级数据/指令 Cache 各 32KB，4MB 片外 2 级 cache, 256M 主存，Debian Linux 2.4.22。测试用例采用 CPU2000 基准测试集，输入采用 ref 输入集。编译时除了打开/关闭 edge profiling 外，其它优化选项全部打开。

图 4、图 5 分别给出了 CPU2000 INT 和 FP 在打开 edge profiling 情况下、相对未打开时的性能加速比。加速比的计算公式为：1+(打开 edge profiling 时的 SPEC 分数–关闭 profiling 时的 SPEC 分数)/关闭 profiling 时的 SPEC 分数。

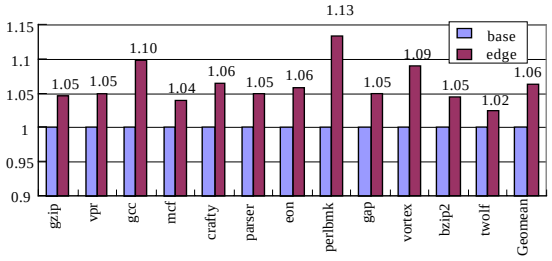

图 4 SPEC2000 INT基准测试加速比

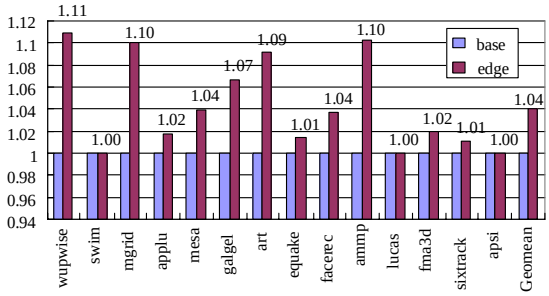

图 5 SPEC2000 FP 基准测试加速比

由图 4、图 5 可以看出 edge profiling 效果很明显。CPU2000 INT 性能提升最高可达 11.3%(perlbmk)，平均达到 6%。CPU2000 FP 性能提升最高可达 10%(ammp)，平均达到 4%。由于借助了 edge profiling 技术，能获得一些热点基本块的准确的分支概率(通常它与启发式算法估算出来的值有较大的出入)，因此，性能得到了大幅度提升，其它编译优化技术的效果也得到突出，最终程序性能得到了提升。

致谢：感谢龙芯编译组的支持，特别是刘章林、黄磊、贺国锦、王磊、吕方、吴家骏、刘弢等同学的帮助。

参考文献

1 Mahlke S. Using Profile Information to Assist Classic Code Optimizations[J]. Software Practice and Experience, 1991, 21(12): 1301-1321.

2 Muth R, Watterson S, Debray S K. Code Specialization Based on Value Profiling[C]//Proc. of the 7th International Static Analysis Symposium. 2000.

3 Wu Youfeng. Efficient Discovery of Regular Stride Patterns in Irregular Programs and Its Use in Compiler Prefetching[C]// Proceedings of Programming Language Design and Implementation. 2002.

4 Kruskal J B. On the Shortest Spanning Tree of a Graph and the Traveling Salesman Problem[C]//Proceedings of the American Mathematical Society. 1956: 48-50.