文章编号:1001 - 9081(2005)10 - 2431 - 03

基于硬件性能计数器的软件性能数据采集与分析研究

程克非[1,2], 张 聪[3], 汪林林[2],张 勤[1]
(1. 重庆邮电学院 计算机科学与技术系,重庆 400065;2. 重庆大学 计算机科学与技术学院,重庆 400030;
3. 重庆交通学院 计算机科学与技术系,重庆 400074)
(chengkf@cqupt.edu.cn)

摘 要:引入了基于 CPU 硬件性能计数器的性能数据采集和分析方法,从软件运行时刻的细粒度参数入手分析软件运行时刻的性能表现,从而更为准确地反映系统实际的动态运行状态。实验证明,这种方法对于需要详细掌握系统动态运行状态的应用能够提供非常有效的分析数据,同时也在一定程度上对编译器的性能优化给出了相关参考数据。

关键词:数据采集; 硬件性能计数器; 性能分析
中图分类号: TP302.7 **文献标识码**:A

Software performance data sampling and analysis based on hardware performance counter

CHENG Ke-fei[1,2], ZHANG Cong[3], WANG Lin-lin[2], ZHANG Qin[1]
(1. *Department of Computer Science and Technology, Chongqing University of Posts and Telecommunications, Chongqing* 400065, *China;*
2. *Institute of Computer Science and Technology, Chongqing University, Chongqing* 400030, *China;*
3. *Department of Computer Science and Technology, Chongqing Jiaotong University, Chongqing* 400074, *China*)

Abstract: This paper introduced a much more precise sampling method using CPU hardware performance counters (CHPC) to provide CPU data like instruction cycles, cache misses, branch prediction, and so on, and given detail scene of software status. With CHPC data, more accuracy software performance analysis and compiler optimization can be reached.

Key words: data sample; hardware performance counter; performance analysis

0 引言

高性能计算的优化研究,一直以来在很大程度上依赖于硬件厂商提供的工具和 bench mark 数据。同时硬件厂商也希望应用系统能够提供足够的优化数据,供其在以后的设计中改进体系结构设计。

现代的 CPU,一般都至少具有 2 级高速缓存,加上高速主存缓存、主存、磁盘交换等不同运行速度的存储,构成内存的层次结构。磁盘缓存与 CPU 缓存的访问速度一般都相差几个数量级[1,2,3]。在应用系统一级上对于性能优化的研究,主要是研究代码如何高效率的利用内存的层次结构,使得慢速的访问尽可能减少。由此,构成了性能分析与优化的主要研究内容:更好的性能评测工具(benchmark)、更好的性能数据采集工具(monitoring)、分析模型(modeling)和自动或半自动代码优化工具[4,5]。

本文的研究主要集中在性能数据采集上,讨论如何有效收集对于评测和分析软件系统代码性能有用的性能数据,包括各类硬件指令的运行频度、内存映射情况、执行代价等数据并支撑有关性能分析实验。性能数据采集的研究结果,可以广泛应用在高性能计算、编译器优化、操作系统和硬件驱动程序改进等底层和核心的技术中。

1 PAPI 性能数据采集

所有性能分析与优化问题的前提是:如何在尽可能不影响软件系统正常运行的条件下,获取尽可能多的和准确的性能数据。多年以来,获取软件的硬件性能数据一直是一个实验性的课题,研究者通过定时器等粗粒度的方法来衡量代码段的执行时间,并估计其运行的状态,以作为系统调整的依据。这些数据是不精确的,不能提供例如内存访问映射、缓存命中率等对于性能分析有实际意义的数据,因此,早期对于软件系统的性能分析是困难的和粗略的。

1.1 CPU 硬件性能计数器

现代的主流 CPU,大都在其内部设计了一组专门用于读取 CPU 运行时刻的操作特性的寄存器,称之为 CPU 硬件性能计数器(Hardware Performance Counter, CHPC),表 1 为常见的一些计数器。上述 CPU 包括 Intel Pentium, IA - 64, AMD Athlon,以及 IBM POWER 系列等等。这些硬件计数器提供了简单而充分的性能分析数据提取接口,可以用来对代码的运行情况进行非常精确的分析。CHPC 对性能计数值的采集是由通过与正常运行指令并行的内部流水线完成,因此可以在很少的执行代价下采集数据(采集自操作系统核心调试数据)。

收稿日期:2005 - 04 - 13;**修订日期**:2005 - 06 - 27

作者简介:程克非(1974 -),男,重庆人,讲师,博士研究生,主要研究方向:软件性能分析与故障诊断; 张聪(1970 -),男,重庆人,讲师,主要研究方向:人工智能、博弈论与故障诊断; 汪林林(1947 -),男,重庆人,教授,主要研究方向:数据库理论、数据挖掘与地理信息系统; 张勤(1956 -),男,重庆人,教授,博士生导师,主要研究方面:人工智能、故障诊断.

表 1 现代 CPU 中可能存在的 CHPC 计数器

计数器	说明	计数器	说明
Cycles	时钟周期	Branches Miss-Predicts	未命中的预测分支数
Load Instructions	有效 Load 指令数	TLB Misses	传输缓冲的未命中数
Store Instructions	有效 Store 指令数	Total Cache Misses	内部 Cache 的未命中数
L1 Misses	L1 Cache 失败数	Float Instructions	浮点数指令数
L2 Misses	L2 Cache 失败数	Request Shared Cache Line	申请独占共享 Cache 线次数

在不同的 CPU 中,以上这些参数可能以不同的命名出现,并且可能有一些细小的区别。本文实验环境中对 CHPC 存取的平均时钟代价如下:

PERFCTR INIT: vendor 0, family 6, model 11, stepping 1, clock 730772 kHz

PERFCTR INIT: rdtsc cost is 30.9 cycles (2214 total)

PERFCTR INIT: rdpmc cost is 33.2 cycles (2362 total)

PERFCTR INIT: rdmsr (counter) cost is 90.4 cycles (6022 total)

PERFCTR INIT: rdmsr (evntsel) cost is 71.7 cycles (4829 total)

PERFCTR INIT: wrmsr (counter) cost is 100.7 cycles (6681 total)

PERFCTR INIT: wrmsr (evntsel) cost is 96.1 cycles (6386 total)

PERFCTR INIT: read cr4 cost is 1.8 cycles (355 total)

PERFCTR INIT: write cr4 cost is 42.2 cycles (2939 total)

PERFCTR INIT: sync_core cost is 147.4 cycles (9671 total)

perfctr: driver 2.6.12 DEBUG, cpu type Intel P6 at 730772 kHz

从上面我们看出对于一个 CHPC 事件的读(rdmsr)周期代价为 90.4,写(wrmsr)CHPC 的代价为 100.7,而 CHPC 读写(rdmsr,wrmsr)事件设定的代价分别 71.7 和 96.1;读系统的时钟周期(rdtsc)代价为 30.9,读 CHPC 计数器(rdpmc)的指令本身执行代价为 33.2,读写系统特权控制寄存器(read cr4 write cr4)的代价分别为 1.8 和 42.2。系统核心层的同步代价为 147.4。这些数据都表明,对于 CHPC 的数据采集的时钟代价相对很低。

1.2 PAPI

CPU 提供了 CHPC 硬件性能计数器,但对 CHPC 的访问必须具有操作系统底层存取权限。由美国航天航空局(NASA)和能源部支持的 PAPI 项目[2,6],针对各种不同的 CPU,封装了对不同 CPU 硬件计数器的低层访问函数集,构成了 PAPI(Performance Application Programming Interface),使得进行性能分析的研究人员可以不用关心底层存取权限、平台和编程语言的差别而获取 CHPC 性能分析数据。

对于需要采集 CHPC 的应用软件系统,可以在程序开始调用 PAPI_library_init 初始化 PAPI 接口,然后创建监控事件集 EventSet,在 EventSet 中加入具体需要监控的事件(CHPC 性能计数器),之后在合适的位置插入 PAPI_read 函数读取当前监控的 CHPC 事件值。在第四节中将看到,通过并发线程同步采集方式,在本文实验环境下,PAPI 对 CHPC 采集所造成的软件系统执行效率的降低基本在 0.2% 以下。

2 数据分类与性能分析模型

2.1 数据分类

朴素贝叶斯分类器(Naive Bayesian)[7~9]是最有效的分类模型之一。朴素贝叶斯分类器从训练数据集合中学习给定类标签 C 条件下属性 $A_i, i \in [1,n]$ 发生的条件概率。经过学习的分类器即可将 Bayes 规则用于输入的实例 $a_i, i \in [1,n]$ 并计算类别标签 C 的不同取值 C_i 发生的后验概率,并预测后验概率最大的 C_i 为当前实例的类别。

在给定类别条件下所有属性相互条件独立的假设是朴素贝叶斯分类算法的基本假设,也是这一算法称为"朴素贝叶斯"算法的缘由。条件独立假设是一个很强的假设,并且在大多数实际问题中均无法满足。但是很多研究表明[7~9],朴素贝叶斯分类的性能相当好,目前尚未发现在每一个数据集上性能均好于朴素贝叶斯分类的分类方法,即使是一些看来设计得更加严密的算法,例如决策树,其分类正确性在测试的数据集上显著地比朴素贝叶斯分类差。本文的各个事件属性可以认为相互条件独立,因此选用朴素贝叶斯分类方法对采集到的性能数据进行分类。

2.2 性能分析模型

根据系统特征,我们设计了基于 CHPC 的软件性能分析模型 PHPC,如图 1 所示。

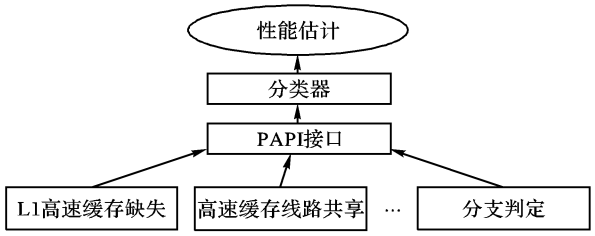


图 1 基于 HPC 的性能分析模型 PHPC

PAPI 负责采集性能数据,送入信息分类器中分类学习和处理,最后提交性能评估模块评价当前软件的运行性能。

3 数值实验

作为对上述数据采集技术研究和分析模型的验证,这里以一个具有大量浮点数运算的求圆周率的程序为例,如图 3 所示,其运行环境为 Linux-2.6.9,PIII730M 编译环境为 GCC-3.4,通过对其计算过程中的各种有效 CHPC 性能计数器的监控和数据采集,利用朴素贝叶斯分类方法,根据分类结果,评价程序的运行性能。求解圆周率的算法如下:

```
double f( double a)
{
 return (4.0 / (1.0 + a*a));
}
...
h = 1/n;
for (i=1;i<=n;i++)
{
 x = h * (i - 0.5);
```

```
    sum += f(x);                    (A)
    sum += 4.0/(1.0+x*x);           (B)
  }
  pi = h * sum;
```

对于(A)是采用函数调用方式,而(B)则直接计算。下面从数据采样对程序执行效率的影响,以及通过分析采样数据得出程序当前性能,并简单给出优化建议两个方面来展开研究。

3.1　数据采样效率

在实验中,我们对程序在不同条件下编译运行,给出结果,见表2。

表2　不同条件下的程序运算状态

序号	函数调用	编译优化	PAPI采集	计算结果精度	实际执行周期
1	是	否	否	5E-16	1592209586
2	是	否	是	5E-16	1595597883
3	是	是	是	5E-16	803509393
4	是	是	否	5E-16	795197314
5	否	是	否	5E-16	797878875
6	否	是	是	5E-16	803480447
7	否	否	否	5E-16	745711667
8	否	否	是	5E-16	745870218

其中函数调用栏“是”表示采用(A)进行求和计算,“否”表示采用(B)进行求和;编译优化指是否让编译器在编译过程中产生优化代码;PAPI采集表示是否在程序中插入对性能数据的采集;计算结果精度为程序运算结果与程序初始设置的运算结果的比对,以说明各种情况下是否对程序的运算结果有影响;实际执行周期指程序在不同情况下,运算出预定结果时的耗时,以CPU的时钟周期为单位。

分析表中的数据,可以得到,对于各种情况下,插入PAPI采集后,整体的计算时间性能降低分别为:

p1 = 0.2128047%;(1,2)

p2 = 1.0452851%;(3,4)

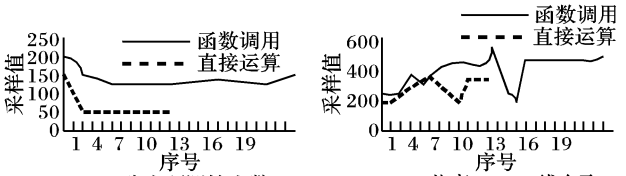
p3 = 0.7021%;(5,6)

p4 = 0.0213%;(7,8)

最高影响约为1.04%(采用了编译优化,并存在函数调用),而最低影响仅为0.02%(不使用编译优化,并直接嵌入求和代码)。

3.2　数据分析与优化

程序中的优化建议根据当前运行的程序的CHPC事件值的大小和变化情况,运用数据分类器完成分类,并由最终的性能评估模块给出优化建议。在程序的优化建议中,将用到编译调试信息反向定位性能瓶颈在源代码中的位置。如果配合gcc调试工具gprof[10],定位效果将更好。



(a) 分支预测缺失数　　(b) 共享 Cache 线存取

图2　采样数据对比分析

图2是根据性能评估模块给出的建议,将程序中的函数调用改为直接插入代码运行后,与未改动前两种不同调用方式下运行得到的采样数据对比。可以看出,改进后分支预测缺失数大大减少,同时对共享Cache线的存取数也有相应的减少,从而最终导致程序的运行性能提高。从表2和图2中都可以得出改进后的执行时间大约是改进前的1/2。

在表2的数据中,对于采用函数调用,编译优化的情况,与不采用函数调用,直接手工优化相比,各自的运算时间分别为795197314和745711667(不插入采集),以及803509393和745870218(插入采集),可以得出这样一个事实:GCC-3.4的C语言编译器的优化能力还有改进的空间,因为实验中仅仅简单地应用了一个代码嵌入的手工优化技巧,就比使用GCC优化的效率高出了近7个百分点,这样的差异对于具有高性能计算特征的软件系统来说,是非常值得改进的。

4　结语

本文主要从性能数据的采集和性能分析模型的建立上,对传统的性能分析方法进行了改进。通过实验证明,采集到的软件动态运行性能数据很好地反映了软件实际运行情况,给出了更为精确的性能数据进行性能分析;同时,经过分析,也从定性上给出了编译器的优化效率。在软件性能数据分类中,采用了现在在分类算法中相对简单并且非常有效的Naive Bayesian方法,取得了较好的分类效果,为进一步的性能评估提供了较为准确的分类数据。在今后的研究中,将通过机器学习和动态编译[11]等技术,分析影响不同类型软件系统的特征CHPC性能计数器,进一步改善数据采集效果,从而提供更好的软件性能优化决策信息。

参考文献:

[1] Performance Evaluation Research Center. Progress Report: PERC Collaborations with SciDAC Scientific Projects [EB/OL]. http://perc.nersc.gov/reports, 2005.

[2] BROWNE S, DONGARRA J, LONDON GK, *et al*. A Scalable Cross-Platform Infrastructure for Application Performance Tuning Using Hardware Counters [A]. Proceedings of the IEEE/ACM SC2000 Conference[C], 2000.

[3] BHATTACHARYA S, DEY S, BRGLEZ S. Performance Analysis and Optimization of Schedules for Conditional and Loop-Intensive Specifications [A]. Proceeding of Design Automation Conference [C], New York: ACM Press, 1994. 491-496.

[4] Analytic Phase Models [EB/OL]. http://perc.nersc.gov/models/models_phase.htm, 2005.

[5] Performance Evaluation Research Center. Progress Report: Performance Modeling Activities in PERC2 [EB/OL]. http://perc.nersc.gov/reports.

[6] ICL of University of Tennessee. PAPI Programmer's Reference [EB/OL]. http://icl.cs.utk.edu/papi.

[7] GOLDSZMIDT FG. Bayesian network classifiers [J]. Machine Learning, 1997, 29: 131-163.

[8] LANGLEY P, IBA W, THOMPSON K. An analysis of Bayesian classifiers [A]. Proceedings of the National conference on Artificial Intelligence (AAAI'92) [C]. Menlo Park, CA: AAAI Press, 1992. 223-228.

[9] DUDA RO, HART PE. Pattern Classification and Scene Analysis [M]. New York: John Wiley & Sons. 1973.

[10] GNU gprof [EB/OL]. http://www.cs.utah.edu/dept/old/texinfo/as/gprof.html, 2005.

[11] TAM D, WU J. Using Hardware Counters to Improve Dynamic Compilation [R]. ECE1724 Project Final Report, 2003.