一个基于硬件计数器的程序性能测试与分析工具*)

车永刚[1] 王正华[1] 李晓梅[2]
(国防科大并行与分布处理国家重点实验室 长沙410073)[1]
(怀柔装备技术指挥学院 北京101416)[2]

摘 要 在Intel P6系列处理器与Microsoft Windows NT平台上开发了一个工具软件PTracker,它利用处理器中的硬件性能计数器来获取程序性能数据,并结合机器体系结构参数对数据进行分析。它无需用户编程,与应用程序所使用的编程语言无关,使用很方便。它不仅能够通过性能计数器获得精确的性能参数,而且还能通过对测试得到的性能数据的分析,揭示程序高层次的性能特征,对程序性能评价与优化具有一定的指导作用。本文介绍了PTracker的技术背景、设计与系统实现,并给出了一个应用实例。

关键词 硬件计数器,程序性能测试,数据分析,性能优化

A Hardware Counter Based Tool for Application's Performance Measurement and Analysis

CHE Yong-Gang[1] WANG Zheng-Hua[1] LI Xiao-Mei[2]
(National Lab. for P & D Processing, NUDT, Changsha 410073)[1]
(Institute of Equipment and Command Technology, Beijing 101416)[2]

Abstract This article introduces PTracker, a useful tool designed and implemented for Intel P6 processors running Microsoft Windows NT operating system. It utilizes hardware performance monitoring counters in modern processors to measure an application's performance data. Furthermore, it performs additional calculation to the measured data based on the target platform's architecture parameters and profiles. PTracker is convenient to use in that it requires no modification to the applications' source codes and is independent of the application's programming language. It not only captures the application's signatures, but also provides more informative performance characteristics of the application, which are helpful for the understanding and tuning of the application. A case of its application in program optimization is presented.

Keywords Hardware performance monitoring counters, Performance measurement, Data analysis, Performance tuning

1 引言

性能测试是理解程序行为的基础,对识别程序性能瓶颈,了解软/硬件资源利用情况和程序对机器性能的发挥程度具有重要作用。程序测试与分析的结果,可用于指导程序性能优化、算法效率评价和性能模型开发。

目前已经出现了很多性能测试或分析方法。计时方法测定程序中指定部分或整个程序的执行时间,对性能对比及定位程序核心模块(即占用时间最长的部分)很有用,但它不能给出性能问题出在何处。静态分析方法(如cache miss equation[1])从源程序出发,结合目标计算机的体系结构及参数,分析预测程序性能,在简单情况下能够获得较准确的结果,而且速度快,适于编译时使用。但是对复杂应用程序与系统,静态分析的准确性难以保证。模拟方法以可执行程序、经过某种编译变换得到的程序或者程序执行产生的踪迹(trace)数据为输入,在一个模拟器(如simplescalar[2])上执行之,通过该模拟器收集性能数据,并进行分析。该方法允许改变程序或机器参数,使用灵活,并可将性能问题与程序代码相关联。但它的执行速度很慢,难以在编译时使用,性能数据的准确性也难以保证。

现代计算平台越来越复杂,如处理器中广泛使用了深流水线、多级存储层次、多功能部件、硬件预取、控制/数据speculation等计数。与此同时,应用程序也日益复杂化,如指针的使用、基于各种各样的库、面向对象等。程序性能测试与分析更加困难,需要更高级的工具[3]。为方便性能测试,很多处理器中出现了专门的硬件装置,称为硬件性能监视计数器(hardware performance monitoring counter,以下简称硬件计数器),通过它们能够获得程序与机器(特别是处理器)相互作用的细节信息,极大地增强了性能数据的质量与可靠性,而且速度快,系统开销小[4,5]。基于硬件计数器的程序性能测试与分析是当前研究的热点之一,已经成为一些程序性能优化方法如动态优化(Dynamic Optimization)的基础。目前已经涌现出了很多相应的使用硬件计数器的系统与工具,如PAPI[6]、PCL[7]、DCPI[8]、Vtune[9]等。

我们基于WinPAPI(PAPI的Windows版本),在Intel P6系列处理器与Windows平台上开发了一个基于硬件计数器的程序性能测试与分析工具PTracker(Performance Tracker)。它不仅使用户能够方便地利用硬件计数器获取应

*)本文得到国家自然科学基金重点项目(69933030)资助。车永刚 博士研究生,主要研究方向为计算机体系结构、程序性能评测与优化。王正华 教授,博士生导师,主要研究方向为计算机体系结构与并行计算、生物信息学。李晓梅 教授,博士生导师,主要研究方向为计算机体系结构与并行计算、科学计算可视化。

用程序的性能数据，而且还结合机器体系结构参数对所获得的数据进行深入的分析，向用户提供更具启发性的数据。本文介绍了 PTracker 的技术背景、设计思想及系统实现，并以矩阵相乘作为例说明如何利用它来测试程序性能并指导程序优化。

2. 技术背景

2.1 硬件计数器

硬件计数器是处理器中一组特殊的寄存器，这些计数器或者计数事件，或者测量事件持续的时间。当计数事件时，计数器的值在特定事件发生或者特定数目的事件发生时候增加。当测量持续时间时，计数器统计在某个特定条件为真期间所经历的处理器时钟周期数。大多数现代微处理器上都提供了硬件计数器，如 MIPS R10000包含4个32位硬件计数器，IBM Power 3处理器包含8个32位硬件计数器，Intel 处理器家族中自 Pentium 以后都提供硬件计数器。

硬件计数器监视与处理器性能相关的事件分为基本事件（如时钟周期、引退指令）、指令执行（如指令译码、流出、执行，存储操作）、分支事件（如分支预测）、存储层次（如 Cache 访问情况）等。

2.2 Intel P6系列处理器的性能监视[10]

P6系列处理器提供2个40位的硬件计数器 PerfCtr0和 PerfCtr1，用来进行实际的性能事件计数。还包含两个性能事件选择寄存器 PerfEvtSel0与 PerfEvtSel1，各对应一个硬件计数器，控制对性能监视计数器的操作。运行于任意优先级的进程都可以使用 RDPMC 指令来读取硬件计数器的值，而只有运行于优先级0的进程可以使用 RDMSR 指令来读取性能事件选择寄存器，使用 WRMSR 指令来写两类寄存器。

为了使用 P6系列处理器的性能监视计数器，操作系统必须提供一个性能监视设备驱动程序，以处理下列操作：

·处理器特征检查：确定当前处理器是否支持硬件计数器，这通过将 CPUID 指令返回的处理器族、型号等来判断。

·初始化与启动硬件计数器：在 PerfEvtSel0和/或 PerfEvtSel1中写入合法的配置，并设置 PerfEvtSel0中的使能计数器标志来启动硬件计数器。

·停止硬件计数器：清除 PerfEvtSel0中的使能计数器标志，或清除 PerfEvtSel0与 PerfEvtSel1的全部位就可以停止硬件计数器。

·读取 PerfCtr0与 PerfCtr1中的性能计数值。

2.3 PAPI

PAPI（Performance Application Programming Interface）是美国田纳西大学创新计算实验室开发的一组与机器无关的例程，提供对硬件计数器的访问，其研究目的是设计、标准化与实现可移植、高效的硬件计数器 API。PAPI 为用户使用硬件计数器提供3种接口：

·低级接口：管理用户定义的事件组中的事件，为工具开发人员和高级用户提供方便。

·高级接口：提供启动、停止和读取特定事件的能力。

·图形界面（Perfometer）：PAPI 性能数据可视化工具。

PAPI 在 x86处理器＋Windows NT/2000/XP 平台的版本称为 WinPAPI，其中使用 WinPMC 内核设备驱动程序控制从用户应用程序中访问性能监视计数器。

PAPI 主要以 API 方式供用户使用（目前包括 C 接口与 Fortran 接口），用户必须在应用程序中包含 PAPI 头文件、库并调用 PAPI 函数，使用很不方便，且对每个应用程序都要重复同样的工作。况且，除非使用与 C 或 FORTRAN 的混合编程，否则使用 PASCAL 或 Java 等语言编写的程序无法使用 PAPI。对无法获得源代码的应用程序，PAPI 无能为力。

PAPI 提供了一个性能可视化工具 Perfometer，它不需要用户程序调用 PAPI 函数，但需要调用 Perfometer 库函数，此外，它使用 Java Applet 来可视化数据，要求用户机器安装 JDK。

从 PAPI 调用获得的数据多数是简单的性能指标，用户需要结合体系结构方面的知识对这些数据进行分析与计算，才能更好地理解程序性能特征，识别潜在的性能问题。

PTracker 的研究主要就是为了解决上述问题而开展的。

3. PTracker

3.1 设计思想

PAPI 的接口与功能可以封装在一个工具软件中，该软件控制性能事件的选择，计数器的启动、停止与硬件计数器值的读取。利用该软件监控目标应用程序的执行，对由该程序引发的性能事件进行计数，这样就可以以无编程的方式使用 PAPI 进而通过硬件计数器获取程序性能数据。

PAPI 提供的主要是程序完成其执行所需要进行的各种操作的计数，如果结合一定的机器模型与参数，对测试得到的“Raw”数据进行分析，就可以向用户提供更高层次的数据，帮助用户更深入地了解程序性能特征，对进一步的程序性能优化提供指导。机器的体系结构可以从目标平台的手册获得，部分性能参数可以通过一些低级的 Benchmark 如 MAPS[11]来获得。

针对不同类型的应用程序以及不同的目的，用户可能需要不同的测试模式来方便地获得所需要的性能数据，因此在该工具软件中应该提供满足多种需要的测试模式。

提供图形化的用户界面，使用户不仅能够选择需要测试的程序与需要计数的事件，还可以进行程序参数（如命令行参数、数据类型等）、机器参数及测试模式的设置。

3.2 系统组成

基于 WinPAPI 的 C 语言编程接口，使用 Microsoft Visual C++编程实现了 PTracker。组成如图1。

3.3 多种测试模式

PTracker 中提供了多种测试模式，包括：

单程序模式：在一次会话中逐个测试单个程序的多个指定性能参数。

批处理模式：在一次会话中逐个测试指定的多个程序的多个指定性能参数。

启动/停止模式：用户选择某个事件，手动控制对该事件计数的启动/停止，测试在启动/停止之间该事件发生的次数。

固定时间段模式：测试定长时间段内的1个或多个性能参数。这对于测试一些持续时间长且均匀执行的程序对系统的性能压力有用。

3.4 数据分析

通过 PAPI 接口调用得到的数据（在下面的计算中它们一般以 PAPI 开头），经过对它们进行分析处理，就可以提取性能特征，分析影响其性能的主要因素。PTracker 除了各级 Cache 命中率的计算外，还主要提供了对程序的访存/计算平衡、程序的访存影响率、流水线影响率与性能发挥比率[12]等的分析。本节涉及到的 PAPI 参数的具体含义参见附录 A 或

文[13]。

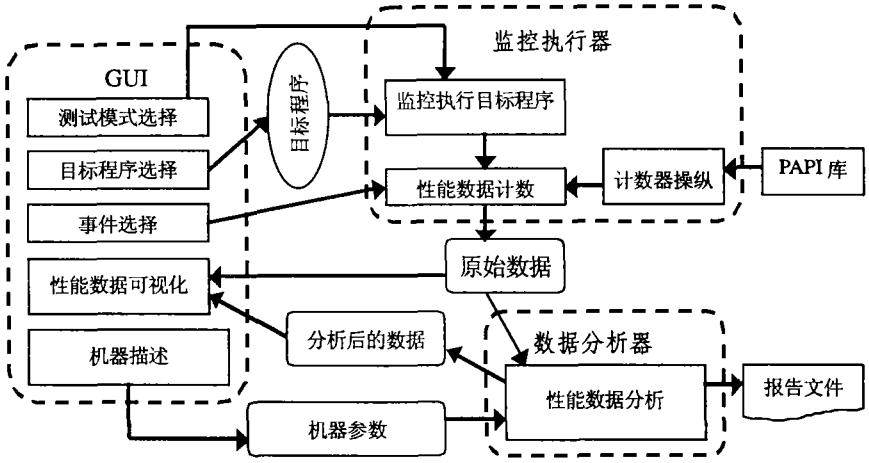

图1 PTracker 的组成

3.4.1 Cache 命中率 Cache 命中率的计算比较简单。L1数据 Cache 命中率：

1.0-PAPI_L1_DCM/PAPI_L1_DCA

L1指令 Cache 命中率：

1.0-PAPI_L1_ICM/PAPI_L1_ICA

L2数据 Cache 命中率：

1.0-PAPI_L2_Tcm/PAPI.L2_TCA

如果某级 Cache 的命中率在95%以上，则说明程序对该 Cache 的访问有较好的局部性。

3.4.2 程序平衡 现代计算机存储带宽与 CPU 处理能力不匹配的问题日益突出，经常成为应用的性能瓶颈[14]。文[15]中提出了浮点程序平衡(程序带宽需求与浮点计算能力需求之比)、机器平衡(机器带宽大小与浮点计算能力之比)的概念。如果程序平衡大于机器平衡，则程序执行期间，存储带宽满足不了 CPU 计算所需的数据传输速率，程序性能受限于存储带宽。

程序平衡与机器平衡都是针对特定存储层次而言的。对P6处理器，存在 CPU 到 L1数据 Cache、L1数据 Cache 到 L2 Cache、L2 Cache 到主存三个层次(分别记为 H1、H2、H3)，相应地具有三个层次上的平衡。

定义3.1 机器执行每条浮点指令期间平均能够从存储层次 $H_i(i=1,2,3)$传输的存储字节数，称为机器在层次 H_i 的平衡 BM_i。

机器平衡可以通过机器的浮点峰值速度、各级存储层次的数据总线宽度与总线频率来计算。

定义3.2 程序执行期间平均每条浮点指令需要从存储层次 $H_i(i=1,2,3)$传输的字节数，称为程序在层次 H_i 的平衡 BP_i。

根据测试时从 PAPI 接口获得的参数，可以计算程序在各个存储层次的平衡(8、32、32分别是 Pentium Ⅲ 处理器的 L1数据 Cache 总线宽度、L1数据 Cache 行长和 L2 Cache 行长)：

$BP_1=(PAPI\_L1\_DCA\times 8)/PAPI\_FP\_INS$

$BP_2=(PAPI\_L1\_DCM\times 32)/PAPI\_FP\_INS$

$BP_3=(PAPI\_L2\_TCM\times 32)/PAPI\_FP\_INS$

3.4.3 性能影响因子 对现代计算机来说，程序对存储层次及指令流水线的有效利用对性能影响极大，特别是存储层次利用情况对程序性能发挥具有决定性的影响[16]。文[12]中使用访存影响率(记为 η_m)来表征 Cache 失效对性能的影响，流水线影响率(记为 η_{pl})表征指令级流水线并行度对性能的影响，性能发挥比率(记为 η_{fp})表征程序实际性能发挥程度，并给出了其计算公式：

η_m=1－程序实际浮点速度/程序理想浮点速度

η_{pl}=1－程序理想浮点速度/处理机峰值浮点速度

η_{fp}=程序实际浮点速度/处理机峰值浮点速度

关系式：

$\eta_{fp}=(1-\eta_m)\times(1-\eta_{pl})$

其中程序理想浮点速度定义为：程序在没有 Cache 不命中情况下的浮点执行速度。

从 PAPI 参数出发，根据 P6系列处理器的两级存储层次，使用简单的 Cache 不命中开销模型，可以计算出程序的访存影响率与流水线影响率。

令 C_1：L1 Cache 不命中但在 L2 Cache 命中的访存延迟(周期)；C_2：L1 Cache、L2 Cache 都不命中的访存延迟；F：处理器主频(Hz)；G：机器的峰值浮点性能(FLOPS)；T：程序执行的时间(秒，使用进程时间)；T_m：Cache 不命中时间开销；T_1、T_2为临时变量。C_1、C_2、F、G 都由用户在开始程序测试前设定。则

$T_1=(PAPI\_L1\_TCM-PAPI\_L2\_TCM)\times C_1$

$T_2=PAPI\_L2\_TCM\times C_2$

$T_m=(T_1+T_2)/F$

$\eta_{pl}=1-PAPI\_FP\_INS/(G\times(T-T_m))$

$\eta_{fp}=PAPI\_EP\_INS/(G\times T)$

若上述计算得到的访存影响率大于50%，说明程序访存局部性差，需要进行 Cache 优化；流水线影响率大于50%，说明程序对指令级并行性开发不好，需要进行指令级并行性优化。

4 应用实例：矩阵相乘

矩阵相乘是科学与工程计算领域广泛使用的数学内核，我们以矩阵相乘代码的性能测试与优化为例来说明 PTracker 的应用。源程序使用 Fortran 语言编制，数据类型为浮点单精度。测试所用的配置如表1。

表1 矩阵相乘测试的软硬件配置

CPU	550 MHz Intel Pentium Ⅲ，峰值性能550 MFLOPS
L1 Cache	16K/16K，4路组相联，行长32B，带宽8.8GB/s
L2 Cache	256K，8路组相联，行长32B，带宽8.8GB/s
主存	PC100SDRAM，带宽 0.8GB/s
操作系统	Windows 2000 Professional
编译器	Compaq Visual Fortran 6.5
优化选项	-O4 (Full Optimizations)
机器平衡	$BM_1=16$，$BM_2=16$，$BM_3=1.45$
不命中开销	$C_1=7$，$C_2=29$

原始矩阵相乘程序的代码(矩阵 a、b 初始化的代码省略)如图2 (a)，从表2的测试数据可以看出，它的存储影响率和流水线影响率很高(都超过75%)，性能只达处理器峰值性能的4.9%，需要进行 Cache 与 ILP 优化。此外，程序平衡 BP2与 BP3也超过机器平衡 BM_2和 BM_3，也说明需要对程序进行 Cache 优化。代码(a)中 c(i,j)的访问没有按照存储顺序(Fortran 按列顺序存储)访问，导致访问步长很大，可以将循环 i 交换到最内层，以增加对数组 a 和 c 访问的空间局部性。得到代码(b)。

```
do i=1,N                do j=1,N                do j0=1,N,jb            do j0=1,N,jb
do j=1,N                do k=1,N                do k0=1,N,kb            do k0=1,N,kb
do k=1,N                do i=1,N                do j=j0,min(j0+jb-1,N)  do j=j0,min(j0+jb-1,N)
c(i,j)=c(i,j)+a(i,k)*b(k,j)  c(i,j)=c(i,j)+a(i,k)*b(k,j)  do k=k0,min(k0+kb-1,N)  do k=k0,min(k0+kb-1,N)
...                     ...                     do i=1,N                do i=1,N,8
                                                c(i,j)=c(i,j)+a(i,k)*b(k,j)  c(i,j)=c(i,j)+a(i,k)*b(k,j)
                                                ...                     (...)
                                                                        c(i+7,j)=c(i+7,j)+a(i+7,k)*b(k,j)
                                                                        ...
        (a)                     (b)                     (c)                     (d)
```

(a)原始代码；(b)循环 i 被交换到最内层后的代码；(c)再对循环 j 和 k 进行分块后的代码；(d)最内层循环展开 8 次后的代码。

图2 矩阵相乘程序优化

从表2第2列的测试数据看到，循环交换显著减少了程序的 L1数据 Cache 和 L2 Cache 的不命中率，程序的访存影响率和流水线影响率一定程度地降低了，但是都仍然高于50%。程序平衡 BP_3也仍然超过机器平衡 BM_3。L1 数据 Cache 的不命中率也仍然高于5%。比较 PAPI_L2_TCM 与 PAPI_L2_TCA 发现，代码(b)的 L2 Cache 访问不命中率非常高，说明 L2 Cache 中的数据没有得到好的重用。考虑对循环 j、k 进行分块，以增加数据访问的时间局部性，使一个分块的数据能够容纳在 Cache 中而多次重用。得到代码(c)。

表2 PTracker 获得的性能数据

性能参数	(a)	(b)	(c)	(d)
PAPI_L1_DCA (M)	507.2	440.8	475.9	418.8
PAPI_L1_DCM (M)	153.57	33.88	8.86	8.86
PAPI_L2_TCA (M)	153.57	33.91	8.89	8.89
PAPI_L2_TCM (M)	150.70	33.25	0.68	0.70
PAPI_BR_CN (M)	34.24	34.24	34.50	5.14
PAPI_BR_MSP (M)	0.279	0.278	0.281	0.280
PAPI_FP_INS (M)	267.0	267.0	267.0	267.0
L1 Miss Rate	0.303	0.077	0.019	0.021
L2 Miss Rate	0.981	0.981	0.076	0.079
BP_1	7.543	6.555	7.078	6.229
BP_2	18.271	4.035	1.058	1.058
BP_3	17.929	3.956	0.081	0.083
T	9.948	3.111	1.057	0.813
T_m	7.982	1.761	0.140	0.141
MFLOPS	27.0	86.5	254.3	330.7
η_{lm}	0.802	0.566	0.132	0.173
η_{lpl}	0.751	0.638	0.467	0.273
η_{lfp}	0.049	0.157	0.462	0.601

表2第3列测试数据显示分块后程序的 L1数据 Cache 与 L2 Cache 不命中率大大降低，程序的访存影响率与流水线影响率下降到13.2%，程序三个存储层次上的程序平衡都小于机器平衡。程序整体性能达到处理器峰值性能的46.2%。但是程序的流水线影响率仍然偏高(46.7%)。考虑对最内层循环进行循环展开，以向处理器提供更多的不相关指令来开发 ILP。得到代码(d)。

表2第4列数据显示，循环展开使程序所执行的分支指令(PAPI_BR_CN)从34.50 million 减少到5.14 million，程序利用的 ILP 大大增加，程序的流水线影响率降低到了27.3%，虽然程序的存储影响率增加了一些(其实是因为总时间缩短，访存不命中开销所占的时间比例相对增加)，但程序整体性能提高，达到了处理器峰值性能的60.1%。

注意上述程序代码中，N=512。我们使用实验的方法确定最优的分块大小(jb=50，kb=2)和展开因子(8)。表2所列出的数据只是 Ptracker 能够测试与分析数据的一部分。

从这个例子可以看出，通过 Ptracker 的测试与分析，能够精确地定位程序性能问题所在，对程序性能优化非常有用。

结论与展望 本文在 P6处理器＋Windows NT 上开发了程序性能测试与分析工具 PTracker，该工具有下列优点：

(1)基于硬件计数器，获得的性能数据丰富、可靠。

(2)直接针对可执行程序进行测试，与编程语言无关，使用方便，应用范围广。

(3)结合体系结构参数的性能数据分析能够准确定位程序存在的主要性能问题，对程序优化具有指导作用。

(4)整个系统只有400kB 大，存储与执行开销小，对被测程序的干扰很小。

目前 PTracker 的主要缺点是缺少自动 Instrumentation (在源程序或者可执行代码中插入监控代码)功能，这限制了它只能针对整个程序进行测试。下一步计划增加静态二进制 Instrumentation 模块，使它能够针对不同粒度的程序单元(如函数或基本块)进行测试。

PTracker 进行的数据分析还比较简单，如 Cache 开销模型中没有考虑 Cache 预取、非阻塞 Cache 访问等因素的影响。将来应该根据不同处理器的特点采用更精细的开销模型。

另外，将底层设备驱动程序替换成能够访问其他处理器(如 Intel Pentium 4等)的硬件计数器的驱动程序，并对接口处作相应的修改，可以使 PTracker 能够适用于这些处理器。

附录 A 本文用到的 PAPI 参数的含义

PAPI_L1_DCA	L1数据 Cache 访问数
PAPI_L1_DCM	L1数据 Cache 不命中数
PAPI_L2_TCA	L2 Cache 总的访问数
PAPI_L2_TCM	L2 Cache 总的不命中数
PAPI_BR_CN	所执行的条件分支指令数
PAPI_BR_MSP	条件分支指令预测错误数
PAPI_FP_INS	所执行的浮点指令数

参 考 文 献

1 Ghosh S, et al. Cache Miss Equations: A Compiler Framework for Analyzing and Tuning Memory Behavior. In ACM Transactions on Programming Languages and Systems, 1999, 21(4): 702～745
2 http://www.cs.wisc.edu/～mscalar/simplescalar.html
3 Merten M C, et al. An Architectural Framework for Run-Time Optimization. IEEE Transactions on Computers, 2001, 50(6): 567～589
4 Lambert, et al. Profiling I/O Interrupts in Modern Architectures. In: 8th Intl. Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems, San Francisco, California, 2000
5 Hirzel M, et al. Bursty Tracing: A Framework for Low-Overhead Temporal Profiling. In: 4th Workshop on Feedback-Directed and Dynamic Optimization (FDDO), Dec. 2001
6 http://icl.cs.utk.edu/projects/papi/
7 http://www.gz-juelich.de/zam/PCL/
8 http://research.compaq.com/SRC/dcpi/
9 http://developer.intel.com/vtune/
10 IA-32 Intel® Architecture Software Developer's Manual: Volume 3: System Programming Guide. Intel Corporation. 2002
11 http://www.sdsc.edu/PMaC/Benchmark/MAPS/
12 Zeyao M, Zhang Baolin. Multilevel averaging weight method for load imbalance problems. International J. Comp. Math, 2000, 77 (3/4)
13 Browne S, et al. A Portable Programming Interface for Performance Evaluation on Modern Processors. The International Journal of High Performance Computing Applications, 2000, 14 (3): 189～204
14 Ding C, Kennedy K. Memory bandwidth bottleneck and its amelioration by a compiler. In: Proc. of the 2000 Intl. Parallel and Distributed Processing Symposium, Cancun, Mexico, May, 2000
15 Carr S. Combining Optimization for Cache and Instruction-Level Parallelism. In: Proc. of PACT '96, Boston, USA, Oct. 1996
16 Bailey D H. Performance Metrics: Out of the Dark Ages. In: High-Speed Computing Conf. Gleneden, OR, Apr. 2001

(上接第83页)

用来进行参照用的，因此在实际应用中关于外键的选择操作是基本上不会发生的。对 MM-DBUG 方法，如果要执行外键上的选择操作，一种可能的执行方案是：直接在外键所在关系中进行选择操作，即对于每一个元组，首先通过外键中存储的 TID 值去间接访问外键的真正取值，再与 C 进行比较；为了提高执行性能(如果有必要的话)，还可以关于外键域专门建立一个索引。

3.2.2 *连接操作的时间代价分析* 与选择操作的分析类似，三种方法的时间代价是同一数量级的。对于 FF 方法，文[3]对几种索引形式下的连接操作算法的性能进行了分析，结论是：①如果两个连接的关系 R_1 和 R_2(设 R_2 为内连接关系)在连接属性上都有索引，则 Tree Merge 方法有最好的性能，此时如果 T Tree 索引也存在的话，则通过扫描该索引还可以进一步减少连接过程中的比较次数，比较的次数大致为 $(|R_1|+2\cdot|R_2|)$。如果连接属性取重复值的话，则比较的次数会增加。②如果两个连接的关系上只有一个索引(设内连接关系上有索引)，则 Hash Join 方法有最好的性能，在一个 Hash 表上查找一个值，有一个固定的代价(记为 k)，它与索引的大小无关，Hash Join 方法的比较次数大致为 $(|R_1|+k\cdot|R_2|)$，其中 $2<k<\log_2(|R_2|)$。

但是需要指出的是，在 MM-DBUG 方法下，对于外键与键之间的等值连接操作基本上是无时间代价的(因为它不需要执行任何比较)，而在实际应用中，绝大部分的连接操作都是外键与键之间的等值连接操作，因此，MM-DBUG 方法对于连接操作的执行性能是明显优于其它方法的。

对于投影操作，在第2部分已经进行过分析，这里就不重复了。

结束语 本文对文[2]中提出的内存数据库的图论存取方法进行了改进，改进后的图论存取方法不仅可以大大节省存储空间，而且对执行性能也有较大提高，特别是对于连接操作的执行性能会有很大的提高。本文还在文[2]的基础上，对几种存取方法在存储空间、执行时间方面的性能进行了进一步的深入分析，分析结果表明改进图论方法是更加有效的。

下一步的工作主要在内存数据库的并发控制、索引结构、查询算法、优化策略等方面。不过对于 MM-DBS 而言，查询优化将比磁盘数据库要简单得多，至少集簇问题以及通过投影来减少元组大小的问题可以不必考虑，因此也可以简化对算法的选择。另一方面，并发控制可能又会引发新的矛盾，由于在 MM-DBS 中所有的数据操作都是在内存中，这样相比之下锁的代价就会更大，例如，如果在元组粒度加锁的话，加锁的代价将与存取的代价相当，这样总的代价就是存取代价的2倍了，这太昂贵了。因此，应该在更大粒度的数据上加锁，这样可能更合算，相应地 MM-DBS 下的事务可能会更短(因为没有磁盘访问了)，因此锁数据的时间也就会更短。另外，如果整个 DB 不能全部装入内存的话，这时就还有一个内存与外存交换数据的策略与算法的问题。最后，内存数据库的恢复也是一个重要的问题，它与磁盘数据库的恢复策略与算法是相差很远的，今后我们将致力于这些问题的研究。

参 考 文 献

1 Liu Yun-Sheng, et al. Data organization and management of real-time main memory databases. Journal of Computer Research & Development, 1998, 35(5): 469～473 (in Chinese)
2 Liu Yun-Sheng, et al. Graph-Theoretic Access Methods for Main Memory Databases. Chinese Journal of Computers, 2001, 24 (10): 1095～1101 (in Chinese)
3 Lehman T J, Carey M J. Query Processing in Main Memory Database Management Systems. In: Proc of the ACM SIGMOD Int'l Conf on Management of Data, Washington, D.C., May 1986. 239～250
4 Ammann A C, et al. Design of a Memory Resident DBMS. In: Proc of the IEEE Compcon, San Francisco, 1985. 54～57
5 DeWitt D J, et al. Implementation Techniques for Main Memory Database Systems. In: Proc of the ACM SIGMOD Annual Meeting, Boston, Massachusetts, June 1984. 1～8
6 Bitton D, et al. Performance of Complex Queries in Main Memory Database Systems. In: Proc of the Third int'l Conf on Data Engineering, Los Angeles, California, Feb. 1987. 72～81