基于硬件的内存 trace 工具——MTT 的设计与实现

阮 元[1,2], 包云岗[1,2], 陈明宇[1], 樊建平[1]
(1. 中国科学院计算机系统结构重点实验室, 北京 100190; 2. 中国科学院研究生院, 北京 100049)

摘 要: 本文提出了一种全新的获得访存 trace 的方式, 并设计实现了基于硬件的零开销多平台实时访存 Trace 工具——MTT(Memory Trace Tool). 详细介绍了 MTT 在采样配置、地址识别、trace 输出等方面的设计细节, 以及接收端配合 MTT 高效接收分析 trace 的流程, 实现了一个通过 MTT 获得程序访存 trace 的完整方案. 相比已有方法, MTT 具有许多特点: (1) 对程序透明; (2) 零开销, 无内存污染问题; (3) 实时获取完整的全系统访存 Trace; (4) 可实时配置的多种在线 Trace 分析手段; (5) 具有操作系统平台无关性.

关键词: 访存轨迹; 插桩; 访存行为分析; 现场可编程门阵列 (FPGA)
中图分类号: TP333 文献标识码: A 文章编号: 0372-2112 (2008) 08-1519-07

The Design and Implementation of MTT——A Hardware-based Memory Trace Tool

RUAN Yuan[1,2], BAO Yun-gang[1,2], CHEN Ming-yu[1], FAN Jiang-ping[1]
(1. *Key Laboratory of Computer System and Architecture, Institute of Computing Technology, Chinese Academy of Sciences. Beijing* 100190, *China*;
2. *Graduate School, Chinese Academy of Sciences. Beijing* 100049, *China*)

Abstract: We propose a new approach to track memory reference trace and present a hardware-based Memory Trace Tool (MTT) for real systems. This paper details the design of MTT, including sampling configure, translating address, outputting trace and analysis of memory trace, which implement a complete solution to acquire and analyze memory reference trace. Compared to existing methods, MTT has several features: (1) be transparent to applications and system software; (2) be nearly zero-overhead and no memory pollutions; (3) be available to obtain full-system trace on real systems; (4) provide various online analysis approaches; and (5) be independent of system platform, support different CPU and OS platform.

Key words: memory reference trace; profiling; memory access behavior analysis; field-programmable gate array(FPGA)

1 引言

程序行为分析已成为体系结构、操作系统改进、编译优化和程序执行效率提高的最重要手段之一. 随着处理器与 Memory 之间的鸿沟越来越大, 如何消除或者改进"Memory Wall"[1] 成为研究的热点. 因此在程序执行过程的各种行为分析中, 访存行为分析尤其重要. 获得访存行为最常见、有效的方式是 Simulation 和 Profiling.

许多模拟器对内存体系结构有很好的支持, 也有一些专门的内存体系模拟器, 它们除了能支持多级 Cache 的模拟, 甚至能支持 RAM 的时序级模拟. 使用模拟器可以获得程序执行的所有访存 Trace, 但执行速度很慢, 而且大多数模拟器并不支持全系统模拟.

Profiling 是另一种有效的获得访存 Trace 的方式. 从软件的插入代码(Instrument) 到硬件方式的条件中断(Condition-Interrupt), 乃至软硬件结合的方式, 基本都是基于采样(Sampling) 的机制. 相比于模拟手段, Profiling 大大的减少了开销, 缩短了获得 Trace 的时间. 但也出现了新的问题, 如内存污染, 采样时机等. 产生这些问题的根源其实就是采样动作和存储 trace 对原始程序行为的干扰. 如果能避开这些干扰, 在速度上有优势的 Profiling 技术将有很强的竞争力.

本文设计实现了一种基于硬件的零开销多平台访存 Profiling 工具——MTT(Memory Trace Tool). 工作原理是将其直接插在主板的 DIMM 插槽内, 侦听并分析所有内存控制器发送至内存的控制命令, 从而实时获得系统

收稿日期: 2007-05-29; 修回日期: 2008-04-29
基金项目: 国家自然科学基金(No. 60633040)

真实、完整的访存Trace. 在内存总线上的采样动作对主机完全透明, 同时把存储 trace 的工作通过 MTT 上的千兆以太网转移给其它机器. 所以相比于其他 Profiling 工具, MTT 有很多新的特点: 全硬件实现, 独立于系统, 对应用程序透明; 不会产生内存污染问题; 可以实时获得全系统访存 Trace; 可动态配置以改变其不同的工作模式; 支持 Linux、Windows 等多种操作系统平台.

基于 MTT, 我们在 Linux 和 Windows 平台采用 SPEC2000 测试集进行实验, 并对一些如 Office 等日常应用程序进行实验, 均方便、有效地获得了访存不同粒度的Trace, 访存频率(Memory Access Frequency) 等重要的 Trace 数据. 实验表明, MTT 是一种高效、灵活的访存 Trace 工具.

2 MTT 的工作原理

MTT 的工作原理是总线监听方式. 内存控制器连接多个 DIMM, 每个 DIMM 槽均能接收到除 CS(片选信号) 外内存总线上的其它信号. 因此, 把 MTT 插入一个 DIMM 槽, 可以侦听到处理器与其它 DIMM 之间传输的命令和数据, 分析之后就能得到访存 trace.

MTT 不是真正的内存, 对处理器不可见, 独立于整个计算机系统, 所以它抓取访存 trace 的动作不会对系统运行产生任何影响. 不足之处是监听不到其它 DIMM 的 cs 信号. 如果处理器访问多个 DIMM 槽上的内存, MTT 将不能区分到底指令是发给了哪个槽上的内存, 所以在实验中只能使用一条内存. 选用大容量的内存可以弥补这个不足.

3 MTT 的结构和实现

MTT 是一块可插入 DIMM 槽的 PCB, 上面集成了 Xilinx 公司的 XC2VP20 和千兆 phy 芯片. 图 1 描述了 MTT 的大致结构和 FPGA 的逻辑模块. 其中, DDR Command Buffer Unit(DCBU) 负责监听采样从内存控制器发送至 DIMM 的命令. 采样后的 DDR 命令作为输入传递给 Config Unit 和 DDR State Machine Unit. Config Unit(CU) 的功能是识别一些定义为 MTT 控制命令的特殊访存地址序列, 将这些特殊地址序列转变为预定义好的命令以控制 MTT 分析模块的工作. DDR State Machine Unit (DSMU) 实现了 JEDEC DDRRAM 规范的子集, 能够对 DDR 内存的命令进行解析, 输出解析后的 DDR 命令组 (r/w, address). Statistic Unit(SU) 实现对地址分析统计功能, 包括记录不同采样粒度的地址流 Trace、不同时间间隔访存频率统计功能等. 分析后的数据将输入到 FIFO, 再通过 Tx thread 传给 GMAC core[18], 最后访存 trace 通过千兆以太网发送至接收端.

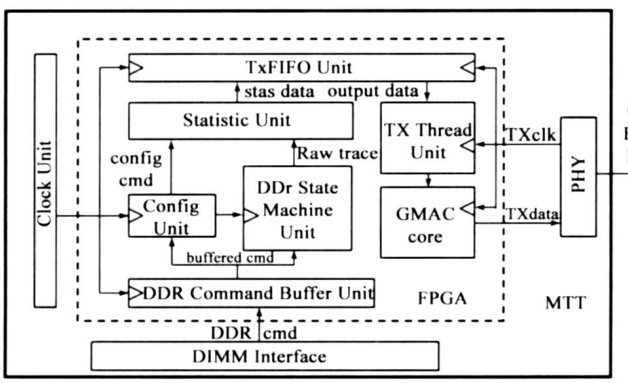


图 1 MTT 结构框图

总结一下, MTT 要抓取并分析访存 trace 需完成三个主要任务: (1) 将内存控制器发出的访存指令识别还原为 cpu 发出的物理地址 trace; (2) 通过与主机端的交互, 选择输出需要的访存 trace; (3) 将访存 trace 封装成以太网帧输出. 下面将分别介绍 MTT 完成这三个任务的具体实现和相关功能模块的详细设计.

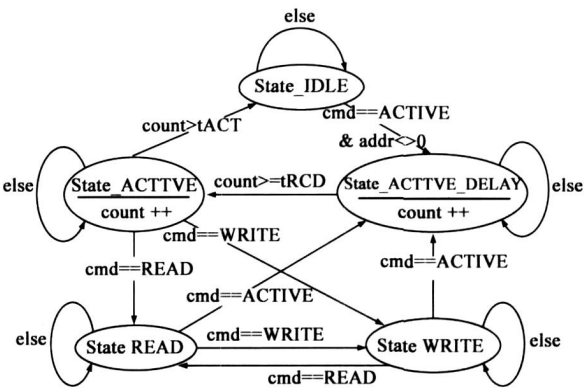


图 2 DDR State Machine

3.1 识别物理地址

MTT 在内存总线上监听的是内存控制器向 DDR SDRAM 发送的控制命令, 由 JEDEC DDR RAM 规范[17]规定. 每次访存操作, 内存控制器首先发出一个 ACTIVE 命令, 接着发出 READ 或者 WRITE 命令. 当发出 ACTIVE 命令时, 地址信号线用来选择需要访问的物理 bank 和 row. 当发出 READ 或者 WRITE 命令时, 地址信号线用来选择 col.

MTT 需要做的与内存控制器相反: 根据侦听到的 cmd、bank、row、col 还原出处理器访问的物理地址. 因为无需考虑传输的数据, 所以 ACTIVE、READ、WRITE 以及 PRECHARGE 这 4 个命令已经提供了足够的信息. 具体实现时, MTT 的 DDR 状态机单元(图 2) 简化了 JEDEC DDRRAM 中定义的状态转换, 遵从 JEDEC DDRRAM 规范的子集, 将内存控制器发送至 DDR SDRAM 的命令解析成 DDR 命令组(r/w,

physical address).

3.2 在线配置 MTT

MTT 输出 trace 的动作必须是可控的, 能根据需要任意启动或停止, 这样才可获得某具体程序的访存 trace. 而且考虑到可以在 FPGA 上实现多种 trace 统计功能, 也需要控制 MTT 选择指定的统计功能并输出分析后的数据. 所以如何配置 MTT 是非常重要的. 如果通过外部接口控制 MTT 的工作, 由于从外界无法获知程序运行的实际状况, 很难准确控制访存 trace 的输出.

为了实现在主机上直接操纵 MTT, 本文设计了基于内存总线上的访存指令实时控制 MTT 的机制, 通过编写程序连续若干次直接访问某个物理地址, 产生特定的物理地址序列, 在 MTT 上的 Config Unit 识别该物理地址序列并将其转换为对应的控制命令, 控制其它单元, 完成配置操作. 其原理如图 3 所示.

在 Linux 和 Windows 平台下, 都通过物理内存设备将 0×0~ 0×1000 这 4k 物理地址空间映射到进程虚拟地址空间, 就可以通过地址指针加上偏移(Offset) 来直接访问.

MTT 识别特定的物理地址序列需要解决几个问题: (1) MTT 配置寄存器定义. 寄存器的偏移(Reg_ Off) 与物理地址(Phy_ Addr) 的关系为 Phy_ Addr= Reg_ Off* 8(见图 3), 因此可得 4096/8byte= 512 个寄存器; (2) 访问冲突问题. 程序或者操作系统在运行过程中也会访问 0×0~ 0×1000 地址空间, Config Unit 可通过判断一段连续地址序列的方式来识别命令. 例如连续 32 次访问物理地址 0×80 会被识别为 Reset 操作; (3) Cache 缓冲问题. 默认情况下, 对配置寄存器的访问序列都会在 Cache 中命中, MTT 无法监听到. 因此通过 Linux 或者 Windows 提供的物理内存设备作地址映射时要显示的指明这段空间不能经过 Cache.

利用上述机制, 主机可按需控制访存 trace 输出的启动或停止, 实现灵活的功能. 如果拥有某类应用的源代码, 通过插入 MTT 的配置命令, 就可以获得任意某段代码执行的访存 trace. 在内核中使用类似手段, 也能成为分析操作系统行为的有力工具. 虽然额外的访存动作会带来一定开销, 但在大多数场景下, 开销是可以忽略的.

3.3 输出 trace

取得 trace 后, MTT 通过以太网帧的形式把访存 trace 输出到接收方. 这个过程需要解决的问题包括: 如何根据物理地址流构造访存 trace; 如何构造以太网帧输出访存 trace. 另外还要控制输出的 trace 以太网帧的大小, 虽然这对 trace 的记录没有影响, 但可减少接收端分析 trace 的开销.

3.3.1 Trace 的构成

MTT 能记录内存控制器发出的所有访存物理地址, 实际中要根据需求选取输出 trace 的精度. 从操作系统的角度来看, 访存是以页面为单位管理的, 以页面大小为单位记录 trace 是比较有效率的选择. 从 cpu 的角度来看, 访存是以 cache line 为单位执行的, 以 cache line 大小为单位的 trace 能完整记录程序的访存细节.

如果以页面为粒度记录 trace, 当连续的访存落入同一页面时, MTT 并不为每一次访存都记录一个 trace, 只通过若干计数器维护这个页面被连续访问的细节, 直到访问其它页面时才输出这个页面的访问 trace. 如果把 SU 输出 trace 的数据宽度定为 64bit, 这样除页面物理地址外, 还可根据需要加入时间戳、各种计数器等信息. 比如连续访问该页面时的读写次数, 连续访问该页面持续的时间等. 这些信息既能完整地描述程序访存行为, 实现的逻辑也并不复杂.

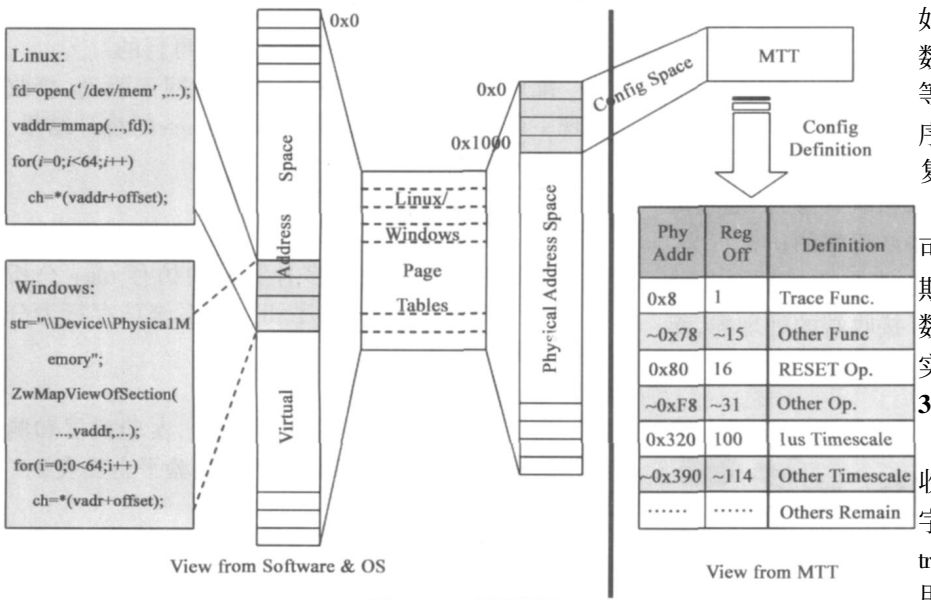


图 3　MTT 配置原理

除了访存的物理地址, SU 还可做些统计工作, 比如程序运行期间一定时间间隔内的访存次数. 统计的时间粒度从 1μs 至 1s 实现起来都很方便.

3.3.2 trace 以太网帧

MTT 的千兆输出直连到接收端, 所以无需包含 tcp/ip 协议字段, 帧的负载可全为访存 trace. 为了接收端的处理方便, 利用 MAC 地址字段记录 MTT 发出的帧的数目, 接收端的处理程序

可据此判断接收处理时有否丢帧.

3.3.3 输出 trace

如果 FIFO 中有数据时就生成新的帧输出, FIFO 为空时就结束当前帧. 当访存不太频繁时就会出现大量小帧, 它们可以拼成大帧输出以减少接收端的开销. 所以在 FIFO 和 Tx thread 之间设定一个控制信号, 当 FIFO 中的数据超过一个预设值时才启动 Tx thread 读数发帧. 即可保证输出的 trace 帧都是 1500 字节的大帧, 减少了接收端的开销.

延迟输出要求 FIFO 提供足够大的空间缓存 trace, 同时一个大的 FIFO 也能适度缓解访存 burst 对 MTT 输出 trace 的压力. 经过实验, 128KB 的 FIFO 可以满足所有 spec2000 程序抓取访存页面地址 trace 的需要. 表 1 列出了运行 ref 规模的 spec2000 测试程序得到的平均访存速率和页面地址、cache line 地址 trace 速率. 以及输出 cache line 地址 trace 时 FIFO 中所缓存数据大小的时间分布情况, 例如运行 bzip2 时, 有 96.9% 的时间, FIFO 中存储的数据超过 256byte. 从表中可以看出, 程序运行的大多数时间里, FIFO 中缓存的数据是比较少的, 1KB 和 2KB 间的巨大落差是由于前面提到的预设值造成的. 1G 以太网口在输出某些测试程序的 cache line 地址 trace 时会丢失少量 trace, 这个问题可以通过在 MTT 上加一个网口或是采用更高速的接口解决.

表 1 SPEC 程序访存 trace 流量统计

Spec2000	访存速率 (times/ms)	输出 trace 速率(MB/s)		输出 cache line 地址时 FIFO 使用情况统计(128KB FIFO)						
		页面地址	Cache line 地址	> 256	> 512	> 1k	> 2k	> 16k	> 64k	Full
bzip2	42234	28.5	36.7	96.9%	84.9%	50%	3.9%	3.1%	0.2%	0.02%
crafty	31788	21.8	27.3	90.7%	73.4%	39%	0	0	0	0
eon	11664	4.8	8.7	90.4%	73.8%	39%	0	0	0	0
gap	30176	14	29.3	99.2%	87.7%	54%	7.6%	7.3%	6.4%	0.27%
gcc	52800	25.8	44.5	93.8%	79.4%	45%	0.2%	0.01	0	0
gzip	36560	20	33.9	90.6%	74.3%	41%	1.3%	1.2%	1%	0.16%
mcf	63037	38.7	63.4	99.7%	84.9%	73%	13%	8.9%	0.2%	0.03%
parser	38577	17.2	36.1	94.3%	80.3%	45%	0	0	0	0
perlbmk	23222	13.5	24	92.5%	76.4%	41%	0	0	0	0
twolf	51634	32.3	48.7	98.9%	80.9%	43%	0	0	0	0
vortex	38556	19.3	32.8	96.7%	79.8%	43%	0.01%	0	0	0
vpr	47629	34.3	44.9	99.3%	91.1%	52%	0.2%	0	0	0

4 接收端实时处理

基于前面定义的 trace 数据构成和 MTT 输出的以太网帧格式, 就可以处理接收到的 trace 数据从而分析程序的访存行为. 比如访存的页面分布情况, 热页面访问频率统计, 对同一页面连续读写次数统计等. 但通过离线方式分析一个 ref 规模的 spec2000 程序的访存行为, 往往意味着从硬盘读取数十 G 的文件. 考虑到接收过程中 trace 数据已经完整地流过了接收端的处理器, 而且 MTT 发出的以太网帧不需要协议栈的处理, 所以接收 trace 的同时进行统计分析工作是可行的.

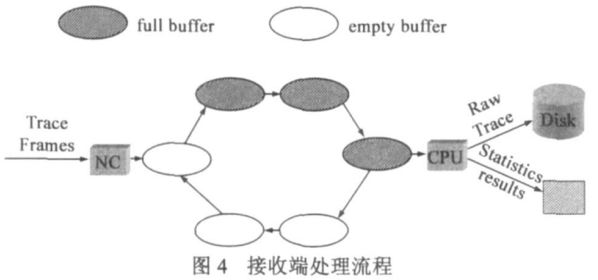


图 4 接收端处理流程

配合使用零拷贝零中断技术的网卡驱动, 接收端实现了存下完整 trace 后同时输出 trace 的统计结果. 工作流程见图 4.

5 MTT 实验评估

本节利用 MTT 获得多种实验的访存 trace 分析数据, 体现 MTT 作为应用程序和操作系统访存行为分析工具的强大功能.

5.1 实验环境

由于 MTT 零开销多平台的特性, 实验机器和测试应用的选择范围很广, 我们使用的实验平台见表 2.

5.2 正确性验证

为了验证 MTT 是否正确的将 DDR 总线上的命令还原为访存地址, 我们设计了如下实验. 在测试用机器的 grub.conf 加入"mem = 384M"保留上面的 128M 内存

(机器物理内存512M), 然后通过 ioremap 将保留的 128M 内存映射到内核空间. 顺序访问这 128M 空间, 然后通过一个简单的验证程序, 确认 MTT 输出的访存 trace 中访存地址大于 384M 的 trace 符合预期, 从而证实了 MTT 卡工作的正确性.

表2　实验平台

(*a*) 机器参数

Item	Machine
CPU	Intel Celeron 2G
L1 Cache	12K I, 6μop/Line, Pseudo-LRU 8K D, 4-way, 64B/Line, Pseudo-LRU
L2 Cache	128K, 2-way, 64B/Line, Pseudo-LRU
Chipset	Intel 845PE
Memory	512M DDR 200
Hard Disk	15G Maxtor
OS	Fedora Core 4(2.6.14 Kernel) Windows 2000

(*b*) 测试程序

OS / Application	Linux	Windows
Benchmark	CPU SPEC 2000	CPU SPEC 2000
Desktop App	Open Office 1.9.3	MS Office 2000
Multimedia	Realplayer 10	Realplayer 10

5.3　应用程序访存行为分析

通过 MTT 可以获得以时间为序的物理地址流. 分析这些原始 trace, 得到宏观的统计数据, 可以很直观地描述出应用程序的访存行为.

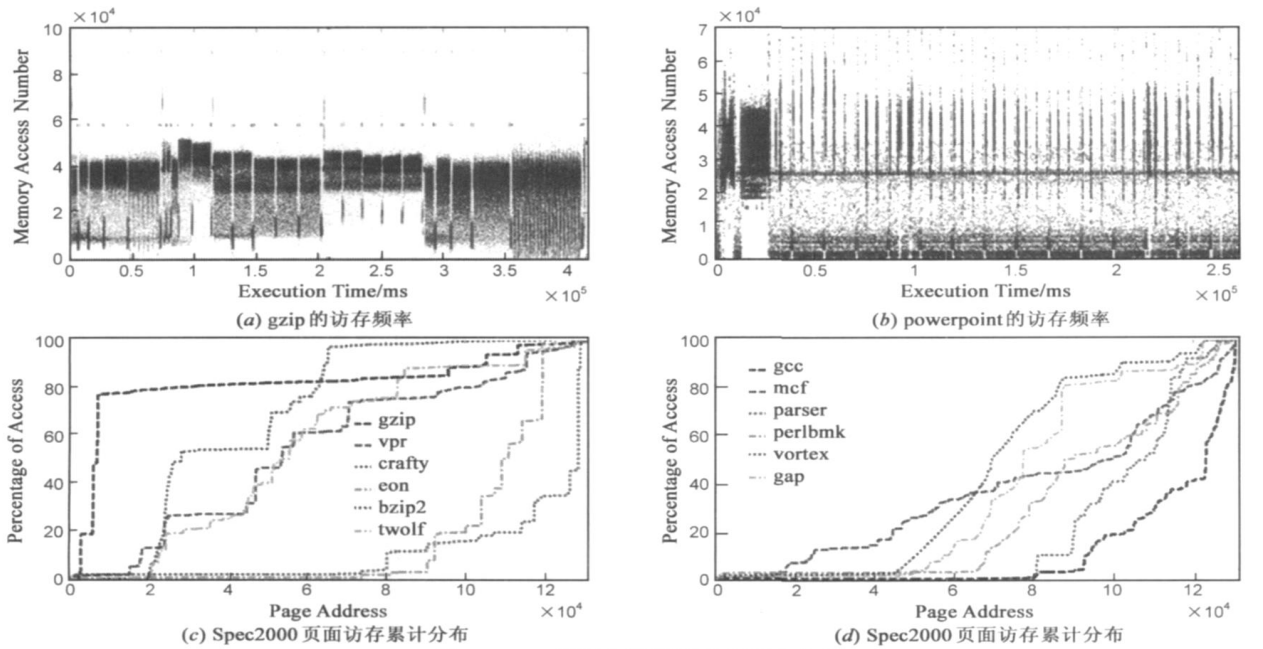
5.3.1　访存频率(Memory access frequency)

访存频率是评估 cache 性能的重要指标, 对系统功耗、分布式系统中共享内存的性能等也是重要的依据. MTT 的 SU 单元能统计各种时间粒度下的访存次数. 图 5(*a*) 是 Linux 下运行 gzip 得到的访存频率, 时间粒度是 1ms. 可以很明显的看出 gzip 的执行分五个阶段, 代表了 5 组不同的输入数据. MTT 还可以方便的获知 Windows 下各种办公应用的访存行为. 图 5(*b*) 是 Windows2000 下启动 powerpoint 并播放一个 25M51 页的 ppt 的访存频率. 从图中我们可以很清楚的分辨出整个访存 trace 分为启动 powerpoint, 打开 ppt 文件和播放三个阶段. 播放阶段中每隔一段时间出现的突发频繁访存即为打开新的一页 ppt 带来的访存操作, 而突发访存之间存在的固定访存则为 windows 操作系统引入的访存操作.

5.3.2　页面访存累计分布(Page Access Accumulation distribution)

页面访存累计分布是统计出每个物理页面在程序执行中被访问的次数, 得出页面访存概率分布, 即每个物理内存页面被访问的次数占总访存的比例, 进而得到页面访存累计分布.

图 5(*c*) 是 spec2000 程序的页面访存累计分布, 横坐标是物理地址去掉低 12 位后的物理页面号, 实验中使用的是 512MB 内存, 所以共有 131072 个物理页面. 图上的点即代表所有低于该点物理页面号的物理页面被访问次数占总访存的比重. 例如 GZIP 的页面访存累计分布, 位于低端地址的前 10000 个物理页面就占据了总



(*a*) gzip 的访存频率　(*b*) powerpoint 的访存频率

(*c*) Spec2000 页面访存累计分布　(*d*) Spec2000 页面访存累计分布

图 5　MTT 用于应用程序访存行为分析

访存数的 80%. 观察图 5(*c*) 列出的部分 spec2000 程序, 可以发现曲线呈阶梯状, 表明这几个基准测试程序都存在被频繁访问的热页面, 页面地址就是斜率较高的竖线所代表的部分. 图 5(*d*) 是 spec2000 中其余程序的页面访存累计分布, 这几条曲线的变化比较平缓, 表明这些程序访存相对比较平均, 热页面不多.

5.4 操作系统访存行为分析

操作系统在办公或服务器应用中会占据相当程度的开销. 如何改进OS 架构以加速各种应用是操作系统研究的热点.

MTT 可以很方便的从全系统访存 trace 中抽离出属于操作系统的部分, 只需在系统调用和中断处理的出入口插入MTT 的配置命令即可. MTT 识别出 OS 相关命令并输出特定 tag 供接收端辨认. 这段配置命令的执行时间仅为 5.9us, 对系统的整体性能影响很小. 经过实验, 通过MTT 获得了多种应用下内核的访存开销, 结果见表 3.

表 **3** OS 内核访存开销

Applications		Kernel Overhead(%)
Spec 2000	gzip	0.05
Desktop Applications	Real player	0.9
	Open Office	2.1
File operations * = Linux 2.6.14	rm- rf *	4.7
	tar- czf *	5.5
Context switch test	{wile(1) getpid( ) ; }	33

实验数据显示, 对于 SPEC 2000 benchmark 这类计算密集型应用, 操作系统的访存只占全系统访存极少的一部分. 而桌面办公应用中, 操作系统的访存比例也不到 3%. 对 IO 密集型应用, 如文件读写操作, 操作系统的访存比例大约为 5%. 为了检验极端情况下内核的上下文切换能占到多少访存, 设计了一个实验: main( ) {while(1) getuid( ) ; }. 测试结果显示, 内核的访存比例在 33% 左右.

实验表明, 利用 MTT 可以方便地区分特定代码段执行的访存 trace, 是研究操作系统行为和性能分析的有利工具.

6 相关研究

计算机体系结构越来越复杂, 计算机应用也越来越复杂, 因此程序行为分析对计算机体系结构的设计以及应用程序性能优化起着很重要的作用. 研究人员主要通过模拟与 Profiling 工具来分析程序行为特征.

现在大量的研究均基于模拟器, 通过模拟器获得数据来分析应用的行为[3~5]. SimpleScalar 是一个用户级模拟器, 研究人员利用 SimpleScalar 可以获得应用程序的行为特征, 但是它不能运行 OS, 因此无法用来研究包括操作系统的全系统行为. SimOS 是一个全系统模拟器, 它甚至可以引导加载商业级操作系统, 从开发至今, 很多人利用 SimOS 进行了全系统特别对操作系统有很深入的研究. 最近, M5 的出现也为全系统模拟器增加了新的选择. 随着体系结构的越来越复杂, 利用模拟器获得的数据也变得也越来越不可靠; 同时, 模拟器速度慢的问题无法得到解决, 这些都会成为将来利用模拟器进行体系结构研究的障碍.

Profiling 是一种有效的分析系统行为的方式. 通过在应用程序代码中插入特定的代码获得数据的代码, 然后直接运行应用程序便可以获得应用的行为数据如 ATOM[6], PIN[7]. 但是这种方式会带来干扰应用的执行行为, 如影响 Cache 的数据等.

硬件 Profiling 一般比软件 Profiling 对应用的影响要小很多, 并且能获得很高的速度. 当代处理器中都提供了相当的硬件寄存器, 利用这些内部寄存器, 可以很容易的统计一些事件的频率, 如 Cache Miss, TLB Miss, Branch 等. Burrows[8] 利用处理器内的Hardware Counter 有效的进行了 Value 的采样, Itzkowitz 等[9] 则对 Memory 进行 profiling. 在处理器内增加额外的部件可以提供更复杂的 profiling 功能, 如 ProfileMe、Fast On-Chip Profiler Memory 等[10~14]. 但由于在处理器内部, 受限于输出带宽, 它们不能输出更多的数据.

MemorIES[15] 和 PHA$E[16] 都是基于 FPGA 实现的实时访存 trace 分析工具, 利用 FPGA 模拟内存体系以优化 cache 设计. 但 MemorIES 只针对于 IBM's 6xx bus, PHA$E 则需要借助于逻辑分析仪进行信号采样, 所以在通用性易用性上不及 MTT. 而且均属于被动接收 trace, 没有提供主机端控制 trace 工具的机制, 无法像 MTT 一样根据程序运行情况按需主动控制 trace 的采样和分析.

7 总结与展望

MTT 是一种基于硬件的零开销多平台实时访存 Trace 工具. 它插在 DIMM 插槽监听系统的访存行为输出访存 trace, 并利用 FPGA 完成多种实时分析统计功能. 本文介绍了 MTT 的工作原理和实现关键, 给出了一套获得程序访存 trace 的完整方案. 通过实验表明, MTT 是一个精确、灵活、高效的 Trace 工具. 它相比于其他 Profiling 工具有诸多特点:

(1) 对应用程序透明;

(2) 零开销, 无内存污染问题(Memory Pollutions) ;

(3) 实时获取完整的全系统访存 Trace;

(4) 可配置的多种在线 Trace 分析手段;

(5) 具有操作系统平台无关性.

基于 MTT 下一步的工作包括: 在主机上跟踪程序的页表变化, 结合 MTT 输出的物理地址 trace 获得应用程序的虚拟地址访存 trace. 在 FPGA 上实现 trace 数据的压缩算法和更高级的分析功能如热页面和页面重用距离的实时分析统计. 我们还计划通过 MTT 研究多处理器多核结构下程序的访存行为.

参考文献:

[1] Win A Wulf, Sally A McKee. Hitting the Memory Wall: implications of the obvious[J]. Computer Architecture News, 1995, 23(1): 20- 24.

[2] SPEC CPU2000 V1. 2, http: //www. spec. org/cpu2000/, 2001- 10- 22.

[3] Timothy Sherwood, Suleyman Sair, Brad Calder. Phase tracking and prediction[A]. Proceedings of the 30th International Symposium on Computer Architecture[C]. San Diego: IEEE Computer Society, 2003. 336- 347.

[4] Suleyman Sair, Mark Charney. Memory Behavior of the SPEC2000 Benchmark Suite[R]. IBM TJ Watson Research Center Technical Report RC 21852, 2000.

[5] M Rosenblum, E Bugnion, etc. The impact of architectural trends on operating system performance[A]. Proceedings of the 17th International Symposium on Operating System Principles[C]. USA: ACM, 1995. 285- 298.

[6] A Srivastava, A Eustace. ATOM: A system for building customized program analysis tools[A]. Proceedings of the Conference on Programming Language Design and Implementation [C]. Orlando: ACM, 1994. 196- 205.

[7] Pin. http: //rogue. colorado. edu/Pin/docs/11889/Doc/Pin/html/, 2007- 05- 13.

[8] M Burrows, U Erlingson, etc. Efficient and flexible value sampling[A]. Proceedings of the 9th International Conference on Architectural Support for Programming Languages and Operating Systems[C]. Cambridge, 2000. 160- 167.

[9] Marty Itzkowitz, Brian J N, Wylie Christopher, Nicolai Kosche. Memory profiling using hardware counters[A]. Proceedings of Supercomputing[C]. Phoenix: ACM, 2003.

[10] Jeffrey Dean, James E Hicks, etc. ProfileMe: Hardware support for instruction level profiling on out of order processors[A]. Proceedings of the 33rd International Symposium on Micro architecture[C]. USA: ACM/IEEE, 1997. 292- 302.

[11] Roman Lysecky, Susan Cotterell, Frank Vahid. A fast on chip profiler memory[A]. Proceedings of the 39th Design Automation Conference[C]. New Orleans: ACM, 2002. 28- 33.

[12] S Subramanya Sastry, Rastislav Bod, James E Smith. Rapid profiling via stratified sampling[A]. Proceedings of the 28th Annual International Symposium on Computer Architecture [C]. Göteborg, 2001. 278- 289.

[13] M Merten, etc. A hardware-driven profiling scheme for identifying program hot Spots to support runtime optimization[A]. Proceedings of the 26th International Symposium on Computer Architecture[C]. Atlanta, 1999. 136- 147.

[14] Ann Gordon-Ross, Frank Vahid. Frequent loop detection using efficient non-intrusive on-chip hardware[A]. Proceedings of the International Conference on Compilers, Architectures and Synthesis for Embedded Systems[C]. San Jose: ACM, 2003. 117- 124.

[15] Ashwini Nanda, Kwok-Ken Mak, Krishnan Sugavanam, Ramendra K Sahoo, Vijayaraghavan Soundararajan, T Basil Smith. MemorIES: a programmable, real-time hardware emulation tool for multiprocessor server design[A]. Proceedings of the 9th International Conference on Architectural Support for Programming Languages and Operating Systems[C]. Cambridge, 2000. 37- 48.

[16] Chalainanont N, Nurvitadhi E, Morrison R, Lixin Su, Kingsum Chow, Shih-Lien Lu, Lai K. Real-time L3 cache simulations using the Programmable Hardware-Assisted Cache Emulator (PHA$E)[A]. IEEE 6th Annual Workshop on Workload Characterization[C]. Austin: IEEE, 2003.

[17] JEDEC SOLID STATE TECHNOLOGY ASSOCIATION. Double Data Rate(DDR) SDRAM Specification[S]. 2004.

[18] Xilinx, Inc. Xilinx 1-Gigabit Ethernet MAC v8. 1[EB/OL]. http: //www. xilinx. com/ipcenter/catalog/logi core/docs/gig_ eth_ ac. pdf, 2006- 09- 21.

[19] Yungang Bao, Mingyu Chen, Yuan Ruan, Jianping Fan, etc. HMTT: a platform independent full-system memory trace monitoring system[A]. International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS 2008)[C]. Annapolis, Maryland, USA, 2008.

作者简介:

阮　元　男, 1983 年生于山东济南, 博士研究生. 主要研究方向为高性能计算、可重构计算. E-mail: ry@ncic. ac. cn

包云岗　男, 1980 年生于江苏宜兴, 博士研究生. 主要研究方向为高性能计算、系统软件.