HMTT: A Platform Independent Full-System Memory Trace Monitoring System

Yungang Bao[†‡], Mingyu Chen[†], Yuan Ruan[†‡],Li Liu[†‡]
Jianping Fan[†], Qingbo Yuan[†‡], Bo Song[†‡], Jianwei Xu[†‡]

† Key Laboratory of Computer System and Architecture, Institute of Computing Technology, Chinese Academy of Sciences, Beijing, China
‡ Graduate School of Chinese Academy of Sciences, Beijing, China

{baoyg,ry,liuli,yuanbor,songbo,xjw }@ncic.ac.cn {cmy,fan}@ict.ac.cn

Abstract

Memory trace analysis is an important technology for architecture research, system software (i.e., OS, compiler) optimization, and application performance improvements. Many approaches have been used to track memory trace, such as simulation, binary instrumentation and hardware snooping. However, they usually have limitations of time, accuracy and capacity.

In this paper we propose a platform independent memory trace monitoring system, which is able to track virtual memory reference trace of full systems (including OS, VMMs, libraries, and applications). The system adopts a DIMM-snooping mechanism that uses hardware boards plugged in DIMM slots to snoop. There are several advantages in this approach, such as fast, complete, undistorted, and portable. Three key techniques are proposed to address the system design challenges with this mechanism: (1) To keep up with memory speeds, the DDR protocol state machine is simplified, and large FIFOs are added between the state machine and the trace transmitting logic to handle burst memory accesses; (2) To reconstruct physical-to-virtual mapping and distinguish one process' address space from others, an OS kernel module, which collects page table information, and a synchronization mechanism, which synchronizes the page table information with the memory trace, are developed; (3) To dump massive trace data, we employ a straightforward method to compress the trace and use Gigabit Ethernet and RAID to send and receive the compressed trace.

We present our implementation of an initial monitoring system, named HMTT (Hyper Memory Trace Tracker). Using HMTT, we have observed that burst bandwidth utilization is much larger than average bandwidth utilization, by up to 5X in desktop applications. We have also confirmed that the stream memory accesses of many applications contribute even more than 40% of L2 Cache misses and OS virtual memory management may decrease stream accesses in view of memory controller (or L2 Cache), by up to 30.2%. Moreover, we have evaluated OS impact on memory performance in real systems. The evaluations and case studies show the feasibility and effectiveness of our proposed monitoring mechanism and techniques.

SIGMETRICS'08, June 2–6, 2008, Annapolis, Maryland, USA.

Categories and Subject Descriptors

B.8 [Performance and Reliability]: Performance Analysis and Design Aids; C.4 [Performance of Systems]: Measurement techniques; D.4 [Operating Systems]: Miscellaneous

General Terms

Measurement Performance

Keywords

Real System, Memory Trace, DIMM, HMTT

1. Introduction

Although the "Memory Wall [50]" problem has been raised for a decade, this trend remains in multicore era. Both memory latency and bandwidth become critical. On the other hand, for high-performance DRAM memories, thermal control has already become a realistic issue [27]. To evaluate thermal models, an interval of at least thousands of seconds is needed [31]. Memory trace analysis is an important technology for architecture research, system software (i.e., OS, compiler) optimization, and application performance improvements.

Uhlig and Mudge [52] suggest that an ideal memory trace collector should be:

- **Complete**: Trace should include all memory references made by OS, libraries and applications;
- **Detail**: Trace should contain detail information to distinguish one process' address space from others;
- **Undistorted**: Trace should not include any additional memory references. Trace should have no time dilation.
- **Portable**: Trace can still be tracked when moving to other machines with different configurations.
- **Other characteristics**: An ideal trace collector should be fast, inexpensive and easy to operate.

Memory trace can be collected in several ways, such as by software simulators, binary instrumentation, hardware counters, hardware monitors, and hardware emulators. Table 1 summarizes these approaches. Nevertheless, all of these approaches have their shortcomings. (Detailed in Section 2)

Table 1. Summary of Memory Trace Trackers

	Simul-ation	Instru-ment	HW counter	HW Monitor	HW Emulate
Complete	*	*	x	√	√
Detail	√	*	x	x	√
Undistorted	√	x	√	√	x
Portable	√	*	*	x	*
Fast	x	x	√	√	√
Inexpensive	√	√	√	*	x

Note: √ – Yes * – Maybe x – No

In this paper we propose a platform independent full system memory trace monitoring system. The system adopts a DIMM-snooping mechanism, which uses hardware boards plugged in DIMM slots to track virtual memory reference trace of full systems (including OS, VMMs, libraries, and applications). Several new techniques are proposed to overcome the system design challenges with this mechanism: (1) To keep up with memory speeds, the DDR state machine [3] is simplified to match high speed, and large FIFOs are added between the state machine and the trace transmitting logic to handle occasional bursts; (2) To reconstruct physical-to-virtual mapping and to distinguish one process' address space from others, an OS kernel module which collects page table information and a synchronization mechanism which synchronizes page table information with memory trace are introduced; (3) To dump full mass trace, we use a straightforward method to compress memory trace and adopt a combination of Gigabit Ethernet and RAID to transfer and save the compressed trace.

The monitoring system with the DIMM-snooping mechanism and techniques has the following advantages:

Complete: It is able to track complete memory reference trace of real systems, including OS, VMMs, libraries, and applications.

Detail: The memory trace includes timestamp, r/w, process' pid, page table information, and kernel entry/exit tags etc. It is easy to differentiate processes' address spaces.

Undistorted: There are almost no additional references except synchronization operations between memory trace and page table information. The operations incur only less than 1% additional references and about 1% additional execution time.

Portability: The hardware boards are plugged in DIMM slots which are widely used. It is easy to port the monitoring system to machines with different configurations (CPU, bus, memory etc.).

Fast: There is no slowdown when we collect memory trace for analysis of L2/L3 cache, memory controller, DRAM performance and power etc. The slowdown factor is about 10X~100X when cache is disabled to collect whole trace.

Inexpensive: We have built an initial monitoring system, from schematic, PCB design, FPGA logic to kernel modules, and analysis programs. The implementation of hardware boards is simple and low cost (< $500).

Easy to operate: It is easy to operate the system, with several toolkits for trace generation and analysis.

We present the implementation of an initial monitoring system, named HMTT (Hyper Memory Trace Tracker). HMTT consists of three parts: a Memory Trace Board (MTB, an FPGA board) plugged in a DIMM slot, a Kernel Synchronization Module (KSMod) synchronizing page table information with memory trace, and a trace capture and analysis toolkit. In addition to tracking trace, it can also be reconfigured to support online analyses, such as memory bandwidth statistical analysis, page reuse distance calculation, and hot pages collection. HMTT can only monitor DDR-200 DIMMs currently. Using HMTT in X86/Linux platforms, we have observed that various applications (SPEC CPU 2000/2006, desktop applications, and SPECJbb 2005) have burst memory access behaviors, and the burst bandwidths may be much more than average bandwidth, by up to 5X. We have also confirmed that stream-based memory accesses of many applications account for more than 40% of L2 Cache miss accesses. But OS virtual memory management may decrease stream accesses in view of memory controller (or L2 Cache), by up to 30.2% (301.apsi). Moreover, we have evaluated OS impact on memory performance in a real system. The evaluations and case studies show the feasibility and effectiveness of the monitoring mechanism and techniques.

In summary, we have made the following contributions in this paper:

- We propose a platform independent full system memory trace monitoring system, which adopts a DIMM-snooping mechanism and several new techniques. We have implemented an initial monitoring system — HMTT (Hyper Memory Trace Tracker). HMTT confirms that the monitoring mechanism and techniques are feasible and effective.
- We propose three new techniques to overcome the system design challenges with this mechanism: a simplified DDR state machine to keep up with memory speeds, a Kernel Synchronization Mechanism (KSM) to differentiate processes' address spaces, and a combined approach of GE and RAID to dump full mass trace.
- With HMTT, we have evaluated memory burst bandwidth, stream-based memory accesses among L2 Cache misses and the impact of OS on memory system of three categories applications (SPEC CPU, Desktop and Java) in Intel Celeron and AMD Opteron platforms. The case studies and evaluations indicate several advantages of HMTT, such as complete, detail, undistorted, portability and fast.

The rest of the paper is organized as follows. Section 2 presents an overview of related work, while Section 3 describes the design goals and challenges of the monitoring system. Section 4 presents the detail design, and section 5 discusses the implementation and validation of the initial system -- HMTT. Section 6 presents several case studies of HMTT to show its feasibility and effectiveness. Section 7 discusses the evaluations and limitations. Finally, Section 8 summarizes the system and discusses the future work of the monitoring system.

2. Related Work

There are several areas of effort related to memory trace monitoring: software simulators, binary instrumentation, hardware counters, hardware monitors, and hardware emulators etc.

Software simulators: Most memory performance and power researches are based on simulators. They utilize cycle-accurate simulators to generate memory trace and then feed trace to trace-driven memory simulators (e.g. DRAMSim [47], MEMsim [39]). SimpleScalar [9] is a popular user-level simulator, but it can not run operating system for analysis of full system behaviors. Several full system simulators (such as SimOS [41], Simics [34], M5 [5] and QEMU [16]), which can boot commercial operating systems, are commonly used in research when deal with OS-intensive applications. However, software simulators usually have speed and scalability limitations. As the computer architectures become more and more sophisticated, more detail simulation models are need, which may lead to a slowdown of 1000X~10000X [15]. Moreover, simulation with complex multicore and multi-threaded applications may incur inaccuracies and could lead to misleading conclusions [36].

Binary instrumentation: Many binary instrument tools (e.g. O-Profile [6], ATOM [46], DyninstAPI [1], Pin [7], Valgrind [10], Nirvana [17] etc.) are popularly utilized to profile applications. They are able to obtain applications' virtual access trace even

without source codes. Nevertheless, few of them can provide full system memory trace because instrumenting kernels is very tricky. PinOS [18] is an extension of the Pin [7] dynamic instrumentation framework for full-system instrumentation. It is built on top of the Xen [12] virtual machine monitor with Intel VT [37] technology and, is able to instrument both kernel and user-level code. However, PinOS can only run on IA-32 in uni-processor mode. Moreover, binary instrumentation method usually only slows down target programs' execution, incurring time distortion and memory access interference.

Hardware counters: Hardware counters are able to provide accurate events statistic (e.g. Cache Miss, TLB Miss, etc.). Itanium2 [2] is even able to collect trace via sampling. The approach of hardware counters is fast, low overhead, but they can not track complete and detailed memory reference trace.

Hardware monitors: Various Hardware monitors, divided into two classes, are able to monitor memory trace online. One class is pure trace collectors, and another is online cache emulators. BACH [22, 23] is a trace collector. It utilizes a logic analyzer to interface with host system and to buffer the collected traces. When the buffer is full, the host system is halted by an interrupt and the trace is moved out. Then, the host system continues to execute programs. BACH is able to collect traces from long workload runs. However, this halting mechanism may alter original behavior of programs. The hardware-based online cache emulation tools (such as MemorIES [36], PHA$E [19], RACFCS [51], ACE [25], and HACS [49]) are very fast and have low distortion and no slowdown. Logic analyzer is also a powerful tool for capturing signals (including DRAM signals) and can be very useful for hardware testing and debugging.

However, these hardware monitors have several disadvantages: (1) they (except BACH) are not able to dump full mass trace but only produce short traces due to small local memories; (2) they can not distinguish one process' address space from others but only track physical address due to the lack of physical-to-virtual mapping information; (3) they depend on proprietary interfaces, for example, MemorIES relies on the IBM's 6xx bus, BACH, PHA$E, ACE, HACS etc. adopt logic analyzer which is quite expensive. RACFCS use a latch board that directly connects to output pins of specified CPUs. So they have poor portability.

Hardware emulators: Several hardware emulators are thorough FPGA-based systems which utilize a number of FPGAs to construct uni-processor/multi-processor research platforms to accelerate research. For example, RPM [14] emulates the entire target system within its emulator hardware. Intel proposed an FPGA-based Pentium system [33] which is an original Socket-7 based desktop processor system with typical hardware peripherals running modern operating systems. RAMP [8] is also a new scheme for architecture research. Although they do not produce any memory traces currently, they are capable of tracking full system trace. But they can only emulate a simplified and slow system with relative fast I/O, which fulfills the "CPU-memory / memory-disk" gaps that may be bottlenecks in real systems.

3. The Platform Independent Full-System Memory Trace Monitoring System

We propose a platform independent full-system memory trace monitoring system. In this section, we discuss the design goals and challenges of such a system.

The system is designed to be able to track **complete, detail, and undistorted** trace. Moreover, the system should be **portable, fast, inexpensive, and easy to operate**. To achieve these goals, we adopt a DIMM-snooping mechanism that uses hardware boards plugged in the DIMMs to snoop. Although the idea is straightforward, it is able to track full-system memory reference trace from OS kernel, VMMs, libraries, and applications. Moreover, DIMMs are independent of hardware platforms (such as CPUs, buses, and memory controllers), and they are widely used in modern machines. The trace monitoring system can be used in various hardware platforms.

Several obstacles must be overcome to design and implement such a monitoring system:

How to keep up with memory speeds?

Although memory frequency is lower than CPU, usually it is still more than 200MHz. For example, the frequency of DDR memory is about 200 ~ 400MHz, and currently dominant DDR2 memory has increased to a 533 ~ 800MHz clock rate. Moreover, DDR3 memory may reach to a 1600MHz rate in the forthcoming future.

On the other hand, DDR commands to multi-bank memories are interleaved, which requires sophisticated logic to interpret. DDR read/write operations are performed in two phases. First, an ACTIVE command is used to open a row in a particular bank for a subsequent access. Then, the READ/WRITE commands are issued to the row of that bank [3]. Multiple access requests can be issued because of the multi-bank architecture, which provides high effective bandwidth. Figure 1 depicts a scenario where a command pattern is issued to a multi-bank memory.

Thus the snooping logic should handle the high frequency and the interleaved commands to keep up with memory speeds.

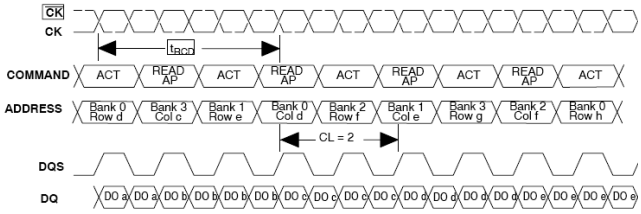


Figure 1. DDR Interleaved Commands to Multi-Bank [3]

How to reconstruct physical-to-virtual mapping?

Although physical traces are useful for memory system research, they can not provide deep insights into specified processes and OS kernels. As known to all, virtual memory mechanism isolates address spaces of processes running on one machine simultaneously. It is impossible to differentiate processes' address spaces without physical-to-virtual mapping information, especially in multi-processes environments.

Moreover, a new challenge is posed when given the mapping information. That is how to synchronize the mapping information with memory trace for replaying virtual memory trace correctly and effectively.

It should be noted that none of previous hardware monitors provides the physical-to-virtual mapping information, although they are able to track complete, undistorted and full-system physical trace fast.

How to dump mass trace?

Usually, memory reference traces are generated at very high speed. Our experiments show that most applications generate memory trace at bandwidths of more than 30MB/s even when utilize the DDR-200MHz memory (Detailed in Section 7.1). An execution of a 10-minutes interval would generate more than

18GB memory reference trace. Moreover, the high frequency of the DDR2/DDR3 memory and the prevalent multi-channel memory technology increase trace data generation bandwidth further, up to 100X MB/s.

Most previous hardware monitors utilize local memory to store trace. BACH has made an improvement, adopting a technique which is able to dump trace from long workload runs by halting host system when local memory is full. However, the local memory will be exhausted in a very short interval due to the trace generation bandwidth of more than 100MB/s. Thus, a new technique is demanded to sustain dumping mass trace.

4. The Trace Monitoring System Design

To overcome the above three challenges, we propose several new techniques in the trace monitoring system design. We also develop some toolkits to facilitate system operations. In this section, we will detail the top-down design of the trace monitoring system.

4.1 Top-Level Design

At the top-level, the monitoring system mainly consists of six procedures for memory trace tracking and replaying. Figure 2 shows the system framework and the six procedures.

From Figure 2, the monitoring system utilizes several hardware monitor boards plugged into DIMM slots of a traced system. The main memories of the traced system are plugged into the DIMM slots integrated on the hardware monitoring boards. The boards snoop on all memory commands via DIMM slots (see ①) . An on-board FPGA converts the commands into memory traces in this format *<address, r/w, timestamp>*. Each hardware monitor board generates trace separately and sends the trace to its corresponding receiver via Gigabit Ethernet (see ②). With the synchronized *timestamps*, the separated traces can be merged in the trace replay phase (see ③). Meanwhile, a module injected into OS kernel collects page table information and synchronizes the information with memory trace dynamically (see ④). Then the page table information is used to reconstruct physical-to-virtual mapping information (see ⑤). Based on the information, i.e. memory trace, virtual-physical mapping and synchronization tags, we are able to perform trace replaying procedure correctly and effectively for offline analysis (see ⑥).

The three challenges mentioned previously are hidden in the six procedures. Procedure ① faces the challenge that how to keep up with memory speeds; procedures ④&⑤ encounter the challenge of physical-to-virtual mapping reconstruction; procedures ②&③ demand to solve the problem of dumping mass trace. We will elaborate on our solutions in the following sections.

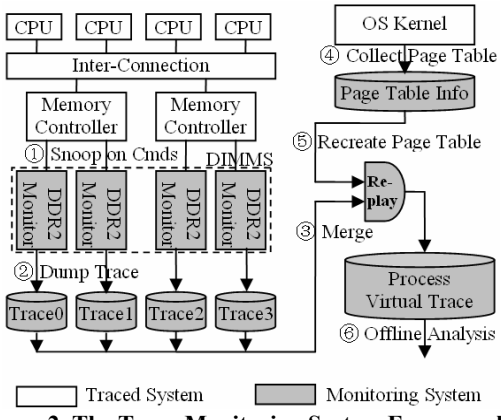


Figure 2. The Trace Monitoring System Framework

4.2 Keeping up with Memory Speeds

As mentioned previously, fast and efficient control logic is demanded to keep up with memory speeds because of high memory frequency and multi-bank technologies. Since only memory address is indispensable for tracking trace, we could only snoop on DDR commands at half memory data frequency. For example, if use DDR2-533MHz memory, the control logic can operate at a frequency of only 266MHz, at which most advanced FPGAs are competent to work.

To interpret the two-phase read/write operations, the DDR SDRAM specification [3] defines seven commands and a state machine which has more than twelve states. Commercial memory controllers may integrate even more complex state machines which cost both time and money to implement and validate. Nevertheless, we find that only three commands, i.e. ACTIVE, READ and WRITE, are necessary for memory reference address extraction. Thus, we design a simplified state machine to interpret the two-phase operations for one memory bank. Figure 3 shows the simplified state machine. It has only four states and performs state transition based on the three commands. The state machine is so simplified that its implementation in a common FPGA is able to work at a high frequency. Our experiments show that the state machine implemented in a Xilinx Virtex II Pro FPGA is able to work at a frequency of more than 300MHz.

On the other hand, applications may generate occasional bursts which may induce dropping trace. A large FIFO between the state machine and the trace transmitting logic is provided to solve this problem. In our initial system HMTT, we have verified that a 16K entries FIFO is sufficient to match the state machine for DDR 200MHz memory and a transmission bandwidth of 1000 Mbps. Moreover, few applications exhaust more than 8K entries except 181.mcf (SPEC CPU2000). Of course, to adopt a higher bandwidth (e.g. use two Gigabit Ethernets) for trace transmission is an alternative to reduce the FIFO size.

4.3 Physical-to-Virtual Mapping Reconstruction

There are two problems to reconstruct physical-to-virtual mapping: 1) how to collect page table information; 2) how to

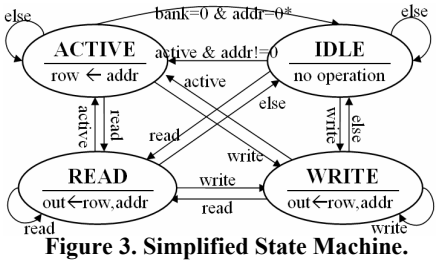


Figure 3. Simplified State Machine.
*** *addr* is used to filter special address for configuration.**

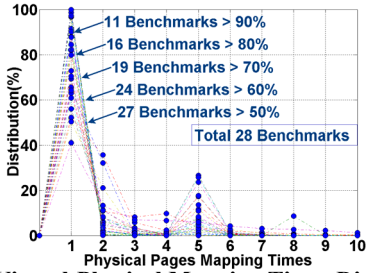


Figure 4. Virtual-Physical Mapping Times Distribution

synchronize the information with physical memory trace. We introduce a Kernel Synchronization Mechanism (KSM) to handle these problems.

The KSM is able to track each update of page table, in the form of <*pid, phy_page, virt_page, page_table_entry_addr*>. The form indicates that a mapping between physical page *phy_page* and virtual page *virt_page* is created for process *pid,* and the mapping information is stored in the memory location of *page_table_entry_addr*. When a page fault occurs, the KSM captures and stores each update of page table in the above form. Thus, given one physical address, the corresponding process and virtual address can be retrieved from the page table information.

On Linux platform, the KSM provides an *hmtt_printk* routine which can be called at any place from the kernel. Unlike Linux kernel's *printk*, the *hmtt_printk* routine supports large buffers and user-defined data format, like some popular kernel log tools, such as LTTng[4]. The KSM requires a kernel buffer to store the page table information. Figure 4 shows that most physical pages are mapped to virtual pages only once during application's entire execution (applications are list in Table 3). Under this observation, we find that it is enough to allocate a kernel buffer by only 0.5% of total memory size for storing page table information.

Every *N* page faults, the *hmtt_printk* routine will send a synchronization tag to the hardware monitor boards. The choice of the number *N* is sensitive. An ideal *N* should satisfy two requirements that 1) one physical page will not be remapped in *N* consecutive page faults and 2) the *N* should be large to increase synchronization interval, consequently reduce the synchronization overhead. It is a tradeoff that a smaller *N* indicates more accuracy and a larger *N* means less overhead. Based on experimental results, we find that one physical page is almost never remapped in 50 consecutive page faults. Moreover, when *N=50*, the synchronization overhead is very small, about one thousand additional synchronization trace per billion. So, we choose *N=50* in our system implementation. We will detail the KSM implementation on Linux platform in Section 5.5.

4.4 Dumping Mass Trace

The memory trace size is dependent on two factors, trace generation bandwidth and application's execution time. Our efforts mainly focus on reducing trace generation bandwidth.

First, we adopt several straightforward methods to reduce the memory trace generation and transmission bandwidth. When memory works in burst mode [3], we only need to track the first memory address of an addresses pattern. For example, when the burst length is equal to four, the latter three addresses of a 4-length addresses pattern are ignored. Trace format is usually defined as <*address, r/w, timestamp*> which needs at least 6~8 bytes to store and transmit. We find that the high bits of the difference of *timestamps* in two consecutive traces are always 0s at most time. We use *duration* ($=timestamp_n – timestamp_{n-1}$) to replace *timestamp* in the trace format. This differencing method reduces the *duration* bits to ensure one trace to be stored and transmitted in 4 bytes. However, the *duration* may overflow. We define a special format <*special_identifier, duration_high_bits*> to handle the overflows. Then, the *timestamps* can be calculated in the trace replay phase. The straightforward compression methods reduce trace generation and transmission bandwidth significantly.

Second, the experimental results show that trace generation bandwidth is still high with the above compressions. As depicted in Figure 2 procedure ②, we utilize multiple Gigabit Ethernets (GE) and RAIDs to send and receive memory traces respectively. In this method, all traces are received and stored in RAID storages (the details about trace generation and transmission bandwidth will be discussed in Section 7.2). Each GE sends trace respectively, so the separated traces need to be merged when replay. As shown in Figure 2 procedure ③, each trace has its own *timestamp.* The *timestamps* are synchronized by a toolkit when the monitoring system starts working. Once the base *timestamps* of all monitor boards are synchronized, they increase at the same memory clock rate respectively. Then the trace merge operation is simplified to be a merge sort problem.

The combination of the straightforward compressions, the GE-RAID approach, and the trace merge procedure makes the monitoring system be able to dump mass trace. Moreover, these techniques are scalable for higher trace generation bandwidth.

4.5 Other Design Issues

The hardware monitor system requires a configuration mechanism. We define some special addresses as configuration registers. To differentiate normal accesses, only a continuous access pattern to one specified address will be translated into an inner-command to control the hardware monitor boards. For example, the inner-command RESET operation is defined as a pattern of 16 continuous references on the 0x80 address. The cache influence is also a problem. Fortunately, pages have a cache attribution which can be altered by OS. When system powers on and OS boots, we can reserve several pages which will be set to be uncacheable later. These pages are defined as a configuration space of hardware monitor boards. More details of implementation will be present in Section 5.4.

As the on-board FPGAs are reconfigurable, we also design some online analysis functionalities, such as memory bandwidth statistic, page-level statistic (hot pages and reuse distance), and reference address bit change statistic.

5. The Implementation of HMTT

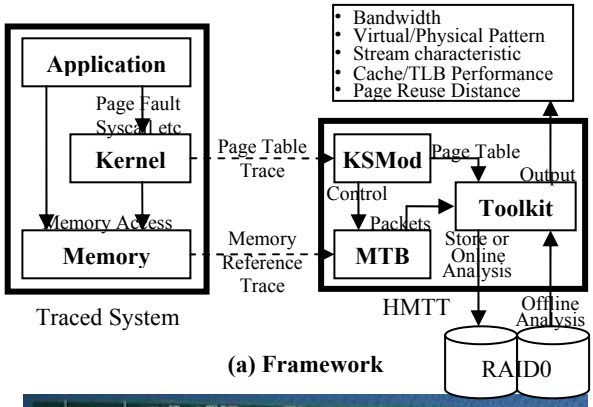
At the first stage, we have implemented an initial monitoring system according to the above design, named HMTT (Hyper Memory Trace Tracker). HMTT, which is able to monitor DDR-200 DIMMs currently, consists of a Memory Trace Board (MTB, an FPGA board) plugged in a DIMM slot, a Kernel Synchronization Module (KSMod) synchronizing page table information with memory trace, and a trace capture and analysis toolkit. Besides tracking trace, HMTT is reconfigurable to support online analysisses, such as memory bandwidth statistic, page reuse distance calculation, and hot pages collections.

5.1 Detail Framework of HMTT

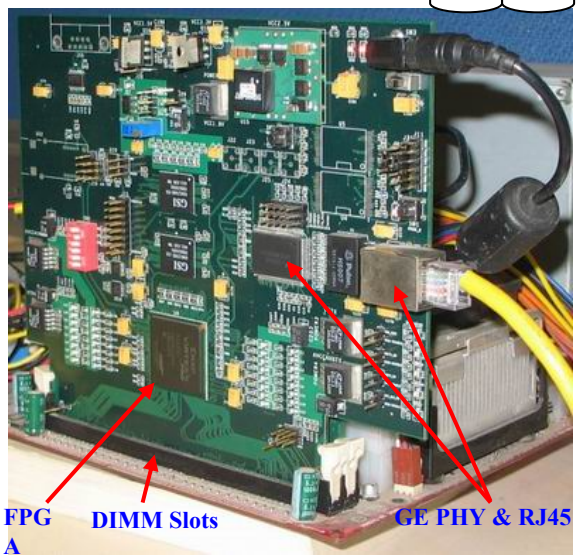
As Figure 5(a) shows, HMTT comprises a Memory Trace Board (MTB) plugged in DIMM slot, a Kernel Synchronization Module (KSMod) and a trace packets capture and analysis Toolkit.

MTB, which is a hardware monitor board without on-board DIMM currently at the first stage, is plugged in an idle DIMM slot (see Figure 5(b)). It snoops on memory command signals which are sent to DDR SDRAM from memory controller. MTB captures the DDR commands, and forwards them to the simplified DDR state machine (described in Section 4.2). The output of state machine is a tuple <*address, r/w, duration*>. These raw traces are sent out via GE directly or inputted for online analysis.

KSMod is an instance of the Kernel Synchronization Mechanism (KSM) on Linux platforms. KSMod comprises two



(a) Framework



(b) HMTT Hardware Board

Figure 5. (a) A detail Framework of the initial monitoring system. HMTT (b) HMTT hardware board in a DIMM

modules and one kernel patch. The two modules collect page table information and synchronize the information with memory trace, and the kernel patch contains a few kernel modifications.

The Toolkit provides several programs for storing trace and offline analysis. Additionally, it can also analyze process' page table information collected by KSMod (as shown in Figure 4).

5.2 HMTT parameters

Table 2 summarizes the parameters of HMTT. MTB utilize a Xilinx Virtex II Pro FPGA which works at 100MHz to support DDR 200MHz. HMTT is able to monitor more than one DIMM simultaneously, so its supported memory size can be up to 8GB. There is only one GE PHY on MTB (see Figure 5(b)), so the max trace transmission bandwidth is 1Gb/s. The full functionalities KSMod has been developed at Linux 2.6.14, and then has been ported to other versions with few efforts, such as 2.6.18. A simple KSMod for Windows has also been developed, but it can not collect page table information currently.

5.3 FPGA Functionalities

Figure 6 shows the physical block diagram of the FPGA. It contains eight logic units. The DDR Command Buffer Unit (DCBU) captures and buffers DDR commands. Then the buffered commands are forwarded to the Config Unit and the DDR State Machine Unit. The Config Unit (CU) translates a specified access

Table 2. HMTT parameters

Component	Description
FPGA	Xilinx Virtex II Pro XCVP20
Frequency	DDR 200MHz
Memory Size	<= 8GB
Output Bandwidth	1Gb/s (1 GE PHY)
Supported OS	Linux Kernel: >= 2.6, Windows*

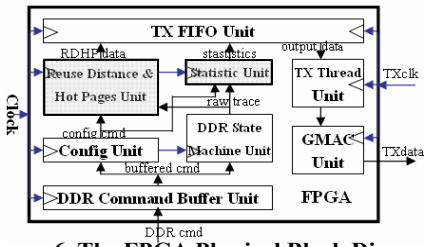


Figure 6. The FPGA Physical Block Diagram

pattern into inner-commands, and then controls MTB to perform corresponding operations, such as switching work mode, inserting synchronization tags to trace. The DDR State Machine Unit (DSMU) interprets two-phase interleaved multi-bank DDR commands to a format of <*address, r/w, duration*>. Then the trace will be inputted into the TX FIFO Unit (TFU) and be sent out via GE. The FPGA is reconfigurable to support two optional units – the Statistic Unit (SU) and Reuse Distance & Hot Pages Unit (RDHPU).

The Statistic Unit is able to do statistic of various memory events in different intervals (1us ~ 1s), such as memory bandwidth, bank behavior, and address bits change. The RDHPU is able to calculate page's reuse distance and collect hot pages. The RDHPU's kernel is a 128-length LRU stack which is implemented in an enhanced systolic array proposed by J.P. Grossman [24].

To keep up with memory speeds, the DDR State Machine Unit adopts the simplified state machine described in Section 4.2. The TX FIFO Unit contains a 16K entries FIFO between the state machine and the trace transmitting logic.

5.4 Memory Trace Board (MTB) Configuration

As mentioned in Section 4.5, the hardware monitor board (MTB) requires a configuration mechanism. Our implementation of MTB configuration supports both Linux and Windows platforms. Figure 7 shows the configuration mechanism. The configuration space of MTB is the first physical page of 0x0~0x1000 (see ①). This physical page is reserved when Linux or Windows boot. The user programs (see ②) then map "/dev/mem" (Linux) or "\\Device\\PhysicalMemory" (Windows) into their virtual address

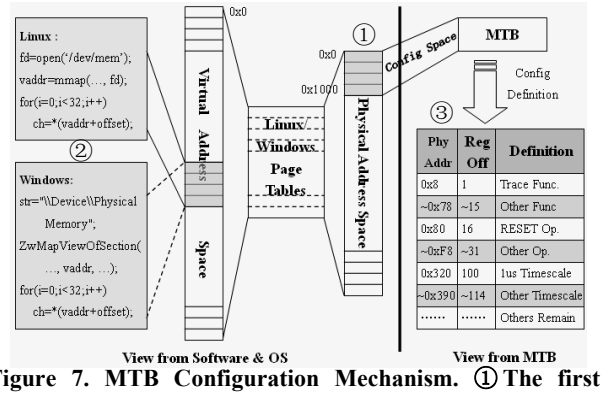


Figure 7. MTB Configuration Mechanism. ① The first physical page is reserved; ② User program maps the page into its virtual address space; ③ Well-defined commands.

spaces and accesses the page directly. The Config Unit of MTB will filter the well-defined address patterns and translate them into inner-commands in order to control MTB's work mode. We have solved the following problems to make MTB work well. First, we have defined the MTB configuration registers elaborately (see ③). Only several continuous accesses to one specified address will be translated into an inner-command. For example, the RESET inner-command is defined as 16 continuous references on the physical address of 0x80. Second, we have eliminated cache influence. The first physical page is set as uncacheable after OS startup, which will not influence system performance because the page is already reserved and only OS may access it un-frequently.

5.5 Kernel Synchronization Module (KSMod)

The KSMod is an instance of the Kernel Synchronization Mechanism (KSM) on Linux platform. It comprises two modules and one kernel patch.

One module is added to alter the first page's cache attribution. Another is a more important module which provides the *hmtt_printk* routine to collect page table information and to store the information in a kernel-user shared buffer. It is not convenient for users to find out the exact points in kernel where they should call *hmtt_printk*, although *hmtt_printk* is so flexible that it can collect any kernel information and can be called at any place from kernel. The patch has done this work.

The kernel patch is less than 30 code lines which modifies two files -- entry.S and pgtable.h (pgtable.h may be different on different CPU platforms). It modifies *set_pte_at* macro in the pgtable.h file (pgtable-2level.h in i386 platform and pgtable-3level.h in x86-64 platform) to record each update of page table. When a page fault occurs, the Linux kernel ultimately calls *set_pte_at* to update application's page table. At that time, the *hmtt_printk* will be called to collect page table trace in the format of <*pid, phy_page, virt_page, page_table_entry_addr*>.

Another patch for entry.S is optional. This patch inserts a few codes at two macros (SAVE_ALL and RESTORE_REGS) and two other points to send identifiers to MTB when kernel-enter and kernel-exit occur. The identifiers are useful for analyzing full system memory behaviors, including OS (see case study in Section 6.3). Of course, they can also be removed when only analyze user application's behavior.

The KSMod will send a synchronization tag to MTB via the configuration mechanism every 50 page faults, where the choice of number 50 has been discussed in Section 4.3.

5.6 Trace Dumping and Replay

HMTT adopts the combination of straightforward compressions, GE-RAID approaches to dump mass trace, which has been described in Section 4.4. (The trace merge procedure is not required, because HMTT has only one GE PHY currently.)

In the initial system, we used a PC with an Intel E1000 GE NIC to receive memory trace. The Toolkit of HMTT includes a zero-copy driver for the NIC. MTB sends trace data to the PC via a GE PHY. Then, the zero-copy driver receives trace data from the NIC and stores them in a shared buffer. Another program reads trace data from the shared buffer and then writes them to a RAID (Contemporary PCs can be setup in BIOS to support RAID).

The Toolkit provides several programs for trace relay and analysis. The programs can read memory trace and page table information simultaneously. They read the page table information to reconstruct physical-virtual mapping table, and extract the physical memory address to be queried in the mapping table to retrieve the process' *pid* and virtual address. When meet a synchronization tag in the memory trace, the mapping table is updated to ensure their consistency. The replayed virtual trace can be used for many analyses, such stream-based access analysis, cache/TLB performance analysis, kernel impact on memory system, application page table statistic.

5.7 Verifications

HMTT is validated in five steps:

1) As a basic verification, we have checked the physical address trace tracked by the monitoring board (MTB) with micro benchmarks which generate sequential reads, sequential writes, sequential read-after-writes and random reads in various level from cache line to page size. As shown in Figure 6, the FPGA is driven by a clock. There are two choices for clock: one is using DDR clock driven by memory controller, and another is using external clock generated by oscillators. HMTT is using an external clock currently. The test results show that there are few incorrect physical addresses, less than 1‰ (one per thousand) owing to the tiny phase shift between DDR clock and external clock. So, in the next version, we have chosen DDR clock.

2) We have used the micro benchmarks to check if virtual pages and physical pages collected by the KSMod are both linear and if they are one-to-one corresponded. This check results are perfect, and all page table information has no errors.

3) We have replayed virtual memory trace to verify the synchronization between physical memory trace and page table information. Figure 8 shows an example of quicksort's virtual memory reference trace with an input of 100,000,000 integers. Figure 8(b) shows the virtual address space and its corresponding physical address space of quicksort's data segment. The virtual address space is linear but the physical address space is discrete. Figure 8(a) shows a piece of virtual memory trace, which presents the exact reference pattern of quicksort. Moreover, the address space (0xA2800~0xA5800) also belongs to the virtual address space of data segment (0xA0000~0xC0000) (Figure 8(b)).

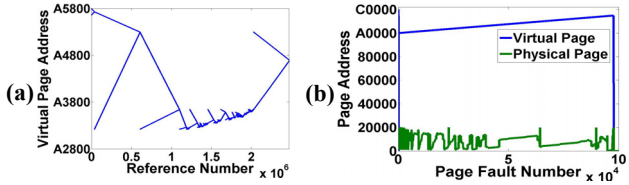


Figure 8. QuickSort_100M: (a) Virtual Memory Reference Pattern; (b) Page Table – Virtual-to-Physical mapping

4) We say a *Miss* occurs, if retrieve result is null when query a physical address in the virtual-physical mapping table. There are two reasons why "*Misses*" occur. One is the incorrect physical addresses introduced by jitters and another is some special I/O operations which are performed without page mapping. The "Miss" column in the Appendix Table shows the *Miss* proportion of various applications on an i386/Linux platform. The *Miss* proportions of most applications are less than 0.5%, except desktop applications whose vary from 1% to 2.4%. However, the *Miss* proportions are reasonable and acceptable.

5) A comparison with performance counter (use O-Profile [6] with DRAM_ACCESS event) is listed in the Appendix Table. Through the table, most differences of memory access numbers acquired by HMTT and performance counter respectively are less than 1%, mainly incurred in initialization and finalization phases.

The above verification works indicate that HMTT is a feasible and convincing memory trace monitoring system.

Table 3. Experimental Machines and Applications

	Machine 1	Machine 2
CPU	Intel Ceteron 2.0GHz	AMD Opteron 1.6GHz
L1 Cache	12K I, 6μop/line 8KB D, 4-way, 64B/line	64KB,2-way,64B/line 64KB,2-way,64B/line
L2 Cache	128KB, 2-way, 64B/line	1MB, 16-way,64B/line
Chipset	Intel 845PE	Integrated MemController
Memory	DDR 200, 512M	DDR 200, 4GB, Dual-Channels
OS	Fedora Core 4 (2.6.14)	Fedora 7 (2.6.18)
Applications	1. SPEC CPU 2000 2. SPECjbb 2005 3. OpenOffice: a 25M slide file 4. Realplayer: 10 mins video	1. SPEC CPU 2006

6. Case Studies

In this section, we will present several case studies of HMTT on two different platforms, an i386/Linux platform and a x86-64/Linux platform respectively. The case studies include: (1) memory bandwidth; (2) stream-based access analysis; (3) OS impact on memory hierarchy in real systems.

We have done experiments on two different machines as listed in Table 3. Because HMTT mainly depends on DIMM, ***it can be ported to various platforms, including multicore platforms***. We have studied memory behaviors of three classes of benchmarks including (See Table 3): computing intensive applications (SEPC CPU2006, SPEC CPU2000, both using ref input sets), OS intensive applications (OpenOffice, Realplayer), and Java Virtual Machine applications (SPECjbb 2005).

6.1 Memory Bandwidth Analysis

Limited memory bandwidth can degrade scalability of a multicore system as the number of cores increases [32]. In this study, we focus on the memory bandwidth utilization of various applications.

Usually, average memory bandwidth is adopted to evaluate an application's memory requirements. However, CPU may not generate memory reference at a stable frequency. Burst accesses can be issued in a short interval. In this study, memory bandwidths are sampled every 1ms, and burst bandwidth is defined as the 90th percentile of the bandwidth samples. Figure 9 shows all of the benchmarks' average bandwidth and burst bandwidth. Throughout Figure 9, the burst bandwidths are more than the average bandwidths, varying from 2% (171.swim) to 5X (OpenOffice) in Intel Celetron platform, as well as in AMD Opteron platform. Moreover, the highest burst bandwidths on both platforms already exceed 90% of peak bandwidths. Moreover, the average bandwidths of SPECjbb2005 from one thread to seven threads increase modestly by 1.9%, but the burst bandwidths increase by 26.3%. The burst bandwidths on AMD platform are much higher than Intel platform because AMD platform uses dual-channels.

We have replayed all samples to do a deep investigation of memory bandwidth. Figure 10 shows the memory access frequency (same meaning as bandwidth) of 255.vortex and OppenOffice in Celetron platform. Figure 10(b) shows that those high burst bandwidths of OpenOffice occur when move on to next slide. The 51 aiguilles indicate 51 slide movements. The 255.vortex in Figure 10(a) represents a typical burst characteristic of most applications. The obvious different memory bandwidth phases indicate different program behaviors. The run-time phases, observed by Sherwood et. al. [44] with SimpleScaler, are interesting and useful. Shen et. al. [43] used ATOM to generate data reference trace for locality phase prediction. HMTT is also able to analyze run-time phases.

Back to the bandwidth issue, since applications generate various behavior phases during long-time running, burst bandwidth should be regarded the same important as average bandwidth, especially in multicore systems.

6.2 Stream-Based Access Analysis

Stream-based memory accesses, also called fixed-stride access, can be used in many optimization approaches, such as prefetching and vector loads. We define "*Stream Coverage Rate (SCR)*" as the proportion of stream-based memory accesses in application's total accesses:

SCR = stream_accesses / total_access * 100%

Previous works have proposed several stream prefetchers in cache or memory controller [11, 26, 29, 38, 45]. These proposed techniques are usually based on one or two stream characteristics, such as stream stride, stream length. However, we find that at least four factors, e.g. stride, length, interval, and active stream number, should be considered when a new prefetcher is to be proposed, because the four factors can influence prefetch effect. In this study, we have used HMTT to reveal the four factors as well as the *SCRs* of various applications in a real system (Intel Celetron Platform).

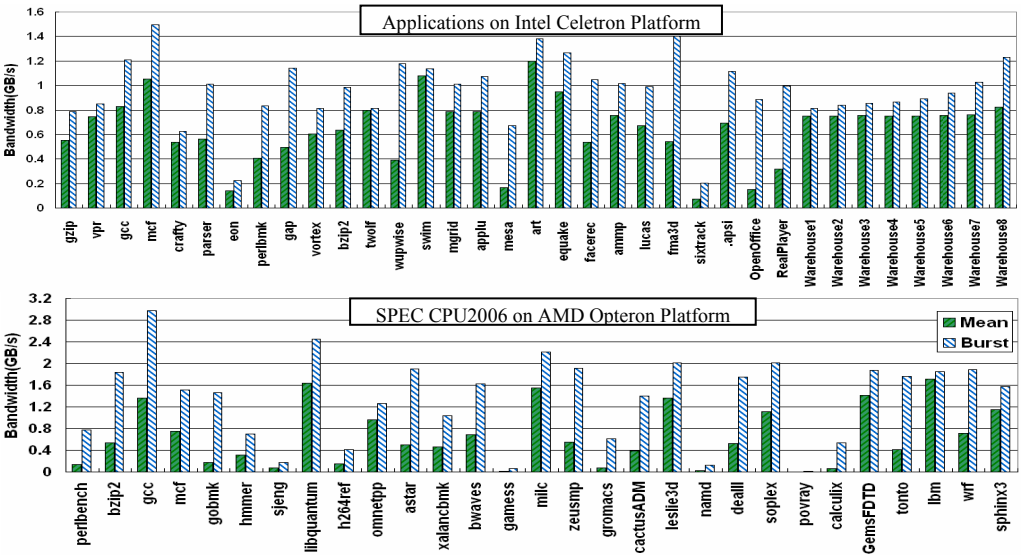
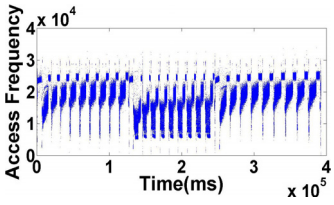
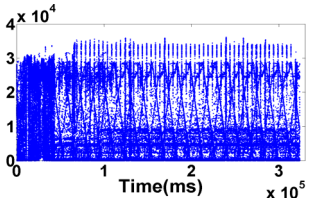


Figure 9. Mean Bandwidth V.S. Burst Bandwidth

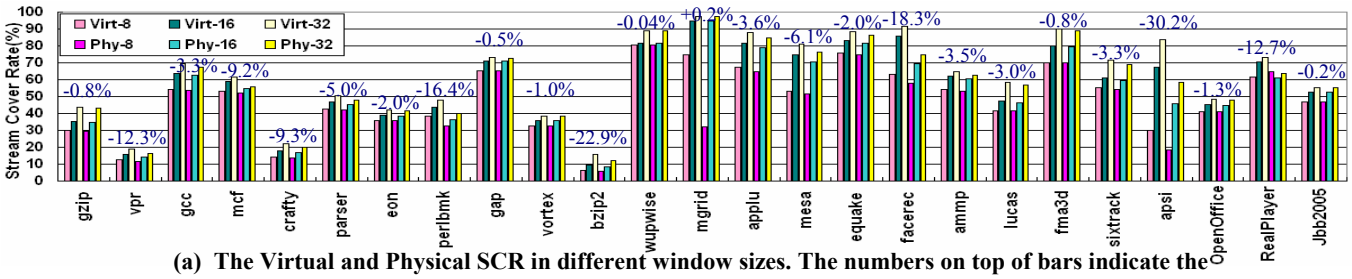


(a) 255.vortex



(b) OpenOffice

Figure 10. Memory Reference Phase



(a) The Virtual and Physical SCR in different window sizes. The numbers on top of bars indicate the decreased proportions between virtual SCRs and physical SCRs due to OS virtual-to-physical mapping.

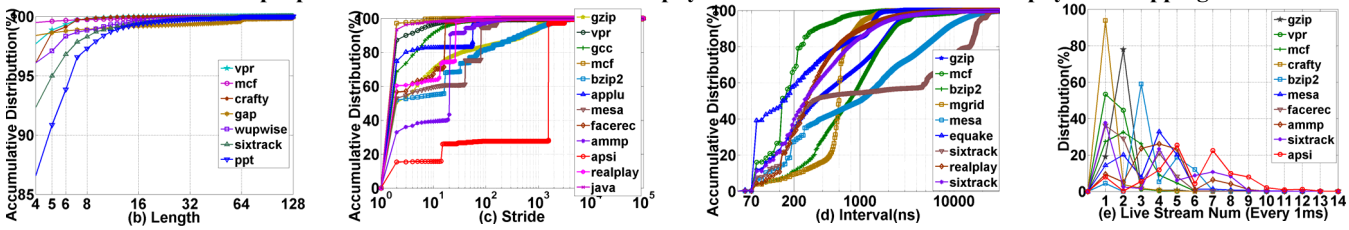


Figure 11. Stream Statistic: (a) SCR; (b) Length Accumulative Distribution; (c) Stride Accumulative Distribution; (d) Interval Accumulative Distribution; (e) Active Stream Distribution;

We adopte an algorithm proposed by Tushar Mohan et. al. [35] to detect stream in cache line level. The algorithm is simple and efficient. We add a *stream_age* attribute, which is calculated as *current_time – stream_last_update_time*. If one stream's *stream_age* is greater than a threshold, it is considered as inactive and can be removed from stream table. The overhead of prefetch hardware mainly depends on stream table capacity which only stores active streams. The choice of the threshold is a tradeoff. Our studies shows that the stream characteristics are almost same when use a static threshold of *one second* and a dynamic thread of *5 * stream_ mean_interval* respectively. Thus, we choose the latter threshold for detecting active streams. Figure 11(a) shows the physical and virtual *SCRs* detected with different scan-window sizes. As shown in Figure 11(a), most applications can achieve *SCRs* of more than 40% under a 32-size window (The following studies are under the 32-size window). The numbers on top of bars indicate the *SCRs* proportion reduced by OS page mapping, from 0.04% (wupwise) to 30.2% (apsi). As shown in Figure 8(b), although the linear virtual page may become un-linear physical pages after page mapping in OS kernel.

We investigated into the four factors of all streams. Figures 11(b) ~ (e) present the stream statistical characteristics of some benchmarks. Figure 11(b) shows the stream length distribution. It is observed that the lengths of most streams are less than 10. Moreover, more than 85% streams are only 4-length. Thus, the stream lengths are so short that traditional prefetching techniques may increase prefetch bandwidth significantly.

Figure 11(c) shows that most streams' strides are also less than 10, within one page. The short strides indicate that most streams have good spatial locality which can be explored with the efforts of cache and memory controller. The results also indicate that OS page mapping may influence the *SCRs* slightly when streams are within one physical page. Nevertheless, the 301.apsi represents another typical class of applications whose SCRs reduce more than 10% from virtual to physical stream detection. We find that most strides are quite large. For example, the stride of 301.apsi is mainly more than 64B*1000 ≈ 64KB, covering several pages.

Figure 11(d) shows that about 90% of average intervals fall between 100ns ~ 10us which are multiple times of one memory access latency. Therefore, hardware has enough time to perform prefetching even in distributed systems. Moreover, Figure 11(e) shows that the number of active streams is less than 10 at most time, so hardware stream table capacity can be small.

Overall, the results show that the *SCRs* of most applications are more than 40%, streams have good spatial locality, the stream interval is appropriate for prefetch, and active streams at a time is few. Thus, prefetching in cache and memory controller is reasonable. Nevertheless, the stream length is so short that prefetchers should be more intelligent to avoid high additional bandwidth.

6.3 OS Impact on memory hierarchy in real systems

Cache and TLB are two of the most important topics in micro architecture fields. Using the full system simulator SimOS [41], Barroso [13], Redstone [40], Rosenblum [42] et. al. have studied the OS impact on cache/TLB performance. The OS impact to buffer cache in main memory has also studied (e.g. [21] and [28]). With the cache/TLB miss reference trace provided by HMTT, we have evaluated the impact of OS in a real i386/Linux system.

We believe that the kernel impact should be divided into two parts: (1) the Cache/TLB misses in the kernel mode; (2) user programs' refilling cache line and TLB entries evicted by kernel data after kernel exiting. We use *"In Kernel (IK)"* and *"Kernel Exit (KE)"* to identify them. With the kernel-enter/kernel-exit identifiers and page table information, we can figure out the OS impact on Cache/TLB performance.

The Appendix Table lists the Cache/TLB performance of all benchmarks. Throughout the table, we can see that the *IK* cache misses in most SPEC CPU2000 benchmarks are less than 10%, with values of around 1~6%. However, *IK* cache misses in SPECjbb 2005 and Realplayer are about 11%, and it is near 30% in OpenOffice. *KE* cache misses are distributed undeterminedly, from 1% to 75% of *IK* misses. The sum of *IK* and *KE* cache misses accounts for quite a proportion of total cache misses.

In X86 CPUs, MMU will perform page table walk on a TLB miss, then incur cache miss. The Appendix Table shows that the cache misses caused by TLB misses are distributed from 0.2% to 13%. Compared to other benchmarks, 255.vortex and 301.apsi are more than 10%. Thirteen benchmark's *IK* TLB misses account for more than 10%, and Realplayer is more than 45%. All benchmarks' *KE* TLB misses are more than 20% of the *IK*, and

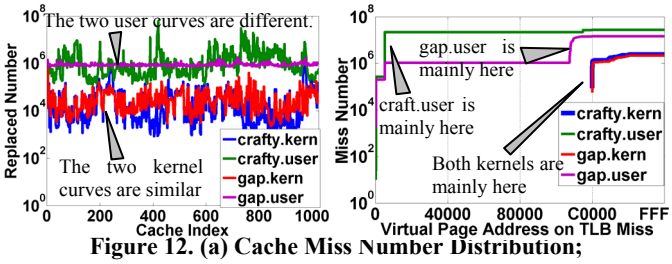


Figure 12. (a) Cache Miss Number Distribution; (b) TLB Miss Number Accumulative Distribution

most fall within 40% ~ 60%. From these statistics, we can summarize that TLB is more kernel-sensitive than cache.

We find that the distributions of cache miss number in cache indexes are absolutely different in programs' user address space (crafty.user and gap.user, the upper two curves in Figure 12(a)). But it is so interesting that their kernel distributions are very similar (crafty.kern and gap.kern in Figure 12(a)). The TLB misses have the same phenomenon. In Figure 12(b), the TLB miss in kernel space distributions are fall within the same regions, but they are different in the user space. Thus, user programs and OS kernel will disturb each other at each kernel-user mode switch.

Li et. al. [30] also addressed the user/OS branch history interference problem. Multicore has already become popular, and the trend of manycore is coming. Since computing resources are no longer inadequate, asymmetric kernels and dedicated OS cores may alleviate kernel impact and improve an application's performance in the forthcoming manycore era [20].

7. Evaluations and Discussions

7.1 Trace Generation Bandwidth

The two "NBW" columns in the Appendix Table list trace generation bandwidth on Intel and AMD platforms, respectively. When utilize DDR-200MHz memory. The bandwidth varies from 5.7MB/s (45.6Mbps) to 72.9MB/s (583.2Mbps) on Intel platform, and from 0.1MB/s (0.8Mbps) to 106.8MB/s (854.4Mbps) on AMD platform. This indicates that a bandwidth of 1000Mbps is sufficient for HMTT to capture all applications' traces on Intel platform and most applications' traces on AMD platform.

However, the high frequency of DDR2/DDR3 memory and the prevalent multi-channel memory technology increase trace data generation bandwidth. The next version of monitoring system will support trace generation bandwidth of at least 2Gbps.

7.2 Overhead

The overheads of HMTT include trace size, kernel buffer for collection of page table information, synchronization latency, additional execution time and additional memory accesses.

The traces are generated on two experimental machines whose parameters are list in Table 3. The "NUM" columns and the "Size" columns in the Appendix Table list the total number and size of reference trace of all applications. There are billions of traces and most trace sizes are more than 10GB. The trace size is quite large, and large capacity disks are demanded. Fortunately, the disks are becoming cheaper and cheaper.

The capacity of kernel buffer for page table information collection is less than 0.5% of total memory size of traced system; because most physical pages are mapped to virtual pages only once during application's entire execution lifetime (see Figure 4).

Figure 7 describes the configuration mechanism of HMTT and shows a small piece of control codes to configure and synchronize HMTT. The codes are run many times on the experimental machines (see Table 3), and the average execution time of control codes is only 5.9us. Moreover, the application's execution time is increased by about 1% when HMTT enables all optional synchronization tags, such as kernel_enter/kernel_exit identifiers. The additional memory access (about 1%) arisen by control codes will not influence the memory reference trace because HMTT filters these accesses as configurations.

7.3 Limitations

It is important to note that the monitoring mechanism can not distinguish the prefetch commands.

Regarding the impact of prefetch on memory trace, it has both up side and down side. The up side is that we can get real memory accesses trace to main memory, which can benefit research on main memory side (such as memory thermal model research [31]). The down side is that it is hard to differentiate the prefetch memory accesses and on-demand memory accesses. Regarding prefetch, caches could generate speculative operations. However, they do not influence memory behaviors significantly. Most memory controllers do not have prefetch unit, although several related efforts have been made, such as Impluse project [53], proposed region prefetcher [48], and the stream prefetcher in memory controller [26]. Thus, it is not a critical weakness of our monitoring system. It is to be noted that all hardware monitors also have the same limitation, prefetching from various levels of the memory hierarchy. In fact, caches can be disabled to eliminate caches' influence. However, the execution time dilation would be 10X~100X, and the trace size would be magnified.

7.4 Discussions

As a new tool, HMTT is a complementary tool to binary instrumentation and full system simulation with software, rather than a thorough substitution. Since it is running in real-time and in a real system, the combination with different techniques would be more efficient for architecture and application research.

Combination with simulators: to combine with simulators, HMTT is used to collect trace from real system, include multicore systems. Then, the trace is analyzed for finding new insights. Some new optimization mechanisms proposed basing on new insights can be evaluated by simulators.

Combination with binary instrumentation: HMTT provides control codes to send synchronization tags into memory trace. So, instrumentation tools can instrument these control codes into application binary as annotations to indicate memory references designated functions/loops/blocks. Moreover, with compiler-provided symbol table, the virtual-address trace can be utilized for semantic analysis. As shown in Figure 10, besides tracking trace, HMTT is also able to analyze run-time phases. Binary instrument tools can be adopted to insert function/loop/block indicators into phase's graphs as well.

8. Conclusion and Future Work

In this paper, we have proposed a platform independent full-system memory trace monitoring system, which is able to track virtual memory reference trace of the full systems. It adopts the DIMM monitoring mechanism, a simplified state machine to keep up with memory speeds, a Kernel Synchronization Mechanism to reconstruct virtual-physical mapping, and a GE-RAID approach to dump full mass trace. We present our implementation and several case studies of an initial monitoring system, named HMTT, to show the feasibility of our proposed monitoring mechanism and techniques.

The current HMTT version 1.0 is the first stage of the monitoring system. At the next stage, we have been building an enhanced monitoring system to monitor multiple DDR2 DIMMs in support of mainstream servers with more than 8GB memory.

Acknowledgments

We would like to thank Prof. Kai Li (Princeton University) for his constructive and valuable suggestions on this study. Moreover, he encouraged us throughout this study and even gave us detailed suggestions on paper organization. We give special thanks to Prof. Xiaodong Zhang (Ohio State University) for shepherding the final version of this paper with detailed and valuable suggestions. His Dragon Star Lecture on "Caching and Buffer Management" held at the Institute of Computing Technology (ICT) in 2005 gave us insights into memory systems and motivated us to work in several memory systems research projects. We thank Jiang Lin (Iowa State University), Guangmin Tan, Wengli Zhang, Peiheng Zhang, Zeng Cao, Lei Li and other teammates from ASL, ICT for their help in discussions, experimental setup and paper writing. We also thank Xia Cao (University of Michigan) and our anonymous reviewers for their insightful suggestions. This research is supported by the National Natural Science Foundation of China (NSFC) under a grant number 60633040 and the National High Technology Research and Development Program of China (863 Program) under grant numbers 2006AA01A102, 2007AA01Z115.

References

[1] DyninstAPI. http://www.dyninst.org/.
[2] Intel Corp. Intel Itanium2 Processor-Reference Manual, 2004.
[3] JEDEC SOLID STATE TECHNOLOGY ASSOCIATION. Double Data Rate (DDR) SDRAM Specification. Jan 2004.
[4] Linux Trace Toolkit Next Generation. http://ltt.polymtl.ca/.
[5] M5. http://m5.eecs.umich.edu/.
[6] O-Profile. http://oprofile.sourceforge.net/.
[7] PIN. http://rogue.colorado.edu/pin/.
[8] RAMP. http://ramp.eecs.berkeley.edu/.
[9] SimpleScalar 3.0. http://www.simplescalar.com/.
[10] Valgrind. http://valgrind.org/.
[11] J. L. Baer and T. F. Chen. Effective hardware-based data prefetching for high-performance processors. *IEEE Transactions on Computers*, 44(5):609–623, 1995.
[12] Paul Barham, Boris Dragovic, Keir Fraser, Steven Hand, Tim Harris, Alex Ho, Rolf Neugebauer, Ian Pratt, Andrew Warfield. Xen and the Art of Virtualization. *SOSP*, 2003.
[13] L. A. Barroso, K. Gharachorloo, E. Bugnion. Memory System Characterization of Commercial Workloads. *ISCA*, June 1998.
[14] L. A. Barroso, S. Iman, J. Jeong, K. Oner, K. Ramamurthy and M. Dubois. RPM: A Rapid Prototyping Engine for Multiprocessor Systems. *IEEE Computer*, February 1995.
[15] L. A. Barroso. Design and Evaluation of Architectures for Commercial Applications. *Technique Reprot*, 1999.
[16] Fabrice Bellard. QEMU, a Fast and Portable Dynamic Translator. *Usenix Annual Technical Conference*, April 2005.
[17] Sanjay Bhansali, Wen-Ke Chen, Stuart de Jong, Andrew Edwards, Ron Murray, M. Drinic, Darek Mihocka, Joe Chau. Framework for Instruction-level Tracing and Analysis of Program Executions. *Proceedings of the 2nd International conference on Virtual execution environments (VEE)*, 2006.
[18] Prashanth P. Bungale and Chi-Keung Luk. PinOS: A Programmable Framework for Whole-System Dynamic Instrumentation. *the 3rd international conference on Virtual execution environments (VEE)*, June 2007.
[19] Nirut Chalainanont, Eriko Nurvitadhi, Roger Morrison, Lixin Su, Kingsum Chow, Shih-Lien Lu, and Konrad Lai. Real-time L3 Cache Simulations Using the Programmable Hardware-Assisted Cache Emulator (PHA$E). *IEEE 6th Annual Workshop on Workload Characterization*, 2003.
[20] K. Chakraborty, P. M. Wells, G. S. Sohi. Computation Spreading: Employing Hardware Migration to Specialize CMP Cores On-the-fly. *ASPLOS*, 2006.
[21] Xiaoning Ding, Song Jiang, Feng Chen, Kei Davis, and Xiaodong Zhang. DiskSeen: exploiting disk layout and access history to enhance I/O prefetch. *Proceedings of the 2007 USENIX Annual Technical Conference*, 2007.
[22] J. K. Flanagan, B. E. Nelson, J. K Archibald, K. S. Grimsrud. BACH: BYU Address Collection Hardware, The Collection of Complete Traces. *Computer Performance Evaluation '92: Modeling Techniques and Tools*, August 1993.
[23] K. Grimsrud, J. Archibald, M. Ripley, K. Flanagan, and B. Nelson. BACH: A Hardware Monitor for Tracing Microprocessor-based Systems. *Microprocessors and Microsystems*, v.17, n.8, October 1993, pp. 443-458.
[24] J.P. Grossman. A Systolic Array for Implementing LRU Replacement. *Technical Memo*, MIT, 2002.
[25] Jumnit Hong, Eriko Nurvitadhi, Shih-Lien L. Lu. Design, implementation, and verification of active cache emulator (ACE). *International Symposium on Field Programmable Gate Arrays (FPGA)*, 2006.
[26] Ibrahim Hur, Calvin Lin. Memory Prefetching Using Adaptive Stream Detection. *Micro*, 2006.
[27] J. Iyer, C. L. Hall, J. Shi, Y. Huang. System memory power and thermal management in platforms built on Intel® Centrino® Duo mobile technology. *Intel Technology Journal*.
[28] Song Jiang, Xiaoning Ding, Feng Chen, Enhua Tan, and Xiaodong Zhang. DULO: an effective buffer cache management scheme to exploit both temporal and spatial localities. *FAST*, 2005.
[29] N. P. Jouppi. Improving direct-mapped cache performance by the addition of a small fully-associative cache and prefetch buffers. *ISCA*, 1990.
[30] T. Li, Lizy K. John, A. Sivasubramaniam, N. Vijaykrishnan, J. Rubio. Understanding and improving operating system effects in control flow prediction. *ASPLOS*, 2002.
[31] Jiang Lin, Hongzhong Zheng, Zhichu Zhu, Howard David and Zhao Zhang. Thermal Modeling and Management of DRAM Memory Systems. *ISCA*, June, 2007.
[32] Jiang Lin, Qingda Lu, Xiaoning Ding, Zhao Zhang, Xiaodong Zhang, and P. Sadayappan. Gaining insights into multicore cache partitioning: bridging the gap between simulation and real systems. *HPCA*, 2008.
[33] ShihLien L. Lu, Peter Yiannacouras,Taeweon Suh, Rolf Kassa, Michael Konow. An FPGA-based Pentium® in a complete desktop system. *FPGA*, 2007.
[34] Peter S. Magnusson, Magnus Christensson, Jesper Eskilson, Daniel Forsgren, Gustav Hållberg, Johan Högberg, Fredrik Larsson, Andreas Moestedt, Bengt Werner. Simics: A Full System Simulation Platform. *IEEE Computer*, Feb 2002.
[35] Tushar Mohan, Bronis R. de Supinski, Sally A. McKee, Frank Mueller, Andy Yoo, Martin Schulz. Identifying and Exploiting Spatial Regularity in Data Memory References. *Supercomputing Conference*, November 2003.

[36] Ashwini K. Nanda, Kwok-Ken Mak, Krishnan Sugavanam, R. K. Sahoo, V. Soundararajan, T. Basil Smith. MemorIES: A Programmable, Real-Time Hardware Emulation Tool for Multiprocessor Server Design. *ASPLOS*, 2000.

[37] Gil Neiger, Amy Santoni, Felix Leung, Dion Rodgers, Rich Uhlig. Intel Virtualization Technology: Hardware Support for Efficient Processor Virtualization. *Intel Technology Journal*, vol. 10, August 2006.

[38] S. Palacharla and R. E. Kessler. Evaluating stream buffers as a secondary cache replacement. *ISCA*, 1994.

[39] K. Rajamani. Memsim users' guide. *IBM research report. Technical Report RC23431*, October 2004.

[40] J. A. Redstone, S. J. Eggers and H. M. Levy. An Analysis of Operating System Behavior on a Simultaneous Multithreaded Architecture. *ASPLOS*, November 2000

[41] M. Rosenblum, S. A. Herrod, E. Witchel, A. Gupta. Complete Computer System Simulation: the SimOS Approach. *IEEE Parallel and Distributed Technology: Systems and Applications*, 1995.

[42] Mendel Rosenblum, Edouard Bugnion, Stephen Alan Herrod, Emmet Witchel, Anoop Gupta. The Impact of Architectural Trends on Operating System Performance. *SOSP*, 1995.

[43] Xipeng Shen, Yutao Zhong, and Chen Ding. Locality Phase Prediction. *ASPLOS*, 2004.

[44] Timothy Sherwood, Suleyman Sair, Brad Calder. Phase Tracking and Prediction. *ISCA*, June 2003.

[45] A. Smith. Sequential program prefetching in memory hierarchies. *IEEE Transactions on Computers*, 1978.

[46] Amitabh Srivastava, Alan Eustace. ATOM: a system for building customized program analysis tools. *PLDI*, 1994.

[47] David Wang, Brinda Ganesh, Nuengwong Tuaycharoen, Katie Baynes, Aamer Jaleel, and Bruce Jacob. DRAMsim: A Memory System Simulator. *SIGARCH Computer Architecture News*, vol. 33, no. 4, September 2005.

[48] Z. Wang, D. Burger, K. S. McKinley, S. K. Reinhardt, and C. C. Weems. Guided Region Prefetching: A Cooperative Hardware/Software Approach. *ISCA*, 2003.

[49] M. Watson and J. Flanagan. Simulating L3 Caches in Real Time Using Hardware Accelerated Cache Simulation (HACS): a Case Study with SPECint 2000. *Symposium on Computer Architecture and High Performance Computing*, 2002.

[50] Win. A. Wulf, Sally A. McKee. Hitting the Memory Wall: Implications of the Obvious. *Computer Architecture News*, 23(1):20-24, March 1995.

[51] Hyung-Min Youn, Gi-Ho Park, Kil-Whan Lee, Tack-Don Han, Shin-Dug Kim, and Sung-Bong Yang. Reconfigurable Address Collector and Flying Cache Simulator. *High Performance Computing Asia*, April 1997.

[52] R. A.Uhilg, T. N. Mudge. Trace-driven memory simulation: A survey. *ACM Computing Surveys*, Jun., 1997.

[53] Lixin Zhang, Zhen Fang, Mide Parker, Binu K. Mathew, Lambert Schaelicke, John B. Carter, Wilson C. Hsieh, Sally A. McKee. The Impulse Memory Controller. *IEEE Transactions on Computers*, pp. 1117 – 1132, 2001.

Appendix Tables. Memory Reference Trace and Cache/TLB Statistics

Benchmarks		HMTT Trace		Perf.Cnt	HMTT Perf.Cnt Diff (%)
		NUM (B)	NBW (MB/s)	DRAM Access(B)	
SPEC CPU 2006 INT	400.perlbench	2.57	8.27	2.56	0.07
	401.bzip2	14.70	33.60	14.61	0.67
	403.gcc	35.13	85.48	34.61	1.51
	429.mcf	28.38	46.71	28.25	0.46
	445.gobmk	3.27	11.00	3.23	1.26
	456.hmmer	7.82	19.29	7.83	-0.23
	458.sjeng	1.91	4.84	1.91	0.02
	462.libquantum	77.46	102.04	76.83	0.82
	464.h264ref	5.30	9.02	5.32	-0.25
	461.omnetpp	19.57	60.14	19.50	0.38
	473.astar	11.98	31.16	11.81	1.40
	483.xalancbmk	14.54	29.16	14.51	0.21
SPEC CPU 2006 FP	410.bwaves	41.83	42.98	41.52	0.74
	416.gamess	0.30	0.50	0.31	-0.95
	433.milc	38.97	96.54	38.68	0.75
	434.zeusmp	14.72	34.14	14.64	0.54
	435.gromacs	1.78	4.92	1.78	-0.23
	436.cactusADM	18.16	24.36	18.00	0.89
	437.leslie3d	43.39	84.84	42.97	0.97
	444.namd	0.35	1.19	0.36	-0.49
	447.dealII	12.94	32.80	12.94	0.03
	450.soplex	29.63	69.39	29.42	0.74
	453.povray	0.02	0.10	0.02	0.75
	454.calculix	3.95	3.67	3.95	-0.01
	459.GemsFDTD	49.94	87.94	49.51	0.87
	465.tonto	11.98	26.06	11.92	0.52
	470.lbm	68.64	106.82	67.66	1.44
	481.wrf	26.68	44.80	26.58	0.35
	482.sphinx3	49.64	71.78	49.63	0.03

Benchmarks		Reference Trace Info				Cache Miss Statistics(%)				TLB Miss Statistics(%)				
		NUM (B)	Size (GB)	NBW (MB/s)	Miss (%)	APP	IK	KE /IK	IK +KE	TLB /NUM	APP	IK	KE /IK	IK +KE
SPEC CPU 2000 INT	164.gzip	2.8	13.7	33.9	0.13	97.2	2.1	37.8	2.9	0.4	57.9	34.8	51.0	52.5
	165.vpr	5.9	25.7	44.9	0.18	96.0	3.7	19.0	4.4	2.6	96.0	3.3	50.1	5.0
	176.gcc	3.0	15	44.5	0.29	96.3	3.0	38.4	4.1	1.2	78.6	12.0	43.6	17.3
	181.mcf	9.4	40	63.4	0.11	92.9	7.0	28.4	9.0	6.3	99.0	0.9	47.0	1.3
	186.crafty	2.1	8.1	27.3	0.13	96.6	2.9	57.4	4.6	1.5	90.4	7.7	55.5	12.0
	197.parser	4.1	18.9	36.1	0.23	95.5	4.2	34.4	5.6	2.7	94.5	4.5	49.4	6.8
	252.eon	3.0	13.3	8.7	0.20	96.5	2.9	57.0	4.5	0.6	74.2	23.4	56.4	36.6
	253.perlbmk	1.9	8	24	0.33	84.9	11.0	35.3	14.9	7.9	94.9	4.5	44.2	6.4
	254.gap	1.0	5.4	29.3	0.39	94.8	4.1	21.7	5.0	1.8	84.3	12.7	36.3	17.3
	255.vortex	3.4	12.7	32.8	0.51	86.1	11.9	38.0	16.4	10.4	98.3	1.5	43.8	2.2
	256.bzip2	3.8	16.1	36.7	0.08	97.3	2.3	14.6	2.7	1.2	90.7	7.9	48.5	11.7
	300.twolf	10.9	50.3	48.7	0.13	98.1	1.5	21.6	1.8	0.5	78.8	18.3	63.9	30.0
SPEC CPU 2000 FP	168.wupwise	1.6	7.4	24.6	0.15	97.0	2.3	6.3	2.5	0.5	62.4	31.5	44.8	45.6
	171.swim	14.5	63.2	65.8	0.1	94.1	5.8	75.2	10.1	5.1	98.8	1.0	48.6	1.5
	172.mgrid	5.6	26.4	48	0.06	98.3	1.4	1.8	1.4	0.4	78.4	18.3	56.0	28.6
	173.applu	4.5	24.2	47.9	0.08	97.4	2.4	30.6	3.1	1.3	91.7	7.1	53.2	10.8
	177.mesa	0.5	3.4	11.2	0.77	88.4	8.1	35.5	10.9	2.0	72.9	24.7	51.3	37.4
	179.art	28.6	128.9	72.9	0.04	99.1	0.8	19.3	1.0	0.3	83.9	13.1	54.0	20.1
	183.equake	3.1	13.2	58.3	0.05	98.2	1.5	2.7	1.5	0.7	89.3	8.8	54.9	13.7
	187.facerec	5.0	24.3	34.4	0.19	95.9	3.6	7.7	3.9	1.1	66.3	30.0	26.3	38.0
	188.ammp	8.9	39.2	46.4	0.11	95.4	4.4	35.5	5.9	3.5	87.2	2.1	42.2	3.1
	189.lucas	3.6	19.2	41.7	0.12	93.3	6.4	1.1	6.5	5.2	97.7	2.0	44.3	2.9
	191.fma3d	3.3	17.7	33.8	0.13	97.5	2.1	15.2	2.4	0.5	72.4	26.1	58.6	41.4
	200.sixtrack	0.4	1.8	5.7	0.19	83.6	12.8	11.5	14.3	2.2	54.4	39.0	48.6	58.0
	301.apsi	9.4	40.7	44.4	0.12	85.8	14.0	33.1	18.6	13.2	99.4	0.5	41.1	0.7
Desktop	OpenOffice	0.7	3.3	10.5	2.40	66.7	29.3	14.5	33.6	7.2	60.7	34.9	24.8	43.1
	ReadPlayer	2.5	13	22.2	1.2	83.9	12.3	8.2	13.3	3.0	50.0	45.2	30.3	59.0
Java	SPECjbb2005	17.2	77.5	41.3	1.01	87.3	11.5	28.1	14.7	7.1	94.0	5.2	49.1	7.7

Notes: **NUM** = Total Memory Reference Trace Number (in Billion); **Size** = Total Trace Size; **NBW** = GE Bandwidth; **PerfCnt** = Performance Counter; **Miss** = Physical address missed in virtual-physical mapping; **APP** = Applications Proportion; **IK** = "*In Kernel*" Proportion; **KE** = "*Kernel Exit*" Proportion; **Diff** = Difference between HMTT and PerfCnt, calculated as *(HMTT.NUM-PerfCnt)/HMTT*100%*; **TLB/NUM** = The ratio of Cache Miss caused by TLB Miss to total trace number; **KE/IK** = The ratio of "*Kernel Exit*" proportion to "*In Kernel*" proportion; **IK+KE** = The sum of "*In Kernel*" proportion and "*Kernel Exit*" proportion, equal to *IK*(1+KE/IK)*.