An Overview of Memory Virtualization Techniques Based on Intel VT *

Manuela K. Ferreira, Henrique C. Freitas, Philippe O. A. Navaux
Informatics Institute, Universidade Federal do Rio Grande do Sul, Brazil
{mkferreira, hcfreitas, navaux}@inf.ufrgs.br

Abstract

Once confined to specialized server and mainframe systems, virtualization is now becoming more broadly available. This is due to processing improvements and the benefits provides by virtualization like concurrent execution of multiple operating systems (OSs) with total isolation of them. Intel VT is a hardware support proposal with the goal of simplify the Virtual Machine Monitor (VMM) and increase the virtualized systems performance. This technology includes definition of new data structures, new forms of CPU operation, a new instruction set architecture (ISA), and new memory techniques. So, the goal of this paper is to present the memory techniques used by Intel VT architecture to manage access to the memory of virtual machines (VMs).

1. Introduction

Virtualizing physical resources of mainframe systems to achieve improved sharing and utilization is well-established concept since 1960's, when IBM began developing the first VM that allowed one computer to be shared as if it was several [8].

The virtualization offers benefits like concurrent execution of different operation systems (OSs) with dynamic load balancing that increases the utilization of the system resources. Users can isolate untrusted applications of unknown quality. The isolation improves system security and prevent software failures in one VM to effect the other VMs. It allows a system manager to configure the environment in which guest OSs will run. This provides the ability to run legacy OS and a developer can easily test application on a virtual environment. These features can be used to construct system software for scalable computers that have anywhere from 10 to 100 processors. Through physical resources abstraction 10 real processors can appears to application software as 1; or 1 processor can appears as 10 [7].

Virtualization is the capacity to run multiple OS at same time on a single physical plataform sharing the system resources. A virtualized system includes a new layer of software, the virtual machine monitor (VMM). The VMM's principal role is to arbitrate accesses to the underlying physical host plataform's resources so the multiple OSs (which are guests of the VMM) can share them. The VMM presents to each guest OS a set of virtual plataform interfaces that constitute a virtual machine (VM).

Once confined to specialized server and mainframe systems, virtualization is now becoming more broadly available due to performance processing improvement. Then more effort is spent to improve the virtualized systems performance. To achieve this goal there are software and, recently, hardware proposals [1].

Intel VT is the Intel Virtualization Technology for x86 processors that provides hardware virtualization support with the goal of simplify the VMMs in order to increase the virtualized systems performance [5]. This technology includes definition of new data structures, new forms of CPU operation, a new ISA, and new memory techniques.

With virtualization there is one level more of address to be translated. In this paper will be presented the memory techniques used by Intel VT architectures to manage the memory and accesses from virtual machines (VMs). These techniques will provide basement to extend the model presented in [4].

First, the memory virtualization techniques used in Intel VT technology are detailed. After, a research proposal is presented in order to extend the MIPS-vt model. Then the ArchC framework, necessary to extend the model, is briefly presented. Finally the conclusions and future works.

2. Intel VT Memory Virtualization

This section presents the memory virtualization techniques used in Intel VT technology. How a guest physical addres is translated to host physical address is detailed.

* Supported by Microsoft Company.

In architectures without explicit support for MMU virtualization the hidden page faults, caused by misses in the shadow page table, cause the switch to the VMM to construct an appropriate page table entry. This incur significant overhead.

In the memory virtualization scheme of Intel VT, the VMM maintains a hardware-consulted nested page table that translates guest physical addresses to host physical addresses. This mapping allows the hardware to dynamically handle guest MMU operations, eliminating the need for VMM interposition. The operation of this scheme is illustrated in Figure 1. While running in hardware-assisted guest execution, the Translation Look-aside Buffer (TLB) contains entries mapping guest virtual addresses to host physical addresses. The process of filling the TLB in case of miss is described foward. Considering the case of a guest reference to virtual address V that missis in the hardware TLB:

1. The hardware uses the guest page table pointer (`%cr3` to locate the top level of the guest's hierarchical page table.
2. `%cr3` contains a guest physical address, which must be translated to a host physical address before derreferencing. The hardware consults the nested page table for the guest's `%cr3` value to obtain the host physical pointer to the top level of the guest's page tabel hierarchy.
3. The hardware reads the guest page directory entry (PDE) correnponding to guest virtual address V.
4. The PDE read in step 3 also yields a guest physical address which must also be translated via the nested page table before proceeding.
5. Having discovered the host physical address of the final level of the guest page table hierarchy, the hardware reads the guest page table entry (PTE) corresponding to V. In our example, this PTE points to guest physical address A, which is translated via a third consult of the nested page table to host physical address B.
6. The translation is complete: virtual address V maps to host physical address B. The page consult hardware can now fill the TLB with an apropriate entry (V,B) and resme guest execution, all without software intervation.

For an *M*-level guest page table on an *N*-level nested page table, a worst-case TLB miss requires *MN* memory accesses to satisfy. Nested paging holds the promise of allowing guest context switches without VMM intervation. By resolving the most important source of overhead in current VMM [1], nested paging hardware should easily repay the cost of slower TLB misses.

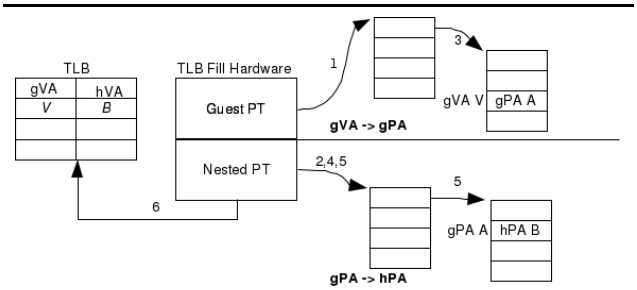

Figure 1. Nested paging hardware [1]. guest virtual address (gVA), guest physical address (gPA), host physical address (hPA).

Intel VT also has Virtual Processor Identifiers (VPID). This feature allows a VMM to assign a different non-zero VPID to each virtual processor (the zero VPID is reserved for VMM). The CPU can use VPIDs to tag translations in the TLBs. This feature eliminates the need for TLB flushes on every each context switching between VMs and VMM and eliminates a significant overhead of those flushes on performance [5].

3. Research Proposal using ArchC

In this section the proposal to model the memory virtualization from Intel VT is described. Then is overviewed the ArchC framework and some ArchC works.

In [4] and [3] a MIPS model increased with instructions and structures of Intel VT-x, the MIPS-vt model, is presented. The proposal is to extend the MIPS-vt model with memory hierarchy, modeling the nested page table and the VPIDs from Intel VT(Section 2) using ArchC.

3.1. ArchC Tool

ArchC is an architecture description language (ADL) that can automatically generate simulators using the SystemC (hardware description language) and it is capable of describing processor and ISA design and memory subsystem. Memory hierarchies can be declared, containing several levels of memories and caches. Caches can be configured to simulate different set associativities, write polices, replacement strategies, and line size [6].

SystemC is a class set in C++ that extends the language to allow hardware and system modeling. Although SystemC supports several computing models, communication and abstraction levels, is dificult to extract from processors description in SystemC all the necessary information to designers experiment and evaluate a new ISA.[9].

The goal of ArchC is to improve the abstraction for SystemC models. ArchC provides enough information, at an appropriate abstraction level, to allow designers explore

and check a new architecture by software framework automatic generation like assemblers, simulators, deployment and communication interface [9].

A processor architecture description in ArchC is divided in to two parts: i) the instruction set architecture description in ArchC (AC_ISA) provides details about instruction formats, size and names combined with all information necessary to decoding and the behavior of each instruction; ii) in the Architecture Resources (AC_ARCHC) description, the user informs storage devices, pipeline structure, memory hierarchies, etc. Based on this two descriptions, ArchC will generate a behavior simulator written in SystemC for the architecture [6] .

ArchC is capable of describing hierarchies composed of caches and memories distributed at different levels. The cache can be customized by an user through parameters as follows:

- Associatity: direct mapped (”dm”) or 2-way associative (”2w”), 4-way associative (”4w”) or fully associative (”fully”).
- Number of lines: the number of cache lines.
- Word per lines: number of words storage in a cache line.
- Replacement strategy: least-recently-used (”lru”) or random (”random”). Only in associative caches.
- Write policies: write-through (”wt”) or write-back (wb).
- Write load policies: if the cache line was loaded when a write miss occurs: write-allocate (”wal”); or if the line was modified only on a down level and was not loaded on cache: write-around (”war”).

An example is illustrated in the cache declarations contained in the example in Figure 2 at lines 2 and 3.

The user creates the hierarchy by describing the connections among these devices, through the method `bindsTo`, as illustrate in the last two lines of the example.

Using these ArchC frameworks it is possible to extend the MIPS-vt model with memory hierarchy from Intel VT.

3.2. ArchC-based Works

The work [10] describes the cache configuration exploration using ArchC. ArchC was extended to support more details of a system memory. As a result the Sparc-V8 processor model had its memory organization optimized for an image processing application.

Current version of ArchC [9] is capable of describing and simulating (using SystemC) a multi-core processor. In [2], ArchC was extended in order to provide a mechanism to support multiprocessor platforms. The main contribution

```
1 AC_ARCH(mips){
2   ac_cache icache("dm",128,"wt","war");
3   ac_cache dcache("2w",64,4,"lru","wt","war");
4   ac_mem MEM:256K;
5
6   ac_regbank RB:34;
7   ac_wordsise 32;
8
9   ARCH_CTOR(mips){
10    ac_isa("mips_isa.ac");
11
12    icache.bindsTo( MEM );  //Memory hierarchy
13    dcache.bindsTo( MEM );  //construction
14  };
15 };
```

Figure 2. Example of cache declaration using ArchC.

for educational activities is related to the efficient design exploration.

4. Conclusions

This paper presented the memory virtualization techniques used in Intel VT technology. These techniques consist in a hardware-consulted nested page table that translates guest physical addresses to host physical addresses, allowing guest context switches without VMM intervation. To avoid TLB and cache flush at each context switching, the Intel VT technology provide a VPID. The CPU can use VPIDs to tag translations in the TLBs. This way in the context switch is not necessary to flush the TLB.

The proposal presented in this paper is to extend the MIPS-vt model, with memory hierarchy, modeling the nested page table and the VPIDs from Intel VT using ArchC.

In Section 3.1 was presented the ArchC framework necessary to modeling the virtualization memory techniques extending the MIPS-vt model. An example of the description of hierarchies composed of caches and memories was presented.

Future works focus on statistical analyses of the MIPS-vt with memory hierarchy model and design this new model with two or more cores, also based on an ArchC model.

References

[1] K. Adams. A Comparison of Software and Hardware Techniques for x86 Virtualization. *12th International Conference on Architectural Support for Programming Languages and Operation Systems*, pages 2–13, March 2006.

[2] C. Arajo. Platform Designer: An Approach for Modeling Multiprocessor Platforms based on SystemC. *Journal of*

Design Automation for Embedded Systems, 10(4):253–283, Springer 2003.

[3] M. K. Ferreira. From Intel VT-x to MIPS: An ArchC-based Model to Understanding the Hardware Virtualization Support. *Workshop on Computer Education Architecture, Beijing, China*, pages 9–15, June 2008.

[4] M. K. Ferreira. Modificaes no MIPS Inspiradas na Intel VT-x para Suporte Virtualizao Utilizando ArchC. *Escola Regional de Alto Desempenho, Santa Cruz/RS*, pages 245–248, March 2008.

[5] G. Neiger. Intel Virtualization Technology: Hardware Support for Efficient Processor Virtualization. *Intel Computer Journal*, 10:166–178, August 2006.

[6] S. Rigo. Teaching Computer Architecture Using an Architecture Description Language. *Workshop on Computer Architecture Education*, 2004.

[7] J. S. Robin. Analysis of the Intel Pentium's Ability to Support a Secure Virtual Machine Monitor. *9th USENIX Security Symposium*, page 16, March 2000.

[8] R. Rose. Survey of System Virtualization Techniques. *CiteSeer.IST*, http://citeseer.ist.psu.edu/720518.html, March 2004.

[9] T. A. Team. The ArchC Architecture Description Language Reference Manual. *Computer System Laboratory (LSC) Institute of Computing*, http://www.archc.org, 2004.

[10] P. Viana. Exploring Memory Hierarchy with ArchC. *Symposium on Computer Architecture and High Performance Computing*, pages 2–9, 2003.