Virtual-GEMS: An Infrastructure To Simulate Virtual Machines

Antonio García-Guirado, Ricardo Fernández-Pascual, José M. García
Departamento de Ingeniería y Tecnología de Computadores
Facultad de Informática - Campus Universitario de Espinardo
Universidad de Murcia, 30100-Murcia (Spain)
{toni, rfernandez, jmgarcia}@ditec.um.es

Abstract

Recently, virtualization has become a hot topic in computer architecture research. The cost reduction and management simplification brought by server consolidation are good reasons why virtualization has become so popular. But there is a lack of tools for researchers to seek new proposals of architectures that improve the performance of virtualized systems. To fill this niche we have developed Virtual-GEMS, a multiprocessor simulator that allows us to simulate the behavior of a virtualized system and research new architectures suitable for virtualization. For testing Virtual-GEMS, we describe and evaluate some configurations for the shared L2 cache of a 16-core CMP running 4 virtual machines.

Our main contribution is the ease of configuration of simulations of virtualized workloads. Virtual-GEMS uses ordinary system checkpoints as virtual machines. This way, it avoids the need to create complex checkpoints including the hypervisor and the images of the virtual machines to simulate.

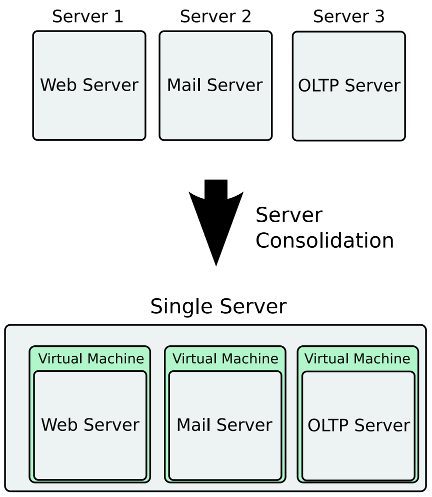


Figure 1: Server Consolidation. *With Server consolidation several servers are brought inside a single (more powerful) server, reducing costs and simplifying management.*

1 Introduction

Nowadays, full system virtualization is receiving renewed interest after years of relatively little activity. One of the main reasons is its application to server consolidation, which is the most important use of virtualization today.

If only one application is being run in a big server, then it is probably underutilized because most applications usually do not have enough parallelism to use all the processors or do not require all the available resources. Additionally, without server consolidation, a typical data center is made up of several small servers for executing different services, making the management more complex and expensive.

The solution to this is server consolidation, achieved through virtualization. In short, consolidation brings several servers into a single big server (Figure 1). This cuts management costs, as the number of machines to manage and maintain gets reduced. In contrast to executing every application on the same operating system instance, using a VM (*Virtual Machine*) for each service provides more flexibility (for example, the ability to use different operating systems for different applications) and more security, isolating each application from the others.

Simulation is used in computer architecture research to validate and evaluate new proposals. For example, simulation can be used to determine which is the best branch prediction scheme or to test new coherence protocols. It would be logical to use simulators for also determining which architectures or architecture parameters are best for a virtualized system. However, the simulators available do not provide adequate mechanisms for researching in this field in an easy way.

In this paper we present Virtual-GEMS, a simulation infrastructure that actually provides the ability to simulate virtual machines. Virtual-GEMS is based on Simics [10] and GEMS [11] simulators, and it provides full-system virtualization, i.e., each virtual machine runs its own operating system instance.

The rest of the paper is organized as follows. In the next section we give some background about virtualization and simulation. Section 3 explains the process of developing Virtual-GEMS and the structure of the simulator. In section 4 we describe the tests performed to show the use of the simulator and its usefulness for researching architectures targeted to the use of virtualization. Section 5 briefly describes the features of Virtual-GEMS. Finally, in sections 6, 7 and 8 we present some related work, proposals of future work and our conclusions, respectively.

2 Background

2.1 Virtualization

Virtualization was first introduced by IBM [5] in the 1960s in the experimental M44/44X. The IBM M44/44X simulated multiple IBM 7044 machines. Later, the IBM CP-40 operating system extended virtualization bringing the full virtualization concept, being able to create up to fourteen virtual S/360 environments in a unique and modified IBM S/360-40 machine.

Virtualization is a commonly used technology that provides the following advantages:

- Executing several operating system instances on the same physical machine. The different instances can be of the same or different OSes.
- Giving users or user groups the impression that they have a different machine each, although in fact there is only one real machine.
- Executing several applications in different virtual machines so they do not interfere with each other.
- Executing in a target architecture some software compiled for another architecture.

Virtualization provides flexibility for using the available resources thanks to removing barriers imposed by incompatible interfaces (such as different OSes or ISAs).

2.2 Simulation

Simulation in the computer architecture research field consists of reproducing the behavior of one machine by means of a computer program that imitates the characteristics of the machine under study. This simulation can include aspects from the simulated machine like functionality, energy consumption or timing, depending on what we are interested in measuring.

In computer architecture research, simulation provides a number of very interesting advantages. It allows the evaluation of the effect of various architectural parameters without needing to actually build the real machine but by setting these parameters in the simulation software. This makes possible to perform an extensive exploration of the design space. Simulation can also provide more diverse statistics than those that can be obtained from the real hardware, and these statistics can be extracted in an easier way than using real machines for measuring.

Additionally, simulation enables a very high degree of automatization, allowing the execution of many parallel simulations.

Unfortunately, simulation has some problems. The main problem is the need to trade between speed and accuracy.

Speed: A typical simulator can be several orders of magnitude slower than a real machine, hence the applications and number of simulations that can be run in it are limited. If we want to boost speed of simulations, we need a more simple and less detailed simulator, and the results become less faithful to the real machine.

Accuracy: It is difficult to faithfully recreate the behavior of a real machine. Additionally, as the simulator becomes more accurate and complex, it also becomes slower.

There are lots of different simulators, with different approaches and scopes. For example, SimpleScalar [2] is a superscalar single-processor simulator, whereas RSIM [8] simulates a multiprocessor machine. While these simulators measure the performance of the processor, other simulators, like Wattch [3], focus on the energy consumption. All these simulators directly execute the workloads (e.g. the SPLASH-2 benchmark suite) on the simulated machine.

On the other hand, GEMS [11] is a multiprocessor simulator developed by the Multifacet Project from the University of Wisconsin-Madison. It is based upon Simics, a full system simulator released in 2002 and developed by Virtutech.

Simics [10] is a simulator accurate enough to execute unmodified operating systems and even device drivers. It focuses on functional simulation, and can be expanded in many ways by building modules that give it new features.

GEMS extends Simics so that it can, among other things, simulate the timing of a detailed and configurable memory system. For that purpose, GEMS implements several Simics modules. One of them, Ruby, is the responsible for simulating the memory system.

This modular simulation infrastructure decouples functional simulation (driven in Simics) and timing simulation (driven in GEMS).

Another important feature when simulating with GEMS and Simics is that checkpoints are used. A checkpoint is a snapshot of the system state taken right at the moment in which performance measures start to be taken, so the checkpoint can be used as many times as necessary to perform simulations.

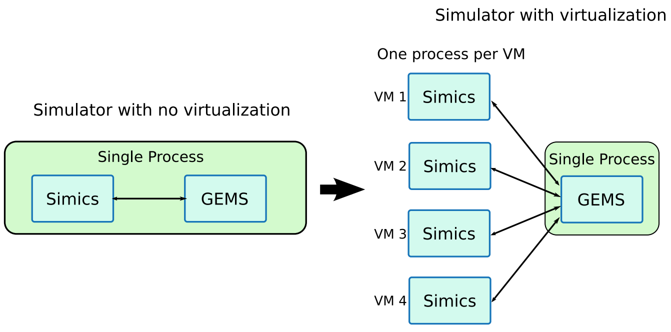
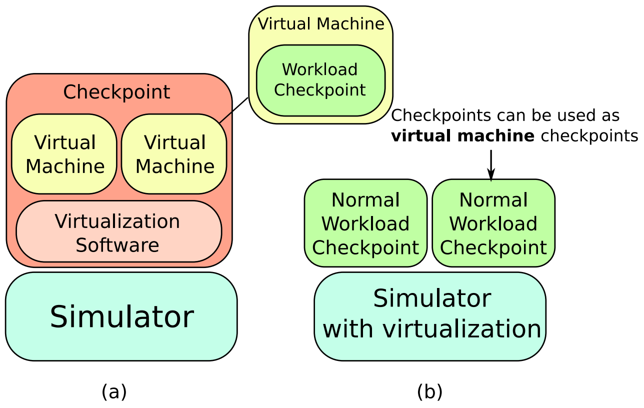


Figure 2: *Evolution from simulating a single real machine to simulating a virtualized system.*



Virtual Machine
Workload Checkpoint
Checkpoint
Virtual Machine
Virtual Machine
Virtualization Software
Simulator
Checkpoints can be used as **virtual machine** checkpoints
Normal Workload Checkpoint
Normal Workload Checkpoint
Simulator with virtualization
(a)
(b)

Figure 3: Virtualization Schemes. *With in-simulation virtualization software very complex checkpoints must be used (a), while with virtualization inside the simulator the already available checkpoints of the workloads can be used as VMs (b).*

3 Virtual-GEMS

Virtual-GEMS intends to simulate several virtual machines. It is based on Simics and GEMS simulators as its basic constructing blocks. Virtual-GEMS simulates each virtual machine by means of a Simics instance (only functional simulation), each one with its own OS and workload. But in order to get a single view of the whole server, a single GEMS instance is used to simulate the timing of all virtual machines. To do that, we first decouple Simics and GEMS, and then we develop a mechanism to connect several Simics instances (i.e. virtual machines) to the same GEMS timing simulation (see Figure 2). This way, we approximate the behavior of a virtualized system. We do not simulate any software virtualization layer, instead, in the current implementation, we assume that every virtualization issue is managed in hardware. The design of Virtual-GEMS allows us to implement the functionality of the hypervisor as part of the simulator.

The first decision is to choose the more adequate virtualization scheme. Virtualization software could be executed directly inside the simulation. If in-simulation virtualization software were used, to create a checkpoint we would have to include the virtual machine images inside the new checkpoint. And these virtual machine images are in turn checkpoints of the workloads to be simulated. Modifying any workload or any virtualization software parameter would require to build a new checkpoint, and creating such a complex checkpoint is a time demanding task.

The other approach is to implement virtualization in the simulator itself, allowing the direct use of the workload checkpoints already used for simulations that do not involve virtualization. This way, changing the workloads to use in each virtual machine or changing the virtualization parameters (which would be set in the simulator) would not require the creation of new checkpoints. The differences between both approaches can be seen in Figure 3. We have followed the later approach in our case.

This is the main contribution of our simulation infrastructure. We avoid the need to create a large number of complex checkpoints to perform simulations. We also provide a simple hypervisor inside our infrastructure. In order to include new virtualization features, this in-simulator hypervisor is easier to expand than in-simulation virtualization software. The latter would also require to create new complex checkpoints everytime the virtualization software is modified, whereas our hypervisor can be freely modified and the original checkpoints can still be used.

3.1 Modifying the Simics-GEMS communication

The Simics simulator provides interfaces to be extended. Users can use these interfaces to extend some missing functionality in the simulator. For example, the memory access timing can be developed in a timing-model interface.

Processors modelled by Simics are simple in-order nonpipelined processors which block at every memory access. GEMS provides a more detailed out-of-order processor simulator in a module called Opal. Due to the core simplification that seems to be the trend nowadays in many-core CMP designs (mainly due to power consumption and heat constraints), we use the Simics simple model in our infrastructure instead of the out-of-order model simulated by Opal.

The most important element in GEMS is the Ruby module, which is responsible for simulating the memory system.

The interface between Simics and Ruby is another module that we call SimicsRuby, which passes and preprocesses the memory requests from Simics to Ruby and returns to Simics the information about completed requests.

The structure of a simulation with Simics and GEMS can be seen in Figure 4.

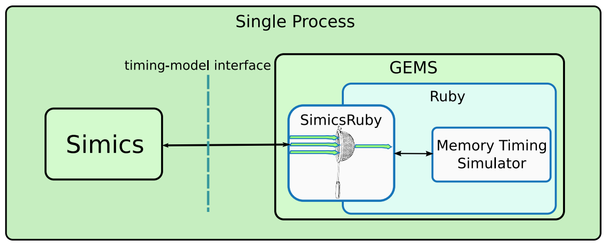


Figure 4: Simics/GEMS structure. *Simics performs the functional simulation, while GEMS performs the timing simulation.*

The module implementing the timing-module interface (SimicsRuby) receives all the information of each memory request as these are performed by the processors simulated by Simics. This module returns the number of cycles that the requesting processor will stall until the access is completed. The information concerning the memory access that is passed through the timing-module interface includes the memory address, the kind of access (read, write), and other necessary information.

The synchronization between Simics and Ruby is an important aspect. Memory requests do not carry any information about the cycle in which they are performed. Thus, some additional mechanism is needed to carry out this synchronization.

To do that, a Simics API function is used for registering a callback in Simics. This function is called each time a fixed number of cycles passes. When the callback is executed, the events in Ruby corresponding to that cycle are executed too.

Unfortunately, when a memory request is issued, we cannot predict how many cycles it will take to complete. Instead, the actual implementation blocks the requesting processor for a huge number of cycles, such as to be sure that the request will be completed before that processor wakes up. As the simulation advances, the events in Ruby keep executing each cycle, and eventually the memory access that blocked the processor will be completed. Then, Ruby will unblock the requesting processor (through SimicsRuby) so it can continue its execution. This way, the requesting Simics processor stays blocked from the moment it sends the request until the time the request is satisfied. This mechanism feeds back the timing information to the functional simulator.

As an intermediate step in the construction of Virtual-GEMS we have divided the SimicsRuby module to decouple Simics and Ruby. Two new modules have been developed. The first one is the module called mi-device, which implements the timing-model interface and receives the Simics memory requests. All the Simics dependent code has been put into this module (e.g. all the calls needed to perform the synchronization).

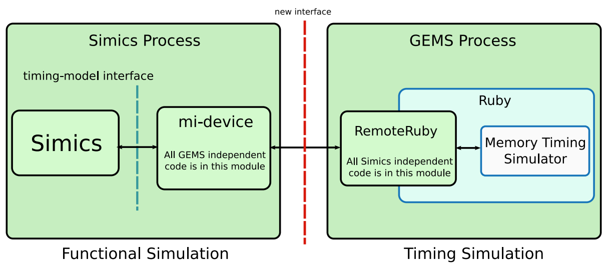


Figure 5: Decoupled Simics-GEMS structure. *The previous SimicsRuby module gets divided into mi-device and RemoteRuby modules.*

The second module is referred to as RemoteRuby, and contains all the calls to Ruby elements. It communicates through a pipe with mi-device using a new and very simple interface which is used to send all the information concerning memory accesses and synchronization.

This way, we move from a single process executing Simics and GEMS to two different processes: one for Simics and other for GEMS, communicating through FIFO pipes. The resulting structure can be seen in Figure 5.

3.2 Virtualization in Virtual-GEMS

First of all, we need to remind that our focus is on full-system virtualization. Each virtual machine runs its own operating system instance.

The virtualization process has to handle three main elements: processors, memory, and I/O devices. In our scheme, processors are physically partitioned so each virtual machine runs in a subset of the processors of the real machine. I/O buses are also physically partitioned, so each VM has its own set of disks and other devices. Memory is logically partitioned. The information of different VMs is interleaved in memory with a memory page granularity. This scheme is depicted in Figure 6.

Virtual memory is a key point to be properly handled in the virtualization context. Broadly speaking, virtual memory is supported by a page table for each process that maps the *virtual memory addresses* handled by each process to *real memory addresses* which can be found in memory or in swap space in the disk. This page table is usually handled by the OS.

In the virtualization environment, an extra level of page tables is needed. This extra level of page tables is managed by the hypervisor, and contains a page table for each VM. This page table maps the *real memory addresses* accessed by a VM to *physical memory addresses* in the physical machine. This is an approach similar to the one taken in Cellular Disco [6].

To build this virtualization scheme, Virtual-GEMS uses different Simics instances as VMs (see Figure 2). Each Simics instance is a different process, simulating a complete

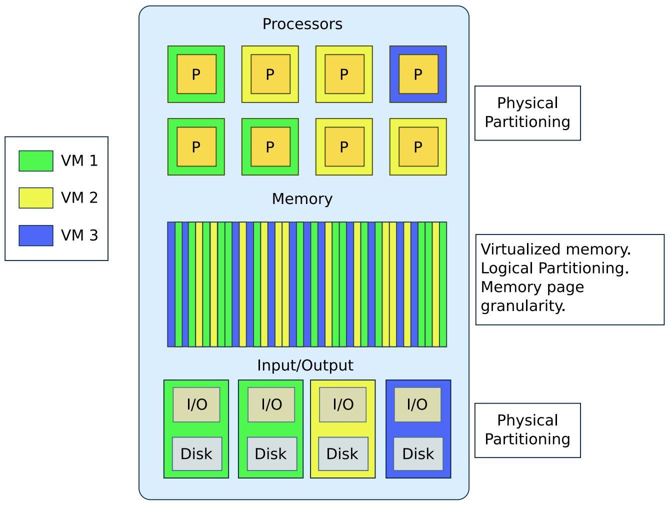


Figure 6: Virtualization Scheme. *Physically partitioned processors. Page level memory virtualization. Physically partitioned I/O.*

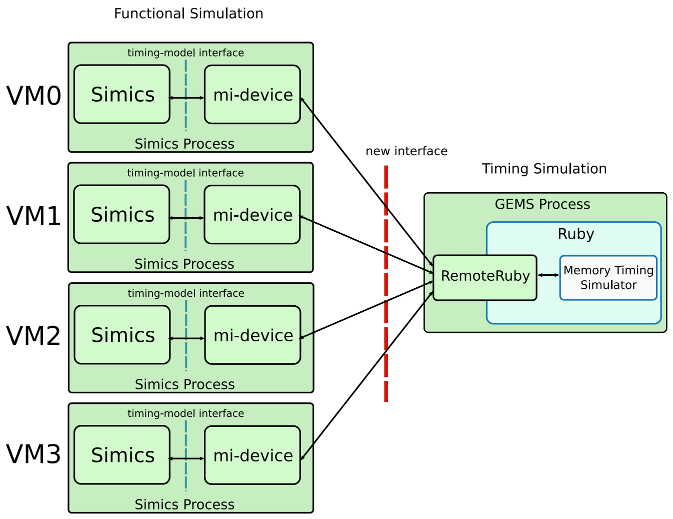


Figure 7: Structure of Virtual-GEMS. *Each VM is supported by a Simics instance. The timing simulation of the whole system is performed in GEMS. RemoteRuby manages all the virtualization issues.*

virtual machine in a functional manner, with its own processors and I/O devices. On the other hand, a specific subset of the processors simulated by Ruby corresponds to each VM. Therefore, we need to do a mapping between Simics processors and Ruby processors. We also need to map real memory pages from each VM to physical pages in the real machine using the new page table level mentioned before. All these virtualization issues (which would be handled by the hypervisor in a real system) are implemented in the RemoteRuby module which also controls all the concurrency and synchronization issues between VMs. Figure 7 shows the final structure of Virtual-GEMS.

3.3 Synchronization and parallelism

We force the execution of all the Simics instances to advance steadily. To do that, every mi-device module attached to a Simics process sends a trigger-event message to RemoteRuby every fixed number of Simics cycles, and then stops and waits for confirmation from RemoteRuby. Once RemoteRuby has received the trigger-event messages from every mi-device module, the virtual machines are synchronized. Then, RemoteRuby executes the events corresponding to that cycle and then sends the confirmation messages to the mi-device modules. The Simics instances can now continue the functional simulation of the virtual machines. With this process, the execution of every virtual machine advances one Ruby cycle. The equivalence between Simics cycles and Ruby cycles is customizable and it is used to poorly approximate an N-way superscalar processor by using the Simics simple processors, where N is the value of the cycle multiplier. Therefore, all the memory requests performed by the Simics instances between two RemoteRuby confirmation messages are considered to be issued in the same Ruby cycle. This process is the same as the one performed in the original GEMS, but it now involves several Simics instances instead of a single one. Thanks to the trigger-event and confirmation messages, the synchronization and advance of all virtual machines does not introduce any loss of accuracy into the simulation.

In contrast to the original GEMS, where the whole simulation was performed in a single thread, Virtual-GEMS shows some level of parallelism. The functional simulation of each VM is performed in a separate Simics process, allowing each VM simulation to execute in a different core of our simulation servers. Unfortunately, the heaviest part of the simulation corresponds with the timing simulation performed by Ruby, which is single threaded.

Therefore, Virtual-GEMS can speed up the simulation when executed in a multicore server, i.e., allowing several parallel simulations of few-core VMs instead of one single simulation of a many-core full system. On the other hand, the communication between processes introduces an important overhead, reducing the benefits of the parallelization.

4 Testing the simulator

Once the structure of the simulator has been shown, in this section Virtual-GEMS is used to evaluate the best L2 configuration for a multicore architecture in which virtualization is used. The main purpose of this evaluation is to test the simulator and check the sanity of the results.

4.1 Simulated Architectures

Virtual-GEMS allows the simulation of a wide range of machine configurations. Almost all parameters that it models can be specified, such as the interconnection network or the memory coherence protocol (and even new protocols can be developed with the SLICC language provided with GEMS).

Our base architecture is a tiled-CMP [16] with 16 tiles. Each tile has a processor, a private L1 cache, and a bank of the shared L2 cache. Since the L2 is shared, cache coherence must also be kept at L1 level.

On top of this base architecture, we set up four virtual machines, with four tiles each. We evaluate two different configurations that we derive from our base architecture, and we call them *Fully Shared L2* and *Partially Shared L2*. In the *Fully Shared L2*, all the L2 banks are shared among all the cores in the chip, regardless of which virtual machine they belong to. In the *Partially Shared L2*, the L2 banks are only shared by the tiles of a specific virtual machine. Therefore, L2 banks are not shared among VMs but they are private to each VM. The simulated architectures are shown in Figure 8.

The placement of data lines in L2 cache banks is determined by a subset of the bits of the physical address. In the case of *Totally Shared L2* these bits choose the specific bank among all the banks of the chip. On the other hand, for *Partially Shared L2*, the bits choose the specific bank among those private to the VM. In both cases, these bits are out of the page offset part of the address, hence the hypervisor has control over them when it performs the mapping from real address on the VM to physical address.

We take advantage of this fact to try to place data as close as possible to the cores that will use them. To do this, the hypervisor chooses the physical address that will correspond to each real address of each VM so that the memory lines of that page will map to the desired L2 bank.

We have used three different real-to-physical address mappings based on the proposals of Cho et al. [4]. The first mapping is a simple arrival order assignment, which maps a new real memory page from a VM to the next free physical page. This approach, called *simple mapping*, does not control where the data is located in the chip (i.e. in which L2 cache bank), so we consider it as an almost random assignment. The second one, called VMCBM (*Virtual Machine Cache Bank Mapping*), maps all the real pages from one VM to the L2 banks belonging to that VM, using a round-robin order to choose the L2 bank inside the VM. The third mapping is referred to as TCBM (*Tile Cache Bank Mapping*) or first-touch policy. It brings the data to the L2 bank of the tile whose core first accessed the page. Hence, when a memory page is first accessed, it is mapped to the L2 of the core that accessed it.

For the *Fully Shared L2* configuration we only use the simple mapping policy, while for the *Partially Shared L2* configuration we use the three real-to-physical address mapping policies described before.

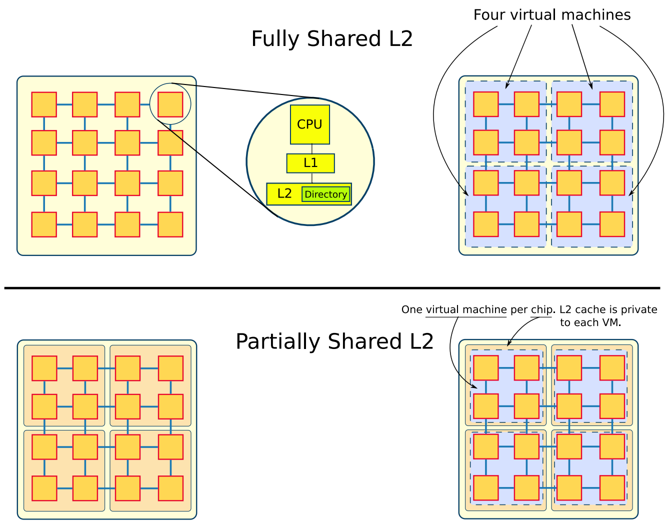


Figure 8: Simulated Architectures. *The tiled CMP with Fully Shared L2 (above) and the Virtual Machines placement on it. The Partially Shared L2 configuration (below) and VMs placement.*

Name	Architecture	Real-to-Physical Address Mapping
config1	Fully Shared L2	simple mapping
config2	Partially Shared L2	simple mapping
config3	Partially Shared L2	VMCBM
config4	Partially Shared L2	TCBM

Table 1: Machine Configurations Tested.

From now on, we refer to the four configurations used as config1/2/3/4, as they can be seen in Table 1. The common characteristics of the architecture can be seen in Table 2.

Finally, for the configurations tested, we use two different memory coherence protocols, both based on a MOESI state scheme. The first one is a two level directory based protocol [12]. Upon an L1 miss, the directory located on L2 is accessed to look for the information about the block state, so the miss can be satisfied. This process is repeated if the data is not found in L2 and the next level of the coherence protocol is used to find the block in memory. The directory introduces one or even two levels of indirection on a cache miss since it needs to access the directory in L2 and maybe also the directory in memory. The second protocol is a token-based one [13]. It uses a set of simple token counting rules that ensure cache coherence. Each memory block has a set of tokens associated to it. For reading a block, a tile needs to hold at least one token associated to that block. For writting, all tokens associated to that block are needed. When an L1 miss occurs, the core looks for tokens by sending a broadcast to all the cores so that the nearest one who possesses tokens can send some tokens back. This removes

Processors	16 UltraSPARC-III+ 4-ways, in-order. 2 GHz
L1 Cache	Split I&D. Size: 128KB. Associativity: 4-ways. 64 bytes/block. Access latency: 2 cycles.
L2 Cache	Size: 1MB each *slice*. 16MB total. Associativity: 4-ways. 64 bytes/block. Data array access latency 15 cycles.
TLB	64 entries, totally associative
RAM	4 GB DRAM. 1 memory controller for each chip. Memory latency 160 cycles + on-chip latency
Interconnection	Bidimensional mesh 4x4. 64 bytes links. 8 latency cycles by link.

Table 2: Common target architecture characteristics.

Workload	**Description**	**Size**	**Simulation**
apache4x4p	Web server with static contents. Surge as workload generator	3000 transactions per VM	Four 4-processor Apache VMs
jbb4x4p	Java server	5000 transactions per VM	Four 4-processor JBB VMs
unstructured4x4p	Fluid dynamics application	Mesh.2K, 5 time steps	Four 4-processor Unstructured VMs
barnes4x4p	Simulation of gravitational forces	8192 particles	Four 4-processor Barnes VMs
commercial	Mix of commercial workloads	See Apache & JBB sizes	Two 4-processor Apache VMs, Two 4-processor JBB VMs
scientific	Mix of scientific workloads	See Unstructured & Barnes sizes	Two 4-processor Unstructured VMs, Two 4-processor Barnes VMs

Table 3: Workload configurations tested.

the level of indirection imposed by the directory but requires more interconnection network bandwidth.

4.2 Workloads

We use a mix of commercial and scientific workloads for testing Virtual-GEMS. We use two commercial workloads: the Apache web server with static contents, and the JBB Java server workload. For the scientific workloads we use one of the SPLASH-2 suite benchmarks [17] called Barnes, which simulates gravitational forces, and another scientific benchmark called Unstructured, which is a computational fluid dynamics application that uses an unstructured mesh to model a physical structure, such an airplane wing or body. The details of these workloads can be found in Table 3.

4.3 Methodology

In the virtual machine environment, with a chip running four different VMs, we need to define a metric for measuring the performance of the full system. One alternative is to run the simulation for a fixed amount of time, and then count the transactions (i.e. units of work) completed by the workload in each VM to measure the performance. But this approach has several problems, as the execution of scientific workloads (Barnes and Unstructured) cannot be easily split in transactions. Moreover, since a transaction is a unit of work which depends on the particular benchmark, counting transactions for a workload made of a mix of different benchmarks is not straightforward. At least, we would need to weight transactions from different benchmarks differently (for example, a transaction of Unstructured would be equivalent to several hundreds of transactions of Apache).

Hence, we have decided to fix the amount of work that each workload performs for the different evaluated configurations. In this way, however, each VM of a simulation can finish its execution at a different time than the others, and this poses the problem that the simulation becomes less significant as the number of running VMs decreases because the interactions between VMs disappear. Hence, we have balanced the size of the different workloads so that every workload take approximately the same time to complete. Despite this adjustment, the finishing times of the simulated VMs do not exactly match, and as the parameters of the simulated machines change and the performance of each workload executed in the VMs improves (or worsen) this effect gets more noticeable. To account for this issue, we use the average number of cycles elapsed by all the VMs in the simulation to complete their workloads as the performance metric regarding the whole system.

4.4 Results

We use config1 as the base configuration to compare the rest of the configurations that have been proposed in this work. Config1 is the base situation of a CMP architecture with no special adaptation to virtualization or multiprogramming, while the rest of configurations try to improve the performance of this one by using the real-to-physical address mapping policies explained before. The results of these evaluations are shown in Figures 9 and 10. These results may differ slightly from the actual performance of the modeled systems because we do not account for the performance overhead introduced by the hypervisor in the current version of Virtual-GEMS.

As we can see in the graphs, config2 obtains a very good performance, with an average speedup of 5.76% using the directory-based protocol and 7.06% using the token-based protocol. Config3 comes close with average speedups of 5.47% and 6.86% using directory and token based protocols respectively. As we noted before, the difference between these two configurations is that config2 physically divides the L2 in four private caches, one for each VM, whereas config3 makes this same division logically. The consequence of this difference is that the physical division of config2 makes the memory blocks accessed by the VM map to one of the four L2 banks of the VM's private cache based

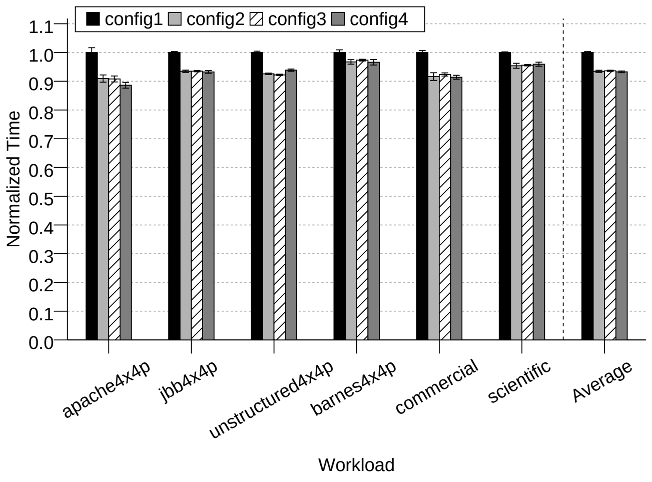


Figure 9: Test results using the directory-based protocol.

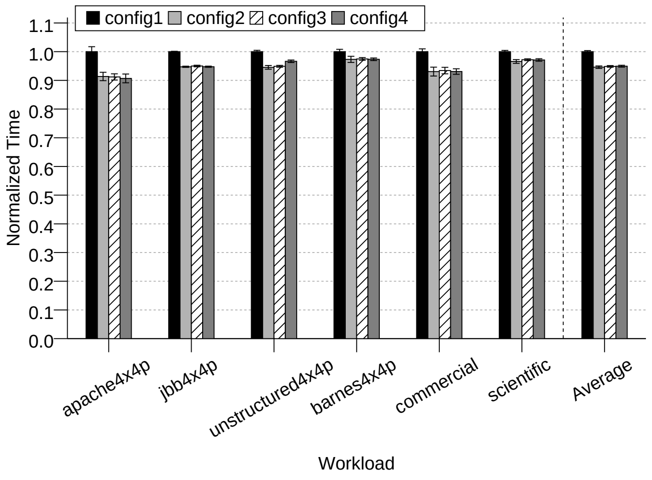


Figure 10: Test results using the token-based protocol.

on the block address, while the logical division of config3 allows us to map each new block accessed by the VM to the next cache bank using a round robin policy among the L2 banks belonging to the VM, balancing better the amount of data that each cache bank of the VM's private L2 must hold.

Config4 also presents a good performance, with an average speedup of 5.41% using the directory-based protocol and 7.3% using the token-based protocol. Config4 shows the highest speedup in a single workload: Apache using token protocol, with a speedup of 12.86%. Config4 maps a memory block to the L2 cache of the first core that accesses it. This can cut the memory latency as the data gets closer to the core that uses it. We are assuming that the core that first touches a block will use it at least as much as any other cores, but if it is not the case, or if the distribution of the data is not balanced among all cores, then we can be overloading some L2 banks and underusing the space in the rest of L2 banks.

The smallest performance improvement achieved by a *Partially Shared L2* configuration, respecting the *Fully Shared L2* configuration, is 5.47% (config3 using the directory protocol), and the smallest improvement in a single benchmark is 2.6% (config3 in Barnes using directory), with the rest of the results showing better improvements. It seems clear that *Partially Shared L2* is a better choice for these workloads than *Fully Shared L2*. Nevertheless, we are not able to point out which one of the three mappings (simple, VMCBM or TCBM) used in combination with *Partially Shared L2* performs best.

5 Simulator Features

Since it is built from Simics and GEMS, Virtual-GEMS allows us to set up most of the characteristics from the simulated machine in the same way than GEMS. It also provides deterministic execution of consolidated workloads. Nevertheless, different executions can be achieved feeding the simulator with different random seeds in order to account for the inherent non-determinism of parallel programs.

The assignment of cores to the VMs can be set up in the configuration of Virtual-GEMS (with a single string), making it possible to bound the virtual machine processors to physical processors as desired.

The hypervisor, which is implemented in the RemoteRuby module (see Figure 7) can perform many tasks. For example, we have previously shown that it carries on the management of the second level of page tables allowing different kinds of memory page mappings.

The hypervisor functions are easily expandable. Implementing a mechanism for page sharing among virtual machines is pretty straightforward. Mechanisms for dynamically reallocating the cores used by the virtual machines can be easily implemented in the hypervisor too.

The checkpoints used in the simulations in Virtual-GEMS are the same as in a normal Simics/GEMS simulation. We can pick and mix these checkpoints as needed, regardless of the specific configuration of each single workload (e.g. the number of processors of each workload used in a simulation).

In addition to the ability to bound virtual machines to physical cores, Virtual-GEMS also provides the possibility of bounding the L2 banks to virtual machines. Hence, if we are using the VMCBM mapping described before, we can set up the simulation to provide more L2 banks to the virtual machines with larger working sets.

Mechanisms for measuring approximately the different hypervisor overheads depending on the functions that the hypervisor performs (e.g. shared pages detection-mechanism overhead) can also be implemented in our infrastructure.

It is also possible to use a different functional simulator, instead of Simics, along with GEMS, just implementing the equivalent mi-device module (see Figure 5) for the simulator that we want to use.

Virtual-GEMS is publicly available at http://skywalker.inf.um.es/~toni/Virtual-GEMS/

6 Related Work

Research in the implications of server consolidation for computer architecture has grown over the last years. Enright et al. [9] showed that server consolidation raise interactions across the consolidated workloads, which open new paths to research. They demonstrated that workloads cannot be evaluated just in isolation when researching consolidated environments. They also used a similar approach to ours to simulate virtual machines in a consolidated server. However, our approach allows flexibility in the use of common workloads, in hypervisor-functionality development and also in the configuration of the computer architecture and the hypervisor, as shown in section 5.

Marty et al. [14] propose two new two-level coherence protocols. These protocols actually provide a virtual cache hierarchy adaptable to the virtual machines executing in the consolidated server, achieving considerable performance improvements in relation to traditional protocols. One important topic in that paper is that their protocols are well suited for inter-VM page sharing. However, their tests do not simulate this feature. In section 7 we consider the use of Virtual-GEMS to check whether the use of inter-VM page sharing can make the performance improvements achieved by their protocols even higher than the ones achieved in the memory-isolated VM environment.

In the work of Apparao et al. [1], an analytical performance model for consolidated workloads is developed, instead of using simulation. It takes into account three components: core interference, cache interference and virtualization overheads.

Finally, Hsu et al. [7] explore the cache design space for CMPs by using traces instead of full system simulation. Unfortunately, traces do not make it possible to consider the timing interactions between the execution of the different cores in the CMP.

7 Future Work

Among the future paths of research that are open, we consider that some of the most interesting are the following:

- Implementing page sharing among virtual machines. It will provide benefits in the cache usage, specially if the virtual machines run the same OS. On the other hand, the hypervisor introduces overhead since it has to run a mechanism to detect shared pages (like calculating some kind of hash function for each page and comparing them, and in case the hashes are the same, then comparing the full pages to check whether they are actually the same). This performance overhead can also be approximated with Virtual-GEMS.

- Reproducing virtual hierarchies [14] actually using page sharing among virtual machines. Also, it will be interesting to test the use of mechanisms like providing more space in L2 to the virtual machines with the largest working sets, by giving more L2 cache banks to those virtual machines that need it. We can also test the impact in performance of virtual machine reassignation to different cores (easily implementable in Virtual-GEMS), or, with a little more work, even killing some virtual machines and/or starting new ones in the middle of the simulation.

- Developing profiling mechanisms in the hypervisor. With live statistics about the execution we can dynamically reassign the resources of the physical machine to fit the needs of the virtual machines. This can provide support to check the potential for performance improvement of new ideas for dynamic reassignment.

8 Conclusions

Simulation is an important and extensively used tool in the computer architecture research arena. Both execution-driven and functional simulators are currently being used to design, evaluate and refine the new multicore architectures.

Virtualization has become a hot topic nowadays, because server consolidation is a good approach to reduce costs and management time. However, there is a lack of simulators usable for research in the virtualization field.

This paper introduces Virtual-GEMS, a new simulator based on Simics and GEMS that allows us to simulate the behaviour of a multicore architecture running several virtual machines on it. Virtual-GEMS is based on virtualizing the functional simulator (by creating several instantes of Simics, as many as the number of VMs), and doing the timing simulation with a single instance of GEMS. We have developed new interfaces between Simics and GEMS, and we have also discussed some issues on virtualization schemes.

The main contribution of our infrastructure is the ease for using ordinary checkpoints as VMs, instead of needing to build complex checkpoints including the hypervisor and virtual machines. Also, the hypervisor included in our infrastructure can be expanded with new functionality without having to create new checkpoints.

For testing the simulator, we have used Virtual-GEMS to evaluate the best configuration of the shared L2 cache of a 16-core CMP running 4 virtual machines, with different workloads, two different coherence protocols and three different mapping policies for the memory blocks to L2 cache.

We consider that our simulator can help to improve the new multicore architectures designed for being used for server consolidation.

Acknowledgements

This work has been jointly supported by the Fundación Séneca (Agencia Regional de Ciencia y Tecnología, Región de Murcia) under grant 00001/CS/2007, and also by the Spanish MEC and European Commission FEDER funds under grants "Consolider Ingenio-2010 CSD2006-00046" and "TIN2006-15516-C04-03". Antonio García-Guirado is also supported by a research grant from the Fundación Séneca.

References

[1] Padma Apparao, Ravi Iyer, and Don Newell. Towards modeling & analysis of consolidated cmp servers. *SIGARCH Comput. Archit. News*, 36(2):38–45, 2008.

[2] Todd Austin, Eric Larson, and Dan Ernst. SimpleScalar: An Infrastructure for Computer System Modeling. *Computer*, 35(2):59–67, February 2002.

[3] David Brooks, Vivek Tiwari, and Margaret Martonosi. Wattch: a framework for architectural-level power analysis and optimizations. In *ISCA '00: Proceedings of the 27th annual international symposium on Computer architecture*, volume 28, pages 83–94. ACM Press, May 2000.

[4] Sangyeun Cho and Lei Jin. Managing Distributed, Shared L2 Caches through OS-Level Page Allocation. In *MICRO*, pages 455–468, 2006.

[5] R. J. Creasy. The origin of the vm/370 time-sharing system. *IBM Journal of Research and Development*, September 1981.

[6] Kingshuk Govil, Dan Teodosiu, Yongqiang Huang, and Mendel Rosenblum. Cellular Disco: resource management using virtual clusters on shared-memory multiprocessors. *ACM Transactions on Computer Systems*, 18(3):229–262, 2000.

[7] Lisa Hsu, Ravi Iyer, Srihari Makineni, Steve Reinhardt, and Donald Newell. Exploring the cache design space for large scale cmps. *SIGARCH Comput. Archit. News*, 33(4):24–33, 2005.

[8] C. J. Hughes, V. S. Pai, P. Ranganathan, and S. V. Adve. RSIM: simulating shared-memory multiprocessors with ILP processors. *Computer*, 35(2):40–49, 2002.

[9] Natalie Enright Jerger, Dana Vantrease, and Mikko Lipasti. An evaluation of server consolidation workloads for multicore designs. *IEEE Workload Characterization Symposium*, 0:47–56, 2007.

[10] P. S. Magnusson, M. Christensson, J. Eskilson, D. Forsgren, G. Hallberg, J. Hogberg, F. Larsson, A. Moestedt, B. Werner, and B. Werner. Simics: A full system simulation platform. *Computer*, 35(2):50–58, 2002.

[11] Milo M. K. Martin, Daniel J. Sorin, Bradford M. Beckmann, Michael R. Marty, Min Xu, Alaa R. Alameldeen, Kevin E. Moore, Mark D. Hill, and David A. Wood. Multifacet's general execution-driven multiprocessor simulator (GEMS) toolset. *SIGARCH Comput. Archit. News*, 33(4):92–99, November 2005.

[12] Michael R. Marty. *Cache coherence techniques for multicore processors*. PhD in Computer science, University of Wisconsin - Madison, 2008.

[13] Michael R. Marty, Jesse D. Bingham, Mark D. Hill, Alan J. Hu, Milo M. K. Martin, and David A. Wood. Improving multiple-cmp systems using token coherence. In *HPCA '05: Proceedings of the 11th International Symposium on High-Performance Computer Architecture*, pages 328–339, Washington, DC, USA, 2005. IEEE Computer Society.

[14] Michael R. Marty and Mark D. Hill. Virtual hierarchies to support server consolidation. *SIGARCH Comput. Archit. News*, 35(2):46–56, May 2007.

[15] Jim Smith and Ravi Nair. *Virtual Machines: Versatile Platforms for Systems and Processes*. Morgan Kaufmann, June 2005.

[16] Michael B. Taylor, Jason Kim, Jason Miller, David Wentzlaff, Fae Ghodrat, Benjamin Greenwald, Henry Hoffman, Jae-Wook Lee, Paul Johnson, Walter Lee, Albert Ma, Arvind Saraf, Mark Seneski, Nathan Shnidman, Volker Strumpen, Matthew Frank, Saman Amarasinghe, and Anant Agarwal. The raw microprocessor: A computational fabric for software circuits and general purpose programs. *IEEE Micro*, 22(2):25–35, May 2002.

[17] Steven Cameron Woo, Moriyoshi Ohara, Evan Torrie, Jaswinder Pal Singh, and Anoop Gupta. The SPLASH-2 Programs: Characterization and Methodological Considerations. In *Proceedings of the 22th International Symposium on Computer Architecture*, pages 24–36, Santa Margherita Ligure, Italy, 1995.