VMcSim: A Detailed Manycore Simulator for Virtualized Systems

Alain Tchana*, Brice Ekane*, Boris Teabe* and Daniel Hagimont*
*University of Toulouse, Toulouse, France. E-mail: first.last@enseeiht.fr

***Abstract*—Designing a cloud infrastructure for HPC applications requires to correlate design choices for virtualization solutions, resources management strategies, and their implementation on specific hardware platforms. Such investigations can hardly be conducted without accurate simulation tools. This paper introduces VMcSim, the first microarchitecture and virtualized manycore simulation framework. VMcSim models a x86-based asymmetric manycore microarchitecture, including a virtualization layer which allows to model a hypervisor, a set of virtual machines with their virtual resources (CPU and memory), their operating system, and their applications. By simulating all the involved hardware and software layers, VMcSim outperforms other simulators. Simulation accuracy is evaluated through several benchmark (SPLASH-2). The simulator is demonstrated with the evaluation of virtual resources placement policies in multicore systems.**

I. Introduction

Cloud computing [1] technology has proven to be an useful technique. Based on virtualization, it allows to acquire very quickly and easily a large range of servers. These recent years have seen HPC applications developers and industries thinking about the migration of their applications to the cloud [2]. Preliminary works [10], [11], [12] point the non isolation of microarchitecture-level components which results to contention. It is crucial to continue investigating such researches at a micro-architecture level. More-often, researchers rely extensively on simulation to understand, explore and evaluate their work. Regarding cloud simulators [13], they do not model neither the fine grained aspects of computers architecture nor internal virtualization functioning. The lowest element they model is most often the virtual machine (VM). Regarding microarchitecture simulators [3], [14], they are not aware of the virtualization technology. Therefore a microarchitecture and virtualized simulation framework that combines the two research domains is needed. This paper introduces VMcSim, the first simulation framework that meets these needs. VMcSim models a x86-based asymmetric manycore microarchitectures in detail for both core and uncore subsystems, including a virtualization layer: set of VMs including their virtual resources (vCPU and memory), their operating system (applications scheduling), and their applications. VMcSim also provides a way to model a hypervisor with its main components: vCPU scheduling on top of physical CPU, and memory allocation to VMs. Using the microarchitecture-level simulator McSimA+ [3] as its foundation, VMcSim models in-order and out-of-order, sophisticated cache hierarchies, coherence hardware, on-chip interconnects, memory controllers, and main memory. In summary, the main contributions we make in this paper are:

- We introduce VMcSim, a microarchitecture and virtualized manycore simulation framework.
- We perform intensive and rigorous experiments in order to validate VMcSim: functioning correctness and performance accuracy.
- We use VMcSim to study some virtual resources placement policies which minimize performance degradation when consolidating several VMs on the same multicore system.

The rest of the article is organized as follows. Section II motivates our work regarding the related works. Section III presents our simulator. Section IV presents experiments results of the validation and the evaluation of the simulator. Section V concludes the paper.

II. Related work

Grid computing is the direct ancestor of the cloud computing technology. Therefore most of the existing cloud simulators are based on grid simulators. GridSim [15] is a java based event driven simulation toolkit which allows to simulate a large scale distributed system. Relying on GridSim which is one of the most popular and open source grid simulator, cloud computing researchers have developed a set of cloud simulation frameworks. CloudSim [13] is an extensible simulation toolkit that allows to model a cloud computing environment. It is the most popular cloud simulator. It allows to model a set of VMs, physical machines, and network topology. For each VM, the user can model a set of jobs which will be simulated under the VM. CloudSim exposes custom interfaces for implementing provisioning strategies for the allocation of VMs. Several amelioration of CloudSim have been proposed.

Compared to research on cloud computing simulation, much more researches [22], [23] have studied the simulation of computers at a microarchitectural level. [14] presents ZSim, a high speed and accuracy simulator for modeling thousand-core chips. It uses dynamic binary translation and instruction-driven timing models. It parallelizes multicore simulation using the two-phase bound-weave algorithm, which breaks the trade-off between accuracy and scalability. [21] presents Time-Based Sampling (TBS), a framework that enables sampling in simulation of multi-core system with high scalability. Based on this framework, [21] introduces the simulator ESESC. The particularity of the latter is the fact that it allows to simulate power and temperature. McSimA+ [3] is a timing

simulation infrastructure that models x86-based asymmetric manycore microarchitectures in detail for both core and uncore subsystems. It dynamically instruments the execution of an application in order to trap all instructions. McSimA+ enforces a sequential execution of a multithreaded application and sequentially simulates instructions within a simulated manycore microarchitecture system.

The related work shows there is no simulator which allows to study the effects of virtualization at microarchitectural level. A simulation framework which is situated at the intersection of a whole cloud simulation and a microarchitecural level simulation is missing. A question one could ask is why not just combine a cloud and a micro-architecture simulator in order to provider this intersection simulator? It would have been nice, but this is not practically possible because existing simulators in the two domains are not compatible. For example, micro-architecture simulators generally take as input a set of instructions (execution or trace driven) from a real execution. Oppositely, cloud simulators (like CloudSim, on which most cloud simulators are based) are not able to provide such data. They either rely on resource consumption traces or internally generate a workload. The problem in this particular example is the translation of workload/resource consumption traces into a set of instructions. For this reason, we chose to develop a new simulator called VMcSim, starting from a micro-architecture simulator.

III. VMcSim: a virtualized microarchitecture simulator

This section presents VMcSim, the first simulator which allows to model and simulate in detail a virtualized multi-core system. It takes into account four levels of simulation: microarchitecture level (core, cache hierarchy, memory banks, etc.), hypervisor level (vCPUs scheduling and VM memory allocation), VM level (a set of VMs with their virtual resources and their operating system), and applications level (each VM runs a set of applications). To simulate the first level, VMcSim relies on McSimA+ [3] (with some improvements). Therefore, this paper does not dwell on the simulation of that level. Fig. 1 presents a high-level illustration of the architecture of VMcSim. It is organized into two main areas: the backend which provides simulation functionality, and the frontend which contains the instrumented applications including the hypervisor (see below). These two areas communicate through socket, either to gather instructions to simulate (from frontend to backend) or to send simulation orders (from backend to frontend). Before detailing this architecture, let us present the general functioning of the simulator. It takes as input:

- a description of the microarchitecture aspects of the computer the user want to simulate. This description includes core/hyperthread information, cache hierarchy (L1, L2, associativity, line size, number of ways, etc.), memory controllers, memory topology (crossbars, mesh, etc.), etc.
- a description of the hypervisor. This is essentially the vCPUs scheduling algorithm to simulate (e.g. Xen credit scheduler [5]), and the configuration parameters for that algorithm.
- a description of the set of VMs to simulate. For each VM the simulator requires its number of vCPUs and how they are pinned onto core/hyperthread, the amount of memory it uses at creation time, and its creation time (expressed in terms of CPU cycles).
- a set of applications. VMcSim simulates for each VM a set of applications which are supposed to run on top of that VM. Each application is specified using the command line which allows to start it. VMcSim sees an application as a set of threads and allows the user to pinned threads onto vCPUs, in the same way as vCPUs are pinned onto cores (see Section III-B).

Using these information, the simulator configures itself and launches applications (either locally or remotely). It does not require the application to run in a virtualized system. Using a Pin tool [6], VMcSim dynamically instruments the execution of each application so that any executed instruction is duplicated. Thus, a replica is sent to the simulator in order to be replayed within the simulated environment. At the end of the simulation process, VMcSim generates a statistics file including microarchitectural and virtualization operations that occurred during the simulation. These are cache miss, IPC (Instructions Per Cycle), CPU cycles, cache coherence, memory pages access, context switch, etc. The following sections describe in detail each level of simulation.

A. Application level simulation

Like McSimA+, VMcSim is execution driven. Therefore, it requires instructions from the execution of applications (a set of threads) which are supposed to run on top of simulated VMs. As mentioned earlier, these instructions are gathered using a Pin tool [6] provided with the simulator. They can either be collected and immediately sent to the simulator at application execution time (in this case the application runs in the frontend area during the simulation process) or the user can store them into a file and submits them later to the simulator as traces. Whatever the method, applications should be launched using the Pin tool provided with VMcSim. Using this tool, the simulator enforces sequential execution of any multithreaded application.

Each thread is given a slot. The latter corresponds to the number of instructions the thread can execute before being suspended by the simulator (in order to resume the execution of another thread, recall that VMcSim enforces sequential execution). During a slot, all instructions executed by the thread are buffered in order to be simulated at the end of that slot, before the start-up of the simulation of the hypervisor (see section bellow). As mentioned earlier, VMcSim does not require applications (in the frontend) to run in a virtualized environment. Indeed, it does not instrument (as it does with applications) the execution of the virtualization system. This may lead the simulator much more slower and not useful[1]. As

[1]The objective of a simulator is to simulate what we don't have!

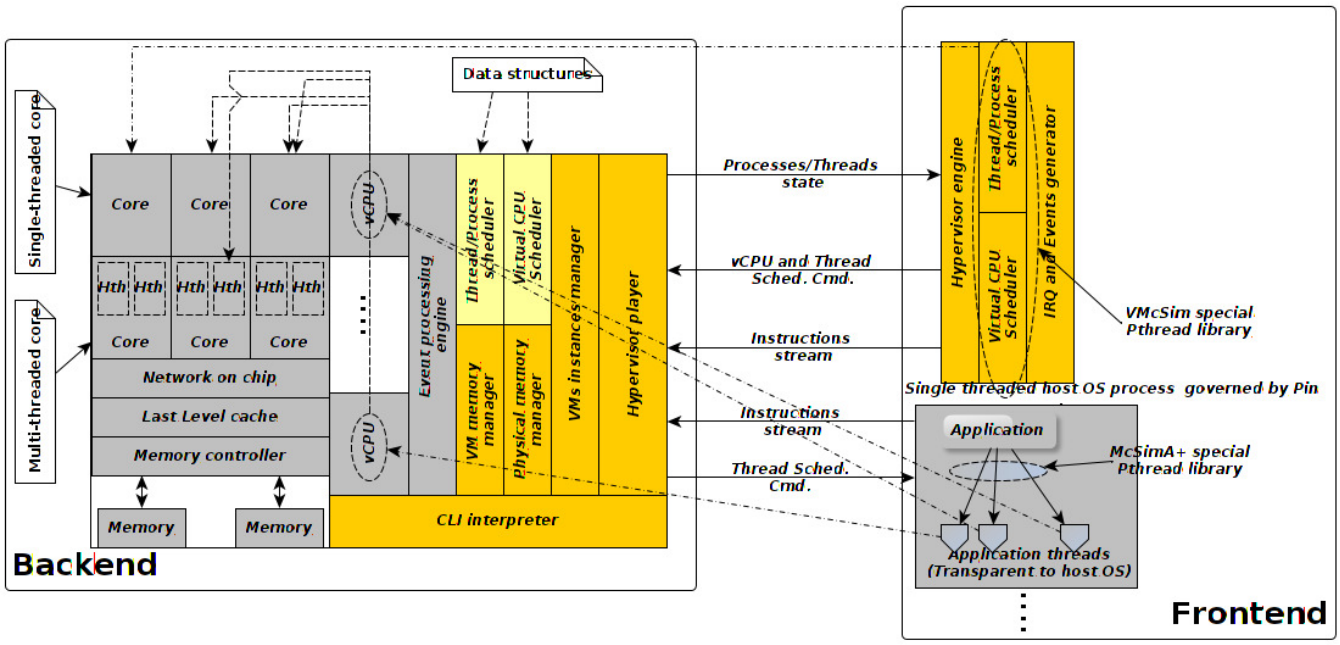

Fig. 1. VMcSim architecture.

we present below, we choose a much less costly solution.

B. Virtual machine level simulation

VMcSim simulates the functioning of a VM (which is an operating system) by simulating threads scheduling, even if the user can add others features (memory management, balloon driver). This scheduler is responsible for assigning vCPUs to threads. It is called by the hypervisor before the latter schedules vCPUs onto cores (see section below). VMcSim allows the user to pin threads onto vCPUs, as done with vCPUs onto processors. This feature can be very useful when the user knows how vCPUs are pinned onto cores at the hypervisor level. It may result to a significant improvement of performance because it allows a fine grained workload collocation ([19] shows a possible gain of about 167% with some benchmark). The scheduling algorithm is actually very simple. It equitably assigns vCPUs to threads. Its general functioning is as follows. The scheduler maintains for each vCPU a list of threads that vCPU can run. Then it sorts the list according to the total execution time (number of CPU cycles) of each thread. The thread with the lowest total execution time is put at the head of the list. The scheduler then parses each vCPU list of threads in order to ensure that a thread does not appear twice at the head of more than one list. After this step, the thread at the head of each vCPU list will receive the vCPU.

C. Hypervisor level simulation

The simulation of the hypervisor in VMcSim consists in simulating on the one hand virtual resources scheduling and on the other hand the execution of the hypervisor itself (we consider its instructions and VMs interruptions). The latter allows VMcSim to provide a high accuracy (e.g. simulate virtualization overhead).

1) Virtual resources scheduling: It concerns both VM memory allocation and vCPUs scheduling. About the former, its simulation is straightforward. The configuration file given to the simulator at start time contains for each VM, its memory size and where this memory will reside (e.g. in the case of a NUMA configuration with more than one memory controllers). Therefore, when the backend receives an instruction of an application from the frontend, it updates all addresses present in that instruction so that they fall within the memory area reserved for the VM to which belongs the application.

Regarding vCPUs scheduling, all threads are suspended before it starts. The hypervisor includes a component (called in this section *scheduler*) which performs vCPUs scheduling. It is executed either at the end of a slot or when the actual thread is suspended or ended. It is responsible for choosing for each VM's vCPU, the core on which it will run. Cores are organized as pools (a set of cores) and a scheduling algorithm is associated to each pool. Although others vCPUs scheduling algorithms can easily be integrated into VMcSim, it actually simulates the Xen credit scheduler [5]. A VM is configured with a set of vCPUs where a vCPU can be pinned onto a set of cores (belonging to the same pool). According to the scheduling algorithm, other parameters can be required (cap and weight in the case of Xen credit scheduler). Each time the scheduler is invoked, it only considers vCPUs on which at least one thread is assigned (see sections above for threads scheduling). Unlike McSimA+ where the scheduling process is very simply (each thread is assigned a dedicated core for its overall lifetime), VMcSim allows the execution of several vCPUs (thus several threads) on the same core. This feature makes sense in the context of the simulation of a virtualized system since it is one of the key study point when collocating

several VMs on top of the same machine. To this end, VMcSim implements a context switching mechanism which works as follows. For each core, the scheduler builds the list of vCPUs that can acquire it. The vCPU at the head of each list will receive the core associated to that list. If the thread which is running on the selected vCPU is different from the previous thread which was on the latest vCPU (on that core), a context switch is performed in the simulator. It consists in keeping back the stack of the latest thread and bind the stack of the actual thread with the core. After the scheduling task, the hypervisor chooses the core which thread will continue its execution (remember that the simulator enforces sequential execution). This selection is done in a round robin way.

2) Hypervisor execution: In addition to instructions which come from the execution of threads, the simulator also replays instructions associated to the execution of the hypervisor (not the real hypervisor, but the simulated one).

Hypervisor instructions simulation
The hypervisor is considered in VMcSim as an application like others which run on top of VMs. Thus its executions is located in the frontend area. This method allows VMcSim be flexible: the user can simulate several hypervisor implementations without having a real functioning hypervisor (which is very costly). We provide a particular Pin tool which instruments the execution of the hypervisor but only traps scheduling instructions, which represent its main activity. These instructions are sent to the backend area at the end of each scheduling process. Unlike VMs applications, hypervisor instructions can be executed by any core (chose randomly by the backend) as in a real virtualized system. The hypervisor has a representative component in the backend. This component responsible for keeping coherent hypervisor's data structures present in both areas. At the end of each slot, it sends threads state to the hypervisor (which is in the frontend area), vCPUs and threads scheduling respectively in the hypervisor and VMs need these information. In turn, at the end of the scheduling process, the hypervisor (in the frontend) sends back the new vCPUs and threads placement to its representative so that it updates backend's data structures.

Interruptions simulation
According to [20], a significant part of virtualization overhead comes from the treatment of interruptions in the hypervisor. VMcSim provides a way to simulate the presence of interruptions from/to VMs. We identify the most important of them (those which require the much more CPU cycles to be handled [20]) and the simulation of the hypervisor randomly generates for each VM a certain number. These are known as vmExit events [20]. For each generated vmExit event, the simulator increases the number of CPU cycles (according to configuration parameters specified by the user) in order to consider their impact. As we will see in the validation section, this allows the simulator to provide much more accuracy regarding hypervisor overhead. This method avoids the full simulation of the virtualized system, which is very costly. Also, it allows to keep VMcSim simply and useful.

D. VMcSim usability and flexibility

VMcSim provides a useful command line interface (CLI) based on socket. It allows the user to deal with the simulator at runtime. The main commands recognized by the simulator are the following.

- (1) *suspend* allows the user to suspend the simulation process. He can then use this breakpoint to have the actual state of the simulation (IPC, cache miss, etc.) by sending a *dump* command. Implementing the suspension of the simulation is straightforward. Indeed, since the backend is responsible for resuming threads (including the hypervisor), suspending the simulation is done by stopping sending execution orders to threads. By doing that the latter will stay blocked on the socket. Note that breakpoints can also be specified at hypervisor configuration time. In this case, the user tells VMcSim to suspend the simulation after a number of instructions or CPU cycles.
- (2) VMcSim allows the user to add or remove VMs and threads during the simulation process. In the same vein, virtual resources (memory and vCPUs) can also be added or moved to/from VMs. These features make sense in the context of virtualized systems which are very dynamic.
- (3) Finally, VMcSim allows to change vCPUs and threads scheduling algorithms. Indeed, each scheduler is clearly identified by a C++ class so that its integration into the simulator is facilitated. Note that any command which requires the modification of hypervisor data structures in the backend implies the same modification in the hypervisor within the frontend.

E. Difference between VMcSim and McSimA+

Figure 2 highlights in yellow new components we added to McSimA+ to implement VMcSim. The latter improves McSimA+ with two additional levels of simulation: hypervisor and VMs (combination of an OS used by a VM and its resources). As mentioned above, the simulation of the VM level could be useful for the user when he knows how VM's resources are mapped on top of physical resources: several threads scheduling on top of vCPUs could be studied. Threads scheduling provided by McSimA+ only allows it to enforce a sequential execution of multi-threaded applications. This feature has been kept in VMcSim and we have added another thread scheduling mechanism which allows the scheduling of many threads (via the vCPUs of their VMs) on the same processor or the scheduling of the same thread on more than one processor. Another difference between VMcSim and McSimA+ concerns flexibility and usability. The latter are not offered by McSimA+ while they represent an important criteria for a simulator.

IV. Validation and Evaluations

This section presents evaluations results of VMcSim. These evaluations include both the validation of the simulator and the exploration of a number of resource allocation policies which minimize performance degradation when collocating

FFT	-p1 -n64 -l6
Cholesky	default parameters
Radix	-p1 -n262144 -r1024 -m524288
Water-spartial	default parameters

TABLE I
CONFIGURATIONS OF SPLASH-2 APPLICATIONS WE USED.

several applications on top of the same manycore architecture machine.

A. Experimental methodology

Six types of experiments were run. The results of any experiment here is an average of 10 executions. We used two benchmarks for these experiments. The first one is a microbechmark we implemented. It is a C program which parses an array of integers and set each entry. By changing the size of the array, we are able to observe the effects of memory caches. In addition to this microbenchmark we used four applications from the SPLASH-2[18] suite: FFT, Cholesky, Radix, and Water-partial. Table I shows the configurations we used for each application. All of the experiments were run on a DELL Optiplex 755 with hyper-threading and L2 cache prefetcher turned off. The microarchitectural characteristics of this machine are summarized in Table II. It runs an Ubuntu 12.04 with 3.2.30 kernel (called "native system"). For virtualized experiments, we used the Xen 4.2.0 version (called "virtualized system") with para-virtualized guest VMs running the same kernel and distribution as the native system. We configured Xen to use the credit scheduler. Unless otherwise noticed, VMcSim is configured according to these characteristics.

For all experiments we used three metrics: cache miss rate, IPC, and execution time (number of CPU cycles). We relied on various tools to capture them. We used Linux Perf tool[2] to collect hardware performance counters in the native system. About the virtualized system, Perfctr-xen[17] provides direct access to hardware performance counters. It relies on the cooperation of the guest kernel and the hypervisor to provide profiling tools in the guest. This can be done by using APIs such as PAPI [16] (Performance Application Programming Interface). For our experiments, we developed five PAPI profiling tools respectively for the microbenchmark and the four SPLASH-2 applications we used.

B. Validation

The validation of an execution-driven simulator like VMcSim should consider two aspects: functional correctness (experiment type 1) that guarantees programs to finish correctly, and performance accuracy (experiments types 2-4) which ensures that the simulator faithfully reflects the reality (execution on the actual target hardware).

[2]https://perf.wiki.kernel.org/index.php/Main_Page

1) Functional correctness: The first type of experiment validates the functional correctness of VMcSim. As argued in [3], functional correctness is typically straightforward to verify, especially for the simulators with decoupled functional simulations such as VMcSim. As we have presented in Section III, the simulation of the microachitectural level (execution of instructions) in VMcSim is based on McSimA+ [3], which is demonstrated to be functional correct. When comparing VMcSim with McSimA+ in terms of the way they simulate the execution of instructions, the only difference is the capability of VMcSim to schedule more than one thread on the same core. As we mentioned, this feature requires the implementation of context switch in VMcSim. The implementation we provide respects two constraints: (1) no instruction is lost during context switch, and (2) each thread manages its own stack. The first condition is ensured by keeping and isolating all instructions of each thread into a separated queue events. Regarding the second constraint, we adopt in VMcSim a new design for thread stack mapping onto cores (see Section III-C1). In McSimA+ the mapping is statically implemented so that a core is absolutely linked to the stack of the unique thread it runs during the simulation process. In VMcSim this mapping is a one to many relationship: a core can be linked to many threads stack during the same simulation process, but only one at a time. We performed several tests with the microbenchmark and SPLASH-2 applications running on top of several VMs. We tested many scenarios from dedicated core mapping to multithreads mapping on the same core. These experiments validate the functional correctness of our implementation. Note that it is not necessary to presents the results of these experiments since they are those obtained when running SPLASH-2 applications within a native environment.

2) Performance accuracy: Experiments types 2-4 are dedicated to the validation of the accuracy of VMcSim. We started by validating the accuracy of McSimA+.

a) Accuracy of McSimA+: The second experiment type validates the accuracy of McSimA+. [3] validates the accuracy of McSimA+ by performing several experiments directly with the simulator and compared them with the results of experiments reported in [18] (SPLASH-2 publication). About the comparison with a native system, [3] just mentioned it but no results are reported. Therefore, we started by performing experiments in both a native system and McSimA+. Then we performed the experiments mentioned in [3]. To these ends, we used both the microbenchmark and SPLASH-2 FFT application (as in [3]) and we measured L1D cache miss rate. Fig. 2 presents the results of these experiments. The left curve presents the normalized values of cache miss ratio when running the microbenchmark while varying the size of the array. We can see that the results provided by McSimA+ are close to those measured in the native system, with an average error of about 2% (as reported in [3]). The two rightmost curves show the execution of FFT in McSimA+ (as did in [3]). The middle one comes from the original publication of McSimA+. The rightmost presents the results of the replay we did. We can see that the two curves have basically the

Freq (GHZ)	2.65	RS entry	32	L2$ Shared	3MB, 12-way, Associative, 64KB line size
Cores/chip	2	L1 I-TLB entry	128	L1 I-$	32KB, 8-way, Associative, 64KB line size
ROB entry	96	L1 D-TLB entry	256	L1 D-$	32KB, 8-way, Associative, 64KB line size
RAM	DIMM 4GB, 2 Ranks/DIMM 128M*8/RANK DDR2 SDRAMs				

TABLE II
CONFIGURATION OF THE REAL SERVER WE USED AS THE BASELINE FOR VALIDATIONS.

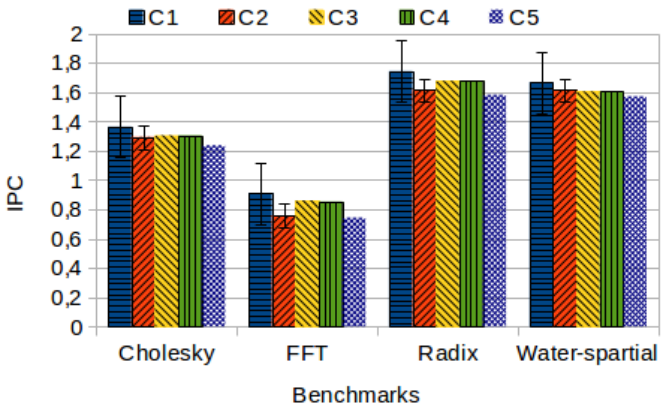

Fig. 3. VMcSim accuracy evaluated using four SPLASH-2 applications: there is an average error of about 2%.

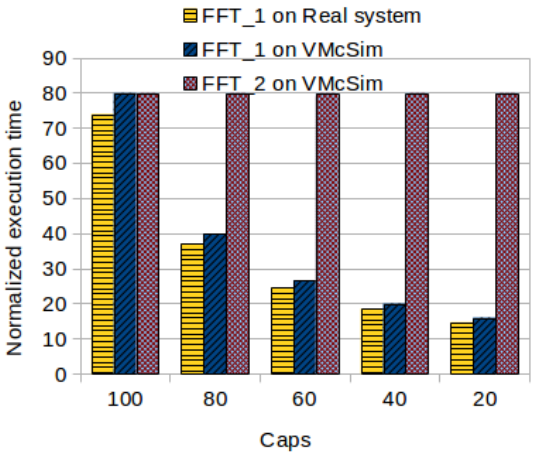

Fig. 4. VMcSim credit scheduler accuracy validated using FTT.

same shape. The slight difference could be explained by the fact that FFT input parameters used in [3] may be different to ours, [3] does not report them.

b) Accuracy of VMcSim: The third experiment type validates the accuracy of VMcSim. Firstly, we repeated the previous experiment with McSimA+ replaced by VMcSim (FFT on VMcSim while varying cache size and associativity), FFT running on top of a VM configured with a single vCPU pinned to a single core. The results of this experiment are close to those presented in the rightmost curve in Fig. 2. This experiment shows that VMcSim behaves like McSimA+ when they run under the same conditions: a single VM with a single vCPU pinned onto a core in VMcSim is similar to a thread pinned onto a core in McSimA+.

Secondly, we performed experiments in the virtualized system and we compared them with VMcSim. For these experiments we used four SPLASH-2 applications. For each experiment, we ran a VM with a single vCPU pinned onto a core. Fig. 3 presents the results of these experiments in four situations: native system (C1), virtualized system (C2), McSimA+ (C3), VMcSim without hypervisor instructions simulated (C4), and finally VMcSim with all its features (C5). We can see that VMcSim highlights virtualization overhead (C1-C2 vs C3-C5), about 3-4% as reported in previous works [4]. Also, VMcSim without hypervisor instructions simulated on cores is equivalent to McSimA+ (C3 vs C4) as mentioned earlier. It introduces the same average error as McSimA+ (2%-3%) when comparing with a real virtualized system.

c) VMcSim credit scheduler accuracy: The fourth experiment type validates the accuracy of VMcSim credit scheduler which claims to be close to Xen credit scheduler. Recall that this scheduler allows to give a ratio of the computation power (called cap) to a VM. For these experiments we ran simultaneously 2 instances of FFT (FFT_1 and FFT_2) on one hand in VMcSim and on the other hand in the virtualized system. The two instances run each other on separated VMs with single vCPU, pinned onto separated cores: the first VM runs FFT_1 while the second one runs FFT_2. We varied the cap of the first VM from 20 to 100 (with 100 means the VM is allowed to use the overall capacity of cores on which its vCPUs are pinned) while it is kept constant (100) for the second VM for all experiments. The objective of these experiments is to check if VMcSim respects the cap, we focus on the first VM (which runs FFT_1). Fig. 4 presents the results of these experiments (execution time for each VM). We can see that VMcSim respects the cap very closely to the original Xen credit scheduler, with an average of error of about 3-4%.

C. Evaluations

The fifth experiment type is dedicated to show how VMcSim can be useful. In this paper we used VMcSim to analyze virtual resources allocation effects in a manycore system.

Workload collocation effects

For these experiments, we only used the microbenchmark. VMcSim was configured to reflect a machine composed of 4 sockets, 4 cores per socket. Cores of the same socket share the same L2 cache but have their own L1 cache. We ran up to four instances of the benchmark simultaneously, each of them on a dedicated VM configured with a single vCPU. We suppose that each VM only needs a quarter of a core (its cap is set to 25). Thus all VMs vCPUs (four in our experiments) can

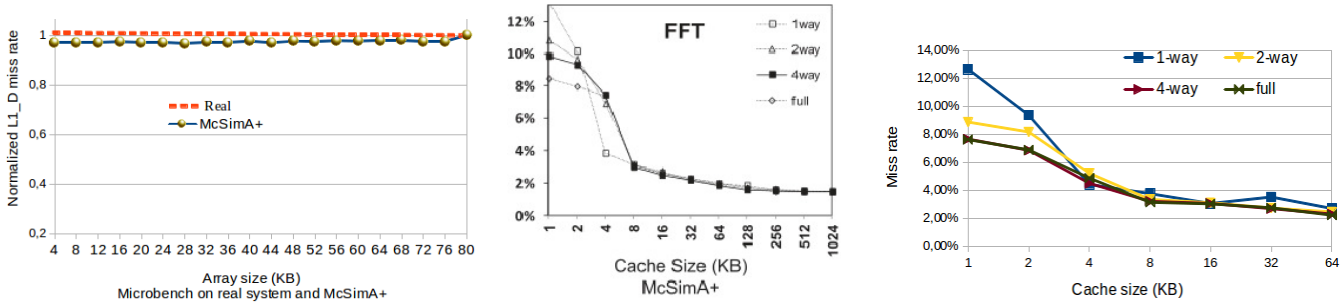

Fig. 2. McSimA+ accuracy evaluated using both the microbenchmark and FFT: there is an average error of about 2%.

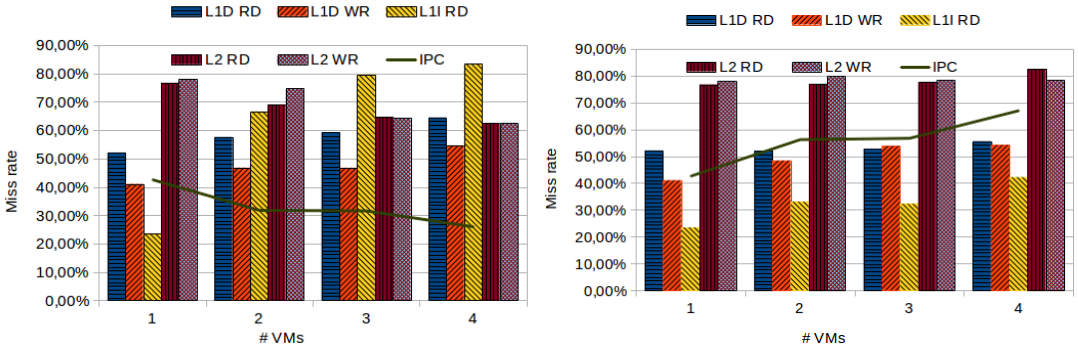

Fig. 5. VMcSim is used to analyze three vCPUs mapping strategies.

theoretically share the same core. We experimented several vCPUs pinning strategies: all the four vCPUs on the same core (S1), one vCPU per core in the same socket (S2), and one vCPU per core per socket (S3). Fig. 5 presents the results of these experiments. According to [8], the higher the cache miss rate, the greater the performance degradation [3]. The first strategy (Fig. 5 left) provides the highest cache miss rate and this rate increases with the number of collocated VM. Notice that L1I RD cache miss rate increases very quickly comparing to others L1 caches. Regarding L2 caches its miss rate is constant, which is normal. For this strategy, performance degradation can be observed by analyzing either L1 cache miss rate or IPC (which decreases). About the second strategy, Fig. 5 right shows that the degradation is not risen than with the first strategy. L2 cache miss rate is the best candidate for analyzing performance degradation in this case. Unlike the first strategy, IPC does not reflect performance degradation. This is conformed to what previous works have reported [7][4]. Finally the last strategy is the best one because its results are similar to those where the application is alone on the host (see the second experiment type, that is the reason we do not present their results here). A lesson one can learn from these experiments is to collocate on the same core only VMs which are not in conflict on the same resource (memory in our case), as concluded by [9]. For example, the user should put together a VM which is CPU bound with a VM which is memory bound.

NUMA effects

In NUMA (Non-Uniform Memory Access) machines, memory accessing times of a program running on a processor depends on the relative distance between that processor and that memory. The cost of accessing non local memory is very high and results to much more performance degradation. Therefore NUMA awareness when pinning vCPUs onto core becomes very important for memory-intensive workloads. To study NUMA awareness effects, we performed experiments with the microbenchmark (configured to use 4KB array size) on VMcSim, configured with 2 sockets and 2 memory controllers (one per socket). We ran the microbenchmark on a VM with a single vCPU, always pinned onto the same core. We evaluated three memory allocation strategies for that VM: (S1) all its memory is allocated on the same memory node, to which belongs its vCPU, (S2) all its memory is allocated on the remote memory node, and (S3) its memory is equitable spread over the two memory nodes.

Fig. 6 presents the results on these experiments. Fig. 6 left presents the amount of data unit that have crossed the NOC (remotely loaded) for each allocation strategy, while Fig. 6 right presents the IPC (which characterizes the efficiency of each strategy). We can see that (S1) is obviously the most efficient strategy, followed by (S3) and (S2).

To limit NUMA effects, most existing virtualization systems such as Xen and VMware make their possible to allocate VM memory in the same location as vCPUs (strategy (S1)). This

[3][8] shows that the cache miss metric is effective in identifying contention for various shared resources.

[4][7] demonstrates that IPC does not always reflect performance and sometime leads to incorrect or misleading interpretation.

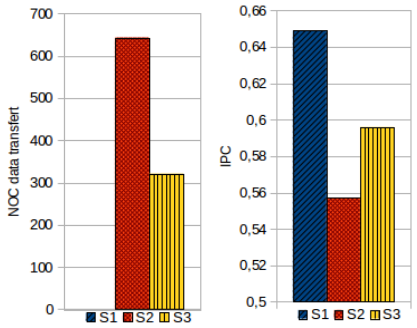

Fig. 6. VMcSim is used to analyze the effects of NUMA systems on virtualized environment.

solution only works when the memory required by the VM can fit within a NUMA memory node. Otherwise, the memory is spread over several nodes (similar to (S3)). A possible solution to face with this situation is the following. Since memory is allocated at VM creation time, the VM is allowed to address all its memory, including the remote memory even if its local memory is not full. We propose an amelioration which consists in modifying the hypervisor so that it forces the VM to only address remote memory when the local memory is used at all. This is a kind of ondemand memory allocation. We validated this solution using VMcSim.

V. Conclusion

In this paper we introduced VMcSim, the first simulation framework which allows to model in detail (at microarchitectural level) a virtualized system. VMcSim simulates all the stack which composes a virtualized system: microarchitectural level (core, cache hierarchy, memory controllers, etc.), hypervisor level (VM memory allocation and vCPUs scheduling), VM level (as operating system), and applications level (a set of threads). Its actual implementation simulates the Xen virtualization system. Based on the microarchitectural level simulator McSimA+, VMcSim is execution driven. It provides a very high level of accuracy (about 2% average of error). Whilst avoiding a full system simulation method, its simulation of the hypervisor allows it to reflect virtualization overhead. Several experiments we performed with a reference benchmark (SPLASH-2) validated that. In addition, we used VMcSim to explore some virtual resources allocation policies in manycore and NUMA systems. Our results are in line with those reported in the literature. As a first step work, VMcSim does not simulate all features provided by virtualization. We are integrating a VM live migration mechanism so that users can study various migration strategies and their impact on applications.

Acknowledgment

This work benefited the support of the French National Research Agency through projects Ctrl-Green (ANR-11-INFR-0012) and FSN Open Cloudware.

References

[1] Michael Armbrust, Armando Fox, Rean Griffith, Anthony D. Joseph, Randy Katz, Andy Konwinski, Gunho Lee, David Patterson, Ariel Rabkin, Ion Stoica, and Matei Zaharia, 'A View of Cloud Computing," Communications of the ACM 2010.

[2] Dejan Milojicic (HP Labs), "High Performance Computing (HPC) in the Cloud," Computing Now journal, September 2012.

[3] Jung Ho Ahn, Sheng Li, O. Seongil, and Norman P. Jouppi, "McSimA+: A manycore simulator with application-level+ simulation and detailed microarchitecture modeling," ISPASS 2013.

[4] Paul Barham, Boris Dragovic, Keir Fraser, Steven Hand, Tim Harris, Alex Ho, Rolf Neugebauer, Ian Pratt, and Andrew Warfield, "Xen and the art of virtualization," SOSP 2003.

[5] Ludmila Cherkasova, Diwaker Gupta, and Amin Vahdat, "Comparison of the Three CPU Schedulers in Xen," SIGMETRICS Performance Evaluation Review, 35(2) 2007.

[6] Chi-Keung Luk, Robert S. Cohn, Robert Muth, Harish Patil, Artur Klauser, P. Geoffrey Lowney, Steven Wallace, Vijay Janapa Reddi, and Kim M. Hazelwood, "Pin: Building Customized Program Analysis Tools with Dynamic Instrumentation," PLDI 2005.

[7] Alaa R. Alameldeen, and David A. Wood, "IPC Considered Harmful for Multiprocessor Workloads," Micro 2006.

[8] Sergey Blagodurov, Sergey Zhuravlev, and Alexandra Fedorova, "Contention-Aware Scheduling on Multicore Systems," TOCS 2010.

[9] Lingjia Tang, Jason Mars, Neil Vachharajani, Robert Hundt, and Mary Lou Soffa, "The impact of memory subsystem resource sharing on datacenter applications," ISCA 2011.

[10] George Kousiouris, Tommaso Cucinotta, and Theodora Varvarigou, "The effects of scheduling, workload type and consolidation scenarios on virtual machine performance and their prediction through optimized artificial neural networks," Journal of Systems and Software 84(8) 2011.

[11] Joydeep Mukherjee, Diwakar Krishnamurthy, Jerry Rolia, and Chris Hyser, "Resource contention detection and management for consolidated workloads," IM 2013.

[12] Xi Chen, Chin Pang Ho, Rasha Osman, Peter G. Harrison, and, William J. Knottenbelt, "Understanding, modelling, and improving the performance of web applications in multicore virtualised environments," ICPE 2014.

[13] Rodrigo N. Calheiros, Rajiv Ranjan, Anton Beloglazov, Cesar A. F. De Rose, Rajkumar Buyya, "CloudSim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms," SPE 41(1) 2011.

[14] Daniel Sanchez and Christos Kozyrakis, 'ZSim: Fast and Accurate Microarchitectural Simulation of Thousand-Core Systems," ISCA 2013.

[15] Rajkumar Buyya and Manzur Murshed, "GridSim: A Toolkit for the Modeling and Simulation of Distributed Resource Management and Scheduling for Grid Computing," CCPE 2002.

[16] Shirley Browne, Jack Dongarra, Nathan Garner, Kevin London, and Philip Mucci, "A scalable cross-platform infrastructure for application performance tuning using hardware counters," SC 2000.

[17] Ruslan Nikolaev and Godmar Back, "Perfctr-xen: a framework for performance counter virtualization," SIGPLAN Notices 2011.

[18] Steven Cameron Woo, Moriyoshi Ohara, Evan Torrie, Jaswinder Pal Singh, and Anoop Gupta, "The splash-2 programs: Characterization and methodological considerations," SIGARCH 1995.

[19] Qasim Ali, Vladimir Kiriansky, osh Simons, and Puneet Zaroo, "Performance evaluation of HPC benchmarks on VMware's ESXi server," Euro-Par 2011.

[20] Hui Lv, Yaozu Dong, Jiangang Duan, and Kevin Tian, "Virtualization challenges: a view from server consolidation perspective," VEE 2012.

[21] Ehsan K. Ardestani and Jose Renau, "ESESC: A Fast Multicore Simulator Using Time-Based Sampling," HPCA 2013.

[22] 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, "Multifacets General Execution-driven Multiprocessor Simulator (GEMS) Toolset," SIGARCH 2005.

[23] Peter S. Magnusson, Magnus Christensson, Jesper Eskilson, Daniel Forsgren, Gustav Hallberg, Johan Hogberg, Fredrik Larsson , Andreas Moestedt, and Bengt Werner, " Simics: A Full System Simulation Platform," Computer 35(2) 2002.