Virtual-machine-based emulation of future generation high-performance computing systems

Patrick G Bridges, Dorian Arnold, Kevin T Pedretti, Madhav Suresh, Feng Lu, Peter Dinda, Russ Joseph and Jack Lange

International Journal of High Performance Computing Applications 2012 26: 125 originally published online 18 March 2012
DOI: 10.1177/1094342012436619

THE INTERNATIONAL JOURNAL of HIGH PERFORMANCE COMPUTING APPLICATIONS

Article

The International Journal of High Performance Computing Applications
26(2) 125–135

DOI: 10.1177/1094342012436619
hpc.sagepub.com
SAGE

Virtual-machine-based emulation of future generation high-performance computing systems

Patrick G Bridges[1], Dorian Arnold[1], Kevin T Pedretti[2], Madhav Suresh[3], Feng Lu[3], Peter Dinda[3], Russ Joseph[3] and Jack Lange[4]

Abstract

This paper describes the design of a system to enable research, development, and testing of new software stacks and hardware features for future high-end computing systems. Motivating uses include both small-scale research and development on simulated individual nodes of proposed high-performance computing systems, and large scaling studies that emulate a sizeable fraction of a future supercomputing system. The proposed architecture combines system virtualization, architectural simulation, time dilation, and slack simulation to provide scalable emulation of hypothetical systems. Virtualization-based full-system measurement and monitoring tools are also included to aid in using the proposed system for co-design of high-performance computing system software and architectural features for future systems. Finally, this paper provides a description of the implementation strategy and status of the system.

Keywords

exascale systems, testbeds, virtualization, operating systems, emulation

1 Introduction

Developing hardware, system software, and applications for next-generation supercomputing systems requires testbeds for investigating new hardware and software features at both the scale of individual nodes and the entire system. Such testbeds allow developers to study the impact of both architectural and software changes on overall application performance and fidelity. Given recent emphasis on hardware/software co-design methodologies, which rely on the continuous evaluation of the impact of hardware, system software, and application changes, these testbeds have become particularly important.

In this paper, we describe our strategy to realizing such testbeds using virtualization-based system emulation. This approach seeks to accelerate the deployment, performance, and utility of testbeds for simulating novel, highly-concurrent architectures, focusing on time-to-result for runs of real applications instead of complete accuracy. To do this, we combine occasional cycle-accurate simulation of key system components with loosely synchronized virtual machine (VM)-based emulation of system hardware features. This is in contrast to past work that focuses *solely* on cycle-accurate node simulations (Bohrer et al. 2004) or high-fidelity cluster-level simulations that rely on skeleton applications or mini-apps to complete simulation runs in reasonable amounts of time (León et al. 2009).

Our approach focuses on the use of a virtual machine monitor (VMM) to emulate individual nodes of the target system, with the software stack under evaluation running as a *guest software stack* in a virtual machine. This allows much of the guest stack to run natively, with the VMM intercepting hardware calls that require additional handling. To achieve this, the VMM coordinates invocation of a linked architectural simulator for architectural features it cannot directly simulate, and controls the passage of time in the guest stack.

[1]Department of Computer Science, University of New Mexico, USA
[2]Scalable System Software Department, Sandia National Laboratories, USA
[3]Department of Electrical Engineering and Computer Science, Northwestern University, USA
[4]Department of Computer Science, University of Pittsburgh, USA

Corresponding author:
Patrick G Bridges, Department of Computer Science, University of New Mexico, Albuquerque, NM 87131, USA
Email: bridges@cs.unm.edu

The VM interfaces with architectural simulation through two different interfaces—a device interface and a processor/system architecture interface. To interface with device simulations such as Structural Simulation Toolkit (SST) device models (Rodrigues et al. 2006), we use a simple host device interface that forwards VM-level memory and I/O port actions to a device simulator for simulation. To interface with cycle-accurate processor simulators such as GEM5 (see http://gem5.org), we leverage the checkpoint-restart features of both VMMs and processor simulators, checkpointing and restoring between the two systems as necessary. Controlling how often hardware features are emulated coarsely in the VM versus simulated in a coupled cycle-accurate simulator provides one mechanism for controlling the accuracy/time-to-solution tradeoff in this system.

The VM manages the passage of time using standard time-dilation techniques (Gupta et al. 2006, 2008), allowing a single node to emulate additional processors and nodes. Unlike past work in this area, we use slack simulation (Chen et al. 2009) to enable the VM to emulate the behavior of low-latency I/O devices such as network interface controllers (NICs). In this approach, distributed VM emulations are coarsely synchronized to a global time source instead of finely synchronized on an operation-by-operation basis. The level of synchronization can also be used to control emulation accuracy versus time-to-solution when evaluating network-intensive high-performance computing (HPC) workloads.

In the remainder of this paper, we first describe several motivating examples that are driving our work in this direction. We then describe the overall architecture of the system we are building, along with selected architectural details from specific portions of the system. Finally, we describe our implementation strategy and status, discuss related work on virtualization-based system emulation, and conclude.

2 Example uses

Our work is driven by several motivating examples of potential relevance to the design of future high-end computing systems which we seek to support with the infrastructure described in this paper. These include architectural changes to processor and memory system design, new networking devices, and novel system use-cases. This section outlines our motivating examples.

2.1 Performance-heterogeneous processors

Many-core systems that include processors with heterogeneous performance characteristics, particularly different clock speeds on different processors, comprise the first kind of system we seek to emulate. Such systems present interesting challenges to both HPC applications and system software design, particularly for examining issues related to application load balancing, node-level resource allocation, and inter-processor communication performance. Enabling development and evaluation of new application, runtime, and system software techniques for these systems is a key motivating factor in the work described in this paper.

2.2 Global addressing

Globally addressable memory in distributed-memory systems is often proposed for deployment in future HPC systems, particularly latency-oriented systems designed to handle large, irregular data sets. Integrating low-latency remote direct memory access (DMA) network devices directly with the hardware memory addressing system could dramatically simplify the system programming model by providing low-latency access to remote memory. It would also avoid the the performance penalties of previous software-based distributed shared memory systems. Because virtualization software can easily intercept virtual and physical memory accesses, global addressing is an ideal use-case for virtualization-based emulation of future large-scale HPC systems.

2.3 Active messaging network interfaces

Active messages are an increasingly important mechanism for low-latency communication in future systems, with recent research demonstrating their usefulness in implementing high-performance distributed graph algorithms (Willcock et al. 2010). New network interface cards are being designed to handle active messages, for example with new control structures or the ability to offload message handlers to the NIC. Evaluation of the performance of these systems on meaningful algorithms and data sets at scale is imperative to understanding the potential benefits and challenges they present. Because no real-world implementations of such cards exist, however, they are another hardware enhancement with impact across the breadth of the software stack that motivates the research described in this paper.

2.4 Many-core processor integration

In the nascent many-core processor era, individual nodes are beginning their transition from tens towards hundreds of cores which will be integrated by scaleable on-chip interconnection networks and distributed shared caches. Furthermore, these cores will have to maximize extremely limited on-chip network bandwidth, cache capacity, chip-wide power budgets, and off-chip memory channels. In many ways, these next-generation processors will resemble small-scale supercomputers and will have to revisit many of the classic large-scale HPC resource-management challenges at the node level. These systems will also raise many questions about how to best exploit local versus global communication patterns to cope with various bottlenecks within the on-chip network, processor to local DRAM channels, and off-chip interprocessor network. Interfacing virtualization-based emulation of a large-scale HPC system

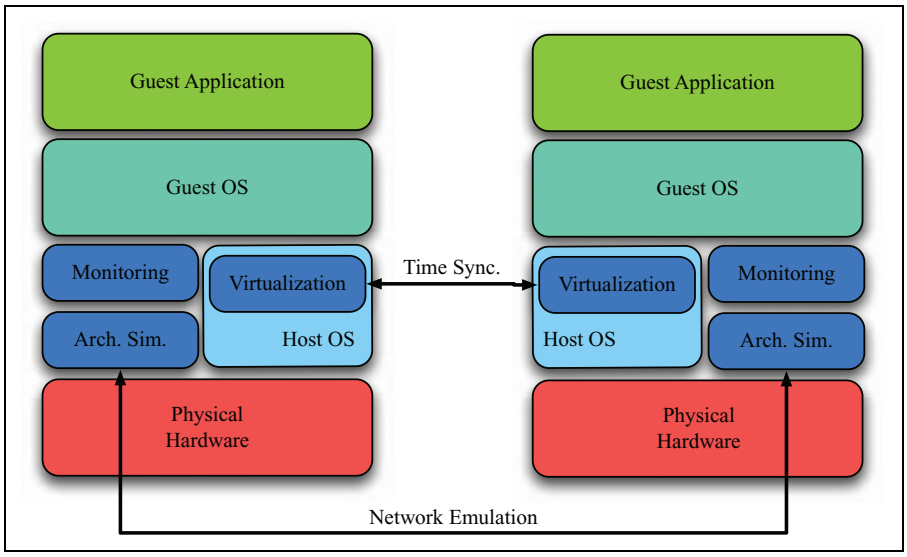

Figure 1. High-level architecture of large-scale VM-based testbed.

with a detailed many-core microarchitecture simulator which models the on-chip network may give some perspective that would otherwise be very difficult to achieve.

2.5 Specialized instruction set architecture features

As the number of transistors on a chip continues to grow, there are increasing opportunities to integrate new programmer-/compiler- and/or operating system-visible features in the instruction set architecture. For example, hardware transactional memory has been proposed as an approach to enhancing the programmability and performance of multicore systems by providing composable transactions instead of locks. Another example is a single-instruction multiple-data (SIMD)/vector instruction set, such as is emerging in the convergence of CPUs and GPUs. Yet another is the loosening of the hardware semantics of instruction execution ordering—a happy medium between fully compiler-based instruction scheduling and fully hardware-based ordering remains to be found. It is our hope that our emulation environment will make it possible to evaluate instruction set architecture extensions in the context of real applications by allowing them to be implemented in the processor simulator yet seamlessly integrated into the hardware-based execution flow of the VMM.

2.6 Exascale on your laptop

Although the primary focus of our work is to investigate the hardware and systems software of future supercomputers, we also envision it helping to broaden the accessibility of those machines. A fully operational supercomputer could equally well be modeled in our system, and operate, albeit slowly and at reduced scale, on cheap commodity hardware that everyone has access to. The system itself is open source and available to all. In such a mode, our system would allow for both application and kernel development and testing in preparation for deployment on the actual supercomputer, or for education.

3 Architecture

We now describe the architecture of our system. We begin with a basic overview of the system architecture, and then provide additional details on the key components of this architecture, including the use of VM monitors, coupling with different types of architectural simulators, our proposed approach to managing the passage of time in the system, and integration with measurement and monitoring tools.

3.1 Overview

Figure 1 shows the general architecture of the system. This system is based on a VMM that intercepts relevant hardware calls from the application and system software being evaluated. The VMM handles these calls to emulate the hardware on which this application/system software is being evaluated, and also provides monitoring and control functionality.

The VMM is the central element in this system. Its primary responsibility is to interact with the guest software stack to provide the illusion that the guest is running on the hardware being emulated. To do this, it intercepts guest software stack hardware accesses when necessary through standard virtualization techniques and performs the following tasks:

- emulate specified processor/system performance by controlling the real and apparent flow of time in the VM;
- invoke external architectural simulation tools to perform detailed simulation of processor features and external devices;

- synchronize the local and remote processor and node clocks to adjust for varying times due to architectural simulation costs;
- provide performance information about the emulated machine to external monitoring tools.

We describe each of these tasks in detail in the remainder of this section.

3.2 VMM-based emulation

In addition to intercepting guest software stack calls and coordinating activity between various system components, the VMM is responsible for general coarse-grained processor emulation functionality. In particular, we will use VMM-based emulation to achieve the following:

1. multiple nodes using a single node;
2. increased numbers of processors on a node;
3. increased and decreased processor speeds on a node.

In each of these cases, well-known past work on VM time-dilation techniques (Gupta et al. 2006, 2008) form the initial basis for our work. Time dilation runs each guest node in a VM that runs at a *fixed* fraction of real time by scheduling the VM less frequently and delivering timer interrupts more or less frequently that real time.

Time dilation is important in our approach because it allows a core or node to emulate more than one node, and provides a virtual global clock source that synchronizes the activities of all nodes in the emulated system. For example, time dilation by a factor of four allows a single node to emulate four hardware nodes or a node with four times as many cores. It also guarantees that time is elapsing at the same rate on all nodes in the system so that causality is preserved in communications between nodes.

Some systems, particularly those with heterogeneous architectural features, cannot be implemented easily using this approach. It may be possible to use external GPUs to simulate more tightly integrated heterogeneous processors, though virtualizing GPUs is already a challenging task. In general, our strategy is to use external architectural features to simulate more diverse architectural features, as described in the following subsection.

3.3 External architectural simulation

Integration with multiple architectural simulators is also a key element of our strategy for providing a testbed for upcoming exascale systems. In particular, we will use both processor simulators such as GEM5 (see http://gem5.org) and device simulators from the SST simulation toolkit (Rodrigues et al. 2006) to broaden the set of architectural features our VM-based testbed can support.

3.3.1 Processor simulation. Periodic processor simulations in cycle-accurate processor simulators such as GEM5 will be used both to evaluate new processor features and to calibrate the speed of VM-based processor emulation as described in Section 4. For evaluating new processor features, the VM will initially be configured to trap and drop into the simulator whenever the guest software stack executes instructions or touches other hardware resources for which processor simulation is required due to a lack of a hardware implementation and/or to make possible detailed performance evaluation. As a run progresses, the speed difference between the host CPU and the simulated guest CPU can also be fed back to the VMM-based processor simulator, allowing the VMM to emulate the performance of the simulated processor more accurately and reduce the simulation slowdown.

3.3.2 Device simulation. Device simulation, particularly of network devices, is also a key element of our virtual testbed strategy. We expect that researchers will extend SST or other system simulators with simulations of proposed network interfaces and networks. Using these simulated devices in the context of VM would allow for their evaluation in the context of an execution environment that could run at the full speed of today's hardware, except when those simulated devices are being used.

Multiple simulated architectural devices will be tied together into a distributed network simulation using an approach similar to that we used in our previous work (León et al. 2009). In particular, we will use Lamport clock-style message timestamps to propagate message transmission and reception times. The global synchronization provided by a time-dilation approach, subject to the complications described in the following subsection, will substitute for the periodic global barrier synchronization used in that approach.

3.4 Distributed slack simulation

The simulation capabilities described above necessitate occasionally pausing simulation of a guest core or node for a relatively substantial length of time. As a result, time in different cores and nodes may occasionally advance at different rates, unlike in traditional time dilation systems. If these differences are not accounted for, simulation accuracy can suffer. For example, if time on node A is progressing at a significantly slower rate than on node B and node A sends a message to node B, the time at which node B receives the message from node A may be incorrect by a large amount. Traditional approaches to addressing this problem, for example optimistic parallel discrete event simulation (Fujimoto 1990), have runtime costs that are potentially expensive and are complex to implement in a virtual machine monitor setting.

We plan to use dilated time simply as a *target* rate at which guest time should advance instead of a fixed rate at which time must advance. Doing so will keep independent cores approximately synchronized while still allowing guests to deviate from dilated time when necessary for

simulation purposes. When such deviation happens, the VM will need to accelerate or decelerate the passage of guest time so that it converges with the target time. This approach trades some accuracy for reduced simulation time, and is a form of slack simulation (Chen et al. 2009), a recent approach for speeding up parallel simulation systems.

Because the operating system (OS) makes assumptions about the accuracy and precision of different timers, converging guest time with target time is a bounded control problem. When the guest is only occasionally monitoring low accuracy timers (e.g. the programmable interrupt timer (PIT) timer), for example, the VM can advance guest time without violating OS assumptions about timing accuracy. However, when the guest is quickly polling high-resolution timers like the time-stamp counter, significant guest time changes would violate such assumptions. By observing guest/timer interactions, the VMM can determine the maximum possible rate at which it can advance or retard guest time without violating the guest's assumptions about timer accuracy and precision.

3.5 Dynamic time dilation

We also plan to explore dynamically adjusting the time dilation factor across nodes, because correctly setting the time-dilation factor is vital for trading off emulation accuracy and time-to-result. For example, if the guest-simulated time deviates by large amounts or diverges from the dilated time, emulation accuracy can suffer, and dilating time further (trading off time-to-result) can be used to improve emulation accuracy. Similarly, if the emulation spends large amounts of time with no VMs to dilate time appropriately, reducing time dilation can improve simulation speed without sacrificing accuracy.

To deal with global effects of changing the time dilation factor, we are exploring gossip-based approaches that include periodic global agreement, similar to our past work on load balancing (Zhu et al. 2009). By including time information in transmitted messages, something that is already necessary for accurate network emulation (see Section 3.3), individual nodes will be able to slowly change their time-dilation factor and stay approximately in sync with the remainder of the simulation. Larger changes in the time-dilation factor that are more likely to lead to de-synchronization of nodes, will still require some form of global agreement.

The goal of this distributed management of time dilation between nodes is to allow nodes making heavy use of simulation features to be more heavily dilated than those that do not. In addition, allowing the time-dilation factor to vary over the course of the run can potentially reduce the time required to complete a full emulation run by allowing emulation to run with less time dilation when less simulation is required.

3.6 VM-based monitoring and analysis

Performance analysis for this emulation framework is necessary for two primary reasons: 1) to help users understand and evaluate the performance of their applications at a micro-level and 2) to help us understand the behavior of the framework itself. To do this, the VMM will provide abstractions and mechanisms for *cross-stack performance monitoring and analysis* to external monitoring tools.

The VMM provides a useful vantage point that makes it possible for profilers to span at least four layers: the application, the OS, the "hardware" interface exported by the VMM, and the actual hardware. This is possible for several reasons. First, many instruction-visible events are naturally intercepted by the VMM (e.g., interrupts) or can be funneled through the VMM via mechanisms such as virtual address translation (e.g., access to a specific memory region). Second, microarchitectural events can be monitored by virtualizing the performance counters. The key challenges are leveraging familiar abstractions for accessing performance data, providing this data to higher level guest OSes or applications, and enabling of non-intrusive analyses.

4 Implementation plan and status

The implementation of our system is in progress. Our work thus far has focused primarily on VMM enhancements to support time dilation and architectural integration, but we have also begun work on other portions of the system. In the remainder of this section, we describe the current state of our design and implementation, as well as planned next steps.

4.1 Virtual machine monitor

We are basing our implementation of the proposed architecture around the Palacios VMM that we have previously developed to support lightweight virtualization in HPC environments (Lange et a;. 2010). Palacios is an HPC-oriented VMM designed to be embedded into a range of different host operating systems, including the lightweight kernels (Riesen et al. 2009), Linux variants potentially including the Cray Linux Environment (Wallace 2007), the MINIX microkernel, and others. Recent work has shown that Palacios can virtualize thousands of nodes of a Cray XT class supercomputer with less that 5% overhead (Lange et al. 2011). Palacios's combination of low overhead on HPC systems and embeddability into traditional HPC operating systems, both lightweight and commodity-based, makes it an ideal platform for our research. Palacios is open source software made available under the BSD license and can be accessed from our project web site, v3vee.org.

4.1.1 Time dilation support. To support time dilation in Palacios, we are augmenting Palacios time management with the necessary scheduling and timer interrupt control features. In particular, to slow down guest time, Palacios uses two quantities:

- **target cycle rate,** the number of cycles that the guest should see execute per *emulated* second of guest time;

- **target time rate,** the time-dilation factor for this guest which determines at what rate timer interrupts are delivered to the guest compared to real time.

At emulation boot time, Palacios uses the sum of the target cycle rates of all of the virtual processors on each host specified to determine the minimum required target time rate for the virtual machines it will emulate. For example, if the VMM must emulate four 3 GHz processors using one 2 GHz core, it sets the minimum required target time rate to 6 $((4 \times 3\ \text{GHz})/2\ \text{GHz})$. Note that the target time rate can be adjusted upward from this point, but cannot go any lower than this minimum.

Given a target cycle rate and target time rate, Palacios then schedules guest cores so that each receives the appropriate number of cycles in each one emulated second. In the example above, Palacios needs to use a 2 GHz processor to give each of four virtual cores three billion cycles in 6 s of real time. In this simple example, that is done simply by giving each core 1/6th of the host processor, but in more complicated cases with higher specified time dilation, Palacios may idle the core periodically so that the correct number of guest cycles elapse for each second of emulated guest time.

One important impact of this is that the guest timestamp counter (TSC) does not have to be completely virtualized; TSC offsetting supported by both Intel VT virtualization and AMD SVM virtualization is sufficient. This is important because virtualizing the TSC is potentially very expensive—full TSC virtualization turns an instruction that takes at worst tens of cycles into one that takes tens of thousands of cycles.

4.1.2 Architectural and slack simulation support. In addition to the time dilation support mentioned above, we have also added the ability to pause, unpause, and synchronize guest time to a provided reference time source to Palacios. In particular, Palacios now keeps track of how often timer events are read or injected into the guest, and uses this information to bound how quickly it offsets guest time towards the desired target time rate. This limits guest-visible timer inaccuracy while still allowing Palacios to control time passage in the guest for slack simulation purposes.

4.1.3 VM–VM communication. For VM-to-VM communication, we are relying on RDMA communication facilities provided by the host OS in which Palacios is embedded, for example Infiniband device support. The low latencies provided by such devices are essential for fast simulation of low-latency network devices, and support for accessing such devices is already being added to Palacios as part of another project.

4.2 General simulator integration

The general structure of our proposed architecture is shown in Figure 2. Palacios already provides mechanisms for hooking specific instructions and regions of guest memory such that they always cause a VM exit, passing control from the guest back to Palacios for handling.

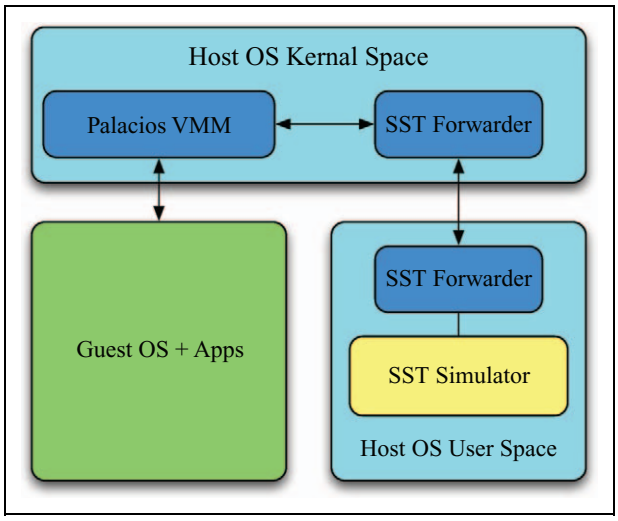

Figure 2. High-level architecture of Palacios VMM and SST architectural simulator integration.

Once Palacios takes control on an exit from the guest processor stack, it forwards information about the system event to the simulator forwarding agent in host OS kernel-space, which then forwards it to the simulator instance running in user-space. After simulation is complete, machine-state updates and timing information returned by the simulator will be used by Palacios to update VM state and control the progression of time in the guest environment.

The primary limitation of this approach is that it limits the extent of what can be simulated. Obviously, if every guest instruction causes a VM exit, performance may be worse than when simulating with SST alone due to the increased overhead. The best situation will be when the vast majority of instructions are executed natively, and only a small percentage are forwarded to SST for handling. We expect that simulating relatively self-contained hardware features such as network interfaces and global address space schemes will demonstrate this behavior and perform well with our approach.

4.3 Host device simulation integration

To support novel HPC hardware devices, we are working on integrating the SST architectural simulator with Palacios. SST is a parallel discrete event simulator that provides a modular framework for constructing hardware device models at various levels of fidelity. SST can be used to simulate large-scale systems, and is itself an MPI program. The downside to SST's software-based approach is performance. Our goal in integrating Palacios with SST is to achieve higher levels of simulation performance by executing most code at near native speed in the hardware-accelerated virtual machine environment, and only passing control to SST when necessary.

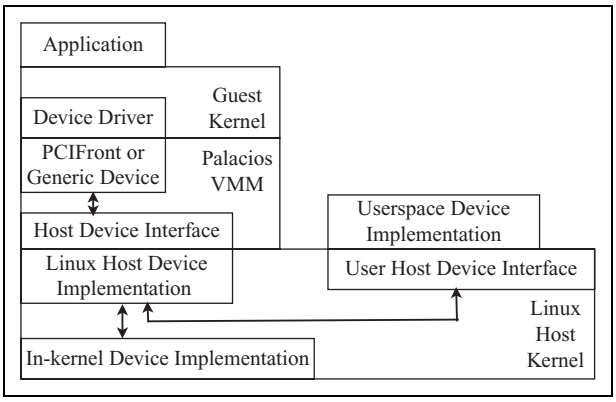

Figure 3. The architecture of host device support in Palacios when embedded into a Linux host kernel.

The integration of simulated network interface and networks into Palacios is supported by its host device framework. Figure 3 illustrates the framework and how it can be used to integrate such simulated devices.

Palacios, like all VMMs, provides virtual devices to its VMs. Normally, these devices are implemented within the Palacios codebase itself. However, Palacios is designed to be independent of the host OS into which it is embedded, and thus simply dropping a previously implemented simulated device into the Palacios codebase may be a challenge. The idea of the host device interface is to make it possible for such a device to be implemented inside of the host operating system instead of Palacios. Thus, for example, in the Linux embedding, a simulated device could be added to the Linux kernel instead of to Palacios. Furthermore, on Linux, we also provide a further interface, the user host device interface, that makes it possible to integrate a simulated device as a Linux userspace program. This latter capability is the likely model that will be used to integrate SST devices.

A host device is made visible to the guest via one of two "front-end" devices, the generic device and the PCIfront device, which are components of Palacios that can be trivially added to a VM via its configuration file. The generic device provides a front-end for legacy (non-PCI) devices. What this means is that reads and writes to device-relevant regions of memory and I/O port space are intercepted by the generic device and are redirected to the host device interface. The PCIfront device acts similarly, but provides additional support for PCI devices, such as allowing for configuration space reads and writes and dynamic remapping of the device-relevant regions of memory and I/O port space. In either case, what is redirected to the host device interface are guest interactions with the device's visible control and status state.

The host device interface redirects these guest interactions to the relevant registered host device implementations, or "back-ends". The implementations can also call to the host device interface to raise interrupts and read and write guest physical memory. The most straightforward host device implementations are those that are implemented directly in the host kernel. In particular, the rendezvous between the back-end implementation and its front-end device is very simple. Rendezvous is based around a URL-like string that encodes the kind of back-end needed and its specific name.

The user host device interface on Linux allows the front-end device to rendezvous with a back-end userspace program that accepts and processes the guest's interactions, and indirectly read/write the guest memory and generate interrupts. In Linux, a Palacios VM is visible as a device in the device namespace (/dev). The userspace process opens this device and then initiates an ioctl to rendezvous with the front-end. Once rendezvous completes, the ioctl returns a file descriptor to the back-end userspace process. This file descriptor is then used to read requests:

- read/write I/O port (I/O port-mapped device state);
- read/write device memory (memory-mapped device state);
- read/write PCI configuration space.

For each request, the back-end returns a response, minimally an acknowledgement. The processing path from the guest action through the front-end device, the host device interface, the user host device interface, the userspace device back-end device implementation, and then back through those interfaces is completely synchronous. As a side-effect of processing a request, or at any other time, the back-end device implementation can also initiate its own requests:

- read/write guest memory;
- raise an interrupt.

These requests operate asynchronously. This combination of semantics and interfaces makes it possible to write a full-fledged device implementation in userspace based around an I/O model as simple as a single select loop, or even a busy-wait loop.

4.4 Processor simulation integration

To support emulation of more complex architectural features, we are also integrating a cycle-accurate processor simulator with Palacios. In the remainder of this section, we provide an overview of our approach to this integration. We then describe in more detail how state is kept consistent between the simulator and the VMM using a checkpointing mechanism, as well as how we will control how often we invoke the simulator from the VMM.

Figure 4. The architecture of checkpoint/restore support in Palacios when embedded into a Linux host kernel. The data path to support migration to and from the GEM5 simulator is illustrated.

4.4.1 Overview. We are in the process of designing and implementing a system to bridge state between the Palacios VMM and the GEM5 architectural simulator (see http://

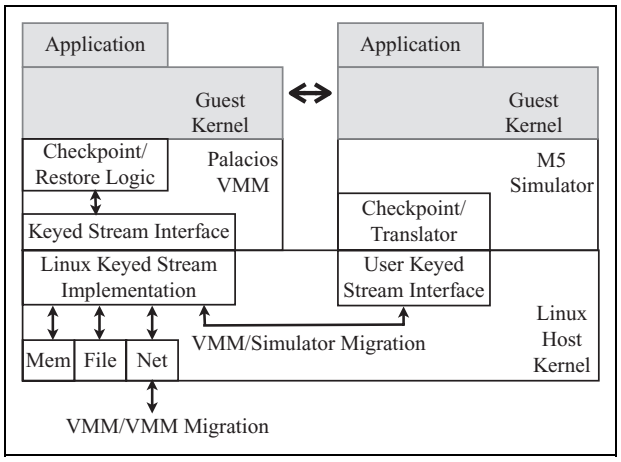

Figure 4. The architecture of checkpoint/restore support in Palacios when embedded into a Linux host kernel. The data path to support migration to and from the GEM5 simulator is illustrated.

gem5.org). The goal of this work is to seamlessly migrate a running guest and its applications between full-speed execution on physical hardware under Palacios to detailed, cycle-accurate simulation under GEM5.

This capability will support not only the high-performance study of next generation supercomputers via SST integration, but also architecture research in general, as GEM5 is not only an integral component of SST, but also an extremely widely used tool in the computer architecture community.

The GEM5 simulation infrastructure combines two long-lived simulation efforts within the architecture community to effectively model current and next-generation computer architectures. The M5 simulator (Binkert et al. 2006) models processor microarchitecture at the detailed level necessary for cycle-accurate, full-system simulation of applications and operating systems. The GEMS toolset (Martin et al. 2005) models future many-core processors which include large shared cache architectures and on-chip interconnection networks in a wide variety of configurations. Independently, these two tools are among the most widely used in the architecture research community. We expect that VMM integration with the merged GEM5 code base will support a wide range of research inquiries spanning the system stack from high-performance computing through compilers, operating systems, and hardware architectures.

4.4.2 Checkpoint-based state migration. The bridge between Palacios and GEM5 is based on the basic checkpoint/restore and migration framework implemented in Palacios. Figure 4 illustrates the framework and shows how GEM5 integration fits into it. Essentially, VM state is checkpointed and restarted between Palacios and GEM5 as necessary, allowing the guest software stack to run in either system as needed. Palacios's *keyed stream* abstraction is used to store checkpointed state to the host, as well as aid in translating state between Palacios's internal VM format and the format needed by the external processor simulator.

Palacios's representation of a VM essentially consists of its memory and architectural state (e.g., registers), the state of attached devices, and architecture-specific state (such as the data structures that interface with Intel or AMD hardware virtualization support). Palacios has been designed so that the architecture-specific state can be restored from the architecture-independent state. That is, the architecture-specific state is soft state. Additionally, memory management state (e.g. nested or shadow page tables) are also soft state. Hence a checkpoint only stores a minimal amount of hard state, dominated by the guest memory contents. Finally, a restore looks very similar to an initial startup of the VM, differing only in the restoration of the hard state.

GEM5's representation of a simulated system is essentially a superset of the VM state maintained by Palacios. As a stand-alone entity, the simulator must maintain virtually the same hard state, namely memory state, register contents, and device state to faithfully capture the functional behavior of the system. In addition, the simulator also holds a large amount of architecture-specific soft state which includes but is not limited to cache states, artifacts of memory management, branch predictors, and pipeline state. Collectively, these items are not strictly required to produce correct functional behavior.

Given the close similarity between the Palacios checkpoint state and the GEM5 checkpoint state, the implementation of a bridge between them boils down to user space code that translates between the two syntaxes. When we want to migrate from Palacios to GEM5, we checkpoint state to a userspace keyed stream implementation corresponding to the translator. Once the stream is closed, the translator writes an equivalent GEM5 checkpoint, and the caller then does an GEM5 resume from that checkpoint. The migration from GEM5 to Palacios is similar.

4.4.3 Soft state reconstruction. While the translation of hard state from Palacios to GEM5 is enough to allow us to correctly simulate a guest system, it cannot reproduce the soft state which would be heavily used and updated during cycle-accurate simulation. There are no direct ways to translate this architecture-specific state from the guest to simulator since much of hardware state including cache/translation lookaside buffer (TLB) state is not accessible to even privileged software. Furthermore, in general, the simulated hardware may differ significantly from the host hardware (e.g. differing cache configurations).

We address this by constructing an internally consistent initial soft state. As GEM5 executes it populates caches, TLBs, and queues, effectively warming up these simulated structures which eventually converge to steady-state behavior. This means that during first instants of resumed simulation GEM5 may not report the same performance that we would expect on a physical implementation. This is unlikely to make a meaningful impact on overall results since we expect that runtime behavior will begin to converge to steady state after a several million cycles (less than

1/1000 of a second of simulated time). We plan to also investigate sampling techniques that may allow the simulator to preload cache state based on access patterns seen during guest execution. This could further reduce simulation warmup time if necessary.

4.4.4 Controlling simulator invocation. In Palacios, it is possible to initiate a migration on any exit. So, for example, suppose we exit on an invalid opcode. The exit handler can determine that the opcode exists in GEM5, and thus migrate to GEM5, which executes the instruction, and then migrates back. As another example, we might use a debug register (or page table manipulation) in Palacios to initiate an exit when execution enters a critical loop in the application, at which point the exit handler could migrate. Finally, random sampling could be done without any Palacios or GEM5 modification, simply by having a user process initiate migrations at random times.

We are currently in the process of enhancing the Palacios/GEM5 bridge with the goal of lowering the migration cost, for example by sharing the guest memory state, supporting migration of individual cores, and allowing callbacks to devices implemented on the opposite side of the bridge.

4.5 Monitoring and analysis

To provide monitoring and analysis support, our baseline approach is to extend the Performance API (PAPI) (Browne et al. 2000) to support VMM performance counters. Our initial focus will be on the high-level PAPI interface that supports simple (start, stop, and read) event measurements. Later, we will include support for PAPI's low-level, programmable interface that supports grouping related events to provide higher-level information. Using this approach means that the myriad of existing PAPI-based tracing, profiling, and analysis tools would become usable with our VMM framework.

Additionally, we will explore mechanisms that perform rudimentary performance analyses. Our approach is to build a framework we call VMM Tuning and Analysis (VTAU) using the TAU (Mohr et al. 1994) framework. The VTAU framework will be used to wrap performance critical components (e.g., functions, code regions, and loops) with tracing or profiling code as appropriate. This will allow us to control the collection of timing and event information mapped to VMM functionality. Our VTAU framework will be able to leverage the feature-rich visualizations of the TAU framework.

5 Related work

Many systems besides the one we propose have been described that trade off complete simulation accuracy for time-to-solution. Functional architectural simulators as opposed to cycle-accurate simulators frequently make this tradeoff. This includes, for example, the functional version of the IBM Mambo simulator (Bohrer et al. 2004).

A number of systems have used techniques similar to the ones we suggest for simulating or emulating large-scale systems. As mentioned above, DieCast's time dilation approach (Gupta et al. 2006, 2008) is closely related to our work, and forms a partial basis for the system we propose. Unlike the system we propose, however, DieCast makes only limited use of architectural simulation, in particular only for high-latency devices such as disk systems. This avoids the time synchronization issues inherent in simulating low-latency devices, but limits DieCast's usefulness in studying the impact of novel low-latency I/O devices in large-scale systems.

Also closely related to the system we propose is past work on cluster-based simulation of cluster systems (León et al. 2009). This system uses network simulation such as we propose in combination with a fine-grained processor simulator, and allows for detailed simulation and analysis of processor and memory system changes not possible in the system we propose. Because of its reliance on cycle-accurate simulation, however, its runtime is bounded by the the runtime of individual node simulations, which can result in slowdowns of several orders of magnitude. As a result, this and similar systems are most appropriate for studying the performance of benchmarks and simplified mini-applications, not full applications such as we seek to study.

6 Conclusions

In this position paper, we have described the architecture of a virtualization-based emulation system. This design is based on the novel combination of a number of existing techniques, including time dilation, slack simulation, and network simulation and emulation, with additional techniques to improve their performance. The resulting system seeks to provide fast, full-scale emulation of future large-scale architectures. Such emulations will aid the development both hardware and software for upcoming exascale class supercomputers.

Funding

This work was supported in part by the DOE Office of Science, Advanced Scientific Computing research (grant number DE-SC0005050) and by a faculty sabbatical appointment from Sandia National Laboratories. Sandia is a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin Company, for the United States Department of Energy under contract DE-AC04-94AL85000. This project was partially supported by the National Science Foundation (grant number CNS-0709168) and the Department of Energy (grant number DE-SC0005343).

References

Binkert N, Beckmann B, Black G, Reinhardt SK, Saidi A, Basu A, et al. (2011) The gem5 simulator. *ACM SIGARCH Computer Architecture News* 39(2): 1–7.

Bohrer P, Elnozahy M, Gheith A, Lefurgy C, Nakra T, Peterson J, et al. (2004) Mambo – a full system simulator for the PowerPC architecture. *ACM SIGMETRICS Performance Evaluation Review* 31(4): 8–12.

Browne S, Dongarra J, Garner N, Ho G and Mucci P (2000) A portable programming interface for performance evaluation on modern processors. *The International Journal of High Performance Computing Applications* 14(3): 189–204.

Chen J, Annavaram M and Dubois M (2009) Exploiting simulation slack to improve parallel simulation speed. In: *38th International Conference on Parallel Processing*, Vienna, Austria, 22–25 September 2009, pp. 371–378. Los Alamitos: IEEE Computer Society.

Fujimoto RM (1990) Parallel discrete event simulation. *Communications of the ACM* 33(10): 30–53.

Gupta D, Vishwanath KV and Vahdat A (2008) Diecast: testing distributed systems with an accurate scale model. In: *5th USENIX Symposium on Networked Systems Design and Implementation*, San Francisco, CA, 16–18 April 2008, pp. 407–422. Berkeley: USENIX Association.

Gupta D, Yocum K, McNett M, Snoeren AC, Vahdat A and Voelker GM (2006) To infinity and beyond: time-warped network emulation. In: *3rd Conference on Networked Systems Design and Implementation*, San Jose, CA 8–10 May 2006, pp. 7–7. Berkeley: USENIX Association.

Lange J, Pedretti K, Dinda P, Bridges PG, Bae C, Soltero P, et al. (2011) Minimal-overhead virtualization of a large scale supercomputer. In: *2011 International Conference on Virtual Execution Environments*, Newport Beach, USA, 9–11 March 2011.

Lange J, Pedretti K, Hudson T, Dinda P, Cui Z, Xia L, et al. (2010) Palacios and Kitten: New high performance operating systems for scalable virtualized and native supercomputing. In: *24th IEEE International Parallel and Distributed Processing Symposium*, Atlanta, USA, 19–23 April 2010.

León EA, Riesen R, Maccabe AB and Bridges PG (2009) Instruction-level simulation of a cluster at scale. In: *2009 International Conference on Supercomputing*, Hamburg, Germany, 23–26 June 2009.

Martin M, Sorin D, Beckmann B, Marty M, Xu M, Alameldeen A, et al. (2005) Multifacet's general execution-driven multiprocessor simulator (gems) toolset. *ACM SIGARCH Computer Architecture News* 33(4): 92–99.

Mohr B, Brown D and Malony AD (1994) TAU: A portable parallel program analysis environment for pC++. In: *1994 Conference on Algorithms and Hardware for Parallel Processing*, Linz, Austria 6–8 September 1994, pp. 29–40. New York: Springer.

Riesen R, Brightwell R, Bridges PG, Hudson T, Maccabe AB, Widener PM, et al. (2009) Designing and implementing lightweight kernels for capability computing. *Concurrency and Computation: Practice and Experience* 21(6): 791–817.

Rodrigues A, Murphy R, Kogge P and Underwood K (2006) The structural simulation toolkit: exploring novel architectures. In: *2006 Conference on Supercomputing*, Tampa, FL, 11–17 November 2006, p. 157. New York: ACM.

Wallace D (2007) Compute Node Linux: Overview, progress to date, and roadmap. In: *2007 Cray User Group Annual Technical Conference*, Seattle, WA, 7–10 May 2007.

Willcock J, Hoefler T, Edmonds N and Lumsdaine A (2010) AM++: A generalized active message framework. In: *19th International Conference on Parallel Architectures and Compilation Techniques*, Vienna, Austria, 11–15 September 2010, pp. 401–410. New York: ACM.

Zhu W, Bridges PG and Maccabe AB (2009) Lightweight application monitoring and tuning with embedded gossip. *IEEE Transactions of Parallel and Distributed Systems* 20(7): 1038–1049.

Author's biographies

Patrick Bridges is an associate professor in the Department of Computer Science at the University of New Mexico. He received his BSc in Computer Science from MIssissippi State University and his PhD in Computer Science from the University of Arizona. His overall research interests are in in system software for large-scale computing systems, particularly large parallel and distributed systems. His recent research in this area has focused on techniques for leveraging virtualization for large-scale systems and on fault tolerance and resilience issues in next-generation supercomputing systems.

Dorian Arnold is an assistant professor in the Department of Computer Science at the University of New Mexico. He holds a BSc in Mathematics and Computer Science from Regis University, a MSc in Computer Science from the University of Tennessee and a PhD in Computer Science from the University of Wisconsin–Madison. His research focuses on high-performance computing (HPC) and extreme-scale distributed systems including scalable software infrastructures, fault-tolerance and scalable tools and services for HPC systems.

Kevin T Pedretti is a senior member of the technical staff at Sandia National Laboratories. His research focuses on operating systems for massively parallel supercomputers, techniques for improving resilience to hardware faults, and high-performance networking. Pedretti has a BSc and an MSc in Electrical and Computer Engineering from the University of Iowa.

Madhav Suresh is an undergraduate at Northwestern University majoring in Computer Science, expecting to graduate in June 2013. His current research interests are in virtualization, operating systems, and HPC.

Feng Lu received a BE degree in Electronic Science and Technology from Tsinghua University, Beijing, China. She is currently working toward the PhD degree in the Department of Electrical Engineering and Computer Science at Northwestern University. Her research involves design and

innovation for reliability-aware computer architecture and high performance architectural simulations.

Peter Dinda is a professor in the Department of Electrical Engineering and Computer Science at Northwestern University, and head of its Computer Engineering and Systems division, which includes 17 faculty members. He holds a BSc in Electrical and Computer Engineering from the University of Wisconsin and a PhD in Computer Science from Carnegie Mellon University. He works in experimental computer systems, particularly parallel and distributed systems. His research currently involves virtualization for distributed and parallel computing, programming languages for parallel computing, programming languages for sensor networks, and empathic systems for bridging individual user satisfaction and systems-level decision-making. You can find out more about him at pdinda.org.

Russ Joseph is an associate professor in the Department of Electrical Engineering and Computer Science at Northwestern University. He holds a PhD in Electrical Engineering from Princeton University and a BSc with majors in Electrical and Computer Engineering and Computer Science from Carnegie Mellon University. His primary research interest is in computer architecture. His work focuses on the design and implementation of power-aware and reliability-aware computer systems. Some of his recent work has examined microprocessor design for reliability and variability tolerance, microarchitecture and compiler technologies to support circuit-level timing speculation, and on-line power management for multi-core systems.

Jack Lange is currently an assistant professor in the Department of Computer Science at the University of Pittsburgh. Prior to joining the faculty at the University of Pittsburgh, he received a BSc, MSc and PhD in Computer Science as well as a BSc in Computer Engineering from Northwestern University. His research specializes in HPC and operating systems, as well as networking, virtualization and distributed systems. His current focus lies in the area of specialized operating systems for supercomputing environments, as well as user-centric data management systems.