Palacios and Kitten: New High Performance Operating Systems For Scalable Virtualized and Native Supercomputing

John Lange*, Kevin Pedretti†, Trammell Hudson†, Peter Dinda*, Zheng Cui‡, Lei Xia*, Patrick Bridges‡,
Andy Gocke*, Steven Jaconette*, Mike Levenhagen†, and Ron Brightwell†
* *Northwestern University, Department of Electrical Engineering and Computer Science*
Email: {jarusl,pdinda,leixia,agocke,jaconette}@northwestern.edu
† *Sandia National Laboratories, Scalable System Software Department*
Email: {ktpedre,mjleven,rbbrigh}@sandia.gov, hudson@osresarch.net
‡ *University of New Mexico, Department of Computer Science*
Email: {zheng,bridges}@cs.unm.edu

***Abstract*—Palacios is a new open-source VMM under development at Northwestern University and the University of New Mexico that enables applications executing in a virtualized environment to achieve scalable high performance on large machines. Palacios functions as a modularized extension to Kitten, a high performance operating system being developed at Sandia National Laboratories to support large-scale supercomputing applications. Together, Palacios and Kitten provide a thin layer over the hardware to support full-featured virtualized environments alongside Kitten's lightweight native environment. Palacios supports existing, unmodified applications and operating systems by using the hardware virtualization technologies in recent AMD and Intel processors. Additionally, Palacios leverages Kitten's simple memory management scheme to enable low-overhead pass-through of native devices to a virtualized environment. We describe the design, implementation, and integration of Palacios and Kitten. Our benchmarks show that Palacios provides near native (within 5%), scalable performance for virtualized environments running important parallel applications. This new architecture provides an incremental path for applications to use supercomputers, running specialized lightweight host operating systems, that is not significantly performance-compromised.**

***Keywords*-virtual machine monitors; lightweight kernels; parallel computing; high performance computing**

I. Introduction

This paper introduces Palacios, a new high performance virtual machine monitor (VMM) architecture, that has been embedded into Kitten, a high performance supercomputing operating system (OS). Together, Palacios and Kitten provide a flexible, high performance virtualized system software platform for HPC systems. This platform broadens the applicability and usability of HPC systems by:

- providing access to advanced virtualization features such as migration, full system checkpointing, and debugging;
- allowing system owners to support a wider range of applications and to more easily support legacy applications and programming models when changing the underlying hardware platform;
- enabling system users to incrementally port their codes from small-scale development systems to large-scale supercomputer systems while carefully balancing their performance and system software service requirements with application porting effort; and
- providing system hardware and software architects with a platform for exploring hardware and system software enhancements without disrupting other applications.

Palacios is a "type-I" pure VMM [1] under development at Northwestern University and the University of New Mexico that provides the ability to virtualize existing, unmodified applications and their operating systems with no porting. Palacios is designed to be embeddable into other operating systems, and has been embedded in two so far, including Kitten. Palacios makes extensive, non-optional use of hardware virtualization technologies and thus can scale with improved implementations of those technologies.

Kitten is an OS being developed at Sandia National Laboratories that is being used to investigate system software techniques for better leveraging multicore processors and hardware virtualization in the context of capability supercomputers. Kitten is designed in the spirit of *lightweight kernels* [2], such as Sandia's Catamount [3] and IBM's CNK [4], that are well known to perform better than commodity kernels for HPC. The simple framework provided by Kitten and other lightweight kernels facilitates experimentation, has led to novel techniques for reducing the memory bandwidth requirements of intra-node message passing [5], and is being used to explore system-level options for improving resiliency to hardware faults.

This project is made possible by support from the National Science Foundation (NSF) via grants CNS-0709168, CNS-0707365, and the Department of Energy (DOE) via a subcontract from Oak Ridge National Laboratory on grant DE-AC05-00OR22725. John Lange was partially supported by a Symantec Research Labs Fellowship. Sandia is a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin Company, for the United States Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.

Kitten and Palacios together provide a scalable, flexible HPC system software platform that addresses the challenges laid out earlier and by others [6]. Applications ported to Kitten will be able to achieve maximum performance on a given machine. Furthermore, Kitten is itself portable and open, propagating the benefits of such porting efforts to multiple machines. Palacios provides the ability to run existing, unmodified applications and their operating systems, requiring no porting. Furthermore, as Palacios has quite low overhead, it could potentially be used to manage a machine, allowing a mixture of workloads running on commodity and more specialized OSes, and could even run ported applications on more generic hardware.

Palacios and Kitten can be used separately or together, and run on a variety of machines ranging from commodity clusters and servers to large scale parallel machines at Sandia. Both Palacios and Kitten are open source tools that are currently available to download and use.

In the remainder of this paper, we describe the design and implementation of both Palacios and Kitten, and evaluate their performance. The core contributions of this paper are the following:

- We introduce and describe the Palacios VMM.
- We introduce and describe the Kitten HPC OS.
- We show how the combination of Palacios and Kitten can provide an incremental path to using many different kinds of HPC resources for the mutual benefit of users and machine owners.
- We show that an integrated virtualization system combining Palacios and Kitten can provide nearly native performance for existing codes, even when extensive communication is involved.
- We present evaluations of parallel application and benchmark performance and overheads using virtualization on high-end computing resources. The overheads we see, particularly using hardware nested paging, are typically less than 5%.

II. Motivation

Palacios and Kitten are parts of larger projects that have numerous motivations. Here we consider their joint motivation in the context of high performance computing, particularly on large scale machines.

Maximizing performance through lightweight kernels: Lightweight compute node OSes maximize the resources delivered to applications to maximize their performance. As such, a lightweight kernel does not implement much of the functionality of a traditional operating system; instead, it provides mechanisms that allow system services to be implemented *outside* the OS, for example in a library linked to the application. As a result, they also require that applications be carefully ported to their minimalist interfaces.

Increasing portability and compatibility through commodity interfaces: Standardized application interfaces, for example partial or full Linux ABI compatibility, would make it easier to port parallel applications to a lightweight kernel. However, a lightweight kernel cannot support the full functionality of a commodity kernel without losing the benefits noted above. This means that some applications cannot be run without modification.

Achieving full application and OS compatibility through virtualization: Full system virtualization provides full compatibility at the hardware level, allowing existing unmodified applications and OSes to run. The machine is thus immediately available to be used by any application code, increasing system utilization when ported application jobs are not available. The performance of the full system virtualization implementation (the VMM) partially drives the choice of either using the VMM or porting an application to the lightweight kernel. Lowering the overhead of the VMM, particularly in communication, allows more of the workload of the machine to consist of VMMs.

Preserving and enabling investment in ported applications through virtualization: A VMM which can run a lightweight kernel provides straightforward portability to applications where the lightweight kernel is not available natively. Virtualization makes it possible to emulate a large scale machine on a small machine, desktop, or cluster. This emulation ability makes commodity hardware useful for developing and debugging applications for lightweight kernels running on large scale machines.

Managing the machine through virtualization: Full system virtualization would allow a site to dynamically configure nodes to run a full OS or a lightweight OS without requiring rebooting the whole machine on a per-job basis. Management based on virtualization would also make it possible to backfill work on the machine using loosely-coupled programming jobs [7] or other low priority work. A batch-submission or grid computing system could be run on a collection of nodes where a new OS stack could be dynamically launched; this system could also be brought up and torn down as needed.

Augmenting the machine through virtualization: Virtualization offers the option to enhance the underlying machine with new capabilities or better functionality. Virtualized lightweight kernels can be extended at runtime with specific features that would otherwise be too costly to implement. Legacy applications and OSes would be able to use features such as migration that they would otherwise be unable to support. Virtualization also provides new opportunities for fault tolerance, a critical area that is receiving more attention as the mean time between system failures continues to decrease.

Enhancing systems software research in HPC and elsewhere: The combination of Palacios and Kitten provides an open source toolset for HPC systems software research that

can run existing codes without the need for victim hardware. Palacios and Kitten enable new systems research into areas such as fault-tolerant system software, checkpointing, overlays, multicore parallelism, and the integration of high-end computing and grid computing.

III. PALACIOS

Palacios[1] is an OS independent VMM designed as part of the the V3VEE project (http://v3vee.org). Palacios currently targets the x86 and x86_64 architectures (hosts and guests) and is compatible with both the AMD SVM [8] and Intel VT [9] extensions. Palacios supports both 32 and 64 bit host OSes as well as 32 and 64 bit guest OSes[2]. Palacios supports virtual memory using either shadow or nested paging. Palacios implements full hardware virtualization while providing targeted paravirtualized extensions.

Palacios is a fully original VMM architecture developed at Northwestern University. Figure 1 shows the scale of Palacios as of the 1.1 release (and the Kitten 1.1.0 release). Note that the Palacios core is quite small. The entire VMM, including the default set of virtual devices is on the order of 28 thousand lines of C and assembly. The combination of Palacios and Kitten is 89 thousand lines of code. In comparison, Xen 3.0.3 consists of almost 580 thousand lines of which the hypervisor core is 50–80 thousand lines, as measured by the `wc` tool. Similarly, Kernel Virtual Machine (KVM) is massive when its Linux kernel dependencies are considered (a performance comparison with KVM is given in Section VI-G). Palacios is publicly available from http://v3vee.org, with additional documentation about its theory of operation available in a technical report [10]. Palacios is released under a BSD license.

Palacios supports multiple physical host and virtual guest environments. Palacios is compatible with both AMD SVM and Intel VT architectures, and has been evaluated on commodity Ethernet based servers, a high end Infiniband cluster, as well as Red Storm development cages consisting of Cray XT nodes. Palacios also supports the virtualization of a diverse set of guest OS environments, including commodity Linux and other OS distributions, modern Linux kernels, and several lightweight HPC OSes such as CNL [11], Catamount [3], and Kitten itself.

A. Architecture

Palacios is an OS independent VMM, and as such is designed to be easily portable to diverse host operating systems. Currently Palacios supports multiple operating systems, but specifically supports Kitten for high performance environments. Palacios integrates with a host OS through a minimal and explicitly defined functional interface that the host OS is responsible for supporting. Furthermore, the interface is modularized so that a host environment can decide its own level of support and integration. Less than 300 lines of code needed to be written to embed Palacios into Kitten. Palacios's architecture, shown in Figure 2, is designed to be internally modular and extensible and provides common interfaces for registering event handlers for common operations.

[1]Palacios, TX is the "Shrimp Capital of Texas."

[2]64 bit guests are only supported on 64 bit hosts

Component	Lines of Code
Palacios	
Palacios Core (C+Assembly)	15,084
Palacios Virtual Devices (C)	8,708
Total	28,112
Kitten	
Kitten Core (C)	17,995
Kitten Arch Code (C+Assembly)	14,604
Misc. Contrib Code (Kbuild/lwIP)	27,973
Palacios Glue Module (C)	286
Total	60,858
Grand Total	88,970

Figure 1. Lines of code in Palacios and Kitten as measured with the SLOCCount tool.

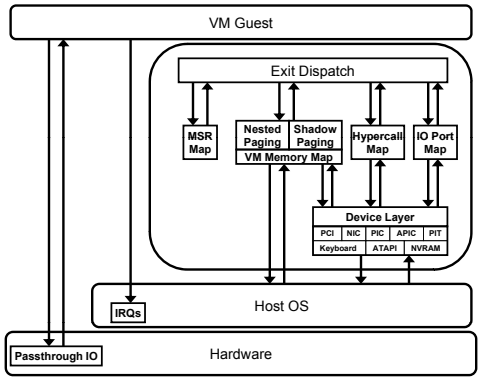


Figure 2. Palacios architecture.

Configurability: Palacios is designed to be highly modular to support the generation of specialized VMM architectures. The modularity allows VMM features and subsystems to be selected at compile time to generate a VMM that is specific to the target environment. The configuration system is also used to select from the set of available OS interfaces, in order to enable Palacios to run on a large number of OS architectures. The build and compile time configuration system is based on a modified version of KBuild ported from Linux.

Palacios also includes a runtime configuration system that allows guest environments to specifically configure the VMM to suit their environments. Virtual devices are implemented as independent modules that are inserted into a runtime generated hash table that is keyed to a device's

ID. The guest configuration also allows a guest to specify core configuration options such as the scheduling quantum and the mechanism used for shadow memory.

The combination of the compile time and runtime configurations make it possible to construct a wide range of guest environments that can be targeted for a large range of host OS and hardware environments.

Resource hooks: The Palacios core provides an extensive interface to allow VMM components to register to receive and handle guest and host events. Guest events that can be hooked include accesses to MSRs, IO ports, specific memory pages, and hypercalls. Palacios also includes functionality to receive notifications of host events such as general interrupts, keystrokes and timer ticks.

Palacios interfaces with the host OS through a small set of function hooks that the host OS is required to provide. These functions include methods for allocating and freeing physical memory pages as well as heap memory, address conversion functions for translating physical addresses to the VMMs virtual address space, a function to yield the CPU when a VM is idle, and an interface for interfacing with the host's interrupt handling infrastructure. In addition to this interface, Palacios also includes an optional socket interface that consists of a small set of typical socket functions.

Interrupts: Palacios includes two models for hardware interrupts, passthrough interrupts and specific event notifications. Furthermore, Palacios is capable of disabling local and global interrupts in order to have interrupt processing on a core run at times it chooses. The interrupt method used is determined by the virtual device connected to the guest.

For most virtual devices, interrupts are delivered via a host event notification interface. This interface requires the presence of a host OS device driver to handle the interrupt and transfer any data to or from the device. The data from the device operation is then encapsulated inside a host event and delivered to Palacios. The event is then delivered to any virtual devices listening on the notification channel. The virtual device is then responsible for raising virtual interrupts as needed.

For high performance devices, such as network cards, Palacios supports passthrough operation which allows the guest to interact directly with the hardware. For this mechanism no host OS driver is needed. In this case, Palacios creates a special virtual passthrough device that interfaces with the host to register for a given device's interrupt. The host OS creates a generic interrupt handler that first masks the interrupt pin, acks the interrupt to the hardware interrupt controller, and then raises a virtual interrupt in Palacios. When the guest environment acks the virtual interrupt, Palacios notifies the host, which then unmasks the interrupt pin. This interface allows direct device IO to and from the guest environment with only a small increase to the interrupt latency that is dominated by the hardware's world context switch latency.

B. Palacios as a HPC VMM

Part of the motivation behind Palacios's design is that it be well suited for high performance computing environments, both on the small scale (e.g., multicores) and large scale parallel machines. Palacios is designed to interfere with the guest as little as possible, allowing it to achieve maximum performance.

Palacios is currently designed for distributed memory parallel computing environments. This naturally maps to conventional cluster and HPC architectures. Multicore CPUs are currently virtualized as a set of independent compute nodes that run separate guest contexts. Support for single image multicore environments (i.e., multicore guests) is currently under development.

Several aspects of Palacios's design are suited for HPC:

- Minimalist interface: Palacios does not require extensive host OS features, which allows it to be easily embedded into even small kernels, such as Kitten and GeekOS [12].
- Full system virtualization: Palacios does not require guest OS changes. This allows it to run existing kernels without any porting, including Linux kernels and whole distributions, and lightweight kernels [2] like Kitten, Catamount, Cray CNL [11] and IBM's CNK [4].
- Contiguous memory preallocation: Palacios preallocates guest memory as a physically contiguous region. This vastly simplifies the virtualized memory implementation, and provides deterministic performance for most memory operations.
- Passthrough resources and resource partitioning: Palacios allows host resources to be easily mapped directly into a guest environment. This allows a guest to use high performance devices, with existing device drivers, with no virtualization overhead.
- Low noise: Palacios minimizes the amount of OS noise [13] injected by the VMM layer. Palacios makes no use of internal timers, nor does it accumulate deferred work.
- Extensive compile time configurability: Palacios can be configured with a minimum set of required features to produce a highly optimized VMM for specific environments. This allows lightweight kernels to include only the features that are deemed necessary and remove any overhead that is not specifically needed.

IV. KITTEN

Kitten is an open-source OS designed specifically for high performance computing. It employs the same "lightweight" philosophy as its predecessors—SUNMOS, Puma, Cougar, and Catamount[3]—to achieve superior scalability on massively parallel supercomputers while at the same time expos-

[3]The name Kitten continues the cat naming theme, but indicates a new beginning.

ing a more familiar and flexible environment to application developers, addressing one of the primary criticisms of previous lightweight kernels. Kitten provides partial Linux API and ABI compatibility so that standard compiler tool-chains and system libraries (e.g., Glibc) can be used without modification. The resulting ELF executables can be run on either Linux or Kitten unchanged. In cases where Kitten's partial Linux API and ABI compatibility is not sufficient, the combination of Kitten and Palacios enables unmodified guest OSes and applications to be loaded on demand.

The general philosophy being used to develop Kitten is to borrow heavily from the Linux kernel when doing so does not compromise scalability or performance (e.g., adapting the Linux bootstrap code for Kitten). Performance critical subsystems, such as memory management and task scheduling, are replaced with code written from scratch for Kitten. To avoid potential licensing issues, no code from prior Sandia-developed lightweight kernels is used. Like Linux, the Kitten code base is structured to facilitate portability to new architectures. Currently only the x86_64 architecture is officially supported, but NEC has recently ported Kitten to the NEC SX vector architecture for research purposes[14]. Kitten is publicly available from http://software.sandia.gov/trac/kitten and is released under the terms of the GNU Public License (GPL) version 2.

A. Architecture

Kitten (Figure 3) is a monolithic kernel that runs symmetrically on all processors in the system. Straightforward locking techniques are used to protect access to shared data structures. At system boot-up, the kernel enumerates and initializes all hardware resources (processors, memory, and network interfaces) and then launches the initial user-level task, which runs with elevated privilege (the equivalent of root). This process is responsible for interfacing with the outside world to load jobs onto the system, which may either be native Kitten applications or guest operating systems. The Kitten kernel exposes a set of resource management system calls that the initial task uses to create virtual address spaces, allocate physical memory, create additional native Kitten tasks, and launch guest operating systems.

The Kitten kernel supports a subset of the Linux system call API and adheres to the Linux ABI to support native user-level tasks. Compatibility includes system call calling conventions, user-level stack and heap layout, thread-local storage conventions, and a variety of standard system calls such as `read()`, `write()`, `mmap()`, `clone()`, and `futex()`. The subset of system calls that Kitten implements natively is intended to support the requirements of existing scalable scientific computing applications in use at Sandia. The subset is also sufficient to support Glibc's NPTL POSIX threads implementation and GCC's OpenMP implementation without modification. Implementing additional system calls is a relatively straightforward process.

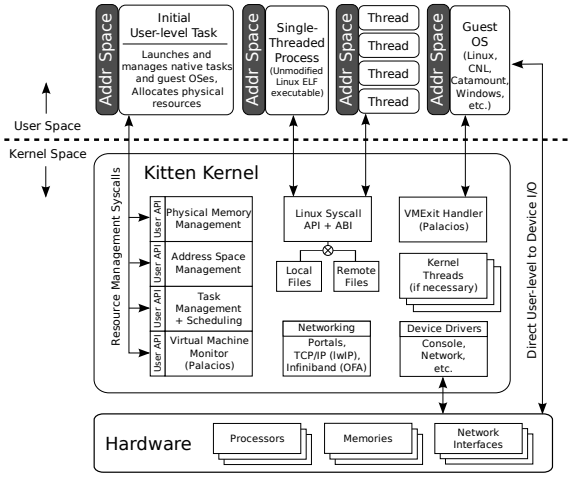


Figure 3. Kitten architecture.

The Kitten kernel contains functionality aimed at easing the task of porting of Linux device drivers to Kitten. Many device drivers and user-level interface libraries create or require local files under `/dev`, `/proc`, and `/sys`. Kitten provides limited support for such files. When a device driver is initialized, it can register a set of callback operations to be used for a given file name. The `open()` system call handler then inspects a table of the registered local file names to determine how to handle each open request. Remote files are forwarded to a user-level proxy task for servicing. Kitten also provides support for kernel threads, interrupt registration, and one-shot timers since they are required by many Linux drivers. The Open Fabrics Alliance (OFA) Infiniband stack was recently ported to Kitten without making any significant changes to the OFA code.

B. Memory Management

Unlike traditional general-purpose kernels, Kitten delegates most virtual and physical memory management to user-space. The initial task allocates memory to new native applications and Palacios virtual machines by making a series of system calls to create an address space, create virtual memory regions, and bind physical memory to those regions. Memory topology information (i.e., NUMA) is provided to the initial-task so it can make intelligent decisions about how memory should be allocated.

Memory is bound to a context of execution before it starts executing and a contiguous linear mapping is used between virtual and physical addresses. The use of a regular mapping greatly simplifies virtual to physical address translation compared to demand-paged schemes, which result in an unpredictable mapping with complex performance implications. Networking hardware and software can take advantage of the simple mapping to increase performance (which is the case on Cray XT) and potentially decrease cost by eliminating the need for translation table memory and table walk hardware on the network interface. The

simple mapping also enables straightforward pass-through of physical devices to para-virtualized guest drivers.

C. Task Scheduling

All contexts of execution on Kitten, including Palacios virtual machines, are represented by a task structure. Tasks that have their own exclusive address space are considered processes and tasks that share an address space are threads. Processes and threads are identical from a scheduling standpoint. Each processor has its own run queue of ready tasks that are preemptively scheduled in a round-robin fashion. Currently Kitten does not automatically migrate tasks to maintain load balance. This is sufficient for the expected common usage model of one MPI task or OpenMP thread per processor.

The privileged initial task that is started at boot time allocates a set of processors to each user application task (process) that it creates. An application task may then spawn additional tasks on its set of processors via the `clone()` system call. By default spawned tasks are spread out to minimize the number of tasks per processor but a Kitten-specific task creation system call can be used to specify the exact processor that a task should be spawned on.

V. Integrating Palacios and Kitten

Palacios was designed to be easily integrated with different operating systems. This leads to an extremely simple integration with Kitten consisting of an interface file of less than 300 lines of code. The integration includes no internal changes in either Kitten or Palacios, and the interface code is encapsulated with the Palacios library in an optional compile time module for Kitten. This makes Palacios a natural virtualization solution for Kitten when considered against existing solutions that target a specific OS with extensive dependencies on internal OS infrastructures.

Kitten exposes the Palacios control functions via a system call interface available from user space. This allows user level tasks to instantiate virtual machine images directly from user memory. This interface allows VMs to be loaded and controlled via processes received from the job loader. A VM image can thus be linked into a standard job that includes loading and control functionality.

SeaStar Passthrough Support: Because Palacios provides support for passthrough I/O, it is possible to support high performance, partitioned access to particular communication devices. We do this for the SeaStar communication hardware on the Red Storm machine. The SeaStar is a high performance network interface that utilizes the AMD HyperTransport Interface and proprietary mesh interconnect for data transfers between Cray XT nodes [15]. At the hardware layer the data transfers take the form of arbitrary physical-addressed DMA operations. To support a virtualized SeaStar the physical DMA addresses must be translated from the guest's address space. However, to ensure high performance the SeaStar's command queue must be directly exposed to the guest. This requires the implementation of a simple high performance translation mechanism. Both Palacios and Kitten include a simple memory model that makes such support straightforward.

The programmable SeaStar architecture provides several possible avenues for optimizing DMA translations. These include a self-virtualizable firmware as well as an explicitly virtualized guest driver. In the performance study we conducted for this paper we chose to modify the SeaStar driver running in the guest to support Palacios's passthrough I/O. This allows the guest to have exclusive and direct access to the SeaStar device. Palacios uses the large contiguous physical memory allocations supported by Kitten to map contiguous guest memory at a known offset. The SeaStar driver has a tiny modification that incorporates this offset into the DMA commands sent to the SeaStar. This allows the SeaStar to execute actual memory operations with no performance loss due to virtualization overhead. Because each Cray XT node contains a single SeaStar device, the passthrough configuration means that only a single guest is capable of operating the SeaStar at any given time.

Besides memory-mapped I/O, the SeaStar also directly uses an APIC interrupt line to notify the host of transfer completions as well as message arrivals. Currently, Palacios exits from the guest on all interrupts. For SeaStar interrupts, we immediately inject such interrupts into the guest and resume. While this introduces an VM exit/entry cost to each SeaStar interrupt, in practice this only results in a small increase in latency. We also note that the SeaStar interrupts are relatively synchronized, which does not result in a significant increase in noise. We are investigating the use of next generation SVM hardware that supports selective interrupt exiting to eliminate this already small cost.

While implicitly trusting guest environments to directly control DMA operations is not possible in normal environments, the HPC context allows for such trust.

Infiniband and Ethernet Passthrough Support: Our integration of Palacios and Kitten also includes an implementation of passthrough I/O for Mellanox Infiniband NICs and Intel E1000 Ethernet NICs. These are similar to the SeaStar implementation.

VI. Performance

We conducted a careful performance evaluation of the combination of Palacios and Kitten on diverse hardware, and at scales up to 48 nodes. We focus the presentation of our evaluation on the Red Storm machine and widely recognized applications/benchmarks considered critical to its success. As far as we are aware, ours is the largest scale evaluation of parallel applications/benchmarks in virtualization to date, particularly for those with significant communication. It also appears to be the first evaluation on petaflop-capable hardware. Finally, we show performance numbers for native

lightweight kernels, which create a very high bar for the performance of virtualization. The main takeaways from our evaluation are the following.

1) The combination of Palacios and Kitten is generally able to provide near-native performance. This is the case even with large amounts of complex communication, and even when running guest OSes that themselves use lightweight kernels to maximize performance.
2) It is generally preferable for a VMM to use nested paging (a hardware feature of AMD SVM and Intel VT) over shadow paging (a software approach) for guest physical memory virtualization. However, for guest OSes that use simple, high performance address space management, such as lightweight kernels, shadow paging can sometimes be preferable due to its being more TLB-friendly.

The typical overhead for virtualization is $\leq 5\%$.

A. Testbed

We evaluated the performance and scaling of Palacios running on Kitten on the development system *rsqual*, part of the Red Storm machine at Sandia National Laboratories. Each XT4 node on this machine contains a quad-core AMD Budapest processor running at 2.2 GHz with 4 GB of RAM. The nodes are interconnected with a Cray SeaStar 2.2 mesh network [15]. Each node can simultaneously send and receive at a rate of 2.1 GB/s via MPI. The measured node to node MPI-level latency ranges from 4.8 μsec (using the Catamount [3] operating system) to 7.0 μsec (using the native CNL [11] operating system). As we stated earlier, even though we can run multiple guests on a multicore Cray XT node by instantiating them on separate cores, our current implementation only allows the SeaStar to be exposed to a single guest context. Due to this limitation, our performance evaluation is restricted to a single guest per Cray XT node.

In addition, we used two dual-socket quad-core 2.3 GHz AMD Shanghai systems with 32GB of memory for communication benchmark testing on commodity HPC hardware. Nodes in this system are connected with Mellanox ConnectX QDR Infiniband NICs and a Mellanox Infiniscale-IV 24 port switch. When not running Kitten, these systems run Linux 2.6.27 and the OpenFabrics 1.4 Infiniband stack.

All benchmark timing in this paper is done using the AMD cycle counter. When virtualization is used, the cycle counter is direct mapped to the guest and not virtualized. Every benchmark receives the same accurate view of the passage of real time regardless of whether virtualization is in use or not.

B. Guests

We evaluated Palacios running on Kitten with two guest environments:

- Cray Compute Node Linux (CNL). This is Cray's stripped down Linux operating system customized for Cray XT hardware. CNL is a minimized Linux (2.6 kernel) that leverages BusyBox [16] and other embedded OS tools/mechanism. This OS is also known as Unicos/LC and the Cray Linux Environment (CLE).
- Catamount. Catamount is a lightweight kernel descended from the SUNMOS and PUMA operating systems developed at Sandia National Labs and the University of New Mexico [17][18]. These OSes, and Catamount, were developed, from-scratch, in reaction to the heavyweight operating systems for parallel computers that began to proliferate in the 1990s. Catamount provides a simple memory model with a physically-contiguous virtual memory layout, parallel job launch, and message passing facilities.

We also use Kitten as a guest for our Infiniband tests. It is important to note that Palacios runs a much wider range of guests than reported in this evaluation. Any modern x86 or x86_64 guest can be booted.

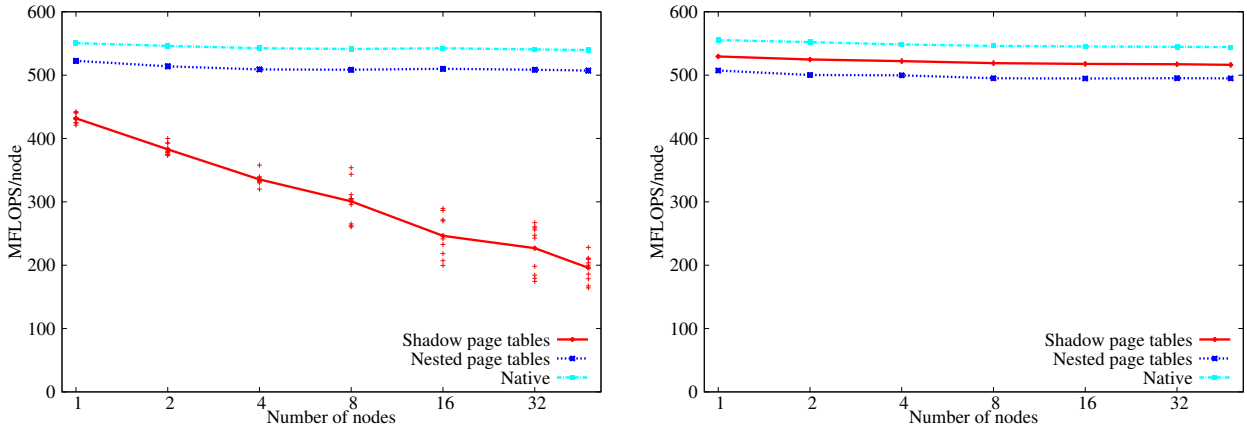
C. HPCCG Benchmark Results

We used the HPCCG benchmark to evaluate the impact of virtualization on application performance and scaling. HPCCG [19] is a simple conjugate gradient solver that represents an important workload for Sandia. It is commonly used to characterize the performance of new hardware platforms that are under evaluation. The majority of its runtime is spent in a sparse matrix-vector multiply kernel.

We ran HPCCG on top of CNL and Catamount on Red Storm, considering scales from 1 to 48 nodes. A fixed-size problem per node was used to obtain these results. The specific HPCCG input arguments were "100 100 100", requiring approximately 380 MB per node. This software stack was compiled with the Portland Group pgicc compiler version 7, and was run both directly on the machine and on top of Palacios. Both shadow paging and nested paging cases were considered. Communication was done using the passthrough-mapped SeaStar interface, as described earlier.

Figures 4(a) and 4(b) show the results for CNL and Catamount guests. Each graph compares the performance and scaling of the native OS, the virtualized OS with shadow paging, and the virtualized OS with nested paging. The graph shows both the raw measurements of multiple runs and the averages of those runs. The most important result is that the overhead of virtualization is less than 5% and this overhead remains essentially constant at the scales we considered, despite the growing amount of communication. Note further that the variance in performance for both native CNL and virtualized CNL (with nested paging) is minuscule and independent of scale. For Catamount, all variances are tiny and independent, even with shadow paging.

The figure also illustrates the relative effectiveness of Palacios's shadow and nested paging approaches to virtu-



(a) CNL Guest

(b) Catamount Guest

Figure 4. HPCCG benchmark comparing scaling for virtualization with shadow paging, virtualization with nested paging, and no virtualization. Palacios/Kitten can provide scaling to 48 nodes with less than 5% performance degradation.

alizing memory. Clearly, nested paging is preferable for this benchmark running on a CNL guest, both for scaling and for low variation in performance. There are two effects at work here. First, shadow paging results in more VM exits than nested paging. On a single node, this overhead results in a 13% performance degradation compared to native performance. The second effect is that the variance in single node performance compounds as we scale, resulting in an increasing performance difference.

Surprisingly, shadow paging is slightly preferable to nested paging for the benchmark running on the Catamount guest. In Catamount the guest page tables change very infrequently, avoiding the exits for shadow page table refills that happen with CNL. Additionally, instead of the deep nested page walk ($O(nm)$ for n-deep guest and m-deep host page tables) needed on a TLB miss with nested pages, only a regular m-deep host page table walk occurs on a TLB miss with shadow paging. These two effects explain the very different performance of shadow and nested paging with CNL and Catamount guests.

It is important to point out that the version of Palacios's shadow paging implementation we tested only performs on demand updates of the shadow paging state. With optimizations, such as caching, the differences between nested and shadow paging are likely to be smaller.

D. CTH Application Benchmark

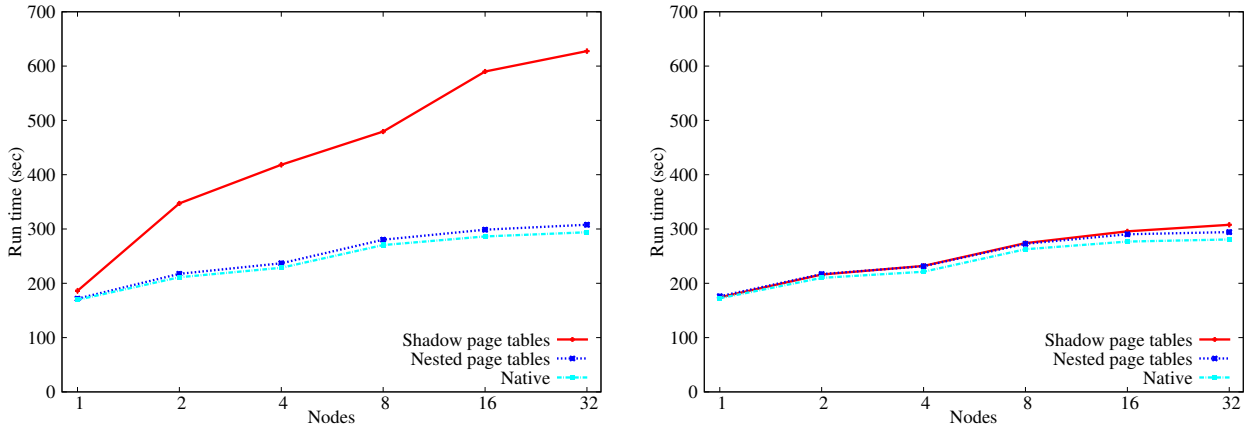
CTH [20] is a multi-material, large deformation, strong shock wave, solid mechanics code developed by Sandia National Laboratories with models for multi-phase, elastic viscoplastic, porous, and explosive materials. CTH supports three-dimensional rectangular meshes; two-dimensional rectangular, and cylindrical meshes; and one-dimensional rectilinear, cylindrical, and spherical meshes, and uses second-order accurate numerical methods to reduce dispersion and dissipation and to produce accurate, efficient results. It is used for studying armor/anti-armor interactions, warhead design, high explosive initiation physics, and weapons safety issues.

Figures 5(a) and 5(b) show the results using the CNL and Catamount guests. We can see that adding virtualization, provided the appropriate choice of shadow or nested paging is made, has virtually no effect on performance or scaling. For this highly communication intensive benchmark, virtualization is essentially free.

E. Intel MPI Benchmarks

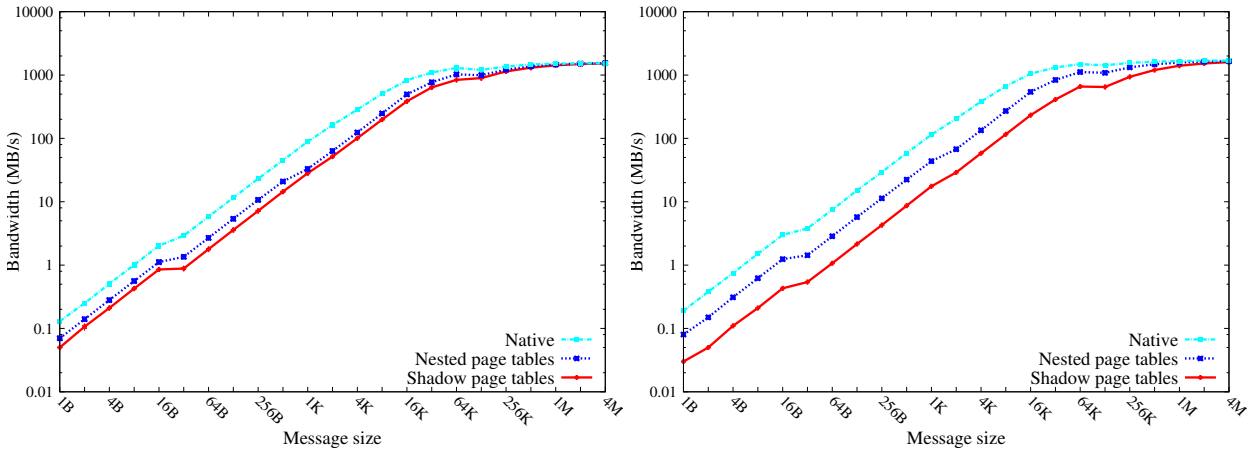
The Intel MPI Benchmarks (IMB) [21], formerly known as PALLAS, are designed to characterize the MPI communication performance of a system. IMB employs a range of MPI primitive and collective communication operations, at a range of message sizes and scales to produce numerous performance characteristics. We ran IMB on top of CNL and Catamount on Red Storm using SeaStar at scales from 2 to 48 nodes. We compared native performance, virtualized performance using shadow paging, and virtualized performance using nested paging. IMB generates large quantities of data. Figures 6 through 7 illustrate the most salient data on CNL and Catamount.

Figure 6 shows the bandwidth of a ping-pong test between two nodes for different message sizes. For large messages, bandwidth performance is identical for virtualized and native operating systems. For small messages where ping-pong bandwidth is latency-bound, the latency costs of virtualization reduce ping-pong bandwidth. We have measured the extra latency introduced by virtualization as either 5 μsec (nested paging) or 11 μsec (shadow paging) for the CNL guest. For the Catamount guest, shadow paging has a higher overhead. Although the SeaStar is accessed via passthrough I/O, interrupts are virtualized. When the SeaStar raises an



(a) CNL Guest (b) Catamount Guest

Figure 5. CTH application benchmark comparing scaling for virtualization with shadow paging, virtualization with nested paging, and no virtualization. Palacios/Kitten can provide scaling to 32 nodes with less than 5% performance degradation.



(a) CNL Guest (b) Catamount Guest

Figure 6. IMB PingPong Bandwidth in MB/sec as a function of message size

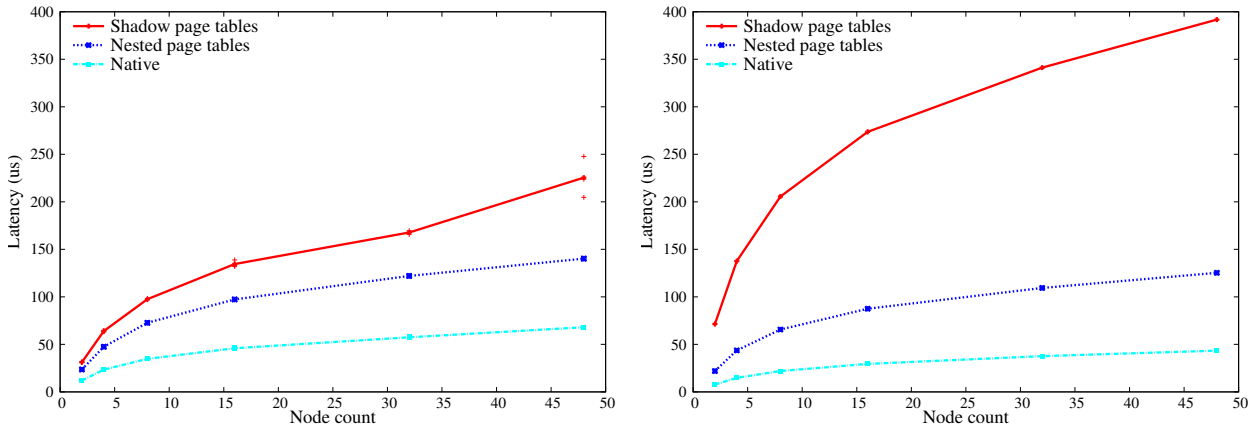
interrupt, a VM exit is induced. Palacios quickly transforms the hardware interrupt into a virtual interrupt that it injects into the guest on VM entry. The guest will quickly cause another VM exit/entry interaction when it acknowledges the interrupt to its (virtual) APIC. Shadow paging introduces additional overhead because of the need to refill the TLB after these entries/exits. This effect is especially pronounced in Catamount since, other than capacity misses, there is no other reason for TLB refills; in addition, Catamount has a somewhat more complex interrupt path that causes two additional VM exits per interrupt. Avoiding all of these VM exits via nested paging allows us to measure the raw overhead of the interrupt exiting process.

In Figure 7, we fix the message size at 16 bytes and examine the effect on an IMB All-Reduce as we scale from 2 to 48 nodes. We can see that the performance impacts of nested and shadow paging diverges as we add more nodes—nested paging is superior here.

The upshot of these figures and the numerous IMB results which we have excluded for space reasons is that the performance of a passthrough device, such as the SeaStar, in Palacios is in line with the expected hardware overheads due to interrupt virtualization. This overhead is quite small. Virtualized interrupts could be avoided using the AMD SVM interrupt handling features, which we expect would bring IMB performance with nested paging-based virtualization in line with native performance. However, at this point, we expect that doing so would require minor guest changes.

F. Infiniband microbenchmarks

To quantify the overhead of Palacios virtualization on a commodity NIC, we ported OpenIB MLX4 (ConnectX) drivers to Kitten along with the associated Linux driver. We also implemented passthrough I/O support for these



(a) CNL Guest

(b) Catamount Guest

Figure 7. IMB Allreduce 16 byte latency in μsec as a function of nodes up to 48 nodes

	Latency (μsec)	Bandwidth (Gb/sec)
Kitten (Native)	5.24	12.40
Kitten (Virtualized)	5.25	12.40
Linux	4.28	12.37

Figure 8. Bandwidth and latency of node-to-node Infiniband on Kitten, comparing native performance with guest performance. Linux numbers are provided for reference.

	HPCCG MFLOPS
Native CNL	588.0
Palacios/Kitten + CNL Guest	556.4
KVM/CNL + CNL Guest	546.4
% Diff Palacios vs. KVM	1.8%

Figure 9. Comparison of Palacios to KVM for HPCCG benchmark.

drivers in Palacios. We then measured round-trip latency for 1 byte messages averaged over 100000 round trips and 1 megabyte message round trip bandwidth averaged over 10000 trips using a ported version of the OpenFabrics `ibv_rc_pingpong`. The server system ran Linux 2.6.27, while the client machine ran either Kitten natively, Kitten as a guest on Palacios using shadow paging, or Linux.

As can be seen in Figure 8, Palacios's pass-through virtualization imposes almost no measurable overhead on Infiniband message passing. Compared to Linux, Kitten both native and virtualized using Palacios slightly outperform Linux in terms of end-to-end bandwidth, but suffers a 1 μsec/round trip latency penalty. We believe this is due to a combination of the lack of support for message-signaled interrupts (MSI) in our current Linux driver support code, as well as our use of a comparatively old version of the OpenIB driver stack. We are currently updating Linux driver support and the OpenIB stack used in in Kitten to address this issue.

G. Comparison with KVM

To get a feel for the overhead of Palacios compared to existing virtualization platforms, we ran the HPCCG benchmark in a CNL guest under both KVM running on a Linux host and Palacios running on a Kitten host. KVM (Kernel-based Virtual Machine) is a popular virtualization platform for Linux that is part of the core Linux kernel as of version 2.6.20. Due to time constraints we were not able to expose the SeaStar to KVM guest environments, so only single node experiments were performed. The same "100 100 100" test problem that was used in Section VI-C was run on a single Cray XT compute node. HPCCG was compiled in serial mode (non-MPI) leading to slightly different performance results. As can be seen in Figure 9, Palacios delivers approximately 1.8% better performance than KVM for this benchmark. Each result is an average of three trials and has a standard deviation less of than 0.66. Note that small performance differences at the single node level typically magnify as the application and system are scaled up.

VII. Future Work

Larger Scale Studies: While our initial results show that Palacios and Kitten are capable of providing scalable virtualization for HPC environments, we intend to continue the evaluation at ever larger scales. We have completed a preliminary large scale study of up to 4096 nodes on the full Red Storm system. The preliminary results show that Palacios continues to impose minimal overhead, delivering performance within 5% as scaling increases.

Symbiotic Virtualization: Based on our results to date, it is evident that the best VMM configuration is heavily dependent on the OS and application behavior inside the guest environment. In other words, *there is no singular VMM configuration suitable for HPC environments*. In order to provide the best performance for every HPC application, a

VMM must be able to adapt its own behavior to the guest's. This adaptability requires that both the VMM and OS cooperate to coordinate their actions. *Symbiotic virtualization* is a new approach to system virtualization where a guest OS and a VMM use high level software interfaces to communicate with each other in order to increase performance and functionality. We are currently exploring the use of *Symbiotic Virtualization* for HPC environments.

VIII. Related Work

Recent research activities on operating systems for large-scale supercomputers generally fall into two categories: those that are Linux-based and those that are not. A number of research projects are exploring approaches for configuring and adapting Linux to be more lightweight. Alternatively, there are a few research projects investigating non-Linux approaches, using either custom lightweight kernels or adapting other existing open-source OSes for HPC.

The Cray Linux Environment [11] is the most prominent example of using a stripped-down Linux system in an HPC system, and is currently being used on the petaflop-class Jaguar system at Oak Ridge National Laboratories. Other examples of this approach are the efforts to port Linux to the IBM BlueGene/L and BlueGene/P systems [22], [23]. Since a full Linux distribution is not used, this approach suffers many of the same functionality weaknesses as non-Linux approaches. In some cases, these systems have also encountered performance issues, for example due to the mismatch between the platform's memory management hardware and the Linux memory management subsystem.

Examples of the non-Linux approach include IBM's Compute Node Kernel (CNK) [24] and several projects being led by Sandia, including the Catamount [2] and Kitten projects as well as an effort using Plan9 [25]. Both CNK and Kitten address one of the primary weaknesses of previous lightweight operating systems by providing an environment that is largely compatible with Linux. Kitten differs from CNK in that it supports commodity x86_64 hardware, is being developed in the open under the GPL license, and provides the ability to run full-featured guest operating systems when linked with Palacios.

The desire to preserve the benefits of a lightweight environment but provide support a richer feature set has also led other lightweight kernel developers to explore more full-featured alternatives [4]. We have also explored other means of providing a more full-featured set of system services [26], but the complexity of building a framework for application-specific OSes is significantly greater than simply using an existing full-featured virtualized OS, especially if the performance impact is minimal.

There has been considerable interest, both recently and historically, in applying existing virtualization tools to HPC environments [27], [28], [29], [30], [31], [32], [33]. However, most of the recent work has been exclusively in the context of adapting or evaluating Xen and Linux on cluster platforms. Palacios and Kitten are a new OS/VMM solution developed specifically for HPC systems and applications. There are many examples of the benefits available from a virtualization layer [34] for HPC. There is nothing inherently restrictive about the virtualization tools used for these implementations, so these approaches could be directly applied to this work.

IX. Conclusion

Palacios and Kitten are new open source tools that support virtualized and native supercomputing on diverse hardware. We described the design and implementation of both Palacios and Kitten, and evaluated their performance. Virtualization support, such as Palacios's, that combines hardware features such as nested paging with passthrough access to communication devices can support even the highest performing guest environments with minimal performance impact, even at relatively large scale. Palacios and Kitten provide an incremental path to using supercomputer resources that has few compromises for performance. Our analysis points the way to eliminating overheads that remain.

References

[1] R. Goldberg, "Survey of virtual machine research," *IEEE Computer*, pp. 34–45, June 1974.

[2] R. Riesen, R. Brightwell, P. Bridges, T. Hudson, A. Maccabe, P. Widener, and K. Ferreira, "Designing and implementing lightweight kernels for capability computing," *Concurrency and Computation: Practice and Experience*, vol. 21, no. 6, pp. 793–817, April 2009.

[3] S. Kelly and R. Brightwell, "Software architecture of the lightweight kernel, Catamount," in *2005 Cray Users' Group Annual Technical Conference*. Cray Users' Group, May 2005.

[4] E. Shmueli, G. Almasi, J. Brunheroto, J. Castanos, G. Dozsa, S. Kumar, and D. Lieber, "Evaluating the effect of replacing CNK with Linux on the compute-nodes of Blue Gene/L," in *roceedings of the 22nd International Conference on Supercomputing*. New York, NY, USA: ACM, 2008, pp. 165–174.

[5] R. Brightwell, T. Hudson, and K. Pedretti, "SMARTMAP: Operating system support for efficient data sharing among processes on a multi-core processor," in *International Conference for High Performance Computing, Networking, Storage, and Analysis*, November 2008.

[6] M. F. Mergen, V. Uhlig, O. Krieger, and J. Xenidis, "Virtualization for high-performance computing," *Operating Systems Review*, vol. 40, no. 2, pp. 8–11, 2006.

[7] I. Raicu, Z. Zhang, M. Wilde, I. Foster, P. Beckman, K. Iskra, and B. Clifford, "Toward loosely-coupled programming on petascale systems," in *ACM/IEEE International Conference on High-Performance Computing, Networking, Storage, and Analysis*, November 2008.

[8] AMD Corporation, "AMD64 virtualization codenamed "Pacific" technology: Secure Virtual Machine Architecture reference manual," May 2005.

[9] Intel Corporation, "Intel virtualization technology specification for the IA-32 Intel architecture," April 2005.

[10] J. R. Lange and P. A. Dinda, "An introduction to the Palacios Virtual Machine Monitor—release 1.0," Northwestern University, Department of Electrical Engineering and Computer Science, Tech. Rep. NWU-EECS-08-11, November 2008.

[11] L. Kaplan, "Cray CNL," in *FastOS PI Meeting and Workshop*, June 2007. [Online]. Available: http://www.cs.unm.edu/~fastos/07meeting/CNL_FASTOS.pdf

[12] D. Hovenmeyer, J. Hollingsworth, and B. Bhattacharjee, "Running on the bare metal with GeekOS," in *35th SIGCSE Technical Symposium on Computer Science Education (SIGCSE)*, 2004.

[13] K. Ferreira, P. Bridges, and R. Brightwell, "Characterizing application sensitivity to OS interference using kernel-level noise injection," in *2008 ACM/IEEE conference on Supercomputing (SC)*, 2008, pp. 1–12.

[14] E. Focht, J. Močnik, F. Unger, D. Sternkopf, M. Novak, and T. Grossmann, *High Performance Computing on Vector Systems 2009*. Springer Berlin Heidelberg, 2009, ch. The SX-Linux Project: A Progress Report, pp. 79–96.

[15] R. Brightwell, K. T. Pedretti, K. D. Underwood, and T. Hudson, "SeaStar interconnect: Balanced bandwidth for scalable performance," *IEEE Micro*, vol. 26, no. 3, pp. 41–57, 2006.

[16] N. Wells, "BusyBox: A Swiss Army knife for Linux," *Linux Journal*, November 2000, http://busybox.net/.

[17] L. Shuler, C. Jong, R. Riesen, D. van Dresser, A. B. Maccabe, L. A. Fisk, and T. M. Stallcup, "The PUMA operating system for massively parallel computers," in *1995 Intel Supercomputer User's Group Conference*. Intel Supercomputer User's Group, 1995.

[18] R. R. Arthur B. Maccabe, Kevin S. Mccurley and S. R. Wheat, "SUNMOS for the Intel Paragon: A brief user's guide," in *Intel Supercomputer Users' Group. 1994 Annual North America Users' Conference*, 1994, pp. 245–251.

[19] M. Heroux, "HPCCG MicroApp," July 2007, https://software.sandia.gov/mantevo/downloads/HPCCG-0.5.tar.gz.

[20] J. E.S. Hertel, R. Bell, M. Elrick, A. Farnsworth, G. Kerley, J. McGlaun, S. Petney, S. Silling, P. Taylor, and L. Yarrington, "CTH: A Software Family for Multi-Dimensional Shock Physics Analysis," in *19th International Symposium on Shock Waves, held at Marseille, France*, July 1993, pp. 377–382.

[21] Intel GmbH, "Intel MPI benchmarks: Users guide and methodology description," 2004.

[22] E. Shmueli, G. Almási, J. Brunheroto, J. Castaños, G. Dózsa, S. Kumar, and D. Lieber, "Evaluating the effect of replacing CNK with Linux on the compute-nodes of Blue Gene/L," in *22nd Annual International Conference on Supercomputing (ICS)*. New York, NY, USA: ACM, 2008, pp. 165–174.

[23] P. Beckman *et al.*, "ZeptoOS project website, http://www.mcs.anl.gov/research/projects/zeptoos/."

[24] J. E. Moreira, M. Brutman, J. Castaños, T. Engelsiepen, M. Giampapa, T. Gooding, R. Haskin, T. Inglett, D. Lieber, P. McCarthy, M. Mundy, J. Parker, and B. Wallenfelt, "Designing a highly-scalable operating system: The Blue Gene/L story," in *ACM/IEEE Supercomputing SC'2006 conference*, 2006.

[25] R. G. Minnich, M. J. Sottile, S.-E. Choi, E. Hendriks, and J. McKie, "Right-weight kernels: an off-the-shelf alternative to custom light-weight kernels," *SIGOPS Oper. Syst. Rev.*, vol. 40, no. 2, pp. 22–28, 2006.

[26] J.-C. Tournier, P. Bridges, A. B. Maccabe, P. Widener, Z. Abudayyeh, R. Brightwell, R. Riesen, and T. Hudson, "Towards a framework for dedicated operating systems development in high-end computing systems," *ACM SIGOPS Operating Systems Review*, vol. 40, no. 2, April 2006.

[27] D. M. Ritchie, "A guest facility for Unicos," in *UNIX and Supercomputers Workshop Proceedings*. USENIX, September 1988, pp. 19–24.

[28] W. Emeneker and D. Stanzione, "HPC cluster readiness of Xen and User Mode Linux," in *2006 IEEE Conference Cluster Computing (CLUSTER)*, 2006, pp. 1–8.

[29] A. Gavrilovska, S. Kumar, H. Raj, K. Schwan, V. Gupta, R. Nathuji, R. Niranjan, A. Ranadive, and P. Saraiya, "High performance hypervisor architectures: Virtualization in HPC systems," in *1st Workshop on System-level Virtualization for High Performance Computing (HPCVirt)*, 2007.

[30] W. Huang, J. Liu, B. Abali, and D. K. Panda, "A case for high performance computing with virtual machines," in *20th Annual International Conference on Supercomputing (ICS)*, 2006, pp. 125–134.

[31] S. Thibault and T. Deegan, "Improving performance by embedding HPC applications in lightweight Xen domains," in *2nd Workshop on System-level Virtualization for High Performance Computing (HPCVirt)*, 2008, pp. 9–15.

[32] A. Tikotekar, G. Vallée, T. Naughton, H. Ong, C. Engelmann, S. L. Scott, and A. M. Filippi, "Effects of virtualization on a scientific application running a hyperspectral radiative transfer code on virtual machines," in *2nd Workshop on System-Level Virtualization for High Performance Computing (HPCVirt)*, 2008, pp. 16–23.

[33] L. Youseff, R. Wolski, B. Gorda, and C. Krintz, "Evaluating the performance impact of Xen on MPI and process execution for HPC systems," in *2nd International Workshop on Virtualization Technology in Distributed Computing (VTDC)*, 2006, p. 1.

[34] A. B. Nagarajan, F. Mueller, C. Engelmann, and S. L. Scott, "Proactive fault tolerance for HPC with Xen virtualization," in *21st Annual International Conference on Supercomputing (ICS)*, 2007, pp. 23–32.