SAND2009-2306C

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

John Lange[⋆] Kevin Pedretti[†] Trammell Hudson[†]
Peter Dinda[⋆] Zheng Cui[‡] Lei Xia[⋆] Patrick Bridges[‡]
Steven Jaconette[⋆] Mike Levenhagen[†] Ron Brightwell[†] Patrick Widener[‡]

{jarusl,pdinda,leixia,jaconette}@northwestern.edu
{ktpedre,mjleven,rbbrigh}@sandia.gov
hudson@osresearch.net {cuizheng,bridges,pmw}@cs.unm.edu
[⋆]Northwestern University, Department of Electrical Engineering and Computer Science
[†]Sandia National Laboratories, Scalable System Software Department
[‡]University of New Mexico, Department of Computer Science

ABSTRACT

Palacios and Kitten are new open source tools that enable applications, whether ported or not, to achieve scalable high performance on large machines. They provide a thin layer over the hardware to support both full-featured virtualized environments and native code bases. Kitten is an OS under development at Sandia that implements a lightweight kernel architecture to provide predictable behavior and increased flexibility on large machines, while also providing Linux binary compatibility. Palacios is a VMM that is under development at Northwestern University and the University of New Mexico. Palacios, which can be embedded into Kitten and other OSes, supports existing, unmodified applications and operating systems by using virtualization that leverages hardware technologies. We describe the design and implementation of both Kitten and Palacios. Our benchmarks show that they provide near native, scalable performance. Palacios and Kitten provide an incremental path to using supercomputer resources that is not performance-compromised.

1. INTRODUCTION

This paper introduces two new operating systems (OSes), Kitten and Palacios, that together provide a flexible, high performance virtualized system software platform for HPC systems. This platform broadens the applicability and usability of HPC systems by:

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. Sandia National Laboratory has also provided assistance. Jack 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.

- 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.

Kitten is a open source operating system substrate for high performance computing under development at Sandia National Labs that provides scalable performance to ported applications. Kitten's simple memory model enables high performance communication and data sharing, and it has a low noise profile in massively multiprocessor environments. Kitten has been developed in the spirit of *lightweight kernels* [27], such as Sandia's Catamount [18] and IBM's CNK [29], that are well known to perform better than commodity kernels for HPC.

Palacios is a "type-I" pure virtual machine monitor [11] (VMM) 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, has emerging support for sophisticated I/O virtualization features [37], pass-through I/O for trusted guests, and *symbiotic virtualization*, a new approach to structuring guests and VMMs. Finally, Palacios is part of an effort to achieve compile-time configurability that will allow developers to generate VMMs of different structures from the same code base.

Kitten and Palacios together provide a scalable, flexible HPC system software platform that addresses the challenges laid out earlier and by others [22]. 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.

Both Palacios and Kitten are open source tools that are available to use and build on right now. Palacios and Kitten can be used separately or together, and are outcomes of community resource development efforts to which everyone is welcome to contribute. They run today on a variety of machines ranging from emulated testing environments, through commodity clusters and servers, all the way to a large scale parallel machine at Sandia.

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

- We introduce and describe the Kitten HPC operating system. Kitten is open source and publicly available.
- We introduce and describe the Palacios virtual machine monitor. Palacios is open source and publicly available.
- 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 the largest scale study to date 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%.

2. MOTIVATION

Palacios and Kitten are parts of larger projects that have numerous motivations. More details are available on their web sites. 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, delivering them so that the application can determine allocation and management policies best suited to maximize its performance. Such kernels provide only the basic services needed to initialize hardware and coordinate application startup. 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. Lightweight kernels can provide nearly maximum possible performance, but they require that applications be carefully ported to their minimalist interfaces.

Increasing portability and compatibility through commodity interfaces. Standardized application interfaces would make it easier to port existing applications, particularly parallel applications, to a lightweight kernel. Even partial ABI compatibility with a common interface, such as the Linux ABI, would allow many existing binaries to run directly on the 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. Furthermore, applications targeting a different commodity kernel require complete rewrites.

Achieving full application and OS compatibility through virtualization. Full system virtualization provides full compatibility at the hardware level, allowing all 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.

Integrating full OS and lightweight kernel application components. Applications in which the majority of the compute nodes would ideally run a lightweight kernel and a smaller subset need the functionality of a full OS are common. For example, coupling multiple simulations using Python requires nodes running an OS that supports the dynamic linking of Python's runtime environment and libraries, while the individual simulation codes require a lightweight kernel for performance. Virtualization makes such combinations straightforward.

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. The alternative is to reboot nodes on a per-job basis. We view this approach as overly restrictive and potentially harmful in several ways: system reliability is jeopardized by more reboot cycles, diagnosing and monitoring the health of individual nodes is difficult, and the system is less available for use. Management based on virtualization would also make it possible to backfill work on the machine using loosely-coupled programming jobs [26] 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. The ability to capture the state of an entire virtual machine and restore it without any direct application involvement is a promising approach for dealing with reliability issues facing future extreme-scale systems.

Enhancing systems software research in HPC and elsewhere. The combination of Kitten and Palacios 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.

3. 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 [1]—to achieve superior scalability on massively parallel supercomputers while at the same time exposing 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 operating systems to be loaded on-demand.

Kitten is being developed as part of a research project at Sandia National Laboratories that is investigating system software techniques for better leveraging multicore processors and hardware virtualization in the context of capability supercomputers. The simple framework provided by a lightweight kernel facilitates experimentation and has led to novel techniques such as SMARTMAP [4], which halves the memory bandwidth requirements of intra-node message passing. Kitten is also being used to explore system-level options for improving resiliency to hardware faults, arguably the most significant issue facing large-scale supercomputers.

Kitten currently targets the x86_64 architecture, but could be easily ported to other architectures. The code base borrows heavily from the Linux kernel when doing so does not compromise scalability or performance (e.g., the bootstrap code). Subsystems that are performance critical, such as memory management and task scheduling, are replaced with code written from scratch for Kitten. To avoid licensing and export control issues, the Kitten kernel uses no code from prior Sandia-developed lightweight kernels. Kitten consists of 61–92-thousand lines of C and assembly, as shown in Figure 2. 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.

3.1 Architecture

Kitten (Figure 1) 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

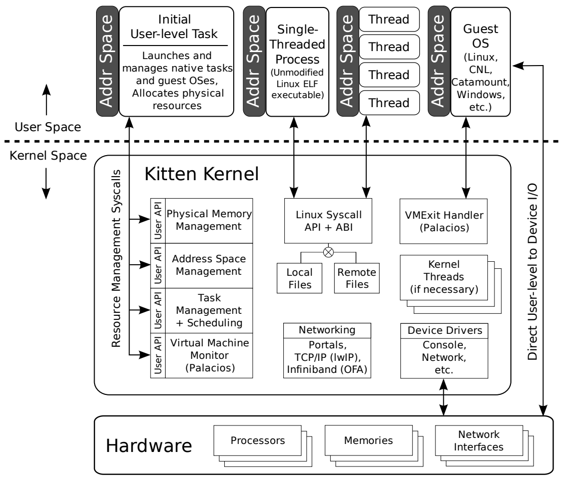

Addr Space
Initial User-level Task
Launches and manages native tasks and guest OSes, Allocates physical resources
Addr Space
Single-Threaded Process (Unmodified Linux ELF executable)
Addr Space
Thread
Thread
Thread
Thread
Addr Space
Guest OS (Linux, CNL, Catamount, Windows, etc.)
User Space
Kernel Space
Kitten Kernel
Resource Management Syscalls
User API
Physical Memory Management
User API
Address Space Management
User API
Task Management + Scheduling
User API
Virtual Machine Monitor (Palacios)
Linux Syscall API + ABI
Local Files
Remote Files
Networking
Portals, TCP/IP (lwIP), Infiniband (OFA)
VMExit Handler (Palacios)
Kernel Threads (if necessary)
Device Drivers
Console, Network, etc.
Direct User-level to Device I/O
Hardware
Processors
Memories
Network Interfaces

Figure 1: Kitten architecture.

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 implemented is intended to support the usage of the high performance computing applications in use at Sandia. The subset is sufficient to support Glibc's NPTL POSIX threads implementation and GCC's OpenMP implementation. Implementing additional system calls is a relatively straightforward process.

The Kitten kernel also 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. We recently ported the Open Fabrics Alliance (OFA) Infiniband stack to Kitten without making any significant changes to the OFA code.

3.2 Memory Management

Unlike traditional general-purpose kernels, Kitten delegates most virtual and physical memory management to user-space. The initial task allocates memory to a new application 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 Kitten application 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

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

Component	Lines of Code sloccount .	wc *.c *.h *.s
Kitten		
Kitten Core (C)	17,995	29,540
Kitten x86_64 Arch Code (C+Assembly)	14,604	22,190
Misc. Contrib Code (Kbuild + lwIP)	27,973	39,593
Palacios Glue Module (C)	286	455
Total	60,858	91,778
Palacios		
Palacios Core (C+Assembly)	15,084	24,710
Palacios Virtual Devices (C)	8,708	13,406
XED Interface (C+Assembly)	4,320	7,712
Total	28,112	45,828
Grand Total	88,970	137,606

Figure 2: Lines of code in Kitten and Palacios as measured with the SLOCCount tool and with the `wc` tool.

Cray XT) and potentially decrease cost by eliminating the need for translation table memory and table walk hardware on the network interface.

3.3 Task Scheduling

All contexts of execution on Kitten 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 initial task allocates a set of processors to each task it creates and starts the task executing on one of them. The task may then spawn additional tasks (threads) on its set of processors via the `clone()` system call. By default tasks created with `clone()` are spread out to minimize the number of tasks per processor but the native Kitten task creation system call can be used to specify the exact processor a task should be spawned on.

4. PALACIOS

Palacios[2] is an OS independent VMM designed as part of the the V3VEE project (`http://v3vee.org`). The V3VEE project is a collaborative community resource development project involving Northwestern University and the University of New Mexico. It seeks to develop a virtual machine monitor framework for modern architectures (those with hardware virtualization support) that will permit the compile-time creation of VMMs with different structures, including those optimized for computer architecture research, computer systems research, operating systems teaching, and research and use in high performance computing. Palacios is the first VMM from the project and will form the basis of the broader framework. Support for high performance computing significantly informed its design.

Palacios currently targets the x86 and x86_64 architectures (hosts and guests) and makes extensive, and non-optional use of the AMD SVM [1] extensions (partial support for Intel VT [15, 35] is also implemented). Palacios uses Intel's XED library from Pin [21, 6], to decode instructions in some cases, and it uses the BOCHS [20] BIOS and VGA BIOS to bootstrap a guest machine. Palacios supports both 32 and 64 bit host OSes as well as 32 and 64 bit guest OSes[3]. Palacios supports virtual memory using either shadow or nested paging. It runs directly on the hardware and provides a non-paravirtualized interface to the guest with optional paravirtualized extensions. An extensive infrastructure for hooking of guest resources facilitates extension and experimentation.

Palacios was developed from scratch at Northwestern University. Figure 2 shows the scale of Palacios, as measured by two different source code analysis tools. Note that the Palacios core is quite small. The entire VMM, including the default set of virtual devices is on the order of 28–45 thousand lines of C and assembly. The combination of Palacios and Kitten is 89–138 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. Palacios is publicly available from `http://v3vee.org`, and a technical report [19] describes the initial release in detail. Palacios is released under a BSD license.

Palacios supports multiple physical host and virtual guest environments. Palacios is compatible with any AMD architecture with SVM features enabled. We have successfully run Palacios on commodity Dell and HP servers, a high end Infiniband cluster, as well as Red Storm development cages consisting of Cray XT nodes. Most of the development is done using the QEMU emulator environment. Palacios also supports the virtualization of a diverse set of guest OS environments. Palacios supports full featured Linux environments such as 32 bit Puppy Linux 3.0 and the 64 bit Finnix 92.0 distributions. Palacios has also successfully virtualized several lightweight HPC OSes including CNL [17], Catamount [18], and Kitten itself.

4.1 Architecture

Palacios is an OS independent VMM, and as such is designed to be easily portable to diverse host operating systems. Currently, Palacios actively supports Kitten, for high performance environments, as well as GeekOS [13], an educational operating system developed to teach operating system development. 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 500 lines of code needed to be written to embed Palacios into Kitten. Palacios is designed to be internally modular and extensible and provides common interfaces for registering event handlers for common operations.

Figure 3 illustrates the Palacios architecture.

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, and specific memory pages, as well as hypercalls.[4] Palacios also includes functionality to receive notifications of host events such as general interrupts, keystrokes and timer ticks. This combined functionality makes it possible to construct a wide range of different guest environments. We include a configuration interface that supports common configuration options (amount of memory, selection of virtual and physical devices, etc).

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

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

[3] 64 bit guests are only supported on 64 bit hosts

[4] Although Palacios is not a paravirtualized VMM, we do allow direct guest calls to the VMM.

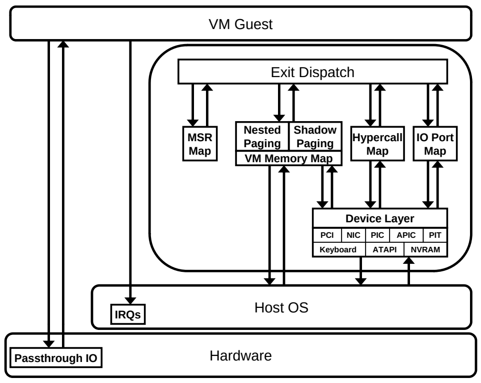


Figure 3: Palacios architecture.

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.

Palacios jointly handles interrupts with the host OS. In general, Palacios can disable local and global interrupts in order to have interrupt processing on a core run at times it chooses. For the most part, handling interrupts correctly requires no changes on the part of the host OS. However, for performance reasons, and for complicated interactions such as passthrough devices, small host OS interrupt handling changes may be necessary.

4.2 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. Several aspects of its implementation facilitate this:

- 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.
- Full system virtualization: Palacios does not require guest OS changes. This allows it to run existing kernels without any porting, including lightweight kernels [27] like Kitten, Catamount, Cray CNL [17], and IBM's CNK [29].
- 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 [9] injected by the VMM layer. Palacios makes no use of internal timers, nor does it accumulate deferred work.

4.3 Symbiotic Virtualization

Palacios also serves as a platform for research on symbiotic virtualization, a new approach to structuring VMMs and guest OSes so that they can better work together without requiring such cooperation for basic functionality of the guest OS either on the VMM or on raw hardware. In symbiotic virtualization, an OS targets the native hardware interface as in full system virtualization, but also optionally exposes a software interface that can be used by a VMM, if present, to increase performance and functionality. Neither the VMM nor the OS needs to support the symbiotic virtualization interface to function together, but if both do, both benefit. Symbiotic virtualization has the potential to provide the compatibility benefits of full system virtualization while providing an incremental path towards the functionality and performance benefits possible with paravirtualization.

The high performance computing context provides a special opportunity for symbiotic virtualization because there can be a much greater level of trust between the VMM, guest OS, and applications. Because of the increased level of trust, a VMM and OS can be designed to coexist symbiotically. This approach allows, for example, a VMM to provide a trusted guest with direct access to hardware resources. Because the guest is symbiotic the VMM can assume that the guest will configure the granted resources in a safe manner, using information provided by the VMM. This allows the guest to perform I/O directly without the overhead of permission checks or a translation layer.

5. INTEGRATING PALACIOS AND KITTEN

The explicit host interface exported by Palacios results in an extremely simple integration with Kitten. The integration includes no internal changes in either Kitten or Palacios. As shown in Figure 2 the interface was implemented in only a few hundred lines of code contained in a single file. The interface file and Palacios library are encapsulated in a an optional compile time module for Kitten.

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 [5]. 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 Kitten and Palacios 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.

Besides memory-mapped IO, 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 which allows for selective interrupt exiting, which would 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. We have developed another technique, virtual passthrough I/O (VPIO), for passthrough I/O in environments where such trust is impossible [37].

6. 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 less than 5%.

6.1 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 Labs. 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 [5]. 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 [18] operating system) to 7.0 μsec (using the native CNL [17] operating system).

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.

6.2 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 of the Red Storm machine at Sandia. CNL is a minimized Linux (2.6 kernel) that leverages BusyBox [36] 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 [31][2]. These operating systems, 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 very simple memory model with a physically-contiguous virtual memory layout, parallel job launch, and message passing facilities.

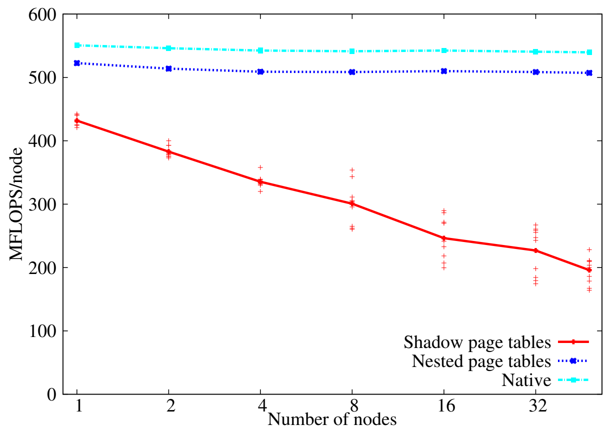
6.3 HPCCG Benchmark Results

We used the HPCCG benchmark to evaluate the impact of virtualization on application performance and scaling. HPCCG [12] 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.

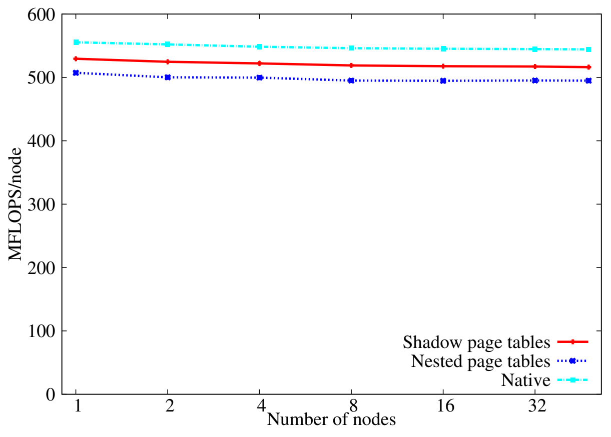
We ran HPCCG on top of CNL and Catamount on Red Storm, considering scales from 1 to 48 nodes. A fixed-size problem per node was used to obtain these results. The specific HPCCG input arguments were "100 100 100", requiring approximately 380 MB per node. This software stack was compiled with the Portland 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 both minuscule and independent of scale. For Catamount, all variances are minuscule and independent, even with shadow paging.

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



(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.

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 does not include either speculative paging or shadow page table caching, features currently in development. With these features, the performance differences between nested and shadow paging are likely to be smaller. Interestingly, the tested shadow paging implementation is 1606 LOC compared to the tested nested paging implementation's 483 LOC—-achieving correctness, much less high performance, in a shadow paging implementation is much more challenging than in a nested paging implementation.

6.4 CTH Benchmark

CTH [8] 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.

6.5 Intel MPI Benchmarks

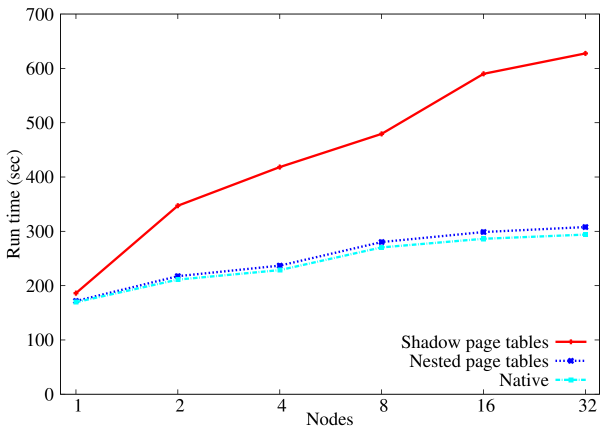
The Intel MPI Benchmarks (IMB) [16], formerly known as PALLAS, are designed to characterize the MPI communication performance of a system. IMB employs a range of MPI primitive and collective communication operations, at a range of message sizes and scales to produce numerous performance characteristics. 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 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. Avoiding these VM exits via nested paging allows us to measure the raw overhead of the interrupt exiting process.

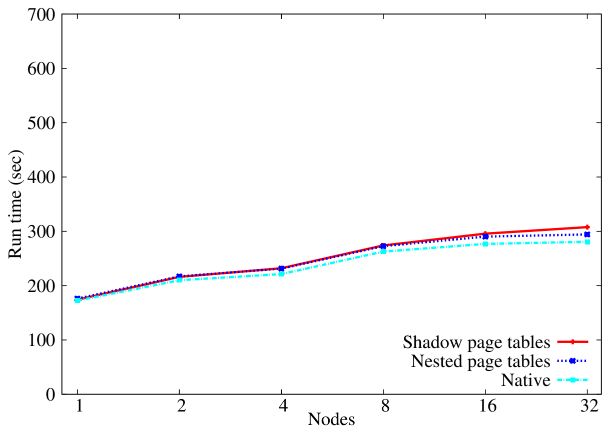
virtualized. When the SeaStar raises an interrupt, a VM exit is

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

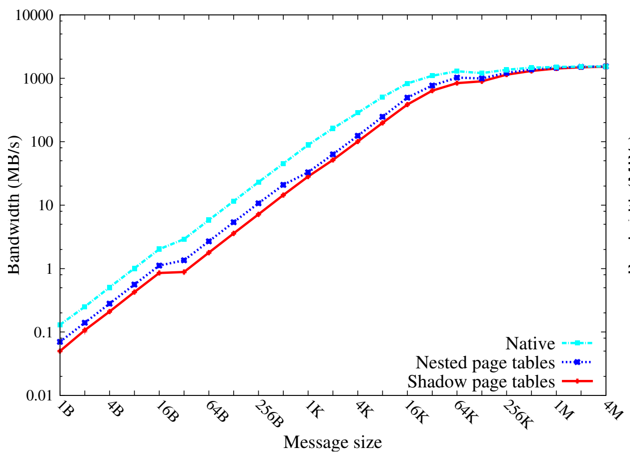


(a) CNL Guest

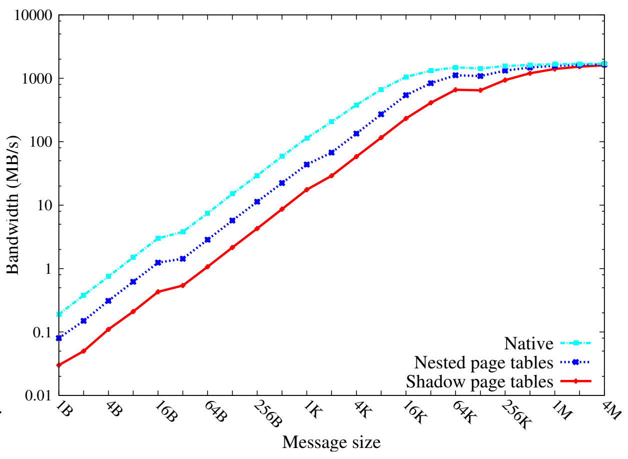


(b) Catamount Guest

Figure 5: CTH 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

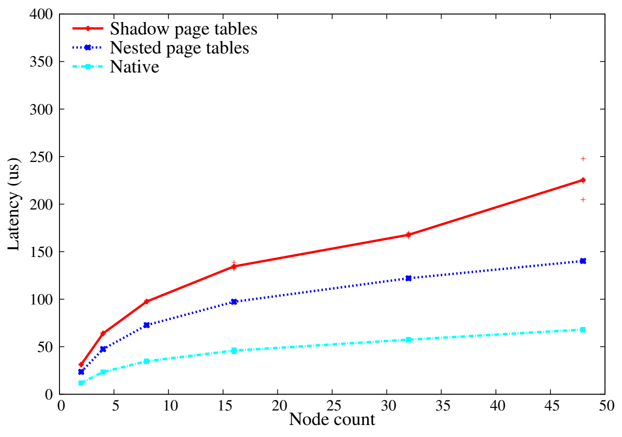
passthrough device, such as the SeaStar, in Palacios is in line with the expected hardware overheads due to interrupt virtualization. This overhead is quite small. Virtualized interrupts could be avoided using the AMD SVM interrupt handling features, which we expect would bring IMB performance with nested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

7. RELATED WORK

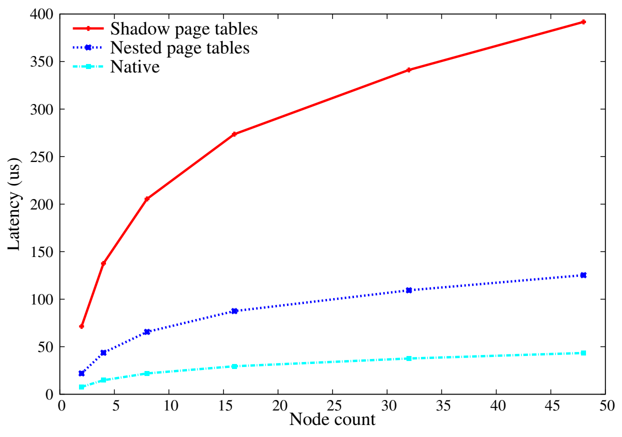
Recent research activities on operating systems for large-scale supercomputers generally fall into two categories: those that are Linux-based and those that are not. A number of research projects are exploring approaches for configuring and adapting Linux to be more lightweight. Alternatively, there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

The Cray Linux Environment [17] is the most prominent example of the Linux-based approach and is currently being used on the petaflop-class Jaguar system at Oak Ridge National Laboratories. Cray's approach is to specially configure a nearly unmodified Linux kernel and combine it with a BusyBox-based initramfs image to provide the compute node environment. Since a full Linux distribution is not used, this approach suffers many of the same functionality weaknesses as the non-Linux approaches, while not providing the performance advantages. Other examples of this approach are the efforts to port Linux to the IBM BlueGene/L and BlueGene/P systems [30, 3]. These projects have encountered performance issues 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



(a) CNL Guest



(b) Catamount Guest

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

Node Kernel (CNK) [24] and several projects being led by Sandia, including the Catamount [27] and Kitten projects as well as an effort using Plan9 [23]. The custom OS approaches have been designed to minimize OS noise and jitter so that tightly-coupled parallel applications can scale to full-system execution with as much performance as possible. 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 [29]. We have also explored other means of providing a more full-featured set of system services [34], 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 [28, 7, 10, 14, 32, 33, 38]. 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 [25] for HPC. There is nothing inherently restrictive about the virtualization tools used for these implementations, so these approaches could be directly applied to Palacios and Kitten.

8. CONCLUSION

Palacios and Kitten open source tools, available now, that support virtualized and native supercomputing on diverse hardware. We described the design and implementation of both Kitten and Palacios, 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 furthermore points the way to eliminating overheads that remain.

9. REFERENCES

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

[2] Arthur B. Maccabe, Kevin S. McCurley, R. R., and Wheat, S. R. 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.

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

[4] Brightwell, R., Hudson, T., and Pedretti, K. 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).

[5] Brightwell, R., Pedretti, K. T., Underwood, K. D., and Hudson, T. SeaStar interconnect: Balanced bandwidth for scalable performance. *IEEE Micro 26*, 3 (2006), 41–57.

[6] Bungale, P., and Luk, C.-K. PinOS: A programmable framework for whole system dynamic instrumentation. In *3rd international conference on Virtual execution environments (VEE)* (June 2007).

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

[8] E.S. Hertel, J., Bell, R., Elrick, M., Farnsworth, A., Kerley, G., McGlaun, J., Petney, S., Silling, S., Taylor, P., and Yarrington, L. 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.

[9] Ferreira, K., Bridges, P., and Brightwell, R. Characterizing application sensitivity to OS interference using

kernel-level noise injection. In *2008 ACM/IEEE conference on Supercomputing (SC)* (2008), pp. 1–12.

[10] GAVRILOVSKA, A., KUMAR, S., RAJ, H., SCHWAN, K., GUPTA, V., NATHUJI, R., NIRANJAN, R., RANADIVE, A., AND SARAIYA, P. High performance hypervisor architectures: Virtualization in HPC systems. In *1st Workshop on System-level Virtualization for High Performance Computing (HPCVirt)* (2007).

[11] GOLDBERG, R. Survey of virtual machine research. *IEEE Computer* (June 1974), 34–45.

[12] HEROUX, M. HPCCG MicroApp. `https://software.sandia.gov/mantevo/downloads/HPCCG-0.5.tar.gz`, July 2007.

[13] HOVENMEYER, D., HOLLINGSWORTH, J., AND BHATTACHARJEE, B. Running on the bare metal with GeekOS. In *35th SIGCSE Technical Symposium on Computer Science Education (SIGCSE)* (2004).

[14] HUANG, W., LIU, J., ABALI, B., AND PANDA, D. K. A case for high performance computing with virtual machines. In *20th Annual International Conference on Supercomputing (ICS)* (2006), pp. 125–134.

[15] INTEL CORPORATION. Intel virtualization technology specification for the IA-32 Intel architecture, April 2005.

[16] INTEL GMBH. Intel MPI benchmarks: Users guide and methodology description, 2004.

[17] KAPLAN, L. Cray CNL. In *FastOS PI Meeting and Workshop* (June 2007).

[18] KELLY, S., AND BRIGHTWELL, R. Software architecture of the lightweight kernel, Catamount. In *2005 Cray Users' Group Annual Technical Conference* (May 2005), Cray Users' Group.

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

[20] LAWTON, K. Bochs: The open source IA-32 emulation project. `http://bochs.sourceforge.net`.

[21] LUK, C.-K., COHN, R., MUTH, R., PATIL, H., KLAUSER, A., LOWNEY, G., WALLACE, S., REDDI, V. J., AND HAZELWOOD, K. Pin: Building customized program analysis tools with dynamic instrumentation. In *ACM SIGPLAN 2005 Conference on Programming Language Design and Implementatin (PLDI)* (June 2005).

[22] MERGEN, M. F., UHLIG, V., KRIEGER, O., AND XENIDIS, J. Virtualization for high-performance computing. *Operating Systems Review 40*, 2 (2006), 8–11.

[23] MINNICH, R. G., SOTTILE, M. J., CHOI, S.-E., HENDRIKS, E., AND MCKIE, J. Right-weight kernels: an off-the-shelf alternative to custom light-weight kernels. *SIGOPS Oper. Syst. Rev. 40*, 2 (2006), 22–28.

[24] MOREIRA, J. E., BRUTMAN, M., CASTAÑOS, J., ENGELSIEPEN, T., GIAMPAPA, M., GOODING, T., HASKIN, R., INGLETT, T., LIEBER, D., MCCARTHY, P., MUNDY, M., PARKER, J., AND WALLENFELT, B. Designing a highly-scalable operating system: The Blue Gene/L story. In *ACM/IEEE Supercomputing SC'2006 conference* (2006).

[25] NAGARAJAN, A. B., MUELLER, F., ENGELMANN, C., AND SCOTT, S. L. Proactive fault tolerance for HPC with Xen virtualization. In *21st Annual International Conference on Supercomputing (ICS)* (2007), pp. 23–32.

[26] RAICU, I., ZHANG, Z., WILDE, M., FOSTER, I., BECKMAN, P., ISKRA, K., AND CLIFFORD, B. Toward loosely-coupled programming on petascale systems. In *ACM/IEEE International Conference on High-Performance Computing, Networking, Storage, and Analysis* (November 2008).

[27] RIESEN, R., BRIGHTWELL, R., BRIDGES, P., HUDSON, T., MACCABE, A., WIDENER, P., AND FERREIRA, K. Designing and implementing lightweight kernels for capability computing. *Concurrency and Computation: Practice and Experience 21*, 6 (April 2009), 793–817.

[28] RITCHIE, D. M. A guest facility for Unicos. In *UNIX and Supercomputers Workshop Proceedings* (September 1988), USENIX, pp. 19–24.

[29] SHMUELI, E., ALMASI, G., BRUNHEROTO, J., CASTANOS, J., DOZSA, G., KUMAR, S., AND LIEBER, D. 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, 2008), ACM, pp. 165–174.

[30] SHMUELI, E., ALMÁSI, G., BRUNHEROTO, J., CASTAÑOS, J., DÓZSA, G., KUMAR, S., AND LIEBER, D. 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, 2008), ACM, pp. 165–174.

[31] SHULER, L., JONG, C., RIESEN, R., VAN DRESSER, D., MACCABE, A. B., FISK, L. A., AND STALLCUP, T. M. The PUMA operating system for massively parallel computers. In *1995 Intel Supercomputer User's Group Conference* (1995), Intel Supercomputer User's Group.

[32] THIBAULT, S., AND DEEGAN, T. 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.

[33] TIKOTEKAR, A., VALLÉE, G., NAUGHTON, T., ONG, H., ENGELMANN, C., SCOTT, S. L., AND FILIPPI, A. M. 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.

[34] TOURNIER, J.-C., BRIDGES, P., MACCABE, A. B., WIDENER, P., ABUDAYYEH, Z., BRIGHTWELL, R., RIESEN, R., AND HUDSON, T. Towards a framework for dedicated operating systems development in high-end computing systems. *ACM SIGOPS Operating Systems Review 40*, 2 (April 2006).

[35] UHLIG, R., NEIGER, G., RODGERS, D., SANTONI, A., MARTIN, F., ANDERSON, A., BENNETTT, S., KAGI, A., LEUNG, F., AND SMITH, L. Intel virtualization technology. *IEEE Computer* (May 2005), 48–56.

[36] WELLS, N. BusyBox: A Swiss Army knife for Linux. *Linux Journal* (November 2000). `http://busybox.net/`.

[37] XIA, L., LANGE, J., AND DINDA, P. Towards virtual passthrough I/O on commodity devices. In *Workshop on I/O Virtualization at OSDI* (December 2008).

[38] YOUSEFF, L., WOLSKI, R., GORDA, B., AND KRINTZ, C. 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.