Improving Compute Node Performance Using Virtualization

Brian Kocoloski John Lange
{briankoco,jacklange}@cs.pitt.edu
Department of Computer Science
University of Pittsburgh
Pittsburgh, PA 15260

Abstract

Modified variants of Linux are likely to be the underlying operating systems for future exascale platforms. Despite the many advantages of this approach, a subset of applications exist in which a lightweight kernel (LWK) based OS is needed and/or preferred. We contend that virtualization is capable of supporting LWKs as virtual machines (VMs) running at scale on top of a Linux environment. Furthermore, we claim that a properly designed virtual machine monitor (VMM) can provide an isolated and independent environment that avoids the overheads of the Linux host OS. To validate the feasibility of this approach we demonstrate that given a Linux host OS, benchmarks running in a virtualized LWK environment are capable of outperforming the same benchmarks executed directly on the Linux host.

1 Introduction

Linux-derived kernels and environments are quickly becoming accepted as the dominant operating systems for large scale supercomputing platforms [8, 1, 4], and by all appearances this trend will continue as we move into the exascale era. This is in large part due to the fact that Linux-based environments provide a number of advantages, such as leveraging existing codebases and providing a high degree of familiarity for application developers. Furthermore, as HPC platforms become more complex, it is becoming increasingly infeasible to develop custom OSes from scratch that adequately leverage the large number of new hardware features and devices. In addition to greater complexity, access to the systems is becoming more opaque and restricted. For instance, Compute Node Linux (CNL) [8], the heavily modified version of Linux distributed by Cray, is very tightly controlled both through export restrictions and NDAs, and also includes binary only device drivers. These restrictions make modifying CNL or developing a new custom OS for the platform exceedingly difficult.

While results have shown that Linux-based environments are fully capable for most applications, there does exist a subset of applications where the overheads associated with a Linux environment have a significant impact on performance. For instance, previous work has compared CNL against a legacy lightweight OS (Catamount) [15]. The results showed that while CNL was able to provide adequate performance (within $\sim$5%) for most applications, there were several cases where CNL's results where on average $\sim$17% worse. While these results are dated, and CNL's performance has undoubtedly improved since, we nevertheless believe that in a supercomputing environment there will always exist a subset of applications that will exhibit superior performance when executing on top of a Lightweight Kernel (LWK). Accordingly, we posit that there will always exist a place for customized LWKs on production exascale systems, even if they are not the primary OS of choice.

In this paper we claim that while Linux will become the standard exascale OS, there should be a mechanism whereby LWKs can be used by the applications that require them. We also believe that a virtualization

This work is made possible by support from the National Science Foundation (NSF) via grants CNS-0709168 and CNS-0707365, and the Department of Energy (DOE) via grant DE-SC0005343.

based approach is capable of providing this functionality through the creation of independent and isolated virtual machine environments. LWKs running inside VMs could be given direct access to system resources without having to incur the overheads of going through the host OS, and so applications would be able to achieve superior performance than would be possible if they executed natively on the host kernel. Furthermore, virtualization will enable Linux-based exascale machines to provision these specialized lightweight environments without the effort of reconfiguring the entire system.

To justify our claim we have evaluated the performance of the Palacios Virtual Machine Monitor in an unmodified Linux host OS. For this work we used version 1.3 of the Palacios VMM, which integrates with a host Linux kernel via the kernel module interface. The codebase is publicly available and has been described in more detail previously [10]. Palacios 1.3 is fully compatible with a wide range of unmodified Linux kernels including versions 2.6.32 through 3.1. In addition, Palacios is fully compatible with CNL, based on versions available to Oak Ridge and Sandia National Labs.

Palacios' approach to virtualizing Linux-based environments is to allocate a pool of resources that is separately managed by the Palacios VMM, outside the context of the host Linux kernel. This approach allows Palacios to provide hardware resources directly to a guest environment without any of the overheads of going through the Linux resource management layers. The benefit of this approach is that Palacios is able to avoid the overheads associated with Linux and so provide near native performance for a LWK executing inside a VM. This allows applications optimized for a lightweight environment to achieve better performance than they would if executing directly on the host Linux kernel.

This paper makes the following contributions:

- We make the claim that virtualization is capable of improving application performance over a native Linux host environment.
- We present preliminary results using Palacios that demonstrate the feasibility of this approach.

The remainder of the paper is laid out as follows: In section 2, we discuss our VMM Palacios. In section 3, we discuss the drawbacks of Linux-based supercomputing environments. We present a performance evaluation, including descriptions of the benchmarks we ran, in section 4. We discuss future work in section 5, related work in section 6, and we conclude in section 7.

2 The Palacios VMM

The work in this paper is based on the Palacios Virtual Machine Monitor whose in-depth description has been presented elsewhere [12, 11]. Palacios provides modular virtualization functionality to x86-based architectures, such as the Cray XT, as well as commodity HPC systems. Our previous work with the Palacios VMM focused on providing a high performance virtualization layer for LWKs and other supercomputing class operating systems, resulting in a demonstration that virtualization can be used on modern supercomputers with minimal overhead. While our initial focus was on providing virtualization support for lightweight kernels, it has become evident that the future petascale and exascale era will be ever more Linux-centric. While we still believe in the utility of lightweight kernel architectures, it is unclear exactly what their role will be in future systems. Based on this observation, the Palacios VMM was ported to Linux in order to evaluate its ability to provide HPC-capable virtualization in a non-lightweight host environment. This effort has culminated in the release of version 1.3 of Palacios in November 2011 (described elsewhere [10]), which provides full support for host OSes based on both Linux and the Kitten lightweight kernel.

Due to the additional complexity and feature set of a typical Linux kernel, we had to take a different approach to host OS integration. Lightweight kernels are notable in their approach to resource management in that they often rely on the resource consumer to handle many of the management functions. The reason for this approach is to allow an HPC application to decide precisely which resources it needs and how they are allocated, which reduces both the management overhead of the host OS and the limitations imposed by the OS architecture. This hands-off approach made implementing the Palacios VMM in a LWK straightforward in that the VMM was able to directly allocate raw hardware resources and expose them directly to a guest

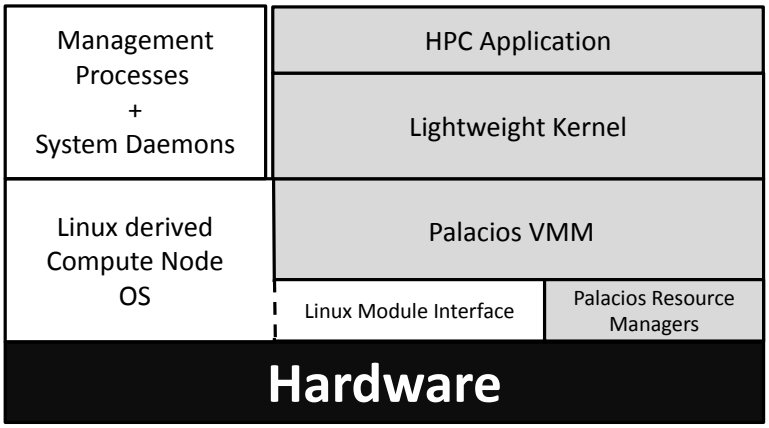


Figure 1: Organization of a virtualized HPC Linux System

environment. Contrary to the LWK approach, Linux-derived kernels tend to impose their own management architectures over the hardware resources. This makes it difficult to directly allocate raw resources inside our VMM, and required a different approach to OS/VMM integration.

When porting Palacios to Linux we focused on two central goals. First, to ensure wide compatibility with existing and future systems, our architecture must not require modifications of any part of the Linux kernel. Second, in order to provide acceptable performance in an HPC environment, Palacios must be provided with direct access to raw hardware resources while avoiding any overheads imposed by Linux's management layers. A high level view of our approach is shown in Figure 1. In order to meet the first goal, we integrated the Palacios VMM into a loadable kernel module that could be directly inserted into a host OS at run time. This approach required that we utilize existing module APIs that remain relatively stable compared to the rest of the kernel. Thus we are able to ensure widespread compatibility while also eliminating the need to rely on any modification of the Linux codebase. For the second goal, we utilized a number of techniques that allow Palacios to effectively bypass the Linux management layers and access hardware resources directly when needed. In this paper, we present two of these techniques for memory management and processor scheduling.

3 The Drawbacks of Linux

While there are a number of strong reasons to adopt Linux-based environments in supercomputing environments, there are a number of shortcomings as well. Historically, the overhead imposed by Linux has been significant and so imposed performance problems when scaling up to supercomputing class systems. However, recent work has done much to minimize Linux's performance penalty, both as part of the mainline kernel development as well as customizations made by the HPC community. Currently, there are two significant Linux distributions targeting large scale supercomputing platforms: Cray's Compute Node Linux (CNL) [8] and ZeptoOS [1]. While these two examples have done a great deal to make Linux-based environments acceptable for petascale platforms, there are still a number of deficiencies (technical and non-technical) to adopting Linux on current and future platforms. We focus first on the non-technical issues.

The largest drawback of using Linux in the supercomputing context is the fact that, while much improved, Linux is still a commodity operating system. As such its goals are necessarily different from those of the HPC community. This results in the necessity of modifying and customizing Linux's architecture in order to avoid many of the overheads that are inherent in its design. Furthermore, once these modifications have been made, there is a large amount of maintenance cost associated with keeping the modifications current with the constantly evolving Linux codebase. These maintenance requirements place a large burden on the OS

developers, and are probably only sustainable in a corporate setting or large research groups. As an example, the latest version of the ZeptoOS compute node kernel is based on Linux kernel version 2.6.29, a codebase released in early 2009. This is not meant to be critical of the ZeptoOS project, but is merely highlighting the fact that continued maintenance is a significant burden that is very difficult to sustain. In fact, current trends with CNL seem to indicate that there is an active effort to reduce the number of modifications made to the kernel itself.

An additional limitation inherent in the supercomputing context is the fact that access to the hardware and software platforms is significantly restricted to a small subset of the HPC community. The source code for CNL is both export-controlled and apparently restricted to those who have signed an NDA with Cray. Furthermore, the device drivers for the Gemini devices (Cray's latest networking device) are distributed only in binary form. These restrictions make it exceedingly difficult for improvements to be made by the broader HPC systems community.

We claim that, with the Palacios VMM, these drawbacks can be largely reduced. First, as stated earlier, Palacios is implemented as a loadable kernel module that requires no modifications to the host Linux kernel. Furthermore, the interfaces used by Palacios are considered to be generally stable, and persist across many kernel versions. For example, we can confirm that the current version of Palacios is fully compatible with Linux versions 2.6.32 through 3.1 (almost a 2 year window). In fact, it has been confirmed that Palacios is compatible with Cray's CNL version 2.6.32 and can launch a Palacios-based VM on a compute node in a Cray XT5 development rack. This was done by a collaborator at Oak Ridge National Lab using a version of Palacios that until now has only been developed with a commodity Linux kernel. Additionally, Palacios and Kitten are fully open source[1] and freely available to the entire community. These features mean that even in a fairly restricted environment, the benefits of a LWK are accessible to a broad range of applications and users.

3.1 Memory Management

Linux's memory management architecture has long been identified as a source of considerable overhead for HPC environments. As such, a significant amount of work has gone into optimizing memory performance on both CNL and ZeptoOS. These optimizations focus on improving memory management performance as well as reducing TLB miss rates by modifying the standard memory architecture to utilize large contiguous memory regions. In ZeptoOS, this takes the form of their "Big Memory" architecture [16] that preallocates a large contiguous memory region at boot time that can then be used directly by applications. While information concerning Cray's CNL is limited due to access restrictions, it does appear that Cray's approach mirrors the standard HugeTLBfs architecture from Linux. It should be noted that the key feature of optimized memory systems is to provide large contiguous and preallocated memory regions that are directly accessible to the application.

In this paper, we propose approaching the memory system optimization problem by splitting it into two halves. In the bottom half, the Palacios VMM handles the allocation of large contiguous regions of physical memory in a way that bypasses the host OS memory management system. Using a minimal memory management layer and the availability of Hot Pluggable memory in the host OS, Palacios is able to completely take over the management of very large blocks of contiguous memory,[2] while at the same time disabling the host OS memory management system from accessing the allocated regions. This allows Palacios to completely avoid any and all overheads associated with the host OS's memory management architecture. Next, the large and contiguous memory regions are exposed to a LWK running inside a VM context. Due to the fact that memory is allocated in large contiguous chunks, Palacios can ensure the use of large page support in the underlying shadow or nested page tables. In fact, in many scenarios, our approach would be able to use large 1GB page table entries when they become available on future architectures. After Palacios has exposed the large preallocated memory region to the VM, the lightweight guest kernel is able to manage

[1] Palacios is distributed under a BSD license, while Kitten is distributed under the GPL

[2] The standard hot pluggable block size is 128MB, and contiguous blocks are generally readily available on a reasonable system configuration

the memory directly with minimal overhead and so provides a very low overhead memory management layer to the running application.

Our architecture has several advantages over existing approaches. First, our system is capable of providing larger contiguous memory regions to an application than is possible in CNL. At minimum our memory blocks are available in 128MB contiguous regions, and generally quite a bit more. This is in contrast to the 2MB maximum available in Cray's CNL. Second, our architecture is capable of allocating these memory regions dynamically at run time, unlike ZeptoOS which requires memory reservation at machine boot time. This means that when a Palacios VM is not running, the entire system memory is available for the host OS to use in any way it wishes. And finally, we are able to deliver these large allocations using existing Linux kernel features and so *require no modifications to the Linux host OS*.

3.2 OS Noise

The second technical issue with Linux that we address is OS noise. Generally speaking, Linux is often considered to be a noisier environment than a LWK, which has implications for application performance. Palacios' approach to noise minimization is similar to that of memory management in that it bypasses the host scheduler as much as possible to control noise levels. The current version of Palacios achieves this in a rather simple way that nevertheless is capable of reducing the latent noise of the host kernel when running a VM. Palacios achieves this by 1) directly controlling when the Linux scheduler is able to run, and 2) taking advantage of a tickless host kernel (enabling the NO_HZ compile-time option).[3]

4 Preliminary Evaluation

In this section, we evaluate the Palacios VMM architecture in the context of a commodity Linux kernel. The purpose of our experiments is to evaluate whether Palacios can provide superior performance to the native Linux host environment when coupled with a LWK. For our initial experiments, we focused on two microbenchmarks that directly measure memory performance and OS noise characteristics. In addition, we ran three other benchmarks that are more representative of typical HPC applications. However, before we discuss the results of these experiments, we must first discuss two issues pertinent to their design.

Page Size By default, Linux uses 4KB pages to physically back memory allocated to user-level processes. For typical applications that do not have particularly large memory footprints, this configuration does not impose problems. However, for others such as HPC applications that have very large memory requirements, this page size can cause memory performance to suffer. For these applications, larger page sizes can lead to improved memory performance for a variety of reasons. As an example, because page size is inversely proportional to the number of page table entries, as page size increases, the total number of page table entries decreases. Therefore, when the operating system needs to fix up a page fault in a small page configuration, the number of page table entries that it must traverse in order to do so is greater than it would be if using a larger page size. Furthermore, the larger number of page table entries that must be translated leads to greater pressure on the TLB, increasing miss rates and further increasing the number of page translations that must occur.

Small pages are particularly problematic for virtual environments that must provide an additional level of translation between the guest's physical address space and the host. This can lead to an exponential increase in the steps needed to complete a translation, leading to large negative impacts on performance [3]. As such, systems configured to use small page sizes pose a worst case scenario for virtualized environments, resulting in significant performance penalties. It should be noted that while most HPC environments rely on large page support as a means of optimizing memory performance, it is not an entirely universal practice. Therefore, in order to fully evaluate the capabilities of the Palacios VMM we evaluated application performance using both large and small pages.

[3]Palacios VMs run as kernel threads and thus are not interrupted by the OS timer in tickless configurations.

NUMA For our evaluation, we ran each experiment on a NUMA (Non-Uniform Memory Access) machine with two equally sized zones of memory. We will see that memory performance for applications that only allocate resources from a single NUMA zone, and applications for which resources are allocated from multiple zones, can vary dramatically.

4.1 Experimental Setup

For our experiments we ran each benchmark on a dedicated Dell R415 server configured with two 6-core Opteron 4174HE CPUs and 16GB of memory. As stated, the memory layout consisted of 2 NUMA zones equally shared between the sockets with memory interleaving disabled. Each benchmark was executed 10 times with 1, 2, 4, and 8 threads each running on a separate core. For the 1, 2, and 4 thread runs the application's threads were tied to the same NUMA node; for the 8 thread runs the application threads were split evenly across the NUMA zones. We ran each benchmark in 4 different software configurations: native Linux without any optimizations, native Linux with NUMA-aware CPU bindings, native Kitten, and Kitten executing inside a Palacios VM on the native Linux OS. Each configuration used explicit CPU assignments with the exception of the default Linux environment, for which we did not apply any thread binding. Our virtual machine image was configured with 1GB[4] of memory implemented using nested page tables. The multicore benchmarks were implemented using OpenMP for shared memory. The host Linux kernel was an unmodified stock kernel from Fedora 15 (2.6.40.6-0.fc15.x86_64) with the NO_HZ option enabled for tickless operation. The versions of Palacios and Kitten were taken from their respective Release 1.3 versions.

It should be noted that we viewed these experiments as preliminary and part of a feasibility study. In particular, we were initially unconcerned with various low level details, such as NUMA-aware memory allocation and page size. As such, each environment was configured with the default page size of 4KB and was free to have memory allocated from either NUMA zone (the Linux NUMA-aware environment pinned threads to CPUs in a NUMA-aware fashion, but not memory; so in a sense, it was only partially NUMA-aware).

However, we also wanted to see how more optimized, HPC-tailored versions of each configuration would perform. As such, we also ran a set of experiments that configured both a native Linux environment and a virtual Kitten environment to use large pages. In Linux, page size can be manipulated through use of the HugeTLBfs interface, in which userspace applications can request that a specific amount of physical memory be backed by a particular page size. In Kitten, page size can be set as a compile-time parameter. Furthermore, as another optimization, we limited both the native Linux and virtual Kitten environments to execute on a single NUMA socket. In Linux, the HugeTLBfs interface also allows processes to declare which NUMA zones they prefer to have memory backed on, and, for the virtual Kitten experiments, Palacios was careful to only allocate resources for Kitten from a single socket.

As a result of these constraints, each benchmark for this subset of experiments tailored for HPC was executed 10 times with only 1, 2, and 4 threads, as an 8 thread configuration would necessarily result in the crossing of NUMA boundaries (again, the test machine had two 6-core sockets). Furthermore, we note that we did not execute benchmarks in a native Kitten environment for this evaluation, as Kitten currently lacks support for NUMA systems. We note, admittedly slightly tongue-in-cheek, that this is a further advantage of running a LWK in a Palacios VM: we are able to add functionality for a feature that the guest OS itself does not provide!

4.2 Methodology

While our evaluation is limited to a commodity Linux kernel, we believe that it is still useful in providing insight into our system. As stated earlier, access to CNL is fairly restricted and so we were not able to experiment with it directly. It should also be noted that our goal is to isolate a VM's performance with Palacios from a given host OS and show that superior performance is possible in this configuration. Based on this, the use of a commodity kernel could be viewed as the worst case scenario for our system since there

[4]At the time of submission, issues arose when creating larger virtual machines. We have since successfully launched virtual machines with up to 4GB of memory

are presumably a larger amount of overheads and noise present. Again, the goal of this evaluation is to show we can deliver performance to a virtualized LWK that is superior to its native performance. As such, the specific Linux version should not matter too much as long as its performance does in fact diverge from the LWK. Additionally, the use of microbenchmarks presents a fairly common case that commodity Linux has been specifically optimized for.

An additional limitation of our evaluation is the fact that we were not able to perform a multinode scaling study. This was due to numerous issues such as the lack of support in Kitten for our network devices, as well as the lack until recently of passthrough device support in Palacios when running on Linux. Nevertheless we believe that our results still validate our earlier claims that Palacios is capable of providing better memory performance with a lower noise profile than the host kernel.

For each benchmark, we calculated not only the average performance but its standard deviation. The reason for this is that consistent per node performance has often been associated with superior scalability in large scale supercomputing environments, as most applications execute in lock step and must wait for the slowest node to complete.

4.3 Benchmarks

As stated previously, we evaluated our approach using two microbenchmarks that directly measure the aspects of the system we were trying to optimize: memory performance and the OS noise profile. These experiments were conducted using the Stream microbenchmark [13] to evaluate the memory system performance, as well as the Selfish Detour benchmark [2] from Argonne National Laboratory to characterize the noise profiles. We also ran three other benchmarks that were more representative of HPC applications: the well known Linpack [6] benchmark, HPCCG from the Mantevo Miniapp collection [7], and a parameterized version of HPCCG using variable data types acquired from Sandia National Labs. Because of the lack of MPI support in Kitten, we ran an OpenMP enabled version of Linpack [5]. We will now describe each benchmark and their respective results in more detail.

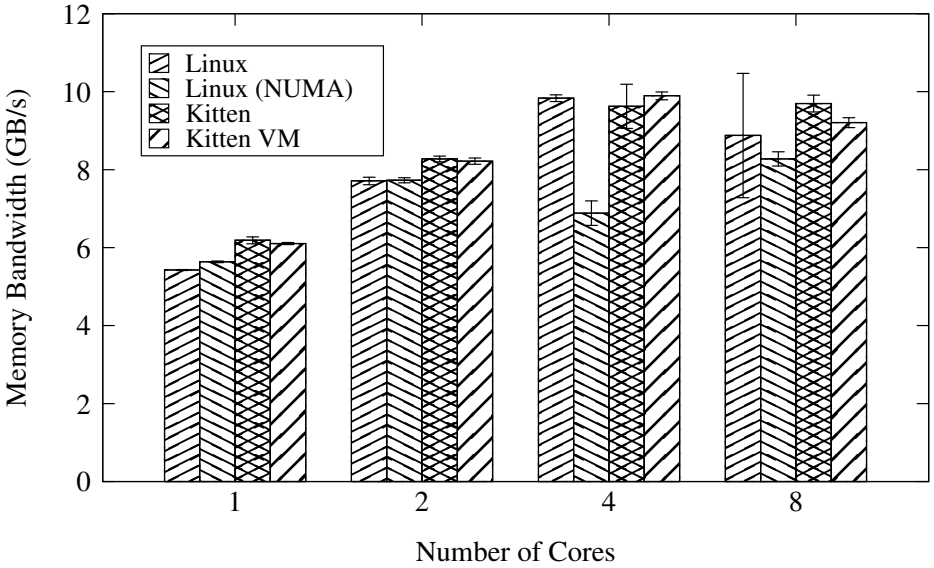
4.4 Experimental Results

In this section we present the performance results we collected from our benchmark runs.

Stream The first priority of our evaluation was to show that our virtualized approach to memory management delivers superior performance over that provided by the host OS. In order to focus our measurements on the memory system directly we ran the Stream benchmark to collect measurements of the available memory bandwidth. The Stream benchmark is designed to directly measure the memory transfer rates provided by the underlying OS and hardware. The Stream microbenchmark is implemented as a small vector kernel, which we configured to use ∼500MB of memory.

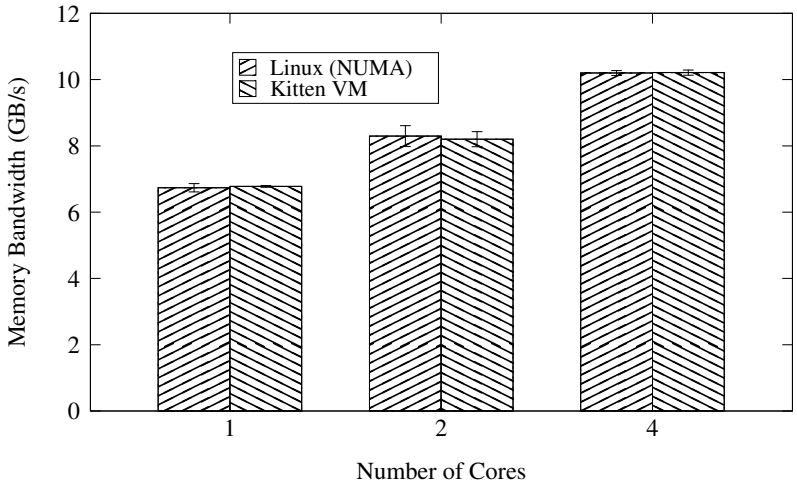
The results for the 4KB page experiments are shown in Figure 2. On average, Palacios was able to provide 400MB/s better memory performance than native Linux, with an average standard deviation that was lower by 0.34. In all cases, Palacios was able to provide better or almost equal performance compared to the native Linux environment. Furthermore, in all cases the virtualized environment provided comparable (and in some cases significantly better) variance in performance compared to the native Linux environment. The results of this benchmark demonstrate that Palacios can provide better memory performance than native Linux, and comparable performance in the worst case. This is largely encouraging since these results were derived from a common memory microbenchmark and so probably represent the most optimized code path for each system.

Conversely, the results of the large page Stream experiments are shown in Figure 3. These results indicate significant improvements in memory performance for both the native Linux host and the virtual Kitten environment. Native Linux experienced an average improvement of over 11%, while Kitten experienced an increase of over 4.5%. The performance for both configurations was also very consistent. As expected, large pages make a positive impact in both settings.



	Mean / Stdev			
Configuration	1 Core	2 Cores	4 Cores	8 Cores
Native Linux	5.43 / 0.01	7.71 / 0.10	9.83 / 0.09	8.88 / 1.59
Native Linux (NUMA)	5.64 / 0.02	7.73 / 0.07	6.88 / 0.31	8.27 / 0.18
Native Kitten	6.19 / 0.09	8.27 / 0.07	9.62 / 0.57	9.69 / 0.21
Kitten VM on Palacios/Linux	6.10 / 0.03	8.22 / 0.08	9.89 / 0.10	9.21 / 0.13

Figure 2: Average memory bandwidths for the Stream benchmark using 4KB pages



	Mean / Stdev		
Configuration	1 Core	2 Cores	4 Cores
Native Linux (NUMA)	6.73 / 0.12	8.30 / 0.31	10.20 / 0.07
Kitten VM on Palacios/Linux	6.78 / 0.03	8.20 / 0.23	10.21 / 0.08

Figure 3: Average memory bandwidths for the Stream benchmark when using 2MB pages

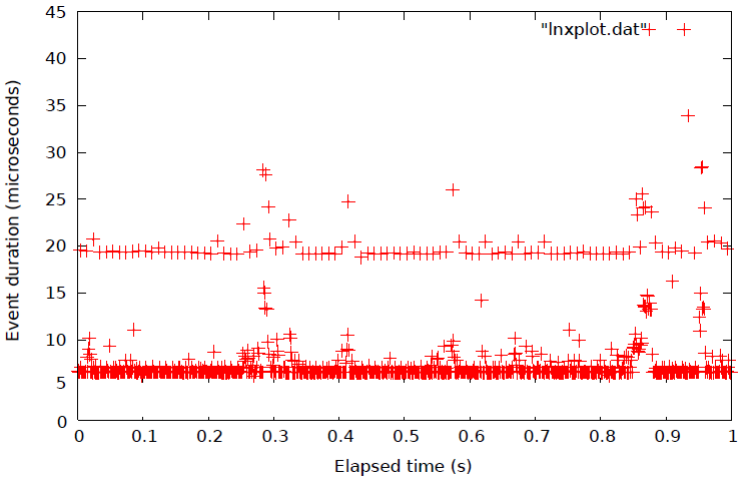


Figure 4: Linux OS noise in the Selfish Detour benchmark

However, the improvement for the native Linux host proved to significantly reduce the gap in raw memory system performance that we saw in the 4KB experiments. While this certainly suggests that there may not be as stark a contrast in memory performance as suggested by the 4KB results in Figure 2, it is nevertheless still reasonable to expect that native Kitten may outperform Linux, as there is certainly some performance penalty, albeit likely a small one, imposed by the VMM.

Selfish The second priority of our evaluation was to determine whether Palacios was capable of providing a superior noise profile over Linux. To achieve this we ran the Selfish Detour benchmark [2] from Argonne National Laboratory. The Selfish benchmark is designed to execute a very tight control loop that measures and detects any interruptions of the benchmark's execution. As such it detects sources of noise that could negatively impact application performance. For this benchmark we ran Selfish on native Linux, native Kitten, and virtualized Kitten.

The results of this benchmark are shown in Figures 4-6. Each figure shows disruption events as a scatter plot with their duration corresponding to the Y-Axis. While there is a fair amount of low level noise for both the native Linux and virtualized Kitten configurations, Figure 4 shows that the native Linux environment exhibits noise as a result of the kernel's 100Hz periodic timer. The timer activation is a result of the application's execution, regardless of the fact that Linux was configured to be tickless. The handling of these timer events takes on average 20 microseconds. In comparison, the noise profile of Palacios demonstrates a much lower number of periodic interrupts. In fact, as shown in Figure 6, the timer interrupts that do occur are the result of the less frequent 10Hz periodic timer inside the Kitten guest kernel, as can be clearly seen near the 0.4 microsecond tick marks in Figure 5. Notably, the timer interrupts from the native Linux run are absent.

These results are not to suggest that Palacios can eliminate all of the noise present in the Linux host OS. Rather, they demonstrate that Palacios can *positively* impact the noise profile in a way that prevents some noise from a host OS from interfering with a virtual machine environment. Indeed, while the periodic ticks on the host are eliminated in the virtual environment, there is a greater degree of low-level noise. On average, the background interruptions experienced by Palacios take 35 microseconds longer to handle than the background noise in Kitten. We note that the reason for this is most likely due to the exit and event handling of the virtual machine monitor. This means that the durations of the interruptions are likely to decrease as VM exit and entry latencies continue to decrease.

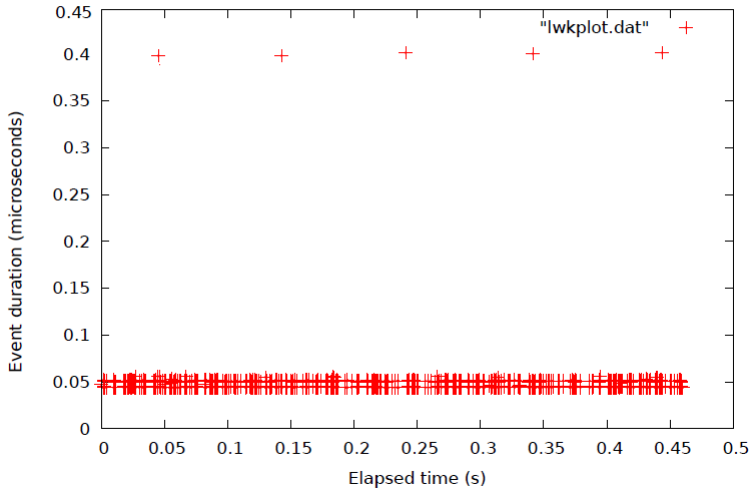


Figure 5: Native Kitten noise in the Selfish Detour benchmark

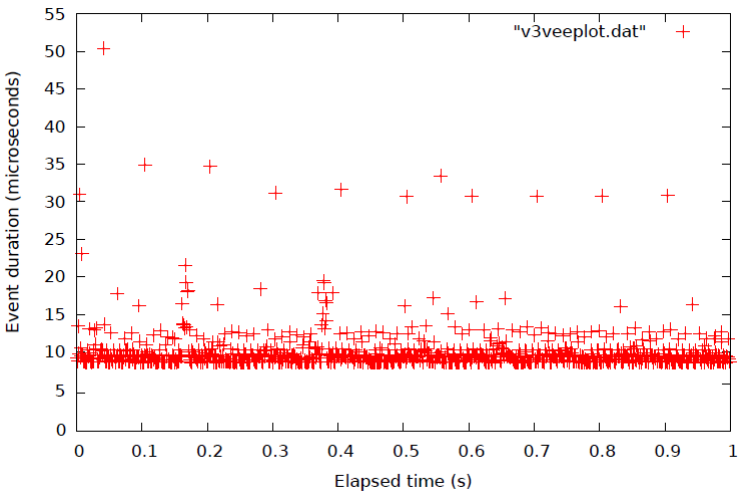
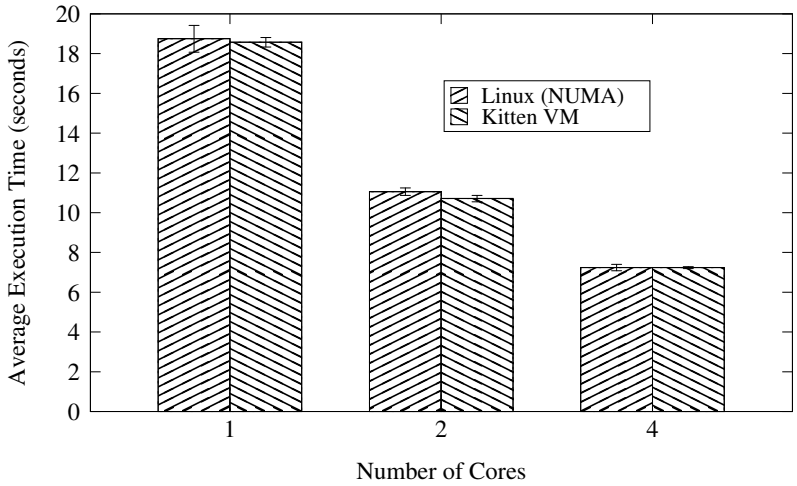


Figure 6: Kitten VM on Linux Host noise in the Selfish Detour benchmark

Linpack We next conducted a set of experiments to measure how well Palacios would perform when executing the Linpack benchmark. The Linpack benchmark, known for its use as a performance metric for the Top500 supercomputing list, is a compute-intensive benchmark that spends much of its execution time solving a dense system of linear equations. As mentioned previously, due to the current lack of MPI support in Kitten, we ran an OpenMP enabled version of Linpack. Furthermore, note that for this benchmark, we tested only the HPC-tailored native Linux and virtual Kitten environments discussed previously.

The results of this benchmark are shown in Figure 7. As we can see, when running on either 1 or 2 cores, Palacios is able to provide a slightly better environment than native Linux. While the performance difference is not huge, it is steadily superior. On the other hand, in the 4 core configuration, the performance is essentially equal. Furthermore, as seen in many of the previous experiments, Palacios clearly provides a



	Mean / Stdev		
Configuration	1 Core	2 Cores	4 Cores
Native Linux (NUMA)	18.75 / 0.68	11.06 / 0.19	7.24 / 0.16
Kitten VM on Palacios/Linux	18.57 / 0.24	10.71 / 0.16	7.24 / 0.05

Figure 7: Average execution times for the Linpack benchmark using 2MB pages

consistent, low noise environment, as the standard deviations in each core configuration are all very low.

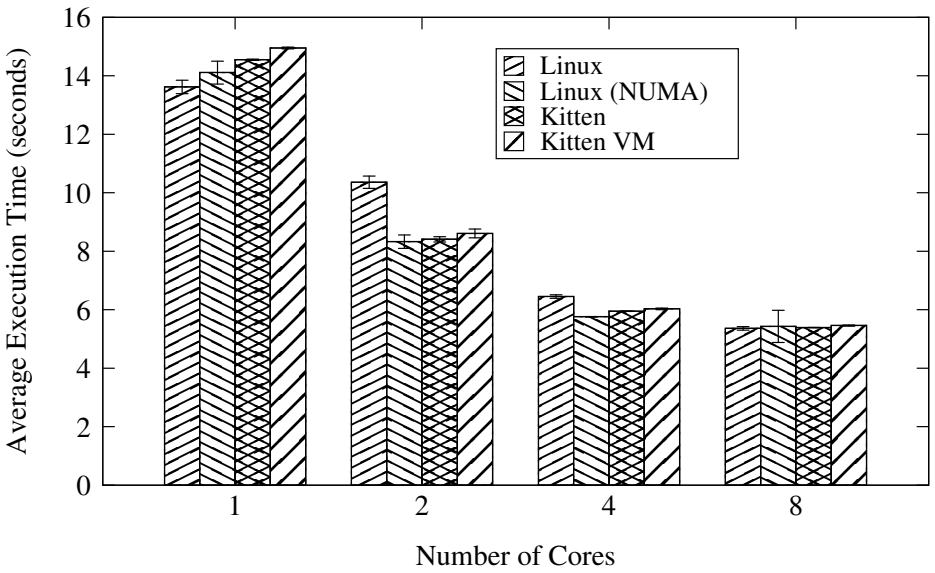
These results are particularly encouraging. Because of the compute-intensive nature of this benchmark, one might expect that the impact of the virtualization layer would be rather pronounced. Instead, not only was the performance mostly equal, but the Palacios configuration actually exhibited slightly superior performance in two of the three configurations. Furthermore, due to its prevalence in evaluating HPC systems, Linpack is likely to represent a rather optimized code path for any operating system targeting HPC applications. Accordingly, we find it very encouraging that the virtual Kitten configuration performed this well.

HPCCG Finally, we ran another two benchmark applications in order to measure how well Palacios performs when executing workloads from more representative applications. For this evaluation we executed two configurations of the HPCCG benchmark as found in the Mantevo Miniapp suite. HPCCG is a simple conjugate gradient solver whose workload is representative of many HPC applications. It performs the conjugate gradient method to solve a system of linear equations represented by a sparse (mostly zero-valued) matrix. We ran two different configurations of this benchmark.

Our first evaluation of HPCCG was configured to use double-precision floating point types in order to evaluate a more memory-intensive task. As with Stream, we tested all four configurations with 4KB pages, and we also tested the optimized native Linux and virtual Kitten environments with 2MB pages. These results are shown in Figures 8 and 9. In addition, we also ran HPCCG with a configuration that used single-precision floating point data types which resulted in a more CPU-intensive workload; these results are shown as the HPCCG benchmark in Figures 10 and 11 for small and large pages respectively.

Figure 8 shows the results of the double-precision, 4KB page HPCCG benchmark experiments. In the single core configuration, we see that Palacios performs worse than the native Linux environment. For each of the 2, 4, and 8 core configurations, however, we see only a small discrepancy between Linux and the Kitten VM. In addition, Palacios is able to provide performance with less variation than either Linux configuration, which indicates that it is likely to exhibit better scalability.

Next, the results of the HPCCG experiments using double-precision data types and large pages are shown



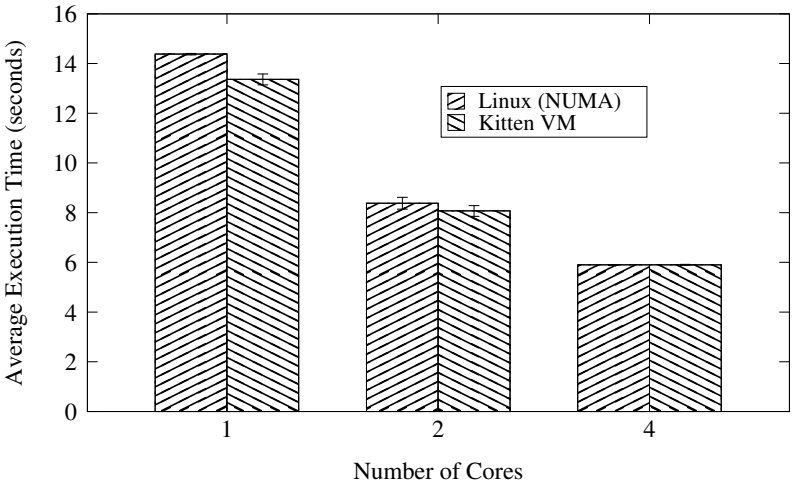
	Mean / Stdev			
Configuration	1 Core	2 Cores	4 Cores	8 Cores
Native Linux	13.62 / 0.23	10.36 / 0.21	6.45 / 0.06	5.36 / 0.06
Native Linux with NUMA control	14.11 / 0.39	8.33 / 0.23	5.76 / 0.01	5.43 / 0.55
Native Kitten	14.55 / 0.02	8.41 / 0.09	5.95 / 0.01	5.39 / 0.01
Kitten VM on Palacios/Linux	14.95 / 0.03	8.61 / 0.15	6.03 / 0.03	5.46 / 0.02

Figure 8: Average execution times for the HPCCG benchmark using double-precision floating point data-types and 4KB pages

in Figure 9. As can be seen, Palacios provides superior performance for the 1 and 2 core configurations, while the performance on 4 cores is essentially equivalent. The surprising result here is that on 1 core, the use of large pages in the native Linux host actually reduces performance by roughly 5%, while Kitten's performance on Palacios improves by roughly 10%. Furthermore, both configurations exhibit highly consistent results, so it is unlikely that these results are anomalous.

In addition to these experiments, we configured HPCCG to use single-precision floating point data types in an effort to create a more CPU-intensive task. The results of these experiments, for small and large pages, are shown in Figures 10 and 11. As shown in Figure 10, Palacios provides better overall application performance than native Linux when executing from the same NUMA node. However, when executing across NUMA nodes (the 8 core run), Palacios exhibits slightly worse performance. It should be noted that for all cases Palacios' performance is considerably more consistent, which again bodes well for the scaling capabilities of Palacios-based environments.

Lastly, the results of the HPCCG experiments using single-precision floating point data types and large pages can be seen in Figure 11. This experiment is important because it is the only configuration in which the virtual Kitten environment is consistently outperformed by the native Linux host. While it is not entirely clear why the performance gap exists, it is likely due in part to the decreased memory requirements in comparison to the double-precision experiments. Furthermore, the use of large pages induced only a small impact on Kitten's performance, while Linux experienced a reduction in run time, averaged across the 1, 2, and 4 core environments, of over 17%. Clearly, the impact of different page sizes on different data types in these environments is interesting in and of itself.



	Mean / Stdev		
Configuration	1 Core	2 Cores	4 Cores
Native Linux (NUMA)	14.38 / 0.03	8.38 / 0.24	5.90 / 0.02
Kitten VM on Palacios/Linux	13.36 / 0.22	8.07 / 0.22	5.90 / 0.02

Figure 9: Average execution times for the HPCCG benchmark using double-precision floating point data-types and 2MB pages

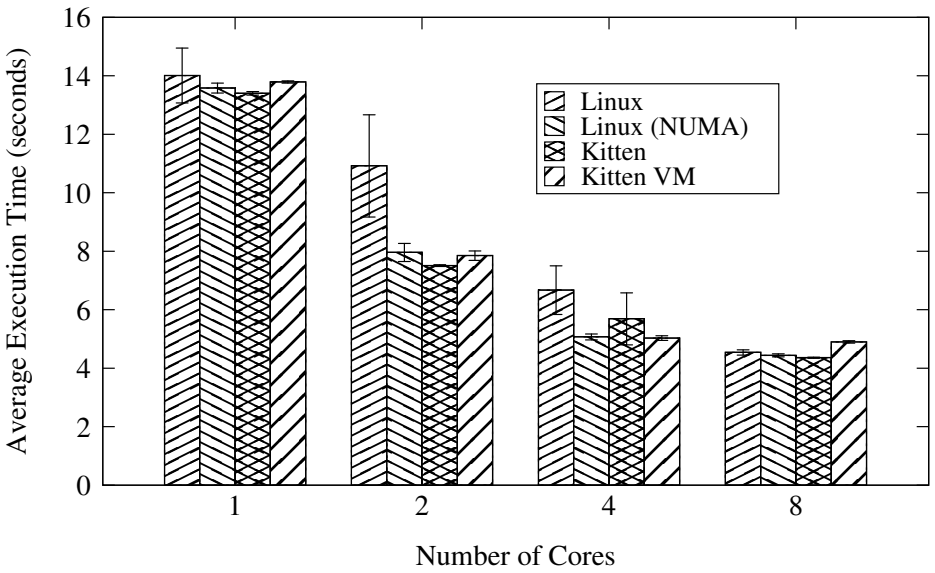
4.5 Analysis

The results of our evaluation showed that across almost all configurations, Palacios was able to provide superior performance for a set of HPC applications and benchmarks. Even when compared against an optimized native Linux environment, Palacios and Kitten were able to deliver comparable if not slightly better performance despite the overheads often associated with a virtualized architecture. Palacios is fully capable of providing an isolated and independent execution environment for lightweight kernels executing on top of a Linux-based environment. Furthermore, the virtualized LWK is capable not only of bypassing the underlying resource managers in the host, but also of providing separate management layers that provide superior performance. While these results are still preliminary, they nevertheless show that our approach is feasible at small scales and could provide a path for fully exploiting LWK architectures on Linux based supercomputing systems.

5 Future Work

We are currently planning on expanding our approach and investigating the capabilities of Palacios at larger scales, on actual Cray hardware, and with a CNL host kernel. As part of this work, we have begun the process of integrating full passthrough I/O functionality on top of Linux-based systems, which will allow Palacios to provide high performance I/O capabilities to guest environments. We note that this functionality will be capable of providing passthrough I/O to systems both with and without IOMMUs. For non-IOMMU equipped systems we can leverage our earlier work with Symbiotic Passthrough [11].

We have recently begun exploring the possibility that our approach can be readily deployed in a cloud setting to provide virtual HPC environments on commodity clouds. Previous work has shown that deploying HPC applications in the cloud is often infeasible due to resource contention, noise issues, and layout problems. We have shown that the approaches presented in this paper allow us to attack many of these problems, thereby making cloud-based HPC much more palatable for the broader HPC community [9].



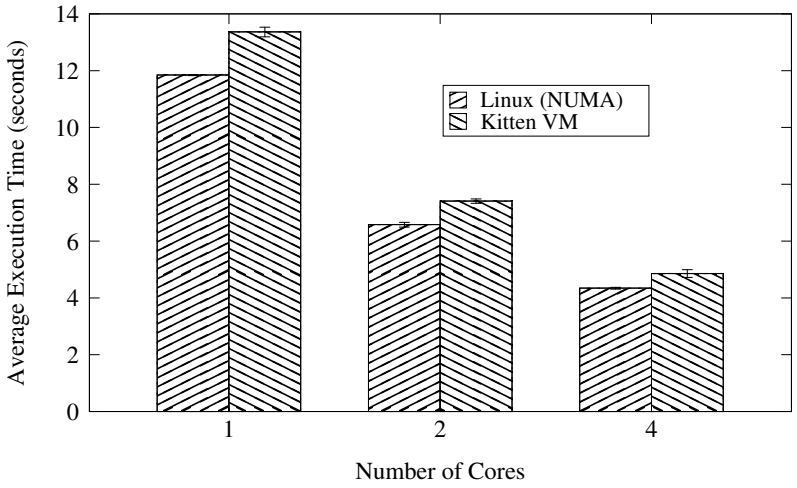
	Mean / Stdev			
Configuration	1 Core	2 Cores	4 Cores	8 Cores
Native Linux	14.01 / 0.94	10.92 / 1.75	6.67 / 0.83	4.54 / 0.09
Native Linux (NUMA)	13.58 / 0.17	7.96 / 0.31	5.07 / 0.10	4.44 / 0.05
Native Kitten	13.40 / 0.06	7.51 / 0.03	5.69 / 0.89	4.36 / 0.02
Kitten VM on Palacios/Linux	13.79 / 0.04	7.85 / 0.16	5.03 / 0.08	4.90 / 0.04

Figure 10: Average execution times for the HPCCG benchmark using single-precision floating point datatypes and 4KB pages

6 Related Work

While a fair amount of work has investigated the feasibility of deploying virtualization on supercomputing platforms, it has always focused on limiting the incurred performance penalties. Our previous work [11, 12] has focused on incorporating the Palacios VMM in a lightweight kernel context, and demonstrating that virtualization can be both scalable and exhibit very low overhead. In [14] the authors presented a light-weight VMM capable of virtualizing the Blue Gene/P platform running IBM's CNK. This work had similar goals in that it sought to provide a compatibility layer to run more standard OSes on top of a lightweight kernel architecture. However, it was tailored to a highly-specialized supercomputer platform, while our approach supports any application in general that requires a lower noise profile and more memory bandwidth than is provided by Linux. Furthermore, we believe that this work is the first to propose the use of virtualization as a means of *improving* performance of a given system for HPC applications. As such, this work marks a significant departure from earlier work investigating the feasibility of virtualization.

While this work has investigated techniques to reduce OS noise and improve memory bandwidth, one issue that we have not yet addressed is that of I/O performance. Recent work has improved virtual I/O performance, reaching near-native levels of performance [**?**]. Of particular interest have been recent advancements to virtual network performance [**?**, **?**]. As previously noted, we have begun work on implementing full passthrough I/O functionality for VMs on Linux hosts, but it is not yet clear how this approach will perform in large scale environments, or on different host operating systems. The ability to scale virtual I/O performance, and network performance in particular, to much larger scales will significantly impact the feasibility of the approaches presented in this work at the supercomputing scale.



	Mean / Stdev		
Configuration	1 Core	2 Cores	4 Cores
Native Linux (NUMA)	11.84 / 0.02	6.58 / 0.09	4.34 / 0.03
Kitten VM on Palacios/Linux	13.34 / 0.17	7.41 / 0.08	4.86 / 0.14

Figure 11: Average execution times for the HPCCG benchmark using single-precision floating point data-types and 2MB pages

7 Conclusion

While it appears that Linux-derived kernels and environments are likely to become the underlying operating systems for future exascale platforms, our work has shown that there is still a significant role that lightweight kernels can play in future supercomputing architectures. We have demonstrated that future exascale systems can have the best of both worlds. HPC applications that can tolerate the overheads can run on a Linux host, whereas those that are significantly impacted by the overheads can run on a virtualized LWK. This approach yields a high degree of usability in that isolated virtual machines can be launched when needed without requiring the reconfiguration of the entire system. Furthermore, the Palacios VMM is fully capable of providing these features for a wide range of Linux kernels while requiring no modifications of the host kernel itself. Our work is freely available in the latest 1.3 versions of Palacios and Kitten.

References

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

[2] Beckman, P., Iskra, K., Yoshii, K., and Coghlan, S. Operating system issues for petascale systems. *ACM SIGOPS Opererating Systems Review 40*, 2 (Apr. 2006).

[3] Bhargava, R., Serebrin, B., Spadini, F., and Manne, S. Accelerating two-dimensional page walks for virtualized systems. In *Proceedings of the 13th international conference on Architectural support for programming languages and operating systems* (2008), ASPLOS XIII.

[4] Brightwell, R. Why nobody should care about operating systems for exascale. In *Proceedings of the 1st International Workshop on Runtime and Operating Systems for Supercomputers* (New York, NY, USA, 2011), ROSS '11, ACM, pp. 1–1.

[5] Burkhardt, J. Sgefa_openmp: Sequential and openmp versions of linpack solver, people.sc.fsu.edu/~jburkhardt/c_src/sgefa_openmp/sgefa_openmp.html.

[6] Dongarra, J. The linpack benchmark: An explanation. In *Proceedings of the 1st International Conference on Supercomputing* (London, UK, UK, 1988), Springer-Verlag, pp. 456–474.

[7] Heroux, M., et al. Welcome to the mantevo project home page, https://software.sandia.gov/mantevo.

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

[9] Kocoloski, B., Ouyang, J., and Lange, J. Dual stack virtulaization: Consolidating hpc and commodity applications in the cloud. In *Proceeedings of the 3rd ACM Symposium on Cloud Computing* (2012), ACM. To appear.

[10] Lange, J., Dinda, P., Hale, K., and Xia, L. An introduction to the palacios virtual machine monitor—release 1.3. Tech. Rep. NWU-EECS-11-10, Department of Electrical Engineering and Computer Science, Northwestern University, October 2011.

[11] Lange, J., Pedretti, K., Dinda, P., Bridges, P., Bae, C., Soltero, P., and Merritt, A. Minimal overhead virtualization of a large scale supercomputer. In *Proceedings of the 2011 ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments (VEE 2011)* (March 2011).

[12] Lange, J., Pedretti, K., Hudson, T., Dinda, P., Cui, Z., Xia, L., Bridges, P., Gocke, A., Jaconette, S., Levenhagen, M., and Brightwell, R. Palacios and kitten: New high performance operating systems for scalable virtualized and native supercomputing. In *Proceedings of the 24th IEEE International Parallel and Distributed Processing Symposium* (April 2010).

[13] Luszczek, P., Dongarra, J., Koester, D., Rabenseifner, R., Lucas, B., Kepner, J., McCalpin, J., Bailey, D., and Takahashi, D. *Introduction to the HPC Challenge Benchmark Suite*, March 2005.

[14] Stoess, J., Appavoo, J., Steinberg, U., Waterland, A., Uhlig, V., and Kehne, J. A lightweight virtual machine monitor for blue gene/p. In *Proceedings of the 1st International Workshop on Runtime and Operating Systems for Supercomputers* (New York, NY, USA, 2011), ROSS '11, ACM, pp. 3–10.

[15] Vaughan, C., VanDyke, J., and Kelly, S. Application performance under different XT operating systems. In *Cray User Group Meeting (CUG 2008)* (2008).

[16] Yoshii, K., Iskra, K., Naik, H., Beckman, P., and Broekema, P. C. Performance and scalability evaluation of 'big memory' on blue gene linux. *International Journal of High Performance Computing Applications 25*, 2 (May 2011).