

Better than Native: Using virtualization to improve compute node performance

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 system for large scale supercomputing platforms [5, 1, 3], 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) [5], 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

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.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

ROSS '12, June 29, 2012, Venice, Italy

Copyright 2012 ACM 978-1-4503-1460-2/12/06 ...\$10.00.

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 in [11], the authors compared CNL against a legacy lightweight OS (Catamount). The results showed that while CNL was able to provide adequate performance (within ~5%) for most applications, there were several cases where CNL's results were on average ~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 lightweight kernels 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 lightweight kernels (LWKs) can be used by a subset of applications that require them. We also believe that a virtualization 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 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 [6]. 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 an 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 [8, 7]. 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 lightweight kernels (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 [6]), 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 but also 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 environment. Contrary to the LWK approach, Linux derived kernels tend to impose their own management architectures

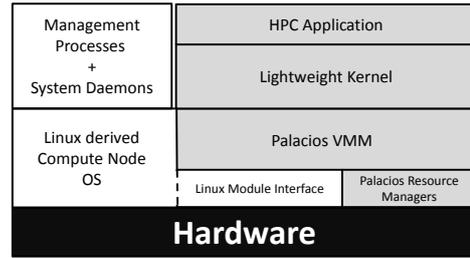


Figure 1: Organization of a virtualized HPC Linux System

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 running host OS at runtime. This approach required that we utilize existing module accessible APIs that remain relatively stable compared to the rest of the kernel. Thus we are able to ensure wide spread 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) [5] 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 code base. 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 2.6.19, a codebase released in late 2006. 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 overly difficult for advances and improvements to be made by the broader HPC systems community.

We claim that, with the Palacios VMM, these drawbacks can be overcome or at least reduced to a large degree. 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 Labs using a version of Palacios that until now has only been developed with a commodity Linux kernel. Additionally, Palacios and Kitten are fully open source¹ and freely available to the entire community. These features mean that even in a fairly restricted environment, the benefits of an 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 [12] 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

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

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 memory², 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 Lightweight Kernel 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 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 Crays 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 an 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 in order to avoid frequent timer interrupts.

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

4. 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 an LWK. For our experiments we focused on two microbenchmarks that directly measure memory performance and OS noise characteristics. In addition we ran two other benchmarks that are more representative of a typical HPC application. We now describe our experimental setup, methodologies, and results.

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. 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 was 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 (lnx), native Linux with NUMA aware processor bindings (lnx-opt), native Kitten (lwk), and Kitten executing inside a Palacios VM on the native Linux OS (v3vee). Each configuration used explicit CPU assignments with the exception of the default Linux environment (lnx). Our virtual machine image was configured with 1GB³ of memory implemented using nested page tables with large page support enabled. 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 view these results as preliminary and part of a feasibility study. In particular, the benchmarks used shared memory and were not configured to be NUMA aware, causing occasional anomalous behavior when executing across NUMA zones.

Methodology.

While our evaluation is limited to a commodity Linux kernel, we contend 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 directly it. 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 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 comparable 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

³At the time of submission, issues arose when creating larger virtual machines. We have since successfully launched virtual machines with up to 4GB of memory

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.

We are currently exploring how to perform the same set of benchmarks using a CNL based host, and intend to provide these results in future work.

4.2 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 [9] to evaluate the memory system performance, as well as the Selfish Detour benchmark [2] from Argonne National Laboratories to characterize the noise profiles. We also ran two other benchmarks that were more representative of HPC applications: pHPCCG from the Mantevo Miniapp collection [4] and an OpenMP enabled HPCCG application acquired from Sandia National Labs. We will now describe each benchmark and their respective results in more detail.

4.3 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 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 is 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

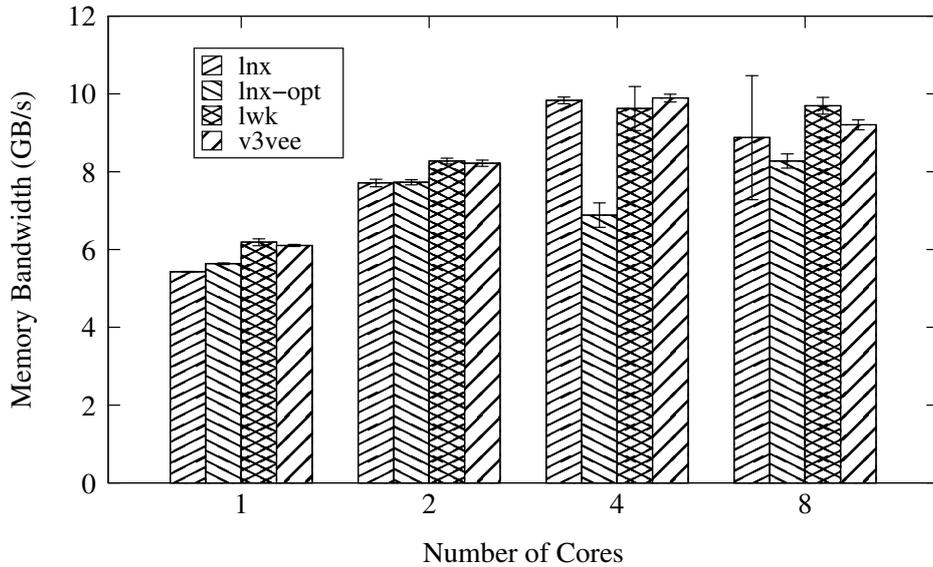


Figure 2: Average memory bandwidths for the Stream benchmark

Configuration	Mean / Stdev			
	1 Core	2 Cores	4 Cores	8 Cores
Linux Host (lnx)	5.43 / 0.01	7.71 / 0.10	9.83 / 0.09	8.88 / 1.59
Linux Host with NUMA control (lnx-opt)	5.64 / 0.02	7.73 / 0.07	6.88 / 0.31	8.27 / 0.18
Kitten (lwk)	6.19 / 0.09	8.27 / 0.07	9.62 / 0.57	9.69 / 0.21
Kitten VM on Linux Host (v3vee)	6.10 / 0.03	8.22 / 0.08	9.89 / 0.10	9.21 / 0.13

Figure 3: Data for the Stream benchmark

a common memory microbenchmark and so probably represent the most optimized code path for each system.

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 Laboratories. 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 clearly exhibits increased 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 take 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 10HZ periodic timer inside the Kitten guest kernel,

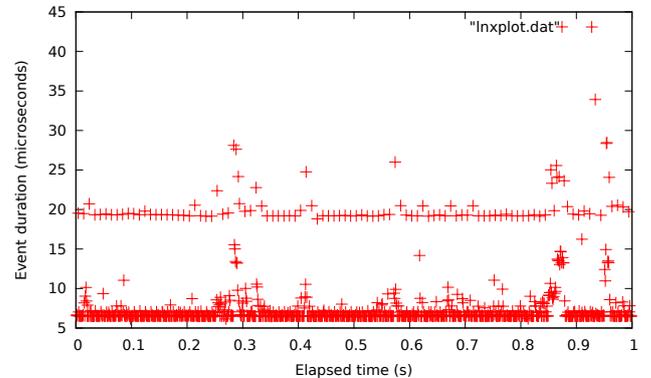


Figure 4: Linux OS noise in the Selfish Detour benchmark

as can be clearly seen in Figure 5. These results show that Palacios can isolate the noise of a Linux host kernel such that it is prevented from interfering with a virtual machine environment.

It should be noted that while Palacios does appear to be capable of isolating the inherent noise of a host Linux kernel, there is a greater degree of low level noise in the virtual environment. On average the background interruptions ex-

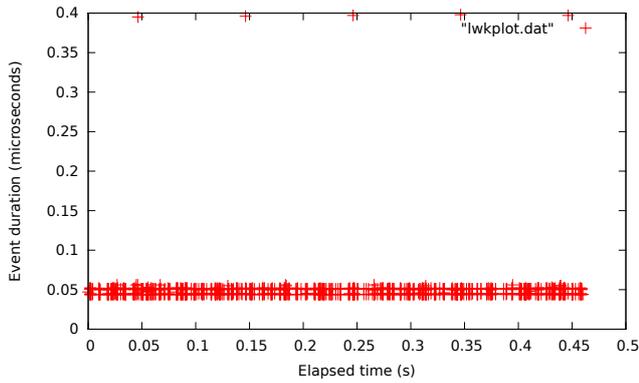


Figure 5: Native Kitten noise in the Selfish Detour benchmark

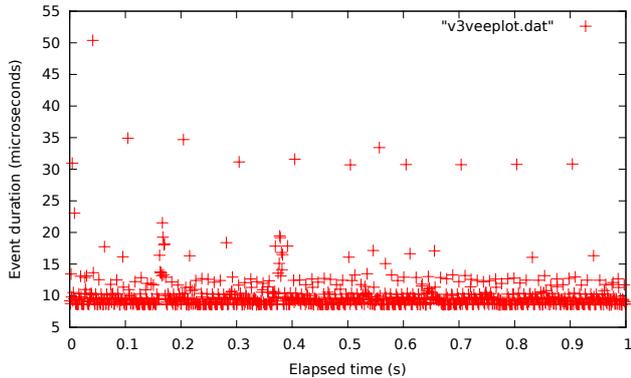


Figure 6: Kitten VM on Linux Host noise in the Selfish Detour benchmark

perienced 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.

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 single-precision floating point types in order to evaluate a more CPU-intensive task. These results are shown as the pHPCCG benchmark in Figure 7. In addition, we also ran HPCCG with a configuration that used double-precision floating point data types which resulted in a more memory-

intensive workload; these results are shown as the HPCCG benchmark in Figure 9.

As shown in Figure 7, 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 indicates that it is likely to exhibit better scalability.

Figure 9 shows the results of the HPCCG benchmark experiments. This benchmark is significant in that it is the only experiment in which Palacios performed consistently worse on average than the native Linux environment. We note however that the performance difference between Linux and the Palacios VM is still fairly constrained, and in fact Palacios was still able to provide performance with less variation than either Linux configuration. This again bodes well for the scaling capabilities of a Palacios based environment.

4.4 Analysis

As our evaluation shows, Palacios is indeed capable of providing superior performance over what can be achieved by a native environment. Our results clearly show that 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 provide Palacios the ability 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 that both have and lack an IOMMU. For non-IOMMU equipped systems we can leverage our earlier work with Symbiotic Passthrough [7].

We are also 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 believe that the approaches presented in this paper will allow us to solve many of these problems, and make cloud based HPC much more palatable for the broader HPC community.

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 [7, 8] has focused on incorporating the Palacios VMM in a lightweight kernel context,

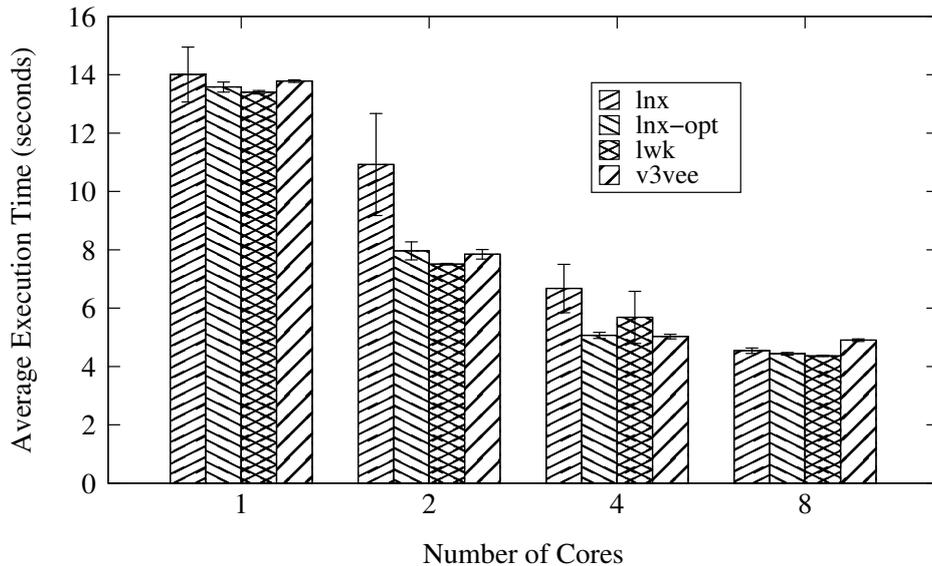


Figure 7: Average execution times for the pHPCCG benchmark using single-precision floating point data-types

Configuration	Mean / Stdev			
	1 Core	2 Cores	4 Cores	8 Cores
Host Linux (lnx)	14.01 / 0.94	10.92 / 1.75	6.67 / 0.83	4.54 / 0.09
Host Linux with NUMA control (lnx-opt)	13.58 / 0.17	7.96 / 0.31	5.07 / 0.10	4.44 / 0.05
Host Kitten (lwk)	13.40 / 0.06	7.51 / 0.03	5.69 / 0.89	4.36 / 0.02
Palacios + Kitten VM (v3vee)	13.79 / 0.04	7.85 / 0.16	5.03 / 0.08	4.90 / 0.04

Figure 8: Data for the pHPCCG benchmark using single-precision floating point data-types

and demonstrating that virtualization can be both scalable and exhibit very low overhead. In [10] 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, 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.

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 provid-

ing 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.

8. 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 Operating Systems Review* 40, 2 (Apr. 2006).
- [3] 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.
- [4] HEROUX, M., ET AL. Welcome to the mantevo project home page, <https://software.sandia.gov/mantevo>.
- [5] KAPLAN, L. Cray CNL. In *FastOS PI Meeting and Workshop* (June 2007).
- [6] 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.

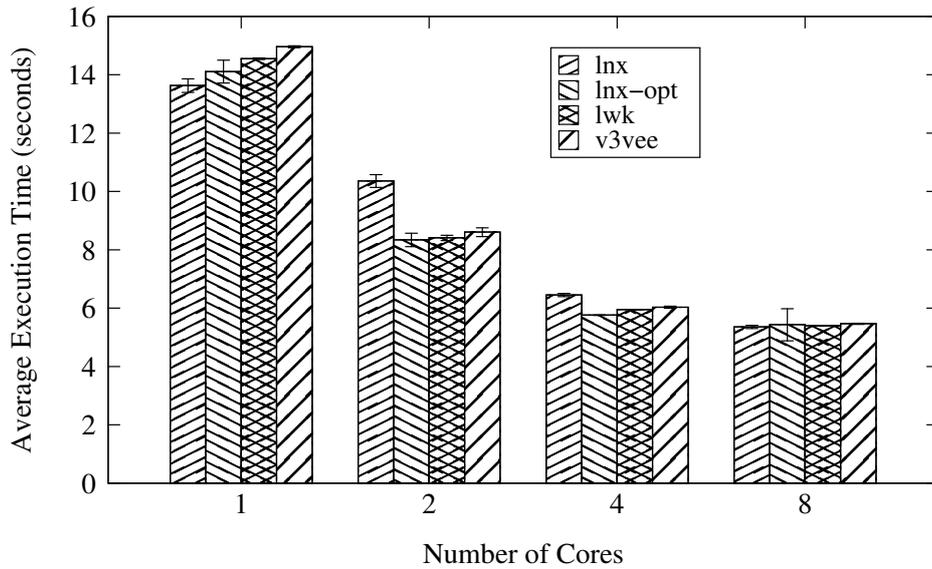


Figure 9: Average execution times for the HPCCG benchmark using double-precision floating point data-types

Configuration	Mean / Stdev			
	1 Core	2 Cores	4 Cores	8 Cores
Host Linux (lnx)	13.62 / 0.23	10.36 / 0.21	6.45 / 0.06	5.36 / 0.06
Host Linux with NUMA control (lnx-opt)	14.11 / 0.39	8.33 / 0.23	5.76 / 0.01	5.43 / 0.55
Host Kitten (lwk)	14.55 / 0.02	8.41 / 0.09	5.95 / 0.01	5.39 / 0.005
Palacios + Kitten VM (v3vee)	14.95 / 0.03	8.61 / 0.15	6.03 / 0.03	5.46 / 0.02

Figure 10: Data for the HPCCG benchmark using double-precision floating point data-types

- [7] 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).
- [8] 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).
- [9] 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.
- [10] STOESS, J., APPAVOO, J., STEINBERG, U., WATERLAND, A., UHLIG, V., AND KEHNE, J. A light-weight 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.
- [11] VAUGHAN, C., VANDYKE, J., AND KELLY, S. Application performance under different XT operating systems. In *Cray User Group Meeting (CUG 2008)* (2008).
- [12] 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).