Effects of Virtualization on a Scientific Application

Running a Hyperspectral Radiative Transfer Code on Virtual Machines

Anand Tikotekar, Geoffroy Vallée,
Thomas Naughton, Hong Ong,
Christian Engelmann & Stephen L. Scott *
Computer Science and Mathematics Division
Oak Ridge National Laboratory
Oak Ridge, TN, USA.

Anthony M. Filippi †
Department of Geography
Texas A&M University
College Station, TX, USA.

Abstract

The topic of system-level virtualization has recently begun to receive interest for high performance computing (HPC). This is in part due to the isolation and encapsulation offered by the virtual machine. These traits enable applications to customize their environments and maintain consistent software configurations in their virtual domains. Additionally, there are mechanisms that can be used for fault tolerance like live virtual machine migration. Given these attractive benefits to virtualization, a fundamental question arises, *how does this effect my scientific application?* We use this as the premise for our paper and observe a real-world scientific code running on a Xen virtual machine. We studied the effects of running a radiative transfer simulation, *Hydrolight*, on a virtual machine. We discuss our methodology and report observations regarding the usage of virtualization with this application.

1. Introduction

Virtualization is being used in commercial settings for server consolidation. Virtual machines enable applications to run in hosted, non-native environments which can offset initial porting issues to new platforms or provide a basis for research testbeds. Recently there has been increasing interest to use virtualization in the area of high-performance computing (HPC), in part, to provide consistent and/or customizable operating environments (3; 5; 8; 16; 17). Additionally, there is interest in leveraging virtualization to address issues of fault tolerance in HPC by making use of techniques like live migration of virtual machines (6; 13).

These interesting capabilities and their use with HPC are often discussed from a strictly systems perspective, often using synthetic benchmarks to display the overheads of virtual machines. Therefore, the developers of scientific codes must rely on synthetic metrics in order to gauge the costs of virtualization.

In this paper we investigate the use of virtual machines for a real-world scientific application. The intent being to provide some insight for scientists interested in employing virtualization for their research. We discuss our methodology and present observations from running a hyperspectral radiative transfer code on both native Linux and Xen virtual machines (1).

2. Background

The application, *Hydrolight*, used in our experimentation was selected based on prior work developing a set of tools, *HydroHPCC*, for running the code on a cluster of workstations (4). The main objective was to decrease the overhead involved in creating input parameters for the simulations, and to reduce the wall clock time for the sequential application by performing runs in a batch parallel fashion.

The Hydrolight (Sequoia Scientific, Inc.) radiative-transfer numerical model (9; 10) solves the radiative transfer equation to determine the radiance distribution within and leaving a water body. Spectral radiance is generated as a function of depth within the water column, wavelength, and direction. Other quantities can be derived from the radiance, such as irradiance and reflectance values (15). For instance, Hydro-

* ORNL's work was supported by the U.S. Department of Energy, under Contract DE-AC05-00OR22725.

† This research was supported in part by an appointment to the U.S. Department of Energy (DOE) Higher Education Research Experiences (HERE) for Faculty at the Oak Ridge National Laboratory (ORNL) administered by the Oak Ridge Institute for Science and Education. A. M. Filippi also thanks Budhendra L. Bhaduri and Eddie A. Bright, Computational Sciences & Engineering Division, ORNL, for their support.

2nd Workshop on System-level Virtualization for High Performance Computing (HPCVirt'08) 31 March 2008, Glasgow, Scotland.

light can be utilized to calculate spectral remote-sensing reflectance, given water-column inherent optical properties (IOPs), as well as various other ocean and atmospheric quantities.

Hydrolight has a variety of uses, including primary productivity and underwater visibility studies; remote-sensing mission planning and algorithm evaluation; modeling contributions to remote sensor signals; and enhancing understanding of physical processes (15).

The Hydrolight code employs invariant imbedding, which, relative to other methods (e.g., Monte Carlo simulation or discrete ordinates), is very fast. Unlike Monte Carlo methods, where computation time exponentially increases with depth, Hydrolight compute time depends linearly on the depth to which radiance is generated. Also, in contrast with discrete ordinates, invariant imbedding is nearly independent of IOP depth variance (9; 15). However, if a large number of runs is required, such as with remote-sensing inversion model development, such an undertaking is computationally expensive (11).

3. Methodology

We ran the Hydrolight simulations on both native and virtual machines in order to better understand the overheads of virtualization. Our approach was to use the widely available OProfile tool to investigate the performance across both platforms. This section describes the simulations and tools used in our experimentation, which is discussed in Section 4.

3.1 Simulations

In the previous work we used the application to perform 2,600 simulations in parallel on a small cluster of workstations to generate training data for an artificial neural network (4). Such pseudo-data can also be used in developing various other types of remote-sensing inversion models, e.g., (11).

Each simulation is a single execution of the model for a given set of parameters, which are provided at startup via an input file. The wall-clock time to run these 2,600 simulations in parallel was approximately 3 hours using 42 compute nodes natively. The simulations themselves are deterministic but the wall-clock time from our experiments vary based upon the input parameters, e.g., depth, ranging from 1 minute to 10 minutes. Since a simulation with the same parameters takes roughly the same time[1] we selected a single experiment from the group with the longest running time, see Figure 1. This reduces the time to perform the experiments but the analysis should be applicable for the larger set of runs. Also, this greatly simplifies the post-processing and analysis when running the experiment on the target platforms.

[1] Any fluctuations should be due to system "jitter" and not the application itself.

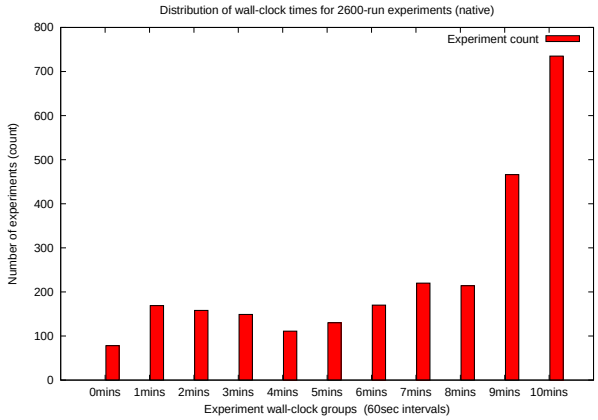

Figure 1. Distribution of wall-clock times for the 2,600 experiments (2600-run) run natively under Linux.

3.2 Tools

Our goal was to study the costs of running these simulations on virtual machines. To try and better understand the execution of the application on both the native and virtualized systems we chose to use the Linux system profiler, *OProfile* (14). This enabled us to use a similar approach for gathering runtime metrics for the entire system for both platforms.

The cluster used for our testing has identical hardware on all compute nodes: 2Ghz Pentium IV, 768MB of memory and 100Mb FastEthernet. The nodes were running Fedora Core 5 (FC5) and the same kernel release version 2.6.16.33 was used for the native and para-virtualized instances of Linux. We used the Xen hypervisor version 3.0.4 for our testing, which has built-in support for OProfile. The systems were configured with OProfile version 0.9.1 and the application was compiled using the GNU Fortran compiler (g77) version 3.2.3.

3.3 Run Parameters

The OProfile samples were taken with kernel and kernel modules shown separately. The actual OProfile command line used for the experiments was:

```
opcontrol --start --separate=kernel \
   --event=GLOBAL_POWER_EVENTS:100000:1:1:1 \
   --event=ITLB_REFERENCE:100000:2:1:1 \
   --event=INSTR_RETIRED:100000:1:1:1 \
   --event=MACHINE_CLEAR:100000:1:1:1 \
   --vmlinux=/opt/vmlinux_location/vmlinux \
```

This gathered the following OProfile samples but we only studied the first two metrics because we wanted to focus on the actual time spent by the application and its relationship with the ITLB miss rate.

1. `GLOBAL_POWER_EVENTS: time during which processor is not stopped`
2. `ITLB_REFERENCE: translations using the instruction translation lookaside buffer; 0x02 ITLB miss`
3. `INSTR_RETIRE: retired instructions; 0x01 count non-bogus instructions which are not tagged`
4. `MACHINE_CLEAR: cycles with entire machine pipeline cleared; 0x01 count a portion of cycles the machine is cleared for any cause`

The OProfile output provides a table of samples gathered from the system over a period of time. The table is sorted in decreasing order with the event listed for each *symbol* name and its associated executable *image* and caller (*application*). For example, the following shows excerpts of an OProfile listing taken from a run of Hydrolight (`maincode.exe`) on a native Linux system:

```
samples   image name     app name        symbol name
9877360   maincode.exe   maincode.exe    rhotau_
    ....
140760    libm-2.4.so    maincode.exe    cos
    ....
10129     vmlinux        maincode.exe    init_pmtmr
```

The output from OProfile is system-wide, therefore in order for us to isolate the data directly associated with our application we performed some basic post-processing to extract these quantities. We applied a set of basic heuristics to classify the OProfile data for the native Linux and Xen system into time spent running either user (T_{usr}) or system (T_{sys}) code. The T_{usr} and T_{sys} classes were also tagged to give further indication as to where the costs were to be attributed. These heuristic classifications were based upon the contents of the columns labeled *image name* and *app name* with a synopsis shown in Table 1. Note, a few special cases were also tagged in the OProfile output: (i) "anon(XXXX...)" entries where no symbol information is accessible, and (ii) the costs for OProfile itself.

image name	app name	Time Class
maincode.exe	maincode.exe	T_{usr} ($T_{usr.app}$)
lib*	maincode.exe	T_{usr} ($T_{usr.lib}$)
ld-*	maincode.exe	T_{usr} ($T_{usr.lib}$)
oprofiled	*	T_{sys} ($T_{sys.prof}$)
oprofile.ko	*	T_{sys} ($T_{sys.prof}$)
*.ko	maincode.exe	T_{sys} ($T_{sys.os}$)
vmlinux	maincode.exe	T_{sys} ($T_{sys.os}$)
xen-syms	maincode.exe	T_{sys} ($T_{sys.vmm}$)
anon(XXXX...)	maincode.exe	*untrackable*

Table 1. Post-processing heuristics applied to OProfile data shown using '`*`' as a wildcard matching anything.

Our post-processing technique has two known limitations. Firstly, we are not able to completely account for T_{sys} costs associated with the hostOS (dom0) when running the application in the VM (domU). This is because there is no way to correlate the costs in the hostOS (dom0) that are specific to our application (in VM), i.e., the application context in the VM is not visible from the hostOS when OProfile reports an event. Secondly, we do not account for some auxiliary overheads caused by Xen, e.g., Python `xend` daemon and associated system/shell invocations. We noticed events like `modulecmd` which we believe are related to the use of Xen but are not obviously tied from the output displayed by OProfile. Therefore, the accounting for a virtual machine's T_{sys} is low as the hostOS portion is not included. To try and give some indication of the cost entirely due to virtualization we include the times for running the application on the hostOS itself. In this context, the differences between native and hostOS are entirely due to virtualization but are not identical to running the application in an unprivileged VM (domU).

4. Experimentation

In this section, we describe our experimental results. As explained in Section 3, we selected the longest running experiment from our set of 2,600 simulations. We also mention some preliminary numbers from running all 2,600 simulation on a cluster using virtualization.

We executed the selected simulation (experiment) 20 times (runs) over three platforms: Native, HostOS and virtual machine (VM). Figures 2-7 on page 5 show the breakdown of CPU as well as ITLB samples from OProfile on the respective platforms for the individual runs. As described in Section 3, samples from OProfile are based on a frequency of 100,000 clock cycles. OProfile extracts CPU samples from `GLOBAL_POWER_EVENTS` events, and ITLB samples from `ITLB_REFERENCE` events. To summarize, greater number of samples indicates more time spent in the respective code symbol for a particular event.

Platform	cpu avg	cpu std	% std	tlb avg	tlb std	% std
Native	12687854	31740	0.25	3007	32	1.06
HostOS	13162659	50532	0.38	1296	37	2.85
VM	13156140	84812	0.64	1299	39	3.00

Table 2. Average and standard deviations for user level (T_{usr}) CPU and ITLB miss samples from one experiment (file) over 20 runs.

Platform	cpu avg	cpu std	% std	tlb avg	tlb std	% std
Native	92984	16908	18.18	69	10	14.49
HostOS	823407	21270	2.58	6475	123	1.89
VM	792183	23082	2.91	6340	134	2.11

Table 3. Average and standard deviations for system level (T_{sys}) CPU and ITLB miss samples from one experiment (file) over 20 runs. Note, the VM's T_{sys} only contains domU portion.

As can be seen from Figures 2-7, the individual runs are quite consistent. However, a few things stand out. First, the standard deviation on the Native platform, for system CPU samples is quite high across 20 runs as reflected by Table 3 with a value of 18.18%. Second, on the Native platform, the standard deviation for system ITLB samples is also high (14.49%) as shown by Table 3. Lastly, when contrasting CPU samples between Native and VM, based on Table 2 the variance of code running in user mode is 0.39%

lower on native and based on Table 3 is 15.27% higher when running in system mode. The variability for native's system mode may be related to our accounting techniques as in some instances the `ide_outsw` symbol was not associated to the "maincode.exe" application but instead "vmlinux" and therefore the samples were not included. Note, this behavior was unique to 2-3 nodes in the test cluster.

The individual runs furnished us with an opportunity to study the variability of similar runs. We would have expected such variability from virtual environments due to the additional complexities added by the virtualization layer. However, the variability in the wall clock times (Table 4) of the same application is less in native than virtual environments. One explanation for this is less variability in the user context for the native as compared to the virtual environments.

Figures 8 and 9 present a summary view of the application for the different platforms. The values in these graphs are based upon averages taken from the individual runs (Figures 2-7). We observe from Figures 8 and 9 that Native outperforms the VM. Specifically, the VM runs 11% slower than Native, with VMs slightly better than the HostOS. Yet, the wall-clock times shown in Table 4 show that running the application on the HostOS, on average, performs marginally better than running the application on the VM. The reason behind this apparent paradox is that, with OProfile in its current form, it is unclear to us how to fully account for the system portion of an application running in the VM (domU). This is due to the fact that when working with OProfile the samples are split into two domains, namely the VM proper (domU) and underlying hostOS (dom0). Since the application is only present in the VM (domU) there are no apparent means to systematically determine the application specific contributions for T_{sys} that lie in the underlying hostOS (dom0) portion. For example, the I/O operations managed by the hostOS (dom0) for a VM (domU) would be part of the data in the "dom0" OProfile file. This shortcoming is only applicable to OProfile samples related to system code for a VM.

Platform	Avg	Min	Max
Native	692.25	690	697
HostOS	771.15	761	782
VM	772.05	763	790

Table 4. Wall clock times (seconds) for one experiment (file) over 20 runs for each platform: Native, HostOS and Virtual Machine.

Further, on all three platforms most of the CPU time is spent in the user context. However, for ITLB misses, the case is not so obvious. On Native, the user context accounts for most of the ITLB misses, but this is not the case for the HostOS or VM. On the HostOS and VM the system side accounts for most of the ITLB misses.

To correlate and quantify Figures 8 and 9, we refer our reader to Tables 5 and 6. Table 5 shows the comparison between Native and VM for CPU ticks.

Table 5 shows that the ratio of user code CPU samples between native and virtual is 0.96, implying that user code is performing slightly faster on native than on virtual. However, Table 6 shows that the ratio of user code ITLB miss samples between native and virtual is 2.31, implying that there are much more ITLB misses on native when running user code. This apparent counter behavior with ITLB misses on native is challenged in Table 8 in Section 6.

Platform relation	Average
N:V T_{usr} / T_{usr}	0.96
N:N T_{usr} / T_{sys}	136
V:V T_{usr} / T_{sys}	16.6
N:V T_{sys} / T_{sys}	0.11

Table 5. Average of one experiment (file) over 20 runs showing the ratios for user and system level CPU time samples, where N=native and V=VM. Note, T_{sys} only contains domU portion.

Platform relation	Average
N:V T_{usr}/T_{usr}	2.31
N:N T_{usr}/T_{sys}	43.27
V:V T_{usr}/T_{sys}	0.20
N:V T_{sys}/T_{sys}	0.010

Table 6. Average of one experiment (file) over 20 runs showing the ratios for user and system level ITBL miss samples, where N=native and V=virtual. Note, T_{sys} only contains domU portion.

Other ratios from Table 5 and 6 are more intuitive. For example, Table 5 shows that the User to System ratio for Native is much higher than that of virtual (43.27), supporting the fact that system code is more expensive in the virtual environments. Further, Table 6 shows that the number of Native ITLB miss samples is higher in user mode as compared to system mode (43.27), and conversely ITLB miss samples are higher in system than user mode (0.20) for the virtual machine.

Lastly, to get a rough estimate of the time to solution when using virtualization we performed the entire 2,600 experiments on all three platforms: Native, HostOS and VM, varying the number of available compute nodes. The execution times[2] for running the 2,600 experiments over these platforms are given in Figure 10. This shows that the difference in wall-clock time between the native and virtual systems is roughly 8%. It is unclear why there is a slight dif-

[2] These numbers were taken without OProfile and on a slightly earlier cluster configuration where the native kernel version did not exactly match that used for the para-virtualized Linux kernel; but this is still using Xen version 3.0.4

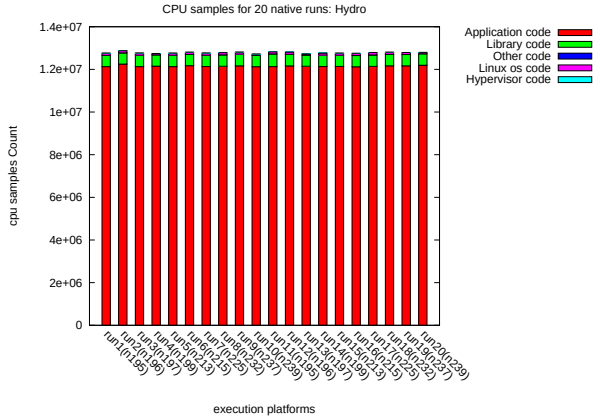

Figure 2. Breakdown of CPU samples for 20 runs on Native.

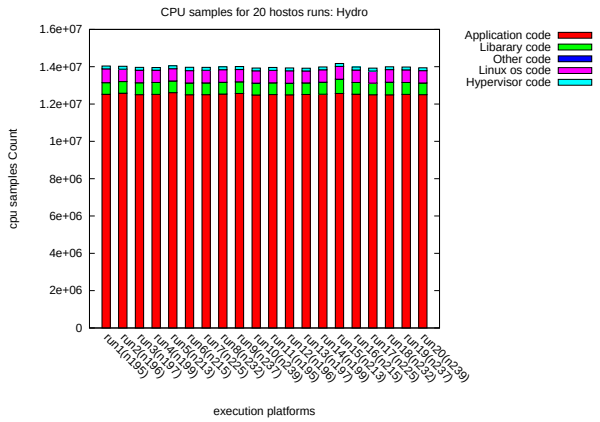

Figure 3. Breakdown of CPU samples for 20 runs on HostOS.

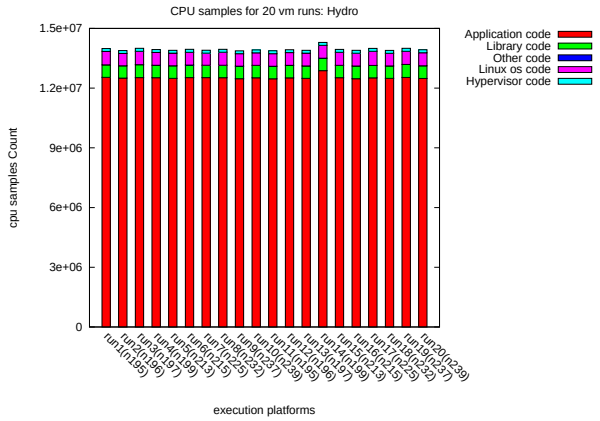

Figure 4. Breakdown of CPU samples for 20 runs on VM. Note, T_{sys} only contains domU portion.

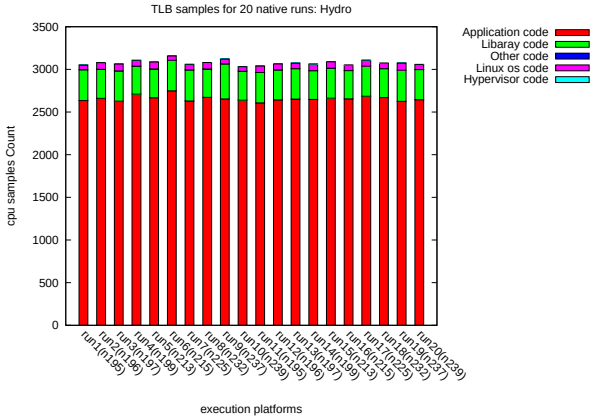

Figure 5. Breakdown of ITLB miss samples for 20 runs on Native.

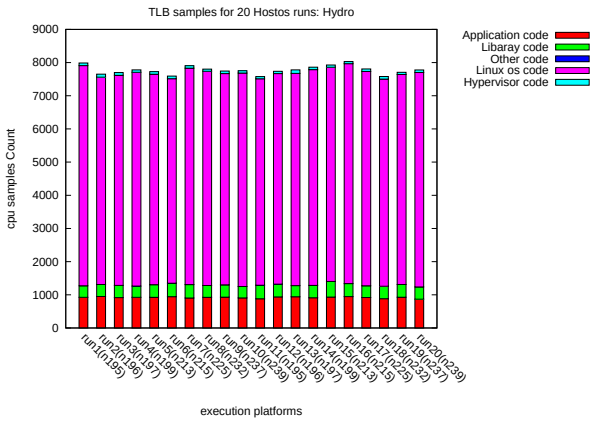

Figure 6. Breakdown of ITLB miss samples for 20 runs on HostOS.

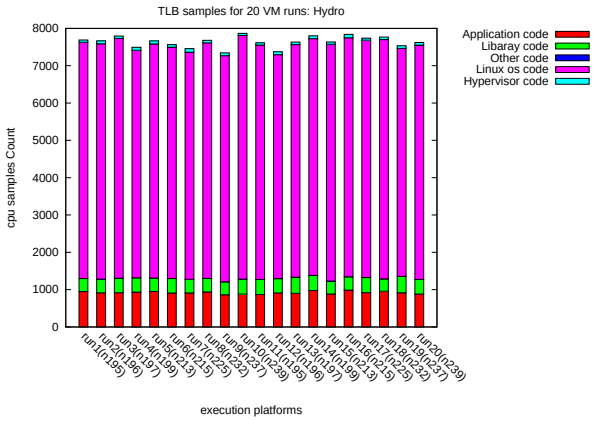

Figure 7. Breakdown of ITLB miss samples for 20 runs on VM. Note, T_{sys} only contains domU portion.

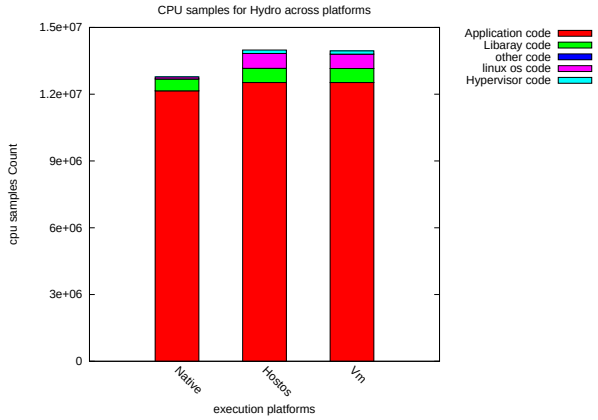

Figure 8. Average CPU samples using 20 runs for each platform. Note, the VM's T_{sys} only contains domU portion.

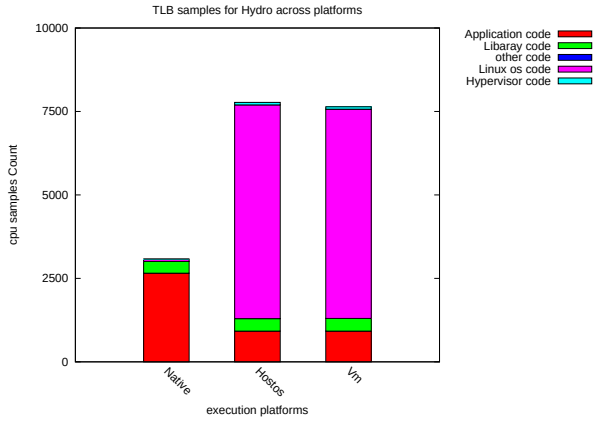

Figure 9. Average ITLB miss samples using 20 runs for each platform. Note, the VM's T_{sys} only contains domU portion.

ference between the hostOS and guestOS, which was not the trend in our other measurements.

5. Related Work

Our work is closely related to (7). Work reported in (7) is one of the first such studies, to the best of our knowledge, to diagnose the performance overheads in Xen using Xenoprof, which they developed, and has been described in their paper. We have used a similar methodology but differ in focus and the type of applications evaluated. The authors in (7) focus on networking applications in a VM running in uni-processor as well as multi-processor systems. Our focus is not just on networking applications, but on real world scientific applications. While the paper in (7) aims to use the information extracted using Xenoprof to uncover bugs and channel the information into optimizing Xen, we hope to provide a scientist or an application's user a view of their application on virtual machines. We hope that a scientist would

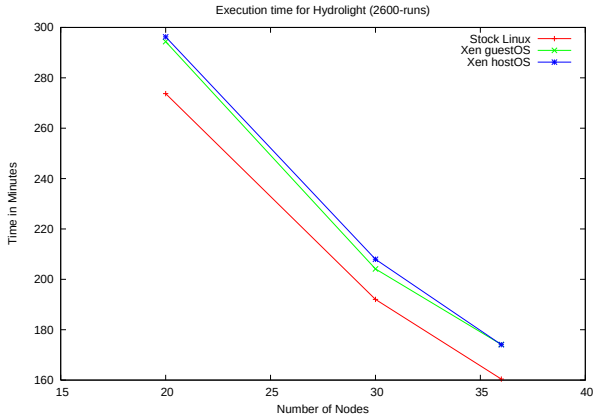

Figure 10. Execution time for Hydrolight experiments (2600-run) using stock Linux, Xen guestOS and Xen hostOS over a range of compute nodes.

then be able to decide whether to run his/her application on virtual machines.

Authors in (17) study the impact of para-virtualization on HPC systems and applications. They study many benchmark applications including HPCC for a full system evaluation. Their conclusion that para-virtualization, specifically Xen, does not introduce significant overhead is based on the performance results of various benchmarks and real world applications.

While we have a similar objective, our approach goes beyond just the performance results, in that we aim to analyze the various overheads associated with applications when running in native as well as virtual environments.

Authors in (5) describe a framework for performance and management overheads in virtual environments. They specifically study VMM I/O bypass and VM image managements. For performance evaluation, they study different MPI tests and parallel HPC benchmarks such as HPL and NAS. Their conclusion also concurs with that of authors in (17). Although the authors in (5) use Xenoprof like we do, they do not use real world scientific applications.

Work in (2) describes another useful study evaluating virtual environments for their efficacy to run HPC applications. The authors study Xen and UML as virtual environments with HPCC as their HPC benchmark applications. The conclusion in their paper (2) is consistent with the other related work reported here, in that authors credit Xen to perform with little overhead in HPC context. They however maintain that more such analysis and evaluation is required. The work in (2) does not describe real world scientific applications, and their focus is on virtual environments being able to run HPC applications efficiently.

The VIVA project at UCSB has developed an enhanced profiler called *VIProf* (12). Their tool extends the system wide profiler *OProfile* to support dynamic code segments for a language level virtual machine, e.g., Java Virtual Ma-

chine (JVM). The modifications enable the profiler to identify the dynamically compiled code from the JVM image that is regularly reported as anonymous by OProfile. The tool as presented in (12) is specific to a Java virtual machine and the dynamic code found in such VMs. They mention longer term goals of working with Xen, which would be more relevant to our efforts with system-level virtualization. This work highlights a similar issue as encountered in our analysis regarding access to inter-domain context for profiling, e.g., host/guest domains.

6. Future Work

We have evidence that, profiles obtained using Xenoprof in its current form, can be difficult to interpret. We mentioned the problem of attributing samples when the privileged domain (dom0) is working on behalf of the processes running in various unprivileged domains, in Section 4. There is, however, another issue with regards to interference caused by events that are profiled together. We profiled four events together as shown in Section 3, and reported results based on the profiles generated using those four events in tandem. While this is an acceptable approach on the native system, in the virtualized environment the results are not consistent when working with OProfile/Xenoprof.

The Table 7 and Table 8, paint a different picture when profiles are obtained for single events – as opposed to multiple events gathered simultaneously.

Platform	Avg User	Avg System
Native	12009823	81559
HostOS	12781305	176387
VM	12697736	163400

Table 7. Average number of samples for profiling the single event `GLOBAL_POWER_EVENTS`, for one experiment (file) over 10 runs.

Platform	Avg User	Avg System
Native	22	7
HostOS	38	45
VM	38	35

Table 8. Average number of samples for profiling the single event `ITLB_REFERENCE (miss_samples)`, for one experiment (file) over 10 runs.

The Table 7 and Table 8 show that, although the overhead values change between separate (single) and simultaneous (multiple) event profiling for `GLOBAL_POWER_EVENTS`, (please refer to Table 2 and Table 3 for simultaneous event profiling results) the behavior of the overheads is preserved on the native platform. But the same is not true for virtual environments, where we see changes in overhead behavior. Therefore, as underscored in our conclusion, much more investigation is needed both to understand the overheads in virtual environments and accordingly enhance the standard tools.

Although, the results mentioned in this section, are not in total accordance with the results mentioned in Section 4, we have not been able to identify the root cause for the difference in overhead behavior of the profiles obtained using Xenoprof. Therefore, we include it as preliminary data and plan to further investigate the matter in our future work.

7. Conclusion

In this paper we analyzed a hyperspectral radiative transfer code, *Hydrolight*, and the overheads associated with its use on both native and virtual machines. The wall-clock time was approximately 8% more when running on the virtualized system without instrumentation and the overhead was close to 11% during the profiled execution of an individual simulation. However, the overhead drops to 8% when profiles are obtained for a single event, e.g, just `GLOBAL_POWER_EVENTS`.

Our study was motivated by the fact that there is a growing interest in using virtualization in HPC environments. However, adoption by the HPC community is not yet wide spread. This is in part due to virtualization overheads, which may be perceived as too high in an HPC context. Due to this delay, there is a lack of information regarding the actual distributions for overheads associated with virtualization in real-world scientific applications. This is further complicated by the fact that standard tools are immature in their support for virtualization and work is needed to aid the analysis of virtualization. Further, as we showed in Section 6, there is a nontrivial interference caused by different events, and thus implies that performance isolation is still a difficult task especially in virtual environments.

References

[1] Paul Barham, Boris Dragovic, Keir Fraser, Steven Hand, Tim Harris, Alex Ho, Rolf Neugebauer, Ian Pratt, and Andrew Warfield. Xen and the art of virtualization. In *Proceedings of the nineteenth ACM symposium on Operating System s Principles (SOSP19)*, pages 164–177. ACM Press, 2003.

[2] W. Emeneker and D Stanzione. HPC Cluster Readiness of Xen and User Mode Linux. In *IEEE International Conference on Cluster Computing*, September 2006.

[3] Christian Engelmann, Stephen L. Scott, Hong Ong, Geoffroy Vallée, and Thomas Naughton. Configurable Virtualized System Environments for High Performance Computing. In *Proceedings of the* 1^{st} *Workshop on System-level Virtualization for High Performance Computing (HPCVirt'07), held in conjunction with the ACM EuroSys'07*, Lisbon, Portugal, March 20, 2007.

[4] Anthony M. Filippi, Budhendra L. Bhaduri, Thomas Naughton, Amy L. King, and Stephen L. Scott. High-performance cluster computing-based approach to hyperspec-

tral aquatic radiative transfer modeling. *(In preparation) Photogrammetric Engineering & Remote Sensing (PE&RS)*, 2008.

[5] W. Huang, J. Liu, B. Abali, and D.K. Panda. A Case for High Performance Computing with Virtual Machines. In *20th ACM International Conference on Supercomputing (ICS '06) Cairns, Queensland, Australia*, June 2006.

[6] Wei Huang, Qi Gao, Jiuxing Liu, and Dhabaleswar K. Panda. High Performance Virtual Machine Migration with RDMA Over Modern Interconnects. In *Proceedings of IEEE Cluster (Cluster'07)*. IEEE, September 17-20, 2007.

[7] A. Menon, J. R. Santos, Y. Turner, G. Janakiraman, and W. Zwaenepoe. Diagnosing performance overhead in the xen virtual machine environment. In *Proceedings of the 1st ACM Conference on Virtual Execution Environments*, June 2005.

[8] Mark F. Mergen, Volkmar Uhlig, Orran Krieger, and Jimi Xenidis. Virtualization for High-Performance Computing. *SIGOPS Oper. Syst. Rev.*, 40(2):8–11, 2006.

[9] Curtis D. Mobley. *Light and Water: Radiative Transfer in Natural Waters*. Academic Press, San Diego, 1994.

[10] Curtis D. Mobley, Bernard Gentili, Howard R. Gordon, Zhonghai Jin, George W. Kattawar, Andre Morel, Phillip Reinersman, Knut Stamnes, and Robert H. Stavn. Comparison of numerical models for computing underwater light fields. *Applied Optics*, 32(36):7484, 1993.

[11] Curtis D. Mobley, Lydia K. Sundman, Curtiss O. Davis, Jeffrey H. Bowles, Trijntje Valerie Downes, Robert A. Leathers, Marcos J. Montes, William Paul Bissett, David D. R. Kohler, Ruth Pamela Reid, Eric M. Louchard, and Arthur Gleason. Interpretation of hyperspectral remote-sensing imagery by spectrum matching and look-up tables. *Applied Optics*, 44(17):3576–3592, 2005.

[12] Hussam Mousa, Chandra Krintz, Lamia Youseff, and Rich Wolski. VIProf: Vertically integrated full-system performance profiler. In *Proceedings of the Workshop on Next-Generation Software (NGS)*, March 2007. Co-hosted with IPDPS 2007.

[13] Arun Babu Nagarajan, Frank Mueller, Christian Engelmann, and Stephen L. Scott. Proactive fault tolerance for HPC with Xen virtualization. In *ICS '07: Proceedings of the 21st annual international conference on Supercomputing*, pages 23–32, New York, NY, USA, 2007. ACM Press.

[14] OProfile: A system-wide profiler for Linux. Available at: http://oprofile.sourceforge.net. (Last visited: Feb. 2008).

[15] Sequoia Scientific, Inc., Redmond, WA, USA. *Hydrolight 4.2*, 2000.

[16] Geoffroy Vallée, Thomas Naughton, and Stephen L. Scott. System Management Software for Virtual Environments. In *Proceedings of the ACM International Conference on Computing Frontiers (CF 2007)*, Ischia, Italy, May 7-9, 2007.

[17] Lamia Youseff, Rich Wolski, Brent Gorda, and Chandra Krintz. Paravirtualization for HPC Systems. In *ISPA Workshop on XEN in HPC Cluster and Grid Computing Environments (XHPC'06)*, pages 474–486, December 2006.