

# An Analysis of HPC Benchmarks in Virtual Machine Environments <sup>\*</sup>

Anand Tikotekar, Geoffroy Vallée, Thomas Naughton, Hong Ong, Christian Engelmann, Stephen L. Scott

Oak Ridge National Laboratory  
Computer Science and Mathematics Division  
Oak Ridge, TN 37831, USA

{tikotekaraa, valleegr, naughtont, hongong, engelmann, scottsl}@ornl.gov

**Abstract.** Virtualization technology has been gaining acceptance in the scientific community due to its overall flexibility in running HPC applications. It has been reported that a specific class of applications is better suited to a particular type of virtualization scheme or implementation. For example, Xen has been shown to perform with little overhead for compute-bound applications. Such a study, although useful, does not allow us to generalize conclusions beyond the performance analysis of that application which is explicitly executed. An explanation of why the generalization described above is difficult, may be due to the versatility in applications, which leads to different overheads in virtual environments. For example, two similar applications may spend disproportionate amount of time in their respective library code when run in virtual environments. In this paper we aim to study such potential causes by investigating the behavior and identifying patterns of various overheads for HPC benchmark applications. Based on the investigation of the overhead profiles for different benchmarks, we aim to address questions such as: Are the overhead profiles for a particular type of benchmarks (such as compute-bound) similar or are there grounds to conclude otherwise.

## 1 Introduction

Increasingly, HPC applications are being deployed on virtual environments such as Xen. The reason for such a trend is that the flexibility provided by virtual environments, such as the ability to facilitate fault-tolerance, could balance any performance costs. Indeed, many studies have indicated that performance penalty arising from virtualization schemes is not significant. Furthermore, research has established that I/O bound applications incur more performance penalty on Xen than compute-bound applications. Yet, we cannot generalize such a performance conclusion even for similar applications only on the basis of a final performance number. For example, it is possible that two different performance overhead

---

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

profiles may ultimately post a similar performance penalty, but for different reasons. Thus, the problem of predicting performance for applications is difficult, and becomes even more difficult in virtual environments due to its complexity.

The complexity inherent in virtual environments can lead to unpredictable application performance such as incurring disproportionate overhead when code contribution is changed (for example, increase in the user code may be impacted disproportionately). Further, the impact of events such as ITLB, DTLB, and cache misses can also contribute towards the difficulty of performance prediction due to an indirection layer of virtualization. But the problem can be alleviated by studying the details of the impact of virtualization on applications.

Understanding the details about the impact of virtualization on HPC application is useful for the following reasons: First, it can uncover an application's behavior in virtual environments. Second, it allows us to identify sources of various overhead costs. Third, insights such as, two applications having different behavior in virtual environments perform with similar performance costs, are possible. Fourth, by analyzing the impact of virtualization, we can state more confidently whether we can generalize the performance conclusions.

Our primary objective in this paper is to study the impact of Xen on the behavior of HPC applications in detail. In particular, we compare the impact of Xen on HPCC [1] and NPB [2] benchmarking applications. In the process we would study how Xen affects various parts of HPCC and NPB.

The organization of our paper is the following: Section 2 presents related work in the area. Section 3 describes the settings used to study the analysis of the impact of Xen on HPC application behavior profiles. In Section 4 we detail our results based on two HPC application profiles. In Section 5, we discuss and analyze our results. In section 6, we present our conclusion and future work.

## 2 Related Work

XenOprof [6] is one of the few tools that can be used as a system wide profiler on Xen. XenOprof was used to diagnose performance overheads in network applications. The authors also study various events such as L2cache misses, ITLB misses, and correlate them in their study. However, the original goal is to identify performance bugs using the data collected by XenOprofile.

The TLB behavior for scientific applications on commodity microprocessors was studied in [5]. Their work is similar in theme to ours. Their conclusion is that while SPEC CPU and HPCC benchmark suits represent cache behaviors of the high-end scientific applications, they fall short when it comes to TLB behavior, and thus can have significant performance consequences. In this paper,

we want to emphasize the difficulty of generalizing performance conclusions in virtual environments.

Work in [8] studies memory hierarchy characteristics of paravirtualized systems. The authors also study hardware counters using XenOprof for memory intensive applications such as DGEMM. The authors conclude that Xen provides near native execution performance and similar memory hierarchy profiles. Our work attempts to compare impacts of Xen on two HPC applications in order to study their profiles.

Work reported in [3] points out that there is a need to consider real HPC applications for performance evaluations and benchmarking. The authors also compare performance results from kernel benchmarks to the real-world applications, and find that kernel benchmarks do not fully represent real world scientific applications.

Other studies such as [4] [9] concluded that Xen impacts HPC applications minimally. Our study extends previous work by attempting to determine if we can generalize such conclusions beyond those applications that are expressly studied.

### **3 Evaluation Methodology**

In this section, we outline the experimental settings used to gather results and perform post-analysis.

#### **3.1 Applications**

We have used HPCC and NPB application benchmark suite for our study. HPL and SP are used as work-loads to study the compute-bound properties of an application. The problem sizes are 6000 and 162 (class C) for HPL and SP respectively.

#### **3.2 Native and Virtual Machine Environments**

Our system environment consists of a 16 node cluster. Each node has a 2Gz Pentium 4 processor, 768MB of RAM, and a 1024KB L2 cache, connected by a gigabit ethernet switch. Our Native environment consists of a Linux 2.6.16.33 kernel with FC5 filesystem distribution. Our Virtual Machine environment runs on Xen 3.0.4, Linux kernel 2.6.16.33, with 512MB for each virtual machine with one virtual machine per node. We use the same filesystem as that of Native for HostOS. The filesystem for a virtual machine is a disk based flat file of 2GB using FC5. We use NFS shared filesystem on all three platforms.

### 3.3 Profiling and Data Collection tools

We use Oprofile 0.9.1 as our data gathering tool. Oprofile is a system wide statistical profiler. Xen 3.0.4 has an inbuilt support for Oprofile. Oprofile uses CPU counters to generate events based on a configurable frequency, which we have set to 100000. This frequency instructs Oprofile to generate a sample for every 100000 occurrences of a specific configurable event such as DTLB miss and will attribute it to the code that caused the counter associated with that event to overflow.

We study four events: Clock-unhalted, ITLB miss, DTLB miss, and L2Cache miss. For each event we gather the breakdown of the samples of an application into various parts such as application code, library code, kernel modules, kernel code, and hypervisor code. Clock-unhalted event is a measure of CPU processing time. ITLB and DTLB miss events measure the time spent by the page walk handler. L2cache miss event is a read level cache misses.

Inhouse scripts [7] are developed to parse the collected data into various parts such as application code, library code, kernel modules, kernel code, and hypervisor code. Our analysis considers the data from the application only, and do not consider data from system services.

## 4 Performance Evaluation

### 4.1 Overall penalty

Table 2 shows us the overall performance penalty on HostOS as well as on VMs in terms of the number of samples. Table 1 shows us the overhead in terms of wall clock time. The two tables show that the overhead in number of samples in virtual environments is more compared to the wall clock time overhead. One explanation is the following: Even though clock-unhalted event, as described earlier, is a measure of CPU processing time, it is a measure of time when the CPU is active. Therefore, when CPU is idle as when there is I/O or memory transfer, this event is unuseful. Further, CPU will execute less number of instructions on native compared to virtual environment as shown by Table 3, and can remain idle longer than say Hostos, which can execute more instructions in parallel to I/O. Please note that, the VMs only contain DomU side of samples. This is because of a known limitation of Oprofile, which does not allow us to isolate samples from Dom0 profile which are part of the applications running in DomU. Therefore, in the following analysis, we indicate Dom0 samples by greek letters such as  $\delta$  and  $\gamma$ . And unless otherwise stated, when we refer to VM, we mean DomU portion of the Virtual Machine.

	Hostos penalty in wall clock time %	VM penalty in wall clock time %
HPCC- HPL	2	12
NPB - SP	1	18

**Table 1.** Performance penalty as compared to native

	Hostos penalty%	VM penalty%
HPCC- HPL	8	$11 + \delta$
NPB - SP	5	$9 + \gamma$

**Table 2.** Performance penalty - Number of Samples as compared to native

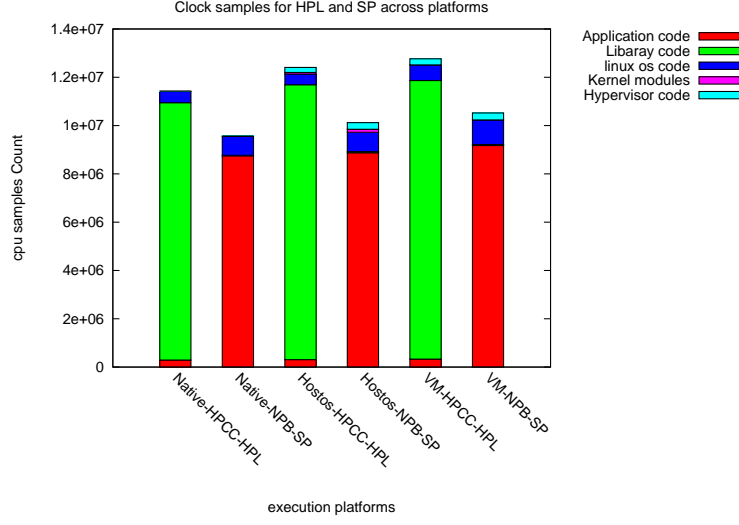
	Hostos penalty%	VM penalty%
HPCC- HPL	2	5
NPB - SP	9	11

**Table 3.** Instructions Executed - Number of Samples as compared to native

## 4.2 Breakdown of overall penalty

Figure 1 shows the breakdown of the time spent by each application into its various parts across native, Hostos and VM environments. Overhead cost of user code in HPL is more than that of SP. One reason is that user code contribution is more in HPL than in SP, and therefore virtualization impacts it disproportionately. Interestingly, the overhead cost of system code under hostos and VM in SP is less than that of HPL even though SP spends twice as much time in system code as HPL on native. This can be because HPL spends proportionately more time in hypervisor code than SP does. Furthermore, the overhead costs are more when applications are running in virtual machines than when they are running in hostos.

Table 4 shows how various parts of HPL and SP are being impacted differently in virtual environments. The most obvious is the small contribution library code. Since the library code only forms a small fraction of the overall code distribution, its impact in virtual environments does not show up in Figure 1. Similarly, the system code is expensive in virtual environments even though it may not be apparent in Figure 1 as the system code is only 10% of the overall code. The system code penalty distribution among kernel modules, kernel core and hypervisor for HPL under hostos is 9%, 62%, 29% respectively. The numbers are 10%, 66%, 24% for SP respectively. Under VMs the system code penalty distribution for HPL is 72% and 28% for kernel core and hypervisor. Similarly for SP, the distribution under VMs is 77% and 23% respectively. The contribu-



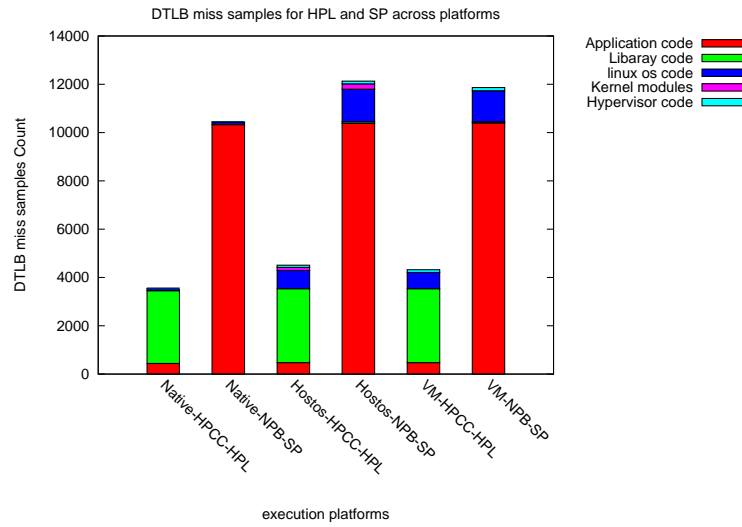
**Fig. 1.** Comparison of breakdown of CPU samples for HPL and SP across platforms - Results for VM do not contain DOM0 samples

tion of kernel modules under VMs is part of Dom0 and therefore not shown as explained previously.

	HPL-App	SP-APP	HPL-Lib	SP-Lib	HPL-Sys	SP-Sys
Hostos penalty%	5	1	6	138	50	49
Vm penalty %	13	4	8	118	$88 + \delta_{clk}$	$62 + \gamma_{clk}$

**Table 4.** Breakdown of performance penalty for Clock samples as compared to native -  $\delta_{clk}$  and  $\gamma_{clk}$ : Dom0 part of HPL and SP respectively

Further, system code penalty for HPL on VMs is more than that of SP. One explanation is that HPL code performs more privileged operations than SP. The reason why the impact of Xen on the library code in SP is so drastic compared to HPL is unclear and may additionally require sophisticated tracing to diagnose the problem. Thus, while the overall performance penalty is only one number, Table 4 shows us the actual behind-the-scene story. In light of this information, it is difficult to generalize the performance conclusions to other applications. In the next few sections, we study other events such as ITLB miss, DTLB miss and L2 cache miss.



**Fig. 2.** Comparison of breakdown of DTLB Miss samples for HPL and SP across platforms - Results for VM do not contain DOM0 samples

### 4.3 Breakdown of DTLB miss samples

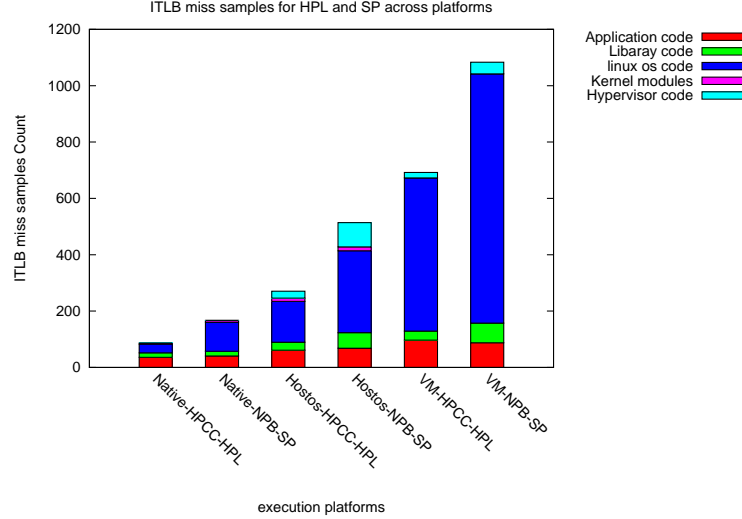
Figure 2 shows the comparison of DTLB misses across platforms for our two applications. From the figure, we can see that the impact of virtualization is limited to the system side. Further, by looking at Figure 1, one might conclude that Xen impacts HPL’s library code more than that of SP’s library code. But as described in our previous section, the contribution of the library code in SP is very small and therefore does not show up in Figure 1. However, Table 4 shows that the library code in SP is impacted drastically, and is supported by the fact that the DTLB miss rate increases for SP’s library code, and remains very low for HPL as shown in Table 5. The huge performance penalty numbers like 1900% arise because the number of DTLB miss samples increases from 3 to 60. The story for the system side described by Figure 1 is also supported by Figure 2, in that DTLB rate increases for both HPL and SP, although in different ways. The impact of Xen on DTLB miss rate is more for SP’s system code than HPL’s under hostos and VM. As stated before we cannot comment on the  $\delta_{dtlb}$  and  $\gamma_{dtlb}$  from Dom0.

### 4.4 Breakdown of ITLB miss samples

Figure 3 shows the comparison of ITLB misses across platforms for our two applications. Figure 3 and Table 6 show that Xen impacts system side more

	HPL-App	SP-APP	HPL-Lib	SP-Lib	HPL-Sys	SP-Sys
Hostos penalty %	7	0.6	1	1900	800	1300
Vm penalty %	7	0.6	1.6	1500	$700 + \delta_{dtlb}$	$1150 + \gamma_{dtlb}$

**Table 5.** Breakdown of performance penalty for DTLB miss samples as compared to native -  $\delta_{dtlb}$  and  $\gamma_{dtlb}$ : Dom0 part of HPL and SP respectively



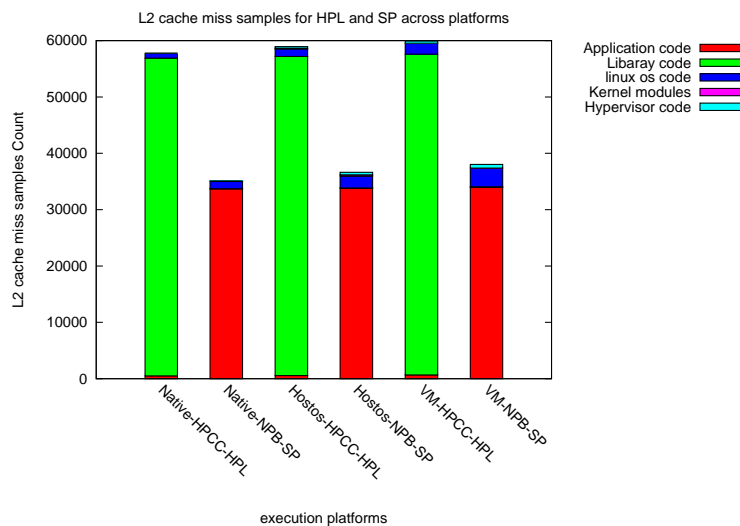
**Fig. 3.** Comparison of breakdown of ITLB miss samples for HPL and SP across platforms - Results for VM do not contain DOM0 samples

than the user side but the impact is not only restricted to the system side. First, ITLB miss rate continues to support the fact that Xen does impact SP's library code drastically. Second, ITLB miss rate (please refer Table 6) is also consistent with Figure 1 in that it partly explains why Xen impacts HPL's system code more than SP's under both hostos and VM. Yet, as shown in Table 5, DTLB miss rate does not explain why Xen impacts HPL's system side more than SP's. Moreover, one can easily see that Table 5 and Table 6 support Table 4 when it comes to application-only code.

	HPL-App	SP-APP	HPL-Lib	SP-Lib	HPL-Sys	SP-Sys
Hostos penalty %	74	70	74	243	417	257
Vm penalty %	177	117	93	331	$1500 + \delta_{itlb}$	$750 + \gamma_{itlb}$

**Table 6.** Breakdown of performance penalty for ITLB miss samples as compared to native -  $\delta_{itlb}$  and  $\gamma_{itlb}$ : Dom0 part of HPL and SP respectively

#### 4.5 Breakdown of L2 cache miss samples



**Fig. 4.** Comparison of breakdown of L2 miss samples for HPL and SP across platforms - Results for VM do not contain DOM0 samples

The comparison of L2 cache miss samples is shown in Figure 4. Table 7 shows that the impact of Xen on L2 cache miss samples is restricted to system side only, except SP's library code. Table 7 shows mixed results. It shows that while Xen impacts L2 miss rate more for HPL on Hostos when code is executing in system side, it also shows that Xen impacts SP more under VM DomU.

	HPL-App	SP-APP	HPL-Lib	SP-Lib	HPL-Sys	SP-Sys
Hostos penalty%	10	0.2	0.4	130	104	101
Vm penalty %	30	0.7	0.9	500	$171 + \delta_{l2}$	$186 + \gamma_{l2}$

**Table 7.** Breakdown of performance penalty for L2 cache samples as compared to native -  $\delta_{l2}$  and  $\gamma_{l2}$ : Dom0 part of HPL and SP respectively

## 5 Discussion

Our previous section establishes that HPL and SP are impacted differently by Xen. As shown in our study, the applications have different characteristics even

though both are compute bound. This supports our premise that we can not generalize performance in virtual environments. A detail analysis is useful to understand the application workload. For instance, HPL spends most of its user code time in BLAS library, while most of the user code in SP is located in the application itself. Second, SP has a bigger system code contribution than HPL.

The conclusion that the impact of Xen is mainly restricted toward the system code and not the user code is accepted based on Xen's paravirtualized architecture. However, this paper has indicated that while Xen impacts system code much more than user code, there is evidence, such as in the case of SP's library code, that the user code may not be immune from Xen's impact.

## 6 Conclusion

We have studied and analyzed HPL and SP from HPCC and NPB respectively. Our goal of the study was to determine the impact of Xen on these applications and compare the penalty profiles of these two applications. It is important to note that we are not only concerned with the "final performance penalty" number but the composition that makes up the overall performance penalty.

We found that, while the overall performance penalty does not differ much between HPL and SP, their overhead profiles are not similar. Further, we found that Xen impacts the various parts of these applications in different ways. It is therefore possible that different applications in the same class may be impacted more differently than HPL or SP.

We also found that the similar final performance impact of HPL and SP is not entirely due to the fact that these are compute-bound benchmark applications, but because the parts that are impacted differently by Xen are too small to influence the final performance number.

Our findings emphasize the difficulty of performance prediction and generalization. Moreover, as we have seen, performance isolation, especially on VMs (the Dom0 part) remains difficult to achieve.

We plan to extend our study to more scientific applications. We would like to determine whether similar benchmark applications have versatility such that Xen impacts them differently or not. Further, we would like to work on the limitations of the performance measurement tools, such as XenOprof, so that we can enhance the performance isolation in Dom0.

## References

1. Hpc challenge. In <http://icl.cs.utk.edu/hpcc>.
2. Nas parallel benchmarks. In <http://www.nas.nasa.gov/Resources/Software/npb.html>.

3. Brian Armstrong, Hansang Baeh, Rudolf Eigenmann, Faisal Saied, Mohamed Sayeed, and Yili Zheng. Hpc benchmarking and performance evaluation with realistic applications. In *2006 SPEC Benchmark Workshop (spec)*, 2006.
4. W. Emeneker and D Stanzione. HPC Cluster Readiness of Xen and User Mode Linux. In *IEEE International Conference on Cluster Computing*, September 2006.
5. Collin McCurdy, Alan Cox, and Jeffrey Vetter. Investigating the tlb behavior of high-end scientific applications on commodity microprocessors (ispass 08). In *IEEE International Symposium on Performance Analysis of Systems and Software*, 2008.
6. 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.
7. Anand Tikotekar, Geffroy Vallee, Thomas Naughton, Hong Ong, Christian Engelmann, and Stephen L Scott. Effects of virtualization on a scientific application. In *2nd Workshop on System-level Virtualization for High Performance Computing (HPCVirt 2008) held in conjunction with EuroSys*, 2008.
8. Lamia Youseff, Keith Seymour, Haihang You, Jack Dongarra, and Rich Wolski. The impact of paravirtualized memory hierarchy on linear algebra computational kernels and software. In *ACM/IEEE International Symposium on High Performance Distributed Computing (HPDC)*, 2008.
9. 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.