Analyzing PAPI Performance on Virtual Machines

John Nelson

I. OVERVIEW

Over the last ten years, virtualization techniques have become much more widely popular as a result of fast and cheap processors. Virtualization provides many benefits making it appealing for testing environments. Encapsulating configurations is a huge motivator for wanting to do performance testing on virtual machines. Provisioning, a technique that is used by FutureGrid, is also simplified using virtual machines. Virtual machines enable portability among heterogeneous systems while providing an identical configuration within the guest operating system.

My work in ICL has focused on using PAPI inside of virtual machines. There were two main areas of focus throughout my research. The first originated because of anomalous results of the HPC Challenge Benchmark reported in a paper submitted by ICL [3] in which the order of input sizes tested impacted run time on virtual machines but not on bare metal. A discussion of this anomaly will be given in section II along with a discussion of timers used in virtual machines. The second area of focus was exploring the recently implemented support by KVM (Kernel-based Virtual Machine) and VMware for guest OS level performance counters. A discussion of application tests run to observe the behavior of event counts measured in a virtual machine as well as a discussion of information learned pertinent to event measurement will be given in section III.

II. HPC CHALLENGE BENCHMARK ANOMALY

In Figure 1 an interesting result can be seen the anomaly reported in a paper by ICL [3]. This anomaly was discovered when comparing HPC Challenge benchmark results on VMware Player, VirtualBox and KVM to the same benchmark results on bare metal. The curious result was that there was a significant difference between results on the virtual platforms tested when comparing tests runs with input size in ascending order with that of test runs with input size in descending order. This behavior was not present in results of test runs on bare metal. Figure 1 shows the percentage difference (absolute value) between run time of ascending input sizes and descending input sizes.

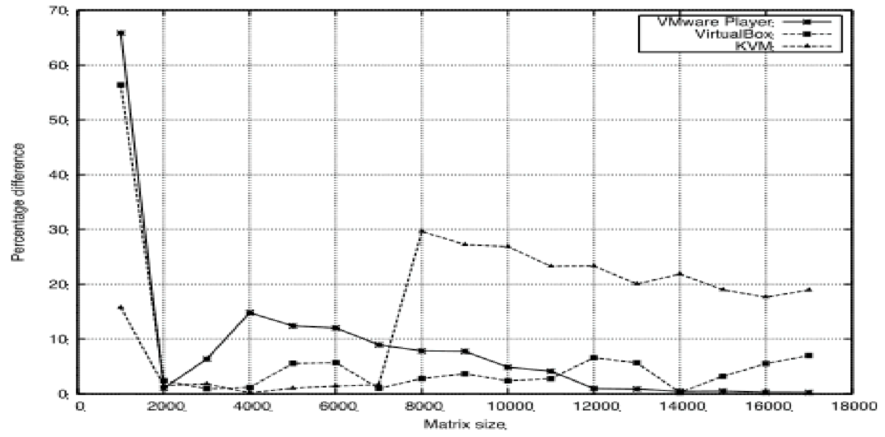


Fig. 1 - HPC Challenge Benchmark Ascending vs Descending Input Sizes

I was tasked with replicating the results reported in [3], specifically in regards to VMware. Figure 2 shows a similar result, albeit less extreme, on VMware ESXi 4.0. In this case, negative values are shown as it is not obvious from the results reported in [3] that neither ascending or descending always run with a shorter run time. Many subsequent tests were performed, all of which only served to produce more operating theories by the PAPI group on the source of this anomaly. The anomaly disappeared in the release of VMware ESXi 5.0. The search for a definitive reason for the source of the anomaly was abandoned at that point.

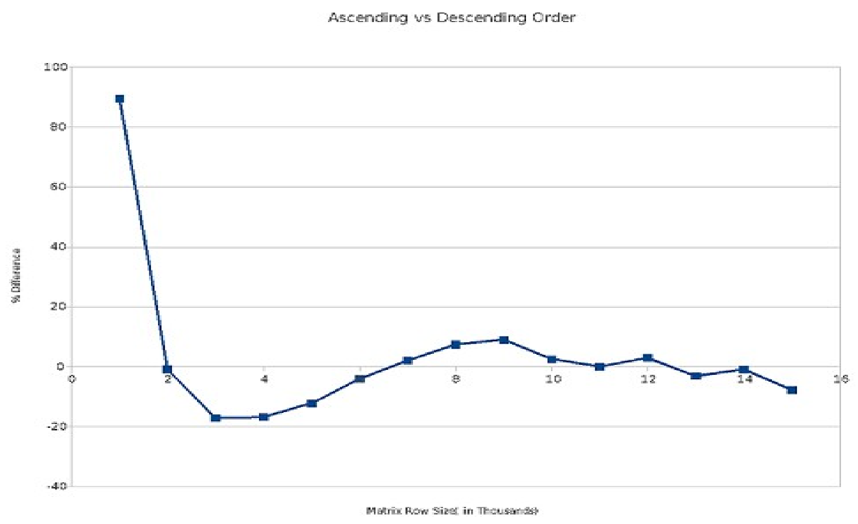


Fig. 2 - HPC Challenge Benchmark Ascending vs Descending Input Sizes: VMware ESXi 4

Assuming another anomaly of a similar nature were to be discovered now, it would be a much more straightforward process to determine the cause. The introduction of support for a virtual performance monitoring Unit (PMU) by VMware and KVM would allow for examining counts for each PAPI event. It is likely that one would determine that one of these such events, such as data cache misses, is much greater for one set of testing conditions than another. Unfortunately, at the time, an examination of performance measurement events was not possible due to the limitations of the virtualization platforms.

B. Timers in Virtual Machines

A theory that was explored to explain the anomaly was that the timers used in the virtual machine tests may not be accurate. There are two techniques to track time used by most operating systems: tick counting and tickless timekeeping. Tick counting is performed by defining a hardware interrupt that will occur at a consistent interval. The OS must handle this interrupt before continuing execution. Tickless timekeeping is performed by reading a hardware provided time counter which will continue to increment without need for handling an interrupt. This requires the hardware to specifically provide such a counter. Both Intel and AMD processors have supported tickless timekeeping for many years.

Tick counting can skew time inside a virtual machine. Interrupts are queued when a VM is descheduled and executed in rapid succession when the VM is rescheduled. This will cause the speed of the passing of time to appear to fluctuate. However, it was determined that the HPC Challenge tests were done on a system supporting tickless timekeeping. The timer used in VMware was also tested by measuring the run time of the HPC Challenge benchmark with an external NTP server. It was found that the external time source consistently differed by less than one percent than the internal timer. Therefore, the timer in VMware was proven accurate and the theory of inaccurate timers was disproved.

III. ANALYSIS OF EVENT COUNT MEASUREMENTS ON VIRTUAL MACHINES

A. Overview

One of the consequences of virtualization is that there is an additional hardware abstraction layer. This prevents PAPI from reading the hardware PMU (Performance Monitoring Unit) directly. However, both VMware and KVM have recently, within the last year, added support for a virtual PMU. Within the guest operating system on a virtual machine that provides a virtual PMU, PAPI can be built without any special build procedure and will access the virtual PMU with the same system calls as a hardware PMU.

B. Testing Environment

Bare Metal

- 16-core Intel Xeon CPU @ 2.9GHz
- 64 GB memory
- Ubuntu Server 12.04
- Linux kernel 3.6

KVM

- Qemu version 1.2.0
- Guest VM - Ubuntu Server 12.04
- Guest VM - Linux kernel 3.6
- Guest VM - 16GB ram

VMware

- VMware ESXi 5.1
- Guest VM - Ubuntu Server 12.04
- Guest VM - Linux kernel 3.6
- Guest VM - 16GB ram

C.Test Suite

Tests performed are taken from the Mantevo Suite of mini-applications [4]. The purpose of the Mantevo suite is to provide miniapplications which mimic performance characteristics of real world large scale applications. The applications listed below were used for testing:

- CloverLeaf
- CoMD
- HPCCG
- MiniGHOST
- MiniXYCE

D. Testing Procedure

For each platform (Bare metal, KVM, VMware), each application was run while measuring 1 PAPI preset event. Source code for each application was modified to place measurements surrounding the main computational work of each application. Ten runs were performed for each event. Tests were run on a "quiet" system; no other users were logged on and only minimal OS services were running at the same time. Tests were performed in succession with no reboots between runs.

E. Reading Results

Results are presented as box plots in Figure 3-8. Along the x-axis is a PAPI preset event. Along the y-axis is the ratio of event counts to bare metal event counts. That is, the ratio of the mean of 10 runs on the virtual machine to the mean of 10 runs on a bare metal system. Therefore, a ratio of 1 corresponds to an identical number of event counts. A ratio of 2 corresponds to twice as many counts on the virtual machine as on bare metal. We would expect in nearly all cases for the ratio to be greater than 1 due to the overhead of virtualization with a few exceptions. As can be seen on the graphs, most boxes appear as a straight horizontal line. This happens because the standard deviation is so low on these events compared to others that the whiskers on the plot appear to overlap the box due to the y-axis scale. Results from CoMD and MiniGHOST have been omitted for space. Results from CoMD and MiniGHOST match the characteristics of Cloverleaf and HPCCG.

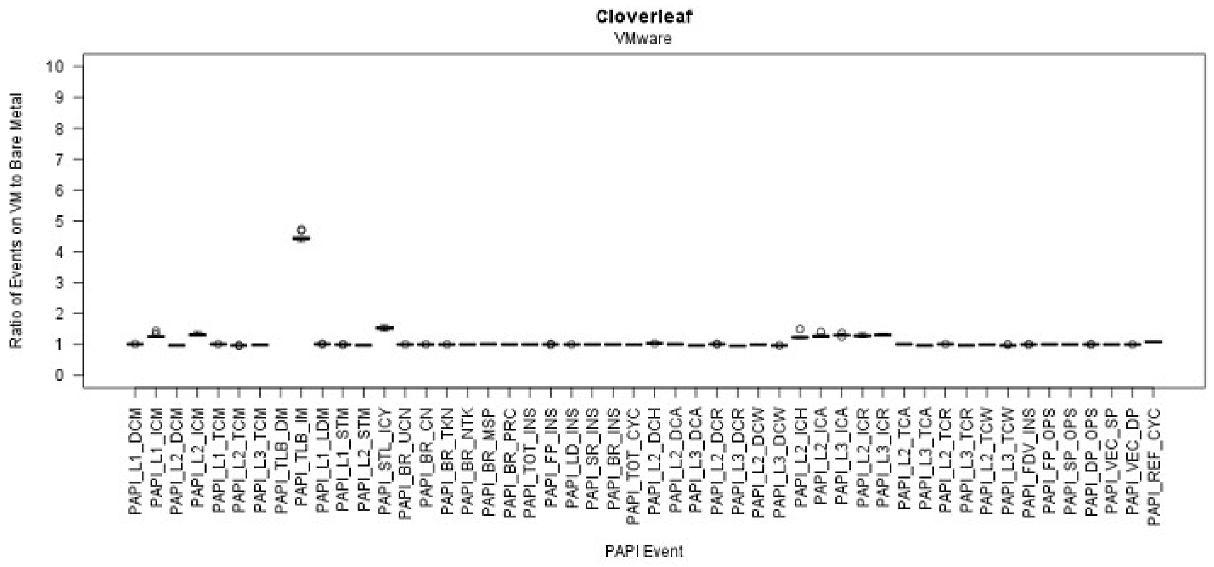


Fig. 3 - Cloverleaf on VMware

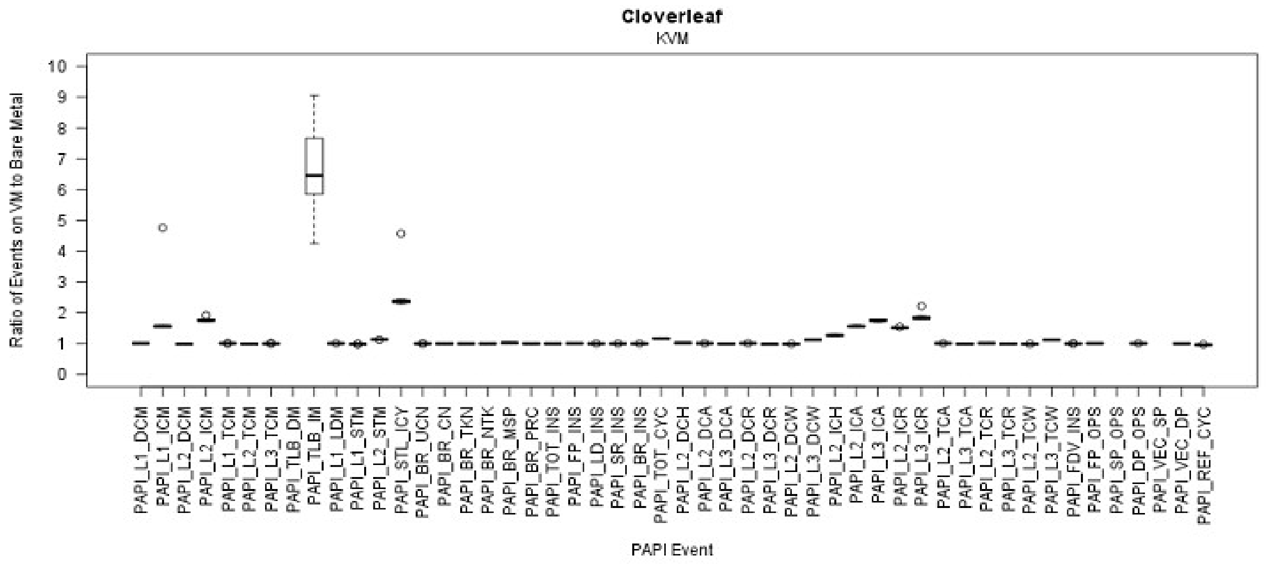


Fig. 4 - Cloverleaf on KVM

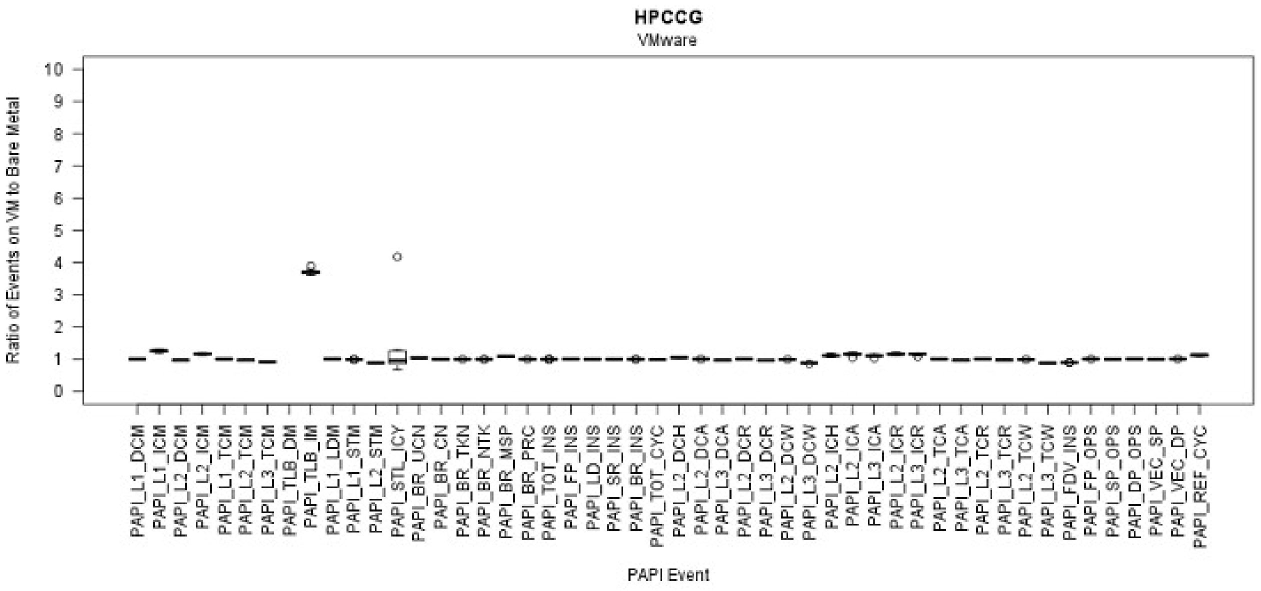


Fig. 5 - HPCCG on VMware

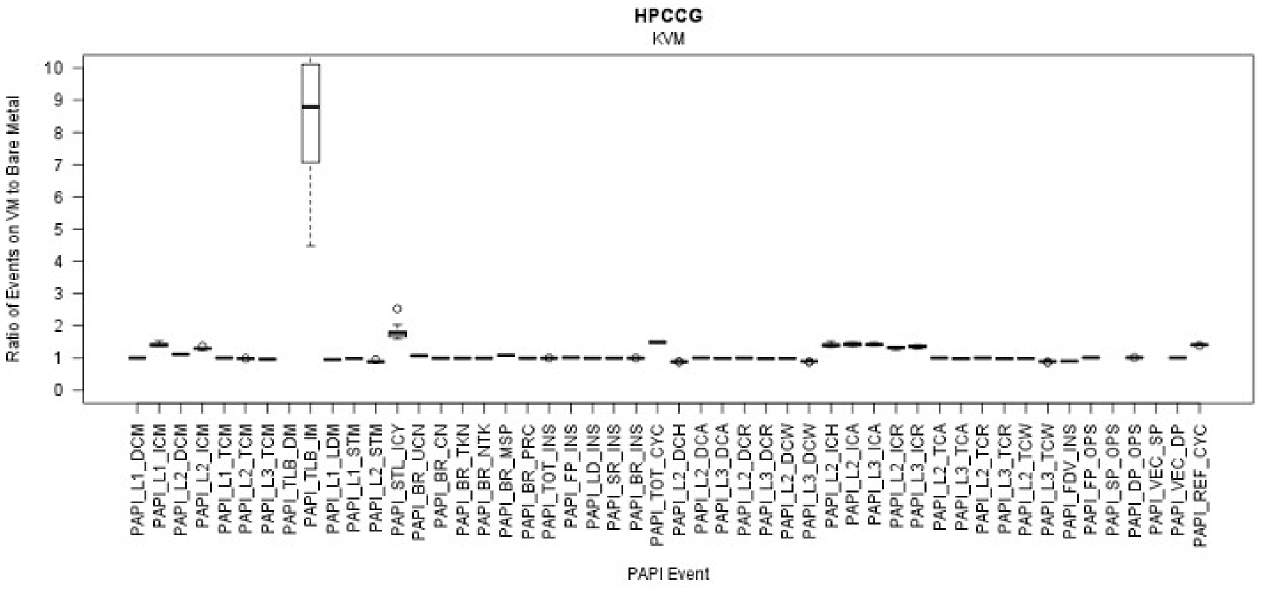


Fig. 6 - HPCCG on KVM

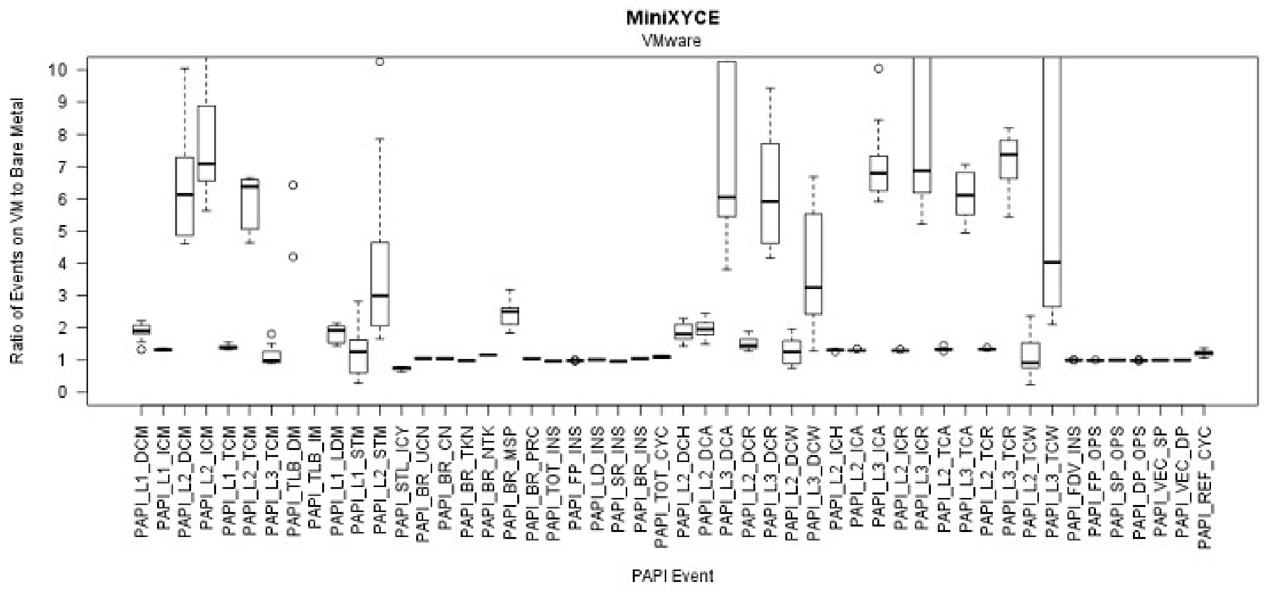


Fig. 7 - MiniXyce on VMware

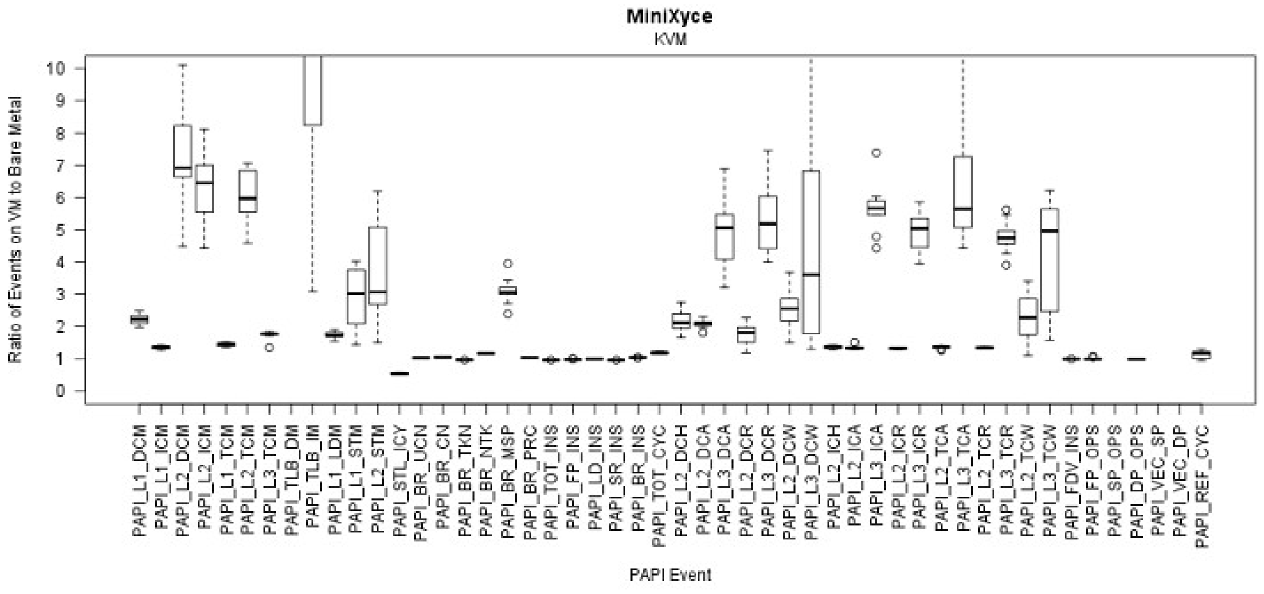


Fig. 8 - MiniXyce on KVM

F. Observations

On inspection of the results, there are two main classes of events which exhibit ratios significantly different than 1. These two classes include instruction cache events, such as PAPI_L1_ICM, and translation lookaside buffer events such as PAPI_TLB_IM. These two classes will be examined more closely below. Another anomaly which has yet to be explained is KVM reporting non-zero counts of

PAPI_VEC_DP and PAPI_VEC_SP (both related to vector operations), whereas bare metal always reports 0.

1. Instruction Cache

From the results, we can see that instruction cache event counts are much more frequent on the virtual machines than on bare metal. These events include: PAPI_L1_ICM, PAPI_L2_ICM, PAPI_L2_ICA, PAPI_L3_ICA, PAPI_L2_ICR, and PAPI_L3_ICR. By a simple deduction, we can see that L2 events are directly related to PAPI_L1_ICM, and likewise L3 events to PAPI_L2_ICM. That is, the level 2 cache will only be accessed in the event of a level 1 cache miss. Miss rate will compound total accesses at each level, and as a result, the L3 instruction cache events appear the most different than on bare metal with a ratio of over 2. Therefore, it is most pertinent to examine the PAPI_L1_ICM results as all other instruction cache events are directly related. Inn Figure 9, we can see the results of the HPCCG tests on bare metal, KVM and VMware side by side. As can be seen on the graph, both KVM and VMware underperform bare metal results. However, VMware is quite a bit better off with only 20% more misses than bare metal whereas KVM exhibits nearly 40% more misses. Both have a large standard deviation than bare metal, but not by a huge margin.

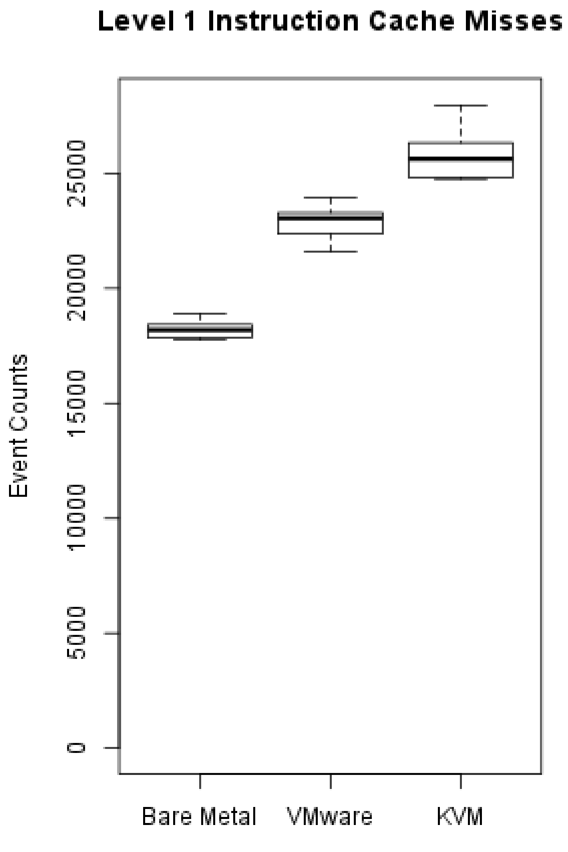
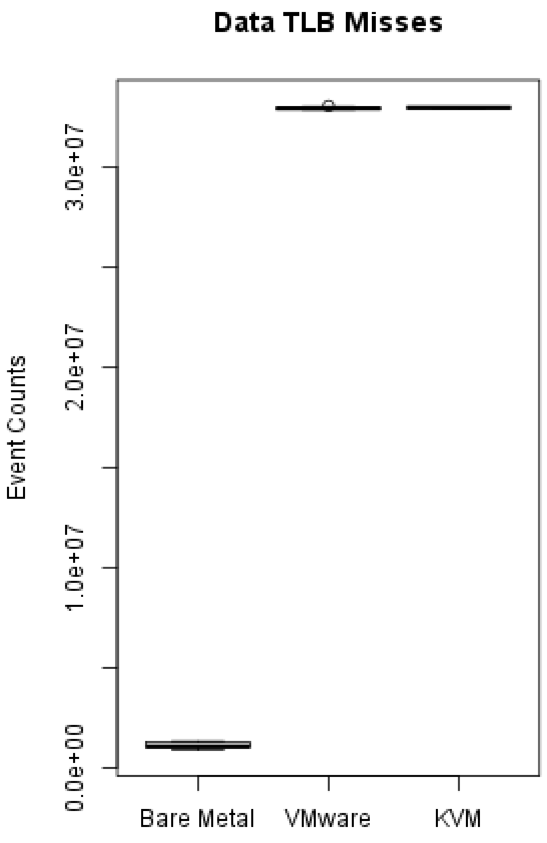


Fig. 9 - L1 Instruction Cache Misses on HPCCG

2. TLB

In Figure 10 (using data from the HPCCG application), we can see that Data TLB misses are a huge issue for both KVM and VMware. Both exhibit around 33 times more misses than runs on bare metal. There is little difference between the two virtualization platforms for this event.

Instruction TLB misses, seen in Figure 11, are also significantly more frequent on both virtualization platforms than on bare metal. However, VMware seems to fair much better in this regard. Not only does VMware incur 50% of the misses seen on KVM, VMware also has a much smaller standard deviation (even smaller than that of bare metal) compared to KVM's extremely unpredictable results.



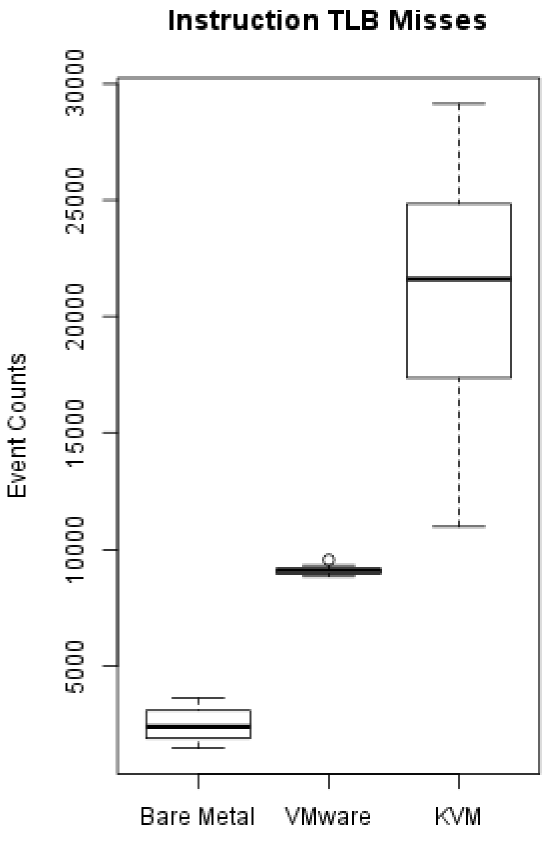


Fig. 10 - Data TLB Misses on HPCCG

Fig. 11 - Instruction TLB Misses on HPCCG

3. *MiniXyce*

The results for MiniXyce warrant special consideration. The behavior of the application itself appears much less deterministic than other tests in the test suite. MiniXyce is a simple linear circuit simulator which attempts to emulate the vital computation and communication of XYCE, a circuit simulator designed to solve extremely large circuit problems on high performance computers for weapon design [5]. Even so, the behavior of the application may have exposed shortcomings in the virtualization platforms. Not only are instruction cache miss and tlb miss events much more frequent on VMware and KVM, data cache misses and branch misspredictions are much more frequent. MiniXyce is the only application tested that displayed significantly different data cache behavior on the virtual machines than on bare metal. This may suggest that only a certain class of applications with similar behavior characteristics may cause VMware and KVM to perform worse than bare metal in regards to the data cache. This would be an interesting area of future research into the performance of virtual machines.

G. Possible Confounding Variable

VMware and KVM do not measure guest-level performance events in the same way. One of the challenges of counting events in a guest vm is determining which events to attribute to the guest. This is particularly problematic when the hypervisor emulates privileged instructions. Domain switch and CPU switch define which events to attribute to the guest. Domain switch is the more inclusive of the two, including all events that the hypervisor contributes when emulating guest I/O.

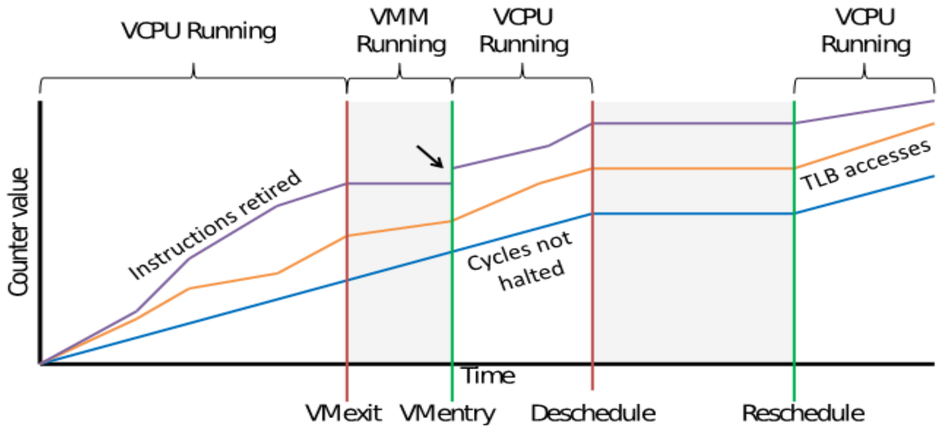


Figure 12 above shows an example time line for VM scheduling events [3]. CPU switch does not include events contributed by the hypervisor. On VMexit, when the hypervisor must process an interrupt (such as a trapped instruction), the performance counter state is saved and then restored on VMentry. Descheduling also causes the save and subsequent restore in CPU switch. Alternately, the performance counter states in domain switch are only saved when the guest is descheduled and restored on rescheduling. Events are counted between VMexit and VMentry in domain switching. Domain switching, therefore, gives a more accurate view of the affects the hypervisor has on execution, whereas CPU switching hides this affect. However, domain switching may also count events that occur due to another VM.

KVM chooses either domain switching or CPU switching, both of which have downsides as discussed above. VMware, however, uses a hybrid approach. Events are separated into two groups: speculative and non-speculative. Speculative events include events which are affected by run-to-run variation. For example, cache and branch miss-prediction are both non-deterministic and may be related to previous executing code which affects the branch predictor or data cache. Both are affected by the virtual machine monitor as well as all virtual machines that are executed. Therefore, any specific virtual machine cache performance, or other non-speculative event performance, will be affected by any other virtual machine that is scheduled. That is, the data cache will be filled with data by a virtual machine while it is executing. Further loads to this data will be cached. However, when execution returns to the virtual machine monitor or to another virtual machine data in the cache may be overwritten by data which is loaded by the virtual machine monitor or another running virtual machine. On the other hand, non-speculative events are events which are consistent such as total instructions retired. VMware will count speculative events between VMexit and VMentry but will not count non-speculative events. This gives a more accurate representation of the effects of the hypervisor on speculative events, while providing accurate results for non-speculative events which should not be affected by the hypervisor

[2].

The difference in how KVM and VMware choose to measure events could potentially skew the results for certain events. It is safe to assume that for the majority of the events tested (28 out of 40), the event counts can be trusted as there is less than one percent difference between either virtualization platform and bare metal. To adequately explain the apparent poor performance of instruction cache and tlb on both KVM and VMware, it would be necessary to have a more concrete understanding of which events are being counted and which are not. This may require additional testing designed to measure system-wide event counts to compare to guest-level event counts.

H. Xen

The Xen virtualization platform was not tested as a configuration that exposed performance counters was not found. However, according to Xen's minimal documentation, there should be support for performance counters in recent versions of Xen. A more fair comparison than VMware versus KVM would be to compare results measured on VMware with Xen. Whereas KVM is a hosted Virtual Platform requiring a host operating system, VMware and Xen both run a kernel directly on hardware. Xen also employs paravirtualization techniques to increase the performance of virtualized TLB [1]. As this study has shown, TLB performance is significantly worse on KVM and VMware.

IV. CONCLUSIONS

The two performance studies described in this paper have lead to a few useful conclusions. The first is that the timers used by PAPI are reliable as confirmed against external time sources. Another useful conclusion is that PAPI can be used within a KVM or VMware virtual machine to reliably measure guest-wide performance events. However, there are a few events that when measured either reveal poor performance of the virtualization platform or are possibly overestimated when attributed to virtual machines. For the large majority of events, one should expect to see almost identical results on a virtual machine as on a bare metal machine on a lightly loaded system. Future work examining event counts when a system is overloaded by many concurrently running virtual machines is necessary to make conclusions on whether results provided by PAPI are accurate in such a situation.

REFERENCES

[1] Barham, Paul, Boris Dragovic, Keir Fraser, Steven Hand, Tim Harris, Alex Ho, Rolf Neugebauer, Ian Pratt, and Andrew Warfield. "Xen and the art of virtualization." *ACM SIGOPS Operating Systems Review* 37, no. 5 (2003): 164-177.

[2] Benjamin Serebrin and Daniel Hecht. 2011. Virtualizing performance counters. In *Proceedings of the 2011 international conference on Parallel Processing* (Euro-Par'11), Michael Alexander, Pasqua D'Ambra, Adam Belloum, George Bosilca, and Mario Cannataro (Eds.). Springer-Verlag, Berlin, Heidelberg, 223-233. DOI=10.1007/978-3-642-29737-3_26 http://dx.doi.org/10.1007/978-3-642-29737-3_26

[3] Luszczek, P., Meek, E., Moore, S., Terpstra, D., Weaver, V. and Dongarra, J., "Evaluation of the HPC Challenge Benchmarks in Virtualized Environments." In 6th Workshop on Virtualization and High-Performance Cloud Computing (VHPC '11). Bordeaux, France, 2011.

[4] Mantevo Project. http://www.mantevo.org.

[5] Xyce. http://xyce.sandia.gov/overview.html.