An Analysis of HPC Benchmarks in Virtual Machine Environments

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

System Research Team
Computer Science Research Group
Computer Science and Mathematics Division

Oak Ridge National Laboratory, Oak Ridge, Tennessee, USA

European Conference on Parallel and Distributed Computing (Euro-Par)
Workshop on Virtualization in High-Performance Cluster and Grid Computing (VHPC)
Las Palmas de Gran Canaria, Spain, August 26, 2008

Outline

- **Background and motivation**
- **Prior work and presented contribution**
- **Experimental setup and results**
- **Observations and conclusion**
- **Topics for discussion and future work**

Our Systems and Applications

- **We are the largest multi-purpose science laboratory within the US Department of Energy**
  - **Over $1 billion budget, over 4000 employees**
- **Oak Ridge National Laboratory has the 2^{nd} largest open scientific computing facility within the US**
  - **Jaguar, a 260 TFlop/s Cray XT4 with 7744 2.1GHz quad-core AMD processors and 60.5 TB total RAM**
- **Upgrade to petascale computing facility 2008/9**
- **Computationally and data intensive applications**
  - **Climate, astrophysics, fusion, nanotechnology, …**

Our Usage Model and System Software

- **Usage model: Capability computing**
  - **Single jobs occupy the entire system or large parts**
  - **Allocations range from days to weeks and months**
  - **Nation-wide resource with competitive process for allocation awards (DOE INCITE)**
- **Production-type system software stack**
  - **Compute node Linux (CNL), a Linux variant with modifications by Cray (drivers and scalability)**
  - **MPI, Open MP, Global Arrays, scientific libraries, ...**
  - **No virtualization solution (still considered research)**

Why Virtualization in HPC?

- **Virtual system environments**
  - **Configurable ‘sandbox’ for system software and scientific application development and deployment**
  - **Simplified application porting through virtualization**
  - **On-demand OS deployment on virtualized systems**
  - **On-demand deployment of isolated virtual testbeds**
- **Lowest-level layer for capturing state**
  - **Transparent VM checkpoint/restart (reactive FT)**
  - **Transparent VM migration (proactive FT)**

Impact of Virtualization on Applications

- Virtual environments are inherently complex
- Certain unpredictability of performance
- Performance impact is application depended
- Measuring the overall performance penalty
  – Can hide important performance differences/details
- Similar applications, but dissimilar code profiles
  – User, library and system code
- Missing knowledge about specific events
  – ITLB, DTLB and L2 cache misses

The Big Question

- **Is there a generalization for performance of compute-intensive applications in virtual environments?**
  - **Yes (results allow for a performance model)**
  - **No (unpredictable performance, no fitting model)**
  - **Can't say (inconclusive results)**
- **Need to investigate the impact of virtualization on HPC application performance in more detail**

Prior Work

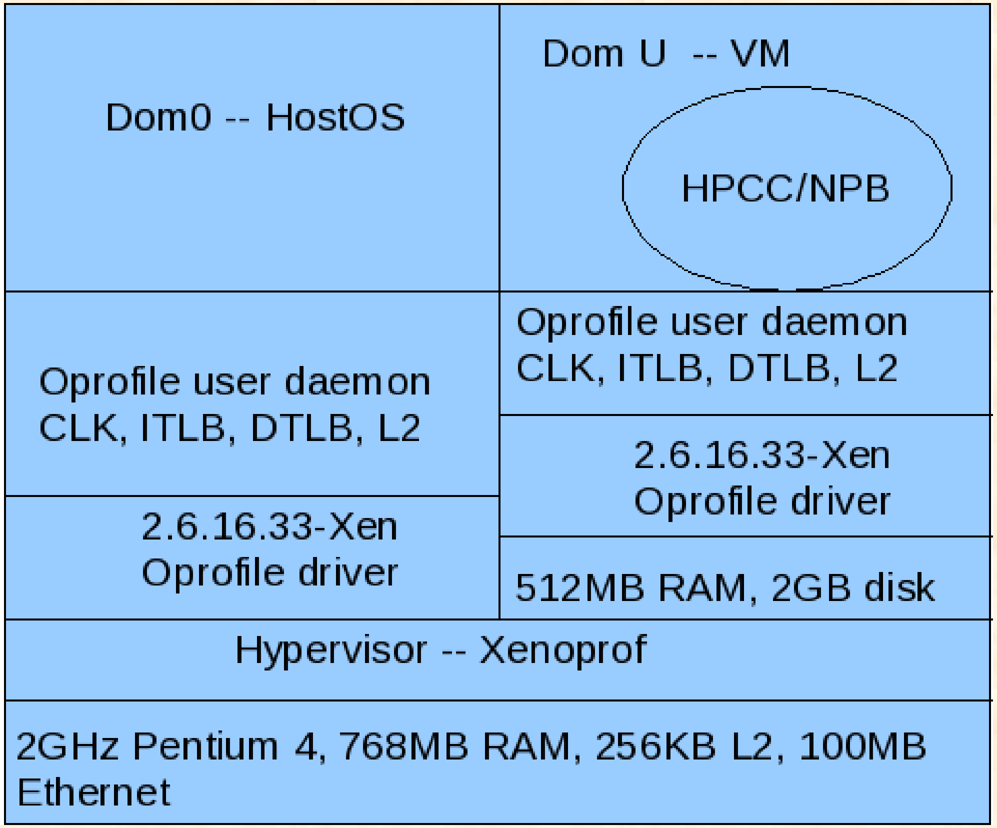
- Xenoprof tool
  - Investigation of performance overheads
  - Primarily used for diagnosing application bugs
- TLB behavior study of scientific applications
  - SPEC CPU and HPCC suites reflect cache behavior of HPC applications, but not TLB behavior
- Memory hierarchy study in virtualized systems
  - Xenoprof data shows near-native performance
- Real HPC applications != HPC kernel benchmarks
- Studies on Xen's impact on HPC applications
  - Small performance overheads, no generalization

Presented Contribution

- Investigation of performance penalties for
  - HPC Challenge (HPCC) Benchmark
    - High-Performance Linpack (HPL) Benchmark
  - NAS Parallel Benchmark (NPB)
    - Pentadiagonal Solver (SP) Benchmark (Class C)
- Study of HPL and SP on
  - Native environment
  - Host OS (Dom0) environment
  - Virtual machine (DomU) environment
- Comparative breakdown for HPL and SP of
  - Events: Clock cycles, DTLB, ITLB and L2 cache
  - Code: User, system, hypervisor, library, modules

Experimental Setup

- **16-node cluster**
- **Linux on Xen 3.0.4**
- **Oprofile 0.9.1**
  - **Frequency: 10,000**
  - **Native|Dom0|DomU**
- **Problem sizes**
  - **HPCC HPL: 6000**
  - **NAS SP: 162**
- **Parallel runs**



Overall Performance Penalty

	HostOS penalty %			VM penalty %		
	Wall clock time	No. of samples	Instructions executed	Wall clock time	No. of samples	Instructions executed
HPCC- HPL	2	8	2	12	$11 + \delta$	5
NPB - SP	1	5	9	18	$9 + \gamma$	11

Table 1. Performance penalty as compared to native

- **All numbers are relative to Native**
- **Wall clock time != # of samples**
  - **Clock-unhalted event is a measure of CPU active time**
  - **Does not account for time when CPU is idle, e.g., for I/O**
- **VM samples are DomU only (Xenoprof limitiation)**
- **δ and γ represent Dom0 (Host OS) samples of VM execution**

Distribution of Clock Cycles

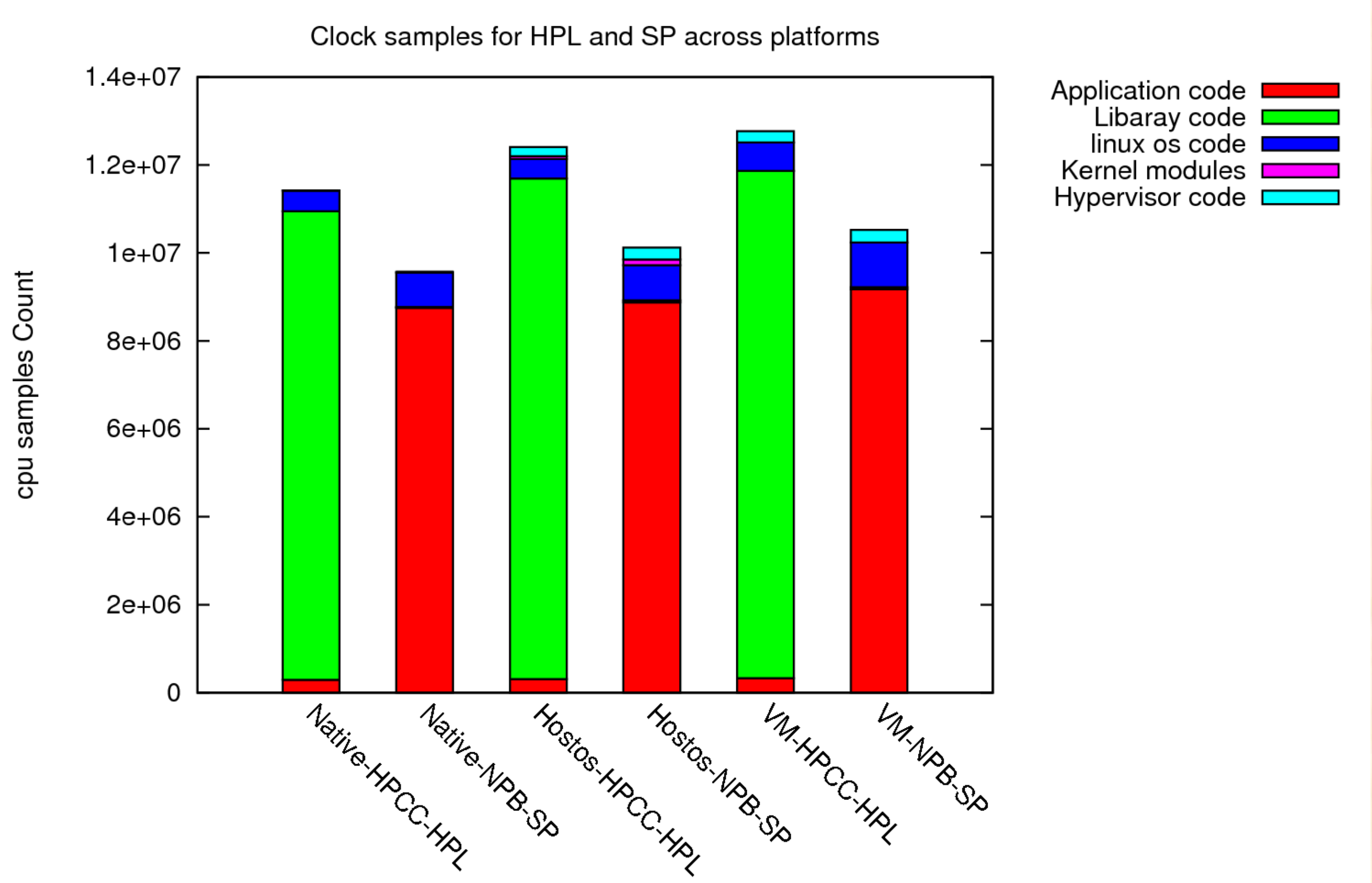


Fig. 1. Comparison of breakdown of CPU samples for HPL and SP across platforms - Results for VM do not contain Dom0 samples

Breakdown for Clock Samples

	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 + δ_{clk}	62 + γ_{clk}

Table 2. Breakdown of performance penalty for clock samples as compared to native - δ_{clk} and γ_{clk}: Dom0 part of HPL and SP respectively

- **All numbers are relative to Native**
- **δ_{clk} and γ_{clk} represent Dom0 samples of VM execution**
- **Overall SP-library and HPL-application sample count is tiny**
- **Application code penalties for HPL and SP**
  - **Host OS: penalty (HPL-app.) > penalty (SP-app.)**
  - **VM: penalty (HPL-app.) > penalty (SP-app.)**

Distribution of Clock Cycles

- Library code penalties for HPL and SP
  - Host OS: penalty (HPL-library) << penalty (SP-library)
  - VM: penalty (HPL-library) << penalty (SP-library)
- System code penalties for HPL and SP
  - Host OS: penalty (HPL-system) ≈ penalty (SP-system)
  - VM: penalty (HPL-system) > penalty (SP-system)
- System code distribution (HPL 5%, SP 10% of overall code)

	% Hypervisor		% Kernel Core		% Kernel Modules	
	HPL	SP	HPL	SP	HPL	SP
Host OS	29	24	62	66	9	10
VM	28	23	72	77	N/A	N/A

Distribution of DTLB Miss Samples

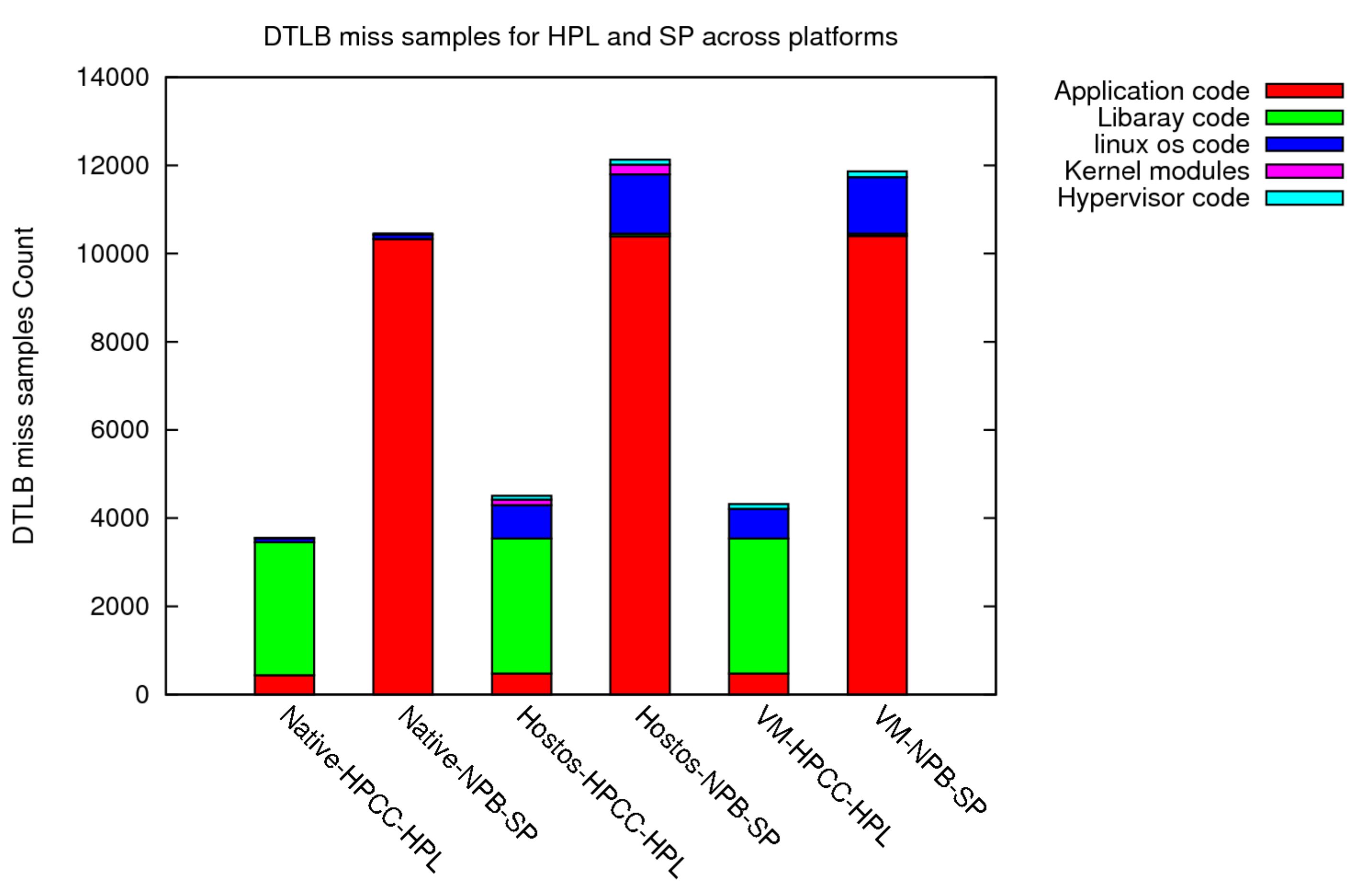


Fig. 2. Comparison of breakdown of DTLB Miss samples for HPL and SP across platforms - Results for VM do not contain Dom0 samples

Breakdown for DTLB Miss Samples

	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 3. Breakdown of performance penalty for DTLB miss samples as compared to native

- **All numbers are relative to Native**
- **δ_{dtlb} and γ_{dtlb} represent Dom0 samples of VM execution**
- **DTLB overhead (host OS) ≈ DTLB overhead (VM)**
- **Large DTLB overheads for HPL- and SP-system code**
- **Small DTLB overheads for HPL- and SP-application code**
- **DTLB overheads for HPL-library code are small**
- **DTLB overheads for SP-library code are huge**

Distribution of ITLB Miss Samples

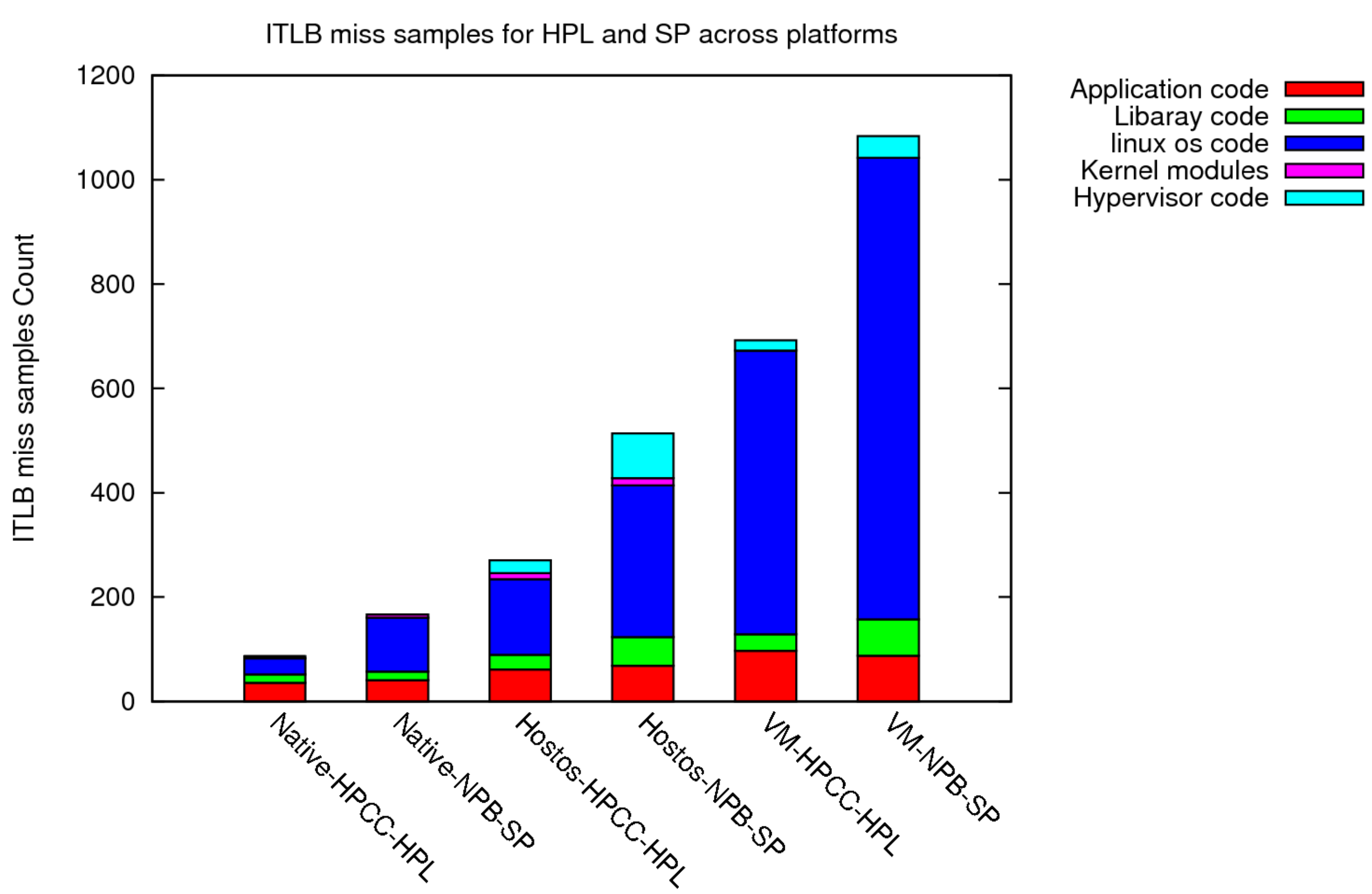


Fig. 3. Comparison of breakdown of ITLB miss samples for HPL and SP across platforms - Results for VM do not contain Dom0 samples

Breakdown for ITLB Miss Samples

	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 4. Breakdown of performance penalty for ITLB miss samples as compared to native - δ_{itlb} and γ_{itlb}: Dom0 part of HPL and SP respectively

- **All numbers are relative to Native**
- **δ_{itlb} and γ_{itlb} represent Dom0 samples of VM execution**
- **ITLB overhead (host OS) << ITLB overhead (VM)**
- **ITLB overhead (user) << ITLB overhead (system)**
- **ITLB overhead (SP-system) << ITLB overhead (HPL-system)**
- **ITLB overheads for SP-library code are big**

Distribution of L2 Cache Miss Samples

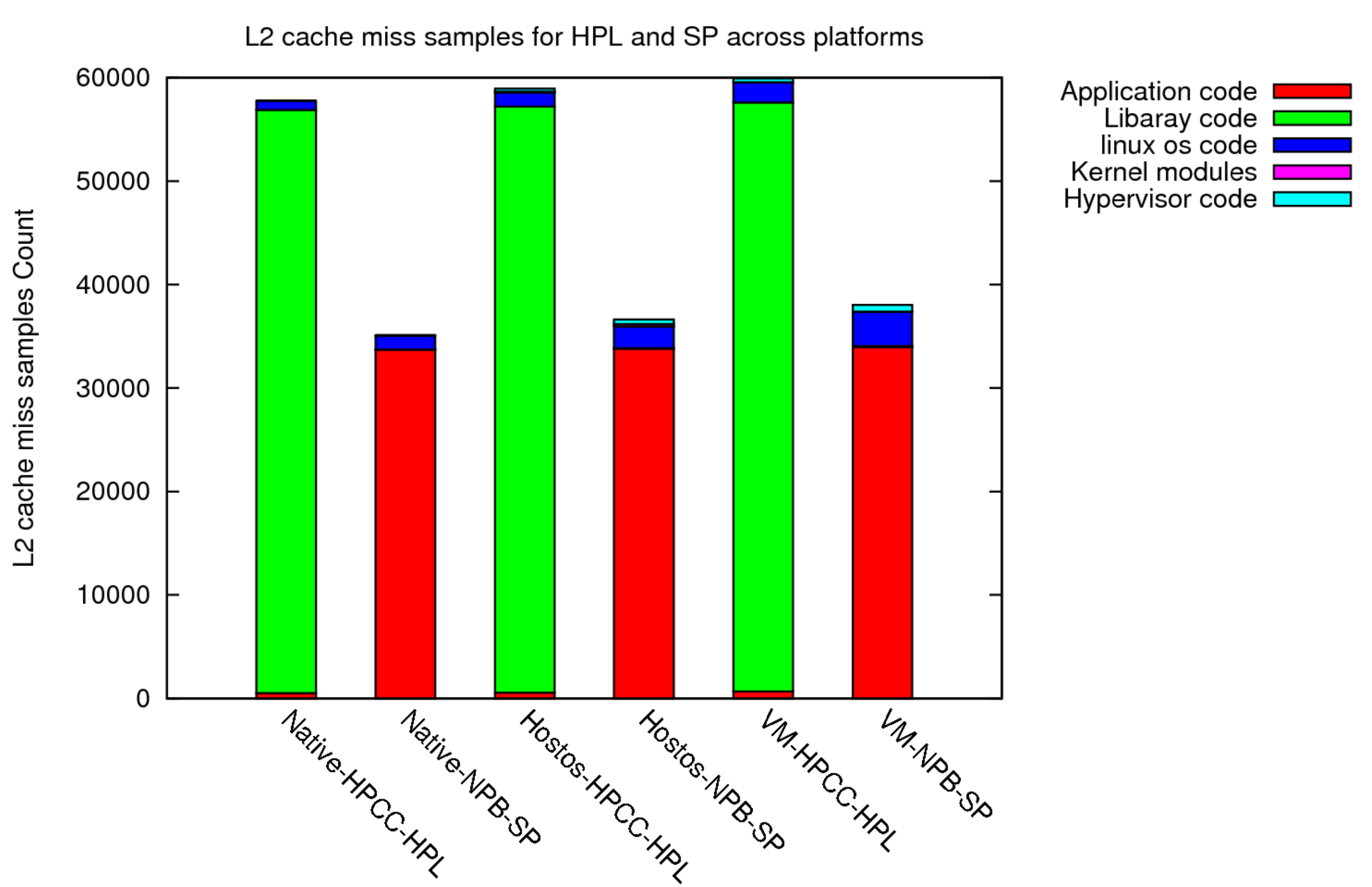


Fig. 4. Comparison of breakdown of L2 miss samples for HPL and SP across platforms - Results for VM do not contain Dom0 samples

Breakdown for L2 Cache Miss Samples

	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 + δ_{l2}	186 + γ_{l2}

Table 5. Breakdown of performance penalty for L2 cache samples as compared to native - δ_{l2} and γ_{l2}: Dom0 part of HPL and SP respectively

- **All numbers are relative to Native**
- **δ_{l2} and γ_{l2} represent Dom0 samples of VM execution**
- **L2 cache overhead (host OS) << L2 cache overhead (VM)**
- **L2 cache overheads for HPL-library code are tiny**
- **L2 cache overheads for SP-library code are large**

Observations

- **Xen impacts HPL and SP differently**
  - **Other applications may be impacted differently**
- **Large impact parts are too small in code weight**
  - **No influence on final performance penalty %**
- **System code penalty >>> User code penalty**
- **Host OS impact not same as VM impact**
- **Missing Dom0 samples could explain the difference of HPL and SP behaviour in VM**
  - **Better tools and more investigation needed**

Conclusion

- **Based on our analysis of HPCC HPL and NAS SP, it is difficult to answer whether performance generalization in virtual environments is possible**
  - Yes (results allow for a performance model)
  - No (unpredictable performance, no fitting model)
  - **Can’t say (inconclusive results)**

Topics for Discussion and Further Work

- **General profiling issues**
  - **I/O, DMA, task accountability**
- **Oprofile/XenOprofile characteristics**
  - **Wall clock time vs. sample count**
  - **Sample frequency**
  - **Interference**
- **Xen's Dom0**
  - **Xentrace and event tracing**
  - **With/without profile comparing**
- **Performance isolation**

Questions or comments?