

ARCHITECTURAL PRINCIPLES AND EXPERIMENTATION OF
DISTRIBUTED HIGH PERFORMANCE VIRTUAL CLUSTERS

by

Andrew J. Younge

Submitted to the faculty of

Indiana University

in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

Department of Computer Science

Indiana University

September 2016

Copyright © 2016 Andrew J. Younge

All Rights Reserved

INDIANA UNIVERSITY

GRADUATE COMMITTEE APPROVAL

of a dissertation submitted by

Andrew J. Younge

This dissertation has been read by each member of the following graduate committee and by majority vote has been found to be satisfactory.

Date

Geoffrey C. Fox, Ph.D, Chair

Date

Judy Qiu, Ph.D

Date

Thomas Sterling, Ph.D

Date

D. Martin Swany, Ph.D

INDIANA UNIVERSITY

As chair of the candidate's graduate committee, I have read the dissertation of Andrew J. Younge in its final form and have found that (1) its format, citations, and bibliographical style are consistent and acceptable and fulfill university and department style requirements; (2) its illustrative materials including figures, tables, and charts are in place; and (3) the final manuscript is satisfactory to the graduate committee and is ready for submission to the university library.

Date

Geoffrey C. Fox, Ph.D
Chair, Graduate Committee

Accepted for the Department

Department Chair Name, Chair
Computer Science Program

Accepted for the College

Dean Name, Associate Dean
School of Informatics and Computing

ABSTRACT

ARCHITECTURAL PRINCIPLES AND EXPERIMENTATION OF DISTRIBUTED HIGH PERFORMANCE VIRTUAL CLUSTERS

Andrew J. Younge

Department of Computer Science

Doctor of Philosophy

With the advent of virtualization and Infrastructure-as-a-Service (IaaS), the broader scientific computing community is considering the use of clouds for their scientific computing needs. This is due to the relative scalability, ease of use, advanced user environment customization abilities, and the many novel computing paradigms available for data-intensive applications. However, a notable performance gap exists between IaaS and typical high performance computing (HPC) resources. This has limited the applicability of IaaS for many potential users, not only for those who look to leverage the benefits of virtualization with traditional scientific computing applications, but also for the growing number of big data scientists whose platforms are unable to build on HPC's advanced hardware resources.

Concurrently, we are at the forefront of a convergence in infrastructure between Big Data and HPC, in which a unified distributed computing archi-

tecture could provide computing and storage capabilities for both differing distributed systems use cases. There is the potential to leverage performance and advanced hardware from the HPC community, and provide it in a virtualized infrastructure using High Performance Virtual Clusters. This will not only enable a more diverse user environment within supercomputing applications, but also bring increased performance and capabilities to big data platform services.

This work proposes to bridge the gap between HPC and cloud infrastructure and enable infrastructure convergence through a framework for virtual clusters. It begins with an evaluation of current hypervisors and their viability to run HPC workloads within current infrastructure, which helps define existing performance gaps. Next, we uncover mechanisms to enable the use of specialized hardware available in many HPC resources, such as advanced accelerators like the Nvidia GPUs and high-speed, low-latency InfiniBand interconnects. The virtualized infrastructure that developed, which leverages such specialized HPC hardware and utilizes best-practices virtualization using KVM, supports advanced Molecular Dynamics simulations at near-native performance. These advances are incorporated into a framework for constructing distributed virtual clusters using the OpenStack cloud infrastructure. With high performance virtual clusters, we look to support a broad range of scientific computing challenges, from HPC simulations to big data analytics with a single, unified infrastructure.

ACKNOWLEDGMENTS

Thanks, Mom! (Acknowledgements TBD)

Contents

Table of Contents	viii
List of Figures	xi
1 Introduction	1
1.1 Overview	1
1.2 Research Statement	6
1.3 Research Challenges	11
1.4 Outline	15
2 Related Research	18
2.1 Virtualization	18
2.1.1 Hypervisors and Containers	21
2.2 Cloud Computing	23
2.2.1 Infrastructure-as-a-Service	28
2.2.2 Virtual Clusters	38
2.2.3 The FutureGrid Project	42
2.3 High Performance Computing	44
2.3.1 Brief History of Supercomputing	44
2.3.2 Distributed Memory Computation	46
2.3.3 Exascale	49
3 Analysis of Virtualization Technologies for High Performance Computing Environments	52
3.1 Abstract	52
3.2 Introduction	53
3.3 Related Research	55
3.4 Feature Comparison	57
3.4.1 Usability	59
3.5 Experimental Design	61
3.5.1 The FutureGrid Project	61
3.5.2 Experimental Environment	63
3.5.3 Benchmarking Setup	63

3.6	Performance Comparison	66
3.7	Discussion	71
4	Evaluating GPU Passthrough in Xen for High Performance Cloud Computing	75
4.1	Abstract	75
4.2	Introduction	76
4.3	Virtual GPU Directions	78
4.3.1	Front-end Remote API invocation	79
4.3.2	Back-end PCI passthrough	81
4.4	Implementation	83
4.4.1	Feature Comparison	84
4.5	Experimental Setup	85
4.6	Results	86
4.6.1	Floating Point Performance	87
4.6.2	Device Speed	88
4.6.3	PCI Express Bus	90
4.7	Discussion	92
4.8	Chapter Summary and Future Work	95
5	GPU-Passthrough Performance: A Comparison of KVM, Xen, VMWare ESXi, and LXC for CUDA and OpenCL Applications	96
5.1	Introduction	96
5.2	Related Work & Background	98
5.2.1	GPU API Remoting	98
5.2.2	PCI Passthrough	98
5.2.3	GPU Passthrough, a Special Case of PCI Passthrough	99
5.3	Experimental Methodology	100
5.3.1	Host and Hypervisor Configuration	100
5.3.2	Guest Configuration	102
5.3.3	Microbenchmarks	102
5.3.4	Application Benchmarks	103
5.4	Performance Results	104
5.4.1	SHOC Benchmark Performance	106
5.4.2	GPU-LIBSVM Performance	111
5.4.3	LAMMPS Performance	113
5.4.4	LULESH Performance	114
5.5	Lessons Learned	116
5.6	Directions for Future Work	118
6	Supporting High Performance Molecular Dynamics in Virtualized Clusters using IOMMU, SR-IOV, and GPUDirect	119
6.1	Introduction	119

6.2	Background and Related Work	122
6.2.1	GPU Passthrough	123
6.2.2	SR-IOV and InfiniBand	124
6.2.3	GPUDirect	126
6.3	A Cloud for High Performance Computing	126
6.4	Benchmarks	127
6.5	Experimental Setup	129
6.5.1	Node configuration	129
6.5.2	Cluster Configuration	131
6.6	Results	131
6.6.1	LAMMPS	132
6.6.2	HOOMD	133
6.7	Discussion	135
6.8	Chapter Summary	136
7	Virtualization advancements to support HPC applications	138
7.1	Memory Page Table Optimizations	139
7.2	Live Migration Mechanisms	142
7.2.1	RDMA-enabled VM Migration	144
7.2.2	Moving the Compute to the Data	147
7.3	Fast VM Cloning	148
7.4	Virtual Cluster Scheduling	151
7.5	Chapter Summary	154
8	Conclusion	155
8.1	Impact	157
	Bibliography	158

List of Figures

1.1	Data analytics and computing ecosystem compared (from [21]), with virtualization included	5
1.2	High Performance Virtual Cluster Framework	9
1.3	Architectural diagram for High Performance Virtual Clusters	10
2.1	Virtual Machine Abstraction	19
2.2	Hypervisors and Containers	22
2.3	View of the Layers within a Cloud Infrastructure	26
2.4	Eucalyptus Architecture	32
2.5	OpenNebula Architecture	36
2.6	Virtual Clusters on Cloud Infrastructure	39
2.7	FutureGrid Participants, Network, and Resources	43
2.8	Top 500 Development over time from 2002 to 2016 [8]	48
3.1	A comparison chart between Xen, KVM, VirtualBox, and VMWare ESX	57
3.2	FutureGrid Participants and Resources	62
3.3	Linpack performance	68
3.4	Fast Fourier Transform performance	69
3.5	Ping Pong bandwidth performance	70
3.6	Ping Pong latency performance (lower is better)	71
3.7	Spec OpenMP performance	72
3.8	Benchmark rating summary (lower is better)	73
4.1	GPU PCI passthrough within the Xen Hypervisor	84
4.2	GPU Floating Point Operations per Second	88
4.3	GPU Fast Fourier Transform	89
4.4	GPU Matrix Multiplication	90
4.5	GPU Stencil and S3D	91
4.6	GPU Device Memory Bandwidth	92
4.7	GPU Molecular Dynamics and Reduction	93
4.8	GPU PCI Express Bus Speed	94

5.1	SHOC Levels 0 and 1 relative performance on Bespin system. Results show benchmarks over or under-performing by 0.5% or greater. Higher is better.	107
5.2	SHOC Levels 1 and 2 relative performance on Bespin system. Results show benchmarks over or under-performing by 0.5% or greater. Higher is better.	108
5.3	SHOC Levels 0 and 1 relative performance on Delta system. Benchmarks shown are the same as Bespin's. Higher is better.	109
5.4	SHOC Levels 1 and 2 relative performance. Benchmarks shown are the same as Bespin's. Higher is better.	110
5.5	GPU-LIBSVM relative performance on Bespin system. Higher is better.	112
5.6	GPU-LIBSVM relative performance on Delta showing improved performance within virtual environments. Higher is better.	113
5.7	LAMMPS Rhodopsin benchmark relative performance for Bespin system. Higher is better.	114
5.8	LAMMPS Rhodopsin benchmark relative performance for Delta system. Higher is better.	115
5.9	LULESH relative performance on Bespin. Higher is better.	116
6.1	Node PCI Passthrough of GPUs and InfiniBand	130
6.2	LAMMPS LJ Performance	132
6.3	LAMMPS RHODO Performance	134
6.4	HOOMD LJ Performance with 256k Simulation	135
7.1	Transparent Huge Pages with KVM	141
7.2	Adding extra specs to a VM flavor in OpenStack	152

¹ Chapter 1

² Introduction

³ 1.1 Overview

⁴ For years visionaries in computer science have predicted the advent of utility-based
⁵ computing. This concept dates back to John McCarthy's vision stated at the MIT
⁶ centennial celebrations in 1961.

⁷ “If computers of the kind I have advocated become the computers of the
⁸ future, then computing may someday be organized as a public utility just
⁹ as the telephone system is a public utility... The computer utility could
¹⁰ become the basis of a new and important industry.“

¹¹ Only recently has the hardware and software become available to support the concept
¹² of utility computing on a large scale.

¹³ The concepts inspired by the notion of utility computing have combined with
¹⁴ the requirements and standards of Web 2.0 [1] to create Cloud computing [2–4].
¹⁵ Cloud computing is defined as, “A large-scale distributed computing paradigm that
¹⁶ is driven by economies of scale, in which a pool of abstracted, virtualized, dynamically-
¹⁷ scalable, managed computing power, storage, platforms, and services are delivered on

18 demand to external customers over the Internet.” This concept of cloud computing
19 is important to Distributed Systems because it represents a true paradigm shift [5]
20 within the entire IT infrastructure. Instead of adopting the in-house services, client-
21 server model, and mainframes, clouds push resources out into abstracted services
22 hosted *en masse* by larger organizations. This concept of distributing resources is
23 similar to many of the visions of the Internet itself, which is where the “clouds”
24 nomenclature originated, as many people depicted the internet as a big fluffy cloud
25 one connects to.

26 As the core of most cloud infrastructure lies virtualization, a computer architecture
27 technology by which 1 or more Virtual Machines (VMs) are run on the same physical
28 host. In doing this, a layer of abstraction is inserted between and around the hardware
29 and Operating System (OS). Specifically, hardware resources such as CPUs, memory,
30 and I/O devices, and software resources analogous to OS functionality and low level
31 libraries are abstracted and provided to VMs directly. While virtualization has existed
32 for many years, its availability with Intel x86 commodity hardware in conjunction with
33 the rise of clouds has brought virtualization to the forefront of distributed systems.

34 While cloud computing is changing IT infrastructure, it also has had a drastic
35 impact on distributed systems itself, which has a different evolution. Gone are the
36 IBM Mainframes of the 1980’s which dominated the enterprise landscape. While
37 some mainframes still exist, they are used only for batch related processing tasks
38 and are relatively unused for scientific applications as they are inefficient at Floating
39 Point Operations. As such, they were replaced with Beowulf Clusters [6], Massively
40 Parallel Processors (MPPs) and Supercomputers of the 90’s and 00’s. A novelty of
41 these distributed memory systems is that instead of just one large machine, many
42 machines are connected together and used to achieve a common goal, thereby maxi-
43 mizing the overall speed of computation. Clusters represent a more commodity-based

44 supercomputer, where off the shelf CPUs are used instead of the highly customized
45 and expensive processors and interconnects found in Supercomputers.

46 Supercomputers and Clusters are best suited for large scale applications. These
47 HPC applications can even include “Grand Challenge” applications [7] and can repre-
48 sent a sizable amount of the scientific calculations done on large-scale Supercomputing
49 resources today. However, there exists a gap of many orders of magnitude between
50 leading-class high performance computing and what is available on the common labo-
51 ratory workshop. This gap, described here as mid-tier scientific computation is a fast
52 growing field that struggles to efficiently harness distributed systems while hoping to
53 minimize extensive development efforts. These mid-tier scientific endeavours need to
54 leverage distributed systems to effectively complete the calculations at hand, however
55 may not require the extreme scale provided by the latest machines at the peak of the
56 Top500 list [8]. It simply may not be feasible for small research teams to effectively
57 handle the development complexity of extreme-scale resources, and instead look to-
58 wards other options. This can include some scientific disciplines such as high energy
59 physics [9], materials science [10], bioinformatics [11], and climate research [12], to
60 name a few.

61 As more domain science turns to the aid of computational resources for con-
62 ducting novel scientific endeavours, there is a continuing and growing need for na-
63 tional cyberinfrastructure initiatives to support an increasingly diverse set of scientific
64 workloads. Substantial growth can be see in the number of computational resource
65 requests [13, 14] from many of the larger computational facilities. Concurrently, there
66 has also been an increase in accelerators and hybrid computing models capable of
67 quickly providing additional resources [15] beyond commodity clusters.

68 Historically, application diversity was separated into High Performance Comput-
69 ing (HPC) and High Throughput Computing (HTC). With HTC, computational

70 problems can be separated into independent tasks that can execute in a pleasingly
71 parallel fashion, happily gaining any available resources and rarely require significant
72 communication or synchronization between tasks. HPC often represents computa-
73 tional problems that require significant communication and coordination to effectively
74 produce results, usually with the use of a communication protocol such as MPI [16].
75 Recently however, many big-data paradigms [17] have been introduced in distributed
76 systems that represent new computational models for distributed computing, such as
77 MapReduce [18] and the corresponding Apache Big Data stack [19, 20]. Supporting
78 these different distributed computational paradigms requires a flexible infrastructure
79 capable of providing computational resources for all possible models in a fast and
80 efficient manner.

81 Currently we are at the forefront of a convergence within scientific computing be-
82 tween HPC and big data computation [21]. This amalgamation of historically differing
83 viewpoints of Distributed Systems looks to combine the performance characteristics
84 of HPC and the pursuit towards Exascale with the data and programmer oriented
85 concurrency models found in Big Data and cloud services.

86 Much of the convergence effort has been focused on applications and platform
87 services. Specifically, significant work towards convergent applications has been out-
88 lined with the Big Data Ogres [22] and the corresponding HPC-ABDS model [23].
89 This convergence can also be seen with efforts in bringing interconnect advances to
90 classically big data platforms such as with InfiniBand and MapReduce [24]. However,
91 the underlying hardware and OS environments are still something to be reconciled,
92 and represents something that virtualization can potentially provide. It is expected
93 that new big data efforts will continue to move in this direction [25], especially if
94 virtualization can make HPC hardware that's traditionally prohibitive in such areas,
95 such as accelerators and high-speed interconnects, readily available to cloud and big

96 data platforms. As the deployment of big data applications and platform services on
 97 virtualized infrastructure is well defined and studied [26], this dissertation has focused
 98 the difficulty of running HPC applications on similar virtualized infrastructure. How-
 99 ever, it is possible and hopeful that research regarding virtualization can also play a
 100 part in bringing advanced hardware and performance-focused considerations to Big
 101 Data applications, effectively cross-cutting the convergence with HPC. In Summary,
 102 success of the research in virtualization could be defined by the ability to support the
 103 convergence between HPC and Big Data.

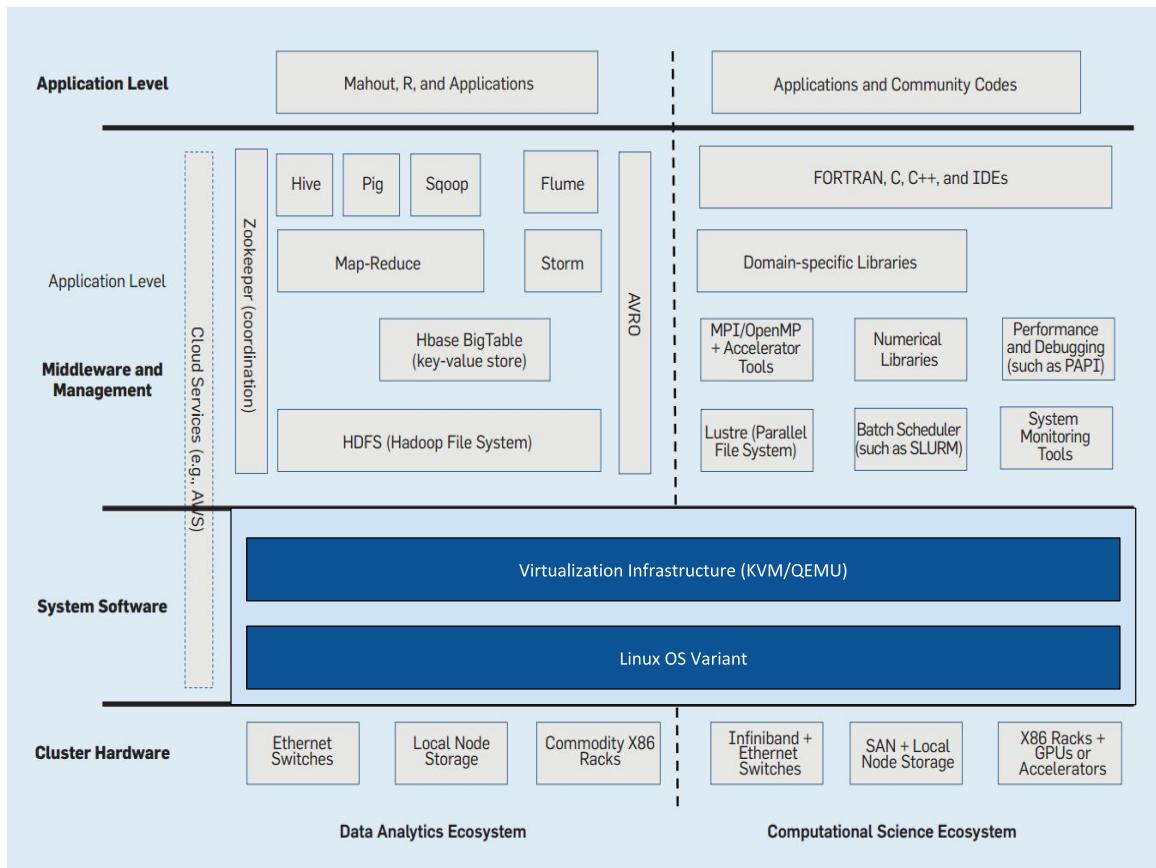


Figure 1.1 Data analytics and computing ecosystem compared (from [21]), with virtualization included

104 To further illustrate where virtualization can play a part in HPC and Big Data
 105 convergence, we look at Figure 1.1 from Reed & Dongarra [21]. While the two ecosys-

106 tems depicted are only representative and in no way exhaustive, they do show how
107 drastically different user environments exist and are reliant on differing hardware. If
108 we insert a performance-oriented virtualization mechanism within the system soft-
109 ware capable of handling the advanced cluster hardware and perform at near-native
110 speeds (at or under 5% overhead, as loosely defined in [27]), it could provide a single,
111 comprehensive *convergent ecosystem* that supports both HPC and Big Data efforts
112 at a critical level.

113 This work proposes the use of virtual clusters [28] to provide distinct, separated
114 environments on similar resources using virtual machines. These virtual clusters,
115 similar in design to the Beowulf clusters and commodity HPC systems, provide a
116 distributed memory environment, usually with a local resource management system
117 [29]. However, past concerns with virtualization have limited the adoption of virtual
118 clusters in many large scale cyberinfrastructure deployments. This has largely been
119 due to the overhead of virtualization, whereby many scientific computations have
120 experienced a significant and notable degradation in performance. In an ecosystem
121 familiarized with HPC systems where performance is paramount, this has been an
122 obstructive hurdle for deploying many tightly coupled applications.

123 1.2 Research Statement

124 With the rise of cloud computing within the greater realm of distributed systems,
125 there have been a number of scientific computing endeavors map well to cloud in-
126 frastructure. This first includes the simple and most common practice of on demand
127 computing where by users can rent-a-workstation [30]. Perhaps these resources are
128 more powerful than a given researcher's laptop and used to run their scientific applica-
129 tions or support greater laboratory collaborative efforts, such as a shared database or

130 Web services. We've also seen virtualized cloud infrastructure support high through-
131 put computing very well. Often times pleasingly parallel applications, be it from high
132 energy physics such as the LHC effort [9, 31] or bioinformatics with BLAST align-
133 ment jobs [11], have proven to run with high efficiency in public and private cloud
134 environments. Furthermore, the rise of public cloud infrastructure has also coincided
135 with increase in big data computation and analytics. Many of these big data plat-
136 form services have evolved complimentary to cloud infrastructure, and as such have
137 a symbiotic relationship with virtualization technologies [32].

138 However, with tightly coupled, high performance distributed memory applications,
139 the same endeavors that support leading class scientific efforts, run very poorly on
140 virtualized cloud infrastructure [33]. This is due to a myriad of addressable reasons
141 ranging from scheduling, abstraction overhead, or a lack of advanced hardware sup-
142 port necessary for tightly coupled communication. This postulates the question on
143 whether virtualization can in fact support such tightly coupled large scale applica-
144 tions without an imposed significant performance penalty. Simply put, *the goal of this*
145 *dissertation is to investigate the viability of mid-tier scientific applications supported*
146 *in virtualized infrastructure.*

147 Historically, mid-tier scientific applications are distributed memory HPC applica-
148 tions that require more complex process communication mechanisms. These systems
149 need far more performance than a single compute resources (such as a workstation)
150 can provide. This could include hundreds or thousands of processes calculating and
151 communicating concurrently on a cluster, perhaps using a messaging interface such
152 as MPI. These applications are likely distinct either in application composition or op-
153 erating parameters, from extreme-scale HPC applications that run at the highest end
154 of the supercomputing resources today operating on petascale machines and beyond.

155 Given the current outlook on virtualization for supporting HPC applications, this

156 dissertation proposes a framework for High Performance Virtual Clusters that enable
157 advanced computational workloads, including tightly coupled distributed memory ap-
158 plications, to run with a high degree of efficiency in virtualized environments. This
159 framework, outlined in Figure 1.2, illustrates the topics to be addressed to provide
160 a supportive virtual cluster environment for high performance mid-tier scientific ap-
161 plications. Areas marked in darker green indicate topics this dissertation may touch
162 upon, whereas light green areas in Figure 1.2 identify outstanding considerations to
163 be investigated. We specifically identify mid-tier distributed memory parallel compu-
164 tations as a focal point for the computational challenges at hand as a way to separate
165 from some of the latest efforts in towards Exascale [34–36] computing. While virtu-
166 alization may in fact be able to play a role towards usable Exascale computing, such
167 efforts fall outside the immediate scope of this dissertation.

168 In order to provide comprehensive high performance virtual clusters, we need to
169 first look at a key area itself, the virtualized infrastructure itself. At the core, we
170 have to consider the hypervisor or virtual machine monitor and the overhead and
171 performance characteristics associated with it. This includes performance tuning
172 considerations, NUMA effects, and advanced hardware device passthrough. Specifi-
173 cally, device passthrough in the context of this manuscript refers to two major device
174 types; GPUs and InfiniBand interconnects (the later using SR-IOV). The virtual in-
175 frastructure also must consider scheduling as a major factor in performing efficient
176 placement of workloads on the underlying host infrastructure, and in particular a
177 Proximity scheduler is of interest [37]. Storage solutions in a virtualized environ-
178 ment is an increasingly important aspect of this framework, as both HPC and big
179 data solutions are continuing to prioritize I/O performance compared to computa-
180 tion. Storage is also likely to be heavily dependent on interconnect considerations as
181 well, as potentially provided by device passthrough, however such I/O considerations

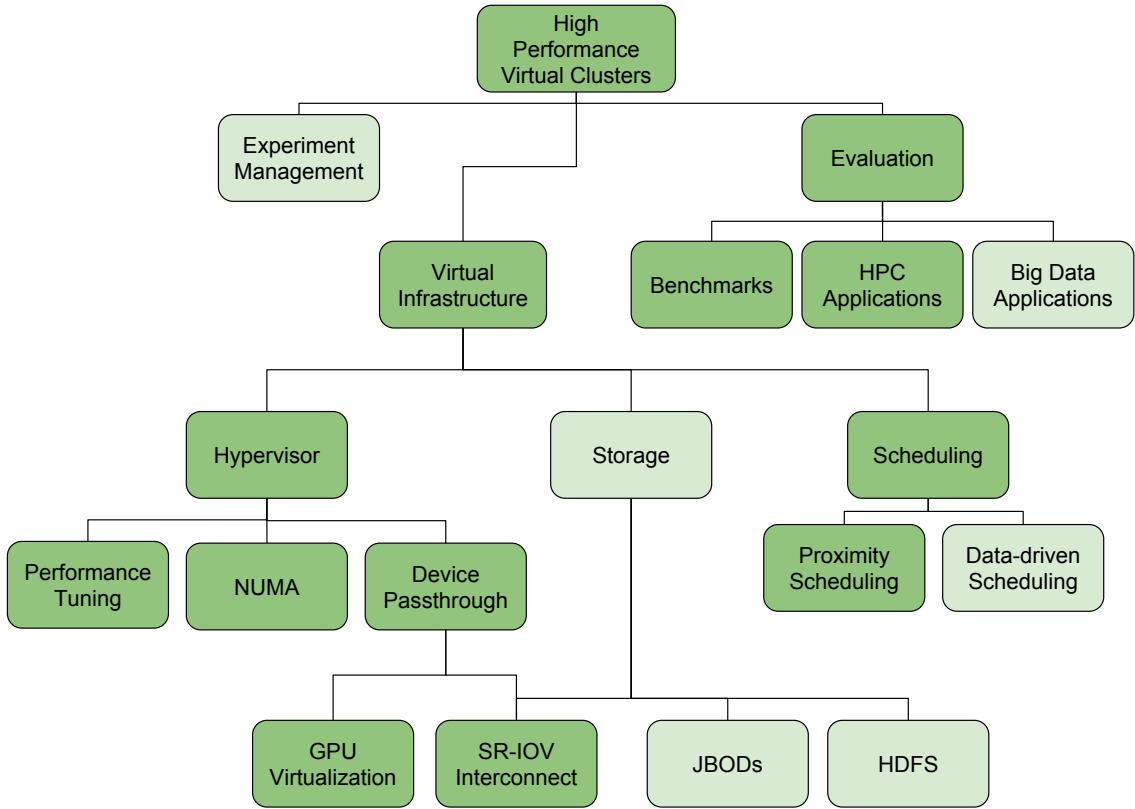


Figure 1.2 High Performance Virtual Cluster Framework

182 lie beyond this dissertation's immediate scope.

183 However, simply providing an enhanced virtualized infrastructure may not guar-
 184 antee that all implementations of high performance virtual clusters are performant.
 185 Specifically, proposed infrastructures need to be properly evaluated in a systematic
 186 way through the use of a wide array of benchmarks, mini-applications, and full-scale
 187 scientific applications. This effort can further be broken separated into three major
 188 problem sets; base level benchmarking tools, HPC applications, and big data appli-
 189 cations. Evaluating the stringent performance requirements of all three sets, when
 190 compared with bare metal (no virtualization) solutions, will illuminate not only suc-
 191 cessful designs but also the focus areas that require more attention. As such, we
 192 look to continually use these benchmarks and applications as a tool to measure the

₁₉₃ viability of virtualization in this context.

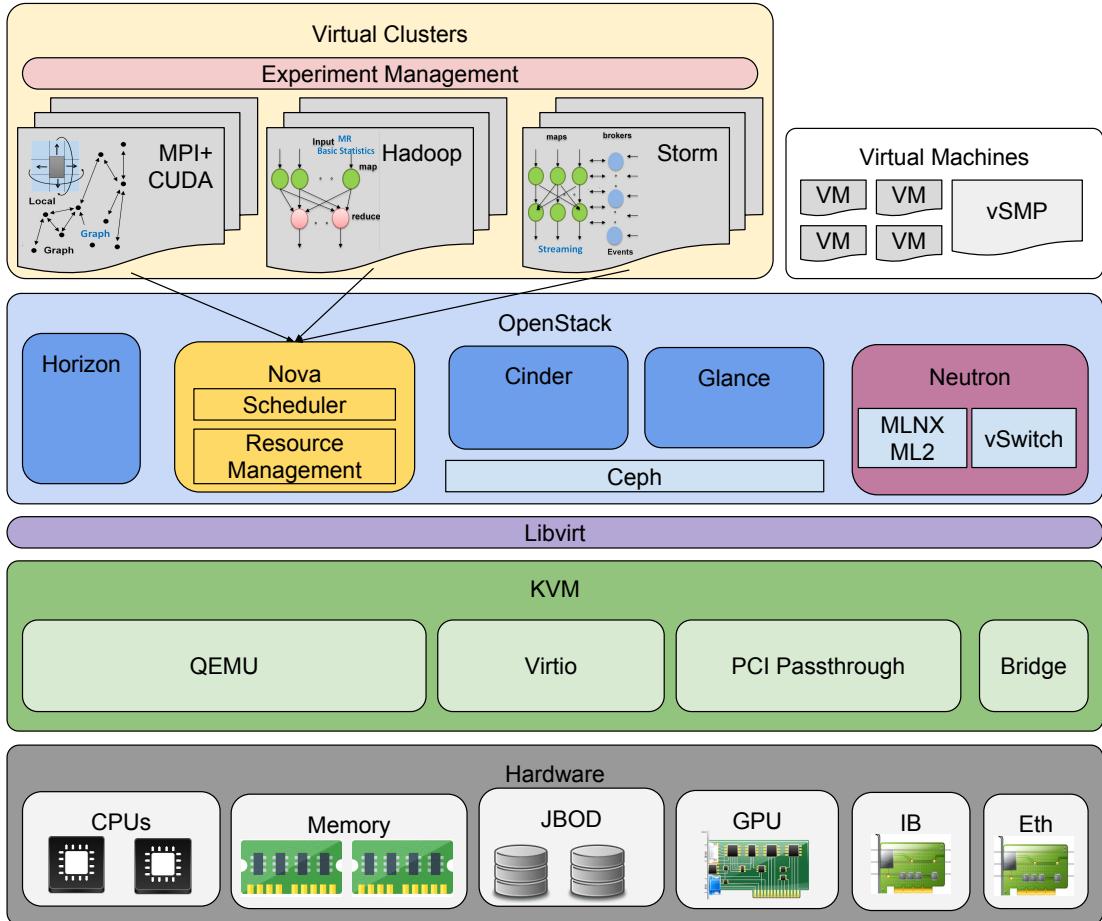


Figure 1.3 Architectural diagram for High Performance Virtual Clusters

₁₉₄ Building from the historical virtual clusters in Grid computing, we see a new archi-
₁₉₅ tectural model for high performance virtual clusters illustrated in 1.3 that is implied
₁₉₆ by this dissertation. Here, we leverage commodity hardware, as well as some ad-
₁₉₇ vanced HPC hardware. While this notion could incorporate a wide array of differing
₁₉₈ technologies, we focus here on GPU-based accelerators and InfiniBand interconnects
₁₉₉ in conjunction with x86 CPUs. Atop this, we leverage KVM and QEMU to provide an
₂₀₀ advanced hypervisor for creating and hosting VMs with direct hardware involvement
₂₀₁ from the lower level. Moving up, the Libvirt API is leveraged due to its hyper-

visor interoperability and popularity. Atop Libvirt is the OpenStack private cloud infrastructure. In Figure 1.3 we illustrate some (but not all) of OpenStack’s services including the Horizon UI, Cinder and Glance storage mechanisms, and the Neutron (previously Quantum) components. The Nova component of OpenStack is the point of focus for providing comprehensive VM management.¹ Atop OpenStack, we can create a wide array of virtual clusters and machines to support the wide ranging scientific computing ecosystems necessary. This includes application models ranging from tightly coupled MPI+CUDA HPC applications, to emerging big data analytics toolkits such as Apache Storm [38].

One of the higher-level aspects of providing high performance virtual clusters is the high level orchestration of the virtual clusters themselves, which we term experiment management. While this largely remains tangential to this immediate research, it is none the less a key aspect for a successful solution. Some effort has been put forth for virtual cluster experiment management [39], and many ongoing open sources solutions also offer compelling options, such as OpenStack Heat [40]. An example of a project delivering advanced orchestration mechanisms and a toolkit to aid in configurable virtual clusters on heterogeneous IaaS deployments is the Cloudmesh effort [41].

1.3 Research Challenges

The framework, architecture, and efforts described in this dissertation represent a movement forward in providing virtualized infrastructure to support a wide arrangement of scientific applications. However, there still exist some challenges that will need

¹While many of the features for nova’s additions in GPU Passthrough and SR-IOV InfiniBand support have been put together at USC/ISI as an OpenStack Nova fork (<https://libraries.io/github/usc-isi/nova>), much of the features have since been modified and matured by the OpenStack community in later releases and made available in upstream Nova.

223 to be addressed. This includes a sigma of virtualization being inherently slow and
224 unable to support tightly coupled computations, limitations with advancing at scale,
225 and even that containers may provide a better alternative. While this work hopes to
226 move beyond these challenges, they none the less must be considered.

227 The notion that virtualization and Cloud infrastructure are not able to support
228 parallel distributed memory applications has been characterized many times. One
229 of the most prominent examples of this is the Department of Energy’s Magellan
230 Project [42], where by the Magellan Final Report [43] states the following finding as
231 a Key Finding:

232 **“Finding 2. Scientific applications with minimal communication
233 and I/O are best suited for clouds.”**

234 We have used a range of application benchmarks and micro-benchmarks
235 to understand the performance of scientific applications. Performance of
236 tightly coupled applications running on virtualized clouds using commod-
237 ity networks can be significantly lower than on clusters optimized for these
238 workloads. This can be true at even mid-range computing scales. For ex-
239 ample, we observed slowdowns of about 50x for PARATEC on the Amazon
240 EC2 instances compared to Magellan bare-metal (non-virtualized) at 64
241 cores and about 7x slower at 1024 cores on Amazon Cluster Compute in-
242 stances, the specialized high performance computing offering. As a result,
243 current cloud systems are best suited for high-throughput, loosely coupled
244 applications with modest data requirements.“

245 These findings underscore how classical usage of virtualization in cloud infrastruc-
246 ture has serious performance issues when running tightly coupled distributed memory
247 applications. Many of these performance concerns are sound, given the limitation of a

248 number of virtualization overheads commonplace at the time, including shadow page
249 tables, emulated Ethernet drivers, experimental hypervisors, and a complete lack of
250 sophisticated hardware commonplace in supercomputers and clusters. As a result, the
251 advantages of virtualization, including on-demand resource allocation, live migration
252 and advanced hybrid migration, and user-defined environments, have not been able
253 to effectively show their value in the context of the HPC community.

254 Other and related efforts within the scientific community too found limitations
255 with HPC applications in public cloud environments. This includes the study by
256 Jackson et. al [44] which illustrates how the Amazon Elastic Compute Cloud (EC2)
257 creates a 6x performance impact compared to a local cluster, due to a large part
258 on the limiting Gigabit Ethernet that benchmarks relied heavily on within the EC2
259 system. Other studies also found similar results such as Ostermann [33], concluding
260 that Amazon EC2 “is insufficient for scientific computing at large, though it still
261 appeals to the scientists that need resources immediately and temporarily.” However,
262 these studies are now outdated and do not take into account the advancements in
263 virtualization and hardware availability detailed in this dissertation. Specifically, it is
264 estimated that with the KVM hypervisor in a performance-tuned environment, using
265 accelerators and most certainly a high-speed, low latency interconnect as detailed in
266 Chapter 6, the results would be drastically different.

267 One limitation in this research in high performance virtual clusters is the fact
268 that the degree to which applications can scale remains relatively unknown. While
269 initial results with SR-IOV InfiniBand are promising, scaling is naturally hard to
270 predict. While unfounded, it would be hypothetically possible that as the number
271 of VM’s increases, tail-latency could also increase, causing notable slowdowns during
272 distributed memory synchronization barriers. It is only when infrastructure comes
273 available to support high performance virtual clusters will scaling beyond to thousands

274 of cores and beyond be investigated.

275 Another potential challenge, and perhaps also a strength, is the rising use of
276 containers within industry, such as with efforts like Docker [45]. Recently, we've seen
277 efforts at NERSC/LBNL adapt a container solution called Shifter with the SLURM
278 resource manager on CRAY XC based systems [46]. Shifter's goal is to provide
279 user defined images for NERSC's bioinformatics users, and it adapts remarkably well
280 to the HPC environment. While further efforts are needed by Cray and NERSC
281 to fully provide a container-bases solution on a large scale Supercomputer for all
282 applications, its efforts are in many ways parallel to using virtualization. In Chapter 5,
283 we specifically compare virtualization efforts with LXC [47], a popular Linux container
284 solution and find performance to be comparable and largely near-native.

285 While the major concern with virtualization in the HPC community is perfor-
286 mance issues, virtualization itself may not be fundamentally limited by the overhead
287 that causes issues in running high performance computing applications. Recent im-
288 provements in performance, along with increased usability of accelerators and high
289 speed, low latency interconnects in virtualized environments as demonstrated in this
290 dissertation, have made virtual clusters a more viable solution for mid-tier sci-
291 entific applications. Furthermore, it is possible for virtualization technologies to bring
292 enhanced usability and enable specialized runtime components to future HPC re-
293 sources, adding significant value over today's supercomputing resources. This could
294 potentially include infrastructure advances for higher level cloud platform services for
295 supporting big data applications [23].

296 1.4 Outline

297 The rest of this dissertation is organized into chapters, each signifying the steps to
298 move forward the notion of a high performance virtual cluster solution.

299 Chapter 2 investigates the related research surrounding both cloud computing
300 and high performance computing. Within cloud computing, an introduction to cloud
301 infrastructure, virtualization, and containers will all be discussed. This also includes
302 details regarding virtual clusters as well as an overview of some national scale cloud
303 infrastructure efforts that exist. Furthermore, we investigate the state of high per-
304 formance computing and supercomputing, as well touch upon some of the current
305 Exascale efforts.

306 Chapter 3 takes a look at the potential for virtualization, in a base case, for high
307 performance computing. This includes a feature comparison for hardware availability
308 of a few common hypervisors, specifically Xen, KVM, VirtualBox, and VMWare.
309 Then, a few common HPC benchmarks are evaluated to determine what overhead
310 exists and where in a single node configuration. This identifies how in some scenarios,
311 virtualization adds only a minor overhead, whereas with other scenarios, overheads
312 can be up to 50% compared to native configurations.

313 Chapter 4 starts to overcome one of the main limitations of virtualization for use
314 in advanced scientific computing, specifically the lack of hardware availability. In this
315 chapter, The Xen hypervisor is used to demonstrate the affect of GPU Passthrough,
316 allowing for GPUs to be used in a guest VM. The efficiency of this method is briefly
317 evaluated using two different hardware setups, and finds hardware can play a notable
318 role in single node performance.

319 Chapter 5 continues where chapter 4 leaves off, by demonstrating that GPU
320 passthrough is possible on many other hypervisors, specifically also KVM and VMWare,

321 and compares with one of the main containerization solutions, LXC. Here, the GPUs
322 are evaluated using not only the SHOC GPU benchmark suite developed at Oak
323 Ridge National Laboratory, but also a diverse mix of real-world applications to ex-
324 amine how and where overhead exists with GPUs in VMs for each virtualization setup.
325 Specifically, we find that with properly tuned hardware and NUMA-balanced configu-
326 rations, that the KVM solution can perform at roughly near-native performance, with
327 on average 1.5% overhead compared to no virtualization. This illustrates that with
328 the combination correct hypervisor selection, careful tuning, and advanced hardware,
329 scientific computations can be supported using virtualized hardware.

330 Chapter 6 takes the findings from the previous chapter to the next level. Specifi-
331 cally, the lessons learned from successful KVM virtualization with GPUs is expanded
332 and combined with a missing key component of supporting advanced parallel com-
333 putations: a high speed, low latency interconnect, specifically InfiniBand. Using
334 SR-IOV and PCI passthrough of QDR InfiniBand interconnect across a small cluster,
335 it is demonstrated that two Molecular Dynamics simulations, both very commonly
336 used in the HPC community, can be run at near-native performance in the designed
337 virtual cluster.

338 Chapter 7 takes a look at the given situation of virtualization, and puts forth an
339 argument for enhancements forthcoming in high performance virtual cluster solutions.
340 Specifically, we look at the given state of the art, how virtual clusters can be used
341 to provide an infrastructure to support the convergence between HPC and big data.
342 Specifically, this chapter outlines and investigates potential next steps for virtualiza-
343 tion, including the potential for advanced live migration techniques and VM cloning,
344 which can be made available with the inclusion of a high-performance RDMA-capable
345 interconnect.

346 Finally, this work concludes with an overall view of the current state of high

³⁴⁷ performance virtualization, as well as it's potential to impact and support a wide
³⁴⁸ array of disciplines.

³⁴⁹ Chapter 2

³⁵⁰ Related Research

³⁵¹ In order to accurately depict the research presented in this article, the topics within
³⁵² Virtualization, Cloud computing, and High Performance Computing are reviewed in
³⁵³ detail.

³⁵⁴ 2.1 Virtualization

³⁵⁵ Virtualization is a way to abstract the hardware and system resources from an oper-
³⁵⁶ ating system. This is typically performed within a Cloud environment across a large
³⁵⁷ set of servers using a Hypervisor or Virtual Machine Monitor (VMM) which lies in
³⁵⁸ between the hardware and the Operating System (OS). From here, one or more vir-
³⁵⁹ tualized OSs can be started concurrently as seen in Figure 2.1, leading to one of the
³⁶⁰ key advantages of virtualization. This, along with the advent of multi-core process-
³⁶¹ ing capabilities, allows for a consolidation of resources within a data center. It then
³⁶² becomes the cloud infrastructure’s job, as discussed in the next section to exploit this
³⁶³ capability to its maximum potential while still maintaining a given QoS. It should
³⁶⁴ be noted that virtualization is not specific to Cloud computing. IBM originally pi-

³⁶⁵ oneered the concept in the 1960's with the M44/44X systems. It has only recently
³⁶⁶ been reintroduced for general use on x86 platforms.

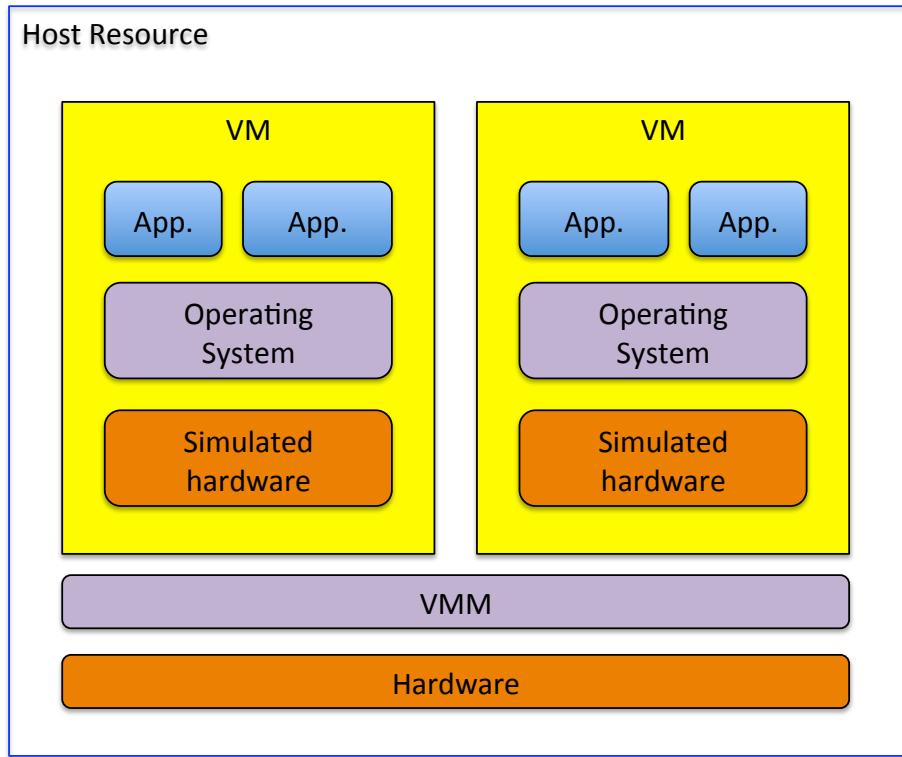


Figure 2.1 Virtual Machine Abstraction

³⁶⁷ However, virtualization is in the most general sense is just another form of ab-
³⁶⁸ straction. As such, there are in fact many levels to virtualization that exist [?].

- ³⁶⁹ • **ISA -** Virtualization can start from the instruction set architecture (ISA) level,
³⁷⁰ where by an entire processor instruction set is emulated. This may be useful for
³⁷¹ running software or services developed for one instruction set (say MIPS) but
³⁷² has to run on modern Intel x86 hardware. ISA level virtualization is usually
³⁷³ emulated through an interpreter which translates source instructions to target
³⁷⁴ instructions, however this can often be extremely inefficient. Dynamic binary
³⁷⁵ translation can help aid in efficiency by translating blocks of source instructions
³⁷⁶ to target instructions, however this still can be limiting.

- 377 ● **Hardware -** One of the most important technologies in cloud computing is
378 hardware level virtualization [48, 49]. Hardware virtualization, in its most pure
379 form, refers to the process of creating virtual abstraction to hardware platforms,
380 operating systems, or software resources. This enables the creation of 1 or
381 more virtual machines (VMs) that are run concurrently on the same operating
382 environment, be it hardware or some higher software. Here, a virtual hardware
383 environment is generated for a VM, providing virtual processors, memory, and
384 I/O devices that allow for VM multiplexing, as depicted in Figure 2.1. This
385 layer also manages the physical hardware on behalf of a host OS as well as for
386 guests. While the most common type of hardware virtualization is with the
387 Xen Virtual Machine Monitor [48], this method has further separated into type
388 1 and type 2 hypervisors, as detailed further in Section 2.1.1.

- 389 ● **Operating System -** Moving up the latter of implementation with virtu-
390 alization, we find OS level virtualization, where multiplexing happens at the
391 level of the OS, rather than the hardware. Usually, this refers to isolating a
392 filesystem and associated process and runtime effects in a single "chroot" or
393 *container* environment at the user level, where all system level operations are
394 still handled by a single OS kernel. This containerization allows for multiple
395 user environments to exist concurrently without the complexity of hardware or
396 ISA level virtualization. Containers are also described in more detail in the
397 next section.

- 398 ● **Library (API) -** Moving up a layer, we find library or Application Level
399 Interface (API) level virtualization, where a programmaming interface is separated
400 and the communication between this API and the back-end library with the
401 computation is virtualized. This method removes the back end services from

402 within the operating environment, which can give the effect that resources are
403 local when really they are remote. This is how GPUs are "virtualized" with
404 tools such as rCUDA [50] and vCUDA [51]. This method's performance often
405 can be very dependent on the underlying communications mechanisms, as well
406 as complete virtualization of the whole API (which may be difficult). API
407 virtualization also exists for entire OS libraries to translate between differing
408 OS, without actual containerization.

409 • **Process -** Virtualization can also take the form at the process or user level,
410 which can then be used to deliver a specialized or high level language that is the
411 same across several different OSs. This is how the Java Virtual Machine (JVM)
412 and Microsoft's .NET platform operate. This type of application predominantly
413 falls outside of the scope of this dissertation.

414 2.1.1 Hypervisors and Containers

415 While there are many types of virtualization, this dissertation predominately focuses
416 on hardware and OS level virtualization. With hardware virtualization, a Virtual
417 Machine Monitor, or hypervisor, is used. Abstractly, a hypervisor is a piece of software
418 that creates virtual machines or *guests*, usually that model the underlying physical
419 machine *host* system.

420 Hardware virtualization can actually be dissected in more detail to two different
421 types of hypervisors. These hypervisor types are illustrated in Figure 2.2 and they
422 can be directly compared to containers. With a Type 1 virtualization system, the
423 hypervisor or VMM sits directly on the bare-metal hardware, below the OS. These na-
424 tive hypervisors provide direct control of the underlying hardware, and are controlled
425 and operated usually through the use of a privileged VM. One example of a type 1

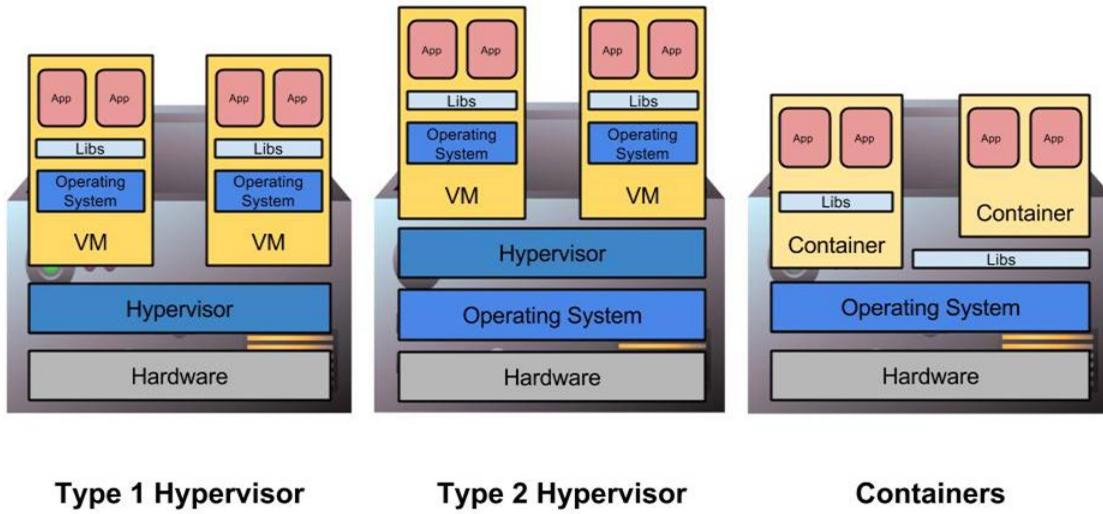


Figure 2.2 Hypervisors and Containers

426 hypervisor is the Xen Virtual Machine Monitor [48], which uses its VMM as well as a
 427 privileged Linux OS, called Dom0, to create and manage other user DomU instances.
 428 The VMM in this place provides the necessary hardware abstraction of CPU, mem-
 429 ory, and some I/O aspects, leaving the control aspects of the other DomUs to Dom0.
 430 With a Type 1 hypervisor, all virtualization functions are kept separate from control
 431 and OS functionality, effectively making a cleaner design. This design could lead to
 432 end application performance implications, as illustrated with the Palacios VMM [27].

433 Type 2 hypervisors utilize a different, and sometimes involve a more convoluted
 434 design. With a Type 2 hypervisor, there is a "host" OS that, like with native OSs,
 435 sits directly atop hardware. This OS is just like any normal native OS. However, the
 436 OS itself can abstract its own hardware, and provide and manage a VM, effectively
 437 as an OS process. In this case, the hypervisor providing the abstraction is effec-
 438 tively built within, atop, or as a module within the kernel. There are many different
 439 Type 2 hypervisors, the most common of which is the Linux Kernel Virtual Machine
 440 (KVM) [52]. KVM is often used in conjunction with QEMU, a ISA level hypervisor
 441 to provide some basic ISA level virtualization and emulation capabilities. KVM is

442 simply provided as a Linux kernel module within a given host, and guest VMs are
443 run as a single process on the host OS.

444 Type 1 and Type 2 hypervisors are very distinct from OS level virtualization, also
445 known as containerization. With containers, there is a single OS, however instead of
446 direct hardware abstraction, a single kernel is used to simultaneously run multiple
447 user-space instances in a jailed-root environment. These environments may look and
448 feel like a separate machine, but in fact are not. Often times the kernel itself pro-
449 vides resource management tools to help control resource utilization and allocations.
450 Linux container (LXC) is a great example of this, with their use of namespaces for
451 filesystem control and cgroups for resource management. With the recent advent of
452 Docker, which looks to control versioning and easy deployment of traceable contain-
453 ers, this aspect of OS level virtualization has grown in popularity, however security
454 and usability concerns still exist, as there is not dedicated isolation mechanisms and
455 the kernel space is the only point of security separation.

456 2.2 Cloud Computing

457 Cloud computing [53] is one of the most explosively expanding technologies in the
458 computing industry today. However it is important to understand where it came
459 from, in order to figure out where it will be heading in the future. While there is no
460 clear cut evolutionary path to Clouds, many believe the concepts originate from two
461 specific areas: Grid Computing and Web 2.0.

462 Grid computing [54, 55], in its practical form, represents the concept of connect-
463 ing two or more spatially and administratively diverse clusters or supercomputers
464 together in a federating manner. The term “the Grid” was coined in the mid 1990’s
465 to represent a large distributed systems infrastructure for advanced scientific and en-

466 gineering computing problems. Grids aim to enable applications to harness the full
467 potential of resources through coordinated and controlled resource sharing by scalable
468 virtual organizations. While not all of these concepts carry over to the Cloud, the
469 control, federation, and dynamic sharing of resources is conceptually the same as in
470 the Grid. This is outlined by Foster et al [3], as Grids and Clouds are compared at an
471 abstract level and many concepts are similar. From a scientific perspective, the goals
472 of Clouds and Grids are also similar. Both systems attempt to provide large amounts
473 of computing power by leveraging a multitude of sites running diverse applications
474 concurrently in symphony.

475 The other major component, Web 2.0, is also a relatively new concept in the
476 history of Computer Science. The term Web 2.0 was originally coined in 1999 in a
477 futuristic prediction by Dracy DiNucci [56]: “The Web we know now, which loads
478 into a browser window in essentially static screenfulls, is only an embryo of the Web
479 to come. The first glimmerings of Web 2.0 are beginning to appear, and we are just
480 starting to see how that embryo might develop. The Web will be understood not
481 as screenfulls of text and graphics but as a transport mechanism, the ether through
482 which interactivity happens. It will [...] appear on your computer screen, [...] on your
483 TV set [...] your car dashboard [...] your cell phone [...] hand-held game machines
484 [...] maybe even your microwave oven.” Her vision began to form, as illustrated in
485 2004 by the O’Riley Web 2.0 conference, and since then the term has been a pivotal
486 buzz word among the internet. While many definitions have been provided, Web 2.0
487 really represents the transition from static HTML to harnessing the Internet and the
488 Web as a platform in of itself.

489 Web 2.0 provides multiple levels of application services to users across the Inter-
490 net. In essence, the web becomes an application suite for users. Data is outsourced
491 to wherever it is wanted, and the users have total control over what they interact

492 with, and spread accordingly. This requires extensive, dynamic and scalable host-
493 ing resources for these applications. This demand provides the user-base for much
494 of the commercial Cloud computing industry today. Web 2.0 software requires ab-
495 stracted resources to be allocated and relinquished on the fly, depending on the Web's
496 traffic and service usage at each site. Furthermore, Web 2.0 brought Web Services
497 standards [57] and the Service Oriented Architecture (SOA) [58] which outline the
498 interaction between users and cyber infrastructure. In summary, Web 2.0 defined
499 the interaction standards and user base, and Grid computing defined the underlying
500 infrastructure capabilities.

501 A Cloud computing implementation typically enables users to migrate their data
502 and computation to a remote location with some varying impact on system perfor-
503 mance [59]. This provides a number of benefits which could not otherwise be achieved:

- 504 ● *Scalable* - Clouds are designed to deliver as much computing power as any
505 user needs. While in practice the underlying infrastructure is not infinite, the
506 cloud resources are projected to ease the developer's dependence on any specific
507 hardware.
- 508 ● *Quality of Service (QoS)* - Unlike standard data centers and advanced com-
509 puting resources, a well-designed Cloud can project a much higher QoS than
510 traditionally possible. This is due to the lack of dependence on specific hard-
511 ware, so any physical machine failures can be mitigated without the prerequisite
512 user awareness.
- 513 ● *Specialized Environment* - Within a Cloud, the user can utilize customized tools
514 and services to meet their needs. This can be to utilize the latest library, toolkit,
515 or to support legacy code within new infrastructure.
- 516 ● *Cost Effective* - Users leverage only the resources required for their given task,

517 rather than putting forth a large capital investment in infrastructure. This re-
518 duces the risk for institutions potentially looking build a scalable system, thus
519 providing greater flexibility, since the user is only paying for needed infrastruc-
520 ture while maintaining the option to increase services as needed in the future.

- 521 • *Simplified Interface* - Whether using a specific application, a set of tools or
522 Web services, Clouds provide access to a potentially vast amount of computing
523 resources in an easy and user-centric way. We have investigated such an interface
524 within Grid systems through the use of the Cyberaid project [60, 61].

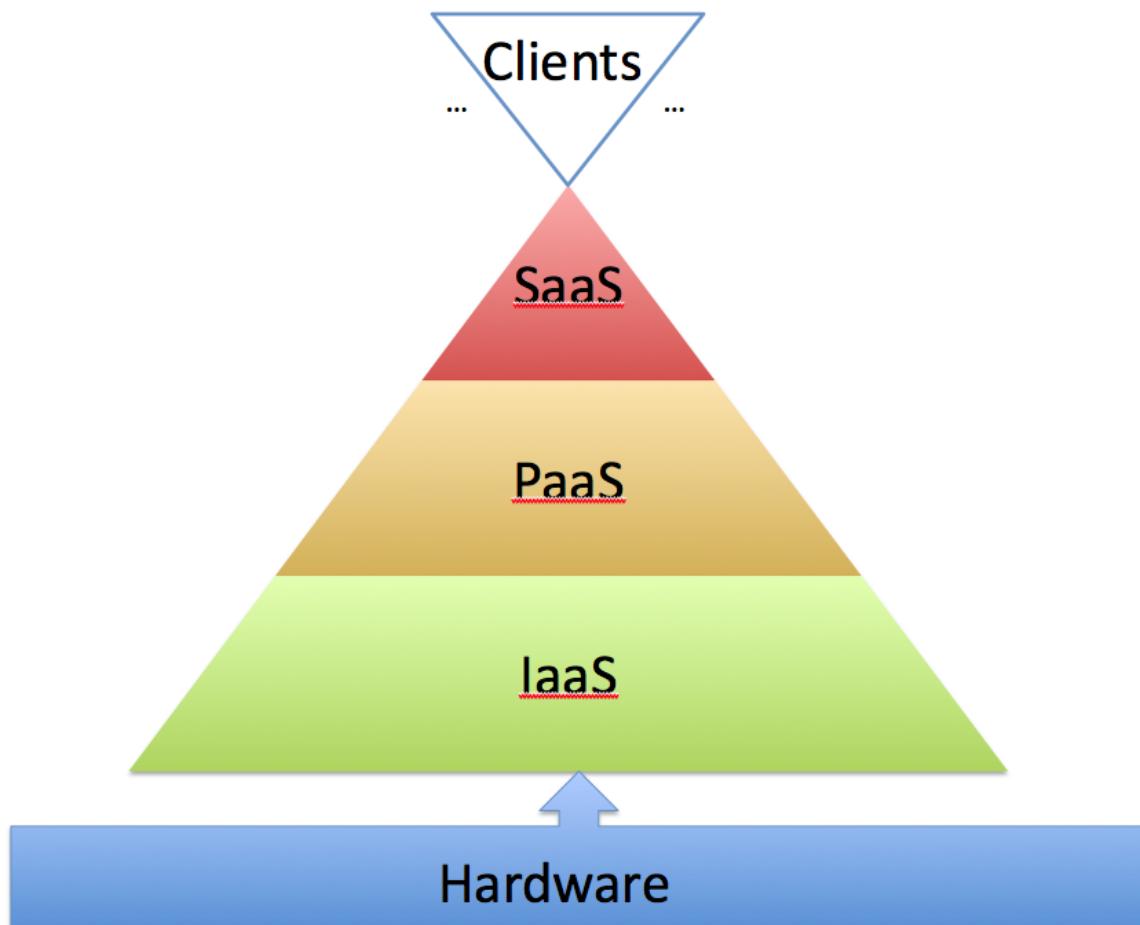


Figure 2.3 View of the Layers within a Cloud Infrastructure

525 Many of the features noted above define what Cloud computing can be from a

526 user perspective. However, Cloud computing in its physical form has many different
527 meanings and forms. Since Clouds are defined by the services they provide and not
528 by applications, an integrated as-a-service paradigm has been defined to illustrate the
529 various levels within a typical Cloud, as in Figure 2.3.

- 530 ● *Clients* - A client interacts with a Cloud through a predefined, thin layer of
531 abstraction. This layer is responsible for communicating the user requests and
532 displaying data returned in a way that is simple and intuitive for the user.
533 Examples include a Web Browser or a thin client application.
- 534 ● *Software-as-a-Service (SaaS)* - A framework for providing applications or soft-
535 ware deployed on the Internet packaged as a unique service for users to consume.
536 By doing so, the burden of running a local application directly on the client's
537 machine is removed. Instead all the application logic and data is managed
538 centrally and to the user through a browser or thin client. Examples include
539 Google Docs, Facebook, or Pandora.
- 540 ● *Platform-as-a-Service (PaaS)* - A framework for providing a unique computing
541 platform or software stack for applications and services to be developed on.
542 The goal of PaaS is to alleviate many of the burdens of developing complex,
543 scalable software by proving a programming paradigm and tools that make ser-
544 vice development and integration a tractable task for many. Examples include
545 Microsoft Azure and Google App Engine.
- 546 ● *Infrastructure-as-a-Service (IaaS)* - A framework for providing entire computing
547 resources through a service. This typically represents virtualized Operating
548 Systems, thereby masking the underlying complexity details of the physical
549 infrastructure. This allows users to rent or buy computing resources on demand
550 for their own use without needing to operate or manage physical infrastructure.

551 Examples include Amazon EC2, and OpenStack, and is the major cloud focal
552 point for this dissertation.

- 553 • *Physical Hardware* - The underlying set of physical machines and IT equipment
554 that host the various levels of service. These are typically managed at a large
555 scale using virtualization technologies which provide the QoS users expect. This
556 is the basis for all computing infrastructure.

557 When all of these layers are combined, a dynamic software stack is created to
558 focus on large scale deployment of services to users.

559 **2.2.1 Infrastructure-as-a-Service**

560 Today there are a number of Clouds that offer solutions for Infrastructure-as-a-Service
561 (IaaS). There have been multiple comparison efforts between various IaaS service
562 [4, 62–64] which provide insight to the similarities and differences between the long
563 array of cloud infrastructure deployment solutions. However at a high level, IaaS can
564 be split into 3 tiers, based on their availability.

- 565 • **Public** - Public IaaS is where the services and virtualizaton of hardware re-
566 sources are provided over the internet. Usually this is in a centralized data
567 centers whereby many users concurrently access these resources from across
568 the globe, often at a pre-negotiated price point and service level agreement
569 (SLA) [65]. Public clouds are the best utilization of economies of scale, by
570 selling hosting services en-masse and thereby offering competitive costs. The
571 Amazon Elastic Compute Cloud (EC2) is a primary example of a public cloud.

- 572 • **Private** - Private IaaS is where the cloud infrastructure is limited to within
573 a distinct group, set of users, business, or virtual organization. Usually such

574 private cloud infrastructure is also on a private, dedicated network that can
575 either be separate or connected to other services. Data within private clouds
576 are often more secure than those on a public cloud, and as such can be the choice
577 for many users who have sensitive data, or who large-scale users who find better
578 cost, performance, or QoS than what's provided within public clouds.

579 • **Hybrid** - Hybrid cloud IaaS combines the computational power of both private
580 and public IaaS, enabling users to keep data or costs within a private IaaS, but
581 then "burst" usage to a public cloud on peak computational demands. Virtual
582 Private Networks (VPN) can be useful to try to handle such a hybrid cloud not
583 only for management and network addressing but also for security.

584 **Amazon EC2**

585 The Amazon Elastic Compute Cloud (EC2) [66], is probably the most popular of
586 cloud infrastructure offerings to date, and is used extensively in the IT industry.
587 EC2 is the central component of Amazon Web Services platform. EC2 allows users
588 to effectively rent virtual machines (called instances), hosted within Amazon's data
589 centers, at a certain price point. Through their advanced UI or a RESTful API, users
590 can start, stop, pause, migrate, and destroy instances exactly as needed to match the
591 required computational tasks at hand.

592 Amazon EC2 predominantly relies on the Xen hypervisor to provide VMs on
593 demand to users, with an equivalent compute unit equal to a 1.7Ghz Intel Xeon
594 processor. However, recent advancements, instance types, and upgrades to EC2 have
595 increased this compute unit's power. Instance reservations have 3 types: On-demand,
596 Reserved, and Spot. With On-demand instances, users pay by the hour for however
597 long the desired instance is running. Users can instead rent reserved instances, where
598 they pay a one-time (discounted) cost based on a pre-determined allocation time.

599 There is also Spot pricing, where VMs are provisioned only when a given spot price is
600 met, which is determined simply based on supply and demand within the EC2 system
601 itself.

602 EC2 supports a wide array of user environments and setups. From an OS per-
603 spective, this includes running Linux, Unix, and even Windows VM instances. EC2
604 also provides instances with persistent storage through Elastic Block Storage (EBS)
605 and Simple Storage Service (S3) object storage mechanisms. These tools are necessary
606 for data persistence, as EC2 instances, as with most IaaS solutions, do not implicitly
607 persist data beyond the lifetime of the instance. EBS-rooted instances use an EBS
608 volume as a root disk device and as such, keep data beyond the lifetime of a given
609 instance. EC2 also offers elastic IPs, whereby public IP addresses are assigned to in-
610 stances at boot (or in-situ), however these elastic IPs do not require the DNS updates
611 to propagate or an administrator to adjust the network.

612 These advanced features, coupled with the first-to-market viability and continual
613 updates have made EC2 the largest cloud infrastructure today. While vendor lock-in
614 is a concern (EC2 is not available for download or replication) and other alternatives
615 exist such as Google's Compute Engine [67], the prevalence and support with EC will
616 likely mean its status quo as the public cloud of choice will continue for the foreseeable
617 future.

618 **Nimbus**

619 Nimbus [68, 69] is a set of open source tools that provide a private IaaS cloud com-
620 puting solution. Nimbus is based on the concept of virtual workspaces previously
621 introduced for Globus [69]. A virtual workspace is an abstraction of an execution
622 environment that can be made dynamically available to authorized clients by using
623 well-defined protocols. In this way, it can create customized environments by de-

624 ploying virtual machines (VMs) among remote resources. To such an end, Nimbus
625 provides a web interface called Nimbus Web. Its aim is to provide administrative and
626 user functions in a friendly interface.

627 Within Nimbus, a storage cloud implementation called Cumulus [68] has been
628 tightly integrated with the other central services, although it can also be used stan-
629 dalone. Cumulus is compatible with the Amazon Web Services S3 REST API [70],
630 but extends its capabilities by including features such as quota management. The
631 Nimbus cloud client uses the Jets3t library [71] to interact with Cumulus. However,
632 since it is compatible with S3 REST API, other interfaces like boto [72] or s2cmd [73]
633 can also be used to interact with Nimbus.

634 Nimbus supports two resource management strategies. The first one is the default
635 “resource pool” mode. In this mode, the service has direct control of a pool of
636 virtual machine managers (VMM) nodes and it assumes it can start VMs. The other
637 supported mode is called “pilot”. Here, the service makes requests, to a cluster’s
638 Local Resource Management System (LRMS), to get a VMM available where deploy
639 VMs.

640 Nimbus also provides an implementation of EC2’s interface that allows you to use
641 clients developed for the real EC2 system on Nimbus based clouds.

642 **Eucalyptus**

643 Eucalyptus is a product from Eucalyptus Systems [74–76], that developed out of
644 a research project at the University of California, Santa Barbara. Eucalyptus was
645 initially aimed at bringing the cloud computing paradigm of computing to academic
646 super computers and clusters. Eucalyptus provides a Amazon Web Services (AWS)
647 complaint EC2 based web service interface for interacting with the Cloud service.

648 The architecture depicted in Figure 2.4 is based on a two level hierarchy of the

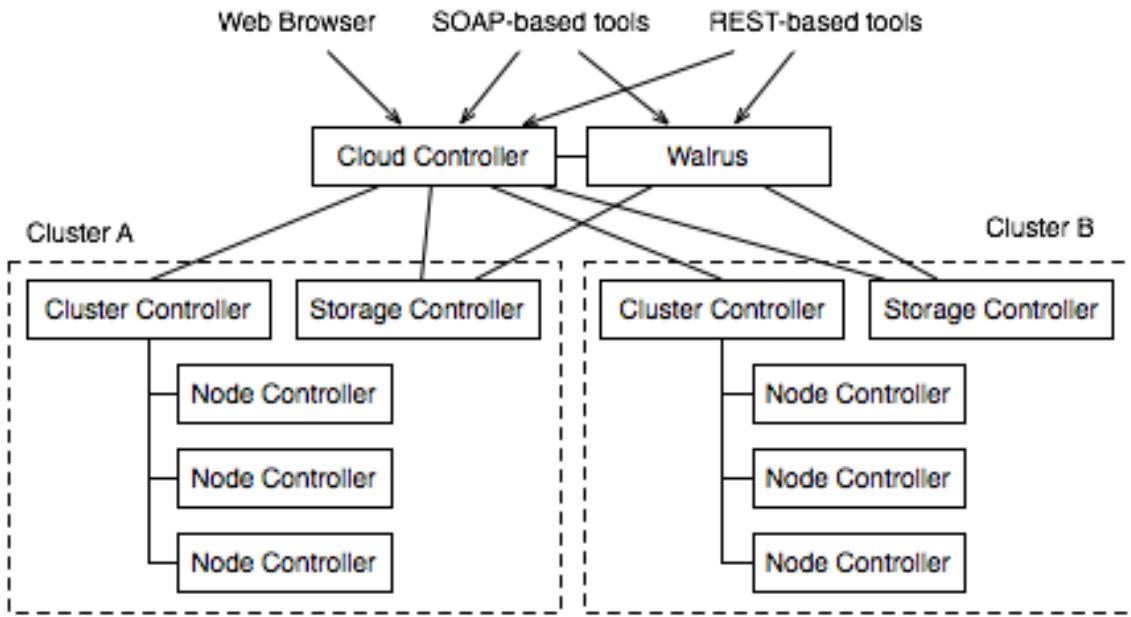


Figure 2.4 Eucalyptus Architecture

649 Cloud controller and the Cluster controller [77]. The Cluster Controller usually man-
 650 ages the nodes within a single cluster and multiple such Cluster Controllers can be
 651 used to connect to a single Cloud Controller. The Cloud Controller is responsible for
 652 the resource management, scheduling and accounting aspects of the Cloud.

653 Being one of the first private cloud computing solutions, Eucalyptus has a focused
 654 user interface. Much of Eucalyptus's design is based on the functionality of Amazon's
 655 EC2 cloud solution, and the user interface is a prime example of that model. While
 656 EC2 is a proprietary public cloud, it uses an open interface through the use of well
 657 designed Web Services which are open to all. Eucalyptus, looking to provide complete
 658 compatibility with EC2 to market the private cloud market, uses the same interface
 659 for all communication to the Cloud Controller. With Eucalyptus's interface being
 660 AWS complaint, it provides the same form of authentication that AWS supports,
 661 namely the shared key and PKI models.

662 While Eucalyptus can be controlled using the EC2 AMI tools, it also provides its

663 own specific tool set; euca2ools. Euca2ools provides support for creating and man-
664 aging keypairs, querying the cloud system, managing VMs, starting and terminating
665 instances, network configuration, and block storage usage. The Eucalyptus system
666 also provides a secure web front end to allow new users to create and manage account
667 information, view available VMs, and download their security credentials.

668 As seen with the user interface, Eucalyptus takes many design queues from Ama-
669 zons EC2 and the Image management system is no different. Eucalyptus stores images
670 in Walrus, the block storage system that is analogous to the Amazon S3 service. As
671 such, any user can bundle there own root filesystem, upload and then register this
672 image and link that image with a particular kernel and ramdisk image. This image
673 is uploaded into a user-defined bucket within Walrus, and can be retrieved anytime
674 from any availability zone. This allows users to create specialty virtual appliances
675 and deploy them within Eucalyptus with ease.

676 In 2014, Eucalyptus was acquired by Hewlett-Packard, which now maintains the
677 HPE Helion Eucalyptus Cloud to have full compatibility with Amazon EC2. The
678 most recent release of Helion eucalyptus is version 4.2.2 in early 2016.

679 OpenStack

680 OpenStack [78, 79], another private cloud infrastructure service, was introduced by
681 Rackspace and NASA in July 2010. The project aims to build an open source commu-
682 nity spanning technologists, developers, researchers, and industry to share resources
683 and technologies to create a massively scalable and secure cloud infrastructure. In
684 tradition with other open source projects the entire software is open source.

685 Historically, OpenStack focuses on the development of two aspects of cloud com-
686 putting to address compute and storage aspects with their OpenStack Compute and
687 OpenStack Storage solutions. According to the documentation “OpenStack Com-

688 pute is the internal fabric of the cloud creating and managing large groups of virtual
689 private servers” and “OpenStack Object Storage is software for creating redundant,
690 scalable object storage using clusters of commodity servers to store Terabytes or even
691 petabytes of data.” However, OpenStack as a platform has evolved much more than
692 its original efforts, and has created a wide array of new sub-projects.

693 As part of the computing support efforts OpenStack utilizes a cloud fabric con-
694 troller known under the name Nova. The architecture for Nova is built on the concepts
695 of shared-nothing and messaging-based information exchange. Hence most commu-
696 nication in Nova are facilitated by message queues. To prevent blocking components
697 while waiting for a response from others, deferred objects are introduced. Nova
698 supports multiple scheduling paradigms, and includes plugins for a wide array of hy-
699 pervisors, including Xen, KVM, and VMWare. The flexibility found within Nova
700 is useful for supporting a wide array of cloud IaaS computational efforts. Recently,
701 OpenStack has even looked to implement containers and bare-metal provisioning to
702 keep on pace with the latest technologies.

703 The OpenStack Swift storage solution is build around a number of interacting com-
704 ponents and concepts including a Proxy Server, a Ring, Object Server, a Container
705 Server, an Account Server, Replication, Updaters, and Auditors. This distributed
706 architecture attempts to have no centralized components, to enable scalability and
707 resiliency for data. Swift represents the long-term, object-based storage, similar to
708 Amazon S3, and attempts to maintain rough API compatibility with S3. As Swift
709 looks to use simple data replication as a main form of resiliency and fast read/write
710 is rarely a priority, Swift is often built using commodity disk drives instead of most
711 costly flash solutions.

712 With OpenStack Nova’s increased prevalence, the number of auxiliary OpenStack
713 projects has also increased to support Nova. While there are many other recent Open-

⁷¹⁴ Stack projects, these listed OpenStack efforts, along with Nova and Swift, represent
⁷¹⁵ the common core of a current OpenStack deployment.

- ⁷¹⁶ • **Cinder** for persistent block-level storage mechanisms to support VM instances
⁷¹⁷ and elastic block storage.
- ⁷¹⁸ • **Neutron** provides advanced networking and SDN solutions, IP addressing, and
⁷¹⁹ VLAN configuration.
- ⁷²⁰ • **Glance** delivers comprehensive image management, including image discovery,
⁷²¹ registration, and delivery mechanisms.
- ⁷²² • **Keystone** identity and authentication service for all OpenStack services.
- ⁷²³ • **Horizon**, a dashboard web-based UI framework, complimentary to the RESTful
⁷²⁴ client API.

⁷²⁵ Currently, OpenStack exists as one of the largest ongoing private IaaS efforts,
⁷²⁶ with over 500 companies contributing to the effort, and thousands of deployments.
⁷²⁷ While releases have pushed forth approximately every 6 months, the latest current
⁷²⁸ release at the time of writing is *Mitaka*, which now includes full support for GPUs
⁷²⁹ and SR-IOV interconnects, as detailed later in this dissertation. It is expected that
⁷³⁰ OpenStack's prevalence in the cloud computing community will only increase in the
⁷³¹ next few years.

⁷³² OpenNebula

⁷³³ OpenNebula [80, 81] is an open-source toolkit which allows to transform existing in-
⁷³⁴ frastructure into an Infrastructure as a Service (IaaS) cloud with cloud-like interfaces.
⁷³⁵ Figure 2.5 shows the OpenNebula architecture and their main components.

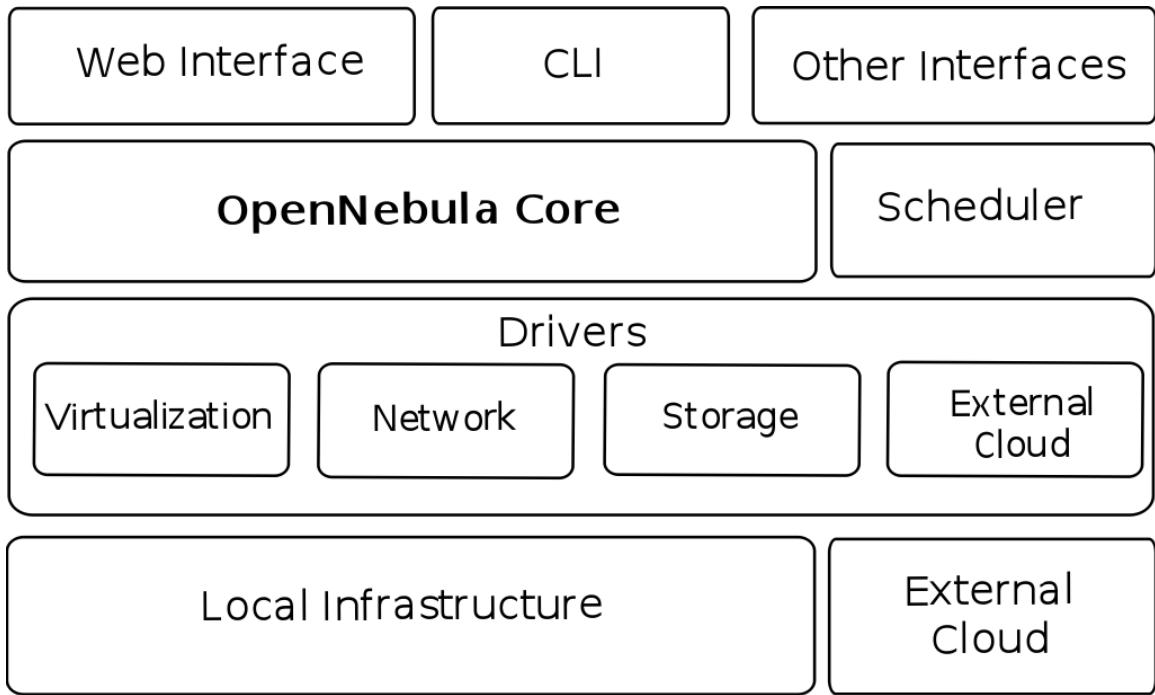


Figure 2.5 OpenNebula Architecture

736 The architecture of OpenNebula has been designed to be flexible and modular to
737 allow its integration with different storage and network infrastructure configurations,
738 and hypervisor technologies. Here, the core is a centralized component that manage
739 the virtual machine's (VM) full life cycle, including setting up networks dynamically
740 for groups of VMs and managing their storage requirements, such as VM disk image
741 deployment or on-the-fly software environment creation. Another important compo-
742 nent is the capacity manager, which governs the functionality provided by the core
743 for scheduling. The default capacity scheduler is a requirement/rank matchmaker.
744 However, it is also possible to develop more complex scheduling policies, through a
745 lease model and advance reservations like Haizea [82]. The last main components are
746 the access drivers. They provide an abstraction of the underlying infrastructure to
747 expose the basic functionality of the monitoring, storage and virtualization services
748 available in the cluster. Therefore, OpenNebula is not tied to any specific environ-

749 ment and can provide a uniform management layer regardless of the virtualization
750 platform.

751 Additionally, OpenNebula offers management interfaces to integrate the core's
752 functionality within other data center management tools, such as accounting or mon-
753 itoring frameworks. To this end, OpenNebula implements the libvirt API [83], an
754 open interface for VM management, as well as a command line interface (CLI). A
755 subset of this functionality is exposed to external users through a cloud interface.

756 Due to its architecture, OpenNebula is able to adapt to organizations with chang-
757 ing resource needs, including the addition or failure of physical resources [64]. Some
758 essential features to support changing environments are the live migration and the
759 snapshotting of VMs [80]. Furthermore, when the local resources are insufficient,
760 OpenNebula can support a hybrid cloud model by using cloud drivers to interface
761 with external clouds. This lets organizations supplement the local infrastructure
762 with computing capacity from a public cloud to meet peak demands, or implement
763 high availability strategies. OpenNebula includes an EC2 driver, which can submit
764 requests to Amazon EC2 and Eucalyptus [74], as well as an ElasticHosts driver [84].

765 Regarding the storage, an OpenNebula Image Repository allows users to easily
766 specify disk images from a catalog without worrying about low-level disk configuration
767 attributes or block device mapping. Also, image access control is applied to the
768 images registered in the repository, hence simplifying multi-user environments and
769 image sharing. Nevertheless, users can also set up their own images.

770 **Others**

771 Other cloud specific projects exist such as In-VIGO [85], Cluster-on-Demand [86],
772 and VMWare's own proprietary vCloud Air [87]. Each effort provides their own in-
773 terpretation of private cloud services within a data center, often with the ability to

774 interplay with public cloud offerings such as Amazon’s EC2. Docker [45] also looks
775 to provide IaaS capabilities with specialized and easily configurable containers, based
776 on LXC and libcontainer solutions. While it is still to be determined how Docker
777 and containers will change the private IaaS landscape, they do provide similar func-
778 tionality for Linux users without some of the complexities of traditional virtualized
779 IaaS.

780 **2.2.2 Virtual Clusters**

781 While virtualization and cloud IaaS provide many key advancements, this technology
782 alone is not sufficient. Rather, a collective scheduling and management for virtual
783 machines is required to piece together a working virtual cluster.

784 Let us consider a typical usage for a Cloud data center that is used in part to
785 provide computational power for the Large Hadron Collider at CERN [88], a global
786 collaboration from more than 2000 scientists of 182 institutes in 38 nations. Such a
787 system would have a small number of experiments to run. Each experiment would
788 require a very large number of jobs to complete the computation needed for the anal-
789 ysis. Examples of such experiments are the ATLAS [89] and CMS [90] projects, which
790 (combined) require Petaflops of computing power on a daily basis. Each job of an
791 experiment is unique, but the application runs are often the same. Therefore, virtual
792 machines are deployed to execute incoming jobs. There is a file server which provides
793 virtual machine templates. All typical jobs are preconfigured in virtual machine tem-
794 plates. When a job arrives at the head node of the cluster, a correspondent virtual
795 machine is dynamically started on a certain compute node within the cluster to ex-
796 ecute the job. While the LHC project and CERN’s cloud effort is a formidable one,
797 it only covers pleasingly parallel HTC workloads, and often times HPC and big data
798 workloads can equally complex yet drastically different.

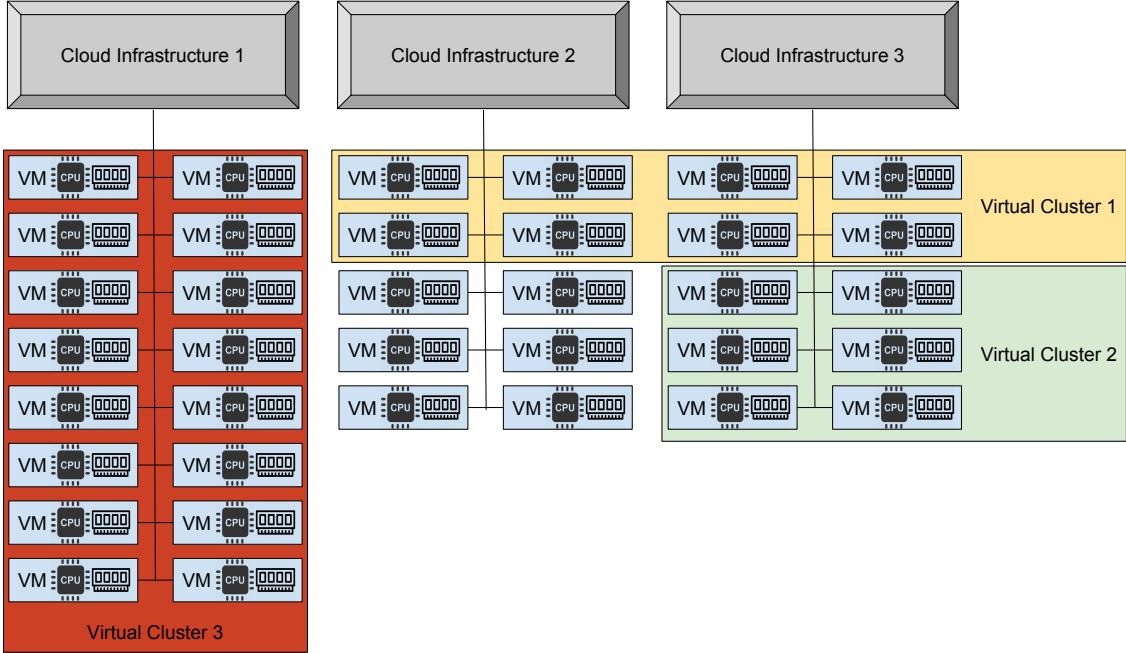


Figure 2.6 Virtual Clusters on Cloud Infrastructure

799 Cluster computing has become one of the core tools in distributed systems for
 800 use in parallel computation. Cluster computing revolves around the desire to get
 801 more computing power and better reliability by utilizing many computers together
 802 across a network to achieve larger computational tasks. Clusters have manifested
 803 themselves in many different ways, ranging from Beowulf clusters [6] which run using
 804 commodity PCs to some of the TOP500 [8] supercomputing systems today. Virtual
 805 clusters represents the growing need of users to effectively organize computational
 806 resources in an environment specific to their tasks at hand, instead of sharing a
 807 common architecture across many users. With the advent of modern virtualization,
 808 virtual clusters are deployed across a set of VMs in order to gain relative isolation
 809 and flexibility between disjoint virtual clusters. Virtual clusters, or a set of multiple
 810 cluster computing deployments on a single, larger physical cluster infrastructure, often
 811 have the following properties and attributes [91]:

- 812 ● Resources allocation based on a VM unit
 - 813 ● Clusters built of many VMs together, or by provisioning physical nodes
 - 814 ● Leverage local infrastructure management tools to provide a middleware solu-
815 tion for virtual clusters
 - 816 – Implementations could be a cloud IaaS such as OpenStack
 - 817 – Some instances use a queueing system such as PBS
 - 818 ● User experience based on virtual cluster management, not single VM manage-
819 ment
 - 820 ● Consolidates functionality on a smaller resource platform using multiple VMs
 - 821 ● Can provide fault tolerance through VM migration and management
 - 822 ● Can utilize dynamic scaling through the addition or deletion of VMs from the
823 virtual cluster
 - 824 ● Connection to back-end storage solution to provide virtual persistent storage
- 825 Given the properties, virtual clusters can take on many forms, however a very
826 simplified set of virtual clusters across cloud infrastructure are provided as a repre-
827 sentation in Figure 2.6. This could lead to the simple provisioning of multiple disjoint
828 OSs on a single physical resource. Virtual clusters generally have the ability to pro-
829 vide and manage their own user environment and tuned internal middleware. Virtual
830 clusters may enable the separation of multiple tasks into separate VMs, which still
831 in fact run on the same or similar underlying physical resources, effectively providing
832 task isolation. Virtual clusters can be deployed to be persistence, stored, shared, or
833 re-provisioned on demand. The size of a virtual cluster could potentially expand and

834 contrast relative to the necessary resource requirements, taking advantage of elastic-
835 ity found with virtualization. Furthermore, VM migration may enable fault tolerance
836 in the event of physical machine errors if properly managed.

837 With virtual clusters, the capability to quickly deploy custom environments be-
838 comes critical. As such, efforts have been put forth to quickly configure and create
839 VM images on-demand. This includes custom efforts with configuration engines such
840 as CfEngine, Chef, Ansible, and others. Within FutureGrid, an image management
841 system was defined to provide preconfigured VM images for cloud infrastructure using
842 the BCFG2 engine [92].

843 Initially, virtual clusters were proposed for the use of Grid communities [28].
844 Specifically, Foster et al look to provide commodity clusters to various Virtual Or-
845 ganizations [93], whereby grid services can instantiate and deploy VMs. This design
846 and implementation was further refined through the use of metadata and contextu-
847 alization using appliances [94]. Some of these ideas have even come to take shape in
848 larger scale supercomputing deployments, such as with SDSC Comet’s virtual cluster
849 availability [95].

850 Virtual clusters require orchestration services to be able to organize, deploy, man-
851 age, and re-play the desired user environment, and there have been a number of
852 efforts to bring this orchestration to utility. One efforts within FutureGrid is with
853 the experiment management design [39], which attempts to define how resources are
854 connected to and monitored, as well as how VMs are stored in a repository and pro-
855 visioned across multiple heterogeneous resources. This effort moved forward with
856 Cloudmesh [41], which provides a simple client interface to access multiple cloud
857 resources with a command-line shell interface.

858 Another virtual cluster orchestration effort has developed within the OpenStack
859 private IaaS solution itself, named OpenStack Heat [40]. Heat provides a method

860 by which you can deploy "stacks", which could essentially be virtual clusters, us-
861 ing OpenStack infrastructure. Specifically, Heat provides a human readable and
862 machine-accessible template for specifying environments and requirements, as well
863 as a RESTful API. In submitting a Heat orchestration template to the API, heat
864 will interpret and build the designed custom environment within a given OpenStack
865 cloud deployment. Kubernetes [96], a related effort, is a infrastructure orchestration
866 framework for managing containerized applications within Docker.

867 **2.2.3 The FutureGrid Project**

868 FutureGrid was a NSF-funded national-scale Grid and Cloud test-bed facility that
869 included a number of computational resources across many distributed locations.
870 This FutureGrid test-bed allowed users can evaluate differing systems for applica-
871 bility with their given research task or application. These areas include computer
872 science research topics ranging from authentication, authorization, scheduling, virtu-
873 alization, middleware design, interface design and cybersecurity, to the optimization
874 of grid-enabled and cloud-enabled computational schemes for Astronomy, Chemistry,
875 Biology, Engineering, High Energy Physics, or Atmospheric Science. This project
876 started at an opportune time, when cloud infrastructure was still in its experimental
877 stages and its applicability to mid-tier scientific efforts were unknown.

878 The FutureGrid features a unique WAN network structure that lent itself to a
879 multitude of experiments specifically for evaluating middleware technologies and ex-
880 periment management services. This network can be dedicated to conduct experi-
881 ments in isolation, using a network impairment device for introducing a variety of
882 predetermined network conditions. Figure 2.7 depicts the geographically distributed
883 resources that are outlined in Table 2.1 in more detail. All network links within Fu-
884 tureGrid are dedicated 10GbE links with the exception of a shared 10GbE link to

885 TACC over the TeraGrid [97,98] network, enabling high-speed data management and
 886 transfer between each partner site within FutureGrid.



Figure 2.7 FutureGrid Participants, Network, and Resources

887 Although the total number of systems within FutureGrid is comparatively conser-
 888 vative, they provide some heterogeneity to the architecture and are connected by the
 889 high-bandwidth network links. One important feature to note is that most systems
 890 can be dynamically provisioned, e.g. these systems can be reconfigured when needed
 891 by special software that is part of FutureGrid with proper access control by users
 892 and administrators. Therefore its believed that this hardware infrastructure can fully
 893 accommodate the needs of an experiment management system.

894 As of Fall 2014, the FutureGrid project has ended. The computing resources
 895 and facilities at Indiana University have continued on as FutureSystems, continuing
 896 to provide a cloud, big data, and HPC testbed to approved researchers. Recently
 897 with new projects as part of the digital Science Center, the FutureSystems effort has
 898 added two new machines, *Romeo* and *Juliet*, presenting clusters with Intel Haswell

Table 2.1 FutureGrid hardware

System type	Name	CPUs	Cores	TFLOPS	RAM(GB)	Disk(TB)	Site
IBM iDataPlex	India	256	1024	11	3072	†335	IU
Dell PowerEdge	Alamo	192	1152	12	1152	15	TACC
IBM iDataPlex	Hotel	168	672	7	2016	120	UC
IBM iDataPlex	Sierra	168	672	7	2688	72	UCSD
Cray XT5m	Xray	168	672	6	1344	†335	IU
ScaleMP vSMP	Echo	32	192	3	5872	192	IU
Dell PoweEdge	Bravo	32	128	2	3072	192	IU
SuperMicro	Delta	32	192	‡20	3072	128	IU
IBM iDataPlex	Foxtrot	64	256	2	768	5	UF
Total		1112	4960	70	23056	1394	

†Indicates shared file system. ‡Best current estimate

899 CPU architectures to be used for big data research.

900 2.3 High Performance Computing

901 2.3.1 Brief History of Supercomputing

902 Supercomputing itself can date back to some of the forefront of computing itself,
903 especially if we consider the ENIAC [99], the first Turing Complete general purpose
904 digital computer, also as the first supercomputer. ENIAC was first deployed to cal-
905 culate artillery firing tables, but was later used during the Second World War for
906 helping the Manhattan project's thermonuclear calculations and later dedicated to
907 the University of Pennsylvania after the war.

908 The first properly termed supercomputer was the Control Data Corporation's 6600

909 mainframe [100], first delivered to CERN in 1965. The CDC 6600 was notably faster
910 than the IBM counterparts, and the first deployments were able to perform on the
911 order of 1 MFLOP. Interestingly, the CPU design that came from Seymour Cray's
912 CDC 6600 took advantage of a simplified yet fast CPU design with silicon-based
913 transistors, which founded the basis of the RISC processor architecture.

914 Cray's efforts eventually lead him to start his own company, and in 1975 released
915 the Cray 1 system [101]. The Cray 1 took the powerful aspects of vector processing
916 and memory pipelining from the STAR architecture (developed later by CDC) and in-
917 troduced scalar performance through splitting vectors and instruction chaining. This
918 resulted in an overall performance of around 250 MFLOPS at peak, but realistically
919 closer to 100 MFLOPS for general applications. The Cray 1 system also helped push
920 forward the integrated circuit design, which was finally performant enough to be used.
921 Interestingly enough, the Cray 1 also required an entirely new freon based coolant
922 system.

923 The Cray 1 system gave way to the Cray X-MP and Y-MP in the mid 1980s.
924 These machines were shared-memory vector processors, with two processors in the
925 X-MP and up to 8 processors for the later Y-MP systems. These first shared-memory
926 systems were aided by increased memory speeds. The X-MP machine was capable
927 of 200 MFLOPS sustained and 400 MFLOPS peak performance, whereas the Y-MP
928 variants were capable of over 2 GFLOPS.

929 Also concurrently during the 1980s the advent of distributed memory architectures
930 for supercomputing were starting to emerge. Specifically work on Caltech's Cosmic
931 Cube, also known as a Hypercube, by Seitz and Fox [102,103], started the movement of
932 concurrent or parallel computing. The Cosmic Cube leveraged new VLSI techniques
933 and assembled Intel 8086/87 processors together with a novel hypercube interconnect
934 which required no switching, creating one of the first truly parallel computers. SIMD

935 programming and computation was done through a novel message passing architecture,
936 instead of shared variables. This design was first commercialized with Intel's
937 iPSC, and later contributing directly to designs in the Intel Paragon, ASCI Red, and
938 Cray T3D/E systems.

939 While concurrent and parallel processor supercomputers continued into the 90s
940 with aforementioned hypercube designs and the IBM Thinking Machines [104], a new
941 commodity-based strategy emerged with Becker and Sterling's Beowulf clusters [6].
942 Effectively, a Beowulf cluster is simply a cluster of commodity x86 machines linked
943 together with a simplified LAN network. Beowulf clusters often (but not always)
944 run Linux OS and leverage Ethernet solutions, and are programmed using a message
945 passing construct such as MPI [16]. This allows for the building of massively parallel
946 systems with relatively low cost and investment. Many specialized clusters today
947 still utilize commodity x86 hardware and Linux OSs similar to the original Beowulf
948 systems.

949 Concurrency and parallel computation on supercomputing resources has only
950 flourished since. This has been even more pronounced as CPU clock frequencies
951 stabilized and multi-core architectures took hold, driving the need for concurrency
952 not only at an increased rate for supercomputing, but also even for commodity sys-
953 tems. While commodity single-CPU, multi-core systems often look towards exploiting
954 shared memory parallelism, distributed memory architectures have become a way of
955 life for high performance computing.

956 2.3.2 Distributed Memory Computation

957 Abstractly, distributed memory architectures consist of multiple instances of a pro-
958 cessor, memory, and an interconnect which allows each instance to perform inde-
959 pendent computations and communicate over the interconnect. These interconnects

960 could be built using point-to-point, or through more complex and advanced switching
961 hardware, building a larger topology. While a simple example could involve several
962 commodity PCs connected through Ethernet switch, this model scales to the latest
963 supercomputing resources of today with millions of cores [105].

964 Programming such distributed memory systems is a nontrivial task that the
965 greater HPC community has been wrangling for years. In the early days, this took
966 the form of either the Message Passing Interface (MPI) [16] or PVM [106], however
967 MPI has been far more successful and dominates the market for distributed memory
968 parallel computation. MPI is a standardized message passing system, which defines
969 the semantics and syntax for writing parallel programs in C, C++, or Fortran. MPI
970 has even been implemented in other languages such as Java [107]. As MPI is a stan-
971 dardization, there are many implementations that exist, including OpenMPI [108],
972 MPICH [109], and MVAPICH [110], to name a few. Many MPI-enabled applications
973 have been shown to be, with proper and careful design, the most efficient way to run
974 a parallel application across a large subset of tightly coupled distributed resources,
975 and can often represent the status-quo for HPC applications today.

976 More recently, the MPI programming model has been modified or joined with
977 other models. This change or deviation largely revolves around the hardware that
978 has changed within HPC resources themselves to meet the need for more computa-
979 tional power. This includes hybrid MPI + OpenMP models which become useful as
980 multi-core and many-core technologies becomes increasingly available [111], or the
981 MPI+CUDA model which interleaves message passing with GPU utilization [112].
982 As some of the latest supercomputing resources have been deployed specifically with
983 Nvidia GPUs for GPGPU programming such as the ORNL's Titan system [113], the
984 need for distributed memory computation with GPUs has increased.

985 To get an idea of some of the hardware advances over the past few years, it is

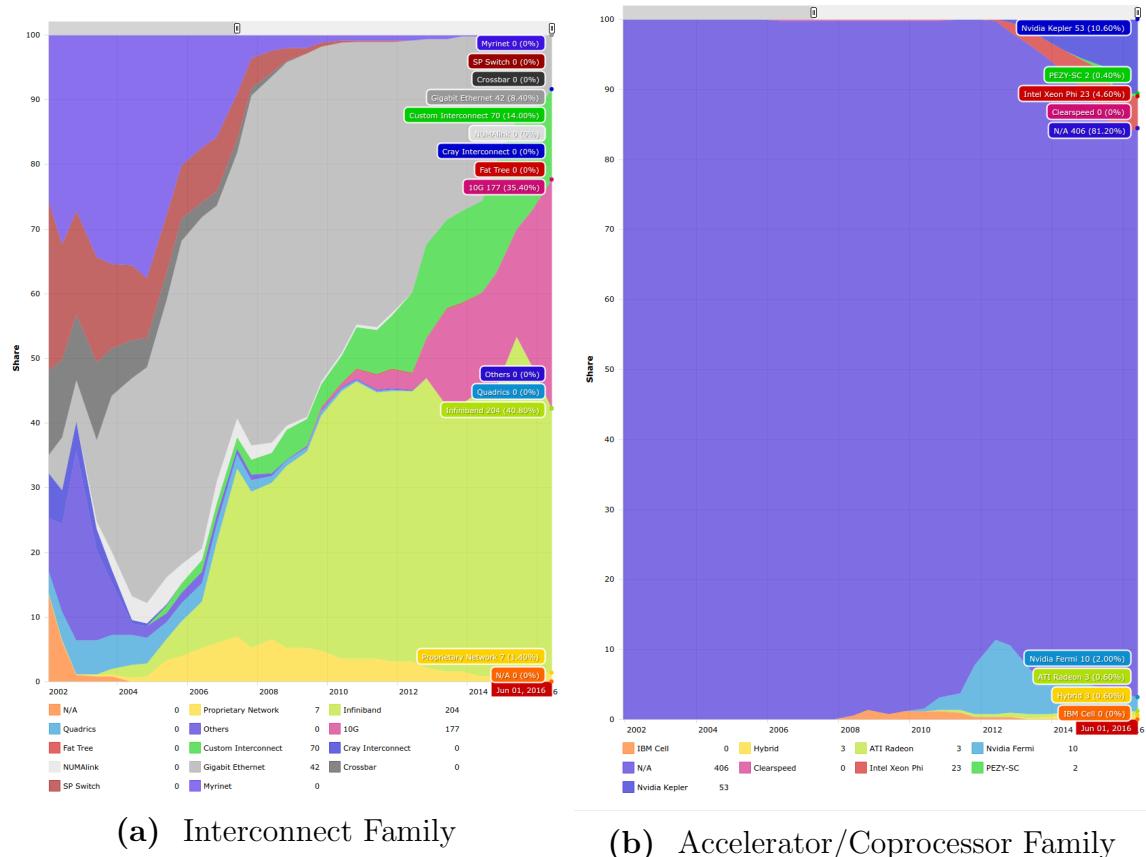


Figure 2.8 Top 500 Development over time from 2002 to 2016 [8]

useful to examine the Top 500 [8], a comprehensive list of the top 500 supercomputers known and their key characteristics. Looking at the past decade in HPC, we can see some trends emerge within hardware. Specifically looking at Interconnect and Coprocessor architectures in Figure 2.8, we see a notable jump in the number of deployed system that are using both InfiniBand and GPUs in the past decade. Specifically, InfiniBand usage has increased to roughly 40% of the total number of deployed systems and remained somewhat stable as an interconnect family. Concurrently, the use of accelerators has increased from only 1 in 500 systems a decade ago, to almost a 20% of the top 500 systems that are using coprocessors. Within that factor, the majority of such accelerator-equipped systems have been using GPUs, with a concurrent increase in the Intel Xeon Phi coprocessor as well. While these do not represent all of supercomputing nor exclusively the most high end of systems, they do represent advanced hardware that has been increasingly common in the last decade, yet relatively underutilized in comparison within cloud infrastructure. As such, much of the effort in this dissertation focuses on, but is not limited to, these two technology families.

2.3.3 Exascale

As the forefront of supercomputing moves beyond the latest petaflop machines in the past few years [113], the HPC community is setting their sights on the next significant milestone: Exascale. Exascale computing refers broadly to performing roughly one exaFLOPS, or 10^{18} floating point operations per second. However, exascale itself is far more than just a theoretical FLOPS goal, but instead a set of new computing advancements and challenges that requires reaching computational power at such magnitude. While FLOPs are often used as the ubiquitous yard stick for supercomputing with the LINPACK benchmark [114], other efforts have taken hold to classify systems under a different set of parameters [115, 116], with the loose understanding

1011 that these may incorporate a richer application set destined for exascale systems. This
1012 could include for instance integer calculations at a similar scale to satisfy defence and
1013 intelligence perspectives, or graph processing with billions of vertices.

1014 With exascale, there are a number of barriers that exist with current technologies
1015 that must be overcome to reach exascale. The exascale Computing Study [35]
1016 specifically lists 4 major focal areas:

1017 1. Energy and Power Challenge

1018 • Describes the physical difficulties in providing the amount of power needed
1019 to drive a sufficiently large exascale system. The US DOE estimates the
1020 maximum power envelope for a deployed first exascale system to be within
1021 20-40MW. Extrapolating current technology power utilization shows an
1022 order of magnitude more energy utilization than the specificity power en-
1023 envelop. As such, new architectures and conversation techniques will need
1024 to be investigated.

1025 2. Memory and Storage Challenge

1026 • This challenge illustrates the problem that has grown in relation to the
1027 memory wall, defined by the exponential difference between processor and
1028 memory performance, as well as the storage capacity limits to support
1029 calculations at the level of performance necessary. This challenge incorpo-
1030 rates not only main memory limitations, but also tertiary storage issues as
1031 well.

1032 3. Concurrency and Storage Challenge

1033 • This challenge is born from the recent limit in CPU clock rates as a way
1034 to gain performance. Instead, performance must be gained through paral-

1035 lelism. The depth of this challenge is especially profound when we consider
1036 parallelism on the order of millions, if not billions of threads.

1037 4. Resiliency Challenge

1038 • The resiliency challenge defines the necessity of computation to recover
1039 and continue in the event of a fault or fluctuation. As parallelism and
1040 the number of individualized components substantially increases in a path
1041 towards exascale, the mean time to failure of any given component also
1042 increases.

1043 While current exascale efforts are as wide as they are varying, not only with
1044 concepts, architectures, and runtime systems, but also with deployment plans and
1045 expectations between future deployments. Of particular interest in current exas-
1046 cale research is in Operating System and runtime (OS/R) developments to support
1047 new extreme-scale applications in an efficient manner. Two examples of novel OS
1048 approaches are the Hobbes project [117] and the ARGO Exascale Operating Sys-
1049 tem [118]. These OS efforts, along with novel programming models for exascale such
1050 as ParalleX [119] look to fundamentally change the relationship between HPC hard-
1051 ware architectures and the libraries and applications to be leveraged on such future
1052 exascale deployments.

1053 It is possible that virtualization itself may have an impact in OS and runtime ser-
1054 vices in exascale [117]. While some of the work herein may be tangentially of utility
1055 to such efforts, the immediate goal of this dissertation is not to investigate the appli-
1056 cability of virtualization for exascale systems, but rather to enable the diversification
1057 of HPC towards cloud infrastructure.

1058 **Chapter 3**

1059 **Analysis of Virtualization**

1060 **Technologies for High Performance**

1061 **Computing Environments**

1062 **3.1 Abstract**

1063 As Cloud computing emerges as a dominant paradigm in distributed systems, it is
1064 important to fully understand the underlying technologies that make Clouds possible.
1065 One technology, and perhaps the most important, is virtualization. Recently virtual-
1066 ization, through the use of hypervisors, has become widely used and well understood
1067 by many. However, there are a large spread of different hypervisors, each with their
1068 own advantages and disadvantages. This chapter provides an in-depth analysis of
1069 some of today's commonly accepted virtualization technologies from feature com-
1070 parison to performance analysis, focusing on the applicability to High Performance
1071 Computing environments using FutureGrid resources. The results indicate virtualiza-
1072 tion sometimes introduces slight performance impacts depending on the hypervisor

1073 type, however the benefits of such technologies are profound and not all virtualization
1074 technologies are equal.

1075 **3.2 Introduction**

1076 Cloud computing [53] is one of the most explosively expanding technologies in the
1077 computing industry today. A Cloud computing implementation typically enables
1078 users to migrate their data and computation to a remote location with some varying
1079 impact on system performance [59]. This provides a number of benefits which could
1080 not otherwise be achieved.

1081 Such benefits include:

- 1082 • *Scalability* - Clouds are designed to deliver as much computing power as any
1083 user needs. While in practice the underlying infrastructure is not infinite, the
1084 cloud resources are projected to ease the developer's dependence on any specific
1085 hardware.
- 1086 • *Quality of Service (QoS)* - Unlike standard data centers and advanced com-
1087 puting resources, a well-designed Cloud can project a much higher QoS than
1088 traditionally possible. This is due to the lack of dependence on specific hard-
1089 ware, so any physical machine failures can be mitigated without the prerequisite
1090 user awareness.
- 1091 • *Customization* - Within a Cloud, the user can utilize customized tools and
1092 services to meet their needs. This can be to utilize the latest library, toolkit, or
1093 to support legacy code within new infrastructure.
- 1094 • *Cost Effectiveness* - Users finds only the hardware required for each project.
1095 This reduces the risk for institutions potentially want build a scalable system,

1096 thus providing greater flexibility, since the user is only paying for needed in-
1097 frastructure while maintaining the option to increase services as needed in the
1098 future.

- 1099 • *Simplified Access Interfaces* - Whether using a specific application, a set of
1100 tools or Web services, Clouds provide access to a potentially vast amount of
1101 computing resources in an easy and user-centric way.

1102 While Cloud computing has been driven from the start predominantly by the in-
1103 dustry through Amazon [66], Google [120] and Microsoft [121], a shift is also occurring
1104 within the academic setting as well. Due to the many benefits, Cloud computing is
1105 becoming immersed in the area of High Performance Computing (HPC), specifically
1106 with the deployment of scientific clouds [122] and virtualized clusters [28].

1107 There are a number of underlying technologies, services, and infrastructure-level
1108 configurations that make Cloud computing possible. One of the most important
1109 technologies is virtualization. Virtualization, in its simplest form, is a mechanism to
1110 abstract the hardware and system resources from a given Operating System. This is
1111 typically performed within a Cloud environment across a large set of servers using a
1112 Hypervisor or Virtual Machine Monitor (VMM), which lies in between the hardware
1113 and the OS. From the hypervisor, one or more virtualized OSs can be started concur-
1114 rently, leading to one of the key advantages of Cloud computing. This, along with the
1115 advent of multi-core processors, allows for a consolidation of resources within any data
1116 center. From the hypervisor level, Cloud computing middleware is deployed atop the
1117 virtualization technologies to exploit this capability to its maximum potential while
1118 still maintaining a given QoS and utility to users.

1119 The rest of this chapter is as follows: First, we look at what virtualization is,
1120 and what current technologies currently exist within the mainstream market. Next

1121 we discuss previous work related to virtualization and take an in-depth look at the
1122 features provided by each hypervisor. We follow this by outlining an experimental
1123 setup to evaluate a set of today's hypervisors on a novel Cloud test-bed architecture.
1124 Then, we look at performance benchmarks which help explain the utility of each
1125 hypervisor and the feasibility within an HPC environment. We conclude with our
1126 final thoughts and recommendations for using virtualization in Clouds for HPC.

1127 **3.3 Related Research**

1128 While the use of virtualization technologies has increased dramatically in the past few
1129 years, virtualization is not specific to the recent advent of Cloud computing. IBM
1130 originally pioneered the concept of virtualization in the 1960's with the M44/44X
1131 systems [123]. It has only recently been reintroduced for general use on x86 plat-
1132 forms. Today there are a number of public Clouds that offer IaaS through the use
1133 of virtualization technologies. The Amazon Elastic Compute Cloud (EC2) [124] is
1134 probably the most popular Cloud and is used extensively in the IT industry to this
1135 day. Nimbus [125] and Eucalyptus [74] are popular private IaaS platforms in both
1136 the scientific and industrial communities. Nimbus, originating from the concept of
1137 deploying virtual workspaces on top of existing Grid infrastructure using Globus, has
1138 pioneered scientific Clouds since its inception. Eucalyptus has historically focused
1139 on providing an exact EC2 environment as a private cloud to enable users to build
1140 an EC2-like cloud using their own internal resources. Other scientific Cloud specific
1141 projects exist such as OpenNebula [126], In-VIGO [127], and Cluster-on-Demand [86],
1142 all of which leverage one or more hypervisors to provide computing infrastructure on
1143 demand. In recent history, OpenStack [128] has also come to light from a joint col-
1144 laboration between NASA and Rackspace which also provide compute and storage

1145 resources in the form of a Cloud.

1146 While there are currently a number of virtualization technologies available today,
1147 the virtualization technique of choice for most open platforms over the past 5 years has
1148 typically been the Xen hypervisor [48]. However more recently VMWare ESX [129]
1149 ¹, Oracle VirtualBox [130] and the Kernel-based Virtual Machine (KVM) [52] are
1150 becoming more commonplace. As these look to be the most popular and feature-
1151 rich of all virtualization technologies, we look to evaluate all four to the fullest extent
1152 possible. There are however, numerous other virtualization technologies also available,
1153 including Microsoft's Hyper-V [131], Parallels Virtuozzo [132], QEMU [133], OpenVZ
1154 [134], Oracle VM [135], and many others. However, these virtualization technologies
1155 have yet to seen widespread deployment within the HPC community, at least in their
1156 current form, so they have been placed outside the scope of this work.

1157 In recent history there have actually been a number of comparisons related to
1158 virtualization technologies and Clouds. The first performance analysis of various hy-
1159 pervisors started with, unsurprisingly, the hypervisor vendors themselves. VMWare
1160 has happy to put out its own take on performance in [136], as well as the original
1161 Xen article [48] which compares Xen, XenoLinux, and VMWare across a number of
1162 SPEC and normalized benchmarks, resulting in a conflict between both works. From
1163 here, a number of more unbiased reports originated, concentrating on server consol-
1164 idation and web application performance [129, 137, 138] with fruitful yet sometimes
1165 incompatible results. A feature base survey on virtualization technologies [139] also
1166 illustrates the wide variety of hypervisors that currently exist. Furthermore, there
1167 has been some investigation into the performance within HPC, specifically with In-
1168 finiBand performance of Xen [140] and rather recently with a detailed look at the
1169 feasibility of the Amazon Elastic Compute cloud for HPC applications [44], however

¹Due to the restrictions in VMWare's licensing agreement, benchmark results are unavailable.

1170 both works concentrate only on a single deployment rather than a true comparison
1171 of technologies.

1172 As these underlying hypervisor and virtualization implementations have evolved
1173 rapidly in recent years along with virtualization support directly on standard x86
1174 hardware, it is necessary to carefully and accurately evaluate the performance impli-
1175 cations of each system. Hence, we conducted an investigation of several virtualization
1176 technologies, namely Xen, KVM, VirtualBox, and in part VMWare. Each hypervisor
1177 is compared alongside one another with base-metal as a control and (with the exception
1178 of VMWare) run through a number of High Performance benchmarking tools.

1179 3.4 Feature Comparison

1180 With the wide array of potential choices of virtualization technologies available, its
1181 often difficult for potential users to identify which platform is best suited for their
1182 needs. In order to simplify this task, we provide a detailed comparison chart between
1183 Xen 3.1, KVM from RHEL5, VirtualBox 3.2 and VMWWare ESX in Figure 2.

	Xen	KVM	VirtualBox	VMWare
Para-virtualization	Yes	No	No	No
Full virtualization	Yes	Yes	Yes	Yes
Host CPU	x86, x86-64, IA-64	x86, x86-64,IA64,PPC	x86, x86-64	x86, x86-64
Guest CPU	x86, x86-64, IA-64	x86, x86-64,IA64,PPC	x86, x86-64	x86, x86-64
Host OS	Linux, UNIX	Linux	Windows, Linux, UNIX	Proprietary UNIX
Guest OS	Linux, Windows, UNIX	Linux, Windows, UNIX	Linux, Windows, UNIX	Linux, Windows, UNIX
VT-x / AMD-v	Opt	Req	Opt	Opt
Cores supported	128	16	32	8
Memory supported	4TB	4TB	16GB	64GB
3D Acceleration	Xen-GL	VMGL	Open-GL	Open-GL, DirectX
Live Migration	Yes	Yes	Yes	Yes
License	GPL	GPL	GPL/proprietary	Proprietary

Figure 3.1 A comparison chart between Xen, KVM, VirtualBox, and VMWare ESX

1184 The first point of investigation is the virtualization method of each VM. Each

1185 hypervisor supports full virtualization, which is now common practice within most
1186 x86 virtualization deployments today. Xen, originating as a para-virtualized VMM,
1187 still supports both types, however full virtualization is often preferred as it does
1188 not require the manipulation of the guest kernel in any way. From the Host and
1189 Guest CPU lists, we see that x86 and, more specifically, x86-64/amd64 guests are all
1190 universally supported. Xen and KVM both support Itanium-64 architectures for full
1191 virtualization (due to both hypervisors dependency on QEMU), and KVM also claims
1192 support for some recent PowerPC architectures. However, we concern ourselves only
1193 with x86-64 features and performance, as other architectures are out of the scope of
1194 this manuscript. Of the x86-64 platforms, KVM is the only hypervisor to require
1195 either Intel VT-X or AMD-V instruction sets in order to operate. VirtualBox and
1196 VMWare have internal mechanisms to provide full virtualization even without the
1197 virtualization instruction sets, and Xen can default back to para-virtualized guests.

1198 Next, we consider the host environments for each system. As Linux is the pri-
1199 mary OS type of choice within HPC deployments, its key that all hypervisors sup-
1200 port Linux as a guest OS, and also as a host OS. As VMWare ESX is meant to be
1201 a virtualization-only platform, it is built upon a specially configured Linux/UNIX
1202 proprietary OS specific to its needs. All other hypervisors support Linux as a host
1203 OS, with VirtualBox also supporting Windows, as it was traditionally targeted for
1204 desktop-based virtualization. However, as each hypervisor uses VT-X or AMD-V in-
1205 structions, each can support any modern OS targeted for x86 platforms, including all
1206 variants of Linux, Windows, and UNIX.

1207 While most hypervisors have desirable host and guest OS support, hardware sup-
1208 port within a guest environment varies drastically. Within the HPC environment,
1209 virtual CPU (vCPU) and maximum VM memory are critical aspects to choosing the
1210 right virtualization technology. In this case, Xen is the first choice as it supports up

1211 to 128 vCPUs and can address 4TB of main memory in 64-bit modes, more than
1212 any other. VirtualBox, on the other hand, supports only 32 vCPUs and 16GB of
1213 addressable RAM per guest OS, which may lead to problems when looking to deploy
1214 it on large multicore systems. KVM also faces an issue with the number of vCPU
1215 supported limited to 16, recent reports indicate it is only a soft limit [141], so deploy-
1216 ing KVM in an SMP environment may not be a significant hurdle. Furthermore, all
1217 hypervisors provide some 3D acceleration support (at least for OpenGL) and support
1218 live migration across homogeneous nodes, each with varying levels of success.

1219 Another vital juxtaposition of these virtualization technologies is the license agree-
1220 ments for its applicability within HPC deployments. Xen, KVM, and VirtualBox are
1221 provided for free under the GNU Public License (GPL) version 2, so they are open
1222 to use and modification by anyone within the community, a key feature for many
1223 potential users. While VirtualBox is under GPL, it has recently also offered with
1224 additional features under a more proprietary license dictated by Oracle since its ac-
1225 quirement from Sun last year. VMWare, on the other hand, is completely proprietary
1226 with an extremely limited licensing scheme that even prevents the authors from will-
1227 fully publishing any performance benchmark data without specific and prior approval.
1228 As such, we have neglected VMWare form the remainder of this chapter. Whether
1229 going with a proprietary or open source hypervisor, support can be acquired (usually
1230 for an additional cost) with ease from each option.

1231 **3.4.1 Usability**

1232 While side by side feature comparison may provide crucial information about a poten-
1233 tial user's choice of hypervisor, that may also be interested in its ease of installation
1234 and use. We will take a look at each hypervisor from two user perspectives, a systems
1235 administrator and normal VM user.

1236 One of the first things on any system administrator's mind on choosing a hypervi-
1237 sor is the installation. For all of these hypervisors, installation is relatively painless.
1238 For the FutureGrid support group, KVM and VirualBox are the easiest of the all
1239 tested hypervisors to install, as there are a number of supported packages available
1240 and installation only requires the addition of one or more kernel modules and the sup-
1241 port software. Xen, while still supported in binary form by many Linux distributions,
1242 is actually much more complicated. This is because Xen requires a full modification
1243 to the kernel itself, not just a module. Loading a new kernel into the boot process
1244 which may complicate patching and updating later in the system's maintenance cycle.
1245 VMWare ESX, on the other hand, is entirely separate from most other installations.
1246 As previously noted, ESX is actually a hypervisor and custom UNIX host OS com-
1247 bined, so installation of ESX is likewise to installing any other OS from scratch. This
1248 may be either desirable or adverse, depending on the system administrator's usage of
1249 the systems and VMWare's ability to provide a secure and patched environment.

1250 While system administrators may be concerned with installation and maintenance,
1251 VM users and Cloud developers are more concerned with daily usage. The first thing
1252 to note about all of such virtualiation technologies is they are supported (to some
1253 extent) by the libvirt API [142]. Libvirt is commonly used by many of today's IaaS
1254 Cloud offerings, including Nimbus, Eucalyptus, OpenNebula and OpenStack. As
1255 such, the choice of hypervisor for Cloud developer's is less of an issue, so long as
1256 the hypervisor supports the features they desire. For individual command line usage
1257 of each tool, it varies quite a bit more. Xen does provide their own set of tools for
1258 controlling and monitoring guests, and seem to work relatively well but do incur a
1259 slight learning curve. KVM also provides its own CLI interface, and while it is often
1260 considered less cumbersome it provides less advanced features directly to users, such as
1261 power management or quick memory adjustment (however this is subject to personal

opinion). One advantage of KVM is each guest actually runs as a separate process within the host OS, making it easy for a user to manage and control the VM inside the host if KVM misbehaves. VirtualBox, on the other hand, provides the best command line and graphical user interface. The CLI, is especially well featured when compared to Xen and KVM as it provides clear, decisive and well documented commands, something most HPC users and system administrators alike will appreciate. VMWare provides a significantly enhanced GUI as well as a Web-based ActiveX client interface that allows users to easily operate the VMWare host remotely. In summary, there is a wide variance of interfaces provided by each hypervisor, however we recommend Cloud developers to utilize the libvirt API whenever possible.

3.5 Experimental Design

In order to provide an unaltered and unbiased review of these virtualization technologies for Clouds, we need to outline a neutral testing environment. To make this possible, we have chosen to use FutureGrid as our virtualization and cloud test-bed.

3.5.1 The FutureGrid Project

FutureGrid (FG) [143] provides computing capabilities that enable researchers to tackle complex research challenges related to the use and security of Grids and Clouds. These include topics ranging from authentication, authorization, scheduling, virtualization, middleware design, interface design and cybersecurity, to the optimization of Grid-enabled and cloud-enabled computational schemes for researchers in astronomy, chemistry, biology, engineering, atmospheric science and epidemiology.

The test-bed includes a geographically distributed set of heterogeneous computing systems, a data management system that will hold both metadata and a growing

library of software images necessary for Cloud computing, and a dedicated network allowing isolated, secure experiments, as seen in Figure 3.2. The test-bed supports virtual machine-based environments, as well as operating systems on native hardware for experiments aimed at minimizing overhead and maximizing performance. The project partners are integrating existing open-source software packages to create an easy-to-use software environment that supports the instantiation, execution and recording of grid and cloud computing experiments.



Figure 3.2 FutureGrid Participants and Resources

One of the goals of the project is to understand the behavior and utility of Cloud computing approaches. However, it is not clear at this time which of these toolkits will become the users' choice toolkit. FG provides the ability to compare these frameworks with each other while considering real scientific applications [39]. Hence, researchers are be able to measure the overhead of cloud technology by requesting linked experiments on both virtual and bare-metal systems, providing valuable information that help decide which infrastructure suits their needs and also helps users that want to

1299 transition from one environment to the other. These interests and research objectives
1300 make the FutureGrid project the perfect match for this work. Furthermore, we expect
1301 that the results gleaned from this chapter will have a direct impact on the FutureGrid
1302 deployment itself.

1303 3.5.2 Experimental Environment

1304 Currently, one of FutureGrid's latest resources is the *India* system, a 256 CPU IBM
1305 iDataPlex machine consisting of 1024 cores, 2048 GB of ram, and 335 TB of storage
1306 within the Indiana University Data Center. In specific, each compute node of India
1307 has two Intel Xeon 5570 quad core CPUs running at 2.93Ghz, 24GBs of Ram, and a
1308 QDR InfiniBand connection. A total of four nodes were allocated directly from India
1309 for these experiments. All were loaded with a fresh installation of Red Hat Enterprise
1310 Linux server 5.5 x86_64 with the 2.6.18-194.8.1.el5 kernel patched. Three of the four
1311 nodes were installed with different hypervisors; Xen version 3.1, KVM (build 83), and
1312 VirtualBox 3.2.10, and the forth node was left as-is to act as a control for bare-metal
1313 native performance.

1314 Each guest virtual machine was also built using Red Hat EL server 5.5 running
1315 an unmodified kernel using full virtualization techniques. All tests were conducted
1316 giving the guest VM 8 cores and 16GB of ram to properly span a compute node.
1317 Each benchmark was run a total of 20 times, with the results averaged to produce
1318 consistent results, unless indicated otherwise.

1319 3.5.3 Benchmarking Setup

1320 As this chapter aims to objectively evaluate each virtualization technology from a
1321 side-by-side comparison as well as from a performance standpoint, the selection of

1322 benchmarking applications is critical.

1323 The performance comparison of each virtual machine is based on two well known
1324 industry standard performance benchmark suites; HPCC and SPEC. These two
1325 benchmark environments are recognized for their standardized reproducible results in
1326 the HPC communit, and the National Science Foundation (NSF), Department of En-
1327 ergy (DOE), and DARPA are all sponsors of the HPCC benchmarks. The following
1328 benchmarks provide a means to stress and compare processor, memory, inter-process
1329 communication, network, and overall performance and throughput of a system. These
1330 benchmarks were selected due to their importance to the HPC community sinse they
1331 are often directly correlated with overall application performance [144].

1332 **HPCC Benchmarks**

1333 The HPCC Benchmarks [145, 146] are an industry standard for performing bench-
1334 marks for HPC systems. The benchmarks are aimed at testing the system on multiple
1335 levels to test their performance. It consists of 7 different tests:

- 1336 ● *HPL* - The Linpack TPP benchmark measures the floating point rate of exe-
1337 cution for solving a linear system of equations. This benchmark is perhaps the
1338 most important benchmark within HPC today, as it is the basis of evaluation
1339 for the Top 500 list [8].
- 1340 ● *DGEMM* - Measures the floating point rate of execution of double precision real
1341 matrix-matrix multiplication.
- 1342 ● *STREAM* - A simple synthetic benchmark program that measures sustainable
1343 memory bandwidth (in GB/s) and the corresponding computation rate for sim-
1344 ple vector kernel.

- 1345 ● *PTRANS* - Parallel matrix transpose exercises the communications where pairs
1346 of processors communicate with each other simultaneously. It is a useful test of
1347 the total communications capacity of the network.

- 1348 ● *RandomAccess* - Measures the rate of integer random updates of memory (GUPS).

- 1349 ● *FFT* - Measures the floating point rate of execution of double precision complex
1350 one-dimensional Discrete Fourier Transform (DFT).

- 1351 ● *Communication bandwidth and latency* - A set of tests to measure latency and
1352 bandwidth of a number of simultaneous communication patterns; based on b_eff
1353 (effective bandwidth benchmark).

1354 This benchmark suite uses each test to stress test the performance on multiple
1355 aspects of the system. It also provides reproducible results which can be verified by
1356 other vendors. This benchmark is used to create the Top 500 list [8] which is the list
1357 of the current top supercomputers in the world. The results that are obtained from
1358 these benchmarks provide an unbiased performance analysis of the hypervisors. Our
1359 results provide insight on inter-node PingPong bandwidth, PingPong latency, and
1360 FFT calculation performance.

1361 **SPEC Benchmarks**

1362 The Standard Performance Evaluation Corporation (SPEC) [147, 148] is the other
1363 major standard for evaluation of benchmarking systems. SPEC has several different
1364 testing components that can be utilized to benchmark a system. For our benchmark-
1365 ing comparison we will use the SPEC OMP2001 because it appears to represent a
1366 vast array of new and emerging parallel applications while simultaneously providing
1367 a comparison to other SPEC benchmarks. SPEC OMP continues the SPEC tradi-

1368 tion of giving HPC users the most objective and representative benchmark suite for
1369 measuring the performance of SMP (shared memory multi-processor) systems.

- 1370 • The benchmarks are adapted from SPEC CPU2000 and contributions to its
1371 search program.
- 1372 • The focus is to deliver systems performance to real scientific and engineering
1373 applications.
- 1374 • The size and runtime reflect the needs of engineers and researchers to model
1375 large complex tasks.
- 1376 • Two levels of workload characterize the performance of medium and large sized
1377 systems.
- 1378 • Tools based on the SPEC CPU2000 toolset make these the easiest ever HPC
1379 tests to run.
- 1380 • These benchmarks place heavy demands on systems and memory.

1381 **3.6 Performance Comparison**

1382 The goal of this chapter is to effectively compare and contrast the various virtual-
1383 ization technologies, specifically for supporting HPC-based Clouds. The first set of
1384 results represent the performance of HPCC benchmarks. Each benchmark was run
1385 a total of 20 times, and the mean values taken with error bars represented using the
1386 standard deviation over the 20 runs. The benchmarking suite was built using the Intel
1387 11.1 compiler, uses the Intel MPI and MKL runtime libraries, all set with defaults
1388 and no optimizations whatsoever.

1389 We open first with High Performance Linpack (HPL), the de-facto standard for
1390 comparing resources. In Figure 3.3, we can see the comparison of Xen, KVM, and
1391 Virtual Box compared to native bare-metal performance. First, we see that native
1392 is capable of around 73.5 Gflops which, with no optimizations, achieves 75% of the
1393 theoretical peak performance. Xen, KVM and VirtualBox perform at 49.1, 51.8 and
1394 51.3 Gflops, respectively when averaged over 20 runs. However Xen, unlike KVM
1395 and VirtualBox, has a high degree of variance between runs. This is an interesting
1396 phenomenon for two reasons. First, this may impact performance metrics for other
1397 HPC applications and cause errors and delays between even pleasingly-parallel appli-
1398 cations and add to reducer function delays. Second, this wide variance breaks a key
1399 component of Cloud computing providing a specific and predefined quality of service.
1400 If performance can sway as widely as what occurred for Linpack, then this may have
1401 a negative impact on users.

1402 Next, we turn to another key benchmark within the HPC community, Fast Fourier
1403 Transforms (FFT). Unlike the synthetic Linpack benchmark, FFT is a specific, pur-
1404 poseful benchmark which provides results which are often regarded as more relative
1405 to a user's real-world application than HPL. From Figure 3.4, we can see rather dis-
1406 tinct results from what was previously provided by HPL. Looking at Star and Single
1407 FFT, its clear performance across all hypervisors is roughly equal to bare-metal per-
1408 formance, a good indication that HPC applications may be well suited for use on
1409 VMs. The results for MPI FFT also show similar results, with the exception of Xen,
1410 which has a decreased performance and high variance as seen in the HPL benchmark.
1411 Our current hypothesis is that there is an adverse affect of using Intel's MPI runtime
1412 on Xen, however the investigation is still ongoing.

1413 Another useful benchmark illustrative of real-world performance between bare-
1414 metal performance and various hypervisors are the ping-pong benchmarks. These

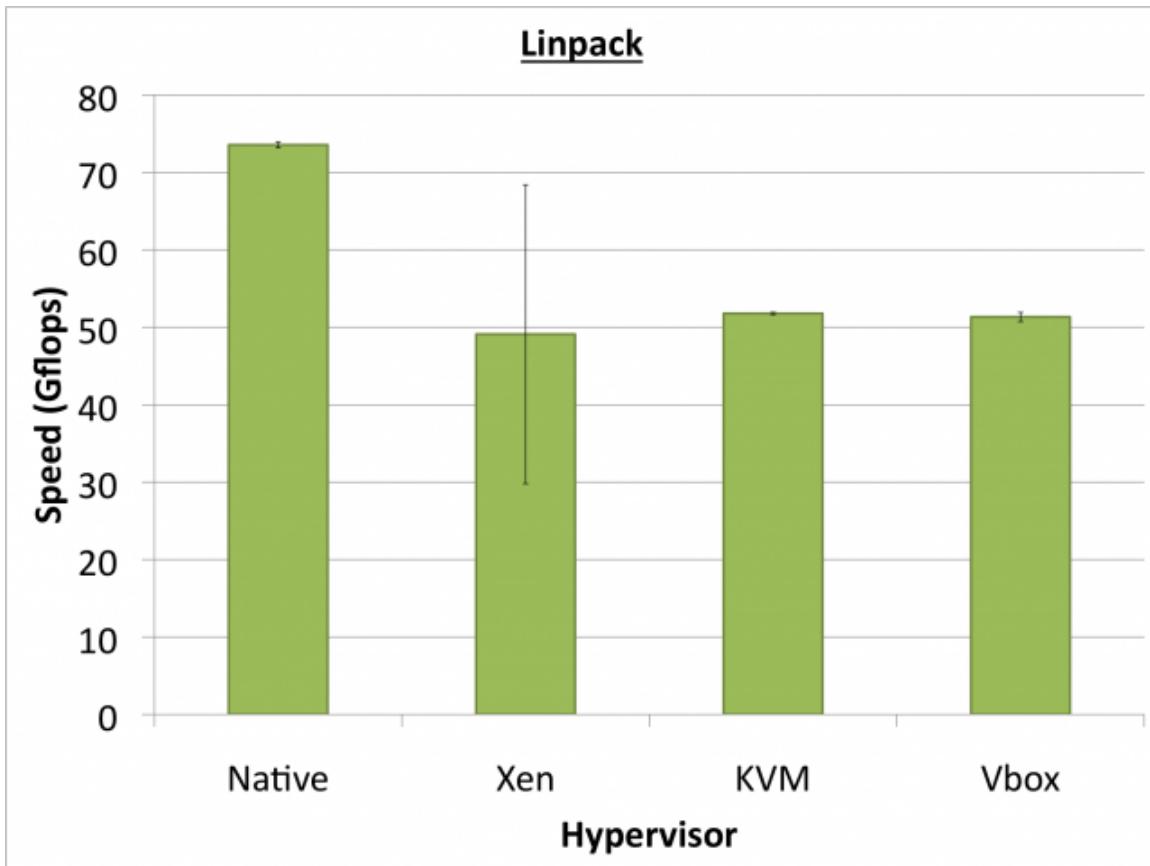


Figure 3.3 Linpack performance

1415 benchmarks measure the bandwidth and latency of passing packets between multiple
 1416 CPUs. With this experiment, all ping-pong latencies are kept within a given node,
 1417 rather than over the network. This is done to provide further insight into the CPU and
 1418 memory overhead within each hypervisor. From Figure 3.5 the intranode bandwidth
 1419 performance is uncovered, with some interesting distinctions between each hypervi-
 1420 sor. First, Xen performs, on average, close to native speeds, which is promising for
 1421 the hypervisor. KVM, on the other hand, shows consistent overhead proportional
 1422 to native performance across minimum, average, and maximum bandwidth. Virtu-
 1423 alBox, on the other hand, performs well, in fact too well to the point that raises
 1424 alarm. While the minimum and average bandwidths are within native performance,
 1425 the maximum bandwidth reported by VirtualBox is significantly greater than native

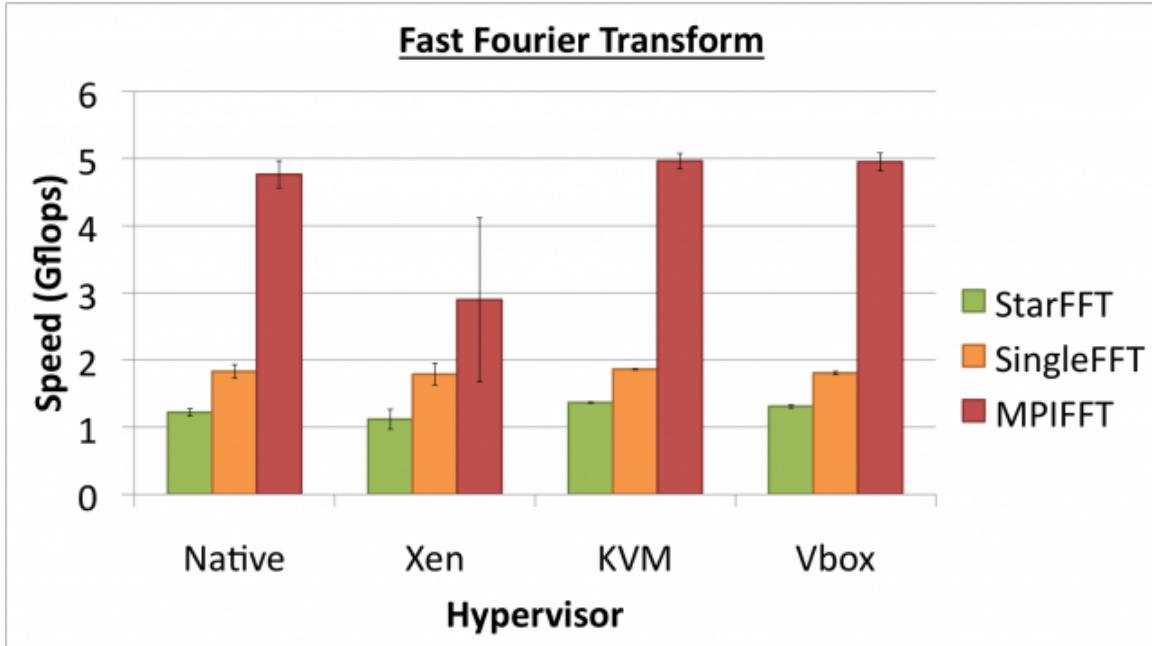


Figure 3.4 Fast Fourier Transform performance

1426 measurements, with a large variance. After careful examination, it appears this is
 1427 due to how VirtualBox assigns its virtual CPUs. Instead of locking a virtual CPU to
 1428 a real CPU, a switch may occur which could benefit on the off-chance the two CPU's
 1429 in communication between a ping-pong test could in fact be the same physical CPU.
 1430 The result would mean the ping-pong packet would remain in cache and result in a
 1431 higher perceived bandwidth than normal. While this effect may be beneficial for this
 1432 benchmark, it may only be an illusion towards the real performance gleaned from the
 1433 VirtualBox hypervisor.

1434 The Bandwidth may in fact be important within the ping-pong benchmark, but
 1435 the latency between each ping-pong is equally useful in understanding the perfor-
 1436 mance impact of each virtualization technology. From Figure 3.6, we see KVM and
 1437 VirtualBox have near-native performance; another promising result towards the util-
 1438 ity of hypervisors within HPC systems. Xen, on the other hand, has extremely high
 1439 latencies, especially at for maximum latencies, which in turn create a high variance

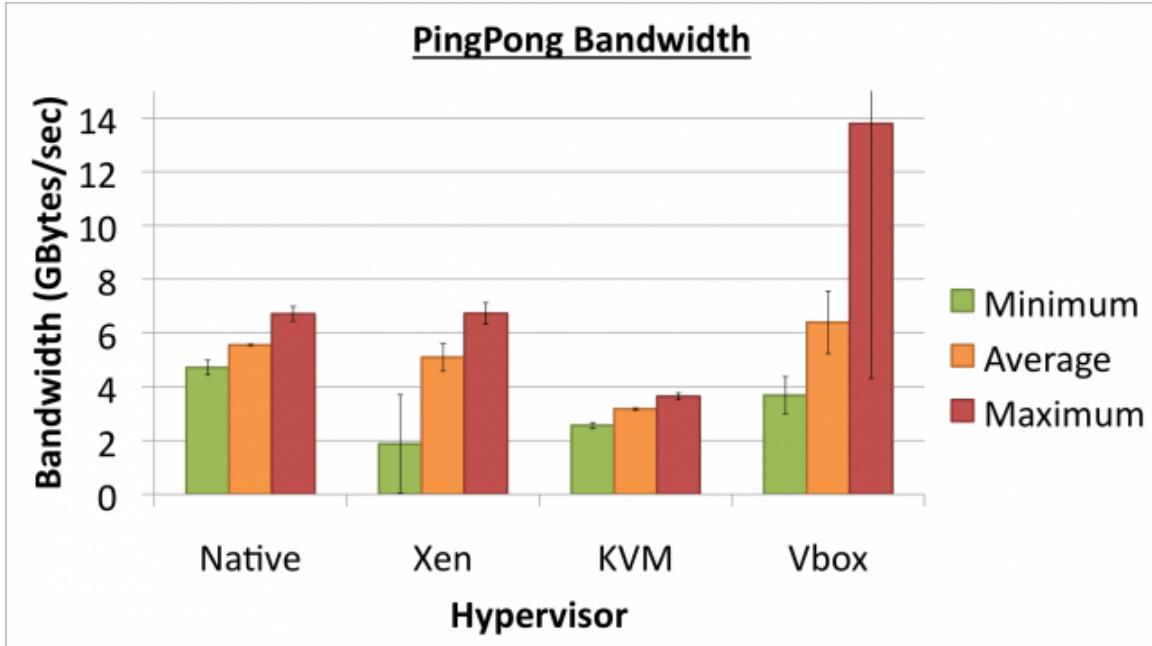


Figure 3.5 Ping Pong bandwidth performance

¹⁴⁴⁰ within the average latency within the VM's performance.

¹⁴⁴¹ While the HPCC benchmarks provide a comprehensive view for many HPC applications including Linpack and FFT using MPI, performance of intra-node SMP applications using OpenMP is also investigated. Figure 3.7 illustrates SPEC OpenMP performance across the VMs we concentrate on, as well as baseline native performance. ¹⁴⁴⁴ First, we see that the combined performance over all 11 applications executed 20 times yields the native testbed with the best performance at a SPEC score of 34465. ¹⁴⁴⁶ KVM performance comes close with a score of 34384, which is so similar to the native performance that most users will never notice the difference. Xen and VirtualBox both perform notably slower with scores of 31824 and 31695, respectively, however ¹⁴⁴⁹ this is only an 8% performance drop compared to native speeds. Further results can ¹⁴⁵¹ be found on the SPEC website [149].

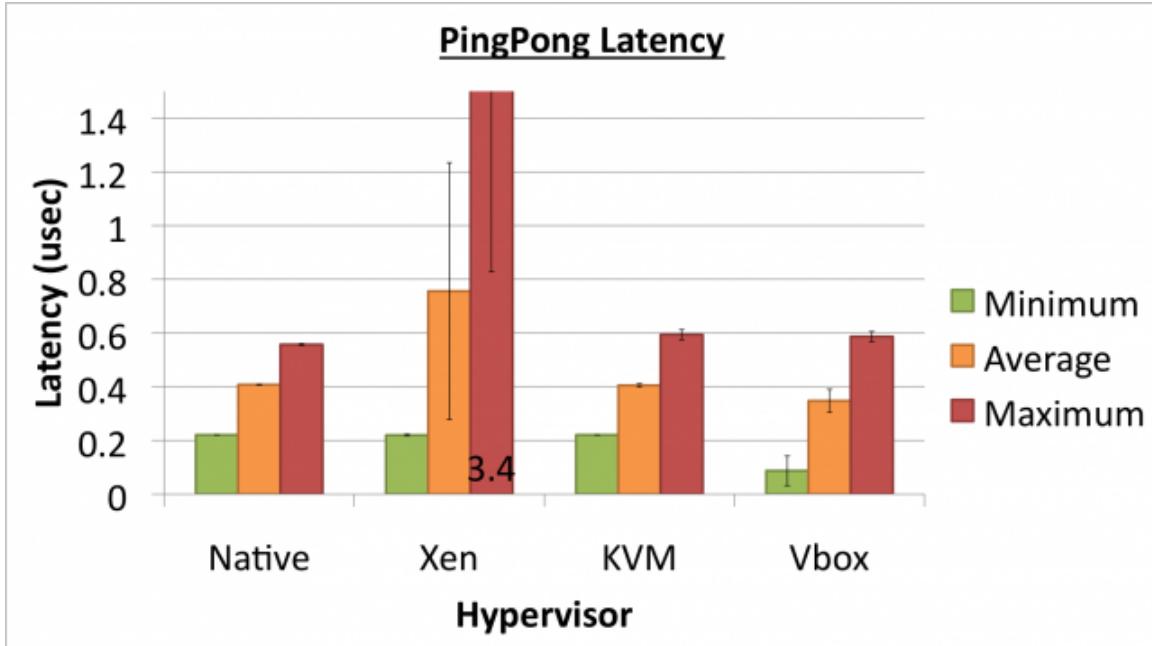


Figure 3.6 Ping Pong latency performance (lower is better)

1452 3.7 Discussion

1453 The primary goal of this chapter is to evaluate the viability of virtualization within
 1454 HPC. After our analysis, the answer seems to be a resounding "yes." However, we
 1455 also hope to select the best virtualization technology for such an HPC environment.

1456 In order to do this, we combine the feature comparison along with the performance
 1457 results, and evaluate the potential impact within the FutureGrid testbed.

1458 From a feature standpoint, most of today's virtualization technologies fit the bill
 1459 for at least small scale deployment, including VMWare. In short, each support Linux
 1460 x86_64 platforms, use VT-X technology for full virtualization, and support live mi-
 1461 gration. Due to VMWare's limited and costly licensing, it is immediately out of
 1462 contention for most HPC deployments. From a CPU and memory standpoint, Xen
 1463 seems to provide the best expandability, supporting up to 128 cpus and 4TB of ad-
 1464 dressable RAM. So long as KVM's vCPU limit can be extended, it too shows promise

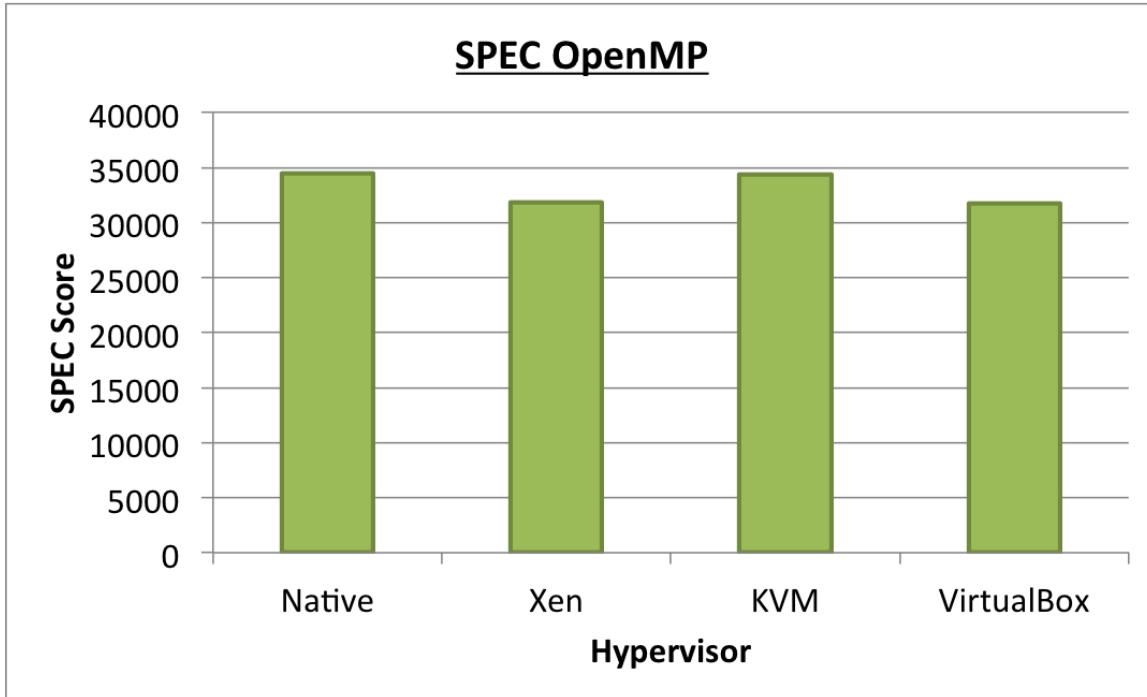


Figure 3.7 Spec OpenMP performance

as a feature-full virtualization technology. One of Virtualbox's greatest limitations was the 16GB maximum memory allotment for individual guest VMs, which actually limited us from giving VMs more memory for our performance benchmarks. If this can be fixed and Oracle does not move the product into the proprietary market, VirtualBox may also stand a chance for deployment in HPC environments.

From the benchmark results previously described, the use of hypervisors within HPC-based Cloud deployments is mixed batch. Figure 3.8 summarizes the results based on a 1-3 rating, 1 being best and 3 being worst. While Linpack performance seems to take a significant performance impact across all hypervisors, the more practical FFT benchmarks seem to show little impact, a notably good sign for virtualization as a whole. The ping-pong bandwidth and latency benchmarks also seem to support this theory, with the exception of Xen, who's performance continually has wide fluctuations throughout the majority of the benchmarks. OpenMP performance

	Xen	KVM	VirtualBox
Linpack	3	1	2
FFT	3	1	2
Bandwidth	2	3	1
Latency	3	2	1
OpenMP	2	1	3
Total Rating	13	8	9

Figure 3.8 Benchmark rating summary (lower is better)

1478 through the SPEC OMP benchmarking suite also shows promising results for the use
 1479 of hypervisors in general, with KVM taking a clear lead by almost matching native
 1480 speeds.

1481 While Xen is typically regarded as the most widely used hypervisor, especially
 1482 within academic clouds and grids, its performance has shown lack considerably when
 1483 compared to either KVM or VirtualBox. In particular, Xen's wide and unexplained
 1484 fluctuations in performance throughout the series of benchmarks suggests that Xen
 1485 may not be the best choice for building a lasting quality of service infrastructure upon.

1486 From Figure 3.8, KVM rates the best across all performance benchmarks, making it
 1487 the optimal choice for *general* deployment in an HPC environment. Furthermore,
 1488 this work's illustration of the variance in performance among each benchmark and
 1489 the applicability of each benchmark towards new applications may make possible the
 1490 ability to preemptively classify applications for accurate prediction towards the ideal
 1491 virtualized Cloud environment. We hope to further investigate this concept through
 1492 the use of the FutureGrid experiment management framework at a later date.

1493 In summary, it is the authors' projection that KVM is the best overall choice for

1494 use within HPC Cloud environments. KVM's feature-rich experience and near-native
1495 performance makes it a natural fit for deployment in an environment where usability
1496 and performance are paramount. Within the FutureGrid project specifically, we hope
1497 to deploy the KVM hypervisor across our Cloud platforms in the near future, as it
1498 offers clear benefits over the current Xen deployment. Furthermore, we expect these
1499 findings to be of great importance to other public and private Cloud deployments, as
1500 system utilization, Quality of Service, operating cost, and computational efficiency
1501 could all be improved through the careful evaluation of underlying virtualization
1502 technologies.

1503 **Chapter 4**

1504 **Evaluating GPU Passthrough in**
1505 **Xen for High Performance Cloud**
1506 **Computing**

1507 **4.1 Abstract**

1508 With the advent of virtualization and Infrastructure-as-a-Service (IaaS), the broader
1509 scientific computing community is considering the use of clouds for their technical
1510 computing needs. This is due to the relative scalability, ease of use, advanced user
1511 environment customization abilities clouds provide, as well as many novel comput-
1512 ing paradigms available for data-intensive applications. However, there is concern
1513 about a performance gap that exists between the performance of IaaS when com-
1514 pared to typical high performance computing (HPC) resources, which could limit the
1515 applicability of IaaS for many potential scientific users.

1516 Most recently, general-purpose graphics processing units (GPGPUs or GPUs) have
1517 become commonplace within high performance computing. We look to bridge the

1518 gap between supercomputing and clouds by providing GPU-enabled virtual machines
1519 (VMs) and investigating their feasibility for advanced scientific computation. Specif-
1520 ically, the Xen hypervisor is utilized to leverage specialized hardware-assisted I/O
1521 virtualization and PCI passthrough in order to provide advanced HPC-centric Nvidia
1522 GPUs directly in guest VMs. This methodology is evaluated by measuring the per-
1523 formance of two Nvidia Tesla GPUs within Xen VMs and comparing to bare-metal
1524 hardware. Results show PCI passthrough of GPUs within virtual machines is a vi-
1525 able use case for many scientific computing workflows, and could help support high
1526 performance cloud infrastructure in the near future.

1527 4.2 Introduction

1528 Cloud computing [4] has established itself as a prominent paradigm within the realm
1529 of Distributed Systems [150] in a very short period of time. Clouds are an internet-
1530 based solution that provide computational and data models for utilizing resources,
1531 which can be accessed directly by users on demand in a uniquely scalable way. Cloud
1532 computing functions by providing a layer of abstraction on top of base hardware
1533 to enable a new set of features that are otherwise intangible or intractable. These
1534 benefits and features include Scalability, a guaranteed Quality of Service (QoS), cost
1535 effectiveness, and direct user customization via a simplified user interface [59].

1536 While the origin of cloud computing is based in industry through solutions such
1537 as Amazon's EC2 [151], Google's MapReduce [152], and Microsoft's Azure [153], the
1538 paradigm has since become integrated in all areas of science and technology. Most
1539 notably, there is an increasing effort within the High Performance Computing (HPC)
1540 community to leverage the utility of clouds for advanced scientific computing to solve
1541 a number of challenges still standing in the field. This can be clearly seen in large-

scale efforts such as the FutureGrid project [154], the Magellan project [43], and through various other Infrastructure-as-a-Service projects including OpenStack [155], Nimbus [69], and Eucalyptus [156].

Within HPC, there has also been a notable movement toward dedicated accelerator cards such as general purpose graphical processing units (GPGPUs, or GPUs) to enhance scientific computation problems by upwards of two orders of magnitude. This is accomplished through dedicated programming environments, compilers, and libraries such as CUDA [157] from Nvidia as well as the OpenCL effort [158]. When combining GPUs in an otherwise typical HPC environment or supercomputer, major gains in performance and computational ability have been reported in numerous fields [159, 160], ranging from Astrophysics to Bioinformatics. Furthermore, these gains in computational power have also reportedly come at an increased performance-per-watt [161], a metric that is increasingly important to the HPC community as we move closer to exascale computing [162] where power consumption is quickly becoming the primary constraint.

With the advent of both clouds and GPUs within the field of scientific computing, there is an immediate and ever-growing need to provide heterogeneous resources, most immediately GPUs, within a cloud environment in the same scalable, on-demand, and user-centric way that many cloud users are already accustomed to [163]. While this task alone is nontrivial, it is further complicated by the high demand for performance within HPC. As such, it is performance that is paramount to the success of deploying GPUs within cloud environments, and thus is the central focus of this work.

The rest of this chapter is organized as follows. First, in Section 2, we discuss the related research and the options currently available for providing GPUs within a virtualized cloud environment. In Section 3, we discuss the methodology for providing GPUs directly within virtual machines. In Section 4 we outline the evaluation of the

1568 given methodology using two different Nvidia Tesla GPUs and compare to the best-
1569 case native application in Section 5. Then, we discuss the implications of these results
1570 in Section 6 and consider the applicability of each method within a production cloud
1571 system. Finally, we conclude with our findings and suggest directions for future work.

1572 4.3 Virtual GPU Directions

1573 Recently, GPU programming has been a primary focus for numerous scientific com-
1574 puting applications. Significant progress has been accomplished in many different
1575 workloads, both in science and engineering, based on parallel abilities of GPUs for
1576 floating point operations and very high on-GPU memory bandwidth. This hardware,
1577 coupled with CUDA and OpenCL programming frameworks, has led to an explosion
1578 of new GPU-specific applications. In some cases, GPUs outperform even the fastest
1579 multicore counterparts by an order of magnitude [164]. In addition, further research
1580 could leverage the per-node performance of GPU accelerators with the high speed,
1581 low latency interconnects commonly utilized in supercomputers and clusters to create
1582 a hybrid GPU + MPI class of applications. The number of distributed GPU appli-
1583 cations is increasing substantially in supercomputing, usually scaling many GPUs
1584 simultaneously [165].

1585 Since the establishment of cloud computing in industry, research groups have
1586 been evaluating its applicability to science [3]. Historically, HPC and Grids have
1587 been on similar but distinct paths within distributed systems, and have concentrated
1588 on performance, scalability, and solving complex, tightly coupled problems within
1589 science. This has led to the development of supercomputers with many thousands
1590 of cores, high speed, low latency interconnects, and sometimes also coprocessors and
1591 FPGAs [166, 167]. Only recently have these systems been evaluated from a cloud

1592 perspective [43]. An overarching goal exists to provide HPC Infrastructure as its own
1593 service (HPCaaS) [168], aiming to classify and limit the overhead of virtualization, and
1594 reducing the bottlenecks classically found in CPU, memory, and I/O operations within
1595 hypervisors [44, 169]. Furthermore, the transition from HPC to cloud computing
1596 becomes more complicated when we consider adding GPUs to the equation.

1597 GPU availability within a cloud is a new concept that has sparked a large amount
1598 of interest within the community. The first successfully deployment of GPUs within
1599 a cloud environment was the Amazon EC2 GPU offering. A collaboration between
1600 Nvidia and Citrix also exists to provide cloud-based gaming solutions to users using
1601 the new Kepler GPU architecture [170]. However, this is currently not targeted
1602 towards HPC applications.

1603 The task of providing a GPU accelerator for use in a virtualized cloud environment
1604 is one that presents a myriad of challenges. This is due to the complicated nature of
1605 virtualizing drivers, libraries, and the heterogeneous offerings of GPUs from multiple
1606 vendors. Currently, two possible techniques exist to fill the gap in providing GPUs
1607 in a cloud infrastructure: back-end I/O virtualization, which this chapter focuses on,
1608 and Front-end remote API invocation.

1609 4.3.1 Front-end Remote API invocation

1610 One method for using GPUs within a virtualized cloud environment is through front-
1611 end library abstractions, the most common of which is remote API invocation. Also
1612 known as API remoting or API interception, it represents a technique where API
1613 calls are intercepted and forwarded to a remote host where the actual computation
1614 occurs. The results are then returned to the front-end process that spawned the
1615 invocation, potentially within a virtual machine. The goal of this method is to provide
1616 an emulated device library where the actual computation is offloaded to another

1617 resource on a local network.

1618 Front-end remote APIs for GPUs have been implemented by a number of differ-
1619 ent technologies for different uses. To solve the problem of graphics processing in
1620 VMs, VMWare [171] has developed a device-emulation approach that emulates the
1621 Direct3D and OpenGL calls to leverage the host OS graphics processing capabili-
1622 ties to provide a 3D environment within a VM. API interception through the use of
1623 wrapper binaries has also been implemented by technologies such as Chromium [172],
1624 and Blink. However these graphics processing front-end solutions are not suitable for
1625 general purpose scientific computing, as they do not expose interfaces that CUDA or
1626 OpenCL can use.

1627 Currently, efforts are being made to provide a front-end remote API invocation
1628 solutions for the CUDA programming architecture. vCUDA [51] was the first of such
1629 technologies to enable transparent access of GPUs within VMs by API call inter-
1630 ception and redirection of the CUDA API. vCUDA substitutes the CUDA runtime
1631 library and supports a transmission mode using XMLRPC, as well as a sharing mode
1632 that is built on VMRPC, a dedicated remote procedure call architecture for VMM
1633 platforms. This share model can leads to better performance, especially as the volume
1634 of data increases, although there may be limitations in VMM interoperability.

1635 Like vCUDA, gVirtuS uses API interception to enable transparent CUDA, OpenCL,
1636 and OpenGL support for Xen, KVM, and VMWare virtual machines [173]. gVirtuS
1637 uses a front-end/back-end model to provide a VMM-independent abstraction layer
1638 to GPUs. Data transport from gVirtuS' front-end to the back-end is accomplished
1639 through a combination of shared memory, sockets, or other hypervisor-specific APIs.
1640 gVirtuS' primary disadvantage is in its decreased performance in host-to-device and
1641 device-to-host data movement due to overhead of data copies to and from its shared
1642 memory buffers. Recent work has also enabled the dynamic sharing of GPUs by

1643 leveraging the gVirtus back-end system with relatively good results [174], however
1644 process-level GPU resource sharing is outside the scope of this manuscript.

1645 rCUDA [50], a recent popular remote CUDA framework, also provides remote API
1646 invocation to enable VMs to access remote GPU hardware by using a sockets based
1647 implementation for high-speed near-native performance of CUDA based applications.
1648 rCUDA recently added support for using InfiniBand’s high speed, low latency network
1649 to increase performance for CUDA applications with large data volume requirements.
1650 rCUDA version 4.1 also supports the CUDA runtime API version 5.0, which supports
1651 peer device memory access and unified addressing. One drawback of this method
1652 is that rCUDA cannot implement the undocumented and hidden functions within
1653 the runtime framework, and therefore does not support all CUDA C extensions.
1654 While rCUDA provides some support tools, native execution of CUDA programs
1655 is not possible and programs need to be recompiled or rewritten to use rCUDA.
1656 Furthermore, like gVirtuS and many other solutions, performance between host-to-
1657 device data movement is only as fast as the underlying interconnect, and in the best
1658 case with native RDMA InfiniBand, is roughly half as fast as native PCI Express
1659 usage when using the standard QDR InfiniBand.

1660 4.3.2 Back-end PCI passthrough

1661 Another approach to using a GPU in a virtualized environment is to provide a VM
1662 with direct access to the GPU itself, instead of relying on a remote API. This chapter
1663 focuses on such an approach. Devices on a host’s PCI-express bus are virtualized
1664 using directed I/O virtualization technologies recently implemented by chip manu-
1665 facturers, and then direct access is relinquished upon request to a guest VM. This
1666 can be accomplished using the VT-d and IOMMU instruction sets from Intel and
1667 AMD, respectively. This mechanism, typically called PCI passthrough, uses a mem-

1668 memory management unit (MMU) to handle direct memory access (DMA) coordination
1669 and interrupt remapping directly to the guest VM, thus bypassing the host entirely.
1670 With host involvement being nearly non-existent, near-native performance of the PCI
1671 device within the guest VM can be achieved, which is an important characteristic for
1672 using a GPU within a cloud infrastructure.

1673 PCI passthrough itself has recently become a standard technique for many other
1674 I/O systems such as storage or network controllers. However, GPUs (even from
1675 the same vendor) have additional legacy VGA compatibility issues and non-standard
1676 low-level interface DMA interactions that make direct PCI passthrough nontrivial.
1677 VMWare has started use of a vDGA system for hardware GPU utilization, however it
1678 remains in tech preview and only documentation for Windows VMs is present [171]. In
1679 our experimentation, we have found that the Xen hypervisor provides a good platform
1680 for performing PCI passthrough of GPU devices to VMs due to its open nature,
1681 extensive support, and high degree of reconfigurability. Work with Xen in [175] gives
1682 hints at good performance for PCI passthrough in Xen, however further evaluation
1683 with independent benchmarks is needed when looking at scientific computing with
1684 GPUs.

1685 Today's GPUs can provide a variety of frameworks for application programmers to
1686 use. Two common solutions are CUDA and OpenCL. CUDA, or the Compute Unified
1687 Device Architecture, is a framework for creating and running parallel applications on
1688 Nvidia GPUs. OpenCL provides a more generic and open framework for parallel
1689 computation on CPUs and GPUs, and is available for a number of Nvidia, AMD, and
1690 Intel GPUs and CPUs. While OpenCL provides a more robust and portable solution,
1691 many HPC applications utilize the CUDA framework. As such, we focus only on
1692 Nvidia based CUDA-capable GPUs as they offer the best support for a wide array of
1693 programs, although this work is not strictly limited to Nvidia GPUs.

1694 4.4 Implementation

1695 In this chapter we use a specific host environment to enable PCI passthrough. First,
1696 we start with the Xen 4.2.2 hypervisor on Centos 6.4 and a custom 3.4.50-8 Linux
1697 kernel with Dom0 Xen support. Within the Xen hypervisor, GPU devices are seized
1698 upon boot and assigned to the xen-pciback kernel module. This process blocks the
1699 host devices from loading the Nvidia or nouveau drivers, keeping the GPUs uninitialized
1700 and therefore able to be assigned to DomU VMs.

1701 Xen, like other hypervisors, provides a standard method of passing through PCI
1702 devices to guest VMs upon creation. When assigning a GPU to a new VM, Xen loads
1703 a specific VGA BIOS to properly initialize the device enabling DMA and interrupts
1704 to be assigned to the guest VM. Xen also relinquishes control of the GPU via the
1705 xen-pciback module. From there, the Linux Nvidia drivers are loaded and the device
1706 is able to be used as expected within the guest. Upon VM termination, the xen-
1707 pciback module re-seizes the GPU and the devices can be re-assigned to new VMs in
1708 the future.

1709 This mechanism of PCI passthrough for GPUs can be implemented using multiple
1710 devices per host, as illustrated in Figure 6.1. Here, we see how the device's connection
1711 to the VM totally bypasses the Dom0 host as well as the Xen VMM, and is managed
1712 by VT-d or IOMMU to access the PCI-Express bus which the GPUs utilize. This is in
1713 contrast to other common virtual device uses, where hardware is emulated by the host
1714 and shared across all guests. This is the common usage for Ethernet controllers and
1715 input devices to enable users to interact with VMs as they would with native hosts,
1716 unlike the bridged model shown in the figure. The potential downside of this method
1717 is there can be a 1:1 or 1:N mapping of VMs to GPUs only. A M:1 mapping where
1718 multiple VMs use a GPU is not possible. However, almost all scientific applications

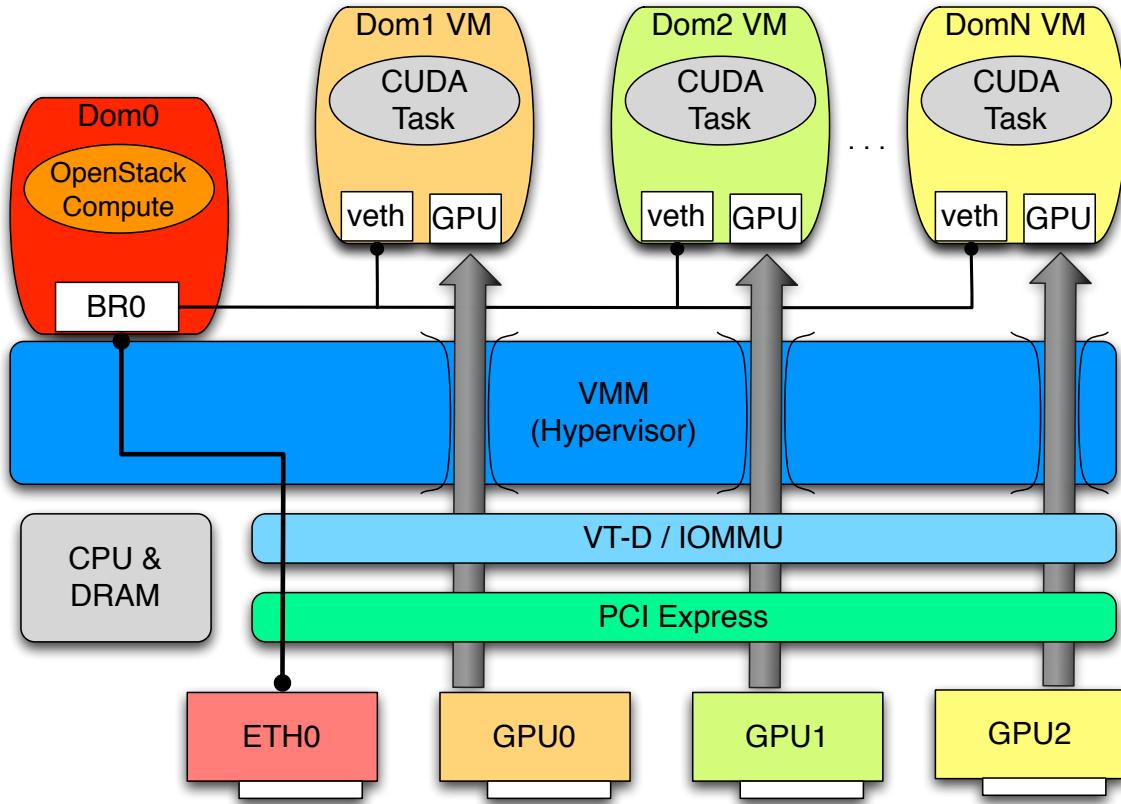


Figure 4.1 GPU PCI passthrough within the Xen Hypervisor

1719 environments using GPUs generally do not share GPUs between processes or other
 1720 nodes, as doing so would cause unpredictable and serious performance degradation.
 1721 As such, this GPU isolation within a VM can be considered an advantage in many
 1722 contexts.

1723 4.4.1 Feature Comparison

1724 Using the GPU PCI passthrough technique described previously has a number of
 1725 advantages compared to front-end API implementations. First, it allows for an op-
 1726 erating environment that more closely relates to native bare-metal usage of GPUs.
 1727 Essentially, a VM provides a nearly identical infrastructure to clusters and supercom-
 1728 puters with integrated GPUs. This lowers the learning curve for many researchers,

and even enables researchers to potentially use other tools within a VM that might not be supported within a supercomputer or cluster. Unlike with remote API implementations, users don't need to recompile or modify their code, as the GPUs are essentially local to the data. Further comparing to remote API implementations, using PCI passthrough within a VM allows users to leverage any GPU framework available, OpenCL or CUDA, and any higher level programming frameworks such as within Matlab or Python.

Through the use of advanced scheduling techniques within cloud infrastructure, we can also take advantage of PCI passthrough implementation for efficiency purposes. Users could request VMs with GPUs which get scheduled for creation on machines that provide such resources, but can also request normal VMs as well. The scheduler can correctly map VM requirement requests to heterogeneous hardware. This enables large scale resources to support a wide array of scientific computing applications without the added cost of putting GPUs in all compute nodes.

4.5 Experimental Setup

In this chapter back-end GPU PCI passthrough to virtual machines using the Xen hypervisor is detailed, however proper evaluation of the performance of such method needs to be properly considered. As such, we ran an array of benchmarks that evaluate the performance of this method compared to the same hardware running native bare-metal GPU code without any virtualization. We focus our tests on single-node performance to best understand low level overhead.

To evaluate the effectiveness of GPU-enabled VMs within Xen, two different machines were used to represent two generations of Nvidia GPUs. The first system at Indiana University consists of dual-socket Intel Xeon X5660 6-core CPUs at 2.8Ghz

1753 with 192GB DDR3 RAM, 8TB RAID5 array, and two Nvidia Tesla "Fermi" C2075
1754 GPUs. The system at USC/ISI uses a dual-socket Intel Xeon E5-2670 8-core CPUs
1755 at 2.6Ghz with 48GB DDR3 RAM, 3 600GB SAS disk drives, but with the latest
1756 Nvidia Tesla "Kepler" K20m GPU supplied by Nvidia. These machines represent
1757 the present Fermi series GPUs along with the recently release Kepler series GPUs,
1758 providing a well-rounded experimental environment. Native systems were installed
1759 with standard Centos 6.4 with a stock 2.6.32-279 Linux kernel. Xen host systems
1760 were still loaded with Centos 6.4 but with a custom 3.4.53-8 Linux kernel and Xen
1761 4.2.22. Guest VMs were also Centos 6.4 with N-1 processors and 24GB of memory
1762 and 1 GPU passed through in HVM full virtualization mode. Using both IU and
1763 USC/ISI machine configurations in native and VM modes represent the 4 test cases
1764 for our work.

1765 In order to evaluate the performance, the SHOC Benchmark suite [176] was used
1766 to extensively evaluate performance across each test platform. The SHOC bench-
1767 marks were chosen because they provide a higher level of evaluation regarding GPU
1768 performance than the sample applications provided in the Nvidia SDK, and can also
1769 evaluate OpenCL performance in similar detail. The benchmarks were compiled us-
1770 ing the NVCC compiler in CUDA5 driver and library, along with OpenMPI 1.4.2 and
1771 GCC 4.4. Each benchmark was run a total of 20 times, with the results given as an
1772 average of all runs.

1773 4.6 Results

1774 Results of all benchmarks are compressed into three subsections: floating point oper-
1775 ations, device bandwidth and pci bus performance. Each represents a different level
1776 of evaluation for GPU-enabled VMs compared to bare-metal native GPU usage.

4.6.1 Floating Point Performance

Flops, or floating point operations per second, are a common measure of computational performance, especially within scientific computing. The Top 500 list [8] has been the relative gold standard for evaluating supercomputing performance for more than two decades. With the advent of GPUs in some of the fastest supercomputers today, including GPUs identical to those used in our experimentation, understanding the performance relative to this metric is imperative.

Figure 4.2 shows the raw peak FLOPs available using each GPU in both native and virtualized modes. First, we observe the advantage of using the Kepler series GPUs over Fermi, with peak single precision speeds tripling in each case. Even for double precision FLOPs, there is roughly a doubling of performance with the new GPUs. However its most interesting that there is less than a 1% overhead when using GPU VMs compared to native case for pure floating point operations. The K20m-enabled VM was able to provide over 3 Teraflops, a notable computational feat for any single node.

Figures 4.3 and 4.4 show Fast Fourier Transform (FFT) and the Matrix Multiplication implementations across both test platforms. For all benchmarks that do not take into account the PCI-Express (pcie) bus transfer time, we again see near-native performance using the Kepler GPUs and Fermi GPUs when compared to bare metal cases. Interestingly, overhead within Xen VMs is consistently less than 1%, confirming the synthetic MaxFlops benchmark above. However, we do see some performance impact when calculating the total FLOPs with the pcie bus in the equation. This performance decrease ranges significantly for the C2075-series GPU, roughly about a 15% impact for FFT and a 5% impact for Matrix Multiplication. This overhead in pcie runs is not as pronounced for the Kepler K20m test environment, with near-native performance in all cases (less than 1%).

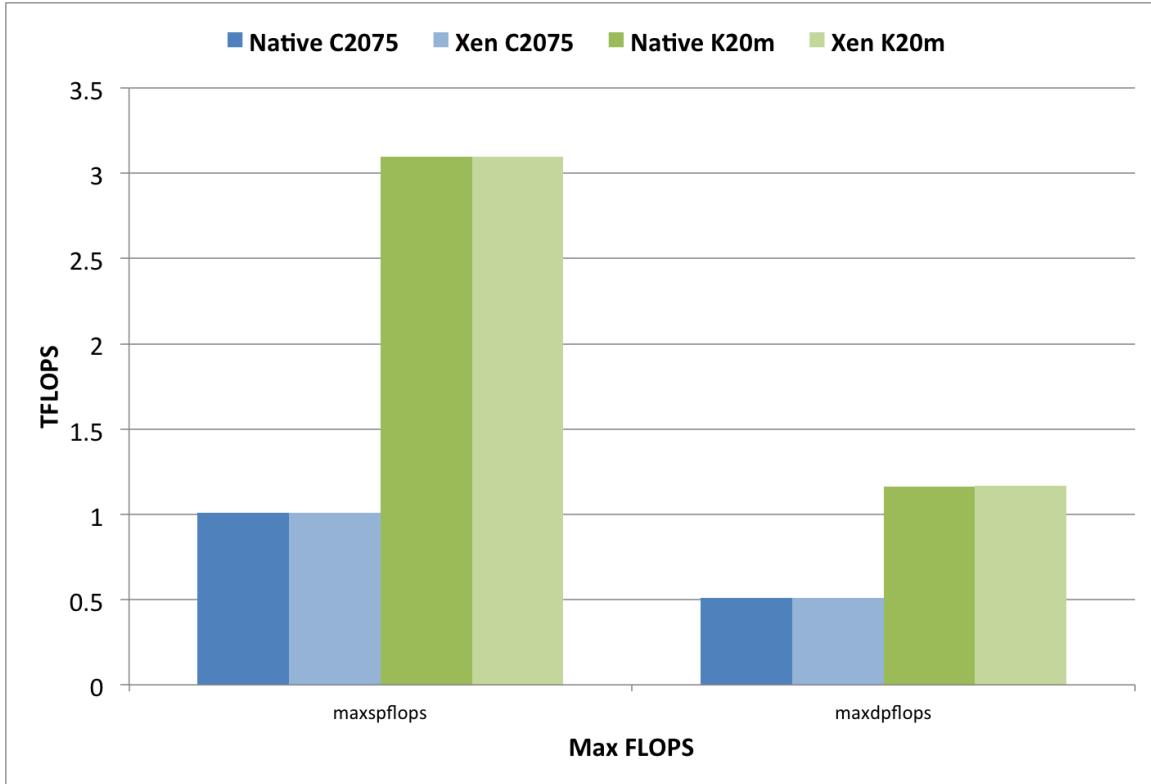


Figure 4.2 GPU Floating Point Operations per Second

1803 Other FLOP-based benchmarks are used to emulate higher level applications.
 1804 Stencil represents a 9-point stencil operation applied to a 2D data set, and the S3D
 1805 benchmark is a computationally-intensive kernel from the S3D turbulent combustion
 1806 simulation program [177]. In Figure 4.5, we see that both the Fermi C2075 and Kepler
 1807 K20m GPUs performing well compared to the native base case, showing the overhead
 1808 of virtualization is low. The C2075-enabled VMs experience slightly more overhead
 1809 when compared to native performance again for pcie runs, but overhead is at most
 1810 7% for the S3D benchmark.

1811 4.6.2 Device Speed

1812 While floating point operations allow for the proposed solution to relate to many tradi-
 1813 tional HPC applications, they are just one facet of GPU performance within scientific

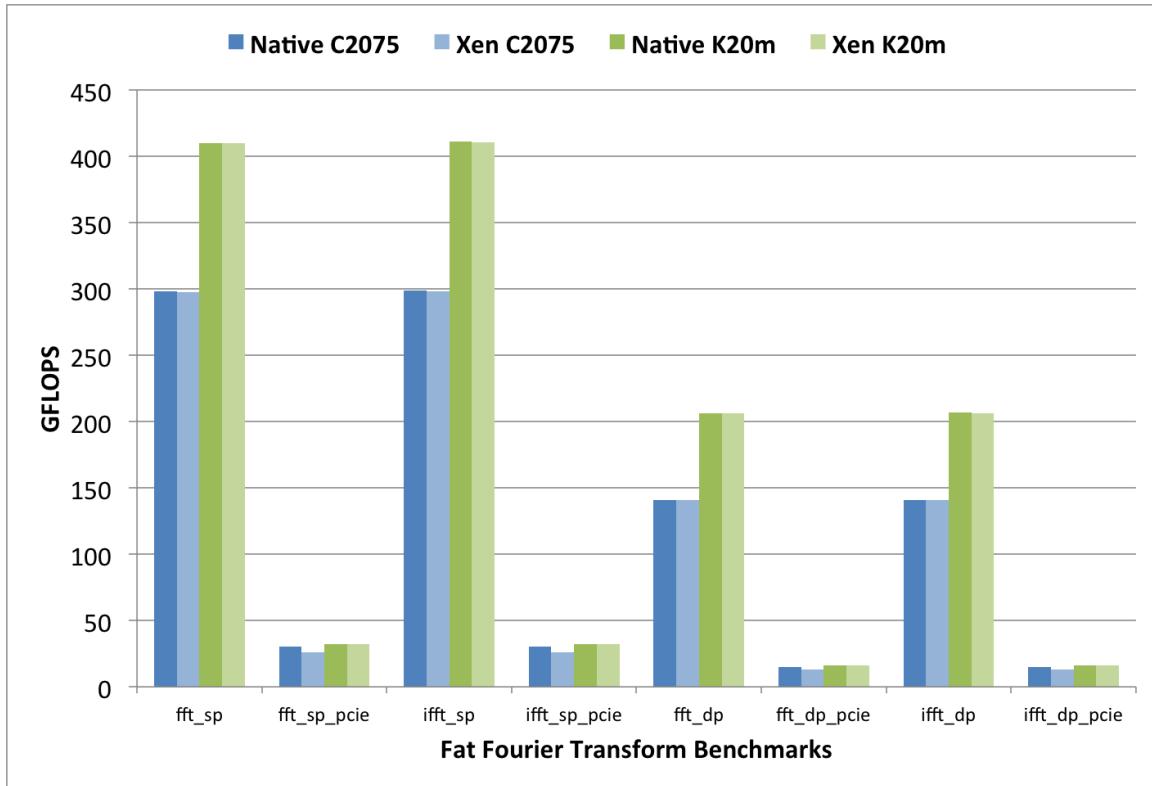


Figure 4.3 GPU Fast Fourier Transform

1814 computing. Device speed, measured in both raw bandwidth and additional bench-
 1815 marks, provides a different perspective towards evaluating GPU PCI passthrough in
 1816 Xen. Figure 4.6 illustrates device level memory access of various GPU device mem-
 1817 ory structures. With both Nvidia GPUs, virtualization has little to no impact on the
 1818 performance of inter-device memory bandwidth. As expected the Kepler K20m out-
 1819 performed the C2075 VMs and there was a higher variance between runs with both
 1820 native and VM cases. Molecular Dynamics and Reduction benchmarks in Figure 4.7
 1821 perform again at near-native performance without the pcie bus taken into account.
 1822 However the overhead observed increases to 10-15% when the PCI-Express bus is
 1823 considered when looking at the Fermi C2075 VMs.

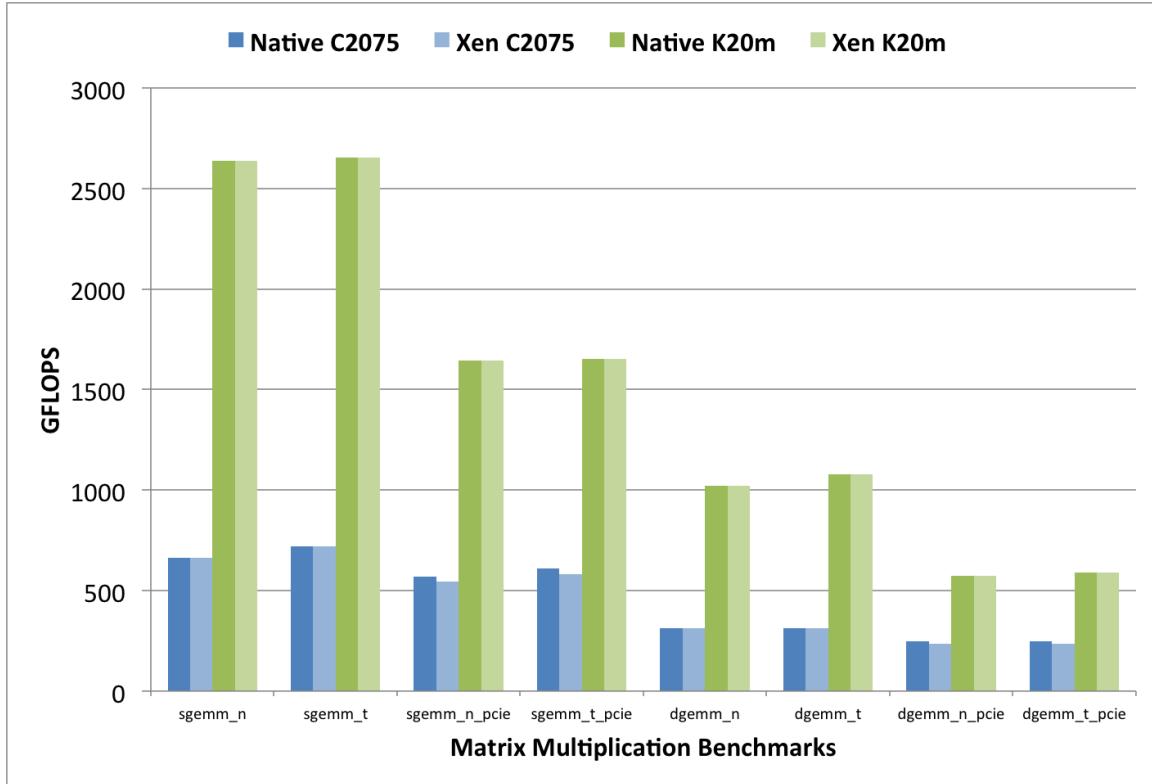


Figure 4.4 GPU Matrix Multiplication

1824 4.6.3 PCI Express Bus

1825 Upon evaluating PCI passthrough of GPUs, it is apparent that the PCI express bus
 1826 is subject to the greatest potential for overhead, as was observed in the Fermi C2075
 1827 benchmarks. The VT-d and IOMMU chip instruction sets interface directly with the
 1828 PCI bus to provide operational and security related mechanisms for each PCI device,
 1829 thereby ensuring proper function in a multi-guest environment but potentially intro-
 1830 ducing some overhead. As such, it is imperative to investigate any and all overhead
 1831 at the PCI Express bus.

1832 Figure 4.8 looks at maximum PCI bus speeds for each experimental implemen-
 1833 tation. First, we see a consistent overhead in the Fermi C2075 VMs, with a 14.6%
 1834 performance impact for download (to-device) and a 26.7% impact in readback (from-

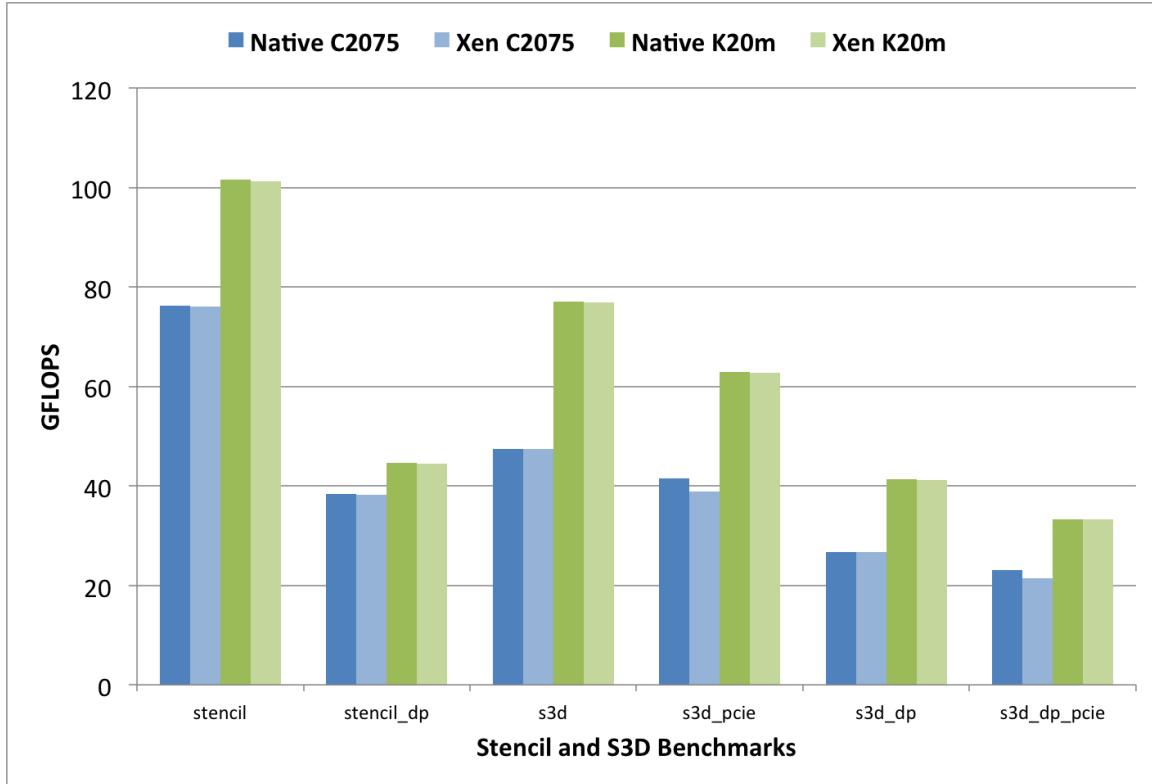


Figure 4.5 GPU Stencil and S3D

device). However, the Kepler K20 VMs do not experience the same overhead in the PCI-Express bus, and instead perform at near-native performance. These results indicate that the overhead is minimal in the actual VT-d mechanisms, and instead due to other factors.

Upon further examination, we have identified the performance degradation within the Fermi-based VM experimental setup is due to the testing environment's NUMA node configuration with the Westmere-based CPUs. With the Fermi nodes, the GPUs are connected through the PCI-Express bus from CPU Socket #1, yet the experimental GPU VM was run using CPU Socket #0. This meant that the VM's PCI-Express communication involved more host involvement with Socket #1, leading to a notable decrease in read and write speeds across the PCI-Express Bus. Such architectural limitations seem to be relieved in the next generation with a Sandy-Bridge CPU ar-

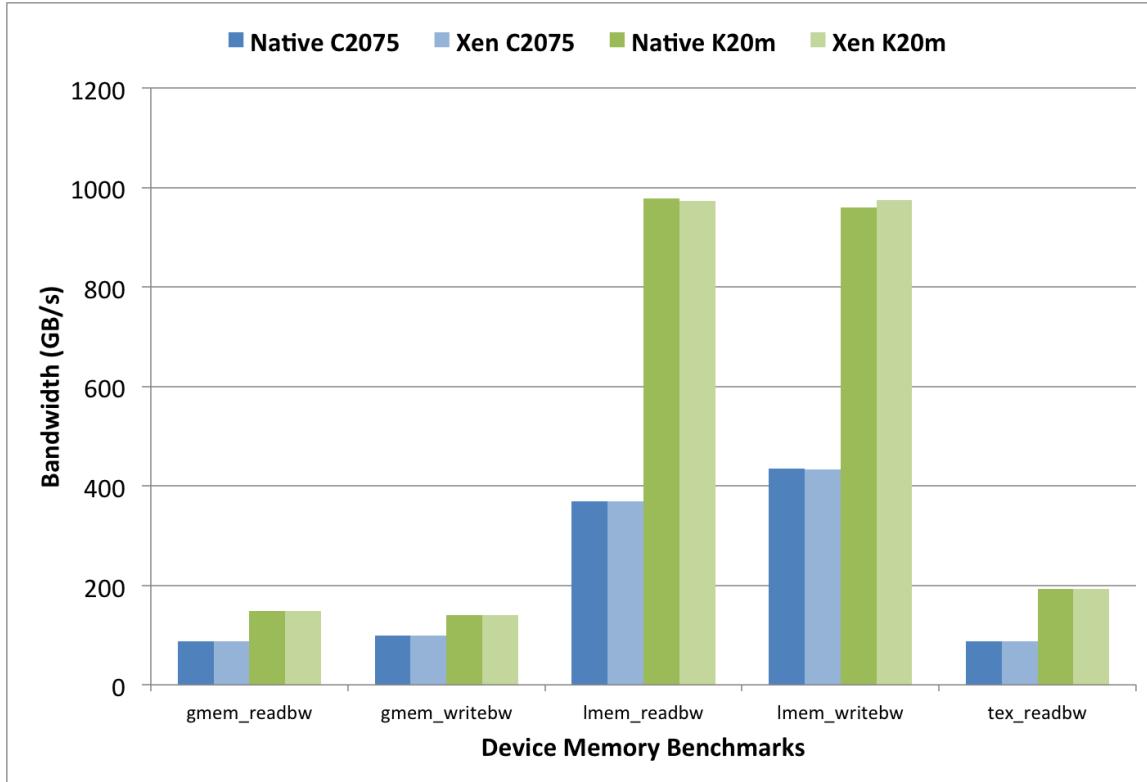


Figure 4.6 GPU Device Memory Bandwidth

chitecture and the Kepler K20m experimental setup. In summary, GPU PCI-Express performance in a VM is sensitive to the host CPU's NUMA architecture and care is needed to mitigate the impact, either by leveraging new architectures or by proper usage of Xen's VM core assignment features. Furthermore, the overhead in this system diminishes significantly when using the new Kepler GPUs by Nvidia.

4.7 Discussion

This chapter evaluates the use of general purpose GPUs within cloud computing infrastructure, primarily targeted towards advanced scientific computing. The method of PCI passthrough of GPUs directly to a guest virtual machine running on a tuned Xen hypervisor shows initial promise for an ubiquitous solution in cloud infrastruc-

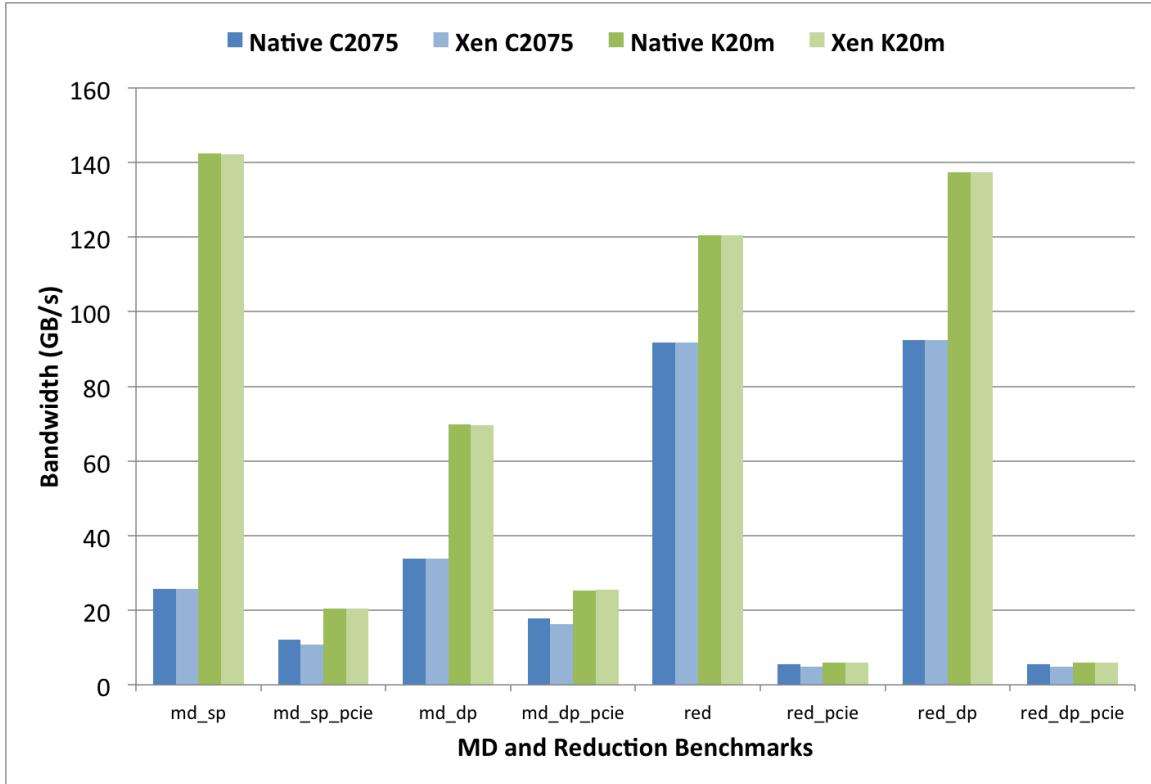


Figure 4.7 GPU Molecular Dynamics and Reduction

ture. In evaluating the results in the previous section, a number of points become clear.

First, we can see that there is a small overhead in using PCI passthrough of GPUs within VMs, compared to native bare-metal usage, which represents the best possible use case. As with all abstraction layers, some overhead is usually inevitable as a necessary trade-off to added feature sets and improved usability. The same is true for GPUs within Xen VMs. The Kepler K20m GPU-enabled VMs operated at near-native performance for all runs, with a 1.2% reduction at worst in performance. The Fermi based C2075 VMs experience more overhead due to the NUMA configuration that impacted PCI-Express performance. However, when the overhead of the PCI-Express bus is not considered, the C2075 VMs perform at near-native speeds.

GPU PCI passthrough also has the potential to perform better than other front-

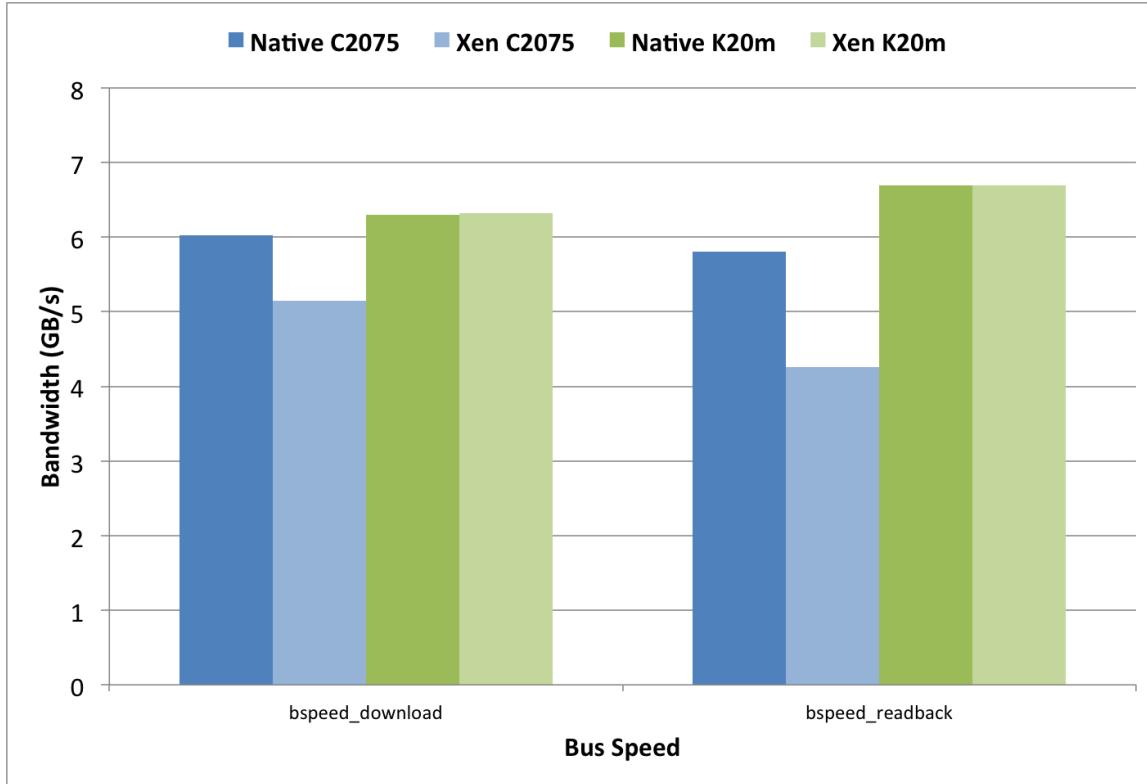


Figure 4.8 GPU PCI Express Bus Speed

end API solutions that are applicable within VMs, such as gVirtus or rCUDA. This is because such solutions are designed to communicate via a network interconnect such as 10Gb Ethernet or QDR InfiniBand [178], which introduces an inherent bottleneck. Even with the theoretically optimal configuration of rCUDA using QDR InfiniBand, the maximum theoretical bus speed is 40Gbs, which is comparably less than the measured 54.4Gps real-world performance measured between host-to-device transfers with GPU-enabled Kepler VMs.

Overall, it is our hypothesis that the overhead in using GPUs within Xen VMs as described in this chapter will largely go unnoticed by most mid-level scientific computing applications. This is especially true when using the latest Sandy-Bridge CPUs with the Kepler series GPUs. We expect many mid-tier scientific computing groups to benefit the most from the ability to use GPUs in a scientific cloud infras-

1881 tructure. Already this has been confirmed in [179], where similar a methodology has
1882 been leveraged specifically for Bioinformatics applications in the cloud.

1883 **4.8 Chapter Summary and Future Work**

1884 The ability to use GPUs within virtual machines represents a leap forward for sup-
1885 porting advanced scientific computing within cloud infrastructure. The method of
1886 direct PCI passthrough of Nvidia GPUs using the Xen hypervisor offers a clean, re-
1887 producible solution that can be implemented within many Infrastructure-as-a-Service
1888 (IaaS) deployments. Performance measurements indicate that the overhead of pro-
1889 viding a GPU within Xen is minimal compared to the best-case native use, however
1890 NUMA inconsistencies can impact performance. The New Kepler-based GPUs oper-
1891 ate with a much lower overhead, making those GPUs an ideal choice when designing
1892 a new GPU IaaS system.

1893 Next steps for this work could involve providing GPU-based PCI passthrough
1894 within the OpenStack nova IaaS framework. This will enable research laboratories
1895 and institutions to create new private or national-scale cloud infrastructure that have
1896 the ability to support new scientific computing challenges. Other hypervisors could
1897 also leverage GPU PCI passthrough techniques and warrant in-depth evaluation in
1898 the future. Furthermore, we hope to integrate this work with advanced interconnects
1899 and other heterogeneous hardware and provide a parallel high performance cloud
1900 infrastructure to enable mid-tier scientific computing.

₁₉₀₁ **Chapter 5**

₁₉₀₂ **GPU-Passthrough Performance: A**
₁₉₀₃ **Comparison of KVM, Xen,**
₁₉₀₄ **VMWare ESXi, and LXC for**
₁₉₀₅ **CUDA and OpenCL Applications**

₁₉₀₆ **5.1 Introduction**

₁₉₀₇ As scientific workloads continue to demand increasing performance at greater power
₁₉₀₈ efficiency, high performance architectures have been driven towards heterogeneity and
₁₉₀₉ specialization. Intel's Xeon Phi, and GPUs from both NVIDIA and AMD represent
₁₉₁₀ some of the most common accelerators, with each capable of delivering improved
₁₉₁₁ performance and power efficiency over commodity multi-core CPUs.

₁₉₁₂ Infrastructure-as-a-Service (IaaS) clouds have the potential to democratize access
₁₉₁₃ to the latest, fastest, and most powerful computational accelerators. This is true
₁₉₁₄ of both public and private clouds. Yet today's clouds are typically homogeneous

1915 without access to even the most commonly used accelerators. Historically, enabling
1916 virtual machine access to GPUs and other PCIe devices has proven complex and
1917 error-prone, with only a small subset of GPUs being certified for use within a few
1918 commercial hypervisors. This is especially true for NVIDIA GPUs, likely the most
1919 popular for scientific computing, but whose drivers have always been closed source.

1920 Given the complexity surrounding the choice of GPUs, host systems, and hypervi-
1921 sors, it is perhaps no surprise that Amazon is the only major cloud provider offering
1922 customers access to GPU-enabled instances. All of this is starting to change, however,
1923 as open source and other freely available hypervisors now provide sufficiently robust
1924 PCI passthrough functionality to enable GPU and other accelerator access whether
1925 in the public or private cloud.

1926 Today, it is possible to access GPUs at high performance within all of the major
1927 hypervisors, merging many of the advantages of cloud computing (e.g. custom images,
1928 software defined networking, etc.) with the accessibility of on-demand accelerator
1929 hardware. Yet, no study to date has systematically compared the performance of PCI
1930 passthrough across all major cloud hypervisors. Instead, alternative solutions have
1931 been proposed that attempt to virtualize the GPU [180] , but sacrifice performance.

1932 In this chapter, we characterize the performance of both NVIDIA Fermi and
1933 Kepler GPUs operating in PCI passthrough mode in VMWare VSphere, Linux KVM,
1934 Xen, and Linux Containers (LXC). Through a series of microbenchmarks as well as
1935 scientific and Big Data applications, we make two contributions:

1936 1. We demonstrate that PCI passthrough at high performance is possible for GPUs
1937 across 4 major hypervisors.

1938 2. We describe the lessons learned through our performance analysis, as well as
1939 the relative advantages and disadvantages of each hypervisor for GPU support.

1940 5.2 Related Work & Background

1941 GPU virtualization and GPU-passthrough are used within a variety of contexts, from
1942 high performance computing to virtual desktop infrastructure. Accessing one or more
1943 GPUs within a virtual machine is typically accomplished by one of two strategies: 1)
1944 via API remoting with device emulation; or 2) using PCI passthrough.

1945 5.2.1 GPU API Remoting

1946 rCUDA, vCUDA, GViM, and gVirtuS are well-known API remoting solutionsFun-
1947 damentally, these approaches operate similarly by splitting the driver into a front-
1948 end/back-end model, where calls into the interposed CUDA library (front-end) are
1949 sent via shared memory or a network interface to the back-end service that executes
1950 the CUDA call on behalf of the virtual machine. Notably, this technique is not limited
1951 to CUDA, but can be used to decouple OpenCL, OpenGL, and other APIs from their
1952 local GPU or accelerator.

1953 The performance of API-remoting depends largely on the application and the
1954 remoting solution's implementation. Bandwidth and latency-sensitive benchmarks and
1955 applications will tend to expose performance bottlenecks more than compute-intensive
1956 applications. Moreover, solutions that rely on high speed networks, such as Infini-
1957 band, will compete with application-level networking for bandwidth.

1958 5.2.2 PCI Passthrough

1959 Input/Output Memory Management Units, or IOMMUs, play a fundamental roll in
1960 the PCI-passthrough virtualization mechanism. Like traditional MMUs that provide
1961 a virtual memory address space to CPUs [181], an IOMMU serves the fundamental
1962 purpose of connecting a direct memory access (DMA) capable I/O bus to main mem-

1963 ory. The IOMMU unit, typically within the chipset, maps device virtual addresses to
1964 physical memory addresses. This process also has the added improvement of guaran-
1965 teeing device isolation by blocking rogue DMA and interrupt requests [182], with a
1966 slight overhead, especially in early implementations [183].

1967 Currently two major IOMMU implementations exist, VT-d and AMD-Vi by In-
1968 tel and AMD, respectively. Both specifications provide DMA remapping to enable
1969 PCI-passthrough as well as other features such as interrupt remapping, hypervisor
1970 snooping, and security control mechanisms to ensure proper and efficient hardware
1971 utilization. PCI passthrough has been studied within the context of networking [184],
1972 storage [185], and other PCI-attached devices; however, GPUs have historically lagged
1973 behind other devices in their support for virtual machine passthrough.

1974 **5.2.3 GPU Passthrough, a Special Case of PCI Passthrough**

1975 While generic PCI passthrough can be used with IOMMU technologies to pass through
1976 many PCI-Express devices, GPUs represent a special case of PCI devices, and a spe-
1977 cial case of PCI passthrough. In traditional usage, GPUs usually serve as VGA
1978 devices primarily to render screen output, and while the GPUs used in this study do
1979 not render screen out, the function still exists in legacy. In GPU-passthrough, another
1980 VGA device (such as onboard graphics built into the motherboard, or a baseboard
1981 management controller) is necessary to serve as the primary display for the host, as
1982 well as providing emulated VGA devices for each guest VM. Most GPUs also have a
1983 video BIOS that requires full initialization and reset functions, which is often difficult
1984 due to the proprietary nature of the cards and their drivers.

1985 Nevertheless, for applications that require native or near-native GPU performance
1986 across the full spectrum of applications with immediate access to the latest GPU
1987 drivers and compilers, GPU passthrough solutions are preferable to API remoting.

1988 Today, Citrix Xenserver, open source Xen [186], and VMWare ESXi [187], and most
1989 recently KVM all support GPU passthrough. To our knowledge, no one has system-
1990 atically characterized the performance of GPU passthrough across a range of hyper-
1991 visors, across such a breadth of benchmarks, and across multiple GPU generations as
1992 we do.

1993 **5.3 Experimental Methodology**

1994 **5.3.1 Host and Hypervisor Configuration**

1995 We used two hardware systems, named Bespin and Delta, to evaluate four hypervisors.
1996 The Bespin system at USC/ISI represents Intel’s Sandy Bridge microarchitecture with
1997 a Kepler class K20 GPU. The Delta system, provided by the FutureGrid project [188],
1998 represents the Westmere microarchitecture with a Fermi class C2075 GPU. Table 5.1
1999 provides the major hardware characteristics of both systems. Note that in addition,
2000 both systems include 10 gigabit Ethernet, gigabit Ethernet, and either FDR or QDR
2001 Infiniband. Our experiments do not emphasize networking, and we use the gigabit
2002 ethernet network for management only.

2003 A major design goal of these experiments was to reduce or eliminate NUMA effects
2004 (non-uniform memory access) on the PCI passthrough results in order to facilitate
2005 fair comparisons across hypervisors and to reduce experimental noise. To this end,
2006 we configured our virtual machines and containers to execute only on the NUMA
2007 node containing the GPU under test. We acknowledge that the NUMA effects on
2008 virtualization may be interesting in their own right, but they are not the subject of
2009 this set of experiments.

2010 We use Bespin and Delta to evaluate three hypervisors and one container-based
2011 approach to GPU passthrough. The hypervisors and container system, VMWare

Table 5.1 Host hardware configurations

	Bespin	Delta
CPU (cores)	2x E5-2670 (16)	2x X5660 (12)
Clock Speed	2.6 GHz	2.6 GHz
RAM	48 GB	192 GB
NUMA Nodes	2	2
GPU	1x K20m	2x C2075

2012 ESXi, Xen, KVM, and LXC, are summarized in Table 5.2. Note that each virtual-
 2013 ization solution imposes its own unique requirements on the base operating system.
 2014 Hence, Xen’s Linux kernel refers to the Domain-0 kernel, whereas KVM’s Linux kernel
 2015 represents the actual running kernel hosting the KVM hypervisor. Linux Containers
 2016 share a single kernel between the host and guests, and VMWare ESXi does not rely
 2017 on a Linux kernel at all.

2018 Similarly, hypervisor requirements prevented us from standardizing on a single
 2019 host operating system. For Xen and KVM, we relied on the Arch Linux 2013.10.01
 2020 distribution because it provides easy access to the mainline Linux kernel. For our
 2021 LXC tests, we use CentOS 6.4 because its shared kernel was identical to the base
 2022 CentOS 6.4 kernel used in our testing. VMWare has a proprietary software stack.
 2023 All of this makes comparison challenging, but as we describe in Section 5.3.2, we are
 2024 running a common virtual machine across all experiments.

Table 5.2 Host Hypervisor/Container Configuration

Hypervisor	Linux Kernel	Linux Version
KVM	3.12	Arch 2013.10.01
Xen 4.3.0-7	3.12	Arch 2013.10.01
VMWare ESXi 5.5.0	N/A	N/A
LXC	2.6.32-358.23.2	CentOS 6.4

2025 5.3.2 Guest Configuration

2026 We treat each hypervisor as its own system, and compare virtual machine guests
 2027 to a base CentOS 6.4 system. The base system and the guests are all composed of
 2028 CentOS 6.4 installation with a 2.6.32-358.23.2 stock kernel and CUDA version 5.5.
 2029 Each guest is allocated 20 GB of RAM and a full CPU socket (either 6 or 8 CPU
 2030 cores). Bespin experiments received 8 cores and Delta experiments received 6 cores.
 2031 VMs were restricted to a single NUMA node. On the Bespin system, the K20m GPU
 2032 was attached to NUMA node 0. On the Delta system, the C2075 GPU was attached
 2033 to NUMA node 1. Hence VMs ran on NUMA node 0 for the Bespin experiments,
 2034 and node 1 for the Delta experiments.

2035 5.3.3 Microbenchmarks

2036 Our experiments are composed of a mix of microbenchmarks and application-level
 2037 benchmarks, as well as a combination of CUDA and OpenCL benchmarks. The
 2038 SHOC benchmark suite provides a series of microbenchmarks in both OpenCL and
 2039 CUDA [189]. For this analysis, we focus on the OpenCL benchmarks in order to
 2040 exercise multiple programming models. Benchmarks range from low-level peak Flops

2041 and bandwidth measurements, to kernels and mini-applications.

2042 **5.3.4 Application Benchmarks**

2043 For our application benchmarks, we have chosen the LAMMPS molecular dynam-
2044 ics simulator [190], the GPU-LIBSVM [191], and the LULESH shock hydrodynamics
2045 simulator [192]. These represent a range of computational characteristics, from com-
2046 putational physics to big data analytics, and are representative of GPU-accelerated
2047 applications in common use.

2048 **LAMMPS** The Large-scale Atomic/Molecular Parallel Simulator (LAMMPS), is a
2049 parallel molecular dynamics simulator [190, 193] used for production MD simulation
2050 on both CPUs and GPUs [194]. LAMMPS has two packages for GPU support, the
2051 USER-CUDA and GPU packages. With the USER-CUDA package, each GPU is used
2052 by a single CPU, whereas the GPU package allows multiple CPUs to take advantage
2053 of a single GPU. There are performance trade-offs with both approaches, but we chose
2054 to use the GPU package in order to stress the virtual machine by exercising multiple
2055 CPUs. Consistent with the existing GPU benchmarking approaches, our results are
2056 based on the Rhodopsin protein.

2057 **GPU-LIBSVM** LIBSVM is a popular implementation [195] of the machine learn-
2058 ing classification algorithm support vector machine (SVM). GPU-accelerated LIB-
2059 SVM [191] enhances LIBSVM by providing GPU-implementations of the kernel ma-
2060 trix computation portion of the SVM algorithm for radial basis kernels. For bench-
2061 marking purposes we use the NIPS 2003 feature extraction gisette data set. This data
2062 set has a high dimensional feature space and large number of training instances, and
2063 these qualities are known to be computational intensive to generate SVM models.

2064 The GPU-accelerated SVM implementation shows dramatic improvement over the
2065 CPU-only implementation.

2066 **LULESH** Hydrodynamics is widely used to model continuum properties and inter-
2067 actions in materials when there is an applied force [196]. Hydrodynamics applications
2068 consume approximately one third of the runtime of data center resource throughout
2069 the U.S. DoD (Department of Defense). The Livermore Unstructured Lagrange Ex-
2070 plicit Shock Hydro (LULESH) was developed by Lawrence Livermore National Lab
2071 as one of five challenge problems in the DARPA UHPC program. LULESH is widely
2072 used as a proxy application in the U.S. DOE (Department of Energy) co-design effort
2073 for exascale applications [192].

2074 5.4 Performance Results

2075 We characterize GPGPU performance within virtual machines across two hardware
2076 systems, 4 hypervisors, and 3 application sets. We begin with the SHOC benchmark
2077 suite before describing the GPU-LIBSVM, LAMMPS, and LULESH results. All
2078 benchmarks are run 20 times and averaged. Results are scaled with respect to a base
2079 CentOS 6.4 system for both systems. That is, we compare virtualized Bespin per-
2080 formance to non-virtualized Bespin performance, and virtualized Delta performance
2081 to non-virtualized Delta performance. Values less than 1 indicate that the base sys-
2082 tem outperformed the virtual machine, while values greater than 1 indicate that the
2083 virtual machine outperformed the base system. In cases where we present geometric
2084 means across multiple benchmarks, the means are taken over these scaled values, and
2085 the semantics are the same: less than 1 indicates overhead in the hypervisor, greater
2086 than 1 indicates a performance increase over the base system.

Table 5.3 SHOC overheads for Bespin (K20) expressed as geometric means of scaled values within a level, while maximum overheads are expressed as a percentage.

		Bespin (K20)							
		KVM		Xen		LXC		VMWare	
		Mean	Max	Mean	Max	Mean	Max	Mean	Max
L0	0.999	1.57	0.997	3.34	1.00	1.77	1.00	1.90	
L1	0.998	1.23	0.998	1.39	1.00	1.47	1.00	0.975	
L2	0.998	0.48	0.995	0.846	0.999	1.90	0.999	1.20	
Bespin PCIe-only									
		KVM		Xen		LXC		VMWare	
		Mean	Max	Mean	Max	Mean	Max	Mean	Max
L0	0.997	0.317	0.999	0.143	0.995	0.981	0.995	1.01	
L1	0.998	0.683	0.997	1.39	1.00	0.928	1.01	0.975	
L2	0.998	0.478	0.996	0.846	1.00	0.247	1.01	0.133	

Table 5.4 SHOC overheads for Delta (c2075) expressed as geometric means of scaled values within a level, while maximum overheads are expressed as a percentage.

		Delta (C2075)							
		KVM		Xen		LXC		VMWare	
		Mean	Max	Mean	Max	Mean	Max	Mean	Max
L0	1.01	0.031	0.969	12.7	1.00	0.073	1.00	4.95	
L1	1.00	1.45	0.959	24.0	1.00	0.663	0.933	36.6	
L2	1.00	0.101	0.982	4.60	1.00	0.016	0.962	7.01	
Delta PCIe-only									
		KVM		Xen		LXC		VMWare	
		Mean	Max	Mean	Max	Mean	Max	Mean	Max
L0	1.04	0.029	0.889	12.7	1.00	0.01	0.995	4.37	
L1	1.00	1.45	0.914	20.5	0.999	0.380	0.864	36.6	
L2	1.00	0.075	0.918	4.60	1.00	N/A	0.869	7.01	

2087 5.4.1 SHOC Benchmark Performance

2088 SHOC splits its benchmarks into 3 Levels, named Levels 0 through 2. Level 0 repre-
2089 sents device-level characteristics, peak Flop/s, bandwidth, etc. Level 1 characterizes
2090 computational kernels: FFT and matrix multiplication, among others. Finally, Level
2091 2 includes “mini-applications,” in this case an implementation of the S3D, a compu-
2092 tational chemistry application.

2093 Because the SHOC OpenCL benchmarks report more than 70 individual mi-
2094 crobenchmarks, space does not allow us to show each benchmark individually. In-
2095 stead, we start with a broad overview of SHOC’s performance across all benchmarks,
2096 hypervisors, and systems. We then discuss in more detail those benchmarks that
2097 either outperformed or underperformed the Bespin (K20) system by 0.50% or more.
2098 We call these benchmarks outliers. As we will show, those outlier benchmarks iden-
2099 tified on the Bespin system, also tend to exhibit comparable characteristics on the
2100 Delta system as well, but the overhead is typically higher.

2101 In Tables 5.3 and 5.4, we provide geometric means for each SHOC level across each
2102 hypervisor and system. We also include the maximum overhead for each hypervisor
2103 at each level to facilitate comparison across hypervisors and systems. Finally, we
2104 provide a comparable breakdown of only the PCIe SHOC benchmarks, based on the
2105 intuition that PCIe-specific benchmarks will likely result in higher overhead.

2106 At a high level, we immediately notice that in the cases of KVM and LXC, both
2107 perform very near native across both the Bespin and Delta platforms. On average,
2108 these systems are almost indistinguishable from their non-virtualized base systems.
2109 So much so, that experimental noise occasionally boosts performance slightly above
2110 their base systems.

2111 This is in sharp contrast to the Xen and VMWare hypervisors, which perform
2112 well on the Bespin system, but poorly on the Delta system in some cases. This is

particularly evident when looking at the maximum overheads for Xen and VMWare across both systems. In this case, we see that on the Bespin system, Xen's maximum overhead of 3.34% is dwarfed by Delta's maximum Xen overhead of 24.0%. VMWare exhibits similar characteristics, resulting in a maximum overhead of 1.9% in the case of the Bespin system, and a surprising 36.6% in the case of the Delta system. We provide a more in-depth discussion of these overheads below.

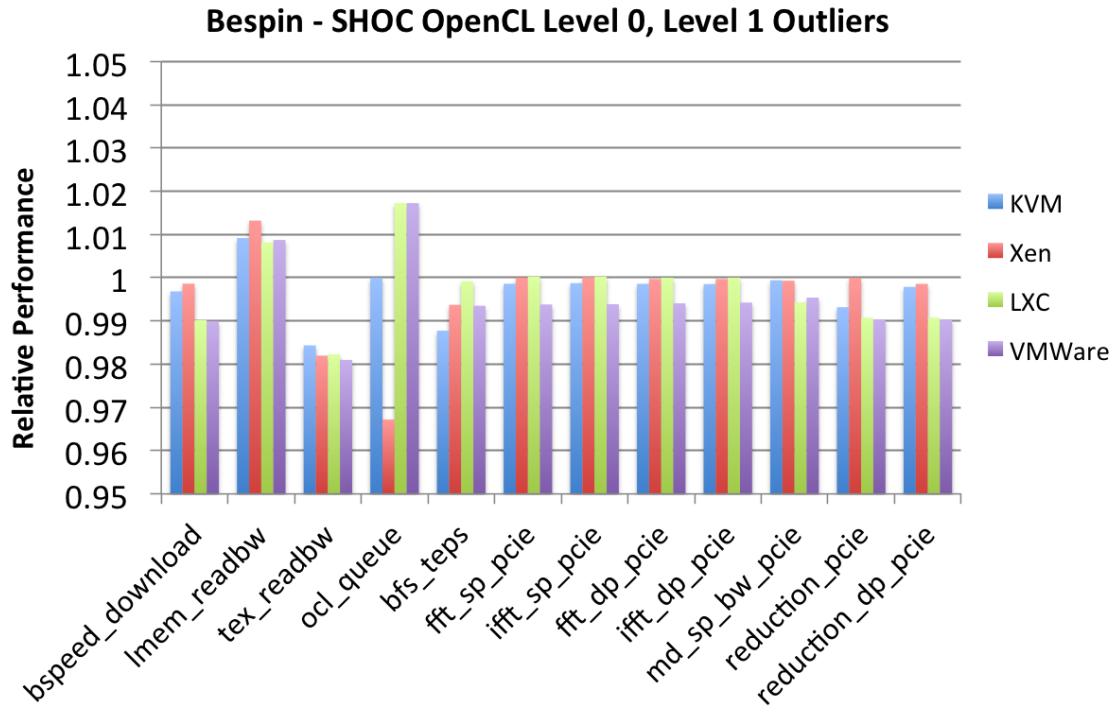


Figure 5.1 SHOC Levels 0 and 1 relative performance on Bespin system. Results show benchmarks over or under-performing by 0.5% or greater. Higher is better.

Of the Level 0 benchmarks, only four exhibited notable overhead in the case of the Bespin system: bspeed_download, lmem_readbw, tex_readbw, and ocl_queue. These are shown in Figure 5.1. These benchmarks represent device-level characteristics, including host-to-device bandwidth, onboard memory reading, and OpenCL kernel queue delay. Of the four, only bspeed_download incurs a statistically significant

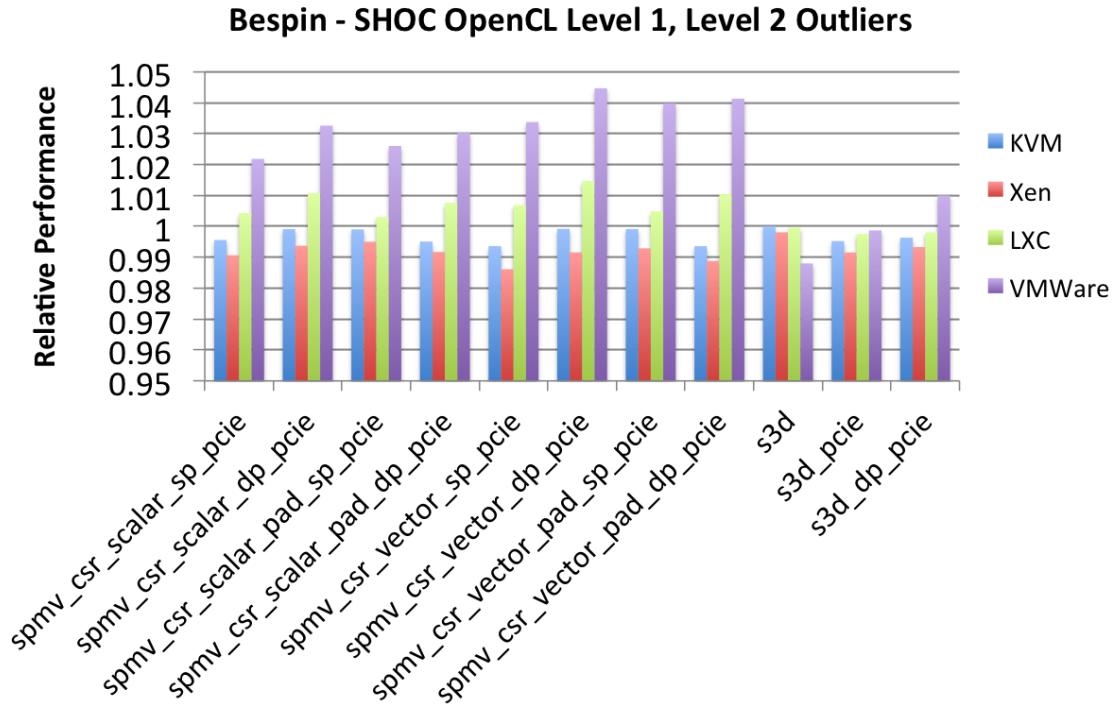


Figure 5.2 SHOC Levels 1 and 2 relative performance on Bespin system. Results show benchmarks over or under-performing by 0.5% or greater. Higher is better.

overhead. The remainder perform within one standard deviation of the base, despite an overhead of greater than 0.5%.

bspeed_download is representative of the most likely source of virtualization overhead, data movement across the PCI-Express bus. The PCIe lies at the boundary of the virtual machine and the physical GPU, and requires interrupt remapping, IOMMU interaction, etc. in order to enable GPU passthrough into the virtual machine. Despite this, in Figure 5.1 we see a maximum of 1% overhead for bspeed_download in VMWare, and less than 0.5% overhead for both KVM and Xen.

The remainder of Figure 5.1 includes a series of SHOC's Level 1 benchmarks, representing computational kernels. This includes BFS, FFT, molecular dynamics, and reduction kernels. Notably, nearly all of the benchmarks exhibiting overhead are the

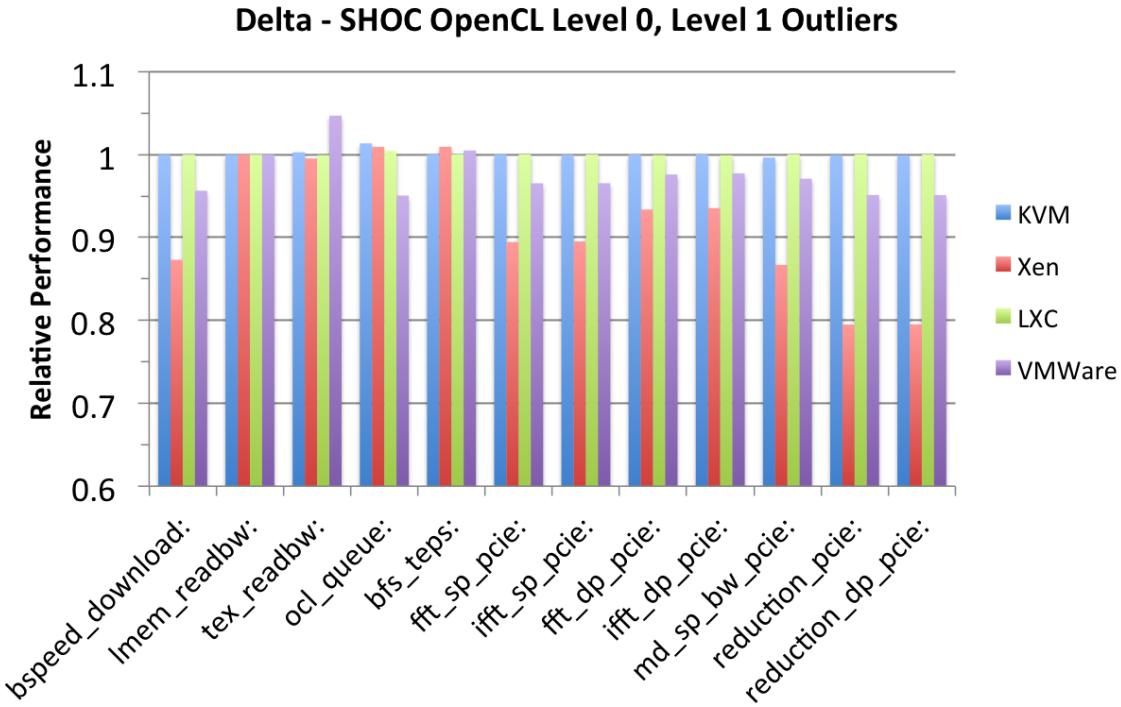


Figure 5.3 SHOC Levels 0 and 1 relative performance on Delta system. Benchmarks shown are the same as Bespin's. Higher is better.

2135 PCIe portion of SHOC's benchmarks. This is unsurprising, since the Level 0 benchmarks
 2136 suggest PCIe bandwidth as the major source overhead. Still, results remain
 2137 consistent with the bspeed_download overhead observed in the Level 0 benchmarks,
 2138 further suggesting that host/device data movement is the major source of overhead.

2139 In Figure 5.2 we present the remaining SHOC Level 1 results as well as the SHOC
 2140 Level 2 results (S3D). In these results we begin to see an interesting trend, namely
 2141 that VMWare consistently outperforms the base system in the Spmv and S3D mi-
 2142 crobenchmarks on the Bespin system. We believe this to be a performance regression
 2143 in CentOS 6.4, rather than a unique improvement due to the VMWare ESXi hyper-
 2144 visor. When running the benchmarks on a non-virtualized Bespin system with Arch
 2145 Linux, the VMWare ESXi performance gains were erased. An interesting finding of
 2146 this was that Spmv was unique in this way - no other benchmarks were affected by

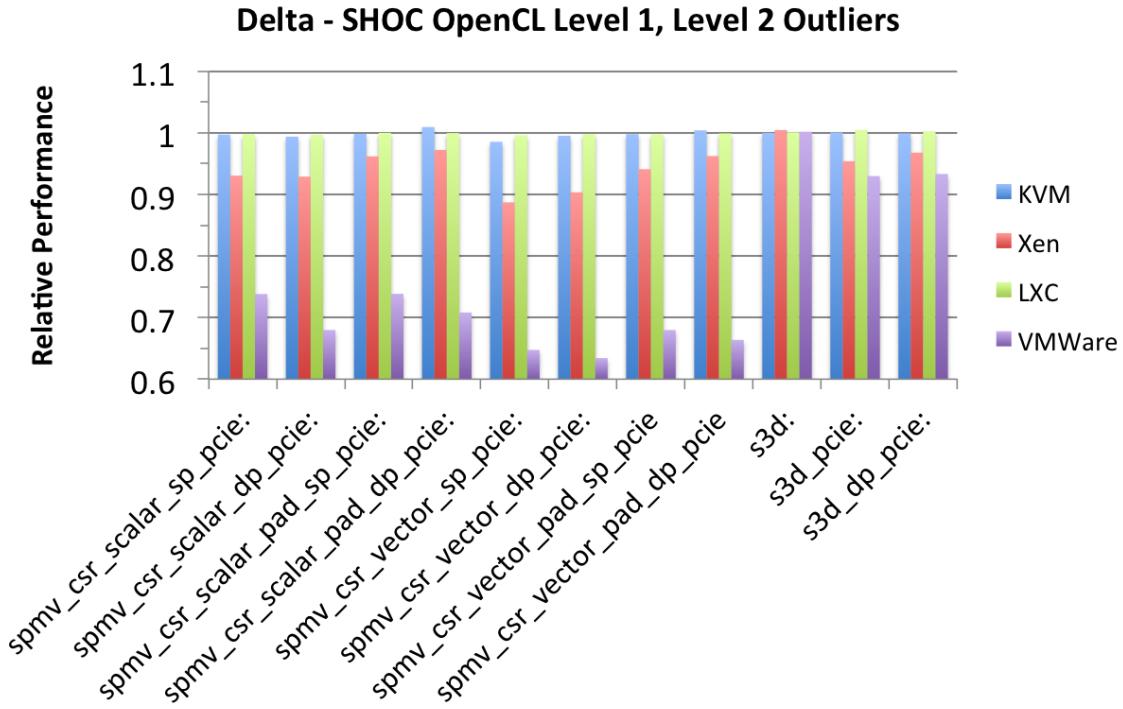


Figure 5.4 SHOC Levels 1 and 2 relative performance. Benchmarks shown are the same as Bespin's. Higher is better.

2147 this performance issue.

2148 Turning to the Delta system, in Figures 5.3 and 5.4, we show the same bench-
 2149 marks for the Delta system as was shown in Figures 5.1 and 5.2. Again, we find that
 2150 the same benchmarks are responsible for most of the overhead on the Delta system.
 2151 This is unsurprising, since PCIe was shown to be the source of the bulk of the over-
 2152 head. A major difference in the case of the Delta system, however, is the amount
 2153 of overhead. While the Bespin system saw overheads of approximately 1%, Delta's
 2154 overhead routinely jumps above 35%, especially in the case of the Spmv benchmark
 2155 for VMWare.

2156 On further examination, we determined that Xen was unable to activate IOMMU
 2157 large page tables on the Delta system. KVM successfully allocated 4k, 2M, and 1G
 2158 page table sizes, while Xen was limited to size 4k page tables. The Bespin system

2159 was able to take advantage of 4k, 2M, and 1G page sizes on both KVM and Xen. It
2160 appears that this issue is correctable and does not represent a fundamental limitation
2161 to the Xen hypervisor on the Nehalem/Westmere microarchitecture. While as a
2162 closed source product, we have limited insight into VMWare ESXi, we speculate that
2163 VMWare may be experiencing a similar issue on the Delta system and not on our
2164 Bespin system.

2165 In light of this, we broadly find that virtualization overhead across hypervisors and
2166 architectures is minimal. Questions remain as to the source of the exceptionally high
2167 overhead in the case of Xen and VMWare on the Delta system, but because KVM
2168 shows no evidence of this overhead, we believe the Westmere/Fermi architecture to
2169 be suitable for VGA passthrough in a cloud environment. In the case of the Bespin
2170 system, it is clear that VGA passthrough can be achieved across hypervisors with
2171 virtually no overhead.

2172 A surprising finding is that LXC showed little performance advantage over KVM,
2173 Xen, and VWMare. While we expected near-native performance from LXC we did
2174 not expect the hardware-assisted hypervisors to achieve such high performance. Still,
2175 LXC carries some advantages. In general, its maximum overheads are comparable to
2176 or less than KVM, Xen, and VMWare, especially in the case of the Delta system.

2177 **5.4.2 GPU-LIBSVM Performance**

2178 In Figures 5.5 and 5.6, we display our GPU-LIBSVM performance results for the
2179 Bespin (K20m) and Delta (C2075) systems. Using the NIPS 2003 gisette data set,
2180 we show the performance across 4 problems sizes, ranging from 1800 to 6000 training
2181 instances. The NIPS 2003 gisette dataset is a standard dataset that was used as a
2182 part of the NIPS 2003 feature selection challenge.

2183 KVM again performs well across both the Delta and Bespin systems. In the case

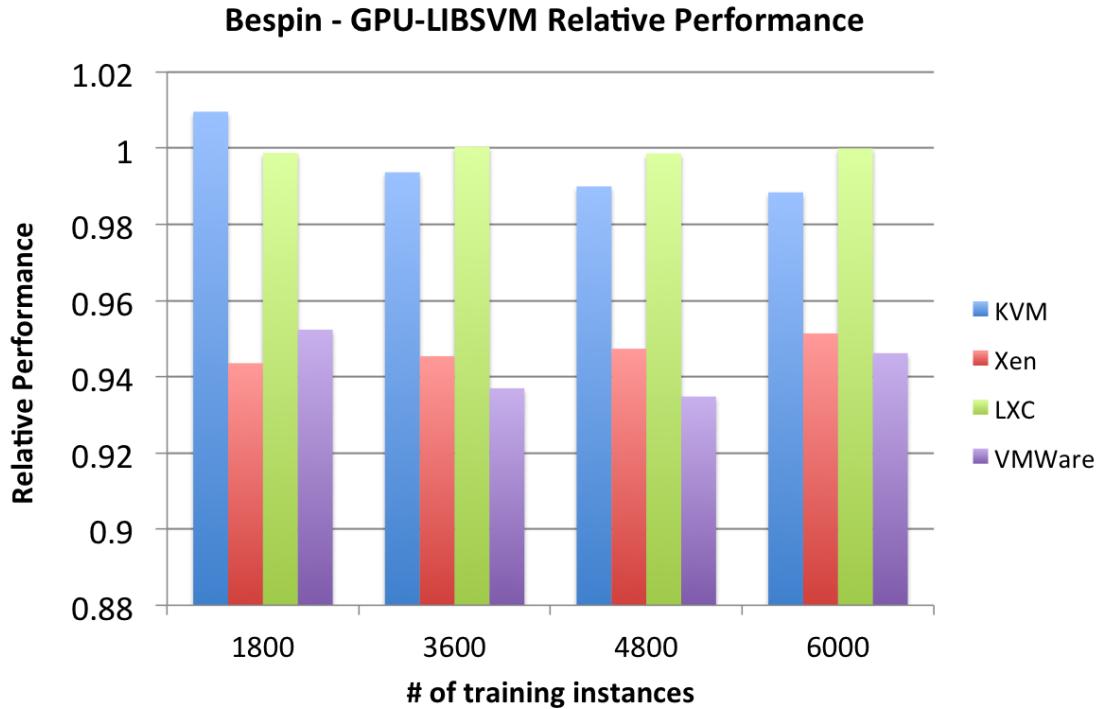


Figure 5.5 GPU-LIBSVM relative performance on Bespin system. Higher is better.

of the Delta system, in fact, KVM, significantly outperforms the base system. We determined this to be caused by KVM's support for transparent hugepages. When sufficient memory is available, transparent hugepages may be used to back the entirety of the guest VM's memory. Hugepages have previously been shown to improve TLB performance for KVM guests, and have been shown to occasionally boost the performance of a KVM guests beyond its host [197]. After disabling hugepages on the KVM host for the Delta system, performance dropped to 80–87% of the base system, suggesting that memory optimizations such as transparent hugepages can substantially improve the performance of virtualized guests under some circumstances. LXC and VMWare perform close to the base system, while Xen achieves between 72–90% of the base system's performance. We speculate that this may be related to Xen's inability to enable page sizes larger than 4k.

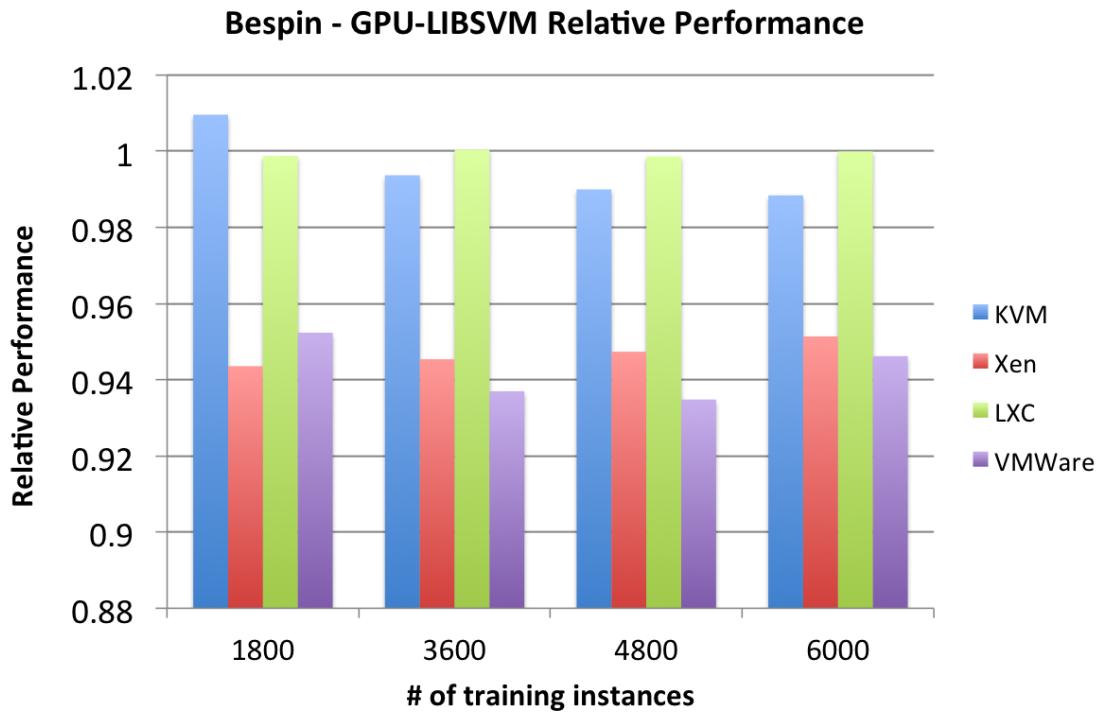


Figure 5.6 GPU-LIBSVM relative performance on Delta showing improved performance within virtual environments. Higher is better.

2196 5.4.3 LAMMPS Performance

2197 In Figures 5.7 and 5.8, we show the LAMMPS Rhodopsin protein simulation results.
 2198 LAMMPS is unique among our benchmarks, in that it exercises both the GPU and
 2199 multiple CPU cores. In keeping with the LAMMPS benchmarking methodology,
 2200 we execute the benchmark using 1 GPU and 1–8 CPU cores on the Bespin system,
 2201 selecting the highest performing configuration. In the case of the Delta system, we
 2202 execute the benchmark on 1–6 cores and 1 GPU, selecting the highest performing
 2203 configuration.

2204 Overall, LAMMPS performs well across both hypervisors and systems. Surpris-
 2205 ingly, LAMMPS showed better efficiency on the Delta system than the Bespin system,
 2206 achieving greater than 98% efficiency across the board, while Xen on the Bespin sys-

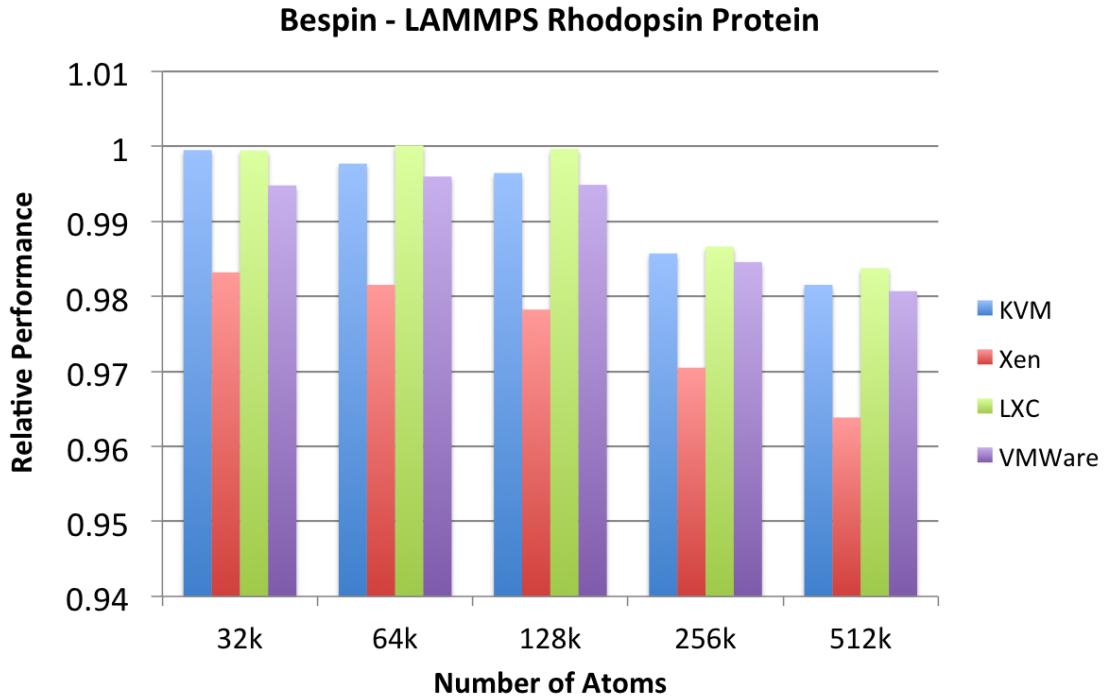


Figure 5.7 LAMMPS Rhodopsin benchmark relative performance for Bespin system. Higher is better.

tem occasionally drops as low as 96.5% efficiency.

This performance is encouraging because it suggests that even heterogeneous CPU + GPU code is capable of performing well in a virtualized environment. Unlike SHOC, GPU-LIBSVM, and LULESH, LAMMPS fully exercises multiple host CPUs, splitting work between one or more GPUs and one or more CPU cores. This has the potential to introduce additional performance overhead, but the results do not bear this out in the case of LAMMPS.

5.4.4 LULESH Performance

In Figure 5.9 we present our LULESH results for problem sizes ranging from mesh resolutions of $N = 30$ to $N = 150$. LULESH is a highly compute-intensive simulation, with limited data movement between the host/virtual machine and the GPU, making

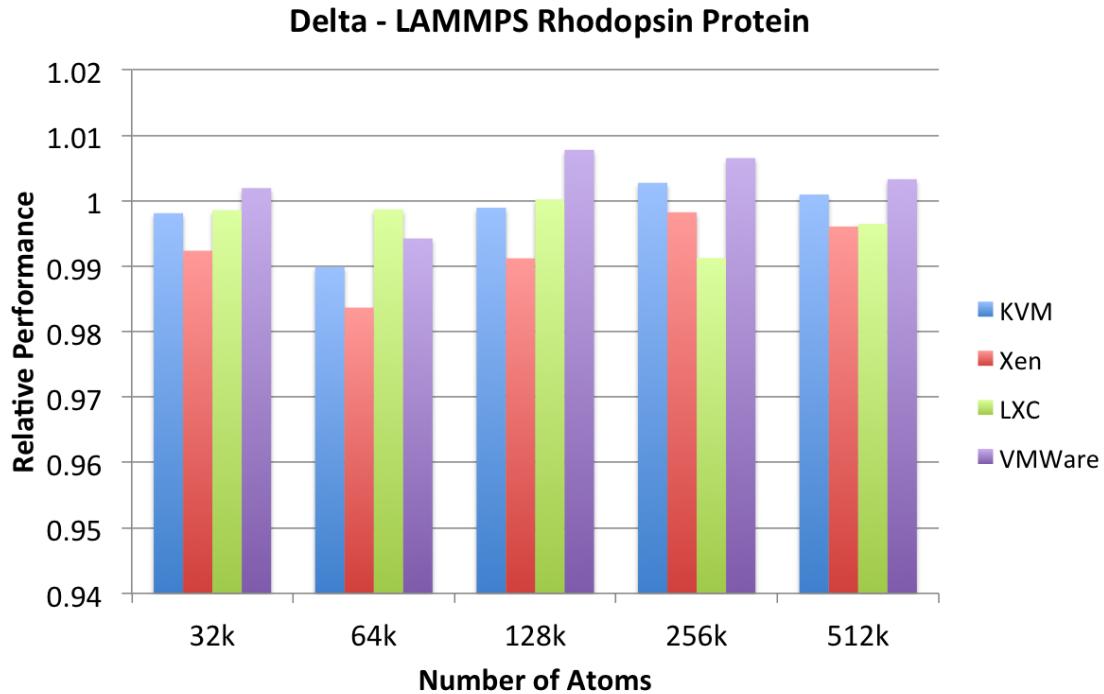


Figure 5.8 LAMMPS Rhodopsin benchmark relative performance for Delta system. Higher is better.

it ideal for GPU acceleration. Consequently, we would expect little overhead due to virtualization. We show LULESH results only on the Bespin system, because differences in the code bases between the Kepler and Fermi implementations led to unsound comparisons.

While, overall, we see very little overhead, there is a slight scaling effect that is most apparent in the case of the Xen hypervisor. As the mesh resolution (N^3) increases from $N = 30$ to $N = 150$, we see that the Xen overhead decreases until Xen performs on-par with KVM, LXC, and VMWare.

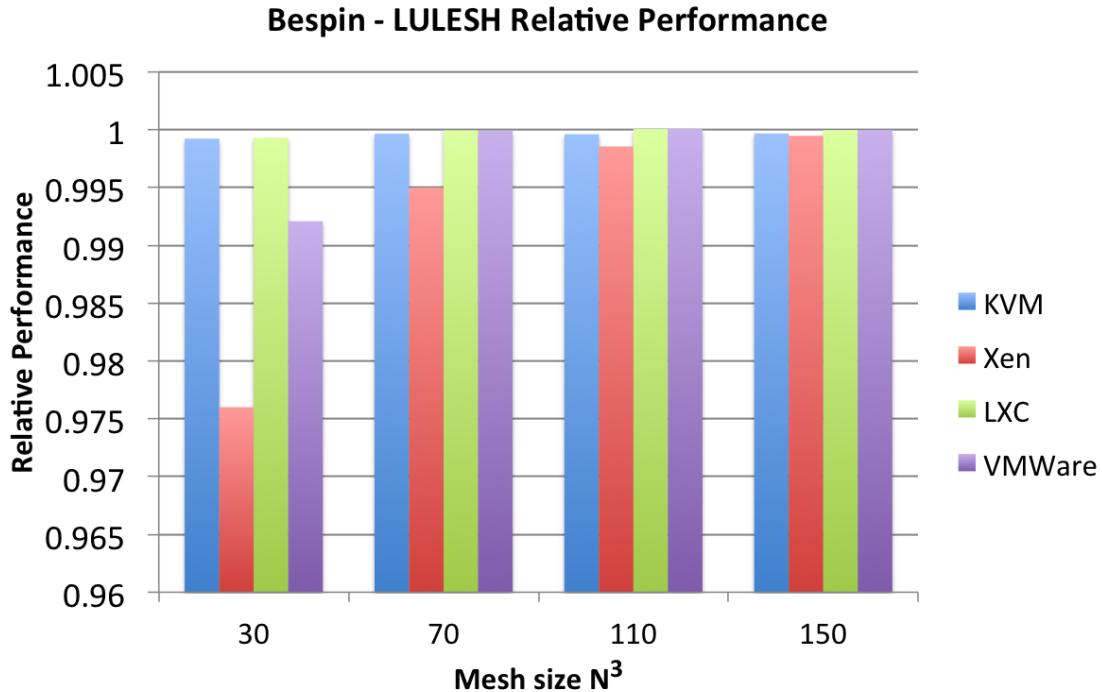


Figure 5.9 LULESH relative performance on Bespin. Higher is better.

2226 5.5 Lessons Learned

2227 Virtualizing performance-critical workloads has always proven controversial, whether
 2228 the workload is CPU-only [169] or CPU with GPU. From our Westmere results, we
 2229 can see that this criticism is in part legitimate, at times resulting in a performance
 2230 penalty of greater than 10%, especially in the case of the VMWare ESXi and Xen
 2231 hypervisors. We believe much of this to be fixable, especially in the case of the Xen
 2232 hypervisor.

2233 At the same time, however, we have shown that the Sandy Bridge processor genera-
 2234 tion has nearly erased those performance overheads, suggesting that old arguments
 2235 against virtualization for performance-critical tasks should be reconsidered. In light of
 2236 this, the primary lesson from this study is that VGA-passthrough to virtual machines
 2237 is achievable at low overhead, and across a variety of hypervisors and virtualization

2238 platforms. Virtualization performance remains inconsistent across hypervisors for the
2239 Westmere generation of processors, but starting with the Sandy Bridge architecture,
2240 performance and consistency increase dramatically. In the case of the Sandy Bridge
2241 architecture, even the lowest performing hypervisor, open source Xen, typically per-
2242 forms within 95% of the base case.

2243 This study has also yielded valuable insight into the merits of each hypervisor.
2244 KVM consistently yielded near-native performance across the full range of bench-
2245 marks. Its support for transparent hugepages resulted in slight performance boosts
2246 over-and-above even the base CentOS system in the case of the Delta system.

2247 VMWare's performance proved inconsistent across architectures, performing well
2248 in the case of Bespin, and relatively poorly in the case of the Delta system. Because
2249 hypervisor configurations were identical across systems, we can only speculate that
2250 VMWare's performance is aided by the virtualization improvements offered by the
2251 Sandy Bridge microarchitecture.

2252 The Xen hypervisor was consistently average across both architectures, perform-
2253 ing neither poorly nor extraordinarily well in any individual benchmark. Xen and
2254 VMWare ESXi are the only two hypervisors from this study that officially support
2255 VGA passthrough. As a result, PCI passthrough support in both Xen and VMWare is
2256 more robust than KVM. We expect that this advantage will not last long, as commer-
2257 cial solutions targeting PCI passthrough in KVM are becoming common, particularly
2258 with regard to SR-IOV and networking adapters.

2259 Linux Containers (LXC), consistently performed closest to the native case. This,
2260 of course, is not surprising given that LXC guests share a single kernel with their
2261 hosts. This performance comes at the cost of both flexibility and security, however.
2262 LXC is less flexible than its full virtualization counterparts, offering support for only
2263 Linux guests. More importantly, LXC device passthrough has security implications

2264 for multi-GPU systems. In the case of a multi-GPU-enabled host with NVIDIA
2265 hardware, both GPUs must be passed to the LXC guest in order to initialize the
2266 driver. This limitation may be addressable in future revisions to the NVIDIA driver.

2267 **5.6 Directions for Future Work**

2268 In this chapter we have characterized the performance of 4 common hypervisors across
2269 two generations of GPUs and two host microarchitectures, and across 3 sets of bench-
2270 marks. We showed the dramatic improvement in virtualization performance between
2271 the Fermi/Westmere and the Kepler/Sandy Bridge system, with the Sandy Bridge
2272 system typically performing within 1% of the base system. Finally, this study sought
2273 to characterize the GPU and CPU+GPU performance with carefully tuned hypervi-
2274 sor and guest configurations, especially with respect to NUMA. Improvements must
2275 be made to today’s hypervisors in order to improve virtual NUMA support. Finally,
2276 cloud infrastructure, such as OpenStack, must be capable of automatically allocating
2277 virtual machines in a NUMA-friendly manner in order to achieve acceptable results
2278 at cloud-scale.

2279 The next step in this work is to move beyond the single node to show that clus-
2280 ters of accelerators can be efficiently used with minimal overhead. This will require
2281 studies in high speed networking, particularly SR-IOV-enabled ethernet and Infini-
2282 band. Special attention is needed to ensure that latencies remain tolerable within
2283 virtual environments. Some studies have begun to examine these issues [198], but
2284 open questions remain.

2285 **Chapter 6**

2286 **Supporting High Performance
2287 Molecular Dynamics in Virtualized
2288 Clusters using IOMMU, SR-IOV,
2289 and GPUDirect**

2290 **6.1 Introduction**

2291 At present we stand at the inevitable intersection between High Performance Com-
2292 puting (HPC) and clouds. Various platform tools such as Hadoop and MapReduce,
2293 among others, have already percolated into data intensive computing within HPC [22].
2294 In addition, there are efforts to support traditional HPC-centric scientific computing
2295 applications in virtualized cloud infrastructure. There are a multitude of reasons for
2296 supporting parallel computation in the cloud [53], including features such as dynamic
2297 scalability, specialized operating environments, simple management interfaces, fault
2298 tolerance, and enhanced quality of service, to name a few. The growing importance

2299 of supporting advanced scientific computing using virtualized infrastructure can be
2300 seen by a variety of new efforts, including the NSF-funded Comet resource part of
2301 XSEDE at San Diego Supercomputer Center [199].

2302 Nevertheless, there exists a past notion that virtualization used in today's cloud
2303 infrastructure is inherently inefficient. Historically, cloud infrastructure has also done
2304 little to provide the necessary advanced hardware capabilities that have become al-
2305 most mandatory in supercomputers today, most notably advanced GPUs and high-
2306 speed, low-latency interconnects. The result of these notions has hindered the use
2307 of virtualized environments for parallel computation, where performance must be
2308 paramount.

2309 A growing effort is currently underway that looks to systematically identify and
2310 reduce any overhead in virtualization technologies. This effort has, thus far, proven
2311 to be a qualified success [169, 200], though further research is needed to address
2312 issues of scalability and I/O. Thus, we see a constantly diminishing overhead with
2313 virtualization, not only with traditional cloud workloads [201] but also with HPC
2314 workloads. While virtualization will almost always include some additional overhead
2315 in relation to its dynamic features, the eventual goal for supporting HPC in virtualized
2316 environments is to minimize what overhead exists whenever possible. To advance
2317 the placement of HPC applications on virtual machines, new efforts are emerging
2318 which focus specifically on key hardware now commonplace in supercomputers. By
2319 leveraging new virtualization tools such as IOMMU device passthrough and SR-IOV,
2320 we can now support the such advanced hardware as the latest Nvidia Tesla GPUs [202]
2321 as well as InfiniBand fabrics for high performance networking and I/O [203, 204].

2322 With the advances in hypervisor performance coupled with the newfound avail-
2323 ability of HPC hardware in virtual machines analogous to the most powerful super-
2324 computers used today, we see can see the possibility of a high performance cloud in-

2325 frastructure using virtualization. While our previous efforts in this area have focused
2326 on single-node advancements, it is now imperative to ensure real-world applications
2327 can also operate in distributed environments as found in today’s cluster and cloud
2328 infrastructures.

2329 Efforts to improve power efficiency and performance in data centers has led to
2330 more heterogeneous architectures. That move toward heterogeneity has, in turn, led
2331 to support for heterogeneity in the cloud. For example, Amazon EC2 supports GPU
2332 accelerators in EC2 [205], and OpenStack supports heterogeneity using flavors [206].
2333 These advancements in cloud-level support for heterogeneity combined with better
2334 support for high-performance virtualization makes the use of cloud for HPC much
2335 more feasible for a wider range of applications and platforms.

2336 In this paper we describe background a related work. Then, we describe a heteroge-
2337 neous cloud platform, based on OpenStack. This effort has been under development
2338 at USC/ISI since 2011 [163]. We describe our work towards integrating GPU and
2339 InfiniBand support into OpenStack, and we describe the heterogeneous scheduling
2340 additions that are necessary to support not only attached accelerators, but any cloud
2341 composed of heterogeneous elements.

2342 We then demonstrate running two molecular dynamics simulations, LAMMPS
2343 and HOOMD, in a virtual infrastructure complete with both Kepler GPUs and QDR
2344 InfiniBand. Both HOOMD and LAMMPS are used extensively in some of the world’s
2345 fastest supercomputers and represent example simulations that HPC supports today.
2346 We show that these applications are able to run at near-native speeds within a com-
2347 pletely virtualized environment, demonstrating just small performance impacts that
2348 are usually acceptable by many users. Furthermore, we demonstrate the ability of
2349 such a virtualized environment to support cutting edge software tools such as RDMA
2350 GPUDirect, illustrating how cutting-edge HPC technologies are also possible in a

2351 virtualized environment.

2352 Following these efforts, we hope to ensure upstream infrastructure projects such
2353 as OpenStack [128, 207] are able to make effective and quick use of these features,
2354 allowing users to build private cloud infrastructure to support high performance dis-
2355 tributed computational workloads.

2356 6.2 Background and Related Work

2357 Virtualization technologies and hypervisors have been seen widespread deployment
2358 in support of a vast array of applications. This ranges from public commercial Cloud
2359 deployments such as Amazon EC2 [208, 209], Microsoft Azure [210], and Google's
2360 Cloud Platform [211] to private deployments within colocation facilities, corporate
2361 data centers, and even national scale cyber infrastructure initiatives. All these sup-
2362 port look to support various use cases and applications such as web servers, ACID
2363 and BASE databases, online object storage, and even distributed systems, to name a
2364 few.

2365 The use of virtualization and hypervisors specifically support various HPC so-
2366 lutions has been studied with mixed results. In [169], it is found that there is a
2367 great deal of variance between hypervisors when running various distributed memory
2368 and MPI applications, finding that KVM overall performed well across an array of
2369 HPC benchmarks. Furthermore, some applications may not fit well into default
2370 virtualized environments, such as High Performance Linpack [200]. Other studies
2371 have specifically looked at interconnect performance in virtualization and found the
2372 best-case scenario to be lacking [212] with up to 60% performance penalties with
2373 conventional techniques.

2374 Recently, various CPU architectures have added support for I/O virtualization

2375 mechanisms in the CPU ISA through the use of an I/O memory management unit
2376 (IOMMU). Often, this is referred to as PCI Passthrough, as it enabled devices on the
2377 PCI-Express bus to be passed directly to a specific virtual machine (VM). Specific
2378 hardware implementations include Intel’s VT-d [213], AMD’s IOMMU [214] from
2379 x86_64 architectures, and even more recently ARM System MMU [215]. All of these
2380 implementations effectively look to aid in the usage of DMA-capable hardware to be
2381 used within a specific virtual machine. Using these features, a wide array of hardware
2382 can be utilized directly within VMs and enable fast and efficient computation and
2383 I/O capabilities.

2384 With PCI Passthrough, a PCI device is handed directly to a running (or booting)
2385 VM, thereby relinquishing control of the device within the host entirely. This is
2386 different from typical VM usage where hardware is emulated in the host and used
2387 in a guest VM, such as with bridged ethernet adapters or emulated VGA devices.
2388 Performing PCI Passthrough requires the host to seize the device upon boot using a
2389 specialized driver to effectively block normal driver initialization. In the instance of
2390 the KVM hypervisor, this is done using the *vfio* and *pci_stub* drivers. Then, this driver
2391 relinquishes control to the VM, whereby normal device drivers initiate the hardware
2392 and enable the device for use by the guest OS.

2393 6.2.1 GPU Passthrough

2394 Nvidia GPUs comprise the single most common accelerator in the Nov 2014 Top 500
2395 List [8] and represent an increasing shift towards accelerators for HPC applications.
2396 Historically, GPU usage in a virtualized environment has been difficult, especially
2397 for scientific computation. Various front-end remote API implementations have been
2398 developed to provide CUDA and OpenCL libraries in VMs, which translate library
2399 calls to a back-end or remote GPU. One common use case of this is rCUDA [50], which

2400 provides a front-end CUDA API within a VM or any compute node, and then sends
2401 the calls via Ethernet or InfiniBand to a separate node with 1 or more GPUs. While
2402 this method is valid, it has the drawback of relying on the interconnect itself and the
2403 bandwidth available, which can be especially problematic on Ethernet. Furthermore,
2404 as this method consumes bandwidth, it can leave little remaining for MPI or RDMA
2405 routines, thereby constructing a bottleneck for some MPI+CUDA applications that
2406 depend on inter-process communication.

2407 Recently efforts have been seen to support such GPU accelerators within VMs
2408 using IOMMU technologies, with implementations now available with KVM [202],
2409 Xen [216] and VMWare [217]. These efforts have shown that GPUs can achieve up
2410 to 99% of their bare metal performance when passed to a virtual machine using PCI
2411 Passthrough. VMWare specifically shows how the such PCI Passthrough solutions
2412 perform well and are likely to outperform front-end Remote API solutions such as
2413 rCUDA within a VM [217]. While these works demonstrate PCI Passthrough perfor-
2414 mance across a range of hypervisors and GPUs, they have been limited to investigating
2415 single node performance until now.

2416 **6.2.2 SR-IOV and InfiniBand**

2417 With almost all parallel HPC applications, the interconnect fabric which enables fast
2418 and efficient communication between processors becomes a central requirement to
2419 achieving good performance. Specifically, a high bandwidth link is needed for dis-
2420 tributed processors to share large amounts of data across the system. Furthermore,
2421 low latency becomes equally important for ensuring quick delivery of small mes-
2422 sage communications and resolving large collective barriers within many parallelized
2423 codes. One such interconnect, InfiniBand, has become the most common implemen-
2424 tation used within the Top500 list. However previously InfiniBand was inaccessible

2425 to virtualized environments.

2426 Supporting I/O interconnects in VMs has been aided by Single Root I/O Virtu-
2427 alization (SR-IOV), whereby multiple virtual PCI functions are created in hardware
2428 to represent a single PCI device. These virtual functions (VFs) can then be passed
2429 to a VM and used as by the guest as if it had direct access to that PCI device. SR-
2430 IOV allows for the virtualization and multiplexing to be done within the hardware,
2431 effectively providing higher performance and greater control than software solutions.

2432 SR-IOV has been used in conjunction with Ethernet devices to provide high perfor-
2433 mance 10Gb TCP/IP connectivity within VMs [218], offering near-native bandwidth
2434 and advanced QoS features not easily obtained through emulated Ethernet offerings.
2435 Currently Amazon EC2 offers a high performance VM solution utilizing SR-IOV en-
2436 abled 10Gb Ethernet adapters. While SR-IOV enabled 10Gb Ethernet solutions offers
2437 a big forward in performance, Ethernet still does not offer the high bandwidth or low
2438 latency typically found with InfiniBand solutions.

2439 Recently SR-IOV support for InfiniBand has been added by Mellanox in the Con-
2440 nectX series adapters. Initial evaluation of SR-IOV InfiniBand within KVM VMs
2441 has proven has found point-to-point bandwidth to be near-native, but up to 30%
2442 latency overhead for very small messages [203, 219]. However, even with the noted
2443 overhead, this still signifies up to an order of magnitude difference in latency between
2444 InfiniBand and Ethernet with VMs. Furthermore, advanced configuration of SR-IOV
2445 enabled InfiniBand fabric is taking shape, with recent research showing up to a 30%
2446 reduction in the latency overhead [204]. However, real application performance has
2447 not yet been well understood until now.

2448 **6.2.3 GPUDirect**

2449 NVIDIA’s GPUDirect technology was introduced to reduce the overhead of data
2450 movement across GPUs [220, 221]. GPUDirect supports both networking as well as
2451 peer-to-peer interfaces for single node multi-GPU systems. The most recent imple-
2452 mentation of GPUDirect, version 3, adds support for RDMA over InfiniBand for
2453 Kepler-class GPUs.

2454 The networking component of GPUDirect relies on three key technologies: CUDA
2455 5 (and up), a CUDA-enabled MPI implementation, and a Kepler-class GPU (RDMA
2456 only). Both MVAPICH and OpenMPI support GPUDirect. Support for RDMA over
2457 GPUDirect is enabled by the MPI library, given supported hardware, and does not
2458 depend on application-level changes to a user’s code.

2459 In this paper, our GPUDirect work focuses on GPUDirect v3 for multi-node
2460 RDMA support. We demonstrate scaling for up to 4 nodes connected via QDR
2461 InfiniBand and show that GPUDirect RDMA improves both scalability and overall
2462 performance by approximately 9% at no cost to the end user.

2463 **6.3 A Cloud for High Performance Computing**

2464 With support for GPU Passthrough, SR-IOV, and GPUDirect, we have the building
2465 blocks for a high performance, heterogeneous cloud. In addition, other common
2466 accelerators (e.g. Xeon Phi [222]) have similarly been demonstrated in virtualized
2467 environments. Our vision is of a heterogeneous cloud, supporting both high speed
2468 networking and accelerators for tightly coupled applications.

2469 To this end we have developed a heterogeneous cloud based on OpenStack [128].
2470 In our previous work, we have demonstrated the ability to rapidly provision GPU,
2471 bare metal, and other heterogeneous resources within a single cloud [163]. Building

2472 on this effort we have added support for GPU passthrough to OpenStack as well as
2473 SR-IOV support for both ConnectX-2 and ConnectX-3 Infiniband devices. Mellanox
2474 separately supports an OpenStack InfiniBand networking plugin for OpenStack’s Neu-
2475 tron service [223], however the Mellanox plugin depends on the ConnectX-3 adapter.
2476 Our institutional requirements depend on ConnectX-2 SR-IOV support, requiring
2477 an independent implementation.

2478 OpenStack supports services for networking (Neutron), compute (Nova), identity
2479 (Keystone), storage (Cinder, Swift), and others. Our work focuses entirely on the
2480 compute service.

2481 Scheduling is implemented at two levels: the cloud-level and the node-level. In our
2482 earlier work, we have developed a cloud-level heterogeneous scheduler for OpenStack,
2483 allowing scheduling based on architectures and resources [163]. In this model, the
2484 cloud-level scheduler dispatches jobs to nodes based on resource requirements (e.g.
2485 Kepler GPU) and node-level resource availability.

2486 At the node, a second level of scheduling occurs to ensure that resources are
2487 tracked and not over-committed. Unlike traditional cloud paradigms, devices passed
2488 into VMs cannot be over-committed. We treat devices, whether GPUs or InfiniBand
2489 virtual functions, as schedulable resources. Thus, it is the responsibility of the indi-
2490 vidual node to track resources committed and report availability to the cloud-level
2491 scheduler. For reporting, we piggyback on top of OpenStack’s existing reporting
2492 mechanism to provide a low overhead solution.

2493 6.4 Benchmarks

2494 We selected two molecular dynamics (MD) applications for evaluation in this study:
2495 LAMMPS and HOOMD [224, 225]. These MD simulations are chosen to represent a

2496 subset of advance parallel computation for a number of fundamental reasons:

- 2497 ● MD simulations provide a practical representation of N-Body simulations, which
2498 is one of the major computational *Dwarfs* [226] in parallel and distributed com-
2499 puting.
- 2500 ● MD simulations are one of the most widely deployed applications on large scale
2501 supercomputers today.
- 2502 ● Many MD simulations have a hybrid MPI+CUDA programming model, which
2503 has often become commonplace in HPC as the use of accelerators increases.

2504 As such, we look to LAMMPS and HOOMD to provide a real-world example for
2505 running cutting-edge parallel programs on virtualized infrastructure. While these
2506 applications by no means represent all parallel scientific computing efforts (as justi-
2507 fied by the 13 Dwarfs defined in [226]), we hope these MD simulators offer a more
2508 pragmatic viewpoint than traditional synthetic HPC benchmarks such as High Per-
2509 formance Linpack.

2510 **LAMMPS** The Large-scale Atomic/Molecular Parallel Simulator is a well-understood
2511 highly parallel molecular dynamics simulator. It supports both CPU and GPU-
2512 based workloads. Unlike many simulators, both MD and otherwise, LAMMPS is
2513 heterogeneous. It will use both GPUs and multicore CPUs concurrently. For this
2514 study, this heterogeneous functionality introduces additional load on the host, allow-
2515 ing LAMMPS to utilize all available cores on a given system. Networking in LAMMPS
2516 is accomplished using a typical MPI model. That is, data is copied from the GPU
2517 back to the host and sent over the InfiniBand fabric. No RDMA is used for these
2518 experiments.

2519 **HOOMD-blue** The Highly Optimized Object-oriented Many-particle Dynamics
2520 – Blue Edition is a particle dynamics simulator capable of scaling into the thou-
2521 sands of GPUs. HOOMD supports executing on both CPUs and GPUs. Unlike
2522 LAMMPS, HOOMD is homogeneous and does not support mixing of GPUs and
2523 CPUs. HOOMD supports GPUDirect using a CUDA-enabled MPI. In this paper
2524 we focus on HOOMD’s support for GPUDirect and show its benefits for increasing
2525 cluster sizes.

2526 **6.5 Experimental Setup**

2527 Using two molecular dynamics tools, LAMMPS [224] and HOOMD [225], we demon-
2528 strate a high performance *system*. That is, we combine PCI passthrough for Nvidia
2529 Kepler-class GPUs with QDR Infiniband SR-IOV and show that high performance
2530 molecular dynamics simulations are achievable within a virtualized environment.

2531 For the first time, we also demonstrate Nvidia GPUDirect technology within such
2532 a virtual environment. Thus, we look to not only illustrate that virtual machines
2533 provide a flexible high performance infrastructure for scaling scientific workloads in-
2534 cluding MD simulations, but also that the latest HPC features and programming
2535 environments are also available in this same model.

2536 **6.5.1 Node configuration**

2537 To support the use of Nvidia GPUs and InfiniBand within a VM, specific and exact
2538 host configuration is needed. This node configuration is illustrated in Figure 6.1.
2539 While our implementation is specific to the KVM hypervisor, this setup represents a
2540 design that can be hypervisor agnostic.

2541 Each node in the testbed uses CentOS 6.4 with a 3.13 upstream Linux kernel for

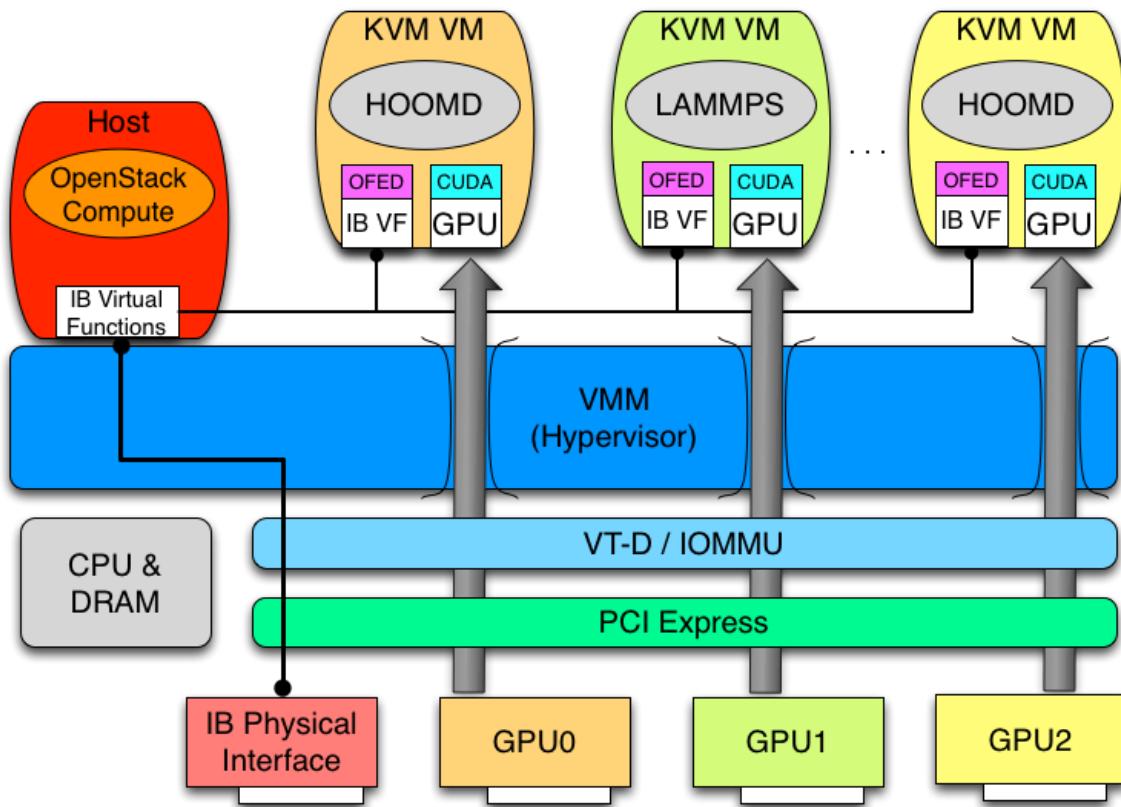


Figure 6.1 Node PCI Passthrough of GPUs and InfiniBand

2542 the host OS, along with the latest KVM hypervisor, QEMU 2.1, and the *vfio* driver.
 2543 Each Guest VM runs CentOS 6.4 with a stock 2.6.32-358.23.2 kernel. A Kepler GPU
 2544 is passed through using PCI Passthrough and directly initiated within the VM via
 2545 the Nvidia 331.20 driver and CUDA release 5.5. While this specific implementation
 2546 used only a single GPU, it is also possible to include as many GPUs as one can fit
 2547 within the PCI Express bus if desired. As the GPU is used by the VM, an on-board
 2548 VGA device was used by the host and a standard Cirrus VGA was emulated in the
 2549 guest OS.

2550 With using SR-IOV, the OFED drivers version 2.1-1.0.0 are used with Mellanox
 2551 ConnectX-3 VPI adapter with firmware 2.31.5050. The host driver initiates 4 VFs,
 2552 one of which is passed through to the VM where the default OFED mlnx_ib drivers

2553 are loaded.

2554 6.5.2 Cluster Configuration

2555 Our test environment is composed of 4 servers each with a single Nvidia Kepler-
2556 class GPU. Two servers are equipped with K20 GPUs, while the other two servers
2557 are equipped with K40 GPUs, demonstrating the potential for a more heterogeneous
2558 deployment. Each server is composed of 2 Intel Xeon E5-2670 CPUs, 48GB of DDR3
2559 memory, and Mellanox ConnectX-3 QDR InfiniBand. CPU sockets and memory are
2560 split evenly between the two NUMA nodes on each system. All InfiniBand adapters
2561 use a single Voltaire 4036 QDR switch with a software subnet manager for IPoIB
2562 functionality.

2563 For these experiments, both the GPUs and InfiniBand adapters are attached to
2564 NUMA node 1 and both the guest VMs and the base system utilized identical software
2565 stacks. Each guest was allocated 20 GB of RAM and a full socket of 8 cores, and
2566 pinned to NUMA node 1 to ensure optimal hardware usage. While all VMs are
2567 capable of login via the InfiniBand IPoIB setup, a 1Gb Ethernet network was used
2568 for all management and login tasks.

2569 For a fair and effective comparison, we also use a native environment without any
2570 virtualization. This native environment employs the same hardware configuration,
2571 and like the Guest OS runs CentOS 6.4 with the stock 2.6.32-358.23.2 kernel.

2572 6.6 Results

2573 In this section, we discuss the performance of both the LAMMPS and HOOMD
2574 molecular dynamics simulation tools when running within a virtualized environment.
2575 Specifically, we scale each application to 32 cores and 4 GPUs, both in a native bare-

2576 metal and virtualized environments. Each application set was run 10 times, with the
 2577 results averaged accordingly.

2578 **6.6.1 LAMMPS**

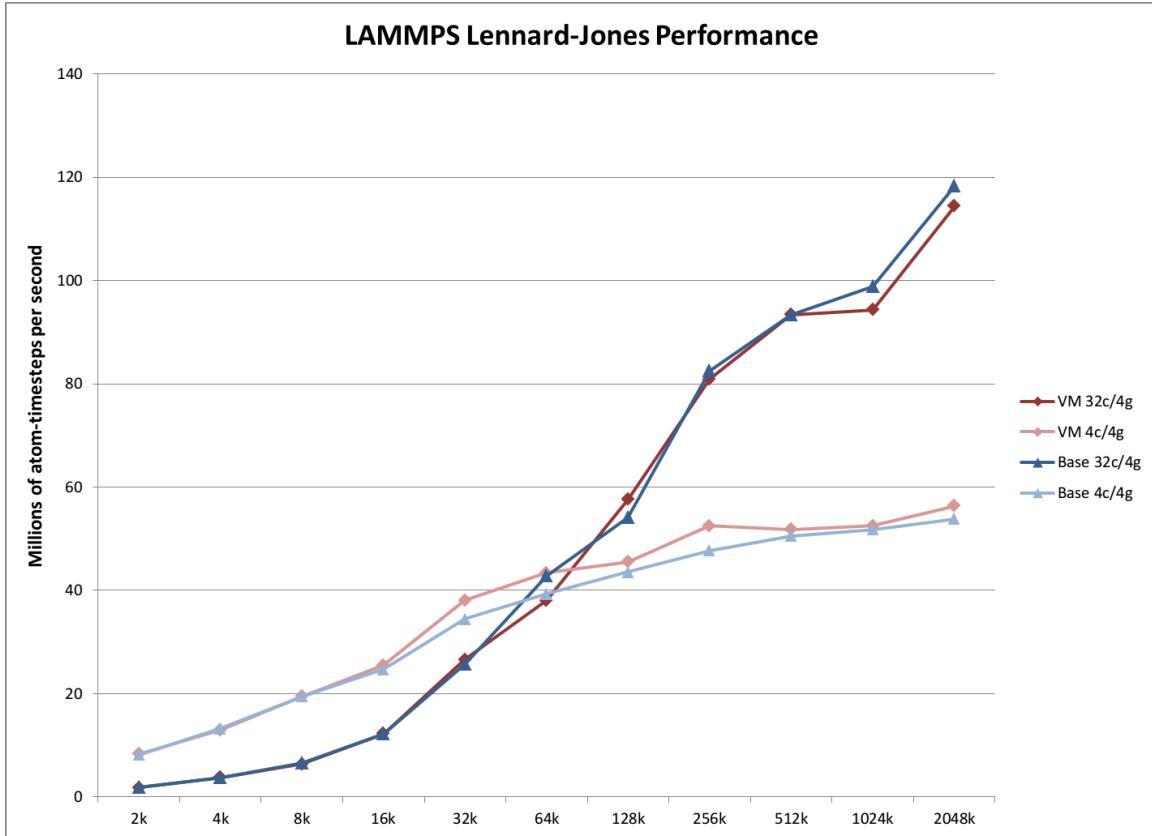


Figure 6.2 LAMMPS LJ Performance

2579 Figure 6.2 shows one of the most common LAMMPS algorithms used; the Lennard-
 2580 Jones potential (LJ). This algorithm is deployed in two main configurations - a 1:1
 2581 core to GPU mapping, and a 8:1 core to GPU mapping. With the LAMMPS GPU
 2582 implementation, a delicate balance between GPUs and CPUs is required to find the
 2583 optimal ratio for fastest computation, however here we just look at the two most
 2584 obvious choices. With small problem sizes, the 1:1 mapping outperforms the more
 2585 complex core deployment, as the problem does not require the additional complexity

2586 provided with multi-core solution. As expected the multi-core configuration quickly
2587 offers better performance for larger problem sizes, achieving roughly twice the perfor-
2588 mance with all 8 available cores. This is largely due to the availability of all 8 cores
2589 to keep the GPU running 100% with continual computation.

2590 The important factor for this manuscript is the relative performance of the virtu-
2591 alized environment. From the results, it is clear the VM solution performs very well
2592 compared to the best-case native deployment. For the multi-core configuration across
2593 all problem sizes, the virtualized deployment averaged 98.5% efficiency compared to
2594 native. The single core per GPU deployment reported better-than native perfor-
2595 mance at 100% native. This is likely due to caching effects, but further investigation
2596 is needed to fully identify this occurrence.

2597 Another common LAMMPS algorithm, the Rhodopsin protein in solvated lipid
2598 bilayer benchmark (Rhodo), was also run with results given in Figure 6.3. As with
2599 the LJ runs, we see the multi-core to GPU configuration resulting in higher computa-
2600 tional performance for the larger problem sizes compared to the single core per GPU
2601 configuration, as expected.

2602 Again, the overhead of the virtualized configuration remains low across all con-
2603 figurations and problem sizes, with an average 96.4% efficiency compared to native.
2604 Interestingly enough, we also see the performance gap decrease as the problem size
2605 increases, with the 512k problem size in yielding 99.3% of native performance. This
2606 finding leads us to extrapolate that a virtualized MPI+CUDA implementation would
2607 scale to a larger computational resource with similar success.

2608 **6.6.2 HOOMD**

2609 In Figure 6.4 we show the performance of a Lennard-Jones liquid simulation with 256K
2610 particles running under HOOMD. HOOMD includes support for CUDA-aware MPI

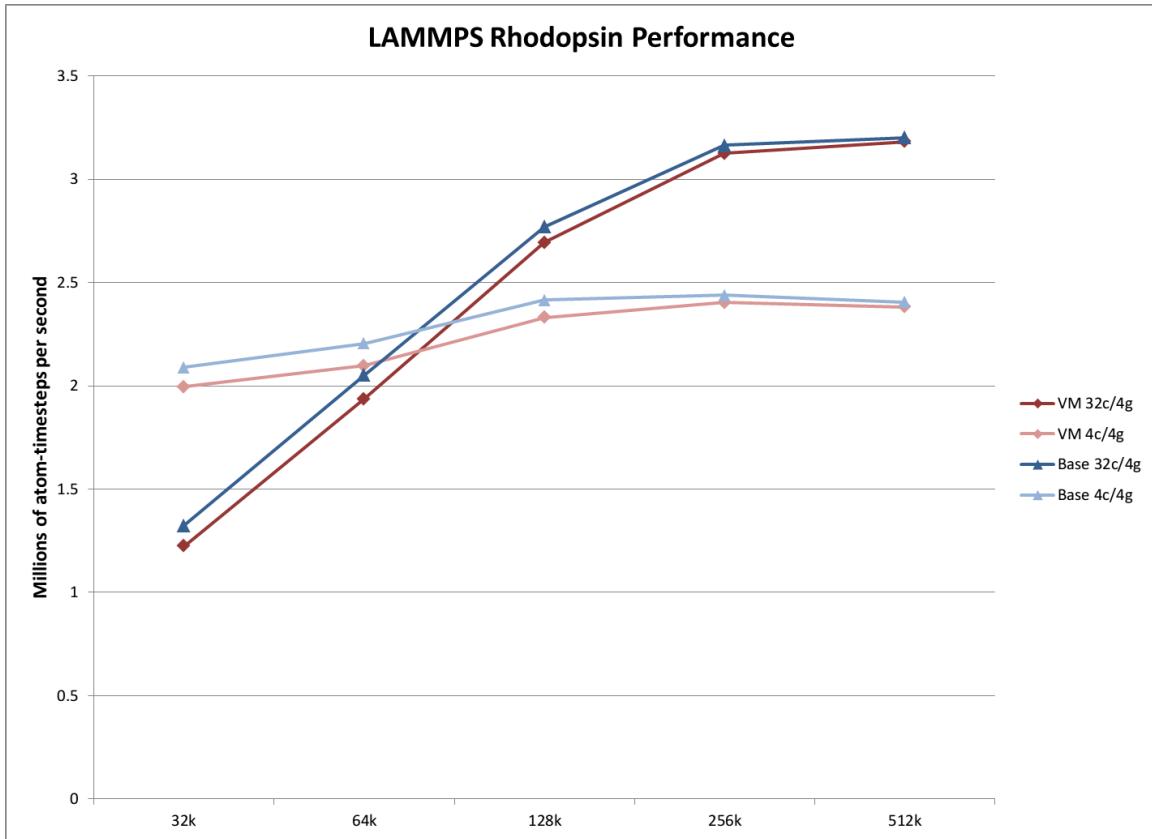


Figure 6.3 LAMMPS RHODO Performance

2611 implementations via GPUDirect. The MVAPICH 2.0 GDR implementation enables
 2612 a further optimization by supporting RDMA for GPUDirect. From Figure 6.4 we
 2613 can see that HOOMD simulations, both with and without GPUDirect, perform very
 2614 near-native. The GPUDirect results at 4 nodes achieve 98.5% of the base system's
 2615 performance. The non-GPUDirect results achieve 98.4% efficiency at 4 nodes. These
 2616 results indicate the virtualized HPC environment is able to support such complex
 2617 workloads. While the effective testbed size is relatively small, it indicates that such
 2618 workloads may scale equally well to hundreds or thousands of nodes.

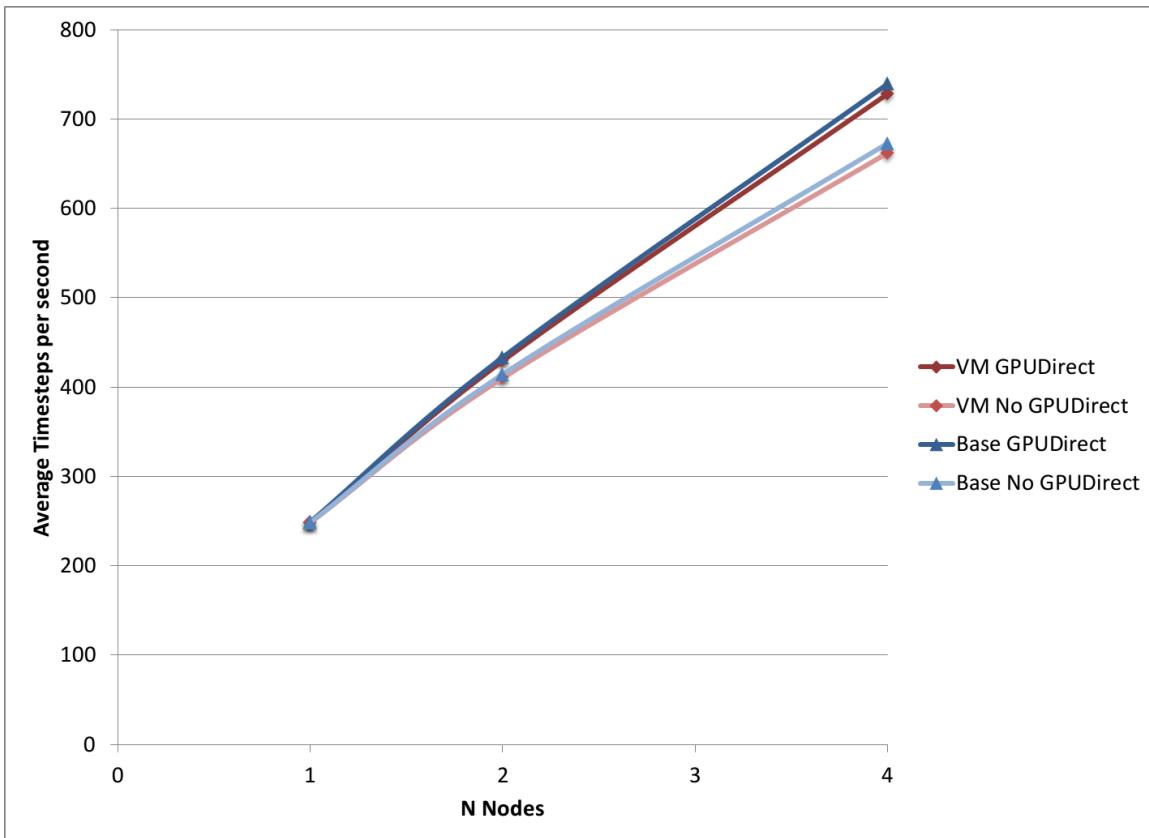


Figure 6.4 HOOMD LJ Performance with 256k Simulation

6.7 Discussion

From the results, we see the potential for running HPC applications in a virtualized environment using GPUs and InfiniBand interconnect fabric. Across all LAMMPS runs with ranging core configurations, we found only a 1.9% overhead between the KVM virtualized environment and native. For HOOMD, we found a similar 1.5% overhead, both with and without GPU Direct. These results go against conventional wisdom that HPC workloads do not work in VMs. In fact ,we show two N-Body type simulations programmed in an MPI+CUDA implementation perform at roughly near-native performance in tuned KVM virtual machines.

With HOOMD, we see how GPUDirect RDMA shows a clear advantage over

2629 the non-GPUDirect implementation, achieving a 9% performance boost in both the
2630 native and virtualized environments. While GPUDirect's performance impact has been
2631 well evaluated previously [220], it is the author's belief that this manuscript represents
2632 the first time GPUDirect has been utilized in a virtualized environment.

2633 Another interesting finding of running LAMMPS and HOOMD in a virtualized
2634 environment is as workload scales from a single node to 32 cores, the overhead does
2635 not increase. These results lend credence to the notion that this solution would also
2636 work for a much larger deployment. Specifically, it would be possible to expand
2637 such computational problems to a larger deployment in FutureGrid [227], Chameleon
2638 Cloud [228], or even the planned NSF Comet machine at SDSC, scheduled to provide
2639 up to 2 Petaflops of computational power. Effectively, these results help support
2640 the theory that a majority of HPC computations can be supported in virtualized
2641 environment with minimal overhead.

2642 6.8 Chapter Summary

2643 With the advent of cloud infrastructure, the ability to run large-scale parallel sci-
2644 entific applications has become possible but limited due to both performance and
2645 hardware availability concerns. In this work we show that advanced HPC-oriented
2646 hardware such as the latest Nvidia GPUs and InfiniBand fabric are now available
2647 within a virtualized infrastructure. Our results find MPI + CUDA applications such
2648 as molecular dynamics simulations run at near-native performance compared to tra-
2649 ditional non-virtualized HPC infrastructure, with just an averaged 1.9% and 1.5%
2650 overhead for LAMMPS and HOOMD, respectively. Moving forward, we show the
2651 utility of GPUDirect RDMA for the first time in a cloud environment with HOOMD.
2652 Effectively, we look to pave the way for large-scale virtualized cloud Infrastructure

2653 to support a wide array of advanced scientific computation commonly found running
2654 on many supercomputers today. Our efforts leverage these technologies and provide
2655 them in an open source Infrastructure-as-a-Service framework using OpenStack.

2656 **Chapter 7**

2657 **Virtualization advancements to**

2658 **support HPC applications**

2659 Many of the previous chapters have focused on identifying and decreasing the per-
2660 formance gap that exists with running HPC workloads in virtualized infrastructure,
2661 as compared to native HPC environments without virtualization. While this is a
2662 significant technical challenge to overcome, the use of virtualization itself has the
2663 potential to offer additional benefits to HPC infrastructure. In this chapter, we look
2664 to identify research, design, and future implementation of advanced virtualization
2665 techniques to enable a new class of HPC infrastructure with added performance and
2666 features that has yet to be realized. It may be possible for these advancements, once
2667 implemented to have an impact not only on cloud infrastructure, but also a dedicated
2668 HPC environment for added usability and performance.

2669 7.1 Memory Page Table Optimizations

2670 As we've seen both in Chapter 3 as well as in other supported literature [43], virtual-
2671 ization of memory structures becomes a point of contention and potential overhead.
2672 This is often due to the extensive effort a hypervisor has to perform in order to
2673 translate memory addresses from guest-virtual addresses to host-virtual addresses,
2674 and then again to machine-physical addresses. Historically, this was handled using
2675 Shadow Page tables [229], which eliminate the need for emulation of physical memory
2676 inside the VM by creating a page table mapping from guest virtual to machine mem-
2677 ory. However, these page tables are not walkable by hardware such as a transition
2678 lookaside buffer (TLB), and as such guest OS page tables require updating of the
2679 shadow page table. This can be costly not only in the additional management, but
2680 also notably by the cost of VMexit and VMentry calls.

2681 Recently, Intel and AMD have implemented Extended Page Tables (EPT) and
2682 nested paging, respectively. With EPT, support is added to allow the TLB hardware
2683 to keep track of both guest pages and hypervisor pages concurrently, effectively re-
2684 moving the need for shadow page table. The downside of EPT and nested paging
2685 occurs when there is a TLB miss (the page isn't in the TLB), and as such each miss
2686 requires a walk through each VMM nested paging, effectively creating TLB miss cost
2687 of 16 table walks (instead of just 3-4) for 4k pages. While many applications find
2688 this TLB miss additional cost less much less than that of managing shadow page
2689 tables, it still can lead to a significant gap in performance between non-virtualized
2690 applications, especially as VM count or an application's memory footprint increase.

2691 One potential way to decrease the chance of a TLB miss (and therefore the cost
2692 of a miss) is by using a larger page size. By default, x86 hardware uses 4k pages
2693 sizes, but newer hardware can support 2M and 1G page sizes as well, effectively

2694 named *transparent huge pages* or THP. Using the KVM hypervisor with transparent
2695 hugepages enabled, we can create guest VMs backed entirely 2M hugepages [197]. We
2696 can also enable transparant hugepage support within the guest as well, to have the
2697 entire guest OS (including kernel and modules) using 2M pages.

2698 The result of THP-enabled guest VMs can be significant. With huge pages on Intel
2699 x86 CPUs with EPT, there exists an entirely separate TLB for hugepages as well. This
2700 will naturally alleviate TLB pressure and therefore reduce TLB contention between
2701 guest and host operating systems. Because 2M pages provide larger addressable
2702 memory, the size of the page tables themselves are also decreased. Effectively, this
2703 reduces the TLB miss cost from 4 to 3 page table walks, which when handling a VM
2704 TLB miss, then requires only 15 walks to the default 24. This in effect can have a
2705 significant improvement in VM performance, as hypothesized in [197].

2706 To evaluate the effect of 2M transparent huge pages on guest performance, we
2707 leverage the same KVM setup in Chapter 5 on the Bespin hardware. Specifically, THP
2708 is switched both in the host as well as the guest OS kernels, and the same LibSVM
2709 application using GPUs is re-run. The libSVM GPU application can have significant
2710 memory requirements, as large chunks of the support vector machine datasets are
2711 stored in CPU memory and tranferred in a sudo-random order to and from GPU
2712 memory, making it an ideal application to use for evaluating THP.

2713 The results of running libSVM in a THP-enabled VM, a VM with no THP, and
2714 natively without virtualization are displayed in Figure 7.1. Comparing first just
2715 the KVM results without THP to the native solution, we can see the impact of
2716 THP the overal application runtime, especially at larger problem sizes (6000 training
2717 sets). However, when TLB is enabled in the guest and host, we actually see the
2718 KVM VM solution *outperform* the native solution. This is because guest privileged
2719 OS memory used to buffer to/from GPU memory is backed by 2M pages, instead



Figure 7.1 Transparent Huge Pages with KVM

of the normal 4k pages as in the native solution. The result is less TLB misses during application runtime compared to 4k TLB misses in a native solution, resulting in improved performance. While this is likely a special case for THP usage with the libSVM application, the fact that a VM can even occasionally outperform a native runtime is a noteworthy accomplishment. This also underscores the need for careful tuning and best-practices for hypervisors when supporting advanced scientific workloads.

2727 7.2 Live Migration Mechanisms

2728 Migration of VMs represents one of the fundamental advantages to virtualization, and
2729 also one of the greatest challenges to efficiency. With VM migration, the complete VM
2730 state is copied from a source to a unallocated destination host, where disk, memory,
2731 and network connections are kept intact. For disk continuity, a distributed and/or
2732 shared filesystem is utilized, most commonly but not exclusively NFS, where both
2733 the source and destination hosts have access to the VM disk. Network continuity is
2734 preserved so long as the destination guest is within the same LAN and generates an
2735 unsolicited ARP reply to maintain the original IP after migration. VM vCPU states
2736 and machine states are recorded from the source and quickly sent to the destination
2737 host when the VM is paused. For live migration, the source VM is paused only
2738 after all state and memory contents are copied to the destination. The last of the
2739 dirtied memory pages are copied over, and the newly formed destination VM is then
2740 un-paused. This pause and transfer time represents the entirety of a VM downtime
2741 during live migration, and is often at or under 100 milliseconds across commodity
2742 Ethernet networks (given a low memory utilization).

2743 VM memory transfers are often the main performance consideration for overhead
2744 during live migration. This is not only due to the potentially large amount of memory
2745 to be sent across the network, but the veracity at which the memory is changed. This
2746 is defined directly by the amount of main memory allocated (or in use) by the source
2747 VM. However, as a VM's memory is sent, the VM is still running and therefore memory
2748 pages can be dirtied, creating the need for any written page to be retransmitted.
2749 Given a small network and a memory bound processes running within a VM, this
2750 can be an infinitely long process of page dirtying. Many live migration strategies
2751 provide an iterative timeout mechanism to avoid this infinite state, but this will lead

2752 to increased downtime during migration.

2753 The copying of memory pages for live migration can take multiple implemen-
2754 tations. Three common options are summarized below:

- 2755 ● Pre-copy Migration: All memory pages are transmitted to the destination before
2756 the VM is paused. The hypervisor will note and track all dirtied memory pages,
2757 and retransmit those pages in iterative rounds. The rounds end when either no
2758 dirtied pages exist or a max iteration count has been reached. The VM state
2759 is then transmitted and resumed on the destination. This method was the first
2760 live migration technique used in [230].
- 2761 ● Post-copy Migration: The Vm state is immediately paused and sent to the
2762 destination hypervisor, and immediately resumed. If the new destination VM
2763 generates a page fault, the VM is paused, and faulted pages are transmitted
2764 across the network on demand from the source and the VM resumed. This
2765 methodology is proposed for use in the Xen hypervisor by Hines et al [231].
- 2766 ● Hybrid-copy Migration: Provides a compromise solution to memory paging.
2767 First, single copy of the VM memory pages, or a subset of known-necessary
2768 memory pages are sent to the destination. Then the source VM is paused,
2769 its VM state sent to the destination, and resumed on the destination. Known
2770 dirtied source pages, or missing pages are then copied to the destination upon a
2771 triggered page fault utilizing the same mechanism as post-copy migration. An
2772 example of hybrid migration can be found via Lu et al [232].

2773 While pre-copy migration is the traditional and most used live migration tech-
2774 nique, there are opportunities about to implement other migration techniques to
2775 advance the mobility of distributed computing in high performance virtual clusters
2776 with virtualization.

2777 7.2.1 RDMA-enabled VM Migration

2778 Currently, most live migration in production environments occur over TCP/IP con-
2779 nections, largely due to the prevalence of commodity Ethernet connections within
2780 cloud infrastructure. However even if RDMA-capable interconnects are available in
2781 such infrastructure as described with InfiniBand in Chapter 6, live migration still
2782 usually occurs over TCP/IP. For the case of InfiniBand, this is via IP over InfiniBand
2783 (IPoIB) [233], which can be an inefficient use of the interconnect bandwidth and add
2784 extra latency [234].

2785 The time it takes to migrate the memory contents from a source to destination
2786 VM can be significantly decreased by using an RDMA based mechanisms. Huang et
2787 al first provided a proposed pre-copy method for RDMA-based migration using the
2788 Xen hypervisor [235]. Specifically, they found a 80% decrease in migration time with
2789 RDMAwrite operations. This speedup is largely due to the removal of overhead nec-
2790 essary for processing TCP/IP communications, largely in CPU utilization for copying
2791 buffers, packet processing, and the included context switch overhead when competing
2792 for resources in a CPU-bound application (which are common in HPC environments).

2793 The live migration algorithm proposed in [235] uses the standard pre-copy mecha-
2794 nism that sends the entire memory contents across the network as RDMA operations,
2795 then iteratively copies dirtied pages before switching the running states. As discussed
2796 in the previous section, a post-copy migration strategy may have benefits for quick
2797 VM migration in high performance virtual clusters, or even for VM cloning as de-
2798 scribed later in Section 7.3. Furthermore, efforts in Chapter 3 have found that the
2799 Xen hypervisor is not best suited for HPC workloads. As such, there is a need to
2800 redefine the use of RDMA for VM migration using a hybrid post-copy mechanism in
2801 a high performance hypervisor.

- 2802 ● Transfer initial CPU state, registers
- 2803 ● Start the page table pages translation process: MFN to PFN and use copy-base
2804 approach
- 2805 ● Concurrently, allocate remote memory on destination VM.
- 2806 ● Set up other machine state settings in destination
- 2807 ● Start destination VM, pause source VM.
- 2808 ● Initiate RDMAwrite of entire memory contents from source to destination
- 2809 ● As page faults occur in destination VM, catch faults and perform RDMAread
2810 requesting pages.

2811 Currently efforts are underway to provide post-copy live migration in KVM/QEMU,
2812 using the `migrate_set_capability x-postcopy-ram` on mode within KVM [236].
2813 This method uses the Linux `userfaultfd` kernel mechanisms from a kernel 4.3 or newer.
2814 At the start, all memory blocks are registered as `userfaultfd` so all faults cause the
2815 running thread to pause. In kernel space, the missing page is requested from the
2816 sender, which prioritized over other pages being sent and is returned and mapped
2817 to the destination guest memory space and the thread or process is unpause. This
2818 mechanism operates asynchronously, so that multiple outstanding page faults will not
2819 stop other executables within the VM. As such, the `xpostcopy-ram` KVM extention
2820 has been identified as an ideal place to implement an RDMA based implementation.

2821 To provide RDMA functionality, the InfiniBand interconnect could first be used
2822 as a proof-of-concept. This choice is due to InfiniBand's rise in popularity, increased
2823 prevalence in virtualized systems with SR-IOV as noted in Chapter 6, and RMDA
2824 functionality. However, other interconnect options exist that may be better suited

2825 for enhanced functionality and performance, such as Intel’s new Omnipath [237], or
2826 an Ethernet solution such as RoCE [238], to name a few alternatives. While RDMA
2827 can be managed through the kernel level, it is often used in user-level APIs, such
2828 as MPI, or in the case of InfiniBand, ibVerbs. However, it may be best to select a
2829 interconnect-agnostic middleware for RDMA, such as Photon [239].

2830 RDMA semantics can be used to either read or write contents of remote mem-
2831 ory, in this case VM guest memory pages. However before such operations can take
2832 place, the target side of the operation must register the remote memory buffers and
2833 send the remote key to the initiator, effectively providing the DMA addressing to
2834 be used. Beyond the increased bandwidth and decreased latency benefits provided
2835 by an advanced interconnect with InfiniBand, RDMA also allows VM memory to be
2836 sent without involving the OS. This is due to InfiniBand’s zero-copy, kernel bypass
2837 mechanisms, and utilizing asynchronous operations. This keeps the CPU load down,
2838 and focused on the computation at hand rather than the I/O transfer, as it often has
2839 to when utilizing a TCP/IP stack.

2840 Selecting the proper RDMA based communication operations to use can make a
2841 notable difference in the overall performance of the migration. Some RDMA oper-
2842 ations are largely a one-sided movement, which can have performance impact and
2843 should be carefully considered in using for post-copy live migration. The RDMAread
2844 operation requires more effort at the destination host, while RDMA write operation
2845 puts burden on the source host. One way to determine which method is best is by
2846 evaluating both source and sink CPU loads. However this method likely will not
2847 be as obvious when we consider VMs will be not be running in an over-subscribed
2848 virtualized environment, but rather a highly optimized one.

2849 When the destination VM page faults, it is necessary to retrieve the page with
2850 as little latency as possible, as the running thread is paused. Using the method

2851 developed in [236], the *userfaultfd* will trigger an RDMAread to retreive the missing
2852 page. RDMAread requires no interaction from the source VM, as RDMAread is one
2853 sided and the memory buffers have been passed in the setup phase. To further enable
2854 quick return time for the missing page, enabling polling on the source host may further
2855 reduce latency, however verification of this will be necessary. When the RDMAread
2856 operation is finished, the running thread will resume and execution will continue.

2857 **7.2.2 Moving the Compute to the Data**

2858 While pre-copy migration is dominant in the live migration techniques of nearly every
2859 mainstream hypervisor today, it is proposed that for HPC applications, post-copy
2860 migration could provide some key new advances. One particular use case would be to
2861 send a lightweight VM to directly act on a large or set of large datasets, and return a
2862 slimmed down result set. This would reduce the requirement of transmitting the data
2863 across a network entirely, and potentially speed up data access latency drastically, as
2864 on result information in the form of memory pages and VM state are transmitted.

2865 With post-copy migration, one could move the computation at hand close to a
2866 data source in significantly less time than full pre-copy live migration. This data
2867 source, and lightweight VM sink, could potentially be something similar to a Burst
2868 Buffer system [240, 241] or a classic HPC I/O node with a distribute filesystem such
2869 as Lustre or GPFS [242]. This data source could even potentially be a remote sci-
2870 entific instrument completely separate from the HPC infrastructure itself, especially
2871 if the network at hand is capable of RoCE [238] or iWARP [243]. A VM would ini-
2872 tiate post-copy live migration, transmitting only the necessary cpu state, registers,
2873 and non-paged memory rather than the full VM memory state. Once migrated, the
2874 VM could connect to a (now local) I/O or storage device, accessing data fast and
2875 perofrming the necessary calculations. The VM could potentially even forgo the copy

2876 of the majority of its memory. During this time, only the necessary memory pages
2877 required to complete the immediate calculation would generate a fault and trigger
2878 their transmission from the source. The VM could even return the result (rather
2879 than a very large dataset) to the original source VM.

2880 This post-copy live migration technique for remote data computation avoids the
2881 cost of spawning a whole new job and/or process with associated running parameters,
2882 as well as the extremely high cost of a full VM live migration using the pre-copy
2883 method. However, one potential downside of post-copy live migration would be the
2884 non-deterministic runtime, as it would be unknown how much remote memory paging
2885 would be required. This could lead to more time spent with the destination VM in
2886 a paused state awaiting remote memory pages, rather than if the entire VM memory
2887 contents were transmitted completely. Careful analysis of memory usage, or a hybrid
2888 copy method based on predetermined memory sections could help overcome this issue,
2889 but require a more advanced migration architecture.

2890 7.3 Fast VM Cloning

2891 In many distributed system environments, concurrency is achieved through the use
2892 of homogeneous compute nodes that handle the bulk of the computational load in
2893 parallel. This can take many forms, including master/slave configurations, or even a
2894 traditional HPC cluster with identical compute nodes, often used to support Single
2895 Process Multiple Data (SPMD) computational models [244]. With high performance
2896 virtual clusters, there is a need to efficiently deploy and manage near identical virtual
2897 machines for distributed computation.

2898 In Snowflock [245,246], the notion of VM cloning is given. Specifically, Lagar et al.
2899 define the notion of VM Fork, where VMs are treated similar to a fork system call for

2900 processes. This process is conceptually similar to VM migration, with the exception
2901 that the source VM is not destroyed after the migration. Starting with a master
2902 VM, an impromptu cluster can be created across a network using the Xen hypervisor.
2903 Snowflock specifically uses Multicast to linearly scale out VM creation to many hosts,
2904 only coping a minimal state and then remotely coping memory pages when requested.
2905 This fetched memory on-demand is similar in principal to the post-copy live migration
2906 technique described in the previous section. This is further augmented with blocktap-
2907 based virtual disks with Copy-on-Write (CoW) functionally, delivering CoW slices for
2908 each child VM.

2909 While Snowflock provides an excellent framework for VM cloning, it is not suitable
2910 for the current implementation. First, it uses Xen, which in previous research de-
2911 scribed in Chapter 3 has found to have limited performance for HPC workloads [169].
2912 Second, SnowFlock is designed for Ethernet and IP based networks, which have signif-
2913 icantly higher latency and lower bandwidth when compared to InfiniBand solutions.
2914 Developing analogous mechanisms with a high performance hypervisor such as KVM
2915 or Palacios [27] to use RDMA for VM memory paging. RDMA could even be utilized
2916 in conjunction with CoW disc blocks.

2917 Potential VM Cloning RDMA mechanism:

- 2918 ● Prepare parent VM state, including registers and info.
- 2919 ● Prepare CoW disk images.
- 2920 ● Create large buffers for all VM memory on child hosts.
- 2921 ● Send vmstate via RDMAwrite or IB Send to N child nodes, where N is the clone
2922 size.
- 2923 ● Resume/start child VMs in tandem.

2924 ● Parent VM set up RDMA multicast (unreliable connection) to send memory
2925 pages to all childs efficiently

2926 ● Child clone VM joins RDMA mlticast.

2927 ● If child page faults, perform RDMAread operation for specific page.

2928 This method allows for efficient cloning of VMs based on a running parent VM.

2929 First, the VM state and images are copied to all child nodes to recieve the VM,
2930 and blank memory is allocated. Then each child VM is started, and page faults are
2931 handled via RDMA read. This will result in an initial slowdown, but as pages are
2932 received the child VMs will start to run. The rest of the memory is eventually sent
2933 via multicast to all VMs simultaniously. As multicast is unreliable, delivery failure
2934 will just trigger a page fault and subsequent page sending via RDMA. It allows for
2935 direct page fault handling, while allowing child VMs to start and run immediately.
2936 As RDMA multicast mechanisms are relatively questionable, more investigation is
2937 needed to evaluate the feasibility of this situation.

2938 It is expected for post-copy VM cloning to also work most efficiently if used in
2939 conjunction with guest VMs backed with hugepages. Transferring memory in 2M
2940 chunks will more effectively utilize network bandwdith by eliminating send/receive
2941 overhead. This also will hide the latency found in SR-IOV enabled interconnects
2942 for small messages. Furthermore, it will reduce the overhead of page fault handling
2943 mechanisms, as less overall pages will fault and be transferred. While hugepages are
2944 expected to improve VM cloning efficency, emperical testing will still be necessary to
2945 properly evaluate their viability.

2946 While a VM fork mechanism leveraging post-copy live migration in KVM will
2947 quickly spool up cloned VMs, the eventual memory transfer will eventually fail to scale
2948 past the network's capacity. This could happen if hundreds or thousands of child clone

2949 VMs are being started simultaneously, as is likely in large scale deployments. As such,
2950 a higherarchical distribution may be necessary. One possible method for this would be
2951 a two-stage cloning mechanism, where child VMs are cloned one to each cabinet, and
2952 the entire memory contents copied using the pre-copy migration mechanism. From
2953 there, further cloning occurs to deploy many cloned VMs to individual nodes within
2954 the cabinet. Organization of mid-tier cloned VMs would likely be determined based
2955 on RDMA fabric configurations within cabinets, as this is a network-bound process.

2956 7.4 Virtual Cluster Scheduling

2957 Historically, cloud infrastructure has taken a simplistic approach when it comes to VM
2958 and workload scheduling. Often, a simple round-robin or greedy [247] scheduling
2959 algorithm is appropriated. With round robin scheduling, a simple list of host ma-
2960 chines are used and iterated over as VM requests are made. This essentially scatters
2961 the VMs without regard to their locality, and oversubscription becomes commonplace
2962 regardless of the number of requested VMs or their interconnection. A greedy algo-
2963 rithm can help keep spacial locality, but focuses specifically on over-subscription as
2964 well to help consolidate VM allocations. While this oversubscription aspect is very
2965 good for public cloud providers such as Amazon EC2, it becomes counterproductive
2966 for high performance virtual clusters.

2967 With high performance virtual clusters, a number of VM instances are needed
2968 that can gain near-native performance. Furthermore, these VMs will be running
2969 tightly coupled applications, and as such need to be allocated in a way in which
2970 communication latency is minimized and bandwidth is maximized for all VMs. This
2971 will help insure the entire virtual clusters can perform optimally. However, given
2972 off-the-shelf private cloud providers or, worse still, public cloud infrasturcture, these

2973 requirements are at best opaque to the user, and at worst extremely suboptimal.

2974 One way in which high performance virtual clusters can operate efficiently is
 2975 by defining specific instance types, or *flavors* within OpenStack, that define what
 2976 resources a VM has allocated. Given previous research on the NUMA effects of
 2977 VMs [248], these specialized flavors should be defined to fit within a NUMA socket.
 2978 Furthermore, CPU pinning should be used to specifically keep the VM within the
 2979 NUMA socket itself. Using libvirt, a common API utilized in many cloud infrastruc-
 2980 ture deployments (including OpenStack), we can specify CPU pinning directly in the
 2981 XML config.

```
2982 <cputune>
2983   <vcpupin vcpu="0" cpuset="0"/>
2984   <vcpupin vcpu="1" cpuset="1"/>
2985   <vcpupin vcpu="2" cpuset="4"/>
2986   <vcpupin vcpu="3" cpuset="5"/>
2987   <vcpupin vcpu="4" cpuset="6"/>
2988   <emulatorpin cpuset="2"/>
2989 </cputune>
```

<input type="checkbox"/>	Key	Value	Actions
<input type="checkbox"/>	pci_passthrough:labels	["gpu", "infiniband"]	<button>Edit</button> <button>More ▾</button>

Displaying 1 item

Figure 7.2 Adding extra specs to a VM flavor in OpenStack

2990 With OpenStack, this NUMA configuration can be executed through the KVM

2991 Nova plugin and a scheduling filter, which defines how to place VMs effectively. Fur-
2992 themore, the instance flavor can also specify the addition of

2993 `instance_type_extra_specs`

2994 within Nova, whereby specialized hardware such as GPUs and InfiniBand intercon-
2995 nects (as described in Chapters 5 and 6) can be passed through to the VMs directly.

2996 Once defined, the same specialized high performance flavors in OpenStack simply have
2997 to add the labels (such as 'gpu' or 'infiniband') to the flavor to gain the requested
2998 hardware, as illustrated in Figure 7.2 with OpenStack Horizon's UI interface. The
2999 implementation put forth in the OpenStack Havana build has since been upgraded by
3000 Intel with Sr-IOV support and additional scheduling filter additions, and is available
3001 in the latest OpenStack releases [249].

```
3002 pci_passthrough_devices=[{"label":"gpu","address":"0000:08:00.0"},  
3003  {"label":"infiniband","address":"0000:21:00.0"}]  
3004 instance_type_extra_specs={'pci_passthrough:labels': '["gpu"]'}  
3005 instance_type_extra_specs={'pci_passthrough:labels': '["infiniband"]'}
```

3006 To provide a space for high performance virtual clusters, we need a scheduling
3007 mechanism within a cloud infrastructure to support the effective and proper plate-
3008 ment beyond controlling for NUMA characteristics. While there are many effective
3009 scheduling algorithms for workload placement within an environment, a Proximity
3010 Scheduler [37], as defined in OpenStack, may work well. Specifically, one could either
3011 define or compute the underlying cloud infrastructure network topology. This could
3012 be as simple as a YAML file that defines an underlying InfiniBand interconnect 2-1 Fat
3013 Tree topology, or a more complex solution that utilizes network performance metrics
3014 to measure bandwidth and latency between disjoint nodes to build a weighted prox-
3015 mity graph. As it is possible that such network parameters could change due to

3016 other usage or to reconfiguration, a method of periodically monitoring and updating
3017 this proximity network using measurement tools [250] may also help keep an effec-
3018 tive proximity metric between hosts. With a metric, one can then apply a proximity
3019 scheduler to handle high performance virtual cluster allocation requests effectively
3020 and in a way that will be far more optimal than a simple round robin scheduling
3021 mechanism. The OpenStack private cloud IaaS framework is proposed for this ef-
3022 fort, however, further development is needed to bring such a scheidlung mechanism
3023 to fruition.

3024 The use of service level agreements (SLAs) within cloud infrastructure allocation is
3025 a well studied aspect [251]. It is also possible that certain SLAs could be incorporated
3026 into a private cloud infrasturcture such as OpenStack to simultaniously gaurantee
3027 performance for virtual clusters with the above defined methods, while concurrently
3028 offering "classical" VM workload scheduling for HTC, big data, or other cloud usage
3029 models. This would provide the same user experience yet support diverse workloads
3030 and performance expectations as defined by a given SLA

3031 7.5 Chapter Summary

3032 In summary, we expect the combination of transparant huge pages, an RDMA-capable
3033 interconnect multiplexed in hardware for use by both guest and hosts, a high perfor-
3034 mant hypervisor, and post-copy migration and cloning mechanisms to enable a novel
3035 architecture for high perofrmance virtual clusters. These mechanisms, if implemented
3036 and combined with an efficient OS, such as Kitten [252], could help enable a new run-
3037 time system to support extreme scale distirbuted memory computations with in-situ
3038 analysis [253, 254] for large scale scientific systems.

3039 **Chapter 8**

3040 **Conclusion**

3041 With the advent of virtualization and the availability of virtual machines through
3042 cloud infrastructure, a paradigm shift in distributed systems has occurred. Many
3043 services and applications once deployed on workstations, private servers, personal
3044 computers, and even some supercomputers have migrated to a cloud infrastructure.
3045 The reasons for this change are vast, however these reasons are not enough to support
3046 all computational challenges within such a virtualized infrastructure.

3047 The use of tightly coupled, distributed memory applications common in High
3048 Performance Computing communities, has seen a number of problems and compli-
3049 cations when deployed in virtualized infrastructure. While the reasons for this can
3050 be vast, many challenges stem from the performance impact and overhead associated
3051 with virtualization, along with a lack of hardware necessary to support such concur-
3052 rent environments. This dissertation looks to evaluate virtualization for supporting
3053 mid-tier scientific HPC applications and provide potential solutions to these issues.

3054 From the beginning, this dissertation proposes the advent of high performance
3055 virtual clusters to support a wide array of scientific computation, including mid-tier
3056 HPC applications, as well as a framework for building such an environment. This

3057 framework aims at identifying virtualization overhead and finding solutions and best
3058 practices with performant hypervisors, providing support for advanced accelerators
3059 and interconnects to enable new class of applications, and evaluating potential meth-
3060 ods using benchmarks and real-world applications.

3061 Chapter 2 studied the related research necessary for defining not only the context
3062 for virtualization and cloud computing, but also virtual clusters, and their history
3063 through supercomputing. Chapter 3 looked to study the applicability of various hy-
3064 pervisors for supporting common HPC workloads through the use of benchmarks from
3065 a single-node aspect. This found challenges and some solutions to these workloads,
3066 and identified missing gaps that exist.

3067 Chapter 4 started the investigation of the utility of GPUs to support mid-tier
3068 scientific applications using the Xen hypervisor. This chapter provided a proof-of-
3069 concept that with proper configuration and utilizing the latest in hardware support,
3070 GPU passthrough was possible and a viable model for supporting CUDA-enabled
3071 applications, a fast-growing application set. Chapter 5 provides an in-depth com-
3072 parison of multiple hypervisors using the SHOC GPU benchmark suite, as well as
3073 GPU-enabled HPC applications. Here we discover our KVM implementation per-
3074 forms at near-native speeds and allows for effective GPU utilization.

3075 Chapter 6 takes the lessons learned with KVM in GPU passthrough and adds in
3076 SR-IOV InfiniBand support, a critical tool for supporting tightly coupled distributed
3077 memory applications, to build a small virtual cluster. This environment supports
3078 two class-leading Molecular Dynamics simulations, LAMMPS and HOOMD-blue, and
3079 shows how both applications can not only perform at near-native speeds, but also
3080 leverage the latest HPC technologies such as GPUDirect for efficient GPU-to-GPU
3081 communication. This framework is also enveloped in an OpenStack environment.

3082 Chapter 7 is an introspective look at other advancements that can be made in

3083 virtualization to support high performance virtual clusters. Specifically, this chap-
3084 ter details the utility of virtual clusters backed with hugepages, added support for
3085 specialized live migration techniques leveraging high speed RDMA-capable intercon-
3086 nects, VM cloning for fast deployment of virtual clusters themselves, and scheduling
3087 considerations for integration of high performance virtual clusters in OpenStack.

3088 *TODO: Answer the research question, can we support HPC with virtual clusters?*
3089 *How could this help with big data convergence?*

3090 8.1 Impact

3091 TBD

3092 Bibliography

- 3093 [1] B. Alexander, “Web 2.0: A New Wave of Innovation for Teaching and Learn-
3094 ing?” *Learning*, vol. 41, no. 2, pp. 32–44, 2006.
- 3095 [2] R. Buyya, C. S. Yeo, and S. Venugopal, “Market-oriented cloud computing:
3096 Vision, hype, and reality for delivering it services as computing utilities,” in
3097 *Proceedings of the 10th IEEE International Conference on High Performance*
3098 *Computing and Communications (HPCC-08, IEEE CS Press, Los Alamitos,*
3099 *CA, USA)*, 2008, pp. 5–13.
- 3100 [3] I. Foster, Y. Zhao, I. Raicu, and S. Lu, “Cloud Computing and Grid Comput-
3101 ing 360-Degree Compared,” in *Grid Computing Environments Workshop, 2008.*
3102 *GCE’08*, 2008, pp. 1–10.
- 3103 [4] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski,
3104 G. Lee, D. Patterson, A. Rabkin, I. Stoica, and M. Zaharia, “Above
3105 the clouds: A berkeley view of cloud computing,” University of
3106 California at Berkeley, Tech. Rep., February 2009. [Online]. Available:
3107 <http://berkeleyclouds.blogspot.com/2009/02/above-clouds-released.html>
- 3108 [5] T. Kuhn, *The structure of scientific revolutions*. University of Chicago press
3109 Chicago, 1970.

- 3110 [6] T. Sterling, *Beowulf cluster computing with Linux*. The MIT Press, 2001.
- 3111 [7] T. Hoare and R. Milner, “Grand challenges for computing research,” *The Computer Journal*, vol. 48, no. 1, pp. 49–52, 2005.
- 3112
3113 [8] J. Dongarra, H. Meuer, and E. Strohmaier, “Top 500 supercomputers,” website,
November 2013.
- 3114
3115 [9] P. Buncic, C. A. Sanchez, J. Blomer, L. Franco, A. Harutyunian, P. Mato, and
3116 Y. Yao, “Cernvm—a virtual software appliance for lhc applications,” in *Journal
of Physics: Conference Series*, vol. 219, no. 4. IOP Publishing, 2010, p. 042003.
- 3117
3118 [10] L.-W. Wang, “A survey of codes and algorithms used in nersc material science
allocations,” *Lawrence Berkeley National Laboratory*, 2006.
- 3119
3120 [11] K. Menon, K. Anala, S. G. Trupti, and N. Sood, “Cloud computing: Applications
3121 in biological research and future prospects,” in *Cloud Computing Technologies, Applications
3122 and Management (ICCCTAM), 2012 International Conference on*. IEEE, 2012, pp. 102–107.
- 3123
3124 [12] Q. He, S. Zhou, B. Kobler, D. Duffy, and T. McGlynn, “Case study
3125 for running hpc applications in public clouds,” in *Proceedings of the 19th ACM
3126 International Symposium on High Performance Distributed Computing*, ser. HPDC ’10. New York, NY, USA: ACM, 2010, pp. 395–401. [Online].
3127 Available: <http://doi.acm.org/10.1145/1851476.1851535>
- 3128
3129 [13] J. Towns, T. Cockerill, M. Dahan, I. Foster, K. Gaither, A. Grimshaw, V. Hazle-
3130 wood, S. Lathrop, D. Lifka, G. D. Peterson *et al.*, “Xsede: accelerating scientific
3131 discovery,” *Computing in Science & Engineering*, vol. 16, no. 5, pp. 62–74, 2014.

- 3132 [14] K. Antypas, “Nersc-6 workload analysis and benchmark selection process,”
3133 *Lawrence Berkeley National Laboratory*, 2008.
- 3134 [15] J. S. Vetter, R. Glassbrook, J. Dongarra, K. Schwan, B. Loftis, S. McNally,
3135 J. Meredith, J. Rogers, P. Roth, K. Spafford *et al.*, “Keeneland: Bringing het-
3136 erogeneous gpu computing to the computational science community,” *Comput-
3137 ing in Science and Engineering*, vol. 13, no. 5, pp. 90–95, 2011.
- 3138 [16] P. Pacheco, *Parallel Programming with MPI*, ser. ISBN. Morgan Kaufmann,
3139 October 1996, no. 978-1-55860-339-4.
- 3140 [17] D. Agrawal, S. Das, and A. El Abbadi, “Big data and cloud computing: cur-
3141 rent state and future opportunities,” in *Proceedings of the 14th International
3142 Conference on Extending Database Technology*. ACM, 2011, pp. 530–533.
- 3143 [18] J. Dean and S. Ghemawat, “Mapreduce: simplified data processing on large
3144 clusters,” *Communications of the ACM*, vol. 51, no. 1, pp. 107–113, 2008.
- 3145 [19] S. Kamburugamuve, G. Fox, D. Leake, and J. Qiu, “Survey of apache big data
3146 stack,” Ph.D. dissertation, Ph. D. Qualifying Exam, Dept. Inf. Comput., Indi-
3147 ana Univ., Bloomington, IN, 2013.
- 3148 [20] M. Chen, S. Mao, and Y. Liu, “Big data: a survey,” *Mobile Networks and
3149 Applications*, vol. 19, no. 2, pp. 171–209, 2014.
- 3150 [21] D. A. Reed and J. Dongarra, “Exascale computing and big data,” *Communi-
3151 cations of the ACM*, vol. 58, no. 7, pp. 56–68, 2015.
- 3152 [22] S. Jha, J. Qiu, A. Luckow, P. K. Mantha, and G. C. Fox, “A tale of two
3153 data-intensive paradigms: Applications, abstractions, and architectures,” in
3154 *Proceedings of the 3rd International Congress on Big Data*, 2014.

- 3155 [23] J. Qiu, S. Jha, A. Luckow, and G. C. Fox, “Towards hpc-abds: An initial high-
3156 performance big data stack,” *Building Robust Big Data Ecosystem ISO/IEC
3157 JTC 1 Study Group on Big Data*, pp. 18–21, 2014.
- 3158 [24] M. W. ur Rahman, N. S. Islam, X. Lu, J. Jose, H. Subramoni, H. Wang, and
3159 D. K. D. Panda, “High-performance rdma-based design of hadoop mapreduce
3160 over infiniband,” in *Parallel and Distributed Processing Symposium Workshops
3161 PhD Forum (IPDPSW), 2013 IEEE 27th International*, May 2013, pp. 1908–
3162 1917.
- 3163 [25] S. Ekanayake, S. Kamburugamuve, and G. Fox, “Spidal: High performance data
3164 analytics with java and mpi on large multicore hpc clusters,” in *Proceedings of
3165 24th High Performance Computing Symposium (HPC 2016)*, 2016.
- 3166 [26] F. Tian and K. Chen, “Towards optimal resource provisioning for running
3167 mapreduce programs in public clouds,” in *Cloud Computing (CLOUD), 2011
3168 IEEE International Conference on*. IEEE, 2011, pp. 155–162.
- 3169 [27] J. Lange, K. Pedretti, T. Hudson, P. Dinda, Z. Cui, L. Xia, P. Bridges, A. Gocke,
3170 S. Jaconette, M. Levenhagen *et al.*, “Palacios and kitten: New high performance
3171 operating systems for scalable virtualized and native supercomputing,” in *Par-
3172 allel & Distributed Processing (IPDPS), 2010 IEEE International Symposium
3173 on*. IEEE, 2010, pp. 1–12.
- 3174 [28] I. Foster, T. Freeman, K. Keahy, D. Scheftner, B. Sotomayer, and X. Zhang,
3175 “Virtual clusters for grid communities,” *Cluster Computing and the Grid, IEEE
3176 International Symposium on*, vol. 0, pp. 513–520, 2006.
- 3177 [29] K. Czajkowski, I. Foster, N. Karonis, C. Kesselman, S. Martin, W. Smith, and
3178 S. Tuecke, “A resource management architecture for metacomputing systems,”

- 3179 in *Workshop on Job Scheduling Strategies for Parallel Processing*. Springer,
3180 1998, pp. 62–82.
- 3181 [30] D. Kondo, B. Javadi, P. Malecot, F. Cappello, and D. P. Anderson, “Cost-
3182 benefit analysis of cloud computing versus desktop grids,” in *Parallel & Dis-*
3183 *tributed Processing, 2009. IPDPS 2009. IEEE International Symposium on*.
3184 IEEE, 2009, pp. 1–12.
- 3185 [31] T. Bell, B. Bompastor, S. Bukowiec, J. C. Leon, M. Denis, J. van Eldik, M. F.
3186 Lobo, L. F. Alvarez, D. F. Rodriguez, A. Marino *et al.*, “Scaling the cern
3187 openstack cloud,” in *Journal of Physics: Conference Series*, vol. 664, no. 2.
3188 IOP Publishing, 2015, p. 022003.
- 3189 [32] T. Gunarathne, T.-L. Wu, J. Qiu, and G. Fox, “Mapreduce in the clouds for
3190 science,” in *Cloud Computing Technology and Science (CloudCom), 2010 IEEE*
3191 *Second International Conference on*. IEEE, 2010, pp. 565–572.
- 3192 [33] S. Ostermann, A. Iosup, N. Yigitbasi, R. Prodan, T. Fahringer, and D. Epema,
3193 “A performance analysis of ec2 cloud computing services for scientific comput-
3194 ing,” in *International Conference on Cloud Computing*. Springer, 2009, pp.
3195 115–131.
- 3196 [34] J. Dongarra *et al.*, “The international exascale software project roadmap,”
3197 *International Journal of High Performance Computing Applications*, p.
3198 1094342010391989, 2011.
- 3199 [35] K. Bergman, S. Borkar, D. Campbell, W. Carlson, W. Dally, M. Denneau,
3200 P. Franzon, W. Harrod, K. Hill, J. Hiller, S. Karp, S. K. and Dean Klein,
3201 R. Lucas, M. Richards, A. Scarpelli, S. Scott, A. Snavely, T. Sterling, R. S.
3202 Williams, and K. Yelick, “Exascale computing study: Technology challenges

- 3203 in achieving exascale systems,” *Defense Advanced Research Projects Agency*
3204 *Information Processing Techniques Office (DARPA IPTO), Tech. Rep*, vol. 15,
3205 2008.
- 3206 [36] J. Shalf, S. Dosanjh, and J. Morrison, “Exascale computing technology chal-
3207 lenges,” in *International Conference on High Performance Computing for Com-*
3208 *putational Science*. Springer, 2010, pp. 1–25.
- 3209 [37] J. Suh, “Proximity scheduler in openstack,” Webpage. [Online]. Available:
3210 <https://wiki.openstack.org/wiki/ProximityScheduler>
- 3211 [38] S. Kamburugamuve, S. Ekanayake, M. Pathirage, and G. Fox, “Towards high
3212 performance processing of streaming data in large data centers,” in *HPBDC*
3213 *2016 IEEE International Workshop on High-Performance Big Data Comput-*
3214 *ing in conjunction with The 30th IEEE International Parallel and Distributed*
3215 *Processing Symposium (IPDPS 2016), Chicago, Illinois USA, Friday*, 2016.
- 3216 [39] G. von Laszewski, G. C. Fox, F. Wang, A. J. Younge, A. Kulshrestha, and
3217 G. Pike, “Design of the FutureGrid Experiment Management Framework,”
3218 in *Proceedings of Gateway Computing Environments 2010 at Supercomputing*
3219 *2010*. New Orleans, LA: IEEE, Nov 2010. [Online]. Available: <http://grids.ucs.indiana.edu/ptliupages/publications/vonLaszewski-10-FG-exp-GCE10.pdf>
- 3220 [40] S. Drake and O. development team, “Heat: OpenStack Orchestration,”
3221 Webpage. [Online]. Available: <https://wiki.openstack.org/wiki/Heat>
- 3222 [41] G. Von Laszewski, F. Wang, H. Lee, H. Chen, and G. C. Fox, “Accessing
3223 multiple clouds with cloudmesh,” in *Proceedings of the 2014 ACM international*
3224 *workshop on Software-defined ecosystems*. ACM, 2014, pp. 21–28.
- 3225

- 3226 [42] “The magellan project,” Webpage. [Online]. Available: <http://magellan.alcf.anl.gov/>
- 3227
- 3228 [43] K. Yelick, S. Coghlan, B. Draney, and R. S. Canon, “The Magellan Report on
3229 Cloud Computing for Science,” U.S. Department of Energy Office of Science
3230 Office of Advanced Scientific Computing Research (ASCR), Tech. Rep., Dec.
3231 2011.
- 3232 [44] K. Jackson, L. Ramakrishnan, K. Muriki, S. Canon, S. Cholia, J. Shalf,
3233 H. Wasserman, and N. Wright, “Performance Analysis of High Performance
3234 Computing Applications on the Amazon Web Services Cloud,” in *2nd IEEE
3235 International Conference on Cloud Computing Technology and Science*. IEEE,
3236 2010, pp. 159–168.
- 3237 [45] D. Merkel, “Docker: lightweight linux containers for consistent development
3238 and deployment,” *Linux Journal*, vol. 2014, no. 239, p. 2, 2014.
- 3239 [46] D. M. Jacobsen and R. S. Canon, “Contain this, unleashing docker for hpc,”
3240 *Proceedings of the Cray User Group*, 2015.
- 3241 [47] M. G. Xavier, M. V. Neves, F. D. Rossi, T. C. Ferreto, T. Lange, and C. A.
3242 De Rose, “Performance evaluation of container-based virtualization for high
3243 performance computing environments,” in *2013 21st Euromicro International
3244 Conference on Parallel, Distributed, and Network-Based Processing*. IEEE,
3245 2013, pp. 233–240.
- 3246 [48] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. L. Harris, A. Ho, R. Neuge-
3247 bauer, I. Pratt, and A. Warfield, “Xen and the art of virtualization,” in *Pro-
3248 ceedings of the 19th ACM Symposium on Operating Systems Principles*, New
3249 York, U. S. A., Oct. 2003, pp. 164–177.

- 3250 [49] “Vmware esx server,” [Online], <http://www.vmware.com/de/products/vi/esx/>.
- 3251 [50] J. Duato, A. J. Pena, F. Silla, J. C. Fernández, R. Mayo, and E. S. Quintana-
3252 Orti, “Enabling CUDA acceleration within virtual machines using rCUDA,” in
3253 *High Performance Computing (HiPC), 2011 18th International Conference on*.
3254 IEEE, 2011, pp. 1–10.
- 3255 [51] L. Shi, H. Chen, J. Sun, and K. Li, “vCUDA: GPU-accelerated high-
3256 performance computing in virtual machines,” *Computers, IEEE Transactions*
3257 on, vol. 61, no. 6, pp. 804–816, 2012.
- 3258 [52] A. Kivity, Y. Kamay, D. Laor, U. Lublin, and A. Liguori, “kvm: the Linux
3259 virtual machine monitor,” in *Proceedings of the Linux Symposium*, vol. 1, 2007,
3260 pp. 225–230.
- 3261 [53] M. Armbrust, A. Fox, R. Griffith, A. Joseph, R. Katz, A. Konwinski, G. Lee,
3262 D. Patterson, A. Rabkin, I. Stoica *et al.*, “A view of cloud computing,” *Com-*
3263 *munications of the ACM*, vol. 53, no. 4, pp. 50–58, 2010.
- 3264 [54] I. Foster, C. Kesselman, and S. Tuecke, “The Anatomy of the Grid: Enabling
3265 Scalable Virtual Organizations,” *Intl. J. Supercomputer Applications*, vol. 15,
3266 no. 3, 2001.
- 3267 [55] I. Foster, C. Kesselman *et al.*, “The Physiology of the Grid:An Open Grid
3268 Services Architecture for Distributed Systems Integration,” Argonne National
3269 Laboratory, Chicago, Tech. Rep., Jan. 2002.
- 3270 [56] D. DiNucci, “Fragmented future,” *AllBusiness-Champions of Small Business*,
3271 1999. [Online]. Available: <http://www.cdinucci.com/Darcy2/articles/Print/Printarticle7.html>

- 3273 [57] A. Arkin, S. Askary, S. Fordin, W. Jekeli, K. Kawaguchi, D. Orchard,
3274 S. Poglani, K. Riemer, S. Struble, P. Takacs-Nagy, I. Trickovic, and S. Zimek,
3275 “Web Services Choreography Interface,” Jun. 2002, version 1.0. [Online].
3276 Available: <http://wwws.sun.com/software/xml/developers/wsci/index.html>
- 3277 [58] D. Krafzig, K. Banke, and D. Slama, *Enterprise SOA: Service-Oriented Archi-*
3278 *tecture Best Practices (The Coad Series)*. Prentice Hall PTR Upper Saddle
3279 River, NJ, USA, 2004.
- 3280 [59] L. Wang, G. von Laszewski, A. J. Younge, X. He, M. Kunze, and J. Tao,
3281 “Cloud Computing: a Perspective Study,” *New Generation Computing*, vol. 28,
3282 pp. 63–69, Mar 2010. [Online]. Available: <http://cyberaide.googlecode.com/svn/trunk/papers/10-cloudcomputing-NGC/vonLaszewski-10-NGC.pdf>
- 3283 [60] G. von Laszewski, A. J. Younge, X. He, K. Mahinthakumar, and
3284 L. Wang, “Experiment and Workflow Management Using Cyberaide
3285 Shell,” in *Proceedings of the 4th International Workshop on Workflow*
3286 *Systems in e-Science (WSES 09) with 9th IEEE/ACM International*
3287 *Symposium on Cluster Computing and the Grid (CCGrid 09)*. IEEE,
3288 May 2009. [Online]. Available: <http://cyberaide.googlecode.com/svn/trunk/papers/09-gridshell-ccgrid/vonLaszewski-ccgrid09-final.pdf>
- 3289 [61] G. von Laszewski, F. Wang, A. J. Younge, X. He, Z. Guo, and M. Pierce,
3290 “Cyberaide JavaScript: A JavaScript Commodity Grid Kit,” in *Proceedings*
3291 *of the Grid Computing Environments 2007 at Supercomputing 2008*. Austin,
3292 TX: IEEE, Nov 2008. [Online]. Available: <http://cyberaide.googlecode.com/svn/trunk/papers/08-javascript/vonLaszewski-08-javascript.pdf>
- 3293
- 3294
- 3295

- 3296 [62] G. von Laszewski, J. Diaz, F. Wang, and G. C. Fox, “Comparison of multiple
3297 cloud frameworks,” in *Cloud Computing (CLOUD), 2012 IEEE 5th International*
3298 *Conference on*, June 2012, pp. 734–741.
- 3299 [63] B. P. Rimal, E. Choi, and I. Lumb, “A taxonomy and survey of cloud computing
3300 systems,” *INC, IMS and IDC*, pp. 44–51, 2009.
- 3301 [64] B. Sotomayor, R. S. Montero, I. M. Llorente, and I. Foster, “Virtual infrastruc-
3302 ture management in private and hybrid clouds,” *IEEE Internet Computing*,
3303 vol. 13, no. 5, pp. 14–22, 2009.
- 3304 [65] S. A. Baset, “Cloud slas: present and future,” *ACM SIGOPS Operating Systems*
3305 *Review*, vol. 46, no. 2, pp. 57–66, 2012.
- 3306 [66] “Amazon Elastic Compute Cloud.” [Online]. Available: <http://aws.amazon.com/ec2/>
- 3307
- 3308 [67] S. Krishnan and J. L. U. Gonzalez, “Google compute engine,” in *Building Your*
3309 *Next Big Thing with Google Cloud Platform*. Springer, 2015, pp. 53–81.
- 3310 [68] “Nimbus Project,” <http://www.nimbusproject.org>. Last access Mar. 2011.
- 3311 [69] K. Keahey, I. Foster, T. Freeman, and X. Zhang, “Virtual workspaces: Achiev-
3312 ing quality of service and quality of life in the grid,” *Scientific Programming*
3313 *Journal*, vol. 13, no. 4, pp. 265–276, 2005.
- 3314 [70] “Amazon web services s3 rest api,” <http://awsdocs.s3.amazonaws.com/S3/latest/s3-api.pdf>. Last Access Mar. 2011.
- 3315
- 3316 [71] “Jets3t project,” <http://bitbucket.org/jmurty/jets3t/wiki/Home>. Last Access
3317 Mar. 2011.

- 3318 [72] “Boto project,” <http://code.google.com/p/boto>. Last Access Mar. 2011.
- 3319 [73] “S3tools project,” <http://s3tools.org/s3cmd>. Last Access Mar. 2011.
- 3320 [74] D. Nurmi, R. Wolski, C. Grzegorczyk, G. Obertelli, S. Soman, L. Youseff, and
3321 D. Zagorodnov, “The Eucalyptus Open-source Cloud-computing System,” *Pro-*
3322 *ceedings of Cloud Computing and Its Applications*, 2008.
- 3323 [75] “Eucalyptus open-source cloud computing infrastructure,” an Overview, Euca-
3324 lypts Systems, Inc. 2009.
- 3325 [76] “Eucalyptus,” <http://www.eucalyptus.com>, Last Access Mar. 2011.
- 3326 [77] I. H. Shaon, “Eucalyptus and its components,” Webpage.
3327 [Online]. Available: <https://mdshaonimran.wordpress.com/2011/11/26/eucalyptus-and-its-components>
- 3329 [78] “Openstack. cloud software,” <http://openstack.org>, Last Access Mar. 2011.
- 3330 [79] O. Sefraoui, M. Aissaoui, and M. Eleuldj, “Openstack: toward an open-source
3331 solution for cloud computing,” *International Journal of Computer Applications*,
3332 vol. 55, no. 3, 2012.
- 3333 [80] “Opennebula project,” <http://www.opennebula.org>. Last access Mar. 2011.
- 3334 [81] I. M. Llorente, R. Moreno-Vozmediano, and R. S. Montero, “Cloud computing
3335 for on-demand grid resource provisioning,” *Advances in Parallel Computing*,
3336 vol. 18, no. 5, pp. 177–191, 2009.
- 3337 [82] B. Sotomayor, R. S. Montero, I. M. Llorente, and I. Foster, “Capacity leas-
3338 ing in cloud systems using the opennebula engine,” *Cloud Computing and its*
3339 *Applications*, 2008.

- 3340 [83] “Libvirt API webpage,” [http:// libvirt.org](http://libvirt.org). Last access Mar. 2011.
- 3341 [84] “Elastichosts webpage,” <http://www.elastichosts.com>. Last Access Mar. 2011.
- 3342 [85] S. Adabala, V. Chadha, P. Chawla, R. Figueiredo, J. Fortes, I. Krsul, A. Mat-
3343 sunaga, M. Tsugawa, J. Zhang, M. Zhao *et al.*, “From virtualized resources
3344 to virtual computing grids: the in-vigo system,” *Future Generation Computer
3345 Systems*, vol. 21, no. 6, pp. 896–909, 2005.
- 3346 [86] J. Chase, D. Irwin, L. Grit, J. Moore, and S. Sprenkle, “Dynamic virtual clus-
3347 ters in a grid site manager,” in *12th IEEE International Symposium on High
3348 Performance Distributed Computing, 2003. Proceedings*, 2003, pp. 90–100.
- 3349 [87] I. VMware, “VMware vCloud Air,” Webpage. [Online]. Available: <http://vcloud.vmware.com>
- 3350
- 3351 [88] CERN, “LHC Computing Grid Project,” Web Page, Dec. 2003. [Online].
3352 Available: <http://lcg.web.cern.ch/LCG/>
- 3353 [89] J. Luo, A. Song, Y. Zhu, X. Wang, T. Ma, Z. Wu, Y. Xu, and L. Ge, “Grid sup-
3354 porting platform for AMS data processing,” *Lecture notes in computer science*,
3355 vol. 3759, p. 276, 2005.
- 3356 [90] “CMS,” Web Page. [Online]. Available: <http://cms.cern.ch/>
- 3357 [91] K. Hwang, J. Dongarra, and G. C. Fox, *Distributed and cloud computing: from
3358 parallel processing to the internet of things*. Morgan Kaufmann, 2013.
- 3359 [92] J. Diaz, A. J. Younge, G. von Laszewski, F. Wang, and G. C. Fox, “Grappling
3360 Cloud Infrastructure Services with a Generic Image Repository,” in *Proceedings
3361 of Cloud Computing and Its Applications (CCA 2011)*, Argonne, IL, Mar 2011.

- 3362 [93] I. Foster, "The anatomy of the grid: Enabling scalable virtual organizations,"
3363 in *Euro-Par 2001 Parallel Processing: 7th International Euro-Par Conference*.
3364 Springer, 2001, pp. 1–5. [Online]. Available: www.globus.org/alliance/publications/papers/anatomy.pdf
- 3366 [94] K. Keahey and T. Freeman, "Contextualization: Providing one-click virtual
3367 clusters," in *eScience, 2008. eScience'08. IEEE Fourth International Conference on*. IEEE, 2008, pp. 301–308.
- 3369 [95] R. L. Moore, C. Baru, D. Baxter, G. C. Fox, A. Majumdar, P. Papadopoulos,
3370 W. Pfeiffer, R. S. Sinkovits, S. Strande, M. Tatineni, R. P. Wagner, N. Wilkins-
3371 Diehr, and M. L. Norman, "Gateways to discovery: Cyberinfrastructure
3372 for the long tail of science," in *Proceedings of the 2014 Annual Conference on Extreme
3373 Science and Engineering Discovery Environment*, ser. XSEDE '14. New York, NY, USA: ACM, 2014, pp. 39:1–39:8. [Online]. Available:
3374 <http://doi.acm.org/10.1145/2616498.2616540>
- 3376 [96] D. Bernstein, "Containers and cloud: From lxc to docker to kubernetes," *IEEE
3377 Cloud Computing*, vol. 1, no. 3, pp. 81–84, 2014.
- 3378 [97] F. Berman, "From TeraGrid to knowledge grid," *Communications of the ACM*,
3379 vol. 44, no. 11, pp. 27–28, 2001.
- 3380 [98] C. Catlett, "The philosophy of TeraGrid: building an open, extensible, dis-
3381 tributed TeraScale facility," in *2nd IEEE/ACM International Symposium on Cluster
3382 Computing and the Grid, 2002*, 2002, pp. 8–8.
- 3383 [99] H. H. Goldstine and A. Goldstine, "The electronic numerical integrator and
3384 computer (eniac)," *Mathematical Tables and Other Aids to Computation*, vol. 2,
3385 no. 15, pp. 97–110, 1946.

- 3386 [100] J. E. Thornton, "Design of a computer - the control data 6600," 1970.
- 3387 [101] R. M. Russell, "The cray-1 computer system," *Communications of the ACM*,
3388 vol. 21, no. 1, pp. 63–72, 1978.
- 3389 [102] C. L. Seitz, "The cosmic cube," *Communications of the ACM*, vol. 28, no. 1,
3390 pp. 22–33, 1985.
- 3391 [103] G. C. Fox, S. W. Otto, and A. J. Hey, "Matrix algorithms on a hypercube i:
3392 Matrix multiplication," *Parallel computing*, vol. 4, no. 1, pp. 17–31, 1987.
- 3393 [104] W. D. Hillis, *The connection machine*. MIT press, 1989.
- 3394 [105] X. Zhang, C. Yang, F. Liu, Y. Liu, and Y. Lu, "Optimizing and scaling hpcg
3395 on tianhe-2: early experience," in *International Conference on Algorithms and*
3396 *Architectures for Parallel Processing*. Springer, 2014, pp. 28–41.
- 3397 [106] A. Geist, *PVM: Parallel virtual machine: a users' guide and tutorial for net-*
3398 *worked parallel computing*. MIT press, 1994.
- 3399 [107] B. Carpenter, V. Getov, G. Judd, A. Skjellum, and G. C. Fox, "Mpj: Mpi-like
3400 message passing for java," 2000.
- 3401 [108] E. Gabriel, G. E. Fagg, G. Bosilca, T. Angskun, J. J. Dongarra, J. M. Squyres,
3402 V. Sahay, P. Kambadur, B. Barrett, A. Lumsdaine *et al.*, "Open mpi: Goals,
3403 concept, and design of a next generation mpi implementation," in *European*
3404 *Parallel Virtual Machine/Message Passing Interface Users Group Meeting*.
3405 Springer, 2004, pp. 97–104.
- 3406 [109] W. Gropp, E. Lusk, and A. Skjellum, *Using MPI: portable parallel programming*
3407 *with the message-passing interface*. MIT press, 1999, vol. 1.

- 3408 [110] M. J. Koop, T. Jones, and D. K. Panda, “Mvapich-aptus: Scalable high-
3409 performance multi-transport mpi over infiniband,” in *IEEE International Sym-
3410 posium on Parallel and Distributed Processing, 2008. IPDPS 2008.* IEEE,
3411 2008, pp. 1–12.
- 3412 [111] R. Rabenseifner, G. Hager, and G. Jost, “Hybrid mpi/openmp parallel program-
3413 ming on clusters of multi-core smp nodes,” in *2009 17th Euromicro international
3414 conference on parallel, distributed and network-based processing.* IEEE, 2009,
3415 pp. 427–436.
- 3416 [112] D. A. Jacobsen, J. C. Thibault, and I. Senocak, “An mpi-cuda implementation
3417 for massively parallel incompressible flow computations on multi-gpu clusters,”
3418 in *48th AIAA aerospace sciences meeting and exhibit*, vol. 16, 2010, p. 2.
- 3419 [113] A. S. Bland, J. Wells, O. E. Messer, O. Hernandez, and J. Rogers, “Titan:
3420 Early experience with the cray xk6 at oak ridge national laboratory,” *Cray
3421 User Group*, 2012.
- 3422 [114] J. J. Dongarra, P. Luszczek, and A. Petitet, “The linpack benchmark: past,
3423 present and future,” *Concurrency and Computation: practice and experience*,
3424 vol. 15, no. 9, pp. 803–820, 2003.
- 3425 [115] R. C. Murphy, K. B. Wheeler, B. W. Barrett, and J. A. Ang, “Introducing the
3426 graph 500,” *Cray Users Group (CUG)*, 2010.
- 3427 [116] M. A. Heroux and J. Dongarra, “Toward a new metric for ranking high perfor-
3428 mance computing systems,” *Sandia Report, SAND2013-4744*, vol. 312, 2013.
- 3429 [117] R. Brightwell, R. Oldfield, A. B. Maccabe, and D. E. Bernholdt, “Hobbes:
3430 Composition and virtualization as the foundations of an extreme-scale os/r,”

- 3431 in *Proceedings of the 3rd International Workshop on Runtime and Operating*
3432 *Systems for Supercomputers.* ACM, 2013, p. 2.
- 3433 [118] S. Perarnau, R. Gupta, and P. Beckman, “Argo: An exascale operating system
3434 and runtime,” in *Poster Proceedings from IEEE/ACM Supercomputing 2015,*
3435 2015.
- 3436 [119] T. Sterling, D. Kogler, M. Anderson, and M. Brodowicz, “SLOWER: A perfor-
3437 mance model for Exascale computing,” *Supercomputing Frontiers and Innova-*
3438 *tions*, vol. 1, pp. 42–57, Sep 2014.
- 3439 [120] E. Ciurana, *Developing with Google App Engine.* Springer, 2009.
- 3440 [121] D. Chappell, “Introducing windows azure,” Microsoft, Inc, Tech. Rep., 2009.
- 3441 [122] K. Keahey, R. Figueiredo, J. Fortes, T. Freeman, and M. Tsugawa, “Science
3442 clouds: Early experiences in cloud computing for scientific applications,” *Cloud*
3443 *Computing and Applications*, vol. 2008, 2008.
- 3444 [123] R. Creasy, “The origin of the VM/370 time-sharing system,” *IBM Journal of*
3445 *Research and Development*, vol. 25, no. 5, pp. 483–490, 1981.
- 3446 [124] “Amazon elastic compute cloud,” [Online], <http://aws.amazon.com/ec2/>.
- 3447 [125] K. Keahey, I. Foster, T. Freeman, X. Zhang, and D. Galron, “Virtual
3448 Workspaces in the Grid,” *Lecture Notes in Computer Science*, vol. 3648,
3449 pp. 421–431, 2005. [Online]. Available: http://workspace.globus.org/papers/VW_EuroPar05.pdf
- 3451 [126] J. Fontan, T. Vazquez, L. Gonzalez, R. S. Montero, and I. M. Llorente, “Open-
3452 NEbula: The Open Source Virtual Machine Manager for Cluster Computing,”

- 3453 in *Open Source Grid and Cluster Software Conference*, San Francisco, CA, USA,
3454 May 2008.
- 3455 [127] S. Adabala, V. Chadha, P. Chawla, R. Figueiredo, J. Fortes, I. Krsul, A. Mat-
3456 sunaga, M. Tsugawa, J. Zhang, M. Zhao, L. Zhu, and X. Zhu, “From virtualized
3457 resources to virtual computing Grids: the In-VIGO system,” *Future Generation
3458 Comp. Syst.*, vol. 21, no. 6, pp. 896–909, 2005.
- 3459 [128] Rackspace, “Openstack,” WebPage, Jan 2011. [Online]. Available: <http://www.openstack.org/>
- 3460
- 3461 [129] P. Padala, X. Zhu, Z. Wang, S. Singhal, and K. Shin, “Performance evaluation
3462 of virtualization technologies for server consolidation,” HP Laboratories, Tech.
3463 Rep., 2007.
- 3464 [130] J. Watson, “Virtualbox: bits and bytes masquerading as machines,” *Linux
3465 Journal*, vol. 2008, no. 166, p. 1, 2008.
- 3466 [131] D. Leinenbach and T. Santen, “Verifying the Microsoft Hyper-V Hypervisor
3467 with VCC,” *FM 2009: Formal Methods*, pp. 806–809, 2009.
- 3468 [132] I. Parallels, “An introduction to os virtualization and paral-
3469 lels virtuozzo containers,” Parallels, Inc, Tech. Rep., 2010. [On-
3470 line]. Available: http://www.parallels.com/r/pdf/wp/pvc/Parallels_Virtuozzo_Containers_WP_an_introduction_to_os_EN.pdf
- 3471
- 3472 [133] D. Bartholomew, “Qemu: a multihost, multitarget emulator,” *Linux Journal*,
3473 vol. 2006, no. 145, p. 3, 2006.
- 3474 [134] J. N. Matthews, W. Hu, M. Hapuarachchi, T. Deshane, D. Dimatos, G. Hamil-
3475 ton, M. McCabe, and J. Owens, “Quantifying the performance isolation prop-

- 3476 erties of virtualization systems,” in *Proceedings of the 2007 workshop on Ex-*
- 3477 perimental computer science, ser. ExpCS ’07. New York, NY, USA: ACM,
- 3478 2007.
- 3479 [135] Oracle, “Performance evaluation of oracle vm server virtualization software,”
- 3480 Oracle, Whitepaper, 2008. [Online]. Available: <http://www.oracle.com/us/technologies/virtualization/oraclevm/026997.pdf>
- 3482 [136] K. Adams and O. Ageson, “A comparison of software and hardware techniques
- 3483 for x86 virtualization,” in *Proceedings of the 12th international conference on Architectural support for programming languages and operating systems*. ACM,
- 3484 2006, pp. 2–13, vMware.
- 3486 [137] Y. Koh, R. Knauerhase, P. Brett, M. Bowman, Z. Wen, and C. Pu, “An analysis
- 3487 of performance interference effects in virtual environments,” in *Performance Analysis of Systems & Software, 2007. ISPASS 2007. IEEE International Symposium on*. IEEE, 2007, pp. 200–209.
- 3490 [138] S. Rixner, “Network virtualization: Breaking the performance barrier,” *Queue*,
- 3491 vol. 6, no. 1, p. 36, 2008.
- 3492 [139] S. Nanda and T. Chiueh, “A survey of virtualization technologies,” Tech. Rep.,
- 3493 2005.
- 3494 [140] A. Ranadive, M. Kesavan, A. Gavrilovska, and K. Schwan, “Performance im-
- 3495 plications of virtualizing multicore cluster machines,” in *Proceedings of the 2nd workshop on System-level virtualization for high performance computing*. ACM,
- 3496 2008, pp. 1–8.
- 3498 [141] R. Harper and K. Rister, “Kvm limits arbitrary or architectural?” IBM Linux
- 3499 Technology Center, Jun 2009.

- 3500 [142] M. Bolte, M. Sievers, G. Birkenheuer, O. Niehorster, and A. Brinkmann, “Non-
3501 intrusive virtualization management using libvirt,” in *Design, Automation Test
3502 in Europe Conference Exhibition (DATE), 2010*, 2010, pp. 574 –579.
- 3503 [143] “FutureGrid,” Web Page, 2009. [Online]. Available: <http://www.futuregrid.org>
- 3504 [144] J. J. Dujmovic and I. Dujmovic, “Evolution and evaluation of spec bench-
3505 marks,” *SIGMETRICS Perform. Eval. Rev.*, vol. 26, no. 3, pp. 2–9, 1998.
- 3506 [145] P. Luszczek, D. Bailey, J. Dongarra, J. Kepner, R. Lucas, R. Rabenseifner,
3507 and D. Takahashi, “The HPC Challenge (HPCC) benchmark suite,” in *SC06
3508 Conference Tutorial*. Citeseer, 2006.
- 3509 [146] J. Dongarra and P. Luszczek, “Reducing the time to tune parallel dense linear
3510 algebra routines with partial execution and performance modelling,” University
3511 of Tennessee Computer Science Technical Report, Tech. Rep., 2010.
- 3512 [147] K. Dixit, “The SPEC benchmarks,” *Parallel Computing*, vol. 17, no. 10-11, pp.
3513 1195–1209, 1991.
- 3514 [148] SPEC, “Standard performance evaluation corporation,” Webpage, Jan 2011.
3515 [Online]. Available: <http://www.spec.org/>
- 3516 [149] R. Henschel and A. J. Younge, “First quarter 2011 spec omp results,” Webpage,
3517 Mar 2011. [Online]. Available: <http://www.spec.org/omp/results/res2011q1/>
- 3518 [150] A. S. Tanenbaum and M. Van Steen, *Distributed systems*. Prentice Hall, 2002,
3519 vol. 2.
- 3520 [151] Amazon, “Elastic Compute Cloud.” [Online]. Available: <http://aws.amazon.com/ec2/>

- 3522 [152] J. Dean and S. Ghemawat, "MapReduce: simplified data processing on large
3523 clusters," *Commun. ACM*, vol. 51, no. 1, pp. 107–113, 2008.
- 3524 [153] "Windows azure platform," [Online], <http://www.microsoft.com/azure/>.
- 3525 [154] G. von Laszewski, G. C. Fox, F. Wang, A. J. Younge, A. Kulshrestha, G. G.
3526 Pike, W. Smith, J. Vockler, R. J. Figueiredo, J. Fortes *et al.*, "Design of the
3527 futuregrid experiment management framework," in *Gateway Computing Envi-
ronments Workshop (GCE), 2010.* IEEE, 2010, pp. 1–10.
- 3528
- 3529 [155] OpenStack, "Openstack compute administration manual," 2013.
- 3530 [156] D. Nurmi, R. Wolski, C. Grzegorczyk, G. Obertelli, S. Soman, L. Youseff, and
3531 D. Zagorodnov, "The Eucalyptus open-source cloud-computing system," in
3532 *Proceedings of Cloud Computing and Its Applications*, October 2008. [Online].
3533 Available: <http://eucalyptus.cs.ucsb.edu/wiki/Presentations>
- 3534 [157] C. Nvidia, "Programming guide," 2008.
- 3535 [158] J. E. Stone, D. Gohara, and G. Shi, "OpenCL: A parallel programming standard
3536 for heterogeneous computing systems," *Computing in science & engineering*,
3537 vol. 12, no. 3, p. 66, 2010.
- 3538 [159] S. Che, M. Boyer, J. Meng, D. Tarjan, J. W. Sheaffer, and K. Skadron,
3539 "A performance study of general-purpose applications on graphics processors
3540 using CUDA," *Journal of Parallel and Distributed Computing*, vol. 68,
3541 no. 10, pp. 1370 – 1380, 2008, k-means implementation on CUDA
3542 with 72x speedup. Compares to 4-threaded CPU version with 30x
3543 speedup. [Online]. Available: <http://www.sciencedirect.com/science/article/B6WKJ-4SVV8GS-2/2/f7a1dccceb63cbbfd25774c6628d8412>
- 3544

- 3545 [160] Z. Liu and W. Ma, "Exploiting computing power on graphics processing unit,"
3546 vol. 2, Dec. 2008, pp. 1062–1065.
- 3547 [161] S. Hong and H. Kim, "An integrated GPU power and performance model,"
3548 in *ACM SIGARCH Computer Architecture News*, vol. 38. ACM, 2010, pp.
3549 280–289.
- 3550 [162] P. Kogge, K. Bergman, S. Borkar, D. Campbell, W. Carson, W. Dally, M. Den-
3551 neau, P. Franzon, W. Harrod, K. Hill *et al.*, "Exascale computing study: Tech-
3552 nology challenges in achieving exascale systems," 2008.
- 3553 [163] S. Crago, K. Dunn, P. Eads, L. Hochstein, D.-I. Kang, M. Kang, D. Mod-
3554 ium, K. Singh, J. Suh, and J. P. Walters, "Heterogeneous cloud computing," in
3555 *Proceedings of the Workshop on Parallel Programming on Accelerator Clusters*
3556 (*PPAC*). *Cluster Computing (CLUSTER), 2011 IEEE International Confer-*
3557 *ence on*. IEEE, 2011, pp. 378–385.
- 3558 [164] J. Sanders and E. Kandrot, *CUDA by example: an introduction to general-*
3559 *purpose GPU programming*. Addison-Wesley Professional, 2010.
- 3560 [165] V. V. Kindratenko, J. J. Enos, G. Shi, M. T. Showerman, G. W. Arnold, J. E.
3561 Stone, J. C. Phillips, and W.-m. Hwu, "GPU clusters for high-performance
3562 computing," in *Cluster Computing and Workshops, 2009. CLUSTER'09. IEEE*
3563 *International Conference on*. IEEE, 2009, pp. 1–8.
- 3564 [166] K. J. Barker, K. Davis, A. Hoisie, D. J. Kerbyson, M. Lang, S. Pakin, and J. C.
3565 Sancho, "Entering the petaflop era: the architecture and performance of Road-
3566 runner," in *Proceedings of the 2008 ACM/IEEE conference on Supercomputing*.
3567 IEEE Press, 2008, p. 1.

- 3568 [167] S. Craven and P. Athanas, "Examining the viability of FPGA supercomputing,"
3569 *EURASIP Journal on Embedded systems*, vol. 2007, no. 1, pp. 13–13, 2007.
- 3570 [168] G. Shainer, T. Liu, J. Layton, and J. Mora, "Scheduling strategies for HPC as a
3571 service (HPCaaS)," in *Cluster Computing and Workshops, 2009. CLUSTER'09.*
3572 *IEEE International Conference on.* IEEE, 2009, pp. 1–6.
- 3573 [169] A. J. Younge, R. Henschel, J. T. Brown, G. von Laszewski, J. Qiu, and G. C.
3574 Fox, "Analysis of Virtualization Technologies for High Performance Computing
3575 Environments," in *Proceedings of the 4th International Conference on Cloud*
3576 *Computing (CLOUD 2011).* Washington, DC: IEEE, July 2011.
- 3577 [170] W. Wade, "How NVIDIA and Citrix are driving the future of virtualized visual
3578 computing," [Online], <http://blogs.nvidia.com/blog/2013/05/22/synergy/>.
- 3579 [171] S. Long, "Virtual machine graphics acceleration deployment guide," VMWare,
3580 Tech. Rep., 2013.
- 3581 [172] G. Humphreys, M. Houston, R. Ng, R. Frank, S. Ahern, P. D. Kirchner, and
3582 J. T. Klosowski, "Chromium: a stream-processing framework for interactive
3583 rendering on clusters," in *ACM Transactions on Graphics (TOG)*, vol. 21.
3584 ACM, 2002, pp. 693–702.
- 3585 [173] G. Giunta, R. Montella, G. Agrillo, and G. Coviello, "A GPGPU
3586 transparent virtualization component for high performance computing clouds,"
3587 in *Euro-Par 2010 - Parallel Processing*, ser. Lecture Notes in Computer
3588 Science, P. D'Ambra, M. Guarracino, and D. Talia, Eds. Springer
3589 Berlin Heidelberg, 2010, vol. 6271, pp. 379–391. [Online]. Available:
3590 http://dx.doi.org/10.1007/978-3-642-15277-1_37

- 3591 [174] K. Diab, M. Rafique, and M. Hefeeda, “Dynamic sharing of GPUs in cloud
3592 systems,” in *Parallel and Distributed Processing Symposium Workshops PhD*
3593 *Forum (IPDPSW), 2013 IEEE 27th International*, 2013, pp. 947–954.
- 3594 [175] C.-T. Yang, H.-Y. Wang, and Y.-T. Liu, “Using pci pass-through for gpu vir-
3595 tualization with cuda,” in *Network and Parallel Computing*. Springer, 2012,
3596 pp. 445–452.
- 3597 [176] A. Danalis, G. Marin, C. McCurdy, J. S. Meredith, P. C. Roth, K. Spafford,
3598 V. Tippuraju, and J. S. Vetter, “The scalable heterogeneous computing (SHOC)
3599 benchmark suite,” in *Proceedings of the 3rd Workshop on General-Purpose*
3600 *Computation on Graphics Processing Units*. ACM, 2010, pp. 63–74.
- 3601 [177] E. R. Hawkes, R. Sankaran, J. C. Sutherland, and J. H. Chen, “Direct numerical
3602 simulation of turbulent combustion: fundamental insights towards predictive
3603 models,” in *Journal of Physics: Conference Series*, vol. 16. IOP Publishing,
3604 2005, p. 65.
- 3605 [178] F. Silla, “rCUDA: share and aggregate GPUs in your cluster,” Nov. 2012, mel-
3606 lanox Booth Presaentation.
- 3607 [179] H. Jo, J. Jeong, M. Lee, and D. H. Choi, “Exploiting GPUs in virtual machine
3608 for biocloud,” *BioMed research international*, vol. 2013, 2013.
- 3609 [180] J. Duato, A. Pena, F. Silla, R. Mayo, and E. Quintana-Orti, “rCUDA: Re-
3610 ducing the number of gpu-based accelerators in high performance clusters,”
3611 in *High Performance Computing and Simulation (HPCS), 2010 International*
3612 *Conference on*, June 2010, pp. 224–231.
- 3613 [181] B. L. Jacob and T. N. Mudge, “A look at several memory management units,
3614 TLB-refill mechanisms, and page table organizations,” in *Proceedings of the*

- 3615 *Eighth International Conference on Architectural Support for Programming*
3616 *Languages and Operating Systems.* ACM, 1998, pp. 295–306.
- 3617 [182] B.-A. Yassour, M. Ben-Yehuda, and O. Wasserman, “Direct device assignment
3618 for untrusted fully-virtualized virtual machines,” IBM Research, Tech. Rep.,
3619 2008.
- 3620 [183] M. Ben-Yehuda, J. Xenidis, M. Ostrowski, K. Rister, A. Bruemmer, and
3621 L. Van Doorn, “The price of safety: Evaluating IOMMU performance,” in
3622 *Ottawa Linux Symposium*, 2007.
- 3623 [184] J. Liu, “Evaluating standard-based self-virtualizing devices: A performance
3624 study on 10 GbE NICs with SR-IOV support,” in *Parallel Distributed Processing*
3625 (*IPDPS*), 2010 IEEE International Symposium on, April 2010, pp. 1–12.
- 3626 [185] V. Jujjuri, E. Van Hensbergen, A. Liguori, and B. Pulavarty, “VirtFS-a virtu-
3627 alization aware file system passthrough,” in *Ottawa Linux Symposium*, 2010.
- 3628 [186] C. Yang, H. Wang, W. Ou, Y. Liu, and C. Hsu, “On implementation of GPU
3629 virtualization using PCI pass-through,” in *4th IEEE International Conference*
3630 *on Cloud Computing Technology and Science Proceedings (CloudCom)*, 2012.
- 3631 [187] M. Dowty and J. Sugerman, “GPU virtualization on VMware’s hosted I/O
3632 architecture,” *ACM SIGOPS Operating Systems Review*, vol. 43, no. 3, pp. 73
3633 – 82, 2009.
- 3634 [188] “FutureGrid,” Web page. [Online]. Available: <http://portal.futuregrid.org>
- 3635 [189] A. Danalis, G. Marin, C. McCurdy, J. S. Meredith, P. C. Roth, K. Spaf-
3636 ford, V. Tipparaju, and J. S. Vetter, “The Scalable Heterogeneous Computing

- 3637 (SHOC) benchmark suite,” in *Proceedings of the 3rd Workshop on General-*
3638 *Purpose Computation on Graphics Processing Units*, ser. GPGPU ’10. ACM,
3639 2010, pp. 63–74.
- 3640 [190] “LAMMPS molecular dynamics simulator,” <http://lammps.sandia.gov/>, [On-
3641 line; accessed Jan. 2, 2014].
- 3642 [191] A. Athanasopoulos, A. Dimou, V. Mezaris, and I. Kompatsiaris, “GPU ac-
3643 celeration for support vector machines,” in *Proc 12th International Workshop*
3644 *on Image Analysis for Multimedia Interactive Services (WIAMIS 2001)*, April
3645 2011.
- 3646 [192] “LULESH shock hydrodynamics application,” <https://codesign.llnl.gov/lulesh.php>, [Online; accessed Jan. 2, 2014].
- 3648 [193] S. Plimpton, “Fast parallel algorithms for short-range molecular dynamics,”
3649 *Journal of Computational Physics*, vol. 117, no. 1, pp. 1 – 19, 1995.
- 3650 [194] W. M. Brown, “GPU acceleration in LAMMPS,” <http://lammps.sandia.gov/workshops/Aug11/Brown/brown11.pdf>, [Online; accessed Jan. 2, 2014].
- 3652 [195] C.-C. Chang and C.-J. Lin, “LIBSVM: A library for support vector machines,”
3653 *ACM Trans. Intell. Syst. Technol.*, vol. 2, no. 3, pp. 27:1–27:27, May 2011.
- 3654 [196] “Hydrodynamics Challenge Problem,” <https://codesign.llnl.gov/pdfs/spec-7.pdf>, [Online; accessed Jan. 2, 2014].
- 3656 [197] A. Arcangeli, “Transparent hugepage support,” <http://www.linux-kvm.org/wiki/images/9/9e/2010-forum-thp.pdf>, 2010, [Online; accessed Jan. 29, 2014].
- 3658 [198] J. Jose, M. Li, X. Lu, K. C. Kandalla, M. D. Arnold, and D. K. Panda, “SR-IOV
3659 support for virtualization on infiniband clusters: Early experience,” in *Cluster*

- 3660 *Computing and the Grid, IEEE International Symposium on.* IEEE Computer
3661 Society, 2013, pp. 385–392.
- 3662 [199] R. L. Moore, C. Baru, D. Baxter, G. C. Fox, A. Majumdar, P. Papadopoulos,
3663 W. Pfeiffer, R. S. Sinkovits, S. Strande, M. Tatineni *et al.*, “Gateways to
3664 discovery: Cyberinfrastructure for the long tail of science,” in *Proceedings of
3665 the 2014 Annual Conference on Extreme Science and Engineering Discovery
3666 Environment.* ACM, 2014, p. 39.
- 3667 [200] P. Luszczek, E. Meek, S. Moore, D. Terpstra, V. M. Weaver, and J. Dongarra,
3668 “Evaluation of the hpc challenge benchmarks in virtualized environments,” in
3669 *Proceedings of the 2011 International Conference on Parallel Processing - Vol-
3670 ume 2*, ser. Euro-Par’11. Berlin, Heidelberg: Springer-Verlag, 2012, pp. 436–
3671 445.
- 3672 [201] N. Huber, M. von Quast, M. Hauck, and S. Kounev, “Evaluating and model-
3673 ing virtualization performance overhead for cloud environments.” in *CLOSER*,
3674 2011, pp. 563–573.
- 3675 [202] J. P. Walters, A. J. Younge, D.-I. Kang, K.-T. Yao, M. Kang, S. P. Crago,
3676 and G. C. Fox, “GPU-Passthrough Performance: A Comparison of KVM, Xen,
3677 VMWare ESXi, and LXC for CUDA and OpenCL Applications,” in *Proceedings
3678 of the 7th IEEE International Conference on Cloud Computing (CLOUD 2014).*
3679 Anchorage, AK: IEEE, 2014.
- 3680 [203] J. Jose, M. Li, X. Lu, K. C. Kandalla, M. D. Arnold, and D. K. Panda, “Sr-iov
3681 support for virtualization on infiniband clusters: Early experience,” in *Cluster,
3682 Cloud and Grid Computing (CCGrid), 2013 13th IEEE/ACM International
3683 Symposium on.* IEEE, 2013, pp. 385–392.

- 3684 [204] M. Musleh, V. Pai, J. P. Walters, A. J. Younge, and S. P. Crago, “Bridging
3685 the Virtualization Performance Gap for HPC using SR-IOV for InfiniBand,”
3686 in *Proceedings of the 7th IEEE International Conference on Cloud Computing*
3687 (*CLOUD 2014*). Anchorage, AK: IEEE, 2014.
- 3688 [205] “Aws high performance computing,” <http://aws.amazon.com/hpc/>, last Access
3689 Nov. 2014.
- 3690 [206] “Openstack flavors,” <http://docs.openstack.org/openstack-ops/content/flavors.html>, last Access Nov. 2014.
- 3692 [207] K. Pepple, *Deploying OpenStack*. O’Reilly Media, Inc., 2011.
- 3693 [208] S. Hazelhurst, “Scientific computing using virtual high-performance computing:
3694 a case study using the amazon elastic computing cloud,” in *Proceedings of the
3695 2008 annual research conference of the South African Institute of Computer
3696 Scientists and Information Technologists on IT research in developing countries:
3697 riding the wave of technology*. ACM, 2008, pp. 94–103.
- 3698 [209] E. Amazon, “Amazon elastic compute cloud (amazon ec2),” *Amazon Elastic
3699 Compute Cloud (Amazon EC2)*, 2010.
- 3700 [210] R. Jennings, *Cloud Computing with the Windows Azure Platform*. John Wiley
3701 & Sons, 2010.
- 3702 [211] “Google cloud platform,” <https://cloud.google.com/>, last Access Nov. 2014.
- 3703 [212] L. Ramakrishnan, R. S. Canon, K. Muriki, I. Sakrejda, and N. J. Wright,
3704 “Evaluating interconnect and virtualization performance forhigh performance
3705 computing,” *SIGMETRICS Perform. Eval. Rev.*, vol. 40, no. 2, pp. 55–60,
3706 Oct. 2012. [Online]. Available: <http://doi.acm.org/10.1145/2381056.2381071>

- 3707 [213] M. Righini, "Enabling intel virtualization technology features and benefits,"
3708 Intel Corporation, Tech. Rep., 2010.
- 3709 [214] AMD, "AMD i/o virtualization technology (IOMMU) specification," AMD Cor-
3710 poration, Tech. Rep., 2009.
- 3711 [215] A. Limited, "Arm system memory management unit architecture specification,"
3712 ARM Limited, Tech. Rep., 2013.
- 3713 [216] A. J. Younge, J. P. Walters, S. Crago, and G. C. Fox, "Evaluating GPU
3714 Passthrough in Xen for High Performance Cloud Computing," in *High-*
3715 *Performance Grid and Cloud Computing Workshop at the 28th IEEE Inter-*
3716 *national Parallel and Distributed Processing Symposium*, IEEE. Pheonix, AZ:
3717 IEEE, 05/2014 2014.
- 3718 [217] L. Vu, H. Sivaraman, and R. Bidarkar, "Gpu virtualization for high
3719 performance general purpose computing on the esx hypervisor," in *Proceedings*
3720 *of the High Performance Computing Symposium*, ser. HPC '14. San Diego,
3721 CA, USA: Society for Computer Simulation International, 2014, pp. 2:1–2:8.
3722 [Online]. Available: <http://dl.acm.org/citation.cfm?id=2663510.2663512>
- 3723 [218] J. Liu, "Evaluating standard-based self-virtualizing devices: A performance
3724 study on 10 gbe nics with sr-iov support," in *Parallel Distributed Processing*
3725 (*IPDPS*), 2010 IEEE International Symposium on, April 2010, pp. 1–12.
- 3726 [219] T. P. P. D. L. Ruivo, G. B. Altayo, G. Garzoglio, S. Timm, H. Kim, S.-Y.
3727 Noh, and I. Raicu, "Exploring infiniband hardware virtualization in opennebula
3728 towards efficient high-performance computing." in *CCGRID*, 2014, pp. 943–948.
- 3729 [220] "NVIDIA GPUDirect," <https://developer.nvidia.com/gpudirect>, [Online; ac-
3730 cessed Nov. 24, 2014].

- 3731 [221] G. Shainer, A. Ayoub, P. Lui, T. Liu, M. Kagan, C. R. Trott, G. Scantlen, and
3732 P. S. Crozier, “The development of mellanox/nvidia gpudirect over infinibanda
3733 new model for gpu to gpu communications,” *Computer Science-Research and*
3734 *Development*, vol. 26, no. 3-4, pp. 267–273, 2011.
- 3735 [222] “Getting Xen working for Intel(R) Xeon Phi(tm)
3736 Coprocessor,” <https://software.intel.com/en-us/articles/getting-xen-working-for-intelr-xeon-phitm-coprocessor>, [Online; accessed
3737 Nov. 24, 2014].
- 3739 [223] “Mellanox Neutron Plugin,” <https://wiki.openstack.org/wiki/Mellanox-Neutron>, [Online; accessed Nov. 24, 2014].
- 3741 [224] S. Plimpton, P. Crozier, and A. Thompson, “Lammps-large-scale
3742 atomic/molecular massively parallel simulator,” *Sandia National Laboratories*, 2007.
- 3743
3744 [225] J. Anderson, A. Keys, C. Phillips, T. Dac Nguyen, and S. Glotzer, “Hoomd-
3745 blue, general-purpose many-body dynamics on the gpu,” in *APS Meeting Abstracts*, vol. 1, 2010, p. 18008.
- 3746
3747 [226] K. Asanovic, R. Bodik, B. C. Catanzaro, J. J. Gebis, P. Husbands, K. Keutzer,
3748 D. A. Patterson, W. L. Plishker, J. Shalf, S. W. Williams *et al.*, “The land-
3749 scape of parallel computing research: A view from berkeley,” Technical Report
3750 UCB/EECS-2006-183, EECS Department, University of California, Berkeley,
3751 Tech. Rep., 2006.
- 3752 [227] G. Fox, G. von Laszewski, J. Diaz, K. Keahey, J. Fortes, R. Figueiredo,
3753 S. Smallen, W. Smith, and A. Grimshaw, “Futuregrida reconfigurable testbed

- 3754 for cloud, hpc and grid computing,” *Contemporary High Performance Comput-*
3755 *ing: From Petascale toward Exascale, Computational Science. Chapman and*
3756 *Hall/CRC*, 2013.
- 3757 [228] K. Keahey, J. Mambretti, D. K. Panda, P. Rad, W. Smith, and
3758 D. Stanzione, “Nsf chameleon cloud,” website, November 2014. [Online].
3759 Available: <http://www.chameleoncloud.org/>
- 3760 [229] M. Rosenblum and T. Garfinkel, “Virtual machine monitors: Current technol-
3761 ogy and future trends,” *Computer*, vol. 38, no. 5, pp. 39–47, 2005.
- 3762 [230] C. Clark, K. Fraser, S. Hand, J. G. Hansen, E. Jul, C. Limpach, I. Pratt, and
3763 A. Warfield, “Live migration of virtual machines,” in *Proceedings of the 2nd*
3764 *conference on Symposium on Networked Systems Design & Implementation-*
3765 *Volume 2*. USENIX Association, 2005, pp. 273–286.
- 3766 [231] M. R. Hines and K. Gopalan, “Post-copy based live virtual machine migration
3767 using adaptive pre-paging and dynamic self-ballooning,” in *Proceedings of the*
3768 *2009 ACM SIGPLAN/SIGOPS international conference on Virtual execution*
3769 *environments*. ACM, 2009, pp. 51–60.
- 3770 [232] P. Lu, A. Barbalace, and B. Ravindran, “Hsg-lm: Hybrid-copy speculative
3771 guest os live migration without hypervisor,” in *Proceedings of the*
3772 *6th International Systems and Storage Conference*, ser. SYSTOR ’13.
3773 New York, NY, USA: ACM, 2013, pp. 2:1–2:11. [Online]. Available:
3774 <http://doi.acm.org/10.1145/2485732.2485736>
- 3775 [233] J. Chu and V. Kashyap, “Transmission of IP over InfiniBand (IPoIB),” IETF,
3776 Tech. Rep., 2006.

- 3777 [234] W. Yu, N. S. Rao, P. Wyckoff, and J. S. Vetter, “Performance of rdma-capable
3778 storage protocols on wide-area network,” in *2008 3rd Petascale Data Storage
3779 Workshop*. IEEE, 2008, pp. 1–5.
- 3780 [235] W. Huang, Q. Gao, J. Liu, and D. K. Panda, “High performance virtual machine
3781 migration with rdma over modern interconnects,” in *2007 IEEE International
3782 Conference on Cluster Computing*. IEEE, 2007, pp. 11–20.
- 3783 [236] D. Gilbert, “Post copy live migration,” Webpage, 2015. [Online]. Available:
3784 <http://wiki.qemu.org/Features/PostCopyLiveMigration>
- 3785 [237] M. S. Birrittella, M. Debbage, R. Huggahalli, J. Kunz, T. Lovett, T. Rimmer,
3786 K. D. Underwood, and R. C. Zak, “Intel omni-path architecture: Enabling
3787 scalable, high performance fabrics,” in *2015 IEEE 23rd Annual Symposium on
3788 High-Performance Interconnects*, Aug 2015, pp. 1–9.
- 3789 [238] M. Beck and M. Kagan, “Performance evaluation of the rdma over ethernet
3790 (roce) standard in enterprise data centers infrastructure,” in *Proceedings of
3791 the 3rd Workshop on Data Center-Converged and Virtual Ethernet Switching*.
3792 International Teletraffic Congress, 2011, pp. 9–15.
- 3793 [239] E. Kissel and M. Swany, “Photon: Remote memory access middleware for high-
3794 performance runtime systems,” in *2016 IEEE International Parallel and Dis-
3795 tributed Processing Symposium Workshops (IPDPSW)*. IEEE, 2016, pp. 1736–
3796 1743.
- 3797 [240] J. Lofstead, I. Jimenez, and C. Maltzahn, “Consistency and fault tolerance con-
3798 siderations for the next iteration of the doe fast forward storage and io project,”
3799 in *2014 43rd International Conference on Parallel Processing Workshops*, Sept
3800 2014, pp. 61–69.

- 3801 [241] C. G. Wright Jr, “Trinity burst buffer-architecture and design,” Los Alamos
3802 National Laboratory (LANL), Tech. Rep., 2015.
- 3803 [242] F. B. Schmuck and R. L. Haskin, “Gpfss: A shared-disk file system for large
3804 computing clusters.” in *FAST*, vol. 2, 2002, pp. 231–244.
- 3805 [243] M. J. Rashti and A. Afsahi, “10-gigabit iwarps ethernet: comparative perfor-
3806 mance analysis with infiniband and myrinet-10g,” in *2007 IEEE International*
3807 *Parallel and Distributed Processing Symposium*. IEEE, 2007, pp. 1–8.
- 3808 [244] P. Duclos, F. Boeri, M. Auguin, and G. Giraudon, “Image processing on a
3809 simd/spmd architecture: Opsila,” in *Pattern Recognition, 1988., 9th Interna-*
3810 *tional Conference on*, Nov 1988, pp. 430–433 vol.1.
- 3811 [245] H. A. Lagar-Cavilla, J. A. Whitney, A. M. Scannell, P. Patchin, S. M. Rumble,
3812 E. De Lara, M. Brudno, and M. Satyanarayanan, “Snowflock: rapid virtual
3813 machine cloning for cloud computing,” in *Proceedings of the 4th ACM European*
3814 *conference on Computer systems*. ACM, 2009, pp. 1–12.
- 3815 [246] H. A. Lagar-Cavilla, J. A. Whitney, R. Bryant, P. Patchin, M. Brudno,
3816 E. de Lara, S. M. Rumble, M. Satyanarayanan, and A. Scannell, “Snowflock:
3817 Virtual machine cloning as a first-class cloud primitive,” *ACM Transactions on*
3818 *Computer Systems (TOCS)*, vol. 29, no. 1, p. 2, 2011.
- 3819 [247] A. J. Younge, G. von Laszewski, L. Wang, and G. C. Fox, “Providing a Green
3820 Framework for Cloud Based Data Centers,” in *The Handbook of Energy-Aware*
3821 *Green Computing*, I. Ahmad and S. Ranka, Eds. Chapman and Hall/CRC
3822 Press, 2011, ch. 17, in press.
- 3823 [248] D. P. Berrange, “Openstack performance optimization,” in *KVM Forum 2014*,
3824 2014.

- 3825 [249] Y. Jiang and Y. He, “Openstack pci passthrough,” Webpage, Intel, Inc, 2016.
3826 [Online]. Available: https://wiki.openstack.org/wiki/Pci_passthrough
- 3827 [250] A. Hanemann, J. W. Boote, E. L. Boyd, J. Durand, L. Kudarimoti, R. Lapacz,
3828 D. M. Swany, S. Trocha, and J. Zurawski, “Perfsonar: A service oriented archi-
3829 tecture for multi-domain network monitoring,” in *International Conference on*
3830 *Service-Oriented Computing*. Springer, 2005, pp. 241–254.
- 3831 [251] D. Serrano, S. Bouchenak, Y. Kouki, T. Ledoux, J. Lejeune, J. Sopena,
3832 L. Arantes, and P. Sens, “Towards qos-oriented sla guarantees for online
3833 cloud services,” in *Cluster, Cloud and Grid Computing (CCGrid), 2013 13th*
3834 *IEEE/ACM International Symposium on*. IEEE, 2013, pp. 50–57.
- 3835 [252] K. Pedretti, “Kitten: A lightweight operating system for ultrascaling supercom-
3836 puters,” 2011.
- 3837 [253] J. Ahrens, D. Rogers, and B. Springmeyer, “Visualization and data analysis at
3838 the exascale,” *National Nuclear Security Administration (NNSA) Accelerated*
3839 *Strategic Computing (ASC) Exascale Environment Planning Process*, vol. 2,
3840 2010.
- 3841 [254] V. Vishwanath, M. Hereld, and M. E. Papka, “Toward simulation-time data
3842 analysis and i/o acceleration on leadership-class systems,” in *Large Data Anal-*
3843 *ysis and Visualization (LDAV), 2011 IEEE Symposium on*. IEEE, 2011, pp.
3844 9–14.