A GPGPU Transparent Virtualization Component for High Performance Computing Clouds

Giulio Giunta, Raffaele Montella, Giuseppe Agrillo, and Giuseppe Coviello

University of Napoli Parthenope - Department of Applied Science
{giulio.giunta,raffaele.montella}@uniparthenope.it,
{giuseppe.agrillo,giuseppe.coviello}@uniparthenope.it

Abstract. The GPU Virtualization Service (gVirtuS) presented in this work tries to fill the gap between in-house hosted computing clusters, equipped with GPGPUs devices, and pay-for-use high performance virtual clusters deployed via public or private computing clouds. gVirtuS allows an instanced virtual machine to access GPGPUs in a transparent and hypervisor independent way, with an overhead slightly greater than a real machine/GPGPU setup. The performance of the components of gVirtuS is assessed through a suite of tests in different deployment scenarios, such as providing GPGPU power to cloud computing based HPC clusters and sharing remotely hosted GPGPUs among HPC nodes.

Keywords: Grid, Cloud, High-performance computing, many core, Virtualization, Graphic processing units, Hypervisor, GPGPU.

1 Introduction

In a grid environment, computing power is offered on-demand to perform large numerical simulations on a network of machines, potentially extended all over the world [13]. Virtualization techniques are a promising effort to run generic complex high performance scientific software on a grid, inspiring a novel computing paradigm in which virtualized resources are spread in a cloud of real high performance hardware infrastructures. This model, well known as Cloud Computing [16], is characterized by resource allocation elasticity, high efficiency in using resources, and pay-per-use charge In cloud computing, hardware appliances and software applications are provisioned respectively by means of hardware virtualization and software-as-a-service solutions giving scientists the chance to deal with their specific research needs [8]. Especially in the field of parallel computing applications, virtual clusters instanced on cloud infrastructures suffers from the poorness of message passing performances between virtual machine instances running on the same real machine and also from the impossibility to access hardware specific accelerating devices as GPUs [11,20]. Recently, scientific computing has experienced on general purpose graphics processing units to accelerate data parallel computing tasks. One the most successful GPU based

P. D'Ambra, M. Guarracino, and D. Talia (Eds.): Euro-Par 2010, Part I, LNCS 6271, pp. 379–391, 2010.

accelerating system is provided by nVIDIA and relies on the CUDA programming paradigm supporting high level languages tools [3]. Presently, virtualization does not allow a transparent use of accelerators as CUDA based GPUs, as virtual /real machines and guest / host real machines communication issues rise serious limitations to the overall potential performance of a cloud computing infrastructure based on elastically allocated resources. In this paper we present the component gVirtuS (GPU Virtualization Service) [1] as results in GPGPUs transparent virtualization targeting mainly the use of nVIDIA CUDA based accelerator boards through virtual machines instanced to accelerate scientific computations. The paper is organized as follows: section 2 contains a brief description of the software architecture and the main design choices; section 3 presents our implementation of the GPU virtualization; section 4 describes the test suite, the performance tests and the obtained results; section 5 shows how gVirtuS can be used in a private high performance cloud computing environment in order to accelerate virtual machines; in section 6 we compare our approach to other existing solutions; finally, section 7 draws conclusions and discusses some future developments.

2 System Architecture and Design

In our prototype the hardware platform consists of two general purpose x86 quad core hyper-threaded processors and two specialized graphics accelerators based on nVIDIA GPUs Tesla 1060C plus a nVIDIA Quadro FX 5600 summing up 8 CPU cores and about 720 GPU cores. A hypervisor concurrently deploys the applications requiring access to the GPU accelerators as VM appliances. The device is under control of the hypervisor, whereas the interface between guest and host machine is performed by a front end/back end system. An access to the GPU is routed via the front end/back end layers under control of a management component, and data are moved from GPU to guest VM application, and vice versa. The front end and the back end layers implement the uncoupling between the hypervisor and the communication layer (Fig. 1a). A key property of the proposed system is its ability to execute CUDA kernels with an overall performance similar to that obtained by real machines with direct access to the accelerators. This has been achieved by developing a component that provides a high performance communication between virtual machines and their hosts. The GPU virtualization is independent of the hypervisor. The choice of the hypervisor deeply affects the efficiency of the communication between guest and host machines and then between the GPU virtualization front end and back end. Xen [7], [9] is a hypervisor that runs directly on the top of the hardware through a custom Linux kernel. Xen provides a communication library between guest and host machines, called XenLoop, which implements low latency and wide bandwidth TCP/IP and UDP connections. XenLoop is application transparent and offers an automatic discovery of the supported virtual machines [21]. VMware [6] is a commercial hypervisor running at the application level. VMware provides a datagram API to exchange small messages, a shared memory API to

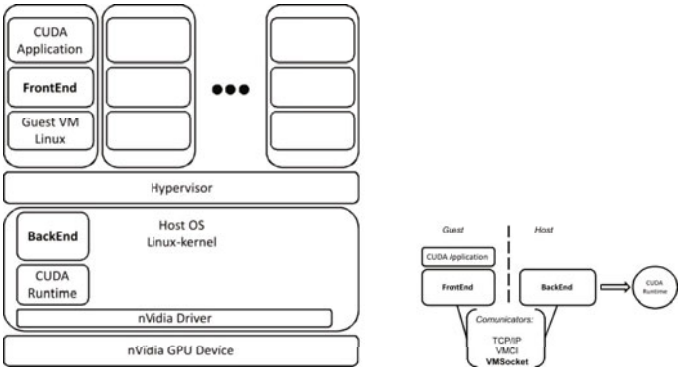

Fig. 1. The gVirtuS architecture (a, left) and components (b, right)

share data, an access control API to control which resources a virtual machine can access and a discovery service for publishing and retrieving resources [5]. KVM (Kernel-based virtual machine) [2] is a Linux loadable kernel module now embedded as a standard component in most Linux distributions. KVM supplies a high performance guest/host communication component that exposes a set of fully emulated inter virtual machines and virtual/real machines serial ports and network sockets, based on a shared memory approach [4]. gVirtuS relies on a specially designed, fully pluggable communication component independent of the hypervisor and of the communication channel. In our system, the CUDA library, instead of dealing directly with the hardware accelerator, interacts with a GPU virtualization front end. This front end packs the library function invocation and sends it to the back end. The back end deals with the hardware using the CUDA driver; it unpacks the library function invocation and suitably maps memory pointers. Then it executes the CUDA operation, retrieves the results and sends them to the front end using the communicator. Finally, the front end interacts with the CUDA library by terminating the GPU operation and providing results to the calling program. This design is hypervisor independent, communicator independent and even accelerator independent, since the same approach could be followed to implement different kinds of virtualization.

3 gVirtuS Implementation

We have developed a wrapper stub CUDA library which exposes the same interface of the nVIDIA CUDA library. The wrapper library intercepts all the CUDA calls made by an application, collects arguments, packs them into a CUDA call packet, and sends the packet to the front end. The front end driver manages the connection between guest virtual machine and host machine. It uses the services offered by the communicator component to establish event channels through both sides, receiving call packets from the wrap CUDA library, sending these requests to the back end for execution and relaying responses back to the wrap

CUDA library (Fig. 1b). In order to test the front end/back end interaction, the wrap CUDA library behavior and the component interface, we developed a TCP-IP based communicator. This component exhibits poor performances due to the network stack, but permits deployment of a complex infrastructure in which the multi-core computing elements and the GPU elements are hosted on diverse hardware systems. We focused on VMware and KVM hypervisors. The VMware Communicator Interface (VMCI) provides a high performance communication among virtual machine instances and the hosting machine. The communicator uses the VMCI SDK APIs to pass packed CUDA function invocations and return the results. VMSocket is the component we have designed to obtain a high performance communicator. vmSocket exposes Unix Sockets on virtual machine instances thanks to a QEMU device connected to the virtual PCI bus. On the host side, the back end is responsible for executing the CUDA calls received from the front end and for returning the computed results. Once a call has been executed, it notifies the guest and passes the results through the connector component. GPU based high performance computing applications usually require massive data transfer between host (CPU) memory and device (GPU) memory. In gVirtuS, the front end / back end interaction turns out to be effective and efficient because there is no mapping between guest memory and device memory and the memory device pointers are never de-referenced on the host side of the CUDA enabled software, since CUDA kernels are executed on our back end side, where the pointers are fully consistent.

4 Performance Evaluation

In order to evaluate and assess the impact of GPU virtualization and its related overhead introduced by gVirtuS, we have carried out an extensive suite of performance tests. We used a workstation Genesis GE-i940 Tesla equipped with a i7- 940 2,93 133 GHz fsb, Quad Core hyper-threaded 8 Mb cache CPU and 12Gb RAM. The GPU subsystem is enforced by one nVIDIA Quadro FX5800 4Gb RAM video card and two nVIDIA Tesla C1060 4 Gb RAM summing up 720 CUDA cores. The testing system has been built on top of the Fedora 12 Linux operating system, the nVIDIA CUDA Driver, and the SDK/Toolkit version 2.3. We have also compared and contrasted the commercial VMware hypervisor and the open source KVM/QEMU. In fact, the aim of our tests is twofold: to stress the reliability of the CPU/GPU virtual system and to gather quantitative information on performance. The first typology of tests is targeted to verify whether our software stack works in a stable and fully transparent way respect to the software system provided by the nVIDIA CUDA SDK. Each program has been set up in order to execute the same algorithm on both CPU and GPU and check the differences between the results of the two executions. All tests have shown that the gVirtuS stack does not affect the accuracy of the numerical results. The performance tests have been developed in order to assess the efficiency of the virtualized systems under different gVirtuS stacks conditions. We compared the efficiency (elapsed time) of the execution of some selected algorithm by varying

Table 1. Averaged results summary of ScalarProd, MatrixMul and Histogram in our test cases (percent respect Host/cpu)

	histogram	matrixMul	scalarProd
host/gpu	9.51	9.24	8.37
kvm/cpu	105.57	99.48	106.75
vmware/cpu	103.63	105.34	106.58
host/tcp	67.07	52.73	40.87
kvm/tcp	67.54	50.43	42.95
vmware/tcp	67.73	50.37	41.54
host/afunix	11.72	16.73	9.06
kvm/vmsocket	15.23	31.21	10.33
vmware/vmci	28.38	52.66	18.03

both the combinations of hypervisors (no hypervisor, VMware, KVM-QEMU) and communication channels (TCP/IP, VMCI and vmSocket), and the size of the input data. The benchmark software is the implementation of three basic numerical algorithms: ScalarProd, MatrixMul and Histogram (Table 1).

ScalarProd computes k scalar products of two real vectors of length m. Notice that each product is executed by a CUDA thread on the GPU so no synchronization is required.

MatrixMul computes a matrix multiplication. The matrices are $m \times n$ and $n \times p$, respectively. It partitions the input matrices in blocks and associates a CUDA thread to each block. As in the previous case, there is no need of synchronization.

Histogram returns the histogram of a set of m uniformly distributed real random numbers in 64 bins. The set is distributed among the CUDA threads each computing a local histogram. The final result is obtained through synchronization and reduction techniques.

The benchmark results are syntetically shown in the Table 1

Fig. 2 shows the results of the ScalarProd program. In the test process, the input size of the problem increases by jointly varying k in the range $[2^8, 2^{12}]$ and m in $[2^{13}, 2^{16}]$. We performed each experiment using both host/cpu and host/gpu setups as reference. The first experiment returns the execution time when only the CPU on the real machine is used; the second experiment returns the execution time when the GPU with the standard CUDA Tools/SDK is used. As the problem size grows exponentially in the SalarProd test, the CPU time increases following the same exponential trend. Notice that, as expected given the scale of the input size, when the GPU is used the execution time is reduced by three orders of magnitude. The first gVirtuS experiment is the host/afunix, i.e our GPU virtualization stack is used in an un-virtualized environment. Both the front end and the back end run on the same real machine and the communicator is implemented through Unix sockets. This test estimates the virtualization stack overhead without the virtualization itself, that is just considering the performance of the packed CUDA function calls. In the top chart as it is shown in Fig. 2, the difference between host/gpu and host/afunix is not noticeable

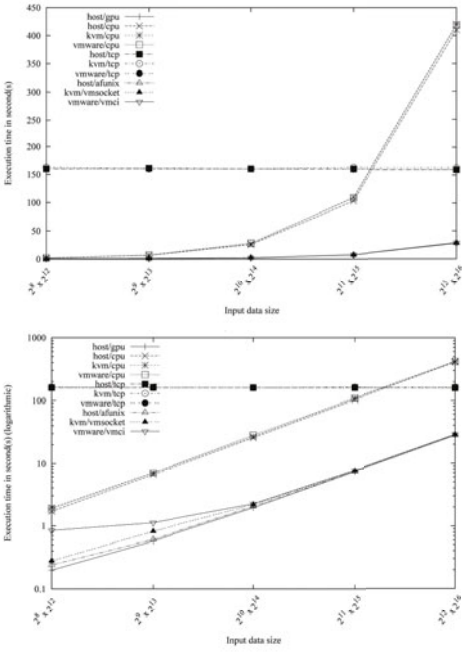

Fig. 2. Results of the ScalarProd performance test. Linear y-scale (top) and logarithmic y-scale (bottom).

on the linear scale (top), and the overhead is emphasized by the logarithmic scale (bottom). The problem size cut off value is near $2^9 \times 2^13$; for a larger size the overhead introduced by the virtualization stack, without considering the communication between virtual and real machines, can be neglected. The experiment host/tcp measures the impact of the standard tcp/ip stack communicator component. The communication between the front end and the back end exhibits low performance when compared to the results of the other communicators. Nevertheless, it is worth noting that the problem size cut occurs near $2^{11} \times 2^{15}$; for a larger size the GPU execution time using gVirtuS appears to be lower than the CPU time. The VMware/cpu and kvm/cpu experiments involve the virtualization. As reported above, we have used two different hypervisors in order to compare and contrast commercial vs open source virtualization solutions. To summarize, the VMware/cpu and kvm/cpu experiments demonstrate that there is no appreciable loss of efficiency in using virtualization with both hypervisors, as long as the computing time is concerned. The VMware/tcp and the kvm/tcp experiments describe the performance results obtained with a tcp based communicator; notice that in both cases the communication is the bottleneck. When the problem size is greater than the cut off value ($2^{11}x2^{15}$, Fig. 2), a deployment scenario where CPU based computing elements (CE) are connected to GPU cores provided CEs seems to be feasible. The VMware/vmci experiment reports the ScalarProd performance when the GPU is used through gVirtuS

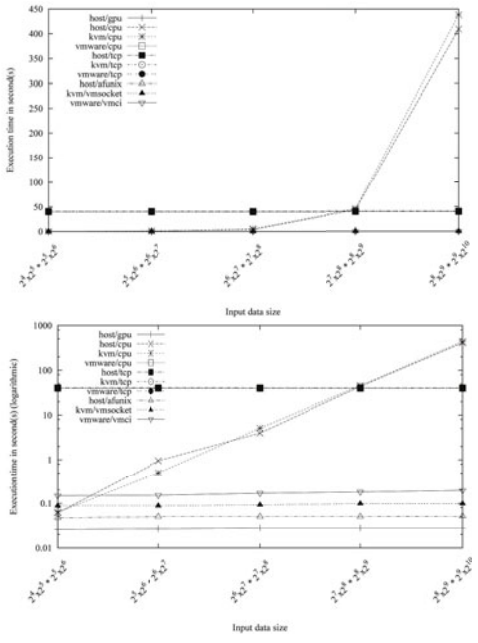

Fig. 3. Results of the MatrixMul performance test. Linear y-scale (top) and logarithmic y-scale (bottom)

with a communicator based on the VMCI using the VMware hypervisor. The experiment defined as kvm/vmsocket is referred to the performance evaluation of our vmSocket based communicator running a fully open source setup relying on the KVM/QEMU hypervisor. The difference in performance between the two setups can be assessed by looking at the results shown in the bottom side of Fig. 2. The best performance is achieved in the kvm/vmsocket experiment, that exhibits an efficiency close to that obtained by host/afunix. From the latter result we can reasonably conclude that, on this test problem, gVirtuS has an overhead near to the minimum and it shows the best virtual/real machine communication performance. Notice that the problem size cut off value is $2^{11}x2^{15}$ and when the size increases further both VMware/vmci and kvm/vmsocket perform in a very similar way. Fig. 3 shows the results of the MatrixMul program. The size of the test problems increases from $2^4 \times 2^5 * 2^5 \times 2^6$ (i.e. $m = 2^4$, $n = 2^5$, $p = 2^6$) to $2^8 \times 2^9 * 2^9 \times 2^{10}$ (i.e. $m = 2^8$, $n = 2^9$, $p = 2^{10}$). The host, VMware, kvm/cpu experiments have shown similar performance results. We note that the host, VMware, kvm/tpc experiments give evidence of a cut off size of $2^7 \times 2^8 * 2^8 \times 2^9$. Because of the very high efficiency of the matrix multiplication algorithm on the GPU, the use of the logarithm scale is needed in order to emphasize the behavior of VMware/vmci vs kvm/vmsocket. The bottom side of Fig. 3 shows that the kvm/vmsocket executes faster than VMware/vmci, confirming the conclusion drawn in the previous ScalProd test. Finally, Fig. 4 shows the results of

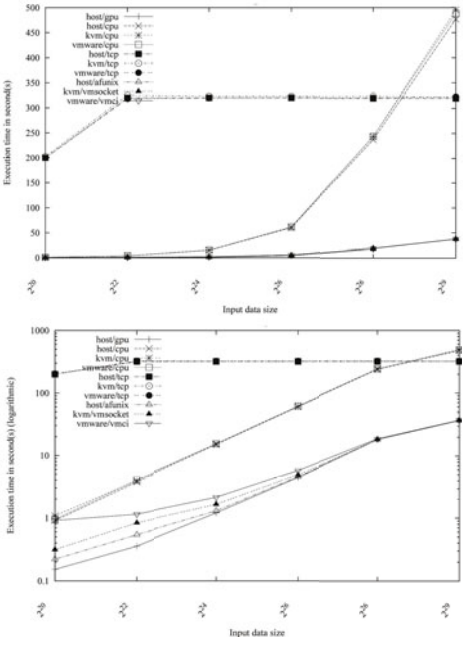

Fig. 4. Results of the Histogram performance test. Linear y-scale (top) and logarithmic y-scale (bottom).

the Hystogram program. This algorithm differs from the previous ones because it needs a synchronization and a reduction step among the concurrent threads in order to compute the final result. Even this benchmark program confirms that gVirtuS-kvm/vmsocket gives the best efficiency with the less impact respect to the raw host/gpu setup.

5 High Performance Computing Cloud Deployment Performance Evaluation

We have designed and implemented the gVirtuS GPU virtualization system with the primary goal of using GPUs in a private computing cloud for high performance scientific computing. We have set up a department prototypal cloud computing system leveraging on the Eucalyptus open source software. Eucalyptus, developed at the University of Santa Barbara in California [18], implements an Infrastructure as a Service (IaaS), as it is commonly referred to. Eucalyptus enables users familiar with existing Grid and HPC systems to explore new cloud computing functionalities while maintaining access to existing, familiar application development software and Grid middle- ware[17]. In our test aimed at assessing the performance of gVirtuS in a high performance computing cloud system, we have used an Intel based 12 computing nodes cluster, where each

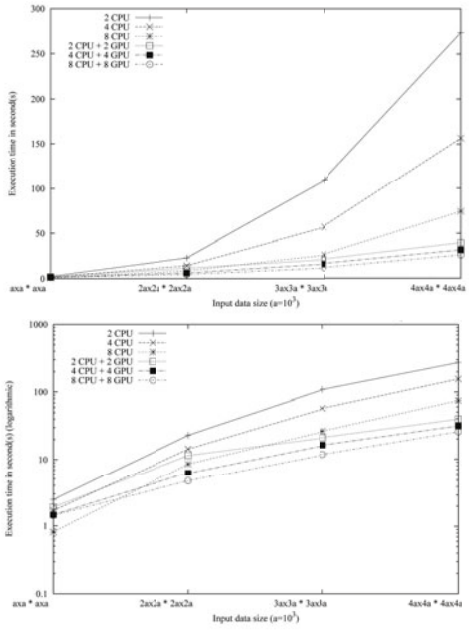

Fig. 5. The high performance computing cloud deployment experiment

node is equipped with a quad core 64 bit CPU and an nVIDIA GeForce GT 9400 video card with 16 CUDA cores and a memory of 1 Gbyte. The software stack as in the Tesla based workstation, i.e. Fedora 12 Linux operating system, nVIDIA CUDA drivers and Tools/SDK 2.3, and KVM-QEMU, was adopted. Moreover, we carried out an experiment focused on the gVirtuS behavior in a private cloud, where the GPUs are seen as virtual computing nodes building a virtual cluster dynamically deployed on a private cloud [12]. We developed an ad hoc benchmark software implementing a matrix multiplication algorithms. This software uses a classic memory distributed parallel approach. The first matrix is distributed by rows, the second one by columns, and each process has to perform a local matrix multiplication. MPICH 2[10] is message passing interface among processes, whereas each process uses the CUDA library to perform the local matrix multiplication. Fig. 5 shows the results of the performance test. We have used virtual clusters composed of 2,4 and 8 nodes computing nodes, respectively. The problem size increases in the range $[2E^3 \times 2E^3 * 2E^3 \times 2E^3$, $4E^3 \times 4E^3 * 4E^3 \times 4E^3]$, i.e. involving square matrices whose order varies from 2×10^3 to 4×10^3. The top side of Fig. 5 presents results obtained by running the MPI-based algorithm on the cluster. The bottom side of Fig. 5 gives a comprehensive summary of the tests of gVirtuS-based algorithm for the case CPU only and the case CUDA-GPU. These results show that the gVirtuS GPU virtualization and the related sharing system allows an effective exploitation of the computing power of the GPUs. We note that without such component the

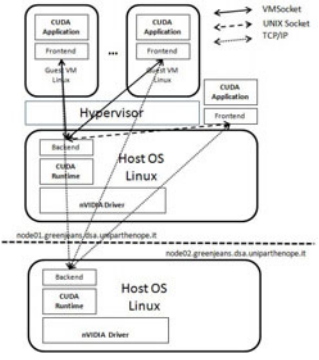

Fig. 6. The high performance computing cloud deployment and GPU virtualization/sharing schema

GPUs could not be seen by the virtual machine and it would be imposssible to run this experiment on a public or private cloud hosting an dynamic (on demand) virtual cluster. The Fig. 6 shows a possible deployment schema in a high performance cloud computing scenario. Virtual machines on computing nodes can use virtualized CUDA acceleration without any regards respect the fact the real machine is equipped or not with a CUDA device. Less GPU performance demanding VMs can share devices available on machines connected with a high performance local network. In a similar manner a high performance cloud computing infrastructure could be build on classic HPC clusters with some nodes dedicated to massive GPU acceleration with many CUDA enabled devices.

6 Comparison to Related Works

The GViM (GPU-accelerated Virtual Machines) [14] is a XEN-based system that provides some features resembling those of gVirtuS. GViM allows virtual machines to access a GPU through a communicator channel between a front end executed on the VM and a back end on the XEN Domain 0. vCUDA [19] is another GPGPU computing solution for virtual machines. It allows an application which is executed within a virtual machine to be accelerated by CUDA cores provided hardware. gVirtuS, GViM, and vCUDA use a similar approach based on the API call interception and redirection through a front end/back end communication system. This is a common choice in virtualization. The main difference lies in the use of the hypervisor and the communication technique. Both GViM, and vCUDA make use of XEN as hypervisor that executes CUDA drivers on the Domain 0. GViM uses the XenStore component to provide communication between the two faces of the CUDA split driver. The performance exhibited by GViM is mainly affected by the XenStore behavior, and in particular by its memory copy operation, as shown in [14]. vCUDA uses a XML-RPC to envelope the CUDA API calls and sends it out the virtual machine through

the frontend/backend architecture. This is a standard and commonly used approach, but it suffers a serious drawback in performance as demonstrated by the tests presented in [19]. Our solution differs from the latter two solutions in the chosen open source hypervisor, i.e. KVM; KVM is not a modified kernel, but a Linux loadable kernel module included in mainline Linux. This ensures future compatibility between the hypervisor and the CUDA drivers. Moreover, gVirtuS is completely independent of the hypervisor as we have proved by comparing and contrasting gVirtuS in a VMware setup and in a KVM setup. We also note that in presence of official nVIDIA CUDA drivers for XEN, it could work even on this hypervisor. Finally, gVirtuS relies on an actual high performance communication channel between the virtual and the real machines, thanks to our mvSocket component. Last but not least, the communicator interface is open and new communicators could be developed and plugged in. A direct compare and contrast among gVirtuS and other related systems performances will done as well the other systems will be released for public use. We also emphasize that neither GViM nor vCUDA are able to use a TCP based communicator for the deployment on asymmetric computing clusters. VMGL [15] is a XEN-based technology, GPGPU independent, aimed at using OpenGL on virtual machine through a specific management and monitor system (VMM). VMGL is dedicated to high performance visual computing and not to general purpose high performance scientific computing on GPUs, so it cannot be used to run CUDA applications on virtual machines. Even though the current version of gVirtuS does not provide CUDA-OpenGL interoperability, the two projects could be joined in order to fix this embarrassing gap.

7 Conclusions and Future Development

In this paper we have proposed the gVirtuS system, a GPU virtualization and sharing service. The main goal was to enable a virtual machine instance running in a high performance computing cloud to properly exploit the computing power of the nVIDIA CUDA system. In discussing the architecture of our framework, the adopted design and implementation solutions, and the key communication component vmSocket, we stressed the main features of gVirtuS, i.e. hypervisor independence, fully transparent behavior and especially its performance. We have reported the results of an extensive test process to assess how gVirtuS performs in different and realistic setups. We have also shown that our system can effectively operate even in a high performance computing cloud on a standard cluster powered by nVIDIA GPUs. Finally, gVirtuS has been successfully compared to other existing virtualization systems for GPUs. gVirtuS leverages on our previous works on high performance computing grids, and our interest in the apparently poor performing TCP communicator is related to other more complex deployment schemes. For example a private cloud could be set up on a massive multi core cluster, where to host general purpose virtual machine instances and GPU powered computing elements for compute-intensive scientific applications. A short term enhancement can be the implementation of OpenGL interoperability to integrate gVirtuS and VMGL for 3D graphics virtualization. Since the

vmSocket component of gVirtuS is able to allow very general connections, we are also investigating the possibility to integrate MPICH2 with vmSocket in order to implement a high performance message passing standard interface. In our vision, a vmSocket enhanced MPI could communicate with other virtual machine instances running on the same host, and increase the communication speed between virtual machines on different clusters using a suitable compression. This could foster the use of cloud technologies in scientific computing and transform the concept of e(lectronic)-science into the more practical and effective tool of e(lastic)e-science.

References

1. Gvirtus website, http://osl.uniparthenope.it/projects/gvirtus
2. Kvm website, http://www.linux-kvm.org
3. Nvidia cuda website, http://www.nVIDIA.com/object/cuda_home_new.html
4. Vmchannel website, http://www.linux-kvm.org/page/VMchannel_Requirements
5. Vmci website, http://pubs.VMware.com/vmci-sdk/
6. Vmware website, http://www.VMware.com/
7. Xen website, http://www.xen.org/
8. Armbrust, M., Fox, A., Griffith, R., Joseph, A.D., Katz, R.H., Konwinski, A., Lee, G., Patterson, D.A., Rabkin, A., Stoica, I., Zaharia, M.: Above the clouds: A berkeley view of cloud computing. Technical Report UCB/EECS-2009-28, Electrical Engineering and Computer Sciences University of California at Berkeley (February 2009)
9. Barham, P., Dragovic, B., Fraser, K., Hand, S., Harris, T., Ho, A., Neugebauer, R., Pratt, I., Warfield, A.: Xen and the art of virtualization. In: SOSP 2003: Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles, pp. 164–177 (2003)
10. Buntinas, D., Mercier, G., Gropp, W.: Design and evaluation of nemesis, a scalable, low-latency, message-passing communication subsystem. In: CCGRID 2006: Proceedings of the Sixth IEEE International Symposium on Cluster Computing and the Grid, Washington, DC, USA, pp. 521–530. IEEE Computer Society, Los Alamitos (2006)
11. Che, S., Li, J., Sheaffer, J.W., Skadron, K., Lach, J.: Accelerating Compute-Intensive Applications with GPUs and FPGAs. In: Symposium on Application Specific Processors, SASP 2008, pp. 101–107 (2008)
12. Fenn, M., Murphy, M.A., Goasguen, S.: A study of a kvm-based cluster for grid computing. In: ACM Southeast Regional Conference (2009)
13. Foster, I., Zhao, Y., Raicu, I., Lu, S.: Cloud computing and grid computing 360-degree compared
14. Gupta, V., Gavrilovska, A., Schwan, K., Kharche, H., Tolia, N., Talwar, V., Ranganathan, P.: Gvim: Gpu-accelerated virtual machines. In: HPCVirt 2009: Proceedings of the 3rd ACM Workshop on System-level Virtualization for High Performance Computing, pp. 17–24. ACM, New York (2009)
15. Andrs Lagar-cavilla, H., Satyanarayanan, M.: Vmm-independent graphics acceleration. In: Proceedings of VEE 2007. ACM Press, New York (2007)
16. Tim Grance Mell, P.: The nist definition of cloud computing. Technical Report Version 15, National Institute of Standards and Technology (July 2009)

17. Nurmi, D., Wolski, R., Grzegorczyk, C., Obertelli, G., Soman, S., Youseff, L., Zagorodnov, D.: Eucalyptus: an open-source cloud computing infrastructure. Journal of Physics: Conference Series 180(1), 012051 (2009)
18. Nurmi, D., Wolski, R., Grzegorczyk, C., Obertelli, G., Soman, S., Youseff, L., Zagorodnov, D.: The eucalyptus open-source cloud-computing system. In: IEEE International Symposium on Cluster Computing and the Grid, pp. 124–131 (2009)
19. Shi, L., Chen, H., Sun, J.: vcuda: Gpu accelerated high performance computing in virtual machines. In: IPDPS 2009: Proceedings of the 2009 IEEE International Symposium on Parallel&Distributed Processing, Washington, DC, USA, pp. 1–11. IEEE Computer Society, Los Alamitos (2009)
20. Tarditi, D., Puri, S., Oglesby, J.: Accelerator: using data parallelism to program gpus for general-purpose uses. In: ASPLOS-XII: Proceedings of the 12th International Conference on Architectural Support for Programming Languages and Operating Systems, pp. 325–335. ACM, New York (2006)
21. Wang, J., Wright, K.l., Gopalan, K.: Xenloop: a transparent high performance inter-vm network loopback. In: Proceedings of the 17th International Symposium on High Performance Distributed Computing (HPDC 2008), pp. 109–118 (2008)