Local and Remote GPUs Perform Similar with EDR 100G InfiniBand

Reaño, C., Silla, F., Shainer, G., & Schultz, S. (2015). Local and Remote GPUs Perform Similar with EDR 100G InfiniBand. In *16th ACM International Middleware Conference (Middleware)* ACM.
<https://dl.acm.org/doi/proceedings/10.1145/2814576>

Published in:
16th ACM International Middleware Conference (Middleware)

Document Version:
Peer reviewed version

Queen's University Belfast - Research Portal:
[Link to publication record in Queen's University Belfast Research Portal](#)

Publisher rights
Copyright 2015 ACM. This work is made available online in accordance with the publisher's policies. Please refer to any applicable terms of use of the publisher.

General rights
Copyright for the publications made accessible via the Queen's University Belfast Research Portal is retained by the author(s) and / or other copyright owners and it is a condition of accessing these publications that users recognise and abide by the legal requirements associated with these rights.

Take down policy
The Research Portal is Queen's institutional repository that provides access to Queen's research output. Every effort has been made to ensure that content in the Research Portal does not infringe any person's rights, or applicable UK laws. If you discover content in the Research Portal that you believe breaches copyright or violates any law, please contact openaccess@qub.ac.uk.

Local and Remote GPUs Perform Similar with EDR 100G InfiniBand

Carlos Reaño and Federico Silla
Universitat Politècnica de València
46022 València, Spain
carregon@gap.upv.es, fsilla@disca.upv.es

Gilad Shainer and Scot Schultz
Mellanox Technologies
Sunnyvale, CA 94085, USA
shainer@mellanox.com, scots@mellanox.com

ABSTRACT

The use of graphics processing units (GPUs) to accelerate some portions of applications is widespread nowadays. To avoid the usual inconveniences associated with these accelerators (high acquisition cost, high energy consumption, and low utilization), one possible solution is sharing them among several nodes in the cluster. Several years ago, remote GPU virtualization middleware systems appeared to implement this solution. Although these systems tackled the aforementioned inconveniences, their performance was usually impaired by the low bandwidth attained by the underlying network. However, the recent advances in InfiniBand fabrics have changed this trend. In this paper we analyze how the high bandwidth provided by the new EDR 100G InfiniBand fabric allows remote GPU virtualization middleware systems not only to perform very similar to local GPUs, but also to improve overall performance for some applications.

Keywords

InfiniBand, GPUs, CUDA, GPU virtualization, performance

1. INTRODUCTION

Many applications rely on the use of graphics processing units (GPUs) to accelerate their execution. This is the case, for instance, of applications from domains such as data analysis (Big Data) [25], chemical physics [18], computational algebra [26], image analysis [27], finance [23], and biology [1].

The use of GPUs as accelerators presents, however, several side effects. For instance, including GPUs in all the nodes of a cluster increases acquisition costs as well as those of maintenance and space requirements. Moreover, although GPUs present a favorable performance/power ratio while being used, they still require non-negligible amounts of energy while idle. Furthermore, these increased costs may not be easily amortized because few applications present enough computational concurrency to keep GPUs in use all the time, thus preventing high GPU utilization rates.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from Permissions@acm.org.

Middleware Industry'15, December 07-11, 2015, Vancouver, BC, Canada

© 2015 ACM. ISBN 978-1-4503-3727-4/15/12 ...\$15.00.

DOI: <http://dx.doi.org/10.1145/2830013.2830015>

In a similar way to virtual machines, the use of virtual GPUs may overcome the concerns associated with the use of real GPU devices. The remote GPU virtualization technique allows an application being executed in a computer that does not have installed a GPU to transparently make use of an accelerator installed in other node of the cluster.

The use of remote GPUs may seem to be a senseless idea. However, it provides several benefits to applications and data center administrators. For example, the use of remote GPUs allows non-MPI (Message Passing Interface) applications to use a number of GPUs larger than those that can fit into a single server box. In this way, non-distributed applications may leverage all the GPUs present in the cluster, thus boosting their performance beyond the limits that the use of local GPUs may report [20]. On the other hand, the remote GPU virtualization mechanism easily allows a cluster that does not include GPUs to be extended with GPUs just by attaching one or several GPU-based servers to it. These GPU-servers would provide GPU services to the rest of the nodes of the cluster. In this regard, cluster nodes that cannot accept GPUs because of power supply limitations or because GPUs do not physically fit into them would be able to transparently execute GPU-accelerated applications with a small hardware investment. Other benefits of virtual GPUs are related to a reduced overall execution time and consumed energy, given that the use of virtual GPUs allows the real devices to be concurrently shared among several applications. In this regard, in the context of batches of jobs managed by schedulers such as SLURM, GPU sharing provides the net effect of reducing total execution time up to 40% [8] as well as decreasing the total energy required for completing execution in a similar percentage [22]. These reductions are the consequence of an increased GPU utilization rate, which is usually low when virtual GPUs are not used.

Despite the benefits reported by the remote GPU virtualization mechanism, its use also presents some drawbacks. The main concern of this technique is that GPUs are accessed across a network, what has traditionally reduced the performance of data movements from/to the GPU with respect to the local case. The main reason for the lower performance is that with local GPUs data has to pass only through the local PCIe link. On the contrary, with remote GPUs data has to traverse the local PCIe link, the entire network, and the remote PCIe link. Additionally, given that the bandwidth of the network has traditionally been lower than the one of the PCIe link, performance when using remote GPUs has traditionally been penalized.

Figure 1: Comparison between the theoretical bandwidth of different versions of PCI Express x16 and those of commercialized InfiniBand fabrics.

However, remote GPU virtualization may experience a noticeable reduction in their overhead with respect to the use of local GPUs when high performance interconnects are used. In this regard, recent advances in networking technologies have considerably reduced the performance gap between the intra-node PCIe link and the inter-node fabric. For example, as depicted in Figure 1, the theoretical throughput of the latest available versions of PCIe and InfiniBand (IB) are relatively close: 15.75 GB/s for PCIe 3.0 x16 vs. 12.5 GB/s for EDR 100G InfiniBand using Mellanox adapters. Furthermore, when considering the effective bandwidth of these technologies instead of the theoretical numbers, EDR IB presents better results, as shown later.

In this manner, the main contribution of this paper is showing how this high effective bandwidth provided by EDR IB allows remote GPU virtualization middleware systems not only to perform very similar to local GPU-accelerators, but also to improve the performance for some applications.

The rest of the paper is organized as follows. Section 2 further investigates the effective bandwidth of EDR IB networks and PCIe 3.0 when used by the latest NVIDIA GPUs. In Section 3 a brief description of various remote GPU virtualization middleware systems is presented. Section 4 provides a performance evaluation of the rCUDA middleware using both popular benchmarks and applications. Finally, Section 5 summarizes the main conclusions of this work.

2. MOTIVATION

As shown in Figure 1, the theoretical bandwidth of EDR IB and that of PCIe 3.0 are relatively close. However, it is well known that the actual performance of any interconnect largely depends on the ability of the upper software layers to obtain as much bandwidth as possible from it. Thus, the effective performance of an interconnect is usually lower than the theoretical one. The real concern, therefore, is not about the theoretical numbers depicted in Figure 1 but about real performance, that is, whether real bandwidth numbers help to reduce the performance difference among the external IB fabric and the internal PCIe or, on the contrary, make that difference larger, thus increasing the penalty of remote GPU virtualization techniques.

To address this concern, in this section we compare the EDR IB fabric against the PCIe 3.0 link, when used by their respective upper software layers: the network driver in the case of IB and the CUDA driver, along with an NVIDIA GPU, in the case of the PCIe 3.0. This analysis will expose the maximum performance that might be achieved by a remote GPU virtualization solution and, additionally, will put the performance of such a virtualization solution into the right perspective.

The testbed used in this analysis consists of two 1027GR-

(a) Performance of the NVIDIA Tesla K40 GPU.

(b) Performance of the NVIDIA Tesla K80 GPU.

(c) Performance of K80 normalized to that of K40.

Figure 2: Bandwidth test for copies between host memory and GPU memory using pinned and pageable host memory. H2D refers to the copies from host to the GPU, whereas D2H refers to the opposite direction.

TRF Supermicro nodes featuring two Intel Xeon E5-2620v2 processors (Ivy Bridge) operating at 2.1 GHz and 32 GB of DDR3 memory at 1600 MHz. The computers also include an EDR IB adapter. One of the nodes has installed an NVIDIA Tesla K40 GPU and also an NVIDIA Tesla K80 GPU. Linux CentOS 6.4 was used along with CUDA 7.0 (NVIDIA driver 340.46) and Mellanox OFED 2.4-1.0.4 (IB drivers and administrative tools).

The test servers used are NUMA machines. NUMA effects matter for the performance evaluation presented in this paper. For this reason, the IB card and the K40 GPU are connected to the same PCIe root (i.e., processor 0) whereas the K80 GPU is connected to processor 1. For all the experiments shown in this paper, processes and memory buffers are appropriately bound either to processor 0 or 1. Additionally, the persistence mode of the NVIDIA GPUs was enabled in order to obtain the best performance.

Figure 2 shows the performance attained by the NVIDIA K40 and K80 GPUs when the `bandwidthTest` benchmark from the NVIDIA samples [14] is used. It can be seen in Figure 2(a) that the bandwidth achieved by copies using pinned and pageable host memory is very different. Actually, copies using pinned memory achieve up to four times the performance of copies using pageable memory. Moreover, it is interesting to notice that even for the largest performance attained by the K40 GPU, maximum effective bandwidth is

about 35% lower than the PCIe 3.0 theoretical bandwidth. Furthermore, applications usually leverage pageable memory. Hence, in practice the performance of memory copies to/from the K40 GPU is almost six times lower than the theoretical performance of PCIe 3.0. This fact supports our initial assumption that effective bandwidth should be considered instead of theoretical numbers. Regarding the performance of the K80 GPU shown in Figure 2(b), it can be seen that similar conclusions can be reached¹. Nevertheless, as Figure 2(c) shows, the performance of the K80 GPU is lower than that of the K40 one. For this reason, in the rest of the paper we will only consider the use of the K40 GPU, because it represents a harder burden for our study.

Once the performance of the PCIe 3.0 link, along with NVIDIA GPUs, has been analyzed, we continue our study with the performance of the EDR IB network. Figure 3 presents the performance attained by this interconnect when the `ib_write_bw`, `ib_read_bw`, and `ib_send_bw` benchmarks from the Mellanox OFED are used. These tests measure the bandwidth when copying different data sizes using the channel semantics (i.e. send/receive verbs not using RDMA, `ib_send_bw` benchmark), and the memory semantics (i.e. RDMA read and write, `ib_read_bw` and `ib_write_bw` benchmarks, respectively). First, Figure 3(a) presents the performance of the `ib_send_bw` benchmark and compares it with the other two IB benchmarks. It can be seen that the three benchmarks provide very similar results. Only for copy sizes below 1K byte there is a maximum of 5% of difference among them. On the other hand, Figure 3(b) compares the performance of the `ib_write_bw` benchmark with that of the K40 GPU for copies H2D using pageable and pinned host memory. The left Y axis shows absolute bandwidth numbers whereas the right Y axis shows the performance of the EDR network normalized to that of the K40 (in logarithmic scale). It can be seen in Figure 3(b) that the performance of the EDR IB fabric is almost 100 times larger than that of the K40 using pageable memory for small copy sizes, whereas for larger copy sizes the K40 GPU is about 4 times slower. In a similar way, Figure 3(c) depicts a performance comparison involving the `ib_read_bw` benchmark as well as D2H memory copies using pageable and pinned memory. Again, the EDR IB network fabric performs better (between 100x and 5x than the K40 GPU when pageable memory is used).

As a summary of the analysis presented in this section, when the effective bandwidth numbers of PCIe 3.0 and EDR IB interconnects are considered instead of the theoretical values, the main conclusion is that the EDR IB network fabric provides higher bandwidth than the latest NVIDIA GPUs. This difference in performance provides an interesting conclusion: the effective bandwidth of the external interconnect (EDR IB) is noticeably larger than the maximum bandwidth attained by the internal interconnect (PCI 3.0 x16 + CUDA). This fact introduces new possibilities for remote GPU virtualization solutions, which so far were strongly limited by the lower performance of the external interconnect. This conclusion is the main motivation for the study presented in this paper about the performance of remote GPU virtualization middleware systems.

¹Notice that only one of the two devices available in the K80 GPU is used in these experiments. When both devices are concurrently leveraged, attained bandwidth is increased. Analyzing the concurrent use of the second device is left for future work.

(a) Comparison among send, read, and write benchmarks.

(b) Comparison among IB RDMA write and H2D CUDA copies.

(c) Comparison among IB RDMA read and D2H CUDA copies.

Figure 3: Performance of the InfiniBand bandwidth tests, compared to the performance of the NVIDIA K40 GPU.

3. REMOTE GPU VIRTUALIZATION

Frameworks such as CUDA [12] assist programmers in using GPUs for general-purpose computing. In addition, several remote GPU virtualization solutions exist for this framework, such as GridCuda [9], DS-CUDA [16], GVirtuS [5], vCUDA [21], and rCUDA [17].

Figure 4 depicts the architecture underlying most of these virtualization solutions, which follow a client-server distributed approach. The client part of the middleware is installed in the cluster node executing the application requesting GPU services, whereas the server side runs in the computer owning the actual GPU. Generally, the client middleware offers the same application programming interface (API) as does the NVIDIA CUDA Runtime API [13]. In this manner, the client receives a CUDA request from the accelerated application and appropriately processes and forwards it to the remote server. In the server node, the middleware receives the request and interprets and forwards it to the GPU, which completes the execution of the request and provides the execution results to the server middleware. In turn, the server sends back the results to the client middle-

Figure 4: Architecture used in remote GPU virtualization.

ware, which forwards them to the initial application, which is not aware that its request has been served by a remote GPU instead of a local one.

Current virtualization frameworks provide different features. For example, DS-CUDA supports CUDA 4.1 and includes specific communication support for IB, although it presents several severe limitations like not allowing data transfers with pinned memory. Regarding the vCUDA technology, it supports the old CUDA 3.2 version and implements an unspecified subset of the CUDA runtime API. Similarly, the gVirtuS approach is based on the old CUDA 2.3 version and implements only a small portion of the runtime API. Furthermore, it provides TCP/IP communications between clients and servers, thus reducing the effective bandwidth in networks such as IB. Regarding GridCuda, it supports CUDA 3.2, although the public information available is fuzzy and it has no public version that may be used for testing and comparison. In the case of rCUDA, it is binary compatible with CUDA 7.0 and implements the entire CUDA Runtime and Driver APIs (except for graphics functions). It also provides support for the libraries included within CUDA (cuBLAS, cuFFT, etc). Additionally, it supports several underlying interconnection technologies by making use of a set of runtime-loadable, network-specific communication modules (currently TCP/IP and IB). Furthermore, as shown in [19], it outperforms the rest of available remote GPU virtualization solutions. For these reasons, we use this middleware in our study. Next we present some additional information on rCUDA relevant to the work presented in this paper.

Firstly, independently from the exact network used, data exchange between rCUDA clients and servers is pipelined in order to attain high performance. Internal pipeline buffers within rCUDA use preallocated pinned memory given the higher throughput of this type of memory [17].

Regarding the IB communication module, it is based on the IB Verbs (IBV) API. This API offers two communication mechanisms: the channel semantics and the memory semantics. The former refers to the standard send/receive operations typically available in any networking library, while the latter offers RDMA operations where the initiator of the operation specifies both the source and destination of a data transfer, resulting in zero-copy transfers with minimum involvement of the CPUs. rCUDA employs both IBV mechanisms, selecting one or the other depending on the exact communication to be carried out.

(a) Copies from pageable host memory to GPU memory.

(b) Copies from GPU memory to pageable host memory.

(c) Copies from pinned host memory to GPU memory.

(d) Copies from GPU memory to pinned host memory.

Figure 5: Bandwidth test for copies between host memory and GPU memory using CUDA and the rCUDA middleware over the EDR InfiniBand fabric.

4. rCUDA EXPERIMENTS

This section presents the impact that the new EDR IB interconnect has on the performance of the rCUDA middleware. For this purpose, we first analyze the maximum bandwidth attained for memory copies between host memory and remote GPU memory. Next, we use the Rodinia benchmark suite [4] to characterize how several application kernels perform in this virtualized scenario. Finally, we make use of seven different applications to analyze how the improved features of the new interconnect influence the performance of applications using rCUDA.

4.1 Bandwidth analysis

Figure 5 presents a comparison between the bandwidth attained by CUDA when using the local K40 GPU, as in the traditional way, and the performance achieved when using the rCUDA middleware with a remote K40 GPU. The testbed used is the same as in Section 2. The `bandwidthTest`

benchmark from the NVIDIA samples is used. Figures 5(a) and 5(b) use pageable host memory whereas Figures 5(c) and 5(d) use pinned host memory. It can be seen that when pageable host memory is used, the bandwidth achieved when using a remote GPU is higher than that obtained for a local GPU for copy sizes larger than 100K bytes. Actually, for the H2D direction, the rCUDA middleware achieves almost 3 times the bandwidth attained by CUDA. In the case of the D2H direction, rCUDA doubles the performance of CUDA. This is a well-known effect thoroughly described in previous works on rCUDA [17] and is due to the use of an efficient pipelined communication based on the use of internal pre-allocated pinned memory buffers. On the other hand, when pinned memory is considered, rCUDA is able to achieve more than 90% of the bandwidth attained by CUDA for the largest copy sizes. Nevertheless, remember than applications usually employ pageable memory instead of pinned memory.

As a brief summary of the results presented in Figure 5, the main conclusion is that CUDA applications may experience some additional acceleration when using the rCUDA middleware due to the higher bandwidth used to move data to/from the GPU. Nevertheless, in addition to bandwidth, other issues should also be considered, as it will be shown in next subsection.

4.2 The Rodinia Benchmark Suite

Rodinia [4] is a popular benchmark suite for heterogeneous computing aimed to help architects study platforms such as GPUs. It includes applications and kernels which target multi-core CPU and GPU platforms. The Rodinia benchmarks cover a wide range of parallel communication patterns and synchronization techniques, which we consider useful for an initial study.

For the experiments shown in this section, we have used the version 3.0 of the Rodinia benchmark, following the instructions inside each benchmark for running them. Figures 6(a), 6(b), and 6(c) compare the execution time of several of the benchmarks included in this suite (for a complete description of each of these benchmarks please refer to [4]). Figure 6(a) presents results for benchmarks whose execution time is under 1 second. Figure 6(b) presents results for benchmarks that require less than 5 seconds to complete their execution and Figure 6(c) is devoted to a benchmark presenting a much longer execution time. Hence, benchmarks with very different execution times are considered in this study. It can be seen in these three figures that the execution of these benchmarks with the rCUDA middleware is, in general, faster than with CUDA. Figure 6(d) summarizes, in terms of percentage, the difference of time between rCUDA and CUDA. This figure shows that for short benchmarks, the remote case is in general noticeably faster than the local case. However, for the medium benchmarks there is not such a predictable improvement. Finally, for the long benchmark analyzed, the remote case performs better than the local case.

One may think that the better performance of rCUDA is due to the higher bandwidth attained for copies using pageable memory. However, a deeper profiling revealed that some of the analyzed codes have synchronization points, such as calls to `cudaDeviceSynchronize` or `cudaStreamWaitEvent`, that take more time when using CUDA than when using the rCUDA middleware. For instance, a call to `cudaDe-`

(a) Execution time of short benchmarks.

(b) Execution time of medium benchmarks.

(c) Execution time of a long benchmark.

(d) Difference in execution time of rCUDA with respect to CUDA.

Figure 6: Execution time of several Rodinia benchmarks using CUDA and the rCUDA middleware over EDR InfiniBand.

`viceSynchronize` takes about 530 microseconds in CUDA, whereas it takes only about 40 microseconds in the rCUDA. The reason for this large difference lies in the internal algorithm used in the rCUDA middleware to determine the finalization of the CUDA tasks, which benefits rCUDA in these short benchmarks [20]. Nevertheless, the time saved in synchronization points in rCUDA does not fully explain the shorter execution time of these benchmarks. One additional factor that affects execution time is the network polling interval (i.e., the frequency used to poll the network for work completions). This interval in the rCUDA implementation is lower than the default polling interval used by CUDA to poll the PCIe link, as demonstrated in [20]. Thus, for short benchmarks where the aggregated weight of these

small waits becomes a large fraction of the total execution time, the net result is that rCUDA performs better than CUDA.

In summary, there are many different factors that simultaneously contribute to increase and reduce the overhead of rCUDA with respect to CUDA. Hence, for each of the benchmarks executed, the exact combination of these factors results in a better or worse execution time. Nevertheless, as shown in Figure 6, the average overhead of rCUDA with respect to CUDA is about 5% (considering only those cases where rCUDA increases total execution time).

4.3 Production Applications

This section completes the analysis of the performance of the rCUDA middleware by applying it to 7 production applications²: CUDASW++, GPU-LIBSVM, LAMMPS, GPU-BLAST, MAGMA, CUDA-MEME, and BarraCUDA.

CUDASW++ [11] is a bioinformatics software for Smith-Waterman protein database searches that takes advantage of the massively parallel CUDA architecture of NVIDIA Tesla GPUs to perform sequence searches. We have used its last release, version 3.1, for our study, along with the Latest Swiss-Prot database and a query with 5,478 sequences. Both the database and the query are available in the application’s website: <http://cudasw.sourceforge.net>.

GPU-LIBSVM [2] is an integrated software that supports vector classification, (C-SVC, nu-SVC), regression (epsilon-SVR, nu-SVR) and distribution estimation (one-class SVM). In addition, it supports multi-class classification. For our experiments, we have used version 3.18, and the input data included in the package. More specifically, we have scaled the available training data, without the use of the shrinking heuristics and utilizing a 10-fold cross validation mode.

LAMMPS [3] is a classic molecular dynamics simulator that can be used to model atoms or, more generically, as a parallel particle simulator at the atomic, mesoscopic, or continuum scale. For the test in this work, we use the release from May 15, 2015, and the benchmark “in.eam” installed with the application. We run the benchmark with one processor and one GPU, scaling by a factor of 5 in all three dimensions (i.e., a problem size of 4 million atoms).

GPU-BLAST [24] has been designed to accelerate the gapped and ungapped protein sequence alignment algorithms of the NCBI-BLAST implementation using GPUs. We utilize release 1.1, following the installation instructions for sorting a database and creating a GPU database. To search the database, we then use a query with 1,400 sequences that comes with the application package.

The MAGMA [7] project aims to develop a dense linear algebra library similar to LAPACK but for heterogeneous/hybrid architectures (current multi-core+GPU systems). We have run the DLARFB tests, which applies a real block reflector H or its transpose H^H to an M by N matrix C , from the left. The range of matrix sizes used ($M=N=K$) was from 1,088 to 4,160 in increments of 1,024.

CUDA-MEME [10] is a parallel formulation and implementation of the MEME motif discovery algorithm using CUDA. Version 3.0.15 was used in our study along with the testcases available in the application website, choosing OOPS model for motif distribution, with 500 sequences using DNA alphabet, and 0.5 million of sites for each motif.

²All the applications have been extracted from the NVIDIA list of GPU applications [15].

Figure 7: Execution time of several applications using CUDA and the rCUDA middleware over the EDR InfiniBand fabric.

The aim of the BarraCUDA [6] project is to develop a sequence mapping software that utilizes the massive parallelism of GPUs to accelerate the inexact alignment of short sequence reads to a particular location on a reference genome. We have used a simulation of the most current genome build called “Illumina” as database, and a query using a method called “STAMPY”. The read length was 37.

Figures 7(a) and 7(b) depict a comparison between the execution time of these applications in the local and remote scenarios. In a similar way to the Rodinia benchmarks, these applications have been selected because they present a wide range of execution times. Figure 7(c) presents the overhead of rCUDA with respect to CUDA. It can be seen that, in general, the overhead is lower than 3%. Moreover, some of the applications present small improvements. Again, a deeper profiling reveals similar conclusions as in Section 4.2. Therefore, as also explained in that section, the exact reasons for the increase or decrease of execution time when using rCUDA depends on each application. Due to the lack of space, it is not possible to detail them in this paper. Nevertheless, the general reasons lay on the same basis as the ones in Section 4.2.

5. CONCLUSIONS

In this paper the impact of EDR IB on the rCUDA middleware has been investigated. The main motivation for this study is the higher bandwidth of the EDR IB network fabric with respect to the bandwidth attained by current NVIDIA GPUs leveraging the PCIe 3.0 link. In addition to basic bandwidth tests, the remote GPU virtualization rCUDA middleware has been analyzed by using 20 different benchmarks from the Rodinia suite and 7 production applications selected from the NVIDIA list of GPU applications.

The main conclusion from this work is the noticeable reduction of the overhead experienced by remote GPU virtualization solutions, generally speaking, and more specifically by the rCUDA middleware. In this regard, for most applications the overhead is below 5%. Furthermore, some applications experience a reduction in their execution time with respect to the local CUDA case. We plan to further investigate the reasons for this behavior in future work.

6. ACKNOWLEDGMENTS

This work was funded by the Generalitat Valenciana under Grant PROMETEOII/2013/009 of the PROMETEO program phase II. Authors are also grateful for the generous support provided by Mellanox Technologies and the equipment donated by NVIDIA Corporation.

7. REFERENCES

- [1] P. K. Agarwal, S. Hampton, J. Poznanovic, A. Ramanathan, S. R. Alam, and P. S. Crozier. Performance modeling of microsecond scale biological molecular dynamics simulations on heterogeneous architectures. *Concurrency and Computation: Practice and Experience*, 25(10):1356–1375, 2013.
- [2] A. Athanasopoulos, A. Dimou, V. Mezaris, and I. Kompatsiaris. GPU Acceleration for Support Vector Machines. In *12th International Workshop on Image Analysis for Multimedia Interactive Services (WIAMIS 2011)*, Apr 2011.
- [3] W. M. Brown, A. Kohlmeyer, S. J. Plimpton, and A. N. Tharrington. Implementing molecular dynamics on hybrid high performance computers: Particle-particle particle-mesh. *Computer Physics Communications*, 183(3):449 – 459, 2012.
- [4] S. Che, M. Boyer, J. Meng, D. Tarjan, J. Sheaffer, S.-H. Lee, and K. Skadron. Rodinia: A benchmark suite for heterogeneous computing. In *Workload Characterization, 2009. IISWC 2009. IEEE International Symposium on*, pages 44–54, Oct 2009.
- [5] G. Giunta, R. Montella, G. Agrillo, and G. Coviello. A GPGPU transparent virtualization component for high performance computing clouds. In *Euro-Par 2010 - Parallel Processing*, 2010.
- [6] L. H and D. R. Fast and accurate short read alignment with burrows-wheeler transform. *Bioinformatics*, 25:1754–1760, 2009.
- [7] Innovative Computing Laboratory, University of Tennessee. MAGMA: Matrix Algebra on GPU and Multicore Architectures, 2014.
- [8] S. Iserte, A. Castello, R. Mayo, E. Quintana-Orti, F. Silla, J. Duato, C. Reaño, and J. Prades. SLURM Support for Remote GPU Virtualization: Implementation and Performance Study. In *26th Symposium on Computer Architecture and High Performance Computing (SBAC-PAD)*, Oct 2014.
- [9] T. Y. Liang and Y. W. Chang. GridCuda: A Grid-Enabled CUDA Programming Toolkit. In *2011 IEEE Workshops of International Conference on Advanced Information Networking and Applications (WAINA)*, pages 141–146, 2011.
- [10] Y. Liu, B. Schmidt, W. Liu, and D. L. Maskell. CUDA-MEME: Accelerating motif discovery in biological sequences using CUDA-enabled graphics processing units. *Pattern Recognition Letters*, 31(14):2170 – 2177, 2010.
- [11] Y. Liu, A. Wirawan, and B. Schmidt. CUDASW++ 3.0: accelerating smith-waterman protein database search by coupling CPU and GPU SIMD instructions. *BMC Bioinformatics*, 14(1), 2013.
- [12] NVIDIA. *CUDA C Programming Guide 7.0*, 2015.
- [13] NVIDIA. *CUDA Runtime API 7.0*, 2015.
- [14] NVIDIA. *CUDA Samples Reference Manual 7.0*, 2015.
- [15] NVIDIA. GPU Applications. <http://www.nvidia.com/object/gpu-applications.html>, 2015.
- [16] M. Oikawa, A. Kawai, K. Nomura, K. Yasuoka, K. Yoshikawa, and T. Narumi. DS-CUDA: A Middleware to Use Many GPUs in the Cloud Environment. In *Proceedings of the 2012 SC Companion: High Performance Computing, Networking Storage and Analysis, SCC '12*, pages 1207–1214, 2012.
- [17] A. J. Peña, C. Reaño, F. Silla, R. Mayo, E. S. Quintana-Orti, and J. Duato. A complete and efficient CUDA-sharing solution for HPC clusters. *Parallel Computing*, 40:574–588, 12/2014 2014.
- [18] D. P. Playne and K. A. Hawick. Data parallel three-dimensional cahn-hilliard field equation simulation on GPUs with CUDA. In *PDPTA*, pages 104–110, 2009.
- [19] C. Reaño and F. Silla. A performance comparison of CUDA remote GPU virtualization frameworks. In *IEEE Cluster Conference*, Sept 2015.
- [20] C. Reaño, F. Silla, A. Castello, A. J. Peña, R. Mayo, E. S. Quintana-Orti, and J. Duato. Improving the user experience of the rCUDA remote GPU virtualization framework. *Concurrency and Computation: Practice and Experience*, 2014.
- [21] L. Shi, H. Chen, and J. Sun. vCUDA: GPU accelerated high performance computing in virtual machines. In *IEEE International Symposium on Parallel & Distributed Processing (IPDPS) 2009*, pages 1–11, 2009.
- [22] F. Silla. rCUDA: Virtualizing CPUs to reduce cost and improve performance. <https://stacresearch.com/fall2014LON>. STAC Summit. London, UK. October 30, 2014. Last accessed: 2015 July 26.
- [23] V. Surkov. Parallel option pricing with fourier space time-stepping method on graphics processing units. *Parallel Computing*, 36(7):372 – 380, 2010. Parallel and Distributed Computing in Finance.
- [24] P. D. Vouzis and N. V. Sahinidis. GPU-BLAST: Using graphics processors to accelerate protein sequence alignment. *Bioinformatics*, 2010.
- [25] H. Wu, G. Damos, T. Sheard, M. Aref, S. Baxter, M. Garland, and S. Yalamanchili. Red Fox: An Execution Environment for Relational Query Processing on GPUs. In *Proceedings of Annual IEEE/ACM International Symposium on Code Generation and Optimization, CGO '14*, pages 44:44–44:54. ACM, 2014.
- [26] I. Yamazaki, T. Dong, R. Solca, S. Tomov, J. Dongarra, and T. Schulthess. Tridiagonalization of a dense symmetric matrix on multiple GPUs and its application to symmetric eigenvalue problems. *Concurrency and Computation: Practice and Experience*, 26(16):2652–2666, 2014.
- [27] D. Yuancheng Luo. Canny edge detection on NVIDIA CUDA. In *Computer Vision and Pattern Recognition Workshops, 2008. CVPRW '08. IEEE Computer Society Conference on*, pages 1–8. IEEE, 2008.