On the Use of Remote GPUs and Low-Power Processors for the Acceleration of Scientific Applications

A. Castelló*, J. Duato*, R. Mayo†, A. J. Peña‡, E. S. Quintana-Ortí†, V. Roca†, F. Silla*
*Universitat Politècnica de València, València, Spain.
Emails: {adcasgi,jduato,fsilla}@gap.upv.es
†Depto. de Ingeniería y Ciencia de Computadores, Universitat Jaume I, Castellón, Spain.
Emails: {mayo,quintana,vroca}@uji.es
‡Mathematics and Computer Science Division, Argonne National Laboratory
Argonne (IL), USA. Email: apenya@anl.gov

***Abstract*—Many current high-performance clusters include one or more GPUs per node in order to dramatically reduce application execution time, but the utilization of these accelerators is usually far below 100%. In this context, remote GPU virtualization can help to reduce acquisition costs as well as the overall energy consumption.**

In this paper, we investigate the potential overhead and bottlenecks of several "heterogeneous" scenarios consisting of client GPU-less nodes running CUDA applications and remote GPU-equipped server nodes providing access to NVIDIA hardware accelerators. The experimental evaluation is performed using three general-purpose multicore processors (Intel Xeon, Intel Atom and ARM Cortex A9), two graphics accelerators (NVIDIA GeForce GTX480 and NVIDIA Quadro M1000), and two relevant scientific applications (CUDASW++ and LAMMPS) arising in bioinformatics and molecular dynamics simulations.

Index Terms*—*High Performance Computing; Graphic Processing Units (GPUs); CUDA; Virtualization; Scientific Computing; Energy-Aware Computing;

I. Introduction

In the quest for the enormous benefits that Exascale applications promise [1]–[4], the Top500 ranking [5] and its greener counterpart, the Green500 list [6], show an impressive 6× improvement in the performance-power ratio of large-scale high performance computing (HPC) facilities over the last five years. Furthermore, a trend clearly visible in these two lists is the adoption of hardware accelerators to attain unprecedented levels of raw performance with reasonable energy costs, which hints that future Exaflop systems will most likely leverage some sort of specialized hardware.

Many supercomputers in the first positions of these two lists currently accommodate mainstream x86 based processors along with top-of-the-line accelerator technologies. The alternative proposed by the Mont-Blanc project [7] investigates how to aggregate a large number of low-power components (specifically ARM processors and accelerators with small numbers of cores) to build a Petascale general-purpose HPC cluster. In any of these cases though, it is unlikely that the accelerators that integrate the system are used 100% of the time. Therefore, for practical purposes, a cluster with one or more accelerators per node surely leads to a waste of energy and money, due to the underutilization of these devices. In contrast to that configuration, a cluster where only a few of the nodes are equipped with hardware accelerators is a more cost-effective approach in terms of energy usage, maintenance and acquisition costs.

In order to render such a reduced amount of accelerators accessible from any node in the cluster, we have heavily invested in the development of `rCUDA`[8]–[11]with seamless access to an NVIDIA GPU residing in a remote node. Although based on a simple remote procedure call (RPC) mechanism, as of today `rCUDA` is the only CUDA 5.0-compatible solution. Furthermore, compared with many of the other remote GPU virtualization solutions [12]–[17], `rCUDA` is not only compatible with CUDA 5.0, but it is also publicly available and it supports several interconnects, including the last FDR InfiniBand fabric.

In this paper we analyze the possibilities of leveraging `rCUDA` in a "heterogeneous" environment, with clients and servers running in nodes with very different capabilities. In particular, this paper makes the following original contributions:

- We provide an experimental analysis that identifies the potential and, to some extent, the overhead sources that affect the performance of distinct CPU-GPU configurations. In particular, our hardware setup includes a variety of scenarios, with the `rCUDA` client and server running, respectively, on nodes equipped with three different types of general-purpose multicore processors and two types of GPUs
- For the evaluation we select two complex CUDA-enabled applications: CUDASW++ [18] and LAMMPS [19].
- Finally, our experimental analysis focuses on the execution time, but also considers average power dissipation and energy consumption.

The rest of the paper is structured as follows. In Section II, we provide a brief overview of the `rCUDA` software solution.

 ISBN: 978-1-61208-332-2

In Section III, we describe the applications and setup employed for the experimental study that follows next, in Section IV. We close the paper with a few concluding remarks and a discussion of future work in Section V.

II. Overview of rCUDA

rCUDA is structured following a client-server distributed architecture, where the client middleware runs in the same cluster node as the application demanding GPU acceleration services, while the server middleware runs in the cluster node where the physical GPU resides.

In rCUDA, the client middleware offers the exact same interface as the regular NVIDIA CUDA Runtime API so that the application is not aware that it is interacting with rCUDA instead of a real GPU. To support a concurrent scenario where GPUs are shared and this sequence of events occurs concurrently with analogous interactions initiated by other applications, rCUDA manages independent GPU contexts for each application.

rCUDA accommodates several underlying client-server communication technologies [11] thanks to its modular layered architecture, which supports runtime-loadable network-specific communication libraries. rCUDA currently provides communication modules tailored for Ethernet- and InfiniBand-based networks and takes advantage of the increased performance of the last FDR InfiniBand fabric [11].

Furthermore, regardless of the specific communication technology, data transfers between rCUDA clients and servers are pipelined for performance, using preallocated buffers of pinned memory.

III. Experimental Setup

In this section, we describe the applications and hardware platforms involved in the experimental study.

A. Applications

CUDASW++ [18] is a bioinformatics software for Smith-Waterman protein database searches that exploits the massively parallel CUDA architecture of NVIDIA Tesla GPUs to perform sequence searches. In our study, we use release 2.0 of the package, with the following execution parameters `-query P010008.fasta -db uniprot_sprot.fasta -use_single 0`

LAMMPS [19] 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 tests in the next section, we used release `3Jan13` of the software, and benchmark `lj` included in the release. The specific parameter list employed in the experimentation with this application was `-sf cuda -v g 1 -v x 76 -v y 76 -v z 76 -v t 2000 < in.lj.cuda`

B. Systems

Our general-purpose platforms included the following three testbeds:

- KAYLA: A SECO mITX board consisting of an NVIDIA Tegra 3 ARM Cortex A9 quad-core CPU (1.4 GHz), 2 GB of DDR3 RAM, and an Intel 82574L Gigabit Ethernet controller.
- ATOM: A board with an Intel Atom quad-core CPU S1260 (2.0 GHz), 8 GB of DDR3 RAM, and an Intel I350 Gigabit Ethernet controller.
- XEON: A server with an Intel Xeon X3440 quad-core processor (2.4 GHz), 8 GB of DDR3 RAM, and an Intel 82574L Gigabit Ethernet controller. When this platform acted as an rCUDA server, three of these cores remained disabled via BIOS to reduce the power dissipation.

All three systems operated under Linux Ubuntu 12.04 with the compiler GNU `gcc/g++` version 4.6.3.

On the other hand, two types of accelerator-equipped systems were involved in the experiments:

- CARMA: A SECO development kit, with an NVIDIA Tegra 3 ARM Cortex A9 quad-core CPU (1.4 GHz) plus an NVIDIA Quadro 1000 M GPU (96 CUDA cores), 2 GB of DDR3 RAM for the ARM and 2 GB of DDR5 RAM for the GPU. These two components communicate via a PCIe $\times 4$ Gen 1 link and the network controller was an Intel 82574L Gigabit Ethernet. This system is operated under Linux Ubuntu 12.04 with the compiler GNU `gcc/g++` version 4.6.3.
- FERMI: An NVIDIA GeForce GTX480 "Fermi" GPU (448 cores), with 1,280 MB of DDR3/GDDR5 RAM. This graphics card was connected to either XEON, through a PCIe $\times 16$ Gen 1 link, or to KAYLA, via a slower PCIe $\times 4$ Gen 1 link.

The CUDA Tool Kit 5.0 was employed for both accelerators.

As these systems offered us a large variety of combinations to evaluate (specifically, 4 clients $\times$ 3 servers), we made a preliminary selection based on some initial tests and considerations:

- A few initial experiments determined that, to attain relevant acceleration factors for these two applications with respect to a parallel execution using a multicore GPU-less platform, the 448-core FERMI GPU had to be involved. We therefore discarded those configurations with CARMA as a server, due to its poweless 96-core GPU.
- When acting as a client, there is no difference between CARMA and KAYLA, as the two systems include the same type of general-purpose processor. From the points of view of power and energy, when the GPU in the CARMA system is not used, it contributes little to these two factors. Therefore, we only considered the former system for the client side in our scenarios.

Table I illustrates the different hardware configurations selected for the evaluation.

Finally, the node interconnect was a CISCO SLM2009 Gigabit Ethernet switch. We note here that only the XEON-

 ISBN: 978-1-61208-332-2

TABLE I: DIFFERENT SCENARIOS INVOLVED IN THE EXPERIMENTAL EVALUATION.

Scenario	Client	Server	Configuration
A	CARMA	KAYLA+FERMI	Low-power ARM-based client; low-power server
B	ATOM	KAYLA+FERMI	Low-power Atom-based client; low-power server
C	XEON	KAYLA+FERMI	Power-hungry client; low-power server
D	CARMA	XEON+FERMI	Low-power ARM-based client; power-hungry server
E	ATOM	XEON+FERMI	Low-power Atom-based client; power-hungry server
F	XEON	XEON+FERMI	Power-hungry client; power-hungry server

based system can be connected to a faster InfiniBand switch. Thus, for the comparison, we restrict the study to use the Gigabit Ethernet network.

C. Power and time measurement

All power data was collected using a WATTSUP?PRO wattmeter, connected to the line from the electric socket to the power supply unit (PSU), which reports instantaneous power with an accuracy of $\pm 1.5\%$ at a rate of 1 sample/s. The measures were recorded in a separate server so that the sampling process did not interfere with the accuracy of the results. In the tests, we initially ran the application under evaluation for an initial warm-up period (about 60 s); then, the execution is repeated 5 times or until enough power samples were available, the slowest and fastest repetitions are discarded, and the result was averaged and multiplied by the corresponding run time in order to obtain the energy consumption.

IV. EXPERIMENTAL EVALUATION

In this section, we present the results obtained from the execution of the two applications chosen in this study. In order to serve as a reference, we first evaluate the performance of the codes with a local GPU (the traditional scenario), and next compare these data with the performance achieved when accessing remote GPUs.

A. Acceleration via a local GPU

We open this section with an initial evaluation of the performance of the two applications, CUDASW++ and LAMMPS, when accelerated on a system equipped with a local GPU. Therefore, the only platforms that can be included in this evaluation are CARMA, KAYLA+FERMI, and XEON+FERMI. We summarize the main results from this evaluation in Table II.

CUDASW++: From the point of view of run time, the clear winner for this application is XEON+FERMI, with an execution time of 4.81 s, which is 4.79× and 7.35× faster than those observed for KAYLA+FERMI and CARMA, respectively. On the other hand, from the perspective of power (important, e.g., for a power capped environment), CARMA exhibits a much lower average power draw, roughly by factor around 5× with respect to the other two alternatives. Nevertheless, when execution time and average power are combined into a single figure-of-merit such as energy, the high-performance XEON+FERMI is again clearly superior to the other two counterparts, by factors of 1.36× and 4.28×.

LAMMPS: The behavior of the CUDA-accelerated version of this application is quite different. Now the execution times for KAYLA+FERMI and XEON+FERMI are much closer (217.94 s and 157.91 s, respectively), and they both outperform CARMA by a wide margin (1,208.29 s). This is indicative that this application makes a much more intensive use of the GPU than CUDASW++. However, for this particular application, the average power is also significantly increased for the former two platforms so that, interestingly, the total energy usage of the three systems is very similar.

To close this initial analysis, a direct comparison between the performances of KAYLA+FERMI and XEON+FERMI for the two applications in Table II illustrates the cost of the reduced PCIe bandwidth for the former platform, as both systems leverage the same type of high-performance GPU (GeForce GTX480) and the bulk of the computation is offloaded to the accelerator. Additionally, the comparison between KAYLA+FERMI and CARMA shows the negative impact caused by the reduced number of cores in the GPU featured by the latter (96 cores vs 448 in the GTX480). That results caused partially our decision of discarding this system for the server side.

B. Acceleration via a remote GPU

In the following, we experimentally analyze the potential and overheads of the different hardware scenarios listed in Table I. In all cases the application runs in a single `rCUDA` client/node that accesses the GPU connected to a single remote `rCUDA` server/node using our software middleware. No changes were made to the original GPU-accelerated CUDASW++ and LAMMPS codes other than those already necessary to install and run these packages in the platforms with the local GPU. The only change was during the compilation phase, where the applications were linked to the current release of `rCUDA`, which replaces the usual CUDA library.

Note that possible sources of overhead (bottlenecks) are the `rCUDA` middleware; the interface to the Gigabit Ethernet interconnect in the client and/or the server; (the bandwidth of) the Gigabit Ethernet; and (the bandwidth of) the PCIe interface in the server. However, the cost of this last factor was already exposed in the previous study for the local GPU case.

 ISBN: 978-1-61208-332-2

TABLE II: PERFORMANCE OF THE SELECTED APPLICATIONS WHEN RUNNING ON A SINGLE PLATFORM ACCESSING THE LOCAL GPU.

System	CUDASW++			LAMMPS		
	Time (s)	Avg. power (W)	Energy (J)	Time (s)	Avg. power (W)	Energy (J)
CARMA	35.40	24.88	880.92	1,208.29	34.14	43,663.34
KAYLA+FERMI	23.07	120.30	2,775.48	217.94	203.31	44,309.17
XEON+FERMI	4.81	134.56	647.27	157.91	268.43	42,388.21

TABLE III: PERFORMANCE, AVERAGE POWER AND CONSUMPTION OF THE TWO SELECTED APPLICATIONS WHEN RUNNING ON AN rCUDA CLIENT ACCESSING THE REMOTE GPU IN THE rCUDA SERVER.

Application	Scenario	System		Time (s)	Avg. power (W)			Energy (J)		
		Client	Server		Client	Server	Total	Client	Server	Total
CUDASW++	A	CARMA	KAYLA+FERMI	73.87	11.11	123.82	134.93	820.65	9,146.69	9,967.35
	B	ATOM	KAYLA+FERMI	61.85	44.23	120.62	164.85	2,735.98	7,460.27	10,196.26
	C	XEON	KAYLA+FERMI	61.54	58.68	117.55	176.23	3,610.90	7,234.02	10,844.92
	D	CARMA	XEON+FERMI	13.72	11.10	138.45	149.56	152.26	1,898.96	2,051.22
	E	ATOM	XEON+FERMI	11.38	42.65	143.09	185.74	485.11	1,627.65	2,112.77
	F	XEON	XEON+FERMI	6.33	58.68	138.51	197.18	371.26	876.38	1,247.65
LAMMPS	A	CARMA	KAYLA+FERMI	2,124.78	10.97	134.28	145.25	23,305.79	285,314.39	308,620.19
	D	CARMA	XEON+FERMI	482.59	12.45	191.86	204.31	6,006.23	92,589.15	98,595.38
	E	ATOM	XEON+FERMI	268.92	44.29	226.48	270.77	11,910.81	60,903.28	72,814.09
	F	XEON	XEON+FERMI	236.91	68.38	251.83	320.21	16,199.09	59,661.11	75,860.21

CUDASW++: The first aspect that is manifest in Table III and the left-hand side plots of Figures 1 and 2 is the much higher execution time of the scenarios that use the KAYLA-based accelerator as a server, even though the GPU is the same as in their XEON-based counterparts (a "Fermi" GPU). Clearly, the source of this much inferior performance must be the interface to the Gigabit interconnect for KAYLA, as the results in Table II already revealed that the slow PCIe of KAYLA, while constraining the performance to a certain extent, does not justify the large gap between the configurations that leverage this type of equipment and those based on the XEON server. Actually, in the previous section, we saw a 4.79× speed up between the KAYLA and XEON platforms, whereas in Table III speed up between both platforms, when using XEON-based client increases up to 9.72×. To confirm that the interface to the Gigabit interconnect is the cause of the higher execution time, we carried out a separate test using `iperf` [20]. The results from these independent experiments, collected in Table IV, show that the network interface for KAYLA delivers much less than the 1 Gbps bandwidth that could be expected from a Gigabit Ethernet, and confirm this as the origin of the bottleneck. On the other hand, from the point of view of average power, the configurations with a KAYLA-based server do not show significant advantages over the XEON-based ones (considering the server only, approximately between 14 and 22 W, or 10 to 15%), which in combination with the time differences explain why the configurations that involve the latter type of server lead to a much more energy-efficient solution.

Let us focus now on the behavior of CUDASW++ when running on the three configurations with the XEON-based server, which should help to identify the source of overheads in the client. The much shorter execution time of the configuration when the XEON acts as a client (6.33 s) compared with the CARMA-client and ATOM-client (13.72 s and 11.38 s, respectively), and the smaller differences in the combined average power from the two systems (149.56 W for CARMA, 185.74 W for ATOM, and 197.18 W for XEON), render the superiority of the XEON-client solution, which is almost twice more efficient in terms of energy than the two alternative configurations. In this case, as all other factors remain the same, the source of the overhead for the CARMA-based client must be the low bandwidth of the interface to the Gigabit Ethernet in the client (see Table IV). On the other hand, for the ATOM-based client, we place responsibility for the low performance in the client CPU itself.

LAMMPS: Although the results for the previous case already identified the serious bottleneck that the interface to the Gigabit Ethernet poses for the KAYLA+FERMI server, for illustrative purposes only, we still evaluate one such type of configuration. Again, this scenario exhibits a much worse performance which, combined with the limited improvement of the average power, render a much higher energy costs for the KAYLA-based server; see Table III and the right-hand side

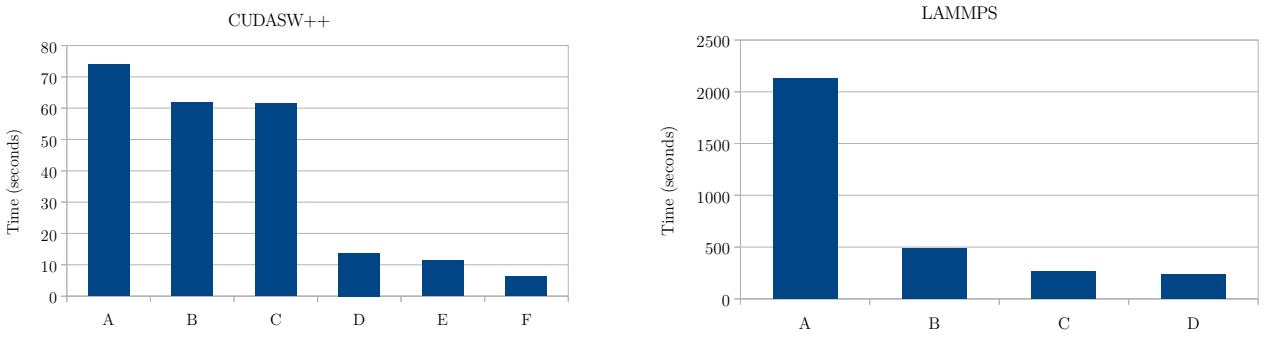

Fig. 1: Performance of CUDASW++ (rigth) and LAMMPS (left) when running on an rCUDA client accessing the remote GPU in the rCUDA server. See Table I for a description of the scenarios.

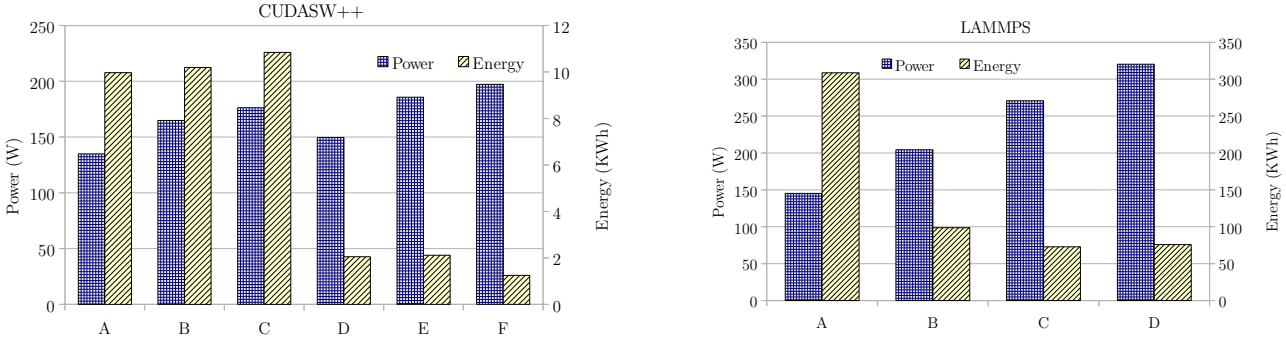

Fig. 2: Average power and energy consumption of CUDASW++ (rigth) and LAMMPS (left) when running on an rCUDA client accessing the remote GPU in the rCUDA server. See Table I for a description of the scenarios.

plots in Figures 1 and 2.

From the perspective of execution time, using the XEON equipment in both the client and server sides offers the best solution (236.91 s, compared to 482.59 s and 268.82 s in the configurations with CARMA and ATOM as clients, respectively). However, in exchange for this increase of the execution time, the ATOM-server client offers a slight but non-negligible advantage in terms of energy efficiency over the XEON-based one, of about 4.13%.

C. Overhead with respect to the local GPU

Before we perform this final analysis, it is important to realize that, from the point of view of time, the execution of an application that interacts with a remote GPU can never outperform the same application running on a platform equipped with a local GPU (provided both systems feature same type of GPU and PCIe connection). Actually, the best we can expect is that the remote access yields a low overhead, so that the execution times between the two cases (local and remote GPU) are close. From the perspective of energy, the single-client single-server configuration also works in favor of the local GPU, as we are now comparing configurations with one system (local GPU) versus two (rCUDA client and GPU-equipped rCUDA server). Nevertheless, the use of the rCUDA middleware allows to build a cluster with a reduced number of GPUs, even adapted for a specific purpose, which can be expected to decrease the total energy (as well as acquisition) costs with respect to a configuration where all cluster nodes are equipped with at least one GPU, that will likely remain idle a significant fraction of the time. In summary, the purpose of rCUDA is to reduce the energy costs for the cluster as a whole while introducing a low overhead in terms of run time.

For simplicity, let us consider only the best configurations in terms of performance, which correspond to the XEON-based client and server. Comparing this with the XEON system with a local GPU, we observe that the overhead introduced by the remote access for these two applications is rather visible, of 24.01% for CUDASW++ and 33.34% for LAMMPS. However, notice that this overhead is mainly due to the use of a slow interconnect, and the use of an FDR InfiniBand network turns them mostly negligible [11]. On the other hand, the potential benefits that a system with a moderate number of GPUs offers can be hinted by comparing the differences in the average power of the platforms when acting as clients or servers. In the same line, the power dissipated by an idle GPU can provide a more accurate estimation. In particular, the data in Table V indicate that the power dissipated by an idle FERMI GPU is between 36.4 W (compare the idle power rate of the XEON with and without the GPU) and 31.2 W (compare the figures for KAYLA –that has been included only for this

 ISBN: 978-1-61208-332-2

TABLE IV: BANDWIDTH BETWEEN TWO NODES, USING THE `iperf` TEST EXECUTED DURING 10 s.

Client	Bandwidth (Mbs)	
	KAYLA server	XEON server
CARMA	392	895
ATOM	439	940
XEON	435	943

TABLE V: POWER DISSIPATED BY THE EQUIPMENT WHILE IDLE.

Client Systems		Server Systems	
System	Power (W)	System	Power (W)
CARMA	10.4	KAYLA	41.3
ATOM	42.5	KAYLA+FERMI	72.5
XEON	54.4	XEON+FERMI	90.8

comparison– and KAYLA+FERMI), which is quite significant when contrasted to the power costs of a GPU-less node.

V. REMARKS AND FUTURE WORK

The tendencies captured by the Top500 and Green500 list portray a landscape for HPC consisting of heterogeneous facilities, with nodes of different types, specifically tailored for certain classes of applications. In this future, we envision that hardware accelerators, either in the form of data-parallel GPUs or more general-purpose architectures like the Intel Xeon Phi, will play a relevant role to build the first Exascale system. Furthermore, independently of whether the Exaflop barrier is reached using a "fat"-node approach (i.e., a large number of highly-multithreaded nodes) or a "thin"-node alternative (i.e., a huge number of low-core nodes), we believe that not all nodes/processes/threads in these systems will have direct access to an accelerator, and therefore some sort of remote access needs to be granted.

The main contribution of this paper is an experimental evaluation of the possibilities that state-of-the-art technology offers in today's HPC facilities, as well as low-power alternatives offer for the acceleration of scientific applications using remote graphics processors. In particular, we have assessed the potential of distinct hardware configurations, including three general-purpose multicore processors (Intel Xeon, Intel Atom and ARM Cortex A9) for the client side, and two types of graphics accelerators (NVIDIA GeForce GTX480 and NVIDIA Quadro M1000) for the server side. Our experiments with two key scientific applications in bioinformatics and molecular dynamics simulations, CUDASW++ and LAMMPS respectively, reveal an important bottleneck in the access to the network for the ARM-based client/server, with significant negative consequences on both the execution time and energy consumption. While the performance of the Xeon-based configuration is much better, the overhead when compared to the access to a local GPU is quite relevant, and clearly asks for the use of a faster interconnect.

In the future we plan to investigate some of the bottlenecks detected in this work, while monitoring new technology that may appear in principle solving the problem in the access to the network. We also plan to act on the applications to improve their performance (as described in the previous paragraph).

ACKNOWLEDGMENT

The researchers at UPV were supported by the the Generalitat Valenciana under Grant PROMETEOII/2013/009 of the PROMETEO program phase II. Researchers at UJI were supported by MINECO, by FEDER funds under Grant TIN2011-23283, and by the Fundacion Caixa-Castell Bancaixa (Grant P11B2013-21). This work was partially supported by the U. S. Department of Energy, Office of Science, under Contract No. DE-AC02-06CH11357. The authors are also grateful for the generous support provided by Mellanox Technologies.

REFERENCES

[1] S. Ashby *et al*, "The opportunities and challenges of Exascale computing," Summary Report of the Advanced Scientific Computing Advisory Committee (ASCAC) Subcommittee, November 2010.

[2] K. Bergman *et al*, "Exascale computing study: Technology challenges in achieving exascale systems," DARPA IPTO ExaScale Computing Study, 2008.

[3] J. Dongarra *et al*, "The international ExaScale software project roadmap," *Int. J. of High Performance Computing & Applications*, vol. 25, no. 1, pp. 3–60, 2011.

[4] M. Duranton *et al*, "The HiPEAC vision for advanced computing in horizon 2020," pp. 1–48, 2013.

[5] "The top500 list," Nov.,2013, available at http://www.top500.org.

[6] "The Green500 list," Nov., 2013, available at http://www.green500.org.

[7] "The Mont Blanc project," http://montblanc-project.eu, Feb.,2014.

[8] J. Duato, F. D. Igual, R. Mayo, A. J. Peña, E. S. Quintana-Ortí, and F. Silla, "An efficient implementation of GPU virtualization in high performance clusters," *Euro-Par 2009, Parallel Processing – Workshops*, vol. 6043, pp. 385–394, 2010.

[9] J. Duato, A. J. Peña, F. Silla, R. Mayo, and E. S. Quintana-Ortí, "Performance of CUDA virtualized remote GPUs in high performance clusters," in *Proceedings of the 2011 International Conference on Parallel Processing (ICPP 2011)*, Sep. 2011, pp. 365–374.

[10] J. Duato, A. J. Peña, F. Silla, J. C. Fernández, R. Mayo, and E. S. Quintana-Ortí, "Enabling CUDA acceleration within virtual machines using rCUDA," in *Proceedings of the 2011 International Conference on High Performance Computins (HiPC 2011)*, Dec. 2011, pp. 1–10.

[11] C. Reaño, R. Mayo, E. S. Quintana-Ortí, F. Silla, J. Duato, and A. J. Peña, "Influence of Infiniband FDR on the performance of remote GPU virtualization," in *IEEE Cluster 2013*, 2013, pp. 1–8.

[12] M. Oikawa *et al.*, "DS-CUDA: a middleware to use many GPUs in the cloud environment," in *SC*, 2012, pp. 1207–1214.

[13] G. Giunta *et al.*, "A GPGPU transparent virtualization component for high performance computing clouds," in *Euro-Par*, 2010, pp. 379–391.

[14] T.-Y. Liang *et al.*, "GridCuda: a grid-enabled CUDA programming toolkit," in *WAINA*, 2011, pp. 141–146.

[15] V. Gupta *et al.*, "GViM: GPU-accelerated virtual machines," in *HPCVirt*, 2009, pp. 17–24.

[16] Zillians, Inc. (Feb.,2014) V-GPU: GPU virtualization.

[17] 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'09)*, 2009, pp. 1–11.

[18] 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*, vol. 14, no. 1, p. 117, 2013.

[19] S. Plimpton, "Fast parallel algorithms for short-range molecular dynamics," *Journal of Computational Physics*, vol. 117, pp. 1–19, 1995.

[20] "TCP and UDP bandwidth performance measurement tool," Feb.,2014, `https://code.google.com/p/iperf/`.

 ISBN: 978-1-61208-332-2