Analyzing the Performance/Power Tradeoff of the rCUDA Middleware for Future Exascale Systems

Carlos Reaño[a], Javier Prades[a], Federico Silla[a]

[a] *Universitat Politècnica de València, Spain.*

Abstract

The computing power of supercomputers and data centers has noticeably grown during the last decades at the cost of an ever increasing energy demand. The need for energy (and power) of these facilities has finally limited the evolution of high performance computing, making that many researchers are concerned not only about performance but also about energy efficiency. However, despite the many concerns about energy consumption, the search for computing power continues. In this regard, the research on exascale systems, able to deliver 10^{18} floating point operations per second, has reached a widely consensus that these systems should operate within a maximum power budget of 20 megawatts. Many efficiency improvements are necessary for achieving this goal. One of these improvements is the usage of ARM low-power processors, as the Mont-Blanc project proposes. In this paper we analyze the combined use of ARM processors with the rCUDA remote GPU virtualization middleware as a way to improve efficiency even more. Results show that it is possible to speed up applications by almost 8x while reducing energy consumption up to 35% when rCUDA is used to access high-end GPUs. These improvements are achieved while maintaining a feasible average power consumption level for future exascale systems.

Keywords: GPU virtualization, HPC, energy, Exascale

Email addresses: carregon@gap.upv.es (Carlos Reaño), japraga@gap.upv.es (Javier Prades), fsilla@disca.upv.es (Federico Silla)

1. Introduction

For many years, designers and administrators of computing facilities, such as data centers and supercomputers, prioritized performance over energy and power consumption. An example of this continuous search for performance is the TOP500 list [1], where the most powerful supercomputers in the world are listed since June 1993. Supercomputers are included in this list according to the computing power they deliver, regardless of the energy required to drive them. Systems included in the TOP500 list are usually built upon mainstream processors such as current Intel Xeon ones, which may require up to 145 watts to work [2]. Furthermore, server nodes in these systems are usually populated with several of the aforementioned power-hungry multi-core CPUs, what increases even more the demand for power. The overall result is a costly electricity bill for the computing facility that represents a large portion of the total cost of ownership (TCO) [3].

Given that the cost of the energy consumed by a computing facility during its lifetime can easily be higher than the cost of acquisition of the hardware itself, more than one decade ago, total cost of ownership related to power consumption became an important concern. In this regard, it is worth mentioning that the cost of the infrastructure dedicated to supply power to data centers is an important fraction of their TCO [4]. Other concerns related to these large computing facilities include, for instance, the variations in the power demand of these installations. In this regard, these peta-scale facilities present peak requirements of over 30 MW with fluctuations that can range up to a few megawatts over short periods of time [5]. One more example of the issues associated with these large facilities is that they have a limited cooling capability when trying to keep a given temperature for the server room by dissipating the heat generated by those servers [6].

Limiting power consumption in these large installations became important not only from environmental and engineering points of view but also because several governments, like the U.S. or the U.K. ones, created new taxes targeted

to facilities that consume too much electricity [7][8]. Related to this, a study [9] showed that large data centers available in Internet consumed about 0.5% of the overall electricity in the world during 2005. When electricity needed for cooling and power distribution was included, that number increased up to 1%. In a similar way, the U.S. Environmental Protection Agency showed that during 2006 the power consumption of U.S. data centers accounted for 1.5% of the total electricity consumed in the country [10]. A few years later, it was pointed out that the amount of electricity used by data centers worldwide during 2010 increased by 56% with respect to the amount required in 2005 [11]. A more recent study, conducted during year 2016, showed that U.S. data centers accounted for about 2% of the total energy consumed in the U.S. that year [12]. This study also revealed that this power consumption could have been much higher if efficiency improvements would not have been applied. Efficiency improvements played a very important role in flattening the ever raising curve for power demand, being the improvements made to server nodes and data center infrastructure the ones that most contributed to energy savings, accounting for more than 95% of the total energy saved.

Efficiency improvements turned out to be the key to address energy consumption in supercomputers and large data centers, which traditionally were more and more powerful at the cost of an unaffordable power consumption trend. In this way, the greedy search for increasing computing power was finally modulated by the power wall [13], which become the primary concern for researchers. Actually, many scientists claimed that the evolution of high performance computing has reached the era when it is limited by power [14]. As a consequence of this awareness, the GREEN500 list [15] was created in November 2007 (fourteen years after the creation of the TOP500 list) with the aim of publishing information about the supercomputers presenting the highest energy efficiency, measured as GFLOPS/watt. In this way, we can see a clear shift from FLOPS to FLOPS per watt, in an attempt to rationalize the energy-hungry computing power developed during the previous years where power consumption was only limited by heat dissipation.

However, despite the many issues related to large computing facilities, the search for more computing power continues. Researchers, as well as many governments, are considering how to implement exascale computing systems. The goal of these studies is to build computing facilities able to deliver a peak-performance in the order of the Exaflop (10^{18} floating point operations per second) while operating with a maximum power budget of 20 megawatts.

One of the efficiency improvements that has increasingly been considered during the recent years is the use of ARM low-power processors for building high performance computing (HPC) installations. Probably, the most widely known example of this trend is the Mont-Blanc project [16], although other projects also exist, such as the Isambard one [17]. Many previous studies also analyzed the usage of ARM processors for HPC, such as [18], [19], [20], among others. Additionally, some companies such as Cray [21] or HPE [22] are creating high performance systems based on ARM processors. The common characteristic of all these projects and proposals is that the use of GPUs is, in general, not considered in these systems. Only the Mont-Blanc project has taken into account the usage of CUDA GPUs by considering the use of the NVIDIA TK1 system [23] for its initial prototype.

In this paper we analyze the use of high-end GPUs in these low-power ARM-based systems by making use of the rCUDA middleware, which allows applications being executed in a cluster node to use GPUs located in other nodes of the cluster. To that end, we leverage the Jetson TX1 NVIDIA board [24], which is equipped with a 4-core ARM processor along with a 256-core CUDA compatible GPU. We also make use of high-end Tesla K20m [25] and Tesla P100 [26] GPUs. The purpose of this study is to analyze whether it is beneficial to use a high-end remote GPU [27] instead of the small local GPU. That is, in this paper we analyze whether it is worth to introduce in computing deployments such as the Mont-Blanc prototype few high-end GPUs that would provide GPU services to applications being executed in the ARM-based nodes.

The rest of the paper is organized as follows. In Section 2 the necessary background about the Mont-Blanc project and about the remote GPU virtualization

mechanism is introduced. The rationale behind our proposal is also described in that section. In Section 3 the Jetson TX1 system, which will be used as testbed in this paper, is described and its performance is analyzed. Later, Section 4 presents the performance results for our proposal: using remote high-end GPUs from the Jetson TX1 system. In this section, we first analyze the performance of several applications when they use the remote high-end GPUs without sharing it with other applications. Next, a performance overview when the remote high-end GPUs are concurrently shared among several Jetson TX1 systems is provided. Moreover, we also analyze in that section our approach in terms of energy consumption. Afterwards, Section 5 introduces a brief discussion about how the results presented in this work, obtained in a small scale cluster, are representative of larger systems. Finally, Section 6 presents the main conclusions of this work.

2. Background, Related Work, and Rationale

2.1. The Mont-Blanc Project for Exascale Computing

The Mont-Blanc project [28] addresses the development of systems that make use of compute efficiency and energy efficiency for future exascale computing facilities. To that end, the Mont-Blanc project makes use of a new type of computer architecture built from energy efficient solutions leveraged in embedded and mobile devices.

The first phase of the Mont-Blanc project, that started in 2011, was granted with 8 million Euros by the European Commission. During this first phase of the project, several approaches were considered to build the underlying hardware infrastructure. The first option for building the prototype was composed of a large amount of compute cards comprising the SoC Samsung Exynos 5 Dual [29], which included two Cortex A15 ARM cores and the Mali T-604 GPU, which was OpenCL 1.1 capable.

Other possibility considered for building the prototype included the use of the NVIDIA Jetson TK1 system [23], composed of one Tegra K1 SoC (four

Cortex A15 ARM cores with one Cortex A7 core and a small 192-core Kepler GPU) and 2 GB of memory. One more option taken into account for building the underlying hardware infrastructure consisted of systems based on the Applied Micro APM883208 SoC [30], which included 8 ARMv8 cores and 16 GB of DDR3 memory.

Among the aforementioned options, the usage of the Jetson TK1 system as basic building block is very interesting, given that it is the only approach that allows the execution of CUDA-accelerated applications. Nevertheless, the small size of the GPU included in the Tegra K1 chip should be remarked, given that this GPU is about 20 times smaller than high-end GPUs such as the Tesla K20 device [25]. The smaller size of the GPU present in the Tegra K1 chip will cause that execution time of accelerated applications is not reduced as noticeably as it is expected with high-end GPUs. Additionally, it is possible that some applications cannot be executed in the Jetson TX1 system due to memory constraints.

2.2. Background on remote GPU Virtualization

Several solutions have been developed to provide remote GPU virtualization. In this regard, we can find DS-CUDA [31], rCUDA [32], vCUDA [33], GridCuda [34], GVirtuS [35] or GViM [36]. All these solutions follow a client-server approach similar to the one presented in Figure 1. The client part of the middleware, which takes the form of a library that replaces the CUDA one [37], is installed in the cluster node that executes the CUDA-accelerated application. This client part receives CUDA requests from the application being executed and forwards them to the server side, where the actual GPU is installed. Once the CUDA request is received at the server, it is forwarded to the GPU, where it is executed. The result of this execution is appropriately returned to the client, which delivers it to the application. This process is transparent to applications, which are not aware of using a remote GPU.

Different GPU virtualization developments provide different features, being rCUDA the most modern one and the one that provides the best perfor-

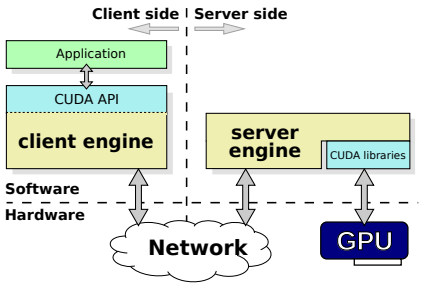
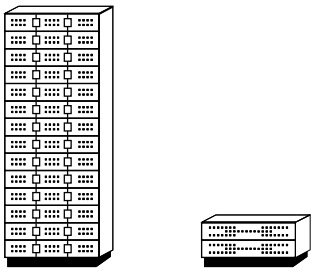


Figure 1: General architecture of remote GPU virtualization solutions.

mance [38]. The rCUDA (remote CUDA) middleware supports version 9.0 of CUDA, being binary compatible with it, which means that CUDA programs do not need to be modified for using rCUDA. Furthermore, it implements the entire CUDA API (except for graphics functions) and also provides support for the libraries included within CUDA, such as cuFFT, cuBLAS, cuSPARSE, cuDNN, cuSOLVER, etc.

2.3. Rationale behind our Proposal

The Mont-Blanc prototype is composed of a large amount of ARM-based processors. The purpose of using ARM low-power processors is to keep energy budget as low as possible given that energy consumption is one of the most important concerns in exascale systems, as well as reducing the impact of peak power requirements. However, low-power processors typically provide less performance than regular processors. This is why the use of CUDA accelerators is very interesting in this scenario. Nevertheless, the acceleration provided by the GPU included in the Jetson TK1 system cannot be compared to the acceleration provided by high-end GPUs such as the Tesla K20 device. Therefore, the best option would be to combine both approaches: using the power-efficient ARM processors along with the high-end GPUs. This can be achieved by means of rCUDA, as shown in Figure 2. This figure shows a cluster composed of 42 small ARM systems (3 nodes per row) that is complemented with two nodes that contain high-end GPUs. In this way, thanks to rCUDA, all the 42 ARM nodes can offload their GPU computations to the high-end GPUs, which are



(a) Cluster of ARM-based servers.

(b) Regular servers with high-end GPUs.

Figure 2: ARM-based cluster complemented with a few nodes comprising high-end GPUs. The high-end GPUs are shared among the many ARM nodes thanks to rCUDA.

much more powerful than the embedded GPU included in the Jetson TK1 system. Additionally, the few high-end GPUs can be concurrently and dynamically shared among all the ARM systems on demand. As a result, applications being executed in the ARM system are noticeably accelerated at the same time that efficiency is maintained.

3. The NVIDIA Jetson TX1 System

According to NVIDIA, the Jetson TX1 [24] is a power-efficient AI (artificial intelligence) supercomputer that features the NVIDIA Maxwell architecture with 256 CUDA cores and includes 4 64-bit ARM A57 CPU cores with an overall power-efficient design. Additionally, it includes the latest technology for deep learning, computer vision, GPU computing, and graphics, therefore making this system an ideal choice for embedded AI computing. The NVIDIA Jetson TX1 is a system that features the Tegra X1 processor, which comprises four ARM A57 big cores, four A53 little cores, and 2 MB of L2 cache along with a small Maxwell GPU composed of 256 CUDA cores. The system is completed with 4 GB of LPDD4 memory, which is shared by all the devices in the Tegra X1 (the CPU cores and the GPU).

Table 1: Comparison among the technical specifications of the GPU included in the Tegra X1 chip and the Tesla K20m GPU (differences highlighted in bold text).

Attribute	GPU included in Tegra TX1 chip	Tesla K20m
CUDA Capability Major/Minor version number	**5.3**	3.5
Total amount of global memory	3,983 MBytes (4,176,248,832 bytes)	**4,742 MBytes** (4,972,412,928 bytes)
Multiprocessors (MP) - CUDA Cores	2MP - 256 CUDA Cores	**13MP - 2,496 CUDA Cores**
GPU Max Clock rate	**998 MHz (1.00 GHz)**	706 MHz (0.71 GHz)
Memory Clock rate	1600 Mhz	**2,600 Mhz**
Memory Bus Width	64-bit	**320-bit**
L2 Cache Size	262,144 bytes	**1,310,720** bytes
Maximum Layered 1D Texture Size, (num) layers	1D=(16,384), 2,048 layers	1D=(16,384), 2,048 layers
Maximum Layered 2D Texture Size, (num) layers	2D=(16,384, 16,384), 2,048 layers	2D=(16,384, 16384), 2,048 layers
Total amount of constant memory	65,536 bytes	65,536 bytes
Total amount of shared memory per block	49,152 bytes	49,152 bytes
Total number of registers available per block	32,768	**65,536**
Warp size	32	32
Maximum number of threads per multiprocessor	2,048	2,048
Maximum number of threads per block	1,024	1,024
Max dimension size of a thread block (x,y,z)	(1,024, 1,024, 64)	(1,024, 1,024, 64)
Max dimension size of a grid size (x,y,z)	(2,147,483,647, 65,535, 65,535)	(2,147,483,647, 65,535, 65,535)
Maximum memory pitch	2,147,483,647 bytes	2,147,483,647 bytes
Texture alignment	512 bytes	512 bytes
Concurrent copy and kernel execution	Yes with 1 copy engine(s)	**Yes with 2 copy engine(s)**
Run time limit on kernels	**Yes**	**No**
Integrated GPU sharing Host Memory	**Yes**	**No**
Support host page-locked memory mapping	Yes	Yes
Alignment requirement for Surfaces	Yes	Yes
Device has ECC support	Disabled	**Enabled**
Device supports Unified Addressing (UVA)	Yes	Yes

Table 1 compares the technical specifications of a regular high-end GPU for accelerating HPC applications, such as the NVIDIA K20 GPU [25], versus the GPU included in the Jetson TX1 system. As can be observed, the main differences are the much smaller amount of CUDA cores in the Jetson TX1 (10 times fewer CUDA cores than in the K20 GPU), the smaller amount of memory available for the GPU in the Tegra chip (notice, additionally, that this memory is shared with the CPU cores), its higher clock rate (although with a slower memory clock rate), the much narrower bus width (8 times narrower with respect to the K20 GPU) and its much smaller L2 cache size as well as fewer registers per block. Additionally, the GPU in the Tegra chip only features a single DMA engine.

In order to better characterize the Jetson TX1 system and put it into the right perspective, Figure 3 shows a performance comparison of this system against a regular system based on Intel Xeon processors. To that end, the Jetson TX1 system, configured with a Linux Ubuntu 16.04.3 LTS (aarch64) along with CUDA 8.0, was compared to a 1027GR-TRF Supermicro server with two Intel Xeon E5-2620v2 processors (Ivy Bridge) operating at 2.1 GHz, 32 GB of DDR3 memory at 1600 MHz, and one NVIDIA K20 GPU. Linux CentOS 7.3 was used along with CUDA 8.0 in the Supermicro server. The `sysbench`[1] benchmark was used for the performance characterization.

It can be seen in Figure 3 that the Tegra X1 chip performs pretty nicely despite being a processor targeted for the embedded domain. Five different metrics are shown in the figure. In the first one, events per second, the computing power of the CPU is measured. To that end, a very small program that fits into the cache is executed. This program looks for prime numbers by leveraging mathematical operations such as divisions, modulo calculations, and square roots. Notice that in this test only one of the cores of the processors under comparison has been used. According to the performance results shown in Figure 3, the ARM core available in the Tegra X1 clearly outperforms the

[1] `https://github.com/akopytov/sysbench`

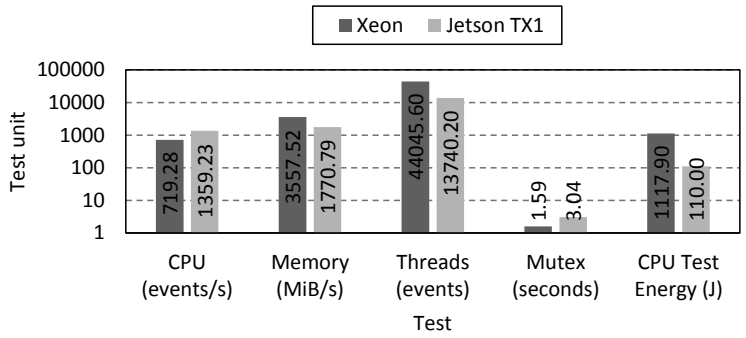


Figure 3: Performance comparison of NVIDIA Jetson TX1 and Supermicro 1027GR-TRF systems using the `sysbench` benchmark. Notice that the Y-axis is in logarithmic scale.

Xeon core, almost doubling performance.

The second test shown in Figure 3 refers to the bandwidth available to main memory. In this test, a single core has been used, as in the previous test. In this case, the Xeon-based system clearly attains a much higher bandwidth than the Jetson TX1 system, providing the former twice the performance of the latter.

The third and fourth tests share dependencies among them. On the one hand, the threads test is intended for those cases when the process scheduler in the operating system has a large amount of active threads competing for a set of mutexes. On the other hand, the mutex test is intended for those cases when all threads are concurrently running most of their execution time and only acquire a mutex lock for very short periods of time. As can be seen, both tests consider the underlying hardware as well as the actual implementation of the operating system. Figure 3 shows that in both tests the Xeon-based system outperforms the Jetson TX1 system.

The last test shows the energy consumed when running the first test. As we can see, the Xeon-based system consumes 10 times more energy than the Jetson TX1.

In summary, results presented in Figure 3 are the expected ones except for the case of the CPU test, where the higher performance of the Tegra X1 chip is surprising.

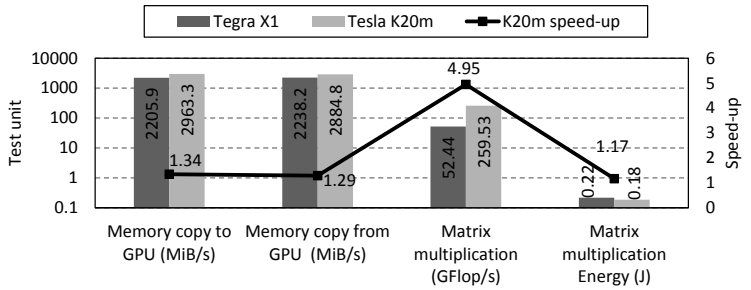


Figure 4: Performance comparison of the GPU included in the Tegra X1 chip and the Tesla K20m GPU in terms of data transfer bandwidth, computational power, and energy. Notice that primary Y-axis is in logarithmic scale.

Next step in characterizing the Jetson TX1 system and putting it into the right perspective is considering the GPU included in the Tegra X1 chip. To that end, we have compared that GPU against the high-end Tesla K20 GPU installed in the aforementioned Supermicro server. Figure 4 shows the performance comparison of both GPUs in terms of memory bandwidth and computational power.

It can be seen in Figure 4 that the Tesla K20 GPU presents higher memory bandwidth than the GPU included in the Tegra X1 chip, despite the former makes use of a PCIe Gen2 link. Notice that pageable memory has been used in this test. The high-end GPU outperforms the embedded GPU when moving data in both directions (from main memory to GPU memory and from GPU memory to main memory). This result is very interesting because, in the case of the Tegra X1 chip, both main memory and GPU memory are located in the system memory (remember that the 4GB of RAM in the Jetson TX1 are shared among the CPU and the GPU) and therefore some memory copies could be avoided. Nevertheless, given that applications are written to perform such copies (unless applications are explicitly ported to the Jetson TX1 system) measuring the performance of moving data is required. In any case, differences in bandwidth are not too large. On the contrary, when computational power is considered instead of data transfer bandwidth, the much smaller amount of CUDA cores present in the Tegra X1 chip translates into a much lower computa-

tional power. It can be seen in the figure that the Tesla K20 GPU provides 4.95 more computing power than the 256-core Maxwell GPU present in the Jetson TX1 system. It can be also observed that the energy consumed by both GPUs is similar.

Considering these results, the efficiency of the high-performance GPU is similar to that of the Jetson TX1, offering the former higher performance. On the contrary, the energy consumption of the Xeon-based system is really high in comparison to that of the Jetson one. For this reason, in our approach we try to combine the best of both systems: the ARM cores for the CPU-part of applications, and the high-performance GPU for the GPU-part of applications. To that end, we propose offloading GPU workloads from ARM systems to remote Xeon-based GPU servers.

4. Offloading Performance of GPU Workloads from ARM Systems to Remote Xeon-based GPU Servers

After the performance characterization of the Jetson TX1 system carried out in the previous section, in this section we introduce a performance analysis of our proposal: offloading the GPU-part of applications from power-efficient ARM-based systems, such as the Jetson TX1, to remote high-end GPUs located in servers built from regular Intel Xeon processors. Notice that it could also be feasible to perform the offloading to remote OpenPower servers built from Power8 or Power9 IBM processors. The idea would just be the same: using powerful remote high-end GPUs along with rCUDA in order to accelerate the execution of applications being run in power-efficient ARM-based systems. In this way, the CPU part of the application would be executed by a low-power processor requiring much less energy than traditional processors, as done in current proposals, such as the Mont-Blanc project, whereas the GPU part of the application would be executed in a powerful high-end GPU thus reducing overall execution time.

To analyze the performance of our proposal, in next sections we have made

use of (i) the Rodinia benchmark suite, (ii) a software library for computing discrete Fourier transforms (FFTW), (iii) routines for performing Basic Linear Algebra Subprograms (BLAS), and (iv) the MNIST Deep Learning application.

4.1. Rodinia benchmark suite

Rodinia [39] is a set of small applications and kernels that try to cover a large number of computing areas, such as medical images, bioinformatics, fluid dynamics, physical simulations, shape recognition, data mining, graph algorithms, linear algebra, video compression and decompression, sorting algorithms, among others. To this end, Rodinia comprises 21 applications and kernels carefully selected, among them: Breadth-First Search, Gaussian Elimination, Particle Filter, Needleman-Wunsch, Back Propagation, SRAD, Heart Wall, PathFinder, HotSpot, Hybrid Sort.

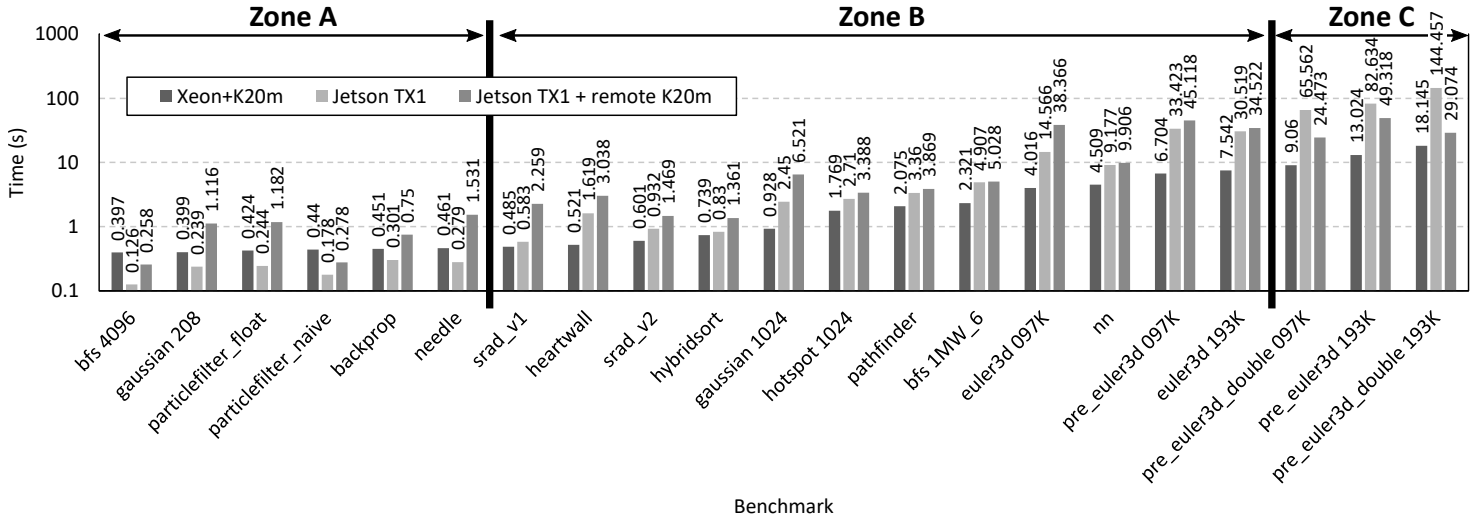


Figure 5: Comparison of Rodinia benchmarks executed in three different scenarios: (i) a regular server with one NVIDIA K20 GPU, (ii) a NVIDIA Jetson TX1, and (iii) a NVIDIA Jetson TX1 offloading GPU work to a remote regular server with one NVIDIA K20 GPU. Notice that the Y-axis is in logarithmic scale.

We have executed the different small programs and kernels within the Rodinia suite in the Jetson TX1 and compared their execution time with a new scenario where a remote high-end GPU located in a regular Xeon-based server is used instead of the small 256-core GPU included in the Jetson TX1 system. We have made use of the rCUDA remote GPU virtualization framework in order to provide applications being executed in the Jetson TX1 system with access to the remote GPU. rCUDAv16.11 was used. Figure 5 shows the execution time for each of the programs within the Rodinia benchmark suite[2]. Bars labeled as "*Jetson TX1*" refer to executions carried out using the small 256-core GPU existing in the Tegra X1 chip. Bars labeled as "*Jetson TX1 + remote K20m*" refer to executions carried out in the Jetson TX1 system while using, thanks to rCUDA, a remote K20 GPU installed in the Supermicro server described in the previous section. Finally, bars labeled as "*Xeon+K20m*" refer to the execution of the programs within the Rodinia suite in the Supermicro server using the Xeon processor and using also the Tesla K20m GPU in that system (without requiring to use rCUDA, given that the GPU is local to the application). This latter case study has been included just for comparison purposes, given that it is the traditional way to execute CUDA-accelerated applications and it has been widely analyzed in the literature.

Figure 5 shows that execution time of benchmarks when making use of our proposal (launching the application in the ARM-based system and using a remote high-end GPU instead of the local one) is sometimes better than using the local small GPU and it is sometimes worse than the local ARM-based scenario. The conclusions that can be drawn from Figure 5 (taking execution time of "*Xeon+K20m*" as reference) are the following:

- When the execution time of the benchmarks is short, then they run faster in the local scenario "*Jetson TX1*". This can be seen for benchmarks with

[2]Readers interested in the overhead of the remote CUDA calls as compared to the local CUDA calls can refer to [32],[40] or [41] for additional information.

execution times (in the Jetson TX1 system) shorter than 0.5 seconds, as pointed out in Figure 5 in the "*Zone A*" left area of the plot.

- When the execution time, in the Jetson TX1 system, is medium to large (from 0.5 to about 30 seconds), benchmarks run faster in the regular server with a high-end GPU (scenario "*Xeon+K20m*"). Furthermore, executions in the local ARM-based scenario are still faster than using a remote GPU from the Jetson TX1 system. The benchmarks that fall into this category can be found in the "*Zone B*" central area of the plot.

- For large execution time benchmarks (more than 60 seconds in the baseline "*Jetson TX1*" scenario), it is worth to use a remote GPU. That is, it is better to use the remote K20 GPU installed in a regular Xeon server (scenario "*Jetson TX1 + remote K20m*") instead of using the local GPU available in the Jetson TX1 system. This can be seen in the "*Zone C*" right part of the plot.

Next, Figure 6 presents a comparison of Rodinia benchmark "`gaussian`" varying the problem size, and executed in the same three different scenarios as in previous section. Similar conclusions to the ones drawn from Figure 5 can be extracted from Figure 6 (taking execution time of "*Xeon+K20m*" as reference):

- Short execution time tests: tests run faster in "*Jetson TX1*"

- Medium execution time tests: tests run faster in "*Xeon+K20m*"

- Large execution time tests: it is worth to use a remote GPU ("*Jetson TX1 + remote K20m*") in comparison to running the tests in "*Jetson TX1*"

The previous results have shown that it is efficient to offload work from ARM-based systems to remote high-end GPUs located in regular Intel Xeon servers. However, notice that the use of ARM-based systems is motivated by energy consumption concerns. Therefore, it is not worth to include as many regular Xeon servers as ARM-based systems exist in the cluster. On the contrary, the really interesting approach is to offload work from multiple ARM-based systems

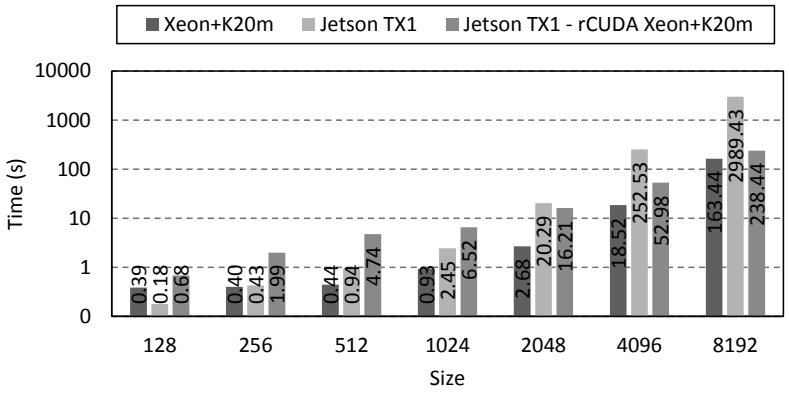


Figure 6: Comparison of Rodinia benchmark "`gaussian`" varying the problem size, and executed in three different scenarios: (i) a regular server with one NVIDIA K20 GPU, (ii) a NVIDIA Jetson TX1, and (iii) a NVIDIA Jetson TX1 offloading GPU work to a remote regular server with one NVIDIA K20 GPU. Notice that the Y-axis is in logarithmic scale.

to the same remote GPU in a shared way. In this manner, just with a few regular servers equipped with GPUs it would be enough to accelerate the applications being executed in the ARM-based nodes, as Figure 2 suggested.

In order to check the feasibility of sharing a remote high-end GPU located in a regular Intel Xeon server among several ARM-based systems, we use the Rodinia benchmark "`gaussian`" with problem size 8,192, Figure 7 shows a comparison of executing this Rodinia benchmark, with problem size 8,192, in the same three different scenarios as in previous figures. In the case of the scenario with the NVIDIA Jetson TX1 offloading GPU work to a remote regular server with one NVIDIA K20 GPU, the number of concurrent clients sharing the same remote server varies from 1 up to 6 (maximum number of instances of benchmark "`gaussian`" that can be concurrently allocated in the remote GPU memory).

It can be seen in Figure 7 that executing the "`gaussian`" Rodinia benchmark in the Jetson TX1 system (using its 256-core GPU) requires almost 50 minutes. Additionally, executing this benchmark in a regular Intel Xeon-based server using a local high-end GPU only requires less than 3 minutes. On the contrary, when the high-end GPU located in the Xeon-based server is used from the Jetson

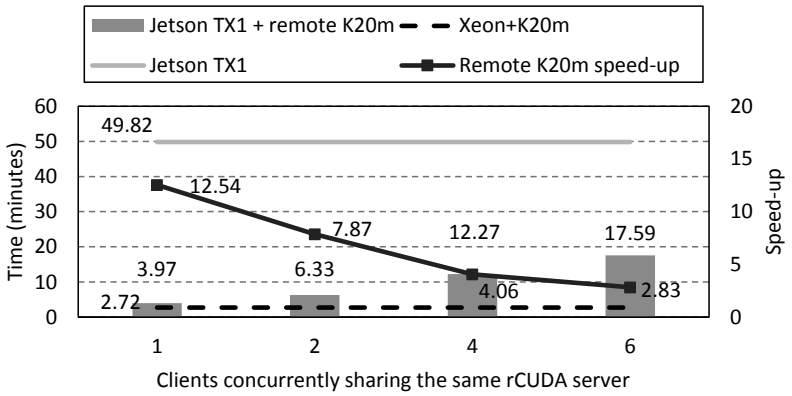


Figure 7: Comparison of Rodinia benchmark "`gaussian`" with problem size 8,192 executed in three different scenarios: (i) a regular server with one NVIDIA K20 GPU, (ii) a NVIDIA Jetson TX1, and (iii) a NVIDIA Jetson TX1 offloading GPU work to a remote regular server with one NVIDIA K20 GPU (with up to 6 concurrent clients sharing the remote GPU).

TX1 system, the benchmark requires almost 4 minutes to be executed. This translates into a speed up of more than 12x. When the high-end GPU located in the Xeon-based server is concurrently shared by several Jetson TX1 systems, speed up is still obtained. In this regard, it can be seen that as more ARM-based systems share the same high-end GPU, speed up is reduced. However, even in the worst case, when 6 Jetson TX1 systems are sharing the same K20 GPU, a speed up of almost 3x is achieved.

In addition to analyze our proposal for offloading the GPU-part of applications from ARM systems to remote Xeon-based GPU servers in terms of performance, it is also possible to carry out such an analysis in terms of power and energy consumption. Next we use the Rodinia benchmark "`gaussian`" with problem size 8,192 to analyze the power and energy consumption when sharing a remote high-end GPU located in a regular Intel Xeon server among several ARM-based systems. Figure 8 shows the energy consumption in the same three different scenarios considered in the performance analysis. Energy has been measured by polling once every second the power distribution unit (PDU) present in the cluster.

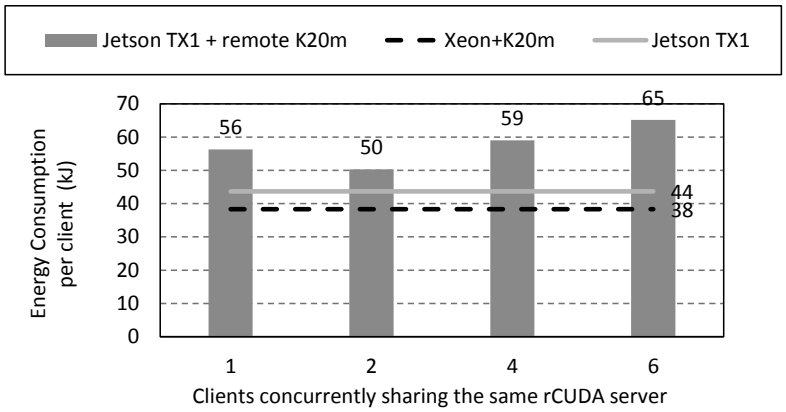
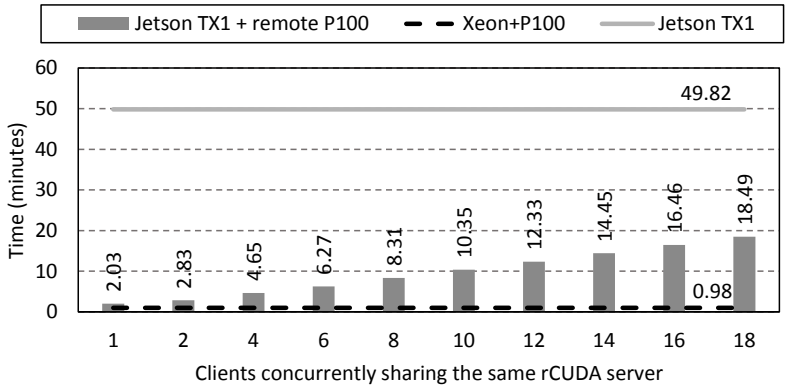


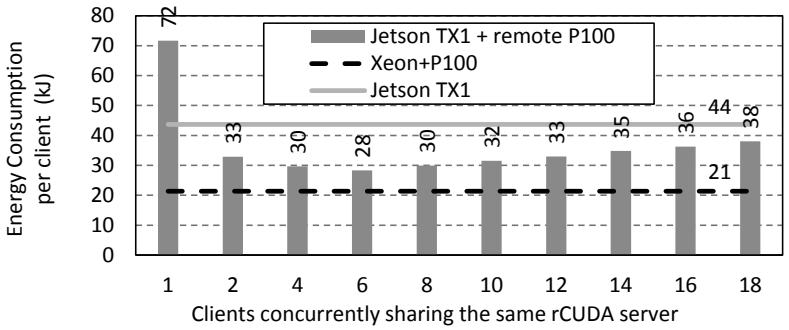
Figure 8: Energy consumption of Rodinia benchmark "`gaussian`" with problem size 8,192 executed in three different scenarios: (i) a regular server with one NVIDIA K20 GPU, (ii) a NVIDIA Jetson TX1, and (iii) a NVIDIA Jetson TX1 offloading GPU work to a remote regular server with one NVIDIA K20 GPU (with up to 6 concurrent clients sharing the remote GPU).

We can see that the scenario with the regular Xeon server provides the best results, followed by the Jetson TX1 one and finally our proposed one. Concerning our approach, we expected that the energy consumption would decrease proportionally to the number of concurrent clients sharing the same remote server. This happens when moving from 1 client to 2 concurrent clients. However, when the number of clients is over 2, the energy consumption starts increasing again unexpectedly. This is due to the fact that the remote GPU has not enough compute capability to run more than 2 concurrent instances of the application without penalizing too much the execution time. This increase in the execution time, translates into more energy consumption. To confirm this, we have repeated the same experiment but using a more powerful remote GPU: an NVIDIA Tesla P100 [26].

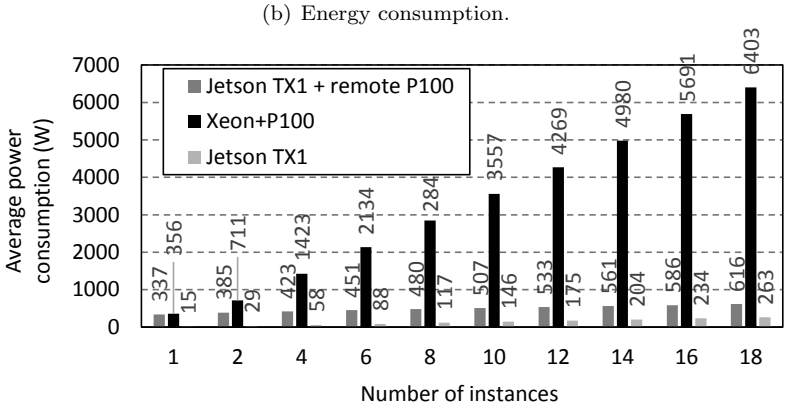
Figure 9 presents the analysis of this test when using a remote P100 GPU instead of the K20 one. In terms of performance, Figure 9(a) shows that the regular Xeon server presents the best results. It can also be seen that using a remote P100 GPU clearly outperforms the Jetson TX1 system, achieving an



(a) Execution time.



(b) Energy consumption.



(c) Average power consumption.

Figure 9: Analysis of Rodinia benchmark "`gaussian`" with problem size 8,192 executed in three different scenarios: (i) a regular server with one NVIDIA P100 GPU, (ii) a NVIDIA Jetson TX1, and (iii) a NVIDIA Jetson TX1 offloading GPU work to a remote regular server with one NVIDIA P100 GPU (with up to 18 concurrent clients sharing the remote GPU).

speed-up of up to 24.48x for 1 client. Regarding energy consumption, shown in Figure 9(b), the regular Xeon server provides again the best results, but in this case our approach provides better results than the Jetson TX1 one. The P100 GPU has more memory (16 GB) than the K20 GPU (4.7 GB) and therefore it can hold more concurrent clients (up to 18 concurrent clients in this test). As we can see, the energy consumption of our proposal decreases proportionally to the number of concurrent clients sharing the same remote server when up to 6 concurrent clients are used. At this point, the energy consumed by our approach reaches a maximum energy reduction with respect to the Jetson TX1 scenario. In this case, the energy reduction of sharing the remote GPU among the Jetson TX1 systems reaches a maximum (35%), with a performance speed-up of 7.94x.

When the number of concurrent clients is over 6, we observe a similar behavior than with the K20 GPU: the energy consumption starts increasing again. This is again due to the fact that the remote GPU compute capabilities are overloaded when running more than 6 concurrent instances of the application, and the execution time is penalized. This translates into more energy consumption. Nevertheless, our approach still consumes less energy than the Jetson TX1.

Another important factor to consider is the average power consumption along application execution, shown in Figure 9(c). In this case, the Jetson TX1 scenario presents the best results, followed by our proposal. As expected, the scenario using the regular Xeon server presents the highest average power consumption, thus making this approach not feasible for future exascale systems.

4.2. FFTW: Fastest Fourier Transform in the West

The Fastest Fourier Transform in the West (FFTW) library [42] is a software library for computing discrete Fourier transforms (DFT). Similarly, the NVIDIA CUDA Fast Fourier Transform library (cuFFT) [43] provides GPU-accelerated FFT implementations. In this section we have used FFTW for experiments not using GPU-accelerators and cuFFT for tests using GPU-accelerators.

Figure 10 shows the execution time of FFTW/cuFFT tests varying the problem size. Bars are labeled in a similar manner to the one described in the pre-

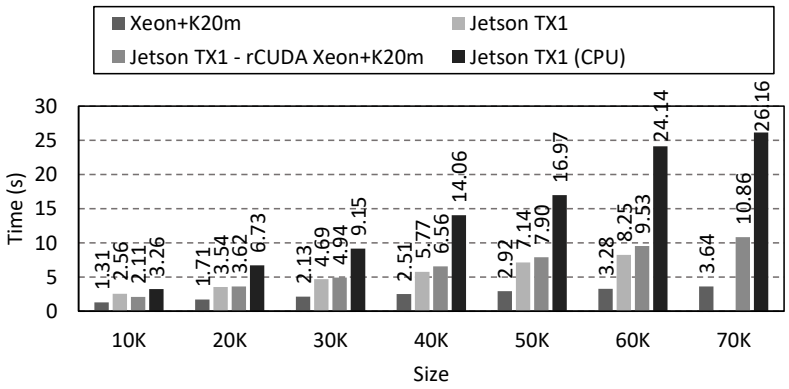


Figure 10: Comparison of the FFTW test when varying the problem size, and executed in four different scenarios: (i) a regular server with one NVIDIA K20 GPU, (ii) a NVIDIA Jetson TX1, (iii) a NVIDIA Jetson TX1 offloading GPU work to a remote regular server with one NVIDIA K20 GPU, and (iv) a NVIDIA Jetson TX1 using only the CPU (not the GPU).

vious section, but adding a new scenario, referred to as "*Jetson TX1 (CPU)*" when the executions are carried out in the Jetson TX1 CPU and not using its GPU. Thus, in all scenarios cuFFT is utilized except in the last one, where FFTW is used.

The motivation to include the latter scenario is two-fold. On the one hand, the Jetson TX1 is not capable of running some problem sizes using the GPU due to memory constraints. As previously explained in Section 3, the Jetson TX1 has 4 GB of memory, which is shared by both the CPU cores and the GPU. Thus, when memory limitations happen, tests can only be run using the CPU cores. On the other hand, as commented in Sections 1 and 2, some ARM systems do not include a GPU-accelerator. Thus, these results will allow us to compare the performance of running the tests with a low-power processor against using a remote GPU.

As we can observe in Figure 10, the best results are obtained when the tests are executed in the Xeon server with the local GPU. On the contrary, running the tests in the Jetson TX1 without using its local GPU is the worst scenario. Running the tests in the Jetson TX1 using its local GPU and using a remote GPU thanks to rCUDA provide similar results, being the former slightly better.

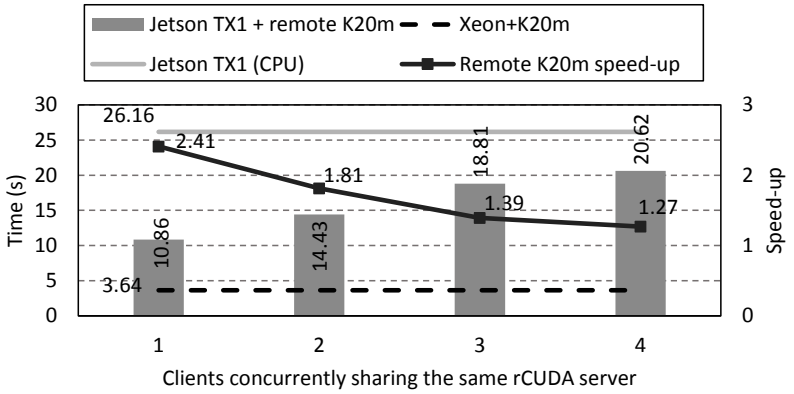


Figure 11: Comparison of FFTW test with problem size 70K executed in three different scenarios: (i) a regular server with one NVIDIA K20 GPU, (ii) a NVIDIA Jetson TX1 using only the CPU (not the GPU), (iii) a NVIDIA Jetson TX1 offloading GPU work to a remote regular server with one NVIDIA K20 GPU (with up to 4 concurrent clients sharing the remote GPU).

However, in the case of the larger test (i.e., size 70K), it is not possible to run it using the local GPU of the Jetson TX1 due to lack of memory. In that case, the benefits of using a remote GPU are significant.

In order to check the feasibility of sharing a remote high-end GPU located in a regular Intel Xeon server among several ARM-based systems, now we use the same FFTW/cuFFT tests with fixed problem size 70K to that purpose. Figure 11 shows a comparison of executing this test in the same three different scenarios as in previous figures. As commented before, notice that it was not possible to run this test using the local GPU of the Jetson TX1 due to lack of memory. For that reason, the test was run using only the CPU cores of the Jetson TX1 (line labeled as "*Jetson TX1 (CPU)*").

As we can observe in Figure 11, offloading the GPU workload to the high-end GPU located in the Xeon-based server from the Jetson TX1 system provides a speed up of over 2.4x. Similarly as in the previous section, when the remote GPU located in the Xeon server is shared among several ARM-based clients, lower speed up is obtained (below 2x). As expected, the more clients concurrently using the remote GPU, the lower speed up is obtained, although some speed

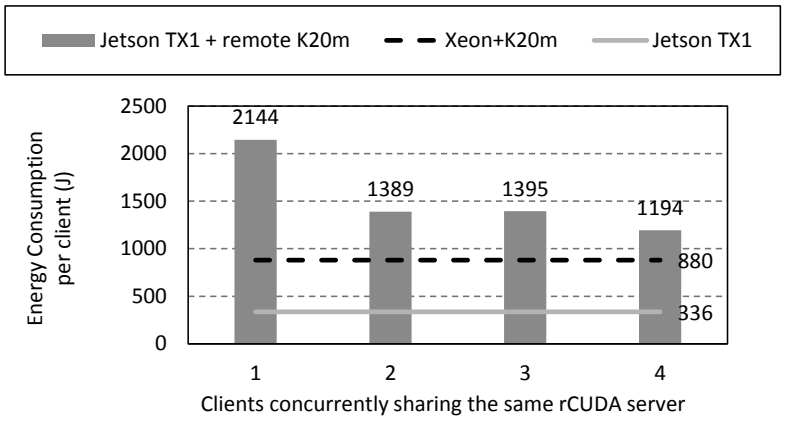
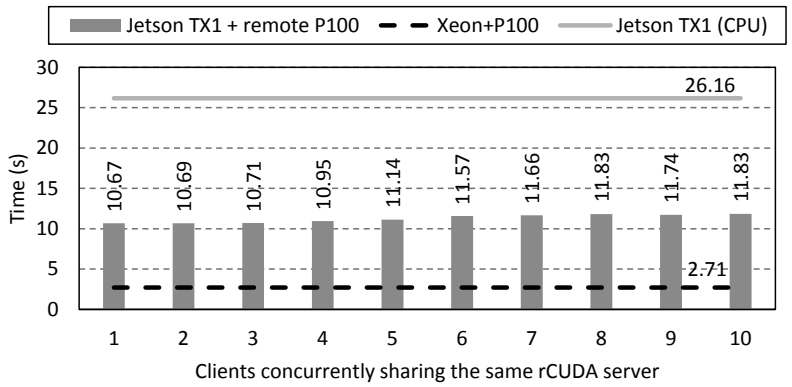


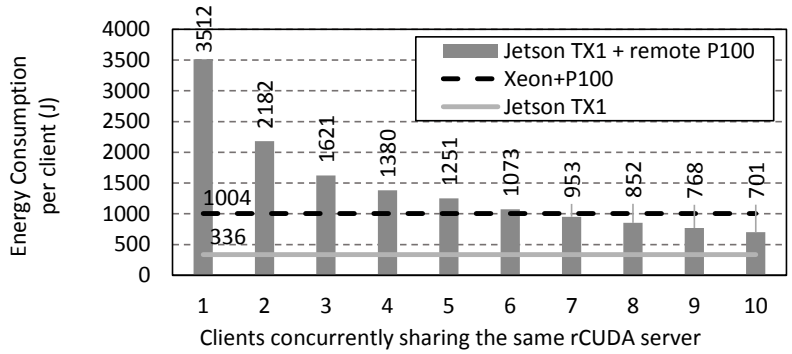
Figure 12: Energy consumption of FFTW with problem size 70K executed in three different scenarios: (i) a regular server with one NVIDIA K20 GPU, (ii) a NVIDIA Jetson TX1, and (iii) a NVIDIA Jetson TX1 offloading GPU work to a remote regular server with one NVIDIA K20 GPU (with up to 4 concurrent clients sharing the remote GPU).

up is still obtained in the case of 4 concurrent clients. In any case, the energy analysis shown next will provide even more interesting insights.

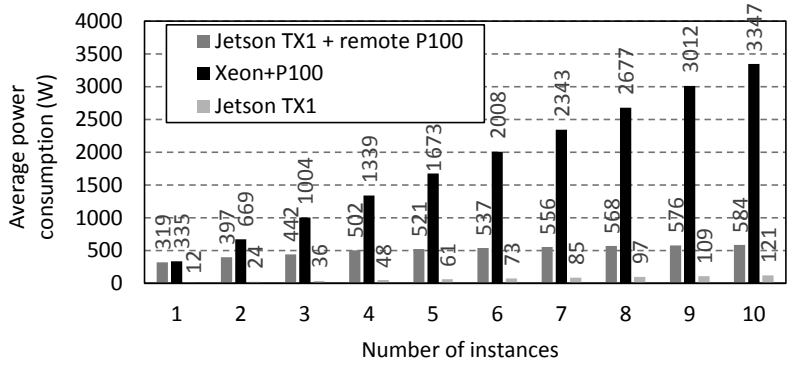
In a similar way to the experiments for the Rodinia benchmarks carried out in previous section, it is possible to analyze the behavior of the FFTW application in terms of power and energy consumption in addition to looking at performance. To that end, we use the FFTW test with problem size 70K to analyze the power and energy consumption when sharing a remote high-end GPU located in a regular Intel Xeon server among several ARM-based systems. Figure 12 shows the energy consumption in the same three different scenarios considered in the performance analysis. In this case the Jetson TX1 presents the lowest energy consumption, followed by the regular Xeon server and finally our approach. Similarly to what happened in the previous test, we can see that the energy decreases as the number of concurrent clients increases. However, the remote GPU has not enough memory to allocate enough clients to observe a higher energy reduction. To check whether energy continues reducing while adding more clients, we have repeated the same experiment but using a remote GPU with more memory: the NVIDIA Tesla P100 [26], also used in the previous



(a) Execution time.



(b) Energy consumption.



(c) Average power consumption.

Figure 13: Analysis of FFTW test with problem size 70K executed in three different scenarios: (i) a regular server with one NVIDIA P100 GPU, (ii) a NVIDIA Jetson TX1, and (iii) a NVIDIA Jetson TX1 offloading GPU work to a remote regular server with one NVIDIA P100 GPU (with up to 10 concurrent clients sharing the remote GPU).

experiment.

Figure 13 presents the analysis of this test when using a remote P100 GPU instead of the K20 one. As shown in Figure 13(a), the Xeon server obtains the best performance, followed by the Jetson TX1 with the remote P100 GPU and lastly the Jetson TX1. Regarding energy consumption, shown in Figure 13(b), the Jetson TX1 presents the lowest one. When using the Jetson TX1 with the remote GPU, energy keeps decreasing while adding more clients. In this way, with more than 6 clients the energy consumption is lower than with the regular Xeon Server. Sharing the remote GPU with the maximum number of clients, 10 in this case, still consumes more energy than the ARM system. However, the trend shows that if the GPU would have more memory to allocate more concurrent clients, it would probably achieve the best energy consumption.

The average power consumption during application execution in these experiments is shown in Figure 13(c). Similarly to what happened in the previous section, the Jetson TX1 scenario presents the best results, followed by our proposal. Again, the scenario using the regular Xeon server exhibits the highest average power consumption.

4.3. BLAS: Basic Linear Algebra Subprograms

The Basic Linear Algebra Subprograms (BLAS) [44] are routines that provide standard building blocks for performing basic vector and matrix operations. In the same manner, the NVIDIA CUDA Basic Linear Algebra Subroutines library (cuBLAS) [45] is a fast GPU-accelerated implementation of the standard BLAS routines. Similarly to the previous section, in this section we have used BLAS for experiments not using GPU accelerators and cuBLAS for those tests using GPU accelerators.

Figure 14 shows the execution time of BLAS/cuBLAS tests varying the problem size. Bars are labeled in a similar manner to the one used in the previous section. As it can be seen, for small problem sizes (i.e., 1K and 2K) running the tests in the Jetson TX1 without using its local GPU provides the best results. For larger sizes, however, the best results are obtained when the

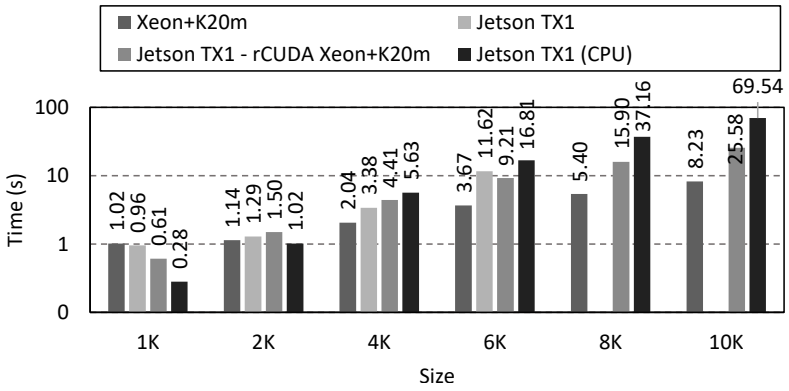


Figure 14: Comparison of the BLAS test when varying the problem size, and executed in four different scenarios: (i) a regular server with one NVIDIA K20 GPU, (ii) a NVIDIA Jetson TX1, (iii) a NVIDIA Jetson TX1 offloading GPU work to a remote regular server with one NVIDIA K20 GPU, and (iv) a NVIDIA Jetson TX1 using only the CPU (not the GPU).

tests are executed in the Xeon server with the local GPU. As in the previous section, running the tests in the Jetson TX1 using its local GPU and using a remote GPU thanks to rCUDA provides similar results, but we can observe that now the latter provides better performance for large problem sizes (i.e., 6K). Similarly to what happened in the previous section, it is not possible to run larger tests (i.e., size 8K and 10K) using the local GPU of the Jetson TX1 due to lack of memory. In that case, using a remote GPU clearly outperforms the results obtained by the executions using the CPUs in the Jetson TX1.

Next we use the BLAS/cuBLAS tests with fixed problem size 10K in order to analyze how its performance varies when an increasing amount of clients share a GPU located in a Xeon server. Figure 15 shows a comparison of executing this test in the same three different scenarios as in previous figures. As commented before, it was not possible to run this test using the local GPU of the Jetson TX1 due to lack of memory. For that reason, the test was run using only the CPU cores of the Jetson TX1 (line labeled as "*Jetson TX1 (CPU)*").

As it can be seen in Figure 15, running the test from the Jetson TX1 system and using the remote GPU located in the Xeon-based server produces a speed up of over 2.7x. The same behavior as in previous sections can be observed

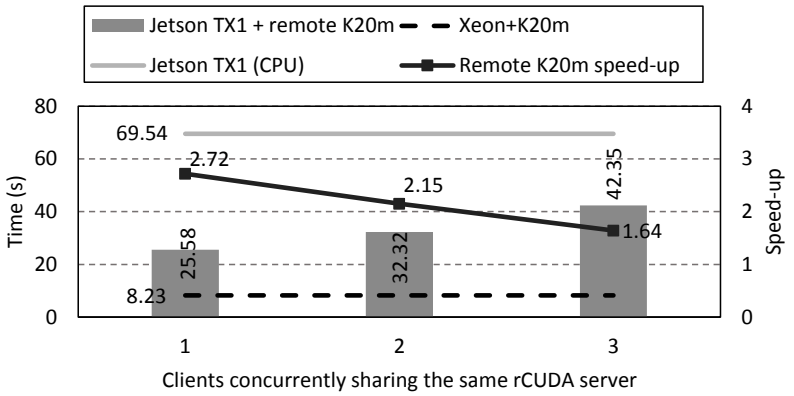


Figure 15: Comparison of BLAS test with problem size 10K executed in three different scenarios: (i) a regular server with one NVIDIA K20 GPU, (ii) a NVIDIA Jetson TX1 using only the CPU (not the GPU), (iii) a NVIDIA Jetson TX1 offloading GPU work to a remote regular server with one NVIDIA K20 GPU (with up to 3 concurrent clients sharing the remote GPU).

when the concurrent usage of the GPU is increased. Thus, sharing the same remote GPU among several clients translates into a reduction of the speed up. As mentioned before, the energy analysis shown later in the paper will provide very valuable insights.

In a similar way as we have done with the previous applications, now we use the BLAS test with problem size 10K in order to analyze the power and energy consumption when sharing a remote high-end GPU located in a regular Intel Xeon server among several ARM-based systems. Figure 16 shows the energy consumption in the same three different scenarios considered in the performance analysis. As in the previous test, the Jetson TX1 presents the lowest energy consumption, followed by the regular Xeon server and finally our approach. Again, we can see that the energy decreases as the number of concurrent clients increases. However, the remote GPU does not have enough memory to allocate an amount of clients large enough to observe a higher energy reduction. As we did in previous section, in order to check whether energy continues reducing while adding more clients, we have repeated the same experiment using a remote NVIDIA Tesla P100.

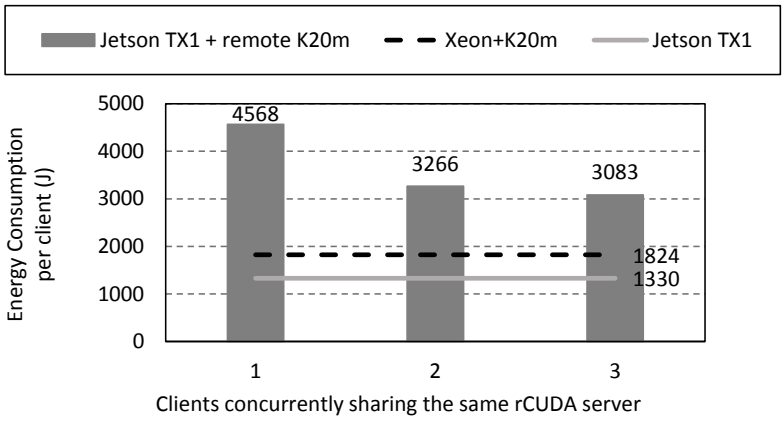
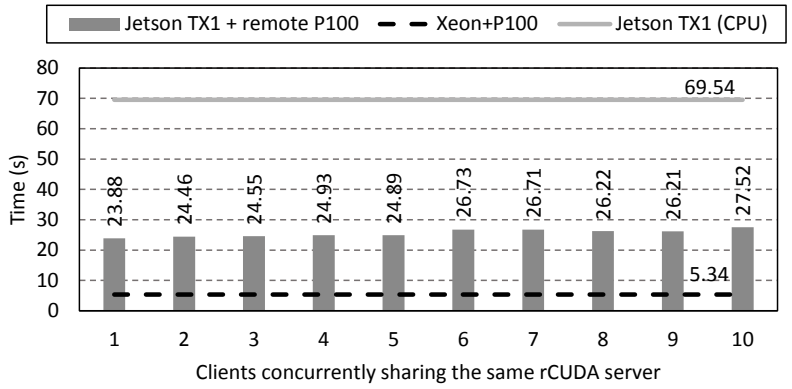


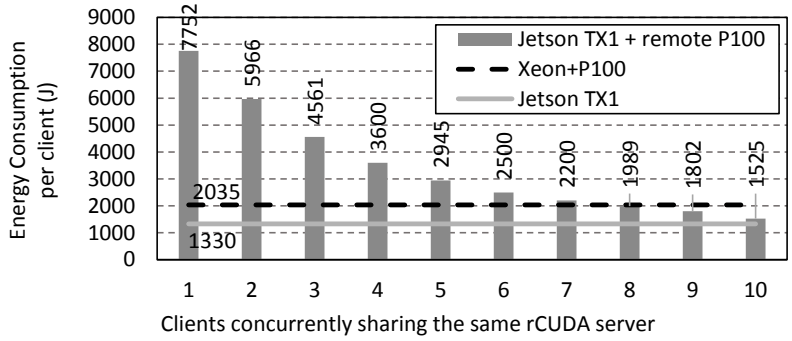
Figure 16: Energy consumption of BLAS with problem size 10K executed in three different scenarios: (i) a regular server with one NVIDIA K20 GPU, (ii) a NVIDIA Jetson TX1, and (iii) a NVIDIA Jetson TX1 offloading GPU work to a remote regular server with one NVIDIA K20 GPU (with up to 3 concurrent clients sharing the remote GPU).

Figure 17 presents the analysis of this test when using a remote P100 GPU instead of a K20 one. As shown in Figure 17(a), the Xeon server achieves the best performance, followed by the Jetson TX1 with the remote P100 GPU and lastly the Jetson TX1. Regarding energy consumption, depicted in Figure 17(b), the Jetson TX1 presents the lowest one. When using the Jetson TX1 with the remote GPU, energy keeps decreasing while adding more clients. In this way, with more than 7 clients the energy consumption is lower than with the regular Xeon server. Sharing the remote GPU with the maximum number of clients, 10 in this case, consumes slightly more energy than the ARM system. However, as in the experiments with FFTW, the trend shows that if the GPU would have more memory to allocate more concurrent clients, it would probably achieve the best energy consumption.

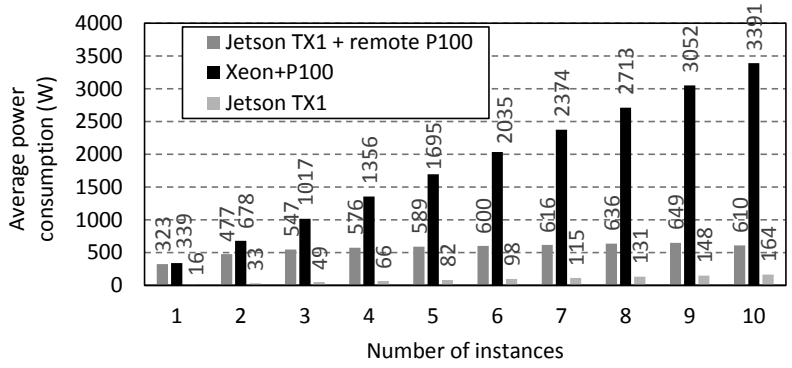
The average power consumption of these experiments is shown in Figure 17(c). In a similar way to what happened in the previous section, the Jetson TX1 scenario presents the best results, followed by our proposal. One more time, the scenario using the regular Xeon server exhibits the highest average power consumption.



(a) Execution time.



(b) Energy consumption.



(c) Average power consumption.

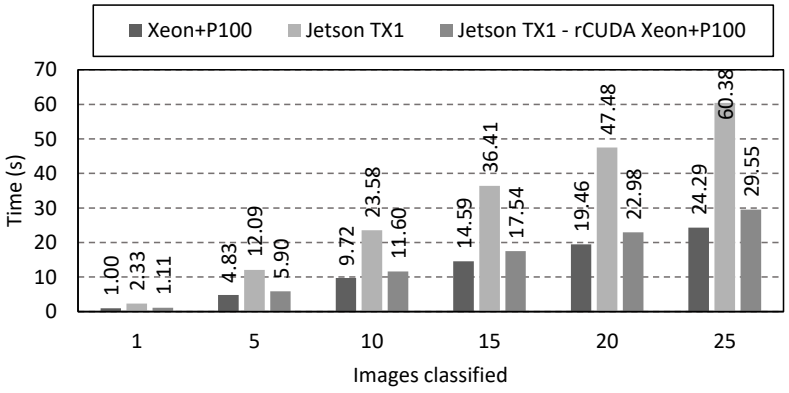
Figure 17: Analysis of BLAS test with problem size 10K executed in three different scenarios: (i) a regular server with one NVIDIA P100 GPU, (ii) a NVIDIA Jetson TX1, and (iii) a NVIDIA Jetson TX1 offloading GPU work to a remote regular server with one NVIDIA P100 GPU (with up to 10 concurrent clients sharing the remote GPU).

4.4. MNIST: Using Deep Learning Applications in Exascale Facilities

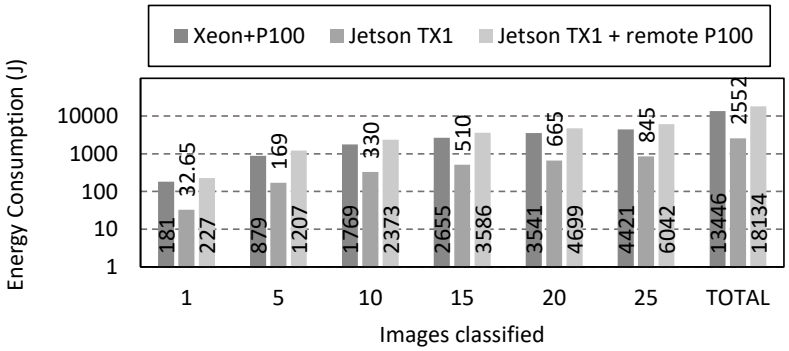
In the previous sections we have considered applications traditionally used in high performance computing clusters. However, future exascale systems can also be used for big data and AI machine/deep learning applications. In this section we use the MNIST application, like the one leveraged in [46], in order to analyze the benefits of offloading the GPU computations to high-end remote accelerators located in other nodes of the cluster. This application loads weights and biases from a trained network, and then takes one or more images of digits and recognizes them. The network was trained on the MNIST dataset using Caffe. The network consists of two convolution layers, two pooling layers, one relu and two fully connected layers. Final layer gets processed by Softmax. cublasSgemv is used to implement fully connected layers.

We first consider the performance of this application when executed in the same three different scenarios as in previous sections. Figure 18(a) shows the performance of this application when an increasing amount of images is classified. It can be seen in the figure that performance results follow the same trend as for previous applications. Figure 18(b) depicts the energy required in the executions of Figure 18(a). It can be seen that energy required to perform the classifications also follows a similar trend as for previous applications.

In a similar way as we did with previous applications, next we analyze how the performance and energy of the MNIST application varies when an increasing amount of clients running in TX1 systems share a high-end GPU located in a remote regular Xeon server. Figure 19 shows a comparison of executing this test in the same three different scenarios as in previous figures when an increasing amount of instances share the remote GPU. As it can be seen in Figure 19(a), running the test from the Jetson TX1 system and using the remote GPU located in the Xeon-based server produces a speed up of about 2x. The same behavior as in previous sections can be observed when the concurrent usage of the GPU is increased. In this way, sharing the same remote GPU among several clients translates into a reduction of the speed up (although some speed up is still achieved). However, if we look at the time per client metric (shown in

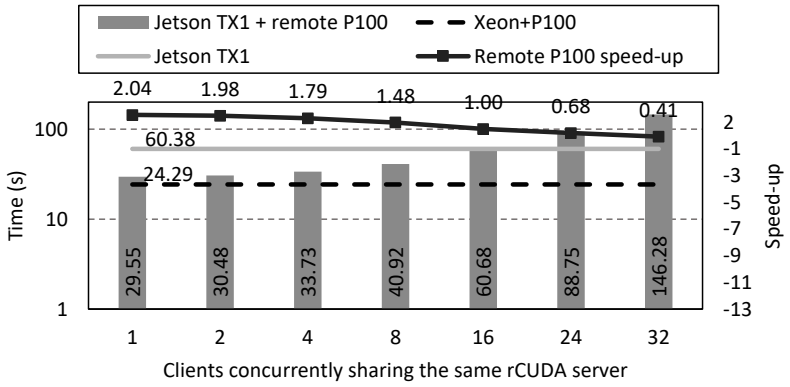


(a) Execution time.

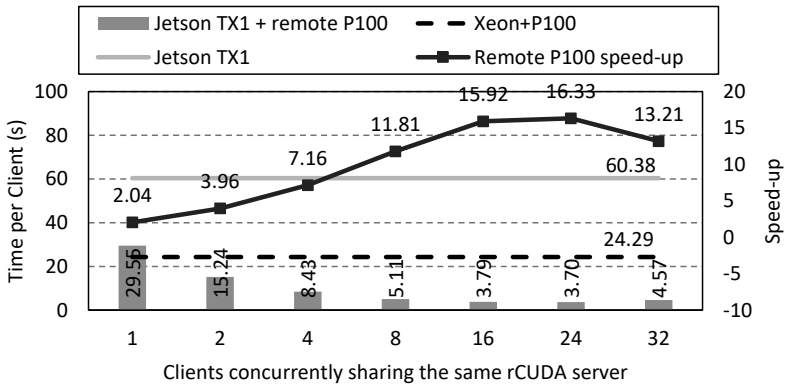


(b) Energy consumption.

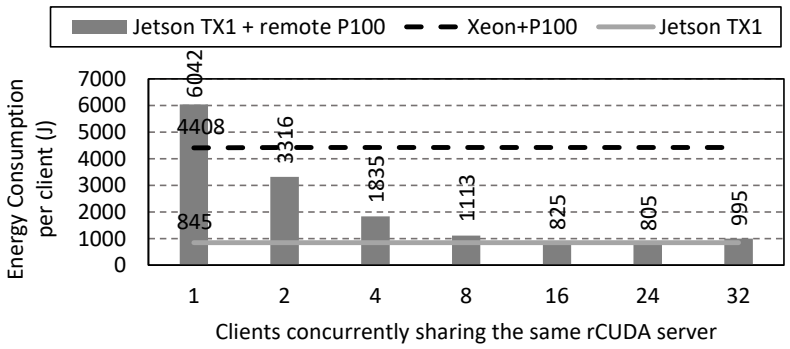
Figure 18: Comparison of the MNIST deep learning application when classifying an increasing amount of images. It is executed in three different scenarios: (i) a regular server with one NVIDIA P100 GPU, (ii) an NVIDIA Jetson TX1, and (iii) an NVIDIA Jetson TX1 offloading GPU work to a remote regular server with one NVIDIA P100 GPU.



(a) Total execution time.



(b) Execution time per client.

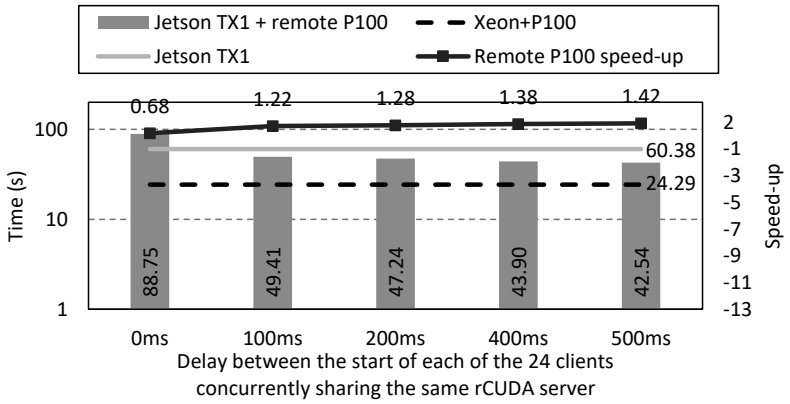


(c) Energy consumption.

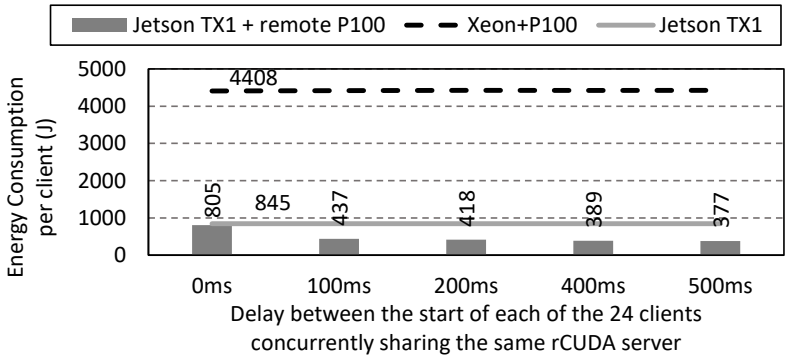
Figure 19: Comparison of the MNIST deep learning application when several instances share the remote GPU. It is executed in three different scenarios: (i) a regular server with one NVIDIA P100 GPU, (ii) an NVIDIA Jetson TX1, and (iii) an NVIDIA Jetson TX1 offloading GPU work to a remote regular server with one NVIDIA P100 GPU. All the different instances sharing the remote GPU have started execution at the same time.

Figure 19(b)), it can be seen that increasing the concurrent usage of the remote GPU provides clear benefits on the overall throughput of the system. In this way, when up to 24 concurrent instances share the remote GPU, we can observe performance benefits. In the case for 32 concurrent instances it can be seen that time per client is increased. This is due to the saturation of the GPU resources (either computing resources or communications resources). Figure 19(c) depicts the energy required by each of the clients running in the TX1 system when the remote GPU is shared. In a similar way to the performance results shown in Figure 19(b), energy consumption per client is reduced up to 24 concurrent clients. It is noteworthy that both performance and energy per client are noticeably better for a large amount of clients than those for the Xeon+P100 scenario. Similarly, these metrics are also better than those for the Jetson TX1 when 16 or 24 concurrent instances share the remote GPU.

Results shown in Figure 19 have been gathered when all the different instances that share the remote GPU start execution at the same time. This is the worst possible scenario because all the instances compete for the GPU communication resources as well as for the GPU computation cores. A more realistic scenario would consider that the different instances that share the remote GPU do not start execution at the same time. In addition to being a more realistic scenario, this case would provide better results because competition among the instances sharing the remote GPU would not be as strong as in the previous case. Figure 20 shows the performance and energy measurements when 24 concurrent instances sharing the remote GPU are launched with some delay among them. Considered delays are 0ms, 100ms, 200ms, 400ms and 500ms. It can be seen that introducing delays between two consecutive instances noticeably improves both performance and energy. Therefore, it is expected that for the applications shown in the previous sections, a real scenario where the applications sharing the remote GPU start execution at different times would also improve performance and energy metrics.



(a) Total execution time.



(b) Energy consumption.

Figure 20: Comparison of the MNIST deep learning application when 24 instances share the remote GPU. It is executed in three different scenarios: (i) a regular server with one NVIDIA P100 GPU, (ii) an NVIDIA Jetson TX1, and (iii) an NVIDIA Jetson TX1 offloading GPU work to a remote regular server with one NVIDIA P100 GPU. All the 24 instances sharing the remote GPU start execution at different times. Values in the X axis represent the delay between two consecutive application launches.

4.5. Summary

We have analyzed our proposal for offloading the GPU part of applications from ARM systems to remote Xeon-based GPU servers using three different kind of high-performance applications depending on their duration: (i) a large execution time application (Rodinia gaussian benchmark, almost 70 minutes in the Jetson TX1 system), (ii) a medium execution time application (BLAS, over 1 minute in the Jetson TX1 system), and (iii) a short execution time application (FFTW, over 20 seconds in the Jetson TX1 system). Additionally, we have used the MNIST deep learning application, which presents a behavior (attending to execution time) similar to that of the Rodinia benchmark despite having a much shorter execution time.

Table 2 summarizes the main results of this analysis, considering not only power and energy consumption, but also performance. Regarding performance, all the four applications present the expected behavior. The regular Xeon server always provides the best performance results, followed by our proposal of sharing remote high-end GPUs among ARM-based systems, and finally the ARM-based system presents the worst performance.

With respect to the average power consumption, the applications under analysis show the inverse behavior to performance. In this manner, the ARM-based system always presents the lowest average power consumption, followed by our approach. Lastly, the Xeon server shows the highest values.

The energy consumption is the metric providing more variability. When the duration of the application is large, then the Xeon server presents the lowest energy consumption, followed by our approach and the ARM system. On the contrary, when the duration of the test is medium or small, then it is the ARM system which shows the lowest one, followed by our proposal and the Xeon server.

One important factor to consider regarding the energy consumption of our approach is the GPU memory and compute power. We have observed that the remote high-end GPU shared among the ARM-based systems must have a large memory to allocate as many concurrent client applications as possible. In

Table 2: Summary of results shown in this section.

Test duration	Scenario	Performance	Energy	Average power
Large (Rodinia)	Jetson TX1 + remote P100	Average	Average	Average
	Xeon + P100	Best	Best	Worst
	Jetson TX1	Worst	Worst	Best
Medium (BLAS)	Jetson TX1 + remote P100	Average	Average	Average
	Xeon + P100	Best	Worst	Worst
	Jetson TX1	Worst	Best	Best
Small (FFTW)	Jetson TX1 + remote P100	Average	Average	Average
	Xeon + P100	Best	Worst	Worst
	Jetson TX1	Worst	Best	Best
Small (MNIST)	Jetson TX1 + remote P100	Best	Average	Average
	Xeon + P100	Average	Worst	Worst
	Jetson TX1	Worst	Best	Best

addition, it is also important that the remote GPU has a high compute power, to be able to concurrently served multiple clients without penalizing performance too much. Nevertheless, these requirements are aligned with current technology trends, given that recent NVIDIA Tesla V100 GPUs [47] feature 32 GB of RAM in addition to the largest core count up to now.

In summary, these results show that our proposal is the one providing the best trade off between performance, energy and power consumption.

5. Moving to Larger System Sizes

In the previous section we have presented a performance and energy analysis of several CUDA-accelerated applications being executed in a small scale Jetson-based cluster. The analysis shows that using remote GPUs is beneficial. However, our purpose in this paper is to target exascale systems, which are composed of hundreds or thousands of nodes. Therefore, the question now is how representative of larger systems are the results presented in previous section.

In order to answer this question, we should identify the main differences between the small Jetson-based cluster used in the experiments in this paper and a large deployment of such systems, as it could be the proposal for the MontBlanc project previously revisited in Section 2.

From our point of view, the main difference among the small cluster used in the experiments in this paper and a large deployment used in the exascale domain is the size of the network fabric used in both cases. In this regard, although both systems would use a switched-based network, the much larger size of the interconnect used in the exascale domain will not only impact the latency of the communications among the Jetson system and the remote high-end GPU but also congestion may occur.

In the absence of congestion, a larger network fabric will necessarily present larger latencies among rCUDA clients and servers. Intuitively, these larger latencies would translate into larger execution times of the CUDA calls forwarded to the remote GPU, thus negatively impacting application performance. Nevertheless, notice that the actual latency introduced by the network hardware is orders of magnitude smaller than the latency introduced by the network software. In this regard, it is well-known that the latency introduced by the hardware of the network fabric is in the nanosecond scale (for high performance interconnects) whereas the latency introduced by the software of the communication stack is in the microsecond scale. As a consequence, when moving to larger deployments, and in the absence of congestion, we can expect similar performance results to those presented in the previous section because the latency due to the software stack will absorb the increment in latency of the larger hardware size.

A different issue is related to congestion in the network. Notice that network congestion is not desired and, therefore, networks are usually designed so that enough resources are provided. A good example of this overprovisioning is fat-tree networks. On the other hand, congestion has been widely studied in the past and several proposals have been made [48][49] in order to tackle this complex problem. In any case, even if the network is designed to provide enough resources and even if an efficient congestion-management solution is included in the network, congestion may still appear. When this happens, congestion will negatively impact the performance of CUDA applications using remote GPUs, in a similar way as congestion impacts performance of general distributed applications using MPI, for instance. In order to reduce the impact of congestion

in the performance of CUDA-accelerated applications using remote GPUs, the resource manager used to assign remote GPUs to applications should be enriched to consider not only the availability of resources across the cluster but also the exact location of the application and that of the remote GPU within the cluster. In this way, by properly choosing a node for the application which is close to an rCUDA server, the impact of network congestion could be lowered. Additionally, in order to support the aforementioned proximity selection policy, rCUDA servers should be placed across the cluster in such a way that every Jetson-based node would have a set of n remote GPUs within j hops. This design policy would increase locality in the communications between rCUDA clients and rCUDA servers thus reducing the negative impact of congestion. Furthermore, the multi-tenancy approach already used in the previous section would help the resource manager to take good decisions. Notice that with the arrival of newer GPU models, such as the V100, which owns 32 GB of memory, multi-tenancy is an increasingly appealing option. The resource manager would also control the multi-tenancy degree applied to the rCUDA servers. Additionally, it is also important to note that even in the case of a wrong placement decision, it is possible to use the GPU-job migration feature included in the rCUDA middleware [50] in order to provide a better choice during application execution. To that end, the resource manager should decide a better placement by leveraging system-wide data such as network conditions, state of the rCUDA servers, etc. In any case, it is important to notice that even with a comprehensive design of the resource manager and the network fabric, the problem of congestion is too complex to be completely solved.

6. Conclusions

A new proposal for improving the efficiency of future exascale systems has been presented in this paper. The proposal considers the use of rCUDA in order to provide ARM-based systems with access to high-end remote GPUs. In this way, the power efficiency of these processors is leveraged along with the power

efficiency of high-end accelerators, which are additionally dynamically shared on demand among multiple ARM-based systems. Results shown in this paper are very important for future exascale systems based on the use of nodes leveraging ARM processors, such as the Mont-Blanc or Isambard proposals, because of two reasons.

The first reason is that these results demonstrate that for scientific applications (usually requiring large amount of time to be executed) it is worth to offload the GPU part of the application to a remote high-end GPU instead of using a small embedded one such as the GPU included in the Jetson TX1 system. Furthermore, presented results have demonstrated not only the benefits of offloading computations to remote high-end GPUs located in regular Xeon servers but also that these results are expected to be stable over time and their impact on the performance of future exascale systems will be even larger as technology improves. In this regard, notice that these results will become even more noticeable for new-generation high-end GPUs such as the V100 [47] one, which is much more powerful and, additionally, features up to 32 GB of RAM memory. In this way, future GPUs are expected to be even better aligned with the results of this paper. Results in this paper show the significance of our proposal, which allows to speed up applications being executed in ARM-based systems just by attaching to the ARM nodes a few regular Intel servers with GPUs. Moreover, regarding energy and power consumption, our study shows that it is possible to speed up applications by almost 8x while reducing energy consumption up to 35%. Furthermore, this is achieved while maintaining a feasible average power consumption level for future exascale systems.

The second reason for the importance of our results for future exascale systems is that, if future exascale systems are based on the use of low-power ARM processors but do not include a small GPU (think about the ThunderX [51] or ThunderX2 [52] processors, for instance, instead of the Jetson TX1 system), the results presented will be even more relevant. Effectively, given that GPUs reduce execution time of applications by one or two orders of magnitude, if future exascale computing systems based on the use of ARM processors do not include

an embedded GPU, then the differences in execution time of the applications when run using CPUs or when run using remote GPUs is more significant, as previously shown.

On the other hand, notice that the presented analysis shows that our proposal is the one providing the best compromise between performance, energy and power consumption. Results also reveal that for our approach being effective, GPUs must have large memory and large compute power in order to serve as many concurrent clients as possible without losing performance. Modern GPUs, such as the new NVIDIA Tesla V100 [47], with up to 32 GB of memory, over 7 TeraFLOPS in double precision, and a maximum power consumption of 300 W, will certainly help to increase the efficiency of our approach. Moreover, it is important to note that when future exascale systems become a reality, GPUs available at that time will likely present more resources than current V100 GPUs.

Finally, it must be noticed that the rCUDA middleware can be seen just as tool to provide remote concurrent access to GPUs in the cluster. In this regard, the rCUDA middleware should be augmented with a resource manager [53], which is the one that has the smartness for carrying out load balancing, thus making an even better usage of the GPU resources. Furthermore, the resource manager could make use of the GPU-job migration feature included in the rCUDA middleware [50]. This feature allows to migrate to another cluster node the GPU part of an application while it is in execution. The CPU part of the application remains in the same node. As can be seen, the rCUDA middleware is only one of the pieces of the puzzle. In this paper we have focused on the possibilities that the rCUDA middleware provides to future exascale systems, without considering the resource manager and/or migration features. If those pieces were considered, performance and energy metric would probably be improved.

Acknowledgments

This work was funded by the Generalitat Valenciana under Grant PROMETEO/2017/077. Authors are also grateful for the generous support provided by Mellanox Technologies Inc and for the equipment donated by NVIDIA Corporation.

References

References

[1] E. Strohmaier, J. Dongarra, H. Simon, M. Meuer, The TOP500 List, `https://www.top500.org/`, accessed 28 December 2017 (2017).

[2] I. Corporation, Intel Xeon Processor E5 v4 Family Product Specification, `https://ark.intel.com/products/series/91287/Intel-Xeon-Processor-E5-v4-Family`, accessed 29 December 2017 (2017).

[3] G. Group, TCO Analyst: A White Paper on Gartner Group's Next Generation Total Cost of Ownership Methodology, `http://www.netvoyager.co.uk/pdf/TCO_analyst.pdf`, accessed 29 December 2017 (1997).

[4] X. Fan, W.-D. Weber, L. A. Barroso, Power provisioning for a warehouse-sized computer, in: Proceedings of the 34th annual international symposium on Computer architecture, ISCA '07, ACM, 2007, pp. 13–23.

[5] N. Bates, G. Ghatikar, G. Abdulla, G. A. Koenig, S. Bhalachandra, M. Sheikhalishahi, T. Patki, B. Rountree, S. Poole, Electrical grid and supercomputing centers: An investigative analysis of emerging opportunities and challenges, Informatik-Spektrum 38 (2) (2015) 111–127.

[6] V. Kontorinis, L. E. Zhang, B. Aksanli, J. Sampson, H. Homayoun, E. Pettis, D. M. Tullsen, T. S. Rosing, Managing distributed ups energy for effective power capping in data centers, in: 2012 39th Annual International Symposium on Computer Architecture (ISCA), 2012, pp. 488–499.

[7] Department of Energy and Climate Change, UK, CRC Energy Efficiency Scheme, http://webarchive.nationalarchives.gov.uk/20121217154717tf_/https://www.decc.gov.uk/en/content/cms/emissions/crc_efficiency/crc_efficiency.aspx, accessed 29 December 2017 (2012).

[8] John Carey, Obama's Cap-and-Trade Plan, https://www.bloomberg.com/news/articles/2009-03-04/obamas-cap-and-trade-plan, accessed 2 January 2019 (2009).

[9] J. G. Koomey, Worldwide electricity used in data centers, Environmental Research Letters 3 (3) (2008) 034008.
URL http://stacks.iop.org/1748-9326/3/i=3/a=034008

[10] R. E. Brown, E. R. Masanet, B. Nordman, W. F. Tschudi, A. Shehabi, J. Stanley, J. G. Koomey, D. A. Sartor, P. T. Chan, Report to congress on server and data center energy efficiency: Public law 109-431 (06/2008 2008).

[11] Jonathan G. Koomey, Growth in data center electricity use 2005 to 2010, https://www.missioncriticalmagazine.com/ext/resources/MC/Home/Files/PDFs/Koomey_Data_Center.pdf, accessed 2 January 2018 (2011).

[12] Yevgeniy Sverdlik, Heres How Much Energy All U.S. Data Centers Consume, http://www.datacenterknowledge.com/archives/2016/06/27/heres-how-much-energy-all-us-data-centers-consume, accessed 29 December 2017 (2016).

[13] O. Villa, D. R. Johnson, M. Oconnor, E. Bolotin, D. Nellans, J. Luitjens, N. Sakharnykh, P. Wang, P. Micikevicius, A. Scudiero, S. W. Keckler, W. J. Dally, Scaling the power wall: A path to exascale, in: SC14: International Conference for High Performance Computing, Networking, Storage and Analysis, 2014, pp. 830–841.

[14] W. A. Ahmad, A. Bartolini, F. Beneventi, L. Benini, A. Borghesi, M. Cicala, P. Forestieri, C. Gianfreda, D. Gregori, A. Libri, F. Spiga, S. Tinti, Design of an energy aware petaflops class high performance cluster based on power architecture, in: 2017 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW), 2017, pp. 964–973.

[15] W. Feng, T. Scogland, The GREEN500 List, `https://www.top500.org/green500/`, accessed 28 December 2017 (2017).

[16] N. Rajovic, A. Rico, F. Mantovani, D. Ruiz, J. O. Vilarrubi, C. Gomez, L. Backes, D. Nieto, H. Servat, X. Martorell, J. Labarta, E. Ayguade, C. Adeniyi-Jones, S. Derradji, H. Gloaguen, P. Lanucara, N. Sanna, J. F. Mehaut, K. Pouget, B. Videau, E. Boyer, M. Allalen, A. Auweter, D. Brayford, D. Tafani, V. Weinberg, D. Brmmel, R. Halver, J. H. Meinke, R. Beivide, M. Benito, E. Vallejo, M. Valero, A. Ramirez, The mont-blanc prototype: An alternative approach for hpc systems, in: SC16: International Conference for High Performance Computing, Networking, Storage and Analysis, 2016, pp. 444–455.

[17] GW4 Alliance, Isambard, `http://gw4.ac.uk/isambard/`, accessed 2 January 2018 (2011).

[18] V. Nikl, M. Hradecky, J. Keleceni, J. Jaros, The Investigation of the ARMv7 and Intel Haswell Architectures Suitability for Performance and Energy-Aware Computing, Springer International Publishing, 2017, pp. 377–393.

[19] E. L. Padoin, D. A. G. de Oliveira, P. Velho, P. O. A. Navaux, B. Videau, A. Degomme, J.-F. Mehaut, Scalability and energy efficiency of hpc cluster with arm mpsoc, in: Workshop on Parallel and Distributed Processing, 2013, pp. 1–4.

[20] A. Selinger, K. Rupp, S. Selberherr, Evaluation of mobile arm-based socs for high performance computing, in: Proceedings of the 24th High Performance Computing Symposium, HPC '16, 2016, pp. 21:1–21:7.

[21] Cray, Cray XC50 Compute Blade for Arm Processors, https://www.cray.com/sites/default/files/Cray-XC50-ARM-Product-Brief.pdf, accessed 2 January 2018 (2017).

[22] HPCWire, HPE Launches ARM-based Apollo System for HPC, AI, https://www.hpcwire.com/2017/11/14/hpe-launches-arm-based-apollo-system-hpc-ai/, accessed 2 January 2018 (2017).

[23] NVIDIA, Jetson TK1 Embedded Development Kit, http://www.nvidia.com/object/jetson-tk1-embedded-dev-kit.html, accessed 3 January 2018 (2013).

[24] NVIDIA, Jetson TX1, https://www.nvidia.com/en-us/autonomous-machines/embedded-systems-dev-kits-modules/, accessed 15 December 2017 (2017).

[25] NVIDIA, TESLA K20 GPU ACCELERATOR Board Specification, http://www.nvidia.com/content/pdf/kepler/tesla-k20-passive-bd-06455-001-v07.pdf, accessed 3 January 2018 (2013).

[26] NVIDIA, NVIDIA Tesla P100 GPU ACCELERATOR, http://images.nvidia.com/content/tesla/pdf/nvidia-tesla-p100-PCIe-datasheet.pdf, accessed 3 January 2018 (2016).

[27] F. Silla, J. Prades, S. Iserte, C. Reaño, Remote GPU Virtualization: Is It Useful?, in: 2016 2nd IEEE International Workshop on High-Performance Interconnection Networks in the Exascale and Big-Data Era (HiPINEB), 2016, pp. 41–48.

[28] M.-B. Project, Mont-Blanc: European Approach towards Energy Efficient High Performance, http://www.montblanc-project.eu/home, accessed 3 January 2018 (2015).

[29] Samsung, Mobile Processor Exynos 5 Dual: The Industry's First Cortex-A15 Dual-core Mobile Processor, http://www.samsung.com/semiconductor/minisite/exynos/products/mobileprocessor/exynos-5-dual-5250/, accessed 3 January 2018 (2011).

[30] applied micro, APM883208-X1: X-Gene Multi-Core 64-bit Processor, https://myapm.apm.com/technical_documents/download/apm883208-product-brief1, accessed 3 January 2018 (2014).

[31] M. Oikawa, A. Kawai, K. Nomura, K. Yasuoka, K. Yoshikawa, T. Narumi, DS-CUDA: A Middleware to Use Many GPUs in the Cloud Environment, in: Proc. of the SC Companion: High Performance Computing, Networking Storage and Analysis, SCC, 2012, pp. 1207–1214.

[32] C. Reaño, F. Silla, G. Shainer, S. Schultz, Local and Remote GPUs Perform Similar with EDR 100G InfiniBand, in: Proceedings of the Industrial Track of the 16th International Middleware Conference, Middleware Industry '15, 2015.

[33] L. Shi, H. Chen, J. Sun, vCUDA: GPU accelerated high performance computing in virtual machines, in: Proc. of the IEEE Parallel and Distributed Processing Symposium, IPDPS, 2009, pp. 1–11.

[34] T. Y. Liang, Y. W. Chang, GridCuda: A Grid-Enabled CUDA Programming Toolkit, in: Proc. of the IEEE Advanced Information Networking and Applications Workshops, WAINA, 2011, pp. 141–146.

[35] G. Giunta, R. Montella, G. Agrillo, G. Coviello, A GPGPU Transparent Virtualization Component for High Performance Computing Clouds, in: Proc. of the Euro-Par Parallel Processing, Euro-Par, 2010, pp. 379–391.

[36] V. Gupta, A. Gavrilovska, K. Schwan, H. Kharche, N. Tolia, V. Talwar, P. Ranganathan, GViM: GPU-accelerated virtual machines, in: Proc. of the ACM Workshop on System-level Virtualization for High Performance Computing, HPCVirt, 2009, pp. 17–24.

[37] NVIDIA, CUDA C Programming Guide. Design Guide, http://docs.nvidia.com/cuda/pdf/CUDA_C_Programming_Guide.pdf, accessed 26 March 2017 (2017).

[38] C. Reaño, F. Silla, A Performance Comparison of CUDA Remote GPU Virtualization Frameworks, in: 2015 IEEE International Conference on Cluster Computing, 2015.

[39] S. Che, M. Boyer, J. Meng, D. Tarjan, J. W. Sheaffer, S. H. Lee, K. Skadron, Rodinia: A benchmark suite for heterogeneous computing, in: 2009 IEEE International Symposium on Workload Characterization (IISWC), 2009, pp. 44–54.

[40] C. Reaño, J. Prades, F. Silla, Exploring the use of remote gpu virtualization in low-power systems for bioinformatics applications, in: Proceedings of the 47th International Conference on Parallel Processing Companion, ICPP '18, 2018, pp. 8:1–8:8.

[41] C. Reaño, F. Silla, A comparative performance analysis of remote gpu virtualization over three generations of gpus, in: 2017 46th International Conference on Parallel Processing Workshops (ICPPW), 2017, pp. 121–128.

[42] M. Frigo, S. G. Johnson, The design and implementation of FFTW3, Proceedings of the IEEE 93 (2) (2005) 216–231, special issue on "Program Generation, Optimization, and Platform Adaptation".

[43] NVIDIA, CUDA Fast Fourier Transform library (cuFFT), https://developer.nvidia.com/cufft, accessed 23 March December 2018 (2018).

[44] University of Tennessee, Basic Linear Algebra Subprograms (BLAS), http://www.netlib.org/blas/, accessed 23 March 2018 (2018).

[45] NVIDIA, CUDA Basic Linear Algebra Subroutines (cuBLAS), https://developer.nvidia.com/cublas, accessed 23 March 2018 (2018).

[46] C. Reaño, F. Silla, A live demo on remote gpu accelerated deep learning using the rcuda middleware, in: Proceedings of the Posters and Demos Session of the 16th International Middleware Conference, Middleware Posters and Demos '15, 2015, pp. 3:1–3:2.

[47] NVIDIA, NVIDIA Tesla V100 GPU ACCELERATOR, https://images.nvidia.com/content/technologies/volta/pdf/tesla-volta-v100-datasheet-letter-fnl-web.pdf, accessed 3 January 2018 (2017).

[48] J. Escudero-Sahuquillo, P. J. Garcia, F. J. Quiles, G. Maglione-Mathey, J. Duato, Feasible enhancements to congestion control in infiniband-based networks, Journal of Parallel and Distributed Computing 112 (2018) 35 – 52.

[49] J. Escudero-Sahuquillo, E. G. Gran, P. J. Garcia, J. Flich, T. Skeie, O. Lysne, F. J. Quiles, J. Duato, Efficient and cost-effective hybrid congestion control for hpc interconnection networks, IEEE Transactions on Parallel and Distributed Systems 26 (1) (2015) 107–119.

[50] J. Prades, F. Silla, Turning gpus into floating devices over the cluster: The beauty of gpu migration, in: 2017 46th International Conference on Parallel Processing Workshops (ICPPW), 2017, pp. 129–136.

[51] Cavium, ThunderX ARM Processors: High Performance Workload Optimized Processors For Data Demanding Applications and Cloud Infrastructure, http://www.cavium.com/product-thunderx-arm-processors.html, accessed 3 January 2018 (2013).

[52] Cavium, ThunderX2 ARM Processors: High Performance ARMv8 Processors for Cloud and HPC Server Applications, http://www.cavium.com/product-thunderx2-arm-processors.html, accessed 3 January 2018 (2018).

[53] S. Iserte, J. Prades, C. Reaño, F. Silla, Increasing the performance of data centers by combining remote gpu virtualization with slurm, in: 2016 16th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid), 2016, pp. 98–101.