Graphical Abstract

Improving the Performance of Physics Applications in Atom-Based Clusters with rCUDA

Federico Silla, Javier Prades, Elvira Baydal, Carlos Reaño

Highlights

Improving the Performance of Physics Applications in Atom-Based Clusters with rCUDA

Federico Silla, Javier Prades, Elvira Baydal, Carlos Reaño

- New proposal for using remote GPU virtualization in Atom-based clusters
- Thorough performance analysis using 5 different applications and 4 GPU generations
- Results show a speed up of up to 140x
- Up to 37x improvement in energy consumption

Improving the Performance of Physics Applications in Atom-Based Clusters with rCUDA

Federico Silla[a], Javier Prades[a,*], Elvira Baydal[a], Carlos Reaño[b]

[a]*Universitat Politècnica de València, DISCA, Camino de Vera s/n, Edificio 1G, 46022 Valencia, Spain*
[b]*Queen's University Belfast, University Road, Belfast, BT7 1NN, Northern Ireland, United Kingdom*

[*]Corresponding author

Email addresses: `fsilla@disca.upv.es` (Federico Silla), `japraga@gap.upv.es` (Javier Prades), `mebaydal@disca.upv.es` (Elvira Baydal), `C.Reano@qub.ac.uk` (Carlos Reaño)

Abstract

Traditionally, High-Performance Computing (HPC) has been associated with large power requirements. The reason was that chip makers of the processors typically employed in HPC deployments have always focused on getting the highest performance from their designs, regardless of the energy their processors may consume. Actually, for many years only heat dissipation was the real barrier for achieving higher performance, at the cost of higher energy consumption. However, a new trend has recently appeared consisting on the use of low-power processors for HPC purposes. The MontBlanc and Isambard projects are good examples of this trend. These proposals, however, do not consider the use of GPUs.

In this paper we propose to use GPUs in this kind of low-power processor based HPC deployments by making use of the remote GPU virtualization mechanism. To that end, we leverage the rCUDA middleware in a hybrid cluster composed of low-power Atom-based nodes and regular Xeon-based nodes equipped with GPUs. Our experiments show that, by making use of rCUDA, the execution time of applications belonging to the physics domain is noticeably reduced, achieving a speed up of up to 140x with just one remote NVIDIA V100 GPU with respect to the execution of the same applications using 8 Atom-based nodes. Additionally, a rough energy consumption estimation reports improvements in energy demands of up to 37x.

Keywords:
CUDA, rCUDA, low-power processors, physics applications, GPU virtualization, InfiniBand

1. Introduction

Recently, a new trend has appeared in the HPC (High-Performance Computing) domain, consisting on the use of low-power processors as a replacement for the traditional x86 mainstream processors, which comprise Intel Xeon chips requiring up to 145 Watts to work [1]. Examples of this new trend include the MontBlanc project [2] being developed in the Barcelona Supercomputing Center, or the Isambard project [3] from the University of Bristol. Both projects propose the use of massive ARM-based computing facilities as a way to achieve the computing power required in current and future data centers. Many other studies also analyzed the usage of low-power processors for HPC, such as [4], [5], [6], etc.

The most recent of the aforementioned initiatives are supported by the Cavium ThunderX processor, which provides 48 64-bit ARM cores in a single chip in the first version of the processor [7] whereas 32 cores are included in the ThunderX2 version [8]. Furthermore, it is possible to create dual-socket systems by using these processors. Well renowned companies such as Cray, HPE or GigaByte are providing the nodes for these projects at the same time that they have created high performance commercial products based on these processors [9][10][11].

It is also remarkable that before these 64-bit ARM processors were commercially available, different systems were proposed based on the use of the Intel Atom processor. Using the Intel Atom processor ensures full compatibility for the application codes already developed. Currently, Supermicro has introduced several of these solutions into the market, such as the Superserver 5039MA16-H12RFT system [12], which comprises 12 Atom sockets providing a total of 192 cores in a very dense 3U format. The new ARM-based systems by HPE, Cray or GigaByte are also available in very dense formats.

The reason for the appearance of the new trend mentioned at the beginning of this section is energy consumption. Traditionally, HPC has prioritized performance over energy consumption, thus being associated with high power demands. In this regard, in order to achieve execution times as low as possible, powerful (but energy hungry) processors have traditionally been used. Unfortunately, the energy demands of supercomputers and data centers reached such a non-sustainable point that the scientific community shifted the focus to energy-efficiency. One of the consequences of this shift was the creation of the GREEN500 list [13] in November 2007, which is the energy-efficient counterpart of the TOP500 list [14], that includes, since

1993, the most powerful supercomputers in the world just attending to the computing power they deliver and regardless of the energy they require.

The energy concern is so important that several governments got involved into the problem. For instance, the U.S. and the U.K. governments created new taxes intended for facilities consuming large amounts of electricity [15][16]. These taxes were the consequence of studies such as [17], which showed that U.S. data centers required 1.5% of the total electricity consumed in the U.S in year 2005. This amount of energy increased to 2% in 2016 [18]. In a wider perspective, large data centers consumed about 0.5% of the overall electricity worldwide during year 2005 [19]. This amount increased by 56% in year 2010 [20].

In this search for energy-efficient computing we should not forget accelerators. These devices have shown to be able to noticeably reduce the execution time of applications while keeping energy consumption at reasonable levels. GPUs (Graphics Processing Units) are one example of accelerators. GPUs are widely used in current data centers and supercomputers as a way to increase their computing capacity (almost 20% of the supercomputers in the TOP500 list include GPUs). Furthermore, GPUs are very power-efficient when they are active (6 out of the 10 best supercomputers in the GREEN500 list include GPUs), although they require non-negligible amounts of energy while idle[1]. In order to avoid the downside of idle GPUs, GPU virtualization can be leveraged. By virtualizing GPUs, their utilization is increased [21], similarly to what happens with traditional CPU virtualization. Furthermore, it is also possible to enhance GPU virtualization by adding a new feature to it: remote access [22]. In this way, it is possible to virtualize GPUs and provide them to applications that are being executed in other nodes of the cluster. This technique is known as remote GPU virtualization [23], and it allows to transparently share the GPUs in a server among many applications running in different nodes of the cluster. The overall effect is that GPU uti-

[1]When referring to the idle state of GPUs, it is important to distinguish between the case when they are assigned to an application but they are not performing any computation and the state when they are not assigned to any application. In the former state, power consumption may easily reach 50 Watts while in the latter state power consumption is lower (up to 25 Watts depending on the exact model of GPU). Unfortunately, the state where a GPU is assigned to an application but it is not performing any computation is much more frequent than expected [21]. In this way, idle GPUs are, in general terms, non-power efficient devices.

lization is noticeably increased, thus reducing the waste of energy caused by the idle state of GPUs.

The several low-power processor based aforementioned projects (Mont-Blanc, Isambard) do not consider the use of GPUs. Several might be the reasons for this decision. For instance, it may be counter intuitive to include these accelerators, which require large power supplies, in these servers which are usually equipped with much smaller power supplies. One more reason might be the lack of enough PCIe lanes to install both a GPU and high performance network card. Another reason for not considering GPUs might be the compact form factor that is usually employed for these low-power systems. Regardless of the exact reason, we believe that GPUs can help these systems to noticeably increase their computing capacity while being power-efficient. To that end, we propose the use of the remote GPU virtualization within these systems in order to access high-end GPUs located in regular x86-based servers.

In this paper we analyze the use of high-end GPUs in the context of these low-power systems by making use of the rCUDA middleware [24], which allows applications being executed in a cluster node to transparently use GPUs located in other nodes of the cluster. To that end, we use low-power Atom systems by Intel. The aim of this study is to analyze how the use of high-end remote GPUs can benefit the execution of applications in these low-power systems. In particular, we will focus on applications from the physics domain. On the other hand, we make use of four different generations of NVIDIA GPUs: Tesla K20 GPUs [25], Tesla K40 GPUs [26], Tesla P100 GPUs [27] and Tesla V100 GPUs [28]. Notice that we could have used ARM-based systems instead of Atom-based ones. It is important to remark that we do not suggest in this study that the choice of Atoms is better than using ARM processors. In this paper we do not enter into that discussion. We simply make use of Atom-based systems because using current application codes in them is straightforward, thus not requiring any tuning or adaptation stage of their code.

A preliminary version of this study was presented in [29]. Contrary to the present paper, that work considered a reduced amount of applications as well as a smaller amount of GPU generations. Also, the present paper provides a noticeably more mature performance analysis. In this regard, the study in [29] used only two applications whereas this paper is based on the analysis of five different applications. This larger amount of applications makes the conclusions of this new paper much more solid. On the other hand, this paper considers more GPU generations than the study in [29]. Furthermore, this new study presents scalability trends for the considered applications based on real executions using up to 256 CPU cores distributed in several x86-based nodes. Finally, the performance analysis in this paper includes additional metrics, such as efficiency.

The rest of this paper is organized as follows. In Section 2 the necessary background about the hardware and software environment used in this study is provided. In addition, the main motivation for this study is also presented. Later, Section 3 presents the performance results for our proposal: using remote high-end GPUs from Atom systems. Section 4 provides an analysis of how the results in this paper could be applied to very large low-power processor-based deployments. Finally, Section 5 presents the main conclusions of this work.

2. Motivation and Background

Section 2.1 explains the rationale for our proposal. Next, we provide an overview of remote GPU virtualization in Section 2.2.

2.1. Motivation for our Proposal

The aim of the aforementioned proposals for using low-power processors in the HPC domain is to maintain energy consumption as low as possible, given that power requirements in general, and energy consumption in particular, is one of the most important concerns in current data centers. Nevertheless, low-power processors usually provide worse performance than traditional mainstream processors. This is the reason why using GPUs might be very interesting in this context. However, as mentioned before, there are several reasons that hinder the use of GPUs in deployments based in low-power processors. Thus, in order to combine the best of both approaches (low-power processors and high-performance GPUs), remote GPU virtualization could be used, as shown in Figure 1. This figure depicts a cluster based on Atom processors, consisting of 42 nodes, which is enhanced with some nodes that contain high-end GPUs, such as the NVIDIA Tesla P100 ones, for instance. In this way, thanks to the sharing features of the remote GPU virtualization mechanism, all the 42 Atom nodes can offload, across the network, their computations to the remote high-end GPUs. Furthermore, these high-end GPUs can be dynamically and concurrently shared among all the Atom nodes. The overall result is that applications being run in the low-power processors are

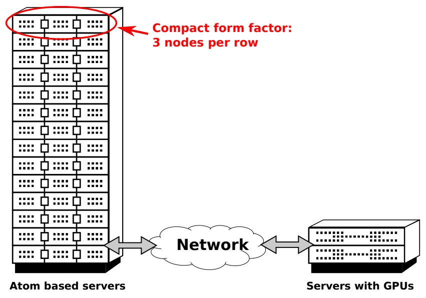


Figure 1: Atom-based cluster complemented with a few nodes comprising high-end GPUs. The high-end GPUs are shared among the many Atom nodes thanks to the remote GPU virtualization mechanism.

accelerated whereas energy-efficiency is maintained because (1) Atom processors are more efficient from an energy point of view than traditional HPC processors and (2) GPUs remain idle much less time because they are concurrently shared among many applications.

2.2. Selection of the GPU Virtualization Solution

Different solutions have been created for virtualizing GPUs. In the context of the CUDA [30] programming environment, frameworks such as vCUDA [31], GridCuda [32], DS-CUDA [33], rCUDA [23], GVirtuS [34] or GViM [35] can be found. In general, all these middleware proposals provide the same API (Application Programming Interface) as the NVIDIA CUDA Runtime API [36] does. Actually, they replace the original NVIDIA CUDA Runtime library by their own implementation. This replacement is their way to virtualize GPUs. Moreover, these solutions also allow GPUs to be concurrently shared among several applications.

The architecture typically used by these middleware solutions is shown in Figure 2. It can be seen in this figure that these GPU virtualization solutions follow a distributed client-server approach. The client side typically consists of the particular implementation of the NVIDIA CUDA Runtime library. This client side is installed in the node that executes the application requesting GPU acceleration. On the other hand, the server side of the middleware typically consists of a daemon running in the node that owns the real GPU.

The distributed architecture shown in Figure 2 works as follows: every time the application performs a call to a CUDA function, that call is received by the client side of the middleware which forwards it to the server side running in the node owning the GPU. There, the request is interpreted and forwarded to the GPU. Upon completing the execution of the CUDA function in the real GPU, the results of such function are returned back from the server to the client side of the middleware. Finally, the client middleware forwards those results to the CUDA application.

It is important to notice that these middleware solutions provide GPU virtualization in a transparent way. That is, applications using remote GPUs are not aware that their calls to CUDA functions are actually being serviced by a GPU located in another cluster node instead of by a GPU located in the local node.

The several GPU virtualization middleware solutions mentioned above present different features. For example, the vCUDA middleware only supports the obsolete CUDA version 3.2. Additionally, it implements only an unspecified subset of the CUDA Runtime API. Furthermore, its communication protocol includes heavy encoding and decoding stages, which considerable increase overall overhead, thus causing a noticeable reduction in performance. On the other hand, GViM is based on the obsolete CUDA version 1.1 and only implements a subset of the CUDA Runtime API. Similarly, GVirtuS is based on the old CUDA version 6.5 while implementing a small fraction of the CUDA API. Regarding GridCuda, it supports the obsolete CUDA version 2.3. Furthermore, there is no public version of this framework which can be used for testing purposes. Finally, DS-CUDA supports the old version 4.1 of CUDA. This middleware includes support for the InfiniBand network fabric, although it presents important constraints such as not implementing data transfers that use host pinned memory.

In this work we use the rCUDA (remote CUDA) middleware because it is the most well maintained framework among the aforementioned ones

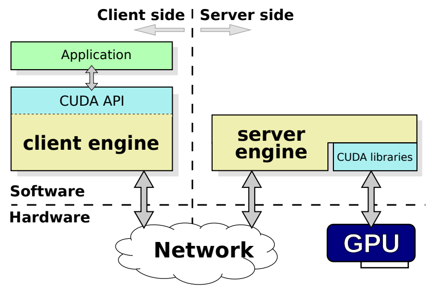


Figure 2: General architecture of remote GPU virtualization solutions.

and, additionally, it provides the best performance when compared to other available remote GPU virtualization solutions [37]. In spite of supporting version 9.2 of CUDA, rCUDA is binary compatible with CUDA. This means that the source code of CUDA applications do not have to be modified for using rCUDA. Moreover, contrary to the rest of remote GPU virtualization solutions revisited above, rCUDA implements the entire CUDA API (except for graphics functions), also providing full compatibility for the other CUDA libraries such as cuSPARSE, cuDNN, cuSOLVER, cuFFT, cuBLAS, etc. Regarding network support, rCUDA features different interconnects, such as TCP/IP, InfiniBand and RoCE. The support for InfiniBand and RoCE network fabrics includes the use of the RDMA features provided by these networks. It is important to notice that support for each of these networks has been implemented and tuned in a specific way so that as much performance as possible is obtained from the underlying interconnect [38][39].

3. Offloading Computations from Atom nodes to Remote GPUs

In this section we provide the performance measurements for five different applications when executed using both Atom CPUs and remote GPUs. Nevertheless, we first describe the hardware and software configurations used in the experiments.

3.1. Hardware and Software Environment

This section presents the details about the hardware and software employed in this study. Experiments have been carried out in a heterogeneous cluster containing 8 Atom-based nodes and four different kinds of regular Xeon-based nodes, owning four different generations of NVIDIA GPUs. The 8 Atom-based nodes were used to execute the applications, either using only CPU cores or using the remote GPUs in the Xeon-based nodes. Atom-based nodes are populated with one C2750 Atom processor, which contain 8 cores working at 2.4 GHz consuming 20 Watts. These nodes have one PCIe v2 x8 connector, where a QDR InfiniBand network adapter is plugged, providing 40 Gb/s[2]. On the other hand, the four different kinds of regular Xeon-based

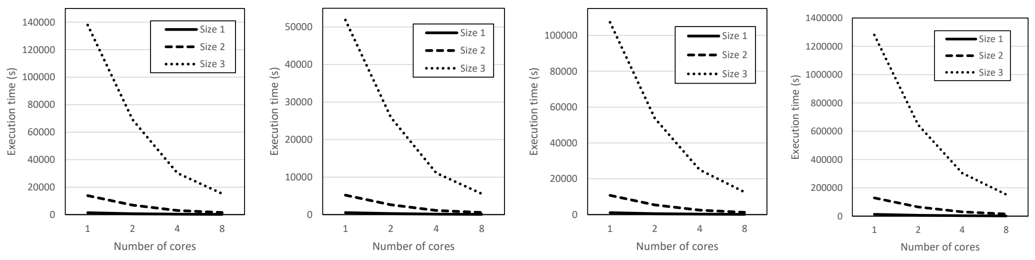
[2]Notice that more modern Atom-based systems are available in the market. For instance, the C3955 Atom processor contains 16 cores working at 2.10 GHz and is equipped with 16 PCIe v3 lanes. However, at the time of writing this paper, the authors only had access to nodes based on the C2750 Atom processor.

nodes contain two 6-core Intel Xeon E5-2620 v2 processors, 32 GB of DDR3 SDRAM memory at 1600 MHz, and one Mellanox ConnectX-3 VPI single-port InfiniBand adapter (FDR InfiniBand). The first kind of these nodes contains Tesla K20 GPUs. The second kind of these nodes contains Tesla K40 GPUs. The third kind of these nodes contains Tesla P100 GPUs. The fourth kind of these nodes contains Tesla V100 GPUs. These Xeon-based nodes equipped with GPUs will be used as rCUDA servers in order to provide GPU services to the Atom-based nodes. Finally, all these nodes (both Atom-based and Xeon-based) are connected by a Mellanox switch SX6025 with FDR compatibility (a maximum rate of 56 Gb/s). Additionally, a set of 32 Xeon-based nodes without GPUs will also be used for some of the experiments in order to execute the applications with up to 256 Xeon cores.

The CentOS 7.3 operating system and the Mellanox OFED 2.4-1.0.4 were used along with the NVIDIA driver 365.96 and CUDA 8.0. The rCUDA version used is 16.11. Regarding the applications to be used in the test, we have used five applications belonging to the physics domain:

1. **Neutral:** a simplified Monte Carlo neutral particle transport solver that supports a number of physical processes as particle histories are tracked [40].
2. **Hot:** a simplified heat diffusion application that uses a conjugate gradient solver without a pre-conditioner to implicitly solve the diffusion equation on a structured grid [41].
3. **CloverLeaf:** solves the compressible Euler equations on a Cartesian two-dimensional grid [41].
4. **Flow:** is a Lagrangian-Eulerian remap hydrodynamics code that is staggered in both space and time [42].
5. **TeaLeaf:** solves the linear heat conduction equation on a two dimensional spatially decomposed regularly grid using a 5 point stencil with implicit solvers [43].

All the five applications provide support for performing their computations on CPU (single node an also multi-node distributed with MPI [44], except for the Neutral application) as well as for carrying out their computations on GPU. Additionally, all the applications are able to use multiple GPUs except the Neutral and Hot applications, which can only use a single GPU. Therefore, these applications are representative of realistic scenarios for comparing the performance of low-power processors deployments when



(a) csp (b) scatter (c) split (d) stream

Figure 3: Execution time of the Neutral application when scaled up to 8 Atom cores in a single node. Different benchmarks for the application are considered as well as several problem sizes for each benchmark.

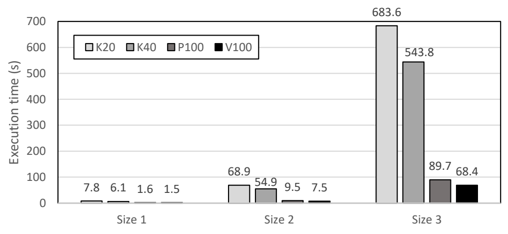
only CPUs are used with the performance of using remote CUDA GPUs located in regular Xeon-based codes.

3.2. Neutral

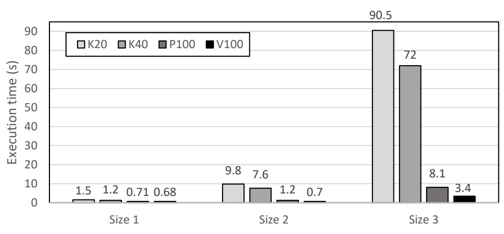
Figure 3 shows the execution time of the Neutral application when executed in a single Atom-based node with up to 8 cores. Four different problems are executed in the experiments:

1. **center square problem (csp):** particles stream until they encounter a region of high density in the center of the slab.
2. **scatter:** particles will mostly collide within the slab.
3. **split:** particles are spread evenly in regions of high and low density to match the number of events of each type.
4. **stream:** particles stream across the mesh multiple times per timestep, colliding infrequently.

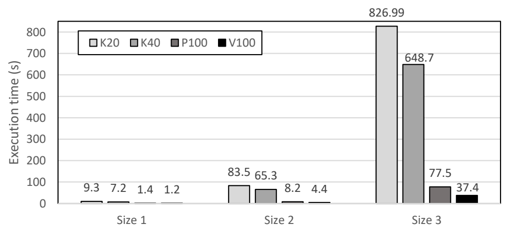
In addition to use four different benchmarks for the Neutral application, each of the benchmarks are configured with three problem sizes, which basically differ in the amount of particles considered during the simulation. In this regard, all the problem sizes use a 4000 x 4000 two dimensional grid. Problem size 1 makes use of 250,000 particles. Problem size 2 considers 2,500,000 particles. Finally, problem size 3 uses 25,000,000 particles in the executions. It can be seen in Figure 3 that execution time grows super-linearly with problem size for all the four benchmarks. Additionally, increasing the amount of cores noticeably reduces total execution time. Notice that this



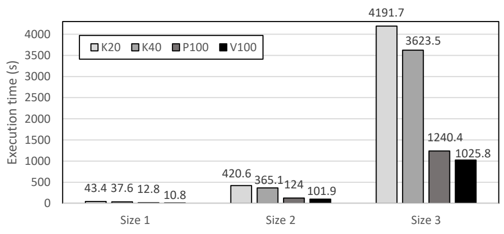
(a) csp



(b) scatter



(c) split



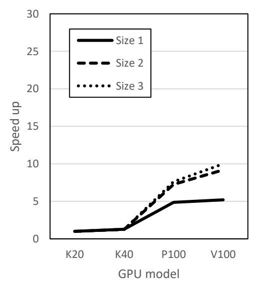
(d) stream

Figure 4: Execution time of the Neutral application when executed in an Atom system using a remote GPU thanks to the rCUDA middleware. Different benchmarks for the application are considered as well as several problem sizes for each benchmark. Experiments are conducted on four generations of GPUs.

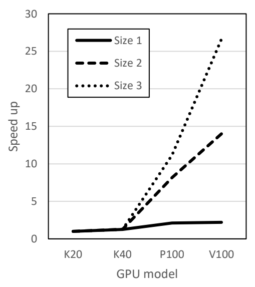
application does not allow the use of the MPI library in order to distribute the simulations across several nodes [40]. Therefore, the performance of this application is constrained to the computing resources of a single node.

Figure 4 depicts the execution time of the Neutral application for the four different benchmarks considered when a remote GPU is used thanks to the rCUDA middleware. In this case, the application is executed in an Atom node by using a single CPU core whereas a remote GPU located in a regular Xeon node is used to carry out the computations. Notice that this application is only able to use a single GPU. Furthermore, the QDR InfiniBand network is leveraged to access the remote GPU. The same three problem sizes used in the previous experiments are also used in this case. Four generations of GPUs are used in the performance analysis.

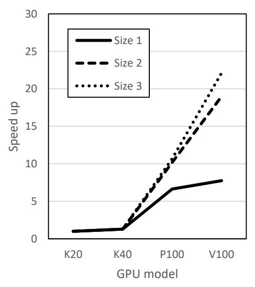
It can be seen in Figure 4 that execution time is reduced as better GPUs are used to execute the Neutral application. Actually, we can see a large performance improvement when moving from the K20 and K40 GPUs (Kepler architecture) to the P100 GPU (Pascal architecture). The Volta architecture used in the V100 GPU further contributes to reduce total execution time.



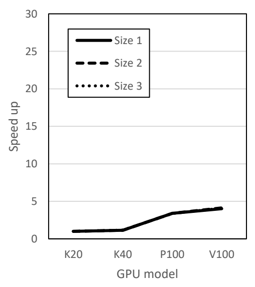
(a) csp



(b) scatter

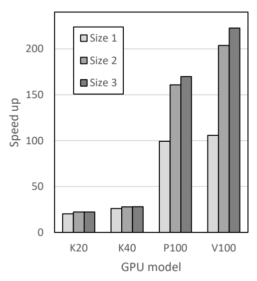


(c) split

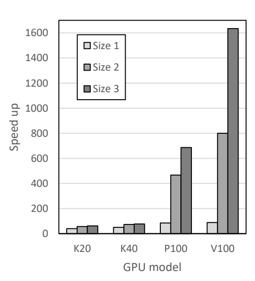


(d) stream

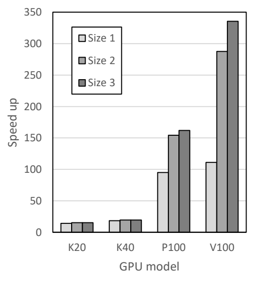
Figure 5: Speed up attained by the Neutral application when executed with different generations of remote GPUs. Speed ups are relative to the execution time obtained with the K20 GPU.



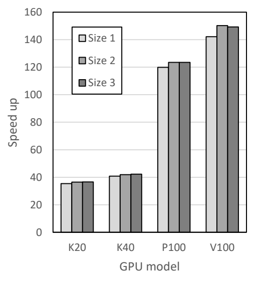
(a) csp



(b) scatter



(c) split



(d) stream

Figure 6: Speed up attained by the Neutral application when executed with different generations of remote GPUs. Speed ups are relative to the execution time obtained when using 8 Atom cores in a single node.

Figure 5 shows the speed up achieved by each of the GPU generations with respect to the K20 GPU. It is remarkable the very different speed up attained by each of the GPU generations for each of the benchmarks considered to test the Neutral application. In this regard, the V100 GPU, for instance, achieves more than 25x speed up for the scatter problem whereas it only achieves 4x speed up for the stream benchmark. Actually, for this benchmark, all the GPU models behave similarly in terms of speed up.

Figure 6 presents the most interesting results in this analysis: the speed up obtained when using a remote GPU with rCUDA instead of performing the computations using the 8 CPU cores of the low-power Atom-based nodes. Remember that this is our proposal in this paper: using remote GPU to boost

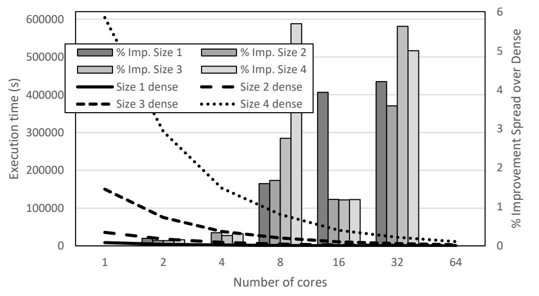


Figure 7: Execution time of the Hot application when scaled up to 64 Atom cores. Different problem sizes are considered.

the performance of low-power deployments. It can be seen in Figure 6 that using a remote GPU is noticeable beneficial, specially when using the recent generations of GPUs. In this regard, the V100 GPU reports an speed up of up to 1600x for the scatter benchmark. Notice that speed ups in Figure 6 increase with problem size. Therefore, we can expect that for bigger problem sizes the benefit of using a remote GPU will be even more remarkable even for older GPU generations.

3.3. Hot

Figure 7 shows the execution time of the Hot application. Contrary to the Neutral application, the Hot application can use MPI in order to distribute the computations across a large number of cores. Figure 7 shows the execution time of the Hot application when up to 64 Atom cores are used (up to 8 Atom-based nodes in our testbed). An important concern is related to the way of distributing the cores participating in the execution of the application across the nodes of the testbed. In this regard, two approaches can be followed. In the first approach, nodes are increasing filled up as we are using more and more cores. In this way, when up to 8 cores are used to execute the application, then a single node is used (remember that each node has eight Atom cores). In the case of using 16 cores, two nodes are used, and so on. The second approach for distributing cores across nodes is spreading the cores across as many nodes as possible. In this way, when two cores are used, two different nodes are employed. Similarly, when 4 cores are used to execute the application, four different nodes are used (each node hosts one of the cores). Given that our testbed has only 8 nodes, when 16 cores are used for executing the Hot application, 2 cores are located at each node. We can refer to the first approach as the "dense" approach whereas the second approach can be referred to as the “spread” approach. Each of them has benefits and drawbacks. The dense approach reduces inter-node communication while reducing the benefit of the processor caches because cache lines will probably stay for shorter in the cache. On the contrary, the spread approach increases inter-node communications but might accelerate computations because smaller competition exists among cores for the cache space. At the end, depending on the application trade-off between communications and computations, one approach might be better than the other.

Figure 7 shows the execution time of the Hot application when the dense approach is followed (lines in the figure). Additionally, bars in the figure show the benefit, in percentage, of using the spread approach with respect to using the dense core distribution. Four problem sizes are considered. Differences among problem sizes mainly consist on using a larger two dimensional grid. Table 1 shows the grid dimensions for each of the problem sizes considered. It can be seen in Figure 7 that is noticeably reduced with the number of cores involved in the execution of the application. Also, it can be seen that differences in performance between the two core distribution policies is not larger than 6%. This behavior is similar to the one obtained when executing the application in a cluster of regular Xeon-based nodes (Figure 8). In this case, up to 32 nodes are used to execute the application. Therefore, when 32 or less cores are involved in the execution of the application, only one core per node is used.

The Hot application is able to offload the computations to a single GPU. Figure 9 shows the execution times attained by this application when the same four GPU generations used in the previous section are used with rCUDA remotely. As with the Neutral application, results in Figure 9 are obtained by running the application using a single core in the Atom-based node while using a remote GPU located in a regular Xeon-based node. It can be seen that using newer GPU generations reduces execution time, specially for the larger

Table 1: Most relevant parameters used for the executions of the Hot application.

Size	Spatial dimensions	# iterations
1	512 x 512	10
2	1024 x 1024	10
3	2048 x 2048	10
4	4096 x 4096	10

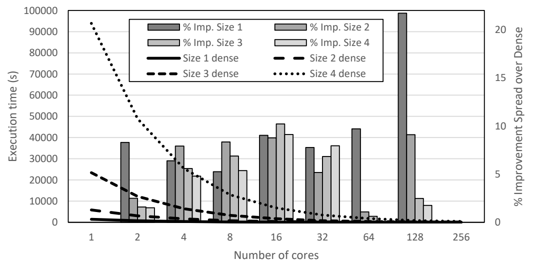


Figure 8: Execution time of the Hot application when scaled up to 256 Xeon cores. Different problem sizes are considered.

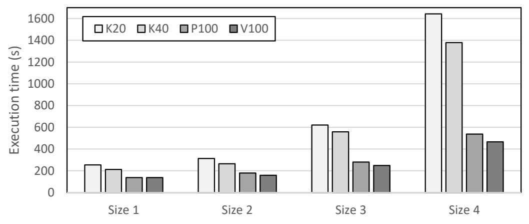


Figure 9: Execution time of the Hot application when executed in an Atom system using a remote GPU thanks to the rCUDA middleware. Several problem sizes for each benchmark. Experiments are conducted on four generations of GPUs.

problem sizes. Figure 10 shows the speed up attained by each of the GPU generation with respect to the use of the K20 GPU. If compared to the speed ups obtained for the Neutral application, it can be seen that in the case of the Hot application the speed ups are smaller. In this case, communications with the remote GPU represent a larger fraction of the execution time, thus reducing the benefits of the newer GPU generations.

Finally, Figure 11 shows the most important kind of results in this study: the speed up of using a remote GPU with respect to using the CPU cores to carry out the computations of the application. In this case, given that the Hot application is able to scale to a large amount of nodes thanks to the use of the MPI library, speed ups shown in Figure 11 are relative to the execution times of the application when 64 Atom cores are used. It can be seen in the figure that speed ups grows with problem size because the ratio between computation and communication is reduced and this benefits to rCUDA even using a relatively slow network as QDR InfiniBand. On the other hand, it is remarkable the large speed up attained by the V100 GPU for the largest problem size considered with respect to the old K20 GPU model.

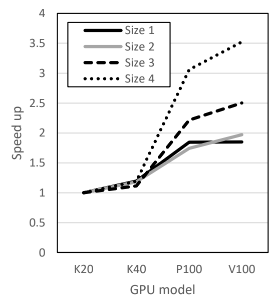


Figure 10: Speed up attained by the Hot application when executed with different generations of remote GPUs. Speed ups are relative to the execution time obtained with the K20 GPU.

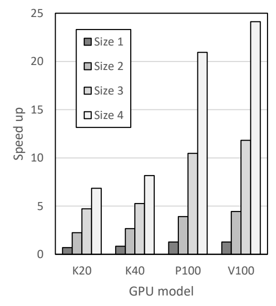


Figure 11: Speed up attained by the Hot application when executed with different generations of remote GPUs. Speed ups are relative to the execution time obtained when using 64 Atom cores.

3.4. CloverLeaf

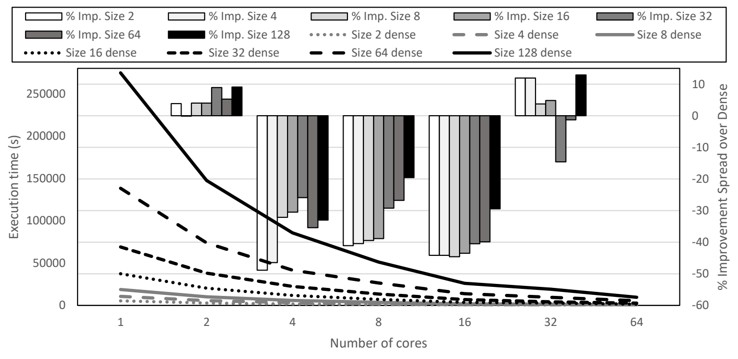


Figure 12: Execution time of the CloverLeaf application when scaled up to 64 Atom cores. Different problem sizes are considered.

Table 2: Most relevant parameters used for the executions of the CloverLeaf application.

Size	Spatial dimensions	# iterations
2	1920 x 960	2955
4	1920 x 1920	2955
8	3840 x 1920	2955
16	3840 x 3840	2955
32	7680 x 3840	2955
64	7680 x 7680	2955
128	15360 x 7680	2955

Figure 12 shows the execution time of the CloverLeaf application when executed with up to 8 Atom-based nodes (8 cores per node) using the dense policy described before. Different problem sizes are considered in the figure in order to better analyze the scalability of this application. Table 2 shows the parameters used for each of the problem sizes depicted in Figure 12. It can be seen in the table that larger problem sizes translate into a larger cartesian 2D grid, although the amount of iterations performed for all the sizes is the same.

It can be seen in Figure 12 that the scalability of the application is very good: as the amount of cores is increased, the execution time of the application is proportionally reduced. Furthermore, the application also scales very well according to problem size: as problem size increases, execution time increases proportionally. On the other hand, and contrary to the Hot

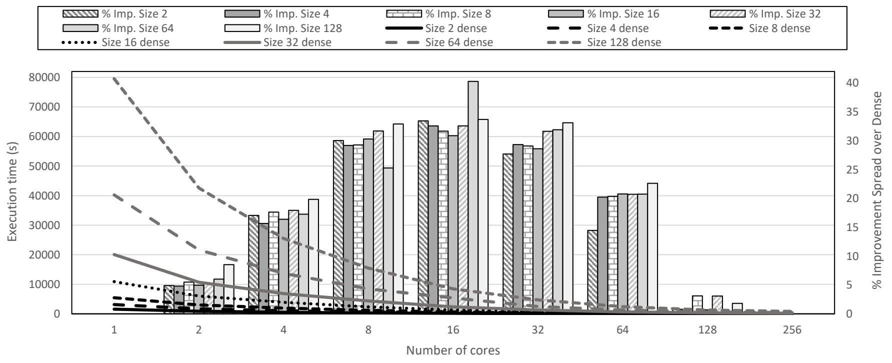


Figure 13: Execution time of the CloverLeaf application when scaled up to 256 Xeon cores. Different problem sizes are considered.

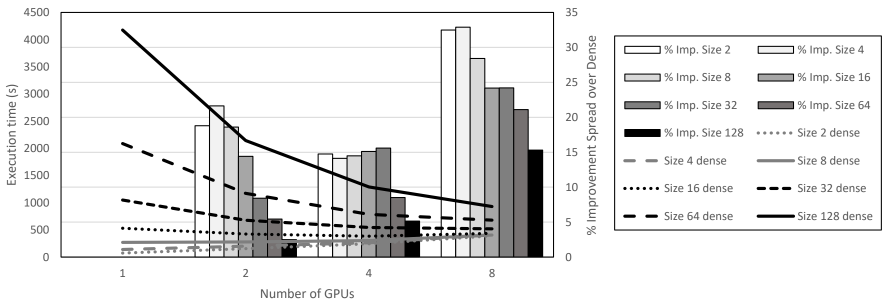
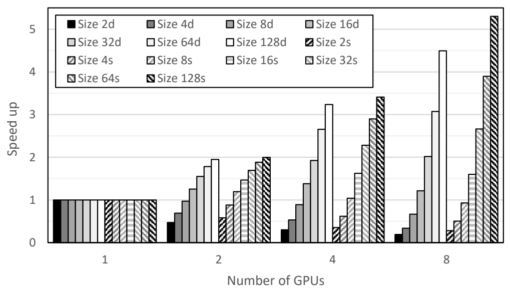
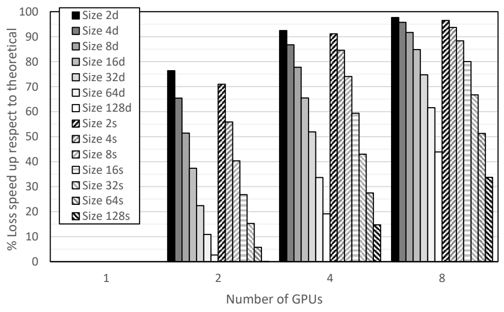


Figure 14: Execution time of the CloverLeaf application when executed in an Atom system using up to 8 remote K20 GPUs thanks to the rCUDA middleware. Different problem sizes are considered.



(a) Speed up



(b) Loss in speed up

Figure 15: Analysis of the execution time of the CloverLeaf application when executed in an Atom system using K20 remote GPUs. Different problem sizes are considered.

application, in general the spread distribution of cores among nodes slows down the execution of the application. This behavior is very different to the one obtained when using up to 256 Xeon cores to execute the application (Figure 13) where the usage of the spread policy noticeably improves performance when the sharing degree is small. Notice, however, that the Xeon-based cluster makes use of an FDR InfiniBand network fabric. Additionally, the characteristics of the processors and the main memory bandwidth are also different. Thus, it is not surprising the different behavior.

Figure 14 shows the execution time when the CloverLeaf application is executed using up to 8 remote K20 GPUs thanks to the rCUDA middleware. In this experiments, each of the GPUs is driven by a single core. Therefore, the amount of CPU cores equals the amount of GPUs used in each of the

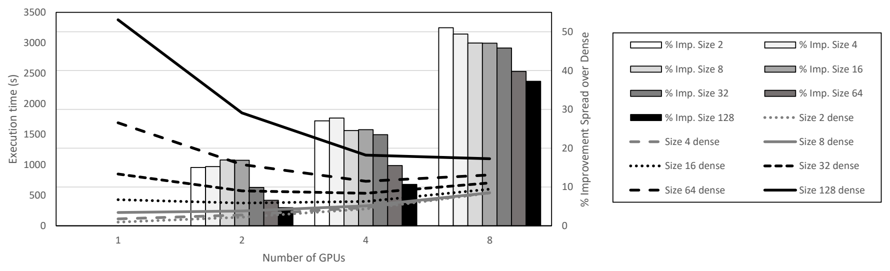
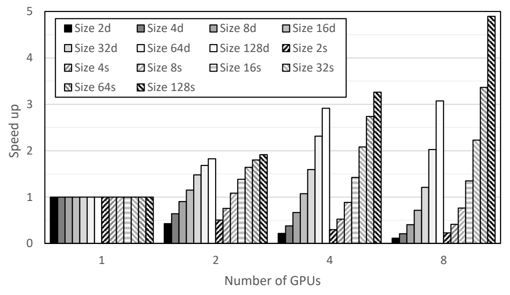
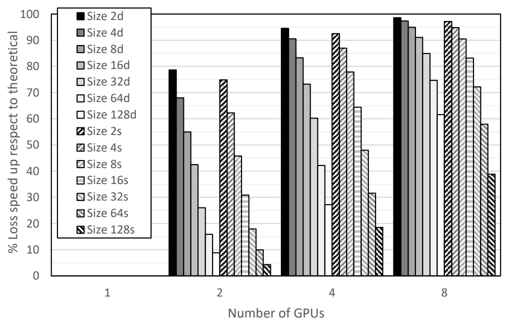


Figure 16: Execution time of the CloverLeaf application when executed in an Atom system using up to 8 remote K40 GPUs thanks to the rCUDA middleware. Different problem sizes are considered.



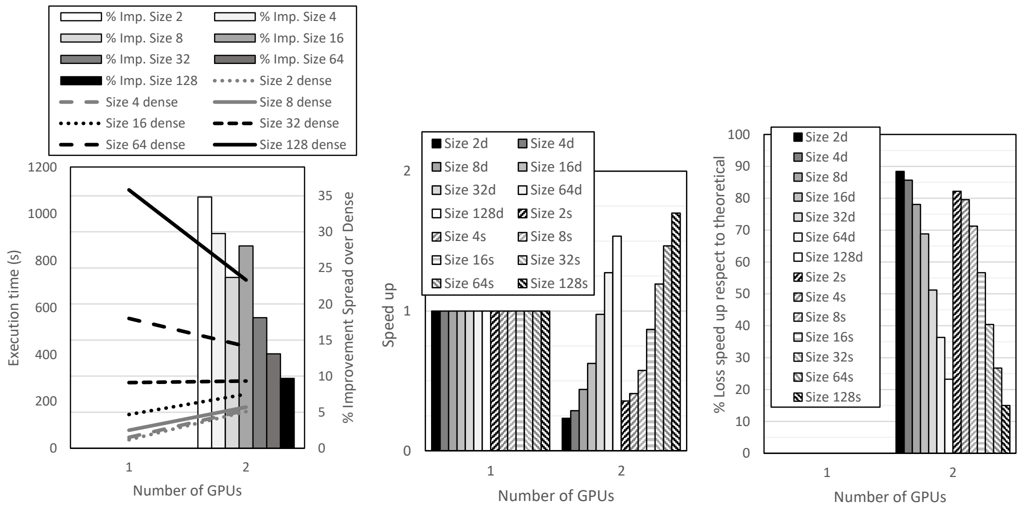
(a) Speed up



(b) Loss in speed up

Figure 17: Analysis of the execution time of the CloverLeaf application when executed in an Atom system using K40 remote GPUs. Different problem sizes are considered.

executions. Additionally, as for the CPU-based executions, two different policies can be followed to locate the CPU cores. In the "dense" approach, all the processes run in the same node (the Atom-based nodes used in this paper have 8 cores). In the "spread" approach, each of the processes of the application are placed in a different Atom-based node. Figure 14 also shows the percentage of improvement of using the spread configuration with respect to using the dense approach. Several conclusions can be derived from Figure 14. The first one is that execution times for small problem sizes are not improved as more GPUs are used. On the contrary, it can be seen that for the smallest problem size, execution times are worsened. The second conclusion that can be derived from Figure 14 is that, for the larger problem sizes, execution time is proportionally reduced as the amount of remote GPUs is increased. Finally, it can also be seen in the figure that the



(a) Execution time (b) Speed up (c) Loss in speed up

Figure 18: Execution time of the CloverLeaf application when executed in an Atom system using P100 remote GPUs. Different problem sizes are considered.

benefit of the spread approach is larger when 8 remote GPUs are used. The reason for this bigger improvement is that when all the cores are placed in the same node, all the data transfers to the remote GPUs (located in different remote nodes) are sequentially performed because there is only one network interface in the node. This makes that data transfers to/from the remote GPUs become an important bottleneck. On the contrary, when cores are distributed across different nodes, the larger amount of network interfaces to concurrently communicate with remote GPUs improves overall performance.

Figure 15 further analyzes the results presented in Figure 14. On the one hand, Figure 15(a) shows the speed up attained as more GPUs are used. It can be seen in this figure that speed up does not reach the amount of GPUs used in the execution of the application. For instance, when 8 remote K20 GPUs are used, speed up for the largest problem size is about 5x. This means that some speed up has been lost with respect to the theoretical one. Figure 15(b) shows this loss in speed up. It can be seen in the figure that the loss in speed up is reduced as problem size increases.

Figures 16 and 17 present similar analysis although in this case up to 8 remote K40 GPUs are used. In this case, it can be seen in Figure 16 that using

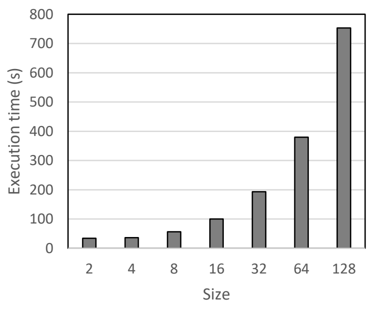


Figure 19: Execution time of the CloverLeaf application when executed in an Atom system using a V100 remote GPU. Different problem sizes are considered.

a large amount of remote K40 GPUs with the dense approach is not beneficial except for the largest problem size. The reason is that the K40 GPU presents a better performance than the K20 GPU whereas the bandwidth to the remote GPU remains the same (QDR InfiniBand network interface at the Atom node). Therefore, the worse ratio between communication and computation experienced by the CloverLeaf application causes that the relative weight of data transfers increases, thus reducing the benefit of using a large amount of remote K40 GPUs.

Figure 18 shows a similar analysis when up to 2 remote P100 GPUs are used (in this case we present results for up to 2 remote P100 GPUs because no more P100 were available in our cluster). Similar conclusions to those derived for the case of using remote K20 and K40 GPUs can be drawn also in this case. Additionally, Figure 19 presents the execution time of the CloverLeaf application when a remote V100 GPU is used (only one V100 GPU is available in our cluster).

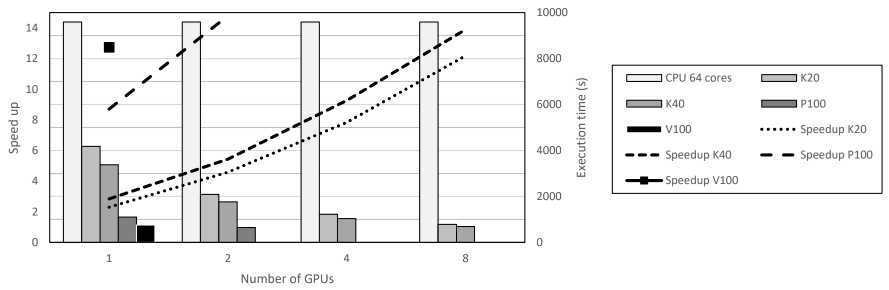


Figure 20: Speed up obtained when executing the CloverLeaf application using up to 4 different generations of remote GPUs. Different problem sizes are considered.

Comparing the execution times when using CPUs and when using remote GPUs presented in the previous figures (Figures 12 to 19), it can be quickly seen that execution times when remote GPUs are used are noticeable smaller. Indeed, noticeable speed ups are achieved, as shown in Figure 20. This figure depicts the execution time for the largest problem size when 64 Atom cores are used and compares that result against the times required when making use of remote GPUs. The spread policy has been used for locating the processes that drive the remote GPUs. The figure also shows the achieved speed up. It can be seen that just by using two remote P100 GPUs provides 15x smaller execution times with respect to the case of using 64 Atom cores. Similarly, using a single remote V100 translates into 14x speed up with respect to using 8 Atom-based nodes.

3.5. Flow

The performance of the Flow application follows a similar trend to that of the CloverLeaf one. Figure 21 shows the execution times for the different problem sizes described in Table 3 when up to 64 Atom cores are used. It can be seen that this application presents similar scalability features to those of the CloverLeaf application. This can also be seen in Figure 22, that depicts the performance of the Flow application when executed with up to 256 Xeon cores using the FDR InfiniBand fabric.

When the Flow application is executed by using remote GPUs with rCUDA, execution times are noticeably smaller, as shown in Figure 23 for the remote K20, K40 and P100 GPU cases. Similarly to the CloverLeaf

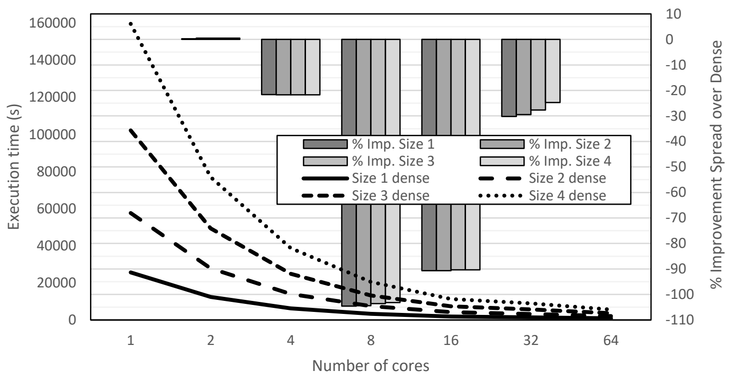


Figure 21: Execution time of the Flow application when scaled up to 64 Atom cores. Different problem sizes are considered.

Table 3: Most relevant parameters used for the executions of the Flow application.

Size	Spatial dimensions	# iterations
1	4000 x 4000	2000
2	6000 x 6000	2000
3	8000 x 8000	2000
4	10000 x 10000	2000

application, using a large amount of remote GPUs reports an increment in the execution time when the dense policy is used. This can be seen in Figures 23(a), 23(d) and 23(g) for the three GPU models. The increment in execution time is more noticeable for smaller problem sizes. Nevertheless, when the spread policy is considered, using more GPUs is beneficial. This can be seen in Figures 23(b), 23(e) and 23(h) where the speed up attained by the spread policy increases as the amount of GPUs used in the execution of the application increases.

Regarding the usage of the remote V100 GPU available in our testbed, Figure 24 shows the execution times of the Flow application for all the four problem sizes considered. It can be seen in the figure that execution time increases with problem size, as expected. However, execution times when the remote V100 GPU is used are noticeably lower than for the other GPU models, specially for the oldest ones (K20 and K40).

Figure 25 compares the execution time for the largest problem size when only CPU cores are used against those achieved when the application is executed by using remote GPUs. In the case of the CPU-based executions,

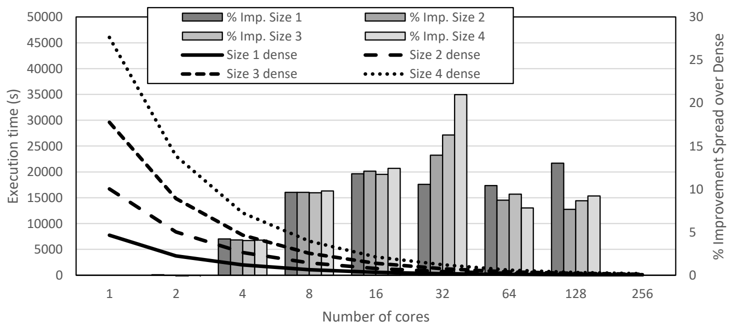
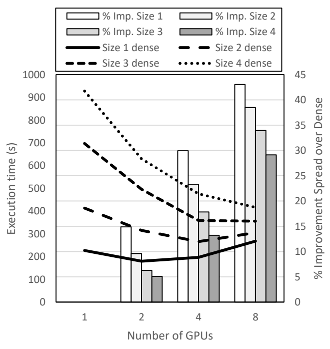
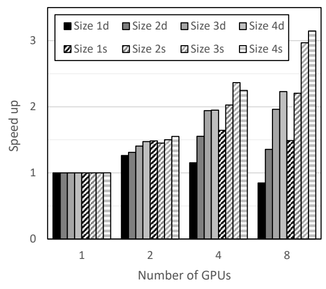


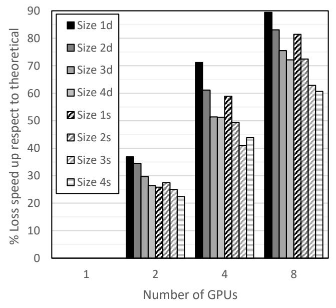
Figure 22: Execution time of the Flow application when scaled up to 256 Xeon cores. Different problem sizes are considered.



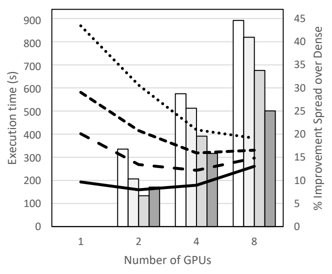
(a) K20 execution time



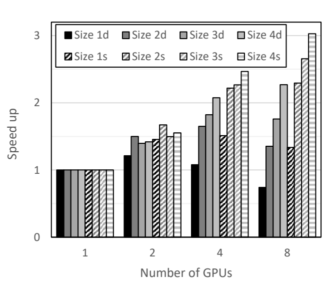
(b) K20 speed up



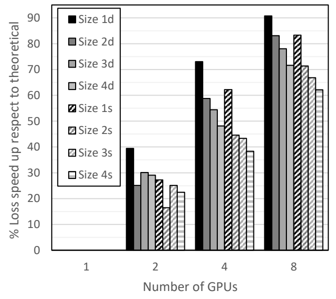
(c) K20 loss in speed up



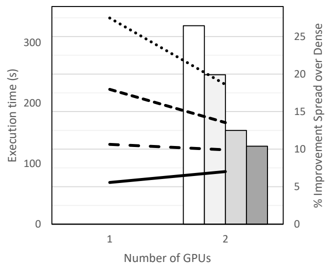
(d) K40 execution time



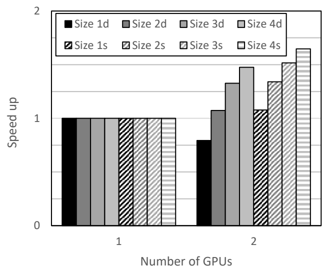
(e) K40 speed up



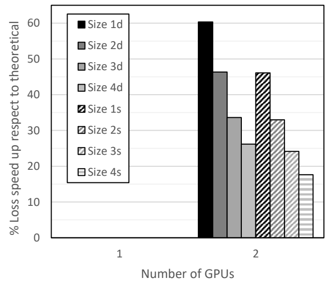
(f) K40 loss in speed up



(g) P100 execution time



(h) P100 speed up



(i) P100 loss in efficiency

Figure 23: Execution time and analysis of the Flow application when executed in an Atom system using remote GPUs. Different problem sizes are considered.

all the 64 Atom cores across the 8 nodes of the testbed are used, as in the case of the CloverLeaf application presented in the previous section. On the other hand, similarly to what we did in the previous section, results for GPU-based executions use the spread policy where each of the processes driving a GPU is placed in a different node. It can be seen in Figure 25 that 27x smaller

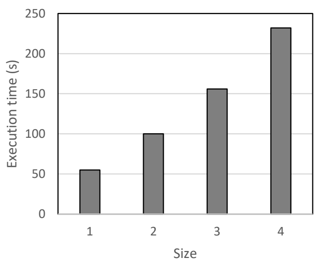


Figure 24: Execution time of the Flow application when executed in an Atom system using a V100 remote GPU. Different problem sizes are considered.

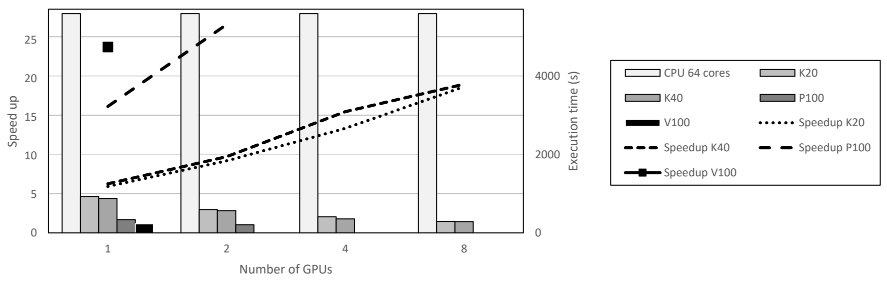


Figure 25: Speed up obtained when executing the Flow application using up to 4 different generations of remote GPUs. Different problem sizes are considered.

execution times are obtained when using just two remote P100 GPUs. It is remarkable that using a single remote V100 GPU located in a Xeon node reports a speed up of 24x.

3.6. TeaLeaf

Similarly to the CloverLeaf and Flow applications, we can perform a CPU versus GPU analysis for the TeaLeaf application. In this case, we have used 5 different problem sizes for the executions of the TeaLeaf application.

Table 4: Most relevant parameters used for the executions of the TeaLeaf application.

Size	Spatial dimensions	# iterations
1	10 x 10	10000
2	250 x 250	10000
3	1000 x 1000	10000
4	4000 x 4000	10000
5	10000 x 10000	10000

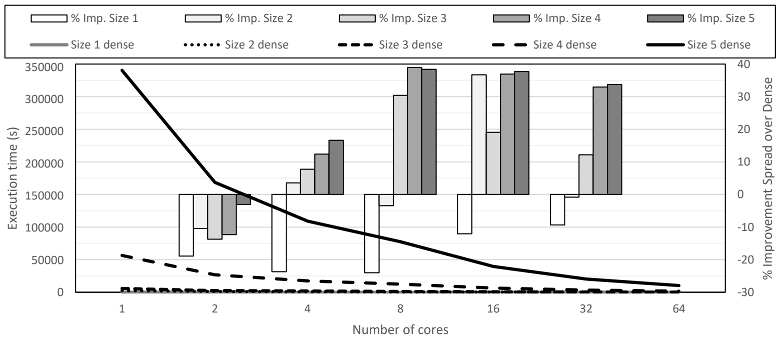
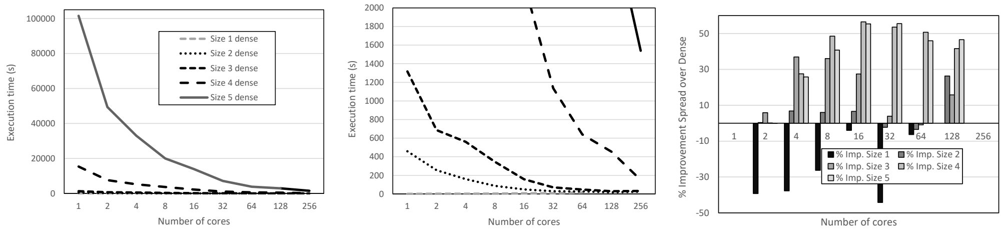


Figure 26: Execution time of the TeaLeaf application when scaled up to 64 Atom cores. Different problem sizes are considered.



(a) Execution time (b) Zoom of execution time (c) Improvement percentage

Figure 27: Execution time of the TeaLeaf application when scaled up to 256 Xeon cores. Different problem sizes are considered.

The difference among the several problem sizes is the dimension of the two-dimensional spatial grid used in the simulations. Table 4 shows the exact dimensions for each of the problem sizes. It can be seen in the table that the spatial dimensions for the different problem sizes encompasses a large size variety.

Figure 26 shows the execution time of the TeaLeaf application when scaled up to 64 Atom cores across all the 8 nodes in our testbed. As in previous sections, we present with lines the execution times obtained with the dense policy whereas bars represent the percentage of improvement attained by the spread policy. It can be seen in the figure that using a larger amount of Atom cores reduces the execution time, as expected. However, differences in performance among both process distribution policies depend on problem size. A similar behavior is obtained when the TeaLeaf application is executed with up to 256 Xeon cores in a cluster leveraging the FDR InfiniBand network

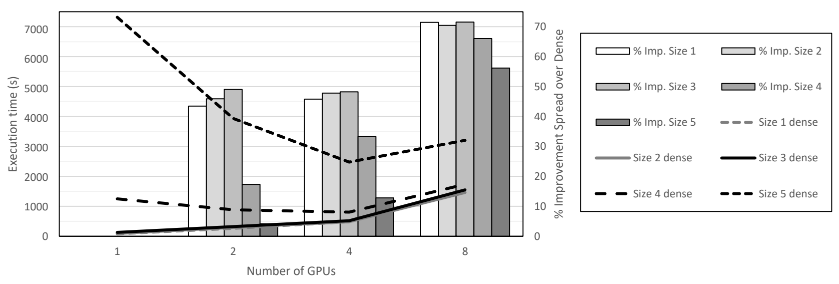
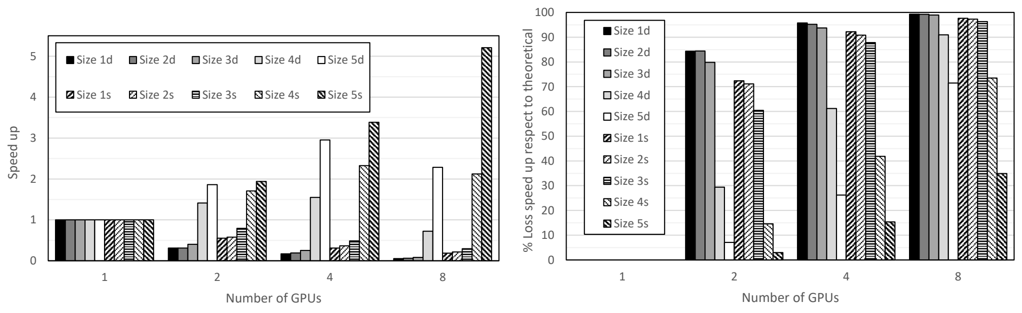


Figure 28: Execution time of the TeaLeaf application when executed in an Atom system using up to 8 remote K20 GPUs thanks to the rCUDA middleware. Different problem sizes are considered.



(a) Speed up

(b) Speed up loss

Figure 29: Analysis of the execution time of the TeaLeaf application when executed in an Atom system using K20 remote GPUs. Different problem sizes are considered.

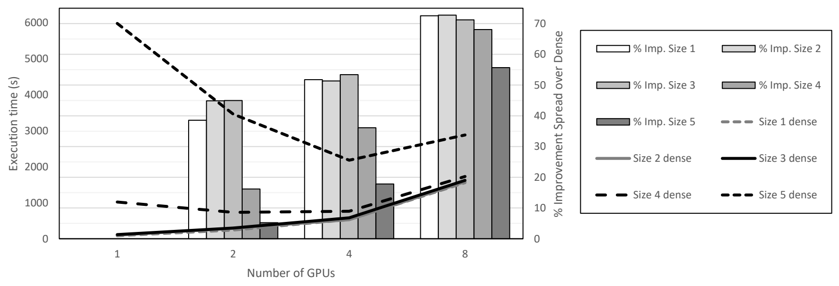
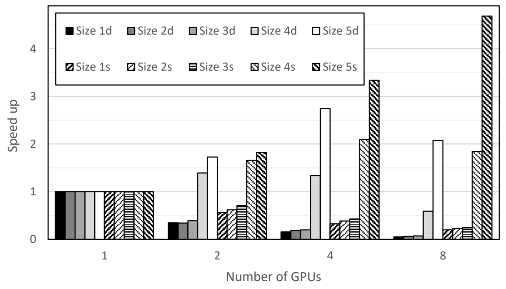


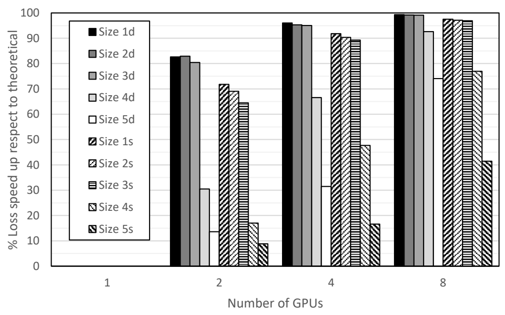
Figure 30: Execution time of the TeaLeaf application when executed in an Atom system using up to 8 remote K40 GPUs. Different problem sizes are considered.

fabric, as shown in Figure 27.

Similarly to the CloverLeaf and Flow applications, the TeaLeaf appli-



(a) Speed up



(b) Speed up loss

Figure 31: Analysis of the execution time of the TeaLeaf application when executed in an Atom system using K40 remote GPUs. Different problem sizes are considered.

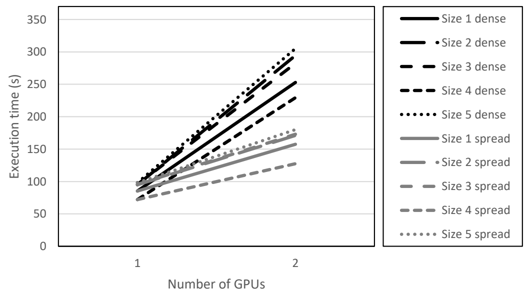


Figure 32: Execution time of the TeaLeaf application when executed in an Atom system using up to 2 remote P100 GPUs. Different problem sizes are considered.

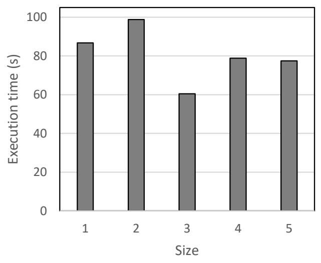


Figure 33: Execution time of the TeaLeaf application when executed in an Atom system using a remote V100 GPU. Different problem sizes are considered.

cation allows to use multiple GPUs. As it was the case for the previous applications, each of the GPUs is driven by a different process. Thus, in order to use multiple GPUs we must leverage the MPI library. Figure 28

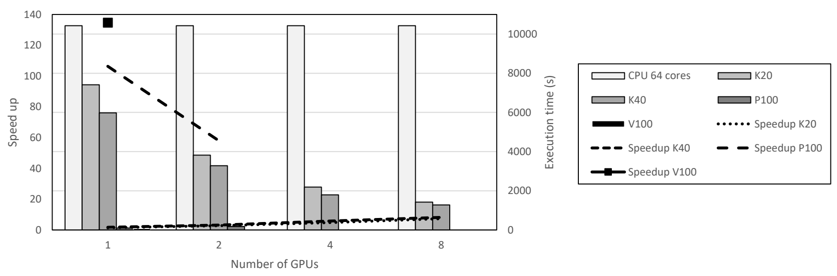
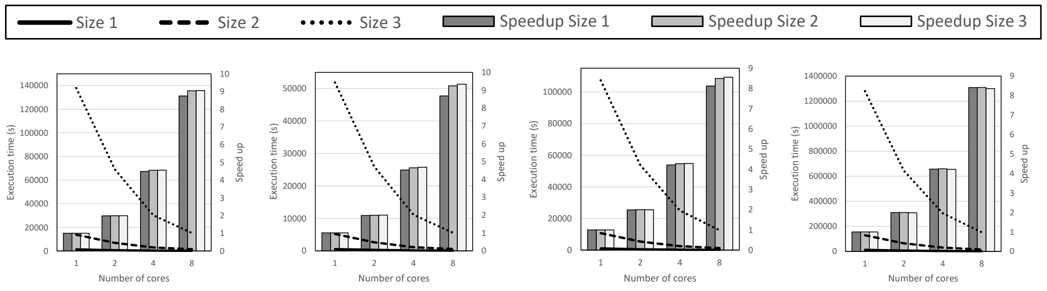


Figure 34: Speed up obtained when executing the TeaLeaf application using up to 4 different generations of remote GPUs. Different problem sizes are considered.

presents the execution time of the TeaLeaf application when using up to 8 remote K20 GPUs. It can be seen that using 8 GPUs is not beneficial in the case of applying the dense process distribution policy. On the contrary, when the spread policy is used, performance is noticeably improved for the larger problem sizes, as shown in Figure 29.

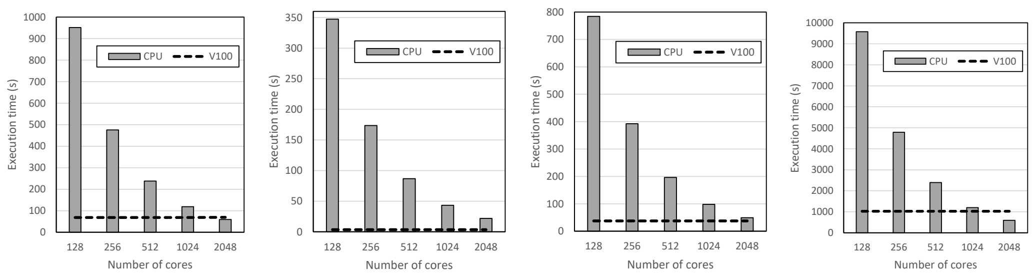
Figures 30 and 31 show a similar analysis when up to 8 remote K40 GPUs are used for executing the TeaLeaf application. It can be seen in these figures that a similar behavior is obtained. On the other hand, Figure 32 presents execution time when up to two remote P100 GPUs are used to execute the application. It can be seen in this figure that using the spread policy reports lower execution times for two remote GPUs with respect to the dense policy. However, in all cases, using two remote P100 GPUs increases execution times. The reason is that problem sizes are too small for this GPU. Similar conclusions about the small size of the problems considered can be derived from Figure 33, where the extraordinary performance of the V100 GPU is hidden behind the communications between the Atom node and the remote GPU.

Finally, Figure 34 presents the main goal of this paper: comparing the performance of using CPU-only executions against the performance of using GPU-based executions in low-power processors thanks to the rCUDA middleware. It can be seen in the figure that a noticeable speed up is attained when the recent V100 GPU model is used to execute the TeaLeaf application. This result is aligned with the results obtained in previous sections.



(a) csp (b) scatter (c) split (d) stream

Figure 35: Speed up attained by the Neutral application when scaled up to 8 Atom cores in a single node. Different benchmarks for the application are considered as well as several problem sizes for each benchmark.



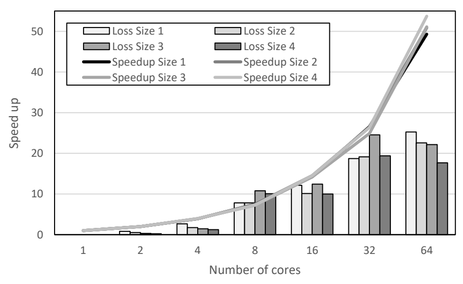
(a) csp (b) scatter (c) split (d) stream

Figure 36: Projection of the execution time of the Neutral application over a large amount of Atom cores. Comparison against the execution time using one remote V100 GPU.

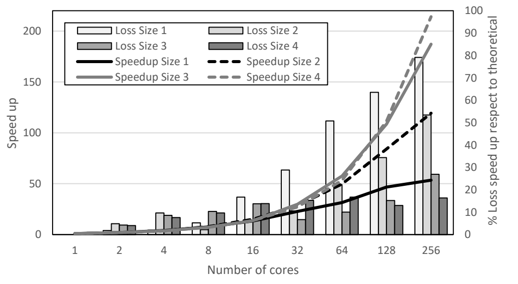
4. Projecting Performance on Larger Systems

All the five applications analyzed in the previous section present good scalability properties. This makes possible to carry out performance estimations for larger systems composed of a very large amount of Atom-based nodes. Additionally, those estimations can be compared against the performance of these applications when using remote GPUs. In this section we compare those estimations with the performance achieved when a single V100 GPU is used with the rCUDA middleware.

Figure 35 shows the speed up experienced by the Neutral application when the amount of cores used to execute it is increased (remember that this application did not use MPI and therefore it is constrained to a single node).

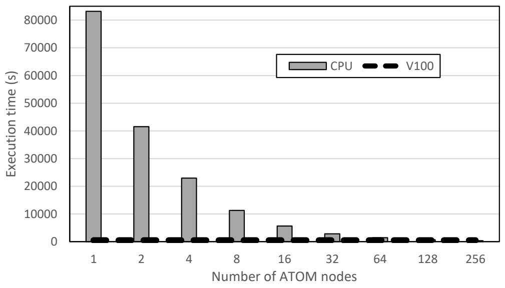


(a) Up to 64 Atom cores

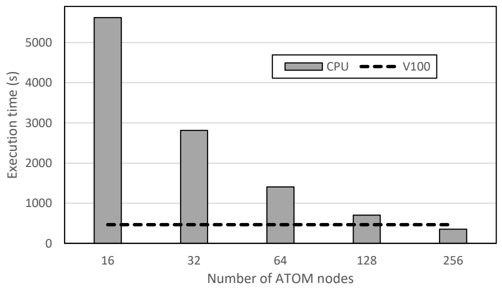


(b) Up to 256 Xeon cores

Figure 37: Speed up attained by the Hot application when scaled up to a large amount of cores. Different problem sizes are considered.



(a) Projection over up to 256 Atom-based nodes

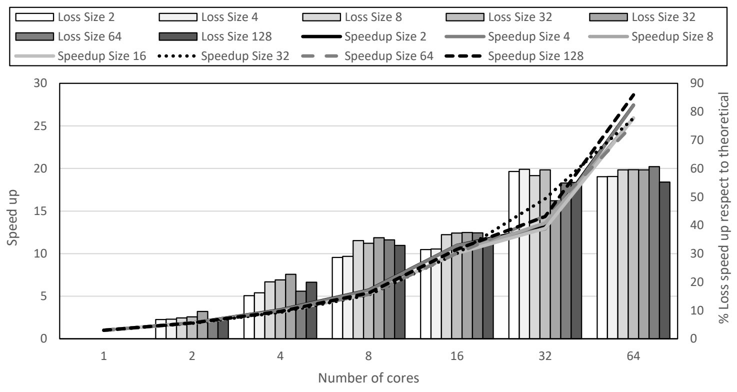


(b) Zoomed view of the projection

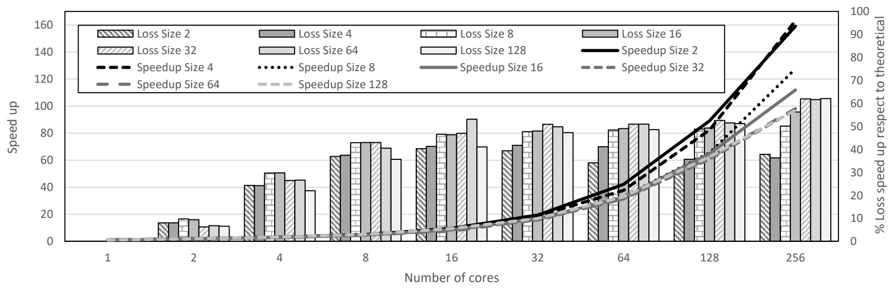
Figure 38: Projection of the execution time of the Hot application over a large amount of Atom-based nodes. Comparison against the execution time using one remote V100 GPU.

It can be seen that the Neutral application presents very good scalability features. Figure 36 shows the performance estimation for the four benchmarks considered when up to 2048 Atom cores are used. The largest problem size has been used for these estimations. Notice that currently it is impossible to integrate 2048 Atom cores in a single node. Thus, the results in Figure 36 show how beneficial is to use remote GPUs with rCUDA.

Figure 37 shows the scalability of the Hot application when executed with up to 64 Atom cores and with up to 256 Xeon cores. It can be seen that this application does not present the extraordinary scalability characteristics of the Neutral application. Figure 37 shows that relative speed up is reduced as more cores are used for executing the application. For the largest problem size, loss in speed up gets steady in about 15%. Taking this trend in



(a) Up to 64 Atom cores

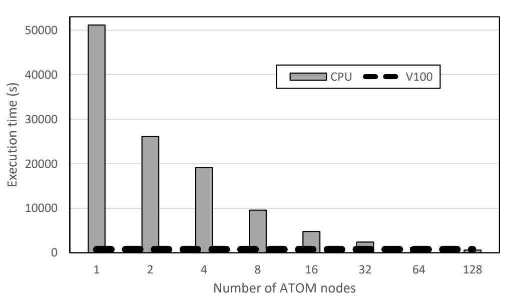


(b) Up to 256 Xeon cores

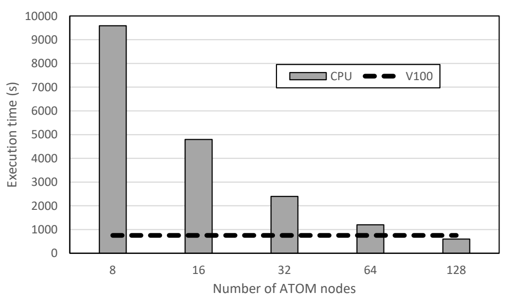
Figure 39: Speed up attained by the CloverLeaf application when scaled up to a large amount of cores. Different problem sizes are considered.

consideration, we can estimate the performance of the Hot application for a very large amount of Atom cores, as show in Figure 38. It can be seen in this figure that only when the amount of nodes is 256 or larger (2048 Atom cores), CPU performance is better than that obtained by using a single remote V100 GPU. Again, this figure shows that introducing GPU acceleration in low-power Atom-based clusters is very beneficial.

A similar analysis can be carried out for the CloverLeaf application. Figure 39 shows that this application presents worse scalability features than the Hot application. As shown in the figure, not only the loss in speed up is larger but this loss increasingly grows with the number of cores used for executing the application. Thus, it is expected that for core counts larger than the ones shown in Figure 39, speed up is increasingly reduced. In any case, taking



(a) Projection over up to 128 Atom-based nodes.



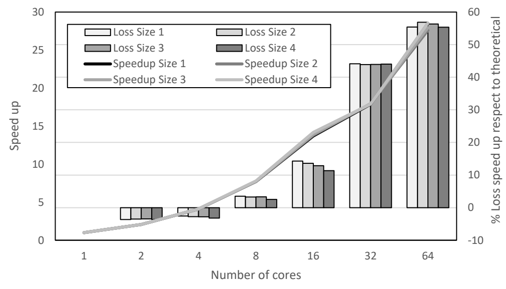
(b) Zoomed view of the projection.

Figure 40: Projection of the execution time of the CloverLeaf application over a large amount of Atom-based nodes. Comparison against the execution time using one remote V100 GPU.

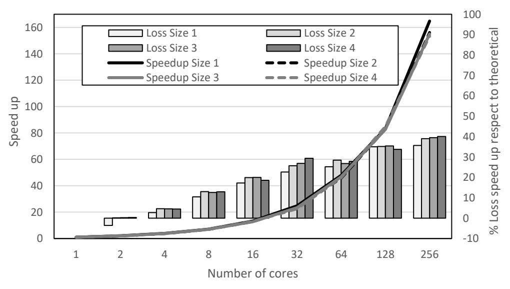
into account the loss in speed up obtained for the 64-Atom core executions in Figure 39, it is possible to estimate the performance of the CloverLeaf application. Notice that this would be an optimistic estimation, which actually represents a worst case for the comparison carried out in Figure 40. This figure shows that only when 128 Atom-based nodes are used, performance of CPU-based executions is better than that of GPU-based executions using a remote V100 GPU.

On the other hand, as in the case for previous applications, given the good scalability properties of the Flow application, it is possible to make estimates about the execution time of this application when a very large amount of Atom-based systems are used to execute it using only CPUs. Additionally, it is also possible to analyze the scalability trends of the Flow application as it has been done with the previous ones. Figure 41 shows how this application scales in performance, both for Atom-based systems as well as for Xeon-based deployments, when the dense core allocation policy is considered. As it was the case for the CloverLeaf application, performance loss is dependent on the number of cores used for executing the application. It can be clearly seen in Figure 41 that speed up is reduced as more and more cores are involved in the execution.

Assuming a constant performance loss equal to that shown in Figure 41(a) for 64-core executions, we can estimate the performance of the Flow application for very large system deployments. Notice that this would be an optimistic estimation because, as shown in Figure 41, it is expected that loss

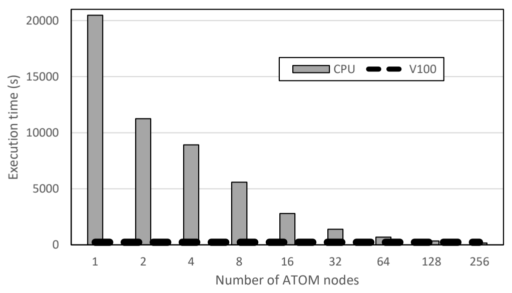


(a) Up to 64 Atom cores

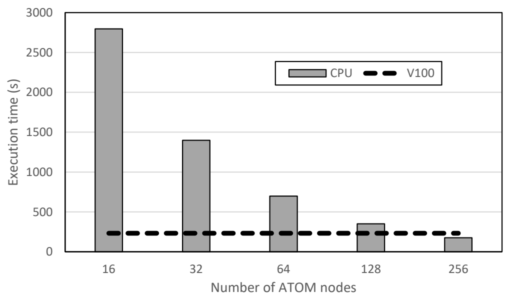


(b) Up to 256 Xeon cores

Figure 41: Speed up attained by the Flow application when scaled up to a large amount of cores. Different problem sizes are considered.



(a) Projection over up to 256 Atom-based nodes.

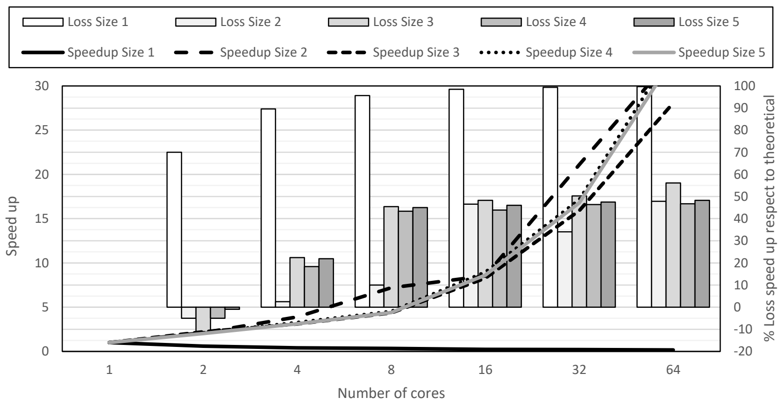


(b) Zoomed view of the projection.

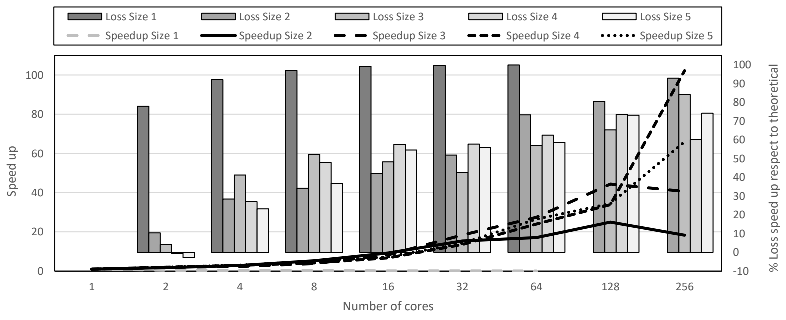
Figure 42: Projection of the execution time of the Flow application over a large amount of Atom-based nodes. Comparison against the execution time using one remote V100 GPU.

in performance would be larger beyond 64-core executions.

Figure 42 depicts such estimation for executions with up to 256 Atom-based nodes (up to 2048 Atom cores). It can be seen in Figure 42(b) that only when the Flow application is executed with 256 Atom nodes, execution time is smaller than when a single remote V100 GPU is used. These results are very important from the point of view of energy consumption given that, making a rough estimation, 256 Atom-based nodes consume at least 5120 Watts (considering only the power required by the CPU sockets, which is 20 Watts per node and excluding power consumption of main memory), whereas using a single remote V100 GPU requires about 530 Watts (250 Watts for the GPU, plus two Xeon sockets requiring 140 Watts each of them; as in the previous



(a) Up to 64 Atom cores

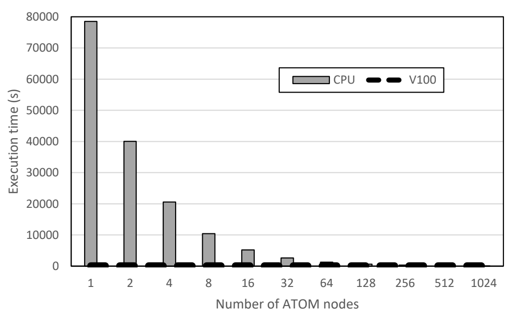


(b) Up to 256 Xeon cores

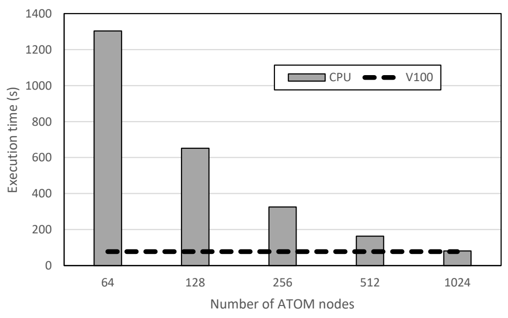
Figure 43: Speed up attained by the TeaLeaf application when scaled up to a large amount of cores. Different problem sizes are considered.

case, power required by main memory is excluded in this rough estimation). According to this simple estimation, using a remote V100 GPU reports a reduction in energy consumption of about 10x. Notice that we should add to these numbers the power required by other elements such as main memory. Considering also the power consumption of those other elements, it is clear that using remote GPUs is advantageous from an energy consumption point of view, although more accurate energy measurements should be carried out. This reasoning is very important because it demonstrates that enhancing low-power Atom-based deployments with rCUDA provides an extraordinary reduction in energy consumption.

Finally, a similar analysis can be carried out for the TeaLeaf application.



(a) Projection over up to 1024 Atom-based nodes



(b) Zoomed view of the projection

Figure 44: Projection of the execution time of the TeaLeaf application over a large amount of Atom-based nodes. Comparison against the execution time using one remote V100 GPU.

Figures 43 and 44 present, respectively, the scalability characteristics of this application and the performance estimations. It can be seen that using a single remote V100 GPU is a better option in terms of performance than using 1024 Atom-based nodes. This would report a 37x improvement in terms of energy consumption (according to the rough estimation made before).

5. Conclusions

Recent proposals for reducing the energy consumption of large HPC deployments consider the use of low-power processors instead of using the traditional power-hungry ones. However, these proposals do not consider the usage of GPUs, which in general are power-efficient while in use.

In this paper we have analyzed the usage of such accelerators by making use of the remote GPU virtualization technology. To that end we have analyzed the behavior of five applications belonging to the physics domain when executed in a cluster based on Atom processors. We have also considered the use of four generations of remote mainstream GPUs with rCUDA.

Results show that it is advantageous to use remote GPUs for improving the performance of the physics applications evaluated, even when the network fabric is a QDR InfiniBand providing 40 Gbps. In terms of energy consumption, a rough estimation shows that required energy can be reduced by an order of magnitude. As for future work, the use of better network technologies should be investigated as well as the benefits provided by multitenancy (several applications sharing the same remote GPU) [45] [46].

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.

References

[1] I. Corporation, Intel Xeon Processor E5 v4 Family Product Specification, `https://ark.intel.com/products/series/91287/Intel-Xeon-Processor-E5-v4-Family`, accessed 22 May 2019 (2017).

[2] N. R. et al., 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.

[3] GW4 Alliance, Isambard, `http://gw4.ac.uk/isambard/`, accessed 22 May 2019 (2011).

[4] 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.

[5] J. Maqbool, S. Oh, G. C. Fox, Evaluating ARM HPC clusters for scientific workloads, Concurrency and Computation: Practice and Experience 27 (17) (2015) 5390–5410.

[6] 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.

[7] Cavium, ThuderX ARM Processors: High Performance Workload Optimized Processors For Data Demanding Applications and Cloud Infrastructure, `https://www.marvell.com/server-processors/thunderx-arm-processors/`, accessed 22 May 2019 (2013).

[8] Cavium, ThuderX2 ARM Processors: High Performance ARMv8 Processors for Cloud and HPC Server Applications, `https://www.marvell.com/server-processors/thunderx2-arm-processors/`, accessed 22 May 2019 (2018).

[9] Cray, Cray XC50 Compute Blade for Arm Processors, https://www.cray.com/products/computing/xc-series, accessed 22 May 2019 (2017).

[10] 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 22 May 2019 (2017).

[11] GigaGyte, H270-T70 High Density Server, http://b2b.gigabyte.com/ARM-Server/H270-T70-rev-110\#ov, accessed 22 May 2019 (2017).

[12] Supermicro, Superserver 5039MA16-H12RFT, https://www.supermicro.nl/products/system/3U/5039/SYS-5039MA16-H12RFT.cfm, accessed 22 May 2019 (2018).

[13] W. Feng, T. Scogland, The GREEN500 List, https://www.top500.org/green500/, accessed 22 May 2019 (2017).

[14] E. Strohmaier, J. Dongarra, H. Simon, M. Meuer, The TOP500 List, https://www.top500.org/, accessed 22 May 2019 (2017).

[15] 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 22 May 2019 (2012).

[16] John Carey, Obama's Cap-and-Trade Plan, https://www.bloomberg.com/news/articles/2009-03-04/obamas-cap-and-trade-plan, accessed 22 May 2019 (2009).

[17] 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).

[18] 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 22 May 2019 (2016).

[19] 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

[20] 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 22 May 2019 (2011).

[21] S. Iserte, J. Prades, C. Reaño, F. Silla, Increasing the Performance of Data Centers by Combining Remote GPU Virtualization with Slurm, in: 16th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid), ACM, 2016, pp. 98–101. doi:10.1109/CCGrid.2016.26.

[22] 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.

[23] 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.

[24] C. Reaño, F. Silla, J. Duato, Enhancing the rCUDA Remote GPU Virtualization Framework: From a Prototype to a Production Solution, in: Proceedings of the 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, CCGrid '17, 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 22 May 2019 (2013).

[26] NVIDIA, TESLA K40 GPU ACCELERATOR Board Specification, http://www.nvidia.com/content/PDF/kepler/Tesla-K40-PCIe-Passive-Board-Spec-BD-06902-001_v05.pdf, accessed 22 May 2019 (2013).

[27] NVIDIA, NVIDIA Tesla P100 GPU ACCELERATOR, http://images.nvidia.com/content/tesla/pdf/nvidia-tesla-p100-PCIe-datasheet.pdf, accessed 22 May 2019 (2016).

[28] NVIDIA, NVIDIA Tesla V100 GPU ACCELERATOR, https://images.nvidia.com/content/technologies/volta/pdf/tesla-volta-v100-datasheet-letter-fnl-web.pdf, accessed 22 May 2019 (2017).

[29] F. Silla, J. Prades, C. Reaño, Leveraging rCUDA for Enhancing Low-Power Deployments in the Physics Domain, in: Proceedings of the 47th International Conference on Parallel Processing Companion, ICPP '18, 2018.

[30] NVIDIA, CUDA C Programming Guide. Design Guide, http://docs.nvidia.com/cuda/pdf/CUDA_C_Programming_Guide.pdf, accessed 22 May 2019 (2017).

[31] 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.

[32] 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.

[33] 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.

[34] 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.

[35] 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.

[36] NVIDIA, CUDA Runtime API. API Reference Manual, `http://docs.nvidia.com/cuda/pdf/CUDA_Runtime_API.pdf`, accessed 22 May 2019 (2016).

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

[38] C. Reaño, F. Silla, Tuning remote GPU virtualization for InfiniBand networks, The Journal of Supercomputing 72 (12) (2016) 4520–4545.

[39] C. Reaño, F. Silla, Extending rCUDA with Support for P2P Memory Copies between Remote GPUs, in: 2016 IEEE 18th International Conference on High Performance Computing and Communications; IEEE 14th International Conference on Smart City; IEEE 2nd International Conference on Data Science and Systems (HPCC/SmartCity/DSS), 2016.

[40] M. Martineau, S. McIntosh-Smith, Exploring on-node parallelism with neutral, a monte carlo neutral particle transport mini-app, in: 2017 IEEE International Conference on Cluster Computing (CLUSTER), 2017.

[41] M. Martineau, S. McIntosh-Smith, The arch project: Physics mini-apps for algorithmic exploration and evaluating programming environments on hpc architectures, in: 2017 IEEE International Conference on Cluster Computing (CLUSTER), 2017.

[42] J. A. Herdman, W. P. Gaudin, S. McIntosh-Smith, M. Boulton, D. A. Beckingsale, A. C. Mallinson, S. A. Jarvis, Accelerating hydrocodes with openacc, opencl and cuda, in: 2012 SC Companion: High Performance Computing, Networking Storage and Analysis, 2012.

[43] M. Martineau, S. McIntosh-Smith, M. Boulton, W. Gaudin, An evaluation of emerging many-core parallel programming models, in: Proceedings of the 7th International Workshop on Programming Models and Applications for Multicores and Manycores, PMAM'16, 2016.

[44] Network-Based Computing Laboratory, MVAPICH: MPI over InfiniBand, Omni-Path, Ethernet/iWARP, and RoCE, `http://mvapich.cse.ohio-state.edu/`, accessed 28 October 2019 (2019).

[45] J. Prades, B. Varghese, C. Reaño, F. Silla, Multi-tenant virtual GPUs for optimising performance of a financial risk application, Journal of Parallel and Distributed Computing 108 (2017) 28 – 44.

[46] J. Prades, C. Reaño, F. Silla, On the effect of using rCUDA to provide CUDA acceleration to Xen virtual machines, Cluster Computing 22 (1) (2019) 185–204.