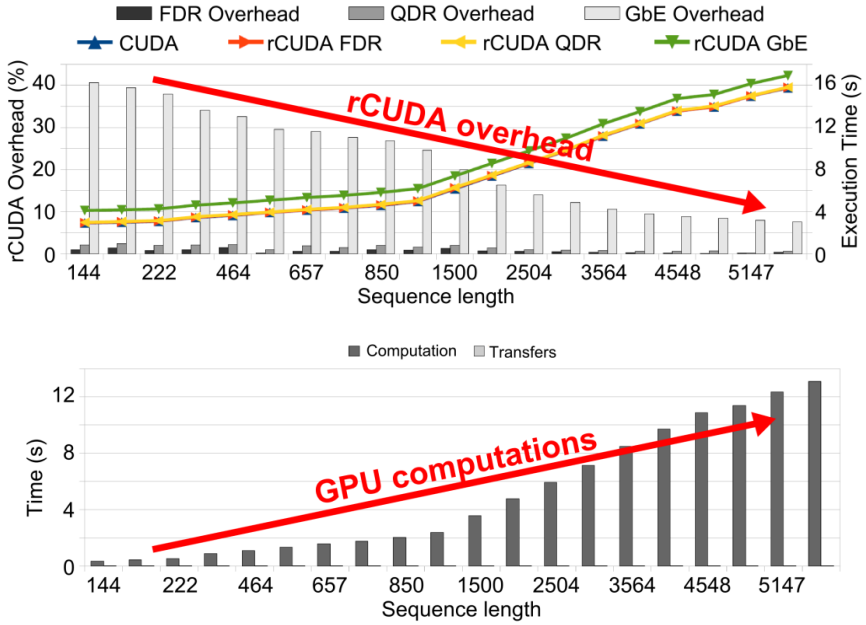
The rCUDA middleware and applications

Will my application work with rCUDA?

rCUDA currently provides binary compatibility with CUDA 5.0, virtualizing the entire Runtime API except for the graphics functions, which are rarely required in HPC scenarios. Regarding the network between the computer executing the application and the rCUDA server with the real GPU, rCUDA provides specific RDMA support for the InfiniBand fabric and also supports the general TCP/IP stack on any interconnect. Therefore, **as long as your application is CUDA 5.0 compliant and your platform is equipped with an InfiniBand network or the network provides TCP/IP compatibility, your application will work** even without having to recompile it.

With rCUDA, which performance should I expect for my application?

rCUDA makes use of remote GPUs instead of local ones. Therefore, some performance loss is expected due to the longer distance to the GPU. The exact reduction in performance depends on the application being run. Applications such as LAMMPS, WideLM, CUDASW++, HOOMDBlue, mCUDA-MEME, GPU-BLAST, and Gromacs have been tested with rCUDA. Visit www.rcuda.net for an updated list of applications. Below we report performance examples for some of them.



CUDASW++

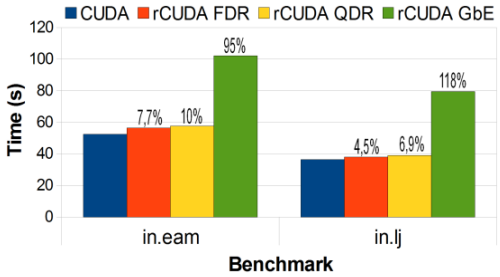
Execution time for queries of different sequence lengths, using CUDA and rCUDA over different networks. Primary Y-axis shows rCUDA's overhead. Secondary Y-axis depicts execution time.

The lower plot shows the time employed by computations (CUDA kernels) and by memory transfers (CUDA memcopy)

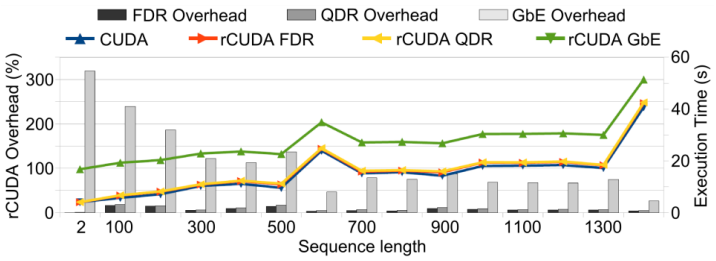
The conclusion is clear: **the more GPU computations, the less rCUDA overhead**

Performance for other applications

LAMMPS executing in.eam and in.lj, scaled by a factor of 5 in all three dimensions. Numbers over the bars are the overhead with respect to CUDA



GPU-BLAST. Execution time for queries of different sequence lengths, using different networks. Primary y-axis shows rCUDA's overhead. Secondary y-axis represents execution time



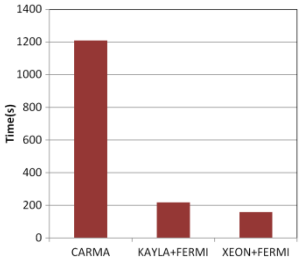
About rCUDA

In the context of HPC and datacenter clusters, the rCUDA framework grants CUDA-accelerated applications being executed in a server transparent access to GPUs installed in other server of the cluster. In this way, applications are not aware of being accessing an external device, as the rCUDA remote GPU virtualization framework hides all the details, while maintaining application performance.

rCUDA is available for the ARM architecture

Why an rCUDA version for the ARM processors?

Some chip and/or motherboard configurations for ARM and Atom include a small CUDA-compatible GPU where computations can be off-loaded to. However, these small accelerators provide noticeable lower performance than regular-sized ones, as shown in the figure on the right, that reports the execution time for the LAMMPS application on three different system configurations (GPUs are local). Notice that the main cause for the performance difference between the KAYLA+FERMI and the XEON+FERMI configurations is the lower bandwidth of the PCIe link in the former.



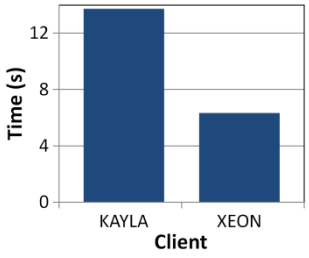
KAYLA: NVIDIA Tegra 3 ARM Cortex A9 quad-core CPU (1.4 GHz), 2GB DDR3 RAM and Intel 82574L Gigabit Ethernet controller
CARMA: NVIDIA Quadro 1000M GPU (96 cores) with 2GB DDR5 RAM. The rest of the system is the same as for KAYLA
XEON: Intel Xeon X3440 quad-core CPU (2.4GHz), 8GB DDR3 RAM and 82574L Gigabit Ethernet controller
FERMI: NVIDIA GeForce GTX480 "Fermi" GPU (448 cores) with 1,280 MB DDR3/GDDR5 RAM

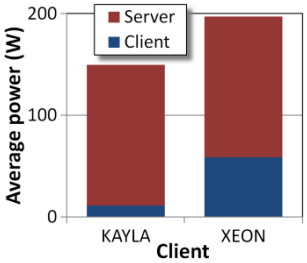
rCUDA allows to attach a "pool of GPUs" to a computing facility, sharing them among the nodes of the cluster. Therefore, with rCUDA you can use external GPUs from your ARM or Atom based cluster, thus mapping to the powerful off-of-the-box GPUs the computationally intensive parts of your application.

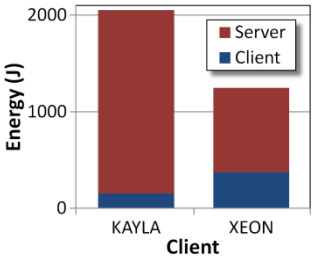
With rCUDA for ARM, which performance should I expect for my application?

rCUDA makes use of remote GPUs instead of local ones. Therefore, some performance loss is expected due to the longer distance to the GPU. The exact reduction in performance depends on the application being run and also on the bandwidth of the network connecting the computer running the application demanding GPGPU services and the computer owning the real GPU. We next illustrate some performance and power examples. The rCUDA server configuration is based on Xeon+Fermi.

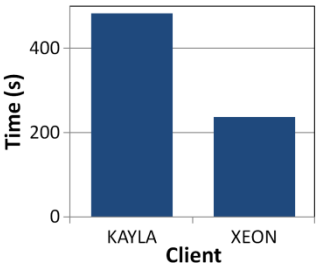
CUDASW++ application

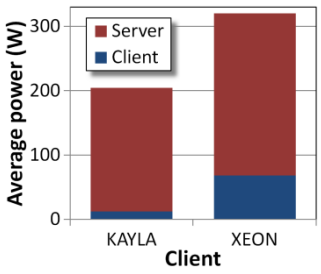


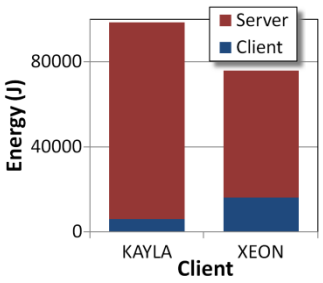




LAMMPS application

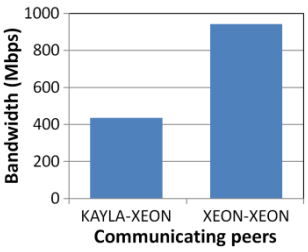






Conclusions on the performance and power numbers above

The numbers presented above should be taken as preliminary figures as they have been gathered with a low performance 1Gbps Ethernet network. Additionally, as depicted in the figure on the right, the adapter in the Kayla system was only able to deliver 435 Mbps whereas the adapter in the Xeon system delivered 943 Mbps. This is the main cause for the performance loss in the Kayla-based configuration above. We are confident that with better interconnects (InfiniBand, for example), the performance of rCUDA remotely accelerated ARM-based systems will be similar to that of Xeon based-systems while requiring noticeably less energy.



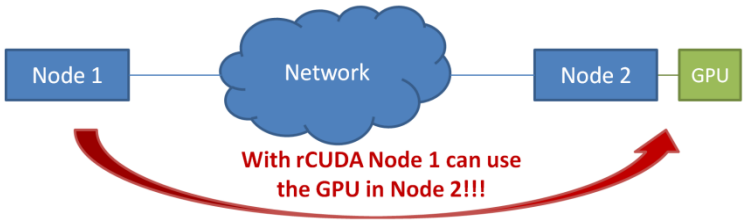
About rCUDA

In the context of HPC and datacenter clusters, the rCUDA framework grants CUDA-accelerated applications being executed in a server transparent access to GPUs installed in other server of the cluster. In this way, applications are not aware of being accessing an external device, as the rCUDA remote GPU virtualization framework hides all the details, while maintaining application performance.

rCUDA Frequently Asked Questions

What is rCUDA?

rCUDA (Remote CUDA) is a middleware that allows CUDA applications to use GPUs located in remote nodes **without any modifications** to original CUDA applications.



Are there any limitations?

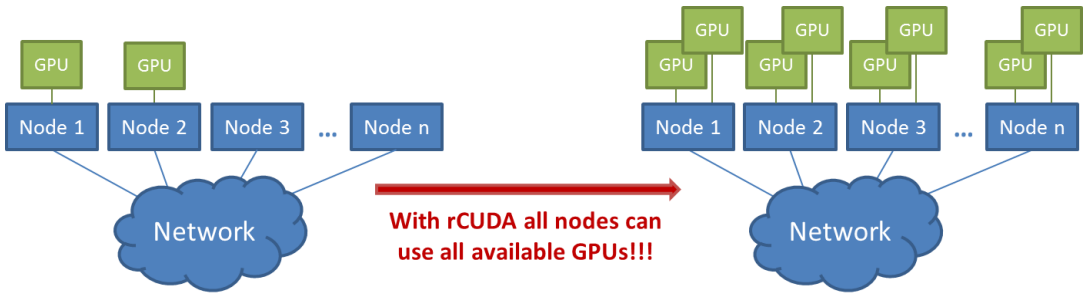
- Support for Linux OS (an alpha version for Windows is also available)
- Support for CUDA Runtime API and CUDA libraries (CUDA Driver API is not supported)
- Graphics interoperability is not supported (OpenGL, Direct3D...)
- Virtualized devices do not offer *zero copy* capabilities
- ARM rCUDA version cannot interact with 64-bit architectures

How can I start using rCUDA?

Just visit **www.rCUDA.net** and complete the software request form.

How many remote GPUs could my application use?

Your application will be able to use **all the GPUs available in the cluster**, as far as it is appropriately programmed to leverage more than one GPU. Briefly, if your regular CUDA application is able to make use of as many GPUs as available in the box, then with rCUDA it will be able to use all the GPUs in the cluster.



Scenario without rCUDA

Scenario with rCUDA

Which network fabrics are supported?

rCUDA has a modular, layered architecture which supports runtime-loadable, network-specific communication libraries. rCUDA currently provides communication modules for **Ethernet (TCP/IP)** and **InfiniBand verbs**.

Do I have to modify my application code in order to execute it with rCUDA?

No code modification is required in order to remotely accelerate your application with rCUDA. Actually, you do not even need to recompile your application, as rCUDA is based on the use of dynamic libraries that replace the CUDA ones and therefore your application code will be appropriately linked to the rCUDA code when started. Furthermore, the rCUDA library will automatically locate the rCUDA servers by using environment variables.

Is it possible to access a remote GPU installed in an x86 box from an ARM box?

Yes, it is possible. You can accelerate your ARM-based system by attaching to it a GPU box based on a regular x86 processor. The contrary is also possible. You can send your GPU kernels from an application being executed in an x86 box to a GPU installed in an ARM system.

What if I had more questions?

More information available at www.rCUDA.net or at info@rCUDA.net

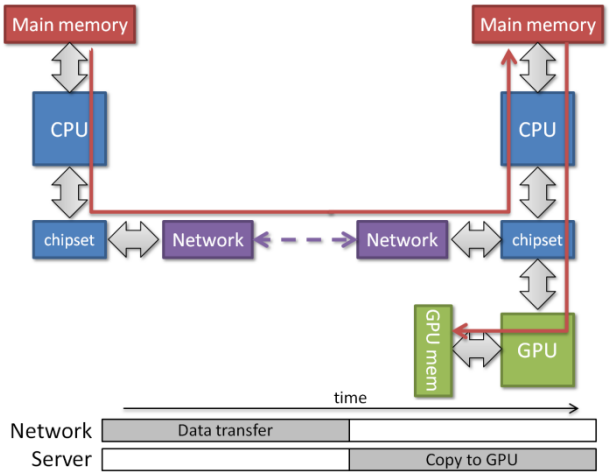
The evolution of rCUDA towards GPUDirect RDMA

rCUDA is designed according to a client-server architecture. The client side is the computer hosting the application requesting GPGPU services. The server side consists of the remote computer with the real GPU. In order to decouple the virtualization features from the communication characteristics, **rCUDA leverages a modular layered architecture**, where each layer evolves independently from each other. The communication between rCUDA clients and servers has evolved with time as shown below:

2010-2012

rCUDA versions 1, 2, and 3

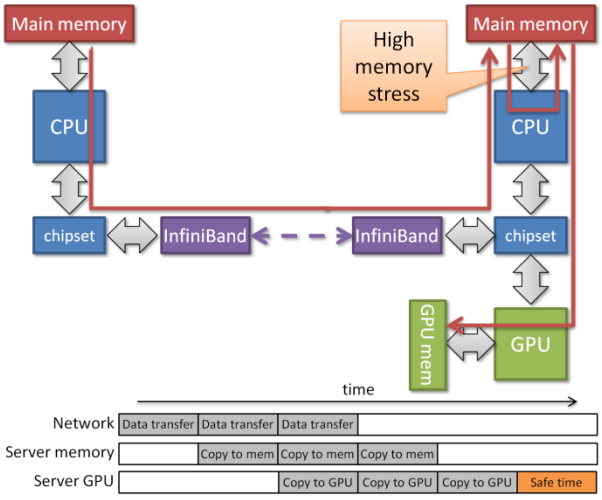
In the initial versions, the focus was on providing the CUDA virtualization services. Therefore, **communications were not elaborated**, just supporting only the TCP/IP protocol



2012

rCUDA version 4beta

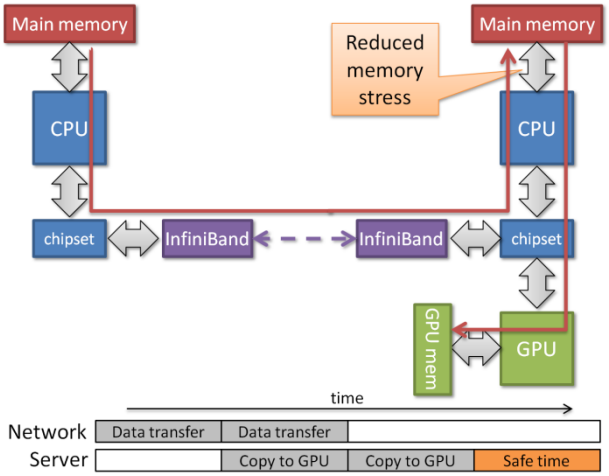
In the next version, **support for the InfiniBand fabric** was included. In addition, memory copies were improved by **pipelining the data transfers** among communicating peers



2012-2013

rCUDA version 4

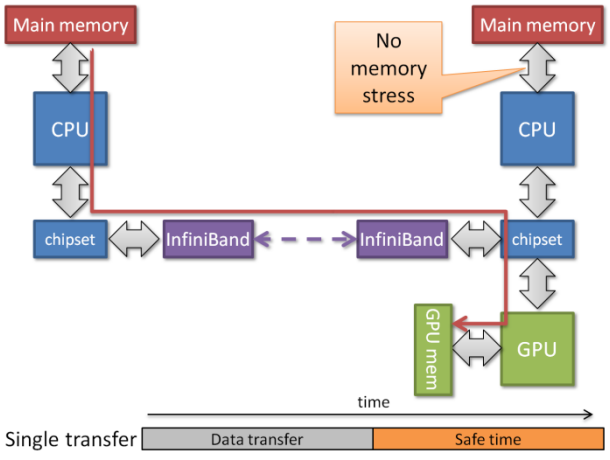
A further improvement in the communication approach enhanced the pipelined transfers with the use of **GPUDirect for the InfiniBand fabric**



2013

rCUDA version 5

rCUDA currently leverages **the new Mellanox OFED GPUDirect RDMA feature**. The new rCUDA version will be publicly **available at the end of Q4 2013**



About rCUDA

In the context of HPC and datacenter clusters, the rCUDA framework grants CUDA-accelerated applications being executed in a server transparent access to GPUs installed in other server of the cluster. In this way, applications are not aware of being accessing an external device, as the rCUDA remote GPU virtualization framework hides all the details, while maintaining application performance.

Extending SLURM with support for rCUDA

Why extending SLURM to support rCUDA?

Previous to this extension, **SLURM did not support the virtual GPUs** provided by rCUDA or any other remote GPU virtualization framework. That is, SLURM was only able to deal with real GPUs. Therefore, when a job included within its computing requirements one or more GPUs per node, SLURM tried to assign that job nodes owning the requested amount of GPUs, thus neglecting the benefits of remote GPU virtualization. By extending SLURM with support for rCUDA, it is possible to make it aware of the fact that now the assignment of GPUs is no longer constrained to the processes running in the same node they are located, but applications are able to make use of GPUs independently of the exact nodes where applications on one side and GPUs on the other are placed.

What does provide the integration of rCUDA with SLURM?

The new extension of SLURM allows:

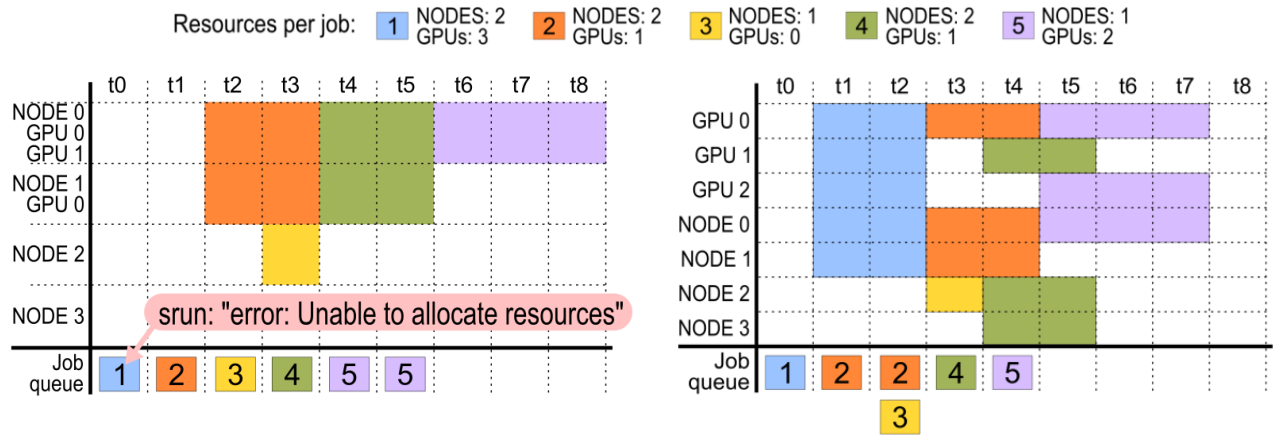
- Scheduling virtual GPUs by SLURM **in a user transparent way**
- Scheduling virtual GPUs either **as exclusive or shared resources**
- Virtual GPUs and standard physical GPUs can be used at the same time

What does scheduling virtual GPUs either as exclusive or shared resources mean?

rCUDA allows to share remote GPUs among several jobs. Therefore, SLURM inherits that feature and can schedule, according to the cluster administration directions, the use of GPUs as in the traditional way, that is, a **GPU is exclusively assigned to a process**, or in a shared way, where a given **GPU is granted to several processes**. The amount of processes that actually share a given GPU currently depends on their memory requirements and will also depend in the near future on the computing availability of the GPU, which will be dynamically monitored.

Cluster throughput is increased with rCUDA+SLURM

The combination of rCUDA+SLURM in your cluster increases its productivity, as shown in the example below, where five different jobs are submitted for execution and must be scheduled by SLURM. The resources requested by each of the jobs are displayed.



SLURM without rCUDA

In the scheduling example above we can see that **job number one cannot be scheduled** because no node in the cluster satisfies the requirements of this job. After that error, the rest of jobs are scheduled and executed. Notice that job five **cannot be immediately executed** after submission **despite of having the required resources available**.

SLURM with rCUDA

When SLURM+rCUDA is in use, GPUs are logically decoupled from nodes and therefore any GPU can be assign to any job, independently of their location. In this way overall cluster throughput is increased. In the example above **all the jobs are executed faster**

About rCUDA

In the context of HPC and datacenter clusters, the rCUDA framework grants CUDA-accelerated applications being executed in a server transparent access to GPUs installed in other server of the cluster. In this way, applications are not aware of being accessing an external device, as the rCUDA remote GPU virtualization framework hides all the details, while maintaining application performance.