rCUDA v5.0
User's Guide

November 7th, 2014

The rCUDA Team

www.rcuda.net **info@rcuda.net**


UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Department of Computer Engineering
Technical University of Valencia
Valencia, Spain

Dpto. Ing. y Ciencia de Computadores
University Jaume I
Castellón, Spain

Please cite the following papers in any published work if you use the rCUDA software:

- J. Duato, A.J. Peña, F. Silla, R. Mayo, E.S. Quintana-Orti, **"Performance of CUDA virtualized remote GPUs in high performance clusters"**, in proceedings of the 2011 International Conference on Parallel Processing, Taipei, Taiwan, September 2011.
- C. Reaño, R. Mayo, E.S. Quintana-Orti, F. Silla, J. Duato, A.J. Peña, **"Influence of InfiniBand FDR on the performance of remote GPU virtualization"**, in proceedings of the 2013 IEEE International Conference on Cluster Computing, Indianapolis, USA, October 2013.

Contents

Chapter 1

Introduction

The rCUDA framework enables the usage of remote CUDA-compatible devices. To enable a remote GPU-based acceleration, this framework creates virtual CUDA-compatible devices on those machines without a local GPU. These virtual devices represent physical GPUs located in a remote host offering GPGPU services. By leveraging the remote GPU virtualization technique, rCUDA allows to decouple CUDA accelerators from the nodes where they are installed, so that GPUs across the cluster can be seamlessly accessed from any node. Furthermore, nodes in the cluster can concurrently access remote GPUs. Figure 1.1 graphically depicts the additional flexibility provided by rCUDA.

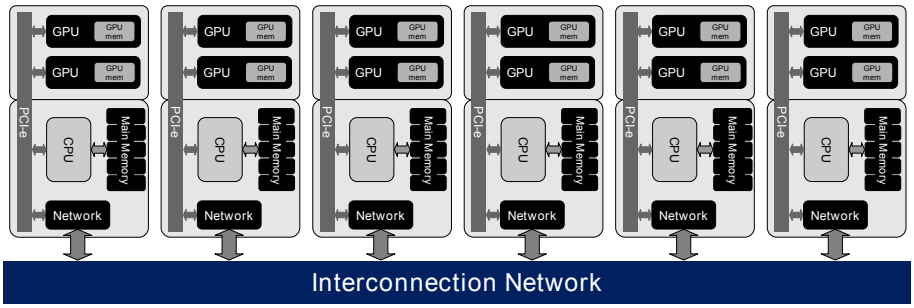
rCUDA can be useful in the following three different environments:

HPC Clusters and datacenters. In this context, rCUDA reduces the number of GPUs installed in the computing facility. This leads to an increased GPU utilization and to energy savings, as well as other related savings like acquisition costs, maintenance, space, cooling, etc.

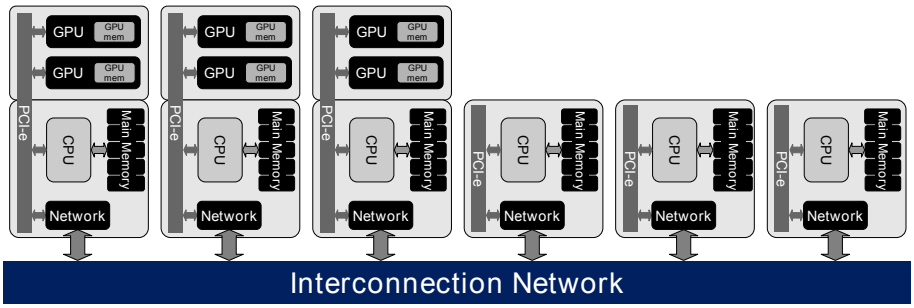
Academia. When using rCUDA in a teaching lab with a commodity network, our middleware offers concurrent access to a few high performance GPUs to several students, thus reducing the acquisition costs of the lab infrastructure.

Virtual Machines. In this scenario, rCUDA enables the access from the inside of the virtual machine to the CUDA accelerator(s) installed in the physical machine. In addition to allow accessing the accelerators installed in the real machine that hosts the virtual ones, it is also possible to access remote GPUs from the inside of virtual machines.

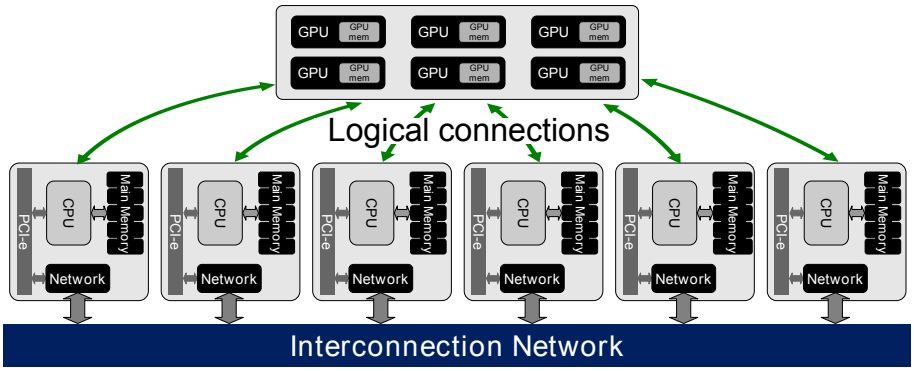
The current version of rCUDA (v5.0) implements all functions in the CUDA Runtime API and CUDA Driver API version 6.5, excluding those related with graphics interoperability, Unified Memory Management and Module management. It also implements all the functions in the cuFFT, cuRAND, cuBLAS (ex-



(a) When rCUDA is not deployed into the cluster, one or more GPUs must be installed in those nodes of the cluster intended for GPU computing. This usually leads to cluster configurations with one or more GPUs attached to all the nodes of the cluster. Nevertheless, GPU utilization may be lower than 100%, thus wasting hardware resources and delaying amortizing initial expenses.



(b) When rCUDA is leveraged in the cluster, only those GPUs actually needed to address overall workload must be installed in the cluster, thus reducing initial acquisition costs and overall energy consumption. rCUDA allows sharing the (reduced amount of) GPUs present in the cluster among all the nodes.



(c) From a logical point of view, the GPUs in the cluster can be seen as a pool of GPUs detached from the nodes and accessible through the cluster interconnect, in the same way as networked storage is shared among all the cluster nodes and concurrently accessed by them.

Figure 1.1: Different cluster configurations: (a) the traditional CUDA-based cluster deployment; (b) physical view of the cluster when leveraging rCUDA; (c) logical view of the cluster with rCUDA.

cluding complex) and cuFFT(excluding complex) 6.5 libraries. Other libraries provided by NVIDIA will be supported in future rCUDA releases. rCUDA 5.0 targets the Linux OS (for 32- and 64-bit x86-based configurations and 32-bit ARM architectures). It provides support for the same Linux distribution as CUDA does.

Chapter 2

Installation

The rCUDA installation is a two-step process if the package manager installation (RPM/DEB) is used or a one-step process if the tarball mode is used.

2.1 Package manager installation

2.1.1 RPM

RPM distribution package is used on Scientific Linux (CentOs), Fedora and OpenSUSE linux distributions. In order to install rCUDA in your system using the rpm package, please complete the steps showed bellow:

1. sudo rpm –Uvh –nodeps rCUDA-<version>.<distro>.<architecture>.rpm
2. rCUDA-install

And then, please follow the instructions showed.

If you want to remove the rCUDA software from your system, rCUDA-uninstall should be executed.

2.1.2 DEB

DEB distribution package is used on Ubuntu linux distributions. In order to install rCUDA in your system using the deb package, please complete the steps showed bellow:

1. sudo dpkg -i rCUDA-<version>.<distro>.<architecture>.deb
2. rCUDA-install

At that point, please follow the instructions showed.

If you want to remove the rCUDA software from your system, rCUDA-uninstall should be executed.

2.2 Tarball installation

In this case, the binaries are distributed whithin a tarball which has to be decompressed manually by the user. The steps to install rCUDA binaries are:

1. Uncompress the rCUDA package.
2. Copy the rCUDAl folder to the client(s) node(s) (without GPU) as it is explained in Section 3.1.
3. Copy the rCUDAd folder to the server node (with GPU) as it is explained in Section 3.2.

Chapter 3

Components and usage

rCUDA is organized following a client-server distributed architecture, as shown in Figure 3.1. The client middleware is contacted by the application demanding GPGPU services, both running in the same cluster node. The rCUDA client presents to the application the very same interface as the regular NVIDIA CUDA Runtime API. Upon reception of a request from the application, the client middleware processes it and forwards the corresponding requests to the rCUDA server middleware. In turn, the server interprets the requests and performs the required processing by accessing the real GPU to execute the corresponding request. Once the GPU has completed the execution of the requested command, the results are gathered by the rCUDA server, which sends them back to the client middleware. There, the output is finally forwarded to the demanding application.

In order to optimize client/server data exchange, rCUDA employs a customized application-level communication protocol. Furthermore, rCUDA provides efficient support for several underlying network technologies. To do so, rCUDA supports runtime-loadable specific communication modules that currently target the InfiniBand network (using InfiniBand verbs) and the general TCP/IP protocol stack (see Figure 3.1). Additional network technologies may be supported in the future.

3.1 Client Side

The client side of the rCUDA middleware is a library of wrappers that replaces the CUDA Runtime (provided by NVIDIA as a dynamic library). In this way, CUDA applications that use rCUDA are not aware of being accessing an external device. Also, they do not need any source code modification.

The rCUDA client is distributed in a set of files: “libcudart.so.6.5, libcuda.so,

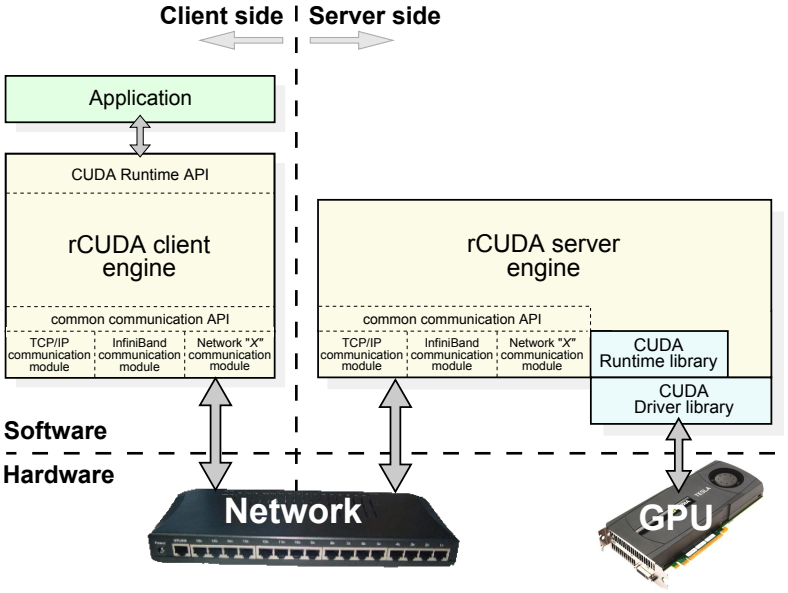


Figure 3.1: rCUDA architecture, showing also the runtime-loadable specific communication modules.

libcublas.so.6.5, libcufft.so.6.5, libcusparse.so.6.5 and libcurand.so.6.5". These shared libraries should be placed in those machines accessing remote GPGPU services. Set the LD_LIBRARY_PATH environment variable according to the final location of these files (typically "`$HOME/rCUDA/framework/rCUDA1`" or "`/usr/local/cuda/lib64`").

In order to properly execute the applications using the rCUDA library, set the following environment variables:

- RCUDA_DEVICE_COUNT: indicates the number of GPUs which are accessible from the current node.
  Usage: "RCUDA_DEVICE_COUNT=<number_of_GPUs>"
  For example, if the current node can access two GPUs:
  "RCUDA_DEVICE_COUNT=2"
- RCUDA_DEVICE_X: indicates where GPU X, for the client being configured, is located.
  Usage: "RCUDA_DEVICE_X=<server[@<port>]>[:GPUnumber]"
  For example, if GPUs 0 and 1 assigned to the current client are located at server "192.168.0.1" (default rCUDA port):
  "RCUDA_DEVICE_0=192.168.0.1"
  "RCUDA_DEVICE_1=192.168.0.1:1"

Furthermore, as nvcc compiler links with CUDA static libraries by default, a compilation using CUDA dynamic libraries is needed to allow the use of rCUDA software. This step can be done by the user in three different ways:

- NVIDIA_CUDA_SDK-6.5 must be compiled after the EXTRA_NVCCFLAGS environment variable has been set to --cudart=shared.
- If nvcc compiler is used, the flag -cudart=shared is needed.
- If gcc/++ compiler is used, the -lcudart flag is needed.

In supported MPI environments (MVAPICH2 2.0b and OpenMPI 1.6.5 and newer versions), the library will distribute the MPI tasks among the different servers. The default port is 8308. If an InfiniBand network connection is used, set the "RCUDAPROTO" environment variable to IB[1].

3.2 Server Side

The rCUDA server is configured as a daemon (rCUDAd) that runs in those nodes offering GPGPU acceleration services.

Set the LD_LIBRARY_PATH environment variable according to the location of the CUDA libraries (typically "`/usr/local/cuda/lib64`"). Notice that this is the path to the original CUDA libraries, not the rCUDA ones.

Set the "RCUDAPROTO" environment variable to IB if an InfiniBand network connection is used[1].

This daemon offers the following command-line options:

-i : Do not daemonize. Instead, run in interactive mode.

-l : Local mode using AF_UNIX sockets (TCP only).

-n <**number**> : Number of concurrent servers allowed. 0 stands for unlimited (default).

-p <**port**> : Specify the port to listen to (default: 8308).

-v Verbose mode.

-h Print usage information.

[1] Not supported in the ARM distribution

Chapter 4

Current limitations

The current implementation of rCUDA features the following limitations:

- Graphics interoperability is not implemented. Missing modules: OpenGL Interoperability, Direct3D 9 Interoperability, Direct3D 10 Interoperability, Direct3D 11 Interoperability, VDPAU Interoperability, Graphics Interoperability.
- Targets the Linux OS (32- and 64-bit architectures) on both client and server sides, but these have to match.
- ARM rCUDA version cannot interact with 64-bit architectures. Nevertheless, the ARM version can interact with the x86 32-bit version.
- Virtualized devices do not offer *zero copy* capabilities.
- Peer-to-Peer communication between 2 GPUs in different nodes is not supported.
- Unified Memory Management is not supported.
- Module management is not supported
- Timing with the event management functions might be inaccurate, since these timings will discard network delays. Using standard Posix timing procedures such as "*clock_gettime*" is recommended.

Chapter 5

Further Information

For further information, please refer to [1, 2, 3, 4, 5, 6, 7, 8, 9]. Also, do not hesitate to contact support@rcuda.net for any questions or bug reports.

Chapter 6

Credits

6.1 Supervision

José Duato and Federico Silla
Grupo de Arquitecturas Paralelas
Departamento de Informática de Sistemas y Computadores
Universitat Politècnica de València
Camino de Vera, s/n
46022 – Valencia, Spain
Email: {jduato, fsilla}@disca.upv.es

Rafael Mayo and Enrique S. Quintana-Ortí
High Performance Computing and Architectures Group
Departamento de Ingeniería y Ciencia de los Computadores
Universidad Jaume I
Av. Vicente Sos Baynat, s/n
12071 – Castellón, Spain
Email: {mayo, quintana}@icc.uji.es

6.2 Development

6.2.1 Initial Developer

Antonio J. Peña
Grupo de Arquitecturas Paralelas
Departamento de Informática de Sistemas y Computadores
Universitat Politècnica de València
Camino de Vera, s/n
46022 – Valencia, Spain
Email: apenya@gap.upv.es

6.2.2 Current Developers

Carlos Reaño
Grupo de Arquitecturas Paralelas
Departamento de Informática de Sistemas y Computadores
Universitat Politècnica de València
Camino de Vera, s/n
46022 – Valencia, Spain
Email: carregon@gap.upv.es

Adrián Castelló
High Performance Computing and Architectures Group
Departamento de Ingeniería y Ciencia de los Computadores
Universidad Jaume I
Av. Vicente Sos Baynat, s/n
12071 – Castellón, Spain
Email: adcastel@icc.uji.es

Acknowledgements

This work was supported by PROMETEO program from Generalitat Valenciana (GVA) under Grant PROMETEO/2008/060 and also by PROMETEO program phase II under Grant PROMETEOII/2013/009. It was also supported by the Spanish Ministry of Science and FEDER (contract no. TIN2011-23283), and by Fundación Caixa-Castelló/Bancaixa (contract no. P1-1B2009-35).

Bibliography

[1] José Duato, Francisco D. Igual, Rafael Mayo, Antonio J. Peña, Enrique S. Quintana-Ortí, and Federico Silla. An efficient implementation of GPU virtualization in high performance clusters. In *Euro-Par 2009, Parallel Processing – Workshops*, volume 6043 of *Lecture Notes in Computer Science*, pages 385–394. Springer-Verlag, 2010.

[2] José Duato, Antonio J. Peña, Federico Silla, Rafael Mayo, and Enrique S. Quintana-Ortí. rCUDA: reducing the number of GPU-based accelerators in high performance clusters. In *Proceedings of the 2010 International Conference on High Performance Computing and Simulation (HPCS 2010)*, pages 224–231, Caen, France, June 2010.

[3] José Duato, Antonio J. Peña, Federico Silla, Rafael Mayo, and Enrique S. Quintana-Ortí. Performance of cuda virtualized remote gpus in high performance clusters. In *Proceedings of the 2011 International Conference on Parallel Processing (ICPP 2011)*, Taipei, Taiwan, September 2011.

[4] José Duato, Antonio J. Peña, Federico Silla, Juan C. Fernández, Rafael Mayo, and Enrique S. Quintana-Ortí. Enabling CUDA acceleration within virtual machines using rCUDA. In *Proceedings of the 2011 International Conference on High Performance Computing (HiPC 2011)*, Bangalore, India, December 2011.

[5] Carlos Reaño, Antonio J. Peña, Federico Silla, Rafael Mayo, Enrique S. Quintana-Ortí, and José Duato. CU2rCU: towards the complete rCUDA remote GPU virtualization and sharing solution. In *Proceedings of the 2012 International Conference on High Performance Computing (HiPC 2012)*, Pune, India, June 2012.

[6] Carlos Reaño, Antonio J. Peña, Federico Silla, Rafael Mayo, Enrique S. Quintana-Ortí, and José Duato. Influence of infiniband FDR on the performance of remote GPU virtualization. In *Proceedings of the IEEE International Conference on Cluster Computing (CLUSTER 2013)*, Indianapolis, USA, October 2013.

[7] Adrián Castelló, José Duato, Rafael Mayo, Antonio J. Peña, Enrique S. Quintana-Ortí, Vicente Roca, and Federico Silla. On the Use of Remote GPUs and Low-Power Processors for the Acceleration of Scientific Applications. In *The Fourth International Conference on Smart Grids, Green Com-*

munications and IT Energy-aware Technologies, Chamonix, France, April 2014.

[8] Carlos Reaño, Federico Silla, Antonio J. Peña, Gilad Shainer, Scot Schultz, Adrián Castelló, Enrique S. Quintana-Ortí, and José Duato. Boosting the Performance of Remote GPU Virtualization Using InfiniBand Connect-IB and PCIe 3.0. In *Proceedings of the IEEE International Conference on Cluster Computing (CLUSTER 2014)*, Madrid, Spain, September 2014.

[9] Sergio Iserte and Adrián Castelló and Rafael Mayo and Enrique S. Quintana-Ortí and Carlos Reaño and Javier Prades and Federico Silla and José Duato. SLURM Support for Remote GPU Virtualization: Implementation and Performance Study. In *Proceedings of the International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD 2014)*, Paris, France, October 2014.