rCUDA User's Guide
v4.0.1

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

February 22, 2013

Contents

Chapter 1

Introduction

The rCUDA framework enables the concurrent usage of CUDA-compatible devices remotely. 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.

rCUDA can be useful in three different environments:

Clusters. To reduce the number of GPUs installed in High Performance Clusters. This leads to increase GPUs use and to energy savings, as well as other related savings like acquisition costs, maintenance, space, cooling, etc.

Academia. In commodity networks, to offer access to a few high performance GPUs concurrently to several students.

Virtual Machines. To enable the access to the CUDA facilities on the physical machine.

The current version of rCUDA (v4.0) implements all functions in the CUDA Runtime API version 5.0, excluding those related with graphics interoperability. rCUDA 4.0 targets the Linux OS (for 32- and 64-bit architectures) on both client and server sides.

Chapter 2

Components and usage

This framework is composed of a client middleware, which is a library of wrappers that replaces the CUDA Runtime (provided by NVIDIA as a dynamic library), and a server middleware, configured as a daemon which runs in those nodes offering GPGPU acceleration services.

rCUDA is organized as a client-server distributed architecture. Clients use an interception library of the CUDA Runtime API to access virtualized devices, while nodes hosting the physical accelerators run a daemon servicing API execution requests.

In order to optimize client/server data exchange, rCUDA employs a customized application-level communication protocol.

2.1 Client Side

The client side middleware is distributed in a file: "libcudart.so.5.0". This shared libraries should be placed in that machine(s) accessing remote GPGPU services. Set the LD_LIBRARY_PATH environment variable according to the final location of these files (typically "`$HOME/rCUDA/framework/rCUDAl`" 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 to two GPUs:
  "RCUDA_DEVICE_COUNT=2"

- RCUDA_DEVICE_X: indicates where is located the GPU X for the current node.
  Usage: "RCUDA_DEVICE_X=<server[@<port>]>[:GPUnumber]"
  For example, if the GPUs 0 and 1 of the current node 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"

In supported MPI environments (MVAPICH2 1.8 and OpenMPI 1.3+), 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" enviorment variable to IB.

2.2 Server Side

Set the "RCUDAPROTO" enviorment variable to IB if an InfiniBand network connection is used.

The rCUDA daemon (rCUDAd) should be run in that machine(s) offering remote GPGPU services.

This daemon offers the following command-line options:

-d <device> : Select device (first working device by default).

-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.

Chapter 3

Current limitations

The current implementation of rCUDA features the next 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.
- Virtualized devices do not offer *zero copy* capabilities.
- As the CUDA APIs do not explicitly provide a method to find and use embedded device code, rCUDA does not support this feature.
- 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 4

Further Information

For further information, please refer to [1, 2, 3, 4]. Also, do not hesitate to contact Antonio J. Peña (apenya@gap.upv.es) for any questions or bug reports (see the next chapter).

Chapter 5

Credits

5.1 Supervision

José Duato and Federico Silla
Grupo de Arquitecturas Paralelas
Departamento de Informática de Sistemas y Computadores
Universitat Politécnica de Valéncia
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

5.2 Development

5.2.1 Original Developer. Current Architect and Development Coordinator.

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

5.2.2 Developers

Carlos Reaño
Grupo de Arquitecturas Paralelas
Departamento de Informática de Sistemas y Computadores
Universitat Politécnica de Valéncia
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 from Generalitat Valenciana (GVA) under Grant PROMETEO/2008/060, by the Spanish Ministry of Science and Innovation under Grant CONSOLIDER INGENIO CSD2006-00046, by the Spanish Ministry of Science and FEDER (contract no. TIN2011-23283), and by the 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.