On the use of remote GPU virtualization for managing the GPUs of your cluster in a flexible way

Federico Silla & Carlos Reaño
Technical University of Valencia
Spain
Tutorial Schedule

- 14.30
  - Basics of remote GPU virtualization
  - Hands on rCUDA
- 16.00
  - Coffee Break
- 16.30
  - rCUDA lab
  - Advanced features of rCUDA
Basics of remote GPU virtualization
On the use of remote GPU virtualization for managing the GPUs of your cluster in a flexible way

Federico Silla
Technical University of Valencia
Spain
Outline

- Why remote GPU virtualization?
- How does rCUDA work?
- The performance of the rCUDA framework
Why remote GPU virtualization?

How does rCUDA work?

The performance of the rCUDA framework
Current computing needs

- Many applications require a lot of computing resources
- Execution time is usually increased
- Applications are accelerated to get their execution time reduced
- GPU computing has experienced a remarkable growth in the last years
GPUs reduce energy and time

blastp –db sorted_env_nr –query SequenceLength_00001300.txt -num_threads X -gpu [t|f]

Dual socket E5-2620 v2 Intel Xeon node with NVIDIA K20 GPU

GPU-Blast: Accelerated version of the NCBI-BLAST (Basic Local Alignment Search Tool), a widely used bioinformatics tool
GPUs reduce energy and time

```
blastp -db sorted_env_nr -query SequenceLength_00001300.txt -num_threads X -gpu [t|f]
```

Dual socket E5-2620 v2 Intel Xeon node with NVIDIA K20 GPU

Green zone: GPUs are **better** than CPUs

Red zone: GPUs are **worse** than CPUs
Current GPU computing facilities

The basic building block is a node with one or more GPUs
GPU-enabled clusters

- From the programming point of view:
  - A set of nodes, each one with:
    - one or more CPUs (with several cores per CPU)
    - one or more GPUs (typically between 1 and 4)
  - An interconnection network
A GPU computing facility is usually a set of independent self-contained nodes that leverage the shared-nothing approach:

- Nothing is directly shared among nodes (MPI required for aggregating computing resources within the cluster)
- GPUs can only be used within the node they are attached to
First concern with accelerated clusters

• Applications can only use the GPUs located within their node:
  • Non-accelerated applications keep GPUs idle in the nodes where they use all the cores

A CPU-only application spreading over these four nodes would make their GPUs unavailable for accelerated applications
Money leakage in current clusters?

For many workloads, GPUs may be idle for significant periods of time:

- Initial acquisition costs not amortized
- Space: GPUs reduce CPU density
- Energy: idle GPUs keep consuming power

![Image of GPU nodes and power consumption](image)

- 1 GPU node: 2 E5-2620V2 sockets and 32GB DDR3 RAM. Tesla K20 GPU
- 4 GPUs node: 2 E5-2620V2 sockets and 128GB DDR3 RAM. 4 Tesla K20 GPUs
Second concern with accelerated clusters

- Applications can only use the GPUs located within their node:
  - Multi-GPU applications running on a subset of nodes cannot make use of the tremendous GPU resources available at other cluster nodes (even if they are idle)

All these GPUs cannot be used by the MPI multi-GPU application in execution
One more concern with accelerated clusters

- Do applications **completely squeeze** GPUs present in the cluster?
  - Even if all GPUs are assigned to running applications, computational resources inside GPUs may not be fully used
    - Application presenting low level of parallelism
    - CPU code being executed
    - GPU-core stall due to lack of data
    - etc …
In summary …

- There are scenarios where GPUs are available but cannot be used
- Accelerated applications do not make use of GPUs 100% of the time

In conclusion …

- We are losing GPU cycles, thus reducing cluster performance
We need something more in the cluster

What is missing is ...

... some flexibility for using the GPUs in the cluster
A way of addressing the “idle” concern is by sharing the GPUs present in the cluster among all the nodes.
Once GPUs are shared, their **amount** can be **reduced**.

This would increase GPU utilization, also lowering power consumption, at the same time that initial acquisition costs are reduced.
What is needed for increased flexibility?

- This new cluster configuration requires:
  - A way of seamlessly sharing GPUs across nodes in the cluster (remote GPU virtualization)
  - Enhanced job schedulers that take into account the new virtual GPUs
Remote GPU virtualization envision

- Remote GPU virtualization allows a new vision of a GPU deployment, moving from the usual cluster configuration:

![Diagram of GPU virtualization]

Interconnection Network

to the following one ....
Remote GPU virtualization envision

Physical configuration

Interconnection Network

Logical configuration

Logical connections

Interconnection Network
Busy cores are no longer a problem

Physical configuration

Interconnection Network

Logical configuration

Logical connections

Interconnection Network
Multi-GPU applications get benefit

- GPU virtualization is also useful for multi-GPU applications

Without GPU virtualization:
- Only the GPUs in the node can be provided to the application

With GPU virtualization:
- Many GPUs in the cluster can be provided to the application
Remote GPU virtualization envision

Without GPU virtualization

With GPU virtualization

Virtualized remote GPUs

Interconnection Network

GPU virtualization allows all nodes to access all GPUs
More about reducing energy consumption

- One step further:
  - enhancing the scheduling process so that GPU servers are put into low-power sleeping modes as soon as their acceleration features are not required.
Going even beyond:
- consolidating GPUs into **dedicated GPU boxes** (with a low-power CPU)
- allowing GPU task migration

**TRUE GREEN GPU COMPUTING**
- Box A has 4 GPUs but only one is busy
- Box B has 8 GPUs but only two are busy
- Move jobs from Box B to Box A and switch off Box B
- Migration should be transparent to applications (decided by the global scheduler)
- Box A has 4 GPUs and only one is busy.
- Box B has 8 GPUs but only two are busy.
- Move jobs from Box B to Box A and switch off Box B.
- Migration should be transparent to applications (decided by the global scheduler).

TRUE GREEN GPU COMPUTING
Main GPU virtualization drawback is the increased latency and reduced bandwidth to the remote GPU.

Influence of data transfers for SGEMM

- Pinned Memory
- Non-Pinned Memory

Data from a matrix-matrix multiplication using a local GPU!!!
Several efforts have been made regarding GPU virtualization during the last years:

<table>
<thead>
<tr>
<th>Framework</th>
<th>CUDA Version</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>rCUDA</strong></td>
<td><strong>CUDA 6.0</strong></td>
</tr>
<tr>
<td><strong>GVirtuS</strong></td>
<td><strong>CUDA 3.2</strong></td>
</tr>
<tr>
<td><strong>DS-CUDA</strong></td>
<td><strong>CUDA 4.1</strong></td>
</tr>
<tr>
<td><strong>vCUDA</strong></td>
<td><strong>CUDA 1.1</strong></td>
</tr>
<tr>
<td><strong>GViM</strong></td>
<td><strong>CUDA 1.1</strong></td>
</tr>
<tr>
<td><strong>GridCUDA</strong></td>
<td><strong>CUDA 2.3</strong></td>
</tr>
<tr>
<td><strong>V-GPU</strong></td>
<td><strong>CUDA 4.0</strong></td>
</tr>
</tbody>
</table>

- Publicly available
- NOT publicly available
Performance comparison of GPU virtualization solutions:

- Intel Xeon E5-2620 (6 cores) 2.0 GHz (SandyBridge architecture)
- GPU NVIDIA Tesla K20
- Mellanox ConnectX-3 single-port InfiniBand Adapter (FDR)
- SX6025 Mellanox switch
- CentOS 6.3 + Mellanox OFED 1.5.3

Latency microseconds (measured by transferring 64 bytes)

<table>
<thead>
<tr>
<th></th>
<th>Pageable H2D</th>
<th>Pinned H2D</th>
<th>Pageable D2H</th>
<th>Pinned D2H</th>
</tr>
</thead>
<tbody>
<tr>
<td>CUDA</td>
<td>34,3</td>
<td>4,3</td>
<td>16,2</td>
<td>5,2</td>
</tr>
<tr>
<td>rCUDA</td>
<td>94,5</td>
<td>23,1</td>
<td>292,2</td>
<td>6,0</td>
</tr>
<tr>
<td>GVirtuS</td>
<td>184,2</td>
<td>200,3</td>
<td>168,4</td>
<td>182,8</td>
</tr>
<tr>
<td>DS-CUDA</td>
<td>45,9</td>
<td>-</td>
<td>26,5</td>
<td>-</td>
</tr>
</tbody>
</table>
Remote GPU virtualization frameworks

- Bandwidth of a copy between GPU and CPU memories

![Graphs showing bandwidth comparisons between different GPU virtualization frameworks.](image-url)
Applications tested with rCUDA

- rCUDA has been successfully tested with several applications:
  - NVIDIA CUDA SDK Samples
  - LAMMPS
  - WideLM
  - CUDASW++
  - OpenFOAM
  - HOOMDBlue
  - mCUDA-MEME
  - GPU-Blast
  - Gromacs
  - GAMESS
  - DL-POLY
  - HPL

- The list keeps growing
Why remote GPU virtualization?

How does rCUDA work?

The performance of the rCUDA framework
A framework enabling a CUDA-based application running in one (or some) node(s) to access GPUs in other nodes

It is useful for:

- Applications that do not make use of GPUs all the time (moderate level of data parallelism)
- Applications for multi-GPU computing
Basics of the rCUDA framework

Basic CUDA behavior

Application

CUDA libraries

GPU
Basics of the rCUDA framework

Client side | Server side

Application

CUDA Runtime API

rCUDA client

rCUDA server

Software

Hardware

Network

CUDA libraries

GUP
Basics of the rCUDA framework
Basics of the rCUDA framework
Basics of the rCUDA framework

Client side

Application

CUDA Runtime API

rCUDA client

Software

Hardware

Server side

rCUDA server

CUDA libraries

Network

GPU
Environment variables are properly initialized in the client side and used by the rCUDA client (transparently to the application).
rCUDA uses a proprietary communication protocol

Example:

1) initialization
2) memory allocation on the remote GPU
3) CPU to GPU memory transfer of the input data
4) kernel execution
5) GPU to CPU memory transfer of the results
6) GPU memory release
7) communication channel closing and server process finalization
rCUDA presents a modular architecture

[Diagram showing a modular architecture withClient side and Server side separated by a dashed line. The Client side includes Application, rCUDA Runtime API, and rCUDA client engine with modules for common communication API, TCP/IP module, InfiniBand module, and Network "X" module. The Server side includes rCUDA server engine with the same modules, CUDA Runtime library, CUDA Driver library, and a GPU. A network connection is shown between the client and server.]
rCUDA uses optimized transfers

- rCUDA features **optimized data transfers:**
  - Use of GPUDirect RDMA to move data between GPUs
  - Pipelined transfers to improve performance
  - Preallocated pinned memory buffers
  - Optimal pipeline block size

![Diagram of data transfer stages](image)
Basic performance analysis

- Pipeline block size for InfiniBand Connect-IB

- NVIDIA Tesla K40; Mellanox Connect-IB + SX6036 Mellanox switch

It was 2MB with IB QDR and 1MB with FDR X3
Basic performance analysis

- Bandwidth of a copy between GPU and CPU memories

![Graph showing Bandwidth vs Copy Size for different memory configurations]
Outline

- Why remote GPU virtualization?
- How does rCUDA work?
- The performance of the rCUDA framework
Performance of the rCUDA framework

- Test system:
  - Intel Xeon E5-2620v2 (6 cores) 2.1 GHz (Ivy Bridge architecture)
  - GPU NVIDIA Tesla K20
  - Mellanox ConnectX-3 single-port InfiniBand Adapter (FDR)
  - SX6025 Mellanox switch
  - Cisco switch SLM2014 (1Gbps Ethernet)
  - CentOS 6.3 + Mellanox OFED 1.5.3

- Correctness of NVIDIA SDK samples verified
- CUDASW++

Bioinformatics software for Smith-Waterman protein database searches
- **GPU-Blast**

  Accelerated version of the NCBI-BLAST (Basic Local Alignment Search Tool), a widely used bioinformatics tool.

![Graph](image)

- **Single-GPU applications**
MonteCarlo Multi-GPU (from NVIDIA SDK)
CUDA-MEME application:

- GPU NVIDIA Tesla K40
- Mellanox ConnectX-3 single-port (FDR) and Connect-IB Adapters
Get a free copy of rCUDA at http://www.rcuda.net

@rcuda_r
Hands on rCUDA
rCUDA (remote CUDA) Hands-on Session

Carlos Reaño
Universitat Politècnica de València (Spain)
Outline

- What is rCUDA?
- Installing and using rCUDA
- rCUDA over HPC networks
  - InfiniBand
- How taking benefit from rCUDA
  - Sample scenarios
- Questions & Answers
Outline

- What is rCUDA?
- Installing and using rCUDA
- rCUDA over HPC networks
  - InfiniBand
- How taking benefit from rCUDA
  - Sample scenarios
- Questions & Answers
What is rCUDA?

- CUDA:
- rCUDA (remote CUDA):

With rCUDA Node 2 can use Node 1 GPU!!!
Outline

- What is rCUDA?
- Installing and using rCUDA
- rCUDA over HPC networks
  - InfiniBand
- How taking benefit from rCUDA
  - Sample scenarios
- Questions & Answers
Installing and using rCUDA

- Where to obtain rCUDA?
  - www.rCUDA.net: Software Request Form

- Package contents. Two important folders:
  - doc: rCUDA user guide
  - framework: rCUDA server and library
    - rCUDAd: rCUDA server daemon
    - rCUDAl: rCUDA library

- Installing rCUDA
  - Just untar the tarball in both the server and the client(s) node(s)
Starting rCUDA server:

- Set env. vars as if you were going to run a CUDA program:

```
export PATH=$PATH:/usr/local/cuda/bin
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/usr/local/cuda/lib64
```

- Start rCUDA server:

```
cd $HOME/rCUDA/framework/rCUDAd
./rCUDAd
```
Starting rCUDA server:

- Set env. vars as if you were going to run a CUDA program:
  
  export PATH=$PATH:/usr/local/cuda/bin
  export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/usr/local/cuda/lib64

- Start rCUDA server:
  
  cd $HOME/rCUDA/framework/rCUDAd
  ./rCUDAd
Starting rCUDA server:

- Set env. vars as if you were going to run a CUDA program:
  
  ```
  export PATH=$PATH:/usr/local/cuda/bin
  export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/usr/local/cuda/lib64
  ```

- Start rCUDA server:
  
  ```
  cd $HOME/rCUDA/framework/rCUDAd
  ./rCUDAd
  ```
Installing and using rCUDA

- Starting rCUDA server:
  - Set env. vars as if you were going to run a CUDA program:
    
    ```
    export PATH=$PATH:/usr/local/cuda/bin
    export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/usr/local/cuda/lib64
    ```

  - Start rCUDA server:
    
    ```
    cd $HOME/rCUDA/framework/rCUDAd
    ./rCUDAd
    ```
Installing and using rCUDA

- Starting rCUDA server:
  - Set env. vars as if you were going to run a CUDA program:
    
```
export PATH=$PATH:/usr/local/cuda/bin
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/usr/local/cuda/lib64
```

  - Start rCUDA server:
    
```
cd rCUDA/framework/rCUDAd
./rCUDAd
```

Start rCUDA server in background

IEEE CLUSTER 2014
Running a CUDA program with rCUDA:

- Set env. vars as follows:

```bash
export PATH=$PATH:/usr/local/cuda/bin
export LD_LIBRARY_PATH=$HOME/rCUDA/framework/rCUDA:$LD_LIBRARY_PATH
export RCUDA_DEVICE_COUNT=1
export RCUDA_DEVICE_0=<server_name_or_ip_address>:0
```

- Compile CUDA program using dynamic libraries:

```bash
cd $HOME/NVIDIA_CUDA_Samples/5.5/1_Utilsites/deviceQuery
make EXTRA_NVCCFLAGS=--cudart=shared
```

- Run the CUDA program as usual:

```bash
./deviceQuery
...
```
Running a CUDA program with rCUDA:

- Set env. vars as follows:
  
  ```
  export PATH=$PATH:/usr/local/cuda/bin
  export LD_LIBRARY_PATH=$HOME/rCUDA/framework/rCUDA1:$LD_LIBRARY_PATH
  export RCUDA_DEVICE_COUNT=1
  export RCUDA_DEVICE_0=<server_name_or_ip_address>:0
  ```

- Compile CUDA program using dynamic libraries:
  
  ```
  cd $HOME/NVIDIA_CUDA_Samples/5.5/1_Utilsities/deviceQuery
  make EXTRA_NVCCFLAGS=--cudart=shared
  ```

- Run the CUDA program as usual:
  
  ```
  ./deviceQuery
  ...```
Installing and using rCUDA

- Running a CUDA program with rCUDA:
  - Set env. vars as follows:
    ```bash
    export PATH=$PATH:/usr/local/cuda/bin
    export LD_LIBRARY_PATH=$HOME/rCUDA/framework/rCUDA:$LD_LIBRARY_PATH
    export RCUDA_DEVICE_COUNT=1
    export RCUDA_DEVICE_0=<server_name_or_ip_address>:0
    ```
  - Compile CUDA program using dynamic libraries:
    ```bash
    cd $HOME/NVIDIA_CUDA_Samples/5.5/1_Utilities/deviceQuery
    make EXTRA_NVCCFLAGS=--cudart=shared
    ```
  - Run the CUDA program as usual:
    ```bash
    ./deviceQuery
    ...
    ```
Running a CUDA program with rCUDA:

- Set env. vars as follows:
  - `export PATH=$PATH:/usr/local/cuda/bin`
  - `export LD_LIBRARY_PATH=$HOME/rCUDA/framework/rCUDA1:$LD_LIBRARY_PATH`
  - `export RCUDA_DEVICE_COUNT=1`
  - `export RCUDA_DEVICE_0=<server_name_or_ip_address>:0`

- Compile CUDA program using dynamic libraries:
  - `cd $HOME/NVIDIA_CUDA_Samples/5.5/1_Utilites/deviceQuery`
  - `make EXTRA_NVCCFLAGS=--cudart=shared`

- Run the CUDA program as usual:
  - `./deviceQuery`
  - `...`
Installing and using rCUDA

- Running a CUDA program with rCUDA:
  - Set env. vars as follows:
    
    ```
    export PATH=$PATH:/usr/local/cuda/bin
    export LD_LIBRARY_PATH=$HOME/rCUDA/framework/rCUDA:$LD_LIBRARY_PATH
    export RCUDA_DEVICE_COUNT=1
    export RCUDA_DEVICE_0=<server_name_or_ip_address>:0
    ```
  - Compile CUDA program using dynamic libraries:
    
    ```
    cd $HOME/NVIDIA_CUDA_Samples/5.5/1_Utilsites/deviceQuery
    make EXTRA_NVCCFLAGS=--cudart=shared
    ```
  - Run the CUDA program as usual:
    
    ```
    ./deviceQuery
    ...
Running a CUDA program with rCUDA:

- Set env. vars as follows:
  
  ```
  export PATH=$PATH:/usr/local/cuda/bin
  export LD_LIBRARY_PATH=/HOME/rCUDA/framework/rCUDA:$LD_LIBRARY_PATH
  export RCUDA_DEVICE_COUNT=1
  export RCUDA_DEVICE_0=<server_name_or_ip_address>:0
  ```

- Compile CUDA program using dynamic libraries:
  
  ```
  cd $HOME/NVIDIA_CUDA_Samples/5.5/1_Utilsities/deviceQuery
  make EXTRA_NVCCFLAGS=--cudart=shared
  ```

- Run the CUDA program as usual:
  
  ```
  ./deviceQuery
  ```
Installing and using rCUDA

- Running a CUDA program with rCUDA:
  - Set env. vars as follows:
    ```
    export PATH=$PATH:/usr/local/cuda/bin
    export LD_LIBRARY_PATH=$HOME/rCUDA/framework/rCUDA1:$LD_LIBRARY_PATH
    export RCUDA_DEVICE_COUNT=1
    export RCUDA_DEVICE_0=<server_name_or_ip_address>:0
    ```
  - Compile CUDA program using dynamic libraries:
    ```
    cd $HOME/NVIDIA_CUDA_Samples/5.5/1Utilities/deviceQuery
    make EXTRA_NVCCFLAGS=-cudart=shared
    ```
  - Run the CUDA program as usual:
    ```
    ./deviceQuery
    ...  
    ```
Running a CUDA program with rCUDA:

- Set env. vars as follows:

```
export PATH=$PATH:/usr/local/cuda/bin
export LD_LIBRARY_PATH=$HOME/rCUDA/framework/rCUDA1:$LD_LIBRARY_PATH
export RCUDA_DEVICE_COUNT=1
export RCUDA_DEVICE_0=<server_name_or_ip_address>:0
```

- Compile CUDA program using dynamic libraries:

```
cd $HOME/NVIDIA_CUDA_Samples/5.5/1_Utilities/deviceQuery
make EXTRA_NVCCFLAGS=--cudart=shared
```

- Run the CUDA program as usual:

```
./deviceQuery
...
```
Installing and using rCUDA

- Live demonstration:
  - deviceQuery
  - bandwidthTest
Installing and using rCUDA

- Live demonstration:
  - deviceQuery
  - bandwidthTest

- Problem: bandwidth with rCUDA is too low!!
  - Why? We are using TCP
Installing and using rCUDA

- Live demonstration:
  - deviceQuery
  - bandwidthTest

- Problem: bandwidth with rCUDA is too low!!
  - Why? We are using TCP

- Solution: HPC networks
  - InfiniBand (IB)
Outline

- What is rCUDA?
- Installing and using rCUDA
- rCUDA over HPC networks
  - InfiniBand
- How taking benefit from rCUDA
  - Sample scenarios
- Questions & Answers

IEEE CLUSTER 2014
Starting rCUDA server using IB:

```bash
export RCUDAPROTO=IB
cd $HOME/rCUDA/framework/rCUDA
dir
```

Run CUDA program using rCUDA over IB:

```bash
export RCUDAPROTO=IB
cd $HOME/NVIDIA_CUDA_Samples/5.5/1Utilities/bandwidthTest
dir
```
Starting rCUDA server using IB:

```
export RCUDAPROTO=IB
cd $HOME/rCUDA/framework/rCUDAd
./rCUDAd
```

Tell rCUDA we want to use IB

Run CUDA program using rCUDA over IB:

```
export RCUDAPROTO=IB
cd $HOME/NVIDIA_CUDA_Samples/5.5/1_Utilitys/bandwidthTest
./bandwidthTest
```

Also in the client!!
rCUDA over HPC networks: IB

- Starting rCUDA server using IB:
  ```
  export RCUDAPROTO=IB
  cd $HOME/rCUDA/framework/rCUDAd
  ./rCUDAd
  ```

- Run CUDA program using rCUDA over IB:
  ```
  export RCUDAPROTO=IB
  cd $HOME/NVIDIA_CUDA_Samples/5.5/1_Utilities/bandwidthTest
  ./bandwidthTest
  ```

- Live demonstration:
  - bandwidthTest using IB
  - Bandwidth is no more a problem!!
Outline

- What is rCUDA?
- Installing and using rCUDA
- rCUDA over HPC networks
  - InfiniBand
- How taking benefit from rCUDA
  - Sample scenarios
- Questions & Answers
How taking benefit from rCUDA

- Sample scenarios:
  - **Scalable applications**: more GPUs, less execution time
    - rCUDA can use all the GPUs of the cluster, while CUDA only can use the ones directly connected to one node: for some applications, rCUDA can get better results than with CUDA
  
  - Typical behavior of CUDA applications: moving data to the GPU and performing a lot of computations there to compensate the overhead of having moved the data
    - This benefits rCUDA: more computations, less rCUDA overhead
Multi-GPU Scenario

- **CUDA:**
  Multi-GPU App running in Node 1 using its 4 GPUs

- **rCUDA (remote CUDA):**
  Multi-GPU running in Node 0 using all GPUs in the cluster
Multi-GPU Configuration

- Configure rCUDA for Multi-GPU:

```bash
export PATH=$PATH:/usr/local/cuda/bin
export LD_LIBRARY_PATH=$HOME/rCUDA/framework/rCUDA1:$LD_LIBRARY_PATH
export RCUDA_DEVICE_COUNT=6
export RCUDA_DEVICE_0=rcu16:0
export RCUDADEVICE_1=rcu16:1
export RCUDADEVICE_2=rcu16:2
export RCUDADEVICE_3=rcu16:3
export RCUDADEVICE_4=rcu17:0
export RCUDADEVICE_5=rcu15:0
```

- Check configuration by running deviceQuery sample
Configure rCUDA for Multi-GPU:

```bash
export PATH=$PATH:/usr/local/cuda/bin
export LD_LIBRARY_PATH=$HOME/rCUDA/framework/rCUDA1:$LD_LIBRARY_PATH
export RCUDA_DEVICE_COUNT=6
export RCUDA DEVICE 0=rcu16:0
export RCUDA DEVICE 1=rcu16:1
export RCUDA DEVICE 2=rcu16:2
export RCUDA DEVICE 3=rcu16:3
export RCUDA DEVICE 4=rcu17:0
export RCUDA DEVICE 5=rcu15:0
```

- Check configuration by running deviceQuery sample

Number of remote GPUs
Configure rCUDA for Multi-GPU:

- Export PATH:
  ```bash
  export PATH=/usr/local/cuda/bin
  ```

- Export LD_LIBRARY_PATH:
  ```bash
  export LD_LIBRARY_PATH=$HOME/rCUDA/framework/rCUDA1:$LD_LIBRARY_PATH
  ```

- Export RCUDA_DEVICE_COUNT:
  ```bash
  export RCUDA_DEVICE_COUNT=6
  ```

- Export RCUDA_DEVICE with device numbers:
  ```bash
  export RCUDA_DEVICE_0=rcu16:0
  export RCUDA_DEVICE_1=rcu16:1
  export RCUDADEVICE_2=rcu16:2
  export RCUDA_DEVICE_3=rcu16:3
  export RCUDA DEVICE_4=rcu17:0
  export RCUDA DEVICE_5=rcu15:0
  ```

- Check configuration by running deviceQuery sample

Location of each GPU:
Sample scenarios:

- **Scalable applications**: more GPUs, less execution time
  - rCUDA can use all the GPUs of the cluster, while CUDA only can use the ones directly connected to one node: for some applications, rCUDA can get better results than with CUDA

- **Typical behavior of CUDA applications**: moving data to the GPU and performing a lot of computations there to compensate the overhead of having moved the data
  - This benefits rCUDA: more computations, less rCUDA overhead
Outline

- What is rCUDA?
- Installing and using rCUDA
- rCUDA over HPC networks
  - InfiniBand
- How taking benefit from rCUDA
  - Sample scenarios
- Questions & Answers
Questions?
Exercises

- Local CPU vs. Remote GPU using CUDA
- Local CPU vs. Remote GPU using CUDA Libs
- Aggregated BW of 4 local GPUs vs. 6 remote GPUs
- Multi-GPU App: 4 local GPUs vs. 6 remote GPUs
- Coordinate with assistants for sharing the same GPU at the same time
  - Performance decreases?
Advanced features of rCUDA
On the use of remote GPU virtualization for managing the GPUs of your cluster in a flexible way

Federico Silla
Technical University of Valencia
Spain
Outline

- Scheduling virtual GPUs with SLURM
- rCUDA and KVM virtual machines
Outline

- Scheduling virtual GPUs with SLURM
- rCUDA and KVM virtual machines
• SLURM does not understand about virtualized GPUs

• Add a new GRES (general resource) in order to manage virtualized GPUs

• Where the GPUs are in the system is completely transparent to the user

• In the job script, or in the submission command, the user specifies the number of rGPUs (remote GPUs) required by the job. The amount of GPU memory required by the job may also be specified

Integrating rCUDA with SLURM
The basic idea about SLURM

Resources per job:
- 2: Nodes: 2 GPUs: 1
- 4: Nodes: 2 GPUs: 1
- 3: Nodes: 1 GPU: 0
- 5: Nodes: 1 GPU: 2

Diagram:
- NODE 0
  - GPU 0
  - GPU 1
- NODE 1
  - GPU 0
- NODE 2
- NODE 3

Job queue:
1 2 3 4 5 5 5

srrun: error: Unable to allocate resources
The basic idea about SLURM + rCUDA

GPUs are decoupled from nodes

All jobs are executed in less time
Sharing remote GPUs among jobs

All jobs are executed even in less time.

GPUs are decoupled from nodes.

GPU 0 is scheduled to be shared among jobs.
Sharing a given GPU among jobs

- Several GPU-Blast instances concurrently executed on the same GPU. 2 threads per instance. Each instance uses 1.5GB of GPU memory.
Sharing a given GPU among jobs

- Several instances of the GPU-Blast application concurrently executed on the same GPU. Each instance is reserved 1.5GB of GPU memory
- NVIDIA Tesla K20
- 2 threads per instance
Sharing a given GPU among jobs

- Several instances of the GPU-Blast application concurrently executed on the same GPU. Each instance uses 1.5GB of GPU memory
- NVIDIA Tesla K40
- 2 threads per instance

Sharing the GPU also provides energy savings!!
Sharing remote GPUs among jobs

Sharing GPUs reduces total execution time and energy.

GPUs are decoupled from nodes.

GPU 0 is scheduled to be shared among jobs.

All jobs are executed even in less time.
### slurm.conf

```plaintext
ClusterName=rcu
...
GresTypes=gpu,rgpu
...
NodeName=rcu16 NodeAddr=rcu16 CPUs=8 Sockets=1
   CoresPerSocket=4 ThreadsPerCore=2
RealMemory=7990 Gres=rgpu:4,gpu:4
```
Submitting jobs with SLURM

Submit a job

```
$ srun -N1 --gres=rgpu:4:512M script.sh...
```

```
--rcuda-mode = [excl | shared]
```

Environment variables are initialized by SLURM and used by the rCUDA client (transparently to the user)

- `RCUDA_DEVICE_COUNT=4`
- `RCUDA_DEVICE_0=rcu16:0`
- `RCUDA_DEVICE_1=rcu16:1`
- `RCUDA_DEVICE_2=rcu16:2`
- `RCUDA_DEVICE_3=rcu16:3`

Server name/IP address: GPU
Test bench for analyzing SLURM+rCUDA performance:

- InfiniBand ConnectX-3 based cluster
- CentOS 6.4 Linux
- Dual socket E5-2620v2 Intel Xeon based nodes:
  - 1 node without GPU
  - 8 nodes with NVIDIA K20 GPU
Applications for testing SLURM+rCUDA

Configuration for each of the applications:

<table>
<thead>
<tr>
<th>Application</th>
<th>Configuration</th>
<th>Execution time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>GPU-Blast</td>
<td>1 process with 6 threads</td>
<td>21</td>
</tr>
<tr>
<td>LAMMPS</td>
<td>4 single-thread processes in 4 different nodes</td>
<td>15</td>
</tr>
<tr>
<td>MCUDA-MEME</td>
<td>4 single-thread processes in 4 different nodes</td>
<td>165</td>
</tr>
<tr>
<td><strong>GROMACS</strong></td>
<td>2 processes, with 12 threads each one, in 2 nodes</td>
<td>167</td>
</tr>
</tbody>
</table>

In our tests, GROMACS does not use GPUs. It is a CPU-only application.

Three different workload sizes used:

<table>
<thead>
<tr>
<th>App</th>
<th>Small</th>
<th>Medium</th>
<th>Large</th>
</tr>
</thead>
<tbody>
<tr>
<td>GPU-Blast</td>
<td>12</td>
<td>43</td>
<td>81</td>
</tr>
<tr>
<td>LAMMPS</td>
<td>18</td>
<td>47</td>
<td>90</td>
</tr>
<tr>
<td>MCUDA-MEME</td>
<td>18</td>
<td>36</td>
<td>77</td>
</tr>
<tr>
<td>GROMACS</td>
<td>23</td>
<td>42</td>
<td>79</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td>71</td>
<td>168</td>
<td>327</td>
</tr>
</tbody>
</table>
Cluster performance with rCUDA+SLURM

![Graph showing cluster performance with rCUDA+SLURM](attachment:image.png)

- CUDA (Time)
- rCUDA (Time)
- CUDA (Throughput)
- rCUDA (Throughput)

---

CLUSTER 2014 tutorial, Madrid
Let’s reduce the amount of GPUs in the cluster

![Chart showing execution time for different GPU configurations across small, medium, and large workload sizes.]

- CUDA 8 GPUs
- CUDA 6 GPUs
- CUDA 4 GPUs
- rCUDA 8 GPUs
- rCUDA 6 GPUs
- rCUDA 4 GPUs
The time that GPUs are allocated is increased
Outline

- Scheduling virtual GPUs with SLURM
- rCUDA and KVM virtual machines
Why rCUDA with virtual machines?

- Current clusters frequently use virtual machines (Xen, KVM, Vmware) in order to attain energy and cost reductions

- Applications being executed inside VMs may need access to GPU computing resources

- NVIDIA drivers do not provide virtualization features, thus avoiding the concurrent usage of GPUs from several VMs

- InfiniBand drivers provide virtualization features, thus enabling that several VMs share an InfiniBand NIC

- rCUDA can be used to provide concurrent access to GPUs
How to attach an IB card to a VM

- PCI pass-through is used to assign an IB card (either real or virtual) to a given VM
- The IB card manages the several virtual copies
Analyzing InfiniBand bandwidth (KVM)

**RealX3**: actual Connect-X3 (1 port) card  
**VF**: virtual card copy used from host OS  
**VM**: virtual card copy used from VM
Analyzing InfiniBand latency (KVM)

RealX3: actual Connect-X3 (1 port) card
VF: virtual card copy used from host OS
VM: virtual card copy used from VM
CUDA BW test from a KVM VM

Computer with KVM VMs

Remote computer with Tesla K20

![CUDA BW test graph]

- CUDA
- RealX3-RealX3
- VF-RealX3
- VM-RealX3
Performance of applications from KVM VMs

- CUDASW++
- GPU-BLAST
- CUDA-MEME
- LAMMPS

Comparison of execution time for different methods:
- CUDA
- rCUDA RealX3
- rCUDA VF
- rCUDA VM

Query Length 3005, Query Length 3000, 2000 DNA sequences, in.eam
Comparing GPU virtualization solutions

The diagram compares different GPU virtualization solutions in terms of bandwidth (MB/s) versus transfer size (Bytes). The solutions include CUDA, rCUDA RealX3-RealX3, rCUDA VF-RealX3, rCUDA VM-RealX3, GVirtuS VM-Host VMMShm, GVirtuS VM-RealX3 TCP over IB, DSCUDA RealX3-RealX3, and DSCUDA VM-RealX3.
Conclusions
rCUDA is the enabling technology for …

• **High Throughput Computing**
  - Sharing remote GPUs makes applications to execute slower … **BUT** more throughput (jobs/time) is achieved
  - Datacenter administrators can **choose between HPC and HTC**

• **Green Computing**
  - GPU migration and application migration allow to devote just the required computing resources to the current load

• **More flexible system upgrades**
  - GPU and CPU updates become independent from each other. Adding GPU boxes to non GPU-enabled clusters is possible
Get a free copy of rCUDA at
http://www.rcuda.net

@rcuda_r