Increasing cluster performance by combining rCUDA with Slurm

Federico Silla
Technical University of Valencia
Spain
rCUDA ... what's that?
Basics of CUDA

Application

CUDA libraries

GPU

HPC Advisory Council Switzerland Conference 2016
No GPU

Network
A software technology that enables a more flexible use of GPUs in computing facilities

rCUDA is a development by Technical University of Valencia
Basics of rCUDA

Client side

Application

CUDA Runtime API

client engine

Server side

server engine

CUDA libraries

Network

No GPU

Software

Hardware

GPU
**Basics of rCUDA**

- Application
- CUDA Runtime API
- Client engine
- Server engine
- CUDA libraries

No GPU

Software

Hardware

Network
Basics of rCUDA
rCUDA allows a new vision of a GPU deployment, moving from the usual cluster configuration:

**Physical configuration**

![Physical configuration diagram]

...to the following one:

**Logical configuration**

![Logical configuration diagram]
Two questions:

- Why should we need rCUDA?
- rCUDA ... slower CUDA?
Two questions:

• Why should we need rCUDA?

• rCUDA ... slower CUDA?
The main concern with rCUDA is the reduced bandwidth to the remote GPU.
Using InfiniBand networks

Client side

Application

CUDA Runtime API

rCUDA client engine

common communication API
TCP/IP module
InfiniBand module
Network "X" module

Server side

rCUDA server engine

common communication API
TCP/IP module
InfiniBand module
Network "X" module

CUDA Runtime library

CUDA Driver library

Software

Hardware

Network

GPU
- CUDASW++

Bioinformatics software for Smith-Waterman protein database searches
Initial transfers with InfiniBand

H2D pageable

D2H pageable

H2D pinned

D2H pinned

Bandwidth (MB/s)

Copy Size (MB)

CUDA K20  rCUDA FDR Orig  rCUDA FDR Opt
CUDA K40  rCUDA EDR Orig  rCUDA EDR Opt
Optimized transfers with InfiniBand

H2D pageable

D2H pageable

Almost 100% of available BW

H2D pinned

D2H pinned

CUDA K20  rCUDA FDR Orig  rCUDA FDR Opt
CUDA K40  rCUDA EDR Orig  rCUDA EDR Opt

Optimized transfers with InfiniBand
rCUDA optimizations on applications

- Several applications executed with CUDA and rCUDA
  - K20 GPU and FDR InfiniBand
  - K40 GPU and EDR InfiniBand

Lower is better
Two questions:

• Why should we need rCUDA?

• rCUDA ... slower CUDA?
rCUDA improves cluster performance
Test bench for studying rCUDA+Slurm

- Dual socket E5-2620v2 Intel Xeon + 32GB RAM + K20 GPU
- FDR InfiniBand based cluster
Applications for studying rCUDA+Slurm

- Applications used for tests:
  - GPU-Blast (21 seconds; 1 GPU; 1599 MB)
  - LAMMPS (15 seconds; 4 GPUs; 876 MB)
  - MCUDA-MEME (165 seconds; 4 GPUs; 151 MB)
  - GROMACS (2 nodes) (167 seconds)
  - NAMD (4 nodes) (11 minutes)
  - BarraCUDA (10 minutes; 1 GPU; 3319 MB)
  - GPU-LIBSVM (5 minutes; 1GPU; 145 MB)
  - MUMmerGPU (5 minutes; 1GPU; 2804 MB)

- Three workloads:
  - Set 1
  - Set 2
  - Set 1 + Set 2
### Workloads for studying rCUDA+Slurm (I)

<table>
<thead>
<tr>
<th>Application</th>
<th>Workload</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Set 1</td>
<td>Set 2</td>
<td>Set 1+2</td>
</tr>
<tr>
<td>GPU-Blast</td>
<td>112</td>
<td></td>
<td>57</td>
</tr>
<tr>
<td>LAMMPS</td>
<td>88</td>
<td></td>
<td>52</td>
</tr>
<tr>
<td>mCUDA-MEME</td>
<td>99</td>
<td></td>
<td>55</td>
</tr>
<tr>
<td>GROMACS</td>
<td>101</td>
<td></td>
<td>47</td>
</tr>
<tr>
<td>BarraCUDA</td>
<td></td>
<td>112</td>
<td>51</td>
</tr>
<tr>
<td>MUMmerGPU</td>
<td></td>
<td>88</td>
<td>52</td>
</tr>
<tr>
<td>GPU-LIBSVM</td>
<td></td>
<td>99</td>
<td>37</td>
</tr>
<tr>
<td>NAMD</td>
<td></td>
<td>101</td>
<td>49</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>400</strong></td>
<td><strong>400</strong></td>
<td><strong>400</strong></td>
</tr>
</tbody>
</table>
Performance of rCUDA+Slurm (I)

- Execution Time:
  - Set 1: CUDA 47%, rCUDA -47%
  - Set 2: CUDA 37%, rCUDA -27%
  - Set 1+2: CUDA 27%, rCUDA 39%

- Energy (kWh):
  - Set 1: CUDA -23%, rCUDA -14%
  - Set 2: CUDA -39%
  - Set 1+2: CUDA -14%

- GPU Utilization:
  - Set 1: CUDA 130%, rCUDA 56%
  - Set 2: CUDA 104%
  - Set 1+2: CUDA 56%
Applications for studying rCUDA+Slurm

- Applications used for tests:
  - GPU-Blast (21 seconds; 1 GPU; 1599 MB)
  - LAMMPS (15 seconds; 4 GPUs; 876 MB)
  - MCUDA-MEME (165 seconds; 4 GPUs; 151 MB)
  - GROMACS (2 nodes) (167 seconds) **Set 1**
  - NAMD (4 nodes) (11 minutes) **Set 2**
  - BarraCUDA (10 minutes; 1 GPU; 3319 MB)
  - GPU-LIBSVM (5 minutes; 1 GPU; 145 MB)
  - MUMmerGPU (5 minutes; 1 GPU; 2804 MB)

- Three workloads:
  - Set 1
  - Set 2
  - Set 1 + Set 2
## Workloads for studying rCUDA+Slurm (II)

<table>
<thead>
<tr>
<th>Application</th>
<th>Set 2</th>
<th>Set 1+2</th>
<th>New Set 2</th>
<th>New Set 1+2</th>
</tr>
</thead>
<tbody>
<tr>
<td>GPU-Blast</td>
<td>57</td>
<td></td>
<td>50</td>
<td></td>
</tr>
<tr>
<td>LAMMPS</td>
<td>52</td>
<td></td>
<td>50</td>
<td></td>
</tr>
<tr>
<td>mCUDA-MEME</td>
<td>55</td>
<td></td>
<td>50</td>
<td></td>
</tr>
<tr>
<td>GROMACS</td>
<td>47</td>
<td></td>
<td>50</td>
<td></td>
</tr>
<tr>
<td>BarraCUDA</td>
<td>112</td>
<td>51</td>
<td>40</td>
<td>20</td>
</tr>
<tr>
<td>MUMmerGPU</td>
<td>88</td>
<td>52</td>
<td>100</td>
<td>50</td>
</tr>
<tr>
<td>GPU-LIBSVM</td>
<td>99</td>
<td>37</td>
<td>160</td>
<td>80</td>
</tr>
<tr>
<td>NAMD</td>
<td>101</td>
<td>49</td>
<td>100</td>
<td>50</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>400</strong></td>
<td><strong>400</strong></td>
<td><strong>400</strong></td>
<td><strong>400</strong></td>
</tr>
</tbody>
</table>
Performance of rCUDA+Slurm (II)

Performance improvements:
- Execution Time:
  - New Set 2: -38%
  - New Set 1+2: -36%
- Energy Consumption:
  - New Set 2: -26%
  - New Set 1+2: -25%
- GPU Utilization:
  - New Set 2: +17%
  - New Set 1+2: +97%
Why does rCUDA improve cluster performance?
1st reason for improved performance

- Non-accelerated applications **keep GPUs idle** in the nodes where they use all the cores

Hybrid MPI shared-memory non-accelerated applications usually span to all the cores in a node (across $n$ nodes)

A CPU-only application spreading over these nodes will make their GPUs unavailable for accelerated applications

![Diagram of network nodes and components](image)

Interconnection Network
Accelerated applications keep CPUs idle in the nodes where they execute. Shared-memory non-accelerated applications usually span to all the cores in a node. An accelerated application using just one CPU core may avoid other jobs to be dispatched to this node.
2\textsuperscript{nd} reason for improved performance (II)

- Accelerated applications keep CPUs idle in the nodes where they execute.

Hybrid MPI shared-memory non-accelerated applications usually span to all the cores in a node (across \( n \) nodes).

An accelerated MPI application using just one CPU core per node may keep part of the cluster busy.
3rd reason for improved performance

- Do applications **completely squeeze** the GPUs available in the cluster?
  - When a GPU is assigned to an application, computational resources inside the GPU may not be fully used
    - Application presenting low level of parallelism
    - CPU code being executed
    - GPU-core stall due to lack of data
    - etc …

(GPU assigned ≠ GPU working)
GPU usage of GPU-Blast

- Core Utilization
- Memory Utilization (accesses)

GPU assigned but not used

Power (W)
GPU usage of CUDA-MEME

GPU utilization is far away from maximum

![Graph showing GPU utilization over time.](image)

- **Utilization**
  - Memory Utilization (accesses)
  - Core Utilization

- **Power (W)**

**Time**

00:00:00 00:00:20 00:00:40 00:01:00 00:01:20 00:02:00 00:02:20 00:03:00 00:03:20 00:04:00 00:04:20 00:05:00 00:05:20 00:06:00 00:06:20 00:07:00 00:07:20 00:08:00

- **Power (W)**

**Time**

00:00:00 00:00:20 00:00:40 00:01:00 00:01:20 00:02:00 00:02:20 00:03:00 00:03:20 00:04:00 00:04:20 00:05:00 00:05:20 00:06:00 00:06:20 00:07:00 00:07:20 00:08:00
GPU usage of LAMMPS

GPU assigned but not used
GPU allocation vs GPU utilization

Results for CUDA!!

GPUs assigned but not used
Sharing a GPU among jobs: GPU-Blast

One instance required about 51 seconds

Two concurrent instances of GPU-Blast
Sharing a GPU among jobs: GPU-Blast

Two concurrent instances of GPU-Blast

First instance

Power (W)

Core Utilization
Memory Utilization (accesses)
Sharing a GPU among jobs: GPU-Blast

Two concurrent instances of GPU-Blast

First instance

Second instance

Power (W)

Core Utilization

Memory Utilization ( accesses)
Sharing a GPU among jobs

- LAMMPS: 876 MB
- mCUDA-MEME: 151 MB
- BarraCUDA: 3319 MB
- MUMmerGPU: 2104 MB
- GPU-LIBSVM: 145 MB
Other reasons for using rCUDA?
Cheaper cluster upgrade

- Let’s suppose that a cluster without GPUs needs to be upgraded to use GPUs

No GPU

- GPUs require large power supplies
  - Are power supplies already installed in the nodes large enough?
- GPUs require large amounts of space
  - Does current form factor of the nodes allow to install GPUs?

The answer to both questions is usually “NO”
Approach 1: augment the cluster with some CUDA GPU-enabled nodes \(\rightarrow\) only those GPU-enabled nodes can execute accelerated applications.
Approach 2: augment the cluster with some rCUDA servers → all nodes can execute accelerated applications
Cheaper cluster upgrade

- Dual socket E5-2620v2 Intel Xeon + 32GB RAM + K20 GPU
- FDR InfiniBand based cluster

16 nodes without GPU + 1 node with 4 GPUs
More workloads for studying rCUDA+Slurm

<table>
<thead>
<tr>
<th>Application</th>
<th>Workload</th>
<th>WL 1</th>
<th>WL 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>GPU-Blast</td>
<td></td>
<td>41</td>
<td>48</td>
</tr>
<tr>
<td>LAMMPS short</td>
<td></td>
<td>39</td>
<td>46</td>
</tr>
<tr>
<td>LAMMPS long 2p</td>
<td></td>
<td>20</td>
<td>10</td>
</tr>
<tr>
<td>LAMMPS long 4p</td>
<td></td>
<td>20</td>
<td>10</td>
</tr>
<tr>
<td>mCUDA-MEME short</td>
<td></td>
<td>39</td>
<td>46</td>
</tr>
<tr>
<td>mCUDA-MEME long 2p</td>
<td></td>
<td>20</td>
<td>10</td>
</tr>
<tr>
<td>mCUDA-MEME long 4p</td>
<td></td>
<td>20</td>
<td>10</td>
</tr>
<tr>
<td>GROMACS</td>
<td></td>
<td>40</td>
<td>40</td>
</tr>
<tr>
<td>BarraCUDA</td>
<td></td>
<td>40</td>
<td>47</td>
</tr>
<tr>
<td>MUMmerGPU</td>
<td></td>
<td>41</td>
<td>47</td>
</tr>
<tr>
<td>GPU-LIBSVM</td>
<td></td>
<td>40</td>
<td>46</td>
</tr>
<tr>
<td>NAMD</td>
<td></td>
<td>40</td>
<td>40</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td></td>
<td>400</td>
<td>400</td>
</tr>
</tbody>
</table>
Additional reasons for using rCUDA?
#1: More GPUs for a single application

Detected 64 CUDA Capable device(s)

Device 0: "Tesla M2090"
CUDA Driver Version / Runtime Version: 5.0 / 5.0
CUDA Capability Major/Minor version number: 2.0
Total amount of global memory: 6144 MB (644213264 bytes)
(16) Multiprocessors x (32) CUDA Cores/MP: 512 CUDA Cores
GPU Clock rate: 1301 MHz (1.30 GHz)
Memory Clock rate: 1848 Mhz
Memory Bus Width: 384-bit
L2 Cache Size: 786432 bytes
Max Texture Dimension Size (x,y,z): 1D=(65536), 2D=(65536,65535), 3D=(2048,2048,2048)
Max Layered Texture Size (dim) x layers: 1D=(16384) x 2048, 2D=(16384,16384) x 2048
Total amount of constant memory: 65536 bytes
Total amount of shared memory per block: 49152 bytes
Total number of registers available per block: 32768
Warp size: 32
Maximum number of threads per multiprocessor: 1536
Maximum number of threads per block: 1024
Maximum sizes of each dimension of a block: 1024 x 1024 x 64
Maximum sizes of each dimension of a grid: 65535 x 65535 x 65535
Maximum memory pitch: 2147483647 bytes
Texture alignment: 512 bytes
Concurrent copy and kernel execution: Yes with 2 copy engine(s)
Run time limit on kernels: No
Integrated GPU sharing Host Memory: No
Support host page-locked memory mapping: No
Alignment requirement for Surfaces: Yes
Device has ECC support: Disabled
Device supports Unified Addressing (UVA): Yes
Device PCI Bus ID / PCI location ID: 2 / 0
Compute Mode: < Default (multiple host threads can use ::cudaSetDevice() with device simultaneously) >

Device 1: "Tesla M2090"
CUDA Driver Version / Runtime Version: 5.0 / 5.0

64 GPUs!
#1: More GPUs for a single application

- MonteCarlo Multi-GPU (from NVIDIA samples)

FDR InfiniBand + NVIDIA Tesla K20

<table>
<thead>
<tr>
<th>Number of GPUs</th>
<th>CUDA</th>
<th>rCUDA</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1000</td>
<td>1000</td>
</tr>
<tr>
<td>2</td>
<td>1000</td>
<td>1000</td>
</tr>
<tr>
<td>4</td>
<td>1000</td>
<td>1000</td>
</tr>
<tr>
<td>6</td>
<td>1000</td>
<td>1000</td>
</tr>
<tr>
<td>8</td>
<td>1000</td>
<td>1000</td>
</tr>
<tr>
<td>10</td>
<td>1000</td>
<td>1000</td>
</tr>
<tr>
<td>12</td>
<td>1000</td>
<td>1000</td>
</tr>
<tr>
<td>14</td>
<td>1000</td>
<td>1000</td>
</tr>
</tbody>
</table>

Lower is better
#2: Virtual machines can share GPUs

- The GPU is assigned by using PCI passthrough exclusively to a single virtual machine
- Concurrent usage of the GPU is not possible
#2: Virtual machines can share GPUs

Virtual machines can share GPUs.
... in summary ...
**Pros and cons of rCUDA**

• **Cons:**
  1. Reduced bandwidth to remote GPU (really a concern??)

• **Pros:**
  1. Many GPUs for a single application
  2. Concurrent GPU access to virtual machines
  3. Increased cluster throughput
  4. Similar performance with smaller investment
  5. Easier (cheaper) cluster upgrade
  6. Reduced energy consumption
  7. Increased GPU utilization
  8. *Migration of GPU jobs*

rCUDA is a development by Technical University of Valencia
Get a free copy of rCUDA at
http://www.rcuda.net
More than 650 requests world wide

@rcuda_

rCUDA is a development by Technical University of Valencia
Thanks!
Questions?

cCUDA is a development by Technical University of Valencia