Virtualización remota de GPUs

Federico Silla
Universidad Politécnica de Valencia
Remote GPU virtualization

Federico Silla
Technical University of Valencia
Spain
What is “remote GPU virtualization”?
We deal with GPUs, obviously!

![NVIDIA CUDA](image)
Basic behavior of CUDA
Remote GPU virtualization

A **software** technology that enables **a more flexible use of GPUs** in computing facilities
Basics of remote GPU virtualization

Client side | Server side

Application

CUDA Runtime API

client engine

server engine

CUDA libraries

Network

Software

Hardware

GPU
Basics of remote GPU virtualization

Client side

Application

CUDA Runtime API

client engine

Software

Hardware

Network

Server side

server engine

CUDA libraries

GPU
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

How rCUDA works
How to declare remote GPUs

Environment variables are properly initialized in the client side and used by the rCUDA client (transparently to the application)

Server name/IP address: GPU

Amount of GPUs exposed to applications
Remote GPU virtualization envision

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

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

**Logical connections**

**Logical configuration**

**Physical configuration**

**Interconnection Network**

**Interconnection Network**

**Interconnection Network**
2nd

Why is “remote GPU virtualization” needed?
Which is the problem with GPU-enabled clusters?
A GPU-enabled cluster is 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, **included GPUs**)
- 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 some 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

![Graph showing idle power over time for 1 GPU node and 4 GPUs node]

- 1 GPU node: Two E5-2620V2 sockets and 32GB DDR3 RAM. One Tesla K20 GPU
- 4 GPUs node: Two E5-2620V2 sockets and 128GB DDR3 RAM. Four Tesla K20 GPUs

Money leakage in current clusters?
Applications can only use the GPUs located within their node:

- non-MPI 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)
One more concern with accelerated clusters

- 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 assigned ≠ GPU working)
    - GPU-core stall due to lack of data
    - etc …
Sharing a given GPU among jobs

- Several **GPU-Blast** instances concurrently executed on the same GPU. Each instance uses about 1.5 of GPU memory.
Why performance in GPU clusters is lost?

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 …

• GPU cycles are lost, thus reducing cluster performance
We need something more in the cluster

The current model for using GPUs is too rigid

What is missing is ...

... some flexibility for using the GPUs in the cluster
We need something more in the cluster

The current model for using GPUs is too rigid

What is missing is ...

... some flexibility for using the GPUs in the cluster

A way of seamlessly sharing GPUs across nodes in the cluster (remote GPU virtualization)
Remote GPU virtualization envision

Without GPU virtualization

With GPU virtualization

Virtualized remote GPUs

GPU virtualization allows all nodes to access all GPUs
Several efforts have been made to implement remote GPU virtualization during the last years:

<table>
<thead>
<tr>
<th>Framework</th>
<th>CUDA Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>rCUDA</td>
<td>CUDA 7.0</td>
</tr>
<tr>
<td>GVirtuS</td>
<td>CUDA 3.2</td>
</tr>
<tr>
<td>DS-CUDA</td>
<td>CUDA 4.1</td>
</tr>
<tr>
<td>vCUDA</td>
<td>CUDA 1.1</td>
</tr>
<tr>
<td>GViM</td>
<td>CUDA 1.1</td>
</tr>
<tr>
<td>GridCUDA</td>
<td>CUDA 2.3</td>
</tr>
<tr>
<td>V-GPU</td>
<td>CUDA 4.0</td>
</tr>
</tbody>
</table>

Remote GPU virtualization frameworks

- Publicly available
- Not publicly available
Remote GPU virtualization frameworks

FDR InfiniBand + K20 !!

H2D pageable

D2H pageable

H2D pinned

D2H pinned

- CUDA
- rCUDA
- GVirtuS
- DS-CUDA

Bandwidth (MB/s)

Transfer Size (MB)
Cons of "remote GPU virtualization"?
The main drawback of GPU virtualization 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
InfiniBandNetwork "X" module

Server side

CUDA Runtime API

rCUDA server engine

common communication API
TCP/IP module
InfiniBandNetwork "X" module

CUDA Runtime library

CUDA Driver library

GPU

Software

Hardware

Network
Initial transfers within rCUDA

H2D pageable

D2H pageable

H2D pinned

D2H pinned
Improving communications within rCUDA

- Several optimizations applied to rCUDA:
  - At the InfiniBand level:
    - Optimization intended for short transfers
    - Optimization intended for long transfers
  - At the rCUDA level:
    - The internal pipeline has been redesigned
Applications use **InfiniBand** by creating **queue pairs (QPs)**:

- QPs do not store data but work requests (WRs)
- Work requests are descriptors of the transfer operation to be done
- Applications can **associate several QPs to the same network adapter**
  - The more QPs, the higher the complexity of keeping them synchronized

**Diagram:**

- Two Network Cards connected by a connection
- Each Network Card has a Port and QP ( send, receive )
- Connections between Ports of different Network Cards
Improving communications within rCUDA

- Optimization for short transfers:
  - increase the capacity of the send/receive queues (i.e., number of work requests that can be allocated)
Improving communications within rCUDA

- **Optimization for long transfers:**
  - increase the number of QPs used for a single transfer
Optimized transfers within rCUDA

- **H2D pageable**: Almost 100% of available BW
- **D2H pageable**: Almost 100% of available BW
- **H2D pinned**: Almost 100% of available BW
- **D2H pinned**: Almost 100% of available BW
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 are the reasons for the better performance of rCUDA:

1. Higher bandwidth for pageable memory

2. Network polling interval to check for work completions: CUDA polls PCIe whereas rCUDA polls InfiniBand network adapter
Rodinia performance with rCUDA

InfiniBand EDR + K40 !!

- Short execution time
- Medium execution time
- Long execution time
Rodinia performance with rCUDA

InfiniBand EDR + K40 !!

Jornadas de Paralelismo. Córdoba 2015
Pros of "remote GPU virtualization"?
1: more GPUs for a single application

- GPU virtualization is 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
1: more GPUs for a single application

Detecting 64 CUDA Capable devices:

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 MBytes (6442123264 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

Higher is better

Lower is better
2: busy CPU cores do not block GPUs

Physical configuration

Logical connections

Logical configuration
A cluster without GPUs may be easily upgraded to use GPUs with rCUDA.
3: easier cluster upgrade

- A cluster without GPUs may be easily upgraded to use GPUs with rCUDA

**GPU-enabled**
4: GPU task migration

- Consolidate GPUs into dedicated GPU boxes (with a low-power CPU)
- Allow GPU task migration
Box A has **4 GPUs** but only **one** is busy

Box B has **8 GPUs** but only **two** are busy

1. Move jobs from Box B to Box A and switch off Box B
2. Migration should be transparent to applications (decided by the global scheduler)
5: virtual machines can easily access GPUs

- The GPU is assigned by using PCI passthrough exclusively to a single virtual machine
- Concurrent usage of the GPU is not possible

Computer hosting several KVM virtual machines
How to attach an InfiniBand 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
5: virtual machines can easily access GPUs

Computer hosting several KVM virtual machines

KVM Host Linux

SW BRIDGE

Gb ETH

IB PF

IB VF

PCI PT

InfiniBand Fabric

rCUDA server

GPU

KVM Guest Linux 1

rCUDA client

vGPU

vETH

IB

KVM Guest Linux n

rCUDA client

vGPU

vETH

IB

Computer hosting several KVM virtual machines

KVM Host Linux

SW BRIDGE

Gb ETH

GPU

rCUDA server

High performance network available

Low performance network available
Application performance with KVM

FDR InfiniBand + K20 !!

LAMMPS

CUDA-MEME

CUDA
CUDA VM-PT
rCUDA non-VM
rCUDA VM IB
rCUDA VM Local

CUDASW++

GPU-BLAST
Application performance with Xen

FDR InfiniBand + K20 !!

LAMMPS

CUDA-MEME

<table>
<thead>
<tr>
<th>Execution Time (s)</th>
<th>Input model</th>
<th>eam</th>
<th>lj</th>
</tr>
</thead>
<tbody>
<tr>
<td>CUDA</td>
<td>40</td>
<td>30</td>
<td>20</td>
</tr>
<tr>
<td>CUDA VM-PT</td>
<td>45</td>
<td>35</td>
<td>25</td>
</tr>
<tr>
<td>rCUDA non-VM</td>
<td>50</td>
<td>40</td>
<td>30</td>
</tr>
<tr>
<td>rCUDA VM IB</td>
<td>55</td>
<td>45</td>
<td>35</td>
</tr>
<tr>
<td>rCUDA VM Local</td>
<td>60</td>
<td>50</td>
<td>40</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Execution Time (s)</th>
<th># DNA sequences</th>
<th>500</th>
<th>1000</th>
<th>2000</th>
</tr>
</thead>
<tbody>
<tr>
<td>CUDA</td>
<td>5000</td>
<td>4000</td>
<td>3000</td>
<td>2000</td>
</tr>
<tr>
<td>CUDA VM-PT</td>
<td>5500</td>
<td>4500</td>
<td>3500</td>
<td>2500</td>
</tr>
<tr>
<td>rCUDA non-VM</td>
<td>6000</td>
<td>5000</td>
<td>4000</td>
<td>3000</td>
</tr>
<tr>
<td>rCUDA VM IB</td>
<td>6500</td>
<td>5500</td>
<td>4500</td>
<td>3500</td>
</tr>
<tr>
<td>rCUDA VM Local</td>
<td>7000</td>
<td>6000</td>
<td>5000</td>
<td>4000</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Execution Time (s)</th>
<th>Query length</th>
<th>144</th>
<th>1000</th>
<th>2005</th>
<th>3005</th>
<th>4061</th>
<th>5147</th>
</tr>
</thead>
<tbody>
<tr>
<td>CUDA</td>
<td>12</td>
<td>8</td>
<td>6</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>CUDA VM-PT</td>
<td>14</td>
<td>10</td>
<td>8</td>
<td>6</td>
<td>4</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>rCUDA non-VM</td>
<td>16</td>
<td>12</td>
<td>10</td>
<td>8</td>
<td>6</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>rCUDA VM IB</td>
<td>18</td>
<td>14</td>
<td>12</td>
<td>10</td>
<td>8</td>
<td>6</td>
<td>4</td>
</tr>
<tr>
<td>rCUDA VM Local</td>
<td>20</td>
<td>16</td>
<td>14</td>
<td>12</td>
<td>10</td>
<td>8</td>
<td>6</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Execution Time (s)</th>
<th>Sequence length</th>
<th>800</th>
<th>1600</th>
<th>2000</th>
<th>2404</th>
<th>2800</th>
</tr>
</thead>
<tbody>
<tr>
<td>CUDA</td>
<td>100</td>
<td>80</td>
<td>60</td>
<td>40</td>
<td>20</td>
<td>10</td>
</tr>
<tr>
<td>CUDA VM-PT</td>
<td>120</td>
<td>100</td>
<td>80</td>
<td>60</td>
<td>40</td>
<td>20</td>
</tr>
<tr>
<td>rCUDA non-VM</td>
<td>140</td>
<td>120</td>
<td>100</td>
<td>80</td>
<td>60</td>
<td>40</td>
</tr>
<tr>
<td>rCUDA VM IB</td>
<td>160</td>
<td>140</td>
<td>120</td>
<td>100</td>
<td>80</td>
<td>60</td>
</tr>
<tr>
<td>rCUDA VM Local</td>
<td>180</td>
<td>160</td>
<td>140</td>
<td>120</td>
<td>100</td>
<td>80</td>
</tr>
</tbody>
</table>

Jornadas de Paralelismo. Córdoba 2015
Overhead of rCUDA within VMs

- **KVM**
  - LAMMPS: 1.6%
  - CUDA-MEME: 2.5%
  - CUDASW++: 0.5%
  - GPU-BLAST: 0.07%

- **Xen**
  - LAMMPS: 6.1%
  - CUDA-MEME: 2.9%
  - CUDASW++: 0.7%
  - GPU-BLAST: 2.3%

- FDR InfiniBand + K20!!
What happens at the cluster level?
• **GPUs can be shared** among jobs running in remote clients
  • Job scheduler required for coordination
  • **Slurm** was selected
The basic idea about SLURM

Resources per job

1. Nodes: 2 GPUs: 3
2. Nodes: 2 GPUs: 1
3. Nodes: 1 GPUs: 0
4. Nodes: 2 GPUs: 1
5. Nodes: 1 GPUs: 2

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

GPUs are decoupled from nodes

All jobs are executed in less time
GPUs are decoupled from nodes. All jobs are executed even in less time.

GPU 0 is scheduled to be shared among jobs.
GPU memory is the limit

![Diagram showing GPU memory allocation for different applications](image)
GPU memory is the limit

Does App 7 fit into the GPU?

GPU memory

App 1
App 2
App 3
App 4
App 5
App 6
Integrating rCUDA with SLURM

- SLURM does not know about virtualized GPUs
- SLURM must be enhanced in order to manage the new virtualized GPUs
- A new resource has been added: \texttt{rgpu}

\texttt{gres.conf}

Name = \texttt{rgpu} File = / dev/ nvidia0 Cuda =3.5 Mem =4726 M \\
[ Name = \texttt{gpu} File = / dev/ nvidia0 ]

\texttt{slurm.conf}

SelectType = select / cons_rgpu \\
SelectTypeParameters = CR\_CORE \\
GresTypes = \texttt{rgpu}, \texttt{gpu}

NodeName = node1 NodeHostname = node1 \\
CPUs =12 Sockets =2 CoresPerSocket =6 \\
ThreadsPerCore =1 RealMemory =32072 \\
Gres = \texttt{rgpu}:1, \texttt{gpu}:1

New submission options:

\texttt{--rcuda-mode=(shared|excl)}
\texttt{--gres=rgpu(X:(Y)?:(Z)?)}?

\begin{align*}
X &= \lbrack 1-9 \rbrack + \lbrack 0-9 \rbrack^* \\
Y &= \lbrack 1-9 \rbrack + \lbrack 0-9 \rbrack \cdot \lbrack \text{KkMmGg} \rbrack \\
Z &= \lbrack 1-9 \rbrack . \lbrack 0-9 \rbrack \cdot \lbrack \text{cc|CC} \rbrack
\end{align*}
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 (167 seconds)
- NAMD (11 minutes)
- BarraCUDA (10 minutes; 1 GPU; 3319 MB)
- GPU-LIBSVM (5 minutes; 1 GPU; 145 MB)
- MUMmerGPU (5 minutes; 1 GPU; 2804 MB)

Non-GPU

Three workloads:
- Set 1
- Set 2
- Set 1 + Set 2

Three workload sizes:
- Small (100 jobs)
- Medium (200 jobs)
- Large (400 jobs)
- Dual socket E5-2620v2 Intel Xeon + 32GB RAM + K20 GPU
- FDR InfiniBand based cluster

4 GPU nodes

8 GPU nodes

node with the Slurm scheduler

node with the Slurm scheduler

16 GPU nodes
Results for execution time

Results for Set 1

Lower is better
Results for Set 1

Higher is better

Lower is better

4 nodes

8 nodes

16 nodes

Jornadas de Paralelismo. Córdoba 2015
Results for energy consumption

Lower is better

Results for Set 1
Reducing the amount of GPUs

Execution Time (s)

- CUDA 16 GPUs
- rCUDA 16 GPUs
- CUDA 8 GPUs
- rCUDA 8 GPUs
- CUDA 4 GPUs
- rCUDA 4 GPUs

Lower is better

35% Less
39% Less
41% Less

Workload size

Small
Medium
Large
Reducing the amount of GPUs

Lower is better

Cost (in Euros):
- a) 16 nodes + 16 K20 … 16*2500 + 16*2000 = 72000 Euros
- b) 16 nodes + 4 K20 … 16*2500 + 4*2000 = 48000 Euros

Reducing 33% acquisition expenses, performance is still 40% better
Energy when removing GPUs

Lower is better

39% Less
42% Less
44% Less
Costs:

... reducing 33% acquisition expenses, performance is still 40% better ...

Additionally, the electricity bill is significantly reduced
GPU utilization when removing GPUs

<table>
<thead>
<tr>
<th>Workload size</th>
<th>CUDA 16 GPUs</th>
<th>rCUDA 16 GPUs</th>
<th>CUDA 8 GPUs</th>
<th>rCUDA 8 GPUs</th>
<th>CUDA 4 GPUs</th>
<th>rCUDA 4 GPUs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Small</td>
<td>0.3</td>
<td>0.25</td>
<td>0.25</td>
<td>0.25</td>
<td>0.25</td>
<td>0.25</td>
</tr>
<tr>
<td>Medium</td>
<td>0.7</td>
<td>0.65</td>
<td>0.6</td>
<td>0.65</td>
<td>0.65</td>
<td>0.65</td>
</tr>
<tr>
<td>Large</td>
<td>0.8</td>
<td>0.75</td>
<td>0.7</td>
<td>0.75</td>
<td>0.75</td>
<td>0.75</td>
</tr>
</tbody>
</table>

Higher is better
... 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. Migration of GPU jobs
  7. Reduced energy consumption
  8. Increased GPU utilization
**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**
Get a free copy of rCUDA at
http://www.rcuda.net
More than 650 requests world wide

@rcuda_
Thanks!

Questions?