Improving overall performance and energy consumption of your cluster with remote GPU virtualization

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

- **9.00 - 10.00  SESSION 1**
  - Introduction to the remote GPU virtualization technique and frameworks

- **10.00 - 10.30  Coffee break**

- **10.30 - 12:00  SESSION 2**
  - Talk on advanced features of remote GPU virtualization (40 minutes)
  - Practical demonstration about how to install and use rCUDA (50 minutes)

- **12:00 - 13:30  Lunch**

- **13:30 - 15:00  SESSION 3**
  - Guided exercises with rCUDA in remote cluster

- **15:00 - 15:30  Coffee break**

- **15:30 - 17:30  SESSION 4**
  - Time for freely exercise with rCUDA in remote cluster
Introduction to the remote GPU virtualization technique and frameworks

Federico Silla
Universidad Politécnica de Valencia
Spain
1. Introduction to GPU computing
2. What is remote GPU virtualization?
3. Why is remote GPU virtualization needed?
4. Frameworks for remote GPU virtualization
5. Cons of remote GPU virtualization
6. Pros of remote GPU virtualization
Introduction to GPU computing
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

**CUDA Tutorial. Middleware’15. December 7th. Vancouver**

**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

GPUs are not the magic solution for all applications
Current GPU computing facilities

The basic building block is a node with one or more GPUs
Basics of GPU computing

Basic behavior of CUDA

Application

CUDA libraries

GPU
Basics of GPU computing

```c
#include <cuda.h>
#include <stdio.h>

const int N = 8;

__global__ void my_gpu_function (int *a, int *b)
{
    b[threadIdx.x] = a[threadIdx.x] * 2;
}

int main()
{
    int a[N] = {0, 1, 2, 3, 4, 5, 6, 7};

    int *ad, *bd;
    const int isize = N * sizeof(int);

    // Allocate GPU memory
    cudaMalloc((void**)&ad, isize);
    cudaMalloc((void**)&bd, isize);

    // Copy data to GPU memory
    cudaMemcpy(ad, a, isize, cudaMemcpyHostToDevice);

    // Execute function in the GPU
    my_gpu_function<<<1, N>>>(ad, bd);

    // Copy results from GPU memory
    cudaMemcpy(b, bd, isize, cudaMemcpyDeviceToHost);

    // Free GPU memory
    cudaFree(ad);
    cudaFree(bd);

    return 0;
}
```
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-enabled cluster is a set of independent self-contained nodes. The cluster follows 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
Outline

What is “remote GPU virtualization”?

Improving overall performance & energy consumption of your cluster with remote GPU virtualization
It has to do with GPUs, obviously!
Remote GPU virtualization

A software technology that enables a more flexible use of GPUs in computing facilities.
Same source code for using virtual GPUs

Application source code does not change. The change is linking against the remote GPU virtualization libraries instead of using the original CUDA libraries.

```c
#include <cuda.h>
#include <stdio.h>

const int N = 8;

__global__ void my_gpu_function (int *a, int *b)
{
    b[threadIdx.x] = a[threadIdx.x] * 2;
}

int main ( )
{
    int a[N] = {0, 1, 2, 3, 4, 5, 6, 7};
    int *ad, *bd;
    const int isize = N * sizeof(int);
    // Perform some computations in the CPU
    … CPU code … CPU code …

    // Allocate GPU memory
    cudaMalloc ((void**)&ad, isize);
    cudaMalloc ((void**)&bd, isize);

    // Copy data to GPU memory
    cudaMemcpy(ad, a, isize, cudaMemcpyHostToDevice);

    // Execute function in the GPU
    my_gpu_function<<<1, N>>>(ad, bd);

    // Copy results from GPU memory
    cudaMemcpy(b, bd, isize, cudaMemcpyDeviceToHost);

    // Free GPU memory
    cudaFree(ad);
    cudaFree(bd);
    return 0;
}
```
Basics of remote GPU virtualization

Client side | Server side
---|---
Application
CUDA Runtime API
client engine

Network

server engine
CUDA libraries

Software
Hardware

GPU
Basics of remote GPU virtualization

Client side | Server side

Application

CUDA Runtime API

client engine

server engine

CUDA libraries

Software

Hardware

Network

GPU
Remote GPU virtualization envision

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

  **Physical configuration**

  **Logical configuration**

  Interconnection Network
Remote GPU virtualization envision

Without GPU virtualization

With GPU virtualization

Virtualized remote GPUs

GPU virtualization allows all nodes to share all GPUs
Why is "remote GPU virtualization" needed?
Which is the problem with GPU-enabled clusters?
The problem is the cluster architecture

- A GPU-enabled cluster is a set of independent self-contained nodes. The cluster follows 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

- 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

---

Network
Network
Network
Network

Interconnection Network
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

![Image of GPU nodes and power consumption chart]

- 1 GPU node: Two E5-2620 v2 sockets and 32GB DDR3 RAM. One Tesla K20 GPU
- 4 GPU node: Two E5-2620 v2 sockets and 128GB DDR3 RAM. Four Tesla K20 GPUs
First concern with accelerated clusters (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 application using just one CPU core may avoid other jobs to be dispatched to this node.
First concern with accelerated clusters (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.
Second concern with accelerated clusters

- Non-MPI multi-GPU applications cannot make use of the tremendous GPU resources available across the cluster (even if those GPU resources are idle)

All these GPUs cannot be used by the multi-GPU application being executed.
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 …
**GPU usage of GPU-Blast**

**Graph 1:**
- **Core Utilization** and **Memory Utilization (accesses)**
- Key: GPU assigned but not used

**Graph 2:**
- **Power (W)**
- Key: Power
GPU usage of CUDA-MEME

GPU utilization is far away from maximum
GPU usage of LAMMPS

- Core Utilization
- Memory Utilization (accesses)

GPU assigned but not used
GPU usage of CUDASW

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

- **GPU assigned but not used**
- **GPU assigned but not used**
GPU allocation vs GPU utilization

- Normalized Workload Execution Time
- Normalized GPU Allocation Time
- GPU Utilization

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
Sharing a GPU among jobs: GPU-Blast

Two concurrent instances of GPU-Blast

First instance

Second instance
Sharing a GPU among jobs: GPU-Blast

One instance required about 51 seconds

Four concurrent instances of GPU-Blast
Several **GPU-Blast** instances concurrently executed on the same GPU. Each instance requires about 1.5 GB of GPU memory.

Execution time mismatch with previous slides due to the use of different input data sets!!
Sharing a GPU among jobs

K20 GPU

- LAMMPS: 876 MB
- mCUDA-MEME: 151 MB
- BarraCUDA: 3319 MB
- MUMmerGPU: 2104 MB
- GPU-LIBSVM: 145 MB
Accompanied 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.
• 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

Idle CPU cores in nodes executing accelerated applications could be used for other GPU applications
In summary …

- There are scenarios where GPUs are available but cannot be used
- Also, CPU cores are available but cannot be used
- Accelerated applications do not make use of 100% of GPU resources

In conclusion …

- GPU and CPU 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

Physical configuration

Interconnection Network

Logical connections

Logical configuration

Interconnection Network
Increasing flexibility: one step further

- Once GPUs are shared, their **amount** can be **reduced** to match the actual workload.
- This would increase GPU utilization, also lowering power consumption, at the same time that initial acquisition costs are reduced.
Sharing GPUs among applications

Without GPU virtualization

With GPU virtualization

GPU virtualization allows all nodes to share all GPUs
What is needed for increased flexibility?

In summary … 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
4th

Frameworks for remote GPU virtualization
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>

rCUDA is a development by Technical University of Valencia
Remote GPU virtualization frameworks

**FDR InfiniBand + K20 !!**

- **H2D pageable**
- **D2H pageable**
- **H2D pinned**
- **D2H pinned**
rCUDA uses optimized transfers

- rCUDA features **optimized data transfers:**
  - Pipelined transfers to improve performance
  - Preallocated pinned memory buffers
  - Optimal pipeline block size
Moving data WITHOUT a pipeline

CLIENT NODE

- Main memory
- CPU
- chipset
- InfiniBand

SERVER NODE

- Main memory
- CPU
- chipset
- InfiniBand
- GPU
- Memory

Pageable memory area to be copied to remote GPU

Destination memory area in the remote GPU
Moving data WITHOUT a pipeline

Pageable memory area to be copied to remote GPU

Memory area copied to network buffers for RDMA

Destination memory area in the remote GPU
Moving data WITHOUT a pipeline

Pageable memory area to be copied to remote GPU

Memory area copied to network buffers for RDMA

Memory area arrives at network buffers at remote server

Destination memory area in the remote GPU
Moving data WITHOUT a pipeline

**CLIENT NODE**

<table>
<thead>
<tr>
<th>Main memory</th>
<th>CPU</th>
<th>InfiniBand</th>
<th>InfiniBand</th>
<th>GPU</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pageable memory area to be copied to remote GPU</td>
<td>Memory area copied to network buffers for RDMA</td>
<td>Memory area arrives at network buffers at remote server</td>
<td>Destination memory area in the remote GPU</td>
<td></td>
</tr>
</tbody>
</table>

**SERVER NODE**

<table>
<thead>
<tr>
<th>Main memory</th>
<th>CPU</th>
<th>chipset</th>
<th>InfiniBand</th>
<th>chipset</th>
<th>GPU</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Moving data WITHOUT a pipeline

Main memory to network buffers

Network from client to remote server

Network buffers to GPU memory

Transmission time without pipeline
Moving data WITH a pipeline

Pageable memory area to be copied to remote GPU

Destination memory area in the remote GPU
Moving data WITH a pipeline

- Pageable memory area to be copied to remote GPU

- Memory area is split into chunks (pipeline blocks)

- Destination memory area in the remote GPU
Moving data WITH a pipeline

**CLIENT NODE**

- Main memory
- CPU
- InfiniBand
- chipset

**SERVER NODE**

- Main memory
- CPU
- InfiniBand
- chipset
- GPU

Pageable memory area to be copied to remote GPU

**Destination memory area in the remote GPU**

Only two small network buffers at source and destination are required

1 2 3 4 5
Moving data WITH a pipeline

Transmission starts by moving forward the first block

1 2 3 4 5

Pageable memory area to be copied to remote GPU

Destination memory area in the remote GPU
Moving data WITH a pipeline

2 pipeline stages concurrently move data forward
Moving data WITH a pipeline

The three stages of the pipeline are concurrently transferring data forward.

Pageable memory area to be copied to remote GPU

Destination memory area in the remote GPU
Moving data WITH a pipeline

CLIENT NODE

SERVER NODE

Pageable memory area to be copied to remote GPU

Destination memory area in the remote GPU
Moving data **WITH** a pipeline

**CLIENT NODE**
- Main memory
- CPU
- InfiniBand
- chipset

**SERVER NODE**
- Main memory
- CPU
- InfiniBand
- chipset
- GPU
- GPU memory

Pageable memory area to be copied to remote GPU

Destination memory area in the remote GPU
Moving data WITH a pipeline

CLIENT NODE

SERVER NODE

Pageable memory area to be copied to remote GPU

Destination memory area in the remote GPU
Moving data WITH a pipeline

CLIENT NODE

SERVER NODE

Pageable memory area to be copied to remote GPU

Destination memory area in the remote GPU
Moving data without a pipeline

Main memory to network buffers
Network from client to remote server
Network buffers to GPU memory

Management of data transfer is much more complex

Transmission time without pipeline
Transmission time with pipeline
Improvement
Pipeline block size for InfiniBand Connect-IB

Basic performance analysis

- NVIDIA Tesla K40; Mellanox Connect-IB + SX6036 Mellanox switch
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
Environment variables are properly initialized in the client side and used by the rCUDA client (transparently to the application)
Cons of "remote GPU virtualization"
The main drawback of remote 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

rCUDA server engine

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

CUDA Runtime library

CUDA Driver library

Software

Hardware

Network

GPU
- CUDASW++

Bioinformatics software for Smith-Waterman protein database searches

Performance of rCUDA

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

Dual socket E5-2620 v2 Intel Xeon node with NVIDIA K20 GPU
Initial transfers of rCUDA with InfiniBand

H2D pageable

D2H pageable

H2D pinned

D2H pinned

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

Almost 100% of available BW

H2D pinned

D2H pinned
rCUDA optimizations on applications

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

Lower is better

[Bar chart showing execution times for different applications and configurations.]
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

Clock sample from NVIDIA SDK
Rodinia performance with rCUDA

InfiniBand EDR + K40 !!
InfiniBand EDR + K40 !!
LAMMPS performance with rCUDA

InfiniBand EDR + K40

InfiniBand FDR + K20
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

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 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

Computation Time (ms)

Options per second

Number of GPUs
2: busy CPU cores do not block GPUs

Physical configuration

Logical configuration

Interconnection Network
3: easier cluster upgrade

• 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 should be “NO”
3: easier cluster upgrade

Approach 1: augment the cluster with some CUDA GPU-enabled nodes → only those GPU-enabled nodes can execute accelerated applications
Approach 2: augment the cluster with some rCUDA servers → all nodes can execute accelerated applications

**GPU-enabled**

Network
3: easier cluster upgrade

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

15 nodes without GPU + 1 node with 4 GPUs
Applications used for tests:
- GPU-Blast (21 seconds; 1 GPU; 1599 MB)
- LAMMPS short (90 seconds; 1 GPU; 2633 MB)
- LAMMPS long 2p (149 seconds; 2 GPUs; 3950 MB)
- LAMMPS long 4p (71 seconds; 4 GPUs; 2385 MB)
- mCUDA-MEME short (510 seconds; 1 GPU; 151 MB)
- mCUDA-MEME long 2p (1182 seconds; 2 GPUs; 152 MB)
- mCUDA-MEME long 4p (631 seconds; 4 GPUs; 152 MB)
- BarraCUDA (10 minutes; 1 GPU; 3319 MB)
- GPU-LIBSVM (5 minutes; 1 GPU; 145 MB)
- MUMmerGPU (5 minutes; 1 GPU; 2804 MB)
- GROMACS (167 seconds)
- NAMD (11 minutes)
3: easier cluster upgrade

<table>
<thead>
<tr>
<th>Application</th>
<th>Workload</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>WL 1</td>
</tr>
<tr>
<td>GPU-Blast</td>
<td>41</td>
</tr>
<tr>
<td>LAMMPS short</td>
<td>39</td>
</tr>
<tr>
<td>LAMMPS long 2p</td>
<td>20</td>
</tr>
<tr>
<td>LAMMPS long 4p</td>
<td>20</td>
</tr>
<tr>
<td>mCUDA-MEME short</td>
<td>39</td>
</tr>
<tr>
<td>mCUDA-MEME long 2p</td>
<td>20</td>
</tr>
<tr>
<td>mCUDA-MEME long 4p</td>
<td>20</td>
</tr>
<tr>
<td>GROMACS</td>
<td>40</td>
</tr>
<tr>
<td>BarraCUDA</td>
<td>40</td>
</tr>
<tr>
<td>MUMmerGPU</td>
<td>41</td>
</tr>
<tr>
<td>GPU-LIBSVM</td>
<td>40</td>
</tr>
<tr>
<td>NAMD</td>
<td>40</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>400</strong></td>
</tr>
</tbody>
</table>
3: easier cluster upgrade

![Bar chart showing execution time comparison between CUDA and rCUDA across different workloads.
- WL 1: CUDA 80k, rCUDA 20k, -68%
- WL 2: CUDA 60k, rCUDA 40k, -60%

![Bar chart showing GPU utilization comparison between CUDA and rCUDA across different workloads.
- WL 1: CUDA 0.4, rCUDA 0.3, +131%
- WL 2: CUDA 0.2, rCUDA 0.2, +119%

![Bar chart showing energy consumption comparison between CUDA and rCUDA across different workloads.
- WL 1: CUDA 40kWh, rCUDA 15kWh, -63%
- WL 2: CUDA 30kWh, rCUDA 13.5kWh, -56%]
4: GPU consolidation

- Concentrate 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)
4: GPU consolidation: one step beyond

Job granularity instead of GPU granularity
5. Easier access from the inside of virtual machines to GPUs
6. Increased performance at the cluster level
   • Cluster throughput (jobs/hour) is **doubled**

**More details in the second part of the tutorial**
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
Get a free copy of rCUDA at
http://www.rcuda.net
More than 650 requests world wide

@rcuda_

Carlos Reaño  Javier Prades  Sergio Iserte

rCUDA is a development by Technical University of Valencia
Thanks!

Questions?
Advanced features of rCUDA

Federico Silla
Universidad Politécnica de Valencia
5. Easier access from the inside of virtual machines to GPUs
6. Increased performance at the cluster level
   • Cluster throughput (jobs/hour) is **doubled**

More details in the second part of the tutorial
5: virtual machines can easily access GPUs

- How to access the GPU in the native domain from the inside of the virtual machines?
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
How to attach an InfiniBand card to a VM

Physical Function; PF

Virtual IB card copies (Virtual Functions; VFs)

- PCI passthrough is used to assign an InfiniBand card (either real or virtual) to a given VM
- The InfiniBand 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

Host HW

InfiniBand Fabric

rCUDA server

GPU

High performance network available

Low performance network available

KVM

rCUDA Tutorial. Middleware’15. December 7th. Vancouver
5: virtual machines can easily access GPUs

- High performance network available
- Low performance network available

Xen
Application performance with KVM

FDR InfiniBand + K20 !!

![Graph showing execution time for LAMMPS with different input models and # DNA sequences.]

- **LAMMPS**
- **CUDA-MEME**

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

![Graph showing execution time for CUDASW++ with different query lengths.]

- **CUDASW++**
- **GPU-BLAST**

![Graph showing execution time for GPU-BLAST with different sequence lengths.]

rCUDA Tutorial. Middleware’15. December 7th. Vancouver
Application performance with Xen

FDR InfiniBand + K20 !!

LAMMPS

CUDA-MEME

CUDASW++

GPU-BLAST

CUDA Tutorial. Middleware’15. December 7th. Vancouver
Overhead of rCUDA within VMs

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

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

FDR InfiniBand + K20 !!
5: increased performance for clusters
• **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

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

NODE 0
- GPU 0
- GPU 1

NODE 1
- GPU 0

NODE 2

NODE 3

Job queue

sr: 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

- GPUs are decoupled from nodes.
- GPU 0 is scheduled to be shared among jobs.
- All jobs are executed even in less time.
GPU memory is the limit

GPU memory

App 1

App 2

App 3

App 4

App 5

App 6
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

App 7

rCUDA Tutorial. Middleware’15. December 7th. Vancouver
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: **rgpu**

**gres.conf**
Name = rgpu File = /dev/nvidia0 Cuda=3.5 Mem=4726 M
[ Name = gpu File = /dev/nvidia0 ]

**slurm.conf**
SelectType = select / cons_rgpu
SelectTypeParameters = CR_CORE
GresTypes = rgpu [gpu]

nodeName = node1 NodeHostname = node1
CPUs=12 Sockets=2 CoresPerSocket=6
ThreadsPerCore=1 RealMemory=32072
Gres = rgpu :1, gpu :1

**New submission options:**
--rcuda-mode=(shared|excl)
--gres=rgpu(:X(:Y)?(:Z)?)
  X = [1-9]+[0-9]*
  Y = [1-9]+[0-9]*[ kKmMgG]
  Z = [1-9].[0-9](cc|CC)
### Ongoing work: 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 (167 seconds)**
- **NAMD (11 minutes)**
- **BarraCUDA (10 minutes; 1 GPU; 3319 MB)**
- **GPU-LIBSVM (5 minutes; 1GPU; 145 MB)**
- **MUMmerGPU (5 minutes; 1GPU; 2104 MB)**

**Non-GPU**

- **Short execution time**
  - **Set 1**
  - **GROMACS (167 seconds)**

- **Set 2**
  - **NAMD (11 minutes)**

**Long execution time**

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

- **Three workload sizes:**
  - Small (100 jobs)
  - Medium (200 jobs)
  - Large (400 jobs)
## Ongoing work: studying rCUDA+Slurm

<table>
<thead>
<tr>
<th>Application</th>
<th>Set 1</th>
<th>Set 2</th>
<th>Set 1+2</th>
</tr>
</thead>
<tbody>
<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>88</td>
<td></td>
<td>52</td>
</tr>
<tr>
<td>GPU-LIBSVM</td>
<td>99</td>
<td></td>
<td>37</td>
</tr>
<tr>
<td>NAMD</td>
<td>101</td>
<td></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>
Test bench for studying rCUDA+Slurm

- Dual socket E5-2620v2 Intel Xeon + 32GB RAM + K20 GPU
- FDR InfiniBand based cluster
Results for 16-node cluster

- **Execution Time (s)**
  - Lower is better

- **Workload size is 400 jobs**

- **Energy (kWh)**
  - Lower is better

- **GPU Utilization**
  - Higher is better
Results for 16-node cluster

![Graph showing time (s) for different workloads and allocations using CUDA and rCUDA, with separate bars for Alloc Time and Exec Time.]
## Results for 16-node cluster

<table>
<thead>
<tr>
<th>Application</th>
<th>Workload</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Set 2</td>
<td>Set 1+2</td>
<td>New Set 2</td>
<td>New Set 1+2</td>
</tr>
<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>
Results for 16-node cluster

- Execution Time (s)
  - CUDA
  - rCUDA ex
  - rCUDA sh

Workload size is 400 jobs

- Workload
  - New Set 2
  - New Set 1+2

- Lower is better

- Higher is better

- Energy (kWh)

- GPU Utilization

- Workload
  - New Set 2
  - New Set 1+2

- Lower is better
non-homogeneous cluster

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

15 nodes with 1 GPU + 1 node with 4 GPUs
Workload size is 400 jobs
non-homogeneous cluster

![Bar chart showing CUDA Alloc Time, CUDA Exec Time, rCUDA sh Alloc Time, and rCUDA sh Exec Time for WL 1 and WL 2. The chart shows a comparison of time in seconds for different workloads.](chart.png)
... 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
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

More than 650 requests world wide

@rcuda_

rCUDA is a development by Technical University of Valencia
Thanks!

Questions?
Hands-on Session
(Part I)

Carlos Reaño
Technical University of Valencia
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. Important folders:
  - doc: rCUDA user guide
  - bin: rCUDA server daemon
  - lib: rCUDA library

- Installing rCUDA
  - Just untar the tarball in both the server and the client(s) node(s)
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/bin
./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/bin
    ./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/bin
    ./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/bin
    ./rCUDA
    ```
Installing and using rCUDA

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

    ```bash
    export PATH=$PATH:/usr/local/cuda/bin
    export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/usr/local/cuda/lib64
    ```
  - Start rCUDA server:

    ```bash
    cd $HOME/rCUDA/bin
    ./rCUDAd
    ```

    Start rCUDA server in background
Running a CUDA program with rCUDA:

- Set env. vars as follows:

  ```
  export PATH=/usr/local/cuda/bin
  export LD_LIBRARY_PATH=$HOME/rCUDA/lib:$LD_LIBRARY_PATH
  export RCUDA_DEVICE_COUNT=1
  export RCUDADEVICE_0=<server_name_or_ip_address>:0
  ```

- Compile CUDA program using dynamic libraries:

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

- Run the CUDA program as usual:

  ```
  ./deviceQuery
  ...```

---

This slide provides instructions on how to run a CUDA program using rCUDA. It includes setting environment variables, compiling the program, and running it as usual.
Running a CUDA program with rCUDA:

- Set env. vars as follows:

  ```
  export PATH=$PATH:/usr/local/cuda/bin
  export LD_LIBRARY_PATH=$HOME/rCUDA/lib:$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/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/lib:$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/1_Utilities/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/lib:$LD_LIBRARY_PATH
  export RCUDADEVICECOUNT=1
  export RCUDADEVICE0=<server_name_or_ip_address>:0
  ```

- Compile CUDA program using dynamic libraries:

  ```
  cd $HOME/NVIDIA_CUDA_Samples/1_Utilities/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/lib:$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/1_Utilities/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/lib:$LD_LIBRARY_PATH
    export RCUDA_DEVICE_COUNT=1
    export RCUDA_DEVICE_0=<server_name_or_ip_address>:0
    ```
  - GPU of remote server to use
  - Compile CUDA program using dynamic libraries:
    ```
    cd $HOME/NVIDIA_CUDA_Samples/1Utilities/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/lib:$LD_LIBRARY_PATH
    export RCUDA_DEVICE_COUNT=1
    export RCUDADEVICE_0=<server_name_or_ip_address>:0
    ```
  - Compile CUDA program using dynamic libraries:
    ```
    cd $HOME/NVIDIA_CUDA_Samples/1_Utilslettes/deviceQuery
    make EXTRA_NVCCFLAGS=---cudart=shared
    ```
  - Run the CUDA program as usual:
    ```
    ./deviceQuery
    ...  
    ```
    Very important!!!
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/lib:$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/1_Utilities/deviceQuery
    make EXTRA_NVCCFLAGS=--cudart=shared
    ```
  - Run the CUDA program as usual:
    ```
    ./deviceQuery
    ```
Live demonstration:
- deviceQuery
- bandwidthTest
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)
• What is rCUDA?
• Installing and using rCUDA
• rCUDA over HPC networks
  • InfiniBand
• How taking benefit from rCUDA
  • Sample scenarios
• Questions & Answers
Starting rCUDA server using IB:

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

Run CUDA program using rCUDA over IB:

```
export RCUDAPROTO=IB
cd $HOME/NVIDIA_CUDA_Samples/1Utilities/bandwidthTest
./bandwidthTest
```
rCUDA over HPC networks: InfiniBand

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

- Run CUDA program using rCUDA over IB:
  ```
  export RCUDAPROTO=IB
  cd $HOME/NVIDIA_CUDA_Samples/1_Utils/bandwidthTest
  ./bandwidthTest
  ```
rCUDA over HPC networks: InfiniBand

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

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

- Live demonstration:
  - bandwidthTest using IB
  - Bandwidth is no more a problem!!
● What is rCUDA?
● Installing and using rCUDA
● rCUDA over HPC networks
  ● InfiniBand
● **How taking benefit from rCUDA**
  ● Sample scenarios
● Questions & Answers
Sample scenarios:

- **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

- **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
How taking benefit from rCUDA

- Three main types of applications:
  - Bandwidth bounded: more transfers, more rCUDA overhead
  - Computations bounded: more computations, less rCUDA overhead
  - Intermediate
LAMMPS Application

- GPU vs. remote GPU
  - Overhead of remote GPUs?
- Live demonstration:
  - LAMMPS with CUDA
  - LAMMPS with rCUDA
LAMMPS Application

- CPU vs. remote GPU
  - What is better: a local CPU or a remote GPU?
- Live demonstration:
  - LAMMPS on CPU (without CUDA)
CPU vs. remote GPU
- What is better: a local CPU or a remote GPU?

Live demonstration:
- LAMMPS on CPU (without CUDA)
- LAMMPS on CPU (using all the cores!)
CPU vs. remote GPU
- What is better: a local CPU or a remote GPU?

Live demonstration:
- LAMMPS on CPU (without CUDA)
- LAMMPS on CPU (using all the cores!)

What if I am using TCP instead of IB?
- LAMMPS using rCUDA over TCP (1GbE)
GPU vs. remote GPU
  - Overhead of remote GPUs?

Live demonstration:
  - MAGMA with CUDA
  - MAGMA with rCUDA
CPU vs. remote GPU
  - What is better: a local CPU or a remote GPU?
Live demonstration:
  - MAGMA on CPU (without CUDA)
CPU vs. remote GPU
- What is better: a local CPU or a remote GPU?

Live demonstration:
- MAGMA on CPU (without CUDA)

What if I am using TCP instead of IB?
- MAGMA using rCUDA over TCP (1GbE)
Sample scenarios:

- 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

- **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
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
Configure rCUDA for Multi-GPU:

- Export PATH:
  ```
  export PATH=$PATH:/usr/local/cuda/bin
  ```
- Export LD_LIBRARY_PATH:
  ```
  export LD_LIBRARY_PATH=$HOME/rCUDA/framework/rCUDA:$LD_LIBRARY_PATH
  ```
- Export RCUDA_DEVICE_COUNT:
  ```
  export RCUDA_DEVICE_COUNT=5
  ```
- Export RCUDA_DEVICE_0:
  ```
  export RCUDA_DEVICE_0=node1:0
  ```
- Export RCUDA_DEVICE_1:
  ```
  export RCUDA_DEVICE_1=node1:1
  ```
- Export RCUDA_DEVICE_2:
  ```
  export RCUDA_DEVICE_2=node2:0
  ```
- Export RCUDA_DEVICE_3:
  ```
  export RCUDA_DEVICE_3=node3:0
  ```
- Export RCUDA_DEVICE_4:
  ```
  export RCUDA_DEVICE_4=node4:0
  ```

- 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=5
export RCUDADEVICE0=node1:0
export RCUDADEVICE1=node1:1
export RCUDADEVICE2=node2:0
export RCUDADEVICE3=node3:0
export RCUDADEVICE4=node4:0
```

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

- Check configuration by running deviceQuery sample

```bash
export PATH=$PATH:/usr/local/cuda/bin
export LD_LIBRARY_PATH=$HOME/rCUDA/framework/rCUDA1:$LD_LIBRARY_PATH
export RCUDA_DEVICE_COUNT=5
export RCUDA_DEVICE_0=node1:0
export RCUDA_DEVICE_1=node1:1
export RCUDA_DEVICE_2=node2:0
export RCUDA_DEVICE_3=node3:0
export RCUDA_DEVICE_4=node4:0
```
Multi-GPU Configuration

- Live demonstration:
  - deviceQuery sample with multiple GPUs
  - LAMMPS multiple remote GPUs
  - MAGMA multiple remote GPUs
  - MonteCarloMultiGPU
- What is rCUDA?
- Installing and using rCUDA
- rCUDA over HPC networks
  - InfiniBand
- How taking benefit from rCUDA
  - Sample scenarios
- Questions & Answers
Get a free copy of rCUDA at
http://www.rcuda.net

@rcuda_r
Hands-on Session
(Part II)

Carlos Reaño
Technical University of Valencia
Spain
Outline

- Short intro to CUDA programming
- Cluster for testing
- Installing rCUDA
- Running some codes
- Questions & Answers
• Short intro to CUDA programming
• Cluster for testing
• Installing rCUDA
• Running some codes
• Questions & Answers
CUDA programming language = C extended:
- For running functions in the GPU: called kernels
- ...

CUDA API:
- Functions to copy data from main memory to GPU memory and vice versa
- ...

![Diagram of CPU, GPU, and memory connections]
Short intro to CUDA programming

CUDA typical code layout:

- Copy input data from main memory to GPU memory
- Perform some computation in GPU (launch kernel)
- Copy output data from GPU memory to main memory
Sample code:

```c
// CUDA Kernel
__global__ void myCUDAKernel(...) {
    // Perform some computation in the GPU
    ...
}

int main() {
    ...
    // Copy input data to GPU mem
    cudaMemcpy(gpuMemPtr, memPtr, size, cudaMemcpyHtoD);
    ...
    // Launch kernel
    myCUDAKernel<<<nThreads, nBlocks>>>(...);
    ...
    // Copy output data to main mem
    cudaMemcpy(memPtr, gpuMemPtr, size, cudaMemcpyDtoH);
}
```
Outline

- Short intro to CUDA programming
- Cluster for testing
- Installing rCUDA
- Running some codes
- Questions & Answers
“rcu” cluster:

- rcu3
- rcu4
- rcu5
- ... (continues)
- rcu29

Network (1 GbE)

- rcu15
  - GPU
- rcu16
  - GPU
  - GPU
  - GPU
  - GPU

Cluster for testing
Cluster for testing

- "rcu" cluster:

Nodes without GPUs for users (rCUDA clients)
Cluster for testing

“rcu” cluster:

Nodes without GPUs for users (rCUDA clients)

rcu3
rcu4
rcu5
...
rcu29

Network (1 GbE)

rcu15

rcu16

Nodes with GPUs for rCUDA servers
Outline

- Short intro to CUDA programming
- Cluster for testing
- Installing rCUDA
- Running some codes
- Questions & Answers
Installing and using rCUDA

- Log into rcu cluster

- rCUDA software package already in $HOME folder

- Installing rCUDA
  - Just untar the tarball in $HOME folder

- Package contents. Important folders:
  - doc: rCUDA user guide
  - bin: rCUDA server daemon
  - lib: rCUDA library
Installing and using rCUDA

- CUDA must be installed in GPU nodes
- In the nodes running rCUDA servers
- Already installed in our cluster
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/bin
  ./rCUDAd
  ```
Installing and using rCUDA

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

Start rCUDA server in background
Running a CUDA program with rCUDA:

- Set env. vars as follows:

  ```
  export PATH=$PATH:/usr/local/cuda/bin
  export LD_LIBRARY_PATH=$HOME/rCUDA/lib:$LD_LIBRARY_PATH
  export RCUDA_DEVICE_COUNT=1
  export RCUDA_DEVICE_0=rcu16:0
  ```

- Compile CUDA program using dynamic libraries:

  ```
  cd $HOME/NVIDIA_CUDA_Samples/1_Utilities/deviceQuery
  make
  ```

- 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/lib:$LD_LIBRARY_PATH
  export RCUDA_DEVICE_COUNT=1
  export RCUDA_DEVICE_0=rcu16:0
  ```
- Compile CUDA program using dynamic libraries:
  ```
  cd $HOME/NVIDIA_CUDA_Samples/1_Utilities/deviceQuery
  make
  ```
- 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/lib:$LD_LIBRARY_PATH
  export RCUDA_DEVICE_COUNT=1
  export RCUDA_DEVICE_0=rcu16:0
  ```

- Compile CUDA program using dynamic libraries:
  ```
  cd $HOME/NVIDIA_CUDA_Samples/1_Utilities/deviceQuery
  make
  ```

- 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=/usr/local/cuda/bin
    export LD_LIBRARY_PATH=$HOME/rCUDA/lib:$LD_LIBRARY_PATH
    export RCUDA_DEVICE_COUNT=1
    export RCUDA_DEVICE_0=rcu16:0
    ```
  - Compile CUDA program using dynamic libraries:
    ```bash
    cd $HOME/NVIDIA_CUDA_Samples/1_Utilities/deviceQuery
    make
    ```
  - Run the CUDA program as usual:
    ```bash
    ./deviceQuery
    ...
    ```
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/lib:$LD_LIBRARY_PATH
  export RCUDA_DEVICE_COUNT=1
  export RCUDA_DEVICE_0=rcu16:0
  ```

- Compile CUDA program using dynamic libraries:
  ```bash
  cd $HOME/NVIDIA_CUDA_Samples/1_Utilsities/deviceQuery
  make
  ```

- Run the CUDA program as usual:
  ```bash
  ./deviceQuery
  ...
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/lib:$LD_LIBRARY_PATH
  export RCUDA_DEVICE_COUNT=1
  export RCUDADEVICE_0=rcu16:0
  ```

- Compile CUDA program using dynamic libraries:

  ```bash
  cd $HOME/NVIDIA_CUDA_Samples/1_Utilities/deviceQuery
  make
  ```

- Run the CUDA program as usual:

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

- Set env. vars as follows:

```shell
export PATH=$PATH:/usr/local/cuda/bin
export LD_LIBRARY_PATH=$HOME/rCUDA/lib:$LD_LIBRARY_PATH
export RCUDA_DEVICE_COUNT=1
export RCUDA_DEVICE_0=rcu16:0
```

- Compile CUDA program using dynamic libraries:

```shell
cd $HOME/NVIDIA_CUDA_Samples/1_Utilites/deviceQuery
make
```

- Run the CUDA program as usual:

```shell
./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/lib:$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/1_Utilsities/deviceQuery
    make
    ```
  - 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/lib:$LD_LIBRARY_PATH
    export RCUDA_DEVICE_COUNT=1
    export RCUDADEVICE_0=<server_name_or_ip_address>:0
    ```
  - Compile CUDA program using dynamic libraries:
    ```
    cd $HOME/NVIDIA_CUDA_Samples/1_Utilities/deviceQuery
    make
    ```
  - Run the CUDA program as usual:
    ```
    ./deviceQuery
    ... Error! Why?
    ```
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/lib:$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/1Utilities/deviceQuery
make EXTRA_NVCCFLAGS=--cudart=shared
```

- Run the CUDA program as usual:

```bash
./deviceQuery
...```

Very important!!!
Running a CUDA program with rCUDA:

- Set env. vars as follows:
  
  ```
  export PATH=$PATH:/usr/local/cuda/bin
  export LD_LIBRARY_PATH=$HOME/rCUDA/lib:$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/6.5/1_Utilities/deviceQuery
  make EXTRA_NVCCFLAGS=--cudart=shared
  ```

- Run the CUDA program as usual:
  
  ```
  ./deviceQuery
  ```
- Short intro to CUDA programming
- Cluster for testing
- Installing rCUDA
- Running some codes
- Questions & Answers
Impact of multiple users sharing the same GPU

- Measure time of running deviceQuery sample 10 times when the remote GPU is only used by one user

- Measure time of running deviceQuery sample 10 times when multiple users are sharing the same GPU

- Does execution time increases? With how many concurrent users the increase is more noticeable?
Impact of multiple users sharing the same GPU

- Measure time of running deviceQuery sample 10 times when the remote GPU is only used by one user

- Measure time of running deviceQuery sample 10 times when multiple users are sharing the same GPU

- Does execution time increases? With how many concurrent users the increase is more noticeable?

- deviceQuery sample: no computations, no transfers
Impact of multiple users sharing the same GPU

- Measure bandwidth when copying from main memory to remote GPU memory when the remote GPU is only used by one user

- Measure bandwidth when copying from main memory to remote GPU memory when the remote GPU when multiple users are sharing the same GPU

- Does bandwidth decreases? With how many concurrent users the decrease is more noticeable? What happens if we increase the size of the transfer?
Impact of multiple users sharing the same GPU

- Measure bandwidth when copying from main memory to remote GPU memory when the remote GPU is only used by one user.

- Measure bandwidth when copying from main memory to remote GPU memory when the remote GPU when multiple users are sharing the same GPU.

- Does bandwidth decreases? With how many concurrent users the decrease is more noticeable? What happens if we increase the size of the transfer?

- bandwidthTest sample: no computations, only transfers
Impact of multiple users sharing the same GPU

- Measure performance of vector addition when the remote GPU is only used by one user
- Measure performance of vector addition when multiple users are sharing the same GPU
- Does performance decreases? With how many concurrent users the decrease is more noticeable? What happens if we increase the size of the vector?
Impact of multiple users sharing the same GPU

- Measure performance of vector addition when the remote GPU is only used by one user
- Measure performance of vector addition when multiple users are sharing the same GPU
- Does performance decreases? With how many concurrent users the decrease is more noticeable? What happens if we increase the size of the vector?

vectorAdd sample: computations and transfers
Impact of multiple users sharing the same GPU

- Measure performance of matrix multiplication when the remote GPU is only used by one user.

- Measure performance of matrix multiplication when multiple users are sharing the same GPU.

- Does performance decreases? With how many concurrent users the decrease is more noticeable? What happens if we increase the size of the matrix?
Impact of multiple users sharing the same GPU

- Measure performance of matrix multiplication when the remote GPU is only used by one user
- Measure performance of matrix multiplication when multiple users are sharing the same GPU
- Does performance decrease? With how many concurrent users does the decrease become more noticeable? What happens if we increase the size of the matrix?
- matrixMul sample: computations and transfers
Impact of multiple users sharing the same GPU

- Measure performance of matrix multiplication when the remote GPU is only used by one user

- Measure performance of matrix multiplication when multiple users are sharing the same GPU

- Does performance decrease? With how many concurrent users does the decrease become more noticeable? What happens if we increase the size of the matrix?
Impact of multiple users sharing the same GPU

- Measure performance of matrix multiplication when the remote GPU is only used by one user

- Measure performance of matrix multiplication when multiple users are sharing the same GPU

- Does performance decreases? With how many concurrent users the decrease is more noticeable? What happens if we increase the size of the matrix?

- matrixMulCUBLAS sample: computations and transfers
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
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=5
export RCUDA_DEVICE_0=rcu16:0
export RCUDA_DEVICE_1=rcu16:1
export RCUDA_DEVICE_2=rcu16:2
export RCUDADEVICE_3=rcu16:3
export RCUDA_DEVICE_4=rcu15:0
```
Configure rCUDA for Multi-GPU:

```
export PATH=$PATH:/usr/local/cuda/bin
export LD_LIBRARY_PATH=$HOME/rCUDA/framework/rCUDA:$LD_LIBRARY_PATH
export RCUDA_DEVICE_COUNT=5
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=rcu15:0
```

Number of remote GPUs
Configure rCUDA for Multi-GPU:

- `export PATH=$PATH:/usr/local/cuda/bin`
- `export LD_LIBRARY_PATH=$HOME/rCUDA/framework/rCUDA1:$LD_LIBRARY_PATH`
- `export RCUDA_DEVICE_COUNT=5`
- `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=rcu15:0`
Configure rCUDA for Multi-GPU:

- Export PATH:
  ```bash
  export PATH=$PATH:/usr/local/cuda/bin
  ```
- Export LD_LIBRARY_PATH:
  ```bash
  export LD_LIBRARY_PATH=$HOME/rCUDA/framework/rCUDA1:$LD_LIBRARY_PATH
  ```
- Export RCUDA_DEVICE_COUNT:
  ```bash
  export RCUDA_DEVICE_COUNT=5
  ```
- Export RCUDA_DEVICE_0:
  ```bash
  export RCUDADEVICE_0=rcu16:0
  ```
- Export RCUDA_DEVICE_1:
  ```bash
  export RCUDADEVICE_1=rcu16:1
  ```
- Export RCUDA_DEVICE_2:
  ```bash
  export RCUDADEVICE_2=rcu16:2
  ```
- Export RCUDA_DEVICE_3:
  ```bash
  export RCUDADEVICE_3=rcu16:3
  ```
- Export RCUDA_DEVICE_4:
  ```bash
  export RCUDADEVICE_4=rcu15:0
  ```

- Check configuration by running `deviceQuery` sample
Outline

- Short intro to CUDA programming
- Cluster for testing
- Installing rCUDA
- Running some codes
- Questions & Answers
Get a free copy of rCUDA at
http://www.rcuda.net

@rcuda_r
Hands-on Session  
(Part III)

Carlos Reaño  
Technical University of Valencia  
Spain
rCUDA provides access to all GPUs in the cluster

rCUDA applications could be developed taking into account an scenario with multiple GPUs

cudaOpenMP: this sample demonstrates how to use OpenMP API to write an application for multiple GPUs

Study and understand cudaOpenMP sample, we will use it later
Adapt clock sample for multi-GPUs

- clock: this implements a standard parallel reduction

```
1 3 0 7 1 4 3 6
4 7 5 9
11 14
25
```

- Modify sample for taking advantage of multiple GPUs

- Use previous cudaOpenMP sample for adding the multi-GPU support
Divide work among available GPUs:
Adapt clock sample for multi-GPUs

Divide work among available GPUs:

Synchronize GPUs
Divide work among available GPUs:

Synchronize GPUs
Adapt clock sample for multi-GPUs

Divide work among available GPUs:

Synchronize GPUs

GPU1  GPU2  GPU3  GPU4

Synchronize GPUs

25
Divide work among available GPUs:

* Notice that the size of the vector to reduce must be bigger enough to improve performance with multiple GPUs!!!
Adapt vectorAdd sample for multi-GPUs

- vectorAdd: this is a very basic sample that implements element by element vector addition.

\[ C[i] = A[i] + B[i] \]

- Modify sample for taking advantage of multiple GPUs

- Each GPU can calculate a different set of elements, thus improving the performance

- Use previous cudaOpenMP sample for adding the multi-GPU support
Adapt vector Add sample for multi-GPUs

Divide work among available GPUs:

Vector A

\[ \begin{array}{cccc}
1 & 3 & 0 & 7 \\
\end{array} \]

+  

Vector B

\[ \begin{array}{cccc}
1 & 3 & 0 & 7 \\
\end{array} \]

=  

Vector C

\[ \begin{array}{cccc}
1 & 3 & 0 & 7 \\
\end{array} \]
Adapt vectorAdd sample for multi-GPUs

Divide work among available GPUs:

Vector A

\[
\begin{array}{cccc}
1 & 3 & 0 & 7 \\
1 & 4 & & 3 & 6 \\
\end{array}
\]

Vector B

\[
\begin{array}{cccc}
1 & 3 & 0 & 7 \\
1 & 4 & & 3 & 6 \\
\end{array}
\]

Vector C

\[
\begin{array}{cccc}
1 & 3 & 0 & 7 \\
1 & 4 & & 3 & 6 \\
\end{array}
\]

* Notice that the size of the vectors to add must be bigger enough to improve performance with multiple GPUs!!!
Get a free copy of rCUDA at http://www.rcuda.net

@rcuda_r