Remote CUDA (rCUDA)
Why?

- High performance clusters:
  - Fast interconnects
  - Hundreds of nodes, with multiple cores per node
  - Large storage systems
  - Hardware accelerators
    - Better performance-watt, performance-cost ratios for certain applications
Why?

- One Hw. accelerator per node?
  - Acquisition costs
  - Power consumption (↑ 20–30%), including cooling
  - Maintenance
  - Space
  - Not all applications can be accelerated

→ In general, waste of resources!
Contents

- Motivation
- Virtualization and CUDA
- rCUDA
  - Architecture
  - Example
  - Performance and power
  - Status and usability
  - Future work
Virtualization taxonomy

- Back-end (hardware facing) virtualization
  - Impossible: clients without direct access to HW
- Front-end (application facing) virtualization
  - Device emulation:
    - Sw. replication of the entire Hw. accelerator
    - Performance is not the target
    - Not appropriate for HPC → emulation overhead
  - API remoting:
    - Time-sharing real device
    - Client-server architecture
- Facilitates general-purpose computing on latest NVIDIA GPUs
- C + extensions: closer to non-GPU programmers than OpenGL, Cg,...
- GPU architecture viewed as a set of SIMD multiprocessors
- 2 APIs: Runtime (high-level) & driver (low-level)
- Device code has to be written as a *kernel*
- *Kernels:* are sent and executed on the GPU at runtime
- Data used by *kernels* has to be sent, too
- Managed by the driver
- Explicitly addressed by the programmer
rCUDA: Architecture

[Diagram showing the architecture of rCUDA, with layers for software and hardware, and connections between the client and server components.]
rCUDA: Architecture

Client-server

- **Client:**
  - Emulates local GPUs
  - Handles communication with the server

- **Server:**
  - Waits for requests
  - Executes GPU code (*kernel*)
  - Reports status
rCUDA: Example

- Matrix product (I):
  1. Socket opening + module sending
  2. Memory allocation
  3. Data transfer
rCUDA: Example

- Matrix product (II):
  
  ...  
  
  4. Kernel execution
  5. Results reception
  6. Memory release
  7. Socket closing
rCUDA: Performance and power

- Execution time (virtualized GPU vs. local CPU):

![Graph showing execution time comparison]
rCUDA: Performance and power

Overhead mostly caused by the network

Matrix product
rCUDA: Performance and power

- HPC interconnects:
rCUDA: Performance and power

- HPC interconnects:

![Graph showing performance comparison between local CPU, local GPU, GigE, 10GBe, InfiniBand, Myrinet, F-HT, and A-HT for different matrix dimensions. The graph indicates that remote GPU is faster than local CPU.]
rCUDA: Performance and power

- HPC interconnects:
rCUDA: Performance and power

- HPC interconnects:
rCUDA: Performance and power

- 100 node cluster (600 W/node)
- NVIDIA Tesla C1060 (193.58 W/GPU)
- 4 InfiniBand Mellanox MTS3600 switches in tree topology (316 W/switch)

<table>
<thead>
<tr>
<th>GPUs</th>
<th>Consumption</th>
<th>Savings</th>
<th>Ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>80.6 KW</td>
<td>0.0 W</td>
<td>0.0%</td>
</tr>
<tr>
<td>50</td>
<td>70.9 KW</td>
<td>9.6 KW</td>
<td>12.0%</td>
</tr>
<tr>
<td>25</td>
<td>66.1 KW</td>
<td>14.5 KW</td>
<td>18.0%</td>
</tr>
<tr>
<td>10</td>
<td>63.2 KW</td>
<td>17.4 KW</td>
<td>21.6%</td>
</tr>
</tbody>
</table>
rCUDA: Status and usability

- rCUDA R.C. 1.0 ready!
- Support for Linux OS (32 & 64 bits) on clients and servers
- CUDA Runtime API 2.3
  - Except OpenGL and Direct3D interoperability
rCUDA: Status and usability

- Limitations:
  - Virtualized devices do not offer zero copy
  - Lack of support for CUDA C extensions
    - Why? hidden and undocumented functions
    - Device and host codes compiled separately
    - Forces to use the plain C API
    - Unable to locate embedded code: No support for precompiled libraries (CUFFT, CUBLAS, etc.)
**Limitations (II):**

- Kernel call using CUDA C extensions:

  \[
  \text{kernel}<<<\text{blocks, threads}>>>(a, b, c)
  \]

- Same call using plain C API:

```c
#define ALIGN_UP(offset, align) (offset) = ((offset) + (align) - 1) & ~((align) - 1)

template<class T>
inline void setupArg(T arg, int *offst) {
    ALIGN_UP(*offst, __alignof(arg));
    cudaSetupArgument(&arg, sizeof(arg), *offst);
    *offst += sizeof(arg);
}
```

```c
cudaConfigureCall(blocks, threads);
int offset = 0;
setupArgument(&a);
setupArgument(&b);
setupArgument(&c);
cudaLaunch("kernel");
```
rCUDA: Status and usability

- Lines of code modified (or added) to avoid using CUDA C extensions in SDK examples:

<table>
<thead>
<tr>
<th>Example*</th>
<th>Original</th>
<th>Modified</th>
<th>Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>bandwidthTest</td>
<td>877</td>
<td>0</td>
<td>0.0%</td>
</tr>
<tr>
<td>binomialOptions</td>
<td>629</td>
<td>4</td>
<td>0.6%</td>
</tr>
<tr>
<td>dct8x8</td>
<td>3174</td>
<td>55</td>
<td>1.7%</td>
</tr>
<tr>
<td>simplePitchLinearTexture</td>
<td>274</td>
<td>32</td>
<td>11.7%</td>
</tr>
<tr>
<td>simpleStreams</td>
<td>216</td>
<td>20</td>
<td>9.3%</td>
</tr>
<tr>
<td>simpleTexture</td>
<td>228</td>
<td>22</td>
<td>9.6%</td>
</tr>
<tr>
<td><strong>Average</strong></td>
<td><strong>900</strong></td>
<td><strong>22</strong></td>
<td><strong>2.4%</strong></td>
</tr>
</tbody>
</table>
rCUDA: Status and usability

CUDA 2.3 SDK:
- 67 examples in total
- 51 ported to plain C API & successfully run w/ rCUDA
  - 10 modified to avoid using OpenGL interoperability
- 16 not feasible:
  - 11 use precompiled CUDA libraries
  - 5 directly use the Driver API
rCUDA: Status and usability

- Current possibilities:
  - HPC clusters: reduced number of GPUs
    - Energy & resource savings
  - Educational: remote GPU servers
  - Virtual Machines: accessing CUDA facilities on the physical machine
Future Work

- Multi-server functionalities:
  - Automatic discovery
  - Load balancing
- Integrate with 3D remoting acceleration works to enable OpenGL and Direct3D interoperability
- Optimized version for VMs (shared memory)
- Windows OS
- Full open API: OpenCL
rCUDA and Bull?

- Customized cluster: optimal configuration for particular needs (client)
  - How many GPUs
  - Distribution of GPUs
  - Network type
  - Which computations in remote GPU vs local CPU
rCUDA and Bull?

- Customized cluster: Given a fixed budget...
  - Present a more competitive offer in terms of energy consumption and performance
References
