An approach to provide remote access to GPU computational power

Federico Silla
Technical University of Valencia, Spain

Joint research effort
- GPU computing
- GPU computing scenarios
- The wish list
- Introduction to rCUDA
- rCUDA functionality
- Basic TCP/IP version
- InfiniBand version
- Current status
- GPU computing
- GPU computing scenarios
- The wish list
- Introduction to rCUDA
- rCUDA functionality
- Basic TCP/IP version
- InfiniBand version
- Current status
GPU computing: defines all the technological issues for using the GPU computational power for executing general purpose code.

- GPU computing has experienced remarkable growth in the last years.

<table>
<thead>
<tr>
<th>Rank</th>
<th>Site</th>
<th>Computer</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>RIKEN Advanced Institute for Computational Science (AICS) Japan</td>
<td>K computer, SPARC64 VIIIfx 2.0GHz, Tofu interconnect / 2011 Fujitsu</td>
</tr>
<tr>
<td>2</td>
<td>National Supercomputing Center in Tianjin, China</td>
<td>Tianhe-1A - NUDT TH MPP, X5670 2.93Ghz 6C, Nvidia GPU / FT-1000 8C / 2010 NUDT</td>
</tr>
<tr>
<td>3</td>
<td>DOE/SC/Oak Ridge National Laboratory United States</td>
<td>Jaguar - Cray XT5-HE Opteron 6-core 2.6 GHz / 2009 Cray Inc.</td>
</tr>
<tr>
<td>4</td>
<td>National Supercomputing Centre in Shenzhen (NSCS) China</td>
<td>Nebulae - Dawning TC3600 Blade, Intel X5650, Nvidia Tesla C2050 GPU / 2010 Dawning</td>
</tr>
<tr>
<td>5</td>
<td>GSIC Center, Tokyo Institute of Technology Japan</td>
<td>TSUBAME 2.0 - HP Proliant SL390s G7 Xeon 6C X5670, Nvidia GPU, Linux/Windows / 2010 NEC/HP</td>
</tr>
</tbody>
</table>
The basic building block is a node with 1 or more GPUs
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 (1-4)
- An interconnection network
Development tools have been introduced in order to ease the programming of the GPUs

Two main approaches in GPU computing development environments:

- CUDA $\rightarrow$ NVIDIA proprietary
- OpenCL $\rightarrow$ open standard
Basically CUDA and OpenCL have the same working scheme:

- **Compilation**: Separate CPU code from GPU code (GPU kernel)

- **Execution**:
  - Data transfers: **CPU and GPU memory spaces**
    1. **Before** GPU kernel execution: data from CPU memory space to GPU memory space
    2. **Computation**: Kernel execution
    3. **After** GPU kernel execution: results from GPU memory space to CPU memory space
- Time spent on data transfers may not be negligible

Influence of data transfers for SGEMM

- Pinned Memory
- Non-Pinned Memory

Matrix Size

Time devoted to data transfers (%)

Influence of data transfers for SGEMM

Time spent on data transfers may not be negligible
- GPU computing
- GPU computing scenarios
- The wish list
- Introduction to rCUDA
- rCUDA functionality
- Basic TCP/IP version
- InfiniBand version
- Current status
For the **right kind of code** the use of GPUs brings huge benefits in terms of performance and energy.

There must be **data parallelism** in the code: this is the only way to take benefit from the hundreds of processors in a GPU.

Different scenarios from the point of view of the application:

- Low level of data parallelism
- High level of data parallelism
- Moderate level of data parallelism
- Applications for multi-GPU computing
GPU computing scenarios

- **Low level of data parallelism**

  Regarding GPU computing?

  No GPU is needed, just proceed with the traditional HPC strategies

- **High level of data parallelism**

  Regarding GPU computing?

  Add one or more GPUs to every node in the system and rewrite applications to use them
GPU computing scenarios

- Moderate level of data parallelism: Application presents a data parallelism around [40%-80%]

Regarding GPU computing?
The GPUs in the system are used only for some parts of the application, remaining idle the rest of the time and, thus, wasting resources and energy.
**Applications for multi-GPU computing**

An application can use a large amount of GPUs in parallel

**Regarding GPU computing?**

The code running in a node can only access the GPUs in that node, but it would run faster if it could have access to more GPUs
GPU computing

GPU computing scenarios

The wish list

Introduction to rCUDA

rCUDA functionality

Basic TCP/IP version

InfiniBand version

Current status
CONCERNS:

- For applications with moderate levels of data parallelism, the GPUs in the cluster may be idle for long periods of time.
- Multi-GPU applications cannot make use of the tremendous GPU resources available across the cluster.
The wish list

- A way of addressing the first concern, the energy concern, is by reducing the number of GPUs present in the cluster.

NEW CONCERN:
- A lower amount of GPUs noticeably increases the difficulty of efficiently scheduling jobs (considering global CPU and GPU requirements).

25% GPUs → 25% initial expenses → 25% power consumption
The wish list

● A way of addressing the **new scheduling problem** is by reducing the number of GPUs present in the cluster **and sharing** the remaining ones among the CPU nodes.

● Additionally, by appropriately sharing GPUs, multi-GPU computing is also feasible.

● This would increase GPU utilization, also lowering power consumption, at the same time that initial acquisition costs are reduced.
The wish list

- Efficiently sharing GPUs across a cluster can be achieved by leveraging GPU virtualization
  - rCUDA ☺
  - gVirtuS ☹
  - vCUDA ☹
  - GViM ☹
  - VGPU ☹
  - GridCuda ☹

The only remote virtualization solution with CUDA 4 support
The wish list

- One step further:
  - enhancing the scheduling process so that GPU servers are put into low-power sleeping modes as soon as their acceleration features are not required.
The wish list

- Going even beyond:
  - consolidating GPUs into dedicated servers (no CPU power) and
  - allowing GPU task migration

TRUE GREEN GPU COMPUTING
The wish list

GPUs are disaggregated from CPUs and global schedulers are enhanced
- GPU computing
- GPU computing scenarios
- The wish list
- Introduction to rCUDA
- rCUDA functionality
- Basic TCP/IP version
- InfiniBand version
- Current status
A framework enabling that a CUDA-based application running in one node can access GPUs in other nodes

It is useful when you have:

- Moderate level of data parallelism
- Applications for multi GPU computing
- Moderate level of data parallelism

Adding GPUs at each node makes some GPUs to remain idle for long periods. This is a waste of money and energy.
• Moderate level of data parallelism

Add only the GPUs that are needed considering application requirements and their level of data parallelism and
• Moderate level of data parallelism

Add only the GPUs that are needed considering application requirements and their level of data parallelism and make all of them accessible from every node.
Applications for multi-GPU computing

From a given CPU it is only possible to access the corresponding GPUs in that very same node.
• Applications for multi-GPU computing

Make all GPUs accessible from every node
• Applications for multi-GPU computing

Make all GPUs accessible from every node and enable the access from a CPU to as many as required GPUs
Outline

- GPU computing
- GPU computing scenarios
- The wish list
- Introduction to rCUDA
- rCUDA functionality
- Basic TCP/IP version
- InfiniBand version
- Current status
rCUDA functionality

CUDA application

Application

CUDA driver + runtime

CUDA device
rCUDA functionality

Client side  ←  CUDA application  →  Server side

Application

CUDA driver + runtime

CUDA device

CUDA driver + runtime

CUDA device
rCUDA functionality

Client side

Application

rCUDA library

Network device

CUDA application

rCUDA daemon

Network device

Server side

CUDA device

CUDA driver + runtime
rCUDA functionality

Client side

Application

rCUDA library

Network device

CUDA application

rCUDA daemon

Network device

Server side

CUDA device

CUDA driver + runtime

HPC Advisory Council Switzerland Conference 2012
**rCUDA functionality**

Client side

- Application
- rCUDA library
- Network device

CUDA application

- rCUDA daemon
- CUDA driver + runtime
- Network device

Server side

- CUDA device
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 compatible with CUDA is rCUDA?
rCUDA functionality

CUDA Runtime API functions implemented by rCUDA

rCUDA does not provide support for graphic functions
NVIDIA CUDA C SDK code samples included in rCUDA SDK

SDK samples using graphic functions or driver API functions (not supported)
rCUDA functionality

- rCUDA does not support the CUDA extensions to C
- In order to execute a program within rCUDA, the CUDA extensions included in its code must be "unextended" to the plain C API

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

// Device code
__global__ void helloWorld(char* str) {
    // GPU tasks.
}

// Host code
int main(int argc, char **argv) {
    char h_str[] = "Hello..World!";
    // ...
    cudaMalloc((void**)&d_str, size);
    // copy the string to the device
    cudaMemcpy(d_str, h_str, size, cudaMemcpyHostToDevice);
    // launch the kernel
    helloWorld<<<BLOCKS, THREADS>>>(d_str);
    // retrieve the results from the device
    cudaMemcpy(h_str, d_str, size, cudaMemcpyDeviceToHost);
    // ...
    cudaFree(d_str);
    printf("%s\n", str);
    return 0;
}
```

NVCC inserts calls to undocumented CUDA functions
rCUDA functionality

CU2rCU tool

- CUDA Program (Device + Host Code)
- Converter
  - Clang Framework
  - Clang Driver
  - AST
  - CU2rCU
- NVCC
  - Device Code
- Host Code
rCUDA functionality

Graphs showing performance comparisons between original compilation and conversion + compilation for different CUDA SDK samples. The graphs illustrate time (s) measurements for various benchmarks.

Bottom graph displays time (s) for Lammps USER-CUDA Package, comparing original code and converted code with and without compilation.
• GPU computing
• GPU computing scenarios
• The wish list
• Introduction to rCUDA
• rCUDA functionality
• Basic TCP/IP version
• InfiniBand version
• Current status
Basic TCP/IP version

- Proof of concept
- It is a basic and **free** version to show the functionality
- Uses TCP/IP stack
- Runs over all TPC/IP networks
- Presents some non-negligible overhead due to the use of TCP/IP
Execution time for matrix-matrix multiplication (GEMM)

- Tesla C1060
- Intel Xeon E5410 2.33 Ghz
- 1Gb Ethernet

CPU

rCUDA

Matrix dimension

Time (sec)

0 10 20 30 40 50 60 70
512 2048 4096 6144 8192 10240 12288 14336 16384

Kernel execution

Data transfers

Basic TCP/IP version
Outline

- GPU computing
- GPU computing scenarios
- The wish list
- Introduction to rCUDA
- rCUDA functionality
- Basic TCP/IP version
- InfiniBand version
- Current status
Why InfiniBand version?

- InfiniBand is the most used HPC network
  - Low latency and high bandwidth

**Interconnect Family System Share**

- Gigabit Ethernet: 44.8%
- Infiniband: 41.8%
- Custom Interconnect
- Proprietary Network
- Cray Interconnect
- Myrinet

**NUMAlink**
- Fat Tree
- Quadrics
- SP Switch
- Mixed Network

Top500 November
• Same user level functionality
• Use of IB-Verbs
  • All TPC/IP stack overflow is out
• Bandwidth client to/from remote GPU near the peak InfiniBand network bandwidth
• Use of GPUDirect
  • Reduce the number of intra-node data movements
• Use of pipelined transfers
  • Overlap intra-node data movements and transfers
Bandwidth for a matrix of 4096 x 4096 single precision

IB peak bandwidth 2900 MB/sec
Execution time for a matrix-matrix product

- *Tcpu*
- *Tgpu*
- *TrCUDA*

- Tesla C2050
- Intel Xeon E5645
- QDR InfiniBand

HPC Advisory Council Switzerland Conference 2012
Overhead time for a matrix-matrix product

- % overhead gpu
- % overhead rcuda

Matrix dimension

- Tesla C2050
- Intel Xeon E5645
- QDR InfiniBand
Execution time for the LAMMPS application, in.eam input script scaled by a factor of 5 in the three dimensions

- rCUDA: 79%
- CUDA: 76%
- CPU: 100%

- Tesla C2050
- Intel Xeon E5520
- QDR InfiniBand
- GPU computing
- GPU computing scenarios
- The wish list
- Introduction to rCUDA
- rCUDA functionality
- Basic TCP/IP version
- InfiniBand version
- Current status
• Final rCUDA developments and debugging for supporting CUDA 4.1
• rCUDA integration into **SLURM** (Simple Linux Utility for Resource Management)
http://www.rcuda.net

Free binary distribution
(not for business, with limited performance)
  o Fedora
  o Scientific Linux
  o Ubuntu
  o OpenSuse

For business licensing you can contact the rCUDA team
http://www.rcuda.net