rCUDA 4: GPGPU as a service in HPC clusters

Rafael Mayo
Antonio J. Peña
Universitat Jaume I, Castelló (Spain)

Joint research effort
Outline

- GPU computing
- GPU computing scenarios
- The wish list
- Introduction to rCUDA
- rCUDA functionality
- rCUDA v3 (stable)
- rCUDA v4 (alpha)
- Current status
Outline

- GPU computing
- GPU computing scenarios
- The wish list
- Introduction to rCUDA
- rCUDA functionality
- rCUDA v3 (stable)
- rCUDA v4 (alpha)
- Current status
GPU computing

- 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.
• 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 → NVIDIA proprietary
- OpenCL → 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

![Graph showing influence of data transfers for SGEMM]

Matrix Size

Time devoted to data transfers (%)
Outline

- GPU computing
- GPU computing scenarios
- The wish list
- Introduction to rCUDA
- rCUDA functionality
- rCUDA v3 (stable)
- rCUDA v4 (alpha)
- 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
Outline

- GPU computing
- GPU computing scenarios
- The wish list
- Introduction to rCUDA
- rCUDA functionality
- rCUDA v3 (stable)
- rCUDA v4 (alpha)
- Current status
Current trend in GPU deployment

Each node one or more GPUs

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.

Deployment and energy costs!
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).
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

As far as we known, rCUDA is the only remote virtualization solution with CUDA 4 support.
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
Outline

- GPU computing
- GPU computing scenarios
- The wish list
- Introduction to rCUDA
- rCUDA functionality
- rCUDA v3 (stable)
- rCUDA v4 (alpha)
- 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
Introduction to rCUDA

- 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
- GPU computing
- GPU computing scenarios
- The wish list
- Introduction to rCUDA
- rCUDA functionality
- rCUDA v3 (stable)
- rCUDA v4 (alpha)
- Current status
rCUDA functionality

CUDA application

Application

CUDA libraries

CUDA device
rCUDA functionality

Client side ➞ CUDA application ➞ Server side

Application

CUDA driver + runtime

CUDA libraries

CUDA device

CUDA device
rCUDA functionality

Client side

Application

rCUDA library

Network device

CUDA application

rCUDA daemon

Network device

Server side

CUDA libraries

CUDA device
rCUDA functionality

Client side

CUDA application

Server side

Application

rCUDA library

Network device

Network device

rCUDA daemon

CUDA driver + runtime

CUDA device
rCUDA functionality

Client side

Application

rCUDA library

Network device

CUDA application

rCUDA daemon

Network device

CUDA driver + runtime

Server side

CUDA device
rCUDA functionality

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
Outline

- GPU computing
- GPU computing scenarios
- The wish list
- Introduction to rCUDA
- rCUDA functionality
- rCUDA v3 (stable)
- rCUDA v4 (alpha)
- Current status
rCUDA v3 (stable)

- Current stable release
- 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 (8 cores)
- Gigabit Ethernet

Outline

- GPU computing
- GPU computing scenarios
- The wish list
- Introduction to rCUDA
- rCUDA functionality
- rCUDA v3 (stable)
- rCUDA v4 (alpha)
- Current status
- Low-level InfiniBand support
  - InfiniBand is the most used HPC network
  - Low latency and high bandwidth
rCUDA v4 (alpha)

- Same user level functionality: CUDA Runtime
- Use of IB-Verbs
  - All TCP/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
- GPUDirect RDMA support is coming
- Remote GPU Transfers (GPUDirect)
- Remote GPU Transfers (GPUDirect)
- Remote GPU Transfers (GPUDirect)
- Remote GPU Transfers (GPUDirect)
Remote GPU Transfers (GPUDirect)
- Remote GPU Transfers (GPUDirect RDMA)
  - CUDA 5, upcoming
• Remote GPU Transfers (GPUDirect RDMA)
  • CUDA 5, upcoming
- Remote GPU Transfers (GPUDirect RDMA)
  - CUDA 5, upcoming
Bandwidth for a matrix of 4096 x 4096 single precision

IB peak bandwidth 2900 MB/sec

QDR InfiniBand

Execution time for a matrix-matrix product

- Tcpu
- Tgpu
- TrCUDA

Matrix dimension

- Tesla C2050
- Intel Xeon E5645
- QDR InfiniBand
Overhead time for a matrix-matrix product

% overhead gpu  % overhead rcuda

- Tesla C2050
- Intel Xeon E5645
- QDR InfiniBand

Matrix dimension

HPC Advisory Council Spain Conference 2012
Execution time for the LAMMPS application, in.eam input script scaled by a factor of 5 in the three dimensions

- Multi-thread capabilities (thread-safe)
- Multi-server capabilities
rCUDA v4 (alpha)

- Multi-thread and multi-server capabilities
GPUDirect 2.0 emulation is coming
• Multi-thread and multi-server performance

rCUDA v4 (alpha)

- 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.

```c
#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 result 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 v4 (alpha)

CU2rCU tool

Diagram showing the flow of the CU2rCU tool:
- CUDA Program (Device + Host Code) flows through the converter.
- NVCC then converts to Device Code.
- Another path shows the same process but with Clang Framework, Clang Driver, and AST.
- The final output is Host Code.
Outline

- GPU computing
- GPU computing scenarios
- The wish list
- Introduction to rCUDA
- rCUDA functionality
- rCUDA v3 (stable)
- rCUDA v4 (alpha)
- Current status
Current Status

How compatible with CUDA is rCUDA?
Current Status

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)
Current Status

- **rCUDA v3.2** stable release
  - Non-pipelined TCP/IP transfers
  - CUDA 4.2 support:
    - **Only** excluding graphics interoperability
  - Not thread-safe
  - Single rCUDA server per application
  - Only one GPU module per application allowed
  - CUDA C extensions unsupported
  - rCUDA 3.0a experimental build for MS Windows
rCUDA v4 alpha features (experimental)
- CUDA 4.2 (excluding graphics interoperability)
- TCP + low-level InfiniBand communications
- GPU & network pipelined transfers (GPUDirect)
- Multithread support for applications (thread-safe)
- Multiserver capabilities
- Support for CUDA C extensions (CU2rCU)
- Multiple GPU modules allowed
- MPI integration (MVAPICH + OpenMPI)
Current Status

- What’s coming?
  - CUDA 5
  - Inter-node GPUDirect 2 emulation
  - GPUDirect RDMA
  - rCUDA integration into SLURM
  - Full Runtime API support:
    - Graphics interoperability
  - rCUDA 4 for MS Windows
  - …
http://www.rcuda.net

Free binary distribution (rCUDA v3.2 + rCUDA v4.0a):
  o  Fedora
  o  Scientific Linux (RHEL compatible)
  o  Ubuntu
  o  OpenSuse
  o  MS Windows
rCUDA Team

Antonio Peña
Carlos Reaño
Federico Silla
José Duato

Adrian Castelló
Rafael Mayo
Enrique S. Quintana-Ortí

Parallel Architectures Group

High Performance Computing and Architectures Group

Thanks to Mellanox and AIC for their kindly support to this work