Remote CUDA (rCUDA)

UNIVERSIDAD POLITECNICA DE VALENCIA
GAP Parallel Architectures Group
UNIVERSITAT JAUME·I
HPC&A High Performance Computing and Architectures

Why?

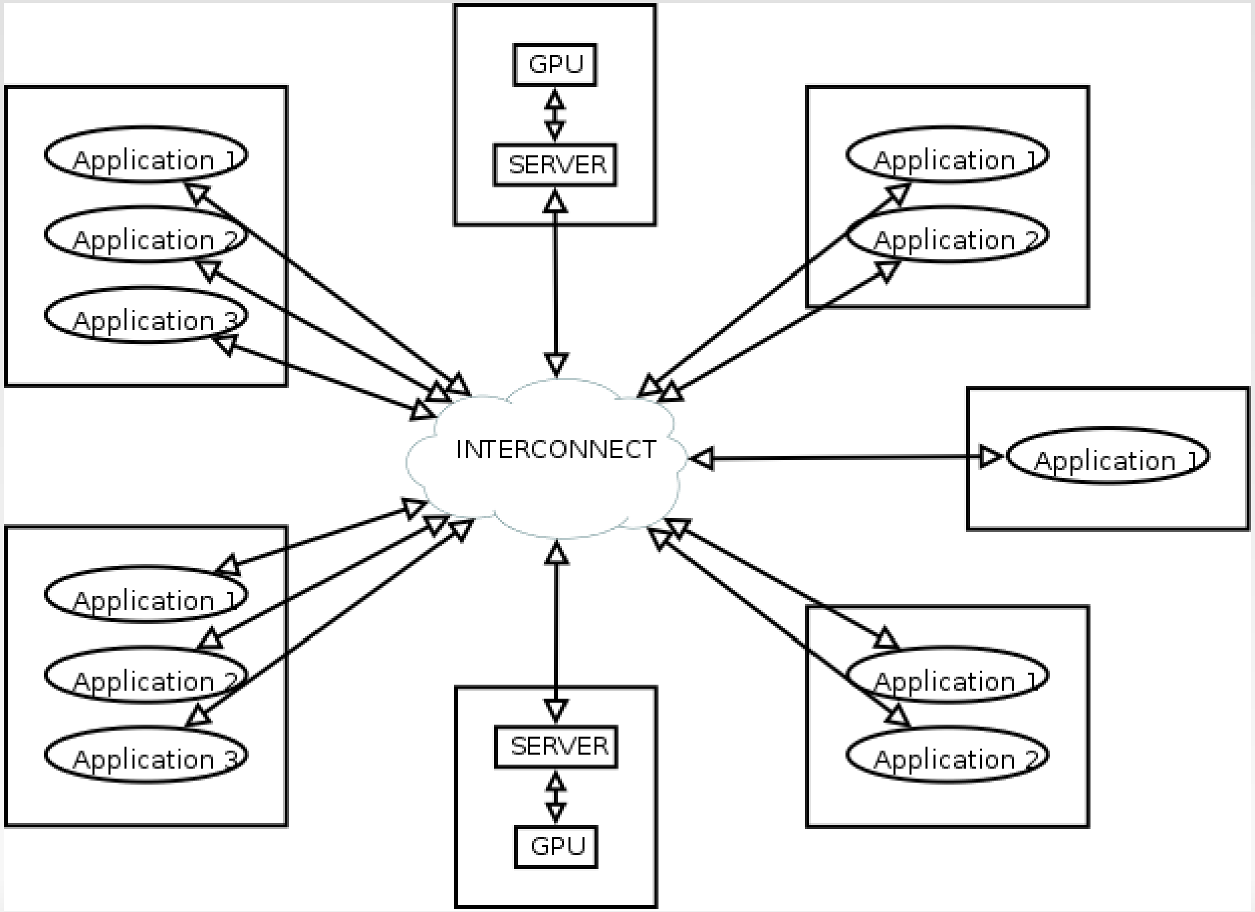
- High performance clusters:
  - Fast interconnects
  - Hundreds of nodes, with multiple cores per node
  - Large storage systems
  - Hardware accelerators
    - Better performance-watt, performance-cost ratios for certain applications

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!

rCUDA


GPU
SERVER
Application 1
Application 2
Application 3
Application 1
Application 2
INTERCONNECT
Application 1
Application 1
Application 2
Application 3
SERVER
GPU
Application 1
Application 2

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

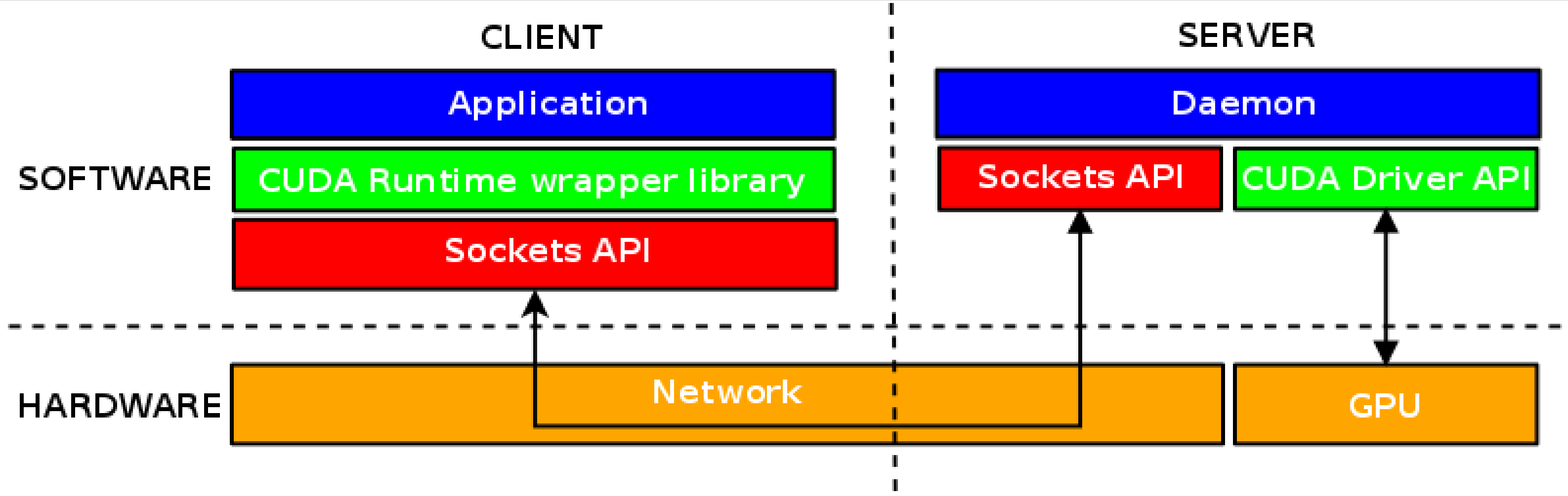
CUDA

- 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)

CUDA

- 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



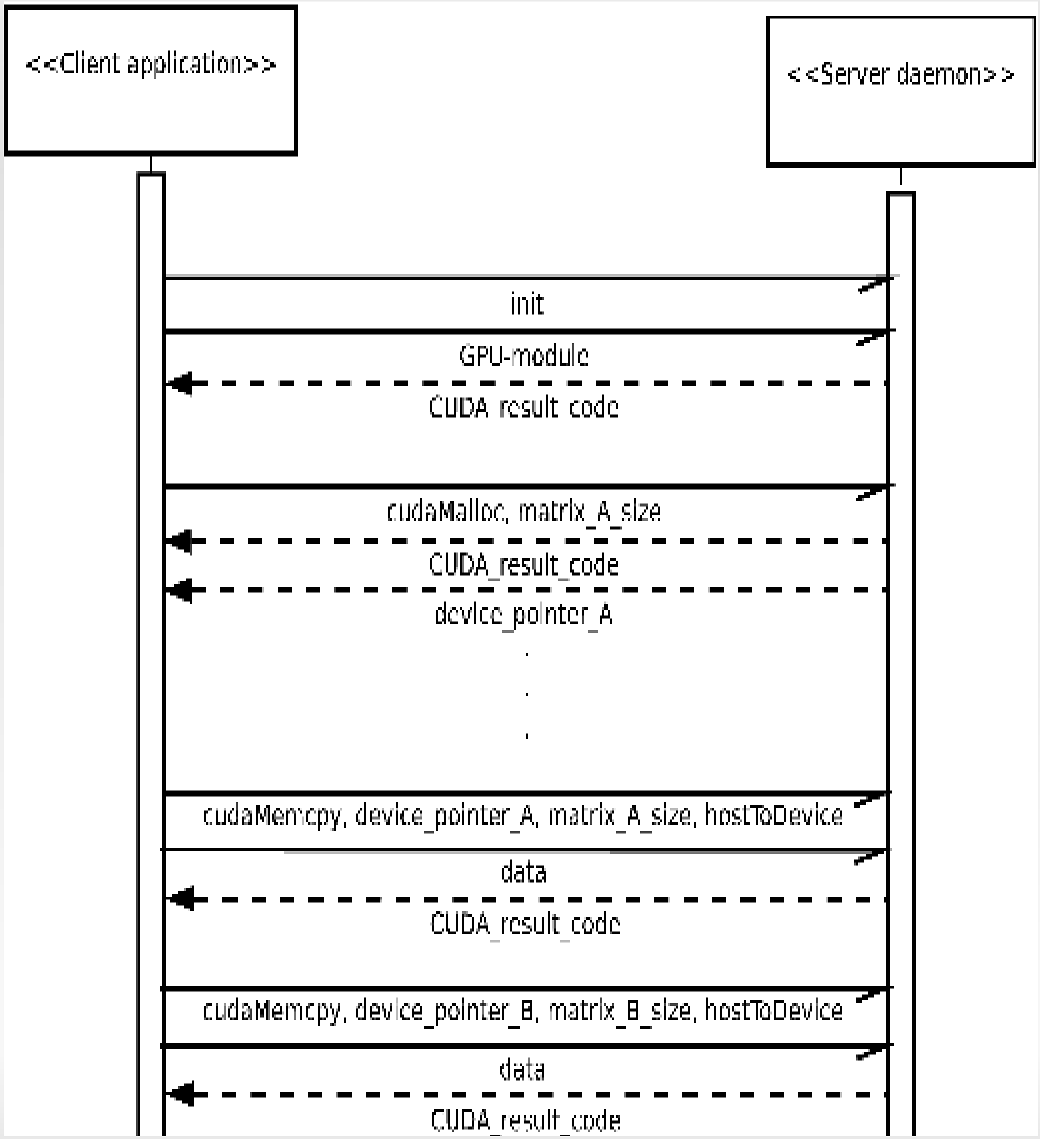
rCUDA: Architecture

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

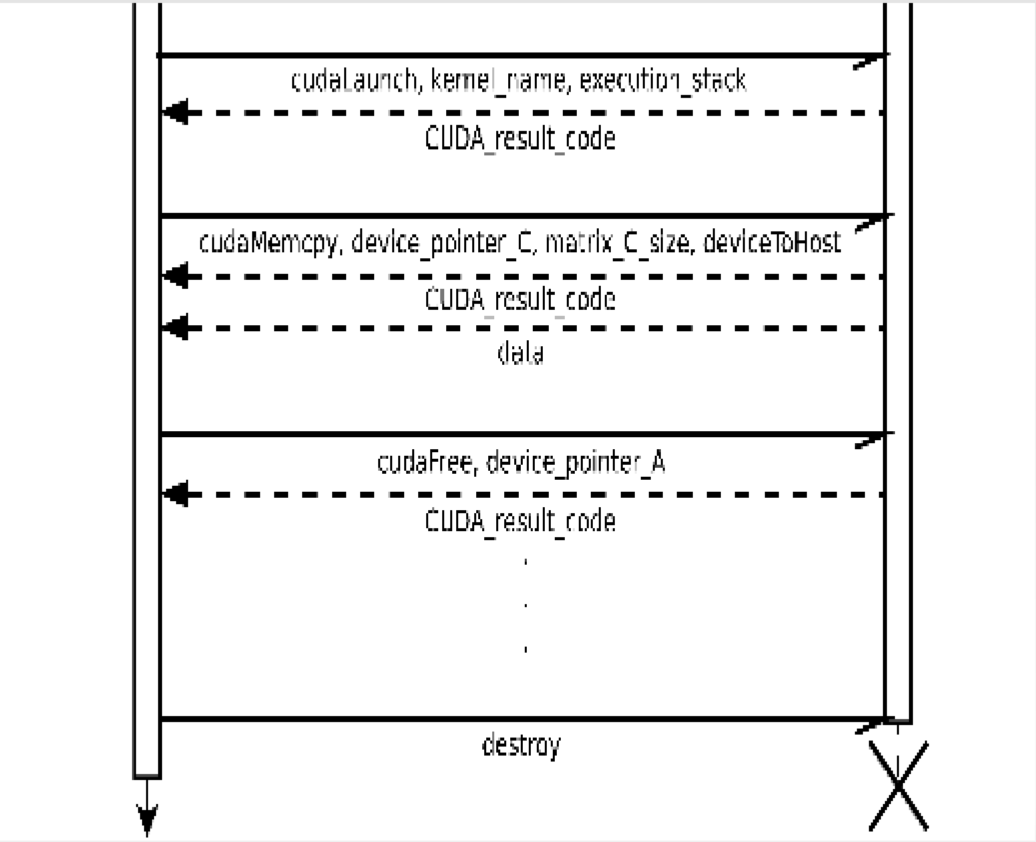


rCUDA: Example

- Matrix product (II):

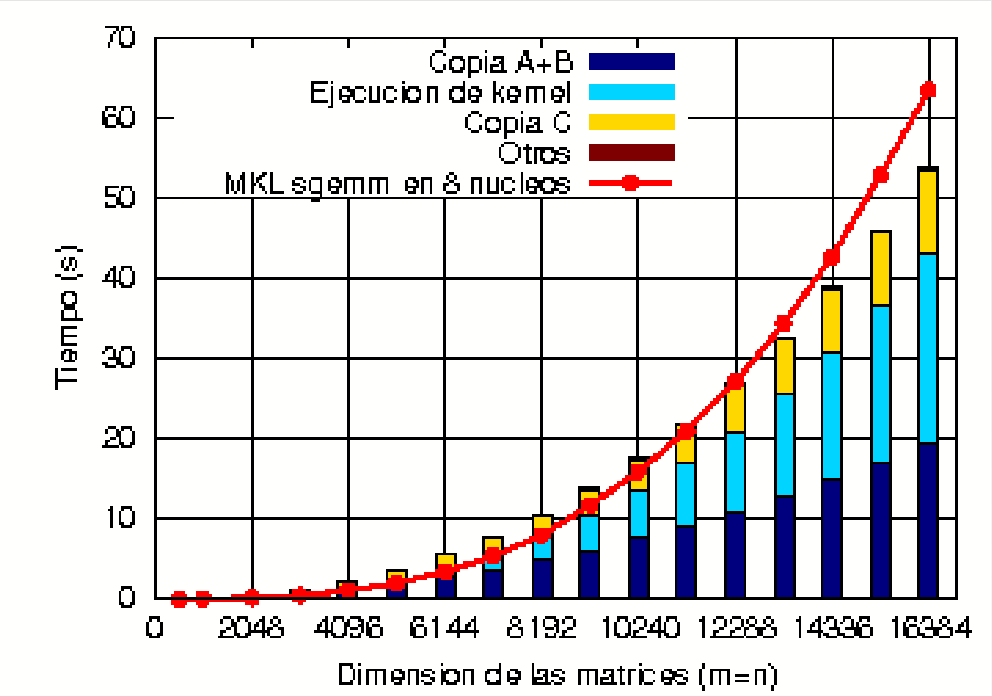
  ...

  4. Kernel execution
  5. Results reception
  6. Memory release
  7. Socket closing



rCUDA: Performance and power

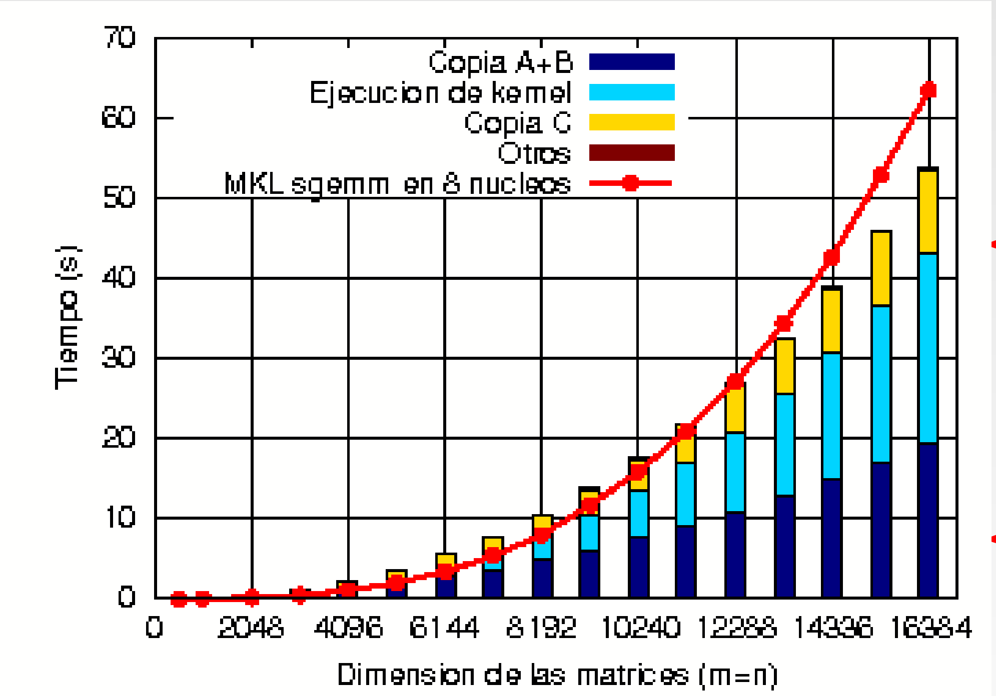
- Execution time (virtualized GPU vs. local CPU):



rCUDA: Performance and power

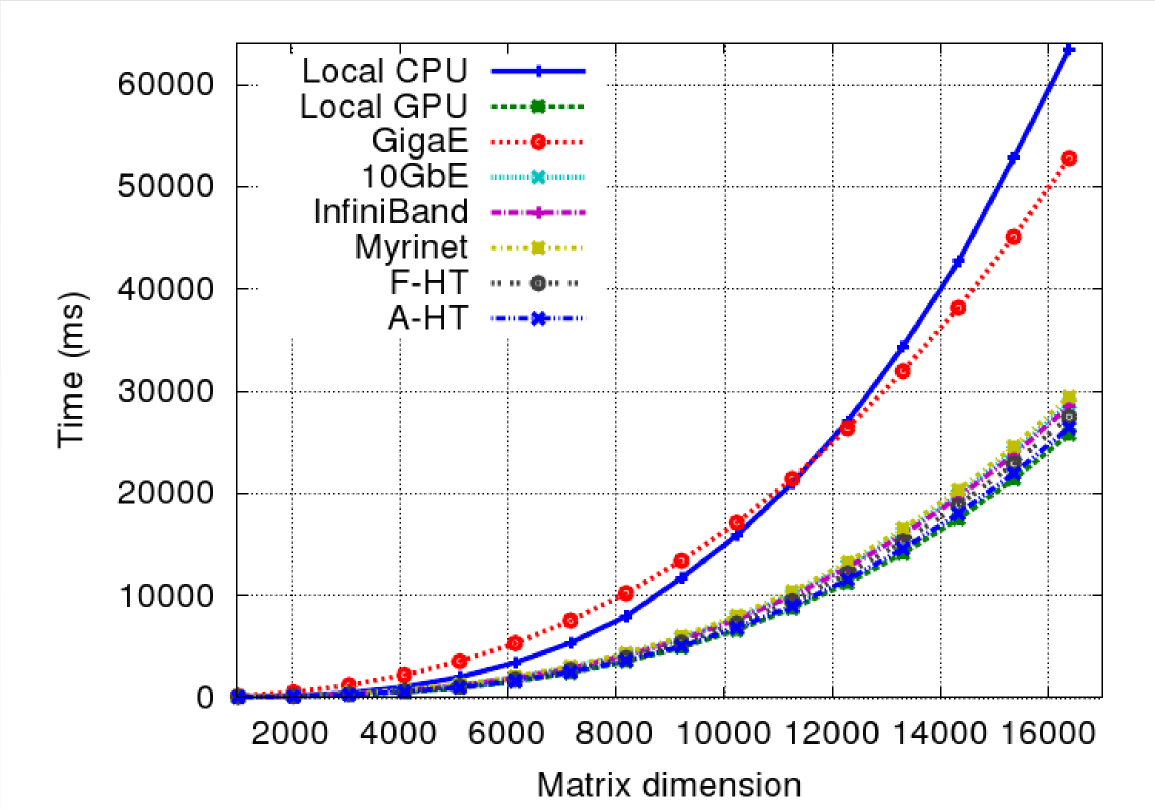
Overhead mostly caused by the network

Matrix product



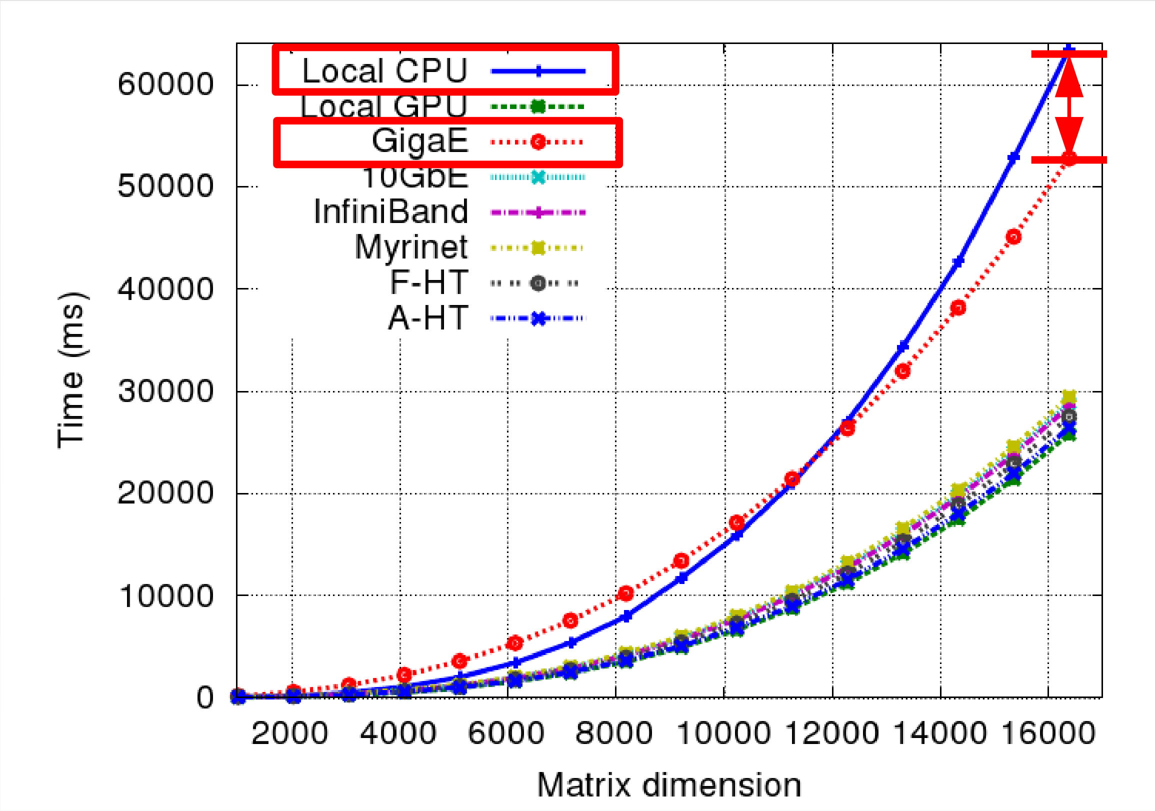
rCUDA: Performance and power

- HPC interconnects:



rCUDA: Performance and power

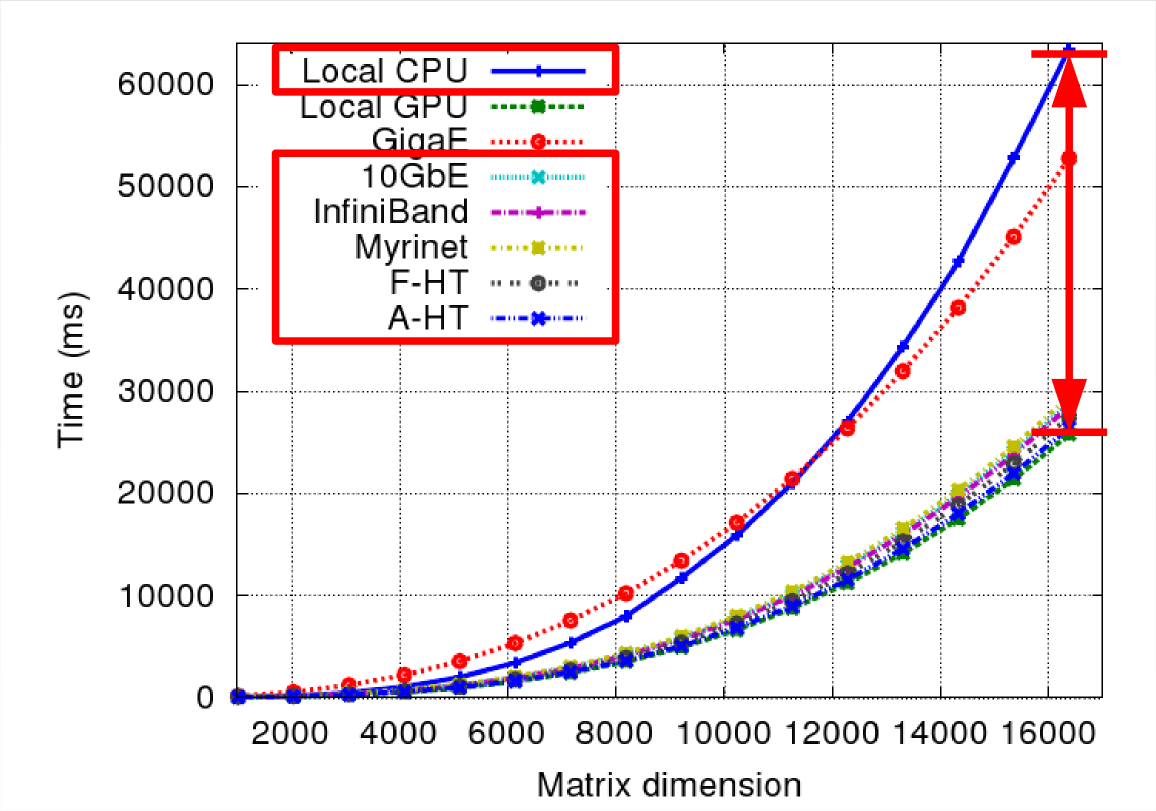
- HPC interconnects:



Remote GPU faster than local CPU

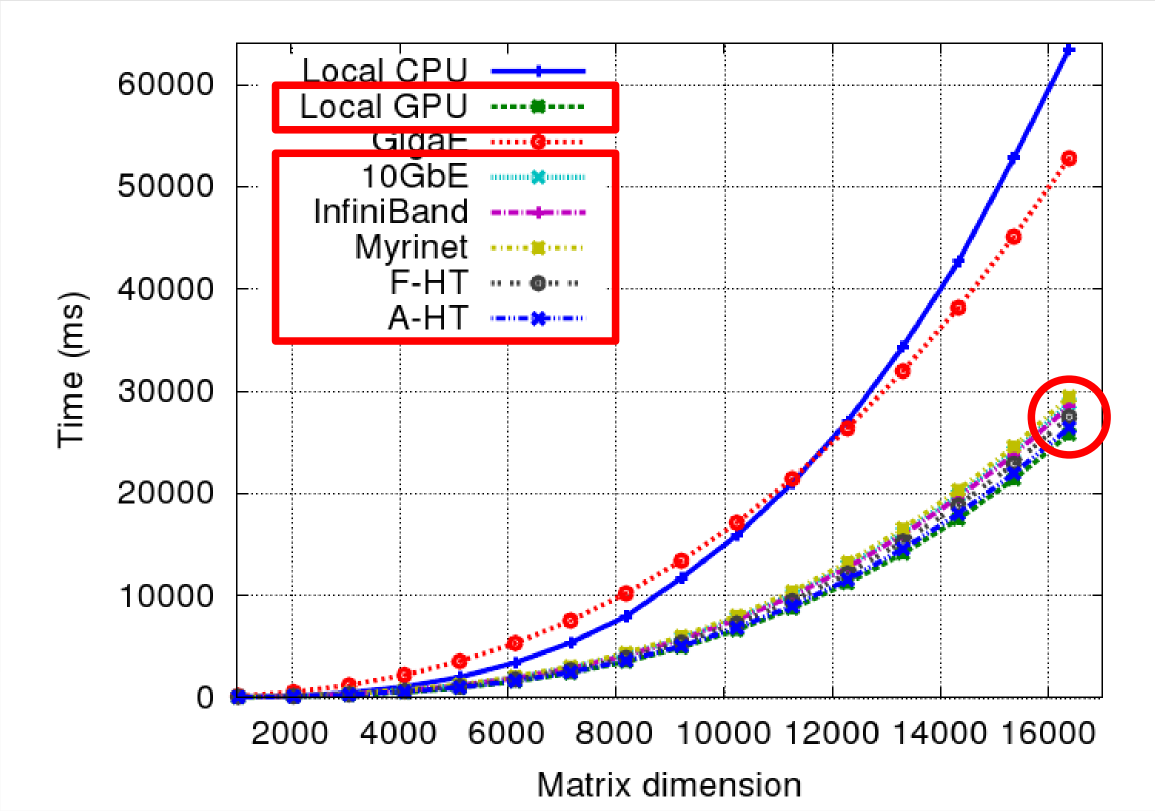
rCUDA: Performance and power

- 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)

GPUs	Consumption	Savings	Ratio
100	80.6 KW	0.0 W	0.0%
50	70.9 KW	9.6 KW	12.0%
25	66.1 KW	14.5 KW	18.0%
10	63.2 KW	17.4 KW	21.6%

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.)

rCUDA: Status and usability

- Limitations (II):
  - Kernel call using CUDA C extensions:

    kernel<<<blocks, threads>>>(a, b, c)

  - Same call using plain C API:

```
#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);
    }
```

Aux. code

```
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:

Example*	**Original**	**Modified**	**Rate**
bandwidthTest	877	0	0.0%
binomialOptions	629	4	0.6%
dct8x8	3174	55	1.7%
simplePitchLinearTexture	274	32	11.7%
simpleStreams	216	20	9.3%
simpleTexture	228	22	9.6%
Average	**900**	**22**	**2.4%**

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

- J. Duato, F. D. Igual, R. Mayo, A. J. Peña, E. S. Quintana-Ortí, and F. Silla, ”An efficient implementation of GPU virtualization in high performance clusters”, in *4th Workshop on Virtualization in High Performance Cloud Computing*, 2009
- J. Duato, A. J. Peña, F. Silla, R. Mayo, and E. S. Quintana-Ortí. ”Modeling the CUDA remoting virtualization behaviour in high performance networks”, in *First Workshop on Language, Compiler, and Architecture Support for GPGPU*, 2010