APENet+: a 3D Toroidal Network Enabling PetaFlops Scale Lattice QCD Simulations on Commodity Clusters

Roberto Ammendola

Istituto Nazionale di Fisica Nucleare, Sezione Roma Tor Vergata

Lattice 2010, Villasimius – 18 June 2010
What is happening to the hardware?

The cost of a FPU is rapidly decreasing with the GPUs:

<table>
<thead>
<tr>
<th></th>
<th>Xeon X5670</th>
<th>Opteron 8439</th>
<th>ATI HD 5870</th>
<th>Tesla C1060</th>
<th>Tesla C2070</th>
</tr>
</thead>
<tbody>
<tr>
<td># of cores</td>
<td>6</td>
<td>6</td>
<td>1600</td>
<td>240</td>
<td>448</td>
</tr>
<tr>
<td>SP GFlops</td>
<td>140</td>
<td>134</td>
<td>2720</td>
<td>933</td>
<td>1030</td>
</tr>
<tr>
<td>DP GFlops</td>
<td>70</td>
<td>67</td>
<td>544</td>
<td>78</td>
<td>515</td>
</tr>
<tr>
<td>GiB of Mem</td>
<td>-</td>
<td>-</td>
<td>1</td>
<td>4</td>
<td>6</td>
</tr>
<tr>
<td>TDP (Watt)</td>
<td>95</td>
<td>105</td>
<td>188</td>
<td>188</td>
<td>247</td>
</tr>
<tr>
<td>Price</td>
<td>1600</td>
<td>2000</td>
<td>400</td>
<td>1500</td>
<td>&lt; 2000</td>
</tr>
<tr>
<td>€ / GFlops</td>
<td>23</td>
<td>30</td>
<td>1.4</td>
<td>19</td>
<td>&lt; 4</td>
</tr>
</tbody>
</table>

GPU technology cannot be considered a transient, it should be exploited. Definitive work: Quda lib [M.A.Clark et al. 2009]:

- Double, Single, Half-precision
- Half-prec solver with reliable updates > 100Gflops
- MIT/X11 Open Source License
How many GPUs for QCD?

Raw estimate of memory footprint:
- Full solver in GPU
- Gauge field + 15 fermion fields
- No symmetry tricks
- No half precision tricks

<table>
<thead>
<tr>
<th>Lattice Size</th>
<th>SP (GiB)</th>
<th>DP (GiB)</th>
<th># ATI HD 5870</th>
<th># Tesla C1060</th>
<th># Tesla C2070</th>
</tr>
</thead>
<tbody>
<tr>
<td>$24^3 \times 48$</td>
<td>1</td>
<td>2.1</td>
<td>3</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>$32^3 \times 64$</td>
<td>3.3</td>
<td>6.7</td>
<td>4 – 8</td>
<td>2</td>
<td>1 – 2</td>
</tr>
<tr>
<td>$48^3 \times 96$</td>
<td>17</td>
<td>34</td>
<td>17 – 35</td>
<td>5 – 9</td>
<td>3 – 6</td>
</tr>
<tr>
<td>$64^3 \times 128$</td>
<td>54</td>
<td>108</td>
<td>55 – 110</td>
<td>14 – 28</td>
<td>9 – 18</td>
</tr>
</tbody>
</table>
If one GPU is not enough

Multi-GPU, the Fastra II approach:
- Stick 13 GPUs together
- 12TFLOPS @ 2KW
- CPU threads feed GPU kernels
- Embarrassingly parallel
- Full problem fits
- Enjoy the warm weather
multi-GPUs need scaling!

Seems easy:

- 1-2-4 GPUs in 1-2U system (or buy Tesla M1060)
- Stack many
- Add an interconnect (IB, Myrinet 10G, custom) & plug accurately
- *Simply* write your program in C+MPI+OpenMP+CUDA/OpenCL
A Commercial solution: InfiniBand

- switched network (e.g. 36 port switch 6000 €)
- 40 Gb Link technology

How do IB switches scale to big numbers?
- fixed maximum number of port (up to 648 port, 29 U, 300k €)
- they don’t have a backbone or uplink connection...
- fat tree networks increase latency and reduce bandwidth
APENet is a 3D network of point-to-point links with toroidal topology.

- Each computing node has 6 bi-directional full-duplex communication channels
- Computing nodes are arranged in a 3D cubic mesh
- Data is transmitted in packets which are routed to the destination node
- Lightweight low level protocol
- Wormhole routing
- Dimension ordered routing algorithm
- 2 Virtual Channels per receiving channel to prevent deadlocks
Scaling with GPUs

APENet Production Installation (2006)

- 128 nodes cluster with full MPI compliance
- $4 \times 4 \times 8$ APENet network
- 1740.8 GFlops ($256 \times 2 \times 3.4$) Peak Performance
- 1235 GFlops (70.9%) Sustained with HPL
new Altera Stratix IV device
6 Remote Channels based on QSFP+ technology (up to 34 Gbps with 4 bonded Altera embedded transceivers)
Host Connection based on PCIe x8 v2.0 (4 GB/s)
SO-DIMM DDR3 socket (512 MB – 2 GB)
USB re-programming
1U Chassis Compliance
APENet+: a network for PetaFlops Scale LQCD Simulations
Waiting for final hardware to be produced, we developed and produced (in May) a daughter card with 3 links:

- validating PCIe communication
- validating firmware
- running BER and electrical characterization tests on serial links up to 34 Gbps
- MPI-level performance tests will be performed as soon as possible.
GPU Specific Optimizations

The key issue of tying together many GPUs will be the latency of data transfer between the memory of a GPU to another.

- CPU Memory zero-copy via native RDMA primitives: Send, Put and Get
- GPU Memory zero-copy is harder, but we'll try
- GPU-initiated network communications, i.e. MPI_Send in GPU code
The key issue of tying together many GPUs will be the latency of data transfer between the memory of a GPU to another.

- CPU Memory zero-copy via native RDMA primitives: Send, Put and Get
- GPU Memory zero-copy is harder, but we’ll try
- GPU-initiated network communications, i.e. MPI_Send in GPU code
The key issue of tying together many GPUs will be the latency of data transfer between the memory of a GPU to another.

- CPU Memory zero-copy via native RDMA primitives: Send, Put and Get
- GPU Memory zero-copy is harder, but we’ll try
- GPU-initiated network communications, i.e. MPI_Send in GPU code
GPU Specific Optimizations

The key issue of tying together many GPUs will be the latency of data transfer between the memory of a GPU to another.

- CPU Memory zero-copy via native RDMA primitives: Send, Put and Get
- GPU Memory zero-copy is harder, but we’ll try
- GPU-initiated network communications, i.e. MPI_Send in GPU code
The key issue of tying together many GPUs will be the latency of data transfer between the memory of a GPU to another.

- CPU Memory zero-copy via native RDMA primitives: Send, Put and Get
- GPU Memory zero-copy is harder, but we’ll try
- GPU-initiated network communications, i.e. MPI_Send in GPU code
The key issue of tying together many GPUs will be the latency of data transfer between the memory of a GPU to another.

- CPU Memory zero-copy via native RDMA primitives: Send, Put and Get
- GPU Memory zero-copy is harder, but we’ll try
- GPU-initiated network communications, i.e. MPI_Send in GPU code

Single Node

- GPU calculate
GPU Specific Optimizations

The key issue of tying together many GPUs will be the latency of data transfer between the memory of a GPU to another.

- CPU Memory zero-copy via native RDMA primitives: Send, Put and Get
- GPU Memory zero-copy is harder, but we’ll try
- GPU-initiated network communications, i.e. MPI_Send in GPU code

![Diagram of APENet+](image-url)
The key issue of tying together many GPUs will be the latency of data transfer between the memory of a GPU to another.

- CPU Memory zero-copy via native RDMA primitives: Send, Put and Get
- GPU Memory zero-copy is harder, but we'll try
- GPU-initiated network communications, i.e. MPI_Send in GPU code
GPU Specific Optimizations

The key issue of tying together many GPUs will be the latency of data transfer between the memory of a GPU to another.

- CPU Memory zero-copy via native RDMA primitives: Send, Put and Get
- GPU Memory zero-copy is harder, but we’ll try
- GPU-initiated network communications, i.e. MPI_Send in GPU code

Single Node

- data back to Memory
The key issue of tying together many GPUs will be the latency of data transfer between the memory of a GPU to another.

- CPU Memory zero-copy via native RDMA primitives: Send, Put and Get
- GPU Memory zero-copy is harder, but we’ll try
- GPU-initiated network communications, i.e. MPI_Send in GPU code
GPU Specific Optimizations

The key issue of tying together many GPUs will be the latency of data transfer between the memory of a GPU to another.

- **CPU Memory zero-copy via native RDMA primitives:** Send, Put and Get
- **GPU Memory zero-copy is harder, but we’ll try**
- **GPU-initiated network communications, i.e. MPI_Send in GPU code**
The key issue of tying together many GPUs will be the latency of data transfer between the memory of a GPU to another.

- CPU Memory zero-copy via native RDMA primitives: Send, Put and Get
- GPU Memory zero-copy is harder, but we’ll try
- GPU-initiated network communications, i.e. MPI_Send in GPU code
The key issue of tying together many GPUs will be the latency of data transfer between the memory of a GPU to another.

- CPU Memory zero-copy via native RDMA primitives: Send, Put and Get
- GPU Memory zero-copy is harder, but we'll try
- GPU-initiated network communications, i.e. MPI_Send in GPU code

Many Nodes without optimizations

- data move to GPUs
GPU Specific Optimizations

The key issue of tying together many GPUs will be the latency of data transfer between the memory of a GPU to another.

- CPU Memory zero-copy via native RDMA primitives: Send, Put and Get
- GPU Memory zero-copy is harder, but we’ll try
- GPU-initiated network communications, i.e. MPI_Send in GPU code

Many Nodes without optimizations

- data move to GPUs
GPU Specific Optimizations

The key issue of tying together many GPUs will be the latency of data transfer between the memory of a GPU to another.

- CPU Memory zero-copy via native RDMA primitives: Send, Put and Get
- GPU Memory zero-copy is harder, but we’ll try
- GPU-initiated network communications, i.e. MPI_Send in GPU code
GPU Specific Optimizations

The key issue of tying together many GPUs will be the latency of data transfer between the memory of a GPU to another.

- CPU Memory zero-copy via native RDMA primitives: Send, Put and Get
- GPU Memory zero-copy is harder, but we’ll try
- GPU-initiated network communications, i.e. MPI_Send in GPU code

![Diagram of GPU network architecture](image.png)
GPU Specific Optimizations

The key issue of tying together many GPUs will be the latency of data transfer between the memory of a GPU to another.

- CPU Memory zero-copy via native RDMA primitives: Send, Put and Get
- GPU Memory zero-copy is harder, but we’ll try
- GPU-initiated network communications, i.e. MPI_Send in GPU code
The key issue of tying together many GPUs will be the latency of data transfer between the memory of a GPU to another.

- CPU Memory zero-copy via native RDMA primitives: Send, Put and Get
- GPU Memory zero-copy is harder, but we’ll try
- GPU-initiated network communications, i.e. MPI_Send in GPU code
GPU Specific Optimizations

The key issue of tying together many GPUs will be the latency of data transfer between the memory of a GPU to another.

- CPU Memory zero-copy via native RDMA primitives: Send, Put and Get
- GPU Memory zero-copy is harder, but we’ll try
- GPU-initiated network communications, i.e. MPI_Send in GPU code
The key issue of tying together many GPUs will be the latency of data transfer between the memory of a GPU to another.

- CPU Memory zero-copy via native RDMA primitives: Send, Put and Get
- GPU Memory zero-copy is harder, but we’ll try
- GPU-initiated network communications, i.e. MPI_Send in GPU code
The key issue of tying together many GPUs will be the latency of data transfer between the memory of a GPU to another.

- CPU Memory zero-copy via native RDMA primitives: Send, Put and Get
- GPU Memory zero-copy is harder, but we’ll try
- GPU-initiated network communications, i.e. MPI_Send in GPU code

Many Nodes without optimizations

- data exchange
The key issue of tying together many GPUs will be the latency of data transfer between the memory of a GPU to another.

- CPU Memory zero-copy via native RDMA primitives: Send, Put and Get
- GPU Memory zero-copy is harder, but we'll try
- GPU-initiated network communications, i.e. MPI_Send in GPU code
The key issue of tying together many GPUs will be the latency of data transfer between the memory of a GPU to another.

- CPU Memory zero-copy via native RDMA primitives: Send, Put and Get
- GPU Memory zero-copy is harder, but we’ll try
- GPU-initiated network communications, i.e. MPI_Send in GPU code
GPU Specific Optimizations

The key issue of tying together many GPUs will be the latency of data transfer between the memory of a GPU to another.

- CPU Memory zero-copy via native RDMA primitives: Send, Put and Get
- GPU Memory zero-copy is harder, but we’ll try
- GPU-initiated network communications, i.e. MPI_Send in GPU code
The key issue of tying together many GPUs will be the latency of data transfer between the memory of a GPU to another.

- **CPU Memory zero-copy via native RDMA primitives**: Send, Put and Get
- **GPU Memory zero-copy** is harder, but we’ll try
- **GPU-initiated network communications**, i.e. `MPI_Send` in GPU code
The key issue of tying together many GPUs will be the latency of data transfer between the memory of a GPU to another.

- CPU Memory zero-copy via native RDMA primitives: Send, Put and Get
- GPU Memory zero-copy is harder, but we’ll try
- GPU-initiated network communications, i.e. MPI_Send in GPU code

Many Nodes without optimizations

- data exchange
The key issue of tying together many GPUs will be the latency of data transfer between the memory of a GPU to another.

- CPU Memory zero-copy via native RDMA primitives: Send, Put and Get
- GPU Memory zero-copy is harder, but we’ll try
- GPU-initiated network communications, i.e. MPI_Send in GPU code
The key issue of tying together many GPUs will be the latency of data transfer between the
memory of a GPU to another.

- CPU Memory zero-copy via native RDMA primitives: Send, Put and Get
- GPU Memory zero-copy is harder, but we’ll try
- GPU-initiated network communications, i.e. MPI_Send in GPU code
The key issue of tying together many GPUs will be the latency of data transfer between the memory of a GPU to another.

- CPU Memory zero-copy via native RDMA primitives: Send, Put and Get
- GPU Memory zero-copy is harder, but we’ll try
- GPU-initiated network communications, i.e. MPI_Send in GPU code
Scaling with GPUs  APENet+  Conclusions

**GPU Specific Optimizations**

The key issue of tying together many GPUs will be the latency of data transfer between the memory of a GPU to another.

- CPU Memory zero-copy via native RDMA primitives: Send, Put and Get
- GPU Memory zero-copy is harder, but we’ll try
- GPU-initiated network communications, i.e. MPI_Send in GPU code

![Diagram of data exchange](image_url)
The key issue of tying together many GPUs will be the latency of data transfer between the memory of a GPU to another.

- CPU Memory zero-copy via native RDMA primitives: Send, Put and Get
- GPU Memory zero-copy is harder, but we’ll try
- GPU-initiated network communications, i.e. MPI_Send in GPU code
The key issue of tying together many GPUs will be the latency of data transfer between the memory of a GPU to another.

- CPU Memory zero-copy via native RDMA primitives: Send, Put and Get
- GPU Memory zero-copy is harder, but we’ll try
- GPU-initiated network communications, i.e. MPI_Send in GPU code

Many Nodes without optimizations

- data exchange
GPU Specific Optimizations

The key issue of tying together many GPUs will be the latency of data transfer between the memory of a GPU to another.

- CPU Memory zero-copy via native RDMA primitives: Send, Put and Get
- GPU Memory zero-copy is harder, but we’ll try
- GPU-initiated network communications, i.e. MPI_Send in GPU code
The key issue of tying together many GPUs will be the latency of data transfer between the memory of a GPU to another.

- CPU Memory zero-copy via native RDMA primitives: Send, Put and Get
- GPU Memory zero-copy is harder, but we’ll try
- GPU-initiated network communications, i.e. MPI_Send in GPU code

Many Nodes with GPU-initiated network communications

- program starts
GPU Specific Optimizations

The key issue of tying together many GPUs will be the latency of data transfer between the memory of a GPU to another.

- CPU Memory zero-copy via native RDMA primitives: Send, Put and Get
- GPU Memory zero-copy is harder, but we’ll try
- GPU-initiated network communications, i.e. MPI_Send in GPU code

Many Nodes with GPU-initiated network communications
- data move to GPUs
The key issue of tying together many GPUs will be the latency of data transfer between the memory of a GPU to another.

- CPU Memory zero-copy via native RDMA primitives: Send, Put and Get
- GPU Memory zero-copy is harder, but we’ll try
- GPU-initiated network communications, i.e. MPI_Send in GPU code
GPU Specific Optimizations

The key issue of tying together many GPUs will be the latency of data transfer between the memory of a GPU to another.

- CPU Memory zero-copy via native RDMA primitives: Send, Put and Get
- GPU Memory zero-copy is harder, but we’ll try
- GPU-initiated network communications, i.e. `MPI_Send` in GPU code

Many Nodes with GPU-initiated network communications

- data move to GPUs

R. Ammendola
APENet+: a network for PetaFlops Scale LQCD Simulations
The key issue of tying together many GPUs will be the latency of data transfer between the memory of a GPU to another.

- CPU Memory zero-copy via native RDMA primitives: Send, Put and Get
- GPU Memory zero-copy is harder, but we’ll try
- GPU-initiated network communications, i.e. MPI_Send in GPU code
GPU Specific Optimizations

The key issue of tying together many GPUs will be the latency of data transfer between the memory of a GPU to another.

- CPU Memory zero-copy via native RDMA primitives: Send, Put and Get
- GPU Memory zero-copy is harder, but we’ll try
- GPU-initiated network communications, i.e. MPI_Send in GPU code
Scaling with GPUs  APENet+  Conclusions

GPU Specific Optimizations

The key issue of tying together many GPUs will be the latency of data transfer between the memory of a GPU to another.

- CPU Memory zero-copy via native RDMA primitives: Send, Put and Get
- GPU Memory zero-copy is harder, but we’ll try
- GPU-initiated network communications, i.e. MPI_Send in GPU code

Many Nodes with GPU-initiated network communications

- data exchange
GPU Specific Optimizations

The key issue of tying together many GPUs will be the latency of data transfer between the memory of a GPU to another.

- CPU Memory zero-copy via native RDMA primitives: Send, Put and Get
- GPU Memory zero-copy is harder, but we’ll try
- GPU-initiated network communications, i.e. MPI_Send in GPU code

Many Nodes with GPU-initiated network communications

- data exchange
The key issue of tying together many GPUs will be the latency of data transfer between the memory of a GPU to another.

- CPU Memory zero-copy via native RDMA primitives: Send, Put and Get
- GPU Memory zero-copy is harder, but we’ll try
- GPU-initiated network communications, i.e. MPI_Send in GPU code

Many Nodes with GPU-initiated network communications

- data exchange

R. Ammendola

APENet+: a network for PetaFlops Scale LQCD Simulations
The key issue of tying together many GPUs will be the latency of data transfer between the memory of a GPU to another.

- CPU Memory zero-copy via native RDMA primitives: Send, Put and Get
- GPU Memory zero-copy is harder, but we’ll try
- GPU-initiated network communications, i.e. 
  MPI_Send in GPU code
Application Level Developments

We are catalyzing activities from different areas:

- test and code porting for single/multiple GPU [F. Simula, N. Tantalo, G. Salina]
- Lattice code development for multi GPUs [C. Bonati, G. Cossu, M. D’Elia, A. Di Giacomo]
- Development of a GPU numerical library for gravitational signals analysis [L. Bosi]
- GPUs for fast triggering and pattern matching at the CERN experiment NA62 [G. Collazuol, G. Lamanna, M. Sozzi]
Outlooks

- expected link peak bandwidth $\sim 3\text{GB/s}$
- delivery of first 6-links prototypes by the end of summer
- assembly of a GPU based cluster by end of 2010
- APENet+ supports MPI as high level APIs $\Rightarrow$ easy porting of your code
- experimenting with GPU-specific optimizations via small MPI-like APIs (very hard without GPU internals)