INFN Istituto Nazionale di Fisica Nucleare

APE group

# Leveraging NVIDIA GPUDirect on APEnet+ 3D Torus Cluster Interconnect

davide.rossetti@roma1.infn.it

GTC2012

San Jose, May 14-17, 2012

# The people involved

APE group


Alessandro Lonardo


Pier Stanislao Paolucci


Davide Rossetti


Piero Vicini


Roberto Ammendola


Andrea Biagioni

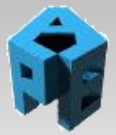

Ottorino Frezza


Francesca Lo Cicero


Francesco Simula

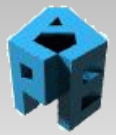

Laura Tosoratto

# The APEnet+ History




- Custom HPC supercomputers: APE (86), APE100 (94), APEmille (99), apeNEXT (04)
- APEnet cluster interconnect:
  - 2003-2004: APEnet V3
  - 2005: APEnet V3+, same HW with RDMA API
  - 2006-2009: APEnet goes embedded, integrate with ARM9
  - 2010: APEnet V4 aka APEnet+
  - 2011: APEnet has NVidia GPU acceleration

# Why a (custom) network



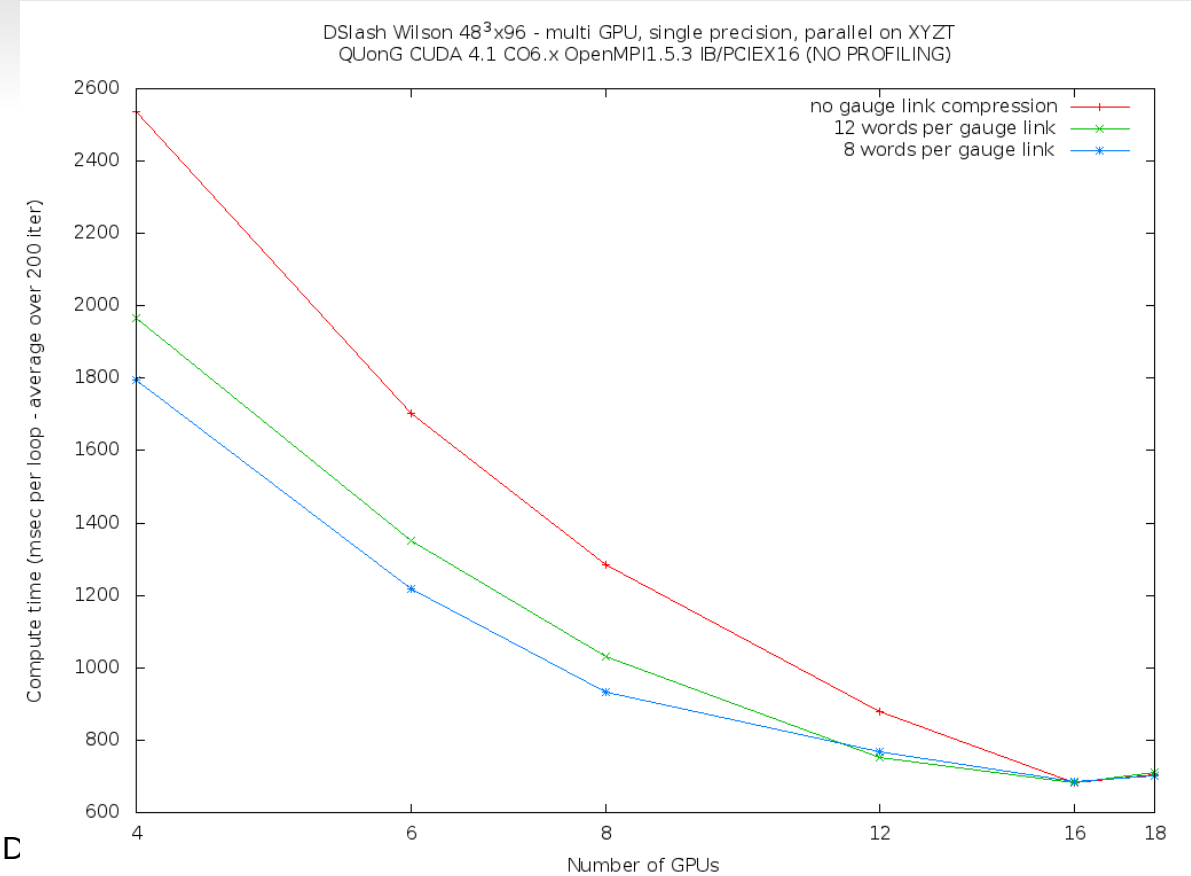
*APE group*

- The guy gluing the computing nodes
- Essential in any parallel (super)computer
- Should be also for CPU vendors if:
  - AMD purch. SeaMicro
  - Intel purch. Cray's 3D Torus interconnect
  - Intel purch. QLogic

# If not convinced…



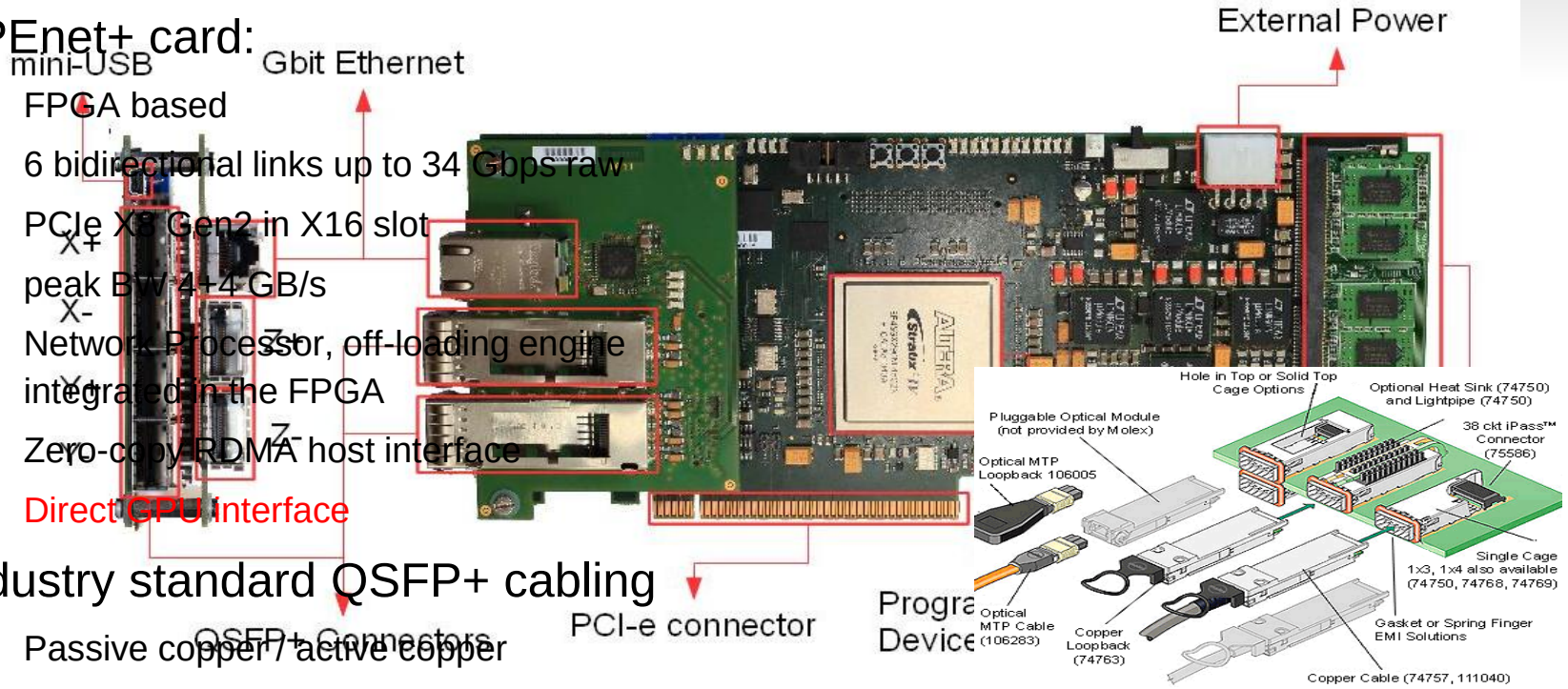
- Theoretical Particle physics
- 4D problem
- Domain decomposed



# APEnet+ at a glance

APE group

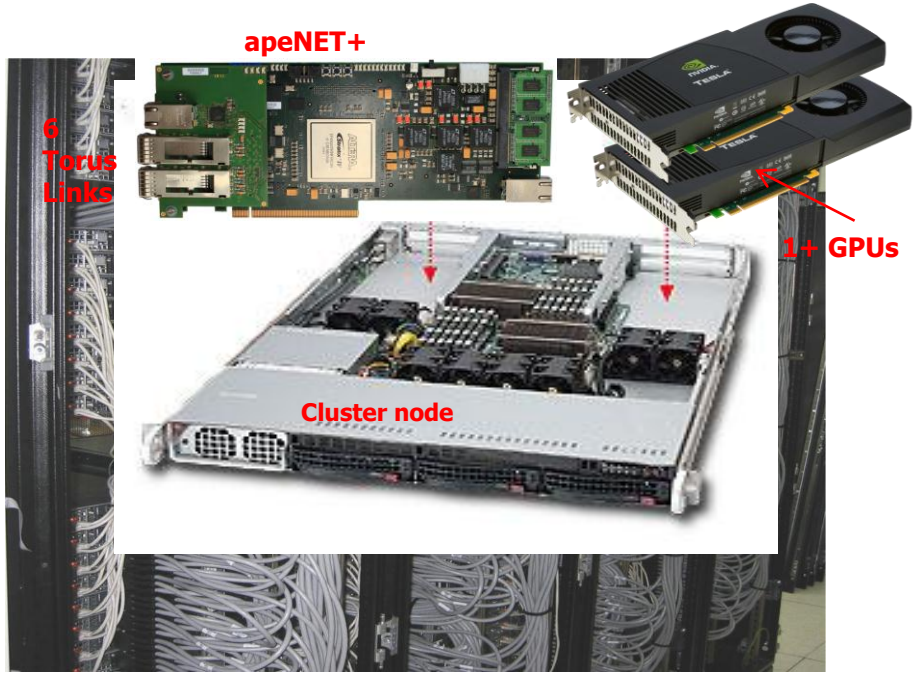
- APEnet+ card:
  - FPGA based
  - 6 bidirectional links up to 34 Gbps raw
  - PCIe X8 Gen2 in X16 slot
  - peak BW 4+4 GB/s
  - Network Processor, off-loading engine integrated in the FPGA
  - Zero-copy RDMA host interface
  - Direct GPU interface
- Industry standard QSFP+ cabling
  - Passive copper / active copper
  - Optical



# GPU clusters with a 3D Torus network

APE group

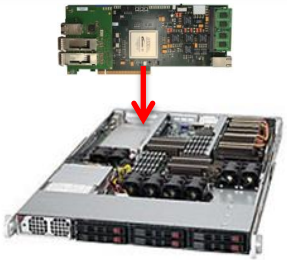
- Buy N cluster nodes
- For each node:
  - plug M GPUs
  - and 1 APEnet+ card
  - connect 6 cables to 1st neighbors
- 128 nodes
  - arranged in a XYZ=8x4x4 3D torus
  - with periodic boundary conditions.



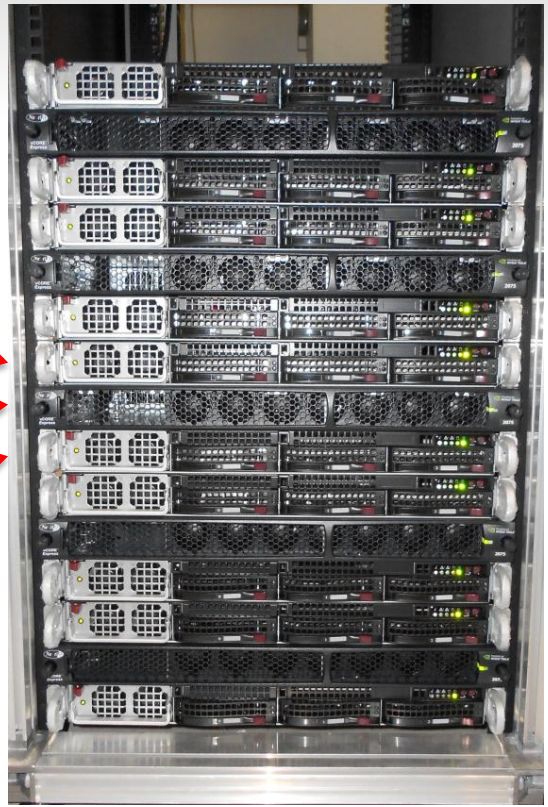
# A handier packaging

APE group

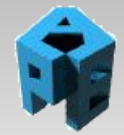
1U, two multi-core INTEL server equipped with APEnet+ card



1U, S2075 NVIDIA system packing 4 Fermi-class GPUs (~4 Tflops)

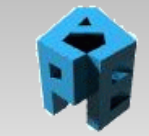


# APEnet programming models




- Native RDMA API:
  - similar to IB VERBS but much much simpler
  - RDMA buffer registration: pinning and posting combined
  - One-sided comm primitives
  - Single async TX message queue
  - Async delivery of completion events (both TX and RX)
- MPI:
  - early prototype
  - APEnet BTL module for OpenMPI

# Why we are here at GTC

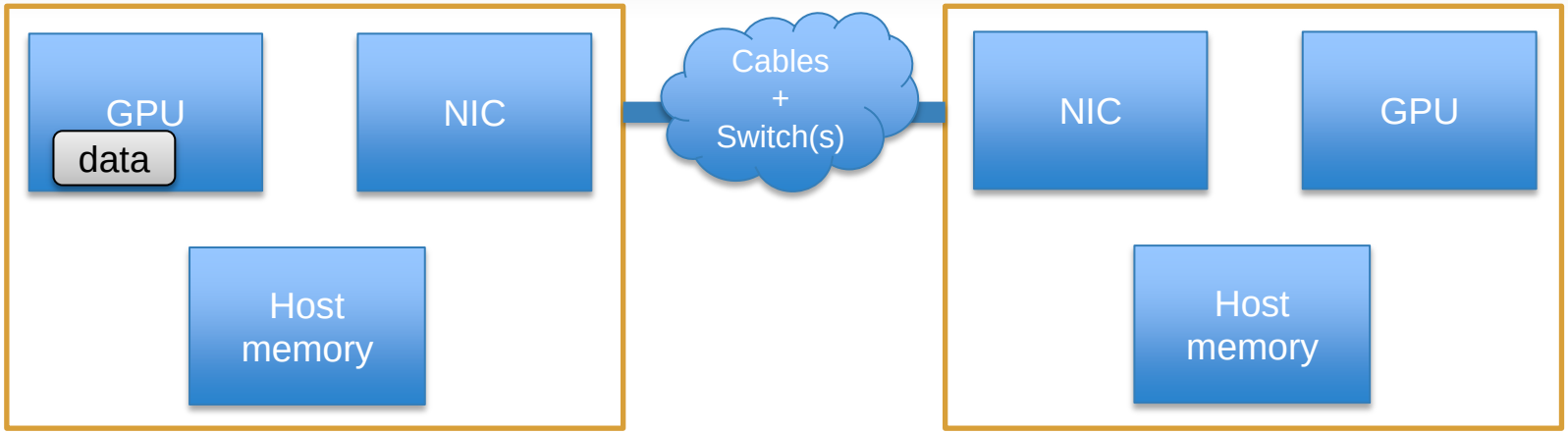



- Then in September 2011 we started a 2 weeks joint act. with Nvidia people
- After 1st week: GPU RX done
- After 2nd week: GPU TX lied down
- End October: most bugs fixed.

We got GPU-to-GPU working!

Presented results at Nvidia booth @SC11

# Normal multi-GPU flow



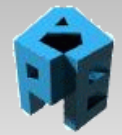

cudaMemcpy(D2H)

MPI_Send()

MPI_Recv()

cudaMemcpy(H2D)

# GPU extensions



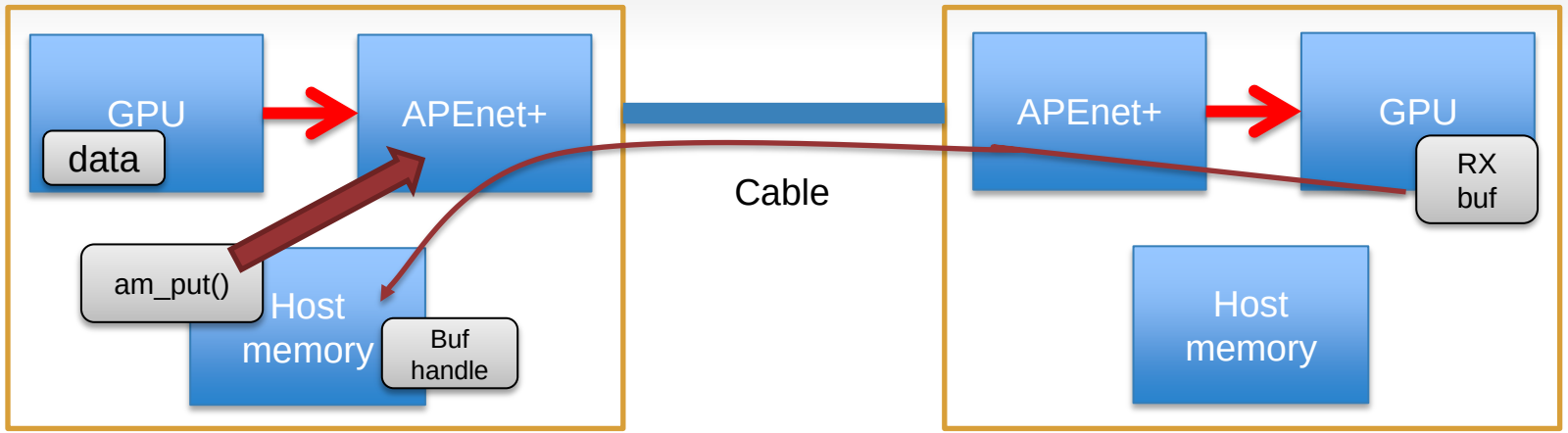

programming model naturally extended for GPUs:

- register GPU buffers as RDMA RX buffers
- destination can be also GPU memory (GPU RX)
- send messages also from GPU memory (GPU TX)
- no need to use cudaMemcpy/Async and streams to copy to/from temporary host buffers
- needs CUDA 4.1 and UVA

supported by new GPU-specific HW blocks…

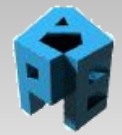
# APEnet+ with GPU Peer-to-Peer

APE group



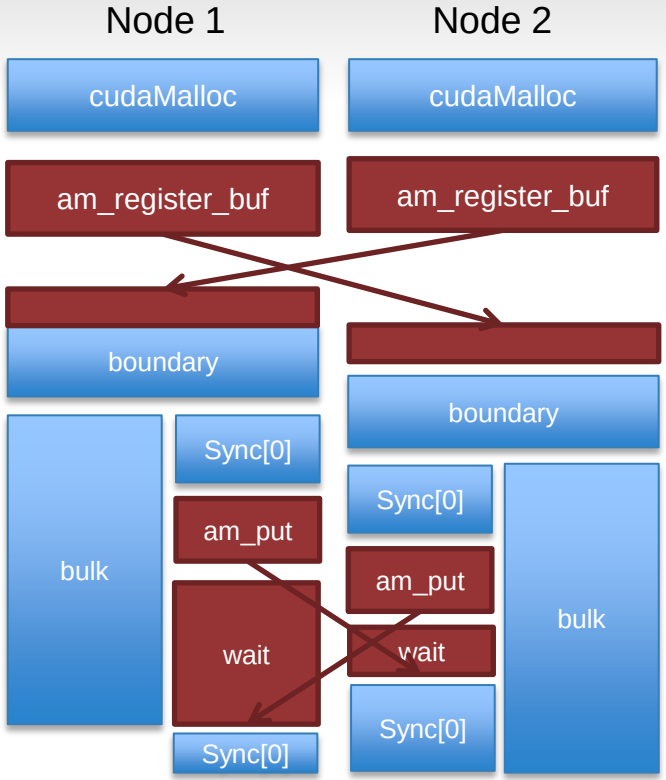

- custom HW blocks (patent pending)
- modified APEnet firmware

# example 1: data parallel algo

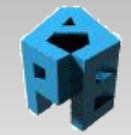


APE group

```
// 1D example along X, up is X+, down is X-
stencil[up][0] = cudaMalloc(LEN); stencil[down][0] = cudaMalloc(LEN);
stencil[up][1] = cudaMalloc(LEN); stencil[down][1] = cudaMalloc(LEN);
volume = cudaMalloc(VOLSIZE); // allocate volume of data
am_register_buf(stencil[up][0], &handle[up][0]); // ...
// send all RDMA handle[][] to my X+ and X- nodes
// collect GPU stencil RDMA address in rem_handle[up|down][0|1]
int even = 0, odd = 1;
// exchange event stencils among nodes
while(some_convergence_criteria_met) {
  calc_boundary<<<,stream_0>>>(volume,stencil[up][even],stencil[down][even]);
  calc_bulk<<<,stream_1>>>(volume);
  cudaStreamSynchronize(stream_0);
  am_put(node[up],volume+boundary_off[up],rem_handle[up][odd],LEN,AM_GPU_MEM);
  am_put(node[down],volume+boundary_off[down],rem_handle[down][odd],LEN,AM_GPU_MEM);
  // wait for send and recv done
  cudaStreamSynchronize(stream_1);
  even = !even; odd = !odd;
}
```



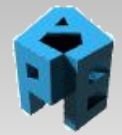
# example 2: task parallel workload




```
// on processing nodes
void *gpu_buf = cudaMalloc(len);
am_vaddr_t buf_handle;
am_register_buf(gpu_buf,len,&buf_handle,
    AM_PERSISTENT_BUF|AM_GPU);
// send gpu_handle to master node
while(true) {


  wait_for_buf_written_event();
  compute_data<<<>>>(gpu_buf, len);
  // send results somewhere
}
```

```
// on master node
data = malloc(len);


for(n=0; n<num_nodes; ++n) {
  // collect GPU buf handles in dest_buf_handles[n]
}
while(data_to_process) {
  ADC_read_data(data, len);
  for(n=0; n<num_nodes; ++n)
    am_put(node[n],data,dest_buf_handles[n],len);
}
```

# benefits from P2P



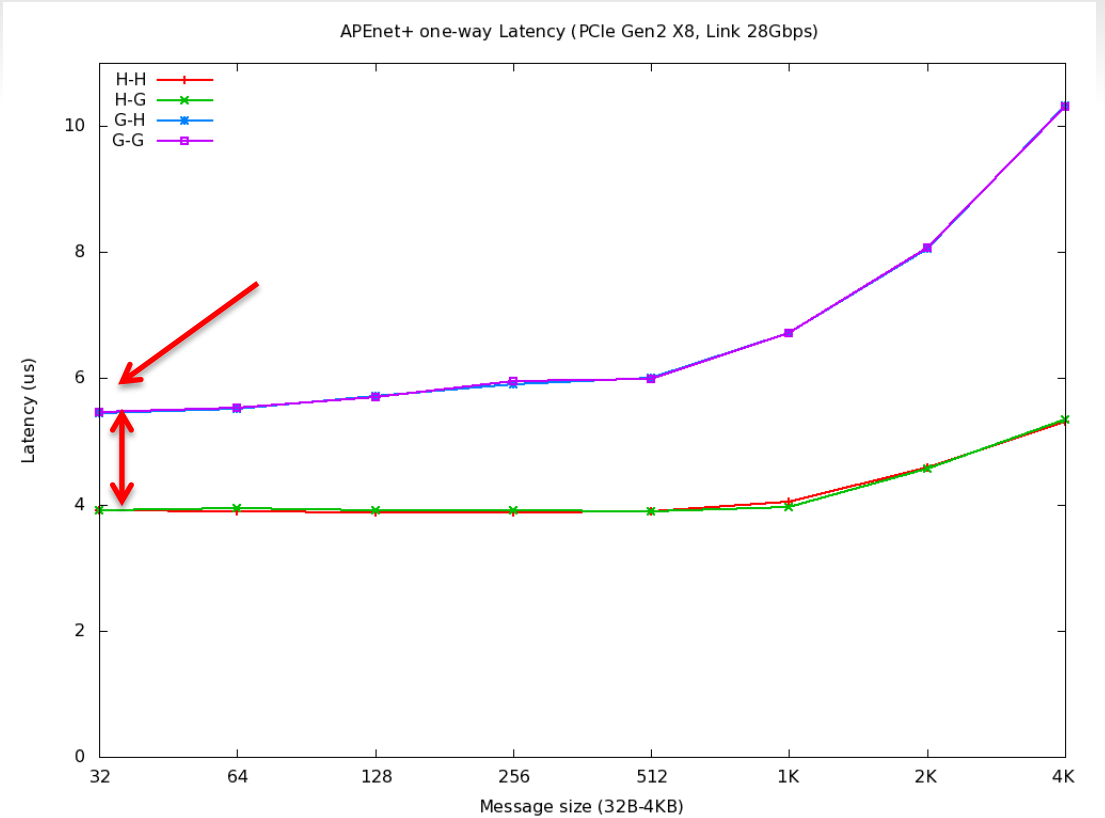

- Free GPU copy engines for other tasks
- Because of that, less interference in your optimized codes
- Latency... see below
- Bandwidth: it depends…
  - not for peak BW in the general case
  - could help in some cases, for effective BW*
    - Depending on PCIe bus topology
    - And on the number of GPUs per host

* *See S0515 Multi-GPU Programming* by Paulius Micikevicius

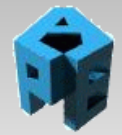
# Latency benchmark

APE group

- MPI-like one-way latency test
  - re-coded using RDMA PUT
  - No small msg optimizations (rendezvous)
  - No big difference with round-trip
- 5.4 us on GPU-GPU test!
- GPU TX demanding !! still …

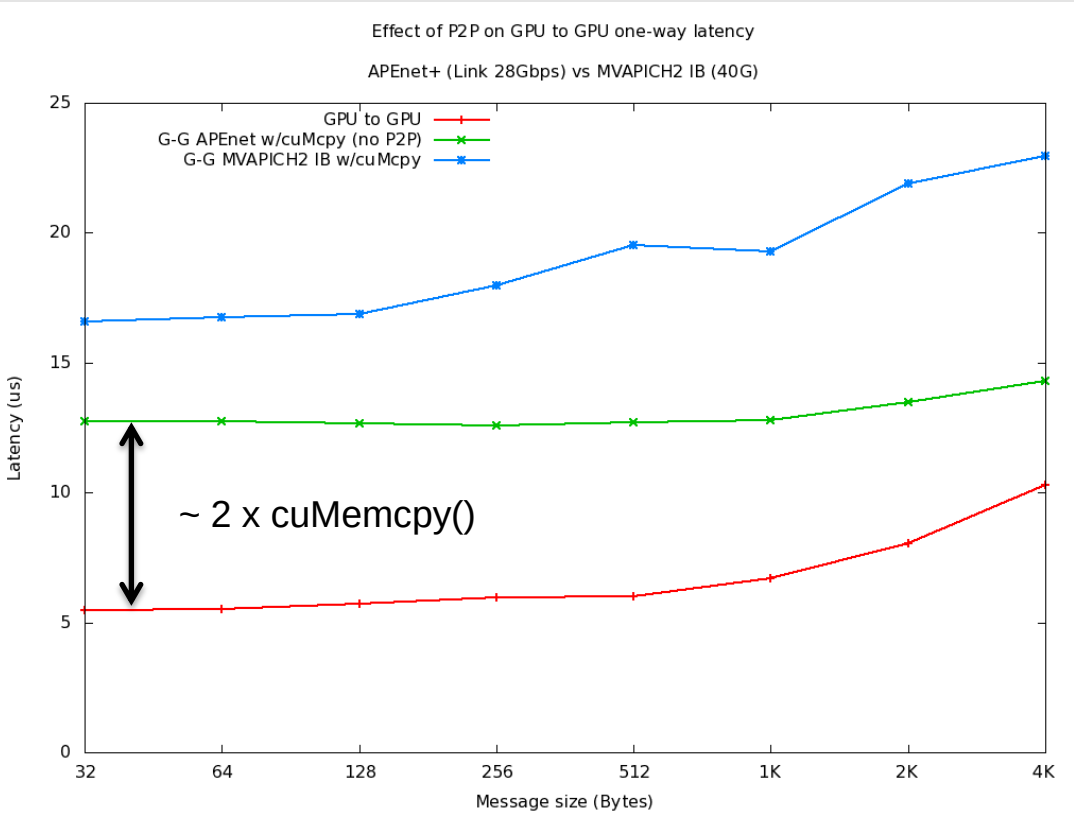


# Latency benchmark: P2P effects



*APE group*

- No P2P = cuMemcpyD2H/H2D() on host bounce buffers
- Buffers pinned with cuMemHostRegister
- 2 cuMemcpy() cost ~ 8-10us
- MVAPICH2 tested on same test system

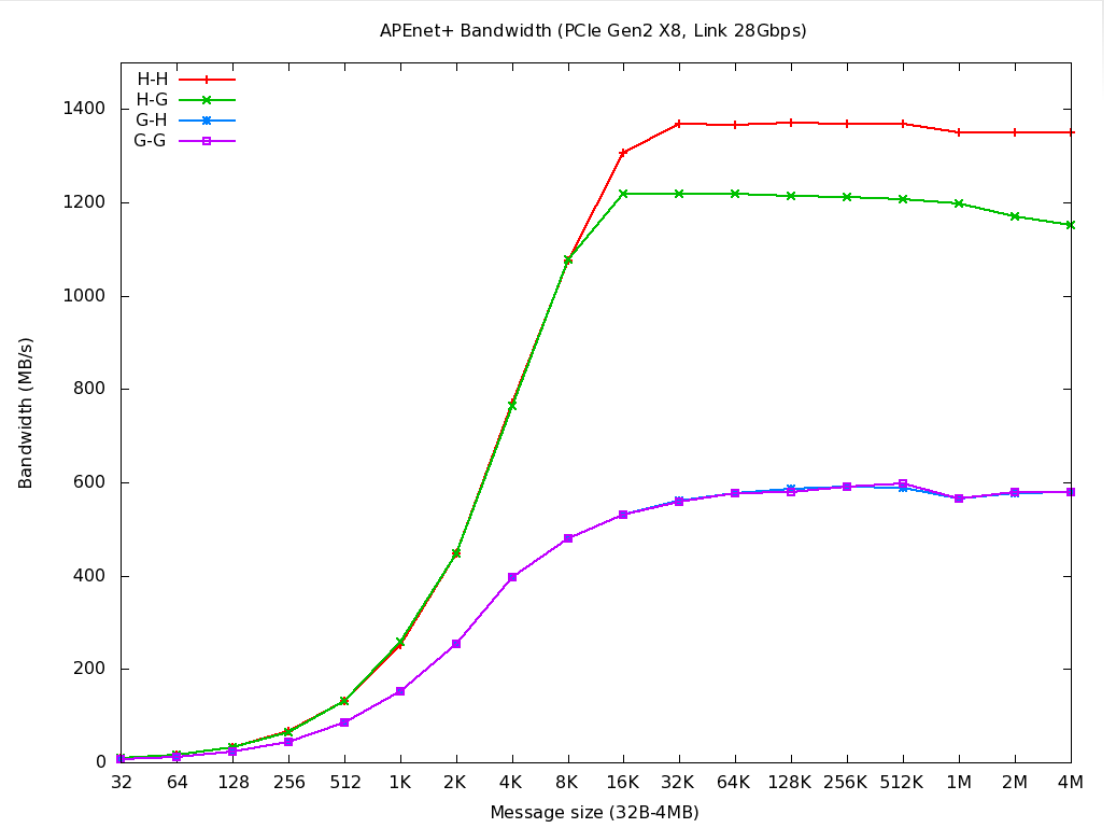


# Bandwidth benchmark

APE group

Preliminary results:

- H-H: TX from host memory, RX on host memory
- H-G: (GPU RX) almost as good
- G-H & G-G: P2P READ protocol overhead
- Still much room for improvements

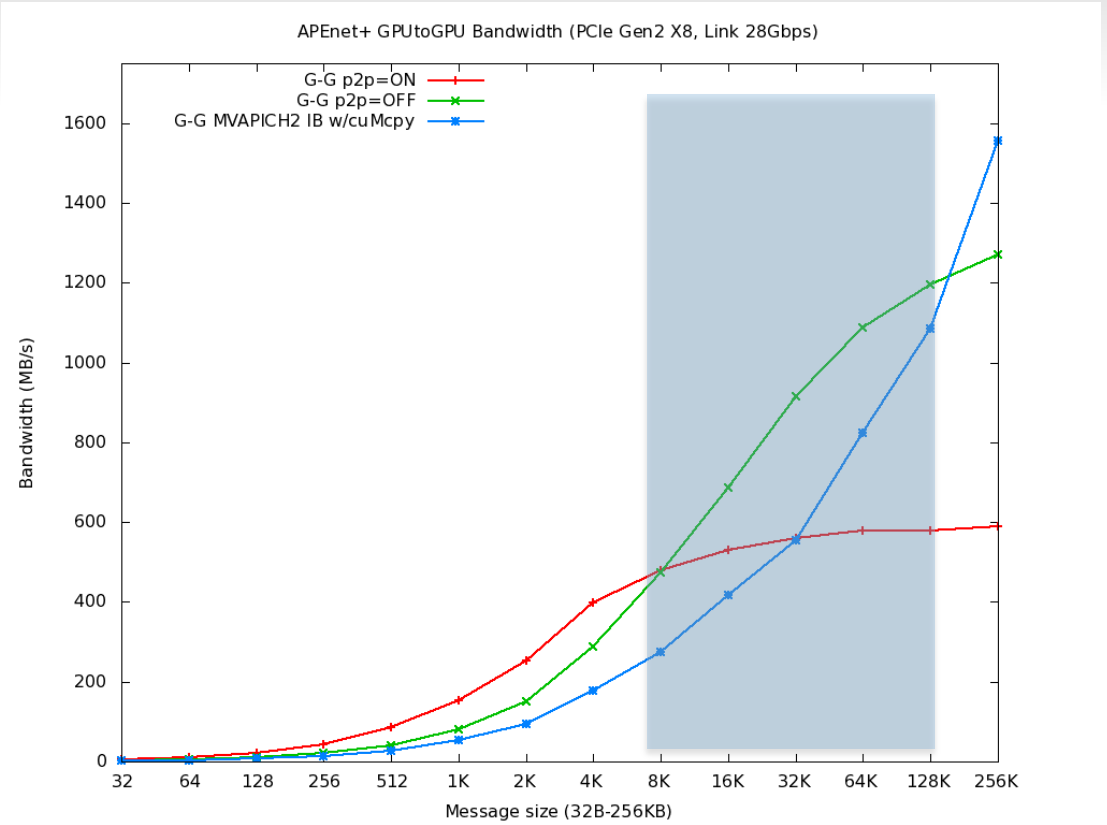


# APEnet+ vs rest of the World

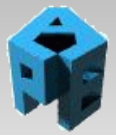
APE group

- GPU to GPU test
- APEnet+ P2P ON
- APEnet+ P2P OFF: cuMemcpy() added
- MVAPICH2 GPU-aware MPI over IB: cuMemcpy() hidden

1. 8KB: threshold for P2P
2. 128KB: threshold for APEnet+ over IB



# APEnet+ roadmap

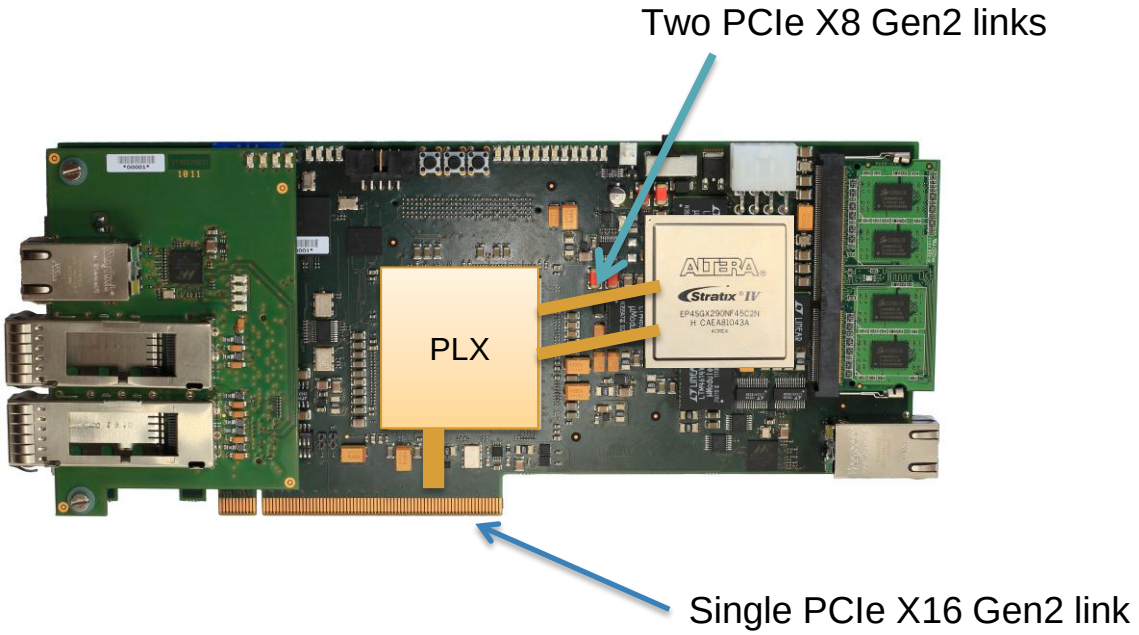



1. PCIe X8 -> X16
2. PCIe Gen3
3. Tesla/APEnet fusion
4. Embedded/cheap/highly packed solution

# 1) Double the data bus width

APE group

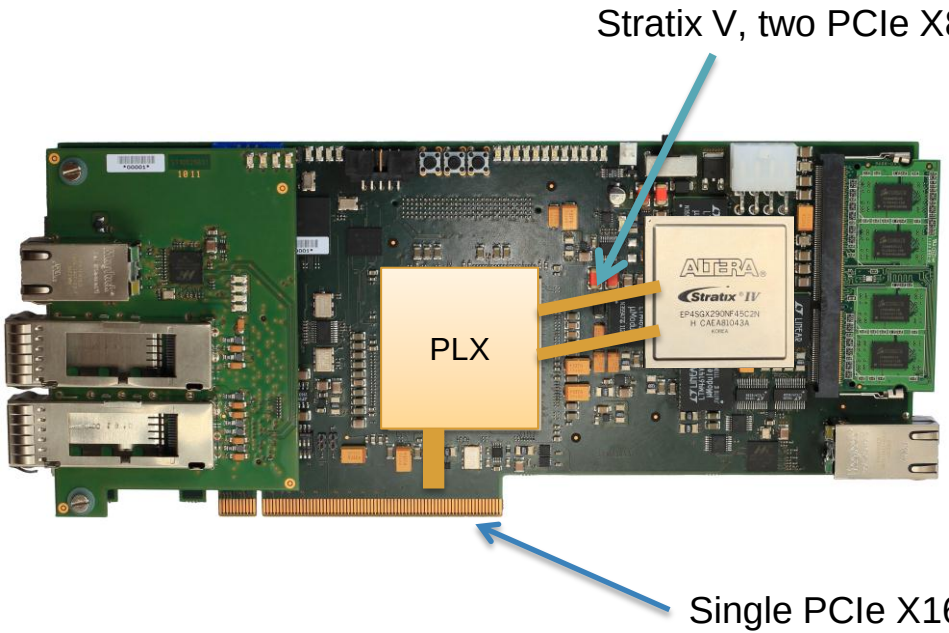
- Today FPGAs limited to PCIe X8 Gen2
- But can have two link!
- Plug PLX "*X16 to 2X8*" PCIe switch
- Split traffic between the 2 links: e.g. one GPU each



# 2) Double the BW

APE group

- Kepler is Gen3
- Move to PCIe Gen3
- Next gen FPGAs have Gen3
- In catalog Gen3 PLX switch
- Combine with prev step for x4 I/O



# GPU/APEnet fusion card

APE group

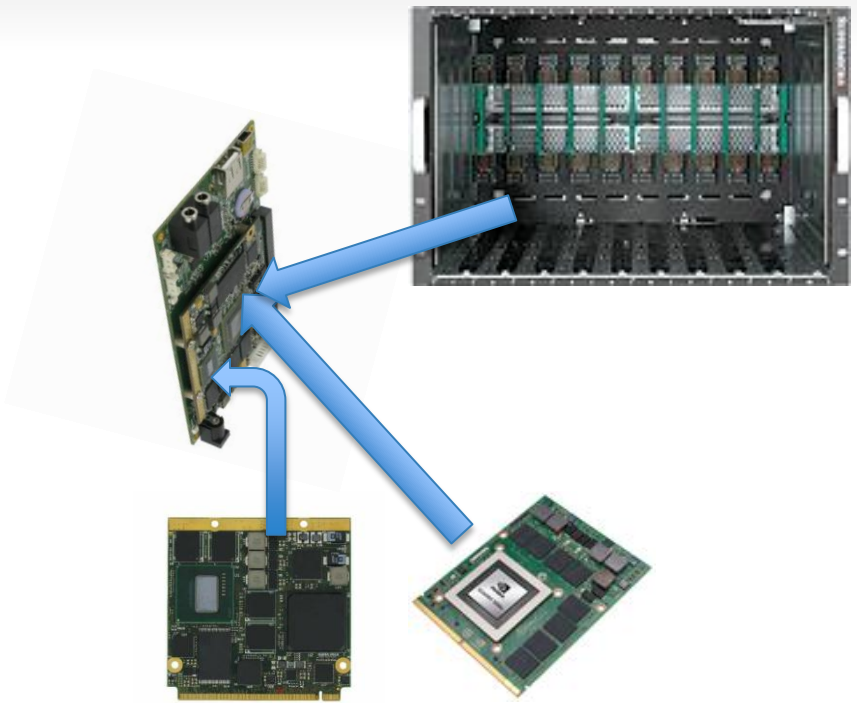
- Frankenstein board
  - Take Half of a “GTX 590”
  - Add dedicated APEnet
- Tune net bandwidth
- Put 1,2,.. per server
- Perfect for HPC

# Embedding APEnet

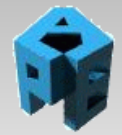
APE group

Idea by mfatica/eorlotti@nvidia:

- Design a 3U blade carrier board with APEnet on it (FPGA + cages)
- Cheaper FPGA
- Plug CPU module: Atom, Tegra, …
- Plug a discrete GPU: Quadro 5010M ?

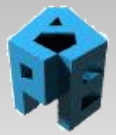


# Executive summary




- The APEnet+ interconnect
- The APEnet programming model
- Performance profiling
- Future work

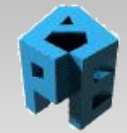
# Conclusions

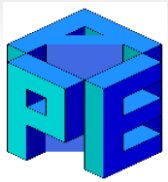



- 1st non Nvidia device to implement Fermi P2P protocol
- Mostly done, smoothing edges
- Alternative approach: P2P on IB, see Joel Scherpelz presentation at OpenFabrics workshop[1]
- We need your help!!! very open to collaborations, to share code, …

[1] https://www.openfabrics.org/press-room/upcoming-events/109.html

# Credits

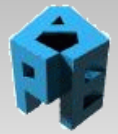


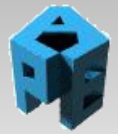


- APEnet design and development in INFN by the APE™ team ☺
- Partially supported by EU EURETILE project (euretile.roma1.infn.it)
- GPU support developed in collaboration with Massimiliano Fatica, Timothy Murray, Joel Scherpelz @ Nvidia

# backup slides



*APE group*



APE group

## S0282 - Leveraging NVIDIA GPUDirect on APEnet+ 3D Torus Cluster Interconnect

**Davide Rossetti ( Researcher, Italian National Institute for Nuclear Physics )**

APEnet+ is a novel cluster interconnect, based on a custom PCI card which features a PCI Express Gen2 X8 link and a re-configurable HW component (FPGA). It supports a 3D Torus topology and has special acceleration features specifically developed for NVIDIA Fermi GPUs. An introduction to the basic features and the programming model of APEnet+ will be followed by a description of its performance on some numerical simulations, e.g. High Energy Physics simulations.

**Topic Areas:** Supercomputing; Computational Physics

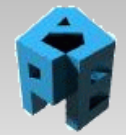
**Session Level:** Intermediate

**Day:** Thursday, 05/17

**Time:** 4:00 pm - 4:25 pm

**Location:** Room K

# example 2: task parallel workload (original slide)



APE group

read data from DAC on master node(s) and distribute it directly on all GPUs memory for processing (Host -> GPU)

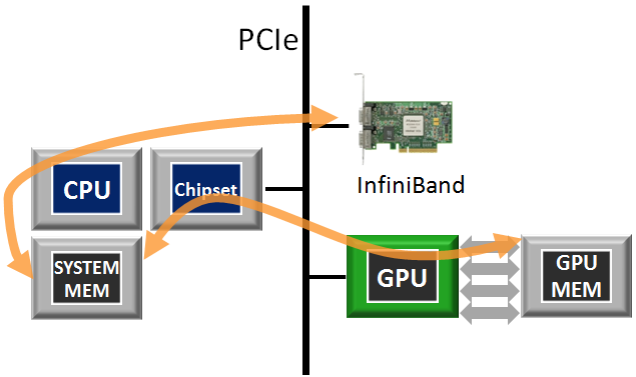
```
// on processing nodes

void *gpu_buf = cudaMalloc(len);

am_vaddr_t buf_handle;
am_register_buf(gpu_buf, len, &buf_handle, AM_PERSISTENT_BUF|AM_GPU);

// send gpu_handle to master node

while(wait_for_buf_written_event) {

      compute_data<<<>>>(gpu_buf, len);

      // send results somewhere

}


// on master node

data = malloc(len);

// collect dest GPU buf handles from other nodes in node_gpu_addr[]

while(data_to_process) {
                DAC_read_data(data, len);

        for(n=0; n<num_nodes; ++n)

                        am_put(node[n], data, node_buf_handle[n], len);

}
```

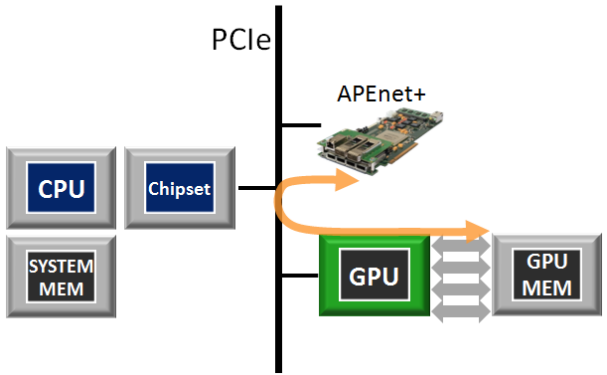
# P2P

APE group

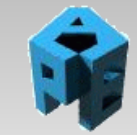
- custom HW blocks (patent pending)
- modified APEnet firmware

to support NV P2P protocol (GPUDirect 2)

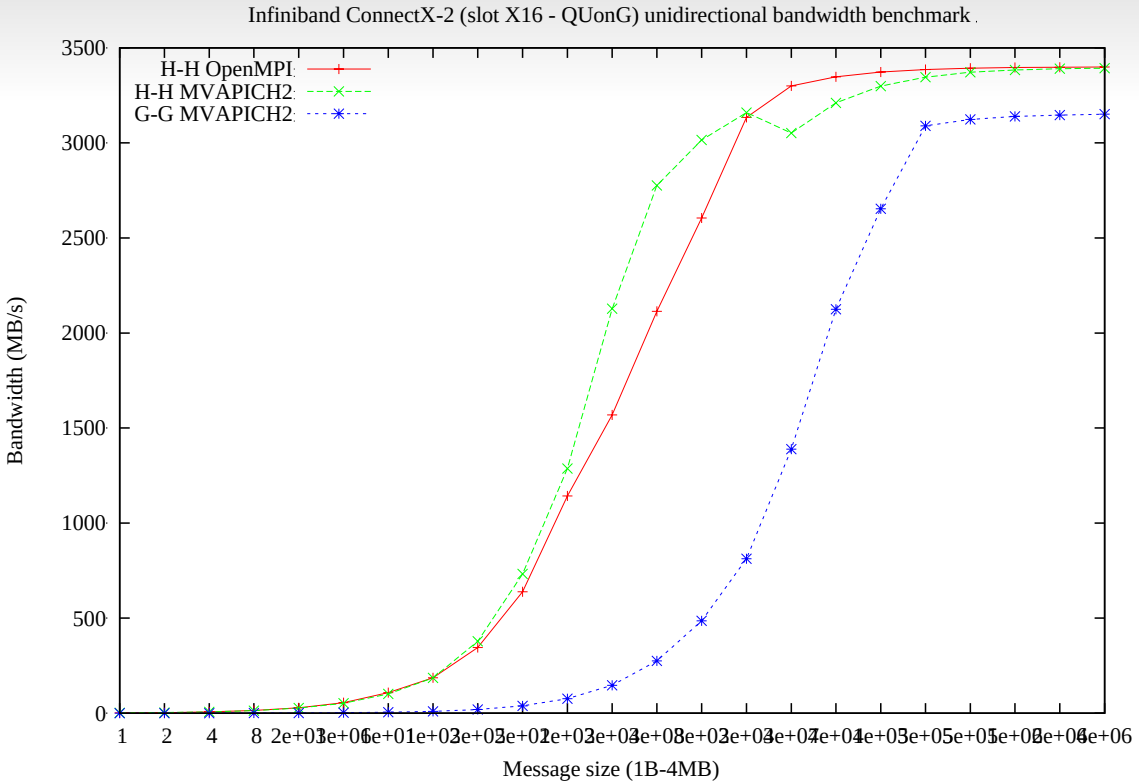


# 2 GPU-aware MPIs on IB




- Only have data for MVAPICH2 on GPU-to-GPU
- MVAPICH2 seems better
- Both break compute-net overlapping

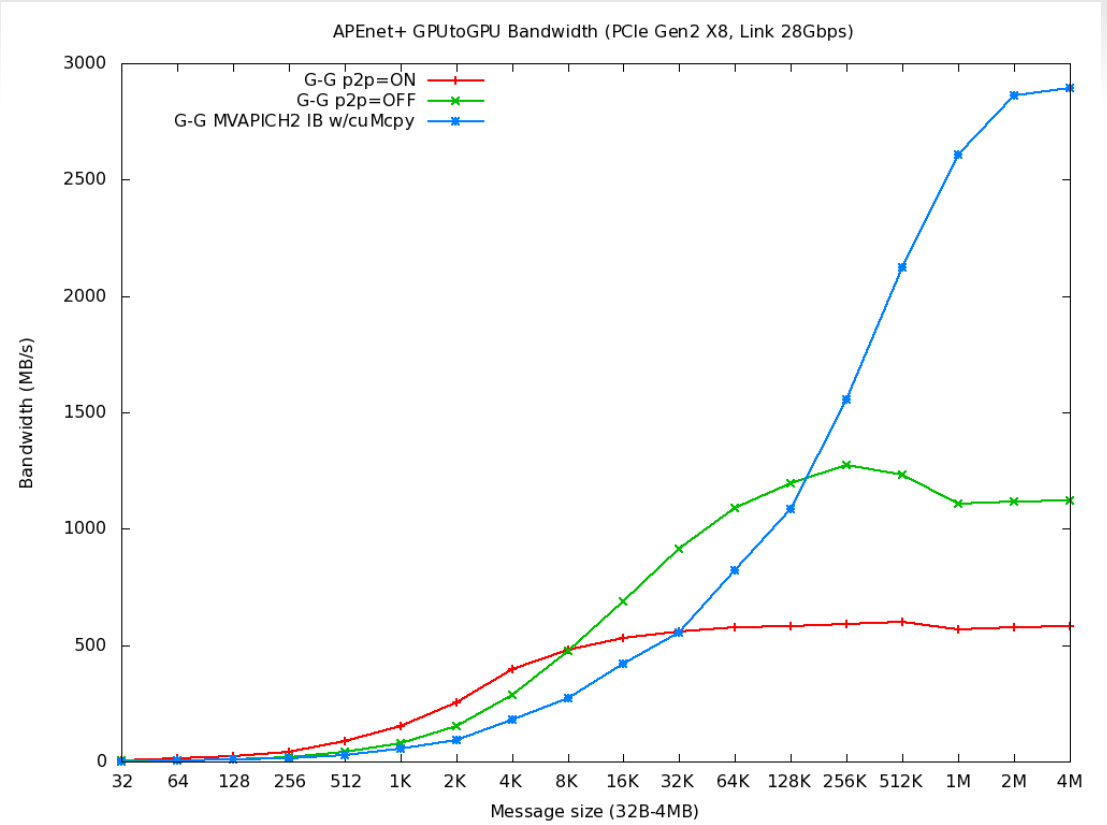


# APEnet+ vs rest of the World

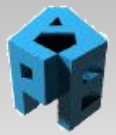
APE group

- GPU to GPU test
- APEnet+ P2P ON
- APEnet+ P2P OFF: cuMemcpy() added
- MVAPICH2 GPU-aware MPI over IB: cuMemcpy() hidden

1. 8KB: threshold for P2P
2. 128KB: threshold for APEnet+ over IB

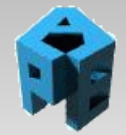


# Current problems




- PERF:
  - P2P READ has limited peak BW (~800MB/s on Gen1), and needs prefetching to get near it (see plots)
  - Working on faster TX DMA engine
- Activating CUDA PROF (text mode) seems to increase perf of APEnet.
  - Timestamp, gpustarttimestamp, gpuendtimestamp, streamid
  - Clues ? x86 mem barriers ?

# CUDA PROF problem



*APE group*

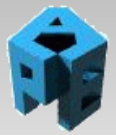
plain + APEnet GPU buffers (P2P on)

| | profiler off | profiler on |
|---|---|---|
| 2 nodes | 639ps | 486ps |
| | 1.070715s | 0.815895s |
| | 638ps | 488ps |
| | 1.070273s | 0.817967s |
| | 638ps | 485ps |
| | 1.071047s | 0.814313s |

plain + APEnet host buffers (P2P off)

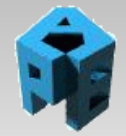
| | profiler off | profiler on |
|---|---|---|
| 2 nodes | 476ps | 463ps |
| | 0.799172s | 0.776773s |
| | 458ps | 463ps |
| | 0.767835s | 0.776532s |
| | 477ps | 448ps |
| | 0.800457s | 0.751133s |
| | 458ps | 447ps |
| | 0.768611s | 0.750125s |
| | | 447ps |
| | | 0.750203s |

# future work




- better integration with CUDA event mechanism: kernel or user-space ?
- BAR1 access: not done yet because scope is still narrow (BIOS, kernel)
- GPU initiated communications: APEnet as a P2P slave

# CUDA events & APEnet



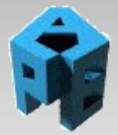

## CUDA event triggers an APEnet TX:

```
kernel_a<<<,stream0>>>();
Kernel_b<<<,stream1>>>();
cudaEventRecord(ev,stream0);
// tx queued but wait for event
am_put(next,ndestaddr,len,ev);
am_put(prev,pdestaddr,len,ev);
```

## APEnet event triggers a kernel/copy launch

```
cudaEvent_t ev=cudaEventCreate();
buf=cudaMalloc(sz);
am_register_buf(buf,sz,&ev);
// kernels queued but wait for RX message landing on buf
cudaStreamWaitEvent(stream0,ev,0);
kernel_a<<<,stream0>>>();
Kernel_b<<<,stream0>>>();
```

# Credits



*APE group*

our work is partially supported by EU
EURETILE project (FP7 FET-ICT-2009.8.1 247846)

many thanks go to...