Review on the GPU-related activities in INFN

Roberto Ammendola

Istituto Nazionale di Fisica Nucleare, Sezione Roma Tor Vergata

Workshop CCR 2010, Catania – 18 May 2010

GPU motivation: the brute force

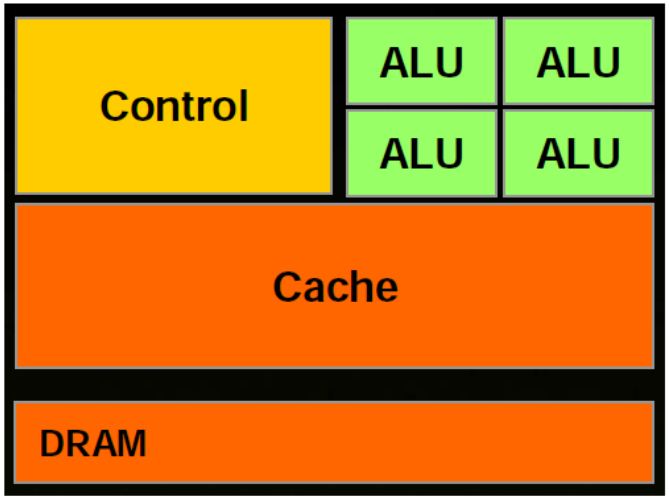
Some raw numbers:

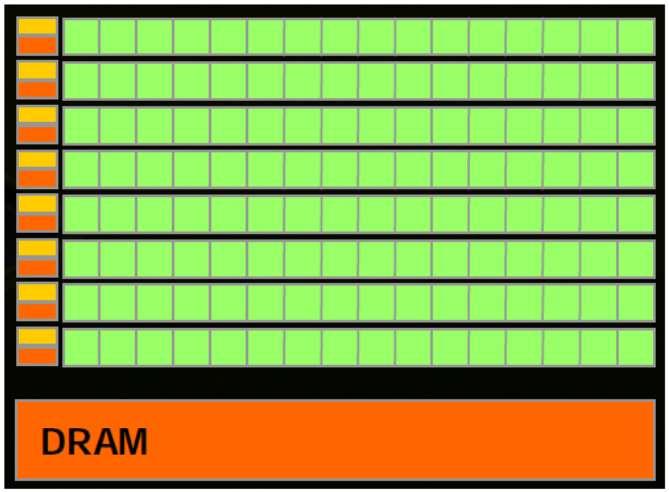
	Xeon X5670	Opteron 8439	ATI HD 5870	Tesla C1060	Tesla C2070
# of cores	6	6	1600	240	448
SP GFlops	140	134	2720	933	1030
DP GFlops	70	67	544	78	515
GiB of Mem	-	-	1	4	6
TDP (Watt)	95	105	188	188	247
Price	1600	2000	400	1500	< 2000
€ / GFlops	23	30	1.4	19	< 4

GPU technology cannot be considered a transient, it should be exploited.

GPU architecture

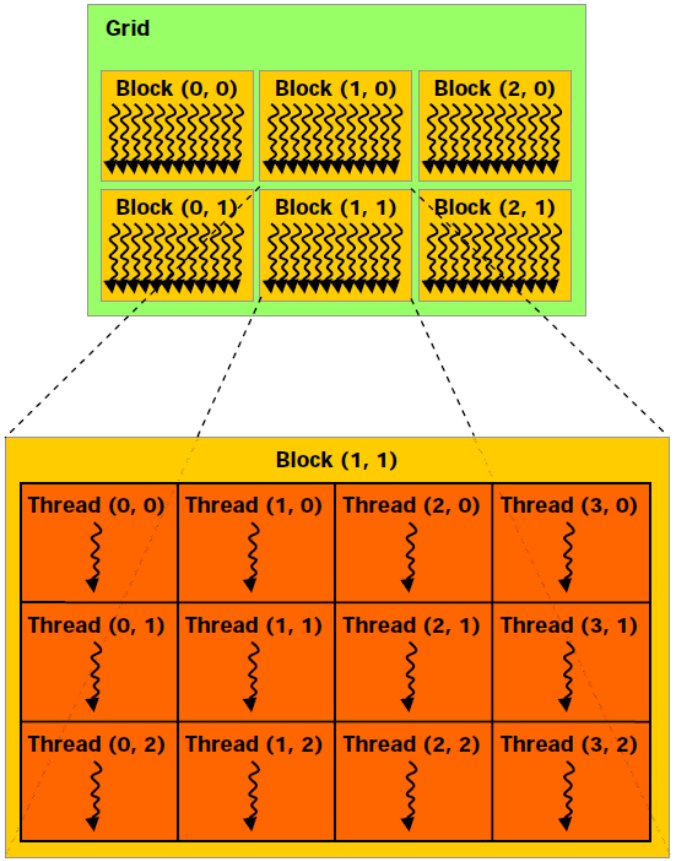
- CPU
  - a relatively small amount of die area is used for FPUs
  - much of area and energy is used for cache management, out-of-order execution, branch prediction, . . .
  - if you are smart you know how to get better performances
- GPU
  - more area reserved to computational units
  - latency is affected by smaller and sparser memories
  - scheduler is very complicated

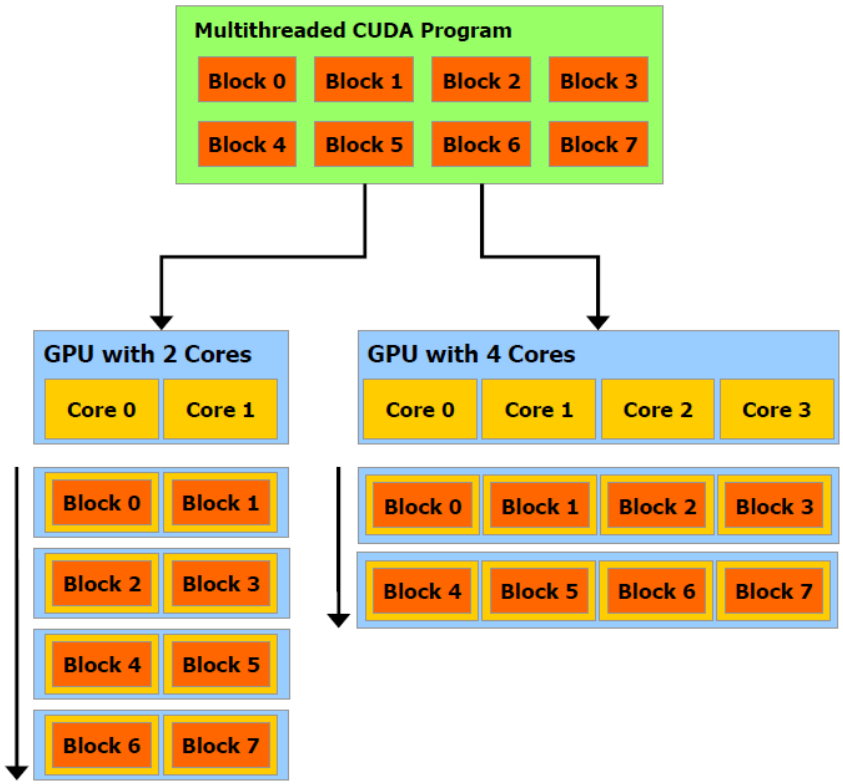




GPU programming model: some concepts

User has two handle: Threads and Blocks
They represent the two level of parallelism and they can be arranged in a 1-,2- or 3-dimensional order.





They are mapped down to hardware by the GPU scheduler, and there is no 1-to-1 relationship with Stream and Multi-Processors

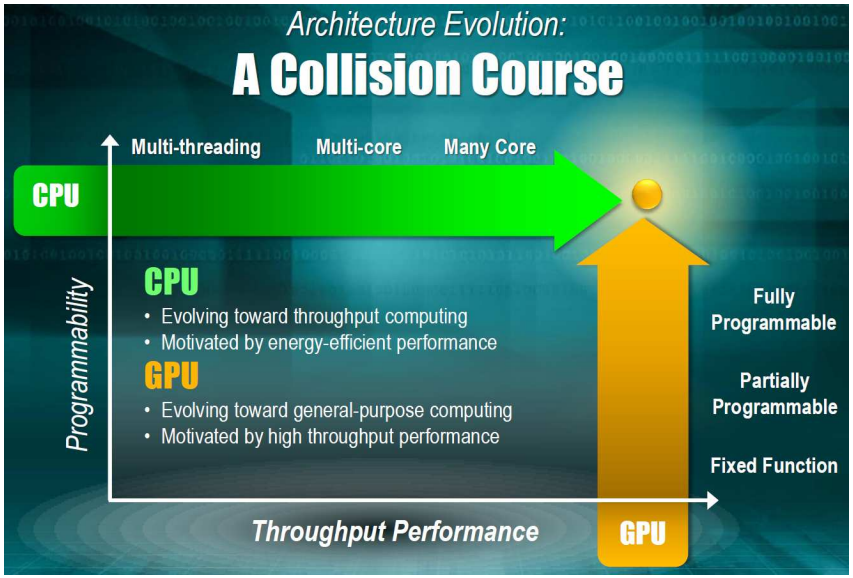
CUDA Code Example

- Simple example of a computational kernel: $\vec{C} = \vec{A} + \vec{B}$
- `for (i=0, i<N, i++) { C[i] = A[i] + B[i] }`
- threads are indexed by the keyword `threadIdx`
- CUDA insertions are marked with `<<< ... >>>`
- threadsPerBlock and blocksPerGrid are user defined

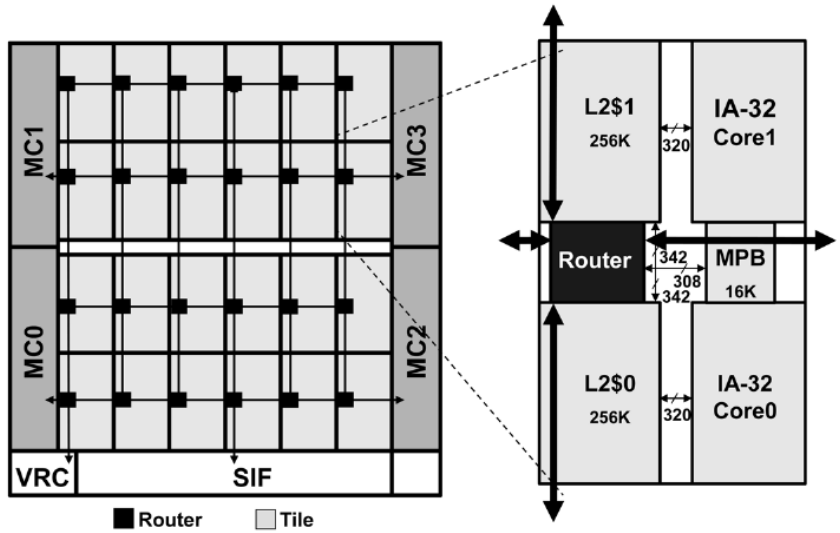
```
// Device code
__global__ void VecAdd(float* A, float* B, float* C)
{
    int i = threadIdx.x;
    if (i < N)
        C[i] = A[i] + B[i];
}

// Host code
int main()
{
    // Allocate vectors in device memory
    size_t size = N * sizeof(float);
    float* d_A;
    cudaMalloc((void**)&d_A, size);
    float* d_B;
    cudaMalloc((void**)&d_B, size);
    float* d_C;
    cudaMalloc((void**)&d_C, size);

    // Copy vectors from host memory to device memory
    // h_A and h_B are input vectors stored in host memory
    cudaMemcpy(d_A, h_A, size, cudaMemcpyHostToDevice);
    cudaMemcpy(d_B, h_B, size, cudaMemcpyHostToDevice);

    // Invoke kernel
    int threadsPerBlock = 256;
    int blocksPerGrid =
            (N + threadsPerBlock - 1) / threadsPerBlock;
    VecAdd<<<blocksPerGrid, threadsPerBlock>>>(d_A, d_B, d_C);

    // Copy result from device memory to host memory
    // h_C contains the result in host memory
    cudaMemcpy(h_C, d_C, size, cudaMemcpyDeviceToHost);

    // Free device memory
    cudaFree(d_A);
    cudaFree(d_B);
    cudaFree(d_C);
}
```

GPUs and CPUs are converging

- Major vendor are developing many-core architectures.
- There will be a pletora of combinations.

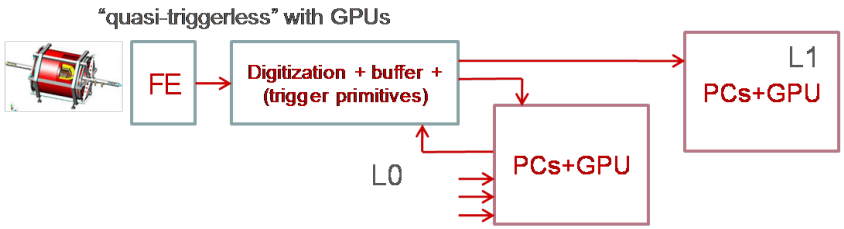


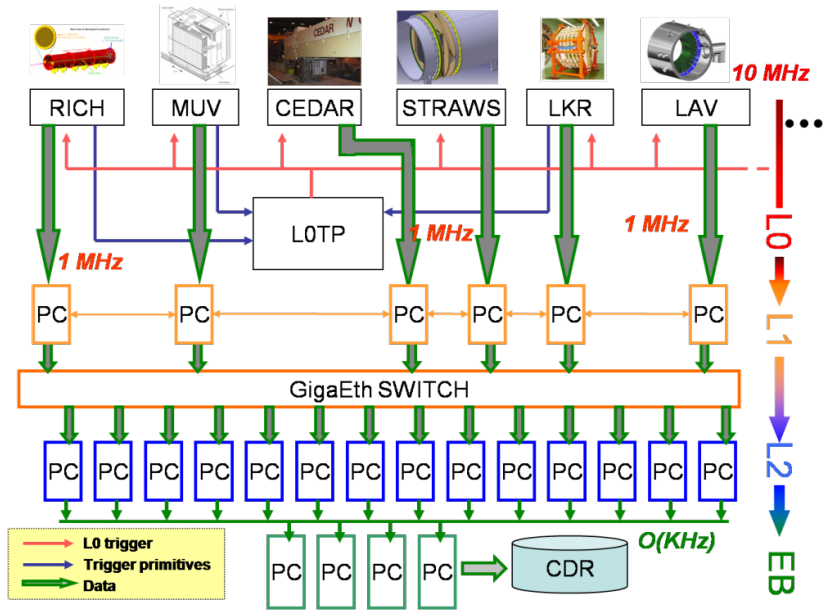
- Intel recently presented a 48-Core IA-32 Message Passing Processor.
- The news is a FIFO-based network-on-chip to reduce internal latency.



GPUs for fast triggering and pattern matching at the CERN experiment NA62

- NA62 aims at measuring the BR of the ultra-rare $K^+ \longrightarrow \pi^+ \nu\nu$ decay with O(100) events, using a very intense kaon beam produced at the CERN SPS
- Effective, selective and lossless readout and trigger (TDAQ) systems are crucial to collect a huge statistics in a reasonable time (2 years of data taking)
- Three level trigger:
  - L0 hardware trigger level with fixed latency: 1ms
  - L1 and L2 software levels with variable latency
- 10 MHz in input at L0, 1 MHz in input at L1, O(100) KHz in input at L2



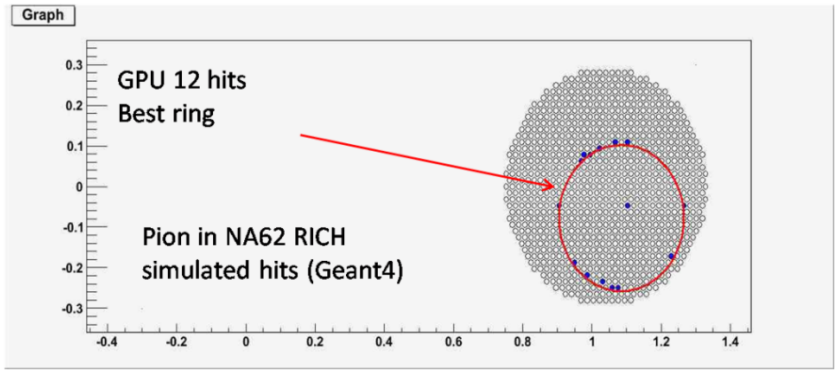


- High computing power, both in hardware and in software levels, could help to design a flexible and powerfull trigger system.
- Investigation to use massive parallel computing power in the Video Card processors (GPU) to implement pattern recognition and trigger algorithm

GPUs for fast triggering and pattern matching at the CERN experiment NA62

- The realtime use of the GPUs is slightly different from the parallel computing: very large input bandwidth, short latency, quasi-deterministic computing time, events processing parallelization, ...
- First attempt to understand the GPUs capabilities: fast pattern recognition in NA62 RICH
- The problem: find rings (with less than 20 hits each) in a sparse matrix (1000 points) at 10 MHz with a latency of 1 ms using a standard PC with a dedicated "video "card
- Several algorithms tested to fit the problem to the processor structure.
- Best result: 3.1 us per ring on NVIDIA TESLA C1060 (1 Teraflops) on sets of 1000 events

- Further development:
  - Use multiple video cards in parallel
  - Upgrade to the next generation GPUs (NVIDIA "FERMI "→ about 2 Teraflops)
  - Study use of GPU in a real time Operating System environment
  - Use a "smart "data link for data pre-processing on FPGA and DMA (Direct Memory Access)

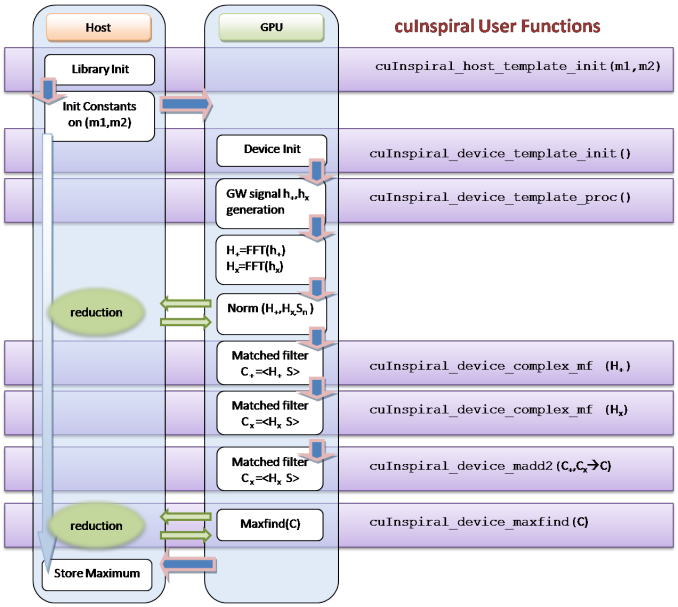


NA62 Pisa Group:

G.Collazuol, G.Lamanna, M.Sozzi

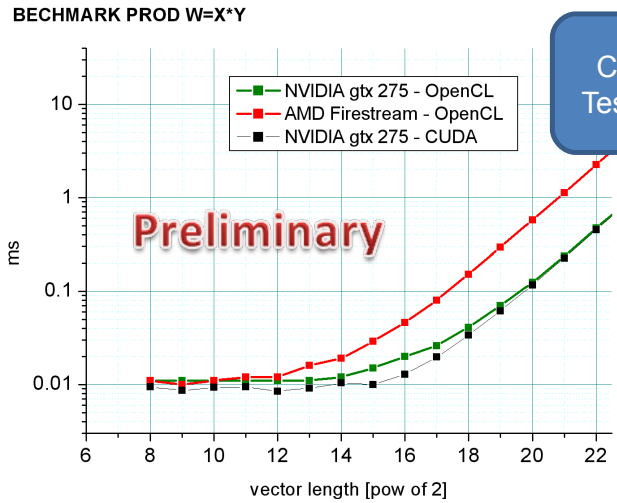
MaCGO Experiment [L. Bosi (PG) et al.]

- Manycore Computing for future Gravitational Observatory
- Funded by INFN Com. V 2010-2011
- Development of a numerical library for gravitational signals analysis (digital signal processing, FFT, . . .).

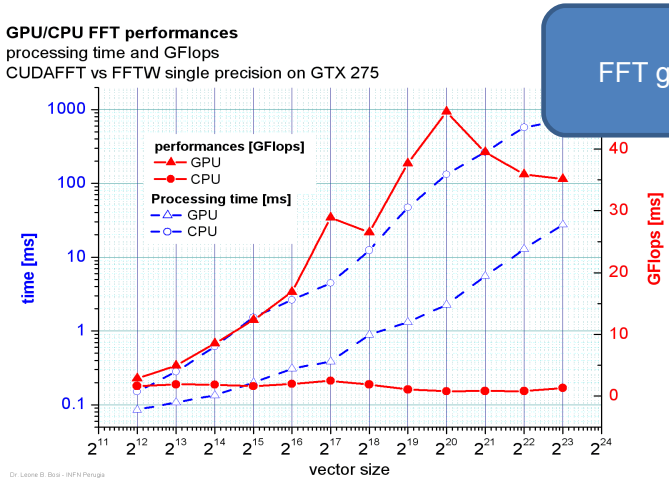


Pipeline detection of gravitational signals:

×50 gain
with GPU GTX275 device respect to a CPU of similar price.



CUDA/OpenCL Test and develop



FFT gain 50x

Lattice QCD and GPU

C. Bonati (Pisa), G. Cossu (KEK), M. D'Elia (Genova), A. Di Giacomo (Pisa)

The physical problem: QCD and confinement

Low energy QCD and confinement are intrinsically nonperturbative phenomena. In Lattice QCD a finite lattice is introduced as a nonperturbative gauge invariant regulator and observables are calculated by using Monte Carlo simulations.

The numerical problem

In a LQCD simulation a lot of linear systems $L \times L$ have to be solved, with $L \sim 10^5 \div 10^6$

$\Rightarrow$

Need for
dedicated machines
(apeNEXT, Blue Gene, ...)

Lattice QCD and GPU

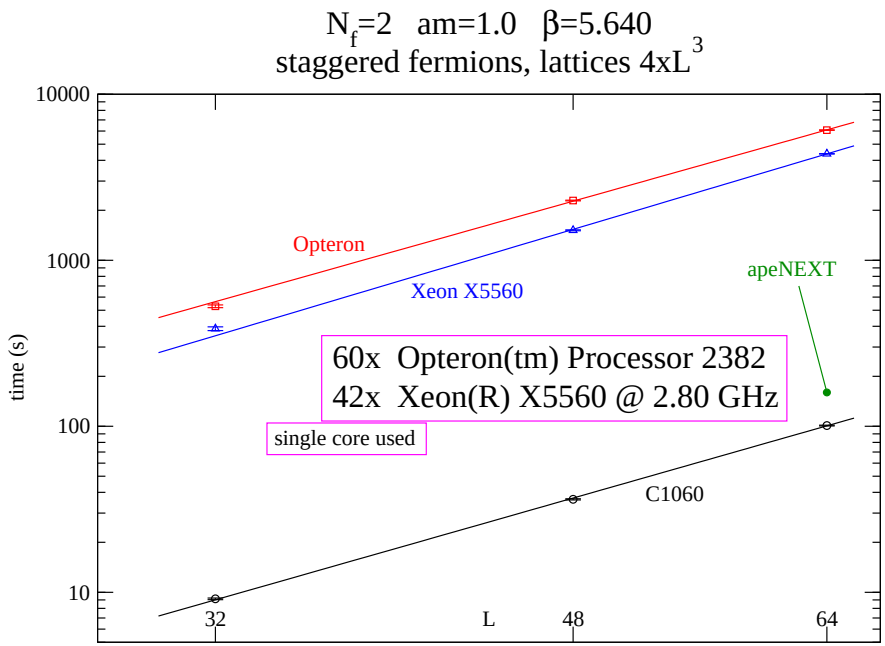
GPUs turned out to be low cost alternative to dedicated machines

With our C++/CUDA code, depending on parameters, we have

$$1\text{cpu} + 1\text{C1060} \sim 0.8 \div 1.5 \text{ apeNext crate} \sim \begin{array}{l}60 \text{ times faster} \\ \text{than 1 Opteron}\end{array}$$

$$1\text{cpu} + 1\text{C1060} \sim 2000\,€$$

tested for $N_f = 2$ and mass from $am = 0.01335$ to $am = 1.0$



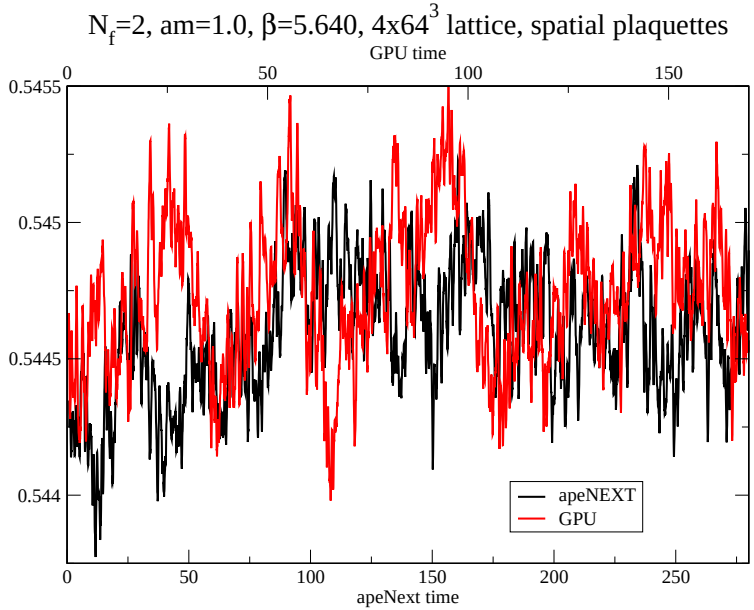
Lattice QCD and GPU

We have:

- Update for staggered fermions *completely* on GPU. This code is already used in production runs.

Wish list:

- improved staggered fermions
- overlap fermions
- parallel code for GPUs



Total times

apeNext	$\sim$ 280h
C1060	$\sim$ 170h

The QUonG project [APE RM1 - RM2 - Salina et al.]

Motivations of using many GPUs come from raw estimate of memory footprint:

- Full solver in GPU
- Gauge field + 15 fermion fields
- No symmetry tricks
- No half precision tricks

Lattice Size	SP (GiB)	DP (GiB)	# ATI HD 5870	# Tesla C1060	# Tesla C2070
$24^3 \times 48$	1	2.1	3	1	1
$32^3 \times 64$	3.3	6.7	$4-8$	2	$1-2$
$48^3 \times 96$	17	34	$17-35$	$5-9$	$3-6$
$64^3 \times 128$	54	108	$55-110$	$14-28$	$9-18$

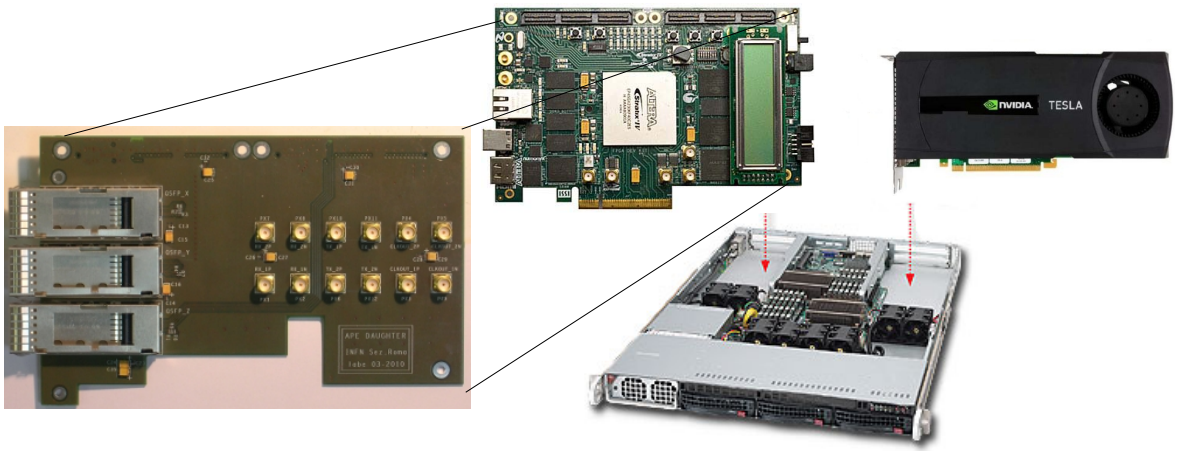
Many levels of parallelism are needed:

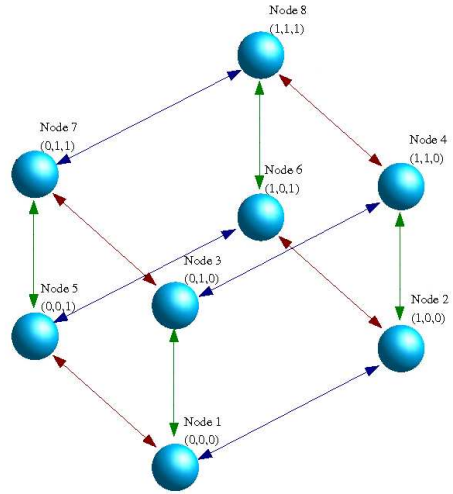
- intra-GPU (which in turns is divided in threads and blocks)
- eventually intra-node
- inter-node

Multi-GPU programming model: C + MPI + Linux threads + CUDA/OpenCL

Tying GPUs together with APENet+ [APE RM1-RM2]

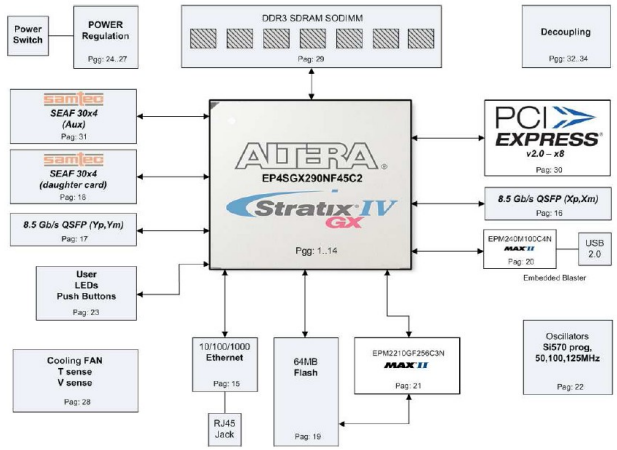
- From applications come a great demand for a high bandwidth and low latency networks
- A great improvement can be achieved if a GPU-to-GPU DMA could be available
- A custom hardware is in development that implements a 3D toroidal network
- It is based on the recent experience APENet and european project SHAPES

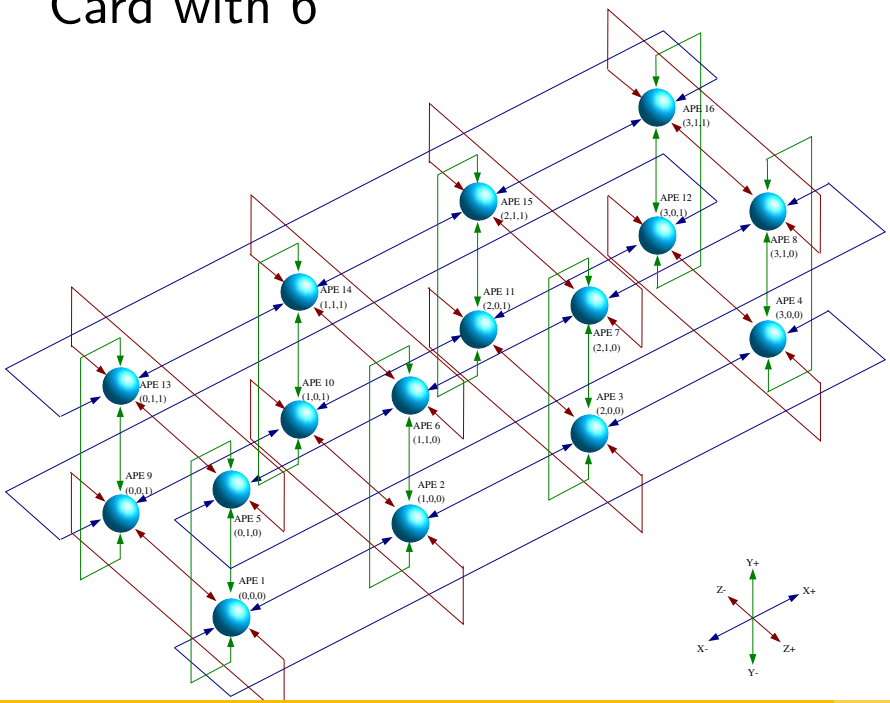


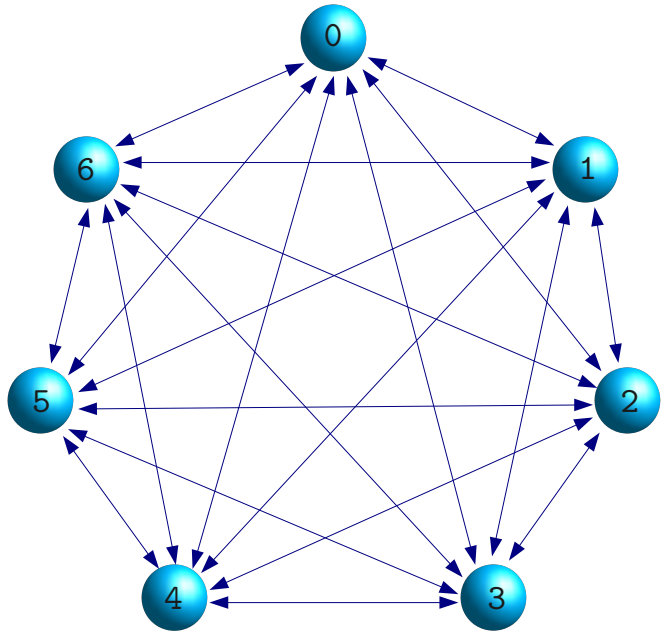


APENet+ characteristics

- 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)
- 1U Chassis Compliance
- Daugher Card with 3 links, Custom Card with 6







Conclusions and Outlooks

- Lot of indipendent work has been done so far.
- The lately born INFN GPU interest group is working on experience exchange, code development collaboration, centralization of computing services.
- The next step is to use APENet+ and its resources to aggregate specialized developments in GPU+MPI environment.
- A critical mass is required of users, hw & sw developers to move from an “Interest Group ”to a real (read funded) HPC project.