APE group

Mastering multi-GPU computing on a torus network

R. Ammendola, A. Biagioni, O. Frezza, F. Lo Cicero, A. Lonardo, P.S. Paolucci, D. Rossetti*, A. Salamon, G. Salina, F. Simula, L. Tosoratto, P. Vicini
(*) email contact: davide.rossetti@roma1.infn.it

INFN

Abstract

Many scientific computations need multi-node parallelism for both space (memory) and time (speed) requirements. The use of GPUs as accelerators introduces yet another level of complexity for the programmer and may potentially result in large overheads due to bookkeeping of memory buffers. Additionally, top-notch problems may easily require more than a PetaFlops of sustained computing power, requiring thousands of GPUs orchestrated via some parallel programming model, mainly Message Passing Interface (MPI). Here we describe APEnet+, the new generation[1] of our 3D torus network which scales up to tens of thousands of cluster nodes with linear cost. The basic component is a custom PCIe adapter with six high-speed links, designed around a programmable HW component (FPGA), a nice environment for studying integration techniques between GPUs and network interfaces. The high-level programming model is MPI, while a low-level RDMA API is also available. As of 3Q10, we are testing the first prototypes, waiting for commissioning a first production batch.

1 A *problem* of Physics

With first data coming out of the Large Hadron Collider (LHC) particle accelerator, precise theoretical predictions from the Standard Model (SM) of particle physics are necessary to match incoming experimental data, in a way to assess presence of physical processes beyond those predicted by the SM.
The Quantum Chromo Dynamics (QCD), describing the nuclear force in the SM, is a highly non-perturbative theory, so it needs some kind of regularization. Lattice QCD[2] is the most successful regularized theory, living on a discretized 4D space-time of volume $V=L_xL_yL_zL_t$ lattice points, and high-performance computers are needed to extract physical predictions from it.
The basic calculation is related to the solution of a linear problem:

$$\hat{M}\times\vec{x}=\vec{y}$$

Where *M* is a huge sparse matrix (non-zero values only along and near the diagonal) representing the nuclear interaction force, *y* is given vector related to a quark particle field, and *x* is the required solution. Most inversion algorithms use matrix-vector multiplication as the basic ingredient.
The extrapolation to the continuum physics involves repeated simulations at increasing lattice volumes $24^3{\times}48$, $32^3{\times}64$, $48^3{\times}96$, $64^3{\times}128$, $96^3{\times}192$, etc. The double-length along time dimension is related to the computation of certain physical quantities.

2 Going parallel

Computational demands roughly scale as L^6 at increasing lattice length (L), so parallelization is needed for all but the smallest volumes. In 2010 the estimated demands of a typical international scientific collaboration is well over 60 TeraFlops sustained over the full year for lattice sizes up to $V=96^3{\times}192$.
LQCD, as other fundamental theories, posses many internal symmetries which can be used to speed up the calculations. Among them there are:
- Isotropy, i.e. no special space-time locations, so the simulation lattice can be sliced into sub-domains and **distributed over many computing nodes** (Domain decomposition). Load balancing is automatically achieved.
- Locality, the interaction mainly involves neighboring space-time sites (i.e. *sparsity* of matrix M), thus **first-neighbor communications** between computing nodes are mostly needed.
Collective communications are used at key points in the algorithms to calculate a few extensive (i.e. global) quantities, but they only involves small sized buffers.
In a single numerical simulation, the minimum number of computing elements is related to the lattice size, mainly due to memory requirements. Many simulations at different lattice sizes are required due to the extrapolation to the continuum limit (see table 1).

Lattice size	DP Memory (GiB)	# Tesla C2070
$24^3{\times}48$	2.1	1
$32^3{\times}64$	6.7	2
$48^3{\times}96$	34	6
$64^3{\times}128$	108	18

Table 1: memory demand and required number of GPUs, at varying lattice size, including contribution of secondary support buffers, using double precision (DP).

3 The hierarchy of parallelism

Mapping an LQCD simulation on a GPU accelerated cluster requires mastering three different technologies throughout the hierarchy of parallelism:
• Coding the application kernels with **CUDA** or **OpenCL** to exploit the multiple cores of a single GPU.
• Extending to multiple GPUs on a single node by the use of multi-threading techniques: **OpenMP/pthreads** (or MPI with shared-memory communications).
• Going multi-node with some networking API, like **MPI**.
In the end, the application has to be written integrating different programming models: CUDA /OpenCL+ OpenMP/pthreads + MPI.
We note that communicating data between two GPUs on different nodes requires a three step procedure:
• cudaMemcpy() from GPU memory to some CPU temporary memory buffers.
• MPI_Send()/MPI_Recv()/MPI_Sendrecv() acting on those buffers.
• cudaMemcpy() from CPU buffers to GPU memory.
Without the use of NVIDIA GPUdirect technology, two additional memory copies are necessary on modern interconnect like APEnet+ and Infiniband.

Bibliography

[1] First generation APENet is described in arXiv:hep-lat/0409071.
[2] For an overview of Lattice QCD, arXiv:1002.4232v2 and references therein.
[3] APEnet web site is http://apegate.roma1.infn.it/APE

Acknowledgments

We really thanks NVIDIA Corp. for contributing keys technologies for APEnet+-GPU optimizations and especially to Massimiliano Fatica, for his restless support and useful discussions.

4 The APEnet+ network

• APEnet+ is a packet-based direct network with 2D/3D torus topology.
• Packets have a fixed size envelope (header+footer) and are auto-routed to their final destinations according to dimension-order static routing, with dead-lock avoidance.
• Error detection is implemented via CRC at packet level.
• Basic RDMA capabilities, PUT and GET, are implemented at the firmware level.
• Fault-tolerance features will be deployed during 2011.

5 The hardware of the APEnet+ card

Depending on the cluster dimension and requirements, the card can be assembled in two ways:
• Basic one, single slot width, 4 torus links, 2D torus topology.
• Secondary Piggy-back card, resulting in a double slot width, 6 links, 3D torus topology.
The APEnet+ card plugs into a PCIe X16 slot but has signaling capabilities for up to X8 Gen2 (peak **4+4 GB/s**).
Each torus link is fully bidirectional and its **raw bandwidth is 34Gb/s** per direction on 4 lanes using QSFP+ cables.
The expected power envelope is 80 Watts.

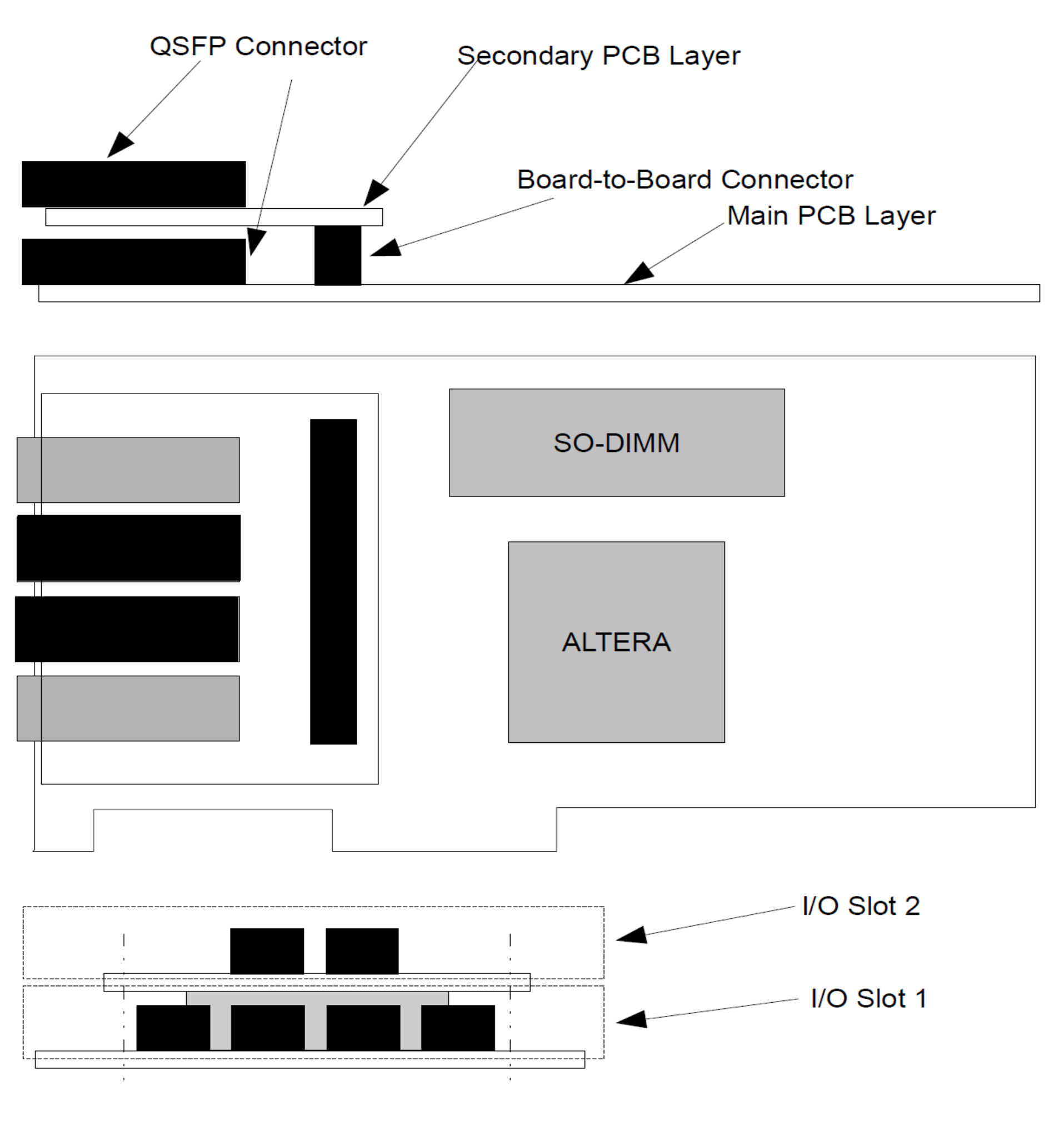

Figure 1: Three views of the APEnet+ card, showing the secondary board which plugs on top of the main on to add two more links, thus enabling the 3D Torus with 6 links in total.

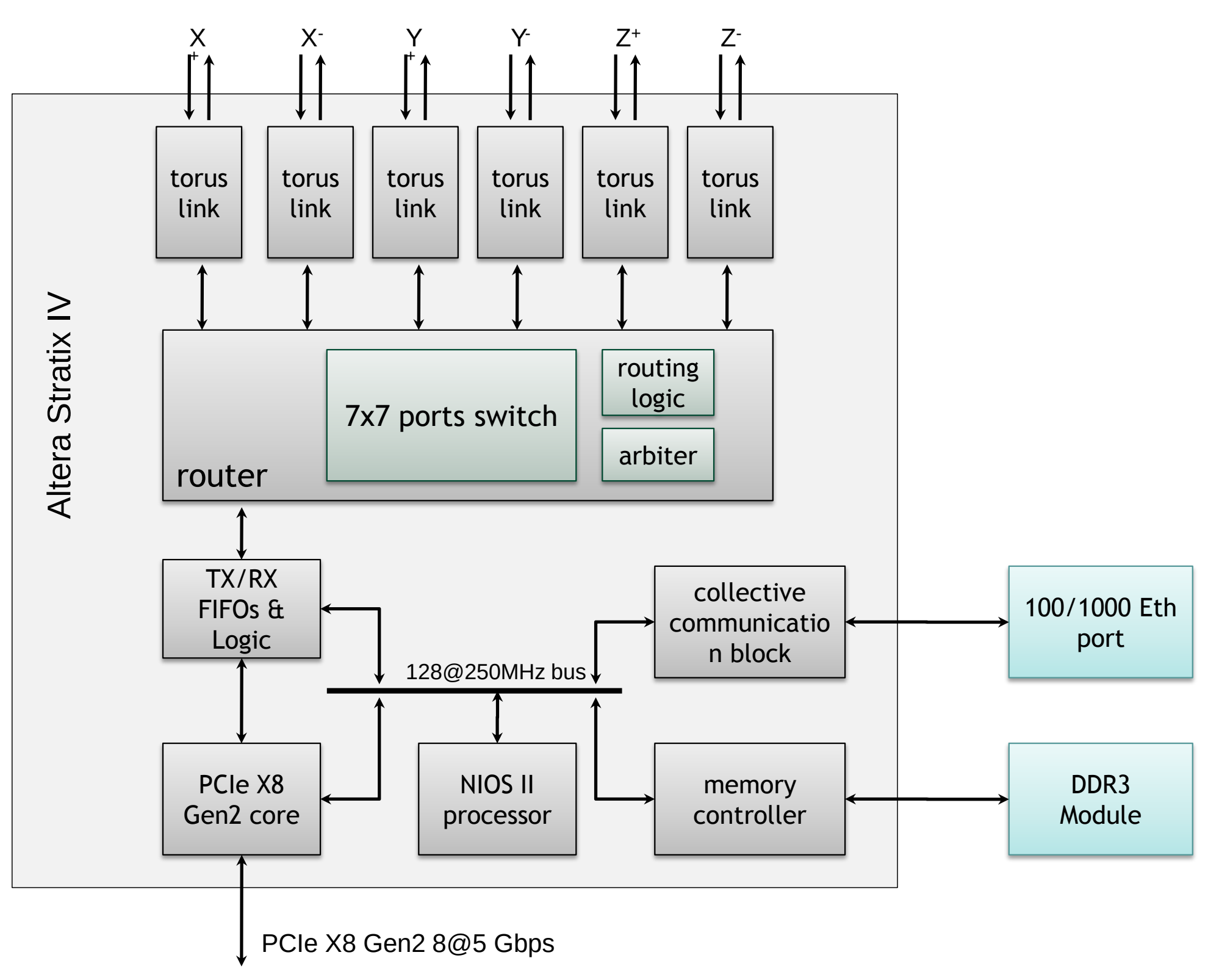

Figure 2: internal FPGA block architecture.

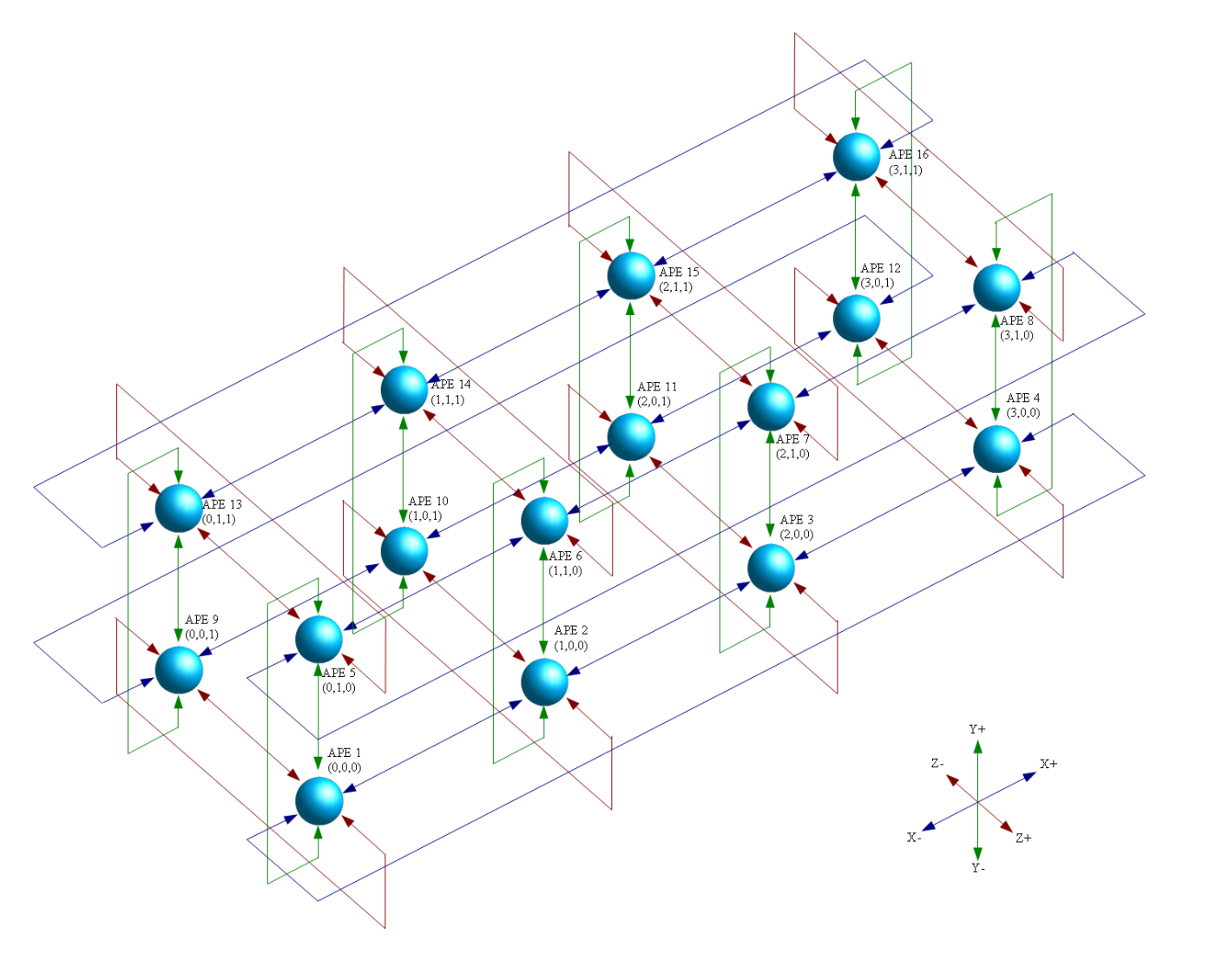

Figure 3: a pictorial representation of a 16 nodes cluster, arranged as a 4x2x2 torus.

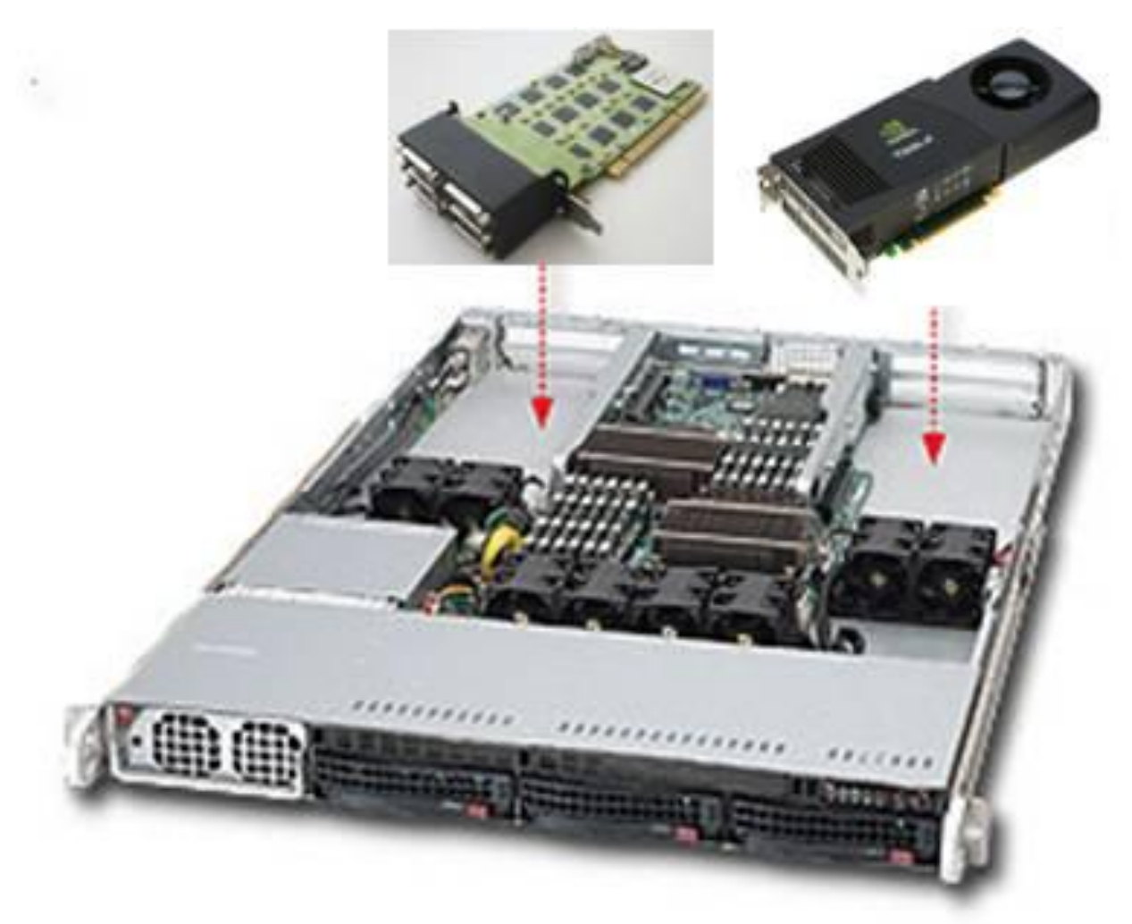

Figure 4: a sample 1U system with an APEnet+ card and a GPU.

Figure 5: APE128 is a previous generation APEnet cluster.

6 Logic architecture

The card firmware is programmed in VHDL and synthesized for the Altera Stratix IV FPGA part. It resides on an on-card flash memory and can be updated at run-time via either the PCIe or an external USB cable.
The majority of the logic modules are custom developed while the PCIe core is a commercial one.
High level functionalities, like RDMA tables look-up, are carried on by a program running on an FPGA embedded processor (NIOS II), which uses the DDR3 module as its program and data memory.
The firmware block structure, depicted in the figure on the left, is split into a so called *network interface* (PCIe ,TX/RX logic, NIOS II processor, etc…) and a *router* (router component and torus links).
The router comprises a fully connected, 7 ports in, 7 ports out switch, plus routing and arbitration blocks:
• The routing block examines a packet header and resolves the destination address in a proper path across the switch. It supports the dimension ordered routing algorithm, with a routing latency of 60ns.
• Deadlock avoidance is implemented via the virtual channels technique, with 2 receive buffers on the torus link module.
• Proper flow control is maintained via handshake of credits between a local RX block and the remote TX block, embedded in the link protocol data layer.

7 Assembling the torus

The direct network design is **intrinsically scalable** as there is no **need of external switch** components, which rapidly becomes expensive as the number of nodes goes beyond a few thousands nodes.
The packet protocol provides space for 256 nodes for each dimension, i.e. theoretically **up to 2^{24} computing nodes.**
In figure 4 we depict one possible arrangement for a computing node: a 1U dual-socket system with one GPU and one APEnet+ card. Just to give an idea of the cabling of a torus network (see figure 3), we added figure 5 which is a shot of a 128 node cluster equipped with a previous generation of APEnet cards assembled in 2005.
Thanks to the flexibility in the APEnet+ HW design, we can build both 2D and 3D torus networks, as well as other less common topologies suited to specific applications.
The resulting cluster can be dynamically split into sub-partitions, to adapt to applications and users demands. Partition choice is done at invocation of a modified `mpirun` program, e.g. `aperun –topo cube222 myprg`., which can be easily integrated with your choice of batch queuing system.

8 APEnet+ programming

All APEnet+ software runs under Linux and is available under the GNU GPL Licence. Two sets of programming APIs are available, one is MPI and the other is a low-level custom RDMA one.
The RDMA APIs are avaliable as a C language library:
- Communication primitives available to applications are: `rmda_put()`,`rdma_get()`, `rdma_send()`.
- Buffer registration allows for exposing memory buffers to RDMA primitives: `register_buffer()`,`unregister_buffer()`.
- Events are routed to applications whenever RDMA primitives are executed by APEnet+: `wait_event()`.
We adapted OpenMPI 1.X to APEnet+ developing an adaptation (BTL) module, which is implemented on top of the RDMA API.

9 GPU optimizations and future work

We are currently working on some GPU related optimizations:
- GPU-initiated communications: we are exploring the architectural extensions needed to implement **CUDA version of the APEnet RDMA transmission primitives**, e.g. CUDA version of rdma_put(), using so called PCIe peer-to-peer transactions and avoiding intermediate copies onto CPU memory buffers.
- APEnet RDMA events delivery: we are implementing **a RDMA event queue on GPU memory**, written in HW by the APEnet firmware and read in CUDA kernels.
- RDMA to GPU device memory: exploring usage of the **GPU memory as an endpoint of RDMA transactions** (PUT or GET), implies the ability for the APEnet+ DMA engine to read/write GPU memory completely in HW, without CUDA API calls on the CPU side.
The presence on the APEnet+ card of a programmable component with a lot of free resources will allows us to explore **reconfigurable computing**, i.e. accelerating some tasks directly in hardware.