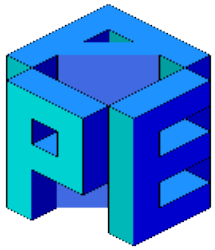
A distributed model of computation for reconfigurable devices based on a streaming architecture

Paolo Cretaro
National Institute for Nuclear Physics



FPL 2019
Barcelona, September 2019

The ExaNeSt project: hardware highlights

Unit: Xilinx Zynq Ultrascale+ FPGA

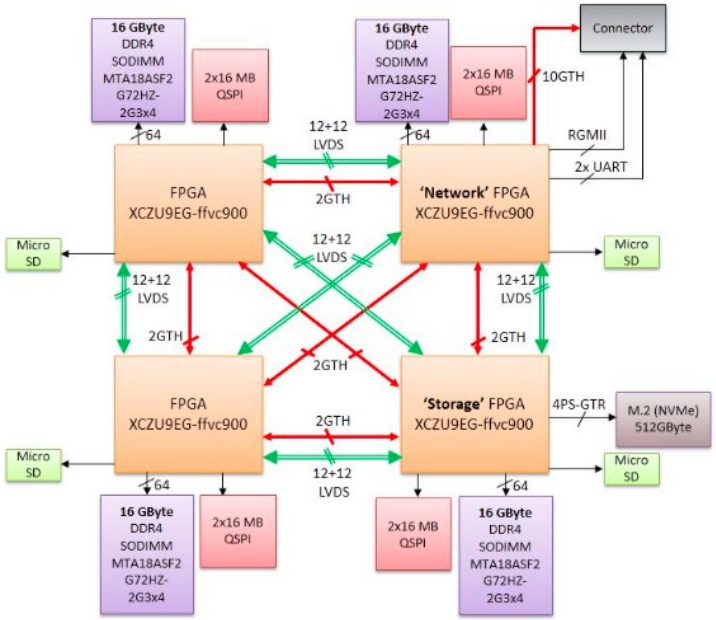
- Four 64bit ARM Cortex A53 @1.5GHz
- Programmable logic
- 16 high speed serial links @16Gbps

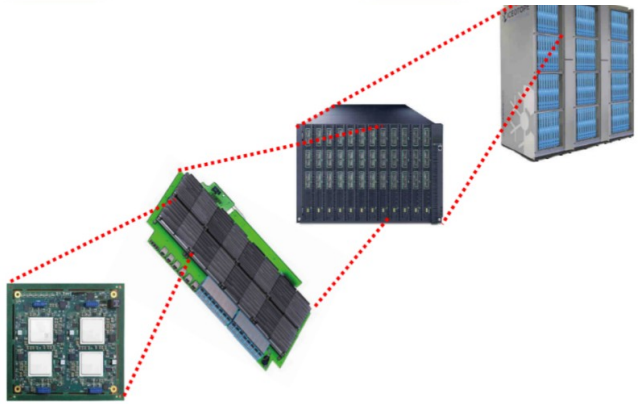
Node: Quad-FPGA Daughter-Board (QFDB)

- All-to-all internal connectivity
- 10 HSS links to remote QFDB (through network FPGA)
- 64 GB DDR4 RAM (16GB per FPGA)
- 512 GB NVMe SSD on storage FPGA

Blade/mezzanine

- 4 QFDB in Track 1
- 2 HSS links per edge (local direct network)
- 32 SFP+ connectors for inter-mezzanine hybrid network

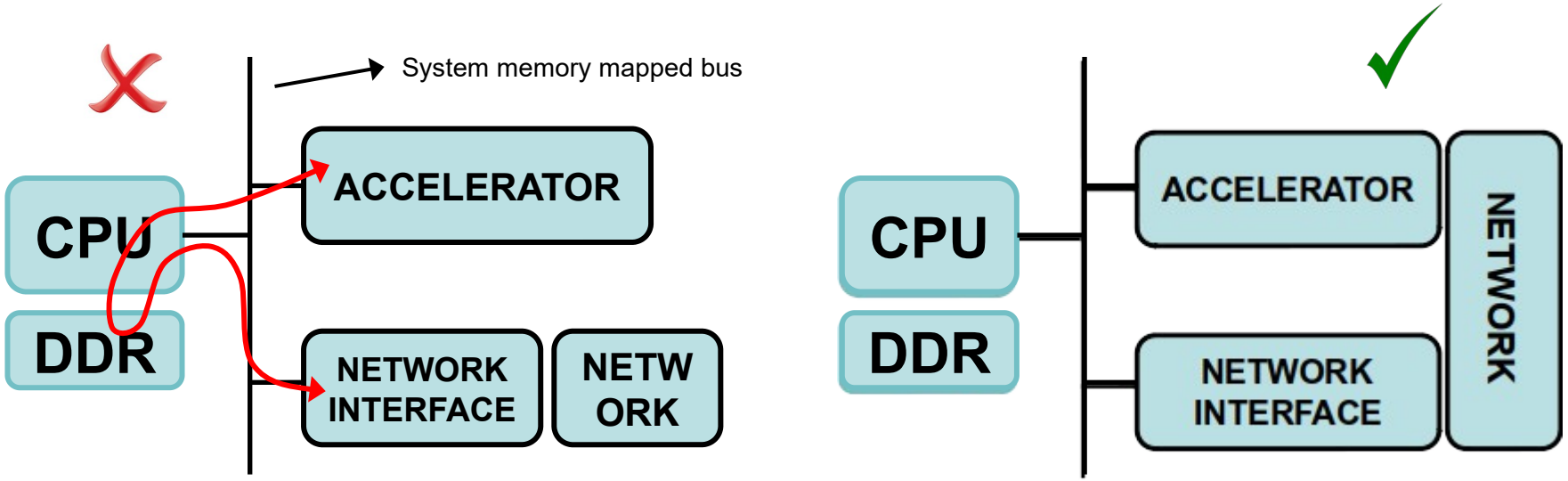




I worked in the team who made the 3D torus network, based on a custom Virtual Cut-Through protocol

Mixing acceleration and network

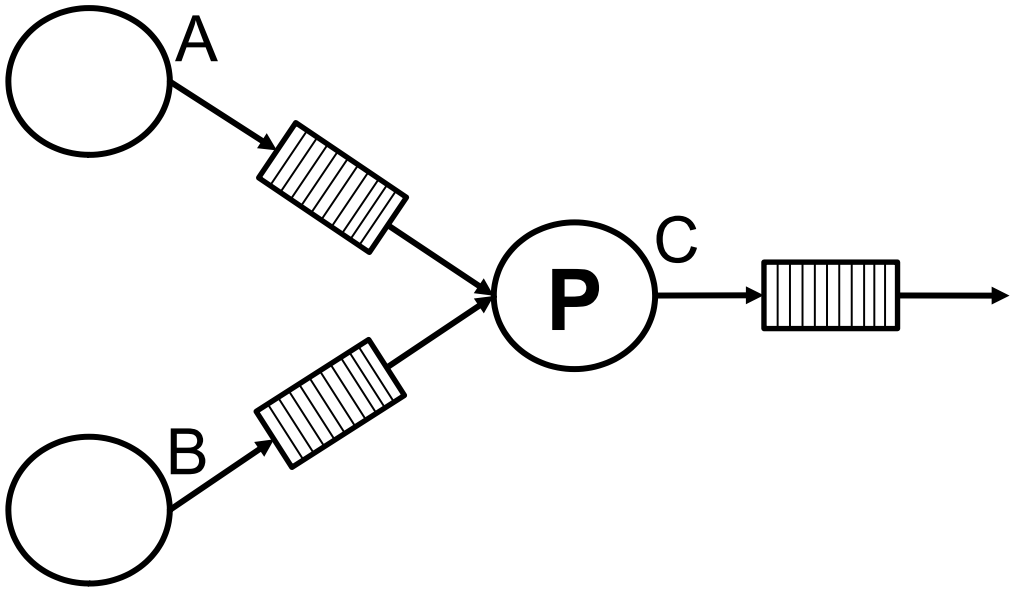
- With High Level Synthesis tools, FPGAs are becoming a viable way to accelerate tasks
- Accelerators must be able to access the network directly to achieve low-latency communication among themselves and other remote hosts
- A dataflow programming paradigm could take advantage of this feature to optimize communication patterns and loads



Kahn processing networks advantages

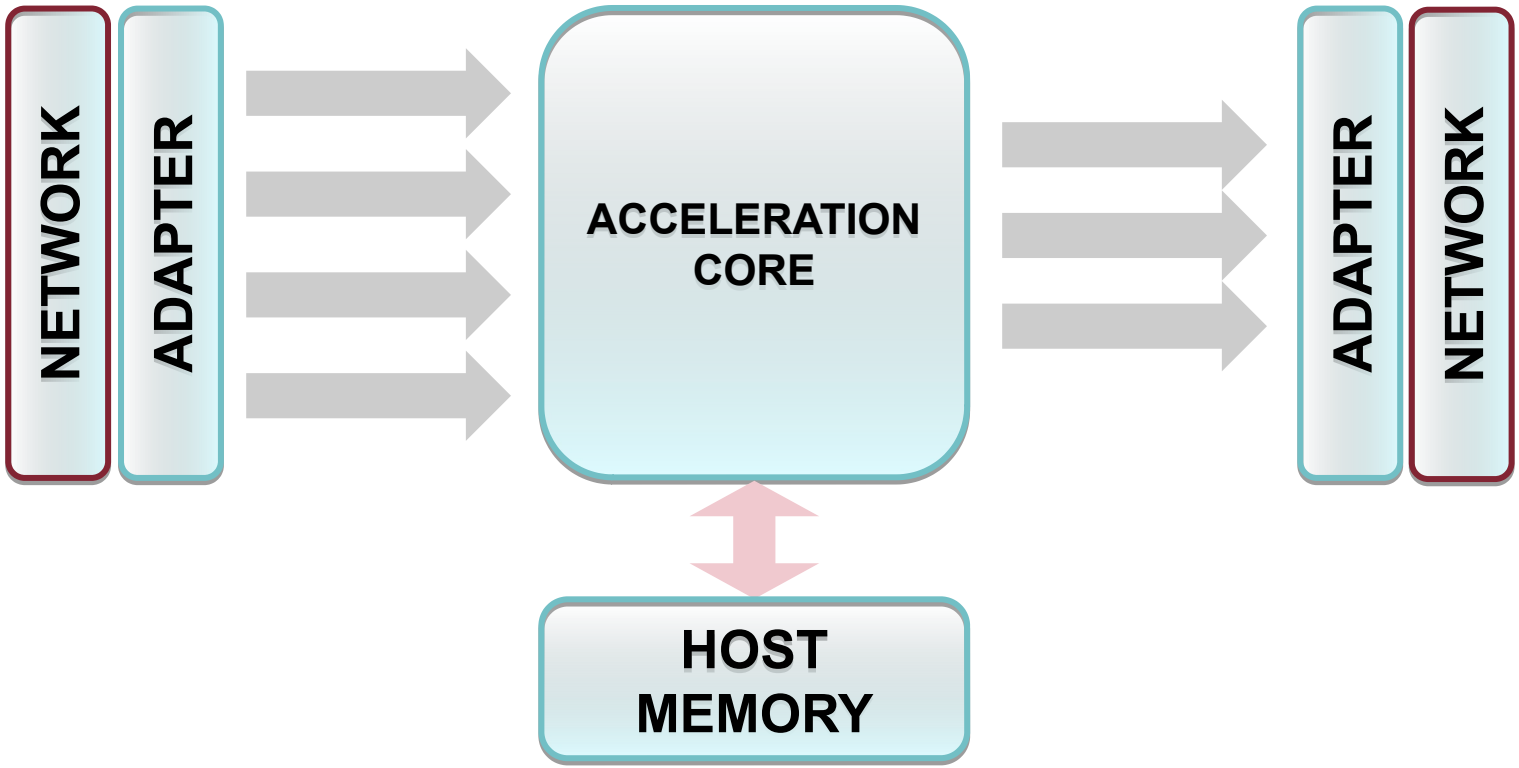
Group of sequential processes communicating through FIFO channels

- Determinism: for the same input history the network produces exactly the same output
- No shared memory: processes can run concurrently and synchronize through blocking read on input channel FIFOs
- Distributing tasks on multiple devices is easy



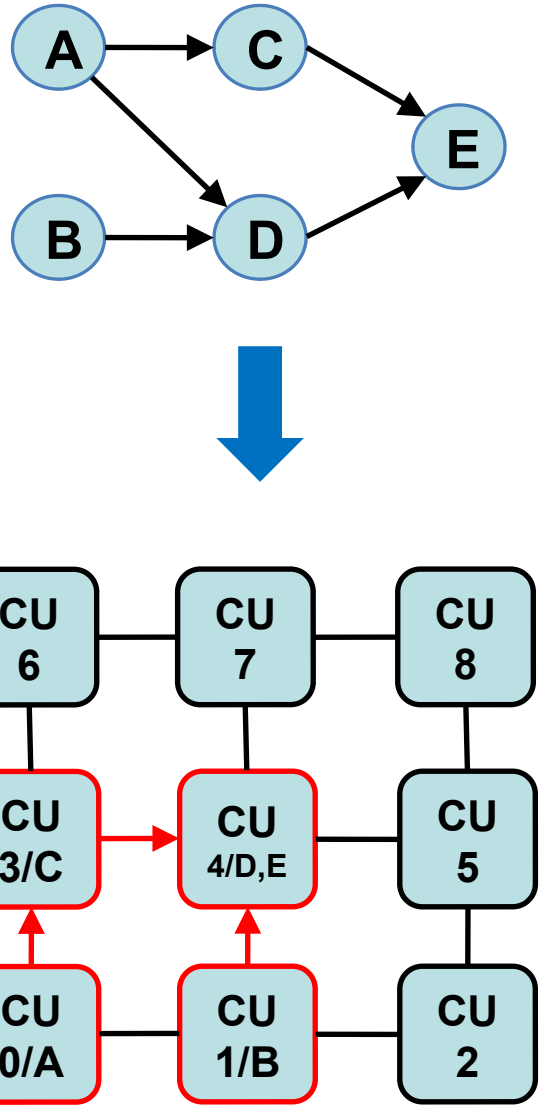
Accelerator hardware interface

- Virtual input/output channels for each source/destination
- Direct host memory access for buffering and configuration (a device driver is needed)
- Direct coupling with the network



Steps description

1. Write kernels in HLS
2. A config file delineates tasks and data dependencies
3. A directed graph is built and mapped on the network topology
4. Accelerator blocks are flashed on targeted nodes
5. Data is fed into entry points and tasks are started
6. Each task consumes its data and send the results to the next ones



Simplified task graph configuration example

```
Device0 {
     Type: FPGA
     Task0 {
          Impl: source_task.c
          Input_channels: 0
          Output_channels {
               Ch0: Device1.Task0.Ch1
          }
     }
     Task1 {
          Impl: source_task.c
          Input_channels: 0
          Output_channels {
               Ch0: Device1.Task0.Ch0
          }
     }
}
Device1 {
     Type: FPGA
     Task0 {
          Impl: example_task.c
          Input_channels: 2
          Output_channels {
               Ch0: Device1.Task1.Ch0
          }
     }
     Task1 {
          Impl: sink_task.c
          input_channels: 1
     }
}
```

Thank you!