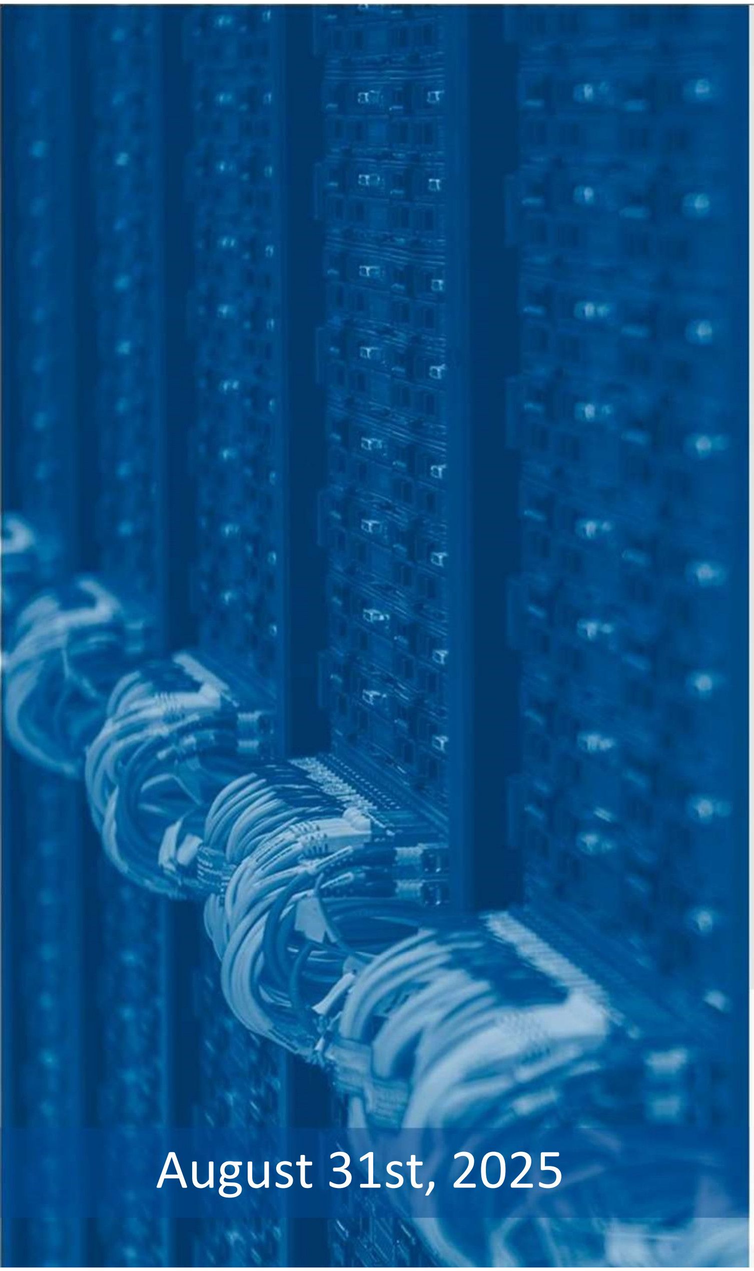


Accelerating Applications on Networks of FPGAs

Antonio Filgueras, Juan Miguel de Haro Ruiz, Carlos Álvarez Martínez, Daniel Jiménez-González, **Xavier Martorell**

Summer School on Supercomputing & AI for Tech Entrepreneurs

Namal University, Mianwali

August 31st, 2025

Introduction

- Heterogeneous systems are everywhere →HPC applications need more performance
- Besides GPUs, FPGAs are an interesting candidate
  - Highly flexible →very specialized code that can adapt to any type of workload
  - Fine-grained optimizations at the hardware level
  - Very power efficient
  - With the rise of High-Level Synthesis, they are closer than ever to non-FPGA experts
- However, managing FPGAs at a large scale is still quite new and they lack a mature ecosystem
  - Programmability is still an issue

We want to take advantage of FPGAs at large scale by providing an easy-to-use programming model and automatic tools to manage FPGA clusters

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

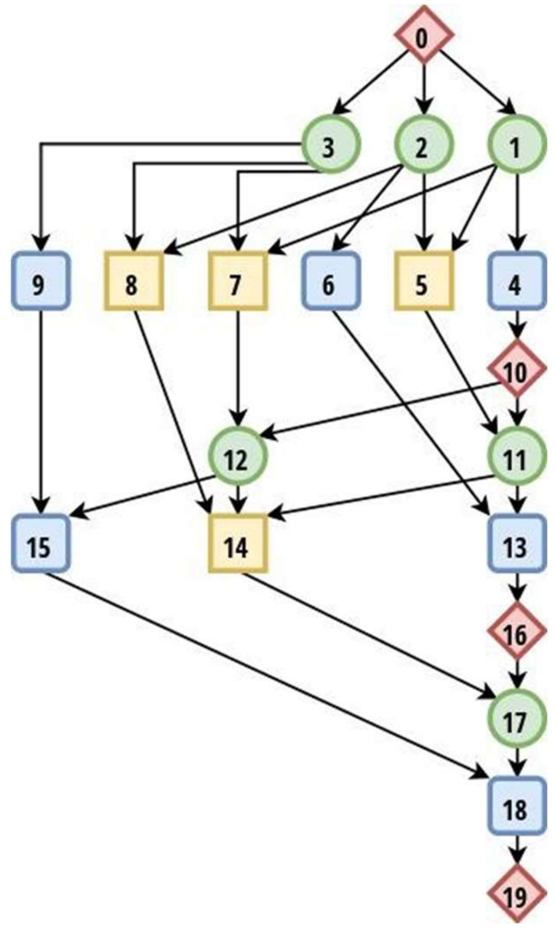
Introduction

OmpS-2 task-based programming model

Cholesky source code

```
void cholesky_blocked(const int nt, float *A[nt][nt])
{
   for (int k = 0; k < nt; k++) {
      #pragma oss task inout(A[k][k])
      potrf( A[k][k] );
      for (int i = k+1; i < nt; i++) {
         #pragma oss task in(A[k][k]) inout(A[k][i])
         trsm( A[k][k], A[k][i] );
      }
      for (int i = k+1; i < nt; i++) {
          for (int j = k+1; j < i; j++) {
             #pragma oss task in(A[k][i], A[k][j]) inout(A[j][i])
             gemm( A[k][i], A[k][j], A[j][i] );
          }
      }
      #pragma oss task in(A[k][i]) inout(A[i][i])
      syrk( A[k][i], A[i][i] );
   }
}
```

Task graph



Introduction

OmpSs@FPGA

- Extension to OmpSs-2 to offload tasks to FPGA accelerators using High-Level Synthesis (HLS).
- Framework that automatically extracts FPGA code from OmpSs pragmas and generates host executable + bitstream

FPGA task declaration

```
#pragma oss task device(fpga) in(pbi[0], pbj[0]) inout(fb[0])
void calculate_forces(particles_t *pbi, particles_t *pbj, forces_t *fb);
#pragma oss task device(fpga) in(pbi[0], pbj[0]) inout(fb[0])
void update_particles(particles_t *pb, forces_t *fb);
#pragma oss task device(fpga) inout(p[0], f[0])
void nbody(particles_t *p, forces_t *f, int nb, int steps)
{
   for (int k = 0; k < nt; k++) {
      for (int i = 0; i < nb; ++i)
         for (int j = 0; j < nb; ++j)
            calculate_forces(p+i, p+j, f+j);
      for (int i = 0; i < nb; ++i)
         update_particles(p+i, f+i);
   }
   #pragma oss taskwait
}
```

Code processed by HLS tool, transformed to HW accelerator

How can we distribute this code on an FPGA cluster?

FPGA hardware

- ❑ FPGAs from 10000 feet...
- ❑ ... and getting closer
- ❑ FPGA components
- ❑ Generating FPGA configurations

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

FPGAs: 10000 feet view

- The boards

Discrete (to be connected to PCIe)

Integrated (standalone)

source: docs.xilinx.com

source: www.axiom-project.eu

FPGAs: 10000 feet view

- The boards

Discrete (to be connected to PCIe)

AMD-Xilinx
- Versal
- Alveo
- VCU128
- ...

Intel-Altera
- Agilex
- Stratix
- Arria
- ...

source: docs.xilinx.com

Integrated (standalone)

AMD-Xilinx
- Zynq 7000
- Zynq Ultrascale+
- ...

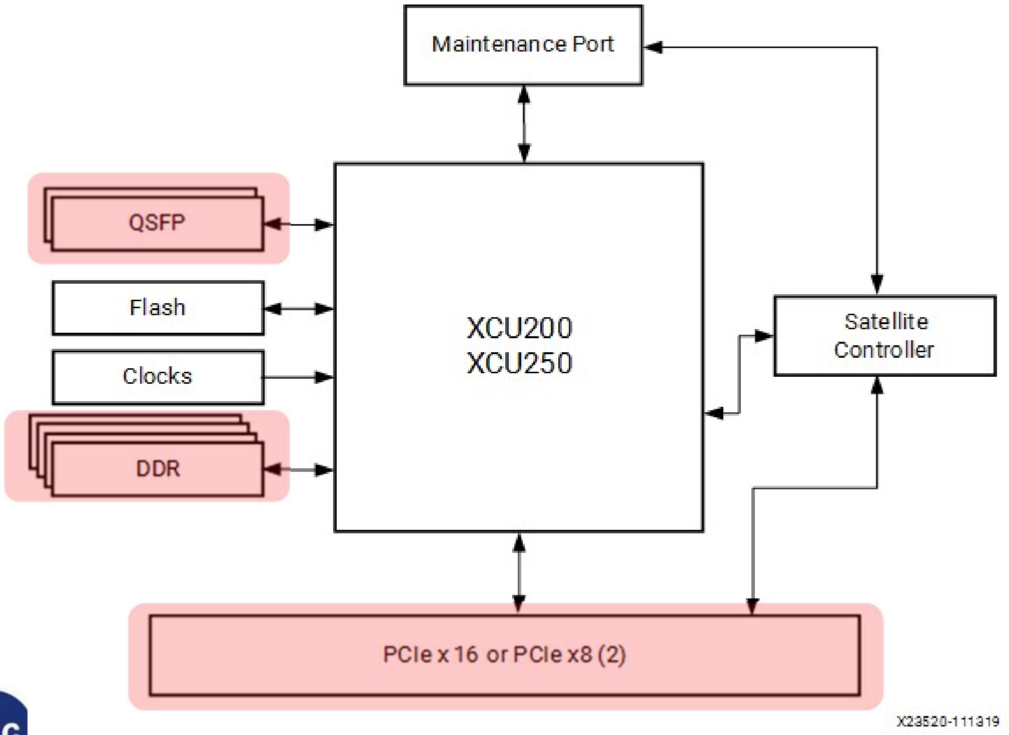
Intel-Altera
- Cyclone
- Max
- ...

source: www.axiom-project.eu

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

FPGAs: 10 feet view

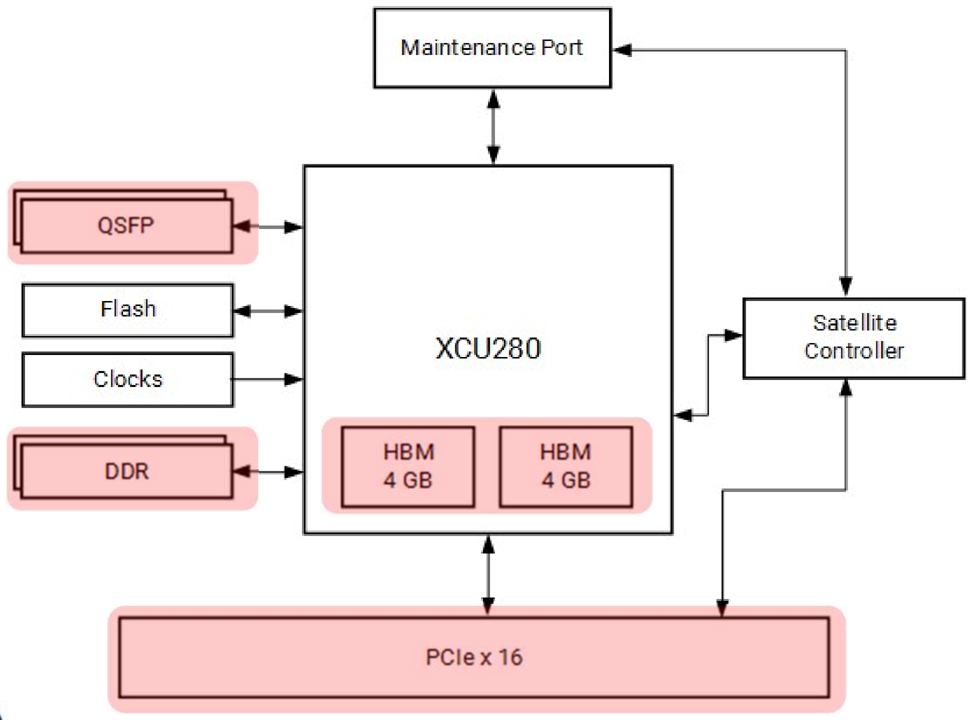
- Alveo U200 card basic hardware blocks

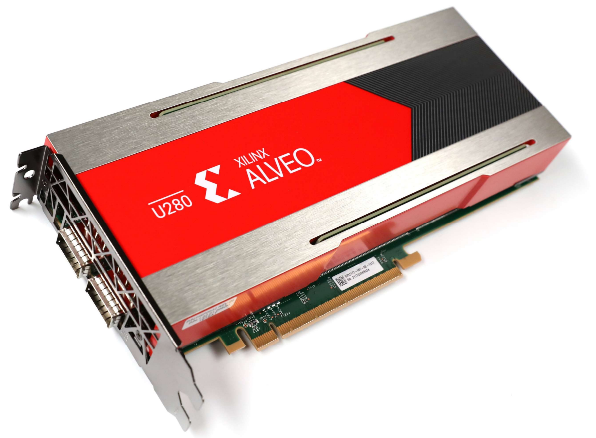


source: docs.xilinx.com

FPGAs: 10 feet view

- Alveo U280 card basic hardware blocks

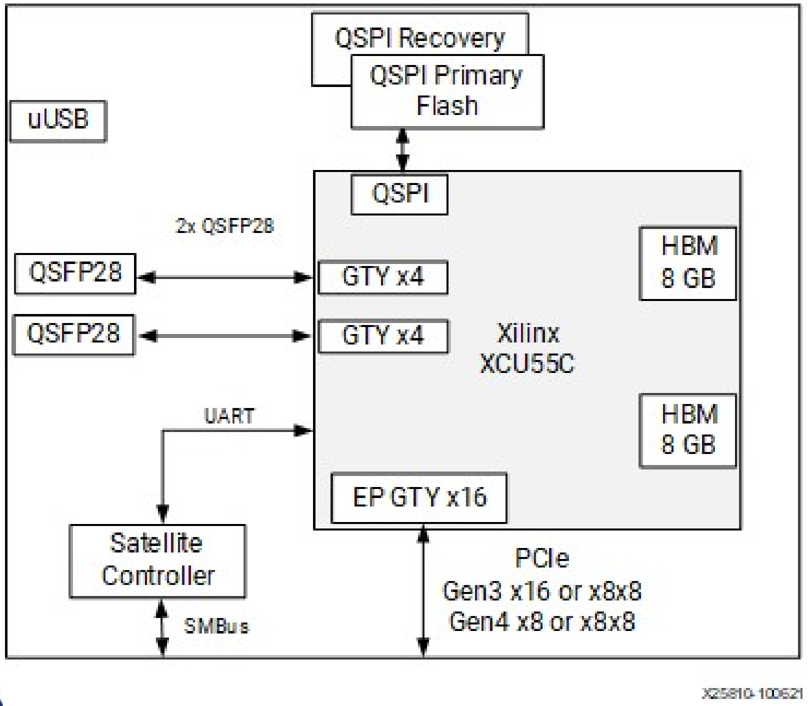




source: docs.xilinx.com

FPGAs: 10 feet view

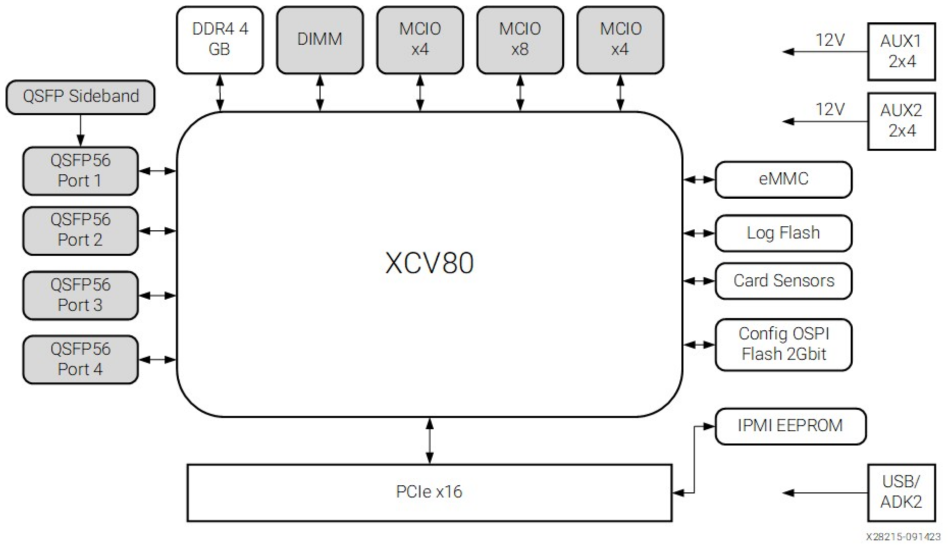
- Alveo U55c card basic hardware blocks

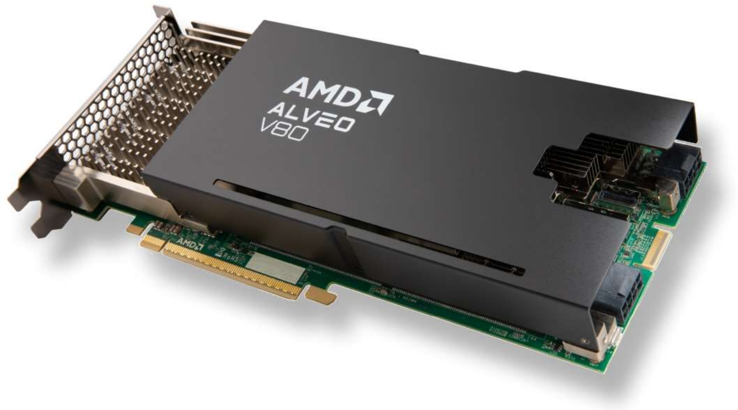


source: docs.xilinx.com

FPGAs: 10 feet view

- Alveo V80 card basic hardware blocks



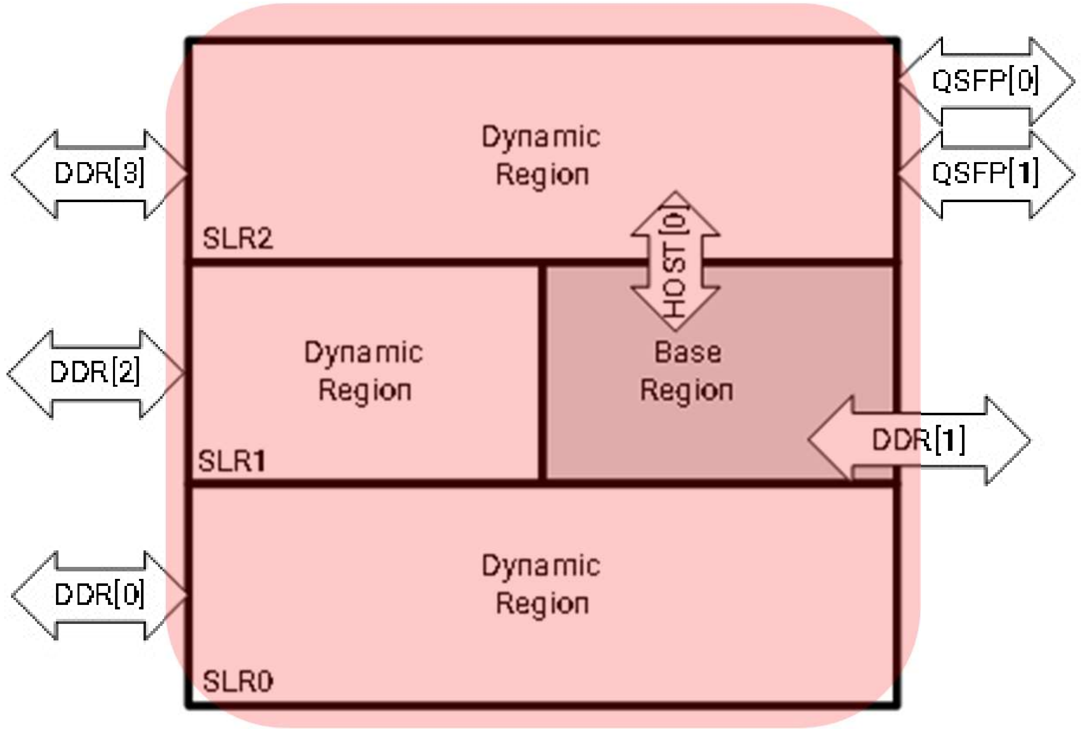


source: docs.xilinx.com

FPGAs: 10 inches view

- U200 FPGA connectivity & layout
  - DDR -> Global memory (RAM)
  - QSFP -> Ethernet 100Gb.
  - PCIe -> to host
  - USB -> serial line, jtag

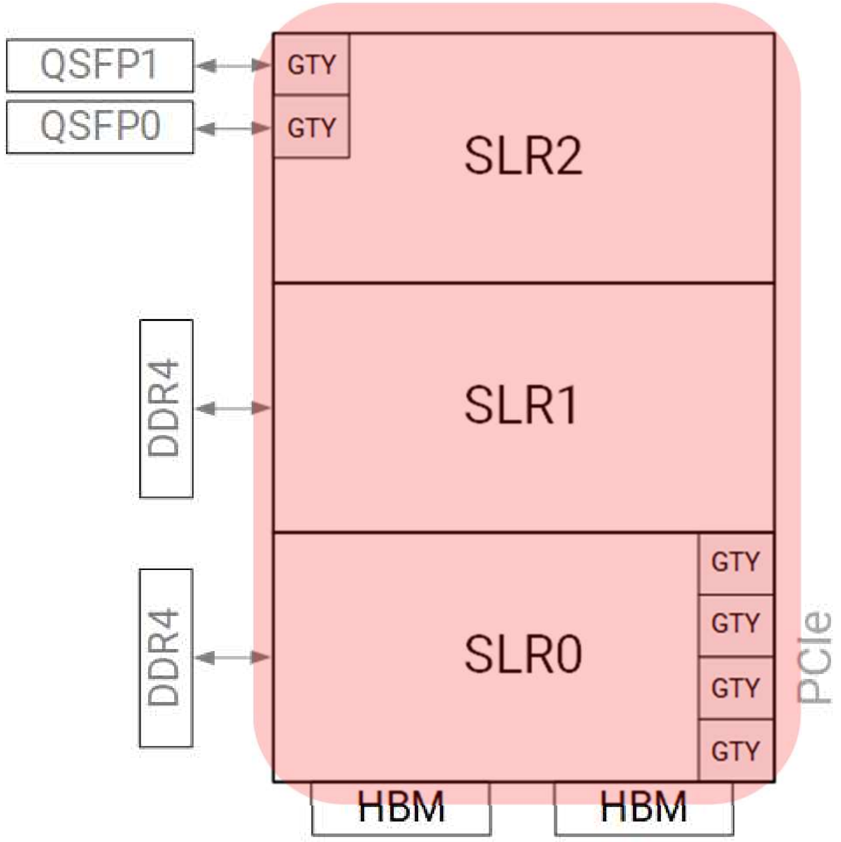
  - Super Logic Regions (SLR)
    - Logic partitions of the area of the FPGA
    - Connectors attached to SLRs



source: docs.xilinx.com

FPGAs: 10 inches view

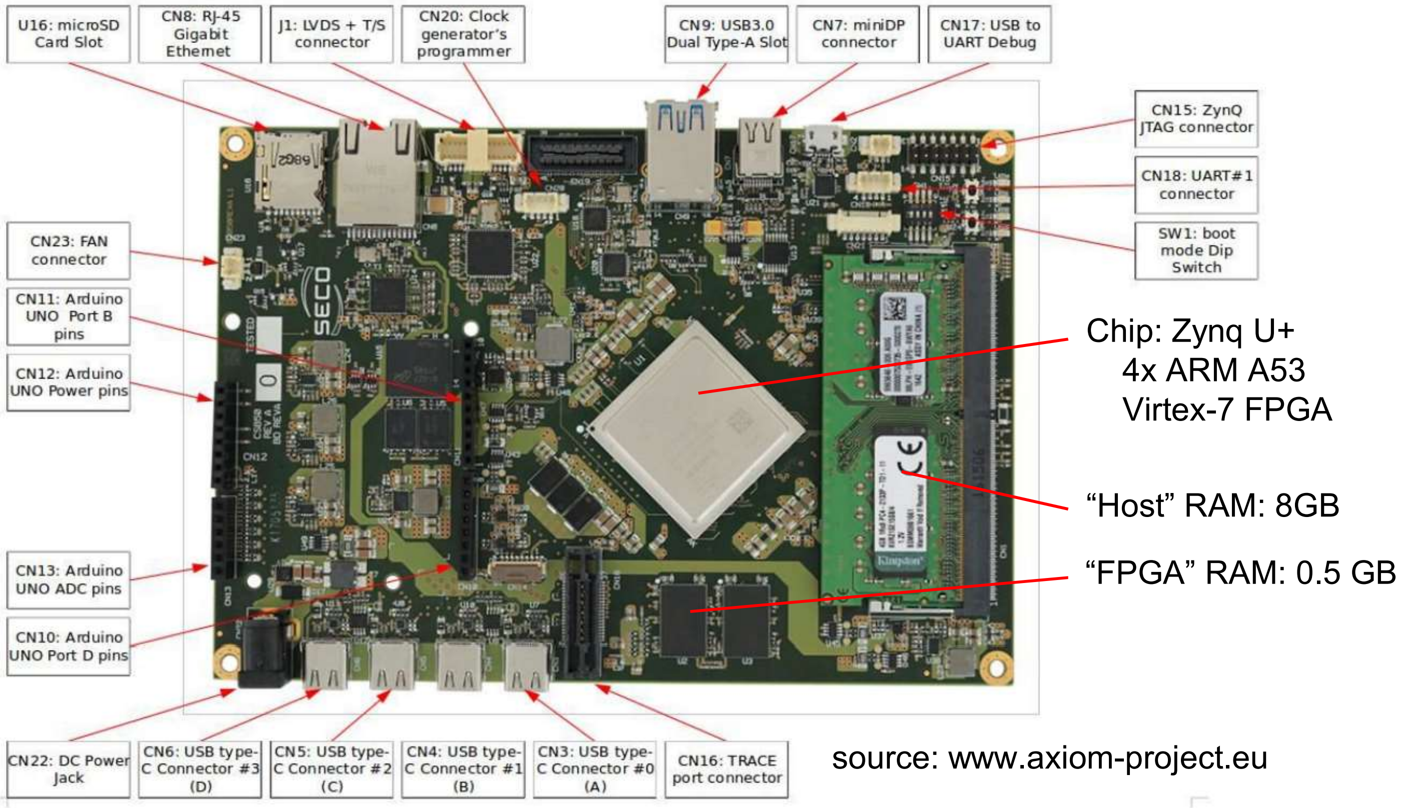
- U280 FPGA connectivity & layout
  - DDR -> Global memory (RAM)
  - HBM -> Global memory (HBM)
  - QSFP -> Ethernet 100Gb.
  - PCIe -> to host
  - USB -> serial line, jtag



source: docs.xilinx.com

FPGAs: 10 inches view

- Axiom board connectivity and layout

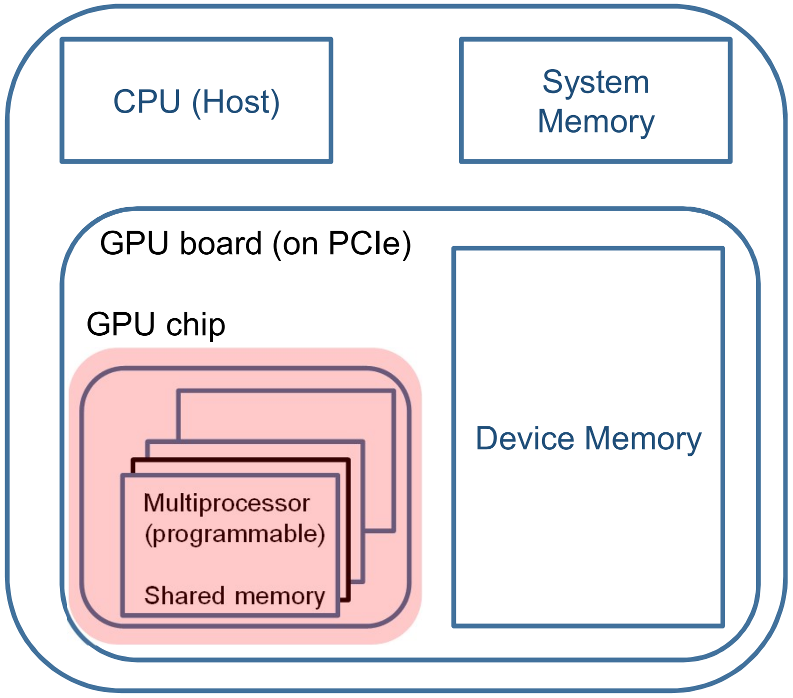


U16: microSD Card Slot
CN8: RJ-45 Gigabit Ethernet
J1: LVDS + T/S connector
CN20: Clock generator's programmer
CN9: USB3.0 Dual Type-A Slot
CN7: miniDP connector
CN17: USB to UART Debug
CN15: ZynQ JTAG connector
CN18: UART#1 connector
SW1: boot mode Dip Switch
CN23: FAN connector
CN11: Arduino UNO Port B pins
CN12: Arduino UNO Power pins
CN13: Arduino UNO ADC pins
CN10: Arduino UNO Port D pins
CN22: DC Power Jack
CN6: USB type-C Connector #3 (D)
CN5: USB type-C Connector #2 (C)
CN4: USB type-C Connector #1 (B)
CN3: USB type-C Connector #0 (A)
CN16: TRACE port connector
Chip: Zynq U+
4x ARM A53
Virtex-7 FPGA
"Host" RAM: 8GB
"FPGA" RAM: 0.5 GB

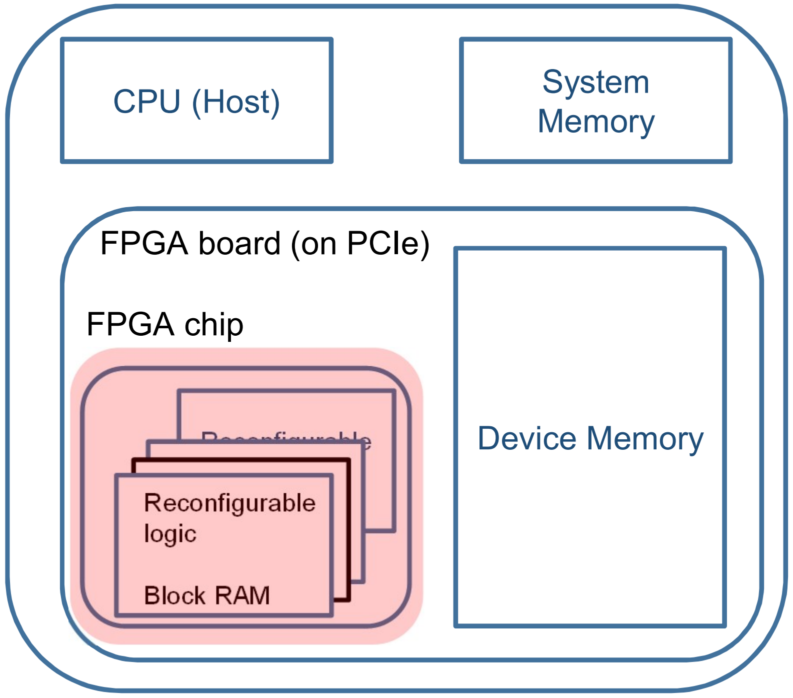
source: www.axiom-project.eu

Comparing FPGAs to GPUs

- “Basically”, replace the GPU compute device with the FPGA device



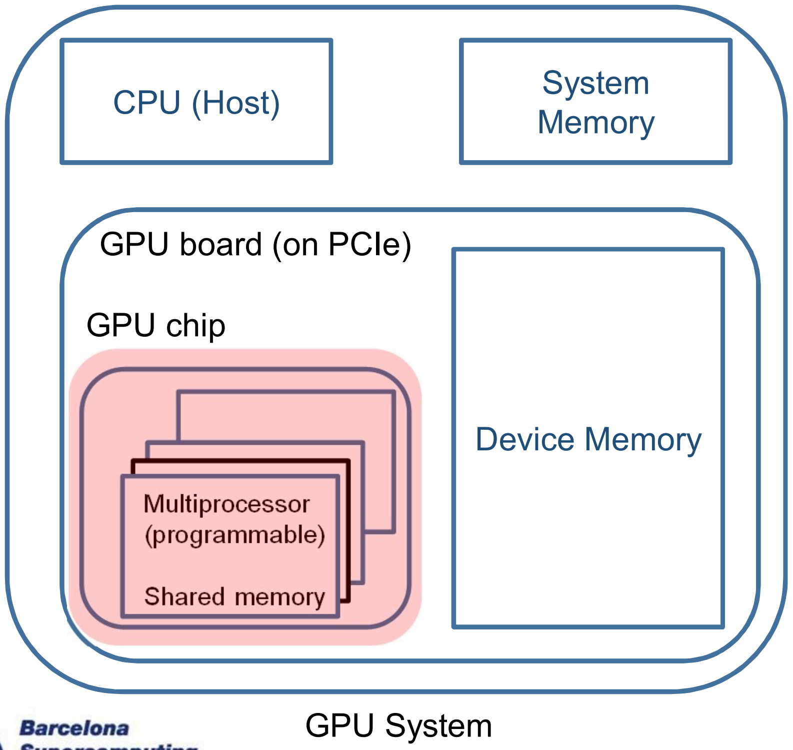
GPU System

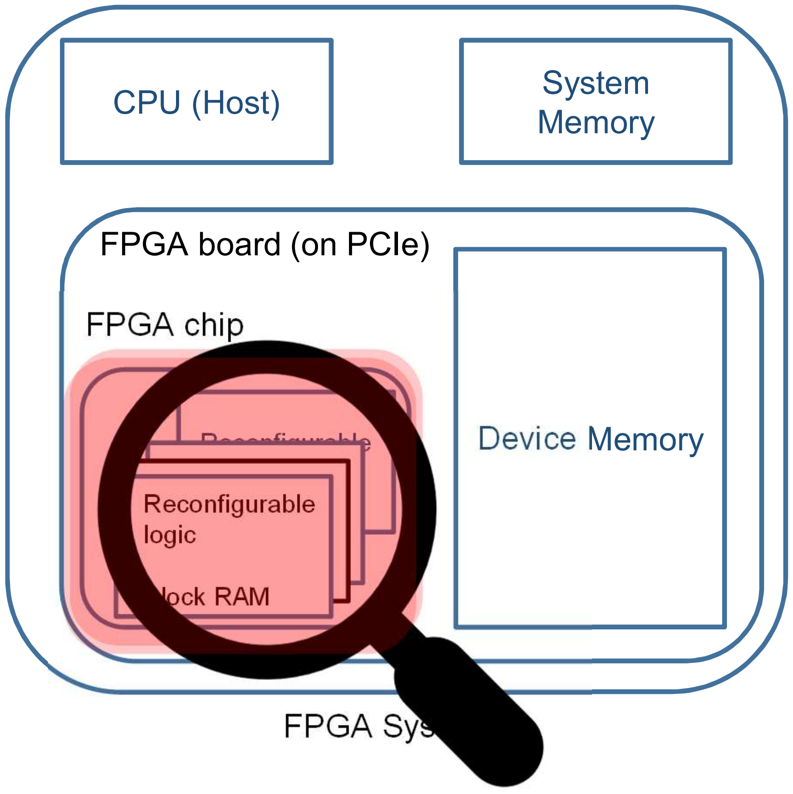


FPGA System

Comparing FPGAs to GPUs

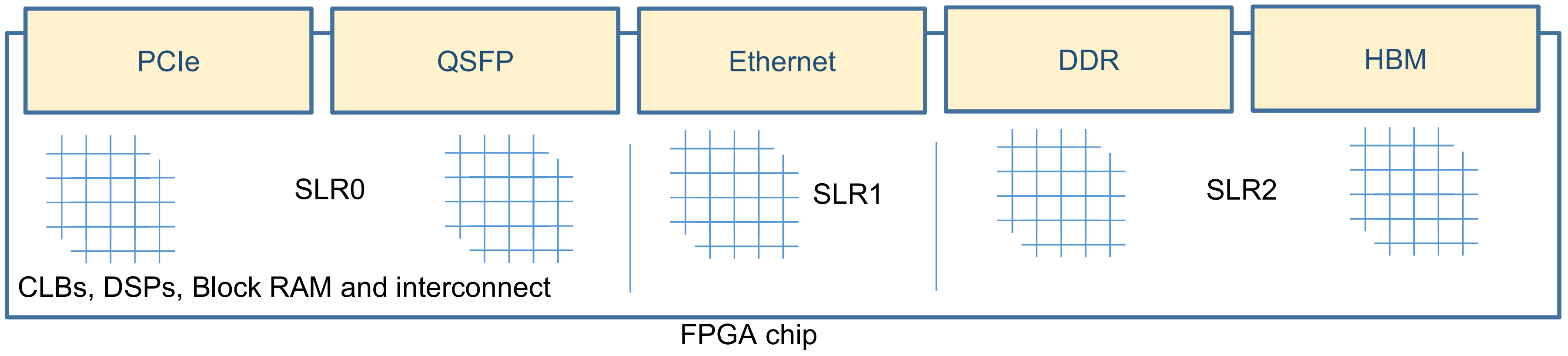
- “Basically”, replace the GPU compute device with the FPGA device





Inside the FPGA chip

- Interfaces
- Super Logic Regions (SLRs)
  - Only on big devices (Alveo)
- Components: Configurable Logic Blocks (CLBs)



SLR components

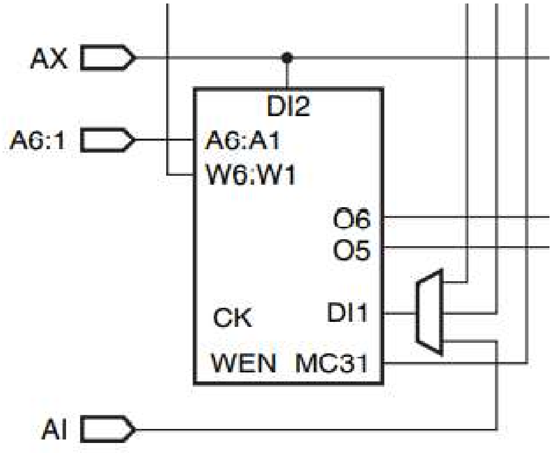
- CLB (Configurable Logic Block)
  - Carry chain (adder, counter, comparator...)
  - Multiplexer (8 / 16 to 1)
  - Shift register
  - Flip-flop
  - Look-up tables (6-input)
  - Distributed RAM
    - Recommended for 1 to 128 bits data
- Block RAM
- DSP (Digital Signal Processing)

Sample components

- Look-up table (8 on a CLB)
  - 6 inputs (A6:1)
  - 2 outputs (O6:5)
  - 128 bits total

Available resources
ZU+: 274K

U200: 388K + 205K + 385K
U250: 420K + 205K + 407K + 424K



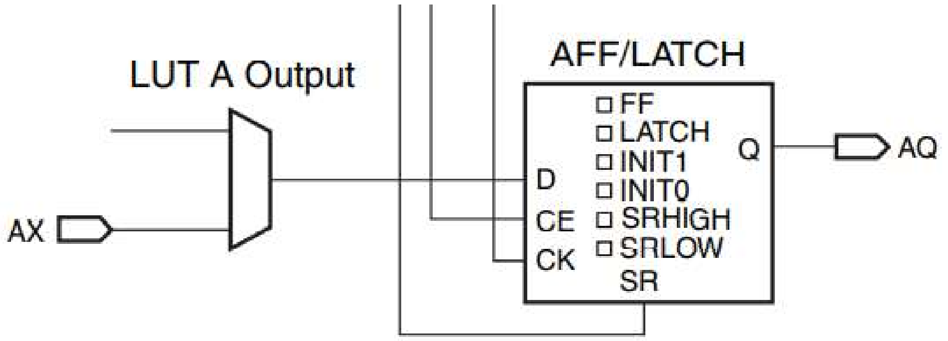
source: docs.xilinx.com

Sample components

- D flip-flop / latch (16 on a CLB)
  - 1 bit memory
  - Can take output from LUT A

Available resources
ZU+: 548K

U200: 776K + 410K + 770K
U250: 840K + 411K + 815K + 849K



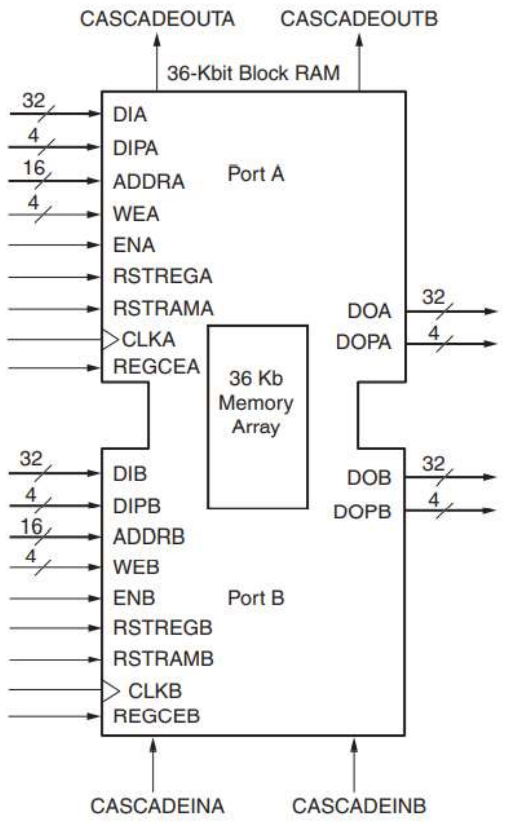
source: docs.xilinx.com

Sample components

- Block RAM
  - Dual-port 36Kb
  - Programmable FIFO
  - Configurable width
    - 32Kb x 1 ... 4Kb x 8 ... 1Kb x 36
- Advise
  - Use as many blocks as possible to achieve data parallelism

Available resources
ZU+: 32Mbit

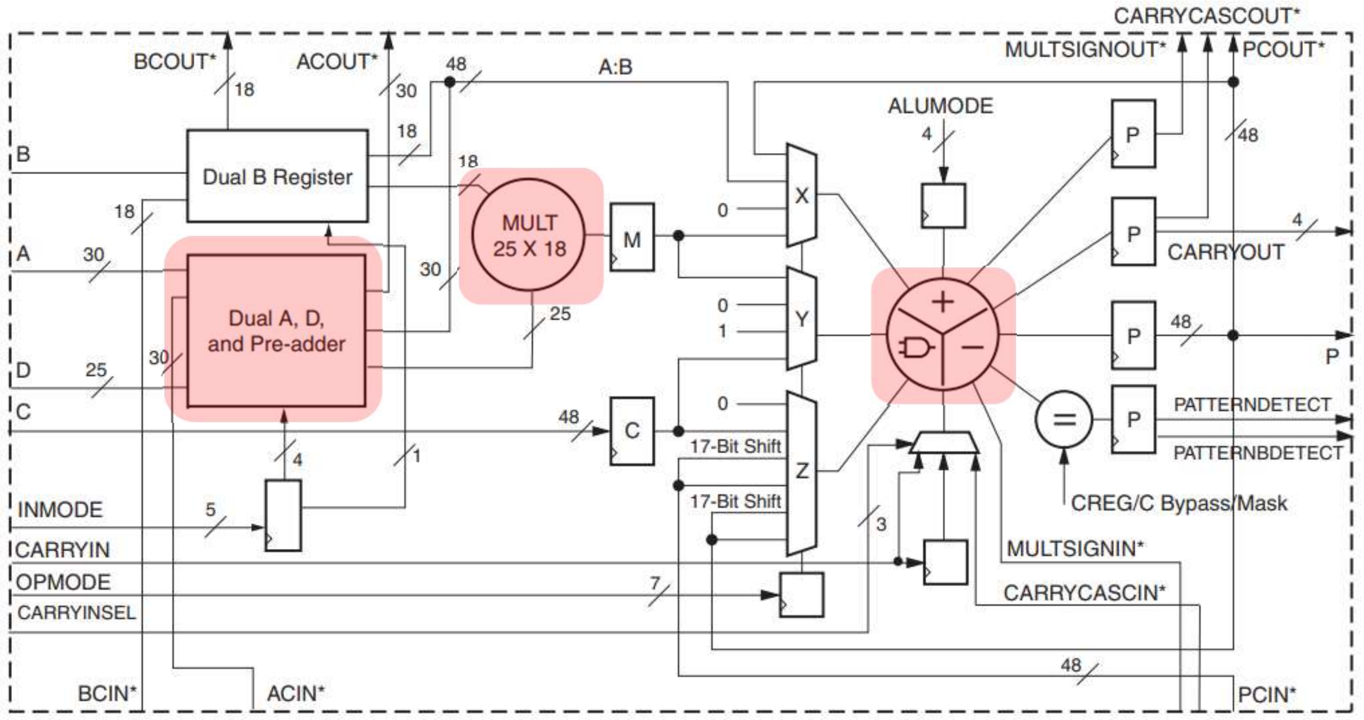
U200: 25Mb + 13Mb + 25Mb
U250: 18Mb * 4



source: docs.xilinx.com

Sample components

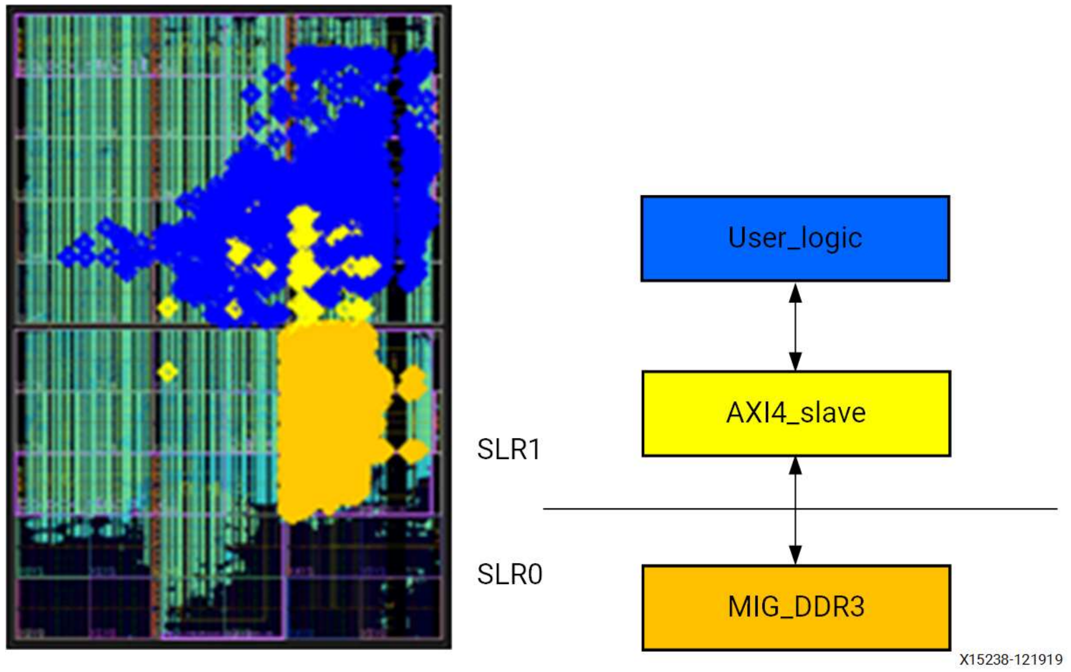
- DSPs (48-bit)
  - Multiplier
  - Accumulator
  - Pre-adder
    - C + B*(D+A) + Carryin
  - SIMD unit (+, - only)
    - 2x24bit, 3x12bit
  - Logic unit
    - And, or, xor...
  - Pattern detector
    - Detect output pattern, or
    - C match with A*B...



source: docs.xilinx.com

Sample design

- Memory accesses are driven by the MIG (Memory Interface Generator)
- Interconnects are implemented with the AXI ports
- User logic contains the application



source: docs.xilinx.com

Generating FPGA configurations

- High-level C/C++ programming
  - No Fortran support from vendors (as far as we know)
  - Compiled to Verilog/VHDL
- Design source files
  - Behavioral simulation
- Design synthesis, HDL to gates, generating FPGA netlists
  - Functional verification
- Design implementation, place & route
  - Static timing analysis
- Bitstream generation
  - Actual configuration for the FPGA

Vendor tools

- AMD – Xilinx
  - Vivado HLS / **Vitis HLS** (from 2020)
  - Vivado / Vitis
    - GUI and batch tools
  - Compile C/C++ code and OpenCL kernels

- Intel – Altera
  - Quartus HLS compiler
  - Quartus Prime Design Software
    - GUI and batch tools
  - Compile C/C++ code and OpenCL kernels
  - Also supports the **oneAPI** interface
    - Originated from Codeplay/Khronos SYCL
    - Kernels on C++ Lambda functions

No FORTRAN support to date

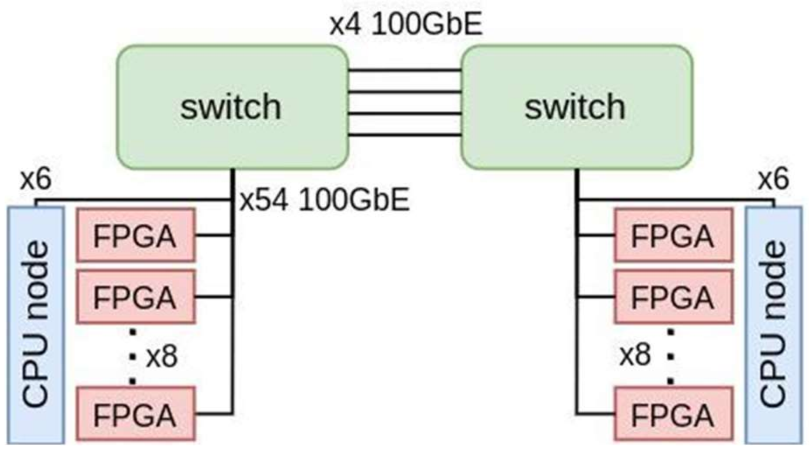
- Impacting the type and number of HPC applications that can be easily adapted

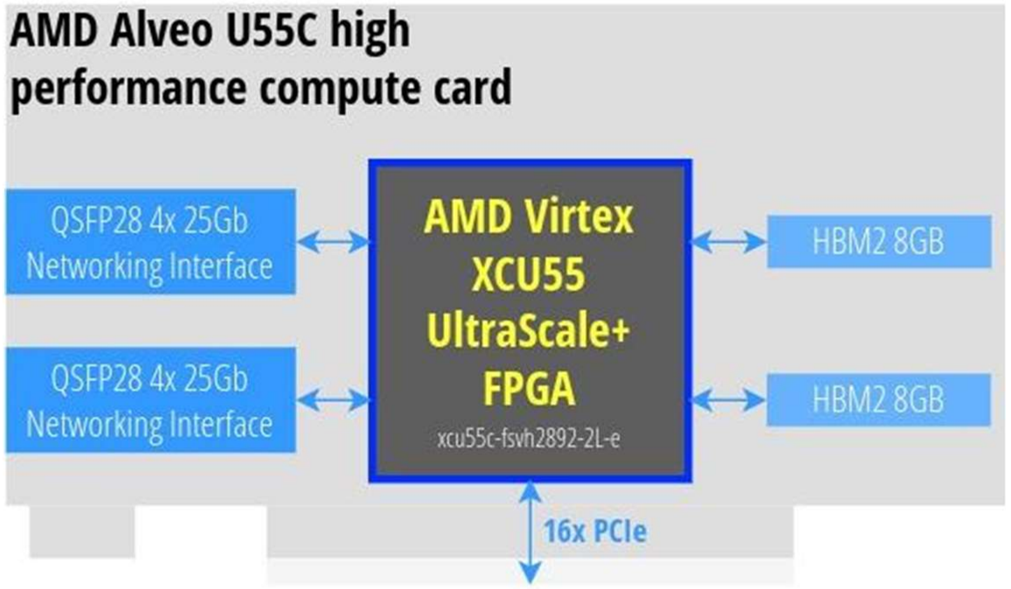
Barcelona Supercomputing Center
Centro Nacional de Supercomputación

Use case: The MareNostrum Experimental Exascale Platform (MEEP)

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

MEEP cluster architecture




AMD Alveo U55C high performance compute card
QSFP28 4x 25Gb Networking Interface
QSFP28 4x 25Gb Networking Interface
AMD Virtex XCU55 UltraScale+ FPGA
xcu55c-fsvh2892-2L-e
HBM2 8GB
HBM2 8GB
16x PCIe

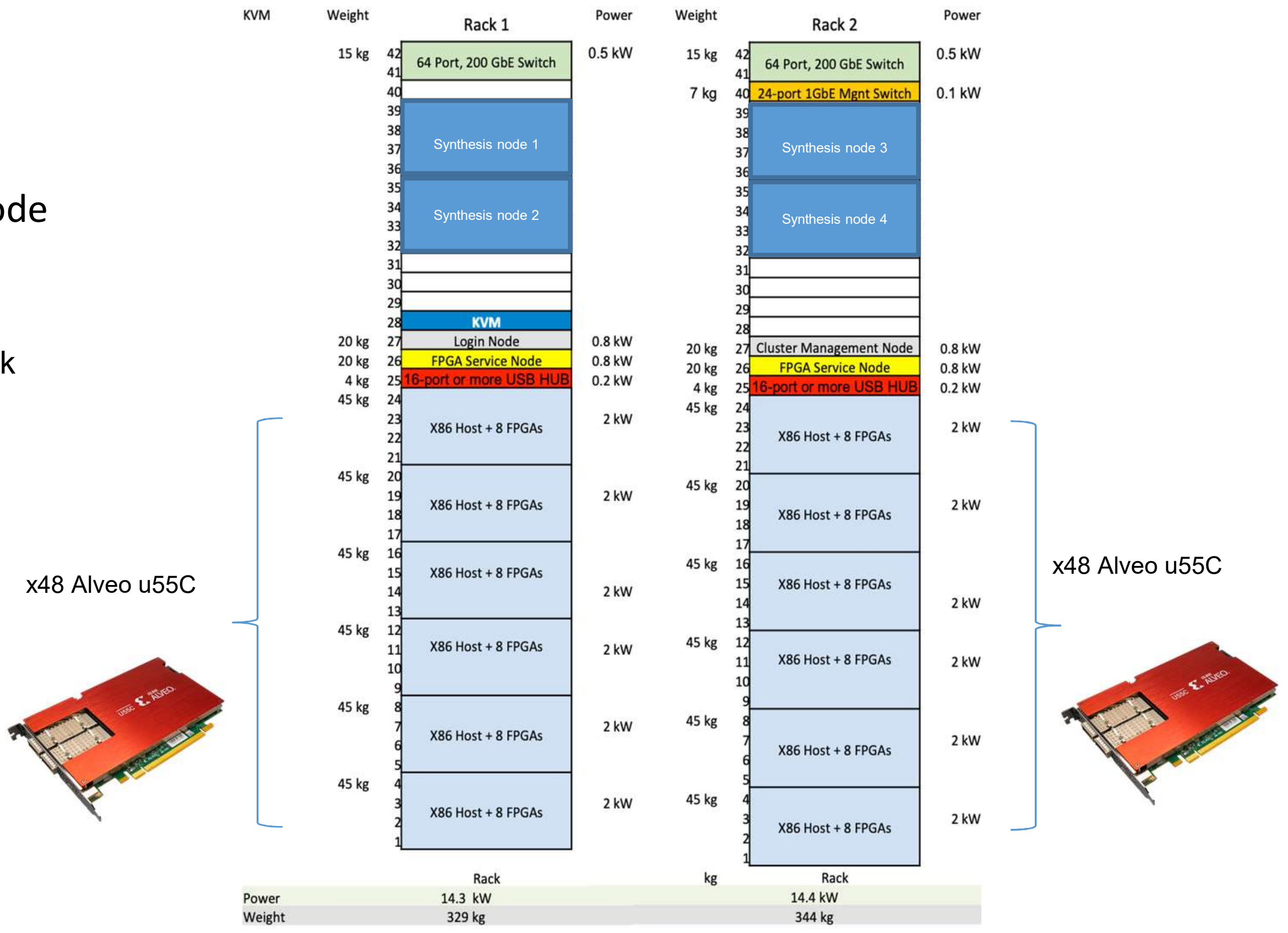
- 96 Alveo U55C FPGAs
- 12 Intel Xeon Gold 6330 CPUs with 256GB RAM
- 2 Nvidia SN4600C 100GbE switches

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

Cluster architecture

- 4 “synthesis” nodes
- 12 “compute” nodes
  - 8 u55c PCIe FPGA boards per node
- 2 switches
  - 96 FPGAs
  - 12 nodes

Private, “MPI” network

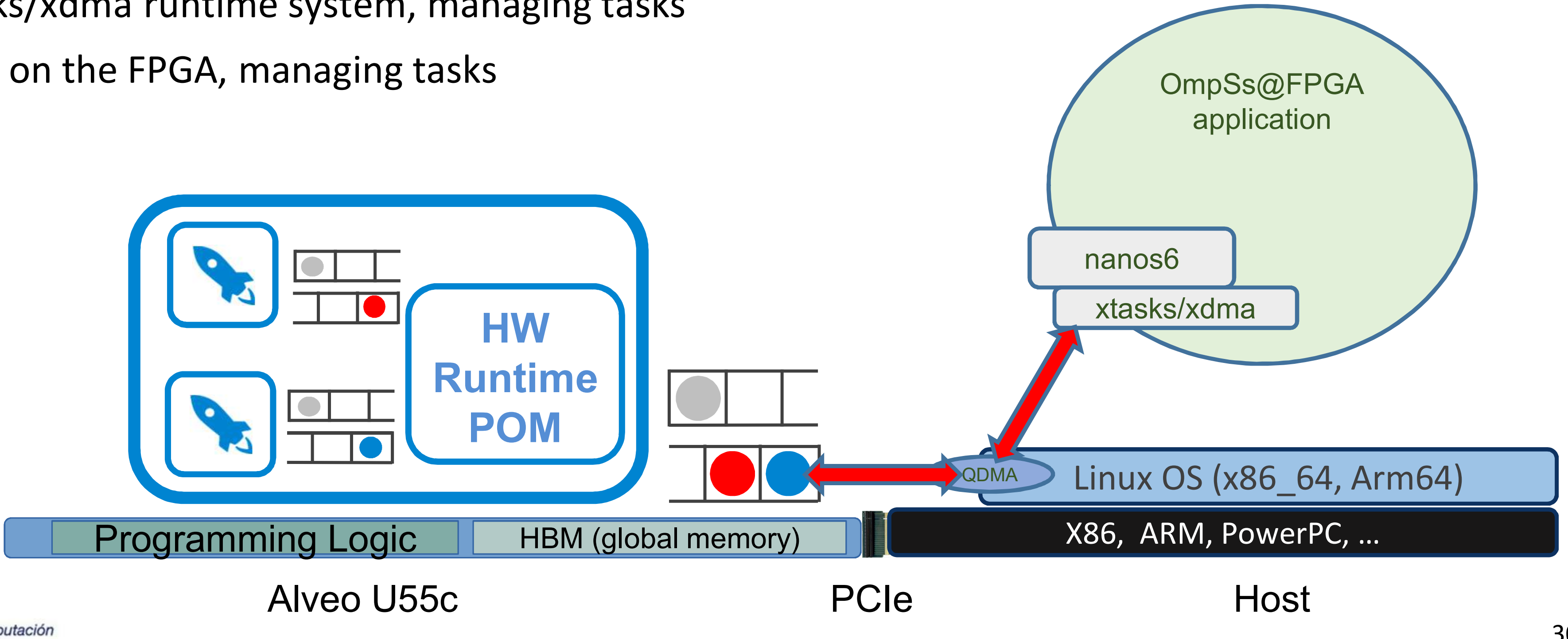


OmpSs@FPGA execution environment (single FPGA)

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

OmpSs@FPGA execution environment

- x86_64 / Arm64 architectures
- Linux (Host), QDMA driver
- Nanos/xtasks/xdma runtime system, managing tasks
- Hw runtime on the FPGA, managing tasks



Barcelona Supercomputing Center
Centro Nacional de Supercomputación

Distributed FPGA models

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

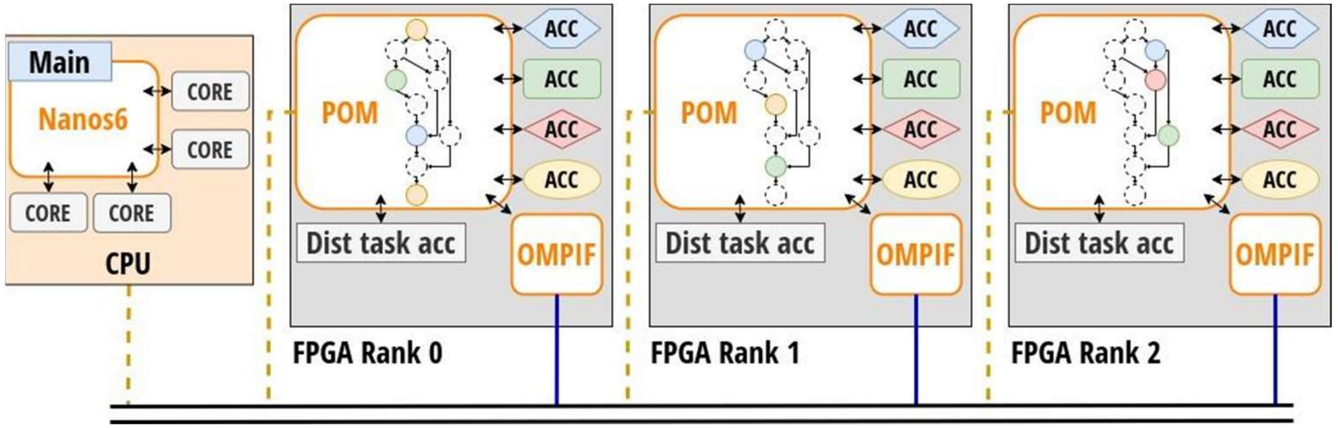
Distributed FPGA models

OMPIF

- MPI-like API called from the FPGA code → **OmpSs MPI for FPGAs (OMPIF)**
  - FPGA-to-FPGA communication
  - OmpSs hardware runtime handles calls to the API

```
void OMPIF_Send(const void* buf, int size, int dest, int tag) → MPI_Send
void OMPIF_Recv(void* buf, int size, int source, int tag)     → MPI_Recv
void OMPIF_Allgather(void* data, unsigned int size)           → MPI_Allgather
void OMPIF_Bcast(void* data, unsigned int size, int root)     → MPI_Bcast
int  OMPIF_Comm_rank()
                → MPI_Comm_rank
int  OMPIF_Comm_size()                                        → MPI_Comm_size
```

- Distributed task: Special type of task used to start the application on all FPGAs.



Barcelona Supercomputing Center
Centro Nacional de Supercomputación

Distributed FPGA models

OmpSs@FPGA+OMPIF example

```
#pragma oss task device(fpga) in(pbi[0], pbj[0]) inout(fb[0])
void calculate_forces(particles_t *pbi, particles_t *pbj, forces_t *fb);
#pragma oss task device(fpga) inout(fb[0], pb[0])
void update_particles(particles_t *pb, forces_t *fb);
#pragma oss task device(fpga) inout(p[0], f[0]) distributed
void nbody(particles_t *p, forces_t *f, int nb, int steps)
{
   int r = OMPIF_Comm_rank();
   int nr = OMPIF_Comm_size();
   for (int k = 0; k < nt; k++) {
      for (int i = 0; i < nb; ++i)
         for (int j = nb/nr*r; j < nb/nr*r+nb/nr; ++j)
             calculate_forces(p+i, p+j, f+j);
      #pragma oss taskwait
      OMPIF_Allgather(f, fbs*sizeof(forces_t));
      for (int i = 0; i < nb; ++i)
         update_particles(p+i, f+i);
   }
}
```

→ Distributed task declaration

→ Similar to MPI Allgather collective, implicit barrier

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

Distributed FPGA models

OmpSs@FPGA IMP example

```
#pragma oss task device(fpga) in(pbi[0], pbj[0]) inout(fb[0])
void calculate_forces(particles_t *pbi, particles_t *pbj, forces_t *fb);
#pragma oss task device(fpga) inout(fb[0], pb[0])
void update_particles(particles_t *pb, forces_t *fb);
#pragma oss task device(fpga) inout(p[0], f[0]) owner("all") \
  data_dist("all", p, nb*PBS)
  data_dist(BS, f, nb*FBS)
void nbody(particles_t *p, forces_t *f, int nb, int steps) {
   for (int k = 0; k < nt; k++) {
      for (int i = 0; i < nb; ++i)
         for (int j = 0; j < nb; ++j)
            calculate_forces(p+i, p+j, f+j);
      for (int i = 0; i < nb; ++i)
         update_particles(p+i, f+i);
   }
   #pragma oss taskwait
}
```

All ranks run this task

Particles are sent to all nodes

Each rank owns a block of forces

No OMPIF calls

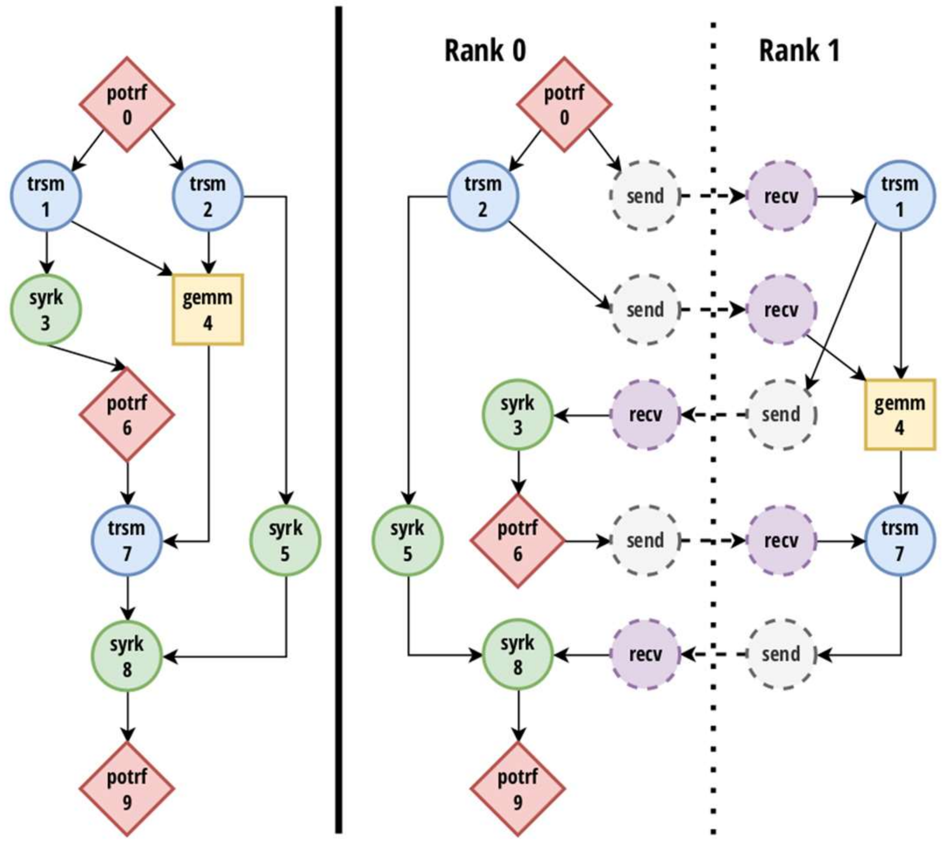
Taskwait before leaving "nbody"
(parallelism between iterations)

Distributed FPGA models

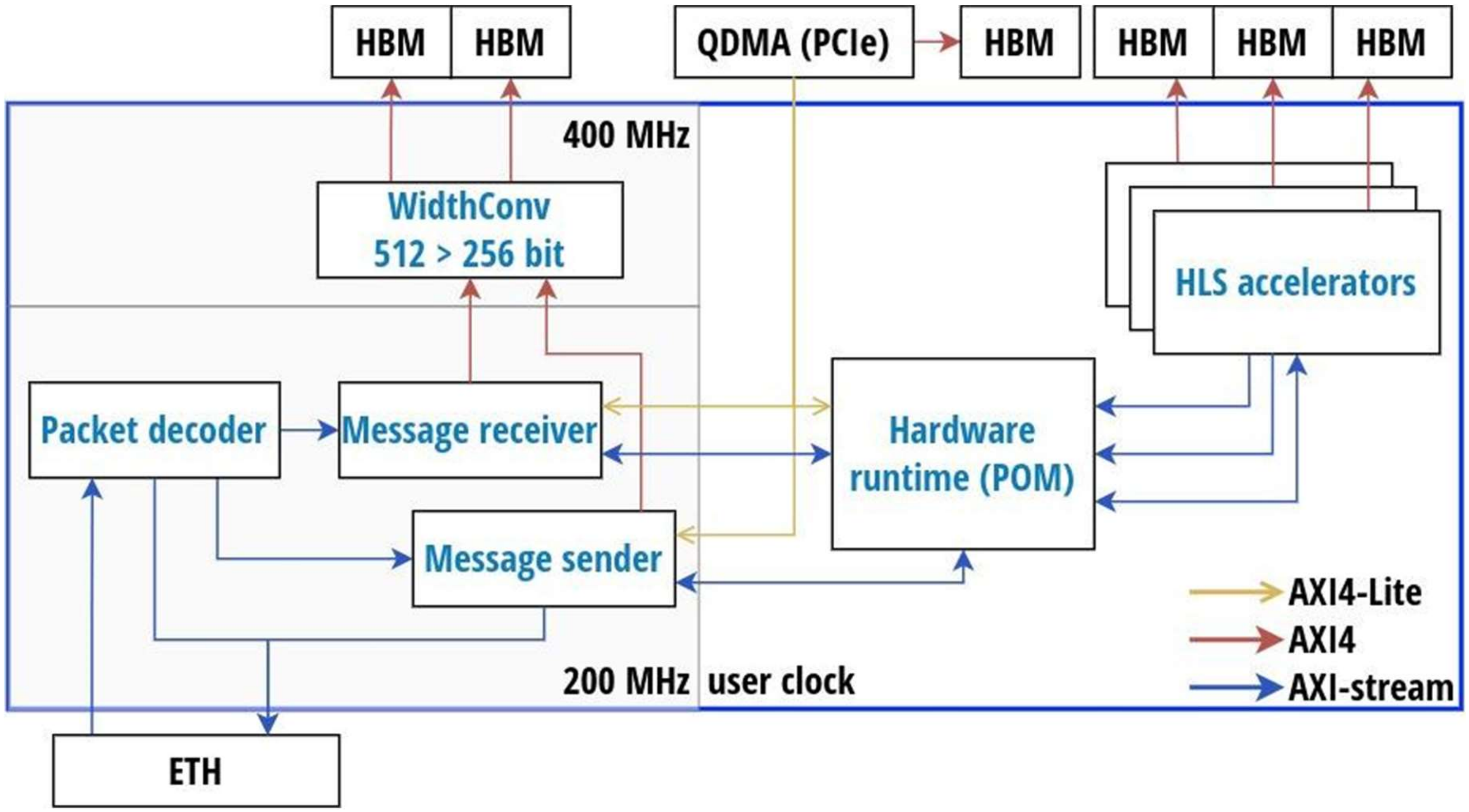
IMP model

- Task graph is distributed between FPGAs
- Data movements are automatically generated based on data dependencies and ownership

```
#pragma oss task device(fpga) in(pbi[0], pbj[0]) inout(fb[0])
void calculate_forces(particles_t *pbi, particles_t *pbj, forces_t *fb);
#pragma oss task device(fpga) inout(fb[0], pb[0])
void update_particles(particles_t *pb, forces_t *fb);
#pragma oss task device(fpga) inout(p[0], f[0]) owner("all") \
  data_dist("all", p, nb*PBS)
  data_dist(BS, f, nb*FBS)
void nbody(particles_t *p, forces_t *f, int nb, int steps) {
   for (int k = 0; k < nt; k++) {
      for (int i = 0; i < nb; ++i)
         for (int j = 0; j < nb; ++j)
             calculate_forces(p+i, p+j, f+j);
      for (int i = 0; i < nb; ++i)
         update_particles(p+i, f+i);
   }
   #pragma oss taskwait
}
```

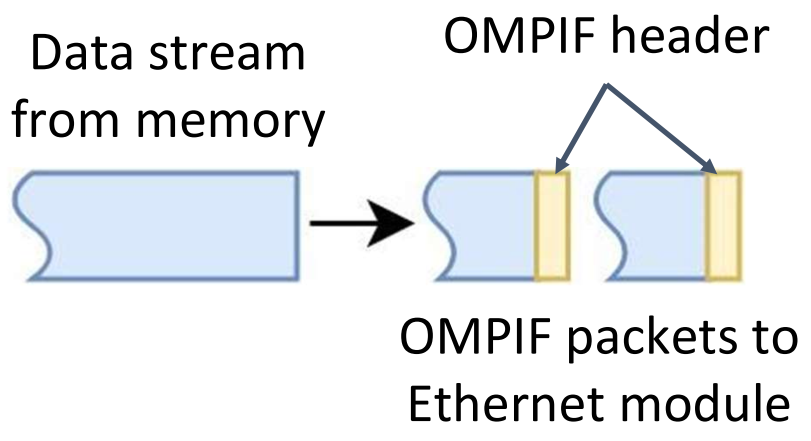


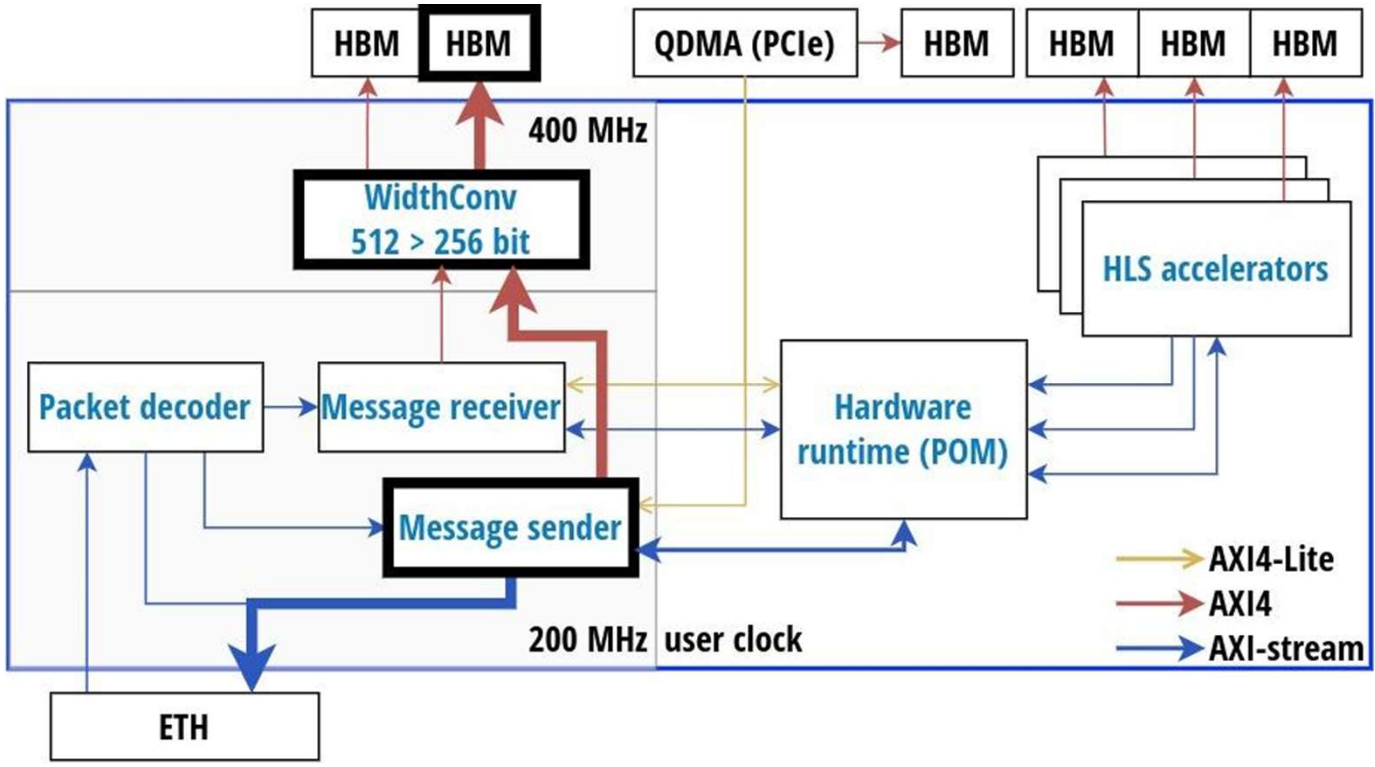
Alveo U55C OmpSs@FPGA design



Barcelona Supercomputing Center
Centro Nacional de Supercomputación

Message send process

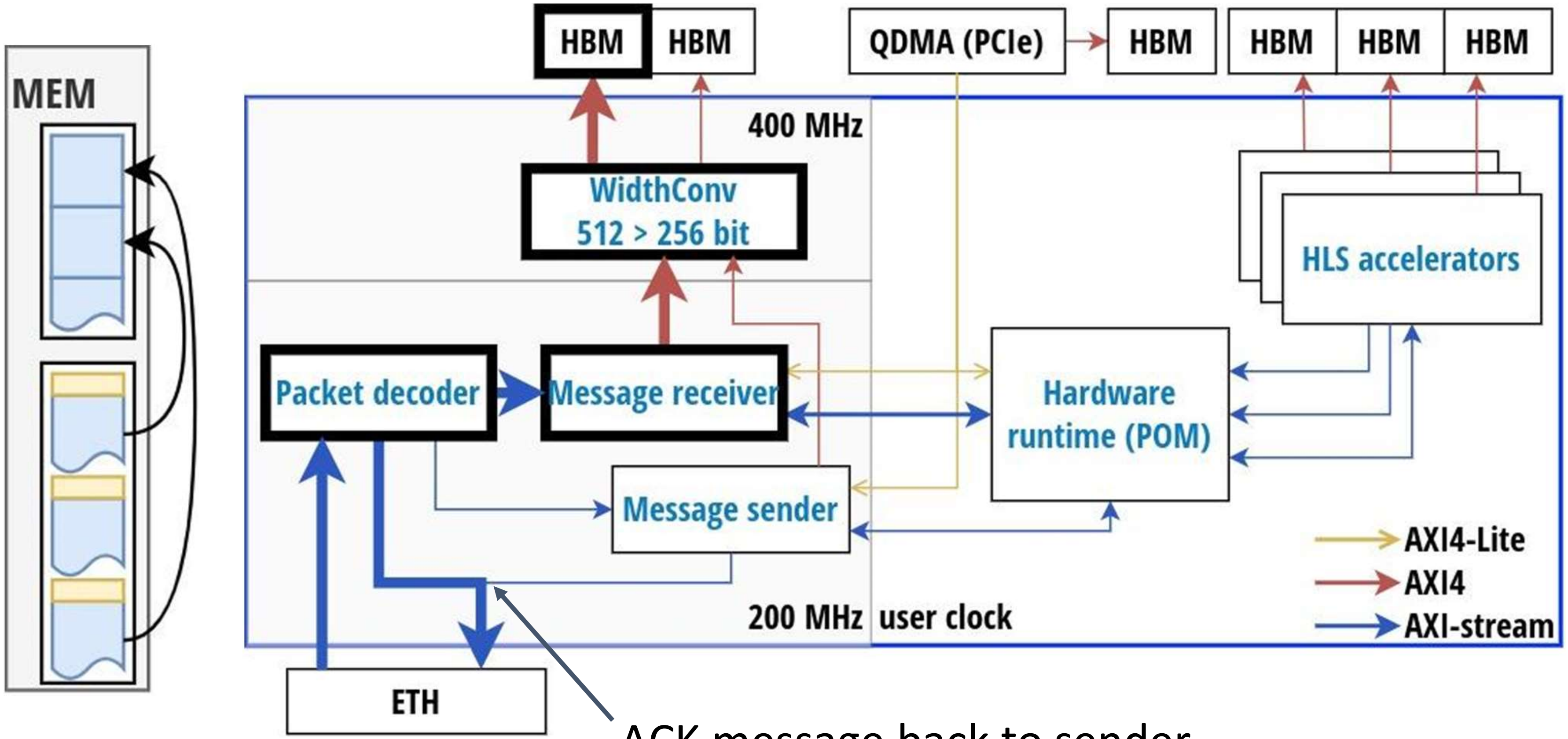




Barcelona Supercomputing Center
Centro Nacional de Supercomputación

Message receive process

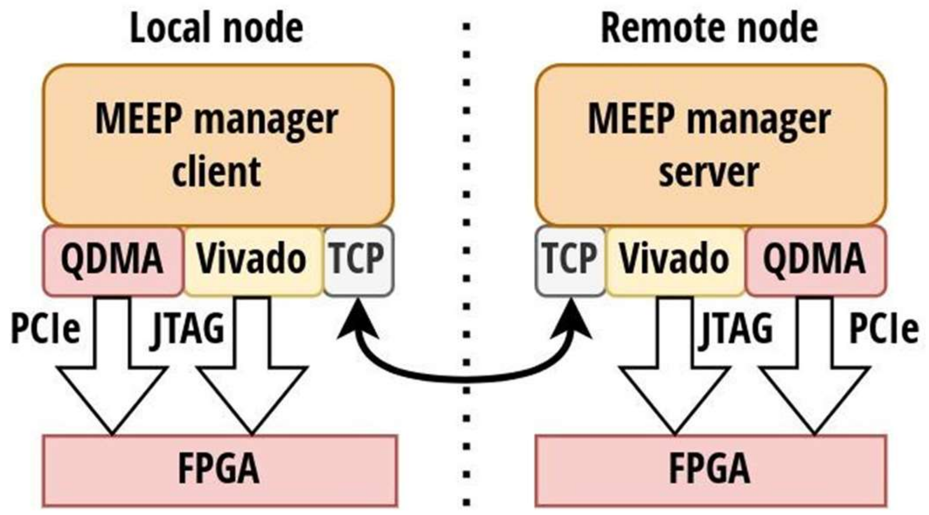
Incoming OMPIF packets are stored in an intermediate buffer because final address is unknown



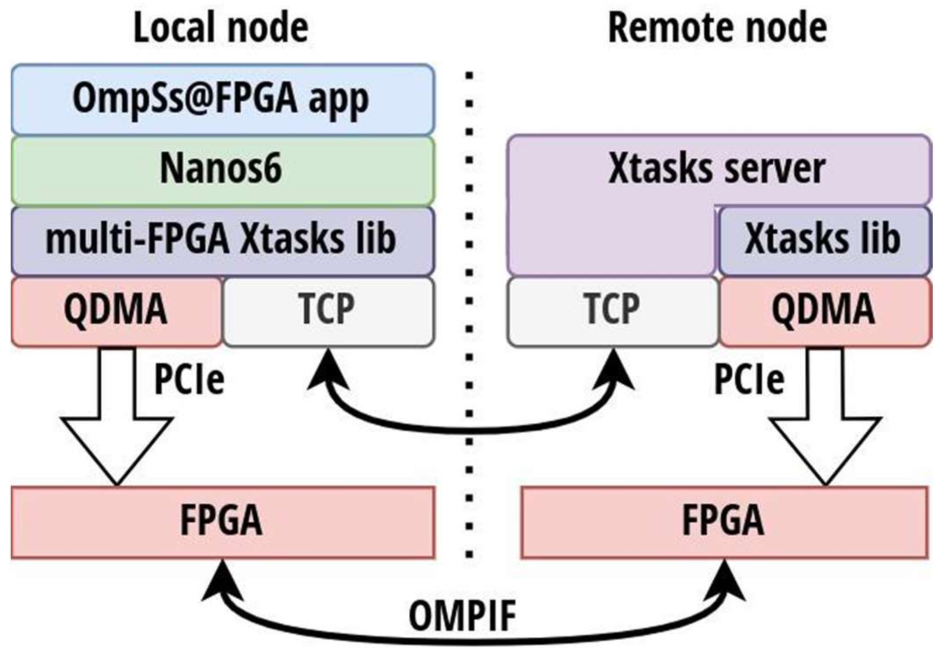
ACK message back to sender

Software stack

Deployment & management



Application execution



Barcelona Supercomputing Center
Centro Nacional de Supercomputación

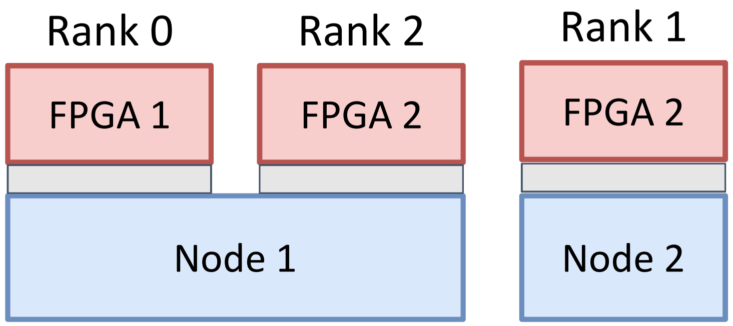
ompifrun

Cluster JSON description

```
[
        { “fpga”: 1, “node”: 1, “bitstream”: “path/to/bitstream” },
        { “fpga”: 2, “node”: 2, “bitstream”: “path/to/bitstream” },
        { “fpga”: 2, “node”: 1, “bitstream”: “path/to/bitstream” }
]
```

- **ompifrun** is a tool that automatically prepares all FPGAs in a cluster allocated by the user:

1. Load the bitstream
2. Wait for the Ethernet IP to align with the switch
3. Enable TX/RX controllers, ARP/ICMP servers, and check that the Ethernet IP is aligned and synchronized correctly
4. Setup IP/MAC addresses and OMPIF rank/size registers
5. Initialize QDMA queues, needed for memory transfers through PCIe

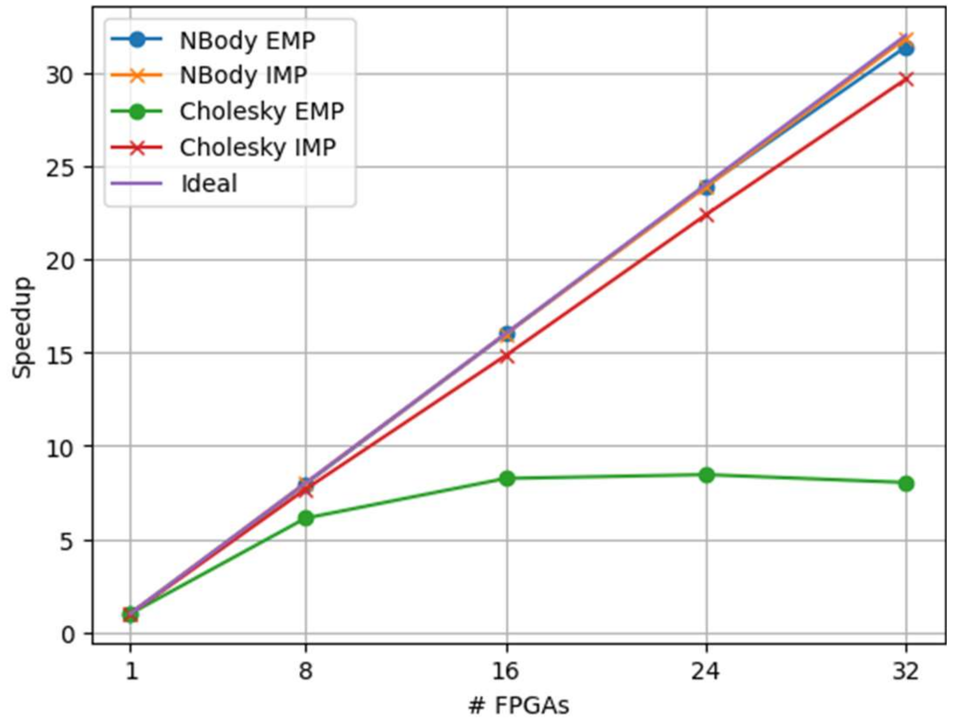


Results

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

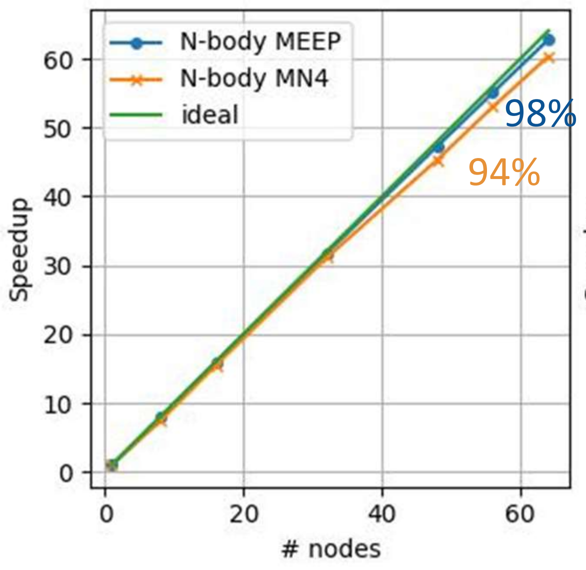
Evaluation: Explicit vs. Implicit Message Passing

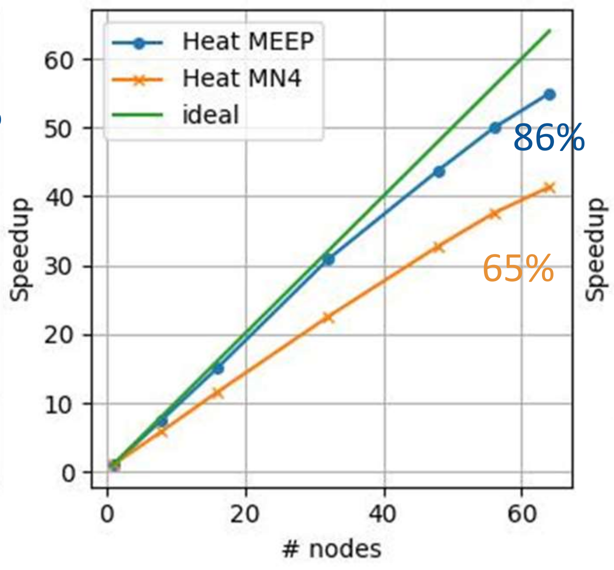
- IMP provide overall better scalability than explicit (OMPIF) message passing
- Collectives (implying global synchronization) are removed in IMP
    - Data transfers are automatically generated
- Easier to overlap computation and communication

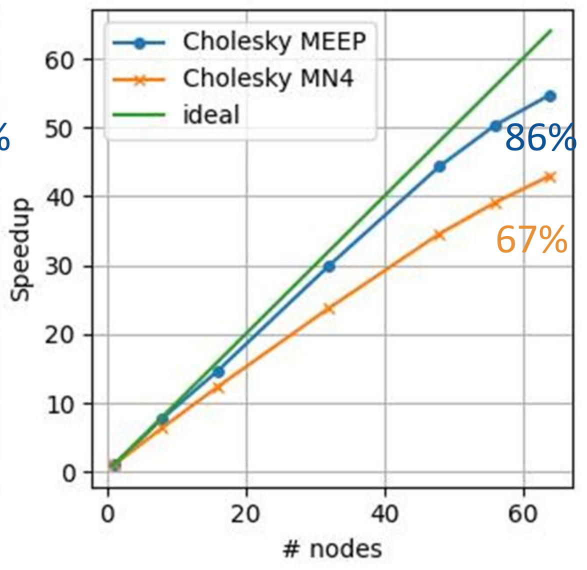


Evaluation: Performance Scalability (IMP)

- We used CPU-only Mare Nostrum 4 (MN4) to compare with the FPGA implementation.





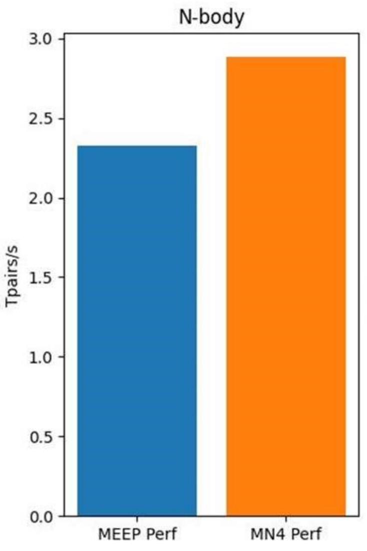


Barcelona Supercomputing Center
Centro Nacional de Supercomputación

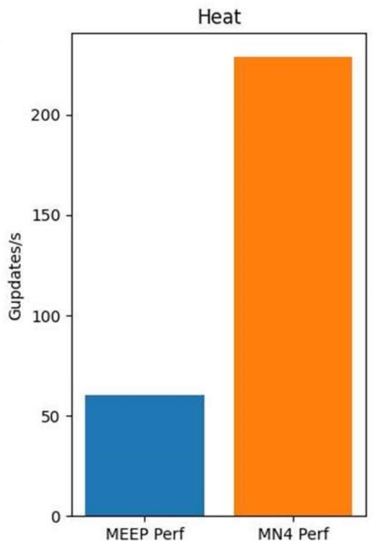
Evaluation: Performance and power efficiency

64 nodes

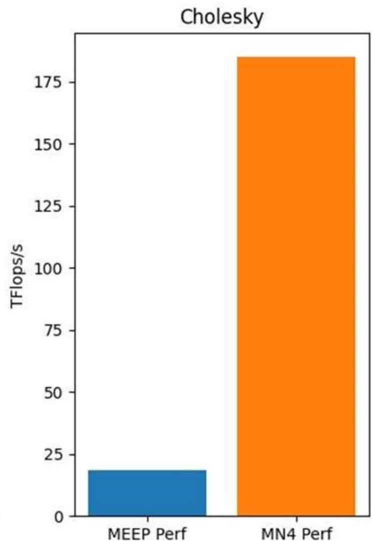
-1.2x



-3.7x

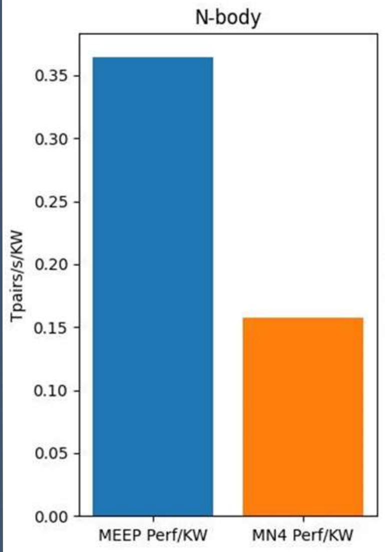


-10x

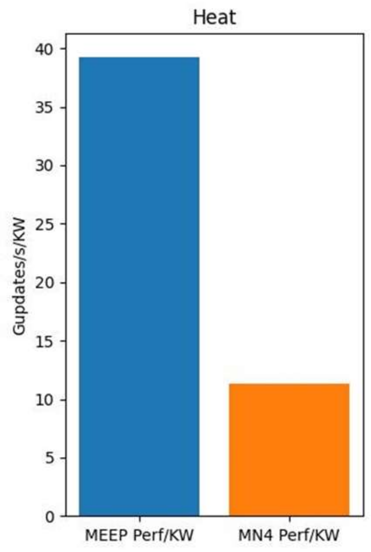


Performance (higher better)

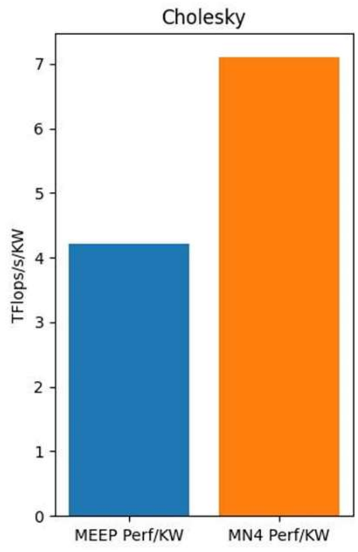
2.3x



3.5x



-1.6x

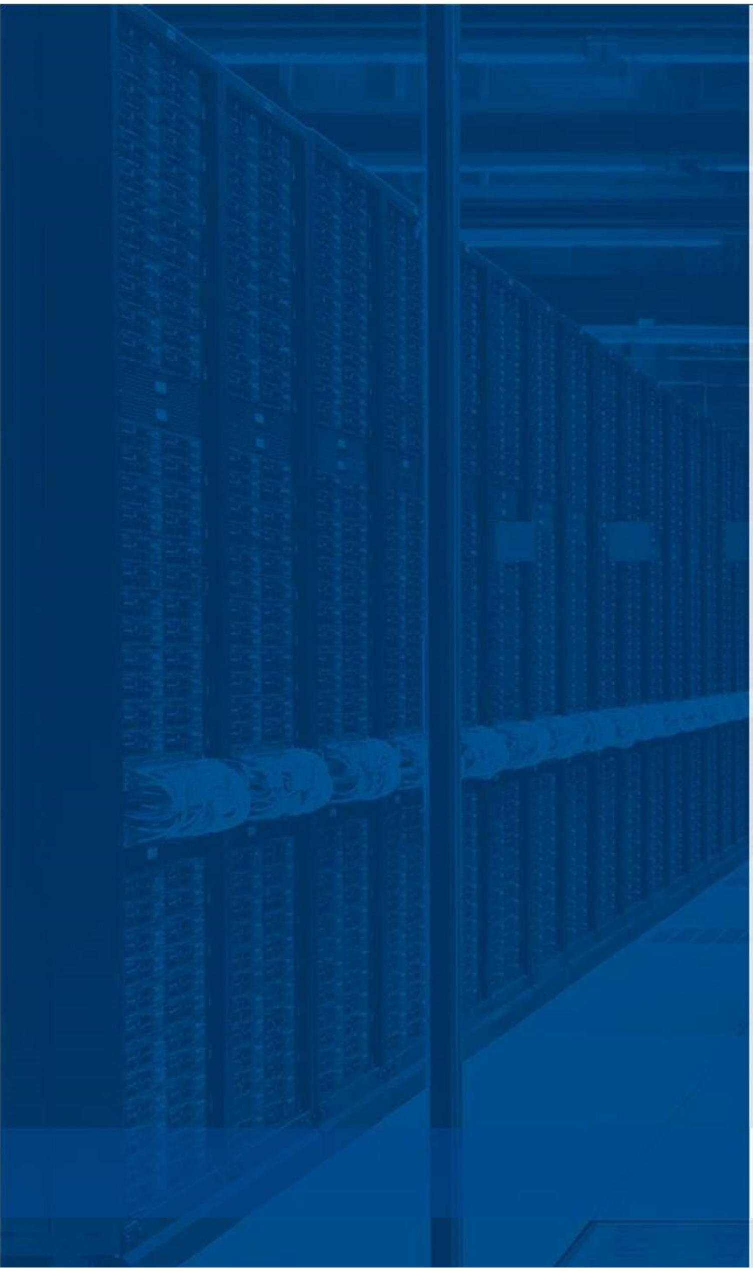


Performance/KW (higher better)

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

Conclusions

- MPI-like programming model with tasking support for FPGA clusters - IMP
- We propose a model that allows Implicit communications based on directives
  - Built on top of the OMPIF infrastructure
- Three benchmarks evaluated, which show very good scalability up to 64 FPGAs
- Compared to a CPU-only cluster (MN4), we have better scalability and performance per power for N-body and Heat



Barcelona Supercomputing Center
Centro Nacional de Supercomputación

Accelerating Applications on Networks of FPGAs

Contact: xavier.martorell@bsc.es
ompss-fpga-support@bsc.es

https://pm.bsc.es/ompss-at-fpga

BZL BARCELONA ZETTASCALE LAB

This work was supported by the Barcelona Zettascale Laboratory, with the support of the Ministry for Digital Transformation and the Civil Service, within the framework of the Recovery, Transformation and Resilience Plan - Funded by the European Union - NextGenerationEU