Esempi di calcolo distribuito con Linux

Dalla distribuzione di applicazioni su più schede embedded alla compilazione veloce su Linux

LinuxDay Pisa, 28/10/2017

Stefano Garzarella, Bruno Morelli
[s.garzarella, b.morelli]@evidence.eu.com

Agenda

- Cluster with embedded systems
  - **Cluster with AXIOM board** (Xilinx UltraScale +)
    - AXIOM board
    - AXIOM Software Stack
    - AXIOM NIC
    - Performance
  - **Cluster with UDOO x86**
    - UDOO boards
    - Docker
      - Docker Swarm
    - Distcc
    - Linux compilation with distcc on the UDOO x86 cluster

The Company

- Founded in 2002 as spin-off company of the Real-Time Systems Lab at Scuola Superiore S.Anna
    - ~20 qualified people with an average age of 34 years
    - 10+ years of experience in academic and industrial projects
    - One third of the company has a PhD degree

Impresa Spin-Off della Scuola Sant'Anna

Our Mission:

design and development software for small electronic devices

Product and services

RTOS , Firmware, Embedded Linux

- AUTOSAR, OSEK/VDX, device drivers
- Embedded Linux: 12 Yrs experience BSPs, GCC, U-Boot, Kernel drivers
- Initial developers of the SCHED_DEADLINE patch
- Hypervisors, Android, Ubuntu Core, QEMU and emulators

Application Development

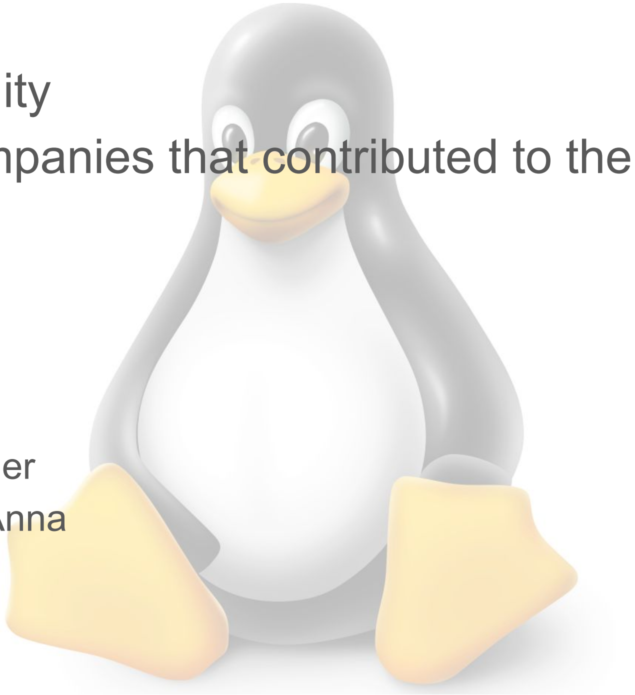
Model-based design

- Matlab/Simulink/Stateflow
- National Instruments LabView
- E4Coder toolset for code generation
- UML/SYSML/Ecore/ Eclipse/Acceleo

Evidence and Linux

- Deep knowledge of Linux kernel internals
- Constant collaboration with the kernel community
- Since 2008 Evidence is in the official list of companies that contributed to the Linux kernel

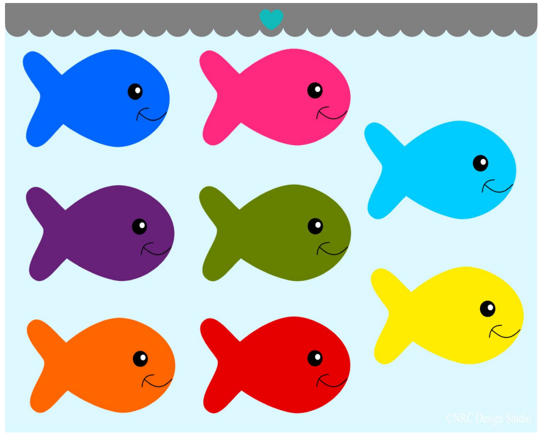
- Original developer of SCHED_DEADLINE
  - Real-time CPU scheduler merged in Linux 3.14
  - Then improved in release 4.13 with the GRUB scheduler
  - Made in collaboration with ReTiS Lab of Scuola Sant'Anna
  - It allows real-time isolation between running tasks
  - http://en.wikipedia.org/wiki/SCHED_DEADLINE

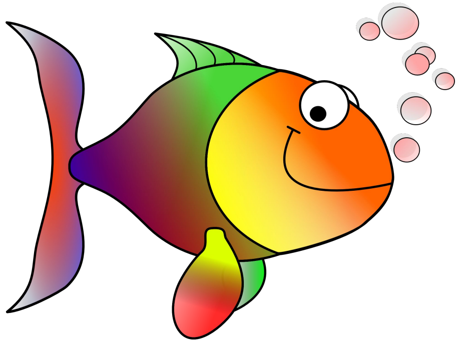


Cluster with embedded systems

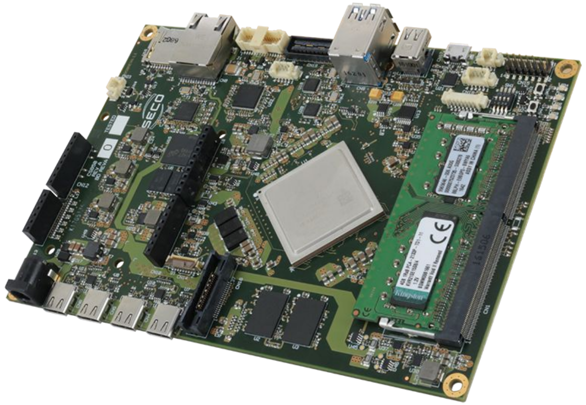
Embedded systems are becoming more and more popular and **interconnected.** Low cost high performance chips are now available.

Can we put together a set of small embedded boards to create a bigger supercomputer?





Cluster with AXIOM board UltraScale+ (Cortex A53 + FPGA)



AXIOM

AXIOM project

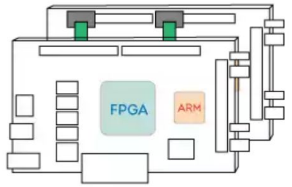
INDUSTRIAL PARTNERS

ACADEMIC PARTNERS

Agile, e**X**tensible, fast **I/O M**odule for the cyber-physical era

- Flexible, energy efficient and multi-board
- Easily Programmable
- Easy Interfacing with the Cyber-Physical Worlds

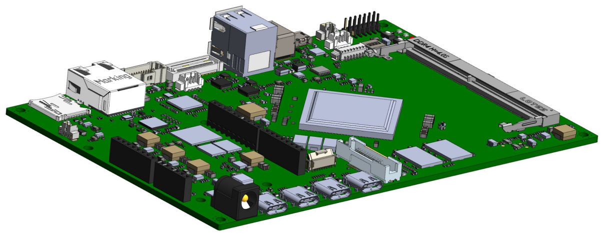
FLEXIBILITY
Flexible board based on low-power ARM and accelerated through FPGA



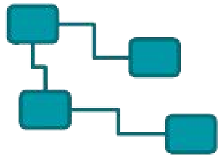
MODULARITY
Board-to-board fast interconnects for scalable and easy programmability

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 645496.

The AXIOM Project in one slide

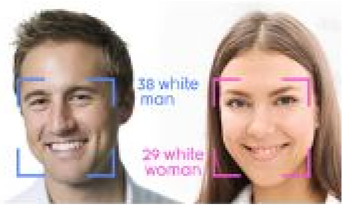


- We are designing a **small embedded board**
  - It bridges High Performance Computing (HPC) and Cyber Physical Systems (CPS)
- We connect a set of boards together using **high-speed transceivers** of Xilinx Zynq Ultrascale+
  - RDMA for fast transfers!
- We develop a **common programming paradigm** OmpSs@Cluster
  - OpenMP on the cluster on top of GASNet OmpSs@FPGA
  - Transparent FPGA acceleration
- We use it for **video and audio processing**
  - Smart surveillance, speech recognition



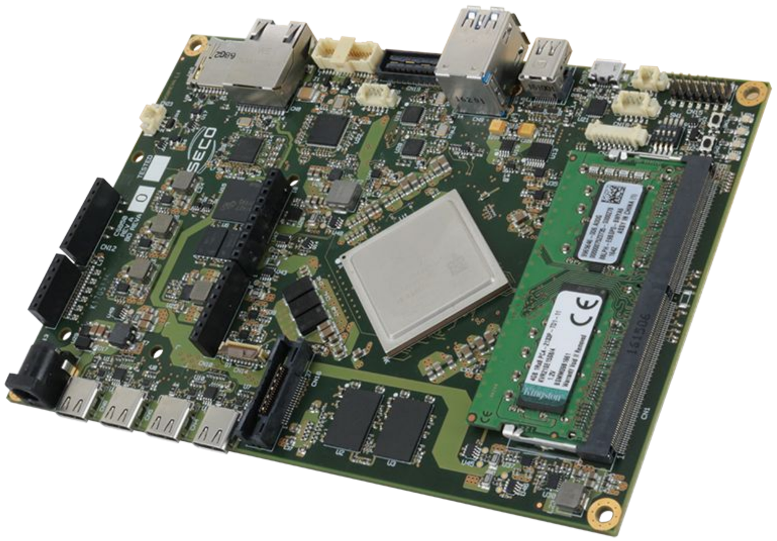
Cluster setup!

Contributing to the standard



AXIOM Board: characteristics

- Small form factor (160cm x 109cm)
- Xilinx Zynq Ultrascale+ ZU9EG
- socket SO-DIMM DDR4 for the PS RAM
- 1Gb DDR4 for the PL RAM
- eMMC: 8 to 32 GB
- Boot from QSPI, eMMC, uSD card, JTAG
- Standard connections:
  (USB, Gb Ethernet, Video output)
- Camera input
- Trace port for software tracing
- Power management measurement



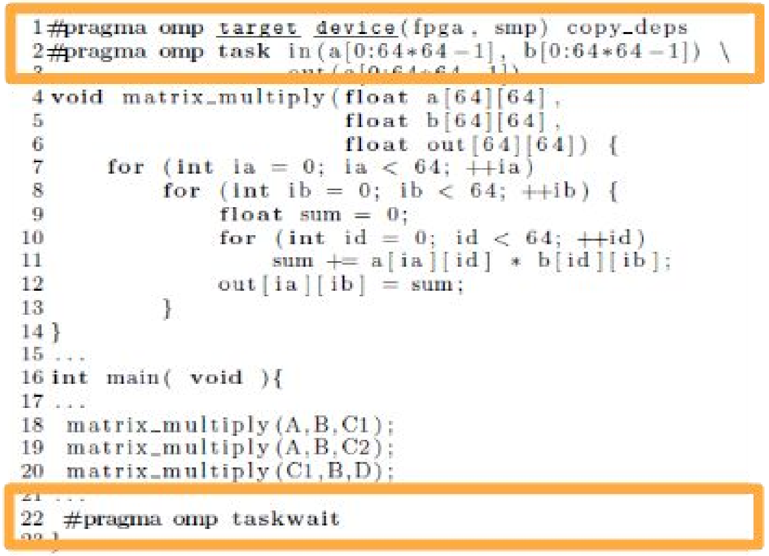
AXIOM Board: AXIOM Link

- USB Type C connector
- Used to get a high-speed connection between boards
- Standard connector with special care for signal integrity

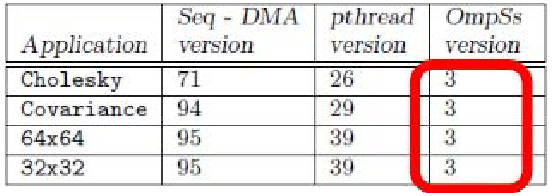
Easy programmability via OmpSs

Only 3 lines of code to:

- accelerate code on FPGAs
- distributed code across several AXIOM boards



```
#pragma omp target device(fpga, smp) copy_deps
#pragma omp task in(a[0:64*64-1], b[0:64*64-1]) \
                  out(c[0:64*64-1])
void matrix_multiply(float a[64][64],
                     float b[64][64],
                     float out[64][64]) {
     for (int ia = 0; ia < 64; ++ia)
          for (int ib = 0; ib < 64; ++ib) {
               float sum = 0;
               for (int id = 0; id < 64; ++id)
                    sum += a[ia][id] * b[id][ib];
               out[ia][ib] = sum;
          }
}
...
int main( void ){
...
  matrix_multiply(A,B,C1);
  matrix_multiply(A,B,C2);
  matrix_multiply(C1,B,D);
...
  #pragma omp taskwait
}
```

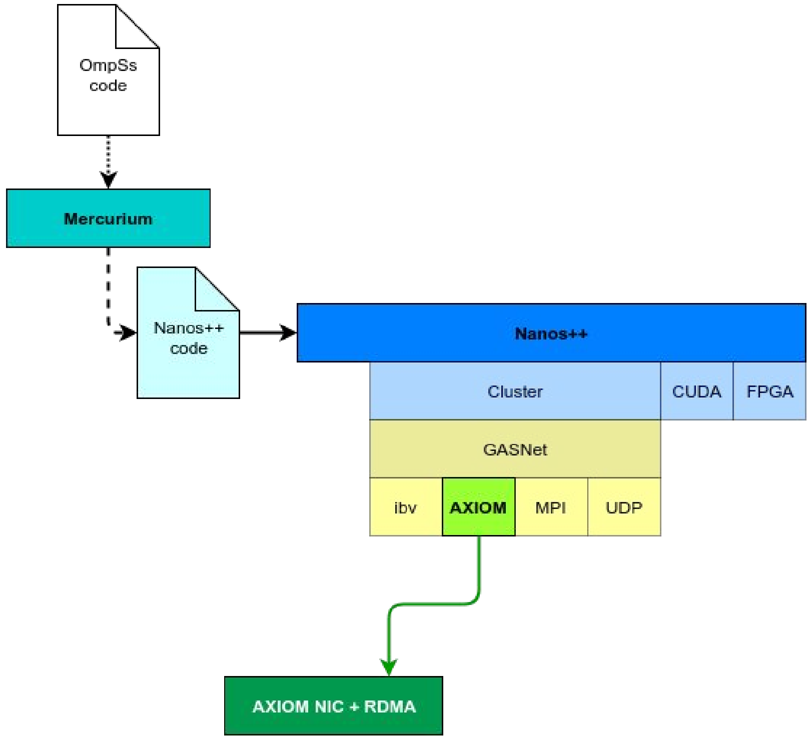


Application	*Seq - DMA version*	*pthread version*	*OmpSs version*
Cholesky	71	26	3
Covariance	94	29	3
64x64	95	39	3
32x32	95	39	3

AXIOM Application runtime

OmpSs@cluster integrated with a GASNet conduit based directly on the AXIOM NIC and RDMA mechanism

- Mercurium compiler
- Nanos++ runtime
- GASNet framework
- AXIOM NIC

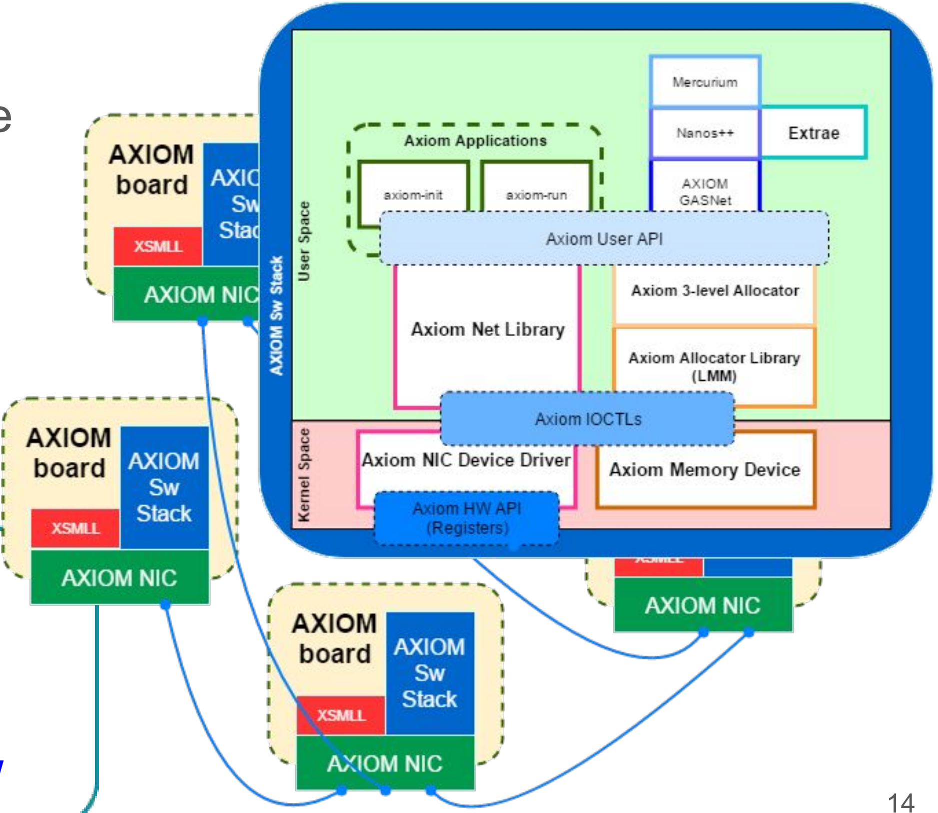


First complete AXIOM software stack now available!

- AXIOM board and QEMU Emulation
- AXIOM-Link software specs available
- Device drivers
- Memory allocator
- Utility apps
- GASNet Spawner
- OmpSs@Cluster

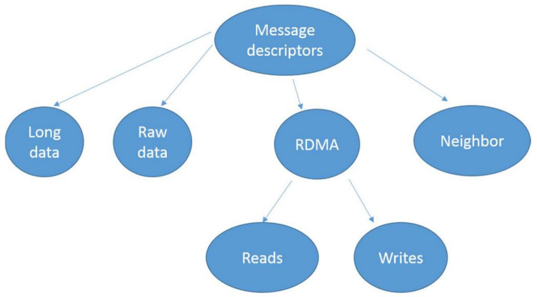
SW stack Available today!

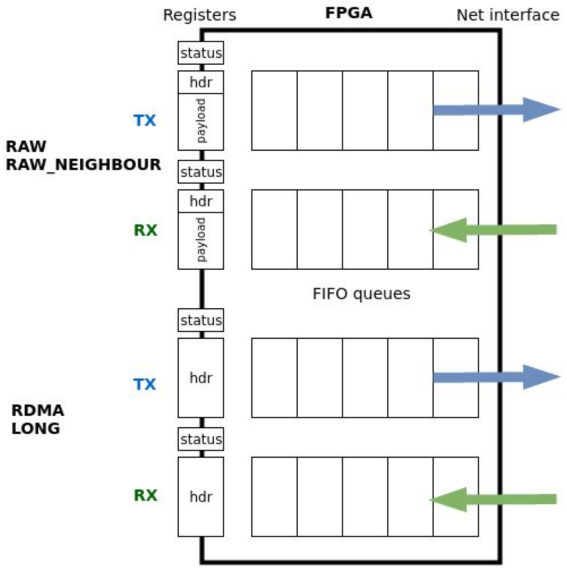
http://www.axiom-project.eu/



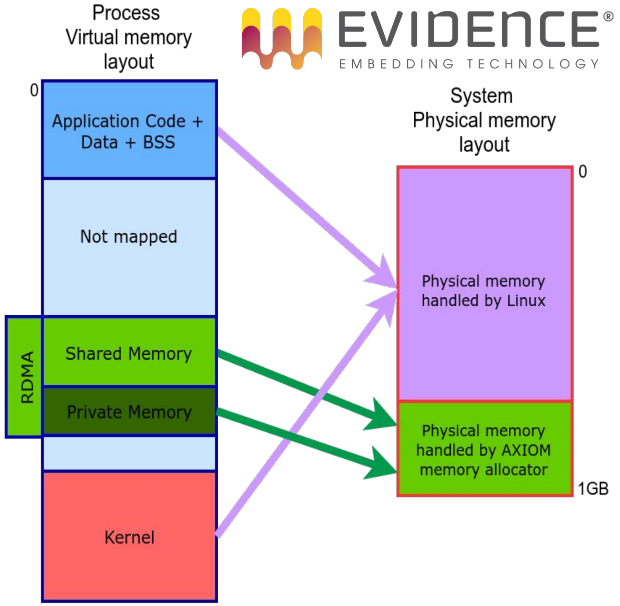
AXIOM Network Interface

- Multiple type of messages and queues
  - Small messages (payload embedded in the descriptor)
    - **RAW**
      - Very short message (up to 256 bytes)
    - **RAW NEIGHBOUR**
      - RAW message to neighbour node
  - Big messages (payload as a pointer in the descriptor)
    - **RDMA read/writes**
      - Remote DMA transfer between two nodes
    - **LONG**
      - Based on RDMA, but without specify a destination address
      - Pool of buffers provided by the receiver node





AXIOM Memory allocator

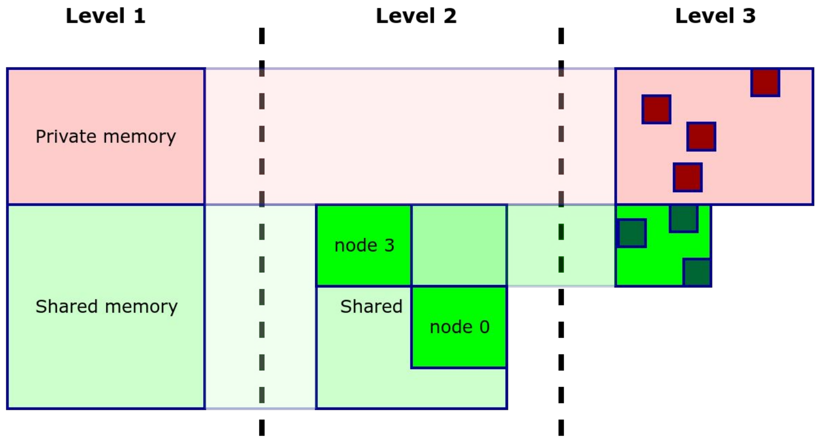


Process Virtual memory layout
0
Application Code + Data + BSS
Not mapped
RDMA
Shared Memory
Private Memory
Kernel
System Physical memory layout
0
Physical memory handled by Linux
Physical memory handled by AXIOM memory allocator
1GB

The AXIOM memory allocator is responsible for the memory subsystem used by the AXIOM drivers and by the AXIOM applications

- handle a part of the memory of each node in the AXIOM cluster in a way compatible with the RDMA support of the AXIOM NIC
- reserve a dedicated range of contiguous physical memory
- **Two kind of memory** can be allocated:
    - **Private memory**
        - guarantees unique address ranges only on the node requesting it (that is, two nodes may end up allocating private memories at the same virtual address)
    - **Shared memory**
        - guarantees that the range of memory allocated is unique among all the AXIOM cluster

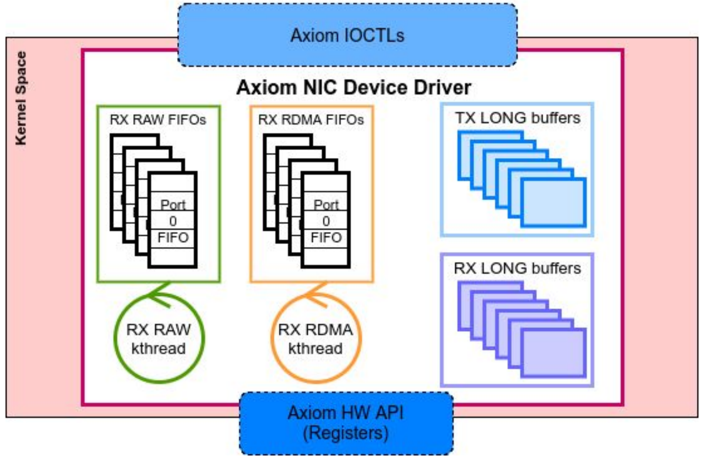
3-level allocator



The AXIOM allocator is internally composed by three levels:

1. **Level 1** is responsible for reserving regions of memory at **cluster level**.
   a. The idea is that this reservation is only inquired at start/end of an application to reserve the maximum (shared or private) memory used by the application
2. **Level 2** is responsible to allocate **macro-blocks of shared memory** to specific nodes.
   a. Macro-blocks are reserved only when the previously allocated blocks are full
3. **Level 3** is finally responsible of each **single allocation** of private/shared memory

AXIOM NI Device Driver



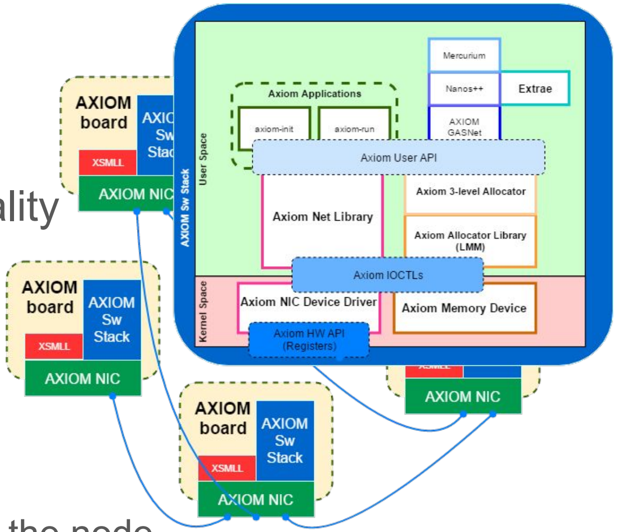
The main components of the NIC kernel driver are:

- A set of **software queues** (one per port)
  - for the small messages and for the descriptors of the long messages;
  - A RDMA queue to store the descriptors of the RDMA requests;
- A **pool of pre-allocated descriptors** to be used for automatically allocate LONG messages upon their arrivals;
- A set of **kernel threads** that are responsible for polling the incoming message queues, demultiplexing their content into local kernel-level buffers, and for filling the long message descriptor FIFOs.

AXIOM UserSpace management

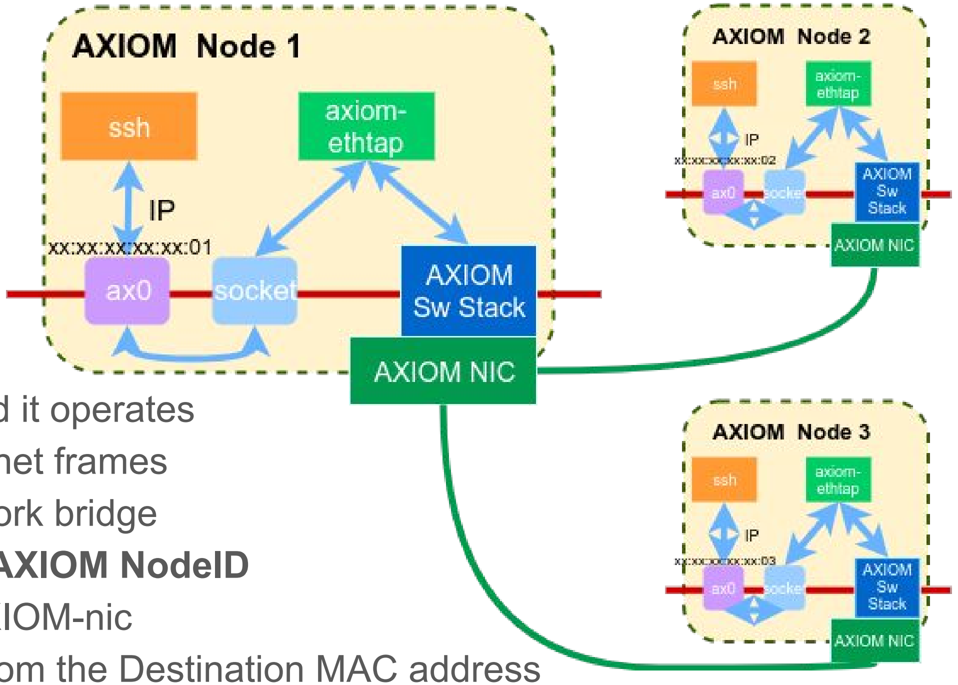
The AXIOM Network drivers have been developed together with a set of applications that complement the driver functionality

- axiom-init daemon
  - includes in user space some of the services that normal networks (like TCP/IP) include in their kernel layers
- axiom-info
  - provide information on the node ID, on the routing table on the local node, and on the set of interfaces available on the node
- axiom-traceroute, axiom-netperf, axiom-ping
  - provide services similar to their Unix counterparts
- axiom-run
  - Process spawner that provides a set of services to AXIOM applications



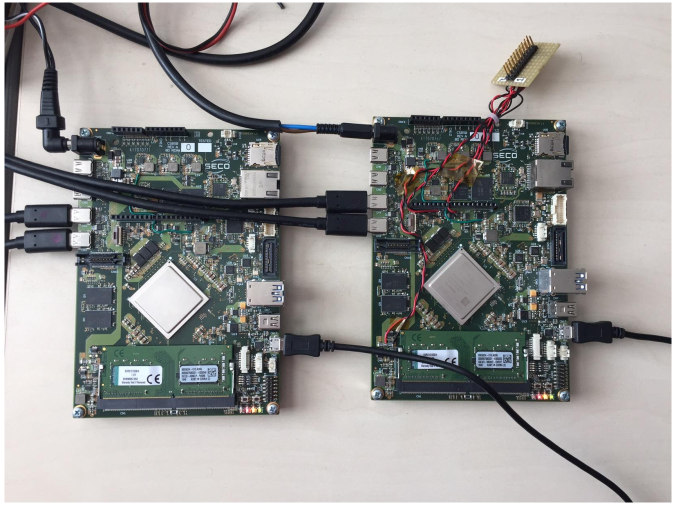
Ethernet over AXIOM

- axiom-ethtap (user space application)
  - creates a TAP interface (ax0)
    - TAP (namely network tap) simulates a link layer device and it operates with “layer 2 packets” like Ethernet frames
    - TAP is used for creating a network bridge
  - **Set last byte of MAC address with AXIOM NodeID**
  - Forwards ethernet frames through AXIOM-nic
    - Destination NodeID extracted from the Destination MAC address



Performance

- Actual throughput performance:
  - **RDMA async**
    - 9.6 Gbps
  - **RDMA sync**
    - 7.7 Gbps
  - **LONG**
    - 1.1 Gbps
  - **LONG multi-thread** (8 threads)
    - 3.7 Gbps
  - **RAW**
    - 0.285 Gbps



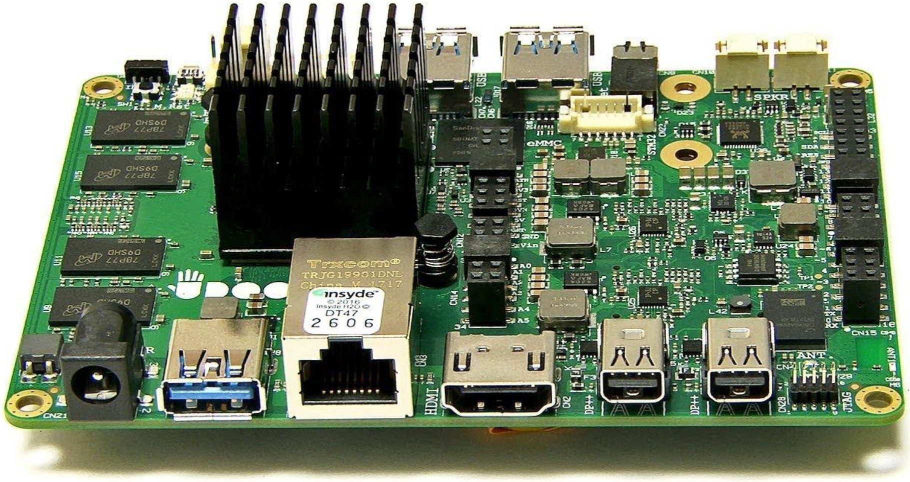
Latest result:
15 Gbps for RDMA transfer using double lane on a single USB-C cable! (under develop)

LINUX DAY A PISA 2017

Benchmark

- **MatrixMultiply** with **OmpSS@cluster** over **AXIOM conduit**
  - https://git.axiom-project.eu/axiom-evi/blob/master/tests/ompss/src/ompss_evimm.c
  - Block Size = 4 - Matrix Size = 800
    - 2 boards (3 working threads per board)
      - Execution time: **5296 msec** (*6=31776)
    - 1 board (4 working threads)
      - Execution time: **7852 msec** (*4=31408)
  - Block Size = 4 - Matrix Size = 1000
    - 2 boards (3 working threads per board)
      - Execution time: **11020 msec** (*6=66120)
    - 1 board (4 working threads)
      - Execution time: **16412 msec** (*4=65648)

Cluster with UDOO x86

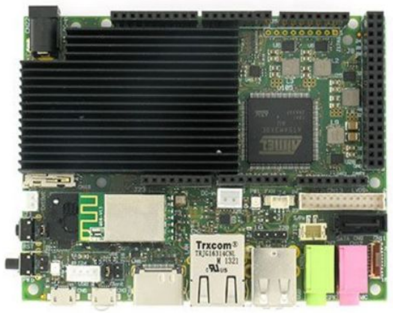
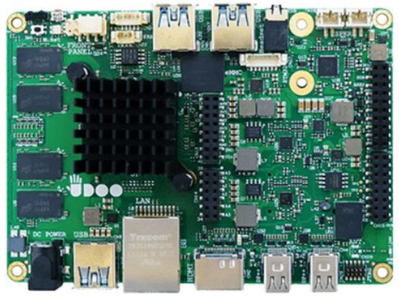


UDOO boards

UDOO boards are made in Tuscany!

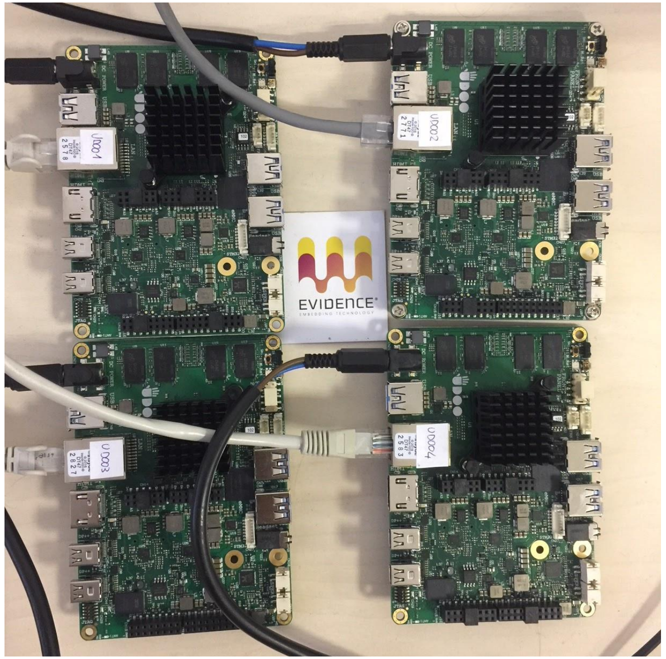
More details: https://www.udoo.org/

MADE IN TUSCANY

UDOO NEO	UDOO QUAD/DUAL	UDOO X86
		

UDOO x86 Cluster

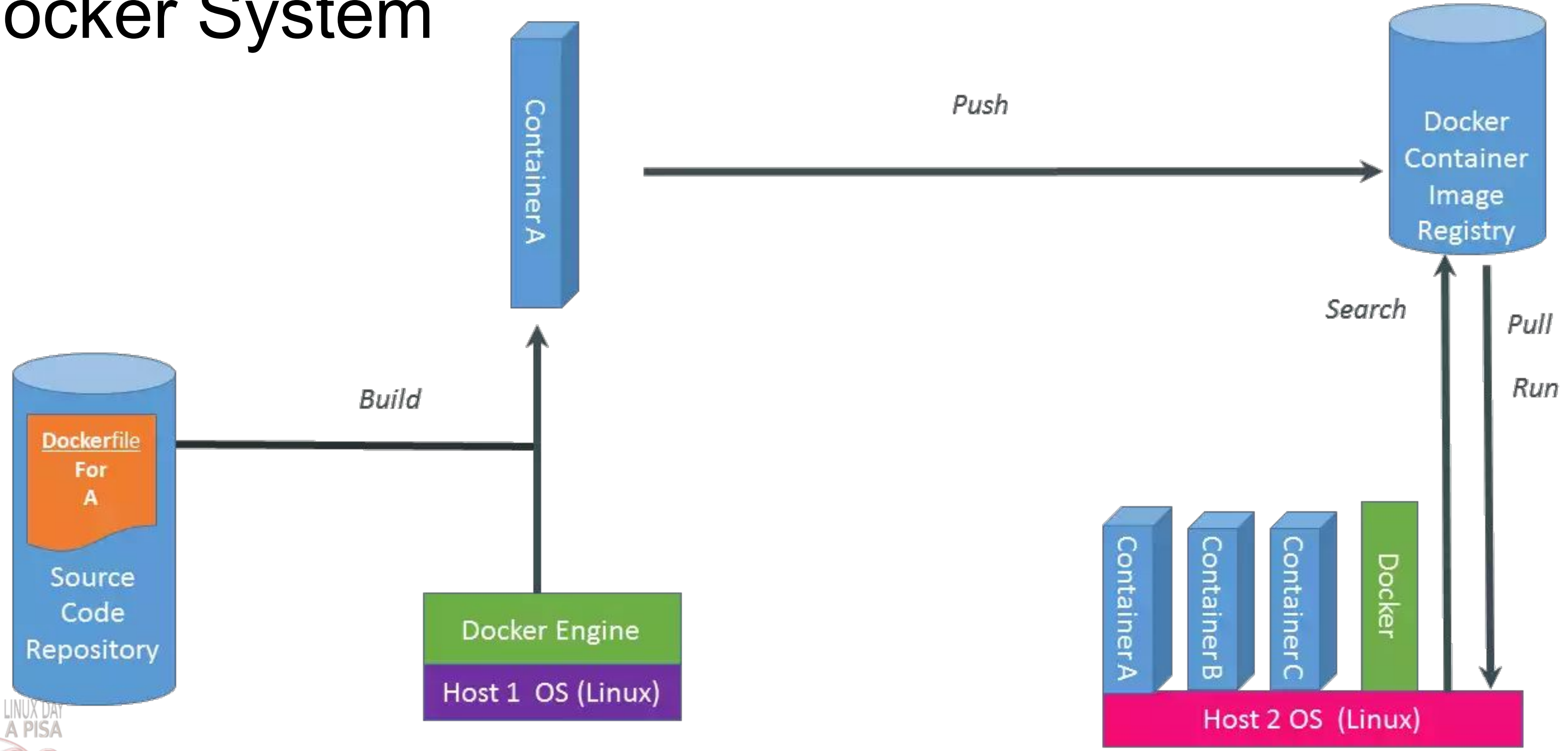
- 4 x UDOO x86 Advanced Plus
  - Intel Celeron N3160 2.24 Ghz (4 cores)
  - 4 GB DDR3L Dual Channel
  - Intel HD Graphics 400
  - 32GB eMMC storage
  - Gigabit Ethernet connector
- Ubuntu 16.04.03 LTS
  - Docker CE
    - Docker Swarm
    - Docker Image with distcc and gcc
      - one image for each version of gcc (gcc5, gcc6, gcc7, cross-gcc, ...)



Docker CE

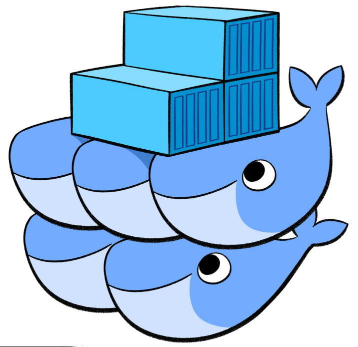
- Docker is the world's leading software containerization platform
- Package your application into a standardized unit for software development
    - wrap a piece of software in a complete filesystem that contains everything needed to run
        - code, runtime, system tools, system libraries – anything that can be installed on a server
    - This guarantees that the software will always run the same, regardless of its environment.

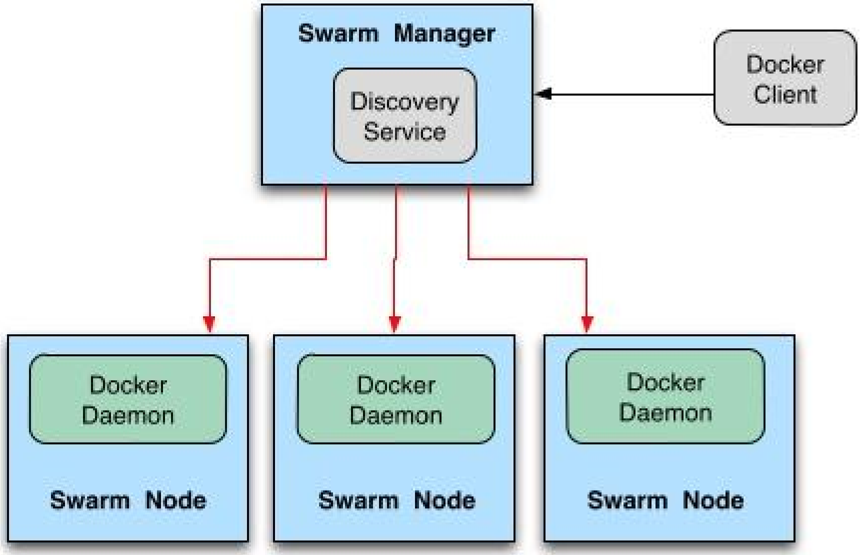
Docker System



Docker Swarm

- Cluster management integrated with Docker Engine
- Scaling
- Desired state reconciliation
- Rolling updates





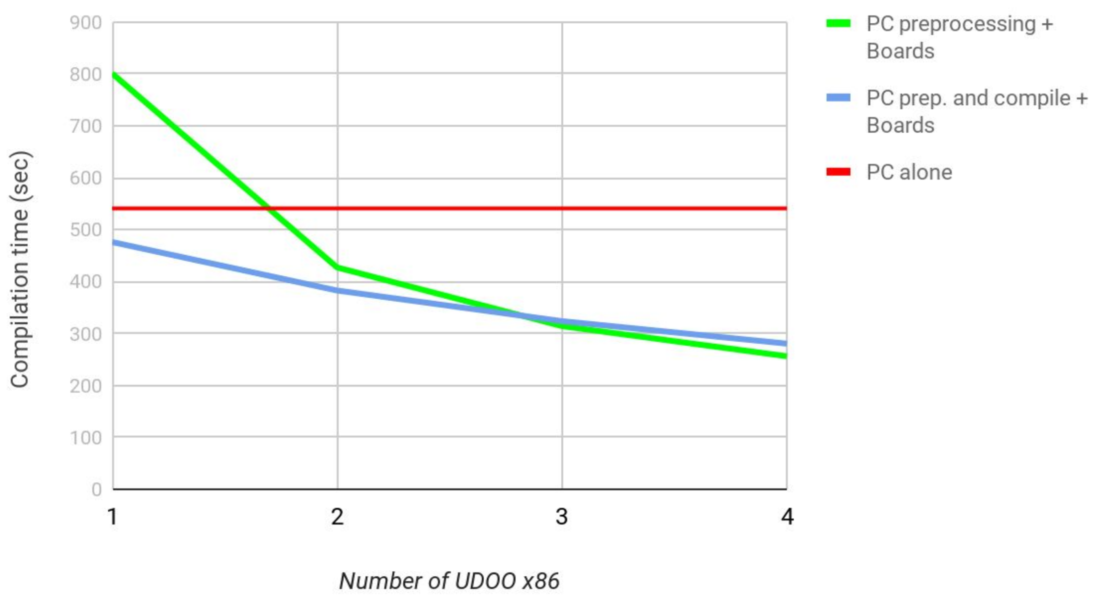
distcc

- `distcc` is a program to distribute compilation of C or C++ code across several machines on a network
- `distcc` is designed to be used with GNU make's parallel-build feature (-j)
- https://github.com/distcc/distcc
- future work
  - Icecream - https://github.com/icecc/icecream
  - based on distcc
  - uses a central server that dynamically schedules the compile jobs to the fastest free server

Performance

- Linux - tag: v4.13
  - PC = Lenovo L450
    - Intel i5-5200U CPU @ 2.20GHz (4 cores - HT enable)
    - 8 GB RAM
  - Lenovo L450 + 4 x UDOO x64

Linux compilation [v4.13 - x86_64_defconfig]



- Compilation
  - `make x86_64_defconfig`
  - `export DISTCC_HOSTS="udoo1:44002 udoo2:44002 udoo3:44002 udoo4:44002"`
  - `make -j X CC="distcc gcc" CXX="distcc g++"`
    - X = number of cores in the cluster

Working with Evidence

```
char msg[]={78, 111, 119, 32, 72, 105, 114, 105, 110, 103, 0};
```

- Master/PhD, Engineering/Computer Science
- C/C++/Qt/Java (Eclipse Ecore/Android)/Control engineers/Python

Passion^3!

- Passion for embedded systems
- Passion for Linux internals
- Passion for software architectures

We are looking for good programmers, geeks allowed!

Questions?

Evidence Srl
Via Carducci 56
56010 S.Giuliano Terme
Pisa - Italy

Web: http://www.evidence.eu.com
E-mail: info@evidence.eu.com
Phone: +39 050 99 11 224