WS on Models, Algorithms and Methodologies for Hierarchical Parallelism in new HPC Systems

The High Performance Internet of Things: using GVirtuS for gluing cloud computing and ubiquitous connected devices

G. Laccetti°, R.Montella*

C. Palmieri#*, V. Pelliccia#*

*Department of Science and Technologies – University of Napoli Parthenope
° Department of Mathematics and Applications – University of Napoli Federico II
#Developer Team members

Motivations

- This **preliminary** work is addressed to speculate about:
  - The next generation of *"off the shelf Beowulf clusters"*

"neoWulf"

  - How to **accelerate** the Internet of Things component: **High Performance Internet of Things**

OpenCV

OpenCL

Generic Virtualization Service

(since March 2010)

- Framework for split-driver based abstraction components
- Plug-in architecture

- **Independent from:**
  - Hypervisor (or no-hypervisor)
  - Communication
  - Target of virtualization
  - **Architecture!**

- **High performance:**
  - Enabling transparent virtualization
  - With overall performances better or not too far from un-virtualized resources

GVirtuS

http://osl.uniparthenope.it/projects/gvirtus/

Generic Virtualization Service

(since March 2010)

- From Google Scholar…

A GPGPU transparent virtualization component for high performance computing clouds
G Giunta, R Montella, G Agrillo, G Coviello
Euro-Par 2010-Parallel Processing, 379-391

A GPU Accelerated High Performance Cloud Computing Infrastructure for Grid Computing Based Virtual Environmental Laboratory
G Giunta, R Montella, G Laccetti, F Isaila, JG Blas

Virtualizing general purpose GPUs for high performance cloud computing: an application to a fluid simulator
R Di Lauro, F Giannone, L Ambrosio, R Montella
Parallel and Distributed Processing with Applications (ISPA), 2012 IEEE 10th ...

A general-purpose virtualization service for HPC on cloud computing: an application to GPUs
R Montella, G Coviello, G Giunta, G Laccetti, F Isaila, JG Blas
Parallel Processing and Applied Mathematics, 740-749

SlaaS-Sensing Instrument as a Service Using Cloud Computing to Turn Physical Instrument into Ubiquitous Service
R Di Lauro, F Lucarelli, R Montella
Parallel and Distributed Processing with Applications (ISPA), 2012 IEEE 10th ...

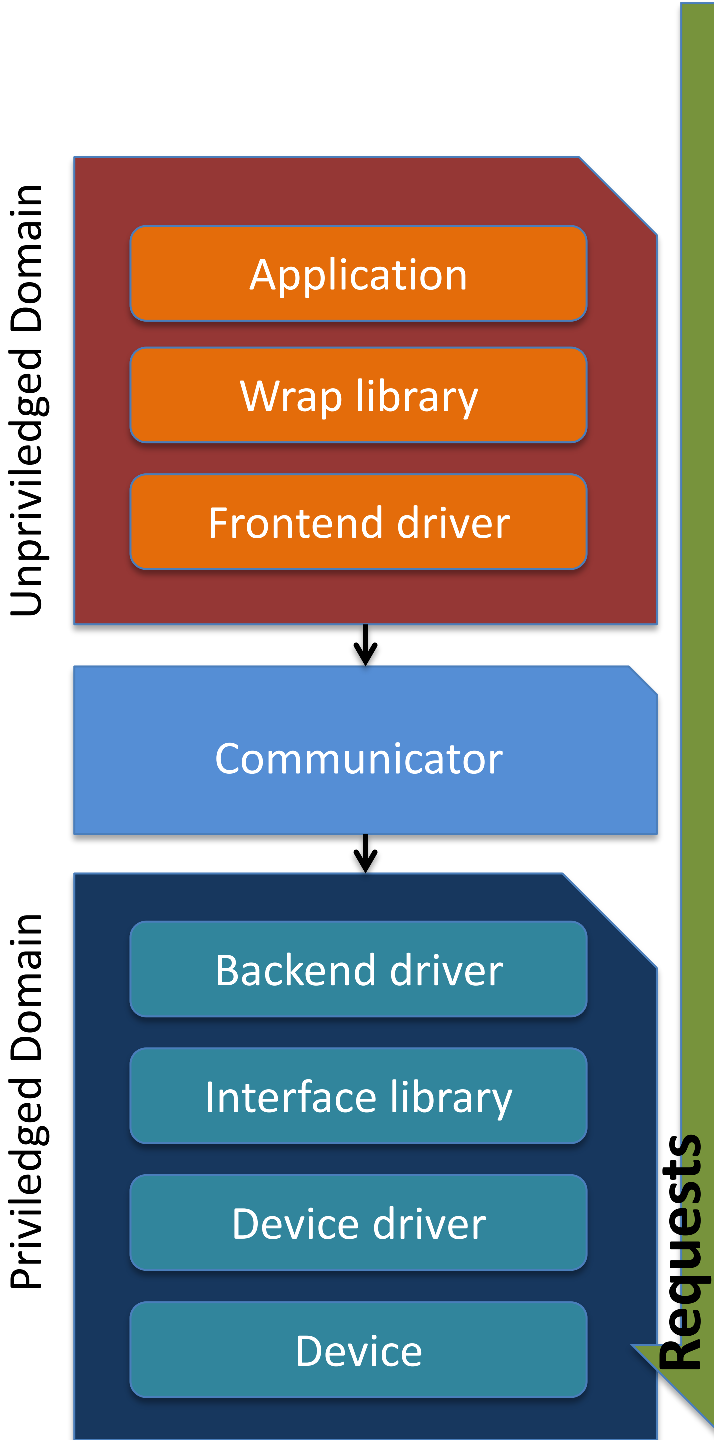
Cite Us!

GVirtuS

http://osl.uniparthenope.it/projects/gvirtus/

Split-Driver approach

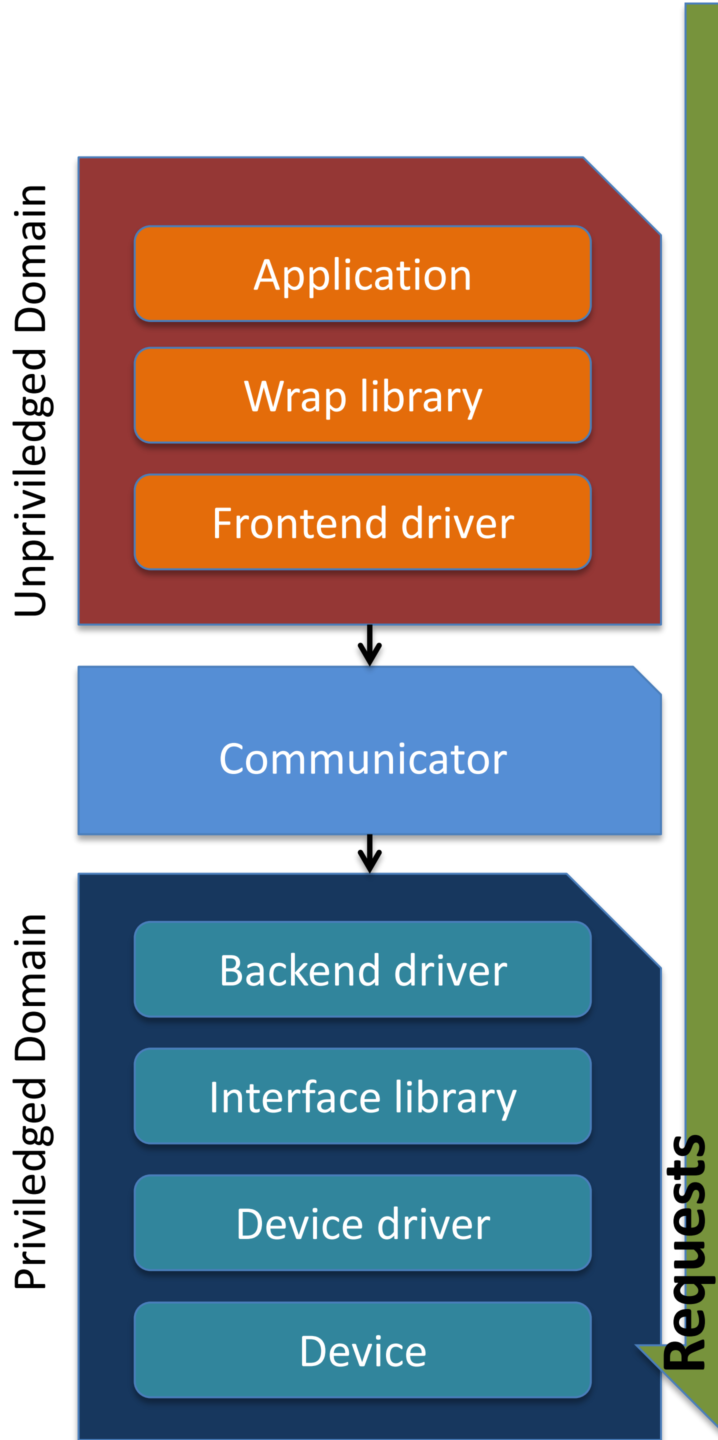
- Split-Driver
  - Hardware access by privileged domain.
  - Unprivileged domains access the device using a frontend/backend approach
- Frontend (FE):
  - Guest-side software component.
  - Stub: redirect requests to the backend.
- Backend (BE):
  - Manage device requests.
  - Device multiplexing.



GVirtuS approach

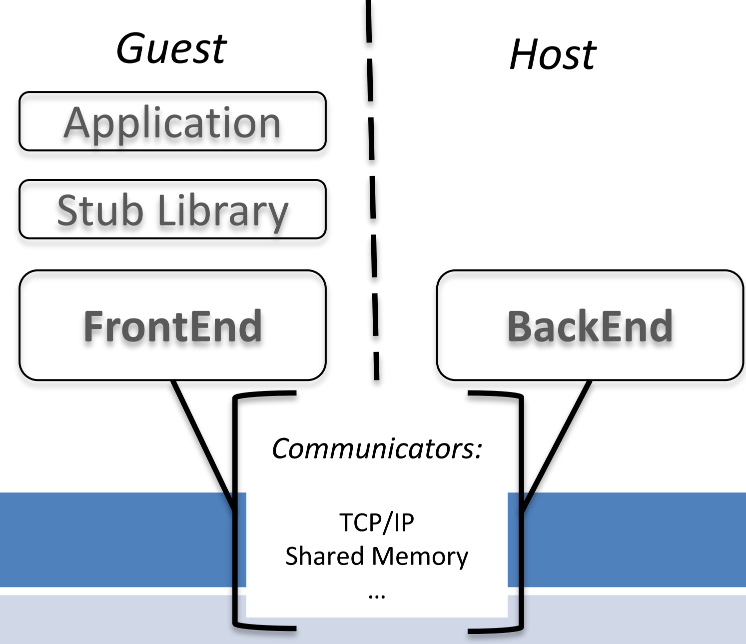
- **GVirtuS Frontend**
  - Dynamic loadable library
  - Same application binary interface
  - Run on guest user space

- **GVirtuS Backend**
  - Server application
  - Run in host user space
  - Concurrent requests



The Communicator

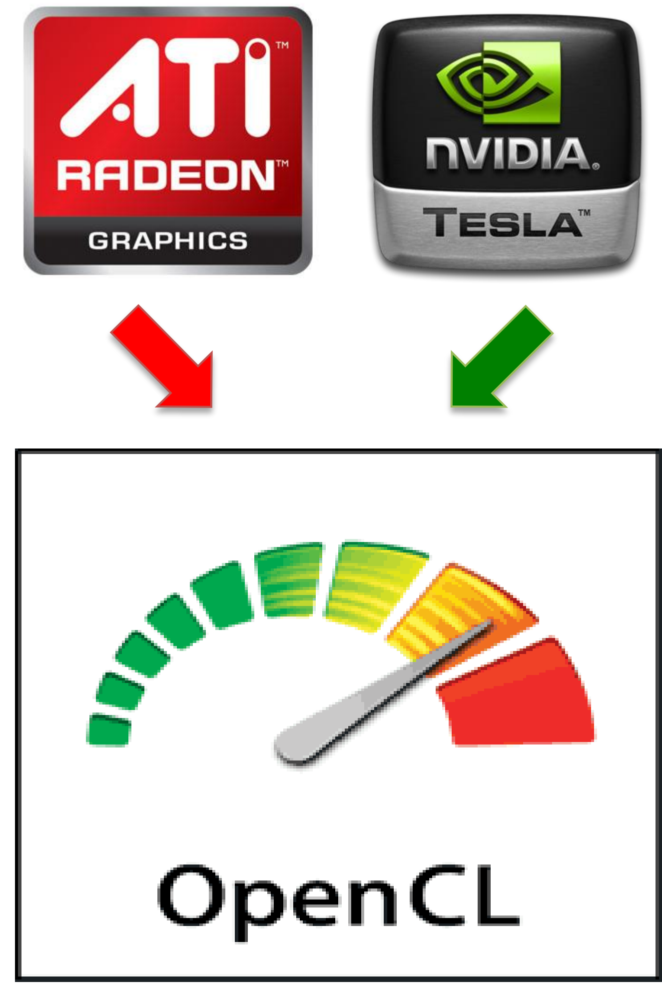
- Provides a high performance communication between virtual machines and their hosts.
- The choice of the hypervisor deeply affects the efficiency of the communication.



Hypervisor	FE/BE comm	Notes
No hypervisor	Unix Sockets	Used for testing purposes
Generic	TCP/IP	• Communication testing purposes • **Remote / Distributed virtualized resources (i.e. GPUs)** • **High Performance Internet of Things**
Xen	XenLoop	• runs directly on the top of the hardware through a custom Linux kernel • provides a communication library between guest and host machines • implements low latency and wide bandwidth TCP/IP and UDP connections • app transparent and offers an automatic discovery of the supported VMs
VMware	Virtual Machine Communication Interface (VMCI)	• commercial hypervisor running at the application level • provides a datagram API to exchange small messages • a shared memory API to share data • an access control API to control which resources a virtual machine can access • and a discovery service for publishing and retrieving resources
KVM/QEMU	VMchannel	• Linux loadable kernel module now embedded as a standard component • supplies a high performance guest/host communication • based on a shared memory approach

An application: Virtualizing GPUs

- **GPUs**
  - Hypervisor independent
  - Communicator independent
  - **GPU independent**



The host plus a collection of devices managed by the OpenCL framework that allow an application to share resources and execute kernels on devices in the platform.

GVirtuS – libOpenCL.so

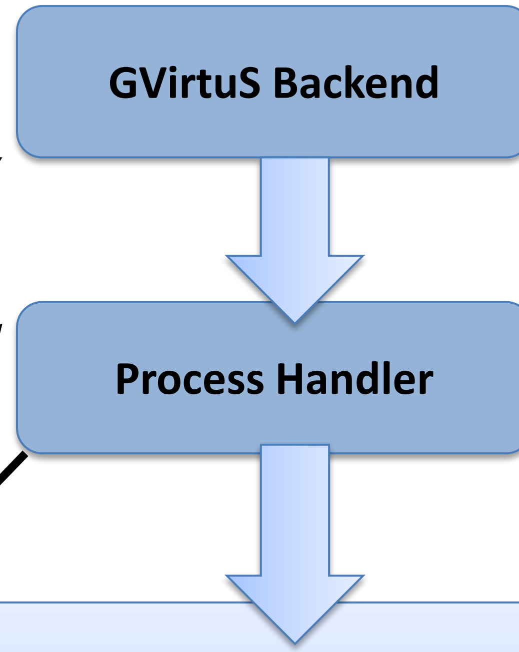
Guest Machine

```
#include <stdio.h>
#include <CL/cl.h>
int main(void) {
    cl_int error, platforms;
    cl_platform_id platform ;
    error=clGetPlatformIDs(1, &platform, &platforms
);
    printf("Number of platforms GPU(s): %d\n",
platforms);
    return 0;
}
```

GVirtuS Frontend

```
extern "C" CL_API_ENTRY cl_int CL_API_CALL
clGetPlatformIDs(cl_uint num_entries,
                 cl_platform_id *platforms,
                 cl_uint *num_platforms){
 OpenclFrontend::Prepare();
 OpenclFrontend::
 AddVariableForArguments(num_entries);
 OpenclFrontend::AddVariableForArguments(platforms);
    OpenclFrontend::AddVariableForArguments(num_platf
orms );
OpenclFrontend:: Execute("clGetPlatformIDs");
if(OpenclFrontend:: Success()){
       cl_uint *tmp_num_platform;
       cl_platform_id *tmp_platform;
       tmp_num_platform =
OpenclFrontend::GetOutputHostPointer<cl_uint>();
       if (tmp_num_platform != NULL)
         *num_platforms= *tmp_num_platform;
       tmp_platform=
(OpenclFrontend::GetOutputHostPointer<cl_platform_id
>());
if (tmp_platform !=NULL)
       *platforms=*tmp_platform;    }
```

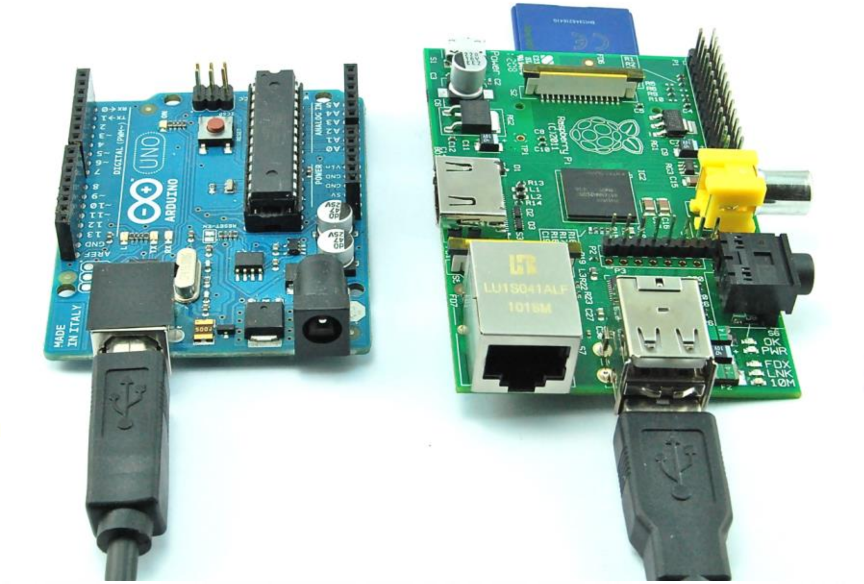
Host Machine




```
OPENCL_ROUTINE_HANDLER(GetPlatformIDs) {
  cl_int num_entries = input_buffer->Get<cl_int>();
  cl_platform_id *platforms=
input_buffer->Assign<cl_platform_id>();
  cl_uint *num_platforms =
input_buffer->Assign<cl_uint> ();
   cl_uint exit_code =
clGetPlatformIDs(num_entries,  platforms,
num_platforms);
    Buffer *out = new Buffer();
    out->Add(num_platforms);
    out->Add(platforms);
    return new Result(exit_code, out);
}
```

Accelerating ARM boards

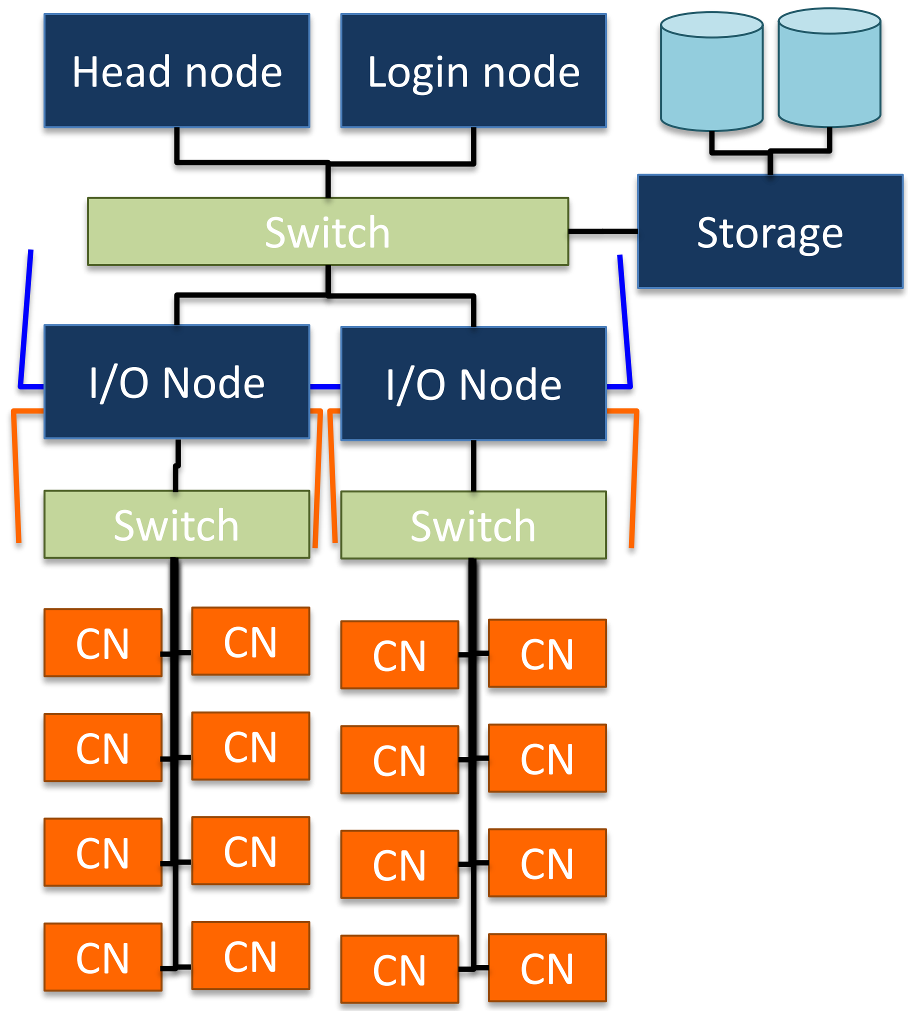
- High Performance Computing will be ARM based
  - Cheaper and powerful
  - Low heat emission
  - High developable



- High Performance Internet of Things
  - Small and smart devices highly pervasive

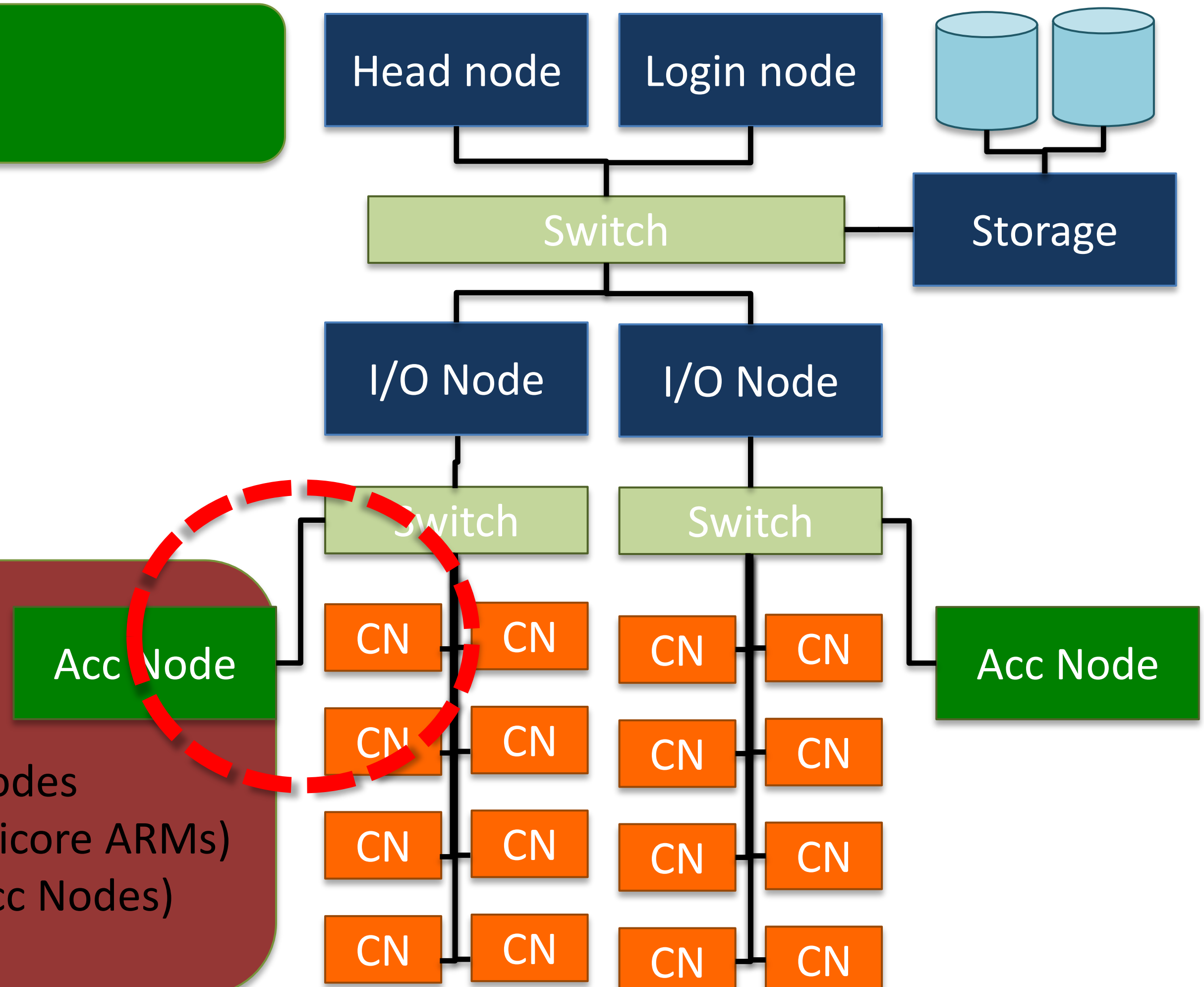
Figuring out the next generation HPC

- x86_64:
  - Head node
  - Login nodes
  - I/O nodes
    - Computing nodes for the cluster
    - I/O nodes for sub clusters
- ARM:
  - Computing nodes as multicore sub clusters



Enhancing the next generation of [low cost/middle end] HPC

GPU:
- **Acceleration nodes**



- High End GPGPU devices
  - Nvidia
    - Tesla
    - Fermi
    - Kepler
  - ATI Radeon

- **Hierarchical parallelism:**
  - DM among I/O Nodes
  - SM in I/O Nodes
  - DM among sub clusters nodes
  - SM in sub cluster CN(multicore ARMs)
  - OpenCL Kernels (in CN/Acc Nodes)

NB: In this scenario we have ARM boards with a low performance GPU useless for HPC jobs

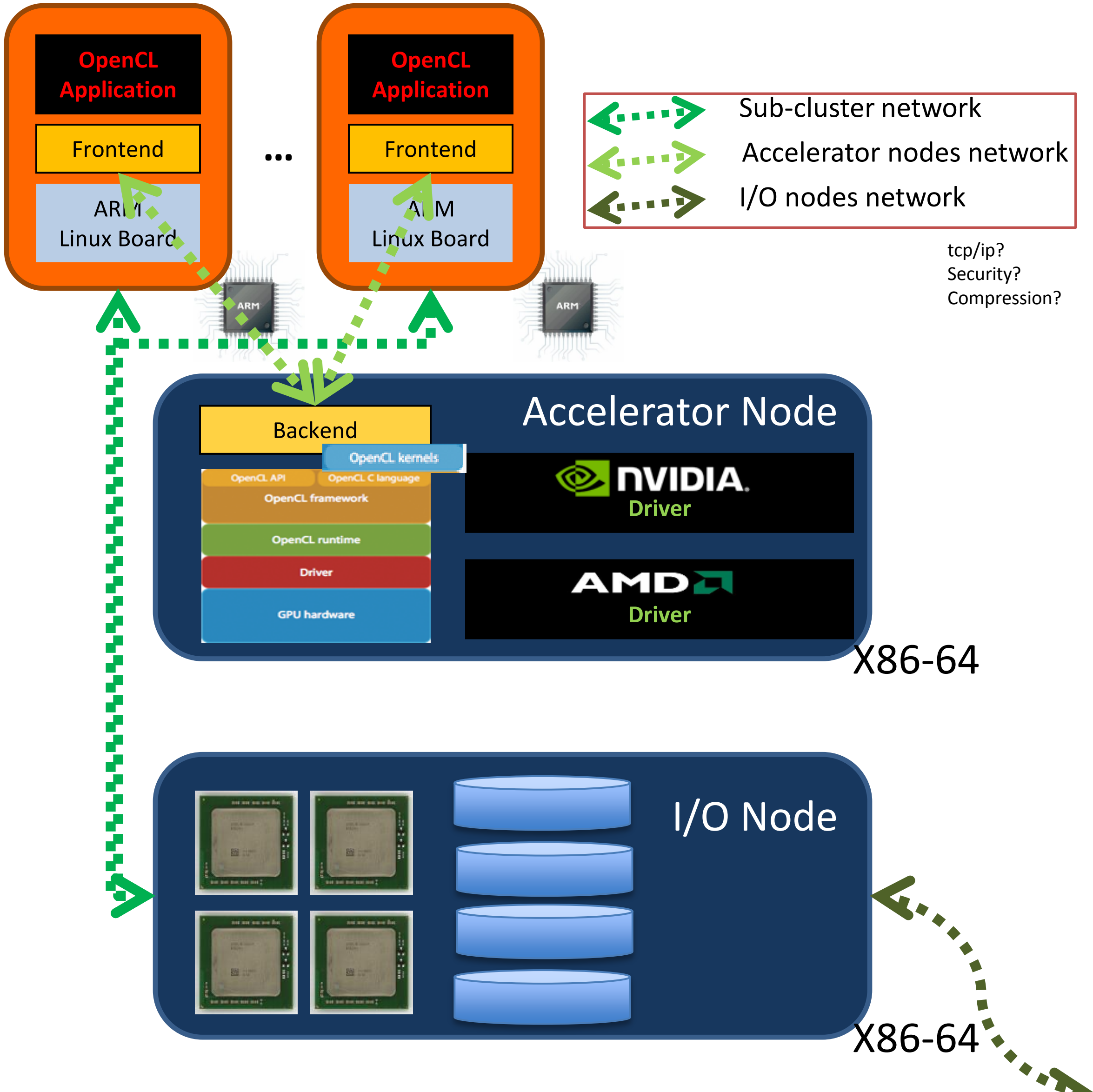
Distributed GPUs

Highlights:

- Using the Tcp/Ip Communicator FE/BE could be on different machines.
- ARM machines can access remote GPUs.

Applications:

- GPU for embedded systems as network machines
- Next generation of High Performance [Cloud] Computing



Prototyping

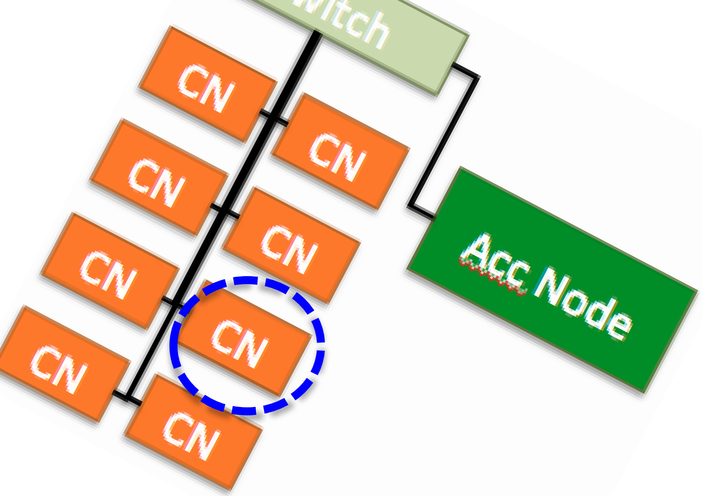
- Computing node:
  - **Raspberry Pi mod. B. rev. 2**
  - Wheezy Raspbian Linux

- Acceleration:
  - Genesis GE-i940 Tesla
  - i7-940 2,93 133 GHz fsb, Quad Core hyper-threaded 8 Mb cache CPU and 12Gb RAM.
  - 1 nVIDIA Quadro FX5800 4Gb RAM video card
  - **2 nVIDIA Tesla C1060 4 Gb RAM**

- I/O:
  - Intel Xeon
    quad core HT
  - Ubuntu Linux 64bit

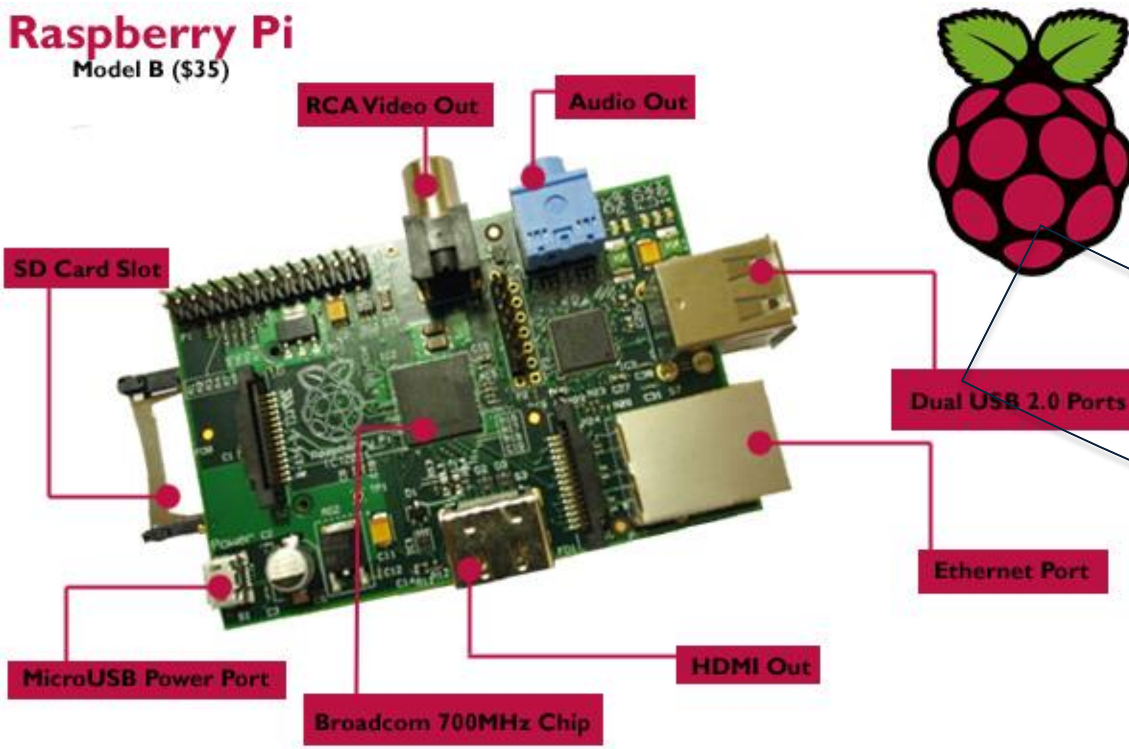

Tesla C1060 Computing Processor

Processor	1 x Tesla T10
Number of cores	240
Core Clock	1.33 GHz
On-board memory	4.0 GB
Memory bandwidth	102 GB/sec peak
Memory I/O	512-bit, 800MHz GDDR3
Form factor	Full ATX: 4.736" (H) x 10.5" (L) Dual slot wide
System I/O	PCIe x16 Gen2
Typical power	160 W



Evaluation

- Single computing node

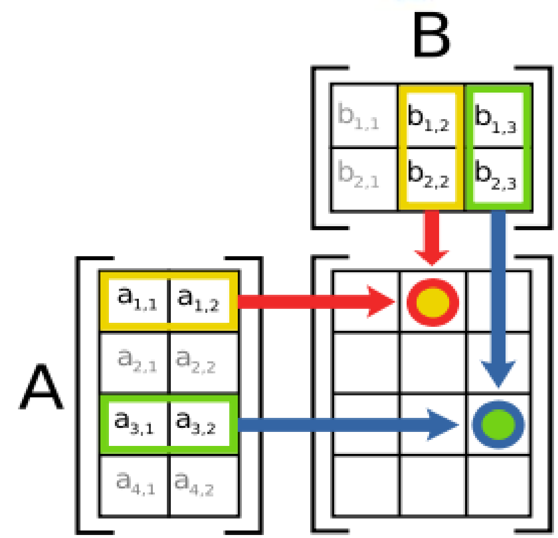


…from (NVIDIA) OpenCL SDK…

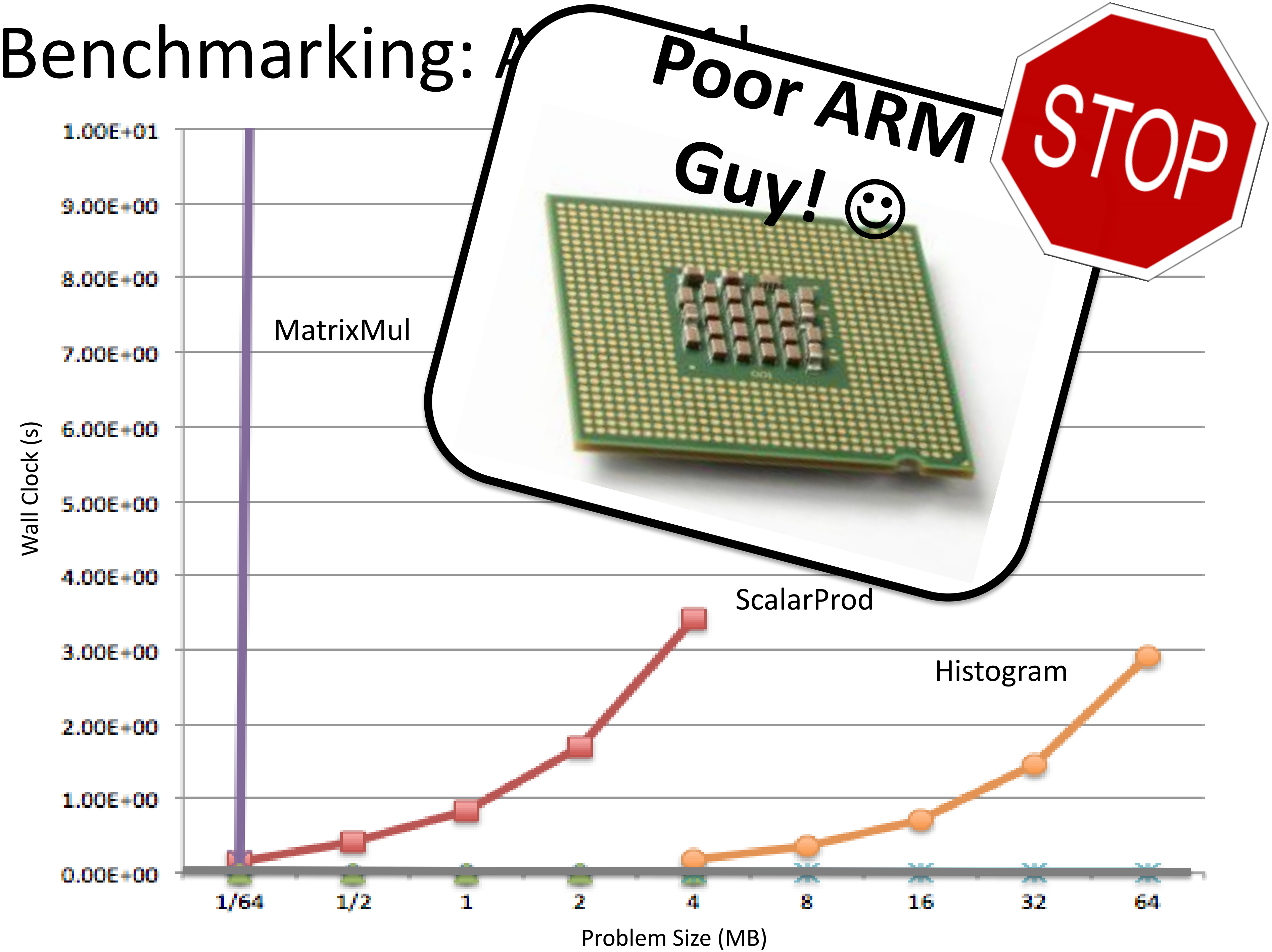
- **ScalarProd** computes *k* scalar products of two real vectors of length *m*.
  - Notice that each product is executed by an OpenCL thread on the GPU.
  - No synchronization is required.

- **MatrixMul** computes a matrix multiplication.
  - The matrices are *mxn* and *nxp*, respectively.
  - It partitions the input matrices in blocks and associates an OpenCL thread to each block.
  - No need of synchronization.

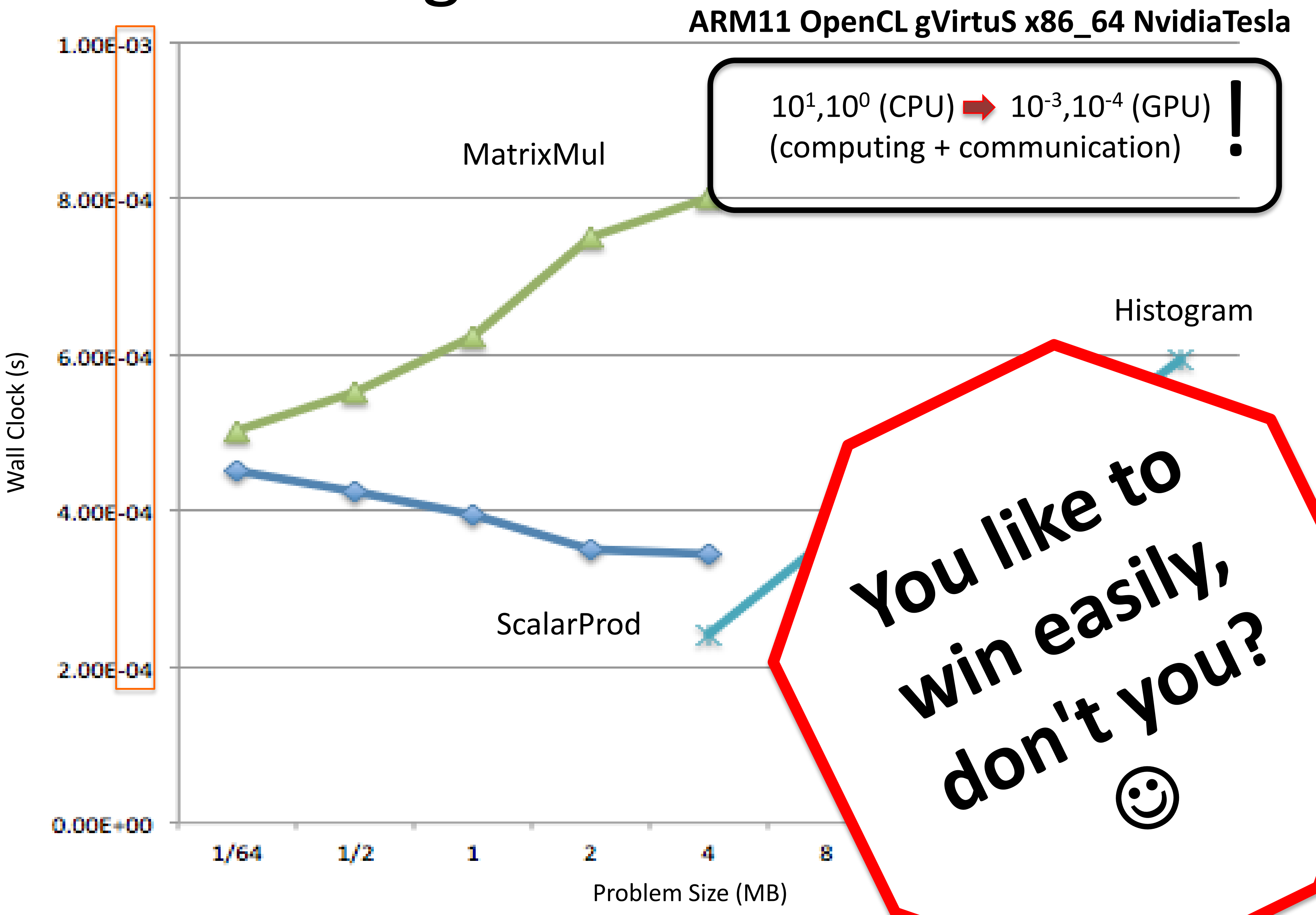
- **Histogram** returns the histogram of a set of *m* uniformly distributed real random numbers in 64 bins.
  - The set is distributed among the OpenCL threads each computing a local histogram.
  - The final result is obtained through synchronization and reduction techniques.



Benchmarking: A



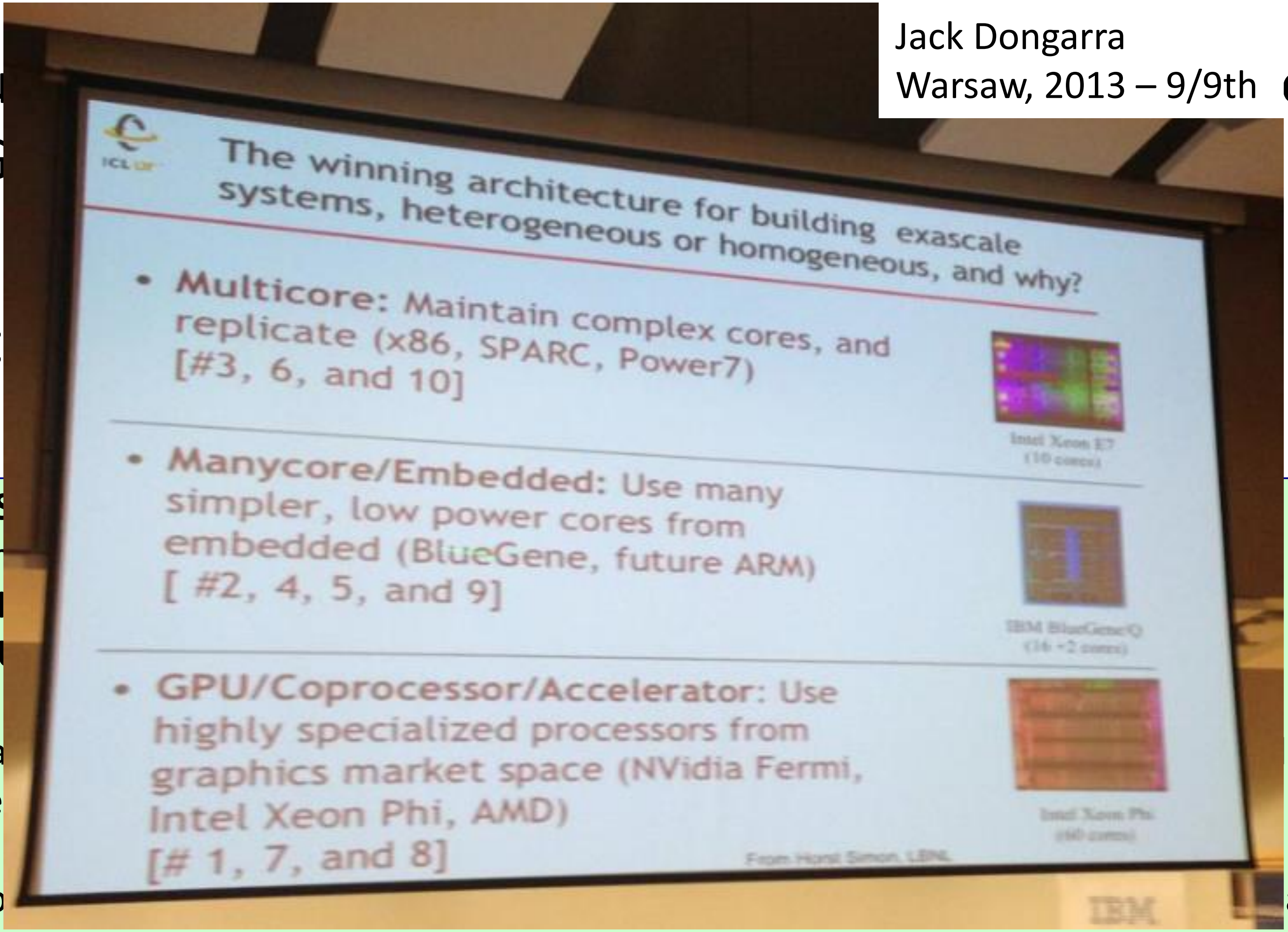
Benchmarking: GPU!



Conclusions and Future Works

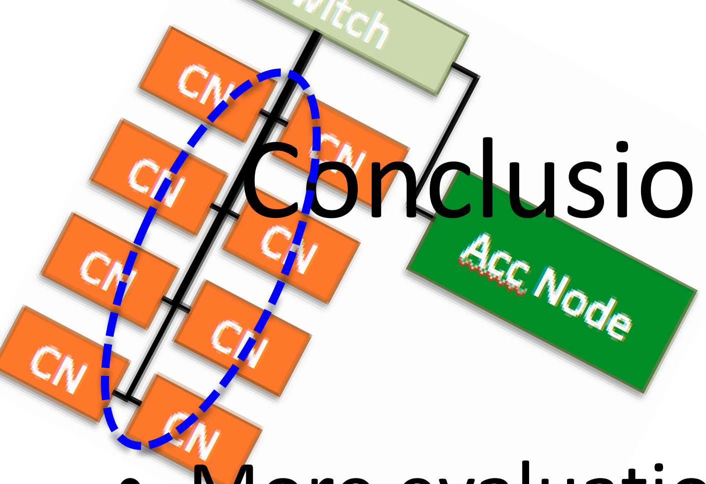
- A su orm
  as G
- Test
  imp

LESSONS
- Techn
  - I
  - U
- Perva in the
  same
- The p al"
  machines

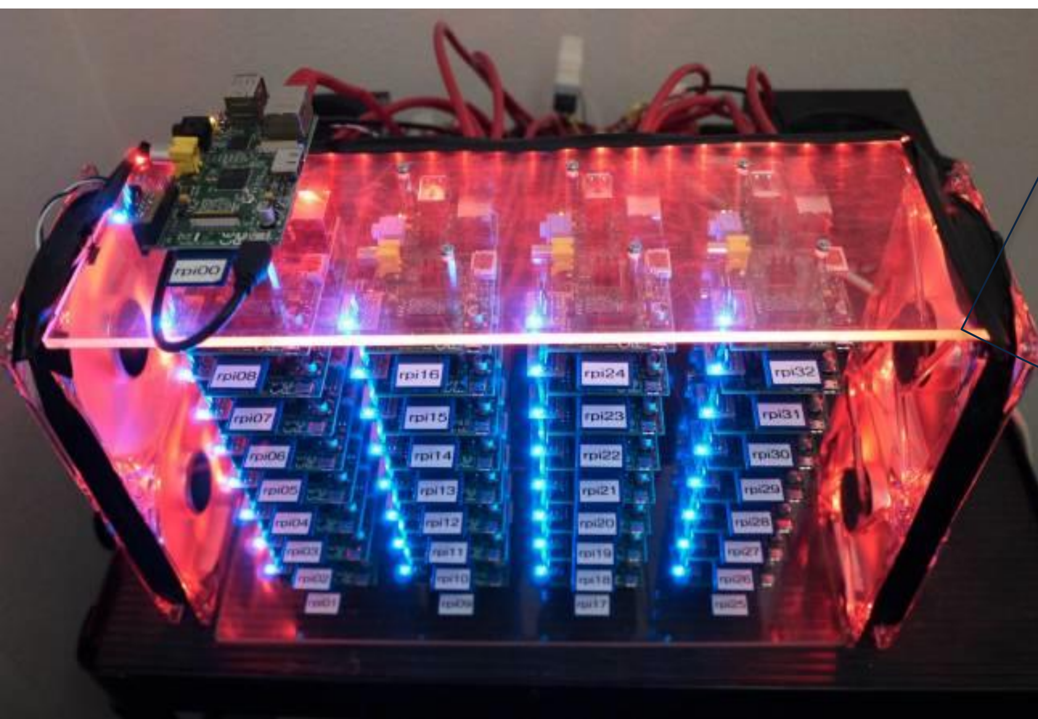


Jack Dongarra
Warsaw, 2013 – 9/9th

Conclusions and Future Works



- More evaluation
- Multiple computing nodes



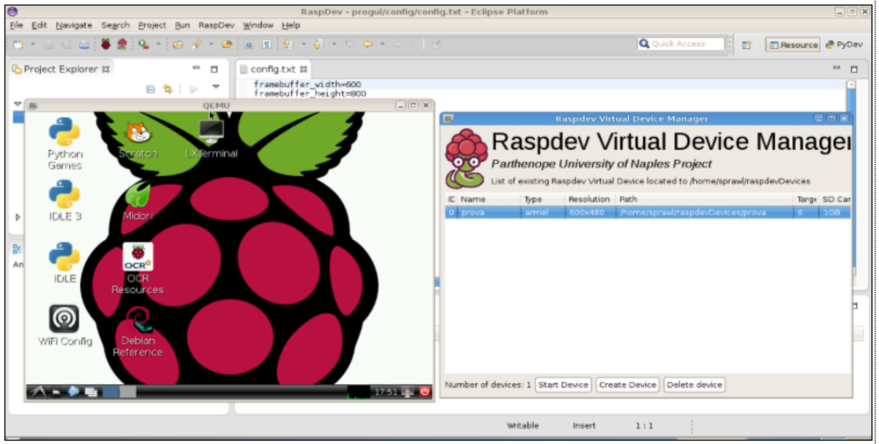
Conclusions and Future Works

- Repeat testing on better ARM board(s) i.e.
  - Quad Core CPU 1.2 GHz
  - 1 GB RAM
  - Giga Ethernet

RaspDev

http://web.uniparthenope.it/raspdev/

- Full OpenCL porting
- Application framework for the ARM “sub cluster” support
- Development of a comfortable SDK for hybrid programming



GVirtuS is GPL3 licensed open source! Download it at http://osl.uniparthenope.it/projects/gvirtus/

The High Performance Internet of Things: using GVirtuS for gluing cloud computing and ubiquitous connected devices

Thank You!

?

http://osl.uniparthenope.it/projects/gvirtus/

GVirtuS