A Real Time Controller for E-ELT

Addressing the jitter/latency constraints

Maxime Lainé, Denis Perret

LESIA / Observatoire de Paris

Project #671662 funded by European Commission under program H2020-EU.1.2.2 coordinated in H2020-FETHPC-2014

Green Flash

RTC prototypes or E-ELT AO system

European project: Horizon 2020

Research and innovation program

3 years project (a year has passed already)

Low cost and low energy consumption RTC

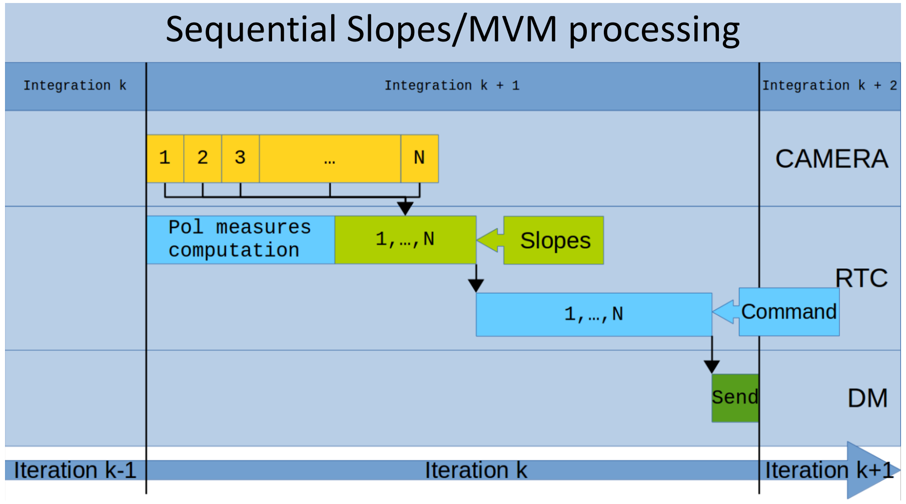
Different kind of accelerators characterization

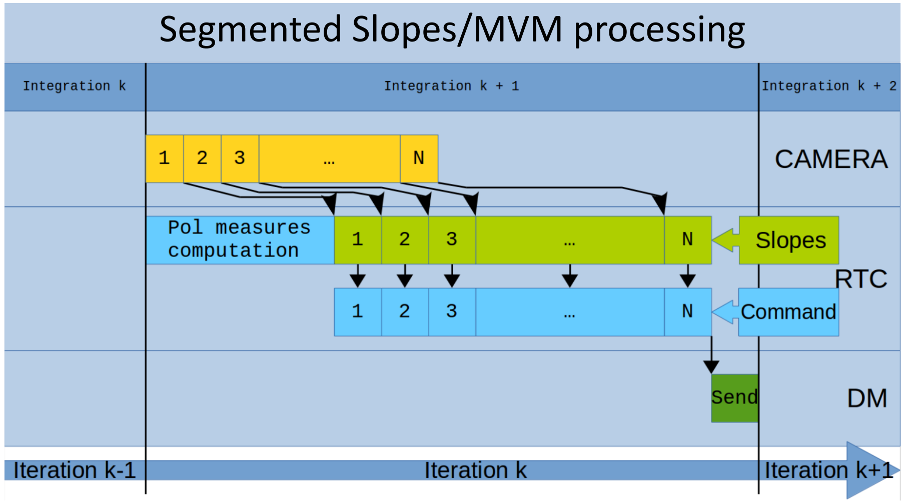
<u>GPU</u> <u>FPGA</u> XeonPhi

Public/private sector partnership

E-ELT – Numbers & Methods

Case	Method	Dimension	Encoding	Frequency	Size	Throughput
MCAO	Shack-Hartmann	1.6k x 1.6k	16 bits	500Hz	40.96Mb (5,12Mo)	20.48 Gb/s
SCAO	Shack-Hartmann	800 x 800	16 bits	1000Hz	10.24Mb (1,28Mo)	10.24 Gb/s
SCAO	Pyramid	240 x 240	16 bits	1000Hz	~1Mb (~125Ko)	1 Gb/s





Dec 21st 2016

E-ELT – Numbers & Methods

Case	Method	Dimension	Encoding	Frequency	Size	Throughput
MCAO	Shack-Hartmann	1.6k x 1.6k	16 bits	500Hz	40.96Mb (5,12Mo)	20.48 Gb/s
SCAO	Shack-Hartmann	800 x 800	16 bits	1000Hz	10.24Mb (1,28Mo)	10.24 Gb/s
SCAO	Pyramid	240 x 240	16 bits	1000Hz	~1Mb (~125Ko)	1 Gb/s

MCAO Case:

40,96 Mb with 40Gb/s network

=> 1ms latency

Latency and jitter constraints

Latency

typical optic fiber: 4,9µs/Km

Closest city 130km (Antofagasta)
implies 0.630ms latency
(+transfer time)

MCAO 2ms/iter, SCAO 1ms/iter

The nearer the better

Jitter

Under 10% of overall latency in order to be “manageable”

Too much jitter

Too much frame skips

Correction stability hard to reach

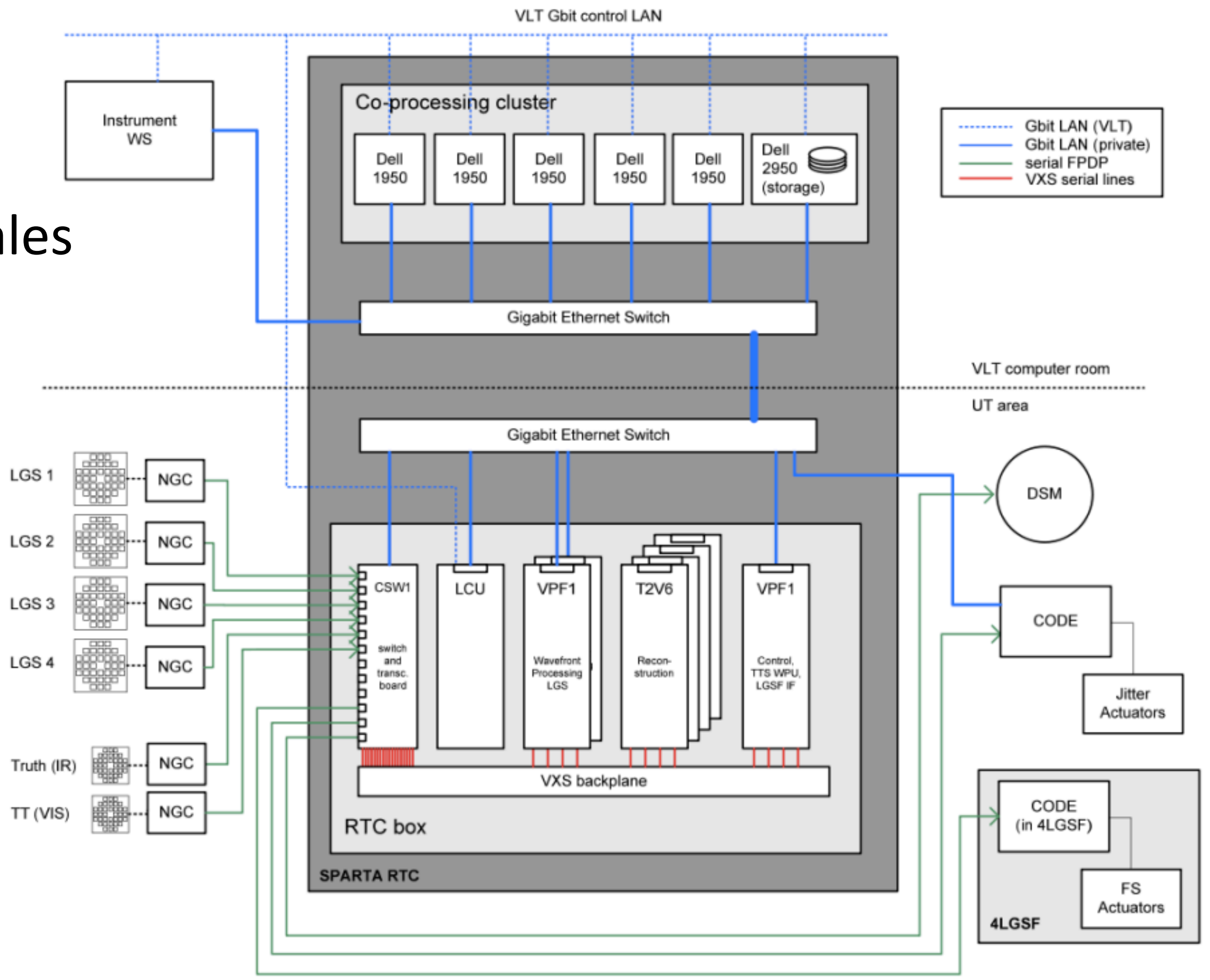
The lesser the better

Usual Telescope RTC – SPARTA

Design for VLT scales

RTC BOX:
x86 CPU
FPGA
DSP

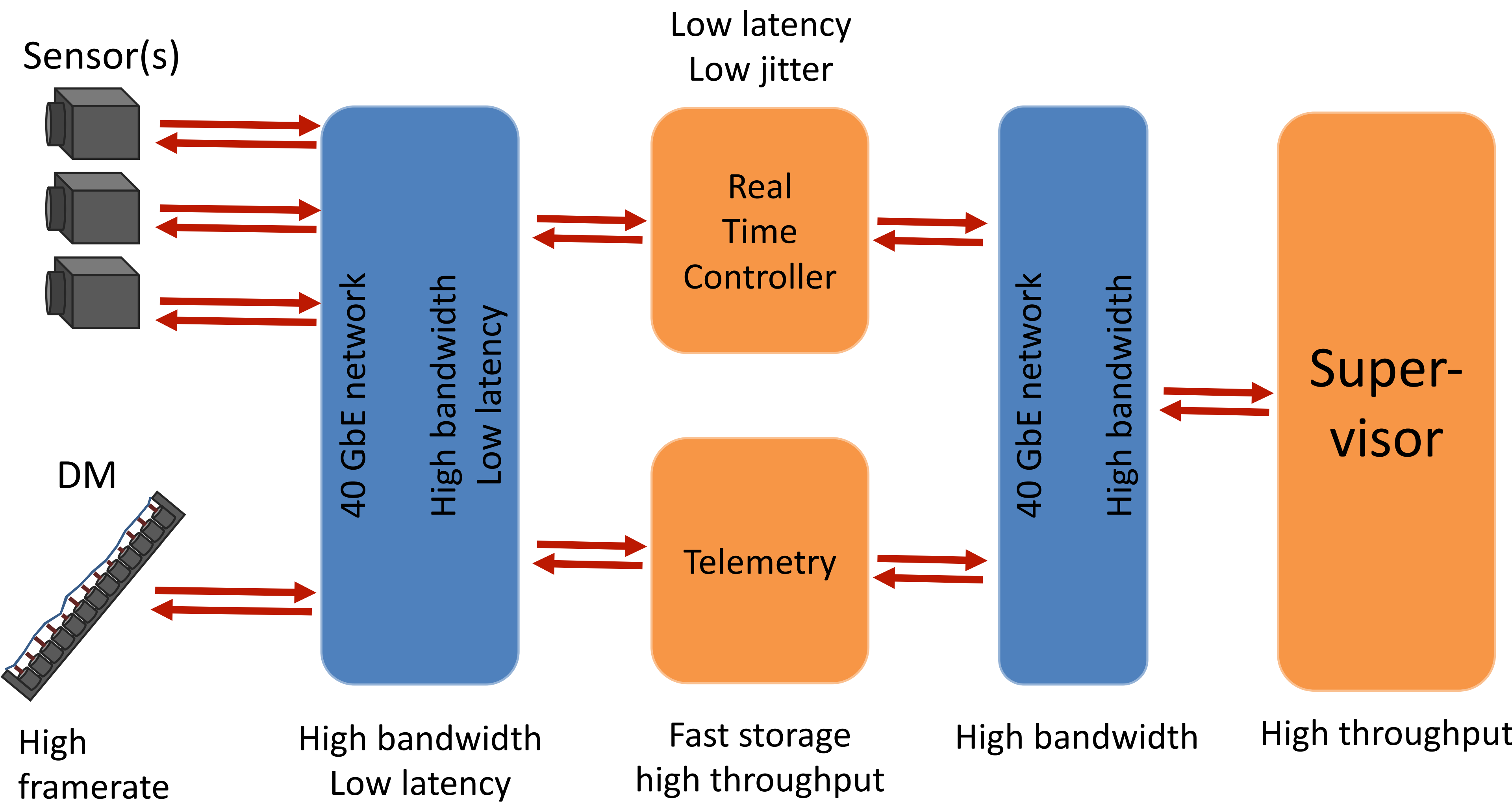
Co-processing Cluster:
X86 CPU



At ELT scales: needs for a “super-calculator” in the observatory

Dec 21st 2016

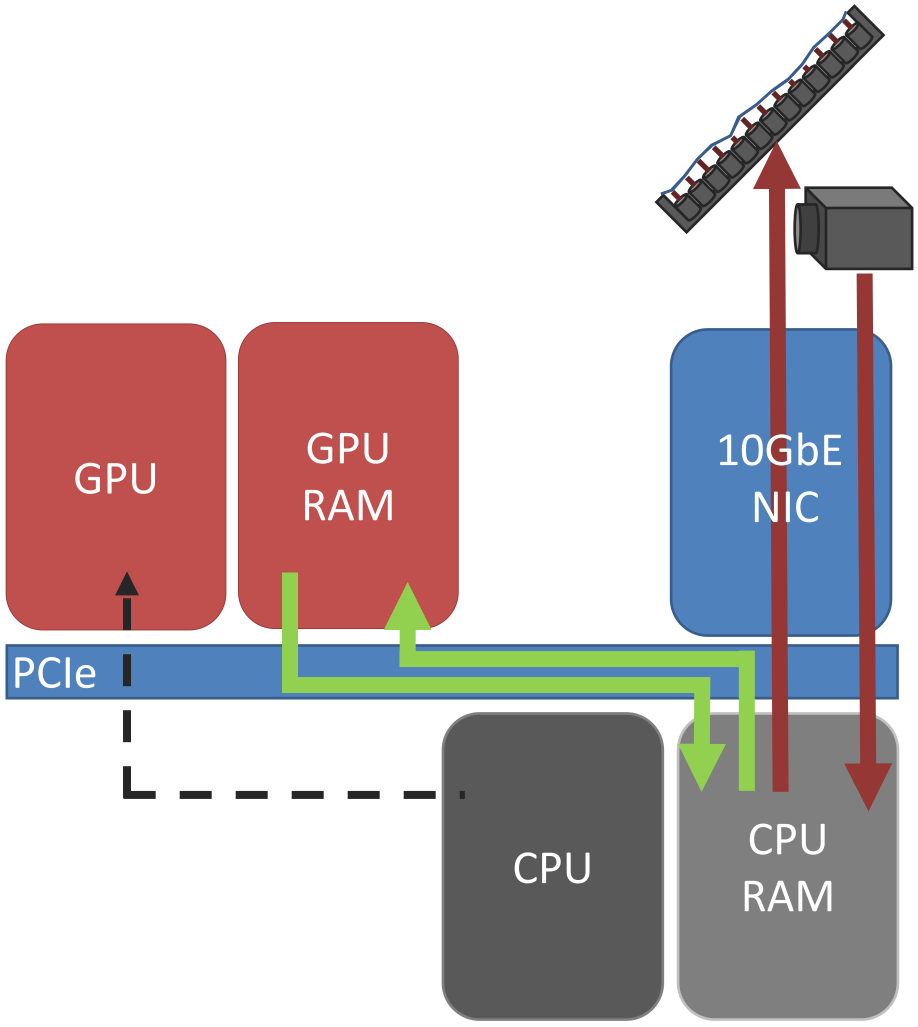
E-ELT – GreenFlash RTC Prototype



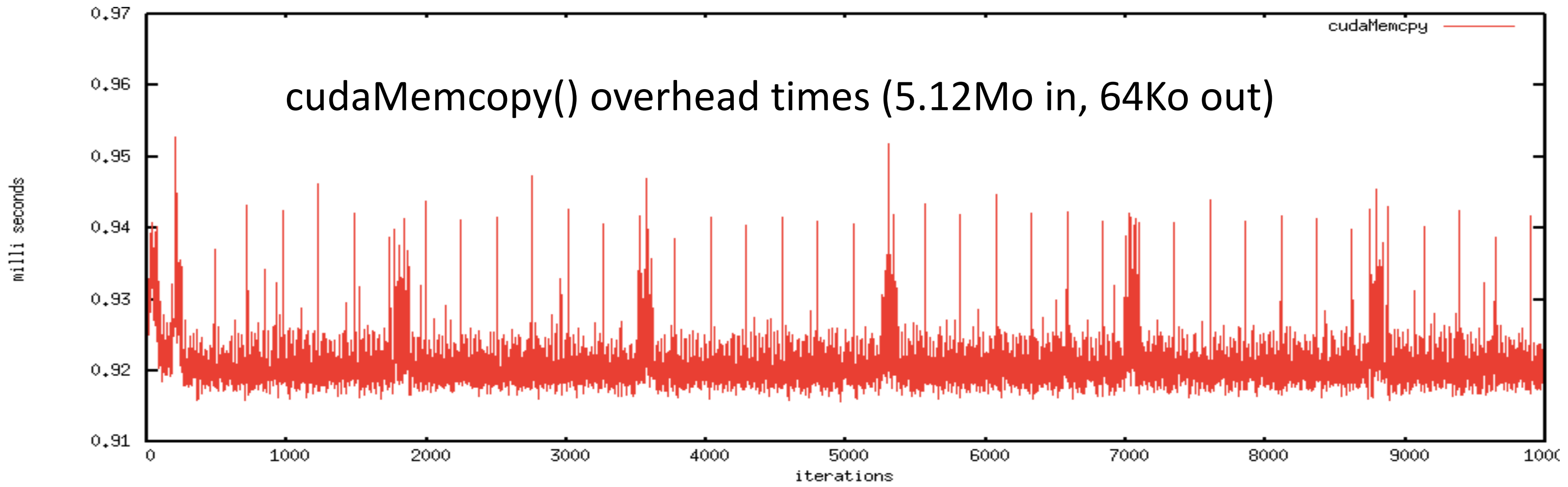
Dec 21st 2016

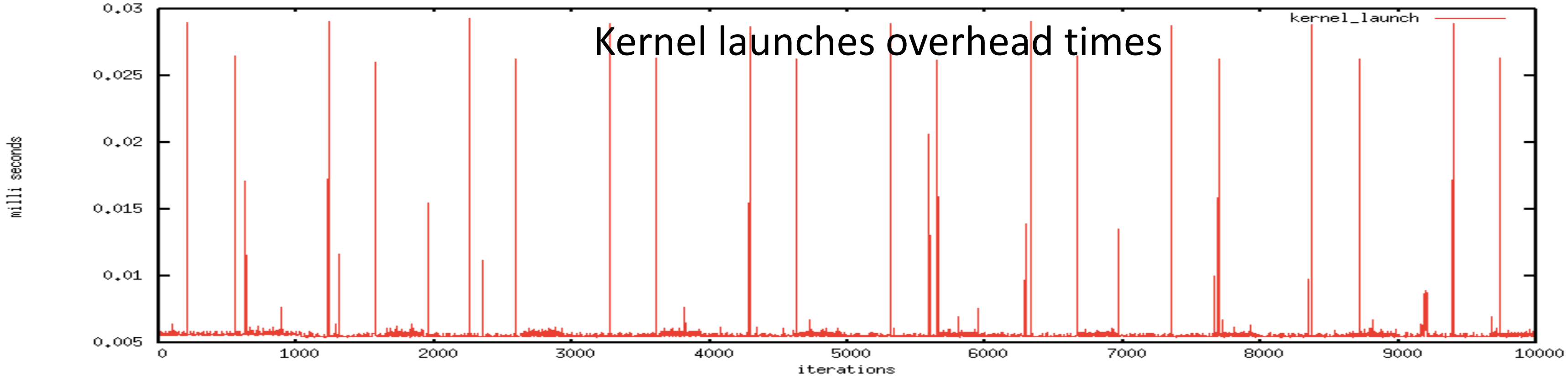
Legacy GPU programming

```
main {
  setup();
  while(run){
    recv(…);
    cudaMemcpy(…, HostToDevice);
    computing_kernel<<<>>>(…);
    cudaMemcpy(…, DeviceToHost);
    send(…);
  }
}
```



Legacy GPU Programming

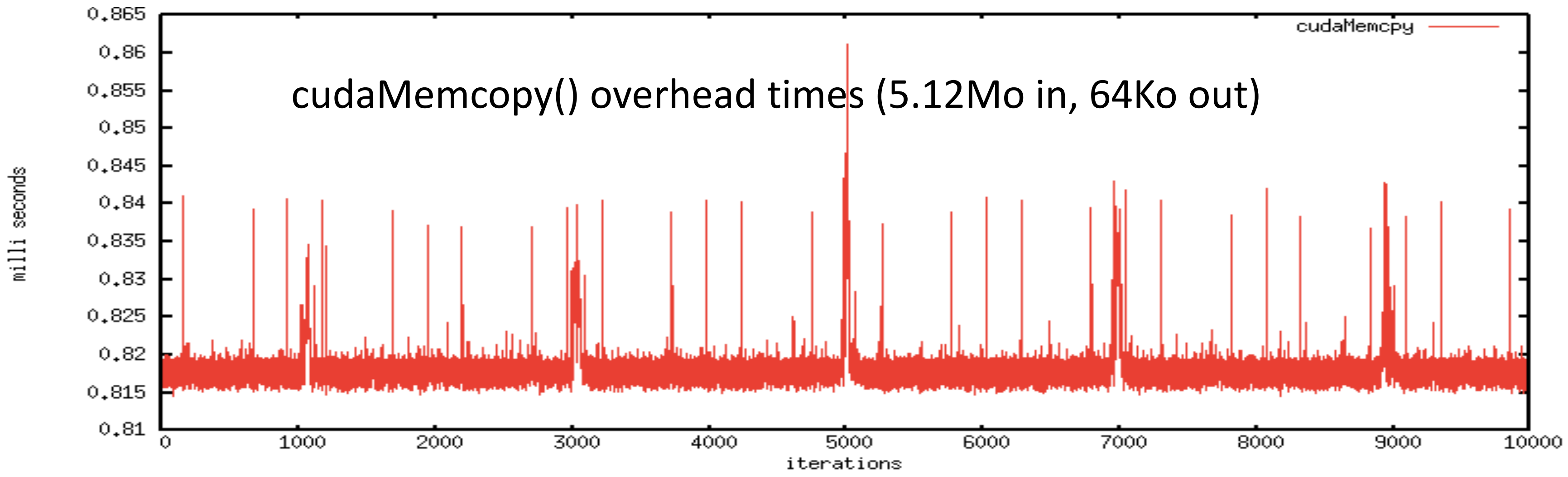


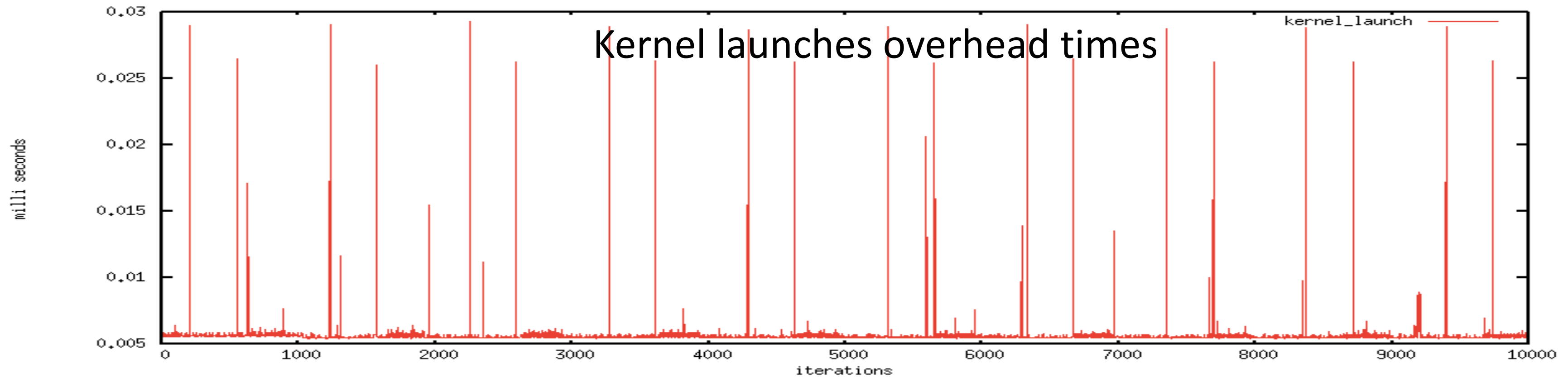


Both cases : jitter of 20 to 30 µsec

Dec 21st 2016

Legacy GPU Programming





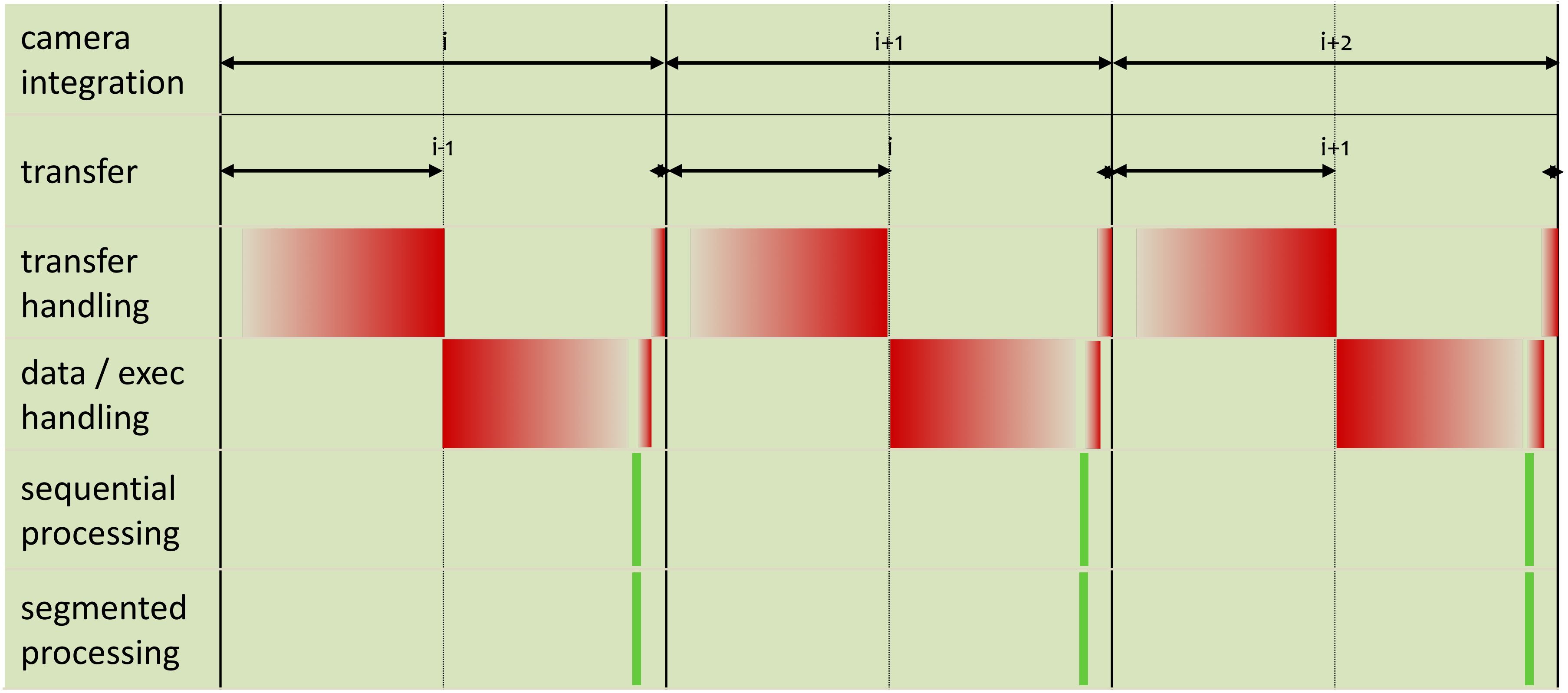
Both cases : jitter of 20 to 30 µsec (40 µsec sometimes)

Dec 21st 2016

Legacy GPU programming

MCAO : image 40.96Mb, commands 512Kb

- Over 40GbE network transfer takes almost 1ms
- Same for cudaMemcpy() operations



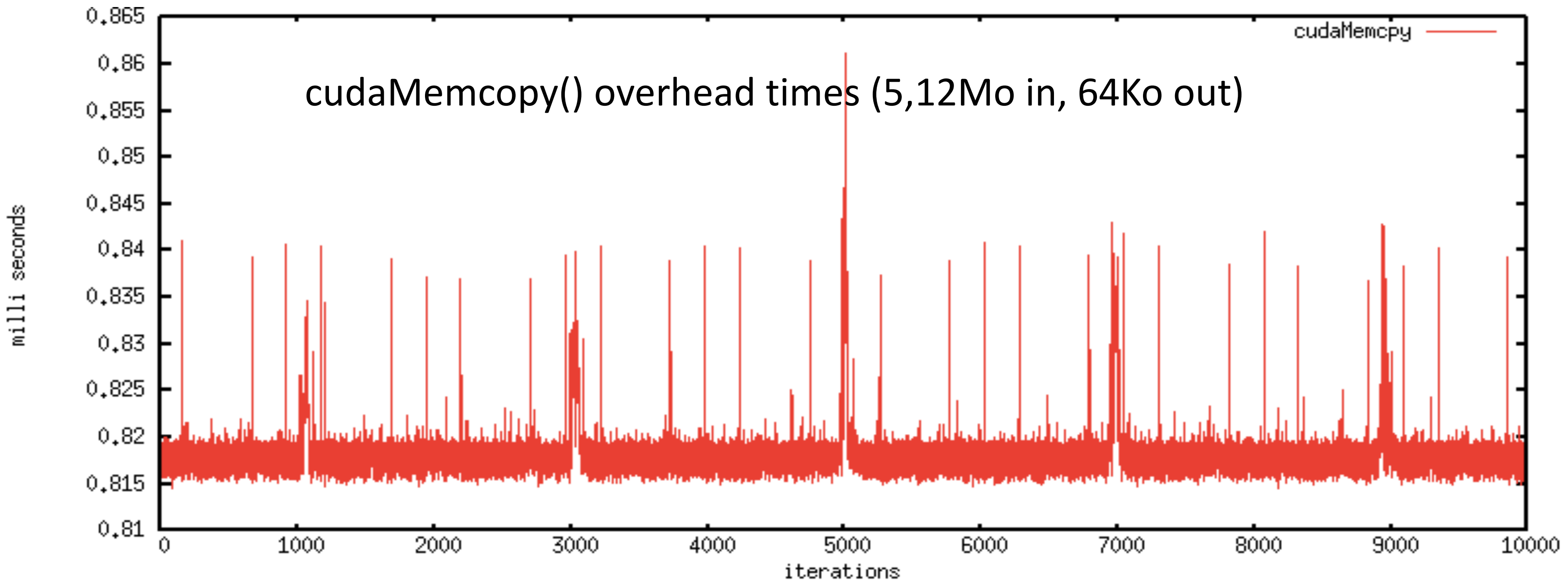
Leaves not enough to no time for computations

GPUDirect + Custom FPGA NIC

Allows third party PCI-e device p2p access

Goals :

Negates latency overhead and reduce jitter induced by cudaMemcpy



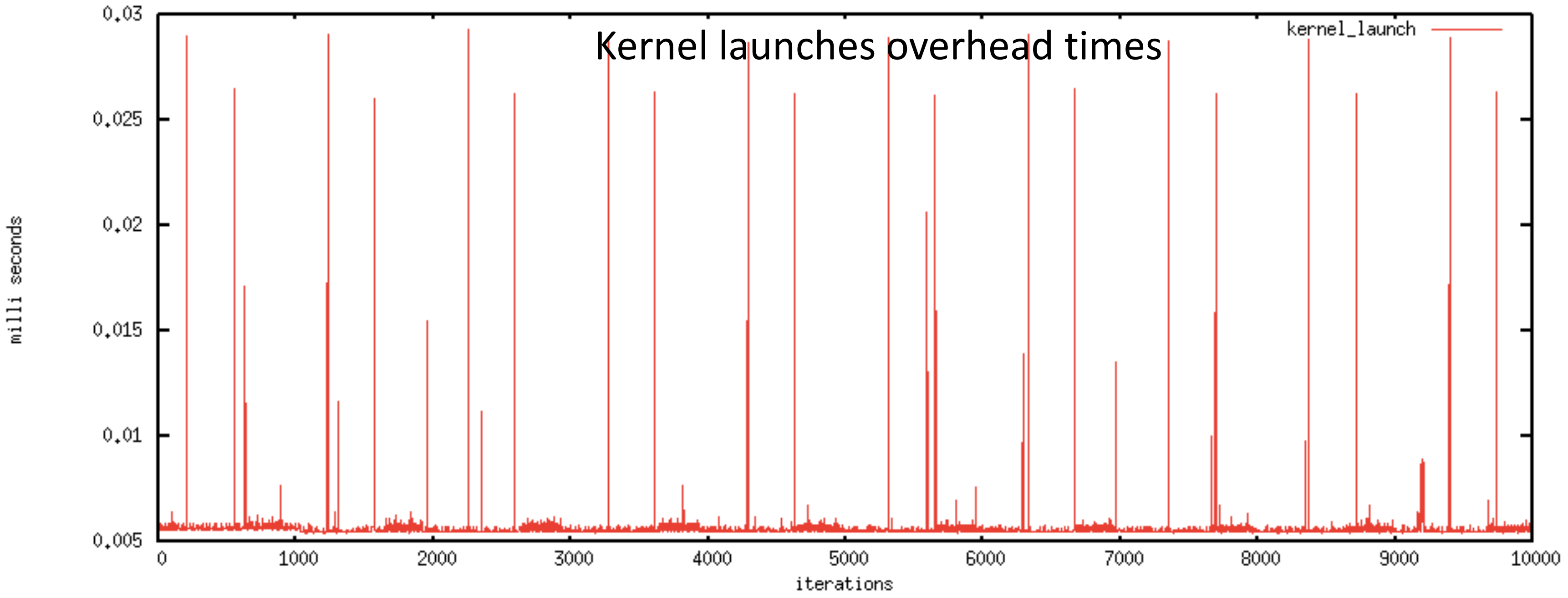
Linux Kernel module: expose CUDA buffers phy@

Persistent CUDA kernel

Exports computation loop on GPU

Goals :

Reduce kernel launch jitter & start computations as soon as data arrive

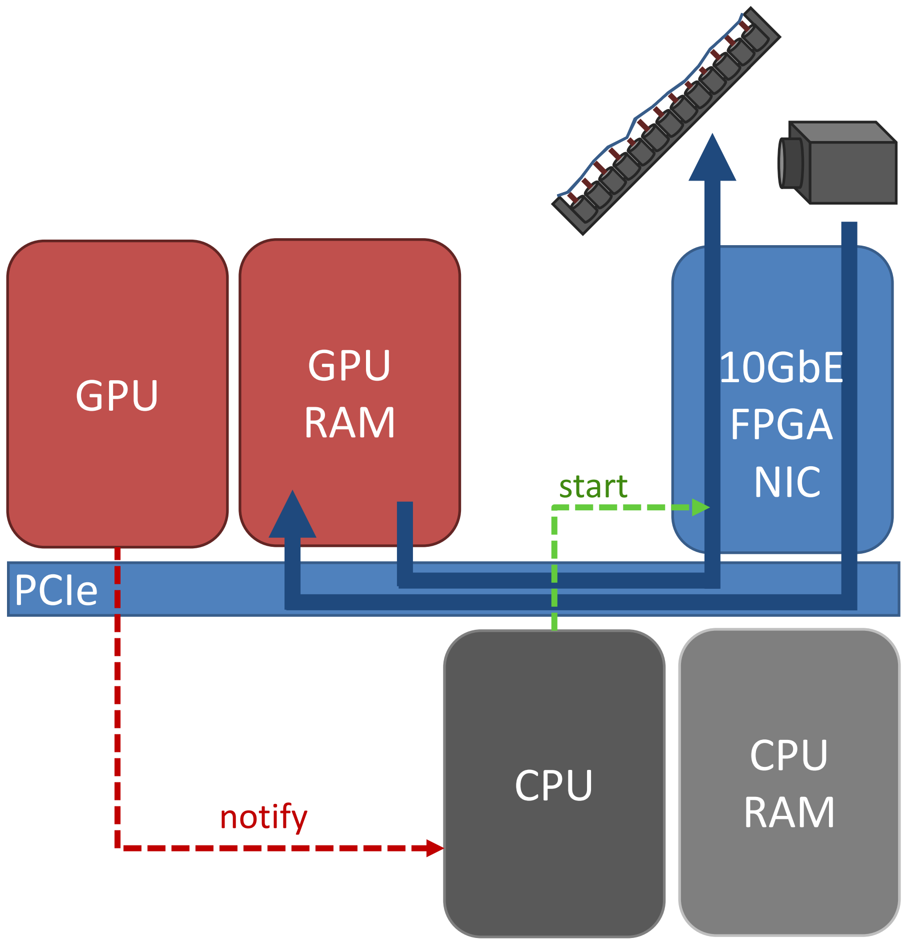


Uses memory polling for data arrival detection

Persistent kernel + GPUDirect

```
main {
  setup();
  persistent_kernel<<<>>>(…);
  while(run){
    waitGPU(…);
    startDMATransfer(…);
  }
}
```

```
persistent_kernel(…){
  while(run){
    pollMemory(…);
    notifyCPU(…);
  }
}
```



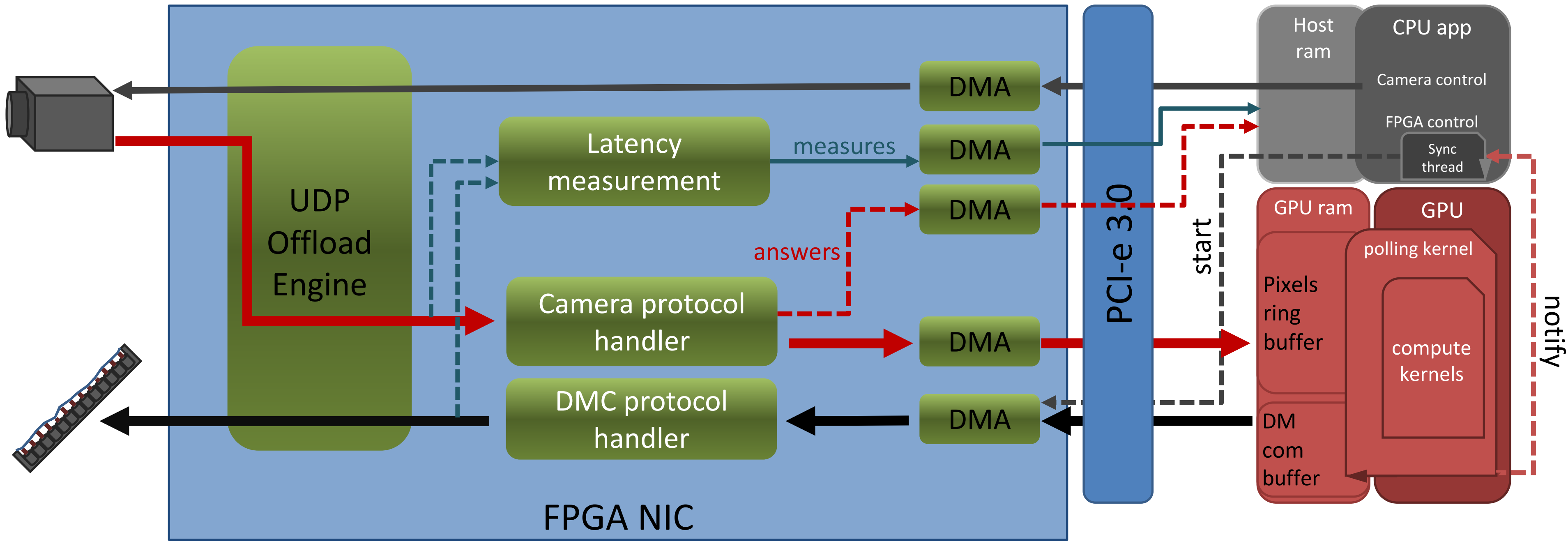
Problem: GPU can't command FPGA

Mapped CPU memory in CUDA for GPU/CPU notification

Persistent kernel + GPUDirect

FPGA writes/reads directly to/from GPU memory

Using only writes would be better though

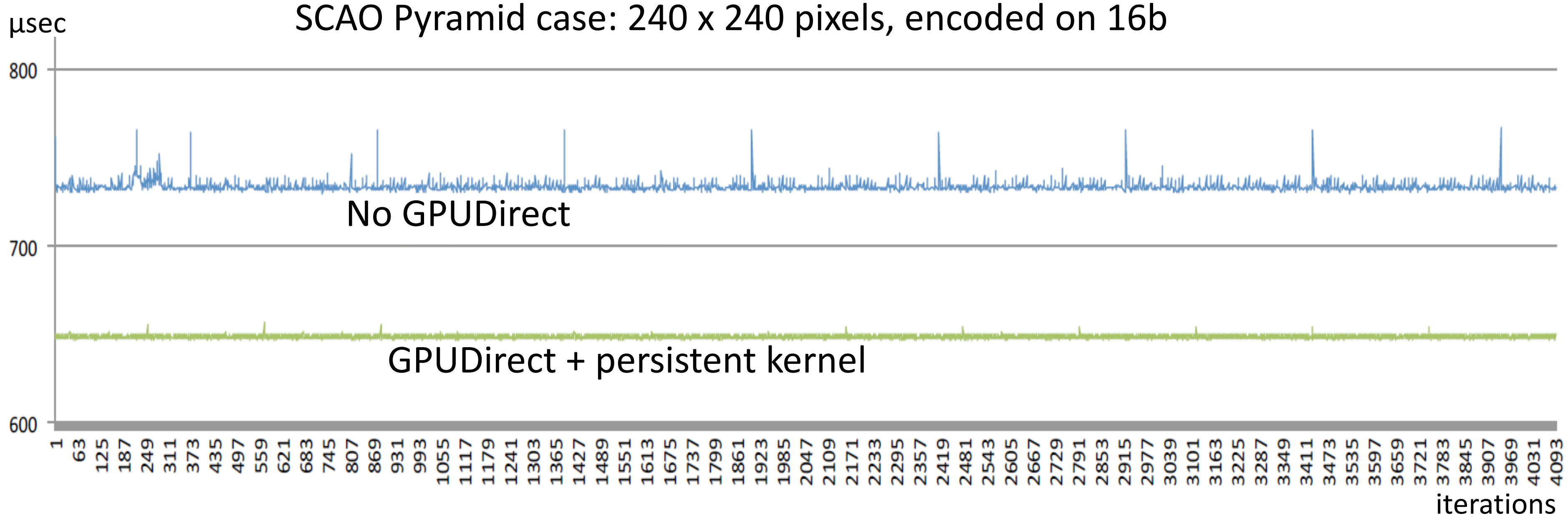


GPUDirect & FPGA NIC - RTT

FPGA PLDA XPressG5
GPU Tesla C2070
OS Debian wheezy

Camera EVT HS-2000M
10GbE network

SCAO Pyramid case: 240 x 240 pixels, encoded on 16b

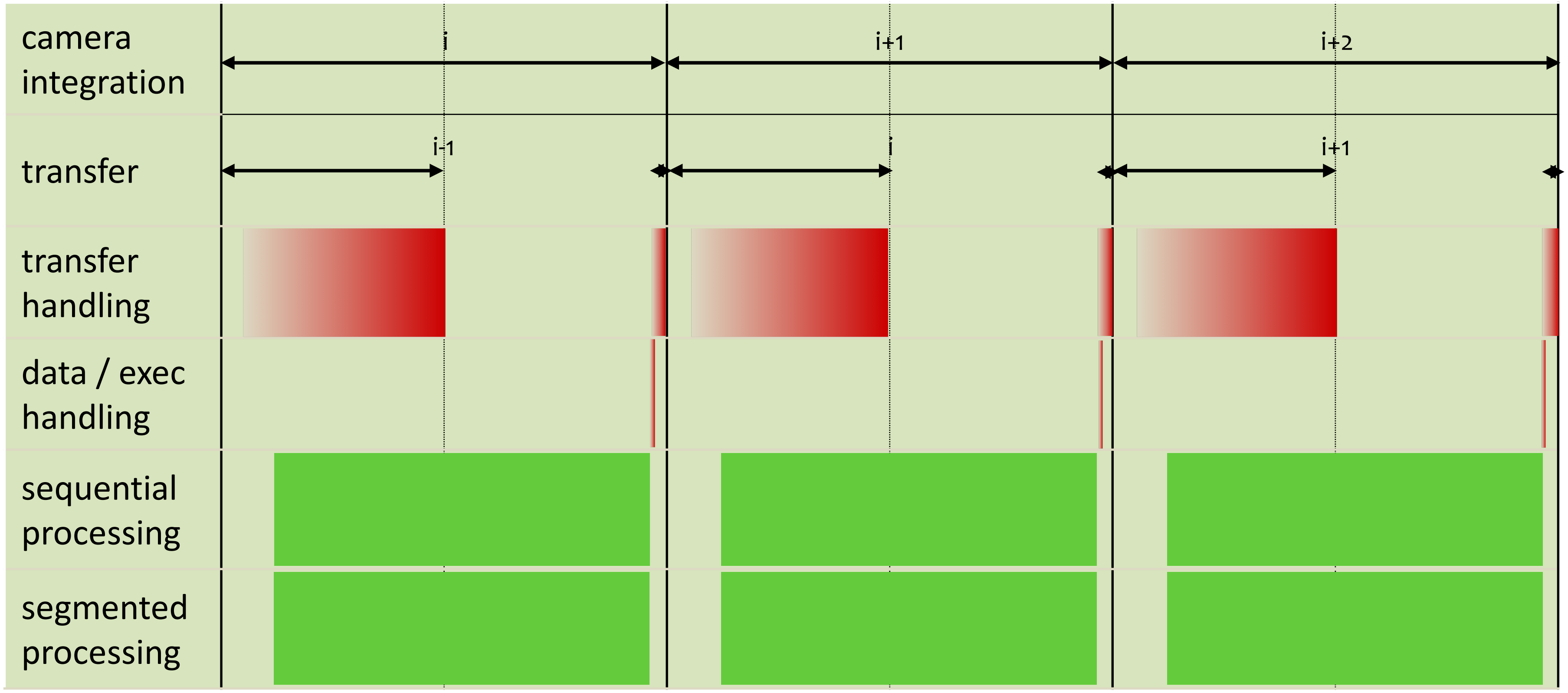


Results are coherent with expectations

Dec 21st 2016

Persistent kernel + GPUDirect

If it scales to MCAO case

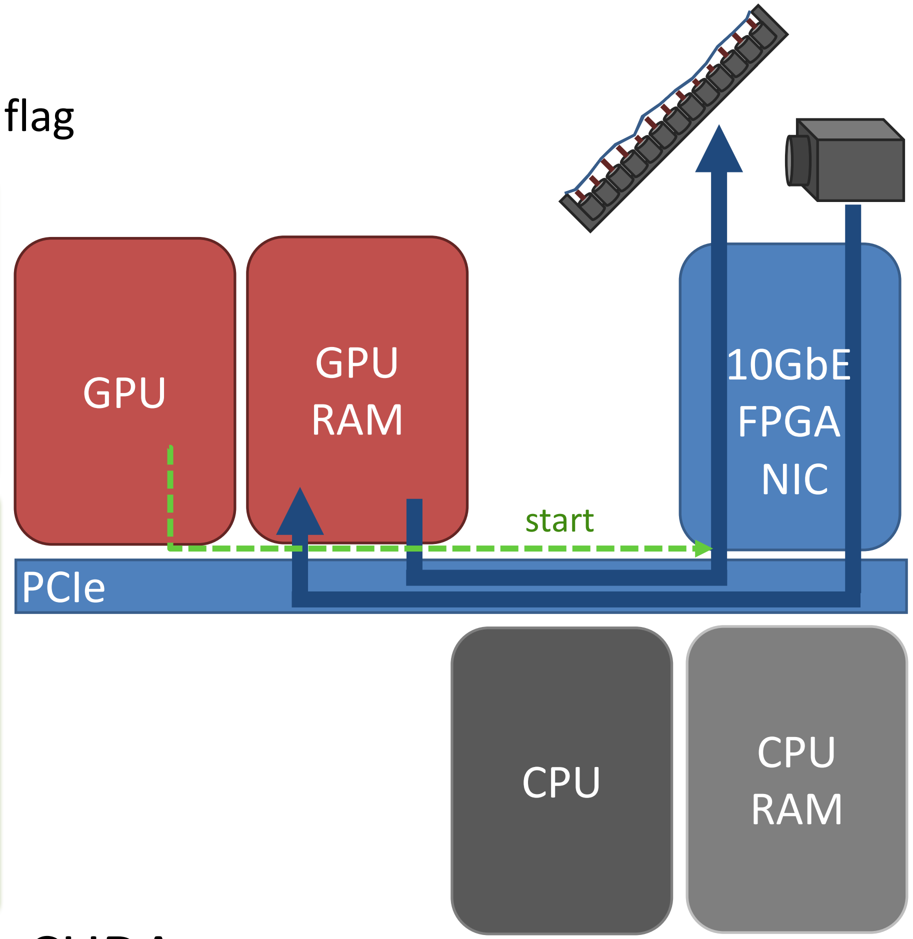


IOMemory mapping (CUDA 7.5)

cuMemHostRegister(…)
using CU_MEMHOSTREGISTER_IOMEMORY flag

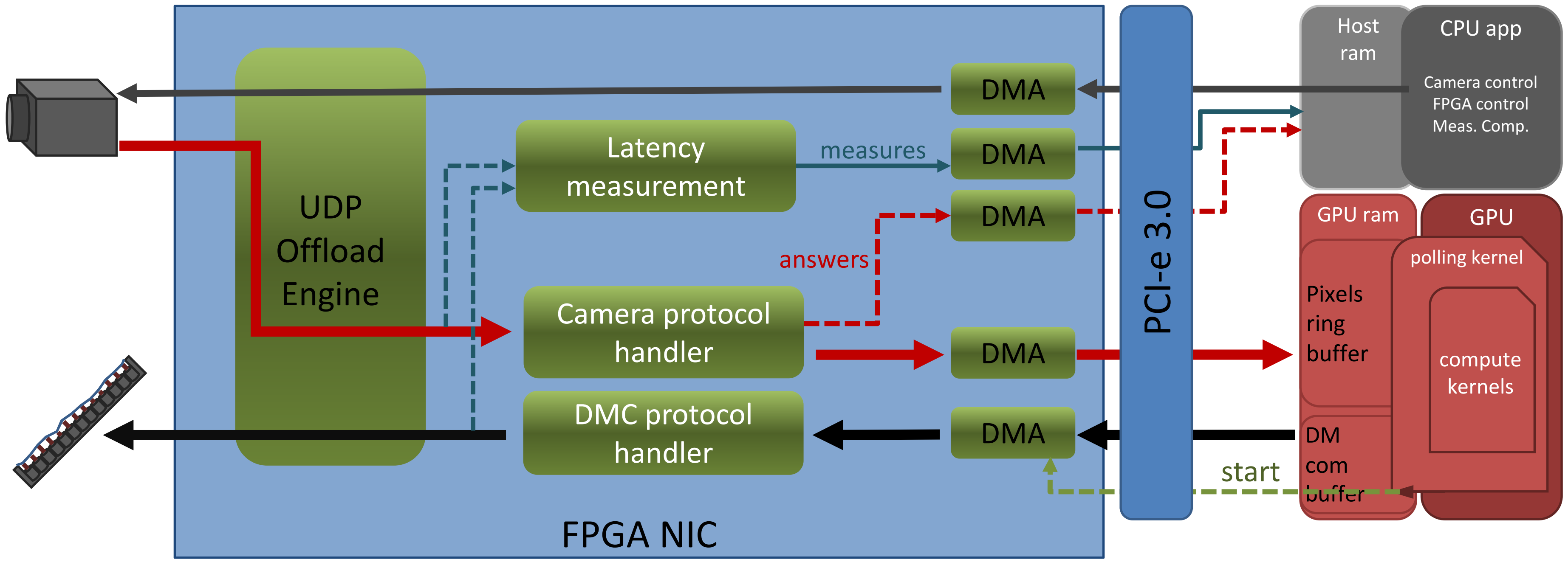
```
main {
  setup();
  persistent_kernel <<<>>>(…);
  …
}
```

```
persistent_kernel(…){
  while(run){
    pollMemory(…);
      actual_computation;
    startDMATransfer(…);
  }
}
```



Mapping FPGA addresses into CUDA memory space allows DMA control from gpu

IOMemory mapping



Little to no improvements, but CPU free for other kind of computations

Conclusion / Perspectives

- Using GPUDirect and a persistent kernel allow efficient data delivery to the Real Time Controller
- GPUDirect approach can be applied to Supervisor using interruptions from FPGA for CPU execution control (kernel launch)

- Simulation setup to benchmark ELT SCAO/MCAO scales thoroughly
- Test with new hardware: PLDA ExpressKUS FPGA and Nvidia K40/80 & P100 GPUs. (and Arria10 FPGAs in a near future)
- Develop computation modules on FPGA to further reduce GPU load
  e.g.: Slopes computation for segmented processing

Thank you for your attention

(Questions?)

Project #671662 funded by European Commission under program H2020-EU.1.2.2 coordinated in H2020-FETHPC-2014