A Real-time Control Computer for the E-ELT

Document: GF-PDR-07

Distributed GPUs for real-time HPC

Version 1.0

20th January 2016

Change Record

Version	Date	Author(s)	Remarks
0.1	11 Jan 2016	D. Gratadour	Initial skeleton version
0.2	11 Jan 2016	J. Bernard	Minor additions
1.0	20 Jan 2016	J. Bernard	Final version

Applicable Documents (AD)

These are the Green Flash PDR documents

No.	Title	Reference	Issue	Date
AD01	Introduction	GF-PDR-01		
AD02	Management plan and WP definition	GF-PDR-02		
AD03	Requirements Specification	GF-PDR-03		
AD04	System Architecture	GF-PDR-04		
AD05	Distributed GPUs for real-time HPC	GF-PDR-05		
AD06	FPGA Solution for hard real-time	GF-PDR-06		
AD07	Smart Interconnect	GF-PDR-07		
AD08	Interface Control Document	GF-PDR-08		
AD09	Supervision Strategy	GF-PDR-09		

Reference Documents (RD)

These are documents external to the Green Flash project

No.	Title	Reference	Issue	Date

Acronyms and abbreviations

Table 1 Acronyms and Abbreviations

AD	Applicable Document
AO	Adaptive Optics
CANARY	Durham/LESIA on-sky AO demonstrator
CPU	Central Processing Unit
CUDA	NVIDIA GPU based software development language
DARC	Durham AO Real-time Controller
DDS	Data Distribution Service
DM	Deformable Mirror
DRAGON	Durham laboratory-based AO demonstrator bench
ELT	Extremely Large Telescope
E-ELT	European ELT
ESO	European Souther Observatory
FPGA	Field Programmable Gate Array
GPU	Graphics Processing Unit
GUI	Graphical User Interface
HLS	High Level Synthesis
HPC	High Performance Computing
MIC	Many Integrated Core
MVM	Matrix-Vector Multiplication
NIC	Network Interface Controller
PCIe	Peripheral Component Interconnect express
RD	Reference Document
RTC	Real-Time Control
RTL	Register Transfer Level
SIMD	Single Instruction Multiple Data
SPARTA	ESO VLT AO Real-time Control System
SHERE	VLT Planet finder instrument
UDP	User Datagram Protocol
UK ATC	United Kingdom Astronomical Technology Centre
VLT	Very Large Telescope
WFS	Wave-Front Sensor
WP	Work Package

Table of Contents

1 Scope

This document specifies the data-pipeline and the hardware sizing in Green FLASH.

In the first time, we will explain the tasks to achieve and the performance level we have to reach to respect the specifications.

We will cover the computation, the bandwidth and the transfer section and approximate as accurate as possible the best sizing for our solution.

2 Computation

In this section, we will describe how to compute a command vector from a WFS and present our time and dimension constraints.

2.1 Process

The goal to reach consists to compute the command vector for the DM from the WFS frame.

It works in three successive steps :

1. Pixel calibration (dark subtraction, flat field, background subtraction)
2. Slopes computation (including pupil registering with a bi-cubic interpolation and reference slopes subtraction)
3. Commands vector computation
    1. Calculate the pseudo-open loop measurement vector the two last command vector and the interaction matrix.

       $\vec{M}_{ol}[k]=\vec{M}[k]-D(a\vec{c}[k-2]+(1-a)\vec{c}[k-1])$
    2. Then, calculate the raw tomographic vector

       $\vec{e}[k]=R\vec{M}_{o1}[k]$
    3. Finally, get the vector command by smoothing the raw vector with the last command

       $\vec{c}[k]=g\vec{e}+(1-g)\vec{c}[k-1]$

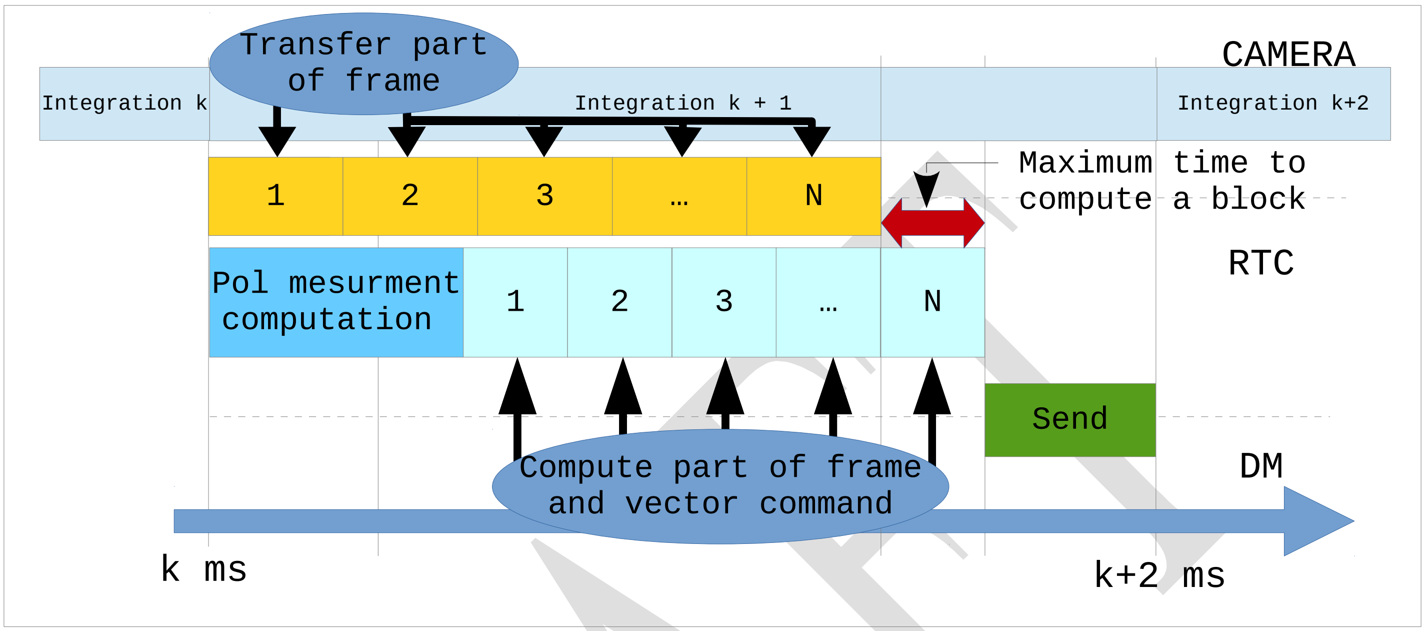
During the next chapters, we will focus on the last point (Commands vector computation) as it takes most of the computation time

2.2 Performance

The integrating time of the camera is fixed at 2ms. All the other components are synchronized at this frequency. But transfer time also impose some constraints such as the time transferring a frame to the RTC.

Luckily, a part of the time during the frame transfer will be used by calculating the last wave front which makes the transfer delay constraint much smaller.

We use the remaining time by computing block per block the frame and vector command during the transfer. So the time between the arrival of the last block and sending the command need to be enough to compute the block.



2.3 Dimension

For our case, all the dimensions and data movement are fixed.

Data	dimension	Representation	Size	Movement frequency	Throughput
Frame	1.6k x 1.6k	Unsigned int 16	5.12 MB	Each iteration	21 Gb/s
R, D matrix	80k x 15k	float 32	5 GB	Each minute	1.3 Gb/s
Command vector	15k	float 32	60KB	Each iteration	240 Mb/s

All the data are updated at different frequencies during the runtime. The frame throughput is by far the most highest due to its hight frequencies and must be applied for the six different cameras.

The two other types of data are less complex to transfer due to low frequency or lower quantity of data.

2.4 Transfer

We have to handle the last frame for each 6 cameras, representing 6 x 5.12 MB (30.72 MB) to transfer at every iteration. It is by far the biggest transfer performed between RTC and other systems. The others two only need ~100MB of bandwidth and are either distributed on many nodes (R, D matrix to all nodes) or only send the one system (command vector to the DM controller).

The frame transfer will clearly size the communication between node and others systems.

2.5 Execution

At each iteration, the main task consist to perform 2 different vector-matrix multiplications (with R and D matrix). We can approach the FLOP requirement by count 1 multiplier–accumulator (MAC) or 2 FLOP per matrix's element. So, for the each 2 matrix we have :

$$2\,\text{matrix}\,x\,80{,}000\,x\,15{,}000 \approx 2.5\,GMAC = 5\,GFLOP$$

per iteration and $2.5\,TFLOPs$ at 500Hz.

3 Reference

3.1 Hardware Reference

3.1.1 Compute performance

Many accelerators exist but GPUs are particularly suitable to the task with their high throughput and high bandwidth. We will consider other hardwares in another time.

Model	Architecture	Theoretical peak performance single precision (TFLOPS)	Memory bandwidth (Gb/s)	ECC	Energy consumption (W)
NVidia Tesla K40	Kepler	4.29	2304	Yes	235
NVidia Tesla K80	Kepler	8.5	3840	Yes	300
NVidia M6000	Maxwell	6.07	2539	No	222

For this case of computation,(io bound problems) the bandwidth are the first performance indicator ahead the computing performances. A simple way to calculate the true performance peak and the number of GPU needed is to use this simple equation :

$$CEIL\left(\frac{CR}{\tilde{B}\,x\,OI}\right) = N$$

CR (Gflops) : Computation requirement or the total amount of computation needed for the task : ~5000 GFlops

$\tilde{B}$ (GB/s) : Sustained peak bandwidth, calculate from the theoretical bandwidth and a constant. This constant is experimental and may vary with the architecture or with features set like ECC (Error-Correcting Code).

	K20C	K40	K80	M6000
B	1664	2304	1920 (x2)	2539
$\widetilde{B}_{ECC\,OFF}$	1400 (84%)	1888 (82%)	1536 (x2, 80%)	1968 (77%)
$\widetilde{B}_{ECC\,ON}$	1200 (72%)	1664 (72%)	1360 (x2, 70%)	NONE

Until the end of the document, we will use the values with ECC memory. The other value was just use for example and for crossing the estimation with experimental values.

OI or Operation Intensity : Ratio of the number of FLOPs executed to the number of bytes read or written to the memory (quantity of non cached memory access), to be relevant, we need to care of the size of element (floating point 16, 32 or 64 bits).

N : Number of GPU needed to execute the task.

With 32 bits floating point number :

	K20C	K40	K80	M600
$N_{ECC\,OFF}$	29	22	14	21
$N_{ECC\,ON}$	34	25	15	NONE

With 16 bits floating point number :

	K20C	K40	K80	M600
$N_{ECC\,OFF}$	15	11	7	11
$N_{ECC\,ON}$	17	13	8	NONE

3.1.2 Interconnect technologies

3.1.2.1 Pci-e

The GPUs are all connected with a pci-e gen 3 bus.

	Frequency	Bandwidth	Flux per line	Flux per 16 lines
PCI-E 3.x	100 MHz	8 GT/s	984.6 MB/s	126 Gb/s

The pci-e bus can provide up to 126 Gb/s for each GPU but this value depend mostly by the copy engine of the GPU.

For example with K80 GPUs, we can reach ~80 Gb/s of peak bandwidth.

3.1.2.2 10 Gigabit Ethernet

The 10 Gb Ethernet link can provide 1.25 GB/s bandwidth and 40 Gb/s with a 4 links configuration. The real bandwidth depends on the protocol used. In practice, with optimized configuration, we almost reach the theoretical bandwidth.

3.2 Compute Reference

3.2.1 KBLAS

KBLAS and Magma are two Basic Linear Algebra Subprogram (BLAS) Library. They implement on different type of hardware linear algebra operations.

Currently, KBLAS achieve the better performance for the matrix vector multiply on multi-GPUs topology. It can be a good indicator for the performance we can achieve and verify our expectation on other hardwares.

We compare the theoretical result with two experimental tests :

- First was performed by KAUST on 8 K20c GPUs cluster, ECC off and 32 bits floating point number.
- The other, in our cluster with 4 k80 GPUs in the same condition.

The KAUST's cluster reach 670 GFLOPs of peak performance. Using the previous equation, we obtain 1340 Gb/s of peak bandwidth per GPU.

Our cluster reach 815 GFLOPs of peak performance that gives us 3248 Gb/s (2 x 1624 Gb/s).

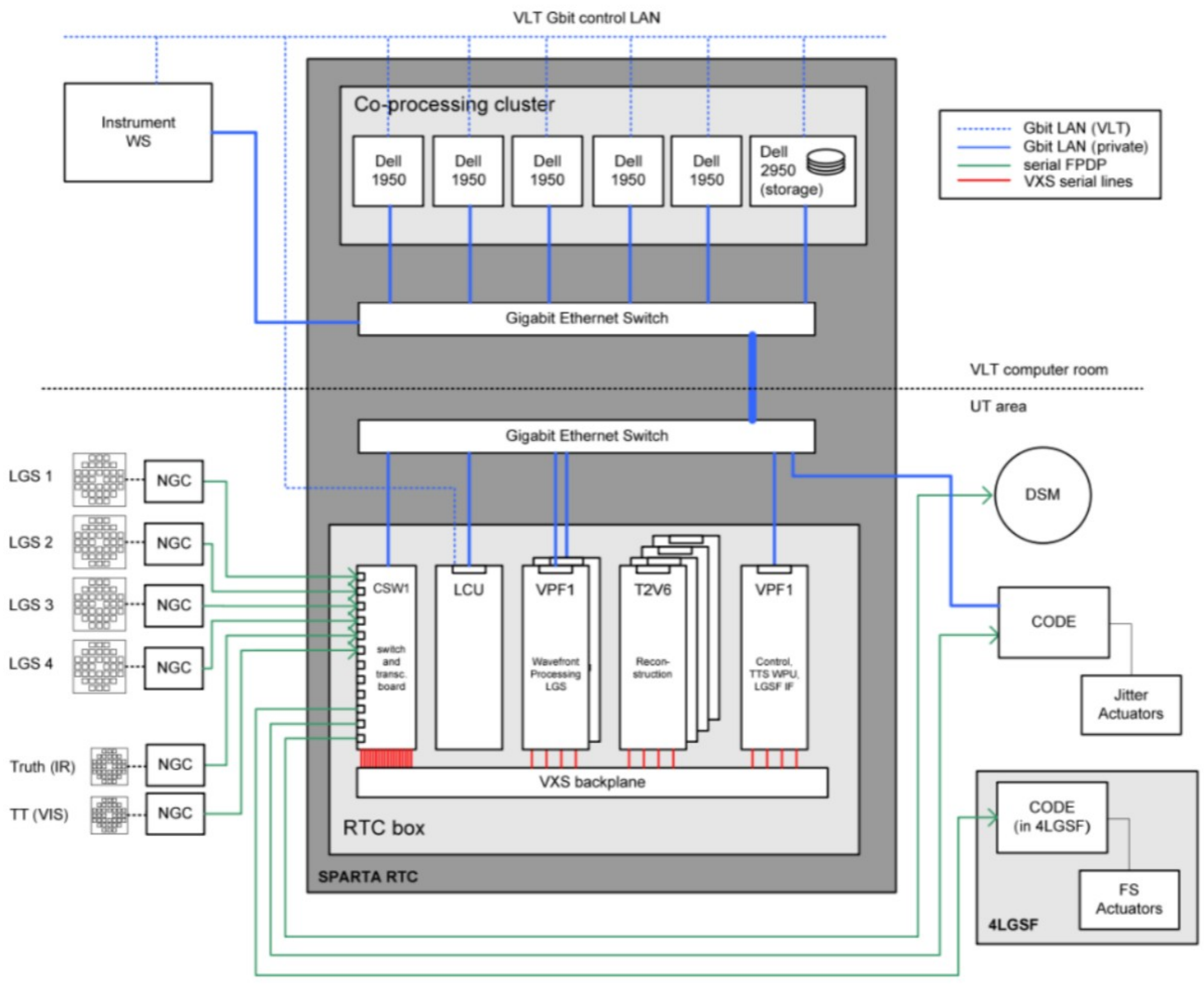
These two results show that the equation is quite close to experimental value and provide a good way to approximate the total quantity of GPUs for the RTC.

3.3 Solution reference

3.3.1 Sparta

SPARTA is an existing solution to provide a standard platform to build adaptive optics real-time computers based on the requirements. It was use on different project like SHERE or AOF.

Instrument	AO class	WFS	# meas.	DM	# com.	Freq. (Hz)	Performance (GMAC/s)
SPHERE	XAO	1	2.6k	1	1.3k	1500	5.2
AOF	LTAO	4	2.4k	1	1.2k	1000	11.8



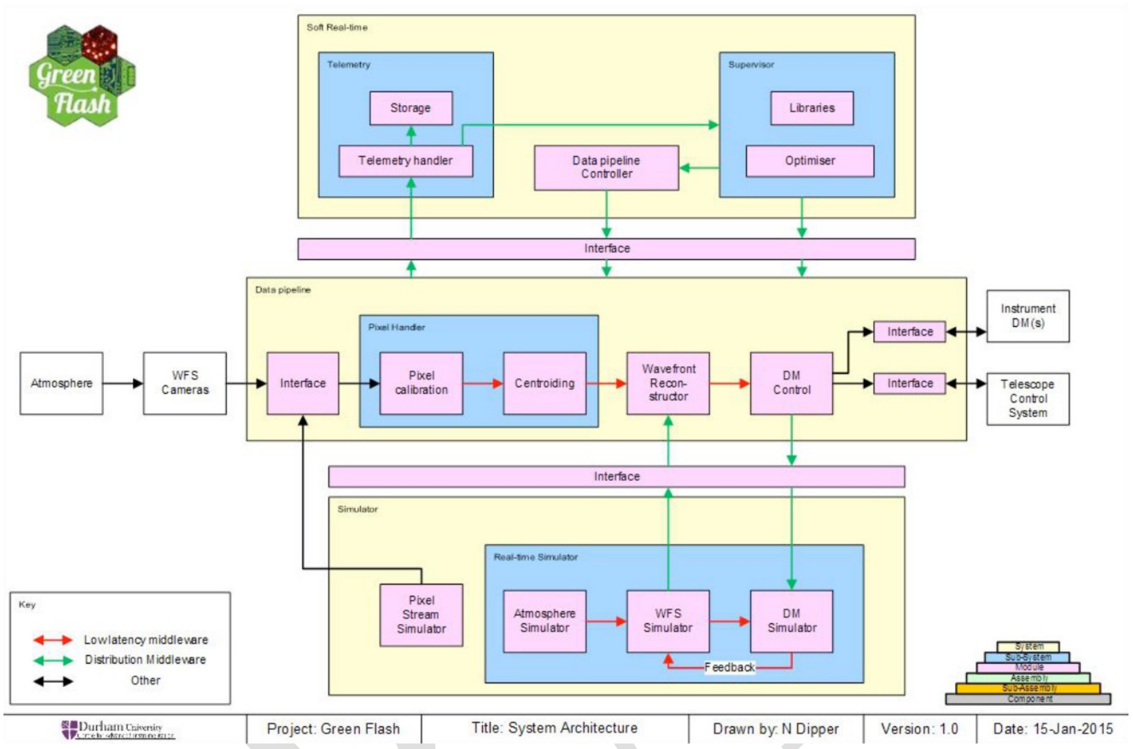
The SPARTA architecture works with different stacks. We will only focus on the RTC box which implement a hard real-time low latency adaptive optics control loop.

The computations are handled with both DSPs and CPUs. They have both benefits and drawback.

- DSP are good for floating point operations which required high bandwidth, DSP can be programmed with C language and have a much faster cycle development than FPGA-based
- CPU are used for high level algorithm that required more complex hardware
- The SPARTA system is a good architecture but can't provide enough bandwidth (VXS backplane is limited at 6.25Gb/s) and compute performance (it is designed for much lower project than the E-ELT).

4 Solution

The Green Flash project starts with a new architecture. It's based on existing strategies (SPARTA) but use new features and hardwares.



The Green Flash RTC box solution is a node oriented architecture. Each node is an independent system with some GPUs and a host controller to manage the GPUs. Each WFS is linked with the RTC box by four links 10 Gibabit Ethernet.

4.1 General architecture

4.1.1 Compute parallelization

In order to fit the computation in all the GPUs, we have to parallelize the data respecting some constraints.

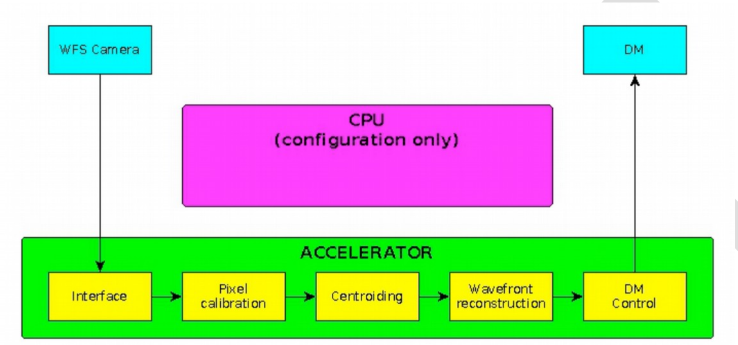
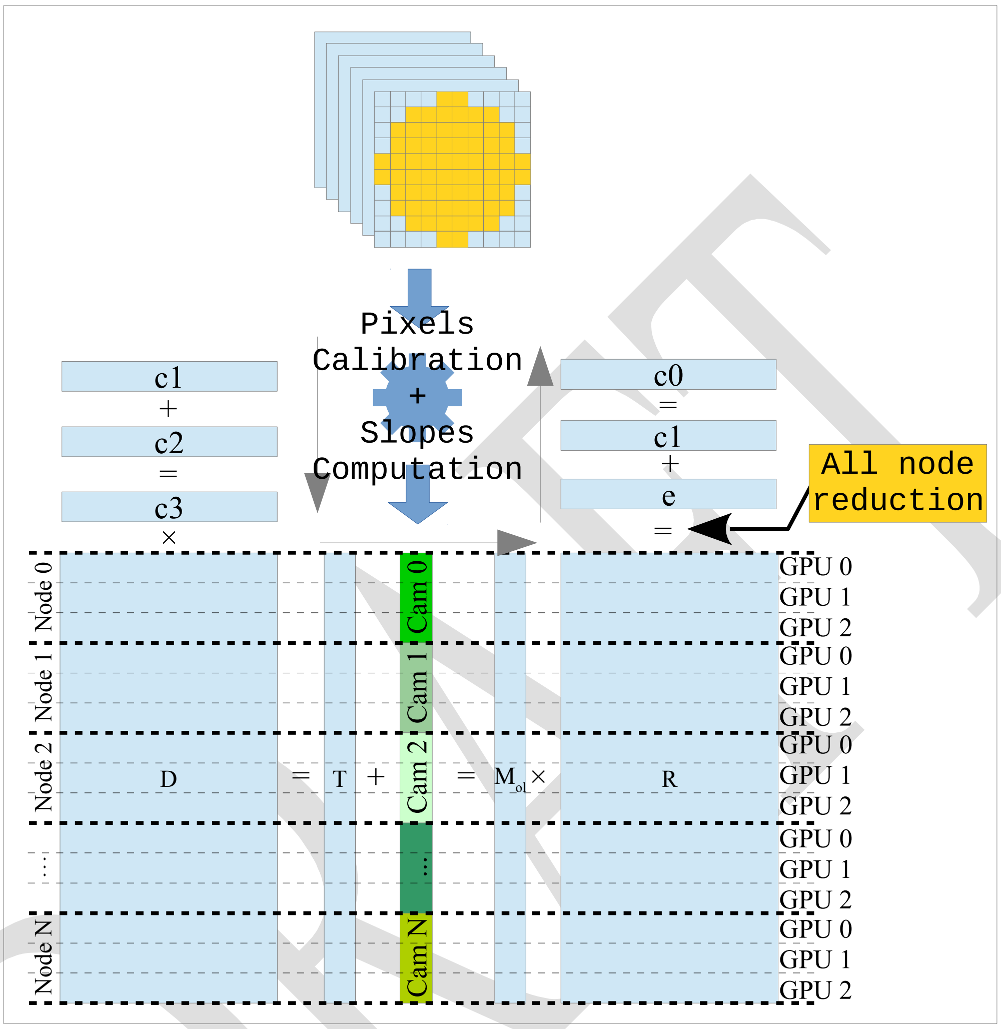


Figure 8: A fully embedded pipeline: the data by-passes the CPU completely.

This first constraint comes from the frame repartition. A node receive at each iteration one frame from one camera. A GPU can only efficiently access the frame of its node. This constraint will lead the rest of the computation.



In order to minimize the communication between nodes, all the data interacting with the frame have to be computed by the same node.

- c1 + c2 = c3

With our strategy, each GPU compute the vector addition. All vectors have 15k elements. The addition can be handled in one iteration.

- c3 x D = T

All GPUs of all nodes have the c3 vector, in this case we just have to segment the D matrix in order to let each GPU computes the part of T vector corresponding to its camera and slopes.

The matrix is perfectly segmented and no data is redundant.

- T + slopes = M_{ol} and M_{ol} x R = e

These calculations are interlaced due to the time needed to obtain the entire frame. But almost all computations remain independent except for the vectors reductions introducing a node communication. One node need to have e to compute the final command but all nodes need it to for the next iteration (c0 becomes c1 at next iteration).

In the worst case communication, 5 nodes send the vector to the last node. This communication use pic-e bus and four links 10 Gb Ethernet but only the 10 Gb Ethernet is a bottleneck. With the theoretical bandwidth, we can transfer all data in 60µs or 6% of the iteration time.

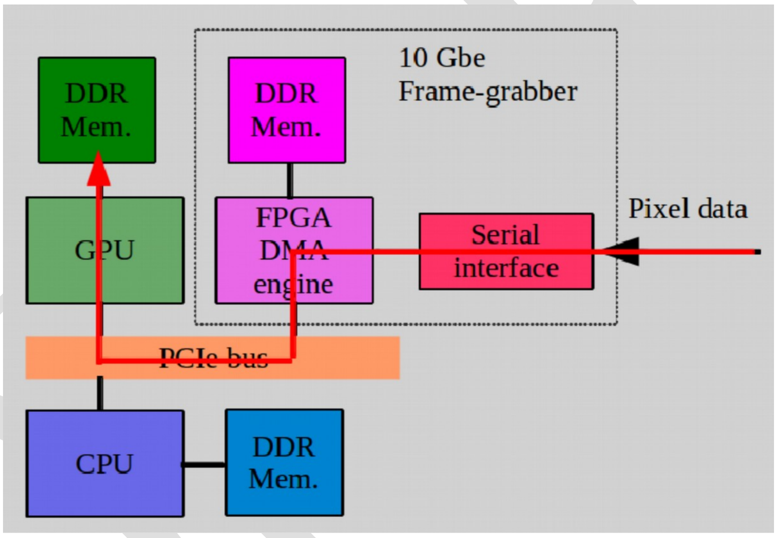
Finally the node with the result has to transfer the command to the DM and all nodes in order to start the next iteration.

4.2 Hardware architecture

In this section we will simply mention the hardware architecture to specify inter-GPU and inter-node bandwidth.

All GPUs on the same node can directly interact using the pci-e bus. The transfer limit depends on the pci-e bandwidth but mostly on the GPU copy engine (~80Gb/s).

Between GPU on different nodes, we will talk about node communication. This communication use a FPGA with a serial interface using four links 10 Gb Ethernet with a theoretical bandwidth of 40 Gb/s.



4.3 Implementation consideration

4.3.1 Precision

For this type of processing, the controllers use mostly floating point numbers encoded in 32 bits (IEEE 754). The accuracy achieved by this encoding allows to control a deformable mirror with values remaining near 0.
One solution under consideration is to encode the numbers with 16 bits. The benefits are numerous: reduction of the required space and bandwidth by 2. On current architectures, the calculation is always performed with 32 bit numbers.

With our type of problem, where the bandwidth is by far the most important constraint, use 16 bits number in place of 32 bit number can reduce by 2 the number of GPUs needed.

Precision	Exponent	Fraction	Min positive	Max positive	Precision near 0

(bit)	(bit)	(bit)	value	value	
32	8	23	1.18×10^{-38}	1.7×10^{38}	1.4×10^{-45}
16	5	10	6.10×10^{-5}	65504	5.96×10^{-8}

This strategy need more test in order to know if the precision is enough.

4.3.2 Determinism

With a hard real time constraint, we need determinism in order to ensure a 500Hz frequency. In a first time, with existent GPU architecture, we need to remove most of the jitter in the compute pipeline. [1]

The main jitter in CUDA come from the kernel scheduler using the CPU and introducing many CPU-GPU communications. But one time a block of a kernel is launched on the GPU, no one can stop it until it finishes its work.

A simple solution consists in launching a perpetual kernel which never stop until a command was send. However the solution introduces a huge constraint in the implementation, it does not allow the use of any existing library. Added to this is the necessity of using the correct number of blocks and threads to use 100% of the GPU. All blocks or threads additionally created beyond the GPU limits will never launched introducing a bad computation.

Another point is about concurrent accesses. We need to discard most of them and introduce deterministic way to perform access. This is handled at the algorithmic level.

5 Future technologies

5.1 NVIDIA Pascal GPU

At this time, the most powerful GPU is the Nvidia K80 that provides the best bandwidth by pci-e port and provide all the professional features (ECC, 2 copy engines).

During the current year, NVIDIA will produce a new generation of GPU based on the Pascal architecture.

This architecture introduces new features like 16 floating point numbers processing unit and a new type of random-access memory that hugely increase the bandwidth. Also, this generation will double the number of transistor resulting in a potential good improvement of compute capacity.

Model	Architecture	Memory bandwidth (Gb/s)	Memory (GB)	ECC	Fp 16 support
NVidia Tesla K80	Kepler	3840 (2 x 1920)	24 (2 x 11)	Yes	No

1 http://proceedings.spiedigitallibrary.org/proceeding.aspx?articleid=1891156

NVidia Pascal	Pascal	8000	32	Yes	Yes

With what is known at this time, it is hard to calculate the exact need of GPUs based on Pascal architecture (unknown sustained peak bandwidth for Pascal) but we approximate the number at 8 for computations based on 32 bit floating point number and only 4 for 16 bit floating point number computations.

6 Implementation plan

This work is mainly concentrated in WP4 of Green FLASH (see AD02 for a full list of WP), in task 4.1.

Objectives This WP aims at assessing solutions relying on hardware accelerators in a distributed memory configuration to address both the RT-box and the supervisor module designs of the AO RTC. The performance of these solutions, based on GPUs, Intel MIC and FPGA will evaluated in terms of determinism for the real-box performance and overall throughput for the supervision process. This WP has strong ties with WP5 on smart interconnects since in such distributed configuration, the achievable performance strongly depends on the ability to enable low latency intranodes communications. The output of this WP is a series of small scale prototypes relying on these various accelerator technologies on which both RT-box and supervisor strategies will be evaluated. This performance assessment will be used for the down selection of technologies during the final design review of the AO RTC prototype.

6.1 Task 4.1: Distributed GPUs for real-time HPC

Objectives. In this task we propose to investigate the level of performance determinism and the scalability of GPU based clusters targeting the E-ELT first light AO RT box specification. This work relies on previous developments at OdP and UoD on the evaluation of this technology for the AO real-time control application in a non-distributed configuration, as well as an operational validation on the telescope.

The main result of these developments, consistently with other studies on the GPU application to real-time tasks, is the necessity of implementing a strategy in which the interaction between the GPU and the host CPU should be minimized (avoided if possible) to ensure low performance jitter. Moreover, efficient communication strategies must be implemented to ensure minimal overall latency.

We propose to extend this study to a distributed configuration and follow an energy efficient approach, since the role of the host CPU should be minimal, by implementing, as a baseline, a small scale cluster of up to 8 nodes each hosting an ARM64 processor, several (depending on available main boards) GPUs and a smart NIC. The NIC will be obtained from WP6 and optimized communication strategies relying on its smart features will be studied in this task.

The performance will be assessed on tailored MVM cases with relevant data stream sizes, and emphasis will be put on the scheduling process, either through a custom approach developed in this task or relying on mainstream solutions selected in task 8.3. These two approaches will be compared. Our goal is to reach 1.5 TMAC/s on the MVM application by the end of the prototyping phase of the project, with 240 Gb/s of streaming data as an input. The market evolutions will be monitored during the course of this task, to get performance projections for the final architecture design, with emphasis on the availabilities of various ARM64 main board options and more powerful GPUs and the consistency with the performance objective. This work includes:

- the HW architecture definition following available main boards and GPUs on the market at the time of the project start
- the implementation of a prototype of 8 nodes with standard and smart interconnects (for data acquisition from the simulated sensors) in relation with WP6
- the deployment of a standard RT environment with mainstream libraries to perform the RT-box tasks in relation with task 8.3
- the implementation of a custom scheduling strategy on the nodes, build on top of the smart interconnect strategy to optimize performance
- the performance assessment with respect to the scale of the facility in terms of determinism and energy consumption