A GPU based RTC for E-ELT Adaptive optics : Real Time Controller prototype

Bernard J.[a], Lainée M.[a], Perret D.[b], Sevin A.[a], and Gratadour D.[a]

[a]LESIA - Observatoire de Paris/ Université Paris Diderot

ABSTRACT

Most of the control laws used in AO require one or several matrix-vector multiplies, at frequencies around 1 kHz. With the Extremely Large Telescope (ELT) generation, the real-time control of the AO loop becomes one of the most challenging issues due to the high computational power required (large matrices and high frequency) and the limited energy footprint inherent to the telescope location (based in isolated regions). The Green Flash European project is in line with this challenge and aims at building a prototype for a Real-Time Controller (RTC) at the ELT scale. We propose a GPU-based solution because of their energy efficiency and throughput capabilities. In order to meet the requirements of an AO loop, in terms of jitter and throughput, we chose a very low level approach relying on persistent kernels to handle all the computational steps from pixels calibration to slopes and command vectors computation. This approach simplifies the latency management by reducing the communication overheads but led us to re-implement an entire AO control loop relying on some GPUs standard features : communication mechanisms (guard, peer-to-peer), algorithms (generalized matrix-vector multiplication, reduce/all reduce) and new synchronization mechanisms on a multi node-multi GPUs system. In this paper we detail the context and the data pipeline of the RTC and report the performance (time, jitter) we obtained and the scalability of the solutions on the NVidia DGX-1 platform.

Keywords: Adaptive Optics, High Performance Computing, Real time, Graphic Processing Unit

1. INTRODUCTION

SPARTA,[1] ESO Standard Platform for Adaptive optics Real Time Applications, has been used to drive a variety of AO systems, ranging from single conjugate AO systems with less than 100 actuators to much larger XAO systems like SPHERE and is used today with the multi-Laser AO Facility (AOF). SPARTA was developed as a response to the need for a standard platform, serving all AO projects foreseen for the VLT instrumentation, by using common components and commercial hardware.

The final design can be represented as an hybrid cluster hosting CPUs (PPC), FPGAs and DSP modules. FPGAs are used for input/output processing and operations which are largely integer based and highly parallel (wavefront processing, i.e. centroiding from sensors images), DSP modules are used to perform floating point operations and operations which require significant memory (wavefront reconstruction, i.e. MVM) and CPUs are used for operations which require significant algorithmic complexity and whose definition can evolve with time. Additionally, a co-processing cluster is coupled to the RTC box, as a supervisor module receiving telemetry data and providing regular updates to the controller. This is a non real-time module interacting with the RT box which is based on a commodity cluster solution.

To reach the delivered performance for the AOF[2] (almost 12 GMAC/s), the RT box includes 4 hybrid FPGA+DSP boards (each hosting 8 DSP modules) and 3 hybrid PPC + FPGA boards. For this dimensioning, the co-processing cluster is composed of 5 nodes each hosting two quad-core x86 CPUs. However, as stated by the SPARTA architects: *the E-ELT with its instruments poses new challenges in terms of cost and computational complexity. Simply scaling the current SPARTA implementation to the size of E-ELT AO system would be unnecessary expensive and in some cases not even feasible.*[3]

this work is sponsored through a grant from project 671662, a.k.a. Green Flash, funded by European Commission under program H2020-EU.1.2.2 coordinated in H2020-FETHPC-2014

Following the recent evolution of the HPC market, switching to a new strategy for the RT box relying on custom arithmetic logic processors such as GPUs as a replacement for DSP boards is an appealing option. This commodity hardware can potentially provide the compute power required at a limited cost and with a relative energy efficiency. Moreover, comprehensive and optimized programming models have appeared, such as CUDA, OpenMP or OpenCL, with support for mainstream mathematical libraries, which fits well with the requirement for an expandable solution based on standards. Finally, the vendors road map is robust and proceeding at a fast pace providing regular architecture and performance enhancements, with for instance the recent development of optimized tools to leverage the compute power of multiple GPUs in shared memory configurations.

In this paper, we review the challenges of designing a full scale RT box prototype based on GPUs, in the context of the Green flash project.[4] We give an overview of the different hardware systems and how we drive then to address the AO constraints.

2. REAL-TIME PIPELINE

The role of the real-time data pipeline is to compute the command vector for the DM from the WFS data. It proceeds in three successive steps (outlined in figure 1):

- Pixel calibration (dark subtraction, flat field, background subtraction)
- Slopes computation (including pupil registering with a bi-linear interpolation, gradients computation (or center or gravity computation for SH WFS) and reference slopes subtraction)
- Commands vector computation

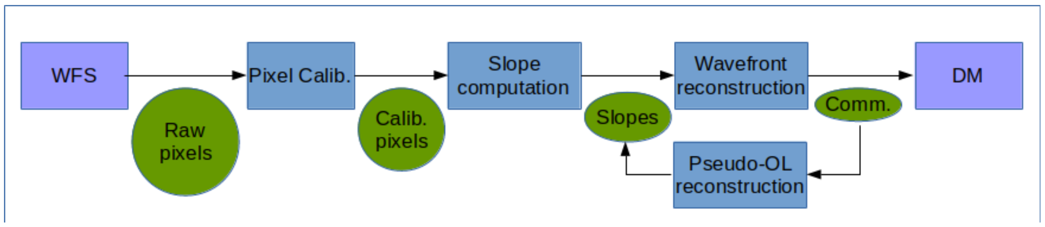

Figure 1: Typical real-time data pipeline, with compute sub-modules in blue, and main data in green.

In the following, we will focus on the last point (Commands vector computation) as it dominates the computation time.

Most of the systems on-sky today rely on a linear control law, involving a measurement vector and a control matrix to produce a command vector. On top of this baseline, predictive control or tomography require additional real-time processing of the produced data. For instance, in the case of a tomographic AO system,[5] a typical scheme is to compute a wavefront reconstructor based on a Minimum Variance approach and use pseudo-open loop data (POL), i.e. data reconstructed from the last command vector. In this POL Control scheme, two matrix vector multiplies of equivalent size are required to compute the wavefront at each iteration.

The main following steps will be considered in the next sections: First, compute the pseudo-open loop measurement vector the two last command vector and the interaction matrix D:

$$\vec{M}_{ol}[k] = \vec{M}[k] - D\left(a\vec{c}[k-2] + (1-a)\vec{c}[k-1]\right) \tag{1}$$

Then, compute the raw tomographic vector, using the reconstructor matrix R

$$\vec{e}[k] = R\vec{M}_{ol}[k] \tag{2}$$

Finally, get the vector command by smoothing the raw vector with the last command :

$$\vec{c}[k] = g\vec{e}[k] + (1-g)\vec{c}[k-1] \tag{3}$$

This data processing pipeline can be implemented using 2 different approach :

- Synchronous execution: in this case, the processing pipeline starts only when the entire frame is transferred.
- Asynchronous execution: in this case, data can be processed by blocks. The pipeline starts when it receives a certain part of the frame, during which the pseudo-open loop measurements are computed. Then, it performs slopes computations and the MVM with the available slopes. It stores the partial result and waits for the next part of the frame. This strategy is more efficient when the fame is large and requires significant time to be uploaded in the real-time data pipeline.

These two execution scheme are outlined in Figure 2.

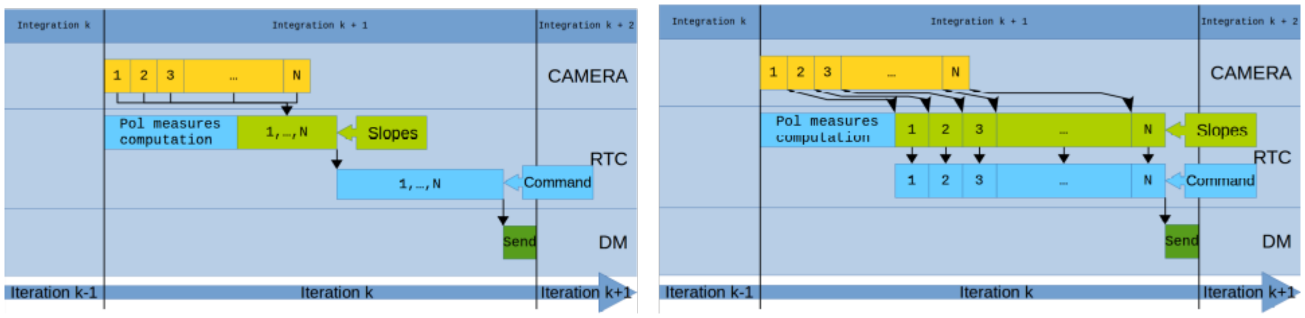

Figure 2: Synchronous (left) and asynchronous (right) execution pipeline.

Considering these two execution schemes, the matrix-vector multiply (MVM) algorithm can be implemented following two approaches described in the following diagram.

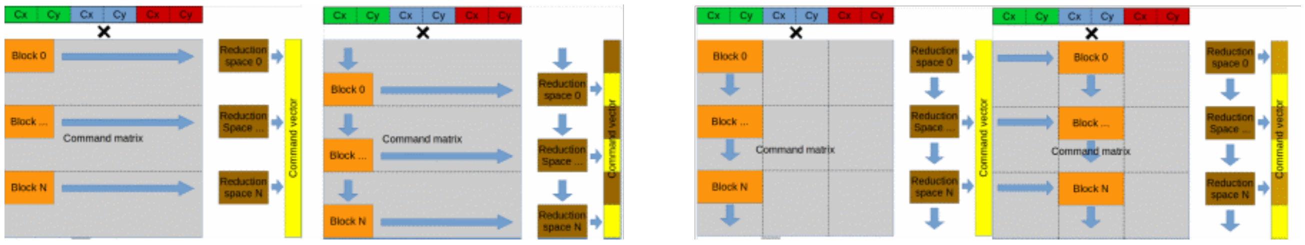

Figure 3: Synchronous (left) and asynchronous (right) MVM computation.

The synchronous MVM computation consists in fully accumulating values on a temporary local memory and then exporting the result to the command vector. In this case, only one reduction is computed for each command vector component.

The asynchronous MVM strategy will do the same operations but limited to a group of slopes. In order to limit the quantity of local memory, the reduction is done for every chunk of available slopes. The main benefit of this strategy is that the MVM computation can be done during the frame transfer.

Several tests were performed and it appears that at this time, the first strategy (synchronous execution with sequential Slopes/MVM processing) is much faster using 95% of the sustained peak bandwidth of the GPUs where the asynchronous execution only uses 44% of the sustained peak bandwidth (this result is due to the high quantity of reduction, it is being optimized). These results show that our implementation is also consistent with reference solutions like KBLAS.

3. SYSTEM DIMENSIONING

In this study, we have considered two different scales of systems for the E-ELT: Multi-conjugate AO with 6 WFS and 3 DMs and Single Conjugate AO with a single WFS and a single DM. The main system parameters are outlined in the table below. The numerical precision on the camera pixels is taken to be *unint16* and the numerical precision on the MVM computation is taken to be *float32*. For the SCAO case, we have considered to kinds of system dimensioning, using either a Shack Hartmann WFS (SCAO1) or a pyramid WFS (SCAO2).

Table 1: System dimensioning

Name	Nwfs	Frame size	matrix size	mem. footprint	frame rate	throughput	MVM FLOP/s
MCAO	6	6x 800x800	77kx15k	39.1Gb	500 FPS	6x5 Gb/s	2.4 TFLOP/s
SCAO1	1	800x800	12kx5k	2.2Gb	1kFPS	10.2 Gb/s	136 GFLOP/s
SCAO2	1	240x240	12kx5k	2.2Gb	1kFPS	1 Gb/s	136 GFLOP/s

At each iteration, the execution profile is dominated by matrix-vector multiplies. The FLOP requirement as been derived by accounting for 1 multiplyaccumulate (MAC) or 2 FLOP per matrix's element. The cost of performing the centroiding is not taken into account here. For MCAO: two MVM $76kx15k$ leads to 2.4 GMAC/s or 4.8 GFLOP/s. At 500 FPS this gives 2.4 TFLOP/s. For SCAO: a single MVM $12kx5k$ leads to 68 MMAC/s or 136 MFLOP/s. At 1k FPS this gives 136 GFLOP/s

In the MCAO case, we consider a framerate of 500 FPS and asynchronous execution. A part of the frame transfer is used to recover the POL data, as needed by the pseudo-open loop scheme described above, which makes the transfer delay constraint much smaller. The remaining time can be used to compute block per block the current WF measurements from the WFS frame and the corresponding vector command during the transfer. So the time between the arrival of the last block from the WFS frame and the time at which the command is sent is the main dimensioning factor for the real-time pipeline.

For the SCAO2 case, considering a pyramid WFS, the framerate goes up to 1k FPS and the frame transfer can't be overlapped with computation. However, the time needed for the transfer is more than 40 times smaller. The lower time transfer allows us to compute the WFS measurement in a first step and proceed with the matrix vector multiply in a second step.

4. HARDWARE SPECIFICATIONS

Different kinds of hardware accelerators are available on the market but GPUs are particularly suitable for these tasks considering their high throughput and high memory bandwidth. We have tested our pipeline on several generation of GPUs to study the scalability over the architecture evolution. The specifications of these GPUs are outlined in the table below.

Table 2: GPU specifications

Name	Architecture	Peak perf (float32)	mem. bandwidth	ECC	Power (W)
Tesla P100	Pascal	9.3 TFLOP/s	5760 Gb/s	Yes	250
Tesla K80	Kepler	8.5 TFLOP/s	3840 Gb/s	Yes	300
Quadro M6000	Maxwell	6.1 TFLOP/s	2539 Gb/s	No	222
Tesla K40	Kepler	4.3 TFLOP/s	2304 Gb/s	Yes	235

MVM is a I/O bound algorithm and for this kind of computation the memory bandwidth is the main dimensioning factor more than the computing performance (estimated from the number of computing cores and their clock rate). A simple way to compute the true performance peak and the number n of GPU needed is to use this simple equation :

$$N = \frac{Req}{\tilde{B}/8 \times OI} \tag{4}$$

in which Req is the compute requirement expressed in GFLOP/s, $\tilde{B}$ is the measured sustained memory bandwidth and OI the operation intensity, i.e. the ratio between the number of FLOPs and the number of non cached memory accesses. For a MVM using 32 bit arithmetic, this ratio is 0.5 (and 1 for 16 bit).

For each of these GPU technologies, we have measured the sustained memory bandwidth using the unitary tests provided with the CUDA toolkit from NVIDIA. The obtained results with and without ECC are reported in the table below.

Table 3: Measured GPU performance

Name	Architecture	Peak BW	sustained BW	sustained BW (ECC)	N	N (ECC)
Tesla P100	Pascal	5760 Gb/s	3945 Gb/s	3945 Gb/s	8	10
Tesla K80	Kepler	3840 Gb/s	3200 (83%)	2760 (72%)	12	15
Quadro M6000	Maxwell	2539 Gb/s	1968 Gb/s (77%)	N/A	20	N/A
Tesla K40	Kepler	2304 Gb/s	1888 (82%)	1664 (72%)	21	23

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 (close loop systems work close to 0). One solution under consideration is to encode the numbers with 16 bits. The benefits are significant: reduction of the required space and bandwidth by 2. On current architectures, the calculation is however always performed with 32 bit numbers. The following table summaries the precision that can be obtained using both these encoding.

Table 4: Precision summary

Precision (bit)	Exponent (bit)	Fraction (bit)	Min positive value	Max positive value	precision near 0
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}

With our problem dimensioning, where the bandwidth is by far the most important constraint, using 16 bit numbers in place of 32 bit number can reduce by 2 the number of GPUs needed. This will be further investigated in a following paper.

5. PROPOSED ARCHITECTURE

With a hard real-time constraint, we need time determinism in order to ensure a stable loop running at framerate in the range of thounsands of FPS. It is commonly assumed that a maximum jitter of 10% on the overall latency budget is acceptable.

As a first estimate, with existing GPU architectures, the main jitter source in CUDA applications comes from the kernel scheduler using the CPU and introducing many CPU-GPU communications. Another significant source of jitter comes from copying the data to and from the GPU using a CPU process. This is well demonstrated in the figure below.

But once a block of threads is launched on the GPU, using only on-board memory, it cannot be stopped until it finishes its work. A simple solution consists in launching a persistent kernel which never stops until a command is sent through a memory polling mechanism. This approach is described in Figure 5. On the left, a typical pipeline involving synchronization between processes on the CPU and on the GPU and on the right, the persistent kernel approach in which the CPU is mostly IDLE.

However the solution introduces a huge constraint in the implementation, it does not allow the use of any existing standard library. Added to this is the necessity to use the correct number of blocks and threads to use

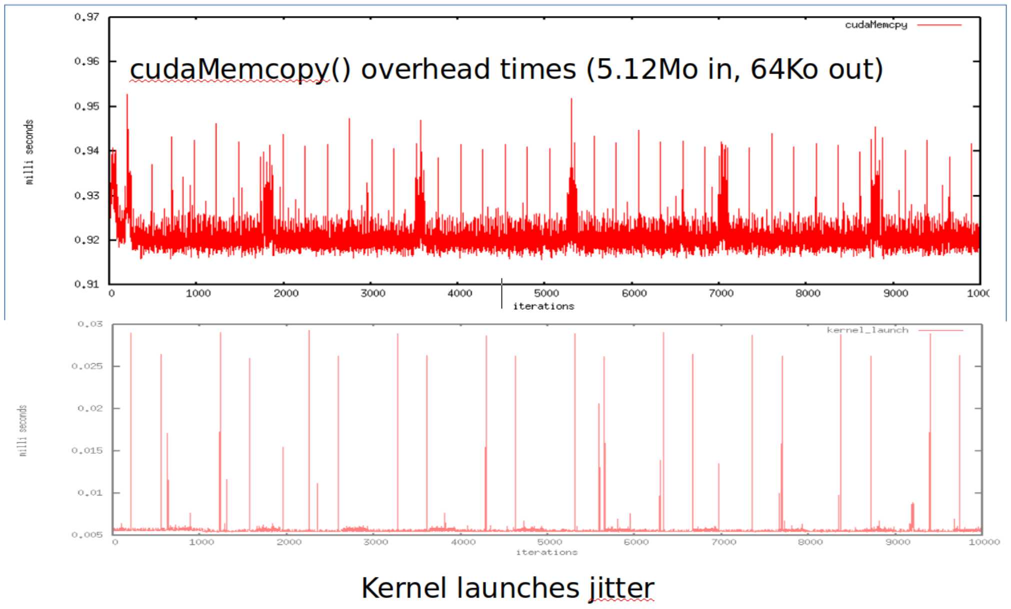

Figure 4: Typical measured execution times when copying data to the GPU sing the standard memcopy API provided in CUDA and typical jitter obtained in launching a CUDA kernel from a CPU process.

100% of the GPU capacities in terms of maximum number of blocks and threads per block. All blocks or threads additionally created beyond the GPU limits will never be launched introducing computation errors. Another critical point is concurrent accesses. We need to discard most of them and introduce a deterministic way to perform access. This is handled at the algorithmic level.

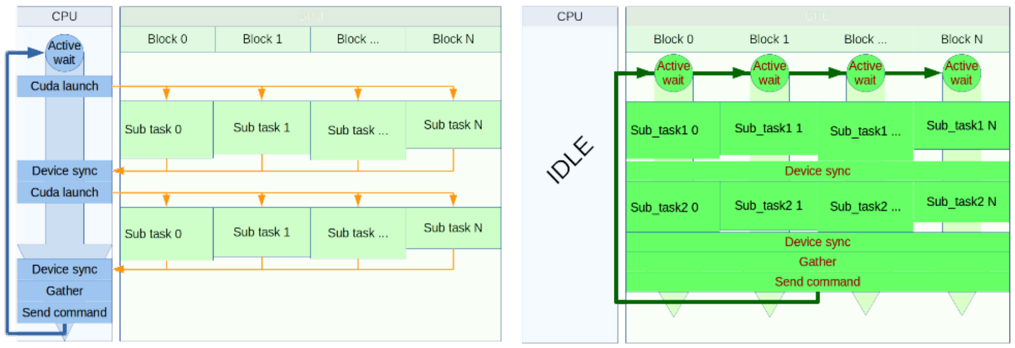

Figure 5: Classical approach for a GPU pipeline (left) compared to the persistent kernel approach (right).

In order to avoid any CPU process to interfere with the RT pipeline on the GPU, it is also mandatory to implement direct memory access between the frame grabber and the GPU and to consider the full pipeline as a stream of data from the WFS camera to the DM. This is depicted in Figure 6. In a classical approach (left), data from the WFS is transferred to the host main memory involving a CPU process, then copied to the GPU memory involving another CPU process. On the data processing side, several kernel launches and synchronizations with a CPU process can also occur.

When direct memory access and persitent kernels are implemented concurrently (right), the CPU remains IDLE as data are exchanged directly through the PCIe bus between the frame grabbing device and the processors.

In the generic node architecture we propose, the node consists in low latency interconnects and compute

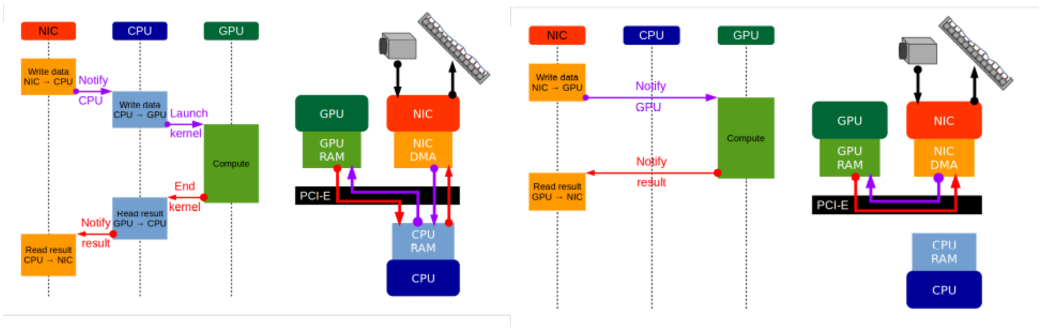

Figure 6: Data transfer to the GPU, with (right) and without (left) direct memory access.

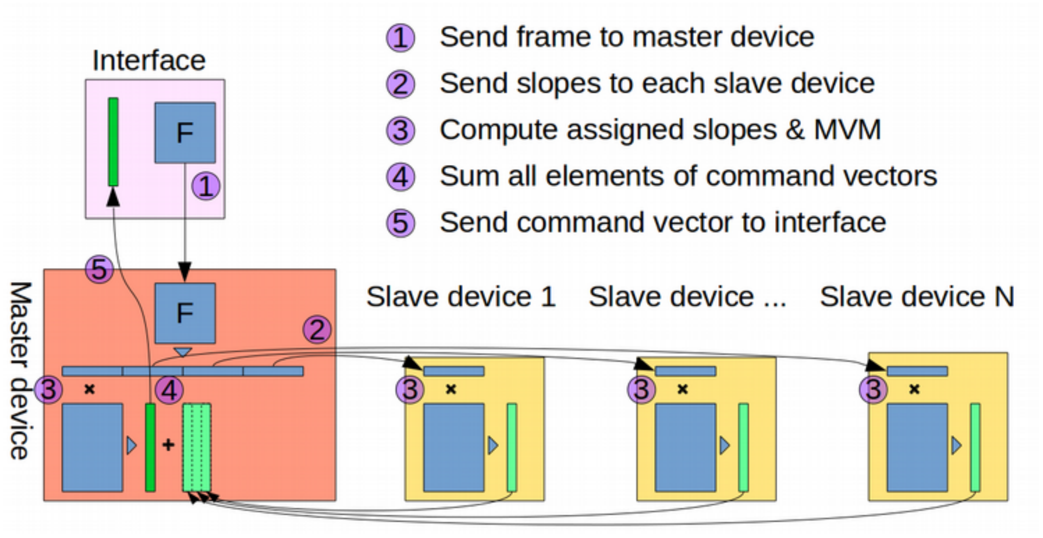
Interface
F
1 Send frame to master device
2 Send slopes to each slave device
3 Compute assigned slopes & MVM
4 Sum all elements of command vectors
5 Send command vector to interface
Master device
Slave device 1
Slave device ...
Slave device N

Figure 7: Generic multi-GPU system architecture.

units (GPUs) linked by a PCI-e bus. The node also need a host (CPU) in order to manage the PCI-e bus and communication with others nodes. All GPUs on the same node can directly interact using the PCI-e bus. The transfer limit depends on the PCI-e bandwidth but also mostly on the GPU copy engine performance (80Gb/s for the Kepler architecture).

This multi-GPU strategy requires the implementation of some hierarchy between the various processor and we have chosen the scheme represented in Figure 7. One of the processor is chosen as the master that directly interacts with the data interface. This is where direct memory access between the fraame grabber and the GPU is implemented. This master processor is connected to other slave processors, through high speed links to distribute the data efficiently. When the load for an iteration is executed, a final synchronization step between the slaves and the master allows to gather the data on the latter to be sent directly to the DM interface.

The Pascal architecture also introduces new features like 16 floating point numbers processing units and a new type of random-access memory that hugely increases the bandwidth. Additionally, NVlink a new kind of interconnect between GPUs, several time faster than PCIe, is available with the P100 architecture. With the new Pascal architecture, NVIDIA has made available a single node with 8 Tesla GP100 and 4 additional PCI-e ports for communication (first designed for infiniband boards). This single server, code named DGX-1, should be sufficient to run the MCAO pipeline (with ECC off). This should be compared to the 3 to 4 servers hosting Tesla K20 GPUs estimated from Table 3.

In this single server configuration, the NVlink interconnection between the GPUs is arranged as a mesh, composed of two main blocks interconnected to two PCIe interfaces attached respectively to the two CPU

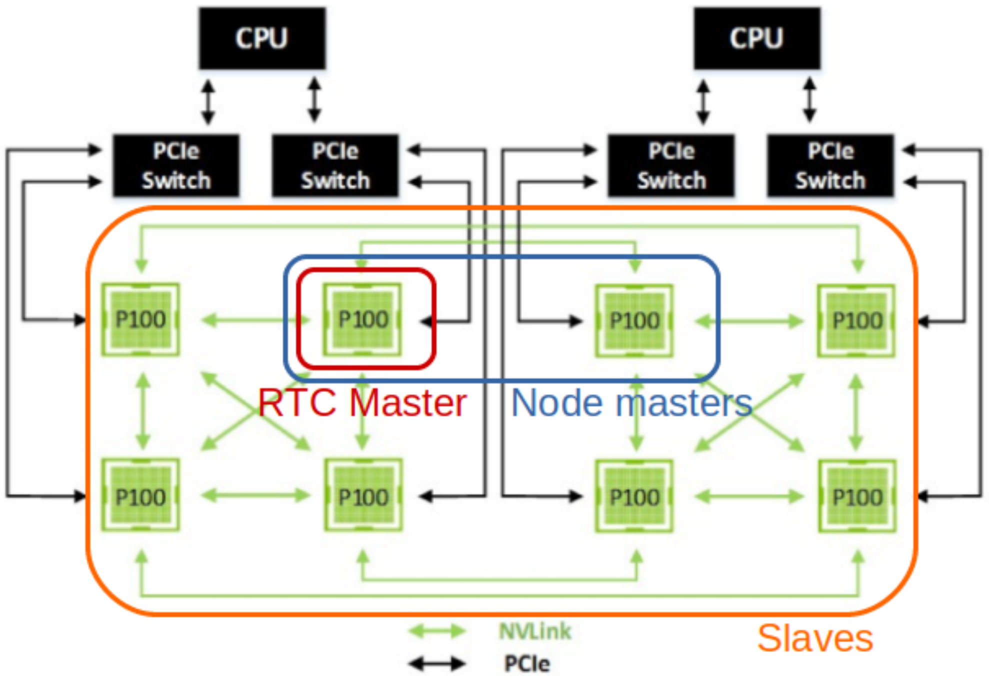

Figure 8: Mapping the generic multi-GPU system architecture on the NVIDIA DGX-1 server.

sockets. Two GPUs of each of these blocks have access to each PCIe bus and the interconnection between these two PCIe interface is done through the CPU sockets.

With this setup we chose to identify one RTC master, that synchronize the execution over the full pipeline and two node masters (on each blocks of the mesh), one of which being also the RTC master. The other GPUs are slaved to both these node masters. This is depicted in Figure 8. Each PCIe slot can then be populated with a frame grabber or generic network interface. This is pictured in Figure 9. Dedicated development for such a smart interconnect are presented by Perret et al. in this conference.

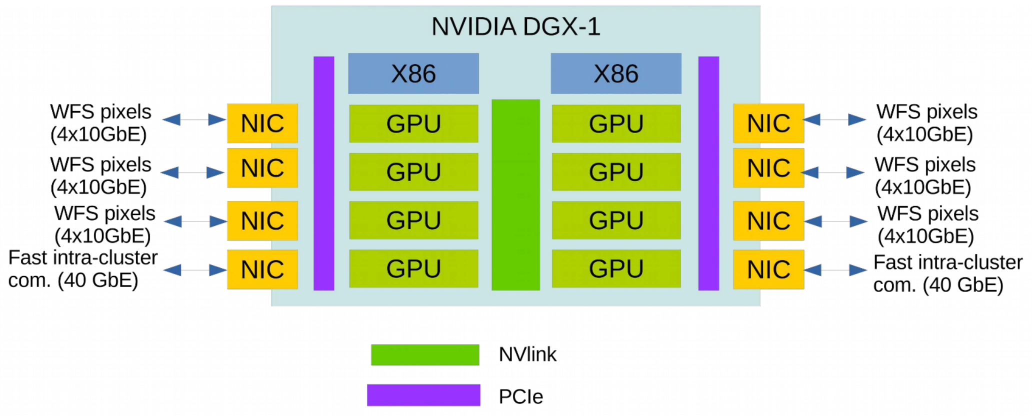

Figure 9: Proposed implementation on the NVIDIA DGX-1 server.

6. PRELIMINARY RESULTS

An experimental setup was implemented using a DGX-1 server and simulated data. Several system dimensioning were assessed from E-ELT scale SCAO to LTAO and MCAO. The following table outlines the various AO system complexities that were simulated.

Table 5: Assessed AO system dimensionings

Name	N slopes	N actuators	Goal frame rate
SCAO	10048	5316	1000 FPS
LTAO	60288	5316	500 FPS
MCAO	60288	15316	500 FPS

For the SCAO case, a single GPU was used to perform the pipeline. For the LTAO case, 4 GPUs, including one master and 3 slaves, were used in a single node, and for the MCAO case, 8 GPUs were used, spanning over 2 nodes in the slaves/master configuration showed in Figure 8.

In the pipeline we have tested, a set of simulated images resides on the host. For the LTAO and MCAO cases 6 images are sent to the GPUs from the host at each iteration (shared equally between the node masters in the MCAO case) while only a single frame is sent to a single GPU in the SCAO case. For the latter we consider the SCAO1 scheme, with a Shack-Hartmann WFS. The slope computation is done on the node masters and the slope vector is then shared equally between the node master and the its slaves. Each GPU then computes a partial command vector corresponding to its share of slopes and of the command matrix. The resulting contributions to the total command vector are then gathered hierarchically by the node masters and then by the RTC master.

In the SCAO case, the pipeline is over at this point. In the LTAO and MCAO cases, where a POL controller is implemented, the final command vector is then dispatched over all the GPUs where each partial slopes vector is computed using the corresponding part of the interaction matrix. This partial POL slopes vector is then added on top of the partial slopes vector produced by each node master from the images before doing the control MVM.

The performance in terms of time to solution and jitter are shown in figure 10 as histograms of the pipeline execution time (in s) over 100k iterations (hence 100 s of operations of the SCAO system and 200 s of operations of the LTAO/MCAO systems).

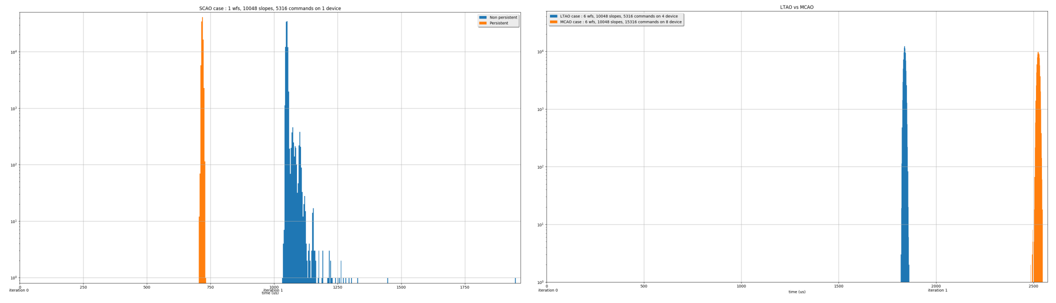

Figure 10: Performance of the pipeline for various system dimensioning and numbers of GPUs. Left: SCAO case for our persistent kernel approach (orange) and a standard (non persistent) kernel approach (blue). Right, performance for LTAO on 4 GPUs (blue) and MCAO on 8 GPUs (orange), using our persistent kernel approach.

In the SCAO case, we have compared the persistent kernel approach developed in this study to a classical (non persistent) kernel approach, in which the pipeline is regrouped in a single kernel which is launched at each iteration when data has been received, through a CPU process. The results show that the persistent kernel approach helps to reduce significantly both time to solution and jitter. These results show that the required performance of 1 kFPS or 1ms per iteration can be obtained on a single newest generation GPU following the persistent kernel approach. In the standard kernel approach, the mean time to solution obtained on a single GPU is larger than our initial specifications and a number of outliers around and beyond 1.5ms are observed that would lead to lost frames in a real system.

Concerning the LTAO and MCAO cases, only the persistent kernel approach has been assessed and the results show that a time to solution below 2ms can obtained for a LTAO system using only 4 GPUs, with reduced jitter. For the MCAO case, 8 GPUs are not enough to bring down the time to solution below 2ms. These plots however demonstrates that scaling up this pipeline to multiple GPUs over several nodes interconnected through NVlink does not impact jitter significantly. This is a major result of our study.

We have also estimated the achieved performance in terms of GMACs obtained to be compared with the sustained memory bandwidth of this kind of GPUs. This is displayed in Figure 11 for each case. In the LTAO and MCAO case, both the total performance and the performance per GPU has been displayed.

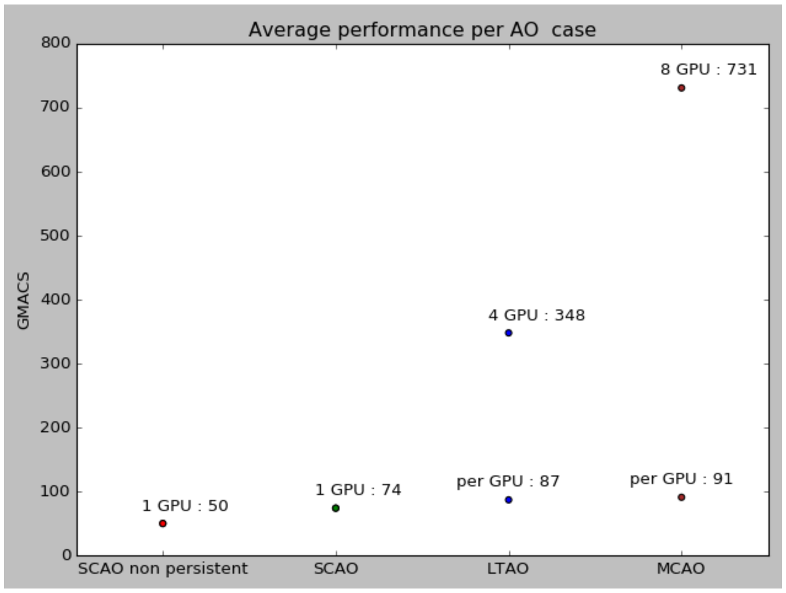

Figure 11: Performance of the pipeline in GMACs for various system dimensioning and numbers of GPUs.

These results show that a better performance per GPU is obtained in the LTAO and MCAO case as compared to SCAO. Indeed, the LTAO case is 6 times larger in terms of number of operations than the SCAO case and is treated with 4 GPUs. The amount of computations done by a single GPU is thus larger and allows to get closer to saturation in terms of compute engines occupation on the GPU. In the MCAO case, the problem scale is 18 times larger than is in the SCAO case and is processed by 8 GPUs.

If we look closer to these results and try to compare them to the GPU specifications recalled in table 2, the Tesla P100 has a theoretical memory bandwidth of 5760 Gb/s. The amount of data to be assimilated by the compute engines to perform the multiply-accumulates for a single iteration in the MCAO case is $60288 \times 15316 \times 2 \times 4 = 7.3$Gb. Each iteration is performed in about 2.5ms which leads to a measured sustained memory bandwidth of 2954 Gb/s or 51% of the theoretical memory bandwidth as given and 77% of the sustained BW as given in Table 3. Even though this probably requires some additional optimization, it shows that our approach is able to leverage most of the available bandwidth to execute the pipeline.

7. CONCLUSION

The developments reported in this paper, show how an efficient real-time data pipeline for AO can be implemented on GPUs while being scalable and fulfilling the time determinism constraint imposed by such a system. Persistent kernels and direct memory access are key to build a system with low latency and jitter.

Future developments will be oriented toward more portability for instance through the use of dedicated runtime systems, allowing to distribute computing load over heterogeneous resources under time deterministic constraint using a static scheduler.[6]

REFERENCES

[1] Fedrigo, E., Bourtembourg, R., Donaldson, R., Soenke, C., Valles, M. S., and Zampieri, S., "Sparta for the vlt: status and plans," (2010).

[2] Oberti, S., Kolb, J., Louarn, M. L., Penna, P. L., Madec, P.-Y., Neichel, B., Sauvage, J.-F., Fusco, T., Donaldson, R., Soenke, C., Valles, M. S., and Arsenault, R., "Aof ltao mode: reconstruction strategy and first test results," (2016).

[3] Enrico Fedrigo, R. D., "Sparta roadmap and future challenges," (2010).

[4] Gratadour, D., Dipper, N., Biasi, R., Deneux, H., Bernard, J., Brule, J., Dembet, R., Doucet, N., Ferreira, F., Gendron, E., Laine, M., Perret, D., Rousset, G., Sevin, A., Bitenc, U., Geng, D., Younger, E., Andrighettoni, M., Angerer, G., Patauner, C., Pescoller, D., Porta, F., Dufourcq, G., Flaischer, A., Leclere, J.-B., Nai, A., Palazzari, P., Pretet, D., and Rouaud, C., "Green FLASH: energy efficient real-time control for AO," in [*Adaptive Optics Systems V*], *Proc. SPIE* **9909**, 99094I (July 2016).

[5] Gendron, E., Morris, T., Basden, A., Vidal, F., Atkinson, D., Bitenc, U., Buey, T., Chemla, F., Cohen, M., Dickson, C., Dipper, N., Feautrier, P., Gach, J.-L., Gratadour, D., Henry, D., Huet, J.-M., Morel, C., Morris, S., Myers, R., Osborn, J., Perret, D., Reeves, A., Rousset, G., Sevin, A., Stadler, E., Talbot, G., Todd, S., and Younger, E., "Final two-stage moao on-sky demonstration with canary," (2016).

[6] Melani, A., Serrano, M. A., Bertogna, M., Cerutti, I., Quiñones, E., and Buttazzo, G., "A static scheduling approach to enable safety-critical openmp applications," in [*Design Automation Conference (ASP-DAC), 2017 22nd Asia and South Pacific*], 659–665, IEEE (2017).