Efficient Supervision Strategy for Tomographic AO Systems on E-ELT

Doucet N.[a,b], Gratadour D.[a], Ltaief H.[b], Gendron E.[a], Sevin A.[a], Ferreira F.[a], Vidal F.[a], Kriemann R.[c], and Keyes D.[b]

[a]LESIA - Observatoire de Paris/ Université Paris Diderot
[b]Extreme Computing Research Center - KAUST
[c]Max Planck Institute - Leipzig

ABSTRACT

A critical subsystem of the tomographic AO RTC is the supervisor module. Its role is to feed the challenging real-time data pipeline with a new reconstructor matrix at a regular rate, computed from a statistical analysis of the measurements, to optimize the performance of the AO system. This process involves solving a system of linear equations defined by the covariance matrix of the wave front sensors' measurements, the size of which may be up to 90k×90k for the E-ELT's first light instruments using tomographic AO modules. The computational load for the solver of this dense symmetric matrix system is quite significant at this scale but can be efficiently handled using state-of-the-art energy-efficient manycore x86 or accelerator-based architectures, such as KNLs or GPUs, respectively. As part of the Green Flash project, we develop a supervisor module and demonstrate its portability by deploying it on each aforementioned hardware system. Finally, we describe different implementations and their trade-offs in terms of performance and accuracy and show preliminary results on the possible impact of hierarchically low-rank approximation methods on the overall supervisor module.

Keywords: ELT, AO, tomography, MOAO, RTC, Dense Linear Algebra, High Performance Computing, Many-core Architectures, Low-Rank Approximations

1. INTRODUCTION

In order to achieve the best performance, Extremely Large Telescopes (ELTs) will rely on Adaptive Optics (AO) systems. Tomographic AO systems are currently developed to exploit the large field of view of this unprecedented scale of telescopes. Due to the ELTs size, the real-time control systems of these instruments become a computational challenge and require an efficient design to raise up to this challenge. Most notably, the supervisor module, which provides optimized tomographic reconstructor to the real-time data pipeline, has to achieve a throughput of around 100 TFlops. This threshold corresponds to the goal of updating the reconstructor matrix at a fast enough pace (in the order of a few minutes[1]) to follow the turbulence evolution and meet the image quality specifications.

For instance, this module is a part of the co-processing cluster (or soft real-time) of the Standard Platform for Adaptive optics Real Time Application (SPARTA) interface designed by ESO[2,3] . The SPARTA concept supports tomographic AO systems (such as the Ground Atmospheric Layer Adaptive Corrector for Spectroscopic Imaging GALACSI) and the supervisor approach, adopted in this platform, relies on the computation of a Minimum Mean Square Error (MMSE) reconstructor, as described in 4. While it seems to adequately perform with the scale of the Very Large Telescope Adaptive Optics Facilities (VLT AOF), the requirements and performance obtained in terms of time-to-solution to produce the new reconstructor is either not addressed or not sufficient to answer the ELT dimensioning challenge. Additionally, recent work on the Real Time Controller (RTC) dimensioning of various tomographic AO concepts for the E-ELT[5,6]) lacks of a proper analysis of how the image quality specification can map on both hardware and optimized software for the RTC.

In this work, we focus on the *Learn* and *Apply* pipeline methodology, which has been extensively tested on the CANARY demonstrator for various cases of atmospheric AO systems: Ground Layer AO (GLAO), Multi-Object AO (MOAO) and Laser Tomography AO (LTAO), as reported in 7. We assess the sustained performance of the supervisor module with state-of-the-art dense linear algebra libraries on latest hardware generations.

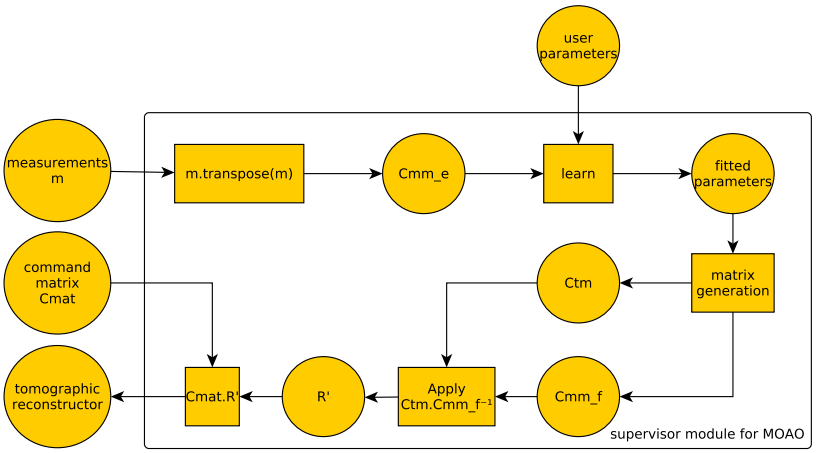

Figure 1: supervisor module diagram

Our baseline is the supervisor module depicted in Fig. 1, which proceeds in a the following steps: the computation of the covariance matrix of the measurements, the *Learn* & *Apply* phase and finally, a matrix-matrix multiplication operation. Although the generation of the covariance matrix is critical because it requires proper memory access management, this paper only focussed on the computational challenging Learn and Apply stage as it turns out to be the most time-consuming phase of the supervisor module.

The remainder of the paper is as follows. Section 2 describes the main algorithmic ideas behind the *Learn* pipeline and highlights qualitative and performance results. Section 3 presents the *Apply* pipeline, which represents the bulk of the computation in the supervisor module, and shows the portability of the implementation across various hardware architectures. In the same section, preliminary results on the performance impact of low-rank matrix approximations are also studied. We conclude in Section 4.

2. THE *LEARN* PIPELINE

The atmosphere can be represented as a set of turbulent layers with their own properties, such as the altitude, the outer scale and the strength. The objective of the *Learn* pipeline is to acquire knowledge or "learn" from the layer distribution of the atmosphere. The information regarding the turbulent layers can be deduced from the covariance matrix Cmm_e of actual measurements of the wave front sensors. Theoretical covariance matrices can be computed with the help of a spatial covariance model function[8] noted f in the subsequent sections. This function requires two sets of parameters. The first one is the set of parameters inherent to the system such as the telescope diameter, the central obscuration, the number Natural Guide Stars and Laser Guide Stars, the number of subapertures, the pixel size, the observation wavelength , the read out noise, the width of the laser beam and the number of targets. The second one is the set of atmospheric parameters. The system parameters are fixed for a given observing mode, and thus, will not be addressed here. The atmospheric parameters are the ones of interest and need to be retrieved. This set of parameters is composed, for each turbulence layer, of its altitude, outer scale and strength.

2.1 Methodology

The *Learn* pipeline consists in the minimization of the following score function:

$$S = \sum_{i,j} \left(Cmm_e(i,j) - f(x,i,j) \right)^2, \tag{1}$$

where x is the unknown and i, j the position in the matrix. In order to find the best atmospheric parameter vector x, we use a Levenberg-Marquardt algorithm,[9] already tested on the the 4.2m William Herschel Telescope

Algorithm 1 The Levenberg-Marquardt Algorithm.

$k = 0$; $\nu = 2$; $x = x_0$;
$H = J(x)^T J(x)$; $g = J(x)^T f(x)$;
$found = \|g\|_\infty \leq \epsilon_1$;
$\mu = \tau.max\{a_{ii}\}$;
while (**not** found) **and** $(k \leq k_{max})$ **do**
  $k = k + 1$;
  $solve((H + \mu.Id)h_{lm} = -g)$;
  if $h_{lm} \leq \epsilon_2(\|x\| + \epsilon_2)$ **then**
    $found =$ **True**;
  else
    $x_{new} = x + h_{lm}$;
    $\rho = (S(x) - S(x_{new})/(h_{lm}^T(\mu h_{lm} - g)/2)$;
    if $\rho > 0$ **then**
      $x = x_{new}$
      $H = J(x)^T J(x)$; $g = J(x)^T f(x)$;
      $found = \|g\|_\infty \leq \epsilon_1$
      $\mu = \mu.max\left\{1/3, 1 - (2\rho - 1)^3\right\}$;
      $\nu = 2$;
    else
      $\mu = \mu.\nu$; $\nu = 2.\nu$
    end if
  end if
end while

(WHT) in La Palma with the CANARY instrument.[7] It is an iterative method of type damped Gauss-Newton, as described in Algorithm 1.

This algorithm is composed of two nested loops. The outer loop where the Hessian H and gradient g of the score function are computed at the current point. The inner loop determines a descent direction step, by solving the system:

$$(H + \mu Id)x = g. \tag{2}$$

It then accepts or rejects this new vector x according to its score, and eventually, updates the damping parameter μ. The inner loop finishes when a new vector is accepted.

For a given matrix size, the overall execution time of a single iteration depends highly on the number of layers. For this reason, the implementation of the *Learn* pipeline calls the Levenberg-Marquardt algorithm two times: the first time with a limited number of layers and parameters in order to provide a rough estimate of a few turbulent layers in a short amount of time, and a second time with up to 40 turbulent layers to refine the atmospheric profile.

2.2 Hessian and Gradient Computations

The dimensions of H and g are defined by the number of parameters. In this case, it should not exceed 50. However, the computation of H and g as well as the score function relies on the core function of the covariance matrix generation function, which may be of dimension up to $100k$. In this regard, the execution time dedicated to solve the inner loop's system of Eq. (2) is not significant compared to any functions of the matrix generation. Since the computation of H and g are the most time-consuming parts of this pipeline, it becomes judicious to optimize these functions.

In the subsequent equations, N is the dimension of the covariance matrix Cmm_e (symmetric positive-definite) and $nparam$ is the number of parameters to be retrieved. To ease the notation, the score function Eq. (1) is

written as:

$$S(x) = \sum_{k=1}^{N \times N} \left(Cmm_k - f_k(x)\right)^2, \tag{3}$$

where $k = i + j \times N$. The Hessian and gradient are respectively defined as:

$$H = J^T \cdot J \tag{4}$$

$$g = J^T \cdot (Cmm - f(x)). \tag{5}$$

The above formulations will not be used since the Jacobian $J \in \mathbb{R}^{nparam \times N^2}$ with N being as large as $100k$. This would require both excessive memory footprint and execution time. It is also noteworthy to mention that given:

$$H_{i,j} = \sum_{k=1}^{N^2} \frac{\partial f_k}{\partial x_i} \cdot \frac{\partial f_k}{\partial x_j} \tag{6}$$

$$g_i = \sum_{k=1}^{N^2} \frac{\partial f_k}{\partial x_i} \cdot (Cmm_k - f_k(x)), \tag{7}$$

this allows to write:

$$H = \sum_{k=1}^{N^2} (J_k)^T \cdot J_k \tag{8}$$

$$g = \sum_{k=1}^{N^2} (J_k)^T \cdot (Cmm_k - fk(x)), \tag{9}$$

where J_k is the line k of the Jacobian. The equations (8) and (9) determine the contribution of each line of the Jacobian to the Hessian and gradient. Furthermore, it is possible to generate one line of the Jacobian from a single element of the covariance matrix. Hence, for each element of the matrix, it is possible to compute its contribution to the Hessian and gradient. This reduces the memory footprint of $H \in \mathbb{R}^{nparam \times nparam}$ and $g \in \mathbb{R}^{nparam}$. A convenient feature of the model function is the possibility to independently compute each element of the covariance matrix. This allows a high degree of parallelism, particularly suited for massively parallel devices, such as GPUs, since each contribution can be generated by a single GPU thread. In addition, the model function can return, for a single element, the contribution of one of its layers. This avoids redundant computations, since an atmospheric parameter only impacts one layer, while the derivatives of the other layers are null and do not need to be computed.

2.3 Filtering the Covariance Matrix

The covariances between the measures along axis x and the ones along the axis y (XY covariances) have a lower magnitude than the covariances between measures of a same axis (XX and YY covariances), as shown in Fig. 2. One can assume the XY covariances are not significantly contributing to the accuracy of to the overall *Learn* pipeline. In this regard, the *Learn* pipeline was tested in both cases: with and without using the XY covariances. As shown by the obtained results, the accuracy of the solution is similar when the XY covariances are not taken into account, as highlighted in Fig. 3. This filtering process allows to reduce memory footprint as well as a better execution time since only half of the data is processed while not degrading the solution accuracy.

2.4 Performance Results

The *Learn* pipeline is tested on two different systems, as described in Tab. 1). The goal is to reduce time to solution and to assess its strong scalability. The strong scaling consists in running the same problem size with an increasing number of processing elements. In this case, the program is using exclusively GPUs (the processing element) for the most time-consuming functions tested here (the computation of the Hessian, gradient and the score function).

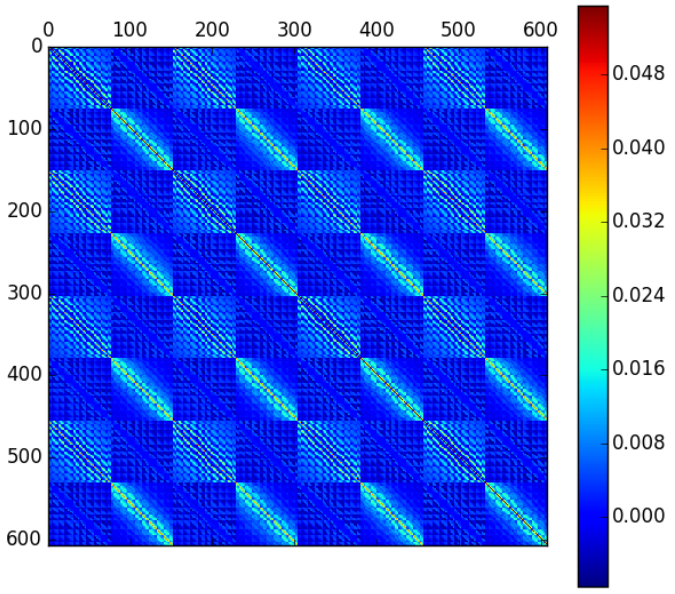

Figure 2: The Cmm_e covariance matrix: the darker the blocks, the lower the magnitude of the covariance between the measures along the x axis of a sensor and the y axis of another sensor.

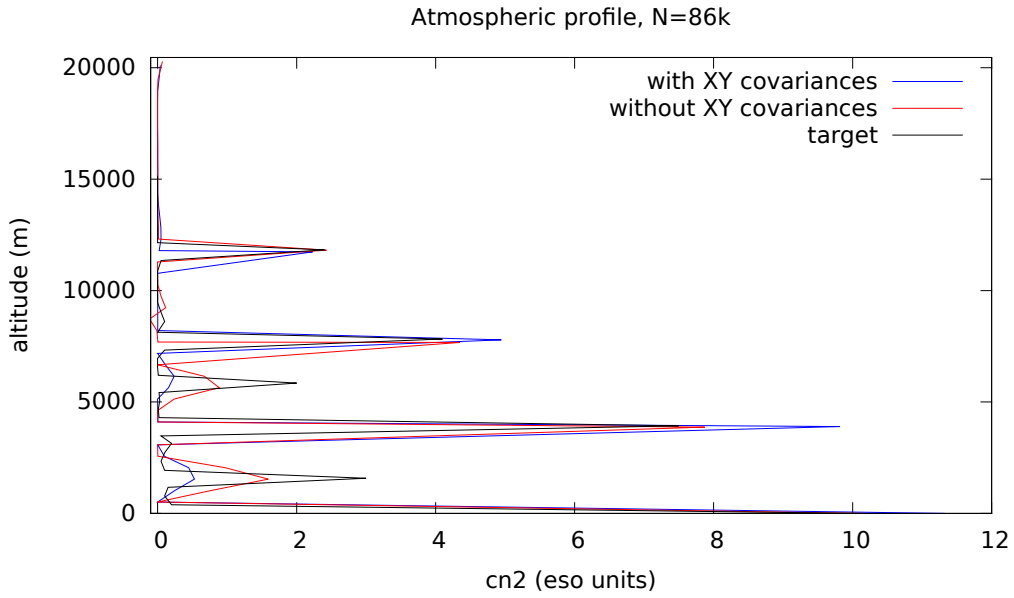

Figure 3: Atmospheric profiles. In black the target, in blue and red the output of the Learn pipeline with and without the XY covariances. The system parameters are the following, telescope diameter=40m, 9 wave front sensors including 6 laser guided stars with 80 subapertures along the diameter, with a total of 86688 measurements.

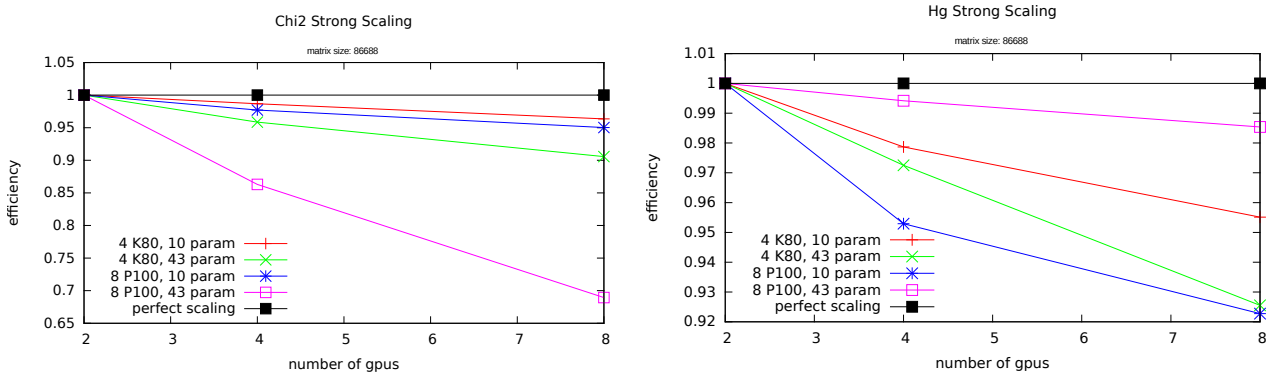

Figure 4: Strong scaling of the computation of the score (left) and the Hessian and gradient (right).

For the strong scaling, we consider a system at the E-ELT scale with a $40m$ diameter telescope and 9 wave front sensors including 6 laser guided stars with 80 subapertures along the diameter, for a total of 86688 measurements. The strong scaling graphs in Fig. 4 are built with the average execution time of single iteration, over all the iterations of the execution.

The efficiency remains above the 90%, except for the score function with 8 P100 and 43 parameters. In the latter case, more investigation is needed to better understand this unexpected behavior, since it requires the same amount of communications as the 10 parameters case with a more challenging computation load. In fact, these two functions contain a reduce operation (the sums in the equations (3),(8) and (9)), that may limit their scalability, although there are not enough computational resources to trigger the efficiency drop that would highlight this limitation.

The full execution of the *Learn* pipeline is currently achieved in about 300 seconds for the 86k case with the 8 NVIDIA P100 DGX-1 system against 1300 seconds with 4 NVIDIA K80 system (8 GPUs total), as depicted in Fig. 5. An execution time speedup of 4 is obtained between two eight GPUs systems that are one generation apart (while there is a factor of five between their peak performance). Reducing further the *Learn* computational time to less than a minute is a challenge that needs to be explored more densely-occupied systems or with distributed-memory systems.

Table 1: Hardware Specifications for the *Learn* Phase.

Name	System	Peak performance
hippo6	Intel(R) Xeon(R) CPU E5-2620 v3 @ 2.40GHz + 4 K80	7.9 TFlops
Nirvana	Intel(R) Xeon(R) CPU E5-2698 v4 @ 2.20GHz + 8 P100 GPUs (DGX-1)	43.8 TFlops

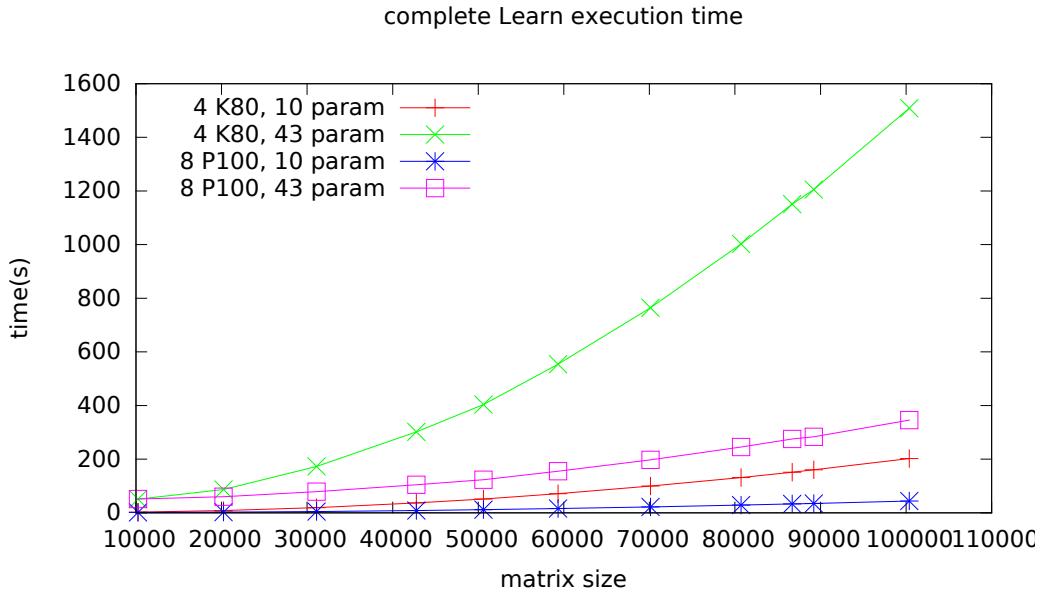

Figure 5: Execution time of the Levenberg-Marquardt algorithm.

3. THE *APPLY* PIPELINE

Once the *Learn* pipeline has been successfully executed, the covariance matrices of the wave front sensors measurements can be computed for the current atmosphere, knowing their observation's directions. This includes virtual sensors, and in particular, the one oriented toward the object of interest named the truth sensor. The *Apply* pipeline aims to provide a projector matrix (R') between the basis of the actual measurements and the one of the truth sensor's measurements. This projector matrix is obtained from the covariance matrix of the measurements of the actual sensors with themselves (Cmm) and the covariance matrix of the truth sensor measurements and those of the actual sensors (Ctm) using the following equation:

$$R' = Ctm \times Cmm^{-1} \tag{10}$$

We do not calculate the explicit inverse of the matrix, as this would require extra operations. Instead, we take advantage of the covariance matrix structure (symmetric positive-definite) and solve instead the following linear system of equations:

$$R' \times Cmm = Ctm, \tag{11}$$

with R' containing the unknowns.

3.1 The Chameleon and StarPU Software Libraries

The state-of-the-art Linear Algebra PACKage (LAPACK)[10] library provides routines to solve dense systems of the form $Ax = B$, where A is symmetric positive-definite and B is the right hand side, based on the Cholesky decomposition (named $DPOSV$). However, our actual system is rather of the form $xA = B$.

Therefore, we have to break down the $DPOSV$ routine into several subsequent calls to be able to solve $xA = B$, including the Cholesky factorization followed by a backward and a forward substitution, using the proper side parameters.

However, LAPACK has shown parallel performance limitations in the context of the manycore era mostly due to its bulk synchronous programming model. Indeed, LAPACK alternates computational memory-bound and compute-bound phases, using highly efficient multithreaded BLAS,[11] which does not allow to seamlessly extract performance from the underlying manycore hardware.[12] Therefore, we use the Chameleon library, a

task-based dense linear algebra library that is part of the Matrix Over Runtime System (MORSE).[13] The main objective of this project is to provide linear algebra methods for large scale multicore systems with hardware accelerators, such as x86 manycore or GPUs, and eventually aims for the exascale systems. As its name suggests, a task-based programming model expresses a sequential program as a succession of tasks and translates it into a directed acyclic graph, where nodes represent tasks and edges define the data dependencies among them. A dynamic runtime system StarPU[14] is then employed to efficiently schedule tasks across computational units (CPUs and/or GPUs, shared or distributed-memory environment), while (1) making sure the data dependencies are not violated and (2) hiding the overhead of data transfers.

Indeed, when it comes to heterogeneous systems, a given task can have several implementations to target different components of the system (CPU and/or GPU). The various tasks are then executed on the different part of the system, in parallel, according to the availability of the computational resources. In order to take advantage of this fine-grained programming model, the task scheduling and data movement need to be taken into consideration in a systematic way, especially when dealing with GPUs remote memory. This is where the runtime system approach allows applications' developers to experience productivity.

3.2 Performance Results

The Chameleon library performs dense linear algebra matrix computations and relies on StarPU for task scheduling across various shared/distributed memory and homogeneous/heterogeneous systems. The *Apply* pipeline has been tested on various systems, as described in Table 2.

Table 2: Hardware Specifications for the *Apply* Phase.

Name	system	Peak performance
Shihab	two sockets eighteen-cores Intel Haswell E5-2699 v3 @ 2.30GHz	1,3TFlops
Condor	one socket 64-cores Intel Xeon-Phi(TM) CPU7210 @ 1,3GHz (KNL)	2,7TFlops
Nirvana	Intel(R) Xeon(R) CPU E5-2698 v4 @ 2.20GHz + 8 P100 GPUs (DGX-1)	43.8TFlops

In our tests, the performance is estimated both in terms of achieved Flops/s and time to solution. We tested various matrix sizes, corresponding to various AO systems dimensioning, up to a typical E-ELT scale tomographic AO system with its $100k \times 100k$ measurements covariance matrix.

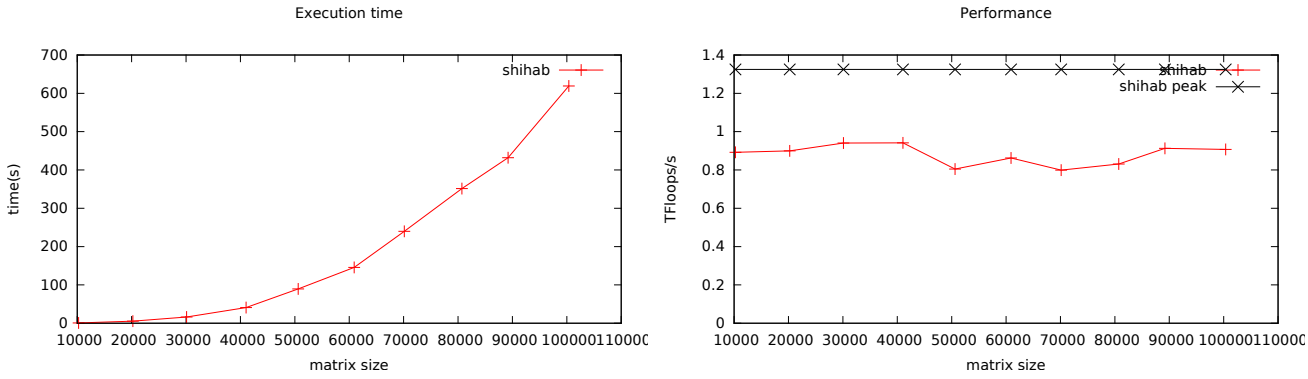

Figure 6: Best performance of the *Apply* pipeline on the Haswell system achieves about 70% of the peak performance with a maximum of approximately 1 TFlops/s and solves a $100k$ case in 600 seconds.

The peak performance speedup factor between condor, the single Intel KNL system and nirvana the 8xP100 GPUs system is about 16, as described in Tab. 2. The obtained performance ratio between the two systems is 16, as shown in Figs. 7 and 8). This also demonstrates the high portability of the code, which is able to run on

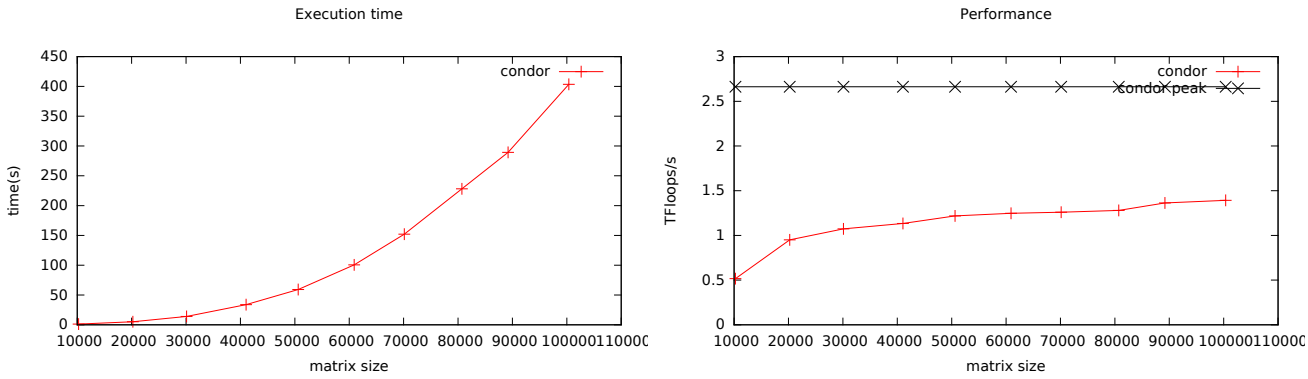

Figure 7: Best performance of the *Apply* pipeline on the Intel Xeon-Phi system achieves about 50% of the peak performance, with a maximum of 1.39 TFlops/s and solves a $100k$ case in 400 seconds.

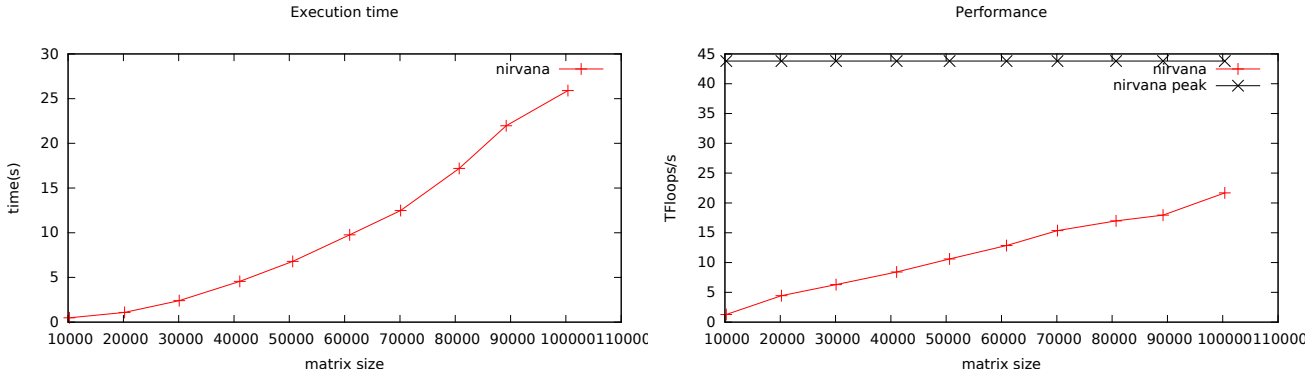

Figure 8: Best performance of the *Apply* pipeline on the DGX-1 (i.e., 8 P100 GPUs) achieves about 50% of the peak performance with a maximum of 21.69 TFlops/s, and solves a $100k$ case in 26 seconds.

Xeon Phi and GPU with similar efficiency, without changing the actual code. There are still lots of rooms for improvements, in particular, from a runtime perspective to further reduce date motion between GPUs.

In terms of time-to-solution, our benchmark shows that with the state-of-the-art dense linear algebra libraries Chameleon associated with the dynamic runtime system StarPU using leading edge densely-occupied GPU server (i.e., DGX-1), the tomographic reconstructor for an AO system of the size of the E-ELT first light instrument can be computed in less than 30 seconds. This is compliant with the initial requirement of a few minutes discussed in 1 for the overall *Learn* & *Apply* pipelines.

3.3 Pushing the Limits with Low-Rank Matrix Approximations

A hierarchical matrix ($\mathcal{H}$-matrix[15,16]) is a storage format for compressible dense matrices, which also permits full matrix arithmetic. $\mathcal{H}$-matrices are based on an hierarchical partitioning of the index sets for the rows and columns of the dense matrix in such a way that thereby constructed matrix subblocks may be identified as compressible. For those subblocks low-rank approximations are computed. For all non-compressible blocks of the matrix, the original dense representation is used.

The rank of the low-rank approximation is either predefined or adaptively chosen based on a given accuracy. Furthermore, low-rank blocks are represented in the factored form $M = A \cdot B^T$ with $M \in \mathbb{R}^{n \times m}, A \in \mathbb{R}^{n \times k}, B \in \mathbb{R}^{m \times k}$ and k being the rank of the low-rank matrix.

Covariance matrices generated here (Cmm, Ctm) are representable in the $\mathcal{H}$-matrix format and the compression rate (defined as $memory(compressed)/memory(dense)$) of these matrices, assessed with the library

HLIBpro,[17] can be up to 40%.

This figure was obtained by first assessing the accuracy required for the low-rank approximation of the covariances matrices (Cmm, Ctm) so that the reconstructor is accurate enough for the AO system (tested on an end to end simulation tool: compass) then tuning the compression of the covariances matrices, testing several different orderings for the rows and columns and tuning compression parameters such as the minimal block size.

But problems arise during $\mathcal{H}$-matrix factorization and solving the system (11). The rank of the low-rank subblocks increases (e.g., in (R')) such that the memory consumption is equal to the dense case. However, when applying the $\mathcal{H}$-matrix compression directly to R' computed in dense format, compression is observable.

Investigating further how this formalism could be applied to integrate the supervisor module is our short term objective, since it may further reduce the time to generate the covariance matrices.

4. CONCLUSION

Even though the *Learn* pipeline could be further optimized and the task-based implementation of the *Apply* pipeline improved, the performance results presented in this paper are still very encouraging. It demonstrates that an efficient supervisor strategy for tomographic AO can be implemented on leading edge hardware and software technologies, with a degree of portability and optimized performance through the use of standard linear algebra and runtime system libraries. The time-to-solution for the full *Learn* & *Apply* pipelines, targeting a tomographic AO system instrument with up to 10 high order WFS on the E-ELT, lies in the range of a few minutes on COTS multi-GPU platforms. This represents a major milestone moving forward, since it is already compatible with rough estimates of the requirements in terms of reconstructor update rates, in order to follow the turbulence evolution structure during a long exposure. If the science requires performance to be pushed further and if a tomographic reconstructor is to be provided every minutes, with this implementation of the *Learn* & *Apply* methodology, one would require several multi-GPU servers, with possible performance loss due to inter-server communication. Also, from a first assessment with the H-Matrix library **HLIBpro**, the *Apply* pipeline structure does not seems to fit well with state-of-the-art H-Matrix frameworks. However, leveraging the compression of the tomographic reconstructor using H-Matrix formalism may have significant impact in reducing the latency of the real time data pipeline.

Acknowledgments

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. The authors would like to thank the Intel and NVIDIA vendors for their hardware donations and/or systems' remote access, the Intel Parallel Computing Center and the NVIDIA GPU Research Center awarded to the Extreme Computing Research Center at KAUST.

REFERENCES

[1] Gendron, E., Morel, C., Osborn, J., Martin, O., Gratadour, D., Vidal, F., Le Louarn, M., and Rousset, G., "Robustness of tomographic reconstructors versus real atmospheric profiles in the ELT perspective," (2014).

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

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

[4] 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).

[5] Schreiber, L., Diolaiti, E., Arcidiacono, C., Baruffolo, A., Bregoli, G., Cascone, E., Cosentino, G., Esposito, S., Felini, C., Foppiani, I., Ciliegi, P., Feautrier, P., and Torroni, P., "Dimensioning the MAORY real time computer," (2016).

[6] Basden, A., Dipper, N., Myers, R., and Younger, E., "An AO real-time control solution for ELT scale instrumentation and application to EAGLE," (2012).

[7] 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).

[8] Gendron, E., Charara, A., Abdelfattah, A., Gratadour, D., Keyes, D., Ltaief, H., Morel, C., Vidal, F., Sevin, A., and Rousset, G., "A novel fast and accurate pseudo-analytical simulation approach for MOAO," (2014).

[9] Madsen, K., Nielsen, H. B., and Tingleff, O., "Methods for non-linear least squares problems (2nd ed.)," (2004).

[10] Anderson, E., Bai, Z., Bischof, C., Blackford, S. L., Demmel, J. W., Dongarra, J. J., Croz, J. D., Greenbaum, A., Hammarling, S., McKenney, A., and Sorensen, D. C., [*LAPACK User's Guide*], Society for Industrial and Applied Mathematics, Philadelphia, Third ed. (1999).

[11] BLAS, "Basic Linear Algebra Subprograms." http://www.netlib.org/blas/.

[12] Agullo, E., Hadri, B., Ltaief, H., and Dongarrra, J., "Comparative study of one-sided factorizations with multiple software packages on multi-core hardware," *International Conference on High Performance Computing Networking, Storage and Analysis* , 1–12, ACM/IEEE, New York, NY, USA (November 2009).

[13] "Morse project." http://icl.cs.utk.edu/projectsdev/morse/. (Accessed: 28 Semptember 2017).

[14] "Starpu." http://starpu.gforge.inria.fr/. (Accessed: 28 Semptember 2017).

[15] Hackbusch, W., "A sparse matrix arithmetic based on H-Matrices. part I: Introduction to H-Matrices," *Computing* **62**, 89–108 (1999).

[16] Grasedyck, L., "Adaptive recompression of H-matrices for BEM," *Computing* **74**, 205–223 (2005).

[17] "Hlibpro." http://www.hlibpro.com/. (Accessed: 28 Semptember 2017).