Performance Evaluation of Dense Linear Algebra Kernels using Chameleon and StarPU on AWS

Vinicius Garcia Pinto[1], **João V. F. Lima**[2], **Vanderlei Munhoz**[3], **Daniel Cordeiro**[4], **Emilio Francesquini**[5], **Márcio Castro**[3]

[1]Federal University of Rio Grande (FURG), Rio Grande, RS, Brazil

[2]Federal University of Santa Maria (UFSM), Santa Maria, RS, Brazil

[3]Federal University of Santa Catarina (UFSC), Florianópolis, SC, Brazil

[4]University of São Paulo (USP), São Paulo, SP, Brazil

[5]Federal University of ABC (UFABC), Santo André, SP, Brazil

vinicius.pinto@furg.br, jvlima@inf.ufsm.br, vanderlei.filho@proton.me, daniel.cordeiro@usp.br, e.francesquini@ufabc.edu.br, marcio.castro@ufsc.br

Abstract. *Due to recent advances and investments in cloud computing, public cloud providers now offer GPU-accelerated and compute-optimized Virtual Machine (VM) instances, allowing researchers to execute parallel workloads in virtual heterogeneous clusters in the cloud. This paper evaluates the performance and monetary costs of running dense linear algebra algorithms extracted from the Chameleon package implemented using StarPU on Amazon Elastic Compute Cloud (EC2) instances. We evaluated these metrics with a single powerful/costly instance with four NVIDIA GPUs (fat node) and with a cluster of five less powerful/cheaper instances with a single NVIDIA GPU in each node. Our results showed that most of the linear algebra algorithms achieved better performance and lower monetary costs on the fat node scenario even with one less GPU.*

1. Introduction

Task-based programming interfaces offer software developers a paradigm where the computation is decomposed into smaller units of work called tasks. Tasks represent individual units of work that can be executed concurrently and independently, thus allowing parallel execution of code. In this model, developers specify tasks and their dependencies, but they do not explicitly manage threads or parallel execution. Instead, a runtime system or scheduler dynamically manages the execution of tasks, mapping them to available computational resources such as CPU cores, GPU accelerators, FPGAs, or distributed nodes.

StarPU [Augonnet et al. 2010] is one of these runtime systems. In particular, using StarPU, a developer can provide different implementations for the same task (*e.g.*, one for CPUs, other for GPUs) and let the runtime environment automatically select the best implementation based on the computing environment and resource availability. This is done in a way that aims at performance optimization. For instance, the runtime might employ intricate resource selection strategies to minimize network transfers between nodes in a shared-memory environment. More importantly, this is done to isolate the developer from the details of the target computing architecture, allowing for the creation of simpler and more readily scalable applications.

Recently, significant progress and investments in Cloud Computing have resulted in a notable expansion of services provided by public cloud providers. One of the key advancements has been the introduction of GPU-accelerated and compute-optimized Virtual Machine (VM) instances. These specialized VM configurations offer researchers and developers the ability to execute parallel workloads in virtual environments, giving them the illusion of physical on-premise heterogeneous computing hardware. This development marks an important shift in Cloud Computing capabilities, enabling users to leverage the power of GPUs and optimized compute resources for a wide range of applications and tasks [Netto et al. 2018].

Understanding the performance and cost dynamics of executing task-based applications on Cloud Computing platforms is crucial since it allows organizations and developers to make more informed decisions about resource allocation and budgeting. By comprehensively evaluating the trade-offs between performance and monetary costs across different deployment scenarios, users can optimize their Cloud Computing strategies to strike a good balance between efficiency/makespan constraints and available budget. In particular, dense linear algebra applications feature CPU and/or memory workloads and it is unclear how they will behave when executed in public Cloud Computing infrastructures, specially when executed on multiple nodes interconnected by a network. Resources offered by public cloud providers are usually shared among different users and high-performance network interconnections between computing nodes may not be available in some public cloud providers.

In this paper, we show an evaluation of the performance and associated financial costs of executing dense linear algebra algorithms extracted from a widely used linear algebra library (Chameleon [Agullo et al. 2012]) with a state-of-the-art task-based runtime environment (StarPU). Experiments were executed on Amazon Elastic Compute Cloud (EC2) instances. The evaluation was performed in two different scenarios: (i) using a single high-powered (yet costly) instance equipped with four NVIDIA GPUs; (ii) employing a cluster configuration consisting of five less potent (but more economical) instances, each equipped with a single NVIDIA GPU. Experimental results showed that, for most of the dense linear algebra applications considered in this study, the first configuration yielded superior performance compared to the latter configuration and also it incurred lower financial costs than the clustered setup (even if it has one GPU less).

These results suggest that while distributed configurations may offer scalability and some degree of fault tolerance in some very specific cases, they may not always provide optimal performance or minimize financial cost, especially for compute-intensive tasks like dense linear algebra computations. Similarly, they also suggest that consolidation of tasks in a single powerful node, when feasible, considering resource constraints, might be desirable. Finally, they indicate that the current state-of-the-art task-based runtime libraries employed in the High Performance Computing (HPC) domain lack specific optimizations to take full advantage of virtualized clusters built on top Cloud Computing infrastructures.

The remainder of this paper is organized as follows. Section 2 outlines the concepts used in this work. Then, in Section 3, we detail our experimental methodology followed by the experimental results in Section 4. Section 5 lists related work, and Section 6 concludes this work. A publicly available companion at `https://gitlab.com/viniciusvgp/companion-chameleon-aws-2024.git` includes the experiments raw data.

2. Background

In this section, we first give a brief overview of the task-based runtime system and the dense linear algebra library used in this paper (Section 2.1). Then, we present some details about the cloud infrastructure employed in this paper (Section 2.2).

2.1. StarPU and Chameleon

StarPU [Augonnet et al. 2010] is a framework written in C for task-based programming on hybrid CPU–GPU platforms. Applications using StarPU define tasks, *i.e.*, wrappers for functions whose parameters are flagged as `read`, `write`, or `readwrite`. To target different hardware architectures, the programmer can provide multiple implementations for the same task, *e.g.*, pure sequential C, C with SSE/AVX extensions, CUDA and OpenCL. At runtime, StarPU will dynamically choose the best task implementation to run, considering resource availability, load distribution, and required data copies. StarPU transparently takes care of required data copies on multicore platforms enhanced with GPUs, allocating/deallocating GPU memory, deciding the appropriate number of threads based on the available hardware, and overlapping transfers with computation through data prefetching. On cluster platforms equipped with GPUs, StarPU relies on MPI by combining usual MPI calls as `Send` and `Recv` with StarPU data management ones. The set of created tasks is unrolled into a Directed Acyclic Graph (DAG) by using creation order and access mode on each parameter. StarPU entirely manages intra-node load balancing among threads by using several scheduling policies, while inter-node one (among MPI processes) relies on static DAG partition based on application runtime parameters.

Chameleon [Agullo et al. 2012] is a framework written in C that provides routines to solve dense general systems of linear equations, symmetric positive definite systems of linear equations, and linear least squares problems using LU, Cholesky, QR, and LQ factorizations. At a higher level, Chameleon is based on PLASMA algorithms [Buttari et al. 2009] and relies on runtime systems such as StarPU, PaRSEC, or OpenMP to dynamically schedule its tasks, while the inner implementation of each task uses classical BLAS and LAPACK routines. This paper uses Chameleon with the StarPU runtime system to execute linear algebra algorithms on multiple heterogeneous (CPU+GPU) nodes in AWS.

Currently, StarPU and Chameleon do not feature any optimization targeting Cloud Computing VMs to enable efficient, economical, and resource-saving execution of parallel applications in the cloud. In this paper, we evaluate how they perform on AWS both in terms of performance and monetary costs.

2.2. Amazon Web Services (AWS)

Public clouds are designed to be accessible to anyone via the Internet without needing long-term contracts or direct interaction with the provider. Three types of services traditionally describe their service model. Infrastructure as a Service (IaaS) refers to online services that provide APIs for users to spawn and manage compute infrastructure, including low-level details such as network, storage, and backups. The user often can choose the computing capacity of the infrastructure to be rented—typically in terms of virtual CPUs (vCPUs)—and can also configure other details such as hypervisor types, pre-installed OS, accelerators, and more. Platform as a Service (PaaS) refers to services that the user can use to create and deploy custom software applications using a configurable environment hosted by the cloud provider. Runtime, middleware, and software features are abstracted from the user and managed by the provider. Finally, Software as a Service (SaaS) are ready-to-use software applications with specific purposes that the provider typically offers through APIs, which users can use directly in their applications.

AWS is one of the most popular public cloud providers. It maintains physical resources in several geographically dispersed data centers, giving users access to computing resources in the form of instances. A VM is a type of virtually allocated instance on a shared physical infrastructure maintained by AWS and is generally the most available and affordable option. AWS offers various types of instances with different characteristics such as type of hypervisor, CPU architecture, number of vCPUs, or amount of memory, allowing users to choose the instance that best suits their needs.

Recently, AWS launched numerous HPC products and services. Its offerings include infrastructure options featuring: (i) high-speed interconnections, which employ its proprietary Elastic Fabric Adapter (EFA)[1] network interface that enables customers to run applications requiring high levels of inter-node communications at scale; (ii) large VM instances equipped with hundreds of vCPUs to cater to applications with substantial processing demands; (iii) instances equipped with accelerators such as GPUs or TPUs; (iv) fully managed shared storage named FSx[2] built on popular high-performance file systems (*e.g.*, Lustre[3] and OpenZFS[4]).

3. Experimental Methodology

In this section, we present our experimental methodology. First, we give an overview of the parallel applications considered in this study. Then, we discuss the cloud infrastructure and the evaluated scenarios.

3.1. Applications

We considered four dense linear algebra kernels from Chameleon (`gemm`, `potrf`, `getrf` and `geqrf`). We also ran each kernel with single precision (`s` variant) and double precision (`d` variant). A brief description of each kernel is given below:

[1]`https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/efa.html`
[2]`https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/storage_fsx.html`
[3]`https://www.lustre.org`
[4]`https://openzfs.org/wiki/Main_Page`

Matrix multiplication (`sgemm/dgemm`): the matrix multiplication has the form $C = AB + C$ with a parallel blocked algorithm with a single task that operates over the data tiles. The task has both CPU and GPU implementations;

Cholesky factorization (`spotrf/dpotrf`): the Cholesky factorization decomposes an $n \times n$ real symmetric positive definite matrix A into the form $A = LL^T$ where L is an $n \times n$ real lower triangular matrix with positive diagonal elements. The task-based algorithm has four tasks types in total: `POTRF`, `SYRK`, `TRSM`, and `GEMM`. The `POTRF` task has only a CPU implementation, while the others have CPU and GPU implementations;

LU factorization (`sgetrf/dgetrf`): the LU factorization of a matrix A has the form $A = LU$ where L is lower triangular and U is upper triangular. The task-based algorithm is the version without pivoting and has three tasks types in total: `GETRF`, `TRSM`, and `GEMM`. The `GETRF` task has only the CPU implementation, while the others have CPU and GPU implementations;

QR factorization (`sgeqrf/dgeqrf`): the QR factorization of an $m \times n$ real matrix A has the form $A = QR$ where Q is an $m \times m$ real orthogonal matrix and R is an $m \times n$ real upper triangular matrix. The task-based algorithm has four tasks types in total: `GEQRT`, `UNMQR`, `TPQRT`, and `TPMQRT`. `GEQRT` and `TPQRT` have only the CPU implementation, while the other two have CPU and GPU implementation.

All four kernels have the tile data layout for matrix representation. They were executed with input matrices of size $67,200 \times 67,200$ (i.e., with 70×70 tiles of size 960×960). Panel factorizations of Cholesky, LU, and QR are tasks with no corresponding GPU implementation.

3.2. Cloud Infrastructure

We leveraged the HPC@Cloud Toolkit [Munhoz et al. 2023, Munhoz and Castro 2024, Munhoz et al. 2022] to set up the cloud infrastructure on AWS in a reproducible manner. HPC@Cloud is a provider-agnostic software toolkit that enables users to manage virtual HPC clusters in public clouds with minimal effort. It offers a suite of tools that can be executed on the user's machine. These tools enable users to configure cloud infrastructure, execute jobs, monitor performance, predict costs, and interact with the cloud in an automated and provider-agnostic manner. HPC@Cloud is open-source software developed using Rust for the primary command-line application and Python for automation and data-gathering scripts.

An Amazon Machine Image (AMI) with all the software needed to compile and run the experiments discussed in this paper is also available in the HPC@Cloud Toolkit repository.[5]

3.3. Evaluated Scenarios

To evaluate Chameleon kernels on AWS, we prepared two configurations:

1. A single **fat node**: a single `g4dn.12xlarge` instance configured with an Intel Cascade Lake P-8259CL processor running at 2.5 GHz (48 vCPUs, 24 physical

[5]`http://github.com/lapesd/hpcac-toolkit`

cores), 4× NVIDIA T4 GPUs (64 GB of memory total), and 192 GB of RAM. During our experiments, the on-demand price of this instance was USD 3.912 per hour.

2. A cluster of five **thin nodes**: five `g4dn.2xlarge` instances, each one configured with an Intel Cascade Lake P-8259CL processor running at 2.5 GHz (8 vCPUs, 4 physical cores), 1× NVIDIA T4 GPU (16 GB of memory each) and 32 GB of RAM. Thus, this cluster has 20 physical CPU cores and 5× NVIDIA T4 GPUs. During our experiments, the on-demand price of the `g4dn.2xlarge` instance was USD 0.752 per hour, so the cost of the full setup with five nodes was USD 3.76 per hour.

When using the fat node setup, we configured StarPU to use 19 threads to execute CPU tasks (computation), 4 threads to control each GPU and 1 thread to run the application's `main` function. StarPU scheduler policy was set to `dmdas` to take load-balancing decisions based on tasks' previous execution and data transfer times.

For the scenarios using the cluster of thin nodes, we deployed, per node, 1 MPI process, with 2 threads to compute CPU tasks (computation), 1 thread to control the node GPU, and 1 thread to run the application's `main` function. The StarPU scheduler policy (intra-node scheduling) was also set to `dmdas` in each node, while the Chameleon P parameter dictates inter-node load balancing.

In both cases, the number of worker threads on the CPU or the GPU was determined by the chosen runtime environment. All experiments were repeated 5 times, and the presented results represent average values. We employed Spack [Gamblin et al. 2015] to install all the software dependencies (Chameleon 1.2.0, StarPU 1.4.1, OpenMPI 4.1.2, and CUDA 11.8.0).

4. Experimental Results

The goal of our experiments is to evaluate the performance and financial costs of the four dense linear algebra kernels on two infrastructure scenarios on AWS. Our experimental evaluation is based on the following analyses: performance evaluation (in GFLOPS) of the four kernels in single and double precision on the cluster and fat node scenarios, evaluation of the performance peak attained on both scenarios for `gemm`, and the trade-off between the cluster and fat node scenarios on performance and monetary costs.

4.1. Evaluation of Chameleon's Static Distribution

Chameleon's P parameter determines how the DAG of tasks will be split across cluster nodes following the ScaLAPACK strategy to distribute a matrix among the processes [Choi et al. 1996]. This is a static workload division and does not change at runtime, which differs from the adaptive load distribution conducted by the StarPU instance running at each node. For this reason, inappropriate values can reflect on application performance bottlenecks [Garcia Pinto et al. 2018].

Since our cluster setup has five nodes, the only two possible values for P are 1 and 5. Figure 1 shows the impact on the GFLOPS rate for the four applications using single precision. For all cases but `sgeqrf` (QR), our choice of P value has a minor influence on

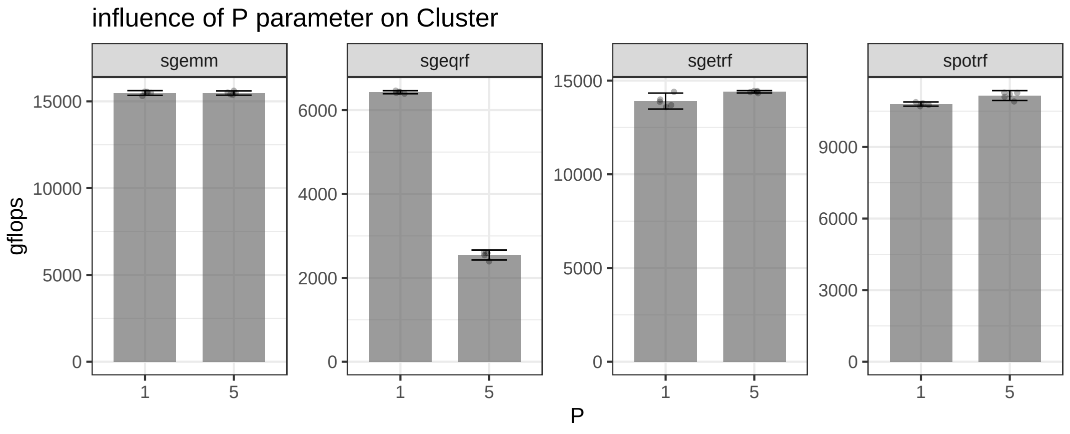

Figure 1. Influence of Chameleon static workload division (P) on the cluster scenario (5 nodes featuring 4 CPUs and 1 GPU each).

Table 1. Performance of the four kernels (in GFLOPS) with single and double precision on the cluster (5 nodes featuring 4 CPUs and 1 GPU each) and fat node (24 CPUs and 4 GPUs) scenarios, followed by the percentage gap between fat node over cluster scenario.

	Single Precision			Double Precision		
Benchmark	cluster	fat node	Diff.	cluster	fat node	Diff.
Matrix multiplication (`gemm`)	15,482.74	18,081.56	14.3%	1,773.91	2,060.04	13.8%
Cholesky (`potrf`)	10,797.13	15,291.12	29.3%	1,647.31	2,077.14	20.6%
LU (`getrf`)	13,906.72	15,213.85	8.5%	1,767.38	2,068.83	14.5%
QR (`geqrf`)	6,426.93	6,006.89	-6.5%	1,161.05	990.31	-14.7%

performance, whereas executions with $P = 5$ present slightly higher GFLOPS. However, with `sgeqrf`, setting P to 5 drops the GFLOPS rate to 40% of the one reached with $P = 1$. Therefore, we set $P = 1$ in all other cluster experiments this paper shows.

4.2. Performance Evaluation

Table 1 shows the performance results of the four applications on each platform and the percentage gap between the fat node over the cluster scenario. We used the GFLOPS rate reported at the application output as a performance metric. The fat node outperforms the cluster in most cases (`gemm`, `potrf`, and `getrf`). The `gemm` and `potrf` kernels had significant gaps between the fat node and cluster scenarios. Nonetheless, `geqrf` performed better on cluster than the fat node with a performance gap of 6.5% and 14.7% for single and double precision, respectively.

These results can be explained by assuming that QR factorization has fewer BLAS-3 tasks than the other three kernels that are compute-bound and their workloads are dominated by BLAS-3 matrix-to-matrix operations that mostly execute on GPUs. On the other hand, `geqrf` panel factorization comprises small BLAS-2 calls that are memory-bound and do not have enough parallelism for GPUs [Agullo et al. 2011]. The hybrid task-based algorithms of Chameleon perform panel factorization on CPUs and trailing matrix updates on GPUs or CPUs. Moreover, we observed that StarPU peer-to-peer GPU

Table 2. Monetary cost (in USD) of the four kernels with single and double precision on the cluster (5 nodes featuring 4 CPUs and 1 GPU each) and fat node (24 CPUs and 4 GPUs) scenarios, followed by the percentage gap between fat node over cluster scenario.

	Single Precision			Double Precision		
Benchmark	cluster	fat node	Diff.	cluster	fat node	Diff.
Matrix multiplication (`gemm`)	0.0409	0.0364	-12.2%	0.3573	0.3201	-11.6%
Cholesky (`potrf`)	0.0097	0.0071	-36.1%	0.0641	0.0529	-21.2%
LU (`getrf`)	0.0152	0.0144	-5.1%	0.1195	0.1062	-12.5%
QR (`geqrf`)	0.0657	0.0731	10.1%	0.3640	0.4440	18.0%

transfers improved the performance of the kernels dominated by BLAS-3 operations on the fat node scenario compared to MPI messages from the cluster scenario.

4.3. Monetary Costs

Table 2 shows the monetary costs of each application running with single and double precision on both platforms and the percentage gap between fat node over the cluster. Since the cluster setup had lower performance results, the fat node had a financial advantage in most cases. For the `dgeqrf` kernel, it is cheaper to use the cluster setup since it spends $\approx 82\%$ (double) and $\approx 89\%$ (single) of the amount required at the fat node.

4.4. GEMM Peak Performance

Table 3 shows an overview of the performance results of `gemm` running with single and double precision. The CPU-only performance was obtained with Chameleon, and results with 1 and 4 GPUs were obtained with CUBLAS and CUBLAS-XT from NVIDIA. We estimated the peak performance, adding results from CPUs and GPUs, and calculated the difference from this peak for the fat node and cluster scenarios. Single precision results outperformed our peak performance estimation by 6.9% and 32% on cluster and fat node scenarios, respectively. On the other hand, double precision results were below our peak performance estimation by 18.1% and 10.4% for cluster and fat node scenarios, respectively. We believe that the single precision results from Chameleon outperformed our peak performance estimation due to its optimizations for multi-GPU, such as the overlap of computation and transfers.

5. Related Work

Dense linear algebra is an important tool of computational sciences. Due to its intrinsic computational complexity, several works have been done on distributing linear algebra computations on HPC clusters. ScaLAPACK [Choi et al. 1992] proposed distributed versions of the Level 3 BLAS implemented in LAPACK [Demmel 1989]. Distributed PLASMA (DPLASMA) [Bosilca et al. 2011] introduced a novel scheduling algorithm capable of dynamically distributing dense linear algebra algorithms on distributed systems. *Poulson et al.* [Poulson et al. 2013] presented Elemental, a framework designed from scratch that leverages the distributed memory of many-core systems. Recent work

Table 3. Matrix multiplication (`gemm`) performance (in GFLOPS) with different CPU/GPU configurations on fat node and cluster scenarios. We obtained the 1x GPU result from CUBLAS and the 4x GPU from CUBLAS-XT.

	Single Precision		**Double Precision**	
	cluster	fat node	cluster	fat node
CPUs	1,784.18	2,672.99	965.35	1,389.24
GPUs	2,537.63	11,018.68	240.23	910.67
Peak (CPUs+GPUs)	14,472.33	13,691.67	2,166,5	2,299.91
Chameleon	15,482.74	18,081.56	1,773.91	2,060.04
Diff. from peak	6.98%	32.06%	-18.12%	-10.43%

from *Beaumont et al.* [Beaumont et al. 2023] investigated how task-based runtime systems like StarPU and Chameleon can be used to build data distribution patterns that can be used on an arbitrary number of nodes.

An overview of the history and main challenges of dense linear algebra software implementation is presented by *Luszczek, Kurzak, and Dongarra* [Luszczek et al. 2014]. CPU/GPU hybridization is one of those challenges. Several libraries provide dense linear algebra algorithms for BLAS and LAPACK [Demmel 1989] routines. Several works [Igual et al. 2012, Buttari et al. 2009, Tomov et al. 2010, Agullo et al. 2012] assume matrix representation with *tile data layout*. Tile algorithms create tasks that operate on contiguous memory tiles to reduce cache penalty and increase performance. However, this representation comes at the price of rigidity in the further decomposition of tiles that could not be made without copy or other matrix representation as in PaRSEC [Wu et al. 2015]. *Gautier and Lima* [Gautier and Lima 2020, Gautier and Lima 2021] showed that the execution of BLAS-3 kernels on multi-GPU systems can be improved by allowing asynchronous function calls to compose BLAS kernels and providing an explicit operator that decreases data movements on the composition of BLAS kernels to avoid unnecessary synchronizations. Performance can be further improved if an optimistic heuristic for topology-aware device-to-device data transfer is used. Their BLAS-3 tiled algorithms were based on PLASMA [Buttari et al. 2009] and Chameleon, replacing tile data layout with the LAPACK matrix representation, which allows dynamic and recursive sub-partitions.

Recent works explored the use of Cloud Computing platforms and their offer of recent hardware to improve dense linear algebra applications. *Thomas and Kumar* [Thomas and Kumar 2018] analyzed the major scalable distributed systems that provide high-level bulk linear algebra (LA) primitives for machine learning application developers (MADlib, ML-lib, SystemML, ScaLAPACK, SciDB, and TensorFlow). They evaluated the scalability, efficiency, and effectiveness of such systems on the Cloud using servers with different memory and CPU profiles but without using any type of accelerator (including GPUs). *Shankar et al.* [Shankar et al. 2020] proposed a serverless-based approach and showed that the disaggregated serverless computing model could be used for computationally intensive programs with complex communication routines such as dense linear algebra. *Lewis et al.* [Lewis et al. 2022] showed that modern hardware from cloud providers such as Google's TPUs can efficiently perform distributed dense linear algebra at a massive scale. The use of Cloud Computing to run scientific applications depending on dense linear algebra was also studied as an opportunity to achieve energy efficiency [Astsatryan et al. 2017, Chen et al. 2016].

6. Conclusion

As computer resources become increasingly accessible through the emergence of cloud computing as a tool for HPC, the utilization of task-based programming interfaces is also on the rise. HPC developers and users often seek runtime systems, tools (such as StarPU and Chameleon), and libraries to facilitate development in heterogeneous computing environments. In contrast to on-premise HPC clusters, cloud environments offer the flexibility to select diverse computing configurations and incur different monetary costs.

This paper describes and analyzes the use of Cloud Computing for solving linear algebra-based problems. To this end, we created two distinct computing environments using AWS EC2 instances and assessed the performance and monetary cost of these solutions using a standard linear algebra benchmark. Our experimental results showed that a single high-powered (yet costly) VM instance equipped with 4 NVIDIA GPUs outperforms a cluster configuration consisting of five less potent (but more economical) instances, each equipped with a single NVIDIA GPU in both performance and monetary costs in most cases due to the overhead associated with parallelization and communication between cluster nodes.

In future work, we intend to consider communication metrics and execution traces to better understand the bottenecks of running the benchmarks in different cluster configurations. Moreover, we intend to evaluate other VM configurations on AWS and to assess whether our conclusions hold true across different cloud providers. We also intend to evaluate the use of AWS *spot* and *burstable* instances, which have different pricing models than on-demand instances. In particular, the monetary costs of running communication-bound parallel applications on *burstable* instances could be lower than on on-demand instances because the CPU utilization is impacted by the communication overhead.

7. Acknowledgments

This work was partially funded by the National Council for Scientific and Technological Development (CNPq) and Amazon Web Services (AWS) through the CNPq/AWS call N° 64/2022 (Cloud Credits for Research) and grants #2019/26702-8, #2021/06867-2, #2023/00811-0, and #2024/01115-0, São Paulo Research Foundation (FAPESP).

References

Agullo, E., Augonnet, C., Dongarra, J., Faverge, M., Ltaief, H., Thibault, S., and Tomov, S. (2011). QR factorization on a multicore node enhanced with multiple GPU accelerators. In *IEEE International Parallel & Distributed Processing Symposium*, pages 932–943. IEEE.

Agullo, E., Augonnet, C., Dongarra, J., Ltaief, H., Namyst, R., Thibault, S., and Tomov, S. (2012). A hybridization methodology for high-performance linear algebra software for GPUs. In *GPU Computing Gems Jade Edition*, pages 473–484. Elsevier.

Astsatryan, H., Narsisian, W., and Costa, G. D. (2017). SaaS for energy efficient utilization of HPC resources of linear algebra calculations. *Scalable Computing: Practice and Experience*, 18(2):145–150.

Augonnet, C., Thibault, S., Namyst, R., and Wacrenier, P. (2010). StarPU: a unified platform for task scheduling on heterogeneous multicore architectures. *Concurrency and Computation: Practice and Experience*, 23(2):187–198.

Beaumont, O., Collin, J.-A., Eyraud-Dubois, L., and Vérité, M. (2023). Data distribution schemes for dense linear algebra factorizations on any number of nodes. In *IEEE International Parallel and Distributed Processing Symposium*, pages 390–401. IEEE.

Bosilca, G., Bouteiller, A., Danalis, A., Faverge, M., Haidar, A., Herault, T., Kurzak, J., Langou, J., Lemarinier, P., Ltaief, H., et al. (2011). Flexible development of dense linear algebra algorithms on massively parallel architectures with DPLASMA. In *IEEE International Symposium on Parallel and Distributed Processing Workshops and Phd Forum*, pages 1432–1441. IEEE.

Buttari, A., Langou, J., Kurzak, J., and Dongarra, J. (2009). A class of parallel tiled linear algebra algorithms for multicore architectures. *Parallel Computing*, 35(1):38–53.

Chen, J., Tan, L., Wu, P., Tao, D., Li, H., Liang, X., Li, S., Ge, R., Bhuyan, L., and Chen, Z. (2016). GreenLA: green linear algebra software for GPU-accelerated heterogeneous computing. In *International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 667–677. IEEE.

Choi, J., Dongarra, J. J., Ostrouchov, L. S., Petitet, A. P., Walker, D. W., and Whaley, R. C. (1996). Design and implementation of the ScaLAPACK LU, QR, and Cholesky factorization routines. *Scientific Programming*, 5(3):173–184.

Choi, J., Dongarra, J. J., Pozo, R., and Walker, D. W. (1992). ScaLAPACK: A scalable linear algebra library for distributed memory concurrent computers. In *Symposium on the Frontiers of Massively Parallel Computation*, pages 120–121. IEEE Computer Society.

Demmel, J. (1989). LAPACK: A portable linear algebra library for supercomputers. In *IEEE Control Systems Society Workshop on Computer-Aided Control System Design*, pages 1–7. IEEE.

Gamblin, T., LeGendre, M., Collette, M. R., Lee, G. L., Moody, A., de Supinski, B. R., and Futral, S. (2015). The Spack package manager: bringing order to HPC software chaos. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–12.

Garcia Pinto, V., Mello Schnorr, L., Stanisic, L., Legrand, A., Thibault, S., and Danjean, V. (2018). A visual performance analysis framework for task-based parallel applications running on hybrid clusters. *Concurrency and Computation: Practice and Experience*, 30(18):e4472.

Gautier, T. and Lima, J. V. F. (2020). XKBlas: a high performance implementation of BLAS-3 kernels on multi-GPU server. In *Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP)*, pages 1–8.

Gautier, T. and Lima, J. V. F. (2021). Evaluation of two topology-aware heuristics on level- 3 blas library for multi-gpu platforms. In *SC Workshops Supplementary Proceedings (SCWS)*, pages 12–22.

Igual, F. D., Chan, E., Quintana-Orti, E. S., Quintana-Orti, G., van de Geijn, R. A., and Zee, F. G. V. (2012). The FLAME approach: From dense linear algebra algorithms to high-performance multi-accelerator implementations. *Journal of Parallel and Distributed Computing*, 72(9):1134–1143. Accelerators for High-Performance Computing.

Lewis, A. G., Beall, J., Ganahl, M., Hauru, M., Mallick, S. B., and Vidal, G. (2022). Large-scale distributed linear algebra with tensor processing units. *Proceedings of the National Academy of Sciences*, 119(33):e2122762119.

Luszczek, P., Kurzak, J., and Dongarra, J. (2014). Looking back at dense linear algebra software. *Journal of Parallel and Distributed Computing*, 74(7):2548–2560.

Munhoz, V., Bonfils, A., Castro, M., and Mendizabal, O. (2023). A performance comparison of HPC workloads on traditional and cloud-based HPC clusters. In *Workshop on Cloud Computing - IEEE International Symposium on Computer Architecture and High Performance Computing Workshops*, pages 108–114, Porto Alegre, Brazil. IEEE Computer Society.

Munhoz, V. and Castro, M. (2024). Enabling the execution of HPC applications on public clouds with HPC@Cloud toolkit. *Concurrency and Computation: Practice and Experience*, 36(8):e7976.

Munhoz, V., Castro, M., and Mendizabal, O. (2022). Strategies for fault-tolerant tightly-coupled HPC workloads running on low-budget spot cloud infrastructures. In *International Symposium on Computer Architecture and High Performance Computing*, pages 263–272. IEEE Computer Society.

Netto, M. A. S., Calheiros, R. N., Rodrigues, E. R., Cunha, R. L. F., and Buyya, R. (2018). HPC cloud for scientific and business applications: Taxonomy, vision, and research challenges. *ACM Comput. Surv.*, 51(1).

Poulson, J., Marker, B., Van de Geijn, R. A., Hammond, J. R., and Romero, N. A. (2013). Elemental: A new framework for distributed memory dense matrix computations. *ACM Transactions on Mathematical Software*, 39(2):1–24.

Shankar, V., Krauth, K., Vodrahalli, K., Pu, Q., Recht, B., Stoica, I., Ragan-Kelley, J., Jonas, E., and Venkataraman, S. (2020). Serverless linear algebra. In *ACM Symposium on Cloud Computing*, pages 281–295.

Thomas, A. and Kumar, A. (2018). A comparative evaluation of systems for scalable linear algebra-based analytics. *VLDB Endowment*, 11(13):2168–2182.

Tomov, S., Dongarra, J., and Baboulin, M. (2010). Towards dense linear algebra for hybrid GPU accelerated manycore systems. *Parallel Computing*, 36(5-6):232–240.

Wu, W., Bouteiller, A., Bosilca, G., Faverge, M., and Dongarra, J. (2015). Hierarchical DAG scheduling for hybrid distributed systems. In *IEEE International Parallel and Distributed Processing Symposium*, pages 156–165. IEEE.