UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
GRUPO DE PROCESSAMENTO PARALELO E DISTRIBUÍDO

PROCEEDINGS

XVIII WORKSHOP DE PROCESSAMENTO PARALELO E DISTRIBUÍDO

WSPPD 2020

18 DE SETEMBRO DE 2020
INSTITUTO DE INFORMÁTICA
PORTO ALEGRE, RS
ISSN: 2175-6848

UFRGS
UNIVERSIDADE FEDERAL
DO RIO GRANDE DO SUL

GPPD

.inf
INSTITUTO
DE INFORMÁTICA
UFRGS

List of Sessions

Exploring I/O Request Sizes of HPC Applications on a Supercomputer

Gessica Azevedo
Institute of Informatics
Federal University of Rio Grande do Sul
Porto Alegre, Brazil
gessica.azevedo@inf.ufrgs.br

Jean Luca Bez
Institute of Informatics
Federal University of Rio Grande do Sul
Porto Alegre, Brazil
jean.bez@inf.ufrgs.br

Pablo Pavan
Institute of Informatics
Federal University of Rio Grande do Sul
Porto Alegre, Brazil
pablo.pavan@inf.ufrgs.br

Francieli Boito
LaBRI, Université de Bordeaux
Inria, CNRS, Bordeaux-INP
Bordeaux, France
francieli.zanon-boito@u-bordeaux.fr

Philippe Navaux
Institute of Informatics
Federal University of Rio Grande do Sul
Porto Alegre, Brazil
navaux@inf.ufrgs.br

***Abstract*—This study seeks to identify the most common input and output (I/O) request sizes used by HPC applications in supercomputers. We use data from the entire year of 2012 of characterization with the Darshan I/O profiling tool, in the Intrepid Blue Gene/P, to identify access patterns and request sizes. We contribute so that new optimization techniques can be evaluated considering these request sizes found in these environments. In general, we can conclude that of the six patterns chosen as a sample, five of them keep the trend of the average size of requests between 1 KB and 4 MB. Only one pattern stands out for its high variability for medium sizes, the MPI-IO write to a shared-file, due to optimizations provided by this interface.**

***Index Terms*—high performance computing, I/O, access pattern, request size, supercomputer**

I. Introduction

Scientific applications that run on high-performance computing systems (HPC) require high storage capacity and efficient data access. Parallel file systems (PFS) work by providing an abstraction of the storage system for these applications, which handle large amounts of data. In such a scenario, hundreds of computing nodes might need to access the shared storage system simultaneously. However, depending on how applications make their I/O requests, i.e., their access pattern, performance may be impaired. For instance, issuing small requests to the PFS might not offset the cost of accessing the remote storage system [1].

To apply optimization techniques to these applications, testing these techniques with benchmarks (such as MPI-IO Test [2], IOR[1], Ifer [3]) is essential but, we first need to understand the application's I/O behavior. Tools such as Darshan [4], developed at the Argonne Leadership Computing Facility (ALCF), provide characterization by creating application profiles. In order to identify the most common request sizes observed when issuing requests for the access patterns of applications on a supercomputer, this study analyzed Darshan data in Intrepid Blue Gene/P[2]. This paper's main contribution is to provide insights into what request sizes should be used when testing new optimization techniques, as those sizes will represent the sizes that HPC applications are issuing.

The remainder of this paper is organized as follows. Information on the data used in this study and its methodology are detailed in Section II. The results and analyzes are presented in Section III. Section IV discusses related work. Finally, in Section V, we conclude this paper and discuss future work.

II. Methodology

Between 2010 and 2013, ALCF collected execution data from various scientific applications using the I/O characterization tool called Darshan, in the Intrepid Blue Gene/P, ranked 23 in the November 2011 Top500 list. Darshan intercepts the flow of I/O functions and records a collection of statistics for each file that is opened [5]. The collected information allows identifying sizes and access patterns, operations, and time spent on I/O operations.

This information was used in a previous work [6], which extracted relevant data for this study. The focus of this study was on data generated in 2012 by the Darshan 2.0, resulting in $91,603$ jobs. The application coverage rate ranged from 20% to 80% per week [5], as Darshan only instrumented applications that successfully called `MPI_Init()` and `MPI_Finalize()`. However, this does not prevent applications from using POSIX, as both of these functions are used only to group information about the application's execution. We group the information by month and access patterns to identify the most common request size for each pattern and their behavior throughout the year. For this, we observe the minimum, median, maximum values, and quartiles.

Darshan continues to be used to collect information transparently, however, it is not common for these data to be public,

[1]https://github.com/hpc/ior

[2]https://www.top500.org/system/176322/

TABLE I
ACCESS PATTERNS REQUEST SIZES - SIX PATTERNS THAT MOST OCCURRED

	Access Patterns	Min. (Bytes)	Q1 (Bytes)	Median (KB)	Average (KB)	Q3 (KB)	Max. (MB)
A	POSIX, Write, Sequential Unique-file	1	99	15,7	3120,3	135	128
B	POSIX, Write, Consecutive Unique-file	1	4	0,003	0,019	0,003	16
C	POSIX, Read, Shared-file	1	160	0,27	93,6	64	256
D	POSIX, Read, Consecutive Unique-file	1	4096	4	12,6	4	16
E	POSIX, Write, Unique-file	1	2184	35,1	580,3	71,8	256
F	MPI-IO, Write, Shared-file	4	4194304	4096	6168,3	8192	122

for security and privacy reasons, as they involve information about users who performed the applications. It is not expensive to collect this profiling information, which makes several HPC centers transparently collect them in their supercomputers. Nonetheless, large updated data sets are often not public.

The 22 access patterns (described in Table II) observed throughout the year of 2012 can be classified into:

- Operation: write or read;
- File layout: single file or shared file;
- Spatiality:
  - Sequential: the `offset` does not have to be adjacent, but bigger than the previous one;
  - Consecutive: where the next `offset` to be accessed is immediately adjacent to the previous one;
- Interfaces: POSIX or MPI-IO.

Using the I/O phase estimates for the 22 patterns, we selected the six patterns observed during a longer period between applications, from these 22 patterns, as a sample to perform our analysis. These are detailed in Table I.

TABLE II
ACCESS PATTERNS REQUEST SIZE - ALL PATTERNS

Access Pattern	Average (KB)	Median (KB)
POSIX, Write, Sequential, Unique-file	3120,3	15,7
POSIX, Write, Consecutive, Unique-file	0,019	0,004
POSIX, Read, Shared-file	93,6	0,27
POSIX, Read, Consecutive, Unique-file	12,6	4
POSIX, Write, Unique-file	580,3	35,1
MPI-IO, Write, Shared-file	6168,3	4096
POSIX, Read, Unique-file	27,9	0,5
POSIX, Write, Consecutive, Shared-file	1597,3	0,094
POSIX, Write, Shared-file	0,2	0,006
MPI-IO, Write, Independent, Shared-file	10,9	0,094
MPI-IO, Read, Collective, Shared-file	662,1	0,3
POSIX, Read, Consecutive, Shared-file	148,2	0,094
POSIX, Write, Sequential, Shared-file	3426,2	4096
MPI-IO, Read, Independent, Shared-file	22,1	0,5
MPI-IO, Write, Collective, Shared-file	1556,6	160,3
MPI-IO, Write, Unique-file	13393,6	4096
MPI-IO, Read, Collective, Unique-file	3119,5	511,1
MPI-IO, Write, Independent, Unique-file	0,083	0,004
POSIX, Read, Sequential, Unique-file	112,6	10
MPI-IO, Read, Independent, Unique-file	6734,7	978,5
MPI-IO, Write, Collective, Unique-file	12460,7	16384
POSIX, Read, Sequential, Shared-file	1407,8	1351,4

III. RESULTS AND ANALYSIS

In this section we present our results and analysis. Figure 1 illustrates the average request size for the six access patterns. The x-axis represents the days, and the y-axis represents the average size (in KB). We group the results by month and access patterns. The latter indicated by the letters of A-F. The description of these patterns is presented in the Table I.

For pattern A, it is observed that the average size remains between 1 KB and 3 MB, but some exceptions can reach up to 122 MB. Pattern B has a tendency for the average size up to 1 KB. However, a more significant variation between sizes was detected, mainly in the second half of the year. Some exceeded 1 MB, but most remained between 1 KB and 1 MB. In pattern C, there was also a concentration of sizes in the range of up to 1 KB, showing a variation increase in the last 3 months of the year. These are between 1 KB and 4 MB (except for a 7 MB). This behavior is also repeated for patterns D and E. The only difference is that for pattern E, the variations are around 24 MB, with some cases exceeding 73 MB and a maximum of 122 MB. The F pattern is the most distinctive, as it registered greater variability in sizes during the year.

In general, we can conclude that of these six patterns, five keep the average size of requests between 1 KB and 4 MB. The only pattern that stands out for its high variability for medium sizes is the pattern F, corresponding to the MPI-IO, Write Shared-file, and the reason for this flexibility of this pattern are the optimizations (like data sieving [7] and collective buffering [8]) that this interface provides.

IV. RELATED WORK

Carns et al. analyzed the behavior of 66 applications on the Intrepid (ALCF) [9] supercomputer over two months of 2010. The authors demonstrated that the most recurring writing request size was between 100 KiB (kibibytes) and 1 MiB (mebibytes), while for read operations, it was between 100 bytes and 1 KiB. It was noticed that few applications influenced the observed access sizes. If these applications were disregarded in the analysis, the most frequent access size would change to 100 KiB and 1 MiB, for both operations.

Although previous work also used Intrepid's I/O load data, the study used a smaller data set. On the other hand, our research investigates the I/O behavior over an entire year (2012). It considers the size observed for the different access patterns and not only by the interface or operation.

Luu et al. [10] analyzed the Darshan logs of a million jobs representing a combined total of six years of I/O behavior across three leading HPC platforms. It was demonstrated that almost a third of the jobs had an aggregated throughput with a limit of 256 MB/s. Furthermore, three-quarters of the jobs only

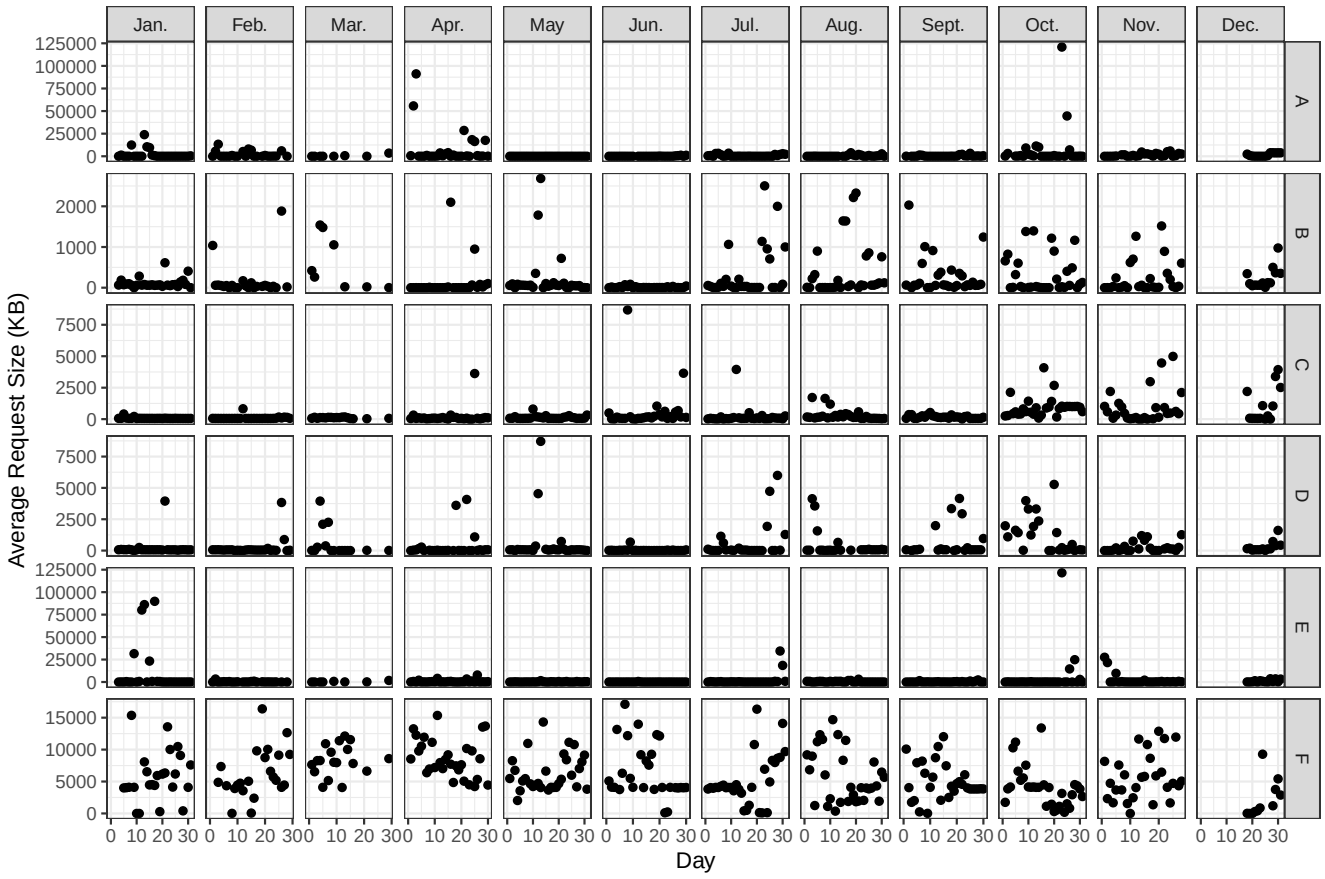

Fig. 1. Average request size (KB) over the days of a year. The y-axis is different for each pattern (row).

used POSIX to perform I/O, despite the existence of high-level parallel libraries.

Wang et al. [11] analyzed the IOR benchmark and two physics applications on a large Linux cluster, at the Lawrence Livermore National Laboratory (LLNL), with more than 800 dual-processor nodes. Each application presented only one or two typical request sizes. Large requests from several hundred KB to several MB were very common. However, small requests accounted for more than 90% of all requests, but the greatest amount of data was still transferred by large request sizes.

These two previous work also observed jobs from HPC applications, the first one focusing on the throughput of these jobs, while we focus on the request size, but in common, both could reach that most jobs used POSIX to perform I/O. The second one inspects the request sizes, same in our study, but we analyzed these sizes with the pattern associated. Besides, this work observed cases that small requests accounted for almost all requests. Our work also detected several instances with a great number of small requests.

V. Conclusion and Future Work

This study used data from an entire year of characterization with Darshan on the Intrepid Blue Gene/P supercomputer. It was possible to determine the most common I/O request sizes considering the different access patterns observed.

To evaluate new optimization techniques (initially using benchmarks such as MPI-IO Test, IOR, Ifer), it is necessary to use parameters close to reality so that the validation is consistent. In this way, identifying the most common request sizes helps the tests maintain reliability since these parameters will represent the reality of HPC applications. As future work, we intend to expand the analysis to the other patterns observed in the machine throughout the year.

Acknowledgment

The research received funding from PIBIC CNPq-UFRGS, from CAPES, grant N. 001, from CNPq and from the Petrobras project, grant N. 2016 / 00133-9. This research used resources from the *Argonne Leadership Computing Facility* at the Argonne National Laboratory, which is supported by the *Office of Science of the U.S. Department of Energy* under DE-AC02-06CH11357.

References

[1] F. Z. Boito, E. C. Inacio, J. Bez, P. O. A. Navaux, M. A. R. Dantas, and Y. Denneulin, "A Checkpoint of Research on Parallel I/O for High-Performance Computing." *In 2018 ACM Computing Surveys (ACM)*, Mar 2018.

[2] LANL, "Los alamos national lab mpi-io test, user's guide," 2006.
[3] O. Yildiz, M. Dorier, S. Ibrahim, R. Ross, and G. Antoniu, "On the Root Causes of Cross-Application I/O Interference in HPC Storage Systems," in *IPDPS 2016 - The 30th IEEE International Parallel and Distributed Processing Symposium*, Chicago, United States, May 2016. [Online]. Available: https://hal.inria.fr/hal-01270630
[4] P. Carns, R. Latham, R. Ross, S. L. K. Iskra, and K. Riley., "24/7 Characterization of petascale I/O workloads." *In 2009 IEEE International Conference on Cluster Computing and Workshops*, pp. 1–10, Aug 2009.
[5] P. Carns., "ALCF I/O Data Repository." Argonne Leadership Computing Facility, Tech. Rep., Feb 2013.
[6] P. J. Pavan, J. Bez, M. S. Serpa, F. Z. Boito, and P. O. A. Navaux, "An Unsupervised Learning Approach for I/O Behavior Characterization," *In 2019 31st International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD)*, Oct 2019.
[7] R. Thakur, W. Gropp, and E. Lusk, "Optimizing noncontiguous accesses in mpi-io," *Parallel Computing*, vol. 28, no. 1, pp. 83–105, Jan. 2002. [Online]. Available: http://dx.doi.org/10.1016/S0167-8191(01)00129-6
[8] J. M. del Rosario, R. Bordawekar, and A. Choudhary, "Improved parallel i/o via a two-phase run-time access strategy," *ACM SIGARCH Computer Architecture News*, vol. 21, no. 5, pp. 31–38, Dec. 1993. [Online]. Available: http://doi.acm.org/10.1145/165660.165667
[9] P. Carns, K. Harms, W. Allcock, C. Bacon, S. Lang, R. Latham, and R. Ross, "Understanding and improving computational science storage access through continuous characterization." *In 2011 IEEE 27th Symposium on Mass Storage Systems and Technologies*, p. 7(3):8:1–8:26, May 2011.
[10] H. Luu, M. Winslett, W. Gropp, R. Ross, P. Carns, K. Harms, M. Prabhat, S. Byna, and Y. Yao, "A multiplatform study of I/O behavior on petascale supercomputers." *In Proceedings of the 24th International Symposium on High-Performance Parallel and Distributed Computing. ACM*, p. 33–44, 2015.
[11] F.Wang, Q. Xin, B. Hong, S. A. Brandt, E. L. Miller, D. D. E. Long, and T. T. Mclarty., "File system workload analysis for large scientific computing applications," April 2004.

Experimental Comparison of Load Imbalance Metrics using Temporal Agregation for Homogeneous and Heterogeneous Platforms

Ana Luisa Veroneze Solórzano, Lucas Leandro Nesi, Lucas Mello Schnorr
Informatics Institute (PPGC/UFRGS), Porto Alegre, Brazil

***Abstract*—A desirable goal in HPC parallel applications is the even distribution of workload among computational resources. While these resources can be homogeneous or heterogeneous, we have partitioning and balancing strategies for each one. Before making decisions regarding load redistribution, it is imperative to identify and measure the load state accurately. Load Imbalance Metrics are a valuable option to characterize computational load distribution by measuring performance, time, and the degree of imbalance. However, the metrics are usually calculated for the whole application execution generating a single value to characterize how the work is distributed and unmodified to consider different resource computational capacities. We select five state-of-art metrics and present an experimental evaluation of them with two applications with well-known load imbalance problems: the Ondes3D Earthquake Simulator, running in a multi-node homogeneous environment, and the Chameleon's Cholesky Factorization in a single heterogeneous node with CPU/GPU. We considered the application's particularities, evaluating existing imbalance metrics tackling homogeneous resources, presenting new metrics tailored for heterogeneous platforms, and a detailed load balancing evaluation per application phase. Our results show that the selected metrics can bring valuable information on different imbalance scenarios, temporal integrations, and architectural environments. We present the benefits and challenges of the metrics, pointing the best options to evaluate the imbalance. Moreover, we believe that such a study can contribute to online load rebalancing decisions for application regions or phases.**

ACKNOWLEDGEMENTS

This study was financed by the "Coordenação de Aperfeiçoamento de Pessoal de Nível Superior" (CAPES) - Finance Code 001, under grant no 88887.481194/2020-00 to the first author, the National Council for Scientific and Technological Development (CNPq), under grant no 141971/2020-7 to the second author, and the projects: FAPERGS (Data Science – 19/711-6, MultiGPU 16/354-8, and GreenCloud – 16/488-9), the CNPq project 447311/2014-0, the CAPES/Brafitec EcoSud 182/15, and the CAPES/Cofecub 899/18. The experiments in this work used the PCAD infrastructure, http://gppd-hpc.inf.ufrgs.br, at INF/UFRGS.

DISCLAIMER

This abstract describes the article entitled "Evaluation of Load Imbalance Metrics for Homogeneous and Heterogeneous Platforms", from the same authors, that has been submitted in August 2020 to the Concurrency and Computation: Practice and Experience from Wiley.

Performance Characterization of a Marine CSEM Inversion Application

Jessica Imlau Dagostini, Henrique Corrêa Pereira da Silva, Vinicius Garcia Pinto,
Eduardo Simões Lopes Gastal, Lucas Mello Schnorr
Institute of Informatics, Federal University of Rio Grande do Sul – UFRGS, Porto Alegre, Brazil
{jidagostini, hcpsilva, vgpinto, eslgastal, schnorr}@inf.ufrgs.br

***Abstract*—The prospection of areas for oil exploration is an expensive and risky process. For this reason, several computational simulations are performed to estimate the probability of finding a viable petroleum reservoir before drilling it. In this paper, we characterize the performance of a parallel application based on the marine Controlled Source Electromagnetic Method. We have identified the dominant step, and after parallelizing it, we outperform the original code with a speed-up of ≈7 times. We also detect that there is room for future gains by optimizing the workload distribution.**

***Index Terms*—oil prospection, CSEM method, MPI, OpenMP**

I. Introduction

One of the main challenges in the oil industry is to correctly map areas in the ocean where there is a higher probability of oil and gas. For this mapping, it is necessary to apply a numerical method that needs to be as precise as possible to give reliable information. Physical oil exploration has a high monetary cost, usually requiring amounts in the order of millions. Having ways to do accurately computational simulations to help with this process is essential to these industries since it can reduce the incidence of unsuccessful operations, thus saving time and monetary resources.

The usage of electromagnetic survey methods in such offshore contexts now sees an increase as another way to remove doubt from the costly exploratory process. One known method to simulate and computationally investigate ocean lands is the marine Controlled Source Electromagnetic (mCSEM) [1]. This method uses maritime receptors – fixed in the ocean floor – and uses a portable source that emits electromagnetic signals that are kept from these receptors to gather data.

This electromagnetic surveying is applied to recognize the seafloor's aspects, thus giving relevant information about the explored region's conditions. From this information, it is possible to run numerical methods and apply mathematical inversion to matching subsurface model. The data processing step takes increased attention, as it is the deciding factor that guarantees efficient employment of the collected data before the visualization phase.

This study was financed by the "Coordenação de Aperfeiçoamento de Pessoal de Nível Superior" (CAPES) - Finance Code 001, the CNPq - Brazil scholarship for the 2º and 3º author, and the projects: FAPERGS MultiGPU (16/354-8) and GreenCloud (16/488-9), the CNPq project 447311/2014-0, the CAPES/Brafitec EcoSud 182/15, the CAPES/Cofecub 899/18, Petrobras (2016/00133-9, 2018/00263-5). The experiments presented in this paper used the PCAD infrastructure, http://gppd-hpc.inf.ufrgs.br, at INF/UFRGS.

As the exploration of specific areas in the seafloor can generate a considerable amount of data, it becomes necessary to use a high-performance application to inversion faster. Adopting a parallel approach to this high-performance application, we need to guarantee that such parallelization is efficient in using of all computing resources by stressing each resource at its maximum.

The present work aims to characterize and demonstrate improvements in the performance of an inversion mCSEM application for mapping possible oil and gas regions. This is a third-party real world application, coded in C language, using MPI and OpenMP for code parallelization. This paper is divided into four sections. Section II introduces a background about the problem, the application, and the data used to run the simulations. Section III presents the initial analysis of the application, alongside the solution of one problem in performance and the discovery of a new one. For the last, Section IV presents the conclusions and future work.

II. Basic Concepts

Electromagnetic surveying is a complex process, containing multiple steps – both physical and informational – before providing data to interpretation. Our characterized application performs a fine-grained electromagnetic data inversion to reconstruct a model of the underlying substrate based on a marine Controlled Source Electromagnetic surveying. This approach enables superior control over all the exploratory stages of offshore reservoirs and a fine-grained control in the interpretation phase of the exploratory process.

A. Offshore Exploration

In the era of rising green energy, the continuing economic development and population growth are expected to push the global energy needs in a way that outpaces the establishment of cleaner alternatives. To meet this demand, the oil and gas industries project to fill as much as 50% of the primary energy sources before the middle of this century [2]. However, easily accessible hydrocarbon reserves are either depleted or already developed, so these companies require faster, more precise acquisition and visualization techniques in order to explore more complex undeveloped options [3].

A particularly complex area of operations is offshore exploration. Deep under salt layers, these reservoirs often pose a challenge to traditional seismic surveys, as different fluids,

like hydrocarbons and water, prove difficult to differentiate [4]. To provide better data before the costly physical exploratory drilling operations, electromagnetic methods can be used to provide important complementary information to aid in the decision-making process of those operations.

Figure 1 diagrams the exploratory process. The operations start with a previous formal geologic knowledge of the area and its formation, which indicates the presence of traps or other characteristic rock sequences. Then, data from one or more surveys feed visualization tools that further assist the decision-making of supplementary actions, which are either more surveys or the drilling of an exploratory well, a rather expensive operation in offshore contexts. Found the existence of oil or gas, further surveys may follow to more precisely define the extension of the reservoirs before the start of the extraction operation.

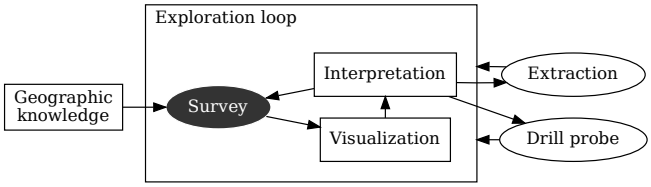

Fig. 1. The decision-making process in resevoir exploration.

B. Controlled Source Electromagnetism

Electromagnetic (EM) surveying is applied in both land and maritime contexts – though these have developed almost independently – and either actively or passively. In maritime controlled source electromagnetism (mCSEM), a towed human-made electromagnetic source emits a low-frequency continuous signal that potentially traverses water, surface, and deep into the seabed before being captured by an array of receptors either placed on the seafloor or also towed along with the source [5]. Figure 2 represents the static receivers mCSEM approach, where receptors placed at the seafloor capture reading from the boat-towed EM source. These approaches aim to collect readings of the EM signal after it crossed the underlying salt and rock sequences, altering it and, ultimately, imprinting its distinct characteristic onto the signal [6].

When crossing conductive substances of varying resistivity, the emitted wave's amplitude gets altered according to the variation on the conductive material. Such process obeys Maxwell's equations, and so could be though as the application of a direct method over an existing input: the traversal of an EM wave over the subsurface salt and rock layers [6]. In other words, the materials underground imprinted their different conductivities onto the now read EM field values. Therefore, the aim is to model back these resistivities from the read values through the inverse method, matching the readings to the most probable model of the subsurface substances that generated such readings.

However, unwanted signals contribute to the reading as well, like the wave that traveled directly to the receptor, the air-

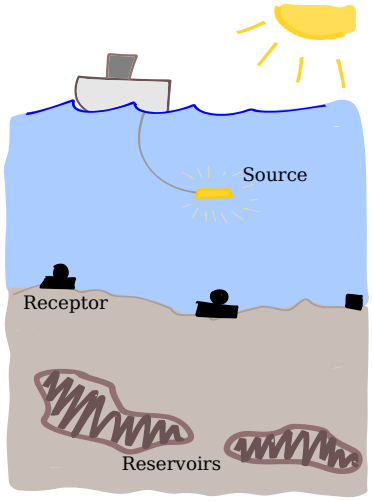

Fig. 2. Representation of the CSEM surveying process.

traversed wave (airwave), and magnetotelluric signals [4]. As to whether preprocess these readings in hopes to alleviate the influence of these signals is part of the interpretation method applied, as deepwater operations do not pick up as much airwave noise as shallow water receivers. However, noise considerably influences the sensitivity of the model to the desired ground-traversed waves.

C. CSEM Data Inversion Processing

The inversion method – as in the reconstruction of a model of resistivities that could have caused our readings – can be done with and without an initial input model. The application we characterize implements the former approach, which can prove especially useful when tied together with *a priori* geographic knowledge of the underlying substrate. This allows fine-grained control in the interpretation phase of the exploratory process, as the model-driven inversion is acutely sensitive to these initial values and so could achieve a more faithful match to the underlying rock layers' resistivities [4].

The industry's CSEM inversion methods are iterative solvers, as this is essentially an optimization problem. We generally see two phases to such algorithms: the **simulation** calculation and the **forward** step. In the former, the solver uses the current model to simulate EM signal values and then to compare the result to the read values read in the survey, checking for convergence according to a misfit delta value. The simulation requires solving of large sparse equation system and is achieved in this application using a Cholesky factorization. In the last-mentioned step, the solver forwards the model to the next iteration. This can be accomplished in several ways, like using CMP (Common Midpoint) [7], but the details of the chosen approach are part of the company's intellectual property and escape the scope of this work.

Our characterized application follows the logic outlined in Figure 3. Initially, from the highlighted input resistivity model, the application simulates synthetic data through the direct method, using the Maxwell equations that model the

EM interactions. The input survey data is then compared to the generated data and, if within a defined misfit delta, the inversion ends, as a well-matching model has been found. Otherwise, the application forwards the existing model, updating it to the next simulation iteration.

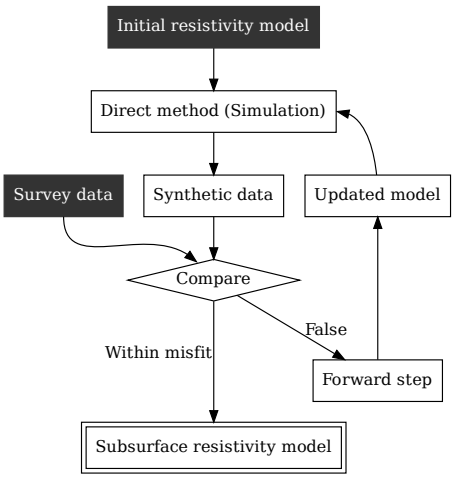

Fig. 3. Diagram representing the inversion algorithm.

III. Methodology and Results

To understand the code's behavior and provide improvements to its execution, we first characterized the application by doing a more in-depth analysis of the code. We then instrumented it by using a tracer application. By these tracing results, we were able to provide some performance gains and found other notable issues.

A. Platform Configuration

Table I shows the platform configuration of the **cei** cluster used in the executions presented at Subsections III-B, III-C, and III-D. All experiments in this work used the PCAD infrastructure, http://gppd-hpc.inf.ufrgs.br, at INF/UFRGS.

TABLE I
Configuration of machines used in the experiments

	cei
Nodes	5
Processors per Node	2
Processor	Intel Xeon 4116 2.10GHz
Cores per Processor	12
Core count	24
Memory	96 GB DDR4
Interconnection	100Mbit/s
Operating System	Debian 10
Linux kernel	4.19.0-8-amd64
OpenMPI	3.1.3
GCC	8.3.0

B. Performance Characterization

The original code was already parallel, exploring both shared and distributed memory paradigms using both OpenMP and MPI. As an initial step, we have instrumented the code using ScoreP [8] to characterize each application phase's performance. The top plot of Figure 4 shows a first visualization of the obtained execution traces of this original code. We split the main region of the application in three phases: **forward**, **cholesky**, and **correction**.

One can observe that only the first node (rank 0) execute the **cholesky** step (in blue) while all other ranks remain idle waiting in a global barrier. The **forward** (red) and **correction** (green) steps are executed in parallel by all ranks, but the last ones (**correction**) are too short to be visible in the graphic.

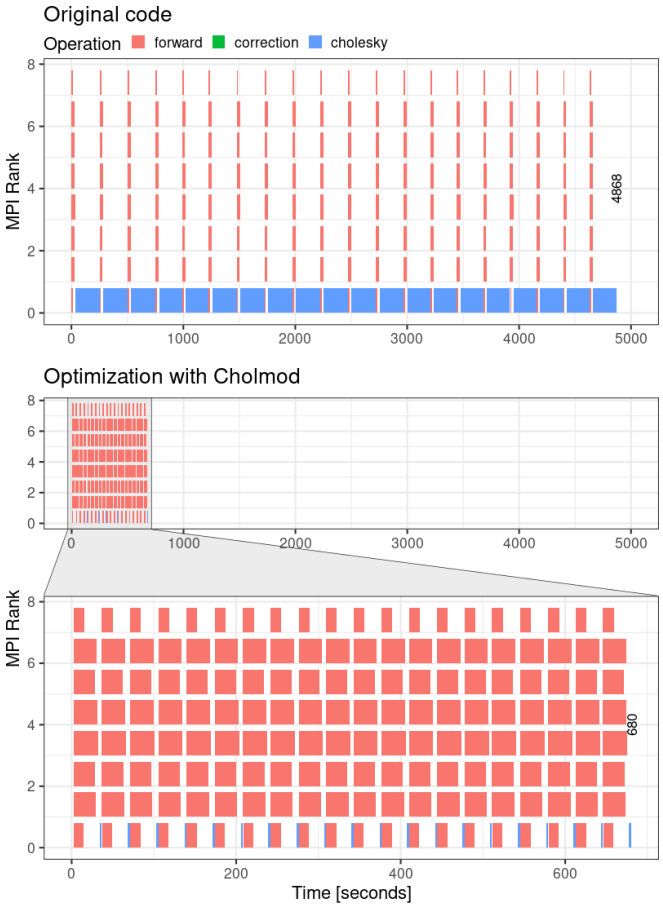

Fig. 4. Execution Trace Visualization: Original implementation (top); after Optimization with Cholmod (bottom)

C. Sparse Cholesky Solver

Standard libraries of linear algebra kernels usually follow a common API as BLAS, which makes easy switch among different implementations (e.g., Intel MKL, OpenBlas, CUBLAS). Since the original code uses a built-in implementation of the Cholesky factorization, we decided to replace it by integrating a standard one. We chose the **Cholmod** module [9] from the Suitesparse library [10], which is multithread, and proved suitable for the application and its current workload. Figure 4 shows the performance gains of this optimization, reducing the makespan from 4868 seconds to 680 seconds, i.e., a speed-up of ≈7 times. The **cholesky** steps are no

more time dominant, with **forward** steps being now the most representatives (see zoom area at the bottom of Figure 4).

D. Load Imbalance Issues

Even with the performance gains obtained through the Cholesky parallelization, we still could notice some load problems. Taking a closer look at Figure 4, we note that the time spent in each MPI Rank during the **forward** execution is different among them. Thus, we are faced with a new load imbalance in the application execution.

The **forward** region is responsible for evolving the model by using CMPs information. Each CMP has a different amount of combinations of three data: the receptor, the source, and the frequency. These combinations define the valid values used in the model evolution and, as we could statistically prove, are directly responsible for the time spent in the **forward** phase.

The distribution algorithm used by the application considers sharing the total CMPs among MPI Ranks. However, each CMP has a different number of combinations to process, which causes the different workload to each rank. Figure 5 shows the direct relation between process time and quantity of the combinations by a rank.

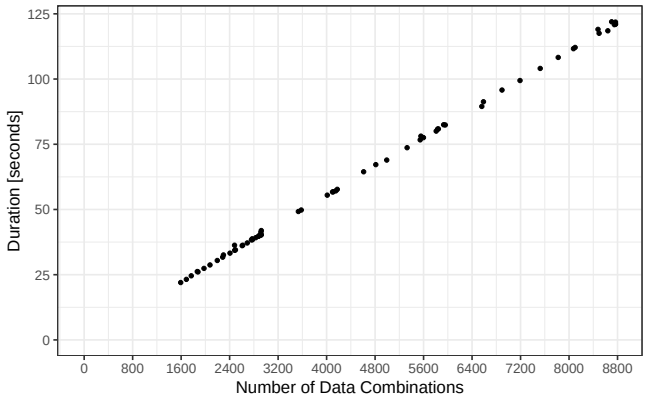

Fig. 5. Direct relation between combinations and workload by rank.

Therefore, we apply ANOVA (ANalysis Of VAriance) [11] to verify the direct association between the number of combinations allocated among MPI ranks and its execution time. The results confirm our hypothesis, with a statistical confidence of 99.7%. By them, we aim to find a better load distribution between the ranks, to balance their workload and then reduce the execution time.

As each combination of three information is directly related to a CMP, we aim to perform this distribution by allocating different CMPs to each rank without splitting its data combinations in different ranks. Thus, we need to allocate the CMPs in a combination that balances how many groups of three combinations we have in each rank. Some distributions algorithms are already in our roadmaps, such as the Round Robin and Greedy algorithms.

IV. Conclusion and Future Work

In this paper, we have characterized the performance of an application for oil prospection. This application uses the marine Controlled Source Electromagnetic Method. Even if the original code is already parallel using MPI and OpenMP, we have identified a sequential step that executes a sparse **cholesky** factorization. After replacing it by a parallel implementation provided by the **Cholmod** module, the makespan reduces by a factor of $\approx$7.

Our optimization on the **cholesky** step has highlighted another performance issue, the unbalanced workload among the MPI ranks. As future work, we plan to explore different load balancing algorithms to achieve more performance gains and to investigate different data decomposition (CMPs). We also plan to extend our analysis with new artificially created workloads in order to evaluate the scalability of the envisaged solutions.

References

[1] V. C. T. de Souza, "Modelagem Numérica de Dados mCSEM 3D Usando Computação Paralela," Ph.D. dissertation, Universidade Federal do Pará, 2007. [Online]. Available: http://repositorio.ufpa.br:8080/jspui/handle/2011/5673

[2] Z. Liu, "Chapter 4 - supply and demand of global energy and electricity," in *Global Energy Interconnection*, Z. Liu, Ed. Boston: Academic Press, 2015, pp. 101 – 182. [Online]. Available: https://doi.org/10.1016/B978-0-12-804405-6.00004-X

[3] Z. Ningning, W. Qing, W. Jianjun, H. Lianhua, L. Haowu, and L. Qian, "Characteristics of oil and gas discoveries in recent 20 years and future exploration in the world," *China Petroleum Exploration*, vol. 23, no. 1, p. 44, 2018. [Online]. Available: http://www.cped.cn/EN/10.3969/j.issn.1672-7703.2018.01.005

[4] A. Ziolkowski and D. Wright, "The potential of the controlled source electromagnetic method: A powerful tool for hydrocarbon exploration, appraisal, and reservoir characterization," *IEEE Signal Processing Magazine - IEEE SIGNAL PROCESS MAG*, vol. 29, pp. 36–52, 07 2012. [Online]. Available: https://doi.org/10.1109/MSP.2012.2192529

[5] M. Wilt and D. Alumbaugh, "Electromagnetic methods for development and production: State of the art," *The Leading Edge*, vol. 17, no. 4, pp. 487–487, 1998. [Online]. Available: https://doi.org/10.1190/1.1437997

[6] S. Constable and L. J. Srnka, "An introduction to marine controlled-source electromagnetic methods for hydrocarbon exploration," *GEOPHYSICS*, vol. 72, no. 2, pp. WA3–WA12, 2007. [Online]. Available: https://doi.org/10.1190/1.2432483

[7] J. L. Silva Crepaldi, M. P. Pereira Buonora, and I. Figueiredo, "Fast marine CSEM inversion in the CMP domain using analytical derivatives," *Geophysics*, vol. 76, no. 5, pp. F303–F313, sep 2011. [Online]. Available: https://doi.org/10.1190/geo2010-0237.1

[8] A. Knüpfer, C. Rössel, D. a. Mey, S. Biersdorff, K. Diethelm, D. Eschweiler, M. Geimer, M. Gerndt, D. Lorenz, A. Malony, and et al., "Score-p: A joint performance measurement run-time infrastructure for periscope, scalasca, tau, and vampir," *Tools for High Performance Computing 2011*, p. 79–91, 2012. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-31476-6_7

[9] T. A. Davis, "User guide for cholmod: a sparse cholesky factorization and modification package," *Department of Computer and Information Science and Engineering, University of Florida, Gainesville, FL, USA*, 2008.

[10] S. Kolodziej, M. Aznaveh, M. Bullock, J. David, T. Davis, M. Henderson, Y. Hu, and R. Sandstrom, "The suitesparse matrix collection website interface," *Journal of Open Source Software*, vol. 4, no. 35, p. 1244, Mar 2019. [Online]. Available: http://dx.doi.org/10.21105/joss.01244

[11] E. George, J. S. Hunter, W. G. Hunter, R. Bins, K. Kirlin IV, and D. Carroll, *Statistics for experimenters: design, innovation, and discovery*. Wiley New York, 2005.

Performance Analysis of Convolutional Neural Networks on GPU, TPU and CPU Platforms Applied to Facial Expression Recognition

1st Leandro Perius Heck
Regional University of Northwest of Rio Grande do Sul (UNIJUI)
Santa Rosa, RS – Brazil
leandro.h@sou.unijui.edu.br

2st Cristiano Alex Künas
Regional University of Northwest of Rio Grande do Sul (UNIJUI)
Santa Rosa, RS – Brazil
cristiano.kunas@sou.unijui.edu.br

3st Edson Luiz Padoin
Regional University of Northwest of Rio Grande do Sul (UNIJUI)
Ijuí, RS – Brazil
padoin@unijui.edu.br

***Abstract*—Considering the growing interest in the field of human-computer interaction and that this iteration has become something more and more natural and social, along with the increase in computational capacity provided by Central Process Unit (CPUs), Graphics Processing Unit (GPUs) and Tensor Processing Units (TPUs), which are application-specific integrated circuits developed to accelerate machine learning workloads. Areas such as recognition of emotions have proven to be of great interest and relevance to the scientific community. However, even with several works performed, detecting and recognizing emotions computationally and with the same ease that humans recognize is still a relevant problem to be explored. In order to explore this theme, this work has adopted the use of Convolutional Neural Networks (CNN) in the recognition of emotions in humans based on facial expressions. The results showed that with the training of an Artificial Neural Network (ANN) in GPUs, it was possible to reduce computational time by up to 90% and increase accuracy to 65%**

***Index Terms*—Artificial Intelligence, GPU, Artificial Neural Networks, Convolutional Neural Networks.**

I. Introduction

Nowadays there is a growing interest in improving the interaction between humans and computers. For an effective intelligent human-computer interface to be achieved, the computer must naturally relate to the user, similar to the way humans interact [1]. One of the ways to find and promote this kind of integration is with the help of Affective Computing, Artificial Intelligence, Machine Learning and Deep Learning.

Through these approaches it is possible to use computers to recognize, model and express emotions and how they can respond to them. According to Leão et al. [1], Making a machine recognize, model and express emotions is not a simple task. When human beings interact with each other, much of this interaction is based on verbal language and the use of body language through gestures and facial expressions that carry and transmit the emotions of the interlocutors.

In recent decades the scientific community has been taking a growing interest in the recognition of emotions. There are several ways to express human emotions, and these have been studied over the years, and the following sources of data have been explored, such as texts, sending emoticons, voice and facial expressions.

In addition to facial expressions playing an important role in the relationship between people, that it, it provides information that is relevant to an individual's emotions and intentions during dialogue or communication. The recognition of such emotions allows us to create numerous applications of human-computer interaction and has attracted the attention of the scientific community.

As facial recognition technology progresses, new ways of using it in our daily lives also emerge, such as to make payments, see who's watching in class, wake up a sleepy driver, personalized menus, and targeted marketing to the trade. Imagine a store being able to organize all its products, according to the reaction of its customers. This way, you can leave the products that most attract the attention of individuals with a greater prominence in the store or window, thus calling even more customers to the store.

This work aims to realize the recognition of the 6 emotions considered as basic by Ekman [2], which are happiness, sadness, fear, surprise, anger, disgust, plus the neutral emotion. For this task, images of human facial expressions will be used, and for the extraction of the characteristics, computational vision and ANN techniques will be used. But the main objective is to perform the analysis and comparison of CNN times on the different CPU, GPU and TPU architectures. Besides observing the performance of ANN in the different architectures.

II. Related Work

In Bartlett's [3] work a study is presented with the objective of automatically locating faces in a video stream and encoding visual expression in a dynamic way. The authors discuss that face-to-face communication is a real-time operation with a time scale of 40 milliseconds. The system is able to detect seven expressions: neutral, anger, disgust, fear, joy, sadness and surprise, with a differential from other works because it operates in real time. This system has been trained and tested using the CohnKanade AU-Coded Expression Database. This

database contains the facial record of 210 adults between 18 and 50 years of age, 69% female and 31% male, and 81% Euro-American, 13% African-American and 6% other ethnic groups. The experiments performed compared the performance of the automatic detection approach recognition with the manual detection approach, finding no significant difference between them. The system showed a level of accuracy of 93% of recognition, in the selection of one of the 7 options of facial expressions.

The work developed by Tang and Huang [4], aims to recognize the six basic and universal emotions through facial expressions using 3D geometry. This approach extracts characteristics that are invariant under the effect of illumination or posture, characteristics that, the authors consider as obstacles for facial recognition in 2D images. In this work, the BU-3DFE database was used as a training and testing basis. This database is composed of 100 individuals, 60% female and 40% male, with ethnic variety, including white, black, East Asian, Middle East Asian, Latin American among others. For this work it was found that the approach produced an absolute increase of 3.5% in the average recognition rate. This approach produced an average accuracy of 87.1%, the highest rate being 99.2% for the recognition of the facial expression of surprise.

In the work developed by Amin et al. [5], an ANN was elaborated that has the purpose of recognizing emotions through facial expressions using the technique of deep learning. According to the author, the use of convolutional neural networks, in the approach to the identification of emotions, reaches an average precision of 60%. The Facial Expression Recognition 2013 (FER-2013) database was used for the development of the work. This database was developed by Pierre Luc Carrier and Aaron Courville, and has a total of 35887 images, with dimensions of 48x48 pixels. The samples have the six facial expressions of Ekman, added from the neutral expression. The samples in this database are distributed in: 7215 images of joy; 436 of disgust; 4097 of fear; 4965 of neutral; 3995 of anger; 3171 of surprise; and 4830 of sadness. The work presented good results, reaching an average accuracy of 61.05% for the classification of the seven emotions. Analyzing the results, the authors found that the emotion of joy has the highest recognition rate. However, the model cannot perfectly classify the emotions of fear and sadness.

Different from the works presented, our proposal seeks to develop an CNN that recognizes facial expressions from images, and identifies the emotion present in the image. Besides proposing the development of an CNN, the performance and training time in the CPU, GPU and TPU architectures will be analyzed. To obtain a satisfactory result in the shortest possible time.

III. Implementation of CNN

For the implementation of the proposed CNN the development environment Google Colab and the programming language Python were used. Google Colab is a free cloud service hosted by Google for Machine Learning and IA, with free GPU accelerators, pre-installed libraries, built based on Jupyter Notebook, supports bash commands and stores the notebooks in the Drive itself. The main libraries used are TensorFlow 2.0 and Keras, which are focused on deep machine learning. Keras is the most used framework in the area for its easiness. In TensorFlow 2.0, Keras has been ”embedded” into TensorFlow through the module tf.keras. CNN also uses the tools: OpenCV, Scikit-Learn, NumPy, Pandas, Matplotlib.

The database used for this work was *FER-2013* Facial Expression Recognition 2013). This database brings together an open-source data set created by Pierre- Luc Carrier and Aaron Couville, then publicly shared for a Kaggle competition[1]. This data set consists of 35,887 gray tone facial images subdivided into seven emotions, with a dimension of 48 x 48 pixels. The faces were registered automatically so that the face is more or less centered and occupies approximately the same amount of space in each image. The task is to categorize each face based on the emotion shown from the facial expression into one of seven categories, which contain the description of the emotion and its descriptor already defined by the database. The file *FER2013.csv* has two columns that are ”emotions” and ”pixels”. The ”emotion” column contains a numerical code ranging from zero to six. The ”pixels” column contains a sequence of numbers that represents the grayscale values for each pixel of the image.

To start the development of CNN it is necessary to apply some transformations in the information in the file *FER2013.csv*. When reading the *csv* file, the information is in string format, it is necessary to convert it to an array, for this is used the function *tolist()*, which converts the database data to a list. In the elaboration, it was opted to implement a convolutional neural network, which has been applied successfully in the processing and analysis of digital images [6]. For this reason there are basically four steps to be followed for its development [7]:

i) **Step 1 Convolution Operator:** This step works as filters that see small frames and goes through the whole image capturing its most relevant features. For example, on a 48x48 image and a filter that covers a 3x3 area of the image with 1 jump movement, the filter will scroll the entire image. In the 3x3 filter it is taken point by point and multiplication is performed by a feature detector, this one defined by the library that is used, forming at the end a feature map or feature map.

ii) **Step 2 Pooling:** In this step has the function of simplifying the information of the previous layer, in this case, the map of characteristics. As in the convolution, an area unit is defined, usually using a 2x2 matrix, to pass through the entire output of the previous layer. The unit is responsible for summarizing the information of that area in a single value. To select this element it is necessary to choose the way the summarization will be made. The most used method is MaxPooling, which returns the largest number of the unit and passes this value to the output. This data summarization serves

[1]Kaggle is a web platform for Data Science competitions. It is currently owned by Google

to reduce the amount of weights to be learned by the Neural Network, and also avoid overfitting.

iii) **Step 3 Flattening:** This step receives the matrix created in the Pooling step as input, and basically operates a transformation in the image matrix, changing its format to an array, that is, it converts the matrix to a characteristics vector.

iv) **Step 4 Neural Network:** At this stage it receives the data from the Flattening layer and submits the characteristics vector as input for Neural Network training, which is a computational model based on the human central nervous system. In this way, you will be able to recognize patterns in a large mass of data in order to classify them in some category.

IV. Results

For the CNN tests, 10 repetitions were performed in order to obtain the average execution times and Speedup, both in CPU, TPU and the four models of GPUs available in Google Colab, which provides the NVIDIA Tesla K80, T4, P4 and P100 for free.

The best Speedup achieved by CNN compared to CPU is shown in Figure 1, where the Tesla P100 obtained the best result. The Speedup of the algorithm running on GPU shows a 10.71 gain over the CPU. Thus reducing the jumping runtime from 3950.44 seconds to 368.81 seconds, presenting a gain of 90.66%. The result of the Speedup of the other GPUs was also satisfactory. The Speedup of the algorithm running on the Tesla K80 GPU showed a gain of 3.51 times over the CPU. Thus reducing the runtime from 3950.44 seconds to 1125.37 seconds, obtaining a gain of 71.21%. The Tesla P4 showed a 7.16 gain over the CPU. Decreasing the runtime from 3950.44 seconds to 551.60 seconds, achieving a 86.04% gain. And finally Tesla T4 pointed to a 9.45 gain compared to the CPU. Limiting the runtime from 3950.44 seconds to 417.98, earning 89.42%. The comparison of TPU use with the CPU had a gain of 2.04 times, decreasing the execution time from 3950.44 seconds to 1935.49 seconds, presenting a gain of 50.01%.

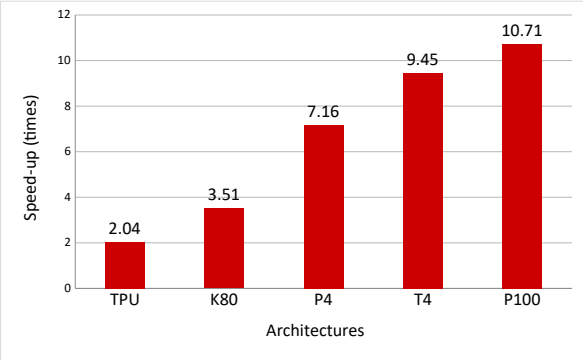

Fig. 1. Speedup of GPU and TPU architectures compared to CPU.

Comparisons were also made between GPUs, for analysis of the gain obtained. The comparison of the Tesla P4 with the Tesla K80, showed a gain of 2.04 times, reducing the execution time from 3950.44 seconds to 551.60 seconds, showing a gain of 86.04%. Tesla T4 was compared to P4 and K80. Tesla T4 earned 1.31 times compared to P4, decreasing the time from 551.60 seconds to 417.98 seconds, making a gain of 24.23%. Compared to K80, T4 earned 2.69 times, limiting the runtime from 3950.44 seconds to 417.98 seconds, earning 89.42%. Finally, the analysis was performed comparing the Tesla P100 with the other GPUs. In the comparison with the T4, a 1.13 times gain was obtained, reducing the execution time from 417.98 seconds to 368.81 seconds, reaching a gain of 11.77%. The conference with P4, came to a 1.49 times gain, reducing the time from 551.60 seconds to 368.81 seconds, coming to a 33.14 times gain. In the confrontation with K80 it reached a 3.05 times gain, decreasing the execution time of 1125.37 seconds to 368.81, presenting a gain of 67.23%.

The comparison of Speedup between GPUs and the Google Colab TPU, illustrated in Figure 2, was also performed. The Speedup of the algorithm running on the Tesla K80 GPU showed a 1.71 times gain over the TPU, reducing the runtime from 1935.49 seconds to 1125.37 seconds, obtaining a 41.86% gain. The Tesla P4 showed a 3.50 gain over the TPU, decreasing the execution time from 1935.49 seconds to 551.60 seconds, reaching a gain of 51.51%. Tesla T4 pointed to a 4.63 gain compared to TPU, limiting the execution time from 1935.49 seconds to 417.98, with a gain of 78.41%. And finally the Tesla P100 GPU showed a 5.24 gain over the TPU, reducing the runtime from 1935.49 seconds to 368.81 seconds, obtaining a 80.95% gain.

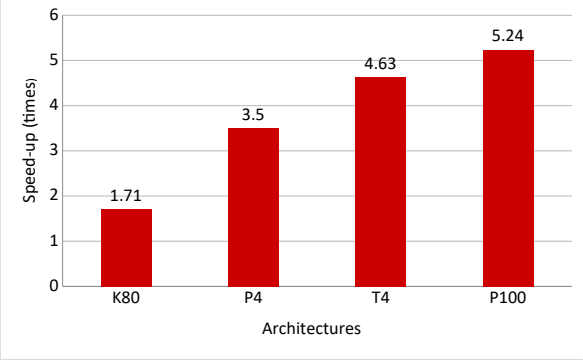

Fig. 2. GPU architecture Speedup compared to TPU .

For the development of CNN, a total number of filters (num_features) was defined as 32, the division of labor, in this case, to perform the adjustments of the RNA weights (batch_size) as 16 and a total of 100 seasons, for the training. A stop metric (EarlyStopping) was also defined, in the period of 15 seasons. In the convolution layer, the ELU (Exponential Linear Unit) activator was used, Dropout was 20%.

The tests were performed using the four video cards available in Google Colab. The accuracy values were compared with the execution performed in CPU and TPU, Figure 3. The presented accuracy values are obtained from the test base and not from the RNA validation base. The GPU that presented the best accuracy was the Tesla K80, obtaining 65.67%. The Tesla T4 obtained an accuracy of 65.39, while the other GPUs obtained a slightly lower accuracy when compared with the

CPU and TPU. The Tesla P4 had 64.05 accuracy and Tesla P100 reached 64.92, TPU had an accuracy of 65.25, while the CPU had an accuracy of 65.17.

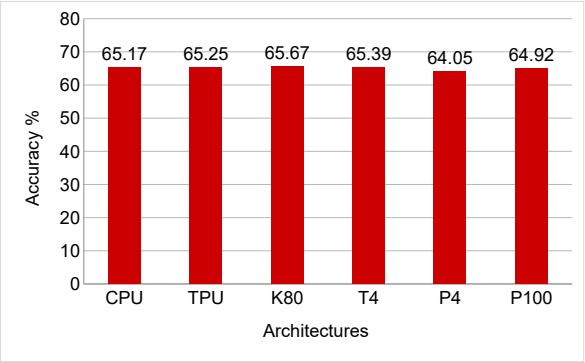

Fig. 3. Result of the accuracy obtained in each architecture.

V. Tests Performed

To perform the tests, before predicting what emotion is present in the image, it is necessary to load an image and recognize the faces. For the recognition was used the OpenCV library, which already has a pre-trained model with the type of attribute to be identified, in this case the faces. The file to be used is called haarcascade_frontalface_default.xml.

With the image and the classifier initialized, it is already possible to perform face identification on the images. However, the classifier used only accepts grayscale images, so it is necessary to convert the original image to grayscale. Having converted the image to grayscale, it is now possible to detect faces and work with the image to recognize the emotion predicted in the image.

CNN tests were performed on all architectures, CPU, GPU and TPU, but to illustrate the test result, Figure 4 shows the result obtained by the architecture that obtained the best precision in the training phase, in this case, the tests were performed on the Tesla K80 GPU.

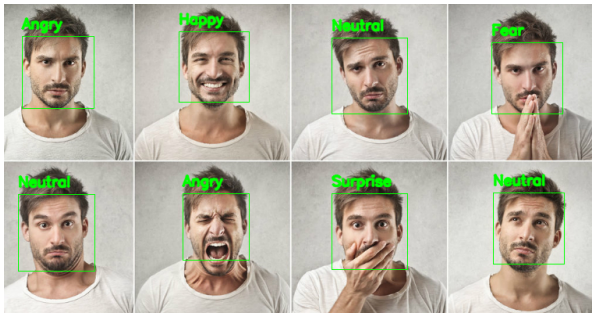

Fig. 4. Prediction Performed on CNN.

In the comparison made in the tests on all the architectures, CNN produced a confusion in the identification of the emotion of fear and surprise. The fact that confusion occurs between the emotion of fear and surprise is that both have similar facial characteristics, which can produce a confusion at the time of their identification. The similarities between the two emotions are raised eyebrows and open jaw [8] [9].

VI. Conclusions and Future Works

This work approached the development of a convolutional neural network for the recognition of emotions through facial expressions, performing numerous tests to analyze and evaluate the performance of the application running on CPU, and GPU architectures. With the new implementation it was possible to increase the accuracy of RNA reaching an accuracy of up to 65.67%. Regarding computational time, the version developed for Tesla P100 GPU, achieved gains, reducing the execution time by up to 10.71 times compared to the CPU execution.

As future works it is intended to apply the solution developed in other databases to validate their behavior. A second initiative would be to develop the neural network to make the recognition of emotions in real time of people in videos. Also modify the algorithm of the Artificial Neural Network so that you can use the Google Cloud TPU execution environment, which has TPUs that have a better archiving, than the free TPU provided in Google Colab, and also analyze the influence of other hyperparameters, such as learning rate, Dropout and Activation Function.

References

[1] L. P. Leão, J. S. Bezerra, L. N. Matos, and M. A. S. N. Nunes, "Detecção de expressões faciais: uma abordagem baseada em análise do fluxo óptico," *Revista GEINTEC-Gestão, Inovação e Tecnologias*, vol. 2, no. 5, pp. 472–489, 2012.

[2] P. Ekman, "Cross-cultural studies of facial expression," *Darwin and facial expression: A century of research in review*, vol. 169222, no. 1, 1973.

[3] M. S. Bartlett, G. Littlewort, I. Fasel, and J. R. Movellan, "Real time face detection and facial expression recognition: development and applications to human computer interaction." in *2003 Conference on computer vision and pattern recognition workshop*, vol. 5. IEEE, 2003, pp. 53–53.

[4] H. Tang and T. S. Huang, "3d facial expression recognition based on properties of line segments connecting facial feature points," in *2008 8th IEEE International Conference on Automatic Face & Gesture Recognition*. IEEE, 2008, pp. 1–6.

[5] D. Amin, P. Chase, and K. Sinha, "Touchy feely: An emotion recognition challenge," *Palo alto: Stanford*, 2017.

[6] R. C. Gonzalez and R. E. Woods, *Processamento de imagens digitais*. Editora Blucher, 2000.

[7] J. D. P. Massucatto, "Aplicação de conceitos de redes neurais convolucionais na classificação de imagens de folhas," B.S. thesis, Universidade Tecnológica Federal do Paraná, 2018.

[8] P. Ekman and W. V. Friesen, *Unmasking the face: A guide to recognizing emotions from facial clues*. Ishk, 2003.

[9] P. Ekman, "A linguagem das emoções," *São Paulo: Lua de Papel*, 2011.

Performance Analysis of Artificial Neural Networks in CPU and GPU Platforms Applied in Text Sentiment Analysis

1st Cristiano Alex Künas
Regional University of Northwest of Rio Grande do Sul (UNIJUI)
Santa Rosa, RS – Brazil
cristiano.kunas@sou.unijui.edu.br

2nd Leandro Perius Heck
Regional University of Northwest of Rio Grande do Sul (UNIJUI)
Santa Rosa, RS – Brazil
leandro.h@sou.unijui.edu.br

3rd Edson Luiz Padoin
Regional University of Northwest of Rio Grande do Sul (UNIJUI)
Ijuí, RS – Brazil
padoin@unijui.edu.br

***Abstract*—Nowadays, a lot of information with all kinds of content circulates on the Internet. Analyzing the sentiments exposed can help in understanding what people are talking about a particular company, brand, event or even other people, working as a way to get feedback. In this work, we present an Artificial Neural Network model for sentiment analysis in English language sentences. The Long Short-Term Memory Recursive Artificial Neural Network was implemented for the training of the sentiment analysis model. With the application of ANN developed over a public database with 50,000 movie records using GPU it was possible to reduce the training time of RNAs by up to 91.8% and increase the accuracy to 87.7%.**

***Index Terms*—Sentiment Analysis, Artificial Neural Network, Long Short-Term Memory, Natural Language Processing, GPU**

I. Introduction

Countless information circulates on the Internet through websites, blogs and social networks. The present content ranges from analysis and comments on films to conversations and experiences of its users. In this sense, using technological resources we can store, retrieve and analyze a huge amount of this data efficiently. One thing is that this data is not structured in an understandable way for a single computer system.

Opinions are so important for almost all human activities that, whenever we need to make a decision, we want to hear the opinions of others [1].They are the main influencers of our behavior. This informal or formal opinion, depending on where it was posted, is extremely important, because it will reflect the uncensored feeling of the user. The great growth of social media, such as blogs and social networks has awakened a lot of interest, especially the Sentiment Analysis (SA). However, this growth makes it impossible for any human being to be aware of all the content that interests them in a timely manner.

Individuals and organizations are increasingly using the content of these media for decision making. But, computer systems still find it very difficult to understand how customers feel about the products and services of a certain company. In some situations, only a few points of the text are relevant: about who is spoken about, and whether what is spoken is good or bad [2]. Artificial Intelligence (AI) can be applied to facilitate the understanding of this data, so that we can use a sentence in natural language as input and extract a set of data in output. In this way, the use of an Artificial Neural Network (ANN) can help in the automatic extraction of the feeling or sensation of sentences.

Among the different sub-areas of study that the field of Sentiment Analysis presents, the task most present in the literature and that we will approach in this work is the analysis of polarity of a document, which aims to classify texts on a scale between positive and negative, since this area of research is of great relevance for consumers and organizations. Therefore, the present work aims at: i) creating a recurrent neural network of the Long Short-Term Memory (LSTM) type; ii) performing its training using a public database; and iii) applying it to the recognition of sentiments expressed in texts and/or posts.

The work is organized as follows. Section II discusses the related work. Section III presents the methodology used in the implementation and the execution environment used in the tests. Results are discussed in Section IV, followed by conclusions and perspectives of future work.

II. Related Work

Several works present studies on Sentiment Analysis. This concept was first introduced in 2003 where techniques were used in opinion mining in history. Algorithms based on machine learning were used in [3] to classify the positioning of a person in relation to an object, idea or posture. In this work, people's positioning was classified and applied to political debates. They used 104 debates of double positioning from convinceme.net for 14 different themes and tried to identify the opinion of those involved. The main objective was to determine the potential contribution of dialogue characteristics in classifying debates. They used Support Vector Machines (SVM), Naive Bayes (NB) and a rules-based classifier for classification purposes and were able to specify the choice of the debate side.

In [4], sets of decision trees are used to do sentiment analysis in movie commentaries. The authors set the number

of trees to 100 as a reasonable default value. However, for the extraction of attributes from the text, they used deep learning methods. The methodology chosen was Word2Vec which allows the capture of semantic characteristics of words. The vectors obtained by the attribute extraction process were clustered by k-means. Each cluster, from a total of 2000, had an average of 5 words. These were selected for their proximity in vector space. The clusters were used as entries for the classifier. At work, the IMDb database was used. The result of the experiment showed that the approach using the extraction of attributes by deep learning was significantly better than the one using bag-of-words with 5000 entries.

Deep neural networks were used in [5] for sentiment analysis in reviews of electronic products, films and hotels. The authors created a classification framework that uses 3 different methods for attribute extraction: based on frequency, context and labeling of parts of speech. Each method feeds a neural subnet that reduces the dimensionality of the attribute space. The outputs of these subnets feed the main network which is responsible for feeling analysis. The structure formed by the subnets and the main net is called a deep hierarchical neural net. The work compared the performance of the structure proposed by the authors with machine learning methods (NB and SVM). The analysis found that the accuracy of the deep neural network model was superior to NB and SVM for a large number of samples in the training set (over 200000). However, the accuracy of the proposed model decreased for cases where the dimensionality of the inputs is large and the number of samples for training is reduced. The work also showed that regardless of the context of the comments the deep learning approach presents increasing precision with the number of samples in the training. The databases used were amazon.com, IMDb and TripAdvisor.

Sentiment analysis at the sentence level was studied in [6]. The authors created a Chinese language sentence database called Recursive Neural Deep Model (RNDM). Unlike other databases that have a global feeling for each sentence, this has feeling ratings for each sentence within each sentence and for each word within each sentence. The work shows the performance of this system compared to others with machine learning methods (NB, SVM, and Logistics Regression).

These works contemplate important contributions in the use of Artificial Neural Networks for classification of feelings. Following this premise, our proposal is to evaluate the use of a Long Short-Term Memory Recurrent RNA, aiming to increase performance in its training through the use of GPU architecture.

III. Methodology

The ANN implementation of this work is done using Python programming language [7]. This was chosen because of its simplicity and the large number of well documented modules that help from preprocessing data to the application of machine learning models. The main modules used are: Keras [8], TensorFlow [9] and Scikit-Learn [10]. Keras is used to model RNA. This is a high level API, developed with user focus, to allow fast experimentation. It supports several network configurations and works with both CPU and GPU. TensorFlow is Keras standard back-end numerical computing library. This is an open source software library for numerical computing using data flow graphs. It is flexible and allows fast and easy interfacing with CUDA (Compute Unified Device Architecture) [11]. To divide the data used in this work, in training and testing, Scikit-Learn is used. Scikit-learn is a Python module that integrates several state-of-the-art machine learning algorithms for supervised and unsupervised medium scale problems. This package is dedicated to bringing machine learning to non-specialists who use a high-level language for general purposes. Emphasis is placed on ease of use, performance, documentation and API consistency [10].

The database used in the experiments was developed in the work of [12]. This database has a collection of $50,000$ IMDb evaluations written in English. The criticisms are well balanced in 2 classes: positive and negative. The preprocessing of these data is done in stages with the objective of removing noise present in the sentences. They are: i) special characters and punctuation marks are removed; ii) all words are written in lowercase letters; iii) stopwords are removed - these are words that do not add much meaning to the text. They are usually articles, conjunctions and prepositions; iv) sequences are limited to a fixed size (300 words);

The creation of the ANN model is defined as a sequence of layers. The Model class of the Keras library functional API is used in the development of the data model. The compilation of the model configures the learning process. It defines the optimizer (adam), the loss function (binary_crossentropy) and the metrics (accuracy). To train the model, the data are divided into batches (batch_size), iterating repeatedly throughout the data set for a certain number of seasons. The model is evaluated using the *evaluate()* function. This function will generate a prediction for each input and output and will collect scores, take an average over the loss value and accuracy. Figure 1 shows the ANN architecture used in this work, simulated in TensorBoard[1].

The execution environment is composed of a device with an Intel Core i7-9750 processor with 6 cores (12 threads) of 2.60 GHz frequency. This equipment has 16GB of DDR4 RAM, NVIDIA GeForce GTX RTx 2060 GPU with 6GB of GDDR6 and 1920 CUDA cores, used the Linux Ubuntu 18.04.3 LTS operating system with kernel version 5.0.0-37. The NVIDIA CUDA Compiler version used was 10.0.130.

IV. Results

Figure 2 shows the accuracy (y-axis) achieved from the training dataset (40000 records), with different batch sizes, in both CPU and GPU executions. It is noticed that as the number of instances for each gradient update increases, the accuracy of the model is reduced, that is, less ANN weights are updated and consequently the loss rate increases.

[1] https://www.tensorflow.org/tensorboard

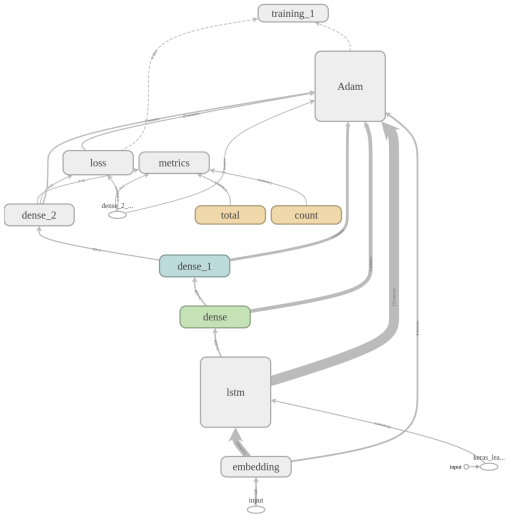

Fig. 1. Simulated ANN architecture in TensorBoard

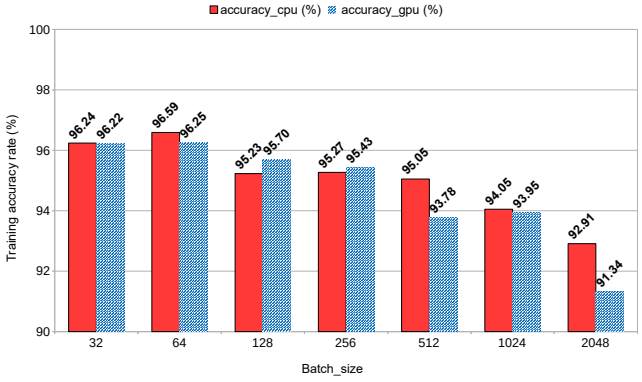

Fig. 2. Metrics for training data of CPU and GPU executions.

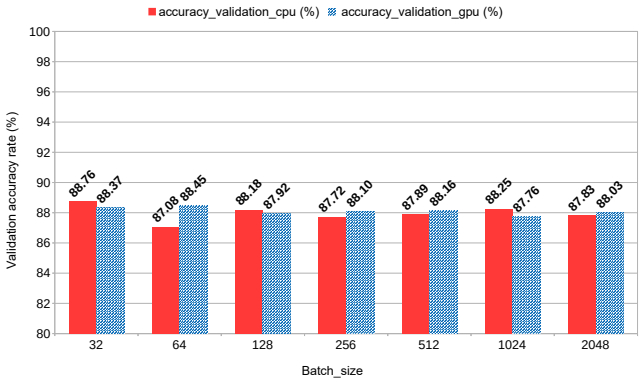

Fig. 3. Metrics for validation data of CPU and GPU executions.

Figure 2 shows the accuracy (y-axis) obtained from the validation dataset with 10000 records by varying the batch_size and running in CPU or GPU. The validation accuracy informs

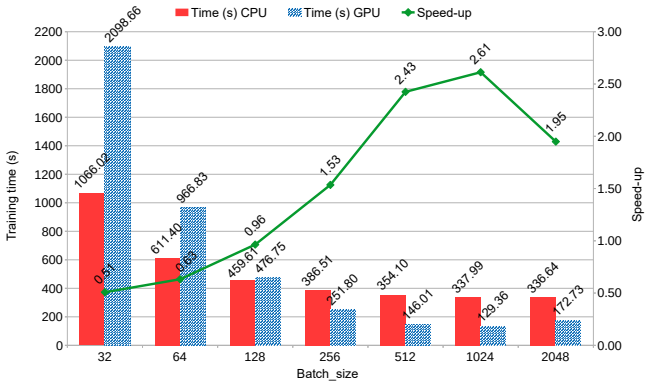

Fig. 4. Training time and ANN speed-up on CPU and GPU architectures.

the percentage that our model has right in the prediction with respect to the label. These tests prove that the implementation accuracy is stable, varying between $87 \sim 89\%$.

Figure 4 shows the training times according to the defined batch_size. It can be seen that as the batch size increases, the runtime is reduced considerably on both architectures. CPU runs showed a gain of 3.17 times. This reduces the execution time from 1066.02 seconds to 336.64 seconds. In GPU executions, the gain reached 12.15 times, jumping from 2098.66 seconds to 172.73 seconds, this represents a gain of 91.8%.

The gain is very expressive if compared to CPU. Yet, in runs with smaller batch_size, it is possible to notice that the execution time in GPU practically doubles in relation to the CPU time. This is because the GPU takes longer to transfer the data than the training itself. When the number of instances increases, it is possible to use more of the capacity the GPU offers.

The speed-up of the application is also demonstrated in Figure 4, presenting its best case in training using batch_size=1024. This represents a gain of 2.61 times over the CPU, reducing the CPU runtime from 337.99 seconds to 129.36 seconds on the GPU. This is due to the GPU devoting more resources to training rather than prioritizing data transfer. However, when the computational resources limit is reached, it also causes loss of performance, as can be seen by increasing the batch_size to 2048. This occurs due to having more processes running and dividing the computational resources.

To validate ANN LSTM, the trained model that presented the lowest loss rate and the highest accuracy rate, both values obtained from the validation dataset, was selected. This resulted in the GPU trained model with batch_size=2048. Figure 5 shows the metrics of the model, with values accuracy = 87.76% and loss = 30.33%. The model is loaded by the ANN algorithm, and then the inputs are submitted for testing. The new entries are random reviews obtained from the official IMDb[2] page about the film Wonder Woman (2017).

[2]https://www.imdb.com/

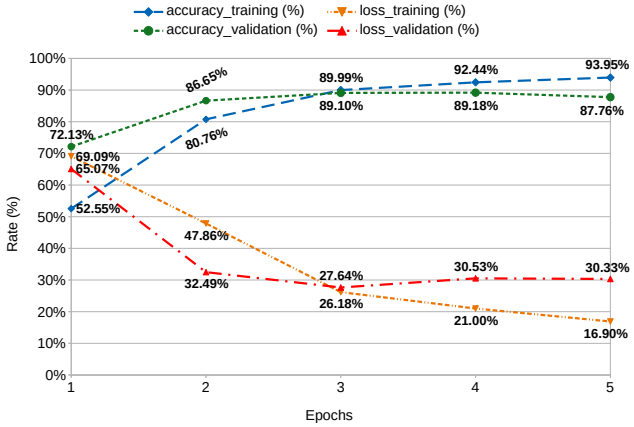

Fig. 5. Training metrics of the selected model.

Review	Evaluation	ANN Prediction
1	1/10	99.22% N
2	5/10	95.80% N
3	7/10	82.83% P
4	9/10	99.74% P
5	1/10	99.54% N
6	10/10	99.71% P
7	9/10	99.93% P
8	5/10	99.01% N
9	8/10	78.01% P
10	3/10	93.40% N

TABLE I
RESULTS OF PROPOSAL VALIDATION. PREDICTION: NEGATIVE (N), POSITIVE (P)

The results obtained are shown in Table I. The first column indicates the selected review. The second column is the number of stars indicated by the author of the review, considering as a negative opinion the evaluations between 1 and 5 and positive opinion the evaluations between 6 and 10. The last column indicates the prediction made by our model. When analyzing the table, it can be seen that the ANN model tested was assertive in all cases submitted.

V. CONCLUSIONS AND FUTURE WORK

This work addressed the use of recurrent artificial neural networks Long Short-Term Memory in the classification of feelings into sentences. Different configurations were used to analyze the performance of the application running on CPU and GPU architectures. Regarding the accuracy rate, for CPU architecture, the best rate is registered for batch size 32. For GPU architecture, the best rate is obtained for batch size 64. The proposal presented good performance in terms of ANN accuracy. Runtime is reduced considerably on both architectures by increasing the batch size. Among the most significant gains, CPU runtime gained 3.17 times. The GPU runtime was reduced by 12.15 times. Comparing the architectures, the GPU runtime showed a reduction of up to 61%. This represents a speed-up of 2.61 times over CPU time. These results indicate that the use of GPU reduces RNAs training time, presenting good yields for larger batch sizes.

As future works, a first initiative could be the use of pre-trained word embeddings. In addition, it is possible to study the influence of other hyperparameters such as learning rate, Dropout rate and Activation Function. It is also possible to evaluate larger databases and in other languages. Another analysis that can be done is to add other classification classes besides positive and negative, such as very positive, very negative and neutral. As well as evaluating other data sources like twitter. Another approach to be studied is a better separation of training, testing and validation bases.

REFERENCES

[1] B. Liu *et al.*, "Sentiment analysis and subjectivity." *Handbook of natural language processing*, vol. 2, no. 2010, pp. 627–666, 2010.

[2] I. P. P. dos Santos, "Análise de sentimento usando redes neurais de convolução," 2017, dissertação (Mestrado em Engenharia Eletrônica), Universidade do Estado do Rio de Janeiro, Rio de Janeiro, Brazil.

[3] M. A. Walker, P. Anand, R. Abbott, J. E. F. Tree, C. Martell, and J. King, "That is your evidence?: Classifying stance in online political debate," *Decision Support Systems*, vol. 53, no. 4, pp. 719–729, 2012.

[4] A. S. Zharmagambetov and A. A. Pak, "Sentiment analysis of a document using deep learning approach and decision trees," in *2015 Twelve international conference on electronics computer and computation (ICECCO)*. IEEE, 2015, pp. 1–4.

[5] Z. Hu, J. Hu, W. Ding, and X. Zheng, "Review sentiment analysis based on deep learning," in *2015 IEEE 12th International Conference on e-Business Engineering*. IEEE, 2015, pp. 87–94.

[6] C. Li, B. Xu, G. Wu, S. He, G. Tian, and H. Hao, "Recursive deep learning for sentiment analysis over social data," in *2014 IEEE/WIC/ACM International Joint Conferences on Web Intelligence (WI) and Intelligent Agent Technologies (IAT)*, vol. 2. IEEE, 2014, pp. 180–185.

[7] G. Rossum, "Python reference manual," 1995.

[8] J. Moolayil, "An introduction to deep learning and keras," in *Learn Keras for Deep Neural Networks*. Springer, 2019, pp. 1–16.

[9] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin *et al.*, "Tensorflow: Large-scale machine learning on heterogeneous distributed systems," *arXiv preprint arXiv:1603.04467*, 2016.

[10] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg *et al.*, "Scikit-learn: Machine learning in python," *the Journal of machine Learning research*, vol. 12, pp. 2825–2830, 2011.

[11] J. Nickolls, I. Buck, M. Garland, and K. Skadron, "Scalable parallel programming with cuda," *Queue*, vol. 6, no. 2, pp. 40–53, 2008.

[12] A. L. Maas, R. E. Daly, P. T. Pham, D. Huang, A. Y. Ng, and C. Potts, "Learning word vectors for sentiment analysis," in *Proceedings of the 49th annual meeting of the association for computational linguistics: Human language technologies-volume 1*. Association for Computational Linguistics, 2011, pp. 142–150.

Evaluating the Performance and Cost Efficiency of HPC Interconnections on the Azure Cloud

Anderson M. Maliszewski*‡, Eduardo Roloff*, Emmanuell D. Carreño†,
Dalvan Griebler‡§, Luciano P. Gaspary*, Philippe O. A. Navaux*
*Informatics Institute, Federal University of Rio Grande do Sul (UFRGS), Porto Alegre, Brazil
†Department of Informatics, Federal University of Paraná (UFPR), Curitiba, Brazil
‡Laboratory of Advanced Research on Cloud Computing (LARCC), Três de Maio Faculty (SETREM), Três de Maio, Brazil
§School of Technology, Pontifical Catholic University of Rio Grande do Sul (PUCRS), Porto Alegre, Brazil
Email: {ammaliszewski,eroloff,paschoal,navaux}@inf.ufrgs.br*, edcarreno@inf.ufpr.br†,
dalvan.griebler@acad.pucrs.br§

***Abstract*—The availability of computing resources has significantly changed due to the growing adoption of the cloud computing paradigm. Aiming at potential advantages such as cost savings through the *pay-per-use* premise and resource allocation in a scalable/elastic way, we witnessed consistent efforts to execute high-performance computing (HPC) applications in the cloud. Performance in this environment heavily depends upon two main system components: processing power and network interconnection. If, on the one hand, allocating more powerful hardware theoretically boosts performance, on the other hand, it increases the allocation cost. In this paper, we evaluated how the network interconnection impacts performance and cost-efficiency. Our experiments were carried out using NAS Parallel Benchmarks and Alya HPC application on Microsoft Azure public cloud provider, with three different cloud instances/network interconnections. The results revealed that through the use of the accelerated networking approach, which allows the instance to have a high-performance interconnect without additional charges, HPC applications' performance can be significantly improved with better cost-efficiency.**

I. DISCLAIMER

This abstract describes the paper entitled "Performance and Cost-aware HPC in Clouds: A Network Interconnection Assessment" from the same authors, that has been accepted for publication on the IEEE symposium on Computers and Communications (ISCC 2020).

A Dynamic Cost Model to Minimize Energy Consumption and Processing Time for IoT Tasks in a Mobile Edge Computing Environment

João L. G. Gross, Kassiano J. Matteussi, Julio C. S. dos Anjos, and Cláudio F. R. Geyer
Federal University of Rio Grande do Sul (UFRGS), Institute of Informatics, Porto Alegre, RS, Brazil, 91509–900

***Abstract*—The rapid growth of devices, e.g., smartphones, wearables, tablets, and other sensors connected to the Internet, has been leading to a complex problem regarding how to optimize energy consumption for data-intensive processing. Most applications tend to offload task processing to remote servers, usually to Data Centers in the Cloud, geographically located away from the end-user and the IoT device, increasing communication latency and energy costs. In such a context, this work proposes a dynamic cost model to minimize energy consumption and task processing time for IoT scenarios in a Mobile Edge Computing environments. The solution presents a Time and Energy Minimization Scheduler (TEMS) that solves the cost model, validated through simulation. Results reveal a reduction in energy consumption by up to 51.61% as well as 86.65% in tasks completion time.**

***Index Terms*—Mobile Edge Computing, Internet Of Things, Cost Minimization Model, Energy Consumption, Scheduling Algorithm.**

DISCLAIMER

This paper has been accepted and registered in the 18th International Conference on Service-Oriented Computing (ICSOC 2020) as short paper. The paper is not yet published.

Advances in GPPD–PCAD Management with 12-months Analysis and Perspectives

Lucas Leandro Nesi, Matheus S. Serpa, Lucas Mello Schnorr, Philippe Olivier Alexandre Navaux
Institute of Informatics, Federal University of Rio Grande do Sul - UFRGS, Porto Alegre, Brazil

***Abstract*—High-Performance computing resources are the basis for many projects, and a shared utilization is the only way to be efficient in the allocation by the users. Managing these resources and enabling a shared use is complex and requires configurations in many levels of the hardware and software stacks. Although different modern tools exist to assist in this task, it remains complex and requires many infrastructure decisions that must cope with the max possible utilization cases. This paper presents the last advances in GPPD–PCAD resource management over the past year. We report the new machines, software changes, and the adoption of new tools to enable users and administrators a better experience. We also present an analysis of the 12 months from August/2019 to August/2020 of resource utilization that focuses on compute nodes and end-users. In this analysis, we offer a general user behavior that includes infrastructure utilization during the COVID-19 pandemic.**

I. Introduction

High-Performance computing resources enable many users to execute their resource-intensive applications [1]. The management of these resources is complex and requires many configurations, sometimes tailored for specific vendor components. Technologies do exist to share these resources across many users, and effectively guarantee that everyone can use them. Examples of such tools are Slurm [2] and OAR [3], of the French Grid5000 [4]. Moreover, the software stack of these platforms is complex, and tools that automatize configuration are desirable. Many configuration management (CM) software exist, such as Ansible [5], Jenkins [6], and Puppet [7].

We present the last year's advances in GPPD–PCAD[1], which stands for *Parque Computacional de Alto Desempenho* of the Grupo de Processamento Paralelo e Distribuído (GPPD)[2] of INF-UFRGS. The last GPPD–PCAD report [8] presented the cluster and initial configuration. This paper focuses on the new advancement related to that publication until today. The significant contributions are as follows: (i) the extension of the cluster with six new machines; (ii) changes in the software stack to enable a better experience, by including virtualization with Docker [9] and Singularity [10]; utilization of Ansible and IPMI (Intelligent Platform Management Interface) [11] to configure and maintain the cluster updates; and employment of BTRFS (B-tree file system) [12] on the main NFS partition; (iii) the analysis of the last 12-months, discussing how it is the general job submission configuration, resource utilization, and the behavior during the COVID-19 pandemic.

Section II presents the hardware highlighting the new machines. Section III details the alterations made in the software stack from the last report [8]. Section IV analyses the 12-months (August 2019 to August 2020) resource utilization with different statistics. Finally, Section V concludes the paper with future perspectives and ideas to be implemented.

Also, we highlight that all the knowledge gathered in the process of building and maintaining GPPD–PCAD is registered in an internal repository for future aspirants. We find relevant the know-how of managing HPC resources and think that it may be useful for prospective students that will maintain it. Students interested in participating may contact the team [3].

II. Hardware Infrastructure

The GPPD–PCAD infrastructure is compost of 33 nodes, four racks, and is interconnected by two networks. First, a Gigabit Ethernet for internal communication (Slurm, NFS, ssh). Second, a 100 Megabit Ethernet that is accessible via the main site (UFRGS-INF) network. Each node has at least one disk split into three partitions, 100GB for the system, 2GB for swap and the rest for scratch user space. Also, most of the machines have IPMI interfaces that vary per vendor. In some cases, the IPMI interface is also connected to the site or internal network depending on individual characteristics.

All the resources are present in Table I with machine name, CPU, RAM, Disk space, and Accelerators. The new additions are marked with a star symbol (⋆). These new machines comprehend new vendor architectures (`apolo` and `sirius` with AMD CPU, and `tsubasa` with NEC SX-Aurora Vector Engine). Also, GPPD–PCAD provides its managing system to machines of other INF-UFRGS research groups, including `saude`, `cidia`, and `cei[1-2]` machines.

III. Software Stack and Managing Tools

The GPPD–PCAD infrastructure uses standard and modern tools to manage its resources [8]. Though, over time, it is expected that alternatives or extra tools start to be utilized as we intend to keep the system updated with the most recent and capable tools. This Section presents the software stack changes of GPPD–PCAD and the new tools employed in this last year to solve and automatize many tasks. First, the operational system of the computing nodes was changed to Debian 10. We believe that this system is more aligned with general users' Linux knowledge and does not force the presence of extra software

[1]The GPPD–PCAD website: http://gppd-hpc.inf.ufrgs.br/
[2]The GPPD website: http://www.inf.ufrgs.br/gppd/site/
[3]GPPD–PCAD administration team e-mail: gt-cluster-l@inf.ufrgs.br

TABLE I
COMPUTATIONAL RESOURCES AVAILABLE ON GPPD–PCAD

Machine	CPU	RAM	Storage	Accelerator
gppd-hpc	2 x Intel Xeon E5-2630	16 GB DDR3	22 TB	
apolo⋆	AMD Ryzen 5 3400G	48 GB DDR4	480 GB	AMD Radeon Vega 11
bali1	2 x Intel Xeon E5-2650	32 GB DDR3	1 TB	
bali2	2 x Intel Xeon E5-2650	32 GB DDR3	4 TB	
beagle	2 x Intel Xeon E5-2650	32 GB DDR3	1 TB	
blaise	2 x Intel Xeon E5-2699 v4	256 GB DDR4	4 TB	4 x NVIDIA Tesla P100-SXM2
cei[1-2]⋆	2 x Intel Xeon Silver 4116	96 GB DDR4	13 TB	
cidia⋆	2 x Intel Xeon Silver 4208	320 GB DDR4	10 TB	2 x NVIDIA GeForce RTX 2080 Ti
draco[1-6]	2 x Intel Xeon E5-2640 v2	64 GB DDR3	2 TB	1 x NVIDIA Tesla K20m
draco7	2 x Intel Xeon E5-2640 v2	128 GB DDR3	1 TB	2 x NVIDIA Tesla K20m
hype[1-3]	2 x Intel Xeon E5-2650 v3	128 GB DDR4	600 GB	
hype[4-5]	2 x Intel Xeon E5-2650 v3	128 GB DDR4	600 GB	2 x NVIDIA Tesla K80
knl[1-4]	Intel Xeon Phi 7250	96 GB DDR4	480 GB	
orion1	2 x Intel Xeon E5-2630	48 GB DDR3	1 TB	NVIDIA Tesla K20m
orion2	2 x Intel Xeon E5-2630	32 GB DDR3	1 TB	NVIDIA Tesla K20m
saude	Intel Xeon CPU E5-2620 v4	128 GB DDR4	8 TB	4 x NVIDIA GeForce GTX 1080Ti
sirius⋆	AMD Ryzen 9 3950X	64 GB DDR4	240 GB	NVIDIA GeForce GT 1030
tsubasa⋆	2 x Intel Xeon Gold 6226	192 GB DDR4	2 TB	4 x NEC SX-Aurora TSUBASA
tupi1	Intel Xeon E5-2620 v4	64 GB DDR4	2.5 TB	2 x NVIDIA GeForce GTX 1080Ti
tupi2	Intel Xeon E5-2620 v4	80 GB DDR4	5.2 TB	
turing	4 x Intel Xeon X7550	128 GB DDR3	5 TB	

of the distribution's organization. Another difference is the revisited Slurm job's priority configuration. Now, we utilize the global and partition components to better schedule tasks by various users (switching faster) and not by submission order.

Moreover, the management of the resources became more accessible with the utilization of IPMI. This feature also highlights the heterogeneity of the resources not only in computational hardware but in this interface. The nodes are from at least three different major vendors with their customs management system that integrates with IPMI. We took a close look at each one to configure them better for our needs. This configuration showed significant importance during the COVID-19 pandemic as the physical access to the infrastructure site was severely limited.

Furthermore, the NFS partition file system present on the front-end node was changed from ext4 to BTRFS [12]. BTRFS (B-tree file system) is a copy-on-write file system currently being developed in the mainline Linux kernel that presents many desirable features. For us, the most important one is that it enables transparent compression for files and can be configured with different algorithms (We use `lzo`). It automatically detects which files would benefit from compression and applies it. Currently, in the 22TB RAID NFS partition present on the front-end, there is a total of 16TB data but only 10TB physical space being used, a saved of 6TB by BTRFS compression. Although BTRFS compression may present a little overhead with some workloads [13], the NFS partition should not be used by time-sensitive applications, as it already has the network overhead. The NFS partition's main objective is to enable a global file-system across all nodes for medium/long period files, typical present in the applications' start or end. Temporary files present during execution should use the intra-node storage (`/scratch`). Other exciting features that BTRFS presents are system snapshots, space-efficient packing of small files, and sub-volumes.

The management of all these resources requires much automatization, and different modern tools exist with this goal, including Jenkins [6], Puppet [7], and Ansible [5]. We select Ansible [5] to use in GPPD–PCAD, as other supercomputers use it for CM (Configuration Management). The basic structure of Ansible compounds hosts (the nodes) and playbooks (the scripts), where a playbook can be performed on a set of nodes. We mainly utilize Ansible to keep machines updated and guarantee the last configuration files on them. These actions are performed recurrently and automatically. Though, if a node is being used (allocated), a Slurm job will be created to respect and not interfere with that user allocation.

We also highlight the availability of two newly installed software, the virtualization tools Docker [9] and Singularity [10]. Because the cluster has permission limitations over some software, including the Linux kernel, the users can virtualize their own system and make modifications as they please, considering their limitations. These changes provide users more freedom and do not require administration interference or software installation requiring admin-level permissions.

IV. RESOURCE UTILIZATION ANALYSIS

This Section presents general resource utilization statistics of GPPD–PCAD from 2019-08-01 to 2020-08-26 with 82713 jobs submitted by 80 users. Currently, GPPD–PCAD already processed 352000 jobs (all durations) with more than 100 registered users. Similar to the last report [8], we present the density of jobs duration in hours in Figure 1. The X-axis is the duration of jobs in hours, and the Y-axis is the total density computed by `stat_density` of the `ggplot2` R package function. The majority of the jobs take less than 3 hours to conclude, with remarkable peaks in half-hour and 2

hours. Also, there are small peaks in 5, 6.5, and 12 hours. Small jobs are a desirable situation as the batch management system can schedule better the jobs of different users across the resources. The duration of the jobs is directly correlated to users' allocation choices and their applications' behavior.

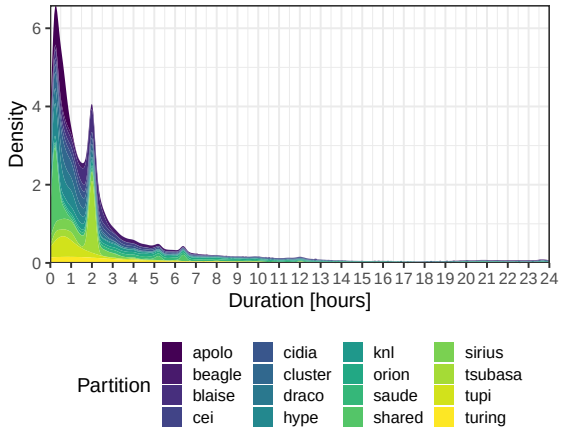

Fig. 1. Jobs duration density in hours (of jobs with a duration greater than 2 minutes) in GPPD–PCAD 16 partitions.

Another interesting metric is the resource utilization across day hours (0-24h). Figure 2 presents this metric (Number of jobs per inner day hours) for the 16 partitions. The X-axis is the daytime (Aggregated per minute), and the Y-axis is the total amount of jobs that were executed during that minute. Like last year's results [8], the day starts (00:01) with the number of jobs decreasing until more or less 8:00 when it starts to grow up again. The number of jobs then reaches a peak at 16:00~17:00 and dropping again until the end of the day. This reflects the regular work hours of the users. Most jobs are submitted between 8:00-17:00 (explaining the increasing). There are few jobs submitted at night (18:00-07:00), while the remaining jobs still are finishing (explaining the decreasing).

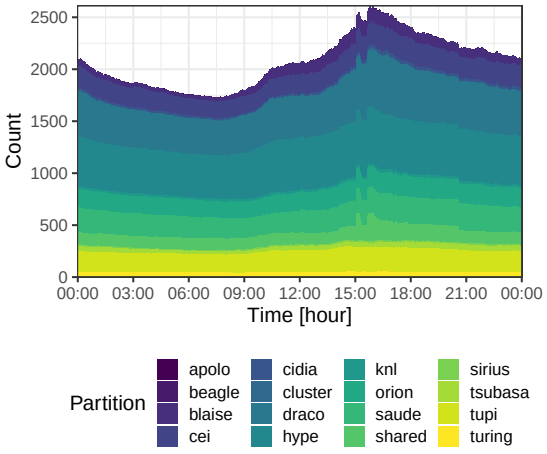

Fig. 2. Utilization of the resources by day minute (of jobs with a duration greater than 2 minutes) in GPPD–PCAD partitions.

Per-user utilization is also another critical metric. Figure 3 shows the cumulative node-hours utilization for the top 30 users in the analyzed period. The utilization of the first four users outlines from the others. Also, users 3 and 4 only use one partition. From rank five to 30, the utilization smoothly drops with utilization in all partitions. This also shows that most users prefer to use the same partitions over time, in general, not using more than four different ones.

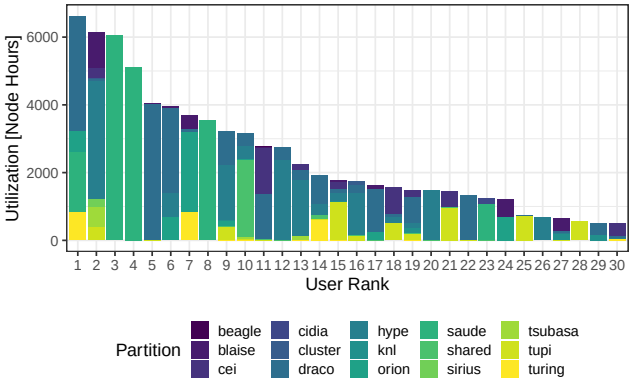

Fig. 3. Resource utilization (Node hours) per partition of the 30th top users.

Figure 4 presents the utilization per machine in the last 12 months (August/2019 to August/2020) in the X-axis. The top panel presents the % of utilization on that day in the cluster. The bottom panel shows a typical Gantt chart per individual resource (Y-axis) where states are running jobs (allocations) at that moment. In this Figure, a resource is utilized if there is at least one job on that day. The black line presents the actual values, while the blue line is the smooth curve computed by `geom_smooth` of the `ggplot2` R package with the `gam` method. Regarding machine utilization, some clusters are much busier than others. For example, `dracos` and `hypes` are intensively utilized, while specific and exotic architectures like the Intel Xeon Phi on `knls` present low utilization. Some machines were integrated into the GPPD–PCAD during the year, so in the plot, there are long non-utilized areas where these machines are absent (`Sirius`, for example).

Moreover, there are two moments highlighted in red areas that change the behavior of GPPD–PCAD utilization that occurred because of external events. First, at the end of January and the beginning of February, along with electrical maintenance, shut down the infrastructure for a few days. Second, in the middle of March, it started the COVID-19 pandemic restrictions, impacting the physical access and utilization of the GPPD–PCAD at our university (UFRGS). A drop in the resources' utilization is perceptive for the first days, but it rapidly returned to normal values.

V. Conclusion

This paper presented the advances in hardware and software of the GPPD–PCAD resource management. The six new compute nodes were shown along with the following improvements in the software stack: (i) remote management and

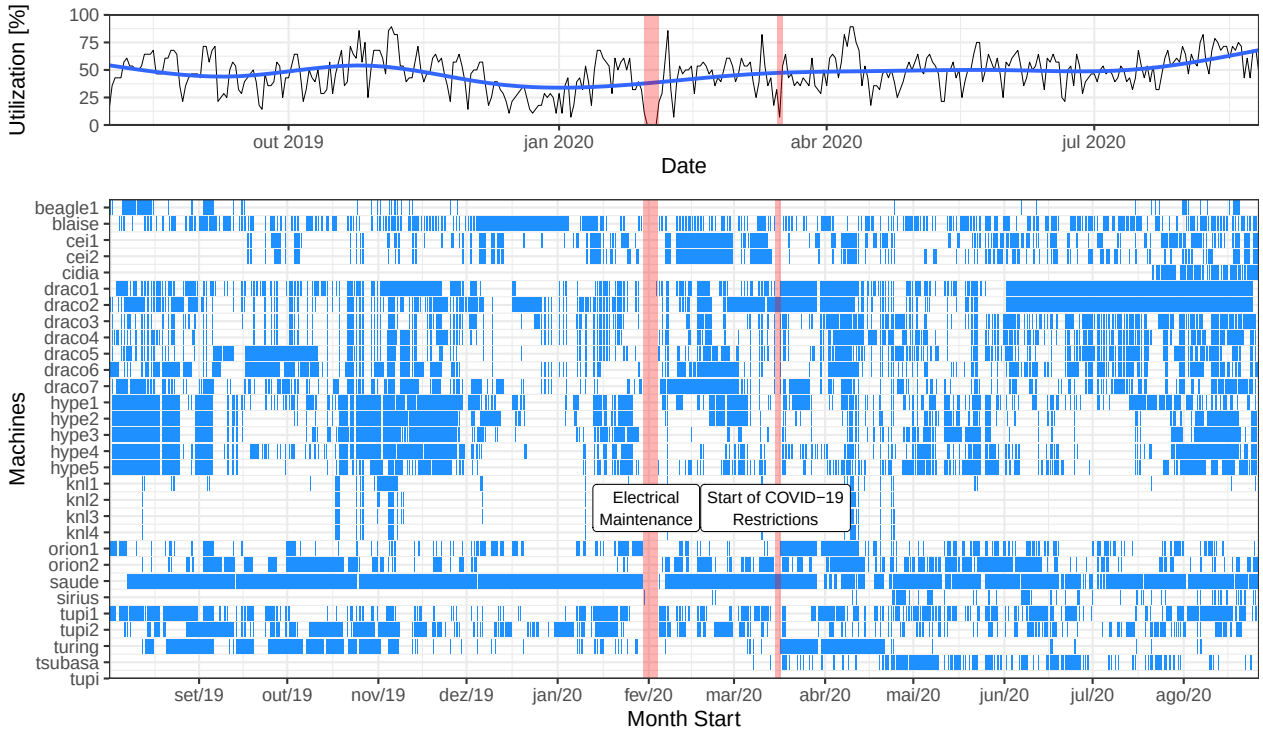

Fig. 4. Cluster aggregated total utilization and per machines Gantt chart jobs.

monitoring using the IPMI; (ii) software provisioning and configuration with Ansible; (iii) virtualization with Docker and Singularity; (iv) B-tree file system, which transparent compress data. Besides, we analyze the utilization during the last year, highlighting the effects throughout the COVID-19 pandemic. Future work includes evaluating new tools and more fine-grained analysis, like the number of jobs that utilize virtualization and energy consumption.

For more information about the GPPD–PCAD, please contact the team at gt-cluster-l@inf.ufrgs.br.

Acknowledgements

This study was financed by the "Coordenação de Aperfeiçoamento de Pessoal de Nível Superior" (CAPES) - Finance Code 001, the National Council for Scientific and Technological Development (CNPq), and the projects: FAPERGS MultiGPU (16/354-8), FAPERGS GreenCloud (16/488-9), FAPERGS Internacionalização (19/711-6), the CNPq project 447311/2014-0, the CAPES/Brafitec EcoSud 182/15, the CAPES/Cofecub 899/18, Petrobras (2016/00133-9 and 2018/00263-5), and donations by HPE and Intel under the *Lei da Informática* grants.

References

[1] J. Dongarra, S. Tomov, P. Luszczek, J. Kurzak, M. Gates, I. Yamazaki, H. Anzt, A. Haidar, and A. Abdelfattah, "With extreme computing, the rules have changed," *Comp. in Sci. Eng.*, vol. 19, no. 3, p. 52, 2017.

[2] A. B. Yoo, M. A. Jette, and M. Grondona, "Slurm: Simple linux utility for resource management," in *Workshop on Job Scheduling Strategies for Parallel Processing*. Springer, 2003, pp. 44–60.

[3] N. Capit *et al.*, "A batch scheduler with high level components," in *CCGrid 2005. IEEE International Symposium on Cluster Computing and the Grid, 2005.*, vol. 2. IEEE, 2005, pp. 776–783.

[4] D. Balouek *et al.*, "Adding virtualization capabilities to the Grid'5000 testbed," in *Cloud Computing and Services Science*, ser. Communications in Computer and Information Science, I. I. Ivanov, M. van Sinderen, F. Leymann, and T. Shan, Eds. Springer International Publishing, 2013, vol. 367, pp. 3–20.

[5] L. Hochstein and R. Moser, *Ansible: Up and Running: Automating Configuration Management and Deployment the Easy Way*. " O'Reilly Media, Inc.", 2017.

[6] V. Armenise, "Continuous delivery with jenkins: Jenkins solutions to implement continuous delivery," in *2015 IEEE/ACM 3rd International Workshop on Release Engineering*. IEEE, 2015, pp. 24–27.

[7] J. Loope, *Managing infrastructure with puppet: configuration management at scale*. " O'Reilly Media, Inc.", 2011.

[8] L. L. Nesi, M. S. Serpa, L. M. Schnorr, and P. O. A. Navaux, "GPPD – PCAD - HPC resources management infrastructure description and 10-month statistics," in *XVII Workshop de Processamento Paralelo e Distribuido*. GPPD, 2019, pp. 21–24.

[9] D. Merkel, "Docker: lightweight linux containers for consistent development and deployment," *Linux journal*, vol. 2014, no. 239, p. 2, 2014.

[10] G. M. Kurtzer, V. Sochat, and M. W. Bauer, "Singularity: Scientific containers for mobility of compute," *PloS one*, 2017.

[11] *IPMI-Intelligent Platform Management Interface Specification Second Generation*, Intel, Hewlett-Packard, NEC and DELL, 10 2013, Rev 1.1.

[12] O. Rodeh, J. Bacik, and C. Mason, "BTRFS: The linux B-tree filesystem," *ACM Transactions on Storage*, vol. 9, no. 3, pp. 1–32, 2013.

[13] M. Larabel, "Btrfs Zstd compression benchmarks on linux 4.14 - Phoronix," https://www.phoronix.com/scan.php?page=article&item=btrfs-zstd-compress&num=1, 2017, accessed: 2020-09-14.

Mitigating the Performance Degradation caused by Execution Units Contention via Instruction-Aware Mapping

Matheus S. Serpa[1], Eduardo H. M. Cruz[2], Matthias Diener[3], Antonio C. S. Beck[1], Philippe O. A. Navaux[1]
[1]Institute of Informatics, Federal University of Rio Grande do Sul, UFRGS, Brazil
[2]Federal Institute of Paraná, IFPR, Brazil
[3]University of Illinois at Urbana-Champaign, USA
{msserpa, caco, navaux}@inf.ufrgs.br, eduardo.cruz@ifpr.edu.br, mdiener@illinois.edu

***Abstract*—Parallel applications running on Simultaneous Multithreading (SMT) processors compete for execution units. This issue is aggravated when threads execute similar instructions such as branch, floating-point, integer, load, and store. In this case, the same kind of instruction is dispatched for execution, leading to sequential execution due to the contention of execution units, resulting in a performance loss. State-of-art focuses mostly on proposing techniques that reduce memory contention on multiprogrammed workloads, which share fewer resources and are mostly heterogeneous. This work proposes a transparent method that automatically maps threads of multiple parallel applications on SMT processors. We focused on improving performance by mitigating the contention on execution units when running parallel applications. To achieve that, we map threads that stress the same execution units to different cores. In order to evaluate our proposal, we use a microbenchmark and the OpenMP version of NAS Parallel Benchmarks (NPB). Results show performance gains of 29.1% and 17.4%, on average, compared to the native scheduler of the operating system and a Round-robin mapping, respectively.**

***Index Terms*—SMT processors, Performance degradation, Resource sharing, Execution unit contention.**

ACKNOWLEDGMENT

This work has been partially supported by Petrobras (2016/00133-9, 2018/00263-5) and Green Cloud project (2016/2551-0000 488-9), from FAPERGS and CNPq Brazil, program PRONEX 12/2014. This study was financed in part by the Coordenao de Aperfeioamento de Pessoal de Nvel Superior - Brasil (CAPES) - Finance Code 001.

Experiments presented in this paper were carried out using the Grid'5000 experimental testbed, being developed under the INRIA ALADDIN development action with support from CNRS, RENATER and several Universities as well as other funding bodies (see https://www.grid5000.fr).

DISCLAIMER

This abstract describes the paper entitled **Mitigating Execution Unit Contention in Parallel Applications using Instruction-Aware Mapping**, from the same authors submitted to the 22nd IEEE International Conference on High Performance Computing and Communications (HPCC 2020).

Assessing the Prefetcher's Role in High Performance Computing

Valéria S. Girelli[1], Francis B. Moreira[2], Matheus S. Serpa[1], Danilo Carastan-Santos[1], Philippe O. A. Navaux[1]
[1]Institute of Informatics, Federal University of Rio Grande do Sul (UFRGS) – Porto Alegre, Brazil
[2]Department of Informatics, Federal University of Paraná (UFPR) – Curitiba, Brazil
{vsgirelli, msserpa, danilo.csantos, navaux}@inf.ufrgs.br, fbmoreira@inf.ufpr.br

***Abstract*—Memory prefetchers have been broadly used in modern processors to mitigate the performance gap between processor and memory. In High Performance Computing (HPC) systems, many complications arise from parallelism, which are crucial to understand the prefetcher's impact over parallel applications' performance. In this work, we aim to evaluate the behavior of the prefetchers available in a Skylake machine and in two simulators of parallel architectures, ZSim and Sniper. Our results demonstrate that prefetching from the L3 to the L2 cache presents the best performance gains, and the prefetcher's efficiency is restrained by the memory contention that emerges from increasing the parallelism. Moreover, both ZSim and Sniper only simulate inclusive L3 caches, while Skylake's L3 is non-inclusive. Consequently, the outcome is a poor simulation of both the memory contention and the prefetcher behavior.**

***Index Terms*—Computer Architecture, Parallel Architecture, Prefetcher, Parallel Simulation.**

ACKNOWLEDGMENT

This work has been partially supported by Petrobras (2016/00133-9, 2018/00263-5) and Green Cloud project (2016/2551-0000 488-9), from FAPERGS and CNPq Brazil, program PRONEX 12/2014. This study was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Finance Code 001.

Experiments presented in this paper were carried out using the Grid'5000 experimental testbed, being developed under the INRIA ALADDIN development action with support from CNRS, RENATER and several Universities as well as other funding bodies (see https://www.grid5000.fr).

DISCLAIMER

This abstract describes the paper entitled "Understanding Memory Prefetcher Performance over Parallel Applications: From Real to Simulated" from the same authors, that has been submitted and is under review for publication on the journal Concurrency and Computation: Practice and Experience (CCPE).

Understanding Thread Mapping Policies Effects on Machine Learning Algorithms with TensorFlow

Matheus W. Camargo[1], Matheus S. Serpa[1], Danilo-Carastan-Santos[2]
Alexandre S. Carissimi[1], Philippe O. A. Navaux[1]
[1]Institute of Informatics, Federal University of Rio Grande do Sul, UFRGS, Brazil
{mwcamargo, msserpa, danilo.csantos, asc, navaux}@inf.ufrgs.br

***Abstract*—Machine Learning (ML) algorithms and models are successfully being used in various scientific and industrial applications. The emergence of more complex ML algorithms, combined with the increase in the amount of data available, lead to increasing demand for computational power. Studying ways to improve the performance of these algorithms, therefore, becomes an essential task. In this work, we investigate the performance effects of several thread mapping policies over new ML algorithms, built on top of TensorFlow. Using thread mappings such as scatter and compact policies, we achieved reductions on the execution time of both training and inference phases of the ML algorithms for up to 46% and 29%, respectively.**

***Index Terms*—Machine Learning, Thread Mapping, multi-core, TensorFlow**

ACKNOWLEDGMENT

This work has been partially supported by Petrobras (2016/00133-9, 2018/00263-5) and Green Cloud project (2016/2551-0000 488-9), from FAPERGS and CNPq Brazil, program PRONEX 12/2014. This study was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Finance Code 001.

DISCLAIMER

This abstract describes the paper entitled "Accelerating Machine Learning Algorithms with TensorFlow using Thread Mapping Policies" from the same authors, that has been accepted for publication on the Latin America Conference on High Performance Computing (CARLA 2020).

Attesting L-3 General Program Anomaly Detection Efficiency with SPADA

Francis Moreira*, Danilo Carastan-Santos† and Philippe Navaux‡
Department of Informatics, Federal University of Paraná
Curitiba, PR – Brazil
Email: *fbmoreira@inf.ufpr.br
Informatics Institute, Federal University of Rio Grande do Sul
Porto Alegre, RS – Brazil
Email: †danilo.csantos@inf.ufrgs.br ‡navaux@inf.ufrgs.br

***Abstract*—One of the main challenges for security systems is the detection of general vulnerability exploitation, especially when the exploit uses valid control flow. Thus, the detection of anomalous behavior provides an exciting research direction, as the research in this field tries to describe what is the standard program execution, to then detect as anomalous any behavior that does not fit that description.**

In this work, we compare two mechanisms that aim to detect general anomalies: SPADA and LAD. SPADA is an L-3 language mechanism that partitions phases and uses simple phase features to detect anomalies. LAD is a constrained L-1 language mechanism that applies complex clustering and machine learning models on specific functions to detect anomalies. In our experimental campaign with several real-world exploits, we show that SPADA's detection mechanism performs better than LAD while being much simpler and easier to implement. We therefore show experimental evidence that further attests the efficiency of L-3 attack detection mechanisms for real attacks.

Keywords*—*Computer Security, Intrusion Detection, Program Profiling.

ACKNOWLEDGMENT

The authors would like to thank CAPES and professor Israel Koren for the financial support during the PVE project nr. 117/2013 . This study was also financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Finance Code 001. We would also like to thank CNPq and Universidade Federal do Rio Grande do Sul (UFRGS) for the financial support.

DISCLAIMER

This abstract describes the paper "Attesting L-3 General Program Anomaly Detection Efficiency with SPADA", from the same authors, that has been published in the IEEE Symposium on Computers and Communications 2020 (ISCC '20).

Evaluating Parallel Sparse Solvers to Accelerate a Radiofrequency Ablation FEM Application

Marcelo Cogo Miletto*, Claudio Schepke†, Lucas Mello Schnorr*
* Graduate Program in Computer Science (PPGC/UFRGS), Porto Alegre, Brazil
† UNIPAMPA, Alegrete, Brazil

***Abstract*—Solving sparse linear systems lies at the heart of many scientific applications, like numerical applications and computer simulations. Such applications can be incredibly time-consuming, demanding lots of computing power to calculate precise results in a feasible amount of time. The Finite Element Method (FEM) is a universal approach to simulating various problems, discretizing the whole system as a finite number of simple-shaped geometric elements. FEM applications are characterized by common steps, like assembling a linear equation system to approximate an unknown function over the discretized domain, leading to a very sparse system of equations that needs to be solved. Hence, the evolution in high-performance computing scenarios through newer parallel hardware and software has been supplying the ever-increasing demand for computational power arising from such applications. An example of a computational simulation application that uses FEM is the Radiofrequency Ablation Finite Element Method (RAFEM), which simulates the radiofrequency ablation procedure, a common treatment for hepatic cancer. We use the RAFEM application in this work as a case study. RAFEM's original version runs sequentially and takes up to 20 hours to produce 15 minutes of simulated results. We use three sparse solvers (MAGMA, cuSOLVER, and QRMumps) among different multicore and GPU architectures to accelerate the application. We investigate the numerical properties of the results provided by each solver using the peak signal-to-noise ratio (PSNR) metric. We also present a detailed performance analysis of the application in different machines using data from application tracing, pointing out the sparse solver role in the overall performance. The acceleration results showed that we managed to reduce the computing time of the original version up to 40 times while keeping numerical results sufficiently close to the original version values.**

ACKNOWLEDGEMENTS

We thank these projects and institutions for supporting this investigation: FAPERGS MultiGPU (16/354-8) and GreenCloud (16/488-9), the CAPES/Brafitec EcoSud 182/15, the Council for Scientific and Technological Development (CNPq) under grant no 131347/2019-5 to the first author, the CAPES/Cofecub 899/18, and the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Finance Code 001. Some experiments in this work used the PCAD infrastructure, http://gppd-hpc.inf.ufrgs.br, at INF/UFRGS.

DISCLAIMER

This abstract describes the article entitled "Optimization of a Radiofrequency Ablation FEM Application Using Parallel Sparse Solvers", from the same authors, that has been submitted in August 2020 to the 18th International Conference on High Performance Computing & Simulation (HPCS).

Performance and Portability of Seismic Imaging on Multicore and GPU Architectures

Matheus S. Serpa[1], Pablo J. Pavan[1], Eduardo H. M. Cruz[2], Alexandre S. Carissimi[1], Philippe O. A. Navaux[1]
[1]Institute of Informatics, Federal University of Rio Grande do Sul, UFRGS, Brazil
[2]Federal Institute of Paraná, IFPR, Brazil
{msserpa, pjpavan, asc, navaux}@inf.ufrgs.br, eduardo.cruz@ifpr.edu.br

***Abstract*—Reverse Time Migration (RTM) is a seismic imaging method used by the Oil & Gas industry. Petrobras, Shell and Total, companies that owned The Libra oil field, have ported their applications to High-Performance Computing architectures, providing more accurate results using less time. However, there are important decisions like choosing the architecture and the parallel programming API, which are strongly related to the programming effort, the performance and energy efficiency of the simulations. In this work, we investigate the impact of parallel programming APIs on the performance, portability and energy efficiency of seismic imaging simulations on multicore and GPU architectures. Our results show that, for seismic methods, the recommendation is programming using the OpenACC API for GPU architectures.**

***Index Terms*—Performance Optimization, Oil and Gas Simulation, Seismic Imaging, Reverse Time Migration, Code Portability.**

ACKNOWLEDGMENT

This work has been partially supported by Petrobras (2016/00133-9, 2018/00263-5) and Green Cloud project (2016/2551-0000 488-9), from FAPERGS and CNPq Brazil, program PRONEX 12/2014. This study was financed in part by the Coordenao de Aperfeioamento de Pessoal de Nvel Superior - Brasil (CAPES) - Finance Code 001.

Experiments presented in this paper were carried out using the Grid'5000 experimental testbed, being developed under the INRIA ALADDIN development action with support from CNRS, RENATER and several Universities as well as other funding bodies (see https://www.grid5000.fr).

DISCLAIMER

This abstract describes the article entitled **Energy Efficiency and Portability of Oil and Gas Simulations on Multicore and GPU Architectures** from the same authors submitted and under review for publication on the journal Concurrency and Computation: Practice and Experience (CCPE).

Optimization Strategies for Scientific Applications on SX-Aurora TSUBASA

Félix Dal Pont Michels Júnior, Matheus da Silva Serpa, Danilo Carastan Santos
Lucas Mello Schnorr, Philippe Olivier Alexandre Navaux
Institute of Informatics, Federal University of Rio Grande do Sul, UFRGS, Brazil
{felix.junior, msserpa, danilo.csantos, schnorr, navaux}@inf.ufrgs.br

***Abstract*—A large number of architectures, while attractive and flexible, poses new challenges for the programmer. The memory subsystem, number of cores, and other design decisions increase the difficulty of implementing efficient applications. Some of these issues have been transparently reduced by compilers to the programmer. However, others require source code modifications that differ according to the application and architecture. This work analyzes the performance of loop unrolling and function inlining optimization techniques applied to a vector processor, the SX-Aurora TSUBASA architecture. As a case study, we apply these techniques to the NAS Parallel Benchmarks and an Oil & Gas real-world application, the Reverse Time Migration (RTM). Our experimental results show performance improvements of up to $1,9\times$ compared with the original parallel implementation.**

***Index Terms*—Performance Optimization, NEC SX-Aurora TSUBASA, Reverse Time Migration, Loop unrolling, Inlining.**

ACKNOWLEDGMENT

This study was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Finance Code 001, Petrobras's project (2016/00133-9, 2018/00263-5) and "GREEN-CLOUD: Computação em Cloud com Computação Sustentável" (#16/2551-0000 488-9) project, from FAPERGS and CNPq, PRONEX program 12/2014.

DISCLAIMER

This abstract describes the article entitled "Otimização de Aplicações Paralelas em Aceleradores Vetoriais NEC SX-Aurora", from the same authors, that has been submitted in August 2020 to the XXI Simpósio em Sistemas Computacionais de Alto Desempenho (WSCAD 2020).