Modeling Parallel AI Applications for Performance Analysis on Cloud Environments

Miquel Albert[1][0009−0004−0781−5656], Alvaro Wong[2][0000−0002−8394−9478], Betzabeth Leon[2][0000−0003−1778−0237], Dolores Rexachs[2][0000−0001−5500−850X], and Emilio Luque[2][0000−0002−2884−3232]

[1] Escoles Universitaries Gimbernat (EUG), Computer Science School, Universitat Autonoma de Barcelona, 08174 Sant Cugat del Valles, Spain
[2] Department of Computer Architecture and Operating Systems, Universitat Autonoma de Barcelona, 08193 Bellaterra, Spain

Abstract. In high-performance computing (HPC) environments, efficient execution of AI applications is critical for optimal performance and resource utilization. In this work, we extend the PAS2P methodology to AI applications through message passing on HPC Cloud systems, defining the AI Application Model to describe their performance behavior. This extension identifies phases within AI applications, enabling analysis to focus on these phases instead of the entire application. By concentrating on them, we can better evaluate AI application efficiency, providing insights into system performance and guiding future optimizations for large-scale AI tasks on HPC infrastructure.

Keywords: Performance in HPC · Cloud Computing · AI Applications

1 Introduction

In recent years, the growing demand for AI applications has increased the need for HPC systems to efficiently run large-scale AI workloads [9]. These applications, especially in deep learning and distributed training, require significant computational resources and optimized strategies. However, as AI models grow in complexity, assessing their efficient resource usage has become a major challenge. The problem lies not only in performing a comprehensive performance analysis but also in the uncertainty of whether applications use resources efficiently.

PAS2P [8] (Parallel Application Signature for Performance Analysis and Prediction) is a methodology originally designed to analyze and predict the performance of parallel scientific applications. PAS2P instruments the application to collect performance data from application processes, which are then analyzed to generate an abstract model of the behavior of the application. This model identifies different phases in the execution of the application, each representing a specific segment of parallel code. By analyzing these phases, PAS2P predicts the performance of the application by executing an application signature.

In this work, we extend the Parallel Application Signature for Performance Analysis and Prediction (PAS2P) methodology, originally designed for scientific

ICCS Camera Ready Version 2025
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-97635-3_26

parallel applications, to AI applications running on HPC systems, including those in cloud computing environments. We propose extending the PAS2P methodology to model the performance of AI applications, enabling the identification of phases within AI workloads. By isolating the phases, we can better analyze the resource efficiency of AI applications, focusing on areas where optimization could have the most significant impact. Quickly evaluating the efficiency of an AI application through its signature helps select cloud resources matched to its performance needs, enabling better resource allocation and optimizing performance and cost.

We have analyzed the MNIST dataset[5], the ResNet-50 model[2], using Horovod[7], k-means[4], a machine learning algorithm for unsupervised learning, DeepGalaxy and and pinn-mpi, Physics-informed neural networks (PINNs) are widely used to solve forward and inverse problems in fluid mechanics. The paper is organized as follows: Section 2 provides the related works. Section 3 describes the methodology proposed for AI programs. Section 4 presents the model validation followed by a conclusion and future work in Section 5.

2 Related Works

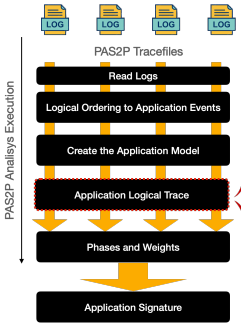
In the realm of HPC for AI applications, several approaches have been developed to evaluate the performance of distributed CPU-based workloads. The growing complexity of AI applications, particularly in the areas of deep learning and distributed training, requires efficient techniques to analyze and optimize their performance. Awan et al.[1] analyzed the communication and computational performance of distributed AI applications, focusing on the efficiency of CPU communication in multinode clusters using MPI. Their approach is centered on detailed profiling of the communication workloads using high-performance interconnects like InfiniBand. However, their method involves intensive, time-consuming profiling processes that can be computationally expensive. In contrast, the application signature methodology provides a much faster analysis by focusing on identifying phases and their recurrence, without requiring such detailed profiling.

In application performance benchmarking, several frameworks target AI applications. Tartan [6] uses a multi-GPU benchmark suite, requiring selection of a benchmark that approximates application behavior. Similarly, HPC AI500 [3] evaluates HPC systems for AI workloads with benchmarks for deep learning. In contrast, the signature models application behavior directly, predicting execution time accurately without external benchmarks. It runs in a bounded time frame, making it more efficient and precise. Unlike Tartan and HPC AI500, which rely on selected benchmarks, the application signature focuses on the application's own performance characteristics, making it more adaptive and accurate.

3 IA Paralel Application Model

In distributed AI applications, workload is designed to be uniform as possible, but data partitioning and dynamic training cause computational load variations

ICCS Camera Ready Version 2025
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-97635-3_26



Type of communication:
1 : MPI_Allreduce, 2 : MPI_Allgather

	Process 0			Process 1			Process 2			Process 3		
Tick	Type of comm	Comm. Size	Compute Instruction	Type of comm	Comm. Size	Compute Instruction	Type of comm	Comm. Size	Compute Instruction	Type of comm	Comm. Size	Compute Instruction
0	1	4	70277	1	4	54689	1	4	57691	1	4	46797
1	2	8	89502	2	8	169396	2	8	120561	2	8	255036
2	2	8	132709	2	8	141746	2	8	125742	2	8	131872
3	2	8	22676004	2	8	21280147	2	8	42877	2	8	20335011
4	...											

Fig. 1. Logical Trace: Abstract view enabling phase identification.

between processes, leading to differences in executed instructions and challenges in identifying similar phases. We present a methodology to adapt the PAS2P model for detecting execution phases that exhibit similar computational characteristics, despite the inherent differences in workloads across iterations, where each iteration may present slightly different workloads due to the dynamic data distribution. However, the current PAS2P methodology is designed for more deterministic HPC applications. If one process executes a different number of instructions in each iteration, this results in finding multiple phases, as each slight variation is treated as a distinct pattern.

To address this issue, we modeled the PAS2P methodology by designing a matching mechanism to detect phases with similar computational characteristics, despite the differences in workload distribution. This allows us to group phases that share common computational characteristics, even if the number of instructions and CPU times differ slightly between processes. The result of this adaptation is that the phases identified by our model can be used to efficiently analyze performance without being influenced by an excessive number of phases. This approach allows us to identify significant phases that impact performance.

The PAS2P methodology starts by instrumenting the application to extract performance data, saved in trace files. After collecting traces, we generate a logical trace that captures the application's behavior, consisting of communication and computation events. Events like MPI_Allgather and MPI_Allreduce are logged with details such as event ID, data size, timestamp, computational time, and instruction count. These events reveal how the application uses CPU and communication resources across processes.

After collecting performance traces with PAS2P instrumentation, the next step is to generate a logical trace that captures the application's behavior. As shown in Figure 1, this trace organizes MPI events by logical time, including key operations like MPI_Allgather and MPI_Allreduce. It records details such as communication size and instructions executed per event. Ordering events by logical time creates an abstract performance model, facilitating the identification of patterns or phases during analysis.

The challenge in analyzing the application lies in the variability of instructions executed by each process across iterations, due to factors like data parti-

ICCS Camera Ready Version 2025
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-97635-3_26

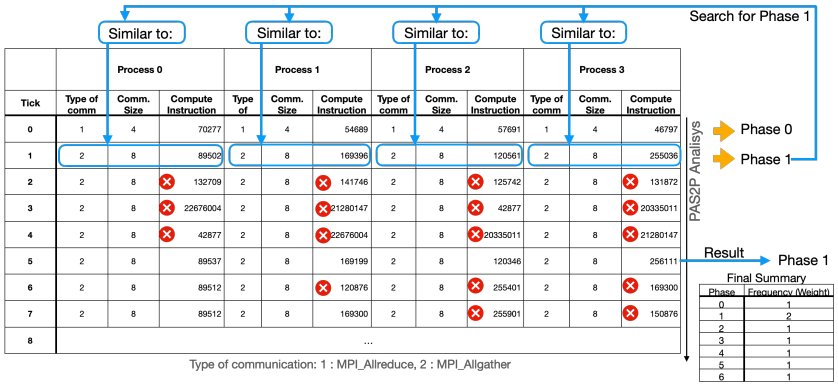


Fig. 2. More processes increase identified phases due to instruction count variations.

Table 1. DeepGalaxy Logical Trace segment with frequencies and SIMO values.

Tick	Processes								Frequency	SIMO	Tick	Processes								Frequency	SIMO
	0	1	2	3	4	5	6	7				0	1	2	3	4	5	6	7		
0	6	7	6	7	4	6	7	7	0	50	20	6	6	6	5	6	5	7	7	0	48
1	7	6	7	6	6	5	7	7	0	51	21	6	6	5	6	5	6	5	7	0	47
2	6	5	6	6	6	5	7	6	0	47	22	6	5	6	6	6	6	5	7	0	48
3	6	6	6	6	4	5	7	6	0	46	23	7	5	6	6	6	6	7	7	0	50
4	6	6	6	5	4	5	7	6	0	45	24	6	6	5	5	6	6	7	6	0	47
5	6	6	6	5	5	6	7	6	0	47	25	6	6	5	5	6	6	7	6	0	48
6	6	6	6	5	5	5	7	6	0	46	26	6	6	5	5	6	6	5	7	0	46
7	7	7	6	7	5	5	7	7	0	51	27	6	6	5	5	6	5	6	6	0	46
8	7	6	6	7	5	5	7	7	0	50	28	6	6	5	5	5	5	6	6	0	44
9	5	5	6	6	8	8	8	8	0	54	29	7	5	7	5	6	5	7	7	0	49
10	7	8	8	8	8	8	8	8	0	63	30	7	5	6	6	5	6	7	7	0	48
11	5	5	5	5	6	7	6	6	0	45	31	7	5	7	6	7	6	7	7	0	52
12	5	6	5	6	6	7	6	6	0	47	32	7	5	7	5	7	5	7	7	0	50
13	5	6	5	5	5	6	7	6	0	46	33	7	5	7	5	7	6	7	7	0	51
14	5	6	5	6	5	6	7	6	0	46	34	8	8	8	7	8	7	8	8	0	62
15	6	6	5	5	6	7	6	6	0	47	35	8	7	8	7	6	5	6	6	0	53
16	5	6	5	6	6	6	5	5	0	44	36	8	5	8	8	8	7	8	8	0	60
17	6	7	6	6	5	6	6	7	0	49	...					...					
18	6	7	6	6	7	7	6	6	0	51											
19	7	6	6	5	6	7	7	7	0	50											

tioning and workload distribution. As processes receive different data amounts, they execute varying instruction counts, making phase identification difficult. Each process may behave differently depending on its data, leading to many phases — a common issue in AI and deep learning models, where data and workload distribution often fluctuate between iterations.

Current version of PAS2P compares the communication type, communication size, and the number of instructions executed by different processes to identify phases. However, due to the variability in instruction counts across iterations, the conventional approach of using a distance metric to compare instruction counts can result in the identification of numerous phases that are not recognized as similar.

As the number of processes increases, the complexity of identifying phases also increases, as shown in Figure 2. Each phase is represented by a segment of events across all processes, and this distribution of events makes it harder to identify similar phases. In this example, as more processes are added, PAS2P struggles to group similar phases, especially when instruction counts vary. This variability leads to the identification of more phases, resulting in a higher number of phases being recognized, even when the phases may be functionally similar.

ICCS Camera Ready Version 2025
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-97635-3_26

Tick	SIMO	Group By SIMO Phase ID	Group By SIMO Tolerance +-1 Phase ID	Frequency	Tick	SIMO	Group By SIMO Phase ID	Group By SIMO Tolerance +-1 Phase ID	Frequency
0	50	1	1	5	20	48	10	2	4
1	51	2		4	21	47	3		6
2	47	3	2	6	22	48	10		4
3	46	4		6	23	50	1	1	5
4	45	5	3	2	24	47	3	2	6
5	47	3	2	6	25	48	10		4
6	46	4		6	26	46	4		6
7	51	2	1	4	27	46			6
8	50	1		5	28	44	8	3	2
9	54	6	4	1	29	49	9	1	2
10	63	7	5	1	30	48	10	2	4
11	45	5	3	2	31	52	11	6	1
12	47	3	2	6	32	50	1	1	5
13	46	4		6	33	51	2		4
14	46			6	34	62	12	5	1
15	47	3		6	35	53	14	4	1
16	44	8	3	2	36	60	15	7	1
17	49	9	1	2	...	...			
18	51	2		4					
19	50	1		5					

Final Summary

Phase	Frequency (Weight)
1	11
2	14
3	4
4	2
5	6
6	1
7	1

Fig. 3. SIMO calculation for a DeepGalaxy trace. Grouping SIMO values into phases and applying ±1 tolerance reduces phases from 15 to 7.

Once the problem was identified, as shown in Table 1, we use a segment of the logical trace from the DeepGalaxy IA application with Horovod and eight processes. The table focuses on the number of instructions executed by each process at each tick. The challenge in identifying similar phases lies in how PAS2P models computation. To address this, we model instruction counts by their order of magnitude, considering segments similar if they share the same order, even with slight variations. After analyzing the trace, we add a column to specify repetitions (weights/frequencies) for each pattern. However, even using the order of magnitude, we did not find similarities between phases, as each tick represents a phase.

As we know, applications aim to balance the workload as evenly as possible across processes, though this depends on the data being processed. In some cases, certain processes must handle more data than others, and this balance can change unpredictably with each iteration. However, we also know that the total workload across all processes remains fixed. That is, the sum of the workload across each process gives the total computational load, as shown by the following equation (where $Compute_i$ represents the workload of each process i, and N is the number of processes):

$$\text{Total Workload} = \sum_{i=1}^{N} Compute_i$$

Building on this, what we observed is that by summing the instructions of a specific phase across all processes, represented by SIMO (Sum of Instructions in Magnitude Order), we can find the total computational load for each phase. As shown in Table 1, the value of SIMO in the table begins to show similarity at times, which helps to analyze the total computational load for the application. The sum of instructions for each phase can give us a better understanding of the workload distribution and its variability between different processes.

ICCS Camera Ready Version 2025
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-97635-3_26

Once we calculate the SIMO, as shown in Figure 3, the next step is to process the results for phase identification. We first remove the columns related to patterns from the previous figure, focusing on grouping patterns by instruction count similarity. In the Group By SIMO Phase ID column, we assign a unique Phase ID to each set of equal SIMO values. Patterns belong to the same phase if their SIMO values match across processes. In this DeepGalaxy trace, we identified up to 16 distinct phases, showing patterns with equal computational behavior.

However, our goal in performance analysis is not only to find equal patterns but to identify phases meaningful in computational load. To refine this, we introduce a ± 1 tolerance for SIMO values in the Group By SIMO Tolerance ± 1 Phase ID column. This tolerance groups patterns with slight instruction count variations but similar computational load. Applying this reduces the number of identified phases from 15 to 7, as shown in the Summary in Figure 3. Furthermore, PAS2P performs a classification of the identified phases, discarding those that are considered insignificant in terms of performance: phases whose execution time considering their weight constitutes less than 1% of whole execution.

4 Experimental Results

To validate our proposed methodology, we selected a set of AI applications that utilize the Horovod framework. Among them is a ResNet50 model[2], trained for image classification. Another application involves training a model to recognize handwritten digits[5]. We also included the DeepGalaxy application, which simulates galaxy clusters, and trained using 4 to 16 processes. Additionally, we run KMeans clustering, implemented with library mpi4py. These programs were run on two AWS instance types: c7a.4xlarge, with 16 vCPUs, 32 GiB memory, up to 12.5 Gbps network, and AMD EPYC 9R14 processor and c5.9xlarge, with 36 vCPUs, 72 GiB memory, 12 Gbps network, and Intel Xeon 8124M CPU.

We ran the application as shown in Table 2, presenting the results obtained after applying the proposed model to DeepGalaxy with 4 processes. Each table shows: the phase identifier, the CPU time per phase, and the frequency or weight, indicating how many times each phase is executed. By multiplying the CPU time by its weight, we can extrapolate the total computation time for each phase and predict the application's overall execution time. The final row displays the Computational Total Time, summing all phase times multiplied by their weights.

Table 2. Phases obtained for DeepGalaxy 4 Processes

Phase	CPU Time(Sec.)	Frequency	CPU Time * Frequency (Sec.)
1	0.000023	2,935	0.067495
2	0.000048	470,550	22.545653
3	0.000037	137,632	5.057869
4	0.000034	202,724	6.828968
5	0.000027	6,550	0.179555
6	0.000634	27,371	17.365881
7	0.000629	152	0.095631
8	0.001395	35	0.048816
Computational Total Time			52.189868

ICCS Camera Ready Version 2025
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-97635-3_26

Table 3. Phase Detection and Execution Time Comparison with the PAS2P Model.

Application	Number of processes	Previous number of phases	Current number of phases	Application Execution Time (AET) (Sec.)	Percentage of AET in Comm. Stage	Percentage of AET in Compute Stage
Resnet 50	8	7,079	16	653.55	97.39	2.61
Resnet 50	16	23,993	35	854.83	97.30	2.70
MNIST	4	419	9	555.50	99.42	0.58
MNIST	8	5,335	15	307.73	99.62	0.38
DeepGalaxy	4	285	8	844.24	94.61	5.39
DeepGalaxy	8	4,929	16	581.54	95.33	4.67
K-Means	8	175	16	751.02	0.20	99.80
K-Means	16	526	28	535.23	0.40	99.60
pinn-mpi	16	7	4	642.435	2.81	97.19
Resnet 50	32	80,948	48	455.48	94.12	5.88
MNIST	32	49,855	37	302.79	94.01	5.99
DeepGalaxy	16	18,627	22	201.33	95.70	4.30
K-Means	64	1,258	28	1503.71	2.64	97.36
pinn-mpi	32	6	4	633.671	4.47	95.53
Resnet 50	64	43,558	37	285.349	96.47	3.53
MNIST	64	47,192	49	403.82	98.72	1.28
K-Means	128	1,575	30	1074.31	7.86	92.14
pinn-mpi	64	7	6	627.75	2.51	97.49

This predictive model condenses the application into a few key phases, enhancing the efficiency of performance analysis.

To validate our methodology, we applied the same analysis to all applications in this study. As shown in Table 3, we ran the applications with 4 to 16 processes across 1 and 4 nodes. The table compares the number of phases detected by the previous PAS2P version ("Previous Number of Phases") and the proposed model ("Current Number of Phases"). This reduction will allow for a quicker execution of the signature as it will consist of a smaller set of phases.

Additionally, we performed an analysis on the reduced set of phases. Using the phase information, we calculated the percentage of the total time that applications spent in communications, represented in the column "Percentage of AET in Comm. Stage", and the percentage of time spent in computation, shown in the "Percentage of AET in Compute Stage" column. The results reveal that, for applications using Horovod, most of the time is dedicated to communications, while the time spent on computation is significantly lower. For k-means clustering, the compute stage dominates, accounting for nearly 99% of execution time. This shows k-means is highly compute-intensive with minimal inter-process communication. Thus, cloud instance optimization should prioritize computational power over communication bandwidth, unlike AI training workloads. This insight aids resource management by guiding instance selection based on application reliance on communication or CPU.

5 Conclusions and Future Works

In this work, we proposed extending the PAS2P methodology to model AI applications in HPC Cloud environments. By identifying phases, we enable performance analysis without evaluating the entire application, helping understand resource use and predict execution times. Our results show the methodology effectively models application behavior, reducing phases to improve analysis efficiency and identify inefficiencies.

ICCS Camera Ready Version 2025
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-97635-3_26

By analyzing the computational load and communication patterns of each phase, we can identify resource-intensive phases to guide cloud instance selection. For AI applications with Horovod, the communication stage consumes a large part of execution time. The next step is to keep applying this methodology to understand resource usage and explain why applications spend more time in communication or computation, aiding resource allocation and improving performance. As part of future work, we will analyze more AI applications and explore the extension of this methodology to other communication libraries, such as NVIDIA NCCL (NVIDIA Collective Communications Library). This will further improve the methodology's applicability to various AI workloads.

6 Acknowledgment

This research has been supported by the Agencia Estatal de Investigación (AEI), Spain and the Fondo Europeo de Desarrollo Regional (FEDER) UE, under contracts PID2020-112496GB-I00 and PID2023-146978OB-I00.

References

1. Awan, A.A., Jain, A., Chu, C.H., Subramoni, H., Panda, D.K.: Communication profiling and characterization of deep learning workloads on clusters with high-performance interconnects. In: IEEE Symposium on High-Performance Interconnects (HOTI). pp. 49–53. IEEE (2019)
2. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 770–778 (2016)
3. Jiang, Z., Gao, W., Wang, L., Xiong, X., Zhang, Y., Wen, X., Luo, C., Ye, H., Lu, X., Zhang, Y., Feng, S., Li, K., Xu, W., Zhan, J.: Hpc ai500: A benchmark suite for hpc ai systems. In: Benchmarking, Measuring, and Optimizing: First BenchCouncil International Symposium, Bench 2018, Seattle, WA, USA. p. 10–22. Springer-Verlag, Berlin, Heidelberg (2018)
4. Jin, X., Han, J.: K-Means Clustering, pp. 563–564. Springer US, Boston, MA (2010), https://doi.org/10.1007/978-0-387-30164-8_425
5. LeCun, Y., Bottou, L., Bengio, Y., Haffner, P.: Gradient-based learning applied to document recognition. Proceedings of the IEEE **86**(11), 2278–2324 (1998). https://doi.org/10.1109/5.726791
6. Li, A., Song, S.L., Chen, J., Liu, X., Tallent, N., Barker, K.: Tartan: Evaluating modern cpu interconnect via a multi-cpu benchmark suite. IEEE Transactions on Parallel and Distributed Systems **30**(7), 1358–1370 (2018). https://doi.org/10.1109/TPDS.2018.2805956
7. Sergeev, A., Del Balso, M.: Horovod: Fast and easy distributed deep learning in tensorflow. arXiv preprint arXiv:1802.05799 (2018), https://arxiv.org/abs/1802.05799
8. Wong, A., Rexachs, D., Luque, E.: Parallel application signature for performance analysis and prediction. IEEE Transactions on Parallel and Distributed Systems **26**(7), 2009–2019 (2015)
9. Yi, G., Loia, V.: High-performance computing systems and applications for ai. The Journal of Supercomputing **75**(8), 4248–4251 (2019), https://doi.org/10.1007/s11227-019-02937-z

ICCS Camera Ready Version 2025
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-97635-3_26