Efficiency Analysis for AI applications in HPC systems. Case study: K-Means

Jose Rivas[1][0009−0004−1725−7133], Alvaro Wong[1][0000−0002−8394−9478], Remo Suppi[1][0000−0002−0373−8292], Emilio Luque[1][0000−0002−2884−3232], and Dolores Rexachs[1][0000−0001−5500−850X]

Universitat Autònoma de Barcelona Barcelona 08193, Spain
{jose.rivas}@autonoma.cat
{alvaro.wong, remo.suppi, emilio.luque, dolores.rexachs}@uab.es

Abstract. Currently, many AI applications require large-scale computing and memory to solve problems. The combination of AI applications and HPC sometimes does not efficiently use the available resources. Furthermore, a lot of idle times is caused by communication, resulting in increased runtime. This paper describes a methodology for AI parallel applications that analyses the performance and efficiency of these applications running on HPC resources to make decisions and select the most appropriate system resources. We validate our proposal by analysing the efficiency of the K-Means application obtaining an efficiency of 99% on a target machine.

Keywords: AI applications · HPC performance and efficiency · PAS2P methodology.

1 Introduction

In recent years, the convergence of High Performance Computing (HPC) and Artificial Intelligence (AI) has become increasingly relevant as the advanced HPC technology has enhanced the processing power needed for large-scale AI applications [1][2]. HPC systems have traditionally been used for scientific simulations and modelling. Nevertheless, with the rise of AI, these systems can now be leveraged for complex AI workloads such as deep learning or neural networks.

As these applications become more complex and are used on HPC systems, it is crusial tu ensure that the AI applications are running efficiently. One problem that parallel AI applications face is ensuring that they are utilising the full processing power of HPC systems on which they are executed.

We propose to provide information about these applications' performance and identify segments of the program that could be improved. To achieve this objective, we first evaluate the PAS2P (Parallel Application Signature for Performance Prediction) tool used to predict the performance of a parallel scientific application as well as to analyse if it is possible to generate a model for AI applications.

ICCS Camera Ready Version 2023
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-36021-3_39

The PAS2P methodology[9] is based on characterising the dynamic behaviour of MPI applications on their execution. PAS2P instruments the application to analyse the events it has captured and search for repetitive patterns defined as phases. Each phase is assigned a weight determined by the number of times a pattern repeats. For the performance prediction, PAS2P generates a signature that is constituted by phases. When executed, we obtain the execution times of each phase. By multiplying these times by the weights, we obtain the execution time prediction.

With the PAS2P tool, we can characterise an application in a reduced set of phases, which allows us to focus the efficiency analysis on the phases of the application and later extrapolate this analysis to the entire application. However, to validate our proposal, it is necessary to analyse the application model generated by PAS2P for AI applications in order to analyse the efficiency of AI applications on HPC systems so as to produce a comprehensive report highlighting the areas (phases) in the code that can be improved. In this case, we apply the efficiency analysis to the K-Means application.

This paper is organised as follows: Section 2 provides an overview of related works in the realm of AI application performance on HPC systems and previous PAS2P works to characterise scientific applications in HPC environments. Section 3 presents how PAS2P models the AI applications, and the proposed methodology for the efficiency analysis outlines the approach and the three steps taken in the study. Section 4 presents the efficiency analysis results applied to a K-Means application. Finally, in the last section, we offer the conclusions and propose future work.

2 Related works

There are tools related to the performance of AI parallel applications running on HPC systems. For example, Z. Fink et al. [4], the autors focus on evaluating the performance of two Python parallel programming models: Charm4Py [5], and mpi4py [6]. The authors argue that Python is rapidly becoming a common language in machine learning and scientific computing, and several frameworks scale Python across nodes. However, more needs to be known about their strengths and weaknesses.

N. Alnaasan et al. [7] introduce OMB-Py, a Python micro-benchmark for evaluating MPI library performance on HPC systems. The authors argue that Python has become a dominant programming language in emerging areas such as machine learning, deep learning, and data science. The paper proposes OMB-Py, Python extensions of the open-source OSU Micro-Benchmark (OMB) suite, in order to evaluate the communication performance of MPI-based parallel applications in Python. There are other proposals on the importance of evaluating and improving the performance of AI applications in HPC environments [3]. However, one of the main difficulties with benchmarks is selecting the benchmark most similar to the application you want to evaluate on a specific system. The difference between these works and ours is that with PAS2P, we obtain the

ICCS Camera Ready Version 2023
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-36021-3_39

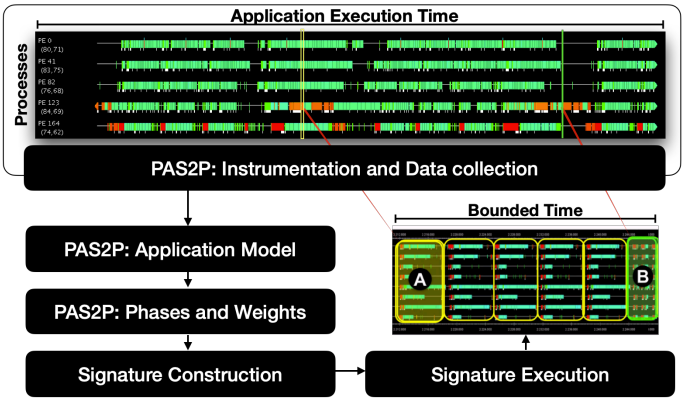

Fig. 1. Overview of PAS2P Methodology.

application's benchmark (Application Signature) executed in a bounded time representing the application behaviour.

In previous work on PAS2P[9][10], methodologies were developed which focused on both applications of SPMD scientific computing and an extension of PAS2P for applications with irregular behaviour, both for HPC environments.

As shown in Figure 1, PAS2P instruments and runs the application in a target machine, producing trace logs. The data obtained is used to characterise the behaviour of computing and communication events that describe the application behaviour. First, PAS2P assigns a global logical clock to the event to obtain the application model according to the causal relationships between the communication events. Then, once PAS2P has the application model, it identifies and extracts the sequences of the most relevant events defined as phases.

Once we have the phases, PAS2P assigns a weight to each phase, defined as the number of times the phase occurs during the execution. Finally, the signature is represented by a set of phases that can be executed to measure the application performance. For performance prediction, the signature execution in different target systems allows us to measure the execution time of each phase. Therefore it calculates the runtime of the entire application in each of those systems. This is achieved by using equation (1), where the $PhaseET_i$ is the estimated time for each phase i, and W_i is the weight for each phase i.

$$PET = \sum_{i=1}^{n} (PhaseET_i)(W_i) \tag{1}$$

3 Efficiency Analysis Model over Application Phases

With the recent advancements in HPC hardware and the surge in AI, we propose to extend the contributions of the PAS2P methodology to the field of AI. To evaluate the PAS2P application model and propose an efficiency model, we take the

ICCS Camera Ready Version 2023
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-36021-3_39

Table 1. Set of executions carried out with a K-Means application.

Exp. ID	Data seed	Centroid seed	Sites per process	Exp. ID	Data seed	Centroid seed	Sites per process
1	31359	5803	522500	7	31359	5803	400250
2	31359	18036	522500	8	31359	18036	400250
3	2450	5803	522500	9	2450	5803	400250
4	2450	18036	522500	10	2450	18036	400250
5	19702	5803	522500	11	19702	5803	400250
6	19702	18036	522500	12	19702	18036	400250

AI application K-Means, an unsupervised classification (clustering) algorithm that groups objects into k groups based on their characteristics[8]. Clustering is carried out by minimising the sum of distances between each object and the centroid of its group or cluster. The quadratic distance is often used.

There is a need to expand the concept of the performance of a parallel application beyond simply predicting its execution time. The previous PAS2P methodology primarily focuses on predicting execution time as a performance measure. We then propose an extension to the meaning of efficiency by defining the performance of a phase as the ratio between the computational time and its total execution time.

As per our proposal, first, we need to evaluate if PAS2P can characterise the AI application. To validate the characterisation, we instrument and analyse the execution of the K-Means application using PAS2P and construct the application signature. We suppose that this signature accurately predicts the application execution time with the same dataset and conditions as the application execution. In that case, we can validate that the PAS2P methodology generates an application model that represents the application behaviour, in order to prove experimentally our hypothesis: the application has the same structure (same phases) for different datasets and different initial conditions, but with different amounts of repetition (different weights).

We have carried out a set of executions on the K-means application; considering the data and input parameters (K defined as the number of centroids) of the K-means application, we vary the dataset size and the initial number of centroids of each execution. We can control the dataset by changing the random seed to initialise a pseudorandom number generator of both the dataset and the initial centroids. The dimensionality of the data points was set to 16 coordinates, and the algorithm was executed, defining 24 clusters. We conducted our experiments on a 256-process distributed system consisting of 4 compute nodes, each equipped with 64 cores, as shown in Table 1.

The procedure mentioned in [9] is applied using the PAS2P methodology. First, the application is analyzed, and its signature is created for each experiment. Then, we use the signature to predict the execution time for a given machine and a given configuration, according to the values in Table 1. Finally, when we compare the execution time of the application with the time predicted

ICCS Camera Ready Version 2023
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-36021-3_39

Table 2. Weight Variation obtained when K-Means is executed with different datasets.

Exp. ID	Phase	Weight	Number of Instructions	Exp. ID	Phase	Weight	Number of Instructions
1	1	291	5306507150	2	1	225	5306813225
1	2	290	463	2	2	224	463
1	3	292	463	2	3	226	463
1	4	291	469	2	4	225	469
3	1	285	5306511267	4	1	232	5306812383
3	2	284	463	4	2	231	463
3	3	286	463	4	3	233	463
3	4	285	469	4	4	232	469
5	1	306	5306506593	6	1	230	5306812411
5	2	305	463	6	2	229	463
5	3	307	463	6	3	231	463
5	4	306	469	6	4	230	469
7	1	284	4064938012	8	1	229	4065170526
7	2	283	463	8	2	228	463
7	3	285	463	8	3	230	463
7	4	284	469	8	4	229	469
9	1	294	4064940486	10	1	234	4065172879
9	2	293	463	10	2	233	463
9	3	295	463	10	3	235	463
9	4	294	469	10	4	234	469
11	1	321	4064938896	12	1	232	4065170309
11	2	320	463	12	2	231	463
11	3	322	463	12	3	233	463
11	4	321	469	12	4	232	469

by PAS2P for each of the experiments, an average prediction error of 4.3% is obtained.

So, experimentally, we can say that PAS2P performs a correct analysis of the phases in such a way as to perform a reduction of an application to a signature of it, with an average reduction of 8.3% in the execution time of the signature in relation to the total execution time.

For the second part of the experiment, we used the PAS2P tool to obtain traces and extract information about the program's structure for each experiment. The analysis of the data, following our hypothesis, is presented in Table 2. The results indicate that the application's structure remains consistent across all cases, with four phases identified. However, the weights of each phase, and the number of times each phase is repeated, vary among the datasets. This observation can be explained by the fact that the application uses the same steps for the experiment but with varying repetitions for each phase.

One proposed objective is to evaluate an application's efficiency on a specific architecture using the application signature. We define computational time as when a process executes computational instructions—the communication time, such as the transmission or reception time of MPI messages, plus the idle time waiting for communication. Therefore we define Phase Execution Time as the

ICCS Camera Ready Version 2023
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-36021-3_39

sum of computing time plus communication time. Consequently, we define $efficiency_{phase}$ as seen in the following equation.

$$efficiency_{phase} = \frac{computing\_time}{phase\_execution\_time} \quad (2)$$

4 Experimental results

To evaluate efficiency, we have validated by selecting the Experiment 1, as mentioned in the previous section (Table 2) with the K-Means application. This validation is characterised by a phased analysis in order to identify the efficiency of each phase in relation to the global execution time. The experimentation methodology runs the K-Means application, analyses it, and extracts signatures using PAS2P on a system with a specific architecture. The computing system is composed of 7 compute nodes of 64 cores (AMD Opteron6262 HE processor) with an interconnection network of 40 Gb/s Infiniband. As shown in Table 3, we present the efficiency results for each phase based on the results of Experiment 1 according to Equation (2).

After analysing the K-Means application with PAS2P, the application behaviour is represented by 4 phases, as is shown in Table 3. The first phase (the more relevant phase from the computational point of view) had the highest efficiency, with a value of 99.93%. The second phase had an efficiency of only 11.64%, whilst the third phase had an efficiency of 24.58%. The fourth and final phase had the lowest efficiency, at only 7.28%. In addition to the efficiency results, Table 3 includes the computational time related to the number of instructions and total phase execution time (in ns) and the communication volume for each phase (in MB).

Table 3 shows the efficiency of each phase individually. However, this information is still not enough to know the global impact of each of these phases on the total execution time of the application. Therefore, in order to see the global effects of each phase, we proceed to carry out the procedure described below, resulting in Table 3 from Table 4.

To determine the "Global Computing Time" and the "Total Phase Execution" columns, we multiply the weights of each phase by their respective execution times. Next, we sum up the "Total Phase Execution" values for each phase to obtain the overall "Execution Time" (ET) of the application. To calculate

Table 3. Efficiency for each phase (Results of experiment 1)

Phase	Computing Time [ns]	Phase Execution Time [ns]	Efficiency [%]	Number of Instructions	Communication Volume
1	1.203028e+12	1.203866e+12	99.930416	5306514959	1536
2	5.770710e+06	4.956300e+07	11.643181	463	0
3	1.239423e+07	5.042100e+07	24.581488	464	1536
4	1.236372e+07	1.697620e+08	7.282971	469	4

ICCS Camera Ready Version 2023
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-36021-3_39

Table 4. Global efficiency (Results of experiment 1)

Phase	Global Comp. Time [ns]	Total Phase Execution [ns]	Global Comp. Time[%]	Max. Comp. [%]	Room for improvement[%]
1	3.500811e+14	3.503249e+14	99.9080	99.9775	0.0695
2	1.673506e+09	1.437327e+10	0.0004	0.0041	0.0036
3	3.619116e+09	1.472293e+10	0.0010	0.0042	0.0031
4	3.597842e+09	4.940074e+10	0.0010	0.0140	0.0130

both "the Percentage Global Computational" time and the "Percentage Max. Computation" for each phase, we divide the "Global Computational Time" and the "Total Phase Execution" columns by the total application Execution Time (ET). Lastly, we subtract the two last columns to obtain the values in the "Room for improvement" column.

Global Comp. Time denotes the total duration taken to complete a computing phase, while Total Phase Execution refers to the entire time required for the phase to run. The forth column shows the percentage of total computing time utilised by the phase, indicated as "Global Computational Time [%]". The fifth column (Max. Comp. [%]) represents the maximum possible percentage that the phase can achieve if it performs computations for the entire phase duration, and it can be considered the theoretical limit. Finally, the last column (Room for improvement) displays the difference (in percentages) between the actual computational time and the theoretical maximum computational time, reflecting the potential for improvement.

In this scenario, the room for improvement is limited since Phase 1 is already highly efficient and dramatically influences the overall application time. Additionally, it can be noted that the impact of the remaining phases is minimal, as evidenced in the "Max. Computation [%]" column. Therefore, enhancing the efficiency of these less impactful phases may yield few benefits. However, if any of the values in the last column were considerably high, as administrators, we would provide the programmer with a report highlighting the corresponding phase's substantial impact and recommend efforts to improve its efficiency.

5 Conclusions

This work aims to extend the PAS2P methodology's contributions to AI by validating the accuracy of PAS2P in predicting the execution time by taking the K-Means application as a case study. The primary goal is to analyse the K-Means clustering algorithm and demonstrate the hypothesis that, although different datasets or initial conditions may affect the number of repetitions of the same phases, the phases remain the same.

Furthermore, the study aims to calculate the efficiency of each phase, its global impact on performance, as well as the improvement margin. Identifying critical phases and optimising their performance can improve the application's efficiency. This work provides a useful report with the efficiency results of each phase for programmers to take necessary steps toward achieving that goal.

ICCS Camera Ready Version 2023
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-36021-3_39

For future work, we will consider establishing mapping policies to improve performance using the methodology presented in this work. As a first approach, we will select three areas of AI for reviewing current tools and their focus. The initial areas of AI with which we will be working are classification algorithms, heuristics, and genetic algorithms.

6 Acknowledgments

This research has been supported by the Agencia Estatal de Investigacion (AEI), Spain and the Fondo Europeo de Desarrollo Regional (FEDER) UE, under contract PID2020-112496GB-I00 and partially funded by the Fundacion Escuelas Universitarias Gimbernat (EUG).

References

1. G. Verma et al., "HPCFAIR: Enabling FAIR AI for HPC Applications," 2021 IEEE/ACM Workshop on Machine Learning in High Performance Computing Environments (MLHPC), St. Louis, MO, USA, 2021, pp. 58-68, doi: 10.1109/MLHPC54614.2021.00011.
2. A. Anandkumar, "Role of HPC in next-generation AI," 2020 IEEE 27th International Conference on High Performance Computing, Data, and Analytics (HiPC), Pune, India, 2020, pp. xx-xx, doi: 10.1109/HiPC50609.2020.00010.
3. A. Khan et al., "Hvac: Removing I/O Bottleneck for Large-Scale Deep Learning Applications," 2022 IEEE International Conference on Cluster Computing (CLUSTER), Heidelberg, Germany, 2022, pp. 324-335, doi: 10.1109/CLUSTER51413.2022.00044.
4. Fink, Z., Liu, S., Choi, J., Diener, M., Kale, L. V. (2021, November). Performance evaluation of Python parallel programming models: Charm4Py and mpi4py. In 2021 IEEE/ACM 6th International Workshop on Extreme Scale Programming Models and Middleware (ESPM2) (pp. 38-44). IEEE.
5. Charm4py documentation, https://charm4py.readthedocs.io/en/latest/. Last accessed 10 Feb 2023
6. MPI for Python documentation, https://mpi4py.readthedocs.io/en/stable/. Last accessed 10 Feb 2023
7. Alnaasan, N., Jain, A., Shafi, A., Subramoni, H., Panda, D. K. (2022, May). OMB-Py: Python Micro-Benchmarks for Evaluating Performance of MPI Libraries on HPC Systems. In 2022 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW) (pp. 870-879). IEEE.
8. S. Lloyd, "Least squares quantization in PCM," in IEEE Transactions on Information Theory, vol. 28, no. 2, pp. 129-137, March 1982, doi: 10.1109/TIT.1982.1056489.
9. Wong, A., Rexachs, D., Luque, E. (2015). Parallel Application Signature for Performance Analysis and Prediction. IEEE Transactions on Parallel and Distributed Systems, 26(7), 2009-2019.
10. Tirado, F., Wong, A., Rexachs, D., Luque, E. (2021). Improving Analysis in SPMD Applications for Performance Prediction. In Advances in Parallel Distributed Processing, and Applications: Proceedings from PDPTA'20, CSC'20, MSV'20, and GCC'20 (pp. 387-404). Springer International Publishing.

ICCS Camera Ready Version 2023
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-36021-3_39