Performance Characterization of the Alya Fluid Dynamics Simulator

Guilherme Antonio Camelo, Lucas Mello Schnorr
Institute of Informatics – Federal University of Rio Grande do Sul
Caixa Postal 15.064 – CEP 91.501-970 – Porto Alegre – RS – Brazil

***Abstract*—This article presents preliminary results of a performance characterization of the *Alya* fluid dynamic simulator. Experiments are conducted in parallel using the MPI specification implemented by *OpenMPI*, and the application is traced using the *Extrae* tracing tool. By taking a closer look at the traces, it is possible to effectively see different performance patterns such as the communications among ranks, and the effective application load imbalance.**

I. Introduction

Numerical simulation is used to understand and model natural behaviors. Very often these simulations are carried out by techniques based on fluid dynamics, where a model of a real world scenario is implemented and solved using iterative numerical methods using time-steps. High performance computers are employed as execution platforms to run these models, as any sequential approach would make certain simulations executions impractical for the level of details required by the scientific community.

Distributed and parallel programming techniques provide high computational power. Interfaces such as OpenMPI enable portable implementations for the execution of complex programs in less time, and are ideal for dealing with physical problems for real world simulations. Alya[1] is an example of MPI-based fluid dynamic framework simulator. It is an open source project, part of the Prace Benchmark[2], from the Barcelona Supercomputing Center (BSC) to solve numerically physical problems.

Very often numerical simulation applications have irregular loads during execution. Such irregularity appears for many reasons, such as irregular control and data structures, adaptative mesh refinement (AMR)[3], or even irregular iteraction patterns among processes. These reasons ultimately lead to a load imbalance among both resources and time as the simulation advances. Finding out the actual application behavior on a particular platform is key to apply optimization techniques, such as better balancing algorithms or a more appropriate communication patterns. The most efficient way to obtain such information, when you do not know the application code is to apply tracing techniques. They use files to keep track of information regarding the application, which is saved under the form of events, for instance, to track MPI operations.

In this article, the Alya fluid dynamic simulation tool is executed on a 4-node platform with 64 cores, using the OpenMPI library[4]. The goal is to investigate whether Alya has a irregular execution behavior regarding the resources and the time for a representative input. We employ the Extrae tracing tool to record all MPI communication operations, and then present preliminary results of the performance analysis of Alya. A similar study has already been conducted[5] in a larger scale platform. Our secondary goal is to understand the behavior in a smaller scale platform.

The paper is structured as follows. Section II presents the platform used for the experiment. The methodology used in the performance characterization is detailed in section III. Results are presented in section IV. The main contributions and the future work are detailed in section V. A reproducible and open scientific research is essential [6]. This work is therefore the result of a effort to create a reproducible project, publicly available at http://github.com/guiacamelo/alya/.

II. Execution Environment

Experiments are conducted on computers that are part of the Draco cluster at the Institute of Informatics of the Federal University of Rio Grande do Sul (UFRGS). The cluster is formed by eight nodes, each one equipped with Intel (R) Xeon (R) E5-2640 v2 CPU @ 2.00GHz processors with 16 physical cores (32 with hyperthreading), 64 gigabytes of RAM, running Ubuntu 14.04.4 LTS. Only physical cores were used in the executions. In the experimentations different configurations have been tested, with various numbers of cores, number of nodes and of time-steps. Extrae version 3.3.0 is the tracing tool, and for the parallel execution, OpenMPI 1.10.2.

III. Methodology

We have used only four nodes of the Draco cluster for a total of 64 cores. The simulation ran only until the end of the third timestep due to the large size of the trace files. The trace files for this execution have size of 2,7 gigabytes. The tool *Extrae* creates individual traces for each processor, keeping information regarding the communication and execution. Multiple files are created in *.mpits* format containing information about what was executed by a given process. It is then necessary to convert and merge them using the tool *mpi2prv* that outputs the Paraver format (*.prv*). After that, a *perl* script is used to filter the relevant data creating a *Comma-Separated Values* (CSV) file used for the analysis scripts.

The resulting CSV output have four columns of data: the MPI rank (Rank), the time in which the process enter the state (Start), the time in which the process finished the state (End), and the MPI state name (State). With this information

it is possible to calculate the actual workload (running time in seconds) of each process, as well as create space/time graphics that tells us the order and what was the time that each state occurred in each case. All calculations are made from scripts written in the R language, making use of *ggplot2* and *dplyr* libraries. Figure 1 gives a summary of the steps to trace Alya and to conduct the performance analysis of the experiment.

In the schematics on figure 1 the first and second blocks represent the execution with *MPI* and trace with *Extrae* in 4 nodes each using 16 cores. The next two blocks in the flow are the output of the simulation results and the trace files in *.mpits* format. The following two blocks represent the conversion of the trace to *.prv* and then to *.CSV* using *mpi2prv* and a *Perl* script, respectively. Then a block representing the *CSV* format is presented followed by the last block that represents the creation of graphics and analysis of the experiment using language *R*.

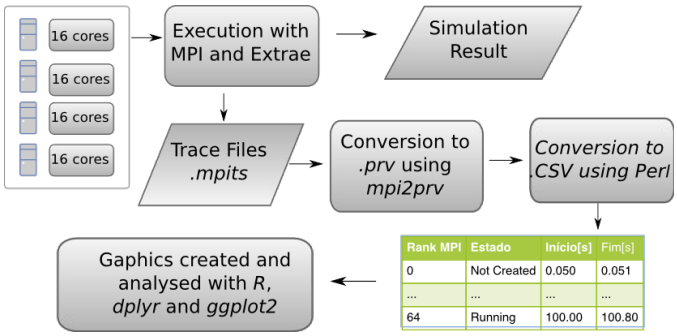

Fig. 1. Methodology used on the execution and analysis of the experiment.

IV. Preliminary Results

The goal of our performance analysis is to identify relevant characteristics of the application and evaluate whether it presents load imbalance among resources and along time. We also intend to refine our methodology to conduct larger scale experiments. We provide as follows an overview about the load imbalance, a detailed analysis using a traditional space/time view, and an attempt to explain the identified behavior using per-rank state statistics. We end the analysis by globally applying the percent imbalance metric to evaluate the load imbalance. All data presented in the graphics are acquired from one execution using 4 node and 64 cores.

A. Overview of Load Imbalance

Figure 2 shows the aggregate time of effective computation for each process. Computation is calculated by summing all time periods in which the process performs data processing. All periods of time on MPI communication, point-to-point synchronization and collective synchronization are not included in these measures. It is observed that most processes have a total computing time in the order of 150 seconds. In some cases, however, the times reaches up to 300 seconds, for instance, the process 60. This is an indicative that a load imbalance occurs considering this specific case study.

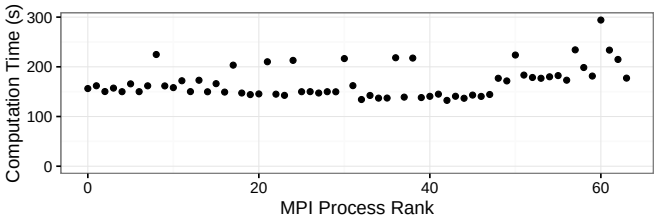

Fig. 2. Aggregated time of effective computation (Y axis) per rank (X).

B. Space/Time Execution

Figure 3 presents detailed information about the load balancing, showing the time that each process was in a state of effective computation. At the beginning of execution, process zero engages in distributing work among processes. This activity continues until the 150 seconds mark. After this time mark, remaining processes begin to work, and, the root process (zero) assumes the organizer role, becoming idle while waiting for the response of other processes.

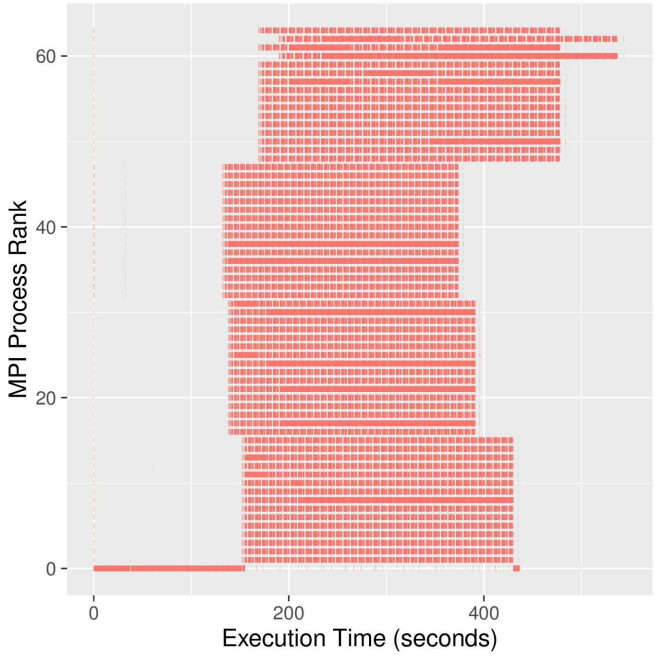

Fig. 3. Timeline (on X axis) showing computation states per process (on Y).

An interesting feature that can also be observed in Figure 3 is the existence of four groups of processes: the 0-15 of 16-31, and so on. This behavior correlates with the number of nodes used in the experiment, indicating that internally in a node all processes begin their computation roughly at the same time. This indicates that the initial load distribution from process zero is not scalable because visually the computations in each one of the four machines start sequentially: first the group between the processes 32-47, After the group of processes 16-31, then the group containing the zero process (0-15) and finally the group of processes 48-63. In the latter group, we

also observe an anomaly in the processes 60 and 62. They start and finish their computation after the other process from their group. Such anomaly is observed only within this group.

The process 60 has a peculiar behavior when compared to the other processes. Figure 2 features the time of effective computation: it runs for roughly 300 seconds, twice as many other processes. Figure 3 shows that process 60 has an anomalous behavior, beginning and ending its execution last. Moreover, the execution state (*Running*), where the computation is actually performed, does not seem to contain spaces as other states. This probably indicates that the time spent on communication functions for this particular case is much lower than in other processes.

C. Process Behavior by State

Figure 4 displays a summary of dedicated time (Y axis) by process (X axis) to each state (different facets). Only the five most important states are presented (*Blocking Send*, *Group Communication*, *Running*, *Send Receive* and *Waiting a message*). The other states present very little or inexistent time. The *Blocking Send* have less influence in load balance than the other states since they are somewhat similar in all processes (except for the 48-63 group, slightly higher). The *Group Communication* have quite different times among processes and a very long time to process zero, as detailed in previous sections. Load imbalance is shown in the facet *Running*, similar to the data presented in Figure 2. The sending and receiving times are relatively homogeneous between processes, except in some cases in which they are smaller. Therefore, it is possible to see in the rightmost facet of the graphic that the possible reason of the anomaly of the processes 60 and 62 is due to additional time spent in the state *Waiting a message*. Such fact contradicts our previous hypothesis drafted in previous section, where the timeline visually indicated a higher less communication time for process 60. This is probably due to drawing much more events than the screen space available [7] to draw them.

D. Percent Imbalance

According to Pearce [8], and then validated by Alles [9], the percent imbalance formula is used to formally calculate the load balance. It is described by the Equation 1 where λ is the load imbalance value to be calculate, L_{max} is the value of the process with the highest load, and $\overline{L}$ is the average computational load among processes. The metric may be calculated either for the entire execution or for different phases e.g. one measure for each time interval or timestep.

$$\lambda = (\frac{L_{max}}{\overline{L}} - 1) * 100 \quad (1)$$

In this work the calculation of the metric is performed considering the entire execution, thus being a global indicator of load imbalance. The metric characterizes the uneven distribution of work, and when applied to our measurements gives the value $\lambda = 74.25161$. That represents a workload imbalance of 75%. This indicates that if the load is more evenly distributed between the computational resources there would be room for a potential performance improvement.

Fig. 4. Time spend on each state by process.

V. Conclusion and Future Work

This paper presents preliminary results of the performance analysis of the Alya fluid dynamic simulation tool. Our goal is to better understand its behavior specially regarding the load balance. Therefore, a simulation is carried out on a platform of four computer nodes totaling 64 cores, leading to a execution of approximately 450 seconds. The execution is traced using the *Extrae*, enabling us to discover relevant information about the core operation of *Alya* in the addressed case. Among the results, we observe that a significant share of the time (about 34% in a run with three timesteps) is somewhat wasted in the beginning of the execution probably to divide the problem, creating considerable overhead since remanining processes are kept idle. Others results include the detection of anomalies in some processes and the perception that the root process (zero) sends the partitioned data sequentially to different nodes, making the start of application considerably slow and not scalable. The calculation of the percent imbalance indicates that there is a potential performance improvement. The reason for such imbalance is still being investigated.

As future work, we will study possible changes in the code that may provide a balanced distribution in a more egalitarian fashion. We intent to execute similar experiments with other configurations, varying the number of nodes, number of cores, and number of timesteps in order to confirm the behavior acquired in this experiment. We also hope to manually mark the beginning of each iteration of the algorithm (changing the application code) so that the balancing metrics may be calculated for each phase of computation/communication. This refinement will allow us to check the load balance along time.

VI. Acknowledgments

The results reported in this study were generated in virtue of the agreement between Hewlett Packard Enterprise (HPE) and the Federal University of Rio Grande do Sul (UFRGS), financed by resources in return for the exemption or reduction of the IPI tax, granted by Brazilian Law nº 8248, 1991, and its subsequent updates.

This investigation also receives funds from the H2020 program EU and MCTI / RNP-Brazil through HPC4E project with code 689772, the FAPERGS / Inria ExaSE design, universal design CNPq 447311 / 2014-0, and international CNRS / LICIA laboratory. We also thank Flavio Alles for the discussions regarding load balancing metrics.

References

[1] M. Vázquez, G. Houzeaux, S. Koric, A. Artigues, J. Aguado-Sierra, R. Aris, D. Mira, H. Calmet, F. Cucchietti, H. Owen, A. Taha, and J. M. Cela, "Alya: Towards exascale for engineering simulation codes," *arXiv.org*, 2014. [Online]. Available: http://arxiv.org/abs/1404.4881

[2] Prace, "Prace Research Infrastructure unified european applications benchmark suite - prace research infrastructure," http://www.prace-ri.eu/ueabs/, 2013, publicado: 2013-10-17, Acessado: 2016-07-15.

[3] M. J. Berger and P. Colella, "Local adaptive mesh refinement for shock hydrodynamics," *Journal of computational Physics*, vol. 82, no. 1, pp. 64–84, 1989.

[4] E. Gabriel, G. E. Fagg, G. Bosilca, T. Angskun, J. J. Dongarra, J. M. Squyres, V. Sahay, P. Kambadur, B. Barrett, A. Lumsdaine *et al.*, "Open mpi: Goals, concept, and design of a next generation mpi implementation," in *European Parallel Virtual Machine/Message Passing Interface Users' Group Meeting*. Springer, 2004, pp. 97–104.

[5] J. Rodríguez, "Performance analysis of alya on a tier-0 machine using extrae," Prace White Papers, Tech. Rep. 151, 2014.

[6] A. Legrand, "Scientific methodology and performance evaluation," https://github.com/alegrand/SMPE, 2015.

[7] L. M. Schnorr and A. Legrand, "Visualizing more performance data than what fits on your screen," in *Tools for High Performance Computing 2012*. Springer, 2013, pp. 149–162.

[8] O. Pearce, T. Gamblin, B. R. De Supinski, M. Schulz, and N. M. Amato, "Quantifying the effectiveness of load balance algorithms," in *Proceedings of the 26th ACM international conference on Supercomputing*. ACM, 2012, pp. 185–194.

[9] F. A. Rodrigues, "Study of Load Distribution Measures for High-performance Applications," Master's thesis, Universidade Federal do Rio Grande do Sul, Porto Alegre, RS, Brasil, 2016.