A new Load Balancer Strategy to reduce the runtime of Iterative Applications

1st Giovane da Rosa Lizot
UNIJUI
Ijuí, RS, Brazil
giovane.lizot@unijui.edu.br

2th Edson L. Padoin
UNIJUI
Ijuí, RS, Brazil
padoin@unijui.edu.br

***Abstract*—In this article is presented the proposal of a new load balancer developed for the CHARM++ programming model. Your project allows you to carry out the processes with a restricted migration number, as well as the total execution time of the applications.. From the initial test results, numbers have been obtained that demonstrate the reduction in overall runtime compared to other load balancers.**

I. Introduction

Computational simulations are increasingly complex, continuously demanding more processing power from High Performance Computing (HPC) systems. However, one of the problems faced is the load unbalance generated by the parallel applicationsWhen parallelized, they present excessive communication between tasks and unbalanced load, preventing the efficient use of its potential. Therefore, some cores can receive tasks with less load or remain idle while others execute the applications, causing, as a consequence, inefficiency in the use of the systems [9].

To solve such problem, load balancers have been proposed, aiming to increase the utilization of the potential of computational systems in processing level and execution time. In this context, many parallel applications that involve simulation with dynamic behaviors or calculations based on complex formulas have used such resources due to the imbalance of load and the amount of communications [10].

Many researches seek to improve the execution time of such computational systems. Thus, different research centers aim at the development of performance-enhancement techniques for scientific applications, improving the processing and execution time, and the second is the challenge, of decreasing the application execution time to be studied and proposed in this work.

Parallel computing has grown from few an processing units to systems with multiple processing units. It can be analyzed how the parallelization of tasks assists in the performance of processors through task migrations, in the same way that the parallelization of the applications exploited the processment competition in order to improve the efficiency in parallel processing.

A divide in groups as loads for control as migrations strategy contributes to more precise load balancing because task migration is a costly process for the system. GroupLB contributes by adapting better with the task of balancing processor loads without having to use more different BCs for this, dividing the loads according to their size. The rest of the paper is organized as follow. Section 2 positions our work against related work. Section 3 presents our proposed load balancer. Section 4 details the methodology used. Section 5 has the evaluation of the proposed load balancer. Conclusion and future work appears in Section 6.

II. Related Work

Parallel programming are developed to platforms with shared or distributed memory. Some scientific applications have regular designs that lead to well balanced load distributions, while others are more imbalanced due to the fact that they have tasks with different processing demands. So, different approaches can be used to deal with imbalanced workloads such as CHARM++, Kaapi and UPC [8]. These runtime systems usually focus on reducing the execution time by applying clever task orchestration strategies and by improving the load distribution among cores.

Different approaches have been employed to make load balancing and reduced runtime of applications. Among them, centralized [3], [7] and distributed [2], [4], [11]. strategies are more employed. However, new hierarchical approaches are being proposed reduce the overhead these approaches [11].

Once that in each one, a different amount of cores are selected to make decision of load balancing where tasks are migrated to different cores of the parallel systems.

Centralized strategies make load balancing decisions on a single processor. To do this, the load and communication data of all the tasks are accumulated in a specific processor, which performs a decision process based on this information. In this type of strategy the GREEDYLB and REFINELB balancers are the most cited and used.

Load balancers like GREEDYLB and REFINELB were implemented and are available in the CHARM++ programming environment.REFINELB is an approach based on load refinement. This load balancer moves tasks of the most overloaded cores to the least loaded ones until reaching an average [1]. The REFINELB moves objects from the most overloaded processors to the least loaded, aiming to reach an average, with the number of objects migrated limited [6].

Balancer AVERAGELB [1] and SMARTLB [5], uses acentralized approach in its strategy, which takes decisions in a single process.

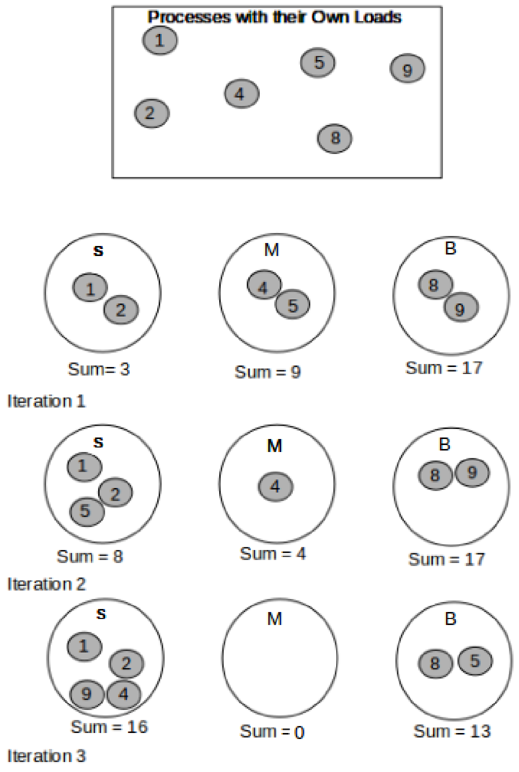
Differently, GREEDYLB uses a greedy approach algorithm that implementing practice of combinatorial optimization. Its algorithm, to migrate heavy objects to the processor with less load. This is repeated until the load of all processors reaches a close proximity to the average load.

III. GROUPLB LOAD BALANCER

Our proposed strategy, called GROUPLB, take into account the AVERAGELB [1] and SMARTLB [5] mechanisms to make decisions. Similar to AVERAGELB, it also uses a centralized approach in its strategy, which takes decisions in a single process. The strategy of the algorithm takes into account the arithmetic mean of each processor, calculating its loads, in order to reduce the number of migrations, seeking a balance between the loads [6].

The algorithms collect system and application information at run-time to dynamically use them in load balancing decision, to reduce execution time. Our strategy divides the processes according to the computational loads into three groups, called *Small* (S), *Medium* (M) and *Bigger* (B), which respectively store the load processes considered relatively small, medium and big.

Fig. 1. Load Balancer Strategy



In this way, when the load balancer is applied, the loads of the cores is verified, and, from this information, the difference of loads between them will be analyzed. Subsequently, the tasks with hight loads are divided into the respective groups. In the algorithm there will be the control variables, where the variable ”less” represents the sum of the loads in the S array, the variable ”bigger” the sum of the loads in the B array, and the variable ”delta” the difference between the variables B and S. The groups S, M, B are created and tasks are placed there based on their initial value and delta receives the difference of the loads of group M.

Finally, it moves processes between groups P and G so that the difference between groups decreases. Thus, the loads tend to be distributed among the colors according to the adopted strategy, reducing the imbalance. Therefore, unnecessary migrations are not performed by reducing the execution time of the application.

Figure 1 shows the run of the algorithm, how is the case of the random as load flow non processor and how divide with agreement with their dimensions, when compared to the group M, are as large will be shifted to the group B, if not to the group S, and the difference of them is displaced to the delta.

Figure 2 shows the adopted iteration strategy. In it, it can be seen the division of the loads in the *Small*, *Medium* and *Bigger* groups, as well as to analyze how the difference of the Small and Large groups, the ”delta” variable, decreases with each iteration. In the end the loads are balanced according to the adopted strategy, reducing the unbalance, the unnecessary migrations and the runtime of the application, the values are compared to the CHARM++ standard without load balancer. The results follow a normal distribution because related and inspired works are developed on the CHARM++ platform.

To implement the proposed load balancer was selected the CHARM++ environment. This environment is supported by several platforms, allowing the programs developed in this model to run in both shared and distributed memory environments.

IV. METHODOLOGY AND EVALUATION

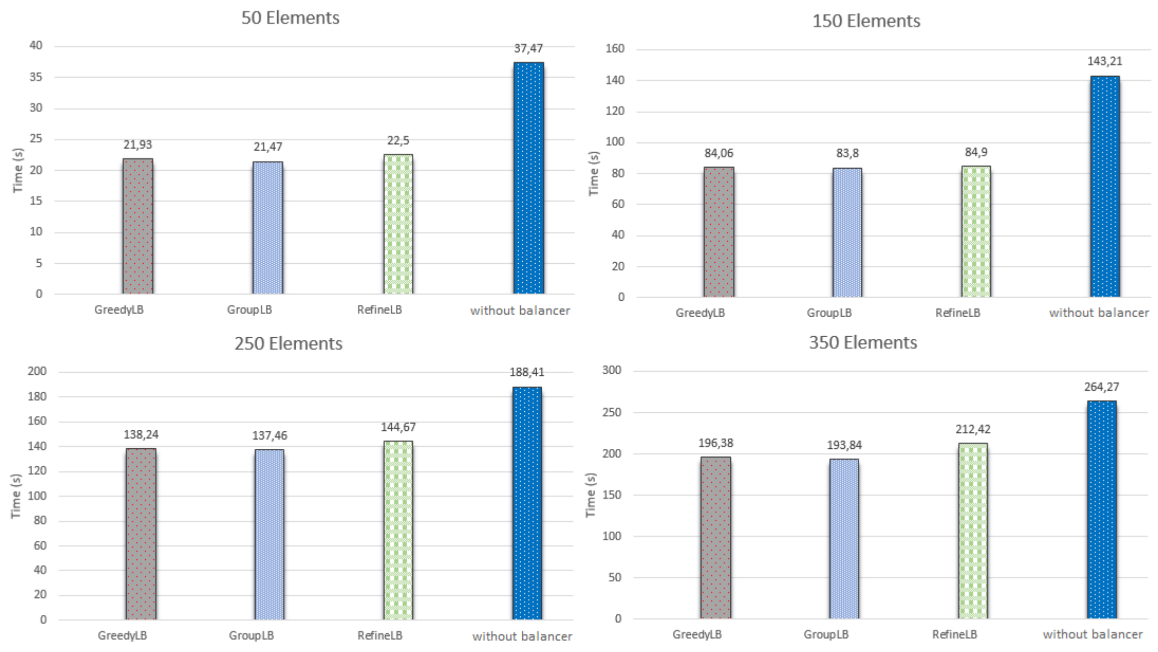
In this section, we present our methodology to validate the load balancer proposal (GROUPLB), and we present ours evaluates about this load balancer comparing it with other state-of-the-art balancers.

To validate our proposal strategy was used an equipment with Intel Core i7M processor wich 8 physical cores. For the tests, the Linux operating system Ubuntu 16.04 with kernel version 4.4.33-1 was used. The version of Charm++ used was 6.5.1 and the g++ compiler in version 6.2.1.

Each of the tests performed in this work was repeated 10 times, the results follow a normal distribution to achieve a relative error less than 5% and 95% statistical confidence for a Student’s t-distribution.

In order to evaluate the performance of proposed load balancer, it was applied in the execution of the lb_test benchmark and compared to the other two balancers, GREEDYLB and REFINELB, which are provided by the CHARM++ programming environment. Tests were performed with 50, 150, 50 and 350 tasks, these being with computational loads varying between

Fig. 2. Results Graph



1500ms and 150000ms. Synchronizations for balance call load was defined every 10 iterations.

In Figure 3 is shown the results of the tests performed with the lb_test benchmark in the platform described.

In the performed tests, the GroupLB load balancer achieved the best performance with the benchmark tested. Using 50 tasks, the time was reduced by 37.47% when compared to the same test without any load balancer. In comparison to the RefineLB, it was lower by 4.58% and 2.09% compared to GreedyLB. With 150 tasks, also presented a reduction compared to the test without any balancer.

With a percentage of 41.50%, though compared to the RefineLB, obtained a percentage of 1.30%, and finally, obtained 0.30% in relation to the GreedyLB. Thus, in the tests with 250 tasks,GroupLB we also obtained gains in the results.

It reduced the time by 27.05% in relation to the execution without any balancer, 4.98% in relation to the RefineLB balancer and 0.56% in relation to the GreedyLB.As well as 350 task, GreedyLB also made gains. It reduced the time by 24.65% in relation to the execution without any balancer, 8.75% in relation to the RefineLB balancer and 1.30% in relation to the GreedyLB balancer.

V. Conclusions

This paper presents a new load balancer proposal, which was developed for the Charm++ programming model. In order to reduce the total execution time of the applications, it adopts a new strategy to allocate the process loads between colors in order to balance the loads and reduce the number of migrations.In the tests performed with the lb_test benchmark, our proposed strategy present results superior to RefineLB and GreedyLB load balancers.

This benchmark was chosen because it is easily configurable to present different levels of unbalance of load, allowing the computational load of each task be configured in different load patterns, both of irregularity, of communication, of being disprovided by the program environment itself. As future works, it is intended to perform tests in parallel systems, using other benchmarks and real problems of scientific computation.

VI. Acknowledgment

This work has been granted by Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES), Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq) and PETROBRAS company. It was also supported by Intel Corporation under the PIBIC UNIJUI Project.

References

[1] G. Arruda, E. L. Padoin, L. L. Pilla, P. O. A. Navaux, and J.-F. Mehaut. Proposta de balanceamento de carga para reduo de migrao de processos em ambientes multiprogramados. In *XVI Simpsio de Sistemas Computacionais (WSCAD-WIC)*, pages 1–8, 2015.

[2] J. M. Bahi, S. Contassot-Vivier, and R. Couturier. Dynamic load balancing and efficient load estimators for asynchronous iterative algorithms. *Parallel and Distributed Systems, IEEE Transactions on*, 16(4):289–299, April 2005.

[3] A. Bhatelé, S. Kumar, C. Mei, J. C. Phillips, G. Zheng, and L. V. Kalé. Overcoming scaling challenges in biomolecular simulations across multiple platforms. In *Proceedings...*, pages 1–12. International Symposium on Parallel and Distributed Processing (IPDPS), IEEE, April 2008.

[4] G. Cybenko. Dynamic load balancing for distributed memory multiprocessors. *Journal of parallel and distributed computing*, 7(2):279–301, 1989.

[5] V. R. S. dos Santos, E. L. Padoin, P. O. A. Navaux, and J.-F. Mhaut. Smartlb: Proposta de um balanceador de carga para reduo de tempo de execuo de aplicaes em ambientes paralelos. In *Congresso da Sociedade Brasileira de Computao (CSBC) - Workshop em Desempenho de Sistemas Computacionais e de Comunicao (WPERFORMANCE)*, jul 2018.

[6] G. Freytag, G. Arruda, R. S. M. Martins, and E. L. Padoin. Anlise de desempenho da paralelizao do problema de caixeiro viajante. In *XV Escola Regional de Alto Desempenho (ERAD)*, pages 1–4, Gramado, RS, 2015. SBC.

[7] S. Ichikawa and S. Yamashita. Static load balancing of parallel pde solver for distributed computing environment. In *Proceedings...*, pages 399–405. International Conf. Parallel and Distributed Computing Systems (PDCS), ISCA Press, 2000.

[8] E. Padoin, M. Castro, L. Pilla, P. Navaux, and J.-F. Mehaut. Saving energy by exploiting residual imbalances on iterative applications. In *High Performance Computing (HiPC), 2014 21st International Conference on*, pages 1–10, Dec 2014.

[9] E. L. Padoin, M. Castro, L. L. Pilla, P. O. Navaux, and J.-F. Méhaut. Saving energy by exploiting residual imbalances on iterative applications. In *2014 21st International Conference on High Performance Computing (HiPC)*, pages 1–10. IEEE, 2014.

[10] L. L. Pilla and E. Meneses. Programao paralela em charm++. pages 1–20, 2015.

[11] G. Zheng, A. Bhatelé, E. Meneses, and L. V. Kalé. Periodic hierarchical load balancing for large supercomputers. *International Journal of High Performance Computing Applications*, 25(4):371–385, 2011.