ISSN 2278-3091
Volume 10, No.4, July - August 2021
International Journal of Advanced Trends in Computer Science and Engineering
Available Online at http://www.warse.org/IJATCSE/static/pdf/file/ijatcse411042021.pdf
https://doi.org/10.30534/ijatcse/2021/411042021

Performance Evaluation of Parallel Data-Intensive Simulations Based on Graph Partitioning Approach

Soumia CHOKRI [a]*, Sohaib BAROUD[b], Safa BELHAOUS[c], Mohamed MESTARI[d], Mohammed YOUSSFI[e]
SSDIA laboratory, ENSET, Hassan II University, Mohammedia, Morocco
[a]chokri.soumaya90@gmail.com, [b]sohaib.baroud@hotmail.fr,[c]safaabelhaous@gmail.com,
[d]mestari@enset-media.ac.ma, [e]med@youssfi.net

ABSTRACT

Scientific simulations require both increasing computing and storage power, parallel and distributed computing are strongly recommended to deal with them. Despite the power of the parallel approach to solving complex simulations, it is more difficult and error-prone, therefore, several factors create significant programming challenges and have the effect of reducing performance, which must be carefully managed to achieve high parallelism.

This paper aims, on the one hand, to evaluate the performance of parallelism based on the graph partitioning approach, and on the other hand, to propose a performance analysis methodology that aims to help scientists to investigate the impact of inter-processor communication on the performance of an application First, a multilevel k way algorithms are used to compute a distribution of the calculations and associated data of simulation. Second, a performance evaluation strategy is used to investigate the efficiency of parallelism. Finally, we examine the results of the experiment and explain the perspective of this study.

Key words: High performance computing, Graph Partitioning; Parallel Computing, Performance evaluation, Resource Allocation; Scientific simulation.

1. INTRODUCTION

To meet the growing needs of computing power in simulations, High-Performance Computing Systems (HPCs), offer potentially very significant computing power and storage capacity, by adopting a parallelization strategy. High performance computing (HPC) is an important and ubiquitous topic in all research fields [1], [2], [16], [17], [18], [25]. That consists of combining the power of several thousand processors to perform complex calculations and massive data processing at high speed. The goal of parallelization is to cut a problem into several sub-problems, in order to concurrently solve all the sub-problems [3].

According to Kennedy's methodology [10], the process of parallelization of a scientific simulation contains 4 stages illustrated by figure 1: first, the problem is partitioned into several parts or tasks. This partitioning is obtained using a decomposition of space or functional decomposition. Then the communications necessary to obtain the data used during the execution of the tasks are established. These communications occur when there are dependencies between tasks. Then some tasks are grouped together to reduce communication costs. Finally, these tasks are then placed on the processors with the objective of reducing the execution time.

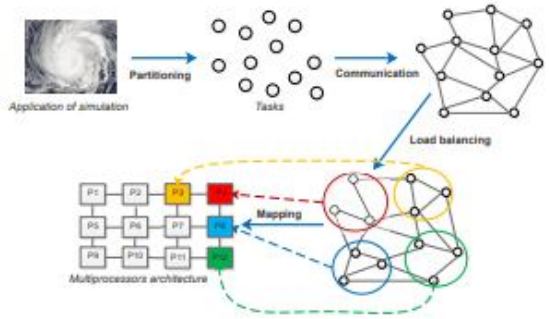

Figure 1: Parallelization process

Despite the ability of the parallel approach to solving complex simulations, it has been documented as error-prone and difficult [11]. Thus several factors create significant programming challenges and have the effect of reducing performance, that must be managed carefully if we are to achieve high parallelism [12]: such as data decomposition, the number of processors, inter-processor communication [15], bandwidth, load balancing...etc.

In order to enhance the performance measures such as efficiency, acceleration, execution time and scalability [13], [14], several performance-based metrics have been proposed in the literature [16], [4]: The parallel execution time is the

sum of the computation time and communication time. It depends not only on the problem size but also on the number of processors used and communication cost between the processors, the speedup is defined as the ratio between the time taken to solve the problem on a single processor and the time taken to solve this same problem on p processors, the efficiency measures the time actually spent by processors in the algorithm. It is defined as the ratio between Speedup and the number of processors.

Parallelizing a scientific simulation application to run in HPC systems presents many problems. A key problem is the decomposition and communication task. Several researchers [18], [19], [20] have studied the impact of decomposition and communication on parallelism, and found that decomposition is one of the most difficult phases. Decomposition of an application is a distribution into tasks and data to be executed on an architecture composed of a number of processors, where each processor executes one or more tasks, and the tasks strictly cooperate by exchanging messages. In such a numerical simulation, it often happens that the execution time is largely dominated by the time needed to perform the communication between processors, as the communication cost can be higher than the execution cost of a normal instruction.

The authors [19], [20] asserts that the process of seeking an appropriate decomposition is a balance of competing forces, during the decomposition of a problem, we must take into account the size of the tasks as granularity. A fine granularity leads to poor performance because of the high additional costs: the creation of threads, synchronization, etc. Too coarse granularity may generate not enough parallelism and an unbalanced load.

In [21] break, the data distribution into two categories: static, the number of tasks is fixed at the start of the simulation, dynamic that evolves during the simulation. The load balancing is a crucial issue influencing the performance of parallel simulations. The aim is to distribute roughly the computational load across multiple processors in order to minimize the execution time. If the tasks are not evenly distributed, then it is possible that one processor is overloaded with work, while another has little work to do, and therefore is misused. The over-all effect of a load imbalance is, therefore, to increase the execution time and reduce the overall efficiency. However, ensuring a truly balanced distribution of the workload, particularly on a multi-processor architecture, can lead to significant additional costs mainly in terms of the inter processor communications.

The graph model is most common approach to overcome these challenges of parallelism by performing a graph partitioning [2], [3] in k-parts. The vertices of the graph represent tasks and edges represent the dependencies between these tasks. Each part represents all the tasks assigned to a processor. Two connected vertices that are in two different parts (one will say that the edge is cut) induce a need for communication between these two processors. The objective of the distribution then becomes to build a partition whose parts are of the same size and cutting a minimum of edges [7]. Generally, this problem is known as an NP-complete [4] problem. However, several strategies have been proposed [9] [22], [23], [24], [25]. There are a wide variety of methods for partitioning meshes. The most common are geometric methods [25], based only on the coordinates of elements in space, and graph-based methods [5], [6], [25] using computational dependencies between elements and focusing only on topology and not to the geometry of the problem.

Graph partitioning(GP) is a popular approach [3], [26] to solve the load balancing challenge in HPC. Consequently, to balance the load of a parallel simulation between processors, it is possible to carry out a GP process into K parts and to assign each part to a processor. GP aims at modelling the support of computation of the simulation, in general, a mesh with a graph and to divide this graph in K part, each part being associate with a processor. the GP problem has the following objectives:

1. on the one hand, to minimize the calculation time by balancing the weight of the parts;
2. and on the other hand, to minimize the communication time by minimizing the weight of the edges cut between the parts, as shown in Figure 2. When GP is used for load balancing in a parallel architecture, a balanced distribution of the workload among the available processors is achieved, while the communication costs of the simulation are reduced. Subsequently, GP emerges as an important parallelization method that ensures high performance by significantly minimizing total execution time.

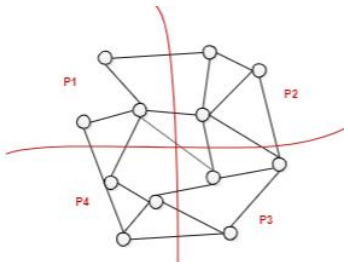

Figure 2: Graph partitioned in k=4

GP is an NP-complete problem [4], [32]. We, therefore, use different heuristics to be able to calculate a partition within a

reasonable time. A plethora of GP-based algorithms for data distribution and load balancing have been developed. The multilevel algorithm appeared as a very efficient method for calculating a k-way balanced partition of a graph [31], [33], [9]. Multi-level Framework uses a divide & conquer approach. In this approach, we build a family of smaller and smaller graphs (by contraction), then we calculate a valid partition on the smallest graph and we project the result from a graph to graph until the initial graph Figure 3. Currently, the well-known partitioning and load balancing algorithms are multi-level algorithms, which use a decomposition algorithm to calculate the partitioning on the smallest contracted graph. The decomposition methods consist of directly cutting the graph into K parts. Obtaining good distributions requires that the K partitions contribute to the minimization of the chosen cost function, generally by minimizing the cut between the parts of the graph, which has the effect of favoring the locality of the communications. Among the GP libraries implementing these techniques, one can cite METIS [34] or SCOTCH [35].

This paper aims, on the one hand, to evaluate the performance of parallel simulation based on the graph partitioning approach, and on the other hand, to propose a performance analysis methodology that aims to help scientists to investigate the impact of inter-processor communication on the performance of an application in order to improve it.
By adopting this analysis method, developers will be able to identify performance problems by investigating the impact of communication on the speedup of applications. In this work a set of scientific simulation models will be used as practical cases to validate our methodology.

The rest of this paper has been organized as follows: Section 2 presents the problem formulation. The detailed methodology is described in Section 3. The experimental results and discussions are presented in Section 4. Ultimately, Section 5 concludes the article.

2. PROBLEM FORMULATION

Given a non-oriented graph G = (V, E), where V is the set of vertices and E is the set of edges that connect pairs of vertices. Vertices and edges can be weighted, where |V| is the weight of the vertex V and where |E| is the weight of the edge E. The problem of partitioning a graph is to divide G into disjoint K partition of same size and minimum dependencies Fig 3. From a mathematical point of view, we can partition the vertices or the edges. On the other hand, in most applications, we are only interested in partitioning graph vertices.

Let $G = (V, E)$ and a set of K subsets of V, denoted $P_k = V_1, V_2, \dots, V_k$. We say that P_k is a partition of G if :

The union of all the elements of PK is V, and No subset of V that is an element of PK is empty.

The elements Vk of Pk are called the parts of the partition:

$$\bigcup_{i=0}^{k} V_i = V \quad V_i \cap V_j = \emptyset \ \forall i \neq j \qquad (1)$$

The parts must be balanced, that is, of the same size:

$$|V_1| \approx |V_2| \approx \dots .. |V_k| \qquad (2)$$

With a minimized cost (cut) function: represents the communication time between the processors:

$$min(\sum |e_{i,j}|), v_i \in V_k, v_j \in V_p \quad \forall \ k \neq p \quad (3)$$

Where: ei,j weight of an edge $ei,j = (vi, vj)$

Partitioning serves, among other things, to solve problems of engineering, high-performance computing, resolution of linear systems, mesh, and in some cases, design of integrated circuits. However, some criticisms have been made about the use of this approach to model these problems, this is particularly the case in [28]. In the literature, graph partitioning is called multi-way graph partitioning [29] or k-way graph partitioning [30]. The problem of k-way graph partitioning is to find a partition in k parts that minimizes an objective function f and whose partitioning scale is unitary, i.e. the parts must have the same weight, to one unit.

Despite the success of existing partitioning algorithms, research challenges remain and a new distribution algorithm should be proposed for efficient execution of simulations. In this context, in the rest of this article, we will study the distribution and load balancing strategies based on the optimal choice of the number of partitions, i.e. the optimal number of processors to execute a simulation efficiently and effectively. Finding the optimal number of partitions (processors) is necessary to optimize the execution time. We express the total simulation time T_{exe} as:

$$T_{exe} = T_{comp} + T_{com} \qquad (4)$$

Where: $Tcomp$: the time of a calculation iteration, taken on the slowest processor and thus minimized by the balancing.

$Tcom$: the time of the communications between processors, corresponding to the edge cut of the graph. The computation time is arithmetically measurable if we combine the

parameters of total load, number of partitions and the balancing constraint. The communication time evolves logically, proportionally to the number of computing nodes (processors). With a large number of nodes, the communication time can exceed the execution time of the program on a single compute node, Figure 3.

In order to find the optimal number of partitions to run a parallel application, we should characterize the evolution of the communication volume.

In this paper, we measure the evolution of the communication volume in function of the number of partitions. In order to show how the structure of the graph influence this evolution by relying on a series of experimentation using METIS Libraries which implement the multilevel k-way Framework. We present experimental results on a large number of graphs arising in various domains including finite element methods, linear etc...

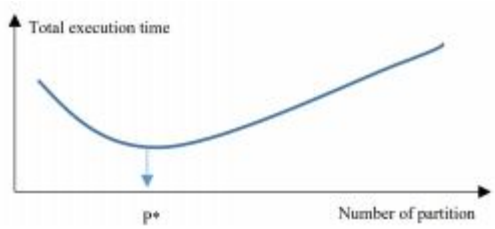

Figure 3: Execution time evolution

3. METHODOLOGY

The goal of our methodology proposed in this section is to be useful to researchers who run simulation applications on HPC environments and are ready to understand its performance, discovering problems related to communication that can potentially affect its efficiency. our methodology consists of the following steps:

3.1. Analysis Steps: The analysis step contains 4:

A. Partitioning step

Firstly, we perform partitioning of each graph, which represents a simulation, several times by varying the number of partitions from 2 to 40, in order to learn more about how fast, it can run when using multiple computing nodes, when it reaches its maximum speed up, and also to investigate more about the problems that limit the performance of the parallelization process.

B. Investigating communication behavior

The purpose of the second step is to focus on the analysis of the behavior of communication volume when increasing the number of partitions. This step, allowed us to understand how the communication volume impacts the performance of parallelization.

During this step, we also compared the evolution of communication volume for each graph and tried to understand what parameter engenders difference between the graphs.

C. Determining optimal number of partitions

Once we understood how the communication volume evolves, and determined the relationship between computational and communication time for each graph, the next step is to determine how many partitions must be invested to reach the peak execution time value. to have high performance and efficiency, the key solution is to choose the right number of processors for a simulation.

Using processors as much as possible is not always the right choice, in addition, if a simulation is run in parallel on too many processors, the communication time between nodes may become too long compared to the computation time. The cost of communication depends not only on the number of messages exchanged between the parties but also on the structure of the graph. This is why the cost of communication cannot be evaluated without taking into account the topology of the graph. We Named P* the number of partitions for which the execution time of a simulation is minimum.

D. Finding the graph characteristics that influence the increase in communication time:

this step aims to find the intrinsic characteristics of the graph, which can impact the evolution of communication, more precisely, finding the issues that are constraining the effectiveness of paralleling simulation.

3.2. Metrics

For the evaluation of performance, we used the following metrics:

A. Communication volume:

Let a graph $G(V, E)$ and consider P be a vector of size V such that $P[i]$ stores the number of the partition that vertex i belongs to.

Let $Vb \subset V$ be the subset of interface (or boarder) vertices. That is, each vertex $v \in Vb$ is connected to at least one vertex that belongs to a different partition. For each vertex $v \in Vb$ *let* $N\ v\ adj[v]$ be the number of domains other than $P[v]$ be the vertices adjacent to v belong to. the equation (5) corresponds to the total communication volume incurred by the partitioning because each interface vertex v needs to be sent to all of its $adj[v]$ partitions [36] , [37] , [38] .

$$TotalV = \sum_{v \in V_b} Nadj[v] \qquad (5)$$

B. Execution time:

Execution time is defined as follows in equation 6:

$$T_{exe} = TotalV + \omega(P_i) \quad (6)$$

w (Pi) is the weight of the partition.

C. Speedup

Speedup or acceleration is the gain in partitioning speed over k partitions formulated in the equation 7:

$$Speedup = |V|/T_{exe}(K_{part}) \quad (7)$$

D. Efficiency

Efficiency is the ratio of speedup to the number of processors, it can be calculated using equation 8:

$$Efficiency = Speedup/K \quad (8)$$

K is number of partitions(processo...

3.3. Data

The test graphs used for the experimental evaluation have been well chosen to be a representative sample of real medium and small scale problems mostly based on a mesh and comprise 2D and 3D examples of dual graphics and nodal graphs.

These graphs are archived in DIMACS'10 collections [39] which contains34 real world graphics from various scientific computing applications, such as finite element computations, matrix computations, and VLSI design. The weights of their vertices and their edges are equal to one; these graphs are not amply available because most applications don't accurately estimate costs and it is challenging to extract significant conclusions from the few examples which we have access.

The list of used graphs, their characteristics; notably the number of vertices and the number of edges are detailed in the table 1. The choice of graphs was carried out in a completely random way and didn't depend on any specific characteristic except that it represents a scientific simulation.

3.4. Tools

To performs the graph partitioning; we choose METIS [38] the most used graph partitioning framework. METIS is a Framework for partitioning large irregular graphs, written at the University of Minnesota, and is freely available, it implements several algorithms based on the multilevel graph partitioning paradigm.

Table 1: Graph used in the experiment

Nom graphe	Nombre de nœuds	Nombre d'arrêts
3elt	4270	13722
4elt	15606	45878
144 graphe	144649	1074393
598a	110971	741934
add20	2395	7462
add32	4960	9462
auto	448695	3314611
bcsstk29	13992	302748
bcsstk30	28924	1007824
bcsstk31	35588	572914
bcsstk32	44609	985064
bcsstk33	8738	291583
Brack	62631	366559
Crack	10240	30380
Cs4	22499	43858
Cti	16840	48232
data	2851	15093
fe4elt2	11143	32818
fe_body	45087	163734
fe_ocean	143437	409593
fe_pwt	36519	144794
fe_rotor	99617	662431
fe_sphere	16386	49152
fe_tooth	78136	452591
finan512	74752	261120
M14b	214765	1679018
memplus	17758	54196
t60k	60005	89440
uk	4824	6837
vibrobox	12328	165250
wave	156317	1059331
whitaker3	9800	28989
wing	62032	121544
wing nodal	10937	75488

4.RESULTS AND DISCUSSION

In our first set of experiments, we observe that the communication volume evolves as the number of partitions does. To make the graph comparable we normalized the communication volume and expressed it as a percentage of total vertex named in this paper "CVa". We judged that it is not useful to represent the detailed results of each graph but we preferred to classify the graphs in four classes according to the connectivity of the graph and to present the results of only one graph per class. The graph we choose are described in the table 2:

Table 2: Results for each graph of our benchmark

Connectivity Range	Chosen Graph	Connectivity
1-5	T60k	3
6-10	3ELT	6
11-15	Data	11
Upper than 15	Vibrobox	27

Graph Name	Connectivity	P*	MaxSpeedup	Efficiency at P*	Range Connectivity	Average P*	Average MaxSpeedup
uk	3	16	7,47	46,69 %	0-5	18,8	8,08
t60k	3	34	11,91	35,03 %			
add32	4	16	12,22	76,38 %			
Cs4	4	12	3,83	31,92 %			
wing	4	16	4,97	31,06 %			
fe_ocean	6	14	6,09	43,50 %	6-10	14	6,086923077
Cti	6	6	2,92	48,67 %			
4elt	6	18	7,65	42,50 %			
fe4elt2	6	12	6,09	50,75 %			
whitaker3	6	14	5,42	38,71 %			
Crack	6	16	5,49	34,31 %			
fe_sphere	6	18	5,37	29,83 %			
memplus	6	8	1,96	24,50 %			
add20	6	4	2,24	56,00 %			
3elt	6	12	5,02	41,83 %			
finan512	7	16	12,1	75,63 %			
fe_body	7	28	10,65	38,04 %			
fe_pwt	8	16	8,13	50,81 %			
data	11	8	3,41	42,63 %	10-15	10,8571429	4,227142857
fe_tooth	12	12	4,52	37,67 %			
Brack	12	14	5,17	36,93 %			
fe_rotor	13	14	5,09	36,36 %			
598a	13	10	4,87	48,70 %			
wave	14	12	4,26	35,50 %			
wing nodal	14	6	2,27	37,83 %			
auto	15	14	6,23	44,50 %	Sup 15	9,11111111	3,91
144 graphe	15	14	4,73	33,79 %			
M14b	16	16	5,72	35,75 %			
vibrobox	27	2	1,34	67,00 %			
bcsstk31	32	10	4,21	42,10 %			
bcsstk29	43	6	3,02	50,33 %			
bcsstk32	44	8	4,48	56,00 %			
bcsstk33	67	4	1,67	41,75 %			
bcsstk30	70	8	3,79	47,38 %			

Figure 6 shows a clear difference in the evolution of the communication volume from one graph to another. It increases slowly for T60K, 3ELT, Data and increases rapidly for the Vibrobox graph.

This evolution (see figure 4) has a direct impact on the evolution of the total execution time, the existing difference between the graphs can be explained by their connectivity. The greater the connectivity, the faster the communication volume evolves.

Figure 5 represents the evolution of speedup when we increase the number of partitions, and demonstrates that when the communication evolves quickly, the acceleration is strongly limited.

Figure 5, 6 helped us to draw another conclusion which is once we reach a number of partitions the gain in speed reaches a maximum value. Once this value is reached the addition of a partition is no longer justified. We call this Number P* and it is the optimal number of partition.

The efficiency of partitioning decreases as we grow the number of partitions.

The table 3 shows for each graph of our Benchmark the connectivity, the optimal number of partitions P* and the

maximum acceleration we can achieve using multilevel K-way partitioning.

In order to facilitate the reading of the results, we have classified the graphs in four classes according to the value of their connectivity. We then calculate the average P* and the average of Maximum Speedup for each class.

Comparing the average of maximum speedup and the average of P* with the average connectivity of each class, we observed a significant correlation between these two parameters. The coefficient of correlation between average P*and average connectivity is estimated at r=-0,8. (see fig 8)

The coefficient of correlation between average maximum speedup and average connectivity is estimated at r=-0,75. (see fig 7)

Figures 7, 8 show the distribution of respectively maximum speedup and optimal number of partitions.

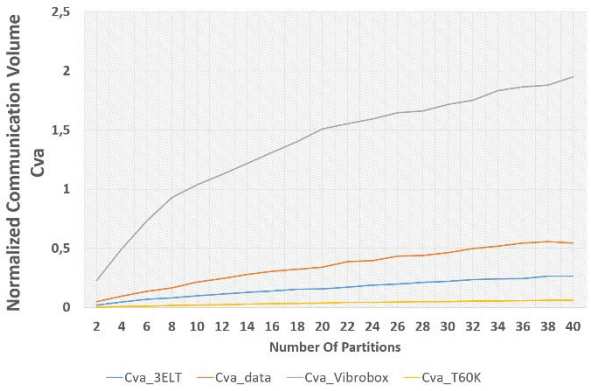

Figure 4: Cva Evolution by increasing number of partitions

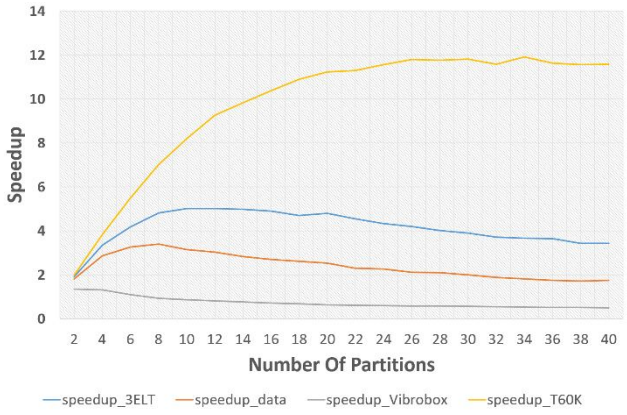

Figure 5: Evolution of Speedup By increasing number of partitions

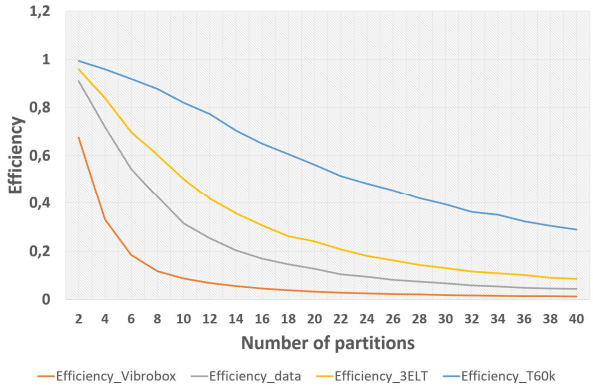

Figure 6: Evolution of Efficiency

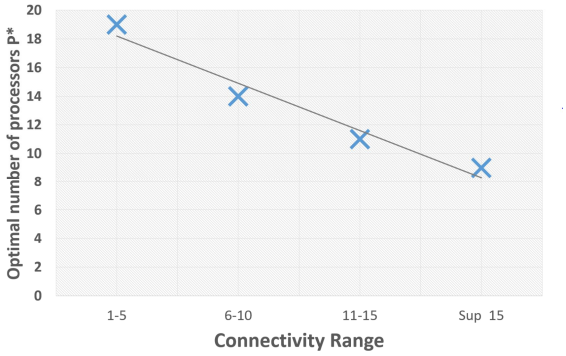

Figure 7: P* by connectivity range

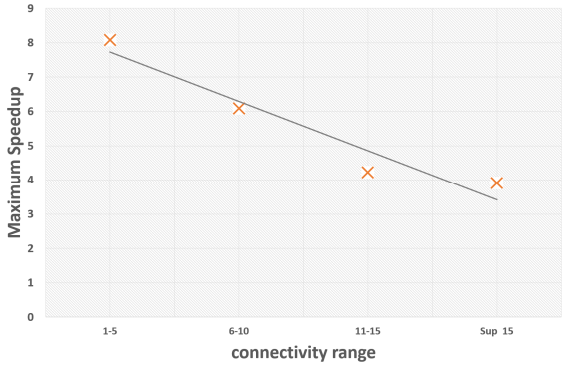

Figure 8: Maximum Speedup by connectivity range

It should be noted that the correlation between connectivity, P * and Maximum Speedup remains very weak. This could be explained by the presence of some aberrant values as it is the case for example for the graphs Memplus and Cti.

The fact that these two graphs represent an exception to the general rule and record low accelerations despite their weak connectivity is probably due to their topology. In the case of the Memplus, on the observation of the graphic representation, we notice that a small number of nodes group

together a large number of edges. This graph records a maximum degree =1100 and an average degree of just 8.

Concerning the graph Cti it shows a balanced distribution of the edges by node however we notice that it is divided into several sub graphs which are disconnected from one another.

Thus we conclude that the performance of parallelism as a technique allowing to reduce the execution time of a scientific simulation is limited by a set of constraints relating to the simulation itself.

The average degree of a graph or otherwise its connectivity is not the only parameter that limits this performance despite the correlation existing between these two parameters.

During the phase of partitioning the graph, it frequently happens that the execution time is largely dominated by the time necessary to carry out the communications between processors, the cost of the communication being able to be higher than the cost of executing the normal instruction. In this case, it may then be sufficient to estimate the complexity of the number of communications required by the simulation.

We have concluded that the most critical point of such execution is the cost of communication. Performance can therefore be significantly increased if the time spent communicating is reduced or covered. This improvement can therefore be obtained either by reducing the number of messages sent or by the optimal choice of the number of processors.

5. CONCLUSION

In this work we have presented the analysis of the behavior of scientific simulations when they are executed on several computation nodes using graph partitioning. This descriptive study made it possible to identify important parameters influencing the performance of parallelism but does not in any case claim to draw up a model making it possible to predict the optimal conditions for the execution of a scientific simulation. Certainly there is a strong correlation between the connectivity of the graph and these parameters but we should investigate more graph's characteristics besides the connectivity that may impact the communication volume that later influence the maximum speedup and the optimal number of partitions. Once these features are defined it would be appropriate to adopt a learning system such as artificial neural network to predict in a more or less precise way the maximum speedup and the P*.

REFERENCES

1. Jannesari A, Wolf F, Tichy WF. SEPS 2014: first international workshop on software engineering for parallel systems. In: SPLASH '14. **Association for Computing Machinery**; 2014; New York, NY, USA: 85–86.
2. Amaral V, Norberto B, Goulão M, et al. **Programming languages for data-Intensive HPC applications: A systematic mapping study**. *Parallel Computing* 2020; 91: 102584. doi: 10.1016/j.parco.2019.102584.
3. Asanovic K, Bodik R, Demmel J, et al. **A view of the parallel computing landscape**. Communications of the ACM 2009; 52(10): 56–67. doi: 10.1145/1562764.1562783.
4. Kwiatkowski J. **Evaluation of Parallel Programs by Measurement of Its Granularity**. In: Wyrzykowski R, Dongarra J, Paprzycki M, Waśniewski J., eds. *Parallel Processing and Applied MathematicsLecture Notes in Computer Science. Springer*; 2002; Berlin, Heidelberg: 145–153.
5. Teresco JD, Devine KD, Flaherty JE. **Partitioning and Dynamic Load Balancing for the Numerical Solution of Partial Differential Equations**. In: Bruaset AM, Tveito A., eds. *Numerical Solution of Partial Differential Equations on Parallel Computers*. 51. Berlin/Heidelberg: Springer-Verlag. 2006 (pp. 55–88)
6. Rus P, Štok B, Mole N. **Parallel computing with load balancing on heterogeneous distributed systems**. *Advances in Engineering Software* 2003; 34(4): 185–201. doi: 10.1016/S0965-9978(02)00141-2
7. Hendrickson B, Leland R. **An Improved Spectral Graph Partitioning Algorithm for Mapping Parallel Computations**. SIAM *Journal on Scientific Computing* 1995; 16(2): 452–469. Publisher: Society for Industrial and Applied Mathematicsdoi:10.1137/0916028
8. Hendrickson B, Kolda TG. **Graph partitioning models for parallel computing** q. *Parallel Computing Elsevier* 2000: 16.
9. Chokri S, Baroud S, Belhaous S, Bentaleb M, Mestari M, Youssfi ME. **Heuristics for dynamic load balancing in parallel computing**. In: 2018 *4th International Conference on Optimization and Applications (ICOA). IEEE Xplore*; 2018: 1–5
10. Pankratius V, Schaefer C, Jannesari A, Tichy WF. **Software engineering for multicore systems: an experience report.** In: IWMSE '08. Association for

Computing Machinery; 2008; New York, NY, USA: 53–60

11. Dongarra J, Foster I, Fox G, et al., **eds.Source book of parallel computing. San Francisco,** CA, USA: Morgan Kaufmann Publishers Inc. . 2003.
12. Meade A, Buckley J, Collins JJ. **Challenges of evolving sequential to parallel code: an exploratory review**. Proceedings of the 12th International Workshop on Principles of Software Evolutionand the 7th annual ERCIM Workshop on SoftwareEvolution. ACM. 2011: 5.
13. Nielsen IM, Janssen CL. **Multicore Challenges and Benefits for High Performance Scientific Computing.** Scientific Programming 2008; 16(4): 277–285. doi: 10.1155/2008/450818
14. McCool MD. **Scalable Programming Models for Massively Multicore Processors**. Proceedings of the IEEE 2008; 96(5): 816–831. Conference Name: Proceedings of the IEEEdoi: 10.1109/JPROC.2008.917731
15. Muresano R, Meyer H, Rexachs D, Luque E. **An approach for an efficient execution of SPMD applications on Multi-core environments**. Future Generation Computer Systems 2017; 66: 11–26. doi: 10.1016/j.future.2016.06.016
16. Yan B, Regueiro R. **Superlinear Speedup Phenomenon in Parallel 3D Discrete Element Method (DEM) Simulations of Complex-shaped Particles**. Parallel Computing 2018; 75. doi: 10.1016/j.parco.2018.03.007
17. Schryen G. **Parallel computational optimization in operations research: A new integrative framework, literature review and research directions**. European Journal of Operational Research 2020; 287(1): 1–18. Publisher: Elsevier.
18. Meade A, Deeptimahanti DK, Buckley J, Collins JJ. **An empirical study of data decomposition for software parallelization**. Journal of Systems and Software 2017; 125. doi: 10.1016/j.jss.2016.02.002
19. Dovolnov E, Kalinov A, Klimov S**. Natural Block Data Decomposition for Heterogeneous Clusters**. In: Proceedings of the 17th International Symposium on Parallel and Distributed Processing IPDPS ’03. IEEE Computer Society; 2003; USA: 102.1.
20. Massingill BL, Mattson TG, Sanders BA. **Reengineering for Parallelism: an entry point into PLPP for legacy applications**. Concurrency and Computation: Practice and Experience 2007; 19(4): 503–529. _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/cpe.1147doi: https://doi.org/10.1002/cpe.1147
21. Pautasso C, Alonso G. **Parallel computing patterns for Grid workflows.** In: 2006 Workshop on Workflows in Support of Large-Scale Science. IEEE Computer Society; 2006: 1–10. ISSN: 2151-1381
22. Campbell PM, Devine KD, Flaherty JE, Gervasio LG, Teresco JD. **Dynamic Octree Load Balancing Using Space-Filling Curves**. Tech. Rep. CS-03-01, Williams College Department of Computer Science; 2003.
23. Meyerhenke H, Monien B, Schamberger S. **Accelerating shape optimizing load balancing for parallel FEM simulations by algebraic multigrid**. In: Proceedings 20th IEEE International Parallel & Distributed Processing Symposium. IEEE; 2006; Rhodes Island, Greece: 10 pp.
24. Sanders P, Schulz C. Think Locally, **Act Globally: Highly Balanced Graph Partitioning**. In: Hutchison D, Kanade T, Kittler J, et al., eds. Experimental Algorithms. 7933. Berlin, Heidelberg: Springer Berlin Heidelberg. 2013 (pp. 164–175). Series Title: Lecture Notes in Computer Science
25. Hendrickson B, Kolda TG. **Graph partitioning for parallel computing.** Parallel Computing 2000; 26(12): 1519–1534.
26. Pothen A, Simon HD, Liou KP. **Partitioning Sparse Matrices with Eigenvectors of Graphs**. SIAM Journal on Matrix Analysis and Applications 1990; 11(3): 430–452. Publisher: Society for Industrial and Applied Mathematicsdoi: 10.1137/0611030
27. Garey MR, Johnson DS. Computers and Intractability; **A Guide to the Theory of NP-Completeness**. New York, NY, USA:W. H. Freeman & Co. . 1990.
28. Andreev K, Räcke H. **Balanced graph partitioning**. In: SPAA ’04. Association for Computing Machinery; 2004; New York, NY, USA: 120–124
29. [29] Pellegrini F. Graph partitioning based methods and tools for scientific computing. ParallelComputing 1997; 23(1): 153–164. doi: 10.1016/S0167-8191(96)00102-0
30. Wang N, Wang Z, Gu Y, Bao Y, Yu G. **TSH: Easy-to-be distributed partitioning for large-scale graphs**. Future Generation Computer Systems 2019; 101: 804–818. doi: 10.1016/j.future.2019.06.033

31. Karypis G, Kumar V. **Multilevel k-way Partitioning Scheme for Irregular Graphs**. Journal of Parallel and Distributed Computing 1998; 48(1): 96–129. doi: 10.1006/jpdc.1997.1404
32. Yarack E, Carletta J. **An evaluation of move-based multi-way partitioning algorithms**. In: Proceedings 2000 International Conference on Computer Design. ; 2000: 363–369. ISSN: 1063-6404
33. Karypis G. **METIS - Serial Graph Partitioning and Fill-reducing Matrix Ordering** | Karypis Lab. 2013. Version 5.1.0, http://glaros.dtc.umn.edu/gkhome/metis/metis/overview.
34. Bader DA, Kappes A, Meyerhenke H, Sanders P, Schulz C, Wagner D. **Benchmarking for Graph Clustering and Partitioning**. In: Alhajj R, Rokne J. , eds. Encyclopedia of Social Network Analysis and MiningNew York, NY: Springer. 2018 (pp. 161–171)
35. Soper A, Walshaw C, Cross M. **A Combined Evolutionary Search and Multilevel Optimisation Approach to GraphPartitioning**. Journal of Global Optimization 2004; 29: 225–241.doi:10.1023/B:JOGO.0000042115.44455.f3
36. Rossi RA, Ahmed NK**. The Network Data Repository with Interactive Graph Analytics and Visualization.** 2015: 2. http//http://networkrepository.com.
37. Schloegel K, Karypis G, Kumar V. **Graph partitioning for high-performance scientific simulations.** In: San Francisco, CA, USA: Morgan Kaufmann Publishers Inc. 2003 (pp. 491–541).
38. Kernighan BW, Lin S. **An efficient heuristic procedure for partitioning graphs**. The Bell System Technical Journal 1970; 49(2): 291–307. Conference Name: The Bell System Technical Journaldoi: 10.1002/j.1538-7305.1970.tb01770.x
39. Karypis G, Kumar V. **Parallel multilevel k-way partitioning scheme for irregular graphs**. In: ACM Press; 1996; Pittsburgh, Pennsylvania, United States: 35–es