Parallel Computing for Multi-core Systems: Current Issues, Challenges and Perspectives

Soumia Chokri(✉), Sohaib Baroud, Safa Belhaous, and Mohammed Mestari

Laboratory SSDIA, ENSET Mohemmadia, University Hassan II, Casablanca, Morocco
chokri.soumaya90@gmail.com

Abstract. Computing machines (supercomputers) have constantly evolved to provide the greatest possible computing power for scientific applications. The trend for a decade has been clearly in favor of massively parallel architectures.

To increase computing power, increasing the frequency of processors is no longer possible; energy consumption is indeed becoming a critical issue and multi-core architectures are a serious avenue to prevent the explosion of this consumption. Parallelism is therefor an interesting solution for computation-intensive simulations and storage capacity which will have to run it on multi-core architectures. This article aims to highlight the possibilities of enhancing the parallelism of applications simulation, in particular, by improving both the partitioning and load balancing quality which are fundamental problems of parallel computing, other relevant aspects are also discussed in order to make this review as complete as possible.

Keywords: Parallel computing · Multi-core architectures · Parallelism challenges · Graph partitioning model · Data-intensive simulation

1 Introduction

Computing machines have constantly evolved to provide the greatest possible computing power for scientific applications Fig. 1. However, the computing power requirements of scientific simulations are constantly increasing, and many applications and questions have so far remained unanswered due to the insufficiency of computing resources. Over the past decades, parallelism has become an important topic of interest for a large scientific community, it has emerged as a response to huge increases requirements in computing power. it is an interesting solution for computation-intensive simulations and storage capacity which will have to run it on multi-core architectures. For example, in the field of molecular dynamics [1], especially for applications based on functional density theory like BigDFT [2], L. Genovese et al. explained that the need for computation and memory increases as the cube of the system size. For a 1,500 atom system, 3,000

M. Ben Ahmed et al. (Eds.): SCA 2020, LNNS 183, pp. 1405–1417, 2021.
https://doi.org/10.1007/978-3-030-66840-2_106

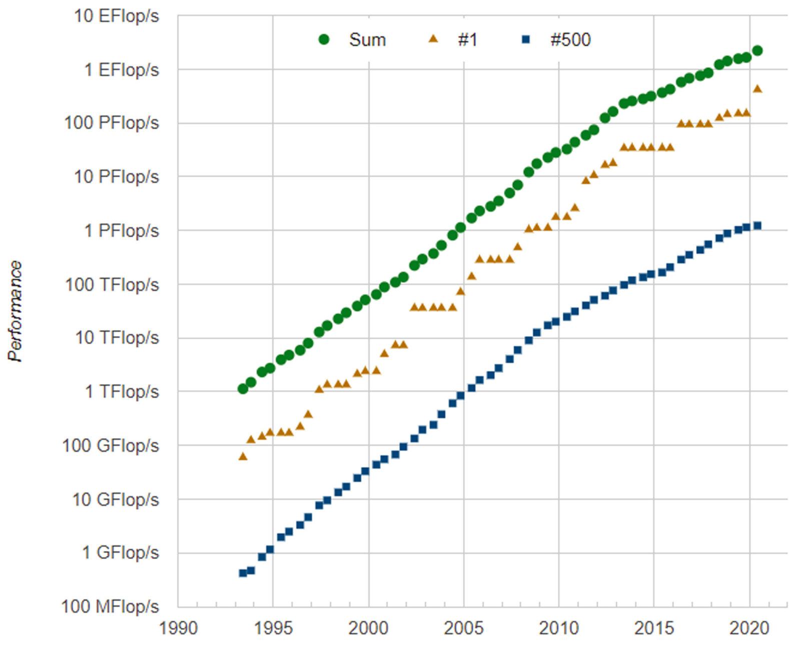

Fig. 1. Power evolution of supercomputers (source: www.top500.org).

GB of memory and one day of calculation are required on 1,500 CPU cores ina modern calculation cluster for the optimization of atomic positions. The parallelization is, therefore, necessary to reduce the execution time but also for the reasons of memory capacity. To achieve such computing capacities, parallel machines are forced to multiply the number of CPUs that will operate jointly.

To achieve such computing capacities, multi-core architectures are forced to use a large number of processors (several thousand), also called CPUs, which will operate jointly. This parallel use of CPUs is a specificity introduced by these multi-core architectures. Indeed, before the emergence of this parallelization movement, computers contained only one CPU whose operating frequency was the only characteristic allowing the performance of the machine to be evaluated. The rule was then relatively simple: the higher the frequency, the faster the CPU. Thus, it was then possible to allow a program limited by the speed of the CPU to execute more quickly without any modification. However, the increase in frequency within a CPU is limited by the appearance of multiple physical obstacles, in particular in terms of miniaturization and heat dissipation. Parallelism, therefore, appeared to be the solution to circumvent these difficulties in order to increase the performance of computers.

The parallelization of an application on several computing nodes has been studied for a long time to provide more possibility of development of the available computing power, in addition, hundreds of research have been carried out for several years in order to develop the concept of parallelism and improving its efficiency in order to improve performance.

The model graph approach is widely used to establish how to distribute the task and data for efficient parallel computation. its aim is to divide computations uniformly over p processors by distributing the vertices, which represent the set of tasks into P partition of the same size, while minimizing the inter-processor communication which is represented by edges. However, the difficulty of treatment of this problem lies in the fact that it is a multidisciplinary research field with many problems that fall within the NP-hard field, in particular, that of the balanced load distribution which is a fundamental problem relating to parallelism. Inter-processor communication, and resource allocation are also research axes among many others.

The purpose of this work is to bring together the future challenges that must be taken up by researchers in order to improve the efficiency of parallelism.

The rest of this article has been organized as follows: We first introduce the technique universally used to model the parallelization process in multi-core systems which is graph partitioning, and some existing frameworks for graph partitioning, then we discuss the different problems that hamper the performance of parallel execution of scientific simulations. Ultimately, we conclude and discuss some perspectives to overcome these challenges.

2 Related Works

2.1 Graph Partitioning Models

Computer scientists are regularly use graphs abstractions to model a simulation applications. Dividing a graph into smaller parts is one of the important algorithmic operations for parallelization.

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 Fig. 2. 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) a set of k subsets of V, denoted $P_k = V_1, V_2, ..., V_k$. We say that P_k is a partition of G if : The union of all the elements of P_k is V, and No subset of V that is an element of P_k is empty: $\bigcup_{i=0}^{k} V_i = V \quad V_i \cap V_j = \emptyset \ \forall i \neq j$, and the elements V_k of P_k are called the parts of the partition . The parts must be balanced, that is, of the same size : $|V_1| \approx |V_2| \approx ... |V_k|$, with a minimized cost (cut) function that represents the communication time between the processors: $min(\sum |e_{i,j}|), v_i \in V_k, v_j \in V_p \quad \forall \quad k \neq p$ Where: $e_{i,j}$ weight of an edge $e_{i,j} = (v_i, v_j)$.

2.2 Graph Partitioning Based Algorithms for Parallel Computing

Graph partitioning is extensively used to model the data dependencies within a computation, and it is also used for solving optimization problems that arise in

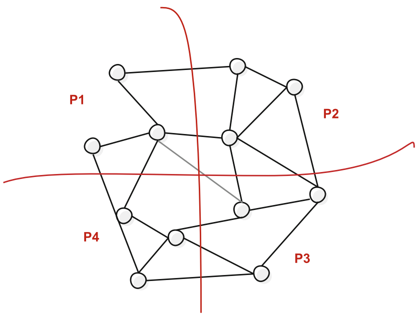

Fig. 2. Example of a graph partitioned in $K=4$ parts.

many real-world applications [30,31]. There are many examples of graph partitioning applications: data mining, design of electronic integrated circuits VLSI, load distribution for parallel machines, fluid dynamics, matrix computing, air traffic, etc.

Graph partitioning (GP) is an NP-complete problem [3,4] . We, therefore, use different heuristics to be able to calculate a partition within a reasonable time. The GP problem is well studied, and various of GP-based algorithms for data distribution and load balancing have been developed: Spectral methods [17], combinatorial approach [18], and multilevel framework [6]. The multilevel algorithm appeared as a very efficient method for calculating a k-way balanced partition of a graph [5,6]. The multi-level approach makes it possible to speed up the classical partitioning methods while maintaining good quality. This approach is broken down into three steps, as shown in Fig. 3.

- The contraction step: First, the size of the graph is reduced by merging vertices. This is repeated for several iterations, until a sufficiently small graph is obtained. A series of graphs of decreasing size has thus been created: (G_0, G_1,..., G_k).
- The initial partitioning: Once the graph has been sufficiently contracted, we apply a partitioning heuristic to calculate the partition P_k of the graph G_k. Any partitioning strategy can be applied here.
- The expansion step: The sequence of the different graphs constructed during the contraction phase is then "reassembled". The partition $P_i + 1$ of the graph $G_k + 1$ is extended on the graph G_i then this new partition P_i is refined using a heuristic which locally improves the cut.

These algorithms have various significant challenges that will be addressed later. In the literature, there are several techniques and software that give good results like: Metis [19] is one of the best known and used partitioners. It has two partitioning method: recursive or k-way but the k-way method is based on recursive bisections for its initial partitioning. ParMetis [20] it is a parallel version of Metis, allowing to create a partition in parallel on at least 2 processes. Scotch [21] is

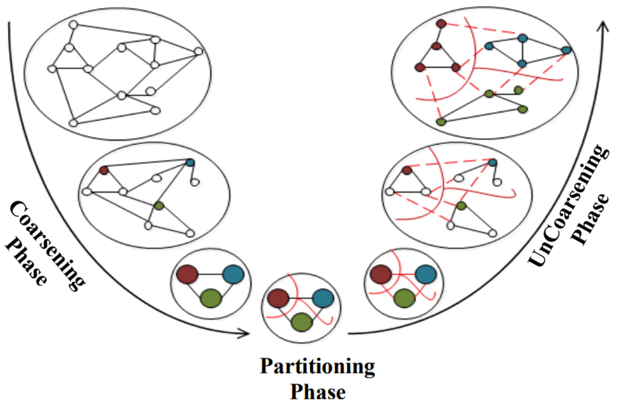

Fig. 3. Phases of a multi-level partitioning.

a free licensed graph and mesh partitioner. It is very configurable and modular. It offers a wide variety of partitioning methods, but only bisection methods are available for initial partitioning. PT-Scotch is the parallel multi-threaded, multi-process version of Scotch. Zoltan [22] is a library allowing to manage the distribution of data for parallel applications. In particular, it offers partitioner functionality.

3 Main Issues and Challenges of Parallelism

Parallelization consists of decomposing a problem into sub-problems which will be solved simultaneously on a parallel architecture. Each of the sub-problems will be treated by one or more processors of the parallel machine. Consequently, the processors of this machine do not have fast and direct access to the data of the other processors because that would constitute a bottleneck. The memory being distributed, the distribution of the calculations implies a distribution of the data. A. Meade et al. claim in [8] that the process of parallelization consist of four phases, as shown in Fig. 4:

- Decomposition phases: is to divide the problem into several sub-problems,it consists in partitioning the computations and relative data into tasks which are as independent as possible from one another. Two partitioning techniques can then be differentiated. The first is task decomposition is partition the computations first and then work with the data. The other technique; data decomposition is to first study the data needed for the problem, then look for the most suitable decomposition of these data and then identify the calculations that will be applied to them. These are two complementary techniques that applied together during the parallelization of simulation.
- Communication: In this phase,a communication model is necessary for the information exchange between the tasks.

– Load balancing: this step is a melting of partitioning and communication requirements, its aim is to divide evenly the computations into N parts of the same size, while minimizing the communication overheads.
– Mapping: during dis phase the tasks of the parallel application are assigned to the processors of the parallel architecture on which will be executed.

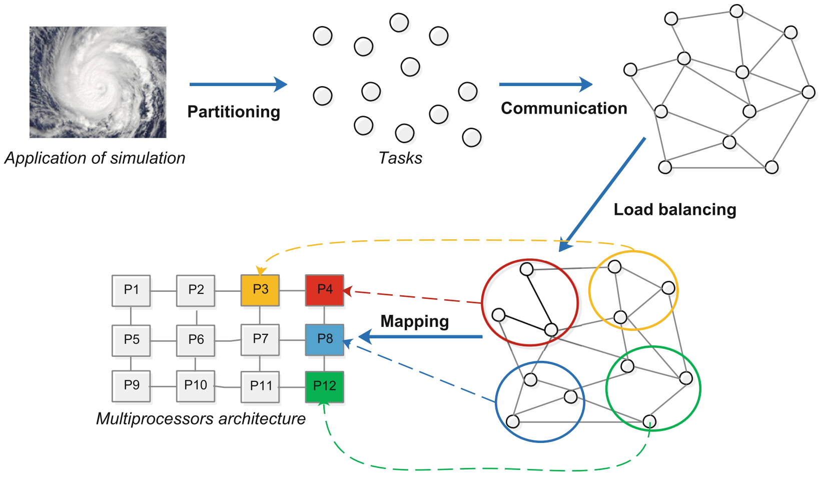

Fig. 4. Process of the parallelization of a simulation

In the literature the prallelization process has been described as more challenging and error prone [7–9,32]. Thus it is advantageous to take into consideration several issues which may be a hindrance to achieving high degrees of parallelism [10,11]: data and task decomposition, the number of processors, inter processor communication, load balancing...etc.

3.1 Data-Task Decomposition

Parallelising simulation applications to operate in parallel architectures has been described as an extremely challenging task. In particular, data decomposition is one key issue among many other [12]. The data decomposition process entails a communication cost that has a negative impact on the performance of the simulation. Consequently, it is of crucial importance to define a data decomposition strategy that increases computation while minimizing communication through an available processor. In [13,14] the authors indicate that the process of finding suitable decomposition of a complex problem is a balance of competing forces, during the decomposition, we must take into account the size of the tasks as granularity. A fine-grained decomposition creates a large number of small tasks, then increases the communication and synchronization overhead.

This might leads to poor performance, while coarse-grained decomposition may generate not enough parallelism and an unbalanced load.

Having defined a very large number of fine-grained tasks in the early stages of developing the parallel algorithm limits the efficiency of its execution on a parallel architecture. The most critical point of such execution is the cost of communication. In fact, with certain types of parallel programming models, calculations are stopped when messages are received or sent. The performance of the algorithm can, therefore, be considerably increased if the time spent communicating is reduced or covered. This improvement can, therefore, be obtained by reducing the number of messages sent. This result can also be achieved by using fewer messages while preserving the same amount of data in transit. This is because the cost of a call includes a fixed cost and is not simply proportional to the amount of data sent.

For a judicious decomposition, the partitioning of the initial problem must include more tasks than processors available on the target machine in order to be as efficient as possible during parallel execution. Otherwise, processors would end up with no task to execute. But we must be careful not to create too small tasks under the pretext of making all the processors work, because they would undoubtedly involve a large number of communications and parallelization would, therefore, be inefficient. There is therefore a compromise to be found between the size of the tasks, their number, the quantity of communication generated, as well as the number of processors to use.

3.2 Communication Cost

Another parallelization issue to consider is the communication cost between the tasks, a large number of messages to be exchanged inter task may decrease overall performance.

The cost of communication can become particularly marked when the architecture on which the program is executed is of the multi-processor type, that is to say, when the communications and exchanges between the processes/processors are carried out by the intermediary of a network. During the parallel execution of a simulation, it frequently happens that the execution time is largely dominated by the time required to carry out the communications between processors the cost of communication can be several orders of magnitude higher than the cost of execution of normal instruction. In this case, it may then be sufficient to estimate the complexity of the number of communications required by the simulation.

Several works are already carried out in this context to investigate the impact of communication on the efficiency of execution of a simulation [12,15,16]. The researchers claim that the most critical point of parallel simulation execution is the communication cost. The evolution of communication cost change depending on several factors: number of partitions , the structure of the graph, and the amount of information that any particular partition needs to send and receive...etc.

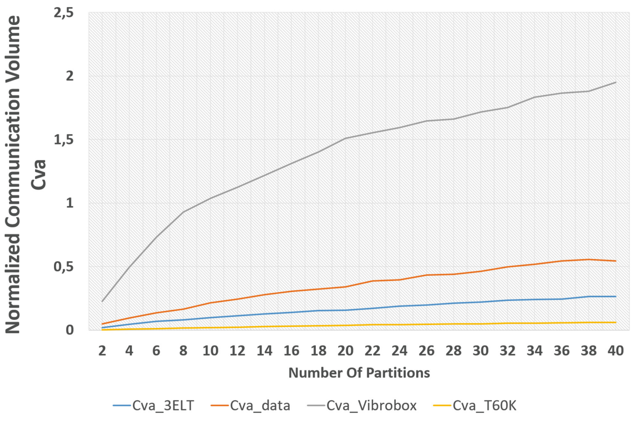

Fig. 5. Evolution of Normalized Communication Volume By increasing number of partitions

As it is shown in Fig. 5, the communication evolves rapidly by increasing the number of partitions, which certainly leads to significant degradation of efficiency as in Fig. 6. Performance can, therefore, be dramatically 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 optimum choice of the number of processors.

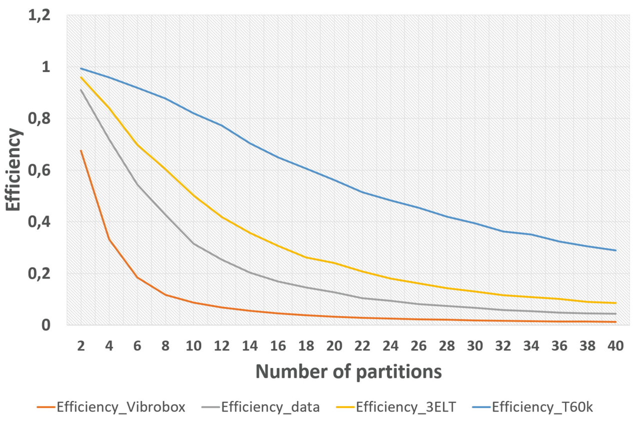

Fig. 6. Evolution of efficiency by increasing number of partitions

R. Muresano et al. in [29] indicate that performing intensive simulations on a multi-core systems balancing computational speed and efficiency is a complicated issue facing parallel computing. Therefor, communications on multi-core systems must be carefully managed in order to enhance performance.

In [16] Tintó Prims, O. et al. are explained that the paradigm of message passing then developed with the introduction of parallel distributed memory machines. Message passing involves establishing a communication channel between two execution streams to send and receive data. The MPI (Message Passing Interface) library is, in this context, the most used. Running an application implemented using this library can be very problematic when using a large number of compute nodes made up of multi-core computers. The problem is caused by the very large number of tasks relying on global communications that a loosely coupled MPI application generates. Document passing is a special, higher level case of message passing where data is typed.

3.3 Load Balancing

Load balancing is a key issue that conditions the performance of parallel numerical simulations. The goal is to distribute evenly the workload among a given number of processors, in order to minimize the overall execution time. Parallel execution of scientific simulations in a multi-core system often imposes a distribution of the workload among the available processors in order to ensure efficient parallelization. Since memory is distributed, the distribution of computations implies a data partitioning procedure. In this context, the distribution of data is done according to two objectives:

- the computational load assigned to the different processors must be balanced to minimize the computing time;
- Inter-processor communications should be minimized.

Load balancing can be done in two different ways [23]: static balancing strategy, assigns the workload to processors at the start of the simulation. In other cases, in applications with changing or unpredictable workloads during run-time, a dynamic strategy for redistributing calculations is necessary at run-time. Load balancing is one of the major challenges facing multiprocessors and multi-cores systems, however, several research works are carried out in this track [24–27], indeed algorithms have been proposed in particular, graph partitioning algorithms [5,6,17,18] described above to distribute the workload. Even if the success and relevance of the existing partitioning methods for load balancing, research challenges remain and new avenues for improvement should be proposed for the efficient execution of simulations under multi-core architectures. In particular, the choice of the right number of (partitions) processors and the efficient use of resources of a parallel architecture are key challenges of load balancing.

3.4 Resource-Aware Load Balancing

- The choice of the number of partition
  The partitioning methods for load balancing presented to calculate the distribution of a simulation's computations and the associated data over a fixed number of partitions, which can have a severe impact on the performance

and scalability of the computation. The number of parts must be taken into account during the partitioning process. Partitioning the problem on a small number of partitions can generate insufficient parallelism and an unbalanced load. However,partitioning in a large number of partitions results in poor performance due to high communication costs, etc. In this context, the choice of the number of partitions is crucial to achieving a high performance of a simulation application.

– Processors allocation for a simulation
  Another similar problem if taking into account the processors of the target machine to which the calculated partitions will be assigned. We suppose that each part will be assigned to a processor, choosing the right number of processors allocated to a simulation is essential to have good performance or efficiency. If a simulation is run in parallel on too many processors, the time spent communicating can become too long compared to the computation time. Using as many processors as possible is not always a good choice depending on the size of the problem. As the size of the problem may vary during simulation, the number of processors should vary accordingly.
  The resource-aware load balancing issue is not well studied, however, there are researchers who affirm that among the issues encountered to reach a maximum speed up it is the number of processors. In [28] L. yang et al. claim that the pan-sharpening algorithms are data and computation intensive, therefore, they have adopted a parallel strategy to solve the existing problems on a multi-core computer. however, they show that the choice of the number of processors is a crucial and essential issue to achieve maximum speed. It is a difficult and complicated problem to select a specific number of processors for an application, as the factors determining the maximum speed are different and varied. They give empirical suggestions based on experimental results to define the number of processors in order to efficiently use the available resource.

4 Perspectives and Future Works

In order to address the different issues and challenges presented in this work, several avenues are considered and many different directions to explore as future work:

4.1 Computation-Data Decomposition

For efficient partitioning, the data-computation decomposition of the initial problem must include more tasks than processors available on the target machine in order to be as efficient as possible during parallel execution. Otherwise, processors would end up with no task to execute. But we must be careful not to create too small tasks under the pretext of making all the processors work, because they would undoubtedly involve a large number of communications and parallelization would therefore be inefficient. There is, therefore, a compromise to be found between the size of the tasks, their number, the quantity of communication generated, as well as the number of processors to use.

4.2 Communication Cost

Knowing that the communication cost of an application depends on several factors such as the topology of the graph, the number of messages sent and received, the number of partitions, ... etc, therefore the optimization of any of these factors is an unaffordable solution, for this reason, a thorough performance analysis approach is required to identify bottlenecks and understand the impact of inter-process communication on the performance.

Once these bottlenecks are revealed and the features impacting model performance are defined, it would be appropriate to adopt a learning system such as an artificial neural network (ANN) to predict in a more or less precise way the maximum speedup and the ideal number of partitions (processors) for an application.

4.3 Resource Allocation and Load Balancing

To address this challenge, it appears advantageous to develop an ANN predictive module to predict the appropriate processor for each workload in order to enhance the energy-efficiency and performance.

Resource allocation and load balancing are very useful in code coupling, several parallel codes run simultaneously and must regularly exchange data. This exchange phase is synchronizing. It is therefore, important that all the codes concerned progress at the same speed to minimize the waiting time during this synchronization.

The choice of the number of processors used by each code must be made by taking into account the relative loads of each. This balancing between several codes can be difficult, more particularly if a load of these codes can vary dynamically. To rebalance these codes, one solution would, therefore, be to reallocate resources from one code to another. The ideal number of processors allocated to each code can therefore be approached experimentally by correcting during the simulation the imbalances which would occur.

5 Conclusion

The computing power and frequency limitations remarked on single-core machines have paved the way for multi-core systems and will be the industry trend to move forward. However, full performance throughput can only be achieved when the challenges of running simulations on multi-core processors are fully resolved.

A graph partitioning model to establish how to distribute the task and data for efficient parallel computation is presented, and the existing algorithms in the literature are detailed.

This paper also highlights the key issues and challenges of performing simulations on multi-core systems, especially computation-data partitioning, communication overhead, resource-aware load balancing...etc. In addition, we try to cite possible avenues for improvement in order to enhance the performance execution on multi-core systems.

Acknowledgements. This project has received funding from the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No 777720.

References

1. Phillips, J.C., Braun, R., Wang, W., Gumbart, J., Tajkhorshid, E., Villa, E., Chipot, C., Skeel, R.D., Kalé, L., Schulten, K.: Scalable molecular dynamics with NAMD. J. Comput. Chem. **26**(16), 1781–1802 (2005)
2. Genovese, L., Neelov, A., Goedecker, S., Deutsch, T., Ghasemi, S.A., Willand, A., Caliste, D., Zilberberg, O., Rayson, M., Bergman, A., Schneider, R.: Daubechies wavelets as a basis set for density functional pseudopotential calculations. J. Chem. Phys. **129**, 014109 (2008)
3. Garey, M.R., Johnson, D.S.: Computers and Intractibility: A Guide to the Theory of NP-Completeness. Freeman W. H., New York City (1979)
4. Andreev, K., Racke, H.: Balanced graph partitioning. In: Proceedings of the Sixteenth Annual ACM Symposium on Parallelism in Algorithms and Architecturesn (SPAA), pp. 120–124. ACM (2004)
5. Pellegrini, F.: Graph partitioning based methods and tools for scientific computing. Parallel Comput. **23**, 153–164 (1997)
6. Karypis, G., Kumar, V.: A fast and high quality multilevel scheme for partitioning irregular graphs. SIAM J. Sci. Comput. **20**(1), 359–392 (1998)
7. Massingill, B.L., Mattson, T.G., Sanders, B.A.: Reengineering for Parallelism: an entry point into PLPP for legacy applications. Concurr. Comput.: Pract. Experience **19**(4), pp. 503–529 (2007)
8. Meade, A. , Buckley, J. , Collins, J.J. . Challenges of evolving sequential to parallel code: an exploratory review. In: Proceedings of the 12th International Workshop on Principles of Software Evolution and the 7th annual ERCIM Workshop on Software Evolution. ACM (2011)
9. Vandierendonck, H., Mens, T.: Averting the next software crisis. IEEE Comput. **44**(4), 88–90 (2011)
10. Nielsen, I.M., Janssen, C.L.: Multicore challenges and benefits for high performance scientific computing. Sci. Program. **16**(4), 277–285 (2008)
11. Mccool, M.: Scalable programming models for massively multicore processors. Proc. IEEE **96**(5), 816–831 (2008)
12. Meade, A., Deeptimahanti, D.K., Buckley, J., Collins, J.J.: An empirical study of data decomposition for software parallelization. J. Syst. Softw. **125**, 401–416 (2017)
13. Dovolnov, E. , Kalinov, A. , Klimov, S. Natural block data decomposition for heterogeneous clusters. In: Proceedings of the International Parallel and Distributed Processing Symposium (2003)
14. Massingill, B., Mattson, T., Sanders, B.: Reengineering for parallelism: an entry point into PLPP for legacy applications. Concurr. Comput.: Pract. Experience **19(**4) 2007, 503–529 (2007)

15. Chokri, S., et al.: Impact of communication volume on the maximum speedup in parallel computing based on graph partitioning. In: 2019 Third International Conference on Intelligent Computing in Data Sciences (ICDS), pp. 1–6 (2019)
16. Tintó Prims, O., et al.: Finding, analysing and solving MPI communication bottlenecks in earth system models. J. Comput. Sci. **36**, 100864 (2019)
17. Fjällström, P.: Algorithms for Graph Partitioning: A survey. Linköping University, Linköping, Sweden, Department of Computer and Information Science (1998)
18. Kernighan, B.W., Lin, S.: An efficient heuristic procedure for partitioning graphs. Bell Syst. Tech. J. **49**(2), 291–307 (1970)
19. George K.: METIS- Family of Multilevel Partitioning Algorithms. http://glaros.dtc.umn.edu/gkhome/views/metis
20. George K.: PARMETIS. http://glaros.dtc.umn.edu/gkhome/metis/parmetis/overview
21. François P.: Scotch and LibScotch 6.0 User's Guide (2012)
22. Zoltan: Parallel partitioning, load balancing and data-management services. http://www.cs.sandia.gov/Zoltan/Zoltan.html
23. Bohme, D.: Characterizing Load and Communication Imbalance in Parallel Applications, Volume 23 of IAS, Forschungszentrum Julich (2014). https://doi.org/10.1109/IPDPSW.2012.321
24. Campbell, P.M., Devine, K.D., Flaherty, J.E., Gervasio, L.G., Teresco, J.D.: Dynamic octree load balancing using spacefilling curves. Technical Report CS-03-01, Williams College Department of Computer Science (2003)
25. Cybenko, G.: Dynamic load balancing for distributed memory multiprocessors. J. Parallel Distrib. Comput. **7**(2), 279–301 (1989)
26. Pilkington, J.R., Baden, S.B.: Dynamic partitioning of non-uniform structured workloads with spacefilling curves. IEEE Trans. Parallel Distrib. Syst. **7**(3), 288–300 (1996)
27. Niemoller, A., Schlottke-Lakemper, M., Meinke, M., Schroder, W.: Dynamic Load balancing for direct-coupled multiphysics simulations. Comput. Fluids **199**, 104437 (2020)
28. Yang, J., Zhang, J., Huang, G.: A parallel computing paradigm for pan-sharpening algorithms of remotely sensed images on a multi-core computer. Remote Sens. **6**, 6039–6063 (2014)
29. Muresano, R., Meyer, H., Rexachs, D., Luque, E.: An approach for an efficient execution of SPMD applications on multi-core environments. Future Gener. Comput. Syst. **66**, 11–26 (2017)
30. Wang, N., Wang, Z., Gu, Y., Bao, Y., Yu, G.: TSH: easy-to-be distributed partitioning for large-scale graphs. Future Gener. Comput. Syst. **101**, 804–818 (2019)
31. Kang, U., Tsourakakis, C.E., Faloutsos, C.: Pegasus: a peta-scale graph mining system implementation and observations. In: Proceedings of ICDM, pp. 229–238. IEEE (2009)
32. Belhaous, S., Baroud, S., Chokri, S., Hidila, Z., Naji, A., Mestari, M.: Parallel implementation of a search algorithm for road network. In: 3rd International Conference on Intelligent Computing in Data Sciences (ICDS 2019) (2019)