Journal of Parallel and Distributed Computing 169 (2022) 211–225

Contents lists available at ScienceDirect

Journal of Parallel and Distributed Computing

www.elsevier.com/locate/jpdc

Optimal load balancing and assessment of existing load balancing criteria

Anthony Boulmier [a,*], Nabil Abdennadher [b], Bastien Chopard [a]

[a] University of Geneva, Department of Computer Science, Route de Drize 7, 1227 Carouge, Switzerland
[b] University of Applied Sciences and Arts, Western Switzerland (HES-SO), Rue de la Prairie 4, 1202 Geneva, Switzerland

ARTICLE INFO

Article history:
Received 4 April 2021
Received in revised form 7 May 2022
Accepted 7 July 2022
Available online 16 July 2022

Keywords:
High performance computing
Parallel computing
Dynamic load balancing
Load balancing criteria
Performance optimization

ABSTRACT

Parallel iterative applications often suffer from load imbalance, one of the most critical performance degradation factors. Hence, load balancing techniques are used to distribute the workload evenly to maximize performance. A key challenge is to know *when* to use load balancing techniques. In general, this is done through load balancing criteria, which trigger load balancing based on runtime application data and/or user-defined information. In the first part of this paper, we introduce a novel, automatic load balancing criterion derived from a simple mathematical model. In the second part, we propose a branch-and-bound algorithm to find the load balancing iterations that lead to the optimal application performance. This algorithm finds the optimal load balancing scenario in polynomial time while, to the best of our knowledge, it has never been addressed in less than an exponential time. Finally, we compare the performance of the scenarios produced by state-of-the-art load balancing criteria relative to the optimal load balancing scenario in synthetic benchmarks and parallel N-body simulations. In the synthetic benchmarks, we observe that the proposed criterion outperforms the other automatic criteria. In the numerical experiments, we show that our new criterion is, on average, 4.9% faster than state-of-the-art load balancing criteria and can outperform them by up to 17.6%. Moreover, we see in the numerical study that the state-of-the-art automatic criteria are at worst 26.43% slower than the optimum and at best 10% slower.

1. Introduction

Parallel iterative applications often exhibit an irregular computational scheme that may create load imbalance over time. Load imbalance is a major performance degradation factor. For that purpose, dynamic load balancing mechanisms are used throughout the application execution to keep processing elements' workloads evenly distributed and their communications minimized. Those mechanisms involve two separate questions *how* and *when* to load balance [25]. "How to load balance" is related to finding the algorithm that divides the computational domain (partitioning algorithm) into several pieces that are distributed (mapping algorithm) on the available processing elements while minimizing their communications. "When to load balance" defines the particular iterations (i.e., a scenario) at which the load balancing mechanism (i.e., using the partitioning and mapping algorithm) is required. Their goal is to minimize the application wall time.

"How to load balance" has been explored by several authors over the years leading to various partitioning and mapping algorithms. In particular, the partitioning algorithm consists of solving a balancing graph partitioning problem, which is known to be NP-Complete [10]. Hence, heuristics have been developed, exhibiting good balancing capabilities for various types of problems. Among the most famous, recursive coordinate bisection (RCB) [30], space-filling curves (SFC) [1], recursive spectral bisection [33], and METIS (multilevel k-way) [14] can be mentioned. For more sophisticated techniques, we suggest the reader to refer to [8,7,9]. However, it is difficult for scientists to know how well a particular technique will perform on their own problem. Moreover, due to the complexity of modern algorithms and the lack of "plug and play" libraries, scientists often use the most famous load balancing techniques, which may not be optimal for their problem. In addition, we pointed out in a previous work that researchers should not select a load balancing technique only based on its capability to correct imbalance but also during how many iterations it keeps a low level of imbalance [3]. This further increases the difficulty to select the most optimal technique. To overcome this challenge, researchers have proposed algorithms to select automatically the most suitable load balancing techniques based on application data [24,18,2].

* Corresponding author.
E-mail address: anthony.boulmier@unige.ch (A. Boulmier).

https://doi.org/10.1016/j.jpdc.2022.07.002

"When to load balance" is a challenging problem that involves finding the iterations at which a parallel iterative application should trigger its load balancing mechanisms to maximize performance. Herein, we refer to the optimal load balancing scenario as the sequence of iterations where the load balancing is applied such that the application wall time is minimized. As the load balancing itself has a cost (compute the new partition and migrate the data), one (most of the time) can not simply re-balance every iteration because the load balancing cost C overcomes the performance gain. To find the optimal scenario for analysis purposes, one straightforward way is to try every load balancing scenario and keep the one that yields the best performance. However, this is unfeasible in practice, even for a small number of iterations. Indeed, for an application comprising of γ iterations, the number of scenarios is 2^γ.

In the literature, load balancing criteria have been proposed to decide whether the load balancing mechanism should be triggered or not. A load balancing criterion is a condition based on application information and/or user data. One of the most straightforward criteria re-balances the application every T iterations enabling the correction of recurring imbalance. However, this is inefficient when load imbalance exhibits a non-periodic pattern. Some more sophisticated criteria use mathematical models taking into account collected data, such as the unbalancing pace (i.e., workload increase rate), the load balancing cost, the expected scalability, the maximum authorized imbalance, and others. For instance, Marquez et al. [4] propose to apply the load balancing algorithm when at least one of the processing elements is below (respectively above) a pre-defined workload lower bound (respectively upper bound). Procassini et al. [27] predict the time per iteration post load balancing using an estimation of the efficiency's improvement and trigger the re-balancing mechanism when the increase in time per iteration is greater than the load balancing cost. Menon et al. [19] propose to re-balance the application when the cumulative load imbalance (i.e., the sum of the current imbalance over time) overcomes the load balancing cost. Pearce et al. [24] perform a cost-benefit analysis of the load balancing process. They use a load model to estimate both the cost of load balancing with various algorithms and the benefit of correcting the imbalance. They activate the load balancing mechanism if its benefit is greater than its cost. Finally, the wide choice of criteria makes the choice of a suitable criterion difficult due to the lack of rigorous comparative studies. Worse, because it is hard to find the optimal load balancing scenario among all the possible candidates, there is no clue how far the performance of the scenario produced by a load balancing criterion is from the optimal scenario's performance. Therefore, finding the optimal scenario and quantifying its performance is an important and challenging task.

This paper introduces a load balanced application theoretical model to derive a novel, automated load balancing criterion that performs at worse on par with state-of-the-art load balancing criteria. Moreover, we propose a new method derived from the A^* algorithm [11] to find the iterations at which the load balancing mechanism must be used to obtain optimal performances. This method can be applied to real applications and synthetic benchmarks built with our theoretical model. Then, we use the optimal scenario to evaluate the performance of several state-of-the-art load balancing criteria on various synthetic benchmarks. Such a study provides insights into the performance gap between state-of-the-art criteria and the optimum. Finally, we implement our novel algorithm in an N-body simulation and discuss the difference of performance and behavior between state-of-the-art load balancing criteria and the optimal scenario. The result of our efforts also includes two implementations of our novel algorithm. The first is a standalone package for studying optimal scenarios within synthetic benchmarks, and the second is an implementation for real applications.

Section 2 proposes a definition of the load balancing decision problem and introduces the challenges to solve it. Section 3 presents the background works related to load balancing criteria. Section 4 introduces a model for parallel applications with dynamic load balancing and shows how to derive a novel, fully automatic load balancing criterion based on the past and current behavior. Section 5 presents an efficient algorithm to find the optimal load balancing scenario. Section 6 assesses the performance of load balancing criteria with respect to the optimal scenario in synthetic benchmarks and within a parallel N-body simulation. Section 7 concludes this work and proposes insight for future works.

2. The load balancing decision problem

Consider an iterative parallel application (e.g., N-body, computational fluid dynamics, etc.) comprising of γ iterations. Computing such an application in parallel on P processing elements requires distributing its workload among the processing units used for the computation while minimizing communications. The time per time-step is equal to the time of the slowest (or most loaded) processing element due to synchronization mechanisms at the end of each iteration. To maximize efficiency, the workload attributed to each processing element must be roughly equal at each iteration. This is achieved through load balancing algorithms that mitigate the load imbalance penalty. For parallel applications that do not exhibit a dynamic nature, only one load balancing is required at the beginning of its execution. This is usually called static load balancing. In contrast, when the processing elements' workload is not the same from iteration to iteration, several load balancing steps may be required. This is known as dynamic load balancing. We call the set of iterations at which the load balancing algorithm is used the "load balancing scenario". The processing elements must coordinate and take a load balancing decision at each iteration (re-balancing or not) to create a scenario. This decision process leads to 2^γ possible scenarios where γ is the number of iterations. The dynamic load balancing decision problem consists of finding the optimal scenario, minimizing the application wall time.

Definition 2.1 *(Dynamic load balancing decision problem).* Given an application comprising of γ iterations and P processing elements, find the set of iterations σ^* (i.e., the scenario) at which the load balancing mechanism must be activated such that the application wall time is minimized.

This decision problem is an optimization problem in which we look for

$$\sigma^* = \underset{\sigma \in \mathcal{S}}{\operatorname{argmin}}\ T(\sigma), \tag{1}$$

where $T(\sigma)$ is a function returning the application wall time given a load balancing scenario and σ is a particular scenario among the 2^γ possible ones (S). Note that $T(\sigma)$ can either be modeled by an equation or computed by the application code itself (i.e., actually measured on a computer).

Solving this problem is non-trivial as the load balancing benefit usually depends on the application's future behavior, the moment at which the load balancing is applied, and the success of the data partitioning. Let us imagine an application where the load imbalance is ephemeral. Molecular dynamic applications may see such behaviors. For instance, the particle density across the computational domain can change periodically due to some forces. Therein, it is unclear whether re-distributing the particles would be beneficial due to the load balancing cost. To accurately answer this

question, one would have to predict that such behavior happens. This implies that the application is predictable (in the long term), which appears to be unfeasible [24]. Therefore, load balancing decisions can only be built on strong and reliable metrics based on prior data.

Asking whether re-balancing the workload is required or not depends on multiple factors. In the literature, scientists base their decision on load balancing criteria that employ various metrics such as the parallel efficiency, the load imbalance, the iteration index, the min/max workload, and others. Usually, a parallel application needing load balancing capability would implement a criterion to decide when to redistribute the workload among the processing elements attributed to the computation of the application. A criterion is essentially an equation that must be evaluated each time a load balancing decision must be taken. Note that while being unaware of system perturbations (e.g., cache misses, OS interrupts, or temporary system faults or malfunctions [21]) that can alter the time-per-iteration of some processing elements, load balancing criteria implicitly take them into account because their decision making is based on load imbalance metrics. Hence, if some perturbations exacerbate the load imbalance, the load balancing criterion will execute the load balancing algorithm to improve the situation. Bear in mind that it is the task of the load balancing algorithm to take care of producing the most suited partitioning by, for instance, ensuring that slower processing elements facing perturbations (e.g., cache contention, off-chip bus saturation, NUMA effects) get less work. However, some open questions remain to be answered, for instance: (i) how to build a perturbation aware load balancing technique?; and (ii) how does a load balancing algorithm would work in conjunction with advanced software and hardware techniques, such as dynamic concurrent throttling (DCT), dynamic voltage and frequency scaling (DVFS), or, more challenging, a combination of both [22]?. How would the algorithm ensure that the produced partitions are well balanced and take into account the overhead of such techniques [29]. These questions, despite being challenging and interesting, are not addressed in the present paper and are left for future works. For a review of load imbalance metrics, we suggest the reader to refer to [28]. We define the criteria that use local information as *local criteria*, whereas the other criteria are considered as *global criteria*. Local information is a data that is related to a single processing element, such as the current processing element workload, the processing element workload increase rate, etc. In contrast, global information concerns all processing elements, such as the time per iteration, the average workload, or the load balancing cost. In the next section, we dig into more details in the various load balancing criteria proposed over time by researchers.

3. Background works

In the literature, scientists often use straightforward load balancing criteria while it is well known that without fine-tuning, they provide poor performance [24,19]. For instance, Fattebert et al. [7], Offenhäuser [23], and Lieber et al. [16] chose to load balance their application respectively every 100, 1000, and 180 iterations while Ishiyama et al. [13] re-balanced every iteration. The rationales behind these choices are manifold. Some argue that the load balancing cost is negligible compared to load imbalance [13], while others use application knowledge to tune their criterion. More recently, Miller et al. [20] performed a study to improve the load balancing in their particle-in-cell code. In this study, they proposed a new shared-memory load balancing strategy, that they apply every 20 and 40 solver iterations, that improves the performance of their application by a factor 2 compared to a no load balancing approach. Another work performed by Prät et al. [26] focused on improving the performance of an adaptive mesh refinement based molecular dynamic application using multi-threading, vectorization friendly data-structure, and dynamic load balancing. In their work, the authors claim to outperform LAAMPS by a factor 1.38 on a micro-jetting scenario and by 2.6 in a steady scenario. They achieve these levels of performance with the RCB algorithm applied every 500 timesteps.

Unfortunately, all these previous works do not study their load balancing triggering strategy. Hence, it is hard to know if they are effective (or not) and why. Moreover, we will see later in this paper that a bad load balancing criterion can suffer from a huge performance penalty compared to the optimal scenario. Indeed, it is likely that many of the load balancing calls are unnecessary, ill-timed, or worse, the application may still suffer from load imbalance. For that purpose, researchers have tried to develop more sophisticated and generic criteria that provide better overall performance.

Marquez et al. [4] have proposed a load balancing criterion based on an acceptable workload variation range for agent-based simulations. The idea is to trigger the load balancing mechanism if any agent's workload goes outside of a comfort zone defined by a minimal acceptable workload $W_{\min}$ and a maximal acceptable workload $W_{\max}$. In other words, when the following condition is true:

$$W_p < W_{\min} \text{ or } W_{\max} < W_p \ \exists p = 1..P. \tag{2}$$

This criterion is considered local as the formula uses the local workload of processing element W_p. The formula proposed by Marquez et al. can be implemented using a "tolerance factor" ξ, which specifies how far a single processing element can get away from the average workload. Then, Equation (2) can be rewritten

$$W_p < \frac{(1-\xi)}{P}\sum_{1}^{P} W_p \text{ or } \frac{(1+\xi)}{P}\sum_{1}^{P} W_p < W_p. \tag{3}$$

Indeed, the tolerance factor has to be tuned by hand as the value may differ from application to application, making it difficult to find a good value for this parameter. Moreover, within a single application, the tolerance factor that provides the best performance may change over time. Unfortunately, an automatic selection of the acceptable workload range has never been proposed.

Procassini et al. [27] use a different strategy to automatically load balance HPC applications. Their global criterion redistributes the workload whether the performance improvement due to load balancing plus the load balancing cost is greater than a fraction of the current time per iteration. In other words, the load balancing mechanism is triggered at iteration t when the following condition is true:

$$T_{\text{withLB}}(t) + C < \rho * T_{\text{withoutLB}}(t), \tag{4}$$

where $T_{\text{withLB}}(\cdot)$ is the iteration time after load balancing, C is the load balancing cost in seconds, ρ is the desired increase in performance post load balancing, and $T_{\text{withoutLB}}(\cdot)$ is the iteration time before load balancing. In their paper, they used $\rho = 0.9$. However, the same idea can be generalized for any $\rho \in \mathbb{R}^{>0}$. Procassini et al. estimate the time per iteration post load balancing by decreasing the current time per iteration proportionally to the expected increase in performance due to load balancing. This reads

$$T_{\text{withLB}}(t) = \frac{\varepsilon_{\text{pre}}(t)}{\varepsilon_{\text{post}}(t)} * T_{\text{withoutLB}}(t), \tag{5}$$

where $\varepsilon_{\text{post}}$ (resp. ε_{pre}) is the parallel efficiency post (resp. pre) load balancing step. While the parallel efficiency post load balancing has to be estimated based on prior data, the efficiency before load balancing is computed with

Table 1
Summary of available load balancing criteria. Note that the periodic criterion belongs to the common knowledge of load balancing. Tracking back its origin is complicated.

Name	User defined parameters	Required data	Type	Foundation	Developed for	Decision
Periodic	Load balancing period T	-	Global	-	Any simulations	Every T it.
Marquez et al. [4]	Tolerance factor ξ	- PEs workload	Local	Experiments	Agent-based simulations	Equation (2)
Procassini et al. [27]	Desired performance improvement post load balancing ρ	- Efficiency - Load Balancing Cost	Global	Experiments	Monte-Carlo Transport	Equation (4)
Menon et al. [19]	-	- Imbalance Increase Rate - Load Balancing Cost	Global	Theory	Any simulations	Equation (19)
Zhai et al. [36]	Evaluation phase $\mathcal{P}$	- Imbalance Increase Rate - Load Balancing Cost	Global	Experiments	Any simulations	Equation (8)
Our criterion	-	- Imbalance time [5] - Load Balancing Cost	Global	Theory	Any simulations	Equation (24)

$$\varepsilon_{\text{pre}}(t) = \frac{T_{\text{seq}}(t)}{P * T_{\text{par}}(t)}. \tag{6}$$

The presence of the factor ρ, which must be fixed by hand, makes the tuning of this criterion for optimal performance difficult. Note that Lieber et al. [16] also implemented an "auto-mode" into their application (FD4), which employs a simple cost-benefit analysis of the load balancing process. The criterion utilized therein is analog to Equation (4), except that they use $\rho = 1$ and they estimate the time post-load balancing using data collected from previous load balancing steps.

Menon et al. [19] have shown that the optimal load balancing scenario for a parallel iterative application, where the maximum and average load can be modeled linearly with time, is a fixed rebalancing frequency. The load balancing time interval τ is equal to the amount of iteration required by the cumulative load imbalance to reach the load balancing cost C. When the workload increase rate is constant, it can be computed by the following formula:

$$\tau = \sqrt{\frac{2C}{\alpha}}, \tag{7}$$

where C is the load balancing cost in seconds and α is the difference in the time-per-iteration increase rate between the "slowest" processing element and the average time-per-iteration increase rate. They derived this global criterion by minimizing the time with respect to the load balancing time interval. Like the criterion proposed by Procassini et al. [27], the information used therein is measured and updated throughout application execution. For instance, the load balancing cost C has to be estimated while the maximum and average workload increase rates are measured at runtime. We refer to criteria analog to Menon criterion as Menon's like criteria.

Zhai et al. [37] have used Menon criterion to improve the performance of CMT-nek. CMT-Nek is a compressible multiphase turbulence application, which enhances the physics of the CESAR Nek5000 application. In particular, they proposed to compute the cumulative time-per-iteration degradation $\mathcal{D}$ during application execution and to trigger a load balancing call when it has reached the load balancing cost C or after τ iterations, as suggested by Menon, leading to this global load balancing criterion:

$$\mathcal{D} \geq C \text{ or } i \equiv 0 \bmod \tau, \tag{8}$$

where i is the current iteration index and the cumulative time-per-iteration degradation $\mathcal{D}$ from the last load balancing iteration LB_p up to the current iteration t is computed using

$$\mathcal{D} = \sum_{i=LB_p}^{\text{t}} \Big(T_{\text{median}}(i, i-2) - T_{\text{avg}}(\mathbb{P}) \Big), \tag{9}$$

where $T_{\text{avg}}(\mathbb{P})$ is the average time per time-step over an user-defined evaluation phase $\mathbb{P}$ and $T_{\text{median}}(i, i-2)$ is the median time per time-step among the three last iterations.

Recently, Mayr et al. [17] have proposed a new load balancing criterion for simulations of contact problems using Mortar methods. Therein, they measure two specific quantities K_t and K_c that must not cross their respective user-defined threshold ν_t and ν_c. K_t is defined as the ratio of the largest by the smallest mortar evaluation time, whereas K_c is defined as the ratio of the largest by the smallest number of elements in the contact zone. In the case where ν_t or ν_c is crossed, the load balancing algorithm is executed. Despite that this strategy seems to work well on their problem compared to a static approach, this work lacks from showing that the proposed load balancing criterion is close to the optimal solution or outperforms other criteria for contact problems. Note that this load balancing criterion will not be used in our experiments later in the paper due to its tight link to the simulations of contact problems.

Finally, the literature lacks rigorous load balancing criteria comparison studies, which would be hard to perform due to the absence of algorithms capable of computing the optimal scenario. Indeed, researchers may think that a load balancing criterion is performing well even though it is, in fact, far from the optimal solution. Therefore, the present work proposes to fill this gap by systematically comparing several state-of-the-art load balancing criteria against the optimal load balancing scenario. The optimal load balancing scenario is computed in polynomial time using our novel algorithm presented in Section 5. Also, one difficulty for HPC developers regarding load balancing is to choose the good load balancing criteria. Indeed, all the criteria available in the literature bring confusion and only a few are backed up by a strong theory. We summarize the load balancing criteria described above in Table 1 to ease the choice of HPC researchers. This table details what we find to be the most useful properties of load balancing criteria. In addition, in Section 5, we propose an efficient branch-and-bound algorithm for finding the optimal load balancing scenario to compare load balancing criteria relative to the optimum and help the selection of load balancing criteria.

4. A workload-aware load balancing criterion

To study the performance of the scenarios produced by load balancing criteria relative to the optimal load balancing scenario's performance, we propose a mathematical framework for computing the CPU time of load balanced parallel applications inspired by

Menon's work [19]. First, let us consider a parallel execution on P processors characterized by two functions, $\mu(t)$ and $m(t)$. The function $\mu(t)$ gives at each iteration t the average load (i.e., the total load on the P processors divided by P), whereas $m(t)$ gives the load of the slowest (or most loaded) processor at iteration t. Note that here, $\mu(t)$ and $m(t)$ are expressed with units of time. Furthermore, let us assume that executing the load balancing mechanism always leads to perfect load balancing.

Second, let us define T_{par} the parallel time on γ iterations given by

$$T_{\text{par}} = \int_0^\gamma m(t)\,\mathrm{d}t = \int_0^\gamma m(t) - \mu(t)\,\mathrm{d}t + \int_0^\gamma \mu(t)\,\mathrm{d}t. \tag{10}$$

Note that this equation has been greatly inspired from the model of Menon et al. [19], however, herein we relax the assumption that $m(t)$ and $\mu(t)$ are represented by line equations. Let us now divide the interval $[0, \gamma]$ in n pieces $[s_i, s_{i+1}]$ with $s_0 = 0$, $s_{i+1} > s_i$ and $s_n = \gamma$. Moreover, we assume that load balancing steps are performed at iterations s_i for $i = 0, 1, \ldots n-1$ and they take an additional time C. Hence, Equation (10) becomes

$$T_{\text{par}} = \sum_{i=0}^{n-1} \Big(\int_{s_i}^{s_{i+1}} u_i^*(t)\mathrm{d}t + C \Big) + \int_0^\gamma \mu(t)\,\mathrm{d}t, \tag{11}$$

where $u_i^*(t)$ is the imbalance time metric proposed by DeRose et al. [5] defined as

$$u_i^*(t) = m(t) - \mu(t) \text{ for } t \in [s_i, s_{i+1}]. \tag{12}$$

Also, we point out that load balancing is done at s_0 but not at the end of the execution (i.e., at s_n). Obviously, $m(t)$ resets to $\mu(t)$ after every load balancing step if the load is perfectly balanced. Thus, always $u_i^*(s_i) = 0$. To increase the readability of Equation (11), let us express it with the following change of variables

$$\tau_i = s_{i+1} - s_i \qquad u_i(x) = u_i^*(t - s_i). \tag{13}$$

Equation (11) now becomes

$$T_{\text{par}} = \sum_{i=0}^{n-1} \Big(\int_0^{\tau_i} u_i(x)\mathrm{d}x + C \Big) + \int_0^\gamma \mu(t)\,\mathrm{d}t. \tag{14}$$

Remark 1. In general, $u_i(x)$ is unpredictable because it depends on s_i and the load balancing technique itself. Indeed, the domain decomposition used by the load balancing mechanism affects the load imbalance growth. In section 6.1, we give a possible solution to this challenge when we use our model as a framework for synthetic benchmarks.

Derivation of Menon criterion. To derivate the criterion from Menon et al. [19] using Equation (14) we need to set $u_i(x)$ as a linear equation such as

$$u_i(x) = u(x) = \alpha x. \tag{15}$$

Then, Equation (14) reads

$$T_{\text{par}} = \frac{\gamma}{\tau}\Big(\int_0^\tau \alpha x\mathrm{d}x + C \Big) + \int_0^\gamma \mu(t)\,\mathrm{d}t. \tag{16}$$

Note that we obtain here the same equation as in the paper of Menon et al. [19]. The optimal value of τ is then obtained by solving and isolating τ in

$$\frac{\partial T_{\text{par}}}{\partial \tau} = 0. \tag{17}$$

That is,

$$\begin{aligned} \frac{\partial T_{\text{par}}}{\partial \tau} &= 0 \\ -\frac{\gamma}{\tau^2}\Big(\frac{\alpha\tau^2}{2} + C\Big) + \frac{\gamma}{\tau}\alpha\tau &= 0 \\ \frac{\alpha}{2} - \frac{C}{\tau^2} &= 0 \\ \tau &= \sqrt{\frac{2C}{\alpha}} \end{aligned} \tag{18}$$

It is worth noticing that for this value of τ, one has

$$\int_0^\tau u(t)\mathrm{d}t = \frac{\alpha\tau^2}{2} = C. \tag{19}$$

In other words, the load balancing mechanism must be used when the load imbalance metric $u(t) = m(t) - \mu(t)$ accumulated over the iterations reaches the load balancing cost C. For the sake of simplicity, this quantity reads

$$U = \int_0^\tau u(t)\mathrm{d}t. \tag{20}$$

Remark 2. In this case, where $u_i(x) = u(x) = \alpha x$, it is possible to obtain the optimal value of ρ for Procassini criterion using Equation (4). If τ is the optimal load balancing interval when $u(x)$ is a linear equation, and the load balancing is perfect, the optimal value ρ_τ is

$$\rho_\tau = \frac{T_{\text{withLB}}(\tau) + C}{T_{\text{withoutLB}}(\tau)} = \frac{\mu(\tau) + C}{\mu(\tau) + u(\tau)}. \tag{21}$$

Therefore, Procassini criterion is equal to Menon criterion provided that $u_i(x) = u(x) = \alpha x$ and that ρ_τ is employed. More generally, this indicates that for each load imbalance function $u(\cdot)$ there exists an optimal ρ value. Unfortunately, as $u(\cdot)$ is in general unpredictable, computing ρ_τ seems highly challenging in practice.

Generalization for any $u(t)$. It is now possible to reformulate this result without assuming any particular form of $u(t)$. Starting from Equation (14), which now reads

$$T_{\text{par}} = \sum_{i=0}^{n-1} \Big(\int_0^{\tau} u(x)\mathrm{d}x + C \Big) + \int_0^\gamma \mu(t)\,\mathrm{d}t. \tag{22}$$

We obtain the optimal value of τ using the same methodology, which is solving and isolating τ in

$$\frac{\partial T_{\text{par}}}{\partial \tau} = -\frac{\gamma}{\tau^2}\Big(\int_0^\tau u(x)\mathrm{d}x + C \Big) + \frac{\gamma}{\tau}\alpha\tau = 0. \tag{23}$$

The solution of this equation is

$$\tau u(\tau) - \int_0^\tau u(x)\mathrm{d}x = C, \tag{24}$$

which leads to a new global load balancing criterion that does not make any assumption on the function $u(t)$ that describes the load balancing metric over the iterations. This result differs from the

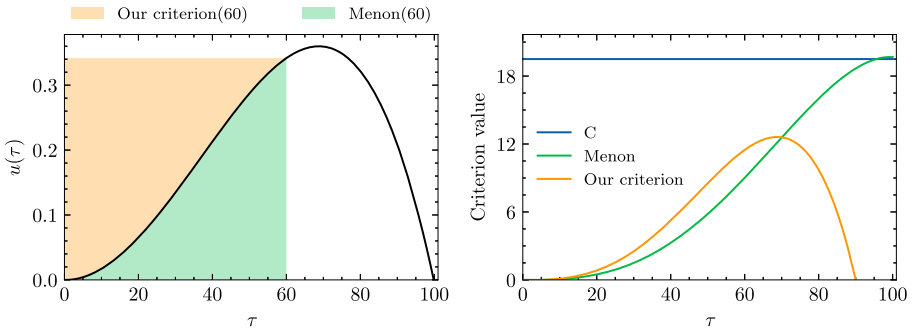


Fig. 1. Toy example illustrating the key difference between our criterion and Menon criterion. The left figure shows a load imbalance that correct itself after a hundred iterations. The colored area in the left figure shows the criterion value of both criteria at iteration 60. The right figure illustrates the evolution of the criterion value over the iterations. A key observation is that Menon criterion will apply the load balancing mechanism at iteration 96 even though it is not needed, whereas our criterion successfully detects this situation.

one presented by Menon et al. [19] because now a load balancing step is done when the area *above* the load imbalance curve equals the load balancing cost C. To illustrate this, we propose in Fig. 1 a toy example showing the main difference between our criterion and Menon criterion and a plot of their value over time. In this example, the load imbalance is ephemeral starting at iteration 0 and it grows until iteration 69, then, it decreases until it reaches $u(100) = 0$. In the figure placed on the right, we see that Menon criterion applies a load balancing at iteration 96 even though the load imbalance is almost completely corrected at this point. In contrast, we observe that our criterion is able to detect that such a situation does not need load balancing. Finally, the colored area in the left figure shows the criterion value of both criteria at iteration 60. We see that, unlike Menon criterion, our criterion corresponds to the area between the load imbalance curve and $u(\tau)$.

To have a better idea of the performance improvement we might gain by using this criterion, we propose, in Section 6, a comparative study of the criteria presented in Section 3 on synthetic benchmarks and real N-body simulations. In the next section, we present an efficient algorithm for finding the optimal load balancing scenario, which we will use to rank the load balancing criteria as a function of their relative performance compared to the optimum.

Remark 3. Following the development of our theoretical model, it clearly appears that the exact solution of this problem can only be obtained using an exhaustive search. Indeed, we observed that this problem is recursive as the load balancing time intervals s_i, s_{i+1} (i.e., the solution) are part of the input data. This seems to prevent us from finding an analytical solution.

5. Finding the optimal load balancing scenario

It is essential to know the performance of the optimal load balancing scenario to analyze the performance of the scenario produced by load balancing criteria. To find this optimum, we need an efficient way to look for the optimal scenario among all the possible ones. Unfortunately, the number of possible scenarios grows exponentially with the number of iterations to compute (i.e., γ). For that reason, it is impossible to use brute force algorithms even for a small number of iterations.

To overcome this problem, we can organize the scenarios in a tree to use efficient tree search algorithms. Indeed, the load balancing decision problem fits well in a binary tree because a decision (using or not the load balancing mechanism) must be made at each iteration. The vertices represent the state of the application (balanced or not). The edges e represent the process of going from an iteration to another (i.e., computing the iteration and applying, or not, the load balancing algorithm). The edge cost $\mathscr{C}(e)$ represents the CPU time for going from an iteration to the next. Fig. 2 shows how load balancing decisions are organized as a binary tree.

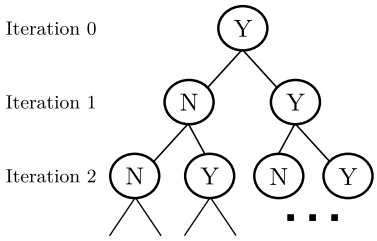


Fig. 2. The load balancing decision problem organized as a binary tree. "Y" (respectively "N") means that the application is balanced (respectively not balanced) at the given iteration. In other words, left edges apply load balancing while right edges do not.

A load balancing scenario is defined as a path from the root (iteration 0) to a leaf node (iteration γ). The cost of a path p from the root node to any subsequent node, $\mathscr{C}(p)$, is the sum of the edge costs that belong to the path, which reads

$$\mathscr{C}(p) = \sum_{e \in p} \mathscr{C}(e). \tag{25}$$

As mentioned in Equation (1), the optimal scenario is the one that minimizes the path cost among all scenarios, minimizing the application wall time.

5.1. Load balancing tree pruning

To reduce the tree size and the complexity of the search, we propose two steps: (i) to merge redundant load balancing nodes and (ii) to prune edges that belong to sub-optimal paths. We assume that the load balancing mechanism is independent of previous load balancing decisions in these two steps. This means that the workload of the processing units post load balancing does not depend on previous decisions but only on current information. Afterward, we apply the A* algorithm proposed by Hart et al. [11], in which we include these two optimizations, to find the optimal load balancing scenario. Note that the algorithm proposed herein has no practical uses in production, but is rather dedicated for analysis purposes because it requires some iterations to be executed multiple times.

Redundant nodes merging. As we saw previously in Section 4, regardless of past decisions, the edge cost $\mathscr{C}(e)$ for going from iteration i to the next is $C + \mu(i)$ if we performed a load balancing step. This is because the data partitioning after a load balancing call is independent of the previous decisions. This is illustrated in Fig. 3 which shows the processing elements' workload within the load

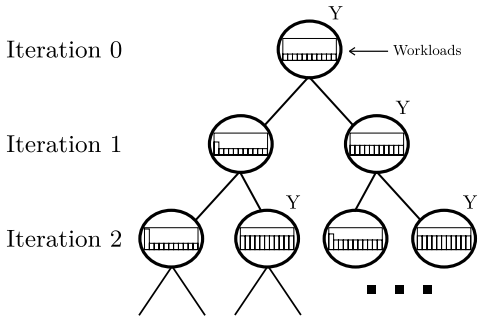


Fig. 3. The impact of load balancing on workloads within the load balancing tree before the merging process. The nodes that share the same data partitioning at the same iteration are redundant.

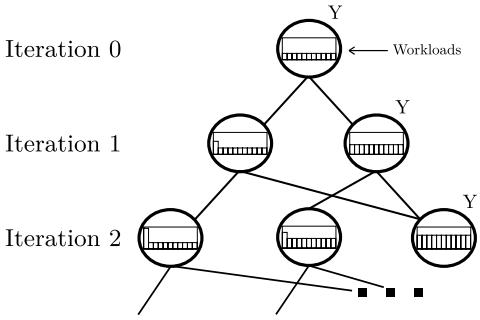


Fig. 4. The load balancing tree after the merging process, now, i paths lead to a unique load balancing node ("Y" node) at each iteration, where i is the node's iteration.

balancing tree. Therein, we see that there is no difference between two load balancing nodes (flattened workloads) at the same level of the tree. The only thing that distinguishes these nodes is their cumulative cost. Therefore, load balancing nodes (i.e., "Y" nodes) at the same iteration can be merged.

Sub-optimal path elimination. Merged nodes may have multiple paths leading to them, as illustrated in Fig. 4. The idea of sub-optimal path elimination is to find the shortest path from the root (iteration 0) to each load balancing node (merged node) and remove the other paths. Indeed, if a load balancing node y is part of the final solution, then the shortest path from the root node to y is also part of the solution. It is true if and only if the load balancing cost C is independent of previous decisions, which is an assumption that we think to be reasonable. Finally, let us assume that the load balancing node y is a merged node at iteration i, therefore, y has i paths leading to it. Then, the shortest path $p^*_{0\to y}$ is obtained by solving

$$p^*_{0\to y} = \underset{p^k_{0\to y}\ \forall k=1..i}{\operatorname{argmin}} \ \mathscr{C}(p^k_{0\to y}), \tag{26}$$

where $p^k_{0\to y}$ is the kth path reaching node y. In practice, only the last edge of each sub-optimal path is removed because the previous edges belong to other paths. Fig. 5 illustrates a possible resulting tree after the sub-optimal path elimination process.

Thanks to the pruning process, the size of the load balancing tree is drastically reduced. The number of vertices decreases from

$$V = 2^\gamma - 1 \tag{27}$$

to

$$V = \sum_{i=0}^{\gamma-1}(i+1) = \frac{\gamma(\gamma+1)}{2} \tag{28}$$

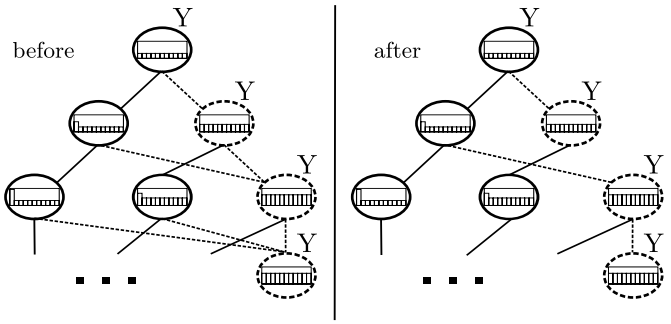


Fig. 5. Example of the sub-optimal path elimination process. A dashed edge, from a dashed node ("Y" node) to its parent, is removed if it does not belong to the shortest path from the root to the node. The total number of edges is reduced from exponential to linear in the number of iterations (i.e., the depth).

and the number of edges decreases from

$$E = 2^\gamma - 2 \tag{29}$$

to

$$E = V - 1. \tag{30}$$

These two optimizations that we include in the A* algorithm allow us to find the optimal load balancing scenario efficiently.

5.2. Optimal scenario search algorithm

The A^* algorithm [11] is a well-known path search algorithm. It aims at finding the path from a source node to a destination with the smallest cost. Besides, A^* is optimal and complete, which means that it will finish and it will find the solution if one exists. It is done by keeping a list of paths and extending them, one edge at a time until the destination is reached. At each iteration, A^* extends the path that minimizes the cost equation

$$f(n) = g(n) + h(n), \tag{31}$$

where n is a candidate node, $g(n)$ is the total cost to reach that node, and $h(n)$ is an optimistic estimation of the path cost from n to the destination node (i.e., the solution) [11]. In our case, we model $g(n)$ as the time taken by the application to reach a particular iteration given a load balancing scenario. $h(n)$ represents the computation time from a particular iteration to the end of the application, given no load imbalance. This mathematically reads

$$h(n) = \sum_{j=i}^{\gamma} \mu(j), \tag{32}$$

where i corresponds to the node's iteration (i.e., depth), γ is the total number of iterations, and $\mu(\cdot)$ refers to the average time per iteration. After a path has been extracted from the queue, its children are generated and inserted into the list of paths. In practice, the whole algorithm is managed by a priority queue where paths are inserted, sorted according to their cost $f(n)$, and at each iteration the path of least cost is extracted. This algorithm belongs to the category of branch-and-bound algorithms given the definition of Horowitz and Sahni [12] because no path is being extracted from the queue before all the children of the current path being expanded have been inserted in the queue.

To apply the two optimizations mentioned earlier, we customize two parts of the algorithm: (i) how new nodes are inserted in the queue (sub-optimal path elimination) and (ii) how the queue is kept sorted and clean from redundant nodes (redundant nodes merging). Algorithm 1 shows the pseudo-code for

finding the optimal load balancing scenario with our branch-and-bound algorithm. Our optimizations appear during the expansion of the current path (i.e., generation of its children and their insertion in the priority queue), hence, our modified A* algorithm still belongs to the branch-and-bound category given the definition of Horowitz and Sahni [12].

As discussed earlier, load balancing nodes at the same depth (i.e., iteration) are redundant. Therefore, instead of inserting load balancing nodes directly in the queue, we check if one already exists at the same iteration and replace it if the new node has a lower cumulative edge cost (line 10 in Algorithm 1 and detailed in Algorithm 2). However, according to the sub-optimal path elimination process, we must guarantee that we can not insert a load balancing node at a given depth if the shortest path has already been discovered at this level. To do this, we implemented a lookup table in which we map the iteration (i.e., depth) to a boolean. This boolean indicates if a load balancing node has already been removed. When a new node has to be inserted, we look up inside the table to see if one has already been seen and if it does, we discard it.

Even though the aforementioned optimizations allow to retrieve the optimal load balancing scenario, they may prune the nth best solution. Those solutions may be of interest to measure the gap between the optimal and close-to-optimal scenarios. Hence, to retrieve them as well, we need to prune fewer paths in the sub-optimal path elimination process. To recall, we previously explained that, in this process, we keep the last edge from a parent of a load balancing node only if it belongs to the shortest path from the root node to the load balancing node. This constraint has to be relaxed to allow the computation of the nth best solution. The idea is to keep the last edge from a parent of a load balancing node whether they belong at least to its nth shortest path. It has a logical meaning; in fact, if we keep all the possible edges, we end up with the original algorithm, which is able to retrieve all the solutions ordered by their cumulative edge cost. Note that the time to the solution will increase because the size of the tree increases as well.

Finally, the last point to discuss is how to find the optimal load balancing scenario in a real application when $\mathscr{C}(e)$ is measured on a real computer and not by an equation. In this setup, the idea remains the same as before. However, when a node produces its children, we compute the two edge costs by executing the corresponding iterations. Indeed, the partition and the state of the application (e.g., the position of the particles in space, their velocities, etc.) must be propagated and updated after each computation. To reduce the memory footprint, we propose to use a lookup table to store the application states as a function of their iteration. Moreover, this is necessary to guarantee that every node at the same iteration has the same application state, which is needed for results consistency.

We made available an implementation of the optimal load balancing scenario algorithm in C++ with two different packages:

- LBOPT [34]: This package includes the customized A^* algorithm and the model presented in Section 3. LBOPT can be used to have a first idea of the performance of various load balancing criteria that can be modeled using equations.
- YALBB [35]: It implements an N-body simulation with a short-range force. YALBB eases the benchmarking of load balancing algorithms and criteria by separating the physics from the code of interest. It employs template meta-programming and an extensive use of modern C++ constructs.

6. Comparison of load balancing criteria

This section proposes a comparison study of four load balancing criteria present in the literature; moreover, we discuss the pros and cons of these criteria. For that purpose, we have two approaches. First, we used synthetic benchmarks that we modeled using the equations presented in Section 4. Therein, we compared only global load balancing criteria, and for Menon's like criteria, we implemented only the original Menon criterion. For instance, we did not consider the criterion from Marquez et al. [4] because it involves the local workload from the processing elements. These synthetic benchmarks target various types of workload increase rates that create load imbalance over time. They are meant to cover as many real-life situations as possible. We studied the following schemes where the load imbalance

- Follows a linear growth, a logarithmic growth, and a quadratic growth.
- Auto-corrects itself periodically.

We used YALBB to assess the efficiency of load balancing criteria on a real-world problem. We then compared their performance against the optimum obtained using the algorithm presented in Section 5. We employed several particle distributions and behaviors to match as closely as possible our synthetic benchmarks.

6.1. Synthetic benchmarks

Two main pieces of information describe a parallel application. First, the total workload associated with the problem itself $W(t)$ (i.e., the time to compute the application on one processing unit). In the case of inherently irregular applications, this workload may change over time. Second, the distribution of the total workload among the processing elements (i.e., load imbalance) is used for the computation $I(t)$. From those two pieces of information, we compute $m(t)$ and $\mu(t)$, which we use in Equation (10), to compute the application parallel time. To recall, Equation (10) reads

$$T_{\text{par}} = \int_0^\gamma m(t)\,\mathrm{d}t = \int_0^\gamma m(t) - \mu(t)\,\mathrm{d}t + \int_0^\gamma \mu(t)\,\mathrm{d}t. \tag{33}$$

Using $W(t)$, we can retrieve the average workload $\mu(t)$ given a number of processing elements, whereas, we can compute $m(t)$ using the load imbalance $I(t)$ and $\mu(t)$ using the well-known percent imbalance metric [24]

$$I(t) = \frac{m(t)}{\mu(t)} - 1,$$
$$m(t) = [I(t) - 1]\mu(t).$$

For that purpose, we have to define the function $W(t)$ and $I(t)$ and how they behave over time.

The first function, $W(t)$, gives at each iteration the total amount of work to do (expressed in units of time). It reads

$$W(t) = W_0 + \sum_{i=1}^{t} \omega(i), \tag{34}$$

where W_0 is the initial application workload and $\omega(t)$ is a function giving the difference of application workload between two iterations. Hence, the average workload $\mu(t)$ is expressed as $\mu(t) = W(t)/P$. The second function, $I(t)$, gives at each iteration the load imbalance, hence it is expressed as

Table 2
Parameters used to define the synthetic benchmarks. Two types of situations have been considered. The first one (top side of the table) considers benchmarks with a static workload and irregular workload distribution. In contrast, the second one (bottom side of the table) targets a benchmark with an irregular workload and an irregular workload distribution. All workloads are expressed in time units.

$\omega(t)$	$\iota(t - \text{LB}_{\text{previous}})$	W_0	P	C	γ
0	0.1	$52 * P$	10,649,600	$W_0 * P * 10^2$	600
0	$1/(0.4 * t + 1)$	$52 * P$	10,649,600	$W_0 * P * 10^2$	600
0	$0.02 * t$	$52 * P$	10,649,600	$W_0 * P * 10^2$	600
0	$-(0.1 * (t\%17)) + 0.8$	$52 * P$	10,649,600	$W_0 * P * 10^2$	600
$\sin \frac{\pi t}{180}$	0.1	$52 * P$	10,649,600	$W_0 * P * 10^2$	600
$\sin \frac{\pi t}{180}$	$1/(0.4 * t + 1)$	$52 * P$	10,649,600	$W_0 * P * 10^2$	600
$\sin \frac{\pi t}{180}$	$0.02 * t$	$52 * P$	10,649,600	$W_0 * P * 10^2$	600
$\sin \frac{\pi t}{180}$	$-(0.1 * (t\%17)) + 0.8$	$52 * P$	10,649,600	$W_0 * P * 10^2$	600

$$I(t) = \begin{cases} I(t-1) + \iota(t - \text{LB}_{\text{previous}}) & \text{if t} > \text{LB}_{\text{previous}}, \\ 0 \quad \text{otherwise}, \end{cases} \tag{35}$$

where $\text{LB}_{\text{previous}}$ is the previous iteration at which the load balancing mechanism has been used and $\iota(t)$ is a function returning the difference in load imbalance between two iterations. Typically, $I(t) \in [0, P-1]$, hence, in practice, a particular attention as to be given to not trespass those bounds. Now, by setting $\omega(\cdot)$, $\iota(\cdot)$, W_0, P, and the number of iterations γ, we can compute $m(t)$ and $\mu(t)$ for each iteration and use Equation (10) to compute the parallel time of the application.

Remark 4. Here, we make a strong assumption on the shape of the load imbalance curve after load balancing. Indeed, in practice, the load balancing mechanism involving data partitioning will influence the load imbalance growth. It is impossible to know the load imbalance function after load balancing without a deep understanding about the partitioning algorithm impact on the problem to solve, which is extremely challenging to incorporate into a mathematical model. Herein, we decide to use $\iota(t - \text{LB}_{\text{previous}})$, which means that each time a load balancing is performed, the load imbalance pattern is repeated. Another possibility could have been to set the y-intercept to 0 (i.e., shift the function $\iota(\cdot)$ down) after each load balancing step. However, this solution was difficult to implement without providing any clear benefits. Finally, we think that this subject is worth a research effort and will be targeted in future works.

The parameters used in the synthetic benchmarks are summarized in Table 2. The initial workload (expressed in time units) is proportional to a 2D Lattice-Boltzmann computational fluid dynamic problem with 10^9 D2Q9 cells per processing unit with a performance of 1 Gflops [31]. The number of processing units is equal to the number of cores available in the supercomputer "Sunway TaihuLight" [32]. We studied two types of situations. First, we targeted benchmarks with a static workload (i.e., the global workload is always the same) but with a workload distribution that changes over time. Then, we focused on the same benchmarks but with an irregular workload that increases/decreases over time. The static workload benchmarks target applications that suffer from load imbalance due to the parallelization. The irregular workload benchmarks target applications with varying workload per time-step, and where the load imbalance comes from both the problem in itself and the parallelization.

Finally, we use our C++ implementation of our branch-and-bound algorithm presented in Section 5 (LBOPT), in which we employ $m(t)$ and $\mu(t)$ (derived from $W(t)$ and $I(t)$) to compute the parallel time to reach any node in the tree and find the optimal load balancing scenario σ^*.

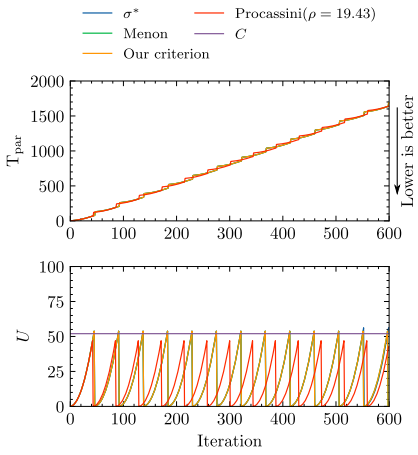
The results of the synthetic benchmarks for static applications are shown in Fig. 6. For Procassini criterion, we tried 5000 values of ρ between 0.5 and 50.0; however, for readability, we decided only to show the scenario that performed the best. The upper figure shows the simulated parallel time that we obtained using the model presented in Section 4. The lower figure indicates the growth of the cumulative time-per-time-step U (defined in Equation (20)), and the horizontal bar gives the value of C in order to track how the criteria differ from Menon criterion given by Equation (19). We use this figure to compare the behavior of the load balancing criteria.

In the constant experiment (Fig. 6a), both Menon criterion and our criterion behave like the optimal strategy. In other words, their load balancing time interval is similar to σ^*. Still, they differ marginally at the end of the simulation. It is important to remark that depending on when the last load balancing step happens, and it may be preferable to delay or schedule some load balancing calls earlier, as we can observe in Fig. 6a. Indeed, wasting a call at the very end of a simulation is useless. However, to take such a decision, one may need to foresee the future and detect if, given the current criterion, a call would appear near the end. Obviously, only the solution from our branch-and-bound algorithm is able to see that, as it tests "all" the possible solutions. Procassini criterion with a ρ value of 19.43 seems optimal. Note that we also tried to use ρ_τ (defined in Equation (21)) for Procassini criterion. We observed that Procassini criterion performs the load balancing steps at the exact same iteration as Menon and our criterion, as suggested in Remark 2. Finally, this experiment fits well the hypothesis of both our criterion and Menon criterion, and thus they behave optimally, as shown in Section 4.

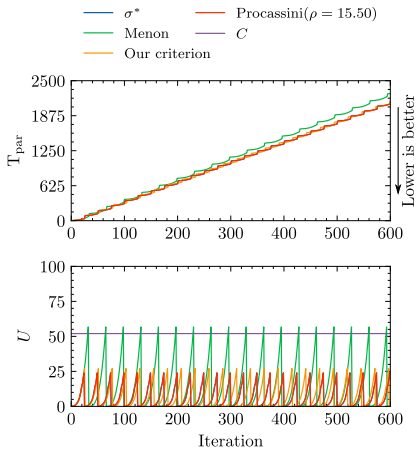
In the linear experiment (Fig. 6b), our criterion and Procassini criterion with a ρ value of 15.5 behave like the optimal scenario and therefore are very close in terms of performance. However, we notice that Menon criterion does not follow the same load balancing time interval as the optimal scenario, leading to a performance loss of approximately 10%. In particular, we remark that Menon criterion does not re-balance frequently enough.

In the sublinear experiment (Fig. 6c), the opposite situation appears (compared to the linear experiment). Herein, Menon criterion re-balance too often, wasting valuable resources. It is expected behavior as we observed in Section 4 that this criterion is optimal only if the load imbalance growth is constant, which is not the case in the sublinear experiment nor in the linear experiment.

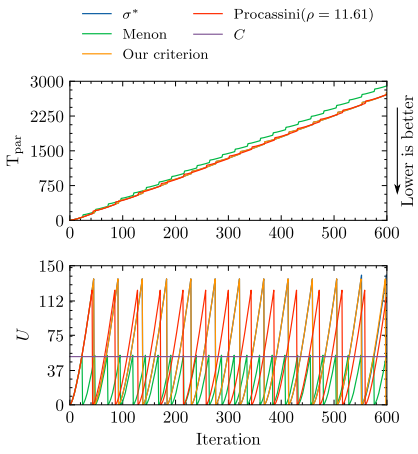
In the auto-correct experiment (Fig. 6d), we see that neither our criterion nor Menon criterion can understand that no load balancing is required because the load imbalance corrects itself pe-



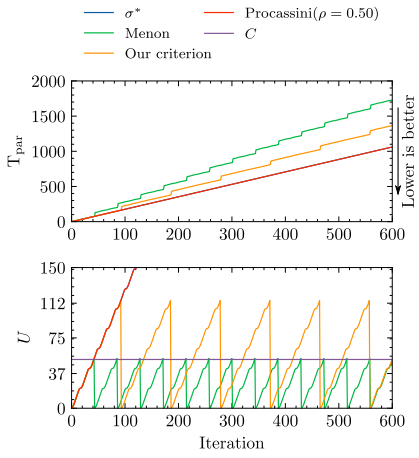
(a) The load imbalance increase rate is constant: $\iota(t) = 0.1$



(b) The load imbalance increase rate is linear with time: $\iota(t) = 0.02t$



(c) The load imbalance increase rate is sublinear with time: $\iota(t) = \frac{1}{0.4t+1}t$



(d) Load imbalance corrects itself every 17 iterations: $\iota(t) = -0.1 * (t\%17) + 0.8$

Fig. 6. Results of the synthetic benchmarks with static workloads for Menon criterion [19], our criterion, and Procassini criterion [27] against the optimal scenario σ^*. Four sources of load imbalance are considered: (i) constant, (ii) linear with time, (iii) sublinear with time, and (iv) linear with time and self-correcting every 17 iterations. Parameters used in this benchmark are summarized in Table 2. U is defined in Equation (20).

riodically. Nevertheless, our criterion is able to detect up to five auto-correcting patterns in a row. For that reason, in this experiment, our criterion is far better in terms of performance than Menon criterion. Only Procassini criterion provided the optimal ρ value is able to match the performance of the optimal scenario.

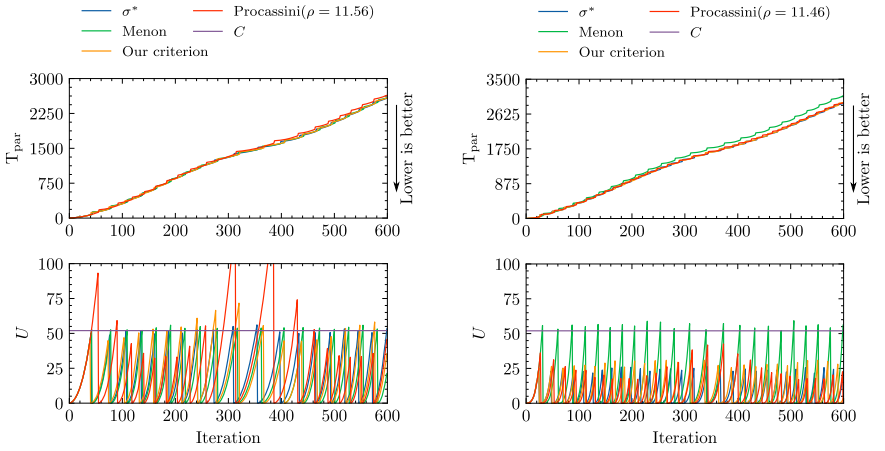
The results of the synthetic benchmarks with irregular workloads are presented in Fig. 7. Like in the previous benchmarks, the same values of ρ have been considered for Procassini criterion, and we show the scenario that performed the best.

In the constant experiment presented in Fig. 7a, we see that the performance of both our criterion and Menon criterion are almost unchanged. However, Procassini criterion decreased in performance compared to the static experiment. In the linear experiment (Fig. 7b), the results are similar to the static experiment where Menon criterion does not re-balance frequently enough, whereas our criterion and Procassini criterion follow the behavior of the optimal strategy.

In the sub-linear experiment (Fig. 7c), Menon criterion improves its performance, whereas Procassini criterion's performance decreases. Our criterion performs on par with both Menon criterion and the optimal scenario. It is worth noticing that during the slow-down around iteration 300, our criterion stops re-balancing while the optimal scenario only decreases the load balancing time interval. This suggests that our criterion is able to adapt its behavior to the current situation. This phenomenon is also visible in the last experiment. However, there, the optimal scenario does not re-balance at all. Finally, in the auto-correct experiment (Fig. 7d), we remark that Procassini criterion is the only criterion able to detect that re-balancing the application is not necessary. Nevertheless, our criterion reduces its load balancing time interval, drastically improving its performance compared to Menon criterion.

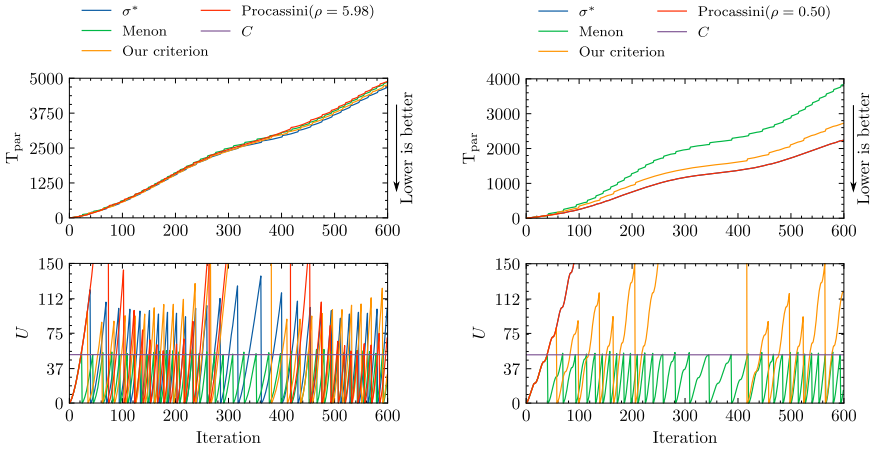
To understand the difference in performance among those criteria in a better way, we show in Fig. 8 the relative performance of our criterion, Menon criterion, and Procassini criterion compared to the optimal scenario. The relative performance is defined as $T_{\text{criteria}}/T_{\sigma^*}$. We see that out of the three criteria we studied Procassini criterion is the best provided the optimal value of ρ. However, not every scientist can afford the effort to find the optimal ρ before executing his/her application, which is not needed with our criterion and Menon's like criteria. Moreover, the performance of both Menon criterion and our criterion are really close to the optimal scenario in these experiments. Finally, Menon criterion performs better in the irregular workload than in the static workload situations.

To confirm these hypotheses, we propose a numerical study of all the criteria presented in Section 3 on YALBB, a home-made load



(a) The load imbalance increase rate is constant: $\iota(t) = 0.1$

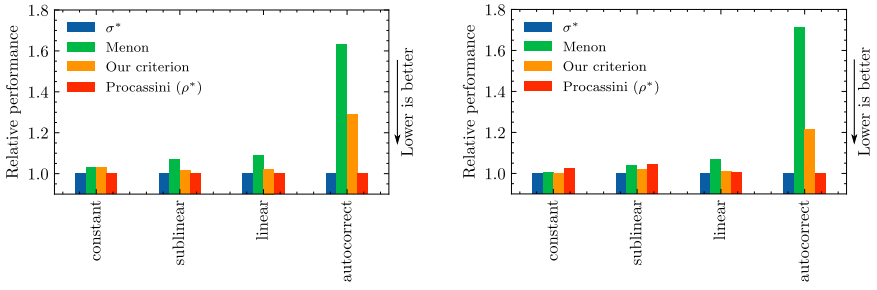
(b) The load imbalance increase rate is linear with time: $\iota(t) = 0.02t$



(c) The load imbalance increase rate is sublinear with time: $\iota(t) = \frac{1}{0.4t+1}t$

(d) Load imbalance corrects itself every 17 iterations: $\iota(t) = -0.1 * (t\%17) + 0.8$

Fig. 7. Results of the synthetic benchmarks with irregular workloads for Menon criterion [19], our criterion, and Procassini criterion [27] with an optimally tuned parameter ρ against the optimal scenario σ^*. Four sources of load imbalance are considered: (i) constant, (ii) linear with time, (iii) sublinear with time, and (iv) linear with time and self-correcting every 17 iterations. Parameters used in this benchmark are summarized in Table 2. U is defined in Equation (20).



(a) Static workload

(b) Irregular workload

Fig. 8. Relative performance of the our criterion, Menon criterion, and Procassini criterion against the optimal scenario in the static workload and irregular workload synthetic benchmarks. The relative performance is defined as $T_{\text{criteria}}/T_{\sigma^*}$.

balancing benchmark based on a N-body simulation with a short-range force.

6.2. Numerical study with YALBB

We carried out three experiments involving 40,000 particles and hundreds of millions of interactions with "YALBB" to evaluate the load balancing criteria presented in Section 3. The experiments were conducted with a standard Lennard-Jones interaction. The inner data structure uses the well-known cell lists algorithm for managing particles neighborhood. In these experiments, we used Zoltan [6] as a load balancing library for partitioning and managing the related data. Fig. 9 shows an example of 40,000 particles distributed among 4 processing elements using the Hilbert space-

Table 3
Physical parameters for the three numerical experiments.

Parameter	Contraction	Expansion	Expansion and Contraction
Box size (x,y,z)		(3.15, 3.15, 3.15)	
Number of particles		40,000	
σ_{LJ}		0.7	
ϵ_{LJ}		1.0	
Initial temperature		3.0	
Time-step	2*e*-5	8.4*e*-5	1.2*e*-4

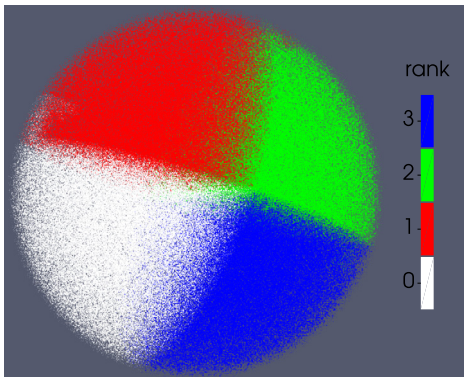


Fig. 9. Example of a sphere of uniformly distributed 40,000 particles. Particles are distributed to 4 processing elements using the Hilbert Space Filling Curve. The particles are colored according to the rank of their attributed processing element.

filling curve algorithm available in Zoltan. The experiments were executed on "Yggdrasil" the University of Geneva's cluster (Intel Xeon Gold 6240 CPU @ 2.60 GHz).

The first experiment consisted of a uniformly distributed sphere of particles expanding in a vacuum. The second experiment simulated the compression of a bigger uniformly distributed sphere of particles in a vacuum. The third experiment was a combination of both, one after the other, starting with the expansion phase. While in expansion, the particles were attracted to the sphere's center by a force proportional to the earth's gravity. Hence, after a few iterations, the sphere started to compress again. The results are obtained over one sequence of expansion-compression of the gas. The physical parameters used in our numerical study are shown in Table 3. The number of interactions to compute over time is shown in Fig. 10 for each experiment. As we can see in this figure, the amount of interactions (i.e., the density of particles) varies a lot over the execution of the experiment changing the requirement for load balancing. At the beginning of the expansion simulations, almost every particle interacts with all the others, this huge density decreases rapidly after the beginning of the simulation, drastically changing the workload of many processing elements. The reverse situation appears in the contraction simulation where there is almost no interaction at the beginning of the code execution, but a very high density is observed towards the end. In these experiments, we used the Hilbert Space Filling curve algorithm available in the Zoltan load balancing library [6].

The results of the three experiments are presented in Fig. 11. We executed the code 5 times for each experiment, and we report the median parallel time for each criterion. As we can observe, state-of-the-art load balancing criteria can achieve close to optimal performance. However, for Procassini criterion and Marquez criterion, the user has to find the optimal value of the parameter (ρ or ξ), which is not something everybody can afford. This is why automatic criteria seem to be the best fit for most situations, even though a ρ value between 1.0 and 1.25 seems to work the best for Procassini criterion. Furthermore, as we can see in Table 4, criteria with an extra parameter often have non-consistent results across experiments. Also, the penalty for using a sub-optimal value can be huge, and there is no rule of thumb to find the right value except testing many of them. Finally, we see that our criterion performs, on average, 6.79% faster than the studied load balancing criteria with a standard deviation of 0.08. In particular, our criterion is 12.47% faster than Zhai criterion in the expansion-contraction experiment and 6.95% faster than Menon criterion in the contraction experiment.

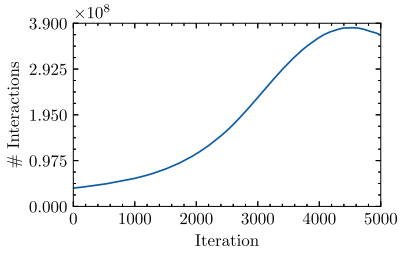
Among Menon's like criteria, the Zhai criterion seems to be the less stable one. Even though it outperforms Menon criterion in one experiment, the Zhai criterion produced a run considerably slower in the other two experiments. The reason for this result is likely to be due to the evaluation phase $\mathbb{P} = 100$ proposed by Zhai et al. in their paper [37] that may be well fitted for the contraction experiment and not in the others. However, the study of the impact of the evaluation phase on the performance of the Zhai criterion is out of the scope of this paper and could be the subject of another work. Finally, these results involving Menon's like criteria suggest that different implementation of the same idea behind load balancing criteria might significantly impact performance.

Finally, we observe that our criterion performs on par with Menon criterion and outperforms it in the expansion and contraction simulation. Menon criterion seems to perform better when the application exhibits an irregular workload, as seen in the synthetic benchmarks. It could be why the gap between the two criteria is much closer in the numerical experiment than in the synthetic benchmark. Overall, our criterion and Menon criterion seem to be the most stable criteria. The optimum is faster than Menon criterion by 36.80% and 32.09% faster than our criterion in the contraction experiment, 19.17% and 18.60% faster in the expansion experiment, and 16.33% and 18.03% faster in the expansion-contraction experiment.

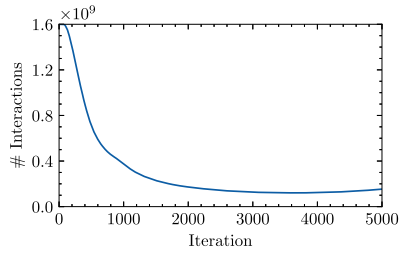
The present study is not enough to conclude that our criterion is better than Menon criterion, even though our criterion outperforms Menon criterion up to 6.9%. In comparison, it was slower by only at most 2.0%. However, it suggests that they are both excellent alternatives. In particular, our numerical study indicates that these two criteria often perform almost optimally. Therefore, we encourage scientists to use our branch-and-bound algorithm to compare the performance of available load balancing criteria to assess which criterion is the most suited for their type of problem.

7. Conclusion

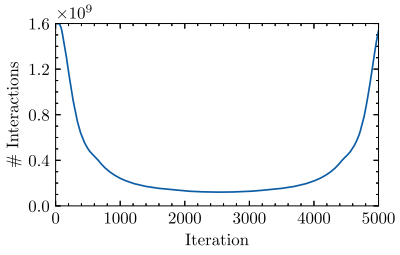
In the present paper, we proposed a review of state-of-the-art load balancing criteria and we introduced a novel fully automatic criterion based on a simple mathematical model inspired from the literature. We tried to classify these criteria as a function of their requirements and the information (external or not) required to compute the load balancing decision. Secondly, we proposed a branch-and-bound method for computing the set of load balanc-



(a) The sphere of particles is contracting, the number of interactions grows.



(b) The sphere of particles is expanding, the number of interactions decreases.



(c) The sphere of particles is expanding and then contracting, the number of interactions decreases and then increases.

Fig. 10. The number of particle interactions to compute at each iteration (i.e., application workload) of the three experiments carried out in the numerical study. Each experiment is composed of 40,000 particles. The first experiment computes a sphere of uniformly distributed particles that contracts on the effect of an external force. The second experiment computes a sphere of uniformly distributed particles that expands. The third experiment starts by expanding the sphere and then the sphere contracts.

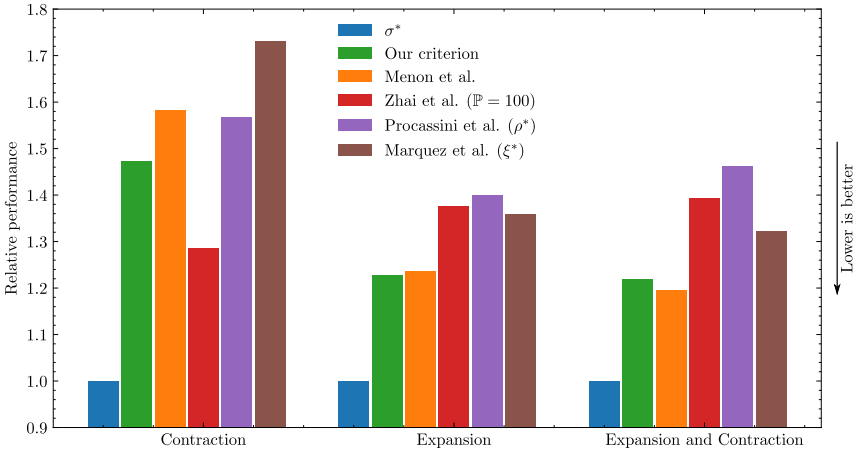


Fig. 11. Comparison of the median performance of each criterion (among 5 executions) in the numerical experiments relative to the optimal scenario σ^*.

ing steps leading to the optimal performance of a given application. Besides, we provide two implementations of this algorithm. The first implementation in LBOPT [35] the package related to the synthetic benchmarks and the second, in YALLB [34] the package related to the N-body solver we used in the numerical experiments. Afterward, we studied the performance of state-of-the-art load balancing criteria and our new criterion on synthetic benchmarks (modeled via our simple mathematical model) and on a parallel N-body solver.

We observed that our novel criterion outperforms automatic state-of-the-art criteria in synthetic benchmarks. However, we pointed out that the performance difference was tighter in the irregular total workload scheme compared to the static total workload scheme. We also identified that modeling the impact of the load balancing method on the load imbalance growth is challenging. This is a topic that is worth the research effort and will be targeted for future work.

We saw that the gain of our criterion with respect to the other criteria is smaller in our N-body numerical experiments due to system perturbations and uncertainties about the load imbalance function. However, we remarked that fully automatic criteria have more reliable results, only at most 36.80% (Menon criterion) slower than the optimal scenario. In particular, a run with our criterion is never more than 32.08% slower than the optimum. Our criterion can outperform Menon criterion by up to 6.9%, while it is outperformed by up to a marginal 2.0% in the worst case. We also noticed that our criterion is, on average, 6.79% faster than the other load balancing criteria with. All these experiments suggest that our criterion is a very good alternative to other automatic load balancing criteria, offering almost optimal performance.

Of course, to further confirm the aforementioned observations, we plan to test our new criterion on production codes. The first step will be to integrate our re-balancing strategy in Palabos [15], a parallel Lattice-Boltzmann solver. Then, we will investigate more

Table 4
Summary of the performance results for the three numerical experiments. The median performance and the median absolute deviation of each criterion are reported from the data gathered during 5 executions. For the criteria with an extra parameter, we reported the performance of the best parameter.

Experiment	Criterion	Median [s]	Median Absolute Deviation
Contraction	σ^*	19.35	0.08
	Menon et al.	30.62	0.18
	Our criterion	28.49	0.42
	Zhai et al. ($\mathbb{P} = 100$)	24.89	0.25
	Procassini et al. ($\rho^* = 1.25$)	30.34	1.64
	Marquez et al. ($\xi^* = 4.00$)	33.51	0.37
Expansion	σ^*	19.77	0.10
	Menon et al.	24.46	0.56
	Our criterion	24.29	0.36
	Zhai et al. ($\mathbb{P} = 100$)	27.20	0.57
	Procassini et al. ($\rho^* = 1.00$)	27.66	0.73
	Marquez et al. ($\xi^* = 0.90$)	26.88	0.49
Expansion and Contraction	σ^*	24.68	0.12
	Menon et al.	29.50	0.49
	Our criterion	30.11	0.30
	Zhai et al. ($\mathbb{P} = 100$)	34.42	0.93
	Procassini et al. ($\rho^* = 1.00$)	36.09	2.66
	Marquez et al. ($\xi = 1.50$)	32.63	1.32

complex load imbalance growth. For instance, we plan to add random bias to the load imbalance growth to simulate perturbations coming from various sources, such as system characteristics. The last step is to improve our understanding about the impact of the partitioning method on the load imbalance growth. It is mandatory to have benchmarks that better reproduce the behavior of real applications.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgment

We want to thank Olivier Belli for his help in proofreading the paper.

Appendix A. Optimal scenario finding algorithm

Algorithm 1: Optimal Load Balancing Scenario Searching Algorithm.

```
   Input: numIter: the number of iterations to compute, priorityQueue: a
          priority queue
 1 foundLb[i] = 0∀i = 1..K;
   // root node
 2 cNode = Node(iter=0, LB=true, cost=0.0, appState, lbState, prev=∅);
 3 while cNode.iter < numIter do
 4 |   if cNode.LB then
 5 |   |   foundLB[cNode.iter] = true;
 6 |   end
 7 |   dontLBNode, doLBNode = cNode.getChildren();
 8 |   if not foundLB[doLBNode.iter] then
   |   |   // Measurement of cost (i.e., time) with a
   |   |      theoretical model or a real application
 9 |   |   doLBNode.computeCost();
10 |   |   replaceOrInsertNode(priorityQueue, doLBNode);
11 |   end
12 |   dontLBNode.computeCost();
13 |   insert(priorityQueue, dontLBNode);
14 |   cNode = priorityQueue.pop();
15 end
```

Appendix B. Replace or insert algorithm

Algorithm 2: replaceOrInsertNode(priorityQueue, doLBNode): void.

```
   Input: priorityQueue: the priority queue, doLBNode: the load balancing
          node to insert or replace
 1 for node ∈ priorityQueue do
 2 |   if node.iter == doLBNode.iter and node.LB == true then
 3 |   |   if node.cost > doLBNode.cost then
 4 |   |   |   priorityQueue.remove(node);
 5 |   |   |   priorityQueue.insert(doLBNode);
 6 |   |   end
 7 |   |   return;
 8 |   end
 9 end
10 priorityQueue.insert(doLBNode);
```

References

[1] R. Borrell, G. Oyarzun, D. Dosimont, G. Houzeaux, Parallel SFC-based mesh partitioning and load balancing, in: Proceedings of ScalA 2019: 10th Workshop on Latest Advances in Scalable Algorithms for Large-Scale Systems - Held in Conjunction with SC 2019: The International Conference for High Performance Computing, Networking, Storage and Analysis, Institute of Electrical and Electronics Engineers Inc., 2019, pp. 72–78.

[2] A. Boulmier, I. Banicescu, F. Ciorba, N. Abdennadher, An autonomic approach for the selection of robust dynamic loop scheduling techniques, in: Proceedings - 2017 IEEE 16th International Symposium on Parallel and Distributed Computing, ISPDC 2017, 2017.

[3] A. Boulmier, F. Raynaud, N. Abdennadher, B. Chopard, On the benefits of anticipating load imbalance for performance optimization of parallel applications, in: Proceedings - IEEE International Conference on Cluster Computing, ICCC, vol. 2019-September, Institute of Electrical and Electronics Engineers Inc., 2019.

[4] Márquez Claudio, Eduardo César, Joan Sorribes, A load balancing schema for agent-based spmd applications, in: Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA), 2013.

[5] L. DeRose, B. Homer, D. Johnson, Detecting application load imbalance on high end massively parallel systems, in: Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), in: LNCS, vol. 4641, Springer Verlag, 2007, pp. 150–159.

[6] K. Devine, E. Boman, R. Heaphy, B. Hendrickson, C. Vaughan, Zoltan data management services for parallel dynamic applications, Comput. Sci. Eng. 4 (2) (2002) 90–97.

[7] J.-L. Fattebert, D.F. Richards, J.N. Glosli, Dynamic load balancing algorithm for molecular dynamics based on Voronoi cells domain decompositions, Comput.

Phys. Commun. 183 (2012) 2608–2615, https://doi.org/10.1016/j.cpc.2012.07.013.
[8] F. Fleissner, P. Eberhard, Parallel load-balanced simulation for short-range interaction particle methods with hierarchical particle grouping based on orthogonal recursive bisection, Int. J. Numer. Methods Eng. 74 (4) (2008) 531–553, https://doi.org/10.1002/nme.2184.
[9] M. Furuichi, D. Nishiura, Iterative load-balancing method with multigrid level relaxation for particle simulation with short-range interactions, Comput. Phys. Commun. 219 (2017) 135–148, https://doi.org/10.1016/j.cpc.2017.05.015.
[10] M.R. Garey, D.S. Johnson, Computers and Intractability: A Guide to the Theory of NP-Completeness, W.H. Freeman & Co., New York, USA, 1979.
[11] P.E. Hart, N.J. Nilsson, B. Raphael, A formal basis for the heuristic determination of minimum cost paths, IEEE Trans. Syst. Sci. Cybern. 4 (2) (1968) 100–107.
[12] E. Horowitz, S. Sahni, Fundamentals of Computer Algorithms, Computer Software Engineering Series, Pitman, 1978, https://books.google.ch/books?id=n8c8PgAACAAJ.
[13] T. Ishiyama, K. Nitadori, J. Makino, 4.45 Pflops astrophysical N-body simulation on K computer - the gravitational trillion-body problem, in: International Conference for High Performance Computing, Networking, Storage and Analysis, SC, 2012.
[14] G. Karypis, V. Kumar, A fast and high quality multilevel scheme for partitioning irregular graphs, SIAM J. Sci. Comput. 20 (1) (1998) 359–392, https://doi.org/10.1137/S1064827595287997.
[15] J. Latt, O. Malaspinas, D. Kontaxakis, A. Parmigiani, D. Lagrava, F. Brogi, M.B. Belgacem, Y. Thorimbert, S. Leclaire, S. Li, F. Marson, J. Lemus, C. Kotsalos, R. Conradin, C. Coreixas, R. Petkantchin, F. Raynaud, J. Beny, B. Chopard, Palabos: parallel lattice Boltzmann solver, Comput. Math. Appl. 81 (2021) 334–350, https://doi.org/10.1016/j.camwa.2020.03.022.
[16] M. Lieber, W.E. Nagel, Highly scalable SFC-based dynamic load balancing and its application to atmospheric modeling, Future Gener. Comput. Syst. 82 (2018) 575–590, https://doi.org/10.1016/j.future.2017.04.042.
[17] M. Mayr, A. Popp, Dynamic load balancing for large-scale mortar contact formulations, PAMM 20 (1) (2021) e202000196, https://doi.org/10.1002/PAMM.202000196, https://onlinelibrary.wiley.com/doi/full/10.1002/pamm.202000196.
[18] H. Menon, Adaptive Load Balancing for HPC Applications, Ph.D. thesis, University of Illinois Urbana-Champaign, 2016.
[19] H. Menon, N. Jain, G. Zheng, L. Kalé, Automated load balancing invocation based on application characteristics, in: 2012 IEEE International Conference on Cluster Computing, 2012, pp. 373–381.
[20] K.G. Miller, R.P. Lee, A. Tableman, A. Helm, R.A. Fonseca, V.K. Decyk, W.B. Mori, Dynamic load balancing with enhanced shared-memory parallelism for particle-in-cell codes, Comput. Phys. Commun. 259 (2021) 107633, https://doi.org/10.1016/J.CPC.2020.107633.
[21] A. Mohammed, F.M. Ciorba, Sil: an approach for adjusting applications to heterogeneous systems under perturbations, in: Euro-Par 2018: Parallel Processing Workshops, Springer International Publishing, Cham, 2019, pp. 456–468.
[22] A. Navarro Muñoz, A.F. Lorenzon, E. Ayguadé Parra, V. Beltran Querol, Combining dynamic concurrency throttling with voltage and frequency scaling on task-based programming models, in: 50th International Conference on Parallel Processing, ICPP 2021, Association for Computing Machinery, New York, NY, USA, 2021.
[23] P. Offenhäuser, Load-balance strategies for CFD-codes on HPC systems, in: Proceedings of the 7th GACM Colloquium on Computational Mechanics for Young Scientists from Academia and Industry, OPUS, Stuttgart, Germany, 2017.
[24] O. Pearce, T. Gamblin, B.R. de Supinski, M. Schulz, N.M. Amato, Quantifying the effectiveness of load balance algorithms, in: Proceedings of the 26th ACM International Conference on Supercomputing - ICS '12, ACM Press, New York, New York, USA, 2012, p. 185.
[25] O.T. Pearce, M.L. Adams, B.R. De Supinski, L. Rauchwerger, V.E. Taylor, Load Balancing Scientific Applications, Ph.D. thesis, Texas A&M University, 2014.
[26] R. Prat, T. Carrard, L. Soulard, O. Durand, R. Namyst, L. Colombet, AMR-based molecular dynamics for non-uniform, highly dynamic particle simulations, Comput. Phys. Commun. 253 (2020) 107177, https://doi.org/10.1016/J.CPC.2020.107177.
[27] R.J. Procassini, M.J. O'brien, J.M. Taylor, Load Balancing of Parallel Monte Carlo Transport Calculations, Tech. Rep., International Topical Meeting on Mathematics and Computation, Supercomputing, Reactor physics and Nuclear and Biological Applications, Avignon, France, 2004.
[28] F.A. Rodrigues, Study of load distribution measures for high-performance applications, Ph.D. thesis, Federal University of Rio Grande do Sul, 2016, https://lume.ufrgs.br/handle/10183/149593.
[29] J. Schwarzrock, M.G. Jordan, G. Korol, C.C. d. Oliveira, A.F. Lorenzon, M. Beck Rutzig, A.C.S. Beck, Dynamic concurrency throttling on numa systems and data migration impacts, Des. Autom. Embed. Syst. 25 (2) (2021) 135–160.
[30] H.D. Simon, S.H. Teng, How good is recursive bisection?, SIAM J. Sci. Comput. 18 (5) (1997) 1436–1445, https://doi.org/10.1137/S1064827593255135.
[31] T. Tomczak, R.G. Szafran, Sparse geometries handling in lattice Boltzmann method implementation for graphic processors, IEEE Trans. Parallel Distrib. Syst. 29 (8) (8 2018), https://doi.org/10.1109/TPDS.2018.2810237.
[32] Top500, https://www.top500.org/lists/top500/2020/11/, November 2020.
[33] R. Van Driessche, D. Roose, An improved spectral bisection algorithm and its application to dynamic load balancing, Parallel Comput. 21 (1) (1995) 29–48, https://doi.org/10.1016/0167-8191(94)00059-J.
[34] xetqL/LBOPT: Lightning fast code for computing load balancing scenario from application parameters, https://github.com/xetqL/LBOPT.
[35] xetqL/yalbb: Yet another load balancing benchmark, https://github.com/xetqL/yalbb.
[36] K. Zhai, T. Banerjee, D. Zwick, J. Hackl, S. Ranka, Dynamic load balancing for compressible multiphase turbulence, in: Proceedings of the 2018 International Conference on Supercomputing - ICS '18, ACM Press, New York, New York, USA, 2018, pp. 318–327.
[37] K. Zhai, T. Banerjee, D. Zwick, J. Hackl, R. Koneru, S. Ranka, Dynamic load balancing for a mesh-based scientific application, Concurr. Comput., Pract. Exp. 32 (9) (2020) e5626, https://doi.org/10.1002/CPE.5626, https://onlinelibrary.wiley.com/doi/full/10.1002/cpe.5626.

Dr. Anthony Boulmier received his B.A.Sc and his M.Sc from the University of Applied Sciences Western Switzerland (HES-SO). He received his Ph.D. in Computer Science from the University of Geneva in 2022. His work focuses on the performance optimization of parallel applications through load balancing.

Prof. Nabil Abdennadher received the Diploma in Engineering (Computer science) from ENSI, Tunisia, and the Ph.D. degrees in Computer Science from University of Valenciennes (France) in 1988 and 1991, respectively. He was an assistant professor at the University of Tunis II from 1992 to 1998 and a research assistant at EPFL from 1999 to 2000. In 2001, he joined the Depart. of Computer Engineering at the University of Applied Sciences, Western Switzerland (HES-SO, HEPIA) as an assistant HES professor. In 2008, he became an associate HES professor and in 2017 he was promoted to full HES professor. He is currently head of both the inIT Research Institute and the LSDS research group. His major research interests include high performance and distributed computing, Internet of Things and urban computing. He is representative of the Swiss Alliance for Data-Intensive Services in Swiss Romande, and member of the Editorial Board of the Journal of Reliable Intelligent Environments.

Prof. Bastien Chopard is full professor at the University of Geneva, and group leader in the Swiss Institute of Bioinformatics. He earned his PhD in theoretical physics from the University of Geneva in 1988. He then spent two years as a postdoc in the laboratory for computer science at MIT (Cambridge, USA), and one year in the Research Center, Juelich (Germany) before joining the computer science department at University of Geneva. His main research interests are the modeling and simulation of complex systems. He is internationally recognized for his work on Cellular Automata and Lattice Boltzmann methods. He wrote more than 200 scientific articles, presenting interdisciplinary research in various fields, such as physics, social and environmental science, bio-medical applications, numerical and optimization methods, parallel computing and multiscale modeling.