Future Generation Computer Systems 127 (2022) 208–224

Contents lists available at ScienceDirect

Future Generation Computer Systems

journal homepage: www.elsevier.com/locate/fgcs

Exploiting multi-level parallel metaheuristics and heterogeneous computing to boost phylogenetics

Sergio Santander-Jiménez [a,b,*], Miguel A. Vega-Rodríguez [a], Leonel Sousa [b]

[a] Department of Computer and Communications Technologies, University of Extremadura, Escuela Politécnica, Campus Universitario s/n, Cáceres 10003, Spain
[b] Instituto de Engenharia de Sistemas e Computadores - Investigação e Desenvolvimento em Lisboa (INESC-ID), Rua Alves Redol 9, Lisboa 1000-029, Portugal

ARTICLE INFO

Article history:
Received 16 October 2020
Received in revised form 12 August 2021
Accepted 4 September 2021
Available online 15 September 2021

Keywords:
Heterogeneous computing
Multi-level parallelism
Evolutionary computation
High performance computing
Bioinformatics

ABSTRACT

Optimization problems are becoming increasingly difficult challenges as a result of the definition of more realistic formulations and the availability of larger input data. Fortunately, the computing capabilities of state-of-the-art heterogeneous systems represent an opportunity to deal with the main complexity factors of these problems. These platforms open the door to the definition of robust metaheuristic solvers, in which parallel computations of different nature can be efficiently mapped to the most suitable architectures and hardware resources. This work investigates the combination of multi-level parallelism and heterogeneous computing to address an important multiobjective problem in bioinformatics: phylogenetics. A parallel metaheuristic approach, based on the joint exploitation of parallel tasks at the algorithm, iteration, and solution levels, is proposed to tackle computationally intensive inferences on CPU+GPU systems. Different heterogeneous design alternatives are also discussed, in accordance with the way the interactions between CPU and GPU are handled. The experimental evaluation of the proposal on real-world biological datasets points out the benefits of using multi-level, heterogeneous strategies, reporting accelerations up to 396× over the baseline metaheuristic as well as significant energy savings with regard to other parallel approaches, without impacting multiobjective solution quality.

© 2021 The Author(s). Published by Elsevier B.V. This is an open access article under the CC BY-NC-ND license (http://creativecommons.org/licenses/by-nc-nd/4.0/).

1. Introduction

Multiple application domains in science are characterized by the need to address optimization problems efficiently. However, solving these problems represents a remarkably challenging task, since their complexity is progressively growing [1]. Different complexity factors, including hard-to-tackle search spaces, sensitivity to problem parameters and sizes, multiple objective functions, high dimensionality, and large input data, represent fundamental issues that impact the execution of optimization algorithms. As a consequence, traditional algorithmic approaches do not longer satisfy the time constraints required to efficiently solve real-world optimization problems.

In order to overcome these issues, research on parallel metaheuristics has played a prevailing role in the last years. The exploitation of the parallelism available in these search algorithms represents a crucial strategy to satisfy execution constraints and achieve high-quality solutions in reasonable time [2]. Particularly, metaheuristics exhibit different, heterogeneous parallelization levels that can be classified according to the following hierarchy [3]:

- Algorithm level: this parallelization level is aimed at allowing the concurrent execution of different instances or components of the metaheuristic through different processes. It usually involves the parallel evolution of multiple structured populations in a collaborative or non-collaborative way;
- Iteration level: this level is focused on parallelizing the tasks that compose a generation of the metaheuristic, in such a way that the evolution of solutions in a particular population is handled in parallel. Problem-independent intra-algorithm strategies are employed to execute evolutionary operators, local searches, etc., using multiple threads;
- Solution level: this level is oriented towards the parallelization of operations treating a single solution (i.e. the calculation of objective functions and constraints verification). Problem-dependent intra-algorithm techniques are herein

* Corresponding author at: Department of Computer and Communications Technologies, University of Extremadura, Escuela Politécnica, Campus Universitario s/n, Cáceres 10003, Spain.
E-mail addresses: sesaji@unex.es (S. Santander-Jiménez), mavega@unex.es (M.A. Vega-Rodríguez), las@inesc-id.pt (L. Sousa).

https://doi.org/10.1016/j.future.2021.09.011
0167-739X/© 2021 The Author(s). Published by Elsevier B.V. This is an open access article under the CC BY-NC-ND license (http://creativecommons.org/licenses/by-nc-nd/4.0/).

applied to take advantage of the data parallelism opportunities exhibited in the problem formulation.

Highly complex optimization problems demand the combination of all the above mentioned parallelization levels to attain satisfactory results. However, each level has specific computing requirements [4,5], thus making their joint deployment on single-device, homogeneous environments not a suitable solution. The heterogeneous computing capabilities offered by state-of-the-art parallel systems represent a promising approach to undertake the design of multi-level parallel metaheuristics, as parallel tasks of different nature can be assigned to different devices according to their characteristics. In this sense, a fundamental question that must be addressed is how to effectively map each level to the underlying hardware resources, since an inappropriate mapping could lead to a poor exploitation of the system, idle resources, increased energy consumption, and other parallel performance issues. This research work is aimed at dealing with these issues, defining and discussing strategies to exploit multi-level parallelism and orchestrate heterogeneous resources for high-performance and energy-aware metaheuristic-based optimization.

This paper investigates the definition of a multi-level and heterogeneous parallel solver to tackle an important bioinformatics problem, phylogeny reconstruction [6]. The proposal is based on the Multiobjective Shuffled Frog-Leaping Optimization Algorithm (MO-SFLA) [7], a bioinspired multiobjective metaheuristic with intrinsic parallelism opportunities. The algorithmic design of MO-SFLA contains the elements required to implement the three parallel levels, thus representing a suitable framework to build the method.

The proposed approach introduces multi-level strategies relying on MPI [8] at the algorithm level, OpenMP [9] at the iteration level, and CUDA [10] at the solution level to allow efficient phylogenetic searches on CPU+GPU heterogeneous systems. An extensive experimental evaluation on five real-world datasets, representing different problem sizes, has been conducted to assess the performance of the proposal from three different perspectives: parallel performance, multiobjective quality, and energy consumption. Moreover, the relevance of the proposed strategies will be analysed through comparisons with state-of-the-art parallel phylogenetic tools.

The main contributions of this work can be listed as follows:

1. Proposal of a fully multi-level, parallel metaheuristic based on MO-SFLA that exploits the potential of CPU+GPU systems to boost optimization tasks through the joint combination of algorithm, iteration, and solution-level parallelism techniques;
2. Definition and discussion of different design strategies and orchestration schemes to implement accurate interactions between the heterogeneous layers involved in the proposal;
3. Identification and analysis of different configurations and variants of the proposal, aimed at promoting different parallel and multiobjective performance achievement goals;
4. In-depth evaluation and discussion of the proposed method and the defined design alternatives, attending to parallel performance, multiobjective results, and energy consumption.

This paper is organized as follows. The next section highlights recent trends on the development of parallel metaheuristics for heterogeneous systems, also identifying research efforts in the phylogenetics field. The formulation of the tackled problem and the associated complexity factors are presented in Section 3. Section 4 summarizes the baseline algorithm and presents the multi-level and heterogeneous strategies investigated in this research work. Section 5 introduces the experimental methodology and evaluates, from different points of view, the results obtained by the proposal. Finally, Section 6 provides conclusions and defines future research directions.

2. Related work

This section provides insight into noteworthy research works that investigated the relationship between optimization problem solvers and modern parallel computing platforms. Significant approaches specifically tailored to the bioinformatics problem herein tackled are also highlighted.

In recent years, the availability of an increasingly larger number of parallel architectures has brought new opportunities to accelerate the solution of NP-hard optimization problems. [4] presented a comparison of different parallel architectures (multicore CPUs, GPUs, clusters, and grid setups) when executing evolutionary algorithms under the island and global parallelization models. A significant conclusion derived from this work lies in the fact that not every platform is equally suitable to parallelize the different workloads of an evolutionary algorithm, thus supporting the need for heterogeneous computing in this context. [11] examined the master–worker parallelization of Particle Swarm Optimization (PSO) for hyper-parameter selection in deep neural networks running on multi-GPU systems. The results obtained for an experimental setup composed of up to six Tesla K80 GPUs denote significant speedups for large population sizes as well as high classification accuracy.

[12] showed that the use of NVIDIA Jetson TK1 embedded platforms represents a promising approach to accelerate the Non-dominated Sorting Genetic Algorithm II (NSGA-II) when large numbers of objectives and population sizes are considered. Moreover, [13] presented an implementation of genetic algorithms in ARM CPUs that relies on remote FPGA boards to conduct the calculation of fitness functions. In fact, the fine-grained parallelization of fitness calculations was found to be a significant approach to deal with the complexity of bioinformatics functions, as shown in [14] for the case of gene expression data and gene selection for cancer classification. More recently, hybrid MPI+SIMD implementations of evolutionary algorithms for SW26010 processors were investigated in [15], reporting speedups up to 3603x on the Sunway TaihuLight supercomputing system. The intersection of parallel computing, metaheuristics, and heterogeneous systems has also seen practical applicability in recent years to boost research on Tensor decomposition [16], big data management and processing [17], deep learning architectures [18], 0–1 knapsack problems [19], and evolutionary biology [20].

Along with these developments, significant research efforts have been oriented towards the exploitation of different parallelism levels. [21] introduced a two-level approach in which multicore resources are employed to parallelize a genetic algorithm for forest fire propagation prediction at the iteration and solution levels. The proposed method was successfully used to address high-workload scenarios in reasonable time. In [22], algorithm and solution-level parallel strategies were applied to boost the performance of population-based metaheuristics (namely genetic algorithms and scatter searches) for docking simulation on CPU+GPU clusters. Later on, [23] addressed the parallelization of the Teaching–Learning-Based Optimization (TLBO) algorithm, using hybrid MPI+OpenMP schemes aimed at exploiting parallelism at the algorithm and iteration levels. In [24], multi-criteria job shop scheduling problems were successfully tackled by defining a dual heterogeneous parallel genetic algorithm. The proposal combines island models with fine-grain parallelism to take advantage of both multicore CPU and GPU resources.

On the other side, [25] proposed a multi-level implementation of differential evolution for learning classification rules. In this proposal, the algorithm, iteration, and solution parallel levels are adapted to run entirely on the GPU by introducing different strategies to improve memory coalescence and throughput. However, other works have demonstrated that relying the execution of multi-level parallelism on just a single device does not guarantee optimal performance. For instance, [5] has shown that the iteration level is more suitable to be run on multicore CPUs, especially when standard population sizes are employed. The idea of going a step forward in the mapping of metaheuristic parallelization levels and the need for collaborative CPU+GPU frameworks is also supported in [26]. Our work is aimed at shedding further light on this point, verifying how the design of a fully multi-level method, accurately orchestrated in a heterogeneous environment, can effectively boost the solution of a complex real-world problem.

The literature also gives account of the recurrent linkage between parallelism and phylogenetic search algorithms. Several key works on the evaluation of different hardware platforms for this problem can be highlighted. [27] reported parallel implementations of the phylogenetic parsimony function on FPGA and CPU architectures. GPU-accelerated phylogenetic distance computations were examined in [28] by devising a parallel approach that optimized data structures and memory transactions. Network-on-chip-based accelerators were investigated in [29] to address the breakpoint phylogeny problem. [30] comparatively analysed CPU-based vectorization schemes and GPU kernels to speed up the phylogenetic likelihood function. Following this research line, [31] undertook the definition of vertical data partitioning schemes to improve likelihood computations on GPUs. More recently, [32] reported new parallel strategies included in the 3.0 release of the BEAGLE library, an API for parallel likelihood computations supporting a wide range of hardware accelerators. Additionally, [33] studied the parallelization of the well-known Fitch's algorithm on different generations of NVIDIA GPUs.

These previous works were mainly focused on accelerating the computations associated to the solution processing level, without exploring in detail other potential sources of parallelism. However, the combination of different parallelization levels has also found applicability in phylogenetics. State-of-the-art single-criterion methods rely on mixed-mode parallel approaches mostly oriented towards CPU architectures, as in the case of RAxML [34] and IQ-TREE [35]. These two tools implement homogeneous parallel schemes that combine MPI with OpenMP (IQ-TREE) or POSIX threads (RAxML) to distribute separate tree searches and parallelize tree likelihood computations on CPU-based systems, with SIMD support. Differently, MrBayes [36] provides a heterogeneous parallel approach that integrates MPI at the CPU side with the CUDA implementation of BEAGLE to calculate tree scores on GPU devices. Therefore, these single-criterion methods rely on the combination of job-level and solution-level parallelism to accelerate computations.

In the case of multiobjective approaches, developments have been focused on the definition of two-level parallel designs through MPI+OpenMP schemes, as reported in [20]. It can be highlighted the cluster-oriented approach implemented in PhyloMOEA, a multiobjective genetic algorithm [37]. This method undertakes, at each generation of the algorithm, the parallel processing of separate solutions in the population using MPI. This first parallel level is complemented by an additional OpenMP layer that distributes likelihood site calculations across execution threads. Consequently, the parallel scheme of PhyloMOEA exploits the iteration and solution parallelization levels on CPU-based cluster infrastructures.

Since the state-of-the-art tools implement mainly two parallelization levels, the accurate exploitation of all the three theoretical levels still represents an open research question. Moreover, most of the previously mentioned methods rely on CPU-only strategies or do not explore alternative orchestration schemes to maximize the exploitation of heterogeneous resources. The herein presented research tries to fill this gap, examining how the problem can benefit from the definition of a fully multi-level parallel metaheuristic (integrating the three key parallelization levels) that accurately exploits the combined capabilities of heterogeneous CPU+GPU platforms.

3. Problem formulation

Phylogenetic inference deals with the definition of the evolutionary events that explain the diversity observed in the molecular characteristics of current organisms or genes [6]. These evolutionary hypotheses are illustrated as tree-shaped structures $T = (V, E)$, commonly designated as phylogenetic trees. The node set V in T is composed of: (1) internal nodes describing hypothetical ancestral organisms, and (2) leaf nodes representing the extant organisms under study. The phylogenetic topology is therefore defined by the node-to-node relationships specified in the branch set E. The input data employed in these biological analyses is given by a multiple sequence alignment of size $N \times M$, where N refers to the number of organisms under consideration and M the length of their molecular sequences. For DNA sequences, the values that each sequence site can assume are defined by the nucleotide state alphabet $\Lambda = \{A, C, G, T\}$, which also includes special characters (gaps '-', unknown '?') and combinations of nucleotides.

Under these conditions, a phylogenetic search algorithm must undertake the processing of a phylogenetic tree decision space with the aim of identifying the evolutionary hypotheses that optimize one or several biological quality criteria. Following current research directions on addressing incongruence issues [37–39], a multiobjective formulation of the problem based on the two most popular phylogenetic optimality criteria, parsimony $P(T)$ and likelihood $L(T)$, is herein adopted:

$$\begin{aligned} &\text{optimize } \vec{f}(T) = \{f_1(T), f_2(T)\}, \\ &\quad \text{where } f_1(T) = \text{minimize } P(T), \\ &\qquad\quad f_2(T) = \text{maximize } L(T). \end{aligned} \tag{1}$$

The first objective function considered in this formulation, parsimony $P(T)$, quantifies the amount of evolutionary change observed in a phylogenetic tree. Under this criterion, priority must be given to the phylogenies that minimize the $P(T)$ score, which is given by the following expression:

$$P(T) = \sum_{i=1}^{M} \sum_{(u,v)\in E} C(u_i, v_i), \tag{2}$$

where $(u, v) \in E$ represents the branch between the nodes $u, v \in V$, $u_i, v_i \in \Lambda$ are the character states at the ith site for u and v, and $C(u_i, v_i)$ measures if a state change (substitution event) has taken place between u_i and v_i. In case a substitution event is observed in this evolutionary step, $C(u_i, v_i)$ will be 1. Otherwise, if both nodes show the same value at the ith site, $C(u_i, v_i)$ will be 0. $P(T)$ can be calculated following Fitch's algorithm [40], which specifies a bottom-up procedure to identify substitution events by defining ancestral character states throughout the topology. For an internal node u with children v, w, the ancestral state set at site i, $A_i(u)$, is calculated as:

$$A_i(u) = \begin{cases} A_i(v) \cap A_i(w) & \text{if } A_i(v) \cap A_i(w) \neq \emptyset, \\ A_i(v) \cup A_i(w) & \text{if } A_i(v) \cap A_i(w) = \emptyset. \end{cases} \tag{3}$$

Herein, an empty intersection ($A_i(v) \cap A_i(w) = \emptyset$) denotes a substitution event, since at least one of the child nodes did not inherit the state value of the ancestor u at the ith site. The definition of ancestral states can then be used to calculate $P(T)$ by identifying these potential substitutions.

The second objective function employed in this formulation, likelihood $L(T)$, measures the conditional probability of observing the input sequences, given a phylogenetic tree and a probabilistic model of nucleotide evolution. In this case, preference is given to the phylogenetic trees that maximize the $L(T)$ score, which can be calculated in the following way:

$$L(T) = \prod_{i=1}^{M} \sum_{x \in \Lambda} \pi_x L_p\,(r_i = x), \tag{4}$$

where π_x represents the stationary probability of the state $x \in \Lambda$, $r \in V$ the root node, and $L_p(r_i = x)$ is the partial likelihood of x being observed at the ith site of r. Felsenstein's algorithm [41] defines a procedure to calculate partial likelihood arrays by processing the topology in a recursive way. For an internal node u with children v, w, $L_p(u_i = x)$ can be defined as:

$$\begin{aligned} L_p(u_i = x) = &\left(\sum_{y \in \Lambda} P_{xy}\,(t_{uv})\, L_p\,(v_i = y) \right) \\ &\times \left(\sum_{y \in \Lambda} P_{xy}\,(t_{uw})\, L_p\,(w_i = y) \right), \end{aligned} \tag{5}$$

where t_{uv}, t_{uw} are the branch lengths, modelling evolutionary times, between u and v, w and $P_{xy}(t)$ is the transition probability of observing a substitution event from x to y within a time t. For a terminal node l, $L_p(l_i = x) = 1$ in case the character state l_i is equal to x. Otherwise, $L_p(l_i = x) = 0$. After calculating the partial likelihood arrays for each node, the $L(T)$ score derived from Eq. (4) is computed and reported in terms of log-likelihood values.

Phylogenetic reconstruction represents a difficult NP-hard problem [6], which shows two main complexity factors. First, the number of possible candidate solutions surpasses the Eddington number even for low-sized datasets. More specifically, the phylogenetic tree space grows exponentially with the number N of input sequences, following the double factorial $(2N-5)!!$. Second, the length M of the sequences and the number S of states in Λ have an impact in the calculations performed at the objective function level, thus contributing to the challenging time constraints of the problem. From scratch, the evaluation procedures used to calculate the parsimony and likelihood scores show complexities of $O(NM)$ and $O(NMS^2)$, respectively. The increasingly larger problem sizes involved in real-world phylogenetic analyses therefore demand innovative optimization strategies, which combine robust metaheuristics and the computing capabilities of state-of-the-art heterogeneous platforms to efficiently address these issues.

4. Multi-level parallel MO-SFLA for heterogeneous systems

In order to efficiently infer phylogenies on CPU+GPU systems, this work proposes a multi-level, heterogeneous approach based on the MO-SFLA metaheuristic. This section provides first a general overview of the baseline metaheuristic, proceeding afterwards with the description of multi-level parallelization strategies and the definition of heterogeneous design alternatives.

4.1. Baseline algorithm: MO-SFLA

MO-SFLA [7] is a population-based metaheuristic aimed at addressing complex optimization problems by combining search strategies from PSO and Shuffled Complex Evolution. The main idea lies in simultaneously exploring multiple directions of the search space by defining different partitions of individuals, designated as memeplexes, which are separately processed and refined through independent learning steps. These memeplexes are initialized and updated at each generation through a shuffling technique that allows the definition of a balanced distribution of n individuals per memeplex. Given a population P to be partitioned into m memeplexes, the best individual P_1 is assigned to the first memeplex Mem_1, the second best individual P_2 to the second memeplex Mem_2, P_m to Mem_m, P_{m+1} to Mem_1, etc. Parallel searches are then performed over each memeplex and, at the end of a generation, the updated memeplexes are merged in order to spread knowledge, thus enhancing the evolution of the whole population.

4.1.1. Application to phylogenetics

In order to tackle the phylogenetics problem with MO-SFLA, the representation of individuals in this algorithm is based on the equivalence between phylogenetic trees and matrix-shaped encodings. Particularly, the algorithm codifies a phylogeny as a symmetric distance matrix δ of size $N \times N$, where each entry $\delta[x, y]$ contains a floating-value measurement of the evolutionary distance between the organisms x and y. The phylogenetic tree associated to a distance matrix is obtained by using neighbour-joining techniques, such as the BIONJ tree-building method [6]. On the other side, the distance matrix for a given phylogenetic tree can be calculated as $\delta[x, y] = \sum_{u,v \in Path_{x,y}} t_{uv}$, where $Path_{x,y}$ refers to the set of nodes in the path between x and y, while t_{uv} represents the length of the branch (u, v). In this way, the search engine can switch from the tree decision space to the matrix space and vice versa according to the status of the optimization process.

Throughout the learning steps of the algorithm, new candidate solutions are generated by applying floating-point search operators over these phylogenetic representations. MO-SFLA can apply three different search strategies to boost the quality of the memeplexes. The first one employs information from the best local solution $Best_{local}$ in the currently processed memeplex Mem_i to generate a new candidate solution Sol_{new} as follows:

$$D_{xy} = rand() * (Best_{local}.\delta[x, y] - Mem_{ij}.\delta[x, y]), \tag{6}$$

$$Sol_{new}.\delta[x, y] = Mem_{ij}.\delta[x, y] + D_{xy}, \tag{7}$$

where Mem_{ij} refers to the jth individual in Mem_i and $rand()$ is a uniformly-distributed random number in the range [0,1]. The second search operator applies a similar learning strategy but taking into account the best global solution $Best_{global}$ in the population, replacing $Best_{local}$ with $Best_{global}$. In the third search strategy, a local search procedure based on the application of topological rearrangements (subtree pruning-regrafting and nearest neighbour interchange) and gradient-based branch length optimization is conducted. The selection of the search strategy to be used over Mem_{ij} depends on an associated counter variable which can take the values 0 (learning from $Best_{local}$), 1 (learning from $Best_{global}$), or 2 (local search). Each memeplex is refined through n_l learning steps, generating new solutions that are integrated into the memeplex in case an improvement in multiobjective quality is observed. Conversely, if Sol_{new} does not improve Mem_{ij}, Mem_{ij} will increase its counter variable to try a different search strategy in the next generation. Fig. 1 illustrates the learning process in MO-SFLA, while Fig. 2 represents an example of a Pareto front generated by the algorithm.

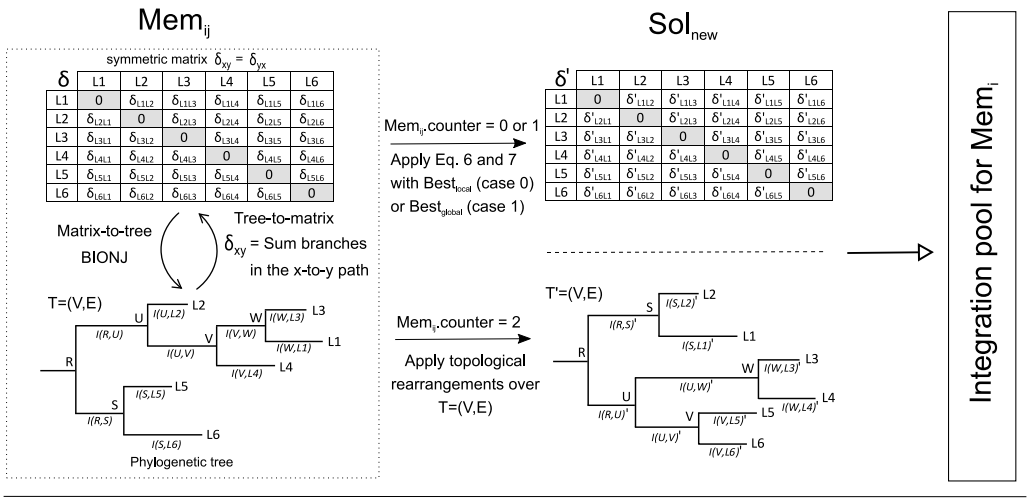


Fig. 1. Adaptation to the problem and learning procedure. Each memeplex Mem_i is processed separately, generating new solutions for n_l learning iterations. Phylogenetic information in an individual Mem_{ij} can be encoded by means of a distance matrix δ and a tree-shaped topology T, which are subject to different learning operators attending to Mem_{ij}.counter. The resulting solutions Sol_{new} are stored in an integration pool and incorporated to Mem_i in case of verifying improvements over the original individuals.

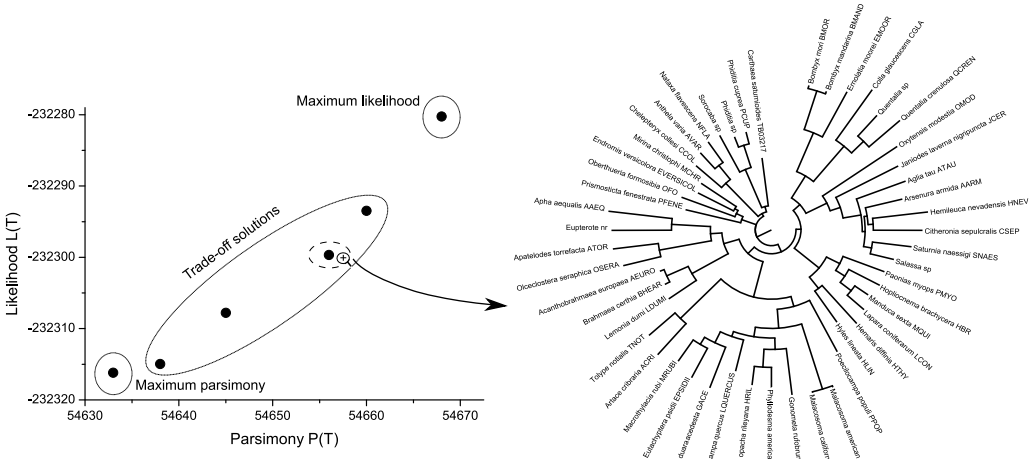


Fig. 2. Output of MO-SFLA: Pareto front and example of a multiobjective phylogenetic topology for a dataset of Bombycoidea gene data (denoted later as 50 × 18,321).

4.1.2. Handling multiobjective information

In this work, an adaptive version of MO-SFLA [42] is applied as a baseline for the proposed multi-level design. Two separate populations P^D, P^I, each one initialized with *popSize*/2 individuals, are employed to define two sets of memeplexes Mem^D and Mem^I. The main difference between P^D and P^I lies in the multiobjective strategy followed to evaluate the quality of individuals. While P^D follows Pareto dominance-based techniques (implemented through fast non-dominated sorts and crowding distance assignments [43]), P^I uses indicator-based fitness strategies (based on hypervolume [44]). In this way, the definition of the Mem^D and Mem^I sets in the shuffling steps is conducted in accordance with the multiobjective quality strategy considered in each population, as well as the identification of best local and global solutions.

The adaptive nature of the approach lies in the use of performance measurements *PerfD*, *PerfI* to update the number of memeplexes, m^D and m^I, assigned to each multiobjective strategy. *PerfD* and *PerfI* are computed by calculating the normalized distance between Sol_{new} and Mem^D_{ij} or Mem^I_{ij} for each objective in each learning step. In this way, it can be quantified the success of each multiobjective strategy, assigning more individuals to the best performing technique by dynamically updating m^D and m^I:

$$m'^D = Round\left(\frac{\sum_{i=1}^{m^D} PerfD_i}{\sum_{i=1}^{m^D} PerfD_i + \sum_{i=1}^{m^I} PerfI_i} * m\right), \tag{8}$$

$$m'^I = Round\left(\frac{\sum_{i=1}^{m^I} PerfI_i}{\sum_{i=1}^{m^D} PerfD_i + \sum_{i=1}^{m^I} PerfI_i} * m\right). \tag{9}$$

A control parameter *ac* is employed to specify the number of generations that must be considered, when capturing performance feedback, before the adaptive update of memeplexes is performed. In the last stages of a generation, the memeplexes assigned to each multiobjective strategy are merged into the associated population (P^D or P^I), thus allowing the evolution of the optimization process according to this adaptive guidance.

4.2. Multi-level parallel design

MO-SFLA represents a suitable algorithm to define a multi-level parallel approach due to its intrinsically parallel design. The following sources of algorithm, iteration, and solution-level parallelism can be identified:

Algorithm 1 Algorithm-level Parallelism in MO-SFLA: MPI Design

Input: int *maxEval* (maximum number of evaluations), int *popSize* (number of individuals in the population), int m (number of memeplexes), int n (number of individuals per memeplex = $popSize/m$), int n_l (number of learning steps per memeplex), int *ac* (adaptive adjustment control).
Output: *PF* (Pareto front).

```
1: Parallel API Initialization (MPI, OpenMP, CUDA, BEAGLE)
2: P^D, P^I ← Initialize Populations (popSize/2), PF ← 0
3: m^D, m^I ← m/2, m_proc ← m/ MPI_Comm_size
4: while ! stop criterion is reached (maxEval) do
5:    if MPI_Comm_rank = 0 then
6:       P^D ← Non-Dominated Sorting and Crowding (P^D, m^D * n)
7:       P^I ← Indicator-based Fitness Assignment (P^I, m^I * n)
8:       {Mem^D_1 ... Mem^D_{m^D}} ← Shuffling and Distribution (P^D, m^D, n)
9:       {Mem^I_1 ... Mem^I_{m^I}} ← Shuffling and Distribution (P^I, m^I, n)
10:      Mem ← Mem^D ∪ Mem^I
11:      /*Sending m_proc memeplexes to each MPI process i*/
12:      for x = m_proc * i to m_proc * i + m_proc do
13:         MPI_Send (Mem_x, n, i)
14:      end for
15:      MPI_Send (Best_global(P^D), 1, i), MPI_Send (Best_global(P^I), 1, i)
16:   else
17:      /*Receiving m_proc memeplexes from the master process*/
18:      for x = 1 to m_proc do
19:         MPI_Recv (Mem_x, n, 0)
20:      end for
21:      MPI_Recv (Best_global(P^D), 1, 0), MPI_Recv (Best_global(P^I), 1, 0)
22:   end if
23:   Pool ← Generate Candidate Solutions (Mem, m_proc, n_l, Best_global) /*Iteration level*/
24:   Pool ← Evaluate Candidate Solutions (Pool) /*Solution level*/
25:   Mem,PerfD,PerfI ← Integrate Solutions and Calculate MO-Performance (Pool)
26:   if MPI_Comm_rank = 0 then
27:      /*Receiving the m_proc updated memeplexes from each process i = MPI_ANY_SOURCE*/
28:      for x = m_proc * i to m_proc * i + m_proc do
29:         MPI_Recv (Mem_x, n, i)
30:      end for
31:      MPI_Recv (PerfD, 1, i), MPI_Recv (PerfI, 1, i)
32:      PerfD_mean, PerfI_mean ← Calculate Mean MO-Performance (PerfD, PerfI, ac)
33:      m^D, m^I ← Redistribute Memeplex Assignment (PerfD_mean, PerfI_mean, ac)
34:      P^D, P^I ← Update Populations (Mem, m^D, m^I)
35:      PF ← Update Pareto Front (PF, P^D, P^I)
36:   else
37:      /*Sending the updated m_proc memeplexes to the master process*/
38:      for x = 1 to m_proc do
39:         MPI_Send (Mem_x, n, 0)
40:      end for
41:      MPI_Send (PerfD, 1, 0), MPI_Send (PerfI, 1, 0)
42:   end if
43: end while
44: Parallel API Termination (MPI, OpenMP, CUDA, BEAGLE)
45: return PF
```

1. Algorithm level: the upper-level components of the metaheuristic, the memeplexes, represent independent subpopulations that can be evolved in parallel. The evolution of memeplexes can then be separately managed by assigning them to different MPI processes.
2. Iteration level: within a particular memeplex, the tasks that compose the learning loop handle the generation of new candidate solutions from one individual to another separately. These iterative tasks can then be independently processed by using OpenMP threads.
3. Solution level: the objective functions defined in the formulation of the problem, parsimony and likelihood, show data parallelism at the nucleotide processing level, which can be handled by exploiting the computing capabilities of GPU devices.

The detailed steps of the multi-level parallel MO-SFLA are shown in Algorithms 1 (for the algorithm level), 2 (iteration level), 3 (solution level — parsimony objective), and 4 (solution level — likelihood objective). These algorithms are described and explained in the next subsections.

4.2.1. Algorithm level

Algorithm 1 illustrates the main skeleton of the multi-level approach, putting emphasis on the algorithm parallelization level. Due to the stochastic components and the coarse granularity of the tasks performed at the algorithm and iteration level, multicore CPUs will be exploited to implement these first two parallelization layers. The initial steps involve the initialization of the parallel programming APIs and the data structures required by the algorithm (lines 1–3 in Algorithm 1). Each MPI process will be responsible for managing the evolution of m_{proc} memeplexes each generation, which is determined in accordance with the number of memeplexes (m) and the communicator group size `MPI_Comm_size`. Processes are organized following a master–worker hierarchy, where the master process (with `MPI_Comm_rank = 0`) undertakes the execution of multiobjective assessment strategies (Pareto dominance and indicator-based fitness) over P^D, P^I and the memeplex shuffling in the initial stages of the generation (lines 6–9). Afterwards, the distribution of the resulting memeplexes is conducted through message passing, employing the `MPI_Send` function at the master side (lines 12–15) and `MPI_Recv` at the worker side (lines 18–21). In this step, the workers also receive copies of the best global solutions in accordance with the multiobjective strategy associated to the assigned m_{proc} memeplexes.

Once the memeplexes have been distributed, each process carries out the learning steps over the assigned partitions, in which iteration-level parallelization strategies are applied to generate a solution pool (line 23), and the calculation of objective functions, with solution-level offloading to the GPU (line 24). The multiobjective performance measurements required by the adaptive design are also registered herein (line 25). It is worth noting that, during these steps, the master process assumes a worker role to avoid idle resources. The retrieval of the obtained results is then undertaken by communicating the updated memeplexes and performance values to the master through `MPI_Send` and `MPI_Recv` (lines 26–42), with `MPI_ANY_SOURCE` enabled at the receiver to soften synchronization constraints. With this information, the master will be able to update the configuration of memeplexes assigned to each multiobjective strategy and merge them in the corresponding population (lines 33 an 34). Finally, the Pareto front structure that contains the non-dominated solutions found by the algorithm is updated (line 35). The contents of this Pareto front are returned in the output of the algorithm once the stop criterion (a certain number of fitness evaluations) has been satisfied.

4.2.2. Iteration level

The iteration-level parallelization, performed within the assigned m_{proc} memeplexes, is described in Algorithm 2. The first task in this stage consists of the definition of index arrays to identify the best global and local solutions for each memeplex (lines 1–3 in Algorithm 2). Due to the fact that this is a simple indexing operation, it can be handled by a single execution thread by using the OpenMP directive `#pragma omp single`. For each memeplex, the generation of new solutions inside the learning loop (lines 6–12) is parallelized by using the worksharing directive `#pragma omp for`. The processing times for each iteration may vary due to the differences between the three learning strategies included in the MO-SFLA design, as well as the effect of topological complexity in the tree-building procedure. As a result, this loop can lead to potential load imbalance issues. In order to address this problem, a dynamic scheduling strategy is set in the `schedule` clause of `#pragma omp for`. Another strategy to enhance performance at this level is the use of core-thread affinity, which is enabled by using the `OMP_PROC_BIND` flag.

Alternatively, other schemes can also be applied to implement this parallelization level, e.g. the use of `#pragma omp for` on

Algorithm 2 Iteration-level Parallelism in MO-SFLA: OpenMP Design

Input: Individual* *Mem* (memeplexes assigned to the MPI process), int m_{proc} (number of memeplexes assigned), Individual $Best_{global}(P^D)$ (best global solution from P^D), Individual $Best_{global}(P^I)$ (best global solution from P^I), int n (number of individuals per memeplex), int n_l (number of learning steps per memeplex).
Output: *Pool* (new candidate solutions generated by the process).

```
1:  #pragma omp single
2:    BGlobalArray ← Index Best Global Solutions (Mem, Best_global(P^D), Best_global(P^I))
3:    BLocalArray ← Index Best Local Solutions (Mem)
4:  for x = 1 to m_proc do
5:    /*Generation of new candidate solutions via Equations 6–7 variants and local searches*/
6:    #pragma omp for schedule (dynamic)
7:    for j = 1 to n_l do
8:      switch (Mem_x(n−j).counter)
9:        case 0: Pool_x(n−j) ← Learn from Best Local (Mem_x(n−j), BLocalArray_x)
10:       case 1: Pool_x(n−j) ← Learn from Best Global (Mem_x(n−j), BGlobalArray_x)
11:       case 2: Pool_x(n−j) ← Apply Local Search (Mem_x(n−j))
12:   end for
13: end for
14: return Pool
```

top of line 4 combined with the `collapse` clause to join the memeplex and learning loops. Nevertheless, this alternative approach could impose some performance penalties, due to the use of collapsed-loop indices in the nested loop and the strided accesses that can arise when n_l does not match the size n of the memeplex.

4.2.3. Solution level

The solution level is focused on parallelizing fine-grained, problem-specific operations for a given candidate solution, which usually take place within the computation of objective functions. For the optimization problem herein tackled, a major source of data parallelism is found in the independent processing of sequence sites for both parsimony and likelihood (Eqs. (2) and (4)), thus allowing the concurrent computation of partial parsimony and likelihood operations for each site. Phylogenetic datasets now enclose a large amount of nucleotides per sequence, which makes suitable the execution of these objective functions in massively parallel accelerators like GPUs.

Algorithm 3 introduces the devised CUDA parsimony kernel. Prior to the instantiation of the kernel, two main data structures must be mapped to the GPU memory hierarchy. The first one is the input sequence alignment *sequences*, which is transferred once at the beginning of the application. In order to allow an efficient processing of character states, a char array of length $N \times M$ is employed to store the input sequences in row-major order. Furthermore, each character in the sequences is codified by using hexadecimal values (given in Table 1), which allow the computation of Fitch's ancestral states through bit-wise AND and OR operations. The second data structure is the phylogenetic topology to be evaluated, which is encoded by an internal node array denoted as *nodes*. Each element of this array contains the information required to process the topology in a bottom-up fashion, namely integer references to the associated child nodes (*id_children*) where the most significant bit is devoted to distinguish between internal (1) and leaf (0) positions.

In Algorithm 3, each GPU thread is responsible for calculating the partial parsimony score $P(T)_{thread\_id}$ at the *thread_id* site of the sequences. After initializing *thread_id*, $P(T)_{thread\_id}$, and the ancestral state set A_{thread_id} (lines 1 and 2 in Algorithm 3), each node in the topology array is processed to determine the corresponding ancestral state value *node_state*, which initially takes the value 0x1F (line 5). By checking the most significant bit of the child node reference, it can be read the child state from A_{thread_id} (in the case of internal nodes previously processed by the kernel) or from *sequences* (in the case of leaf nodes) (lines 8–12). After this step, Fitch's intersections and unions are performed between the current value of *node_state* and the child one (lines 14–15). $P(T)_{thread\_id}$ will be increased any time an empty intersection is detected, since it denotes the presence of a substitution event at the *thread_id* site. These operations are performed for each child, storing the resulting *node_state* value in A_{thread_id} (line 17). After the topology array has been entirely processed, shared-memory parallel reductions are applied over the $P(T)_{thread\_id}$ values in a thread block (lines 20–22). The resulting reductions are then arranged at the host side to compose the final $P(T)$ score.

Regarding likelihood, the proposed multi-level design implements this objective function by using BEAGLE [32], a high-performance library that provides an efficient, fine-grained parallel implementation of the likelihood calculations. More specifically, BEAGLE assigns the calculation of each entry in the partial likelihood arrays to a separate GPU thread, along with exploiting other sources of parallelism related to the computation of transition probabilities and the reduction of site likelihoods.

Algorithm 4 enumerates the steps required to calculate likelihood with BEAGLE. The first one involves the initialization of the BEAGLE library, performed at the beginning of the application, by using the function `beagleCreateInstance` (line 2 in Algorithm 4). Some essential parameters related to the computations (such as the number of sequences, number of partial likelihoods to be computed, character states in the alphabet, sequence lengths, number of transition probability matrices and rate categories, and

Table 1
Hexadecimal codification of the input sequences.

Nucleotide state	Hex value	Nucleotide state	Hex value
A (Adenine)	0 × 08	Y (C or T)	0 × 05
C (Cytosine)	0 × 04	K (G or T)	0 × 03
G (Guanine)	0 × 02	V (A or C or G)	0 × 0E
T (Thymine)	0 × 01	H (A or C or T)	0 × 0D
M (A or C)	0 × 0C	D (A or G or T)	0 × 0B
R (A or G)	0 × 0A	B (C or G or T)	0 × 07
W (A or T)	0 × 09	? (Unknown)	0 × 0F
S (C or G)	0 × 06	- (Gap)	0 × 10

Algorithm 3 Solution-level Parallelism in MO-SFLA: CUDA Parsimony

Input: `__constant__` NodePars* *nodes* (phylogenetic node array), `__constant__` int *num_inner* (number of internal nodes), `__global__` char* *sequences* (input nucleotide sequences), `__constant__` int *seq_length* (sequence length)
Output: `__global__` int* *Acc_P(T)* (accumulated partial parsimony scores)

```
1:  thread_id ← blockIdx.x * blockDim.x + threadIdx.x
2:  P(T)_thread_id ← 0, A_thread_id ← 0
3:  for i = 1 to num_inner do
4:    /*Initializing ancestral value for node i and reading states from its children*/
5:    node_state ← 0x1F
6:    for j = 1 to nodes[i].num_children do
7:      child_id ← nodes[i].id_children[j]
8:      if child_id is an internal node then
9:        child_state ← A_thread_id[child_id] /*read state from the ancestral set*/
10:     else
11:       child_state ← sequences[seq_length*(child_id)+thread_id] /*read state from the dataset*/
12:     end if
13:     /*Performing Fitch's set operations (Equation 3)*/
14:     fitch_op ← node_state & child_state
15:     P(T)_thread_id,node_state ← (fitch_op=0) ? {P(T)_thread_id+1, node_state | child_state} : {P(T)_thread_id, fitch_op}
16:   end for
17:   A_thread_id[i] ← node_state
18: end for
19: /*Performing parallel reduction of P(T)_thread_id values within a thread block*/
20: __shared__ TBlock_P(T) ← 0
21: TBlock_P(T)[threadIdx.x] ← P(T)_thread_id
22: Acc_P(T)[blockIdx.x] ← Shared Memory Parallel Reduction (TBlock_P(T))
23: return Acc_P(T) /*Final parallel reduction at the host side*/
```

Algorithm 4 Solution-level Parallelism in MO-SFLA: BEAGLE Likelihood

Input: char* *sequences* (input nucleotide sequences), NodeLike* *nodes* (phylogenetic node array), int *num_sequences* (number of sequences), int *num_partials* (number of partial likelihood arrays to be calculated), int *num_states* (number of state values), int *seq_length* (sequence length), *num_trMatrices* (number of transition matrices), *num_rCategories* (number of rate categories), *scale_buffers* (likelihood scaling buffers), *GPU_id* (GPU to be employed), *config* (model configuration file)
Output: int *Global_L(T)* (likelihood score)

```
1: /*Beagle Instance Initialization (only in the first generation)*/
2: Beagle ← Create BeagleLib Instance (num_sequences, num_partials,
   num_states, seq_length, num_trMatrices, num_rCategories, scale_buffers, GPU_id,
   BEAGLE_FLAG_FRAMEWORK_CUDA | BEAGLE_FLAG_PRECISION_DOUBLE |
   BEAGLE_FLAG_PROCESSOR_GPU)
3: Beagle ← Set Tip States (sequences)
4: Beagle ← Set Model Rate Matrix and State Frequencies (config)
5: Beagle ← Set Among Site Rate Heterogeneity (config)
6: /*Conducting likelihood computations*/
7: likeOps ← Define Likelihood Operations (nodes, Beagle)
8: tMatrices ← Update Transition Matrices (likeOps.Transitions, Beagle)
9: PartialLikelihoods ← Calculate Partials (likeOps.Partials, Beagle)
10: Global_L(T) ← Calculate Log-Likelihood (PartialLikelihoods, tMatrices, Beagle)
11: return Global_L(T)
```

likelihood scaling buffers), as well as the targeted GPU device and computation flags, are provided in the initialization procedure. Afterwards, the input sequence data (*sequences*) and the specifications of the sequence evolution model (contained in a BEAGLE configuration file *config*) must be provided to the library, accordingly transferring the structures to the GPU memory hierarchy in contiguous data patterns. `beagleSetTipStates` is used to provide the BEAGLE instance with the sequence data observed for each leaf node (line 3), while the character state frequencies and the instantaneous substitution rate matrix that define the model are set by using `beagleSetStateFrequencies` and `beagleSetEigenDecomposition` (line 4). In case the selected evolutionary model supports among-site rate heterogeneity ($+\Gamma$ models [6]), the `beagleSetCategoryRates` and `beagleSetCategoryWeights` functions are used to specify this characteristic (line 5).

Once initialized and configured the library, the calculation of the likelihood function for a topology array *nodes* is conducted as follows. First, it must be created from *nodes* the list of likelihood operations *likeOps* to be performed, namely those related to the computation of transition probabilities and partial likelihoods (line 7). In practical terms, this list of operations corresponds to different steps in Felsenstein's algorithm. Transition probabilities *tMatrices* are defined throughout the topology according to the parameters of the evolutionary model and the branch lengths of the targeted nodes, employing the `beagleUpdateTransitionMatrices` function for this purpose (line 8). After that, the operations list is processed to update the partial likelihood structures *PartialLikelihoods* by using `beagleUpdatePartials` (line 9). Once the transition matrices and the partial likelihoods are set, the library can proceed with the computation of the log-likelihood of the phylogeny through the `beagleCalculateEdgeLogLikelihoods` function (line 10). The resulting *Global_L(T)* value, which specifies the likelihood of the candidate solution, is then transferred from the GPU to the host side. Further details on the implementation of likelihood with BEAGLE can be found in [32,45].

4.3. Heterogeneous alternatives

Following the multi-level strategies previously introduced, different heterogeneous design alternatives can be devised to implement the interactions between the coarse-grained CPU-based levels and the fine-grained GPU-based layer. This work investigates two main heterogeneous schemes, which are graphically illustrated in Fig. 3.

In the first design variant (designated as MPI-to-CUDA, Fig. 3(a)), the MPI processes are responsible for conducting the instantiations of the solution-level, GPU evaluation procedures and the retrieval of objective function scores. Given *num_proc* processes, a total of *num_proc* CUDA streams and BEAGLE library instances will be used to offload parsimony and likelihood computations to the GPU. The key idea consists of transferring a large chunk of phylogenetic topologies to be processed in each instantiation of the kernels, thus minimizing the overhead introduced when initializing the communications between CPU and GPU. More specifically, the evaluation of the candidate solutions generated in the memeplex learning loop will be carried out once all the learning steps have been completed, such that $m_{proc} \times n_l$ solutions are offloaded per kernel instantiation. That is, the GPU operates at each generation over the complete *pool* of solutions generated by the *i*th MPI process, interacting through the *i*th CUDA stream and BEAGLE instance.

In the second alternative (designated as OpenMP-to-CUDA, Fig. 3(b)), the responsibility of interacting with the GPU is assumed by the OpenMP execution threads operating inside the memeplex learning loop. The main goal lies in computing the objective scores for a candidate solution immediately after generating it, so that the evaluation can take place without waiting for the termination of the whole iteration-level parallel loop. Therefore, each kernel instantiation deals with the evaluation of a single phylogenetic topology. In order to implement this idea, $num\_proc \times num\_threads$ CUDA streams and BEAGLE instances are required. Whenever the *j*th thread in a process generates a new candidate solution (via best local learning, best global learning, or local search), the resulting phylogenetic topology is communicated through the corresponding *j*th stream and the GPU proceeds with the parsimony and likelihood evaluation of the transferred solution.

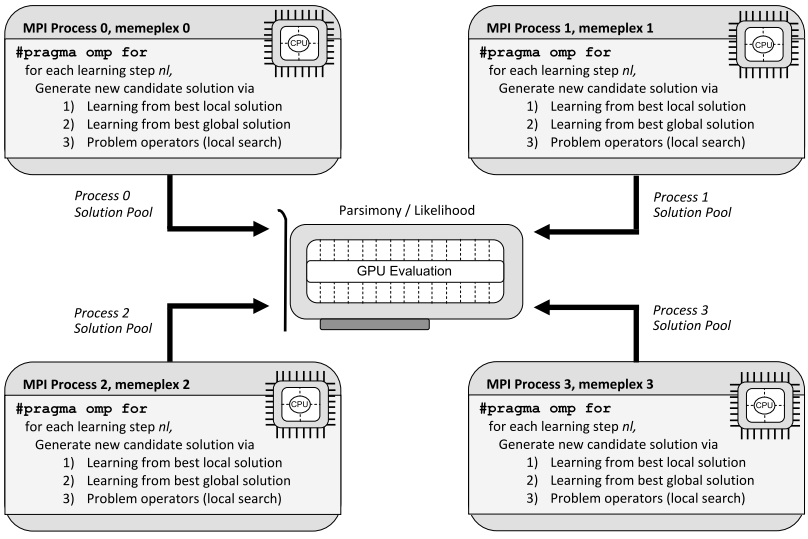
In both MPI-to-CUDA and OpenMP-to-CUDA design variants, the use of different CUDA streams/BEAGLE instances and asynchronous functions allow the achievement of concurrent kernel execution, according to the amount of resources available in the employed GPU device.

5. Experimental methodology and evaluation

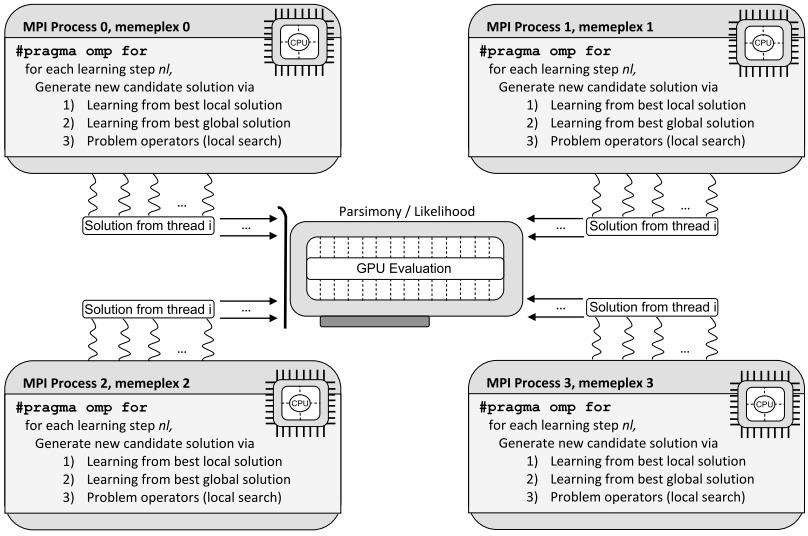
The experimental assessment of the proposed multi-level, heterogeneous approach is undertaken in this section. In a first step, different parametric configurations of the proposal will be analysed, in order to identify the most satisfying settings in terms of execution time and solution quality. Afterwards, the benefits of the multi-level parallel strategy will be examined by providing insight into the parallel performance, multiobjective results, and energy consumption of the proposal. This evaluation is followed by the comparative assessment of heterogeneous design alternatives. Finally, comparisons with other state-of-the-art phylogenetic methods are presented.

The hardware platform used in the experimentation is a CPU+GPU system comprising, in the CPU side, two Intel Xeon Gold 5218 processors at 2.30 GHz (a total of 32 physical cores available), with 22MB L3 cache and 64GB DDR4 RAM. Two different CUDA-enabled GPUs have been plugged in this system with the aim of examining our heterogeneous approach on high-performance and commercial GPGPU devices:

- Tesla V100, based on the Volta architecture, composed of 80 streaming multiprocessors (5120 CUDA cores) with a GPU clock of 1380 MHz, a global memory of 16 GB at 877 MHz and 4096-bit memory bus width;
- RTX 2080Ti, based on the Turing architecture, composed of 68 streaming multiprocessors (4352 CUDA cores) with a GPU clock of 1545 MHz, a global memory of 11GB at 7000 MHz and 352-bit memory bus width.



(a) MPI-to-CUDA alternative



(b) OpenMP-to-CUDA alternative

Fig. 3. Heterogeneous design variants: (a) depicts a proposal based on MPI-to-CUDA interactions, in which each process instantiates the parsimony and likelihood GPU kernels once all the learning steps have been completed, transferring the complete pool of new solutions to the GPU prior to the kernel instantiations. On the other side, (b) shows an OpenMP-to-CUDA interaction scheme, where each thread assigns to the GPU a single candidate solution immediately after generating it inside the parallel learning loop.

The software in this system includes Ubuntu 18.04LTS, the GCC 8.4.0 compiler implementing OpenMP 4.5, MPICH 3.3.1 as the adopted MPI platform, and the CUDA compilation tools 10.1. The software herein tested was compiled by enabling the corresponding optimization and architecture-specific flags e.g. `-O3, -gencode:arch,code`. This hardware infrastructure and the considered GPU testing environments were chosen in accordance with the characteristics of the setups currently found in real biological studies [46]. In this way, it is possible to investigate the implications and the applicability of the proposed multi-level method in scenarios commonly found in parallel bioinformatics research.

In order to test the performance of the proposal on a heterogeneous range of problem sizes, five real-world datasets have been employed in the experimentation (assuming the General Time Reversible evolutionary model [6]):

1. 50 × 18321: lepidopteran superfamily Bombycoidea gene data, containing 50 sequences with 18,321 characters per sequence (9141 distinct alignment patterns) [47];
2. 55 × 1314: rbcL plastid gene data, containing 55 sequences with 1314 characters per sequence (789 distinct alignment patterns) [48];
3. 169 × 24251: Mammalian organisms DNA data, containing 169 sequences with 24,251 characters per sequence (24,234 distinct alignment patterns) [49];
4. 218 × 4182: Prokaryotic RNA data, containing 218 sequences with 4182 characters per sequence (1846 distinct alignment patterns) [50];
5. 500 × 759: rbcL plastid gene data, 500 sequences with 759 characters per sequence (759 distinct alignment patterns) [51].

These datasets cover diverse, representative evaluation scenarios from real-world phylogenetic analyses, with different characteristics according to the two key dimensions of the problem: number of sequences and sequence length. In this way, it can be thoroughly evaluated the relevance of the proposed

multi-level approach in a diversified set of problem instances comprising real data, each one with different implications on parallel performance, energy requirements, and multiobjective search constraints.

In order to ensure a rigorous, statistically-reliable experimental evaluation, 31 independent runs of the methods under study were performed per experiment. Furthermore, the statistical testing of solution quality results followed the methodology described in [52]. The Kolmogorov–Smirnov normality test was applied, in a first step, to examine the distribution of the results samples. Homoscedasticity was then analysed by using the Levene test in case of dealing with Gaussian distributions, accordingly applying ANOVA afterwards if homogeneity in variances was observed. For the remaining cases (no Gaussian distributions, no homoscedasticity), the evaluation of statistically significant differences among samples was conducted by using the Wilcoxon–Mann–Whitney tests. A confidence level of 95% was set in the statistical tests herein applied.

5.1. Configuration of the parallel metaheuristic

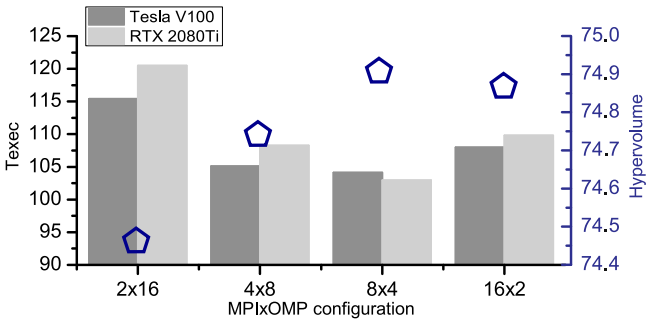
The first step in this experimental evaluation involves the configuration of input parameters to determine their influence in the computational and optimization capabilities of the proposal. With this purpose in mind, parametric studies were conducted to evaluate parallel and multiobjective performance under a range of uniformly-distributed candidate input values, assuming the MPI-to-CUDA version of the application. In accordance with previous studies [42], an adaptive adjustment control (ac) value of 5 generations and a stop criterion ($maxEval$) of 12,000 evaluations were initially established, so that the parametric studies could be focused on the key parameters that govern the behaviour of the proposal: the population size ($popSize$), the distribution of memeplexes (m, managed by MPI processes), and the learning steps (n_l, handled by OpenMP threads).

Using for testing purposes the dataset 218x4182, the execution time of each candidate configuration profile was first measured and evaluated. Second, the multiobjective quality of the achieved solutions was examined by using hypervolume. Hypervolume (I_H) is a widely-adopted multiobjective metric that calculates the volume of the objective space (area for bi-objective problems) that is covered by at least one of the solutions x reported by a multiobjective optimizer, that is, the volume of the orthogonal polytope $\prod^n$:

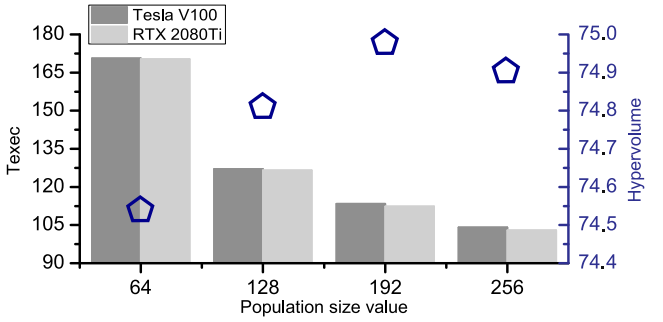
$$\prod^n = \{p \in \mathbb{R}^n : p \preceq x, \text{ for some } x \in X\}. \tag{10}$$

When comparing candidate settings and optimizers, the ones achieving higher hypervolume scores are preferred from a multiobjective point of view, since they denote more satisfying optimization capabilities.

Fig. 4(a) illustrates the median execution time and hypervolume scores reported by different distributions of memeplexes and learning steps, mapped through the MPI and OpenMP layers. Taking into account the characteristics of the hardware platform herein used, configurations of 2 processes × 16 threads, 4 processes × 8 threads, 8 processes × 4 threads, and 16 processes × 2 threads were evaluated, with a fixed population size of 256 individuals. The 2 × 16 configuration reported the worst results for both execution time and hypervolume, due to the limited concurrency capabilities observed at the evaluation procedures and the restrictions imposed on the adaptive evolutionary process by the use of a low number of memeplexes. In contrast, the 4 × 8 and 8 × 4 configurations led to the most satisfying behaviour from a time perspective. In fact, the 8 × 4 configuration was found to be the best performing candidate in terms of both execution time and hypervolume. In the case of 16 × 2, it can be observed a worsening in execution time due to the impact of communications with an increased number of MPI processes, as well as a reduction in hypervolume quality with regard to 8 × 4 as a result of the lower solution diversity available within the memeplexes.



(a) Configuration of MPI processes and OpenMP threads



(b) Configuration of the population size

Fig. 4. Comparison of execution time (T_{exec}, in seconds) and multiobjective performance (hypervolume, in %) for different parametric configurations of MO-SFLA on 218x4182.

Using the 8 × 4 distribution, the effect of varying the population size, with 64, 128, 192, and 256 individuals, can be observed in Fig. 4(b). From an execution time perspective, the best performance is attained when a population size of 256 individuals was used. This input value implies that the parallel loops and kernels can operate over a larger number of solutions, reducing the number of generations and therefore the impact of serial components. Although this approach enhances parallel performance, it has a negative impact in multiobjective quality, as the algorithm is given less iterations to evolve the populations until the stop criterion is satisfied. Consequently, a worsening in hypervolume is observed when going beyond 192 individuals, being this value the optimal population size from a multiobjective perspective.

From these results, it can be determined two recommended configuration profiles for the population size, in accordance with the desired parallel and solution quality properties. In case of putting priority to execution time, the use of 256 individuals represents the strategy that attains the most satisfying behaviour, without implying a relevant degradation in multiobjective performance. On the other hand, strict multiobjective quality requirements can be better satisfied by using a population size of 192 individuals. These two profiles, designated as *BestPar* and *BestMO*, have been considered throughout the experimental evaluation herein presented.

5.2. Parallel, multiobjective, and energy results

Once these configuration profiles have been identified, the evaluation of the multi-level parallel design can be undertaken.

In order to highlight the benefits of multi-level parallelism in this context, three evaluation criteria have been considered: parallel performance, multiobjective quality, and energy consumption. The main idea lies in exploring the parallel performance gains attained by the proposal over other traditionally-adopted approaches i.e. CPU-only, mixed-mode multicore designs, as well as potential impacts in multiobjective capabilities and energy consumption.

The median parallel results observed in the experimentation are presented in Table 2. For each dataset, the execution times reported by the CPU-only, multicore version of MO-SFLA and the proposed multi-level design on Tesla V100 and RTX 2080Ti are shown in the T_{exec} entries (columns 2, 3, and 6 respectively). Moreover, the *SU* entries outline the speedups achieved by the multi-level designs over the CPU serial implementation (*SU Serial* columns) and the multicore version (*SU Multicore* columns, using 32 physical cores). The results from the *BestPar* configuration profile are reported in the upper side of Table 2, while the bottom side shows results from the *BestMO* profile.

This comparative evaluation points out the attainment of effective speedups over CPU implementations in all the evaluation scenarios. The comparison with the serial version of the algorithm suggests that the exploration and exploitation of multi-level parallelism represents a suitable solution to accelerate time-consuming phylogenetic analyses, with performance gains up to 396×. In comparison to the multicore implementation, the best case scenarios are observed in the datasets with the highest number of unique sequence patterns (169 × 24,251 and 50 × 18,321), which represent the problem sizes with the largest available data parallelism. For the specific case of 169 × 24,251, speedups of 14.3× (*BestPar*) and 13.1× (*BestMO*) are achieved when using Tesla V100 at the solution-level layer, maintaining similar performance on the commercial RTX 2080Ti (13.5× and 12.4× respectively). Even in the case of datasets with limited data parallelism e.g. 500 × 759, the multi-level approach shows improved parallel results thus justifying the applicability of the proposal on different real-world phylogenetic scenarios.

A comparison of time profiles between the CPU-only, multicore version and the multi-level proposal is depicted in Fig. 5. This figure reports the normalized percentage of time spent in the evaluation of candidate solutions (objective functions), evolutionary operators (learning from best global, best local, and local searches), and serial operations (including overhead sources i.e. synchronizations and communications). On analysing the CPU implementation, it can be observed the significant role played by the evaluation procedures in the computations, representing the task that dominates the execution time in a higher degree. When applying multi-level parallelism, with solution-level calculations on the GPU, the resulting profile denotes a more balanced distribution of times, in which the evaluation time issues shown by the reference version are successfully tackled. Other operations, especially the local searches, become increasingly noticeable under this approach.

The second aspect to be evaluated is the potential impact in multiobjective quality, with the aim of determining if the multi-level parallel design is able to preserve the search capabilities of the original algorithm. Table 3 introduces the comparison of multiobjective results between the multi-level proposal and the CPU-only baseline implementation, detailing median hypervolume scores I_H (columns 2 and 4), interquartile ranges *IQR* (columns 3 and 5), and the P-values obtained from the statistical tests applied over the attained result samples (column 6).

In terms of hypervolume, the multi-level proposal is able to reach the multiobjective quality of the baseline metaheuristic under the two considered configuration profiles, reporting high-quality hypervolume scores (over 71%) in all the datasets herein evaluated. In all these evaluation scenarios, the attained P-values are noticeable higher than the significance threshold 0.05, which means that statistically significant differences were not found between the multi-level proposal and the baseline version of MO-SFLA. Consequently, this multiobjective performance analysis confirms that the proposed multi-level parallel design maintains the optimization capabilities of MO-SFLA, with significantly reduced execution times. A representation of the Pareto front distributions obtained by the proposal is depicted in Fig. 6.

The last evaluation point in this experimental analysis is focused on the assessment of energy consumption. In order to examine this aspect, power measurements were taken, with timestamps of 100 ms, by using (1) Intel's Running Average Power Limit (RAPL) counters [53] at the CPU side; and (2) the NVIDIA System Management Interface (NVIDIA-SMI) [54] at the GPU side. Table 4 illustrates energy results from the CPU-only multicore implementation (columns 2–3) and the multi-level proposal on Tesla V100 (columns 4–6) and RTX 2080Ti (columns 7–9). For each approach, the energy consumed per device in Joules is reported, in accordance with the measurements observed at the RAPL packages ($J(CPU)$), the RAPL DRAM domains ($J(DRAM)$), and the NVIDIA-SMI GPU samples ($J(GPU)$). In order to simplify this analysis, the results reported in Table 4 correspond to the *BestPar* configuration profile of the metaheuristic.

The comparison of energy results gives account of significant savings in the overall consumed energy when the proposed multi-level design is used. More specifically, the multi-level parallel approach is able to attain improvements in the accumulated energy consumption of 85.9% (50 × 18,321), 81.5% (55 × 1314), 92.0% (169 × 24251), 77.8% (218 × 4182), and 37.7% (500 × 759) over the baseline multicore version when using the Tesla V100 GPU. It can be highlighted that these improvements become more noticeable in datasets with a larger number of unique sequence patterns, as observed in the case of 50 × 18,321 and 169 × 24251. If the commercial RTX 2080Ti GPU is employed instead, effective energy improvements of 82.2% (50 × 18,321), 78.6% (55 × 1314), 89.8% (169 × 24251), 73.1% (218 × 4182), and 29.3% (500 × 759) are achieved. These results suggest that, despite the differences in energy efficiency between high-performance and commercial GPUs, the adoption of multi-level heterogeneous strategies consistently leads to significant energy improvements in all the evaluation scenarios herein considered.

In order to further examine the behaviour of the proposal, Fig. 7 shows the evolution of power per device (in W) throughout the execution of the algorithm, using the RTX 2080Ti GPU in the 50 × 18,321 dataset. These power measurements provide a clear depiction of the different steps defined in the algorithmic design of MO-SFLA. Initially, the CPU is in charge of conducting the initialization of populations, data structures, and APIs, while the GPU remains waiting for the initial set of individuals to be scored. Afterwards, it can be observed alternating patterns between the CPU and GPU, which correspond with the execution of algorithm-level, iteration-level, and solution-level parallel tasks within a generation. Due to the fact that the algorithm defines different search operators, it can be noticed that the generations do not show homogeneous time and power requirements. Particularly, the gaps observed in some generations prior to the GPU offloading are motivated by the execution of the most time-demanding operator, local searches, in those situations where stagnated individuals were identified in the memeplexes.

In conclusion, these results denote the satisfying parallel and energy performance achieved by the multi-level parallel approach. The proposed design is also consistent with the original search capabilities of the baseline MO-SFLA algorithm, which are successfully preserved as suggested by the statistical assessment of multiobjective quality.

Table 2
Multi-level MO-SFLA: Comparison of parallel performance (median execution time T_{exec} in seconds and speedups *SU*) with CPU-only implementations.

	BestPar configuration						
Dataset	CPU-only	Multi-level GPU Tesla V100			Multi-level GPU RTX 2080Ti		
	Multicore T_{exec}	Time T_{exec}	*SU* Serial	*SU* Multicore	Time T_{exec}	*SU* Serial	*SU* Multicore
50 × 18321	656.202	85.491	216.031×	**7.676**×	89.972	205.272×	**7.293**×
55 × 1314	57.980	11.358	131.855×	**5.105**×	11.174	134.034×	**5.189**×
169 × 24251	14124.429	987.662	396.063×	**14.301**×	1047.741	373.352×	**13.481**×
218 × 4182	486.137	104.126	140.019×	**4.669**×	102.966	141.596×	**4.721**×
500 × 759	861.774	481.850	50.598×	**1.788**×	486.995	50.064×	**1.770**×
	BestMO configuration						
50 × 18321	684.488	95.518	193.353×	**7.166**×	98.886	186.768×	**6.922**×
55 × 1314	58.936	11.554	129.619×	**5.101**×	11.462	130.657×	**5.142**×
169 × 24251	14469.671	1105.350	353.894×	**13.091**×	1166.745	335.272×	**12.402**×
218 × 4182	495.455	113.426	128.539×	**4.368**×	112.384	129.730×	**4.409**×
500 × 759	875.528	490.780	49.678×	**1.784**×	494.839	49.270×	**1.769**×

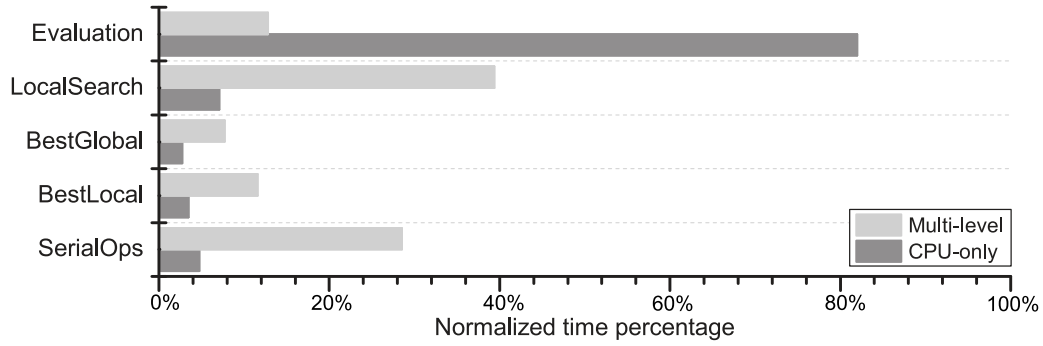


Fig. 5. Normalized time profiles for the CPU-only and multi-level heterogeneous versions of MO-SFLA, illustrating the percentages of time spent on evaluations, local searches, learning from best local or from best global, and serial operations (including overhead).

Table 3
Multi-level MO-SFLA: Comparison of multiobjective quality (median hypervolume I_H, interquartile range *IQR*, and P-values) with CPU-only implementations.

	BestPar configuration				
Dataset	CPU-only		Multi-level		*P*-value
	I_H	*IQR*	I_H	*IQR*	test
50 × 18321	77.145%	±0.010	77.145%	±0.006	**0.910**
55 × 1314	71.778%	±0.017	71.783%	±0.017	**0.722**
169 × 24251	77.780%	±0.105	77.776%	±0.145	**0.451**
218 × 4182	74.896%	±0.027	74.902%	±0.052	**0.322**
500 × 759	73.019%	±0.037	73.022%	±0.021	**0.978**
	BestMO configuration				
50 × 18321	77.146%	±0.010	77.148%	±0.007	**0.804**
55 × 1314	71.778%	±0.010	71.783%	±0.014	**0.558**
169 × 24251	77.839%	±0.061	77.835%	±0.059	**0.468**
218 × 4182	74.936%	±0.020	74.966%	±0.047	**0.429**
500 × 759	73.037%	±0.019	73.033%	±0.015	**0.699**

Table 4
Multi-level MO-SFLA: Comparison of energy consumption per device (in Joules *J*) with CPU-only implementations.

Dataset	CPU-only		Multi-level GPU TeslaV100			Multi-level GPU RTX2080Ti		
	J(CPU)	*J*(DRAM)	*J*(CPU)	*J*(DRAM)	*J*(GPU)	*J*(CPU)	*J*(DRAM)	*J*(GPU)
50 × 18321	157382.7	9232.2	18256.3	1034.6	**4195.3**	19068.6	1059.9	9565.3
55 × 1314	13206.6	863.6	1957.9	106.3	**539.2**	1910.9	104.2	989.6
169 × 24251	3334761.3	266915.5	228968.4	17006.2	**42821.9**	237920.7	17394.4	111730.9
218 × 4182	110821.6	6987.7	20120.6	1136.2	**4899.7**	19820.9	1108.7	10738.7
500 × 759	192305.1	10219.8	102190.1	4575.0	**19455.6**	103225.4	4598.8	35241.7

5.3. Evaluation of heterogeneous design alternatives

Next, the experimental assessment of the heterogeneous design alternatives herein devised is undertaken. We will first focus on examining the differences in parallel performance between the two CPU–GPU interaction schemes, MPI-to-CUDA and OpenMP-to-CUDA, by analysing execution times. For each dataset, Table 5 shows the median time results attained by each strategy (columns 2–5 for MPI-to-CUDA and columns 6–9 for OpenMP-to-CUDA) on the Tesla V100 and RTX 2080Ti GPUs.

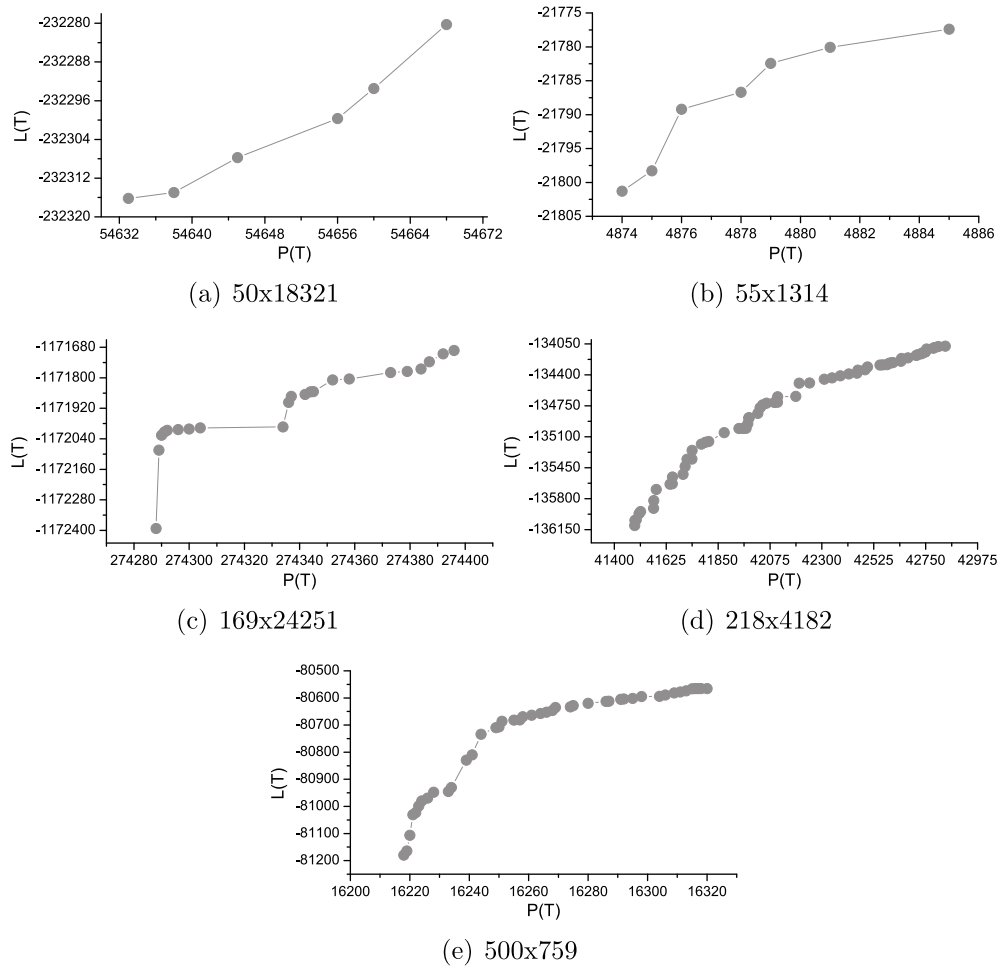


Fig. 6. Representation of the median-hypervolume Pareto fronts generated by the multi-level approach (using the BestMO configuration). These plots illustrate the heterogeneity of the considered problem instances, in terms of diverging front shapes and distributions.

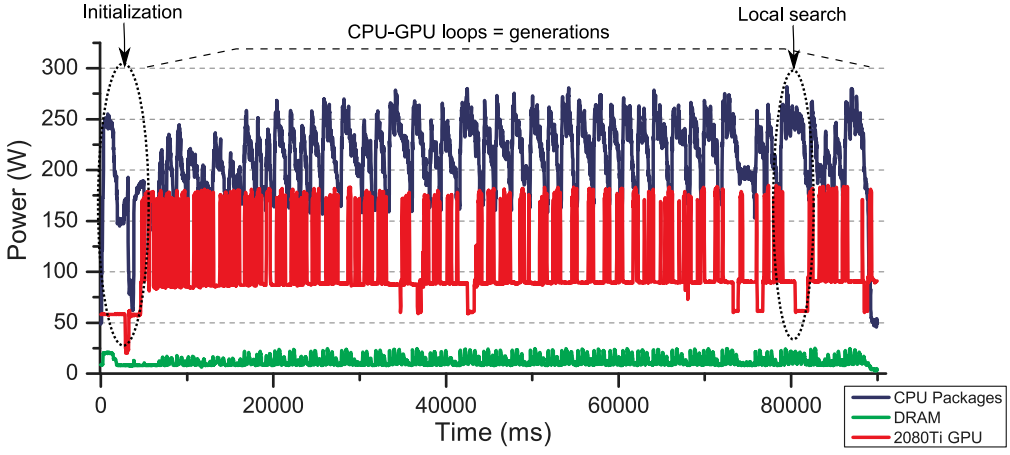


Fig. 7. Evolution of power measurements (CPU packages, DRAM, and GPU RTX 2080Ti) during the execution of the multi-level approach on the 50 × 18,321 dataset.

For both *BestPar* and *BestMO* configuration profiles, the OpenMP-to-CUDA scheme leads to general improvements over the MPI-to-CUDA variant. The most noteworthy scenario is given by the 55 × 1314 dataset, for which the OpenMP-to-CUDA interaction mapping allows the algorithm to finish execution 10%–13% faster than the MPI-to-CUDA approach. However, our results highlight one particular scenario that OpenMP-to-CUDA failed to run: 169 × 24251. This dataset, which was successfully handled by the MPI-to-CUDA variant, implied increased memory consumption requirements for OpenMP-to-CUDA that resulted into `OutOfMem` exceptions at the GPU side.

In order to understand the impact of memory requirements on each heterogeneous alternative, NVIDIA-SMI was employed to monitor memory usage in runtime. The results retrieved by this tool are reported in Table 6. In average terms, the memory consumption of OpenMP-to-CUDA is about 1.61 (Tesla V100) and

Table 5
Heterogeneous design alternatives: comparisons of execution time (in seconds).

Dataset	Execution time T_{exec}							
	MPI-to-CUDA version				OpenMP-to-CUDA version			
	Tesla V100		RTX 2080Ti		Tesla V100		RTX 2080Ti	
	BestPar	BestMO	BestPar	BestMO	BestPar	BestMO	BestPar	BestMO
50 × 18321	85.491	95.518	89.972	98.886	**84.824**	**94.871**	**86.763**	**97.150**
55 × 1314	11.358	11.554	11.174	11.462	**10.212**	**10.410**	**9.702**	**9.935**
169 × 24251	**987.662**	**1105.350**	**1047.741**	**1166.745**	OutMem	OutMem	OutMem	OutMem
218 × 4182	104.126	113.426	102.966	112.384	**100.739**	**112.451**	**100.596**	**109.349**
500 × 759	481.850	490.780	486.995	494.839	**473.656**	**484.602**	**479.148**	**488.291**

Table 6
Heterogeneous design alternatives: comparisons of GPU memory consumption (in MiB).

	GPU memory usage			
	MPI-to-CUDA version		OpenMP-to-CUDA version	
	Tesla V100	RTX 2080Ti	Tesla V100	RTX 2080Ti
50 × 18321	**3432**	**2200**	6296	5080
55 × 1314	**2584**	**1352**	2888	1672
169 × 24251	**10936**	**9704**	OutMem	OutMem
218 × 4182	**3320**	**2088**	5848	4632
500 × 759	**3288**	**2056**	5736	4520

1.99 (RTX 2080Ti) times higher than MPI-to-CUDA, as a result of the increased memory allocation pressure demanded by the thread-stream mapping. In this sense, the BEAGLE library instances contribute the most to the consumed memory, due to the size of the additional data structures required to perform likelihood calculations.

From these results, it can be highlighted the main benefits and drawbacks of each heterogeneous alternative. First, the OpenMP-to-CUDA scheme represented the best performing approach in terms of execution time, yet its applicability directly depends on the characteristics of the input data (i.e. sequence length and site patterns) and the memory size of the employed GPU device. A practical strategy to increase the viability of this heterogeneous variant lies in the introduction of dataset split techniques, so that the GPU can perform partial scoring tasks on different partitions of the input sequences without overflowing memory capacity. Nevertheless, the MPI-to-CUDA scheme provides the multi-level metaheuristic with increased flexibility to satisfactorily tackle different problem sizes, at the expense of a potential impact in the achievable parallel results.

5.4. Comparisons with the state of the art

In order to validate the relevance of the proposed multi-level design of MO-SFLA, comparisons with state-of-the-art parallel methods for phylogenetic inference are introduced next. This comparative analysis has been conducted with regard to four reference methods: the single-criterion biological tools RAxML [34], IQ-TREE [35], and MrBayes [36], as well as the multiobjective approach PhyloMOEA [37]. For comparison purposes, we will employ the MPI-to-CUDA heterogeneous scheme and the *BestPar* configuration profile of MO-SFLA.

In a first step, comparisons with RAxML, IQ-TREE, and MrBayes are introduced. These biological tools integrate different algorithmic strategies and parallel schemes to carry out single-objective phylogenetic analyses. RAxML defines several search strategies, e.g. rapid hill climbing and lazy subtree rearrangements, within a parallel design that implements job-level parallelism to distribute independent phylogeny reconstructions via MPI and the combination of POSIX threads and SIMD vectorization to accelerate likelihood scoring operations. On the other side, IQ-TREE implements a master–worker approach, based on mixed mode MPI+OpenMP programming and SIMD CPU instructions, to parallelize a hill-climbing optimization approach with stochastic perturbation components. Finally, MrBayes is a Markov Chain Monte Carlo method for Bayesian phylogenetics, which combines MPI and the BEAGLE library to efficiently process independent Markov chains on CPU+GPU systems. These tools are widely adopted due to their search features and their capabilities to exploit hardware resources, boosting performance in important research scenarios [55–57].

Since successful comparisons of phylogenetic quality between MO-SFLA and these biological tools were reported in previous research [42], the analysis herein presented is focused on examining parallel performance and energy consumption. With this purpose in mind, we have evaluated the median execution times and energy results attained by the reference methods on each dataset, considering as a stop criterion the same number of phylogenetic trees generated by MO-SFLA. Table 7 provides insight into parallel performance, reporting execution times T_{exec} for RAxML, IQ-TREE, and MrBayes (columns 2–4) and the speedups *SU* attained by our multi-level MO-SFLA proposal over each method (columns 5–7). For all the evaluated problem sizes, the proposed parallel approach successfully leads to the best execution times, achieving speedups up to 20.8× over RAxML, 27.0× over IQ-TREE, and 35.5× over MrBayes.

The use of optimized data representations and the accurate orchestration of the multi-level layers benefit the processing of complex datasets, not only when it comes to large sequence lengths (e.g. 169 × 24,251) but also in the case of instances with a high number of sequences (500 × 759). Even for datasets with limited data parallelism, the multi-level proposal is able to report effective improvements in the execution time required to complete the phylogenetic analysis. The best-case scenario is observed for 218 × 4182, which is a dataset characterized by site patterns with poor uniformity and gaps. While the other phylogenetic tools are impacted by the challenging features of this dataset, the strategies implemented in the proposal lead to significant results in this difficult scenario. It is also important to remark that the multi-level MO-SFLA addresses a harder, more computationally demanding multiobjective formulation of the problem, in comparison with the single-objective nature of the biological methods herein tested. This fact further highlights the benefits of the proposal and the significant parallel performance attained by accurately exploiting multi-level parallelism on heterogeneous systems.

Regarding energy consumption, Table 8 reports the accumulated energy results *J* observed for the biological methods (columns 2–4) and the improvements achieved by the multi-level MO-SFLA (columns 5–7). These results denote the satisfying behaviour of the proposal from an energy consumption perspective, improving the results of RAxML in almost all the datasets while also outperforming IQ-TREE and MrBayes in all the evaluation scenarios. More specifically, MO-SFLA reports average energy improvements of 41.7% (over RAxML), 60.6% (IQ-TREE), and 56.0%

Table 7
Comparisons with other parallel phylogenetic methods: parallel performance (execution times in seconds) from each tool and improvements obtained by the proposed multi-level MO-SFLA.

Dataset	Execution time T_{exec}			MO-SFLA speedups *SU*		
	RAxML	IQ-TREE	MrBayes	RAxML	IQ-TREE	MrBayes
50 × 18321	111.048	123.322	108.980	**1.299**×	**1.443**×	**1.275**×
55 × 1314	11.883	19.812	26.780	**1.046**×	**1.744**×	**2.358**×
169 × 24251	4060.592	6742.647	1662.210	**4.111**×	**6.827**×	**1.683**×
218 × 4182	2160.601	2813.444	3692.420	**20.750**×	**27.020**×	**35.461**×
500 × 759	1127.645	2493.391	3015.360	**2.340**×	**5.175**×	**6.258**×

Table 8
Comparisons with other parallel phylogenetic methods: energy consumption (accumulated J) from each tool and improvements obtained by the proposed multi-level MO-SFLA.

Dataset	Energy consumption *J*			MO-SFLA energy improvements		
	RAxML	IQ-TREE	MrBayes	RAxML	IQ-TREE	MrBayes
50 × 18321	24593.405	29167.997	25426.717	△**4.502**%	△**19.480**%	△**7.632**%
55 × 1314	2379.591	3830.229	5574.066	▽9.407%	△**32.029**%	△**53.294**%
169 × 24251	1010391.235	1523083.634	486026.767	△**71.417**%	△**81.039**%	△**40.580**%
218 × 4182	489577.856	614903.163	869106.382	△**94.657**%	△**95.746**%	△**96.990**%
500 × 759	240021.156	500153.239	682048.375	△**47.413**%	△**74.764**%	△**81.494**%

Table 9
Comparisons with other parallel phylogenetic methods: speedups for PhyloMOEA and MO-SFLA (CPU-only multicore and CPU+GPU multilevel versions). The results from PhyloMOEA and the CPU-only MO-SFLA refer to executions with 16 cores.

Dataset	Speedups *SU*		
	PhyloMOEA	CPU-only MO-SFLA	Multi-level MO-SFLA
55 × 1314	8.300×	**14.446**×	**131.855**×
218 × 4182	10.200×	**15.665**×	**140.019**×
500 × 759	6.300×	**15.283**×	**50.598**×

(MrBayes). The energy-aware nature of the method is therefore supported by the comparisons herein presented, thus suggesting that both performance gains and energy savings can be attained when solving high-complexity optimization problems through multi-level, heterogeneous strategies.

In order to further examine the performance of MO-SFLA, we introduce in Table 9 additional comparisons with PhyloMOEA, an alternative parallel multiobjective method. This multiobjective approach adopts MPI processes to distribute, under a master–worker scheme, the evaluation of new candidate solutions at each generation, complementarily using OpenMP threads to parallelize likelihood site calculations. Table 9 illustrates the speedups reported by PhyloMOEA in [37] for 16 cores, in comparison to the results obtained by the CPU-only and multi-level variants of MO-SFLA. This comparative analysis is focused on the datasets 55 × 1314, 218 × 4182, and 500 × 759, which were also employed in [37].

In these datasets, PhyloMOEA is able to report effective speedups of 8.3×, 10.2×, and 6.3× over the corresponding serial version. When the same amount of processing resources is considered (16 cores), it can be observed that the CPU-only, multicore version of MO-SFLA achieves better performance from a parallel perspective, with speedups up to 15.7×. Furthermore, when all the heterogeneous resources are employed, the multi-level implementation of MO-SFLA significantly boosts the speedup results up to 140.0× in these datasets. In practical terms, this enhanced parallel exploitation leads to execution times below 10 min for the 500x759 dataset, thus improving the 6 h originally reported by PhyloMOEA in [37]. These results denote the importance of the heterogeneous parallel strategies devised at the algorithm, iteration, and solution layers of MO-SFLA. The proposed multi-level design allows increased parallel performance by taking full advantage of the three key parallelization levels available in metaheuristic optimizers, in comparison to alternative approaches that partially combine them.

According to this experimental analysis, the proposal therefore represents a promising approach for orienting the parallel processing capabilities of state-of-the-art heterogeneous platforms towards the efficient solution of hard-to-tackle bioinformatics problems.

6. Conclusions and future research works

As the complexity of real-world optimization problems keeps growing with the increasing availability of larger, hard-to-process data and the definition of more accurate mathematical models, the development of efficient parallel metaheuristics has become a fundamental research direction. This research work has investigated the combination of multi-level parallelism and heterogeneous computing to address complex multiobjective optimization problems, focusing on a challenging bioinformatics problem: phylogenetic reconstruction. The proposed method was built upon the algorithmic design of MO-SFLA, an intrinsically parallel multiobjective metaheuristic, and exploits parallelism at three different layers: (1) algorithm level, (2) iteration level, and (3) solution level. Different strategies have been defined to accurately map each parallelism level to the most suitable computing device on heterogeneous CPU+GPU environments, using MPI, OpenMP, and CUDA/BEAGLE in accordance with the characteristics of the computations to be performed at each step. Moreover, two main heterogeneous schemes have been devised to orchestrate CPU–GPU interactions, with the aim of examining the performance attained by different design alternatives.

An in-depth experimental evaluation was carried out to examine the benefits of the multi-level, heterogeneous proposal on a range of different problem sizes, represented by five real-world biological datasets. Two main configuration profiles for the parallel metaheuristic were identified from the initial parametric studies, each one representing the parameter settings recommended to achieve the desired parallel and multiobjective performance goals. Using these profiles, the experimental evaluation gave account of the significant results attained by the proposal from a parallel performance perspective, accelerating the baseline serial algorithm up to 396×.

In comparison to an alternative CPU-only multicore implementation, effective accelerations up to 14.3× were observed, with no performance degradation in multiobjective quality. Furthermore, noticeable energy savings on both high-performance

and commercial GPUs were verified in all the evaluation tests, being the best case scenario observed in the dataset with the longest sequence length (energy improvements of 92.0% when using a Tesla V100 GPU to offload solution-level tasks). Additionally, the comparison of heterogeneous schemes revealed performance-memory consumption tradeoffs, depending on the attachment of CUDA/BEAGLE streams to MPI processes or OpenMP threads. Finally, the comparisons with state-of-the-art biological tools suggested the satisfying time and energy savings achieved by the proposal, thus confirming the key role that multi-level parallelism and heterogeneous systems can play in the solution of grand computational challenges.

Our future research lines are mainly oriented towards the integration of new high-performance strategies to further optimize the proposal. In a first step, dataset split strategies will be investigated to address potential memory consumption issues on large datasets. The adoption of asynchronous, non-generational parallel schemes also represents a promising approach to minimize the effect of overhead sources. Moreover, the increased heterogeneity of current hardware architectures provides opportunities to extend the proposal to more sophisticated platforms and cluster configurations, combining CPUs, integrated and dedicated GPUs, and other hardware co-processors. In this context, efficient load balancing and orchestration techniques will be devised to maximize the exploitation of the underlying hardware resources. An important question to be addressed lies in the comparison between internal power monitoring tools and external measurement devices to analyse energy consumption accurately in the presence of such heterogeneity. Finally, the evaluation of the proposal in other optimization problems, fundamentally from the bioinformatics domain, will be undertaken.

CRediT authorship contribution statement

Sergio Santander-Jiménez: Conceptualization, Methodology, Investigation, Software, Data curation, Validation, Visualization, Writing – original draft. **Miguel A. Vega-Rodríguez:** Conceptualization, Methodology, Investigation, Resources, Project administration, Funding acquisition, Writing – review & editing. **Leonel Sousa:** Conceptualization, Methodology, Formal analysis, Resources, Project administration, Funding acquisition, Writing – review & editing.

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.

Acknowledgements

This work was partially funded by the MCIU (Ministry of Science, Innovation and Universities, Spain), the AEI (State Research Agency, Spain), and the ERDF (European Regional Development Fund, EU), under the contract PID2019-107299GB-I00/AEI/10.13039/501100011033 (Multi-HPC-Bio project), as well as Portuguese funds through FCT (Fundação para a Ciência e a Tecnologia, Portugal) projects UIDB/50021/2020 and LISBOA-01-0145-FEDER-031901 (PTDC/CCI-COM/31901/2017, HiPErBio).

References

[1] C. Coello, S. González, J. Figueroa, M.G. Castillo, R. Hernández, Evolutionary multiobjective optimization: open research areas and some challenges lying ahead, Complex Intell. Syst. 6 (2020) 221–236.
[2] G. Schryen, Parallel computational optimization in operations research: A new integrative framework, literature review and research directions, European J. Oper. Res. 287 (1) (2020) 1–18.
[3] E.G. Talbi, A unified view of parallel multi-objective evolutionary algorithms, J. Parallel. Distrib. Comput. 133 (2019) 349–358.
[4] S. Limmer, D. Fey, Comparison of common parallel architectures for the execution of the island model and the global parallelization of evolutionary algorithms, Concurr. Comp. Pract. E. 29 (9) (2017) e3797, 1–31.
[5] M. Abbasi, M. Rafiee, M.R. Khosravi, A. Jolfaei, V.G. Menon, J.M. Koushyar, An efficient parallel genetic algorithm solution for vehicle routing problem in cloud implementation of the intelligent transportation systems, J. Cloud Comput. 9 (2020) 6, 1–14.
[6] T. Warnow, Computational Phylogenetics: An Introduction To Designing Methods for Phylogeny Estimation, Cambridge Univ. Press, Cambridge, UK, 2017.
[7] S. Santander-Jiménez, M.A. Vega-Rodríguez, L. Sousa, Multiobjective frog-leaping optimization for the study of ancestral relationships in protein data, IEEE Trans. Evol. Comput. 22 (6) (2018) 879–893.
[8] W. Gropp, W. Lusk, A. Skjellum, Using MPI: Portable Parallel Programming with the Message Passing Interface, third ed., The MIT Press, Cambridge, MA, USA, 2014.
[9] R. van der Pas, E. Stotzer, C. Terboven, Using OpenMP - the Next Step, The MIT Press, Cambridge, MA, USA, 2017.
[10] N. Wilt, The CUDa Handbook: A Comprehensive Guide To GPU Programming, Addison Wesley, Pearson, NJ, USA, 2013.
[11] P. Ribalta, J. Nalepa, L. Sánchez, J. Ranilla, Hyper-parameter selection in deep neural networks using parallel particle swarm optimization, in: Proc. of GECCO 2017, ACM, 2017, pp. 1864–1871.
[12] J.J. Moreno, G. Ortega, E. Filatovas, J.A. Martínez, E.M. Garzón, Using low-power platforms for evolutionary multi-objective optimization algorithms, J. Supercomput. 73 (2017) 302–315.
[13] R. Gadea-Gironés, R. Colom-Palero, V. Herrero-Bosch, Optimization of deep neural networks using SoCs with OpenCL, Sensors 18 (5) (2018) 1384, 1–23.
[14] J.A. Gomez-Pulido, et al., Fine-grained parallelization of fitness functions in bioinformatics optimization problems: gene selection for cancer classification and biclustering of gene expression data, BMC Bioinformatics 17 (2016) 330, 1–13.
[15] Y. Liu, Q. Liao, J. Sun, M. Hu, L. Liu, L. Zheng, Heterogeneous parallel genetic algorithm based on SW26010 processors, in: Proc. of IEEE HPCC/SmartCity/DSS 2019, IEEE, 2019, pp. 54–61.
[16] W. Yang, K. Li, K. Li, A pipeline computing method of SpTV for three-order tensors on CPU and GPU, ACM Trans. Knowl. Discov. Data 13 (6) (2019) 63, 1–27.
[17] C. Chen, K. Li, A. Ouyang, K. Li, FlinkCL: An OpenCL-based in-memory computing architecture on heterogeneous CPU-GPU clusters for big data, IEEE Trans. Comput. 67 (12) (2018) 1765–1779.
[18] J. Chen, K. Li, K. Bilal, X. Zhou, K. Li, P.S. Yu, A bi-layered parallel training architecture for large-scale convolutional neural networks, IEEE Trans. Parallel Distrib. Syst. 30 (5) (2019) 965–976.
[19] K. Li, J. Liu, L. Wan, S. Yin, K. Li, A cost-optimal parallel algorithm for the 0–1 knapsack problem and its performance on multicore CPU and GPU implementations, Parallel Comput. 43 (2015) 27–42.
[20] S. Santander-Jiménez, M.A. Vega-Rodríguez, Parallel multiobjective metaheuristics for inferring phylogenies on multicore clusters, IEEE Trans. Parallel Distrib. Syst. 26 (6) (2015) 1678–1692.
[21] T. Artés, A. Cencerrado, A. Cortés, T. Margalef, Time aware genetic algorithm for forest fire propagation prediction: exploiting multi-core platforms, Concurr. Comp. Pract. E. 29 (9) (2017) 1–18.
[22] B. Imbernón, J. Prades, D. Giménez, J.M. Cecilia, F. Silla, Enhancing large-scale docking simulation on heterogeneous systems: An MPI vs rCUDA study, Future Gener. Comput. Syst. 79 (Part 1) (2018) 26–37.
[23] A. García-Monzó, H. Migallón, A. Jimeno-Morenilla, J.L. Sánchez-Romero, H. Rico, R.V. Rao, Efficient subpopulation based parallel TLBO optimization algorithms, Electronics 8 (1) (2019) 19, 1–21.
[24] J. Luo, D.E. Baz, R. Xue, J. Hu, Solving the dynamic energy aware job shop scheduling problem with the heterogeneous parallel genetic algorithm, Future Gener. Comput. Syst. 108 (2020) 119–134.
[25] A. Cano, B. Krawczyk, Learning classification rules with differential evolution for high-speed data stream mining on GPUs, in: Proc. of IEEE CEC 2018, IEEE, 2018, pp. 1–8.
[26] J.J. Escobar, J. Ortega, A.F. Díaz, J. González, M. Damas, Multi-objective feature selection for EEG classification with multi-level parallelism on heterogeneous CPU-GPU clusters, in: Proc. of GECCO 2018, ACM, 2018, pp. 1862–1869.
[27] N. Alachiotis, A. Stamatakis, Acceleration of the phylogenetic parsimony kernel? in: Proc. of FPL 2011, IEEE, 2011, pp. 417–422.
[28] W.S. Martins, T.F. Rangel, D.C.S. Lucas, E.B. Ferreira, E.N. Caceres, Phylogenetic distance computation using CUDA, in: BSB 2012, 7409 of LNCS, Springer Verlag, 2012, pp. 168–178.
[29] T. Majumder, S. Sarkar, P.P. Pande, A. Kalyanaraman, Noc-based hardware accelerator for breakpoint phylogeny, IEEE Trans. Comput. 61 (6) (2012) 857–869.

[30] F. Izquierdo-Carrasco, N. Alachiotis, S. Berger, T. Flouri, S.P. Pissis, A. Stamatakis, A generic vectorization scheme and a GPU kernel for the phylogenetic likelihood library, in: Proc. of IEEE IPDPS 2013, IEEE, 2013, pp. 530–538.
[31] C. Ling, J. Gao, G. Lu, Phylogenetic likelihood estimation on GPUs using vertical partitioning scheme, in: Proc. of the 2016 IEEE Trustcom/BigDataSE/ISPA, IEEE, 2016, pp. 1210–1217.
[32] D.L. Ayres, et al., BEAGLE 3: Improved performance, scaling, and usability for a high-performance computing library for statistical phylogenetics, Syst. Biol. 68 (6) (2019) 1052–1061.
[33] S. Santander-Jiménez, M.A. Vega-Rodríguez, A. Zahinos-Márquez, L. Sousa, GPU Acceleration of fitch parsimony on protein data: From Kepler to Turing, J. Supercomput. 76 (2020) 9827–9853.
[34] A. Stamatakis, RAxML Version 8: A tool for phylogenetic analysis and post-analysis of large phylogenies, Bioinformatics 30 (9) (2014) 1312–1313.
[35] L.T. Nguyen, H.A. Schmidt, A. von Haeseler, B.Q. Minh, IQ-TREE: A fast and effective stochastic algorithm for estimating maximum likelihood phylogenies, Mol. Biol. Evol. 32 (1) (2015) 268–274.
[36] F. Ronquist, et al., MrBayes 3.2: Efficient Bayesian phylogenetic inference and model choice across a large model space, Syst. Biol. 61 (3) (2012) 539–542.
[37] W. Cancino, L. Jourdan, E.G. Talbi, A.C.B. Delbem, Parallel multi-objective approaches for inferring phylogenies, in: Proc. of EVOBIO'2010, 6023 of LNCS, Springer, 2010, pp. 26–37.
[38] S. Santander-Jiménez, M.A. Vega-Rodríguez, Applying a multiobjective metaheuristic inspired by honey bees to phylogenetic inference, BioSystems 114 (1) (2013) 39–55.
[39] X. Min, et al., Using MOEA with redistribution and consensus branches to infer phylogenies, Int. J. Mol. Sci. 19 (1) (2018) 62, 1–10.
[40] W. Fitch, Toward defining the course of evolution: Minimum change for a specific tree topology, Syst. Zool. 20 (4) (1972) 406–416.
[41] J. Felsenstein, Evolutionary trees from DNA sequences: A maximum likelihood approach, J. Mol. Evol. 17 (6) (1981) 368–376.
[42] S. Santander-Jiménez, M.A. Vega-Rodríguez, L. Sousa, A multiobjective adaptive approach for the inference of evolutionary relationships in protein-based scenarios, Inform. Sci. 485 (2019) 281–300.
[43] K. Deb, A. Pratap, S. Agarwal, T. Meyarivan, A fast and elitist multi–objective genetic algorithm: NSGA-II, IEEE Trans. Evol. Comput. 6 (2) (2002) 182–197.
[44] E. Zitzler, S. Künzli, Indicator-based selection in multiobjective search, in: Parallel Problem Solving from Nature VIII, 3242 of LNCS, Springer Verlag, 2004, pp. 832–842.
[45] P.O. Lewis, STROM Tutorial: Calculating the likelihood (Chapter 10), 2020, https://stromtutorial.github.io/linux/steps/, (accessed 12 2021).
[46] B. Hie, E.D. Zhong, B. Berger, B. Bryson, Learning the language of viral evolution and escape, Science 371 (6526) (2021) 284–288.
[47] A. Zwick, J.C. Regier, C. Mitter, M.P. Cummings, Increased gene sampling yields robust support for higher-level clades within Bombycoidea (Lepidoptera), Sys. Entomol. 36 (1) (2011) 31–43.
[48] P.O. Lewis, A genetic algorithm for maximum-likelihood phylogeny inference using nucleotide sequence data, Mol. Biol. Evol. 15 (3) (1998) 277–283.
[49] R.W. Meredith, et al., Impacts of the cretaceous terrestrial revolution and KPg extinction on mammal diversification, Science 334 (6055) (2011) 521–524.
[50] J.R. Cole, et al., The ribosomal database project (RDP-II): sequences and tools for high-throughput rRNA analysis, Nucleic Acids Res. 33 (suppl_1) (2005) D294–D296.
[51] M.W. Chase, et al., Phylogenetics of seed plants: An analysis of nucleotide sequences from the plastid gene rbcl, Ann. the Missouri Botanical Gard. 80 (3) (1993) 528–580.
[52] D.J. Sheskin, HandBook of Parametric and Nonparametric Statistical Procedures, fifth ed., Chapman & Hall/CRC, NY, USA, 2011.
[53] K.N. Khan, M. Hirki, T. Niemi, J.K. Nurminen, Z. Ou, RAPL In action: Experiences in using RAPL for power measurements, ACM Trans. Model. Perform. Eval. Comput. Sys. 2018 (2018) 9, 1–26.
[54] NVIDIA Corporation, Nvidia system management interface, 2016, https://developer.nvidia.com/nvidia-system-management-interface, (accessed 12 2021).
[55] R. Lu, et al., Genomic characterisation and epidemiology of 2019 novel coronavirus: implications for virus origins and receptor binding, Lancet 395 (10224) (2020) 565–574.
[56] H. Tegally, et al., Sixteen novel lineages of SARS-CoV-2 in South Africa, Nature Med. 27 (2021) 440–446.
[57] C.H. Yu, et al., Nucleic acid binding by SAMHD1 contributes to the antiretroviral activity and is enhanced by the GpsN modification, Nature Commun. 12 (2021) 31, 1–14.

Sergio Santander-Jiménez received the Ph.D. degree in Computer Engineering from the University of Extremadura (UEx), Spain, in 2016. He is currently an Assistant Professor of computer organization and design in the Department of Computer and Communications Technologies, UEx. He has authored or co-authored more than 60 publications, also co-organizing multiple international workshops on high-performance bioinformatics. He has edited 3 journal special issues and served as a reviewer for over 30 JCR-indexed journals. His research interests include multiobjective evolutionary computation, high performance computing, and bioinformatics.

Miguel A. Vega-Rodríguez received the Ph.D. degree in Computer Engineering from the University of Extremadura (UEx), Spain, in 2003. He is currently a Full Professor of computer architecture in the Department of Computer and Communications Technologies, UEx. He has authored or co-authored more than 700 publications including journal papers (more than 150 JCR-indexed journal papers), book chapters, and peer-reviewed conference proceedings, for which he got several awards. He has edited more than 20 special issues of JCR-indexed journals. His research interests include parallel and distributed computing, evolutionary computation, bioinformatics, and reconfigurable and embedded computing.

Leonel Sousa received the Ph.D. degree in Electrical and Computer Engineering from the Instituto Superior Técnico (IST), Universidade de Lisboa (UL), Portugal, in 1996. He is currently a Full Professor with UL and a Senior Researcher with the R&D Instituto de Engenharia de Sistemas e Computadores (INESC-ID). He has authored or co-authored more than 250 papers in journals and international conferences, and has edited four special issues of international journals. Professor Sousa is a Distinguished Scientist of the ACM and the recipient of multiple awards. His research interests include VLSI architectures, computer architectures and arithmetic, parallel computing, and signal processing.