The Journal of Supercomputing (2020) 76:9827–9853
https://doi.org/10.1007/s11227-020-03225-x

GPU acceleration of Fitch's parsimony on protein data: from Kepler to Turing

Sergio Santander-Jiménez[1] · Miguel A. Vega-Rodríguez[2] · Antonio Zahinos-Márquez[2] · Leonel Sousa[1]

Published online: 4 March 2020

Abstract

The analysis of complex biological datasets beyond DNA scenarios is gaining increasing interest in current bioinformatics. Particularly, protein sequence data introduce additional complexity layers that impose new challenges from a computational perspective. This work is aimed at investigating GPU solutions to address these issues in a representative algorithm from the phylogenetics field: Fitch's parsimony. GPU strategies are adopted in accordance with the protein-based formulation of the problem, defining an optimized kernel that takes advantage of data parallelism at the calculations associated with different amino acids. In order to understand the relationship between problem sizes and GPU capabilities, an extensive evaluation on a wide range of GPUs is conducted, covering all the recent NVIDIA GPU architectures—from Kepler to Turing. Experimental results on five real-world datasets point out the benefits that imply the exploitation of state-of-the-art GPUs, representing a fitting approach to address the increasing hardness of protein sequence datasets.

Keywords Parallel computing · Hardware accelerators · GPU architectures · Bioinformatics

✉ Sergio Santander-Jiménez
sergio.jimenez@tecnico.ulisboa.pt

Miguel A. Vega-Rodríguez
mavega@unex.es

Antonio Zahinos-Márquez
azahinos@alumnos.unex.es

Leonel Sousa
leonel.sousa@ist.utl.pt

[1] INESC-ID, Instituto Superior Técnico, Universidade de Lisboa, 1000-029 Lisbon, Portugal

[2] Department of Computer and Communications Technologies, University of Extremadura, Avda. de la Universidad s/n, 10003 Cáceres, Spain

1 Introduction

Advances in life sciences have been boosted throughout the years by the proposal and application of computational techniques to address biological problems. As a result, bioinformatics has become one of the most important and challenging areas in the current research context [3]. An especially remarkable issue for bioinformaticians is the hard-to-tackle computational complexity of these problems. More specifically, time-consuming workflows involving analysis, inference, and prediction procedures are often required to process biological data. On top of that, bioinformatics tasks are governed by complex mathematical models and strict statistical validation requirements. It is consequently mandatory the application of advanced algorithmic designs that exploit the computing capabilities of the underlying hardware to minimize execution time and maximize the throughput. High-performance computing is progressively playing a key role in solving these problems, allowing researchers to take advantage of parallelism to accelerate bioinformatics codes. Multiple works have given account of the soundness of these approaches, improving the solution of problems in systems biology, genomics, and other related fields [2].

However, the computationally demanding nature of these problems keeps growing due to the increasing explosion of biological data. As an illustration, next-generation sequencing [15] has led to datasets whose sizes and dimensions exceed in several orders of magnitude the number of cores available in traditional parallel architectures. Along with this, there is nowadays special emphasis on the analysis of complex data beyond classic DNA scenarios [35]. The study of protein data, in particular, has a significant impact on the computational costs of these problems, since the consideration of twenty amino acids instead of four nucleotides imposes new challenges over the data types, the memory access patterns, and the effective exploitation of functional units.

All these complexity factors give as a result a significant rise in the execution time of bioinformatics problems. It is thus required the design and development of parallel algorithms aimed at addressing these issues by accurately exploiting current accelerator devices, particularly the well-established GPUs [31]. Moreover, it is mandatory to carry out the validation of parallel designs over a wide spread of architectures and problem sizes, in order to understand how the proposed solutions take advantage of the offered computing capabilities and their benefits in different evaluation scenarios.

This work is aimed at studying the use of GPUs to improve phylogenetic analyses in protein data. Phylogenetic reconstruction is a key problem in current bioinformatics, since it deals with the processing of biological sequences of organisms to provide hypotheses about their evolutionary history [42]. We will focus on the GPU parallelization of Fitch's algorithm, which provides an implementation of the well-known phylogenetic parsimony function [12]. We will detail a GPU design of this function based on CUDA [43], adapting the kernel to the specific features of protein sequences. In addition, we will undertake an in-depth analysis of the solution over the different GPU architectures proposed by NVIDIA in

recent years: Kepler, Maxwell, Pascal, Volta, and Turing. Five real-world protein datasets (comprising over 600,000 amino acids per sequence) will be employed for experimental purposes, discussing the evolution of computing capabilities in different problem instances. Furthermore, the relevance of the observed parallel results will be validated through comparison with other parallel platforms, namely multicore multiprocessor systems and Intel Xeon Phi co-processors. The contributions of this work are listed below:

- Definition of the first functional GPU design of Fitch's algorithm for protein data, adopting proper data encoding and structures in accordance with the characteristics of protein-based alignments.
- Comprehensive analysis of compute performance and execution time on five CUDA-enabled GPUs: Tesla K40m, GeForce TITAN X, GeForce TITAN Xp, Tesla V100, and GeForce RTX 2080Ti.
- Discussion on the impact of problem sizes in the observed parallel results, pointing out their relationship with the computing capabilities of the analysed GPU devices.
- Validation of the GPU solution through comparisons with alternative programming approaches, parallel platforms, and other authors' tools.

This paper is organized as follows: The next section overviews some of the most relevant works on the application of GPU approaches to bioinformatics, with special emphasis on the particular case of phylogenetics. Section 3 introduces the basis of the problem and explains Fitch's algorithm, while Sect. 4 details the design and implementation of the proposed GPU-based parallel solution. The experimental analysis and assessment of parallel performance are conducted in Sect. 5. Finally, Sect. 6 highlights the conclusions of this paper and future research directions.

2 Related work

Parallel computing and the use of GPUs have played a key role in the acceleration of bioinformatics codes during the last decade [31]. Among others, it can be highlighted the work by Klus et al. [22], who proposed the BarraCUDA method to speed up biological sequence alignments with GPUs. By using their GPU approach, significant temporal gains were attained while also preserving the solution accuracy level of the original serial implementation. Other methods for multiple sequence alignment, such as T-Coffee [6] and ClustalW [18], have also benefited from the parallel processing features of GPUs. In addition, Ohue et al. tackled the protein-protein docking prediction problem by proposing a scalable parallel design, based on OpenMPI and CUDA, designated as MEGADOCK [32]. Jünger et al. [20] combined novel filtering methods and massively parallel processing with GPUs to detect high-order epistasis interactions among genes. In 2018, Quang et al. [34] proposed the application of deep learning techniques on GPUs to efficiently address the motif discovery problem, achieving noticeable improvements over the reference tool MEME, both in terms of execution time and solution quality.

The development of GPU-based approaches has also gained increasing interest as a promising research direction in computational phylogenetics. Martins et al. described in [28] a parallel GPU algorithm to accelerate the computation of phylogenetic distances. By optimizing the employed data structures and memory transactions, the proposal led to accelerations up to 243x in comparison with the serial version. Other works put special emphasis on the parallelization of the phylogenetic likelihood function, especially when integrated into Bayesian analysis frameworks. For example, Ling et al. proposed an improved implementation of MrBayes based on the exploitation of potential parallelism within likelihood computations, allowing the processing of nucleotides sequences eight times faster [25]. Following this research line, Pratas et al. presented alternative implementations for multicore systems, Cell/BE, and GPUs [33], contributing with balanced and scalable approaches to parallelize the computation of partial likelihood arrays in DNA data. Bao et al. studied the use of multiple GPUs simultaneously to conduct compute-intensive likelihood scoring tasks with different granularity [5]. Under the defined multi-GPU approach, speedups up to 278x were attained in a GPU cluster composed of 32 computing nodes. The proposal from Izquierdo-Carrasco et al. in [19] was oriented towards the parallel orchestration of likelihood calculations, proposing a design to minimize data transfers between the host and the GPU. Besides CUDA, other parallel programming models for GPU computing like OpenACC have been applied to accelerate MrBayes [23]. Similarly, different data partitioning schemes have been proposed to maximize the exploitation of GPU threads [26]. In 2019, it was released the version 3.0 of the BEAGLE library [4], which includes new implementations and parallel strategies to improve likelihood computations in hardware accelerators.

It is also possible to find research works that target the application of GPUs, CPUs, and hardware customized accelerators to other well-known phylogenetic algorithms and scoring functions. This is the case of the UPGMA method for reconstructing ultrametric phylogenetic trees [17, 24]. As for the phylogenetic parsimony function, we can highlight the parallel designs proposed by Alachiotis and Stamatakis for CPU and FPGA platforms [1]. Majumder et al. investigated two representative network-on-chip architectures, namely mesh and four-way hierarchical star, to address the breakpoint parsimony phylogeny problem [27]. The proposed architectures implemented innovative approaches to compute breakpoints in terms of solving multiple instances of the classic travelling salesman problem. Other alternative FPGA designs were proposed by Block and Maruyama, who implemented improved approaches for indirect tree length calculations on KINTEX-7 XC7K325T-FF2-900 [7] and incremental tree optimization on VIRTEX-7 XC7VX690T [8]. Moreover, Santander-Jiménez et al. proposed different strategies and implementations for parsimony computation on heterogeneous [37] and GPU-based [38] systems. However, these previous approaches do not support protein data, thus being their practical interest limited to evaluation scenarios involving DNA sequences only. These previous works neither investigate the additional complexity issues introduced by protein-based formulations, nor shed light on how the problem can benefit from the enhanced features of state-of-the-art GPUs. This paper is aimed at addressing these questions by (1) introducing, to the best of our knowledge, the first functional GPU design of the phylogenetic parsimony function for protein data; and (2) undertaking

an exhaustive, in-depth performance evaluation over the different GPU architectures published by NVIDIA in recent years, from Kepler to Turing.

3 Parsimony formulation

Biological data from natural organisms can be processed to investigate the evolutionary events that led to the current biodiversity. Phylogenetic analyses represent the core of this kind of studies, since they allow the reconstruction of evolutionary hypotheses from the observed biological evidence [42]. More specifically, phylogenetic methods take as input a set of N aligned biological sequences $S = \{S_0, S_1, \ldots, S_{N-1}\}$, comprising M characters per sequence. The state values for each character in these sequences are defined in accordance with an alphabet Λ, which corresponds to the amino acid state alphabet in the case of protein-based analyses. Evolutionary hypotheses can then be inferred from this input data, being represented by means of phylogenetic trees $T = (V, E)$, where V refers to the node set and E the edge set. V classifies the organisms in the phylogenetic topology into two groups: (1) terminal nodes, which represent the organisms from the input alignment; and (2) internal nodes, which describe hypothetical ancestral species. The branching patterns in E establish the proposed relationships among the organisms under analysis.

One of the most challenging aspects of the phylogenetic reconstruction problem is given by the fact that the number of possible phylogenetic trees grows exponentially with N [35]. More specifically, the size of the phylogenetic tree space is governed by the double factorial $(2N-5)!!$. Therefore, the task of reconstructing phylogenetic trees usually requires the application of heuristic optimization under the guidance of phylogenetic objective functions. In this work, we are interested in studying the phylogenetic parsimony function, which quantifies the amount of evolutionary change observed in the phylogenetic topology. Parsimony represents one of the most well-known objective functions in the field, playing a key role in the execution of large-scale phylogenetic studies [9, 29] where likelihood-based measurements are difficult to apply. Given a multiple sequence alignment of size $N \times M$ and a candidate phylogenetic hypothesis $T = (V, E)$, the parsimony function computes an integer-valued score $P(T)$ that measures the number of mutation events verified between related nodes at the sequence level:

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

where $(u, v) \in E$ represents the tree branch connecting the nodes $u, v \in V$, $u_i, v_i \in \Lambda$ are the state values at the i-th character for u and v, and $C(u_i, v_i)$ is a cost function for modelling the emergence of mutation between u_i and v_i. Particularly, $C(u_i, v_i) = 1$ if $u_i \neq v_i$, and $C(u_i, v_i) = 0$ otherwise. The parsimony criterion is aimed at finding the simplest evolutionary hypothesis, thus giving more support to the phylogenetic trees that minimize $P(T)$.

One of the most commonly adopted methodologies to compute $P(T)$ is through Fitch's algorithm [12]. The basic idea lies in the computation, for each character ($i = 1$ to M), of ancestral state sets F_i throughout the phylogenetic topology, proceeding from the leaves to the root. Given a terminal node $l \in V$, $F_i(l)$ corresponds to the state value observed at the i-th character of the associated input sequence S_l. For an internal node $u \in V$, $F_i(u)$ is calculated by processing the state sets $F_i(v)$ and $F_i(w)$ of its children $v, w \in V$ as follows:

$$F_i(u) = \begin{cases} F_i(v) \cap F_i(w) & \text{if } F_i(v) \cap F_i(w) \neq \emptyset, \\ F_i(v) \cup F_i(w) & \text{if } F_i(v) \cap F_i(w) = \emptyset. \end{cases} \tag{2}$$

By using the expressions in Eq. 2, it is possible to identify mutation events for determining the parsimony score of the topology. If $F_i(v) \cap F_i(w) \neq \emptyset$, it means that the state value of the parent node u is inherited by both children v and w. Consequently, this evolutionary step does not imply a mutation event. However, if $F_i(v) \cap F_i(w) = \emptyset$ is observed, it implies that at least one of the children nodes v and w has been subject to mutation during the evolutionary process. As a mutation has been detected, the parsimony score of the topology is increased accordingly. For illustration purposes, examples of $P(T)$ calculations under this procedure are provided in Fig. 1. It is worth remarking that Fitch's algorithm includes a second labelling procedure, where final state values for each internal node are computed, starting from the root

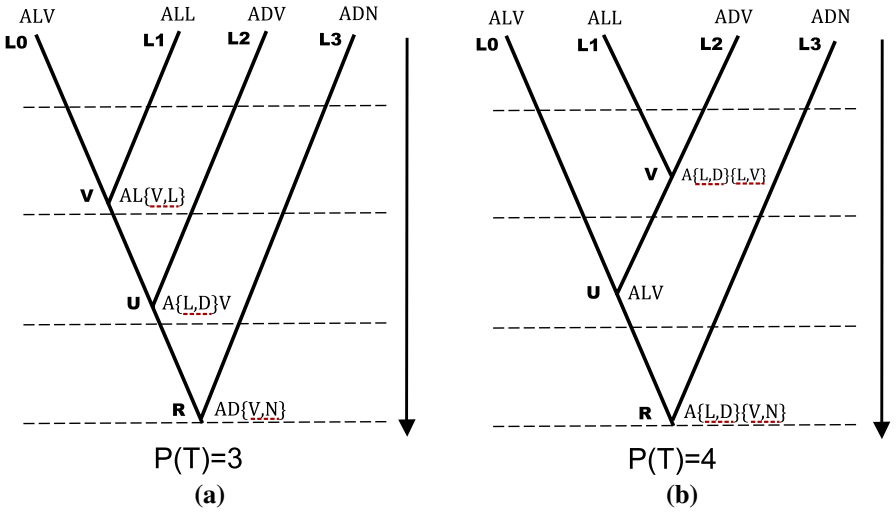

Fig. 1 Representation of parsimony calculations for two phylogenetic topologies. These examples involve protein sequence data from four leaf organisms *L0*, *L1*, *L2*, and *L3*, where A stands for alanine, L for leucine, V for valine, D for aspartic acid, and N for asparagine. For (a), three empty Fitch intersections take place at $F_3(V)(F_3(L0) = V \neq L = F_3(L1))$, $F_2(U)(F_2(V) = L \neq D = F_2(L2))$, and $F_3(R)(F_3(U) = V \neq N = F_3(L3))$. For (b), four empty intersections are identified at $F_2(V)$, $F_3(V)$, $F_2(R)$, and $F_3(R)$. Parsimony will give preference to the hypothesis described in **a**, since it shows less evolutionary changes than **b**

node. We are herein interested in the first procedure, which is the one that performs the core calculations associated with the parsimony function.

From a computational perspective, there are different complexity factors that justify the development of parallel approaches to accelerate these computations. First, the computational complexity of the algorithm when applied from scratch is O(NM), thus depending on both the number of sequences and the number of characters per sequence. Although current datasets are complex from both perspectives, sequence lengths tend to be several orders of magnitude higher than the number of sequences, playing a more important role in the computational costs of the algorithm. On top of that, the consideration of protein data imposes additional issues regarding the codification of sequences, since more complex data types are required to encode the 20 basic amino acid states as well as ambiguous states and the possible combinations required by the union operations of the algorithm. Second, it is important to remark that although indirect calculations can be applied to improve parsimony computations in phylogenetic search algorithms, significant numbers of full instantiations of the parsimony function are still required within those methods aimed at large-scale sequence processing [36, 40]. The major role that the targeted function plays in phylogenetic analyses is also stressed by the needs of providing bootstrap support to the proposed hypotheses [42], thus being required a high number of independent runs to ensure statistical reliability.

Fitch's parsimony follows the idea that each character is subject to evolutionary events individually. Consequently, the calculations for different characters are independent and can be performed simultaneously. The parsimony score can then be calculated from partial parsimony scores for each amino acid in the sequences. This algorithm shows intrinsic data parallelism suitable to be exploited by the enhanced parallel processing capabilities of modern GPUs.

4 GPU parsimony design for protein data

This work examines the GPU parallelization of Fitch's parsimony on protein data. With this purpose in mind, we will define a GPU-oriented parallel design based on the well-established CUDA programming framework [43]. We assume a programming model where two main elements, the CPU host and the GPU accelerator, interact with each other to undertake the definition and processing of parallel tasks. The host side is in charge of specifying, allocating, and initializing the data structures required by these tasks, which are offloaded to the GPU for parallel processing purposes. The GPU carries out computations in a data parallel fashion, with each GPU thread processing one or several pieces of data in accordance with the operations specified by a kernel. Upon termination, the results of the kernel are then transferred from the GPU to the host, carrying out post-processing operations if required.

4.1 GPU memory structures

The key idea in the proposed GPU design lies in the parallel computation of amino acid states for each internal node in the phylogenetic topology. Hence, the

calculations associated with the i-th character are assigned to the i-th GPU thread, which updates its local partial parsimony score whenever mutations are detected. In order to implement the GPU kernel, three main data structures must be defined and allocated accordingly in the GPU memory hierarchy:

1. *Phylogenetic topology*: instead of using a tree-shaped structure, the phylogeny encoding involves an array that codifies information about each internal node. The u-th element of this array represents the u-th internal node under a post-order tree traversal, storing the number of children of u (by means of a short integer) and their identifiers (integers). In these identifiers, the most significant bit is reserved and set to: 1, in case the child refers to an internal node; 0, in case the child is a terminal node. The reasoning behind this design decision lies in defining an optimized way to store the position of each node in the topology, as well as the indexes to the corresponding amino acid (from the sequences for terminal nodes or from the state sets computed by the algorithm for internal nodes). In this design, the topology array will be allocated in the GPU constant memory, which allows faster read accesses throughout the execution of the kernel.
2. *Amino acid sequences*: the second main structure required by the algorithm derives from the input aligned sequences. In the case of DNA sequences, small-sized data types could be used to store character states, since the four nucleotides and their combinations can be encoded with four bits (or five if gaps are codified as a separate state). However, in protein-based analyses, a total of 20 baseline amino acid states and two ambiguous pairs have to be considered, along with their combinations (Fitch's unions), unknown characters, and gaps. Therefore, more complex data types are needed. The proposed design encodes the amino acid sequences as a one-dimensional array of $N \times M$ integers, employing the hexadecimal values referred in Table 1 for codification purposes. This array follows a row-major order organization of the sequences with the aim of allowing adjacent data accesses within a thread group (warp), thus avoiding operations over scattered elements and the consequent impact on memory transfers. Since the sequences represent the largest data structure employed by the kernel, this array will be stored in the GPU global memory. Figure 2 provides a graphical example of the employed topology and sequence encoding.
3. *Shared reduction space*: since each GPU thread will be responsible for computing a local partial parsimony score, it is required the integration of parallel reduction strategies to accumulate the scores computed within a thread block. In our design, we have used the reduction algorithm described in [43] (chapter 12.5), which employs integer structures located at the GPU shared memory to perform these calculations.

4.2 GPU kernel

Algorithm 1 outlines the GPU kernel designed to conduct Fitch's computations on protein data. Along with the topology and sequence arrays mentioned in the previous subsection (designated as *nodes* and *sequences*, respectively), the kernel uses

Table 1 Codification of amino acid states for protein sequences

State	Hex. value	Description	State	Hex. value	Description
A	0x080000	Alanine	R	0x040000	Arginine
B	0x030000	Asparagine or Aspartic acid	N	0x020000	Asparagine
D	0x010000	Aspartic acid	C	0x008000	Cysteine
Z	0x006000	Glutamine or Glutamic acid	Q	0x004000	Glutamine
E	0x002000	Glutamic acid	G	0x001000	Glycine
H	0x000800	Histidine	I	0x000400	Isoleucine
L	0x000200	Leucine	K	0x000100	Lysine
M	0x000080	Methionine	F	0x000040	Phenylalanine
P	0x000020	Proline	S	0x000010	Serine
T	0x000008	Threonine	W	0x000004	Tryptophan
Y	0x000002	Tyrosine	V	0x000001	Valine
?	0x0FFFFF	Unknown character	–	0x100000	Gap

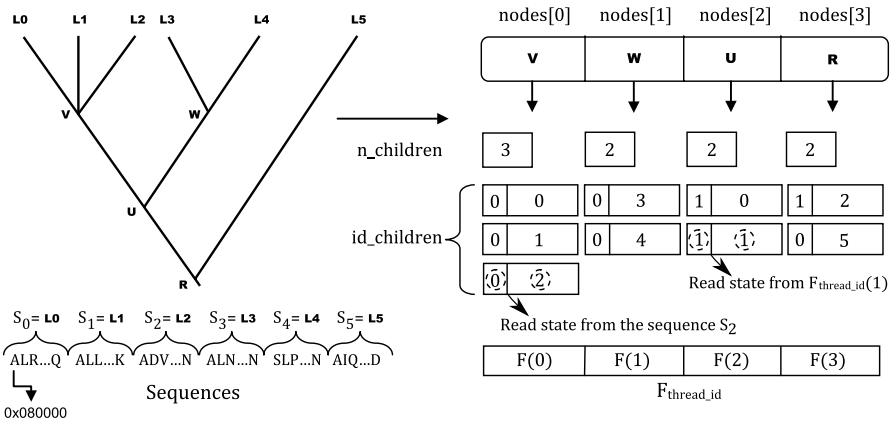

Fig. 2 Representation of data structures: phylogenetic tree encoding and protein sequences. The nodes array is organized according to a post-order tree traversal to ensure data dependencies, while the sequences array follows a row-major order to improve memory accesses within a warp. F_{thread_id} refers to the amino acid states of the internal nodes at the *thread_id*-th character, calculated during the execution of the algorithm

two constant values that represent the number of internal nodes in the topology (*num_inner*) and the number of amino acids per sequence (*seq_length*). The first steps in the kernel involve the retrieval of the GPU thread identifier *thread_id*, and the initialization of the local partial parsimony score $P(T)_{thread\_id}$ and the ancestral amino acid state sets F_{thread_id} (lines 1–3 in Algorithm 1). Afterwards, the kernel proceeds with the main loop of Fitch's algorithm, where each component of F_{thread_id} is computed and the $P(T)_{thread\_id}$ is updated accordingly (lines 4–18). More in detail, this loop processes the topology array to compute, for each amino acid in parallel, the ancestral state $F_{thread\_id}(u)$ of each internal node u, so that the *thread_id*-th

GPU thread will be responsible for conducting Fitch's operations (and, therefore, $P(T)_{thread\_id}$ calculations) for the *thread_id*-th amino acid.

Algorithm 1 Fitch's parsimony GPU kernel for protein data

```
Input Parameters: __constant__ GPU_node* nodes,
    __constant__ int num_inner,
    __global__ int* sequences,
    __constant__ int seq_length
Returns: __global__ int* Global_P(T)
 1: thread_id ← blockIdx.x * blockDim.x + threadIdx.x
 2: P(T)_thread_id ← 0
 3: F_thread_id ← 0
 4: for i = 0 to num_inner-1 do
 5:    node_aa ← 0x1FFFFF
 6:    num_children ← nodes[i].n_children
 7:    for j = 0 to num_children-1 do
 8:       child_id ← nodes[i].id_children[j]
 9:       if child_id & 0x80000000 = 0x80000000 then
10:          child_aa ← F_thread_id[child_id & 0x7FFFFFFF]
11:       else
12:          child_aa ← sequences[seq_length*(child_id & 0x7FFFFFFF)+thread_id]
13:       end if
14:       fitch_op ← node_aa & child_aa /* Fitch's intersection operation */
15:       P(T)_thread_id,node_aa ← (fitch_op=0) ? {P(T)_thread_id++, node_aa | child_aa} :
          {P(T)_thread_id, fitch_op} /* Fitch's union operation in case fitch_op=0 */
16:    end for
17:    F_thread_id[i] ← node_aa
18: end for
19: __shared__ Block_P(T) ← 0
20: Block_P(T)[threadIdx.x] ← P(T)_thread_id
21: Global_P(T)[blockIdx.x] ← Reduction with Shared Memory (Block_P(T))
```

The processing of each internal node begins with the initialization of an auxiliary variable *node_aa*, which is aimed at storing the results of Fitch's operations (line 5). The kernel then obtains the number of child nodes (*num_children* line 6) and carries out the reading of the first child state from F_{thread_id} (most significant bit in the identifier *child_id* = 1, that is, a previously processed internal node, lines 9–10) or from the sequences (most significant bit = 0, that is, a terminal node, lines 11–12). The execution path followed by the threads is the same when encountering this if-else condition, as the scheduled warp operates over the same child node. As previously stated, the remaining information in the child identifier is used to index the position where the child amino acid is located. While F_{thread_id} is a private structure, the *thread_id* value points to the specific character to be read from the corresponding global sequence.

Once retrieved the amino acid state of the current child, Fitch's intersections are conducted with the state value stored in *node_aa* by using a bitwise AND instruction (line 14). Whenever an empty intersection is detected, Fitch's unions are performed through a bitwise OR instruction and $P(T)_{thread\_id}$ is increased (line 15). The algorithm then iterates over the next child node until all of them have been processed. The resulting amino acid state for the current internal node is stored in F_{thread_id}, so

that it can be used when processing its ancestors in the topology. Upon termination of these calculations on the whole topology, the parsimony scores for each thread block are obtained by applying parallel reduction over the computed $P(T)_{thread\_id}$ values (lines 19–21).

4.3 Host side

Prior to the kernel execution at the GPU side, the CPU host must perform a number of operations related to the initialization and transference of the input data. More specifically, the host carries out in a first step the allocation of the required GPU memory (`cudaMalloc`), the initialization of the sequences array, and the conversion of the input phylogenetic topologies from the standard Newick code representation [42] to the proposed topology array encoding. Once the input data are ready, the structures can be transferred from the host to the GPU memory (`cudaMemcpy`). Then, the kernel is instantiated and, upon termination, a final reduction step is performed by the host to obtain the final $P(T)$ score from the parsimony values calculated by each thread block (previously transferred to the host side via `cudaMemcpy`). The implemented host code processes each topology in an iterative way, carrying out the previous operations over the different phylogenies available in the input of the procedure. Since the sequence alignment is a static structure, which maintains the same data for multiple instantiations of the kernel, no additional transfers of this data are needed once the sequence array has been allocated and written in the GPU global memory.

Figure 3 provides insight into the execution time profile of the application. While the kernel contributes to a 60.75% of the overall execution time, the data transfers and operations performed at the host can impose a significant performance penalty (representing a time percentage of 39.25%) on real-world protein data scenarios. In order to minimize the impact of these operations, we can take advantage of the fact that the topology pre-processing and transference tasks can be conducted concurrently with previous instantiations of the kernel. CUDA streams are hence implemented to allow such overlap between host and GPU operations, employing the corresponding asynchronous functions defined in the programming interface (such as `cudaMemcpyAsync` and `cudaMemcpyToSymbolAsync`). In this way, the

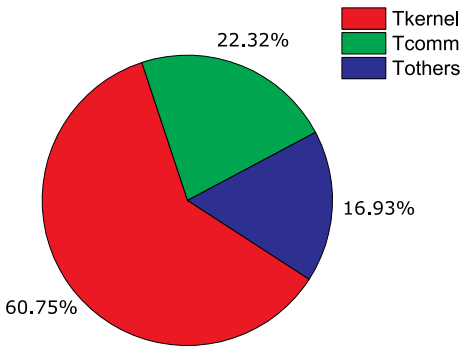

Fig. 3 Average time profile of the application in Tesla V100, where T_{kernel} refers to the kernel execution times, T_{comm} the communication/data transfer times between the host and the GPU, and T_{others} other operations including the initialization of data structures, tree topologies, and final parsimony reductions at the host side

GPU computation of $P(T)$ scores for a phylogeny can be conducted on parallel with the initialization and communication of the next one, with the additional advantage of potentially allowing concurrent execution of different kernel instantiations.

5 Experimental results and performance analysis

This section undertakes the experimental assessment of the GPU approach designed to conduct parsimony calculations on protein data. First, the experimental methodology herein followed is detailed, reporting hardware specifications and the datasets used for evaluation purposes. The validation of the CUDA design, in comparison with other alternative GPU programming approaches, is conducted afterwards. Then, we proceed with the analysis of parallel performance for the GPU architectures under study. Finally, comparisons with different parallel devices and other authors' proposals are presented.

The experiments conducted in this research work involve different real-world protein datasets, each one with different characteristics attending to the number of sequences and characters per sequence. Thus, the evaluation is performed over a range of different, representative problem sizes to understand the implications of the proposed design:

1. M9x607784: *Aspergillus westerdijkiae* protein data, nine sequences containing 607,784 amino acids per sequence [16].
2. M37x29352: *Xylona heveae* protein data, 37 sequences containing 29,352 amino acids per sequence [13].
3. M88x2269: *Thermophilic fungi* protein data, 88 sequences containing 2,269 amino acids per sequence [30].
4. M355x934: *Hemiascomycete yeasts* protein data, 355 sequences containing 934 amino acids per sequence [10].
5. M720x5873: *Bacteria* and *Archaea* protein data, 720 sequences containing 5,873 amino acids per sequence [44].

Thirty-one independent runs were conducted in each experiment, considering 2,000 phylogenetic topologies (generated via bootstrapping) in the input of the procedure. The reasoning behind this number of topologies is given by the fact that it represents the number of full instantiations of the parsimony function involved in large-scale phylogenetic analyses, e.g. under the Rec-I-DCM3 search method [36]. The attained results were tested under the following methodology to provide statistical support to the comparisons [39]. Normality tests under Kolmogorov–Smirnov were first conducted to study if the samples followed a Gaussian distribution. If so, Levene's tests were applied to analyse variance homogeneity. In case the samples met these two previous requirements, ANOVA tests were used to check for statistical significant differences in the comparisons. In the remaining cases, Wilcoxon–Mann–Whitney tests were applied instead. A confidence level of 95% was employed in this statistical analysis.

Special emphasis has been put in this work on the evaluation of the GPU design on a wide range of CUDA-enabled GPU devices, covering the modern architectures from NVIDIA since Kepler. Particularly, we have evaluated five different GPUs: Tesla K40m, GeForce TITAN X, GeForce TITAN Xp, Tesla V100, and GeForce RTX 2080Ti. Specific details on the compute and memory features of these GPUs are provided in Table 2. The hardware setup employed in the experimentation additionally comprises two Intel Xeon CPU E5-2630v3 processors (16 physical cores, 32 execution threads) at 2.40 GHz with 80 GB DDR3 RAM, as well as an Intel Xeon Phi 7120P (61 cores—60 for computations, 240 execution threads) at 1.33 GHz with 16 GB GDDR5 ECC RAM. This system is managed under Ubuntu 14.04LTS, with the GCC 5.5.0 compiler and the NVIDIA CUDA Toolkit 10.1. As for compilation details, we used the `-O3` flag for the host side and the `-gencode=arch,code` flags in accordance with the compute capability and architecture of the evaluated GPU.

5.1 Comparing CUDA with alternative programming approaches

The first step in this experimental analysis consists in the validation of the CUDA design, in comparison with alternative frameworks for GPU programming. In previous research [38], we conducted comparisons between CUDA, OpenCL [21], and OpenACC [11] for the case of DNA sequence data, pointing out the improvements achieved by the CUDA and OpenCL approaches. Since we are herein dealing with a new and more complex problem formulation based on protein sequence data, it is worth conducting such comparisons to confirm the relevance of the adopted approach, also taking into account the enhanced capabilities of state-of-the-art GPU architectures.

With this purpose in mind, we have implemented alternative OpenCL and OpenACC designs of the protein-based parsimony kernel, using NVIDIA OpenCL 1.2 and OpenACC PGI 19.04. The median execution times reported by each design variant, using the Tesla V100 GPU, are reported in the left side of Table 3. Moreover, the right side of this table outlines the P values obtained from the statistical testing of time results. Our experiments confirm that the CUDA design represents the most satisfying approach for all the considered evaluation scenarios, reporting time results in the interval 76.2–780.7 ms. On the other hand, the execution time observed under OpenCL and OpenACC is within the ranges 155.0–821.2 ms (OpenCL) and 110.6–906.6 ms (OpenACC). These results then suggest the improved management and exploitation of GPU resources attained by the optimizations introduced by the CUDA compiler. As a consequence, the CUDA approach leads to speedups up to 2.0x and 1.5x over OpenCL and OpenACC, being the observed time improvements statistically significant in all the datasets under study (P values < 0.05).

An interesting conclusion that can also be drawn from these results is related to the evolution of OpenACC compilers. The comparison of execution times in Table 3 points out that OpenACC is able to achieve similar or better results than OpenCL in the intermediate datasets. More in detail, OpenACC verifies statistically significant improvements in M37x29352 and M88x2269, while the

Table 2 Specifications of the GPUs employed in this work (SM = streaming multiprocessor)

GPU name	GPU architecture	Number of SMs	CUDA cores	Global memory	GPU clock	Memory clock	Memory bus width
Tesla K40m	Kepler	15	2880	12 GB	745 MHz	3004 MHz	384-bit
TITAN X	Maxwell	24	3072	12 GB	1076 MHz	3505 MHz	384-bit
TITAN Xp	Pascal	30	3840	12 GB	1582 MHz	5705 MHz	384-bit
Tesla V100	Volta	80	5120	16 GB	1380 MHz	877 MHz	4096-bit
RTX 2080Ti	Turing	68	4352	11 GB	1545 MHz	7000 MHz	352-bit

Table 3 Comparisons of execution time (T_{exec}, in ms) and statistical evaluation of CUDA, OpenCL, and OpenACC in protein data (GPU = Tesla V100)

Dataset	T_{exec}			*P* values CUDA		*P* values OpenCL
	CUDA	OpenCL	OpenACC	vs. OpenCL	vs. OpenACC	vs. OpenACC
M9x607784	**120.023**	*163.354*	177.253	2.84E−06	2.84E−06	1.98E−05
M37x29352	**76.221**	155.029	*110.563*	2.84E−06	2.84E−06	2.84E−06
M88x2269	**115.746**	184.009	*148.121*	2.84E−06	2.84E−06	2.84E−06
M355x934	**339.394**	*401.966*	402.294	2.84E−06	2.84E−06	0.699
M720x5873	**780.667**	*821.191*	906.622	2.84E−06	2.84E−06	5.67E−06

Bold values refer to the best values observed in the comparisons; italics refer to the second best values

differences in M355x934 are not significant (*P* value = 0.70). These results suggest that the advances in OpenACC compilers and the increasing support to the directive-driven approach are allowing this programming model to represent a real and competitive alternative for GPU programming. As a reference, the comparisons conducted in [38] for DNA data pointed out a more negative scenario for OpenACC, which in its 17.4 release was outperformed by OpenCL in all the datasets under study. Nevertheless, OpenCL still represents a viable approach when it comes to the processing of more complex protein instances, in accordance with the results observed for M9x607784 (longest sequence length) and M720x5873 (highest number of sequences).

It is also worth highlighting the suitability of the proposed approach, based on CUDA and NVIDIA GPUs, with regard to OpenCL-based GPU solutions from other vendors. In order to illustrate it, Table 4 presents the kernel times reported by OpenCL (ROCm 2.7) on an AMD Vega Frontier Edition GPU. This GPU has been compared with the alternative architecture from NVIDIA, TITAN Xp, also by using OpenCL to ensure fair comparisons under the same programming framework. The results reported in Table 4 give account of the improved OpenCL kernel times reported by the TITAN Xp in all the datasets under evaluation. Taking into account that the CUDA version leads to average time improvements of 11.6% over OpenCL on TITAN Xp, these results further support the relevance of the proposed CUDA-oriented solution.

Table 4 Comparison of OpenCL kernel times (2,000 instantiations, in ms) on alternative GPU platforms: AMD Vega Frontier Edition and NVIDIA TITAN Xp

Dataset	Vega Frontier	TITAN Xp
M9x607784	316.492	**278.790**
M37x29352	114.484	**100.866**
M88x2269	218.572	**161.528**
M355x934	849.820	**529.034**
M720x5873	1528.068	**1140.276**

Bold values refer to the best values observed in the comparisons

5.2 GPU performance evaluation

A key aspect of this experimental study lies in examining the performance of different generations of GPU devices when dealing with real-world protein data. With this purpose in mind, we undertake next the evaluation of the proposed design on a range of CUDA-enabled GPUs covering modern NVIDIA architectures, from Kepler to the most recent Turing one. Along with the registered execution times, a problem-specific performance indicator, named GigaFitch Operations per Second (GFOS), has been employed in the analysis. This metric provides a compute performance measurement based on the number of Fitch operations conducted by the parsimony scoring function:

$$\text{GFOS} = \left[\left(\sum_{u \in V} (\text{Degree}(u) - 1) * M + P(T) \right) / t \right] / 10^9, \tag{3}$$

where Degree(u) represents the degree of an internal node $v \in V$, M the number of amino acids per sequence, $P(T)$ the parsimony value of the input phylogeny, and t the kernel execution time. The expression $\sum_{u \in V}(\text{Degree}(u) - 1) * M$ accounts for the amount of intersection operations, taking into account the presence of potential polytomies in the processed phylogenetic topology [42]. On the other hand, the parsimony score $P(T)$ is equivalent to the number of union operations issued upon detection of mutations.

The median execution times and GFOS attained by the considered GPU devices are shown in Table 5 (for Tesla K40m, TITAN X, and TITAN Xp) and Table 6 (Tesla V100 and RTX 2080Ti). Moreover, the P values retrieved from the statistical evaluation of execution times are detailed in Table 7. On analysing Tables 5 and 6, it can be observed the evolution of computational performance in accordance with the characteristics of the input sequences. Particularly, the sequence length is the factor that has the most impact on the achieved parallel results. For the case of Tesla K40m, the attained performance varies from 2.4 GFOS (in M355x934) to 30.8 GFOS (M9x607784), being these differences more remarkable as we move to more recent GPU architectures. More specifically, median GFOS values of 146.5 and 112.5 can be achieved by using Tesla V100 and RTX 2080Ti GPUs when

Table 5 Performance evaluation: execution times (T_{exec} in ms) and GigaFitch operations per second (GFOS) for Tesla K40, TITAN X, and TITAN Xp

Dataset	Tesla K40m		TITAN X		TITAN Xp	
	T_{exec}	GFOS	T_{exec}	GFOS	T_{exec}	GFOS
M9x607784	571.475	30.760	339.576	51.767	249.577	70.434
M37x29352	153.566	30.526	106.457	44.034	104.863	44.704
M88x2269	206.525	4.087	153.017	5.516	149.683	5.639
M355x934	605.612	2.358	513.738	2.779	391.492	3.647
M720x5873	1375.130	12.930	1086.922	16.359	789.379	22.525

Table 6 Performance evaluation: execution times (T_{exec} in ms) and GigaFitch operations per second (GFOS) for Tesla V100 and RTX 2080Ti

Dataset	Tesla V100		RTX 2080Ti	
	T_{exec}	GFOS	T_{exec}	GFOS
M9x607784	**120.023**	**146.462**	*156.294*	*112.473*
M37x29352	*76.221*	*61.502*	**70.023**	**66.946**
M88x2269	*115.746*	*7.292*	**88.518**	**9.535**
M355x934	*339.394*	*4.207*	**291.458**	**4.899**
M720x5873	*780.667*	*22.776*	**649.008**	**27.397**

Bold values refer to the best values observed in the comparisons; italics refer to the second best values

Table 7 Statistical testing of execution times (P values) from the GPU evaluation

M9x607784				
GPU device	vs. TITAN X	vs. TITAN Xp	vs. Tesla V100	vs. RTX 2080Ti
Tesla K40m	2.84E–06	2.84E–06	2.84E–06	2.84E–06
TITAN X		2.84E–06	2.84E–06	2.84E–06
TITAN Xp			2.84E–06	2.84E–06
Tesla V100				2.84E–06
M37x29352				
Tesla K40m	2.84E–06	2.84E–06	2.84E–06	2.84E–06
TITAN X		0.116	2.84E–06	2.84E–06
TITAN Xp			2.84E–06	2.84E–06
Tesla V100				3.94E–04
M88x2269				
Tesla K40m	5.67E–06	2.84E–06	2.84E–06	2.84E–06
TITAN X		0.023	2.84E–06	2.84E–06
TITAN Xp			8.51E–05	8.51E–05
Tesla V100				2.84E–06
M355x934				
Tesla K40m	2.84E–06	2.84E–06	2.84E–06	2.84E–06
TITAN X		2.84E–06	2.84E–06	2.84E–06
TITAN Xp			7.07E–05	2.84E–06
Tesla V100				3.40E–05
M720x5873				
Tesla K40m	2.84E–06	2.84E–06	2.84E–06	2.84E–06
TITAN X		2.84E–06	2.84E–06	2.84E–06
TITAN Xp			6.63E–03	2.84E–06
Tesla V100				2.84E–06

performing parsimony calculations over protein sequences with more than 600,000 characters. This dataset represents a scenario in which an intensive exploitation of compute resources can be achieved, in comparison with instances like M355x934 and M88x2269 in which the number of CUDA cores exceeds the length of the comprised sequences.

On comparing across different GPUs, it can be verified in first instance the effect of the increased number of cores and the improvements in basic arithmetic latencies in Maxwell (TITAN X) and Pascal (TITAN Xp) over the baseline Kepler architecture. Table 5 shows how the execution times evolve from 153.6–1375.1 ms (K40m) to 106.5–1086.9 ms (TITAN X) and 104.9–789.4 ms (TITAN Xp), thus leading to speedups up to 1.7x for TITAN X and 2.3x for TITAN Xp. The comparison between

the two TITAN GPUs also reveals diverging behaviour for different datasets. Although TITAN Xp provides more satisfying performance in overall terms, TITAN X manages to report closer results in the case of intermediate datasets. In fact, the statistical evaluation reveals that the Maxwell and Pascal GPUs obtain statistically comparable times in M37x29352, with a P value $= 0.12 > 0.05$. Nevertheless, the computing features of TITAN Xp, e.g. in terms of operational frequencies, play a more relevant role when complex alignments, attending to the number of sequences and lengths, are considered, in accordance with the results obtained for M9x607784 (70.4 vs. 51.8 GFOS) and M720x5873 (22.5 vs. 16.4 GFOS).

Moving on to the most recent GPUs, Table 6 shows that the Volta and Turing architectures represent a step forward in the processing of complex protein alignments. In comparison with the previous GPU generations, both Tesla V100 and RTX 2080Ti manage to attain statistically significant improvements in all the datasets under study (P values < 0.05, as shown in Table 7). Moreover, our results denote how different problem instances can take advantage of the computing capabilities of these two GPUs. In the case of M37x29352, M88x2269, M355x934, and M720x5873, the most satisfying performance in the comparison was attained by the RTX 2080Ti GPU. That is, protein alignments with low or moderate parallelism potential benefit the most from the architectural features of this GPU. On the other hand, the dataset with the largest sequence length (M9x607784) represents a more fitting scenario for Tesla V100, which attained a GFOS performance improvement of 30% over RTX 2080Ti. When dealing with very large amino acid sequences, Tesla V100 offers several advantages, not only related to the increased number of compute resources (5,120 CUDA cores distributed over 80 streaming multiprocessors) but also in terms of memory features. For example, this GPU counts with a 4096-bit memory bus width that can be useful to improve memory transfers on large amounts of data, thus providing enhancements regarding memory bandwidth. These features benefit the processing of the referred problem instance, with over 600K characters per sequence, for parsimony computations.

A comparison of the mean GFOS from each GPU is represented in Fig. 4. In detail, the leading GPUs in the comparison, Tesla V100 and RTX 2080Ti, obtain performance improvements of 200.3%/174.3% over Tesla K40m, 101.1%/83.7%

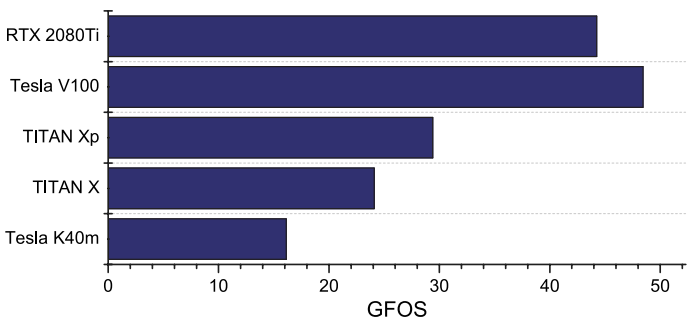

Fig. 4 Comparison of GFOS from the GPU devices under study (mean values for the different problem instances herein evaluated)

over TITAN X, and 64.8%/50.6% over TITAN Xp. It is important to remark that our results confirm the processing issues that the study of protein data imposes on real-world phylogenetic analyses. In the case of DNA sequences [38], a TITAN X with Maxwell architecture was enough to achieve satisfying parallel performance in the range of 140 GFOS. However, the results herein presented reveal that this GPU attains at most 51.8 GFOS when dealing with protein data. Our evaluation shows that the adoption of state-of-the-art GPUs represents a fitting solution to handle such increased hardness, taking advantage of the advanced computing features of these devices.

5.2.1 Power and energy consumption

When analysing GPU approaches, an important question lies in understanding the energy efficiency that can be attained in the considered architectures. With this purpose in mind, Table 8 introduces an analysis of power and energy consumption, using for reference purposes the best performing GPUs identified in the performance evaluation (Tesla V100 and RTX 2080Ti). The power results presented in this table (maximum peak power and average power) have been measured by using the gpowerSAMPLER tool from [14].

It can be observed from the results in Table 8 that among the considered modern GPUs, the Tesla V100 represents a more suitable choice from an energy consumption perspective. More specifically, the average power required by this GPU in the analysed protein datasets lies within the interval 47.0W–107.4W. On the other hand, the RTX 2080Ti demands increased power requirements (average values within 66.6W–116.9W) to conduct the experimentation. The difference between these two GPUs becomes more noticeable in the dataset with the largest execution time requirements (M720x5873), as illustrated in Fig. 5. In fact, this figure depicts how the power demanded by the RTX 2080Ti counterbalances the improvement attained in execution time for the datasets with low or moderate sequence lengths, thus impacting the overall energy results obtained in comparison with Tesla V100 (up to 75.8 J vs. 51.4 J).

As a consequence, the V100 GPU verifies more satisfying energy results not only in its best-case scenario from a time performance perspective (M9x607784), but

Table 8 Power and energy evaluation: maximum peak power, average power during execution, and total energy per dataset for the Tesla V100 and RTX 2080Ti GPUs

Dataset	Max (Power) W		Average (Power) W		Energy J	
	Tesla V100	RTX 2080Ti	Tesla V100	RTX 2080Ti	Tesla V100	RTX 2080Ti
M9x607784	118.269	118.388	107.448	110.054	**12.896**	17.201
M37x29352	81.437	87.262	72.237	80.045	**5.506**	5.605
M88x2269	49.896	69.475	47.036	66.588	**5.444**	5.894
M355x934	53.030	83.910	50.408	79.020	**17.108**	23.031
M720x5873	68.318	137.746	65.832	116.856	**51.393**	75.841

Bold values refer to the best values observed in the comparisons

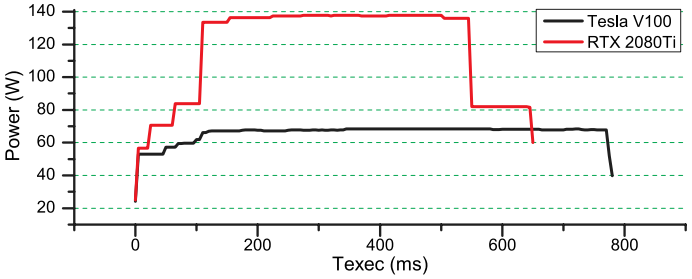

Fig. 5 Power consumption for the dataset with the largest execution time (M720x5873)

also in the remaining problem instances herein examined. The reported time-energy tradeoffs can therefore represent a potential discerning factor for the choice of suitable GPUs to undertake parsimony calculations in real-world scenarios, according to the specifications and desired priorities.

5.3 Comparison with multiprocessors and other co-processors

In order to make a relative evaluation of the performance gains attained by the GPU design, comparisons with other parallel computing platforms are undertaken. In particular, we have considered results for the Intel Xeon E5-2630v3 multicore multiprocessor system and the Intel Xeon Phi 7120P co-processor. For comparison purposes, protein-oriented parsimony kernels for these devices have been developed under OpenCL, due to its device heterogeneity support and the significant results observed in this kind of platforms [38]. CPU and Xeon Phi kernels have been implemented by adopting different strategies to benefit the architectural features of these parallel devices, such as coarse-grained computations (*M*/*NT* characters processed per thread, where *NT* is the number of supported execution threads), memory layouts (e.g. column-major order in the sequence array), and SIMD support (vector data types comprising up to 16 amino acids).

Table 9 introduces the median execution times and GFOS reported by the Xeon multiprocessor and the Xeon Phi co-processor, in comparison with the

Table 9 Performance evaluation: comparisons of execution times (T_{exec} in ms) and GigaFitch operations per second (GFOS) with multiprocessors (2×Xeon CPU E5-2630v3) and co-processors (Xeon Phi 7120P)

Dataset	Intel Xeon CPU		Intel Xeon Phi		Tesla V100		RTX 2080Ti	
	T_{exec}	GFOS	T_{exec}	GFOS	T_{exec}	GFOS	T_{exec}	GFOS
M9x607784	770.298	22.821	1261.752	13.932	**120.023**	**146.462**	*156.294*	*112.473*
M37x29352	375.203	12.494	897.223	5.225	*76.221*	*61.502*	**70.023**	**66.946**
M88x2269	318.684	2.648	437.789	1.928	*115.746*	*7.292*	**88.518**	**9.535**
M355x934	599.949	2.380	678.083	2.106	*339.394*	*4.207*	**291.458**	**4.899**
M720x5873	2024.504	8.783	2136.909	8.321	*780.667*	*22.776*	**649.008**	**27.397**

Bold values refer to the best values observed in the comparisons; italics refer to the second best values

Table 10 Statistical testing of execution times (*P* values) for the comparisons with multiprocessors and co-processors

Dataset	*P* values (Tesla V100)		*P* values(RTX 2080Ti)	
	vs. Intel Xeon CPU	vs. Intel Xeon Phi	vs. Intel Xeon CPU	vs. Intel Xeon Phi
M9x607784	5.67E−06	5.67E−06	5.67E−06	5.67E−06
M37x29352	5.67E−06	5.67E−06	5.67E−06	5.67E−06
M88x2269	5.67E−06	5.67E−06	5.67E−06	5.67E−06
M355x934	5.67E−06	5.67E−06	5.67E−06	5.67E−06
M720x5873	5.67E−06	5.67E−06	5.67E−06	5.67E−06

Table 11 Performance evaluation: GPU speedups over the serial version, multiprocessors (2×Xeon CPU E5-2630v3) and co-processors (Xeon Phi 7120P)

Dataset	Speedup with Tesla V100			Speedup with RTX 2080Ti		
	Serial	Xeon CPU	Xeon Phi	Serial	Xeon CPU	Xeon Phi
M9x607784	**389.534x**	**6.418x**	**10.513x**	299.136x	4.929x	8.073x
M37x29352	196.914x	4.923x	11.771x	**214.345x**	**5.358x**	**12.813x**
M88x2269	28.771x	2.753x	3.782x	**37.620x**	**3.600x**	**4.946x**
M355x934	18.595x	1.768x	1.998x	**21.653x**	**2.058x**	**2.327x**
M720x5873	106.173x	2.593x	2.737x	**127.712x**	**3.119x**	**3.293x**

Bold values refer to the best values observed in the comparisons

results obtained by the GPU design on Tesla V100 and RTX 2080Ti. Moreover, the *P* values returned by the statistical testing of time samples are provided in Table 10. As pointed out in Table 9, the median execution times on the multiprocessor platform are within the interval 318.7–2024.5 ms, while the times from the Intel Xeon Phi vary from 437.8 to 2136.9 ms. On comparing these two alternatives (Xeon CPU and Xeon Phi), the multiprocessor setup reports better results in the datasets M9x607784 and M37x29352 (verifying a compute performance of 22.8 and 12.5 GFOS), while similar performance is attained in the instance with the highest number of sequences, M720x5873.

On the other hand, the GPU alternatives verify the most significant parallel results in all the problem sizes herein analysed, with statistically significant time improvements as suggested by the *P* values < 0.05 in Table 10. In order to provide further insight into this, Table 11 introduces the speedups observed over the Xeon CPU, in both serial and parallel multiprocessor versions, and the Xeon Phi. Effective accelerations of 6.4x/10.5x (for M9x607784), 5.4x/12.8x (M37x29352), 3.6x/4.9x (M88x2269), 2.1x/2.3x (M355x934), and 3.1x/3.3x (M720x5873) are achieved with regard to the alternative parallel platforms herein considered. Finally, the representation of mean GFOS in Fig. 6 gives account of performance improvements of 393.1%/668.7% (on Tesla V100) and 350.4%/602.1% (on RTX

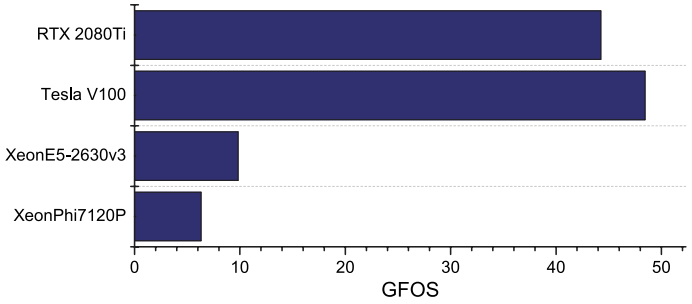

Fig. 6 Comparison of GFOS with multiprocessors and co-processors (mean values for the different problem instances herein evaluated)

2080Ti), thus suggesting the improved behaviour shown by the GPU devices in the considered experimental scenarios.

Different factors must be taken into account to assess the importance of the attained speedups. Among them, one particularly interesting point lies in the diverse nature of the protein datasets under evaluation. The reported results refer to practical real-world problem instances, with divergent available parallelism and bounds that may not match the characteristics of ideal GPU workloads. Furthermore, protein alignments and the corresponding data layouts introduce additional complexity issues that can affect, in the shape of potential overheads, the intrinsic constraints of co-processing architectures (e.g. host–device interactions). Considering the additional difficulties of these computation scenarios, our results denote the practical, performance-wise interest of the proposed GPU designs even in datasets with limited parallelism, reporting significant speedups over other platforms and parallel alternatives.

In order to examine the added value contributed by modern GPUs in this problem, Table 12 presents comparisons of performance per dollar among the considered hardware platforms. Columns 2–4 in Table 12 compare GFOS/$ values across

Table 12 Scaled GFOS per dollar for each hardware platform, using commercial prices from 2019. For each dataset, the best performing GPU device is employed for comparison purposes (Tesla V100 for M9x607784, RTX 2080Ti for the remaining ones)

Dataset	2×Xeon CPU	Xeon Phi	GPU (V100 or 2080Ti)	Observed GFOS/$	
	(1145.99$)	(699.99$)	(7276.99$ / 1299.99$)	GPU Improvement	
	GFOS/$	GFOS/$	GFOS/$	vs. Xeon CPU	vs. Xeon Phi
M9x607784	*1.991*	1.990	**2.013**	**1.011x**	**1.012x**
M37x29352	*1.090*	0.746	**5.150**	**4.725x**	**6.903x**
M88x2269	0.231	*0.275*	**0.733**	**3.173x**	**2.665x**
M355x934	0.208	*0.301*	**0.377**	**1.813x**	**1.252x**
M720x5873	0.766	*1.189*	**2.107**	**2.751x**	**1.772x**

Bold values refer to the best values observed in the comparisons; italics refer to the second best values

devices, while columns 5 and 6 denote the improvement achieved in GFOS/$ when using GPUs. According to the attained results, the GPU approaches lead to the most satisfying behaviour per invested dollar in overall terms. Focusing first on the dataset M9x607784, it can be concluded that the performance gains obtained with the best GPU (Tesla V100) over the 2×Xeon CPU and Xeon Phi alternatives correspond to the required monetary investment. On the other hand, if we consider the second best performing GPU instead (the commercial RTX 2080Ti), GFOS/$ improvements around 4.4x are accomplished over the Xeon multiprocessor and the Xeon Phi co-processor in M9x607784. For the remaining protein datasets, the RTX 2080Ti represented the most satisfying alternative, leading to GFOS/$ improvements up to 4.7x (over CPU) and 6.9x (over Phi). Even in the dataset with the lowest sequence length (M355x934), this GPU also reported added value with regard to the other devices herein considered.

All in all, these comparisons support the conclusion that the use of accurate GPU designs represents a suitable solution to conduct parsimony computations on complex protein alignments. The experimental analysis conducted on different problem sizes points out that statistically significant improvements can be achieved, not only in the case of problem instances with large data parallelism but also under shorter sequence length constraints.

5.4 Comparison with other approaches

Research works on the acceleration of phylogenetic parsimony and, particularly, Fitch's algorithm have fundamentally focused on adapting its calculations to the particular case of DNA alignments [1]. Therefore, it can be stated that the approach herein presented represents, to the best of our knowledge, the first functional GPU parallel design for processing protein sequences. Despite the unavailability of other open-source parallel implementations for this type of data, the literature provides some proof-of-concept methods that can be employed for comparison purposes. For example, Vazquez-Ortiz et al. [41] introduced a benchmark tool that performs 32-bit parsimony calculations on simulated data. Although this tool does not have support

Table 13 Comparisons of execution times (T_{exec} in ms) and speedups observed over the benchmarking tool from Vazquez-Ortiz et al. [41]. 'Unav.' refers to executions where the reference tool reported segmentation faults

Dataset	Tesla V100			RTX 2080Ti		
	T_{exec} [41]	T_{exec} (proposal)	Speedup	T_{exec} [41]	T_{exec} (proposal)	Speedup
M9x607784	1484.360	120.023	**12.367x**	1795.936	156.294	**11.491x**
M37x29352	375.239	76.221	**4.923x**	359.462	70.023	**5.133x**
M88x2269	352.168	115.746	**3.043x**	320.906	88.518	**3.625x**
M355x934	663.120	339.394	**1.954x**	613.920	291.458	**2.106x**
M720x5873	Unav.	780.667	**inf**	Unav.	649.008	**inf**

Bold values refer to the best values observed in the comparisons

for real-world alignments, it can be specified as input parameters the dimensions of the problem size to be benchmarked (number of sequences and sequence length).

We have conducted comparisons with this tool by configuring it according to the dimensions of the five datasets analysed in this work. Table 13 presents the median execution times of the tool, considering 31 independent runs per experiment, on Tesla V100 (column 2 in Table 13) and RTX 2080Ti (column 5). Columns 3–4 and 6–7 refer to the median execution times of our GPU proposal, along with the speedups attained over the benchmarking tool. It can be observed that the GPU design proposed in this work leads to improved execution time in all the considered computing scenarios. In the case of problem sizes governed by large sequences lengths, speedups of 12.4x and 11.5x can be effectively attained by using the strategies implemented in our GPU design. As for problem sizes with high number of sequences, it is worth pointing out that the reference tool was not able to analyse alignments like M720x5873, since it is restricted to a maximum of 512 sequences. On the other hand, our proposal successfully undertakes the processing of this kind of datasets, thus covering a representative range of problem instances usually tackled in real-world biological analyses.

This comparative assessment therefore confirms the improved parallel performance reported by the proposed GPU solution. In conclusion, our results give account of the enhanced processing capabilities attained by the combination of efficient design strategies and modern GPU architectures, thus representing a suitable approach to address the increasing hardness of protein-based phylogenetic studies.

6 Conclusions and future research work

The efficient processing of complex biological datasets represents one of the current major concerns in applied high-performance computing. Particularly, the consideration of protein sequences has imposed new challenges in the development of parallel bioinformatics workflows, due to the hard-to-tackle complexity factors associated with this type of biological data. This work has investigated the GPU parallelization of the phylogenetic Fitch's parsimony on protein data scenarios. Keeping in mind the computational issues that arise from the use of protein data, strategies to encode and perform protein-based parsimony calculations were proposed, accurately mapping the required data structures to the GPU memory hierarchy. Under the assumption that each character in the sequences is subject to separate evolutionary events, the proposed GPU design operates by assigning the processing of different amino acids to different GPU threads. In this way, partial parsimony scores can be calculated and then combined by applying block-based parallel reductions.

One of the main goals of this research work consisted in examining the behaviour of the proposed design on a wide range of CUDA-enabled GPU devices, covering the different modern microarchitectures published by NVIDIA. With this purpose in mind, we performed an in-depth experimental evaluation on five GPUs, namely Tesla K40m (Kepler architecture), TITAN X (Maxwell), TITAN Xp (Pascal), Tesla V100 (Volta), and RTX 2080Ti (Turing). Moreover, five real-world protein datasets were employed in the experimentation, with the aim of covering representative

problem sizes encountered in current phylogenetic analyses. The validation of the CUDA design over other alternatives for GPU programming revealed that the CUDA-based approach provided the most satisfying performance in terms of execution time, yet the advances in directive-based compilers are progressively closing the gap with regard to other programming models. Furthermore, the evaluation of time and compute performance, for the different GPU devices considered in this work, confirmed the challenging nature of the protein-based formulation of the problem, being required the use of recent, state-of-the-art GPU architectures to tackle it. The evaluation also gave account of the relationship between GPU computational capabilities and problem sizes. More specifically, datasets with long sequence lengths (> 600,000 amino acids) benefited the most from the advanced architectural features of Tesla V100, while the remaining datasets were satisfactorily handled by using a commercial RTX 2080Ti (at the expense of higher power consumption requirements). Finally, the relevance of the attained results was examined by performing comparisons with other parallel platforms and parsimony benchmarking tools. The obtained results suggested that GPU computing represents a fitting solution to conduct in an efficient way the processing of challenging protein sequence alignments.

Future research is aimed at designing advanced algorithmic strategies to undertake phylogenetic analyses in compute-hungry, large-scale multiobjective contexts. The main idea lies in the integration of different layers of parallelism (distributed memory, shared memory devices, and hardware accelerators) to take full advantage of the capabilities of current cluster infrastructures. A proposal oriented towards computing nodes with homogeneous characteristics will be initially investigated, considering different design strategies to minimize idle times and execution dependencies among the involved processing components. We will study in a second step the definition of load balancing mechanisms to allow efficient exploitation of heterogeneous nodes, with different CPUs, GPUs, and other co-processors. In this context, a promising direction lies in the evaluation of advanced reconfigurable and adaptable architectures, such as the Alveo U Data Center accelerators [45], in complex phylogenetic spaces and sequence data. Multi-GPU and multi-accelerator implementations with different granularity (coarse-grained distribution of independent phylogenies and fine-grained concurrent subtree processing) will be investigated in the pursuit of maximized performance. Finally, the application of multi-level parallel designs to other bioinformatics problems will be explored.

Acknowledgements This work was partially funded by the AEI (State Research Agency, Spain) and the ERDF (European Regional Development Fund, EU), under the contract TIN2016-76259-P (PROTEIN Project), as well as Portuguese national 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). Sergio Santander-Jiménez is supported by the Post-Doctoral Fellowship from FCT under Grant SFRH/BPD/119220/2016.

Compliance with ethical standards

Conflict of interest The authors declare that they have no conflict of interest.

References

1. Alachiotis N, Stamatakis A (2011) FPGA acceleration of the phylogenetic parsimony kernel? In: Proceedings of FPL 2011. IEEE, pp 417–422
2. Aluru S, Jammula N (2014) A review of hardware acceleration for computational genomics. IEEE Des Test 31(1):19–30
3. Attwood TK, Pettifer SR, Thorne D (2016) Bioinformatics challenges at the interface of biology and computer science: mind the gap. Wiley, Oxford
4. Ayres DL et al (2019) BEAGLE 3: improved performance, scaling, and usability for a high-performance computing library for statistical phylogenetics. Syst Biol 68:1052–1061. https://doi.org/10.1093/sysbio/syz020
5. Bao J, Xia H, Zhou J, Liu X, Wang G (2013) Efficient implementation of MrBayes on multi-GPU. Mol Biol Evolut 30(6):1471–1479
6. Blazewicz J, Frohmberg W, Kierzynka M, Wojciechowski P (2013) G-MSA—a GPU-based, fast and accurate algorithm for multiple sequence alignment. J Parallel Distrib Comput 73(1):32–41
7. Block H, Maruyama T (2014) An FPGA hardware acceleration of the indirect calculation of tree lengths method for phylogenetic tree reconstruction. In: Proceedings of FPL 2014. IEEE, pp 1–4
8. Block H, Maruyama T (2017) An FPGA hardware implementation approach for a phylogenetic tree reconstruction algorithm with incremental tree optimization. In: Proceedings of FPL 2017. IEEE, pp 1–8
9. Bouktila D, Khalfallah Y, Habachi-Houimli Y, Mezghani-Khemakhem M, Makni M, Makni H (2014) Large-scale analysis of NBS domain-encoding resistance gene analogs in triticeae. Genet Mol Biol 37(3):598–610
10. Dias PJ, Sá-Correia I (2013) The drug:H+ antiporters of family 2 (DHA2), siderophore transporters (ARN) and glutathione:h+antiporters (GEX) have a common evolutionary origin in hemiascomycete yeasts. BMC Genom 14(901):1–22
11. Farber R (2017) Parallel programming with OpenACC, 1st edn. Morgan Kaufmann Publishers, Cambridge
12. Fitch W (1972) Toward defining the course of evolution: minimum change for a specific tree topology. Syst Zool 20(4):406–416
13. Gazis R et al (2016) The genome of Xylona heveae provides a window into fungal endophytism. Fungal Biol 120(1):26–42
14. Guerreiro J, Ilic A, Roma N, Tomás P (2018) GPGPU power modelling for multi-domain voltage-frequency scaling. In: Proceedings of IEEE HPCA 2018. IEEE, pp 530–538
15. Goodwin S, McPherson JD, McCombie WR (2016) Coming of age: ten years of next-generation sequencing technologies. Nat Rev Genet 17(1):333–351
16. Han X, Chakrabortti A, Zhu J, Liang Z, Li J (2016) Sequencing and functional annotation of the whole genome of the filamentous fungus *Aspergillus westerdijkiae*. BMC Genom 17(633):1–14
17. Hua GJ, Hung CL, Lin CY, Wu FC, Chan YW, Tang CY (2017) MGUPGMA: a fast UPGMA algorithm with multiple graphics processing units using NCCL. Evolut Bioinform 13:1–7
18. Hung CL, Lin YS, Lin CY, Chung YC, Chung YF (2015) CUDA ClustalW: an efficient parallel algorithm for progressive multiple sequence alignment on multi-GPUs. Comput Biol Chem 58:62–68
19. Izquierdo-Carrasco F, Alachiotis N, Berger S, Flouri T, Pissis SP, Stamatakis A (2013) A generic vectorization scheme and a GPU kernel for the phylogenetic likelihood library. In: Proceedings of IEEE IPDPS 2013. IEEE, pp 530–538
20. Jünger D, Hundt C, González-Domínguez J, Schmidt B (2017) Speed and accuracy improvement of higher-order epistasis detection on CUDA-enabled GPUs. Clust Comput 20(3):1899–1908
21. Kaeli DR, Mistry P, Schaa D, Zhang DP (2015) Heterogeneous computing with OpenCL 2.0. Morgan Kaufmann Publishers, Waltham
22. Klus P et al (2012) BarraCUDA—a fast short read sequence aligner using graphics processing units. BMC Res Notes 5(27):1–7
23. Kuan L, Neves J, Pratas F, Tomás P, Sousa L (2014) Accelerating phylogenetic inference on GPUs: an OpenACC and CUDA comparison. In: Proceedings of the 2nd International Work-Conference on Bioinformatics and Biomedical Engineering, pp 589–600

24. Lin YS, Lin CY, Hung CL, Chung YC, Lee KZ (2015) GPU-UPGMA: high-performance computing for UPGMA algorithm based on graphics processing units. Concurr Comput Pract Exp 27(13):3403–3414
25. Ling C, Benkrid K, Hamada T (2012) High performance phylogenetic analysis on CUDA-compatible GPUs. ACM SIGARCH Comput Archit News 40(5):52–57
26. Ling C, Gao J, Lu G (2016) Phylogenetic likelihood estimation on GPUs using vertical partitioning scheme. In: Proceedings of the 2016 IEEE Trustcom/BigDataSE/ISPA. IEEE, pp 1210–1217
27. Majumder T, Sarkar S, Pande PP, Kalyanaraman A (2012) NoC-based hardware accelerator for breakpoint phylogeny. IEEE Trans Comput 61(6):857–869
28. Martins WS, Rangel TF, Lucas DCS, Ferreira EB, Caceres EN (2012) Phylogenetic distance computation using CUDA. In: de Souto MC, Kann MG (eds) BSB 2012: advances in bioinformatics and computational biology, LNCS, vol 7409. Springer, Berlin, pp 168–178
29. Mirande JM (2017) Combined phylogeny of ray-finned fishes (Actinopterygii) and the use of morphological characters in large-scale analyses. Cladistics 33(4):333–350
30. Morgenstern I et al (2012) A molecular phylogeny of thermophilic fungi. Fungal Biol 116(4):489–502
31. Nobile M, Cazzaniga P, Tangherloni A, Besozzi D (2017) Graphics processing units in bioinformatics, computational biology and systems biology. Brief Bioinform 18(5):870–885
32. Ohue M, Shimoda T, Suzuki S, Matsuzaki Y, Ishida T, Akiyama Y (2014) MEGADOCK 4.0: an ultra-high-performance protein–protein docking software for heterogeneous supercomputers. Bioinformatics 30(22):3281–3283
33. Pratas F, Trancoso P, Sousa L, Stamatakis A, Shi G, Kindratenko V (2012) Fine-grain parallelism using multi-core, Cell/BE, and GPU systems. Parallel Comput 38(8):365–390
34. Quang D, Guan Y, Parker SCJ (2018) YAMDA: thousandfold speedup of EM-based motif discovery using deep learning libraries and GPU. Bioinformatics 34(20):3578–3580
35. Rokas A (2011) Phylogenetic analysis of protein sequence data using the randomized axelerated maximum likelihood (RAxML) program. Curr Protoc Mol Biol 96:19.11.1–19.11.14
36. Roshan UW, Moret BME, Williams TL, Warnow T (2004) Rec-I-DCM3: a fast algorithmic technique for reconstructing large phylogenetic trees. In: Proceedings of the 3rd IEEE Computational Systems Bioinformatics Conference. IEEE, pp 98–109
37. Santander-Jiménez S, Ilic A, Sousa L, Vega-Rodríguez MA (2017) Accelerating the phylogenetic parsimony function on heterogeneous systems. Concurr Comput Pract Exp 29(8):1–15
38. Santander-Jiménez S, Vega-Rodríguez MA, Vicente-Viola J, Sousa L (2019) Comparative assessment of GPGPU technologies to accelerate objective functions: a case study on parsimony. J Parallel Distrib Comput 126:67–81
39. Sheskin DJ (2011) Handbook of parametric and nonparametric statistical procedures, 5th edn. Chapman & Hall/CRC, New York
40. Thomson R, Shaffer H (2010) Sparse supermatrices for phylogenetic inference: taxonomy, alignment, rogue taxa, and the phylogeny of living turtles. Syst Biol 59:42–58
41. Vazquez-Ortiz KE, Richer JM, Lesaint D (2016) Strategies for phylogenetic reconstruction—for the maximum parsimony problem. In: Proceedings of the 9th International Joint Conference on Biomedical Engineering Systems and Technologies (BIOSTEC 2016), pp 226–236
42. Warnow T (2017) Computational phylogenetics: an introduction to designing methods for phylogeny estimation. Cambridge University Press, Cambridge
43. Wilt N (2013) The CUDA handbook: a comprehensive guide to GPU programming. Addison Wesley, Pearson
44. Wu D et al (2009) A phylogeny-driven genomic encyclopedia of bacteria and archaea. Nature 462(7276):1056–1060
45. Xilinx: breathe new life into your data center with alveo adaptable accelerator cards. White Paper: Alveo Accelerator Cards, 1–12 (2018)