IEICE TRANS. INF. & SYST., VOL.E100–D, NO.2 FEBRUARY 2017

PAPER

FPGA Hardware Acceleration of a Phylogenetic Tree Reconstruction with Maximum Parsimony Algorithm

Henry BLOCK[†a)], ***Nonmember and*** **Tsutomu MARUYAMA**[†b)], ***Member***

SUMMARY In this paper, we present an FPGA hardware implementation for a phylogenetic tree reconstruction with a maximum parsimony algorithm. We base our approach on a particular stochastic local search algorithm that uses the Progressive Neighborhood and the Indirect Calculation of Tree Lengths method. This method is widely used for the acceleration of the phylogenetic tree reconstruction algorithm in software. In our implementation, we define a tree structure and accelerate the search by parallel and pipeline processing. We show results for eight real-world biological datasets. We compare execution times against our previous hardware approach, and TNT, the fastest available parsimony program, which is also accelerated by the Indirect Calculation of Tree Lengths method. Acceleration rates between 34 to 45 per rearrangement, and 2 to 6 for the whole search, are obtained against our previous hardware approach. Acceleration rates between 2 to 36 per rearrangement, and 18 to 112 for the whole search, are obtained against TNT.

key words: *FPGA, hardware acceleration, phylogenetic tree reconstruction, maximum parsimony*

1. Introduction

In biology, phylogenetics is the field that attempts to reconstruct the evolutionary relationships among entities. These entities (taxa) can be species, genomes, genes, regions of a gene, protein sequences, molecule sequences, etc. [1], [2]. The relationships are graphically represented on a tree as ancestor-descendant relationships in a branching pattern. The tree starts from a common ancestor, known as the root of the tree, and branches into distinct lineages that descend to the taxa that are being analyzed. An example of a phylogenetic tree for the true flies is shown in Fig. 1.

Different methods exist to infer a phylogenetic tree, and they are classified into four main categories: Distance, Maximum Likelihood, Bayesian, and Parsimony methods [2]. In this work, we are concerned with phylogenetic reconstruction of molecular DNA sequence data with maximum parsimony. Using molecular sequence data has become possible thanks to advances in DNA sequencing, and is now an important field with applications in biology and medicine [4].

The maximum parsimony criterion is based on the assumption that the most likely tree is the one that requires the fewest number of evolutionary changes to explain the given data [5]. This implies evaluating all possible trees. However,

Manuscript received October 19, 2015.
Manuscript revised August 3, 2016.
Manuscript publicized November 14, 2016.
[†]The authors are with Faculty of Engineering, Information and Systems, University of Tsukuba, Tsukuba-shi, 305–8573 Japan.
a) E-mail: henry@darwin.esys.tsukuba.ac.jp
b) E-mail: maruyama@darwin.esys.tsukuba.ac.jp
DOI: 10.1587/transinf.2015EDP7433

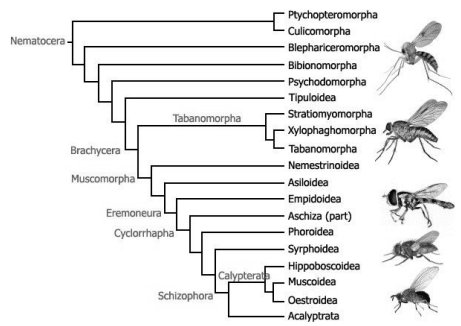

Fig. 1 True Flies (Diptera) Tree. [3]

due to the computational complexity [6], heuristic methods are used to find a suboptimal solution without having to do a complete search through the tree space. Different heuristic methods have been proposed, and the most well-known are based on a stochastic local search that uses a neighborhood relation to explore the tree space [7]. Nevertheless, a software implementation of these methods can still require a considerable amount of time for a larger phylogenetic problem.

Hence, FPGA hardware approaches have been proposed to reduce the execution time. For example, in some previous works [8]–[11], accelerations for the maximum parsimony problem were proposed. However, in this previous work [8], the approach is based on the evaluation of all possible tress, and is limited to a number of only 12 taxa. In this other previous work [9], the approach is not restricted by the number of taxa, but it only addresses the parsimony function, not the whole search algorithm. Next, in our previous work [10], we proposed an approach for the acceleration of the local search algorithm that uses the Progressive Neighborhood [7]. Although the speedup obtained was considerable against a software implementation of the same algorithm, it didn't exceed TNT, the fastest available parsimony program [12]. Then, in our other previous work [11], we added to our approach the Indirect Calculation of Tree Lengths method [14]. This method allowed us to exceed by far the acceleration rates obtained in our previous work [10], and surpass those of TNT.

In this work, we continue with the same approach, but here we focus more on the following points: describing each of the units involved in the FPGA implementation, evaluating the approach for even larger problems, and analyzing the performance in more detail.

We decided to adopt an FPGA rather than a GPU or

other many-core accelerators, because the algorithm under consideration involves a lot of memory accesses and detailed low-level hardware operations that are not suit for high-level languages. In addition, GPUs or other many-core accelerators impose a fixed programming model, whereas FPGAs do not; thus, they allow a higher level of customization [13]. This means that we can obtain a higher performance by designing a specific circuit that best suits the phylogenetic tree reconstruction algorithm.

This paper is organized as follows. In the next section, we give an overview of the main steps involved in the algorithm used for phylogenetic tree reconstruction. In Sect. 3, we explain the hardware approach taken. In Sect. 4, we show the implementation and simulation results obtained. Then, in Sect. 5, we discuss these results. Finally, we end our work with the conclusions.

2. Overview of the Algorithm for Phylogenetic Tree Reconstruction

In this section, we explain the main steps involved in the algorithm for phylogenetic tree reconstruction: the first-pass optimization, the second-pass optimization, the evaluation of rearrangement trees by applying the Indirect Calculation of Tree Lengths method, and the Progressive Neighborhood. These steps are used in the stochastic local search algorithm shown at the end of the section.

2.1 Tree Optimization

The tree optimization process comprises two phases. The first phase is called the first-pass optimization. The second phase is called the second-pass optimization.

2.1.1 First-Pass Optimization

The first-pass optimization is the phase where the preliminary ancestral character states of the nodes, along with the length, i.e the score, are found. There exist two widely used methods that depend on the algorithm applied: Fitch's algorithm [15] and Sankoff's algorithm [16]. Here, we use Fitch's, because it is less complex. The first-pass optimization proceeds from the tips of the tree (taxa) by formulating an ancestral character state for each node in the tree, working backwards, until the character state for the most distant node (root) has been formulated. In addition, each time an evolutionary change takes place, the score of the tree is increased by one. An ancestral node state, along with the tree length, is inferred by using the algorithm in Fig. 2 for each of the characters in the sequence data matrix: where PA refers to the preliminary character state of the current node, and PB and PC to that of its descendant nodes. TL refers to the length of the tree. The final score of the tree can then be calculated as the sum of the lengths obtained for each of the characters in the sequence data matrix. An example of the first-pass optimization is shown in Fig. 3 for a tree with 5 taxa.

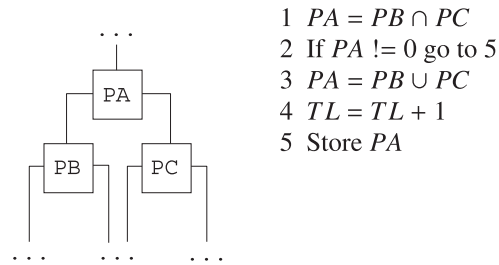

Fig. 2 First-pass Optimization Algorithm [17]

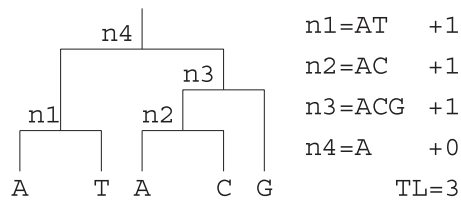

Fig. 3 First-pass optimization example

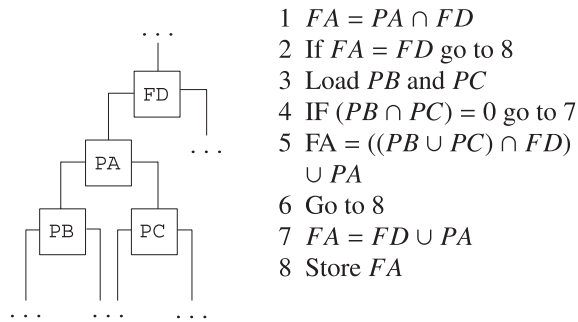

Fig. 4 Second-pass Optimization Algorithm [17]

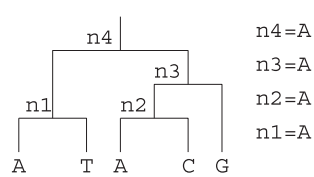

Fig. 5 Second-pass optimization example

2.1.2 Second-Pass Optimization

The second-pass optimization is the phase where the final ancestral character states are found. This pass proceeds in the reversed order, from the root to the tips of the tree. The final character state for a node is obtained by applying the algorithm in Fig. 4 for each of the characters in a node: where FA refers to the final character state of the current node PA, FD to that of its parent node, and PB and PC to the preliminary character states of its descendants. An example of the second-pass optimization is shown in Fig. 5 for the previous tree.

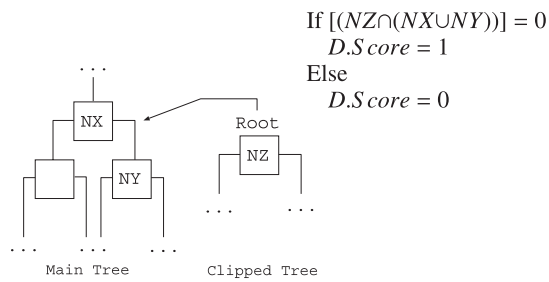

Fig. 6 Difference Score Calculation [14]

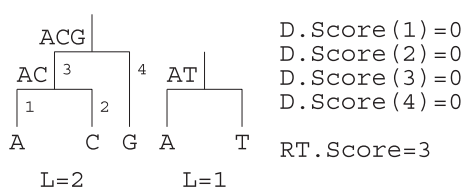

Fig. 7 Rearrangement evaluation example

2.2 Indirect Calculation of Tree Lengths

A first-pass optimization for a tree with T taxa requires visiting each one of the $T-1$ internal nodes. Thus, the time required increases with T. On the other hand, the Indirect Calculation of Tree Lengths method allows to evaluate all rearrangements that can be constructed after clipping the tree without having to visit each internal node. The time required is approximately $1/T$, and does not increase with T [14]. The score of a rearrangement can be obtained by the following Eq. (1).

$$RT.Score = MT.Score + CT.Score + D.Score \tag{1}$$

where $RT.Score$ is the score of the rearrangement tree being evaluated, $MT.Score$ is that of the main tree, $CT.Score$ is that of the clipped tree, and $D.Score$ is that of the difference between the clipped tree's root and the insertion branch in the main tree. This difference can be calculated as shown in Fig. 6: where NZ is the final character state of the root of the clipped tree, NX is the final character state of a node in the main tree, and NY is that of its descendant. In other words, NX and NY form the branch where NZ could be reinserted. This method is exact, i.e. it will always produce the right score. However, it requires a second-pass optimization on top of the first every time the tree is clipped, because the final states of all nodes are required [14]. An example for the rearrangement evaluation using the Indirect Calculation of Three Lengths method is shown in Fig. 7 for a main tree with 5 taxa and a subtree with 2 taxa.

$D.Score(i)$ is the $D.Score$ when the clipped tree is inserted in i. In this particular example, all the possible rearrangements that can be generated have the same score equal to 3, because the score differences of all of them are 0.

2.3 Progressive Tree Neighborhood

The Progressive Neighborhood aims to combine the properties of large and small neighborhoods by changing its size as the search progresses. It starts with a large neighborhood and ends with a small neighborhood. Starting the search with a large neighborhood allows examining more neighbors, and ensures a more global search. Then, as the search progresses the neighborhood gets reduced, so the changes applied to the tree topology are more restricted. A simple progressive tree neighborhood uses Sub-tree Pruning and Regrafting (SPR) as the large neighborhood and Nearest Neighbor Interchange (NNI) as the small neighborhood. To change the size of the neighborhood during the search, a distance parameter is used to constrain the distance between the pruned branch and the branch where it is reinserted [7]. The progressive tree neighborhood that uses SPR and NNI can be described by Eqs. (2) and (3) [7].

$$\begin{matrix} N^{SPR}_{d_{init}} \equiv N^{SPR} \\ N^{SPR}_{d_{final}} \equiv N^{NNI} \end{matrix} \rightarrow \begin{pmatrix} d_{init} \\ d_{final} \end{pmatrix} = \begin{pmatrix} max\delta(v_i, v_j) \\ 1 \end{pmatrix} \tag{2}$$

where v_i is the node at which the branch is cut, v_j is the node at which the pruned branch is reinserted, and $\delta(v_i, v_j)$ is the distance between these two nodes. It is given by

$$d = d_{init}(1 - \frac{i}{M}), \qquad i < M \tag{3}$$

where i is the i_{th} local search iteration and M is the maximum number of local search iterations. The distance parameter starts at d_{init} and ends at a value close to 1.

2.4 Stochastic Local Search Algorithm

The algorithm is based on the iterative descent, which is guided by the length of the tree as the score function. It starts from a randomly generated tree in the search space, and tries to improve it on each iteration. For the initial tree, a list for all the branches is created. This list will denote which rearrangements have to be tried. Then, a first-pass optimization is done to calculate the initial score of the tree. Following this, the next steps are performed repeatedly until the algorithm comes to a stop.

1. A branch to clip from the tree is chosen at random from the list.
2. The main tree and clipped tree derived from the previous clipping are created.
3. A first-pass optimization is done on both the main and clipped tree. If the score of the sum is equal or greater (worse) than the current score, it proceeds to step 7.
4. A second-pass optimization is done on the main tree.
5. All rearrangements within the neighborhood are evaluated. If the score of the best rearrangement found is equal or greater than the current score, it proceeds to step 7.

6 The clipped tree is inserted in the main tree, the current score is updated, all branches to the list are added again, and it proceeds to step 1.
7 The branch is removed from the list. If there are still branches in the list, it proceeds to step 1.

This algorithm will always converge to a local optimum after all branches in the list have been tried. In other words, after it has been found that no rearrangement is better than the current tree.

3. Our Hardware Approach

In this section, we explain our hardware approach for the implementation of the algorithm previously stated. First, we talk about the data structure used to store the phylogenetic information of a given problem. Then, we proceed to show the proposed hardware architecture. It was designed for parallel and pipeline processing. That is why we divided the architecture in different units, each of which performs a particular task of the algorithm. We show how this works. Finally, we give details of each of the main units of the hardware architecture.

3.1 Data Structure for Phylogenetic Tree Information

For a given phylogenetic tree reconstruction problem consisting of N taxa, each of which has a sequence of L nucleotides, the sequence data matrix is an N rows $\times$ L columns matrix. The characters in the sequences include not only the DNA nucleobases Adenine (A), Cytosine (C), Guanine (G) and Thymine (T), but also the '-' character, which represents a gap, and the '?' character, which represents an undefined character. These are the six basic characters, but a combination of them is also possible due to the five-bit binary representation used [18]. Hence, a memory of $N \times L \times 5$ bits is required to store the sequence data matrix. On the other hand, the tree topology shows the connections between the internal nodes of the tree and the leaves. A tree with N taxa has $N-1$ nodes, including the root node. Since the tree is a binary tree, each node has a left branch and a right branch. And it has a parent node. Thus, the size of the memory required to store the tree topology is $(N-1) \times 3[log_2(N)]$ bits. Finally, the size of the memory required to store the node states is of $(N-1) \times L \times 5$ bits. For example, the tree topology, the sequence data matrix and the node data matrix memories of a tree with six taxa can be represented as shown in Fig. 8.

Taxa are labeled according to their memory position on the Sequence Data Matrix Memory. Likewise, nodes are labeled according to their memory position on the Tree Topology Memory. Since both nodes and taxa appear on the same, we use an additional bit to distinguish between the two of them: 0 for a node and 1 for a taxon (leaf). The root node doesn't have a parent. Instead, a full sequence of 1s can be used to identify it.

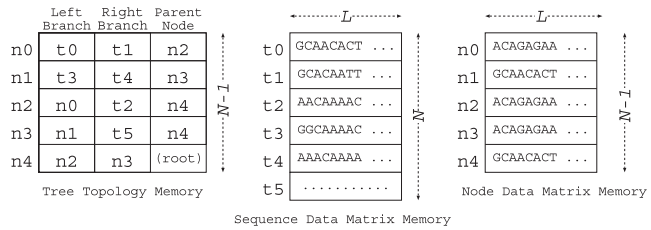

Fig. 8 Example. Tree Topology Memory, Sequence and Node Memory

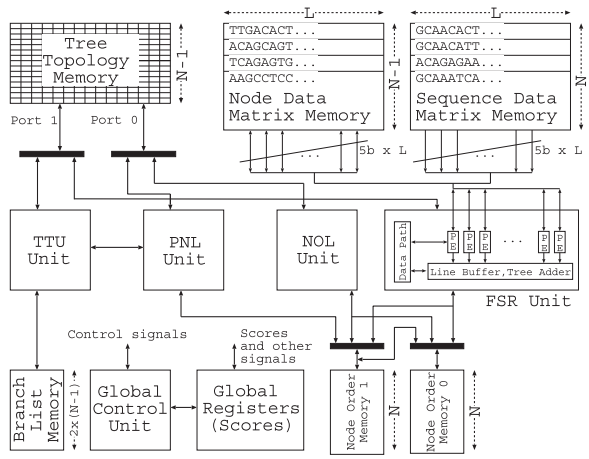

Fig. 9 General Block Diagram of the Proposed Architecture

3.2 Proposed Hardware Architecture

Our system consists of the following four main units.

1. Tree Topology Update (TTU)
2. Progressive Neighborhood Listing (PNL)
3. Node Order Listing (NOL)
4. First-, Second-pass optimization and Rearrangement evaluation (FSR)

The TTU unit is in charge of updating the tree topology memory to reflect the changes produced by the clipping and reinserting process. The PNL unit is in charge of listing all possible nodes in the main tree where the clipped branch could be reinserted. The NOL unit has the task of listing the nodes of the given tree for a post-order tree traversal. The FSR unit has the most important tasks, which are doing a first, and second-pass optimization, and evaluating all possible rearrangements. A simplified general block diagram of the architecture is shown in Fig. 9.

It consists of the following elements: the dual-port Tree Topology Memory (TTM), the dual-port Sequence Data Matrix Memory (SDM), the dual-port Node Data Matrix Memory (NDM), two Node Order Memories (NOM), the Branch List Memory (BLM), the TTU unit, the PNL unit, the NOL unit, the FSR unit, and a Global Control unit with some registers. Black bars on the diagram make reference to multiplexers.

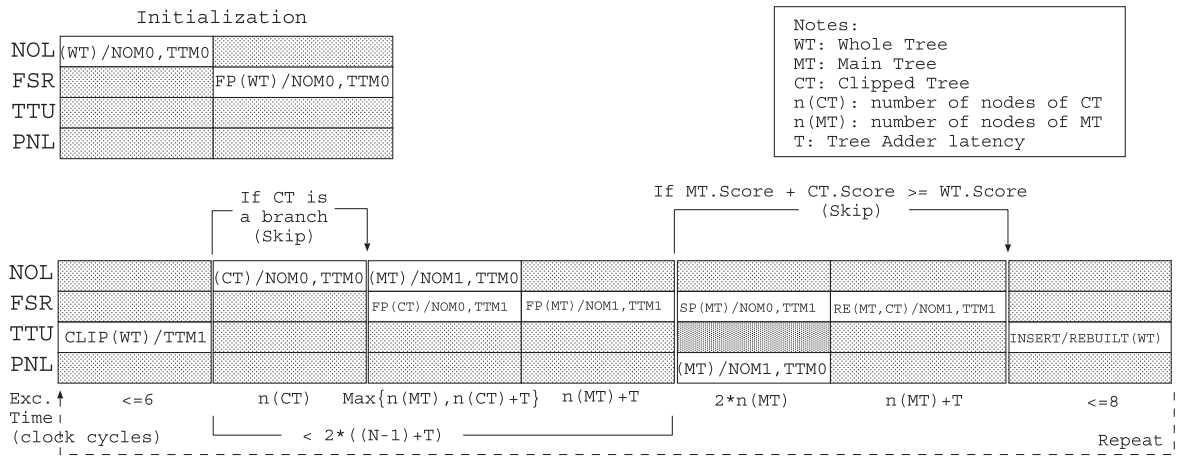

Fig. 10 Parallel Processing of the four units

3.3 Parallel and Pipeline Processing

The four units run using the memory banks as shown in Fig. 10, where FSR-FP refers to the first-pass optimization, FSR-SP to the second-pass optimization, and FSR-RE to the rearrangement evaluation. This explains the use of a dual-port Tree Topology Memory and two Node Order Memories as shown in Fig. 9. NOL and FSR-FP, PNL and FSR-SP work in parallel.

Other than this parallel processing, pipeline is used inside the FSR unit to further accelerate the performance. This unit has L processing elements, which allow processing all L characters in the taxa in parallel.

3.4 Details of the Four Main Units

3.4.1 TTU Unit

This unit is in charge of updating the tree topology memory to reflect the changes produced by the clipping and reinserting process. It has three main tasks:

1. Clip the whole tree to create the main tree and sub tree.
2. Insert the clipped tree in the chosen insertion branch.
3. Rebuilt the whole tree.

And two sub tasks:

1. Keep track of the clipped branch and reinsertion branch.
2. Keep track of the whole, main and sub tree roots.

The clipping process involves updating at most three nodes. Since it takes two clock cycles to update a node from the tree topology memory, the execution time is of 6 clock cycles at most. Similarly, inserting the clipped tree involves updating at most 4 nodes. Thus, it takes 8 clock cycles at most. Rebuilding the tree is equivalent to revert the clipping process; hence, it takes also 6 clock cycles at most.

3.4.2 PNL Unit

This unit is in charge of listing all possible nodes in the main tree where the clipped branch could be reinserted. It takes into account that the distance between the clipped branch and that where it could be reinserted is restricted by the distance parameter according to the progressive neighborhood relation (refer to section 2.3). The resulting listing is stored in one of the two node order memories. The PNL unit requires an execution time that depends on the number of nodes that have to be listed. However, since it requires two clock cycle to list each node at maximum, its execution time never exceeds $2 \times (N - 2)$ clock cycles for the worst-case scenario where all nodes except one have to be listed. Moreover, this unit works in parallel with FSR-FP, so its execution time does not add any delay.

3.4.3 NOL Unit

This unit has the task of listing the nodes of the given tree for a post-order tree traversal. This order is used for the score calculation in the first-pass optimization. It can generate the post-order of the whole, main or sub tree depending on the root node chosen. The listing is stored in one of the two node order memories. It uses an internal stack memory as a temporal storage. It works by following the next algorithm, where LB refers to the left branch and RB to the right branch:

1. Read the memory position of the given root node from the tree topology memory.
2. Repeat:
   - 2.1 Push current memory position into Node Order Memory
   - 2.2 Case (LB, RB)
     (Node, Node): Push RB, and go to memory position of LB
     (Node, Leaf): Go to memory position of LB

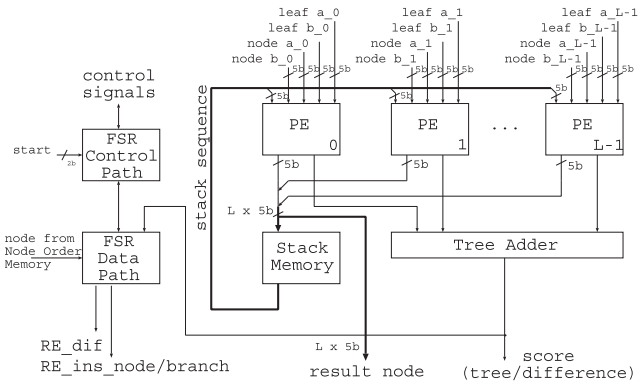

Fig. 11 General Block Diagram of FSR Unit

(Leaf, Node): Go to memory position of RB
(Leaf, Leaf): If stack is not empty: pop a node, and go to that memory location Else: break

The NOL Unit takes one clock cycle to list each node, so it requires $n - 1$ clock cycles to list all n nodes.

3.4.4 FSR Unit

This unit is the most important unit. It has three main tasks:

1. Do a first-pass optimization following the order saved in the Node Order Memory (post-order).
2. Do a second-pass optimization following the order saved in the Node Order Memory (reversed).
3. Evaluate all the possible rearrangements following the order saved in the Node Order Memory (PNL-order).

Its general block diagram is shown in Fig. 11. It is composed of L processing elements (PE), a tree adder, a stack memory, and a control logic unit along with a data path. The inputs of the FSR Unit are two taxa (leaves) from the Sequence Data Matrix Memory, two nodes from the Node Data Matrix Memory, and the node order from one of the Node Order Memories. These L PEs allow to process all the characters of two nodes or leaves in parallel.

The block diagram of a PE is shown in Fig. 12. This unit implements the three main tasks of the FSR unit described previously, but for a single character. Depending on a control signal (*ctrl* in Fig. 12) the PE changes its functionality. The idea behind this is to share the same resources for the first, second-pass optimization, and for the rearrangement evaluation, since they need not to work at the same time. Each of these tasks works using pipeline. Now, we will describe in general lines how they work.

FSR-FP:

The FSR-FP uses a 4-stage pipelined algorithm, as shown in the next listing:

Stg01 Take a node from NOM; read that node from TTM
Stg02 Read LB and RB from SDM1/SDM2
Stg03 Optimize the node (first-pass)
Stg04 Save resulting sequence into NDM1

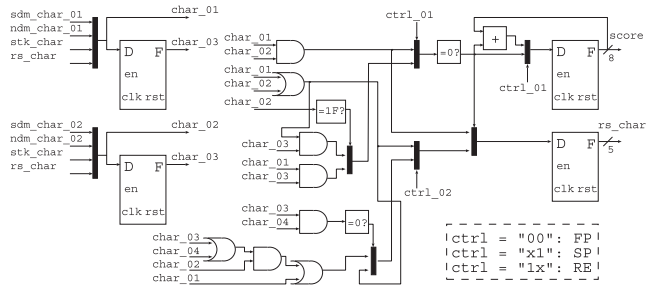

Fig. 12 Block Diagram of a PE

It finishes when all nodes in the Node Order Memory have been evaluated. Then, the score is obtained after summing the individual results from all the PEs using the tree adder. The total execution time will approximate $n + T$ clock cycles, where n is the number of nodes of the main or clipped tree, and T the length of the tree adder.

FSR-SP:

The FSR-SP uses a 5-stage pipelined algorithm, as shown in the next listing:

Stg01 Take a node from NOM; read that node from TTM
Stg02 Read LB and RB from SDM1/SDM2 and/or NDM1/NDM2
Stg03 Decode and store temporal results into the registers; read node from NDM1
Stg04 Optimize node (second-pass)
Stg05 Save resulting sequence into NDM2

These pipeline works in two phases. In phase one, stages 1, 3 and 5 work in parallel. In the other phase, stages 2 and 4 work in parallel. It was divided in two phases, because not all operands can be read at the same time from the memories. The execution time approximates 2 clock cycles per node as the number of nodes increases. Since the tree adder is not used for this task, the total execution time approximates $2n$.

FSR-RE:

The FSR-RE uses a 4-stage pipelined algorithm, as shown in the next listing:

Ini Read subtree root from NDM1 or SDM1 (NZ)
Stg01 Take a node from NOM; read that node from TTM
Stg02 Read node from NDM1 and read LB from SDM2 or NDM2
Stg03 Read node from NDM2 and read RB from SDM2 or NDM2; Evaluate first rearrangement (NX=Node, NY=LB)
Stg04 Evaluate second rearrangement (NX=Node, NY=RB)

This unit also works in two phases. In phase one, stages 1 and 3 work. In the other, stages 2 and 4 work. Furthermore, the difference score is summed using the tree adder. Finally, the rearrangement with the lowest score is kept as the reinsertion branch. For this purpose, a line buffer and some comparison registers in the FSR Datapath are used.

Table 1 Datasets used [19]

ID	M972	M2355	M3452	M3875
#taxa	155	150	116	228
#characters	355	829	1157	1435
ID	M17200	M2616	M21001	M24084
#taxa	326	330	364	414
#characters	1434	1711	4119	4584

Table 2 Implementation Results on a Kintex-7

Logic Utilization	Used	Available	Utilization
Number of Slices	23610	50950	46%
Number of Slice Registers	55174	407600	13%
Number of Slice LUTs	94442	203800	46%
Number of BRAMs (36 Kb)	402	445	90%
Maximum Frequency	163.826MHz		

Since two rearrangements are evaluated consecutively, the execution time will approximate 1 clock cycle per rearrangement as the number of nodes increases. The total execution time will be $n + T$.

4. Implementation and Simulation Results

In this section we show implementation and simulation results for eight real-word biological datasets. The last two were not evaluated in our previous work [11]. The datasets were obtained from the repository of phylogenetic information TreeBASE [19], see Table 1. Results for the first four datasets were obtained from the implementation of our system on a Kintex-7 XC7K325T-FF2-900 FPGA. Results for the last four problems were obtained from simulation only, because the required resources exceed those available on the FPGA used. Though, a larger FPGA, e.g. XC7VX690T, could be used if their implementation is required.

Evaluating more datasets would be ideal, but due to reasons of time, we limited our evaluation to only these eight datasets. However, we consider them enough and adequate to measure the performance of our approach. Each dataset was chosen carefully based on its number of taxa and characters. As can be seen from Table 1 the number of characters increases from one dataset to the other, starting at 355 and ending at $4,584$. The number of taxa also has a tendency to increase (except for problems M2355 and M3452). These datasets can be considered of medium to large size, which is an optimal size for evaluating the performance.

4.1 Logic Resources and Performance Results

The hardware resources/performance results are shown in Table 2. This implementation covers any of the first four datasets, as mentioned previously. Problems up to $N = 1,024$ and $L = 1,500$ can be processed. The number of LUTs is almost proportional to L, the number of BRAMS to $L \times N$.

Table 3 Results for the Local Search

Dataset		[10]	Our Approach	TNT
M972	Total time (ms)	62.2	11.27	890
	Time/tree (μs)	1.243	0.031	0.065
	Visited trees	50,031	364,177	13,637,086
	Best score	1548	1533	1543
M2355	Total time (ms)	55.5	9.9	500
	Time/tree (μs)	1.109	0.025	0.059
	Visited trees	50,032	400,368	8,497,522
	Best score	2749	2724	2771
M3452	Total time (ms)	49.9	9.64	180
	Time/tree (μs)	0.997	0.029	0.103
	Visited trees	50,046	329,025	1,749,117
	Best score	3633	3632	3624
M3875	Total time (ms)	82.8	27.76	510
	Time/tree (μs)	1.65	0.037	0.021
	Visited trees	50,172	760,180	23,818,061
	Best score	605	567	564
M17200*	Total time (ms)	No Data	53.56	2260
	Time/tree (μs)	No Data	0.019	0.046
	Visited trees	No Data	2798944	49662276
	Best score	No Data	4344	4340
M2616*	Total time (ms)	No Data	42.25	4700
	Time/tree (μs)	No Data	0.027	0.09
	Visited trees	No Data	1564515	53330179
	Best score	No Data	10003	10004
M21001*	Total time (ms)	No Data	99.54	3060
	Time/tree (μs)	No Data	0.02	0.45
	Visited trees	No Data	4,930,362	6,805,502
	Best score	No Data	118,734	118,468
M24084*	Total time (ms)	No Data	114.18	7850
	Time/tree (μs)	No Data	0.017	0.3
	Visited trees	No Data	6,656,788	26,185,821
	Best score	No Data	103,516	103,684

4.2 Execution Times for Phylogenetic Tree Reconstruction Problems

Here, we compare our hardware implementation with our previous approach [10], one of the fastest FPGA systems to the best of our knowledge, and with TNT version 1.1 of March 2014 [12], the fastest available parsimony program. Our previous approach didn't use the Indirect Calculation of Tree Lengths method. Each rearrangement tree was evaluated by using a complete first-pass optimization. On the other hand, TNT does make use of the Indirect Calculation of Tree Lengths method. Furthermore, it uses multi-core processing (4 cores in this evaluation) with SIMD instructions. To make the comparison as fair as possible, we use the traditional search of TNT based on SPR, and start from a random tree. This is the closest setting of TNT that resembles our algorithm, and the one proposed in our previous work [10]. Moreover, since the total number of examined trees is not the same, we also show the average execution time required for each tree. The CPU used is an Intel Core i7 860@2.80GHz with 4 GB RAM. The targeted FPGA runs at 156.25 MHz. The results are summarized in Table 3. (*Note: results for these datasets are simulation results).

Now using these results, we show the speedups obtained for the whole local search, and for the evaluation of a single tree in Fig. 13 and 14, respectively.

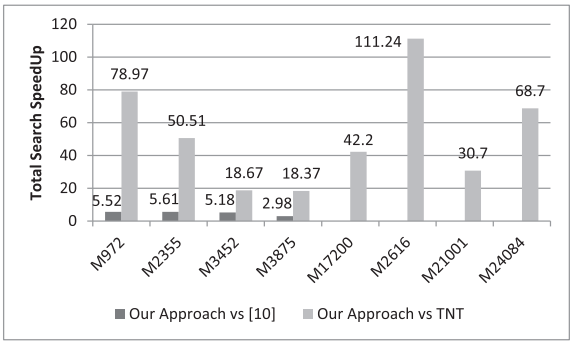

Fig. 13 Speedup for the Local Search

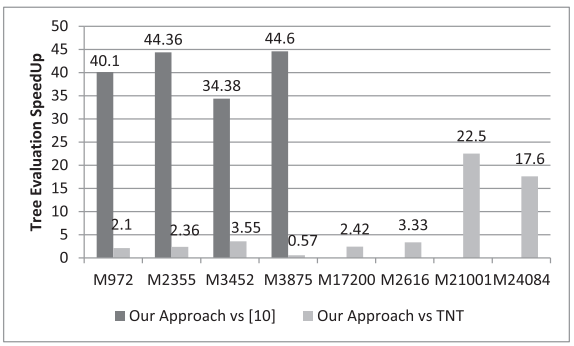

Fig. 14 Speedup for the Evaluation of a Tree

5. Discussion

Compared to our previous work [10], our current approach provides a speedup between 2 and 6 for the whole local search, and between 34 and 45 for the evaluation of a single tree. Although our approach evaluates more trees than our previous work [10], there is a speedup for the whole local search. Moreover, the speedup for the evaluation of a single tree is considerable. This is thanks to having applied the Indirect Calculation of Tree Lengths Method. On the other hand, in comparison to TNT, our approach yields a speedup between 18 and 112 for the whole local search, and between 2 and 23 for the evaluation of a single tree, except for problem $M3875$, for which there is no speedup. We think that the reason for this might lie in the particularities of problem $M3875$ itself, which make it easier for TNT. Naturally, there is a high speed up for the whole local search, because our approach evaluates fewer trees. However, it should be noted that our approach reaches a similar score (see Table 3) in much less time. For problems $M972$, $M2355$, $M2616$ and $M24084$ the score is better, and for problems $M3452$, $M3875$, $M17200$ and $M21001$ it is worse. The speedup for the evaluation of a single tree grows roughly with the number of characters. As can be seen from Fig. 14, our approach reaches a speed up of 22.5 and 17.6 for the last two problems, which are the ones with more characters, 4119 and 4584, respectively. The speedup should increase theoretically with the number of characters, because in our approach we process all characters in parallel. On the contrary, a software approach like TNT requires to process characters serially; thus, taking more time as its number increases.

We believe that our FPGA implementation is the best approach to achieve the highest performance. We think that the performance that could be obtained by implementing the algorithm on a GPU or other many-core accelerators such as CUDA-implementation would be poor for the following reasons. First, the amount of shared memory (on-chip fast-access memory) available on a GPU or other many-core accelerators is much limited. For example, the GTX 980 Ti GPU has a total of 1,152kB of shared memory, which is much less than the 6,615kB of the Virtex-7 690T FPGA. Thus, using a GPU would either limit the size of phylogenetic datasets that can be processed or it would decrease the performance significantly by having to use the global memory. Second, to calculate the score of a tree, we simply add up the individual results from each processing element (PE) by using a tree adder that has a latency equal to its depth. However, in the case of a GPU, the individual results from each thread block, which would be stored in the shared memory, would have to be written first into the global memory before they can all be added. This would signify a great delay that would reduce the overall performance. Third, the first- and second-pass optimization algorithms are composed of some conditional clauses and low-level hardware operations. In the FPGA we implemented them as a group of logical gates and multiplexers inside the processing elements (PEs). Thus, it only takes one or two clock cycles at most to obtain the desired output. However, in a GPU this would not be possible. It would take more clock cycles to calculate the output, since the conditional clauses cannot be flattened and the low-level hardware operations cannot be directly implemented.

Regarding logic resources, we estimate that by using the Virtex-7 XC7VX690T FPGA, problems that have up to $1,024$ taxa and $4,900$ characters could be implemented. This estimation is based on the number of BRAMs needed to store the phylogenetic information in memories inside the FPGA. For example, problem M24084 takes $1,277$ BRAMs (86%) on the Virtex-7 FPGA. This maximum capability woud allow us to process $7,624$ of the $8,609$ DNA-type datasets (88.56%) from TreeBASE. Moreover, we estimate that by using the latest Virtex UltraScale XCVU190 FPGA, problems that have up to $1,024$ taxa and $13,000$ characters could be implemented. This new maximum capability would allow us to process $8,290$ datasets (96.26%) from TreeBASE. If datasets cannot be held in FPGA on-chip memory, our approach would not be able to outperform the CPU implementation TNT. Nonetheless, we think that our approach has a huge practical contribution since most of the datasets from TreeBASE can be processed with it. And, we believe that in the near future, the number of datasets that can be processed inside one FPGA will continue to grow.

6. Conclusions

In conclusion, we showed implementation results for four, and simulation results for four other real-world biological datasets, which consist of hundreds of taxa and hundreds or thousands of characters. Our hardware approach can be applied for even larger problems as long as there are enough BRAM resources. We compared execution times against our previous approach [10] and TNT. Compared to our previous approach [10], our current approach was faster for all the problems. Compared to TNT our current approach was shown to be faster for all problems, except for problem $M3875$. The implementation of Indirect Calculation of Tree Lengths method and the parallel and pipeline processing used served to exceed by far our previous approach [10] and TNT.

Implementing other search strategies or different versions of the Indirect Calculation of Tree Lengths method, such as those described in these works [14], [18], is part of our future work. Furthermore, we think that an array of FPGAs can be used for phylogenetic tree reconstruction, since the computation of the first- and second-pass optimization, as well as the rearrangement evaluation is performed independently for each column of the data matrix. Thus, each FPGA could hold a portion of the data memory, and the individual results could be added to obtain the score of the tree. This would allow evaluating even larger datasets with current FPGAs. This is also part of our future work.

Acknowledgements

The authors would like to thank the Willi Hennig Society for providing a free edition of TNT [12].

References

[1] E.O. Wiley and B.S. Lieberman, Phylogenetics: Theory and Practice of Phylogenetic Systematics, 2nd ed., Wiley-Blackwell, 2011.
[2] N.H. Barton, D.E.G. Briggs, J.A. Eisen, D.B. Goldstein, and N.H. Patel, Evolution, 1st ed., Cold Spring Harbor Laboratory Press, 2007.
[3] Flytree, http://wwx.inhs.illinois.edu/research/flytree/flyphylogeny/, last access on October, 2015.
[4] D.M. Hillis, C. Moritz, and B.K. Mable, Molecular Systematics, 2nd ed., Sinauer Associates, 1996.
[5] A.W.F. Edwards and L.L. Cavalli-Sforza, "The reconstruction of evolution," Annals of Human Genetics, vol.27, pp.105–106, 1963.
[6] L.R. Founds and R.L. Graham, "The Steiner problem in phylogeny is NP-complete," Advances in Applied Mathematics, vol.3, no.1, pp.43–49, 1982.
[7] A. Goffon, J.-M. Richer, and J.-K. Hao, "Progressive tree neighborhood applied to the maximum parsimony problem," IEEE/ACM Trans. Comput. Biol. Bioinf., vol.5, no.1, pp.136–145, 2008.
[8] S. Kasap and K. Benkrid, "High performance phylogenetic analysis with maximum parsimony on reconfigurable hardware," IEEE Trans. Very Large Scale Integr. (VLSI) Syst., vol.99, pp.1–13, 2010.
[9] N. Alachiotis and A. Stamatakis, "FPGA acceleration of the phylogenetic parsimony kernel?," Proc. 21st International Conference on Field Programable Logic (FPL), pp.417–422, 2011.
[10] H. Block and T. Maruyama, "A hardware acceleration of a phylogenetic tree reconstruction with maximum parsimony algorithm using FPGA," International Conference on Field-Programmable Technology (FPT), pp.318–321, 2013.
[11] H. Block and T. Maruyama, "An FPGA hardware acceleration of the indirect calculation of tree lengths method for phylogenetic tree reconstruction," International Conference on Field Programmable Logic and Applications (FPL), pp.1–4, 2014.
[12] P.A. Goloboff, J.S. Farris, and K.C. Nixon, "TNT, a free program for phylogenetic analysis," Cladistics, The International Journal of the Willi Hennig Society, vol.24, no.5, pp.774–786, 2008.
[13] S. Che, J. Li, J.W. Sheaffer, K. Skadron, and J. Lach, "Accelerating compute-intensive applications with GPUs and FPGAs," Application Specific Processors, SASP, pp.101–107, 2008.
[14] P. Goloboff, "Methods for faster parsimony analysis," Cladistics, vol.12, no.3, pp.199–220, 1996.
[15] W.M. Fitch, "Towards defining course of evolution: minimum change for a specified tree topology," Systematic Zoology, vol.20, no.4, pp.406–416, 1971.
[16] D. Sankoff and P. Rousseau, "Locating the vertices of a Steiner tree in an arbitrary metric space," Mathematical Programming, vol.9, no.1, pp.240–246, 1975.
[17] F. Ronquist, "Fast fitch-parsimony algorithms for large data sets," Cladistics, vol.14, no.4, pp.387–400, 1998.
[18] J.-M. Richer, A. Goëffon, and J.-K. Hao, "A memetic algorithm for phylogenetic reconstruction with maximum parsimony," Evolutionary Computation, Machine Learning and Data Mining in Bioinformatics, Lecture Notes in Computer Science, vol.5483, pp.164–175, Springer Berlin Heidelberg, Berlin, Heidelberg, 2009.
[19] TreeBASE: A database of phylogenetic knowledge: http://www.treebase.org/ (last access on October, 2015).

Henry Block received his Master's degree in Engineering from the University of Tsukuba in 2015. He is currently a PhD student at the Faculty of Engineering, Information and Systems, University of Tsukuba. His research interest is in reconfigurable computing systems.

Tsutomu Maruyama received his PhD in Engineering from the University of Tokyo in 1987. He is currently a professor at the Faculty of Engineering, Information and Systems, University of Tsukuba. His research interest is in reconfigurable computing systems.