# FPGA Acceleration of Phylogeny Reconstruction for Whole Genome Data

Jason D. Bakos, Panormitis E. Elenis, Jijun Tang
Dept. of Computer Science and Engineering
University of South Carolina
Columbia, SC USA
{jbakos, elenis, jtang}@cse.sc.edu

*Abstract*—**In this paper we describe our design and characterization of a co-processor architecture to accelerate median-based phylogenetic reconstruction for gene-rearrangement data. Our current design performs a parallelized version of the breakpoint median computation and achieves an average speedup of 876 for simulated input data having a high evolution rate. After integrating our hardware-based median computation into the GRAPPA toolset, we have achieved an average speedup of 189 over the entire phylogenetic reconstruction procedure. The results in this paper suggest that FPGA-based acceleration is a promising approach for computationally expensive phylogenetic problems that are based on combinatorial optimization.**

***Keywords-FPGA; reconfigurable computing; phylogeny; high-performance; gene-rearrangment; whole-genome; genome analysis***

## I. INTRODUCTION

The recent reductions in cost associated with DNA sequencing have resulted in an explosion in the amount of genomic data of all types. These include large collections of isolated genes, entire prokaryotic genomes, genomes of unknown organisms (i.e. the now famous Sargasso Sea data), and complex eukaryotic genomes (model organisms). Making effective use of this data, from understanding small local changes in the genome of tumor cells to reconstructing the Tree of Life, will require an evolutionary perspective. In particular, the availability of fully sequenced and well-annotated prokaryotic genomes allows us to move beyond the mere sequence level and into the study of genomic evolution.

Once a genome has been annotated to the point where gene homologs can be identified, each gene family can be assigned a unique integer and each chromosome can be represented by an ordering (a string) of signed integers (where the sign indicates the strand). Rearrangement of genes under inversion, transposition, and other operations such as duplications, deletions, and insertions are known to be an important evolutionary mechanism. Their use in reconstructing phylogenies has been studied intensely since the pioneering papers of Sankoff [1]. Understanding these rearrangements is also a crucial step in comparative genomics, gene prediction, and other analyses. Biologists have embraced this new source of data in their phylogenetic and comparative genomics work [2], while computer scientists are slowly solving the difficult problems posed by the manipulations of these gene orders [3].

In the past several years, substantial progress has been made in understanding genome rearrangements and computing with such data. Pevzner's group provided the first breakthrough with a solution for computing shortest sequences of rearrangements (so-called edit distances) that would transform one ordering into the other [4]. Subsequent work from Moret's group gave a linear-time algorithm to compute these edit distances [5], techniques to tackle the NP-hard median problems [6], faster and better-scaling approaches to phylogenetic reconstruction [7], and the software package GRAPPA which has become one of the most accurate methods for inversion phylogenies [8]. Moreover, the extension of GRAPPA that uses the heuristic technique of disk-covering [9] (DCM-GRAPPA [10]) runs quickly on large datasets with more than 1,000 genomes.

As such, the number of taxa in the dataset is no longer the main issue. However, scoring a candidate tree requires solving many instances of the median computation, which may take days or even months to finish when the involved genomes are distant. Thus finding efficient median solvers is still desired.

Parallelizing the current methods on clusters or SMPs is one obvious way to alleviate this problem. However, large-scale parallel computers are extremely expensive to acquire and maintain. As such, this approach is not feasible for many biological and medical labs. Our approach is to apply High-Performance Reconfigurable Computing (HPRC) to finely parallelize GRAPPA and achieve cluster-class performance using an inexpensive, lightweight, and efficient desktop platform.

In the HPRC model, one or more Field Programmable Gate Array (FPGA) devices are attached to a general purpose CPU and used as an application-specific co-processor. An FPGA is a reconfigurable logic device that can be electronically configured (programmed) to implement any arbitrary digital logic circuit using the FPGA's programmable logic gates and integrated memory blocks.

Aside from the obvious differences in the way they are programmed, developing an efficient HPRC application is fundamentally very similar to developing a traditional HPC application. In both cases, the developer must explicitly identify parallelism and extract it. However, due to the lower communication and systems overhead, an HPRC system allows

1-4244-1509-8/07/$25.00 ©2007 IEEE

for the extraction of finer-grain parallelism as compared to an HPC system.

In this paper, we present our new results for adapting the breakpoint median and tree scoring algorithms to an HPRC platform. In our previous published work toward this goal, we achieved a 26X average speedup for the breakpoint median and a 23X speedup for the entire reconstruction procedure [11]. In this paper we present our improved architecture that achieves an 876X speedup for the breakpoint median and a 189X speedup for the entire reconstruction procedure.

To the best of the authors' knowledge, this is the first attempt to adapt this application to the HPRC computational model.

## II. Gene Rearrangement Data

We assume a reference set of $n$ genes $\{g_1, g_2, \ldots, g_n\}$. Thus, a genome can be represented by an ordering of some multisubset of these genes and each gene is given with an orientation that is either positive, written $g_i$, or negative, written $-g_i$. A genome can be *linear* or *circular*. A linear genome is simply a permutation on the multisubset, while a circular genome can be represented in the same way under the implicit assumption that the permutation closes back on itself. A genome can undergo various rearrangement events such as *inversion*, *transposition*, *deletion*, *duplication*, etc.

Let $G$ be the genome with signed ordering of $g_1, g_2, \ldots, g_n$. An *inversion* (also called a *reversal*) between indices $i$ and $j$ ($i \leq j$) produces the genome with linear ordering:

$$g_1, g_2, \ldots, g_{i-1}, -g_j, -g_{j-1}, \ldots, -g_i, g_{j+1}, \ldots, g_n$$

A *transposition* acts on three indices $i$, $j$, and $k$, with $i \leq j$ and $k \notin [i, j]$, picking up the interval $g_i, g_{i+1}, \ldots, g_j$ and inserting it immediately after $g_k$. Thus genome $G$ is replaced by (assume $k > j$):

$$g_1, \ldots, g_{i-1}, g_{j+1}, \ldots, g_k, g_i, g_{i+1}, \ldots, g_j, g_{k+1}, \ldots, g_n$$

An *inverted transposition* is a transposition followed by an inversion of the transposed subsequence (it is sometimes called a *transversion*). An *insertion* is the addition of one or a segment of genes, and a *deletion* is the loss of a section of the chromosome.

The generalized Nadeau-Taylor model [12] postulates that only rearrangement events—namely, inversions, transpositions, and inverted transpositions. The number of each of these three events obeys a Poisson distribution on each edge and the relative probabilities of each type of event are fixed across the tree.

Methods for reconstructing trees based on genome rearrangement data include distance-based methods (for example neighbor-joining [13]), maximum-likelihood methods [14], maximum parsimony methods based on encodings [15], and direct optimization methods. The latter, pioneered by Sankoff and Blanchette [16] in their package BPAnalysis and improved on by GRAPPA [8] and MGR [17], are the most accurate methods.

Direct optimization methods rely on finding median genomes. The median problem on $k$ genomes is to find a single genome that minimizes the median score (sum of the pairwise distances) between itself and each of the $k$ given genomes. This problem is NP-hard [18] even for three genomes.

GRAPPA (Genome Rearrangements Analysis under Parsimony and other Phylogenetic Algorithms) is an exhaustive search method, which moves systematically through the space of all $(2N-5)\times(2N-7)\times\bullet\bullet\bullet\times 3$ possible trees on $N$ genomes. For each tree, the program tests a lower bound to determine whether the tree is worth scoring. For every tree that is scored, the program will iteratively solve the median problems at internal vertices until convergence, as outlined in Figure 1.

*Initially label all internal nodes with gene orders*
*Repeat*
  *For each internal node* ***v*** *with neighbors* ***A***, ***B*** *and* ***C****, do*
  *Solve median problem on* ***A***, ***B***, ***C*** *to yield* ***m***
  *If relabeling* ***v*** *with* ***m*** *improves the tree score, then do it*
*Until no change occurs*

Figure 1. Tree Scoring Algorithm.

As shown in Figures 2 and 3, the time required to perform a median computation using the current algorithm is an exponential function of the sum of edge distances between the three input gene orders and their corresponding optimal median (i.e. the *diameter* of the inputs). As a consequence, the portion of GRAPPA's total execution time that is spent labeling the internal vertices of candidate trees sharply increases with the evolutionary rate the inputs. In practice, even moderately distant input sets will cause GRAPPA's to spend over 99.9% of its total execution time computing medians.

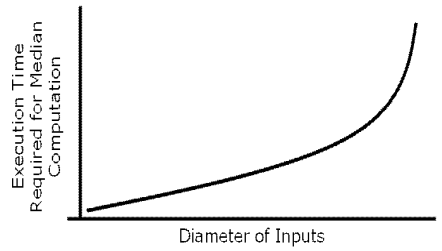


Figure 2. Execution time for the median computation as a function of input evolution rate.

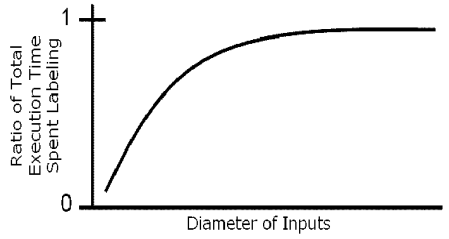


Figure 3. The relative amount of total execution time that GRAPPA spends labeling internal vertices (performing median computations) increases asymptotically to 100% with the diameter of the input set.

So far, GRAPPA provides several median solvers to choose from, including several breakpoint median solvers as well as Caprara's [19] and Siepel's [6] inversion median solver. The breakpoint median is generally considered to be an outdated technique for computing ancestral genomes. However, because both the inversion median and the breakpoint median rely on the same fundamental class of computation, we have decided to target the breakpoint median in our initial study. This will serve to demonstrate whether FPGA acceleration can yield significant performance benefits for such computations without the immediate need to implement the additional complexity of the inversion median. In the next phase of our work we will implement a hardware design for a parallelized inversion median using similar techniques to the ones we present in this paper. We expect the performance characteristics of the hardware breakpoint median to closely match those of the inversion median.

## III. High-Performance Reconfigurable Computing

In the past several years, high-performance reconfigurable computing (HPRC) has emerged as a promising new direction for providing inexpensive and efficient HPC platforms. In the HPRC computational model, often repeated computations are off-loaded to an FPGA-based application co-processor. Historically, FPGAs provide a technique for improving the performance of applications that contain more inherent fine-grain parallelism than can be exploited by the fixed number of functional units within a general-purpose microprocessor.

HPRC research has yielded many achievements over the years. For example, many control-independent dataflow-based arithmetic computations (such as FFTs [20], convolution/filtering [21], matrix multiplication [22], and encryption [23]) have been implemented on FPGAs and have been shown to achieve an order of 100X to 1000X speedup relative to a software-only implementation.

HPRC techniques have not, as of yet, been widely applied to computational biology, but there has been recent work in adapting sequence alignment for FPGA implementation (i.e. [24]). There is also recent commercial interest in applying HPRC to computational biology. Progeniq [25] has recently launched its BioBoost line of add-on FPGA boards which claim to achieve a 100X speedup for applications such as ClustalW, SmithWaterman, HMM, and BLAST. Although these applications analyze sequence data only, it does indicate interest in applying HPRC to computational biology.

## IV. Breakpoint Median Algorithm

The breakpoint distance between genomes $A$ and $B$ is defined as the number of adjacent gene-pairs $gh$ that appear in $A$ when neither $gh$ nor $-h$-$g$ appear in $B$. For example, (circular) genomes $A$=(1 -2 -3 4) and $B$=(4 2 -1 -3) have a breakpoint distance of 2, because gene pairs (-2 -3) and (4 1) appear in $A$ but neither {(-2 -3) or (3 2)} nor {(4 1) or (-1 -4)} appear in $B$.

As shown in Figure 4, computing a breakpoint median for three genomes requires solving a traveling salesman problem (TSP) formulated in the following way [10]. Given genomes $A$, $B$, and $C$, each consisting of an ordering of $n$ signed genes, construct a fully-connected undirected graph having vertices = ($-g_n$, …, $-g_1$, $g_1$, …, $g_n$) where $w(g,h)$ is defined as the weight of (undirected) edge ($g$,$h$). For each gene $g$, w($g$,$-g$) = $-\infty$, guaranteeing that each gene will appear alongside its reverse polarity counterpart in the TSP solution. Define $u(g,h)$ to be the number of times vertices $-g$ and $h$ are adjacent in the three genomes, and define $w(g,h) = 3 - u(g,h)$. If $s_1$, $-s_1$, $s_2$, $-s_2$, …, $s_n$, $-s_n$ is the solution of the TSP, then the resultant breakpoint median is $m = s_1, s_2, \ldots, s_n$. This solution guarantees that $d(A,m) + d(B,m) + d(C,m)$ is optimally minimal where $d(a,b)$ is the breakpoint distance.

As shown in Figure 5, the breakpoint median algorithm bundled with GRAPPA performs a depth-first branch-and-bound search of the space of all possible paths through the graph formed by the three input genomes. Its implementation was carefully designed to utilize the small number of choices for the pairwise costs, thus further significant speed-up on the software implementation is very difficult.

The search algorithm begins by reading the input genomes and constructing the resultant graph. By definition, each edge in the graph has weight $-\infty$, 0, 1, 2, or 3. Once this is complete, it organizes the weight 0, 1, and 2 edges into a list sorted by edge weights. Note that a weight-0 edge is equivalent to three

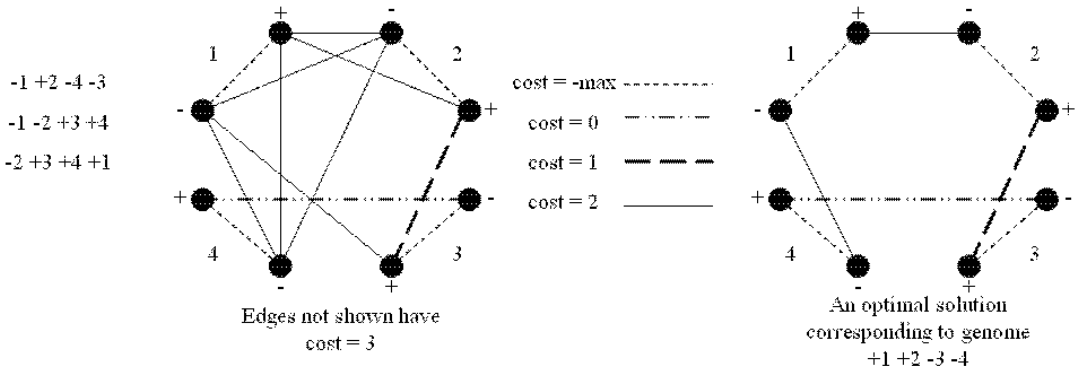


Figure 4. Breakpoint median TSP formulation.

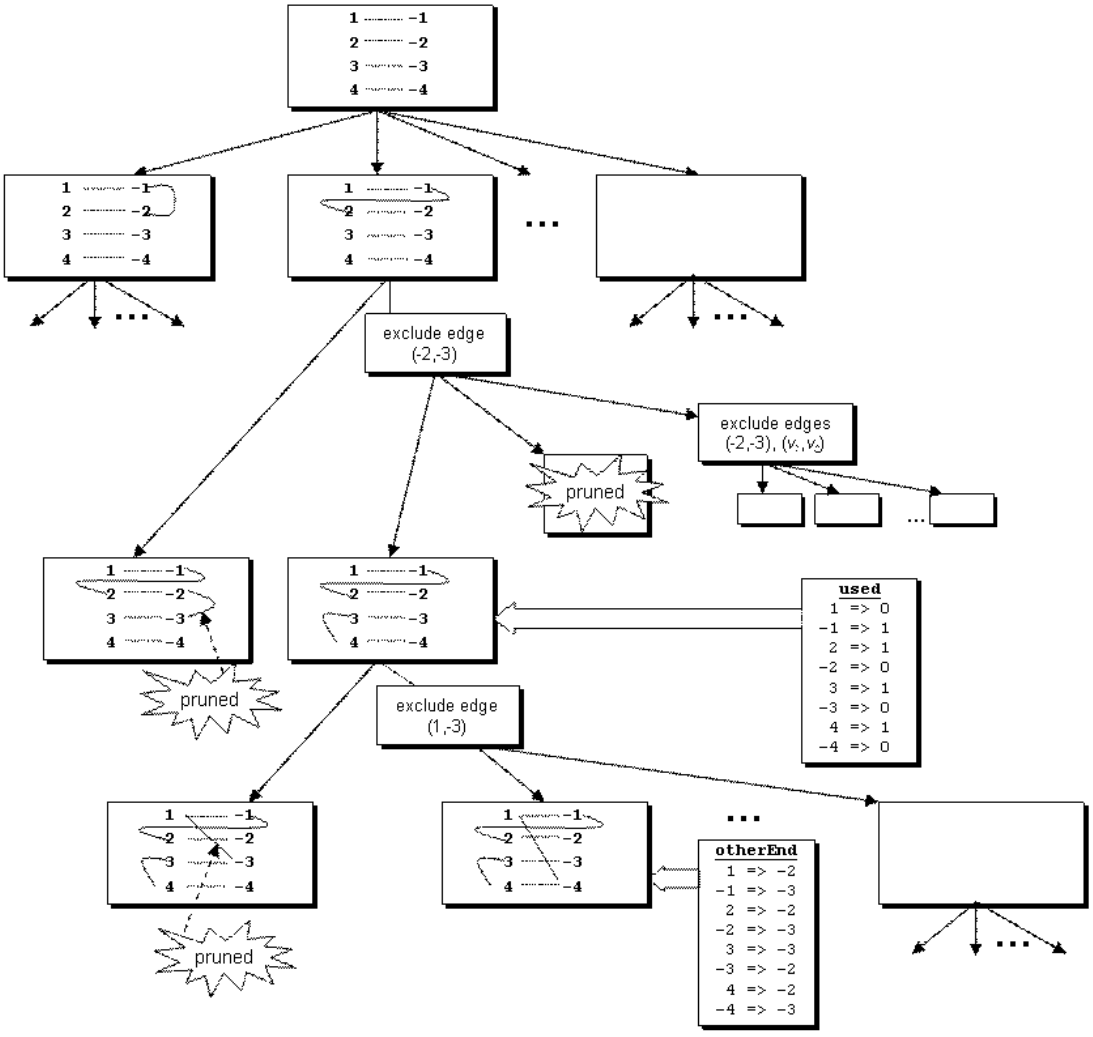


Figure 5. Graphical representation of a breakpoint median TSP depth-first search tree and associated data structures. Pruned edges are excluded from the lower bound computation from the level they are pruned to the bottom of the tree, the "otherEnd" array stores TSP path end-points to prevent cycles that do not include all vertices, and the "used" array keeps track of which vertices in the current solution state have degree 2.

parallel weight-2 edges and a weight-1 edge is equivalent to two parallel weight-2 edges. As such, in this representation, more distant input genomes yield longer sorted edge lists and thus require more searching time to find an optimal combination of edges. Note that the relative ordering of equal-weight edges dictates the search order of the algorithm, and these orderings have a significant impact on the runtime of the search. Unfortunately, to our knowledge it is not possible to determine the most optimal relative ordering of equal-weight edges

The algorithm creates an empty edge set to serve as the current search state, which we refer to as the *partial solution*. All the edges with weight $-\infty$ are assumed to be included in this set, making every vertex have a degree of one in the current partial solution.

The search iterates through the sorted edge list in order and adds any edge to the solution set that obeys two conditions. First, the edge must not cause any of the vertices in the graph implied by the current partial solution to have a degree of greater than two (since the salesman tour must not contain branches). Second, the edge must not create a cycle in the current partial solution unless the addition of this edge results in a full tour.

If no edges remain that satisfy these conditions from the current point forward in the list, the algorithm will record the path implied by the partial solution as a best-found-so-far solution if its score (including any weight-3 edges that must be included to complete the tour) is less then the current upper bound. Either way, the search prunes the last added edge and begins iterating from the edge immediately following the last added edge in the list. The search terminates when it exhausts the search space.

Each time the search adds a new edge, it computes a lower bound for the partial solution. If the lower bound exceeds the score of the upper bound, it prunes the last added edge.

The search computes the lower bound using the following technique [16]. First, initialize the lower bound to zero. Then, for each vertex that currently has a degree of one in the current

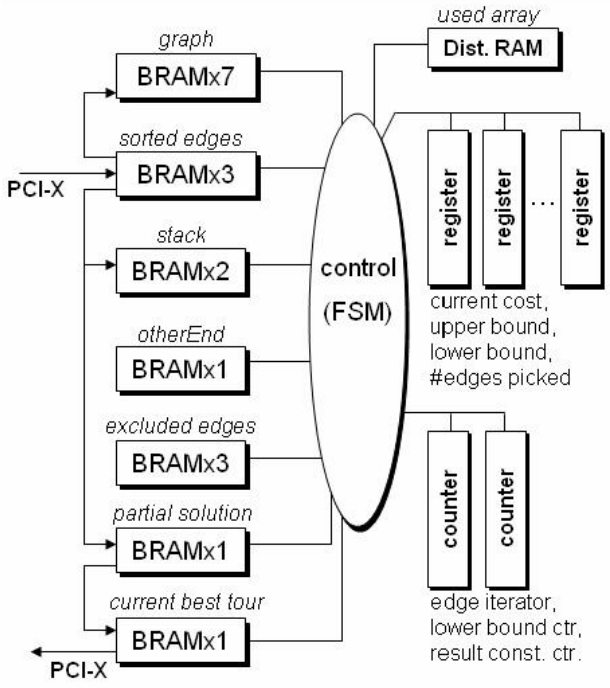


Figure 6. Simplified block diagram for the breakpoint median core. The core design is a large sequential logic circuit that establishes datapaths among several memory elements in each clock cycle. Static interconnects between memory elements are shown, while multiplexed interconnects among memory elements are established through the control unit (integrated within its output logic). Note that the resource requirements shown assume two lower bound units.

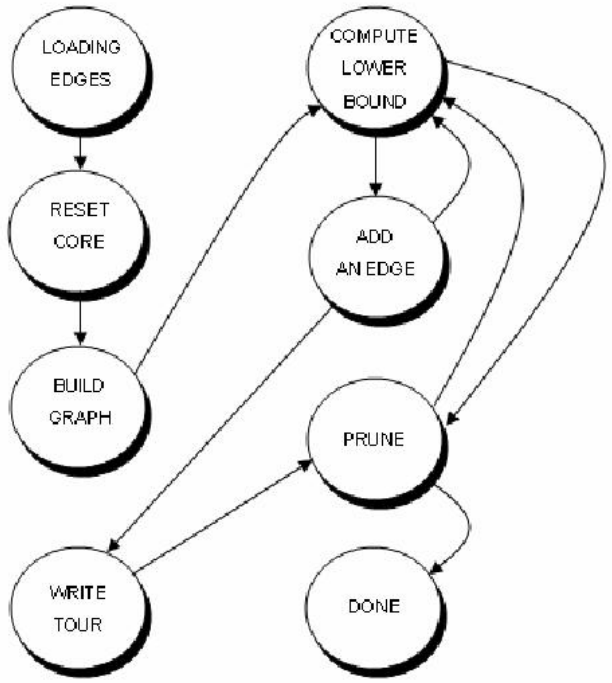


Figure 7. Finite state machine representation of median core controller.

partial solution, add the weight of the lowest weight edge that leads to another vertex of degree one. This technique adds twice as many edges as required, so after adding all valid edges, divide this value by two and add this value to the cost of the current partial solution.

The lower bound computation disregards any edges that were previously pruned at or above the current level in the search tree. It also disregards any edges that would result in a tour cycle if that edge were added to the partial solution (unless the cycle includes all vertices).

Each time the search prunes an edge, the search re-computes the lower bound because the exclusion of the pruned edge constitutes information that was not available before the search added the pruned edge originally.

## V. Breakpoint Median Core Design

Although there has been previous work in designing FPGA architectures for the TSP problem, to our knowledge all of this work involved approximate solvers (i.e. [26]). Since our goal is to find exact solutions of the breakpoint medians using branch-and-bound searches, this previous work is not applicable to this application.

FPGA designs are implemented by writing code in a high-level programming language called hardware description language (HDL). HDL differs from traditional high-level languages in that it has concurrent semantics, where each statement is executed when the value of any inputs change regardless of the statement's relative location in the code. In addition, verifying the functionality of HDL follows the same model as verifying a digital logic circuit -- a discrete-event simulator is used with a waveform-based interface.

We used custom-written VHDL to design our breakpoint median core. It. implements the same basic breakpoint median algorithm as the one bundled with GRAPPA with a few notable differences. GRAPPA's breakpoint median core relies on recursion such that its depth-first search is realized using the program activation stack. In order to achieve similar run-time behavior, we have implemented a stack memory using an on-chip block RAM (BRAM). The median core uses this stack to keep track of the information required to restore the state of the search when a branch of the search tree is pruned. This includes the added edge indices, excluded (pruned) edges, and the previous state for the "otherEnd" memory.

As shown in Figures 6 and 7, the median core design consists of a single block of control logic that is interconnected to a set of on-chip block RAMs (BRAMs) and registers that are used to store the state of the search. The controller is designed as a finite state machine (sequential logic circuit) with integrated multiplexers that establish datapaths among the set of BRAMs and registers. The median core is capable of computing breakpoint medians of any reasonable size using only on-chip memory.

Before the median core begins operation, the host system performs various startup tasks in software. These include the computation of the initial upper bound, construction of the TSP graph from the input genomes, and organizing the weight-0, weight-1, and weight-2 edges in the sorted edge list. The

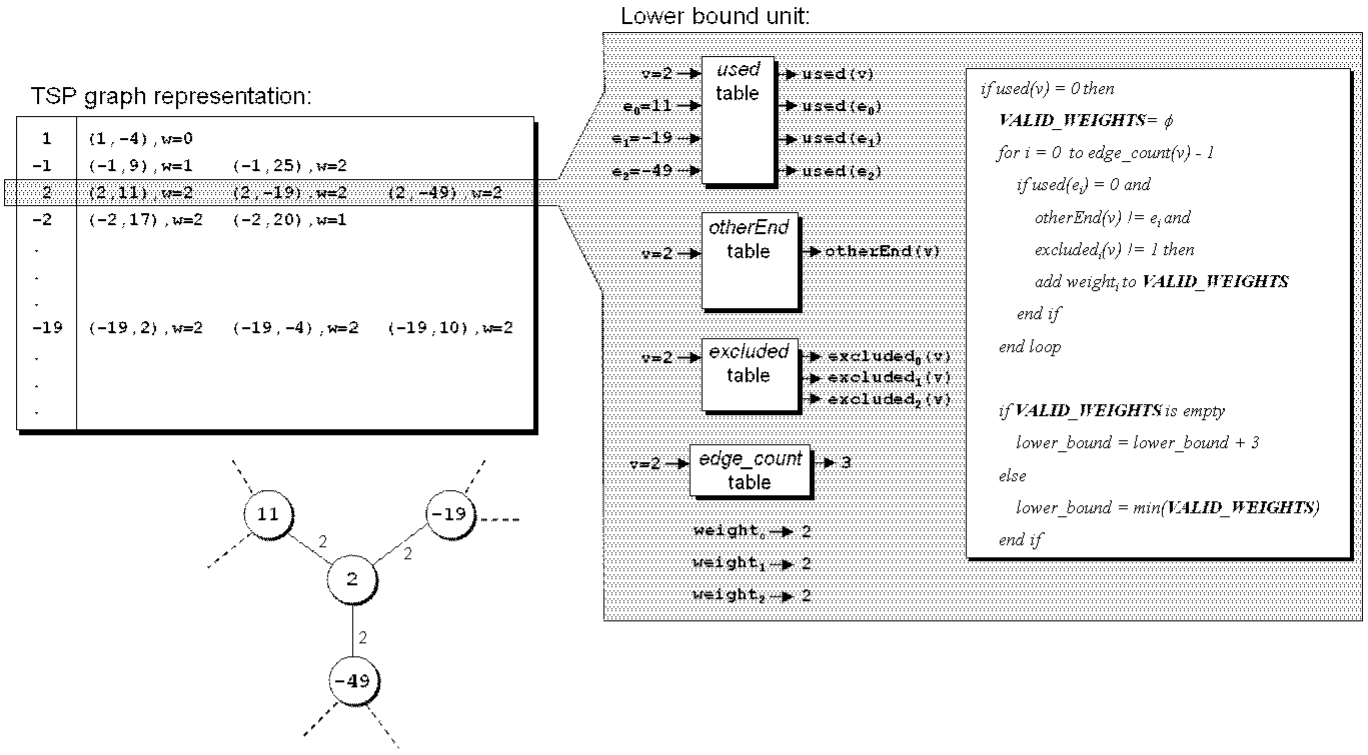


Figure 8. Illustration of the operation of a lower bound unit and a portion of the TSP group focusing on vertex 2. During the lower bound computation, the TSP graph (contructed from the sorted edge list) is scanned in parallel by mutliple lower bound units. In this example, a lower bound unit is inspecting vertex 2, which has three weight-2 edges to verteces 11, -19, and -49. Since vertex 2 is unused in the current solution state, the lower bound unit must add to the lower bound value the minimum edge weight of the edges that (1) lead to another vertex that is not used, (2) do not form a tour cycle (unless the cycle includes all verteces), and has not been excluded (pruned at or above the current level in the search tree).

software driver then uses a programmed I/O write operation to transmit the sorted edge list and initial upper bound into a specific set of on-chip memory corresponding to a specific median core on the FPGA (the core will later reconstruct the graph from the sorted edge list, as this extra initialization is less expensive than the additional I/O overhead of transferring the graph to the core). Using a programmed I/O read operation, the host can poll any core to determine its execution state, allowing the host to determine when any specific core has completed computation. When this occurs, the host performs another programmed I/O read operation to read the result genome from the core.

## VI. Extracting Parallelism from the Breakpoint Median

FPGAs allow fine-grain parallelism to be extracted from applications. In many cases, this involves performing many independent arithmetic operations in parallel, as is naturally possible when performing many types of matrix operations. FPGAs can also exploit course-grain parallelism. In this case, the FPGA implements multiple independent "cores" that each execute a thread, as is the case for SMP and cluster machines. In this application, we exploit both fine- and course-grain parallelism.

### A. Fine-Grain Parallelism

In the breakpoint median algorithm, adding or pruning an edge is an inexpensive operation and requires a small, fixed amount of time (2-8 clock cycles for the median core). However, after any edge is added or pruned, the core must perform a lower bound computation that requires a traversal of the entire TSP graph.

Aside from the FPGA-host communication overheads (which can be significant), the median core spends nearly all of its execution time performing the lower bound computation. Fortunately, as described above, the lower bound computation consists of a bounded loop and each loop iteration is data-independent. As such, the lower bound contains much fine-grain parallelism that we exploit in the median core design.

As illustrated in Figure 8,, the lower bound computation is parallelized by duplicating both the TSP graph and the search state into multiple memories that can be read in parallel. Essentially, we parallelized the lower bound computation by "unrolling" the lower bound loop and inspecting multiple graph vertices in each clock cycle. Since each of the FPGA's on-chip memories has two ports, we need to replicate copies of the necessary memories $n/2$ times (each memory maintained as exact copies), allowing the design to perform $n$ reads per clock cycle. We refer this as a core having $n$ "lower bound units".

This approach can technically be scaled up to the point where all graph vertices are read simultaneously, allowing the lower bound to be computed in a single clock cycle (assuming sufficient memory resources). However, we have found that scaling beyond twenty lower bound units adds significant routing complexity. Therefore, we use twenty lower bound units in our current design.

### B. Course-Grain Parallelism

The median computation also has potential for course-grain parallelism, i.e. the ability to use parallel median cores to perform a single median computation. Our current technique for exploiting course-grain parallelism relies on two concepts.

The first concept is to force each core to explore an identical TSP search space using a unique search order. To do this, we initialize each core with a consistent sorted edge list but with a unique ordering. In other words, each sorted edge list represents the same graph and is sorted by edge weight, but in each list equal-weight edges are arranged in different relative orders.

The second concept is to allow the cores to communicate with each other in order to maintain a global minimum upper bound value. Before each core computes its lower bound, it compares its current local upper bound with the minimum upper bound among all parallel cores. If this global upper bound is less than the core's local upper bound, the core adjusts its local upper bound to become the global upper bound plus one. Adding one to the local copy of the global minimum prevents any core from pruning the optimal solution. In this case, all cores will eventually find the optimal solution, so the FPGA-based median computation is considered complete when the first core completes its search

Although we are able to synthesize four cores on our current FPGA, our experimental results indicate that we cannot achieve further performance benefit beyond two cores using our current technique to extract course-grain parallelism.

## VII. Characterizing the Breakpoint Median Core

Our test system consists of a Dell Precision 650 server containing a 3.06 GHz Intel Pentium Xeon- processor. The FPGA accelerator card is an Annapolis Microsystems Wild-Star II Pro card with a single Xilinx Virtex-2 Pro 100 FPGA. It is connected to the host though a PCI-X interconnect.

In order to determine the hardware speedup, we generated 1000 random three-leaf phylogenies and extracted the leaves to use as median inputs. The number of rearrangement events along each edge for each phylogeny is chosen from a uniform random distribution with range *distance* +/- 2, where *distance* is a parameter. We performed these tests for a genome size of 100 genes.

For each set of genomes, we invoke GRAPPA's breakpoint median routine **bbtsp** and record its execution time. We then dispatch the same three genomes to the FPGA's breakpoint median architecture and record its execution time. Note the FPGA execution time includes the CPU-to-FPGA communication time and the time required for the host to construct the TSP graph, construct the corresponding sorted edge list(s), and compute the initial best score (all of which occurs in software).

Speedup is measured in the traditional way, i.e. $time_{sw}$ / $time_{hw}$. A speedup of 1 would indicate equivalent performance between the software median computation and hardware median computation. Our results list the arithmetic mean of the individual speedups relative to software for the set of 1000 individual median computations for each input distance:

$$speedup = \frac{\sum_{i=1}^{1000} time_{sw}(i)}{\sum_{i=1}^{1000} time_{hw}(i)}$$

, where $time_{hw}(i)$ represents the hardware execution time of input data $i$. Table 1 lists these results.

TABLE I. Performance Results for the Median Core.

| Average Events per Edge | Average Median Speedup |
|---|---|
| **17** | 11.468 |
| **18** | 12.833 |
| **19** | 21.730 |
| **20** | 44.799 |
| **21** | 51.936 |
| **22** | 90.871 |
| **23** | 136.28 |
| **24** | 153.724 |
| **25** | 876.304 |

Our performance results indicate that the median speedup increases exponentially with the evolutionary rate of the inputs, with lower-rate inputs achieving a one order-of-magnitude speedup and higher rate inputs achieving a two-order-of-magnitude speedup. It is clear from these results that the median architecture has a faster search rate (when searching for the optimal TSP tour), but suffers a high penalty in host-FPGA communication overhead. Since this overhead is fixed for any inputs, median computations that require longer searches spend a larger relative amount of time searching and thus enjoy a higher speedup. This relative amount of time increases exponentially with the number of non-weight-3 graph edges, which is a function of the evolutionary rate of the inputs.

## VIII. Accelerated-GRAPPA

We made several modifications to the GRAPPA code to accelerate the tree scoring procedure by forcing it to dispatch its median computations to the median cores on the FPGA.

Table 2 shows our average speedups for entire GRAPPA runs over 10 unique 8-leaf, 100-gene synthetic datasets. The input sets were produced by synthesizing phylogenies using a specified average edge distance. The leaves are extracted for use as inputs. The speedup for each experimental run was computed as $time_{SW}$ / $time_{HW}$). The results shown are the arithmetic mean of the individual speedups relative to software over each set of 10 GRAPPA runs for each input distance, as described for the breakpoint median performance results.

As with the breakpoint median performance results, the results show a clear trend where the average speedup increases with the evolution rate of the input set. However, these results are even more sensitive to the input set's evolution rate. There are two reasons for this. First, higher evolutionary rate input sets force GRAPPA to spend higher portions of its execution

time computing medians. In other words, more difficult data sets force the median computation to become more significant a bottleneck. Thus accelerating the median computation has a higher impact on overall application speedup. Second, the median computations themselves are more greatly accelerated as the diameter of the median inputs increase. Speedup results range from 5X to 189X as the average input diameter increases.

TABLE II. PERFORMANCE RESULTS FOR ACCELERATED-GRAPPA.

| Average Events per Edge | Average Application Speedup |
|---|---|
| **10** | 5.604 |
| **11** | 9.431 |
| **12** | 129.097 |
| **13** | 189.273 |

## IX. CONCLUSIONS AND FUTURE WORK

Our results indicate that Accelerated GRAPPA is capable of achieving an order 100 speedup for input sets that have a relatively large diameter (high evolution rate).

We are currently developing a tree generation and bounding core that performs tree space exploration. Our current design requires only two BRAMs, indicating that it is possible to implement approximately 100 parallel tree generation cores on a single Virtex-2 Pro 100. Since this organization matches the behavior of GRAPPA in cluster mode, we refer to this approach as "cluster-on-a-chip". Our next goal is to combine tree generation and bounding cores with median cores on a single FPGA, allowing candidate trees from any of the tree generation and bounding cores to be scored with median cores on the same FPGA.

## REFERENCES


[1] M. Blanchette, T. Kunisawa, and D. Sankoff, "Parametric genome rearrangement," Gene, 172, GC11--GC17, 1996.

[2] J. Felsenstein, J.,"The number of evolutionary trees," Systematic Zoology 27, 27–33, 1978.

[3] R.Olmstead, J. Palmer, "Chloroplast DNA systematics: a review of methods and data analysis," Amer. J. Bot. 81, 1205–1224, 1994.

[4] G. Bourque, P. Pevzner, "Genome-scale evolution: Reconstructing gene orders in the ancestral species," Genome Research 12, 26-36, 2002.

[5] D.A. Bader, B.M.E. Moret, M. Yan, "A fast linear-time algorithm for inversion distance with an experimental comparison," J. Comput. Biol. 85, 483-491.

[6] A. Siepel, B.M.E. Moret, "Finding an optimal inversion median: experimental results," 1st Workshop on Algs. in Bioinformatics (WABI'01), Volume 2149 of Lecture Notes in Computer Science, 189--203.

[7] B.M.E Moret, J. Tang, L.-S. Wang, T. Warnow, "Steps toward accurate reconstructions of phylogenies from gene-order data," Comput. Syst. Sci., 65(3), 508-525, 2002.

[8] B.M.E. Moret, J. Tang, T. Warnow, "Reconstructing phylogenies from gene-content and gene-order data," Mathematics of Evolution and Phylogeny, O. Gascuel, ed., Oxford Univ. Press, 321--352, 2005.

[9] D. Huson, S. Nettles, and T. Warnow, "Disk-covering, a fast converging method for phylogenetic tree reconstruction," J. Comput. Biol. 6(3), 369—386, 1999.

[10] J. Tang, B.M.E. Moret, "Scaling up accurate phylogenetic reconstruction from gene-order data," Proc. 11th Conf. on Intelligent Systems for Mol. Biol. ISMB'03, in Bioinformatics 19, i305–i312.

[11] Jason D. Bakos, "FPGA Acceleration of Gene Rearrangement Analysis," IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM 2007), April 23-25, 2007.

[12] J.H. Nadeau, B.A. Taylor, "Lengths of chromosome segments conserved since divergence of man and mouse," Proc. Nat'l Acad. Sci. USA 81 (1984), 814–818.

[13] N. Saitou, N. Nei, "The neighbor-joining method: A new method for reconstrucing phylogenetic trees," Mol. Biol. & Evol., 4:406-425, 1987.

[14] F. Ronquist, J. P. Huelsenbeck, "MrBayes 3: Bayesian phylogenetic inference under mixed models," Bioinformatics Vol. 19 no. 12 2003, pp 1572-1574.

[15] L. Wang, R. Jansen, B. Moret, L. Raubeson, T. Warnow. "Fast phylogenetic methods for genome rearrangement evolution: An empirical study," Proc. 7th Pacific Symp. On Biocomputing (PSB'02), 524--535. World Scientific Pub.

[16] M. Blanchette, G. Bourque, D. Sankoff, "Breakpoint phylogenies," S. Miyano and T. Takagi, editors, Genome Informatics 1997, pages 25-34, Univ. Academy Press, Tokyo, 1997.

[17] G. Bourque and P. Pevzner, "Genome-scale evolution: Reconstructing gene orders in the ancestral species," Genome Research 12, 26—36 2002.

[18] I. Pe'er, R. Shamir,"The median problems for breakpoints are NP-complete," Elec. Colloq. on Comput. Complexity, 71, 1998.

[19] A. Caprara, "On the practical solution of the reversal median problem," Proc. 1st Workshop on Algorithms in Bioinformatics, (WABI'01), Volume 2149 of Lecture Notes in Computer Science, 238-251, 2001.

[20] K.S. Hemmert, K.D. Underwood, "An analysis of the double-precision floating-point FFT on FPGAs," Proc. 13th Annual IEEE Symp. on Field-Programmable Custom Computing Machines, 2005. FCCM 2005, 18-20 April 2005 p. 171 - 180.

[21] F. Cardells-Tormo, P.-L. Molinet, "Area-efficient 2-D shift-variant convolvers for FPGA-based digital image processing," IEEE Trans. on Circuits and Systems II: Express Briefs, Vol. 53, Issue 2, Feb. 2006 p. 105 - 109.

[22] J.-W. Jang, S.B. Choi, V.K. Prasanna, "Energy- and time-efficient matrix multiplication on FPGAs," IEEE Trans. on Very Large Scale Integration (VLSI) Systems, Vol. 13, Issue 11, Nov. 2005, p. 1305 - 1319.

[23] E. Allen Michalski, D.A. Buell, "The Scalable Architecture for RSA Cryptography on Large FPGAs," Proc. 16th Int'l Conf. on Field Prog. Logic and Appl. (FPL 2006), Madrid, Spain, August 28-30, 2006.

[24] E. Sotiriades, C. Kozanitis, A. Dollas, "FPGA based architecture for DNA sequence comparison and database search," 20th International Parallel and Distributed Processing Symposium, 2006. IPDPS 2006, 25-29 April 2006.

[25] http://www.progeniq.com, Feb. 2007.

[26] I. Skliarova, A. B. Ferrari, "FPGA-Based Implementation of Genetic Algorithm for the Traveling Salesman Problem and Its Industrial Application," Proc. Applications of Artificial Intelligence and Expert Systems, IEA/AIE 2002, Cairns, Australia, June 17-20, 2002.