

Accelerating Phylogenetics Computing on the Desktop: Experiments with Executing UPGMA in Programmable Logic

J. P. Davis, S. Akella¹, P. H. Waddell²

¹Department of Computer Science and Engineering, University of South Carolina, Columbia, SC, USA

²Departments of Biology and Statistics, University of South Carolina, Columbia, SC, USA

Abstract— Having greater computational power on the desktop for processing taxa data sets has been a dream of Biologists/Statisticians involved in Phylogenetics data analysis. Many existing algorithms have been highly optimized—one example being Felsenstein's PHYLIP code, written in C, for UPGMA and Neighbor Joining algorithms. However, the ability to process more than a few tens of taxa in a reasonable amount of time using conventional computers has not yielded a satisfactory speedup in data processing, making it difficult for Phylogenetics practitioners to quickly explore data sets—such as might be done from a laptop computer. In this paper, we discuss the application of Custom Computing techniques to Phylogenetics. In particular, we apply this technology to speed up UPGMA algorithm execution by a factor of a hundred, against that of PHYLIP code running on the same PC. We report on these experiments and discuss how Custom Computing techniques can be used to not only accelerate Phylogenetics algorithm performance on the desktop, but also on larger, high-performance computing engines, thus enabling the high-speed processing of data sets involving thousands of taxa.

Keywords— Phylogenetics, UPGMA Algorithm, Custom Computing, VLSI Integrated Circuits, Programmable Logic.

I. INTRODUCTION

The study of the relationships between groups of organisms is called *taxonomy*, an ancient and venerable branch of classical biology. The branch of taxonomy that deals with numerical data such as DNA sequences is known as *Phylogenetics*. Biological systematists who wanted to reconstruct evolutionary genealogies of species based on morphological similarities originally developed phylogenetic analysis. The results of phylogenetic analysis may be depicted as a hierarchical branching diagram, a "cladogram" or "phylogenetic tree" as shown in Fig. 1 [1].

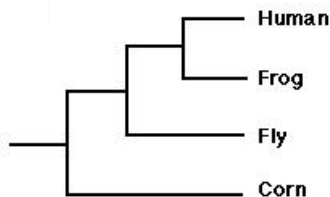


Fig. 1: A phylogenetic tree showing a relationship between four species [1]

The tree represents the genealogical evolution of the different species, linking them through a certain set of similarities and differences. Similarities and differences between organisms can be coded as a set of characters, each with two or more alternative character states. In an alignment of DNA sequences, for example, each aligned site is a separate character, each with four character states, corresponding to the four DNA nucleotides adenine, thymine, cytosine, and guanine.

All the trees are assumed to be binary, meaning that each node branches into two daughter edges as shown in Fig. 1. The edges meet at a branch node, a node being at the endpoint of an edge. Each edge has a certain amount of evolutionary divergence associated with it, quantified by some distance between DNA sequences. These distances are referred to as 'edge lengths' or 'branch lengths'. Terminal nodes or leaves correspond to the observed sequences that might connect up to an ultimate ancestor or 'root' of the tree. A true biological phylogeny has a 'root' but only some phylogenetics algorithms provide information about the location of the root.

For a specific set of n leaves, the nodes and edges of a tree can be counted as follows: There would be $(n-1)$ nodes in addition to the n leaves, giving a total of $(2n-1)$ nodes and one fewer edges, that is $(2n-2)$, discounting the edge above the root node.

The UPGMA (Unweighted Pair-Group Method with Arithmetic means) algorithm is an example of a *distance method*, one of several classes of Phylogenetics problem formulations being studied [2], whose aim is to find a tree whose path distance matches closely to observed distances. UPGMA has relevancy beyond phylogenetics, since it is a hierarchical clustering method that is both fast and useful with gene expression or micro-array data. The algorithm's running time complexity (asymptotic worst case) is of the order $O(N^3)$.

UPGMA's behavior is well understood [2, 3], and the software implementations have reached a level of optimization beyond which only minimal performance improvement is likely to be obtained. Other more optimal algorithms are currently favored for distance methods [3]. UPGMA was thus chosen for implementation in programmable logic using VLSI custom-logic architecture and design techniques primarily because it has been so optimized. It has been our intention to demonstrate the efficacy of using Custom Computing techniques in Phylogenetics by extracting much greater performance from this algorithm than has been possible before.

II. METHODOLOGY

An algorithm can be thought of as a set of processing steps for transforming data by executing a series of computations [4]. The algorithm needs to be interpreted by a machine to perform the work. Choosing the elements that make up the machine defines its architecture, and this necessitates looking at different architectural design approaches.

Custom Computing (sometimes referred to as *reconfigurable computing* or *adaptive computing*) is a systems development approach that uses VLSI programmable logic devices as the means for implementing algorithms. Rather than writing algorithms in a software language and executing them on a CPU running an operating system (such as the Intel Pentium®-based PC running Microsoft Windows®), smaller Custom Computing platforms generally plug into a PC's card slot as "compute accelerators", where a data stream is fed to an add-in board, and high-speed computations are carried out.

These computations can be performed much faster than on the host PC because the "logic" on the accelerator has been designed specifically for the application computing problem at hand. Thus, Custom Computing modules--and the FPGA devices on which such applications are built--offer a good medium for implementing complex computational tasks having high throughput and low latency requirements, providing orders of magnitude speedup in application processing at a fraction of the cost per processing operation. Many computational tasks in different application domains have been implemented and evaluated on Custom Computing systems [5, 6, 7].

III. ARCHITECTURE

In analyzing the UPGMA algorithm, we identified two bottlenecks where we believed we could improve performance by implementation in VLSI programmable logic. The first is in deciding which of the $N(N-1)/2$ pairwise distances is minimal at each step of the star-decomposition clustering. Following this, the data matrix is reduced by dimension 1, due to clustering of two objects. This introduces the second bottleneck, the need to calculate an average distance between the two objects, (i and j) as a single cluster (k), and all other objects. This involves complex computational units, costly on general-purpose microprocessors, but which we posited could be implemented efficiently in custom logic on an FPGA programmable logic device, allowing us to obtain better performance results.

In considering the principal data structure of this algorithm—a matrix—we define the distances between two clusters C_i and C_j to be the average distance between pairs of sequences from each cluster:

$$d_{ij} = (1/|C_i||C_j|) \sum d_{pq} \quad (1)$$

$|C_i|$ and $|C_j|$ denote the number of sequences in clusters i and j, respectively and p and q denote the sequences in each cluster C_i and C_j respectively. If C_k is the union of clusters C_i and C_j , and if C_l is another cluster, then

$$d_{kl} = (1/|C_i||C_j|)(d_{il}|C_i| + d_{jl}|C_j|) \quad (2)$$

This forms the average distance calculation for obtaining the distance of the new cluster C_k to the any other cluster C_l .

The distances are represented in the form of a matrix given in Fig. 2, with each row or column corresponding to one node. The nodal distance between node i, j would be in the position [i, j] of the matrix. So, $D[i, j]$ would form the distance between nodes i and j.

x	6	8	3
6	x	7	9
8	7	x	4
3	9	4	x

Fig. 2. Example UPGMA distance matrix.

$D[i, i]$ is not a valid distance since there can be no distance between the same node. This is therefore marked as "x" in the matrix.

The steps of UPGMA algorithm are as given below [2]:

- Initialization:
 - Assign each sequence i to its own cluster C_i ,
 - Define one leaf at each T for each sequence, and place at height zero
- Iteration:
 - Determine the two clusters i, j, for which d_{ij} is minimal. (if there are several equidistant pairs pick one randomly.)
 - Define a new cluster k by $C_k = C_i \cup C_j$, and define d_{kl} for all l by (2).
 - Define a node k with daughter nodes i and j, and place it at height $d_{ij}/2$.
 - Add k to the current clusters and remove i and j.
- Termination:
 - When only two clusters i, j remain, place the root at height $d_{ij}/2$.

We now consider the design of the data path and control portions of the UPGMA architecture. The datapath is broken into two graphs, one used for finding the least distance, or minima, and the other for calculating the average distances.

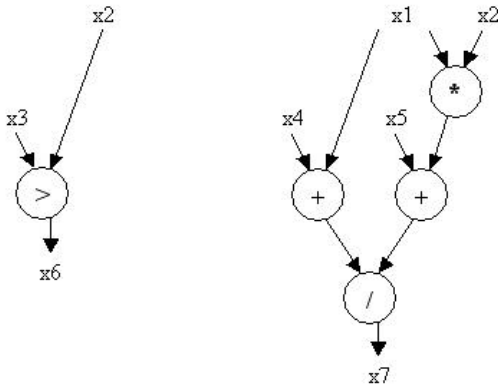


Fig. 3. UPGMA algorithm data flow graphs.

The datapath on the left of Fig. 3, with the less-than operator, is used to find the minima. The graph has the distance value and the current minima as inputs. The current minima is stored in a register and is fed back into the comparator.

The second datapath on the right is used to calculate the average distance. It has the distance value d_{ik} and the height of node i as inputs. The Multiplier obtains the product of this height and the distance, sending it to the Adder unit, which adds this value to an Accumulator register. The Multiplier and Adder together obtain the numerator part of the average distance, given in equation (2) earlier. During the same time slice that the Multiplier Accumulator pair are computing the numerator, the second Adder, taking height H_i as input, computes the denominator of equation (2). Once these two are computed, the resulting values, as defined below, are sent to the Divider to obtain the average distance.

$$\begin{aligned} \text{Numerator} &= (d_{ik}h_i + d_{jk}) \\ \text{Denominator} &= h_i + h_j \\ \text{Average distance} &= (d_{ik}h_i + d_{jk})/(h_i + h_j) \end{aligned}$$

The architecture has a controller module implementing the coordinated sequencing of the datapath operations, as defined by the algorithm's control flow graph in Fig. 4. The three main operations performed by the controller for every single pass through the matrix are as follows:

- Find the new minima
- Compute the Average distance
- Reduce the matrix size

The controller accomplishes this by stepping through the set of defined algorithmic steps of Fig. 4, repeating the process until all the nodes in the tree have been processed.

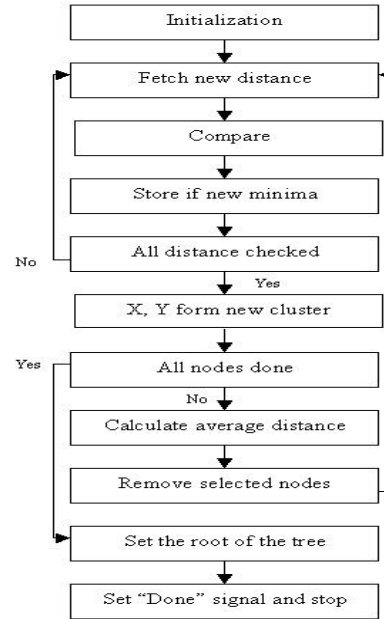


Fig. 4. The UPGMA algorithm control flow.

IV. IMPLEMENTATION

We implemented the functionality of the UPGMA algorithm as custom-logic on an Annapolis WILDCARD™ PCMCIA card that is plug-compatible into a laptop's Cardbus slot. A standard HDL-based design methodology is employed. We model the algorithm using the VHDL hardware description language, we functionally verify the algorithm's correctness in the custom logic architecture, and then we synthesize the architecture onto a set of resources to produce a circuit mapped to a target FPGA device's component library. The resulting circuit is implemented on a Xilinx Virtex E® series FPGA device and is subjected to functional and performance analysis.

The WILDCARD™ system comes as a PC card and can plug into a PCMCIA card slot adapter, making it a very portable low-end compute "accelerator". It has a very compact architecture, with a single Xilinx Virtex XCV300E processing element (PE) and two independent memory modules, one on the either side, forming the core of the system. The architectural block diagram is given in Fig. 5 below.

Each of the two memory blocks, referred to as the Right and Left memory banks, is a 64K x 32-bit RAM module, with a 19-bit address bus and a 32-bit data word. The PE can write and read from the right and left memories independently. The host interface is through a 32-bit CardBus (PCMCIA) controller that operates at a 33 Mhz clock frequency. The CardBus controller interfaces with the PC host through the PCI Bus interface, and with the PE through the LAD Bus interface [8].

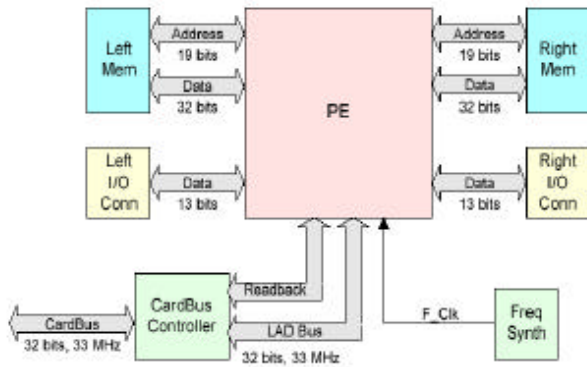


Fig. 5. The WILDCARD™ platform block diagram [8]

Data transfers to and from the PC host are done through control of a set of C program driver calls that interface with the CardBus controller which, in turn, interfaces with the LAD Bus to send data to, and retrieve data from, the PE. Data can be written from the host to the memory through these specific interfaces by making the C program calls provided by the PC host's Application Programming Interface (API) provided by the WILDCARD vendor.

The inter-nodal distances and output tree data are stored in the memory banks on the WILDCARD™ board. The read and write operations take a specific number of cycles to be performed successfully, with the read operation having a larger, four clock cycle latency that we found to adversely affect the performance of the design as the dataset size increased.

V. EXPERIMENTS AND RESULTS

The WILDCARD host-programming environment provides an API to program the board. The WILDCARD interface routines are used in a host program to perform the following functions: (1) read and write to the on-board SRAM memories; (2) wait for the Virtex® PE to interrupt (or, alternately, poll the status register for completion of a WILDCARD™ controller operation), and (3) process the results of the API-initiated operation.

A program written in C++ was created to generate the test data for testing the UPGMA implementation. It takes as input the following parameters: (1) the number of taxa; (2) the maximum value of inter-node distance; and, (3) the number of repetitions of a single distance value in the data set.

The test data were generated for taxa sizes of 10, 16, 32, 50, 64, 75, 100, 128, 150, 175, 200, 225 and 256. For each taxa size, ten different data sets were randomly generated for that number. Furthermore, each created data set had its data values subjected to permutations, creating up to 10 permutations per data set per number of taxa. This step was taken to insure that the resultant latency measurements did not wildly vary but, rather, converged around a mean value.

After actually running the experiments, the resultant data sets were tabulated and plotted in terms of a bounded clock cycle count using the clocking frequency of the host PC's CPU clock, giving us a count of the total number of host clock cycles for a given computation run. We use this, as opposed to using the on-board FPGA clock, as the former takes into account the communication overhead of getting data to and from the WILDCARD. This communication overhead is an important consideration when comparing the efficacy of using programmable logic over other possible approaches to speeding up the UPGMA application.

We took randomly generated data sets, permuted them, and executed them on the WILDCARD. We executed the UPGMA computation for these different data sets while also increasing the number of taxa considered in the input distance matrix. Our expectation was that permuting the taxa data would not affect latency values as borne out, because the time to perform actual computations in programmable logic on fixed-width data operands is independent of the actual data values passed.

In order to be able to make a comparison, the same data sets were executed on the same PC running the PHYLIP program [10, 11]. The data was collected, collated and plotted, as before. The Average number of clock ticks taken for each of the taxa data runs was tabulated for both implementations. Fig. 6 displays comparative plots of the averaged runs for both implementations.

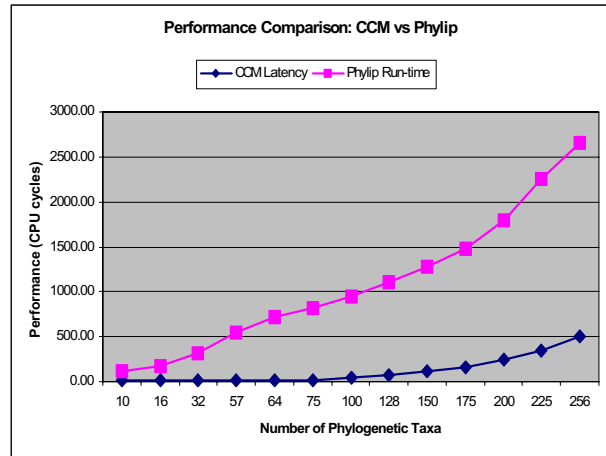


Fig. 6. Computing cycles plotted against number of taxa for both PHYLIP and the Custom Computing solution on the WILDCARD.

A 256 taxa maximum was set as an upper bound for the experiments, based on estimates of on-board memory requirements of the input matrix and output tree data structures. Within this range of taxa, it can be seen from the graph that the programmable logic implementation of the UPGMA algorithm provides significant speedup, at some taxa values delivering two-orders of magnitude in speedup over the PHYLIP software implementation. In addition,

from the graph data, it can be expected that the disparity in computation times between the software and hardware versions of the UPGMA algorithm would continue to be within the 50X to 100X range if we were able to extend the on-board processing capability beyond the 256 taxa limit.

The small size of the memory banks on the WILDCARD™ board not only limited the number of taxa being implemented on the system, but also adversely affected the addressing scheme for the memory. This necessitated exploring different addressing schemes and coming up with a workaround requiring modifications to the addressing scheme itself, which added to the cycle latency of the algorithm's design. For datasets of 74 taxa and above, the performance improvement between the two began to lessen as a result of this memory limitation, although the programmable logic implementation was still considerably faster. At its data size "sweet spot", however, the custom computing machine implementation in custom logic demonstrated a two-orders of magnitude improvement over the software implementation, showing that even with a low-end "compute" accelerator, considerable benefit in computation speed can be obtained.

VI. CONCLUSION

In this paper, we have presented the architecture, implementation and subsequent experimentation that has been conducted in applying a Custom Computing solution to Phylogenetics computation—namely through the design and implementation of an architecture for running UPGMA computations using a PCMCIA-compatible programmable logic board. The results presented in this paper provided us with an insight into the performance of both the hardware and software implementations, providing results that appear consistent with other such work in different application domains [5, 6, 7]. The resultant solution demonstrated significant performance improvement over a benchmark UPGMA software implementation based on execution of Felsenstein's PHYLIP code [11] on a Pentium-IV® class PC.

It has been our intention to demonstrate the efficacy of using Custom Computing techniques in the Phylogenetics domain by implementing the algorithm in programmable logic, thereby extracting much greater performance from this algorithm than has been possible before. Our results are consistent with other published results [5, 6, 7]. These results obtained from a "desktop" computing accelerator—having an additional hardware cost of about \$1000 above the base price of a Pentium® PC or laptop computer—demonstrate the price-performance value of Custom Computing to improve computational throughput and bring new life to older algorithms that have been deprecated in favor of newer ones. We see no reason why our results should not facilitate (1) scaling the existing UPGMA to larger taxa data sets, once we moved the application to a higher-performance Custom Computing machine, and (2)

extending the performance benefit to other Phylogenetics algorithms, making it possible to achieve similar improvements even as the number of taxa increase.

The next phase of experimentation with the UPGMA architecture will be to, first, migrate it to the SRC Computers SRC-6E®, which has a larger memory address space in addition to having two of the larger Virtex-II® FPGA devices. Second, once we move this basic architecture to the SRC platform, we will extend beyond this most basic of Phylogenetics algorithms to consider accelerated computing for parsimony, maximum likelihood and other phylogeny methods. Finally, we would expect to benchmark against a wider array of software implementations available for computing with the different algorithms.

REFERENCES

- [1] Peter H. Weston, Michael D. Crisp, "Introduction to Phylogenetic Systematics", *Invited Contributions of the Society of Australian Systematic Biologists*, SASB. Available <http://www.science.uts.edu.au/sasb/WestonCrisp.html>.
- [2] R. Durbin, S. Eddy, A. Krogh, G. Mitchison, Building phylogenetic trees, Chapter 7, *Biological Sequence Analysis*, pg 160-190. Cambridge University Press, 1998.
- [3] D.L. Swofford, G.J. Olsen, P.J. Waddell, and D.M. Hillis, Phylogenetic Inference, Chapter 11, *Molecular Systematics*, pg 45-572, 2nd edition, (ed. D.M. Hillis, and C. Mortiz), Sinauer Association, Sunderland, MA, 1996.
- [4] John V. Oldfield, Richard C. Dorf, "System Implementation Strategies", Chapter 1, *Field Programmable Gate Arrays, Reconfigurable logic for Rapid Prototyping and Implementation of Digital Systems*, pg 1-26, Wiley-Interscience Publishing, 1995.
- [5] Jeffrey Arnold, Kenneth L. Pocek, "Genetic Algorithms In Software and In Hardware - A Performance Analysis of Workstation and Custom Computing Machine Implementations", *Proceedings of IEEE symposium of Field Programmable Custom Computing Machines*, IEEE Computer Society, pp. 216-225, April 1996.
- [6] Andre DeHon, Wawrzynek, *The case of reconfigurable processors*. Berkeley Reconfigurable Architectures Systems, and Software. University of California, Berkeley. <http://citeseer.nj.nec.com/dehon97case.html>.
- [7] Duncan A. Buell, Jeffrey M. Arnold, Walter J. Kleinfelder, *SPLASH-2: FPGAs in a Custom Computing Machine*, IEEE Computer Society Press, 1996.
- [8] Annapolis Microsystems Inc, *Annapolis WILDCARD™ System Reference Manual*, Revision 2.6, 2003. www.annapmicro.com
- [9] SRC Computers Inc., <http://www.srccomputers.com>.
- [10] Felsenstein, J. 1989. PHYLIP -- Phylogeny Inference Package (Version 3.2). *Cladistics* 5: 164-166.
- [11] J. Felsenstein, *PHYLIP source code*, Department of Genome Sciences, University of Washington. Available: <http://evolution.genetics.washington.edu/phylip.html>