Custom Computing for Phylogenetics: Exploring the Solution Space for UPGMA

Sreesa Akella and James P. Davis
Department of Computer Science and Engineering
University of South Carolina
Columbia, SC 29208 USA
akella@engr.sc.edu, jimdavis@cse.sc.edu

Abstract

There is great interest in the use of custom computing machines (CCMs) to accelerate the performance of existing, highly optimized software algorithms in Bioinformatics. But, like many applications for CCMs in the sciences, there are many architecture and experimentation "data points" to be obtained before a solid understanding can be obtained as to the issues of migrating specialized scientific computing software applications to high-performance CCM platforms. This paper presents one such data point, namely, the design and comparative analysis of a custom computing application architecture for the UPGMA algorithm implemented on an FPGA-based platform. We present the UPGMA problem domain and explore design issues encountered in the transition from software algorithm executing on a standard PC platform to a hybrid custom compute "server" running a VLSI logic-based architecture for the algorithm. Using the low-cost Annapolis WildCard® platform, we discuss the CCM development method, and present an architecture created that takes into consideration the processing limitations of the platform. Given this implementation, we present results of experiments on system performance, as measured and compared against that of the benchmark UPGMA algorithm written in C, executing on a single-processor Pentium® PC.

1. Introduction

In recent years, custom computing has become more recognized as a possible means for achieving cost-effective, high-performance computing for applications in the sciences. With the reduction in price points of CCM platforms (relative to comparable HPC or supercomputing machines), and the availability of many commercial products—from add-in boards to entire systems—custom computing as a paradigm for realizing significant gains in computing throughput is attracting the attention of scientists across the disciplines. Specifically, in our work with Biologists and Statisticians involved in the analysis of Phylogeny (the study and construction of evolutionary trees, with potential applications in gene therapy and drug development), there is great excitement about the potential gains of using custom computing architectures and platforms with even modest levels of improvement in the processing speeds available using current algorithms [15].

However, with the excitement of low-cost, desktop high-performance computing comes uncertainty on the part of practicing Biologists regarding how to harness a custom computing platform to achieve significant taxa data processing improvement over that currently obtained in this domain through conventional programming. As cited in much of the CCM literature, the issue of making CCM "programming" amenable to non-hardware designers is a key objective in much of our collective research [1, 2, 17]. Many efforts focus on the creation of compilers that will transparently take a C application and partition it between the host and custom compute engine components of a hybrid CCM platform [17].

A key question in our minds is whether a compiler can know how to generate a target CCM architecture that addresses both the functional requirements and the data dependencies for the myriad possible applications of CCM technology. As has been seen with conventional Behavioral Synthesis technology for ASICs, the answer is not all types of applications are so amenable [21]. Therefore, to understand how compilation technology might benefit a particular class of applications, we first need to examine and explore the characteristics of an application's problem space, manually mapping to candidate CCM architectures using available custom-logic design techniques, in order to better understand how to facilitate CCM systems synthesis for the class of problems being considered. Just such a class of problems exists in the Bioinformatics domain, specifically those associated with Phylogenetics[1].

This paper examines one such data point in the space of possible application solutions where high-performance

[1] We state that first considering a manual means of exploring CCM development for a class of applications can be a logical precursor for developing automated compilation techniques. However, the scope of the current research discussed here does not cover the definition of compilation techniques, nor does it cover an evaluation of the efficacy of such techniques to the Phylogenetics domain, as this is outside the scope of this paper.

computing using CCM hardware is required for operating on large, streaming data sets. Namely, we are looking at the Phylogenetics domain that provides us with a rich set of algorithms that can be studied to see if they can be implemented efficiently on custom computing machines to provide orders of magnitude speedup in the algorithm execution over that available on standard Von Neumann processor architectures using conventional programming techniques.

In this Bioinformatics domain, the Unweighted Pair-Group Method with Arithmetic means (UPGMA) algorithm used for Phylogenetic tree-reconstruction purposes has a certain computational complexity that makes it an application of specific interest. Furthermore, it is understood to have a software implementation that is particularly optimal, that is, it cannot be further optimized to achieve significant speedup in performance [12, 15].

The research work documented in this paper involves exploring the Phylogenetics domain as a candidate for CCM-based computational acceleration, even when considering use of a modest CCM platform. The objectives of this work were as follows:

- *Problem exemplar selection*: select the UPGMA algorithm [11], which performs phylogenetic analysis by building an evolutionary tree, as our exemplar problem domain.
- *Algorithm analysis*: identify and analyze the various computational tasks and bottlenecks associated with the algorithm, and asses its complexity.
- *Architecture analysis*: evaluate the issues that need to be addressed in implementing this algorithm as a custom computing architecture using custom-logic design techniques.
- *Custom-logic design*: explore various design mapping issues onto an FPGA substrate while refining the register-level architecture for the particular problem algorithm at hand.
- *HDL language-based implementation*: implement and verify the design for an FPGA-based CCM platform as VHDL models, using the platform's available PE, bus and memory resources as efficiently as possible.
- *CCM platform integration*: implement appropriate application and driver interface routines, for efficient integration between the application host's processor and the custom-logic application downloaded onto the CCM platform.
- *Comparative benchmark evaluation*: evaluate the CCM solution's performance by measuring its throughput, by running the system with increasing number of species taxa, and benchmark these results against those obtained from the software benchmark standard, namely, Felsenstein's PHYLIP program [12], executing on a conventional CPU-based system.

The Annapolis WILDCARD^{TM} system has been selected as the target CCM platform. The WILDCARD^{TM} FPGA board has a Xilinx Virtex® XCV 300E[2] as a processing element, along with two 256K byte memory units, and external I/O connections. This reconfigurable computing platform was chosen primarily based on cost and the availability of a reasonable set of platform development tools. In addition, it provides a data point for a computational "lower-bound" for exploring possible performance improvements for Phylogenetic applications on a custom computing platform.

2. Problem Background

We provide background on the application domain associated with the UPGMA algorithm and its context in the space of Bioinformatics computational problem solving.

2.1. Phylogenetics and Tree Reconstruction

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 [13].

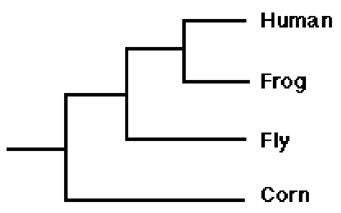


Figure 1: A phylogenetic tree showing a relationship between four species [13]

2.2. The Phylogenetic Tree

The tree represents the genealogical evolution of the different species, linking them through a certain set of similarities and differences. Similarities and differences

[2] Xilinx®and Virtex® are Registered Trademarks of Xilinx Inc.

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 phylogenetic 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.

2.3. Phylogenetic Algorithms and UPGMA

Phylogenetic algorithms cover three main classes of problems [14]: *(1) parsimony*, which is like a vertex coloring problem of graph theory; *(2) distance methods*, which aim to find a tree whose path distance matches closely to observed distances; and, *(3) likelihood methods*, where the likelihood of the data is calculated using Markov transition matrices. Each approach possesses certain problems in terms of the computational bottlenecks that occur.

Working with a biologist, we selected a particular phylogenetic *distance method* algorithm for this research, namely the UPGMA (Unweighted Pair-Group Method with Arithmetic means). 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 approximately of the order $O(N^3)$.

The value of N is typically around 10,000 to 50,000 in micro-array applications. Thus, even though software-based phylogenetics applications run this method at a rate of 1 second each [15], for $N = 100$, there is a run-time increase by a factor of perhaps 10,000 in micro array applications even before we consider the impact of memory bottlenecks. This last factor causes considerable problems, since memory usage also scales as $O(N^2)$. Thus, this problem might take days to complete with larger taxa data sets. This algorithm is well understood [15, 16], and the software solutions have reached a level of optimization beyond which only minimal performance improvement can be obtained. Thus, UPGMA is an appropriate candidate for exploring an implementation on a reconfigurable platform using custom-logic architecture and design techniques.

We define the distances between two clusters C_i and C_j to be the average distance between pairs of sequences from each cluster:

$$d_{ij} = (1/|C_i||C_j|) \Sigma d_{pq} \qquad (1)$$

$|C_i|$ 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

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

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 below 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.

$$\begin{bmatrix} x & 6 & 8 & 3 \\ 6 & x & 7 & 9 \\ 8 & 7 & x & 4 \\ 3 & 9 & 4 & x \end{bmatrix}$$

Figure 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 [16]:

- Initialization:
    - Assign each sequence i to its own cluster C_i,
    - Define one leave 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 $d_{ij}/2$.

2.4. Complexity and Bottlenecks on UPGMA

We believe the UPGMA algorithm has two bottlenecks. 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 device, giving better performance results.

3. The Custom Computing System

We implemented the functionality of the UPGMA algorithm as a custom-logic architecture, for deployment on an Annapolis WILDCARD^{TM} low-end CCM platform. 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.

3.1. Motivation for a Custom Computing Solution

The process of converting a specification into an implementation on FPGA devices can be addressed in different ways. Different design styles lead to different interpretations of the specification—a formal or informal description of the application's algorithm. An algorithm can be thought of as a set of processing steps for transforming data by executing a series of computations [5]. 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, or design, approaches.

Traditionally there have been two generic architectural styles: the stored-program machine (or, software) paradigm and the custom-logic (or, hardware) paradigm [5]. The stored program machine paradigm involves implementing an algorithm through use of an instruction code sequence that is interpreted by a microprocessor. In contrast, using the custom-logic paradigm, an algorithm is mapped onto an architecture employing storage and functional units that perform the computation without the use of an intermediate instruction set architecture (ISA) layer. Descriptions of algorithm behavior are mapped to an architecture that is, in turn, used to synthesize a circuit structure on a target substrate. For a CCM platform, the custom-logic paradigm is used to map the algorithm onto a hardware architecture consisting of an FPGA device as its processor.

Comparing performance between these two paradigms includes looking at the *latency* associated with carrying out computation, comparing between an application-specific software solution running on a conventional processor architecture (or even among a collection of processors, thus distributing the algorithm's execution across multiple, communicating processing elements) and, additionally, *throughput* of the architecture to run streaming computation, if appropriate for the application. However, evaluating performance could also include comparing the *design time* of the application [20]—comparing the time to architect, design, implement and test the application according to the requisite engineering processes of each paradigm.

Custom computing--and the predominant FPGA device architecture on which such applications are built at present--offers a good medium for implementing complex computational tasks having high throughput, low latency requirements. Many computational tasks spread over a range of application domains have been implemented and evaluated on reconfigurable computing systems [6, 7, 8, 9, 10]. However, different aspects of application architecture and performance must continue to be explored, while many new and novel computational problems must be implemented using custom computing machines before a general understanding of the characteristics of the custom computing paradigm can be obtained (such as to provide guidance in the development of compilers, as mentioned earlier). This provides a greater repertoire of custom computing solutions in addition to patterns for mapping between high-level problem-solving architecture and lower-level device architectures, which can be used to assess the cost/benefit ratio for effective and optimal implementation of more general programming problems on CCM platforms.

3.2. The Annapolis WILDCARD^{TM} System

The WILDCARD^{TM} system comes as a PC card and can plug into a PCI/PCMCIA card slot adapter, making it a very portable low-end reconfigurable platform. 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. 3 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 [18].

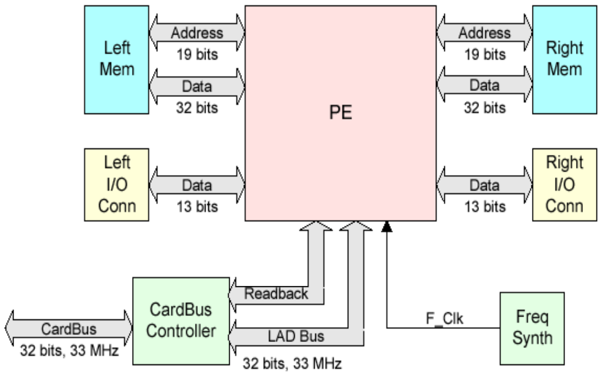


Figure 3. The WILDCARD^{TM} platform block diagram [18]

Data transfers to and from the PC host are done through control of a set of C program driver calls that interface with 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 Host Programming Application Programming Interface (API) provided by the vendor.

4. Custom Computing Design of UPGMA

The UPGMA algorithm undertaken for this project forms the problem statement and an HDL-based design methodology employed for its implementation [14, 20]. It was analyzed and a system specification generated. The design was partitioned into easily manageable blocks, and the RTL modeling of each of these modules was done in VHDL. The top-level hierarchical model is structural and merely connects the different sub modules to form the final top-level design. We have employed a certain amount of adaptation by reusing Xilinx cores to implement certain modules in the design [4]. This was done mainly to ensure that the design used no more resources than were available in the Virtex XCV300E chip of the WILDCARD^{TM} board.

The component simulation was conducted on each of the sub modules to verify their functionality. This step eased the final system-level simulation process, as errors within the sub modules implementing specific portions of the algorithm's data path and control were identified and corrected prior to that point. The final top-level structural model was then written, and system simulation conducted to verify the functionality of the design and the algorithm as a whole. The ModelSim® simulation environment was used for debugging and testing purposes.

Logic synthesis was conducted to generate a circuit netlist, and was also used as a planning tool for identifying the critical paths and determining resource usage of the design. The Synplify Pro® 7.3 FPGA synthesis toolset was used for this purpose. The synthesized gate level netlist does not entirely form the final design being implemented on the Processing Element (PE) of the WILDCARD^{TM} system. The functionally verified design must be embedded within the vendor-supplied VHDL architecture model for the PE.

This PE model was then placed within the WILDCARD^{TM} system simulation model and the final functional testing was conducted. The verified PE model was then synthesized and the EDIF netlist generated. The EDIF netlist was then placed and routed using a makefile provided for the WILDCARD^{TM} system. The makefile contains calls to the Xilinx place and route tools for generating the final PE image used to configure the device. This image is then used as the application target to proceed with the WILDCARD^{TM} host programming process.

4.1 UPGMA Data Path

We now consider the design of the data path and control portions of the UPGMA CCM 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.

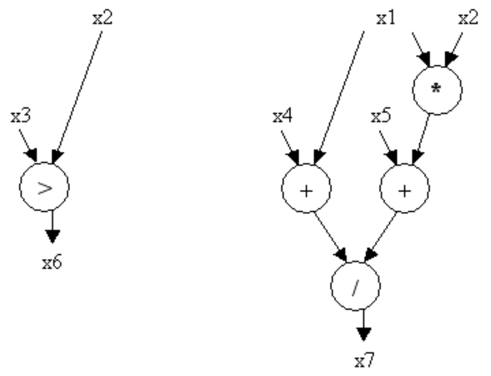


Figure 4. UPGMA algorithm data flow graphs.

The datapath on the left of Fig. 4, 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) in Section 2.3. 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.

Numerator = $(d_{ik}h_i + d_{jk})$
Denominator = $h_i + h_j$
Average distance = $(d_{ik}h_i + d_{jk})/(h_i + h_j)$

4.2. UPGMA Control Flow and Memory Management

The design utilizes a controller module implementing the coordinated sequencing of the datapath operations, as defined by the algorithm's control flow graph in Fig. 5. 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. 5, repeating the process until all the nodes in the tree have been processed.

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. Also the small size of the memory banks on the WILDCARD™ board not only limits 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 ultimately added to the cycle latency of the executing design. (The specific details of this addressing scheme workaround are outside the scope of this paper.)

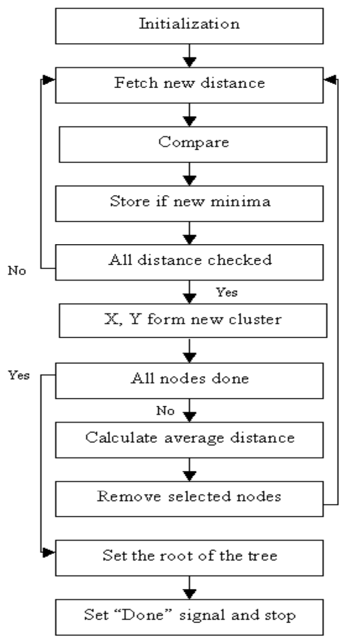


Figure 5. The UPGMA design control flow.

5. Experimental Setup

5.1. Apparatus for Evaluating UPGMA

The WILDCARD™ host-programming environment provides us with an API to program the WILDCARD™ system. The WILDCARD™ CCM platform interface routines are used in the 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.

The UPGMA host program was written based on the example templates provided by the vendor for setting up a custom computing application for reading and writing the SRAM memory banks, reading and writing data to the Virtex® Processing Element (PE) register space, and for processing PE interrupts. Using these examples as guides, we created a complete host-based, experiment "driver" application, employing the above three components, to perform the following host-to-computing server protocol steps:

- Initialize the WILDCARD™ system;
- Program the PE from the image file;
- Set the Clock frequency;
- Enable PE interrupt line;
- De-assert PE Reset line;
- Read distance data from the file into a distance array;

- Transfer distance data from the distance array to WILDCARD^{TM} Left memory;
- Write the value of the number of taxon being operated upon into a PE register;
- This triggers the design to start running and assert "done signal" after it finishes;
- The C program waits until the done is set;
- Reads data from the Right memory; and,
- After reading, it outputs the data into a destination file in the PC host file system.

5.2. UPGMA Test Data Sets

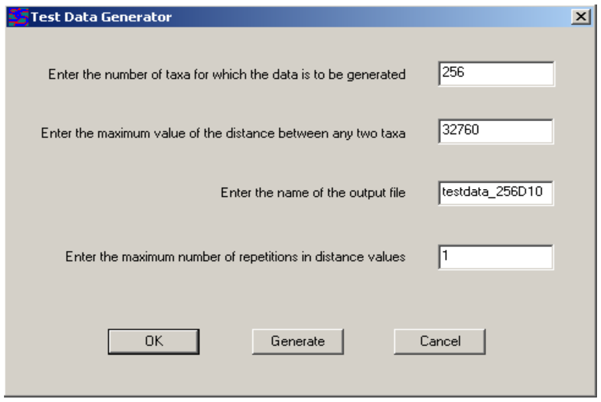
A program written in C++ using the MFC programming environment was created to generate the test data for testing the implementation of the UPGMA algorithm. This program takes as input the following parameters: (1) the number of taxa; (2) the maximum value of inter-node distance; and, (3) the number of 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 are randomly generated for that number of taxa. Furthermore, each created data set has its data values subjected to permutations, creating up to 10 permutations per data set per number of taxa. Part of the experimental analysis is to insure that the resultant latency measurements are not wildly varying but, rather, converge around a mean value.

Fig. 6 presents a screenshot of the dialog box used by the program for allowing the user to specify the parameters for generating test data. The taxa size, maximum nodal distance, and the number of repetitions of a particular distance value are given as inputs. When the data has been generated, the program pops up a confirmation dialog box.

The data values are generated randomly, making sure that each value is within the maximum nodal distance limit set by the user. Also, the specified number of repetitions of each value in the data set is constrained to be less than or equal to that specified for repetitions by the user. For each taxa size, ten different datasets were generated in our experiments. Furthermore, for each of these ten datasets, ten different permutations were generated, by changing the positions of the distance values within the distance matrix.

The time taken for the UPGMA implementation to execute on the WILDCARD^{TM} system is measured using standard C time function calls. Time measurements are collected for the time taken for the program to transfer the distance data to the memory, generate the tree, and read back the output tree data from the memory. The time is measured in terms of CPU clock ticks using the standard C language *clock( )* function call.


Test Data Generator

Enter the number of taxa for which the data is to be generated: 256

Enter the maximum value of the distance between any two taxa: 32760

Enter the name of the output file: testdata_256D10

Enter the maximum number of repetitions in distance values: 1

Figure 6. Test data generator input dialog box.

6. Comparative Experimental Results

We present the results of running phylogenetics data sets against the UPGMA implementation on the WILDCARD^{TM}-based custom computing machine. Tafter actually running the experiments, the resultant data sets are tabulated and plotted in terms of a bounded clock cycle count using the clocking frequency of the host PC's CPU clock, which gives 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^{TM} board. This communication overhead is an important consideration when comparing the efficacy of a CCM solution over other possible ones, such as the baseline PHYLIP program executing solely on the PC host.

We take randomly generated data sets, permute them, and execute them on the WILDCARD^{TM}. We execute the UPGMA computation for these different data sets while also increasing the number of taxa considered in the input distance matrix. The test data for taxa sizes of 10, 16, 32, 50, 64, 75, 100, 128, 150, 175, 200, 225 and 256 were executed. For each taxa size, ten different data sets along with 10 different permutations of certain datasets were run and timing results collected.

One aspect of defining the data set for purposes of running experiments is permuting the data to assess whether permutation impacts the execution latency. In some implementations of UPGMA in software, permutation might affect the execution of a given data set at some number of taxa. The permutations were randomly generated along with the data sets. However, our expectation was that permuting the data would not be much of a factor in variation of latency values, because the time to perform actual computations on fixed-width operators is largely independent of the actual data values passed as the operators. From the data collected during the experimental permutation runs, we concluded that this was indeed the case. Tallying the data and plotting it in MS-Excel® verified this behavior in the data.

Taxa	Hardware	Software	Improvement
10	8.4	121	14.4
16	8.5	170.1	20
32	9.4	315.3	33.5
57	12	541.7	45.1
64	14	713	50.9
75	20.3	816	40.2
100	39.6	942.1	23.8
128	71.6	1107.5	15.5
150	110.2	1278	11.6
175	162.5	1479	9.1
200	242.6	1788.8	7.4
225	342.1	2250.7	6.6
256	504.4	2659.9	5.3

Table 1. Comparison Between UPGMA Implementations.

We now consider the performance comparison of the UPGMA algorithm, as implemented in CCM hardware versus the software implementation. In order to be able to make this comparison, the same data sets were executed on the same PC running the PHYLIP program. The data was collected, collated and plotted in MS-Excel®, as before. The Average number of clock ticks taken for each of the taxa data runs, for both implementations, is given in Table 1. Fig. 7 displays comparative plots of the averaged runs for both implementations, while Fig. 8 presents the same data plotted on a logarithmic scale. From both plots, it can be seen that the custom computing implementation of the UPGMA algorithm provides significant speedup, at some taxa counts delivering approximately two-orders of magnitude in speedup over the software implementation.

Fig. 9 displays the difference in the rate of change of speedup of the CCM implementation over the PHYLIP execution. The results show a significant improvement for taxa up to 64, but then the rate of improvement starts to decline as the taxa size increases to the 256 maximum set for the experiments[3].

This behavior in the CCM implementation was determined to be caused by the fact that a design change made to implement a taxa count of 75 and above was adversely affected by the memory addressing scheme made in the final architecture modifications, as mentioned in the previous section. The negative impact in performance of the design was also attributed to the four-cycle latency associated with a WILDCARD™ on-board SRAM memory read. This latency induces wait states in the control structure, causing the design to run slower. The address modification would have not been necessary had the WILDCARD™ system had larger memory banks and the original addressing scheme of concatenating nodal values were still used.

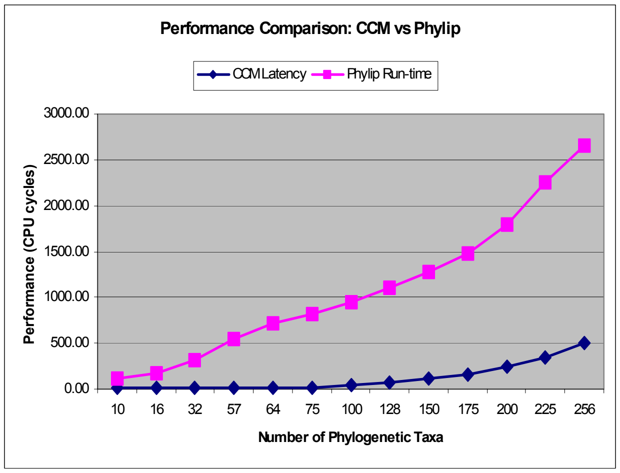


Figure 7. Performance difference as a function of taxa count.

However, if we look at the plot of the performance data itself, and compare the two curves, in Fig. 7., we see that the performance improvement does indeed seem to scale, as the performance curve for the PHYLIP implementation of UPGMA grows at a faster rate than that of the implementation of UPGMA as a custom computing machine architecture.

If we observe the trend as a logarithmic plot, Fig. 8., we see that, for the peak performance point for the custom computing implementation on the WILDCARD™ (between 64 and 75 taxa), we are operating close to two-orders of magnitude faster than the software PHYLIP implementation.

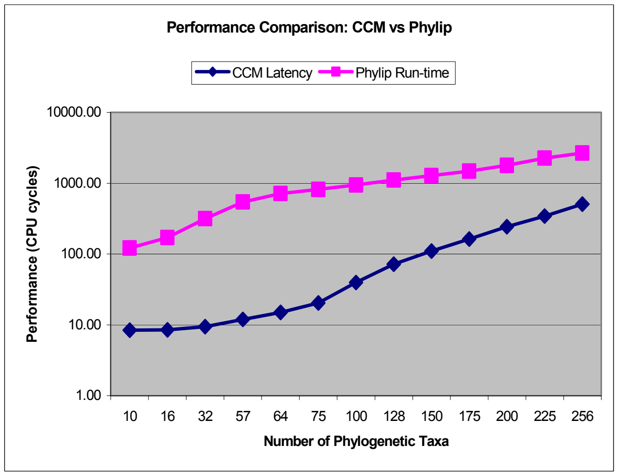


Figure 8. Performance difference as a function of taxa count (log scale).

[3] A 256 taxa maximum was set as a target for the experiments, based on initial estimates using the sizes of the various data structures and their memory usage. Considerable architecture and design iteration work was carried out in order to achieve the estimated taxa maximum. This number was defined solely based on analysis of the WILDCARD™ platform.

This achievement of approximately two-orders of magnitude speedup in computation—even if only observed for a "sweet spot" in the size of the data sets being considered—is consistent with other comparative findings in the literature [17].

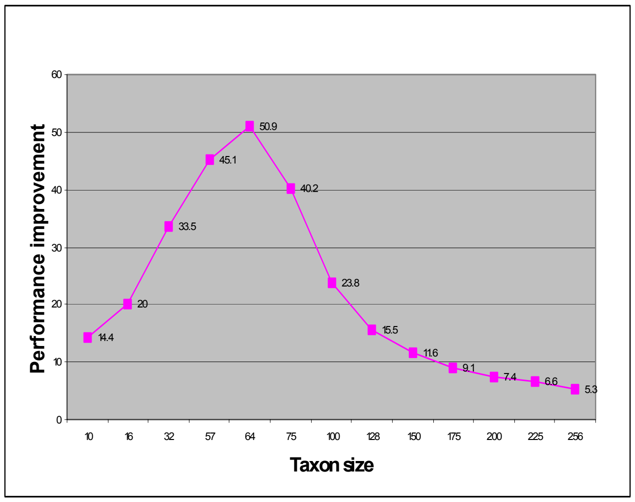


Figure 9. Performance improvement rate of change over PHYLIP as a function of taxa count.

If we consider what is happening in terms of algorithm time complexity, we would expect some change in the coefficients of recurrence equations for the plot functions, as shown in Fig. 7, not a wholesale change in the upper bound of the $O(N^3)$ complexity itself [22, 23]. When we generate the recurrence equations through linear regression using MS-Excel®, this is indeed what we observe. (Note that the actual recurrence equations and their coefficients are not presented in this paper, but can be found in [24] along with other details of the work reported in this paper.)

7. Summary and Future Work

In this paper, we have presented the analysis, design, implementation and subsequent experimentation that has been conducted in applying the custom computing paradigm to the domain of Phylogenetics—namely through the design and implementation of a CCM for running UPGMA computations on species taxa data sets. 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.

The CCM solution results showed significant performance improvement over the benchmark UPGMA software implementation, peaking at the 64 taxa dataset size. For datasets of 74 taxa and above, the performance began to degrade considerably, compared to that of the software implementation—although the custom computing implementation was still between half- to a full-order of magnitude faster.

The performance degradation for datasets of 75 taxa and above is attributed to bottlenecks in the memory addressing scheme and several modifications to the design to allow implementation of a maximum taxa data set size of 256. These workarounds added considerable latency to the design and negatively affected the results for datasets of 75 taxa and above. However, given the limitations of the chosen platform, these workarounds were necessary for use to achieve our objective of supporting computation on a data set of 256 taxa.

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 CCM platform, considerable benefit in computation speed could be obtained. We expect the benefits to far exceed those observed here when the application architecture is moved to a larger CCM platform.

The next phase of experimentation with the UPGMA architecture will be to, first, migrate it to the SRC Computers SRC–6E [19], which has a larger memory address space in addition to the larger Virtex-II® FPGA devices. Second, once we move this basic architecture to the SRC platform, we will optimize it for that CCM platform. We expect to considerably extend the number of taxa that can be processed using the UPGMA custom-computing machine.

8. Acknowledgements

We acknowledge the contributions of Dr. Peter Waddell, Associate Professor, jointly appointed in the Departments of Biology and Statistics, University of South Carolina, for his assistance with the algorithm definition and with provision of the baseline phylogenetic data set. We also acknowledge the assistance of Dr. Duncan Buell and Dr. John Rose for their project support, and for their review and comments on the results of this research.

9. References

[1] 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.

[2] Nick Tredennick, *The case of reconfigurable computing*. Micro Design Resources, Microprocessor Report, Vol.10, No.10, Aug 1996.

[3] Stephen Brown and Jonathan Rose, "Architecture of FPGAs and CPLDs: A Tutorial", *IEEE Design and Test of Computers*, Vol. 13, No. 2, pp. 42-57, 1996.

[4] Xilinx Inc, *Virtex-E 1.8V FPGA Complete Datasheet*, March 2003

[5] 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.

[6] Paul Graham, Brent Nelson, "FPGA Based Sonar Processing", *ACM/SIGDA International Symposium for Field Programmable Gate Arrays*. Pg 201-208. February 1998. http://www.dynamicsilicon.com/Articles/Reconfigurable.pdf

[7] 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*, pg 216-225, April 1996. IEEE Computer Society.

[8] Jason R. Hess, David C. Lee, Scott J. Harper, Mark T. Jones, and Peter M. Athanas, "Implementation and Evaluation of a Prototype Reconfigurable Router", *Proceedings of IEEE symposium of Field Programmable Custom Computing Machines*, pg 44-50, April 1999. IEEE Computer Society.

[9] R. Petersen, B. L. Hutchings, "An Assessment of the Suitability of FPGA-Based Systems for Use in Digital Signal Processing", In *5th International Workshop on Field Programmable Logic and Applications*, pp 293-302, August 1995, Oxford, England.

[10] P.W. Dowd, J.T. McHenry, F.A. Pellegrino, T.M. Carrozzi and W.B. Cocks, "An FPGA-Based Coprocessor for ATM Firewalls", *Proceedings IEEE Symposium on FPGA's for Custom Computing Machines (FCCM-97)*, pg 30-39, April 1997.

[11] R. Shamir, *UPGMA*, Tel Aviv University, http://www.math.tau.ac.il/~rshamir/algmb/00/scribe00/html/lec08/node21.html

[12] Joe Felsenstein, *PHYLIP source code*, Department of Genome Sciences, University of Washington, http://evolution.genetics.washington.edu/phylip.html

[13] Peter H. Weston, Michael D. Crisp, "Introduction to Phylogenetic Systematics", *Invited Contributions of the Society of Australian Systematic Biologists*, SASB, http://www.science.uts.edu.au/sasb/WestonCrisp.html.

[14] James P. Davis, Peter J. Waddell, Sreesa Akella, "Methods and Architectures for Realizing Fast Phylogenetic Computation Engines Using VLSI Array Based Logic", Submitted to *IEEE Bioinformatics Conference*.

[15] 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.

[16] R. Durbin, S. Eddy, A. Krogh, G. Mitchison, Building phylogenetic trees, Chapter 7, *Biological Sequence Analysis*, pg 160-190. Cambridge University Press, 1998.

[17] Duncan A. Buell, Jeffrey M. Arnold, Walter J. Kleinfelder, *SPLASH-2: FPGAs in a Custom Computing Machine*, IEEE Computer Society Press, 1996.

[18] Annapolis Microsystems Inc, *Annapolis WILDCARD^{TM} System Reference Manual*, Revision 2.6, 2003. http://www.annapmicro.com

[19] SRC Computers Inc., http://www.srccomputers.com.

[20] Yutana Jawchinda, Hideaki Kobayashi, "Quantifying Design Reuse: An HDL-Based Design Experiment", *Proceedings International Verilog HDL Conference*, April, 1999.

[21] Wolf, W., *Modern VLSI Design: System on Chip Design*, 3rd Ed., Prentice-Hall Publishers, Inc., 2003.

[22] Wood, D., *Data Structures, Algorithms, and Performance*, Addison-Wesley Publishing Company, 1993.

[23] Finnegan, J., "The VLSI Approach to Computational Complexity", in Kung, H. T., B. Sproull, and G. Steele, *VLSI Systems and Computations*, Computer Science Press, Inc., 1981, pp. 124-125.

[24] Akella, S., *From Custom Logic to Custom Computing: Exploring the Architecture and Implementation of Bioinformatics Algorithms in Custom Computing*, Unpublished Master's Thesis, University of South Carolina, 2003.