February 2014 UILU-ENG-14-2201

DECOMPOSING GENOMICS ALGORITHMS: CORE COMPUTATIONS FOR ACCELERATING GENOMICS ANALYSES

Arjun P. Athreya, Subho S. Banerjee, C. Victor Jongeneel, Zbigniew T. Kalbarczyk, and Ravishankar K. Iyer

Coordinated Science Laboratory
1308 West Main Street, Urbana, IL 61801
University of Illinois at Urbana-Champaign

REPORT DOCUMENTATION PAGE

Form Approved
OMB NO. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	February 2014	

4. TITLE AND SUBTITLE
Decomposing Genomics Algorithms: Core Computations for Accelerating Genomics Analyses

5. FUNDING NUMBERS
NSF CNS 13-37732 (National Science Foundation)

6. AUTHOR(S)
Arjun P. Athreya, Subho S. Banerjee, C. Victor Jongeneel, Zbigniew T. Kalbarczyk, and Ravishankar K. Iyer

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Coordinated Science Laboratory, 1308 W. Main St., Urbana, IL 61801

8. PERFORMING ORGANIZATION REPORT NUMBER
UILU-ENG-14-2201

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
National Science Foundation, 4201 Wilson Blvd., Arlington, VA 22230; Infosys, Hosur Road, Electronics City, Bangalore, 560100, India; IBM Faculty Award, 1101 Kitchawan Road, Yorktown Heights, NY 10598; Office of the Vice Chancellor for Research, University of Illinois at Urbana-Champaign, 4601 E. John, Champaign, IL 61820.

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*
Technological advances in genomic analyses and computing sciences has led to a burst in genomics data. With those advances, there has also been parallel growth in dedicated accelerators for specific genomic analyses. However, biologists are in need of a reconfigurable machine that can allow them to perform multiple analyses without needing to go for dedicated compute platforms for each analysis. This work addresses the first steps in the design of such a reconfigurable machine. We hypothesize that this machine design can consist of some accelerators of computations common across various genomic analyses. This work studies a subset of genomic analyses and identifies such core computations. We further investigate the possibility of further accelerating through a deeper analysis of the computation primitives.

14. SUBJECT TERMS
Bioinformatics; Genomic analyses; Algorithms

15. NUMBER OF PAGES
8

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL

Decomposing Genomics Algorithms: Core Computations for Accelerating Genomics Analyses

Arjun P. Athreya*, Subho S. Banerjee†, C. Victor Jongeneel‡, Zbigniew T. Kalbarczyk* and Ravishankar K. Iyer*†
*Dept. of Electrical and Computer Engineering
†Dept. of Computer Science
‡The Institute for Genomic Biology University of Illinois at Urbana-Champaign, USA
{athreya2, ssbaner2, vjongene, kalbarcz, rkiyer}@illinois.edu

***Abstract*—Technological advances in genomic analyses and computing sciences has led to a burst in genomics data. With these advances, there has also been parallel growth in dedicated accelerators for specific genomic analyses. However, biologists are need of a re-configurable machine that can allow them to perform multiple analyses without needing to go for dedicated compute platforms for each analysis. This work addresses the first steps in the design of such a re-configurable machine. We hypothesize that this machine design can consist of some accelerators of computations common across various genomic analyses. This work studies a subset of genomic analyses and identifies such core computations. We further investigate the possibility of further accelerating through a deeper analysis of the computation primitives.**

I. Introduction

An unprecedented growth in genomics has created a data deluge available for various analyses. The nature of computations on these large data-sets are very complex. Several tools exist that have accelerated or have been optimized for performance of various genomic analyses. These optimizations have been for specific algorithms and have been implemented on compute platforms such as FPGA or GPU [1] [2] [3] [4]. However, the future of computations on the genomic data calls for a re-configurable machine that can be optimized for not one but many algorithms of genomic analyses.

In this work, we hypothesize that some core computations are common across many algorithms of genomic analyses. We believe that this approach is valid because mathematical models are used to best describe the various phenomenon in biology. Therefore, if there are several such analyses that study similar or causal phenomenon in biology, the underlying mathematical computations could also be similar across many those analyses. If this belief is true, as computer scientists we can design and build machines which support dedicated hardware for such computations and allow for analysis specific software libraries ("glue-code") that pre-process data inputs to these dedicated hardware for computations. An intelligent programming interface or a scheduler that knows the computations in the algorithm for the specific analysis schedules those computation accelerators and the software libraries to complete the analysis.

The advantages of such a re-configurable machine is three-fold as illustrated in Figure 1. First, multiple analyses can be run on one machine for the same data-sets which could use intermediate computation's output as inputs for other analyses (Multiple sequence alignment algorithm's outputs can directly feed to a motif-finding algorithm). Second, if the computation hardware already exists and new algorithms are proposed in the future, only software libraries need to be developed. Else, new dedicated computation blocks can be added when the need arises without necessarily changing the rest of the machine's design. Finally, in cloud hosted services, the virtualization can be put to best use to create instances of these software libraries that can reuse the dedicated compute hardware.

In this work we call these common core computations of various analyses as *memory-less computations*. What we mean by memory-less in this work is, the dedicated hardware for these computations given an input compute the output in the same manner irrespective of what genomic analysis is using them. A simple example to illustrate this is, if two analyses use matrix multiplication in their pipeline and let there exists a dedicated hardware for the same. Once the dedicated matrix multiplier hardware is fed with its input data, it computes the resulting matrix the same way irrespective of which of the analysis is making it multiply two matrices. To this end, the first half of this work studies a subset of popular genomic analyses and identifies such core computations. We then look at a particular example of such a decomposition. And how it might benefit from the plethora of research dealing with acceleration of these core-computations on several hardware platforms (like FPGAs and GPUs) and software programming frameworks (MPI, Charms++ etc.).

Remainder of this paper is organized as follows. In Section II, we identify the core computations of some well used genomic analyses. Then in Section III, we look at how the multi-sequence alignment problem can be accelerated under this scheme of breaking up larger genomic work-flow components into their constituent computations and accelerating them. In Section IV, we explore possible optimizations that can be applied to the core computations that might be difficult to apply to the original application. Finally, we gather our conclusions and discuss futures avenues expanding on this work in Section V.

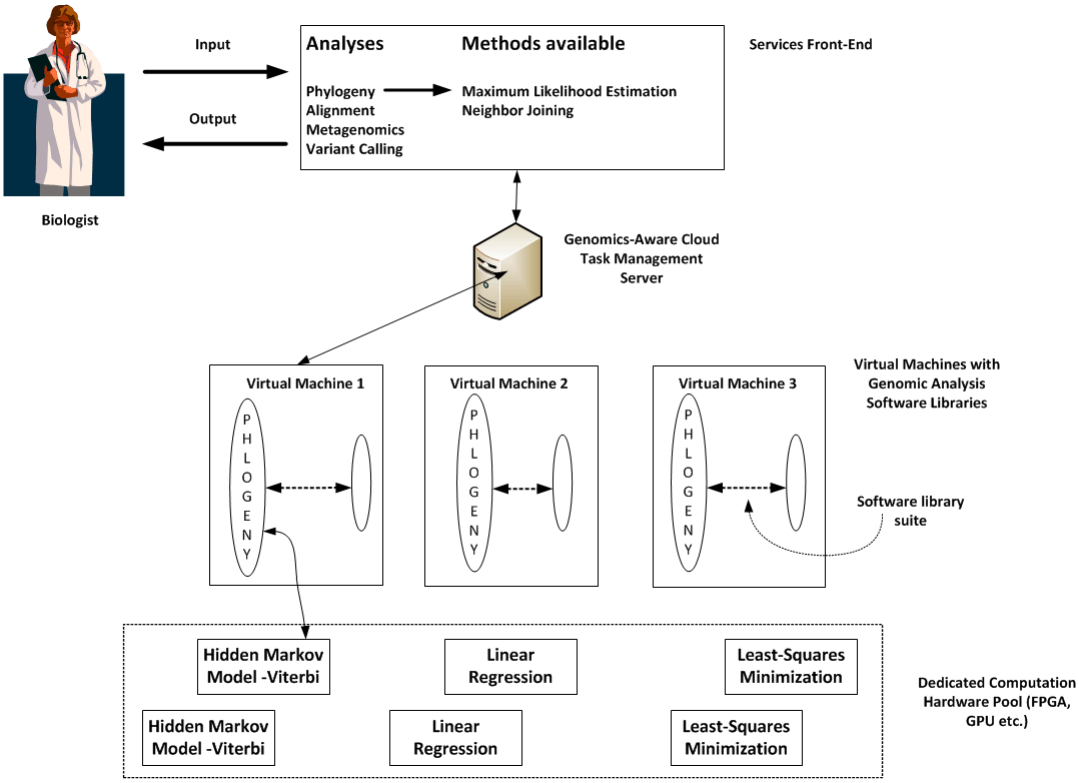

Fig. 1: This figure illustrates a cloud-hosted genomics service with a re-configurable machine design. Biologists provide the inputs, choose the analysis and the methods. The task server creates an instance of a virtual machine (VM) providing the biologist's inputs. The VM accesses the dedicated hardware for core computations in the hardware pool. The output is then fed to the biologists and the hardware is released for other needing analysis.

II. Core Computations of Genomic Analyses

In this section we discuss some well-studied genomic analyses and identify their core computations. We present three classes of genomic analysis, their algorithms and computations in those algorithms. The three classes we present are *phylogeny*, *sequence alignment (pairwise and multiple)* and *single-nucleotide polymorphism (SNP) calling*.

A. Observation

We discuss numerous algorithms and methods for various genomic analyses later in this section. However, we present a summary of our observation of the various genomic analyses tabulated in Figure 2. Our observation quantifies the common computations across the analyses studied in this work. In SectionsII-B - II-D, we analyzed 24 tools and identified their core computations.

We group arithmetic mean and least-squares computations as linear algebraic computations. With found that 46% (11 of 24) of the algorithms or tools comprised arithmetic means or least-squares as their core computation while the remaining 54% (13 of 24) of had solving HMM (Hidden Markov Model) as their core computation. About 9% of the tools comprised both core computations, these were flexible software libraries that allow users to specify the method of performing a particular genomic analysis. We further note that, in particular Viterbi algorithm was used in 80% of these tools to estimate the sequence of hidden states, while the remaining used Baum-Welch algorithm. Our study of these analyses and algorithms though don't encompass the entire space of genomics, it atleast gives us the needed insight of whether there exists such common core computations across many analyses.

We thus show that *Some core computations are common across many genomic analyses*. We next discuss how we arrived these core computations for the genomic analyses. We will later discuss discuss the significance of the existence of these common core computations for re-configurable machines for genomic analysis workflows.

B. Phylogeny

Phylogeny or the area of Phylogenetics is the study of evolutionary relationship among a group of organisms given their DNA or amino acid sequences. A phylogenetic tree is used to describe the evolutionary relationship among the organisms, which is hierarchical. The tree can be rooted or unrooted depending on the existence of a common ancestor. The process of using such sequences to arrive at the phylogenetic relationship is a two-step process. First, the tree's structure is established and then branch length of the trees is estimated. The biological significance of these two steps is that, the tree's structure explains the ancestry of the organisms being studied

Analysis	Method	Algorithm	Core Computation	Tools
Phylogeny	Distance Method	UPGMA, Neighbor Join	Arithmetic mean, Minimization of Least-Squares	PHYLIP, MEGA, TNT, ClustalW, PAUP, T-Rex, DARWIN
	Probabilistic Method	Maximum Likelihood Estimation	HMM with Viterbi/Baum-Welch	PHYLIP, PhyML, mrBAYES, RAxML, MEGA5, Clustal-Omega, CLustalW
Sequence Alignment	Pairwise Sequence Alignment - Dynamic Programming	Smith-Waterman, Needleman-Wunsch	Arithmetic mean and graph traversal	AlignME, Bioconductor, Bioperl, EMBOSS
	Multiple Sequence Alignment - Probabilistic Method	Score graph constructions	HMM with Viterbi/Baum-Welch	Clustal-Omega, UGENE, SAM, Phylo, Porbalign, T-COFFEE
Single Nucleotide Polymorphism Calling	Probabilistic Method	Variant Calling	Profile HMM with Viterbi/Baum-Welch	GATK, SAMTOOLS

Fig. 2: This table summarizes our study of three major genomic analyses, their algorithms and our finding of their core computations.

and the branch length is an indication of the amount of genetic changes during the evolution.

There exists three classes of methods that can be used to construct the phylogenetic trees. These classes are 1) distance methods, 2) parsimony methods and 3) probabilistic methods. We discuss the distance methods and the probabilistic methods, popular algorithms in each of them and identify their core computations.

Distance Methods

Distance methods use genetic distance between sequences input studying their evolutionary relationships. Distances are a function of rate of mismatches in aligned regions of the input sequences. Several methods have been proposed to calculate these distances, with the Jukes-Cantor distance calculation being among the most used [5].

Generally, the distance methods use some variants of *hierarchical data clustering* algorithms. The goal of these algorithms is to create clusters for inputs which have the least of the *genetic distances* between them and then recursively reduce the inter-cluster distances to arrive at the phylogenetic tree. We can broadly define genetic distance as the occurrences of nucleotide changes in regions of aligned sequences. Popular algorithms that perform these actions are the *Unweighted Pair Group Method with Arithmetic Mean* and the *Neighbor-joining* [6], [7]. The assumptions in these two algorithms differ with respect to the fact that the evolution happens at a constant rate. There has been several works on improving the computation speeds of neighbor-joining algorithm by speeding up the distance calculation and the search for nucleotides to form the cluster [8]. However, the core computation in both these algorithms is the *arithmetic mean*. Tools or software packages that use these methods with the underlying core computation of arithmetic mean are PHYLIP, MEGA, TNT, ClustalW, PAUP, T-REX [9] [10] [11] [12], [13], [14].

Another way of obtaining the phylogenetic tree is by minimizing the least-squares of genetic distances [15]. Naturally here, the core computation is the minimization of least-squares. It has been shown that finding a phylogenetic tree with optimal branch lengths using this method is a NP-complete problem. However, assumptions in constructing the distance matrices and minimization for least-squares allow for a polynomial run-time [16]. Tools or software packages that use minimization of least-squares methods as the underlying core computation to construct the phylogenetic trees are PHYLIP, PAUP and DARWIN [17].

Probabilistic Methods

Maximum likelihood (MLE) method is among the most popularly used probabilistic methods to obtain the phylogenetic tree. A substitution matrix is used to account for mutations (nucleotide change at a position in one sequence to another nucleotide in another sequence). The way MLE is used to derive the phylogenetic tree is as follows. Each site of a nucleotide (A, C, T or G) in a sequence has a likelihood and is defined by a model (examples include Markov Model, Hidden Markov Model, Poisson Model). Total likelihood of the sequence is a product of individual likelihood of the sites in the sequence, assuming each site undergoes mutations independently. Now, MLE of the tree is the topology that yields the highest likelihood for the model used to describe the mutations [18].

Among the many probabilistic models, Hidden Markov Model (HMM) is the most popularly used to describe the probability of a sequence arising from another sequence. The observed states of the HMM are the nucleotides of the known sequence and the hidden states are the sequence of states in the original sequence that emit the observed sequence. Given the independence nature of mutations, Viterbi algorithm is most popularly used to estimate the HMM's parameters and thereby solve the MLE. Other popular algorithm used to estimate the parameters of the HMM is the Baum-Welch algorithm. Variants of the HMM are also proposed to obtain phylogenetic trees in a more computationally efficient man-

ner while still improving the accuracy [19]. Some software packages that output the phylogenetic tree given the sequences are PHYLIP, PhyML, mrBAYES, RAxML, MEGA5, ClustalOmega, ClustalW [9] [20] [21] [22] [23] [11]. Recent work has also looked at using Markov-Chain Monte Carlo (MCMC) to infer the phylogenetic tree and the mentioned tools are support MCMC as well [24].

C. Sequence Alignment

Sequence alignment is a process of establishing the conserved regions between sequences. If it is between two sequences, then it is called pairwise sequence alignment, or it is called as multiple sequence alignment (MSA). We will discuss the algorithms for sequence alignment and identify their core computations.

Pairwise Sequence Alignment

Alignment can be local or global. In global alignment, an attempt is made to align every residue of the two sequences. Local alignment aligns one sequence with smaller regions of the larger sequence which have high similarity. Both these types of alignments are largely solved dynamic programming problems [25] [26]. The algorithms for global and local alignment are popularly known as Needleman-Wunsch and Smith-Waterman algorithms respectively. Both these methods strive towards establish an optimal alignment that is the maximum weight trace of the path from the originating nucleotide to the last nucleotide visited in the sequence. The algorithm recursively solves the problem by increasing the size of the comparison by one nucleotide each time. On reaching the end of the sequences, back-tracking is used to trace the optimal alignment. Both these popular algorithms are deterministic methods of pairwise alignment. Core computations in these algorithms are matrix traversal with comparisons and additions. Some of the many tools which use these computations are AlignMe, Bioconductor, Bioperl and EMBOSS [27] [28] [29] [30]. There exists heuristics based approaches using pattern matching which allow for searching databases which output possible sequences that show alignment with an input sequence such as BLAST and FASTA [31] [32]. We will not discuss these in this work.

Multiple Sequence Alignment

We particularly discuss progressive methods for multiple sequence alignment in this work. As the name indicates, progressive methods align multiple sequences in stages. In the first stage, pairwise distances between sequences are computed. Dynamic programming techniques are used to compute the divergence between sequences. In the second stage, guide tree similar to a phylogenetic tree for the sequences is constructed using the distances constructed in the first stage. The hierarchical clustering techniques discussed in Section II-B are used in the second stage. In the third stage, the sequences are aligned with reference to the phylogenetic tree starting with the sequences showing least of similarity and then eventually the most similar sequences get matched. CLUSTALW is the most popular tool that uses this approach to align multiple sequences. One core computation of the second stage of progressive method is the use of HMM with Viterbi algorithm for arriving at the guide tree. Some other tools which use this computation to align multiple sequences are ClustalOmega, UGENE, SAM, Phylo, Probalign and T-COFFEE [33] [34] [35] [36] [11].

D. Single Nucleotide Polymorphism Calling

Single Nucleotide Polymorphism (SNP) is a variation in the DNA sequence when a single nucleotide in the genome differs between organisms. These single nucleotide mutations are also referred to as variants, and the process of detecting these is known as SNP calling or variant calling. SNP calling is used heavily to for various analysis such as how prone is an individual to a particular disease.

In this work, we do not describe the workflow or pipeline of variant calling, which includes alignment of sequences and other functions as error corrections. We focus only on the algorithms or implementations of variant calling that helps detect these mutations. GATK and SAMTOOLS are popular tools which detect these variants and the basic idea of the detection is as follows [37] [38]. A likelihood model is used to estimate the best model for both the type of genome and the variants (allele frequencies to be specific). They output the posterior probability of the variant at each site. The model used for the likelihood in both these tools is the Profile HMM. Profile HMM provide the added benefit of using a well-defined probabilistic theory and insertion scores instead of using heuristic methods. Viterbi algorithm is used for training and estimation of the profile HMM's parameters in these tools.

E. Memory-less Computations

The outcome of these observations leads us to think of the following. If there exists these core computations common across many genomic analyses, then we could provide dedicated (optimized for performance and speed) compute engines in hardware for the same. This means that we will need the software "*glue-code*" to prepare and feed the input data to these dedicated compute engines. These compute engines will process the input data, perform computations on them and provide an output. Thus it brings us to a point where these compute engines and their computations are *completely oblivious* to the genomic analysis calling these computations. We call these application oblivious computations as *memory-less computations*.

The impact of identifying memory-less computations is that it allows biologists to only write and optimize the glue-code for genomic analysis of their interest and let the dedicated compute engines perform action on the input data. Therefore, as and when new genomic analysis techniques are identified, only the glue-code has to be written and/or new compute blocks need to be installed and the machine is ready. This means that the same machine with a combination compute blocks can be reconfigured for different applications with an efficient scheduler and programming interface, on one machine

as shown in Figure 1. Thus this could reduce cost of running genomic applications and could lead to realizing more such applications which could be hosted on the cloud.

III. Decomposing the Clustal-W algorithm

In this Section, we demonstrate how an algorithm can be decomposed into the *memory-less* computations described in Section II and consequently be constructed as a streaming computation. We use the Clustal-W algorithm for Multi-Sequence alignment as an example.

A. The Clustal-W Algorithm

Since its publication in the late 1980s[39][40], the Clustal family of multiple sequence aligners have seen widespread adoption and are used routinely as parts of bioinformatics workflows. The current set of Clustal programs all derive from Clustal-W [41] (the 'W' in the name of the algorithm stands for "weights") , whose main contribution is a novel scoring scheme to determine the goodness of the aligned sequence and a weighting scheme for down weighting frequently appearing sequence groups.

The Clustal family of algorithms are progressive aligners which look at solving the global alignment problem using a neighbor joining approach to reduce the multi-sequence alignment problem to that of pairwise alignment. Like other progressive algorithms for MSA, Clustal-W is guaranteed to find highly conserved blocks first and outliers are added last – together, this increases the chances that conserved blocks survive. The specification of the algorithm can be divided into 3 logical steps, where each step carries out an unique sort of computation (Figure 3) –

1) Building a pairwise distance metric – This step in the algorithm computes all pair-wise alignments and store in similarity matrix M so that,

$$M_{i,j} = sim(s_i, s_j)$$

Where $sim(s_i, s_j)$ is a similarity metric between between the sequences i and j. The output accuracy of the Clustal algorithms are quite sensitive to the choice of similarity metric.

2) Finding a guide tree such that the two most common (sets of) sequences have a common root – The original Clustal-W algorithms computes the distance from the ancestor of s_i and s_j to all other sequences as the average of the distances to s_i and s_j and continue this process untill there is only one row and column in M. This is akin to hierarchical clustering of sequences to form a tree. We focus on the Clustal-ω variant [42] which uses a HMM to construct the best possible neighbor tree using priors. Several other schemes of generating the guide tree have been suggested (using statistical models dealing with phylogeny trees and motifs, but these are beyond the scope of this discussion).
3) Use the neighbor-tree to guide pairwise alignments – Perform pair-wise alignments in the order given by the guide tree thereby reducing the problem of muti-sequence alignment to set of pairwise alignments.
4) Sometimes an optional step of calculating the "goodness" of the alignment is performed. We do not discuss this step and the computations involved in this paper.

B. The Decomposition

Building upon the algorithm described in the previous sub-section, we now look at how we map the computations of the Clustal-W algorithm to the previously mentioned core computations. Each of the steps of the algorithm can now be broken down into a set of core-computations and a set of *glue codes*, which convert the outputs of one core computation to the input of the next. It is to be noted that all decompositions of algorithms might not require this glue code.

We decompose the Clustal-W algorithm as follows (second sequence of blocks in figure 3)–

1) Calculating the pairwise distance between all the input sequences – This can be accomplished by using a pairwise alignment using the *Smith- Waterman* or the *Needleman-Wunsch* algorithms. Whose alignment score is calculated using the Clustal-W scoring function. We can divide this task as a traditional Smith-Waterman algorithm block with a glue code logic, that accumulates all pairwise alignments and returns a matrix of containing the pairwise scores of each alignment.
2) Constructing the best neighbor tree for pairwise alignments – Using the pairwise alignment matrix, a tree is constructed in which the two most common sequences have a common root. In our investigation of the Clustal-ω implementation, we see that this is done by using the Viterbi algorithm over the set of input sequences. The glue code relating to this stage, takes the output of the most likely sequence of hidden states of the HMM model (this corresponds to a nucleotide sequence) and aggregates the data into a into a tree, where the two nodes with a common parent are most likely to be most similar.
3) Calculating the pairwise alignments taking into account the guide tree – The final stage of the Clustal-W MSA pipeline is a Smith-Waterman based alignment and assembly by taking nodes in the order specified in the guide tree.

C. Discussion

We envision the decomposition of bioinformatics algorithms being used in the following ways –

1) We can use a data-streaming paradigm for defining these analyses. In such a streaming solution, each of the core computations and the glue codes form a set of operators. This allows us to explore the avenue of using commodity software packages that allow streaming on large distributed systems like Apache Storm or Apache Spark to allow for migration of these genomics applications to the cloud.
2) A framework for defining generic bioinformatics analyses – We can use these memory-less computations to define

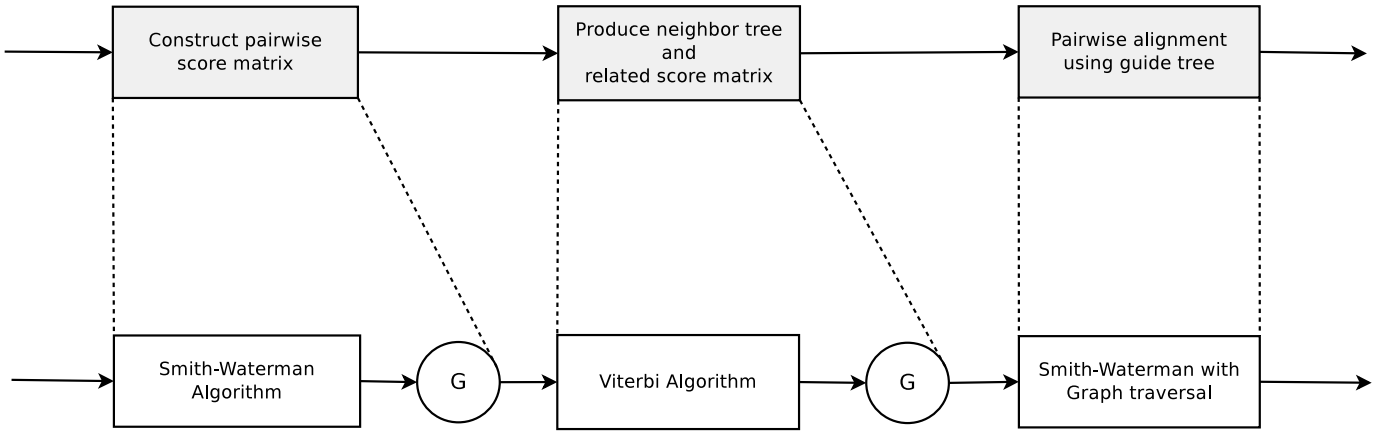

Fig. 3: Decomposing the Clustal-W algorithm – The first sequence of blocks shows the algorithm broken into logical blocks. The second sequence of blocks demonstrates how each block in the first sequence is broken into a core computation and related glue code to convert the outputs of one core computation to the input of the next.

a set of primitives which can be used as a language to describe most current genomics algorithms. We can reduce the problem of constructing these algorithms as that of chaining together these memory-less computations and necessary glue codes together.

3) In such a framework we could expect that the computationally expensive blocks (the core computations), are actually implemented by computer scientists/engineers, who have an understanding of the underlying hardware and can write code that can extract optimal performance from it by using cutting edge accelerators and programming paradigms. Acceleration of these core computations on GPUs[4][2][43] and FPGAs[3][1] is a well studied problem. Having these resources available, the biologists/bioinformaticians , would then only have to restrict themselves to write the glue logic for their particular application, which in most cases is performing data aggregation and not the "meat" of the computation.

IV. Need of Further Analysis

Building upon the decomposition of the algorithms in the previous sections, in this section, we now look at possible optimizations that can be applied to the core computations, that were difficult to apply to the original problem, because of its complexity and size.

To demonstrate our point, we consider the straightforward example of optimizing memory access patterns in the implementation of the Viterbi algorithm in the HMMer toolset. We compare the performance, in terms of completion times of an alignment of short reads to a human chromosome-1 reference. Our comparison is between a naive Viterbi algorithm and a memory optimized Viterbi algorithm[2]. Our results (figure 4) are interesting, because they show a large improvement in performance for the task, when the size of the model is small, but for larger models, a reduced but still considerable speedup. This leads us to believe that this line of investigation, looking at optimization patterns at the core computation level warrants further and more detailed examination.

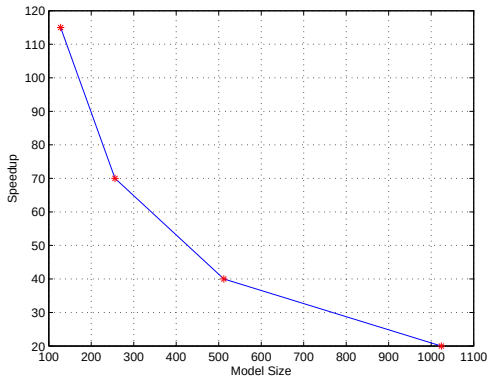

Fig. 4: Comparing the speedup of the HMMer based viterbi algorithm with model size. The values of speedup are normalized to the running time of the naive implementation

V. Conclusions and Future Work

We hope to use this analysis of a representative set of algorithms and tools, to motivate our design and construction of –

1) A single machine with re-configurable hardware which uses these accelerated primitives to perform these analyses at much higher throughput than is available in the current generation of machines (and software).
2) Extending this concept to a cluster of machines, where each machine is optimized for particular classes of computation and we try to stream data across the nodes for better performance.

We conclude by listing some points of discussion, which might prove to be interesting, to pursue in the future and in fact, it might be necessary to answer a few of these questions before we go about implementing the system design presented here –

1) Completeness of our set of memory-less computations – Drawing from our analysis, we present here a set of

computations that we find occur frequently in a representative set of algorithms that we study here. However it is not guaranteed that this set of memory-less computations will cover the entire space of algorithms. This can be especially problematic in dealing with the extensibility of the current system with new algorithms.

2) Is further decomposition of these memory-less computations in yet simpler blocks possible? – Decomposition of the algorithms into memory-less computations demonstrated in this paper, might not be the last level of decomposition that we are interested to halt at. It is interesting to note that decomposing these algorithms further might be helpful, especially in making it easier to implement some of these computations as primitive instructions using reconfigurable hardware like FPGAs.
3) What happens in the case that the glue codes are more computationally intensive than the memory-less computations? – Our approach of decomposing strongly coupled algorithms into it's constituent blocks and then assembling them as weakly coupled blocks is limited by the assumption that we are not introducing more computationally intensive blocks. From our current understanding, this problem can avoided by using the memory-less computations as *gray boxes*, where intermediate computation is available as input for other stages of the computation. This holds for our MSA example, but does this generally hold true?

REFERENCES

[1] S. Derrien and P. Quinton, "Hardware acceleration of hmmer on fpgas," *Journal of Signal Processing Systems*, pp. 53–67, 2010.

[2] Z. Du, Z. Yin, and D. A. Bader, "A tile-based parallel viterbi algorithm for biological sequence alignment on GPU with CUDA," in *IPDPS Workshops*. IEEE, 2010.

[3] P. Zhang, G. Tan, and G. R. Gao, "Implementation of the smith-waterman algorithm on a reconfigurable supercomputing platform." in *HPRCTA*, 2007, pp. 39–48.

[4] Y. Liu, B. Schmidt, and D. Maskell, "Cudasw++2.0: enhanced smith-waterman protein database search on cuda-enabled gpus based on simt and virtualized simd abstractions," *BMC Research Notes*, p. 93, 2010.

[5] T. H. Jukes and C. R. Cantor, "Evolution of protein molecules," 1969.

[6] P. Legendre and L. Legendre, *Numerical ecology*. Elsevier, 2012.

[7] N. Saitou and M. Nei, "The neighbor-joining method: a new method for reconstructing phylogenetic trees." *Molecular biology and evolution*, pp. 406–425, 1987.

[8] I. Elias and J. Lagergren, "Fast computation of distance estimators," *BMC bioinformatics*, vol. 8, no. 1, p. 89, 2007.

[9] D. PLOTREE and D. PLOTGRAM, "Phylip-phylogeny inference package (version 3.2)," 1989.

[10] S. Kumar, M. Nei, J. Dudley, and K. Tamura, "Mega: a biologist-centric software for evolutionary analysis of dna and protein sequences," *Briefings in bioinformatics*, pp. 299–306, 2008.

[11] M. Larkin, G. Blackshields, N. Brown, R. Chenna, P. A. McGettigan, H. McWilliam, F. Valentin, I. M. Wallace, A. Wilm, R. Lopez *et al.*, "Clustal w and clustal x version 2.0," *Bioinformatics*, pp. 2947–2948, 2007.

[12] D. L. Swofford, *PAUP*. Phylogenetic Analysis Using Parsimony (*and Other Methods). Version 4*. Sinauer Associates, 2003.

[13] A. Boc, V. Makarenkov *et al.*, "T-rex: a web server for inferring, validating and visualizing phylogenetic trees and networks," *Nucleic acids research*, vol. 40, no. W1, pp. W573–W579, 2012.

[14] P. Goloboff, S. Farris, and K. Nixon, "Tnt (tree analysis using new technology)." 2000.

[15] W. M. Fitch, E. Margoliash *et al.*, "Construction of phylogenetic trees," *Science*, pp. 279–284, 1967.

[16] D. Bryant and P. Waddell, "Rapid evaluation of least-squares and minimum-evolution criteria on phylogenetic trees," *Molecular Biology and Evolution*, pp. 1346–1359, 1998.

[17] G. H. Gonnet, M. T. Hallett, C. Korostensky, and L. Bernardin, "Darwin v. 2.0: an interpreted computer language for the biosciences," *Bioinformatics*, pp. 101–103, 2000.

[18] J. P. Huelsenbeck and K. A. Crandall, "Phylogeny estimation and hypothesis testing using maximum likelihood," *Annual Review of Ecology and Systematics*, pp. 437–466, 1997.

[19] A. Siepel and D. Haussler, "Phylogenetic hidden markov models," in *Statistical methods in molecular evolution*. Springer, 2005, pp. 325–351.

[20] S. Guindon, J.-F. Dufayard, V. Lefort, M. Anisimova, W. Hordijk, and O. Gascuel, "New algorithms and methods to estimate maximum-likelihood phylogenies: assessing the performance of phyml 3.0," *Systematic biology*, pp. 307–321, 2010.

[21] F. Ronquist, M. Teslenko, P. van der Mark, D. L. Ayres, A. Darling, S. Höhna, B. Larget, L. Liu, M. A. Suchard, and J. P. Huelsenbeck, "Mrbayes 3.2: efficient bayesian phylogenetic inference and model choice across a large model space," *Systematic biology*, pp. 539–542, 2012.

[22] A. Stamatakis, "Raxml-vi-hpc: maximum likelihood-based phylogenetic analyses with thousands of taxa and mixed models," *Bioinformatics*, pp. 2688–2690, 2006.

[23] K. Tamura, D. Peterson, N. Peterson, G. Stecher, M. Nei, and S. Kumar, "Mega5: molecular evolutionary genetics analysis using maximum likelihood, evolutionary distance, and maximum parsimony methods," *Molecular biology and evolution*, pp. 2731–2739, 2011.

[24] J. A. Nylander, J. C. Wilgenbusch, D. L. Warren, and D. L. Swofford, "Awty (are we there yet?): a system for graphical exploration of mcmc convergence in bayesian phylogenetics," *Bioinformatics*, pp. 581–583, 2008.

[25] T. F. Smith and M. S. Waterman, "Identification of common molecular subsequences," *Journal of molecular biology*, pp. 195–197, 1981.

[26] S. B. Needleman and C. D. Wunsch, "A general method applicable to the search for similarities in the amino acid sequence of two proteins," *Journal of molecular biology*, pp. 443–453, 1970.

[27] K. Khafizov, R. Staritzbichler, M. Stamm, and L. R. Forrest, "A study of the evolution of inverted-topology repeats from leut-fold transporters using alignme," *Biochemistry*, pp. 10 702–10 713, 2010.

[28] R. C. Gentleman, V. J. Carey, D. M. Bates, B. Bolstad, M. Dettling, S. Dudoit, B. Ellis, L. Gautier, Y. Ge, J. Gentry *et al.*, "Bioconductor: open software development for computational biology and bioinformatics," *Genome biology*, p. R80, 2004.

[29] J. E. Stajich, D. Block, K. Boulez, S. E. Brenner, S. A. Chervitz, C. Dagdigian, G. Fuellen, J. G. Gilbert, I. Korf, H. Lapp *et al.*, "The bioperl toolkit: Perl modules for the life sciences," *Genome research*, pp. 1611–1618, 2002.

[30] P. Rice, I. Longden, and A. Bleasby, "emboss: the european molecular biology open software suite," *Trends in genetics*, pp. 276–277, 2000.

[31] S. F. Altschul, T. L. Madden, A. A. Schäffer, J. Zhang, Z. Zhang, W. Miller, and D. J. Lipman, "Gapped blast and psi-blast: a new generation of protein database search programs," *Nucleic acids research*, pp. 3389–3402, 1997.

[32] W. R. Pearson and D. J. Lipman, "Improved tools for biological sequence comparison," *Proceedings of the National Academy of Sciences*, pp. 2444–2448, 1988.

[33] K. Okonechnikov, O. Golosova, M. Fursov *et al.*, "Unipro ugene: a unified bioinformatics toolkit," *Bioinformatics*, pp. 1166–1167, 2012.

[34] R. Hughey and A. Krogh, "Sam: Sequence alignment and modeling software system," 1995.

[35] U. Roshan and D. R. Livesay, "Probalign: multiple sequence alignment using partition function posterior probabilities," *Bioinformatics*, pp. 2715–2721, 2006.

[36] C. Notredame, D. G. Higgins, and J. Heringa, "T-coffee: A novel method for fast and accurate multiple sequence alignment," *Journal of molecular biology*, pp. 205–217, 2000.

[37] A. McKenna, M. Hanna, E. Banks, A. Sivachenko, K. Cibulskis, A. Kernytsky, K. Garimella, D. Altshuler, S. Gabriel, M. Daly *et al.*, "The genome analysis toolkit: a mapreduce framework for analyzing next-generation dna sequencing data," *Genome research*, pp. 1297–1303, 2010.

[38] H. Li, B. Handsaker, A. Wysoker, T. Fennell, J. Ruan, N. Homer,

G. Marth, G. Abecasis, R. Durbin *et al.*, "The sequence alignment/map format and samtools," *Bioinformatics*, pp. 2078–2079, 2009.

[39] D. G. Higgins and P. M. Sharp, "Clustal: a package for performing multiple sequence alignment on a microcomputer," *Gene*, vol. 73, no. 1, pp. 237 – 244, 1988.

[40] ——, "Fast and sensitive multiple sequence alignments on a microcomputer," *Computer applications in the biosciences : CABIOS*, vol. 5, no. 2, pp. 151–153, 1989.

[41] J. D. Thompson, D. G. Higgins, and T. J. Gibson, "CLUSTAL W: improving the sensitivity of progressive multiple sequence alignment through sequence weighting, position-specific gap penalties and weight matrix choice," *Nucleic Acids Res.*, vol. 22, no. 22, pp. 4673–4680, Nov 1994.

[42] F. Sievers, A. Wilm, D. Dineen, T. J. Gibson, K. Karplus, W. Li, R. Lopez, H. McWilliam, M. Remmert, J. Soding, J. D. Thompson, and D. G. Higgins, "Fast, scalable generation of high-quality protein multiple sequence alignments using Clustal Omega," *Mol. Syst. Biol.*, p. 539, 2011.

[43] D. R. Horn, M. Houston, and P. Hanrahan, "Clawhmmer: A streaming hmmer-search implementation," in *Proceedings of the 2005 ACM/IEEE Conference on Supercomputing*. IEEE Computer Society, 2005.