FMSA: FPGA-Accelerated ClustalW-Based Multiple Sequence Alignment Through Pipelined Prefiltering

Atabak Mahram Martin C. Herbordt
Department of Electrical and Computer Engineering
Boston University, Boston, MA

***Abstract*—Multiple Sequence Alignment (MSA) is perhaps second only to sequence alignment in overall importance in Bioinformatics, being critical, e.g., in determining the structure and function of molecules from putative families of sequences. But while pairwise sequence alignment has been the subject of scores of FPGA acceleration studies, MSA only a few. The most important of these accelerate Clustal-W, the most commonly used MSA code, by either implementing the first of three phases (over 90% of the run time) with Dynamic Programming (DP) methods, or by accelerating the third phase which consumes most of the remaining time. We use a new approach: we apply prefiltering of the kind commonly used in BLAST to perform the initial all-pairs alignments. This results in a speedup of from 80x to 190x over the CPU code (8 cores) and speedup of from 2.5x to 8x over DP/FPGA- and GPU-based methods. When combined with a recently published method for phase 3, and using the original software for phase 2, the end-to-end speedup is at least 50x over an 8-core implementation of the original code. The quality is comparable to the original according to a commonly used benchmark suite evaluated with respect to multiple distance metrics.**

***Keywords*-High Performance Reconfigurable Computing; Bioinformatics; Multiple Sequence Alignment**

I. Introduction

A fundamental insight of bioinformatics is that biologically significant polymers such as proteins and DNA can be abstracted into character strings (sequences). This allows biologists to use approximate string matching for pairwise sequence alignment (SA) to determine, e.g., how a newly identified protein is related to those previously analyzed, and how it has diverged through mutation [1]. Multiple Sequence Alignment (MSA) extends this idea in the obvious way to more than two sequences: gaps are inserted as necessary to define a mapping of the sequences to rows of a matrix such that all columns have at least one letter.

MSA is the critical tool for extracting and representing biologically important, yet (potentially) faint or widely dispersed, commonalities from a set of strings [2]. While SA is used to assign possible functions to a protein, MSA goes to the next level. Among its uses are prediction of function and secondary (2D and 3D) structure, identification of the residues important for specificity of function, creation of alignments of distantly related sequences, and revealing clues about evolutionary history [3].

This work was supported in part by the NIH through award #R01-RR023168-01A1 and by the MGHPCC.

While SA is typically used in database search (finding correlations of one sequence with millions of anonymous candidates), MSA is generally applied to some number of sequences that are already hypothesized to have some commonality. And though it is often the case that some sequences are better understood or more important than others, MSA is basically an all-to-all matching problem. Another difference is that while there is a consensus on the evaluation of SA alignments, based on Karlin-Altschul statistics, with MSA there is no objective way to define an unambiguously correct alignment [4]. These last facts have the following consequence. Evaluating MSA applications requires either expert knowledge, or its surrogate through preselected sets of related sequences (e.g., BAliBASE [5]) and encoded evaluation metrics (e.g., MetAl [6] and BaliScore [5]).

In an MSA workflow some number of sequences k of length n are aligned. The median value for n is about 300, but is often closer to 1000; k can range from a few to a few thousand sequences or more. Optimal MSA algorithms have been created by extending DP-based SA to higher dimensions. These have exponential complexity $O(n^k)$. Applying restrictions (e.g., [7], [8]) results in tremendous speedups making them plausible for k up to small double digits. Larger k, however, requires use of heuristics such as progressive refinement [9]. These codes typically run in three phases: (i) an all-to-all phase where all pairs are aligned and scored, (ii) a tree building phase where a guide tree is built that has sequences as its leaves and whose interior nodes represent alignments, and (iii) a final phase where all pairs of nodes are aligned.

The most commonly used MSA code is ClustalW [10]. But while it is exponentially more efficient than the optimal methods, it still takes hours to days of CPU time for larger runs. And since MSA is often a subroutine of a more complex task, such as finding evolutionary relationships, its acceleration is critical. The first phase of ClustalW consists of over 90% of the execution time. It has been accelerated both with FPGAs [11] and GPUs [12]. Both of these studies follow the serial code in using dynamic programming (DP) for the all-pairs alignments and report factors of $40\times$ to $50\times$ speedups over a single core, respectively. We find that

when the FPGA-DP method is ported to updated FPGAs and multicore CPUs the speedup is in a similar range, but with some variance, i.e., from $18\times$ to $58\times$. Recently, Lloyd and Snell have accelerated a generic third phase, which for ClustalW takes most of the remaining time, on FPGAs and obtained a speedup of up to $150\times$ versus a single core [13].

We use a different approach in creating a ClustalW-based FPGA-accelerated MSA (FMSA). Just as BLAST applies multiple passes of heuristics to emulate DP-based SA, so we apply BLAST-inspired filters to the pairwise alignments. In particular we use a two-hit filter (seeding pass) [14] followed directly in a pipeline by an ungapped alignment (ungapped extension pass) [1], [15]. For the latter we emulate the ungapped mode of NCBI BLASTP [16]. There are two versions of FMSA, *fast* (FMSA-f) and *emulation* (FMSA-e). In both cases we use a scoring function analogous to that used by ClustalW: rather than returning an e-value, FMSA computes a function based on identity counts. In fast mode, these scores are sent directly to the second phase of ClustalW to complete the processing. In emulation mode, some fraction of the high-scoring pairs are rescored using the DP-based method of Oliver et al. [17] that emulates the ClustalW scoring function precisely. The result is a factor of from $80\times$ to $189\times$ speedup with respect to 8-way parallel CPU code with the lower number corresponding to achieving results with quality comparable to the original.

The primary result is an FPGA-accelerated version of ClustalW that achieves both substantial speedup over previous methods for computationally intensive data sets and high quality results that, especially in emulation mode, closely agree with those generated by the original code. The mechanism is the primary intellectual contribution of the work and has three parts: (i) the overall approach where we apply prefiltering based on ungapped alignment and rescore as necessary, (ii) the modification of the original components to support an MSA rather than an SA scoring function, and (iii) the redesign of the filter sequence into a pipeline to avoid costly system overhead and reconfiguration. The significance is that—when coupled with the work of Oliver, et al. [17], and of Lloyd and Snell [13]—this could become the FPGA-accelerated MSA method of choice.

The rest of the paper is organized as follows. We begin with a brief review of progressive alignment for MSA and ClustalW. We continue with details of our design. After that comes the methods and results, followed by a discussion and future work.

II. Background

A. Basics of MSA for Biological Sequences

There are a number of heuristic MSA codes that use progressive sequence alignment. They differ in (i) the order of alignments, (ii) whether there is a single growing alignment or multiple subfamilies that are later aligned to each other, and (iii) the procedure used to align score sequences and alignments against an existing sequence or alignment. Generally a binary tree is constructed to guide the order of alignments with the most similar pairs, being the most reliable, being aligned first [4].

Scoring MSAs is an active area of research, but a commonly used metric is the Sum-of-Pairs or SP score. This is a direct extension of pairwise scoring. We follow the discussion in Gusfield [2]: Given a multiple alignment M of k strings, an induced pairwise alignment between strings S_i and S_j is obtained by removing all rows except for the ith and jth. The pairwise score can be calculated using a standard SA function, or one selected for MSA. The SP score is the sum of all of the pairwise scores.

In order to test the quality of an MSA method, a preferred method is to evaluate it with a *golden* or *reference* data set; i.e., an alignment that has been created by a domain expert to deal with a specific, realistic, biological scenario. The BAliBASE 2.1 benchmark alignment database [5] contains a number of case studies giving both sequences and a putative ideal reference alignment. Quality (determined by running the program BAliScore) is based on the SP score [18] and computed as follows. For all columns in the test alignment, for each pair of residues, a score of 1 is given if the residues are aligned with each other in the reference alignment and a 0 otherwise. This sum is normalized by the scores computed for the reference alignment.

A recent paper describes another set of distance measures for MSAs [6]. Three of the four are appropriate here:
hd: *Simple homology distance* gives the possibility that a randomly selected base x from an MSA will be aligned to a different location against a sequence randomly selected from the remaining sequences that do not contain x.
ssp: *Symmetrized SP* score is a symmetrized version of the SP score defined above.
pi: *Positional information* is incorporated into **hd** where gaps occur.

B. ClustalW Overview

Table I
PROFILE OF CLUSTALW BASE CODE W.R.T. VARIOUS DATA SETS. BB = BALIBASE, NCBIN = GENERATED FROM NCBI BLASTP RESULTS.

Benchmark	# of seq.	phase 1	phase 2	phase 3
BB: MYB	180	88.0%	0.3%	11.7%
BB: 7tm	128	90.4%	0.04%	9.5%
NCBI1	231	95.4%	0.2%	4.4%
NCBI2	1000	91.4%	0.4%	8.0%
Average	—	91.3%	0.2%	8.4%

ClustalW improves on the original progressive alignment methods by adding a number of heuristics. Most of these are incorporated into the second and third phases, however, and need not be detailed here. From Table 1 we see that most of the effort is in Phase 1.

The first phase creates a matrix of alignment scores for all sequence pairs. Note that the ClustalW code appears to

have been modified since the original paper; it is the code to which we refer. The scoring requires multiple passes. In the first two, a best local alignment is found using a variation of the Myers and Miller algorithm [19], which itself uses a variation of global alignment with DP. In the third pass, the actual score is determined from a count of the number of identical residue pairs in the optimal alignment. This number is then divided by the minimum of the lengths of the two sequences to create a similarity measure. The result is then subtracted from one to get the distance measure between the two sequences. One important point is that Myers and Miller is a memory efficient recursive global alignment algorithm: it divides the alignment space in half by dividing one sequence in half. It then finds the optimal point on the other sequence such that the concatenation of the two alignments of the subsequences on either side of the midpoint maximizes the global score. In this process it properly handles possible gaps in the neighborhood of the optimal midpoint.

We give this detail to show the challenge in exactly emulating the ClustalW scoring function. Besides the complexity, there can be multiple optimal alignments or traces between sequences; choosing the wrong one will lead to disagreement with the reference code. Oliver, et al., in their acceleration of the pairwise alignment phase [17] (with a method based on Smith-Waterman), have achieved near perfect agreement.

C. Target Systems

We briefly state our assumptions about the target systems with FPGA-based accelerators. They are typical for current products; details of appropriate FPGA-based systems can be found, e.g., in [20].

- The overall system consists of some number of standard nodes. Typical node configurations have 1-4 accelerator boards plugged into a high-speed connection (e.g., the Front Side Bus or PCI Express). The host node runs the main application program. Nodes communicate with the accelerators through function calls.
- Each accelerator board consist of 1-4 FPGAs, memory, and a bus interface. On-board memory is tightly coupled to each FPGA either through several interfaces (e.g., 6 x 32-bit) or a wide bus (128-bit). 4GB-64GB of memory per FPGA is currently standard.
- Besides configurable logic, the FPGA has dedicated components such as independently accessible multiport memories (e.g., 1000 x 1KB) called Block RAMs (or BRAMs) and a similar number of multipliers. FPGAs used in High Performance Reconfigurable Computing typically run at 200 MHz, although with optimization substantially higher operating frequencies can sometimes be achieved.

We have been developing FMSA using a particular *reference system* consisting of a standard high-end PC with a Gidel PROCe III accelerator board. The FPGA is an Altera Stratix III 260E, a 2007-era 65nm device.

All algorithms and implementations map immediately onto similar systems (i.e., from other vendors for both boards and FPGAs). FMSA also maps to multi-FPGA accelerators such as the Gidel PROCStar III, which has four FPGAs, and to large FPGA-based systems. For the latter we are targeting the Novo-G, which has over 288 FPGAs [21].

III. Design and Implementation

A. Design Overview

In all-pairs alignment FMSA constructs a database of the sequences to be multiply aligned and consecutively matches sequences against the remainder. While DP-based methods have excellent performance with respect to software, heuristic methods are much faster still, in the same way that BLAST is substantially faster than Smith-Waterman. FMSA uses two FPGA-based filters:

- The Two-Hit Filter is based on the two-hit seeding algorithm. All alignments are evaluated as to whether or not they contain a two-hit seed. We base our Two-Hit Filter on the two-hit seeding algorithm created for Mercury BLAST [14] and refined for CAAD BLAST [1].
- The Exhaustive Ungapped Alignment (EUA) Filter scores every possible alignment between the sequence and the database. For each sequence in the database it returns the scores of the highest scoring alignments. We base our EUA Filter on the TreeBLAST algorithm described in [15].

For FMSA-e we add an additional FPGA pass. The idea here is to expend marginal additional effort to improve agreement with the original code. For each sequence we rescore the highest scoring 10% of the sequences with the DP-based scoring function [17] that nearly perfectly emulates the ClustalW scoring function.

B. FMSA Scoring

Before describing the details of the FMSA filters we present some results of various possible scoring functions. We begin by introducing some terminology. S_i is an input string, $LGA(S_i, S_j)$ represents the Optimal Local Gapped Alignment between S_i and S_j, $Score(Alignment_i)$ returns the raw score of an Alignment i, $NID(Alignment_i)$ returns the number of identical residue pairs in an Alignment i, $LUA(S_i, S_j, k)$ represent the kth best Local Ungapped Alignment between S_i and S_j based on the raw scores.

For each pair of sequences S_i and S_j, ClustalW calculates the distance as follows:

$$dist_{ref}(S_i, S_j) = 1 - \frac{NID(LGA(S_i, S_j))}{min(len(S_i, S_j))}$$

Because our proposed method involves rescoring top pairs to improve agreement, we are looking for the scoring function

which returns, as much as possible, the same top scores as ClustalW. We tested five methods:

1) Find the best ungapped local alignment and count the number of identities:

$$dist_1(S_i, S_j) = 1 - \frac{NID(LUA(S_i, S_j, 1))}{min(len(S_i), len(S_j))}$$

2) Same as 1, except add the top two best local ungapped alignments:

$$dist_2(S_i, S_j) = 1 - \frac{\sum_{i=1}^{2} NID(LUA(S_i, S_j, i))}{min(len(S_i), len(S_j))}$$

3) Same as 1 and 2, except add the top five best local ungapped alignments:

$$dist_3(S_i, S_j) = 1 - \frac{\sum_{i=1}^{5} NID(LUA(S_i, S_j, i))}{min(len(S_i), len(S_j))}$$

This method is similar to the ungapped option in NCBI BLASTP.

4) Find the best local ungapped alignment and use the raw score:

$$dist_4(S_i, S_j) = 1 - \frac{\sum_{i=1}^{5} Score(LUA(S_i, S_j, i))}{min(len(S_i), len(S_j))}$$

5) Find the best local gapped alignment and use the raw score (Smith-Waterman):

$$dist_5(S_i, S_j) = 1 - \frac{Score(LGA(S_i, S_j))}{min(len(S_i), len(S_j))}$$

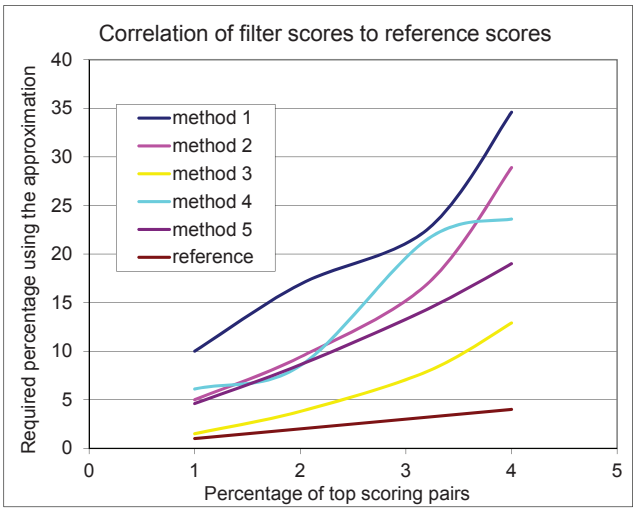

Figure 1. For a number of potential scoring functions, graph shows the number that must be rescored to cover a given fraction of the top ClustalW scores.

The evaluation of these scoring functions is shown in Figure 1. The series show the fraction of high-scoring pairs that must be rescored to guarantee that some top fraction of the high scoring pairs of ClustalW are matched. The lower the series the better. Note that for method 3 rescoring 10% of the highest scoring pairs covers the top 3.6% from ClustalW. For method 1, nearly 30% must be rescored to achieve the same result.

Table II
MEASURE OF THE BIAS AND STANDARD DEVIATION IN PAIRWISE SCORES BETWEEN ORIGINAL CLUSTALW AND FILTER OUTPUT. THE TWO MEDIUM SIZED DATABASES FROM BALIBASE ARE USED TOGETHER WITH A SYNTHETIC DATA SET GENERATED FROM NCBI BLASTP.

Database	# of seq.	Avg. of diffs.	STD of diffs.
7tm	128	-0.01	0.03
myb	180	-0.02	0.09
NCBI	231	-0.05	0.03

We selected method 3 for use by FMSA-e. Table II shows the result of comparing all of the scores generated by FMSA-e with ClustalW. There is a slight bias, probably the result of some redundancy from summing multiple alignments without checking for redundancy. We compensate for this as a function of average sequence length.

C. Filter Details

From the previous section we see that the scoring function requires finding the top ungapped alignments and then scanning them for identities. We use a two stage process. In the first, we follow standard BLAST procedure and eliminate all alignments where there are not two seed matches within a certain distance (two-hit filter). In the second, we simultaneously perform two functions: we exhaustively scan all alignments for high-scoring ungapped local alignments and we count the identities in those alignments.

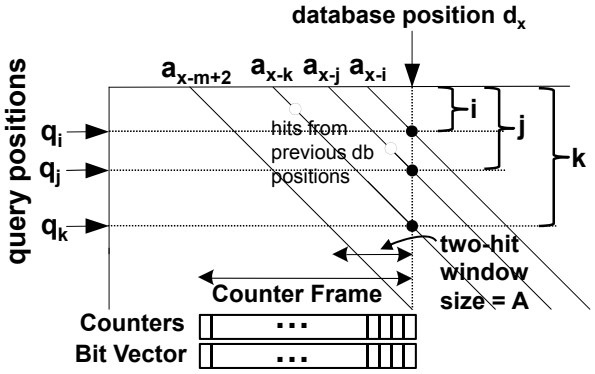

Figure 2. The Two-Hit filter is processing the xth 3-mer in the database sequence. There are three hits. The hit on alignment a_{x-j} is within A of a previous hit, and so is part of a two-hit event. This is determined by comparison with the corresponding Counter value; its bit in the Bit Vector will be set.

The basic function of the two-hit filter is shown in Figure 2. The design is generally similar to that used in the Mercury BLAST seeding pass [14] and refined in CAAD BLASTP [1].

The exhaustive ungapped alignment filter is based on the TreeBLAST scheme described in [15]. The Tree-BLAST

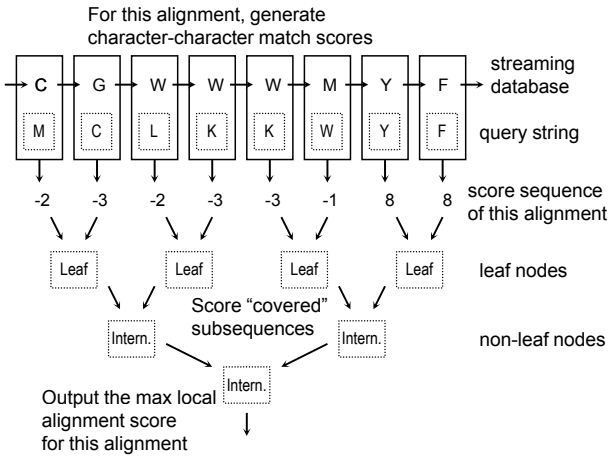

Figure 3. TreeBLAST structure for $m = 8$.

filter has two essential properties. First, it exhaustively evaluates every possible ungapped alignment between query and database. And second, it is fully pipelined: it evaluates the database at streaming rate.

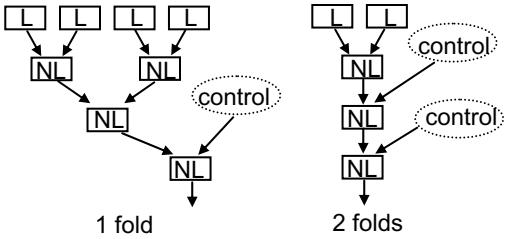

Figure 4. An 8 leaf tree folded once and twice. L = leaf node, NL = non-leaf node.

Queries larger than the tree size can be handled through folding as shown in Figure 4. The key idea is that we can trade off space, and therefore parallelism, for latency. We take advantage of this idea when we combine the two filters. The method is as follows. We stream the database through the filters. But alignments that do not pass the two-hit criterion (around 97%) are marked. These can then be skipped by the TreeBLAST filter. Folding allows the TreeBLAST filters to both be more compact and more efficient.

Figure 5 shows the overall scheme: parallel database streams feed the two-hit filters, which in turn feed a TreeBLAST filter. This structure is replicated some number of times depending on the sizes of the strings and of the FPGA.

TreeBLAST is capable of consuming 3 to 5 characters at each clock thanks to the two-hit filter data. TreeBLAST reads in 16 characters and the corresponding filter bits as required. After processing of the data of one two hit-filter, it starts working on the next sequence from the next two-

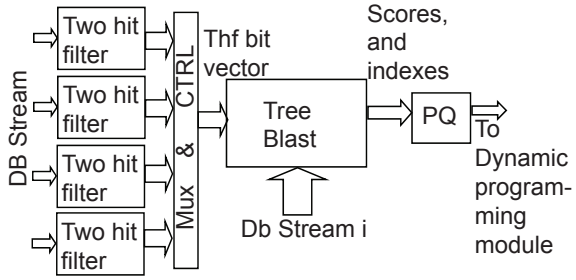

Figure 5. Shown is the pipelining scheme of the two sets of filters.

hit filter. In order to keep all of the units busy, the database sequences are sorted based on length and multiplexed among multiple two-hit filters. As a result the time required to process successive sequences is nearly equal. Note that there are 2 priority queues in the design. One priority queue is inside the TreeBLAST module and stores the top 5 local ungapped alignment scores. Another priority queue is outside TreeBLAST module to store the indexes of the top 10% scoring sequences. After TreeBLAST streams the entire database, the data in this second priority queue is passed to the dynamic programming module to perform the refinement of the top scoring pairs.

D. Implementation Details

We now briefly describe the resource utilization for different sequence sizes. For the Stratix III, for most problem sizes, we are able to map 16 two-hit filter units and 4 TreeBLAST units in a pipeline as just described. For small problems with a maximum sequence size of less than 256, we can map eight replications (32 two-hit filters); if larger than 1024 we can map two (8 two-hit filters). The maximum sequence size in the database is used to pick the proper programming file to load to the FPGA. With 1 GB/s DMA capacity on the board, the transfer of sequence neighborhoods from host to the device memory takes a negligible fraction of a second. On the device, with 2 memory modules, each with nearly 4 GB/s bandwidth, 16 two hit filters and 4 tree BLAST modules easily work in parallel without hitting the bandwidth barrier. With the current working frequency of 140Mhz the required bandwidth adds up to 32*140M =4.5 GBs.

For the emulation mode, each DP processing element requires 160 ALMs and one memory block RAM (M9k). On the Stratix III, if the maximum sequence size is less than 256, we can map two replications of the systolic array to FPGA, while for 256-512 (average case) we can fit 1 instance. For larger sequences, folding is necessary. When folded n times, the streaming rate is reduced by a factor of n.

Table IV
QUALITY MEASURE OF FMSA-F AND ORIGINAL CLUSTALW WITH RESPECT TO THE BALIBASE BENCHMARK SUITE USING SP FROM THE BALISCORE CODE.

Database	ClustalW v. Reference	FMSA-f v. ClustalW	FMSA-f v. Reference	FMSA-e v. ClustalW	FMSA-e v. Reference
myb	0.97	0.84	0.85	0.98	0.99
7tm	0.82	0.78	0.76	0.88	0.80
NCBI	—	0.93	—	0.94	—

Table V
QUALITY MEASURE OF FMSA-F, FMSA-E, AND ORIGINAL CLUSTALW WITH RESPECT TO THE BALIBASE BENCHMARK SUITE USING VARIOUS DISTANCE METRICS FROM METAL [6]. NOTE THAT SMALLER NUMBERS DENOTE CLOSER MATCHES AND HIGHER QUALITY.

	ClustalW v. Reference			FMSA-f v. Reference			FMSA-e v. Reference		
Database	**hd**	**ssp**	**pi**	**hd**	**ssp**	**pi**	**hd**	**ssp**	**pi**
myb	0.28	0.55	0.26	0.38	0.61	0.33	0.27	0.54	0.24
7tm	0.30	0.38	0.24	0.37	0.47	0.31	0.32	0.41	0.26

Table VI
PERFORMANCE FOR MSA RUNS OF 1000 SEQUENCES. ALL TIMES IN SECONDS. THE 8 CORE PC ASSUMES IDEAL PARALLELIZATION. DP, FMSA-F, AND FMSA-E COLUMNS GIVE BOTH TIME AND SPEED-UP WITH RESPECT TO 8 CORE PC.

Max Seq. Len.	Overhead	FGPA filter	FPGA rescore	PC (8 cores)	DP	FMSA-E	FMSA-F
256	0.9	0.2	0.3	150	3.5/43x	1.4/107x	1.1/136x
512	1.6	0.7	0.7	434	7.5/58x	3.0/145x	2.3/189x
1024	2.4	1.2	3.0	549	31/18x	6.6/83x	3.6/152x

Table III
QUALITY MEASURE OF FMSA-F AND ORIGINAL CLUSTALW WITH RESPECT TO THE BALIBASE BENCHMARK SUITE USING SP FROM THE BALISCORE CODE.

Database	# of seq.	ClustalW	FMSA-F
7tm	128	0.822	0.747
myb	180	0.969	0.850
Kinase3	19	0.777	0.827
Kinase2	18	0.739	0.738
1ajsa	28	0.405	0.464
1idy	27	0.591	0.554
1lvl	24	0.836	0.881
1aboA	5	0.688	0.558
1lcf	6	0.947	0.928

IV. RESULTS

Recall that FMSA can run in two modes, *fast* and *emulation*. Table III shows a measure of quality of the FMSA-f with respect to the BAliBASE benchmark and the SP metric. While not conclusive, we note that the results are not unreasonable, even without rescoring.

More detailed quality results, albeit for fewer sequences, are given in Tables IV and V. We use the two larger studies from BAliBASE, plus a synthetic database generated from NCBI BLASTP where we simply scanned a random sequence and retained the top 231 scoring sequences from NR. In Table IV we use the SP metric from BaliScore. We compare the original ClustalW code, FMSA-f, and FMSA-e all to the reference MSA. We find that FMSA-e has nearly identical quality as ClustalW, but with FMSA-f also showing good agreement. We also compare FMSA with ClustalW. For FMSA-e we find a high correlation; not surprisingly FMSA-f is not as correlated, but still high.

In Table V we show results with respect to the MetAl distance metrics. Again, we compare the original ClustalW code, FMSA-f, and FMSA-e all to the reference MSA. For FMSA-e we again find that the distance from the reference MSA is nearly identical to that of ClustalW with FMSA-f lagging somewhat, but still clearly in the same range.

Table VI shows performance of the original ClustalW, its DP-based acceleration, and the acceleration with FMSA-f and FMSA-e on the reference system. For the CPU-only version we simply assume the best case of perfect 8-way threading. This appears to be about a factor of two more than has been achieved so far (see [13] for a discussion), but appears to be plausible. The performance of FMSA-f is that of FMSA-e minus the FPGA rescore time and a small amount of the overhead. We note that FMSA is from $83\times$ to $189\times$ faster than the CPU version and from $2.5\times$ to $8.4\times$ faster than the DP-based method. The greater advantage is for the larger problem size.

V. DISCUSSION AND FUTURE WORK

We have described an FPGA-accelerated MSA program based on ClustalW. It differs from previous accelerations in that it uses BLAST heuristics rather than dynamic programming. It achieves many-fold speedup over the DP-based code, which itself has better performance than the CUDA version. We have created two versions, one that successfully emulates ClustalW, the other that gives results of somewhat lower quality, but with roughly twice the performance.

Our next step is to combine FMSA with the recently published work of Lloyd and Snell [13] that concentrates on the third phase. Even with the second phase entirely in software, such an FPGA-accelerated code should easily achieve a $50\times$ speedup over a fully parallelized CPU version of ClustalW (8-way). Perhaps the greatest utility, however, will be in

applying these heuristics to more complex problems where the advantage will be still greater.

REFERENCES

[1] A. Mahram and M. Herbordt, "Fast and Accurate NCBI BLASTP: Acceleration with Multiphase FPGA-Based Prefiltering," in *Proceedings of the 24th ACM International Conference on Supercomputing*, 2010, pp. 73–82.

[2] D. Gusfield, *Algorithms on Strings, Trees, and Sequences: Computer Science and Computational Biology*. Cambridge, U.K.: Cambridge U. Press, 1997.

[3] G. Barton, "Creation and analysis of protein multiple sequence alignment," in *Bioinformatics: A Practical Guide to the Analysis of Genes and Proteins*, A. Baxevanis and B. Ouellette, Eds. Wiley, 2001.

[4] R. Durbin, S. Eddy, A. Krogh, and G. Mitchison, *Biological sequence analysis*. Cambridge, U.K.: Cambridge University Press, 1998.

[5] A. Bahr, J. Thompson, J.-C. Thierry, and O. Poch, "BAliBASE (Benchmark Alignment dataBASE): enhancements for repeats, transmemebrane sequences and circular permutations," *Nucleic Acids Research*, vol. 29, no. 1, pp. 323–326, 2001.

[6] B. Blackburne and S. Whelan, "Measuring the distance between multiple sequence alignments," *Bioinformatics*, vol. Online preprint posted 12/23/2011, 2011.

[7] Y. Bilu, P. Agarwal, and R. Kolodny, "Faster algorithms for optimal multiple sequence alignment based on pairwise comparisons," *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, vol. 3, no. 4, pp. 408–422, 2006.

[8] H. Carrillo and D. Lipman, "The multiple sequence alignment problem in biology," *SIAM Journal of Applied Math.*, vol. 48, no. 5, pp. 1073–1081, 1988.

[9] D.-F. Feng and R. Doolittle, "Progressive sequence alignment as a prerequisite to correct phylogenetic trees," *Journal of Molecular Evolution*, vol. 25, pp. 351–360, 1987.

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

[11] T. Oliver and et al., "Multiple Sequence Alignment on an FPGA," in *Proc. International Conference on Parallel and Distributed Systems*, 2005, pp. 326–330.

[12] W. Liu, B. Schmidt, G. Voss, and W. Mueller-Wittig, "GPU-Clusatl W: Using Graphics Hardware to Accelerate Multiple Sequence Alignment," in *International Conference on High Performance Computing*, 2006, pp. 363–374.

[13] S. Lloyd and Q. Snell, "Accelerated large-scale multiple sequence alignment," *BMC Bioinformatics*, vol. 12, p. 466, 2011.

[14] A. Jacob, J. Lancaster, J. Buhler, B. Harris, and R. Chamberlain, "Mercury BLASTP: Accelerating protein sequence alignment," *ACM Transactions on Reconfigurable Technology and Systems*, vol. 1, no. 2, 2008.

[15] M. Herbordt, J. Model, B. Sukhwani, Y. Gu, and T. VanCourt, "Single pass streaming BLAST on FPGAs," *Parallel Computing*, vol. 33, no. 10-11, pp. 741–756, 2007.

[16] J. Park, Y. Qui, and M. Herbordt, "CAAD BLASTP: NCBI BLASTP Accelerated with FPGA-Based Pre-Filtering," in *Proc. IEEE Symp. on Field Programmable Custom Computing Machines*, 2009, pp. 81–87.

[17] T. Oliver, B. Schmidt, Y. Jakop, and D. Maskell, "Accelerating the viterbi algorithm for profile hidden markov models using reconfigurable hardware," in *LNCS v3991*, 2006.

[18] J. Thompson, F. Plewniak, and O. Poch, "A comprehensive comparison of multiple sequence alignment programs," *Nucleic Acids Research*, vol. 27, no. 13, pp. 2682–2690, 1999.

[19] E. Myers and W. Miller, "Optimal alignments in linear space," *Computational Applications in the Biosciences*, vol. 4, no. 1, 1988.

[20] S. Hauck and A. DeHon, *Reconfigurable Computing: The Theory and Practice of FPGA-Based Computing*. Morgan Kaufmann, 2008.

[21] A. George, H. Lam, and G. Stitt, "Novo-G: At the Forefront of Scalable Reconfigurable Computing," *Computing in Science and Engineering*, vol. 13, no. 1, 2011.