Parallel Computing 39 (2013) 819–833

Contents lists available at ScienceDirect

Parallel Computing

journal homepage: www.elsevier.com/locate/parco

Fine-grained parallel implementations for SWAMP+ Smith–Waterman alignment

Shannon Steinfadt *

Los Alamos National Laboratory, Los Alamos, NM 87545, USA

ARTICLE INFO

Article history:
Available online 4 September 2013

Keywords:
SIMD Parallel computing
Bioinformatics
Parallel co-processor
FPGAs
Sequence alignment
Smith–Waterman

ABSTRACT

More sensitive than heuristic methods for searching biological databases, the Smith–Waterman algorithm is widely used but has the drawback of a high quadratic running time. The faster approach extends Smith–Waterman using Associative Massive Parallelism (SWAMP+) for three different parallel architectures: ASsociative Computing (ASC), the ClearSpeed coprocessor, and the Convey Computer FPGA coprocessor. We show that parallel versions of Smith–Waterman can be successfully modified to produce multiple BLAST-like sub-alignments while maintaining the original precision. SWAMP+ combines parallelism and the novel extension producing multiple sub-alignments for pairwise comparisons.

Two parallel SWAMP+ implementations for the ASC model and the ClearSpeed CSX-620 use a wavefront approach. Both perform a full traceback in parallel memory, returning multiple sub-alignments. Results show a linear speedup for the 96 processing elements (PEs) on a single ClearSpeed chip.

The third SWAMP+ adaptation uses the non-associative Convey Computer FPGA coprocessor. The hybrid system has a Smith–Waterman algorithm suite designed to produce high-speed, high-throughput alignments, optimized for large databases. The Convey Computer Smith–Waterman algorithm suite was extended to produce the additional SWAMP+ sub-alignments efficiently.

The parallel sequence alignment algorithms were designed for three different computer systems, all of which contain extensions to produce multiple, additional sub-alignments. This work creates a speedup while providing a deeper exploration of the matched query sequences previously unavailable.

1. Introduction

One of the most used operations in bioinformatics domain is aligning sequences of DNA, RNA, amino acids and proteins. Modern "next-generation" or high-throughput sequencing platforms take in a genetic sample and produce thousands to millions of short sequences called *reads* for every run. These reads can be mapped onto other known *sequences* for study and comparison, or read mapping. The reads can also be pieced together (assembled) into one or more strings of DNA or RNA. Once assembled, these strings are often entered into a database such as a GenBank [1], UniProt [2], and EMBL [3].

When information or the functionality of a region of the sequence is known from prior study, that information is captured by *annotating* the subsequence. A functional region's utility is generally derived from the folded, three-dimensional shape of the corresponding protein, synthesized from the underlying genetic code. The protein folds are due to the interaction and the

* Tel.: +1 505 606 0136.
E-mail address: shannon@lanl.gov

0167-8191/$ - see front matter © 2013 Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.parco.2013.08.008

order of the molecules represented in the reads by the bases C, A, T and G in DNA or C, A, U, and G in RNA. Similar underlying molecular structure tends towards similar function. By comparing sequences, information from one annotated sub-sequence can be translated to a sequence that is highly similar or homologous. Sequences are often compared against part or all of a given database, which makes sequence alignment so popular. From short read mapping, genome assembly, and database comparisons, sequence alignment is one of the most used operations in bioinformatics.

There are three general approaches used for sequence alignment: exact (Smith–Waterman [4]), heuristic (BLAST [5,6]), and a combination of the two (the FASTA algorithm [7]).

The Smith–Waterman algorithm [4] provides high sensitivity, which does not miss the most similar (best scoring) alignments. The modified algorithm with optimizations by Gotoh [8] has an $O(n^2)$ running time and $O(n)$ memory requirements, where n is the sequence size of two sequences roughly the same size. Given the large size of genomic datasets, the computation time can be significant enough to make it infeasible.

To reduce the run time, several heuristic approaches, including BLAST [5,6] were developed. BLAST and its many variations including [9,10] have led researchers to expect pairwise alignment results that can contain *more than one sub-alignment between two sequences*. This is a powerful difference, since Smith–Waterman was designed to return only the single, best scoring sub-alignment.

This paper describes an extended Smith–Waterman algorithm using Associative Massive Parallelism known as SWAMP+. The main issues addressed through parallelization are a reduced run time and an augmented algorithm that returns multiple alignments. The parallel architectures used for SWAMP+, including the Associative Computing (ASC) model, ClearSpeed CSX600 and the Convey Computer, allow for the efficient discovery of multiple non-overlapping, non-intersecting sub-alignments. This creates a better workflow by automating portions of the existing pairwise alignment process to avoid manual intervention. These BLAST-like results reflect a format that bioinformatics users are familiar with. This extension is ideal for a deeper investigation, including drug design, biofuel refinement, sequence reassembly validation against the "gold standard" of alignments, and possible motif discovery.

SWAMP+ is unlike many of the previous parallel implementations, including GPU versions, since it is a full implementation of the Smith–Waterman algorithm extended to return multiple, subsequence results. It does not just focus on the neatly parallelizable first phase of the improved algorithm [8] using high-throughput hardware to filter and discard unlikely (poorly scoring) alignments; instead it utilizes parallelism to quickly find sub-alignments between a pair of sequences, actually returning the multiple sub-alignments instead of doling them out to a CPU to re-run the entire algorithm. This is a powerful difference from other implementations, producing multiple sub-alignments. SWAMP+ runs in $O(n)$ time on $O(m)$ processing elements (PEs), where n and m are the sequence lengths. Additional alignments are found with the same reduced run time.

Section 2 introduces the Smith–Waterman sequence alignment algorithm. Additional discussion of the Smith–Waterman parallelization efforts are presented in Section 3. Section 4 reviews the ASC single instruction, multiple data (SIMD) model [11] for which SWAMP was initially developed [12,13]. The specific details of the ASC SWAMP+ implementation and results are discussed in Section 5, with results presented in Section 6.

A second implementation of SWAMP+ for the ClearSpeed CSX-620 parallel co-processor is presented in Section 7. The results show a perfect linear speedup for up to 96 processing elements (PEs) on a single ClearSpeed chip, with a highly scalable implementation for additional PEs.

The final sections discuss the Convey Computer FPGA coprocessor, introducing adaptations to allow the extended sub-alignment results to be executed on a non-associative, not strictly SIMD system. The paper concludes with a summary section.

2. Smith–Waterman sequence alignment

The Smith–Waterman algorithm with Gotoh optimizations [8] is used to identify the similarity between two sequences, S1 and S2, by computing the best local alignment score. The sizes of these strings are m and n, respectively.

The dynamic programming approach computes a matrix to preserve values and avoid re-computation. This creates data dependencies between the matrix cells. A matrix entry cannot be computed without prior computation of its north (N), west (W) and northwestern (NW) neighbors, as shown in Fig. 1 and Eqs. (1)–(4).

Within each matrix cell, values are computed for every combination of insertions (I), deletions (D), and matches (C) [8]. Insertions and deletions are also referred to as *indels*, since their naming is based on the relative relationship between the query and reference sequences. Affine gap penalties for indels make the alignments more biologically relevant. The affine gap penalties are the cost to open up a new gap (σ) and the cost to extend an already opened gap (g). A gap indicates a mismatch between the sequences, i.e. an *indel*. Eq. (1) computes the deletion value with affine gap penalties of the current matrix cell ($D_{i,j}$) using the north neighbor's values.

$$D_{i,j} = \max\left\{ \begin{matrix} C_{i-1,j} - \sigma \\ D_{i-1,j} \end{matrix} \right\} - g \tag{1}$$

Insertion is similar, using the western neighbor's match value (C) and an existing open gap (I), subtracting the cost to extend a gap, where the goal is to maximize the score for the best match.

$$I_{i,j} = \max\left\{ \begin{matrix} C_{i,j-1} - \sigma \\ I_{i,j-1} \end{matrix} \right\} - g \tag{2}$$

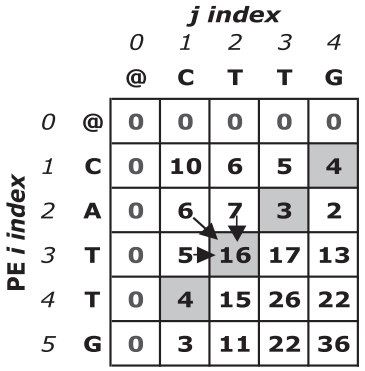

Fig. 1. An example of the sequential Smith–Waterman matrix. The N, W and NW dependencies of cell (3, 2) are shown with arrows. The final calculated C values for the entire matrix are shown here. The shaded anti-diagonal highlights a wavefront or logical parallel "step" since those cells can be computed in parallel. Affine gap penalties are used in this example as well as in the parallel code that produces the top alignment and other top scoring alignments. The alignment produced is CATTG to C-TTG, where the "-" represent an indel.

To compute a match where a base from both sequences is aligned, the actual base pairs are compared via Eq. (3). This is a simple match/mismatch scoring system often used with DNA and RNA. When amino acids are the target for alignments, there are scoring tables such as BLOSUM that can be substituted here.

$$\mathrm{d}(S1_i, S2_j) = \begin{cases} match\_cost & \text{if } S1_i = S2_j \\ miss\_cost & \text{if } S1_i \neq S2_j \end{cases} \tag{3}$$

The value from Eq. (3) is combined with the overall score from the corner northwest neighbor ($C_{i-1,j-1}$) before the maximum value of zero, deletion (Eq. (1)), insertion (Eq. (2)) and a match (Eq. (3)) produce the final cell score in Eq. (4).

$$C_{i,j} = \max \begin{Bmatrix} D_{i,j} \\ I_{i,j} \\ C_{i-1,j-1} + \mathrm{d}(S1_i, S2_j) \\ 0 \end{Bmatrix} \tag{4}$$

3. Parallel Smith–Waterman implementations

Previous efforts to develop techniques for Smith–Waterman on high performance architectures often include some form of vectorization. The four basic approaches to vectorization of Smith–Waterman alignment have been classified by Rognes [14] as:

A) Vectors along the anti-diagonal (a wavefront) approach, described by Woznizak [15]
B) Vectors along the query (a single column split downward), described by Rognes and Seeberg [16]
C) A striped approach, introduced by Farrar [17]
D) Multi-sequence vectors, described by Alpern et al. [18] and again in Rognes [14]

Fig. 2 contains graphical representations of these four approaches.

SWAMP+ utilizes an anti-diagonal or wavefront vectorization approach similar to the first vectorization approach characterized by [14].

Other parallelization approaches have been designed for systems including: the X86 SIMD extensions by performing the calculations in parallel by dividing wide registers into several units [17,18], the Sony–Toshiba–IBM Cell Broadband Engine [16,17,19–23], graphics card processing units (GPUs) [24–27], and field-programmable gate arrays (FPGAs) [28–30]. MPsrch [31] is designed for large-scale SIMD MasPar family of computers and the Connection Machine. There have been many clever optimizations, from striping computations with a given stride and "lazy-F loop" evaluation by Farrar [17,21], to further architecture-specific optimization by Rudnicki et al. [23] for the Cell/BE coprocessors.

SWAMP+ was originally designed for the Associative Computing (ASC) model and the ClearSpeed CSX600. Utilizing architecture-specific optimizations as the GPU and Cell Smith–Waterman versions do, the SWAMP+ approach differs in that approach maintains the full sensitivity of Smith–Waterman without heuristics while returning the top alignment, in addition to a richer set of non-overlapping, non-intersecting sub-alignments.

The majority of the cited body of parallel work focus on obtaining a high cell updates per second (CUPS) rate—a throughput metric. High throughput is important when scanning the large and fast growing sequence databases. The exponential growth of database size is thanks to the high-throughput, next-generation sequencing platforms generating orders of magnitude larger data sets than their predecessors. As their price drops and they become more accessible for both research and personalized medicine, data production will only continue to grow.

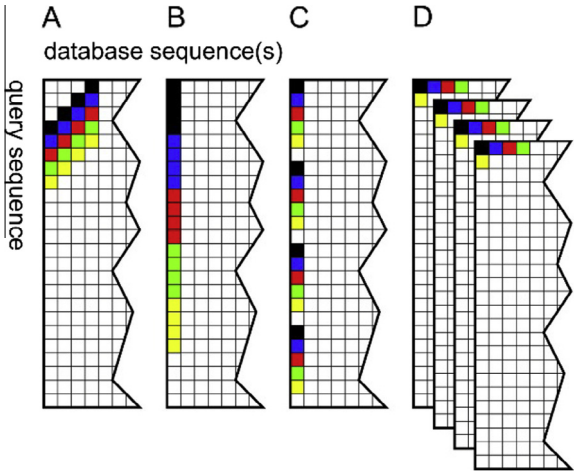

Fig. 2. Approaches to vectorization of Smith–Waterman alignments (from [14]). Alignment matrices are shown with the elements that form the first five vectors processed indicated in black, blue, red, green and yellow. For simplicity, vectors of only 4 elements are shown, while 16 elements would normally be used. (A) Vectors along the anti-diagonal, described by [15]. (B) Vectors along the query, described by [16]. (C) Striped approach, described by [17]. (D) Multi-sequence vectors, described by [18] and in [14]. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article [14].)

In order to achieve high throughput, most of the parallel algorithms cited above do not complete the traceback portion of the algorithm [15–17,22,24,29,21]. Instead, they return only the maximum score computed by the Smith–Waterman technique, stopping the algorithm early. When the computed score is above a certain threshold, that sequence or set of sequences with high scores are marked and a full (often sequential) Smith–Waterman alignment is re-run, to return the single highest scoring alignment of the query sequence(s).

The sequential Smith–Waterman alignment algorithm has been parallelized using a wavefront vectorization method with architecture-specific optimizations for the ASC and ClearSpeed models, called SWAMP. SWAMP+ is the extended version of SWAMP that has been augmented to return the top scoring alignment and the additional, top scoring sub-alignments above a given threshold.

The following sections discuss the ASC architecture, the ASC SWAMP+ implementation and results. This is followed by the SWAMP+ for the ClearSpeed architecture, and concludes with the latest work extending SWAMP+ to the Convey Computer platform.

4. The Associative Computing model

Initial development was on the *AS*sociative *C*omputing (ASC) platform or model [11]. Chosen for SWAMP [13] and SWAMP+ [32], the ASC model is ideal for algorithm development with less overhead, less complication for communication patterns, and parallel variable masking. SWAMP+ and other ASC algorithms have been adapted to run efficiently on several other SIMD-related hardware platforms [33,34] including the ClearSpeed CSX620 accelerator board. In addition, ASC algorithms can be efficiently translated for other systems closely related to SIMDs, including most vector machines.

ASC is based on the STARAN associative SIMD computer and its heavily Navy-utilized successor the ASPRO, designed by Dr. Kenneth Batcher at Goodyear Aerospace.

The word "associative" relates to the use of searching to locate data by content rather than memory addresses. The ASC model does not employ associative memory—instead it is an associative *processor* where the general cycle is *search* → *process* → *retrieve* [11].

As a pure SIMD model it uses synchronous data-parallel programming, avoiding both multi-tasking and asynchronous point-to-point communication routing. As seen in Fig. 3, every processing element (PE) has access to data located in its own local memory. The data remain in place, and any qualified, responding PEs are activated to process their own local data in parallel with other active PEs. PEs are generally activated through a conditional search, i.e. if the parallel variable *S*2 is equal to true (`if S2[$] = TRUE`). Parallel variables, known as `pvar`, `poly` or `multi` in other parallel languages are indicated with the "`$`" symbol in the ASC programming language syntax.

The tabular nature of ASC lends itself to the Smith–Waterman algorithm due to the natural table structure of the dynamic programming matrix, as first seen in Fig. 1. Fig. 4 is an example of the shifted vectorization for the Smith–Waterman algorithm as modified for SWAMP+. The ASC version is tabular but distinctly different from the original. Section 5 discusses in further detail the rationale for the shifting.

ASC operations are applied in parallel to all of the active PE's local data as directed by the programming code. One operation used by SWAMP+ is the associative *MAXDEX* operator is used to find the maximum score. The ASC-specific *MAXDEX*

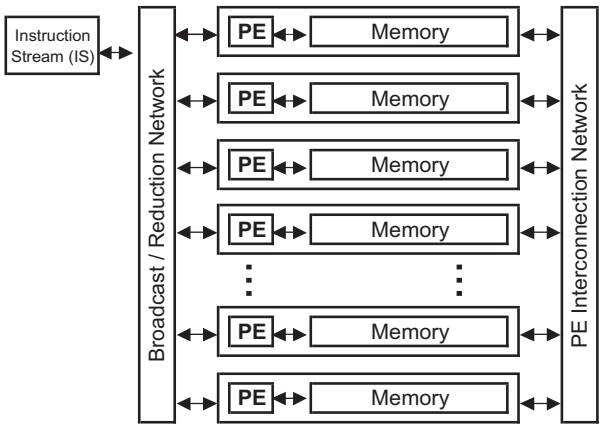

Fig. 3. A high-level view of the extended SIMD Associative Computing (ASC) model of parallel computation. The Instruction Stream (IS) relays instructions over the Broadcast/Reduction Network. Every active Processing Element (PE) will execute using the synchronous clock cycles. Instructions are executed on the local data stored in the memory associated with a particular PE. Data can be transferred between PEs over the PE Interconnection Network.

operator is a very fast maximum across the PEs that returns the index corresponding to the PE containing the largest value in a given single parallel variable (a logical column, such as `S2[$,5]` in Fig. 4). The associative operations are executed in constant time relative to the architecture's word size, which will not change for a given set of hardware. This efficiency is due to the additional hardware that is a fundamental aspect of the ASC model.

While many different parallel versions of Smith–Waterman exist, including for the massively parallel Connection Machine and the MasPar family of computers [31], the authors are not aware of any variations and extensions similar to SWAMP+.

5. SWAMP+: Smith–Waterman using Associative Massive Parallelism plus additional alignments

The SWAMP+ algorithm was originally developed for the ASC model. It is the basis of the additional work that was extended to the ClearSpeed and Convey Computer platforms. This section discusses the algorithm design and performance on the ASC architecture.

Whereas SWAMP [13] provided the precision of the Smith–Waterman algorithm in parallel, SWAMP+ has the benefit of returning multiple, BLAST-like pairwise sub-alignments without the disadvantages of a heuristic algorithm. SWAMP+ computes and finds the best local subsequence between the two strings; subsequent mining of the remaining sequence characters for additional local alignments use ASC architectural optimizations of data masking and data movement.

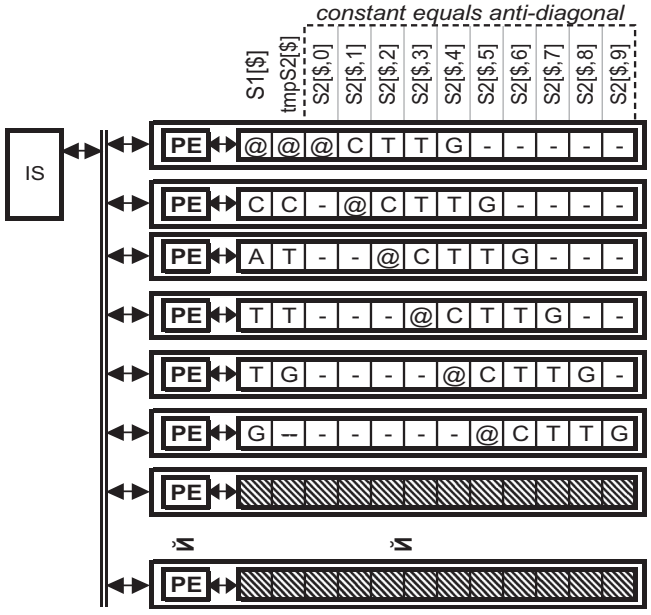

Fig. 4. Mapping the data onto ASC with S2 "shifted" roughly 45° counter-clockwise across the S2[$] parallel array. Every S2[$] column stores one full anti-diagonal from the original sequential matrix. Each active PE holds a single character from S1 and an offset version of S2. The final alignment is CATTG and C-TTG with one indel.

5.1. Anti-diagonal wavefront approach

SWAMP+ uses the Wozniak [15] wavefront approach to parallelize the matrix computation. This falls under "Type A" in the Rognes classification [14], shown in Fig. 2.

Limitations of the wavefront vectorization are due to the data dependencies of the North, West and Northwest neighbors defined in Section 2, as well as the local memory constraints within each PE. To better overcome the local memory constraints, the shorter of the two sequences is stored in the PEs local memory for the wavefront approach. The longer sequence is scattered across the PEs S1[$] in (Fig. 4). Once the sequences are scattered and shifted, each S2[$] parallel variable (array column in Fig. 4) contains a full anti-diagonal from the original Smith–Waterman algorithm. This allows each anti-diagonal to be computed in SIMD parallel lock-step. Regardless of data dependency limitations, this approach does result in a significant speedup by computing the data-independent matrix entries along the anti-diagonal in parallel.

5.2. ASC SWAMP+ program overview

SWAMP+ was programmed using the ASC language. The language shares the name of its corresponding model. It contains scalar (traditional, non-parallel) variables as well as parallel variables. Italicized in the pseudocode examples, the "$" symbol indicates computation or processing on the *active* PEs' values for that parallel variable concurrently. The active PEs may contain all or a subset of the PEs and their local data for parallel variables.

Bold text in the Listing 1 inset indicates the extension from SWAMP to SWAMP+. Starting at Line 10, the repeatable traceback for sub-alignments with the corresponding user output is introduced. A detailed description of the parallelized compute-intensive section is available in [13], including the architecture-optimized initialization (scattering) in Step 2 and shown in Fig. 4.

The first major change to Smith–Waterman for SWAMP+ is during traceback, Lines 12.1 and 12.2. Any time two residues are aligned, those characters in the first string S1 (`S1[row_id]`) and the specific diagonal of the second string S2 (`S2[diag - PEid]`) are masked as belonging to the traceback. The index manipulation for S2 is due to SWAMP+'s reorganization of data, turning S2 horizontally and shifting it into active PEs.

Listing 1: SWAMP+ Local Alignment Algorithm with traceback

```
1.  Read in S1 and S2
In Active PEs (those with data values for S1 or S2):
2.  Initialize the two-dimension parallel array variables D[$], I[$], C[$] to 0.
3.  Shift string S2
4.  For every diag from 2 to m + n − 1 do in parallel {
5.    If S2[$,diag] is valid data then {
6.1.      Calculate score for deletion for D[$,diag] (Eq. (1))
6.2.      Calculate score for a insertion for I[$,diag] (Eq. (2))
6.3.      Calculate matrix score for C[$,diag]} (Eq. (3))
7.        local_maxPE=MAXDEX(C[$, diag]s) (Eq. (4))
8.    if C[local_maxPE, diag] > max_Val then {
9.1     max_PE = local_maxPE
9.2     max_Val = C[local_maxPE, diag])}}
10.  Starting at max_Val, max_PE // get row and col indices
10.1.     diag = max_col_id
10.2.     row_id = max_id
11.  Output the very last two characters that are aligned
12.  While (C[$,diag] >0) and traceback_direction!= ''x''{
12.1.     S1_in_tB[row_id] = TRUE
12.2.     S2_in_tB[diag - PEid] = TRUE
13.1.     if traceback_direction == ''c'' {//Northwest diag = diag − 2; row_id = row_id -1}
13.2.     if traceback_direction == ''n'' {//North diag = diag − 1; row_id = row_id -1}
13.3.     if traceback_direction == ''w'' {// West diag = diag − 1; row_id = row_id}
14.       Output C[row_id, diag], S1[row_id], and S2[row_id, diag]}
15.     if S1_in_tB[$] = TRUE then {S1[$] = ''Z''}
16.     if S2_in_tB[$] = TRUE then {S2[$] = ''O''}
17.  Go to Step 2 while # of iterations < k or maxVal < threshold (d* overall_maxVal)
```

Any residues (characters) involved in an alignment, including insertions and deletions (indels), are marked so that they will not be considered in later alignments. S1's aligned residues are changed in Line 15 to the character "Z" which does not code for any DNA or an amino acid. A similar step is taken in Line 16 to disable the region that has already been matched in S2, using the character "O," since that does not encode for an amino acid. The characters "X" and "N" have been avoided since they are often used as a "Don't Know" character in genomic data thereby avoiding incorrect or chimeric sub-alignments.

The second through kth iterations of SWAMP+ now contain "do not match to" characters in S1 and S2, where k is the desired number of sub-alignments. While S1 is directly altered in place (Line 15), S2 is more problematic because every PE holds a slightly shifted copy of S2. 'The most efficient way to handle the changes to S2 is to alter the single S2 string and reinitialize the parallel array S2[$,0] through S2[$,$m + n$]. Discussed in detail in [13], the efficient initialization utilizes the linear PE interconnection network available for communication between the PEs in ASC. The temporary parallel variable named *shiftS2*[$] is where that shifted array is stored. This basic re-initialization of the S2[$,x] array in Step 2 is run for every kth sub-alignment. Re-initializing ensures local memory consistency across PEs, similar to cache coherency.

The number of additional sub-alignments is limited by two parameters: k, the number of local alignments sought and δ, a maximum degradation factor. The parameter k can be set arbitrarily high when a reasonable quality factor is set by δ. When $\delta = 0.5$, the repeat sub-alignment loop is stopped when the new sub-alignment score is $\leqslant$50% of the initial (highest) alignment score.

The design of the ASC SWAMP+ allows additional regions of interest to be discovered without additional data movement between the parallel platform and the scalar platform, reducing parallel I/O, one of the main bottlenecks when using an off-chip co-processor.

5.3. Analysis of the associative SWAMP+ algorithm

A single full run of the SWAMP algorithm has $O(m + n)$ steps using $m + 1$ PEs, where m and n are the size of the compared sequences, S1 and S2 for a single traceback. The worst case for the first, longest alignment is when $m + n$ characters are aligned. That would consist of the width of the computed matrix, $O(m + n)$. This is asymptotically no larger than the computation portion and therefore only adds to the coefficient, maintaining an $O(m + n)$ running time.

SWAMP+ run time is equal to the amount of time for a single run multiplied by the number of desired sub-alignments for $k^*O(m + n)$. The size of the coefficient k is limited; k can be no larger than the minimum(m,n) because there cannot be more local alignments than the number of residues.

This worst case would only occur if every alignment is a single base long, where every other base being a match with an indel. This worst-case would result in an $n^*(m + n)$ run time where $m = n$ to produce an $O(n^2)$ algorithm. This worst case is unlikely, even difficult to produce on purpose. The designed use for sequence alignment means that the sequences are homologous, retaining evolutionary similarities, and comparisons are relevant. It would be undesirable for the expected biological models if this is not the case, and sequence alignments would be worthless. Additionally, the use of the degradation parameter δ with affine gap penalties make it highly unlikely that the worst case would occur since the local alignments of homologous sequences will be greater than a length of one, violating the worst case characterization.

6. ClearSpeed and ASC

The ASC model is a natural fit for the Smith–Waterman algorithm. The ASC architecture has been realized in hardware at an academic level, but has not been commercialized. In order to have a deployable system for SWAMP+ to run on, the commercial off the shelf (COTS) ClearSpeed SIMD CSX 620 PCI-X accelerator board is used as a surrogate platform. Consisting of 96 PEs per chip, with two chips per board for 192 PEs overall, the board uses the MTAP architecture [35] (Fig. 5).

Used for programming on the ClearSpeed platform, the C^n language is similar to standard C, with the addition of parallel `poly` variables that contain a local copy of that variable in all of the active processing elements (PEs).

The ClearSpeed CSX620 was chosen because it is a SIMD-like board. As a SIMD, it lacks the associative functions, namely the associative maximum and associative search utilized by SWAMP and SWAMP+. Whereas ASC natively supports those operations via the broadcast and reduction network in constant time [36], the Associative functionality is handled at the software level on ClearSpeed. These needed functions have been written and optimized for speed and efficiency in the ClearSpeed assembly language.

The SWAMP+ algorithm requires a linear array to communicate with the northern neighboring PE for the north and northwest values that were previously computed. The ClearSpeed MTAP architecture does have a linear network between PEs with wrap-around, making it well suited for the application. The communication between neighboring PE's local memories across the full chip makes it ideal for this application, without complex communication workarounds necessary in a GPU. In addition, the MTAP architecture is much more similar to the ASC architecture, for which the original algorithm was designed for. Many GPU versions of the "Smith–Waterman Algorithm" strictly implement the parallelizable matrix computation, returning the maximum score or an index instead of the full traceback [24–27]. SWAMP+ returns the full traceback as its first result with additional multiple alignments as previously specified.

Using the C^n language, SWAMP+ was implemented for the ClearSpeed accelerator. There are essentially two parts to the code: the parallel computation of the matrix and the sequential traceback. The analysis breaks these two sections down, first

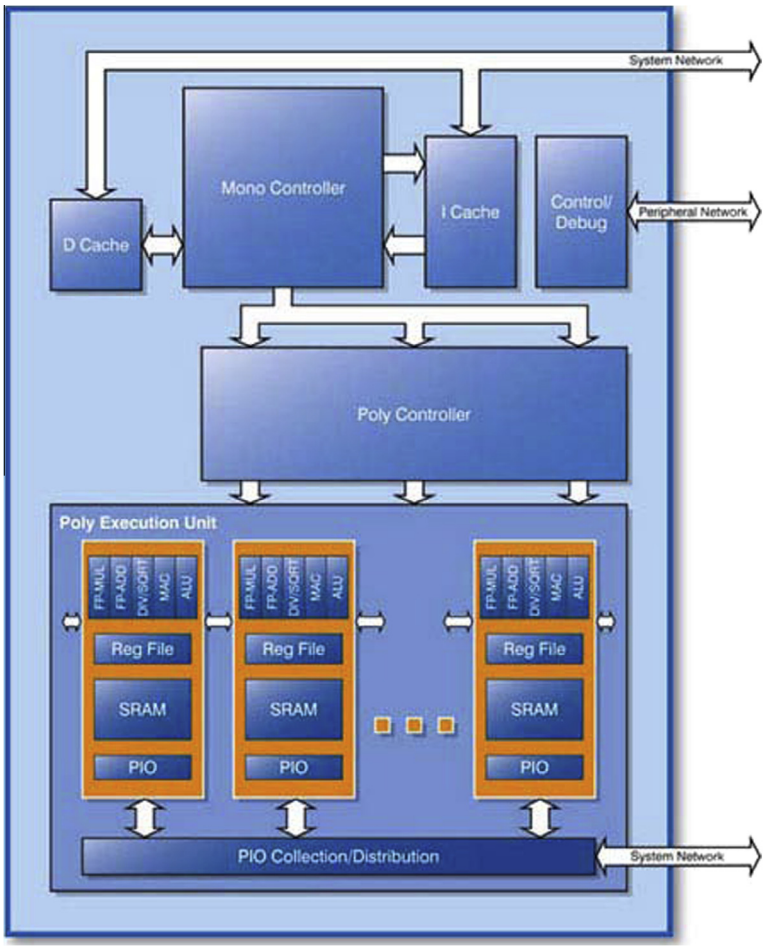

Fig. 5. ClearSpeed CSX MTAP processor organization. Diagram courtesy of ClearSpeed. http://www.clearspeed.com/products/csx700.

looking at the parallel matrix computation. The matrix computation performance analysis is often the only analysis provided for many of the parallel Smith–Waterman implementations. The second half of the ClearSpeed analysis is for the sequential traceback, including performing the extended non-overlapping, non-intersecting multiple alignment performance. This traceback information is novel for most parallel versions of Smith–Waterman as mentioned above, in addition to the additional multiple sub-alignments returned.

6.1. ClearSpeed implementation

The wavefront approach listed in the ASC pseudocode is similar to the implementation discussed here. The main difference is the initialization of the shifted second S2 sequence string. In C^n, an entire string is copied at a time instead of character by character. The computation of the three matrices for the north, west and northwestern values use the poly execution units on a single CSX chip. The logical "diagonals" are processed in a manner, similar to the ASC implementation, using a column-order compute pattern over the shifted S2 string.

The parameters used were suggested by [37] for nucleotide alignments. The affine gap penalties are -10 to open a gap, -2 to extend. A match is worth +5 and the mismatch between base characters is -4.

For performance metrics, the number of cycles were counted using the available *get_cycles*() function. Running at 250 MHz (250 million cycles per second), Fig. 6 shows the average number of cycles for computing the first through *k*th matrices. This parallel operation, whether 10 characters or 96 characters being compared at a time, have similar overall run time. This shows the linear speedup projected by the implementation on the ASC model.

Fig. 6 shows the normalized time differences for the run times on the ClearSpeed SWAMP+ for the computation section of the algorithm. Higher time differences in the graph are better, since the baseline difference is taken from the slowest run. In reality, the running time differences are so minor, with a maximum difference was only 0.005430317 ms, that they can be considered essentially the same.

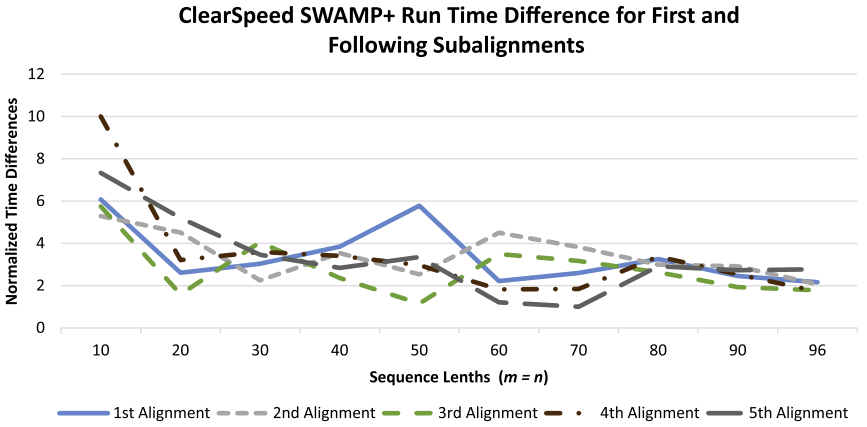

Fig. 6. Average normalized time difference for run times for the ClearSpeed SWAMP+ implementation. Each line represents 30+ runs for a different sub-alignment k value, normalized from the highest run time (0.2660762 ms), where higher is better. The run times are heavily dependent on the data. The difference between the highest and lowest run times is negligible (0.005430317 ms).

Fig. 7 captures the cell updates per second, CUPS, throughput metric, where higher a CUPS value is better. Since the time to compute the matrix for two sequences of length 10 or length 96 is roughly the same on the ClearSpeed board with 96 PEs, the CUPS measurement increases as the aligned sequence lengths reach their maximum of 96 characters each. The CUPS increases as the length of the sequences grows, since the execution time holds steady (see Fig. 6). The highest update rate is 36.13 million cell updates per second or 36.13 MCUPS when aligning two strings of 96 characters. The slight dip in the curve at the end is due to the fact that the sequence length went from 90 to 96, when all other sequence lengths grew by intervals of ten.

6.2. Traceback

For a full Smith–Waterman implementation, the second half of the C^n code must produce alignments, not just find the maximum score or terminal character. This traceback step is often overlooked or ignored by other parallel implementations. Our innovative approach uses the power of in-memory associative search to discover multiple, non-overlapping, non-intersecting subsequence alignments (with gaps allowed) efficiently. This is very useful for in-depth study, as well as to return a richer results set that many bioinformaticians are used to.

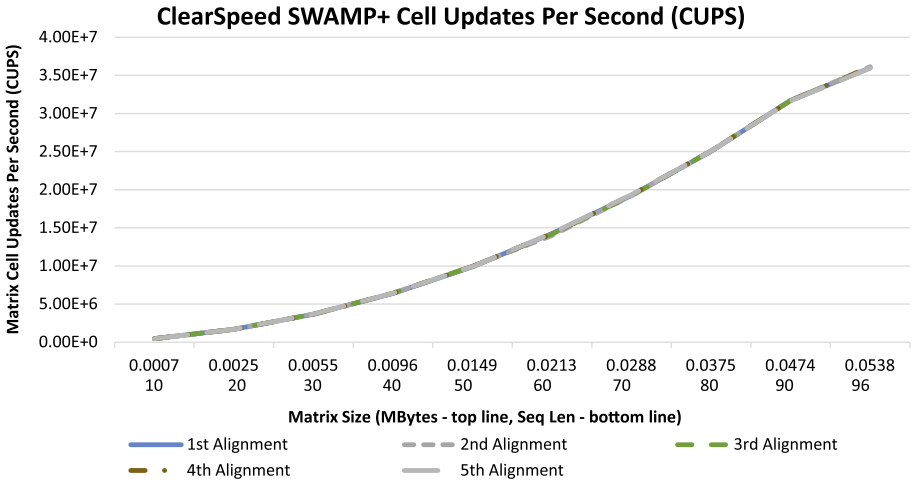

Fig. 7. CUPS value in relation to sequence length. The CUPS value rises as the sequence length increases. The larger the anti-diagonal, the more computations per second can occur with higher parallelism. In this SIMD-style implementation, the computation time was not dependent upon the sequence length (assuming the length $\leqslant$ #PEs available). The slight leveling off is due to the partial interval (sequence size increase by six instead of ten).

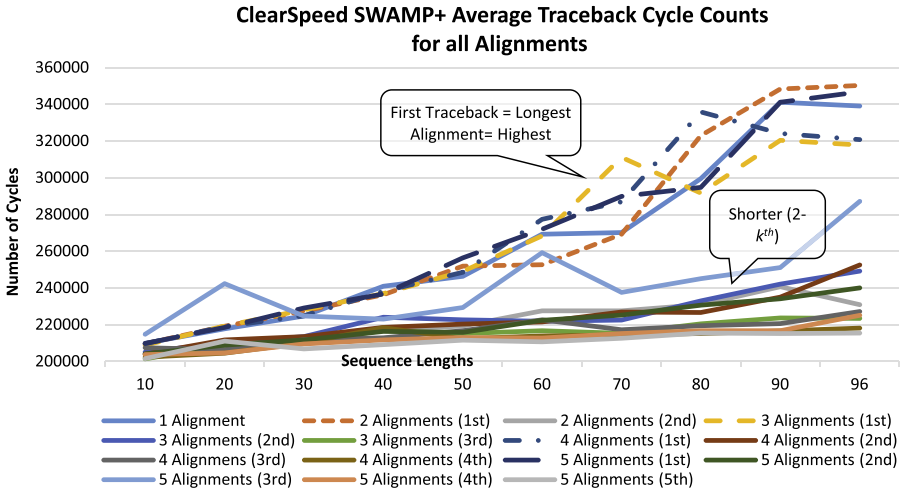

Fig. 8. The mean number of traceback cycles (averaged over 30+ runs). The longest alignment is the first alignment (traditional Smith–Waterman), therefore the first traceback requires the highest amount of time. This is true regardless of how many sub-alignments are run.

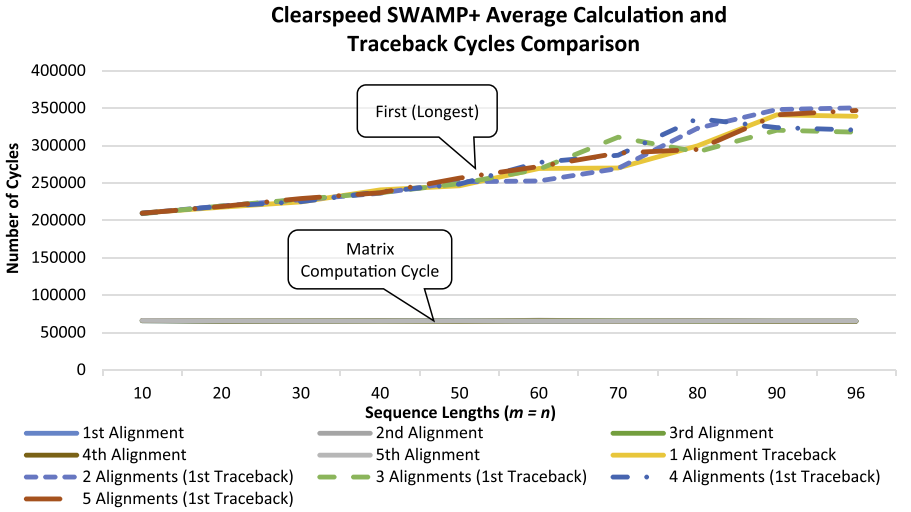

Fig. 9. Comparison of cycle counts for both the parallelized computation and the sequential traceback. The sequential traceback data for the first and longest alignment is taken directly from Fig. 8 to show against the first matrix computation section. The traceback increases with the sequence size, since this artificial data set ensures poor behavior. The parallelized matrix computations have a steady run time even as the length increases up to the full 96 PEs on the board. As long as the sequences fit on a chip, the matrix computation run time remains constant.

Similar to the ASC SWAMP+ implementation, the traceback is started at the maximum computed value in the C matrix. From there, the algorithm traces back to the beginning of the subsequence alignment, tracking indels, substitutions and matches. The amount of time this sequential process takes is proportionate to the match length. Fig. 8 demonstrates that the first alignment takes the largest amount of time. This is due to the initial alignment being the longest and best scoring alignment, given a set of parameters that discourage high gap content. The second through kth alignments are shorter, or they would have been discovered first, and therefore take less time to trace. The trend shows that the overall time of the alignments grows linearly with the size of the sequences themselves, matching the $O(m+n)$ performance predicted by the ASC implementation. The run times for the ClearSpeed implementation confirm the expected performance initially demonstrated in ASC.

The performance of both the sequential and parallel sections of code are combined in Fig. 9. The constant curve for the parallelized section of the code confirms the $O(m+n)$ optimal running time on m PEs. An alignment of ten sequences with 100 matrix elements took as long as two sequences of 96 with 9216 matrix elements. This shows perfect linear speedup in the parallel algorithm with a very low coefficient. Beyond 96 elements (characters), the scalability is limited by hardware

since additional PEs are not available. The second chip of PEs on the CSX620 can be utilized in tandem to run a separate SWAMP+ instance. The CSX 600 series limitation is that a single program can be run on a single chip. The communication between the two 96 PE chips on the CSX 620 is limited to from host to chip separately, and not chip-to-chip. Another option to expand capability while maintaining performance is to use the newer ClearSpeed CSX700 with 192 PEs.

The ClearSpeed system is close to ASC with a SIMD-style paradigm, making it ideal to transition from ASC to a COTS system. The linear speedup in performance is excellent for small strings. The drawback is that the system and the SWAMP+ algorithm are not tuned to handle large strings, resulting in higher running times (a larger coefficient for the $O(m+n)$ run time) when the strings are very large. To overcome these limitations and explore the feasibility of using a hybrid FPGA system, the Convey Computer HC-1 was chosen in order for SWAMP+ to be able to handle more realistic data sizes.

7. Convey Computer

The Convey hybrid-core server is similar to ClearSpeed in that it combines a host x86 processor and a coprocessor in a single physical hardware system. In the Convey Computer approach, the host processor and coprocessor share a cache coherent global memory address space, and a program can execute code on both processors concurrently [38].

The Convey coprocessor contains field-programmable gate array (FPGA) technology. An FPGA is a fabric of logic elements, each with a small amount of combinational logic and a register that can be used to implement everything from simple circuits to complete microprocessors. While generally slower than traditional microprocessors, FPGAs are able to exploit a high degree of fine-grained parallelism, and are, as the name suggests, reprogrammable on the fly.

With the reprogrammable coprocessor, different configurations or *personalities* are available. Personalities are designed at the hardware level to configure the FPGA to perform optimally for a given task. For instance, there are single and double precision vector personalities depending upon the needs of your task.

Convey Computer has designed a specific Smith–Waterman personality. The Smith–Waterman personality shares some similarities with [39], including that it uses a systolic array and vectorization based on the anti-diagonal wavefront.

The overall Convey Computer Smith–Waterman suite utilizes a combination of programming techniques that involve C code and some high level calls to functions defined in the specific Smith–Waterman personality. The suite is designed for fast high-throughput alignments over large databases containing sequences of small to medium length. Due to the length limitation, it is ideal for protein or amino acid database searches.

Convey Computer's Smith–Waterman implementation can be run strictly on the x86/64-bit CPU processor, just on the coprocessor, or on both scalar and parallel processors concurrently. This last combination results in very high throughput. Convey Computer's earlier generation HC-1 coprocessor achieved 259 billion cell updates per second (GCUPS) and their larger FPGA HC-1^{ex} system achieved 768 GCUPS [40], one of the fastest known GCUPS ratings at its release. This is several thousand times faster than a more traditional software implementation.

Given that the profile is highly optimized for the hardware, the challenge was to provide the extended Smith–Waterman SWAMP+ sub-alignments without associative functionality and without negatively impacting performance.

7.1. Running Smith–Waterman on a Convey Computer system

Running the Convey Computer Smith–Waterman on a configured system requires two FASTA formatted files, discussed below, one containing the query sequences and the other the database sequences against which the queries are to be aligned. The earlier ASC and ClearSpeed implementations use one multiFASTA file that contain two sequences for input, the query and database.

The FASTA file format is a text-based representation that contains at least two lines; the first is a header with information about the sequence, followed by one or more lines of the sequence data consisting of nucleotides, amino acids, or peptides. Multiple FASTA sequence entries are allowed in a single file, known as a multiFASTA file, a widely used format that the Convey Computer Smith–Waterman program accepts.

In the recently updated Convey Computer software there are two options (flags) for performing a full traceback on the scalar portion of the system: `traceback` and `tbcoords`. We use the latter flag, recording the coordinates, or indexes, of the match start and stop locations, in order to replace the characters that have been matched in an alignment with the do-not-match-to characters ("Z" or "O" depending upon whether it is part of the query or the reference sequence). To run the Convey Computer program, the following command is issued:`cnysws –tbcoords query.faa dblib.faa`

Convey's `cnysws` program computes scores for nucleotides with a DNA substitution matrix, and proteins using BLOSUM50 or BLOSUM62 scoring matrixes. These scoring matrices allow for models of evolution to be included, with different scores based on more complex relationships than a single match/mismatch scoring system.

7.2. Capturing the SWAMP+ sub-alignments

In order to create the multiple alignment SWAMP+ results, a combination of the Convey Computer Smith–Waterman suite, system shell scripts, and Python are used. The outline for this non-associative SWAMP+ version is shown in Listing 2.

Listing 2: Pseudocode for the SWAMP+ extension on the Non-Associative Parallel System (Convey)

1. Set up value for k and δ
2. Initial run of `cnysws –traceback query.faa dblib.faa5`
3. Capture accession numbers, start/stop indices, scores for every query sequence and related hit in the database sequence, append alignments in `bestMatches.txt`
4. For each *query sequence*:
   4.1. Seek the accession number in `query.faa`
   4.2. Capture the FASTA entry for that query sequence
   4.3. For each *hit sequence* to that query:
      4.3.1. Copy single FASTA query entry to a new file: `alteredQuery_`**`query#_hit#`**`.faa`
         4.3.1.1. Set the query characters that were aligned in Step 2 to 'Z'
      4.3.2. Seek accession number of the current hit sequence in `dblib.faa`
      4.3.3. Copy single FASTA entry to new file: `alteredDb_`**`query#_hit#`**`.faa` (bold values correspond to the same from 4.3.1)
         4.3.3.1. Set the database hit characters that were aligned in Step 2 to 'O'
5. While # iterations < $(k-1)$ and sequences exist to be aligned:
   5.1. For every file created in Step 4:
      5.1.1. Run (x86 only) `cnysws –traceback alteredQuery_`**`query#_hit#`**`.faa alteredDb_`**`query#_hit#`**`.faa`
      5.1.2. Repeat step 3
      5.1.3. If score for hit * δ < current score:
         5.1.3.1. Set the query characters that were aligned in this round of Step 2 to 'Z' in `alteredQuery` file
         5.1.3.2. Set the database hit characters that were aligned in Step 2 to 'O' in `alteredDb` file
6. Remove the created "`altered`"-named files
7. Output the k sub-alignments for each query

*Note: **Bold** Values are set dynamically*

Instruction 2 will run the first initial full Convey Computer Smith-Waterman suite with all queries in the query file aligned against all reference sequences in the database file. The results that contain the FASTA headers for each query and reference alignment, including the coordinates and run times, are output into the `bestMatches.txt` file. These are parsed in Steps 3–4.2.

Instruction 4 iterates through the initial results set. For each query and its associated reference alignment, a pair of files is generated (Listing 2, Step 4.3). A pair of files containing at least a single FASTA or FASTQ formatted query and database sequence is necessary to run Convey's `cnysws` program. These generated pairs of files are named: `alteredQuery_`**`query#_hit#`**`.faa and alteredDb_`**`query#_hit#`**`.faa`, where the **`_query#_hit#`** values are set to 1_1, 1_2, 2_1, etc. These files record the cumulative bases matched for each and every sub-alignment within the database up to iteration k. Remember that each sub-alignment "knocks out" characters that have already been aligned; query sequences use 'Z' characters and database sequence change to non-matching 'O' characters. The additional files are produced in pairs for every sub-alignment after the first. This is necessary to ensure alignments remain non-overlapping, non-intersecting, and that each alignment to the query retains the "history" of matched bases. This file generation is necessary since the "history" of the matched bases cannot be translated between runs of the Convey Computer Smith–Waterman algorithm, as is done within the ASC local PE memory and in the emulated associative environment on ClearSpeed. The `cnysws` program is re-instantiated for each new file pair in Step 5.1.1. Sub-alignments are tracked for each input query (5.1.2–5.1.3).

7.3. Running SWAMP+ on the Convey Computer

The HC-1 coprocessor is used with a quad core Intel Xeon 2.13 GHz host system with 181 GB of RAM. The Convey Computer `cnysws` program is unmodified by the author. Several Python and bash scripts were executed following the outline given in Listing 2.

For the data runs completed on the Convey Computer system, the same parameters suggested by [37] for nucleotide alignments were used. The affine gap penalties are -10 to open a gap, -2 to extend, as they were for the ClearSpeed runs. The data used in the ClearSpeed runs were re-used on the Convey system. In addition, significantly larger data sets including sequences of length 100, 1000, and 10000 nucleotide bases long were added for benchmarking performance. The test data files follow the same constraints as those used on the ClearSpeed implementation. The aligned sequences are of the same size with differing data. The sizes of sequences are shown along the y-axis in Fig. 10. In addition, the example amino acid (AA) run used the Convey Computer input example.

The amino acid (AA) data consisted of two queries in the query file and 87 database reference sequences that have an average size of 528 amino acids, but includes larger sequences upwards of 750+ characters. The amino acid files used the

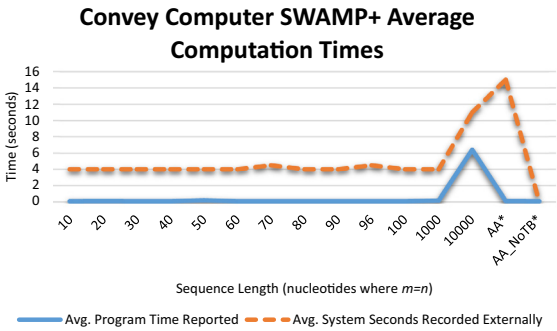

Fig. 10. Average computation times for the Convey Computer SWAMP+ runs. The sequence lengths along the *y*-axis indicate the number of nucleotides in the two sequences being aligned. The asterisk after the label "AA" and "AA_noTB" indicate that the input files differ. The AA files consist of amino acids instead of nucleotides and are multiFASTA files. There are two query sequences and 87 database reference sequences the queries are aligned to. The AA_NoTB data point represents runs using the AA files with the traceback option turned off (NoTB). The low values demonstrate a system optimized for high-throughput, avoiding calculating the traceback portion unless specifically tasked. The system will run a traceback for the Smith–Waterman on the host processor and not on the parallel co-processor.

BLOSUM50 substitution matrix, an evolutionary relationship lookup table that scores more likely substitutions of one AA with another. The complexity of moving from nucleotides to amino acids can be significant when parallelizing, with lookup table read access bottlenecks, or local memory limitations if the table is copied into local memory, a common problem in most genetic assembly and alignment algorithms.

As is the case in the other two implementations of SWAMP+, the first alignment is the most time consuming alignment, since it will be the longest alignment that can be found using the dynamic programming approach. With the multiFASTA file format allowed the volume of alignments is much greater as all queries are aligned against all database entries the first time that `cnysws` is run (Step 2). For the second through *k*th alignments the pairs of files for each query-to-reference alignment are generated, where every file contains a single sequence, either the query or the database reference. The quality factor δ was seen when the number of runs was limited for the 50-nucleotide-base-long alignments, where two alignments out of the four possible requested completed.

Like many of the other parallel implementations discussed in Section 3, the Convey Computer implementation only finds *if* there is a quality match between a sequence and a reference, then records it. The HC-1 co-processor has a benchmarked peak of 259 GCUPS (billion cell updates per second). Any traceback calculation is an additional computation that is computed off of the parallel co-processor on the x86/64-bit host system. Traceback must be specifically requested using the "`tbcoord`" or the "`traceback`" flags. With the higher file input and output traffic, one optimization for this platform is to run the small alignments solely on the x86/64-bit host system. The Convey code allows for this, and utilizes the highly efficient SSE optimizations developed by Farrar [17].

To demonstrate the difference in runtime with and without traceback, look at the AA and the AA_NoTB data points in Fig. 10. The same data are aligned, but the AA_NoTB does not perform the traceback, keeping the computations solely on the co-processor to yield 71.192 GCUPS on the relatively small multiFASTA dataset. Longer datasets with thousands of short to medium length sequences can achieve a much larger GCUPS value, closer to the published peak 259 GCUPS on the HC-1.

This Convey Computer SWAMP+ implementation is more complex than its ASC and ClearSpeed counterparts, including the integrated scoring matrices and multiFASTA sequence files. The Smith–Waterman personality and supporting code reflect the current state of the art for Smith–Waterman with regard to both the x86/64-bit and custom co-processor optimizations. It is the most robust and commercially available system to use for SWAMP+.

8. Summary

This work introduces the extended Smith–Waterman using Associative Massive Parallelism or SWAMP+. It demonstrates that SWAMP+ can be efficiently implemented on two separate parallel SIMD and SIMD-like architectures. In addition, a third parallel system was used with the Convey Computer hybrid-core system. These implementations each produce BLAST-like multiple sub-alignments with the full, non-heuristic precision of the Smith–Waterman algorithm.

This richer set of non-overlapping, non-intersecting BLAST-like sub-alignments does not require parameter changes or data file altering after each sub-alignment; in fact it requires no user intervention at all for additional sub-alignment discovery.

The initial design focused on the ASC architecture, with thousands to hundreds of thousands of processing elements in mind, making associative algorithms highly scalable as the hardware scales. Acceleration via parallelization on ASC gained speed with ASC's tabular organization, fast maximums, as well as the easy ability to mask already aligned base pairs to be excluded from subsequent alignments.

As a means to implement the algorithm on commercially available hardware, the ClearSpeed accelerator was introduced. The ClearSpeed chip with its 96 PEs proved to be optimal, providing linear speedup over a sequential implementation, and achieved a running time of $O(m + n)$. When $m = n$, the running time is $O(n)$ with a coefficient of two, i.e. a running time of $2n$, a far reduction from the $O(n^2)$ quadratic running time.

In addition, SWAMP+ on ASC and ClearSpeed increases the compute to input/output (I/O) time ratio, making more efficient use of the compute capabilities of the ClearSpeed coprocessor. As systems increase to massively parallel scales, it has been noted that the I/O operations per second (IOPS) can have a larger performance impact than the achievable Floating Point Operations per second (FLOPS).

The SWAMP+ concepts were also extended to run on the hybrid-core FPGA Convey Computer system. The high-throughput system was able to produce sub-alignments with the repeated application of the Convey Computer Smith–Waterman algorithm. The results demonstrate that a system optimized for high-throughput can be used in conjunction with repeated traceback to produce BLAST-like, multiple sub-alignments efficiently.

Future work includes further investigation of the flags that control algorithm use such as the Streaming SIMD Extensions (SSE) options and the parallel co-processor on the Convey Computer `cnysws` program in order to reach an optimal SWAMP+ execution between co-processor and host. In addition, there are plans to investigate how SWAMP+ can be expanded to explore massive datasets in other ways, including work to discover expressed sequence tags (ESTs), regulatory regions, and evolutionary rearrangements.

Overall, two weaknesses of Smith–Waterman were addressed: the time complexity through parallelism, and the limited single alignment result. Given that most high performing parallel versions of Smith–Waterman do not produce a single alignment, let alone multiple subsequences with full traceback, SWAMP+ is a marked improvement. Of the three implementations, the Convey SWAMP+ implementation is the most robust, scalable system that meets the large data demands of many users. This work presents a fine-grained parallel approach that provides a deeper look into the evolutionary relationships between sequences, outstripping heuristic methods through the use of what many consider the "gold-standard" of the Smith–Waterman approach.

9. Publication release information

SWAMP+ is part of the U.S. Patent Application No. 13/423,085 for computer-facilitated parallel information alignment and analysis filed on March 16, 2012.

This work is released under the Los Alamos Unlimited Release publication number LA-UR-12-24883.

Los Alamos National Laboratory, an affirmative action/equal opportunity employer, is operated by Los Alamos National Security, LLC, for the National Nuclear Security Administration of the U.S. Department of Energy under contract DE-AC52-06NA25396. Neither Los Alamos National Security, LLC, the U.S. Government nor any agency thereof, nor any of their employees make any warranty, express or implied, or assume any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represent that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise does not necessarily constitute or imply its endorsement, recommendation, or favoring by Los Alamos National Security, LLC, the U.S. Government, or any agency thereof. The views and opinions of authors expressed herein do not necessarily state or reflect those of Los Alamos Nat0069onal Security, LLC, the U.S. Government, or any agency thereof.

Acknowledgments

Thanks to Convey for access to their HP-1 system and for the helpful assistance from Mark Kelly and John Amelio.

References

[1] D.A. Benson, M. Cavanaugh, K. Clark, I. Karsch-Mizrachi, D.J. Lipman, J. Ostell, E.W. Sayers, GenBank, Nucleic Acids Research 41 (2013) D36–D42.
[2] The UniProt Consortium, Reorganizing the protein space at the Universal Protein Resource (UniProt), Nucleic Acids Research, 40 (2012) D71–D75.
[3] P. Flicek, I. Ahmed, M.R. Amode, D. Barrell, K. Beal, S. Brent, D. Carvalho-Silva, P. Clapham, G. Coates, S. Fairley, S. Fitzgerald, L. Gil, C. García-Girón, L. Gordon, T. Hourlier, S. Hunt, T. Juettemann, A.K. Kähäri, S. Keenan, M. Komorowska, E. Kulesha, I. Longden, T. Maurel, W.M. McLaren, M. Muffato, R. Nag, B. Overduin, M. Pignatelli, B. Pritchard, E. Pritchard, H.S. Riat, G.R.S. Ritchie, M. Ruffier, M. Schuster, D. Sheppard, D. Sobral, K. Taylor, A. Thormann, S. Trevanion, S. White, S.P. Wilder, B.L. Aken, E. Birney, F. Cunningham, I. Dunham, J. Harrow, J. Herrero, T.J.P. Hubbard, N. Johnson, R. Kinsella, A. Parker, G. Spudich, A. Yates, A. Zadissa, S.M.J. Searle, Ensembl 2013, Nucleic Acids Research 41 (2013) D48–D55.
[4] M.S. Waterman, M. Eggert, A new algorithm for best subsequence alignments with application to tRNA–rRNA comparisons, Journal of Molecular Biology 197 (1987) 723–728.
[5] S.F. Altschul, W. Gish, W. Miller, E.W. Myers, D.J. Lipman, Basic local alignment search tool, Journal of Molecular Biology 215 (1990) 403–410.
[6] S.F. Altschul, T.L. Madden, A.A. Schaffer, J. Zhang, Z. Zhang, W. Miller, D.J. Lipman, Gapped BLAST and PSI–BLAST: a new generation of protein database search programs, Nucleic Acids Research 25 (1997) 3389–3402.
[7] W.R. Pearson, Empirical statistical estimates for sequence similarity searches, Journal of Molecular Biology 276 (1998) 71–84.
[8] O. Gotoh, An improved algorithm for matching biological sequences, Journal of Molecular Biology 162 (1982) 705–708.
[9] W. Gish, WU-BLAST, 1996.
[10] Z. Zhang, S. Schwartz, L. Wagner, W. Miller, A greedy algorithm for aligning DNA sequences, Journal of Computational Biology 7 (2000) 203–214.
[11] J. Potter, J.W. Baker, S. Scott, A. Bansal, C. Leangsuksun, C. Asthagiri, ASC: an associative-computing paradigm, Computer 27 (1994) 19–25.

[12] S.I. Steinfadt, M. Scherger, J.W. Baker, A Local Sequence Alignment Algorithm Using an Associative Model of Parallel Computation, IASTED's Computational and Systems Biology (CASB 2006), Dallas, TX, 2006, pp. 38–43.
[13] S. Steinfadt, J.W. Baker, SWAMP: Smith–Waterman using associative massive parallelism, in: IEEE International Symposium on Parallel and Distributed Processing, 2008, pp. 1–8.
[14] T. Rognes, Faster Smith–Waterman database searches with inter-sequence SIMD parallelisation, BMC Bioinformatics 12 (2011) 221.
[15] A. Wozniak, Using video-oriented instructions to speed up sequence comparison, Computer Applications in the Biosciences (CABIOS) 13 (1997) 145–150.
[16] T. Rognes, E. Seeberg, Six-fold speed-up of Smith–Waterman sequence database searches using parallel processing on common microprocessors, Bioinformatics (Oxford, England) 16 (2000) 699–706.
[17] M. Farrar, Striped Smith–Waterman speeds database searches six times over other SIMD implementations, Bioinformatics (Oxford, England) 23 (2007) 156–161.
[18] B. Alpern, L. Carter, K.S. Gatlin, Microparallelism and high performance protein matching, in: ACM/IEEE Supercomputing Conference, San Diego, 1995.
[19] V. Sachdeva, M. Kistler, E. Speight, T.H. Tzeng, Exploring the viability of the cell broadband engine for bioinformatics applications, in: Proc. of the 6th IEEE International Workshop on High Performance Computational Biology, 2007.
[20] A.M. Aji, W.-c. Feng, F. Blagojevic, D.S. Nikolopoulos, Cell-SWat: modeling and scheduling wavefront computations on the cell broadband engine, in: Proceedings of the 5th Conference on Computing frontiers, ACM, Ischia, Italy, 2008, pp. 13–22.
[21] M. Farrar, Optimizing Smith–Waterman for the Cell Broadband Engine, Sequence Analysis (2008).
[22] A. Szalkowski, C. Ledergerber, P. Krahenbuhl, C. Dessimoz, SWPS3 – fast multi-threaded vectorized Smith–Waterman for IBM Cell/B.E. and x86/SSE2, BMC Research Notes 1 (2008) 107.
[23] W.R. Rudnicki, A. Jankowski, A. Modzelewski, A. Piotrowski, A. Zadrozzny, The new SIMD implementation of the Smith–Waterman algorithm on cell microprocessor, Fundamenta Informaticae 96 (2009) 181–194.
[24] S.A. Manavski, G. Valle, CUDA compatible GPU cards as efficient hardware accelerators for Smith–Waterman sequence alignment, BMC Bioinformatics 9 (Suppl. 2) (2008) S10.
[25] S. Jung, Parallelized pairwise sequence alignment using CUDA on multiple GPUs, BMC Bioinformatics (2009) A3.
[26] L. Ligowski, W. Rudnicki, An efficient implementation of Smith Waterman algorithm on GPU using CUDA, for massively parallel scanning of sequence databases, in: Proceedings of the 2009 IEEE International Symposium on Parallel Distributed Processing, IEEE Computer Society, 2009, pp. 1–8.
[27] Y. Liu, B. Schmidt, 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 3 (2010) 93. <http://www.biomedcentral.com/1756-0500/3/93>.
[28] T. Oliver, B. Schmidt, D.L. Maskell, Reconfigurable architectures for bio-sequence database scanning on FPGAs, IEEE Transactions on Circuits and Systems II (52) (2005) 851–855.
[29] I.T. Li, W. Shum, K. Truong, 160-fold acceleration of the Smith–Waterman algorithm using a field programmable gate array (FPGA), BMC Bioinformatics 8 (2007) 185.
[30] Z. Nawaz, M. Nadeem, H. van Someren, K. Bertels, A parallel FPGA design of the Smith–Waterman traceback, in: Proceedings of the International Conference on Field-Programmable Technology (FPT), Beijing, 2010, pp. 454–459.
[31] S.S. Sturrock, J.F. Collins, MPsrch V1.3 User Guide, Biocomputing Research Unit, University of Edinburgh, UK, 1993.
[32] S. Steinfadt, Computer-facilitated Parallel Information Alignment and Analysis, Los Alamos National Security, LLC, Los Alamos, NM, USA, 2012.
[33] M. Yuan, J. Baker, F. Drews, W. Meilander, An Efficient Implementation of Air Traffic Control using the ClearSpeed CSX620 System, Parallel and Distributed Computing Systems (PDCS), Cambridge, MA, 2009.
[34] J. Trahan, M. Jin, W. Chantamas, J. Baker, Relating the power of the multiple associative computing model (MASC) to that of reconfigurable bus-based models, Journal of Parallel and Distributed Computing (JPDC) 70 (2010) 458–466.
[35] Clearspeed, Clearspeed Technology CSX600 Runtime Software User Guide, September 2008.
[36] M. Jin, J. Baker, K. Batcher, Timings for associative operations on the MASC model, in: 15th International Parallel and Distributed Processing Symposium (IPDPS'01) Workshops, San Francisco, CA, 2001, pp. 193.
[37] FASTA Search Results Tutorial, 2007.
[38] Convey Programmers Guide, Richardson, TX, 2010, pp. 143.
[39] P. Zhang, G. Tan, G.R. Gao, Implementation of the Smith–Waterman algorithm on a reconfigurable supercomputing platform, in: HPRCTA '07: Proceedings of the 1st International Workshop on High-performance Reconfigurable Computing Technology And Applications, ACM, New York, NY, USA, 2007, pp. 39–48.
[40] G. Vacek, Convey Smith–Waterman personality slide, in: S. Steinfadt (Ed.), Personal Communication, 2012.