On the Impact of Widening Vector Registers on Sequence Alignment

Jeff Daily*, Ananth Kalyanaraman†, Sriram Krishnamoorthy* and Bin Ren*
*High Performance Computing Group, Pacific Northwest National Laboratory
†School of Electrical Engineering and Computer Science, Washington State University

***Abstract*—Vector extensions, such as SSE, have been part of the x86 since the 1990s, with applications in graphics, signal processing, and scientific applications. Although many algorithms and applications can naturally benefit from automatic vectorization techniques, there are still many that are difficult to vectorize due to their dependence on irregular data structures, dense branch operations, or data dependencies. Sequence alignment, one of the most widely used operations in bioinformatics workflows, has a computational footprint that features complex data dependencies. In this paper, we demonstrate that the trend of widening vector registers adversely affects the state-of-the-art sequence alignment algorithm based on striped data layouts. We present a practically efficient SIMD implementation of a parallel scan based sequence alignment algorithm that can better exploit wider SIMD units. We conduct comprehensive workload and use case analyses to characterize the relative behavior of the striped and scan approaches and identify the best choice of algorithm based on input length and SIMD width.**

I. Introduction

Vectorization is an effective way to improve the performance of many kinds of applications via replacing a batch of scalar instructions by vector (SIMD) instructions. In addition, with respect to power consumption, vectorization is considered *free* because it needs relatively little extra hardware support, like SIMD extensions.

Since the 1990s, when Streaming SIMD Extensions (SSE) were introduced as part of the x86, they have been widely used in many areas. Recent years have seen SIMD widths expand. Sandy Bridge doubled the SSE SIMD width from 128-bit to 256-bit with new intrinsics called AVX. Moreover, the latest Xeon Phi Coprocessor is equipped with a 512-bit Vector Processing Unit (VPU) with new intrinsics called AVX-512 that can process 16 floating point operations with the same type concurrently. Considering the upcoming Knights Landing CPU with an even more powerful SIMD instruction set, vectorization will provide us more benefits for many applications and algorithms.

There have been many efforts focusing on the vectorization area, especially for dense matrix algorithms. In recent years, many irregular applications have been mapped to various SIMD architectures. However, there are few works considering the effect of increasing SIMD widths on the design and implementation of existing SIMD algorithms. Starting from this viewpoint, we perform a careful study on multiple SIMD sequence alignment algorithms.

Sequence alignment is a fundamental operation in many bioinformatics data processing workflows, at times comprising much of the computational workload. This is due, in part, to the quadratic computational complexity of the dynamic programming algorithm. Aligning two sequences of lengths m and n requires $\mathcal{O}(mn)$ time. Accelerating sequence alignment is of paramount importance in a number of bioinformatics applications. To this end, multiple approaches have been developed using SIMD instructions, practically one for every new instruction set architecture (ISA) [35], [28], [27], [18], with Striped [6] representing the current state of the art.

There are three classes of sequence alignment algorithms, namely Needleman-Wunsch [23] (NW) global alignment, semi-global (SG), and Smith-Waterman [30] (SW) local alignment. SW is predominantly studied in literature, though NW and SG are also useful in their respective contexts. To date, vectorized implementations of NW and SG have not been studied despite their wide use. In fact, we find that each class of sequence alignment presents unique performance trends, especially in light of widening vector registers on future hardware. We also find that the Striped approach experiences diminished performance returns as SIMD widths increase in addition to underperforming when adapted for NW and SG.

This paper presents a new SIMD sequence alignment implementation based on a parallel prefix scan algorithm for the three classes of sequence alignment mentioned previously. For the latest SIMD widths currently supported, the Scan implementation outperforms Striped for many of the characteristic inputs we study. Our comprehensive analysis shows that the Scan approach is better able to utilize the available parallelism of increasing SIMD widths.

The key contributions are as follows:

- New SIMD implementation of a parallel scan-based sequence alignment algorithm.
- Comprehensive workload and use case characterization.
- Comparative analysis of Scan vs. Striped on some of the largest workloads.
- Study of algorithm behavior across vector widths and different classes of sequence alignment.
- Prescriptive solutions on the choice of algorithms given different classes of sequence alignment and across SIMD widths.

II. Sequence Alignment Algorithms

Sequence alignment is an order-preserving way to map characters between two DNA or amino-acid (protein) sequences. It is a pervasive operation in bioinformatics workflows used to identify regions of high similarity between sequences. Similarity is generally measured by assigning a positive score to matches and a negative score to mismatches. For proteins, a substitution matrix, such as BLOSUM [10] or PAM [31], is used to score amino acid similarity for each possible residue pair. In addition to negative scores, alignments may

Algorithm 1 Dynamic Programming (DP) Algorithm

```
Align($s_1[1 \dots m], s_2[1 \dots n]$)
1: Initialize the $0^{th}$ row and $0^{th}$ column of the DP table
2: for $i$: 1 to $m$ do
3:     for $j$: 1 to $n$ do
4:         $S_{i,j} \leftarrow T_{i-1,j-1} + W(i,j)$.
5:         $D_{i,j} \leftarrow \max(D_{i-1,j}, T_{i-1,j} + G_{\text{open}}) + G_{\text{ext}}$.
6:         $I_{i,j} \leftarrow \max( I_{i,j-1}, T_{i,j-1} + G_{\text{open}}) + G_{\text{ext}}$.
7:         $T_{i,j} \leftarrow \max( S_{i,j}, D_{i,j}, I_{i,j})$.
```

be penalized by the insertion of gaps or deletion of characters. Gap penalties are often linear (a fixed negative value per gap) or affine [9], where the gap opening penalty is typically larger than the gap extension penalty.

There are three primary classes of sequence alignment, namely global, semi-global, and local. A *global alignment* causes the alignment to span the entire length of each sequence and is used when the two sequences are similar in length and presumed to be related. A *local alignment* identifies highly conserved regions or subsequences though the rest of the sequence may be divergent. A *semi-global alignment* does not penalize beginning or end gaps in a global alignment such that the resulting alignment will tend to overlap one end of a sequence with an end of the other sequence.

Sequence alignments are computed using dynamic programming because it is guaranteed to find an optimal alignment given a particular scoring function. Regardless of the class of alignment being computed, a dynamic programming recurrence of the following form is computed. Given two sequences $s_1[1 \dots m]$ and $s_2[1 \dots n]$, three recurrences are defined as follows: Let $S_{i,j}$ denote the optimal score for aligning the prefixes $s_1[1 \dots i]$ and $s_2[1 \dots j]$ such that the alignment ends by substituting $s_1[i]$ with $s_2[j]$. $D_{i,j}$ denotes the optimal score for aligning the same two prefixes such that the alignment ends in a deletion, i.e., aligning $s_1[i]$ with a gap character. Similarly, $I_{i,j}$ denotes the optimal score for aligning the prefixes such that the alignment ends in an insertion, i.e., aligning $s_2[j]$ with a gap character. Given the above three ways to end an alignment, the optimal score for aligning the prefixes corresponding to the subproblem $\{i, j\}$ is given by:

$$T_{i,j} = max(S_{i,j}, D_{i,j}, I_{i,j}) \tag{1}$$

The dependencies for the individual dynamic programming recurrences are as follows: $S_{i,j}$ derives its value from the solution computed for the subproblem $\{i-1, j-1\}$, while $D_{i,j}$ and $I_{i,j}$ derive their values from the solutions computed for subproblems $\{i-1, j\}$ and $\{i, j-1\}$, respectively.

A typical implementation of this dynamic programming algorithm builds a table of size $\mathcal{O}(m \times n)$ with the characters of each sequence laid out along one of the two dimensions. Each cell (i, j) in the table stores three values $S_{i,j}$, $D_{i,j}$, and $I_{i,j}$, corresponding to the subproblem $\{i, j\}$. Given the dependencies of the entries at a cell, the dynamic programming algorithms for all three sequence alignment classes can be represented using the pseudocode outlined in Algorithm 1. The algorithm has a time complexity of $\mathcal{O}(mn)$.
An optional post-processing step retraces an optimal alignment and can be completed in $\mathcal{O}(m + n)$ time assuming the entire table is stored. Details of that step are omitted.

The three classes of sequence alignment initialize the dynamic programming (DP) table differently (line 1 in Algorithm 1). SW and SG alignments initialize the first row and column of the table to zero, while NW alignments initialize the first row and column based on the gap function. The table values for SW alignments are not allowed to become negative, while NW and SG allow for negative scores.

Hereafter, for convenience according to common practice, we call the sequence with characters along the rows of the table the "query" sequence and the sequence with characters along the columns of the table the "database" sequence.

III. Challenge: Maximizing Vectorization Opportunities in Sequence Alignment

As stated previously, aligning two sequences of lengths m and n requires $\mathcal{O}(mn)$ time. This computation time becomes much more significant when computing many alignments as done in many bioinformatics applications, such as database search, multiple sequence alignment, genome assembly, and short read mapping. There have been many approaches to making this operation faster including heuristic methods such as BLAST[1]; however such heuristic methods may generate sub-optimal alignments.

There have been numerous efforts to parallelize optimal sequence alignments using vector instructions [35], [28], [6], [27], [18]. However, not all of these approaches necessarily address the same bioinformatics application. For example, database search may group database sequences to improve performance [27], while protein homology graph applications may prohibit such optimizations [5]. That said, vectorized sequence alignments generally fall into two categories: intra-task and inter-task. A vectorized alignment of a single query sequence against a single database sequence is *intra-task*. On the other hand, *inter-task* vectorization describes the alignment of a single (query) sequence against a set of (database) sequences [27]. Said another way, each vector lane represents a cell of a dynamic programming table – intra-task vectorization uses cells from the same table, whereas inter-task vectorization uses cells from independent tables. Inter-task vectorization is mainly limited to database search applications, though performance of either approach is comparable depending on the lengths of the sequences involved [4]. We focus here on the more generally applicable intra-task pairwise alignments.

Figure 1 enumerates the ways to vectorize sequence alignment. Each approach operates in a series of *vector epochs*, where each vector epoch signifies a timestep during execution when all processing elements (p) of the vector processor are concurrently working on different parts of computation, contributing to the calculation of different cells in the dynamic programming table.

In the *Blocked* approach (Figure 1 (Blocked)), proposed by [28], a vector epoch spans a subset of p contiguous cells along the dimension of the query sequence (i.e., columns). Each vector initially ignores the contributions of the upward cells. After computing a block, the new cell values are checked for correctness and potentially recomputed. Once the values converge, the last value of the current vector is used by the next vector. The drawback of the Blocked approach is that the data dependencies both within and between vectors limit the overall performance.

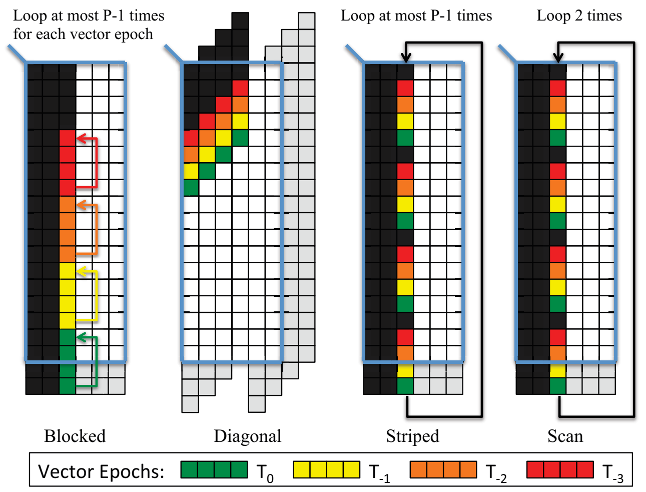

Loop at most P-1 times for each vector epoch
Loop at most P-1 times
Loop 2 times
Blocked
Diagonal
Striped
Scan
Vector Epochs: 🟩 T_0 🟨 T_{-1} 🟧 T_{-2} 🟥 T_{-3}

Fig. 1. Known ways to vectorize Smith-Waterman alignments using vectors with four elements. The tables shown here represent a query sequence of length 18 against a database sequence of length 6. Alignment tables are shown with colored elements, indicating the most recently computed cells. In order of most recently computed to least recently, the order is green, yellow, orange, and red. Dark gray cells were computed more than four vector epochs ago. Light gray cells indicate padded cells, which are required to properly align the computation(s) but are otherwise ignored or discarded. The blue lines indicate the relevant portion of the tables with the table origin in the upper left corner. (Blocked) Vectors run parallel to the the query sequence. Each vector may need to recompute until values converge. First described by Rognes and Seeberg [28]. (Diagonal) Vectors run parallel to the anti-diagonal. Fist described by Wozniak [35]. (Striped) Vectors run parallel to the query using a striped data layout. A column may need to be recomputed at most $P-1$ times until the values converge. First described by Farrar [6]. (Scan) This is the approach taken in this paper. It is similar to (Striped) but requires exactly two iterations over a column.

In the *Diagonal* approach (Figure 1 (Diagonal)), an epoch spans a subset of p contiguous cells along a single diagonal of the table [35]. Note that the cells along the same diagonal have no interdependencies as their dependent values come from the cells in the previous two diagonals. However, wasteful computation is caused in this approach by padding the table with cells to properly align the computation. Another disadvantage is the irregular memory access along the diagonal.

In the *Striped* approach (Figure 1 (Striped)), proposed by Farrar [6], a vector epoch spans a subset of p evenly spaced cells along the dimension of the query sequence. This scheme eliminates the data dependencies both within and between vectors by striping the vector parallel to the query sequence. Similar to Blocked, this approach also initially ignores the contributions of the upward values and makes additional passes over each column until the values converge. Often, the values converge before having to compute the column entirely a second time. This significantly improves overall performance. That said, in the worst case, the column would be recomputed as many times as there are elements in the vectors.

Lastly, our solution leverages the striped layout, but it uses a prefix scan formulation of the dynamic programming recurrence. The prefix scan recurrence is straightforward though it was initially designed for GPUs and requires a lengthy proof to confirm its equivalence to the original problem [14]. Compared to Blocked and Striped, which initially ignore the upward cells, the prefix scan calculates a temporary value and later uses the temporary value to find the final cell value. As shown in Figure 1 (Scan), our solution requires exactly two iterations over each column.

Algorithm 2 A generic pseudocode for a column-wise vectorized sequence alignment

```
Align($s_1[1 \dots m], s_2[1 \dots n]$)
  for each character in database sequence do
    for each vector epoch in column do
      Load substitution scores from query profile.
      Load previous column's corresponding cell values.
      Compute next cell values.
```

Algorithm 3 A generic pseudocode for a diagonal vectorized sequence alignment

```
Align($s_1[1 \dots m], s_2[1 \dots n]$)
  for every $p$ characters in database sequence do
    for each vector epoch in diagonal do
      Gather substitution scores for each $s_1[i], s_2[j]$ pair.
      Use previous vector epoch directly.
      Compute next cell values.
```

TABLE I
Relative Performance of Each Vectorized Approach, Reaffirming Similar Results from Farrar [6] for Striped.

Approach	Scalar	Blocked	Diagonal	Striped
Time (s)	70.5	10.6	9.9	4.7
Speedup	1.0	6.6	7.2	15.1

All four vectorization schemes can be summarized using the generic pseudocodes in Algorithm 2 and Algorithm 3, with Blocked, Striped, and Scan mapping to Algorithm 2 and Diagonal to Algorithm 3.

The order in which we presented the vectorized approaches corresponds generally to their relative performance. Table I briefly lists the relative performance improvement of each approach. For this analysis, we compare every sequence to each other using a small but representative protein sequence dataset. We implemented each vectorization technique shown here using the SSE4.1 ISA via the compiler intrinsics found in the immintrin.h header, splitting the 128-bit vector register into eight 16-bit integers. The results of each vector implementation were validated against the scalar result. The first two approaches, namely Blocked and Diagonal, are improved over the scalar implementation, while the Striped approach performs significantly better. For this reason, we only consider Striped and our new Scan implementation for the remainder of our paper. These results reaffirm similar findings from Farrar [6] for Striped.

The objective of our work is to understand the impacts of widening vector registers on a broad class of sequence alignment algorithms in light of their workload characteristics and parameter ranges.

IV. Algorithmic Comparison of Striped and Scan

Prior to our experimental evaluation of the Striped and Scan approaches to vectorizing sequence alignment, it is important to understand the algorithmic differences between these approaches. First, we look at the new recurrences for Scan and discuss how to optimally implement them using vectors. This is followed by an analysis of each algorithm's computational complexity.

There are two known formulations for linearizing the data dependencies within the sequence alignment recurrences by using parallel prefix (scan) computation. The approach was first described by Aluru et al. [2], however the formulation by Khajeh-Saeed et al. [14] is simpler though it requires a lengthy proof to confirm its equivalence to the original problem. For comparison with the description in Section II, equations from [14] are repeated here in Equations 2 through 5. Note that this recurrence, computing column by column, initially ignores the influence of the column maximum $D_{i,j}$ and calculates a temporary variable $\widetilde{T}_{i,j}$.

$$I_{i,j} = max(I_{i,j-1}, T_{i,j-1} + G_{\text{open}}) + G_{\text{ext}} \tag{2}$$

$$\widetilde{T}_{i,j} = max(T_{i-1,j-1} + W(i,j), I_{i,j}) \tag{3}$$

$$\widetilde{D}_{i,j} = max_{1<k<j}(\widetilde{T}_{i-k,j} - kG_{\text{ext}}) \tag{4}$$

$$T_{i,j} = max(\widetilde{T}_{i,j}, \widetilde{D}_{i,j} + G_{\text{open}}) \tag{5}$$

The parallel scan approach is the focus of our implementation. Ideally, the parallel scan would be implemented as described in Blelloch [3], mapping a balanced binary tree over the values and using an upsweep followed by a downsweep and applying the associative operator at each node. This is indeed the approach taken by Khajeh-Saeed et al. in [14] though the implementation is written for a GPU. The optimal time complexity of this operation is $\mathcal{O}(n/p + \lg(n))$.

Unfortunately, such operations are not efficient to implement using SIMD vectors. Instead, the parallel scan is implemented in two passes. The first pass has each vector element p compute its portion of the scan in n/p iterations, where n is the number of cells in one column of the DP table (equal to the length of the query sequence). Next, a "horizontal" scan is performed on the resulting vector in $p-1$ operations. Though horizontal operations were added starting in SSE3, our scan requires a combination of addition and maximum rather than just addition or subtraction. Further, the latency and throughput of the horizontal operations are large relative to our approach of shifting the vector $p-1$ times. After the horizontal scan is performed, the resulting vector is shifted to prepare it for the second pass, where it becomes the initial conditions. The second pass is performed in n/p iterations. Instead of the ideal time complexity for the parallel prefix scan, we are left with a time complexity of $\mathcal{O}(n/p + p)$. The pseudocode for the Scan implementation appears in Algorithm 4.

Comparing Algorithms 4 and 5, the Scan approach is only superficially similar to Striped. For example, as in [6], the Scan implementation is also striped parallel to the query sequence. In addition, both approaches make at least one full pass over each column in the DP table, but this is where the similarities end. The amount of work performed by each differs in two ways. First, the Striped approach calculates three values per cell, while the Scan approach calculates an additional, temporary value. Second, the Striped approach is often able to abort its additional passes over the column if the upper cell values within the current column no longer contribute to the current cell value. However, in the worst case, it may recompute the column $p-1$ times. The Scan approach will iterate over a column exactly twice and performs the horizontal scan of the intermediate vector $p-1$ times for each column.

Algorithm 4 Pseudocode for Scan

```
Align_Scan($s_1[1 \dots m], s_2[1 \dots n]$)
 1: Create striped query profile
 2: $L \leftarrow (m + p - 1)/p$                ▷ number of vector epochs
 3: for each column $j$ along database sequence do
 4:     for each vector epoch $i$ in $1 \dots L$ do
 5:         Load query profile
 6:         Compute and store $I$
 7:         Compute and store $\widetilde{T}$
 8:         Compute initial pass of $\widetilde{D}$
 9:     Local prefix scan of $\widetilde{D}$ result
10:     for each vector epoch $i$ in $1 \dots L$ do
11:         Compute second pass of $\widetilde{D}$
12:         Compute and store $T$
```

Algorithm 5 Pseudocode for Striped

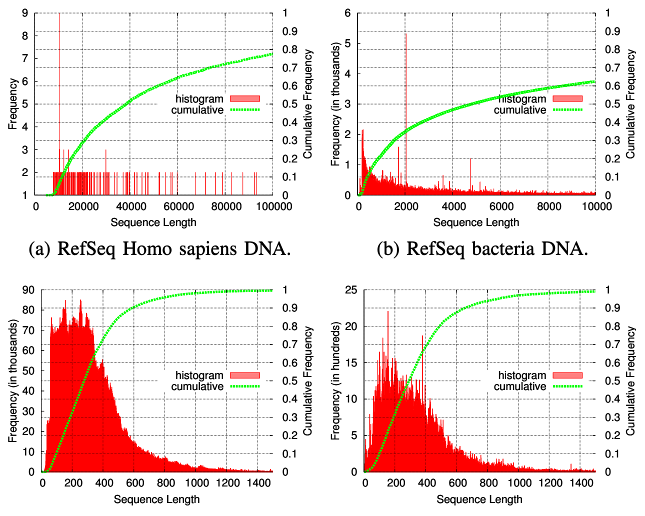
```
Align_Striped($s_1[1 \dots m], s_2[1 \dots n]$)
 1: Create striped query profile
 2: $L \leftarrow (m + p - 1)/p$                ▷ number of vector epochs
 3: for each column $j$ along database sequence do
 4:     Initialize $D$
 5:     Load $T_{j-1}[L]$
 6:     for each vector epoch $i$ in $1 \dots L$ do
 7:         Load query profile
 8:         Compute $S$
 9:         Load $I_{j-1}$
10:         Compute and store $T$
11:         Compute and store next $I$
12:         Compute next $D$
13:         Load previous $T_{j-1}[i]$ for next iteration
14:     while any $D > T$ do
15:         for each vector epoch $i$ in $1 \dots L$ do
16:             Recompute $T$
17:             Recompute $D$
18:             if not any $D > T$ then
19:                 Break
```

Summarizing, the time complexity to compute a column of the DP table using the Scan approach is $\mathcal{O}(2n/p + p)$, where n is the length of the query sequence and p is the number of lanes, i.e., processing elements. The Striped approach is nearly identical in its computational complexity with $\mathcal{O}((1+C)*n/p)$, where the additional parameter C is the corrective factor. Given the total number of corrections made k during the processing of a DP table with a database sequence of length m, C can be calculated as $C = k/m/\lfloor(n+p-1)/p\rfloor$. The corrective factor C is not necessarily a whole number. For example, a column might converge before reaching its end. For Striped to be effective, $0 <= C << (p-1)$, and ideally it would be zero.

The detailed evaluation in Section VI shows that, due to C, each algorithm has its respective strengths.

V. Workload Characterization

The performance of the sequence alignment algorithms depends, in part, on the length of the input. Therefore, it



(a) RefSeq Homo sapiens DNA. (b) RefSeq bacteria DNA.

(c) RefSeq bacteria proteins. (d) UniProt proteins.

Fig. 2. Distribution of sequence lengths for all [5,448] RefSeq Homo sapiens DNA (a), all [2,618,768] RefSeq bacteria DNA (b), all [33,119,142] RefSeq bacteria proteins (c), and full [547,964] UniProt protein (d) datasets. Protein datasets are skewed toward shorter sequences, while DNA datasets contain significantly longer sequences. Due to the presence of long sequences, the figures are truncated before their cumulative frequencies reach 100 percent.

is important to know the distributions of sequence lengths for any given set of sequences. We observe that the majority of protein sequences tend to be 300 characters or less, which will have a direct impact on later performance studies.

Figure 2 characterizes the length distributions of DNA and protein sequences. Genomic DNA sequences tend to vary greatly and can be of significant length. For example, the longest Homo sapiens sequence is 125 Mbp (million base pairs), and the longest genomic bacteria sequence is 14.8 Mbp. Because of such long sequences, Figures 2a and 2b are truncated before their cumulative frequencies reach 100 percent. Protein sequences tend to be much shorter than DNA sequences. Figures 2c and 2d show that in two widely used datasets, half of the sequences are length 300 or less. This observation has significant implications for our performance analysis. These four datasets are representative of the various datasets used in other studies.

For many analyses involving the RefSeq bacteria protein dataset, we used a random sampling of 2,000 protein sequences. This dataset is hereafter "bacteria 2K". The sequences within this dataset have an average length of 314 with the longest sequence being 3,206. The frequency of sequence lengths and its cumulative distribution are similar to those found in Figure 2c.

For some experiments, we also used the UniProt release mentioned previously (hereafter "UniProt"). In our experiments involving querying a database, UniProt represented our database of sequences. The sequences within this dataset have an average length of 356 with the longest sequence being 35,213. The frequency of sequence lengths and its cumulative distribution appear in Figure 2d.

Use Cases

Database Search: Many of our experimental analyses represent the problem domain of searching an annotated database of sequences using a set of query sequences. Each query sequence is aligned, in turn, with each database sequence, returning a score or other data product for each alignment. For all analyses involving database search, we use the Bacteria 2K dataset as our query and search against the UniProt dataset as our database. Based on the sequence distributions described for each protein dataset, this use case is representative of most protein database searches.

Homology Detection: The remainder of our experimental analyses represent the problem of homology detection. Homology detection often starts with a set of unknown proteins that need to be clustered based on their similarity. Similar to database search, homology detection can be treated as if the database dataset was also used as the query dataset, the distinction being that the number of alignments to perform increases more rapidly (quadratically) as the database sizes increase.

VI. Empirical Characterization

To fully understand the impact of future vector widths on sequence alignments, a number of tests were performed to assess overall algorithm viability. We focus on single-node, single-thread performance to precisely understand the effect of hardware trends within this application domain.

Systems and Compilers: The following results were taken on single nodes of two clusters within the PNNL Institutional Computing infrastructure, namely *constance* and *philo*. Constance is based on the Intel Haswell CPU architecture featuring the AVX2 instruction set architecture (ISA). Each node has dual 12-core Intel Haswell E5-2670 v3 CPUs running at 2.3 Ghz with 64 GB 2133 Mhz DDR4 memory per node. The compiler used was Intel ICC 15.0.1 using level three optimization (-O3). The philo cluster consists of nodes with dual 8-core Intel Sandy Bridge E5-2670 CPUs running at 2.6 Ghz with 64 GB of memory. Each philo node has one Intel Xeon Phi 7110P accelerator with 61 cores running at 1.1 Ghz. The compiler used was Intel ICC 13.1.1 using level three optimization (-O3) targeting the MIC architecture (-mmic). The constance cluster was used to test the SSE4.1 and AVX2 ISAs. This was done intentionally to keep the compiler and hardware identical for each of these ISAs to compare the effects of the ISAs rather than the hardware or compiler. The philo cluster was used exclusively to test the performance of the Xeon Phi accelerator that uses the Knights Corner (KNC) ISA.

Scoring Scheme Defaults: As stated in Section II, sequence alignments require a scoring scheme as input. The components of the scoring parameters include the substitution matrix, as well as the gap open and gap extension penalties. Unless stated otherwise, all of our experiments use the BLOSUM62 substitution matrix and gap open and extension penalties of -11 and -1, respectively. As with BLOSUM62 or any of the other BLOSUM substitution matrices, we use the default gap open and gap extension penalties as prescribed by the web interface to the NCBI blastp program [13].

Datasets: For all analyses, we used sequence datasets from the NCBI Reference Sequence [32] (RefSeq) database and the Universal Protein Resource [33] (UniProt) database. The

TABLE II
CACHE ANALYSIS OF ALL-TO-ALL SEQUENCE ALIGNMENT FOR THE BACTERIA 2K DATASET ON HASWELL.

DP	Method	Lanes	I-refs	D-refs
NW	striped	4	1.3e12	3.7e11
NW	striped	8	9.7e11	2.8e11
NW	striped	16	8.6e11	2.3e11
NW	scan	4	1.6e12	4.8e11
NW	scan	8	8.6e11	2.9e11
NW	scan	16	5.9e11	1.9e11
SG	striped	4	1.1e12	3.5e11
SG	striped	8	7.3e11	2.4e11
SG	striped	16	5.9e11	1.8e11
SG	scan	4	1.6e12	4.8e11
SG	scan	8	8.5e11	2.9e11
SG	scan	16	5.8e11	1.9e11
SW	striped	4	1.3e12	3.4e11
SW	striped	8	7.3e11	2.3e11
SW	striped	16	6.1e11	1.8e11
SW	scan	4	1.8e12	4.7e11
SW	scan	8	9.0e11	2.9e11
SW	scan	16	6.1e11	1.9e11

RefSeq project is an ongoing effort to provide a curated, non-redundant collection of sequences, grouped by taxonomy, e.g., fungi, bacteria. UniProt is a comprehensive resource for protein sequence and annotation data. Specifically, from RefSeq we used release 69 which incorporates data available as of January 2, 2015. and from UniProt we used release 2015_02 from 04-Feb-15.

A. Cache Analysis

We performed a cache analysis of the homology detection problem to examine the total instruction counts and cache efficiencies for both Striped and Scan across lane counts, as well as on the Xeon Phi. We used cachegrind to generate reports for the Haswell system and Intel's vtune amplifier for the Xeon Phi system. We found, in general, both Striped and Scan exhibit negligible instruction and data cache miss rates of no more than 1 percent. All measurements, except instruction (I-refs) and data (D-refs) references, are comparable between Scan and Striped, which is why much of that information is omitted from Table II and Table III.

All implementations are extremely cache efficient. This is attributed in part to the use of the striped query profile for both implementations which was already proven by Farrar [6] to be efficient. The primary reason for the cache efficiencies in our case is the size of the problems being computed. The longest sequence in the bacteria 2K database is 3,206 and easily fits within the cache on both the Haswell CPU and the Xeon Phi CPU. Other cached data includes the values for the DP column being computed. The primary factors affecting performance are the number of instruction and data references.

As expected, the number of instruction and data references decrease as the number of vector lanes increase. However, they decrease more rapidly for Scan than Striped. Striped initially has fewer instructions than Scan when using 4 lanes. By the time 16 lanes are used, Scan has surpassed Striped. Except for the case of NW Striped, where Scan is significantly better, it is not clear whether Scan will continue to outperform Striped for SG and SW.

B. Instruction Mix Analysis

Section VI-A described the cumulative instruction counts for the homology detection problem. To understand the lane count trends shown in Table II and Table III, we ran the same homology detection problem with 16 lanes using Intel's Pin tool [20] to capture the instruction mix as shown in Figure 3. For space reasons, we omit similar results for the Xeon Phi which also uses 16 lanes. We observe that for each category of instructions, Scan rarely varies between the three classes of alignments performed. Affirming previous results, NW Striped executes more instructions relative to any other case. We observe that Striped performs more scalar operations, while Scan performs more vector operations. Scan uses more vector memory and swizzle operations, while Striped is the only one of the two that uses vector mask creation operations.

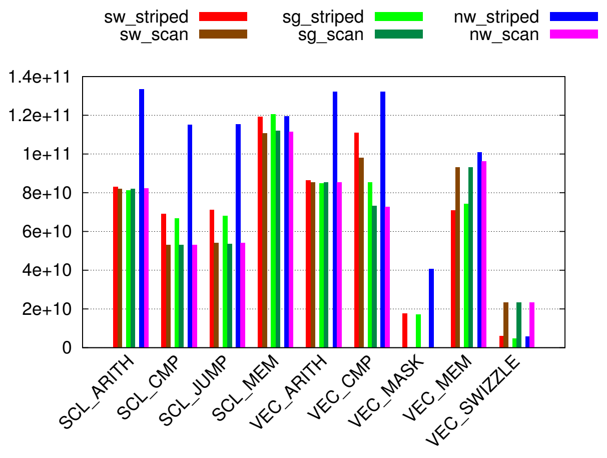


Fig. 3. Instruction mix for the homology detection problem. For each category of instructions, Scan rarely varies between the three classes of alignments performed, while NW Striped executes more instructions relative to any other case. Striped performs more scalar operations, while Scan performs more vector operations. Scan uses more vector memory and swizzle operations, while Striped is the only one of the two that uses vector mask creation operations.

Many of these instruction mix differences can be explained by the algorithmic differences noted in Section IV. The Striped approach recomputes a column until the values converge. It computes a vector mask and uses additional scalar jumps to check for convergence and break out of the recompute loop. Scan does not compute any vector masks. Assuming the best case, Striped would not need to recompute a column. In such a case, we would expect to see Scan perform more vector arithmetic and comparison instructions because it computes each column twice and computes an additional temporary value per table cell. However, the Striped approach uses more arithmetic and comparison instructions overall. This can only be explained by recomputing the columns a significant number of times. Lastly, Striped performs a few vector swizzle operations before starting a column, while Scan performs more vector swizzle operations because of the $p-1$ horizontal scan operations performed for each column of an alignment.

C. Query Length versus Performance

The analyses performed thus far indicate that the performance of Striped and Scan depends on the number of vector lanes applied to the problem. Because the vectors run parallel to the query sequence, the number of vector lanes determines the number of vector epochs based on the lengths of the queries. Therefore, we present the effect of both the query length and

TABLE III
CACHE ANALYSIS OF ALL-TO-ALL SEQUENCE ALIGNMENT FOR THE BACTERIA 2K DATASET ON XEON PHI.

	NW-Scan	NW-Striped	SG-Scan	SG-Striped	SW-Scan	SW-Striped
Instructions-Retired	6.3e11	9.1e11	6.0e11	6.3e11	6.4e11	6.5e11
CPI-Rate	2.85	2.68	2.72	2.84	2.70	2.72
L1-Misses	2.8e09	1.8e09	2.0e09	1.8e09	2.0e09	1.9e09
L1-Hit-Ratio	0.98	0.99	0.99	0.99	0.99	0.99
Vectorization-Intensity	14.84	13.81	14.82	13.94	14.98	14.10
L1-Compute-to-Data-Access-Ratio	27.34	29.14	29.79	26.00	32.02	30.39
L2-Compute-to-Data-Access-Ratio	1731.86	3375.18	2539.59	2324.74	2761.39	2582.76

number of vector lanes on the relative performance of Striped and Scan.

To that end, we used the bacteria 2K dataset as our query set and performed a database search against the UniProt database. Figures 4a through 4c show the relative speedup of the Scan approach over the Striped approach as the query lengths increase.

The relative performance of Scan versus Striped shows that both approaches have their merits in light of increasing the number of lanes. Shorter queries perform better for NW Striped, SG Scan, and SW Scan; longer queries perform better for NW Scan, SG Striped, and SW Striped.

The different classes of sequence alignments cross over the relative performance threshold (1.0 on the y-axis) at different points. For NW, the cross over points are for query lengths of 149, 149, and 149 for lane counts of 4, 8, and 16, respectively. For SG, the cross overs occur at 121, 188, and 253. For SW, the cross overs occur at 77, 77, and 152. The performance peak at the top of the bubble for SW occurs at 30, 40, and 87. In general, for SG and SW, the cross over points increase with lane counts. For SW, the cross over appears to be jumping dramatically from 8 to 16 lanes. Whether such a dramatic change occurs at 32 lanes needs to be carefully evaluated as new hardware emerges. As for NW, it appears consistently to cross over at query lengths around 150.

The cross over points are particularly concerning in light of many protein datasets being skewed toward shorter sequences. As shown in Figures 2c and 2d, the majority of protein sequences are 300 amino acids in length or shorter. Therefore, when used as query sequences for database search applications, the point at which SW performance crosses over becomes extremely relevant. Our analysis used, at most, 16 lanes, though 32 lanes will be available in the next wave of CPUs supporting the AVX-512 ISA. Future CPUs and GPUs may continue to adopt even wider vector registers, which, based on these results, is expected to further diminish the return of widening vector registers for this problem domain.

D. Query Length versus Number of Striped Corrections

Combining the results from the previous analyses, the primary factor influencing Striped performance is the number of corrections that must be made until the column values converge. The number of corrections is further impacted by the number of vector lanes utilized for the Striped computation. This validates the complexity analysis in Section IV.

For the Striped approach, combining the results from the previous analyses, the primary factor limiting the performance gains afforded by increasing the number of vector lanes is the total number of corrections. Using the same database search application as in Section VI-C, Figures 4d through 4f confirm this observation by showing the plots of query length versus total number of corrections.

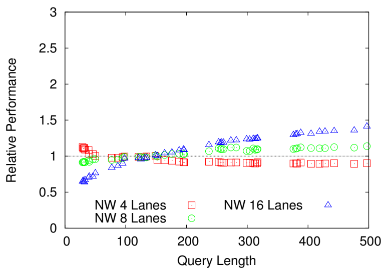
The Striped approach, for each column, initializes its D values to zero in the case of SW and to a large negative number in the cases of NW and SG. These are, of course, incorrect values that are later corrected as part of the corrective loop. There is one incorrect value introduced for each vector lane utilized. As the lanes increase, so do the number of incorrect values that can propagate across vector epochs.

The trends displayed show that query length has a direct impact on the number of Striped corrections. For NW, query length is proportional to the number of corrections. In addition, the number of corrections is increasing as lane counts increase. For SG, the number of corrections also increases as lane counts increase, though query length has less of an effect. At shorter query lengths, the number of corrections is less predictable. For SW, there is a clear trend, forming a bubble in the number of corrections relative to the number of lanes. The bubble consistently plateaus when the query length reaches ten times the number of lanes. The peak of the bubbles for SW start at $5E9$ for 4 lanes, then $8E9$ for 8 lanes, and $16E9$ for 16 lanes—roughly doubling as the number of lanes double. Coupled with the computational complexity discussion in Section IV, this trend will have a severe impact on SW performance as lanes continue to widen. The total number of corrections *increases* as the number of lanes p increase. This implies a correlation between the number of lanes and worst-case performance for the Striped approach.

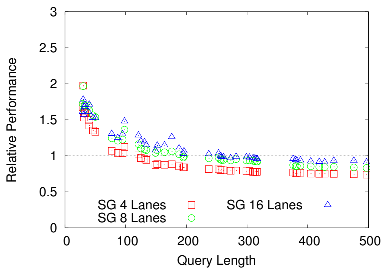
E. Scoring Criteria Analysis

Having performed a detailed, low-level analysis of Striped and Scan, it remains to be seen whether the observations hold for a user-level analysis. The next experiment is a typical evaluation of the effect that the gap and substitution matrix scoring criteria has on the various implementations. We used the homology detection application with the Bacteria 2K dataset. The substitution matrices used were BLOSUM{45,50,62,80,90} with their corresponding default gap open and extension penalties of $-15 - 2k$, $-13 - 2k$, $-11 - k$, $-10 - k$, and $-10 - k$, respectively. The scoring criteria analysis appears in Figure 5.

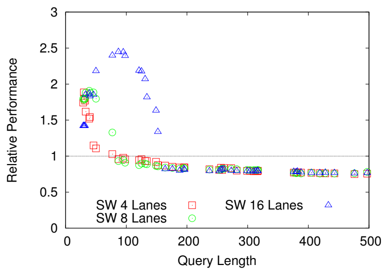
Because the convergence criteria for the column computation in Striped depends on the values of T and D, different substitution matrices and gap penalties affect how quickly the values of T and D diverge—the more divergent, the more corrections must be made. A similar analysis was done by Farrar [6], though it did not consider NW or SG. Because the Scan approach does not conditionally compute any of its values,



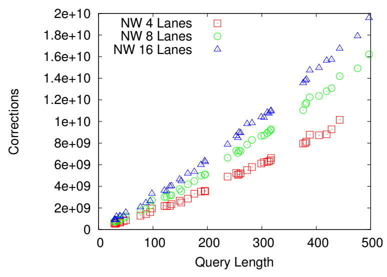
(a) NW: Query Length vs. Relative Performance of Scan over Striped



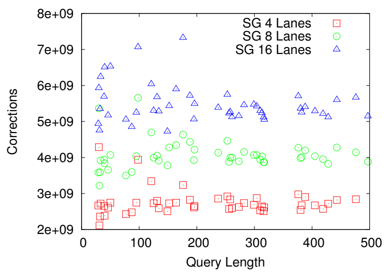
(b) SG: Query Length vs. Relative Performance of Scan over Striped



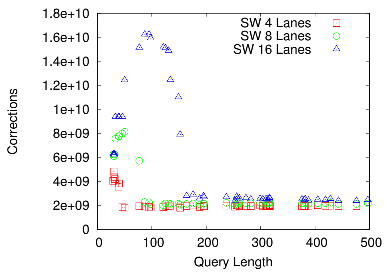
(c) SW: Query Length vs. Relative Performance of Scan over Striped



(d) NW: Query Length vs. Total Corrections



(e) SG: Query Length vs. Total Corrections



(f) SW: Query Length vs. Total Corrections

Fig. 4. The relative performance of Scan versus Striped (a-c) shows that both approaches have their merits in light of increasing the number of vector lanes. Shorter queries perform better for NW Striped, SG Scan, and SW Scan. Longer queries perform better for NW Scan, SG Striped, and SW Striped. The reasons for the relative performance differences can be attributed to the number of times the Striped approach must correct the column values before reaching convergence (d-f).

TABLE IV
DECISION TABLE SHOWING WHICH ALGORITHM SHOULD BE USED GIVEN A PARTICULAR CLASS OF SEQUENCE ALIGNMENT AND QUERY LENGTH.

	Short	Crossover Point			Long
	$<$ Cross	4 Lanes	8 Lanes	16 Lanes	$>$ Cross
NW	Striped	149	149	149	Scan
SG	Scan	121	188	253	Striped
SW	Scan	77	77	152	Striped

the runtimes are stable regardless of the selected substitution matrix or gap penalties.

The Scan approach has stable performance relative to the scoring scheme because it unconditionally makes two passes over each DP table column. The Striped approach varies a moderate amount between selected scoring schemes, generally performing better for smaller gap penalties. As the lane counts increase, the Scan approach eventually overtakes the Striped approach, confirming the results in Section VI-C.

F. Prescriptive Solutions on Choice of Algorithm

For the particular input datasets we studied, the choice of algorithm to use given a particular class of sequence alignment and query length is summarized in Table IV.

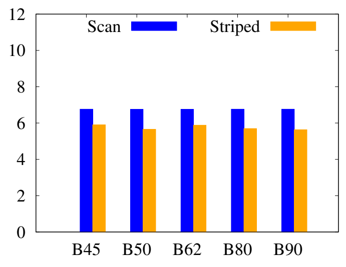
We observe that the three algorithms, despite their similarities, exhibit distinct characteristics. Specifically, NW requires a different choice of schemes as compared to SG and SW. In addition, the choice of the schemes is clearly dictated by the input size. Whereas NW performs better with the Striped implementations for short sequences, SG and SW are faster when using the Scan implementation. The choices are reversed for the long sequences, with NW performing better with Scan and SG/SW performing better with the Striped implementation. The cross-over between what is classified as a short vs a long sequence depended on the SIMD lane width. These widths are shown in columns 3–5 in Table IV. In general, the cross-over points are less than 300, falling within the first half of length distributions in Figure 2. Therefore, for longer sequences toward the right of the length distributions in Figure 2, the choice of schemes is clear. The cross-over point increased with SIMD lane width for SG and SW. The number of corrections for NW does not vary significantly with lane width, explaining the stability of the cross-over point across SIMD width for NW. In all the cases, widening vector registers makes the parallel scan implementation of the sequence alignment algorithms more attractive.

G. Multi-Threaded Performance

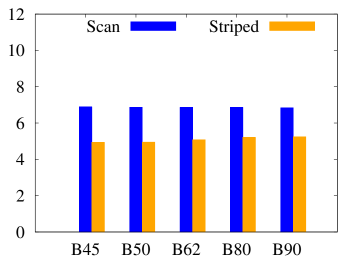
Single-node, multi-threaded performace was measured but is not presented due to space limitations. Cache behavior, instruction mix, relative performance of Scan over Striped, as well as the user-level performance analysis did not exhibit any statistically significant change.

VII. Related Work

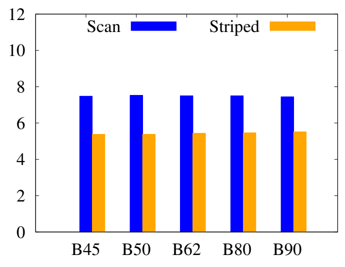
Vector extensions, such as SSE, have been part of the x86 since the 1990s, with applications in graphics [11], signal processing [7], and scientific applications [8]. Based on such vector extensions, auto-vectorization has also been a widely studied topic for many years [24], [25], [34]. Recently, Maleki *et al.* [21] provided a comprehensive evaluation of modern vectorizing compilers and discussed the limitations of auto-vectorization performed by these compilers.



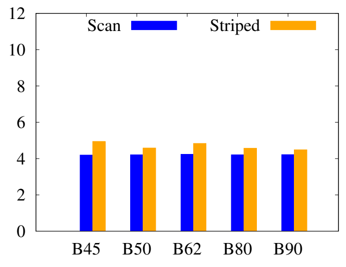
(a) NW 4 Lanes SSE41



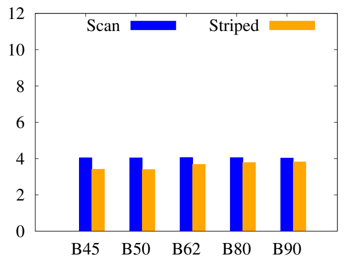
(b) SG 4 Lanes SSE41



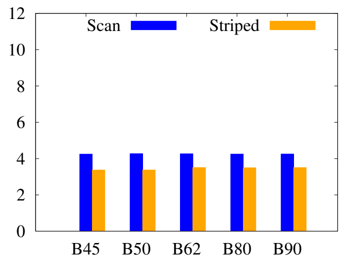
(c) SW 4 Lanes SSE41



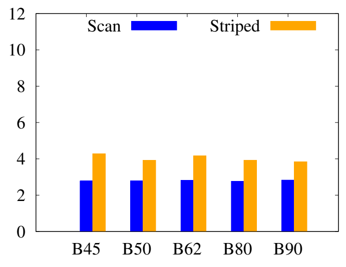
(d) NW 8 Lanes AVX2



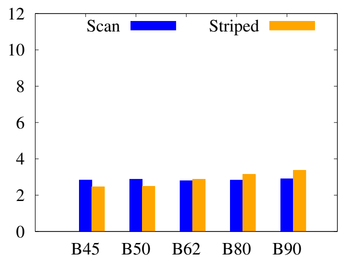
(e) SG 8 Lanes AVX2



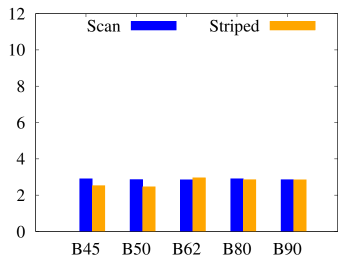
(f) SW 8 Lanes AVX2



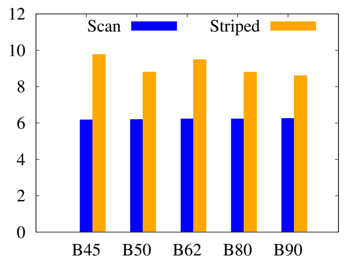
(g) NW 16 Lanes AVX2



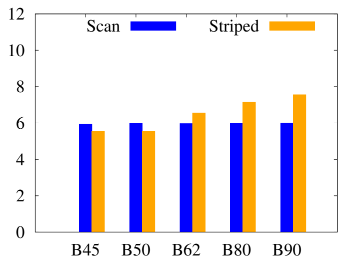
(h) SG 16 Lanes AVX2



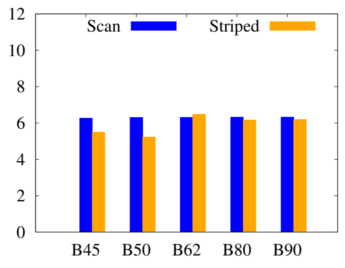
(i) SW 16 Lanes AVX2



(j) NW 16 Lanes KNC



(k) SG 16 Lanes KNC



(l) SW 16 Lanes KNC

Fig. 5. Total compute times in seconds (Y-axis) for global (NW, left column), semi-global (SG, center column), and local (SW, right column) alignments using the bacteria 2K dataset for a homology detection application. The lane counts increase moving from the first row to the third row, increasing from 4 to 8 and lastly to 16. The fourth row consists of the results for KNC which is also 16 lanes. For each BLOSUM matrix analyzed, the default gap open and extension penalties from NCBI were used as in Section VI-E. By the time 8 lanes are used, NW Scan consistently outperforms NW Striped. At 16 lanes, Scan begins to outperform Striped for many of the selected scoring schemes.

Although many algorithms and applications can naturally benefit from the auto-vectorization techniques, there are still many that are difficult to vectorize due to their dependence on irregular data structures, dense branch operations such as trees or graph traversals, or data dependencies.

Recently, there were many efforts focusing on handling this challenge. For example, featuring many query inputs, both Kim *et al.* [15] and Jo *et al.* [12] proposed a set of techniques to vectorize binary tree search and tree traversal operations, respectively. Ren *et al.* [26] vectorized decision tree forests and regular expression matching algorithms on SSE. Liu *et al.* [17] parallelized sparse matrix-vector multiplication on the Xeon Phi architecture. All of these works emphasize carefully laying out the irregular data structures to improve the cache performance and exploring the potential of fine-grained parallelism among multiple tasks.

Within the context of sequence alignment, a very important algorithm in the bioinformatics domain, our work considers the vectorization problem from a totally new perspective. We carefully study the impact of increasing SIMD widths on various classes of this algorithm and propose some useful predictions for researchers to choose which implementations to use in the future. More closely related to our work, Schaub *et al.* [29] carefully studied the impact of varying the SIMD width on control-flow and memory divergence, while our work focuses on the impact on different SIMD algorithms for sequence alignment.

In the past few years, sequence alignment has been studied comprehensively. There are numerous implementations of the SW algorithm across varied hardware technologies, including SIMD microprocessors [35], [28], [6], [27], GPU accelerators [22], [19], Xeon Phi accelerators [18], and FPGAs [16]. Here, we focused on SIMD microprocessors. None of these aforementioned efforts consider the impact of increasing SIMD widths as ours does.

VIII. Conclusions and Future Work

Current and future CPU architectures are trending toward wider vector registers. Therefore, it is imperative that vectorized codes are not adversely affected by these widening trends. This paper selected one of the fundamental algorithms from bioinformatics to analyze against these trends. The results were clear: the state-of-the-art implementations based on a striped data layout were inadequate when it comes to realizing the full potential of wider vector registers. At 8 lanes, NW Scan consistently outperforms NW Striped. At 16 lanes, SG and SW Scan outperform Striped for many of the selected scoring schemes. We expect Scan to fully surpass Striped in the next generation of SIMD widths.

We presented a novel SIMD implementation of a parallel scan based algorithm and demonstrate that it overcomes the limitations of the striped scheme. Experimental evaluation demonstrates the three classes of sequence alignment—Needleman-Wunsch, semi-global, Smith Waterman—though very similar in their algorithmic structures, differ widely in their execution times with the Striped and Scan implementations, and in their effective use of wide vector units. We identify the input lengths and vector widths for which one scheme is preferable to the other.

Since the Scan approach is favorable to smaller query lengths, it would be amenable to partitioning the SW problem into smaller tiles. Such tiling approaches are being studied currently in other domains in order to improve cache utilization. This would be one strategy for the efficient alignment of much longer sequences, i.e., DNA.

Acknowledgment

The research was performed using PNNL Institutional Computing at Pacific Northwest National Laboratory. The research was supported, in part, by the DOE award DE-SC-0006516.

References

[1] S. F. Altschul, T. L. Madden, A. A. Schffer, J. Zhang, Z. Zhang, W. Miller, and D. J. Lipman. Gapped BLAST and PSI-BLAST: a new generation of protein database search programs. *Nucleic Acids Research*, 25(17):3389 3402, 1997.

[2] S. Aluru, N. Futamura, and K. Mehrotra. Parallel biological sequence comparison using prefix computations. *JPDC*, 63(3):264–272, 2003.

[3] G. E. Blelloch. Prefix sums and their applications. 1990.

[4] J. Daily. Parasail: Simd c library for global, semi-global, and local pairwise sequence alignments. *BMC Bioinformatics*, 17(1):1–11, 2016.

[5] J. Daily, A. Kalyanaraman, S. Krishnamoorthy, and A. Vishnu. A work stealing based approach for enabling scalable optimal sequence homology detection. *JPDC*, 7980:132 – 142, 2015.

[6] M. Farrar. Striped smith–waterman speeds database searches six times over other simd implementations. *Bioinformatics*, 23(2):156–161, 2007.

[7] F. Franchetti and M. Puschel. A SIMD vectorizing compiler for digital signal processing algorithms. In *IPDPS*, pages 7–pp, 2002.

[8] C. García, R. Lario, M. Prieto, L. Piñuel, and F. Tirado. Vectorization of multigrid codes using SIMD ISA extensions. In *IPDPS*, 2003.

[9] O. Gotoh. An improved algorithm for matching biological sequences. *Journal of Molecular Biology*, 162(3):705–708, Dec. 1982.

[10] S. Henikoff and J. G. Henikoff. Amino acid substitution matrices from protein blocks. *Proceedings of the National Academy of Sciences*, 89(22):10915–10919, 1992.

[11] N. Ide, M. Hirano, Y. Endo, S. Yoshioka, H. Murakami, A. Kunimatsu, T. Sato, T. Kamei, T. Okada, and M. Suzuoki. 2.44-GFLOPS 300-MHz floating-point vector-processing unit for high-performance 3D graphics computing. volume 35, pages 1025–1033, 2000.

[12] Y. Jo, M. Goldfarb, and M. Kulkarni. Automatic vectorization of tree traversals. In *PACT*, pages 363–374, 2013.

[13] M. Johnson, I. Zaretskaya, Y. Raytselis, Y. Merezhuk, S. McGinnis, and T. L. Madden. Ncbi blast: a better web interface. *Nucleic Acids Research*, 36(suppl 2):W5–W9, 2008.

[14] A. Khajeh-Saeed, S. Poole, and J. Blair Perot. Acceleration of the smith-waterman algorithm using single and multiple graphics processors. *J. Comput. Phys.*, 229(11):4247–4258, June 2010.

[15] C. Kim, J. Chhugani, N. Satish, E. Sedlar, A. D. Nguyen, T. Kaldewey, V. W. Lee, S. A. Brandt, and P. Dubey. FAST: fast architecture sensitive tree search on modern CPUs and GPUs. In *ACM SIGMOD/PODS*, pages 339–350, 2010.

[16] I. Li, W. Shum, and K. Truong. 160-fold acceleration of the smith-waterman algorithm using a field programmable gate array (fpga). *BMC bioinformatics*, 8(1):185, 2007.

[17] X. Liu, M. Smelyanskiy, E. Chow, and P. Dubey. Efficient sparse matrix-vector multiplication on x86-based many-core processors. In *ICS'13*, pages 273–282, 2013.

[18] Y. Liu and B. Schmidt. Swaphi: Smith-waterman protein database search on xeon phi coprocessors. In *ASAP'14*, pages 184–185, June 2014.

[19] Y. Liu, A. Wirawan, and B. Schmidt. Cudasw++ 3.0: accelerating smith-waterman protein database search by coupling cpu and gpu simd instructions. *BMC Bioinformatics*, 14(1):117, 2013.

[20] C.-K. Luk, R. Cohn, R. Muth, H. Patil, A. Klauser, G. Lowney, S. Wallace, V. J. Reddi, and K. Hazelwood. Pin: Building customized program analysis tools with dynamic instrumentation. PLDI '05, pages 190–200, New York, NY, USA, 2005. ACM.

[21] S. Maleki, Y. Gao, M. J. Garzaran, T. Wong, and D. A. Padua. An evaluation of vectorizing compilers. In *PACT*, pages 372–382, 2011.

[22] S. Manavski and G. Valle. Cuda compatible gpu cards as efficient hardware accelerators for smith-waterman sequence alignment. *BMC bioinformatics*, 9(Suppl 2):S10, 2008.

[23] S. Needleman and C. Wunsch. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of Molecular Biology*, 48(3):443–453, Mar. 1970.

[24] D. Nuzman, I. Rosen, and A. Zaks. Auto-vectorization of interleaved data for SIMD. In *PLDI*, volume 41, pages 132–143. ACM, 2006.

[25] D. Nuzman and A. Zaks. Outer-loop vectorization: revisited for short SIMD architectures. In *PACT*, pages 2–11, 2008.

[26] B. Ren, G. Agrawal, J. R. Larus, T. Mytkowicz, T. Poutanen, and W. Schulte. SIMD parallelization of applications that traverse irregular data structures. In *CGO*, pages 1–10, 2013.

[27] T. Rognes. Faster Smith-Waterman database searches with inter-sequence SIMD parallelisation. *BMC Bioinformatics*, 12(1):221, 2011.

[28] T. Rognes and E. Seeberg. Six-fold speed-up of smithwaterman sequence database searches using parallel processing on common microprocessors. *Bioinformatics*, 16(8):699–706, 2000.

[29] T. Schaub, S. Moll, R. Karrenberg, and S. Hack. The impact of the SIMD width on control-flow and memory divergence. *ACM Transactions on Architecture and Code Optimization (TACO)*, 11(4):54, 2015.

[30] T. Smith and M. Waterman. Identification of common molecular subsequences. *Journal of Molecular Biology*, 147(1):195 – 197, 1981.

[31] D. J. States, W. Gish, and S. F. Altschul. Improved sensitivity of nucleic acid database searches using application-specific scoring matrices. *Methods*, 3(1):66 – 70, 1991.

[32] T. Tatusova, S. Ciufo, B. Fedorov, K. ONeill, and I. Tolstoy. Refseq microbial genomes database: new representation and annotation strategy. *Nucleic Acids Research*, 42(D1):D553–D559, 2014.

[33] The UniProt Consortium. UniProt: a hub for protein information. *Nucleic Acids Research*, 43(D1):D204–D212, 2015.

[34] K. Trifunovic, D. Nuzman, A. Cohen, A. Zaks, and I. Rosen. Polyhedral-model guided loop-nest auto-vectorization. In *PACT*, pages 327–337, 2009.

[35] A. Wozniak. Using video-oriented instructions to speed up sequence comparison. *CABIOS*, 13(2):145–150, 1997.