An In-Storage Implementation of Smith-Waterman in Resistive CAM

Roman Kaplan, Leonid Yavits, Uri Weiser and Ran Ginosar
Dept. of Electrical Engineering, Technion, IIT
Haifa, Israel
sromanka@tx.technion.ac.il

Abstract—An in-storage implementation of the Smith-Waterman sequence alignment algorithm on a resistive content addressable memory (ReCAM) based storage is proposed. The ReCAM native compare operation is used to find matching base-pairs in a fixed number of cycles, regardless of the sequence length. Our in-storage implementation is simulated and compared to state-of-the-art systolic arrays and GPU-based solutions. We show a 3.4x higher throughput and an order of magnitude lower power dissipation.

Index Terms—Local Sequence Alignment, Near Data Computing, Processing In Storage, Resistive RAM.

1 Introduction

DNA sequences are frequently compared and searched for matching or near-matching patterns. The Smith-Waterman algorithm [1] (*S-W*) is based on dynamic programming and designed to find optimal local alignment between two sequences. This alignment is based on the computation of a scoring matrix. The original S-W algorithm was later enhanced by Gotoh [2] to support the affine gap model.

The quadratic running time complexity of S-W makes it a good candidate for hardware acceleration. A recent survey on S-W implementations [6] identified the following main acceleration platforms: FPGAs, GPUs and Intel Xeon Phi. The state-of-the-art performance of each will be presented in Section 4.

In this paper, we propose an in-storage implementation of the S-W scoring step based on Resistive Content Addressable Memory (ReCAM) [5]. ReCAM combines data storage with data processing. We show that in-storage ReCAM implementation may achieve on average 3x higher throughput than a GPU implementation, while having 12x lower power dissipation.

The rest of this paper is organized as follows. Section 2 presents the architecture of a ReCAM based storage. Section 3 explores the in-storage implementation of S-W. Section 4 discusses simulation results and Section 0 offers conclusions.

2 ReCAM Based Storage

Resistive memories store information by modulating the resistance of nanoscale storage elements. Resistive memories are nonvolatile, free of leakage power, and emerge as long-term potential alternatives to charge-based memories, including NAND flash.

STT-MRAM has generally lower write energy, shorter write latency, and higher write endurance than other resistive memory technologies [10]. A storage element in an STT-MRAM cell is a magnetic tunnel junction (MTJ), which relies on magnetoresistance to encode information. A MTJ consists of two ferromagnetic layers and a tunnel barrier layer. One of the ferromagnetic layers has a fixed magnetic spin, whereas the spin of the second layer can be influenced by applying high-amplitude current. The direction of spins relative to one another determines the cell resistance. Parallel (anti-parallel) spins result in low (high) resistance.

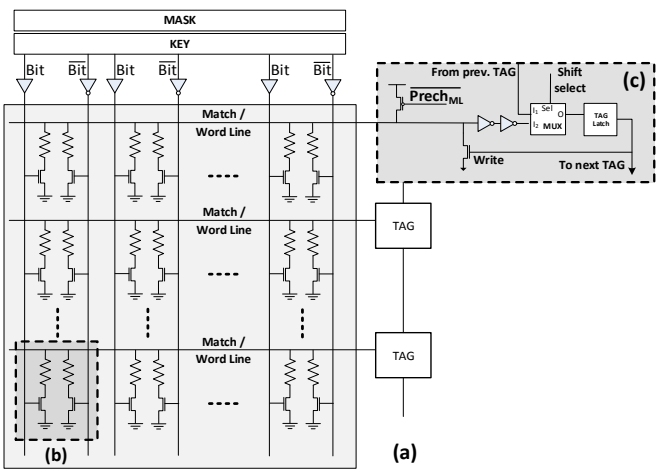

Fig 1. Resistive CAM crossbar array (a) Resistive crossbar, (b) STT-MCAM bitcell (c) TAG logic

Fig 1 (a) shows the resistive CAM crossbar. A bitcell (Fig 1 (b)) consists of two transistors and two resistive elements (2T2R). The KEY register contains a key data word

- *Roman Kaplan, E-mail: sromanka@tx.technion.ac.il*
- *Leonid Yavits, E-mail: yavits@tx.technion.ac.il*
- *Uri Weiser, E-mail: uri.weiser@ee.technion.ac.il*
- *Ran Ginosar, E-mail: ran@ee.technion.ac.il*

Authors are with the Department of Electrical Engineering, Technion-Israel Institute of Technology, Haifa 32000, Israel.

to be written or compared against. The MASK register defines the active fields for write and read operations, enabling bit selectivity. The TAG register (Fig 1 (c)) marks the rows that are matched by the compare operation and may be affected by a parallel write.

In a conventional CAM, a compare operation is typically followed by a read of the matched data word. When in-storage processing involves arithmetic operations, a compare is usually followed by a parallel write into the unmasked bits of all tagged rows.

Any computational expression can be efficiently implemented in ReCAM storage using line-by-line execution of the truth table of the expression [5]. Arithmetic operations are typically performed bit-serially. Table 1 lists operations used in S-W implementation. Parallel compare and write are assumed to take a single cycle each. Parallel shift down requires three cycles per bit (compare with const '1' copies the source bitcolumn into the TAG; shift the TAG vector down by setting the shift-select line (see Fig 1(c)); write '1' copies the TAG to the destination bitcolumn). Addition is performed in a bit-serial manner using a truth table approach [5]. Arithmetic operation (4th instruction in Table 1) where one of the operands is constant (shared by all rows of ReCAM) requires half the cycles compared to the same operation where both operands are variables (different in each row). Row-wise max, which finds the maximum between two elements in all rows in parallel, scans the elements bit-serially starting from the MSB. The first two bits are compared. In case of equality, the operation continues to the next pair of bits. Otherwise, the element containing '1' is selected as max.

TABLE 1
OPERATIONS USED IN S-W SCORE CALCULATION

Instruction	Cycles
32 bit	
Shift 1 row down	96
B <= A + B	256
C <= A + B	512
C <= A +/- const_vector	256
Row-wise Max (A, B)	64
Max Scalar (A)	64
2 bit	
Match	10

3 RECAM BASED IN-STORAGE SMITH-WATERMAN IMPLEMENTATION

3.1 Smith-Waterman Algorithm

S-W identifies the optimal local alignment of two sequences by computing a two-dimensional scoring matrix H. Each element of the matrix, $H_{i,j}$, is calculated according to (3); $\sigma(a_i, b_j)$ is the match score between the base-pairs in row i and column j. Matching base-pairs have positive score (e.g., +2), while mismatching have negative one (e.g., -1). The optimal alignment score between two sequences is the highest score in the matrix H.

The alignment may contain gaps in both sequences which are penalized in the score calculation (i.e., have a negative score). According to the affine gap model [2], opening a gap is harder than extending it, therefore the penalty for opening a gap is larger. This affine penalty scheme is calculated with two additional matrices, E and F, as presented in (1) and (2); G_{first} and G_{ext} are the penalties for starting and extending a gap, respectively. The matrices E, F and H are initialized with $E_{0,j} = E_{i,0} = F_{0,j} = F_{i,0} = H_{0,j} = H_{i,0} = 0$ for all i and j.

$$E_{i,j} = \max\left\{E_{i,j-1} - G_{ext}\ \text{(i)}\ ;\ H_{i,j-1} - G_{first}\ \text{(ii)}\right\} \quad (1)$$

$$F_{i,j} = \max\left\{F_{i-1,j} - G_{ext}\ \text{(i)}\ ;\ H_{i-1,j} - G_{first}\ \text{(ii)}\right\} \quad (2)$$

$$H_{i,j} = \max\left\{\begin{matrix} H_{i-1,j-1} + \sigma(a_i, b_j)\ \ \text{(i)}; \\ E_{i,j}\ \text{(ii)};\ F_{i,j}\ \text{(iii)};0\ \text{(iv)} \end{matrix}\right\} \quad (3)$$

Filling the scoring matrix H is the computationally intensive part of S-W. In a sequential implementation of the algorithm, the cell filling order is either row or column-wise. A parallel implementation allows all independent cells to be computed in the same iteration. Such cells reside in the same antidiagonal. The direction of filling the matrix is along the main diagonal, as illustrated in Fig 2.

Sequential time complexity is $O(nm)$, where n and m are the lengths of the sequences, respectively. Parallel time complexity is $O(nm/p)$, where p is the number of parallel processing units. In ReCAM, processing unit is a memory row. Therefore, unlike in GPU or FPGA implementations, p could be larger than $max\{n, m\}$ even for very large n and m. Hence ReCAM can achieve linear time complexity of $O(max\{n, m\})$.

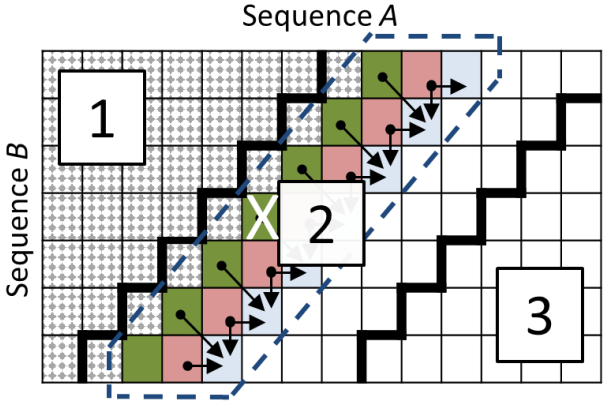

Fig 2. Parallel S-W scoring snapshot. The dot-patterened cell scores were already computed (entire section 1 and left-most antidiagonal of section 2). Plain-colored cells are stored in ReCAM (antidiagonals within the dashed borders). The cell marked with X contains the global maximum score. Thick borders separate ReCAM implementation to different logical sections.

3.2 ReCAM Implementation of S-W

In this work we focus on finding the maximal alignment score, therefore storing the entire matrix in memory, as typically required for backtracking, is not needed. A total of four antidiagonals are required to compute a new antidiagonal of H: two of H (see (3) and Fig 2), one of E (see (1)) and one of F (see (2)). Five matrix antidiagonals are stored

in ReCAM in each iteration. An additional field *temp* is reserved to store partial results. The overall space complexity required for executing the algorithm is therefore $O(\min\{n,m\})$.

Each of the five antidiagonals is mapped to a 32-bit column in ReCAM. Every ReCAM row retains one element of the vectors *temp, AD[2], AD[1], AD[0], F, E, seqA, seqB*, where *AD[0-2], E, F* contain the antidiagonals of *H, E, F*, respectively, $SeqA$ and $SeqB$ contain the 2-bit elements of sequences A and B respectively (Fig 3).

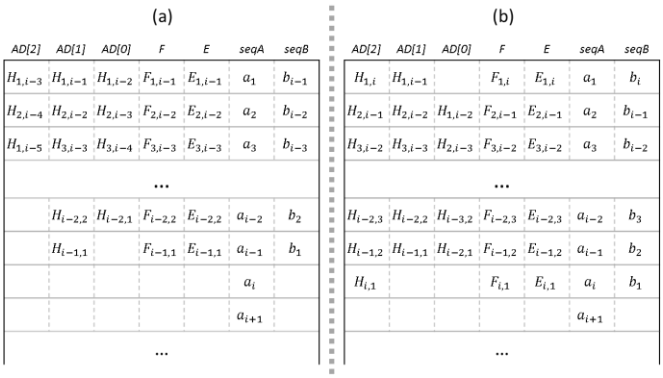

Fig 3. Organization of data within ReCAM corssbar array at the beginning (a) and end (b) of iteration *i* (of section 1 in Fig 2). *AD[2]* contents is being replaced with the new result.

S-W implementation on ReCAM can be divided into three logical sections. In the first section, marked 1 in Fig 2, the most recently scored antidiagonal is longer by one cell than the previous one. During the second section (2 in Fig 2) all antidiagonals are of the same length. In the third section (3 in Fig 2) every new scored antidiagonal is one cell shorter than the previous.

Fig 4 presents the pseudocode of the S-W score finding on ReCAM. Its main loop can be adjusted to implement each of three logical sections presented in Fig 2, as is explained below. The elements of the last three scored antidiagonals (plain-colored cells in Fig 2), retained in *AD[0-2]*, are accessed in a round-robin manner, where the left one (*AD[2]* in Fig 3 and the index *left_AD* in Fig 4) is replaced by the new result. The result of the previous iteration is stored in *AD[middle_AD]* (*AD[1]* in Fig 3).

Fig 3 shows a ReCAM crossbar snapshot, at the beginning (a) and the end (b) of a single iteration of the section 1 of Fig 2. Before calculating the match score, $seqB$ is shifted one row down in order for all to-be-matched base-pairs to reside in the same ReCAM rows (right-most column in Fig 3). *AD[left_AD]* is also shifted one row down for the matching cells to be aligned (*AD[0]* in Fig 3). After calculating the matching score, *AD[left_AD]* is no longer required and is therefore used to store temporary results. Next, the max between the match score and zero is calculated. We note that (ii) in (1) and (2) belong to the same antidiagonal, therefore it is enough to calculate (ii) once and use both for *E* and *F*. After calculating *E* its columns are shifted one row down to have the values of *E* aligned with the appropriate ones in the result vector. In section 1 and 2 of Fig 2, the down-shifted columns require zero-padding of the top-most ReCAM row. Once all four elements of (3) are calculated, the result column is searched for the maximal *H* cell score, and the global max is updated.

The total number of iterations equals the sum of lengths of the two sequences. Each iteration performs 17 instructions. Shift is performed in a single cycle. The number of ReCAM rows affected by instruction is marked by [*]. It increases (decreases) in the 1st (3rd) section and remains constant in the 2nd section. Both 1st and 2nd sections require zero padding at the first row of every down-shifted ReCAM column to initialize the first rows of E, F, H at zero. The 3rd section does not require such padding.

At the beginning of the execution (section 1 in Fig 2), only the top-most ReCAM row is active. Each subsequent iteration activates an additional row until reaching the total of $\min\{m,n\}$ rows. In section 2, the number of active rows remains constant. In section 3, the number of active rows decreases, starting with the top-most row and advancing downward in each iteration. The average number of active rows is: $seqA \cdot seqB/(seqA + seqB)$.

```
SmithWatermanScore(A, n, B, m) {
1   init (temp, AD[2…0][*], F[*], E [*], seqA[*], seqB[*]) ← (0,…,0,A,0)
2   max_score ← 0  //scalar to hold the maximal cell value
3   for i=0 to n+m-1 do {
4     right_AD ← i mod 3; middle_AD ← (i – 1) mod 3;  left_AD ← (i – 2) mod 3
5     seqB[*] ← B[i…1] // Prepare subsequence B for next iteration
6     shift AD[left_AD][*]  1  row down
7     AD[right_AD][*]← AD[left_AD][*] + match(seqA[*], seqB[*])  // (i) in Eq. (3)
      //AD[left_AD] is not needed anymore. Will be used as a temp variable
8     AD[right_AD] ← max{ AD[right_AD][*], 0}                    // (iv) in (3)
9     AD[left_AD][*] ← AD[middle_AD][*] – G_first [*]            // (i) in (1) & (2)
10    temp ← F[*] – G_ext [*]
11    F[*] ← max{ AD[left_AD][*], temp}                          // (ii) in (2)
12    AD[right_AD][*] ← max{ AD[right_AD][*], F[*]}              // (ii) in (3)
13    temp ← E[*] – G_ext [*]
14    E[*] ← max{ AD[left_AD][*], temp }                         // (ii) in (1)
15    shift E[*]  1  row down
16    AD[right_AD][*] ← max{ AD[right_AD][*], E [*]}             // (iii) in (3)
17    max_score ← max{ maxColumnScalar( AD[right_AD][*]), max_score }
18  }
```

Fig 4. Pseudo-code of S-W implementation on ReCAM

4 Simulation

We simulate the ReCAM using a cycle-accurate simulator introduced in [5], employing ReCAM performance and power figures obtained by SPICE simulations. The simulated ReCAM parameters are listed in Table 2. Compare and write energy figures are taken from [5].

TABLE 2
COMPARISON OF RECAM SIMULATED PERFORMANCE VS. THE HIGHEST-PERFORMING MULTI-GPU OF [7]

ReCAM Parameter	Value
Active storage size	8GB
Frequency	500Mhz
Compare energy per bit	1fJ
Write energy per bit	100fJ

Our simulation used sequence data retrieved from the National Center for Biotechnology Information (NCBI),

comparing human (GRCh37) and chimpanzee (panTro4) homologous chromosomes, similar to [7]. To measure the performance of SW, the metric CUPS (Cell Updates per Second) is used. We compare our performance results to those of different S-W platforms identified in Section 1. A multi-GPU implementation presented in [7] reached 11.08 TCUPS using a cluster of 128 compute nodes with a total of 384 Tesla M2090 GPUs. The FPGA-based highest reported performance of S-W is 6.02 TCUPS, obtained by the RIVYERA S6-LX150 platform [3] which uses 128 FPGAs. Highest performance achieved on a Xeon Phi based platform is 0.23 TCUPS, reported by [4] with 4 Xeon Phi accelerators. ReCAM performance is obtained when comparing the chromosome 1, resulting in a total of 57.2 peta score cells. Table 3 summarizes the performance comparison of all the above S-W implementations.

Although the TCUPS values were obtained in different platforms, with different sequences, and cannot be compared directly, they do provide an indication of the potential of each platform.

TABLE 3
SUMMARY OF STATE-OF-THE-ART PERFORMANCE FOR S-W SCORING STEP IN PREVIOUS WORKS AND IN RECAM

Accelerator	FPGA	GPU	Xeon Phi	ReCAM
Performance (TCUPS) / # of PEs	6.02 / 128 [3]	11.08 / 384 [7]	0.23 / 4 [4]	52.68 / 1

Table 4 shows further comparison of ReCAM performance with the highest performing implementation, the multi-GPU cluster in [7]. We present the simulated ReCAM performance for several input sizes, alongside the scoring performance reported in [7].

TABLE 4
COMPARISON OF RECAM SIMULATED PERFORMANCE VS. THE HIGHEST-PERFORMING MULTI-GPU OF [7]

Chr.	Table Size (Peta Cells)	Max. Perf. of [7] (TCUPS)	ReCAM Perf. (TCUPS)
chr1	56.91	-	52.68
chr5	33.04	11.08	37.92
chr8	21.07	10.43	30.78
chr16	8.13	9.7	19.29

The simulated ReCAM power dissipation for aligning chr5 with the parameters presented in Table 2 is 6,600W (to sustain this power figure, the ReCAM is divided into 32 separate 256MB ICs). The multi-GPU implementation using 384 Tesla M2090 GPUs dissipates roughly 80kW. To conclude: ReCAM solution provides 12x lower power dissipation and 3.4x faster execution compared to GPU based platform.

5 CONCLUSIONS AND FUTURE WORK

We have seen the potential benefits of using a resistive CAM to calculate the S-W score of two DNA sequences. By exploiting ReCAM's content addressability, the presented implementation can be extended to align two protein sequences, with a modest degradation in performance. Such implementation can store the entire substitution matrix (e.g., BLOSUM[8]) as truth-table to be accessed when calculating the match score of every pair of proteins.

Due to ReCAM's highly parallel architecture, the presented implementation of S-W scoring step can be extended to align complete organism sequences. For example, the alignment discussed in Section 4 can be extended so that each section of the S-W presented in Fig 2 will be executed for all chromosomes in parallel. The runtime of each section will be determined by the pair containing the longest chromosome (execution time is $\max\{m, n\}$). Once a pair has completed the execution, the next section can start executing for all chromosomes in parallel. This implementation requires keeping record of rows active in each instruction for every chromosome, which may result in a performance overhead of 15% compared with aligning only two sequences.

This work can be extended to implement the second part of S-W algorithm, i.e., finding the alignment. We aspire to maintain the linear time and space complexities. Such solution has been suggested by Myers and Miller [9] and is also used by [7].

ACKNOWLEDGMENT

Present work was partially funded by the Intel Collaborative Research Institute for Computational Intelligence.

REFERENCES

[1] Smith, Temple F., and Michael S. Waterman. "Identification of common molecular subsequences." Journal of molecular biology 147.1 pp. 195-197, 1981.

[2] Gotoh, Osamu. "An improved algorithm for matching biological sequences."Journal of molecular biology 162.3 (1982): 705-708.

[3] L.Wienbrandt. The FPGA-based High-Performance Computer RIVYERA for Applications in Bioinformatics. In *Language, Life, Limits: 10th CiE*, pages 383-392. Springer 2014.

[4] Y. Liu and B. Schmidt. 2014b. SWAPHI: Smith-Waterman protein database search on Xeon Phi coprocessors. In *IEEE ASAP*. 184–185.

[5] L. Yavits et al, "Resistive Associative Processor", IEEE Computer Architecture Letters, 14(2), pp. 148 – 151, 2015.

[6] E.F.D.O. Sandes, A. Boukerche, and A.C.M.A.D. Melo, Parallel Optimal Pairwise Biological Sequence Comparison: Algorithms, Platforms, and Classification. *ACM Computing Surveys (CSUR), 48*(4), p.63, 2016.

[7] Sandes, E. F. O., et al. "CUDAlign 4.0: Incremental Speculative Traceback for Exact Chromosome-Wide Alignment in GPU Clusters."

[8] S. Henikoff, and J. G. Henikoff. "Amino acid substitution matrices from protein blocks." *Proceedings of the National Academy of Sciences* 89, no. 22, pp. 10915-10919, 1992.

[9] E. W. Myers and W. Miller. Optimal alignments in linear space. Computer Applications in the Biosciences, 4(1):11–17, 1988.

[10] Huai, Yiming. "Spin-transfer torque MRAM (STT-MRAM): Challenges and prospects." *AAPPS Bulletin* 18.6 (2008): 33-40.