A Resistive CAM Processing-in-Storage Architecture for DNA Sequence Alignment

A novel processing-in-storage (PRinS) architecture based on Resistive CAM (ReCAM) is described and proposed for Smith-Waterman (S-W) sequence alignment. The ReCAM PRinS massively parallel compare operation finds matching base pairs in a fixed number of cycles, regardless of sequence length. The ReCAM PRinS S-W algorithm is simulated and compared to FPGA, Xeon Phi, and GPU-based implementations, showing at least 4.7 times higher throughput and at least 15 times lower power dissipation.

Roman Kaplan,
Leonid Yavits,
Ran Ginosar,
Uri Weiser
Technion—Israel Institute of Technology

With the approaching end of Moore's law, academia and industry have an increased interest in non–von Neumann computing paradigms. One such example is content-addressable associative processing.[1] CMOS-based content addressable memories (CAMs) require large bit-cells, limiting chip capacity and forcing most data-intensive applications to employ less functional RAM. Novel resistive materials dissipate little heat and allow for 3D stacking. Combined with CMOS, resistive materials can be used in a CAM bit-cell, resulting in a small cell area, low leakage power, and increased overall chip area efficiency.

This article presents a novel resistive CAM-based storage system architecture with a processing-in-storage (PRinS) computing paradigm. The system, called Resistive CAM or ReCAM, is an in-storage accelerator that may scale up to hundreds of millions of processing units (PUs) spread across multiple silicon dies, each containing several million PUs. In addition, the system performs the computations in situ, resulting in increased performance and reduced energy consumption on massively parallel workloads.

The first part of this article presents ReCAM's PRinS system architecture and describes its main components. The second part demonstrates ReCAM's PRinS implementation of

Published by the IEEE Computer Society

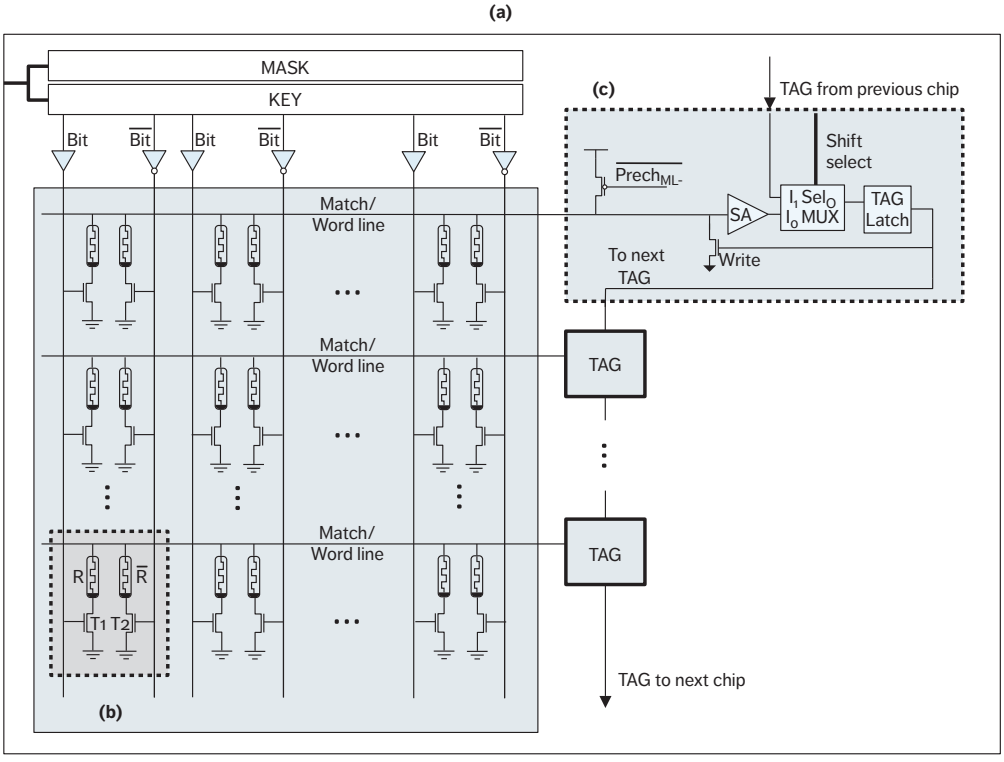

Figure 1. A single resistive CAM (ReCAM) crossbar IC. (a) Two-transistor, two-resistor (2T2R) ReCAM bitcell. (b) TAG logic.

a key algorithm in bioinformatics, the Smith-Waterman (S-W) DNA local sequence alignment. We also present simulation results and compare the performance of ReCAM PRinS with four state-of-the-art large-scale accelerator systems. We show that an in-storage implementation of S-W on ReCAM can achieve on average 4.7 times higher throughput while dissipating 15 times lower power compared with a 384-GPU implementation, the largest S-W implementation found in the literature.

ReCAM-Based Processing-in-Storage

Resistive memories store information by modulating the resistance of nanoscale storage elements. They are nonvolatile and free of leakage power, and they emerge as long-term potential alternatives to charge-based memories, including NAND flash. The metal-oxide resistive RAM (ReRAM), employing one resistive device and possibly also one transistor (1R1T) per bitcell, is considered a potential technology to replace next-generation nonvolatile memories. Its main features are high reliability and fast access speed. Researchers have developed a test-chip of a 32-Gbyte device with two ReRAM-based memory layers and a CMOS logic layer underneath,[2] demonstrating design techniques to achieve a high-density functional chip.

ReCAM Crossbar Array

While ReRAM may employ one transistor and one memristor (1T1R) cells, ReCAM uses 2T2R cells, following work by Tohru Miwa and colleagues[3] and Shoun Matsunaga and colleagues.[4] Figure 1 shows the resistive CAM crossbar. A bitcell, shown in Figure 1a, comprises two transistors and two resistive elements (2T2R). The KEY register contains a data word to be written or compared against. The MASK register defines the active columns for write and read operations, enabling bit selectivity. The TAG register (see Figure 1b) marks the rows that are matched by the compare operation and may be affected by a parallel write. The TAG register enables the chaining of multiple ReCAM integrated circuits (ICs).

Table 1. Operations used in Smith-Waterman (S-W) score calculation. ("DNA base-pair match" operates on 2 bits; all other rows of data operate on 32 bits.)

Instruction	Cycles
Shift down one row	96
B ← A + B	256
C ← A + B	512
Row-wise maximum (A, B)	64
Max scalar (A)	64
DNA base-pair match	10

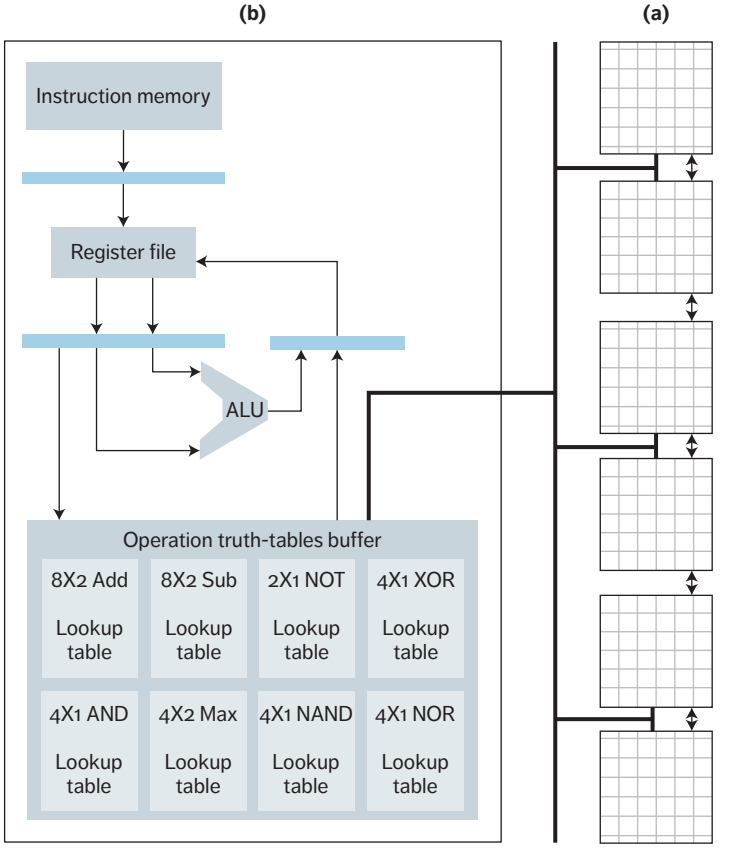

Figure 2. ReCAM-based storage system. (a) Multiple separate ReCAM ICs connected by a reduction network. (b) Microcontroller.

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, and additional capabilities, such as read and reduction operations, are included.[5]

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 the operations used in S-W implementation and the number of cycles required per each one. Shifting down a consecutive block of rows by one row position requires three cycles per bit. First, "compare-to-1" copies the source bit-column of all rows into the TAG. Second, "shift" moves the TAG vector down by setting the shift-select line (see Figure 1c). Third, "write-1" copies the shifted TAG to the same bit-column. Shifting 32-bit numbers thus requires 96 cycles. Addition (in-place or not) is performed in a bit-serial manner using a truth table approach[5] (32 bits times 8 truth-table rows times 2 for compare and write equals 512 cycles). "Row-wise maximum" compares in parallel two 32-bit numbers in each row. "Max scalar" tags all rows that contain the maximal value in the selected element. Additional operations, such as parallel and reduction arithmetic, may be required for other algorithms.

ReCAM PRinS System Architecture

Conceptually, the ReCAM comprises hundreds of millions of rows, each serving as a computational unit. The entire array may be divided into multiple smaller ICs (due to power per die restrictions; see Figure 2a), which use the same MASK and KEY. A row is fully contained within an IC. All ICs are daisy chained for shift and max scalar operations. Therefore, in practice, operations listed in Table 1 take several more cycles to enable inter-IC shift operations.

The ReCAM processing-in-storage system uses a microcontroller (see Figure 2b), similar to work by Qing Gao.[6] It issues instructions, sets the key and mask registers, handles control sequences, and executes read requests. In addition, the microcontroller holds the associative instructions buffer, containing the truth tables

for associative instructions. Because instructions are performed bit-serially, these tables are typically small, as Figure 2b shows. Part of the associative instructions buffer is user-programmable with custom instructions, such as the match operation in Table 1.

ReCAM PRinS Smith-Waterman Implementation

This section presents the scoring step of the S-W dynamic programming algorithm. It then explains how to exploit the inherent parallelism of S-W and details the algorithm implementation on ReCAM.

Smith-Waterman Algorithm

S-W identifies the optimal local alignment of two sequences by computing a 2D scoring matrix, H.[7] Each $H_{i,j}$ element is calculated according to Equation (3). $\sigma(a_i, b_i)$ is the match score between the base pairs in row i (ith element of sequence A) and column j (jth element of sequence B). Matching base pairs score positively (for example, +2), whereas mismatching results in a negative score (for example, –1). The optimal alignment score between two sequences is the highest score in the matrix H.

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

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

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

The alignment may contain gaps in both sequences that are penalized in the score calculation (by negative scores). According to the affine gap model,[8] opening a gap is harder than extending it; therefore, the penalty for opening a gap is larger. The affine penalty scheme is calculated with two additional matrices, E and F (see Equations 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.

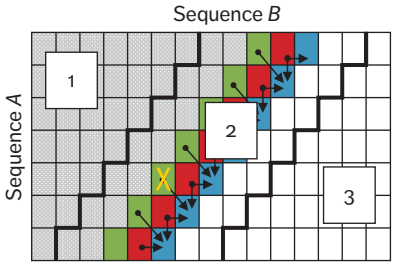

Figure 3. Parallel S-W scoring snapshot of matrix H. Thick borders separate ReCAM implementation to three logical sections. The grayed-out cell scores have already been computed. The three plain-colored antidiagonals are stored in ReCAM. The green and red antidiagonals are used to compute the score of the blue antidiagonal. White-colored cell scores are yet to be computed. The cell marked with "X" contains the global maximum score.

Filling the scoring matrix H is the computationally intensive part of S-W. In a sequential implementation of the algorithm, cell filling is performed in either row- or column-wise order. A parallel implementation allows all independent cells to be computed in the same iteration. Such cells reside on the same antidiagonal. The matrix is filled along the main diagonal, shown in Figure 3.

The sequential time complexity is $O(nm)$, in which n and m are the respective lengths of the sequences. Parallel time complexity on p parallel processing units is $O(nm/p)$. In ReCAM, the processing unit is a memory row. Since ReCAM may comprise hundreds of millions of rows, unlike 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 $O(\max\{n, m\})$.

Smith-Waterman Algorithm in ReCAM PRinS

In this work, we focus on finding the maximal alignment score. Therefore, storing the entire matrix in memory is not needed. This is in contrast to the full algorithm, which also contains the traceback part for finding the alignment.[7]

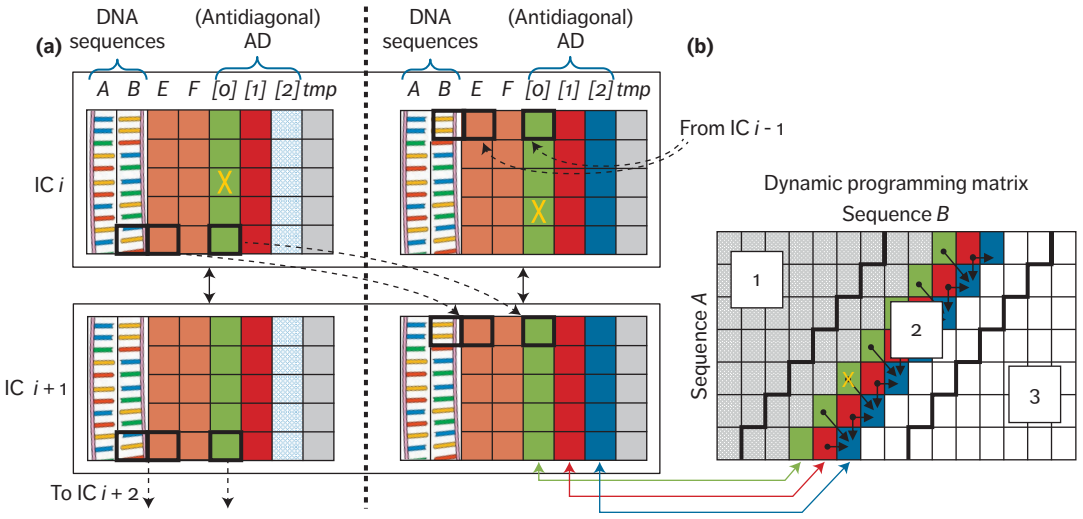

Figure 4. Organization of data in the ReCAM crossbar array at the (a) beginning and (b) end of a single iteration of Figure 5. The contents of *AD*[2] are being replaced with the new result. Bottom rows in a crossbar IC are daisy-chained to the next IC in a shift instruction.

Four antidiagonals are required to compute a new antidiagonal of *H*: two of *H* (see Equations 1–3 and the green and red parts of Figure 3), one of *E* (see Equation 1), and one of *F* (see Equation 2). Thus, five matrix antidiagonals are stored in the ReCAM in each iteration (*E*, *F*, *AD*[0], *AD*[1], and *AD*[2] in Figures 3 and 4). A *tmp* field stores partial results. The overall space complexity required for executing the algorithm is therefore $O(\min\{n, m\})$.

Each of the five antidiagonals is mapped onto a 32-bit column in the ReCAM. Every ReCAM row retains one element of the vectors *seqA*, *SeqB*, *E*, *F*, *AD*[0], *AD*[1], *AD*[2], and *tmp*. The first two numbers are the 2-bit elements of sequences *A* and *B*, respectively.

S-W algorithm implementation on ReCAM can be divided into three logical sections. The first section, marked 1 in Figure 3, starts at the top-left cell and covers a triangle with each edge of length $\min\{m, n\}$ cells. In it, the most recently scored antidiagonal is longer by one cell than the previous one. The third section (marked 3 in Figure 3) is of a similar shape and the same dimensions, ending at the bottom-right cell. In it, every new scored antidiagonal is one cell shorter than the previous one. The second section (marked 2 in Figure 3) is a parallelogram between the first and third sections. In it, all antidiagonals are of the same length.

Figure 5 presents the pseudocode of the S-W score finding on ReCAM. Three ReCAM columns are required to store last two scored antidiagonals of *H* and the presently computed one, notated as *AD*[2]–*AD*[0] in code. During execution, these columns are cyclically buffered; the oldest scores are replaced by the new ones (see line 4 in Figure 5). Three additional 32-bit columns are used to store antidiagonals of *E*, *F*, and *tmp*.

Figure 4 shows a ReCAM crossbar snapshot at the beginning (Figure 4a) and end (Figure 4b) of a single iteration of Figure 5. At line 5, seqB is shifted one ReCAM row down in order for all to-be-matched base pairs to reside in the same ReCAM rows (second column from the left in Figure 4). *AD*[*left_AD*] is also shifted one row down for the matching cells to be aligned (see line 6 in Figure 5, and *AD*[0] in Figure 4). After calculating the matching score (line 7), *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 (see line 8). Note that (ii) in Equations 1 and 2) belongs to the same antidiagonal; therefore, it is enough to calculate (ii) once for both *E* and *F* (see line 9). Lines 10 to 16 compute Equations 1 to 3. In line 15, after *E* is calculated, its columns are shifted one row down to have the values of *E* aligned with the appropriate

```
SmithWatermanScore(A, n, B, m) {
1    init (tmp, 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      // (ii) in (1)&(2)
10       tmp ← F[*] − G_ext
11       F[*] ← max{AD[left_AD][*], temp}          // (i) 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][*], tmp}           // (i) in (1)
15       shift E[*] 1 row down
16       AD[right_AD][*] ← max{AD[right_AD][*], E[*]}     // (iii) in (3)
17       max_score ← max{maxScalar(AD[right_AD][*]), max_score }      //scalar      inst.
       }
```

Figure 5. Pseudocode of the S-W algorithm on ReCAM.

ones in *AD*[*right_AD*]. In sections 1 and 2 of Figure 3, the down-shifted columns require zero-padding of the top-most ReCAM row (not shown in Figure 5). At the end (line 17), the global max is updated with the maximal *H* cell score. After a specific base pair of seqB has been aligned with all seqA base pairs, it is cyclically shifted to its original position (not shown in the figure).

The total number of iterations is the sum of lengths of the two sequences. Each iteration performs 17 instructions. The number of ReCAM rows affected by an instruction is marked by [*]. That number increases (decreases) in section 1 (3) and remains constant in section 2 (the minimum of the lengths of the two sequences).

At the beginning of execution (the first cell of section 1), only the topmost ReCAM row is active. Each subsequent iteration activates an additional row until reaching min{*m*, *n*} active rows in an iteration. During section 2, the number of active rows remains constant. During section 3, the number of active rows decreases, starting with the topmost row to be inactive and subtracting one active row in each iteration. The average number of active rows is $(n \times m)/(n + m)$.

Simulation

The S-W algorithm is simulated on ReCAM using the cycle-accurate simulator introduced in our previous work,[5] employing ReCAM performance and power figures obtained by Spice

Table 2. Simulated ReCAM parameters.

ReCAM parameter	Value
Active storage size	8 Gbytes
Frequency	1 GHz
Power per integrated circuit	200 W
No. of integrated circuits	32

simulations. The simulated ReCAM parameters are listed in Table 2. The power figure was taken from our earlier work.[5]

The simulation employs sequence data retrieved from the National Center for Biotechnology Information (NCBI), comparing human (GRCh37) and chimpanzee (panTro4) homologous chromosomes, similar to work by Edans Flavius de Oliveira Sandes and colleagues.[9] The CUPS (cell updates per second) metric is used to measure S-W performance. Results are compared to other works in Table 3. A multi-GPU implementation[9] reached 11.1 TCUPS on a cluster of 128 computing nodes with a total of 384 Tesla M2090 GPUs. An FPGA implementation of S-W reaches 6.0 TCUPS on the RIVYERA platform[10] having 128 Xilinx Spartan-6 LX150 FPGAs. A four-Xeon-Phi implementation achieves 0.23 TCUPS.[11] On ReCAM, we demonstrate 53 TCUPS, computing a total of 57.2×10^{12} scores. The table also shows computed GCUPS/W ratios; ReCAM is close to twice better than the FPGA solution and 80 times better than the GPU system.

The simulated ReCAM PRinS power dissipation is 6.6 kW. The optimal setting to sustain this power figure with minimal performance overhead is dividing the ReCAM into 32 separate ICs, each with 256 Mbytes and 8 million rows. The multi-GPU implementation using 384 Tesla M2090 GPUs and 256 Intel Xeon E5-2670 CPUs might dissipate 100 kW, 15 times higher power. Table 4 shows additional comparisons of ReCAM and the multi-GPU cluster,[9] demonstrating up to 3.7 times faster execution and 4.7 times higher throughput on ReCAM.

Scalability of ReCAM PRinS Sequence Alignment

Consider the case of 1 billion organism sequences. Each sequence is hundreds of millions of base pairs in size, on average. Analyzing the contents of these sequences can lead to discoveries such as identification of disease-carrying genes, determination of evolutionary events, and identification of regions that can be used to silence genes.[12] Performing an all-to-all alignment of the entire sequence database in a conventional datacenter is not scalable. Every two sequences will require fetching to the main memory, close to the processing unit (CPU or accelerator). The high communication cost between separate storage units causes the system to be I/O-bound in an all-to-all type of computation.

On the other hand, ReCAM-based storage is more scalable. Its inherent parallelism allows for scalability when adding more ICs, increasing storage capacity at no performance cost. The computing capability is linearly scalable in the number of ICs. Therefore, performing an all-to-all alignment of large sets, such as 1 billion sequences, does not require external communication for the ReCAM, in contrast to datacenter-scale storage. A more effective solution, in terms of performance and energy, is using ReCAM as primary storage when large alignment operations are constantly performed.

This article explores PRinS implementation for the scoring step of the Smith-Waterman DNA sequence alignment algorithm on a novel solid-state storage device, based on ReCAM. ReCAM enables storage with in-situ processing capabilities. It can contain hundreds of millions of data rows, each serving as a processing unit. The proposed ReCAM PRinS system is divided into multiple ICs to accommodate power density constraints.

This research can be extended in several ways. First, the ReCAM PRinS S-W scoring algorithm can be extended to provide complete DNA sequence alignment (that is, both matrix-fill and traceback steps), maintaining the same performance and power advantages. Second, the ReCAM PRinS algorithm can be applied in parallel to complete DNA sequences of two organisms, and not only to specific chromosomes.

Table 3. State-of-the-art performance for S-W scoring step in previous works and in ReCAM.

Accelerator	Xeon Phi[11]	FPGA[10]	GPU[9]	ReCAM
Performance (tera cell updates per second, or TCUPS)	0.23	6.0	11.1	53
No. of ICs	4	128	384	32
Power (kW)	0.8	1.3	100.0	6.6
GigaCUPS (GCUPS) per W	0.3	4.7	0.1	8.0

Table 4. ReCAM and multi-GPU[9] performance.

Chromosome	Table size (10^{12} cells)	Maximum performance[9] (TCUPS)	ReCAM performance (TCUPS)
chr1	57.2	—	53
chr5	33.5	11.1	41.8
chr8	21.1	10.4	30.8
chr16	8.1	9.7	19.3

Third, the proposed S-W ReCAM PRinS algorithm can be applied to the wider challenge of aligning protein sequences. That problem is more challenging than DNA alignment because the required substitution matrix is typically 20 × 20 rather than 2 × 2, and the ReCAM could store the entire substitution matrix, resulting in efficient parallel processing.

ReCAM PRinS architecture, capable of general-purpose associative processing, can also be applied to other challenging problems, such as machine learning and graph algorithms.

Acknowledgments

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

References

1. C.C. Foster, *Content Addressable Parallel Processors*, John Wiley & Sons, 1976.
2. T.Y. Liu et al., "A 130.7-mm2 2-Layer 32-Gb ReRAM Memory Device in 24-nm Technology," *IEEE J. Solid-State Circuits*, vol. 49, no. 1, 2014, pp. 140–153.
3. T. Miwa et al., "A 1 Mb 5-Transistor/Bit Non-volatile CAM Based on Flash-Memory Technologies," *Proc. IEEE Int'l Solid-State Circuits Conf.*, 1996, pp. 40–41.
4. S. Matsunaga et al., "Fully Parallel 6T-2MTJ Nonvolatile TCAM with Single-Transistor-Based Self Match-Line Discharge Control," *Proc. Symp. VLSI Circuits*, 2011, pp. 298–299.
5. L. Yavits et al., "Resistive Associative Processor," *IEEE Computer Architecture Letters*, vol. 14, no. 2, 2015, pp. 148–151.
6. G. Qing et al., "AC-DIMM: Associative Computing with STT-MRAM," *Proc. 40th Ann. Int'l Symp. Computer Architecture*, 2013, pp. 189–200.

7. T.F. Smith and M.S. Waterman, "Identification of Common Molecular Subsequences," *J. Molecular Biology*, vol. 147, no. 1, 1981, pp. 195–197.
8. O. Gotoh, "An Improved Algorithm for Matching Biological Sequences," *J. Molecular Biology*, vol. 162, vol. 3, 1982, pp. 705–708.
9. E.F. de Oliveira Sandes et al., "CUDAlign 4.0: Incremental Speculative Traceback for Exact Chromosome-Wide Alignment in GPU Clusters," *IEEE Trans. Parallel and Distributed Systems*, vol. 27, no. 10, 2016, pp. 2838–2850.
10. L. Wienbrandt. "The FPGA-Based High-Performance Computer RIVYERA for Applications in Bioinformatics," *Proc. Conf. Computability in Europe*, 2014, pp. 383–392.
11. Y. Liu and B. Schmidt, "SWAPHI: Smith-Waterman Protein Database Search on Xeon Phi Coprocessors," *Proc. IEEE 25th Int'l Conf. Application-Specific Systems, Architectures and Processors*, 2014, pp. 184–185.
12. E.F. De Oliveira Sandes, A. Boukerche, and A.C.M. Alves De Melo, "Parallel Optimal Pairwise Biological Sequence Comparison: Algorithms, Platforms, and Classification," *ACM Computing Surveys*, vol. 48, no. 4, 2016, article 63.

Roman Kaplan is a PhD candidate in the Department of Electrical Engineering at the Technion—Israel Institute of Technology. His research interests include in- and near-data processing architectures, many-core computers, and parallel algorithms. Kaplan received an MSc in electrical engineering from the Technion. Contact him at romankap@gmail.com.

Leonid Yavits is a postdoc fellow in electrical engineering at the Technion—Israel Institute of Technology and a cofounder of both VisionTech and Horizon Semiconductors. His research interests include non–von Neumann computer architectures and processing in memory. Yavits received a PhD in electrical engineering from the Technion. Contact him at leonid.yavits@nububbles.com.

Ran Ginosar is a professor in the Department of Electrical Engineering and serves as head of the VLSI Systems Research Center at the Technion—Israel Institute of Technology. His research interests include VLSI architecture, many-core computers, asynchronous logic and synchronization, networks on chip, and biologic implant chips. Ginosar received a PhD in electrical and computer engineering from Princeton University. He has co-founded several companies in various areas of VLSI systems. Contact him at ran@ee.technion.ac.il.

Uri Weiser is a professor emeritus in the Department of Electrical Engineering at the Technion—Israel Institute of Technology and is on the advisory board of numerous startups. Weiser received a PhD in computer science from the University of Utah, Salt Lake City. He is an Intel Fellow, IEEE Fellow, and ACM Fellow. He has received the IEEE/ACM Eckert-Mauchly Award. Contact him at uri.weiser@ee.technion.ac.il.