PARC: A Processing-in-CAM Architecture for Genomic Long Read Pairwise Alignment using ReRAM

Fan Chen, Linghao Song, Hai "Helen" Li, Yiran Chen
Department of Electrical and Computer Engineering, Duke University, Durham, NC, USA
{fan.chen, linghao.song, hai.li, yiran.chen}@duke.edu

Abstract—Technological advances in long read sequences have greatly facilitated the development of genomics. However, managing and analyzing the raw genomic data that outpaces Moore's Law requires extremely high computational efficiency. On the one hand, existing software solutions can take hundreds of CPU hours to complete human genome alignment. On the other hand, the recently proposed hardware platforms achieve low processing throughput with significant overhead. In this paper, we propose PARC, an Processing-in-Memory architecture for long read pairwise alignment leveraging emerging resistive CAM (content-addressable memory) to accelerate the bottleneck *chaining* step in DNA alignment. *Chaining* takes 2-tuple *anchors* as inputs and identifies a set of correlated *anchors* as potential alignment candidates. Unlike traditional main memory which organizes relational data structure in a linear address space, PARC stores tuples in two neighboring crossbar arrays with shared row decoder such that column-wise in-memory computational operations and row-wise memory accesses can be performed in-situ in a symmetric crossbar structure. Compared to both software tools and state-of-the-art accelerators, PARC shows significant improvement in alignment throughput and energy efficiency, thanks to the in-site computation capability and optimized data mapping.

I. Introduction

Long-read sequencing, also known as third-generation sequencing technologies [1], [2], [3] have demonstrated their potential in revolutionizing the field of genomics and beyond, spanning over precision medicine [4], investigation of infectious disease outbreaks [5], global food security [6], and many other applications. It produces longer *reads* of contiguous DNA base pairs (i.e., >10,000 *bps*) compared to previous sequencing technologies, which greatly improves the quality of genome assembly and significantly benefits other genetic researches [7]. The state-of-the-art technology can even offer portable, real-time sequencing via a USB-sized sequencer [8]. Nevertheless, managing and analyzing the obtained raw genomic data with an exponentially growing size, also known as Moore's Law in molecular biology [9], imposes new computational challenges to the processing speed and efficiency of existing computing platforms.

In general, large-scale genomic *reads* can be analyzed by read mapping or *de novo* assembly based on the characteristic of the underlying data and the availability of reference genomes [10]. Although different approaches have distinct strategies and thus provide different insights, the key step of these methods is to examine the similarity between pairs of sequences via pairwise alignment. In generally, most genome aligners follow a typical *seed-chain-align* [11] procedure, among which *chain* phase consumes >70% of the total computation time, which thus is the focus of this work.

The *chain* algorithm is an one-dimensional dynamic programming model, where each input element is compared to N previous elements to determine the best predecessor. This computation procedure has three notable features: 1) it is a memory-intensive search process; 2) the execution of *chain* is dominated by simple subtraction and comparison operations; 3) it requires little support for floating-point computation. Therefore, we identify that ReRAM-based associative computing is well suited for the acceleration of *chain* due to its high density, low power consumption, and the ability to perform parallel in-situ subtraction/comparisons. In this work, we present PARC, a Processing-in-Memory (PIM) architecture that utilizes configurable ReRAM-based content-addressable memory (CAM) to perform in-situ logic operations and conventional memory access. On top of this architecture, We also propose an efficient data mapping method and a pipelined processing scheme. In summary, we make the following contributions:

- To support read, write and computation functionalities, we exploit the symmetry of the crossbar structure, which can naturally enable dual-addressing memory architecture to support both row-wise in-situ computation and column-wise memory accesses for *chaining* workloads.
- We share the peripheral circuitry including word-line drivers and sense amplifiers across data arrays, thereby minimizing the area overhead.
- We present an efficient data mapping and management scheme for relational data structured (i.e., tuple). Based on the mapping scheme, computations are performed within the ReRAM array in a word-parallel, bit-serial fashion.
- We compare the performance of PARC against both software tool and state-of-the-art hardware solutions. Results show that our design achieves significant performance improvement and energy reduction.

This paper is organized as follows: Section II introduces the background and the motivation; Section III proposes the PARC architecture; Section IV describes the strategy of mapping *chain* phase onto the proposed architecture; Section V present the evaluation setup for PARC; Section VI discuss our experimental results. We conclude ths work in Section VII.

978-1-7281-4123-7/20/$31.00

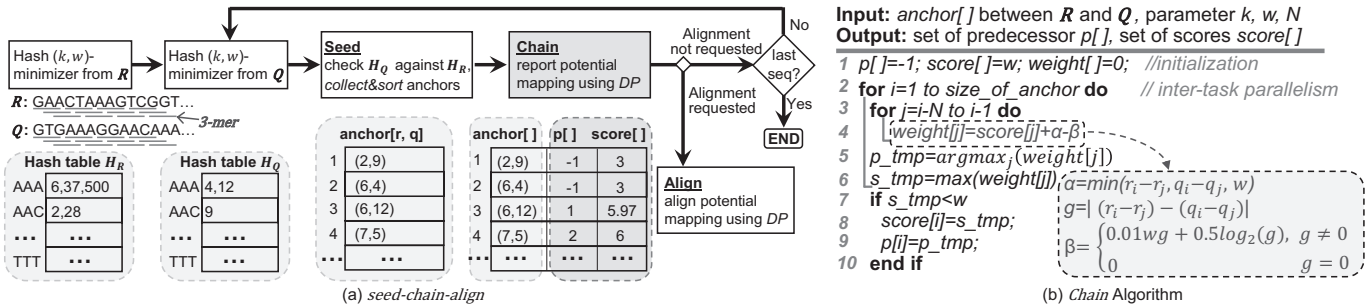

Fig. 1. Basics of pairwise alignment. (a) *seed-chain-align*. Sequence pairs are first *seeded* into hash tables and then *chains* are identified as mapping candidates. A SW-based *align* step is selectively applied if requested; (b) *Chain* algorithm. Each *anchor* is compared against N previous *anchors* to identify its predecessor with the maximum score.

II. Background, Related Work and Motivation

This section introduces the background knowledge that facilitates our design. The basics of *seed-chain-align* pairwise alignment procedure is first described, followed by the basics of ReRAM and its application in in-memory computing. At last, we review related works and highlight our motivation.

A. Pairwise Alignment

Problem statement. Pairwise alignment finds the optimal alignment of two genomic sequences $X = x_1x_2\ldots x_m$ and $Y = y_1y_2\ldots y_n$, where the x_i and y_i are chosen from a finite alphabet. Specifically, for protein sequences the alphabet is 20 amino acids; for DNA sequences of interest in this work, the alphabet is 4 nucleotide bases $\Sigma = \{A, C, G, T\}$. The alignment score is defined in the form of a list of "columns" types including (1) *match*, (2) *mismatch*, and (3) *gap* (*insertion* or *deletion*).

Figure 2 illustrate one possible alignment for sequences $X = ACGATCGGAT$ and $Y = GCTCGGTAT$. In this example, we reward matching bases with score +10, penalize mismatching bases and gaps (denoted by '-') with -4 and -5, respectively. Therefore, the final score of the alignment is +51. Clearly, different alignments between two sequences yield different scores. The goal of pairwise alignment is to identify the alignment between sequence pairs which maxmize the score.

Alignment algorithm. Sequence alignments typicall use the two-phase Smith-Waterman (SW) algorithm [12] which first fills a $m{\times}n$ (m, n respectively represent the length of two sequences) matrix using Dynamic Programming (DP) to find the maximal alignment score, followed by a traceback phase which reconstruct the optimal local alignment by reversing the traceback pointers starting from the cell with highest-score. Although the SW algorithm guarantees an best local alignment between two sequences, its $O(mn)$ time and $O(mn)$ spatial complexity limit its use in genomic analysis, since in practice it is often necessary to process millions of sequences simultaneously. Instead, BLAST [13], [14] is used as a heuristic of the SW algorithm which employs a two-step *seed-and-extend* approach. However, for long reads with high error rates, BLAST gives a high false positive rate that results in excessive computation in the more expensive *extend* stage. To address this issue, minimap [11] *chains* the *seeds* to potential mapping regions with a longer length before

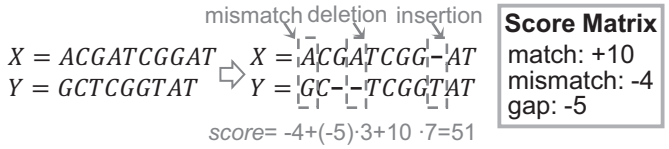

Fig. 2. Pairwise alignment.

extend. The *chain* step itself provides a higher accuracy than many aligners, in which case the *align* (i.e., equivalent to *extend*) phase becomes optional and is performed only when necessary. Minimap's superior performance in processing latency and mapping accuracy with reduced computational efforts make it the *de facto* general-purse pairwise alignment tool.

Seed-chain-align. Figure 1(a) illustrats the *seed-chain-align* procedure in minimap. *Seed* collects *k-mer* minimizers (i.e., a k-long DNA sequence with minimum value in a w-sized surrounding window) of the targeting sequences and indexes them in a hash table by performing a hash function $\Phi : \Sigma^k \rightarrow Z$. Then the query minimizers are taken as *seeds* to find exact matches (also called as *anchor*) against the hash table of the reference sequence. The collected *Anchors* are stored in 2-tuples and enter into a *chain* phase, which performs one-dimensional DP to identifies correlated anchors as *chains*. Finally an optional *align* step applies SW algorithm [12] to extend the *chains* with approximate matches.

As *chain* step is the focus of this paper, we show the detailed algorithm in Figure 1(b). Essentially, each *anchor* is compared against N previous anchors to identify its predecessor with the highest score. The grey block highlights the key operations involved in the *chain* step, which include additions, subtractions and simple scaling. Table I summarizes the notations that are used for explanation of genome pairwise alignment. Due to page limit, we do not give the full explanation of those parameters or the *chain* algorithm. We refer interested reader to [11], [15].

TABLE I
Notations for explanation of genome pairwise alignment.

Symbols	Description
$\Sigma = \{A, C, G, T\}$	the alphabet of nucleotides
$s = a_1a_2\ldots a_n$	a DNA sequence, $a_i \in \Sigma$
$\lvert s\rvert$	length of sequence s
k-mer	a k-long DNA sequence
s_i^k	a k-mer from s starting at i
$\Phi : \Sigma^k \rightarrow Z$	a k-mer hash function

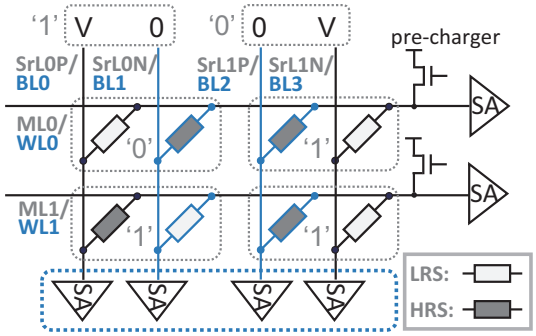

Fig. 3. Morphable ReRAM CAM cell topology.

B. Resistive Crossbar-based Process-in-memory.

ReRAM is an emerging type of non-volatile memory that stores information by changing cell resistances. The resistance of a ReRAM cell can be programmed by applying a current or voltage with proper pulse-width or magnitude. ReRAM has been extensively studies as a promising candidate for future main memory [16]. Very recently, many ReRAM-based in-memory accelerator were proposed for hardware acceleration of machine learning applications [17], [18]. However, the current ReRAM based PIM accelerators mainly leverages ReRAM to perform in-situ matrix-vector multiplications. The computational requirement in genomic analysis is different. It barely require complex computation (i.e. dot-product in machine learning). Instead, it is a memory-intensive search process. Thus we identify ReRAM-based associative computing as a perfect fit. The details of the proposed dual-addressing ReRAM CAM (content addressable memory) is elaborated in Section III.

C. Related Work and Motivation

RADAR [19] proposed to accelerate the *ungapped extension* stage of BLAST [14] using 3D vertical ReRAM. However, BLAST is only suitable for the previous generation of short read alignments. In addition, the 3D vertical implementation of ReRAM is still not a mature technology that can be used in commodity chips. Racelogic [20] and BioSEAL [21] are two accelerators which accelerate dynamic programming in traditional CMOS and ReRAM, respectively. These two works, however, lack the critical features to support backtrace. Darwin [7] is the first accelerator for long reads mapping. It integrates hash table based seeding and SW-based extension. Instead, we are targeting a more efficient and accurate *seed-chain-align* procedure. The work most similar to this work is proposed in [22], which proposed an FPGA-based *chain* accelerator, while we realize *chain* acceleration by leveraging in-situ computation in ReRAM CAMs.

III. PARC

A. ReRAM CAM Cell

A content-addressable memory broadcasts a *search word* onto the *searchlines* (SrLs) against a table of stored data. Each stored word has a *matchline* (ML) that indicates whether it is a match or mismatch. This matching information normally is fed to an encoder to generates further instructions corresponding to the ML that is in the match state. Since the area of a CAM cell affects both the cost-per-bit and the speed/energy in a megabit array, we propose an area-efficient CAM design shown in Figure 3 utilizing high-density ReRAM cells. In the proposed design, each CAM unit consists of two adjacent complementary cells located in the same row. Specifically, a high resistance cell R_H and a low resistance cell R_L pair up to represent logic '1' and a R_L/R_H cell pair represents logic '0'. *Search words* on *SrLs* are also encodes by a pair of complementary voltages: $V/0$ and $0/V$ represents input signal '1' and '0', respectively. Here V denotes the minimum voltage required for a read operation.

Column-wise search. To search a word, MLs are precharged to V. Each bit of the search word is represented with a pair of complementary voltages on the SrLs. Figure 3 shows an example for search "10" in an array which stores "01" and "11". The voltage on the ML of each ReRAM cell is V. Thus, within each pair, only the cell with a 0 voltage on SrL is activated as highlighted in blue. We denote the read current of a cell in high resistance state (HRS) and low resistance state (LRS) as I_H and I_L, respectively. In this case, a sensed current I_L (I_H) at the end of a row indicates a 1-bit match (mismatch). Normally, the fine-grained bit-wise matching information is not needed. For a n-bit search, the threshold current can simply be set as nI_L. Rows with a current lower than nI_L indicates a word mismatch.

Row-wise read/write. For regular read/write, we reserve a group of column sense amplifiers (SAs) at the vertical end of each SrL. As denoted in light blue text in Figure 3, MLs and SrLS serves as regular *wordlines* (WLs) and *bitlines* (WLs) when performing row-wise memory access.

CAM-based addition. The execution of *chain* is dominated by simple subtraction, and comparison such as min, max operation as illustrated in the grey block in Figure 1(b). The calculation of min, max can be realized by a read followed by a parallel 1-bit comparison. with negligible overhead. The bottleneck of computation is the calculation of *anchor* distance $r_i - r_j$ and $q_i - q_j$. We proposed to implement abstraction on CAM without moving huge *anchor* dataset (i.e., averagely 8MB in our profiling) between memory hierarchies. We store *anchors* in CAM in consecutive rows which facilitates *in-situ* calculation

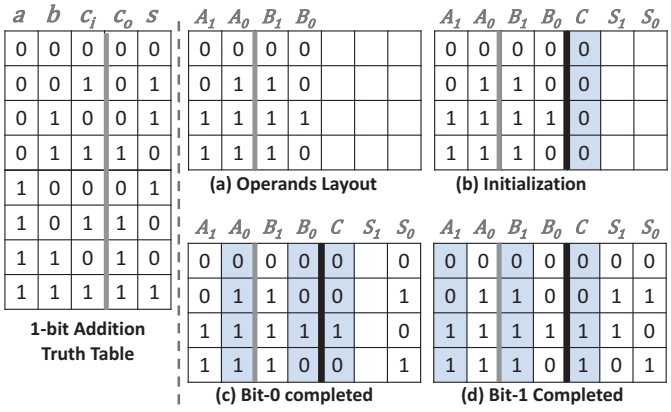

Fig. 4. CAM-based 1-bit addition.

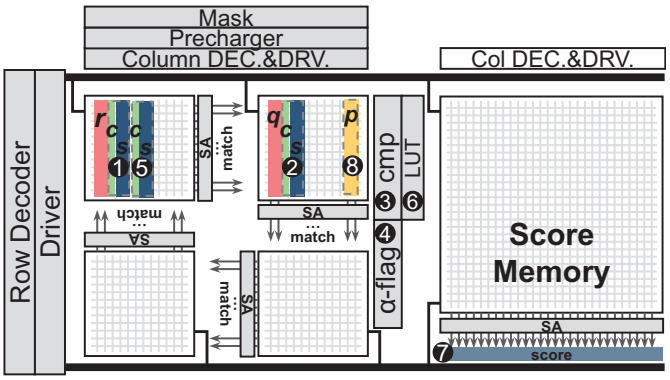

Fig. 5. PE organization in PARC.

of distance. Detailed mapping scheme is illustrated in Section IV. In general, the two elements in each tuple is located in neighboring arrays. The subtraction can be viewed as addition with 2's complement. We structure multiple-bit addition as series of 1-bit addition.

Figure 4 demonstrated the computation flow of CAM-based 1-bit addition. The left part shows 1-bit addition truth table, where a, b, C_i are the two operands and the carry-in bit, c_o, s denotes the carry-out bit and sum. The operands are located in each single row with carry bit initialized as '0'. A controller first sequentially searches all the 8 possible input combinations of a, b, and c, and the corresponding values are written into sum s and carry bit c in the following cycle. For instance, as shown in the right part of Figure 4. The carry-bit column is first initialized as '0'. Then the SrLs of bit 'A_0', 'B_0' and 'C' are activated (denoted in color blue). The first entry in the addition truth table is first searched in the CAM array. In this case we get a match in row 1. So C and S_0 will be written as '0' and '0' accordingly in the next cycle. Following the similar search procedure, we searches every entry in the truth table to perform the addition for every bit to obtain the final results. Note that in each cycle, the carry bit C is updated in-situ to save memory space.

B. PARC Architecture

PE organization. Figure 5 illustrates the organization of one PE structure in PARC, which allows each memory row to be searched, read, and write. PARC leverages ReRAM CAM to perform in-memory execution. Each array contains four dual-accessing ReRAM crossbars and a score memory which stores the floating-point scores for each anchor. The four ReRAM CAM arrays share a row decoder, a wordline driver, a match line pre-charger, a search line mask, and a column decoder and driver. A PE also has a small array of 1-bit comparator to support comparison, a registers to record α, and a look-up table based memory for calculating β in *chain* algorithm, which are respectively denoted as *cmp, α-flag*, and *LUT* in Figure 5.

Sharing of peripheral circuitry. The baseline design requires a set of vertical sense amplifiers connected to MLs and a set of horizontal sense amplifiers connected to BLs, as illustrated in Figure 3. These two sets of SAs introduce significant overhead in terms of both area and computational energy. Hence, we adopt the sense amplifier sharing scheme in [23] by rotating the arrays 90° in turn. As demonstrated in Figure 5, adjacent SAs are used on a search, while the neighboring SAs a litter further are leveraged for conventional memory access. Moreover, wordline drivers are also shared among the crossbar arrays within a PE by global wordline. At a time, only a pair of crossbar array in a PE can be activated and used to perform column-wise search or row-wise access.

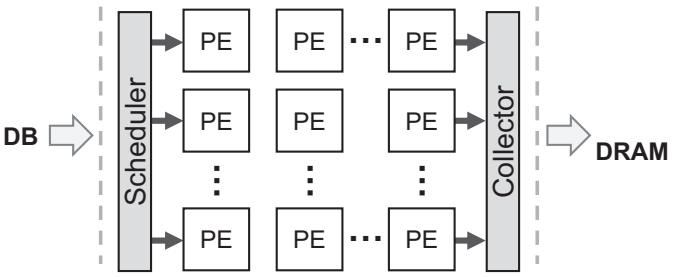

Fig. 6. PARC architecture overview.

PARC architecture overview. Figure 6 demonstrated the PARC architecture. At the top level, it consists a scheduler which fetch *anchor* list from the database and distributes tasks among PEs, an array of PEs which *chain* the *anchors*, and a collector that output the results (i.e., *chains* and scores) to DRAM. Each row of PEs handles the *anchors* from the same reference sequence, such that multiple PE rows perform calculations in an independent manner.

IV. Data Layout and Application Mapping

A. Data Structure and Data Layout

Anchors are organized as 2-tuples (r, q), where r and q are both 32-bit integer [11]. Conventional main memory is a linear address space starting at zero. Normally, all fields of a tuple are stored consecutively. In PARC, tuples are mapped into two neighboring arrays which shares a wordline driver as highlighted in red in Figure 5. Based on this efficient relational data layout, we explain how the execution of *chain* is mapped onto PARC.

B. Application Mapping

Computing distance $r_i - r_j$ and $q_i - q_j$. The 2's complement of current target anchor (r_i, q_i) is applied onto the *SrLs*. In this case, we obtain $r_j - r_i$ and $q_j - q_i$ $(j \in [i - N, i - 1])$ in parallel. The addition operation is performed in a bit-serial fashion as explained in Section III which takes 32 logic cycles. Within each cycle, the sing-bit addition table entries are searched and corresponding results are written into both sum and carry regions, denoted by ❶ and ❷ in Figure 5.

Computing α, g, β. To compute $\alpha = min(r_j - r_i, q_j - q_i, w)$, we read both r_j and q_j out in a bit-serial fashion starting from the most-significant-bit (MSB). With each PE, the dedicated comparator array *cmp* ❸ compares 32 pair of 1-bit value in parallel and store the results in *α-flag* ❹. To compute g, we utilize the CAM-based addition again. Results are store back into the same PE denoted as ❺ in Figure 5. After g is obtained, we look up in *LUT* ❻ to obtain the value of β.

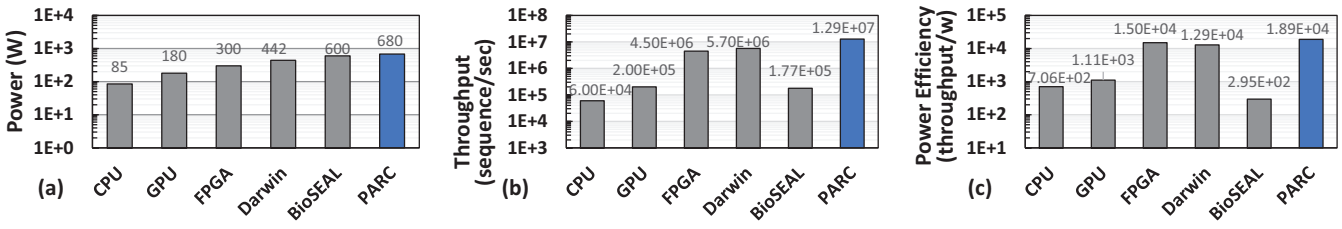

Fig. 7. Comparison results on power, throughput and power efficiency.

Computing *weight*. First $score(j)$ is read out from score memory and temporarily stored in ❼. then we compute $score(j) + \alpha - \beta$. It takes another 32 cycles for 1-bit addition. To find the $s\_tmp = max(weight(j))$ and $p\_tmp = argmax(weight(j))$, *cmp* ❸ compares 32 pairs of 1-bit value in parallel and stores the results back into the same PE denoted as ❽.

Putting It All Together. Following the flow ❶ through ❽. The comparison between current *anchor* and N previous *anchors* are calculated in parallel in one run.

C. System Integration

Similar to [23], PARC is implemented modularly in a DIMM attached to the main memory system. While PARC features non-volatile CAM data arrays, PARC arrays do not include any of the priority index logic, population count logic, the reduction network, or programmable microcontrollers. Instead, PARC tailors the data and control paths to pariwise alignment, which entails minimal logic for *chaining*. At the same time, the peripheral circuitry of the array and the processing units are all pipelined, which significantly improves the processing throughput and makes the design overhead tolerable.

V. Experiment Methodology

Long Reads Dataset. We generate 10Kbp long reads using a PacBio reads simulator [24]. Without loss of generality, We set the genome sequencing error rate as 20% and the coverage as 30×.

Baseline. As comparison baselines, we run minimap [11] on the 8-core Intel E5-2630v3 CPU with 128GB main memory at a maximum frequency of 2.4 GHz and on the NVIDIA Geforce GTX 1080 with 8 GB GDDR5 graphic memory, respectively. Detailed configuration of CPU/GPU baselines are summarized in Table II.

TABLE II
Configuration of CPU/GPU.

CPU	Intel Xeon E5-2630 V3, 8 cores 2.4 GHz, 8×(32+32)KB L1 Cache 8×256KB L2 Cache,20MB L3 Cache 128GB main memory
GPU	NVIDIA Geforce GTX 1080 1607 Base Clock CUDA version 8

TABLE III
Simulated scheme comparison

Name	Year	Description
FPGA [22]	2019	FPGA-based *chain* accelerator
Darwin [7]	2018	CMOS Read Mapping Accelerator
BioSEAL [21]	2019	ReRAM-based DP accelerator

We use NVSIM-CAM [25] to model the latency, power and area of ReRAM arrays. We adopted the ReRAM cell model from [26] with a crossbar size of 256 × 256. The comparators, drivers and other peripheral circuitry are synthesized by Cadence Design Compiler with 32nm PTM process technology. The bus and connections are modeled and estimated using Cadence Virtuoso with TSMC 32nm technology. We also evaluated the PARC architecture using four state-of-the-art alignment accelerators shown in Table III.

VI. Evaluation

We built pairwise alignment for long reads on various platforms to study the performance and throughput. Note that although all of the accelerators under-test support long reads alignment, the underlying algorithms they use are not the same. Specifically, we run minimap [15] on CPU, GPU, FPGA [22], and PARC. Compared with general CPU/GPU platforms, FPGA and PARC are customized hardware accelerators for minimap. For Darwin and BioSEAL, we run Dynamic Programming based alignment algorithm as both design are specialized for DP models. We summarize the detailed die size, off-chip memory capacity, power, read alignment throughput, and power efficiency in Table IV.

As visualized in Figure 7 (a), the power consumption increases from general CPU/GPU platform to customized FPGA and ASIC accelerators. For FPGA and Darwin, the increased power is mainly due to the the intrinsic high computation and memory complexity. For instance, Darwin require huge capacity of off-chip memory to store complex intermediate data, and hash tables. While BioSEAL and PARC both utilize CAM arrays which consume more power than conventional RAMs.

We show the throughput (sequence per seconds) in Figure 7 (b). PARC stands out as the best high-throughput solution. The second best is Darwin. Figure 7 (b) We further compares the power efficiency (throughput per

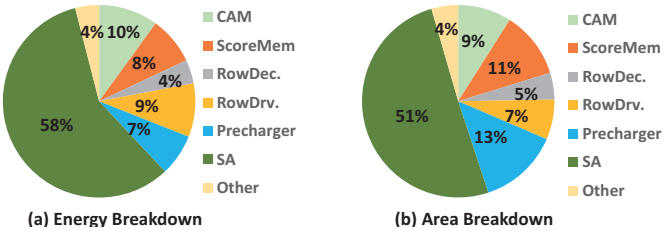

Fig. 8. Energy and area breakdown of PARC.

TABLE IV
THE COMPARISON BETWEEN DIFFERENT PLATFORMS.

	CPU	**GPU**	**FPGA** [22]	**Darwin** [7]	**BioSEAL** [21]	**PARC**
Die Size (mm^2)	2.4K	314	16K	28K	230	87
Off-chip Memory(GB)	128	8	64	128	0	0
Power(W)	85	180	300	442	600	680
Throughput(query/sec)	60K	200K	4.5M	5.7M	177K	12.9M
Throughput/Watt	706	1.11K	15K	12.9K	295	18.9K

watt) in Figure 7 (c), which clearly demonstrates that PARC provides the best solution to accommodate the intensive computation in long reads alignment in terms of throughput per unit power.

Figure 8 also shows the energy and area breakdown in PARC. In general, sense amplifier consumes 58% energy and 51% area. The ReRAM CAM array together with the score memory consume less than 20% energy and area. In our future work, we plan to design more efficient architecture to reduce the overhead of peripheral circuitry.

VII. Conclusions

In conclusion, we propose a ReRAM CAM based acclerator for DNA long read pairwise alignment. We exploit the symmetry of the crossbar structure to enable dual-addressing memory architecture which supports both row-wise *in-situ* computation and column-wise memory accesses. We also present an efficient data mapping and management scheme. Based on the mapping scheme, computations are performed within the ReRAM arrays in a word-parallel, bit-serial fashion. Compared to software tools on CPU and GPU, PARC provides 215× and 64× throughput improvement, respectively. Compared to the state-of-the-art accelerator, Darwin, PARC shows respective 2.3× and 1.5× improvement in terms of throughput and power efficiency.

Acknowledgment

This work was supported in part by NSF CSR-1717885, NSF CCF-1725456, NSF-1822085, and AFRL FA8750-18-2-0121. Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of grant agencies or their contractors.

References

[1] E. E. Schadt, S. Turner, and A. Kasarskis, "A window into third-generation sequencing," *Human Molecular Genetics*, 2010.

[2] A. Rhoads and K. F. Au, "Pacbio sequencing and its applications," *Genomics, Proteomics & Bioinformatics*, 2015.

[3] D. Branton, D. W. Deamer, A. Marziali, H. Bayley, S. A. Benner, T. Butler, M. D. Ventra, S. Garaj, A. Hibbs, X. Huang, S. B. Jovanovich, P. S. Krstic, S. Lindsay, X. S. Ling, C. H. Mastrangelo, A. Meller, J. S. Oliver, Y. V. Pershin, J. M. Ramsey, R. Riehn, G. V. Soni, V. Tabard-Cossa, M. Wanunu, M. Wiggin, and J. A. Schloss, "The potential and challenges of nanopore sequencing," *Nature Biotechnology*, 2008.

[4] N. Lauran, "Obama proposes 'precision medicine' to end one-size-fits-all," 2015.

[5] S. K. Gire, A. Goba, K. G. Andersen, R. S. G. Sealfon, D. J. Park, L. Kanneh, S. Jalloh, M. Momoh, M. Fullah, G. Dudas, S. Wohl, L. M. Moses, N. L. Yozwiak, S. Winnicki, C. B. Matranga, C. M. Malboeuf, J. Qu, A. D. Gladden, S. F. Schaffner, X. Yang, P.-P. Jiang, M. Nekoui, A. Colubri, M. R. Coomber, M. Fonnie, A. Moigboi, M. Gbakie, F. K. Kamara, V. Tucker, E. Konuwa, S. Saffa, J. Sellu, A. A. Jalloh, A. Kovoma, J. Koninga, I. Mustapha, K. Kargbo, M. Foday, M. Yillah, F. Kanneh, W. Robert, J. L. B. Massally, S. B. Chapman, J. Bochicchio, C. Murphy, C. Nusbaum, S. Young, B. W. Birren, D. S. Grant, J. S. Scheiffelin, E. S. Lander, C. Happi, S. M. Gevao, A. Gnirke, A. Rambaut, R. F. Garry, S. H. Khan, and P. C. Sabeti, "Genomic surveillance elucidates ebola virus origin and transmission during the 2014 outbreak," *Science*, 2014.

[6] D. Edwards and J. Batley, "Plant genome sequencing: applications for crop improvement," *Plant Biotechnology Journal*, 2010.

[7] Y. Turakhia, G. Bejerano, and W. J. Dally, "Darwin: A genomics co-processor provides up to 15,000x acceleration on long read assembly," in *ASPLOS*, 2018.

[8] M. Jain, H. E. Olsen, B. Paten, and M. Akeson, "The oxford nanopore minion: delivery of nanopore sequencing to the genomics community," *Genome Biology*, 2016.

[9] M. Vendruscolo and C. M. Dobson, "Protein dynamics: Moore's law in molecular biology," *Current Biology*, 2011.

[10] D. Senol Cali, J. S. Kim, S. Ghose, C. Alkan, and O. Mutlu, "Nanopore Sequencing Technology and Tools for Genome Assembly: Computational Analysis of the Current State, Bottlenecks and Future Directions," *arXiv e-prints*, 2017.

[11] H. Li, "Minimap and miniasm: fast mapping and de novo assembly for noisy long sequences," *Bioinformatics*, 2016.

[12] T. Smith and M. Waterman, "Identification of common molecular subsequences," *Journal of Molecular Biology*, 1981.

[13] S. F. Altschul, W. Gish, W. Miller, E. W. Myers, and D. J. Lipman, "Basic local alignment search tool," *Journal of Molecular Biology*, 1990.

[14] D. J. Lipman, J. Zhang, T. L. Madden, S. F. Altschul, A. A. Schäffer, W. Miller, and Z. Zhang, "Gapped BLAST and PSI-BLAST: a new generation of protein database search programs," *Nucleic Acids Research*, 1997.

[15] H. Li, "Minimap2: pairwise alignment for nucleotide sequences," *Bioinformatics*, 2018.

[16] C. Xu, D. Niu, N. Muralimanohar, R. Balasubramonian, T. Zhang, S. Yu, and Y. Xie, "Overcoming the challenges of crossbar resistive memory architectures," in *HPCA*, 2015.

[17] A. Shafiee, A. Nag, N. Muralimanohar, R. Balasubramonian, J. P. Strachan, M. Hu, R. S. Williams, and V. Srikumar, "Isaac: A convolutional neural network accelerator with in-situ analog arithmetic in crossbars," in *ISCA*, 2016.

[18] L. Song, X. Qian, H. Li, and Y. Chen, "Pipelayer: A pipelined reram-based accelerator for deep learning," in *HPCA*, 2017.

[19] W. Huangfu, S. Li, X. Hu, and Y. Xie, "Radar: A 3d-reram based dna alignment accelerator architecture," in *DAC*, 2018.

[20] A. Madhavan, T. Sherwood, and D. Strukov, "Race logic: A hardware acceleration for dynamic programming algorithms," in *ISCA*, 2014.

[21] R. Kaplan, L. Yavits, and R. Ginosar, "BioSEAL: In-Memory Biological Sequence Alignment Accelerator for Large-Scale Genomic Data," *arXiv e-prints*, 2019.

[22] L. Guo, J. Lau, Z. Ruan, P. Wei, and J. Cong, "Hardware acceleration of long read pairwise overlapping in genome sequencing: A race between fpga and gpu," in *FCCM*, 2019.

[23] Q. Guo, X. Guo, R. Patel, E. Ipek, and E. G. Friedman, "Ac-dimm: Associative computing with stt-mram," *SIGARCH Comput. Archit. News*, 2013.

[24] Y. Ono, K. Asai, and M. Hamada, "PBSIM: PacBio reads simulator–toward accurate genome assembly," *Bioinformatics*, 2012.

[25] S. Li, L. Liu, P. Gu, C. Xu, and Y. Xie, "Nvsim-cam: A circuit-level simulator for emerging nonvolatile memory based content-addressable memory," in *ICCAD*, 2016.

[26] F. Chen, L. Song, H. H. Li, and Y. Chen, "Zara: A novel zero-free dataflow accelerator for generative adversarial networks in 3d reram," in *DAC*, 2019.