IEEE TRANSACTIONS ON
EMERGING TOPICS
IN COMPUTING

Received 16 July 2021; revised 7 February 2022; accepted 19 February 2022.
Date of publication 1 March 2022; date of current version 6 December 2022.

Digital Object Identifier 10.1109/TETC.2022.3153613

CRAM-Seq: Accelerating RNA-Seq Abundance Quantification Using Computational RAM

ZAMSHED I. CHOWDHURY, S. KAREN KHATAMIFARD, SALONIK RESCH, HÜSREV CıLASUN, ZHENGYANG ZHAO, MASOUD ZABIHI, (Graduate Student Member, IEEE), MEISAM RAZAVIYAYN, (Member, IEEE), JIAN-PING WANG, (Fellow, IEEE), SACHIN S. SAPATNEKAR, (Fellow, IEEE), AND ULYA R. KARPUZCU, (Member, IEEE)

Zamshed I. Chowdhury, S. Karen Khatamifard, Salonik Resch, Hüsrev Cılasun, Zhengyang Zhao, Masoud Zabihi, Jian-Ping Wang, Sachin S. Sapatnekar, and Ulya R. Karpuzcu are with the Department of Electrical and Computer Engineering, University of Minnesota, Minneapolis, MN 55455 USA
Meisam Razaviyayn is with the University of Southern California, Los Angeles, CA 90007 USA

CORRESPONDING AUTHOR: ZAMSHED I. CHOWDHURY (chowh005@umn.edu)

This work was supported by National Science Foundation (NSF) under Grant SPX-1725420.

ABSTRACT RNA Sequence (RNA-Seq) abundance quantification is an important application in different fields of genomic studies, e.g., analysis offunctionally similar genes in a biological sample. This application depends on the availability of high volume of sequence data for high accuracy abundance estimation, which is made possible by next generation sequencing platforms. Large scale data processing requirements of this quantification application push conventional computing systems to their limits due to excessive data movement required between processing and memory elements. Processing-In-memory presents a viable solution to this drawback, through *in-situ* processing of the genomic data. In this paper, we present CRAM-Seq, an accelerator for RNA-Seq abundance quantification based on Computational RAM (CRAM) – an in-memory processing substrate capable of high degree of parallel processing with very low energy consumption. Through hardware/software co-design, we demonstrate that CRAM-Seq outperforms a commonly used state-of-the-art software abundance quantification algorithm, Kallisto – in terms of throughput and energy efficiency, while being highly scalable.

INDEX TERMS Abundance, accelerator, CRAM, quantification, RNA- seq, SHE-MTJ, spintronics

I. INTRODUCTION

Given a biological molecule, *sequencing* is the process of constructing its genomic composition in terms of the basic building blocks –i.e., nucleotide bases A(denine), T(hymine)/U (racil), C(ytosine), G(uanine)– where each base is represented by a character. Next Generation Sequencing (NGS) machines can typically generate very high volumes of sequence data per run, easily reaching hundreds of Giga (10^9) bases that translates into millions of fixed length strings of base characters called *reads*. Figure 1 shows sequencing throughput over time as a proxy for the volume of sequencing data produced by NGS platforms [28]. This trend is expected to hold and result in a steady increase in the volume of sequence data available for genomic analysis, enabling unprecedented advances in bioinformatics and medical research.

RNA molecules, as well, represent chains of nucleotide bases A, U, C and G. *RNA-Seq(uencing) Abundance Quantification* is an important emerging application that particularly benefits from the growth in sequence data volume as more data translates into higher computational accuracy. The goal is to estimate the relative distribution of a given set of RNA sequences in a biological sample. Hence, RNA-Seq in a sense "learns" from data. Important use cases for RNA-Seq include novel gene identification, gene expression quantification, mutation analysis, protein synthesis, precision and personalized medicine research (to identify active genes in cells, e.g.) to name a few.

Each RNA-Seq *read* usually is a smaller sub-sequence of a longer RNA sequence called an RNA *transcript*. Such reads are typically sequenced from a biological sample such as a single cell. The set of transcripts that is fixed and already known for a given sample is called RNA *transcriptome*, and uniquely characterizes that sample. The reads sequenced from a sample come from different transcripts in that sample, where

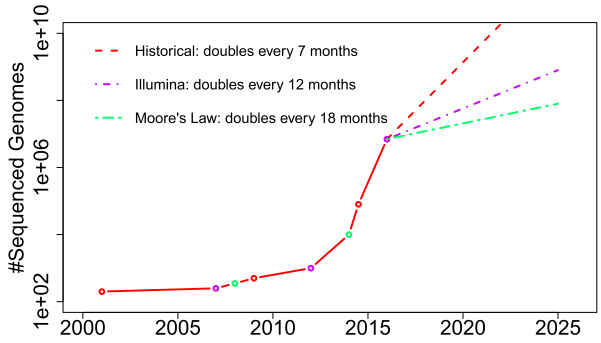

FIGURE 1. Evolution of sequencing throughput over time.

the transcripts vary in length and are more likely to generate a read if longer. Therefore, to get a representative RNA-Seq read dataset where the relative number of reads from each transcript follow a similar distribution to the transcripts in that sample, a very large number of RNA-Seq reads are required.

Technically, the *abundance* of the RNA-Seq reads refers to the relative distribution of the transcripts in a sample. Quantifying this distribution requires *alignment* between a large number of RNA-Seq reads and the transcriptome. (Exact) Alignment entails finding out where in each transcript (from that transcriptome) an RNA-Seq read matches the most–through base by base comparison. Classic (exact) alignment algorithms such as TopHat2 [16] or Cufflinks [29], however, require extensive computational resources due to exact alignment that relies on base by base comparison between RNA-Seq reads and the transcriptome. This results in a large number of slow and energy-hungry data transfers between the compute and memory elements (in a traditional setting), which inevitably degrades the performance– in terms of quantification throughput. Luckily, for abundance quantification, the actual location of alignment (i.e., exact alignment) is not required. Commonly used algorithms such as Sailfish [22] and Kallisto [4] hence avoid the costly exact alignment process by exploiting the characteristics (e.g., order) of common subsequences (called *k-mers*) in both RNA-Seq reads and transcripts, while maintaining a comparable accuracy to exact-alignment based methods. Such *pseudo-alignment* approximation significantly reduces the volume of data transfers during quantification, although for representative problem sizes it still remains forbidding.

A processing-in-memory (PIM) solution such as Computational RAM (CRAM) [30] can effectively address this performance bottleneck by fusing memory and compute elements together. CRAM is a generic PIM architecture and can be used with a wide-range of non-volatile (e.g., spintronic, resistive) memory cell technologies to perform both memory and *in-situ* computing operations. In this paper, we introduce and detail the hardware-software co-design of CRAM-Seq, a CRAM-based accelerator for RNA-Seq abundance quantification. We demonstrate that CRAM-Seq can achieve accuracy similar to state-of-the-art software solutions such as Kallisto [4], while operating faster and more energy-efficiently. The key contributions of this paper are as follows:

- We show that *only* presence (i.e., not order) of unique k-mers suffices to perform RNA-Seq abundance quantification.
- We present an end-to-end PIM-based accelerator architecture to achieve higher quantification throughput with lower energy consumption and comparable accuracy when compared to a commonly used high throughput abundance quantification algorithm, even in the presence of noise in the sequence data.

The rest of this article is organized as follows: Section II discusses the core concepts related to RNA-Seq, transcripts and abundance quantification. Section III illustrates the architecture and discusses different design aspects. Section IV and Section V present the evaluation setup and experimental findings. Related work is discussed in Section VI with Section VII concluding this article.

II. BACKGROUND

A. RNA-SEQ

Typically, both transcripts and RNA-Seq reads don't contain U(racil) as they are sequenced from complementary DNA molecules composed of {A, T, C, G} where each character represents a base pair (bp). Transcripts are longer than fixed length reads, and read length depends on the sequencing technology. Reads from Illumina [25] platforms, e.g., usually are ≈100 bp long.

B. RNA-SEQ ABUNDANCE QUANTIFICATION

Given a set of transcripts, abundance quantification determines the distribution of each transcript in a biological sample when a large number of RNA-Seq reads are sequenced from that sample. If $T = \{t_0, t_1, t_2, \ldots t_{N_T-1}\}$ is the set of transcripts, the quantified abundance of a transcript is: $Q(t_i) = C(t_i)/\sum_{j=0}^{N_T-1} C(t_j)$ where $C(n)$ is the number of RNA-Seq reads mapped to transcript n.

Figure 2 shows a typical abundance quantification pipeline: A biological sample (containing varying length transcripts *T0*, *T1* and *T2* in different quantities) produces a large (N) number of fixed length RNA-Seq reads. The first step involves, for each RNA-Seq read, identifying the transcripts that have the highest degree of *similarity* with that read (e.g., the total number of locations where both the transcript and RNA-Seq read have a common sub-sequence of a given length in bp). Be it exact or approximate, this alignment step (as enclosed in the dashed box) helps to identify the (group of) transcript(s) that more likely originated that particular read. Each unique group of transcripts denotes a *similarity class*. As the read dataset is exhausted, upon the processing of each read, a collection of such similarity classes with corresponding counts of RNA-Seq reads (i.e., members) that map to them gets updated. The similarity class of a read can have one transcript (which is uniquely identified as the source of that read), or multiple transcripts (where maximum similarity applies to all transcripts in the class). In the final step, an iterative clustering algorithm –typically, Expectation Maximization (EM) [8]– distributes read counts to the cluster

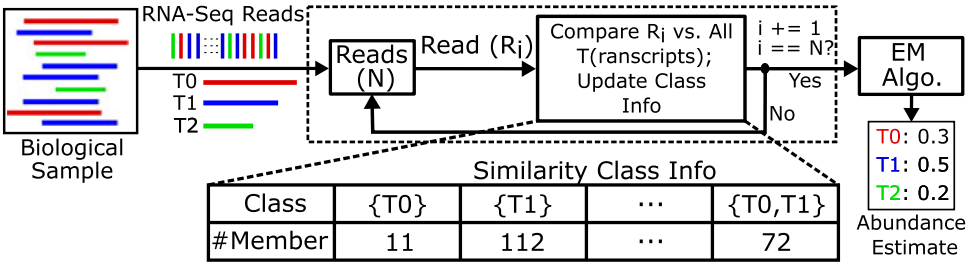

FIGURE 2. **Quantification Overview.**

points, i.e., the transcripts, utilizing all similarity class information, to maximize the abundance of each transcript. The most compute-resource-heavy part of this problem is the alignment step. As an example, even pseudo alignment (which is typically faster than exact alignment) can consume $>85\%$ of total runtime in Kallisto [4] or Sailfish [22]. Therefore, accelerating this step is of particular importance to improve the overall quantification throughput as shown in Figure 3. Being independent from (and significantly faster than) the alignment step, EM algorithm for one set of reads can overlap with the alignment step of the next set of reads. The alignment step determines the rate of quantification as the slower step of computation, i.e., accelerating alignment is critical in improving the overall rate of quantification.

C. ERRORS IN RNA-SEQ READS

Due to imperfection in sequencing technology, RNA-Seq reads can have errors at random locations along the sequence, which typically take the form of insertion, deletion or substitution of one or more bps. The correctness of the abundance algorithm depends on the ability of the quantification algorithm to tolerate such noise. Transcripts are considered to be free from noise since these are verified across multiple iterations of sequencing.

D. CRAM BASICS

A CRAM tile is a 2D array of non-volatile memory cells. Different options exist for the memory device technology, which can give rise to subtle differences in inter-cell connectivity in a tile due to differences in the number of terminals per memory device[1]. Without loss of generality, we consider Spin-Hall-Effect (SHE)-MTJ (also known as Spin-Orbit-Torque or SOT-MTJ), as the memory device, connected to the rest of the tile through two access transistors (Figure 4) [31]. SHE-MTJ has lower switching latency and energy consumption in comparison to Spin-Transfer Torque (STT)-MTJ, Phase Change Memory (PCM) and Resistive RAM (ReRAM). In terms of endurance and data retention as well, SHE-MTJ outperforms PCM and ReRAM. On top of these, due to separation of read and write current paths, SHE-MTJ is not susceptible to read disturbance errors. Each SHE-MTJ has one Magnetic Tunneling Junction (MTJ), juxtaposed on top of a heavy metal SHE channel. Each MTJ consists of two layers of ferromagnets– pinned and free layers, insulated by a thin layer. The pinned layer has its magnetic spin orientation fixed, which is controllable in the free layer. A current through SHE channel, depending on the direction and magnitude, can switch the spin orientation of free layer. The relative spin orientation of the free layer compared to the pinned layer gives rise to two distinct MTJ resistance levels: R_{high} and R_{low} which encode logic 1 and 0, respectively. A wire along each column, logic line (*LL*), connects memory cells in that column via access transistors to perform logic operations in the CRAM tile, controlled by Read (Write) Word lines (RWL and WWL, respectively).

Memory operation: A voltage is applied between LL and Bit Select Line (BSL). For reads(writes), current flows through SHE-MTJ (only SHE channel) (Figure 4(a)).

Logic operation: Figure 4(b) illustrates a logic gate with two inputs (*Input1* and *Input2*), and one output (*Output*) which is preset to a known value (0 or 1, depending on the type of logic operation). Input and output cells are attached to opposite BSL: odd or even (OBSL or EBSL), alternately connected to adjacent cells in each column. Figure 4(c) also depicts the equivalent circuit: a voltage, V_{gate}, is applied between OBSL and EBSL. The currents through the inputs, I_1 and I_2, depend on V_{gate} and their respective resistance values, R_1 and R_2. RWL and WWL, set to 1 (bold and colored), connect the input and output cells to LL. $I_{out} = I_1 + I_2$ flows through the output resistance, R_{OUT}. The orientation of the free layer is switched (in a direction specified by the direction of the current) when $I_{out} \geq I_{crit}$, the threshold switching current, causing the logic state of *Output* to change. If not, previous (preset) state is maintained by *Output*. The use of logic gate specific voltages (V_{gate}) and presets enable CRAM to be Boolean complete.

III. DESIGN

A. PROBLEM STATEMENT

k-mers are sub-sequences of length k (e.g., a RNA-Seq "TCGAC" has three 3-mers: TCG, CGA and GAC). *The*

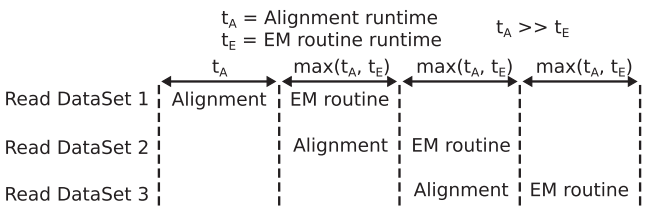

FIGURE 3. **Timing diagram of abundance quantification.**

[1]We direct interested readers to literature that use CRAM architectures with 2- and 3-terminal spintronic devices [6], [24], for more insight on portability of CRAM-Seq to other cell technologies.

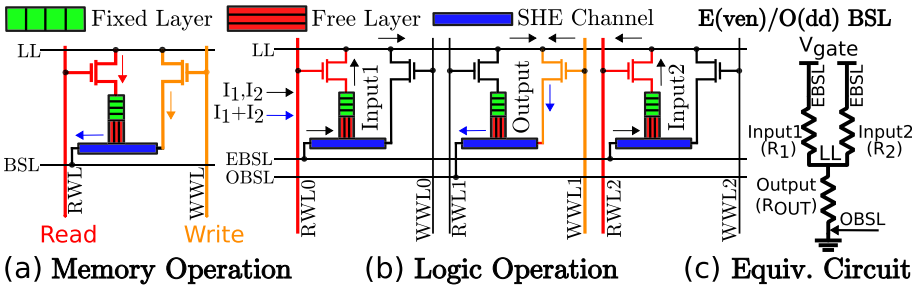

FIGURE 4. **Bit-wise logic operation in CRAM.**

presence, and not the order, of distinct k-mers in a RNA-Seq read sequenced from a transcript matters. Abundance quantification only requires information regarding the presence of different k-mers in transcripts and corresponding RNA-Seq reads to produce an estimate with an accuracy comparable to commonly used quantification algorithms. Accordingly, we convert a (read/transcript) sequence to a bit-vector (of length determined by k), using the sequence's k-mers. Since k-mers are overlapping in a sequence, the maximum number of k-mers in a sequence of length N is equal to $N - k + 1$. Each unique k-mer generates a unique numerical value when passed through a hash function (due to position and value of each base character in that k-mer). For example, with a hash function $\sum_{i=0}^{k-1} 4^i \times \{A, T, C, G\}$ – where {A = 0, C = 1, G = 2, T = 3} and i demarcates the location in the k-mer – a 5-mer "CTCGA" gets the value of 157, as shown in Figure 5 (a). The maximum number of unique hash values from a sequence depends on k, e.g., hash values in Figure 5(a) have a range between 0 and $4^k - 1$. By using hash values to identify individual bit locations on a bit-vector and setting those bits, the presence of unique k-mers can be marked in the bit-vector. In a nutshell, a bit-vector records all unique k-mers in a (read/transcript) sequence, but the order of 1's in the bit-vector does not necessarily reflect the order of k-mers in the corresponding sequence.

A longer k-mer yields longer bit-vectors that are more likely to be unique, hence such sparser bit-vectors can determine similarity between a RNA-Seq and a transcript more accurately. A small k-mer, on the other hand, can save hardware resources due to smaller (but denser) bit-vectors. Considering noise in sequence data, however, large k may not always deliver higher accuracy. The choice of k depends on the trade-off between required hardware resources and the desired quantification accuracy. Ideally, a bit-vector should be sparse enough such that it uniquely represents a sequence to distinguish it from the other sequences.

A set of very long sequences, e.g., transcripts, might have all unique k-mers (for a given k) in each of them, resulting in identical bit-vectors. This loss of information (i.e., distinction between different sequences) is very likely if the sequence length $L >>$ #uniquely identifiable k-mers. To handle this issue, we break the transcripts down into overlapping segments (as shown in Figure 5(b)) of constant maximum length, and generate for each segment the corresponding bit-vector. These bit-vectors, together, are more likely to uniquely represent the corresponding transcript. The maximum segment length (> read length) is selected to be neither much greater than read length nor too small to produce too many segments. Without overlapping between segments, a RNA-Seq read that spans across two consecutive segments of a transcript could have low similarity with both segments even though that RNA-Seq read comes from that particular transcript.

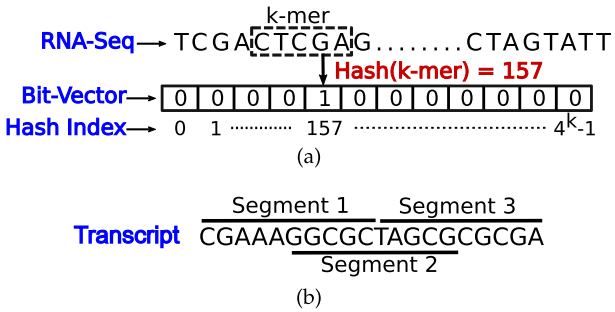

FIGURE 5. **(a) k-mer in RNA-Seq; (b) Segmentation of transcript.**

B. ALGORITHM DESIGN

A set of RNA-Seq reads, $\{RS_1, RS_2, \ldots, RS_N\}$ and a set of transcripts, $\{TS_1, TS_2, \ldots, TS_M\}$ are given. Reads are transformed into the corresponding bit-vectors $\{R_1, R_2, \ldots, R_N\}$. Each transcript is transformed into a number of bit-vectors $(T_{10}, T_{11}, T_{12}, T_{20}, T_{21}, \ldots, T_{M0}, T_{M1})$ since each transcript is also divided into smaller segments of a fixed maximum size. In the first stage of the algorithm, each RNA-Seq read bit-vector is bit-wise compared with all transcript segment bit-vectors to compute the similarity between each {RNA-Seq read, transcript segment} pair. The comparison operation (bit-wise AND), in Figure 6, generates a string of 1 s and 0 s, the *Match String*, where logic 1 marks the presence of a unique k-mer in

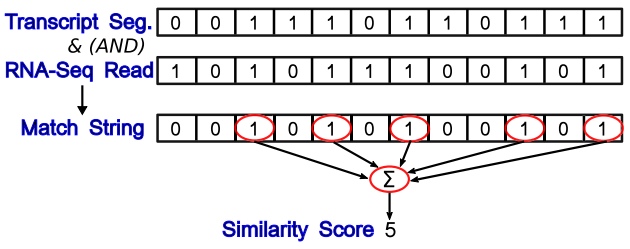

FIGURE 6. **Similarity computation.**

TABLE 1. 2-Input and Truth Table (*Out* Preset = 1).

In_1	In_2	Out	$I_{out} = I_1 + I_2$	
0 (R_{low})	0 (R_{low})	0	I_{00}	$\geq I_{crit}$
0 (R_{low})	1 (R_{high})	0	I_{01}	$\geq I_{crit}$
1 (R_{high})	0 (R_{low})	0	$I_{10} = I_{01}$	$\geq I_{crit}$
1 (R_{high})	1 (R_{high})	1	I_{11}	$< I_{crit}$

both RNA-Seq read and transcript segment. The number of logic 1's in the *Match String* is a proxy of similarity between that {RNA-Seq read, transcript segment} pair. Once a RNA-Seq read is evaluated against all transcript segments, transcript segments with maximum similarity are identified, and member counts of corresponding classes are updated accordingly. Once all reads are processed, member counts suffice to compute abundance.

Algorithm 1 summarizes CRAM-Seq's processing steps. For each read, the *for* loop (lines 7-10) counts the number of common k-mers with each transcript segment by performing bit-wise comparisons, and a sequence of bit-wise additions, i.e., pop(ulation)_count. Then, for each read, *SimClass* function (Algorithm 2) returns the transcript indices with maximum similarity score (line 11). The list of indices (stored in *sclass*) serves as the ID of the similarity class of the read currently being processed. If this class hasn't been encountered before, it is recorded as a new entry in *SCT*, an array of key-value stores with key = class ID, value = member count. Finally, the corresponding member count in *SCT* is incremented by 1 (line 15). The *MAX* operation in Algorithm 2 (line 2) returns all transcript segment indices with the maximum score. All such segment indices are used to access a look-up-table (LUT) that stores the mapping between the segment and transcript indices, to extract the corresponding transcript indices (line 6).

Algorithm 1. CRAM-Seq algorithm

```
1: // Number of transcripts: T
2: // Number of transcript segment bit-vectors: S (S >> T)
3: // Number of RNA-Seq read bit-vectors: N (for N reads)
4: Initialize entries of Similarity Class Table SCT to 0
5: Initialize entries of count array (of size S) to 0
6: for all R in N do
7:    for all t in S do
8:       MS = R & t        // MS: Match String
9:       count[t] = pop_count(MS)
10:   end for
11:   sclass = SimClass(count)
12:   if !(sclass in SCT) then
13:      Create class entry in SCT and return sclass
14:   end if
15:   SCT[sclass] += 1
16: end for
```

Finally, after all reads are processed, the class information from *SCT* feeds the EM algorithm to quantify the abundance of the transcripts. All operations in Algorithm 1 and (except for LUT accesses) Algorithm 2 – AND, pop_count, MAX, class ID check and member count increment – are highly parallel bit-wise operations, hence can effectively be mapped to CRAM.

Algorithm 2. SimClass(count)

```
1: Initialize transcript_indices array
2: index_list = MAX(count) // List of all segment indices
   with maximum score
3: for all E in index_list do
4:    transcript_indices.append(LUT[E])
5: end for
6: return transcript_indices
```

C. BUILDING BLOCKS

For the *AND* gate to perform bit-wise comparison, the corresponding bits of the transcript segment and RNA-Seq read bit-vectors act as inputs (stored in the same column of a CRAM tile). Table 1 provides the truth table. V_{gate} is selected such that I_{11} does not exceed I_{crit}, while I_{00}, and $I_{01} = I_{10}$ do, so that *Out* would switch from its preset value of 1 to 0, for all input combinations but 11. Logic *OR* can be configured similarly with preset 1 and specific V_{gate}.

Bit-wise addition of 1 s in *Match String* (pop_count in Algorithm 1) is performed by a sequence of MAJ(ority)-3, MAJ (ority)-5 and INV(erter) operations: C_o = MAJ(In_1, In_2, C_i); S_1 = S_2 = INV(C_o); *Sum* = MAJ($In_1, In_2, C_i, S_1, S_2$). In_1 and In_2 represent the augend and addend, respectively; C_i, the initial carry input (= 0). Steps- 2 and 3 are fused together by a 2-output *INV* gate. The operation principle of these gates is no different than any other bit-wise logic operation, except for the combination of the specific preset and V_{gate}.

D. HIGH LEVEL ARCHITECTURE

Figure 7 (a-c) show the functional blocks necessary for executing Algorithm 1 and Algorithm 2. CRAM-Seq is comprised of three top-level modules, namely vector transformation unit (VTU), similarity class unit (SCU) and a collection of core computational units, termed processing elements (PE). A given set of transcripts are segmented and transformed into corresponding bit-vectors in VTU (cost of which is amortized over multiple iterations of quantification with millions of reads), and mapped to PEs before (global controller orchestrated) computations take place. After a RNA-Seq read arrives at CRAM-Seq from the host, the read is first transformed in VTU into the corresponding bit-vector, and mapped to all PEs to compute the similarity scores between that read and all pre-stored transcript segments. Through a sequence of *in-situ* logic and arithmetic operations, the similarity scores are computed and sifted through to derive the similarity class statistics for that particular read. SCU stores and updates the statistics, and sends the statistics back to the host once all reads are processed.

Vector Transformation Unit (VTU): This unit is responsible for receiving the data, e.g., RNA-Seq reads from the host, and for generating the corresponding bit-vectors. The k-mer length used by VTU is the same as what is used in the

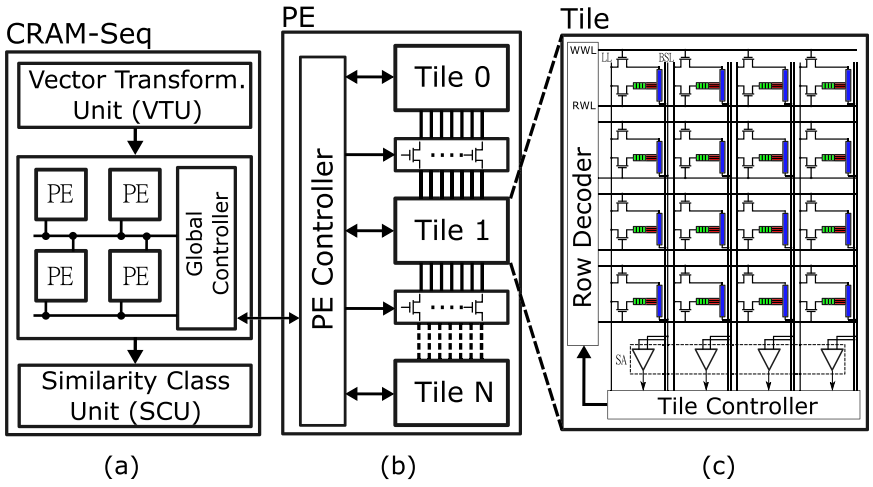

FIGURE 7. High-level architecture of CRAM-Seq.

transformation of the transcript segments in order to make all bit-vectors to have the same length for bit-wise comparisons. Figure 8 illustrates the transformation operation performed by the VTU. The bit-vector in VTU is a collection of data-latches that can be *SET*(*RESET*) to logic 1(0). At the beginning of the VTU operation, the bit-vector is initialized to all logic *0 s*. Transformation of a RNA-Seq read (/transcript segment) entails *SET*-ting individual bit-locations on that bit-vector where each bit-location corresponds to a unique k-mer along that read (k-mers are considered overlapping). k-mers in a RNA-Seq read, in 2-bit encoded format, represent overlapping groups of $2\,k$ bits. In Figure 8, the group of 10-bits represents the first 5-mer, *ATAGC*, in a RNA-Seq read, *ATAGCTGAC*. The second k-mer, *TAGCT*, refers to the subsequent group of 10-bits–skipping the first two bits (i.e., one character). The *group of bits* refers to the location of the *bit* in the bit-vector that would be *SET*. This process is repeated for all k-mers in that read. Once all overlapping k-mers are processed, i.e., all corresponding bit locations on the bit-vector are *SET*, the transformation is complete, and VTU notifies the global controller. As soon as the PEs are done processing the previous read, the global controller writes the transposed read bit-vector to PE(s) through the standard write mechanism

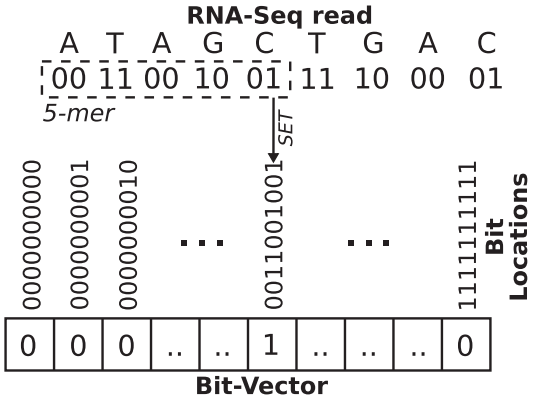

FIGURE 8. Vector transformation unit (VTU).

(Section II-D). Such pipelined operation hides the latency incurred by VTU.

Processing Elements (PE) are the core computational units, which are responsible for performing computations with CRAM tiles. Each PE consists of a number of connected CRAM tiles. The number of columns in a PE is the number of columns in any PE tile, whereas the number of rows in a PE is the sum of all rows of all tiles in that PE. Figure 7(c) shows the internal details of the *tile*, the basic building block of each PE. Each tile is a 2D array of SHE-MTJ cells, capable of performing bit-wise universal logic operations on the stored data.

Tile connectivity: The tiles are connected through an array of switches (i.e., transistors) that are controlled by the corresponding PE controller, which column-wise connects the LL(s) of adjacent tiles (Figure 7(b)). When switches are *OFF*, LL(s) across adjacent tiles are effectively disconnected, hence the tiles can compute in parallel, using individual LL (s) in each tile. However, when turned *ON*, the switches effectively connect LLs across tiles and along PE columns. Therefore, logic operations such as COPY can be performed along columns across the boundary between adjacent tiles. The *ON* resistance of transistors is much lower than that of SHE-MTJ cells, which makes the overhead of transistor based connection between two LLs across tiles negligible. This control over computation across tiles enables us to exploit tile-level parallelism efficiently.

E. DATA LAYOUT

As shown in Figure 9, each PE column is organized in four compartments: transcript segment and RNA-Seq read bit-vectors, result (for storing *Match String* and similarity score) and scratch bits (to store intermediate data during computation). To exploit tile-level parallelism within a PE, a transcript segment bit-vector is divided into a number (equal to the number of tiles in a PE) of smaller sub-vectors and consecutive sub-vectors are stored in consecutive tiles in a PE column. Same applies for read bit-vectors. To summarize, a PE column stores the (transcript segment and read) bit-vectors as a whole, spread across all tiles in that PE.

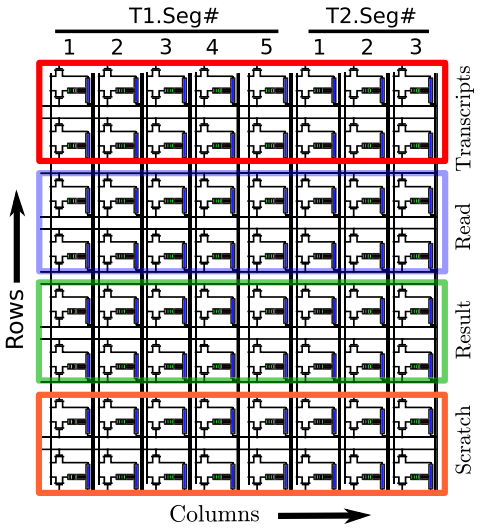

FIGURE 9. **PE data layout in CRAM-Seq.**

F. PE OPERATIONS

Next, we describe the sequence of computational steps within a PE, as captured by Figure 10 considering all tiles in the PE. Each step (e.g., St_1) shows a snapshot of the PE state after a sequence of CRAM logic operations are completed. Note that, for ease of illustration and explanation, rows and columns are transposed. This figure illustrates how (transcript segment and read) bit-vectors, *T1–T4* and *R1*, are mapped to a PE, and how the corresponding similarity scores are generated. Initially at step St_0, the transcript segment bit-vectors are stored in the PE: each such bit-vector is divided into equal length sub-vectors with adjacent tiles storing consecutive sub-vectors. For instance, the transcript segment bit-vector, *T1*, is divided into 4 equal length sub-vectors *T1.0–T1.3* and stored in *row-0* of tile-0 – tile-3. Each row of each tile contains one sub-vector of a transcript segment bit-vector. The read bit-vector (of same length) is mapped in the same way and written to rows of each tile. All rows in a tile contain the same read sub-vector (e.g., *R*1.0 in tile-0, *R*1.1 in tile-1 and so on).

In step St_1, partial *Match String*, MX.Y, is produced through bit-wise *AND* operations between the stored transcript segment sub-vectors (TX.Y) and RNA-Seq read sub-vectors (R1.Y), e.g., *M*1.0 represents the partial *Match String* between *T*1.0 and *R*1.0. The *Match String*, *MX*, which is the output of *AND* operations between *T*1 and *R*1, is the concatenation of MX.Y, e.g., $M1 = \{M1.0, M1.1, M1.2, M1.3\}$. Subsequent steps compute the number of 1's in *MX*. Recall that each row of the PE tiles contains partial *Match strings*. Specifically, tile-*Y* computes a partial count of 1's in *MX.Y* at step St_2. E.g., tile-0 computes number of 1's in *M1.0* through *M4.0*, in a row parallel fashion. The result is partial similarity scores (e.g., *S1.0-S4.0* in tile-0), where *S1.0-S1.3* represent the partial scores of the final score *S1*. To compute the total counts of 1's (i.e., the final similarity score) in individual *Match Strings*, these partial similarity scores in different tiles need to be reduced to single similarity score.

To this end, partial scores are transferred to the adjacent tile(s) at step St_3, to perform binary (i.e., ripple-carry) addition between partial scores (i.e., a sequence of standard bit-wise additions starting from the least significant bits). E.g., *S1.0-S4.0* from tile-0 are transferred to tile-1, in corresponding rows. Similar transfers from tile-3 to tile-2 are conducted simultaneously. Step St_4 (not shown explicitly) performs another round of bit-wise addition between the partial scores, in tile-1 and tile-2 simultaneously. Step St_5 again transfers partial outputs of the binary additions from tile-1 to tile-2. A final step of binary addition, at Step St_6, produces the final similarity scores in tile-2, *S1-S4* (shown in green).

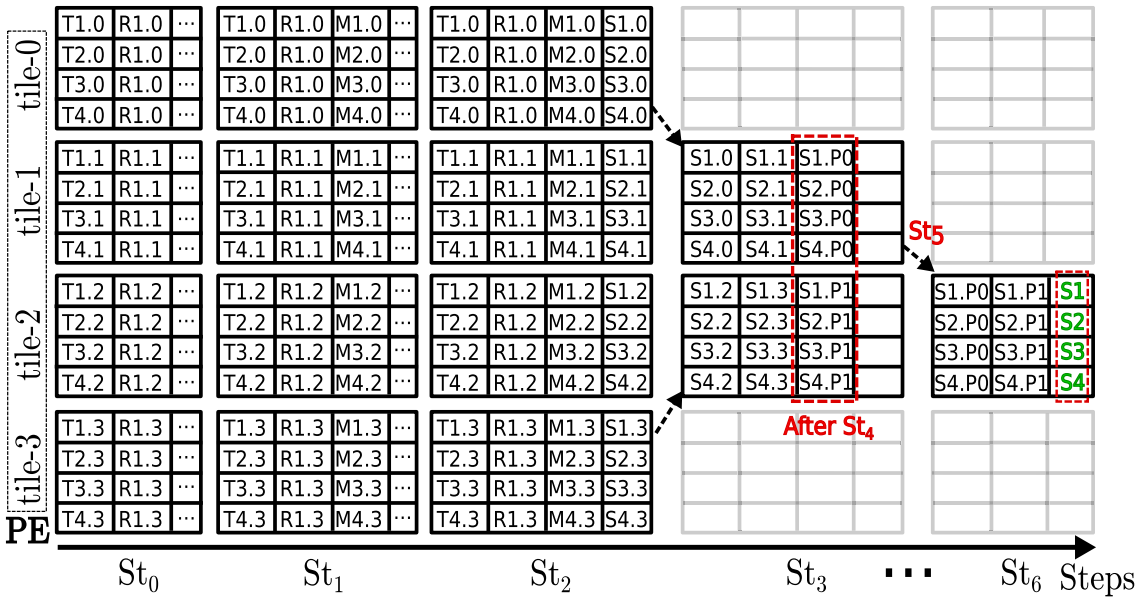

FIGURE 10. **Sequence of computational steps within a PE (rows and columns are transposed).**

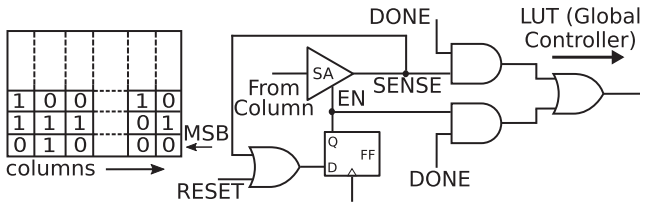

FIGURE 11. **In-Memory MAX operation on similarity scores.**

With the PE configuration shown in Figure 10, a bit-vector length of 128-bits would be representative, spread across 4 tiles, with each row storing 32-bits of transcript segment bit-vector each (Section IV). Not shown in the figure (to simplify illustration) are scratch bits in each row which are preset accordingly before computation starts, and peripheral overhead. In this case, Step St_1 encapsulates 32 AND operations. The outputs of the bit-wise (reduction) addition of *Match String* in each tile are available at Step St_2, after having performed a total of 139 more CRAM logic operations. The copy operations on the 6-bit (bit-wise reduction addition) outputs between adjacent tiles are completed at step St_3, after performing 6 more CRAM operations. The first round of binary additions on partial reduction outputs are completed at St_4, after performing 18 more CRAM operations. The subsequent copy at Step St_5 encapsulates 7 more operations. The final 8-bit similarity score is available after performing 21 more CRAM operations at St_6, i.e., after completing 7-bit binary additions at each row.

G. CLASS EXTRACTION

Class extraction corresponds to *MAX* function in Algorithm 2. This is achieved in two steps: (i) finding the transcript segment(s) with maximum similarity score, i.e., PE columns with the maximum score (across all PEs, simultaneously); and (ii) finding out the transcript indices of the segment(s) identified in step (i). In hardware, this is achieved through the use of sense amplifiers (SA) present in PE tiles. Figure 11 shows the basic idea: The tile in the figure holds the final scores along columns, in binary form, between a RNA-Seq read and all transcript segments in the corresponding PE. To find the maximum of two or more such binary values, we simply perform bit-wise comparison starting from the most significant bit (MSB) position. As an example, the tile in Figure 11 stores similarity scores {011_2 (3_{10}), 110_2 (6_{10}),..., 010_2 (2_{10})}.

In each PE, each column of *one* specific tile (which keeps the multi-bit end outcomes, e.g., tile 2 from Figure 10) needs

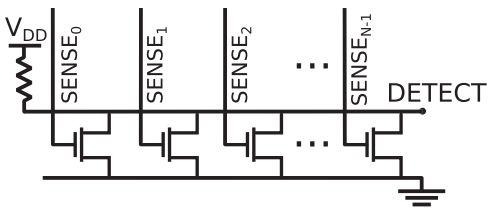

FIGURE 12. **Transistor array to detect logic 0 on all SENSE lines.**

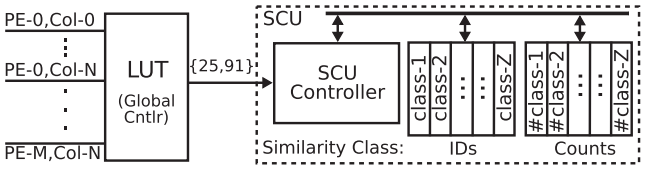

FIGURE 13. **Similarity class store and update.**

to be scanned (at one bit position at a time, from MSB to LSB positions, across all PEs simultaneously), using the corresponding SAs. Prior to reading out the column bits at the MSB position, PE controller sets the *RESET* feeding the input of the D-FlipFlop (D-FF), which in turn enables (via *EN*) all participating SAs in all PEs. In the first round of comparison, the bits at the MSB position in all columns are read out simultaneously. If at least one bit is 1, all read-out values are stored in the corresponding D-FFs. This in turn enables only the columns (i.e., SA) with logic 1 at the MSB position to participate in the next round of comparison at the next bit position. However, if all values are 0, the sensed data is not stored and all participating SAs remain enabled for the next round. A transistor array pulled-up by a resistor, as shown in Figure 12, generates a *HIGH* on *DETECT* when *all* SENSE lines are *LOW*; SENSE lines are connected to gates of individual transistors. A logic *LOW* on *DETECT* is required to store the read-out values in the D-FFs. This self-filtering process continues until the final round (reaching the LSB), when the PE controller sets *DONE*. Thereby the global controller gets either the read-out value (*SENSE*) *OR* the content of D-FF corresponding to column(s) with the maximum value only, i.e., columns that survived until the last round. Each column, i.e., transcript segment is connected to a distinct memory address in a lookup table (LUT) that stores the corresponding transcript index. Only the columns with maximum score determine which LUT address(es) to access. The area overhead of additional logic (for MAX operation and all logic 0 detection) is insignificant (only one tile per PE is involved in these operations) compared to a SOT(SHE)-MRAM substrate of similar size.

H. SIMILARITY CLASS UNIT (SCU)

SCU keeps track of similarity class information. The {*class ID*, *member count*} values are stored in *Similarity Class IDs* and *Similarity Class Counts* modules respectively. Figure 13 illustrates the process. *Similarity Class IDs* and *Similarity Class Counts* are collections of CRAM tiles (similar to PE), which have the same number of columns. For each entry (i.e., column) in *Similarity Class IDs*, there is a corresponding entry in *Similarity Class Counts* that stores and updates the member count of that class. SCU controller receives the transcript indices (which form a similarity class) from LUT (in global controller) for a RNA-Seq read, such as {25, 91} in Figure 13. Each similarity class is represented by an *ID bit-vector* of length equal to the total number of *transcripts* stored, where transcripts in a class are marked by setting the corresponding bits to 1. This *ID bit-vector* is used to check if

IEEE TRANSACTIONS ON EMERGING TOPICS IN COMPUTING

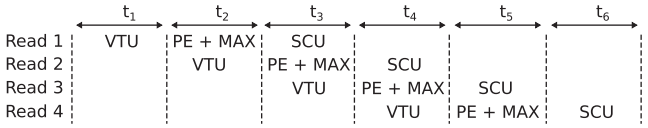

FIGURE 14. **Pipeline stages in CRAM-Seq.**

a class already exists in *similarity class IDs* module through bit-wise *AND* operations with the stored *ID bit-vectors*, followed by bit-wise *OR* operations to reduce the outcome to 0 or 1: If 0, no such class exists. A new similarity class is created by storing that ID bit-vector in *Similarity Class IDs*, and incrementing the corresponding entry in *Similarity Class Counts* by 1 through *in-situ* bit-wise addition. Otherwise, the corresponding entry in *Similarity Class Counts* is incremented by 1. An alternative approach with lower compute (i.e., lower latency and energy) and memory requirements, to distinguish between similarity classes with common subset of transcripts more efficiently, uses *ID bit-vector* as a bit-mask to selectively perform bit-wise *reduction AND* on the bits in a stored *ID bit-vector*. The remaining bits are also *reduced* using bit-wise *OR*, followed by an *INV*. Finally, performing *AND* on the outputs from both reduction steps reveals (0/1) whether the *ID bit-vector* already exists in *similarity class IDs* module.

Optimized encoding for *class IDs* using an advanced hashing algorithm can reduce SCU energy consumption by reducing #bits to store *class IDs*. This would not only simplify the underlying computations but also translate into more room for transcript(-segment)s per chip. Algorithm 3 shows an example, where *class ID*s are derived from the higher-order bits of transcript indices only through shift-left and bit-wise *XOR* operations (performed by *SCU controller*). The number of higher-order bits and shift-left operations depend on the maximum number of transcripts a class can hold.

Algorithm 3. Optimized hash

1: T_{max}: maximum #transcripts per class
2: N_{bits}: #higher-order bits in a transcript index
3: $Transcript_{index} = \{100, 157, 852, 223, \ldots\}$// *list of transcript indices*
4: *MBL*: // *maximum length of a transcript index, in bits*
5: *hash_val* $= 0$ // *to store final hash value*
6: **for** *i* in range(T_{max}) **do**
7: $hash\_val = hash\_val \oplus Transcript_{index}[i][MBL : MBL - N_{bits}]$;
8: $hash\_val << 1$
9: **end for**
10: return *hash_val*

I. PIPELINE STAGES

Figure 14 illustrates the life-cycle of RNA-Seq reads in the CRAM-Seq pipeline. A read flows through three functional blocks, each constituting a pipeline stage: VTU, PE+MAX and SCU. The VTU can operate on the next read while PEs process the previous one. However, PEs cannot start processing the next read until the maximum similarity score for the current read is calculated and sent to the SCU. Therefore MAX has to immediately follow PE operations, and together they form the longest latency pipeline stage, PE+MAX. Once the PE+MAX stage is done, the SCU begins immediately, and overlaps with the VTU and PE+MAX stages operating on subsequent reads. Once all reads are processed (i.e., exit the pipeline), the SCU transfers the final class information to the host where the next step of quantification (i.e., EM algorithm) runs.

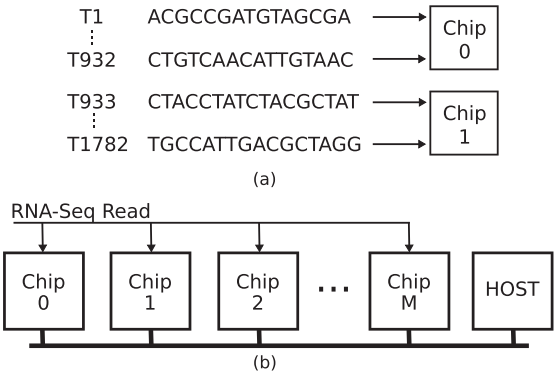

FIGURE 15. **Scale-out system of CRAM-Seq chips.**

J. SYSTEM INTEGRATION

CRAM-Seq connects to the host through the standard memory interface. As an accelerator substrate, CRAM-Seq does not share the virtual memory space with the host. Data (i.e., RNA-Seq reads) stream into the accelerator from the main memory for pseudo-alignment. Once CRAM-Seq computations finish, the similarity class information is sent back to the host which continues with the EM algorithm. Since no fine-grain control over CRAM-Seq operations is necessary, the programming interface involves instructions for sending and receiving data to/from CRAM-Seq only.

K. MULTI-CHIP DESIGN

The number of transcript segments in a dataset may exceed the storage capacity of a single CRAM-Seq chip, necessitating deployment of multiple chips. Figure 15 illustrates a scale-out system with *M* CRAM-Seq chips. In this case, the total number of transcript segments in the dataset is divided into chunks and each chunk is assigned to a separate chip, as shown in Figure 15(a). Each CRAM-Seq chip stores only the class information corresponding to the chunk of transcripts it is responsible for. The *throughput* is maintained across all chips in the system due to weak scaling (i.e., addition of more PEs across multiple CRAM-Seq chips).

As each bit-vector corresponding to a RNA-Seq read is processed by all chips in parallel, there is no need for a separate VTU in each CRAM-Seq chip. Sharing the VTU between multiple chips increases area efficiency, leaving more room for actual computation units (i.e., PEs) in the same chip area, or alternatively resulting in smaller chips for the same PE count. As the amount of memory required per

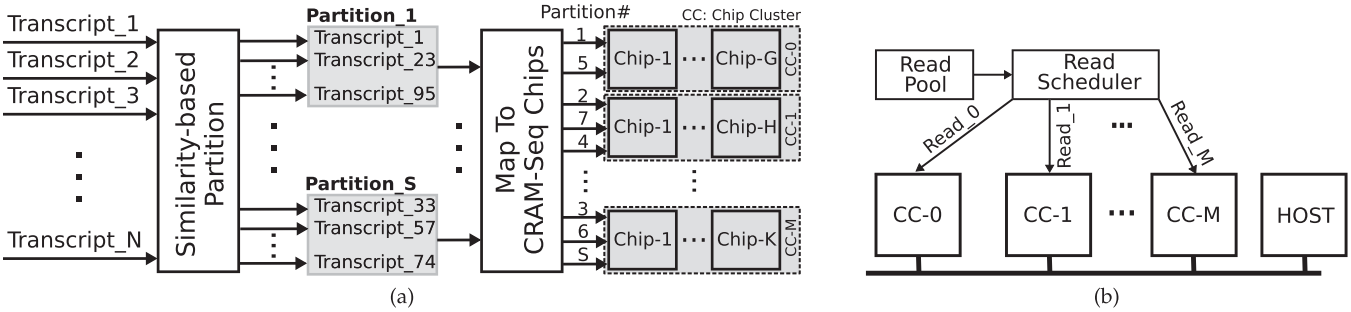

FIGURE 16. (a) Transcriptome partitioning and mapping to CRAM-Seq chip clusters; (b) read scheduling to clusters of chips.

chip depends on the number of transcript(segment)s stored, and all chips in the scale-out design can store the same number of transcript(segment)s, the scale-out design is as memory efficient as a single CRAM-Seq chip.

On the other hand, the energy consumption of the scale-out design is by construction higher due to the higher number of pseudo-alignments per read. While exploiting read characteristics (such as the number of unique k-mers) to assign reads only to a subset of chips for pseudo-alignment may reduce the number of pseudo-alignments, a more effective solution is combining such read scheduling with clustering of transcript(segment)s into partitions (as shown in Figure 16 (a)) based on the pair-wise similarity between the transcripts in the dataset [32], where each transcript partition is uniquely identified by k-mer(s), called *sig-mer(s)*. This partitioning of the transcriptome is based on the following observation: For typical datasets, the bit-wise *AND*s between the transcript segment bit-vectors and a given read bit-vector tend to generate a non-sparse output only for a specific subset of transcripts. Therefore, confining the quantification within such subset of transcripts on a per read basis can cut the number of unnecessary pseudo-alignment computations significantly. Assigning each such partition to a cluster of CRAM-Seq chips and confining the pseudo-alignment of the reads to the chips in that cluster with the respective partitions only would be especially useful.

Transcript partitioning and subsequent mapping to chips are done only once (before storing the transcripts in CRAM-Seq chips) and therefore, the overhead of such pre-processing is amortized over many iteration of quantification with millions of reads in each iteration. In this case, successive transcripts can go into non-adjacent partitions (e.g., partition-0: {T0,T11,T526...}, partition-1: {T1,T2,T87,T901...},...). This does not have any impact on quantification since each chip keeps track of its transcripts using the original transcript indices within the transcript dataset.

Since a partition of transcripts can have a different #transcript(segment)s than another, a large partition might require more than one CRAM-Seq chip to be stored. In the scale-out design, we pack most similar transcripts forming a partition in a single cluster of chips (where #chips in a cluster ≥ 1), and schedule reads with the same similarity signature as the partition to the corresponding cluster, thereby confining the processing of each read to a single cluster. Such clusters of chips are logical clusters where all chips within one (logical) cluster receive the same read for pseudo-alignment. If the transcript database and the corresponding partitions are changed, then simply updating the partition information in the read scheduler would suffice. Depending on the dataset, multiple partitions may reside in the same chip cluster, as well. Even then, the energy consumption per read wouldn't exceed the energy consumed in the cluster with maximum number of CRAM-Seq chips, as opposed to brute-force pseudo-alignment involving all chips in the system on a per read basis.

Cluster workload balance: As the entire pool of reads is available for quantification at once, groups of reads from the pool are considered for scheduling simultaneously (and in any order), and placed on a queue of length X in the read scheduler ($X > \#chip\_clusters$). The *sig-mer(s)* of the transcript partitions stored by all chips are known to the read scheduler, and are used to select a particular cluster of chips for a read, from these X reads, based on the presence of the *sig-mer(s)* within that read. The k for *sig-mer*s can be greater than the k used in PEs, for fine-grain partitioning of transcripts. The read scheduler, feeding from the *RNA-Seq read pool*, thereby can schedule different reads to different chip clusters, effectively increasing the throughput by a factor of $M = \#chip\_clusters$. Only the scheduled reads are removed from the queue (and the read pool), and the rest of the queue is filled up by reads from the pool, for the next scheduling cycle. With a practical (randomized) read pool and large enough X, it is highly likely that, within that window of X reads, distinct RNA-Seq reads would be scheduled to most (if not all) of the chip clusters, maximizing the utilization of the chip clusters. The hardware overhead (e.g., memory, logic) of the read scheduler is determined by X. Figure 16(b) illustrates the idea. Partitioning of the transcript dataset (i.e., transcriptome) along with selective scheduling enable ideal performance scaling without compromising energy efficiency.

IV. EVALUATION SETUP

To model a PE, a step-accurate simulator is used where each step corresponds to a logic function such as *AND*. Latency and energy for sense-amplifier based MAX operation, transistors and VTU are derived from HSPICE estimates. Peripheral overheads associated with PE are obtained from NVSIM [9], including parasitic effects. All estimates use 22 nm technology

IEEE TRANSACTIONS ON EMERGING TOPICS IN COMPUTING

TABLE 2. Technology Parameters.

Parameters	Value
MTJ Type	Interfacial PMTJ
MTJ Diameter (nm)	10
TMR (%)	100
RA Product ($\Omega\mu m^2$)	20
Critical Current I_{crit} (μA)	3.0 (SHE Channel)
Switching Latency (ns)	1
R_P, R_{AP} ($K\Omega$)	253.97, 507.94
R_{SHE}, $R_{trans.}$ ($K\Omega$)	64,1

TABLE 4. Kallisto Throughput (KSeq/s).

#Transcripts	Throughput
100	566.44
200	663.86
400	839.67
800	669.11
1000	653.29
3000	647.01
5000	622.57
8000	620.87

node. SHE-MTJ specific parameters come from [12], [20]. Latency and energy for CRAM logic functions are extracted from equivalent circuits as shown in Figure 4(c). Table 2 provides technology parameters.

PE specifics: Each tile is 128×128 that ensures signal integrity (both within the tile and across tiles in a PE), and enough tile-level parallelism while keeping the peripheral circuit overheads moderate. Each PE incorporates 32 tiles. Note that the number of PEs required is determined by the total number of transcript *segments*, therefore, the total number of PEs required may vary across different datasets even with the same number of transcripts. For simplicity we keep the transcripts in transcript datasets of particular sizes unchanged.

Chip size: Since each 2T(ransistor)1M(agnet) SHE-MTJ cell consumes $\sim 2\times$ area of a 1T1M STT-MTJ cell (where the number of transistors used dominates the cell area), we assume a maximum chip size of 64 MB, considering the maximum STT-MRAM chip size available commercially [10]. This chip size estimate is conservative based on the current state of SHE-MTJ technology, which is expected to improve as the technology matures.

Problem datasets: Table 3 lists the problem sizes used in the evaluation. To eliminate selection bias, for a dataset of M transcripts, the first M transcripts are taken from [13] (the nucleotide sequences of all transcripts in the reference chromosomes in a human) with lengths $\geq$ 100bp (read length). Transcript segments are selected to be maximum 200 bp long (with 100 bp overlap between successive segments). CRAM-Seq in this implementation uses 5-mers, i.e., 1024-bit vectors that ensure a sparse representation of transcript segments and RNA-Seq reads (unlike 4-mer) without a big memory footprint (unlike 6-mer). The number of RNA-Seq reads in read datasets is selected in proportion to the number of transcript segments in the transcript datasets. 5 read datasets from each transcript dataset are generated, by randomly sampling 100-bp long sub-sequences from the transcripts. Then, noise is added to the reads, assuming a single substitution rate of 0.13% and insertion/deletion error rates of 0.01% to imitate practical Illumina sequencing platforms. The maximum number of similarity classes supported by CRAM-Seq for different problem datasets is derived by profiling the datasets.

TABLE 3. Problem Sizes Evaluated.

#Transcripts	#Segments	#PE	#Reads ($\times 10^6$)	#Sim. Class
100	931	8	0.2	1000
200	2091	17	0.45	2000
400	4724	37	1.01	6000
800	10691	84	2.3	13000
1000 (default)	14687	115	3.2	17000
3000	46065	360	10.0	34000
5000	78144	611	17.1	60000
8000	123185	963	27.0	80000

Baseline for comparison: We use Kallisto [4], the state-of-the-art abundance quantification software which outperforms popular (exact and pseudo-alignment based) quantification approaches in terms of runtime and accuracy, and is widely adopted in stand-alone, as well as cloud based quantification [17], [18]. Kallisto uses a pre-generated colored *de Bruijn* graph (DBG) of k-mers present in the transcripts for similarity class detection, and EM algorithm for quantification. Note that, the graph structure in Kallisto is based on the *order* of the k-mers in transcripts, and the accuracy of quantification depends on storing this *order* information correctly. Therefore, Kallisto is significantly sensitive to k-mer length and their relative orders, unlike CRAM-Seq. There is no standard GPU baseline for abundance quantification, and alignment accelerators which map reads to (several orders of magnitude) longer references do not qualify as baselines without changing the designs to introduce problem-specific data structures. Since CRAM-Seq accelerates the pseudo-alignment part of quantification, for a fair comparison, we profile only the pseudo-alignment routine in Kallisto with 8 threads on an Intel i9-9900 system. Table 4 lists the pseudo-alignment throughput values derived from profiling. Note that, these values depend on: i) #transcripts, ii) the similarity among (and length of) these transcripts, and iii) the reads. To maximize the throughput of Kallisto, a cache-friendly read file organization (non-randomized, reads from a transcript are placed together) is used. The energy consumption of Kallisto is derived by feeding Kallisto's dynamic instruction mix to an optimistic energy model based on [15], [27].

Evaluation metrics: We characterize the performance of CRAM-Seq in terms of throughput in KSeq(uences)/s(econd) and energy efficiency in KSeq/s/mJ, and normalize to those of Kallisto. The accuracy of (per transcript) abundance estimates is calculated as the absolute difference between the

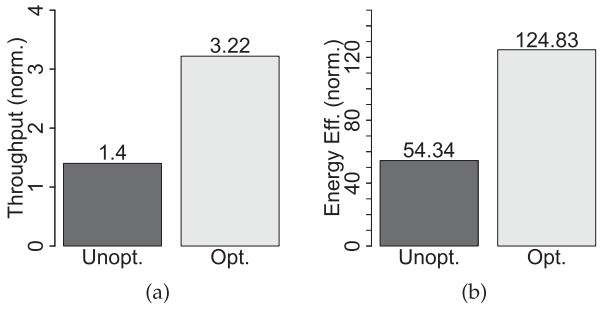

FIGURE 17. **CRAM-Seq (a) throughput and (b) energy efficiency.**

known abundance of the synthetic read datasets and the reported estimates (by CRAM-Seq and Kallisto), and expressed as a percentage of the known abundance (i.e., relative error). Although, both mean and median (of the) relative errors are used to evaluate the overall accuracy of quantification, unless otherwise specified, mean relative error is the default accuracy metric.

V. EVALUATION

A. PERFORMANCE ANALYSIS

Figure 17(a) illustrates the improvement in throughput exhibited by CRAM-Seq, over Kallisto, with 1000 transcripts. The unoptimized column represents the naive design, without any optimization, as explained in Section III. The corresponding improvement in energy efficiency, over Kallisto, is shown in Figure 17(b). Even with an unoptimized design, CRAM-Seq outperforms Kallisto on both throughput ($1.4\times$) and energy efficiency ($\sim 54\times$) fronts. The throughput of CRAM-Seq is determined by the PE latency and the class extraction (*MAX*) where PE operations make up for $\sim 98\%$ of the total latency, as shown in Figure 18(a). The VTU and SCU components are hidden (pipelined) and have lower latency in comparison. Figure 18(b) captures the relative contributions to energy of PE (including MAX) and SCU operations (share of the VTU is negligibly small) consuming $>99\%$ of total CRAM-Seq energy.

To improve the throughput of CRAM-Seq, we examine optimization opportunities in PE operations since PE operations dominate the latency of the CRAM-Seq pipeline stages. Breakdowns of unoptimized PE latency and energy, illustrated in Figure 19(s) (a) and (b), respectively, reveals that PE latency is dominated by the *preset* ($\sim 50\%$ of total

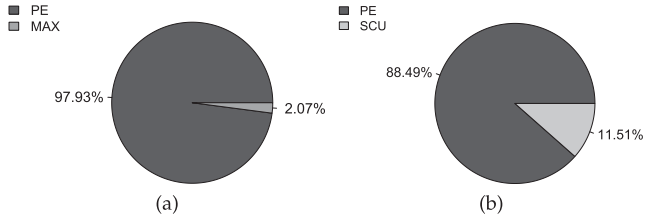

FIGURE 18. **CRAM-Seq (a) latency and (b) energy breakdown.**

and $\sim 1.5\times$ of compute since 2-output *INV* logic requires $2\times$ more *preset* per logic operation), an enabler step to perform the actual computations, while the actual computations, e.g., *AND* operations, consume a mere one-third of the total. The presets in the unoptimized design are performed in a sequential manner right before a logic operation, although in multiple columns at the same time. Luckily, *gang-preset*s are possible (where target output cells in multiple rows and multiple columns are preset at the same time) due to the low *preset* energy, coupled with separate read and write paths of SHE-MTJ cells that translate into a peak current draw of ~ 12 mA per tile. This optimization reduces the share of the preset latency in overall PE latency by $>95\%$, as illustrated in Figure 19(c). The corresponding change in throughput is reflected in the optimized column (Figure 17(a)) that shows $>3\times$ throughput improvement over the baseline, i.e., a $\sim 130\%$ increase in throughput over the unoptimized design. Since the total number of presets (and the corresponding energy) in the optimized design is unchanged, the improvement over unoptimized design in terms of the energy efficiency is similar to that of the throughput, as observed in Figure 17(b). These improvements clearly demonstrate the benefits of data communication reduction (vs. Kallisto), and efficient *in-situ* computation in CRAM-Seq.

B. SCALABILITY

Performance scaling within a single chip: Figure 20 illustrates how CRAM-Seq performance scales with the increasing number of transcripts per CRAM-Seq chip. Throughput remains stable as more transcript segments are processed by (i.e., more PEs are added to) CRAM-Seq. Such weak scaling reduces the energy efficiency improvement which decreases linearly with growing #transcripts. Energy for *class ID* check increases quickly with higher number of transcripts (e.g., $\sim 20\%$ of the total energy with 3000 transcripts) due to the

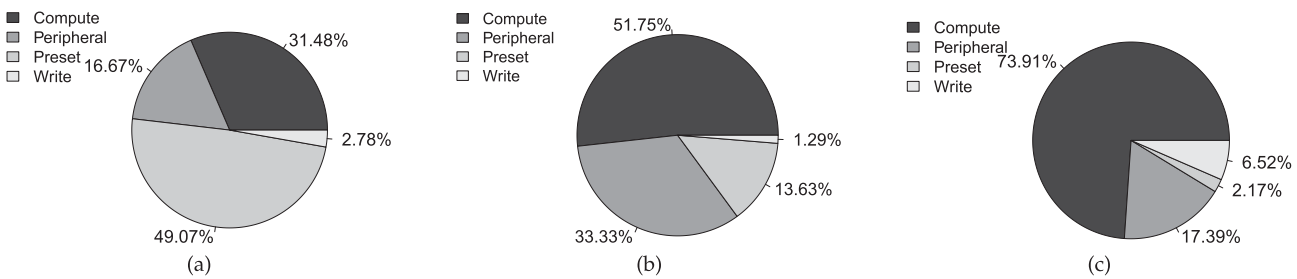

FIGURE 19. **(a) Latency and (b) energy breakdowns for unoptimized PE; (c) Latency breakdown for optimized PE.**

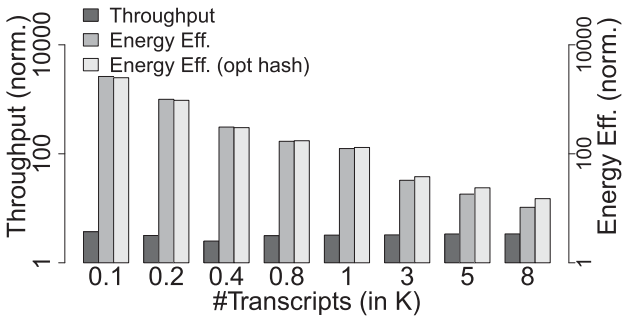

FIGURE 20. Performance scaling of CRAM-Seq.

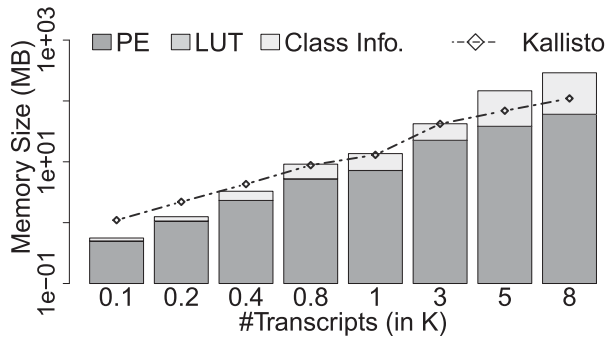

FIGURE 21. Memory required for CRAM-Seq and Kallisto.

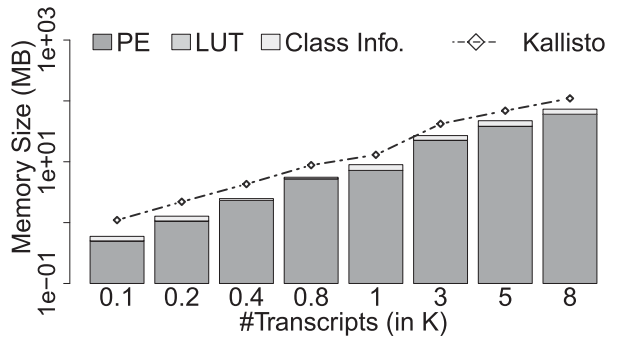

FIGURE 22. Memory required for CRAM-Seq with optimized hash.

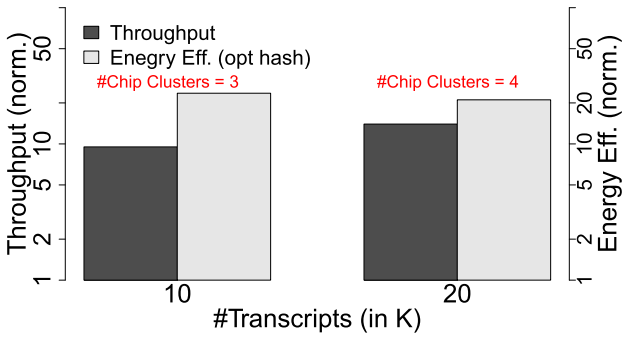

FIGURE 23. Throughput and energy efficiency of a system of CRAM-Seq chips.

naive one-hot encoding of *class IDs*. Even then, with 3000 transcripts (maximum number of segments that fits on chip) CRAM-Seq still provides $>30X$ improvement in energy efficiency over the baseline.

The *opt hash* bars in Figure 20 capture the impact of optimized hashing (Algorithm 3). Optimized hashing makes the energy efficiency scale better (vs. the naive one-hot encoding) due to the lower #bits to encode the *class ID*, which reduces the energy consumption in the SCU. Moreover, optimized hashing itself scales better as the number of transcripts per chip increases. The maximum number of transcripts per chip also improves due to the more compact representation of *class ID*, as we will detail next. That all said, hashing in this context still has room for improvement (e.g., a hashing algorithm that converts a set of transcript indices to a unique value in a fixed 64-bit number space).

Memory scaling: The memory sizes for CRAM-Seq (and the baseline) with varying number of transcripts are shown in Figure 21. For Kallisto, only the memory required for transcript index file is considered– to favor the baseline. PEs in CRAM-Seq, which store the transcript segments, require less memory than Kallisto (with default 31-mer) for all datasets, making CRAM-Seq more memory efficient for transcript storage even though CRAM-Seq stores multiple segments for each transcript. The contribution of LUT in system size is negligible. However, the class information storage in SCU increases quickly and with >3000 transcripts, total CRAM-Seq storage ($\sim$60 MB) exceeds the Kallisto transcript index file size due to inefficient encoding of *similarity class IDs*. Optimized hashing (Algorithm 3) reduces memory overhead of SCU significantly, as shown in Figure 22. With optimized hashing, more (up to 5000) transcripts (i.e., more PEs) fit on a single chip, at the same time, SCU energy reduces accordingly. Hence, the total memory requirement still remains less than Kallisto's. This trend holds in a system of multiple CRAM-Seq chips, as well.

Performance scaling across multiple chips: Figure 23 captures the impact of scaling out a system of CRAM-Seq chips, following the methodology from Section III-K, to handle larger transcriptomes. In this case, we assume the same number of PEs in each CRAM-Seq chip (corresponding to the 5 K transcript dataset). We experiment with transcriptomes of 10 K and 20 K. The read scheduler considers 10 (=X) reads at a time and tries to schedule one distinct read to each cluster of chips where the most similar transcripts partition to the read resides. As Figure 23 suggests, the mean throughput keeps improving with the increasing number of transcripts in the dataset. The theoretical *peak* throughput is limited by a factor equal to #clusters of chips in the system (=3 for 10 K and 4 for 20 K), whereas the minimum throughput is equal to that of a single CRAM-Seq chip. The energy efficiency remains stable due to balancing between increasing energy consumption and improvement in throughput. It is possible to further improve the energy efficiency by optimizing partitioning of transcripts (i.e., changing similarity threshold [32])–to minimize the partition sizes and consequently, have smaller cluster of chips to avoid unnecessary computation within a cluster.

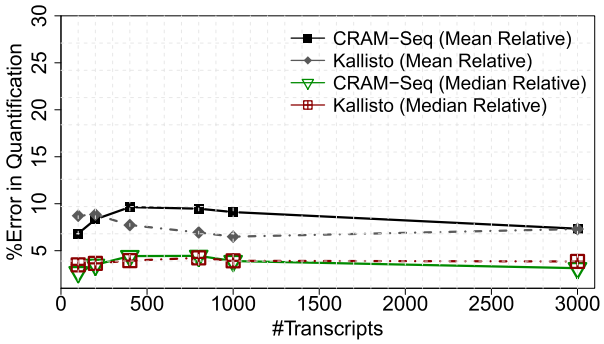

FIGURE 24. **Accuracy of quantification.**

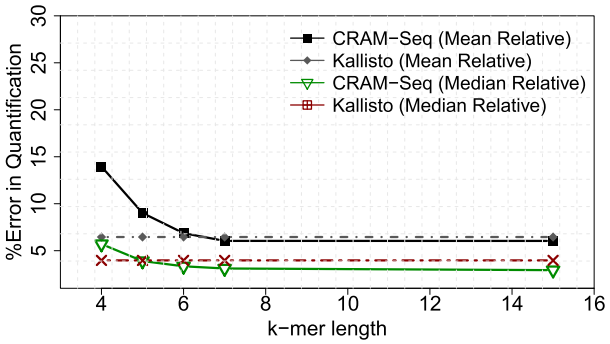

FIGURE 25. **Variation in CRAM-Seq accuracy with k-mer length.**

C. ACCURACY ANALYSIS

Figure 24 shows the trend in accuracy, i.e., mean % (relative) error in quantification for all datasets, up to 3000 transcripts (i.e., the single chip limit with naive SCU encoding scheme). The error reported by both Kallisto and CRAM-Seq remains $< 10\%$ across all datasets. Although CRAM-Seq utilizes smaller k-mers than Kallisto, the mean difference between reported errors is 0.78% (with the maximum reaching 2.61%) across all datasets. The mean relative error metric is not robust against local outliers (i.e., large difference between per-transcript actual and the estimated abundance values), leading to the $\sim$3% loss shown for a few transcript dataset sizes (0.4 K, 0.8 K, 1 K) in Figure 24. Pearson's correlation coefficient [3], a more robust metric against local outliers in capturing correlation between the ground truth and estimated abundances, is 0.9822633935 for CRAM-Seq, and 0.9966648441 for Kallisto (for 1 K case). These values indicate very strong correlations between the actual abundance and the estimates from both CRAM-Seq and Kallisto, even for the cases showing some sizable difference in accuracy – as the Pearson's coefficient is >0.9 for both CRAM-Seq and Kallisto, the difference (between these coefficients) of 0.014 does not bear any significance. Furthermore, with more robust accuracy metrics such as median relative error, which provide more insight about the center of the abundance distribution (i.e., objective of abundance quantification), the accuracy of CRAM-Seq and Kallisto are very similar (e.g., 3.9% in CRAM-Seq vs. 4.1% in Kallisto, with 1 K transcripts), as shown in Figure 24. Overall, considering mean relative error, the accuracy of CRAM-Seq and Kallisto are nearly identical. Having said that, irrespective of the metric used, the contribution of these infrequent outliers is expected to be insignificant due to much higher number of transcripts in practical datasets. The trend in accuracy holds beyond 3000 transcripts, as well. For 5000 transcripts, CRAM-Seq experiences 8.07% of mean relative error as opposed to 8.11% by Kallisto.

Our experiments show that the accuracy of Kallisto with 5-mers is much worse ($>90\%$) than CRAM-Seq, attributable to the much smaller (than transcripts) k-mer length used in building the DBG, where segmented transcripts used in CRAM-Seq help improve accuracy.

Figure 25 illustrates the impact of k on the accuracy. For 1000 transcripts and respective read datasets, we vary k and report accuracy. The mean relative error in quantification reported by Kallisto is for the default k (=31). For 4-mers, the number of distinct k-mer patterns are only 256 (not significantly higher than the RNA-Seq read length assumed), which degrades accuracy and renders a mean relative error of $>10\%$, about 2X of that of Kallisto. The accuracy improves significantly as k increases to 5, considering both mean and median relative error metrics. As explained in Section III-A, 5-mers result in 1024 distinct k-mer patterns which is sufficient to correlate reads with individual transcript(s). However, increasing k in CRAM-Seq beyond 5 does not improve accuracy by a significant margin, but decreases throughput and increases energy consumption (Table 5; with mean relative error). The improvement in accuracy becomes increasingly smaller with increasing k due to larger PEs (more CRAM tiles for each PE to accommodate longer bit-vectors resulting from longer k-mers, translating into more computation to derive similarity scores) which also increases the memory footprint of CRAM-Seq.

VI. RELATED WORK

CRAM-Seq is the first to accelerate RNA-Seq abundance quantification on a PIM substrate. A typical RNA-Seq abundance quantification algorithm entails alignment of each read from a set of (usually millions of) reads to a large number of long transcripts, followed by EM which quantifies abundance using the alignment outcome. CPU based exact alignment is too time consuming. Many GPU [19] and FPGA based implementations [2], [11], [23], [26] to accelerate popular exact sequence alignment algorithms such as Smith-Waterman

TABLE 5. **Impact of k on Quantification Error, Throughput, Energy Efficiency and Memory Size (Relative to k=4). δ in %.**

k	δ Error	δ Throughput	δ Efficiency	δ Memory
5	−35.42	−6.1	−6.3	+35.8
6	−50.81	−11.4	−12.0	+107.4
7	−56.70	−16.2	−16.9	+179.0
15	−56.72	−41.5	−43.2	+680.1

exist. PIM based exact alignment is also gaining traction due to potentially higher alignment throughput and lower energy consumption than both GPU and FPGA. Recent designs that use SOT-MRAM [1], [7], ReRAM [14], [33] and conventional technology [5] have reported significant throughput and energy improvements over GPU based implementations. However, exact alignment is redundant for abundance quantification simply because the exact location of alignment is not required. Therefore, such PIM based exact alignment accelerators would result in sizable latency and energy overhead if employed in the abundance quantification pipeline. Moreover, unlike typical alignment accelerators, abundance quantification involves matching against multiple reference (i.e., transcript) sequences simultaneously as opposed to a very long one. To summarize, PIM-based based alignment accelerators, in principle, can be deployed to build a RNA-Seq abundance quantification accelerator using the design principles presented in this paper. However, such designs cannot be used off-the-shelf for quantification without design modifications which would warrant separate end-to-end accelerator designs. Therefore, such designs are not considered as baselines in this paper. The pseudo-alignment algorithms such as Kallisto [4] and Sailfish [22], on the other hand, rely on *order* or *frequency* of k-mer patterns that are common in both read and transcripts to quantify abundance. Matataki [21], another application that utilizes k-mer based approximation, is highly susceptible to errors in RNA-Seq reads and therefore limited to large-scale reanalysis, unlike CRAM-Seq that is impacted minimally by errors in reads due to the use of small k-mers. While these CPU based software solutions provide significant speedup over exact alignment based quantification, they suffer from significant data movement overhead, which is addressed by CRAM-Seq effectively.

VII. CONCLUSION

RNA-Seq abundance quantification is an essential tool in gene expression analysis, useful in many applications such as differential expression analysis to infer biological function at the gene/transcript level. As a compute intensive application (due to the use of millions of RNA-Seq reads), it suffers from significant data movement overheads in classical von Neumann systems. In this paper, we propose computational RAM (CRAM) as a PIM substrate to accelerate abundance quantification. The resulting design, CRAM-Seq, performs the pseudo-alignment steps in CRAM. We demonstrate the key design challenges and opportunities to achieve a significant improvement in throughput and energy efficiency, while maintaining a similar level of accuracy when compared to the state-of-the-art RNA-Seq abundance quantification algorithm, Kallisto.

REFERENCES

[1] S. Angizi, J. Sun, W. Zhang, and D. Fan, "AlignS: A processing-in-memory accelerator for DNA short read alignment leveraging SOT-MRAM," in *Proc. 53nd ACM/IEEE Des. Automat. Conf.*, 2019, pp. 1–6.

[2] J. Arram, T. Kaplan, W. Luk, and P. Jiang, "Leveraging FPGAs for accelerating short read alignment," *IEEE/ACM Trans. Comput. Biol. Bioinf.*, vol. 14, no. 3, pp. 668–677, May/Jun. 2017.

[3] J. Benesty, J. Chen, Y. Huang, and I. Cohen, "Pearson correlation coefficient," in *Noise Reduction in Speech Processing*. Berlin, Heidelberg: Springer, 2009, pp. 1–4.

[4] N. L. Bray, H. Pimentel, P. Melsted, and L. Pachter, "Near-optimal probabilistic RNA-seq quantification," *Nature Biotechnol.*, vol. 34, no. 5, pp. 525–527, 2016.

[5] D. S. Cali *et al.*, "GenASM: A high-performance, low-power approximate string matching acceleration framework for genome sequence analysis," in *Proc. 53rd Annu. IEEE/ACM Int. Symp. Microarchit.*, 2020, pp. 951–966.

[6] Z. Chowdhury *et al.*, "Efficient in-memory processing using spintronics," *IEEE Comput. Archit. Lett.*, vol. 17, no. 1, pp. 42–46, Jan.–Jun. 2018.

[7] Z. I. Chowdhury *et al.*, "A DNA read alignment accelerator based on computational RAM," *IEEE J. Explor. Solid-State Computat.*, vol. 6, no. 1, pp. 80–88, Jun. 2020.

[8] A. P. Dempster, N. M. Laird, and D. B. Rubin, "Maximum likelihood from incomplete data via the EM algorithm," *J. Roy. Stat. Soc. Ser. B. Stat. Methodological*, vol. 39, no. 1, pp. 1–22, 1977.

[9] X. Dong, C. Xu, Y. Xie, and N. P. Jouppi, "NVSim: A circuit-level performance, energy, and area model for emerging nonvolatile memory," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 31, no. 7, pp. 994–1007, Jul. 2012.

[10] Everspin Technologies, EMD4E001G, pp. 1–174, 2020. [Online]. Available: "www.everspin.com/spin-transfer-torque-mram-products"

[11] X. Fei, Z. Dan, L. Lina, M. Xin, and Z. Chunlei, "FPGASW: Accelerating large-scale Smith-Waterman sequence alignment application with backtracking on FPGA linear systolic array," *Interdiscipl. Sci.: Comput. Life Sci.*, vol. 10, no. 1, pp. 176–188, 2018.

[12] K. Garello *et al.*, "Ultrafast magnetization switching by spin-orbit torques," *Appl. Phys. Lett.*, vol. 105, no. 21, 2014, Art. no. 212402.

[13] J. Harrow *et al.*, "GENCODE: The reference human genome annotation for the ENCODE project," *Genome Res.*, vol. 22, no. 9, pp. 1760–1774, 2012.

[14] W. Huangfu, S. Li, X. Hu, and Y. Xie, "Radar: A 3D-reram based DNA alignment accelerator architecture," in *Proc. 55th Annu. Des. Automat. Conf.*, 2018, pp. 1–6.

[15] S. W. Keckler, W. J. Dally, B. Khailany, M. Garland, and D. Glasco, "GPUs and the future of parallel computing," *IEEE Micro*, vol. 31, no. 5, pp. 7–17, Sep./Oct. 2011.

[16] D. Kim, G. Pertea, C. Trapnell, H. Pimentel, R. Kelley, and S. L. Salzberg, "TopHat2: Accurate alignment of transcriptomes in the presence of insertions, deletions and gene fusions," *Genome Biol.*, vol. 14, no. 4, pp. 1–13, 2013.

[17] A. Lachmann, D. J. Clarke, D. Torre, Z. Xie, and A. Ma'ayan, "Interoperable RNA-Seq analysis in the cloud," *Biochimica Biophysica Acta-Gene Regulatory Mechanisms*, vol. 1863, no. 6, 2020, Art. no. 194521.

[18] A. Lachmann, Z. Xie, and A. Ma'ayan, "Elysium: RNA-seq Alignment in the cloud," *bioRxiv*, 2018, Art. no. 382937.

[19] C.-M. Liu *et al.*, "SOAP3: Ultra-fast GPU-based parallel alignment tool for short reads," *Bioinformatics*, vol. 28, no. 6, pp. 878–879, 2012.

[20] D. Mahendra *et al.*, "Room-temperature high spin-orbit torque due to quantum confinement in sputtered $Bi_xSe_{(1-x)}$ films," *Nature Mater.*, vol. 17, no. 9, pp. 800–807, 2018.

[21] Y. Okamura and K. Kinoshita, "Matataki: An ultrafast mRNA quantification method for large-scale reanalysis of RNA-Seq data," *BMC Bioinf.*, vol. 19, no. 1, pp. 1–9, 2018.

[22] R. Patro, S. M. Mount, and C. Kingsford, "Sailfish enables alignment-free isoform quantification from RNA-seq reads using lightweight algorithms," *Nature Biotechnol.*, vol. 32, no. 5, pp. 462–464, 2014.

[23] C. Pham-Quoc, B. Kieu-Do, and T. N. Thinh, "A high-performance FPGA-based BWA-MEM DNA sequence alignment," *Concurrency Comput.: Pract. Experience*, vol. 33, no. 2, 2019, Art. no. e5328.

[24] S. Resch *et al.*, "PIMBALL: Binary neural networks in spintronic memory," *ACM Trans. Archit. Code Optim.*, vol. 16, no. 4, pp. 1–26, Oct. 2019.

[25] S. Rodrigue *et al.*, "Unlocking short read sequencing for metagenomics," *PLoS One*, vol. 5, no. 7, 2010, Art. no. e11840.

[26] E. Rucci, C. Garcia, G. Botella, A. De Giusti, M. Naiouf, and M. Prieto-Matias, "SWIFOLD: Smith-waterman implementation on FPGA with opencl for long DNA sequences," *BMC Syst. Biol.*, vol. 12, no. 5, pp. 43–53, 2018.

[27] Y. S. Shao and D. Brooks, "Energy characterization and instruction-level energy model of intel's Xeon Phi processor," in *Proc. Int. Symp. Low Power Electron. Des.*, 2013, pp. 389–394.

[28] Z. D. Stephens *et al.*, "Big Data: Astronomical or genomical?," *PLoS Biol.*, vol. 13, no. 7, 2015, Art. no. e1002195.

[29] C. Trapnell *et al.*, "Transcript assembly and quantification by RNA-Seq reveals unannotated transcripts and isoform switching during cell differentiation," *Nature Biotechnol.*, vol. 28, no. 5, p. 511, 2010.
[30] J.-P. Wang and J. D. Harms, "General structure for Computational Random Access Memory (CRAM)," U.S. Patent 9,224,447, Dec. 29, 2015.
[31] M. Zabihi *et al.*, "Using spin-hall MTJs to build an energy-efficient in-memory computation platform," in *Proc. 20th Int. Symp. Qual. Electron. Des.*, 2019, pp. 52–57.
[32] Z. Zhang and W. Wang, "RNA-Skim: A rapid method for RNA-Seq quantification at transcript level," *Bioinformatics*, vol. 30, no. 12, pp. i283–i292, 2014.
[33] F. Zokaee, H. R. Zarandi, and L. Jiang, "Aligner: A. process-in-memory architecture for short read alignment in ReRAMs," *IEEE Comput. Archit. Lett.*, vol. 17, no. 2, pp. 237–240, Jul.–Dec. 2018.

Zamshed I. Chowdhury received the BSc and MS degrees in applied physics, electronics and communication engineering from the University of Dhaka, Bangladesh. He is currently working toward the PhD degree with the Department of Electrical and Computer Engineering, University of Minnesota, Twin Cities, USA. He is a faculty member (on leave) with Jahangirnagar University, Bangladesh. His primary research interests include in-memory computing, hardware accelerator design, and approximate computing.

S. Karen Khatamifard received the BSc degree in electrical engineering from the Sharif University of Technology, Tehran, Iran, in 2013. He is currently working toward the PhD degree with the Department of Electrical Engineering, University of Minnesota, Minneapolis. His primary research interests include improving energy efficiency of computing, designing application-specific hardware accelerators, approximate computing, and reliability implications of process technology scaling.

Salonik Resch received the MS degree in electrical engineering from the University of Minnesota, where he is currently working toward the PhD degree. His research interests include quantum computing, processing in memory, and intermittent computing.

Hüsrev Cılasun is currently working toward the PhD degree with the University of Minnesota. Between 2016 and 2019, he was with Aselsan Inc. He has authored several conference and journal papers in his research field, which include computer architecture, FPGA prototyping, digital signal, and image processing.

Zhengyang Zhao received the BS degree in electrical engineering from Xi'an Jiaotong University, China. He is currently working toward the PhD degree in electrical and computer engineering with the University of Minnesota, Minneapolis, MN. His research focuses on development of novel spintronic devices to implement energy-efficient memory cells and logic applications. His recent work includes studying current-induced magnet reversal using spin-orbit torque (SOT), and also voltage-induced magnet reversal using piezoelectric strain or VCMA effect. More specific work includes the stack design, MTJ cell nanofabrication, advanced device characterization and physics study.

Masoud Zabihi (Graduate Student Member, IEEE) received the BSc degree in electrical engineering and electronics from the University of Tabriz in 2010, and the and MS degree in electrical engineering and electronics from the Sharif University of Technology in 2013. He is currently working toward the PhD degree in electrical engineering with the University of Minnesota. His research interests include in-memory computing and spintronic based memory technologies.

Meisam Razaviyayn (Member, IEEE) received the PhD degree in electrical engineering with minor in computer science from the University of Minnesota. He is currently an assistant professor with the Department of Industrial and Systems Engineering, University of Southern California (USC), with courtesy appointments at the electrical engineering and computer science departments. Prior to joining USC, he was a postdoctoral research fellow with the Department of Electrical Engineering, Stanford University. Dr. Razaviyayn was the recipient of the Signal Processing Society Young Author Best Paper Award in 2014, Best Paper Award in IEEE Data Science Workshop in 2019, ICCM Best Paper Award in Mathematics in 2020, and 3M's Non-Tenured Faculty Award in 2021. He was the finalist for Best Paper Prize for Young Researcher in Continuous Optimization in 2013 and 2016.

Jian-Ping Wang (Fellow, IEEE) is currently the Robert F. Hartmann chair and a Distinguished McKnight University professor of electrical and computer engineering with the University of Minnesota. He is the director of the Center for Spintronic Materials for Advanced Information Technology (SMART), one of two SRC/NIST nCORE research centers. He was the director of the Center for Spintronic Materials, Interfaces and Novel Architectures (C-SPIN). CSPIN was one of six SRC/DARPA STARnet program centers. His inventions have been used in both HDD products and STT-RAM products. He was the recipient of the Information Storage Industry Consortium (INSIC) Technical Award in 2006 for his pioneering work in exchange coupled composite magnetic media. He was the recipient of the 2019 SRC Technical Excellence Award for his innovations and discoveries in nanomagnetics and novel materials that accelerated the production of magnetic random-access memories. He is an APS fellow.

IEEE TRANSACTIONS ON
EMERGING TOPICS
IN COMPUTING

Sachin S. Sapatnekar (Fellow, IEEE) received the BTech degree from the Indian Institute of Technology, Bombay, the MS degree from Syracuse University, and the PhD degree from the University of Illinois. He taught with Iowa State University from 1992 to 1997 and has been with the University of Minnesota since 1997, where he holds the Distinguished McKnight University professorship and the Robert and Marjorie Henle chair with the Department of Electrical and Computer Engineering. He was the recipient of seven conference best paper awards, the Best Poster Award, two ICCAD 10-year Retrospective Most Influential Paper awards, SRC Technical Excellence Award, and SIA University Researcher Award. He is a fellow of ACM.

Ulya R. Karpuzcu (Member, IEEE) received the MS and PhD degrees in computer engineering from the University of Illinois, Urbana-Champaign. She is currently an associate professor with the Department of Electrical and Computer Engineering, University of Minnesota. Her research interests include the impact of technology on computing, energy-efficient computing, application domain specialized architectures, approximate computing, and computing at ultra-low voltages.