A Customized Processing-in-Memory Architecture for Biological Sequence Alignment

Nasrin Akbari[1], Mehdi Modarressi[1,2], Masoud Daneshtalab[3]

[1] Department of Electrical and Computer Engineering, College of Engineering, University of Tehran, Tehran, Iran.
[2] School of Computer Science, Institute for Research in Fundamental Sciences (IPM), Tehran, Iran.
[3] Mälardalen University (MDH) and Royal Institute of Technology (KTH), Sweden

nasrin.akbari@ut.ac.ir, modarressi@ut.ac.ir, masoud.daneshtalab@mdh.se

***Abstract*—Sequence alignment is the most widely used operation in bioinformatics. With the exponential growth of the biological sequence databases, searching a database to find the optimal alignment for a query sequence (that can be at the order of hundreds of millions of characters long) would require excessive processing power and memory bandwidth. Sequence alignment algorithms can potentially benefit from the processing power of massive parallel processors due their simple arithmetic operations, coupled with the inherent fine-grained and coarse-grained parallelism that they exhibit. However, the limited memory bandwidth in conventional computing systems prevents exploiting the maximum achievable speedup. In this paper, we propose a processing-in-memory architecture as a viable solution for the excessive memory bandwidth demand of bioinformatics applications. The design is composed of a set of simple and light-weight processing elements, customized to the sequence alignment algorithm, integrated at the logic layer of an emerging 3D DRAM architecture. Experimental results show that the proposed architecture results in up to 2.4x speedup and 41% reduction in power consumption, compared to a processor-side parallel implementation.**

***Keywords*—Sequence Alignment, Accelerator, Processing-in-memory.**

I. Introduction

Aligning two biological sequences in a pairwise manner is the primary operation in many computational biology and genomics problems. The sequences can be a chain of amino acids that make a protein sequence or an ordered set of nucleotides that form a DNA [1].

Among various solutions, dynamic programming-based methods make a better compromise between accuracy and speed, and thereby, has gained popularity [1][2][3]. Dynamic programming-based implementations, as will be discussed shortly, involve simple character comparison and low-precision integer arithmetic operations. However, the entire alignment procedure can be quite time-consuming, as it has to handle extremely large datasets.

First, a biological sequence can be very long: the size of protein sequences can be as large as tens of thousands of amino acids (characters) and the sizes of DNA sequences can range from a few to hundreds of millions of nucleotides (characters) [4]. For example, the longest human sequence in the related databases is about 249 million nucleotides long [4]. Further, genomic databases against which a given sequence should be aligned are growing at an exponential rate and major widely-used public databases currently contain up to 200 million sequences [5].

This huge amount of data makes the sequence alignment a heavy and expensive task, in terms of the bandwidth and processing power demand.

The simple logic and inherent parallelism of the sequence alignment problem can be exploited to accelerate them on massive parallel processors. The parallelism can be either fine-grained (by parallelizing the alignment algorithm [4]) or coarse-grained (by concurrent alignment of a given sequence to multiple sequences of a database).

Prior work have explored the speedup of alignment algorithms on various hardware platforms, including application-specific architectures, graphical processing units (GPUs), and high-performance clusters [1][6]. Particularly, the inexpensive simple processing units of the problem enable designers to integrate a large amount of processing units into a customized accelerator chip to fully exploit the potential coarse- and fine-grained parallelism of sequence alignment algorithms.

However, the expected speedup can hardly be achieved due to the limited insufficient memory bandwidth in conventional processing architectures [7]. In particular, low operational density (operations per byte) is the key reason for the excessive memory bandwidth demand of bioinformatics algorithms. Moreover, the lack of temporal locality makes the cache less effective in reducing the bandwidth demand.

To tackle this bandwidth challenge, this paper presents a novel processing-in-memory (PIM) architecture for the sequence alignment problem. The architecture consists of a set of processing units, customized for the sequence alignment processing requirements, stacked on top of multiple layers of DRAM in a 3D fashion.

PIM minimize data movement and access latency by moving the computation closer to data. In addition to access latency, PIM also increases memory bandwidth by stacking the memory and logic layers on top of each other and providing data through high bandwidth vertical links.

Taking advantage of abundant memory bandwidth, a massive parallel accelerator at the logic layer of a 3D memory chip can push the bandwidth wall to exploit much higher potential parallelism of the alignment problem, effectively yielding higher speedup.

Some major memory chip vendors already ship 3D-memory chips with an integrated logic layer [8][9]. The logic layer often implements memory controller and, in some recent versions (such as HMC 2.0), can execute a set of simple in-memory instructions [9].

The memory-side accelerator presented in this paper consists of a set of processing elements and a programmable address generator logic that is programmed by the host processor to automatically direct data to the right memory-side processing element. Experimental results show it can considerably increase the performance and reduce the power consumption.

The rest of the paper is organized as follows. Section 2 covers important background and related studies. Section 3 describes the proposed PIM-based design. Section 4 outlines the implementation and evaluation methodology, followed by experimental results in Section 5. Finally, Section 6 concludes the paper.

II. Background

A. *Pairwise sequence alignment*

Biological sequences can be either a chain of nucleotides that make DNA sequences or a chain of amino acids that form protein sequences [4]. DNA sequences are composed of four types of nucleotides, but protein sequences are composed of 20 types of amino acids. We refer to amino acids and nucleotides as characters, hereinafter.

Pairwise global sequence alignment is the most basic operation in many bioinformatics applications that aims to find the most similar sequence of a database to a given query sequence [1][3][6][10]. It compares every pair of the characters of the sequences and assigns a predefined score to them, which are then summed up to get the alignment score of the two sequences. Score for each pair of aligned characters are got from a score matrix.

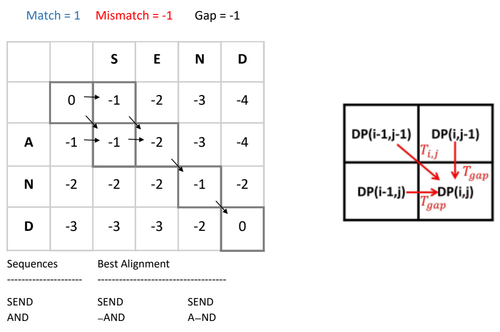
The strings can also be extended by so called gap (blank) characters, which can be inserted at any position in the strings to get a better alignment. The sequence alignment problem then aims at finding the best alignment (with the highest possible score) of two sequences of characters by appropriately inserting the gap characters in either sequence.

The Needleman-Wunsch algorithm is the most well-known solution for the pairwise sequence alignment problem [3].

Using a dynamic programing approach, computing an optimal alignment between two sequences A=(a1a2…am) of length m, and B= (b1b2…bn) of length n involves computing a (m+1)×(n+1) matrix, called dynamic programming (DP) matrix, in two passes.

In the forward pass, the DP matrix it iteratively filled from cell (0,0) to cell (m,n), with the value of each cell DP(i,j) is computed based on the values at some neighboring cells. Actually, DP(i,j) keeps the highest alignment score of partial sequences a1a2…ai of A and b1b2…bj of B (see Figure 1.a) [10].

In the DP matrix, as shown in Figure 1.b, the value of a cell is computed based on its north, west, and northwest neighbors as [10]:



(a) (b)

Fig. 1. (a) Dynamic programming matrix for alignment of “SEND” and “AND” sequences and (b) maximum score propagation flow

$$DP(i,j)=MAX\begin{cases} \mathrm{DP}(i-1,j-1)+\mathrm{T}_{i,j} \\ \mathrm{DP}(i-1,j)\ +\mathrm{T}_{gap} \\ \mathrm{DP}(i,j-1)\ +\mathrm{T}_{gap} \end{cases} \tag{1}$$

The flow depicted in Figure 1.b fills the DP cells row-by-row until reaching to cell DP(m,n). To track the alignment with the highest score, each cell keeps a backward pointer to the cell from which the highest score is received. The pointers are stored in another matrix, called direction matrix (Dir). By traversing the pointers in backward direction from Dir(m,n) to Dir(0,0), the best alignment is formed.

B. *Processing in memory*

Our proposed method is based on the Micron’s Hybrid Memory Cube (HMC) architecture, which is a complete example of a 3D stacked DRAM with an integrated logic layer [9]. Thanks to the recent advances in semiconductor fabrication, which allows integrating dies with different technologies on top of each other in a 3D fashion, the logic layer can benefit from the abundant memory bandwidth provided by the high-speed TSVs that interconnect the layers. HMC can have up to 8 DRAM layers for a total of 8GB capacity. Each DRAM layer is divided into 32 partitions and the vertically adjacent partitions (that form a column of vertically stacked partitions) are called a vault. Each vault has its own DRAM controller, or vault controller, implemented on its logic die. Vaults act as independent memory channels and can be accessed simultaneously.

HMC is connected to the host processor through up to four 16-bit full-duplex serial links. The logic layer provides a crossbar switch that connects the links to vault controllers. The processor-memory communication in HMC is carried out by a packet-based approach, in which each memory request/response in the form of packets.

In addition to implementing memory controller, HMC 2.0 implements several simple in-memory arithmetic and logic instructions at its logic layer that can be called by the host processor by some specific packet types. In this paper, we extend the PIM capability of HMC by adding sequence alignment operations to it.

III. Proposed Processing in Memory Architecture

GPGPUs, as the most efficient class of massive parallel machines, have been extensively used to solve the sequence alignment problem in several previous studies [1]. However, several previous works show that FPGA- and ASIC-based application-specific architectures that is designed to fit the specific computation and data movement pattern of the alignment algorithm can outperform a GPU by up to 15x [11]. As a result, this paper proposes an application-specific architecture, as a replacement for GPGPUs, for the sequence alignment problem.

An ideal accelerator has sufficient processing power to exploit the maximum available parallelism in the alignment algorithm: it calculates each DP matrix cell immediately after the three surrounding cells are calculated (See Figure 1.b). The maximum number of cells that can be calculated in parallel varies between one and the sequence size.

Using 32-bit data to keep DP cells and 2-bit data to keep sequence letters, our evaluation shows that each pairwise alignment task between two sequences requires about 18GB/s memory bandwidth. This evaluation assumes (1) 1GHz clock frequency for the accelerator and (2) single cycle DP matrix calculation.

Obviously, this high bandwidth demand of a single cell prohibits extracting the inherent parallelism of the sequence alignment algorithm. This bandwidth demand exceeds the maximum bandwidth of many conventional DDR generations and even modern memory technologies; this lengthens the alignment process, particularly when a query is to be aligned with a large database of reference sequences. By moving the alignment process closer to the memory, as we will show shortly, we can push this bandwidth wall and narrows the gap to the maximum achievable performance.

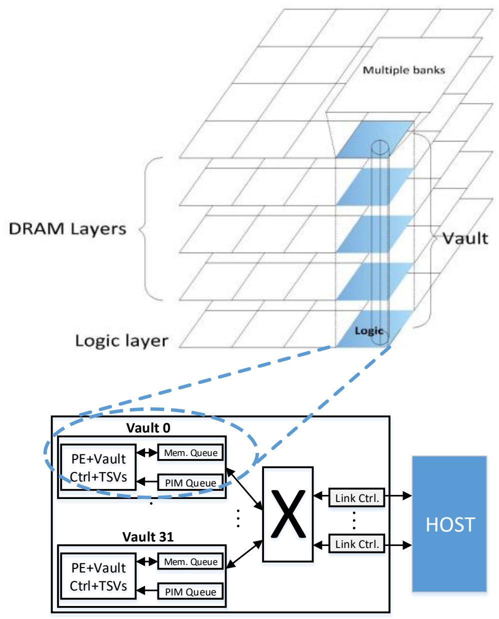
A. Memory-side accelerator design

Figure 2 illustrates the architecture of the proposed PIM design. The vault controller at the logic layer of a baseline HMC-like 3D memory is now equipped with multiple processing elements (PEs).

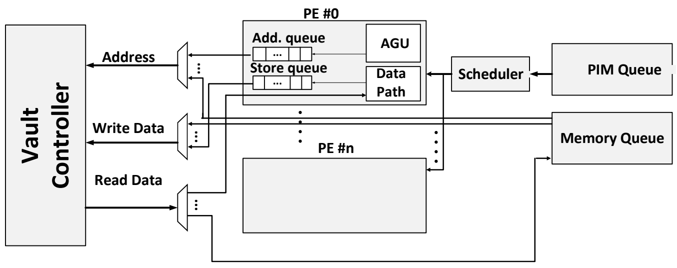
The reference sequences of the target database are read from disk and are distributed across the vaults at the initialization phase. Afterwards, all PEs receive the same query sequence and align it to the reference sequences placed in their vaults. The algorithm is executed concurrently on all vaults, with each vault finds the best alignment in the portion of the database assigned to it. PEs can just communicate with the DRAM banks of their vault through high-speed TSVs.

The key components of each PE are a programmable DRAM address generation unit (AGU) and a customized datapath. In each PE, the datapath is controlled by AGU through a simple handshaking mechanism. AGU fetches the required data from memory and passes them, along with a start signal, to datapath. The datapath then computes the output in a single cycle. The AGU, in the next cycle, writes the datapath output back to the memory.

In order to generate the right sequence of addresses, AGU itself is programmed by the host processor through special PIM packets. PIM packets contain the AGU programming data and are tagged with the target vault number. PIM packets are directed to the right vault by the input crossbar of the memory.



(a)



(b)

Fig. 2. (a) 3D memory and memory-side PEs, (b) inside a vault logic layer

At each vault, the PIM packets are separated from read/write packets by writing them to a separate queue (PIM queue in Figure 2.a). A credit-based flow control mechanism between the memory and host guarantees that the queues never overflow.

B. Memory management and program control at host

A host processor initiates the processing of PEs by configuring (programming) the AGUs with the size and address of the under-test sequences and the corresponding DP matrix. Calling a PE is done by sending a PIM packet to its vault.

The host initiates the execution of a single pairwise alignment (between the query and one of the reference sequences) at a time. For example, if a vault accommodates 10k reference sequences, the AGUs of that vault should be programmed 10k times.

Memory initialization. Before the execution phase, the reference sequences are read from the target database and are mapped into the physical address space of the memory module. The host keeps the start address and length of each reference sequence to later use to program the memory-side AGUs.

When a specialized memory-side accelerator is invoked by a host processor, a hardware mechanism automatically builds and sends a packet to the memory-side PEs.

Like some prior work, the address range of sequences is marked as non-cacheable to prevent cache coherency overhead [12].

The programs on the host work with virtual address. However, when host processor sends a packet to PEs in order to initiate memory-side computation, it simply translates the virtual address of the target sequences to physical address by accessing its TLB, as memory-side PEs work with physical addresses only.

The PEs at each vault align the query sequence to all reference sequences mapped into their vault to find the maximum score of that vault. The host then compares the local maximum value at each vault and finds the sequence with the global maximum alignment score across all vaults.

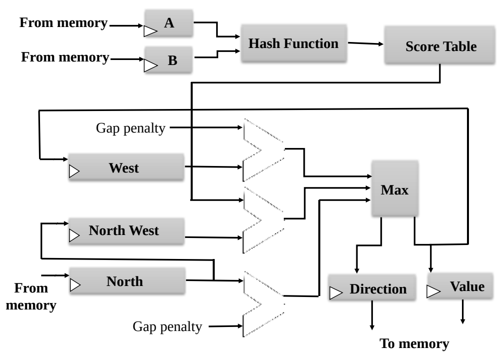


Fig. 3. The datapath of a PE

C. *The datapath structure*

At each invocation, the datapath calculates the value of a single DP matrix cell. According to Equation 1, the datapath that computes the value of DP cells can be implemented by a few comparator and adder units. The architecture of the datapath is depicted Figure 3. In this architecture, once the input data, i.e. two characters from the two sequences under comparison and the required DP cell value, are written to input registers the process starts.

As mentioned before, the forward pass starts form the first row of the DP matrix and proceeds row by row. For each row of DP, it starts from the first column and proceeds column-by-column. As Figure 1.b indicates, in addition to the string characters, we need two cells of the previous row of DP (stored in North and North-West registers in Figure 3) and one cell of the current row (stored in West register in Figure 3) to calculate the value of each DP cell.

The West register is the most recent cell value calculated by the PE and we directly forward the last PE output to the West register (see the connection between the Max. unit and West register in Figure 3). Further, the DP cell that is currently the north cell, will act as the north-west cell in the next iteration of the algorithm. Thus, at each iteration, the North-West register receives the old value of the North register.

However, the content of the North register should be read from memory. To accelerate accessing the cells of the previous row of DP when filling the North register, we can also cache a complete row (once completely calculated) in the PE. Whereas this scheme works well for short sequences, the in-PE storage requirements of sequences with thousands or millions of characters are prohibitive. Nonetheless, we evaluate the effect of cache size on the performance in the next sections.

In Figure 3, the score table keeps the matching score of each pair of characters. For the current implementation, we set the alignment score a match (when two characters are identical) to 1 and a mismatch and matching with gap to -1. For proteins that feature a 20-value alphabet, the matrix is larger (20x20), in which each cell may contain a different score based on the biological problem. For such problems, the two head characters of the sequences are used as the address to the score matrix (combined by hash function in Figure 3 to make the matrix address).

D. *Address generation unit (AGU)*

The address generation unit (AGU) is responsible to generate the sequence of memory addresses required to calculate the value of DP cells and manage data movement between memory and the datapath. Note that we use a separate AGU for each individual PE.

A light-weight scheduler at each vault distributes the PIM packets across the PEs of each vault (Figure 2.b). AGUs have an agu_ready signal to indicate whether they are ready to receive a new task from scheduler. Since four data structures are involved in the process (that are the two sequences (A and B), dynamic programming matrix (DP), and direction matrix (Dir)) the packet contains six parameters: the start address of these four data structures and the length of A and B.

After receiving a packet, AGU is programmed to generate a sequence of addresses according to pseudo code in Figure 4. The pseudo code is designed based on the computation flow depicted in Figure 1.a. The registers in the pseudo code (A, B, West, North, North West, Value, and Direction) appear with the same name in Figure 3. As mentioned earlier, AGU just provides values for A, B, and North registers in each iteration (the other registers are filled internally).

The addresses made by each AGU are formatted as a memory request and are queued in the vault controller to be serviced in a FIFO manner (Figure 2.b). In a vault, the contents of the queues of each PE are sent to the vault controller in a round robin manner. We set the length of this queue to 10 and an AGU is stalled if the queue is full.

Once the two requested data are ready, AGU asserts a data_ready signal to initiate data processing at datapath. Then, as the latency of datapath is one cycle, AGU generates appropriate addresses to write the output of the datapath to the memory.

The matrix-based structures are stored as a linear row-major array in the memory, but we use two-dimensional addressing in the pseudo code for the sake of readability. We also ignore the column and row inserted for gap.

In addition, as each memory access returns a 32-bit word, we should read a new word from the DNA sequences every

16 cycles (as each DNA character can be encoded in two bits). If the sequences are protein, we should read a new word every four to six cycles (as each protein character is encoded in five bits). This involves slightly modifying the pseudo code to reduce the sequence access rate proportional to the target sequence type. This pseudo code can be easily implemented by a state machine.

Once the maximum alignment score is found across all database strings, the Dir matrix is used to specify the alignment path. We omit the details of this step, as it is not executed for all sequences. Once the sequence with the highest score is found across all vaults, the host processor can perform the alignment aging to find the alignment path. So, although the pseudo code of Figure 4 includes data accesses required by this step, it is called only one time during the alignment of a query sequence to a database and its overhead can be completely ignored.

```
Wait Until &A, &B, &DP, &Dir, A.length, and B.length are
received from scheduler // read from buffer

agu_ready = 0;                // busy state

For i = 0 to A.length
  Register A = A[i];          //generate address &A[i] and
                              forward the received data to A
   For j = 0 to B.length
    Register B = B[j];        //generate address &B[j] and
                              forward the received data to B
    Register North = DP[i-1,j];
                              //generate address &DP[i-1,j]and
                              forward the data to North

    data_ready= 1;            //send ready signal to the datapath
    Wait For one clock;
                              //wait for datapath computation to
                              finish
    DP[i,j] = Value Register; //generate address &DP[i,j] to
                              write Value register
    Dir[i,j] = Direction Register; //generate address &Dir[i,j] to
                              write Direction register
   End For //j
End For //i

agu_ready = 1;                // signal the scheduler to send a new
task
```

Fig. 4. AGU description

IV. Experimental Setup and Methodology

In this paper, we evaluate the efficiency of the proposed PIM-based alignment design in terms of throughput and power consumption.

Simulator. We developed a cycle-accurate description for performance evaluation in C++ and validated it with HMCsim [13], which is the most accurate publically-available model of HMC 2.0. In the simulator, the memory specification parameters are set based on the HMC 2.0 specifications [9][12] and are outlined in Table 1. The timing of DRAM and vault controller is taken from HMCsim.

We assume that all memory accesses come from the alignment algorithm and no other programs on the host access the memory during the algorithm execution. We model the host at a high abstraction level: it only copies the sequences from the disk to memories and sends PIM packets to initiate processing at PEs. We believe this level of abstraction is sufficient for current work, as this paper focuses on comparing the memory-side and processor-side accelerator speedups under the sequence alignment application. The behavior and performance gain of the proposed PIM accelerators are agnostic to the type of the host processor and this method can be used with any host that can meet the requirements discussed in Section 3.b.

The major contributor to the speedup of memory-side accelerators is the ratio of the available internal to external bandwidth of the memory module. The internal bandwidth, which is defined as the memory bandwidth available to the PEs at the logic layer, depends on the number of vaults, number of TSVs per vault, and the TSV speed (working frequency). The considered internal bandwidths in recent state-of-the-art PIM designs vary from 160 GB/s to 512 GB/s [14][15]. In this work, we adopt the HMC.20 configuration that provides 10 GB/s bandwidth per vault (320 GB/s bandwidth per memory module).

The external bandwidth, which is defined as the bandwidth that memory provides to the host processor, is also determined by the number of links, number of lanes per link, and the lane speed. This important parameter also comes with different values in recent state-of-the-art PIM designs, with the value varying from 40 GB/s to 320 GB/s [12][15]. We set this parameter to 240 GB/s, which is the maximum available external bandwidth of HMC2.0 (with four 16-bit 60GB/s links).

We further evaluate the impact of different internal and external bandwidth numbers in the next section. The access size through external links is 32 bytes and we have implemented several basic HMC transactions in our simulator according to the protocol and packet format presented in the HMC specifications [9]. The implemented packets include read request, read reply, write request, PIM alignment initiation packet (to program an AGU), and flow control packet (from memory to processor to inform PIM buffer occupancy status).

The results of the simulations show 27% protocol overhead on the external bandwidth (mainly due to the need for 8-byte header and tail flits) that corroborate prior overhead reports (including Micron reports [16]). Even with this overhead, HMC provides significantly higher bandwidth than the fastest conventional DDR memory [9].

Table I. Simulation parameters [12][13]

External bandwidth (up to 4 links)	Up to 240 GB/s
Internal bandwidth (peak per vault)	10 GB/s
DRAM layers	4
Layer size	1 GB
Number of vaults	32
Word Size	32 bits
Row buffer size	256B

Workloads. In this paper, we focus on the global DNA alignment problem, in which a query sequence is compared to a large set of reference sequences. Since DNA sequences can be much longer than protein sequences, our experiments target DNA sequences. We use strings with different sizes as benchmark. We can select realistic benchmarks from standard repositories, but the algorithm and architecture are agnostic to the sequence contents: it is just the string length that affects the performance. Thus, we consider DNA sequences with different lengths, in which every character can take four different values, and randomly fill the sequences.

Each DP matrix cell is 32-bit, as it should keep the cumulative alignment score, which can become quite large as the algorithm approaches to the last cells (upper left corner of the matrix) for long sequences. We store four characters of a DNA sequence in one byte. Since each HMC access returns 32 bits, each access provides either 16 characters of a sequence or one cell of a DP matrix.

PE implementation. The clock frequency, power consumption, and area of the PEs (datapath and AGU) are calculated by synthesizing the VHDL model of the design in 32nm TSMC using Synopsis synthesis and power evaluation tools. According to the synthesis results in 32nm, the critical path of the datapath depicted in Figure 3, is approximately 320ps, resulting in 3.1GHz operating frequency. Working at this frequency, however, may result in thermal issues, particularly when considering the small area footprint of the datapath. According to the synthesis results, the area of a PE is 0.0018mm^2: this small area increases power density of PEs, and hence makes them prone to thermal problems. Note that thermal issues are more sever in 3D architectures, because the logic and DRAM layers share the same path to the heat sink.

We use the Hotspot 6.0 temperature estimation tool with its 3D stacking feature [17] to evaluate the thermal behavior of the proposed design. The technology-dependent parameters of the thermal simulation are obtained from [18].

To estimate the temperature of the memory-side PEs in our design, we calculate the temperature of a PE surrounded by eight neighboring PEs (direct and orthogonal neighbors in a grid-like floorplan) in the logic layer of a 3D memory system. The logic layer is sandwiched between the heat sink and four DRAM layers. This way, the impact of neighboring PEs, the power consumption of the target PE itself, and the DRAM slices in the upper layers has been taken into account, but the impact of elements with less significant contribution, i.e. farther PEs, TSVs, and vault controller is ignored. All PEs (the target and neighboring PEs) work the same frequency. The power consumption of the PE and DRAM slices are calculated by the HDL model and DRAMsim 2.0 [19], respectively, based on the activity factor back-annotated from the C++ simulator.

The HMC 2.0 specification states that the operating temperature of the logic and DRAM dies has to be less than 383˚K and 378˚K, respectively. Thermal simulation shows that the peak steady-state temperature of PEs violates the temperature threshold when working at 3.4GHz, but will go lower than the threshold across all PEs when the frequency is reduced to 0.673 GHz. So, we set the clock frequency of the memory-side PEs to 0.67 GHz in our experiments.

Number of PEs. Each iteration of the datapath execution, as discussed in Section 3.5, requires four data accesses (two data reads and two data writes). One read operation fills the 32-bit North register by some DP cell. The other read operation targets the 2-bit reference sequence characters, where one access can provide data for 16 cycles. Write operations also transmit the 2-bit Dir matrix cells (done every 16 cycles) and the calculated 32-bit DP cell value to memory. So, ignoring the infrequent access to the query sequence characters (which are read one time for each row), each iteration of the algorithm involves approximately 2.125 memory accesses (8.5 bytes), on average. Consequently, approximately two memory-side PEs can work concurrently at each vault to fully exploit the 10 GB/s internal vault bandwidth (calculated as: (10 GB/s)/(0.67 GHz × 8.5 B/clock)). This gives a total of 64 PEs per HMC (with 32 vaults). Note that area constraints do not limit the PE count, as more than half of the 68mm^2 area of an HMC module is free to implement accelerators [12][9]: it is by far larger than the total area footprint of our accelerators (that is 64×0.0018mm^2).

Processor-side accelerator configuration. We compare the results with the case where the PEs are implemented at the processor side and receive data from an off-chip memory (with the bandwidth listed as "external bandwidth" in Table 1). We refer to this configuration as processor-side. Note that this configuration also adopts the 3D HMC-like memory (and not a conventional 2D DDR), but the PEs are implemented at the processor side.

The same numbers of PEs as the memory-side configuration are considered for the conventional design (processor-side PEs). In the simulations, the processor-side PEs work at 2.2 GHz, as our temperature estimation shows that the aforementioned 383˚K thermal constraint cannot be met beyond this frequency.

V. Experimental Results

A. Throughput evaluation

Figure 5 compares the throughput of the conventional processor-side and the proposed memory-side configurations. The results are normalized to the memory-side configuration to give a better insight into the obtained improvements.

The system aligns a 100k-character query sequence against a database containing 320k reference sequences. We repeat the evaluation with multiple reference sequence lengths and processor-side cache sizes to evaluate the impact of them on speedup. Each vault accommodates 10k reference sequences. The query sequence is replicated in all vaults.

As Figure 5 illustrates, the proposed PIM-based method outperforms the conventional processor-side accelerator design by 2.2x, on average.

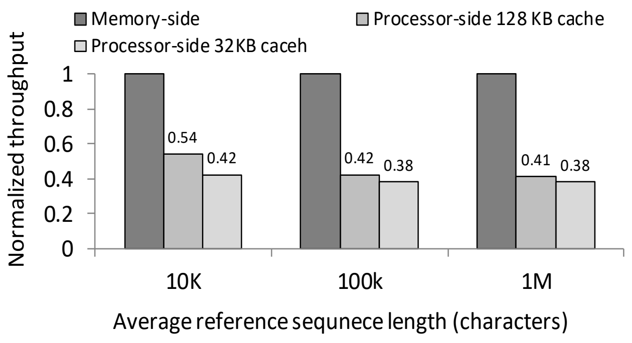


Fig. 5. Throughput comparison for different reference sequence lengths and cache sizes

The main reason is the higher bandwidth available for in-memory PEs, but processor-side PEs (although working at higher frequency) are bound by the more limited off-chip bandwidth. As a result of this underutilization, processor-side PEs offer lower throughput than their memory-side counterparts.

As the figure shows, the throughput improvement is almost agnostic to sequence length and is approximately proportional to the ratio of the internal to the external bandwidth.

To evaluate the impact of cache, we repeat the previous experiment with two private data cache sizes (32KB and 128KB) for each PE of the processor-side configuration (for a total of 2MB and 8MB caches for the entire system, respectively). The cache features single cycle access latency and is configured as four-way set-associative.

Bioinformatics algorithms exhibit poor temporal locality, because once a reference sequence is fetched, it will never be used again for the current query. DP matrix cells also suffer from a large reuse distance that is proportional to the sequence length.

As Figure 5 shows, the cache can improve the performance for short sequences, because they exhibit better caching behavior for the DP matrix. The figure shows that long DNA sequences cannot benefit much from reasonably-sized data caches. This poor caching behavior do not justify the area and power overheads of further using more sophisticated multi-level cache structures.

B. *Power evaluation*

Figure 6 shows the power consumption of a PE in the two configurations. The power consumption of the PEs are extracted from the hardware implementation of the designs (as described earlier in this section) and the memory access power consumption for memory-side and processor-side PEs are set to 3.7 pJ/bit and 10 pJ/bit, respectively, based on the energy results reported in [12].

The figure shows that the proposed PIM-based implementation can also outperform the processor-side design in terms of power consumption. The main source of power reduction in the proposed architecture is the elimination of on-board data transfer that has a considerable contribution to the total power consumption of computers [20].

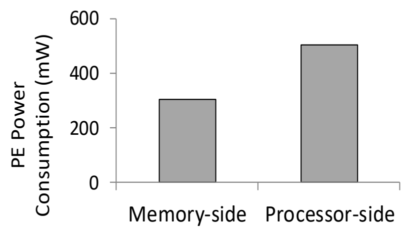


Fig. 6. The power usage comparison of PEs

C. *Sensitivity to internal and external bandwidths*

To study the effect of the ration of the internal to external bandwidths on the benefits of our design, Figure 7 shows the speedup of the memory-side accelerators when the internal bandwidth is 1x, 2x, 3x, and 4x higher than the external bandwidth. In the simulations, we fix the internal bandwidth to 320 GB/s and scale down the external bandwidth proportionally. The processor-side accelerators have 128K data cache per PE. As we expect, the speedup increases proportional to the bandwidth ratio and as Figure 7 demonstrates, reaches to 3.8x when the internal bandwidth is 4 times higher.

Again, the cache can partially bridge the gap between the memory-side and processor-side designs for shorter sequences, but fall short in reducing the stress on the off-chip bandwidth for long sequences.

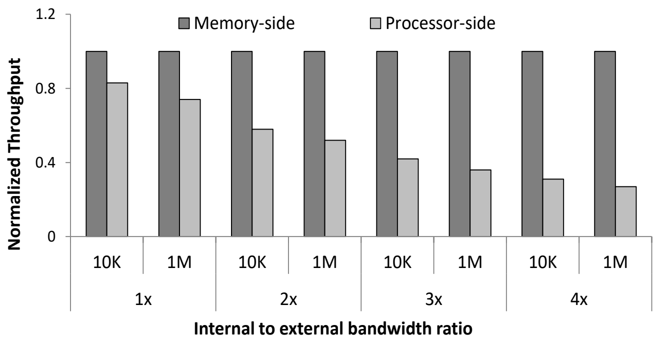


Fig. 7. Throughput comparison for different internal to external bandwidth ratios with 128 KB processor-side cache

VI. Conclusion

In this paper, we proposed a processing-in-memory approach to accelerate bioinformatics algorithms. As biological sequence alignment is the base of the majority of bioinformatics algorithms, we target accelerating this operation and focused on the well-known Needleman–Wunsch dynamic programming approach. The main motivation behind this proposal is the low operation density (operation per byte) of sequence alignment algorithms that make them bandwidth-limited. The proposed architecture consists of a set of processing elements implemented at the logic layer of a 3D DRAM chip. Each processing element is composed of an address generation unit and a datapath customized for the sequence alignment problem. Experimental results show that moving the processing elements to the memory side leads to more than 2.4x improvement in throughput and 41% reduction in power consumption for the global alignment problem.

References

[1] S. Aluru and N. Jammula, “A Review of Hardware Acceleration for Computational Genomics,” in *IEEE Design & Test*, vol. 31, no. 1, 2014, pp. 19-30.

[2] J. Zhang, et al., "cuBLASTP: Fine-Grained Parallelization of Protein Sequence Search on CPU+GPU," in *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, vol. 14, no. 4, pp. 830-843, 2017.

[3] J. Cohen, “Bioinformatics-an introduction for computer scientists,” in *ACM Computing Surveys, vol.* 36, no. 2, 2004, pp. 122-158.

[4] E. Sandes, et al., “Parallel Optimal Pairwise Biological Sequence Comparison: Algorithms, Platforms, and Classification,” in *ACM Computing Surveys*, vol. 48, no. 4, 2016.

[5] https://www.ncbi.nlm.nih.gov/genbank/statistics, Apr. 2018.

[6] S. Sarkar, et al., “Hardware accelerators for bio-computing: A survey”, in Proc. of ISCAS, 2010.

[7] D. Jevdjic, S. Volos, and B. Falsafi, "Die-stacked DRAM caches for servers: hit ratio, latency, or bandwidth? have it all with footprint cache," in *Proc. of ISCA*, 2013, pp. 404-415.

[8] D. U. Lee et al., "25.2 A 1.2V 8Gb 8-channel 128GB/s High-Bandwidth Memory (HBM) Stacked DRAM with Effective Microbump I/O Test Methods Using 29nm Process and TSV," in *Proc. of ISSCC*, 2014, pp. 432-433.

[9] "Hybrid memory cube specification 2.0," Hybrid Memory Cube Consortium, Tech. Rep., Nov. 2014.

[10] M. Modarressi, et al., "Hardware accelerator for biological protein sequence alignment on reconfigurable Networks-on-Chip," in *Proc. of East-West Design & Test Symposium*, 2015, pp. 1-4.

[11] K. Benkrid, A. Akoglu, C. Ling, Y. Song, Y. Liu, and X. Tian, "High performance biological pairwise sequence alignment: FPGA versus GPU versus cell BE versus GPP," in *International Journal of Reconfigurable Computing*, vol. 2012, no. 7, pp. 1-11, Feb.2012

[12] D. Kim, et al., "Neurocube: A Programmable Digital Neuromorphic Architecture with High-Density 3D Memory," in *Proc. of ISCA*, 2016, pp. 380-392.

[13] J. D. Leidel and Y. Chen, "HMC-Sim-2.0: A Simulation Platform for Exploring Custom Memory Cube Operations," in Proc. of International Parallel and Distributed Processing Symposium Workshops, 2016, pp. 621-630.

[14] K. Hsieh, et al., "Transparent offloading and mapping (TOM): enabling programmer-transparent near-data processing in GPU systems," in Proc. of ISCA, 2016, pp. 204-216.

[15] J. Ahn, S. Hong, S. Yoo, O. Mutlu and K. Choi, "A scalable processing-in-memory accelerator for parallel graph processing," in *Proc. os ISCA*, 2015, pp. 105-117.

[16] Hybrid Memory Cube Webinar, available: *https://www.micron.com/*, Apr 2018.

[17] Hotspot, available: http://lava.cs.virginia.edu/HotSpot.

[18] M. Keramati, et al., "Thermal management in 3d networks-on-chip using dynamic link sharing," Elsevier Journal of Microprocessors and Microsystems, vol. 52, 2017, pp. 69-79.

[19] DRAMSim 2.0, available at: https://eng.umd.edu/~blj/dramsim/, Apr. 2018.

[20] A. Boroumand, et al., "Google Workloads for Consumer Devices: Mitigating Data Movement Bottlenecks", in *Proc. of ASPLOS*, 2018.

.