MEDAL: Scalable DIMM based Near Data Processing Accelerator for DNA Seeding Algorithm

Wenqin Huangfu[1], Xueqi Li[1,3], Shuangchen Li[1,2], Xing Hu[1], Peng Gu[1], Yuan Xie[1,2]
[1]University of California, Santa Barbara [2]DAMO Academy, Alibaba Group Inc.
[3]Institute of Computing Technology, Chinese Academy of Sciences
{wenqin_huangfu,shuangchenli,yuanxie}@ece.ucsb.edu

ABSTRACT

Computational genomics has proven its great potential to support precise and customized health care. However, with the wide adoption of the Next Generation Sequencing (NGS) technology, 'DNA Alignment', as the crucial step in computational genomics, is becoming more and more challenging due to the booming bio-data. Consequently, various hardware approaches have been explored to accelerate DNA seeding - the core and most time consuming step in DNA alignment.

Most previous hardware approaches leverage multi-core, GPU, and FPGA to accelerate DNA seeding. However, DNA seeding is bounded by memory and above hardware approaches focus on computation. For this reason, Near Data Processing (NDP) is a better solution for DNA seeding. Unfortunately, existing NDP accelerators for DNA seeding face two grand challenges, i.e., fine-grained random memory access and scalability demand for booming bio-data. To address those challenges, we propose a practical, energy efficient, Dual-Inline Memory Module (DIMM) based, NDP Accelerator for DNA Seeding Algorithm (MEDAL), which is based on off-the-shelf DRAM components. For small databases that can be fitted within a single DRAM rank, we propose the intra-rank design, together with an algorithm-specific address mapping, bandwidth-aware data mapping, and Individual Chip Select (ICS) to address the challenge of fine-grained random memory access, improving parallelism and bandwidth utilization. Furthermore, to tackle the challenge of scalability for large databases, we propose three inter-rank designs (polling-based communication, interrupt-based communication, and Non-Volatile DIMM (NVDIMM)-based solution). In addition, we propose an algorithm-specific data compression technique to reduce memory footprint, introduce more space for the data mapping, and reduce the communication overhead. Experimental results show that for three proposed designs, on average, MEDAL can achieve 30.50x/8.37x/3.43x speedup and 289.91x/6.47x/2.89x energy reduction when compared with a 16-thread CPU baseline and two state-of-the-art NDP accelerators, respectively.

ACM Reference Format:
Wenqin Huangfu[1], Xueqi Li[1,3], Shuangchen Li[1,2], Xing Hu[1], Peng Gu[1], Yuan Xie[1,2] . 2019. MEDAL: Scalable DIMM based Near Data Processing Accelerator for DNA Seeding Algorithm. In *Proceedings of The 52nd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO'19).* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3352460.3358329

1 INTRODUCTION

DNA seeding, as the bottleneck stage in computational genomics, has drawn tremendous attention [3, 9]. Computational genomics is developing rapidly and is well motivated by its potential adoption to the precise and customized medical care, such as helping physicians to select a particular drug or treatment suitable for a specific pathology of a cancer [10]. With the higher throughput and cost efficiency of the Next Generation Sequencing (NGS) technology [26], bio-data is booming, leading to severe stress on genomics analysis due to more data but a shorter time budget. DNA alignment, which aligns short reads, i.e., DNA subsequences, to reference genomes to locate positions of the short reads, is commonly considered as the core and one of the most time-consuming steps in genomics analysis [9]. DNA alignment contains two major time-consuming steps, i.e., DNA Seeding and Seed Extension. DNA Seeding generates exact matches, i.e., seeds, between the short reads and the reference genomes, and Seed Extension extends the seeds to longer matches with gaps allowed. DNA seeding and seed extension equally dominate the end-to-end computing of DNA alignment, thus they are both worth being accelerated [3, 9, 20].

The importance of DNA seeding motivates plenty of researches to accelerate it. Various compute-centric hardware approaches, such as multi-core [56], GPU [38, 40], and FPGA [9, 14] have been explored to accelerate DNA seeding. However, innovations that only focus on the computation have limited space for improvement, since DNA seeding is memory bound [3, 9]. DNA seeding introduces huge amount of random data movement [56], resulting in bottlenecks of both performance and energy [21, 28]. Near Data Processing (NDP) architecture emerges as a better acceleration solution for DNA seeding by addressing the issue of data movement [37]. Such architecture integrates computation and memory closely to embrace the larger internal memory bandwidth and reduce the overhead of data movement. For example, MPU-BWM [55] accelerates DNA seeding with a RISC-V core in the logic die of a HMC. Chameleon [6] and AIM [10] provide more practical solutions by leveraging the Dual-Inline Memory Module (DIMM) to build accelerators, with off-the-shelf commodity DRAM components.

However, existing NDP accelerators for DNA seeding face two grand challenges. The first challenge is the **fine-grained random memory access**. The randomness stresses memory bandwidth due to the bank/channel conflict and the fine-granularity results in low bandwidth utilization. For example, in BWA-MEM [34] - the most widely used software tool for DNA alignment - only 32

Bytes data is useful on average for each 64 Bytes cacheline from one memory access [35]. In addition, the inter-task divergence in memory access makes the application difficult to run efficiently on a SIMD-based hardware. Previous work Chameleon [6] is a general purpose SIMD NDP accelerator. As a result, it fails to tackle this challenge of fine-grained random memory access and shows sub-optimal performance on DNA seeding even if a communication mechanism is added. The second challenge is the demand for **scalability** due to the exploding DNA data. Biological data has been growing exponentially [26]. For example, data in Whole Genome Shotgun(WGS) project [17] of the National Center for Biotechnology Information (NCBI) have doubled the size approximately every 18 months and now the WGS project contains more than 3.44 trillion bases. The rapidly, ever-increasing data demand scalability. Previous work AIM [10] proposes to accelerate DNA seeding with DRAM by linking customized FPGA accelerators and dedicated data buses to DIMMs. However, because there is no rank-level parallelism within AIM and AIM accesses memory in coarse-granularity (64 Bytes), potential memory bandwidth is not fully utilized. For Chameleon [6], its bandwidth utilization ratio is very low (about 10% in our experiments), although leveraging rank-level parallelism is one of its design purposes, simply due to the inter-task divergence in DNA seeding and Chameleon's SIMD-style processing.

The **goal** of this paper is to build a NDP accelerator for DNA seeding with fine-grained memory accessibility, high bandwidth utilization, and scalability. We propose an accelerator, i.e., MEDAL, on DIMM between DRAM components and the standard data bus. MEDAL highlights practicability by using off-the-shelf DRAM components and the standard DDR protocol. MEDAL leverages both the rank-level and the fine-grained, chip-level memory bandwidth.

Within a rank, we propose three techniques to address the challenge of fine-grained random memory access, improving parallelism as well as bandwidth utilization. The proposed methods take advantages of our in-depth characterization of the target DNA seeding algorithm. The first technique is the algorithm-specific address mapping, which maps the continuous data together in a single DRAM chip to improve locality, provides potential for chip-level parallelism and fine-grained memory access, and reduces communication, instead of interleaving data across multiple chips. The second technique is the bandwidth-aware data mapping. It duplicates or remaps data across all available chips to fully utilizes potential memory bandwidth. The third technique is the Individual Chip Selection (ICS), which leverages the Chip Selection (CS) signal to support chip-level parallelism and fine-grained memory access, further improving the bandwidth efficiency.

Across ranks, when the index data cannot be fit into one rank, we then propose three design options to address the multi-rank scaling out issue caused by the exploding data challenge. The proposed methods highlight the practicability for scaling out. The first, also the basic, design leverages CPU polling for inter-rank communication. Compared with the first design option, our second design, i.e., interrupt-based design, doesn't need to occupy the host and memory bus for polling operations. The Reserved for Future Use (RFU) pin in DDR is used for triggering interupts. The third design alternatively introduces the NVDIMM-P, in which we store the large DNA index within the dense Non-Volatile Memory (NVM) to reduce/eliminate inter-rank communication. In addition, we propose an algorithm-specific data compression technique to reduce memory footprint, introduce more space for data mapping to utilize, and reduce communication overhead. Our specific contributions are listed as follows.

- We propose a practical and energy-efficient NDP accelerator architecture, i.e., MEDAL, for DNA seeding with off-the-shelf DRAM components.
- For the intra-rank design, we propose three application-specific techniques (algorithm-specific address mapping, bandwidth-aware data mapping, and ICS) to address the challenge of fine-grained random memory access and improve parallelism as well as bandwidth utilization.
- For the inter-rank design, we propose three alternative approaches (polling-based communication, interrupt-based communication, and NVDIMM-based solution) to overcome the challenge of big data and system scaling.
- In addition, we propose an algorithm-specific data compression technique to reduce memory footprint, introduce more space for data mapping to utilize, and reduce communication overhead, leading to performance improvement.
- Our experimental evaluation shows that MEDAL can provide 30.50x/8.37x/3.43x better performance and 289.91x/6.47x/2.89x better energy-efficiency than a 16-thread CPU baseline and two state-of-the-art NDP accelerators, respectively.

2 BACKGROUND

This section introduces the basics of DNA seeding algorithm and the buffered DIMM.

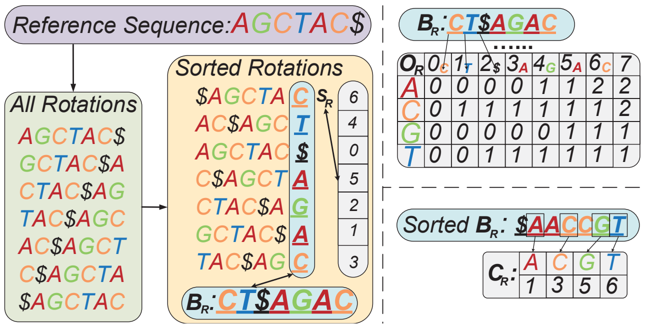


Figure 1: An example of data structures in FM-index based DNA seeding.

DNA Seeding Algorithm: DNA seeding refers to the process of matching seeds (small sequence fragments chopped from a given read) back to the long reference genome. DNA seeding algorithms usually pre-build an index of the reference genome to speedup the locating. FM-index [32, 34] and Hash-index [2] are two mainstream seeding indexes used by modern DNA aligners. Both of the algorithms are preferable, depending on the combination of the seeding and the extension approach [2]. For example, FM-index based algorithm has good performance for local alignment and BLAST-like seed extension [2]. Different from the Hash-index based algorithm, FM-index based algorithm suffers from more irregular memory access and longer data reuse distance, and hence is more challenging. Therefore, our main target is to accelerate FM-index based algorithm. For generality, we keep the design compatible for the Hash-index based algorithm, and evaluate both of them.

The flow of the FM-index based seeding algorithm contains the offline preprocessing (index building) and the online searching. During the preprocessing, the following data for the reference genome R are calculated and prepared.

- $B_R[len]$: the Burrows-Wheeler Transform [9] of R (B_R);
- $S_R[len]$: the Suffix Array (S_A), i.e., record the original id of sorted rotations before sorting;
- $C_R[4]$: the accumulative count array, i.e., the index of the first appearance of A, T, C, G in sorted B_R;
- $O_R[len+1][4]$: the occurrence array, i.e., the occurrences of each nucleotide (A, T, C, G) before the i^{th} symbol of B_R;

An example is shown in Fig. 1, the reference sequence is $AGCTAC$. The algorithm first terminates the reference sequence with a unique character \$. Then, all rotations of above character string are generated. Next, those rotations are sorted in alphabet order. The last characters of all entries in above sorted rotations form $B_R[len]$. $S_R[len]$ is also derived during this process. Finally, with $B_R[len]$ and sorted $B_R[len]$, $O_R[len+1][4]$ and $C_R[4]$ can be generated.

Algorithm 1: FM-index based Seeding Algorithm

```
Input: Query sequence $SD[d]$, Reference genome $R[len]$
Output: Matching locations
Preprocess: Derive $B_R[len]$, $S_R[len]$, $C_R[4]$, and $O_R[len+1][4]$;
while $I^{lower} <= I^{upper}$ do
    $x \leftarrow SD.getchar()$;
    if $x = EoF$ then break;
    $I^{lower} = C_T[x] + O_T[x][I^{lower} - 1]$;
    $I^{upper} = C_T[x] + O_T[x][I^{upper}]$;
end
for $I^{lower} <= i <= I^{upper}$ do
    $Match\ location[i] = S_R[i]$;
end
return $Match\ location$;
```

During the searching, the algorithm extends current match by one nucleotide each iteration, reading C_R and O_R to locate the range of matches until no match can be found. Algorithm 1 shows the searching flow. I^{lower} and I^{upper} represent the first and the last index of the suffix sequence with current prefix of SD in S_R. This range contains all occurrences of current prefix of SD in R. $I^l(D) \leq I^u(D)$ if and only there is at least one match in R.

Buffered Dual-Inline Memory Module: Dual-Inline Memory Module (DIMM) is a widely used memory package with 64 data (DQ) pins. In a DIMM, multiple DRAM chips form a rank, and one or more ranks are packaged together to form a DIMM. Load-Reduced DIMM (LRDIMM), as shown in Fig. 5(b), is introduced to address the signal integrity issue for high frequency memory interface. The key component in LRDIMM is the Memory Buffer (MB) that enhances the C/A and DQ signals. The MB is divided into two pieces:

- Registering Clock Driver (RCD): One per DIMM to buffer and repeat C/A signals.
- Data Buffer (DB): One for a set of (e.g., 2/4/8) DRAM chips to improve the signal integrity of DQ signals.

3 MOTIVATIONS AND CHALLENGES

We first justify the selection of our target application, i.e., DNA seeding, for two reasons. First, DNA seeding dominates the computation-expensive DNA alignment in BWA-MEM [34] - the most widely used tool for DNA alignment, which contains both DNA seeding and seed extension. Previous work [9] has shown that DNA seeding is the most time-consuming step in DNA alignment and takes 29.35% of the runtime, whereas the seed extension takes the second most time at 27.89%. Furthermore, our evaluation shows that in the scenario of metagenomics (aligning sequences from unknown species to huge reference databases), DNA seeding takes up to 48% of the runtime, because of the less presence of the extension part due to the lower hit rate during the seeding. Second, seed extension is computing-bound [3] and has been extensively studied [19, 36, 39]. ASIC/GPU/FPGA [19, 36, 39] has shown up to 652x speedup, compared to the CPU baseline. With those advanced accelerators for seed extension already developed, the memory-bound DNA seeding becomes the outstanding bottleneck.

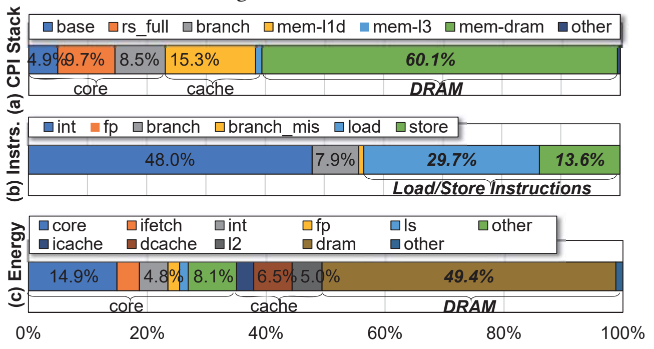


Figure 2: Profiling the DNA seeding in BWA-MEM [34] with Sniper [7] and configuration in Table 1: (a) CPI stack; (b) Instruction statistics; (c) Energy breakdown.

Memory system is the bottleneck [3, 9, 20] of DNA seeding, motivating our key idea of adopting the NDP architecture. Profiling results in Fig. 2 quantitatively show the bottleneck. As Fig. 2 (a) shows, the DRAM access accounts for 60% in the CPI stack analysis; The *Load/Store* instructions take 43.4% among total instructions in Fig. 2 (b); The energy breakdown in Fig. 2 (c) shows that 49.4% of the total energy consumption is consumed by the DRAM.

While designing NDP architecture for the DNA seeding application is well motivated, it faces two grand challenges.

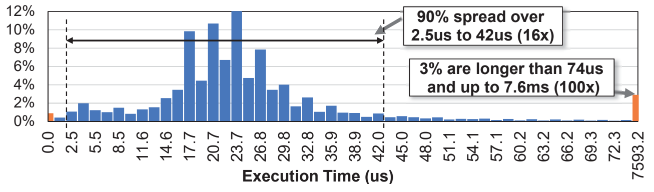


Figure 3: Execution time distribution of the seeding's elemental task (`bwt_smem1a` [34], the atomic function for parallel DNA seeding).

Challenge-1 Fine-Grained Randomness and Divergence: This challenge leads to the memory bandwidth under-utilization and workload imbalance with the SIMD architecture. The memory access pattern of FM-index based DNA seeding is random and fine-grained. We profile the Last-Level Cache (LLC) miss rate for the seeding's elemental task, showing 32.5% on average and up to 93.24% peak miss rate. Fine-granularity of memory access is due to the reason that the actual data needed in each iteration are only 4 integers of O_R, while 512 bits are fetched from the memory. The **challenge** of this fine-grained random memory access pattern results in two problems: First, the inevitable demands for larger memory bandwidth; Second, the low bandwidth utilization. For each 64B memory access to get $O_R(x, i)$, on average only 50% of the cacheline is actually used [34, 35].

Furthermore, the behaviors of individual seeding tasks are divergent. The profiling results in Fig. 3 show that the majority of the elemental seeding task (90%) spreads from 2.5μs to 42μs (16× difference) with a long tail effect that 3% of the tasks run longer than 74μs (up to 7.6ms).

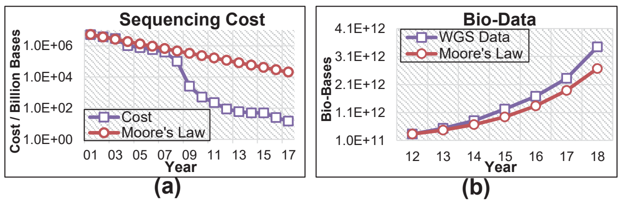


Figure 4: (a) Reduction of sequencing cost [22]. (b) Boom of the bio-data [17].

Challenge-2 Big Data: The bio-data has shown even faster growth than the Moore's law. Fig. 4(a) shows that the cost of sequencing genomes reduces much faster than Moore's law, indicating that larger databases for more species are becoming practical and affordable [22]. Fig. 4(b) shows that bio-data grows exponentially. The speed of the data growth is also faster than the Moore's law. Currently, it takes ~6.75 GPU hours [45] for seeding on human genome[1]. For larger genome library (e.g., Ambystoma Mexicanum [42] is 10x bigger than the human's), the seeding time scales super linear. The challenge of the big data demands scaling out, since the capacity of DRAM in one node is limited. The communication overhead, e.g., Non-Uniform Memory Access (NUMA), Remote Direct Memory Access (RDMA), further stresses the memory system. In addition, our profiling results show that the searching performance degrades 3.4×, if the average memory latency doubles.

4 MEDAL ARCHITECTURE

In this section, we introduce the MEDAL architecture. After an overview of the architecture, we describe the working flow and techniques for intra and inter rank scenarios.

4.1 Architecture Overview

The goal of MEDAL is to leverage NDP architecture for exploiting extra bandwidth for DNA seeding, while remaining practical by using off-the-shelf host processors and DRAM components. To this end, MEDAL exploits extra bandwidth and parallelism inside per channel for DIMM-based memory systems, while it only conducts modifications to the DIMM printed circuit board (PCB) design. Such design simultaneously activates the DRAM in different ranks. Assuming a typical memory system in Table 1, compared with the conventional case where only ranks in different channels can be accessed in parallel, MEDAL exploits 12× more bandwidth. Compared with NDP previous work [10] that only exploits DIMM-level parallelism instead of rank-level parallelism, MEDAL exploits 4× more bandwidth. Furthermore, MEDAL even leverages the chip-level parallelism within the same rank via decoupling their CS signals.

Specifically, MEDAL is built by modifying the commercial LRDIMM, as shown in Fig. 5(a) and (b). Five components below are added.

DB-Side Accelerator: We attach 4 DNA seeding specific hardware accelerators to each DB in the LRDIMM, as shown in Fig. 5 (c) and (d). The DB-side accelerator inputs/outputs data with the FIFO connected to the inter-chip hierarchical data bus, and sends a pair of current accelerator ID and its read/write request to the FIFO, which connects to the inter-chip hierarchical ID/address bus. To perform the task described in Algorithm 1, the accelerator contains

- registers to store the query sequence q,
- a 4×64-bit register file to store $C_R[4]$,
- a data reorganization engine to calculate $O_R[x]$ from its stored data structure,
- two 64-bit unsigned adders to update I^{uppper} and I^{lower},
- an address translation engine to convert the virtual address to DRAM device address (see Section 4.2 for detail).

DB-Side Multiplexer: We add a multiplexer to the output of DB, so that in addition to sending data to the DDR bus, DB can also send data to the inter-chip hierarchical data bus through the DB-side FIFO. The multiplexer is controlled by a dedicated enable signal from RCD-side MC.

RCD-Side Memory Controller (MC): The DB-side accelerator creates a new challenge. Since both the host and the accelerators can access the DRAM and the host is not aware of the requests issued by the accelerators. Requests from the two sides will conflict if they are not well coordinated.

Our design philosophy is to enable the RCD to coordinate those memory requests, since the RCD has the information of memory requests from both the host and the accelerators. We modify the RCD in the original LRDIMM, design the RCD-side MC, as shown in Fig. 5 (c), and propose a host-prioritized request scheduling (details in Section 4.3), to address this issue. The original RCD only serves as an enhancement module for the C/A signals. In MEDAL, the following modifications and new components are added.

- The C/A signal from the host is detoured to the RCD-side MC before going to the DRAM components.
- A MC, which merges and schedules the requests from both the host and the DB-side accelerators, is added with a request queue and a scheduling engine. The scheduler will prioritize the host-side requests. This is because host-side MC is not aware of the accelerator-side requests, so the host memory accesses must be served soon to meet the expectation of the host (details in Section 4.3). For other accelerator requests, it applies the first-ready first-come-first-serve scheme. The scheduler also makes sure DRAM timing constraints are met, and helps the following controllers to generate C/S and enable signal on right time.
- A controller to generate dedicated CS signals to each DB according to timing information from the MC scheduler, instead of one global CS bus. Details of the chip selection optimization is introduced in Section 4.2.
- A controller to generate enable signal for the FIFO in the DB-side multiplexer. Since data accessed by the accelerator is transferred

[1]50× coverage, 76bp reads, with the reference library of human genome, on a Tesla K80 GPU.

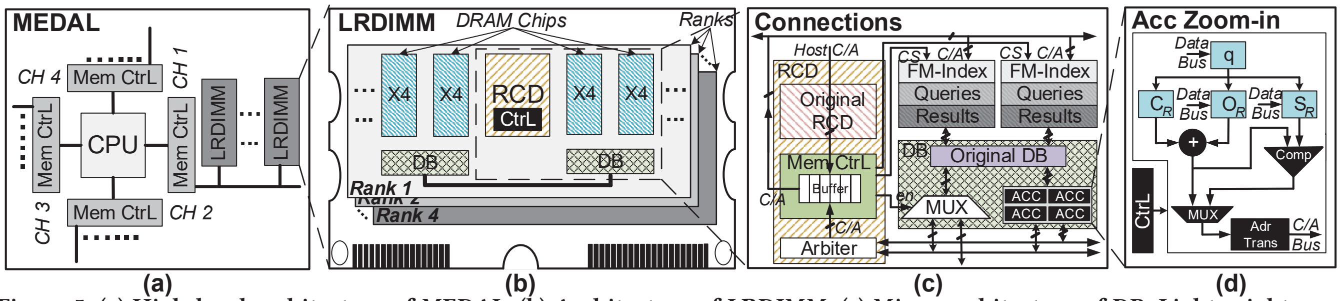


Figure 5: (a) High-level architecture of MEDAL. (b) Architecture of LRDIMM. (c) Micro-architecture of DB. Lightweight, customized logics are inserted within it. (d) Connections between RCD, DB, and DRAM chips.

to the data bus, the enable signal makes sure the FIFO catch and buffer the data when the bus is not ready for data transfer.

Inter-Chip Hierarchical Bus: Another design challenge is that, with proposed algorithm-specific data mapping discussed in Section 4.2, the $O_R[x][i]$ is spread over all DRAM components across the rank. The distributed data mean the DB-side accelerator may need the data from other DRAM components in this rank. However, there is no connection between different DRAM components within a rank in the vanilla DIMM, forbidding inter-chip communication.

To address this challenge, we design the inter-chip hierarchical bus. Specifically, we have the *ID/address bus* and the *data bus*. The *ID/address bus* transfers the pair of the accelerator ID and its memory requests from the accelerator to the RCD-side MC. Accelerators are masters writing the data (i.e., ID/addresses) to the only slave (i.e., the RCD-side MC). The *data bus* transfers the data from a DB belonging to a group of DRAM components to its destination accelerator. The FIFOs in the DB-side multiplexer are the masters writing the data (i.e., DRAM data) to the slaves (accelerators). Both of the buses are multi-master single-channel buses. The bus is simplified from standard bus like AMBA [5], and it contains (1) a shared clock signal and reset signal for both buses, (2) 1-bit dedicated master/slave selection signal for each master/slave, (3) 8-bit write data signals for the *ID/address bus*, 64-bit bi-direction data signals for the *data bus*, (4) burst length 5 for the *ID/address bus* to transfer 8-bit accelerator ID and 32-bit address, burst length 8 for the *data bus*. Bus address signals are eliminated since the RCD-side arbitrator, which is introduced in detail below, has already been aware of the source and destination module of every transfer though the RCD-side MC, and can simply assign the bus using the master/slave selection signals.

Note that we make the such a design choice under the consideration of minimizing the wiring complexity on PCB. The simplified single-channel buses conduct $(4 \cdot n + 75)$ extra wires, where n represents the number of DBs per DIMM.

RCD-Side Bus Arbitrator: The arbitrator assigns the bus to the masters sharing the bus. For the C/A bus, the arbitrator applies first-come-first-server scheme to grant the bus for each accelerator, so that all of them can send their memory request to the RCD-side MC. For the data bus, the arbitrator sets a pair of the master/slave selection signal so that DRAM data can be transferred from a DB-side multiplexer to an accelerator. The arbitrator works with the aid of the RCD-side MC. Since accelerators send their ID and read request to the MC, the MC can provide (1) when the data will be ready from which DRAM and (2) which accelerator requests this data. With above information, the arbitrator then picks a pair of master and slave for data transfer after the data is ready.

The rest of this section describes the detailed data flow on MEDAL. First, we will show the simpler case when the memory footprint is small enough to be fit in one rank, focusing on intra-rank optimizations to address the **Fine-Grained Randomness and Divergence** challenge. Then, we describe the general case when the memory footprint is large and inter-rank communication is required, focusing on techniques to address the **Big Data** challenge.

4.2 Intra-Rank Workflow and Optimizations

In this subsection, after a detailed description of the architecture and control flow, we address the **Fine-Grained Randomness and Divergence** challenge by proposing algorithm-specific data mapping, bandwidth-aware data mapping, and the Individual Chip Selection (ICS). Note that we start with the simple case when the data can be fitted in a single rank to provide a better focus on addressing the challenge. However, these techniques are applicable for larger databases as well.

Working Flow: We go through the working flow of MEDAL. Before execution, the genome data are stored in DRAM with the address mapping and data mapping to be introduced later. To get started, the host sends a DDR command of writing a reserved DRAM mode register. The RCD-side MC catches this command and broadcasts it to every DB-side accelerator by resetting the reset signal in the inter-chip *data bus*. To reads O_R and S_R from the DRAM, each DB-side accelerator first sends the read request to the RCD-side MC through the inter-chip *ID/address bus*. After scheduling, the MC sends the C/A signals to the DRAM component through the original C/A bus, and informs the DB-side accelerator to get ready via the selection signal in the inter-chip *data bus*. Finally the DB-side accelerator receives the data through the inter-chip *data bus*. The accelerators keep iterating till the end of search.

We propose an algorithm-specific address mapping to improve data locality and provide potential for chip-level parallelism as well as fine-grained memory access, a bandwidth-aware data mapping to fully leverage memory bandwidth, and ICS to leverage chip-level parallelism as well as support fine-grained memory access.

Algorithm-Specific Address Mapping: The key idea of proposed address mapping is to aggregate previous interleaved data during address mapping to reduce communication and provide potential of chip-level parallelism as well as fine-grained memory access, improving bandwidth utilization. Thus, we propose a logic-device address mapping scheme to address the challenge of fine-grained random memory access. The scheme includes two optimizations.

The original address mapping is shown in Fig. 6 (a) with an example of memory configuration in Table 1. Channel, rank, and bank indexes are mapped to the lower significant bits of the logic address, in order to improve memory-level parallelism and effective bandwidth. However, interleaving data across channels/ranks destroys locality for NDP and always requires remote data access to other DRAM components. Instead of interleaving data across channels/ranks, the optimization-1 as shown in Fig. 6(b), maps the lower significant bits to column and row addresses to aggregate adjacent data within a rank locally. Still, another problem remains that the data are still interleaved across 16 chips in each rank. This chip-level interleaving means only 64B coarse-grained memory access is supported within a rank, which cannot be fully utilized in DNA seeding. Moreover, this prevents different chips from working in parallel. To solve above challenges, the optimization-2 is proposed as shown in Fig. 6(c).

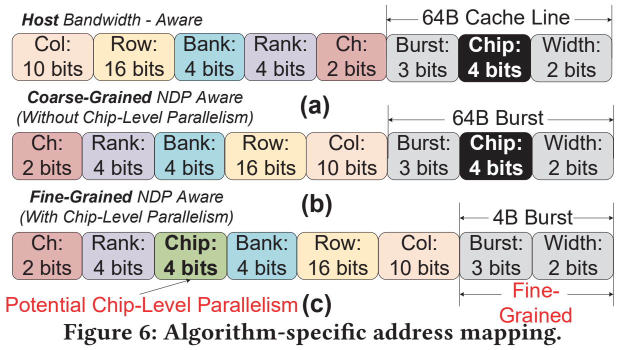


Figure 6: Algorithm-specific address mapping.

We consider CS signals as a part of the device address, which is discussed in detail in 'Enabling Individual Chip Select'. Originally, such chip selection address space is embedded in the 9 least significant bits of the logic address to access the 64B cacheline. We change this and map the chip index to more significant bits in the address. In this manner, adjacent data are stored in chips connected with the same DB and will not be stored in chips connected with the second DB until chips connected with the first DB are full. It improves the task locality so that inter-chip communication is minimized, and hence improves chip-level parallelism.

Bandwidth-Aware Data Mapping: The key idea of proposed data mapping is to fully leverage the available memory bandwidth from different chips through bandwidth-aware data placement. If there are enough free chips to hold duplicated index data, bandwidth-aware data mapping will duplicate the index and allow different copies of the index to be accessed in parallel. If the free chips are not enough to hold another copy of index data, bandwidth-aware data mapping will evenly map the index into all chips to fully leverage available memory bandwidth from those chips.

Enabling Individual Chip Select (ICS): As mentioned in Section 3, inter-task divergence prevents different DB-side accelerators working in SIMD style. Although the algorithm-specific address mapping provides potential for chip-level parallelism and fine-grained memory access, the lock-step working pattern in conventional DIMM prevents this from happening. We propose to enable the individual CS signal for each DRAM chip to overcome this challenge. We first introduce the problem of conventional DIMM, followed by the description of the proposed technique. Convention DIMM uses a shared CS signal for all DRAM chips in the same rank, causing the lock-step working pattern and making DIMM suffer from divergence in DNA seeding. Fig. 7 (upper part) shows the DB-side accelerators perform read operations on Chip-0 and Chip-1 in a lock-step manner. Since the read address is random, there is a whole *tRC* cycle between two reads. Even worse, only 50% of the output data (either from Chip-0 or Chip-1) is useful.

We propose the ICS technique, designing dedicated CS wires for each DRAM chip [33], controlled by the RCD-side MC. A disabled CS signal blocks the input command and the address, but the DRAM chip still receives the System Clock (CLK) signal, the Clock Enable (CKE) signal, and keeps working on previous memory commands.

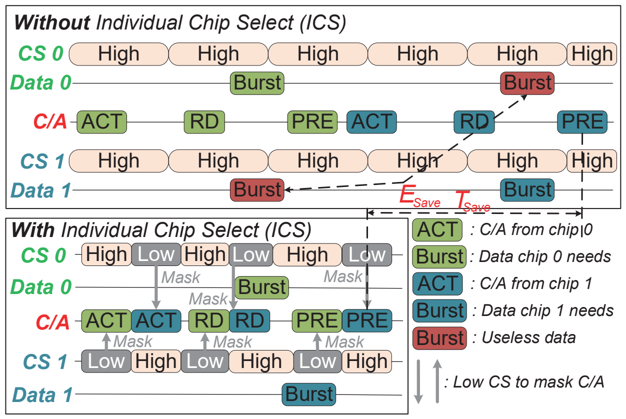


Figure 7: With/Without Individual Chip Select (ICS).

Fig. 7 (lower part) shows latency and energy saving with the ICS technique. We first enable Chip-0 and disable Chip-1 with the CS signals, and then strobe the first activation commend and address. The first C/A are only taken by the enabled Chip-0. Right after that, we switch the CS signal, enabling Chip-1 but disabling Chip-0. The second C/A are then taken by the enabled Chip-1. The disabled Chip-0 locks out the second activation command but keeps working on the previous command it took. Similarly, we send read, precharge command to Chip-0 and Chip-1, respectively, and get data from them one after another. By adopting proposed ICS technique, the latency is reduced due to pipelined commands. Furthermore, all output data are fully useful, from where the energy is saved.

4.3 Inter-Rank Design for Scaling Out

In this subsection, we look at applications with larger memory footprint, involving multiple ranks.

Besides adopting the intra-rank optimizations described above, we further propose four methods to reduce the inter-rank communication overhead, overcoming the big data challenge. First, we will describe two methods to support the inter-rank scaling out without modification to the current hardware. Then, we describe how to use the incoming NVDIMM hardware to address the issue of scalability. Finally, we propose an algorithm-specific data compression scheme, which can work with any of these three methods above, reducing memory footprint and the communication overhead.

Support Inter-Rank Comm. with CPU-polling: Our goal is to support the inter-rank scaling out without modifications on either the host or DIMM hardware. To this end, we leverage the host CPU to poll all connected DIMMs periodically. If an inter-rank data access is requested from a rank, the host will coordinate the data transfer. Note that AIM [10] deals with the similar scaling out problem by designing an additional bus across DIMMs. In addition to the design simplicity, our method can achieve 3.43x speedup and 2.89x energy reduction, compared with AIM (details in Section 6).

The CPU-polling based inter-rank communication works in the following steps, as shown in Fig. 8. ❶ The host issues a polling request to a DIMM. ❷ Address router redirects this polling request to the region of indicator bits in the Remote Data Buffer (RDB). ❸ If the bits fetched back to the host show remote data access is needed, the host issues another request to bring back the information about the remote data access. ❹ Address router redirects this request for remote data access information to the region of remote data info in the RDB. ❺ After receiving the information about remote data access, the host issues memory access request to the destination DIMM and fetches back the data. ❻ The host sends the target data back to the DB that needs the data.

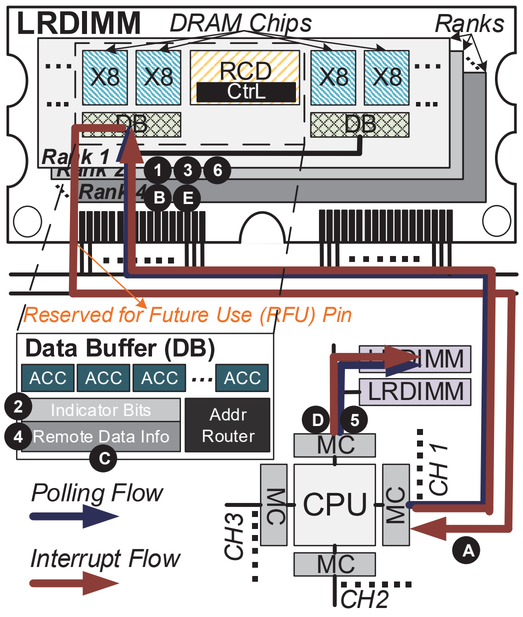


Figure 8: Work flow of polling-based and interrupt-based inter-rank communication.

Support Inter-Rank Comm. with Interruption: The polling-based method suffers from occupying the host and the DDR bus even without data transfer. Due to the occupancy of the memory bus during polling, the effective bandwidth for the memory bus to transfer data is reduced. In addition, since polling operation is read operation in MEDAL and proposed host-prioritized request scheduling will be applied, extra latency is needed, leading to performance degradation. To solve above issues, we propose to leverage the interrupt mechanism and the Reserved for Future Use (RFU) pin in LRDIMM [49], so that requests from the DB-side accelerators will notify CPU through the RFU pin which will be connected to the Advanced Programmable Interrupt Controller (APIC).

Specifically, the interrupt based inter-rank communication works in following steps, as shown in Fig. 8, 🅐 The DB that needs to access remote data issues interrupt signal to the host via the RFU pin. 🅑 The host issues request to the DB to bring back the information about remote data access. 🅒 Address router redirects the request for remote data access information to the region of remote data info in the RDB. 🅓 After receiving the information about remote data access, the host issues memory access request to the destination DIMM and fetches the data back. 🅔 The host sends the target data back to the DB that needs the data.

Host-prioritized Request Scheduling: As mentioned previously, since both the host-side MC and RCD-side MC can send requests to the DRAM and the host-side MC is not aware of requests from the RCD-side MC, timing issue may arise. Request scheduling is needed to satisfy the DDR timing constraint for the host-side MC.

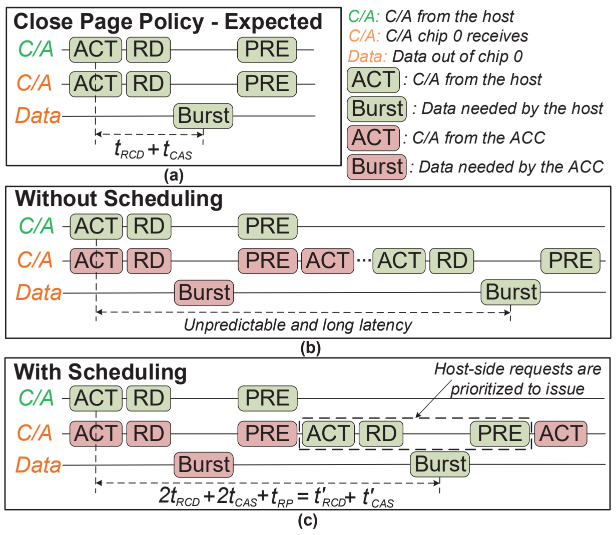


Figure 9: Host-prioritized request scheduling.

We choose to implement close-page policy in the host-side MC and design a host-prioritized request scheduling for the RCD-side MC. As shown in Fig. 9, with close-page policy, the host-side MC will expect its memory requests to the DRAM to be back after $t_{RCD} + t_{CAS}$. However, because the RCD-side MC also issues memory requests to the DRAM, without specific scheduling, the latency for memory requests from the host-side MC is unpredictable and there will be issues with the DDR timing constraint. To address this issue, RCD-side MC follows host-prioritized request scheduling to serve memory requests from the host as soon as the DRAM finishes its current task. For the host-side MC, we modify its DDR timing parameters, i.e., t_{RCD} and t_{CAS}, so the host-side MC has a longer expectation of the data return time to allow the RCD-side MC to be able to schedule those requests.

Reduce Inter-Rank Traffic with NVDIMM: Both the polling-based and the interruption-based techniques serve the goal of supporting the inter-rank communication. Further, we propose the NVDIMM-P approach to eliminate the inter-rank communication.

Different from NVDIMM-F/N, which either requires pairing a storage DIMM near the memory DIMM or only leverages Non-Volatile Memory (NVM) on DIMM for backup purpose. NVDIMM-P integrates both DRAM and NVM on the same DIMM and is close to release [44, 48]. Alongside DRAM, NVM on NVDIMM-P can also be memory-mapped, e.g., Intel Optane Technology [24]. With much higher capacity, NVM can act as a near-memory cache. Different from NVDIMM-F/N, in NVDIMM-P, the host and the DB can have byte accessibility to both the DRAM and the NVM.

We leverage NVMs, which can be up to 10× denser than DRAM [53], on NVDIMM-P to eliminate the inter-rank traffic. Specifically, we place index data used to be stored in remote ranks into the NVM locally. The work flow is described below, as shown in Fig. 10 (a). ❶ DRAM will be accessed if the target data is within DRAM. ❷ Otherwise, the memory request will go to the NVM. NVDIMM-P based MEDAL converts remote memory accesses to remote ranks into local memory accesses to on-DIMM NVM.

Reduce Memory Footprint and Comm. with Data Compression: To reduce the memory footprint and communication, we propose an algorithm-specific data compression. Note that proposed data compression can work together with all designs described above and provide additional benefits.

(1) Counters as the key data structure: During DNA seeding, the occurrence array $O_R(x, i)$ needs to be accessed frequently. The entry $O_R(x, i)$ is the occurrence of a nucleotide x before the i^{th} symbol of B_R. An example is shown in Fig. 10 (b) and (c), there are 2 A, 1 T, 1 C, and no G in the first 4 nucleotides in B_R ($AATC$). $O_R(A, 3)$, $O_R(T, 3)$, $O_R(C, 3)$, and $O_R(G, 3)$ are 2, 1, 1, and 0, respectively. To summarize, the occurrence array $O_R(x, i)$ is an array of counters.

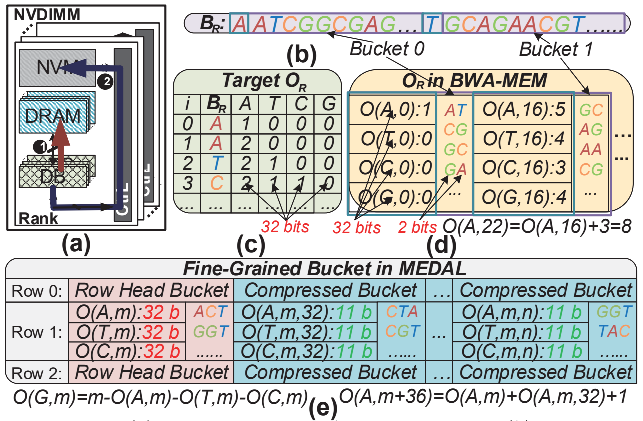

NVDIMM
NVM
DRAM
DB
Rank
❶
❷
(a)
B_R: A AATCGGCGAG… T GCAGAACGT……
Bucket 0
Bucket 1
(b)
Target O_R
i | B_R | A | T | C | G
0 | A | 1 | 0 | 0 | 0
1 | A | 2 | 0 | 0 | 0
2 | T | 2 | 1 | 0 | 0
3 | C | 2 | 1 | 1 | 0
(c)
32 bits
O_R in BWA-MEM
O(A,0):1
O(T,0):0
O(C,0):0
O(G,0):0
AT CG GC GA
O(A,16):5
O(T,16):4
O(C,16):3
O(G,16):4
GC AG AA CG
32 bits
2 bits
O(A,22)=O(A,16)+3=8
(d)
Fine-Grained Bucket in MEDAL
Row 0: Row Head Bucket | Compressed Bucket | … | Compressed Bucket
Row 1: O(A,m):32 b ACT | O(A,m,32):11 b CTA | … | O(A,m,n):11 b GGT
O(T,m):32 b GGT | O(T,m,32):11 b CGT | O(T,m,n):11 b TAC
O(C,m):32 b …… | O(C,m,32):11 b …… | O(C,m,n):11 b ……
Row 2: Row Head Bucket | Compressed Bucket | … | Compressed Bucket
O(G,m)=m-O(A,m)-O(T,m)-O(C,m)
O(A,m+36)=O(A,m)+O(A,m,32)+1
(e)

Figure 10: (a) Processing within NVDIMM. (b) Burrows-Wheeler Transform of the reference sequence. (c) The target O-table. (d) Bucket structure in BWA-MEM. (e) Fine-grained bucket in MEDAL with two types of buckets, i.e., row head bucket and compressed bucket.

(2) Bucket data structure in software: As shown in Fig. 10 (c), for human genome, each entry in $O_R(x, i)$ is 32-bit, the size of $O_R(x, i)$ is $4 \times 32/2 = 64\times$ larger than the size of B_R. To reduce the memory footprint, the widely used software, i.e., BWA-MEM, leverages a data structure called 'Bucket'. A bucket consists of a bucket head and a bucket body. The bucket head is a checkpoint, storing the up-to-date values of $O_R(x, i)$. B_R is stored within the bucket body. In this manner, when an entry in $O_R(x, i)$ is needed, BWA-MEM will first locate the target bucket. Then, the values of $O_R(x, i)$ in the bucket head and nucleotides in B_R within the bucket body will be read out. Finally, the values of target $O_R(x, i)$ can be reconstructed with the up-to-date values in the checkpoint and B_R via counting. As shown in Fig. 10 (d), with this bucket structure, only values of $O_R(x, i)$ in the bucket head are 32-bit, nucleotide in the bucket body is only 2-bit.

(3) Compressed fine-grained bucket: Observing that the precision of data within the bucket head is much higher than that of data in the bucket body, the key idea of data compression is to reduce the precision of data in the bucket head via fine-grained checkpoint. Compared with BWA-MEM, instead of storing global checkpoints, fine-grained checkpoints are used with proposed data compression.

There are two types of buckets with proposed data compression, i.e., row head bucket and compressed bucket. As shown in Fig. 10 (e), with data compression, each DRAM row begins with a row head bucket, containing a global checkpoint with up-to-data values for $O_R(x, i)$. The row head bucket is followed by many compressed buckets, which contains a fine-grained, local checkpoint for only $O_R(x, i)$ in this row, meaning much lower precision is enough for data in the bucket head of compressed bucket.

With proposed data compression, MEDAL first accesses the bucket head in the row head bucket. Then, it retrieves the target compressed bucket. Next, the target $O_R(x, i)$ can be derived by adding the local counters from the compressed bucket with the global counters in the row head bucket. Further, in the bucket head, we only store 3 values, the last value can be derived by subtraction.

5 DISCUSSION

Extension to Other Applications: MEDAL solve the problem of fine-grained random memory access. Our future work will extend MEDAL to other applications by replacing the logics in the DBs with general purpose processors or FPGAs. We expect applications such as graph processing [4], database searching [31], and sparse matrix computing [41] will also benefit from the proposed techniques.

Interface Choice: Similary to [6, 50, 57], we choose DDR as the interface for MEDAL due to two reasons: First, with DDR as the interface, we can configure the DIMMs into regular memory when no DNA seeding is performed, providing more flexibility; Second, DIMM based approach can be easily scale out. Note that the optimizations/techniques proposed can be easily applied to build PCIe/IO based accelerator.

System Integration and User Interface: MEDAL does not require modification of either the DRAM components or the CPU chip. MEDAL connects to the system with standard DDR bus, i.e, with DIMM slots. As described in Section 4.2, the host controls MEDAL with memory instructions.

To this end, the software stack needs modifications. Similar to other NDP/PIM solutions [4, 54, 57], we will need the OS to reserve the memory space in DIMMs to MEDAL, and provide I/O mapping for these space, so that the user can access the space with the sense of their physical address. Memory channels performing DNA seeding will be dedicated to this task. The host can work on other tasks with data mapped to other memory channels. The programming model of MEDAL is similar to CUDA. We will provide an Application Programming Interface (API) for programmers to control the application memory space allocation for MEDAL and a *memcpy* function to copy data between the user memory space and the application memory space of MEDAL. With the data ready in the memory space of MEDAL, users can launch the accelerator to perform DNA seeding.

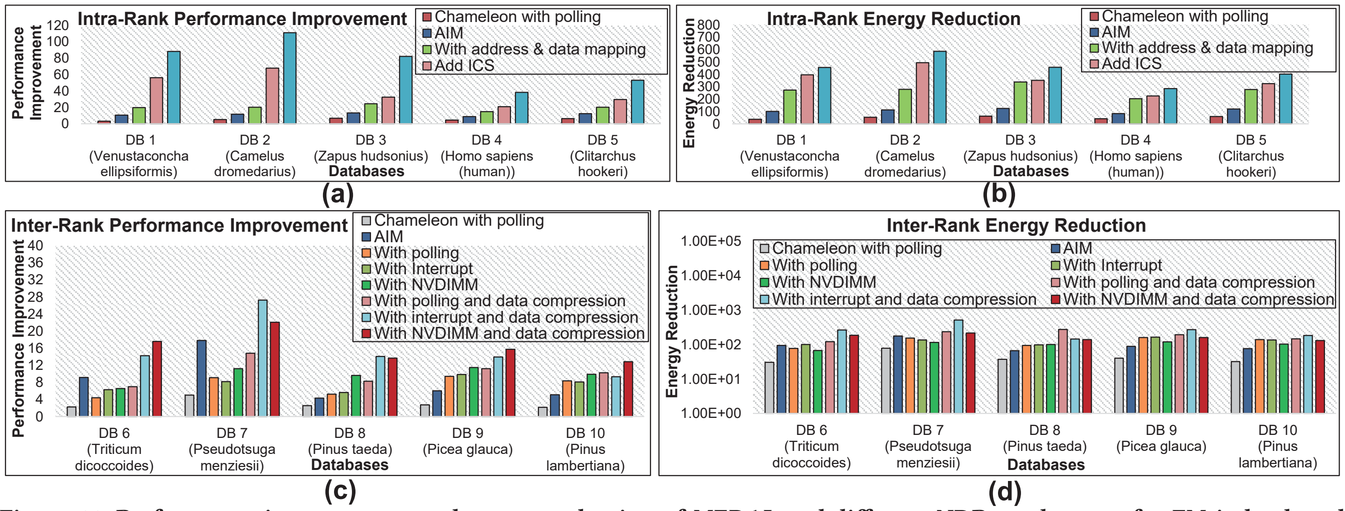


Figure 11: Performance improvement and energy reduction of MEDAL and different NDP accelerators for FM-index based DNA seeding. Results are normalized to that of a 16-thread CPU. (a). Intra-rank performance improvement. (b). Intra-rank energy reduction. (c). Inter-rank performance improvement. (d). Inter-rank energy reduction.

Table 1: Configure of the Server and MEDAL

Configuration of the Server	
CPU Model	Intel Xeon E5-2680 v3
CPU Clock Frequency (GHz)	2.50
Memory Capacity (GB)	400
L1 (KB)/L2 (KB)/L3 (MB) Cache	64 / 256 / 32
Configuration of MEDAL	
Memory Capacity (GB)	384
Memory Channels	4
DIMMs per Memory Channels	3
Ranks per DIMM	4
DRAM Chips per Rank	16
DRAM Chips per DB	2
Parameters of DDR4 DRAM	
Capacity	4Gb × 4
Bank Groups	2
Banks per BankGroup	2
Clock Frequency (1/tCK)	1,200MHz
tRCD-tCAS-tRP (ns)	16-16-16

Table 2: Design Parameters of Customized Logics in DB

Module	Latency (Cycles)	Power (mW)	Leakage (uW)	Area (um^2)
Addr Trans	20	4.05	18.39	4600.35
SMEM	1	6.00	13.46	3325.45
Suffix	5	0.52	4.31	1015.59

6 EXPERIMENTAL RESULTS

The experimental setup, results, and analysis of the experimental results are presented in this section.

6.1 Experimental Setup

Configuration of the Baseline: The baseline for FM-index based DNA seeding and Hash-index based DNA seeding are BWA-MEM [34] and SMALT [46], respectively, running in a server with an Intel Xeon E5-2680 v3 CPU. The detailed configuration information of the server is shown in Table 1.

Configuration of MEDAL: Ramulator [30] is modified to build a cycle-accurate simulator for MEDAL. The configuration of MEDAL is shown in Table 1. The timing, energy, and area parameters of the DB customized logics are estimated by pre-layout Design Compiler [51] with 28 nm technology[1]. We set timing constraint as 1.2GHz using tt design corner. Since it is a very simple circuit with lots of design slacks, we expect similar post-layout results. Those parameters of customized logics in DB are shown in Table 2. Since the address router only involves a few comparators to decide which components the read command should go to, it's not included in Table 2. The timing parameters of DRAM chip used in our experiments are shown in Table 1. The energy consumption of DRAM is derivated by feeding the command trace of DRAM from Ramulator to DRAMPower [8]. We use the parameters of energy for datapath from CACTI-IO [27]. The timing and energy parameters for NVM in NVDIMM are estimated with Intel's Optane memory [23, 25]. The correctness of our simulation is guaranteed, since the hardware design follows the same (1) computing arithmetic, (2) execution order, and (3) data access order as the software. Our simulator ensures correctness by using traces from the software.

NDP Accelerators for Comparison: We modified Ramulator as well to build cycle-accurate simulators for Chameleon and AIM. We use the same memory configuration for those two accelerators. For Chameleon, because it doesn't support any kind of communication, we add polling-based communication mechanism to it.

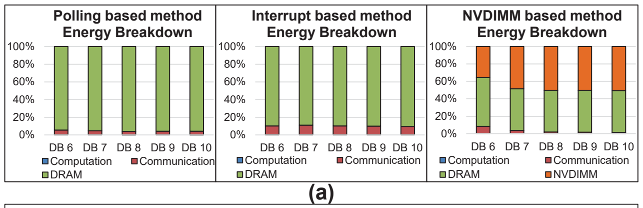
Databases: Ten different genomes with different sizes from 1.59 billion bases to 27.60 billion bases from NCBI [43] are used in our experiments. The name of those ten databases are shown in Fig. 11. We name them as DB1 to DB10 for short in other figures.

Query Sequences: Ten million query sequences with length of 101 were extracted exactly from corresponding genomes.

6.2 Intra-Rank Evaluation

For small databases which can be fitted in a single DRAM rank, the performance and energy-efficiency comparisons between MEDAL and other DIMM based NDP accelerators for DNA seeding are shown in Fig. 11 (a) and Fig. 11 (b). All results are normalized to the that of the 16-thread CPU.

For intra-rank tasks, with only proposed address and data mapping, MEDAL outperforms the 16-thread CPU, Chameleon, and AIM by 19.62x, 3.91x, and 1.75x. Then, ICS improves the performance of



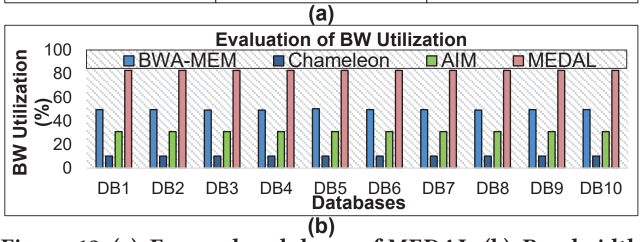
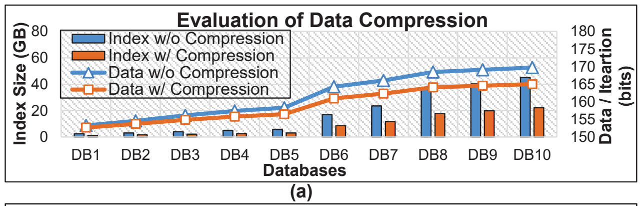


Figure 12: (a). Energy breakdown of MEDAL. (b). Bandwidth utilization of different platforms.



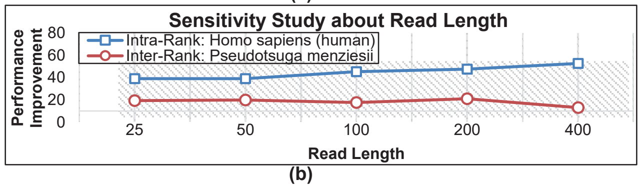


Figure 13: (a). Evaluation of data compression. (b).Sensitivity study about read length.

MEDAL by 1.92x via enabling efficient fine-grained memory access and chip-level parallelism. Further, the algorithm-specific data compression improves the performance of MEDAL by 1.85x, because it effectively reduces the memory footprint and leaves more space for data mapping to utilize. Putting all proposed techniques together, MEDAL outperforms the 16-thread CPU, Chameleon, and AIM by 69.69x, 13.90x, and 6.23x, respectively.

As comparison, AIM performs coarse-grained memory access without extra memory bandwidth. Chameleon performs fine-grained memory access. However, most data fetched out of memory in Chameleon is useless due to the inter-task divergence of DNA seeding and the SIMD-style processing in Chameleon. Also, there is no chip-level parallelism in Chameleon. The good performance of MEDAL, compared with others, comes from its full parallelism, i.e., both rank-level and chip-level parallelism, and its fine-grained memory access with high bandwidth utilization.

Energy-wise, MEDAL reduces energy consumption of the 16-thread CPU, Chameleon, and AIM by 426.27x, 8.54x, and 3.95x, respectively. High bandwidth utilization and short processing time contribute to its the high energy efficiency.

6.3 Inter-Rank Evaluation

For databases that cannot be fitted within a single rank and need inter-rank communication, similarly, the comparisons are shown in Fig. 11 (c) and Fig. 11 (d).

For inter-rank tasks, on average, polling-based design outperforms the 16-thread CPU, Chameleon, and AIM by 9.97x, 3.57x, and 1.35x, respectively. Interrupt-based design outperforms the above platforms by 14.81x, 5.31x, and 2.01x, respectively. NVDIMM-based design outperforms them by 16.09x, 5.76x, and 2.19x, respectively.

Compared with polling-based design, interrupt-based design has better performance due to two reasons. First, interrupt-based design doesn't need to occupy the memory channel for polling operation, meaning no negative effect on data transfer. Second, polling operation is read operation and due to proposed host-prioritized request scheduling, extra latency is needed for read operation from the host, which will degrade the performance. Polling-based design, on the other hand, requires less modifications. For example, it doesn't require to connect RFU to APIC in the host and add hardware interrupt vector. The superior strength of NVDIMM based approach is that we can use off-the-shelf NVDIMM to deal with the issue of communication without occupation of the memory channel and it provides very good performance.

Energy-wise, polling-based design reduces energy consumption of the 16-thread CPU, Chameleon, and AIM by 185.95x, 4.54x, and 1.97x, respectively. Interrupt-based design reduces energy consumption of above platforms by 251.08x, 6.13x, and 2.66x, respectively. NVDIMM-based design reduces energy consumption of above platforms by 164.18x, 4.01x, and 1.74x, respectively.

6.4 Energy Breakdown

The energy breakdown for MEDAL is in Fig. 12 (a). For all designs, computation consumes less than 1.0% energy. Because DNA seeding only involves simple integer operations, customized lightweight logics will be much more efficient. As for communication, it consumes 10.0% energy at most, which means our communication mechanisms are energy-efficient. For polling-based design, interrupt-based design, and NVDIMM-based design, DRAM consumes 95.4%, 89.9%, and 48.9% energy, respectively. DRAM's domination on the energy consumption is due to the energy-efficient lightweight logics and communication mechanisms. For NVDIMM-based approach, NVM consumes 48.2% energy on average. The portion of energy consumed by NVM grows with the size of databases, because the larger the database, the higher the possibility that memory requests will go to NVM.

6.5 Bandwidth Utilization

We define bandwidth utilization as the ratio between useful data and the actual amount of data fetched out the memory. As shown in Fig. 12 (b), on average, MEDAL has the highest bandwidth utilization ratio - 82.81%, while Chameleon has the lowest bandwidth ratio - only 10.29%.

The high bandwidth utilization ratio of MEDAL comes from its fine-grained memory accessibility. Proposed address mapping provides potential for fine-grained memory access and ICS makes it reality. As comparison, coarse-grained memory access lags the bandwidth utilization ratio of AIM. For Chameleon, since there is no optimization for non-SIMD processing, data from most DRAM chips become useless, reducing its bandwidth utilization ratio.

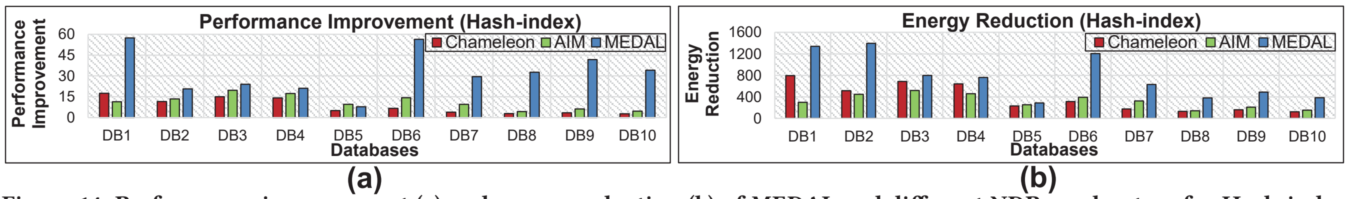


Figure 14: Performance improvement (a) and energy reduction (b) of MEDAL and different NDP accelerators for Hash-index based DNA seeding. Results are normalized to that of a 16-thread CPU.

6.6 Performance of Data Compression

As shown in Fig. 13 (a), data compression reduces the sizes of DNA indexes for 48.9% on average, leading to reduction in memory footprint and providing more space for data mapping to utilize. In addition, the amount of data needs to be fetched each iteration is also reduced due to the smaller size of compressed bucket. Thus, there is no extra memory accesses and performance degradation after data compression.

6.7 Sensitivity Study about Read Length

Reads with length of 101 are typical with the next generation sequencing technology [12, 18], thus we choose 101 as the representative length. In fact, MEDAL can support reads with different lengths. The experimental results on an intra-rank case and an inter-rank case with reads with various length is in Fig. 13 (b). For the intra-rank design, MEDAL with interrupt provides higher speedup (over CPU) for longer reads due to benefits from more memory traffic. For the inter-rank design, the performance is stable with respect to the read length, because communication compensates the benefits from more memory traffic. When the read length is even longer, in which cases other algorithms are used, e.g., D-SOFT in Darwin [52], we can change customized logics inside the DBs to match the algorithms. We expect similar performance gain, since the seeding of ultra-long read is still memory bound, which MEDAL is good at.

6.8 Hash-index based DNA Seeding Algorithm

For Hash-index based DNA seeding, the experimental rsults are shown in Fig. 14 (a) and Fig. 14 (b). The experimental results show that MEDAL outperforms the 16-thread CPU, Chameleon, and AIM by 28.60x, 4.33x, and 2.90x, respectively. About the energy-efficiency, MEDAL outperforms above platforms by 668.95x, 2.22x, and 2.27x, respectively.

7 RELATED WORK

This section introduces related work of MEDAL.

Accelerator for DNA Seeding: Most previous work proposes their designs with FPGA [9, 14–16] and GPU [38, 40] to deal with DNA seeding. Darwin [52] and GenAx [18] are two accelerators for DNA alignment based on ASIC design and automata. Both of them majorly focus on DNA extension. For the seeding part, both of them use hash-index, instead of FM-index, while MEDAL majorly focuses on FM-index and can deal with hash-index as well.

Although above architectures have enough computation capability, data movement is a serious issue. Moreover, as mentioned before, DNA seeding is bounded by memory, there is limited improvement space for approaches which only optimize computation. In contrast, MEDAL focuses on memory and performs computation near the data.

NDP Solutions for DNA Seeding: MPU-BWM leverages HMC to accelerate DNA seeding by placing a RISC-V core in the logic die [55]. However, 3D-stacked memory is not cost-efficient, has limited capacity [47], and doesn't provide extra internal bandwidth [11]. AIM [10] attaches FPGAs and dedicated buses to DIMMs to accelerate DNA seeding. However, AIM doesn't leverage rank-level parallelism in DIMMs, leading to limited performance improvement. Compared with AIM, MEDAL specifically optimizes fine-grained memory access with address mapping, data mapping, and individual chip selection, providing more bandwidth and parallelism. Furthermore, MEDAL explores four novel approaches to enhance the scalability. Chameleon [6] provides a general-purpose, SIMD-style, DIMM based NDP architectures. However, because Chameleon focuses on SIMD-style processing, even after a communication mechanism is added, it's not good at DNA seeding. MEDAL outperforms AIM and Chameleon by 3.43x and 8.37x on average. EMU Technology also has the potential to accelerate DNA seeding by coupling lightweight logics to memory and migrating threads [13].

PIM Solutions for DNA Seeding: UPMEM modifies the DRAM die to accelerate DNA alignment [54, 57]. RADRA [21] and PRinS [29] leverage Resistive Random Access Memory (ReRAM) to perform DNA seeding and Seed Extension. Above platforms either require modifications to the DRAM die or are long-term architecture. In contrast, MEDAL leverages off-the-shelf DRAM components and is a practical approach.

8 CONCLUSION

To accelerate DNA seeding cost and energy efficient, we propose MEDAL, a practical and energy efficient NDP architecture. For small databases, we propose the intra-rank design, together with an algorithm-specific address mapping, bandwidth-aware data mapping, and Individual Chip Select (ICS) to address the challenge of random memory access, improving parallelism and bandwidth utilization. Furthermore, to address the challenge of scalability, we propose three inter-rank designs (polling-based communication, interrupt-based communication, and NVDIMM-based solution). In addition, we propose an algorithm-specific data compression technique to reduce memory footprint, introduce more space for the data mapping, and reduce the communication overhead. Experimental results show that for three proposed designs, on average, MEDAL can achieve 30.50x/8.37x/3.43x speedup and 289.91x/6.47x/2.89x energy reduction when compared with a 16-thread CPU baseline and two state-of-the-art NDP accelerators, respectively.

ACKNOWLEDGMENTS

This work was supported in part by NSF 1816833, 1533933, 1719160, 1730309, and CRISP, one of six centers in JUMP, a SRC program sponsored by DARPA.

REFERENCES

[1] 2019. 28nm Technology Libraries. https://www.faraday-tech.com/cn/category/BrowseByTechnology?method=browserCategory&tech=28nm&master.ipSearchForm.technology=28nm.

[2] Nauman Ahmed, Koen Bertels, and Zaid Al-Ars. 2016. A comparison of seed-and-extend techniques in modern DNA read alignment algorithms. In *Bioinformatics and Biomedicine (BIBM), 2016 IEEE International Conference on*. IEEE, 1421–1428.

[3] Nauman Ahmed, Vlad-Mihai Sima, Ernst Houtgast, Koen Bertels, and Zaid Al-Ars. 2015. Heterogeneous hardware/software acceleration of the BWA-MEM DNA alignment algorithm. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*. IEEE Press, 240–246.

[4] Junwhan Ahn, Sungpack Hong, Sungjoo Yoo, Onur Mutlu, and Kiyoung Choi. 2016. A scalable processing-in-memory accelerator for parallel graph processing. *ACM SIGARCH Computer Architecture News* 43, 3 (2016), 105–117.

[5] ARM. 1999. AMBA Specification (Rev 2.0). (1999).

[6] Hadi Asghari-Moghaddam, Young Hoon Son, Jung Ho Ahn, and Nam Sung Kim. 2016. Chameleon: Versatile and practical near-DRAM acceleration architecture for large memory systems. In *Microarchitecture (MICRO), 2016 49th Annual IEEE/ACM International Symposium on*. IEEE, 1–13.

[7] Trevor E Carlson, Wim Heirman, and Lieven Eeckhout. 2011. Sniper: exploring the level of abstraction for scalable and accurate parallel multi-core simulation. In *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*. ACM, 52.

[8] Karthik Chandrasekar, Christian Weis, Yonghui Li, Sven Goossens, Matthias Jung, Omar Naji, Benny Akesson, Norbert Wehn, and Kees Goossens. 2012. DRAM-Power: Open-source DRAM power & energy estimation tool. *URL: http://www.drampower. info* 22 (2012).

[9] Mau-Chung Frank Chang, Yu-Ting Chen, Jason Cong, Po-Tsang Huang, Chun-Liang Kuo, and Cody Hao Yu. 2016. The SMEM Seeding Acceleration for DNA Sequence Alignment. In *Field-Programmable Custom Computing Machines (FCCM), 2016 IEEE 24th Annual International Symposium on*. IEEE, 32–39.

[10] Jason Cong, Zhenman Fang, Michael Gill, Farnoosh Javadi, and Glenn Reinman. 2017. AIM: accelerating computational genomics through scalable and noninvasive accelerator-interposed memory. In *Proceedings of the International Symposium on Memory Systems*. ACM, 3–14.

[11] Hybrid Memory Cube Consortium. 2013. Hybrid memory cube specification 1.0. *Last Revision Jan* (2013).

[12] Mark A DePristo, Eric Banks, Ryan Poplin, Kiran V Garimella, Jared R Maguire, Christopher Hartl, Anthony A Philippakis, Guillermo Del Angel, Manuel A Rivas, Matt Hanna, et al. 2011. A framework for variation discovery and genotyping using next-generation DNA sequencing data. *Nature genetics* 43, 5 (2011), 491.

[13] EMUTechnology. 2017. EMU Architecture. https://www.emutechnology.com/technology/.

[14] Edward Fernandez, Walid Najjar, and Stefano Lonardi. 2011. String matching in hardware using the FM-index. In *Field-Programmable Custom Computing Machines (FCCM), 2011 IEEE 19th Annual International Symposium on*. IEEE, 218–225.

[15] Edward B Fernandez, Walid A Najjar, Stefano Lonardi, and Jason Villarreal. 2012. Multithreaded FPGA acceleration of DNA sequence mapping. In *High Performance Extreme Computing (HPEC), 2012 IEEE Conference on*. 1–6.

[16] Edward B Fernandez, Jason Villarreal, Stefano Lonardi, and Walid A Najjar. 2015. FHAST: FPGA-based acceleration of Bowtie in hardware. *IEEE/ACM Transactions on Computational Biology and Bioinformatics (TCBB)* 12, 5 (2015), 973–981.

[17] National Center for Biological Information. 2018. GenBank and WGS Statistics. https://www.ncbi.nlm.nih.gov/genbank/statistics/.

[18] Daichi Fujiki, Aran Subramaniyan, Tianjun Zhang, Yu Zeng, Reetuparna Das, David Blaauw, and Satish Narayanasamy. 2018. GenAx: a genome sequencing accelerator. In *Proceedings of the 45th Annual International Symposium on Computer Architecture*. IEEE Press, 69–82.

[19] Laiq Hasan, Yahya M Khawaja, and Abdul Bais. 2008. A Systolic Array Architecture for the Smith-Waterman Algorithm with High Performance Cell Design.. In *IADIS European Conf. Data Mining*. 35–44.

[20] Ernst Joachim Houtgast, Vlad-Mihai Sima, Koen Bertels, and Zaid Al-Ars. 2016. GPU-Accelerated BWA-MEM Genomic Mapping Algorithm Using Adaptive Load Balancing. In *International Conference on Architecture of Computing Systems*. Springer, 130–142.

[21] Wenqin Huangfu, Shuangchen Li, Xing Hu, and Yuan Xie. 2018. RADAR: a 3D-ReRAM based DNA alignment accelerator architecture. In *2018 55th ACM/ESDA/IEEE Design Automation Conference (DAC)*. IEEE, 1–6.

[22] National Human Genome Research institute. 2018. DNA Sequencing Costs: data. https://www.genome.gov/27541954/dna-sequencing-costs-data/.

[23] Intel. 2018. Intel Optane DC Persistent Memory. https://www.intel.com/content/www/us/en/products/docs/memory-storage/optane-persistent-memory/what-is-intel-optane-dc-persistent-memory-video.

[24] Intel. 2018. Intel's 3D XPoint Technology Products. https://software.intel.com/en-us/articles/3d-xpoint-technology-products.

[25] Joseph Izraelevitz, Jian Yang, Lu Zhang, Juno Kim, Xiao Liu, Amirsaman Memaripour, Yun Joon Soh, Zixuan Wang, Yi Xu, Subramanya R Dulloor, Jishen Zhao, and Steven Swanson. 2019. Basic Performance Measurements of the Intel Optane DC Persistent Memory Module. *arXiv preprint arXiv:1903.05714* (2019).

[26] Hanlee Ji Jay Shendure. 2008. Next-generation DNA sequencing. *Nature biotechnology* 26, 10 (2008), 1135–1145.

[27] Norman P Jouppi, Andrew B Kahng, Naveen Muralimanohar, and Vaishnav Srinivas. 2015. CACTI-IO: CACTI with off-chip power-area-timing models. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 23, 7 (2015), 1254–1267.

[28] Jung-Yup Kang, Sandeep Gupta, and J-L Gaudiot. 2004. Accelerating the kernels of BLAST with an efficient PIM (processor-in-memory) architecture. In *Computational Systems Bioinformatics Conference, 2004*. IEEE, 552–553.

[29] Roman Kaplan, Leonid Yavits, Ran Ginosar, and Uri Weiser. 2017. A resistive CAM Processing-in-Storage architecture for DNA sequence alignment. *arXiv preprint arXiv:1701.04723* (2017).

[30] Yoongu Kim, Weikun Yang, and Onur Mutlu. 2016. Ramulator: A Fast and Extensible DRAM Simulator. *Computer Architecture Letters* 15, 1 (2016), 45–49.

[31] Onur Kocberber, Boris Grot, Javier Picorel, Babak Falsafi, Kevin Lim, and Parthasarathy Ranganathan. 2013. Meet the walkers: Accelerating index traversals for in-memory databases. In *Proceedings of the 46th Annual IEEE/ACM International Symposium on Microarchitecture*. ACM, 468–479.

[32] Ben Langmead and Steven L Salzberg. 2012. Fast gapped-read alignment with Bowtie 2. *Nature Methods* 9 (04 03 2012), 357 EP –. https://doi.org/10.1038/nmeth.1923

[33] Yebin Lee, Soontae Kim, Seokin Hong, and Jongmin Lee. 2013. Skinflint DRAM system: Minimizing DRAM chip writes for low power. In *High Performance Computer Architecture (HPCA2013), 2013 IEEE 19th International Symposium on*. IEEE, 25–34.

[34] Heng Li. 2013. Aligning sequence reads, clone sequences and assembly contigs with BWA-MEM. *arXiv preprint arXiv:1303.3997* (2013).

[35] Heng Li and Richard Durbin. 2009. Fast and accurate short read alignment with Burrows–Wheeler transform. *bioinformatics* 25, 14 (2009), 1754–1760.

[36] Isaac TS Li, Warren Shum, and Kevin Truong. 2007. 160-fold acceleration of the Smith-Waterman algorithm using a Field Programmable Gate Array (FPGA). *BMC bioinformatics* 8, 1 (2007), 185.

[37] Pei Liu, Ahmed Hemani, Kolin Paul, Christian Weis, Matthias Jung, and Norbert Wehn. 2017. 3D-stacked many-core architecture for biological sequence analysis problems. *International Journal of Parallel Programming* 45, 6 (2017), 1420–1460.

[38] Yongchao Liu and Bertil Schmidt. 2012. Evaluation of GPU-based seed generation for computational genomics using Burrows-Wheeler transform. In *Parallel and Distributed Processing Symposium Workshops & PhD Forum (IPDPSW), 2012 IEEE 26th International*. IEEE, 684–690.

[39] Yongchao Liu, Adrianto Wirawan, and Bertil Schmidt. 2013. CUDASW++ 3.0: accelerating Smith-Waterman protein database search by coupling CPU and GPU SIMD instructions. *BMC bioinformatics* 14, 1 (2013), 117.

[40] Mian Lu, Yuwei Tan, Ge Bai, and Qiong Luo. 2012. High-performance short sequence alignment with GPU acceleration. *Distributed and Parallel Databases* 30, 5-6 (2012), 385–399.

[41] Yusuke Nagasaka, Akira Nukada, and Satoshi Matsuoka. 2016. Adaptive multi-level blocking optimization for sparse matrix vector multiplication on GPU. *Procedia Computer Science* 80 (2016), 131–142.

[42] NCBI. [n. d.]. Genome of Ambystoma Mexicanum (Axolotl). https://www.ncbi.nlm.nih.gov/genome/381.

[43] NCBI. 2018. Genome Database. https://www.ncbi.nlm.nih.gov/genome.

[44] Kazuichi Oe, Takeshi Nanri, and Koji Okamura. 2016. Feasibility study for building hybrid storage system consisting of non-volatile DIMM and SSD. In *Computing and Networking (CANDAR), 2016 Fourth International Symposium on*. IEEE, 454–457.

[45] University of Cambridge Metabolic Research Labs. 2012. The BarraCUDA Project. http://seqbarracuda.sourceforge.net/.

[46] Dr Hannes Ponstingl. 2016. SMALT. http://www.sanger.ac.uk/science/tools/smalt-0.

[47] Seth H Pugsley, Jeffrey Jestes, Rajeev Balasubramonian, Vijayalakshmi Srinivasan, Alper Buyuktosunoglu, Al Davis, and Feifei Li. 2014. Comparing implementations of near-data computing with in-memory map reduce workloads. *IEEE Micro* 34, 4 (2014), 44–52.

[48] Arthur Sainio. 2016. NVDIMM: Changes are Here So What's Next. *In-Memory Computing Summit* (2016).

[49] Samsung. 2017. 288pin Load Reduced DIMM based on 4Gb E-die. https://www.samsung.com/semiconductor/global.semi/file/resource/2018/04/DDR4_4Gb_E_die_LRDIMM_Rev1.1_Jun.17.pdf.

[50] Vivek Seshadri, Donghyuk Lee, Thomas Mullins, Hasan Hassan, Amirali Boroumand, Jeremie Kim, Michael A Kozuch, Onur Mutlu, Phillip B Gibbons, and Todd C Mowry. 2017. Ambit: In-memory accelerator for bulk bitwise operations using commodity DRAM technology. In *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture*. ACM, 273–287.

[51] Synopsys. 2018. Design Complier. https://www.synopsys.com/support/training/rtl-synthesis/design-compiler-rtl-synthesis.html.

[52] Yatish Turakhia, Gill Bejerano, and William J Dally. 2018. Darwin: A genomics co-processor provides up to 15,000 x acceleration on long read assembly. In *ACM SIGPLAN Notices*, Vol. 53. ACM, 199–213.

[53] Theo Ungerer and Ing Dietmar Fey. 2016. Report on Disruptive Technologies for years 2020-2030. (2016).

[54] UPMEM. [n. d.]. UPMEM. https://www.upmem.com/use-cases/.

[55] Thiruvengadam Vijayaraghavan, Amit Rajesh, and Karthikeyan Sankaralingam. 2018. MPU-BWM: Accelerating Sequence Alignment. *IEEE Computer Architecture Letters* 17, 2 (2018), 179–182.

[56] Jing Zhang, Heshan Lin, Pavan Balaji, and Wu-chun Feng. 2013. Optimizing Burrows-Wheeler transform-based sequence alignment on multicore architectures. In *Cluster, Cloud and Grid Computing (CCGrid), 2013 13th IEEE/ACM International Symposium on*. IEEE, 377–384.

[57] Vasileios Zois, Divya Gupta, Vassilis J Tsotras, Walid A Najjar, and Jean-Francois Roy. 2018. Massively parallel skyline computation for processing-in-memory architectures. In *Proceedings of the 27th International Conference on Parallel Architectures and Compilation Techniques*. ACM, 1.