NEST: DIMM based Near-Data-Processing Accelerator for K-mer Counting

Wenqin Huangfu[1], Krishna T. Malladi[2], Shuangchen Li[1], Peng Gu[1], Yuan Xie[1]
[1]University of California, Santa Barbara [2]Samsung Semiconductor, Inc.
{wenqin_huangfu,shuangchenli,peng_gu,yuanxie}@ucsb.edu
{k.tej}@samsung.com

ABSTRACT

With the ability to help wildlife conservation, precise medical care, and disease understanding, genomics analysis is becoming more and moe important. Recently, with the development and wide adoption of the Next-Generation Sequencing (NGS) technology, bio-data grows exponentially, putting forward great challenges for *k*-mer counting - a widely used application in genomics analysis.

Many hardware approaches have been explored to accelerate *k*-mer counting. Most of those approaches are compute-centric, i.e., based on CPU/GPU/FPGA. However, the space for performance improvement is limited for compute-centric accelerators, because *k*-mer counting is a memory-bound application. By integrating memory and computation close together and embracing higher memory bandwidth, Near-Data-Processing (NDP) is a good candidate to accelerate *k*-mer counting. Unfortunately, due to challenges of communication, bandwidth utilization, workload balance, and redundant memory accesses, previous NDP accelerators for *k*-mer counting cannot fully unleash the power of NDP. To build a practical, scalable, high-performance, and energy-efficient NDP accelerator for *k*-mer counting, we perform hardware/software co-design and propose the DIMM based Near-Data-Processing Accelerator for *k*-mer counting (NEST). To fully unleash the potential of NEST architecture, we modify the *k*-mer counting algorithm and propose a dedicated workflow to support efficient parallelism. Moreover, the proposed algorithm and workflow are able to reduce unnecessary inter-DIMM communication. To improve memory bandwidth utilization, we propose a novel address mapping scheme. The challenge of workload balance is addressed with the proposed task scheduling technique. In addition, scattered memory access and task switching are proposed to eliminate redundant memory access. Experimental results show that NEST provides 677.33x/27.24x/6.02x performance improvement and 1076.14x/62.26x/4.30x energy reduction, compared with a 48-thread CPU, a CPU/GPU hybrid approach, and a state-of-the-art NDP accelerator, respectively.

1 INTRODUCTION

Genomics analysis is becoming more and more important and it is developing rapidly, since it is helpful to understanding of complex human disease [26], wildlife conservation [23], precise medical care, and so on [7]. As an example, the Next Generation Sequencing (NGS) technology helps a lot with characterization of the global pandemic Coronavirus Disease 2019 (COVID-19) [12, 22], which infects and causes death to thousands of people around the world after its outbreak in 2019.

With the improving throughput and cost efficiency of the NGS technology [32], bio-data is growing exponentially, putting forward great challenges to genomics analysis [14]. In genomics analysis, frequency information of *k*-mers (DNA subsequences with length of *k*) in the sequencing data is needed for many applications, including de novo genome assembly, repeat identification, error correction, variant calling, and so on [4, 15, 24]. For example, during DNA error correction, if a *k*-mer appears only once in the sequencing reads, this *k*-mer is assumed to contain sequencing errors and will be converted to other *k*-mers with higher frequencies via error correction [31]. *k*-mer counting occupies a significant portion of the runtime in many genomics workflows. For instance, as shown in Fig 1, *k*-mer counting is the most time consuming step in the de novo genome assembly, consuming nearly half of the total runtime in the entire de novo assembly pipeline [6]. Considering that the second time consuming step - 'Assembly' has drawn tremendous amount of attention and has been accelerated up to 710x [29, 34], *k*-mer counting becomes even more important.

Motivated by its importance, many compute-centric approaches, such as multi-core [6, 9], GPU [4, 10], and FPGA [3, 4] have been explored to accelerate *k*-mer counting. However, the acceleration of *k*-mer counting is non-trivial due to the large dataset size and characteristics of this application. *k*-mer counting is memory-bound and involves a large amount of irregular memory access [4, 15]. Moreover, since *k*-mer counting requires frequent random read/write to Bloom filters and hash table, memory bandwidth will be wasted without the ability to perform fine-grained random memory access [4, 25]. Conventional architectures above cannot address the memory bottleneck in *k*-mer counting, because they provide limited memory bandwidth with higher read/write latencies and there is no optimization for fine-grained random memory access. Besides conventional architectures, previous work leverages Near-Data-Processing (NDP) architectures to accelerate *k*-mer counting, because NDP architectures merge computation and memory closer to provide higher memory bandwidth and greatly reduce the overhead of data movement. For example, emerging monolithic 3D integration is utilized to accelerate *k*-mer counting in [15] and MEDAL provides a more practical approach via leveraging the Dual-Inline Memory Module (DIMM) to build accelerators with commercially available DRAM components [14].

ICCAD '20, November 2–5, 2020, Virtual Event, USA

ACM ISBN 978-1-4503-8026-3/20/11…$15.00
https://doi.org/10.1145/3400302.3415724

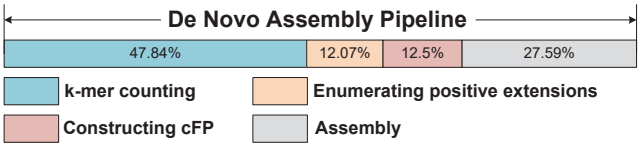

Figure 1: Time breakdown of the de novo assembly. *k*-mer counting dominates the runtime.

However, previous NDP accelerators have their own drawbacks, when they are used for *k*-mer counting. To be specific, monolithic 3D integration is an emerging technology, which means it's a long-term NDP architecture. Moreover, those 3D integration based architectures do not consider and optimize for fine-grained memory access. MEDAL is a near-term NDP architecture. Unfortunately, when MEDAL is used to perform *k*-mer counting, communication becomes the bottleneck. As the experiments show, for more than 60% percent of the time, the Processing Elements (PEs) in MEDAL are idle due to the communication. Furthermore, workload balance is a serious challenge in MEDAL and there is no optimization to deal with redundant memory accesses.

The goal of this paper is to address the challenges of performing *k*-mer counting with NDP architecture. Modified and optimized on the base of MEDAL, a practical, scalable, energy-efficient, and commodity DRAM components based NDP accelerator, i.e., NEST, is proposed. NEST has efficient communication, balanced workload, high bandwidth/PE utilization, and fine-grained memory accessibility. The main contributions of this paper are listed as follows.

- From the hardware perspective, we build a practical, scalable, high-performance, and energy-efficient NDP accelerator for *k*-mer counting, i.e., NEST, with off-the-shelf DRAM components.
- From the software perspective, we modify the *k*-mer counting algorithm and propose a dedicated workflow to enable parallel processing in NEST. Moreover, proposed algorithm and workflow are able to reduce unnecessary inter-DIMM communication.
- About the detailed optimizations, we enhance support for intra-DIMM communication, improve bandwidth/PE utilization, address the challenge of workload balance, and eliminate unnecessary memory accesses via architecture design, address mapping, task scheduling, and memory access management.
- We perform extensive experiments for the NEST architecture and proposed techniques. The experimental evaluation shows that NEST provides 677.33x/27.24x/6.02x performance improvement and 1076.14x/62.26x/4.30x energy reduction, compared with a 48-thread CPU, a CPU/GPU hybrid approach, and a state-of-the-art NDP accelerator, respectively.

2 BACKGROUND

Before talking about *k*-mer counting, this section first introduces data structure used in *k*-mer counting, i.e., Bloom filter and Counting Bloom filter. Then, description of *k*-mer counting is presented. Finally, buffered DIMM, which is the base of NEST, is introduced.

Bloom Filter: Bloom filter is a space efficient data structure based on hash table and it supports efficient membership checking [2, 19]. In *k*-mer counting, Bloom filter is used to determine whether a *k*-mer is unique or not, i.e., if a *k*-mer appears more than once in the dataset or not. Bloom filter consists of a bit array with the capacity of m and involves n independent hash functions. The bit array is

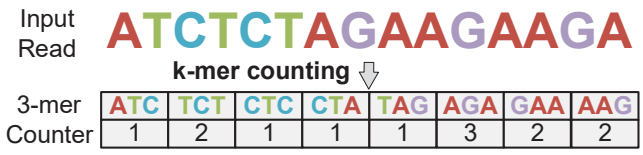

Figure 2: An example of *k*-mer counting.

initialized with zeros. To insert an item into the Bloom filter, n independent hash values are computed and the corresponding entries in the bit array are written to ones. To check the existence of an item, n independent hash values are computed and the corresponding entries are checked to see if they are all ones. If some of the Bloom filter entries are zeros, this item is not in the Bloom filter for sure. Otherwise, if all entries in the Bloom filter are ones, this item is supposed to be in the Bloom filter with a low rate of false positive.

Counting Bloom Filter: Instead of storing a bit array, a counting Bloom filter [11] contains an array with small counters. For example, with a 4-bit counter array, counting Bloom filter is able to handle counts from 0 to 15. Similar to Bloom filter, to insert an item into the counting Bloom filter, n independent hash values are computed and the corresponding entries in the counter array are increased by one. To lookup the counter of an item, n independent hash values are computed and the corresponding entries are read out. The smallest hash value is assumed to be the counter of the target item.

k-mer Counting: As we've mentioned previously, the frequencies of different *k*-mers are needed for many applications in genomics analysis. *k*-mer counting is an essential and time consuming step for deriving the frequencies of *k*-mers in the sequencing reads. To be specific, *k*-mer counting refers to the process of counting the occurrences of DNA substrings with length of k among the sequencing data. An example of *k*-mer counting is shown in Fig 2, the frequencies of different 3-mers are derived after *k*-mer counting.

In the sequencing data, *k*-mers can be divided into two categories, i.e., unique *k*-mers and non-unique *k*-mers. Unique *k*-mers refer to *k*-mers that appear only once in the dataset. Non-unique *k*-mers refer to *k*-mers that appear more than once in the dataset. Because the unique *k*-mers are highly likely to be sequencing errors [25, 31] and, for some sequencing data, up to 75% *k*-mers can be unique [25], *k*-mer counting often removes those unique *k*-mers and only counts the frequencies of non-unique *k*-mers [4, 25]. *k*-mer counting usually includes the following two steps [4, 25]:

- Prune: With a chain of two Bloom filters, each time a *k*-mer comes in, check existence of this *k*-mer in the first Bloom filter. If this *k*-mer is in the first Bloom filter, write this *k*-mer into the second Bloom filter. Otherwise, write this *k*-mer into the first Bloom filter. After all *k*-mers have gone through this process, the non-unique *k*-mers are stored in the second Bloom filter and the unique *k*-mers are filtered out. The first Bloom filter can be discarded after this step.
- Count: For each input *k*-mer, check existence of this *k*-mer in the second Bloom filter. If this *k*-mer is in second Bloom filter, increase the corresponding frequency counter in the Hash table. After all *k*-mers have gone through this process, the occurrences of the non-unique *k*-mers are stored in the Hash table.

Buffered Dual-Inline Memory Module: Dual-Inline Memory Module (DIMM) is a widely used memory package with 64 data

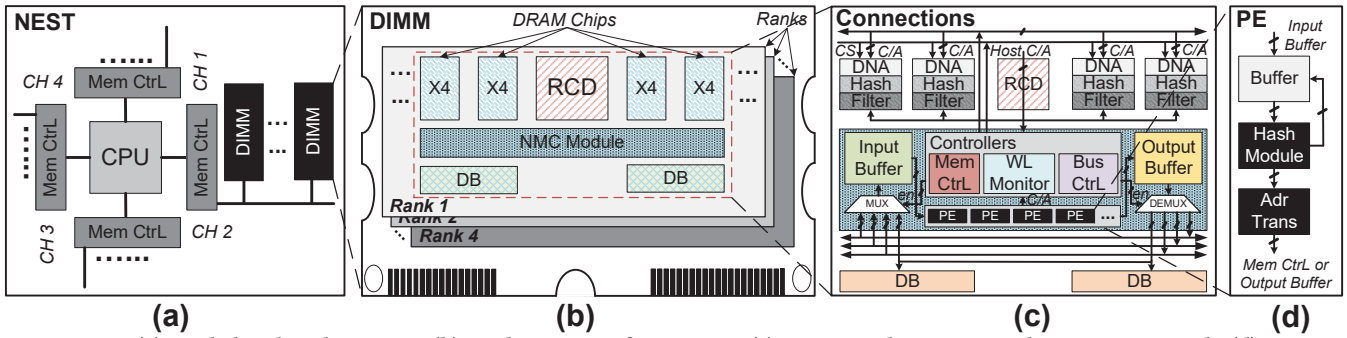

Figure 3: (a) High-level architecture. (b) Architecture of LRDIMM. (c) Micro-architecture within a DRAM rank. (d) PEs.

(DQ) pins, excluding the ones for ECC. Within a DIMM, multiple DRAM chips form a rank. One or more ranks are packaged together to form a DIMM. Load-Reduced DIMM (LRDIMM), as shown in Fig. 3 (b), is introduced to address the signal integrity issue for high frequency and heavy load 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 ARCHITECTURE

NEST is built by modifying the LRDIMM, as shown in Fig. 3 (a) and (b). NEST is scalable and communication between different LRDIMMs in NEST is achieved via the standard DDR channel with the help of the host. We add a Near-Memory Computing (NMC) module to each rank within each LRDIMM to perform k-mer counting. The NMC module is described below:

NMC Module: Different from MEDAL, which modifies the DBs in LRDIMM and inserts customized computing logics into them, we attach a NMC module to each rank in the LRDIMM, as shown in Fig. 3 (b) and (c). The controllers and computing logics in NEST are centralized inside the NMC module. Compared with the approach of distributing logics in MEDAL, centralization of the logics provides better communication and synchronization. Further, centralized logics enable task scheduling and improves the ability of memory access management, which are introduced in Section 5.

To enhance intra-DIMM communication and reduce inter-DIMM communication, the fully hierarchical buses are added.

Fully Hierarchical Buses: Communication becomes a serious challenge, if MEDAL is used to perform k-mer counting. Details about the communication overhead in MEDAL are described in Section 7.6. To address the issue of communication, we design fully hierarchical buses for NEST to better support intra-DIMM communication. Besides the inter-chip buses in MEDAL, the following two types of inter-rank buses are added to NEST:

- *rank-rank* C/A bus: Transfer C/A signals between different ranks within the same DIMM.
- *rank-rank* data bus: Transfer data between different ranks within the same DIMM.

With the inter-rank buses, intra-DIMM communication can be achieved locally without going through the memory channel, which becomes the communication bottleneck in previous work.

To support the functionalities of computation, communication, task schedule, memory access management and so on, the following six components are added inside the NMC module:

Processing Elements (PEs): As shown in Fig. 3 (c) and (d), there are a few PEs inside each NMC module. The number of PEs is configurable. The PEs read/write the input/output data from/to the Input/Output Buffer in the NMC module. The major function of the PE is to perform hash function. About the hash function, MurmurHash3 is used in NEST [28]. Each PE contains:

- Buffer to store the input k-mers
- Lightweight logics to perform hash function
- An address translation engine to convert the virtual address to DRAM device address. Details of the address mapping are described in Section 5.2.

Data Direct Multiplexer: As shown in Fig. 3 (c), NEST connects a multiplexer with the Input Buffer and a multiplexer with the Output Buffer. With the help of those two multiplexers, in addition to receiving/sending data to the DDR bus, the buffers can receive/send data to proposed fully hierarchical buses. The multiplexers are controlled by a dedicated enable signals from the Memory Controller (MC) we add inside the NMC module.

Memory Controller (MC): In order to coordinate memory accesses from both the host and PEs, we add a MC into the NMC module, as shown in Fig. 3 (c). The coordination between the host-side MC and the MC within the NMC module is achieved with the 'Host-prioritized Request Scheduling' proposed in MEDAL. Compare with MEDAL, putting the MC and other logics together provides better communication/synchronization, enables task scheduling, and improves the ability of memory access management.

Workload Monitor: To address the challenge of workload balance and improve PE utilization, we add a Workload Monitor inside the NMC module. The Workload Monitor monitors and cooperates with the Input Buffer and the PEs to tackle the challenge of workload balance in performing k-mer counting. Details of addressing the challenge of workload balance are described in Section 5.3.

Bus Arbiter: The bus arbiter regulates the data and C/A transfer. It takes charge of the assignment of the fully hierarchical buses and assigns them to the PEs sharing those buses.

Input Buffer and Output Buffer: The Input Buffer stores the states and information of the input tasks, i.e., k-mer and the corresponding task progress. The Output Buffer stores the output of the PEs, i.e., information of the memory access.

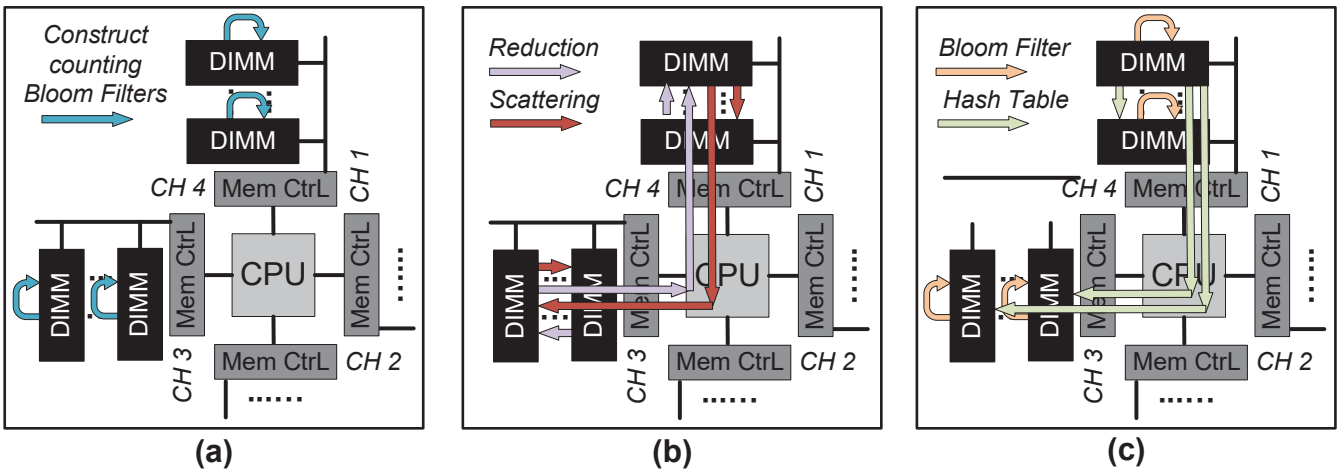

Figure 4: (a) Different DIMMs construct the counting Bloom filters in parallel. Memory accesses are localized. (b) Reduction and scattering of the counting Bloom filters. (c) Count k-mers. The merged Bloom filters are localized inside each DIMM. The hash table is distributed among DIMMs. Only verified non-unique k-mers involve memory access to the hash table.

From the hardware perspective, compared with MEDAL, NEST provides better communication/synchronization, enables task scheduling, improves the ability of memory access management, enhances intra-DIMM communication, and has the potential to tackle the challenge of workload balance.

4 ALGORITHM AND WORKFLOW

In NEST, to enable parallel processing of k-mer counting, the dataset is evenly partitioned into different DIMMs. As we've described in section 2, during k-mer counting, the *prune* phase constructs a chain of two Bloom filters and the *count* phase accesses the second Bloom filter constructed in the *prune* phase to perform k-mer counting. Unfortunately, there are two major drawbacks, if we naively implement the k-mer counting algorithm in NEST architecture:

- Limited Bandwidth and Parallelism: PEs in different DIMMs need to access the same data region of Bloom filters, which underutilizes available memory bandwidth and hinders the parallelism between different PEs.
- Frequent Inter-DIMM Communication: Frequent memory accesses to the same data region of Bloom filters introduces frequent inter-DIMM memory accesses, which brings significant performance overhead. Details of the performance penalty are described in Section 7.6.

To address above issues, our key idea is to provide local Bloom filters for each DIMM to access independently. With localized and independent Bloom filters, PEs in different DIMMs access different memory region for Bloom filter entries, available memory bandwidth and PE parallelism is fully leveraged. Furthermore, inter-DIMM communication is greatly reduced.

However, naively assign different copies of Bloom filters to different DIMMs and make PEs in different DIMMs work in parallel independently do not work. In the original k-mer counting algorithm, the Bloom filters contain global information about the entire dataset, and there will be error if the local Bloom filters are constructed independently. For example, assume a 3-mer *ATC* appears four times in a dataset, those four *ATC* are evenly partitioned into four DIMMs and the local Bloom filters are constructed independently. After the *prune* step, if we check the uniqueness of *ACT* in the local Bloom filters, *ACT* will be confirmed as a unique k-mer in all four local Bloom filters, because *ATC* only appear once in each DIMM. However, *ATC* is a non-unique 3-mer globally, it appears fourd times in the entire dataset.

To address the challenges above, we leverage the counting Bloom filter and modify the k-mer counting algorithm. There are three steps in the modified k-mer counting algorithm:

1. Construct the Counting Bloom Filters: During the construction of the Counting Bloom filters, compared with the original k-mer counting algorithm, instead of using two 1-bit Bloom filters, we leverage one 2-bit counting Bloom filter. Each DIMM constructs their local counting Bloom filter, recording how many times (0, 1 or 2) each k-mer appears in this sub-dataset.

2. Merge the Counting Bloom Filters: After different DIMMs finish constructing local counting Bloom filters, NEST merges those local counting Bloom filters to a merged Bloom filter via reduction and scattering. Reduction of the counting Bloom filter is performed by adding the corresponding entries in those counting Bloom filters. In the end of reduction, if a counter entry is larger than 2 in the reduced counting Bloom filter, the corresponding entry in the merged Bloom filter will be one, otherwise it will be zero. Scattering of the merged Bloom filter is performed by distributing the merged Bloom filter to each DIMM.

3. Count k-mers: After scattering of the merged Bloom filter, each DIMM contains a copy of the merged Bloom filter. k-mer counting is performed in different DIMMs in parallel. For each k-mer, NEST first checks the merged Bloom filter locally to see if this k-mer is non-unique. If current k-mer is non-unique, Memory access to the distributed hash table will be performed and NEST will increase the corresponding frequency counter in the hash table by one.

The workflow of performing proposed algorithm in NEST is shown in Fig. 4. Construction of the counting Bloom filters doesn't involve inter-DIMM communication, which will greatly degrade performance of the system. Merge of the counting Bloom filter only

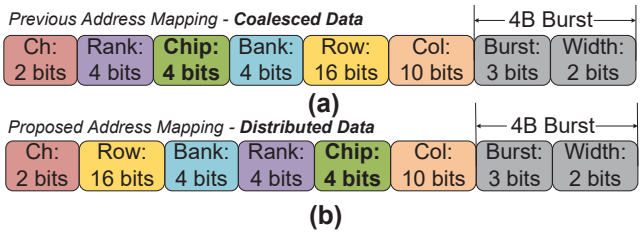

Figure 5: (a) Previous address mapping scheme aggregates fine-grained data to better leverage locality. (b) Proposed address mapping scheme distributes fine-grained data for better memory bandwidth utilization.

involves continuously sequential read and write operations, which have little impact on performance. About the step of counting k-mer, unnecessary inter-DIMM memory accesses are avoided via first checking the merged Bloom filter locally, which contributes to the good performance of NEST.

5 CHALLENGES AND OPTIMIZATIONS

The challenges of performing k-mer counting in commodity DRAM components based NDP accelerator together with corresponding techniques proposed to address those challenges are presented in this section.

5.1 Bottleneck of Communication

Inter-DIMM communication becomes the bottleneck, if MEDAL is used for k-mer counting (details in Section 7.6). In order to ensure no hardware modification is made to the host-side memory controller and maintain the DDR timing constraint, worse case timing scenario is considered for inter-DIMM memory access, which means there is an extra delay for each inter-DIMM memory access. Because of this, inter-DIMM memory access involves significant performance penalties in DIMM based NDP architecture.

To address above challenge, the following hardware and software optimizations are used in NEST to reduce the number of inter-DIMM memory access and relieve the bottleneck of communication:

NEST Workflow: Proposed workflow greatly reduces unnecessary inter-DIMM memory access by dividing k-mer counting into multiple steps and localizing data within each DIMM in each step as much as possible.

Fully Hierarchical Buses: We add inter-rank buses, including the *rank-rank* C/A bus and *rank-rank* data bus, to enable efficient communication between different ranks within a DIMM, minimizing the amount of inter-DIMM communication.

5.2 Bandwidth Utilization

For address mapping in commodity DRAM components based NDP architecture, as shown in Fig 5 (a), MEDAL coalesces data within a DRAM chip to better leverage data locality. However, in proposed k-mer counting, there are lots of memory accesses to counting Bloom filter entries or Bloom filter entries, i.e., 1-bit or 2-bit random memory access, which means there is no locality at all. Thus, as shown in Fig 5 (b), compared with MEDAL, we re-order the address bits to prioritize distributing data in different DRAM chips. With proposed address mapping, instead of trying to coalesce data within the same DRAM chip, we try to distribute data in different DRAM chips to improve the memory bandwidth utilization.

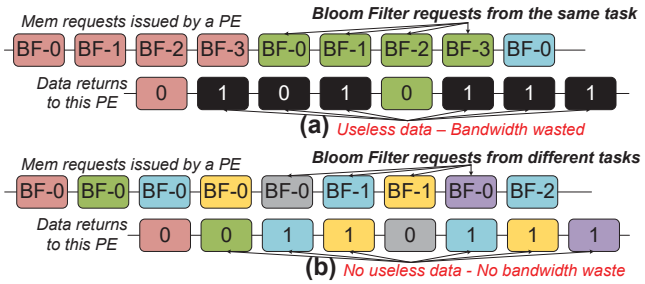

Figure 6: (a) Memory bandwidth is wasted if memory requests from the same task are issued sequentially. (b) No memory bandwidth is wasted with proposed optimizations.

5.3 Workload Balance

The key idea of addressing the challenge of workload balance is to keep an eye on the states of different PEs and perform task scheduling correspondingly. As mentioned before, we add a Workload Monitor in the NMC module. The Workload Monitor tries to keep all PEs busy and it's in charge of the task scheduling. Tasks come from the DRAM will be put into the Input Buffer first. The Workload Monitor monitors the states of different PEs and the Input Buffer. If a PE needs more tasks to process and there are pending tasks in the Input Buffer, the Workload Monitor will dispatch tasks to this PE to keep it busy. The challenge of workload balance is addressed with proposed task scheduling via dispatching tasks to PEs in fine-granularity dynamically.

5.4 Redundant Memory Access

During the step 'Count k-mer', we need to verify if all Bloom filter entries related to current k-mer in the merged Bloom filter are ones. If all of those Bloom filter entries are ones, we need to write to the hash table. Otherwise, no write operation is needed. However, if memory accesses to the merged Bloom filter are issued sequentially, memory bandwidth may be wasted. For example, assume for each k-mer, four Bloom filter entries need to be checked. As shown in Fig 6 (a), four memory accesses belong to the same k-mer are issued sequentially. However, value of the first Bloom filter entry returned is zero, meaning that no write operation is needed and we don't need to check other Bloom filter entries at all. However, because the memory accesses are issued sequentially, useless Bloom filter entries will be fetched out from the DRAM and memory bandwidth is wasted. To address this issue, we propose the two-step optimization to eliminate redundant memory accesses:

Scattered Memory Access: As shown in Fig 6 (b), instead of issuing memory accesses belong to a k-mer sequentially, we scatter those memory accesses and issue them with time intervals. We will issue another memory access, only if the previous memory access related to a k-mer has returned and the returned value is one. With this approach, the redundant memory accesses are eliminated and the available memory bandwidth can be utilized efficiently.

Task Switching: Although redundant memory access is eliminated with scattered memory access, memory bandwidth is still being wasted due to the lack of enough memory access to DRAM between the memory access intervals. To solve this issue, we propose to switch tasks between memory accesses. PEs will switch to another task and issue a memory access belong to another k-mer after

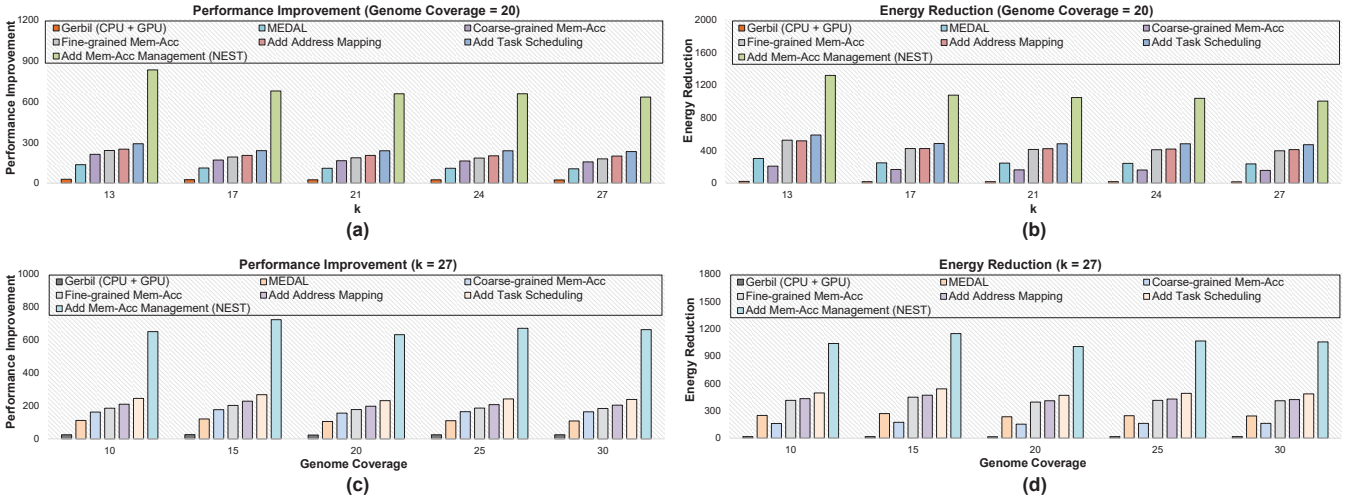

Figure 7: Performance improvement and energy reduction of CPU/GPU hybrid approach, MEDAL, and NEST with different architectures/optimizations. Results are normalized to that of a 48-thread CPU. (a) Performance improvement for different k with 20x genome coverage. (b) Energy reduction for different k with 20x genome coverage. (c) Performance improvement for different genome coverage ratios with $k = 27$. (d) Energy reduction for different genome coverage ratios with $k = 27$.

Table 1: Configure of the baselines and NEST

Configuration of the CPU basline	
CPU Model	Intel Xeon E5-2680 v3
CPU Clock Frequency (GHz)	2.50
Memory Capacity (GB)	384
L1 (KB)/L2 (KB)/L3 (MB) Cache	64 / 256 / 32
Configuration of the CPU + GPU basline	
GPU Model	Nvidia Titan X
CPU Model	Intel Xeon E5-2603 v3
CPU Clock Frequency (GHz)	1.60
Memory Capacity (GB)	24
L1 (KB)/L2 (KB)/L3 (MB) Cache	64 / 256 / 16
Configuration of MEDAL and NEST	
Memory Capacity (GB)	512
Memory Channels	4
DIMMs per Memory Channels	2
Ranks per DIMM	4
DRAM Chips per Rank	16
Rank-Rank C/A buses per DIMM (NEST)	4
Rank-Rank Data buses per DIMM (NEST)	1
Chip-Chip Data buses per Rank	1
PEs per Rank	6
Parameters of DDR4 DRAM	
Capacity	8Gb × 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 the Lightweight logics

Module	Latency (Cycles)	Power (mW)	Leakage (uW)	Area (um^2)
Hash Module	17	5.99	8.38	5297.58
Addr Trans	4	2.13	16.45	11423.54

issuing previous memory access belong to a certain k-mer. With this approach, time intervals due to scattered memory accesses will be filled with memory accesses belong to different k-mers.

Combine above two techniques, the redundant memory accesses are eliminated and memory bandwidth can be utilized efficiently.

6 DISCUSSION

Algorithm Equivalence: Proposed algorithm leverages counting Bloom filter to perform k-mer counting. In counting Bloom filter, the counter returned may be higher than the actual frequency of the k-mer in the dataset. This is not a problem for two reasons. First, higher counter value in the counting Bloom filter is equivalent to the false positive rate of the Bloom filter in the original k-mer counting algorithm and is generally considered insignificant [24]. Second, in general, retaining k-mers with low occurrences doesn't degrade the final results of the following applications [25].

Generality of NEST: From hardware perspective, NEST provides a practical, scalable, high-performance, and energy efficient NDP accelerator with hierarchical communication schemes and support for fine-grained random memory access, it can be beneficial to memory-bound applications require hierarchical communication and fine-grained memory access. For example, NEST can be easily configured to support the application of 'DNA seeding' which MEDAL is designed for simply by replacing the PEs inside NEST with customized PEs for 'DNA Seeding'. From software perspective, proposed algorithm and workflow provides a solution to reduce memory access in distributed NDP architectures with software/hardware co-design and the divide-and-conquer approach.

7 EXPERIMENTAL RESULTS

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

7.1 Experimental Setup

Configuration of the Baselines: For CPU and CPU + GPU, we use two widely used software tools, i.e., BFCounter [25] and Gerbil [10], as the baselines. The detailed configuration information of the two servers running those two baselines is shown in Table 1.

For MEDAL, as shown in Table 1, the configuration of memory and number of PEs are the same as those in NEST. The differences between MEDAL and NEST are those architecture modifications and communication optimizations we make. Proposed k-mer counting algorithm is also used in MEDAL to improve its performance,

Configuration of NEST: We modify Ramulator [18] to build a cycle-accurate simulator for NEST. The configuration of NEST is

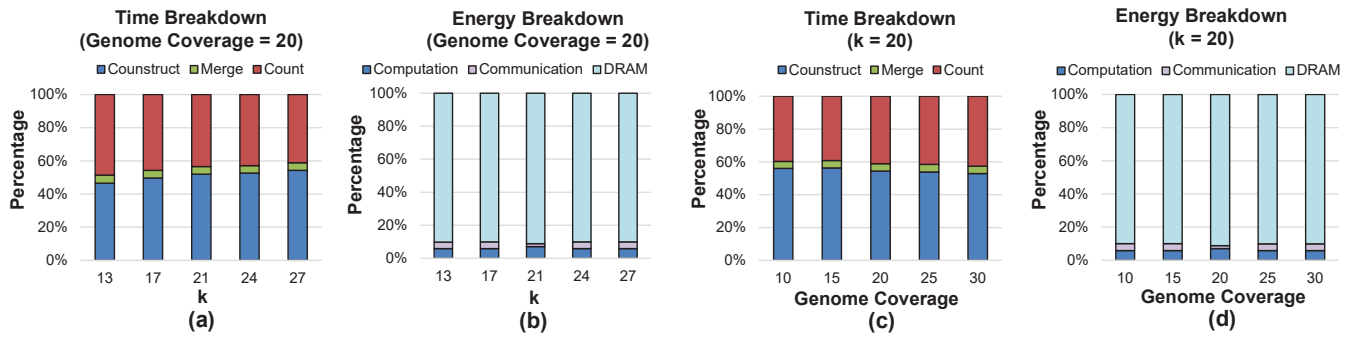

Figure 8: (a) Time breakdown for different steps with 20x genome coverage. (b) Energy breakdown for computation, communication, and memory access with 20x genome coverage. (c) Time breakdown for different steps with $k = 20$. (d) Energy breakdown for computation, communication, and memory access with $k = 20$.

shown in Table 1. We use pre-layout Design Compiler [33] with 28 nm technology [1] to estimate the timing, energy, and area parameters of the PEs. The timing constraint is set to be 1.2GHz. The parameters of the PEs are shown in Table 2. The timing parameters of the DRAM components are shown in Table 1. The energy consumption of DRAM is derived by feeding the memory traces from Ramulator to DRAMPower [5]. The parameters of energy consumption for the datapath used in this paper are from CACTI-IO [16].

Datasets: The datasets used in the experiments are sequenced human genome [27] with different coverage ratio.

7.2 Performance Improvement

The performance of different architectures/optimizations are in Fig. 7 (a) and (c). All the data are normalized to the performance of the 48-thread CPU baseline.

As shown in Fig. 7 (a), when the genome coverage is 20x, for different k, naive coarse-grained memory access in NEST architecture improves the performance of the 48-thread CPU, the CPU/GPU hybrid approach, and MEDAL by 171.78x, 6.88x, and 1.51x, respectively. Compared with naive coarse-grained memory access, naive fine-grained memory boosts the performance of NEST by 1.13x. About the optimizations, proposed address mapping improves the performance of naive fine-grained memory access by 1.08x. Moreover, performance improvement of 1.17x is achieved with proposed task scheduling. Memory access management gains 2.79x performance improvement. Combine above optimizations together, NEST outperforms the 48-thread CPU, the CPU/GPU hybrid approach, and MEDAL by 687.48x, 27.53x, and 6.06x, respectively. When k is 27, similar trend of speedup can be observed in Fig. 7 (c).

Compared with MEDAL, performance improvement from the configuration of naive coarse-grained memory access in NEST comes from the enhanced support for intra-DIMM communication. Compared with naive coarse-grained memory access, performance improvement of naive fine-grained memory access comes from its ability to perform fine-grained memory access. Further, performance improvement of task scheduling comes from the balanced workloads in different PEs. Finally, performance improvement of memory access management comes from its reduction of redundant memory access and task switching to efficiently utilize available memory bandwidth. Overall, compared with the CPU baseline, chip-level fine-grained memory access provides 16x more bandwidth, rank-level parallelism provides 4x more bandwidth, and allowing different copies of counting Bloom filters/Bloom filters in different DIMMs to be accessed in parallel provides 8x more bandwidth. Combine those benefits together, NEST provides 512x more memory bandwidth than the CPU baseline. Further, proposed k-mer counting algorithm reduces the amount of memory access needed, i.e. one counting Bloom filter vs. two Bloom filters. Moreover, NEST has efficient task scheduling and memory access management. Combine all above advantages, NEST provides significant performance improvement, compared with the CPU baseline.

7.3 Energy Reduction

The energy reduction of different architectures/optimizations are in Fig. 7 (b) and (d). All the data are normalized to the energy consumption of the 48-thread CPU baseline.

As shown in Fig. 7 (b), when the genome coverage is 20x, for different k, naive coarse-grained memory access in NEST architecture reduces energy consumption of the 48-thread CPU and the CPU/GPU hybrid approach by 160.16x and 9.75x, respectively. Compared with MEDAL, this approach consumes 50% more energy. Compared with naive coarse-grained memory access, energy consumption is reduced by 2.54x with naive fine-grained memory reduces. About proposed optimizations, address mapping slightly reduces the energy consumption by 1.01x. Task scheduling reduces energy consumption by 1.15x. Energy reduction of 2.19x is achieved via memory access management. Combine proposed optimizations together, compared with the 48-thread CPU, the CPU/GPU hybrid approach, and MEDAL, NEST reduces the energy consumption by 1091.91x, 62.90x, and 4.32x, respectively. When k is 27, similar trend of energy reduction can be found in Fig. 7 (d).

7.4 Time and Energy Breakdown

The time breakdown for NEST is shown in Fig. 8 (a) and (c). The results indicate that the phases of 'Merge the Counting Bloom Filters' are negligible, because this phase only takes less than 5% of the total runtime. The dominant phases in the workflow are 'Construct Counting Bloom Filters' and 'Count k-mers'. By introducing the negligible phases of 'Merge the Counting Bloom Filters', proposed workflow separates the phases of 'Construct Counting Bloom Filters' and 'Count k-mers' to reduce inter-DIMM communication, leading to performance improvement.

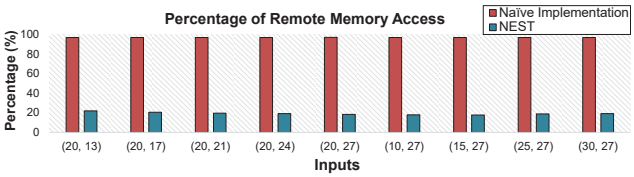

Figure 9: Percentage of remote memory access in the naive implementation of the original k-mer counting algorithm and in NEST.

The energy breakdown of NEST is shown in Fig. 8 (b) and (d). More than 90% energy is consumed by DRAM. Less than 10% energy is consumed by computation and communication combined together. The observations indicate that computation and communication in NEST is very energy efficient.

7.5 Remote Memory Access

The percent of remote memory access among all memory access with naive implementation of the original k-mer counting algorithm in MEDAL architecture and the percent of remote memory access in NEST are shown in Fig. 9. The x-axis standards for different inputs in the form of (Genome Coverage, k). Compared with the naive implementation, NEST effectively reduces the percent of remote memory access from 96.90% to 19.20% on average.

7.6 PE Utilization

The breakdown of different PE states for the phases of 'Construct Counting Bloom filters' and 'Count k-mers' are shown in Fig. 10 (a) and (b). The results indicate that, for k-mer counting, communication becomes the bottleneck in MEDAL due to its frequent inter-DIMM communication with extra performance penalty. Proposed step-by-step optimizations tackles the challenge in communication and memory. For the phase of 'Construct Counting Bloom filters', compared with MEDAL, NEST architecture increases the PE utilization ratio from 12.39% to 20.13%. Combine proposed techniques together, PE utilization is improved to 56.62%. For the phase of 'Count k-mers', similar trend can be observed. Compared with MEDAL, NEST has a much higher PE utilization ratio.

8 RELATED WORK

This section introduces related work of NEST.

Accelerators for K-mer Counting: Most previous accelerators for k-mer counting are based on multi-core [6, 9], FPGA [3, 4], and GPU [4, 10]. Although above computing platforms have enough computation capability, they are not suitable for k-mer counting. As we have discussed in previous sections, the space for performance improvement is limited for above compute-centric architectures, because k-mer counting is a memory-bound application and those approaches mostly focus on computation part.

Compared with those compute-centric approaches, NEST is a memory-centric accelerator focusing on optimization of the memory, which fits k-mer counting better and can have more significant performance improvement

NDP solutions for K-mer Counting: Hybrid Memory Cube (HMC) has been leveraged to accelerate k-mer counting in a few works due to its high memory bandwidth [15, 24]. However, compared with commodity DRAM, 3D-integration is not cost-efficient, has limited

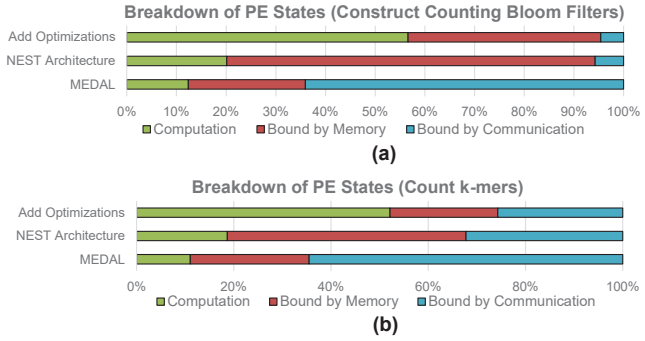

Figure 10: Breakdown of the PEs states. (a) Constructing counting Bloom Filters. (b) Counting k-mers.

capacity [30], and doesn't provide extra internal bandwidth [8]. MEDAL is a NDP accelerator based on commodity DRAM components with the ability of performing fine-grained memory access [14], which seems to be suitable for k-mer counting. Unfortunately, according to the evaluation, when MEDAL is used to perform k-mer counting, inter-DIMM communication becomes the bottleneck, which greatly degrades performance of the system. Moreover, other serious issues, such as workload imbalance and redundant memory access, also have significantly negative impact on its performance.

Compared with above NDP accelerators, NEST is cost-efficient, scalable, and communication-optimized. In addition, NEST addresses all the challenges mentioned above, i.e., workload imbalance and redundant memory access, in k-mer counting. Moreover, NEST can be easily extended to support many applications require hierarchical communication and fine-grained memory access.

NDP vs. PIM: The reason that we don't choose PIM architecture for k-mer counting is mainly due to the consideration of practicality. To be specific, although PIM architectures leverage emerging technology often have very good performance [13, 17], those emerging technologies are usually not mature enough and difficult be put into real use. For PIM architectures based on conventional DRAM [20, 21], they usually require modifications to the DRAM die, which are difficult for real implementation as well.

9 CONCLUSION

This paper proposes NEST, a practical, scalable, high-performance, and energy efficient NDP accelerator for k-mer counting. To fully unleash the performance of NEST, we modify the k-mer counting algorithm and propose a dedicated workflow to support efficient parallelism. Proposed algorithm and workflow are able to reduce unnecessary inter-DIMM communication. In addition, we propose a novel address mapping scheme to improve memory bandwidth utilization. The challenge of workload balance is addressed with proposed task scheduling. Scattered memory access and task-switching are proposed to eliminate redundant memory access. Experimental results show that NEST provides 677.33x/27.24x/6.02x performance improvement and 1076.14x/62.26x/4.30x energy reduction, compared with a 48-thread CPU, a CPU/GPU hybrid approach, and a state-of-the-art NDP accelerator, respectively.

ACKNOWLEDGMENTS

This work was supported in part by NSF 1533933, 1719160, and 1730309.

REFERENCES

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

[2] Burton H Bloom. 1970. Space/time trade-offs in hash coding with allowable errors. *Commun. ACM* 13, 7 (1970), 422–426.

[3] Simmi M Bose, Varsha S Lalapura, S Saravanan, and Madhura Purnaprajna. 2019. k-core: Hardware Accelerator for k-mer Generation and Counting used in Computational Genomics. In *2019 32nd International Conference on VLSI Design and 2019 18th International Conference on Embedded Systems (VLSID)*. IEEE, 347–352.

[4] Nicola Cadenelli, Zoran Jaksić, Jordà Polo, and David Carrera. 2019. Considerations in using OpenCL on GPUs and FPGAs for throughput-oriented genomics workloads. *Future Generation Computer Systems* 94 (2019), 148–159.

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

[6] Rayan Chikhi and Guillaume Rizk. 2013. Space-efficient and exact de Bruijn graph representation based on a Bloom filter. *Algorithms for Molecular Biology* 8, 1 (2013), 22.

[7] 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.

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

[9] Sebastian Deorowicz, Marek Kokot, Szymon Grabowski, and Agnieszka Debudaj-Grabysz. 2015. KMC 2: fast and resource-frugal k-mer counting. *Bioinformatics* 31, 10 (2015), 1569–1576.

[10] Marius Erbert, Steffen Rechner, and Matthias Müller-Hannemann. 2017. Gerbil: a fast and memory-efficient k-mer counter with GPU-support. *Algorithms for Molecular Biology* 12, 1 (2017), 9.

[11] Li Fan, Pei Cao, Jussara Almeida, and Andrei Z Broder. 2000. Summary cache: a scalable wide-area web cache sharing protocol. *IEEE/ACM transactions on networking* 8, 3 (2000), 281–293.

[12] Wei-jie Guan, Zheng-yi Ni, Yu Hu, Wen-hua Liang, Chun-quan Ou, Jian-xing He, Lei Liu, Hong Shan, Chun-liang Lei, David SC Hui, et al. 2020. Clinical Characteristics of Coronavirus Disease 2019 in China. *New England Journal of Medicine* (2020).

[13] 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.

[14] Wenqin Huangfu, Xueqi Li, Shuangchen Li, Xing Hu, Peng Gu, and Yuan Xie. 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*. 587–599.

[15] Biresh Kumar Joardar, Priyanka Ghosh, Partha Pratim Pande, Ananth Kalyanaraman, and Sriram Krishnamoorthy. 2019. NoC-enabled software/hardware co-design framework for accelerating k-mer counting. In *Proceedings of the 13th IEEE/ACM International Symposium on Networks-on-Chip*. 1–8.

[16] 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.

[17] 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).

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

[19] Adam Kirsch and Michael Mitzenmacher. 2006. Less hashing, same performance: building a better bloom filter. In *European Symposium on Algorithms*. Springer, 456–467.

[20] Marzieh Lenjani, Patricia Gonzalez, Elaheh Sadredini, Shuangchen Li, Yuan Xie, Ameen Akel, Sean Eilert, Mircea R Stan, and Kevin Skadron. 2020. Fulcrum: a simplified control and access mechanism toward flexible and practical in-situ accelerators. In *2020 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 556–569.

[21] Shuangchen Li, Dimin Niu, Krishna T Malladi, Hongzhong Zheng, Bob Brennan, and Yuan Xie. 2017. Drisa: A dram-based reconfigurable in-situ accelerator. In *2017 50th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 288–301.

[22] Roujian Lu, Xiang Zhao, Juan Li, Peihua Niu, Bo Yang, Honglong Wu, Wenling Wang, Hao Song, Baoying Huang, Na Zhu, et al. 2020. Genomic characterisation and epidemiology of 2019 novel coronavirus: implications for virus origins and receptor binding. *The Lancet* (2020).

[23] Xingliang Ma, Martin Mau, and Timothy F Sharbel. 2018. Genome editing for global food security. *Trends in biotechnology* 36, 2 (2018), 123–127.

[24] Nathaniel Mcvicar, Chih-Ching Lin, and Scott Hauck. 2017. K-mer counting using Bloom filters with an FPGA-attached HMC. In *2017 IEEE 25th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*. IEEE, 203–210.

[25] Pall Melsted and Jonathan K Pritchard. 2011. Efficient counting of k-mers in DNA sequences using a bloom filter. *BMC bioinformatics* 12, 1 (2011), 333.

[26] Jason D Merker, Aaron M Wenger, Tam Sneddon, Megan Grove, Zachary Zappala, Laure Fresard, Daryl Waggott, Sowmi Utiramerur, Yanli Hou, Kevin S Smith, et al. 2018. Long-read genome sequencing identifies causal structural variation in a Mendelian disease. *Genetics in Medicine* 20, 1 (2018), 159–163.

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

[28] Tony C Pan, Sanchit Misra, and Srinivas Aluru. 2018. Optimizing high performance distributed memory parallel hash tables for DNA k-mer counting. In *SC18: International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 135–147.

[29] Carl Poirier, Benoit Gosselin, and Paul Fortier. 2015. DNA assembly with de bruijn graphs on FPGA. In *2015 37th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)*. IEEE, 6489–6492.

[30] 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.

[31] Anand Ramachandran, Yun Heo, Wen-mei Hwu, Jian Ma, and Deming Chen. 2015. FPGA accelerated DNA error correction. In *2015 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 1371–1376.

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

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

[34] 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.