University of California
Santa Barbara

Memory-Centric Accelerators for Genome Analysis

A dissertation submitted in partial satisfaction
of the requirements for the degree

Doctor of Philosophy
in
Electrical and Computer Engineering

by

Wenqin Huangfu

Committee in charge:

Professor Yuan Xie, Chair
Professor Tim Sherwood
Professor Dmitri Strukov
Professor Li-C Wang
Professor Zheng Zhang

September 2022

The Dissertation of Wenqin Huangfu is approved.

Professor Tim Sherwood

Professor Dmitri Strukov

Professor Li-C Wang

Professor Zheng Zhang

Professor Yuan Xie, Committee Chair

June 2022

Memory-Centric Accelerators for Genome Analysis

by

Wenqin Huangfu

This dissertation is dedicated to my parents, Chaochun Huangfu and Shicui Shen, for bringing me to this world, raising me, and giving me unconditional love throughout my life.

This dissertation is also dedicated to my grandparents, Defeng Huangfu and Zhengfen Wang, for giving me unconditional love, company, and support during my childhood.

Acknowledgements

During my Ph.D. program, I've received generous help from many people. I would like to express my sincere gratitude to everyone who helped me along this journey. It would be impossible for me to complete this dissertation without their help.

First of all, I would like to express my gratitude to my advisor, Prof. Yuan Xie. As a hard-working, passionate, and visionary researcher, he sets a high standard for me to learn and follow. He always encourages me to expand my knowledge domain, explore emerging research directions, and get involved in impactful projects. His precious guidance during my Ph.D. program greatly shapes my research tastes, thinking mode, and working style. It's my pleasure to be able to work with and learn from him. Words are pale in accurately describing the depth my gratitude.

Second, many thanks to the members of my Ph.D. committee, Prof. Tim Sherwood, Prof. Dmitri Strukov, Prof. Li-C. Wang, and Prof. Zheng Zhang, for serving in my dissertation committee and providing valuable feedback to my research and dissertation.

Third, I would like to thank my wonderful collaborators and labmates, including but not limited to Dr. Shuangchen Li, Dr. Xinfeng Xie, Dr. Xing Hu, Dr. Xueqi Li, Dr. Peng Gu, Dr. Fengbin Tu, Dr. Jiayi Huang, Dr. Lixue Xia, Dr. Lei Deng, Dr. Mingyu Yan, Nan Wu, and Bangyan Wang. I would like to especially thank Dr. Shuangchen Li. Dr. Shuangchen Li is my most important research collaborator during my Ph.D. program. He taught me and helped me a lot when I was a junior Ph.D. student.

In addition, I am thankful to my acadmic mentors both at school and in the industry, including but not limited to Prof. Yu Wang, Prof. Yuefei Ding, Dr. Hongzhong Zheng, Dr. Krishna Teja Malladi, Dr. Dimin Niu, and Andrew Chang. I really appreciate their time and efforts in helping me develop the abilities to conduct research and build professional skills for the industry.

Last but not least, I sincerely thank my parents, Chaochun Huangfu and Shicui Shen, for bringing me to this world and encouraging me to pursue my dreams unconditionally. I also sincerely thank my grandparents, Defeng Huangfu and Zhengfen Wang, for giving me unconditional love, company, and support during my childhood.

Curriculum Vitæ

Wenqin Huangfu

Education

2022	Ph.D. in Electrical and Computer Engineering (Expected), University of California, Santa Barbara, United States.
2019	M.S. in Electrical and Computer Engineering, University of California, Santa Barbara, United States.
2016	B.Eng. in Electronic Information Science and Technology, Tsinghua University, Beijing, China.

Publications

[C1] **Wenqin Huangfu**, Krishna T. Malladi, Andrew Chang, Yuan Xie. "BEACON: Scalable Near-Data-Processing Accelerators for Genome Analysis near Memory Pool with the CXL Support." Proceedings of the 55th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO), 2022.

[C2] Hussam Amrouch, Jian-Jia Chen, Kaushik Roy, Yuan Xie, Indranil Chakraborty, **Wenqin Huangfu**, Ling Liang, Fengbin Tu, Cheng Wang, Mikail Yayla. "Brain-Inspired Computing: Adventure from Beyond CMOS Technologies to Beyond von Neumann Architectures." IEEE/ACM International Conference On Computer Aided Design (ICCAD), 2021.

[C3] **Wenqin Huangfu**, Krishna T. Malladi, Shuangchen Li, Peng Gu, Yuan Xie. "NEST: DIMM based Near-Data-Processing Accelerator for K-mer Counting." IEEE/ACM International Conference On Computer Aided Design (ICCAD), 2020.

[C4] **Wenqin Huangfu**, Xueqi Li, Shuangchen Li, Xing Hu, Peng Gu, Yuan Xie. "MEDAL: Scalable DIMM based Near Data Processing Accelerator for DNA Seeding Algorithm." Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO), 2019.

[C5] Dylan Stow, Itir Akgun, **Wenqin Huangfu**, Yuan Xie, Xueqi Li, Gabriel H Loh. "Efficient System Architecture in the Era of Monolithic 3D: Dynamic Inter-tier Interconnect and Processing-In-Memory." 56th ACM/IEEE Design Automation Conference (DAC), 2019.

[C6] **Wenqin Huangfu**, Shuangchen Li, Xing Hu, Yuan Xie. "RADAR: A 3D-ReRAM based DNA Alignment Accelerator Architecture." 55th ACM/IEEE Design Automation Conference (DAC), 2018.

[C7] **Wenqin Huangfu**, Lixue Xia, Ming Cheng, Xiling Yin, Tianqi Tang, Boxun Li, Krishnendu Chakrabarty, Yuan Xie, Yu Wang, Huazhong Yang. "Computation-Oriented Fault-Tolerance Schemes for RRAM Computing Systems." 22nd Asia and South Pacific Design Automation Conference (ASP-DAC), 2017.

[C8] Lixue Xia, Tianqi Tang, **Wenqin Huangfu**, Ming Cheng, Xiling Yin, Boxun Li, Yu Wang, Huazhong Yang. "Switched by Input: Power Efficient Structure for RRAM-based Convolutional Neural Network." 53th ACM/IEEE Design Automation Conference (DAC), 2016.

[J1] Peng Gu, Benjamin S Lim, **Wenqin Huangfu**, Krishan T Malladi, Andrew Chang, Yuan Xie. "NMTSim: Transaction-Command Based Simulator for New Memory Technology Devices." IEEE Computer Architecture Letters, 2020.

[J2] Lixue Xia, **Wenqin Huangfu**, Tianqi Tang, Xiling Yin, Krishnendu Chakrabarty, Yuan Xie, Yu Wang, Huazhong Yang. "Stuck-at Fault Tolerance in RRAM Computing Systems." IEEE Journal on Emerging and Selected Topics in Circuits and Systems, 2017.

[J3] Lixue Xia, Peng Gu, Boxun Li, Tianqi Tang, Xiling Yin, **Wenqin Huangfu**, Shimeng Yu, Yu Cao, Yu Wang, Huazhong Yang. "Technological Exploration of RRAM Crossbar Array for Matrix-Vector Multiplication." Journal of Computer Science and Technology, 2016.

Abstract

Memory-Centric Accelerators for Genome Analysis

by

Wenqin Huangfu

Genome analysis benefits precise medical care, wildlife conservation, pandemic treatment, e.g., COVID-19, and so on. Unfortunately, the speed of data processing in genome analysis lags far behind the speed of data generation and hardware acceleration turns out to be necessary. As many key applications in genome analysis are memory-bound, the computation-centric accelerators, e.g., CPU, GPU, and FPGA, face the challenges of limited memory bandwidth and frequent data movement, which leads to sub-optimal performance and energy efficiency.

To address these challenges, this dissertation focuses on exploring efficient memory-centric accelerators for genome analysis, including designs and optimizations in both hardware and software. This dissertation proposes four memory-centric accelerators for genome analysis, covering both the emerging memory technology, i.e., ReRAM, and the conventional memory technology, i.e., DRAM. By performing in-situ computation inside the emerging memory array to leverage massive parallelism and eliminate data movement, the proposed emerging memory technology based design provides ultra-high performance and energy efficiency. As a comparison, by integrating the processing elements near the conventional memory array to utilize extra memory bandwidth and reduce data movement, the proposed conventional memory technology based designs highlight practicality and cost-effectiveness without making any modifications to the cost-sensitive DRAM dies. Multiple key applications in genome analysis are covered in this dissertation, including k-mer counting, DNA seeding, and DNA pre-alignment.

Contents

Chapter 1

Introduction

Genome analysis is getting more and more attention, due to its important usage in evolutionary studies [1], wildlife conservation [2], disease understanding [3], precise medical care [4], and so on. In reality, genome analysis is closely related to people's health and daily lives. For example, genome analysis is useful in understanding and designing optimal drug cocktail for cancer-causing mutations [5]. In addition, genome analysis also helps a lot in dealing with the Coronavirus Disease 2019 (COVID-19) [6,7], which infects and causes death to millions of people around the world after its outbreak in 2019.

In the past 30 years, with the rapid development of the Next Generation Sequencing (NGS) technology [8], the cost of genome sequencing reduces faster than the Moore's law. According to the National Human Genome Research Institute (NHGRI), as shown in Figure 1.1, the cost of genome sequencing reduces 882065x over the past 20 years, outpacing the Moore's law [9]. Due to the reduced cost of genome sequencing and the large amount of sequencing data required for precise medicine [10], the growth rate of genome data also becomes faster than the Moore's law [11]. According to the National Center for Biotechnology Information (NCBI), as shown in Figure 1.2 (a), the amount

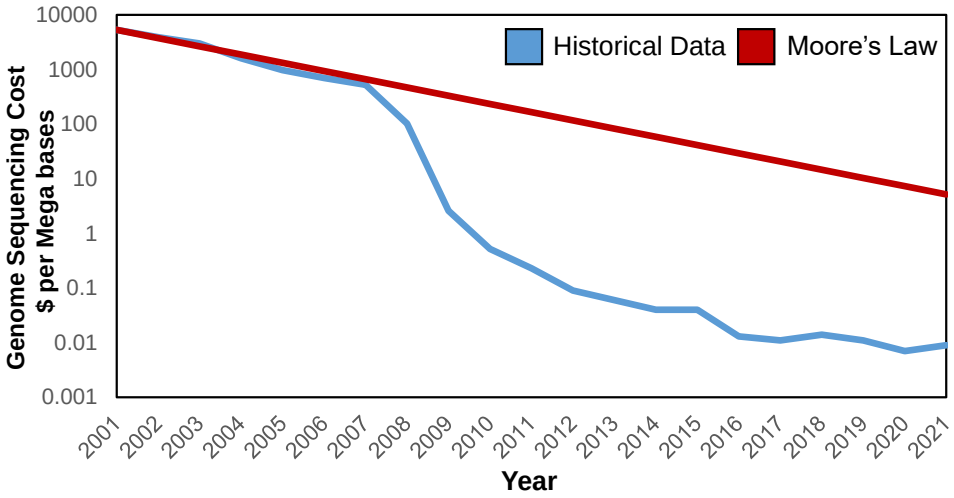

Figure 1.1: Genome sequencing cost, i.e., dollars per mega bases, over the past 20 years.

of genome data in the GenBank database grows 1725589x over the past 30 years, which is faster than the Moore's law [12]. Similarly, as shown in Figure 1.2 (b), the amount of genome data in the Whole Genome Shotgun (WGS) project grows 22286x over the past 20 years, which also outpaces the Moore's law [12].

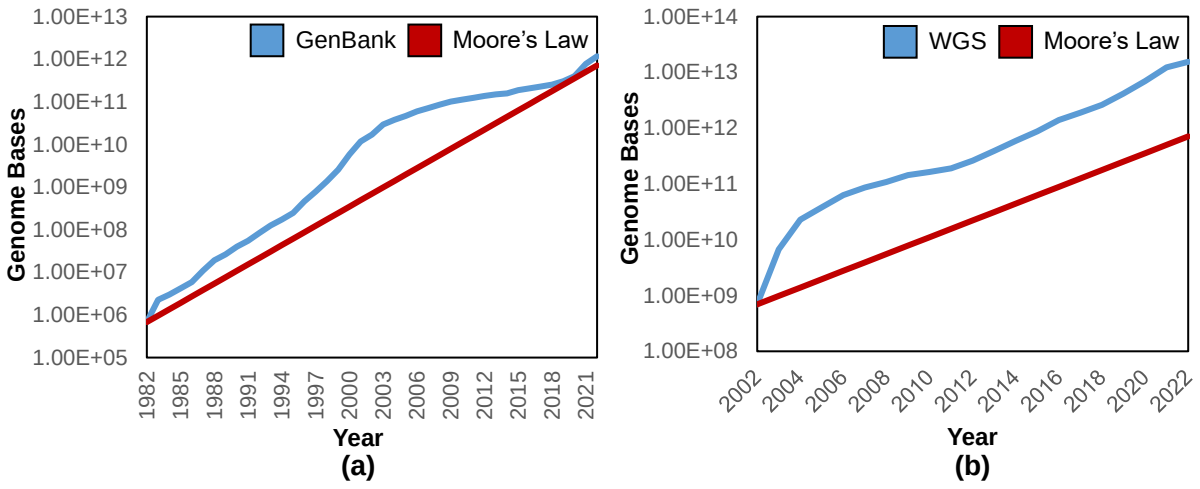

Figure 1.2: (a) Genome amount in the GenBank database over the past 30 years. (b) Genome amount in the WGS project over the past 20 years.

Illumina, i.e., a leading company for genome sequencing, expects the data produced in genome analysis to double every 12 months [13]. As shown in Table. 1.1, projecting

Table 1.1: Data Growth Projection for Four Big Data Domains in 2025 [13]

Astronomy	**Twitter**	**YouTube**	**Genome Analysis**
1EB per year	1PB to 17PB per year	1EB to 2EB per year	2EB to 40EB per year

to the year of 2025, genome analysis are going to produce much more data, i.e., 2 EB to 40 EB per year, than the amount of data produced in three other big data domains, i.e., Astronomy, Twitter, and YouTube [13]. This massive amount of data put forward great challenges for the data processing in genome analysis [4,13]. For example, in 2025, variant calling is going to require 2 trillion CPU hours per year and all-pairs genome alignments is going to require 10,000 trillion CPU hours per year [13]. Unfortunately, the speed of data processing in genome analysis lags far behind the speed of data generation [13]. As a result, hardware acceleration turns out to be necessary for genome analysis [4, 11, 14, 15].

1.1 Motivations

Due to the importance and time-consuming fact of genome analysis, various computation-centric hardware approaches, such as multi-core [16–18], GPU [19–24], and FPGA [19,25–29] have been explored to accelerate applications in genome analysis, e.g., k-mer counting [19], DNA seeding [26], DNA pre-alignment [30], seed extension [23], and variant calling [24]. However, innovations that only focus on the computation have limited space for improvement of performance and energy efficiency, since many key applications in genome analysis are memory-bound [19, 26, 30–32]. For example, the profiling results in Fig. 1.3 quantitatively demonstrate that the bottleneck of DNA seeding is the memory, instead of the computation. As Fig. 1.3 (a) shows, the DRAM access accounts for 60% of the CPI stack analysis. The *Load/Store* instructions take 43.4% among the total instructions in Fig. 1.3 (b). The energy breakdown in Fig. 1.3 (c) shows that 49.4% of the total energy consumption is consumed by the DRAM.

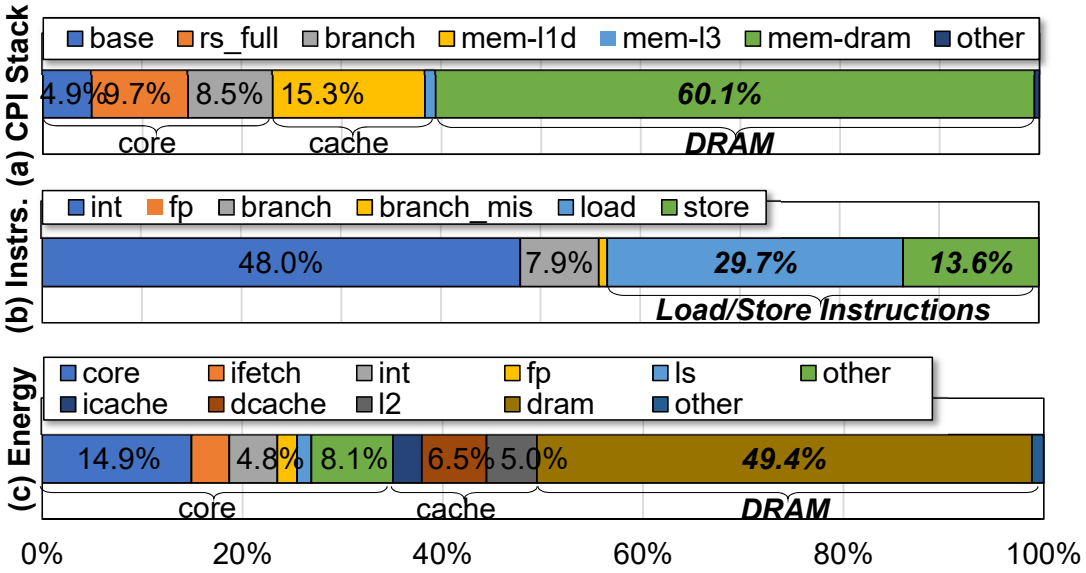

Figure 1.3: Profiling the DNA seeding in BWA-MEM [33] with Sniper [34] and configuration in Table 4.1: (a) CPI stack; (b) Instruction statistics; (c) Energy breakdown. (© 2019 IEEE)

As a comparison, the memory-centric architecture, including the Processing-In-Memory (PIM) architecture and the Near-Data-Processing (NDP) architecture, focuses on optimization of the memory. It integrates the computation and the memory closely to embrace the larger internal memory bandwidth and reduce the overhead of data movement [35–37].], leading to better performance and energy efficiency for memory-bound applications. Thus, the memory-centric architecture is also a promising candidate for these memory-bound applications in genome analysis [30, 38, 39].

1.2 Challenges

Although the memory-centric architecture is promising to accelerate the memory-bound applications in genome analysis, there are four significant challenges in designing memory-centric accelerators for genome analysis:

Fine-Grained Random Memory Access: Many applications in genome analysis involve fine-grained memory access [4,11,19,33,40], leading to memory bandwidth under-utilization and performance degradation in the computation-centric architectures. For example, for FM-index based DNA seeding and k-mer counting, only 32 Bytes data and 1 bit data is actually used for each 64 Bytes data from one memory access [19, 40, 41], indicating that the actual memory bandwidth utilization is only about 50% and 0.2% for these two applications. The memory bandwidth under-utilization leads to performance degradation for these memory-bound applications [4, 11].

In addition, the memory access patterns are usually random for these applications. For example, we profile the Last-Level Cache (LLC) miss rate for the FM-index based DNA seeding, showing 32.5% on average and up to 93.24% peak miss rate. The high LLC miss rate indicates that the memory access patterns are highly random and it's difficult to leverage data locality for these applications.

Inter-Task Divergence: The behaviors of different tasks in these applications highly depends on the data and are divergent. For example, the profiling results for FM-index based DNA seeding in Fig. 1.4 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 and up to 7.6ms. The inter-task divergence makes the SIMD accelerators inefficient and unsuitable for genome analysis [11, 36].

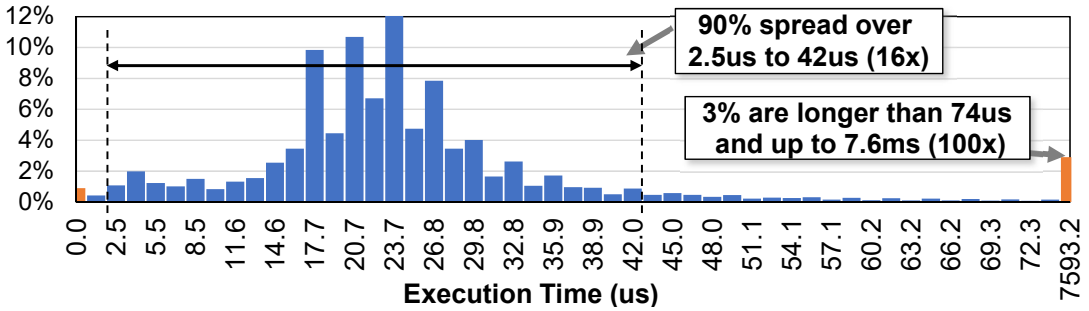

Figure 1.4: Execution time distribution of the seeding's elemental task (`bwt_smem1a` [33], the atomic function for parallel DNA seeding). (© 2019 IEEE)

Requirement for Efficient Scalability: The requirements of memory capacity for different applications in genome analysis vary significantly, depending on the specific application to work on, the dataset to deal with, the algorithm to use, and the parameters of the algorithm. For example, BWA-MEM uses 64GB memory for FM-index based DNA seeding in [42] and SMUFIN needs near 2TB memory for k-mer counting in [19]. To ensure the memory-centric accelerators for genome analysis can be used for different scenarios, it's important to provide efficient scalability without harming the performance and energy efficiency.

Emerging Memory Technology vs. Conventional Memory Technology: Memory-centric architectures can leverage both the emerging memory technology, e.g., ReRAM [43], STT-RAM [44], and PCRAM [45], and the conventional memory technology, e.g., DRAM [46] and SRAM [47]. The emerging memory technology based designs often have the potential to provide ultra-high performance and energy efficiency by enabling in-situ computation inside the emerging memory array [43]. As a comparison, the conventional memory based designs usually highlight practicality and cost-effectiveness by placing the processing elements near the conventional memory array [36]. How to design efficient memory-centric accelerators for genome analysis based on these two types of memory technologies and maintain their key advantages is an important question.

1.3 Contributions

The **goal** of this dissertation is to explore the memory-centric architecture for genome analysis, covering designs and optimizations in both hardware and software. Meanwhile, we have addressed the related challenges, i.e., fine-grained random memory access, inter-task divergence, requirement for scalability, and efficient designs for both the emerging memory technology and the conventional memory technology.

Specifically, we have one emerging memory technology, i.e., ReRAM, based PIM accelerator design.

- RADAR (Chapter 3), which leverages the high memory bandwidth, high energy efficiency, and the ability to perform in-situ parallel comparison operations within the 3D ReRAM crossbar to accelerate DNA alignment.

In addition, we also have three conventional memory technology, i.e., DRAM, based NDP accelerator designs.

- MEDAL (Chapter 4), a Dual-Inline-Memory-Module (DIMM) based NDP accelerator, which utilizes the extra intra-DIMM memory bandwidth to efficiently accelerate DNA seeding by placing the processing elements within the DIMM. Without making any modifications to the cost-sensitive DRAM dies, MEDAL is practical and cost-effective.

- NEST (Chapter 5), which combines the proposed architecture-specific k-mer counting algorithm, workflow, and optimizations with the DIMM based NDP architecture, to efficiently accelerate k-mer counting.

- BEACON (Chapter 6), which migrates the DIMM based NDP architecture to the dis-aggregated memory pool with the CXL support to provide efficient scalability and communication for k-mer counting, DNA seeding, and DNA pre-alignment.

The memory-centric architectures proposed in this dissertation explore both the emerging memory technology, i.e., ReRAM, and the conventional memory technology, i.e., DRAM. Different applications in genome analysis, i.e., k-mer counting, DNA seeding, DNA pre-alignment, and DNA alignment, are covered.

To conclude, we propose novel memory-centric architecture designs and optimizations for genome analysis to leverage the high memory bandwidth and reduce data movement,

leading to significant performance improvement and energy reduction. The proposed architectures highlight the contributions to provide efficient accelerators for genome analysis, while maximizing the key advantages of the emerging memory technology, i.e., ultra-high performance and energy efficiency, and the conventional memory technology, i.e., practicality and cost-effectiveness.

Chapter 2

Backgrounds and Related Work

In this chapter, we first introduce the backgrounds about genome analysis and memory-centric architectures. Then, we present the related work of the hardware acceleration for genome analysis.

2.1 Genome Analysis

Genome analysis is becoming more and more important, since it sets up the foundation of disease understanding [3], precise medical care [37], wildlife conservation [2], and so on [3]. As shown in Fig. 2.1, we notice that three key applications in genome analysis, i.e., DNA seeding [11,26,31], k-mer counting [4,17,48], and DNA pre-alignment [30,39,49], are good candidates for memory-centric architecture due to two reasons [11,31,48]: First, they only involve very simple arithmetic operations, e.g., integer addition and 2 bit comparison. Second, they are memory-bound and require lots of fine-grained random memory access, e.g., 1 bit useful data per memory access for k-mer counting. The following subsections present the details of these three applications.

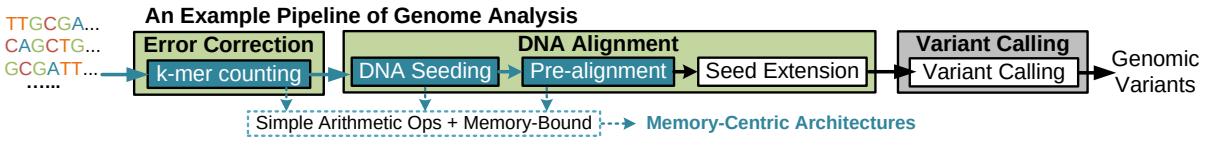

Figure 2.1: An example pipeline of genome analysis.

2.1.1 k-mer Counting

In genome analysis, the frequency information of k-mers, i.e., DNA sub-sequences 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 [19, 32, 50]. 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 are converted to other k-mers with higher frequencies via error correction [48]. k-mer counting occupies a significant portion of the runtime in many genome analysis workflows. For instance, as shown in Fig 2.2, k-mer counting is the most time consuming step in the de novo genome assembly, consuming nearly half of the total runtime in the pipeline [17].

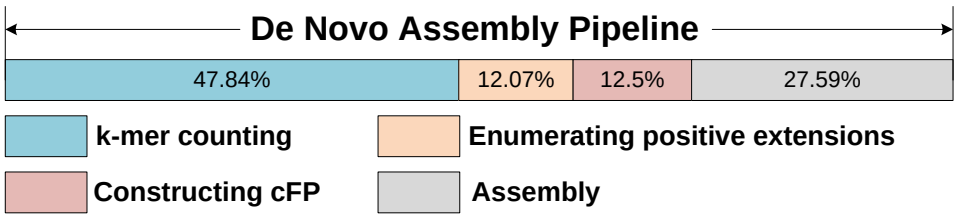

Figure 2.2: Time breakdown of the de novo assembly. k-mer counting dominates the runtime. (© 2020 IEEE)

Next, the data structures related to k-mer counting, i.e., Bloom filter and Counting Bloom filter, are introduced. Then, the workflow of k-mer counting is presented.

Bloom Filter: Bloom filter is a space efficient data structure based on hash table and it supports efficient membership checking [51, 52]. 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 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. On the other hand, 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 [53] contains an array with small counters. For example, with an array of 4-bit counters, the counting Bloom filter is able to handle counts from 0 to 15. Similar to the 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 read out is assumed to be the counter of the target item.

k-mer Counting: k-mer counting refers to the process of counting the occurrences of DNA substrings with length of k in the sequencing data. An example of k-mer counting is shown in Fig 2.3, the frequencies of different 3-mers are derived after k-mer counting.

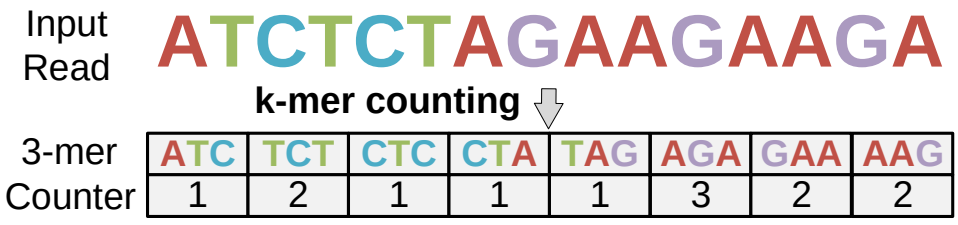

Figure 2.3: An example of k-mer counting. (© 2020 IEEE)

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 sequencing data. Non-unique k-mers refer to k-mers that appear more than once in the sequencing data. Because the unique k-mers are highly likely to be sequencing errors [40,48] and, for some sequencing data, up to 75% of the k-mers can be unique [40], k-mer counting often removes these unique k-mers and only counts the frequencies of non-unique k-mers [19, 40]. The conventional k-mer counting usually includes the following two steps [19, 40]:

- Prune: With a chain of two Bloom filters, each time a k-mer comes in, check the 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 the existence of this k-mer in the second Bloom filter constructed in the first step. If this k-mer is in the second Bloom filter, increase the corresponding frequency counter in the hash table by one. After all k-mers have gone through this process, the occurrences of the non-unique k-mers are stored in the hash table.

2.1.2 DNA Seeding

DNA seeding refers to the process of matching seeds, i.e., small sequence fragments chopped from a given read, back to the long reference genome. As one of the most time-consuming step in DNA alignment, DNA seeding takes up to 48% of the runtime for DNA alignment [11, 26].

DNA seeding algorithms usually pre-build an index of the reference genome to speedup the process of seed locating. FM-index [33, 54] and Hash-index [55] are the two main-

stream seeding indexes used by modern DNA alignment tool. Both of the algorithms can be preferable, depending on the combination of the seeding and the extension approach [55]. For example, FM-index based algorithm has good performance for BLAST-like seed extension [55] and Hash-index has good performance for local alignment [55]. FM-index based DNA seeding and Hash-index based DNA seeding are introduced in details below:

FM-index based DNA Seeding: The flow of the FM-index based seeding contains the offline pre-processing, i.e, index building, and the online seed searching, i.e., seed locating [33, 54]. During the pre-processing, the following data for the reference genome R are calculated and prepared.

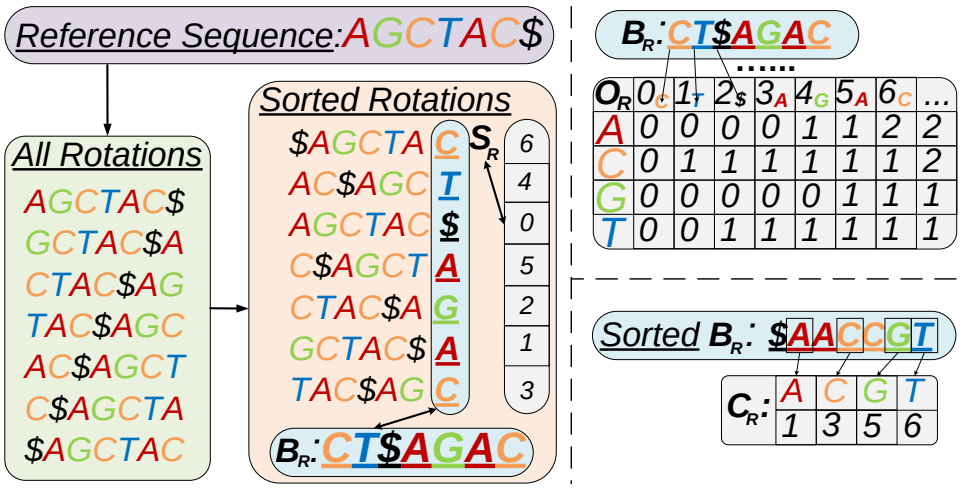

Figure 2.4: An example of data structures in the FM-index based DNA seeding. (© 2019 IEEE)

- $B_R[len]$: the Burrows-Wheeler Transform [26] of R (B_R);
- $S_R[len]$: the Suffix Array (S_A), i.e., record the original id of the 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 the 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 for the index building is shown in Fig. 2.4, the reference sequence is $AGCTAC$. The algorithm first terminates the reference sequence with a unique character \$. Then, all rotations of the above character string are generated. Next, these rotations are sorted in alphabet order. The last characters of all entries in the above sorted rotations form the $B_R[len]$. The $S_R[len]$ is also derived during this process. Finally, with the $B_R[len]$ and the sorted $B_R[len]$, the $O_R[len+1][4]$ and the $C_R[4]$ can be generated.

Algorithm 1 FM-index based DNA Seeding (© 2019 IEEE)

Input: Query sequence $SD[d]$, Reference genome $R[len]$)
Output: Matching locations
Pre-process: 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
  end if
  $I^{lower} = C_T[x] + O_T[x][I^{lower} - 1]$;
  $I^{upper} = C_T[x] + O_T[x][I^{upper}]$;
end while
for $I^{lower} <= i <= I^{upper}$ **do**
  $Match\ location[i] = S_R[i]$;
end for
Return $Match\ location$

During the online seed searching, the algorithm extends the current match by one nucleotide each iteration, reading the C_R and the 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 the current prefix of SD in S_R. This range contains all occurrences of the current prefix of SD in R. $I^l(D) \leq I^u(D)$ if and only if there is at least one match in R.

Hash-index based DNA Seeding: The Hash-index based DNA seeding enables fast retrieval of fixed length seeds [55]. Similar to the FM-index based DNA seeding, the Hash-index based DNA seeding also consists of the offline pre-processing and the online seed searching. During the pre-processing, the starting positions of the k-mers in the reference genome, where k is the length of the seed, are stored into the hash table. During the seed searching, the k-mers are used as the inputs for the pre-built hash table and the corresponding starting locations of these k-mers can be retrieved from the hash table efficiently.

2.1.3 DNA Pre-alignment

As shown in Fig. 2.1, after the seed locating with DNA seeding, seed extension is performed to check the similarity between the DNA segments extracted at these candidate locations and the corresponding DNA sub-sequences in the reference genome [11, 56].

Seed extension is computationally expensive and time-consuming. To reduce the amount of candidate locations to be examined in seed extension, a filtering method called DNA pre-alignment is adopted by DNA alignment tools [56]. DNA pre-alignment quickly determines if a candidate location is valid or not by counting the number of matching DNA bases between the query sequence and the reference genome near the candidate location [49, 56]. Candidate locations with the nearby matching DNA bases below a threshold are recognized as invalid candidate and thrown away to avoid performing the expensive seed extension.

2.2 Memory-Centric Architectures

The memory-centric architectures can be classified as the Processing-In-Memory (PIM) architectures and the Near-Data-Processing (NDP) architectures. The PIM architec-

tures, the NDP architectures, examples for both of them, and their key features are described in the following subsections.

2.2.1 Processing-In-Memory (PIM) Architectures

For the PIM architectures, in-situ computation can be performed within the memory array to provide massive parallelism and eliminate data movement. For example, as a PIM accelerator for neural networks based on the emerging ReRAM, PRIME enables in-situ computation inside the ReRAM crossbars [43]. Similarly, as a multi-purpose PIM accelerator based on the conventional DRAM, DRISA supports in-situ computation inside the DRAM memory arrays [46].

The PIM architectures usually have ultra-high performance and energy efficiency because of the massive parallelism and eliminated data movement. On the other hand, since the PIM architectures often leverages the emerging memory technologies, e.g., PRIME [43] and Pipelayer [57], or require invasive designs to the conventional memory technologies, e.g., DRISA [46] and SCOPE [58], the PIM architectures usually require relatively long time to be put into deployment.

2.2.2 Near-Data-Processing (NDP) Architectures

For the NDP architectures, the processing elements are integrated near the memory array to leverage extra memory bandwidth and reduce data movement. For example, as a NDP accelerator based on the Dual-Inline-Memory-Module (DIMM), Chameleon [36] places the processing elements on the DIMM to perform computation near the memory, i,e., the DRAM chips. Similarly, as a NDP accelerator for neural networks based on 3D-stacked memory, i.e., Hybrid Memory Cube (HMC), TETRIS [59] places the processing elements in the logic die of the HMC, which performs computation and communicates

with the memory, i.e., the DRAM dies above, through the high-bandwidth Through-Silicon Via (TSV).

Compared with the computation-centric architectures, the NDP architectures have better performance and energy efficiency because of the extra memory bandwidth and reduced data movement. Compared with the PIM architectures, the NDP architectures usually don't provide as good performance and energy efficiency, because the computation and memory are still separated as two parts. On the other hand, since the NDP architectures often rely on and don't require invasive designs to the conventional memory technology, e.g., Chameleon [36] and AIM [37], the NDP architectures usually can be put into deployment in a relatively short time.

2.3 Related Work

This subsection introduces the related work for the acceleration of genome analysis, including the computation-centric accelerators and the memory-centric accelerators.

Computation-Centric Accelerators for Genome Analysis: Different computation-centric accelerators, i.e., multi-core CPU [16–18], FPGA [19, 25–28, 60–64], GPU [19–24, 65, 66], and ASIC [14, 67], have been proposed to accelerate different applications in genome analysis, including k-mer counting [16, 17, 19, 19, 20, 25], DNA seeding [18, 21, 22, 26, 27, 60, 61], DNA pre-alignment [63, 66], seed extension [23, 28, 67], variant calling [24, 62, 65], read assembly [14, 64], and so on. For example, KMC2 is a multi-core CPU based accelerator for k-mer counting, FHAST is a FPGA based accelerator for DNA seeding [61], Gerbil is a GPU based accelerator for k-mer counting, and Darwin is an ASIC design for read assembly [14].

Unfortunately, many key applications in genome analysis are memory-bound, e.g., k-mer counting, DNA seeding, and DNA pre-alignment, the computation-centric acceler-

ators provide sub-optimal performance and energy efficiency due to the limited memory bandwidth and the frequent data movement.

Memory-Centric Accelerators for Genome Analysis: For the emerging memory technology, ReRAM has been explored to accelerate genome analysis. As an example, ReCAM PRinS accelerate seed extension by performing in-situ associative computing in the ReRAM crossbar [68]. Unfortunately, device endurance and storage efficiency are serious issues in ReCAM PRinS, because associative computing in ReCAM PRinS requires frequent write operations and uses 75% of the ReRAM cells to store the intermediate data [69]. Hamdioui *et al.* proposes a ReRAM based PIM accelerator, which can be used for DNA alignment [70]. However, this work isn't specifically designed for genome analysis, leading to its inefficiency in accelerating DNA alignment [70,71]. As a summary, the previous emerging memory based accelerators for genome analysis have the issues of limited device endurance, inefficient storage, and sub-optimal performance.

For the conventional memory technology, both the conventional DRAM and the 3D-stacked DRAM have been explored to accelerate genome analysis. For the conventional DRAM based work, AIM attaches FPGAs and dedicated buses to the DIMMs to accelerate DNA seeding [37]. However, AIM doesn't leverage rank-level parallelism inside the DIMMs, leading to limited performance improvement. Chameleon provides a general-purpose, SIMD, and DIMM based NDP architecture that can be used for genome analysis [36]. Unfortunately, because Chameleon is a SIMD accelerator, it doesn't address the challenges of inter-task divergence in genome analysis efficiently. For the 3D-stacked DRAM based designs, MPU-BWM leverages HMC to accelerate DNA seeding by placing RISC-V cores in the logic die [35]. Joardar *et al.* and Mcvicar *et al.* also propose HMC based accelerators for k-mer counting [32,50]. Although the 3D-stacked DRAM based designs provide good performance and energy efficiency, they are not as cost-efficient as the conventional DRAM based work [36]. On the other hand, they has limited capacity and

scalability [72]. To summarize, the previous conventional memory based accelerators for genome analysis have the issues of limited performance, high cost, and low scalability.

Chapter 3

RADAR: A 3D-ReRAM based DNA Alignment Accelerator

[1] BLAST, the Basic Local Alignment Search Tool [73], is one of the most widely used sequence alignment tool. To keep pace with booming genome data, acceleration of BLASTN, the most heavily used DNA version of BLAST, is necessary.

Software approaches provide speedup for BLASTN by substantially sacrificing sensitivity or by indexing the entire database with large overhead [74]. In response to the shortcomings of the software approaches, many hardware approaches, such as multi-core CPU [75], FPGA [76], and GPU [77] have been proposed to further accelerate BLASTN by leveraging its inner parallelism. However, these approaches do not address the issues of performance and energy overhead which are caused by moving the entire DNA database from memory to CPU/FPGA/GPU. ReCAM PRinS [68] addresses the data movement issue in the Smith-Waterman (SW) algorithm by performing associative computing in Re-

[1] © 2018 IEEE. Reprinted, with permission, from Wenqin Huangfu, Shuangchen Li, Xing Hu, Yuan Xie. "RADAR: A 3D-ReRAM based DNA Alignment Accelerator Architecture." 55th ACM/IEEE Design Automation Conference (DAC), 2018

sistive Random Access Memory (ReRAM) based Content Addressable Memory (CAM), i.e., ReCAM. Unfortunately, device endurance and storage efficiency are serious issues in ReCAM PRinS, because associative processing requires frequent write operations and uses 75% of the ReRAM cells to store the intermediate data [69]. In addition, the SW algorithm is different from BLASTN: the SW algorithm is used in the last stage of BLASTN and only consumes less than 1% of BLASTN's runtime [76].

Observing that the execution of BLASTN is dominated by comparison operations and introduces huge amount of data movement, we identify that 3D ReCAM [78] [79] is suitable for the acceleration of BLASTN due to its high density, low power, and ability to perform parallel in-situ comparison operations. We propose RADAR, a Processing-In-Memory (PIM) architecture which utilizes ReCAM with the aid of ASIC units, to accelerate BLASTN, eliminating data movement without encountering the issue of write endurance. However, mapping DNA sequences into 3D ReCAM is non-trivial due to huge variations in the lengths of DNA sequences, which may affect both storage efficiency and computation efficiency. Hence, we propose efficient data mapping scheme to improve the efficiencies of both storage and computation in RADAR.

The main contributions of this chapter are:

- We propose RADAR, a novel PIM architecture that utilizes 3D ReCAM to accelerate BLASTN efficiently. RADAR enables in-situ computation to achieve ultra-high performance and energy efficiency. In addition, RADAR greatly reduces write operations to address the issue of device endurance in ReRAM.

- We design a dense data mapping scheme to map DNA sequences with various lengths efficiently in RADAR. Furthermore, we propose a Tail Bits Duplication (TBD) technique to eliminate the row-level, CAM-level, and unit-level data dependencies and communication to boost the performance.

- We conduct extensive design space exploration to study the trade-offs between response time, energy consumption, energy efficiency, and area of RADAR.

3.1 Background and Motivation

This section introduces the basics of BLASTN, 3D ReCAM, and the motivation for this work.

3.1.1 BLASTN

As a DNA alignment tool, BLASTN can be divided into three stages as shown in Fig. 3.1. The inputs of BLASTN [73] are query sequences to be compared against the reference genome, and the output of BLASTN are the gapped alignments generated after the third stage.

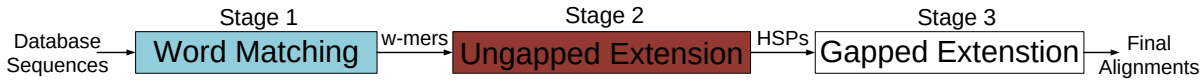

Figure 3.1: Three stages of BLASTN. The first two stages are the bottlenecks. (© 2018 IEEE)

- Word Matching, i.e., DNA Seeding: Find exact matches with length of w between the query sequences and the reference genome. These short matches are referred as w-mers.

- Ungapped Extension, i.e., Seed Extension: w-mers are extended on both sides to identify longer pairs of sequences around these w-mers, which should have more matches and fewer mismatches. These longer pairs are called High-Scoring Segment Pairs (HSPs).

- Gapped Extension, i.e., Seed Extension: Dynamic programming algorithms, e.g., the SW algorithm [80], are used to extend the HSPs into gapped alignments, which are sequence pairs that differ only by a few mismatches and gaps.

The bottlenecks of BLASTN are 'Wording Matching' and 'Ungapped Extension', which consume 83.9% and 15.9% of the runtime [76], respectively. The dominant operations in these two stages are comparison operations. Consequently, the acceleration of BLASTN should focus on these two stages and comparison operations.

3.1.2 3D ReCAM

Parallel match/mismatch operations provided by the Content Addressable Memory (CAM) make it a good candidate for search based applications. With the small cell size, non-volatility, zero standby power, and the potential to be stacked in 3D [78, 79], the emerging ReCAM provides an alternative solution to SRAM-based CAM.

The structure of 3D ReCAM is shown in Fig. 3.2(b). The vertical pillar electrode and the surrounding metal oxide forms the metal-oxide-metal structure of ReRAM cells, where the metal oxide is in contact with the horizontal plane electrodes. Multiple bits can be stored in one pillar, because 3D ReCAM has multiple horizontal layers. The horizontal plane is used as the Source Line (**SrL**) and a row of pillars are connected together to a Match Line (**ML**). The match/mismatch signal from a certain pillar will be sent to the Sense Amplifier (SA) for reading. A comparison key can slide through the input ports (**SrL**) to search the data stored in 3D ReCAM. Only one access transistor is needed per pillar. This is an extremely high-density, multi-layer, and transistor-less design of non-volatile CAM. It works in column-serial, row-parallel mode.

This chapter aims to leverage the potential of 3D ReCAM, i.e., the high density to store the reference genome and the ability to perform in-situ comparison operations,

to accelerate the 'Word Matching' and the 'Ungapped Extension' stages of BLASTN without encountering the issue of device endurance in ReCAM [69, 81].

3.2 RADAR Architecture

This section introduces the RADAR architecture, the workflow of RADAR, and the proposed data mapping scheme.

3.2.1 Architecture

The high-level architecture of RADAR is shown in Fig. 3.2(a). RADAR consists of multiple BLASTN Units, i.e., the computational units, connected by the shared bus.

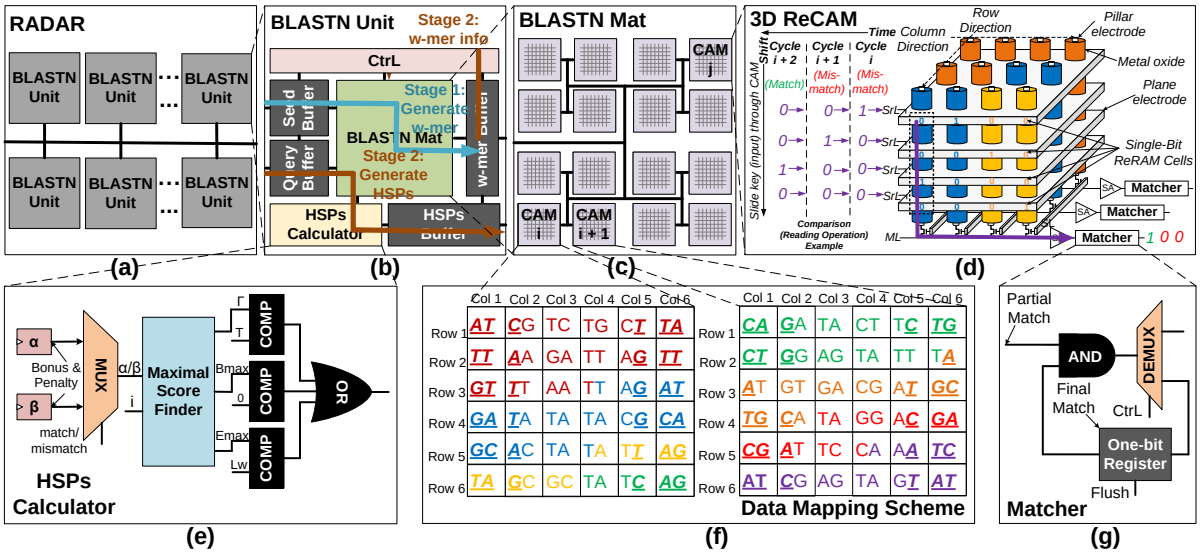

Figure 3.2: Architecture, data mapping scheme, and circuit design of RADAR. (a) Architecture of RADAR. (b) BLASTN Unit. (c) BLASTN Mat. (d) 3D ReCAM. (e) Circuit design of the HSPs Calculator. (f) Dense data mapping scheme and the TBD technique. Multiple DNA bases can be stored in one pillar. Different colors represent different DNA sequences. DNA bases in bold and with underscore stand for tail bits duplicated in ReCAM. (h) Circuit design of the Matcher. (© 2018 IEEE)

BLASTN Unit: As the computational unit in RADAR, the architecture of the BLASTN Unit is shown in Fig. 3.2(b). For the input, it receives the original query sequences and

the de-duplicated seeds in this query sequence, i.e., candidate w-mers, from the host. The de-duplicated seeds are used for 'Word Matching' and the original query sequences is used for 'Ungapped Extension'. For the output, it sends the HSPs to the host to perform 'Gapped Extension'. Different BLASTN Units perform computation in parallel, providing unit-level parallelism.

As shown in Fig. 3.2(b), a BLASTN Unit contains buffers, a BLASTN Mat, a HSPs Calculator, and the Controller (CtrL). The buffers in the BLASTN Unit store the inputs (the original query sequences and the de-duplicated seeds), the intermediate data (the w-mers), and the outputs (the HSPs) of the BLASTN Unit. The BLAST Mat stores the reference genome and performs in-situ comparison operations inside the 3D ReCAMs. The HSPs Calculator calculates HSPs according to the matching information provided by the BLAST Mat. The CtrL is connected to different modules to regulate the computational process.

BLASTN Mat: The functionality of the BLASTN Mat is to perform in-situ comparison operations inside the 3D ReCAMs for 'Word Matching' and 'Ungapped Extension'. The architecture of the BLASTN Mat is shown in Fig. 3.2(c). The BLASTN Mat contains multiple 3D ReCAMs connected to each other with a H-tree. The 3D ReCAM stores the reference genome and supports in-situ comparison operations with the target seeds as the inputs and the matching results as the outputs. The comparison operations are performed in different ReCAMs and different rows within the same ReCAM concurrently, providing both row-level parallelism and CAM-level parallelism.

In each ReCAM, the comparison operations are performed in column-serial, row-parallel order. An example of the comparison operation is shown in Fig. 3.2(d), in which the comparison key is shifted by one position per cycle. If a match occurs, the output of the corresponding row will be 1. Otherwise, the outputs of all rows will be 0. The data flow of the example comparison operation is indicated by the purple arrow. If

a comparison involves data stored in different vertical pillars, which happens when the comparison window crosses multiple pillars, partial comparisons are performed each time. The **Matcher** in Fig. 3.2(g) is designed to merge the partial comparison results. The final comparison result is available after all the partial comparisons have been performed. The data mapping scheme has significant influence on the searching process and is discussed in detail in Section 3.3.

HSPs Calculator: The circuit design of the HSPs Calculator is shown in Fig. 3.2(e). The functionality of the HSPs Calculator is to calculate HSPs for 'Ungapped Extension' according to the matching information provided by the BLAST Mat. First, the HSPs Calculator finds the maximal score for the matching scores within a fixed-size sliding window containing the w-mer. Then, the HSPs Calculator compares the maximal score with a threshold and determines if the current matching pair is a valid HSP. The scoring rule is a bonus α for matches and a penalty β for mismatches. The maximal score finding process is performed in the Maximal Score Finder. The HSPs are stored in the HSPs buffer and forwarded to the host to perform 'Gapped Extension'.

3.2.2 Workflow

This subsection describes the workflow of RADAR, including the workflow for 'Word Matching' and 'Ungapped Extension'.

Word Matching: The data flow of 'Word Matching' is indicated by the blue arrow in Fig. 3.2(b). The original query sequences and the de-duplicated seeds are passed to the buffers in different BLASTN Units from the host through the bus. Then, the de-duplicated seeds with length w are transferred to different ReCAMs via the H-tree to perform 'Word Matching'.

RADAR uses the input seed with length of w as comparison key and shifts it through the DNA sequences stored within each ReCAM to perform searching. If the final com-

parison result of an input seed is a match, this seed is recognized as a w-mer and stored in the w-mer buffers for 'Ungapped Extension'.

Ungapped Extension: The data flow of 'Ungapped Extension' is indicated by the green arrows in Fig. 3.2(b). The original query sequences in the buffers within the BLASTN Units are the inputs to the BLASTN Mat.

First, the information of the w-mer is transferred to the CtrL to determine the location for performing 'Ungapped Extension' in the original query sequence. Then, the characters within a fixed-sized window with size L_w, centered on the target w-mer, are transferred from the query buffer to ReCAMs to perform pair-wise DNA base comparison. These pair-wise match/mismatch information is forwarded to the HSPs Calculator, as shown in Fig. 3.2(e). Next, the HSPs Calculator finds the max score of the HSPs candidate, i.e. character pairs. If the maximal score passes a threshold, this HSPs candidate is recoginized as a HSP and is passed to the host to perform 'Gapped Extension'.

3.3 Data Mapping Scheme

It is challenging to map DNA sequences with various lengths into RADAR due to the following two challenges:

Challenge-1: Managing sequences in an aligned manner in 3D ReCAMs and leaving the remaining bits unused bring significant storage overhead.

Challenge-2: Segmenting sequences into multiple rows and ReCAMs introduces data dependency across multiple rows and ReCAMs, leading to performance degradation.

We propose the dense data mapping scheme and the Tail Bits Duplication (TBD) technique to address these two challenge in RADAR.

3.3.1 Dense Data Mapping Scheme

To address challenge-1, we propose a dense data mapping scheme to avoid storage overhead. The proposed dense data mapping scheme sequentially allocates the DNA sequences one after another in the 3D ReCAMs as shown in Fig. 3.2(d) and (f). Different DNA sequences are represented with different colors.

Because there are 4 nucleic acids, i.e., 'A/T/C/G', ideally we can encode these 4 nucleic acids with 2 bits. Unfortunately, ReCAM cannot distinguish seeds with different numbers (> 0) of matches, i.e., 1s, in one comparison. If there is any match within a comparison, the corresponding comparison result is a match, i.e., high current. As a result, with the 2-bit encoding, ReCAM cannot distinguish different nucleic acids in one comparison. For example, the comparison results of '01' against '01' (1 match) and '11' (2 matches) are both matches. To distinguish 4 nucleic acids in one comparison, RADAR uses 4 single-bit ReRAM cells to encode 1 nucleic acid.

3.3.2 Tail Bits Duplication (TBD)

To address challenge-2, we propose the TBD technique. As shown in Fig. 3.2(f), the TBD technique duplicates the tail bits in each row. If the DNA sequences cross multiple rows or multiple ReCAMs, the head bits in the following row or the following ReCAM are set as the tail bits in the previous row or the previous ReCAM. The number of the tail bits is at most $w - 1$.

The TBD technique eliminates data dependency across multiple rows, enabling row-level, CAM-level, and unit-level parallelism. Assuming the length of the seeds is w and each 3D ReCAM has M rows, N columns, and L layers. The storage overhead, i.e. Redundancy Ratio, has an upper bound given by the equation:

$$Redundancy\ Ratio \leq \frac{w-1}{N*(L/4)} \times 100\% \tag{3.1}$$

Considering the fact that the typical value for w is 11 for BLASTN and $N*(L/4)$ is in the order of a thousand, the *Redundancy Ratio* is typically far below 1%.

3.4 Discussion

Reduction in Data Movement: Unlike the computation-centric accelerators [74–76], as a memory-centric accelerator, RADAR performs in-situ comparison operations within the 3D ReCAMs. In this way, the amount of data movement is greatly reduced, leading to performance improvement and energy reduction.

Design Configuration: RADAR provides row-level, CAM-level, and unit-level parallelism. Adjustments can be made between these three levels of parallelism by changing the size of 3D ReCAMs, the number of ReCAMs per BLASTN Mat, and the number of BLASTN Units.

Device Endurance: RADAR doesn't confront the issue of device endurance in ReCAM [69], because there are no write operations in RADAR other than initially writing the reference genome into the ReCAMs.

Scalability: Due to the high density of the 3D ReCAM, RADAR is able to store the reference genome in a single chip, as the experiments demonstrate. If the target genome happens to be too large to be fitted into a single chip, distributed RADAR can be used. Different RADAR nodes perform BLASTN locally and these local results are merged to generate the final result after different RADAR nodes complete their own computation.

Migration to Other Technologies: Although RADAR utilizes 3D ReCAM, RADAR can also be implemented with other CAMs, including other NVM-based CAMs [82] and

conventional SRAM-based CAM. Minor modifications to the RADAR architecture are needed for these CAM technologies.

Extension to Other Applications: Because searching is a commonly used operation in various applications, RADAR can be extended to accelerate other applications such as text searching and BLASTP [83], the protein version of BLAST.

3.5 Experiments

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

3.5.1 Experimental Setup

Configuration of the Baseline: The baseline in the experiments is NCBI BLASTN 2.6.0+, running in a server with an Intel Xeon E5-2680 v3 CPU.

Configuration of RADAR: We build a simulator in C++ to simulate the performance of RADAR. The parameters of 3D ReCAM are extracted from NVSim [84]. The ASIC circuits are realized in Verilog and synthesized in Design Complier [85] to get the area, latency, and power. We evaluate 5 configurations of RADAR with different numbers of ReCAM Rows, ReCAM Columns, ReCAM Layers, and ReCAMs per BLASTN Mat. As shown in Fig. 3.3 and Fig. 3.4, the configurations of RADAR are represented in the following format:

($Row\ Number$, $Column\ Number$, $Layer\ Number$, $CAMs\ per\ Mat$).

Datasets and Query Sequences: We evaluate 6 datasets with different sizes using query sequences of length 100.

All the experimental are normalized to the corresponding data of the CPU baseline.

3.5.2 Performance Improvement

Compared with the CPU baseline, the performance improvement of RADAR is shown in Fig. 3.3. All RADAR configurations have very good performance improvement in accelerating BLASTN, because RADAR provides massive parallelism, including the row-level, the CAM-level, and the unit-level parallelism.

According to the experimental results, the performance improvement of RADAR grows with the size of the datasets, e.g., the performance improvement of the first configuration grows from 299x up to 16,900x when the size of the dataset grows from 1.28GB to 67.01GB, indicating that RADAR is very good at handling large datasets.

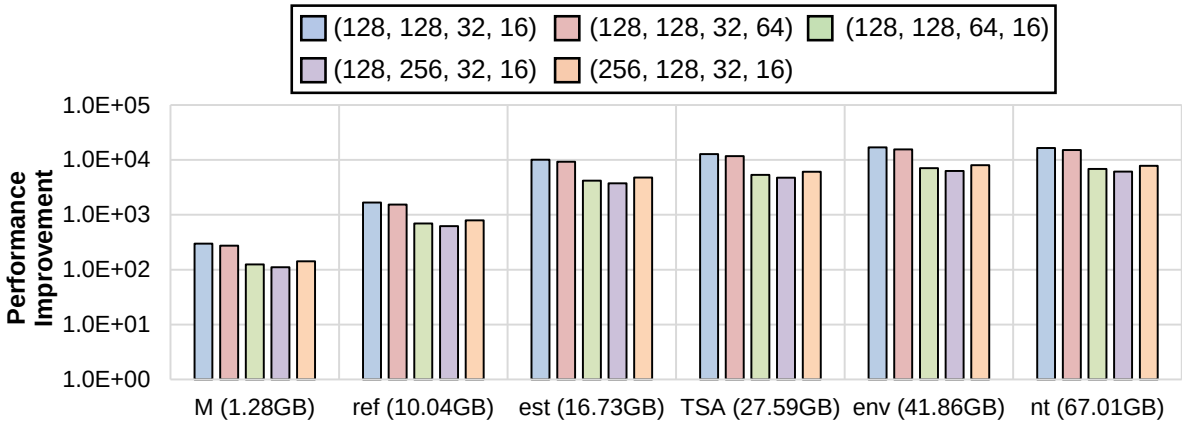

Figure 3.3: Performance improvement of RADAR. (© 2018 IEEE)

3.5.3 Energy Reduction

Compared with the CPU baseline, the energy reduction of RADAR is shown in Fig. 3.4. All RADAR configurations achieves very good energy reduction in accelerating BLASTN, because RADAR greatly reduces the data movement by performing in-situ comparison operations inside the 3D ReCAMs.

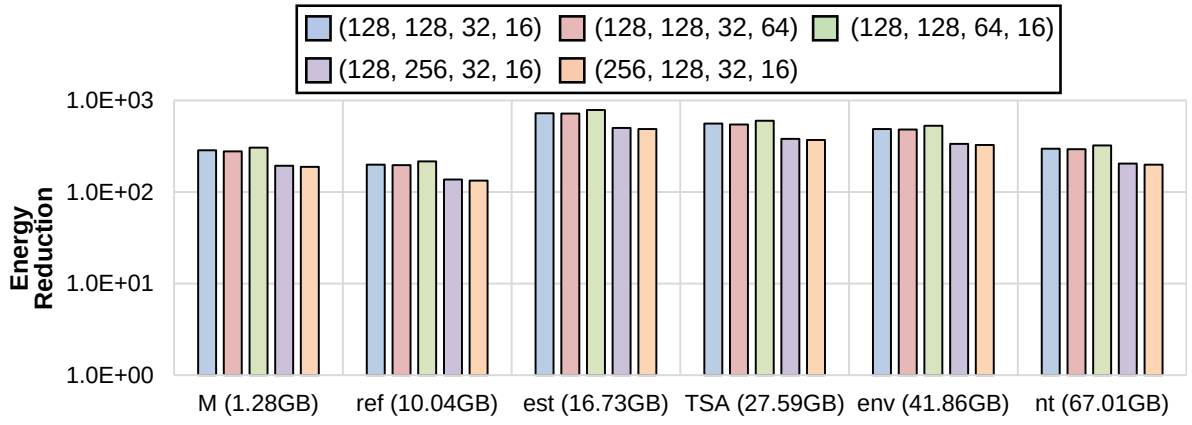

Figure 3.4: Energy reduction of RADAR. (© 2018 IEEE)

3.5.4 Trade-Offs

The trade-offs between performance improvement, energy reduction, and chip area can be achieved with different configurations, as shown in Fig. 3.3, Fig. 3.4, and Fig. 3.5.

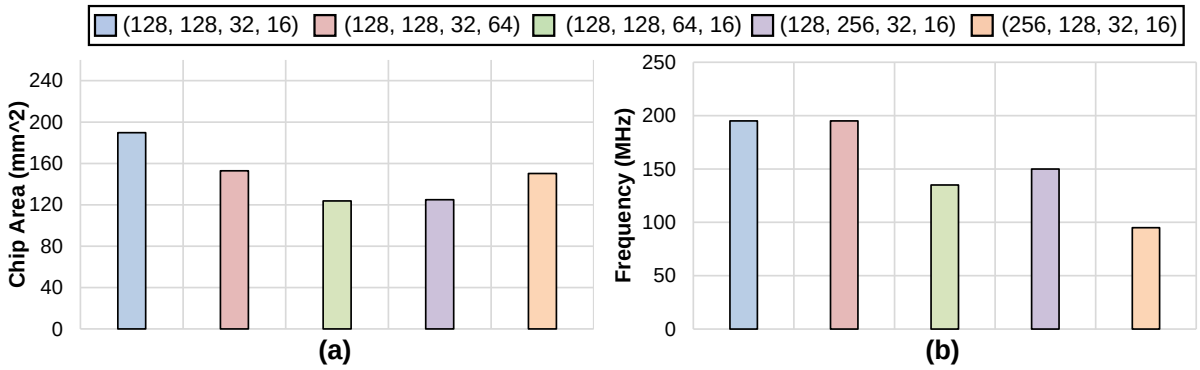

Figure 3.5: (a) Chip area for different RADAR configurations. (b) Working frequencies for different RADAR configurations. (© 2018 IEEE)

According to the experimental results, the configurations with more ReCAMs per BLASTN Unit, layers, columns, and rows reduce the chip area, but decrease the working frequency. Larger ReCAMs and larger BLASTN Mats lead to longer read time within the ReCAM, leading to degradation in performance improvement and energy reduction.

3.5.5 Comparison to Other Accelerators

As shown in Table. 3.1, compared to the Multi-Core CPU/FPGA/GPU based accelerators of BLASTN, RADAR has the best performance and outperforms them between 53x and 1896x in performance.

Table 3.1: Performance Improvement of Different Accelerators for NCBI BLASTN (© 2018 IEEE)

Design	RADAR	Paracel BLAST [75]	Mercury BLAST [86]	TUC BLAST [87]	CUDA-BLAST [77]
Hardware	3D ReCAM	32-CPU Cluster	FPGA	FPGA	GPU
Speedup	5114x	96x	11x	37x	2.7x

3.5.6 Performance of Tail Bits Duplication (TBD)

The TBD eliminates data dependencies between different rows and ReCAMs, leading to performance improvement and energy reduction. To evaluate the performance of TBD, we conduct experiments on RADAR with and without TBD.

As shown in Fig. 3.6 (a) and (b), performance improvement and energy reduction between 1.63x and 3.52x can be achieved with TBD for different configurations of RADAR. Because more rows introduce extra inter-row data dependencies and TBD can eliminate these inter-row data dependencies, the performance gain of TBD increases with the number of rows per ReCAM.

3.5.7 Energy and Area Breakdown

The energy and area breakdown for the first configuration of RADAR is shown in Fig. 3.7. For the energy, according to the experimental results in Fig. 3.7(a), 99.95% of the energy is consumed by the ReCAMs and the leakage energy of the ReCAMs is only 5.18%. As for the area, as shown in Fig. 3.7(b), 74.13% of the area is occupied

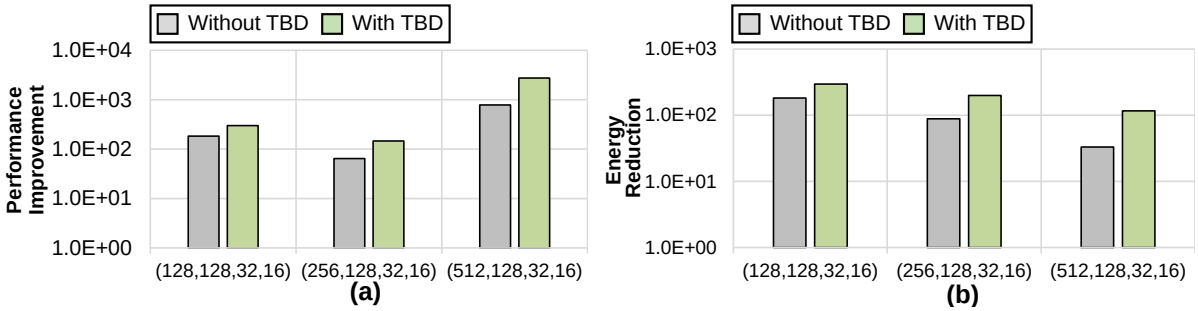

Figure 3.6: Performance of TBD. (a) Performance improvement. (b) Energy reduction. (© 2018 IEEE)

by the ReCAMs. The buffers and HSPs Calculators occupy 21.75% and 4.12% of the area, respectively. This experimental results show that RADAR is a memory-centric accelerator based 3D ReCAMs with low leakage power and small area overhead.

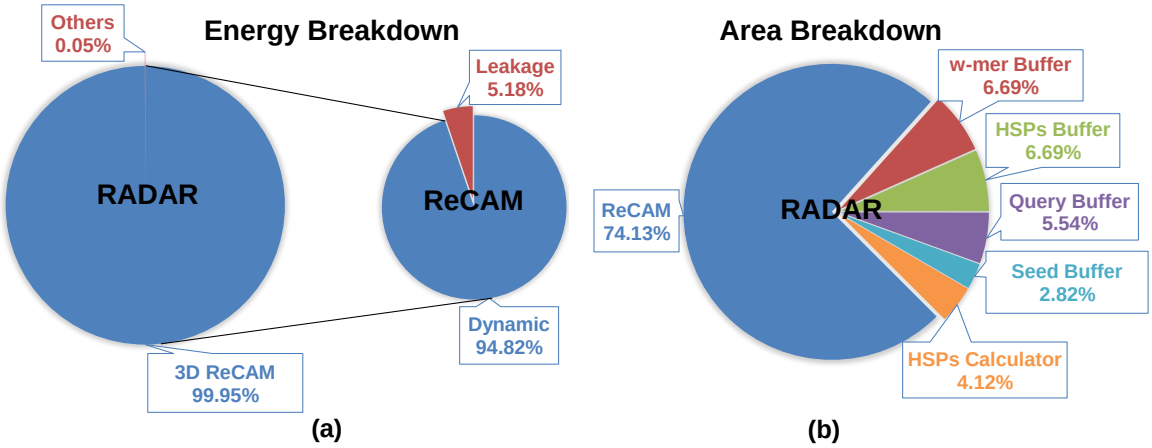

Figure 3.7: (a) Energy breakdown. (b) Area breakdown. (© 2018 IEEE)

3.6 Conclusion

To address the issue of frequent data movement in BLASTN, we propose RADAR, a high performance and energy efficient PIM architecture based on the 3D ReCAM, to accelerate BLASTN. In addition, we propose a data mapping scheme, including the dense data mapping scheme and the TBD technique, to enable efficient data storage and

support parallel computation. RADAR distinguishes itself from the previous ReRAM based PIM accelerator for DNA alignment with efficient data storage and no issue of device endurance.

Experimental results show that, with less than 1% storage overhead, the proposed TBD technique improves the performance and reduces the energy of RADAR from 1.63x to 3.52x. Compared with the CPU baseline, 5114x performance improvement and 386x energy reduction can be achieved with RADAR. Compared with the Multi-Core/FPGA/GPU based accelerators, RADAR outperforms them between 53x and 1896x in performance.

Chapter 4

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

[1] The **goal** of this chapter 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 the DRAM chips and the standard data bus. MEDAL highlights practicability by using off-the-shelf DRAM chips 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

[1] © 2019 IEEE. Reprinted, with permission, from Wenqin Huangfu, Xueqi Li, Shuangchen Li, Xing Hu, Peng Gu, Yuan Xie. "MEDAL: Scalable DIMM based Near Data Processing Accelerator for DNA Seeding Algorithm." Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO), 2019.

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 naïvely interleaving data across multiple chips. The second technique is the bandwidth-aware data mapping. It duplicates or remaps data across all the available DRAM 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 boosting the bandwidth efficiency.

Across ranks, when the index data cannot be fitted into one rank, we then propose three design options to support multi-rank scaling out. The proposed methods highlight the practicability. 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 leverages 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 the memory footprint, introduce more space for data mapping to utilize, and reduce the communication overhead.

The main contributions of this chapter are listed as follows.

- We propose a high-performance, energy-efficient, scalable, practical, and cost-effective NDP accelerator architecture, i.e., MEDAL, for DNA seeding with off-the-shelf DRAM chips.

- 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 support efficient scaling out.
- In addition, we propose an algorithm-specific data compression technique to reduce the memory footprint, introduce more space for data mapping to utilize, and reduce communication overhead, leading to performance improvement.
- The experimental results show 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.

4.1 Background

This section introduces the background of this chapter, including the supported algorithm and the Buffered Dual-Inline Memory Module (DIMM).

Supported Algorithms: As described in Section 2.1, 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, the main target of this chapter 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.

Buffered Dual-Inline Memory Module (DIMM): Dual-Inline Memory Module (DIMM) is a widely used memory package with 64 data (DQ) pins, excluding the ones

for ECC. 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. 4.1(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 integrity of DQ signals.

4.2 MEDAL Architecture

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

The goal of MEDAL is to leverage the NDP architecture for exploiting extra bandwidth for DNA seeding, while remaining practical by using off-the-shelf host processors and DRAM chips. To this end, MEDAL exploits extra bandwidth and parallelism inside the DIMMs 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 4.1, compared with the conventional case where only ranks in different channels can be accessed in parallel, MEDAL exploits 12× more bandwidth. Compared with the previous NDP work [37] 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. 4.1(a) and (b). Five components below are added into the commercial LRDIMM and these components are described in details below:

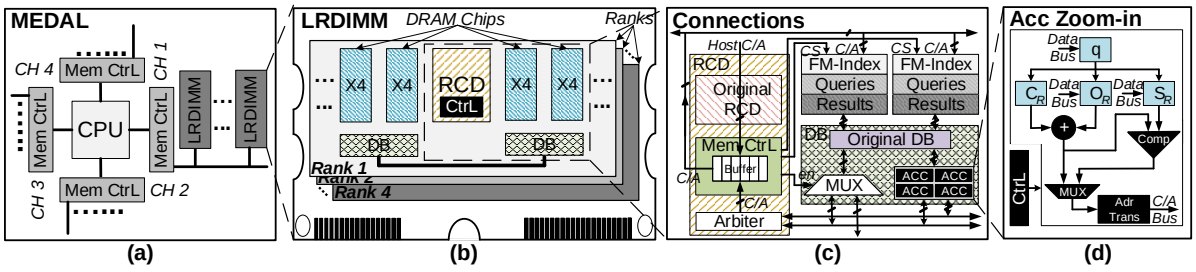

Figure 4.1: (a) High-level architecture of MEDAL. (b) Architecture of the LRDIMM. (c) Micro-architecture of the DB. Lightweight, customized logic is inserted into it. (d) Connections between the RCD, the DB, and the DRAM chips. (© 2019 IEEE)

DB-Side Accelerator: We attach 4 DNA seeding specific hardware accelerators to each DB in the LRDIMM, as shown in Fig. 4.1 (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 is connected with 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 re-organization 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 (please refer to Section 4.2.1 for the details).

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 Memory Controller (MC).

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

Our design philosophy is to enable the RCD to coordinate these memory requests, since the RCD has the information of memory requests from both the host and the accelerators. We modify the RCD in the LRDIMM, design the RCD-side MC, as shown in Fig. 4.1 (c), and propose a host-prioritized request scheduling (please refer to Section 4.2.2 for the details), to address this issue. The original RCD only serves as an enhancement module for the C/A signals. In MEDAL, the following modifications are performed:

• The C/A signal from the host is detoured to the RCD-side MC before going to the DRAM chips.

• 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 scheduling engine prioritizes the host-side requests. This is because the 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 (please refer to Section 4.2.2 for the details). For the accelerator-side requests, the scheduling engine applies the first-come-first-serve rule. The scheduling engine also makes sure the DRAM timing constraints are met, and helps the following controllers to generate the C/S signal and the enable signal on the right time.

• A controller to generate the dedicated CS signals to each DB according to the timing information from the MC scheduler, instead of a global CS bus. The details of the chip selection optimization is introduced in Section 4.2.1.

• A controller to generate the enable signal for the FIFO in the DB-side multiplexer. Since the data accessed by the accelerator is transferred to the data bus, the enable

signal makes sure the FIFO catch and buffer the data when the bus is unavailable.

Inter-Chip Hierarchical Bus: Another design challenge is that, with the proposed algorithm-specific data mapping discussed in Section 4.2.1, the $O_R[x][i]$ is spread over all the DRAM chips across the rank. The distributed data mean the DB-side accelerator may need the data from other DRAM chips within this rank. However, there is no connection between different DRAM chips within a rank in the vanilla LRDIMM, 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., the 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 chips to its destination accelerator. The FIFOs in the DB-side multiplexer are the masters writing the data, i.e., the DRAM data, to the slaves, i.e., the accelerators. Both of the *ID/address bus* and the *data bus* are multi-master single-channel buses. The bus is simplified from standard bus like AMBA [88], and it has the following features:

• a shared clock signal and reset signal for both buses,

• a 1-bit dedicated master/slave selection signal for each master/slave,

• 8-bit write data signals for the *ID/address bus*, 64-bit bi-direction data signals for the *data bus*,

• 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*.

The bus address signals are eliminated, since the RCD-side arbitrator, which is introduced in details below, has already been aware for 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 the 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 rule to grant the bus for each accelerator, so that all the accelerators 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 the 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 the 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 the 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 of MEDAL. First, we show the simpler case when the memory footprint is small enough to be fitted in one rank, focusing on intra-rank optimizations to address the challenges of **Fine-Grained Random Memory Access** and **Inter-Task Divergence** we mentioned in Section 1.2. Then, we describe the general case when the memory footprint is large and inter-rank communication is necessary, focusing on techniques to address the challenge of **Requirement for Efficient Scalability** we mentioned in Section 1.2.

4.2.1 Intra-Rank Workflow and Optimizations

In this subsection, after a detailed description of the architecture and control flow, we address the challenges of **Fine-Grained Random Memory Access** and **Inter-Task Divergence** 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 this 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 chips 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 the proposed address mapping is to aggregate the 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. 4.2 (a) with an example of memory configuration in Table 4.1. Channel, rank, and bank indexes are mapped to the lower significant bits in the logic address, in order to improve memory-level parallelism and

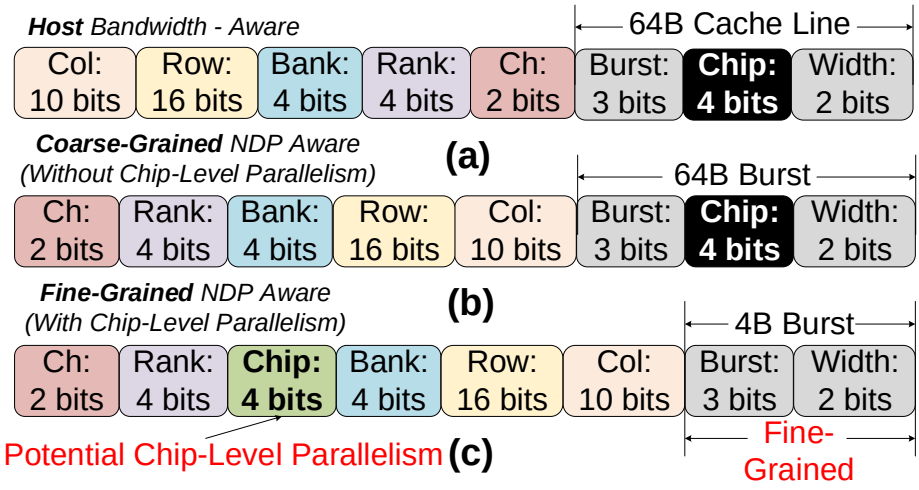

Figure 4.2: Algorithm-specific address mapping. (© 2019 IEEE)

effective bandwidth. However, interleaving data across channels/ranks destroys locality for NDP and always requires remote data access to other DRAM chips. Instead of interleaving data across channels/ranks, the optimization-1 as shown in Fig. 4.2(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. 4.2(c).

We consider the CS signals as a part of the device address, which is discussed in detail in 'Enabling Individual Chip Select' below. Originally, as shown in Fig. 4.2(a) and (b), 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. The proposed change improves the

task locality so that the inter-chip communication is minimized, leading to improvement in the chip-level parallelism.

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

Enabling Individual Chip Select (ICS): As mentioned in Section 1.2, 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 the 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 the 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. The upper part in the Fig. 4.3 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 [89], 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 the previous memory commands.

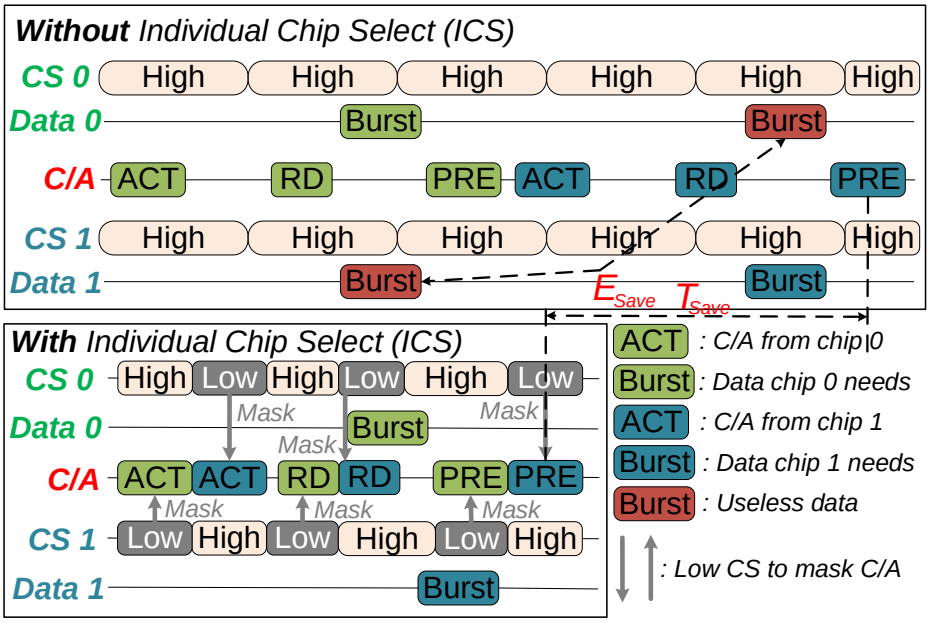

Figure 4.3: With/Without Individual Chip Select (ICS). (© 2019 IEEE)

The lower part in Fig. 4.3 shows the 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 the proposed ICS technique, the latency is reduced due to the pipelined commands. Furthermore, all output data are useful, from where the energy is saved.

4.2.2 Inter-Rank Design for Scaling Out

In this subsection, we look at application scenarios 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 describe two methods to support the inter-rank scaling out without making 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, to reduce memory footprint and the communication overhead.

Support Inter-Rank Comm. with CPU-polling: Our goal is to support the inter-rank scaling out without making 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 [37] deals with the similar scaling out problem by designing an additional bus across DIMMs. In addition to the design simplicity, the proposed method can achieve 3.43x speedup and 2.89x energy reduction, compared with AIM (please refer to Section 4.4 for details).

The CPU-polling based inter-rank communication works in the following steps, as shown in Fig. 4.4. 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. The information about remote memory access is sent back to the host. 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.

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 the polling operation is read operation in MEDAL and the proposed host-prioritized request scheduling is be applied, extra latency is needed, leading to performance degradation. To address the above issues, we propose to leverage the interrupt mechanism and the Reserved for Future Use (RFU) pin in LRDIMM [90], so that requests from the DB-side accelerators notify CPU through the RFU pin which we connect to the Advanced Programmable Interrupt Controller (APIC).

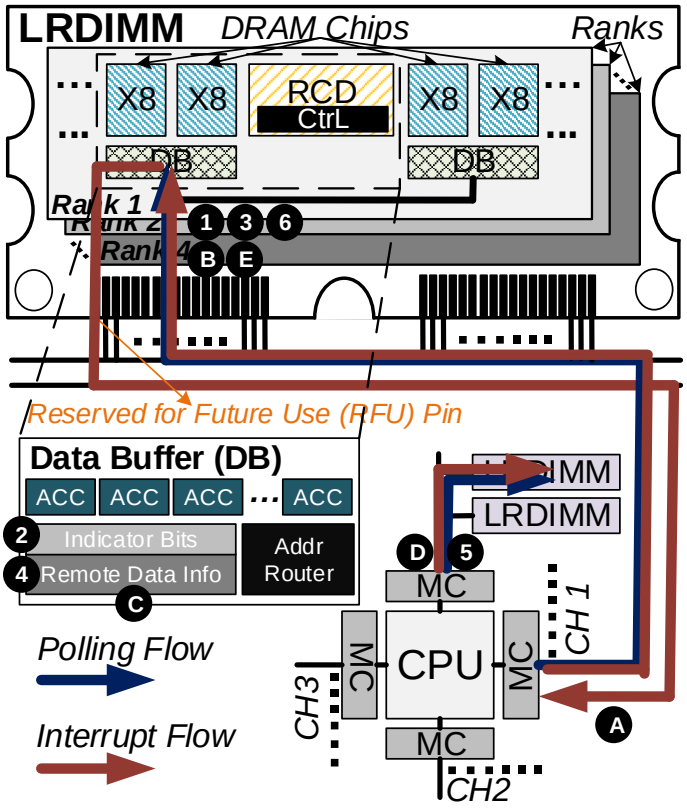

Figure 4.4: Work flow of polling-based and interrupt-based inter-rank communication. (© 2019 IEEE)

Specifically, the interrupt based inter-rank communication works in following steps, as shown in Fig. 4.4, Ⓐ 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. The information about remote memory access is sent back to the host. Ⓓ 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 we've 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 constraints for the host-side MC.

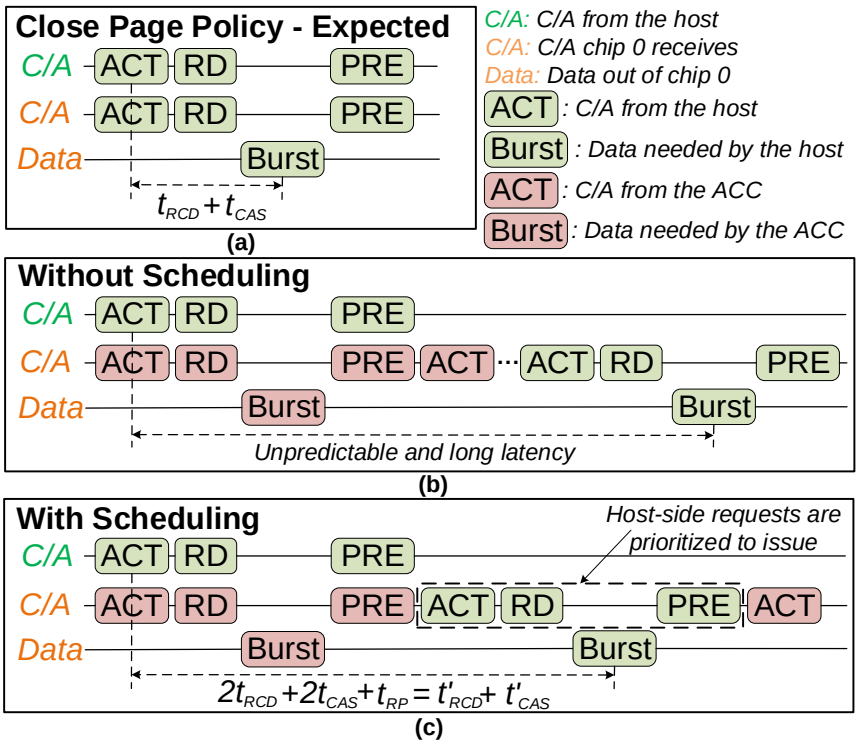

Figure 4.5: Host-prioritized request scheduling. (© 2019 IEEE)

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. 4.5, with close-

page policy, the host-side MC expects 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 are issues with the DDR timing constraints. 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 these 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 [91,92]. Alongside the DRAM, the NVM on NVDIMM-P can also be memory-mapped, e.g., Intel Optane Technology [93]. 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 the NVMs, which can be up to 10× denser than DRAM [94], 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. 4.6 (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 the on-DIMM NVM.

Reduce Memory Footprint and Comm. with Data Compression: To reduce the memory footprint and inter-rank communication, we propose an algorithm-specific data compression. The key idea of the proposed data compression is to compress data and increase locality. Note that the proposed data compression can work together with all designs described above and provide additional benefits.

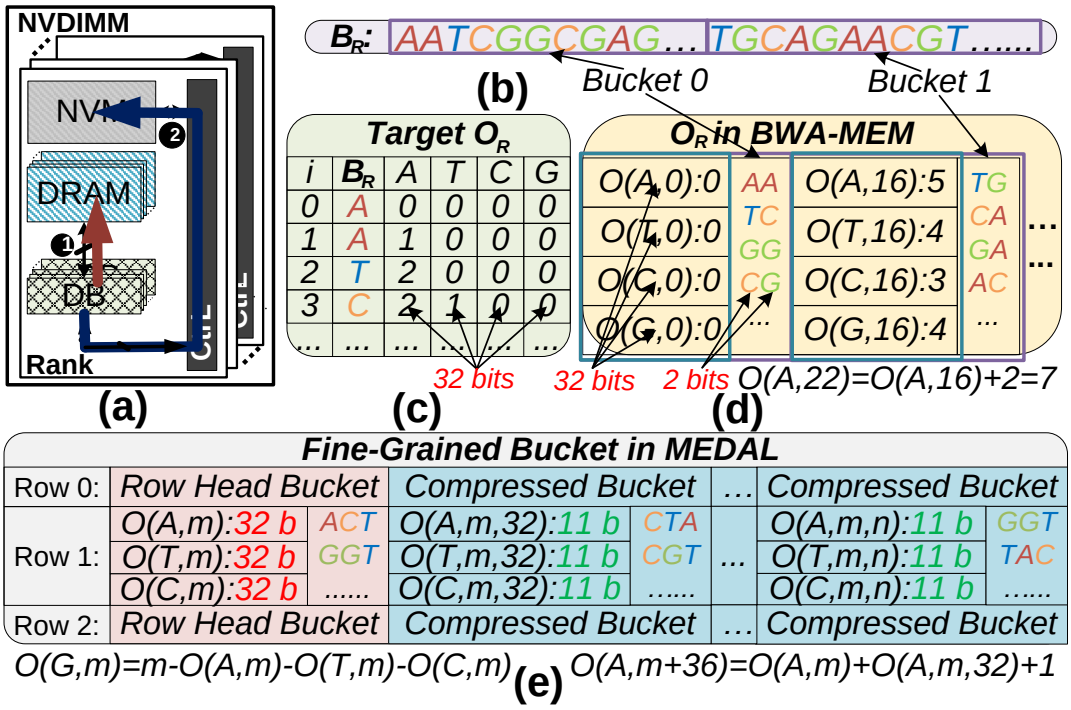

Figure 4.6: (a) Processing within NVDIMM. (b) Burrows-Wheeler Transform of the reference sequence. (c) The target O-table. (d) Bucket structure in BWA-MEM. (e) Fine-grained bucket in MEDAL with two types of buckets, i.e., row head bucket and compressed bucket. (© 2019 IEEE)

(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. 4.6 (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.

(2) Bucket data structure in software: As shown in Fig. 4.6 (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 first locates the target bucket. Then, the values of $O_R(x,i)$ in the bucket head and the nucleotides in B_R within the bucket body are read out. Finally, the values of target $O_R(x,i)$ can be reconstructed with the up-to-date values in the checkpoint, i.e., the bucket head, and B_R, i.e., the bucket body, via counting. As shown in Fig. 4.6 (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 the proposed data compression.

There are two types of buckets with the proposed data compression, i.e., row head bucket and compressed bucket. As shown in Fig. 4.6 (e), with the 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 the 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. Furthermore, in the bucket head, we only store 3 values, the last value can be derived by subtraction.

4.3 Discussion

Extension to Other Applications: MEDAL solves the problem of fine-grained random memory access. Our future work will extend MEDAL to other applications by replacing the logic in the DBs with general purpose processors or FPGAs. We expect applications such as graph processing [95], database searching [96], and sparse matrix computing [97] to also benefit from the proposed techniques.

Interface Choice: Similary to [36, 98, 99], 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 scale out easily. Note that the proposed optimizations/techniques can be easily applied to build PCIe/IO based accelerator.

System Integration and User Interface: MEDAL does not require modification to either the DRAM chips or the CPU chip. MEDAL connects to the system with standard DDR bus, i.e, with DIMM slots. As described in Section 4.2.1, the host controls MEDAL with memory instructions.

To this end, the software stack needs modifications. Similar to other NDP/PIM solutions [95, 99, 100], we need the OS to reserve the memory space in the 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 are 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 provide an Application Programming Interface (API) for programmers to control the application

Table 4.1: Configure of the Server and MEDAL (© 2019 IEEE)

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

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.

4.4 Experiments

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

4.4.1 Experimental Setup

Configuration of the Baseline: The baseline for the FM-index based DNA seeding and the Hash-index based DNA seeding are BWA-MEM [33] and SMALT [101], respectively, running in a server with an Intel Xeon E5-2680 v3 CPU. The detailed configuration information of the server is shown in Table 4.1.

Table 4.2: Design Parameters of Customized Logics in DB (© 2019 IEEE)

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

Configuration of MEDAL: Ramulator [102] is modified to build a cycle-accurate simulator for MEDAL. The configuration of MEDAL is shown in Table 4.1. The timing, energy, and area parameters of the customized logics in the DB are estimated by pre-layout Design Compiler [103] with 28 nm technology [104]. We set the 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.

These parameters of the customized logics in the DB are shown in Table 4.2. 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 4.2. The timing parameters of the DRAM chips used in the experiments are shown in Table 4.1. The energy consumption of DRAM is derivated by feeding the command trace of DRAM from Ramulator to DRAMPower [105]. We use the parameters of energy for datapath from CACTI-IO [106]. The timing and energy parameters for NVM in the NVDIMM are estimated with Intel's Optane memory [107, 108]. 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 [36] and AIM [37]. We use the same memory configuration for these two accelerators. For Chameleon, because it doesn't support any kind of communication, we add the 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 [109] are used. The name of these ten databases are shown in Fig. 4.7. We name them as DB1 to DB10 for short in other figures.

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

4.4.2 Intra-Rank Evaluation

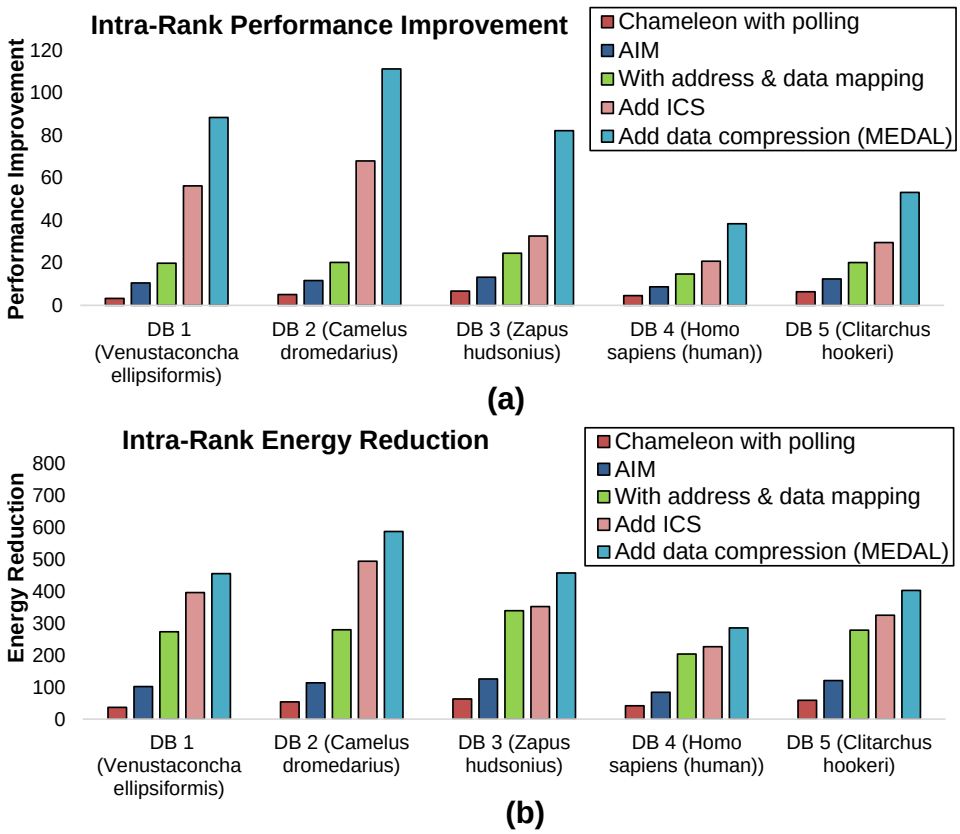

Figure 4.7: Intra-rank evaluation. (a). Performance improvement. (b). Energy reduction. (© 2019 IEEE)

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. 4.7 (a) and (b). All results are normalized to the that of the 16-thread CPU.

For intra-rank tasks, with only the proposed address and data mapping, MEDAL outperforms the 16-thread CPU, Chameleon, and AIM by 19.62x, 3.91x, and 1.75x. Then, the ICS improves the performance of 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. Combining all proposed techniques together, MEDAL outperforms the 16-thread CPU, Chameleon, and AIM by 69.69x, 13.90x, and 6.23x, respectively.

As a comparison, AIM performs coarse-grained memory access with low memory bandwidth utilization for DNA seeding. 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 ability to perform fine-grained memory access, which provides higher memory 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.

4.4.3 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. 4.8 (a) and (b).

For inter-rank tasks, on average, the polling-based design outperforms the 16-thread CPU, Chameleon, and AIM by 9.97x, 3.57x, and 1.35x, respectively. The interrupt-based

design outperforms the above platforms by 14.81x, 5.31x, and 2.01x, respectively. The NVDIMM-based design outperforms them by 16.09x, 5.76x, and 2.19x, respectively.

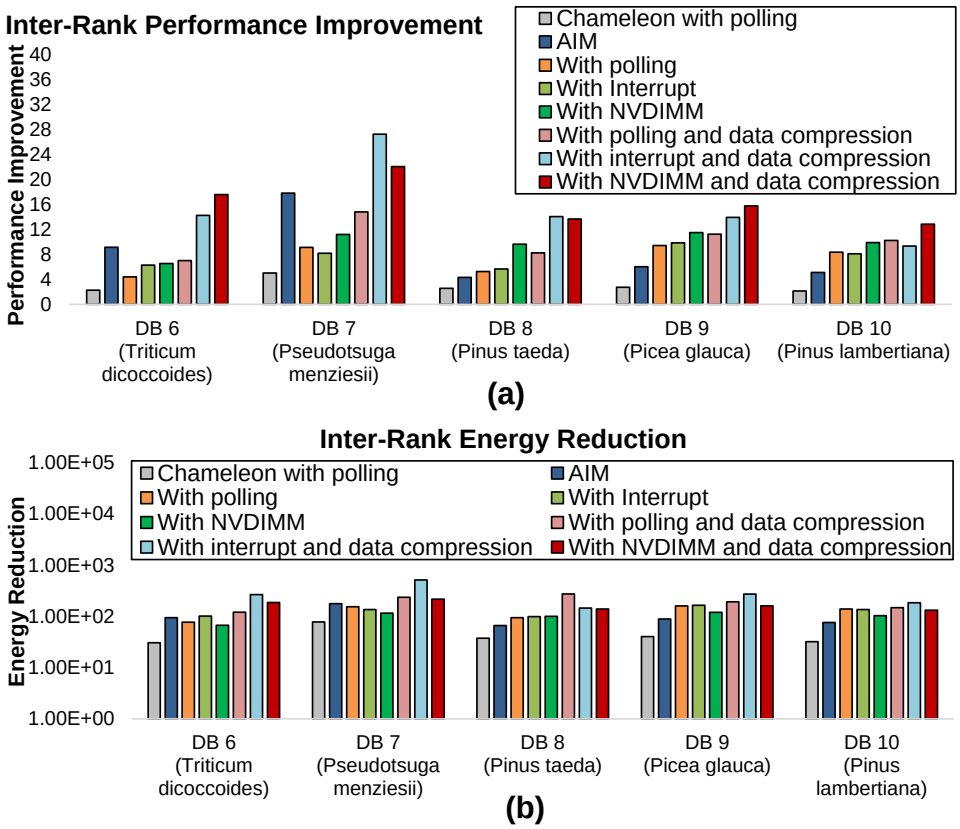

Figure 4.8: Inter-rank evaluation. (a). Performance improvement. (b). Energy reduction. (© 2019 IEEE)

Compared with the polling-based design, the interrupt-based design has better performance due to two reasons: First, the interrupt-based design doesn't need to occupy the memory channel to perform polling operation, meaning there is no negative effect on memory bandwidth. Second, the polling operation is read operation and due to the proposed host-prioritized request scheduling, extra latency is needed for read operation from the host, which degrades the performance. The tolling-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 the NVDIMM based approach is that we can use off-the-shelf NVDIMM to deal with the issue of inter-rank communication without occupation of the memory channel and it provides very good performance.

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

4.4.4 Energy Breakdown

The energy breakdown for MEDAL is in Fig. 4.9. For all designs, computation consumes less than 1.0% energy. Because DNA seeding only involves simple integer operations, the customized lightweight logic is more efficient. As for communication, it consumes 10.0% energy at most, which means the proposed communication mechanisms are energy-efficient. For the polling-based design, the interrupt-based design, and the 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 logic and communication mechanisms. For the NVDIMM-based approach, NVM consumes 48.2% energy on average. The portion of energy consumed by NVM increases with the size of databases, because the larger the database, the higher the possibility that memory requests go to NVM.

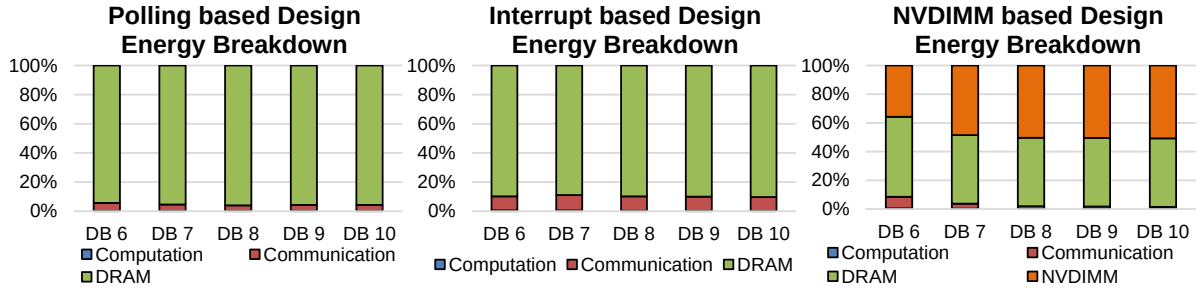

Figure 4.9: Energy breakdown for MEDAL with different communication designs. (© 2019 IEEE)

4.4.5 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. 4.10, on average, MEDAL has the highest memory bandwidth utilization - 82.81%, while Chameleon has the lowest bandwidth ratio - only 10.29%.

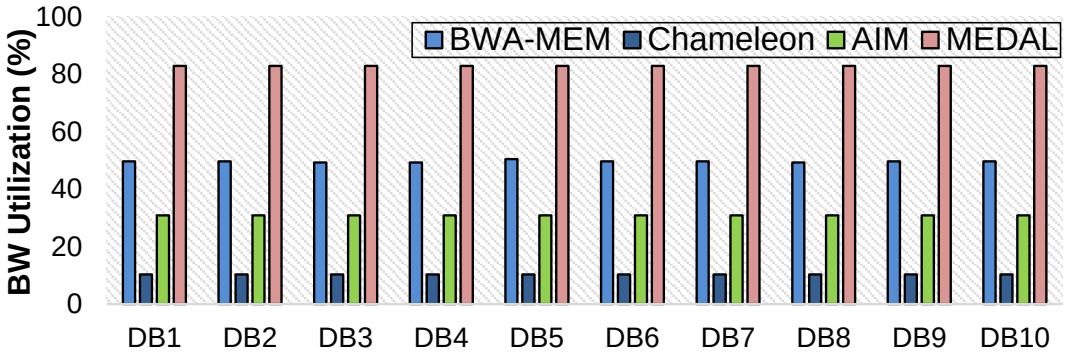

Figure 4.10: Memory bandwidth utilization for different accelerators. (© 2019 IEEE)

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

4.4.6 Performance of Data Compression

As shown in Fig. 4.11, the proposed 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.

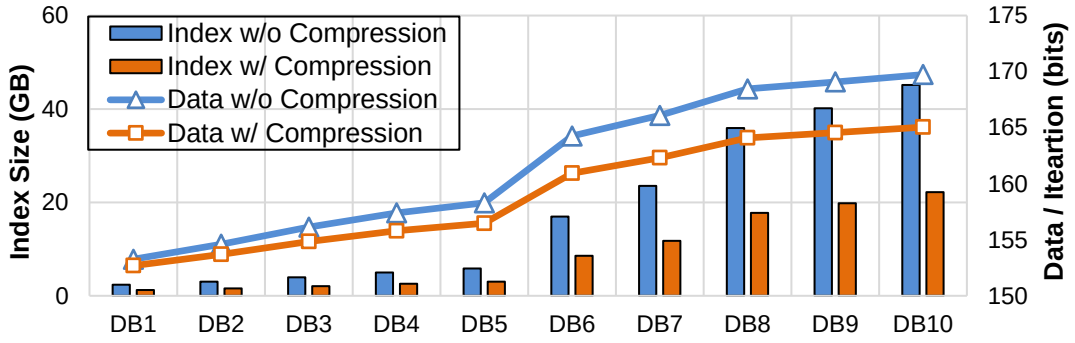

Figure 4.11: Evaluation of data compression. (© 2019 IEEE)

4.4.7 Sensitivity Study about Read Length

Reads with length of 101 are typical with the next generation sequencing technology [110, 111], 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 are in Fig. 4.12. For the intra-rank design, MEDAL with the interrupt based design provides higher performance 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 [14], we can change the customized logic inside the DBs to match the algorithms. We expect similar performance gain, since the seeding of ultra-long read is also memory bound, which MEDAL is good at.

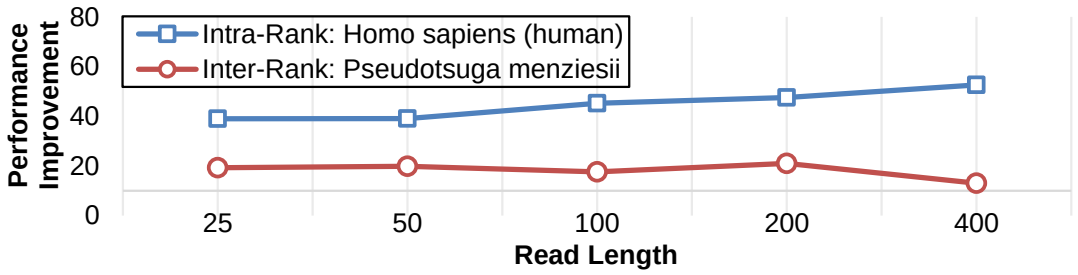

Figure 4.12: Sensitivity study about the read length. (© 2019 IEEE)

4.4.8 Hash-index based DNA Seeding Algorithm

In addition to the FM-index based DNA seeding, MEDAL also has good performance for the Hash-index based DNA seeding. The experimental results for the Hash-index based DNA seeding are shown in Fig. 4.13 and Fig. 4.14. 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 the above platforms by 668.95x, 2.22x, and 2.27x, respectively.

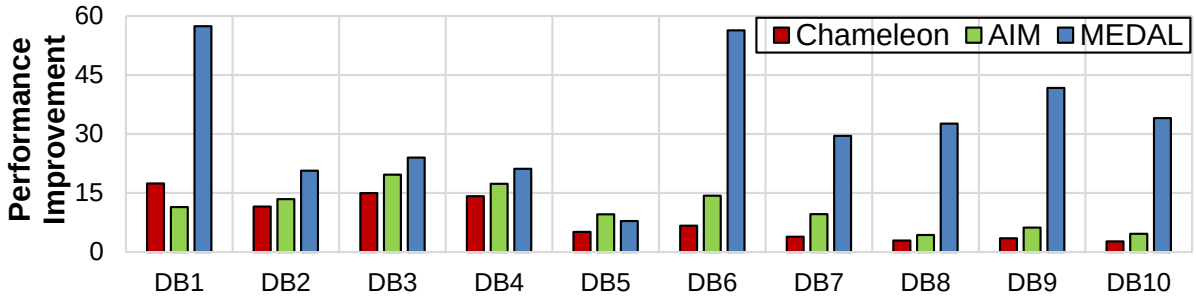

Figure 4.13: Performance improvement for Hash-index based DNA seeding. (© 2019 IEEE)

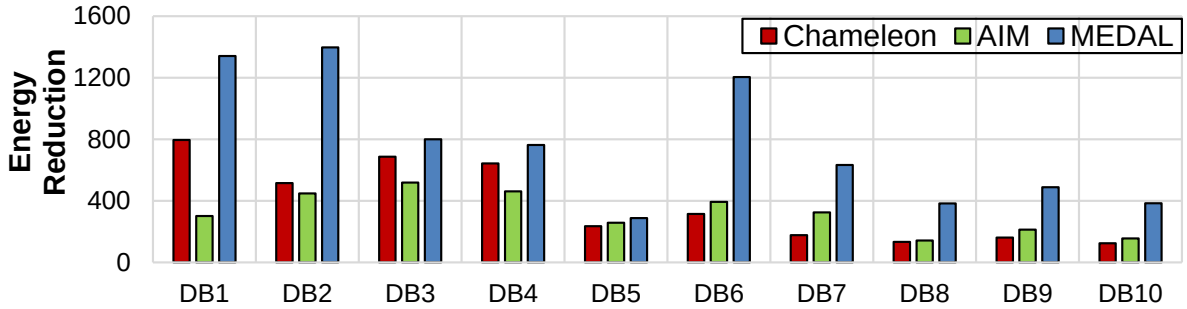

Figure 4.14: Energy reduction for Hash-index based DNA seeding. (© 2019 IEEE)

4.5 Conclusion

To accelerate DNA seeding efficiently, we propose MEDAL, a high-performance, energy-efficient, scalable, practical, and cost-effective NDP accelerator 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 challenges of fine-grained random memory access and inter-task divergence, 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 the three proposed designs, on average, MEDAL can achieve 30.50x/8.37x/3.43x performance improvement 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.

Chapter 5

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

[1] Motivated by its importance, many computation-centric approaches, such as multi-core [16, 17], GPU [19, 20], and FPGA [19, 25] have been explored to accelerate k-mer counting. However, k-mer counting involves a large amount of fine-grained memory access and is memory-bound [19, 32]. Conventional computation-centric architectures cannot address the memory bottleneck in k-mer counting, because they provide limited memory bandwidth and there is no optimization for fine-grained random memory access. Because the Near-Data-Processing (NDP) architectures architectures can provide higher memory bandwidth and reduce data movement by integrating computation and memory closer, the NDP architectures also have been leveraged to accelerate k-mer counting. For example, monolithic 3D integration is utilized to accelerate k-mer counting in [32]

[1] © 2020 IEEE. Reprinted, with permission, from Wenqin Huangfu, Krishna T. Malladi, Shuangchen Li, Peng Gu, Yuan Xie. "NEST: DIMM based Near-Data-Processing Accelerator for K-mer Counting." IEEE/ACM International Conference On Computer Aided Design (ICCAD), 2020

and MEDAL provides a more practical approach via leveraging the Dual-Inline Memory Module (DIMM) to build accelerators with the commercially available DRAM chips [11].

However, the 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, these 3D integration based architectures have no optimizations 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. According to the experiments, for more than 60% percent of the time, the Processing Elements (PEs) in MEDAL are idle due to the communication. Moreover, workload balance is a serious challenge in MEDAL and there is no optimization to deal with redundant memory access for k-mer counting.

The goal of this chapter is to address the challenges of performing k-mer counting with NDP architecture. Modified and optimized on the base of MEDAL, a practical, scalable, and energy-efficient 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 chapter 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 chips.
- From the software perspective, we propose an architecture-specific k-mer counting algorithm and a dedicated workflow for NEST, enabling parallel processing and reducing unnecessary inter-DIMM communication.
- About the optimizations, we enhance the 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 the proposed techniques. The 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.

5.1 Architecture

NEST is built by modifying the LRDIMM, as shown in Fig. 5.1 (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:

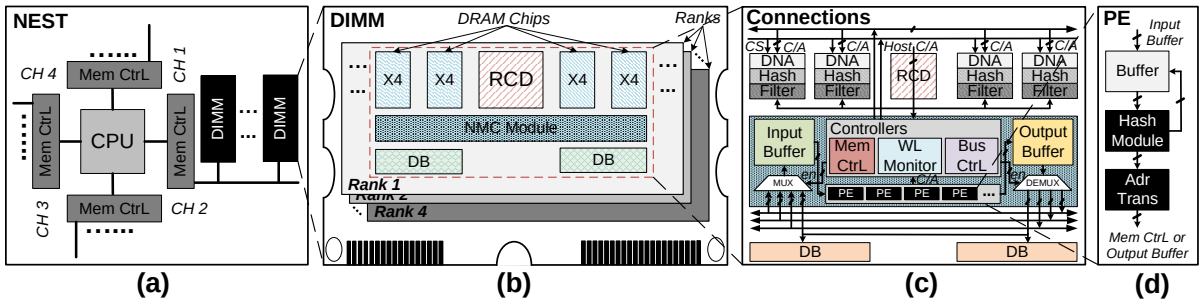

Figure 5.1: (a) High-level architecture. (b) Architecture of LRDIMM. (c) Micro-architecture within a DRAM rank. (d) PEs. (© 2020 IEEE)

NMC Module: Different from MEDAL, which modifies the DBs in LRDIMM and inserts customized computing logic into them, we attach a NMC module to each rank in the LRDIMM, as shown in Fig. 5.1 (b) and (c). The controllers and computing

logic in NEST are centralized inside the NMC module. Compared with the approach of distributing customized logic in MEDAL, centralization of the customized logic provides better communication and synchronization. Further, centralization of the customized logic enable task scheduling and improves the ability of memory access management, which are introduced in Section 5.3.

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 5.5.6. To address the issue of communication, we design the 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:

• *rank-rank* C/A bus: To transfer the C/A signals between different ranks within the same DIMM.

• *rank-rank data* bus: To 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 is the communication bottleneck in the previous work.

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

Processing Elements (PEs): As shown in Fig. 5.1 (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 [112]. Each PE contains:

• Buffer to store the input k-mers,

• Lightweight logic 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 5.3.2.

Data Direct Multiplexer: As shown in Fig. 5.1 (c), NEST connects a multiplexer with the Input Buffer and a multiplexer with the Output Buffer. With these two multiplexers, in addition to receiving/sending data to the DDR bus, the buffers can receive/send data to the 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 the PEs, we add a MC into the NMC module, as shown in Fig. 5.1 (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 logic 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 5.3.3.

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.

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.

From the hardware perspective, compared with MEDAL, NEST provides better communication/synchronization, enables task scheduling, improves the ability of memory ac-

cess management, enhances intra-DIMM communication, and has the potential to tackle the challenge of workload balance.

5.2 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 2.1, 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 under-utilizes 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, bringing significant performance overhead (please refer to Section 5.5.6 for details).

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, the available memory bandwidth and PE parallelism is fully leveraged. Furthermore, the 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, these 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 four times in the entire dataset.

To address the challenges above, we leverage the counting Bloom filter and propose an architecture-specific k-mer counting algorithm for NEST:

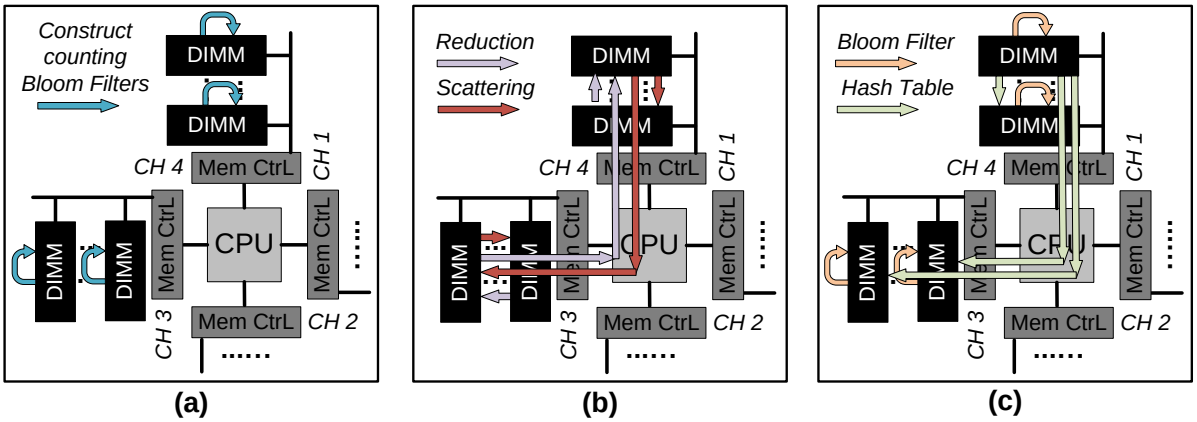

Figure 5.2: (a) Different DIMMs construct the counting Bloom filters in parallel. Memory accesses are localized. (b) Reduction and scattering of the counting Bloom filters. (c) Count k-mers. The merged Bloom filters are localized inside each DIMM. The hash table is distributed among DIMMs. Only verified non-unique k-mers involve memory access to the hash table. (© 2020 IEEE)

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 their sub-dataset.

2. Merge the Counting Bloom Filters: After different DIMMs finish constructing their local counting Bloom filters, NEST merges these 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 these 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 is set as one, otherwise it is set to zero. Scattering of the merged Bloom filter is performed by distributing the merged Bloom filter to all the DIMMs.

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 the proposed algorithm in NEST is shown in Fig. 5.2. 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 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.

5.3 Challenges and Optimizations

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

5.3.1 Bottleneck of Communication

Inter-DIMM communication becomes the bottleneck, if MEDAL is used for k-mer counting (please refer to details in Section 5.5.6). In order to ensure no hardware modification is made to the host-side memory controller and maintain the DDR timing constraints, worse case timing scenario is considered for the inter-DIMM memory access, which means there is an extra delay for each inter-DIMM memory access [11]. Due to this reason, inter-DIMM memory access involves significant performance penalties in DIMM based NDP architecture.

To address the 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: The proposed workflow greatly reduces unnecessary inter-DIMM memory access by dividing k-mer counting into multiple steps and localizing data 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.

5.3.2 Bandwidth Utilization

For address mapping in commercial DRAM chips based NDP architecture, as shown in Fig 5.3 (a), MEDAL coalesces data within a DRAM chip to better leverage data locality. However, in the 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 5.3 (b), compared with MEDAL, we re-order the address bits to prioritize distributing data in different

DRAM chips. With the 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.

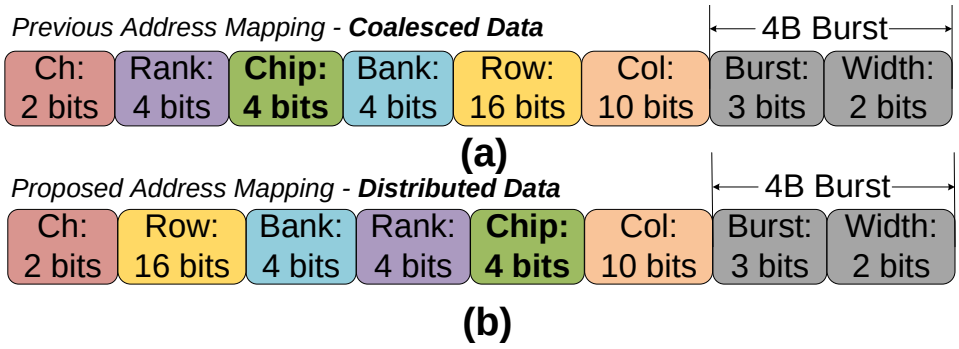

Figure 5.3: (a) The previous address mapping scheme aggregates fine-grained data to better leverage locality. (b) The proposed address mapping scheme distributes fine-grained data for better memory bandwidth utilization. (© 2020 IEEE)

5.3.3 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 are 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 dispatches tasks to this PE to keep it busy. The challenge of workload balance is addressed with the proposed task scheduling via dispatching tasks to PEs in fine-granularity dynamically.

5.3.4 Redundant Memory Access

During the 'Count k-mer' step, we need to verify if all Bloom filter entries related to the current k-mer in the merged Bloom filter are ones. If all of these 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 5.4 (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 are fetched out from the DRAM and memory bandwidth is wasted. To address this issue, we propose the following optimizations to eliminate the redundant memory accesses:

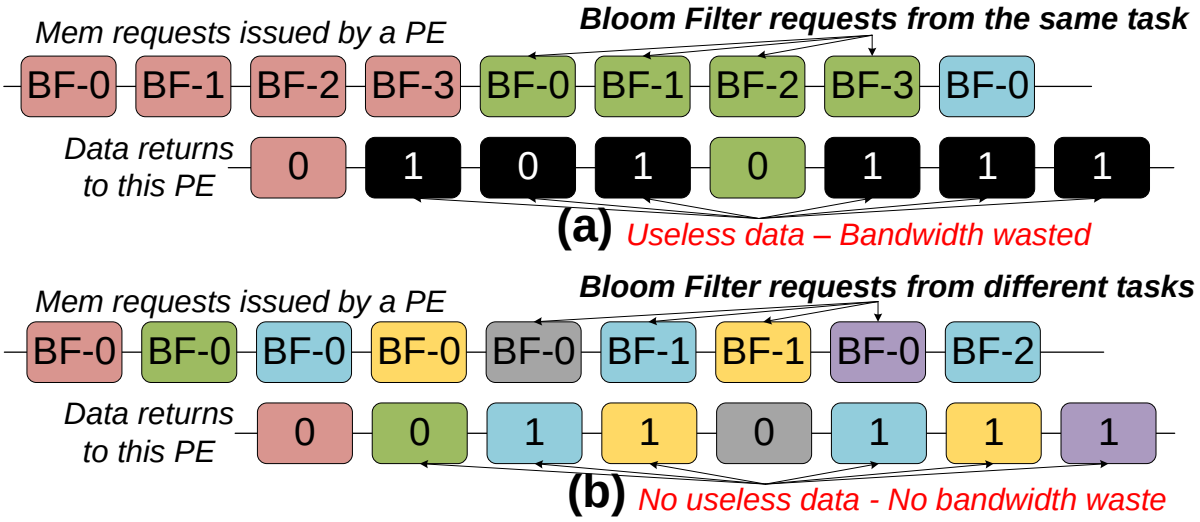

Figure 5.4: (a) Memory bandwidth is wasted without the proposed optimizations. (b) No memory bandwidth is wasted with the proposed optimizations. (© 2020 IEEE)

Scattered Memory Access: As shown in Fig 5.4 (b), instead of issuing memory accesses belong to a k-mer sequentially, we scatter these memory accesses and issue them

with time intervals. We issue another memory access, only if the previous memory access related to the same 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 the redundant memory access is eliminated with the 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 switch to another task and issue a memory access belong to another k-mer after issuing the previous memory access belong to a certain k-mer. With this approach, time intervals due to scattered memory accesses are filled with memory accesses belong to different k-mers.

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

5.4 Discussion

Algorithm Equivalence: The 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 [50]. Second, in general, retaining k-mers with low occurrences doesn't degrade the final results of the following applications after k-mer counting [40].

Generality of NEST: From the 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

Table 5.1: Configure of the CPU and CPU + GPU baselines (© 2020 IEEE)

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

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 the software perspective, the 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.

5.5 Experimental Results

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

5.5.1 Experimental Setup

Configuration of the Baselines: For CPU and CPU + GPU, we use two widely used software tools, i.e., BFCounter [40] and Gerbil [20], as the baselines. The detailed configuration of the two servers running these two baselines is shown in Table 5.1.

Table 5.2: Configure of MEDAL and NEST (© 2020 IEEE)

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

For MEDAL, as shown in Table 5.2, the configuration of memory and number of PEs are the same as these in NEST. The differences between MEDAL and NEST are these architecture modifications and communication optimizations we make. In addition, the proposed k-mer counting algorithm is also used in MEDAL to improve its performance,

Configuration of NEST: We modify Ramulator [102] to build a cycle-accurate simulator for NEST. The configuration of NEST is shown in Table 5.2. We use pre-layout Design Compiler [103] with 28 nm technology [104] 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 5.3. The timing parameters of the DRAM components are shown in Table 5.2. The energy consumption of DRAM is derived by feeding the memory traces from Ramulator to DRAMPower [105]. The parameters of energy consumption for the datapath used in this chapter are from CACTI-IO [106].

Datasets: The datasets used in the experiments are sequenced human genome [109] with different coverage ratio.

Table 5.3: Design Parameters of the Lightweight logics (© 2020 IEEE)

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

5.5.2 Performance Improvement

The performance of different architectures/optimizations is in Fig. 5.5 (a) and (b). All the data are normalized to the performance of the 48-thread CPU baseline.

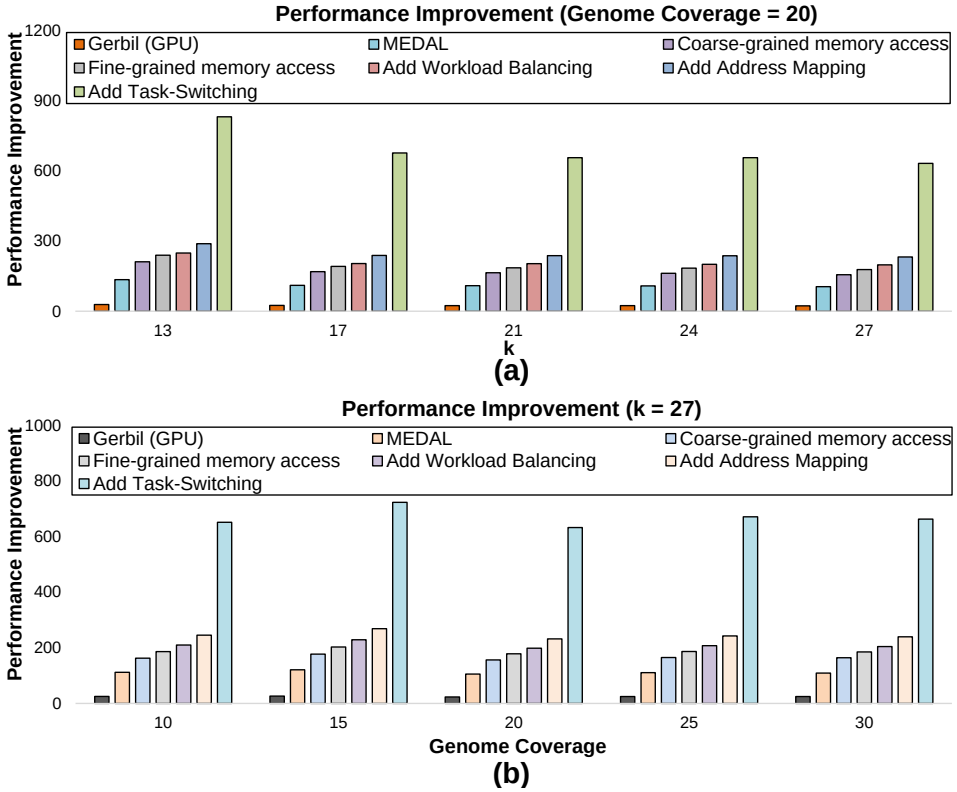

Figure 5.5: Performance improvement of the CPU/GPU hybrid approach, MEDAL, and NEST with different architectures/optimizations. (a) Performance improvement for different k. (b) Performance improvement for different genome coverage. (© 2020 IEEE)

As shown in Fig. 5.5 (a), when the genome coverage is 20x, for different k, the 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 the naive coarse-grained memory access, the naive fine-grained memory boosts the performance of NEST by 1.13x. About the optimizations, the proposed address mapping improves the performance of the naive fine-grained memory access by 1.08x. Moreover, performance improvement of 1.17x is achieved with the proposed task scheduling. Memory access management gains 2.79x performance improvement. Combine the 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. 5.5 (b).

Compared with MEDAL, performance improvement from the configuration with naive coarse-grained memory access in NEST comes from the enhanced support for intra-DIMM communication. Compared with the naive coarse-grained memory access, performance improvement of the 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 the 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 these benefits together, NEST provides 512x more memory bandwidth than the CPU baseline. Further, the 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.

5.5.3 Energy Reduction

The energy reduction of different architectures/optimizations is in Fig. 5.6 (a) and (b). All the data are normalized to that of the 48-thread CPU baseline.

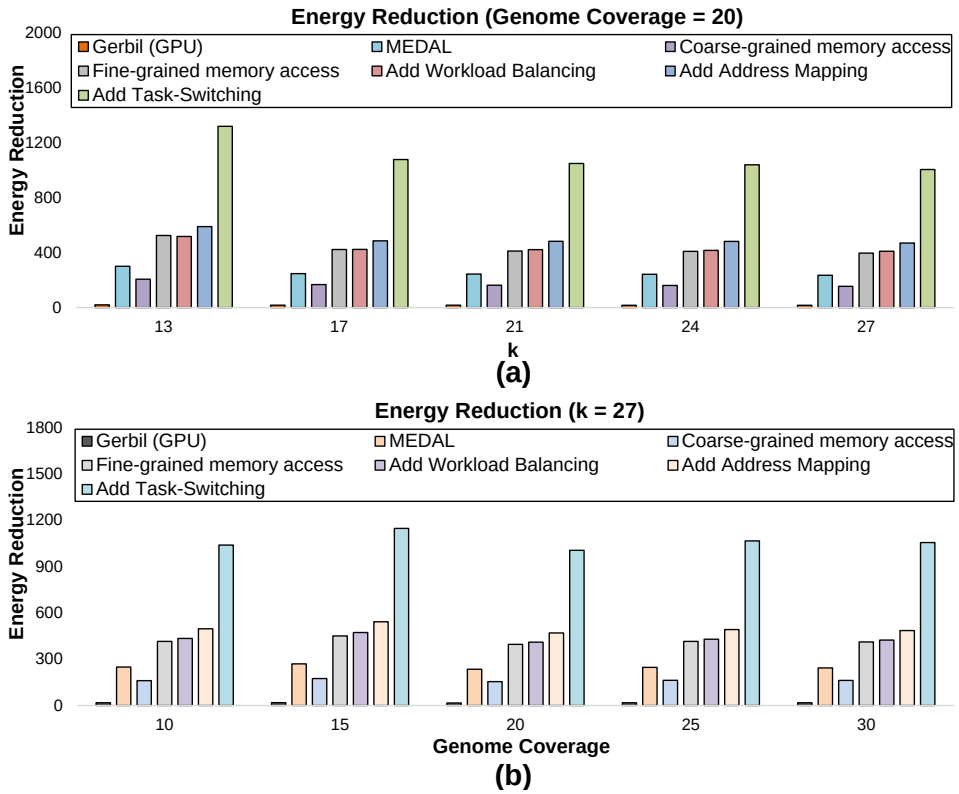

Figure 5.6: Energy reduction of the CPU/GPU hybrid approach, MEDAL, and NEST with different architectures/optimizations. (a) Energy reduction for different k. (b) Energy reduction for different genome coverage. (© 2020 IEEE)

As shown in Fig. 5.6 (b), when the genome coverage is 20x, for different k, the 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 the naive coarse-grained memory access, energy consumption is reduced by 2.54x with the naive fine-grained memory reduces. About the 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 the 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. 5.6 (b).

5.5.4 Time and Energy Breakdown

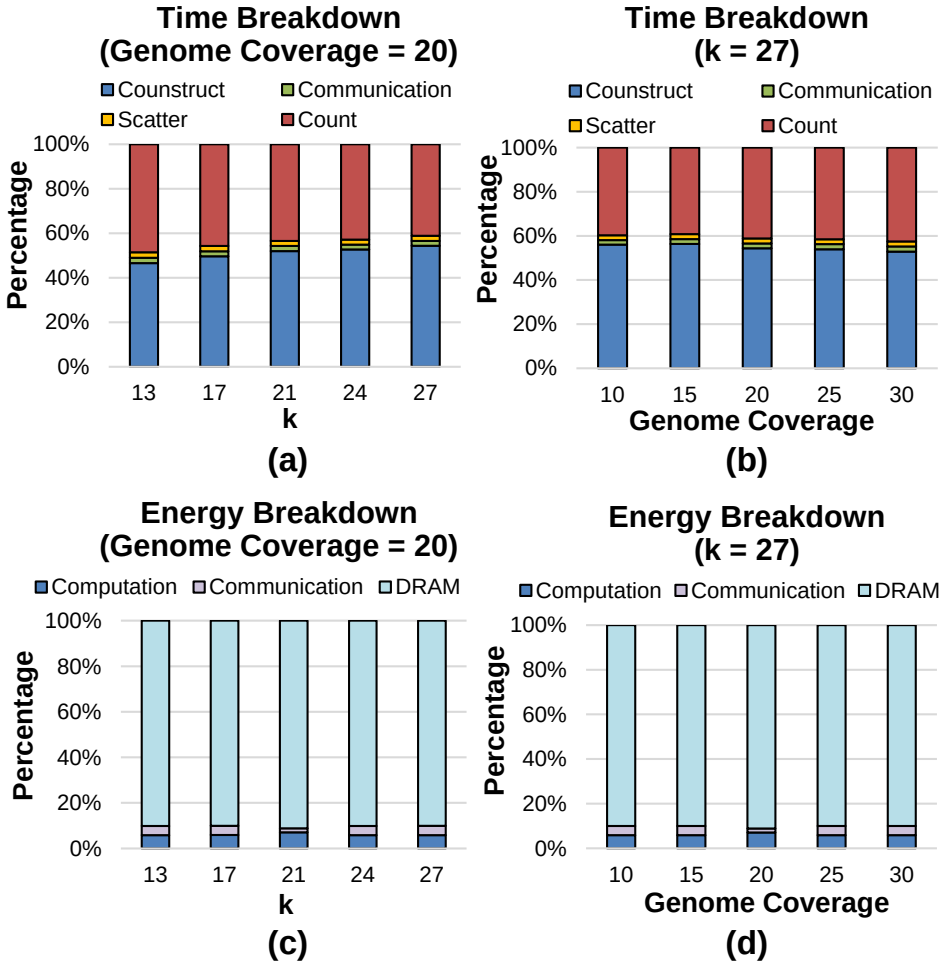

Figure 5.7: (a) Time breakdown with 20x genome coverage. (b) Time breakdown with $k = 20$. (c) Energy breakdown with 20x genome coverage. (d) Energy breakdown with $k = 20$. (© 2020 IEEE)

The time breakdown for NEST is shown in Fig. 5.7 (a) and (b). 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', the proposed workflow separates the phases of 'Construct Counting Bloom Filters' and 'Count k-mers' to reduce inter-DIMM communication, leading to performance improvement.

The energy breakdown of NEST is shown in Fig. 5.7 (c) 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.

5.5.5 Remote Memory Access

The percent of remote memory access among all memory access with the naive implementation of the original k-mer counting algorithm in the MEDAL architecture and the percent of remote memory access in NEST are shown in Fig. 5.8. 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.

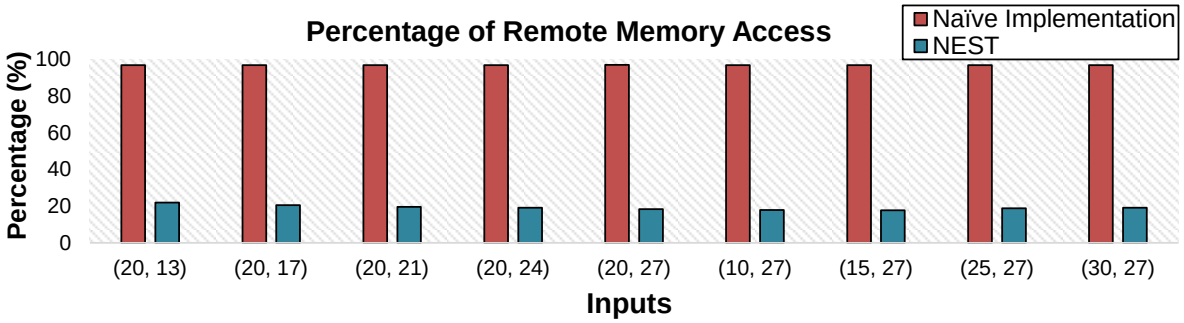

Figure 5.8: Percentage of remote memory access in the naive implementation of the original k-mer counting algorithm and in NEST. (© 2020 IEEE)

5.5.6 PE Utilization

The breakdown of different PE states for the phases of 'Construct Counting Bloom filters' and 'Count k-mers' are shown in Fig. 5.9 (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. The proposed step-by-step optimizations tackle 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 the 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.

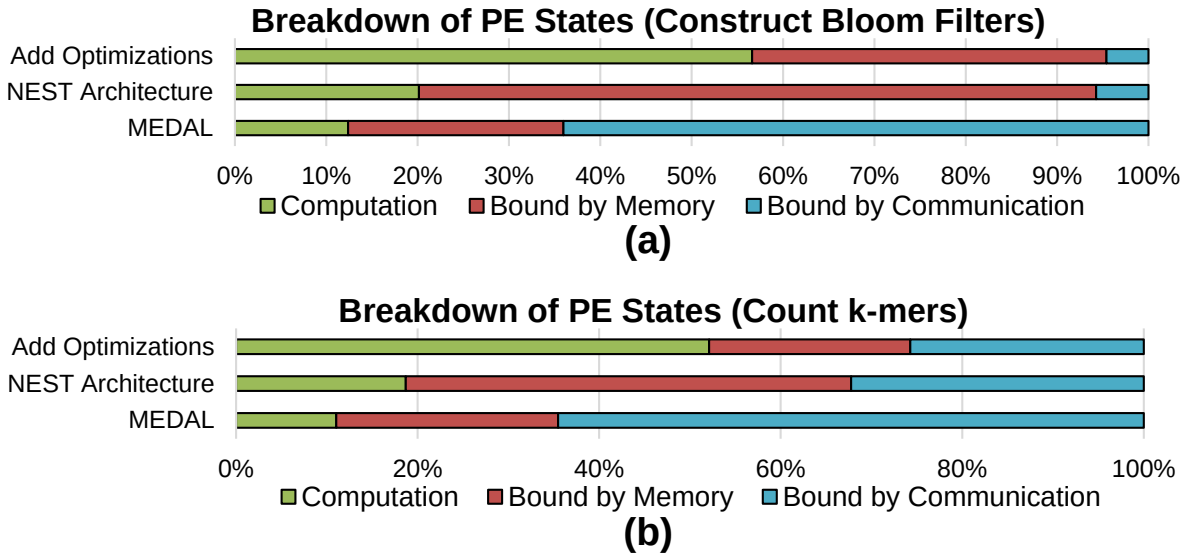

Figure 5.9: Breakdown of the PEs states. (a) Constructing counting Bloom Filters. (b) Counting k-mers. (© 2020 IEEE)

5.6 Conclusion

This chapter proposes NEST, a practical, scalable, high-performance, and energy-efficient NDP accelerator for k-mer counting. To fully unleash the performance of NEST,

we propose an architecture-specific k-mer counting algorithm for NEST, together with a dedicated workflow, to reduce unnecessary inter-DIMM communication and improve parallelism. In addition, we propose a novel address mapping scheme to improve memory bandwidth utilization. The challenge of workload balance is addressed with the proposed task scheduling. Scattered memory access and task-switching are proposed to eliminate the redundant memory access. Experimental results demonstrate 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.

Chapter 6

BEACON: Scalable Near-Data-Processing Accelerators for Genome Analysis near Memory Pool with the CXL Support

[1] Due to the importance and the time-consuming fact of genome analysis, researchers are paying more and more attention to its hardware acceleration. Because of the large amount of data involved, the simple arithmetic operations, and the memory-bound feature, many applications in genome analysis are well-suited for the memory-centric architectures [56], i.e., Processing-In-Memory (PIM) and Near-Data-Processing (NDP). Different PIM and NDP approaches, including ReRAM [15, 113], HMC [35, 50], and Dual-Inline-Memory-Module (DIMM) [4, 11], have been explored to accelerate the memory-

[1] © 2022 IEEE. Reprinted, with permission, from Wenqin Huangfu, Krishna T. Malladi, Andrew Chang, Yuan Xie. "BEACON: Scalable Near-Data-Processing Accelerators for Genome Analysis near Memory Pool with the CXL Support." Proceedings of the 55th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO), 2022.

bound applications in genome analysis. Among these different PIM and NDP work, the DIMM based designs stand out to be highly promising, because they are more practical and cost-effective.

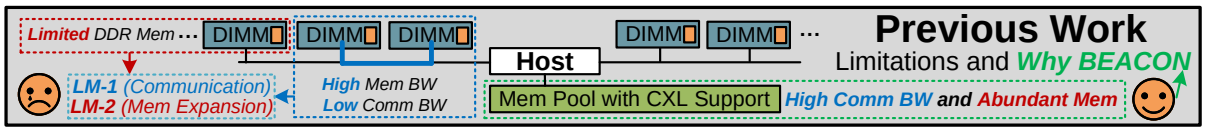

Figure 6.1: Architecture and limitations of the previous work, together with the motivations for designing BEACON. (© 2022 IEEE)

However, as shown in Fig. 6.1, the previous DIMM based accelerators are built upon the DDR-DIMMs attached to the host and rely on the DDR memory channel for the inter-DIMM communication. These DDR-DIMM based designs have two critical limitations: First, communication has become the performance bottleneck. As shown in Fig. 6.1, the large gap, e.g., 12x in MEDAL [11], between the intra-DIMM memory bandwidth and the inter-DIMM communication bandwidth seriously degrades the performance [4, 11]. Second, the potential for memory expansion is limited. The memory capacity requirements for different applications in genome analysis vary significantly, depending on the specific application, the dataset, the algorithm to use, and the parameters of the algorithm. For example, BWA-MEM uses 64GB memory for FM-index based DNA seeding [42] and SMUFIN uses near 2TB memory for k-mer counting [19]. Ideally, it's desired to be able to conveniently adjust the memory capacity of the DIMM based accelerators with unmodified DIMMs on-demand to deal with different usage scenarios for genome analysis [4, 11]. Unfortunately, as shown in Fig. 6.1, the constraints of the DDR-DIMMs, e.g., the limited number of memory channels and slots [114, 115], and the trend of memory dis-aggregation, i.e., migrating local DDR memory to a memory pool, greatly reduces the potential of memory expansion in the previous work. For example, NEST provides 512GB memory in the DDR memory channel [4], which cannot meet the

2TB memory requirement for SMUFIN [19]. As a comparison, the memory pool easily provides 4.5TB memory and can scale-out far beyond this [116]. In addition, memory expansion with unmodified DIMMs totally contradicts with the design philosophy of the previous DDR-DIMM based accelerators, i.e., modifying the DIMM to reduce the expensive inter-DIMM communication and leverage the high intra-DIMM memory bandwidth with intra-DIMM data manipulation enabled by DIMM-customization.

*The **goal** of this chapter is to build DIMM based accelerators for genome analysis under the scenario of memory dis-aggregation, supporting efficient on-demand memory expansion with unmodified DIMMs and eliminating the performance bottleneck of communication.* To this end, we propose BEACON, i.e., CXL-DIMM based NDP accelerators for genome analysis located near the dis-aggregated memory pool with the CXL support. BEACON enables four key features: First, it embraces the emerging trend-of memory dis-aggregation [117–119] with CXL as one of the most promising enabler [114,120,121]. Second, BEACON leverages the abundant memory in the memory pool to enable on-demand memory expansion with unmodified CXL-DIMMs. Third, BEACON fully leverages the high communication bandwidth provided by CXL to address the challenge of communication. Fourth, BEACON maintains the promising and non-invasive feature, i.e., no modifications to the cost-sensitive DRAM dies, of the previous DIMM based designs. BEACON provides two design choices, i.e., BEACON-D and BEACON-S. BEACON-D and BEACON-S perform the computation within the enhanced CXL-DIMMs and enhanced CXL-Switches, respectively.

The architecture of BEACON is designed to set the foundation for efficient memory expansion and communication by reducing data movement and leveraging the high bandwidth provided by CXL. Based on the BEACON architecture, the memory management framework is proposed to enable memory expansion with unmodified CXL-DIMMs and optimize communication by improving data locality. In addition, algorithm-specific

optimizations are proposed to further boost the performance of BEACON. Combining the architecture design, the memory management framework, and the algorithm-specific optimizations together, BEACON enables efficient on-demand memory expansion with unmodified CXL-DIMMs and eliminates the performance bottleneck of communication. The main contributions of this chapter are listed as follows:

- We propose BEACON, including BEACON-D and BEACON-S, under the scenario of memory dis-aggregation. Located near the memory pool and focusing on genome analysis, BEACON leverages the abundant memory within the memory pool and the high communication bandwidth provided by CXL without making any modification to the cost-sensitive DRAM dies.

- With the proposed architecture design (e.g., processing in the memory pool and in-switch data routing) and memory management framework (e.g., memory allocation and address mapping), BEACON enables efficient on-demand memory expansion with unmodified CXL-DIMMs and eliminates the performance bottleneck of communication of the previous work.

- We adopt algorithm-specific optimizations (multi-chip coalescing for FM-index based DNA seeding and single-pass k-mer counting) to improve the performance. In addition, BEACON can be used for multiple applications in genome analysis.

- Experiments are performed to demonstrate the performance benefits of BEACON and different optimizations. Overall, compared with state-of-the-art DIMM based NDP accelerators, BEACON-D and BEACON-S improves the performance by 4.70x and 4.13 on average

6.1 Background

This section introduces the background of this work, including memory dis-aggregation and the CXL.

Memory Dis-aggregation Memory dis-aggregation include memory expansion and memory pooling [115, 122, 123]. Memory expansion refers to the process of enlarging the memory capacity to improve the memory bandwidth per core [114, 123]. Memory pooling addresses the issue of resource fragmentation, i.e., resource under-utilization due to the mismatched requirements of different workloads [119], by providing a shared memory pool that can be accessed on-demand for different servers. Memory dis-aggregation is becoming a trend [117–119], because it can improve system performance and resource utilization as well as reduce cost. Many different types of interconnect technologies have been proposed for memory dis-aggregation, including OpenCAPI [124], GenZ [125], FMC Cable [118], Optical memory channel [126], CXL [120, 121], and so on.

Compute Express Link (CXL) As an open industry standard interconnect, CXL offers high-bandwidth and low-latency connectivity between the host processor and devices such as smart I/O devices, accelerators, and memory buffers [127]. CXL enables cache coherence and memory semantics for heterogeneous processing and memory systems based on the PCI Express (PCIe) 5.0 I/O semantics for optimized performance in evolving usage models [120, 121]. CXL supports memory dis-aggregation, including memory expansion and memory pooling [114, 122, 123, 128]. In fact, CXL is one of the most promising technology in enabling memory dis-aggregation. For example, the full-stack memory dis-aggregation design based on CXL has been proposed [128] and the Intel CPU Memory Management Unit (MMU) are going to integrate the CXL Memory Expander in the future [114].

The architecture of the naïve memory pool with the CXL support is: The host is connected to multiple CXL-Switches via the CXL buses and each CXL-Switch is connected with multiple CXL-DIMMs via the CXL buses.

6.2 Motivations

This section introduces the limitations of the previous work and the reasons for our design choice.

Limitation-1 (Communication): The large gap between the intra-DIMM memory bandwidth and the inter-DIMM communication bandwidth makes inter-DIMM communication the performance bottleneck for the previous DDR-DIMM based accelerators. For example, by leveraging the rank-level memory bandwidth within the DDR-DIMMs, the intra-DIMM memory bandwidth is 12x higher than the inter-DIMM communication bandwidth in MEDAL [11].

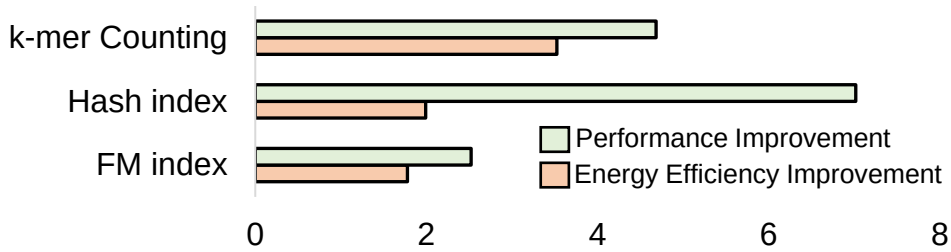

Figure 6.2: Improvement of performance and energy efficiency for the previous DDR-DIMM based accelerators with imaginary idealized communication, i.e., infinite bandwidth and zero latency. The experimental configuration is in Section 6.5.1. (© 2022 IEEE)

According to the experiments, as shown in Fig. 6.2, with imaginary idealized communication (infinite bandwidth and zero latency, i.e., instant data delivery), on average, performance improvement of 4.36x and energy efficiency improvement of 2.32x can be achieved for the the previous DDR-DIMM based accelerators. The data indicate that the bottleneck of communication greatly degrades the performance and energy efficiency.

Limitation-2 (Memory Expansion): As we've mentioned in Section 6, the ability of memory expansion with unmodified DIMMs is desired for accelerators of genome analysis to deal with different usage scenarios. Unfortunately, the potential of memory expansion in the previous DIMM based accelerators is greatly limited due to three reasons: First, the previous work are based on the DDR-DIMMs, which have poor scalability due to many constraints, e.g., the limited number of memory channels and slots [114, 115]. Second, memory dis-aggregation, which further reduces the amount of local DDR memory, is becoming a trend [118,119]. Third, since the inter-DIMM communication is the performance bottleneck, the design philosophy of the previous work is to reduce the expensive inter-DIMM communication and leverage the high intra-DIMM bandwidth by intra-DIMM data manipulation enabled with DIMM-customization. Leveraging unmodified DIMMs (i.e., no intra-DIMM data manipulation) as memory expansion means frequently accessing data from the remote unmodified DIMMs (i.e., no intra-DIMM bandwidth) and bring the data back (i.e., frequent inter-DIMM communication), which totally contradicts with the design philosophy of the previous work.

To tackle these two limitations and maintain the non-invasive feature of the previous DIMM based designs, we embrace the trend of memory dis-aggregation to design novel DIMM based accelerators for genome analysis. Located near the memory pool with the CXL support, BEACON fully leverages the abundant memory in the memory pool and the high communication bandwidth of CXL to address the challenges related to memory expansion and communication.

6.3 BEACON Design

This section presents the BEACON architecture. After an overview of the architecture, we describe the components design, the memory management framework, and the algorithm-specific optimizations in order.

6.3.1 Architecture Overview

The goal of BEACON is to build DIMM based accelerators for genome analysis under the scenario of memory dis-aggregation, supporting efficient on-demand memory expansion with unmodified DIMMs and eliminating the performance bottleneck of communication. To this end, BEACON resides in the memory pool with the CXL support to leverage the abundant memory within the memory pool and the high communication bandwidth provided by CXL.

As shown in Fig. 6.3, BEACON provides two different design choices, i.e., BEACON-D and and BEACON-S. The key difference between BEACON-D and BEACON-S is: BEACON-D is a Processing-In-DIMM accelerator and performs the computation within the enhanced CXL-DIMMs. As a comparison, BEACON-S is a Processing-In-Switch accelerator and performs the computation within the enhanced CXL-Switches. Both BEACON-D and BEACON-S maintain the non-invasive feature and practicality of the previous DIMM based accelerators for genome analysis without making any modification to the cost-sensitive DRAM dies.

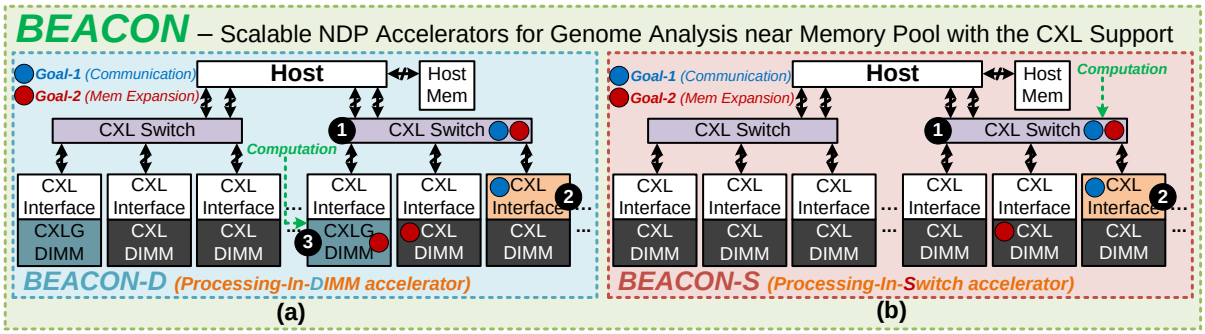

Figure 6.3: High-level architecture of BEACON. (a) BEACON-D. (b) BEACON-S. (© 2022 IEEE)

BEACON-D have better performance than BEACON-S in some applications with its customized CXL-Genome-DIMM (CXLG-DIMM). The CXLG-DIMMs are computation and fine-grained memory access enabled CXL-DIMMs. CXLG-DIMMs have the

ability to provide fine-grained memory access, which is useful in genome analysis [4, 11]. On the other hand, BEACON-S provides overall comparable performance, i.e., 87.87% performance of BEACON-D, with less hardware modifications.

BEACON-D: For BEACON-D, as shown in Fig. 6.3 (a), three components are modified in the naïve memory pool with the CXL support, i.e., the CXL-Switch (❶), the CXL-Interface (❷), and the CXLG-DIMM (❸). Modifications to the CXLG-DIMM are done to the Printed Circuit Board (PCB) board of the CXL-DIMM, leaving the cost-sensitive DRAM dies untouched.

From the architecture perspective, besides performing computation within the CXLG-DIMM, keep our design goal in mind, as shown in Fig. 6.3 (a), the components labeled with the blue and red circles aim to achieve the goal of supporting efficient communication and providing efficient memory access, respectively. BEACON-D achieves efficient communication by transferring data via the high-bandwidth CXL buses with the architecture support from the CXL-Switch and the CXL-Interface. For memory expansion, as shown in Fig. 6.3 (a), different CXLG-DIMMs can access data within each other and can also access data in the unmodified CXL-DIMMs. The CXL-Switch helps with the memory management.

BEACON-S: For BEACON-S, as shown in Fig. 6.3 (b), the CXL-Switch (❶) and the CXL-Interface (❷) are modified in the naïve memory pool with the CXL support.

In addition to performing computation within the CXL-Switch, similar to BEACON-D, as shown in Fig. 6.3 (b), the components labeled with the blue and red circles aim to achieve the goal of supporting efficient communication and providing efficient memory access, respectively. To achieve our design goal, the CXL-Switch and the CXL-Interface work together to fully leverage the potential from the high-bandwidth CXL bus to eliminate the performance bottleneck of communication. For memory expansion, the unmodified CXL-DIMMs are regulated and can be accessed by the CXL-Switch efficiently.

6.3.2 Components Design

The architectures of the modified components in the CXL based memory pool, i.e., the CXLG-DIMM, the CXL-Interface, and the CXL-Switch, are shown in Fig. 6.4 (a). Similarly, the components labeled with the blue and red circles aim to achieve the goal of supporting efficient communication and providing efficient memory access. As mentioned before and shown in Fig. 6.3 (a), the functionality of the CXLG-DIMM is to perform computation and the access memory efficiently. The CXL-Interface helps to fully leverage the potential of the high-bandwidth communication. As for the CXL-Switch, it's responsible for communication regulation and memory management. In BEACON-S, as shown in Fig. 6.3 (b), the CXL-Switch is also responsible for performing computation.

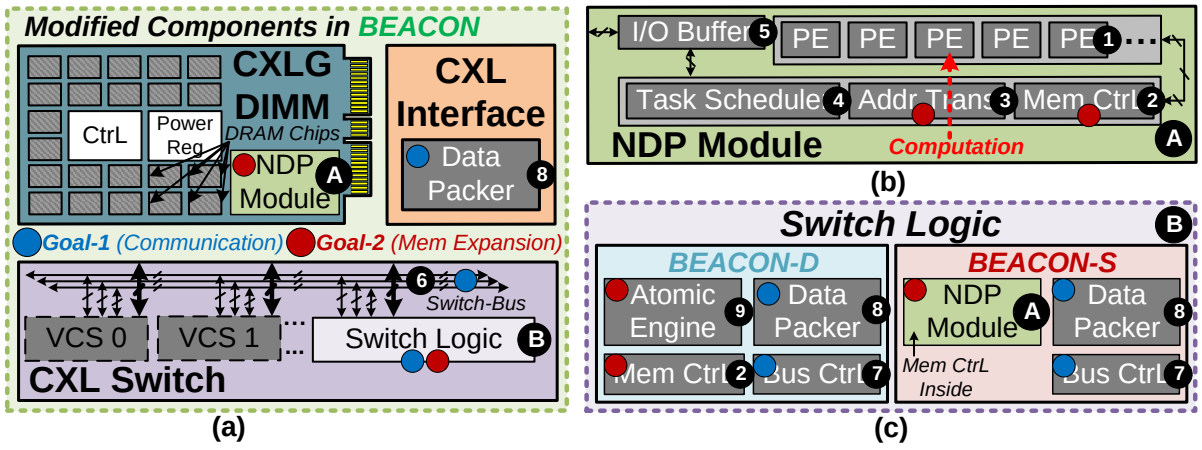

Figure 6.4: (a) The architectures of the three modified components within BEACON. (b) Architecture of the NDP module. (c) Architecture of the Switch-Logic in BEACON-D and BEACON-S. (© 2022 IEEE)

6.3.2.1 CXLG-DIMM

As shown in Fig. 6.4 (a), the NDP module (A) is added to the CXL-DIMM to build the CXLG-DIMM. In addition, similar to MEDAL [11], in order to improve utilization of memory bandwidth for genome analysis, fine-grained memory access is enabled

in CXLG-DIMM by providing individual Chip Select (CS) signals to different DRAM chips within the CXLG-DIMM. To summarize, CXLG-DIMMs are computation and fine-grained memory access enabled CXL-DIMMs. CXLG-DIMMs are the customized accelerator modules in BEACON-D.

NDP Module Design: The design goal of the NDP module are to enable computation for multiple applications in genome analysis and provide efficient memory accessibility to the other CXLG-DIMMs and CXL-DIMMs. Five different components, i.e., the PE, the Memory Controller (MC), the Address Translator, the Task Scheduler, and the I/O Buffer, are added to the NDP module for the design goal.

• ***To Enable Computation:*** To enable computation for multiple applications in genome analysis, multiple PEs are added into the NDP module:

❶ ***PEs:*** The PEs, as shown in Fig. 6.4 (b), are designed to be multi-purpose. They can be programmed to perform computation of four algorithms for three different applications in genome analysis, including DNA seeding, k-mer counting, and DNA pre-alignment. As for the inputs, tasks, i.e., DNA sequences to be processed with related information, are received from the Task Scheduler. As for the outputs, memory requests are sent to the Address Translator to derive the physical memory addresses. If operands from memory requests are needed for an active task to continue computation, the PE puts that task into the Task Scheduler and switches to process another task in its task queue.

• ***To Provide Efficient Memory Access:*** The Memory Controller (MC) and the Address Translator are added into the NDP module to support the proposed memory management and eliminate the need for help from the host-side memory MC during memory access:

❷ ***Memory Controller (MC):*** The MC in the NDP module, as shown in Fig. 6.4 (b), is responsible for the purpose of memory management in BEACON, including memory allocation/de-allocation, data placement, address mapping, memory requests scheduling,

and so on. In addition, the MCs in BEACON also enable localized memory access. For example, in BEACON-D, the MC in the NDP on the CXLG-DIMM enables localized memory access inside the CXLG-DIMM. Similary, in BEACON-S, the MC in the NDP module on the CXL-Switch enables localized memory without asking for help from the host-side MC.

❸ ***Address Translator:*** The Address Translator, as shown in Fig. 6.4 (b), receives memory requests from the PEs and translates the memory requests into their physical memory addresses according to the proposed data placement and address mapping schemes. Then, the Address Translator forwards the memory requests to the Data Packer within the Switch-Logic to transfer these memory requests towards their destinations.

• ***Other Components:*** Besides the components related to computation and memory access, the Task Scheduler and the I/O Buffer are added into the NDP module to support its functionalities and improve the efficiency:

❹ ***Task Scheduler:*** We define a task as a DNA sequence to be processed with related information, e.g, algorithm and current processing status. The Task Scheduler, as shown in Fig. 6.4 (b), maintains two queues for the inactive tasks, i.e., the incoming task queue and the out-going task queue.

The inputs to the incoming task queue are tasks waiting for operands from memory requests and are sent from the PEs. When all the operands needed for a task within the incoming task queue are ready, this task is pushed to the out-going task queue. The Task Scheduler monitors the statuses of different PEs and the tasks in the out-going task queue are assigned to the PEs that need more tasks to process.

❺ ***I/O Buffer:*** The Input Buffer, as shown in Fig. 6.4 (b), receives inputs to the NDP module, including the remote memory requests and the data back with remote/local destinations. Both the remote memory requests and the data back with remote/local destinations are first forwarded to the MC. The remote memory requests wait at the MC

to be issued out. The data back with remote destinations are forwarded to the Data Packer (introduced below within the CXL-Switch) to be packed together and transferred towards the remote destinations. The data back with local destinations are forwarded to the Task Scheduler, which means the needed operands for the tasks within the Task Scheduler have been back.

For BEACON-D, the Output Buffer in the CXLG-DIMM, as shown in Fig. 6.4 (b), forwards memory requests to the Data Packer within the CXL-Interface. For BEACON-S, the Output Buffer in the CXL-Switch forwards memory requests to the Data Packer within the CXL-Switch. These memory requests are forwarded towards their destination from the Data Packer.

6.3.2.2 CXL-Switch

To support the functionalities of the CXL-Switch, as shown in Fig. 6.4 (a), the Switch-Bus and the Switch-Logic (❽) are added to the CXL-Switch.

❻ ***Switch-Bus:*** To achieve our design goal of supporting efficient communication, as shown in Fig. 6.4 (a), the Switch-Bus is added to the CXL-Switch to support efficient in-switch data routing between different components within the same CXL-Switch, e.g, the Virtual CXL Switch (VCS) and the Switch-Logic, eliminating unnecessary data movement between the CXL-Switch and the host.

Switch-Logic Design: For BEACON-D, the design goal of the Switch-Logic is to support efficient communication and provide efficient memory accessibility to the CXL-DIMMs. To achieve this design goal, four components, i.e., the Bus Controller (Bus CtrL), the Data Packer, the MC, and the Atomic Engine, are added into the Switch-Logic for BEACON-D.

For BEACON-S, since the computation is performed within the CXL-Switch, besides the design goal above, we also need to enable computation within the Switch-Logic.

Overall, three components, i.e., the NDP module, the Bus CtrL, and the Data Packer, are added into the Switch-Logic.

• ***To Enable Computation:*** BEACON-S needs the computation capability in its Switch-Logic, we add the same NDP module placed within the CXLG-DIMM in BEACON-D into its Switch-Logic to achieve this goal.

• ***To Support Efficient Communication:*** The Bus CtrL and the Data Packer are added into the Switch-Logic to efficiently coordination in-switch data routing and improve the utilization of communication bandwidth for fine-grained data:

❼ ***Bus CtrL:*** As shown in Fig. 6.4 (c), the Bus CtrL is responsible for the coordination of communication and in-switch data routing within the CXL-Switch on the Switch-Bus.

❽ ***Data Packer:*** Applications in genome analysis involve frequent fine-grained random memory access, e.g., 32 Bytes for DNA seeding and 1 bit for k-mer counting [4,11]. However, the default data transfer granularity in the CXL is 64 Bytes, leading to movement of the useless data. The key idea to address this issue in BEACON is to discard the useless data and pack the useful data together before the data transfer. After receiving the data, the packed fine-grained data are unpacked and separated. This approach eliminates the movement of the useless data, leading to reduction in communication bandwidth and energy consumption. In Fig. 6.5 (a) and (b), an example with 16 Bytes data chunk is used to demonstrate the idea of data packing. Before data packing, the data chunk only contains 2 Bytes useful data, i.e, data 0. After data packing, useful data are packed together to eliminate the movement of useless data.

The Data Packer is added into the Switch-Logic to enable data packing. The Data Packer, as shown in Fig. 6.4 (c), packs fine-grained data together according to the format defined by the CXL protocol before the data transfer. It also unpacks and separates the packed fine-grained data coming in.

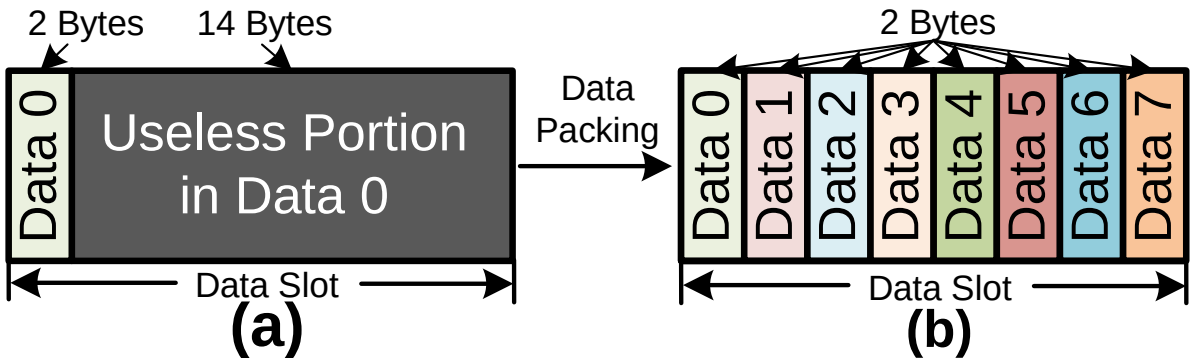

Figure 6.5: Data chunk to be transferred. (a) Before data packing. (b) After data packing. (© 2022 IEEE)

To Provide Efficient Memory Access: To provide efficient memory access to the unmodified CXL-DIMMs connected with the CXL-Switch, a MC is needed in the CXL-Switch. The MC is responsible for the customized memory management, including memory allocation/de-allocation, data placement, address mapping, DRAM states maintaining, memory requests scheduling, and so on. For BEACON-D, as shown in Fig. 6.4 (c), the MC (❷) is added into the Switch-Logic. For BEACON-S, as shown in Fig. 6.4 (c), because its NDP module in the CXL-Switch contains a MC, we reuse the MC within its NDP module and no extra components are needed.

To address the issue of Read-Modify-Write (RMW) data race during memory access, the Atomic Engine is added into the Switch-Logic in BEACON-D. For BEACON-S, because its Switch-Logic contains many PEs, we reuse the PEs as the Atomic Engines and no extra components are needed.

❾ ***Atomic Engine:*** With parallel processing, Read-Modify-Write (RMW) data race, i.e., simultaneously reading and updating the same memory location, is a challenge. For example, during k-mer counting, multiple tasks may try to read, increase, then write back the same k-mer counter at the same time. Undetermined order of these operations may lead to incorrect final results. Atomic memory operations can solve the issue of RMW

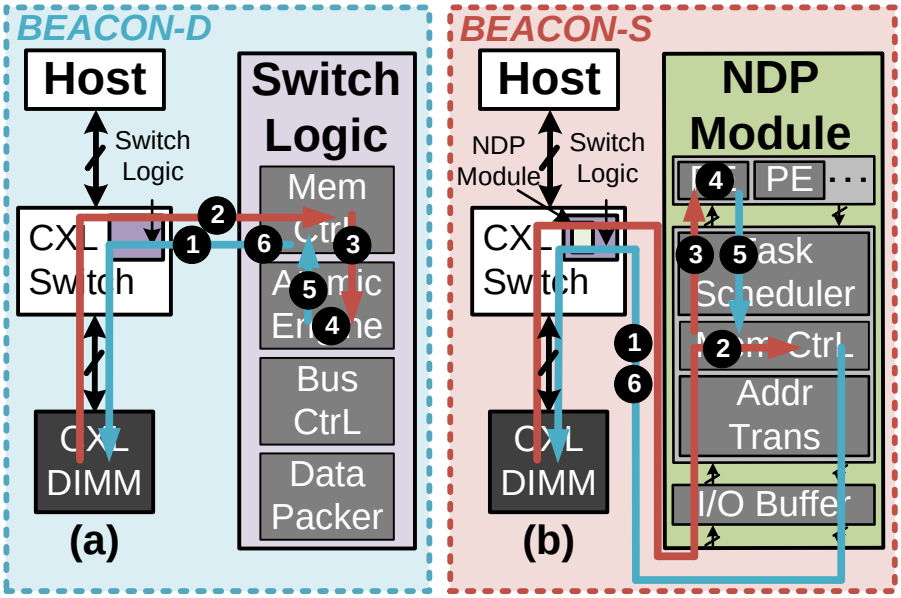

Figure 6.6: Workflow of performing atomic memory operations. (a) BEACON-D. (b) BEACON-S. (© 2022 IEEE)

data race as well as reduce bandwidth consumption [129]. In addition, atomic memory operations are useful for many applications [129]. Due to these reasons, we enable atomic memory operations in BEACON to address the issue of RMW data race.

In BEACON-D and BEACON-S, the arithmetic part of the atomic memory operations are performed in the Atomic Engine, shown in Fig. 6.4 (c), and the PEs within the Switch-Logic, respectively. The workflows of performing atomic memory operations in BEACON are described below:

For BEACON-D, as shown in Fig. 6.6 (a), ❶ The MC in the Switch-Logic issues memory request to the target CXL-DIMM. ❷ The data is brought back to the MC. ❸ The data is forwarded to the Atomic Engine. ❹ The arithmetic operations are performed within the Atomic Engine. ❺ The arithmetic result is sent back to the MC. ❻ The MC issues memory request to write back the result.

Similar workflow for BEACON-S is shown in Fig. 6.6 (b). The difference between the workflows of BEACON-S and BEACON-D is: For BEACON-D, the arithmetic operations

are performed in the Atomic Engine. For BEACON-S, the arithmetic operations are performed in the PEs within the Switch-Logic.

6.3.2.3 CXL-Interface

To reduce the movement of the useless data and improve utilization of the communication bandwidth, as shown in Fig. 6.4 (c), the Data Packer (❽) described above is also added into the CXL-Interface to perform data packing before the data transfer towards the CXL-Switch.

6.3.3 Memory Management Framework

The workflow of the proposed memory management framework is shown in Fig. 6.7. The host coordinates with the CXL-Switch, the unmodified CXL-DIMM, and the CXLG-DIMM (if BEACON-D) to perform memory management.

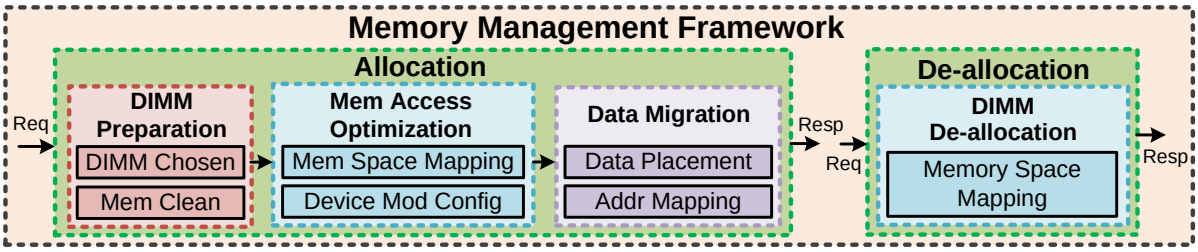

Figure 6.7: The workflow of the memory management framework in BEACON. (© 2022 IEEE)

Memory Allocation: First, the host sends the memory allocation request with the detailed information, e.g, application, algorithm, dataset, parameters, to the CXL-Switch via the framework interface. Second, as shown Fig. 6.7, the the CXL-Switch works with the host, the unmodified CXL-DIMMs, and the CXLG-DIMMs (if BEACON-D) to do memory allocation, including DIMM allocation, memory access optimization, and data migration. Third, the CXL-Switch sends the response, i.e., successful or failed, back to the host via the framework interface. The details in the second step are described below:

• ***DIMM Allocation:*** The proposed memory management framework manages memory in the granularity of CXL-DIMM. According to the memory allocation request, the memory framework tries to allocate the unmodified CXL-DIMMs in proximity to the NDP module, e.g., within the same CXL-Switch. After deciding which CXL-DIMMs to allocate, memory clean is performed to migrate the active data for other applications within these CXL-DIMMs to the other CXL-DIMMs. The host and the CXL-Switch work together to update the related page tables during memory clean. After the memory clean, these chosen CXL-DIMMs are used dedicated for BEACON. The memory address within these CXL-DIMMs are marked as non-cacheable for the host.

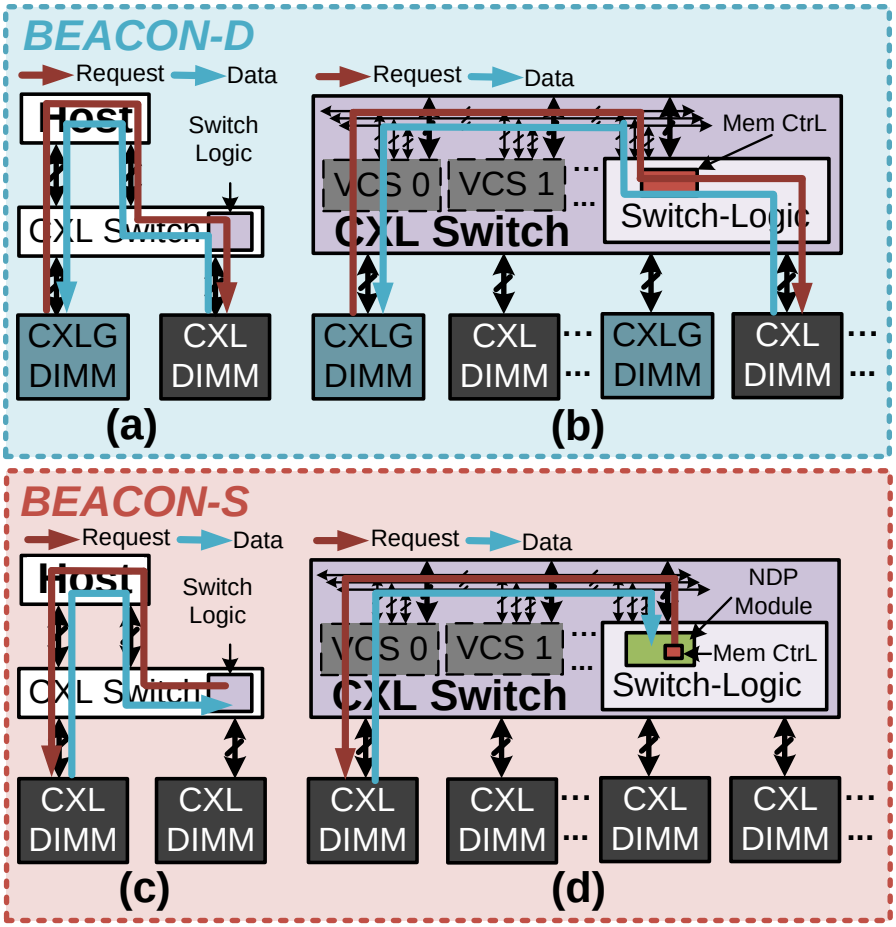

Figure 6.8: (a), (b) Memory access flows before/after the proposed optimizations for BEACON-D. (c), (d) Memory access flows before/after the proposed optimizations for BEACON-S. (© 2022 IEEE)

• ***Memory Access Optimization:*** Because CXL enables cache coherence, with the naïve implementation, as shown in Fig. 6.8 (a) and (c), the memory requests and data from/to the unmodified CXL-DIMMs need to go through the host for coherence purpose, leading to redundant data movement.

Leveraging the protocol of CXL [120], the memory space in the unmodified CXL-DIMMs is mapped to the device memory space in BEACON and BEACON is set to the device-biased mod. With the software configuration and the architecture support from the MC and the Switch-Bus inside the Switch-Logic, the memory access flows to the unmodified CXL-DIMMs in BEACON-D and BEACON-S are show in Fig. 6.8 (b) and (d), respectively. The redundant data movement is eliminated.

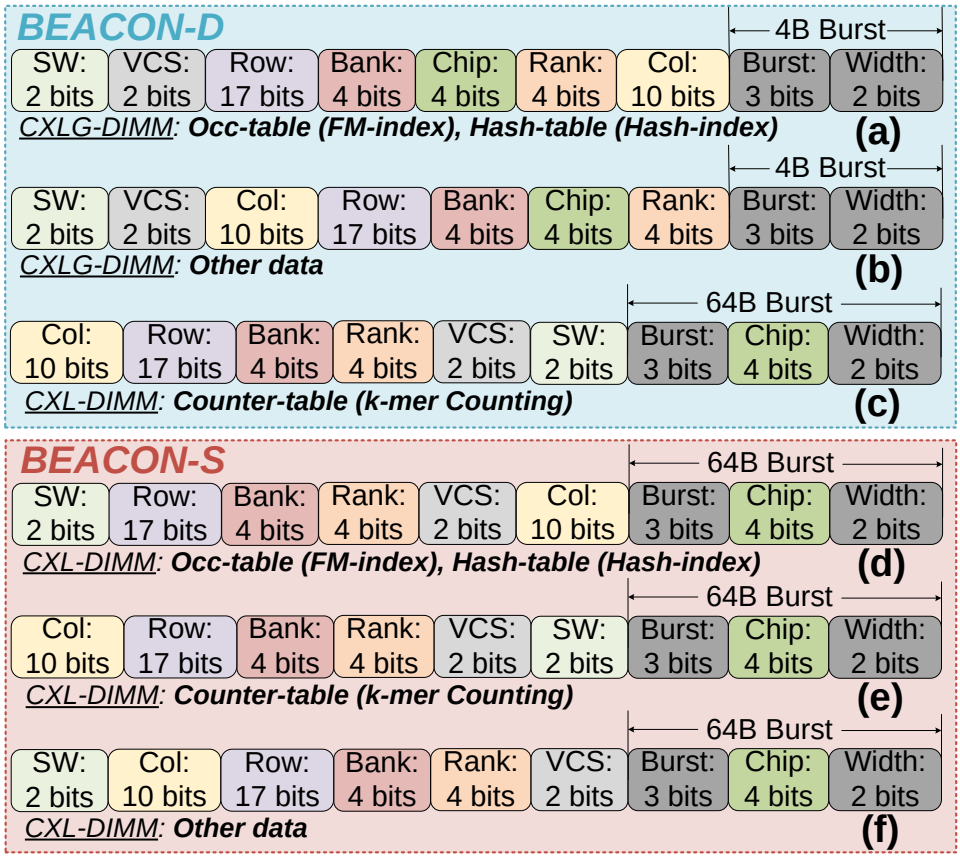

Figure 6.9: Architecture and data aware address mapping. (a), (b), and (c) are designed for BEACON-D. (d), (e), and (f) are designed for BEACON-S.
(© 2022 IEEE)

• ***Data Migration:*** To reduce data movement, the key idea of the proposed data placement scheme is to make full utilization of the data locality. BEACON always tries to put the more frequently accessed data to memory locations in proximity to the NDP module within BEACON.

Different from the previous work [4, 11, 130], which provides a fixed address mapping scheme, we propose the architecture and data aware address mapping scheme to better leverage features of both the architecture and the specific data. The two architecture and data related principles in the proposed address mapping scheme are:

1. Memory address interleaving is performed according to the architecture of the DIMMs to fully leverage the available memory bandwidth. For the CXLG-DIMMs, the memory address interleaving can be done at the chip-level to leverage its ability to perform fine-grained memory access, while the memory address interleaving can only be done at the rank-level for the unmodified CXL-DIMMs.

2. For data with spacial locality [11], we prioritize to map these data in the DRAM row-by-row. As an example, in Hash-index based DNA seeding, multiple matching locations for a seed are stored continuously within the same DRAM row to fully leverage locality.

Following the above two principles, the detailed address mapping schemes in BEACON are shown in Fig. 6.9. The related information, e.g., application, algorithm, data access granularity, are included in the memory requests to the customized MC in BEACON and the customized MC schedules the memory accesses based on these information.

Memory De-alocation: For memory de-allocation, first, host send the memory de-allocation request with detailed information, e.g, how much memory to de-allocate, to the CXL-Switch via the framework interface. Second, the CXL-Switch works with the the CXLG-DIMMs (if BEACON-D) to map the corresponding allocated memory to the host memory space. Third, the CXL-Switch sends the response, i.e., successful or failed, back to the host via the framework interface.

6.3.4 Algorithm-Specific Optimizations

FM-index - Multi-Chip Coalescing: FM-index based DNA seeding requires fine-grained memory access [11]. As shown in Fig. 6.10 (a), fine-grained memory access leads to low memory bandwidth utilization in the conventional DIMM, because only a portion of the data read out from the memory are useful. As shown in Fig. 6.10 (b), the previous work supports fine-grained memory access, i.e., read a single DRAM chip multiple times to get the fine-grained data, to improve the memory bandwidth utilization [11]. Unfortunately, with this method, the amount of memory access to some DRAM chips are greatly increased, while some DRAM chips stay idle. In a word, although better than the conventional approach, the method adopted in the previous work leads to memory bandwidth under-utilization as well.

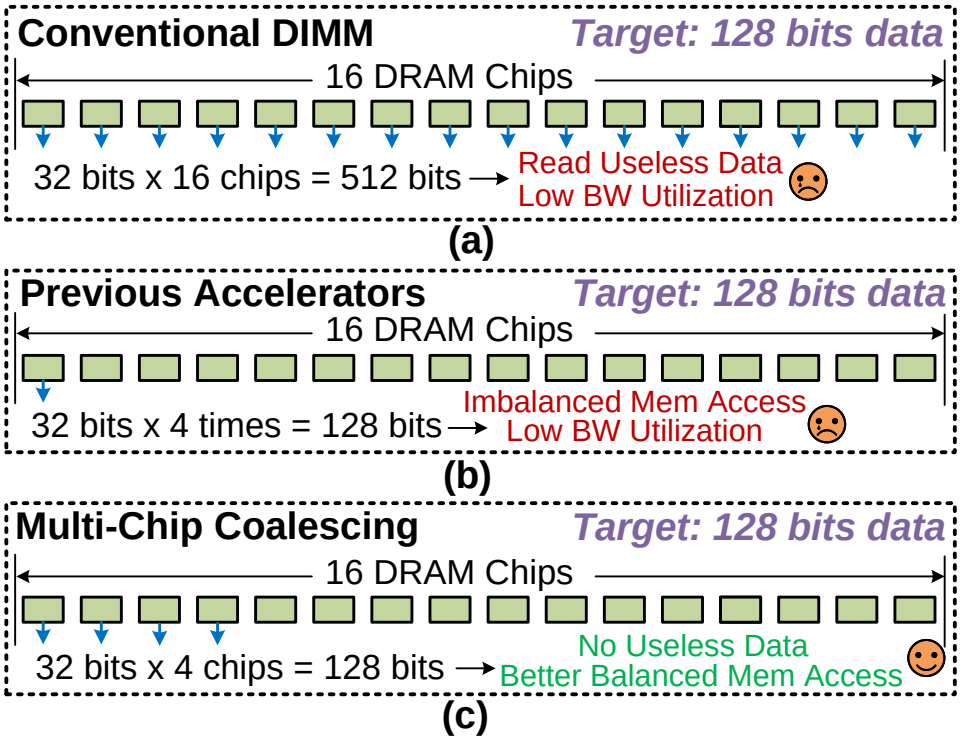

Figure 6.10: Examples of different approaches for fine-grained memory access. (a) Conventional DIMM. (b) The previous accelerators. (c) Multi-chip coalescing. (© 2022 IEEE)

To address this issue, we propose multi-chip coalescing for data placement and memory access in the CXLG-DIMMs within BEACON. As shown in Fig. 6.10 (c), the data is placed and accessed in multi-chip granularity to achieve a sweet point, i.e., no useless data read out and better balanced memory access. The amount of DRAM chips to be coalesced is fine-tuned to achieve the best performance.

k-mer Counting - Single-Pass k-mer Counting: In the previous DIMM based accelerator for k-mer counting, i.e., NEST [4], a multi-pass method is adopted. Specifically, first, each DIMM constructs its own counting Bloom filter locally (processing the entire input for the first time). Next, the counting Bloom filters from different DIMMs are merged to derive the global Bloom filter and the global Bloom filter is distributed to all the DIMMs. Finally, each DIMM performs k-mer counting independently and access its own copy of the global Bloom filter (processing the entire input for the second time). This method eliminates the expensive random remote memory access in the first and the last steps at the cost of processing the entire input twice.

However, BEACON-S performs the computation in the CXL-Switch and the data are distributed in different CXL-DIMMs. With the previous approach, we cannot get the benefits of data localization, while we still suffer from its overhead of processing the entire input twice. For this reason, the method of single-pass k-mer counting is adopted in BEACON-S. To be specific, BEACON-S directly deals with the global Bloom filter distributed to different CXL-DIMMs, eliminating the first step and the last step above.

6.4 Discussion

Extension to Other Applications: BEACON can be easily extended as a practical, cost-effective, and scalable accelerator for other memory-bound applications, such as image processing [131], graph processing [95], and database searching [96], by replacing

the PEs within the NDP module. BEACON can also be extended as general purpose NDP accelerator by replacing the PEs with light-weight, low-power, general-purpose processing units [132].

Extension to Other Architectures: Besides CXL, there are other emerging communication specifications [119], such as Gen-Z [125], OpenCAPI [124], and CCIX [133], the designs and optimizations in BEACON could also be migrated to and be used in the NDP architectures with these emerging communication specifications.

6.5 Experimental Results

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

6.5.1 Experimental Setup

Configuration of the Baselines: For FM-index and Hash-index based DNA seeding, the software we use for the CPU baselines are BWA-MEM [33] and SMALT [101], respectively. The hardware baseline is the previous DIMM based accelerator for DNA seeding, i.e., MEDAL [11]. For k-mer counting, the software we use for the CPU baseline is BFCounter [40]. The hardware baseline is the previous DIMM based accelerator for k-mer counting, i.e., NEST [4]. For DNA pre-alignment, the software we use for the CPU baseline is Shouji [49]. The detailed configurations of these baselines are shown in Table 6.1. The DIMMs used in the experiments have the same parameters. The same PEs, which are described below, are used in the NDP baselines and BEACON. We've ensured that BEACON and the NDP baselines have **the same area overhead**.

We need to mention that the experiments are **in favor of the NDP baselines**. As mentioned in Section 6.2, the baselines cannot use unmodified DIMMs for memory ex-

Table 6.1: Experimental Configuration (© 2022 IEEE)

Configuration of the CPU basline	
CPU Model/CPU Frequency(GHz)	Intel Xeon E5-2680 v3/2.50
Memory(GB)/L1(KB)/L2(KB)/L3(MB) Cache	384/64/256/32
Configuration of MEDAL and NEST	
Memory(GB)/DDR Channels	512/4
Customized DIMM per Channel	2
Configuration of BEACON	
Memory(GB)/Switch/VCS per Switch	512/2/2
CXLG/CXL-DIMM per VCS (BEACON-D)	1/1
CXL-DIMM per VCS (BEACON-S)	2
Configurations of DIMM	
DIMM Capacity(GB)/DRAM Chip	64/8Gb ×4
Ranks per DIMM/Chips per Rank	4/16
BankGroups per Chip/Banks per BankGroup	4/4
Clock Frequency(MHz)/tRCD-tCAS-tRP	1,600/22-22-22

pansion. In the experiments, as shown in Table 6.1, all the DIMMs in the NDP baselines are customized DIMMs according to their designs. These customized DIMMs provide better performance and higher energy efficiency for genome analysis. In BEACON-D, only a portion of the DIMMs are customized. In BEACON-S, none of the DIMMs are customized. Due to the hardware configuration in favor of the baselines, the previous DIMM based accelerators have slightly higher energy efficiency than BEACON. BEACON can also achieve higher energy efficiency with more customized CXLG-DIMMs, but this is not our design goal. Because it's obviously more convenient, scalable, and cost-effective to scale-out by leveraging existing unmodified CXL-DIMMs in the memory pool, instead of keep inserting customized DIMMs into the system.

Configuration of BEACON: Ramulator [102] is modified to build a cycle-accurate simulator for BEACON. The configuration of BEACON is shown in Table 6.1. We use pre-layout Design Compiler [103] with 28 nm technology [104] to synthesize the timing, energy, and area of the PEs. The area of each PE is 14090.23 um^2. The dynamic power

and leakage power of each PE is 9.48 mW and 18.97 uW. The computational latencies for FM-index based DNA seeding, Hash-index based DNA seeding, k-mer counting, and DNA pre-alignment are equal to 16, 10, 59, and 82 DRAM cycles. For BEACON-D, there are 128 PEs within each CXLG-DIMM. For BEACON-S, there are 256 PEs within each CXL-Switch. The configuration of the DIMM is shown in Table 6.1. The energy consumption of DRAM is calculated with DRAMPower [105]. The energy consumption for the communication come from [106, 134].

Datasets: For FM-index based DNA seeding, Hash-index based DNA seeding, and DNA pre-alignment, five different genomes, i.e., Pinus taeda (Pt), Picea glauca (Pg), Sequoia sempervirens (Ss), Ambystoma mexicanum (Am), and Neoceratodus forsteri (Nf), from NCBI [109] are used in the experiments. For k-mer counting, human genome with a coverage ratio of 50x is used in the experiments.

Next, for different applications, the performance improvement and energy reduction of BEACON are presented. All the data are normalized to the corresponding data of the 48-thread CPU baselines.

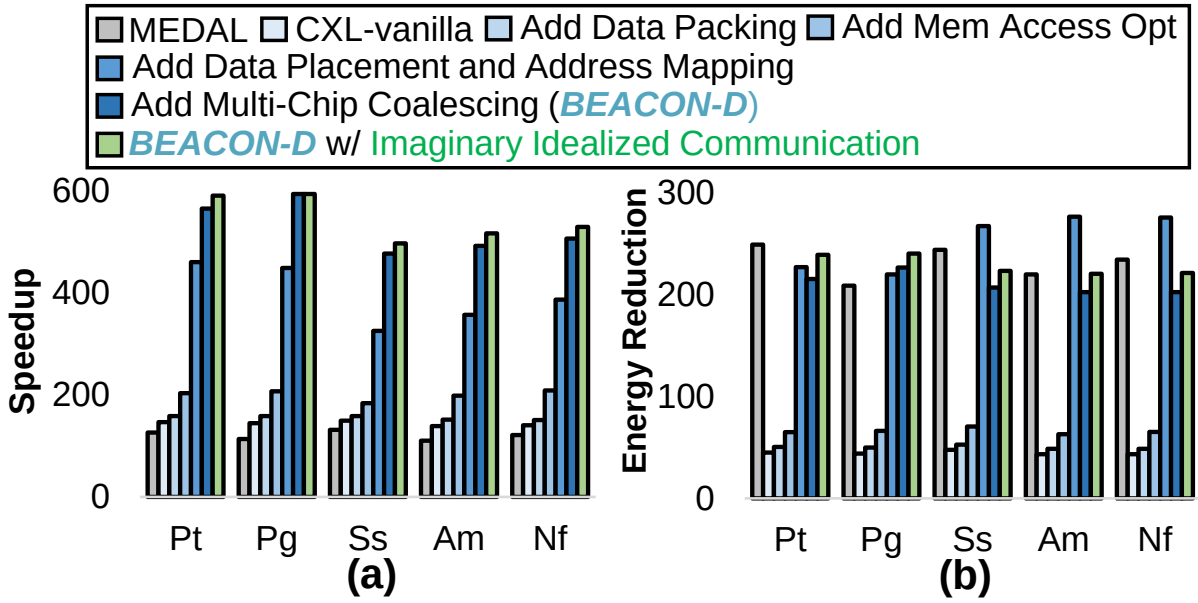

Figure 6.11: FM-index based DNA seeding for BEACON-D. (a) Performance improvement. (b) Energy reduction. (© 2022 IEEE)

6.5.2 FM-index based DNA Seeding

BEACON-D: For BEACON-D, the performance improvement of designs with step-by-step optimizations on FM-index based DNA seeding is shown in Fig. 6.11 (a). On average, CXL-vanilla, i.e., the naïve NDP accelerator near the memory pool with the CXL support, improves the performance of the 48-thread CPU baseline and MEDAL by 144.18x and 1.20x. As for the optimizations, data packing improves the performance by 1.08x. Performance improvement of 1.29x is achieved with memory access optimization. Moreover, the proposed data placement and address mapping gain performance improvement of 1.96x. In addition, multi-chip coalescing brings performance improvement of 1.34x. Overall, BEACON-D outperforms the 48-thread CPU and MEDAL by 525.73x and 4.36x in performance, respectively. BEACON-D achieves 96.52% performance of the corresponding design with idealized communication (infinite bandwidth and zero latency), indicating that communication is no longer an issue.

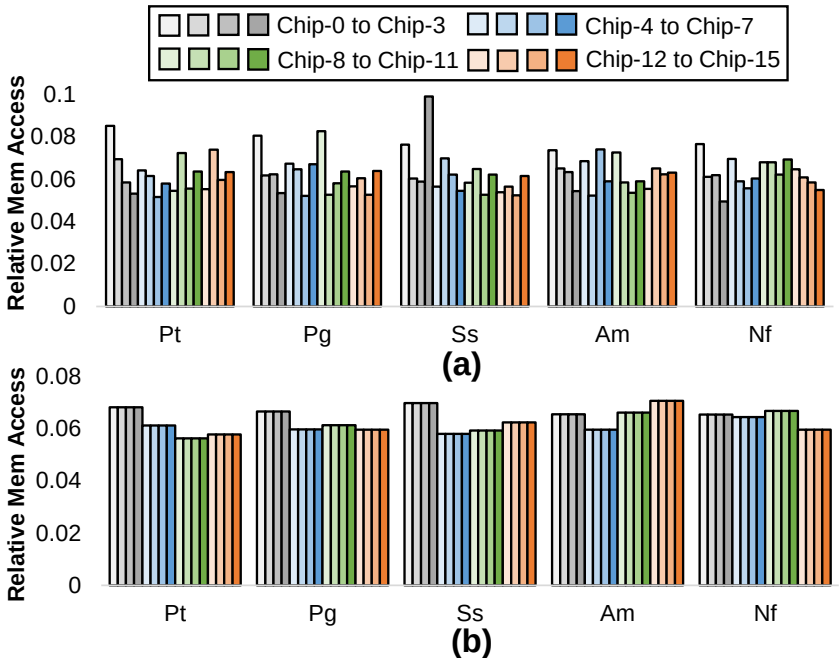

Figure 6.12: Normalized memory access to different DRAM chips for FM-index based DNA seeding. (a), (b) Without and with multi-chip coalescing. (© 2022 IEEE)

The performance benefits of multi-chip coalescing comes from balanced memory access to different DRAM chips and better utilization of memory bandwidth. As shown in Fig. 6.12 (a), without multi-chip coalescing, the memory access to different DRAM chips are unevenly distributed, leading to memory bandwidth under-utilization. As shown in Fig. 6.12 (b), with multi-chip coalescing, the memory access to different DRAM chips are well-balanced, i.e., with less variations, and memory bandwidth is better utilized.

For energy efficiency, as shown in Fig. 6.11 (b), BEACON-D reduces energy consumption of the CPU baseline by 210.59x and achieves 91.26% energy efficiency of MEDAL. Compared with the design with idealized communication, BEACON-D achieves 92.09% of its energy efficiency.

BEACON-S: For BEACON-S, the performance improvement of designs with step-by-step optimizations on FM-index based DNA seeding is shown in Fig. 6.13 (a). On average, CXL-vanilla improves the performance of the 48-thread CPU baseline and MEDAL by 146.64x and 1.22x. As for the optimizations, data packing improves the performance by 1.08x. Performance improvement of 1.57x is achieved with memory access optimization. Moreover, the proposed data placement and address mapping gain performance improvement of 1.18x for BEACON. Overall, BEACON-S outperforms the 48-thread CPU and MEDAL by 291.62x and 2.42x in performance, respectively. BEACON-S achieves 98.48% performance of the corresponding design with idealized communication, eliminated the issue of communication.

Energy wise, as shown in Fig. 6.13 (b), BEACON-S reduces energy consumption of the CPU baseline by 100.60x and achieves 43.59% energy efficiency of MEDAL. Compared with the corresponding design with idealized communication, BEACON-S achieves 82.57% of its energy efficiency.

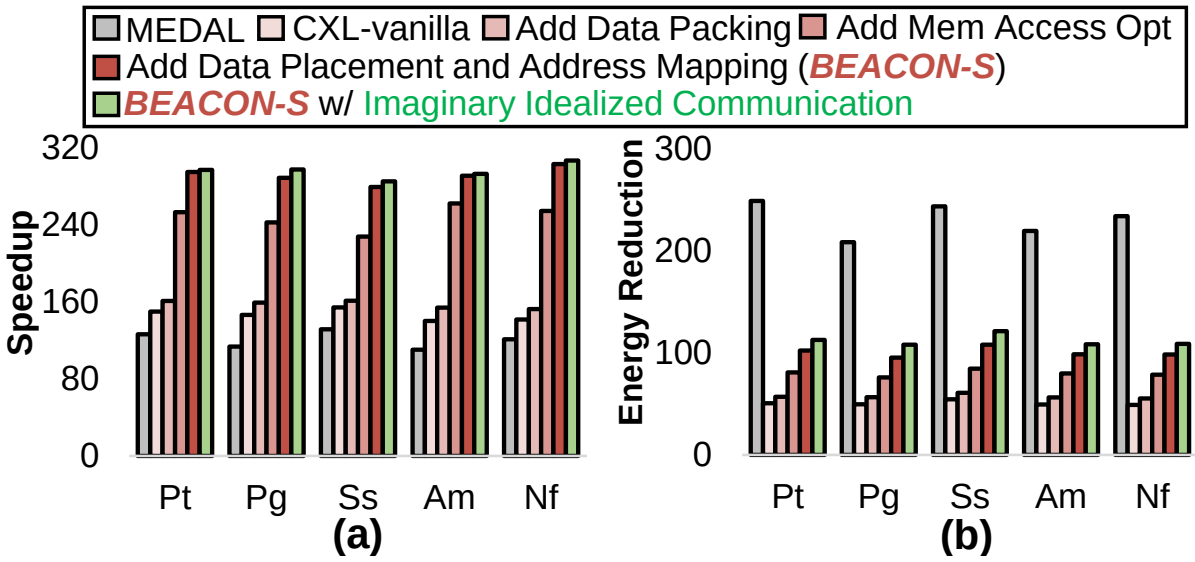

Figure 6.13: FM-index based DNA seeding for BEACON-S. (a) Performance improvement. (b) Energy reduction. (© 2022 IEEE)

6.5.3 Hash-index based DNA Seeding

BEACON-D: For BEACON-D, the performance improvement of designs with step-by-step optimizations on Hash-index based DNA seeding are shown in Fig. 6.14 (a). On average, CXL-vanilla improves the performance of the 48-thread CPU baseline and MEDAL by 309.13x and 2.54x. As for the optimizations, performance improvement of 1.81x is achieved with memory access optimization. Data packing, data placement, and address mapping don't bring much performance benefits, because of two reasons: First, Hash-index based DNA seeding doesn't require many fine-grained memory access. Second, the communication is not a bottleneck in BEACON-D for this application even only with the proposed architecture support. Overall, BEACON-D outperforms the 48-thread CPU and MEDAL by 572.17x and 4.70x, respectively. BEACON-D achieves 98.59% performance of the corresponding design with idealized communication, indicating that communication is no longer an issue.

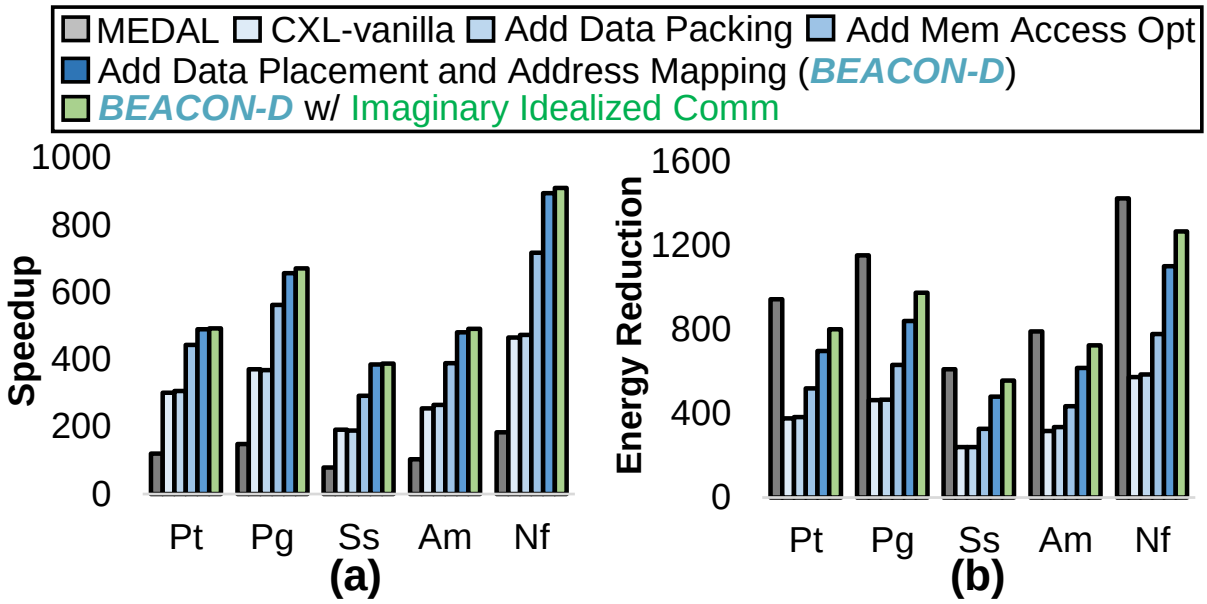

Figure 6.14: Hash-index based DNA seeding for BEACON-D. (a) Performance improvement. (b) Energy reduction. (© 2022 IEEE)

For energy efficiency, as shown in Fig. 6.14 (b), BEACON-D reduces energy consumption of the CPU baseline by 960.72x and achieves 85.58% energy efficiency of MEDAL. Compared with the design with idealized communication, BEACON-D achieves 84.05% of its energy efficiency.

BEACON-S: For BEACON-S, the performance improvement of designs with step-by-step optimizations on Hash-index based DNA seeding are shown in Fig. 6.15 (a). On average, CXL-vanilla improves the performance of the 48-thread CPU baseline and MEDAL by 302.48x and 2.48x. As for the optimizations, performance improvement of 1.50x is achieved with memory access optimization. Moreover, the proposed data placement and address mapping gain performance improvement of 1.21x. Data packing doesn't bring much benefits, because of the limited amount of fine-grained memory access in Hash-index based DNA seeding. Overall, BEACON-S outperforms the 48-thread CPU and MEDAL by 556.66x and 4.57x. BEACON-S achieves 98.64% performance of the corresponding design with idealized communication, eliminated the issue of communication.

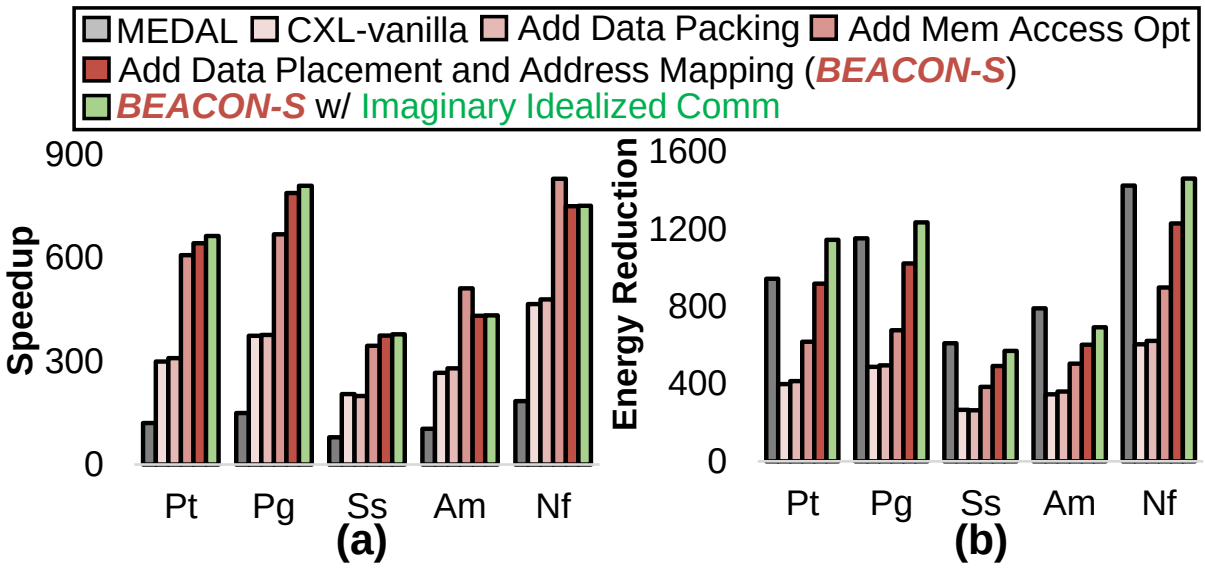

Figure 6.15: Hash-index based DNA seeding for BEACON-S. (a) Performance improvement. (b) Energy reduction. (© 2022 IEEE)

Energy wise, as shown in Fig. 6.15 (b), BEACON-S reduces energy consumption of the CPU baseline by 832.32x and achieves 76.11% energy efficiency of MEDAL. Compared with the design with the corresponding design with idealized communication, BEACON-S achieves 86.27% of its energy efficiency.

6.5.4 k-mer Counting

BEACON-D: For BEACON-D, the performance improvement of designs with step-by-step optimizations on k-mer counting are shown in Fig. 6.16 (a). On average, CXL-vanilla improves the performance of the 48-thread CPU baseline and NEST by 124.88x and 1.46x. As for the optimizations, data packing improves the performance by 1.07x. Performance improvement of 2.75x is achieved with memory access optimization. Moreover, the proposed data placement and address mapping gain performance improvement of 1.21x. With all proposed optimizations, BEACON-D outperforms the 48-thread CPU and NEST by 443.08x and 5.19x, respectively. BEACON-D achieves 92.98% performance of the corresponding design with idealized communication, indicating that communica-

tion is no longer an issue.

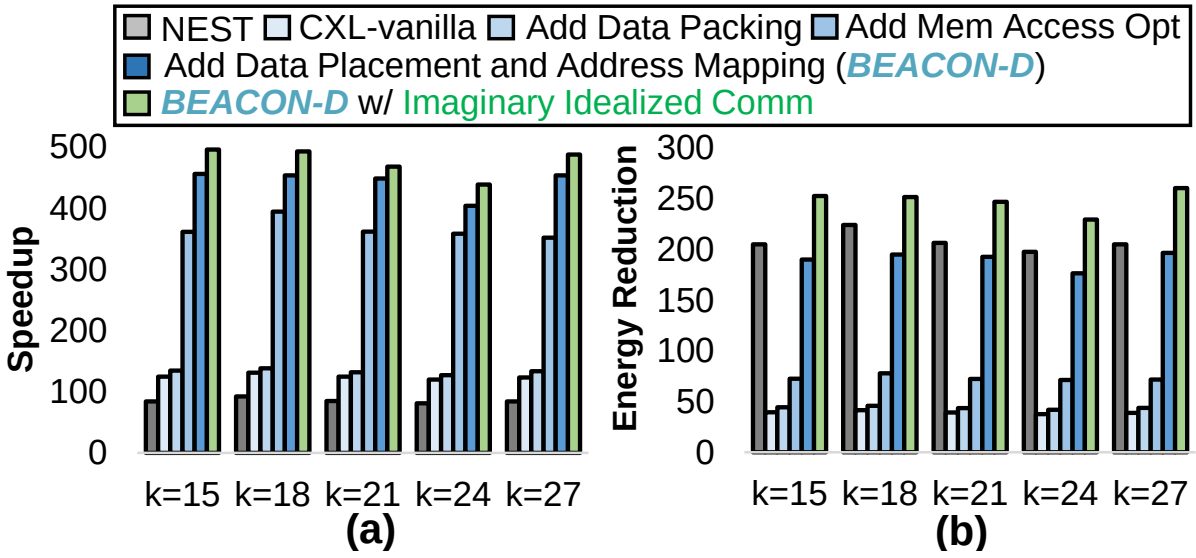

Figure 6.16: k-mer counting for BEACON-D. (a) Performance improvement. (b) Energy reduction. (© 2022 IEEE)

For energy efficiency, as shown in Fig. 6.16 (b), BEACON-D reduces energy consumption of the CPU baseline by 190.23x and achieves 91.57% energy efficiency of NEST. Compared with the corresponding design with idealized communication, BEACON-D achieves 76.71% of its energy efficiency.

BEACON-S: For BEACON-S, the performance improvement of designs with step-by-step optimizations on k-mer counting are shown in Fig. 6.17 (a). On average, CXL-vanilla improves the performance of the 48-thread CPU baseline and NEST by 125.57x and 1.47x. As for the optimizations, data packing improves the performance by 1.09x. Performance improvement of 2.83x is achieved with memory access optimization. The proposed data placement and address mapping achieves 92.16% performance of the previous case, because communication is not a issue for k-mer counting in BEACON-S and data localization undermines utilization of the available memory bandwidth. The proposed data placement and address mapping scheme reduce data movement and lead to

1.12x improvement in energy efficiency. In addition, single-pass k-mer counting brings performance improvement of 1.48x. Overall, BEACON-S outperforms the 48-thread CPU and NEST by 527.99x and 6.19x, respectively. BEACON-S achieves 99.48% performance of the corresponding design with idealized communication, eliminated the performance bottleneck of communication.

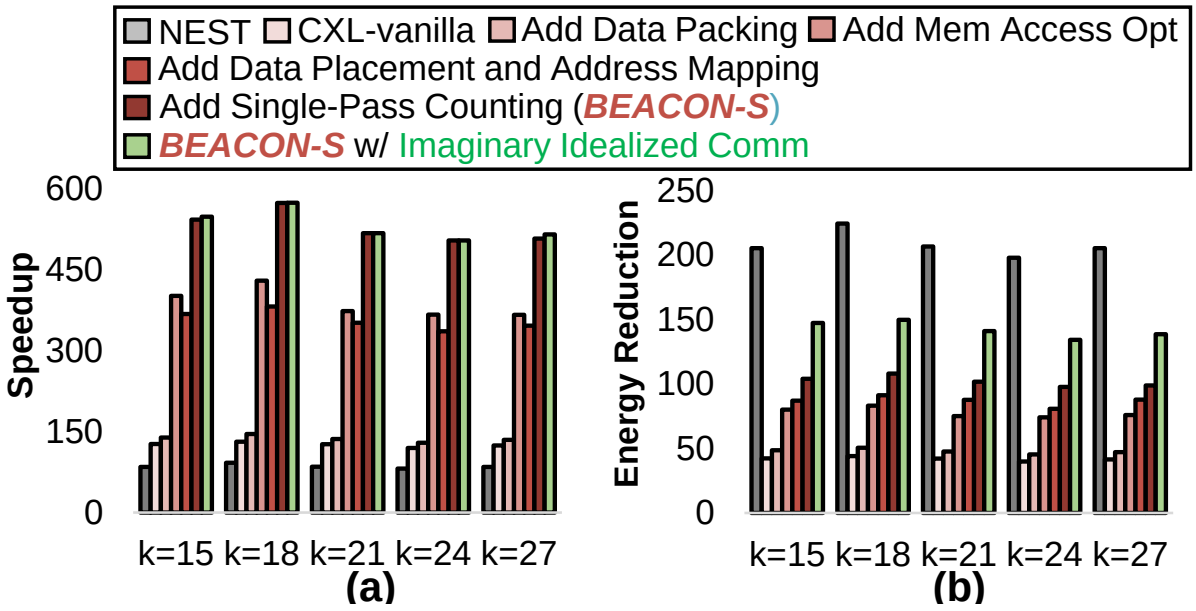

Figure 6.17: k-mer counting for BEACON-S. (a) Performance improvement. (b) Energy reduction. (© 2022 IEEE)

Energy wise, as shown in Fig. 6.17 (b), BEACON-S reduces energy consumption of the CPU baseline by 102.05x and achieves 49.12% energy efficiency of NEST. Compared with the corresponding design with idealized communication, BEACON-S achieves 71.82% of its energy efficiency.

6.5.5 DNA Pre-alignment

For DNA pre-alignment, as shown in Fig. 6.18, BEACON-D and BEACON-S improve performance of the 48-thread CPU baseline by 362.04x and 359.36x, respectively.

They also reduce the energy consumption of the 48-thread CPU baseline by 387.05x and 382.80x, respectively.

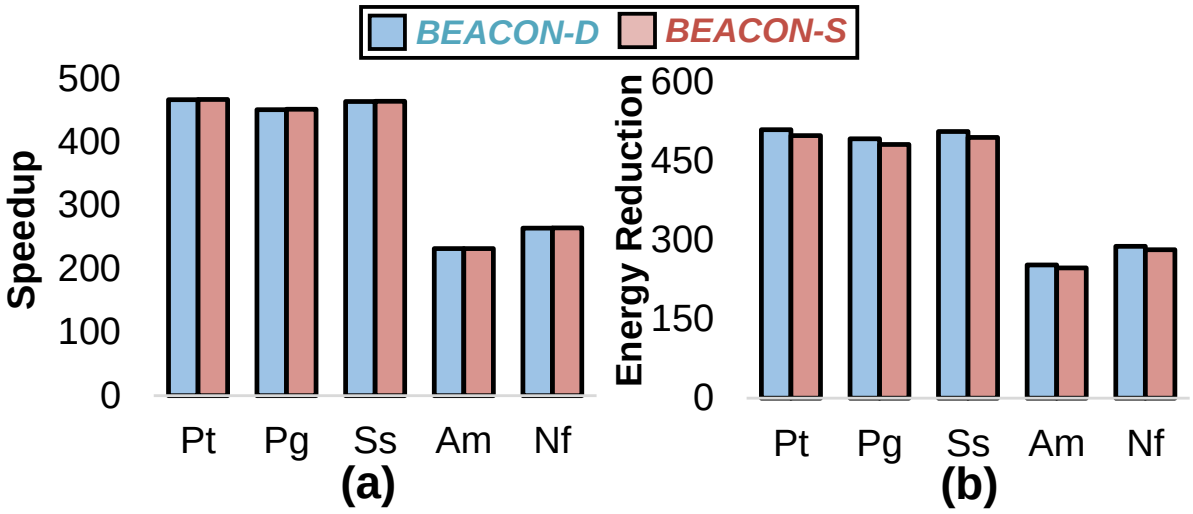

Figure 6.18: DNA pre-alignment. (a) Performance improvement. (b) Energy reduction. (© 2022 IEEE)

6.6 Conclusion

To address the limitations of the previous DIMM based accelerators for genome analysis, i.e., performance bottle-necked by communication and limited potential for memory expansion under the scenario of memory dis-aggregation, this chapter proposes BEACON, including BEACON-D and BEACON-S. As DIMM based NDP accelerator located near the dis-aggregated memory pool with the CXL support, BEACON enables efficient on-demand memory expansion with unmodified CXL-DIMMs and eliminates the performance bottleneck of communication without making any modification to the cost-sensitive DRAM dies. In addition, BEACON adopts algorithm-specific optimizations to further improve its performance and can be used for multiple applications in genome analysis. Experimental results demonstrate that BEACON-D and BEACON-S improve performance of state-of-the-art DIMM based NDP accelerators by 4.70x and 4.13x, respectively.

Chapter 7

Summary

The booming genome data brings great challenges to the data processing in genome analysis, especially for the memory-bound applications in genome analysis. Memory-centric architecture is a promising solution to tackle this problem. This dissertation focuses on exploring the memory-centric architecture for genome analysis, covering designs and optimizations in both hardware and software.

We first explore to build memory-centric accelerators for genome analysis based on the emerging memory technology, i.e., ReRAM. Our emerging memory technology based PIM design for genome analysis provides ultra-high performance and energy efficiency by performing in-situ computation inside the emerging memory array to leverage massive parallelism and eliminate data movement.

RADAR, a high performance and energy efficient PIM architecture based on the 3D ReCAM, to accelerate BLASTN, i.e., a widely used DNA alignment tool, is proposed. RADAR merges memory and computation together to support in-situ computation with massive parallelism and minimal data movement, leading to its ultra-high performance and energy efficiency. In addition, we propose a data mapping scheme, including the dense data mapping scheme and the Tail Bits Duplication (TBD) technique, to further optimize computation and storage. RADAR distinguishes itself from the previous ReRAM based

PIM accelerator for DNA alignment with efficient data storage and no issue of device endurance. Compared with the CPU baseline, we demonstrate that RADAR provides ultra-high performance and energy efficiency.

We then explore to leverage the conventional memory technology, i.e., DRAM, to design memory-centric accelerators for genome analysis. Our emerging memory technologies based NDP designs for genome analysis highlight practicality and cost-effectiveness by integrating the processing elements near the conventional memory array to utilize extra memory bandwidth and reduce data movement.

First, we propose MEDAL, a high-performance, energy-efficient, practical, cost-effective, and DIMM based NDP accelerator to accelerate DNA seeding efficiently. 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 fine-grained random memory access and inter-task divergence, 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 MEDAL outperforms the 16-thread CPU baseline and two state-of-the-art NDP accelerators.

Then, based on MEDAL, we propose NEST, a high-performance, energy-efficient, practical, cost-effective, and DIMM based NDP accelerator for k-mer counting. To fully unleash the performance of NEST, we propose an architecture-specific k-mer counting algorithm, together with a dedicated workflow, to reduce unnecessary inter-DIMM communication and improve parallelism. In addition, we propose a novel address mapping scheme to improve memory bandwidth utilization. The challenge of workload balance

is addressed with the proposed task scheduling. Scattered memory access and task-switching are proposed to eliminate the redundant memory access. Evaluation results demonstrate that NEST achieves good performance improvement and energy reduction, compared with a 48-thread CPU, a CPU/GPU hybrid approach, and a state-of-the-art NDP accelerator.

Finally, we propose BEACON, including BEACON-D and BEACON-S, to address the limitations of the previous DIMM based accelerators for genome analysis, i.e., performance bottle-necked by communication and limited potential for memory expansion under the scenario of memory dis-aggregation. As DIMM based NDP accelerator located near the dis-aggregated memory pool with the CXL support, BEACON enables efficient on-demand memory expansion with unmodified CXL-DIMMs and eliminates the performance bottleneck of communication without making any modification to the cost-sensitive DRAM dies. In addition, BEACON adopts algorithm-specific optimizations to further improve its performance and can be used for multiple applications in genome analysis. We compare BEACON with the 48-thread CPU baseline and state-of-the-art DIMM based NDP accelerators for multiple applications with different datasets, demonstrating that BEACON provides good performance and energy efficiency.

We hope the work in this thesis would be helpful and inspirational for software-hardware co-design, memory-centric accelerator design, genome analysis accelerator design, and design of memory-centric accelerator for genome analysis.

Bibliography

[1] H. Ellegren, *Genome sequencing and population genomics in non-model organisms*, *Trends in ecology & evolution* **29** (2014), no. 1 51–63.

[2] X. Ma, M. Mau, and T. F. Sharbel, *Genome editing for global food security*, *Trends in biotechnology* **36** (2018), no. 2 123–127.

[3] J. D. Merker, A. M. Wenger, T. Sneddon, M. Grove, Z. Zappala, L. Fresard, D. Waggott, S. Utiramerur, Y. Hou, K. S. Smith, *et. al.*, *Long-read genome sequencing identifies causal structural variation in a mendelian disease*, *Genetics in Medicine* **20** (2018), no. 1 159–163.

[4] W. Huangfu, K. T. Malladi, S. Li, P. Gu, and Y. Xie, *NEST: DIMM based near-data-processing accelerator for k-mer counting*, in *2020 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*, pp. 1–9, IEEE, 2020.

[5] G. S. L. Center, "Your doctorÂ¡Â¯s new genetic tools." `http://learn.genetics.utah.edu/content/precision/example/`, 2017.

[6] R. Lu, X. Zhao, J. Li, P. Niu, B. Yang, H. Wu, W. Wang, H. Song, B. Huang, N. Zhu, *et. al.*, *Genomic characterisation and epidemiology of 2019 novel coronavirus: implications for virus origins and receptor binding*, *The Lancet* (2020).

[7] W.-j. Guan, Z.-y. Ni, Y. Hu, W.-h. Liang, C.-q. Ou, J.-x. He, L. Liu, H. Shan, C.-l. Lei, D. S. Hui, *et. al.*, *Clinical characteristics of coronavirus disease 2019 in china*, *New England Journal of Medicine* (2020).

[8] J. Shendure and H. Ji, *Next-generation DNA sequencing*, *Nature biotechnology* **26** (2008), no. 10 1135–1145.

[9] N. H. G. R. Institute, "DNA sequencing costs: Data." `https://www.genome.gov/about-genomics/fact-sheets/DNA-Sequencing-Costs-Data`, 2022.

[10] A. Nag, C. Ramachandra, R. Balasubramonian, R. Stutsman, E. Giacomin, H. Kambalasubramanyam, and P.-E. Gaillardon, *Gencache: Leveraging in-cache*

operators for efficient sequence alignment, in *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*, pp. 334–346, 2019.

[11] W. Huangfu, X. Li, S. Li, X. Hu, P. Gu, and Y. Xie, *MEDAL: Scalable DIMM based near data processing accelerator for DNA seeding algorithm*, in *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*, pp. 587–599, 2019.

[12] N. C. for Biotechnology Information, "Genbank and WGS statistics." `https://www.ncbi.nlm.nih.gov/genbank/statistics/`, 2022.

[13] Z. D. Stephens, S. Y. Lee, F. Faghri, R. H. Campbell, C. Zhai, M. J. Efron, R. Iyer, M. C. Schatz, S. Sinha, and G. E. Robinson, *Big data: astronomical or genomical?*, *PLoS biology* **13** (2015), no. 7 e1002195.

[14] Y. Turakhia, G. Bejerano, and W. J. Dally, *Darwin: A genomics co-processor provides up to 15,000 x acceleration on long read assembly*, in *ACM SIGPLAN Notices*, vol. 53, pp. 199–213, ACM, 2018.

[15] W. Huangfu, S. Li, X. Hu, and Y. Xie, *RADAR: a 3d-ReRAM based DNA alignment accelerator architecture*, in *2018 55th ACM/ESDA/IEEE Design Automation Conference (DAC)*, pp. 1–6, IEEE, 2018.

[16] S. Deorowicz, M. Kokot, S. Grabowski, and A. Debudaj-Grabysz, *Kmc 2: fast and resource-frugal k-mer counting*, *Bioinformatics* **31** (2015), no. 10 1569–1576.

[17] R. Chikhi and G. Rizk, *Space-efficient and exact de bruijn graph representation based on a bloom filter*, *Algorithms for Molecular Biology* **8** (2013), no. 1 22.

[18] J. Zhang, H. Lin, P. Balaji, and W.-c. Feng, *Optimizing burrows-wheeler transform-based sequence alignment on multicore architectures*, in *Cluster, Cloud and Grid Computing (CCGrid), 2013 13th IEEE/ACM International Symposium on*, pp. 377–384, IEEE, 2013.

[19] N. Cadenelli, Z. Jaksić, J. Polo, and D. Carrera, *Considerations in using OpenCL on GPUs and FPGAs for throughput-oriented genomics workloads*, *Future Generation Computer Systems* **94** (2019) 148–159.

[20] M. Erbert, S. Rechner, and M. Müller-Hannemann, *Gerbil: a fast and memory-efficient k-mer counter with GPU-support*, *Algorithms for Molecular Biology* **12** (2017), no. 1 9.

[21] M. Lu, Y. Tan, G. Bai, and Q. Luo, *High-performance short sequence alignment with GPU acceleration*, *Distributed and Parallel Databases* **30** (2012), no. 5-6 385–399.

[22] Y. Liu and B. Schmidt, *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*, pp. 684–690, IEEE, 2012.

[23] Y. Liu, A. Wirawan, and B. Schmidt, *CUDASW++ 3.0: accelerating Smith-Waterman protein database search by coupling CPU and GPU SIMD instructions*, *BMC bioinformatics* **14** (2013), no. 1 117.

[24] J. Wang, X. Xie, and J. Cong, *Communication optimization on GPU: A case study of sequence alignment algorithms*, in *2017 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pp. 72–81, IEEE, 2017.

[25] S. M. Bose, V. S. Lalapura, S. Saravanan, and M. Purnaprajna, *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)*, pp. 347–352, IEEE, 2019.

[26] M.-C. F. Chang, Y.-T. Chen, J. Cong, P.-T. Huang, C.-L. Kuo, and C. H. Yu, *The SMEM seeding acceleration for DNA sequence alignment*, in *Field-Programmable Custom Computing Machines (FCCM), 2016 IEEE 24th Annual International Symposium on*, pp. 32–39, IEEE, 2016.

[27] E. Fernandez, W. Najjar, and S. Lonardi, *String matching in hardware using the FM-index*, in *Field-Programmable Custom Computing Machines (FCCM), 2011 IEEE 19th Annual International Symposium on*, pp. 218–225, IEEE, 2011.

[28] I. T. Li, W. Shum, and K. Truong, *160-fold acceleration of the Smith-Waterman algorithm using a Field Programmable Gate Array (FPGA)*, *BMC bioinformatics* **8** (2007), no. 1 185.

[29] M. Alser, H. Hassan, H. Xin, O. Ergin, O. Mutlu, and C. Alkan, *Gatekeeper: a new hardware architecture for accelerating pre-alignment in dna short read mapping*, *Bioinformatics* **33** (2017), no. 21 3355–3363.

[30] J. S. Kim, D. S. Cali, H. Xin, D. Lee, S. Ghose, M. Alser, H. Hassan, O. Ergin, C. Alkan, and O. Mutlu, *GRIM-Filter: Fast seed location filtering in DNA read mapping using processing-in-memory technologies*, *BMC genomics* **19** (2018), no. 2 23–40.

[31] N. Ahmed, V.-M. Sima, E. Houtgast, K. Bertels, and Z. Al-Ars, *Heterogeneous hardware/software acceleration of the BWA-MEM DNA alignment algorithm*, in *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, pp. 240–246, IEEE Press, 2015.

[32] B. K. Joardar, P. Ghosh, P. P. Pande, A. Kalyanaraman, and S. Krishnamoorthy, *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*, pp. 1–8, 2019.

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

[34] T. E. Carlson, W. Heirman, and L. Eeckhout, *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*, p. 52, ACM, 2011.

[35] T. Vijayaraghavan, A. Rajesh, and K. Sankaralingam, *MPU-BWM: Accelerating sequence alignment*, *IEEE Computer Architecture Letters* **17** (2018), no. 2 179–182.

[36] H. Asghari-Moghaddam, Y. H. Son, J. H. Ahn, and N. S. Kim, *Chameleon: Versatile and practical near-DRAM acceleration architecture for large memory systems*, in *Microarchitecture (MICRO), 2016 49th Annual IEEE/ACM International Symposium on*, pp. 1–13, IEEE, 2016.

[37] J. Cong, Z. Fang, M. Gill, F. Javadi, and G. Reinman, *AIM: accelerating computational genomics through scalable and noninvasive accelerator-interposed memory*, in *Proceedings of the International Symposium on Memory Systems*, pp. 3–14, ACM, 2017.

[38] P. Liu, A. Hemani, K. Paul, C. Weis, M. Jung, and N. Wehn, *3D-stacked many-core architecture for biological sequence analysis problems*, *International Journal of Parallel Programming* **45** (2017), no. 6 1420–1460.

[39] R. Kaplan, L. Yavits, and R. Ginosar, *RASSA: Resistive prealignment accelerator for approximate DNA long read mapping*, *IEEE Micro* **39** (2018), no. 4 44–54.

[40] P. Melsted and J. K. Pritchard, *Efficient counting of k-mers in DNA sequences using a bloom filter*, *BMC bioinformatics* **12** (2011), no. 1 333.

[41] H. Li and R. Durbin, *Fast and accurate short read alignment with burrows–wheeler transform*, *bioinformatics* **25** (2009), no. 14 1754–1760.

[42] Y. Wang, X. Li, D. Zang, G. Tan, and N. Sun, *Accelerating FM-index search for genomic data processing*, in *Proceedings of the 47th International Conference on Parallel Processing*, pp. 1–12, 2018.

[43] P. Chi, S. Li, C. Xu, T. Zhang, J. Zhao, Y. Liu, Y. Wang, and Y. Xie, *PRIME: a novel processing-in-memory architecture for neural network computation in ReRAM-based main memory*, in *ACM SIGARCH Computer Architecture News*, vol. 44, pp. 27–39, IEEE Press, 2016.

[44] S. R. Kulkarni, D. V. Kadetotad, S. Yin, J.-S. Seo, and B. Rajendran, *Neuromorphic hardware accelerator for SNN inference based on STT-RAM crossbar arrays*, in *2019 26th IEEE International Conference on Electronics, Circuits and Systems (ICECS)*, pp. 438–441, IEEE, 2019.

[45] Q. Wang, G. Niu, W. Ren, R. Wang, X. Chen, X. Li, Z.-G. Ye, Y.-H. Xie, S. Song, and Z. Song, *Phase change random access memory for neuro-inspired computing*, *Advanced Electronic Materials* **7** (2021), no. 6 2001241.

[46] S. Li, D. Niu, K. T. Malladi, H. Zheng, B. Brennan, and Y. Xie, *DRISA: A DRAM-based reconfigurable in-situ accelerator*, in *2017 50th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pp. 288–301, IEEE, 2017.

[47] H. Jiang, S. Huang, X. Peng, J.-W. Su, Y.-C. Chou, W.-H. Huang, T.-W. Liu, R. Liu, M.-F. Chang, and S. Yu, *A two-way SRAM array based accelerator for deep neural network on-chip training*, in *2020 57th ACM/IEEE Design Automation Conference (DAC)*, pp. 1–6, IEEE, 2020.

[48] A. Ramachandran, Y. Heo, W.-m. Hwu, J. Ma, and D. Chen, *FPGA accelerated DNA error correction*, in *2015 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 1371–1376, IEEE, 2015.

[49] M. Alser, H. Hassan, A. Kumar, O. Mutlu, and C. Alkan, *Shouji: a fast and efficient pre-alignment filter for sequence alignment*, *Bioinformatics* **35** (2019), no. 21 4255–4263.

[50] N. Mcvicar, C.-C. Lin, and S. Hauck, *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)*, pp. 203–210, IEEE, 2017.

[51] B. H. Bloom, *Space/time trade-offs in hash coding with allowable errors*, *Communications of the ACM* **13** (1970), no. 7 422–426.

[52] A. Kirsch and M. Mitzenmacher, *Less hashing, same performance: building a better bloom filter*, in *European Symposium on Algorithms*, pp. 456–467, Springer, 2006.

[53] L. Fan, P. Cao, J. Almeida, and A. Z. Broder, *Summary cache: a scalable wide-area web cache sharing protocol*, *IEEE/ACM transactions on networking* **8** (2000), no. 3 281–293.

[54] B. Langmead and S. L. Salzberg, *Fast gapped-read alignment with bowtie 2*, *Nature Methods* **9** (03, 2012) 357 EP –.

[55] N. Ahmed, K. Bertels, and Z. Al-Ars, *A comparison of seed-and-extend techniques in modern DNA read alignment algorithms*, in *Bioinformatics and Biomedicine (BIBM), 2016 IEEE International Conference on*, pp. 1421–1428, IEEE, 2016.

[56] M. Alser, Z. Bingöl, D. S. Cali, J. Kim, S. Ghose, C. Alkan, and O. Mutlu, *Accelerating genome analysis: A primer on an ongoing journey*, *IEEE Micro* **40** (2020), no. 5 65–75.

[57] L. Song, X. Qian, H. Li, and Y. Chen, *Pipelayer: A pipelined ReRAM-based accelerator for deep learning*, in *2017 IEEE international symposium on high performance computer architecture (HPCA)*, pp. 541–552, IEEE, 2017.

[58] S. Li, A. O. Glova, X. Hu, P. Gu, D. Niu, K. T. Malladi, H. Zheng, B. Brennan, and Y. Xie, *Scope: A stochastic computing engine for DRAM-based in-situ accelerator*, in *2018 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pp. 696–709, IEEE, 2018.

[59] M. Gao, J. Pu, X. Yang, M. Horowitz, and C. Kozyrakis, *TETRIS: Scalable and efficient neural network acceleration with 3D memory*, in *Proceedings of the Twenty-Second International Conference on Architectural Support for Programming Languages and Operating Systems*, pp. 751–764, 2017.

[60] E. B. Fernandez, W. A. Najjar, S. Lonardi, and J. Villarreal, *Multithreaded FPGA acceleration of DNA sequence mapping*, in *2012 IEEE Conference on High Performance Extreme Computing*, pp. 1–6, IEEE, 2012.

[61] E. B. Fernandez, J. Villarreal, S. Lonardi, and W. A. Najjar, *FHAST: FPGA-based acceleration of Bowtie in hardware*, *IEEE/ACM Transactions on Computational Biology and Bioinformatics (TCBB)* **12** (2015), no. 5 973–981.

[62] G. J. Manikandan, S. Huang, K. Rupnow, W.-M. W. Hwu, and D. Chen, *Acceleration of the pair-hmm algorithm for dna variant calling*, in *2016 IEEE 24th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, pp. 137–137, IEEE, 2016.

[63] D. Castells-Rufas, S. Marco-Sola, J. C. Moure, Q. Aguado, and A. Espinosa, *FPGA acceleration of pre-alignment filters for short read mapping with HLS*, *IEEE Access* **10** (2022) 22079–22100.

[64] B. S. C. Varma, K. Paul, M. Balakrishnan, and D. Lavenier, *FAssem: FPGA based acceleration of de novo genome assembly*, in *2013 IEEE 21st Annual International Symposium on Field-Programmable Custom Computing Machines*, pp. 173–176, IEEE, 2013.

[65] S. Huang, G. J. Manikandan, A. Ramachandran, K. Rupnow, W.-m. W. Hwu, and D. Chen, *Hardware acceleration of the pair-HMM algorithm for DNA variant calling*, in *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, pp. 275–284, 2017.

[66] Z. Bingöl, M. Alser, O. Mutlu, O. Ozturk, and C. Alkan, *GateKeeper-GPU: Fast and accurate pre-alignment filtering in short read mapping*, in *2021 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, pp. 209–209, IEEE, 2021.

[67] L. Hasan, Y. M. Khawaja, and A. Bais, *A systolic array architecture for the Smith-Waterman algorithm with high performance cell design.*, in *IADIS European Conf. Data Mining*, pp. 35–44, 2008.

[68] R. Kaplan, L. Yavits, R. Ginosar, and U. Weiser, *A resistive CAM Processing-in-Storage architecture for DNA sequence alignment*, *arXiv preprint arXiv:1701.04723* (2017).

[69] L. Yavits, S. Kvatinsky, A. Morad, and R. Ginosar, *Resistive associative processor*, *IEEE Computer Architecture Letters* **14** (2015), no. 2 148–151.

[70] S. Hamdioui, L. Xie, H. A. Du Nguyen, M. Taouil, K. Bertels, H. Corporaal, H. Jiao, F. Catthoor, D. Wouters, L. Eike, *et. al.*, *Memristor based computation-in-memory architecture for data-intensive applications*, in *2015 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 1718–1725, IEEE, 2015.

[71] S. Kvatinsky, G. Satat, N. Wald, E. G. Friedman, A. Kolodny, and U. C. Weiser, *Memristor-based material implication (IMPLY) logic: Design principles and methodologies*, *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* **22** (2013), no. 10 2054–2066.

[72] S. H. Pugsley, J. Jestes, R. Balasubramonian, V. Srinivasan, A. Buyuktosunoglu, A. Davis, and F. Li, *Comparing implementations of near-data computing with in-memory map reduce workloads*, *IEEE Micro* **34** (2014), no. 4 44–52.

[73] S. F. Altschul, W. Gish, W. Miller, E. W. Myers, and D. J. Lipman, *Basic local alignment search tool*, *Journal of molecular biology* **215** (1990), no. 3 403–410.

[74] Z. Ning, A. J. Cox, and J. C. Mullikin, *SSAHA: a fast search method for large DNA databases*, *Genome research* **11** (2001), no. 10 1725–1729.

[75] P. BLAST, "Paracel blast." `http://www.paracel.com/index.html`, 2015.

[76] J. Lancaster, J. Buhler, and R. D. Chamberlain, *Acceleration of ungapped extension in Mercury BLAST*, *Microprocessors and microsystems* **33** (2009), no. 4 281–289.

[77] C. Ling and K. Benkrid, *Design and implementation of a CUDA-compatible GPU-based core for gapped BLAST algorithm*, *Procedia Computer Science* **1** (2010), no. 1 495–504.

[78] S. Li, L. Liu, P. Gu, C. Xu, and Y. Xie, *NVSim-CAM: a circuit-level simulator for emerging nonvolatile memory based content-addressable memory*, in *Proceedings of the 35th International Conference on Computer-Aided Design*, p. 2, ACM, 2016.

[79] C. Xu, P.-Y. Chen, D. Niu, Y. Zheng, S. Yu, and Y. Xie, *Architecting 3D vertical resistive memory for next-generation storage systems*, in *Proceedings of the 2014 IEEE/ACM International Conference on Computer-Aided Design*, pp. 55–62, 2014.

[80] T. F. Smith and M. S. Waterman, *Identification of common molecular subsequences*, *Journal of molecular biology* **147** (1981), no. 1 195–197.

[81] S. Li, P. Chi, J. Zhao, K.-T. Cheng, and Y. Xie, *Leveraging nonvolatility for architecture design with emerging NVM*, in *Non-Volatile Memory System and Applications Symposium (NVMSA), 2015 IEEE*, pp. 1–5, IEEE, 2015.

[82] S. Matsunaga, S. Miura, H. Honjou, K. Kinoshita, S. Ikeda, T. Endoh, H. Ohno, and T. Hanyu, *A 3.14 um 2 4T-2MTJ-cell fully parallel TCAM based on nonvolatile logic-in-memory architecture*, in *Symposium on VLSI Circuits (VLSIC)*, pp. 44–45, 2012.

[83] A. Jacob, J. Lancaster, J. Buhler, B. Harris, and R. D. Chamberlain, *Mercury BLASTP: Accelerating protein sequence alignment*, *ACM Transactions on Reconfigurable Technology and Systems (TRETS)* (2008).

[84] X. Dong, C. Xu, N. Jouppi, and Y. Xie, *Nvsim: A circuit-level performance, energy, and area model for emerging non-volatile memory*, in *Emerging Memory Technologies*, pp. 15–50. Springer, 2014.

[85] Synopsys, "Design complier (dc)." `https://www.synopsys.com/implementation-and-signoff/rtl-synthesis-test.html`, 2017.

[86] J. D. Buhler, J. M. Lancaster, A. C. Jacob, R. D. Chamberlain, *et. al.*, *Mercury BLASTN: Faster DNA sequence comparison using a streaming hardware architecture*, *Proc. of Reconfigurable Systems Summer Institute* (2007).

[87] E. Sotiriades, C. Kozanitis, and A. Dollas, *Some initial results on hardware BLAST acceleration with a reconfigurable architecture*, in *Parallel and Distributed Processing Symposium, 2006. IPDPS 2006. 20th International*, pp. 8–pp, IEEE, 2006.

[88] ARM, *AMBA specification (rev 2.0)*, .

[89] Y. Lee, S. Kim, S. Hong, and J. Lee, *Skinflint DRAM system: Minimizing DRAM chip writes for low power*, in *High Performance Computer Architecture (HPCA2013), 2013 IEEE 19th International Symposium on*, pp. 25–34, IEEE, 2013.

[90] Samsung, "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`, 2017.

[91] A. Sainio, *NVDIMM: Changes are here so what's next*, *In-Memory Computing Summit* (2016).

[92] K. Oe, T. Nanri, and K. Okamura, *Feasibility study for building hybrid storage system consisting of non-volatile DIMM and SSD*, in *Computing and Networking (CANDAR), 2016 Fourth International Symposium on*, pp. 454–457, IEEE, 2016.

[93] Intel, "Intel's 3D XPoint technology products." `https://software.intel.com/en-us/articles/3d-xpoint-technology-products`, 2018.

[94] T. Ungerer and I. D. Fey, *Report on disruptive technologies for years 2020-2030*, .

[95] J. Ahn, S. Hong, S. Yoo, O. Mutlu, and K. Choi, *A scalable processing-in-memory accelerator for parallel graph processing*, *ACM SIGARCH Computer Architecture News* **43** (2016), no. 3 105–117.

[96] O. Kocberber, B. Grot, J. Picorel, B. Falsafi, K. Lim, and P. Ranganathan, *Meet the walkers: Accelerating index traversals for in-memory databases*, in *Proceedings of the 46th Annual IEEE/ACM International Symposium on Microarchitecture*, pp. 468–479, ACM, 2013.

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

[98] V. Seshadri, D. Lee, T. Mullins, H. Hassan, A. Boroumand, J. Kim, M. A. Kozuch, O. Mutlu, P. B. Gibbons, and T. C. Mowry, *Ambit: In-memory accelerator for bulk bitwise operations using commodity DRAM technology*, in *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture*, pp. 273–287, ACM, 2017.

[99] V. Zois, D. Gupta, V. J. Tsotras, W. A. Najjar, and J.-F. Roy, *Massively parallel skyline computation for processing-in-memory architectures*, in *Proceedings of the 27th International Conference on Parallel Architectures and Compilation Techniques*, p. 1, ACM, 2018.

[100] UPMEM, "UPMEM." `https://www.upmem.com/use-cases/`.

[101] D. H. Ponstingl, "SMALT." `http://www.sanger.ac.uk/science/tools/smalt-0`, 2016.

[102] Y. Kim, W. Yang, and O. Mutlu, *Ramulator: A fast and extensible DRAM simulator.*, *Computer Architecture Letters* **15** (2016), no. 1 45–49.

[103] Synopsys, "Design complier." `https://www.synopsys.com/support/training/rtl-synthesis/design-compiler-rtl-synthesis.html`, 2018.

[104] F. Technology, "28nm technology libraries." `https://www.faraday-tech.com/cn/category/BrowseByTechnology?method=browserCategory&tech=28nm&master.ipSearchForm.technology=28nm`.

[105] K. Chandrasekar, C. Weis, Y. Li, S. Goossens, M. Jung, O. Naji, B. Akesson, N. Wehn, and K. Goossens, *DRAMPower: Open-source DRAM power & energy estimation tool*, *URL: http://www. drampower. info* **22** (2012).

[106] N. P. Jouppi, A. B. Kahng, N. Muralimanohar, and V. Srinivas, *CACTI-IO: CACTI with off-chip power-area-timing models*, *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* **23** (2015), no. 7 1254–1267.

[107] Intel, "Intel Optane memory series 32GB M.2 80MM." `https://www.intel.com/content/www/us/en/products/memory-storage/optane-memory/optane-32gb-m-2-80mm.html`, 2018.

[108] J. Izraelevitz, J. Yang, L. Zhang, J. Kim, X. Liu, A. Memaripour, Y. J. Soh, Z. Wang, Y. Xu, S. R. Dulloor, J. Zhao, and S. Swanson, *Basic performance measurements of the Intel Optane DC persistent memory module*, *arXiv preprint arXiv:1903.05714* (2019).

[109] NCBI, "Genome database." `https://www.ncbi.nlm.nih.gov/genome`, 2018.

[110] M. A. DePristo, E. Banks, R. Poplin, K. V. Garimella, J. R. Maguire, C. Hartl, A. A. Philippakis, G. Del Angel, M. A. Rivas, M. Hanna, *et. al.*, *A framework for variation discovery and genotyping using next-generation DNA sequencing data*, *Nature genetics* **43** (2011), no. 5 491.

[111] D. Fujiki, A. Subramaniyan, T. Zhang, Y. Zeng, R. Das, D. Blaauw, and S. Narayanasamy, *GenAx: a genome sequencing accelerator*, in *Proceedings of the 45th Annual International Symposium on Computer Architecture*, pp. 69–82, IEEE Press, 2018.

[112] T. C. Pan, S. Misra, and S. Aluru, *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*, pp. 135–147, IEEE, 2018.

[113] R. Kaplan, L. Yavits, and R. Ginosasr, *BioSEAL: In-memory biological sequence alignment accelerator for large-scale genomic data*, in *Proceedings of the 13th ACM International Systems and Storage Conference*, pp. 36–48, 2020.

[114] W. Kwon, *ETRI extended memory pool system architecture using Gen-Z protocol*, in *2021 International Conference on Information and Communication Technology Convergence (ICTC)*, pp. 123–125, IEEE, 2021.

[115] K. Lim, J. Chang, T. Mudge, P. Ranganathan, S. K. Reinhardt, and T. F. Wenisch, *Disaggregated memory for expansion and sharing in blade servers*, *ACM SIGARCH computer architecture news* **37** (2009), no. 3 267–278.

[116] W. Kwon, C. Park, and M. Oh, *Gen-Z memory pool system architecture*, in *2020 International Conference on Information and Communication Technology Convergence (ICTC)*, pp. 1356–1360, IEEE, 2020.

[117] B. Benton, *CCIX, Gen-Z, OpenCAPI: overview & comparison*, in *OpenFabrics Workshop*, 2017.

[118] M. Bielski, C. Pinto, D. Raho, and R. Pacalet, *Survey on memory and devices disaggregation solutions for hpc systems*, in *2016 IEEE Intl Conference on Computational Science and Engineering (CSE) and IEEE Intl Conference on Embedded and Ubiquitous Computing (EUC) and 15th Intl Symposium on Distributed Computing and Applications for Business Engineering (DCABES)*, pp. 197–204, IEEE, 2016.

[119] C. Pinto, D. Syrivelis, M. Gazzetti, P. Koutsovasilis, A. Reale, K. Katrinis, and H. P. Hofstee, *ThymesisFlow: A software-defined, HW/SW co-designed interconnect stack for rack-scale memory disaggregation*, in *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pp. 868–880, IEEE, 2020.

[120] C. E. Link, "Compute Express Link specifications - revision 2.0." `https://www.computeexpresslink.org/download-the-specification`, Nov., 2020.

[121] C. E. Link, "Compute Express Link 2.0 white paper." `https://www.computeexpresslink.org/`, Nov., 2020.

[122] R. Iyer, V. De, R. Illikkal, D. Koufaty, B. Chitlur, A. Herdrich, M. Khellah, F. Hamzaoglu, and E. Karl, *Advances in microprocessor cache architectures over the last 25 years*, *IEEE Micro* **41** (2021), no. 6 78–88.

[123] T. Coughlin, *Digital storage and memory*, *Computer* **55** (2022), no. 1 20–29.

[124] O. Consortium, "OpenCAPI specification." `https://opencapi.org/`, Nov., 2021.

[125] G.-Z. Consortium, "Gen-Z specifications." `https://genzconsortium.org/specifications/`, Nov., 2021.

[126] B. Abali, R. J. Eickemeyer, H. Franke, C.-S. Li, and M. A. Taubenblatt, *Disaggregated and optically interconnected memory: when will it be cost effective?*, *arXiv preprint arXiv:1503.01416* (2015).

[127] S. Van Doren, *Hoti 2019: Compute Express Link*, in *2019 IEEE Symposium on High-Performance Interconnects (HOTI)*, pp. 18–18, IEEE Computer Society, 2019.

[128] H. Li, D. S. Berger, S. Novakovic, L. Hsu, D. Ernst, P. Zardoshti, M. Shah, I. Agarwal, M. Hill, M. Fontoura, *et. al.*, *First-generation memory disaggregation for cloud platforms*, *arXiv preprint arXiv:2203.00241* (2022).

[129] J. D. Leidel and Y. Chen, *HMC-Sim-2.0: A simulation platform for exploring custom memory cube operations*, in *2016 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, pp. 621–630, IEEE, 2016.

[130] A. Farmahini-Farahani, J. H. Ahn, K. Morrow, and N. S. Kim, *NDA: Near-DRAM acceleration architecture leveraging commodity DRAM devices and standard memory modules*, in *High Performance Computer Architecture (HPCA), 2015 IEEE 21st International Symposium on*, pp. 283–295, IEEE, 2015.

[131] P. Gu, X. Xie, Y. Ding, G. Chen, W. Zhang, D. Niu, and Y. Xie, *iPIM: Programmable in-memory image processing accelerator using near-bank architecture*, in *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*, pp. 804–817, IEEE, 2020.

[132] G. Singh, J. Gómez-Luna, G. Mariani, G. F. Oliveira, S. Corda, S. Stuijk, O. Mutlu, and H. Corporaal, *Napel: Near-memory computing application performance prediction via ensemble learning*, in *2019 56th ACM/IEEE Design Automation Conference (DAC)*, pp. 1–6, IEEE, 2019.

[133] C. Consortium, "Ccix specification." https://www.ccixconsortium.com/, Nov., 2021.

[134] S. W. Keckler, W. J. Dally, B. Khailany, M. Garland, and D. Glasco, *GPUs and the future of parallel computing*, *IEEE micro* **31** (2011), no. 5 7–17.