applied sciences

MDPI

Article

SARDIMM: High-Speed Near-Memory Processing Architecture for Synthetic Aperture Radar Imaging

Haechan Kim [1], Jinmoo Heo [2], Seongjoo Lee [3,4] and Yunho Jung [1,2,*]

[1] School of Electronics and Information Engineering, Korea Aerospace University, Goyang-si 10540, Republic of Korea; ft0241@kau.kr
[2] Department of Smart Air Mobility, Korea Aerospace University, Goyang-si 10540, Republic of Korea; jmz416@kau.kr
[3] Department of Electrical Engineering, Sejong University, Seoul 05006, Republic of Korea; seongjoo@sejong.ac.kr
[4] Department of Convergence Engineering of Intelligent Drone, Sejong University, Seoul 05006, Republic of Korea
* Correspondence: yjung@kau.ac.kr; Tel.: +82-2-300-0133

Abstract: The range-Doppler algorithm (RDA), a key technique for generating synthetic aperture radar (SAR) images, offers high-resolution images but requires significant memory resources and involves complex signal processing. Moreover, the multitude of fast Fourier transform (FFT) and inverse fast Fourier transform (IFFT) operations in RDA necessitates high bandwidth and lacks data reuse, leading to bottlenecks. This paper introduces a synthetic aperture radar dual in-line memory module (SARDIMM), which executes RDA operations near memory via near-memory processing (NMP), thereby effectively reducing memory accesses, execution time, and energy consumption. The embedded NMP module in SARDIMM optionally supports a combination of FFT, IFFT, and matched filter operations of the RDA for range and azimuth compression. The operator within the NMP module accelerates the FFT by performing two radix-2 single butterfly operations in parallel. The NMP module was implemented and validated on a Xilinx UltraScale+ field-programmable gate array (FPGA) using Verilog-HDL. The acceleration performance of RDA for images of various sizes was evaluated through a simulator modified with gem5 and DRAMSim3 and achieved a 6.34–6.93× speedup and 41.9–48.2% energy savings.

Keywords: synthetic aperture radar; range-Doppler algorithm; fast Fourier transform; memory bottleneck; near-memory processing; acceleration

Citation: Kim, H.; Heo, J.; Lee, S.; Jung, Y. SARDIMM: High-Speed Near-Memory Processing Architecture for Synthetic Aperture Radar Imaging. *Appl. Sci.* **2024**, *14*, 7601. https://doi.org/10.3390/app14177601

Academic Editor: Atsushi Mase

Received: 23 July 2024
Revised: 25 August 2024
Accepted: 26 August 2024
Published: 28 August 2024

1. Introduction

Synthetic aperture radar (SAR) is an active radar system that employs radio waves to generate high-resolution images of large areas, such as land or ocean surfaces. It has applications in various fields, including military and civilian domains [1,2]. Achieving high performance in radar systems depends on obtaining high-quality images [3,4]. A high azimuth resolution is a prerequisite for obtaining such images. Although the azimuth resolution typically increases with the antenna size, there are limits to enlarging antennas. However, SAR systems typically mount radar on mobile platforms such as aircraft. As the platform moves, the radar continuously acquires data, enabling the generation of high-resolution images through pulse compression processes in both the range and azimuth directions. Therefore, SAR systems can achieve high resolution using small antennas [5,6].

Optical sensors are widely used in remote sensing and primarily detect objects using visible, infrared, and ultraviolet light. Examples include cameras, telescopes, and infrared cameras. However, optical sensors may be sensitive to adverse weather conditions such as fog, rain, snow, and clouds, leading to performance degradation. Additionally, insufficient light at night may hinder image generation. In contrast, SAR employs electromagnetic

waves to detect the range and velocity of the objects. Therefore, unaffected by visual features or lighting conditions, SAR can produce high-quality images regardless of the time of day or prevailing weather conditions [7,8]. These advantages render SAR particularly valuable for surveillance and reconnaissance applications. In these fields, real-time image acquisition and rapid response are crucial [9]. However, real-time SAR image generation requires complex signal processing operations, which pose a challenge owing to the large volume of raw data typically involved.

Algorithms for efficiently generating SAR images include the range-Doppler algorithm (RDA) [10–14], chirp scaling algorithm (CSA) [15–17], polar format algorithm (PFA) [18–20], and back-projection algorithm (BPA) [21–23]. Among these, the RDA, which has the most intuitive physical concepts and the best trade-off between image quality and computational efficiency, is the most widely used. The RDA mainly comprises a range compression process, a range cell migration correction (RCMC) process, and an azimuth compression process. Because both range and azimuth compression processes primarily operate in the frequency domain, they involve fast Fourier transform (FFT) and inverse fast Fourier transform (IFFT) operations. The RDA requires a long execution time because several FFT and IFFT operations are repeatedly performed on two-dimensional (2D) images. These operations are crucial for accurately reconstructing the amplitude and phase of the signals reflected from each point on the ground, which is essential for high-resolution image generation. However, processing a large volume of raw data with the low data reuse rate inherent in FFT operations necessitates a significant number of memory requests. Additionally, FFT and IFFT operations demand high memory bandwidth, leading to memory bottlenecks that not only slow down execution but also contribute to significant energy consumption. Given that SAR systems must handle large volumes of data to generate high-resolution ground images, optimizing both the performance and energy efficiency of these operations is critical. In particular, the numerous memory accesses required by multidimensional Fourier transforms, a key part of SAR signal processing, pose a substantial challenge. Each memory access contributes to the overall energy consumption, which can be considerable given the volume of data processed. Therefore, in addition to accelerating the FFT in the RDA, optimizing the energy efficiency of these memory-intensive processes is equally important. Addressing these issues can lead to more sustainable and cost-effective SAR imaging, aligning with the broader goal of improving both performance and resource utilization in SAR technology.

Figure 1 illustrates the RDA 2D-FFT operation within the roofline model to demonstrate that it is memory-bound. The roofline model serves as a framework for analyzing performance constraints in computer systems, focusing on the balance between memory bandwidth and operations per second, and particularly emphasizing floating-point operations [24]. It is worth noting that the approach we used to apply the FFT within the roofline model is similar to the methodology presented in [25]. However, while [25] utilized a GPU platform for their analysis, our experiments were conducted on a CPU platform, specifically chosen to align with the DIMM-based memory configuration of our system. Despite the difference in hardware platforms, our results are consistent with those of [25], further validating our methodology. To construct the roofline model, we used a PyTorch profiler to assess the 2D-FFT performance. The CPU used in the experiments was an Intel(R) Core(TM) i7-10700K. It features two memory channels, each equipped with DDR4-2933 dual in-line memory modules (DIMMs), providing a theoretical peak floating-point performance of 486.4 GFLOPs/s and a peak memory bandwidth of 45.8 GB/s [26]. The observed performance of the 8192×8192 2D-FFT was 55.2 GFLOPs/s, representing only 11% of the peak floating-point performance. Figure 1 shows that the 2D-FFT is located in the memory-bound area of the roofline model. The numerous memory requests and bottlenecks associated with the 2D-FFT, which is a memory-constrained operation, can be mitigated through near-memory processing (NMP), a technique that has gained attention in recent research.

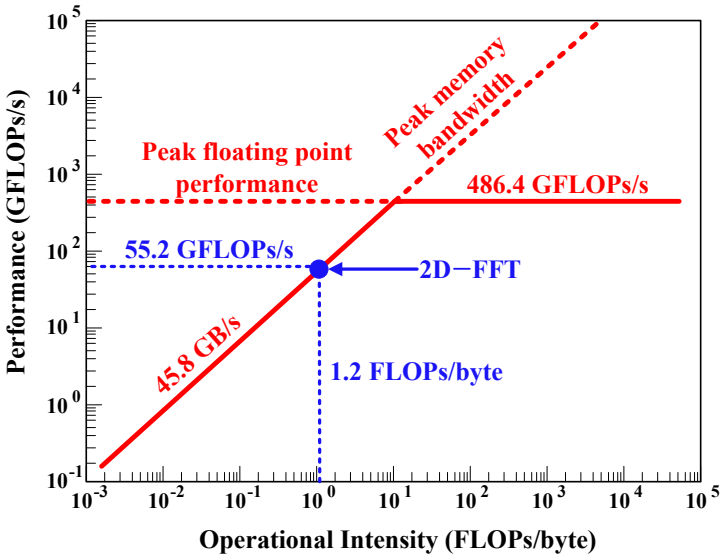

Figure 1. Roofline model of the 8192 × 8192 2D-FFT.

Modern computer systems have limitations in terms of improving performance owing to memory bandwidth constraints and delays caused by data movement. NMP is one of the methods proposed for overcoming these limitations. NMP is a technology that minimizes unnecessary data movement, improves performance by processing operations in a memory area close to the data, and can effectively solve memory-bound challenges. Although various NMP-related studies primarily designed to accelerate artificial intelligence (AI) computations, such as vector addition and matrix multiplication, exist [27–31], no studies have focused on accelerating FFT for SAR image generation. Considering the critical role of FFT in SAR image generation and its inherent memory-bound nature, applying NMP to algorithms for radar signal processing holds the potential for significant improvements in execution time and energy efficiency. Therefore, exploring the application of NMP in SAR image generation to achieve these improvements represents a promising direction for future research. In this study, we propose a synthetic aperture radar dual in-line memory module (SARDIMM), which effectively improves the performance of the RDA by applying rank-level parallel NMP to FFT, IFFT, and matched filter operations, which are core processes in the RDA. To the best of our knowledge, this is the first instance where SAR image generation has been implemented using NMP. Our research demonstrates that by applying NMP to SAR applications, we can achieve significant gains in both speed and power. We believe that these contributions are valuable in advancing the current state of SAR imaging technology.

SARDIMM achieves a significant reduction in unnecessary data movement, improved execution time, and reduced power consumption by executing specific RDA operations in NMP modules located near memory. Because the NMP module is located between the memory and the memory controller, SARDIMM uses a commodity dynamic random-access memory (DRAM) chip without additional modifications. The NMP module selectively supports a combination of FFT, IFFT, and matched filter operations to perform the RDA range and azimuth compression. In the FFT and IFFT algorithms, the operation is performed sequentially by dividing it into several stages, and data ordering for each stage is essential. SARDIMM does not perform such ordering using the host processor but instead uses the NMP module. In addition, the NMP module effectively reduces the execution time by performing an FFT using two single butterfly operation modules in parallel.

In this study, the performance of SARDIMM was evaluated by experimenting with RDAs on images of various sizes. The evaluation results show that SARDIMM effectively accelerated the RDA operation. Compared with the baseline system, SARDIMM achieved a maximum speedup of 6.926× and an energy saving of 48.2%. The remainder of this paper

is structured as follows: Section 2 provides a background that explains the concepts of RDA and NMP. Section 3 describes the hardware architecture and execution flow of the proposed SARDIMM. Section 4 presents the acceleration performance evaluation results of the SARDIMM for RDA operations on various images, and Section 5 concludes the paper.

2. Background

In large-scale data processing applications, addressing the issue of bottlenecks is crucial. This problem arises due to the high data transfer rates between memory and processors, and it can be solved through NMP. With the recent increase in large-scale AI applications, many studies have focused on using NMP to resolve the significant issues related to execution time delays and high energy consumption caused by bottlenecks. For example, refs. [27,28,30] highlight the bottlenecks in the deep learning recommendation model (DLRM). Specifically, they demonstrate performance improvements by performing the embedding lookup operations and vector addition operations near the memory, which require substantial data transfer. Refs. [29,31] address bottlenecks in graph neural networks (GNNs), identifying sparse aggregation, combination computation, and matrix multiplication operations as the causes, and propose NMP as a solution.

RDA, a SAR imaging algorithm, also requires processing large volumes of data, which typically involves substantial data transfer from memory to computational units. Recognizing this similarity, we see the potential of applying NMP to SAR imaging to achieve significant improvements in speedup and energy efficiency. Unlike previous studies focused on AI applications, our research uniquely implements critical operations in RDA such as FFT, IFFT, and matched filtering using an NMP architecture. This approach represents a novel application of NMP technology in the domain of SAR imaging, differentiating our work from existing studies and demonstrating the versatility and potential of NMP. The following sections will provide a detailed explanation of the RDA and NMP, including their principles, challenges, and the innovations presented in this study.

2.1. Range-Doppler Algorithm

The RDA, an algorithm originally created for radar signal processing, was developed with a primary focus on maritime surveillance. An RDA can process signals received from radar to obtain high-quality images, furnishing detailed and precise information about the detected objects. Consequently, the RDA is an important part of radar technology, facilitating reliable and efficient object detection and tracking in various applications, and has been widely studied [10–14].

Figure 2 shows the operation flow of the RDA for SAR image generation. The RDA consists of range compression, RCMC, and azimuth compression, and it is efficient by performing calculations in the frequency domain for both range and azimuth. Range compression proceeds with FFT, followed by range-matched filtering and IFFT in the range direction. Matched filtering involves multiplying the received signal by the reference signal, enhancing the desired signal, and minimizing the noise. Azimuth compression follows a process similar to range compression but includes an additional RCMC operation after azimuth FFT. Range cell migration (RCM) refers to the continuous change in the range between the radar sensor and the target during target detection using SAR. As these changes contribute to low-quality SAR images, the RDA corrects the RCM through RCMC using sinc interpolation to obtain high-quality SAR images.

The RDA is characterized by the inclusion of multiple FFT and IFFT operations on 2D images in both the range and azimuth directions. The radix-2 FFT algorithm, first introduced in 1963, is widely utilized in signal processing as a foundational method [32]. In this study, the radix-2 algorithm was selected for its fundamental yet highly flexible approach to FFT computation. Additionally, it is more suitable than radix-4 or radix-8 for supporting variable FFT lengths, as it allows lengths that are powers of 2, ensuring both versatility and efficiency. The butterfly structure at the radix-2 FFT algorithm receives two inputs and performs arithmetic operations, such as addition, subtraction, and multiplication

to generate two outputs. The number of butterfly operations increases proportionally as the length N of the FFT. Because of the substantial volume of raw input data required for the RDA in SAR image generation, performing FFT operations with extensive lengths, denoted as N, becomes imperative. This translates into numerous butterfly operations during the FFT process. However, FFTs with numerous butterfly operations require a significant number of memory requests, resulting in extended execution times. Hence, the acceleration of FFT operations is paramount for enhancing the overall performance of the RDA.

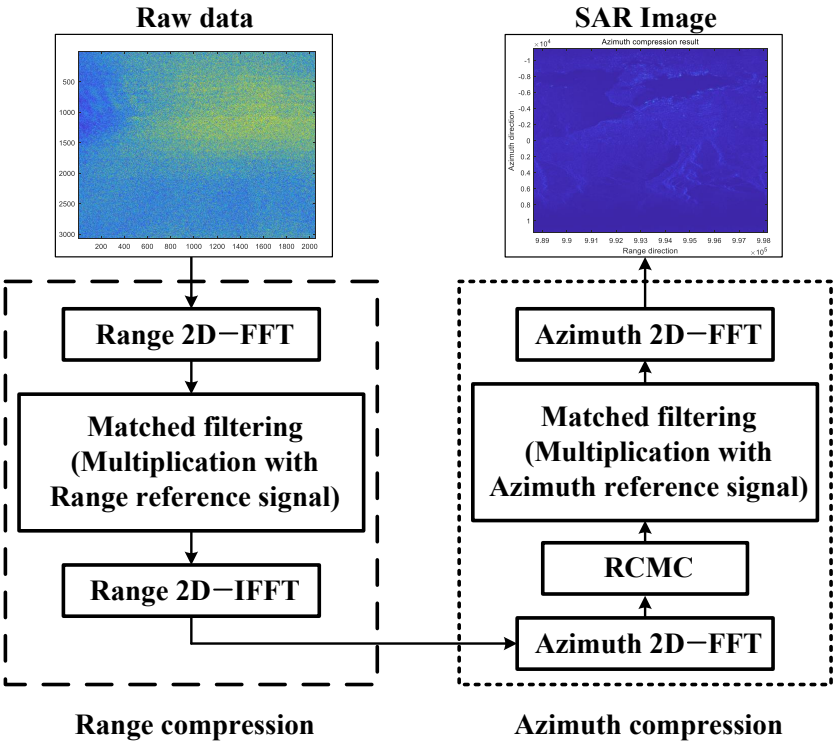

Figure 2. Operation flow of the RDA.

2.2. Near-Memory Processing

The von Neumann architecture, the foundation of modern computer architecture, comprises a CPU, memory, and program structure. In this architecture, the CPU reads programs and data from the same memory space, executes the programs, and stores the results back into memory. This process is based on the sequential fetching and execution of instructions from memory. However, this sequential execution method creates a problem known as the von Neumann bottleneck, which is caused by the limited data transfer rate between the CPU and memory and unnecessary data movement. This bottleneck is particularly prominent in data-intensive algorithms that process large volumes of data with low reuse rates, such as deep learning recommendation models (DLRMs) [28,33] and graph neural networks (GNNs) [29,34]. Additionally, modern memory systems consist of multiple ranks connected to a memory controller that share a single data path. Because the data path can transmit only one rank and data at a time, there is a bandwidth limit even when multiple DIMMs are used to increase the number of ranks. Consequently, this limitation on memory bandwidth constrains the overall system peak performance, despite improvements in CPU performance.

Processing in memory (PIM) and NMP have the potential to overcome the limitations of modern computers and memory systems. Figure 3 illustrates conventional memory semiconductors and memory semiconductors integrated with PIM and NMP technologies. A PIM semiconductor combines computational logic and memory cells on a single chip, whereas an NMP semiconductor features separate chips for computational logic and

memory cells. Instead of transferring data to the CPU for processing during memory read and write operations, the PIM and NMP technologies enable data processing directly inside or near the memory. This minimizes unnecessary data movement and enhances the efficiency of data processing. Furthermore, employing PIM or NMP modules per bank allows for parallel data handling by the bank, employing modules per rank enables parallel data handling by rank, thereby expanding the bandwidth. Owing to these capabilities, the PIM [25,35–38] and NMP [27–31] technologies have garnered significant attention in recent years and are promising for effectively alleviating memory bottlenecks.

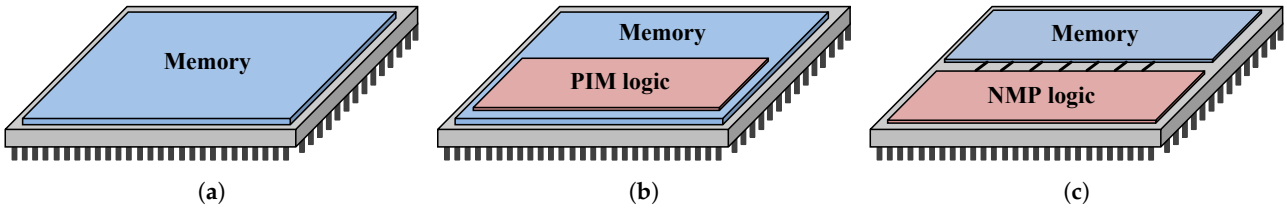

Figure 3. Architecture of memory semiconductors: (**a**) commodity memory semiconductor; (**b**) processing-in-memory (PIM) memory semiconductor; (**c**) NMP memory semiconductor.

Previous studies have indicated that existing memory cells require modification to implement PIM-related technologies in real systems [25,35–38]. However, integrating PIM technology into existing commercial memory systems poses challenges, because the current general protocol defined by the Joint Electron Device Engineering Council (JEDEC) [39] does not support modified memory cells. Even if the JEDEC were to define a protocol for PIM in the future, it is anticipated that applying existing research and development to that protocol would be complex. In contrast, the NMP technology is relatively compatible with existing systems because it adds a separate processor near the memory without altering the memory cells. Therefore, SARDIMM is designed to manage NMP operations as standard memory write requests and to adhere to a standard DRAM interface, thereby facilitating its integration into commercial memory systems.

3. SARDIMM: The Proposed NMP Architecture for RDA

As shown in Figure 4, the proposed SARDIMM performs FFT, reference signal multiplication, and IFFT, which are core operations in the RDA range and azimuth compression. In other words, all the operations except the RCMC and transposition of the RDA are processed near the memory, whereas the RCMC and transposition are performed by the host processor. The RDA range compression and azimuth compression involve performing an IFFT after the FFT. The SARDIMM applies DIF to the FFT and decimation in time (DIT) to the IFFT, allowing the FFT output to be used as input for the IFFT without bit-reverse ordering. The NMP module that accelerates the RDA is positioned in the buffer device, and the memory adopts dual-rank DIMMs, which are commonly used in general computer systems and each DRAM chip has a capacity of 8 GB.

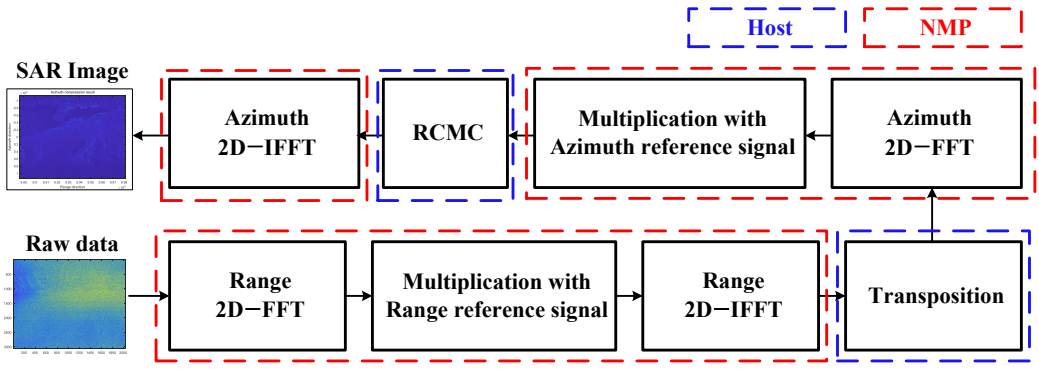

Figure 4. RDA flow diagram of the proposed SARDIMM for SAR image generation.

The data type supported by the SARDIMM is the half-precision floating-point format (FP16), and the data path size is 64 bit. The data utilized for the RDA operations comprise complex numbers with real and imaginary parts. Since each datum consisting of real and imaginary parts is represented as FP16, one complex datum can be represented as 32 bit. The choice of FP16 over single-precision floating-point format (FP32) is driven by the need to perform computations efficiently within limited hardware resources. FP16 uses half the memory compared to FP32, which is crucial when dealing with large datasets or complex models. This reduced memory usage allows for more data to be loaded into memory, contributing to performance improvements. The SARDIMM conducts an FFT by employing a single butterfly structure. In a typical computer system, two data points are necessary for a single butterfly operation, which requires the loading of 32-bit data twice. However, with SARDIMM employing NMP, a single butterfly operation can be executed once by directly reading 64-bit data containing two 32-bit data points. Additionally, SARDIMM stores the result of each FFT stage in the static random-access memory (SRAM) within the NMP module, rather than in memory. The result of the previous stage is then read, and the next stage is performed. These features greatly reduce the number of memory accesses, thereby improving performance.

3.1. Overview of the Proposed SARDIMM Architecture

Figure 5 illustrates the system architecture of the proposed SARDIMM. SARDIMM is a rank-level parallel structure, in which one NMP module is inserted per rank. Therefore, the proposed architecture incorporates two NMP modules operating in parallel per DIMM, and the level of parallelism scales with the number of DIMMs. For instance, in the RDA processing of an $N \times N$ image, an N-point FFT must be executed N times. If the system comprises M SARDIMMs, $2M$ parallel N-point FFTs are conducted $N/2M$ times.

To operate the NMP module, the host processor must generate and transmit the NMP instructions to the NMP module. The host and memory exchange data via the C/A and the DQ paths. The C/A path transmits commands and addresses, and the DQ path transmits data for reading or writing. The NMP module is located between the host and the memory, and it exchanges data through these two paths. Hence, the NMP instruction must be transmitted through one of these two paths. The instructions must contain sufficient information for the operation, but the C/A path has a relatively small bandwidth; therefore, the amount of information that can be contained is limited. On the other hand, the DQ path has a larger bandwidth than the C/A path and can include a large amount of information. For this reason, previous NMP studies have utilized the DQ path to transmit instructions, ensuring that it includes the necessary operational details [27,28,30]. Similarly, SARDIMM employs a standard memory write request to transmit an NMP instruction to the DQ path, where the NMP module that receives the instruction decodes it. Subsequently, the data are read/written in the buffer, and RDA calculations are executed through computational logic in the NMP module.

Figure 5 shows the architecture of the NMP module responsible for executing RDA operations. The instruction buffer, which is 0.016 KB SRAM, can accommodate up to two NMP instructions. As SARDIMM supports up to 65,536-point FFTs, both the reference signal buffer and the data buffer, each with a capacity of 262 KB SRAM, can store up to 65,536 complex data. The twiddle factor buffer, with a capacity of 131 KB SRAM, can hold up to 32,768 twiddle factors for single butterfly operations. The status register stores the status of the NMP module, which is used to determine the start and end of NMP operations. When the NMP controller receives the NMP start signal in the status register, it initiates the NMP operation. The NMP controller orchestrates various tasks, including generating buffer addresses; selecting the appropriate multiplexer (MUX) for FFT, IFFT, and reference signal multiplication; and issuing an NMP end signal to mark the completion of the NMP operation. The decoder extracts essential information for the NMP operation, such as Opcode and N-point, from the NMP instructions read from the instruction buffer. The floating-point butterfly (FPBF) module selectively executes the FFT, IFFT, and reference

signal multiplication based on the data extracted from the buffers and the MUX selection signal provided by the NMP controller. The results of each FFT stage calculated by the FPBF module are stored in the data buffer.

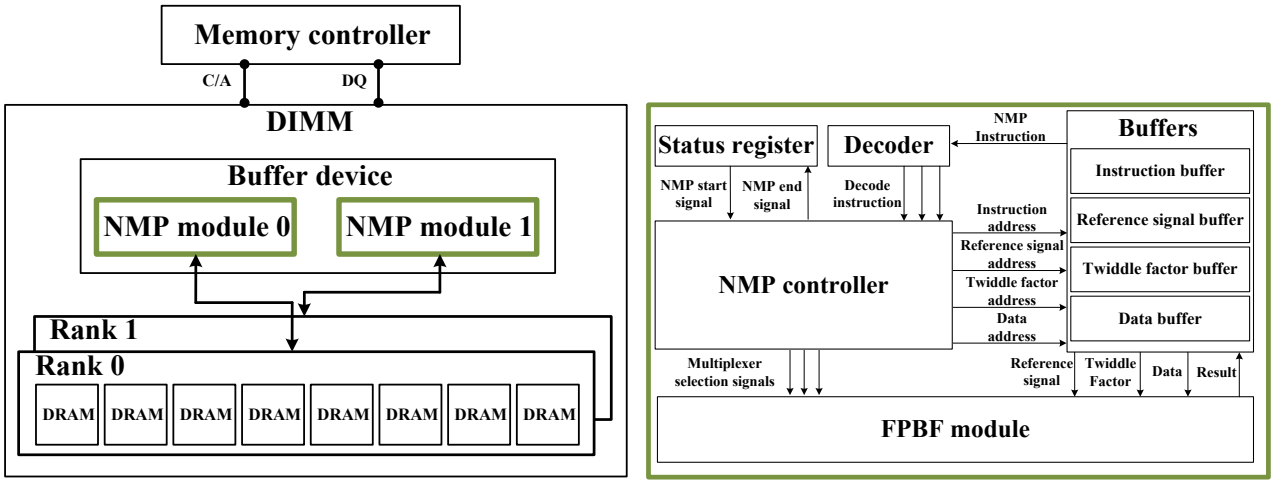

Figure 5. Hardware architecture of the proposed system: (**a**) SARDIMM system architecture; (**b**) NMP module architecture.

3.2. FPBF Module

Figure 6 shows the architecture of the FPBF module, which is responsible for the floating-point butterfly operations. This module executes two floating-point butterfly operations in parallel. Each operator receives two 32-bit complex data inputs and a twiddle factor and performs a single butterfly operation using a floating-point adder and multiplier. In addition, the FPBF module supports the IFFT through IQ swapping and floating-point division. IQ swapping involves exchanging data in the real and imaginary parts that are applied to both the input and output of the FFT. Furthermore, the IFFT is implemented by dividing the result of each stage of the FFT by two.

There are four types of MUXs, labeled A, B, C, and D, within the FPBF module, each supporting four different behaviors: normal FFT, normal IFFT, FFT followed by reference signal multiplication, and reference signal multiplication followed by IFFT. MUX A determines whether to perform a normal IFFT or an IFFT after the reference signal multiplication. MUX B selects between the FFT and IFFT operations. MUX C decides between the normal FFT and reference signal multiplication after the FFT. Finally, MUX D determines whether IQ swaps the FFT results. Table 1 details the operations conducted by the FPBF module based on the control signal of each MUX. Note that the control signal for MUX D is set to 1 during the final stage of the IFFT process.

Table 1. Operation of the FPBF module according to the MUX control signals.

Control Signal			Operation of the FPBF Module
MUX A	MUX B	MUX C	
0	0	0	Normal FFT
0	1	0	Normal IFFT
0	0	1	FFT & reference signal multiplication
1	1	0	Reference signal multiplication & IFFT

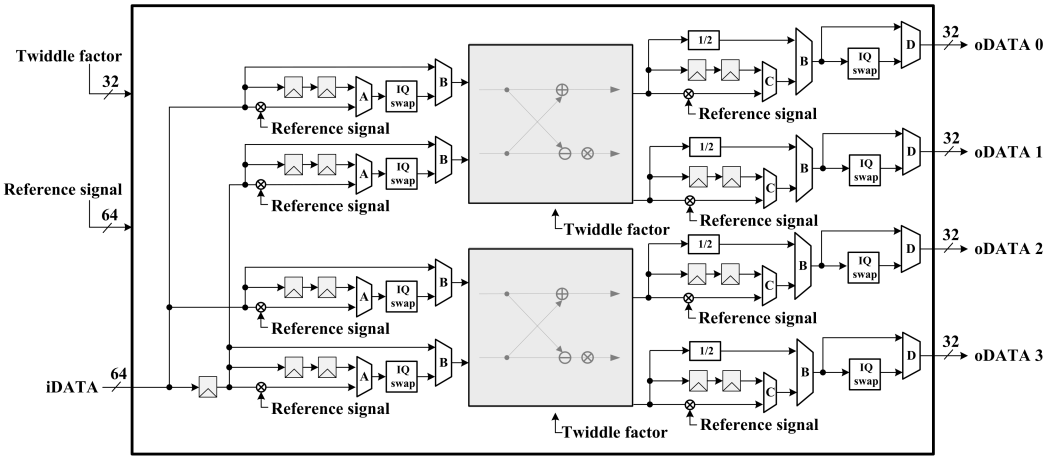

Figure 6. Hardware architecture of the FPBF module.

3.3. Data Storage Order for FFT Operation

Figure 7 shows the operation flow and state of the data buffer during the performance of a 16-point DIF FFT using an FPBF module. Figure 7a depicts the data buffer, sequentially storing 16 input data points from x0 to x15 for the first stage of the 16-point DIF FFT. The width of the data buffer is 64 bits, capable of storing two 32-bit complex number data within a storage space corresponding to one address. Initially, for the butterfly operation, 64-bit Rdata0 and Rdata1 are read from the data buffer across the two clocks. Rdata0 corresponds to input data x0 and x1 in the first stage, while Rdata1 corresponds to x8 and x9. In the butterfly operation of the first stage, x0 and x8, as well as x1 and x9, should be processed as pairs. Thus, as depicted in Figure 7b, the most significant 32 bits from both Rdata0 and Rdata1 are input to the first operator, while the least significant 32 bits from both Rdata0 and Rdata1 are input to the second operator.The first operator's high path output is represented as y0, and the low path output as y8, whereas the second operator's high path output is denoted as y1, and the low path output as y9. The FPBF module separately concatenates the outputs of the high and low paths. Subsequently, two data points are selected individually per clock cycle through the toggle signal, and Wdata is written into the data buffer. The data buffer, utilizing the dual-port SRAM, reads the input data of the first stage while simultaneously writing the output. As depicted in Figure 7c, the output data of the first stage are sequentially stored in the data buffer from y0 to y15, as shown in Figure 7a. This process ensures that SARDIMM sequentially stores the results in the data buffer at each stage of the FFT to facilitate the correct reading of the input data for the subsequent stage.

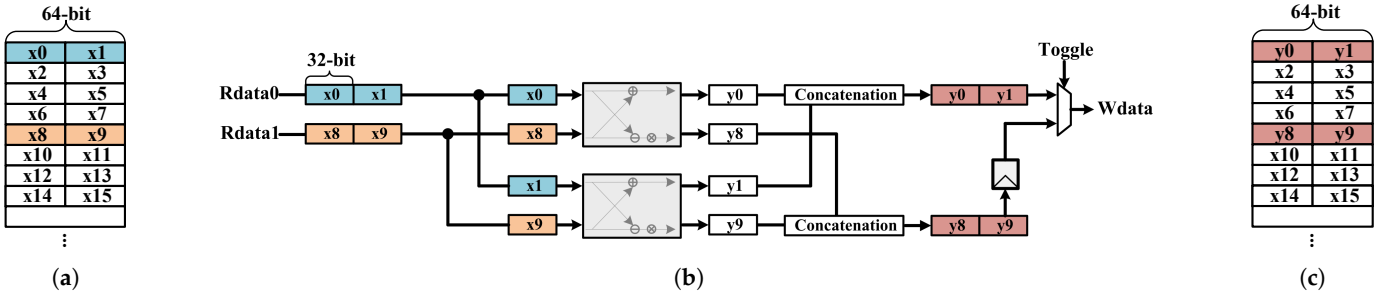

Figure 7. Representation of the FPBF module operation for stage 1 of the 16-point DIF FFT: (**a**) data buffer storing stage 1 input data; (**b**) simplified block diagram of the FPBF module; (**c**) data buffer storing part of stage 1 output data.

3.4. Execution Flow

The execution flow of RDA with a SARDIMM consists of five main steps:

1. Range FFT, range reference signal multiplication, and range IFFT in the NMP module;
2. Transpose the range compression result in the host processor;
3. Azimuth FFT in the NMP module;
4. RCMC in the host processor;
5. Azimuth reference signal multiplication and azimuth IFFT in the NMP module.

The NMP instructions for RDA include FFT and IFFT instructions, each containing Opcode, *N*-point, and RM. The Opcode distinguishes between FFT and IFFT instructions, whereas the *N*-point indicates the FFT length and number of stages. The NMP controller generates addresses to read the data required for each FFT stage accordingly. The RM determines whether reference multiplication is enabled or disabled. SARDIMM performs RDA using appropriate NMP instructions. First, the FFT and IFFT instructions are sent to the NMP module for range compression. When SARDIMM completes the range compression, the host processor receives the NMP end signal and reads the compression result. The host processor then transposes the results for azimuth compression. To handle transposition within the NMP module, data sharing between ranks would be necessary. However, the SARDIMM architecture does not support data sharing between ranks. Therefore, the host processor is responsible for performing the transposition. Second, an FFT instruction is sent for the azimuth FFT. After completion, the host processor reads the FFT results for RCMC. Finally, an IFFT instruction is sent to finalize azimuth compression. The detailed execution flow for each phase of the NMP operation is as follows:

1. The host processor initializes the buffer data in the NMP module;
2. The host processor sends NMP instructions to the NMP module;
3. The host processor sets the status register in the NMP module;
4. The NMP module operates and loads the result.

Figure 8 shows the flow of NMP operation in the SARDIMM system. Initially, the host processor stores the data necessary for operation, including the RDA input image, reference signal, and twiddle factor, in the NMP module buffer. SARDIMM employs a memory-mapped I/O method to distinguish the buffer addresses from memory addresses. After initialization, it transmits the corresponding NMP instruction for the operation. At this stage, SARDIMM must send instructions to each NMP module simultaneously, as the NMP modules operate in rank-level parallelism.

In a traditional memory system, all the ranks on the same channel as the memory controller are connected by sharing the DQ and C/A paths. Therefore, when data are sent to one rank, the same data are also sent to the other ranks, but the other ranks ignore the data. This feature of the memory system allows SARDIMM to simultaneously send the same NMP instruction to all NMP modules on the same channel. When the host processor issues a command to write the NMP instruction to the instruction buffer, the NMP module responds to it regardless of the rank [29].

After transmitting an NMP instruction, the status register is set. This is also performed simultaneously by all the NMP modules. Upon checking the NMP start signal in the status register, the NMP controller begins the NMP operation. The controller starts reading the data from the instruction buffer and decodes the first instruction to obtain information regarding the Opcode, *N*-point, and RM. Next, it performs FFT or IFFT, with or without reference signal multiplication, until the end of the last stage and stores the result of the operation in the data buffer. When the last stage is completed, the next instruction is decoded to perform the next operation. If there is no next instruction, the status register is set to the NMP end state. After confirming that the status register is in the end state, the host processor can read the results of the operation stored in the data buffer.

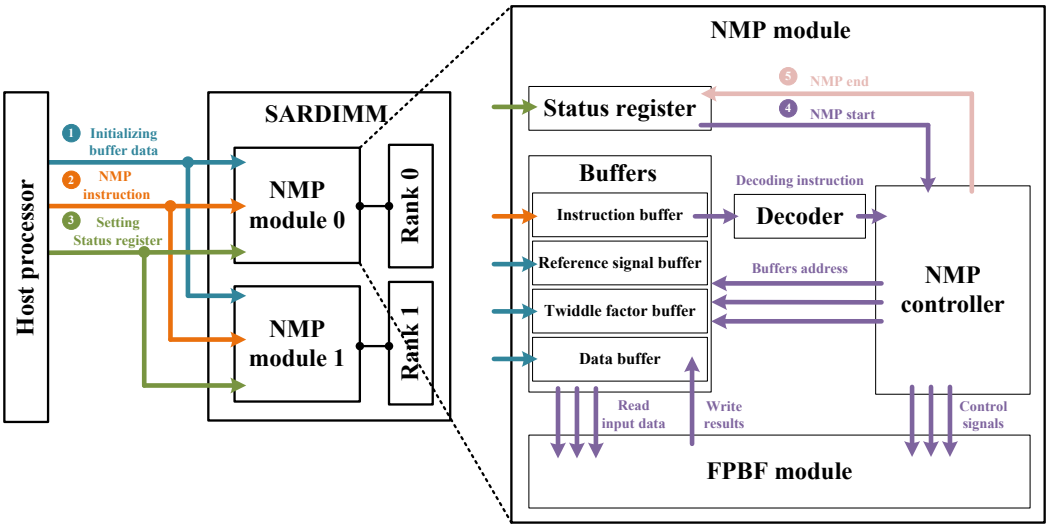

Figure 8. NMP operation flow of the SARDIMM system.

4. Evaluation

To evaluate the acceleration performance of the proposed system, the SARDIMM was modeled by modifying the cycle-level simulators gem5 [40] and DRAMsim3 [41], which are capable of measuring the execution time and energy according to memory requests. The baseline system used for performance evaluation comprised an Intel x86 CPU operating at a clock speed of 3 GHz. This processor was chosen due to its widespread use in standard computing environments, making it a suitable reference for comparison. For memory, the system employed a single-channel dual-rank x8 DDR4-2666 DIMM. This memory module was selected for its balance between speed and capacity, providing enough bandwidth and storage for intensive data processing tasks. The memory timing parameters used were those defined by the JEDEC [39] ensuring standardization and reliability in the memory performance. These parameters include crucial metrics such as column address strobe (CAS) latency, row address strobe (RAS) to CAS delay, and RAS precharge time, which together define the speed and efficiency of memory access and data retrieval processes. This configuration provides a standardized platform for evaluating the performance improvements achieved by the proposed SARDIMM architecture. The RDA, the application under evaluation, was modeled in C++ with both the normal version running on the baseline system and the NMP version running on the SARDIMM system. A performance comparison between the baseline system and the SARDIMM was performed by running the compiled executables in a simulator. Input images of various sizes were utilized for SAR image generation in the experiments. To address the need for a more robust evaluation, we used actual SAR data collected from the RADARSAT-1 satellite, specifically from 16 June 2002. This dataset includes high-resolution images of Vancouver, Canada, from RADARSAT-1's Fine Beam2 [42]. For our experiments, we upscaled this dataset to create larger SAR images, thereby simulating various sizes and types of SAR data.

4.1. SARDIMM without RCMC

To begin, we measured the RDA execution time on images of different sizes on the baseline system. Figure 9 presents the execution time ratios of different RDA operations, including range compression, azimuth FFT, RCMC, azimuth reference signal multiplication, and azimuth IFFT. Notably, RCMC exhibited a significantly lower percentage, averaging 0.8%, compared with the other operations. In contrast, range compression averaged 49.4%, and azimuth compression (excluding RCMC) averaged 49.8%, collectively constituting the majority of the execution time. This implies that when a module to accelerate the RCMC is inserted into the SARDIMM, the overhead due to the increase in area is greater than

the gain in performance. Consequently, the SARDIMM accelerates all RDA operations except RCMC.

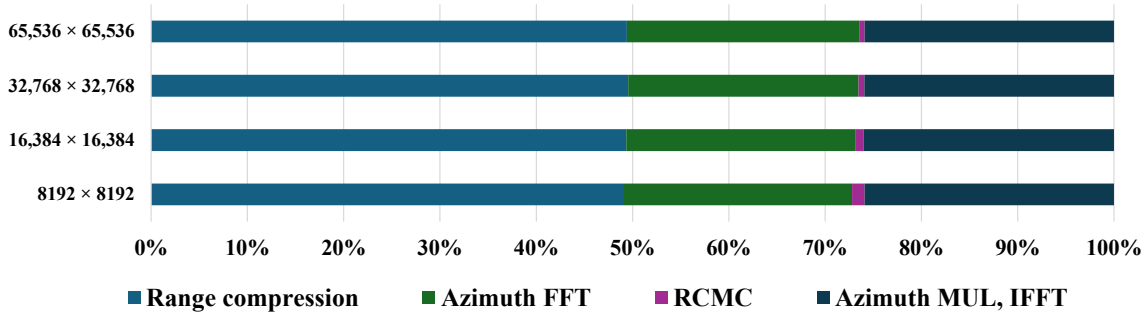

Figure 9. Execution time ratios of the RDA on the baseline system; MUL denotes reference signal multiplication.

4.2. SARDIMM Performance

We evaluated the RDA acceleration performance of the SARDIMM. As depicted in Figure 10, the normalized performance of both the baseline system and SARDIMM is presented. For the RDA execution, the SARDIMM conducts three primary operations: range compression, azimuth FFT, and azimuth reference signal multiplication followed by azimuth IFFT. The SARDIMM achieved speedups of 6.89×, 7.02×, 7.09×, and 7.17× for images of sizes 8192 × 8192, 16,384 × 16,384, 32,768 × 32,768, and 65,536 × 65,536, respectively, in range compression. Similarly, in azimuth FFT, speedups of 6.52×, 6.68×, 6.8×, and 6.97× were attained, while in azimuth IFFT after multiplication, speedups of 6.97×, 7.11×, 7.2×, and 7.23× were achieved. Between the range compression and azimuth FFT, the host transposes the compression result to the azimuth direction. Additionally, after the azimuth FFT, the host executes the RCMC operation based on the FFT results. Consequently, in the end-to-end RDA encompassing all host actions, speedups of 6.33×, 6.62×, 6.8×, and 6.94× were achieved.

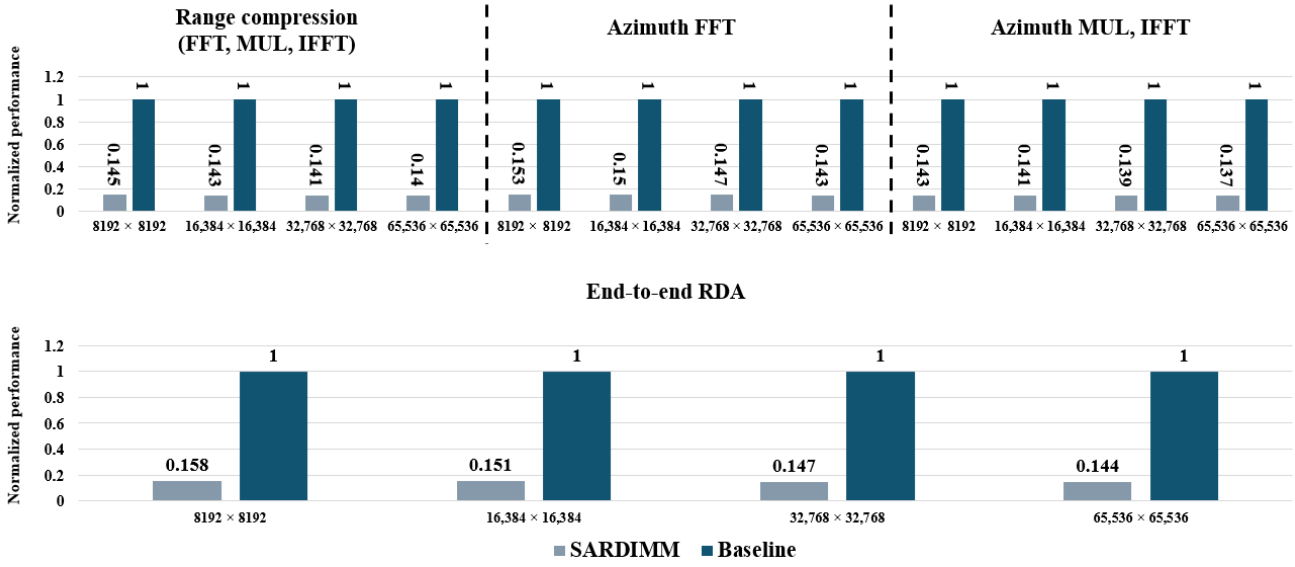

Figure 10. Normalized performance of the range compression, azimuth FFT, azimuth reference signal multiplication followed by azimuth IFFT, and end-to-end RDA on the baseline system and SARDIMM; MUL denotes reference signal multiplication.

Figure 11 shows the normalized total energy consumption by the DRAM in both the baseline system and SARDIMM. The SARDIMM, with reduced memory accesses, demonstrates energy savings in terms of overall DRAM energy consumption. Specifically, energy

savings of 43.1, 47.88, 48.5, and 48.57% were achieved for range compression and 39.39, 45.09, 45.87, and 47% for azimuth FFT. Similarly, the azimuth IFFT with multiplication exhibited energy savings of 44.16, 48.55, 49.2, and 49.64%. Notably, the end-to-end RDA achieved energy savings of 41.9, 46.97, 47.74, and 48.21%. In this study, performance evaluation was conducted on a single-channel, dual-rank SARDIMM, and achieved an average speedup of 6.67× and overall DRAM energy savings of 46.21% for RDA. Considering that parallelism scales with the number of SARDIMMs, we anticipate further performance enhancements by utilizing multiple SARDIMMs.

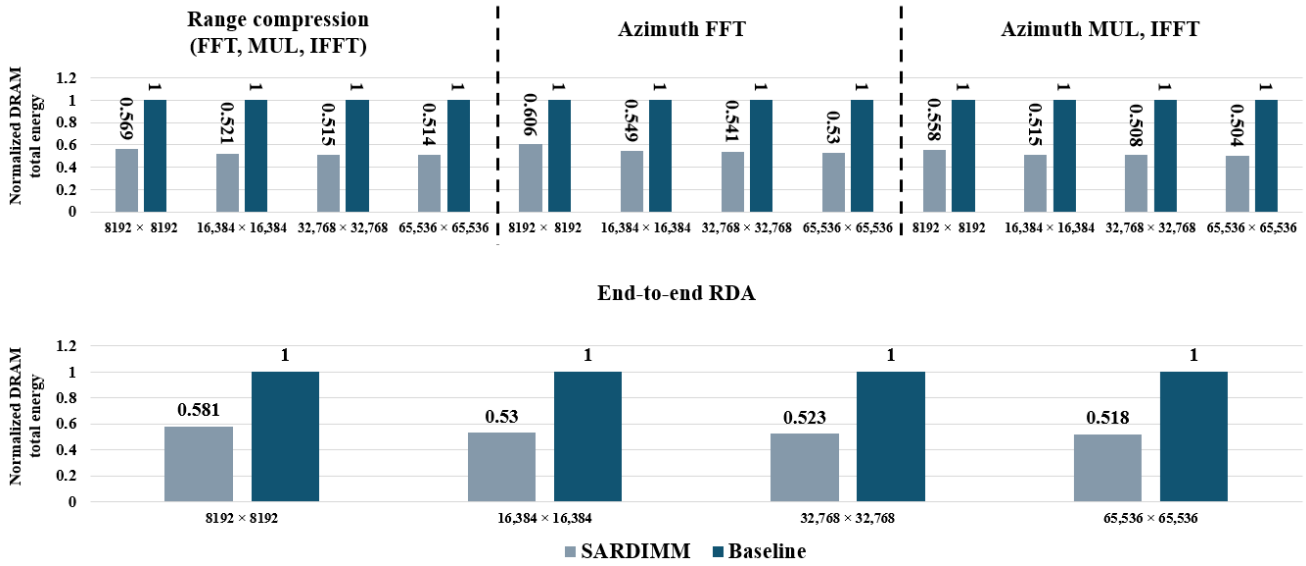

Figure 11. Normalized DRAM total energy of the range compression, azimuth FFT, azimuth reference signal multiplication followed by azimuth IFFT, and end-to-end RDA on the baseline system and SARDIMM; MUL denotes reference signal multiplication.

4.3. Design Overheads

The proposed SARDIMM integrates an NMP module into a buffer device without requiring any additional modification of the DRAM. The NMP module comprises buffers, an NMP controller, and an FPBF module, designed using Verilog HDL and implemented on the Xilinx UltraScale+ ZCU104 FPGA board. Table 2 presents a summary of the hardware resources utilized in the NMP module, which includes 10,493 look-up tables (LUTs), 2224 flip-flops (FFs), 40 digital signal processors (DSPs), and 144 block RAMs (BRAMs). In addition, the power consumption per NMP module was 0.678 W, since two NMP modules were inserted into the SARDIMM, with 1.356 W consumed per SARDIMM. Considering that the power of the DIMM was 13 W [28], the power of the NMP module was negligible.

Table 2. Implementation results of the proposed NMP module.

Module	CLB LUTs	CLB Registers	DSPs	BRAMs
Buffers	384	15	-	144
FPBF module	9474	2051	40	-
NMP controller	635	158	-	-
Total	10,493	2224	40	144

5. Conclusions

In this study, we introduced the SARDIMM, an NMP architecture based on a single-channel dual-rank DDR4 DIMM. The SARDIMM offers optional support for FFT, reference signal multiplication, and IFFT to accelerate the RDA for SAR image generation. By inte-

grating the NMP module into a buffer device positioned between the memory and memory controller, the SARDIMM operates seamlessly without requiring additional modifications to the DRAM chip. This design allows the NMP behavior to be controlled via standard memory write requests, ensuring compatibility with existing commercial memory systems. The SARDIMM supports the FP16 data type, and its NMP module is equipped with a floating-point adder, multiplier, and divider. We implemented and validated the NMP module of SARDIMM on an FPGA board, utilizing 10,493 LUTs, 2224 FFs, 40 DSPs, and 144 BRAMs. The performance evaluation conducted on images of varying sizes using customized gem5 and DRAMSim3 demonstrated an average speedup of 6.67× in end-to-end RDA, along with overall DRAM energy savings of 46.21% compared to the baseline system. In addition to the RDA, which includes FFT, reference signal multiplication, and IFFT operations, we plan to conduct future work exploring other algorithms for SAR image generation. For example, we intend to investigate the effectiveness of SARDIMM in accelerating the CSA and omega-k algorithm. These experiments will further demonstrate the versatility and performance benefits of SARDIMM in a broader range of SAR image processing applications.

Author Contributions: H.K. designed the SARDIMM and implemented the NMP module, performed the simulation and evaluation, and wrote the paper. J.H. and S.L. evaluated the proposed SARDIMM and revised this manuscript. Y.J. conceived of and led the research, analyzed the experimental results, and wrote the paper. All authors read and agreed to the published version of the manuscript.

Funding: This work was supported by the Technology Innovation Program (No. 00144288, 00144290), funded by the Ministry of Trade, Industry, and Energy (MOTIE, Korea), and CAD tools were supported by the IDEC.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: The original contributions presented in the study are included in the article, further inquiries can be directed to the corresponding author.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Chen, J.; Xing, M.; Yu, H.; Liang, B.; Peng, J.; Sun, G.C. Motion Compensation/Autofocus in Airborne Synthetic Aperture Radar: A Review. *IEEE Geosci. Remote Sens. Mag.* **2022**, *10*, 185–206. [CrossRef]
2. Kellogg, K.; Hoffman, P.; Standley, S.; Shaffer, S.; Rosen, P.; Edelstein, W.; Dunn, C.; Baker, C.; Barela, P.; Shen, Y.; et al. NASA-ISRO Synthetic Aperture Radar (NISAR) Mission. In Proceedings of the 2020 IEEE Aerospace Conference, Big Sky, MT, USA, 7–14 March 2020; pp. 1–21. [CrossRef]
3. Wu, Q.; Zhang, Y.D.; Amin, M.G.; Himed, B. High-Resolution Passive SAR Imaging Exploiting Structured Bayesian Compressive Sensing. *IEEE J. Sel. Top. Signal Process.* **2015**, *9*, 1484–1497. [CrossRef]
4. Bucciarelli, M.; Pastina, D.; Cristallini, D.; Sedehi, M.; Lombardo, P. Integration of Frequency Domain Wideband Antenna Nulling and Wavenumber Domain Image Formation for Multi-Channel SAR. *Int. J. Antennas Propag.* **2016**, *2016*, 2834904. [CrossRef]
5. Quegan, S. Spotlight synthetic aperture radar: Signal processing algorithms: Carrara W. G., Goodman R. S. and Majewski R. M., 1995, 554 pp. Artech House, Boston, London, £63, hb, ISBN 0-89006-728-7. *J. Atmos. Sol.-Terr. Phys.* **1997**, *59*, 597–598. [CrossRef]
6. Moreira, A.; Prats-Iraola, P.; Younis, M.; Krieger, G.; Hajnsek, I.; Papathanassiou, K.P. A tutorial on synthetic aperture radar. *IEEE Geosci. Remote Sens. Mag.* **2013**, *1*, 6–43. [CrossRef]
7. Curlander, J.C.; McDonough, R.N. *Synthetic Aperture Radar*; Wiley: New York, NY, USA, 1991; Volume 11.
8. Chan, Y.K.; Koo, V. An introduction to synthetic aperture radar (SAR). *Prog. Electromagn. Res. B* **2008**, *2*, 27–60. [CrossRef]
9. Malanowski, M.; Krawczyk, G.; Samczyński, P.; Kulpa, K.; Borowiec, K.; Gromek, D. Real-time high-resolution SAR processor using CUDA technology. In Proceedings of the 2013 14th International Radar Symposium (IRS), Dresden, Germany, 19–21 June 2013; Volume 2, pp. 673–678.
10. Woo, J.-C.; Lim, B.G.; Kim, Y.S. Modification of the Recursive Sidelobe Minimization Technique for the Range-Doppler Algorithm of SAR Imaging. *J. Electromagn. Waves Appl.* **2011**, *25*, 1783–1794. [CrossRef]
11. Araujo, G.F.; d'Amore, R.; Fernandes, D. Cost-sensitive FPGA implementation of SAR range-doppler algorithm. *IEEE Aerosp. Electron. Syst. Mag.* **2018**, *33*, 54–68. [CrossRef]

12. Hossain, M.A.; Elshafiey, I.; Alkanhal, M.A.; Mabrouk, A. Real-time implementation of UWB-OFDM synthetic aperture radar imaging. In Proceedings of the 2011 IEEE International Conference on Signal and Image Processing Applications (ICSIPA), Kuala Lumpur, Malaysia, 16–18 November 2011; pp. 450–455. [CrossRef]
13. Clemente, C.; Soraghan, J.J. Range Doppler SAR processing using the Fractional Fourier Transform. In Proceedings of the 11-th International Radar Symposium, Vilnius, Lithuania, 16–18 June 2010; pp. 1–4.
14. Hou, N.; Zhang, D.; Du, G.; Song, Y. An FPGA-based multi-core system for synthetic aperture radar data processing. In Proceedings of the 2014 International Conference on Anti-Counterfeiting, Security and Identification (ASID), Macau, China, 12–14 December 2014; pp. 1–4. [CrossRef]
15. Hawkins, D.; Gough, P. An accelerated chirp scaling algorithm for synthetic aperture imaging. In Proceedings of the IGARSS'97, 1997 IEEE International Geoscience and Remote Sensing Symposium Proceedings. Remote Sensing—A Scientific Vision for Sustainable Development, Singapore, 3–8 August 1997; Volume 1, pp. 471–473. [CrossRef]
16. Zhang, X.; Tang, J.; Zhong, H. Multireceiver Correction for the Chirp Scaling Algorithm in Synthetic Aperture Sonar. *IEEE J. Ocean. Eng.* **2014**, *39*, 472–481. [CrossRef]
17. Lee, J.; Jeong, D.; Lee, S.; Lee, M.; Lee, W.; Jung, Y. FPGA Implementation of the Chirp-Scaling Algorithm for Real-Time Synthetic Aperture Radar Imaging. *Sensors* **2023**, *23*, 959. [CrossRef]
18. Li, W.; Xu, Z.; Zhu, D. The FPGA Implementation of Real-Time Spotlight SAR Imaging. In Proceedings of the IGARSS 2018—2018 IEEE International Geoscience and Remote Sensing Symposium, Valencia, Spain, 22–27 July 2018; pp. 6703–6706. [CrossRef]
19. Zhu, D.; Zhang, J.; Mao, X.; Zhang, Y.; Wang, X.; Li, Y.; Ding, Y.; Guo, J.; Shi, J. A Miniaturized High Resolution SAR Processor Using FPGA. In Proceedings of the EUSAR 2016: 11th European Conference on Synthetic Aperture Radar, Hamburg, Germany, 6–9 June 2016; pp. 1–4.
20. Linchen, Z.; Jindong, Z.; Daiyin, Z. FPGA Implementation of Polar Format Algorithm for Airborne Spotlight SAR Processing. In Proceedings of the 2013 IEEE 11th International Conference on Dependable, Autonomic and Secure Computing, Chengdu, China, 21–22 December 2013; pp. 143–147. [CrossRef]
21. Duarte, R.P.; Cruz, H.; Neto, H. Reconfigurable accelerator for on-board SAR imaging using the backprojection algorithm. In *International Symposium on Applied Reconfigurable Computing*; Springer: Cham, Swizterland, 2020; pp. 392–401. [CrossRef]
22. Crasto, N.; Kumar, T.K.; Anuradha, D.; Barua, P.; Nemani, S. FPGA implementation of back projection algorithm for radar imaging. In Proceedings of the 2013 International Conference on Radar, Adelaide, Australia, 9–12 September 2013; pp. 97–100. [CrossRef]
23. Hettiarachchi, D.L.N.; Balster, E.J. Fixed-Point Processing of the SAR Back-Projection Algorithm on FPGA. *IEEE J. Sel. Top. Appl. Earth Obs. Remote Sens.* **2021**, *14*, 10889–10902. [CrossRef]
24. Williams, S.; Waterman, A.; Patterson, D. Roofline: An insightful visual performance model for multicore architectures. *Commun. ACM* **2009**, *52*, 65–76. [CrossRef]
25. Leitersdorf, O.; Boneh, Y.; Gazit, G.; Ronen, R.; Kvatinsky, S. FourierPIM: High-throughput in-memory Fast Fourier Transform and polynomial multiplication. *Mem.-Mater. Devices Circuits Syst.* **2023**, *4*, 100034. [CrossRef]
26. Corporation, I. Export Compliance Metrics for Intel Microprocessors. Available online: https://intel.com/content/www/us/en/support/articles/000005755/processors.html (accessed on 8 May 2024).
27. Ke, L.; Zhang, X.; So, J.; Lee, J.G.; Kang, S.H.; Lee, S.; Han, S.; Cho, Y.; Kim, J.H.; Kwon, Y.; et al. Near-Memory Processing in Action: Accelerating Personalized Recommendation with AxDIMM. *IEEE Micro* **2022**, *42*, 116–127. [CrossRef]
28. Kwon, Y.; Lee, Y.; Rhu, M. TensorDIMM: A Practical Near-Memory Processing Architecture for Embeddings and Tensor Operations in Deep Learning. In Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture, MICRO'52, Columbus, OH, USA, 12–16 October 2019; Association for Computing Machinery: New York, NY, USA, 2019; pp. 740–753. [CrossRef]
29. Yun, S.; Nam, H.; Park, J.; Kim, B.; Ahn, J.H.; Lee, E. GraNDe: Efficient Near-Data Processing Architecture for Graph Neural Networks. *IEEE Trans. Comput.* **2023**, 1–14.. [CrossRef]
30. Ke, L.; Gupta, U.; Cho, B.Y.; Brooks, D.; Chandra, V.; Diril, U.; Firoozshahian, A.; Hazelwood, K.; Jia, B.; Lee, H.H.S.; et al. RecNMP: Accelerating Personalized Recommendation with Near-Memory Processing. In Proceedings of the 2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA), Valencia, Spain, 30 May–3 June 2020; pp. 790–803. [CrossRef]
31. Tian, T.; Wang, X.; Zhao, L.; Wu, W.; Zhang, X.; Lu, F.; Wang, T.; Jin, X. G-NMP: Accelerating Graph Neural Networks with DIMM-based Near-Memory Processing. *J. Syst. Archit.* **2022**, *129*, 102602. [CrossRef]
32. Cooley, J.W.; Tukey, J.W. An algorithm for the machine calculation of complex Fourier series. *Math. Comput.* **1965**, *19*, 297–301. [CrossRef]
33. Pumma, S.; Vishnu, A. Semantic-Aware Lossless Data Compression for Deep Learning Recommendation Model (DLRM). In Proceedings of the 2021 IEEE/ACM Workshop on Machine Learning in High Performance Computing Environments (MLHPC), St. Louis, MO, USA, 15 November 2021; pp. 1–8. [CrossRef]
34. Alon, U.; Yahav, E. On the Bottleneck of Graph Neural Networks and its Practical Implications. *arXiv* **2021**, arXiv:2006.05205
35. Shin, H.; Kim, D.; Park, E.; Park, S.; Park, Y.; Yoo, S. McDRAM: Low Latency and Energy-Efficient Matrix Computations in DRAM. *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.* **2018**, *37*, 2613–2622. [CrossRef]

36. Deng, Q.; Jiang, L.; Zhang, Y.; Zhang, M.; Yang, J. DrAcc: A DRAM based accelerator for accurate CNN inference. In Proceedings of the 55th Annual Design Automation Conference, DAC'18, San Francisco, CA, USA, 24–29 June 2018; Association for Computing Machinery: New York, NY, USA, 2018. [CrossRef]
37. He, M.; Song, C.; Kim, I.; Jeong, C.; Kim, S.; Park, I.; Thottethodi, M.; Vijaykumar, T.N. Newton: A DRAM-maker's Accelerator-in-Memory (AiM) Architecture for Machine Learning. In Proceedings of the 2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO), Athens, Greece, 17–21 October 2020; pp. 372–385. [CrossRef]
38. Long, Y.; Na, T.; Mukhopadhyay, S. ReRAM-Based Processing-in-Memory Architecture for Recurrent Neural Network Acceleration. *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.* **2018**, *26*, 2781–2794. [CrossRef]
39. JEDEC. DDR4 SDRAM Standard. 2012. Available online: https://www.jedec.org/standards-documents/docs/jesd79-4 (accessed on 25 August 2024).
40. Lowe-Power, J.; Ahmad, A.M.; Akram, A.; Alian, M.; Amslinger, R.; Andreozzi, M.; Armejach, A.; Asmussen, N.; Beckmann, B.; Bharadwaj, S.; et al. The gem5 Simulator: Version 20.0+. *arXiv* **2020**, arXiv:2007.03152. [CrossRef]
41. Li, S.; Yang, Z.; Reddy, D.; Srivastava, A.; Jacob, B. DRAMsim3: A Cycle-Accurate, Thermal-Capable DRAM Simulator. *IEEE Comput. Archit. Lett.* **2020**, *19*, 106–109. [CrossRef]
42. Cumming, I.G.; Wong, F.H. Digital processing of synthetic aperture radar data. *Artech House* **2005**, *1*, 108–110.