Received: 24 March 2024 | Revised: 12 July 2024 | Accepted: 14 August 2024

DOI: 10.4218/etrij.2024-0141

SPECIAL ISSUE

ETRI Journal WILEY

XEM: Tensor accelerator for AB21 supercomputing artificial intelligence processor

Won Jeon | **Mi Young Lee** | **Joo Hyun Lee** | **Chun-Gi Lyuh**

Hyperscale AI SoC Research Section, Electronics and Telecommunications Research Institute, Daejeon, Republic of Korea

Correspondence
Won Jeon, Hyperscale AI SoC Research Section, Electronics and Telecommunications Research Institute, Daejeon, Republic of Korea.
Email: jeonwon@etri.re.kr

Funding information
National Research Foundation of Korea, Grant/Award Number: 2021M3H6A1017683

Abstract

As computing systems become increasingly larger, high-performance computing (HPC) is gaining importance. In particular, as hyperscale artificial intelligence (AI) applications, such as large language models emerge, HPC has become important even in the field of AI. Important operations in hyperscale AI and HPC are mainly linear algebraic operations based on tensors. An AB21 supercomputing AI processor has been proposed to accelerate such applications. This study proposes a XEM accelerator to accelerate linear algebraic operations in an AB21 processor effectively. The XEM accelerator has outer product-based parallel floating-point units that can efficiently process tensor operations. We provide hardware details of the XEM architecture and introduce new instructions for controlling the XEM accelerator. Additionally, hardware characteristic analyses based on chip fabrication and simulator-based functional verification are conducted. In the future, the performance and functionalities of the XEM accelerator will be verified using an AB21 processor.

KEYWORDS
artificial intelligence, high-performance computing, neural processing unit, system verification, tensor computation

1 | INTRODUCTION

Following the ever-increasing demand for high-performance computing (HPC) and data centers, the importance of supercomputing processors is increasing. Advances in HPC have led to advancements in simulation-based scientific fields, such as molecular dynamics, climate modeling, computational chemistry, and astrophysical simulation. Furthermore, the advent of generative pre-trained transformers (GPTs) has expanded HPC applications from conventional scientific areas to hyperscale artificial intelligence (AI) areas. As HPC applications have become more specialized and compute-intensive, dedicated accelerators designed for application characteristics have become more important than general-purpose central processing units (CPUs). In this context, domain-specific throughput processors, such as graphics processing units (GPUs) and neural processing units (NPUs) have been extensively used in data centers. Many global groups are developing their own NPU chips for use in data centers that train and service hyperscale AI applications. For instance, the NVIDIA Blackwell B100/B200 GPUs, Google Tensor Processing Unit (TPU), Graphcore Intelligence Processing Unit (IPU), Tesla D1 processor, and Meta Training and Inference Accelerator (MTIA) have been developed as domain-specific throughput processors to increase the computing power of data centers for hyperscale AI [1].

One of the key workloads in both HPC and hyperscale AI applications are the linear algebraic computations, such as matrix-matrix and matrix-vector multiplications [2, 3]. The artificial brain-21 (AB21) processor was proposed by the Electronics and Telecommunications Research Institute (ETRI) to accelerate HPC and AI applications [4]. In particular, the AB21 processor efficiently performs tensor computations in these applications. Computing systems based on AB21 processors consist of various hardware and software components, such as ARM processors, DDR5 memory interfaces, PCIe interfaces, on-chip interconnection networks, an OpenCL software stack, device drivers, a compiler, and intrinsic [4–6]. Among the complex system components of AB21 processors, *XEM* is the main computational module that performs matrix or vector operations on floating-point numbers, thereby providing the AB21 processor with primary Floating-point Operations Per Second (FLOPS) performance.

In this paper, we present a detailed description of XEM implementations, including hardware architectures, computational characteristics, cycle-level timing behavior analysis, and custom Instruction Set Architecture (ISA) for XEM. XEM employs a unique computing method, *outer product*, to process matrix-matrix multiplications [7]. Tensor accelerators inside most AI processors compute matrix multiplications using a systolic array or an element-wise inner product architecture [1]. With the outer-product-based architecture, XEM can effectively reduce the number of data wires for the input matrix operands. We validate the hardware designs and software functionalities for the XEM using a software-based architectural simulation and register-transfer level (RTL) simulation. In future work, the AB21 processor, including the XEM modules, will be verified in terms of functionalities and performance.

The contributions of this paper are summarized as follows:

- We propose a novel computing architecture including an outer-product-based tensor accelerator and provide detailed explanations of peripheral hardware structures for the accelerator and their management techniques.
- Detailed descriptions of functional and timing behaviors of the custom ISA for XEM architecture are provided. The behavior and performance analysis are conducted based on software-based architectural simulations.
- The proposed XEM hardware is implemented and synthesized in the TSMC 12-nm process and the characteristics of the synthesized XEM modules are provided.

The remainder of this paper is organized as follows. Section 2 briefly introduces the characteristics of the target applications, overall hardware architecture, and development purpose of the AB21 processor. Section 3 presents the detailed hardware architecture, newly introduced ISA lists, overall execution flow, and scalability of the XEM accelerator. The experimental results and design validation data of XEM are described in Section 4. In Section 5, we introduce related works of XEM and AB21 and conclude our paper in Section 6.

2 | BACKGROUND

2.1 | HPC and hyperscale AI applications

Conventionally, HPC applications have focused on simulation-based scientific computations such as molecular dynamics, climate modeling, and structural mechanics. Although this demand remains in conventional areas, the role of hyperscale machine learning, such as in training large language models or autonomous driving, has recently emerged. Early small transformer models could be trained using relatively light computing systems other than the HPC. However, as the scale of the model increases, as shown in Figure 1, training hyperscale machine learning models requires HPC and supercomputing systems (a few thousand GPUs to train a GPT-4 model) [1, 8, 9]. The core workloads in HPC applications are linear algebraic computations (e.g., matrix-matrix or matrix-vector multiplications) for tensor data. Thus, Linpack (or high-performance Linpack) is used to quantify the performance of HPC and supercomputers [2]. Similarly, training hyperscale machine learning models, such as GPT, requires massive linear algebraic computations owing to their huge multi-layer perceptron and attention operations. In summary, the efficient acceleration of tensor computations is the key to achieving high performance in both HPC and hyperscale AI.

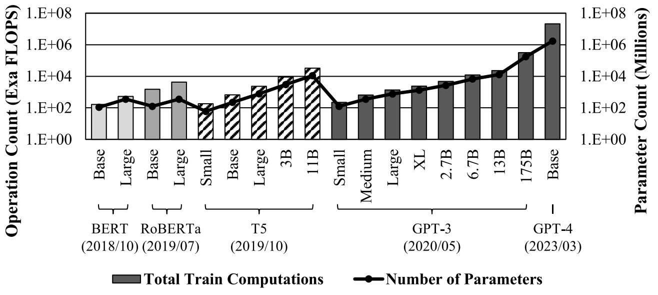


FIGURE 1 Total FLOPS for training each transformer model (bar graph) and a total number of parameters required to train each model (line graph).

2.2 | AB21 supercomputing AI processor

In this paper, we describe the detailed architecture of XEM, which is a tensor accelerator inside an AB21 processor [4]. AB21 is a custom-designed processor specialized in accelerating HPC and AI applications, designed by ETRI as a successor to the AB9 processor [10, 11].

Figure 2 briefly illustrates the overall architecture of an AB21 processor, and Table 1 lists the abbreviations used for the hardware components of an AB21 processor. A single AB21 chip contains six DDR5 and three PCIe5 interfaces for communication outside the processor. To provide high single-core performance and control the custom-designed accelerators, two ARM Neoverse V1 (PE) cores are integrated into AB21 [12]. Furthermore, AB21 achieved a particularly high parallel performance through eight XEMIS Cluster modules. *Nature* is a network on chip (NoC) set used to connect computing components and interfaces to each other. NoC is a mesh network built using ARM CMN-700 [13]. The PEs, XEMIS Clusters, Nature, and other modules constitute the *Earth* module. SCP assists the AB21 processor in initializing and booting the system.

As mentioned previously, XEMIS Clusters are key modules for the high-throughput performance of AB21. Figure 3 illustrates the detailed architecture of a single XEMIS Cluster. It contains a clock, reset, power (CRP) module, advanced peripheral bus (APB) module, memory management unit (MMU), and eight XEMIS modules. Each XEMIS includes four computing modules (XECM), four AXIC, an AXICS, an AXDCS, and an AXSP. The XECM and AXIC are dedicated to each other, whereas the AXICS, AXDCS, and AXSP modules are shared in one XEMIS module. With the support of the CRP, APB, and MMU modules, a XEMIS module can operate as an independent processing unit, similar to an NVIDIA Streaming Multiprocessor [14] or AMD Compute Unit [15].

The basic computing module of AB21, XECM, consists of a XEC core and a XEM accelerator. The XEC core is a RISC-V ISA-based custom processor that performs general-purpose parallel computations and controls the

TABLE 1 List of abbreviations for AB21.

Abbr.	Description
AB21	*Artificial Brain-21*, an HPC/AI processor developed by ETRI.
PE	*Processing Element*, an ARM Neoverse V1 core.
SCP	*System Control Processor*, a core that controls AB21 initialization and boot-up process.
XEMIS	*XElerator for Matrix multiplication with Input data Sharing*, a group of XECM modules.
XEC	*eXEcution Core*, a RISC-V core that control a XEM module.
XEM	*XElerator for Matrix multiplication*, a tensor accelerator proposed in this paper.
AXIC	*Ab21 Xemis Instruction Cache memory*, an L0 instruction cache memory.
AXICS	*AXIC Shared*, an L1 instruction cache memory shared in a XEMIS.
AXDCS	*Ab21 Xemis Data Cache Shared*, a data cache memory shared in a XEMIS.
AXSP	*Ab21 Xemis ScratchPad memory*, a user-programmable data memory.
AFPU	*Ab21 FPU*, an FPU top module which includes DPFU and SFPU.
DFPU	*Double-precision FPU*, an FPU which computes single FP64 or two FP32 numbers.
SFPU	*Single-precision FPU*, an FPU which computes two FP32 numbers.
XACC	*Xem ACCumulate register*, a register where computation results of XEM are accumulated.

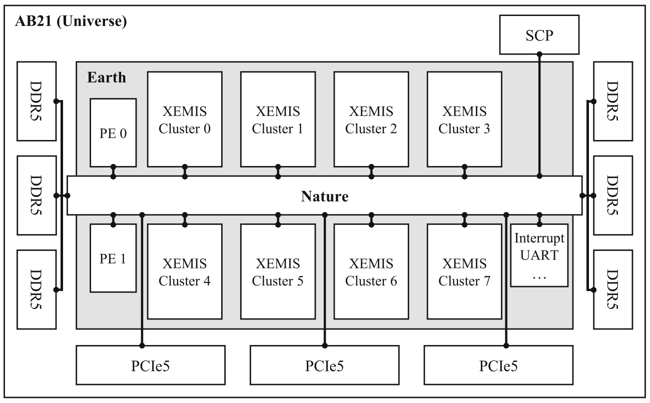


FIGURE 2 Illustrative overview of main hardware modules in the AB21 processor.

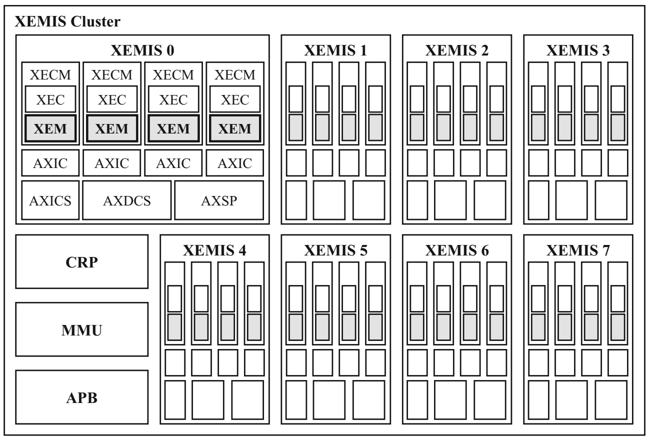


FIGURE 3 Hardware architecture of a XEMIS Cluster. In total 32 XEM modules are included in a single XEMIS Cluster.

XEM accelerator module. The XEC core supports part of the RISC-V 64G instruction set [16] and 2-way multi-thread execution to hide memory latency and efficiently operate the XEM module. In addition to the native RISC-V instruction set, we implement several custom instructions to the ISA. Detailed explanations of custom instructions are provided in Section 3.3. Each XECM unit can simultaneously execute different instructions because of the independent XEC cores and instruction cache memories. More detailed implementations of the AB21 hardware architectures and software stacks can be found in previous works [4–6]. In this paper, we focus on XEM architecture, which is the main computing unit that accelerates tensor computations in an AB21 processor.

3 | XEM ARCHITECTURE

As the core computation module inside the ETRI AB21 supercomputing AI processor, we propose a tensor accelerator architecture named XEM, which efficiently processes linear algebraic computations in HPC and hyperscale AI applications. XEM mainly performs outer-product-based matrix multiplications on two input vectors, controlled by the RISC-V core. The input vectors can be 64-bit double-precision floating-point numbers (FP64) or 32-bit single-precision floating-point numbers (FP32). The Floating Point Units (FPUs) of XEM are located close to each dedicated accumulation register (XACC). In this section, we provide detailed hardware implementations, ISA functional descriptions, and execution flow of XEM.

3.1 | Microarchitecture of XEM accelerator

To perform matrix multiplication operations on the XEM module, XEC reads the vector operands from AXSP and fetches them with XEM instructions to the XEM module. XEM produces incomplete matrix multiplication results by performing an outer product on two vectors, a row from matrix A and a column from matrix B. The outer product is repeatedly computed and accumulated to the XACC registers to complete a matrix multiplication operation. Figure 4 illustrates the matrix multiplication process and hardware microarchitecture of a XEM module and its sub-module. In this process, an incomplete result matrix is stored near FPUs, imposing a large size requirement for XACC. Furthermore, owing to the nature of the outer-product method, the required memory size for storing the incomplete result matrix increases in proportion to the square of the size of the input vector. For instance, while operands for FP32 operations can be fetched with the same memory interface as FP64 operations, the required XACC capacity for FP32 results is doubled, thus XACC 1 and 2 are not used in FP64 mode, as shown in Figure 4.

As shown in the figure, the XEM module has 16 AFPUs (or *AB21 FPU*) to compute the floating-point operations. The AFPUs in the XEM are implemented in a 4×4 configuration. The 64-bit fractions of the fetched vector operands are broadcasted to the corresponding rows or columns (e.g., A[63:0] to AFPU 0, 4, 8, and C, and B[255:192] to AFPU C, D, E, and F). The 64-bit input operand can be a single FP64 number or two vectored FP32 numbers. Once an AFPU module receives the

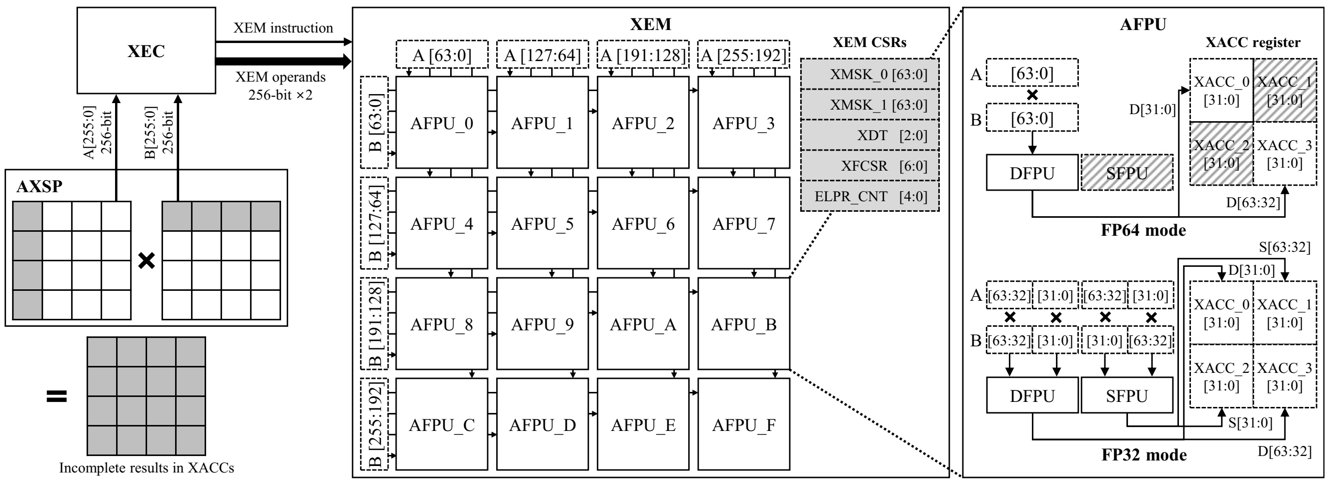


FIGURE 4 Overall architectures of XEM accelerator and the matrix multiplication method for XEM. AFPU is the arithmetic sub-module of XEM, performing one FP64 operation or four FP32 operations at a time. A single XEM module includes 16 AFPUs in a 4×4 configuration. Computation results are stored in XACC registers located inside each AFPU. A XEM MAC operation generates an incomplete matrix output with outer product operation.

64-bit operands A and B, it can compute one FP64 or four (2×2) FP32 operations. The types of supported operations include addition, subtraction, multiplication, and multiply-accumulate (MAC) operations between operands A and B. Further details on the supported operations are provided in Section 3.3. Consequently, when all AFPUs are active, a XEM module can compute 4×4 FP64 or 8×8 FP32 operations using the given input operands.

In the XEM module, several control and status registers (CSRs) are used to control the XEM operations and store the statuses of the XEM module. Most XEM CSRs can be accessed and modified by the XEC core with custom instructions. The list of XEM CSRs and their functions is as follows:

- *XMSK*: XEM masking register. Each bit of the register turns on or off the FPUs of XEM, thus when an FPU is masked, all arithmetic operations are ignored for the corresponding FPU.
- *XDT*: XEM data type register. XDT stores the data type (e.g., FP64 or FP32) information of the XACC register.
- *XFCSR*: XEM floating point control and status register. The lower 4 bits of XFCSR represent floating point flags and the upper 3 bits are used to store the rounding mode of XEM FPUs.
- *ELPR_CNT*: Cycle counter register for the end loop pipeline reduction (ELPR) operation.

For the XMSK register, only the lower 16 bits are used to represent 4×4 FPUs in the case of FP64 mode, whereas all 64 bits are used in the case of the FP32 mode. The XEM module has two XMSK registers to support the 2-way multithread of the XEC core. The detailed XMSK register indexing method is presented in Section 3.3. XDT is updated when the XACC register changes, for example, by performing AFPU operations or by directly updating the XACC register using the XEC core. The floating-point flags of XFCSR represent unexpected operations such as *invalid operation*, *overflow*, *underflow*, and *inexact operation*. When these operations occur during XEM computations, the corresponding bits are updated, and XEC can read the XFCSR with a custom instruction. *Division by zero* is not used because XEM does not support the division operation on input operands or XACC registers. For the rounding mode, XEM supports RISC-V standard rounding modes, such as *round to nearest, ties to even*, *round towards zero*, *round down*, *round up*, and *round to nearest, ties to maximum magnitude* [16]. XEC can access and update the rounding mode part of XFCSR with custom RISC-V instructions to change the rounding mode of XEM. ELPR_CNT register cannot be accessed by the XEC core and is used only inside the XEM module to control the ELPR process. A detailed description of the ELPR process is provided in the following section.

3.2 | Implementation details of AFPU

An AFPU computes the floating-point operations and stores the computation results inside the module, as shown on the right part of Figure 4. The AFPU module can take two 64-bit operands at a time and includes a double-precision FPU (DFPU), single-precision FPU (SFPU), and XACC register. When the AFPU computes FP64 data, the DFPU processes the computation, divides the 64-bit result into two 32-bit data, and stores them in XACC 0 and 3. In FP64 mode, SFPU, XACC 1, and 2 are inactive, thus XACC 1 and 2 hold the existing data. On the other hand, when the input data are in FP32 format, all DFPU, SFPU, and XACC are used. Two FP32 numbers are concatenated in each of the 64-bit A and B inputs. Owing to the vectored computation feature [17–19], both DFPU and SFPU can compute two floating-point operations in parallel and store the results in XACC. For SFPU, 32 bits of MSB and LSB are swapped to support the fully connected computations among the concatenated two A and B inputs, and the computation results of SFPU are stored in XACC 1 and 2. Combining DFPU and SFPU, one AFPU can perform four floating-point operations at once in FP32 mode.

Generally, arithmetic operations on high-precision floating-point numbers, such as FP64 and FP32, require several cycles to meet nano second-level frequency timing [17–19]. The target frequency of the AB21 processor and XEM accelerator is 1 GHz. To meet the timing requirements, both DFPU and SFPU are implemented with 4-staged pipeline FPUs. As a result, the latency of arithmetic operations on DFPU and SFPU is 4 cycles for all types of computations.

The pipeline latency of an FPU can cause data dependency and hazard problems. Unlike general processors, AFPU has a fixed output path from an FPU to XACC registers. Thus, it cannot solve the dependency of XACC registers by register renaming. Among XEM computations, addition, subtraction, and multiplication do not use XACC as operands. XEC simply blocks other XEM instructions when such operations are being executed. On the other hand, a MAC operation causes an XACC dependency issue because it accumulates a multiplication result to XACC ($XACC = A \times B + XACC$). Furthermore, because MAC is the key operation in matrix multiplications, processing a MAC operation for every cycle is important. To this end, we implement intermediate registers, named pipeline registers (PREG), between FPUs and XACCs to store temporary results as shown in Figure 5. Each XACC register has four PREGs because the FPUs have 4-staged pipelines, thus the number of PREGs can be changed depending on the number of pipeline stages. When XEM processes consecutive MAC operations, DFPU (or SFPU) reads the contents of PREG and performs a Fused Multiply-Add (FMA) operation on PREG, A, and B in every cycle. The accessed PREG index increases at every cycle from 0 to 3 in a round-robin manner. After each MAC operation is completed, the operation results are stored back to the point at which the PREG was read.

After a series of MAC operations are completed, the partial sums of the final results are stored in multiple PREGs. To obtain the complete MAC result, the distributed partial sums must be accumulated into one register. XEM performs the reduction process, named ELPR after a loop of MAC operations ends. ELPR is a custom signal between XEC and XEM, not a RISC-V instruction. XEC automatically issues an ELPR command to XEM after a loop of MAC operations is completed. During the ELPR process, DFPU (or SFPU) sequentially accumulates between the numbers stored in XACC and multiple PREGs. The loops of the MAC and ELPR processes can be separated because the addition of MAC operations does not require orders. The process requires multiple cycles, and XEC cannot fetch other XEM instructions during the process because all FPUs and registers in the XEM module are busy. After the ELPR process is completed, XACCs have the complete results of the MAC operations, thus XEC can access XACCs to get the computational results. Furthermore, all PREGs are reset to zero at the end of the ELPR process.

The unique implementations of the distributed PREGs and ELPR processes enable the execution of MAC operations in every cycle, thus improving the throughput of the FPUs. In addition, they can solve the swamping issue that occurs when many floating-point numbers accumulate [20]. As MAC operations continue, the accumulated results may become so large that adding small numbers will have no effect. The PREG implementation of AFPU alleviates this problem by storing results in multiple distributed registers.

3.3 | RISC-V instructions for XEM

As mentioned previously, the proposed XEM accelerator is controlled by a XEC core, which is a RISC-V ISA-based custom core. We extend the existing RISC-V ISA to include new custom instructions for controlling the XEM. Figure 6 presents the bit formats of the five new RISC-V instructions for XEM, *XMM*, *ALS*, *AAS*, *XFRCSR/XFSCSR*, and *MSK*. Programmers can accelerate various linear algebraic computations, access the computation results, and handle arithmetic details (e.g., floating-point rounding mode, flags, and masking) using the instructions. Although we implement system software support, intrinsic functions, and library support for AB21 and XEM [5, 6], we focus on the hardware-oriented behaviors of the new instructions in this study.

3.3.1 | XMM

XMM is the main computational instruction for XEM. When XEC executes the instruction, XEC decodes the memory addresses for operands A and B using RS1, RS2, and RS3, and then accesses AXSP using the address. After

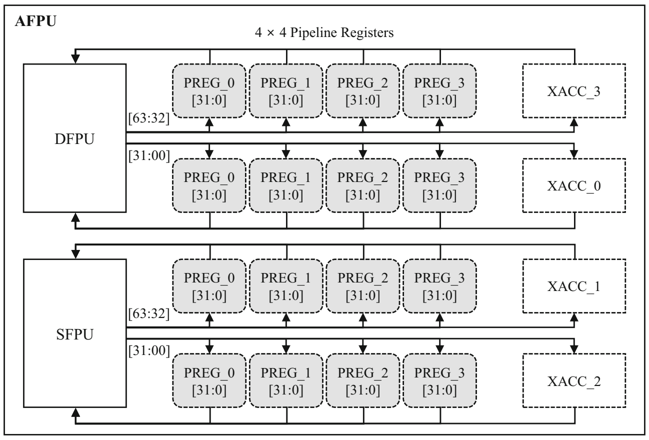
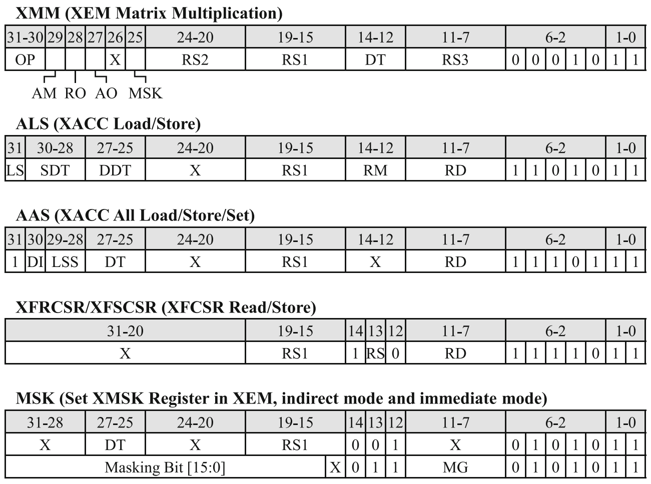


FIGURE 5 Register architectures of an AFPU module. Distributed pipeline registers can improve the throughput of FPUs and the accuracy of accumulated results.



XMM (XEM Matrix Multiplication)

31-30	29	28	27	26	25	24-20	19-15	14-12	11-7	6-2					1-0	
OP				X		RS2	RS1	DT	RS3	0	0	0	1	0	1	1

AM RO AO MSK

ALS (XACC Load/Store)

31	30-28	27-25	24-20	19-15	14-12	11-7	6-2					1-0	
LS	SDT	DDT	X	RS1	RM	RD	1	1	0	1	0	1	1

AAS (XACC All Load/Store/Set)

31	30	29-28	27-25	24-20	19-15	14-12	11-7	6-2					1-0	
1	DI	LSS	DT	X	RS1	X	RD	1	1	1	0	1	1	1

XFRCSR/XFSCSR (XFCSR Read/Store)

31-20	19-15	14	13	12	11-7	6-2					1-0	
X	RS1	1	RS	0	RD	1	1	1	1	0	1	1

MSK (Set XMSK Register in XEM, indirect mode and immediate mode)

31-28	27-25	24-20	19-15		14	13	12	11-7	6-2					1-0	
X	DT	X	RS1		0	0	1	X	0	1	0	1	0	1	1
Masking Bit [15:0]				X	0	1	1	MG	0	1	0	1	0	1	1

FIGURE 6 New extended instruction formats to the existing RISC-V ISA to control the XEM accelerator.

the data arrive, XEC fetches the instruction to XEM along with the operands. Detailed explanations of the bit fields are provided below:

- *OP*: Operation mode of XMM; 00 for addition, 01 for subtraction, 10 for multiplication, and 11 for MAC.
- *AM*: Address mode when accessing AXSP from XEC. When the field is 0, XEC uses the 64-bit absolute values stored in RS1 and RS2. Conversely, when AM is 1, XEC calculates the target memory addresses using RS1, RS2, and RS3.
- *RO*: Register operand for operand B. When the field is 1, operand B is not loaded from AXSP but from the register file of the XEC core.
- *AO*: XACC operand for operand A. When the field is 1, each AFPU uses its XACC as operand A.
- *MSK*: Enables masking mode. FPUs in the XEM are turned on/off according to the XMSK register when the field is 1.
- *RS1,2,3*: Register source indices. RS1 and RS2 store the memory addresses of operands A and B, and RS3 stores the memory address offset for operands A and B.
- *DT*: XMM data type; 000 for FP64, 001 for FP32, and others are reserved for futures uses.

When the AM field is 1 (relative addressing mode), RS3 stores memory offsets for operands A and B by dividing the 64-bit register into two 32-bit values, thus the target addresses for A and B are calculated as follows: $A = RS1 + RS3[31:0], B = RS2 + RS3[63:32]$. In the relative addressing mode, XEC can perform repeated XMM operations without redundant address calculations. In the RO mode case, the XEM module copies and concatenates the 64-bit or 32-bit operands to make a 256-bit vector operand B. Note that the effects of AM, RO, AO, and MSK are independent and can be turned on at the same time.

3.3.2 | ALS and AAS

After XMM operations are completed, the computational results are stored in XACC inside the XEM module. To process the results and read them from RISC-V programs, the contents of XACC need to be transferred to XEC register files or AXSP. However, XACC may need to be initialized with certain values before the main XEM computation begins. For this purpose, ALS and AAS instructions are used to read or write XACC in XEM modules. Furthermore, ALS accesses a single element at a time, whereas AAS accesses all XACCs over multiple cycles.

In the bit field of the ALS instruction shown in Figure 6, the LS field selects the data movement direction (load or store); for the 0 case, a data in XACC is moved to XEC's floating-point register, and for the 1 case, vice versa. In the transfer process, SDT and DDT determine the data types of the source and destination (XACC or XEC register), and the RM field selects the rounding mode for possible data type conversion. RS1 and RD store the target memory address or XACC index introduced in Figure 7.

The AAS instruction can write all XACC registers using data from the memory space (load), move all XACC contents to the memory space (store), or set all XACC registers with a single value (set). The operation mode is selected using an LSS field (00: load, 01: store, and 10: set). The DT field determines the data type, and RS1 and RD store the target memory address or source register number, respectively. The DI field turns the diagonal mode on or off for AAS load and AAS store. When the diagonal mode of AAS is turned on, the instruction does not access all XACCs, but only the diagonal part from the top left to the bottom right (only 0, 5, 10, and 15 XACCs in FP64 mode in Figure 7).

Using AAS instructions, redundant XEC instructions can be significantly reduced, as shown in Figure 8. The detailed evaluation environment for the figure is introduced in Section 4. Each execution of ALS instructions requires several instructions to calculate the register index of XACC or to control branches, whereas an AAS instruction does not. As a result, the use of AAS instructions can remove 77.79% of XEC's computing instructions, thus reducing 38.42% of the total instruction count from the same GEMM kernel.

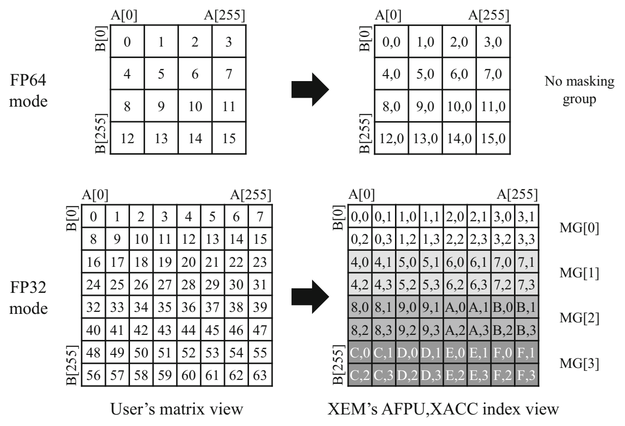


FIGURE 7 Indexing policy and masking group of XACC in 16 AFPUs in a single XEM module. We assume the raster scanning method for the user's matrix view. The indexing method is used in ALS, AAS, and MSK instructions.

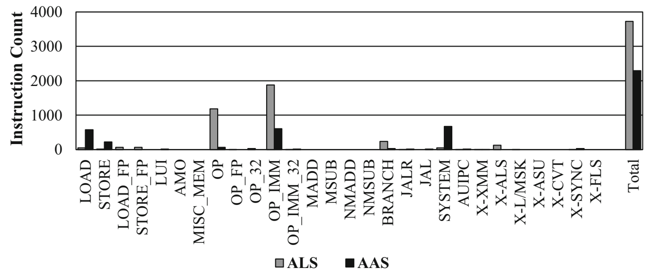


FIGURE 8 Instruction count comparison of FP32 GEMM kernel using ALS or AAS instruction, respectively (X: XEM instructions, others: original RISC-V ISA).

3.3.3 | XFCSR read and store

As mentioned in Section 3.1, XFCSR includes floating-point flags and the rounding mode of XEM FPUs. The XEC core can read the XFCSR to check whether unexpected floating-point operations have occurred or store it to set a policy in the rounding mode. The RS field determines the read or store operation on the XFCSR. When the XFCSR is read, data of the XFCSR are written to the XEC register pointed to by the RD field. Conversely, to store the XFCSR with specific values, XEC stores a 7-bit value in a XEC register in advance and designates that register as RS1 of the XFSCSR instruction.

3.3.4 | MSK

Depending on the application, users may want to turn computations on or off for specific elements of the matrix. XMSK registers in a XEM module and stores bit vectors that determine whether the corresponding FPUs are turned on or off. For a XEM module, 16 FP64 or 64 FP32 FPUs can be used. Thus, a XMSK register can store a 64-bit vector. Using MSK instructions, the XEC core can manipulate the XMSK register in the XEM module. The MSK instruction has two modes determined by the 13th bit: indirect and immediate, as shown in Figure 6. In indirect mode, a 64-bit masking bit vector is stored in a XEC core register designated by the RS1 field. Although this mode requires an additional step to store the bit vector in the register, all the 64 bits can be fetched using a single MSK instruction. In the immediate mode, the masking bit vector is stored in the MSK instruction during the compile time. Owing to the limitation of the instruction length, a single MSK immediate instruction holds only a 16-bit masking vector. To control all 64 FPUs with the MSK immediate instruction, we define masking groups as depicted in Figure 7.

```
#define XEM_XMM_A_OP(_opstring, _a_addr, _b_addr, _ao, _cn, _msk, _xdt)
{
  unsigned long long aaddr = _a_addr;
  unsigned long long baddr = _b_addr;
  __asm__ volatile (
"xpu.xem.xmm."TOSTR(_opstring)".a %[aaddr], %[baddr], %[ao], %[cn],
%[msk], %[xdt] "
  :
  : [aaddr]"r"(aaddr), [baddr]"r"(baddr), [ao]"I"(_ao), [cn]"I"(_cn),
[msk]"I"(_msk), [xdt]"I"(_xdt)
  : );
}
…
__kernel void xmm_a_f64_test(
    …
    XEM_XMM_A_OP(add, aa, ba, 0, 0, 0, XDT_F64);
    STORE_XACC_DIAGONAL_double(ca, XRM_RNE);
    aa += stride; ba += stride; ca += stride;

    XEM_XMM_A_OP(sub, aa, ba, 0, 0, 0, XDT_F64);
    STORE_XACC_DIAGONAL_double(ca, XRM_RNE);
    aa += stride; ba += stride; ca += stride;
    … (repeat)
}
```

FIGURE 9 An example OpenCL kernel code for using XEM accelerator including intrinsic functions for XMM instruction.

3.4 | Scalability of the XEM accelerator

As depicted in Figures 2 and 3, an AB21 processor includes eight XEMIS clusters, a XEMIS cluster has eight XEMIS modules. There are four XECM modules per XEMIS module, implementing 256 XECM modules in the AB21 processor. Therefore, a single AB21 processor can execute 256 RISC-V XEC cores in parallel. To control these parallel cores, RISC-V kernel codes can be programmed in the OpenCL format [6] as shown in Figure 9. Two ARM Neoverse V1 cores (PEs) deliver RISC-V kernels to XEC cores, and each XEC core executes the assigned kernel. Because one XEM module can perform 16 FP64 or 64 FP32 floating-point operations per cycle, one AB21 processor can perform 4,096 FP64 or 16,384 FP32 operations (addition, subtraction, multiplication, or MAC) per cycle.

4 | EVALUATION

4.1 | Methodology

We developed software-based simulation environments to evaluate the sketching performance and functionalities of the AB21 processor and XEM accelerator [6]. Specifically, architectural simulations of the AB21 processor and XEM accelerator were conducted using AXPUSIM. AXPUSIM exploits *Multi2Sim* [21] and *Spike* [22] as baseline simulators to model ARM processors (PE) and RISC-V cores (XEC), respectively. Furthermore, because the Spike simulator has no performance models, AXPUSIM adds timing models to the baseline model. Detailed architectural modules, interconnection networks, memory models, and ISA are implemented in the simulator. The simulation parameters for AXPUSIM related to the XEM accelerator are shown in Table 2. We evaluated various configurations of the AB21 processor using 70 test benchmark applications programmed to exploit the AB21 processor, including XEM accelerators. Figure 10 shows the distribution of instruction counts for the target benchmark applications. Although these applications are not extensively used, they have been developed to test the AB21 processor and XEM accelerator, and include essential linear algebraic computations, such as general matrix–matrix multiplication (GEMM), matrix–vector multiplication (GEMV), and AXPY.

TABLE 2 Simulator parameters for AB21 and XEM.

Parameters	Value
Core clock frequency	1000 MHz
PEs/AB21	2
XEMIS Clusters/AB21	8
XEMIS/XEMIS Cluster	8
XECM/XEMIS	4
AXICS/XEMIS	16 KB, 2-way
AXDCS/XEMIS	128 KB, 4-way
AXSP/XEMIS	256 KB
Threads/XEC	2
AXIC/XEC	2 KB, direct-mapped
FP64 FPUs/XEM	16
FP32 FPUs/XEM	64
XACC/XEM	256 B

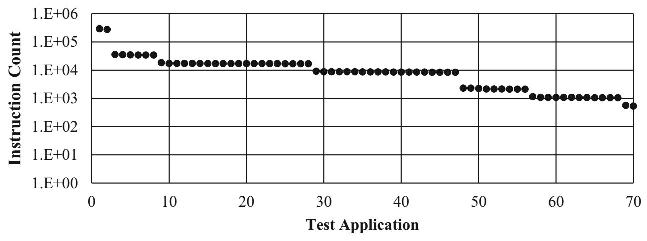


FIGURE 10 Distribution of the total number of instructions in the test applications in log scale (in descending order by the number of instructions).

4.2 | Performance and scalability

Figure 11 shows the normalized speedups of each hierarchical computing module (XECM, XEMIS, XEMIS Cluster, and AB21) for the 70 test applications. We define the normalized speedup as the relative ratio of the execution times between a target module and XECM. We can see that there is a significant difference in performance scalability depending on the application and a high correlation between the instruction count and scalability. Note that XEMIS, XEMIS Cluster, and AB21 have theoretical performances that are ×4, ×32, and ×256 greater than that of XECM, respectively. In small applications, it is difficult to observe the effects of scalability and parallel computing. In future work with the fabricated AB21 chip, the XEM accelerator will be tested with more compute-intensive benchmark applications.

As shown in the figure, we create groups of four according to the instruction count and scalability, and the average performance of each group is listed in Table 3. Most GEMM benchmarks are in group 1 which has the highest scalability. In particular, GEMM tests, which are the main targets of the XEM accelerator, show 70 to 93 relative speedups in AB21 compared with XECM. Group 2 includes GEMV tests that still exhibited

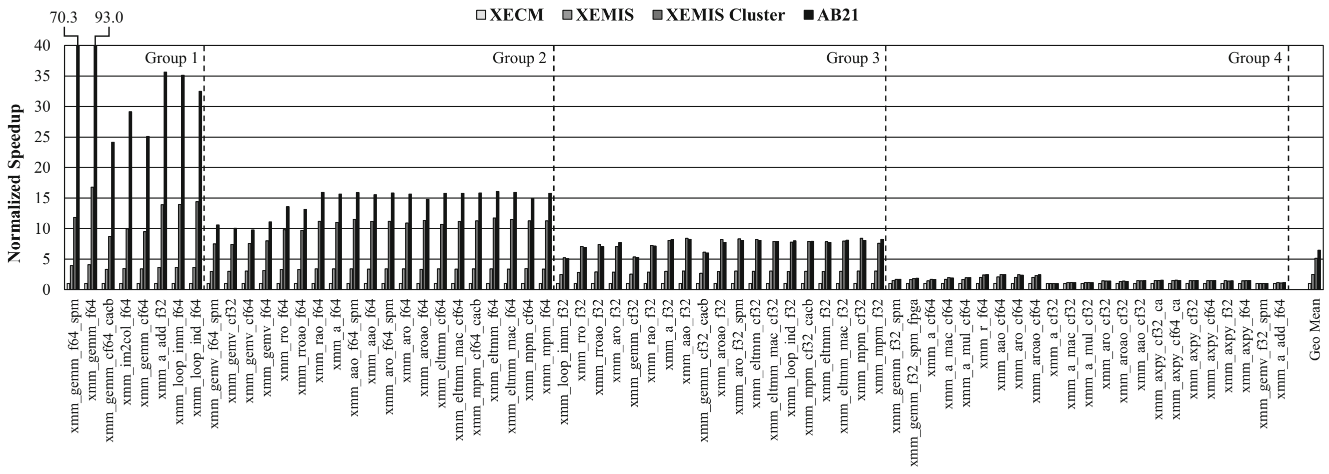


FIGURE 11 Normalized speedups of a XECM, XEMIS, XEMIS Cluster, and AB21 processor on 70 test applications. Execution times are normalized to XECM. Applications are sorted by the total number of instructions showed in Figure 10.

TABLE 3 Geometric means of the normalized speedup of each group and all applications.

Group #	XECM	XEMIS	Cluster	AB21
1	1	3.61	12.07	38.36
2	1	3.29	10.23	14.19
3	1	2.90	7.40	7.36
4	1	1.46	1.55	1.56
Total	1	2.46	5.14	6.44

high parallelism. Finally, many AXPY tests are included in group 4, which show very low speedups. According to the results, element-wise vector computations in AXPY are not effectively processed using the XEM accelerator. Despite the diagonal computation support introduced in Section 3.3, the arrays of FPUs are severely underutilized for AXPY computations. Reaching theoretical performance increase by ×256 by further optimizing the software and computing algorithm for XEM is critical for utilizing the AB21 processor.

4.3 | Hardware characteristic

The XEM accelerator and AB21 processor are designed, synthesized, and fully fabricated using the TSMC 12-nm process, as shown in Figure 12. The target die size of an AB21 chip is $26000 \times 28000 \mu m^2$, and its expected power consumption is listed in Table 4. The table lists the power consumption of each block, as shown in Figure 2. Note that the *Universe* block contains SCP but does not include DDR and PCIe modules. The power consumption of PEs, XEMIS Clusters, and Nature NoC are included in the *Earth* block. Because the Earth block has the most computational units, it consumes nearly 85% of the total power.

According to the operating frequency analysis, 820 MHz can be achieved in the worst case, 945 MHz in the typical case, and 1118 MHz in the best case. When synthesizing the XEM module separately, we observed that there was no negative slack for a clock period of 0.6 ns across the four staged FPU pipelines, allowing XEM to comfortably achieve an operating frequency of 1 GHz. With 4096 FP64 or 16 384 FP32 effective FPUs and the best-case frequency scenario, 4.58 TFLOPS ($\times 10^{12}$ FLOPS) on FP64 numbers or 18.32 TFLOPS FP32 numbers can be achieved theoretically. In particular, as the XEM accelerator supports one-cycle processing for the MAC (a combination of multiplication and addition) computation, consecutive MAC operations can be processed using the entire throughput rather than half (e.g., 4.58 or 18.32 $\times 10^{12}$ FMAs per second for FP64 and FP32, respectively).

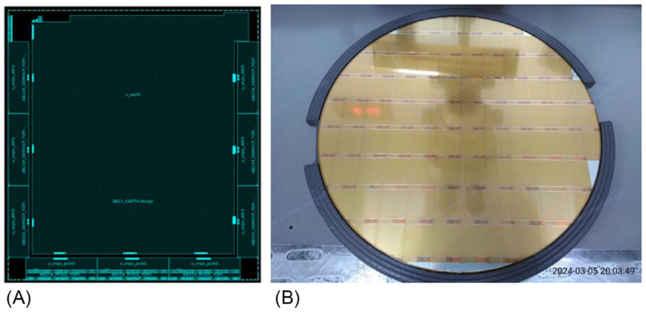


FIGURE 12 Floorplan for an AB21 processor (A) and die wafer photo of AB21 processor chips (B).

TABLE 4 Expected power consumption of an AB21 processor according to the back-end process.

Block name	Power consumption (W)	Relative power (%)
Universe	0.6	0.15
Earth	349.6	84.69
DDR5 ×6	39.9	9.67
PCIe5 ×3	22.7	5.50
Total	412.8	100.00

5 | RELATED WORK

To the best of our knowledge, XEM is the first tensor accelerator implemented in large-scale chip fabrication that computes matrix multiplications using the outer product method. XEM aims to provide high-performance matrix multiplication for an AB21 supercomputing AI processor. Similarly, the systolic tensor core (STC) in an AB9 processor targets the acceleration of matrix multiplication [10]. Furthermore, Chung et al. proposed a dynamic frequency scaling technique to improve the performance and save energy of the AB9 processor [11]. STC in the AB9 processor is a systolic-array-based tensor accelerator that is significantly different from XEM in the AB21 processor. Furthermore, the AB9 processor primarily targets the acceleration of the convolutional neural network-based vision applications, while the AB21 processor's target applications are HPC and hyperscale AI. Nevertheless, after the AB21 processor's chip fabrication and packaging are completed, performance comparisons between AB9 and AB21 can be conducted.

With the advent of LLM applications, many supercomputing AI processors have been introduced in industrial fields [1]. NVIDIA recently proposed the Blackwell GPU architecture, B100 and B200 GPUs [23]. NVIDIA GPUs exhibit specialization in training hyperscale AI models such as LLM by exploiting their high tensor processing performance enabled by Tensor Cores. Google developed the 5th generation of the TPU, which is extensively used in its data centers for training the LLM and deep learning recommendation model (DLRM) [24]. TPU features a systolic-array-based computing architecture similar to that of the STC in AB9. Graphcore introduced the MK2 IPU, a processor that utilizes high on-chip memory capacity and possesses advanced inter-node data connectivity technologies [25]. Meta also proposed the MTIA for use in data centers to support various AI applications, including DLRM [26]. Furthermore, many AI chips have been proposed as accelerators for data centers and supercomputing, including AMD Machine Intelligence 300 (or MI300) [15], Intel Gaudi-3 [27], Tesla D1 [28], FuriosaAI Renegade [29], Rebellions REBEL [30], SAPEON X330 [31], and Cerebras Wafer-Scale Engine-3 (or WSE-3) [32]. Because processors were proposed to accelerate AI applications, most support computations on low-precision floating-point numbers. Among the introduced accelerators, only GPUs of NVIDIA or AMD support high-performance computations on FP64 tensors, whereas the proposed XEM for the AB21 processor also provides such high-precision computations.

6 | CONCLUSION

As the demand for HPC and hyperscale AI training continues to increase, the demand for high-performance processors that can accelerate large linear-algebraic computations is also increasing. In this paper, we introduce in detail the hardware structures, operation methods, and newly added instructions for the XEM accelerator, which is the core computational unit in the AB21 processor developed to meet these demands. XEM performs matrix multiplication operations based on outer products and operates under the control of RISC-V cores. In this study, we demonstrate the feasibility of high-performance XEM accelerators and guide future users of AB21 processors to optimize their performance.

ACKNOWLEDGMENTS
This work was supported by the Supercomputer Development Leading Program of the National Research Foundation (NRF) funded by the Korea government (MSIT) (2021M3H6A1017683, Supercomputer Processor Research and Development).

CONFLICT OF INTEREST STATEMENT
The authors declare that there are no conflicts of interest.

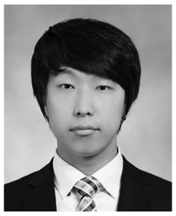
ORCID
Won Jeon https://orcid.org/0000-0002-5304-2007

REFERENCES

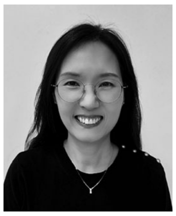
1. W. Jeon and C.-G. Lyuh, *Technical trends in hyperscale artificial intelligence processors*, Electron. Telecommun. Trends **38** (2023), no. 5, 1–11.
2. J. J. Dongarra, J. R. Bunch, C. B. Moler, and G. W. Stewart, *Linpack users' guide*, Vol. **8**, SIAM, 1979.
3. A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, *Attention is all you need*, Adv. Neural Inform. Process. Syst. **30** (2017), 1–11.
4. C. G. Lyuh, B. J. Kim, C. Kim, H. Kim, K. H. Park, J. H. Suk, K. Shin, M. Y. Lee, J. H. Lee, and W. Jeon, *Supercomputer SoC design and verification/FPGA platform development*, (Summer Annu. Conf. IEIE, Jeju, Republic of Korea), 2023, pp. 2732–2734.
5. C. Kim, J. H. Suk, S. Jun, and C.-G. Lyuh, *Porting linux on an FPGA board for ARM64 SoC test*, (Fall Annu. Conf. IEIE, Gwangju, Republic of Korea), 2022, pp. 125–127.
6. M. Y. Lee, J. H. Lee, and C.-G. Lyuh, *Intrinsic functions, libraries, and test application environment for accelerated parallel computing in matrix and vector operations*, (Fall Annu. Conf. IEIE, Seoul, Republic of Korea), 2023, pp. 363–365.
7. W. Jeon, Y. C. P. Cho, H. M. Kim, H. Kim, J. Chung, J. Kim, M. Lee, C.-G. Lyuh, J. Han, and Y. Kwon, M3FPU: Multiformat matrix multiplication FPU architectures for neural network computations, (IEEE 4th International Conference on Artificial Intelligence Circuits and Systems, Incheon, Rep. of Korea), 2022, pp. 150–153.
8. J. Achiam, S. Adler, S. Agarwal, L. Ahmad, I. Akkaya, F. L. Aleman, D. Almeida, J. Altenschmidt, S. Altman, and S. Anadkat, Gpt-4 technical report, 2023. https://doi.org/10.48550/arXiv.2303.08774
9. T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, and A. Askell, *Language models are few-shot learners*, Adv. Neural Inform. Process. Syst. **33** (2020), 1877–1901.
10. Y. C. P. Cho, J. Chung, J. Yang, C.-G. Lyuh, H. Kim, C. Kim, J. Ham, M. Choi, K. Shin, J. Han, and Y. Kwon, *AB9: A neural processor for inference acceleration*, ETRI J. **42** (2020), no. 4, 491–504.
11. J. Chung, H. Kim, K. Shin, C.-G. Lyuh, Y. C. P. Cho, J. Han, Y. Kwon, Y.-H. Gong, and S. W. Chung, *A layer-wise frequency scaling for a neural processing unit*, ETRI J. **44** (2022), no. 5, 849–858.
12. ARM, *ARM neoverses V1 core technical reference manual*, 2023.
13. ARM, *ARM neoverses CMN-700 coherent mesh network technical reference manual*, 2023.
14. NVIDIA, *NVIDIA H100 tensor core GPU architecture: exceptional performance, scalability, and security for the data center*, 2023.
15. AMD, *AMD CDNA3 architecture: the all-new AMD GPU architecture for the modern era of HPC and AI*, 2023.
16. A. Waterman and K. Asanovic, *The RISC-V instruction set manual volume I: user-level ISA v2.2*, 2017.
17. H. Kaul, M. Anders, S. Mathew, S. Hsu, A. Agarwal, F. Sheikh, R. Krishnamurthy, and S. Borkar, *A 1.45 GHz 52-to-162gflops/w variable-precision floating-point fused multiply-add unit with certainty tracking in 32nm CMOS*, (IEEE Int. Solid-State Circuits Conf., San Francisco, CA, USA), 2012, pp. 182–184.
18. S. Mach, F. Schuiki, F. Zaruba, and L. Benini, *Fpnew: An open-source multiformat floating-point unit architecture for energy-proportional transprecision computing*, IEEE Trans. Very Large Scale Integr. Syst. **29** (2020), no. 4, 774–787.
19. H. Zhang, D. Chen, and S.-B. Ko, *Efficient multiple-precision floating-point fused multiply-add with mixed-precision support*, IEEE Trans. Comput. **68** (2019), no. 7, 1035–1048.
20. N. Wang, J. Choi, D. Brand, C.-Y. Chen, and K. Gopalakrishnan, *Training deep neural networks with 8-bit*

floating point numbers, Adv. Neural Inform. Process. Syst. **31** (2018), 1–10.
21. R. Ubal, B. Jang, P. Mistry, D. Schaa, and D. Kaeli, *Multi2sim: A simulation framework for CPU-GPU computing*, (Proc. 21st Int. Conf. Parallel Architectures and Compilation Techniques, Minneapolis, MN, USA), 2012, pp. 335–344.
22. RISC-V International, *Spike RISC-V ISA simulator*, 2019.
23. NVIDIA, *NVIDIA blackwell architecture technical brief: Powering the new era of generative ai and accelerated computing*, 2024.
24. A. Vahdat and M. Lohmeyer, *Enabling next-generation AI workloads: announcing TPU v5p and AI hypercomputer*, 2023.
25. Graphcore, *Graphcore documents*. Available from: https://docs.graphcore.ai/en/latest/
26. A. Firoozshahian, J. Coburn, R. Levenstein, R. Nattoji, A. Kamath, O. Wu, G. Grewal, H. Aepala, B. Jakka, and B. Dreyer, *M TIA: First generation silicon targeting meta's recommendation systems*, (Proceedings of the 50th Annual International Symposium on Computer Architecture, Orlando FL USA), 2023, pp. 1–13.
27. Intel, *Intel Gaudi 3 AI accelerator*, 2024.
28. E. Talpes, D. D. Sarma, D. Williams, S. Arora, T. Kunjan, B. Floering, A. Jalote, C. Hsiong, C. Poorna, and V. Samant, *The microarchitecture of DOJO, Tesla's exa-scale computer*, IEEE Micro. **43** (2023), no. 3, 31–39.
29. FuriosaAI, *RNGD: The most efficient data center accelerator for high-performance LLM and multimodal deployment*. Available from: https://furiosa.ai/renegade-spec
30. Rebellions, *REBEL: shaping the future of gen AI*. Available from: https://rebellions.ai/products/
31. SAPEON, *Sapeon x330 product brief*, 2024.
32. Cerebras, *Wafer-scale engine 3: the largest chip ever built*, 2024.

AUTHOR BIOGRAPHIES



Won Jeon received his BS and PhD degrees in electrical and electronic engineering from Yonsei University, Seoul, Republic of Korea, in 2014 and 2021, respectively. He is currently a senior researcher at the Hyperscale AI SoC Research Section of the Electronics and Telecommunications Research Institute, Daejeon, Republic of Korea. His current research interests include neural processor architecture designs, floating-point unit designs for neural network computations, hyperscale artificial intelligence hardware systems, processing-in-memory architecture designs, and approximate computing for neural network applications.



Mi Young Lee received her BS degree in electrical and electronic engineering and MS degree in information and communication engineering from Ewha Womans University, Seoul, Republic of Korea, in 1999 and 2001, respectively. She joined Electronics and Telecommunications Research Institute, Daejeon, Republic of Korea, in 2001, and is currently a principal member of the research staff. Her area of interest is in the hardware and software architecture for accelerating transformer-based learning algorithms.

Joo Hyun Lee received his MS degree in electrical engineering from Pohang University of Science and Technology, Pohang, Republic of Korea, in 1998, and his PhD degree in Korea Advanced Institute of Science and Technology. During his academic years, he conducted research on SPARC micro-processor architecture. From 1998 to 2000, he worked with advanced DRAM design team at HYNIX. Since 2000, he has been with Electronics and Telecommunications Research Institute, Republic of Korea, where he researched mobile communication, broadcasting technology, and high-speed optical network. His current research interests include high-performance AI processor and AI compiler technology.

Chun-Gi Lyuh received his BS degree in computer engineering from Kyungpook National University, Daegu, Republic of Korea, in 1998. He received his MS and PhD degrees in electrical engineering and computer science from the Korea Advanced Institute of Science and Technology, Daejeon, Republic of Korea, in 2000 and 2004, respectively. He joined Electronics and Telecommunications Research Institute, Daejeon, Republic of Korea, in 2004, and is currently a principal member of the research staff. His current research interests include mobile deep learning processor, high-performance CPU design for HPC, and SoC architecture optimization for artificial neural network based on transformer.

How to cite this article: W. Jeon, M. Y. Lee, J. H. Lee, and C.-G. Lyuh, *XEM: Tensor accelerator for AB21 supercomputing artificial intelligence processor*, ETRI Journal **46** (2024), 839–850, DOI 10.4218/etrij.2024-0141