An LPDDR-based CXL-PNM Platform for TCO-efficient Inference of Transformer-based Large Language Models

Sang-Soo Park†, KyungSoo Kim†, Jinin So†, Jin Jung†, Jonggeon Lee†, Kyoungwan Woo†, Nayeon Kim†, Younghyun Lee†, Hyungyo Kim†, Yongsuk Kwon†‡, Jinhyun Kim†, Jieun Lee†, YeonGon Cho†, Yongmin Tai†, Jeonghyeon Cho†, Hoyoung Song†, Jung Ho Ahn‡, and Nam Sung Kim§
Samsung Electronics†, Seoul National University‡, and University of Illinois Urbana-Champaign§
{ss23.park, ks322.kim, jinin.so, jina.jung, jg1021.lee, kw93.woo, nayeonni.kim, youhyu36.lee}@samsung.com,
{hyungyo.kim, yssh.kwon, kjh5555, jieun308.lee, yeongon.cho, ym.tai, caleb1, shy.song}@samsung.com,
gajh@snu.ac.kr, nskim@illinois.edu

2024 IEEE International Symposium on High-Performance Computer Architecture (HPCA) | 979-8-3503-9313-2/24/$31.00 ©2024 IEEE | DOI: 10.1109/HPCA57654.2024.00078

***Abstract*—Transformer-based large language models (LLMs) such as Generative Pre-trained Transformer (GPT) have become popular due to their remarkable performance across diverse applications, including text generation and translation. For LLM training and inference, the GPU has been the predominant accelerator with its pervasive software development ecosystem and powerful computing capability. However, as the size of LLMs keeps increasing for higher performance and/or more complex applications, a single GPU cannot efficiently accelerate LLM training and inference due to its limited memory capacity, which demands frequent transfers of the model parameters needed by the GPU to compute the current layer(s) from the host CPU memory/storage. A GPU appliance may provide enough aggregated memory capacity with multiple GPUs, but it suffers from frequent transfers of intermediate values among GPU devices, each accelerating specific layers of a given LLM. As the frequent transfers of these model parameters and intermediate values are performed over relatively slow device-to-device interconnects such as PCIe or NVLink, they become the key bottleneck for efficient acceleration of LLMs.**

Focusing on accelerating LLM inference, which is essential for many commercial services, we develop CXL-PNM, a processing near memory (PNM) platform based on the emerging interconnect technology, Compute eXpress Link (CXL). Specifically, we first devise an LPDDR5X-based CXL memory architecture with 512GB of capacity and 1.1TB/s of bandwidth, which boasts 16× larger capacity and 10× higher bandwidth than GDDR6- and DDR5-based CXL memory architectures, respectively, under a module form-factor constraint. Second, we design a CXL-PNM controller architecture integrated with an LLM inference accelerator, exploiting the unique capabilities of such CXL memory to overcome the disadvantages of competing technologies such as HBM-PIM and AxDIMM. Lastly, we implement a CXL-PNM software stack that supports seamless and transparent use of CXL-PNM for Python-based LLM programs. Our evaluation shows that a CXL-PNM appliance with 8 CXL-PNM devices offers 23% lower latency, 31% higher throughput, and 2.8× higher energy efficiency at 30% lower hardware cost than a GPU appliance with 8 GPU devices for an LLM inference service.

I. Introduction

Transformer-based large language models (LLMs) have become the de facto standard for natural language processing (NLP)-driven applications, including text generation, classification, and translation [8], [15], [49]. Among them, the text generation component automatically generates human-readable text, playing an essential role in emerging applications, such as language translation, dialogue systems, and story generation. In particular, the generative pre-trained transformer (GPT) has delivered remarkable text-generation performance and enabled popular commercial services, exemplified by ChatGPT [34].

As larger models with longer input sequences improve the performance of LLMs, the model size and the sequence length have increased to hundreds of GB and thousands, respectively. This has proportionally increased the amount of computation and the size of the memory working set. For example, GPT-3.5, which is currently used by ChatGPT, with 175-billion parameters requires 1,425 TFLOPs of FP16 (16-bit floating-point) computation for inference with input and output sequence lengths of 2,048, while requiring 326 GB of the memory capacity. Putting another way, GPT-3.5 needs over 300,000× more computation and over 3,000× larger memory working set than ResNet-50 [10], [18].

Prior work has investigated the use of GPUs to accelerate LLM training and inference [1], [38], [44], [50]. However, such LLMs are simply too large to fit into the memory of a single GPU [1], [38]. For instance, an NVIDIA A100 GPU provides up to 80 GB of memory capacity, whereas GPT-3.5 demands 326 GB for storing the model parameters, as noted earlier. Besides, these LLMs require more memory bandwidth than such a GPU can provide with high bandwidth memory (HBM) [25] for efficient utilization of GPU compute resources and high throughput. To satisfy these requirements for large-scale LLM-based inference services, the prior work has turned to model parallelism in such LLMs, which allows us to distribute LLM layers and model parameters associated with the layers across multiple GPUs [1], [38], [44]. Nonetheless, it incurs a significant communication cost to aggregate the intermediate results from multiple GPUs at the end of computation for each layer through relatively low-bandwidth interconnects, such as NVLink or PCIe connecting the host CPU and GPU devices [1], [6].

We aim to tackle these challenges with CXL-PNM, a processing near memory (PNM) platform based on the emerging interconnect technology, Compute Express Link (CXL [41]). Specifically, CXL-PNM consists of three **C**omponents: **(C1)** LPDDR-based CXL memory architecture, **(C2)** CXL-PNM

2378-203X/24/$31.00 ©2024 IEEE
DOI 10.1109/HPCA57654.2024.00078

controller architecture integrated with an LLM inference accelerator, and **(C3)** a software stack.

(C1) LPDDR-based CXL memory architecture (§IV). To cost-effectively provide both large capacity and high bandwidth for a *single* CXL memory module, we propose to use *commodity* LPDDR5X [26] DRAM for CXL-PNM (§IV). Compared to DDR5 [23] and GDDR6 [24] DRAM, LPDDR5X provides unique properties and characteristics that are not widely known to the computer architecture community focusing on DDR and HBM DRAM for servers and accelerators. Specifically, the current LPDDR5X DRAM packaging technology allows us to provide up to 64 GB capacity and 136 GB/s bandwidth per package. It starts with 3D-stacking four 16 Gb DRAM dies but without using an expensive TSV technology, places eight such DRAM stacks on the package substrate, and then connects power and I/O pins from each DRAM die of these DRAM stacks to package pins with a cheap wire-bonding technology. Such large capacity and high bandwidth at reasonable costs and power consumption are owing to three advantages of LPDDR5X: (1) 1.56× higher Gbps/pin and 4× wider I/O width per DRAM die than DDR5 with 5.6Gbps/pin; (2) 14% lower pJ/bit than GDDR6 with 24 Gbps/pin; and (3) the use of a much cheaper wire-bonding-based 3D-stacking technology and 2.67× larger capacity per package than TSV-based 3D-stacking technologies for HBM. No commodity DDR5, GDDR6, and HBM packages have yet been designed to provide such large capacity and high bandwidth at the same time with a single DRAM package because of various constraints imposed by the JEDEC standard.

(C2) CXL-PNM controller architecture integrated with an LLM inference accelerator (§V). We first discuss four disadvantages of the recent PIM and PNM devices from the industry: HBM-PIM [29] and AxDIMM [27]. (1) HBM-PIM incurs high recurring development costs and provides limited capacity. (2) AxDIMM offers limited bandwidth and capacity scaling. (3) AxDIMM adopts inefficient arbitration of DIMM-PNM for serving concurrent memory requests from both the host CPU and the PNM accelerator. (4) Both HBM-PIM and DIMM-PNM suffer from conflicts with memory address interleaving techniques employed by the host CPU. Then, we present a CXL-PNM controller architecture, demonstrating that CXL-PNM can efficiently overcome these disadvantages, exploiting the unique capability of an LPDDR5X-based CXL memory module. Lastly, we propose a high-performance and energy-efficient accelerator architecture for LLM inference.

(C3) Software stack (§VI). We develop a software stack including a Python library and a device driver for CXL-PNM. The CXL-PNM Python library reimplements all the APIs in the standard Python library for CXL-PNM. This allows users to seamlessly and transparently use CXL-PNM for programs based on Python by simply replacing standard Python APIs in the programs with the corresponding CXL-PNM Python APIs. The CXL-PNM device driver provides APIs for the CXL-PNM Python library to configure, program, and control CXL-PNM.

For our evaluation, we take an LLM that fits into a single CXL-PNM device's memory but does not fit into the memory capacity of a single GPU device. Then, we run it on CXL-PNM and GPU appliances, each with 8 devices. With a large memory capacity for each device, the CXL-PNM appliance gives more flexibility than the GPU appliance for exploiting the model and data parallelism at the same time. The CXL-PNM appliance configured to maximize model parallelism delivers 23% lower latency, 31% higher throughput, and 2.8× higher energy efficiency at 30% lower hardware cost than the GPU appliance. The CXL-PNM appliance configured to maximize data parallelism gives 53% higher throughput and 4.4× higher energy efficiency than the GPU appliance. We also evaluate an LLM that fits into the memory capacity of a single GPU device and show that a CXL-PNM device provides 2.9× higher energy efficiency than a GPU device while offering latency and throughput comparable to the GPU.

II. Background

In this section, we first introduce CXL memory and discuss its benefits as a PNM platform. Subsequently, we describe the LLM model architecture and its computational characteristics.

A. CXL Memory as a PNM Platform

Built on the physical layer of the standard PCIe interface, the CXL standard defines three protocols: CXL.io, CXL.cache, and CXL.mem. Among them, CXL.mem allows the CPU to access CXL memory with load and store instructions in the same way as the CPU accesses memory in a remote NUMA node. As such, compared with PCIe-based acceleration platforms, CXL.mem-based acceleration platforms do not need to duplicate and/or explicitly transfer data between the host CPU's memory/storage and the accelerator's memory when both are provisioned with equally large enough memory capacity. This brings a remarkable advantage, especially when a given application demands large memory working sets (*e.g.*, 326 GB for storing GPT-3.5 model parameters) and memory has become relatively expensive due to stagnant memory technology scaling and growing memory capacity demand.

A typical architecture of CXL.mem controllers primarily consists of (1) a PCIe physical layer, (2) a CXL link layer, (3) a CXL transaction layer, and (4) memory controller blocks [47]. For a CXL.mem controller, the CXL transaction layer block not only supports CXL.mem but also CXL.io that uses the protocol features of the standard PCIe to initialize the interface between a CPU and a device [11]. The CXL link and transaction layer blocks along with other CXL-related components are collectively referred to as "CXL IP" hereafter. As of today, in addition to some research prototypes, multiple CXL memory modules have been designed by major memory manufacturers, such as Samsung [39], SK hynix [45], and Micron [37]. To facilitate diverse memory functionalities and PNM capabilities, Intel and AMD have also enabled the CXL protocols in the latest Agilex-I series FPGA [22] and Virtex UltraScale+ FPGA [51] integrated with hard CXL IPs.

B. GPT-3 Architecture and Inference

GPT-3 architecture. Fig. 1 depicts a GPT-3 model, which is based on the Transformer architecture [48], specifically

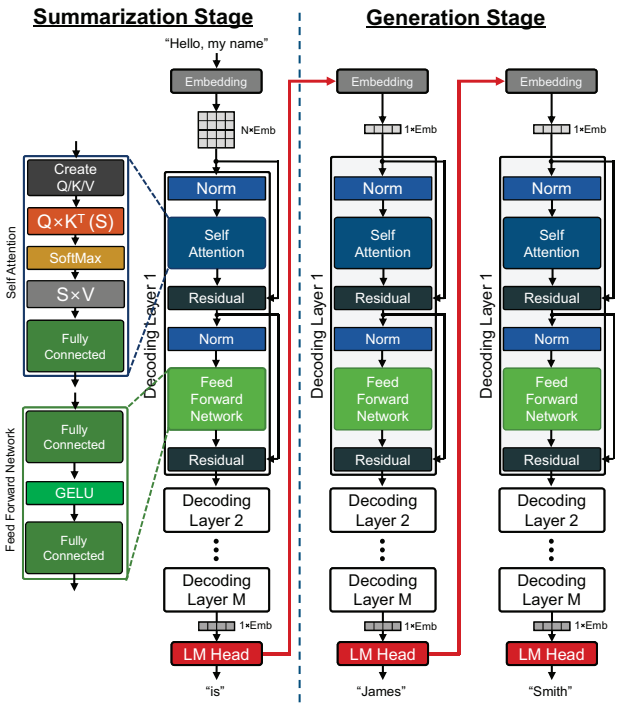


Fig. 1: GPT-3 architecture and inference process.

a decoder-only variant. It takes an input sequence (*i.e.*, a sequence of L_{in} tokens, each of which roughly corresponds to a word), and passes the tokens to its token embedding layer. Then the token embedding layer transforms the tokens into embedding vectors, and then passes the embedding vectors to M cascaded decoding layers, each with different pre-trained parameters, and the last M^{th} decoding layer generates an output embedding vector. Lastly, the language model (LM) head takes the output embedding vector from the last decoding layer and transforms it to an output token.

A decoding layer consists of two key layers from computation and memory-access perspectives: self-attention and feedforward network layers, both of which are accompanied by layer normalization (`LayerNorm`) and residual functions. The self-attention layer consists of three layers: (1) query, key, and value (QKV) generation, (2) attention, and (3) projection layers. (1) generates query (Q), key (K), and value (V), (2) determines the contextual representation between tokens, and (3) projects the outputs of the attention layer. The feedforward layer comprises two fully-connected (FC) layers and a Gaussian error linear unit (GELU) layer.

GPT-3 inference. A given input sequence goes through a summarization (`sum`) stage first, and then multiple generation (`gen`) stages, as shown in Fig. 1. The `sum` stage attempts to understand the context of the input sequence, based on which the `gen` stages generate new tokens. The `sum` stage, mostly comprising GEMM operations, gives each decoder a matrix of $L_{in} \times d_{emb}$ (*i.e.*, the dimension of embedding) as an input, processes the matrix through FC and attention layers, and generates an output matrix with the same size as the input

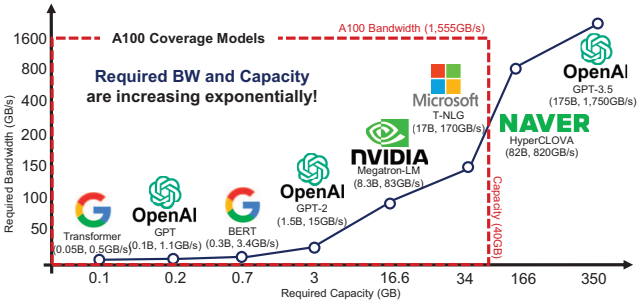


Fig. 2: Required memory capacity and bandwidth for the GPU to meet a latency constraint of 200 ms per output token.

matrix. The attention layer of the `sum` stage generates key and value (KV) matrices for the subsequent `gen` stages; The KV matrices contain the context of the input sequence, with a size of $2 \times L_{in} \times d_{emb}$.

The `gen` stages proceed similarly to the `sum` stage but with two key differences. First, a `gen` stage mainly consists of GEMV operation as it takes a single token generated in the previous stage as its input, and the input of each decoder is also a vector. Second, the attention layer of the `gen` stage operates on the Q vector of the current input token with an aggregated KV matrix. That is, each `gen` stage appends newly created KV vectors of an input token to the KV matrices passed from the previous stage to generate the aggregated KV matrix on the fly. When L is defined as the sum of L_{in} and the number of tokens generated up to the current stage, the dimensions of the Q vector and the KV matrix are $1 \times d_{emb}$ and $L \times d_{emb}$, respectively. The output token of a `gen` stage becomes the input of the next `gen` stage, being repeated until an end-of-sequence token is generated.

III. Challenges in Accelerating LLM Inference

The GPU has been the most widely used platform for training and inference of various AI/ML models, including LLM [1], [38], [44], [50], with its pervasive software development ecosystem and powerful computing capability. However, as the size of LLM models and their input sequence lengths further increase to improve the text-generation performance, the GPU can no longer efficiently accelerate LLM inference. In this section, we discuss two challenges in efficiently accelerating LLM (e.g., GPT-3) inference with the GPU.

A. Limited Memory Capacity and Bandwidth of a GPU Device

The size of LLMs has been increasing rapidly (see Fig. 2). For example, GPT-3.5 with 175-billion parameters, which ChatGPT is built on, requires 326 GB of memory while a single NVIDIA A100 GPU has only up to 80 GB of memory. A GPU device with a conventional software framework such as PyTorch cannot execute the code at all, or it must employ a sophisticated software framework such as DeepSpeed [1] or FlexGen [43] to intelligently transfer model parameters required by computing a given LLM layer from the host CPU's large memory/storage. Fig. 3 shows that almost 99% of the

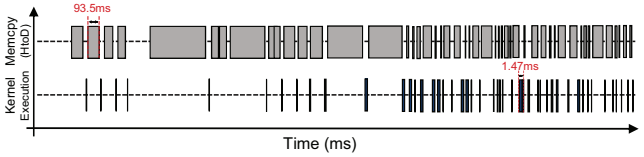


Fig. 3: Kernel execution time vs. memory copy time of an NVIDIA A100 GPU running OPT-30B inference.

execution time is spent for the host CPU's memory to the GPU's memory copy while executing Open Pre-trained Transformers (OPT [52]), open-source decoder-only pre-trained transformers resembling GPT, with 30-billion parameters.

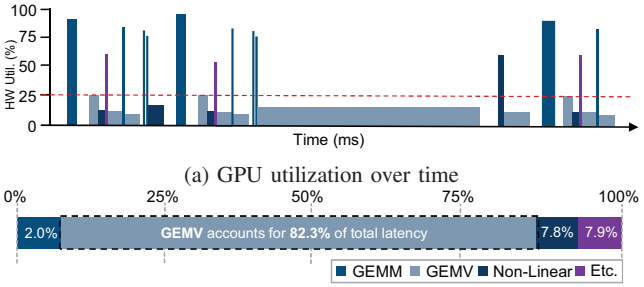
Even if we assume a hypothetical GPU device that may have large enough memory capacity to store all the model parameters, it still cannot efficiently accelerate GPT-3.5 inference due to the GPU device's limited memory bandwidth; we discuss this in §III-B in detail. For instance, if a given service is required to satisfy a latency requirement (*i.e.*, 200 ms per token generation), the GPU device needs to provide 1.75 TB/s of memory bandwidth for GPT-3.5 inference, as depicted in Fig. 2. Yet, the NVIDIA A100 GPU with 40 GB memory capacity can provide only 1.55 TB/s of memory bandwidth.

To overcome the memory capacity and bandwidth limitation of a single GPU device, a GPU appliance with multiple GPU devices can be used to accelerate GPT inference. The acceleration of GPT inference with multiple GPU devices exploits the model parallelism, distributing computation of layers and storage of model parameters associated with the layers across multiple GPU devices [1], [38], [44]. As one DGX system with eight A100 GPUs provides 320 GB[1] and 12.4 TB/s of aggregated memory capacity and bandwidth, respectively, two DGX systems may accelerate GPT-3.5 inference without transferring model parameters from the host CPU's memory/storage. Nonetheless, such GPU appliances are still unable to efficiently accelerate GPT inference because of frequent GPU-to-GPU transfers of intermediate values computed by each GPU device computing one or more layers of a given LLM [1]. These model parameters for single-GPU acceleration and intermediate values for multiple-GPU acceleration are transferred through relatively high-latency/low-bandwidth interconnects such as PCIe and NVLink compared to the aggregated memory bandwidth, which becomes the key bottleneck for efficient acceleration of LLM inference [6].

B. Inefficient Execution of Generation Stages with GPU

As described in §II-B, the `sum` stage exhibits different computational characteristics and thus efficiency from the `gen` stages. Specifically, the `sum` stage takes a batch of tokens as input; then it performs GEMM operations that can be efficiently executed by the GPU with its highly optimized BLAS library implemented, fully exploiting many GPU cores and small but high-bandwidth on-chip memory [35]. By contrast, a `gen` stage takes only a single token at a time, and thus performs GEMV operations that cannot be efficiently handled by the GPU when the size of a given matrix exceeds that of the on-chip memory [19], [21]. In such a case, the throughput of GEMV operations is constrained by the off-chip memory bandwidth [21]. Furthermore, LLM inference needs to sequentially go through multiple `gen` stages. To demonstrate the inefficient execution of the `gen` stages, we take an NVIDIA A100 GPU device, run OPT-6.7B inference on the GPU, and measure the utilization of GPU over time and the breakdown of execution time. Fig. 4 shows that (a) the GPU utilization for performing GEMV operations of the `gen` stages is under 25% while that for executing GEMM operations of the `sum` stage is up to 94%, and (b) 83% of the inference time is consumed by the GEMV operations.

[1]The DGX system available for our evaluation consists of A100 GPUs, each with 40 GB of memory capacity.



(a) GPU utilization over time

(b) Execution time breakdown

Fig. 4: The utilization and execution time breakdown of an NVIDIA A100 GPU running OPT-6.7B inference when L_{in} is 32 and the number of output tokens is 1024.

TABLE I: Comparison of DDR5, GDDR6, HBM3, and LPDDR5X-based CXL memory modules. The power consumption of DDR5, GDDR6, and HBM3-based CXL memory modules is normalized to that of a LPDDR5X-based memory module. The per-module values of HBM3 are based on an assumption that HBM is integrated in an NVIDIA H100 SXM [31] package.

	DDR5	GDDR6	HBM3	LPDDR5X
Bandwidth/pin	5.6 Gb/s	24 Gb/s	6.4 Gb/s	8.5Gb/s
I/O width/package	×4	×32	×1024	×128
Bandwidth/package	2.8 GB/s	96 GB/s	819 GB/s	136 GB/s
Capacity/package	16 GB	2 GB	16 GB	64 GB
Packages/module	32	16	5	8
I/O width/module	128	512	5120	1024
Bandwidth/module	89.6 GB/s	1.5 TB/s	4.1 TB/s	1.1 TB/s
Capacity/module	512 GB	32 GB	80 GB	512GB
Core voltage	1.1 V	1.35 V	1.1 V	1.05 V
IO voltage	1.1 V	1.35 V	0.4 V	0.5 V
Power/module	0.35	0.96	3.00	1.00

IV. LPDDR-based CXL Memory Module

In §III, we discussed the challenges in efficiently accelerating LLM inference with a single GPU and a GPU appliance. Because these challenges primarily stem from the fact that a

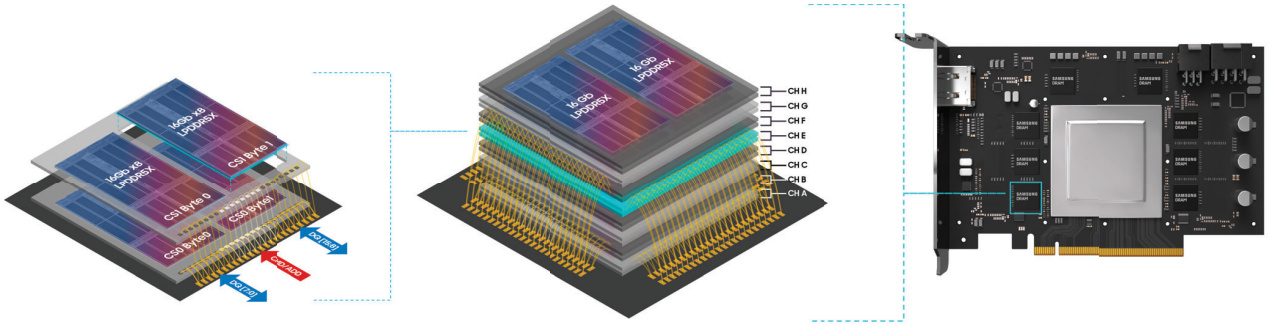


Fig. 5: LPDDR5X-based CXL memory modules: (left) an LPDDR5X single-channel configuration, (middle) a 512Gb LPDDR5X package with 8 channels, and (right) an LPDDR5X-based CXL memory module with 8 LPDDR5X ×128 DRAM packages in full-height/half-length (FHHL) form factor.

single GPU does not provide enough memory capacity, we propose a CXL memory module that can provide both large capacity and high bandwidth for an LLM inference accelerator. Specifically, we first examine DRAM and packaging technologies of DDR5, GDDR6, and HBM3 [25], focusing on their limitations in providing both large capacity and high bandwidth at the same time. Subsequently, we propose using LPDDR5X DRAM and its packaging technologies to build a CXL memory module after discussing their advantages.

Table I compares various aspects of LPDDR5X with those of DDR5, GDDR6, and HBM3, including the total deliverable memory capacity and bandwidth under the form-factor constraint of a CXL memory module, as well as (normalized) power consumption per CXL memory module. As each DRAM technology provides multiple packaging options for each DRAM technology, we pick the option that can maximize the total memory capacity for a CXL memory module. Below, we discuss the memory capacity and bandwidth of each DRAM technology for a CXL memory module in more detail.

DDR5. On a CXL memory module with the full-height/half-length (FHHL) form factor (referred to as a CXL memory module hereafter), we can place as many as 32 DDR5 DRAM packages, each with 4 DQ pins (*i.e.*, a DDR5 ×4 DRAM package). The DDR5 DRAM packaging technology can 3D-stack up to eight 16 Gb DRAM dies in a package using an expensive TSV technology. Such a DDR5 DRAM package is used to build high-capacity DRAM modules used by servers, and it can provide 512 GB (16 Gb/die × 8 dies/package × 32 packages) for the memory capacity of a CXL memory module. At the same time, with 128 DQ pins (4 pins/package × 32 packages) and 5.6 Gb/s per DQ pin, it can offer 89.6 GB/s (5.6 Gb/s/pin × 128 pins) at most for the memory bandwidth that can be exposed to the CXL controller package on a CXL memory module. We may consider using the DDR5 ×16 DRAM package, which will increase the memory bandwidth of the CXL memory module by 4× (*i.e.*, 358.4 GB/s), but the ×16 DRAM package is made for small-form-factor desktop or high-end laptop computers. As such computers do not require large memory capacity, the manufacturers do not make a DDR5 ×16 DRAM package 3D-stacking multiple 16 Gb DRAM dies for larger capacity. Thus, a DDR5 ×16 DRAM package with a single 8 Gb DRAM die effectively reduces the memory capacity of a CXL memory module by 16× (*i.e.*, 32 GB), compared with the DDR5 ×4 DRAM package.

GDDR6. On a CXL memory module, we can place up to 16 GDDR6 DRAM packages, each with 32-bit DQ pins (*i.e.*, GDDR6 ×32 DRAM package). Although GDDR6 ×32 DRAM package (14 mm × 12 mm) is 53% larger than a DDR5 ×4 DRAM package (10mm × 11mm)[2], its size does not limit the number of GDDR6 ×32 DRAM packages on a CXL module. Instead, the number of PCB traces between GDDR6 DRAM ×32 packages and the CXL controller package limits the number of GDDR6 DRAM ×32 packages on the CXL memory module to 16. As each GDDR6 DRAM ×32 package provides 2 GB and the bandwidth per DQ pin is 24 Gb/s, 16 GDDR6 ×32 DRAM packages can offer 32 GB (2 GB/package × 16 packages) and 1.5 TB/s (24 Gb/s/pin × 32 pins/package × 16 packages) for the memory capacity and bandwidth of a CXL memory module. While using GDDR6 ×32 DRAM packages gives higher memory bandwidth than using DDR5 ×4 and ×16 DRAM packages, the capacity is too small for accelerating LLM inference. Due to stringent constraints imposed by high-speed signaling, GDDR6 cannot support multiple ranks (*e.g.*, 3D-stacking GDDR6 dies to constitute multiple ranks) per channel for larger memory capacity like DDR5. Lastly, the high-speed signaling of GDDR6 consumes a significant amount of power, which is the highest one per package among the DRAM technology choices. Increasing the number of such GDDR6 packages will further reduce the power budget for the accelerator under the power constraint of a CXL memory module (150 Watts).

HBM3. Unlike DDR5 and GDDR6 DRAM, HBM DRAM is in an MPGA (Micro-Pillar Grid Array) package, and such a special package cannot be placed on a PCB directly. To provide large capacity and/or higher bandwidth for a CXL memory module, one or more HBM MPGA packages are in-

[2]A GDDR6 package may provide more pins with a finer bump-to-bump pitch (*e.g.*, GDDR6 ×32 DRAM: 0.75 mm vs. DDR5 ×4 DRAM: 0.8 mm).

tegrated with an xPU die on a silicon interposer, which is then placed in a conventional BGA package, as a system-in-package (SiP). An HBM3 package consists of eight 3D-stacked 16 Gb DRAM dies and uses an expensive TSV technology to provide high bandwidth for the xPU with 1024 6.4 Gbps DQ pins. In an NVIDIA H100 GPU, up to 5 HBM3 MPGA packages are integrated into an SiP, providing 80 GB (16 Gb/die $\times$ 8 dies/package $\times$ 5 packages) and 4.1 TB/s (6.4 Gb/s/pin $\times$ 1024 pins $\times$ 5 packages) for the memory capacity and bandwidth of a GPU, respectively. While it delivers the highest memory bandwidth, the memory capacity is still too small for accelerating GPT inference. To increase the memory capacity, we can 3D-stack more DRAM dies per package and/or integrate more HBM packages within an SiP. Yet, such options are limited to 12 DRAM dies per package and 6 HBM DRAM packages per SiP at the moment, because of power, thermal, and mechanical constraints imposed on HBM DRAM and SiP packages.

LPDDR5X. LPDDR DRAM was originally developed for mobile devices, but it has been used for a wider range of devices, including AI/ML accelerators because it gives a unique trade-off among memory capacity, bandwidth, and power, compared to DDR, GDDR, and HBM DRAM. Fig. 5(left) depicts a 16-bit LPDDR5X channel. It consists of two 3D-stacks, each 3D-stacking two LPDDR5X DRAM dies with a wire-bonding technology [46]. The latest LPDDR5X delivers 16 Gb DRAM dies and 8.5 Gb/s per DQ pin. That is, each LPDDR5X channel can provide 64 Gb (16 Gb/die $\times$ 4 dies) and 17 GB/s (8.5 Gb/s/pin $\times$ 16 pins). As shown in Fig. 5(middle), we can place up to 8 such 3D-stacks of LPDDR5X DRAM dies in a single package with 128 DQ pins (LPDDR5X $\times$128 DRAM package). This gives us up to 64 GB (64 Gb/channel $\times$ 8 channels) and 136 GB/s (17 GB/s/channel $\times$ 8 channels) for the capacity and bandwidth of an LPDDR5X $\times$128 DRAM package, respectively. Under the FHHL form factor constraint, we can place up to 8 LPDDR5X $\times$128 DRAM packages along with a CXL controller package on a CXL memory module, as illustrated in Fig. 5(right). Since an LPDDR5X $\times$128 DRAM package is slightly larger than a GDDR6 $\times$32 DRAM package, we may place more LPDDR5X $\times$128 DRAM packages on a CXL memory module. Nonetheless, the number of LPDDR5X $\times$128 DRAM packages is limited by the number of PCB traces between LPDDR5X $\times$128 DRAM and CXL controller packages. Overall, LPDDR5X $\times$128 DRAM packages can provide 512 GB (64 GB/package $\times$ 8 packages) and 1.1 TB/s (136 GB/package $\times$ 8 packages) for the memory capacity and bandwidth of a CXL memory module, respectively.

LPDDR has an established manufacturing ecosystem to provide far more commercial packaging options than DDR and GDDR to meet various form-factor constraints demanded by diverse mobile devices. This allows us to cost-efficiently change the number of LPDDR5X channels (from two to eight) per package and LPDDR5X DRAM dies per 3D-stack (from two to four). As such, we can increase the capacity of a package up to 128 GB with four LPDDR5X DRAM dies per 3D-stack. This can provide 1 TB for the memory capacity of a CXL memory module.

V. CXL-PNM Architecture

In §IV, we demonstrated that LPDDR5X DRAM can provide both large capacity and high bandwidth for the CXL controller on a CXL memory module. Building on such a CXL memory module, we first propose CXL-PNM controller architecture after discussing the unique advantages and capabilities of a CXL memory module as a PNM platform. Subsequently, we describe our LLM inference accelerator architecture designed to exploit the CXL memory module.

A. CXL Memory Module as a PNM Platform

To reduce the (bandwidth and energy) costs of transferring data between memory and processors/accelerators in computing systems, diverse PIM (processing-in-memory) and PNM architectures have been proposed. In this section, we take HBM-PIM and AxDIMM as the state-of-the-art industry PIM and PNM architectures, bring up four **D**isadvantages from development and usage perspectives, and discuss how the CXL memory module we present in §IV obviates the disadvantages.

(D1) High development cost of PIM. We recognize PIM as the most powerful architecture for such a purpose since PIM can expose higher bandwidth to accelerators with shorter interconnects (*i.e.*, lower energy per bit transfer) than PNM architectures. However, among many challenges, PIM requires the memory manufacturers to develop (application- or domain-specific) accelerators under a tight die area constraint, make necessary changes in memory, and integrate them together. Further, mass manufacturing of a memory product demands them to go through various procedures, such as verification, validation, and internal/external qualification, which also significantly consumes as much development resources as the memory product design itself. Meanwhile, the development resources of the memory manufacturers have been constrained by developing continuously evolving commodity products, which has become more difficult with aggressive memory technology scaling. As such, memory manufacturers prefer solutions that allow them to reuse commodity memory designs.

(D2) Limited memory capacity and bandwidth scaling with DIMM-based PNM. As an alternative to PIM, PNM is an attractive architecture because commodity memory packages can be reused. A noteworthy example is AxDIMM, a DIMM-based PNM (DIMM-PNM) architecture that places DDR DRAM and FPGA packages on a DIMM. However, a DIMM has limited space for the interconnects between memory and accelerator packages, imposed by the tight DDR DIMM form factor, which is in turn dictated by the popular 1U server form factor. This limits not only the memory bandwidth, which can be exposed to the accelerator on a DIMM, to at most 2$\times$ of the bandwidth of a single DDR memory channel to the host CPU, but also the memory capacity to less than the memory capacity of a standard DIMM due to the DIMM space occupied by the accelerator package.

(D3) Inefficient arbitration of concurrent memory requests from both the host CPU and the PNM accelerator. There are cases that both the host CPU and the PNM accelerator need

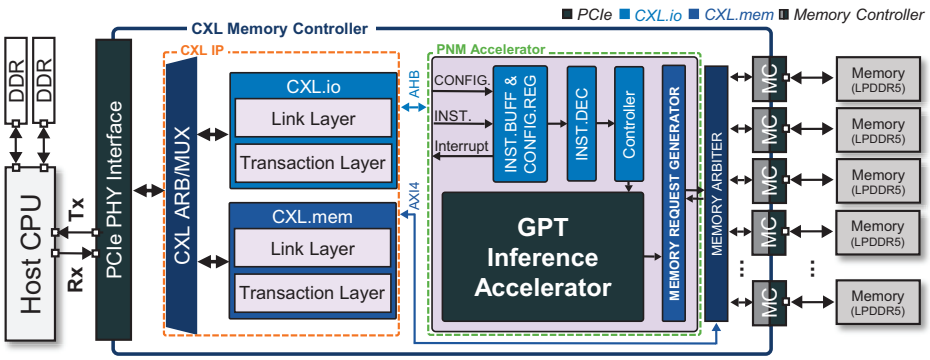


Fig. 6: CXL-PNM controller architecture.

to access the memory local to the PNM accelerator. As the memory channel is shared between the host CPU and the PNM accelerator, we need an arbitration mechanism. Nonetheless, a hardware-based arbitration is impossible because of the tight DDR timing parameters. Alternatively, a hand-shaking-based arbitration is feasible but not efficient because of the following reasons. While the PNM accelerator accesses its local memory, DIMM-PNM needs to completely block the memory requests from the host CPU until it completes a given acceleration task. In the meantime, the host CPU must keep polling a designated address shared between the host CPU and the PNM accelerator until DIMM-PNM informs the host CPU of completing the task and releasing the memory channel by setting a value to the address. DIMM-PNM must rely on such a polling mechanism because the JEDEC-compliant DDR interface does not have any pin to send an interrupt to the host CPU.

(D4) Conflicts with memory address interleaving techniques. In conventional systems, a host CPU has multiple memory channels, each memory channel comprises one or more ranks, and each rank consists of dozens of banks. To exploit memory level parallelism, the host CPU interleaves memory addresses across channels, DIMMs, and banks. While such interleaving techniques improve the efficiency of memory accesses by the host CPU, a contiguous memory region is interleaved across multiple memory channels, DIMMs, and banks. That is, an accelerator can efficiently access only a fraction of the memory region mapped to a bank and a DIMM for PIM and PNM, respectively. To tackle this problem, special data mapping techniques have been proposed to place a large contiguous memory region on a bank or a DIMM [14]. However, such mapping techniques hurt the performance/efficiency of memory accesses by the host CPU especially when the host CPU needs to operate on the memory region together.

In contrast, CXL-PNM can efficiently overcome these four disadvantages. Specifically, as a PNM solution, CXL-PNM reuses standard LPDDR5X DRAM packages, naturally tackling (D1). Obviating (D2), CXL-PNM is based on a popular PCIe card form factor, which is much larger than the DIMM form factor and plugged even into 1U servers with a 90-degree PCIe riser card. The right form factor and choice of a DRAM technology together allow CXL-PNM to expose 10× higher PNM bandwidth than the DIMM-PNM based on DDR5 DRAM. As CXL is developed to support variable latency between the CXL IP and the memory controllers in the CXL controller, CXL-PNM can easily accommodate a hardware-based arbiter for (D3). Lastly, a single CXL memory module provides a large enough capacity to store an LLM model as a memory expansion device, and thus a single CXL controller on the module is able to access all the DRAM packages on the module (with its own local interleaving techniques). Besides, the CXL memory module is exposed to the host CPU as a NUMA node, and each NUMA node accesses its local memory as a large contiguous memory region. As such, CXL-PNM neither demands any special data mapping techniques nor affects the efficiency of memory accesses to both the host CPU and the accelerator, effectively resolving (D4).

B. CXL-PNM Controller Architecture

Building on the CXL memory module (§IV), we present a CXL controller architecture integrating an LLM inference accelerator (referred to as CXL-PNM controller hereafter). Fig. 6 depicts the overall CXL-PNM controller architecture. Compared to a standard CXL controller for a memory expansion capability (§II-A), the CXL-PNM controller places an accelerator between the 'CXP IP' and the memory controllers (MCs). It is designed not only to operate as a CXL Type 3 memory expansion device but also to efficiently manage its accelerator. For such purposes, it leverages both CXL.io and the CXL.mem IPs that are also in the standard CXL controller. Specifically, the 'CXL.mem IP' is connected to an 'Arbiter' which arbitrates memory requests from the host CPU and the PNM accelerator through AXI4 [3], an open-standard, high-performance parallel bus used to connect on-chip IP blocks. This is a unique capability of CXL-PNM compared to DIMM-PNM (*cf.* (D3) in §V-A). The CXL.io IP is connected to the accelerator's controller through AHB [2], an open-standard, on-chip interconnect to manage functional blocks in a system-on-a-chip. CXL.io is used as a side-band interface for the host CPU to configure, program, and control the accelerator.

C. LLM Inference Accelerator Architecture

The LLM inference accelerator architecture consists of (1) a control unit, (2) a register file manager, (3) a matrix processing

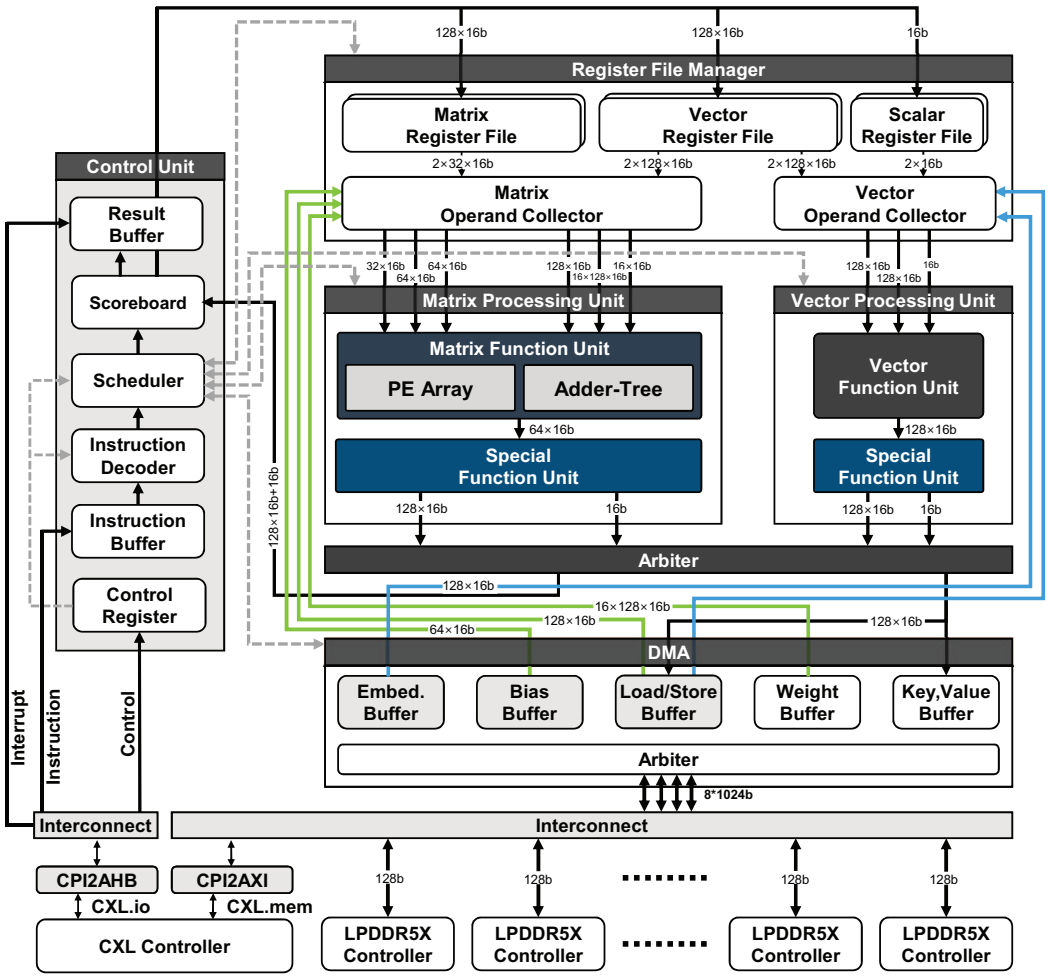


Fig. 7: LLM inference engine architecture.

unit (MPU), (4) a vector processing unit (VPU), and (5) DMA (direct-memory-access) engine as depicted in Fig. 7. To design our LLM inference accelerator, we take the DFX architecture [19] as a baseline and modify/enhance it to fully exploit the capabilities of LPDDR5X-based CXL-PNM.

Specifically, first, the DFX's matrix function unit (MFU) in the MPU has only adder-tree units designed to perform GEMV operations. However, we observe that the `sum` stage demanding GEMM operations can be a latency and throughput bottleneck for the DFX architecture without a dedicated GEMM unit; as the number of input tokens increases, the `sum` stage begins to dominate the latency and throughput. Hence, we add a PE (processing-element) array of 64×32 FP16 MAC (multiply-accumulate) units and matrix manipulation units such as `transpose` and `im2col` to the DFX's MFU to accelerate GEMM operations, as shown in Fig. 8.

Second, we also six new instructions to the DFX instruction set architecture for user programs to use the PE array. These instructions are `MPU_MM_PEA`, `MPU_MM_REDUMAX_PEA`, `MPU_MASKEDMM_PEA`, `MPU_MASKEDMM_REDUMAX_PEA`, `MPU_CONV2D_PEA`, and `MPU_CONV2D_GELU_PEA` to the original instruction set architecture (ISA) of the DFX architecture, where `PEA`, `MM`, `REDUMAX`, and `CONV2D` denote PE array, matrix multiplication, reduced max, and 2D convolution operations, respectively.

Third, the original DFX populates only one HBM2 providing 460 GB/s of memory bandwidth, whereas our LPDDR5X-based CXL-PNM offers 1.1 TB/s. Exploiting the fact that CXL-PNM provides more than $2\times$ higher memory bandwidth than DFX does and the dimension size of an attention head is a multiple of 128 [10], we double the tile dimension l [19] from 64 to 128. As such, for example, the number of FP16 MACs at each adder-tree lane in the MFU increases from 64 to 128.

Lastly, we remove the router from DFX. The router is implemented to facilitate device-to-device direct communications over PCIe, which was needed for multiple-device acceleration leveraging the model parallelism. Instead, we exploit the fact that multiple CXL devices constitute a unified memory address space with the host CPU. As such, we let the host CPU orchestrate the device-to-device communications using the DMA engine in each CXL-PNM device.

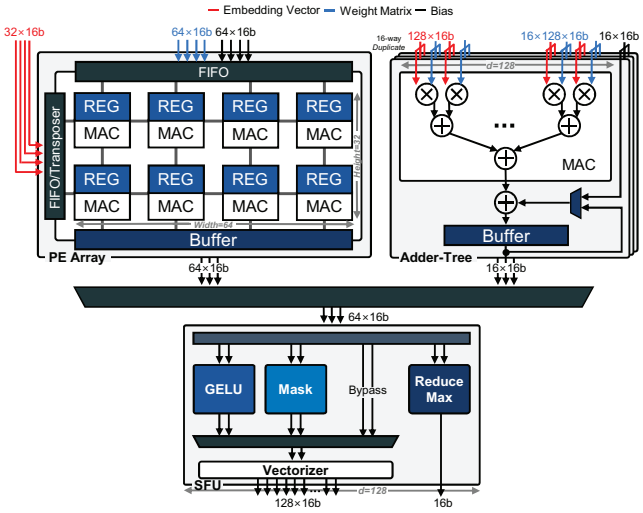


Fig. 8: MPU architecture.

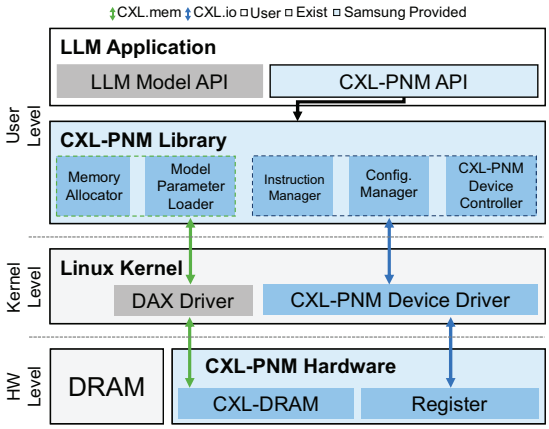


Fig. 9: The CXL-PNM software stack.

VI. Software Stack for CXL-PNM and LLM Inference Acceleration

For users to seamlessly and transparently use the CXL-PNM platform, we also develop a software stack. Fig. 9 depicts our CXL-PNM software stack at a high level, which consists of a device driver and a Python library.

CXL-PNM device driver. The host CPU may see the CXL memory as a DAX device through the default Linux DAX driver. Then, using `mmap`, the CXL-PNM device driver registers CXL memory and register (CXL.mem and CXL.io) regions to `/dev/dax0.0` and `/dev/mem`, respectively. This allows user-space programs such as the CXL-PNM library running on the host CPU to directly access CXL.mem regions for model parameters with load/store instructions. It is one of the key benefits of using CXL-PNM that the host CPU can directly access the memory space and share it with the accelerator, unlike PCIe-based accelerators (§II-A). Building on such a mechanism, the CXL-PNM device driver provides APIs that facilitate the user-space programs to configure and program CXL-PNM device-specific registers in CXL.io regions (*e.g.*, ❶ in §V-C). Lastly, the CXL-PNM device driver implements an interrupt service routine (ISR) that defines what to handle when the host CPU receives an interrupt from the CXL-PNM accelerator through the MSI-X (Message Signaled Interrupt X [5]) interface; the CXL-PNM accelerator asserts an interrupt when it completes the execution of code. Note that CXL-PNM also supports a polling-based mechanism to notify the host CPU of the completion.

CXL-PNM Python library. In this work, we choose to support Python first for CXL-PNM because it is one of the most popular ML/AI programming frameworks. Users can run their Python-based code on a CXL-PNM platform without any change to the code, as APIs of the standard Python-based library in the code can be automatically replaced with those of the CXL Python library at the dynamic linking time. As depicted in Fig. 9, the CXL-PNM library provides APIs for allocating memory space and loading model parameters to CXL memory, as well as having the CXL-PNM accelerator accelerate layer functions commonly used by LLMs, such as `LayerNorm`, `Conv1D`, `MaskedMM`, `Softmax`, and `GELU`.

The flow of accelerating an API for a layer function using CXL-PNM is as follows. ❶ When an API is invoked by a user program, it configures the control register and programs the instruction buffer with acceleration code[3] (§V-C). The control registers in the control unit (Fig. 7) provide ten 32-bit registers not only to store various architectural parameters of a given LLM model, such as the number of decoding layers, input tokens, and output tokens but also the addresses of memory regions that the GPU inference engine will operate on. ❷ The API stores the given input to a designated input buffer of the accelerator, and then the accelerator executes the code through CXL-PNM driver APIs. ❸ When the CXL-PNM accelerator completes executing the code, it stores the result (*e.g.*, an output token) to a designated output buffer, and then sends an interrupt to the host CPU. ❹ Receiving the interrupt, the CXL-PNM driver runs an ISR. This makes the API bring the result from the output buffer and returns it to the user program that invoked the API.

VII. Evaluation Methodology

As our baseline, we use a GPU appliance, an NVIDIA DGX system comprising eight A100 GPUs, each with 40 GB memory and 1.55 TB/s bandwidth [32]. We run the open pre-trained transformer (OPT) models [52] on the GPU appliance with GPU-optimized FasterTransformer [33] that provides tensor parallelism and pipeline parallelism with multiple GPU inference. Among the six OPT models, we focus on OPT-13B whose memory requirement can be accommodated by a single GPU device and OPT-66B, which is the largest one requiring all the GPU devices in the DGX system.

[3]Each layer function accelerated by CXL-PNM includes a sequence of accelerator instructions (acceleration code). For example, `LayerNorm` comprises an acceleration code that calculates mean and variance, multiplies each weight by the inverse of standard deviation, and adds bias.

TABLE II: CXL-PNM platform architecture and operating parameters. The power consumption of the CXL-PNM controller comprises that of the CXL IPs and the LLM accelerator.

# of PEs	2,048 (peak 4.09 TFLOPS)
# of adder-tree multipliers/adders	2,048/2,032 (peak 4.09 TFLOPS)
Matrix/Vector/Scalar register files	63 MB
DMA Buffers	1 MB
I/O width of DRAM/SRAM	1,024/16,384
Technology/Frequency/Voltage	7 nm/1.0 GHz/1.0V
CXL-PNM controller max power	∼90 W
DRAM total power	∼40 W
CXL-PNM platform total power	∼150 W

We develop a prototype CXL-PNM platform using a custom CXL module, which comprises an FPGA package integrated with CXL.io and CXL.mem IPs and DRAM packages (*cf.*, Fig. 5). Then we plug it into a system based on the Intel 4th-generation Xeon CPU. We design a CXL-PNM controller including the PNM accelerator (§V-C) in RTL and map it to the FPGA for the functional verification of the CXL-PNM controller and the CXL-PNM software stack at the system level; currently, an ASIC version of the CXL-PNM controller is under development. Further details are specified in Table I and Table II.

For performance evaluation, we also develop a cycle-level performance simulator and validate it against the prototype CXL-PNM platform. Our validation shows that the performance estimated by the simulator configured with the design and timing parameters of the prototype CXL-PNM platform is within a 0.5% difference compared to the performance measured with the prototype CXL-PNM platform. For our target LPDDR5-based CXL-PNM platform, we extract the timing and power parameters from the design synthesized with a 7 nm technology, and the timing and power parameters are applied to the simulator. Lastly, we evaluate the end-to-end (including both `sum` and `gen` stages) performance of both the GPU appliance and the CXL-PNM appliance with the same number of devices after running the OPT models for 64 input and up to 1024 output tokens as representative text generation workloads in datacenters [7].

VIII. Evaluation

A. Performance and Energy Efficiency

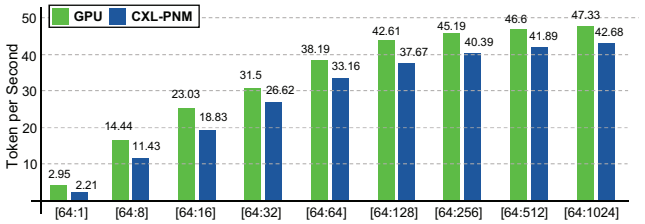
Single GPU. For a comparison between a GPU device and a CXL-PNM device, we chose to run OPT-13B, because a single GPU does not have sufficient memory capacity for running OPT-30B and 66B, and it will unacceptably increase the execution time (§III-A). Note that the OPT guideline [30] recommends to run OPT-13B on two GPU devices as the needed memory capacity to run it is close to the full memory capacity of a GPU device. However, to stress the compute resources and memory bandwidth of the GPU and CXL-PNM devices, we run it using a single device.

Fig. 10 plots the throughput (tokens/second) and energy efficiency (tokens/energy) values with the GPU and the CXL-PNM, as we increase the number of output tokens from 1 to 1024 for 64 input tokens (*i.e.*, a typical range of input and output tokens observed by inference services [7]). This shows that the CXL-PNM gives only 10.8% lower throughput but 2.9× higher energy efficiency for 1024 output tokens. For using a single acceleration device, the latency per inference is inversely proportional to the throughput, and the CXL-PNM offers 10.9% higher latency than the GPU. The CXL-PNM device provides slightly lower throughput than the GPU because it is provisioned with less compute resource and lower bandwidth than the GPU (1.1TB/s vs. 1.55TB/s). However, when the CXL-PNM device accelerates smaller OPT models, such as OPT-1.3B, 2.7B, and 6.7B, which demand less compute resources and memory bandwidth than OPT 13B, it gives 59%, 38%, and 2% lower latency than the GPU for 1024 output tokens. When compute resources (*i.e.*, maximum FLOPS) and memory bandwidth are not constrained (*e.g.*, OPT-1.3B, 2.7B, and 6.7B), an application-specific accelerator generally performs better than a general-purpose processor like GPU with much less von Neumann overheads. This also explains why the CXL-PNM device offers higher energy efficiency than the GPU; the CXL-PNM device consumes 69.91% lower power than the GPU (253 Watts vs. 77.1 Watts) for processing 1024 output tokens with OPT-13B.

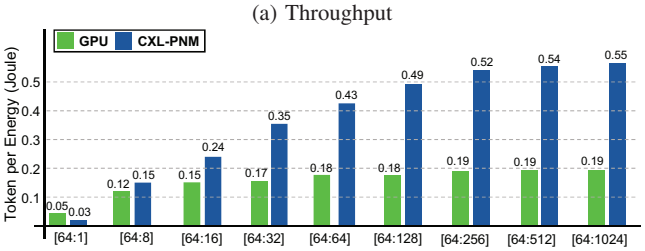
Lastly, these results do not capture the benefit provided by a larger memory capacity of CXL-PNM in Fig. 10. When we run OPT-30B with a single GPU and a single CXL-PNM device, we observe that the CXL-PNM device gives 138.8× lower latency (*i.e.*, ∼138× higher throughput) and 127.9× higher energy efficiency for 1024 output tokens, as the GPU spends most of the execution time for frequent transfers of model parameters from the host CPU memory/storage.

GPU appliance with multiple GPU devices. To compare CXL-PNM with a GPU appliance, we choose OPT-66B that a single GPU device with limited memory capacity cannot efficiently handle. As such, we exploit the model parallelism to distribute the layers of OPT-66B and model parameters corresponding to the layers across eight GPU devices, each with 40GB of memory capacity. In contrast, a single CXL-PNM device with 512GB of memory capacity can accelerate the OPT-66B model without the need to exploit the model parallelism or transfer model parameters required by computing a given GPT layer from the host CPU's large memory/storage. Nonetheless, we use a CXL-PNM appliance with eight CXL-PNM devices, where each CXL-PNM device running a single instance of OPT-66B and 8 CXL-PNM devices concurrently process eight inference requests, leveraging the data parallelism.

Fig. 11 shows that a CXL-PNM appliance can offer 53% higher throughput and 4.4× higher energy efficiency than the GPU appliance. Note that a single CXL-PNM device accelerating OPT-66B suffers from increased latency per inference request due to the limited bandwidth and compute resources, compared to those of the GPU appliance. To reduce the latency, we may employ both the model parallelism and the data

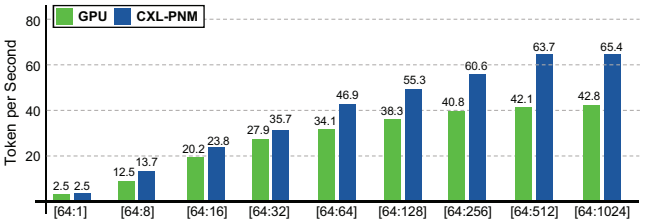


(a) Throughput

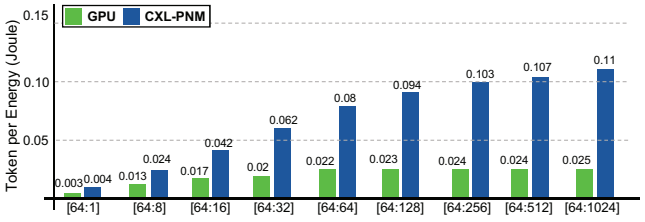


(b) Energy efficiency

Fig. 10: Comparison of the throughput and energy efficiency of a CXL-PNM device with those of a single GPU device running OPT-13B.



(a) Throughput



(b) Energy efficiency

Fig. 11: Comparison of the throughput and energy efficiency of a GPU appliance with eight GPU devices with those of a CXL-PNM appliance with eight CXL-PNM devices running OPT-66B. While the GPU appliance uses model parallelism and processes a given input sequence at a time, the CXL-PNM appliance exploits data parallelism handling eight input sequences in parallel with eight CXL-PNM devices.

parallelism for the CXL-PNM. For example, we can distribute an instance of OPT-66B across two CXL-PNM devices and accelerate four instances with eight CXL-PNM devices in the CXL-PNM appliance. This gives 44% lower latency per inference than the CXL-PNM appliance accelerating eight instances while providing 36% higher throughput and 3.3× higher energy efficiency than the GPU appliance. Lastly, as we increase the model parallelism further, we can distribute an instance across all eight CXL-PNM devices. This incurs the same amount of device-to-device communications, but each CXL-PNM device accelerates fewer layers, which demands much less compute resources and memory bandwidth, than the two previous cases. In such a case, as explained previously, a CXL-PNM device can more efficiently accelerate them than each GPU device. As a result, the CXL-PNM appliance delivers 23% lower latency, 31% higher throughput, and 2.9× higher energy efficiency than the GPU appliance. These are the key benefits of CXL-PNM along with superior cost benefits to the GPU, which we will discuss subsequently.

TABLE III: Comparison of the hardware and operating costs.

Metric	GPU appliance	CXL-PNM appliance
Compute device	8×A100 GPU	8×CXL-PNM
Hardware cost	8×$10,000	8×$7,000
Throughput	3.7 M tokens/day	5.65 M tokens/day
Energy consumption	43.2 KWh/day	15.4 KWh/day
Operation cost	4.47 $/day	1.59 $/day
CO_2 emission	2.46 Kg/day	0.88 Kg/day
Cost efficiency	0.83 M tokens/$	3.54M tokens/$
CO_2 efficiency	1.5 M tokens/Kg	6.42M tokens/Kg

B. Cost

We compare the hardware and operating costs of a GPU appliance with those of a CXL-PNM appliance with eight CXL-PNM devices (Table III). For the operating cost, we consider the consumed electricity which is proportional to the environmental cost (*i.e.*, CO_2 emission). We assume the electricity cost [13] is based on the cheapest one in the U.S. (*i.e.*, 10.35 cents per kWh in Idaho). This analysis shows that the GPU appliance is 1.42× and 2.8× more expensive than the CXL-PNM appliance for the hardware and energy costs, respectively; we compare the costs of only the GPU and CXL-PNM devices after excluding the cost of other appliance components such as that of the host CPU and its DRAM and SSD). The energy efficiency of the CXL-PNM appliance reduces the amount of CO_2 emission from 2.46Kg to 0.88Kg per day. In terms of electricity cost and CO_2 emission, the CXL-PNM appliance is 4.3× more efficient than the GPU appliance.

IX. Discussion

Scalability. As the size of LLMs increases, it may require memory capacity larger than what CXL-PNM can provide (*e.g.*, 512 GB). However, efforts have been made to curb further increases in memory capacity requirements, such as the Mixture of Experts (MoE) [42]. Furthermore, with a 6.4× larger capacity than GPU, CXL-PNM can decrease the number of devices required to run larger LLMs, thereby proportionally reducing the costs associated with (1) system hardware and (2) device-to-device communications, which is a key performance bottlenecks (§III-A). For example, to handle a hypothetical LLM requiring a memory capacity of 1.25 TB, we need 3 CXL-PNM and 16 GPU devices, respectively. In such a scenario, CXL-PNM costs 87% less than GPU. Besides, our conservative estimation for such a model shows that the GPU

and CXL-PNM devices spend 30% and 10% of runtime, respectively, for the device-to-device communications. Lastly, note that LPDDR capacity per die has been increasing and CXL-PNM will be able to support up to 2TB with 24Gb and 32Gb dies and more advanced die-stacking technology that are currently under development.

Error Correcting Capability. As CXL-PNM targets a datacenter-scale deployment, advanced RAS (Reliability, Availability, and Serviceability) features, such as ECC (error correcting code) [40] and ECS (error check and scrub), must be considered. Like the latest DDR5 and HBM3 [16] DRAM types, LPDDR5X is implemented with on-die ECC. DDR5 supports side-band ECCs, such as SECDED (single-bit error correction and double-bit error detection) and Chipkill (single-symbol error correction), cost-effectively because a DRAM rank consists of several DRAM devices with narrow datapath (few to several I/O widths per package) (see Table I). By contrast, the DRAM types with wide datapaths, such as HBM3 and LPDDR5X, involve few devices per transaction at best, incurring high costs for side-band ECCs. Instead, LPDDR5X supports inline ECC, which stores the parity bits in the same device as the data bits of a codeword, thus dedicating a fraction of the memory to ECC code storage to enhance RAS. Lastly, LPDDR5X adopts link ECC to detect and correct errors occurring while transferring data to/from the memory devices. Combined with ECS, LPDDR5X provides enough error detection and correction mechanisms to CXL targeting datacenter scale memory.

X. Related Work

LLM acceleration. A body of prior work has proposed to accelerate LLMs by distributing computation of layers and storage of model parameters associated with the layers across multiple GPUs [1], [38], [44], [50]. The key point of inference speedup comes from combing tensor and pipeline parallelism in conjunction with memory optimization [38], [44]. DeepSpeed proposes a memory optimization technique to improve the speed of training and inference [1], [38]. Besides, another body of prior work has proposed domain-specific accelerator architectures [17], [19], [21], exploiting the compute characteristics of the `gen` stage. DFX [19] and FlexRun [21] employ an adder-tree architecture that accelerates only GEMV operations in the `gen` stage. However, they only consider optimization of the `gen` stage, not end-to-end acceleration encompassing the `sum` stage that we address in this work.

Processing-Near/In-Memory. A large body of prior work has explored diverse PIM and PNM architectures [4], [9], [12], [20], [27]–[29], [36]. These architectures share a common concept, accelerating memory-bandwidth-intensive operations in and near memory. The PIM architectures exploit the array and/or bank-level parallelism to expose the high on-chip memory bandwidth [12], [29]. The PNM architecture is typically based on placing an accelerator on a DIMM and exploiting rank-level parallelism [27]. However, the PIM and PNM architectures suffer from the limitations discussed in §V-A.

XI. Conclusion

We have proposed CXL-PNM, a CXL-based processing-near-memory platform to accelerate Transformer-based large language models (LLMs). We identified that the LPDDR5X provide unique and intriguing trade-offs between bandwidth and capacity, which are adequate for processing ever-growing LLMs. Through designing a CXL-PNM controller architecture augmented with an LLM inference accelerator, developing a corresponding software stack, and bringing up an FPGA prototype, we demonstrated that a CXL-PNM appliance with 8 CXL-PNM devices offers 23% lower latency, 31% higher throughput and 2.9× higher energy efficiency at 30% lower hardware cost than a GPU appliance with 8 GPU devices.

Acknowledgment

This work was supported in part by grants from Samsung Electronics, PRISM, one of the seven centers in JUMP 2.0, a Semiconductor Research Corporation (SRC) program sponsored by DARPA, and by Institute of Information Communications Technology Planning Evaluation (IITP) grant funded by the Korea government (MSIT) [NO.2021-0-01343, IITP-2023-RS-2023-00256081].

References

[1] R. Y. Aminabadi, S. Rajbhandari, A. A. Awan, C. Li, D. Li, E. Zheng, O. Ruwase, S. Smith, M. Zhang, J. Rasley, and Y. He, "DeepSpeed-Inference: Enabling Efficient Inference of Transformer Models at Unprecedented Scale," in *SC22*, 2022.

[2] ARM, "AMBA® AHB Protocol Specification," https://developer.arm.com/documentation/ihi0033/, accessed in 2023.

[3] ARM, "AMBA® AXI Protocol Specification," https://developer.arm.com/documentation/ihi0022/, accessed in 2023.

[4] H. Asghari-Moghaddam, Y. H. Son, J. Ahn, and N. S. Kim, "Chameleon: Versatile and Practical near-DRAM Acceleration Architecture for Large Memory Systems," in *MICRO*, 2016.

[5] R. Blankenship, "Compute Express Link (CXL): Memory and Cache Protocols," https://snia.org/sites/default/files/SDC/2020/130-Blankenship-CXL-1.1-Protocol-Extensions.pdf, 2020.

[6] A. Borzunov, D. Baranchuk, T. Dettmers, M. Ryabinin, Y. Belkada, A. Chumachenko, P. Samygin, and C. Raffel, "Distributed Inference and Fine-tuning of Large Language Models Over The Internet," 2023. [Online]. Available: https://openreview.net/forum?id=HLQyRgRnoXo

[7] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei, "Language Models are Few-Shot Learners," in *NeurIPS*, 2020.

[8] P. Budzianowski and I. Vulić, "Hello, It's GPT-2 – How Can I Help You? Towards the Use of Pretrained Language Models for Task-Oriented Dialogue Systems," *arXiv:1907.05774*, 2019.

[9] B. Y. Cho, J. Jung, and M. Erez, "Accelerating Bandwidth-Bound Deep Learning Inference with Main-Memory Accelerators," in *SC21*, 2021.

[10] J. Choi, J. Park, K. Kyung, N. S. Kim, and J. Ahn, "Unleashing the Potential of PIM: Accelerating Large Batched Inference of Transformer-Based Generative Models," *IEEE Computer Architecture Letters*, vol. 22, no. 2, 2023.

[11] CXL Consortium, "Compute Express Link (CXL)," https://www.computeexpresslink.org, accessed in 2023.

[12] F. Devaux, "The true Processing In Memory accelerator," in *IEEE Hot Chips 31 Symposium*, 2019.

[13] ENERGYSAGE, "How much does electricity cost by state?" [Online]. Available: https://www.energysage.com/local-data/electricity-cost/

[14] A. Farmahini-Farahani, J. Ahn, K. Morrow, and N. S. Kim, "NDA: Near-DRAM Acceleration Architecture Leveraging Commodity DRAM Devices and Standard Memory Modules," in *HPCA*, 2015.

[15] P. Gao, S. Geng, Y. Qiao, X. Wang, J. Dai, and H. Li, "Scalable Transformers for Neural Machine Translation," *arXiv:2106.02242*, 2021.

[16] S. Gurumurthi, K. Lee, M. Jang, V. Sridharan, A. Nygren, Y. Ryu, K. Sohn, T. Kim, and H. Chung, "HBM3 RAS: Enhancing Resilience at Scale," *IEEE Computer Architecture Letters*, vol. 20, no. 2, 2021.

[17] T. J. Ham, Y. Lee, S. H. Seo, S. Kim, H. Choi, S. J. Jung, and J. W. Lee, "ELSA: Hardware-Software Co-design for Efficient, Lightweight Self-Attention Mechanism in Neural Networks," in *ISCA*, 2021.

[18] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," in *CVPR*, 2016.

[19] S. Hong, S. Moon, J. Kim, S. Lee, M. Kim, D. Lee, and J.-Y. Kim, "DFX: A Low-latency Multi-FPGA Appliance for Accelerating Transformer-based Text Generation," in *MICRO*, 2022.

[20] R. B. Hur and S. Kvatinsky, "Memory Processing Unit for in-memory processing," in *2016 IEEE/ACM International Symposium on Nanoscale Architectures (NANOARCH)*, 2016.

[21] S. Hur, S. Na, D. Kwon, J. Kim, A. Boutros, E. Nurvitadhi, and J. Kim, "A Fast and Flexible FPGA-Based Accelerator for Natural Language Processing Neural Networks," *ACM Trans. Archit. Code Optim.*, vol. 20, no. 1, Feb 2023.

[22] Intel Corporation, "Intel® Agilex™ 7 FPGA I-Series Development Kit," https://www.intel.com/content/www/us/en/products/details/fpga/development-kits/agilex/i-series/dev-agi027.html, accessed in 2023.

[23] JEDEC, "DDR5 SDRAM," 2022.

[24] JEDEC, "GRAPHICS DOUBLE DATA RATE (GDDR6) SGRAM STANDARD," 2023.

[25] JEDEC, "High Bandwidth Memory DRAM (HBM3)," 2023.

[26] JEDEC, "Low Power Double Data Rate (LPDDR) 5/5X," 2023.

[27] L. Ke, X. Zhang, J. So, J.-G. Lee, S.-H. Kang, S. Lee, S. Han, Y. Cho, J. H. Kim, Y. Kwon, K. Kim, J. Jung, I. Yun, S. J. Park, H. Park, J. Song, J. Cho, K. Sohn, N. S. Kim, and H.-H. S. Lee, "Near-Memory Processing in Action: Accelerating Personalized Recommendation With AxDIMM," *IEEE Micro*, vol. 42, Jan 2022.

[28] Y.-C. Kwon, S. H. Lee, J. Lee, S.-H. Kwon, J. M. Ryu, J.-P. Son, O. Seongil, H.-S. Yu, H. Lee, S. Y. Kim, Y. Cho, J. G. Kim, J. Choi, H.-S. Shin, J. Kim, B. Phuah, H. Kim, M. J. Song, A. Choi, D. Kim, S. Kim, E.-B. Kim, D. Wang, S. Kang, Y. Ro, S. Seo, J. Song, J. Youn, K. Sohn, and N. S. Kim, "25.4 A 20nm 6GB Function-In-Memory DRAM, Based on HBM2 with a 1.2TFLOPS Programmable Computing Unit Using Bank-Level Parallelism, for Machine Learning Applications," in *IEEE International Solid-State Circuits Conference (ISSCC)*, 2021.

[29] S. Lee, S.-h. Kang, J. Lee, H. Kim, E. Lee, S. Seo, H. Yoon, S. Lee, K. Lim, H. Shin, J. Kim, O. Seongil, A. Iyer, D. Wang, K. Sohn, and N. S. Kim, "Hardware Architecture and Software Stack for PIM Based on Commercial DRAM Technology : Industrial Product," in *ISCA*, 2021.

[30] Meta, "OPT," https://github.com/facebookresearch/metaseq/tree/main/projects/OPT, accessed in 2023.

[31] NVIDIA, "NVIDIA H100 Tensor Core GPU." [Online]. Available: https://resources.nvidia.com/en-us-tensor-core/nvidia-tensor-core-gpu-datasheet

[32] NVIDIA, "NVIDIA DGX A100 System Architecture," https://images.nvidia.com/aem-dam/en-zz/Solutions/data-center/dgx-a100/dgxa100-system-architecture-white-paper.pdf, 2020.

[33] NVIDIA, "FasterTransformer," https://github.com/NVIDIA/FasterTransformer, accessed in 2023.

[34] OpenAI, "OpenAI: Introducing ChatGPT," 2023. [Online]. Available: https://openai.com/blog/chatgpt

[35] G. Park, B. Park, M. Kim, S. Lee, J. Kim, B. Kwon, S. J. Kwon, B. Kim, Y. Lee, and D. Lee, "LUT-GEMM: Quantized Matrix Multiplication based on LUTs for Efficient Inference in Large-Scale Generative Language Models," *arXiv:2206.09557*, 2023.

[36] J. Park, B. Kim, S. Yun, E. Lee, M. Rhu, and J. Ahn, "TRiM: Enhancing Processor-Memory Interfaces with Scalable Tensor Reduction in Memory," in *MICRO*, 2021.

[37] S. Patodia, "Micron – Investor Day," https://investors.micron.com/static-files/8d23a61f-0c3d-46bf-81a1-c466525afd82, accessed in 2023.

[38] J. Rasley, S. Rajbhandari, O. Ruwase, and Y. He, "DeepSpeed: System Optimizations Enable Training Deep Learning Models with Over 100 Billion Parameters," in *26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2020.

[39] Samsung, "Samsung Develops Industry's First CXL DRAM Supporting CXL 2.0," https://news.samsung.com/global/samsung-develops-industrys-first-cxl-dram-supporting-cxl-2-0, accessed in 2023.

[40] V. Sankaranarayanan, "Error Correction Code (ECC) in DDR Memories," https://www.synopsys.com/designware-ip/technical-bulletin/error-correction-code-ddr.html, accessed in 2023.

[41] D. D. Sharma, R. Blankenship, and D. S. Berger, "An Introduction to the Compute Express Link (CXL) Interconnect," 2023.

[42] N. Shazeer, A. Mirhoseini, K. Maziarz, A. Davis, Q. Le, G. Hinton, and J. Dean, "Outrageously Large Neural Networks: The Sparsely-Gated Mixture-of-Experts Layer," *arXiv:1701.06538*, 2017.

[43] Y. Sheng, L. Zheng, B. Yuan, Z. Li, M. Ryabinin, D. Y. Fu, Z. Xie, B. Chen, C. Barrett, J. E. Gonzalez, P. Liang, C. Ré, I. Stoica, and C. Zhang, "FlexGen: High-Throughput Generative Inference of Large Language Models with a Single GPU," *arXiv:2303.06865*, 2023.

[44] M. Shoeybi, M. Patwary, R. Puri, P. LeGresley, J. Casper, and B. Catanzaro, "Megatron-LM: Training Multi-Billion Parameter Language Models Using Model Parallelism," *arXiv:1909.08053*, 2019.

[45] SK hynix Inc, "SK hynix Introduces Industry's First CXL-based Computational Memory Solution (CMS) at the OCP Global Summit," https://news.skhynix.com/sk-hynix-introduces-industrys-first-cxl-based-cms-at-the-ocp-global-summit/, 2022.

[46] V. Solberg and W. Zohni, "Near term solutions for 3D packaging of high performance DRAM," in *IEEE 13th Electronics Packaging Technology Conference*, 2011, pp. 39–43.

[47] Y. Sun, Y. Yuan, Z. Yu, R. Kuper, C. Song, J. Huang, H. Ji, S. Agarwal, J. Lou, I. Jeong, R. Wang, J. Ahn, T. Xu, and N. S. Kim, "Demystifying CXL Memory with Genuine CXL-Ready Systems and Devices," in *MICRO*, 2023.

[48] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. u. Kaiser, and I. Polosukhin, "Attention is All you Need," in *NeurIPS*, 2017.

[49] Q. Wang, B. Li, T. Xiao, J. Zhu, C. Li, D. F. Wong, and L. S. Chao, "Learning Deep Transformer Models for Machine Translation," in *57th Annual Meeting of the Association for Computational Linguistics*, 2019.

[50] X. Wang, Y. Xiong, Y. Wei, M. Wang, and L. Li, "LightSeq: A High Performance Inference Library for Transformers," *arXiv:2010.13887*, 2021.

[51] Xilinx, "Virtex UltraScale+," https://www.xilinx.com/products/silicon-devices/fpga/virtex-ultrascale-plus.html.

[52] S. Zhang, S. Roller, N. Goyal, M. Artetxe, M. Chen, S. Chen, C. Dewan, M. Diab, X. Li, X. V. Lin, T. Mihaylov, M. Ott, S. Shleifer, K. Shuster, D. Simig, P. S. Koura, A. Sridhar, T. Wang, and L. Zettlemoyer, "OPT: Open Pre-trained Transformer Language Models," *arXiv:2205.01068*, 2022.