MoNDE: Mixture of Near-Data Experts for Large-Scale Sparse Models

Taehyun Kim[†‡] Kwanseok Choi[†] Youngmock Cho[†‡] Jaehoon Cho[†] Hyuk-Jae Lee[†‡] Jaewoong Sim[†]
†Seoul National University ‡Inter-University Semiconductor Research Center
{taehyunzzz, kwanseok.choi, fudsla, jaehoon.cho, hyukjae, jaewoong}@snu.ac.kr

ABSTRACT

Mixture-of-Experts (MoE) large language models (LLM) have memory requirements that often exceed the GPU memory capacity, requiring costly parameter movement from secondary memories to the GPU for expert computation. In this work, we present Mixture of Near-Data Experts (MoNDE), a near-data computing solution that efficiently enables MoE LLM inference. MoNDE reduces the volume of MoE parameter movement by transferring only the *hot* experts to the GPU, while computing the remaining *cold* experts inside the host memory device. By replacing the transfers of massive expert parameters with the ones of small activations, MoNDE enables far more communication-efficient MoE inference, thereby resulting in substantial speedups over the existing parameter offloading frameworks for both encoder and decoder operations.

1 INTRODUCTION

Transformer-based large language models (LLMs) have demonstrated impressive performance in a variety of natural language processing tasks such as question answering, machine translation, and even software code generation [1, 8, 13]. This outstanding model performance can largely be attributed to *unprecedented* model sizes that are constantly growing over the past years. However, scaling model capacity inevitably leads to an increase in computational costs and memory requirements, thereby making it increasingly difficult to train and serve due to limited hardware budgets, even for many leading companies in the industry [11].

Mixture of Experts (MoE) has gained attention as a method to scale model sizes *without* proportionally increasing the computation cost [2, 14]. In MoE Transformers, the feed-forward network (FFN) layer in the Transformer is replaced by the MoE FFN layer that contains multiple *expert* FFNs with a *gating* network. Because *only* a few experts are selected by the sparse gating function to perform computation for a given input token, the computational cost is relatively cheaper than non-MoE models with the same number of parameters. As such, MoE is beginning to be adopted in production LLMs, as recently demonstrated in OpenAI's GPT-4 [8].

Although MoE Transformers can significantly increase model capacity without proportionally increasing training costs, serving such MoE Transformers for inference remains challenging because *all expert parameters* still need to reside in the GPUs, which can be costly for inference serving scenarios. Existing deep learning frameworks, such as Microsoft's DeepSpeed [11], alleviate memory capacity requirements by offloading model parameters to the CPU memory or SSDs and bringing them back to the GPU when needed for computation [10, 12]. However, such parameter offloading techniques lead to considerable data movement overhead, which adversely impacts inference latency. Moreover, the latency of expert transfers cannot be effectively hidden by computation through well-known techniques such as parameter prefetching. This is because the expert parameters to transfer are *dynamically* determined just before the expert FFN computation unlike non-MoE models, based on the input activations and sparse gating functions.

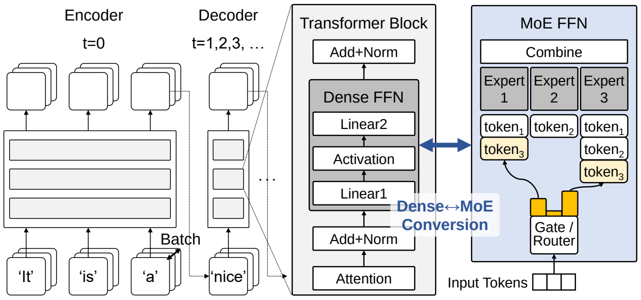


Figure 1: Overview of a Transformer block (left) and an MoE FFN layer (right) with $E = 3$ experts and top-2 routing.

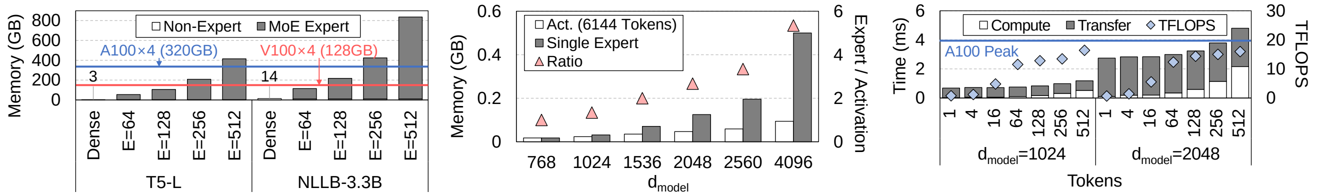
In this work, we present *Mixture of Near-Data Experts (MoNDE)*, a near-data processing (NDP) solution that efficiently serves MoE Transformers for inference in a cost-effective way. The key observation of our work is that the majority of experts in the MoE layer receive a significantly small number of tokens. Consequently, for these experts, the transfer of expert parameters to the GPU takes substantially longer than the subsequent expert FFN computation in the GPU. Furthermore, with only a few tokens to process for these *cold* experts, the high compute throughput offered by the commodity GPU is severely underutilized.

Based on the observation, MoNDE enables the paradigm of *Activation Movement*, in which the costly transfers of expert parameters are replaced with *relatively cheap* activation transfers between the GPU and the host memory device that contains the MoNDE NDP units. With the activations from the GPU (i.e., outputs from attention layers), the MoNDE NDP units perform expert computations. The resulting output activations from MoNDE are then transferred back to the GPU for subsequent Transformer operations (i.e., operations in the attention layers). We also present a novel GPU-MoNDE load-balancing scheme that exploits the *skewed* nature of MoE and runs expert computations concurrently in both the GPU and MoNDE to further reduce inference latency. Our evaluation shows that MoNDE outperforms the existing expert parameter offloading framework by up to 7.5× and 3.7× for encoder and decoder operations with an area overhead of $3.0mm^2$ for our MoNDE NDP units. In summary, this paper makes the following contributions:

- We propose MoNDE, a near-data processing system that targets MoE Transformer inference. To our knowledge, this is the *first* work that enables efficient MoE inference by exploiting NDP units.
- We provide characterizations of MoE operations and analyze the factors that contribute to performance degradation.

DAC '24, June 23–27, 2024, San Francisco, CA, USA

ACM ISBN 979-8-4007-0601-1/24/06.
https://doi.org/10.1145/3649329.3655951



(a) Scaling with E

(b) Scaling with d_{model}

(c) Expert Compute & Transfer

Figure 2: Characterization of MoE Transformers: (a) MoE scaling with E (b) MoE scaling with d_{model} (c) Latency comparison of computation and transfer of a single expert across input token sizes and d_{model} on NVIDIA A100 + PCIe Gen4 ×16.

- We demonstrate the benefit of latency reduction by replacing *Parameter Movement* with *Activation Movement* and show the effectiveness of the *GPU-MoNDE load-balancing* scheme.

2 BACKGROUND AND MOTIVATION

2.1 Mixture of Experts

The Mixture of Experts (MoE) is an ensemble technique that aims to enhance model performance. The key motivation behind using MoE is to increase the model capacity *without* a proportional increase in computation. Recent studies show that Transformer-based models that adopt MoE (dubbed MoE Transformers) achieve substantial improvements in model performance over conventional *dense* Transformers [2].

Figure 1 shows a high-level overview of MoE Transformers, where MoE is applied to the feed-forward network (FFN) within the Transformer block. The MoE FFN layer combines multiple copies of dense FFNs, called *experts*. Each expert comprises two linear layers and an activation layer in between, which is the same as the FFN layer in conventional Transformer blocks. The key component in MoE is the gating/routing network, which determines the experts to which an input token is routed. For each input token, the gating function computes the probability distribution over the experts and creates a (token × expert) score map that is used to route each token to the top-k experts. Once the experts process the routed tokens, the outputs are combined and re-organized into the original order of the input tokens. These are then forwarded to the subsequent Transformer block. Table 1 summarizes the notations regarding the model and embedding dimensions used throughout this work.

Table 1: Notations for model and embedding parameters.

Term	Description	Term	Description
B	Sequences per batch	S	Tokens per sequence
E	Experts per MoE	d_{model}	Embedding dim.
d_{ff}	Intra-FFN dim.		

2.2 Characterization of MoE Transformers

MoE Parameter Scaling. Figures 2(a) and 2(b) illustrate the parameter scaling trend of MoE models across the number of experts and embedding dimensions. The parameter sizes of MoE LLMs exhibit an asymptotically-linear growth in relation to the number of experts and can easily exceed the GPU memory capacity even for multi-GPU computing nodes. For instance, T5-Large [9] requires approximately 3 GB of memory, whereas Switch Transformers-Large [2], which is *a 128-expert MoE version* of T5-Large, demands approximately 100 GB (34×). Scaling d_{model}, which is one of the common methods of scaling LLMs, quadratically increases the gap between the expert parameters and linearly-scaling activation data.

Parameter Transfer Bottleneck. As MoE Transformers scale to trillions of parameters, relying solely on GPU memory for hosting entire model parameters is unlikely to be a viable solution. Meanwhile, emerging interconnect technologies, such as Compute Express Link (CXL), enable the addition of tens of terabytes of memory capacity to the CPU. As such, considerable efforts have recently been made to leverage the large host memory by offloading model parameters to the CPU memory and fetching them on-the-fly to the GPU when required for computation [11, 15]. For example, in an MoE-specific offloading approach [15], dense non-expert parameters are permanently stored in the GPU memory, while the sparse (but massive) MoE expert parameters are offloaded to the CPU due to the constraints of GPU memory capacity.

However, we observe that *naïvely* transferring the offloaded expert parameters back to the GPU becomes a major performance bottleneck for MoE inference. Figure 2(c) shows the execution time for an expert across the numbers of routed tokens and d_{model} sizes, which we decompose into two components: expert computations and parameter transfers. The results show that transferring a single expert parameter to the GPU takes significantly longer than the expert computation, particularly when the expert receives a small number of tokens (e.g., up to 30× longer for a single routed token).

Expert Skew and Load Imbalance. As discussed in Section 2.1, the gating network determines the expert to which a token is routed. We observe that the number of tokens that each expert receives substantially differs across the experts in the MoE layer. Figure 3 illustrates the imbalance in the distribution of tokens routed to experts, with the x-axis representing the number of routed tokens and the y-axis indicating the number of experts for each token range. Only a small number of *hot* experts process a large number of tokens, while the majority of the remaining *cold* experts process significantly fewer tokens (e.g., 0-7 tokens). This implies that the operational intensity (i.e., compute-to-memory ratio) varies among experts, with the majority of expert computations being memory-bound, thereby leading to severe underutilization of GPU cores.

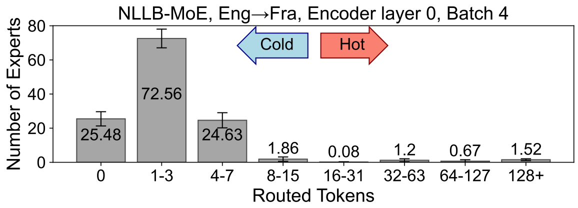
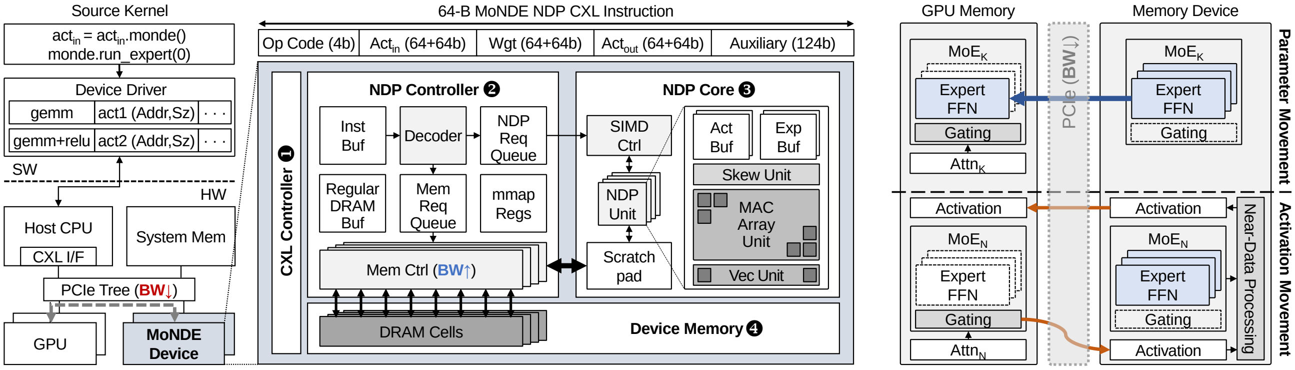


Figure 3: Average token distribution across all inputs for translation task using the NLLB-MoE model and FLORES-200 dataset.



(a) MoNDE System

(b) Parameter and Activation Movement

Figure 4: MoNDE overview: (a) MoNDE system (b) Parameter Movement and Activation Movement.

2.3 Opportunities for Near-Data MoE

The discussion in Section 2.2 implies that specialized hardware with lower peak compute could provide performance similar to the GPU for expert computation. In addition, the overall execution time of MoE can even improve over using the GPU with expert transfers if the parameter movement could be avoided, which motivates our work. To this end, we design specialized NDP hardware for effectively processing *fat and wide* matrix multiplications of cold experts (Section 3.1) with the AMove strategy (Section 3.2). We also exploit the opportunity for expert load balancing by assigning the hot/cold experts to more workload-friendly hardware (Section 3.3).

3 MONDE

This section presents Mixture of Near-Data Experts (MoNDE), a CXL-enabled memory system with custom near-data computing capabilities for MoE FFN inference. The key idea of MoNDE is to store and process large expert parameters within the CXL-based memory. Prior works move large parameters to the GPU for computation. In contrast, MoNDE takes a different approach by *transferring activations to where parameters reside* and performing expert computation on-site, which significantly reduces the burden of data movement.

3.1 MoNDE Device Architecture

Figure 4(a) shows an overview of the MoNDE system. This subsection introduces each internal component of the MoNDE device.

CXL & NDP Controllers. The CXL controller (❶) manages the CXL protocols for communicating with the host system. The CXL controller identifies host-invoked NDP instructions wrapped in the CXL *Request with Data* (RwD) messages by using the NDP flag defined in the reserved bits of a message flit. The NDP instructions are forwarded and queued to the internal memory-mapped instruction buffer of the MoNDE NDP controller. The NDP controller (❷) generates memory requests and NDP requests that load data tiles into the NDP core, and triggers expert computations. Once the computation is finished, the NDP controller stores the output activations in the designated device memory address and signals the host by setting the memory-mapped *done* register.

MoNDE NDP Core. The MoNDE NDP core (❸) processes the MoE experts located in the device memory. Our design exploits the expert skew to efficiently process *fat and wide* matrix multiplications of cold experts. An expert computation is essentially a matrix multiplication of Output C = Input $A \times$ Expert B, where matrices A and C correspond to the input and output activations, while B corresponds to the expert parameter. For cold experts, matrices A and C have small height dimensions due to the small number of routed tokens. In contrast, the width dimensions, either d_{model} or d_{ff}, are constantly large, often multiples of 256. In order to maintain high compute utilization for small token counts, our NDP core design adopts small-height 4×4 multiply-and-accumulate (MAC) processing element (PE) arrays. We use 64 of such arrays that are controlled by a SIMD controller. Using the aforementioned architecture, the MoNDE NDP core processes 4×256 matrix operations in a consecutive tile-by-tile, output-stationary manner.

Device Memory. We refer to a corporate CXL memory device [5] to build on a realistic DRAM module (❹). MoNDE uses LPDDR SDRAM, which uses wide-I/O technology for high memory bandwidth and low power. Each ×16 chip has a total of 16 Gb density and a maximum transfer rate of 8533 MT/s. Each DRAM module is composed of 32 chips that provide in total, 64 GB of memory capacity and 68 GB/s of bandwidth. By utilizing 8 memory channels, the MoNDE memory device allows access to 512 GB of memory capacity and approximately 512 GB/s of bandwidth.

3.2 Offload and Fetch Strategy

The MoE Transformers consist of the unconditionally-used *dense* parameters and the MoE expert parameters that are dynamically and only conditionally activated by the gating function. Because the latter have massive sizes that often do not fit into a single GPU system, we choose to offload all expert parameters to the MoNDE memory device, while keeping the dense parameters in the GPU memory as in [15]. The existing LLM offloading frameworks [11] need to transfer the corresponding expert parameters from the CPU memory for MoE computation and evict other experts from the GPU. We refer to this as the *Parameter Movement* (PMove) strategy. In contrast, we propose the *Activation Movement* (AMove) strategy in which MoE expert operations are processed using the MoNDE NDP core by transferring the input activation for experts from the GPU to the MoNDE device and returning the output of expert computations back to the GPU afterwards. Figure 4(b) shows the overview of the proposed AMove as compared to PMove.

Analytical Comparison of PMove and AMove. We formulate Equations 1 and 2 to show the data movement complexities of the two strategies. Typically, MoE Transformers scale towards d_{model},

d_{ff} and E, thereby making PMove scale in a *cubic* fashion [2]. Moving such massive amount of data from the memory device to the GPU on-demand will incur long latency overheads since the PCIe bandwidth is limited. In contrast, the data volume scaling of AMove can be reduced to $O(d_{model})$ when B and S are small, which is the case for many MoE inference tasks. As illustrated in Figure 2(b), there is a significant gap between the data volumes of PMove and AMove for LLMs that scale towards the expert size and embedding dimensions.

$$\text{Parameter Movement} = 2 \times E \times d_{model} \times d_{ff} \quad (1)$$

$$\text{Activation Movement} = 2 \times B \times S \times d_{model} \quad (2)$$

3.3 GPU-MoNDE Load Balancing

We propose GPU-MoNDE load-balancing, which leverages the two hardware to concurrently process different expert operations and reduce the execution time for MoE layers. We exploit the imbalance in expert load and the computing power of the hardware units to assign workloads to each hardware based on the compute and memory intensity of each expert. Our algorithm assigns the top-H compute-intensive hot experts to the GPU and the remaining experts to the MoNDE NDP, and overlaps the GPU (PMove-to-expert) and MoNDE NDP (AMove-to-expert) execution.

The H value sensitively affects performance, as assigning too many experts to the GPU can cause excessive data movement, whereas the opposite can underutilize GPU resources. Our goal is to find the H value that balances the runtime of the GPU and MoNDE workflows (t_{GWF} and t_{MDWF}) shown in Equation 3. We use two intuitions for determining H. First, the GPU computation latency t_{GPU} and AMove latency t_{AM} are negligibly small for inference and thus are removed from consideration. Second, we approximate the PMove latency t_{PM} and MoNDE NDP computation latency t_{MD}, as shown in Equation 4, considering their bandwidth-bound nature. Here, $Expert_{GPU}$ and $Expert_{MD}$ each represent the expert parameters processed on each hardware, and BW_{PCIe} and BW_{MD} represent the PCIe and the MoNDE device memory bandwidth.

Equation 5 denotes the number of activated experts $Expert_{Activ}$, that is, experts with at least 1 input token. Lastly, equating the formulas in Equation 4 and applying Equation 5, we find the H value as Equation 6. H is computed during runtime after the gating function. We use the bandwidth value from the hardware specification, but this can be replaced by profiled bandwidths. We add a scaling factor α to micro-control H for when the overall NDP experts have increased compute intensity, making our second intuition invalid. In such cases, increasing H to offload more experts to the GPU workflow reduces end-to-end latency. Finding the scaling factor is untrivial, as many factors (e.g., tokens-per-expert, embedding size, GPU compute power) need to be considered collectively. Inspired by auto-tuning features used by [11], the MoNDE framework auto-tunes the scaling factor by periodically running profiled inference on a small set of past input batches and finding the local optima among H candidates (e.g., $H+1$, $H+2$).

$$t_{GWF} = t_{PM} + t_{GPU} \qquad t_{MDWF} = t_{AM} + t_{MD} \quad (3)$$

$$t_{PM} \approx \frac{Expert_{GPU}}{BW_{PCIe}} \qquad t_{MD} \approx \frac{Expert_{MD}}{BW_{MD}} \quad (4)$$

$$Expert_{Activ} = Expert_{GPU} + Expert_{MD} \quad (5)$$

$$H = \alpha \times Expert_{GPU} = \alpha \times \frac{BW_{PCIe}}{BW_{MD} + BW_{PCIe}} \times Expert_{Activ} \quad (6)$$

For multi-MoNDE device scenarios, the MoNDE algorithm obtains the H value by using the *aggregate* MoNDE device bandwidth. The NDP units are load-balanced by distributing expert workloads sorted by compute intensity in a round-robin manner. The expert input activations are separately transferred to each MoNDE device, after which each MoNDE NDP device processes the given input data. The output activations are retrieved from each MoNDE device to the GPU sequentially for the MoE combine operation.

3.4 Programming Model

Host Interface. MoNDE adopts a heterogeneous programming model (e.g., CUDA) in which the host launches a kernel and the NDP device executes the offloaded instruction. The host does this through the host-side MoNDE device driver, which generates and offloads MoNDE NDP instructions via the CXL interface. The MoNDE NDP controller raises the memory-mapped *done register* once a kernel execution is completed. We define two kernels: `gemm` and `gemm+relu`. The `gemm` kernel offloads an expert GEMM instruction to the MoNDE NDP. The `gemm+relu` kernel is an extension that runs a tailing activation function (i.e., ReLU or GeLU). A host kernel is compiled into a 64-Byte CXL instruction, which includes a 4-bit opcode (including reserved ops), a 48-Byte (address, data size) metadata of the input/output activation and expert parameters, and auxiliary NDP flags such as `isNDP` for identifying NDP instructions.

Memory Allocation. The host-side device driver allocates fixed-sized memory space for the MoE expert parameters and input/output activations in the MoNDE device memory during MoE layer initialization. Data in the MoNDE memory space is mapped to the DRAM *ro-ba-bg-ra-co-ch*, in order to fully utilize the DRAM bandwidth for contiguous memory accesses. To mitigate memory contention from accessing expert parameters and activations simultaneously, we map each data in different *banks*: the parameters and activations are each mapped to the even and odd-indexed banks.

Execution Flow. We demonstrate the execution flow of a MoNDE expert operation. First, the input activation data is transferred to the MoNDE memory with AMove. Once the input activations had been written to device memory, the host-side device driver issues and queues gemm instructions in the memory-mapped instruction buffer at the MoNDE NDP controller, which are decoded to generate device-side memory and NDP requests. The MoNDE memory quickly populates the MoNDE NDP scratchpad and operand buffers with expert parameter and input activation *tiles*. The tiled operands are reshaped into skewed formats and processed by the systolic array, after which the output activation tiles are written to the designated output memory space. Finally, the NDP controller raises the memory-mapped *done* flag.

4 EVALUATION

4.1 Experimental Setup

MoNDE NDP Model. To evaluate the MoNDE workflow, we first use the NVIDIA Nsight profiler to obtain a detailed MoE latency breakdown and isolate the latency for expert computation for the evaluated models. We then implement a cycle-level expert computation simulator for which we use Ramulator [6] to model our MoNDE memory. Based on the number of tokens routed to each MoNDE-offloaded expert, which is determined at runtime, the simulator outputs the latency required for the MoNDE NDP to process

Table 2: Workloads and system configurations.

Model	Non-Expert Params (GB)	Expert Params (GB)	d_{model}	E
Switch-Large-128	1.1	51.5	1024	128
NLLB-MoE	5.7	103.1	2048	128

Model	Gating	Task
Switch-Large-128	top-1	XSum Language Modeling
NLLB-MoE	top-2	FLORES-200 Machine Translation

Platform		System Configuration
CPU		Intel Xeon Silver 4310 CPU, 187 GB/s Memory Bandwidth
GPU		1× NVIDIA A100 GPU PCIe
MoNDE	Compute	64 units of 4×4 Systolic Array 264 KB Buffers @ 1 GHz
	Memory	512 GB/s Bandwidth, 512 GB Capacity
Interconnect		PCIe Gen4 ×16

the same expert computations. Finally, we replace only the CPU computation latency obtained from the profiler with the NDP computation latency obtained from the cycle-level simulator to estimate the MoNDE inference latency. Overall, the GPU performs Transformer operations while fetching the expert parameters from the CPU memory or offloading expert computations to the CPU. We focus on the single-GPU setting enhanced by CXL-expanded memory with and without NDP support, because we aim to replace costly GPU resources with more affordable NDP-enabled memory devices.

Implementation. We implement the MoNDE operation flows using Pytorch APIs. We modify the Hugging Face MoE model implementations [16] to implement a drop-less and padding-less token routing algorithm similar to [4]. We implement on-demand PMove [4], in which only the *activated experts* are fetched to the GPU, instead of over-fetching the entire experts as in [11, 15]. We use the PyTorch CUDA memory copy API for AMove between the GPU and CPU for expert computation on the CPU, which we use to model the MoNDE NDP behavior. Lastly, we implement the GPU-MoNDE load-balancing algorithm in our codebase.

Workloads. Table 2 summarizes our workloads and system configurations. We evaluate MoNDE using the pre-trained Switch Transformers and NLLB-MoE provided in the Hugging Face repository [3, 7]. For both models, we run the 128-expert MoE models, which are typically hard to fit in a single commercial GPU. We use the bfloat16 datatype, which is widely-adopted for inference tasks. We evaluate the encoder and decoder performance individually, as MoNDE can be applied to any encoder-only [1] or decoder-only [8] MoE LLMs. We use the input sequence length of 512 for each batch.

4.2 Performance

We compare the following configurations. The GPU with PMove support (GPU+PM) is the memory offloading scheme where activated MoE experts are moved to the GPU for computation. The MoNDE NDP (MD+AM) scheme runs all expert operations using the MoNDE NDP. The input activations are AMoved between the GPU and MoNDE device. The GPU-MoNDE load-balanced scheme (MD+LB) uses the MoNDE load-balancing algorithm to collaboratively use the GPU and MoNDE NDP. Both PMove and AMove are used. Lastly, the ideal single-GPU (Ideal) models a GPU with infinite memory capacity, where all MoE and non-MoE layers reside in the GPU memory. Figure 5 depicts the workflow of MoE Transformer block execution schemes with regard to parallel hardware streams.

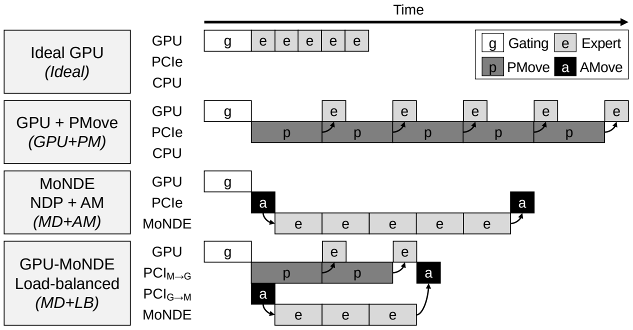


Figure 5: Comparsion between MoE workflows.

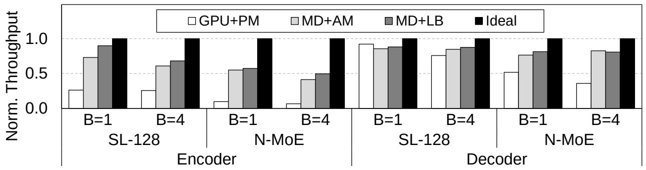
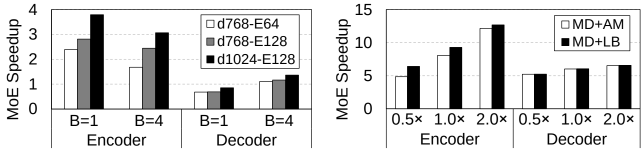


Figure 6: Normalized end-to-end throughput.



(a) d_{model} and E (b) Memory Bandwidth

Figure 7: Sensitivity study.

End-to-End Throughput. Figure 6 shows the MoNDE throughput normalized to the Ideal scenario across different batch sizes. The results show that the MD+LB scheme improves encoder and decoder throughput over GPU+PM by 3.1× and 1.1× for Switch-Large (SL-128), and by 6.7× and 1.9× for NLLB-MoE (N-MoE) on average. The performance benefit of MD+AM comes from avoiding the long PMove latency by replacing it with small AMove. MD+LB improves upon MD+AM by leveraging both the GPU and MoNDE NDP, achieving an average speedup (across SL-128 and N-MoE) of 4.9× and 1.5× over GPU+PM for the encoder and decoder.

Sensitivity. Figure 7(a) shows the speedup of MD+LB over GPU+PM for three variants of Switch Transformers with different d_{model} and E configurations. MD+LB shows increasingly higher speedups for larger models, reflecting its robustness to d_{model} and E scaling. Figure 7(b) shows the speedups of MD+AM and MD+LB over GPU+PM for NLLB-MoE (batch-4) with 0.5×-2.0× the MoNDE memory bandwidth and rate-matching NDP compute. For both the encoder and decoder, the speedups increase since higher memory bandwidth leads to latency reduction for cold experts, which comprise the majority of the MoE experts. The MD+LB constantly shows better performance over MD+AM by adaptively controlling the H value to utilize both the GPU and MoNDE NDP for expert computation. Higher memory bandwidth leads to lower and more conservative H value, which explains why the gap between the two policies is reduced. We see smaller gains for the decoder because only a small number of experts are activated.

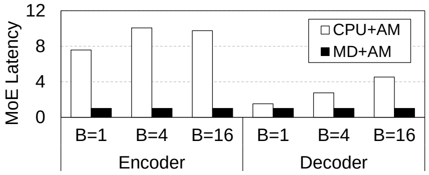


Figure 8: Comparison with the CPU.

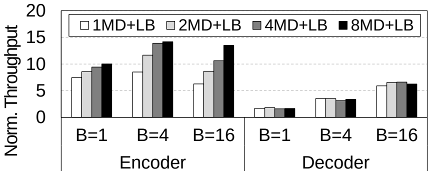


Figure 9: Multi-MoNDE evaluation.

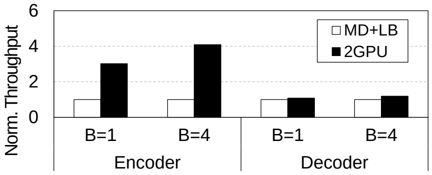


Figure 10: Comparison with multi-GPU.

Comparison with the CPU. Figure 8 compares the MoE latency of CPU expert computation (CPU+AM) and MD+LB for NLLB-MoE. MD+AM shows an average of 9.1× and 1.9× latency reductions for the encoder and decoder, which can be attributed to higher MoNDE memory bandwidth (2.7×) than the CPU memory. Even with higher CPU memory bandwidth, however, fully utilizing the CPU memory is often challenging due to remote NUMA accesses, which can degrade CPU performance. Furthermore, the CPU performance is not scalable, whereas MoNDE performance can be scaled by adding more MoNDE devices via the PCIe slots for improved throughput.

Scalability. Figure 9 presents the multi-MoNDE inference throughput for the MoE layers of NLLB-MoE, which we normalize to GPU+PM. For the encoder, employing more MoNDE devices improves performance, largely due to the increase in compute power and memory bandwidth. For the decoder, the performance gain over GPU+PM is similar across the numbers of MoNDE devices within each batch size because the small number of input tokens (i.e., 1/4/16) cannot fully utilize multiple MoNDE NDP units.

Comparison with Multi-GPU. With MoE expert parallelism, expert parameters can be distributed across multiple GPUs to fit in the GPU memory [11]. However, multi-GPU systems are inefficient when serving MoE decoders as they are auto-regressive; each input token activates only one or two experts, and the GPUs with inactive experts remain idle. Figure 10 compares the throughput between MD+LB and a 2-GPU setting for the encoder and decoder of NLLB-MoE. The multi-GPU system shows a higher throughput for the encoder due to a larger number of activated experts in each GPU, whereas for the decoder, MoNDE shows a throughput comparable to the multi-GPU system. Because a single MoNDE device provides memory capacity that is comparable to dozens of modern GPUs, MoNDE is more cost-effective in serving generative LLMs.

4.3 Area and Power Consumption

Table 3 presents the area and power of the MoNDE NDP core. We use Synopsys Design Compiler to synthesize the MoNDE systolic array with a 28 nm technology node at 1 GHz clock. We also generate on-chip buffers with a commercial memory compiler using the same technology. Our MoNDE NDP design adds 3.0 mm^2 of area overhead, which corresponds to approximately 0.9 Gb DRAM cells of our target memory. We estimate the power consumption of the base memory expander device to which we apply the MoNDE NDP unit, by using Micron DDR4-3200 power calculator and scaling to our target LPDDR device with operating voltage. The estimation shows that our memory device consumes 114.2 W and our NDP unit incurs only 1.6% of power overhead to the base memory system.

Table 3: Summary of MoNDE area and power.

Component	Systolic Array			Scratchpad
	PE	Control	Operand Bufs	
Area (mm^2)	2.042	0.053	0.289	0.570
Power (W)	0.993	0.033	0.258	0.526

5 CONCLUSION

This paper explores Mixture of Near-Data Experts (MoNDE) for enhancing MoE LLM inference through near-data processing. By replacing massive MoE expert movement invoked by data offloading techniques with cheap activation movement and processing MoE experts near-the-data, MoNDE significantly reduces MoE inference latency. When collaborating with the GPU to concurrently process MoE expert computations that are suited for each compute hardware, MoNDE achieves inference latency comparable to an ideal GPU system with infinite memory. Evaluation on MoE LLMs shows up to a 7.5× end-to-end inference speedup over a strong baseline that implements the latest parameter offloading technique.

ACKNOWLEDGMENTS

This work was supported in part by Institute of Information & Communications Technology Planning & Evaluation (IITP) grants funded by the Korean government (MSIT) (No. 2021-0-00863 and RS-2023-00256081) and a research grant from SK Hynix. The Institute of Engineering Research at Seoul National University provided research facilities for this work, and the EDA tool was supported by the IC Design Education Center (IDEC), Korea. Jaewoong Sim is the corresponding author.

REFERENCES

[1] Jacob Devlin et al. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *NAACL-HLT*.
[2] William Fedus et al. 2022. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity. *JMLR* (2022).
[3] Google. 2021. Hugging Face Switch Transformers. https://huggingface.co/docs/transformers/model_doc/switch_transformers.
[4] Haiyang Huang et al. 2023. Towards MoE Deployment: Mitigating Inefficiencies in Mixture-of-Expert (MoE) Inference. *arXiv preprint arXiv:2303.06182* (2023).
[5] Jin Hyun Kim et al. 2023. Samsung PIM/PNM for Transformer Based AI: Energy Efficiency on PIM/PNM Cluster. In *HCS*.
[6] Yoongu Kim et al. 2015. Ramulator: A fast and extensible DRAM simulator. *IEEE CAL* (2015).
[7] Meta. 2022. Hugging Face NLLB MoE Model Hub. https://huggingface.co/facebook/nllb-moe-54b.
[8] OpenAI. 2023. GPT-4 Technical Report. *arXiv preprint arXiv:2303.08774* (2023).
[9] Colin Raffel et al. 2020. Exploring the limits of transfer learning with a unified text-to-text transformer. *JMLR* (2020).
[10] Samyam Rajbhandari et al. 2021. Zero-infinity: Breaking the gpu memory wall for extreme scale deep learning. In *SC*.
[11] Jeff Rasley et al. 2020. Deepspeed: System optimizations enable training deep learning models with over 100 billion parameters. In *KD*.
[12] Jie Ren et al. 2021. ZeRO-Offload: Democratizing Billion-Scale Model Training.. In *ATC*.
[13] Swapnil Sharma et al. 2023. Stochastic Code Generation. *arXiv:2304.08243* (2023).
[14] Noam Shazeer et al. 2017. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. In *ICLR*.
[15] Liang Shen et al. 2023. SE-MoE: A Scalable and Efficient Mixture-of-Experts Distributed Training and Inference System. *arXiv:2205.10034* (2023).
[16] Thomas Wolf et al. 2020. Transformers: State-of-the-Art Natural Language Processing. In *EMNLP*.