Deepseed

Paper

https://arxiv.org/pdf/2207.00032.pdf

Paper Introduces…

- Deepseed inference
    - System solution for transformer inference
    - Has a **Multi-gpu** inference solution and a **Heterogenous** inference solution that uses gpu and NVMe in addition to gpu memory
        - NVM Express or Non-Volatile Memory Host Controller Interface Specification is an open, logical-device interface specification for accessing a computer's non-volatile storage media usually attached via the PCI Express bus.
- Deep seed inference reduces latency by up to 7.3x

Introduction

- Necessary because transformer model scale continues to grow and model size can be over a trillion parameters
- Challenges
    - Latency
    - Throughput
    - Feasibility

Latency challenges

- Using transformers in prod requires strict latency requirements
    - → batch sizes usually small
    - In small batch sizes, inference latency is lower bounded by time it takes to load all model params from memory to registers → overall memory bandwidth == meeting latency requirements of transformer model inference
- Maximizing effective memory bandwidth at small batch sizes requires reading memory at near peak memory bandwidth
    - General Matrix to Matrix Multiply (GeMM) focuses on this problem
    - Read more about gemm here: https://docs.nvidia.com/deeplearning/performance/dl-performance-matrix-multiplication/index.html
    - peak memory bandwidth = how much mem you can use before memory degrades

- For some large models, peak memory bandwidth is not sufficient for meeting inference latency constraints
    - Would require aggregating memory across many devices and optimal parallelism strats
- Optimal parallelism strate depends on whether model is Dense or sparse MoE models

Throughput Challenges

- production workloads also have throughput targets to meet cost budget
- **At small batch sizes, where the workload is still memory bandwidth bound, the latency of the workload does not increase as long as the computation is entirely overlapped with model weight reads**
    - → maximizing throughput while meeting latency requirements means also overlapping compute w model weight reads and achieving high compute efficiency
- inference kernels must achieve high memory bandwidth utilization and high compute utilization **at small batch sizes,** whereas training kernels simply need to achieve high compute utilization at much larger batch sizes
    - Meaning Inference kernels are more complicated

Feasibility Challenges Under Limited Resources

- Many ppl don't have large gpu's and models w many params cannot fit on one gpu
- They introduce deepseed inference to tackle those challenges
- Deepseed inference consists of
    - 1. Deepseed transformer - a **gpu only** solution
        - 3 layered system architecture with single gpu transformer kernels optimized for memory bandwidth, many gpu dense transformer layer for scaling, and massive gpu scale sparse transformer layer for scaling MoE transformers
        - Each layer addresses aspects of the latency challenge
    - 2. ZeRO-Inference - GPU+CPU+NVMe based solution
        - For applications that are not latency sensitive but resource constrained

Main contributions

- Single GPU transformer kernels for minimizing latency and maximizing throughput via memory-bandwidth centric fusion schedules and GeMM kernels
- A many-GPU dense transformer inference system that combines tensor-parallelism to minimize latency with inference optimized pipeline parallelism schedules and memory optimizations to maximize throughput
- A massive-GPU sparse model inference system that combines: i) expert, data, and tensor parallelism, ii) novel communication optimizations and iii) sparse kernel

optimizations to scale sparse inference on trillions of parameters across hundreds of GPUs
- ZeRO-Inference that leverages CPU, NVMe and GPU memory along with GPU compute to make massive model inference accessible with limited resources
- Evaluation
    - For **latency sensitive** scenarios, **DeepSpeed Transformer** shows latency reduction
    - For **throughput** oriented scenarios, **DeepSpeed Transformer** demonstrates over 1.5× gain over state-of-the-art
    - Evaluation of ZeRO-Inference on GPU **resource constrained** systems that shows ZeRO-Inference can support inference with 25× larger models than with GPU only solution while achieving over 50% of peak hardware performance.

Background and related work

Dense transformer models

- Size has been increasing 10x each year which many models w billions of parameters

Sparse transformers

- MoE technique essentially
- Moe adds feedforward layer w gating function and multiple experts
- Moe can be much bigger than dense models
- System and hardware needed to keep up with scale of Dense transformer models
- MoE are much bigger than dense models even though they take less computation time (through conditional computation)

System Optimizations for memory scaling

- Major challenge in scaling model sizes is in **memory bottleneck**
- Many parallelism techniques have been proposed to solve the memory bottleneck
    - Tensor parallelism
        - Splits model layers horizontally across gpu's
        - Results in lower compute due to small local problem size, all-reduce communications in each transformer layer to aggregate the partial activations
    - Pipeline parallelism
        - Splits a model vertically into pipeline stages
        - microbatching https://www.deepspeed.ai/tutorials/pipeline/#:~:text=Pipeline%20parallelism%20improves%20both%20the,can%20be%20processed%20in%20parallel.

    - ZeRO
        - Removes memory redundancies by partitioning instead of replicating
        - https://www.deepspeed.ai/tutorials/zero/
        - “ZeRO reduces the memory consumption of each GPU by partitioning the various model training states (weights, gradients, and optimizer states) across the available devices (GPUs and CPUs) in the distributed training hardware”
    - Expert parallelism
        - Places different experts on different gpu’s and executes them in parallel
        - learned top-k gating function
        - Taken from https://www.jmlr.org/papers/v23/21-0998.html
    - Optimized transformer kernels
        - stochastic transformer kernels that fused operators and reduced activation memory to support large batch sizes
        - Transformer dataflow graphs to fuse elementwise and reduction operators and accelerate training
    - DNN inference optimizations
        - Basically optimizing inference through platforms, libraries, complications, etc
            - TVM
            - Onnxruntime
            - tensorRt
        - Mainly focused on single gpu optimizations
        - Most similar lib to deepseed is fastTransformer, which which supports multiGPU inference for transformer models
        - Paper notes that model compression techniques like distillation, quantization, and sparsification can be combined w the deepseed technique

Inference optimized transformer kernels

Challenges + Solutions

- small batch performance is limited by the memory bandwidth utilization in reading model weights (repeat)
- 3 main challenges to optimizing memory bandwidth at small batch inference
    - Kernel invocation overhead
    - data-transfer between GPU cores and global memory adds an additional overhead
        - each kernel-invocation writes data to global memory which is read by GPU cores during the next kernel invocation

    - neither cuBLAS nor CUTLASS GeMM libraries are well tuned for extremely small batch sizes, and cannot achieve good memory-bandwidth utilization
- Large batch inference performance limited by compute utilization
    - Overall utilization is limited by kernel launch overhead, data transfers, and global memory across different kernels
- For these challenges, they introduce deep fusion to reduce kernel invocation and data movement overhead by fusing multiple kernels
- Also introduce custom GeMM kernel for improving memory bandwidth when batch is small
- Gemm can be used with deep fusion

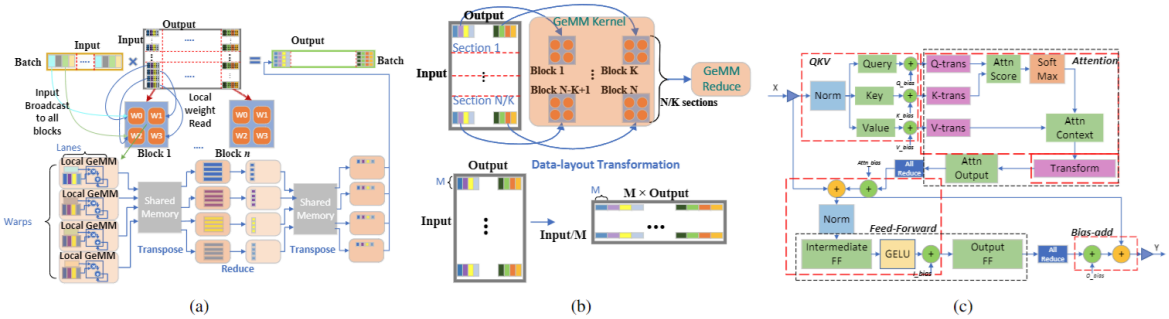
DeepFusion

- Operator fusion
    - “merging one operator (typically, an activation function) into a different operator so that they are executed together without requiring a roundtrip to memory.”
    - Typically limited to element wise operators
- In contrast, a transformer consists of operators like data layout transformations, reductions, and GeMMs which create data dependencies across thread blocks, making them difficult to fuse. This is because on GPU, if a data produced by a thread-block is consumed by a different one, a global memory synchronization is needed which invokes a new kernel
- To avoid need for global synchronization, deepfusion tiles computation

SBI-GeMM - custom GeMM for small batch size

- Can be used with deep-fusion
- Design of SBI-GeMM has 3 parts
    - Tiling strategies
    - Cooperative group reduction
        - With the previous tiling strategy, each warp in a thread block is responsible for producing a partially reduced result for a tile of outputs and a final reduction is needed across all the warps within the thread block
        - https://developer.nvidia.com/blog/cooperative-groups/
    - Leveraging full cache line
        - Reads elements on the cache line

Putting it together



- 1) qkv Gemm and input layer norm
- 2) transpstiion pluus attention
- 3) post attention layer norm and intermediate Gemm
- 4) bias and residual addition

Inference Adapted Dense transformer models on many gpu systems

- Presents model parallelism techniques used on top of single transformer kernels

Tensor parallelism

Pipeline parallelism

Massive scale sparse model inference

- More considerations needed for non dense transformer models such as sparse transformer models (i think sparse refers to MoE)
- Tensor parallelism and expert parallelism

We optimize each of the four steps in the gating function in
the following way: 1) we replace the one-hot representation of
the token to expert mapping using a table data-structure, greatly
reducing the memory overhead from eliminating all the zeros in
the one-hot vectors; 2) we create the inverse mapping (expert-totokens mapping table) from the tokens-to-expert mapping table
by simply scanning though the token-to-expert table in parallel.
3) we replace the sparse einsum based scatter operation using
a data-layout transformation that achieves the same result by

first identifying the token IDs assigned to an expert using the expert-to-token mapping table created in the previous step, and then copying these tokens to the appropriate expert location; 4) after the tokens are processed by their corresponding experts, we use a similar data-layout transformation toreplace the sparse einsum based gather operation

Democratization of large model inference

Performance evaluation

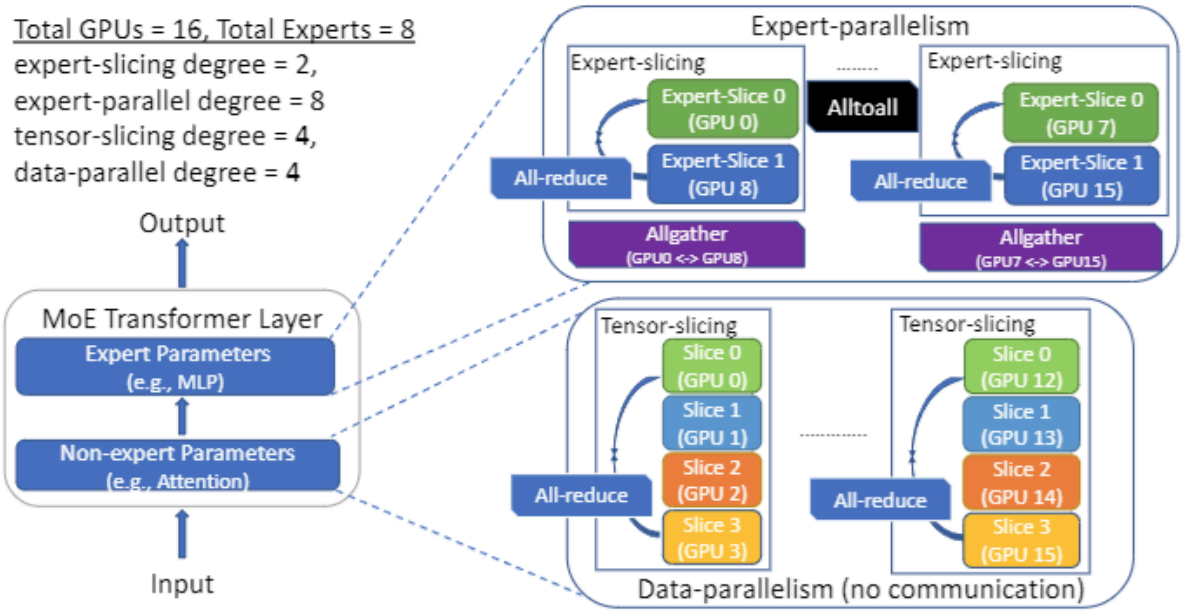


Fig. 4. Expert, data and tensor parallelism in DeepSpeed-MoE.