deepspeed

SC22 Dallas, TX | hpc accelerates.

DeepSpeed Inference: Enabling Efficient Inference of Transformer Models at Unprecedented Scale

Reza, Samyam, **Ammar**, Cheng, Du, Elton, Olatunji, Shaden, Minjia, Jeff, and Yuxiong He

https://github.com/microsoft/DeepSpeed

Model Scale
- 10+ Trillion parameters

Speed
- Fast & scalable training

Democratize AI
- Bigger & faster for all

Compressed Training
- Boosted efficiency

Accelerated inference
- Up to 10x faster & cheaper

Usability
- Few lines of code changes

What is DeepSpeed?

Multi-purpose DL optimization suite

Training
- Speed
- Scale
- Cost
- Democratization

Inference
- Latency
- Throughput
- Serving cost
- Ease-of-use

Compression
- Model size
- Latency
- Tuning cost
- Composability

...

DeepSpeed Website: https://www.deepspeed.ai/
DeepSpeed GitHub: https://github.com/microsoft/DeepSpeed

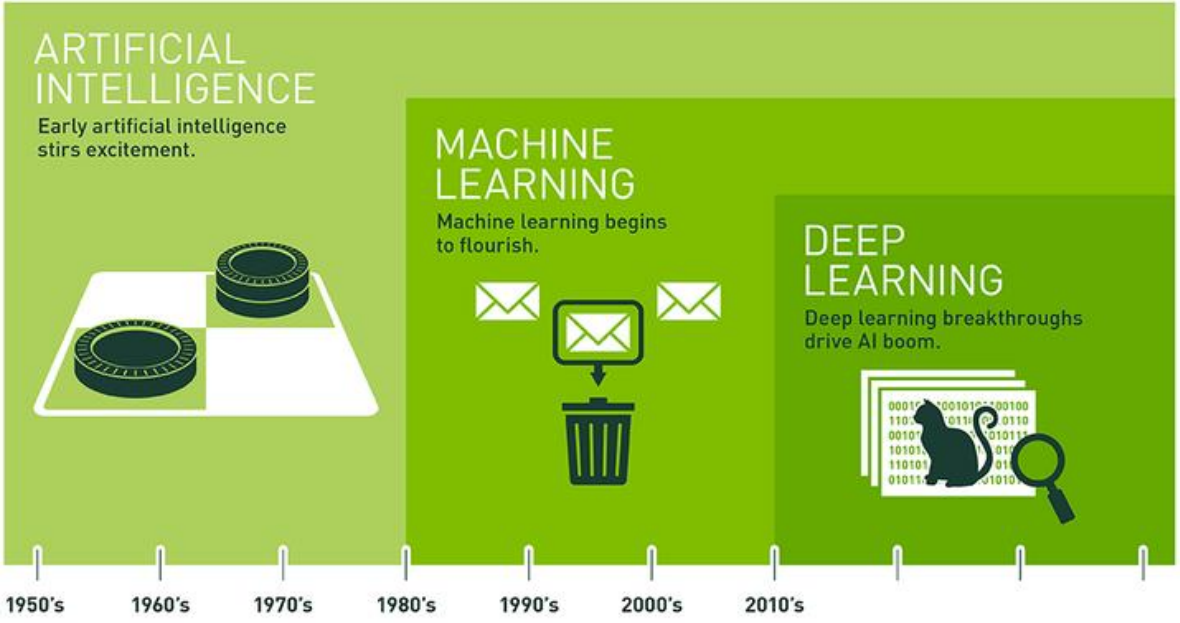
deepspeed

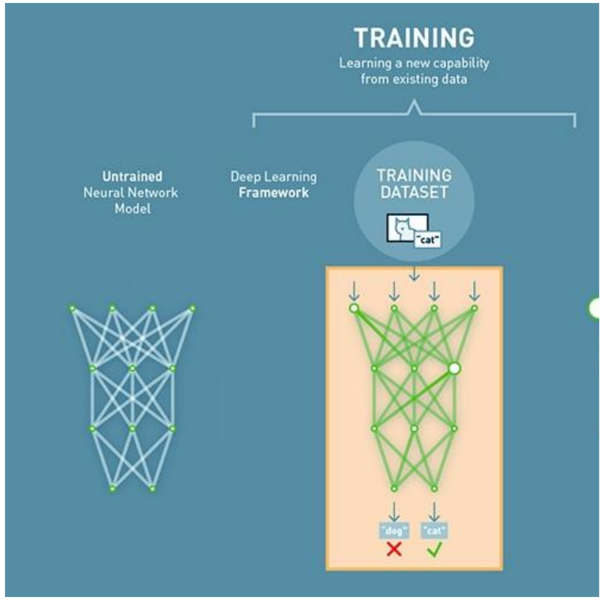
Agenda

- **Introduction**
  - **Inference Landscape**
  - **Challenges**
- Proposed Optimizations
  - Single-GPU inference-optimized transformer kernels
  - Many-GPU dense transformer optimizations
  - Massive-scale sparse (MoE) model optimizations
  - ZeRO-Offload inspired optimizations
- Performance Evaluation
- Conclusion

Introduction

- Deep Learning
  - Training: most state-of-the-art studies and papers –
    - Focused on distributed training on hundreds of GPUs
    - Optimizing for computation and communication efficiency
      - Training time and FLOPS are key metrics
  - Inference
    - CPU based inference is common
    - GPUs – increased adoption
    - Large transformer models especially sparse MoE models can exploit hundreds of GPUs!
    - Latency and throughput are the main metrics

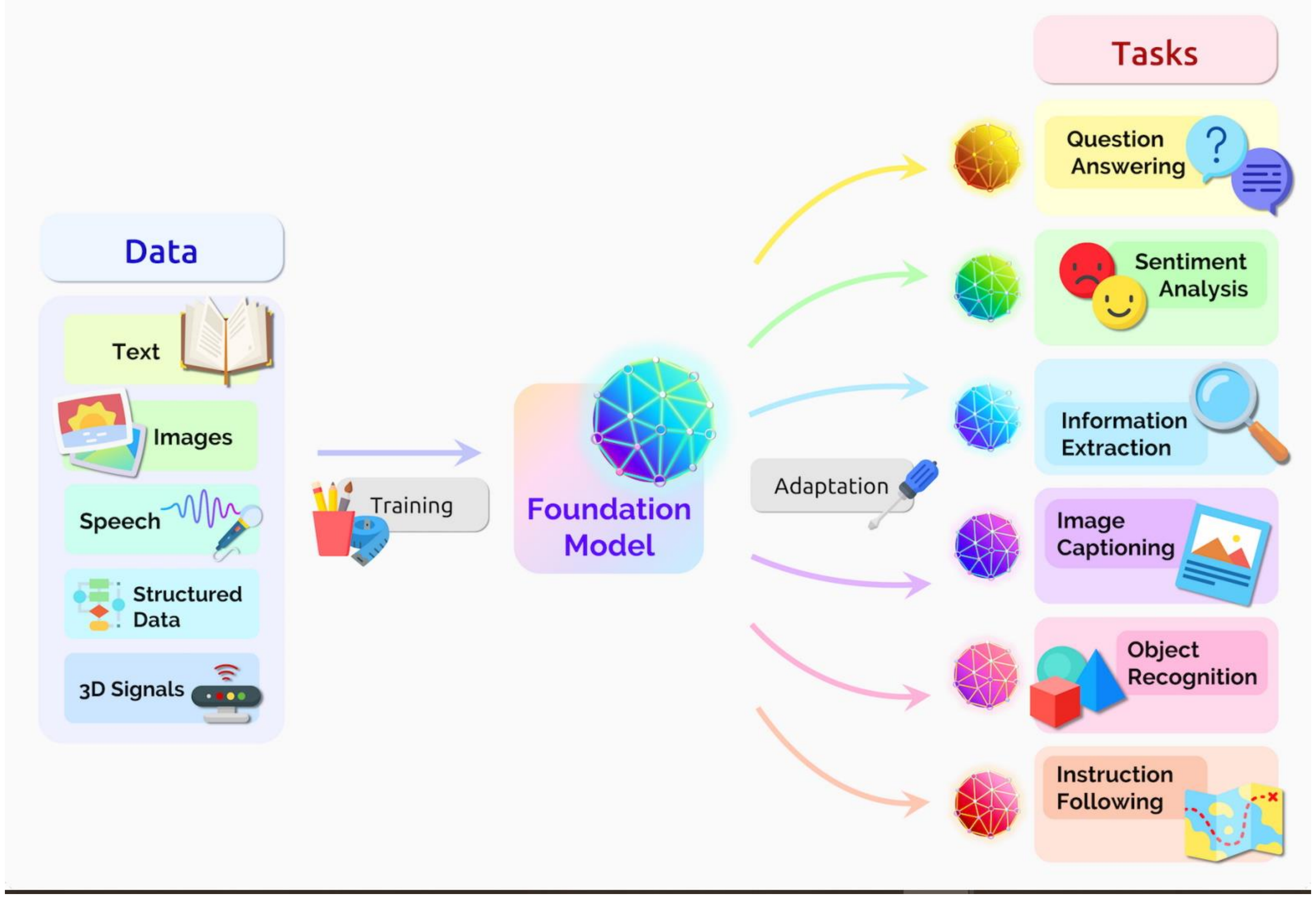




Courtesy: https://www.exxactcorp.com/blog/HPC/discover-the-difference-between-deep-learning-training-and-inference and http://www.zdnet.com/article/caffe2-deep-learning-wide-ambitions-flexibility-scalability-and-advocacy/

Transformer Models

- Transformer models are everywhere
  - Language, Vision, Speech, multi-modal, etc.
  - And they are becoming bigger and better!
  - Being re-branded as *“foundation models”*
- Training large transformer models is hard
  - *Deploying (inference) them is even more challenging!*



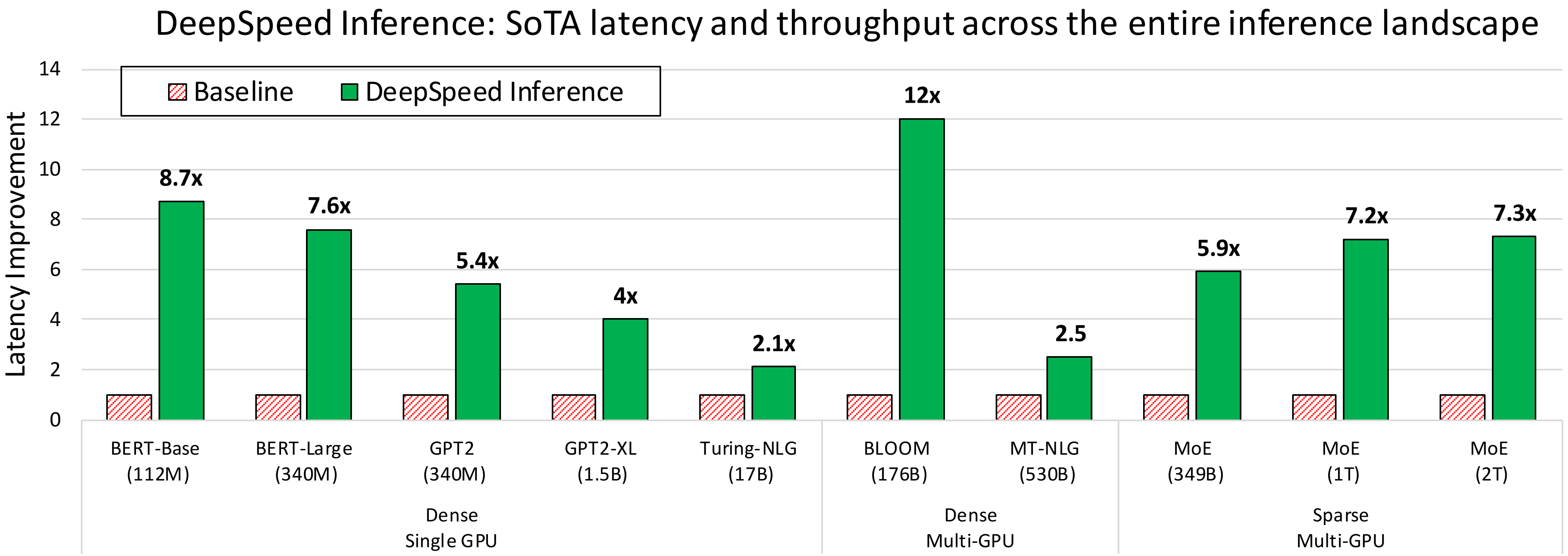
https://blogs.nvidia.com/blog/2022/03/25/what-is-a-transformer-model/

Inference Landscape

- Diverse Inference Landscape
  - ***Model Size***: millions to trillions of parameters
  - ***Architecture***: Dense vs Sparse
  - ***Heterogeneous Hardware***: Single-GPU, Single-node, and Multi-node
  - ***Batch Size***: few (*latency sensitive*) to thousands (*throughput sensitive*)
- Challenges
  - Single Solution cannot be efficient across the entire landscape
  - Hardware accessibility limitations for large model inference

BERT-Base (112M)	BERT-Large (340M)	GPT2 (340M)	GPT2-XL (1.5B)	Turing-NLG (17B)	BLOOM (176B)	MT-NLG (530B)	MoE (349B)	MoE (1T)	MoE (2T)

DeepSpeed Inference



A systematic composition of diverse set of optimizations

- ❑ Inference Optimized transformer kernels – achieve best single GPU performance
- ❑ Many-GPU Dense transformer optimizations – *powering large and very large models like Megatron-Turing 530B*
- ❑ Massive Scale Sparse Model Inference– *a trillion parameter MoE model inference under 25ms*

Democratizing Massive Model Inference

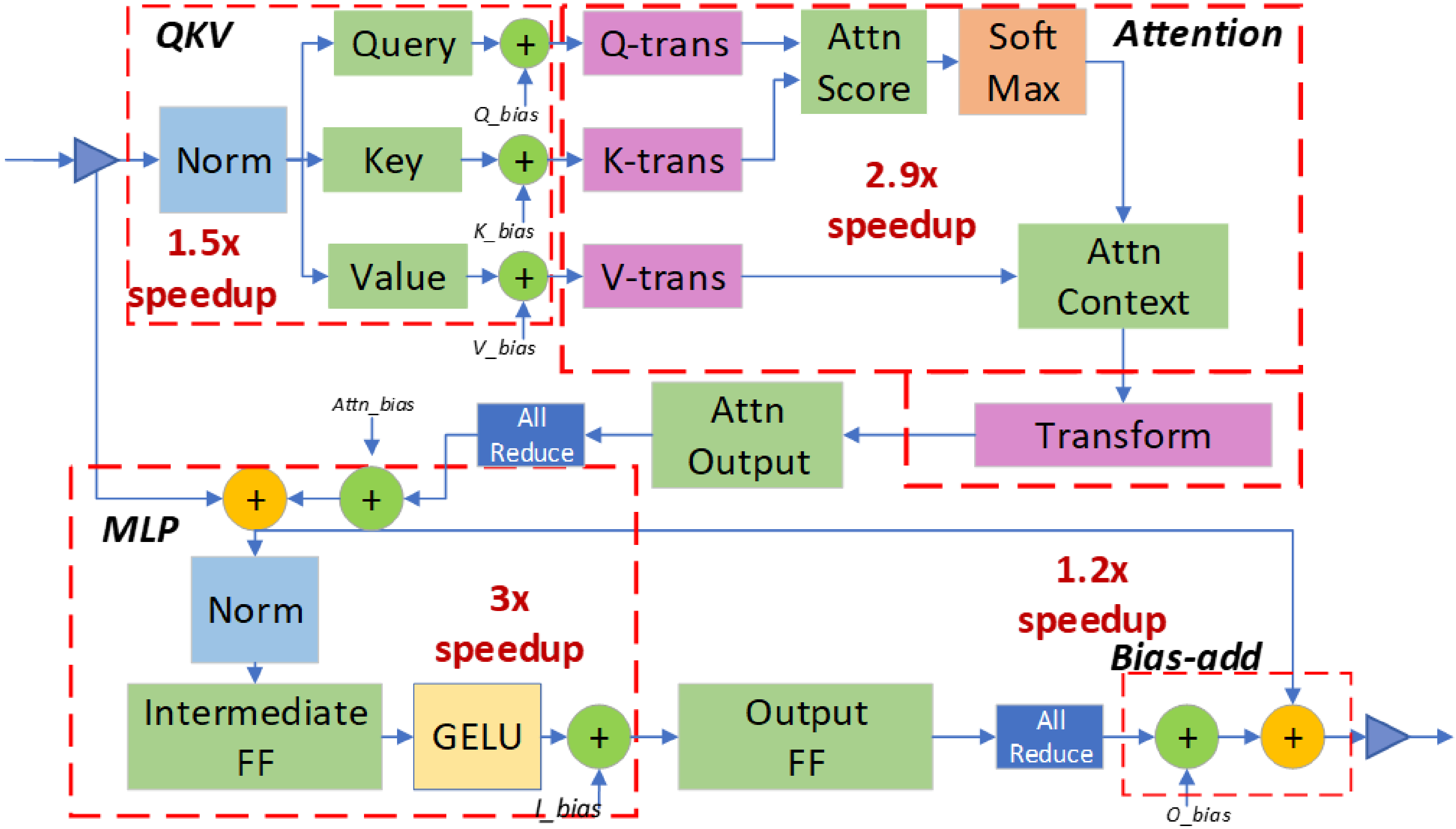
- ❑ ZeRO-Inference – *50x bigger model inference on single-GPU device*

Agenda

- Introduction
  - Inference Landscape
  - Challenges
- **Proposed Optimizations**
  - **Single-GPU inference-optimized transformer kernels**
  - **Many-GPU dense transformer optimizations**
  - **Massive-scale sparse (MoE) model optimizations**
  - **ZeRO-Offload inspired optimizations**
- Performance Evaluation
- Conclusion

Single GPU transformer kernels

- DeepFusion
  - Fuse all the kernels we can
- Optimized GEMM kernels
- Kernels injected to models at runtime
  - Easy-to-use
  - No model code change
- End to end improvement for transformer models



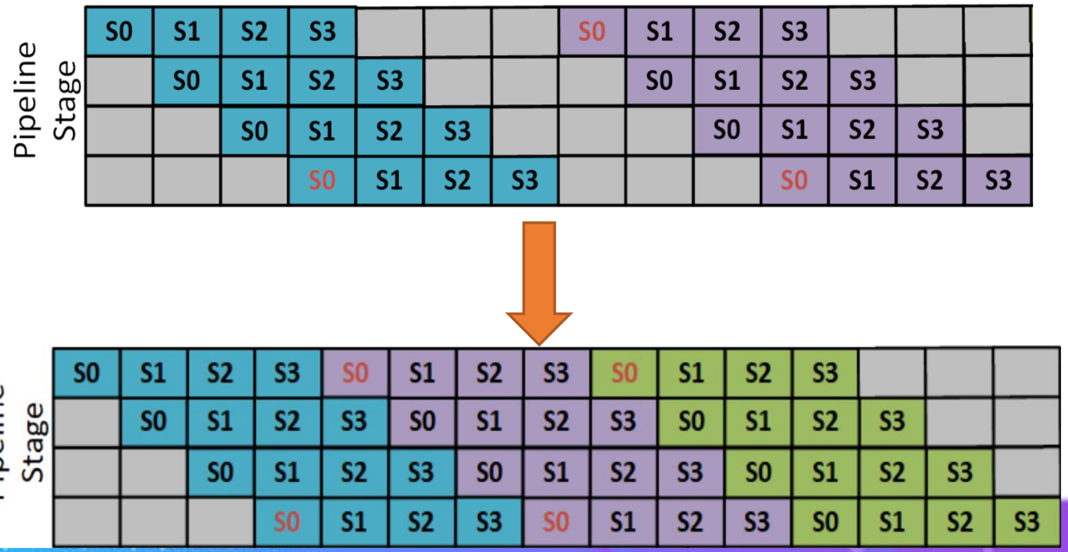
Many GPU Dense Optimizations

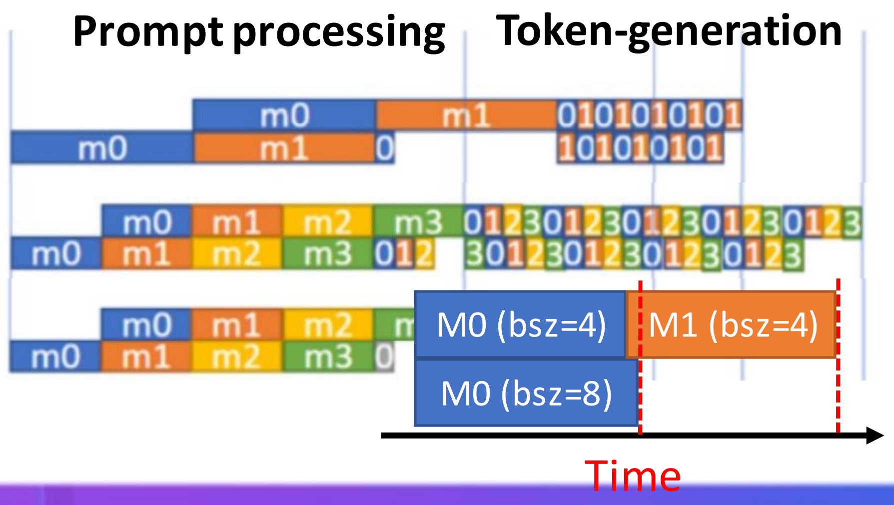
- **Challenges of supporting a massive model for inference**
  - Tensor-slicing: can't scale across nodes
  - Pipeline parallelism can scale but has its own challenge
    - Dependency between current and next token generation
    - Load imbalance between prompt processing and token generation
  - Memory and communication overheads

- **Inference optimized pipeline parallelism in DeepSpeed**
  1. Efficient Pipeline schedule to handle token dependencies
  2. Hybrid Scheduling to address load balancing
  3. Memory and communication optimizations

Efficient Pipeline Scheduling

- 1. Adapt training pipeline-schedule to handle token-generation
  - Inference pipeline load imbalance
    - Prompt: process a batch of input (large tokens)
    - Token-generation: generating one token per batch
- 2. Hybrid Schedule for higher inference throughput





3. Memory and communication optimizations

- High memory usage to handle large batch inference
  - Saving Key/Value for all the pipeline stages (in-flight requests)
- Use CPU memory to offload a part of KV-cache
  - Double-buffer memory to write the current data while prefetching the next
  - Offload policy: to the point that we saturate PCIE bandwidth
  - Increase batch size by ~30%, resulting in 25% speedup
- Communication optimization
  - Handle the dynamic data-transfer at inference time
    - We use some meta-data to handle the serialization-deserialization issue
  - Customized implementation to eliminate the GPU-to-CPU traffic

Primary Challenge with MoE Inference

- Inference latency lower-bounded by parameter load time
  - Model-Size / Achievable Memory Bandwidth
  - Ex. 200B model on a single V100@900GB/s takes 444ms
- Tensor-Parallelism to achieve lowest latency
  - Higher aggregate bandwidth
  - Ex. 200B model on 16x V100@500GB/s takes 50ms
- Tensor-Parallelism is limited
  - Fine-grained Parallelism -> hard to achieve good bandwidth per device
  - Communication volume overhead -> does not decrease with more devices
- **4x larger MoE model size than quality-equivalent-dense**
  - Requires 4x higher bandwidth/parallelism/scalability for latency parity

Designing a highly scalable MoE Inference System

- Goal:
  - Scale beyond tensor-slicing
  - Achieve aggregate memory bandwidth across **<u>hundreds of devices</u>**

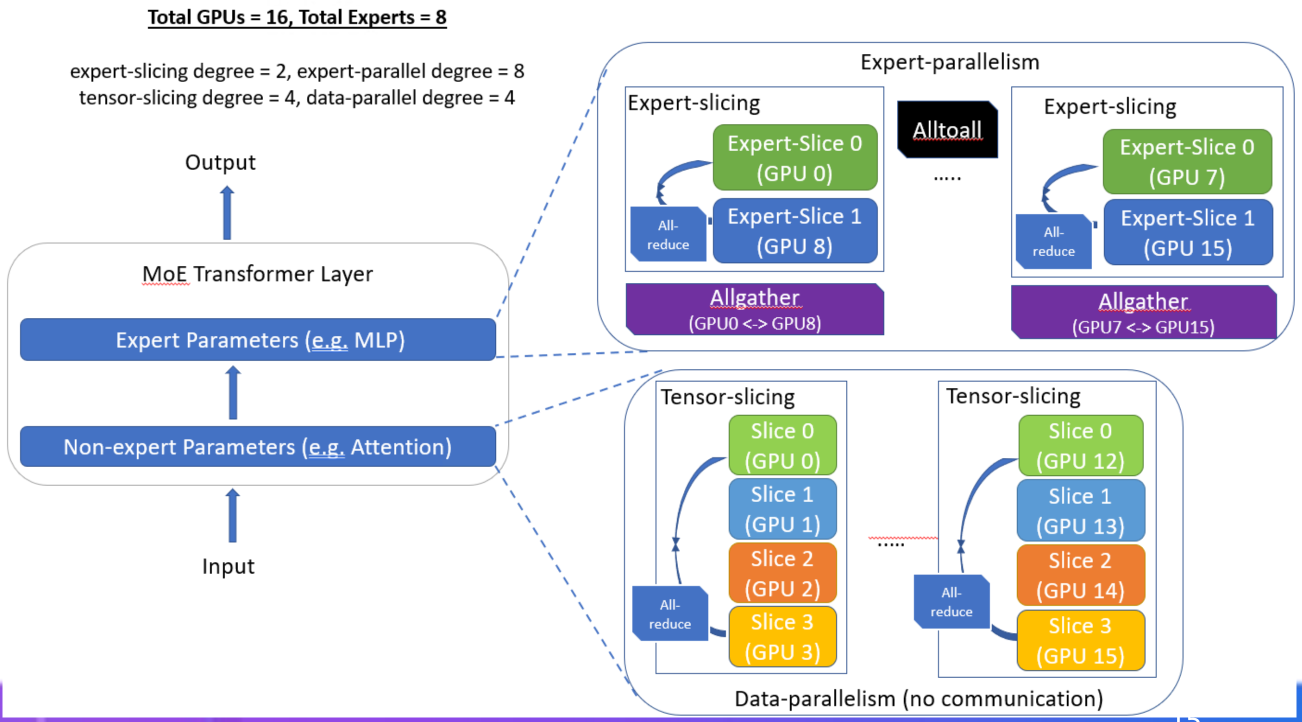
- Three main area of optimizations for maximizing aggregate bandwidth
  - A symphony of parallelism
    - Careful orchestration of tensor, data and expert parallelism
  - Parallelism coordinated Communication Optimization Strategies
    - Minimize communication overhead
  - Kernel Optimizations
    - Maximize bandwidth utilization per device

A symphony of Parallelism

- Observation
  - Each token is processed by at most a single expert
  - Each token can be processed independently of the other

Tensor-Parallelism	Expert-Parallelism
Fine-grained Parallelism	Coarse-grained parallelism
Comm Vol: O(batch)	Comm Vol: O(batch/devices)

- Expert Parallelism
  - Group tokens based on experts
  - Run experts in parallel
  - Coordinate with all-to-all communication



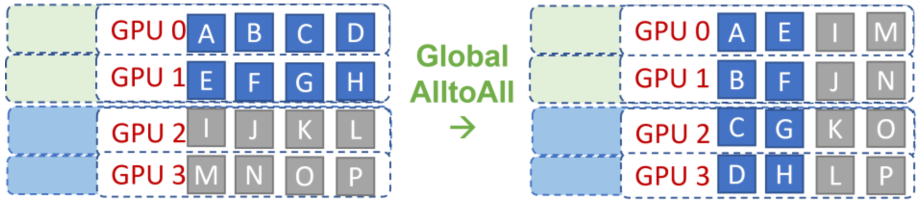
- Tensor Parallelism:
  - Tensor-slicing for non-expert parameters
  - Expert-Slicing for expert parameters

- Data Parallelism:
  - Scale non-expert parameters to match expert parallelism

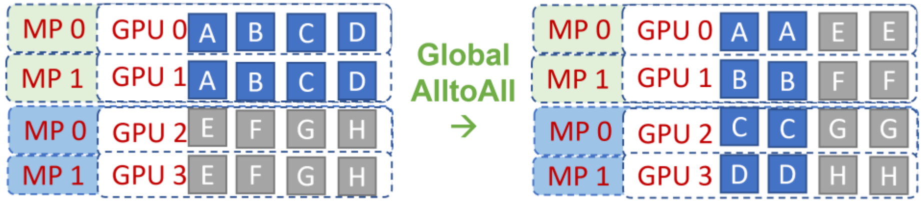
Communication Optimizations

- Communications:
  - All-to-all, all-gather, all-reduce
- Communication optimizations
  - NUMA aware, SCCL
  - Hierarchical, parallelism-coordinated, all-gather based, data mapping strategies
- All-to-All latency
  - Increases linearly with devices
  - Massive overhead at hundred gpu scale
- Parallelism-Coordinated All-to-All
  - Leverage redundancy in data
  - Reduce critical communication path
  - O(gpus) -> O(gpus/tensor-slicing)

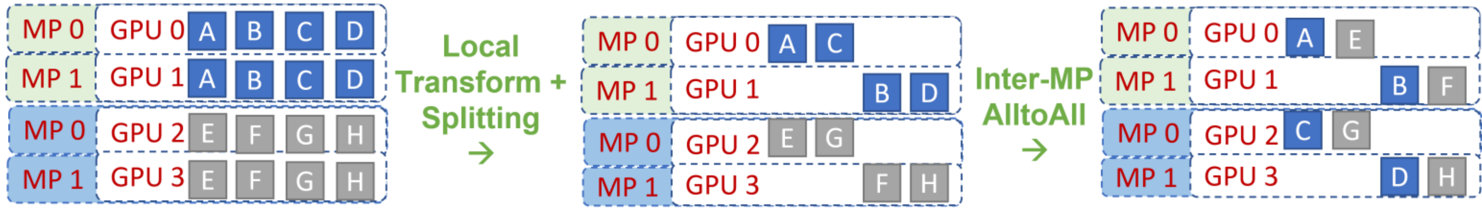
Baseline AlltoAll:



Baseline All-to-All with Tensor-Slicing:



Parallelism-coordinated AlltoAll optimization

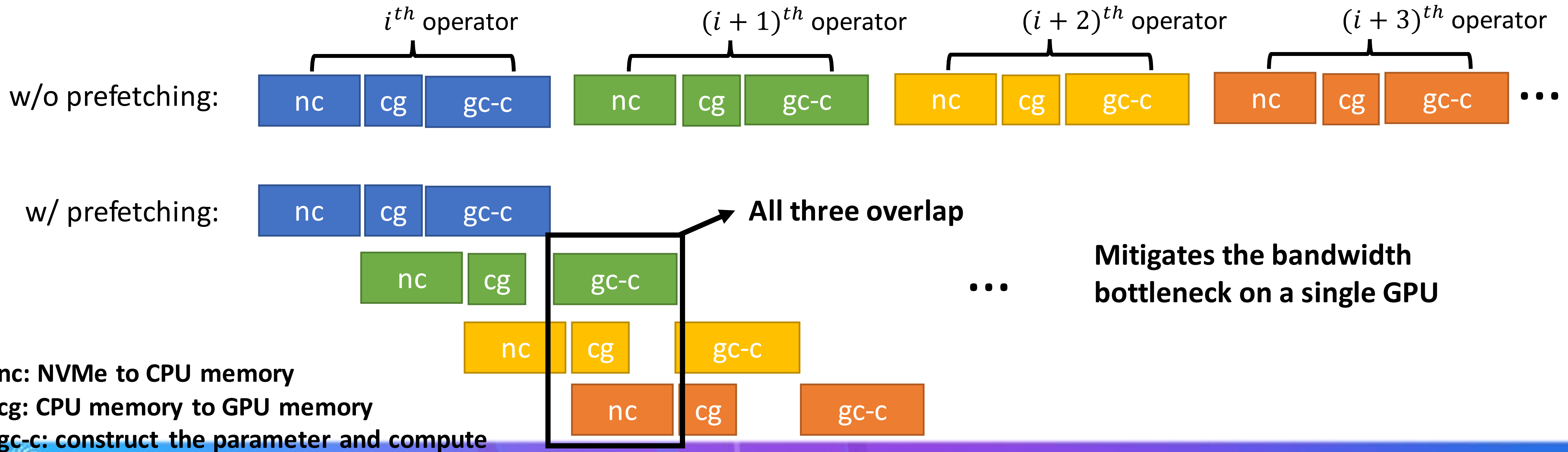


Democratizing Massive Model Inference

- Many model scientists only have access to one or a few GPUs
- ZeRO-Inference utilizes heterogeneous memory (GPU/CPU/NVMe) to fit massive models
- Built on top of ZeRO-Infinity and optimized for inference
  - DeepNVMe, a powerful C++ NVMe read/write library
    - Supports bulk asynchronous data transfer
    - Achieves near the peak bandwidth
  - Pinned Memory Manager, manages the limited supply of pinned memory
    - Reuses a small amount (tens of GBs) for offloading the entire model states (up to tens of TBs) to CPU or NVMe
  - Dynamic Prefetching (detailed in next slide)

Dynamic Prefetching

- Trace the operator sequence ahead of time
- At runtime prefetch parameters needed for future operators while computing the current one



Mitigates the bandwidth bottleneck on a single GPU

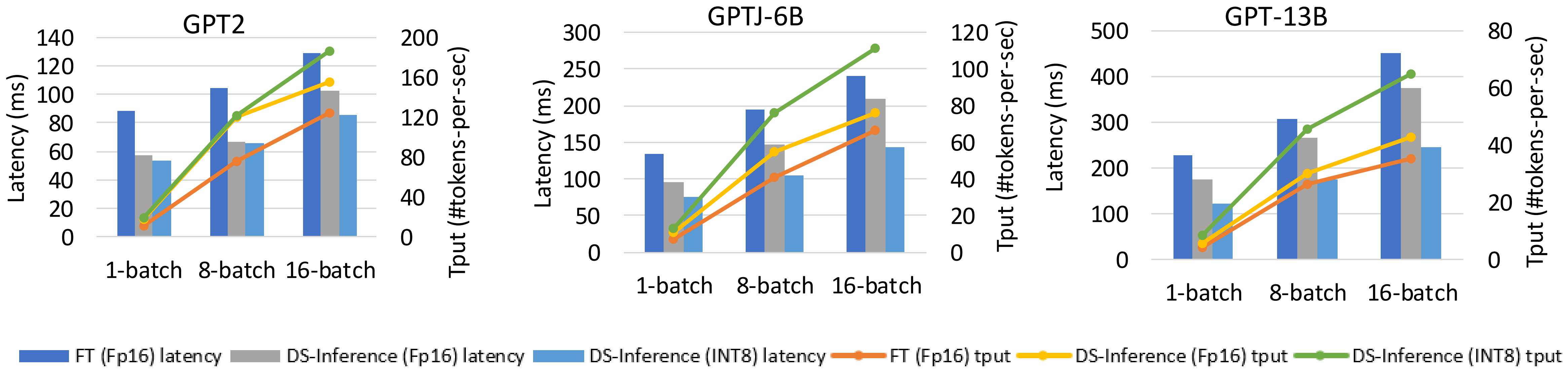
nc: NVMe to CPU memory
cg: CPU memory to GPU memory
gc-c: construct the parameter and compute

Agenda

- Introduction
  - Inference Landscape
  - Challenges
- Proposed Optimizations
  - Single-GPU inference-optimized transformer kernels
  - Many-GPU dense transformer optimizations
  - Massive-scale sparse (MoE) model optimizations
  - ZeRO-Offload inspired optimizations
- **Performance Evaluation**
- Conclusion

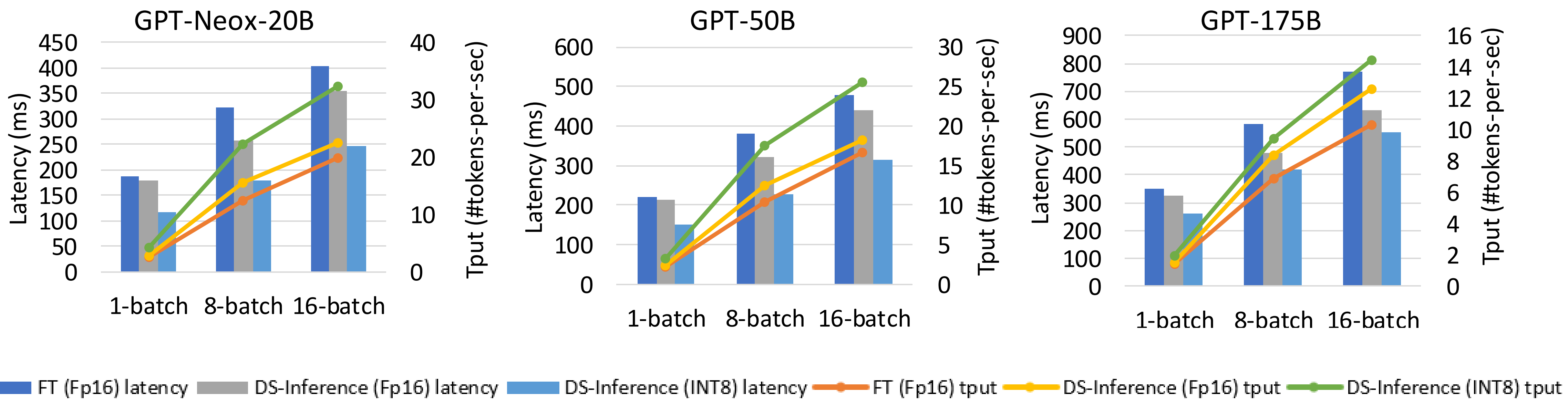
DS-Inference: Up to 2x better single-GPU performance

- Low latency and high throughput for various model sizes
  - GPT2-345M to GPT-13B



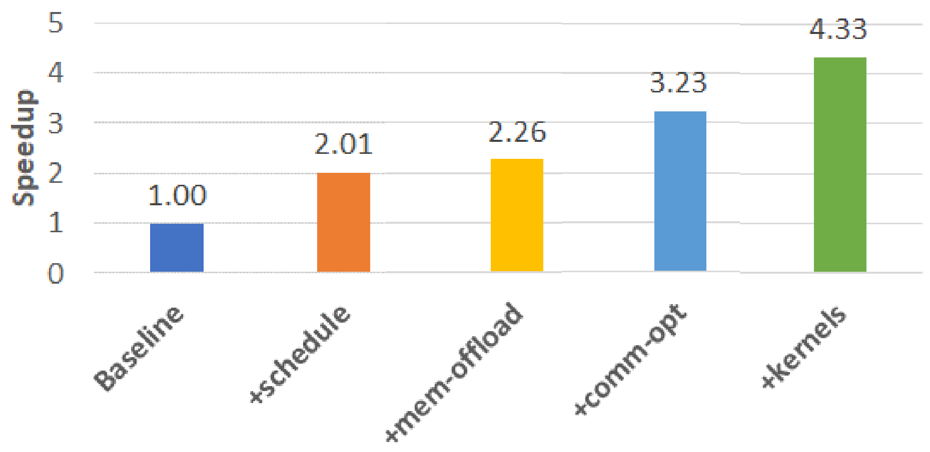
DS-Inference: Up to 1.5x better multi-GPU performance

- Low latency and high throughput for models larger than a GPU memory
  - GPT-NeoX 20B → GPT3-like 175B model



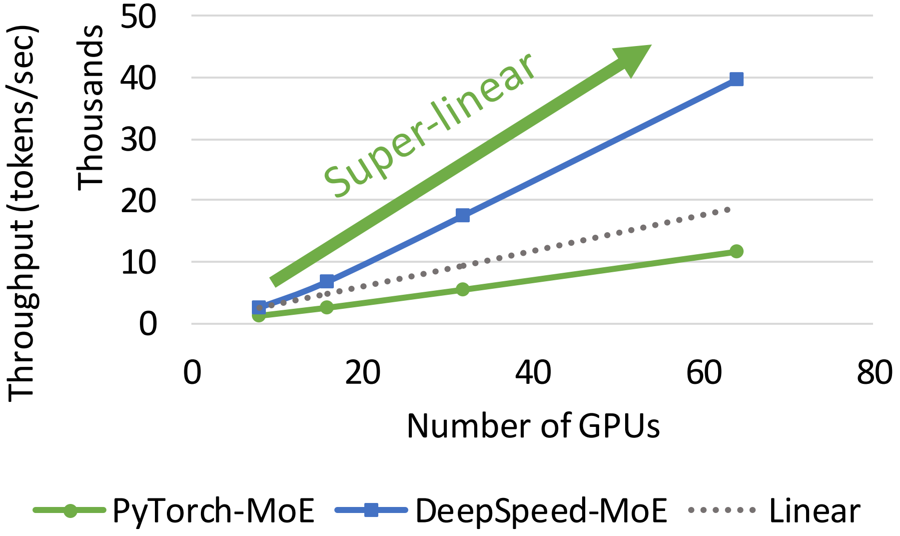
Symphony of all optimizations: Deploying Megatron-Turing-NLG 530B

- **Inference optimized pipeline parallelism in DeepSpeed**
  - Pipeline Schedule
    - Efficient schedule to handle token dependencies
    - Hybrid Scheduling to address prompt/token load-imbalance
  - Memory-offload: utilize CPU for large-batch inference
  - Communication optimizations
  - High-performance Transformer kernels

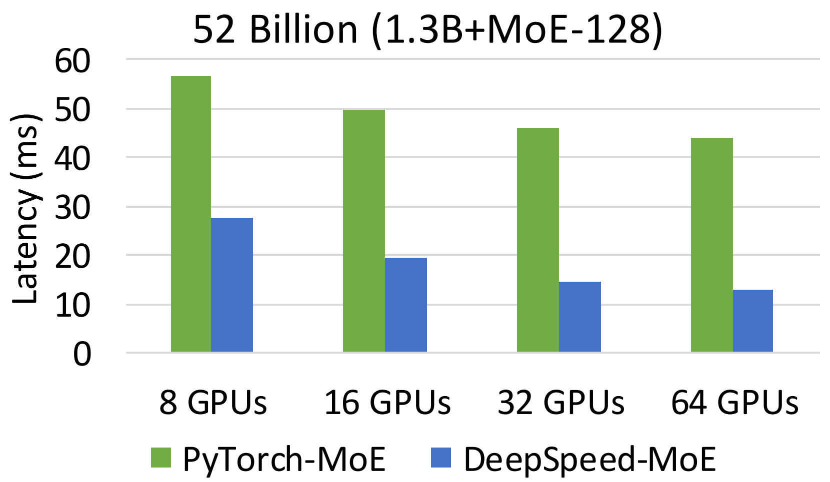


Unique properties of MoE inference

- Super-linear increase in throughput
  - Exploiting aggregate memory bandwidth across all GPUs
- For dense models, the best-case is linear

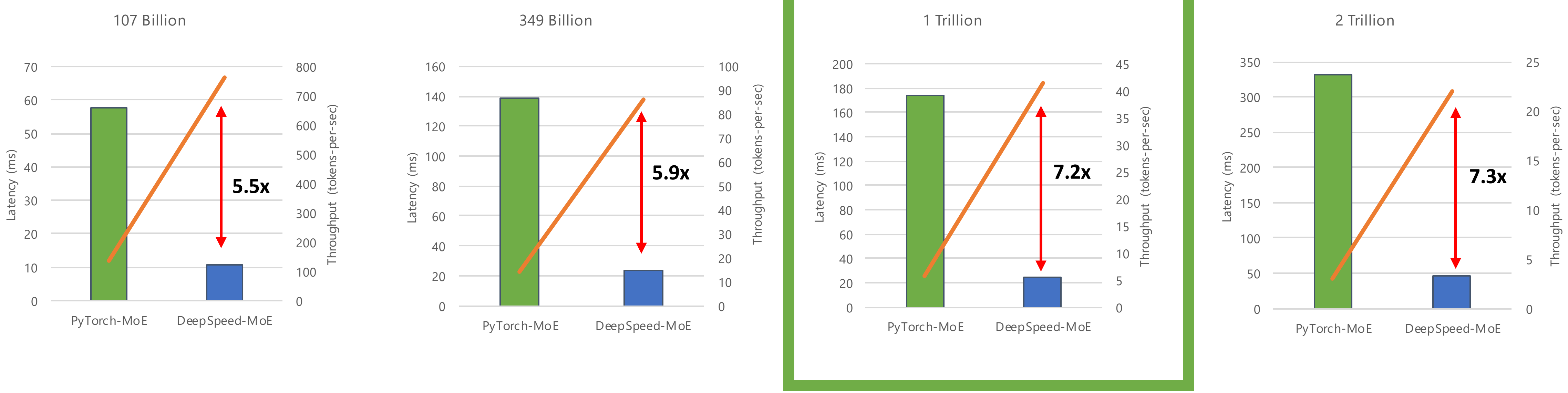


- Latency reduction with more GPUs
- DeepSpeed-MoE: *Achieve low-latency along with the super-linear throughput increase!*



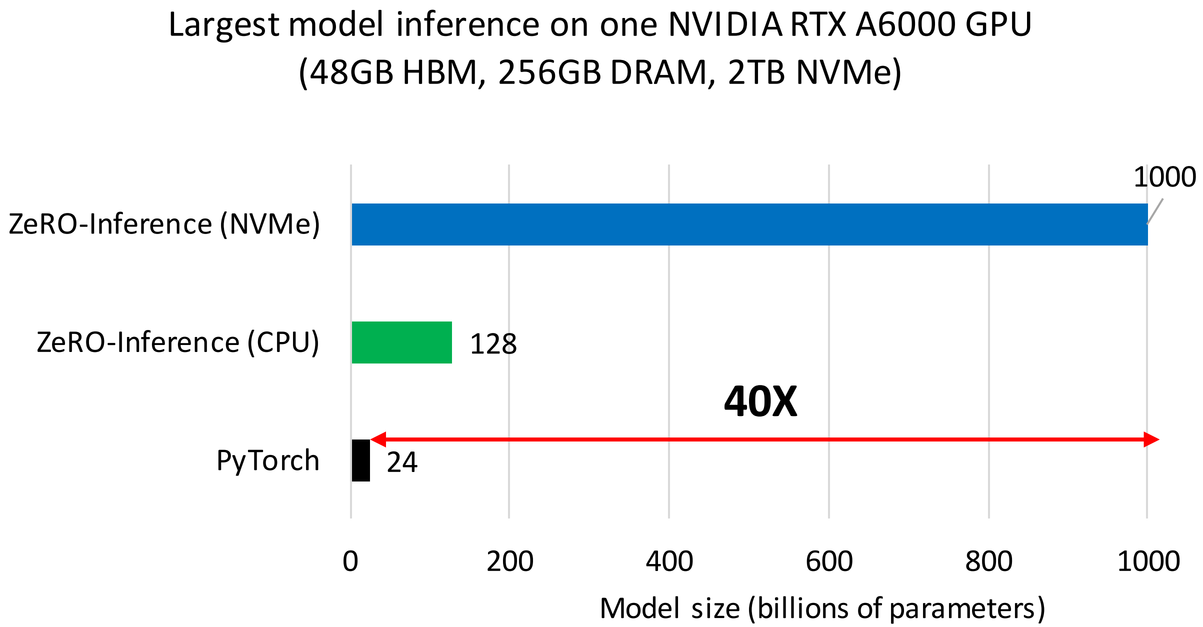
Sparse MoE model optimizations

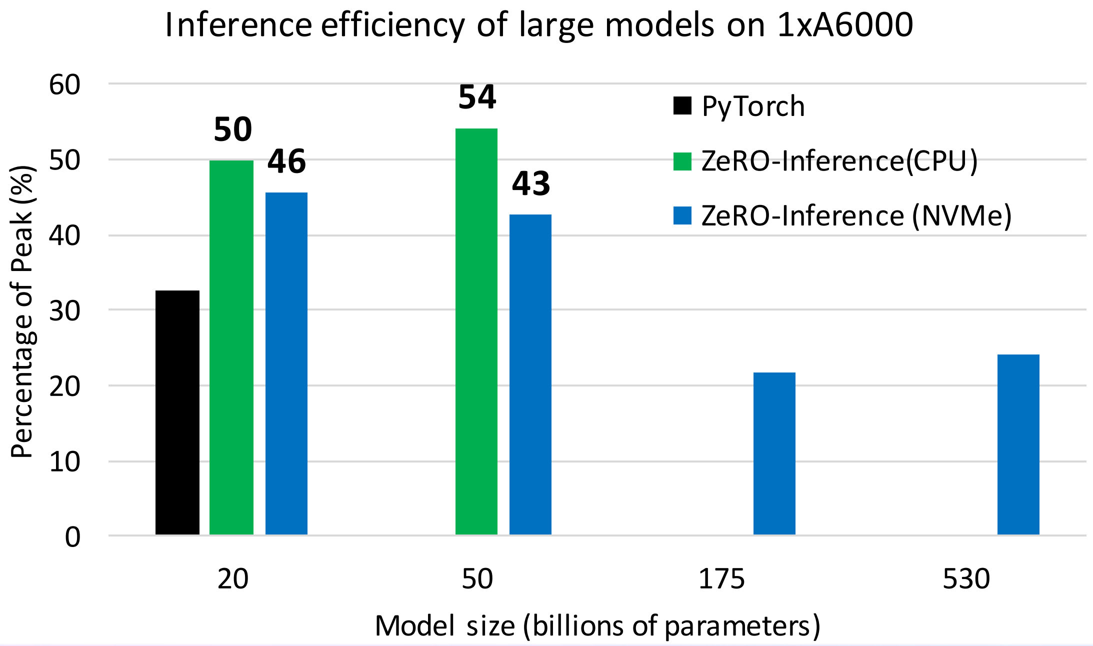
- 7.3x Lower-latency & Higher-throughput at Unprecedented Scale
- 25ms for serving a 1T model
  - 50ms for fastest 200B dense model



Democratizing Inference on a single A6000 GPU

- Large model scale: > 40x bigger model inference on single-GPU
- High efficiency:
  - > 50% hardware peak throughput
  - Better than GPU-only inference due to larger batch size





Agenda

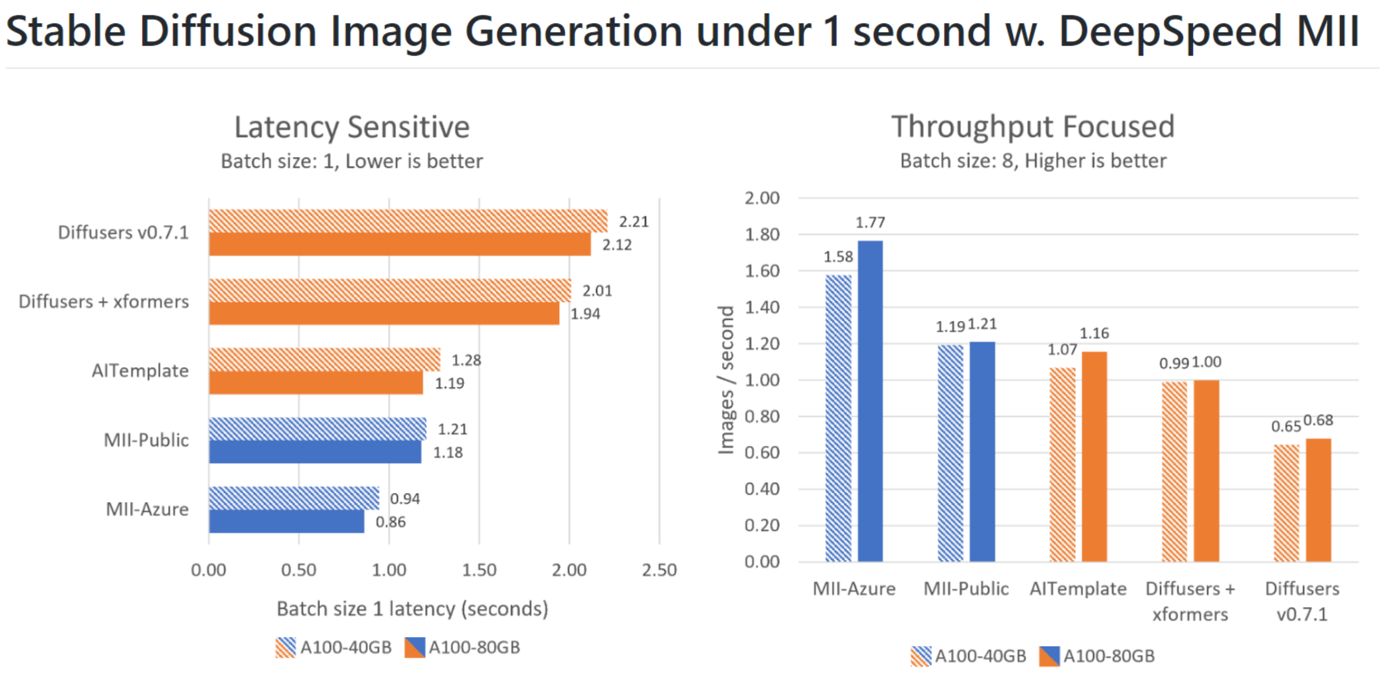
- Introduction
  - Inference Landscape
  - Challenges
- Proposed Optimizations
  - Single-GPU inference-optimized transformer kernels
  - Many-GPU dense transformer optimizations
  - Massive-scale sparse (MoE) model optimizations
  - ZeRO-Offload inspired optimizations
- Performance Evaluation
- **Conclusion**

Conclusion and Future work

- DeepSpeed inference is addressing the diverse landscape of
  - Model sizes
  - Model architectures
  - Platform scale
- A coordinated set of optimizations are needed
  - Single GPU kernels
  - Multi GPU and multi-node inference for dense models requires coordination
  - Multi-node sparse MoE models require a different set of optimizations
  - DeepSpeed-inference combines them in one system
- We are laser focused on fast performance and ease-of-use

So, what’s next?

- Fast moving field; new models everyday
  - Image generation is taking over the fun model experimentation space!
- DeepSpeed-MII: Our latest effort to make DeepSpeed-inference accessible and reproducible
  - *We are enabling the fastest Stable Difusion (under 1 sec.) with MII!*



https://github.com/microsoft/DeepSpeed-MII/tree/main/examples/benchmark/txt2img

Thank You!

- Questions and feedback

Training
- Speed
- Scale
- Cost
- Democratization

Inference
- Latency
- Throughput
- Serving cost
- Ease-of-use

Compression
- Model size
- Latency
- Tuning cost
- Composability

...

Multi-purpose DL optimization suite

DeepSpeed Website: https://www.deepspeed.ai/
DeepSpeed-MII: https://github.com/microsoft/DeepSpeed-MII
DeepSpeed GitHub: https://github.com/microsoft/DeepSpeed

deepspeed