Large Language Models:

The Rise of Data

Introduction

Large Language Models (LLM) have become widely adopted for Natural Language Processing (NLP) applications because of the greater accuracy, broader vocabulary, and richer creativity over more traditional language tools. LLM-powered platforms such as ChatGPT, MidJourney, and Google Bard are demonstrating the potential power of these more complex tools in real-world applications.

These large-scale generative AI models with billions or even trillions of parameters can consistently outperform fine-tuned models given the same number of training steps. Between 2019 and 2023, LLMs have seen a surge in model size, growing by 1000x in just a few years. However, model training and inference at this scale can be challenging, with enormous pressure on memory resources.

In this white paper, we take a platform approach to accelerate LLMs and show how GPU, memory, and storage collectively contribute to overall LLM performance. We also look at how LLMs can be made more efficient by optimizing resource usage and how system throughput is the key element to performance for techniques such as offloading GPU models. By comparing different types of model offloading, we show how DDN's A^{3}I parallel storage (AI400X2) can speed up LLM inference throughput:

- x16 faster compared with traditional NFS storage;
- 1.8x faster than the local RAID storage of a typical GPU-accelerated system;
- Matching the performance of offloading to main CPU memory in some scenarios;

We also demonstrate how even larger future models (up to 24 trillion parameters during our testing) could run efficiently by maximizing GPU utilization on just one host system combined with a single DDN AI400X2 flash storage appliance.

This testing aims to understand how to use GPU memory more efficiently for a given capacity and how GPU model offload techniques with shared storage can improve the efficiency and utilization of LLM solutions. An alternate strategy to improve performance could be to deploy additional GPU-enabled host systems and extend the available GPU memory resources.

Optimizing LLM Resource Usage

Several approaches are available to reduce LLM training times, resource usage and address memory limitations to achieve faster deployment. One common method is to fine-tune pre-trained **"Foundation Models"** to handle a broad range of specific use cases.

Foundation Models are already trained at scale with a core dataset, and customizations usually consume a relatively short time – typically, this would involve full-scale training cycles, although on a much smaller dataset.

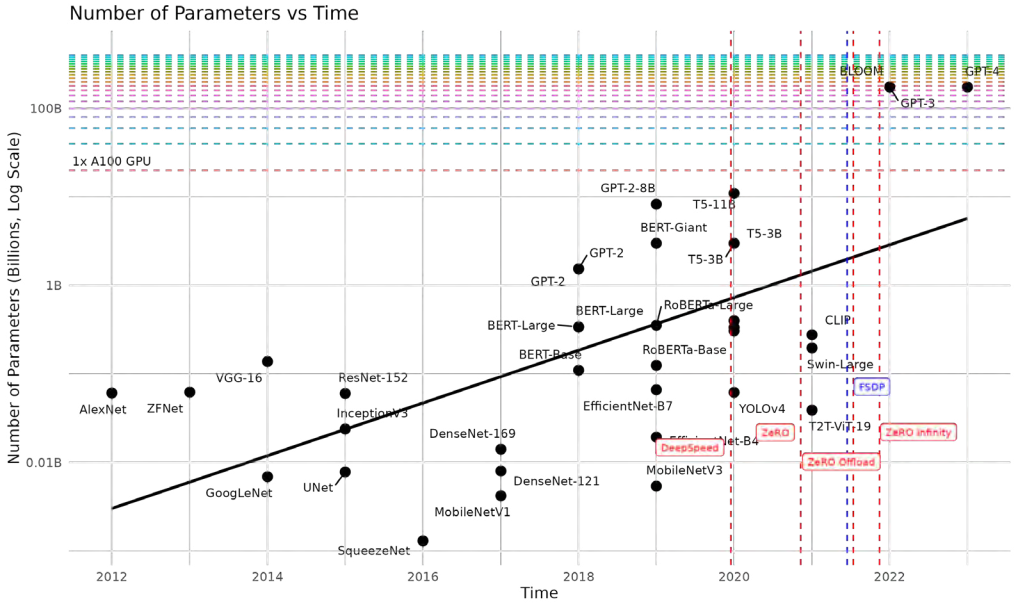
More focused methods exist, such as **"Low-Rank Adaptation (LoRA),"** which freezes most of the model parameters and introduces a smaller block of variable parameters, offering faster model tuning in a compact package. Again, this reduces the resource requirements for customizing pre-trained models since only a fraction of the parameters need training – making them ideal for Edge AI deployment.

As model sizes have increased in recent years, other more aggressive techniques, such as lower-precision parameters and quantization, have been used to further reduce resource requirements, often with compromises of lower precision, lower accuracy, and less richness of generated output. For example, quantization is used to convert a high-precision model parameter into a constrained lower-precision representation, significantly reducing the memory space required, with a trade-off against the accuracy and complexity of the output.

A complementary approach to memory optimization known as **"GPU Model Offload"** is used to overcome these compromises. This technique offloads some parts of the LLM model to other forms of storage, such as CPU memory, local NVME drives, or shared storage, swapping model states in and out of GPUs during the model training cycles. Freeing GPU memory supports larger LLMs, greater batch size, and improves GPU utilization. In addition, it also allows GPU resources to be concentrated on GPU-intensive forward passes (for inference) and backward propagation (for training), while other computations, such as parameter updates, are offloaded to the CPU.

Breaking the Memory Wall Without Breaking Performance

Using larger models to solve complex problems has been recognised as a trend for the past 10 years, but the emergence of the transformer architecture in 2017 through the "Attention is all you need" paper has helped accelerate adoption, leading to a 1000x model size increase in just a few years, outpacing the growth in GPU and CPU memory capacity by 3 orders of magnitude.



Number of parameters over time. Vertical lines identify software tools developed to overcome the memory usage.

Horizontal lines represent the number of fp16 which can be stored in a single GPU with 80GB of memory.

While the GPU memory has quickly become a limiting factor, the offloading of LLMs to local storage introduces significant performance challenges. The IO throughput needed to offload from GPU memory can be extremely high, ranging from a couple of GigaBytes/s to offload activation checkpoints, up to TeraBytes/s to offload optimizer states – which can even exceed the throughput and even capacity of most local NVME RAID systems (from ZeRO-Infinity: Breaking the GPU Memory Wall for Extreme Scale Deep Learning). Network attached storage can bring the needed capacity, but they need to deliver extreme performance to satisfy the throughput requirements.

Solution: The AI400X2 Storage Appliance from DDN

While local memory or on-board storage typically offers a latency advantage compared to an external storage system, it does not scale to the capacity or throughput needed to run an LLM efficiently, especially on a single node. By comparison, DDN's AI400X2 system includes the EXAScaler parallel filesystem, pre-configured and optimized for AI/ML workloads, with unmatched performance and capacity. The AI400X2 outperforms the throughput of local storage by a significant margin, and at much greater efficiency of any other network attached storage, achieving performance levels which are aligned with
LLM needs:

A. I/O throughput: EXAScaler can scale out to hundreds of GB/s and even TB/s for both read and write operation

B. Capacity: LLMs need very large capacity storage in a very compact and efficient form factor

C. GPU Integration: Tight technical integration with GPU systems, including multirail parallel networking, hot node client-side caching, and DPU integration to drive data at high throughput into individual GPUs and GPU systems

We have evaluated the inference throughput of LLM models using the ZeRO-Infinity offloading feature from the DeepSpeed library on a single host system. It demonstrates that the DDN AI400X2 can achieve 16x the performance of an NFS system and close to 2x the performance of the local RAID storage. At the same time, it enables larger models to be used, up to 24 trillion parameters on a single node during our testing, representing 24 times the number of parameters of a GPT4 model, exceeding the capacity of the GPU, CPU, and local NVME in the system. This model offloading method is also available for training and fine-tuning, which might also become useful for sparse models, such as Mixture-of-Expert models, that decouple the computation from the number of parameters.

Analysis of LLM Offload Performance

TESTBED CONFIGURATION

As a compute client, we used a typical GPU-accelerated system containing 8x GPUs, each with 40 GB memory, along with 8x HDR200 links for the storage network and 1TiB of RAM. As an example of LLM we use the Open-Source BLOOM model, comparing four different offload strategies:

- Offload to the CPU memory of the GPU host system (1TiB of RAM)
- Offload to the local RAID in the GPU host system (4x 3.84 TB, U.2 NVME drives, 15TB total)
- Offload to a traditional NFS filesystem with all-flash storage (DDN Intelliflash H6202, 12x NVME)
- Offload to an EXAScaler parallel filesystem powered by a single DDN AI400X2 appliance (24x NVME drives with 7.0TiB each, SAMSUNG MZWLR7T6HALA-00007, 8x HDR200 links total)

SOFTWARE STACK

To run the inference tests, we relied on the HuggingFace github repository transformer-bloom-inference along with pytorch version 2.0.0 and DeepSpeed version 0.9.1.

For the storage, we used the EXAScaler parallel filesystem, which is an industrialized version of the open source Lustre filesystem distributed by DDN. It runs a patched version of Lustre with additional features and tuning. The software is flexible and supports an array of performance options, which can be configured for the desired workload. In our experiments we disabled checksums for performance purposes.

The following EXAScaler parameters have been used on the host system.

```
lctl set_param osc.*.max_pages_per_rpc=1M
lctl set_param osc.*.max_rpcs_in_flight=32
lctl set_param osc.*.max_dirty_mb=512
lctl set_param llite.*.max_read_ahead_mb=2048
lctl set_param osc.*.checksums=0
```

The DDN AI400X2 appliance hosts the EXAScaler high-performance parallel filesystem used to serve the host system. We installed the latest GA version of EXAScaler (6.1.0) with default settings for this test.

A successful offloading strategy relies on careful parameterization. The offloading engine has been set to use 1MB transaction.

Analysis of LLM Offload Performance

LARGE LANGUAGE MODELS

We used 4 models during our experiments. The two first models are pre-trained, existing **BLOOM7B1** and **BLOOM176B models**, while **BLOOM-mod-1** and **BLOOM-mod-2** are models based on BLOOM-560m where the number of hidden layers and hidden-size have been extended to reach respectively 1.2 trillion parameters and 24.1 trillion parameters, as these two numbers most closely represent the number of parameters of transformer models. The number of hidden layers and attention heads utilized in both **BLOOM-mod-1** and **BLOOM-mod-2** differ significantly from the expected values for a model with 100 trillion parameters. Future tests could use more realistic models.

MODEL NAME	BLOOM 7B1	BLOOM 176B	BLOOM-mod-1	BLOOM-mod-2
#Hidden Layers	30	70	960	4800
Hidden Size	4096	14336	10240	20480
# Attention Heads	32	112	16	16
Batch Size Used	32	16	8	2
# Parameters	7.1B	176B	1.2T	24.1T
Memory Needed to Fit the Model for Inference (GB)	3.5	350	2300	44000

Analysis of LLM Offload Performance

MEASURING LLM OFFLOAD PERFORMANCE

To test the inference performance, we used a modified version of the transformer-bloom-inference source code from the HuggingFace github repository. This code leverages ZeRO-Infinity from the DeepSpeed library, and has an option to run inference tests both with and without model offloading.

Modifications were made to tune the libaio configuration to increase the queue depth and the number of threads. The commands used to run the different tests are provided below.

```
deepspeed --num_gpus 8 bloom-inference-scripts/bloom-ds-zero-inference.py
--name ${model}
--batch_size ${batchsize}
--benchmark
```

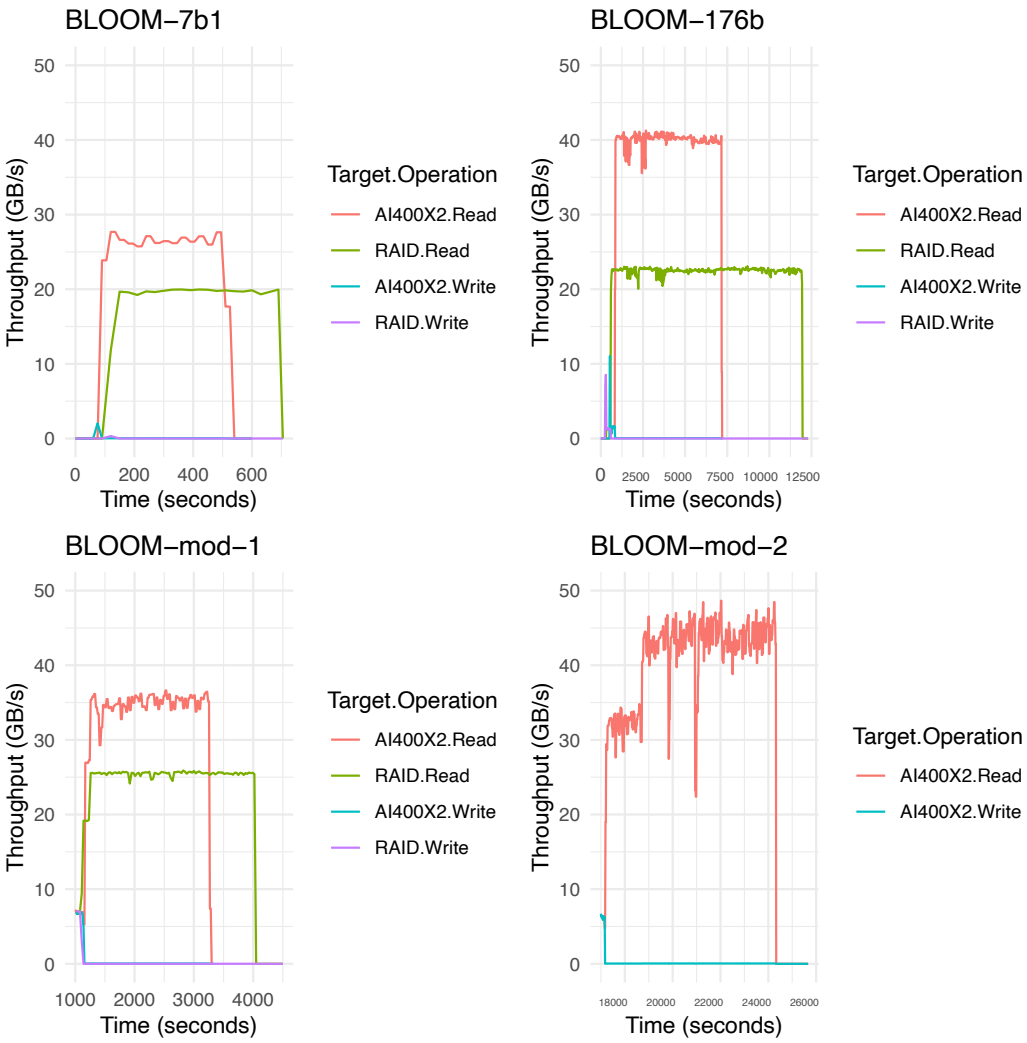
```
deepspeed --num_gpus 8 bloom-inference-scripts/bloom-ds-zero-inference.py
--name ${model}
--batch_size ${batchsize}
--benchmark
--cpu_offload
```

```
deepspeed --num_gpus 8 bloom-inference-scripts/bloom-ds-zero-inference.py
--name ${model}
--batch_size ${batchsize}
--benchmark
--nvme_offload_path=»${filesystemused}»
```

GPU Model Offloading Test Results

THROUGHPUT

The results measured for inference workloads demonstrate that offloading model states to the DDN AI400X2 system outperforms the local RAID storage of the GPU host system for all tests cases, and comparable performance with the offloading to CPU memory for large models (<1%). The key enabler of this offload performance is the high end-to-end throughput of the storage system; the following charts show the results of each offload target for each model (lower times are better). Note that since direct IO is used, the Linux pagecache is not used. This means that the data is transferred between the host and the target even if it would have been able to fit in CPU memory.



GPU Model Offloading Test Results

THROUGHPUT - DETAIL

BLOOM7B1 – With this smallest model size, this LLM model fits entirely in GPU memory for best performance. Offloading to CPU memory, external DDN AI400X2 system, and local RAID take successively longer. Inference times are less than 5ms in this configuration, except for offload to NFS storage, which takes around 10x longer.

BLOOM176B – With 176B parameters, this LLM model is larger than the available GPU memory in a single GPU host system with 8x 40GB in fp16, and so GPU model offloading would provide a way to manage larger models. In this chart, offloading to the external DDN AI400X2 almost matches the performance of offloading to CPU memory, and is again faster than local RAID. As before, offloading to NFS storage is around 10x longer. Inference times are still as low as 50ms in this configuration.

BLOOM_mod_1 – With 1.2T parameters, this model is larger than both the GPU and CPU memory in our single GPU-enabled host system, and so we can only consider offload to storage. In this case, the performance of the DDN AI400X2 is 30% faster even than the local RAID NVMe in the host system – and 16x faster than NFS storage in this test. Inference times are as low as 2 seconds in this configuration.

BLOOM_mod_2 – Finally, with 24T parameters, this model exceeds the available capacity of both the CPU memory and local RAID storage. The DDN AI400X2 system is still able to process this very large model, by offloading from a single GPU system – although inference times are now very high in this limited configuration. Inference times for the NFS storage were measured in days.

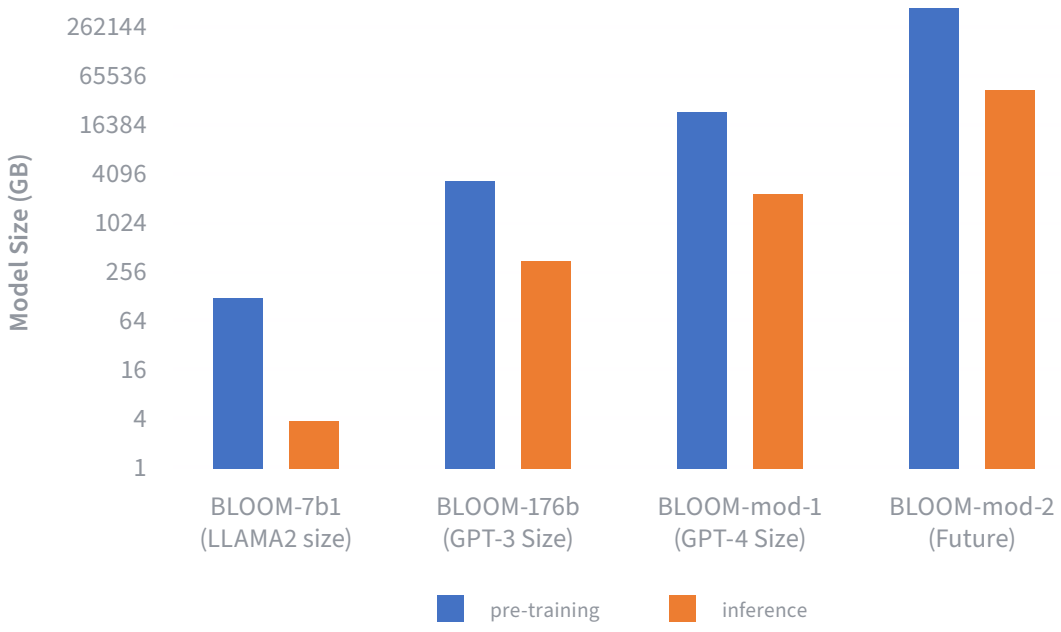
Inference throughput per model and offloading target, lower is better.

GPU Model Offloading Test Results

MODEL SIZE VS MEMORY CAPACITY

We can use the calculation from the ZeRO Infinity paper to extrapolate the memory requirement for the training and fine-tuning using the formula (240 x nl x hd^2), where "nl" is the number of transformer layers, and "hd" is the number of hidden dimensions. We can see that the amount of memory needed to run these LLM is in the range of parallel storage capacity (both with and without using offloading). The DDN AI400X2 system is the only target capable of handling all tested model sizes for inference. The memory requirements would have been even more exaggerated for training and fine-tuning, as the memory usage is much higher for these operations to hold the gradient and optimizer states.

The following chart shows the relative sizes of the LLM models in pre-training (Mp), and for inference (Mi). The memory for inference (Mi) was measured during this test, but the pre-training memory (Mp) is a calculation based on the formula in the ZeRO Infinity paper. The ratio of Mp/Mi varies depending on the type of model, but Mp is always much greater than Mi – in some cases as much as ten times greater.



Conclusion

As the interest in generative AI has caught the public imagination, tools like large language models have become of particular interest as a way to create value and differentiation from their competitors. Organizations are now in the process of making architecture and technology choices that will impact their ability to compete and thrive for years to come.

Historically, computing power has been the focus for conversations on how efficiency could be improved for large-scale AI models. This approach overlooks the potential gains that could be made by applying a system-wide approach and integrating a high performance parallel filesystem to drive increased efficiency while maintaining or even increasing LLM performance.

In this paper we demonstrated how DDN can help customers anticipate the resource requirements and increased model size driven by transformers and NLP. Moreover, we showed how storage performance could become an accelerator for LLM performance as models and evolve.

We recommend that Enterprises take a long term approach to deploying large language models and other AI tools, and that they should consider model growth over the next five years and choose an architectural path that accommodates this future growth with efficiency and simplicity.