The Alan Turing Institute

Turing Technical Report

Benchmarking the performance of GPT-2 type applications on GPU-accelerated computing resources

Tomas Lazauskas and David Llewellyn-Jones

August 2024

Report number 3

`https://doi.org/10.5281/zenodo.13349541`

Abstract

In this report we look at eight GPU-accelerated systems representing a cross-section of the available Tier 2 HPC systems in the UK. These include a mixture of GPU-accelerated platforms from NVIDIA, AMD and Intel. For each system, we perform a set of benchmarking experiments by training a GPT-2 model using a mixture of parameters and hyperparameters. These include model size, number of GPUs, floating point data type, training data size, distribution strategies and batch size. Our interest is in performance measured by time taken to complete an epoch of training rather than convergence speed. We also measure memory usage. The overall aim is to compare systems in order to provide researchers intending to perform AI training with some benchmarks for what to expect in terms of training speed for a model in a non-optimised real-world scenario.

1 Introduction

The growing reliance on deep learning models, notably recently large language models (LLMs) like GPT [1], BERT [2], LLAMA [3], FALCON [4], and others necessitates understanding their performance on diverse compute resources, a requirement especially pertinent to the UK's High Performance Computing (HPC) services.

The UK's HPC services, driven by a tiered approach to HPC provision, offer a variety of compute nodes, primarily within the Tier 2 (regional/specialist hubs) and Tier 3 (local/institutional systems) levels. Given the diversity in computational resources, including different types of accelerators, we have conducted a limited benchmarking study to gather insights into the performance of some of these compute resources.

2 Resources

In the study, we utilised resources available through the UK's HPC services and the ExCALIBUR [5] research program. It is important to note that *these resources have varying characteristics and architectures, making direct comparisons problematic*. Table 1 below lists the systems covered by this study, table 2 the accelerator resources for each system and table 3 their key characteristics.

FP16 (Half Precision), FP32 (Single Precision), FP64 (Double Precision), and BF16 (Brain Floating Point) refer to different floating-point precision formats used

Name	Service	Server	Launch	References
J-V100-32	JADE 2	NVIDIA DGX-1	2017-06	[6, 7]
B-A100-40	Baskerville	Lenovo ThinkSystem SD650-N V2	2020-06	[8, 9]
B-A100-80	Baskerville	Lenovo ThinkSystem SD650-N V2	2021-06	[8, 9]
S-H100-80	Stanage	Dell PowerEdge R7525	2023-03	[10, 11]
C-MI100-32	COSMA8	Dell PowerEdge R7525	2020-11	[12, 11]
C-MI210-64	COSMA8	Dell PowerEdge R7525	2022-03	[12, 11]
D-MX1550-128	Dawn	Dell PowerEdge XE9640	2023-01	[13, 14]
IPU-POD-16	Graphcore	Dell PowerEdge R6525	2021-03	[15, 16]

Table 1: Overview of the systems.

Name	Memory (GB)	Type	Accelerator	Interface
J-V100-32	32	GPU	NVIDIA V100	SXM2
B-A100-40	40	GPU	NVIDIA A100	SXM4
B-A100-80	80	GPU	NVIDIA A100	SXM4
S-H100-80	80	GPU	NVIDIA H100	PCIe 4.0
C-MI100-32	32	GPU	AMD MI100	PCIe 4.0
C-MI210-64	64	GPU	AMD MI210	PCIe 4.0
D-MX1550-128	128	GPU	Intel Data Center Max 1550	PCIe 5.0
IPU-POD-16	14.4	IPU	IPU-M2000	100 GE RoCEv2

Table 2: Summary of accelerator resources.

Name	FP16	BF16	FP32	FP64	References
J-V100-32	31.33	—	15.70	7.80	[17, 18]
B-A100-40	312.00	312.00†	19.50	9.70	[19]
B-A100-80	312.00	312.00†	19.50	9.70	[19]
S-H100-80	1 513.00	1 513.00†	51.00	26.00	[20]
C-MI100-32	184.60	92.30	23.10	11.50	[21]
C-MI210-64	181.00	181.00	22.60	22.60	[22]
D-MX1550-128*	52.43	832.00‡	52.43	52.43	[23, 24]
IPU-POD-16	3 994.00	—	998.00	—	[25, 26]

* GPUs are split into two tiles. The performance shown is per GPU.
† Tensor core performance.
‡ Matrix Engine performance.

Table 3: Floating point performance (in peak TFLOPS).

to represent real numbers in computing. FP16 and BF16 use 16 bits to represent a real number, while FP32 uses 32 bits. The BF16 numerical format consists of 1 sign bit, 8 exponent bits, and 7 fraction bits, while FP16 consists of 1 sign bit, 5 exponent bits, and 10 fraction bits. FP32 consists of 1 sign bit, 8 exponent bits, and 23 fraction bits, and FP64 consists of 1 sign bit, 11 exponent bits, and 52 fraction bits.

BF16 is a truncated version of FP32, offering comparable range and precision to FP32, but with the memory footprint of FP16. Its truncated form means conversion between BF16 and FP32 can be performed with minimal overhead making it attractive for mixed precision calculations. In recent years, BF16 has become the industry standard for machine learning workloads.

BF16 has been supported on AMD GPUs since the MI100, whereas NVIDIA GPUs have supported BF16 since the introduction of the Ampere architecture, such as the A100.

Since the Volta architecture (V100 and newer), NVIDIA GPUs also support Tensor Core [27] technology, which is based on performing matrix operations (multiplication and addition) in parallel and allows for mixed-precision matrix arithmetic (using FP16 for multiplication and FP32 for addition and accumulation). It significantly accelerates computation and helps balance precision and performance.

Alternatively, AMD GPUs, beginning with the MI100, incorporate Matrix Core technology [28, 29], which notably accelerates generalized matrix multiplication computations through Matrix Core Processing Units (MCPUs). These MCPUs are specialized hardware units that conduct matrix operations in parallel and support mixed-precision matrix arithmetic, bearing similarity to NVIDIA's Tensor Cores.

The latest AMD GPUs are known for their strong focus on supporting FP64 calculations, which are important for scientific simulations and high-performance computing tasks. While the latest NVIDIA GPUs can handle FP64 calculations their primary focus is on delivering performance for workloads using lower-precision data types, especially for neural network training and inference.

Like NVIDIA, Intel also seems to be focusing on lower- and mixed-precision data types with their Data Center Max (Ponte Vecchio) series of GPUs. On paper, BF16 calculations run 16 times faster on the Intel Data Center Max 1550 compared to other floating-point data types, thanks to the Matrix Engine technology used to accelerate BF16 matrix operations, including matrix multiplication, addition, and accumulation. This technology supports mixed-precision matrix arithmetic. FP16, FP32, and FP64 calculations, in theory, all run at the same—much lower—speed. However, the real-world performance of Ponte Vecchio is still somewhat of an unknown quantity. A further difference is that each GPU is split into two tiles, which are best treated as two separate entities. Hence, in our experiments on Dawn, a four-GPU node is treated much as an eight-GPU node might be on one of the other HPC platforms, but the performance and timings are reported per GPU (2 tiles).

Graphcore devices also support mixed precision. FP32.32 Accumulating Matrix Product (AMP) operations have FP32 input multiplicands and use FP32 partial sums of products. FP16.32 AMP operations have FP16 input multiplicands and use FP32 partial sums of products. Finally, FP16.16 AMP operations have FP16 multiplicands and use FP16 partial sums of products [30, section 5]. Graphcore is the only device with native support for stochastic rounding, which helps alleviate the precision loss when using FP16.16 AMPs. This involves probabilistically rounding the carry into the result mantissa in proportion to the value of the integer formed by the intermediate bits that must be truncated as a result of the destination format precision [30, section 7].

It's important to note that some of the systems we measured are experimental or in early access and therefore we may not have been able to realise their full compute potential.

3 Models

As an LLM representative, we have chosen GPT2, a transformer-based model, implemented by Andrej Karpathy in PyTorch as a library called minGPT [31]. We have chosen this library as it is a lightweight implementation of GPT2, with a limited number of dependencies, straightforward to install and run, and has a small memory footprint, which allows us to run the model on a variety of compute resources, including those with limited memory. In this work, we have chosen to use lightning-GPT [32], a minimal wrapper around minGPT, enabling the use of PyTorch Lightning [33] to train the models.

We acknowledge that this is not a production-ready implementation of GPT2, and that there are other more performanant and feature-rich implementations, but we believe that for this limited study, lightning-GPT, is sufficient for our purposes.

The model sizes shown in table 4 were used.

Model	Num. of hidden layers	Num. of attention heads	Dimensionality of the embeddings and hidden states	Model parameters (M)	Model size (MB), 16 bit
`gpt2`	12	12	768	85.21	170.51
`gpt2-medium`	24	16	1 024	302.51	605.16
`gpt2-large`	36	20	1 280	708.64	1 417.45
`gpt2-xl`	48	25	1 600	1 475.87	2 951.96
`gpt2-xxl`	60	30	1 920	2 656.08	5 312.43
`gpt2-xxxl`	84	40	2 560	6 609.33	13 219.00

Table 4: Summary of GPT2 model sizes used.

4 Benchmarking

The benchmarking was carried out by training the lightning-GPT models on different systems whilst keeping the software and hardware configuration as similar as possible. Due to the number of benchmarks planned for the study, each experiment was limited to two epochs, and the epoch time and peak memory usage of the second epoch were recorded. This is because the first epoch can be potentially slower than the subsequent epochs, due to the fact that the data is being loaded, and the model is compiled. For many of our experiments training is distributed across multiple processes. Since there may be small differences in how long each process takes to complete, the *average epoch time* for a single epoch refers to the average of the durations across all processes.

The software configuration was fixed across all of the systems at the following:

1. Python 3.9
2. PyTorch 2.0.1a0
3. PyTorch Lightning 1.9.5

On the NVIDIA system we used:

1. CUDA 11.7.0
2. NCCL 2.12.12

On the Intel system we used:

1. Intel Extension for PyTorch 2.0.120+xpu
2. oneCCL Bindings for PyTorch 2.0.200

5 Strategies

For our experiments, we used strategies implemented in PyTorch and PyTorch Lightning [33], in particular DDP [34], DeepSpeed [35, 36], and FSDP [37, 38].

DDP [34], or Distributed Data Parallel, is a strategy that allows for data parallelism. It is the most commonly used strategy for training large models. When using DDP, each device holds a replica of the model, and the gradients are synchronised across all devices during each backward pass. DDP is limited by the amount of memory available on a single accelerator, and therefore, it is not suitable for training very large models. See figure 1 (a).

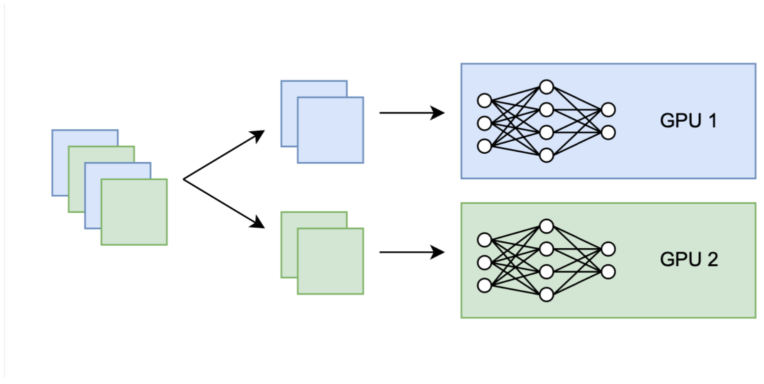

Figure 1 (a): Schematic representation of data parallelism on multiple GPUs.

DeepSpeed [35, 36, 39, 40] is a library that implements several strategies for training large models. One of these strategies is ZeRO [41], which allows for special data parallelisms, or stages. In our experiments, we used three stages of ZeRO: optimizer state partitioning (ZeRO stage 1), gradient partitioning (ZeRO stage 2), and parameter partitioning (ZeRO stage 3). Due to the scope of this work, we did not experiment with the Offload or Activation Checkpointing strategies, which leverage the host CPU to offload optimizer state, gradients, and parameters for additional memory savings, and free activations after the forward pass, respectively. DeepSpeed splits the model horizontally; see figure 1 (b).

FSDP [37, 38], or Fully Sharded Data Parallel, is a type of data parallelism achieved by partitioning (sharding) model parameters, optimiser states and gradi-

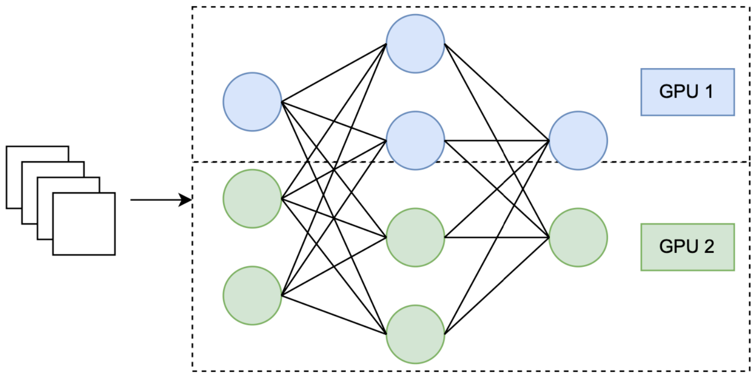

Figure 1 (b): Schematic representation of horizontal model parallelism on multiple GPUs.

ents across DDP rank. FSDP is similar to DeepSpeed's ZeRO stage 3, but it splits the model vertically; see figure 1 (c).

GPUs use SIMD (Single Instruction Multiple Data) to provide data parallelism. Graphcore IPUs are in contrast MIMD (Multiple Instruction Multiple Data) [42, section 3]. Their Tile architecture and Bulk-Synchronous Parallel execution model that splits a task into compute-sync-exchange steps, means that Graphcore IPUs have their own strategies which are separate from those applicable to GPUs. We save the discussion of these strategies to the section on Graphcore devices in the Appendix.

6 Single accelerator comparison

This experiment aimed to compare the performance of different accelerators when training the `gpt2` and `gpt2-medium` models using FP16, BF16, and FP32 precision, with batch sizes of 64 and 128, employing the DDP strategy. The results of the experiments are presented in figure 2.

Observations:

- The `S-H100-80` is the fastest GPU accelerator in terms of theoretical performance and matched the `D-MX1550-128` in terms of actual performance. The theoretical performance doesn't always align with the performance we see in practice. For example, the `B-A100` devices gave better performance

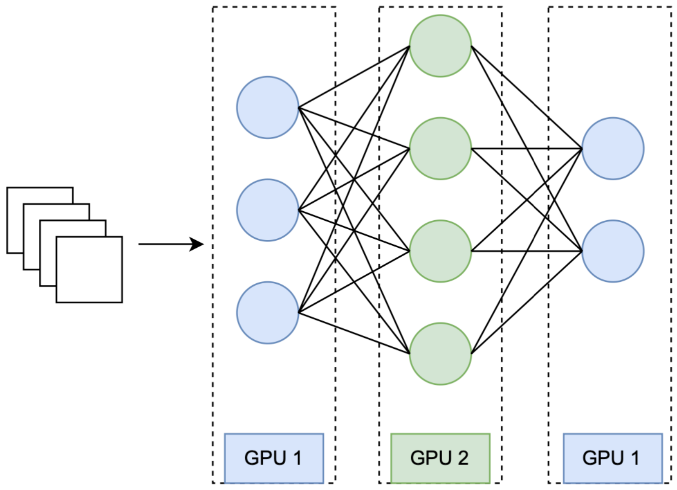

Figure 1 (c): Schematic representation of vertical model parallelism on multiple GPUs.

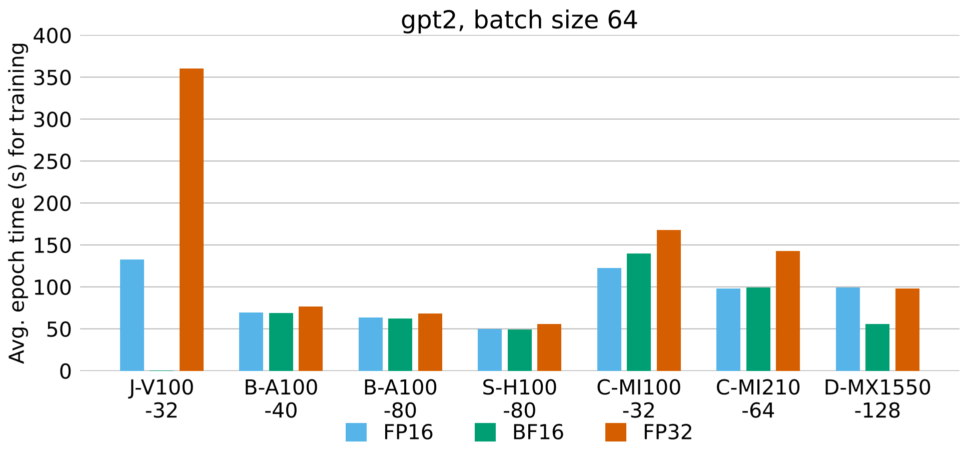

Figure 2 (a): Average epoch training time of the gpt2 model with a batch size of 64.

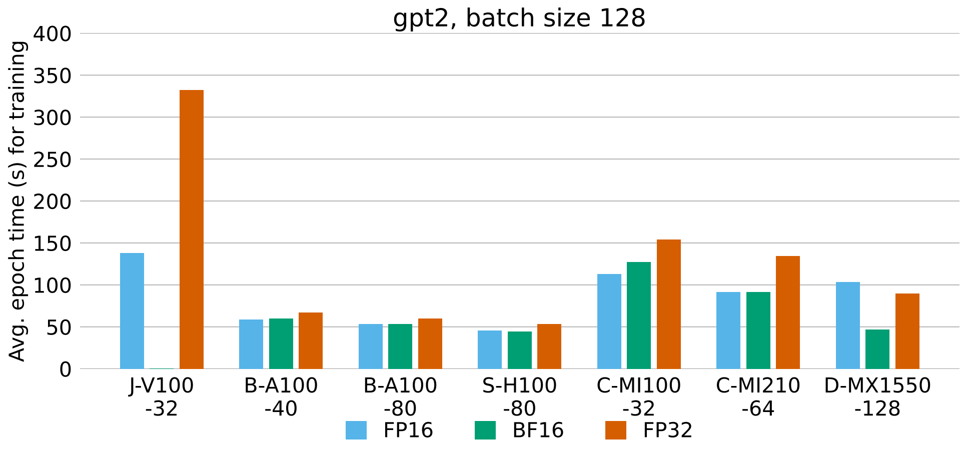

Figure 2 (b): Average epoch training time of the gpt2 model with a batch size of 128.

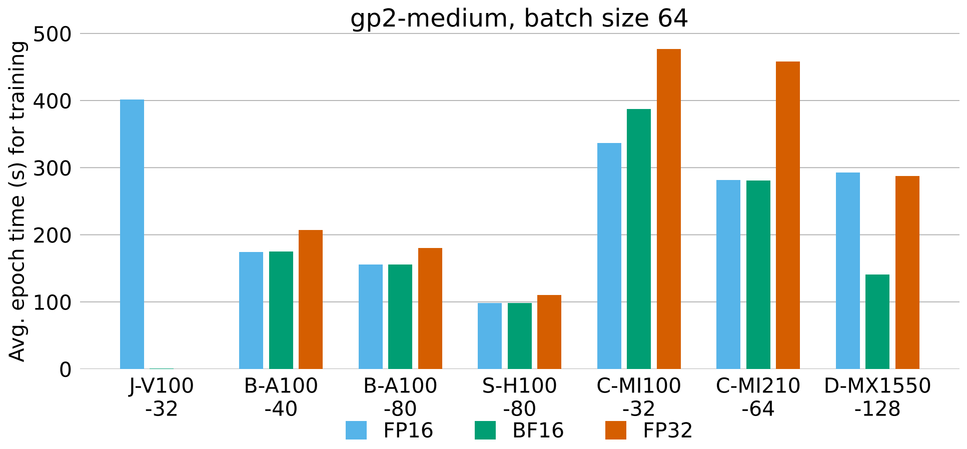

Figure 2 (c): Average epoch training time of the gpt2-medium model with a batch size of 128.

than the C-MI100-32 and C-MI210-64 devices, despite the latter two having higher theoretical FP16 and FP32 peak performance.

- Although the S-H100-80 offers the best performance, the performance gap between it and the B-A100s was not as large as expected. While the peak BF16 performance is 4.8 times higher, the actual performance ranged between 1.26 and 1.89 times faster. This may be attributed to various factors, including suboptimal batch and model sizes for the hardware, code optimization, as well as differences in system architectures.

- Nevertheless, we conclude the differences in performance among various accelerators largely depend on the peak performance of the accelerator and the precision used. The B-A100 has a peak BF16 performance of 312 TFLOPs, which is 3.4 and 1.7 times higher than that of the C-MI100-32 and C-MI210-64 respectively, and 2.5 times higher than the J-V100-32. In our experiments, the B-A100 is 1.5–2.0 times faster than the C-MI100-32 and C-MI210-64, and 2.0–2.5 times faster than the J-V100-32 when training the gpt2 model with using FP16 precision.

- We were impressed by the performance of the D-MX1550-128, which is somewhat comparable to that of the S-H100-80 for BF16 performance. However, performance suffered when using traditional FP16 calculations.

- The reported peak FP16 and FP32 performances (see table 3) do not reflect the performance observed in our experiments. For example, the B-A100 has a peak FP16 performance of 77.97 TFLOPs and a peak FP32 performance of 19.5 TFLOPs, while the C-MI210-64 boasts a peak FP16 performance of 181 TFLOPs and a peak FP32 performance of 22.6 TFLOPs. However, in our experiments, the B-A100 outperformed the C-MI210-64. This is likely due to the fact that the B-A100 utilises Tensor Core [27] technology, which allows for mixed-precision training and, therefore, provides higher throughput for AI workloads. In contrast, both the C-MI100-32 and C-MI210-64 can offer higher peak performances for FP16 and FP32 workloads, potentially making them more suitable for traditional HPC workloads.

- The difference in performance between 16-bit and 32-bit precision, except for J-V100-32, is less significant when training a smaller model. In the case of B-A100, the difference ranges from 8% to 15%, while for C-MI100-32 and C-MI210-64, the difference is between 35% and 48% when training the gpt2 model. However, the difference ranges between 16% and 19%, and 42% and 63%, respectively, when training the gpt2-medium model. This can potentially be attributed to the fact that larger models require more memory, thus increasing memory bandwidth requirements.

- Increasing the model size from gpt2 to gpt2-medium results in a significant increase in training time, between 2.5 and 3.2 times, for all accelerators. This is expected, as the number of parameters increases from 85.21M to 302.51M, approximately 3.5 times.
- Doubling the batch size from 64 to 128 does not significantly improve training time. B-A100 showed a more significant improvement, between 10% and 15%, in reduced average epoch time, compared to C-MI100-32 and C-MI210-64, which showed a 6% to 8% improvement.
- Experiments with the gpt2-medium model, using a batch size of 128, were not able to fit on accelerators with 40 GB of memory or less.

7 Scaling up and out with DDP

While we have demonstrated that the B-A100 and D-MX1550-128 are among the fastest accelerators for the models used in this study, we followed up on the previous experiment by investigating how scaling on multiple GPUs affects the performance of training the gpt2 model. This investigation was conducted using the DDP strategy and executed on both B-A100-40 and D-MX1550-128 using FP16 and BF16 precision, respectively. We varied the number of GPUs and the batch size; see figure 3. Additionally, we investigated how the peak memory usage is affected by the batch size; see figure 4.

Observations:

- The scaling between 1 and 16 GPUs on B-A100-40 is almost linear. We note a 3.5–3.9× speedup when using 4 GPUs compared to 1 GPU, a 7.0–7.6× speedup when using 8 GPUs compared to 1 GPU, and a 12.9–14.1× speedup when using 16 GPUs compared to 1 GPU. This is expected, as the number of GPUs is directly proportional to the number of training steps required to complete an epoch. Therefore, the more GPUs used, the fewer training steps are required to complete an epoch, thereby reducing the training time.

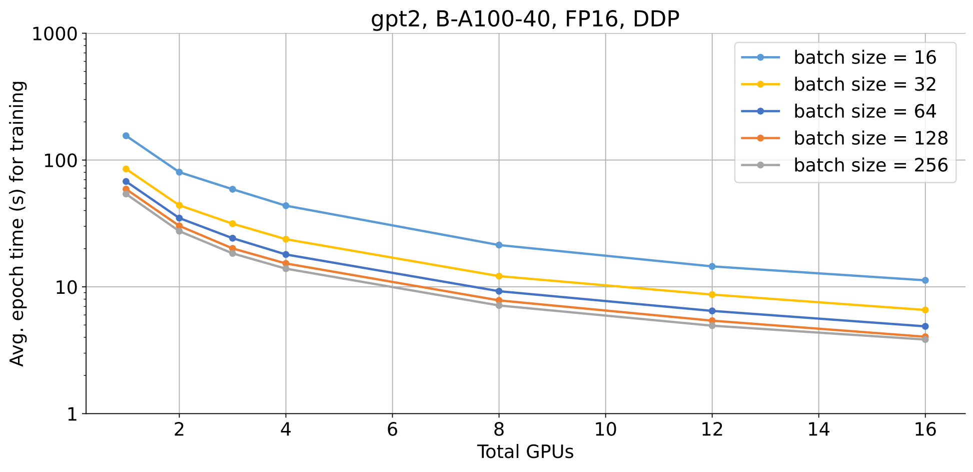

Figure 3 (a): Average epoch training time of the gpt2 model running on B-A100-40, varied by altering the number of GPUs and the batch size.

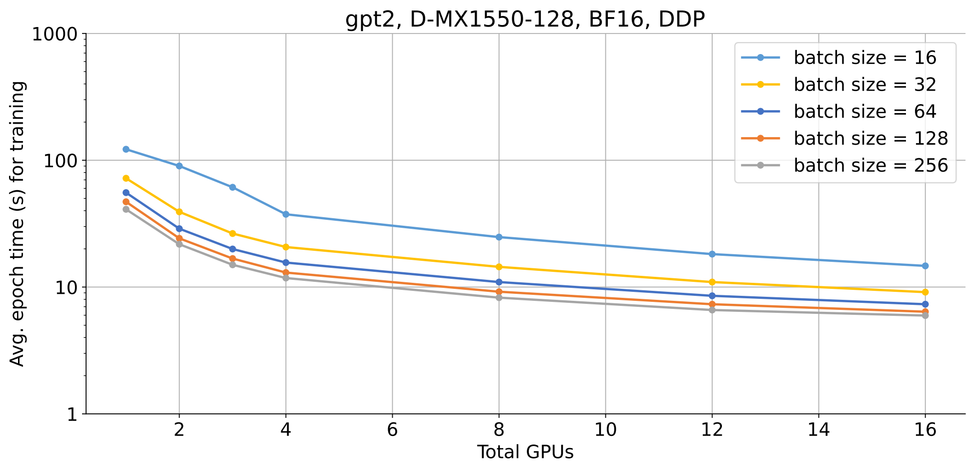

Figure 3 (b): Average epoch training time of the gpt2 model running on D-MX1550-128, varied by altering the number of GPUs and the batch size.

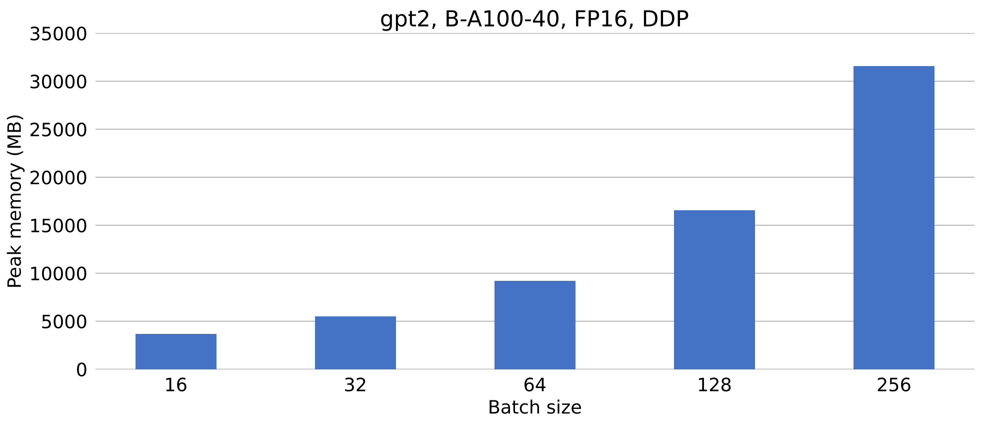

Figure 4 (a): Peak memory usage of the gpt2 model running on B-A100-40, varied by altering the batch size.

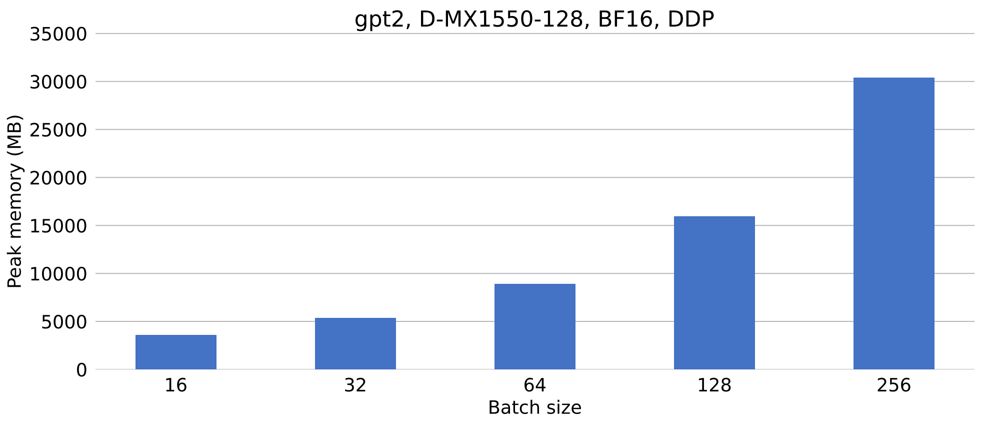

Figure 4 (b): Peak memory usage of the gpt2 model running on D-MX1550-128, varied by altering the batch size.

- When doubling the B-A100-40 batch size from 64 to 128, training time decreases by 15%, and when quadrupling it from 64 to 256, training time reduces by 22%. Conversely, if halving the batch size from 64 to 32, training time increases by 31%. When quartering it from 64 to 16, training time experiences a significant 137% increase. These observations align with the fundamental concept that batch size is directly related to the number of training steps necessary to complete an epoch. Larger batch sizes lead to a reduced number of training steps and subsequently shorter training times, while smaller batch sizes necessitate more steps, resulting in longer training durations.

- Comparing B-A100 with D-MX1550-128 we find the results are remarkably similar. Initially with just a single GPU D-MX1550-128 outperforms B-A100 irrespective of batch size. As the number of GPUs and nodes increases, the gap narrows until at 8 GPUs the B-A100 starts to outperform the D-MX1550-128 and then remains ahead. This may be due to increased communication due to the tiled nature of the Ponte Vecchio GPUs, meaning that we were executing twice as many processes compared to CUDA on the B-A100.

- When the model size is fixed, the limiting factor enhancing performance is the batch size, which is, in turn, limited by the memory available on the accelerator.

- Doubling the batch size increases the peak memory usage by a factor of 1.5.

- We also note that peak memory usage did not significantly change when the number of GPUs varied between 1 to 16.

We also looked into determining the largest model size that can potentially be trained using FP16 precision and the DDP strategy. We chose B-A100-80 as it has the largest amount of memory available at 80 GB (while the D-MX1550-128 has more memory on paper, this is split across the two tiles when using DDP). We also set the batch size to 1 to allow for the largest possible model size to be trained on a single accelerator. The results of the experiment are summarised in table 5.

Observations:

- The DDP strategy allows only for data parallelism; therefore, the model size is limited by the amount of memory available on a single accelerator.

Model	Peak memory (MB)	Avg. epoch time (s)
gpt2	2 298.80	2 555.80
gpt2-medium	8 093.53	4 067.32
gpt2-large	19 424.99	4 951.83
gpt2-xl	32 586.84	6 289.65
gpt2-xxl	57 239.73	8 128.52
gpt2-xxxl	Out of Memory	-

Table 5: Summary of the largest FP16 model size that can be trained on a single `B-A100-80` accelerator using the DDP strategy.

- The largest model size that we were able to train on a single `B-A100-80` accelerator using DDP is `gpt2-xxl`, which has 2.7B parameters and requires 57.2 GB of memory.
- To train larger models, model or pipeline parallelism is required as it enables the splitting of the model across multiple accelerators, thereby reducing the memory requirements on a single accelerator and allowing for larger models to be trained.

7.1 Larger Datasets

The results presented so far have all relied on a small amount of input training data: the classic shakespeare_text.txt file [43]. This comprises the works of Shakespeare containing 4.4 MiB of data, which in ML training terms is very small. The entire dataset can easily be stored in main or GPU memory, reducing the impact of data transfer speeds on our benchmarking results.

We wanted to compare results with a larger, but otherwise similar dataset. We used data from The Pile dataset [44] processed to maintain otherwise similar characteristics to the Shakespeare dataset used for all other experiments. This processing involved projecting the UTF-8 text down to an 8-bit ASCII format. The resulting 361 GiB of data was truncated to the first 4.5 GiB to avoid excessive runtimes. This is still small by AI training dataset standards, capable of being stored entirely on the GPU. Nevertheless, it is orders of magnitude larger than our small dataset.

Apart from the basic size, the other main difference between the Shakespeare data loader and the Pile data loader was that the former stores the data in CPU memory, whereas the latter spools the data from disk as needed. For the larger dataset, we copy the files to temporary scratch storage in order to maximise throughput.

In figure 5 we show the results of experiments varying the quantity of training data and the number of GPUs. In all runs we used DDP strategy, FP16 precision, model size of 85.3 million parameters and running on `B-A100-40`.

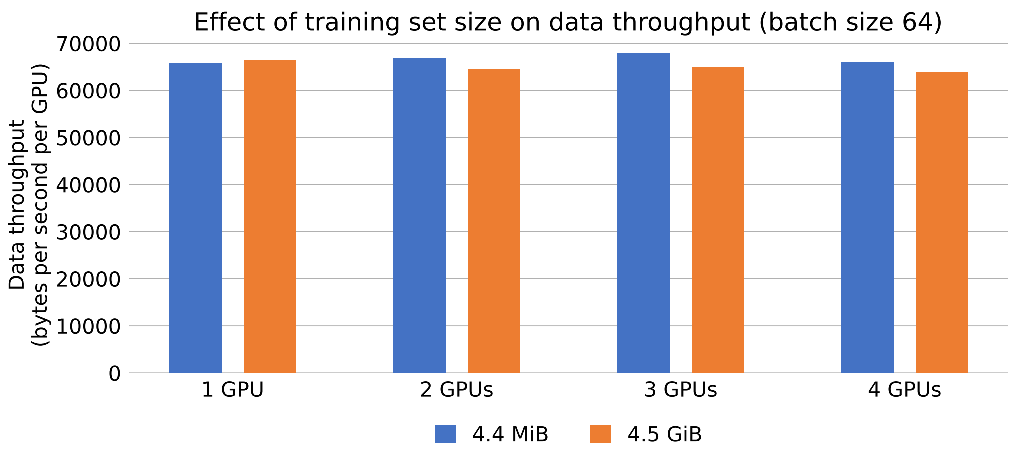

Figure 5 (a): Data throughput using a batch size of 64.

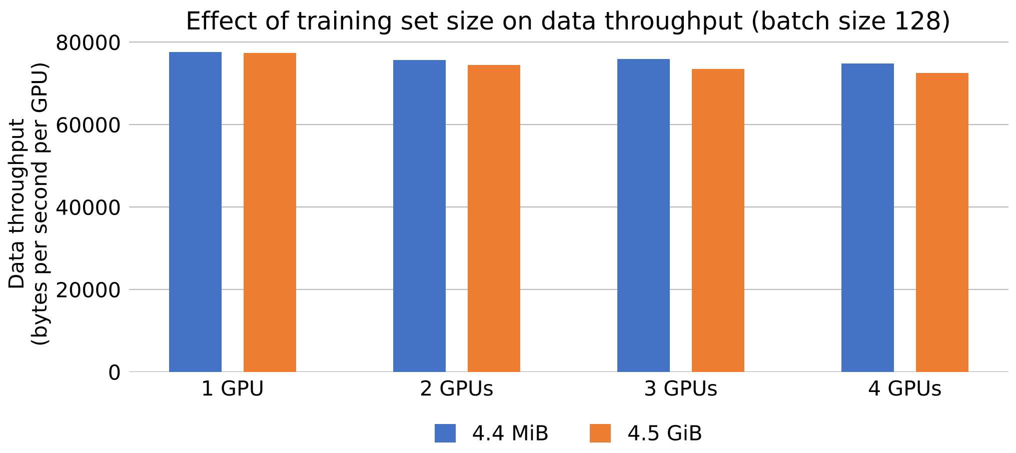

Figure 5 (b): Data throughput using a batch size of 128.

Observations:

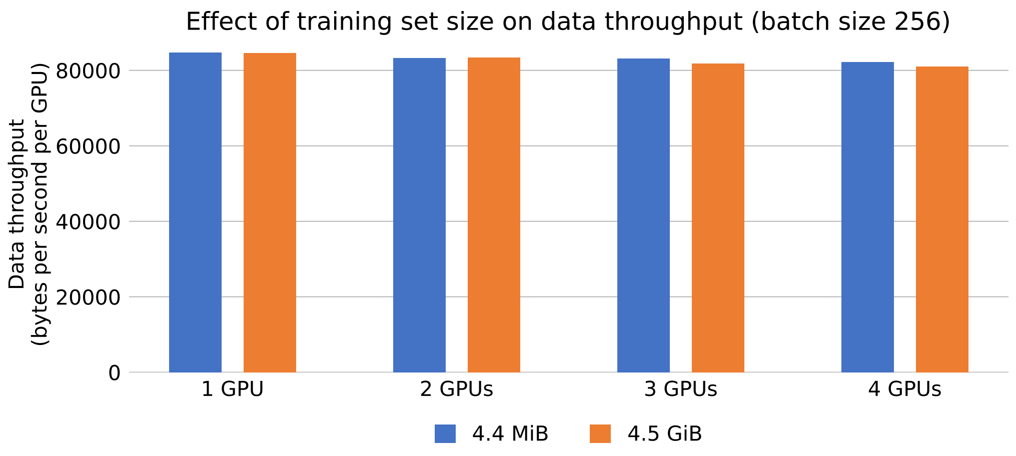

Figure 5 (c): Data throughput using a batch size of 256.

- Although we do see a drop in throughput for the larger dataset, the impact is small, much smaller than we were expecting in fact. In practice, we see average epoch time scaling only slightly more than linearly with training dataset size.
- Batch size has a larger impact on data throughput than dataset size. Increasing the batch size allowed us to get greater throughput with the larger dataset than we were able to achieve with a smaller batch size using the smaller dataset. In practice, larger batch sizes are a good way to increase training speed where GPU memory allows for it.
- In all cases we see throughput reducing slightly as the number of GPUs increases. This is to be expected since using DDP the larger number of GPUs will increase data transfer requirements during the back-propagation stage.
- We see broadly the same levels of PCI data sent and received for both the small and large training datasets. Average data transfer drops slightly as the batch size increases. For the small training dataset and 1 GPU, we see an average transfer of 33.5 MB/s, 27.8 MB/s and 22.9 MB/s for batch sizes of 64, 128 and 256 respectively. For the larger dataset, the equivalent values are 32.6 MB/s, 27.0 MB/s and 22.06 MB/s. As soon as we introduce more GPUs, data transfer unsurprisingly increases by multiple orders of magnitude (ranging between 1458 MB/s and 3027 MB/s).

- The rule of thumb for training data is that training time will increase linearly with training data size.

7.2 Model Parallelism

One of the reasons we chose lightning-GPT [32] for our experiments is that it has already integrated DeepSpeed and FSDP strategies, which allow for model parallelism. The following experiments were carried out with the intention of better understanding the performance of these strategies when training different sizes of the gpt2 models, as well as to determine the largest model size that can be trained using these strategies.

Experiments using the DeepSpeed and FSDP strategies were carried out using their default settings, FP16 precision, a batch size of 16, and by varying the model size and the number of GPUs. `B-A100-40` was also chosen due to the availability of the resource when the experiments were carried out. The results are summarised in figure 6.

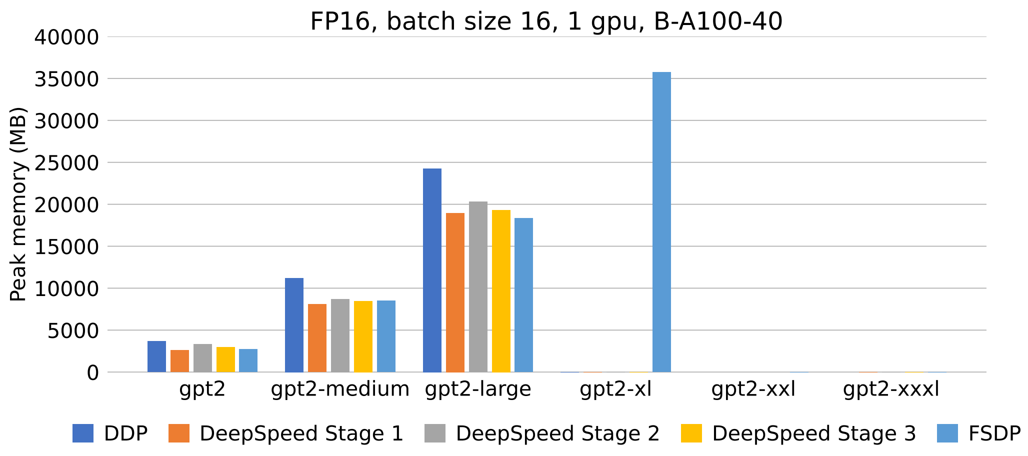

Figure 6 (a): Peak memory using 1 GPU.

Observations:

- Utilising the DeepSpeed Stage 3 strategy enables training of the largest model size, `gpt2-xxxl`, which necessitates a minimum of 16 GPUs.
- The FSDP strategy facilitates training of the `gpt2-xxl` model — the largest possible size with this method — when at least 4 GPUs are utilised.

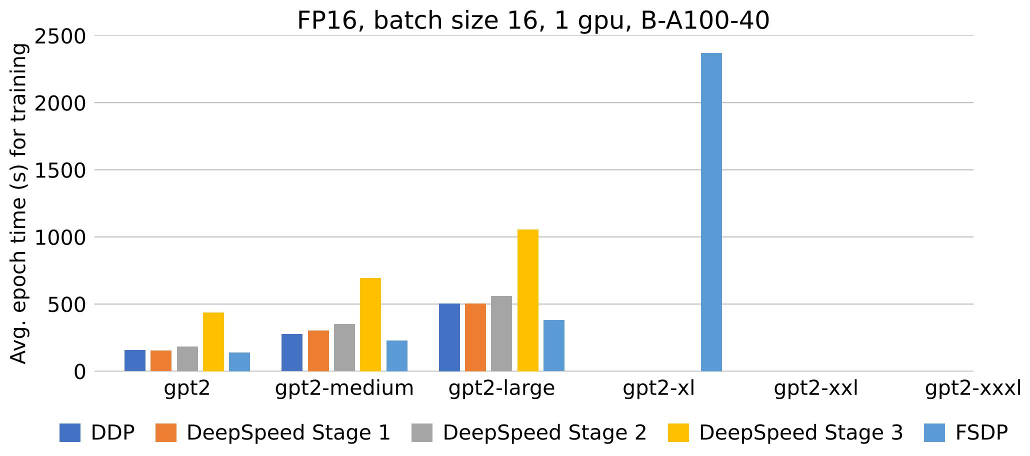

Figure 6 (b): Average epoch training time using 1 GPU.

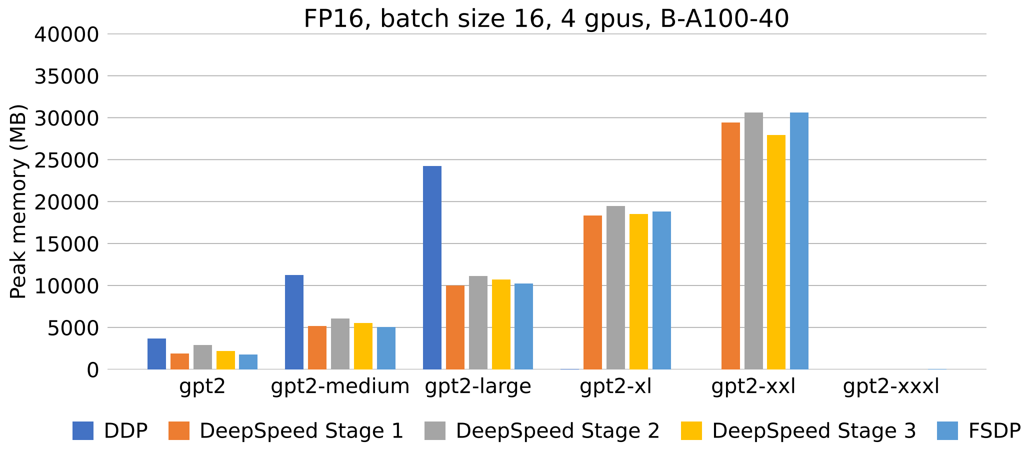

Figure 6 (c): Peak memory using 4 GPUs (1 node).

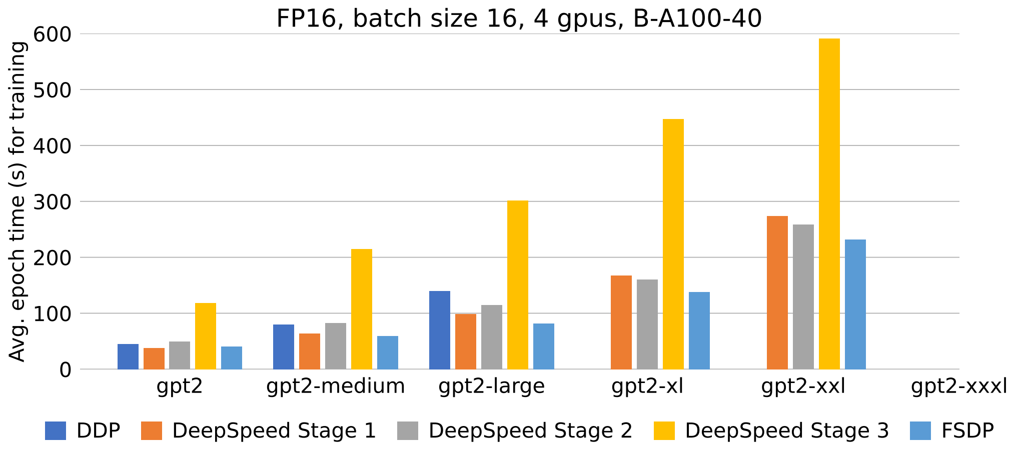

Figure 6 (d): Average epoch training time using 4 GPUs (1 node).

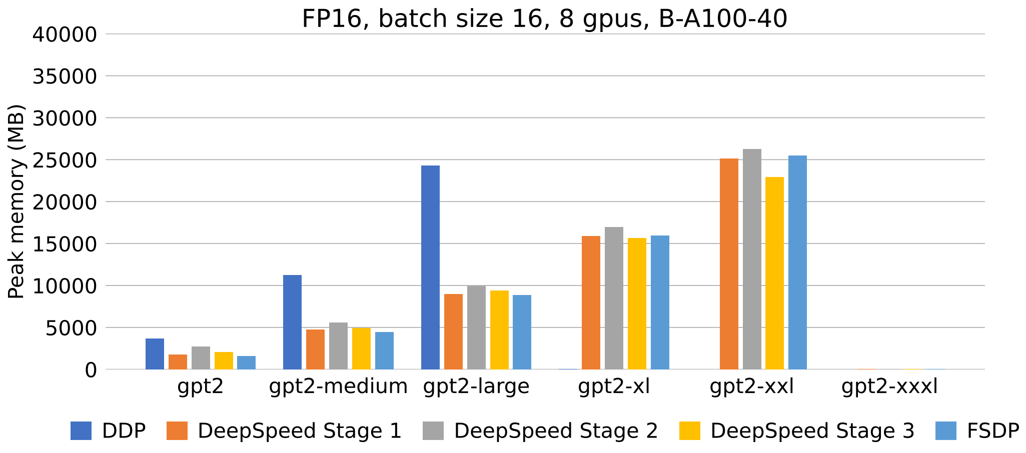

Figure 6 (e): Peak memory using 8 GPUs (2 nodes).

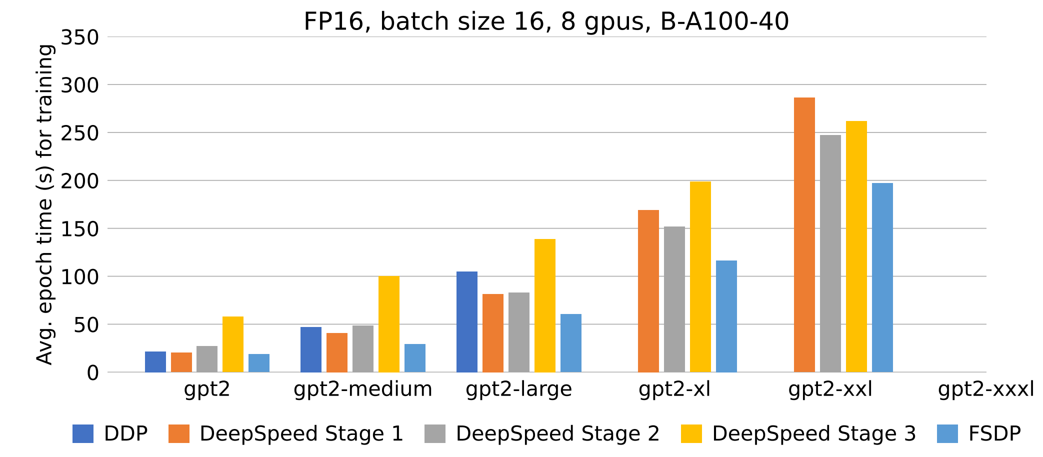

Figure 6 (f): Average epoch training time using 8 GPUs (2 nodes).

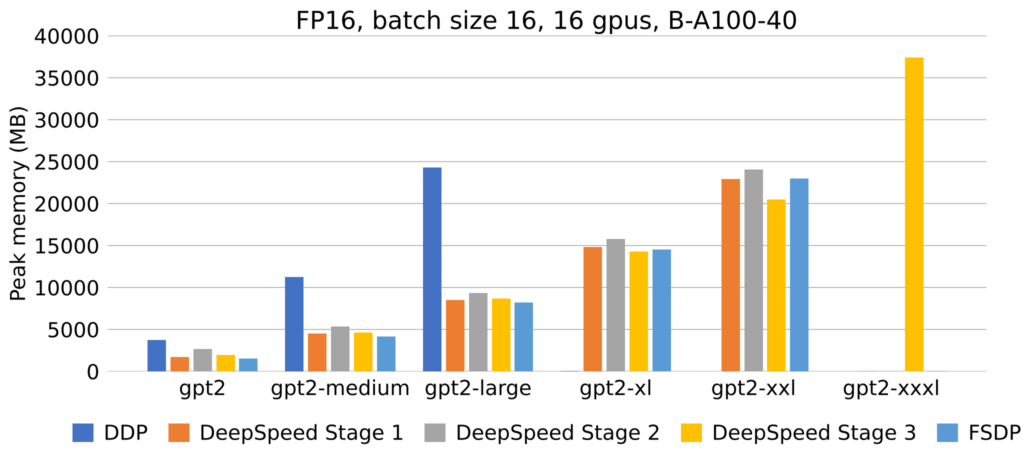

Figure 6 (g): Peak memory using 16 GPUs (4 nodes).

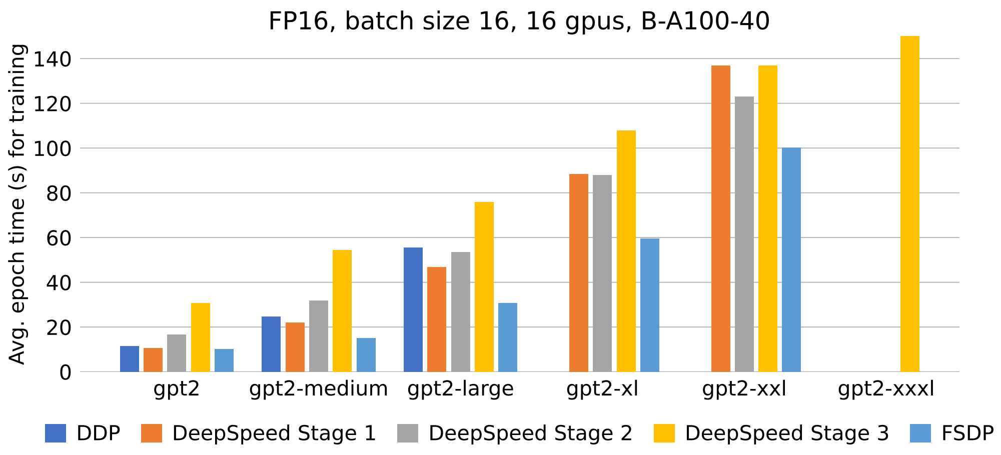

Figure 6 (h): Average epoch training time using 16 GPUs (4 nodes). Note that DeepSpeed Stage 3 average epoch time for `gpt2-xxxl` was 1050.21 s and did not fit on the graph.

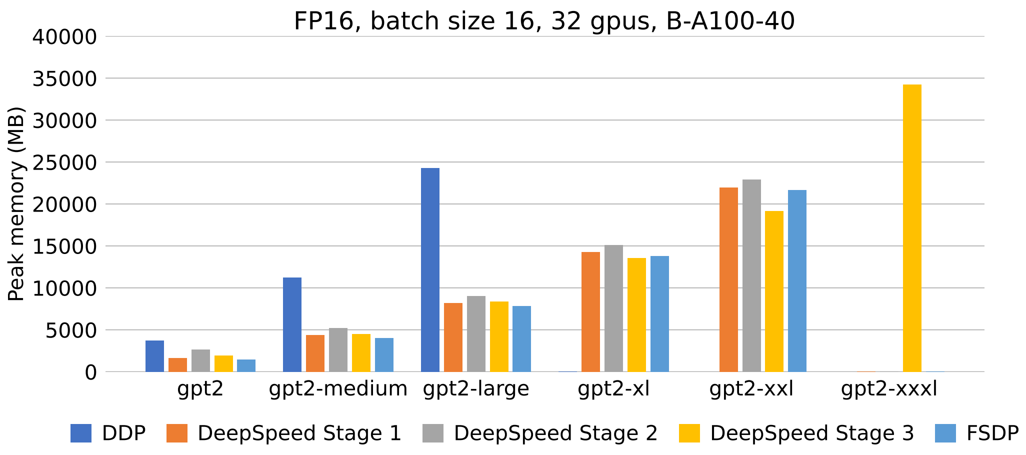

Figure 6 (i): Peak memory using 32 GPUs (8 nodes).

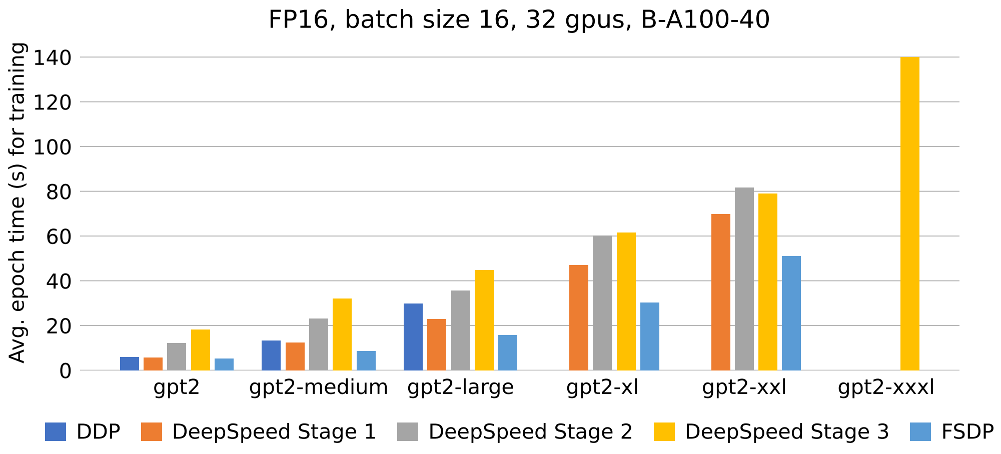

Figure 6 (j): Average epoch training time using 32 GPUs (8 nodes).

- Employing solely 1 GPU resulted in lower peak memory usage with the DeepSpeed and FSDP strategies as compared to the DDP strategy; see figure 6 (a). Depending on model size and strategy, the peak memory usage was, on average, 22% lower. While the FSDP not only reduced peak memory usage but also decreased the average epoch time by 17% compared to the DDP strategy, the DeepSpeed strategies lowered peak memory usage but escalated the average epoch time by 3%, 19%, and 146% for Stages 1, 2, and 3, respectively, relative to the DDP strategy (figure 6 (b)). Notably, only the FSDP strategy was able to train the `gpt2-xl` model on a single GPU, a feat that was unattainable with the DDP and DeepSpeed strategies.

- Leveraging 4 GPUs (1 node) resulted, on average, in a 50% reduction in peak memory usage with both the DeepSpeed and FSDP strategies compared to the DDP strategy (see Figures 5(a) and (c)), leading to more than a 2-fold reduction relative to using 1 GPU for the `gpt2-large` model. Concerning the differential in peak memory usage between DeepSpeed Stages 1, 2, and 3 and FSDP, DeepSpeed was, on average, 1%, 14%, and 6% higher than FSDP, respectively. For larger model sizes, namely `gpt2-xl` and `gpt2-xxl`, the differential in peak memory usage between DeepSpeed and FSDP remained within 10%, with stages 1 and 3 of DeepSpeed presenting marginally lower peak memory usage than FSDP. As for the average epoch time, FSDP was, on average, 26%, 12%, 26%, and 221% faster compared to DDP and DeepSpeed Stages 1, 2, and 3, respectively; see figure 6 (d).

- Analogous trends were observed for 8, 16, and 32 GPUs (or 2, 4, and 8 nodes). The average peak memory usage, when applying both the DeepSpeed and FSDP strategies relative to the DDP strategy, diminished by 54%, 57%, and 58% respectively, for model sizes ranging from `gpt2` to `gpt2-large`.

- When employing more than 8 GPUs, the DeepSpeed Stage 3 strategy exhibited the lowest average peak memory usage for model sizes of `gpt2-xl` or larger. It was the sole strategy capable of training the `gpt2-xxxl` model, necessitating at least 16 GPUs. However, the memory savings with DeepSpeed Stage 3 were counterbalanced by a significantly increased average epoch time, being between 1.3 to 2.2 times slower compared to FSDP.

8 Conclusions

- When comparing GPU accelerators, BF16 peak performance seems to be a better indicator of performance for AI workloads than FP16 or FP32 peak performance.

- The `B-A100` utilises Tensor Core [27] technology, enabling mixed-precision training and, therefore, higher throughput for AI workloads. In contrast, the `C-MI100-32` and `C-MI210-64` can offer higher peak performance for FP16 and FP32 workloads, potentially making them more suitable for traditional HPC workloads.

- The `D-MX1550-128` utilises Matrix Engine technology, which is well-suited for BF16 workloads.

- The DDP strategy allows for data parallelism only; as a result, the model size is limited by the amount of memory available on a single accelerator.

- The largest model size we were able to train on a single `B-A100-80` accelerator using DDP is `gpt2-xxl`, which has 2.7 billion parameters and requires 57.2 GB of memory.

- To train larger models, model or pipeline parallelism is required, as it enables the splitting of the model across multiple accelerators, thereby reducing the memory requirements on a single accelerator and allowing for the training of larger models.

- The Intel Ponte Vecchio GPU compared favourably with equivalent NVIDIA GPUs on a single node but lost ground as we scaled up, potentially due to increased communication overhead from having twice as many processes or system architecture differences. It's possible this could be alleviated through software and configuration changes but requires further exploration.
- Both DeepSpeed and FSDP strategies showed improvements in peak memory usage and average epoch time compared to the DDP strategy, even on a single GPU.
- Incrementing the GPU count consistently demonstrated diminishing average peak memory usage across a range of model sizes when implementing both DeepSpeed and FSDP strategies, highlighting the clear benefits of using these strategies for large-scale experiments and model training.
- In this study, the largest model sizes that can be trained using DeepSpeed and FSDP strategies are `gpt2-xxxl` (6.6 billion parameters) and `gpt2-xxl` (2.7 billion parameters) with at least 16 and 4 GPUs, respectively.
- The FSDP strategy is faster than the DeepSpeed strategy, but DeepSpeed Stage 3 appears to be more memory-efficient for the largest models.
- Nuances in peak memory usage and epoch time between DeepSpeed and FSDP strategies necessitate a balanced consideration of both memory and time efficiency, especially when scaling to larger models.

9 Limitations and future work

The future work would greatly benefit from addressing the many limitations of this study:

- Diversity of accelerators and their types: Only specific NVIDIA, AMD, and Intel GPUs, as well as a Graphcore IPU were used. It would be beneficial to include other GPUs or different types of accelerators, such as FPGAs, TPUs, IPUs, etc., in the study.
- Diversity of models: This study exclusively utilized a specific GPT-2 model. However, including other models, such as BERT, XLNet, RoBERTa, GPT-3 would be beneficial, as different models exhibit distinct memory and computational characteristics and requirements.

- Diversity of parallelism techniques: This study only examined DDP, DeepSpeed, and FSDP. However, including other parallelism techniques, such as tensor, pipeline, and hybrid parallelisms, would be beneficial for the study.
- Diversity of workflows: The study was also limited to examining the training of the models. However, fine-tuning and inference are also important parts of the ML/AI workflow. It would be beneficial to investigate the inference performance of the models.
- Diversity of workloads: This work would also benefit from exploring not only large language models (LLMs) but also other types of ML/AI workloads, such as computer vision, speech recognition, etc.
- Consideration of data throughput: We have considered the effect of varying the training data size, but only to a limited extent. Testing much larger datasets (larger than the available GPU memory) and different storage devices, along with analysing data throughput rates to identify bottlenecks, would provide additional insight.

10 Acknowledgements

We're grateful to Edwin Brown (jointly from the University of Sheffield and the Alan Turing Institute) for his input and generous support, especially in running various experiments for us on Stanage.

This work was funded by The Alan Turing Institute under the EPSRC grant EP/N510129/1. Additionally, it was partially supported by Baskerville, a national accelerated compute resource under the EPSRC Grant EP/T022221/1; JADE: Joint Academic Data Science Endeavour - 2 under the EPSRC Grant EP/T022205/1; The Exascale Computing: Algorithms and Infrastructures Benefiting UK Research (ExCALIBUR) program, which is funded under Wave 2 of the Strategic Priorities Fund (SPF); and using resources provided by the Cambridge Service for Data Driven Discovery (CSD3) operated by the University of Cambridge Research Computing Service, provided by Dell EMC and Intel using Tier-2 funding from the Engineering and Physical Sciences Research Council (capital grant EP/T022159/1).

We acknowledge IT Services at The University of Sheffield for the provision of services for High Performance Computing. We also acknowledge the use of the

Mandelbrot system at the UCL Centre for Advanced Research Computing, and associated support services, in the completion of this work.

This work used the DiRAC@Durham facility managed by the Institute for Computational Cosmology on behalf of the STFC DiRAC HPC Facility. The equipment was funded by BEIS capital funding via STFC capital grants ST/P002293/1, ST/R002371/1 and ST/S002502/1, Durham University and STFC operations grant ST/R000832/1. DiRAC is part of the National e-Infrastructure.

This support granted access to the Baskerville, JADE2, DAWN, Stanage, Graphcore and Cosma-8 computing facilities, respectively. We're grateful to all involved for allowing us to use these services and for the helpful feedback on this manuscript.

References

[1] URL: https://openai.com/research/gpt-4 (visited on 24/07/2024).

[2] Jacob Devlin et al. *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*. 2019. arXiv: 1810.04805.

[3] URL: https://github.com/facebookresearch/llama (visited on 24/07/2024).

[4] URL: https://falconllm.tii.ae (visited on 24/07/2024).

[5] Exascale Computing ALgorithms & Infrastructures Benefiting UK Research (ExCALIBUR). URL: https://excalibur.ac.uk (visited on 24/07/2024).

[6] *NVIDIA DGX-1*. NVIDIA. URL: https://www.nvidia.com/en-gb/data-center/dgx-systems/dgx-1/ (visited on 24/07/2024).

[7] *JADE HPC UK*. JADE. URL: https://www.jade.ac.uk/ (visited on 24/07/2024).

[8] *Baskerville System*. Baskerville. URL: https://docs.baskerville.ac.uk/system/ (visited on 24/07/2024).

[9] *Lenovo ThinkSystem SD650-N V2 Server*. Lenovo. URL: https://lenovopress.lenovo.com/lp1396-thinksystem-sd650-n-v2-server (visited on 24/07/2024).

[10] *Stanage*. University of Sheffield. URL: https://docs.hpc.shef.ac.uk/en/latest/stanage/ (visited on 24/07/2024).

[11] *PowerEdge R7525 Spec Sheet*. Dell Technologies. URL: https://i.dell.com/sites/csdocuments/Product_Docs/en/PowerEdge-R7525-Spec-Sheet.pdf (visited on 24/07/2024).

[12] *The Cosma 7 Supercomputer*. Durham University. URL: https://www.dur.ac.uk/icc/cosma/cosma8/ (visited on 24/07/2024).

[13] *DAWN*. University of Cambridge, Research Computing Services. URL: https://www.hpc.cam.ac.uk/d-w-n (visited on 24/07/2024).

[14] *PowerEdge XE9640 Rack Server*. Dell Technologies. URL: https://www.dell.com/en-uk/shop/ipovw/poweredge-xe9640 (visited on 24/07/2024).

[15] *IPU-POD16 Direct Attach Datasheet - Overview*. Graphcore. URL: https://docs.graphcore.ai/projects/ipu-pod16-datasheet/en/latest/overview.html (visited on 24/07/2024).

[16] *PowerEdge R6525 Rack Server*. Dell Technologies. URL: https://www.dell.com/en-uk/shop/ipovw/poweredge-r6525 (visited on 24/07/2024).

[17] *NVIDIA Tesla V100 SXM2 32 GB Specs*. TechPowerUp. URL: https://www.techpowerup.com/gpu-specs/tesla-v100-sxm2-32-gb.c3185 (visited on 24/07/2024).

[18] *NVIDIA V100 Tensor Core GPU Datasheet*. NVIDIA. URL: https://images.nvidia.com/content/technologies/volta/pdf/volta-v100-datasheet-update-us-1165301-r5.pdf (visited on 24/07/2024).

[19] *NVIDIA A100 Tensor Core GPU Datasheet*. NVIDIA. URL: https://www.nvidia.com/content/dam/en-zz/Solutions/Data-Center/a100/pdf/nvidia-a100-datasheet-us-nvidia-1758950-r4-web.pdf (visited on 24/07/2024).

[20] *NVIDIA H100 Tensor Core GPU Datasheet*. NVIDIA. URL: https://nvdam.widen.net/content/vuzumiozpb/original/h100-datasheet-2287922.pdf (visited on 24/07/2024).

[21] *AMD Instinct MI100 Product Brief*. AMD. URL: https://www.amd.com/content/dam/amd/en/documents/instinct-business-docs/product-briefs/instinct-mi100-brochure.pdf (visited on 24/07/2024).

[22] *AMD Instinct MI210 Product Brief*. AMD. URL: https://www.amd.com/content/dam/amd/en/documents/instinct-business-docs/product-briefs/instinct-mi210-brochure.pdf (visited on 24/07/2024).

[23] *Intel Data Center GPU Max 1550 Specs*. TechPowerUp. URL: https://www.techpowerup.com/gpu-specs/data-center-gpu-max-1550.c4068 (visited on 24/07/2024).

[24] *Intel Data Center GPU Max Series Technical Overview*. Intel. URL: https://www.intel.com/content/www/us/en/developer/articles/technical/intel-data-center-gpu-max-series-overview.html (visited on 24/07/2024).

[25] *IPU-POD16 Direct Attach Datasheet - Technical specifications*. Graphcore. URL: https://docs.graphcore.ai/projects/ipu-pod16-datasheet/en/latest/product-description.html#technical-specifications (visited on 24/07/2024).

[26] *Mixed-Precision Arithmetic for AI: A Hardware Perspective - The IPU 16-bit floating point format*. Graphcore. URL: https://docs.graphcore.ai/projects/ai-float-white-paper/en/latest/ai-float.html#the-ipu-16-bit-floating-point-format (visited on 24/07/2024).

[27] *NVIDIA Tensor Cores*. URL: https://www.nvidia.com/en-gb/data-center/tensor-cores/ (visited on 24/07/2024).

[28] *AMD CDNA Architecture*. URL: https://www.amd.com/en/technologies/cdna.html (visited on 24/07/2024).

[29] *AMD CDNA 2 Architecture White Paper*. URL: https://www.amd.com/content/dam/amd/en/documents/instinct-business-docs/white-papers/amd-cdna2-white-paper.pdf (visited on 24/07/2024).

[30] *Mixed-Precision Arithmetic for AI: A Hardware Perspective*. Graphcore Ltd. URL: https://docs.graphcore.ai/projects/ai-float-white-paper/en/latest/ (visited on 24/07/2024).

[31] Andrej Karpathy. *minGPT: A Minimalist, PyTorch-based Transformer implementation*. URL: https://github.com/karpathy/minGPT (visited on 24/07/2024).

[32] URL: https://github.com/llewelld/lit-GPT (visited on 24/07/2024).

[33] *PyTorch Lightning*. URL: https://www.pytorchlightning.ai (visited on 24/07/2024).

[34] *Distributed Data Parallel*. URL: https://pytorch.org/docs/stable/generated/torch.nn.parallel.DistributedDataParallel.html#torch.nn.parallel.DistributedDataParallel (visited on 24/07/2024).

[35] *DeepSpeed*. URL: https://github.com/microsoft/DeepSpeed (visited on 24/07/2024).

[36] URL: https://lightning.ai/docs/pytorch/latest/advanced/model_parallel/deepspeed.html#deepspeed (visited on 24/07/2024).

[37] *Fully Sharded Data Parallel: faster AI training with fewer GPUs.* URL: https://engineering.fb.com/2021/07/15/open-source/fsdp/ (visited on 24/07/2024).

[38] *Train models with billions of parameters using FSDP.* URL: https://lightning.ai/docs/pytorch/latest/advanced/model_parallel/fsdp.html (visited on 24/07/2024).

[39] *DeepSpeed: Extreme-scale model training for everyone.* URL: https://www.microsoft.com/en-us/research/blog/deepspeed-extreme-scale-model-training-for-everyone/ (visited on 24/07/2024).

[40] *Accelerate Large Model Training using DeepSpeed.* URL: https://huggingface.co/blog/accelerate-deepspeed (visited on 24/07/2024).

[41] Samyam Rajbhandari et al. *ZeRO: Memory Optimization Towards Training A Trillion Parameter Models.* 2019. arXiv: 1910.02054.

[42] *IPU Programmer's Guide.* Graphcore Ltd. URL: https://docs.graphcore.ai/projects/ipu-programmers-guide/en/latest/ (visited on 24/07/2024).

[43] Andrej Karpathy. *All works of Shakespeare concatenated.* URL: https://cs.stanford.edu/people/karpathy/char-rnn/ (visited on 24/07/2024).

[44] Leo Gao et al. *The Pile: An 800 GB Dataset of Diverse Text for Language Modeling.* 2020. arXiv: 2101.00027.

[45] *PyTorch for the IPU: User Guide.* Graphcore Ltd. URL: https://docs.graphcore.ai/projects/poptorch-user-guide/en/latest/ (visited on 24/07/2024).

A Graphcore IPU-IPOD 16

In addition to the five GPU-based systems covered in the document earlier, we also performed the same benchmarking tests on a Graphcore IPU-POD-16 system, which makes use of "Intelligent Processor Units" (IPUs) specifically designed for Artificial Intelligence based workflows.

The characteristics and programming models used by the system are quite different, making direct comparisons with the GPU systems challenging. For example, the optimisation strategies are different, so we aren't able to use DDP, DeepSpeed or FSDP with the IPUs. Instead, four alternative strategies are supported which we describe below. The execution characteristics are also quite different. Before running the code the compiler performs a simulated run in order to automatically optimise execution for use on the IPUs. This can increase execution time with the result that the first epoch is always considerably slower than subsequent epochs. Finally, the memory structure of the system is also quite unusual. Each IPU has a large amount of SRAM (referred to as "In-Processor-Memory") organised into smaller independent distributed memory units. It also has a set of attached DRAM (referred to as "Streaming Memory") which can transfer to the SRAM by explicit request of the code [42, section 2].

The `IPU-POD-16` device used for our testing has 3.6 GiB of In-Processor-Memory per IPU and 512 GiB of Streaming Memory, giving a total of 526.4 GiB. However, because of the way this memory is arranged, we found using the same parameters (model size and batch size) as with the GPUs-based systems to be problematic. This also makes it more challenging to compare the results.

The four strategies available for use with PyTorch with IPUs are Pipelined Execution, Sharded Execution, Serial Phased Execution and Parallel Phased Execution. Our benchmarking code is built using PyTorch Lightning and so is restricted to making use of these strategies.

`Pipelined Execution` [45, Pipelined Execution] requires that the model is split into *phases* with ideally the same number of phases as IPUs (up to 16 in our case). Each phase can be further subdivided into *stages*. PopTorch will stagger the phases across the IPUs so that once a single batch has been processed by the first phase on one IPU the outputs can be passed to the second phase on a different IPU. The next batch can then be passed to the first IPU to run the first phase again in parallel with the second phase on the second IPU. The aim is to split the model across the IPUs and have them all processing in parallel to the maximum extent. For a short time at the start of each epoch, there is a "ramp-up" period during which IPUs later in the chain are waiting for data; similarly, towards the end of each epoch, there is a "ramp-down" period.

With this strategy, the forward and backward passes are interleaved across the IPUs so that the last forward stage can be combined with the first backward stage. Using this strategy the effective batch size becomes the configured batch size multiplied by the number of IPUs in operation.

`Sharded Execution` [45, Sharded Execution] has each IPU sequentially execute a distinct part (a *shard*)) of the model. These shards split the model vertically, similarly to the FSDP approach shown in Figure 1.3. Each shard is executed sequentially on a single GPU. This approach is generally inefficient compared to other approaches because only one IPU is being used at any point in time. It's recommended if only a single sample is being processed or for debugging.

`Phased Execution` [45, Phased Execution] comes in two flavours: Parallel Phased Execution and Serial Phased Execution. With this strategy, some portion of weights and activations are offloaded to Streaming Memory between phases. This can be useful for models with larger memory footprints at the expense of some efficiency, although the compiler optimises the code to use cross-IPU copies of weights and activations rather than using Streaming Memory where possible. As with Pipelined Execution, phases must be defined to aggregate processing across IPUs.

In the case of Parallel Phased Execution phases are executed in parallel split between even and odd numbered IPUs (so odd IPUs hand off to even IPUs and *vice versa*). In the case of Serial Phased Execution phases operate sequentially on a single set of IPUs.

A notable characteristic of all of these strategies is that they require annotations to be added to the model code in order to distribute the model effectively across the IPUs. This introduces a challenge for benchmarking since there's no canonical approach for partitioning the model into phases or shards in an ideal way for the IPUs (the `AutoStage` method assigns stages to blocks, but still requires the blocks to be manually defined in the model).

Simplified representations of Pipelined Execution and Sharded Execution are shown in figure 7.

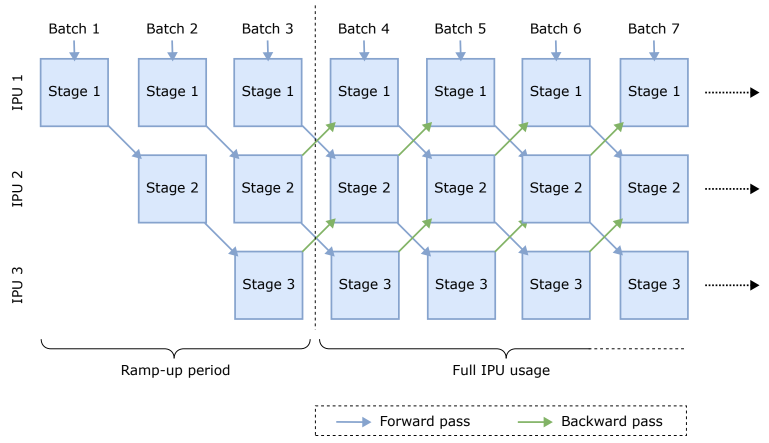

Figure 7 (a): Schematic representation of Pipelined Execution on a set of IPUs.

We ran tests to benchmark the `IPU-POD-16` system using different strategies and numbers of IPUs. IPUs must be utilised in powers of two. We used the same

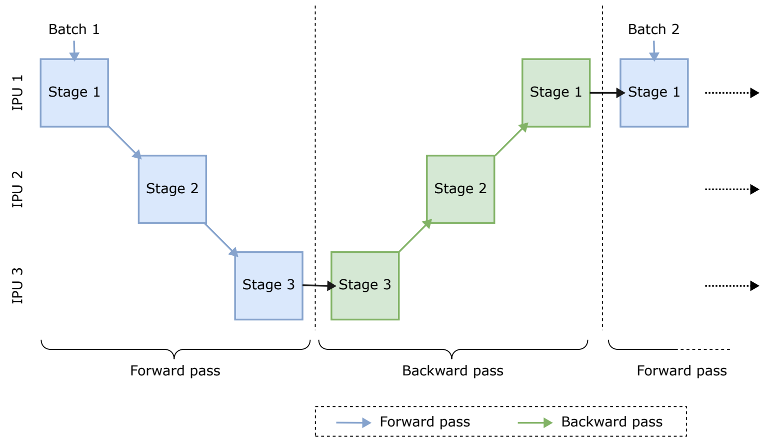

Figure 7 (b): Schematic representation of Sharded Execution on a set of IPUs.

min-GPT model as for the other experiments but due to the way the memory is arranged had to use different hyperparameters. Our training model contained 12 layers and 12 heads with an embedding of either 384 or 768 in order to test different model sizes (21.4 million and 85.3 million parameters respectively, amounting to 42.79 MiB and 170.514 MiB respectively). We constrained training to a batch size of 1. Precision was set to 16-bit and epoch time is the average over a total of 10 epochs.

For the larger model size, a minimum of four IPUs was needed to avoid memory exhaustion.

As discussed above, an important difference between GPU-based training and the Graphcore IPU approach is that an initial compilation stage is needed to generate a multi-operation computation graph. This process ensures that tile usage is maximised while also minimising memory requirements. This compilation stage can take some time and although the result can be cached it is highly dependent on the parameters, including hyperparameters, of the model [42, section 4.4, "Compilation"].

Compilation is therefore a normal part of operation and an important consideration. In the figure 8 we detail both the compilation and average epoch times.

Observations:

- For both Sharded and Pipelined Execution strategies the compilation time generally increases with the number of IPUs used (approximately logarithmic in the number of IPUs). This relationship holds for both model sizes.

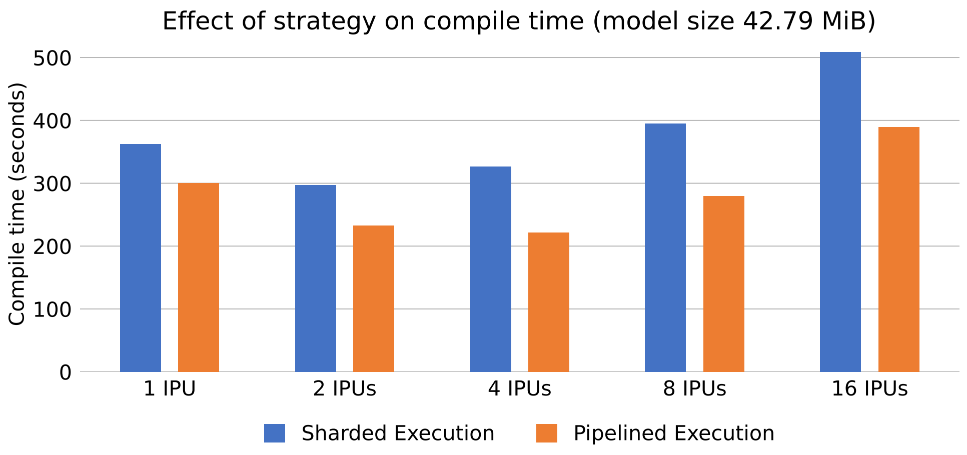

Figure 8 (a): The effect of strategy and number of IPUs on compile time, smaller model.

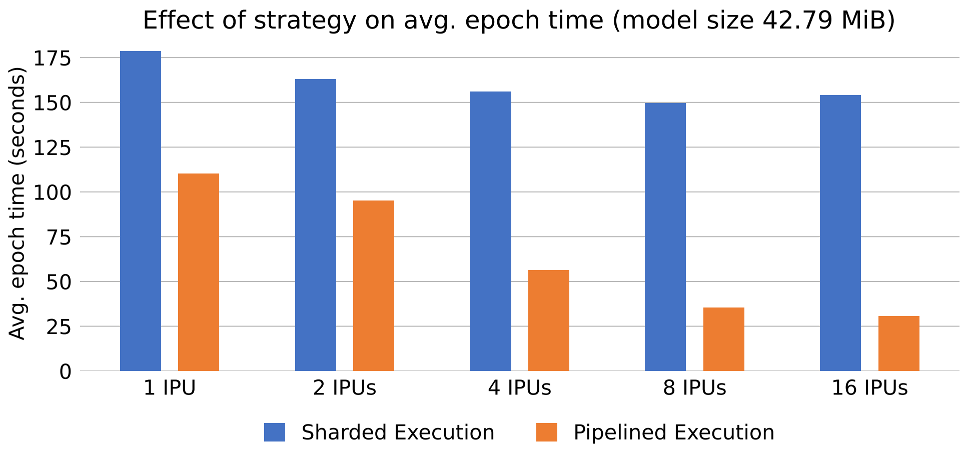

Figure 8 (b): The effect of strategy and number of IPUs on average training epoch time, smaller model.

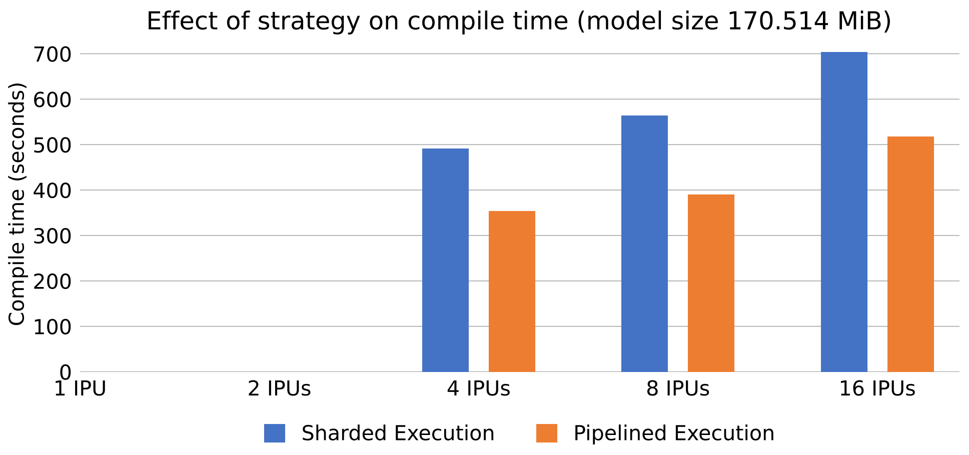

Figure 8 (c): The effect of strategy and number of IPUs on compile time, larger model.

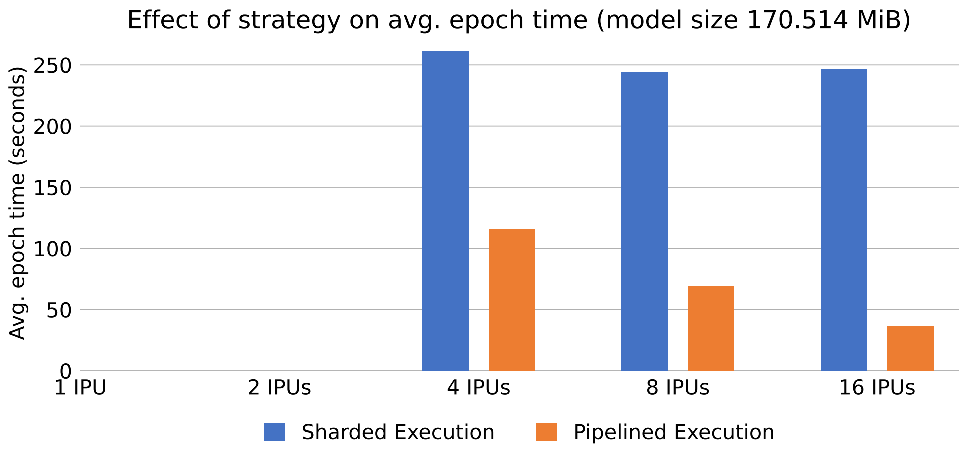

Figure 8 (d): The effect of strategy and number of IPUs on average training epoch time, larger model.

- Parallel Phased Execution ran considerably slower — on average around 50 times slower — than Pipelined Execution; we've skipped the results from the figure because of this big difference. Phased Execution moves parameters from In-Processor-Memory to Streaming Memory between phases, which likely accounts for the slower execution. One important benefit is that we were able to train the larger 170.514 MiB model even on a single IPU using Parallel Phased Execution, even though this failed with both Sharded and Pipelined.

- We were unfortunately unable to get a working Serial Phased Execution implementation due to the need for execution phases to be data-independent. The structure of the code made it challenging to annotate the code in a suitable way.

- Increasing the model size also causes the compilation time to increase in a non-trivial way.

- Epoch time decreases with the number of IPUs in use. As expected, the decrease is far more marked for pipelined execution in comparison to sharded execution. In the latter case, increasing the number of IPUs has little practical effect.

- For these examples, the compile time was nearly double the time taken to run a single training epoch. The impact of compile time therefore reduces proportionally as the number of epochs increases. For our ten-epoch experiments compilation constituted a non-trivial portion of the overall execution.

- As expected, in general, Pipelined Execution is the far superior strategy in terms of model training efficiency compared to Sharded Execution.