GRAPHCORE

BUILDING LARGE MODELS ON IPU SYSTEMS

White Paper

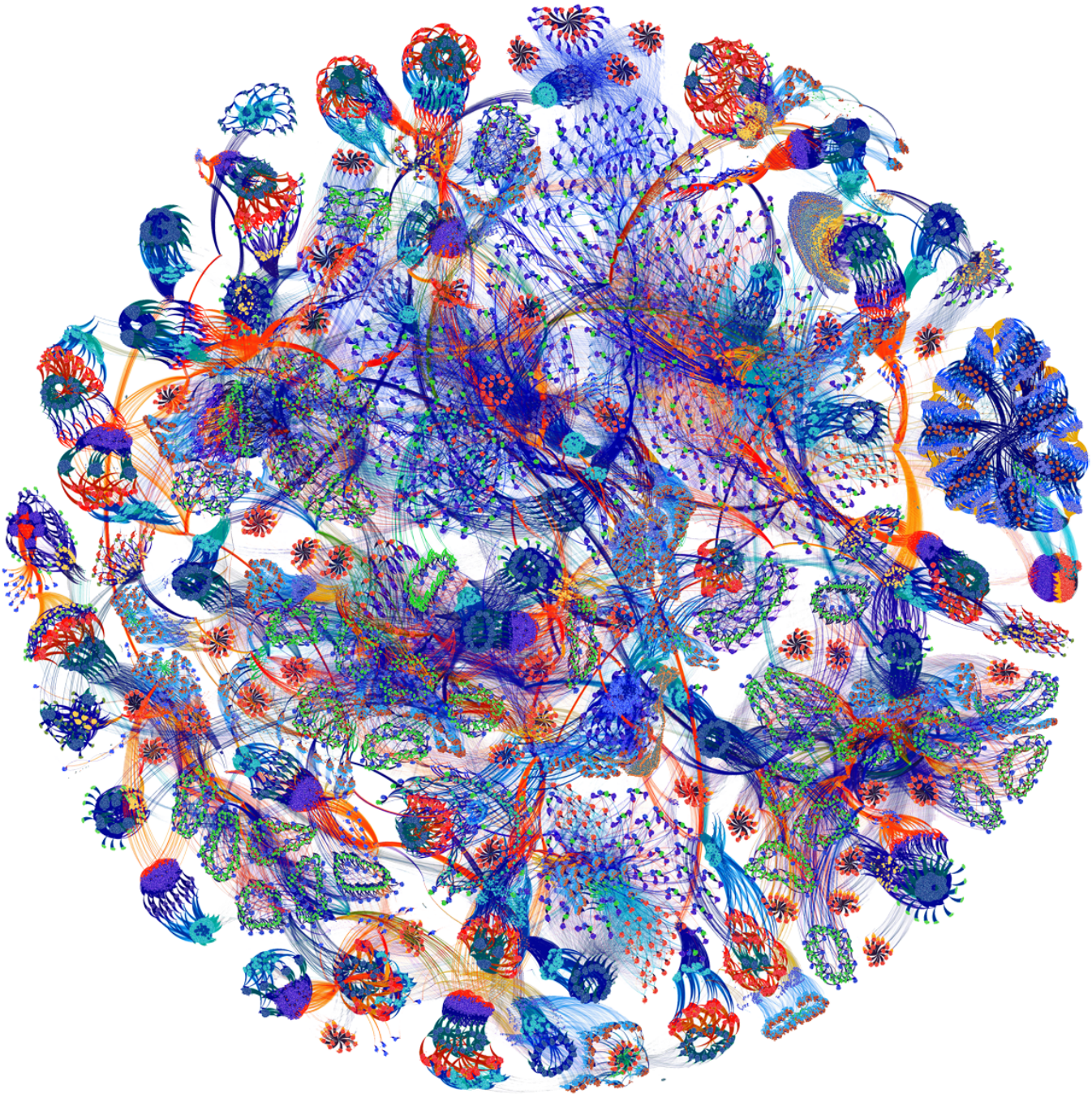
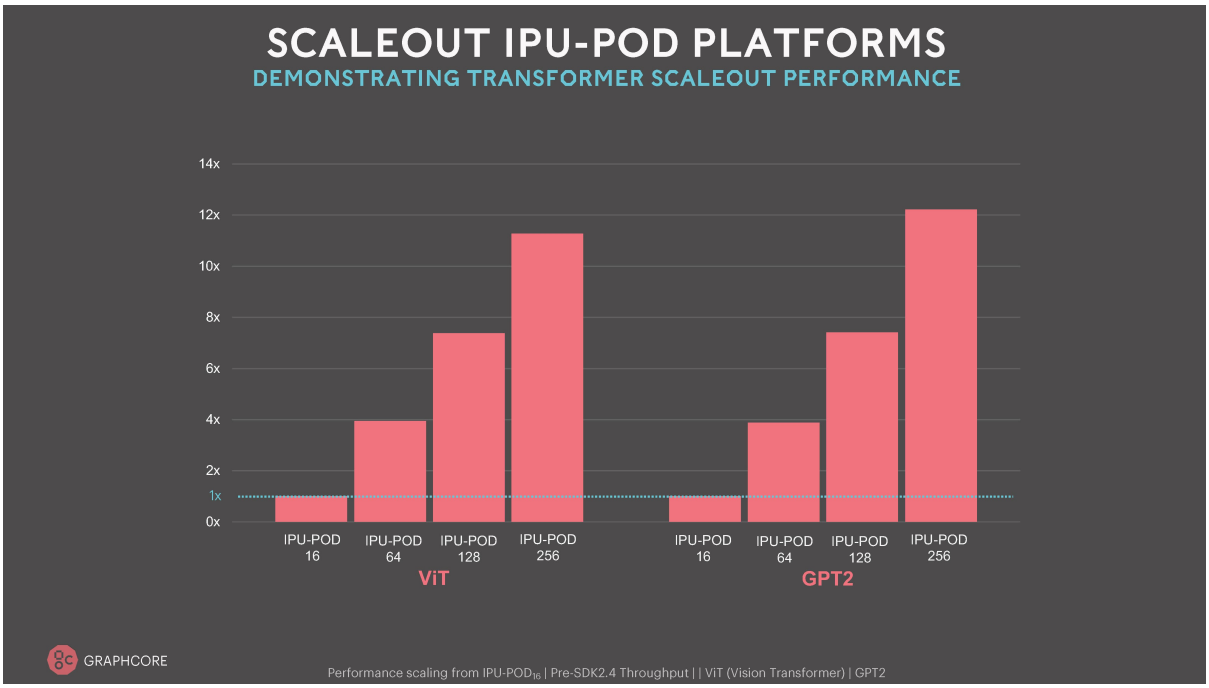


TABLE OF CONTENTS

TOWARDS TRILLION-PARAMETER MODELS

Graphcore IPU-POD systems are being deployed today by customers for training and fine-tuning large models. We have huge demand for natural language processing with an increasing interest in Generative Pre-Trained Transformer (GPT) models from forward-thinking organisations in banking, healthcare, insurance, government, manufacturing and other AI-first enterprises.

Our results in MLPerf have shown our ability to surpass other solutions in cost efficiency for today's models. The compute scaling we shared at the same time as our MLPerf results on innovative, up and coming Transformer models like GPT and ViT (Vision Transformer) shown in the diagram below on our flagship large IPU-POD$_{128}$ and IPU-POD$_{256}$ systems shows the bandwidth and capacity of our IPU-Link and IPU-Fabric. We also demonstrated the maturity of our software stack, Poplar, which consistently delivers impressive performance optimisation across all machine learning models.



The performance capability of our large IPU-POD systems is truly impressive. No other innovative AI systems can deliver this much compute with so many independent programs and with this capacity of memory that delivers PetaByte/sec memory bandwidth.

	IPU-POD$_{128}$	IPU-POD$_{256}$
Compute	32 PetaFlops FP16	64 PetaFlops FP16
Separate Processor Cores	188,416	376,832
Independent parallel programs	1.1 Million	2.2 Million
In-Processor Memory Capacity	115GiB	230GiB
In-Processor Memory Bandwidth	5.8 PetaByte/s	11.5 PetaByte/s
IPU Exchange Memory Capacity	8 TeraByte	16 TeraByte

This new level of performance is allowing customers to experiment and to build new types of large models on IPU-POD$_{128}$ and IPU-POD$_{256}$ systems. The IPU-POD$_{256}$ has the potential to hold models with tens of billions of parameters all inside the IPU's

In-Processor Memory that can be accessed at tens of PetaByte/sec memory bandwidth. This combination of model size with such high bandwidth is allowing innovators to explore new types of large models that are unlocking the potential of sparse computing and opening the door to new approaches in AI. We can now start to develop models where only the relevant training data is directed to the correct parameters for much more efficient model training. We can start to break free from the exponential growth in compute that conventional processors demand with today's large model approaches.

But you are not limited to **billions** of parameters because Graphcore's IPU Exchange Memory, that is supported through attached streaming memory and our Poplar software SDK, allows much larger models to be easily supported. The 16 TeraBytes of IPU Exchange Memory capacity in an IPU-POD$_{256}$ system opens the possibility to support models with **trillions** of parameters. These brain-scale models can also benefit from the new compute approaches made possible by our IPU processors.

In this white paper, we will show some high-level examples of how developers are mapping these large models on to IPU-POD systems, all supported by our mature and easy to use software stack, Poplar.

GRAPHCORE IPU MEMORY SYSTEM

When we developed the IPU we focused on building a highly scalable memory system. This is one of the most exciting and forward-looking aspects of our IPU processor. We developed a radical new approach to memory organisation.

First, we focused on having a huge amount of In-Processor Memory. Each Colossus GC200 IPU has nearly 1 GigaByte of memory inside the processor that can be accessed at the full compute speed of the processor - this is unprecedented. Secondly, we made sure that the IPU could access other memory sources, which we call Streaming Memory. Our IPU-Machine M2000 system contains plug-in DDR-DIMM modules just for this purpose. These are typically configured with 256 GigaByte of streaming memory but can also be configured with larger DIMM modules for even larger capacity memories. Under Poplar SDK software control, data and programs can be streamed from external memory sources into the IPU In-Processor Memory. Our Poplar software can orchestrate the exchange of data between the internal memory and the external streaming memory. The design of this sophisticated memory system is centred around two key principles:

- no assumptions about memory access behaviour are built into the hardware
- we allow full explicit software control of all memory accesses

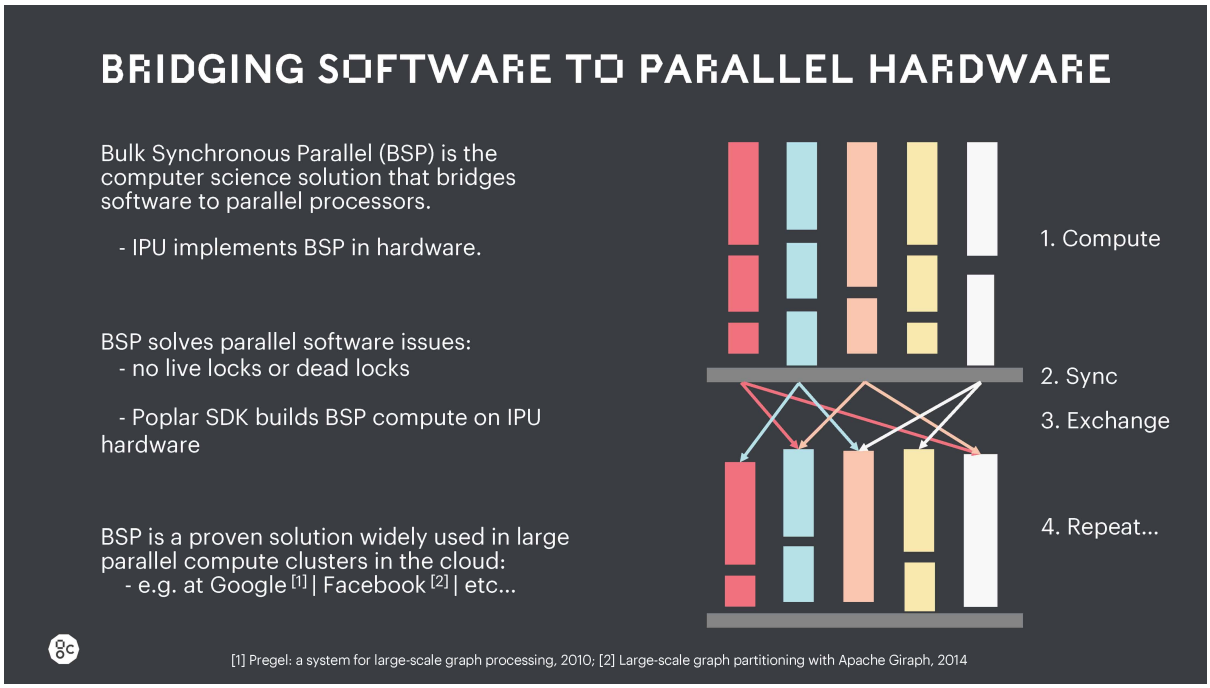
The high speed IPU-Links built into each IPU-Processor and the IPU-Fabric that is built into each of our IPU-Machine M2000 systems, not only allows us to build larger scale-out IPU-POD systems, but it also allows both the In-Processor Memory and the Streaming Memory to be shared across the full IPU-POD system. The same principles of streaming memory from the local DDR-DIMM in an IPU-Machine applies for streaming data over the IPU-Links from other IPUs and from other IPU-Machine memory. This can be orchestrated from any part of the IPU-POD system to any other part. All memory use is directed by our Poplar SDK which allows us to provide a platform for applications that deliver the most flexibility together with highly efficient memory accesses.

Exchange Memory is the term we use for these Poplar SDK features that manage the use of In-Processor and Streaming Memory both inside IPU, inside a single IPU-Machine and between IPU-Machines in an IPU-POD system. The Exchange Memory features that Poplar supports are extensive and have been carefully designed and are constantly being added to. Like any memory hierarchy, the tools must efficiently balance between fast local memory (of which the IPU has vastly more than most processors) and Streaming Memory (which gives a much larger capacity).

OPTIMISED DATA COMMUNICATION

Streaming memory can be used to optimise the time taken to move data to and from the local IPUs or to IPUs in other IPU-Machines spread across the IPU-POD. Data can be replicated and the accesses sharded to support different compute approaches. The Streaming Memory can also act as an intermediate buffer for communication from the host. These optimisations are supported in the Poplar SDK and new optimisations help users easily pipeline and shard models. One of the great aspects of these optimisations is that they can work automatically with your application – no changes to the high-level framework code that defines your model is required.

The movement of data inside the IPU, between IPUs and between IPU-Machines is all supported under the IPU Bulk Synchronous Communication (BSP) scheme.



This robust parallel processing approach ensures that for highly parallel operations there are no race conditions and no live locks and there are no dead locks. This level of explicit parallel execution control is missing from all other processors but is critical in making the execution of large models robust and scalable across a large scale-out machine.

In addition to this BSP execution, the Poplar SDK manages all communication between the different processor cores and between the different independent parallel programs. The Poplar SDK also orchestrates streaming memory accesses so that they

happen in the background, hidden behind compute. This ensures that the data and code that is required next will always be available before this data and code is needed in the execution of the full program.

This sophistication of control makes it possible to orchestrate large compute tasks across a complete IPU-POD$_{128}$ system or IPU-POD$_{256}$ system. It is even possible to connect IPU-POD$_{256}$ systems together to build even larger systems and the Poplar SDK tools can support these systems too. The developer can easily explore different pipeline model parallel and tensor model parallel approaches to ensure that the highest performance is achieved. Activations or weights required later in the model training cycle can be offloaded to streaming memory and restored by Poplar later in the computing cycle. Sometimes it might be better to store only a small sample of the activations and then recompute the intermediate activations on the backward pass when training a model. This can save memory bandwidth and allows the very high level of compute available on IPUs to be used efficiently. This trade off can be used to balance compute vs. bandwidth. Some models may require larger state with less compute while other models may require larger compute activity on smaller state. This trade-off may also vary across the model training cycle and so it is possible to maintain a high level of control over the balance of compute and I/O bandwidth.

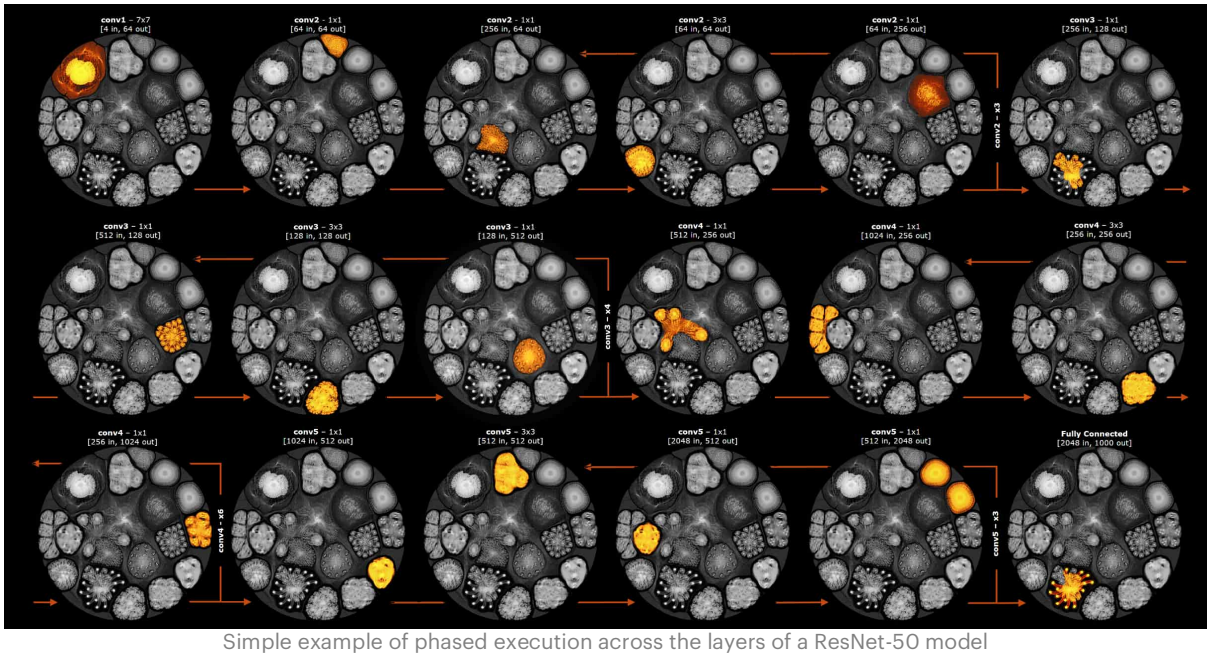
PHASED EXECUTION

Our Poplar SDK software takes a high-level model description from AI frameworks such as TensorFlow or PyTorch and maps this highly abstracted graph representation into a low-level compute graph that can then be compiled to run inside the IPU or across a group of IPUs.

The Poplar SDK system not only builds the compute, but it also orchestrates all the communication across the IPU exchange fabric that is inside each IPU processor and the communication across the IPU-Links and IPU-Fabric that connect the separate IPUs inside an IPU-POD system. Poplar also manages all the data storage and ensures that the correct data is in the right place at the right time to support the next phase of compute. This highly sophisticated approach is what makes it possible to scale large models across all the IPU processors in an IPU-POD system and even to spread the compute across multiple IPU-POD systems.

However, the Compute Graphs that Poplar produces can also be partitioned into smaller sub-Graphs that can then execute in phases on the IPU. The sub-graph state is moved in and out in background mode from the streaming memory, as the compute passes across the whole graph. We call this Phased Execution.

A very simple example of the subgraphs used in phased execution, showing the separate phases over the layers of a ResNet-50 model is shown below. The heat map shows the executing part of the compute graph at each computing phase:



Simple example of phased execution across the layers of a ResNet-50 model

By controlling the size of the sub-graph the developer is able to take full advantage of the large In-Processor Memory. Unlike a GPU which must constantly shuffle data backwards and forwards from its connected memory, the IPU is able to hold large amounts of state inside the processor and this approach dramatically reduces the external memory bandwidth requirements allowing IPUs to take advantage of much larger capacity DDR-DIMM modules. Graphcore CTO Simon Knowles provided a clear description of the advantages of the IPU's In-Processor Memory during his talk at the Hot Chips Conference 2021.

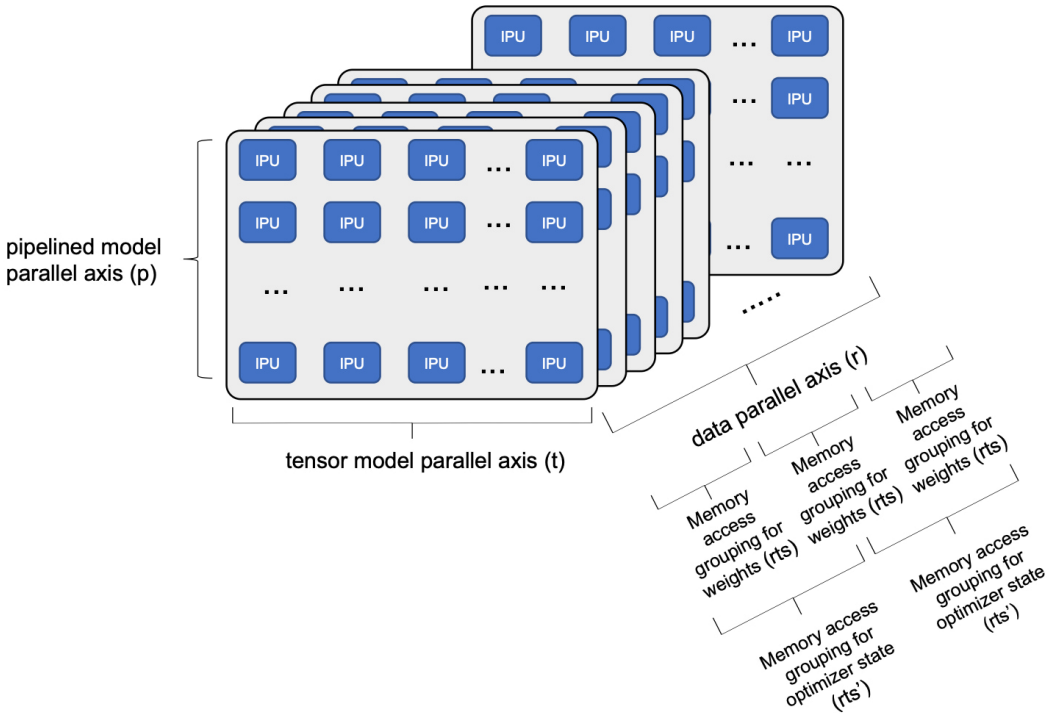
Phased Execution effectively provides an additional dimension of flexibility for mapping large models to arrays of IPU processors.

LARGE MODEL MAPPING

IPU-POD$_{128}$ and IPU-POD$_{256}$ systems provide a rich set of resources that allow large models with very large parameter counts, large activation state and complex optimiser state, all to be supported. The extensive set of software features in Poplar SDK allow a developer to map large models to these powerful computing platforms.

Models can be decomposed across a large number of IPUs to support fast training (see diagram below). It is possible to pipeline the stages of a deep learning model across IPUs connected in the p-axis. It is also possible to shard the model tensors across multiple IPUs connected in the t-axis. The training data can then be split into conventional data-parallel paths that are also commonly used on GPUs. This is the data replica dimension, the r-axis. Associated with the data replica dimension we can use streaming memory to access the model weight data. The weights for the whole model can be split amongst the replicas for more efficient use of memory and then only replicated for each phase of execution. We call this replicated tensor sharding (rts). The

optimiser state (momentum terms etc.) is accessed less often than the main weights so we may spread these over a different number of replicas than the weights (rts'). In addition to this parallel execution on a sub-graph, we can implement phased execution across the layers of the model, optimising the size of the sub-graphs, to achieve high levels of compute with all the state required for the sub-graph computation held inside the large IPU In-Processor Memory.



Model decomposition across an IPU-POD

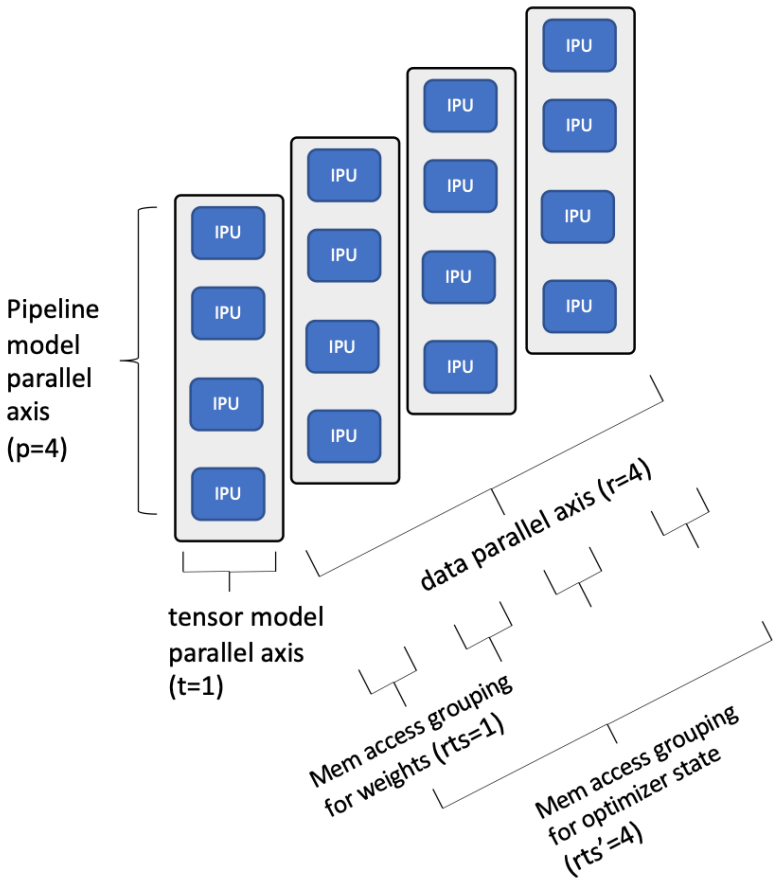
The total number of IPUs in any specific model decomposition is calculated as t*p*r. By varying the level of model pipelining, tensor sharding and the data parallel dimension, it is possible to build large models that efficiently use all the available compute and memory resources in an IPU-POD$_{128}$ or IPU-POD$_{256}$ system.

Each IPU will use a proportion of weights given by 1/ (t * p) but the proportion of weights stored in each IPU In-Processor Memory can be calculated as 1 / (t * p * rts) given replicated tensor sharding. Phased execution can then be used to optimise the proportion of weights that are available in each sub-graph and to optimise for the best use of the IPU In-Processor Memory as well as to maximise the compute efficiency.

The global batch size is calculated as b * r where b is the replica batch size. The developer can select an appropriate replica batch size to achieve the desired global batch size for their model.

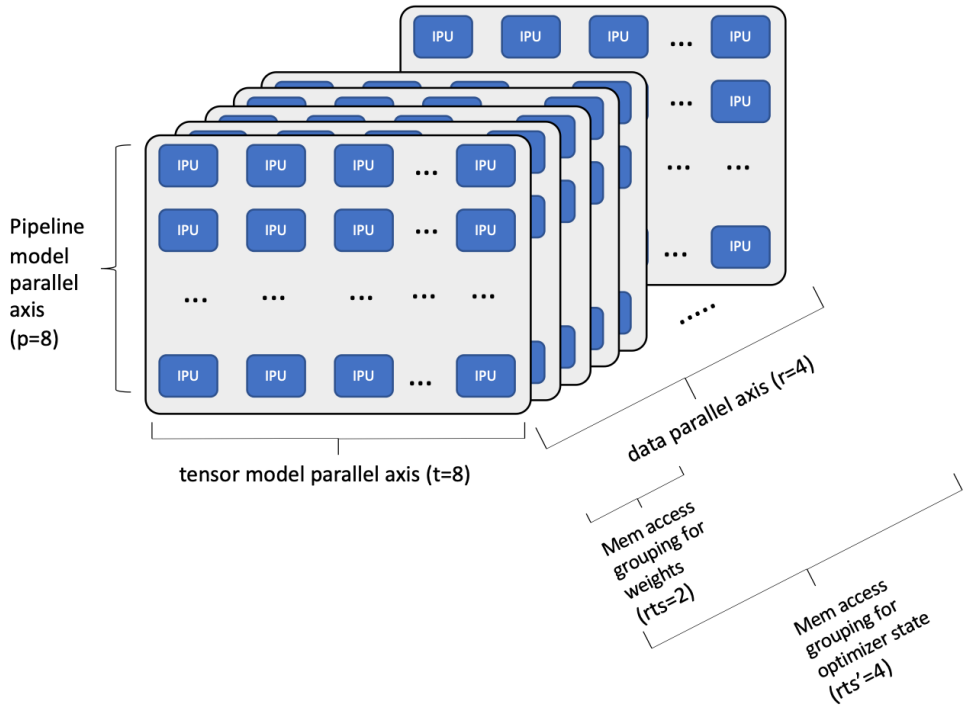
Total number of IPUs	t * p * r
Copies of model weights	r / rts
Copies of optimizer state	r / rts'
Approx proportion of weights used per IPU	1 / (t * p)
Operations over axis t	- In general: allgather, reducescatter - For transformer all {reducescatter, allgather} pairs can be implemented as allreduce -Executed every layer
Operations over axis p	-All-2-all, usually pass to neighbour. -Execute every N layers for N layers in a pipeline stage
Operations over axis rts	-allgather, reducescatter -Executed every layer
Operations over axis rts'	None
Operations over axis r/rts	-Allreduce -Executed every fwd/bwd pass * gradient accumulation
Global batch	b * r where b is replica batch size

To understand this better we can look at a simple example of a BERT-Large model mapped to a smaller IPU-POD$_{16}$ system:



In this example we have chosen to pipeline the model over 4 IPU processors and not to shard the tensors. A data parallel dimension of 4 was chosen to maximise the use of all 16 IPU processors in the IPU-POD$_{16}$ system. If we expand the pipeline depth to a dimension of 8, we can fit the whole model including all the optimiser state inside the IPU In-Processor Memory on this single IPU-POD$_{16}$ system and can achieve class leading training performance and fine-tuning results as demonstrated by our MLPerf performance results.

A more complex example is shown below, with a 175Bn parameter GPT-3 NLP model mapped over an IPU-POD$_{256}$ system.



Example of 175Bn parameter GPT-3 mapped to IPU-POD$_{256}$

In this example, we have a much larger model that is now mapped over a single IPU-POD$_{256}$ system. We have chosen a pipeline model depth of 8 in this case, and we have sharded the tensors across 8 IPUs for each pipeline stage. This gives a total of 64 IPUs for each data replica in the data parallel axis and with the data parallel axis width set to 4 we can use all 256 IPU processors in the system. With a smaller GPT class model we can fit all the weights inside the IPU In-Processor Memory but with 175Bn parameters we take advantage of Phased Execution to split this larger model into a small number of sub-graphs. With 96 layers in a GPT-3 model we can still use all the available In-Processor Memory and all the compute resources efficiently. By expanding to more sub-graphs implemented using Phased execution we can support even larger models with trillions of parameters. To further reduce training time, we can also scale across multiple, network-connected IPU-POD$_{256}$ systems.

SUMMARY

We can see that IPU-POD systems can easily support today's large AI models, including GPT and beyond. IPU-POD systems can scale to support the largest models. However, the extremely rich In-Processor Memory and much more highly parallel IPU processor opens the opportunity to explore a much richer set of next generation large models. Interesting areas of research are already being investigated on IPUs around sparse models and Graph Neural Networks. We are looking forward to seeing the incredible new breakthroughs that innovators will be able to achieve using the advanced IPU-POD$_{128}$ and IPU-POD$_{256}$ systems.

GET STARTED WITH THE IPU

Learn more about how Graphcore can accelerate your next machine intelligence project.

TALK TO US

Get Started >