Optimising the data parallelism of ECMWFs AIFS model

Cathal O'Brien, ECMWF

Introduction

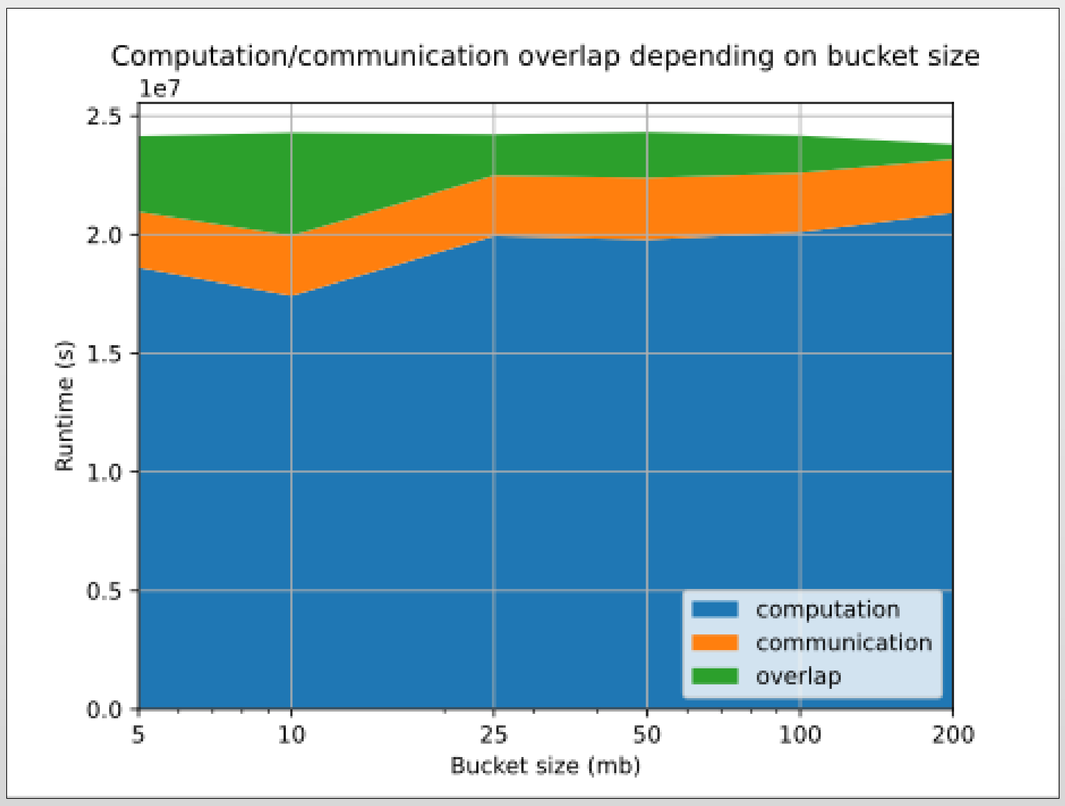
Data parallelism is a popular way to scale deep learning models across many GPUs. The model is replicated across GPUs and each model receives a unique chunk of the dataset. Model gradients are combined during the backward pass using all-reduce communication operations.
This poster presents work done optimizing the data parallelism of ECMWFs AIFS model. Additionally, large-scale runs on up to thousands of GPUs are shown on Leonardo and MareNostrum5.

Finetuning PyTorch.distributed

AIFS uses the "PyTorch distributed" package to scale across multiple nodes. This has a number of presets which can be tweaked depending on the machine and model you're using.

Bucket size

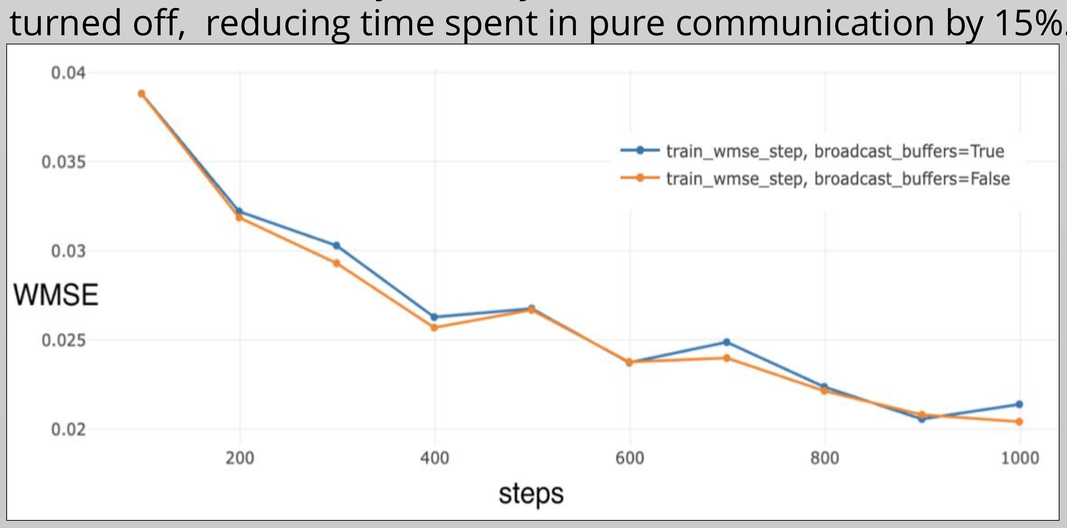
As gradients are computed, they are stored in buckets of a given size (default 25mb). Filled buckets are synchronized across models with an all-reduce operation. This all-reduce happens asynchronously, leading to overlap of communication and gradient computation.
If network latency is a bottleneck, bucket size can be increased. This will decrease the number of all-reduction operations, at the cost of reducing the amount of overlapped computation and communication.



20 training steps of an 8 node distributed run on Leonardo (GNN model, n320 res, 1 model per node, 1024 channels per model), with different bucket sizes.

Broadcast buffers

By default, PyTorch distributed will synchronize buffers across all ranks by broadcasting the buffers of rank 0 at the beginning of each step. Certain layers like BatchNorm require this synchronization. AIFS does not have any such layers so BroadcastBuffers can be turned off, reducing time spent in pure communication by 15%.



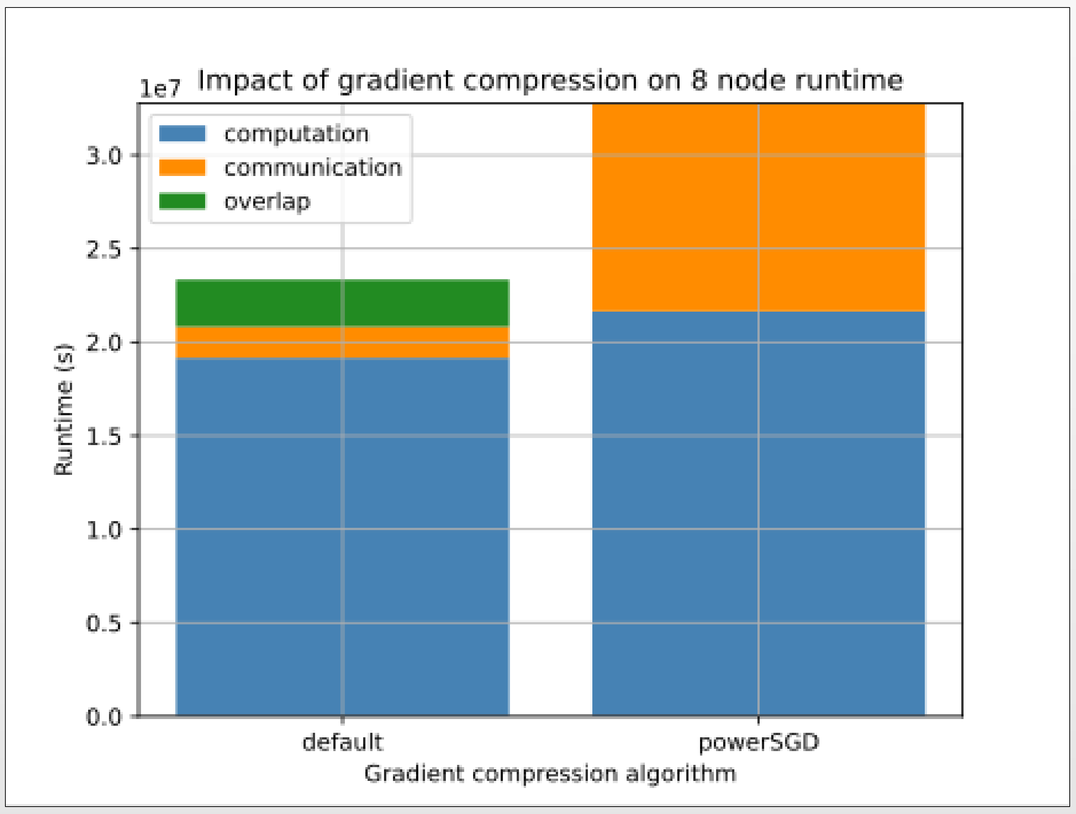
WMSE error over 1000 training steps of an 8 node training run on Leonardo (GNN model, n320 res, 1 model per node, 1024 channels per model, same seed). WMSE is almost equal, suggesting broadcasting the buffers is unnecessary.

Gradient Compression

If network bandwidth is a bottleneck, "PyTorch distributed" offers gradient compression to alleviate this. The PowerSGD algorithm offers a 100x compression factor.
PowerSGD led to a 40% increase in runtime for an 8 node AIFS training run, due to:

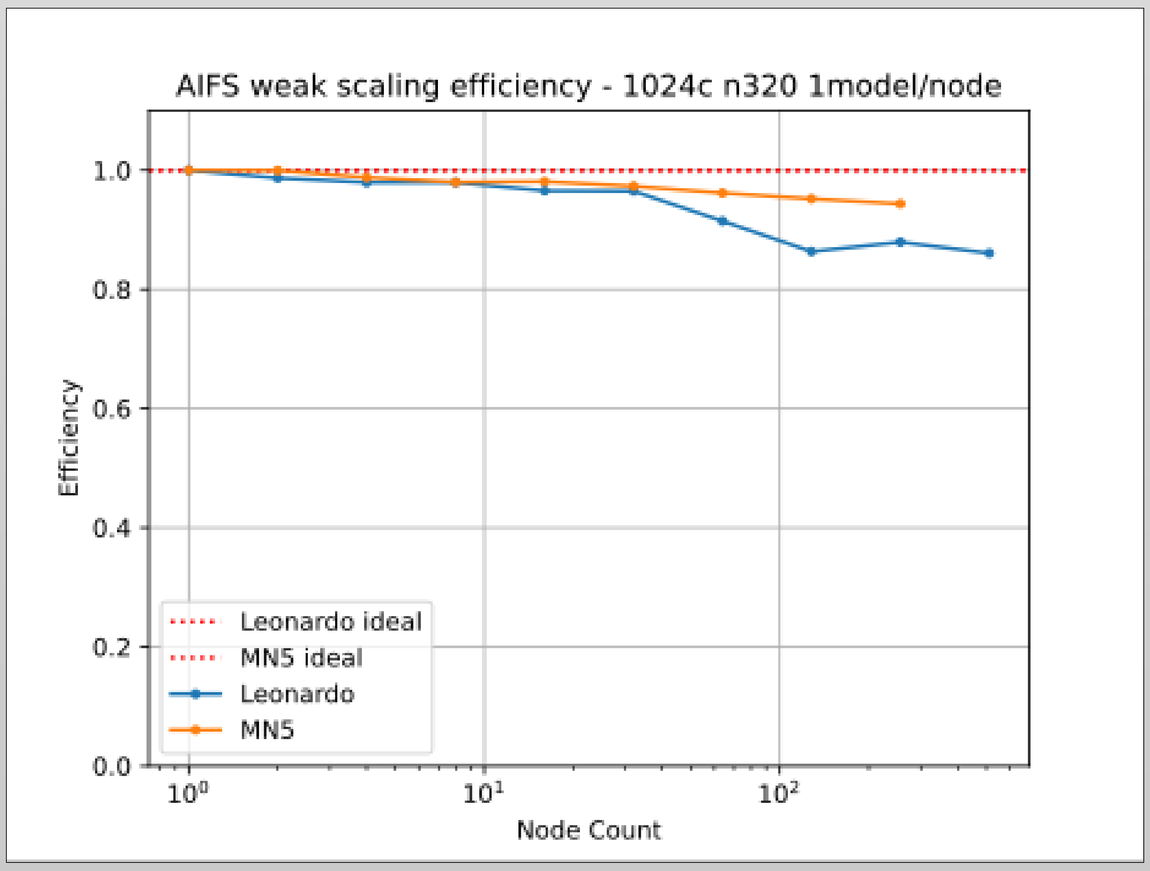
1. PowerSGD requiring 2x the number of all-reduce calls.
2. Due to algorithmic constraints, overlapping is not possible
3. and the more efficient ring all-reduce algorithm can't be used.

(Zhang et al, 2023) suggests a modified PowerSGD algorithm to alleviate these issues, but this is not integrated into PyTorch.



Scaling AIFS to 100s of nodes

Weak scaling results for AIFS on Leonardo and MareNostrum5 (MN5) on up to 512 nodes (4 GPUs per node) . Plotting efficiency ($E = \frac{T_1}{T_N}$) instead of runtime allows us to compare differences at the interconnect level, rather than at a single node level.
MN5 begins to scale better than Leonardo after 32 nodes. This is likely because MN5 has an 800Gbit/s interconnect compared to the 400Gbit/s interconnect of Leonardo.



AIFS weak scaling efficiency on Leonardo and MareNostrum5 (N320, GNN backend, 1024 channels per model, 1 model per node, 200 training steps, 100 validation steps, 3 epochs per run, median of 3 runs). Dataset was replicated across models.

Acknowledgements

Thanks to Ana Prieto Nemesio (ECMWF) for her work on the AIFS profiler which was used to collect these results