Model parallelism with Pytorch

Gyula Ujlaki

DKF Kft., NCC Hungary – HPC Scientific expert

EURO²

DKF

Use cases for model parallelism

Calculating model memory requirements:

Inference:

num_para * precision / 8bit = mem_req
mem_req * 1.2 = inference_mem_req

Training:

optim_states $_{AdamW}$ = 4xmem_req
optim_states $_{SGD}$ = 1.5x mem_req

gradients = mem_req

activations = sbhL
activations = sbhL(10 + 24/t)
activations = sbhL(10 + 24/t + (5a*s)/h*t)

s – sequence length in tokens
b – batch size per GPU
h – dimension of the hidden size within each transformer layer
L – the number of layers in the transformer model
a – the number of attention heads in the transformer model
t – degree of tensor parallelism being used (1 if not)
No sequence parallelism being used
No activations stored in fp16

Source: https://blog.eleuther.ai/transformer-math/

Training memory requirements of Llama 70B + AdamW optimizer, without calculating sequence lengths, batch size, etc.:

~70 * 10^9 parameters, fp16 precision

70 * 10^9 *2 (bytes) = 140 * 10^9 (model memory requirements)
140 * 10^9 * 4 (bytes) = 560 * 10^9 (with AdamW optimizer states)
560 * 10^9 + 140 * 10^9 = 700 * 10^9 bytes (with gradients)

Activation size depends on a lot of factors, thus we will not approximate this size. Even without that:

700 / 1024^3 = 652 GB (memory requirements) * 1.2 (for overhead)

Officially 500 GB: https://huggingface.co/blog/llama31
Depends on specific implementations

EURO²

DKF

Pytorch - Layer-wise model parallelism

a. If only 1 GPU is available: CPU offloading, a few layers run on CPU
b. If at least 2 GPU available: divide layers evenly between the two GPUs.

Only 1 model, running on multiple devices, increasing/multiplying the memory available for the model!
Slower than 1 GPU, because of the communication overhead between GPUs.

Model can be distributed even among nodes as well, with Remote Procedure Calls: https://docs.pytorch.org/docs/stable/rpc.html

Cases when implementing an LWMP strategy is useful:
c. Model size is slightly larger than GPU memory.
d. If pipeline parallel logic implemented – no exercise for this part, as this is significantly more difficult to implement, than LWMP
e. Large language models cannot fit into a single GPU, and e.g. Tensor Parallelism also used for in-layer distribution.

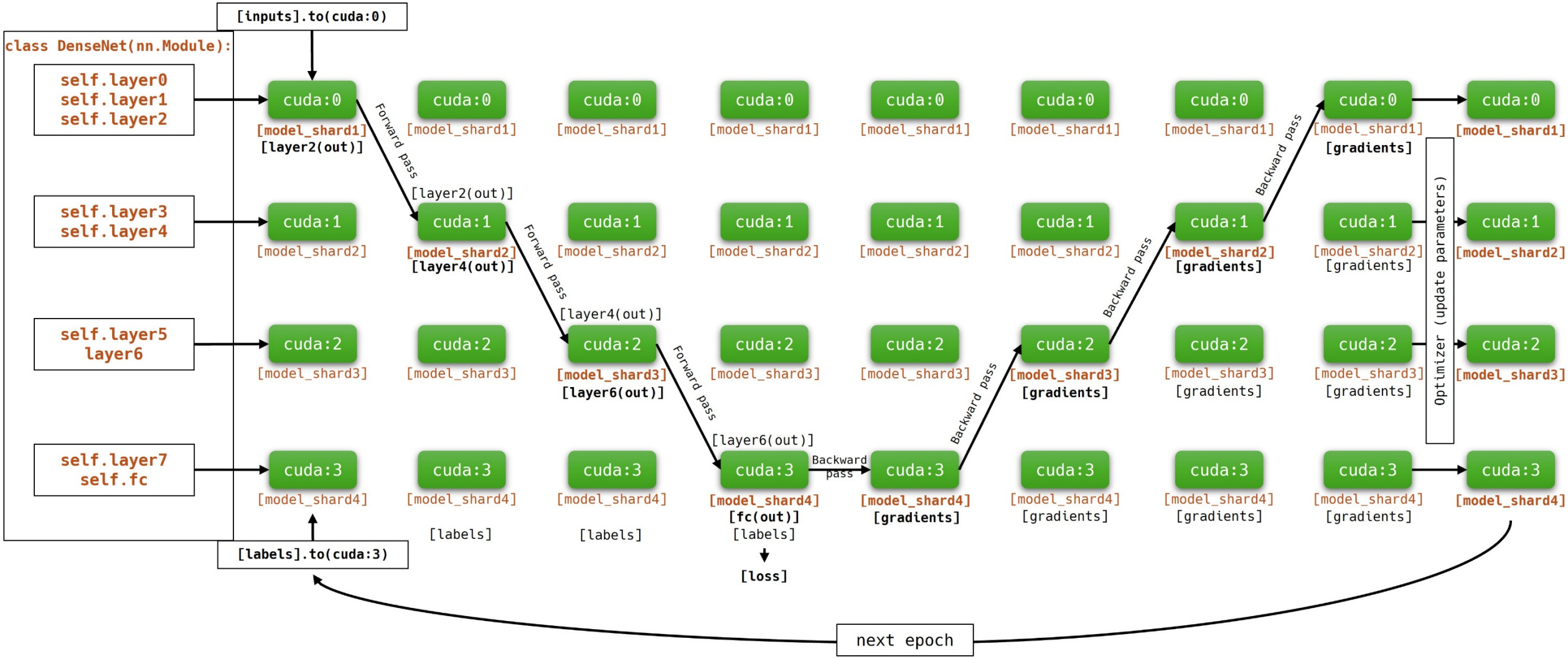
LWMP can be combined with other parallelism techniques as well:
- Distributed Data Parallel – DDP → inefficiency upscaled
- Pipeline parallelism

etc.

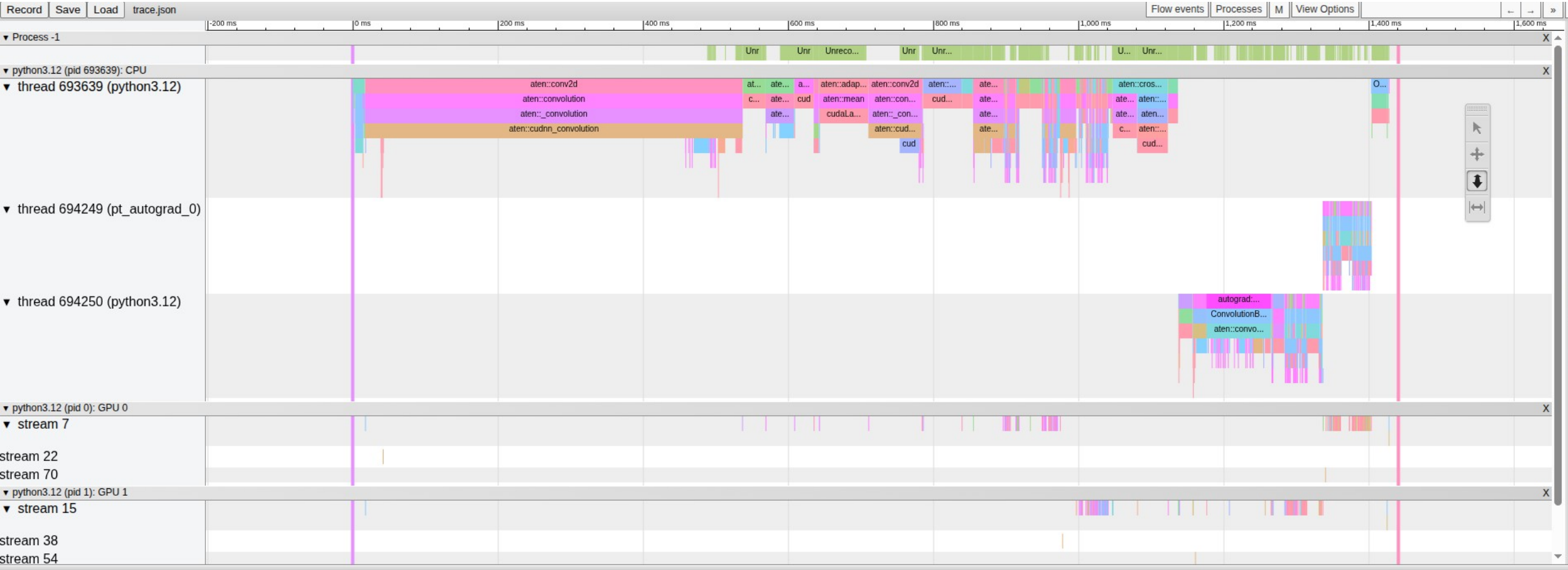
EURO²

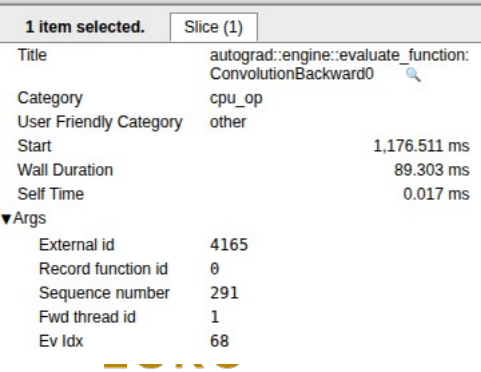
DKF

Layer-wise model parallelism



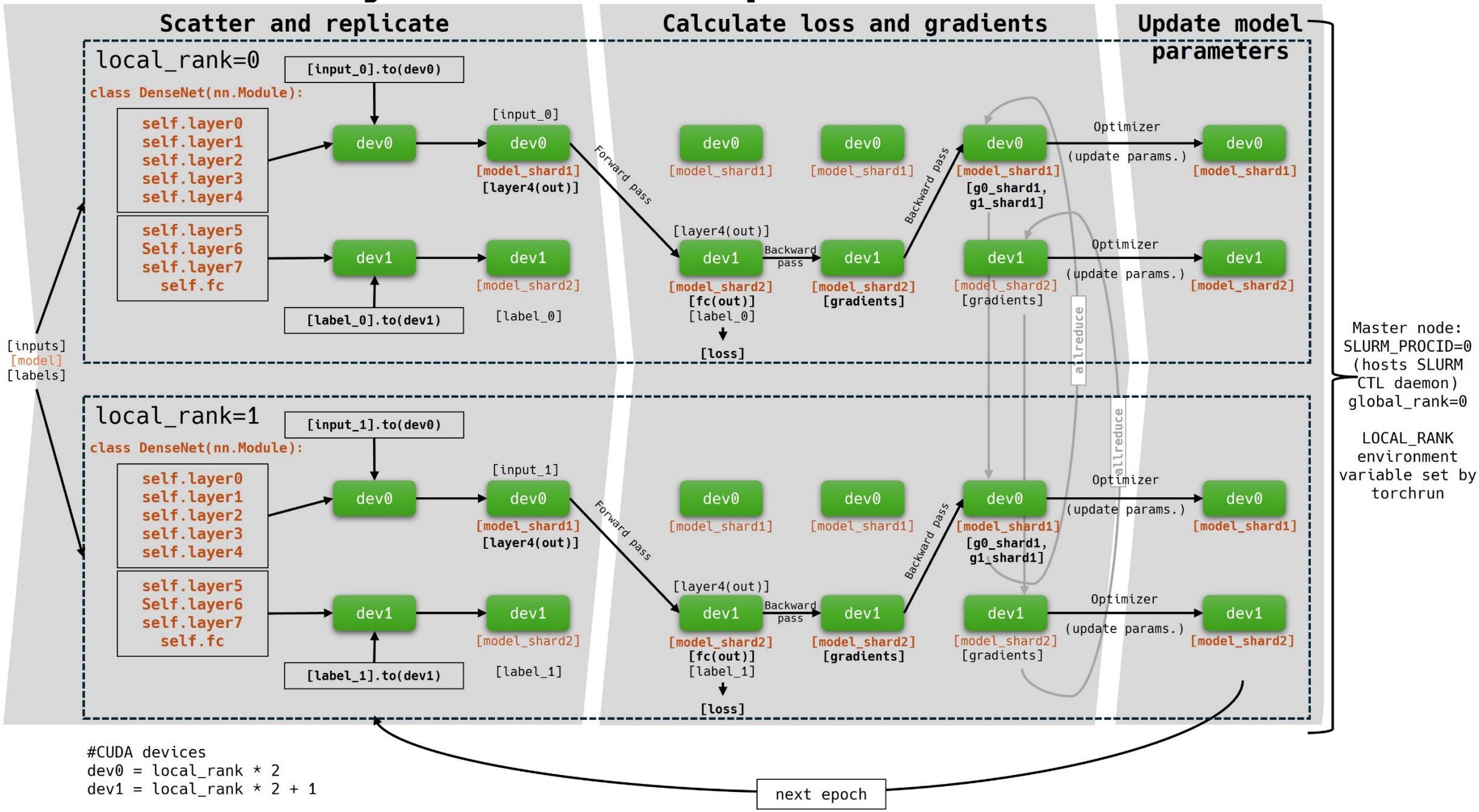
Tracing operations and tensors during training





ssh -L 16001:localhost:16001 youruser@login.Leonardo.cineca.it
tensorboard --logdir=/home/user/log/path --port 16001 --host localhost
(everyone has to use a unique port number)
Browser: localhost:16001

Layer-wise model parallelism + DDP



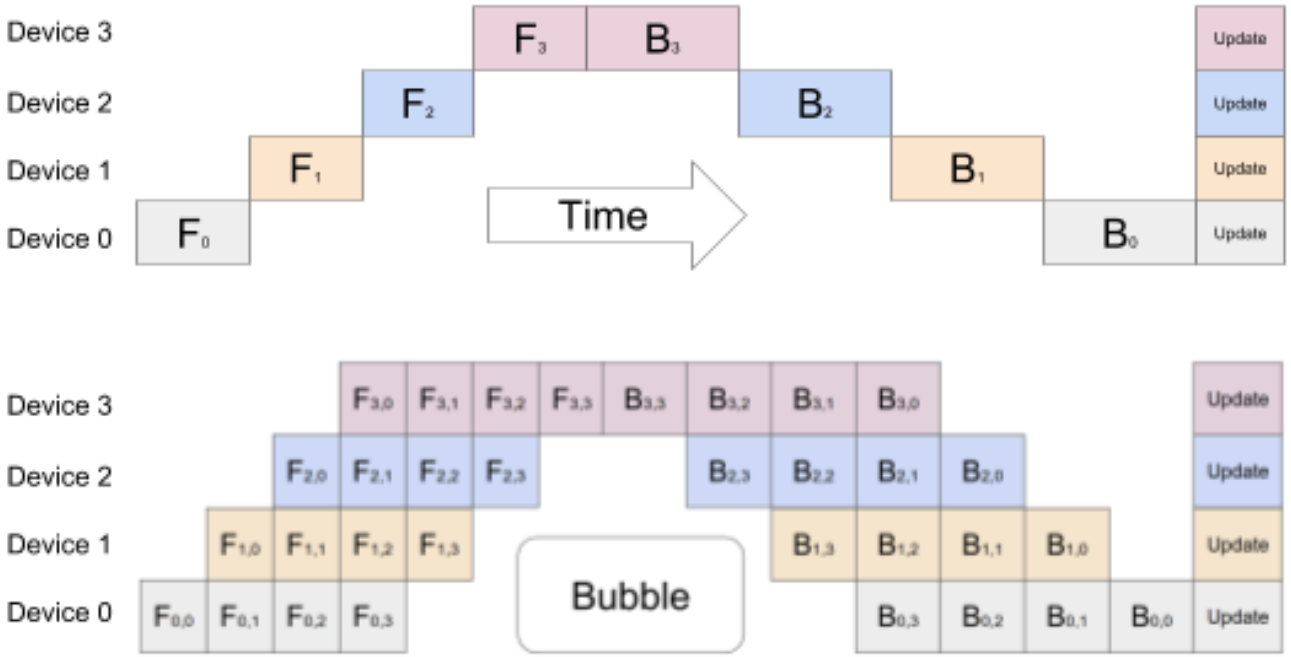
Basic model parallel techniques with Pytorch

Large amount of data	Too large model for 1 GPU	Both
No parallelism, infinite time	Layer-wise model parallel, or naïve model parallel	FSDP – Fully Sharded Data Parallel
DP – Data Parallel	TP - Tensor parallel	DDP + PP or TP
DDP – Distributed Data Parallel	PP - Pipeline parallel	+ any other combination of methods

EURO²

DKF

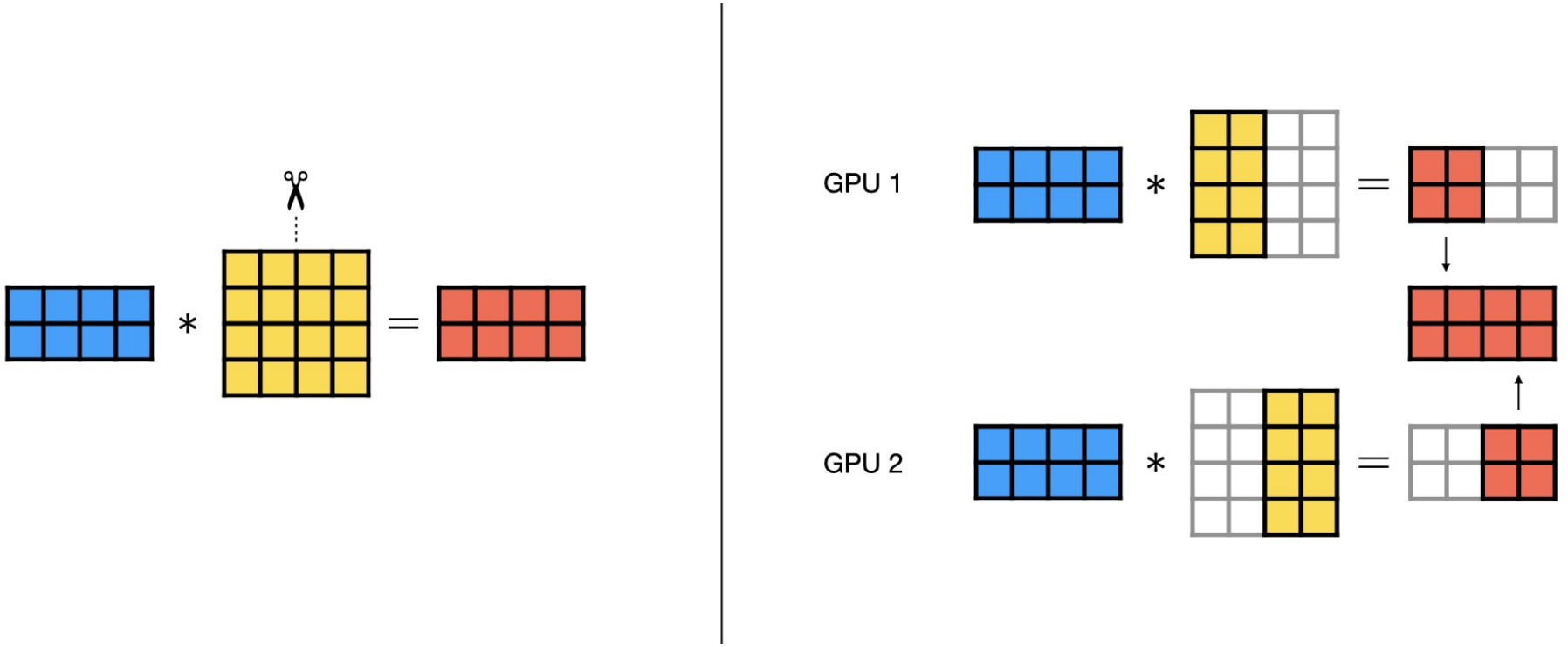
Pipeline Parallelism



Top: The naive model parallelism strategy leads to severe underutilization due to the sequential nature of the network. Only one accelerator is active at a time. Bottom: GPipe divides the input mini-batch into smaller micro-batches, enabling different accelerators to work on separate micro-batches at the same time.

https://research.google/blog/introducing-gpipe-an-open-source-library-for-efficiently-training-large-scale-neural-network-models/

Tensor Parallelism in principle



https://lightning.ai/docs/pytorch/stable/advanced/model_parallel/tp.html

Thank you!