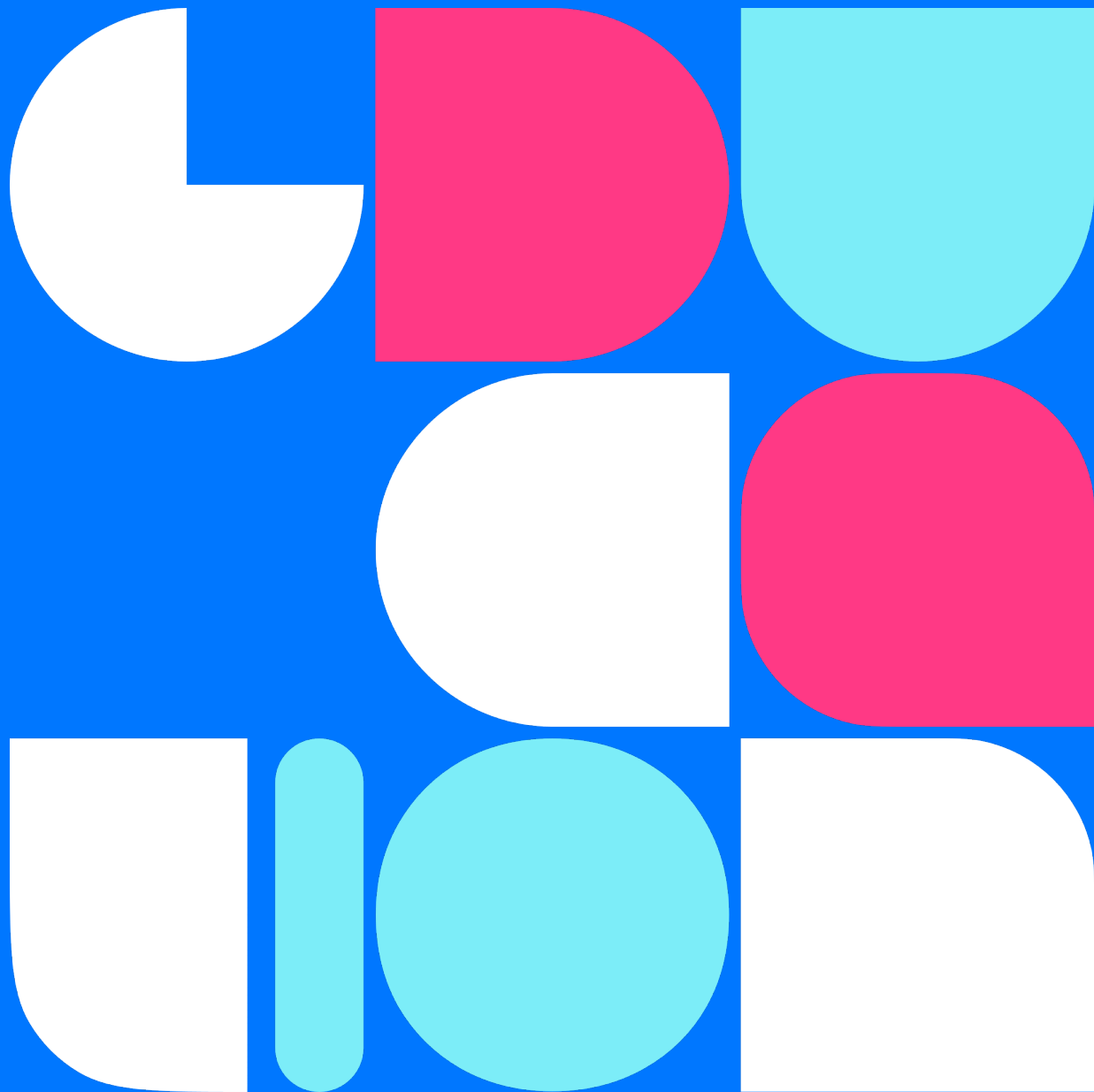
education

Dealing with LARGE models

Егор Спирин



Проблема 🔍

7B модель уже не редкость, но

👉 7B параметров, каждый float32 $\Rightarrow \frac{7 \cdot 10^9 \cdot 4}{1024^3} \approx 26$ GB памяти

👉 А еще надо хранить градиенты — 26 GB

👉 Adam хранит 1 и 2 момент — $26 \cdot 2 = 52$ GB памяти

Проблема 🔍

7B модель уже не редкость, но

👉 7B параметров, каждый float32 $\Rightarrow \frac{7 \cdot 10^9 \cdot 4}{1024^3} \approx 26$ GB памяти

👉 А еще надо хранить градиенты — 26 GB

👉 Adam хранит 1 и 2 момент — $26 \cdot 2 = 52$ GB памяти

Уже надо 104 GB видеопамяти, при этом еще необходимо выделить под активации, которые зависят от длины последовательности и батча, а обучать надо на триллионах токенов...

Проблема 🔍

7B модель уже не редкость, но

👉 7B параметров, каждый float32 $\Rightarrow \frac{7 \cdot 10^9 \cdot 4}{1024^3} \approx 26$ GB памяти

👉 А еще надо хранить градиенты — 26 GB

👉 Adam хранит 1 и 2 момент — $26 \cdot 2 = 52$ GB памяти

Уже надо 104 GB видеопамяти, при этом еще необходимо выделить под активации, которые зависят от длины последовательности и батча, а обучать надо на триллионах токенов...

Nvidia H100 — 80 GB

AMD Mi300X — 192 GB

LLaMA-3 — 405B

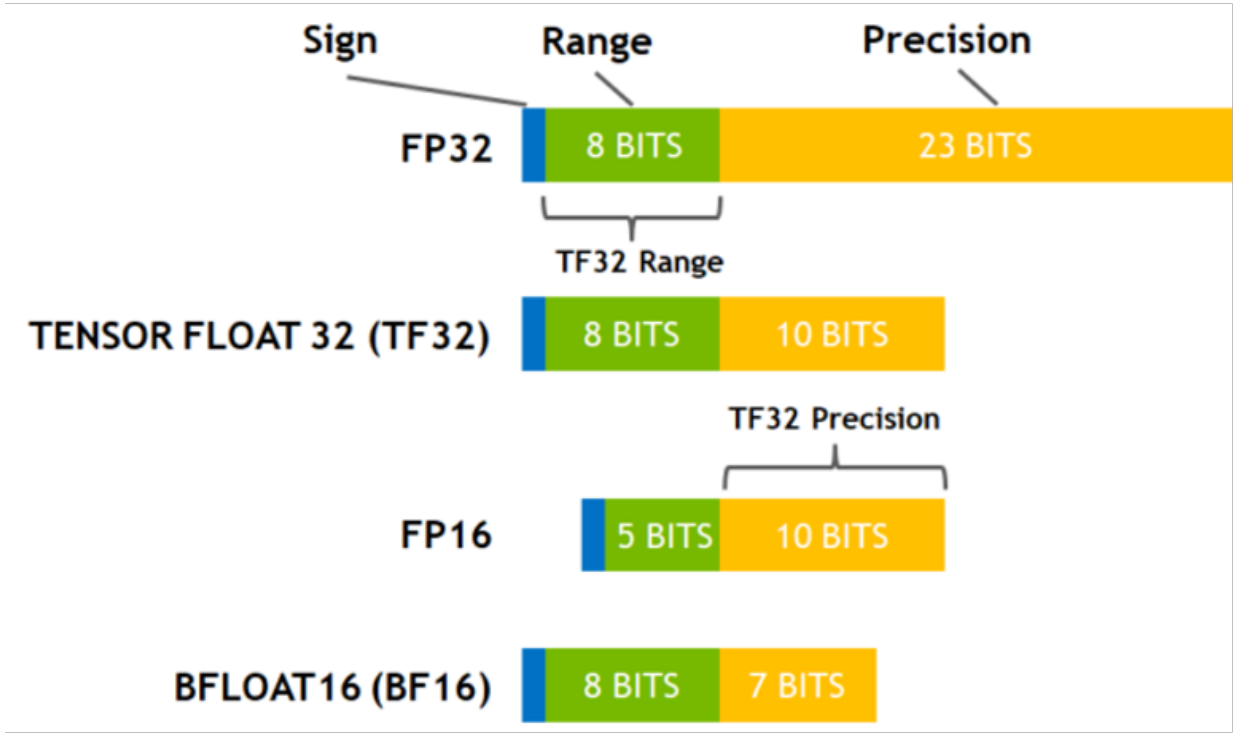
Mistral-Large — 123B

Nemotron-4 — 340B

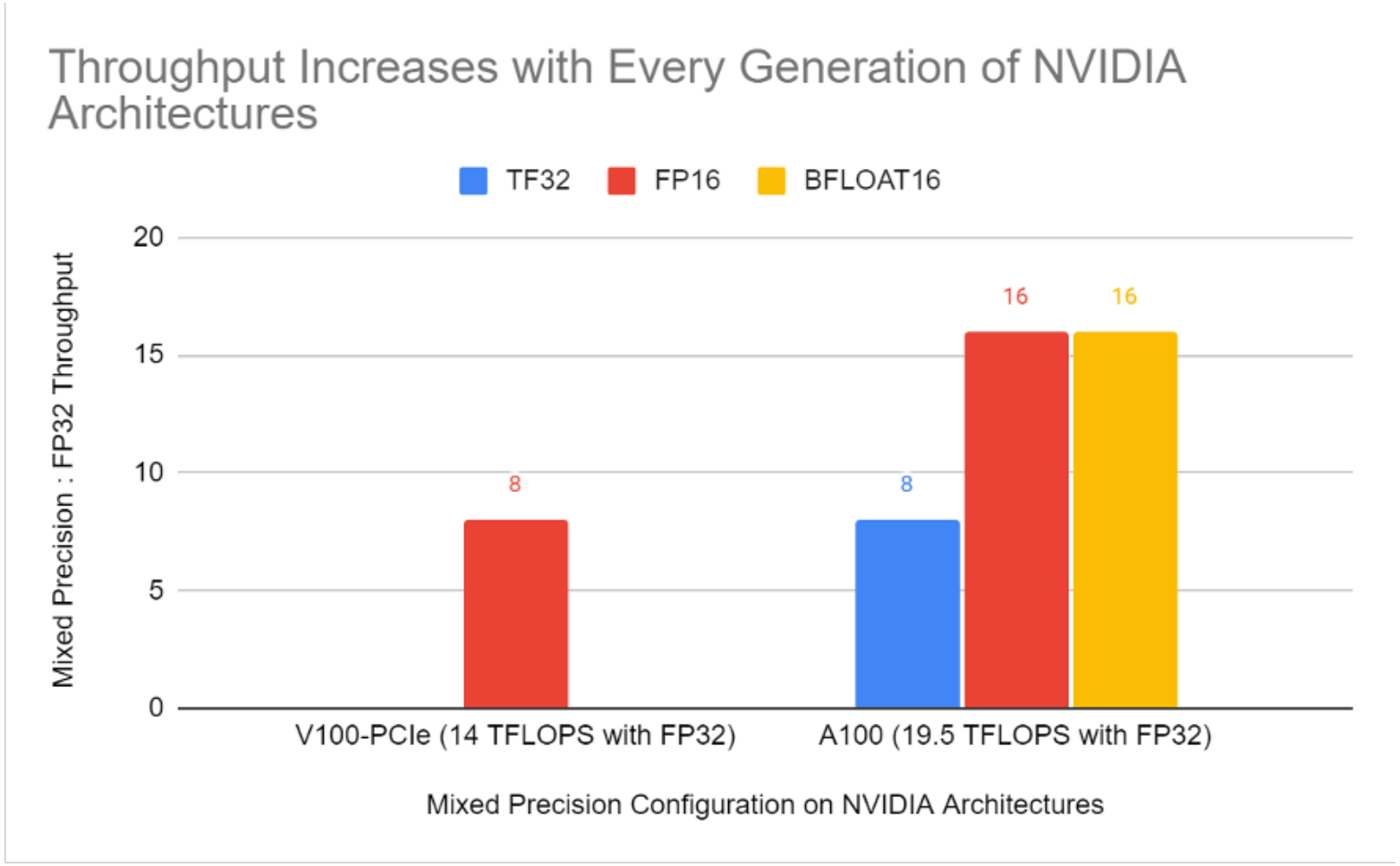
Half-precision

FP32 – использования 32 бит под запись числа

- 👉 Можно использовать и 16 бит, жертвуем макс. значениями и точностью
- 👉 **NVidia** предлагает собственные форматы чисел для более быстрой арифметики



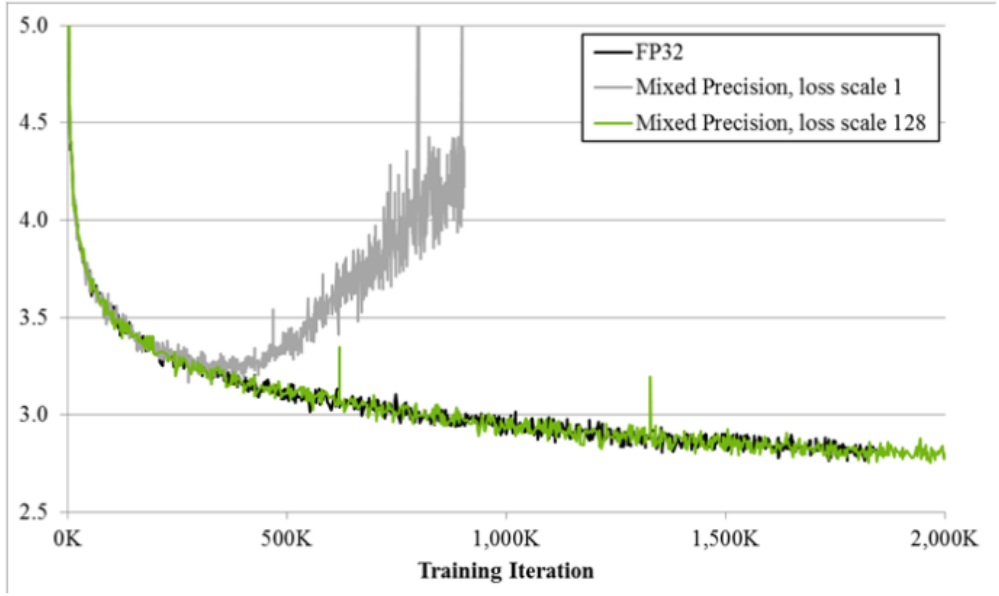
How fast is it?



Relative peak throughput of float16 (FP16) vs float32 matrix multiplications on Volta and Ampere GPUs.

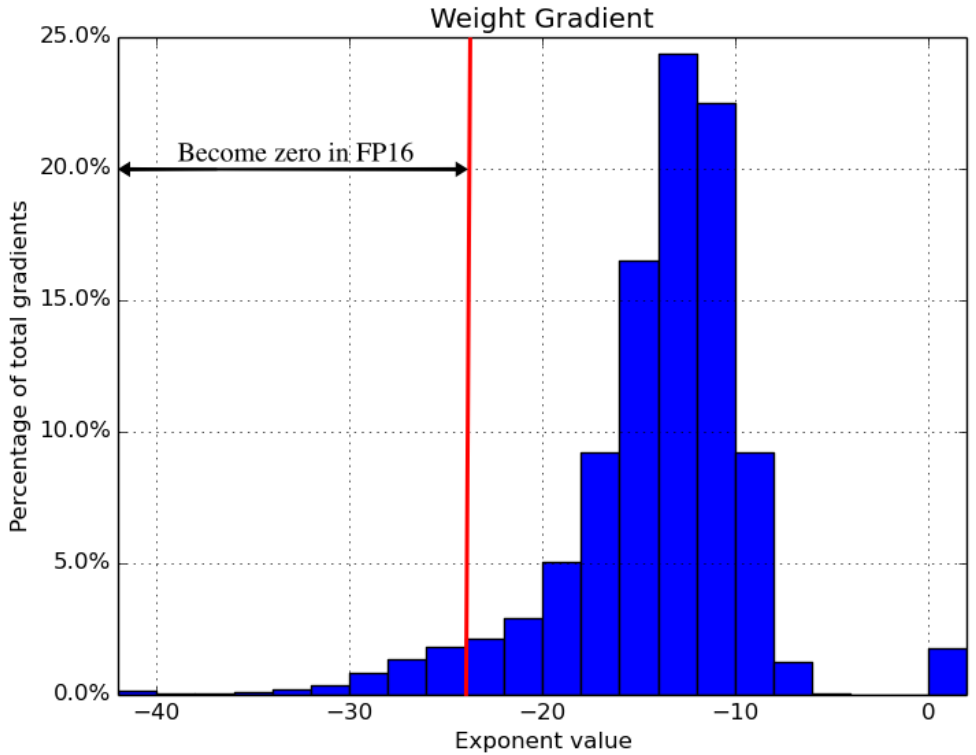
Нюанс!

Обучение в FP16 крайне нестабильно – градиенты зануляются из-за маленьких чисел



loss scaling:

1. Считаем forward в FP16
2. Умножаем loss на S
3. Считаем градиенты в FP16
4. Делим градиенты на 1/S
5. Обновляем веса



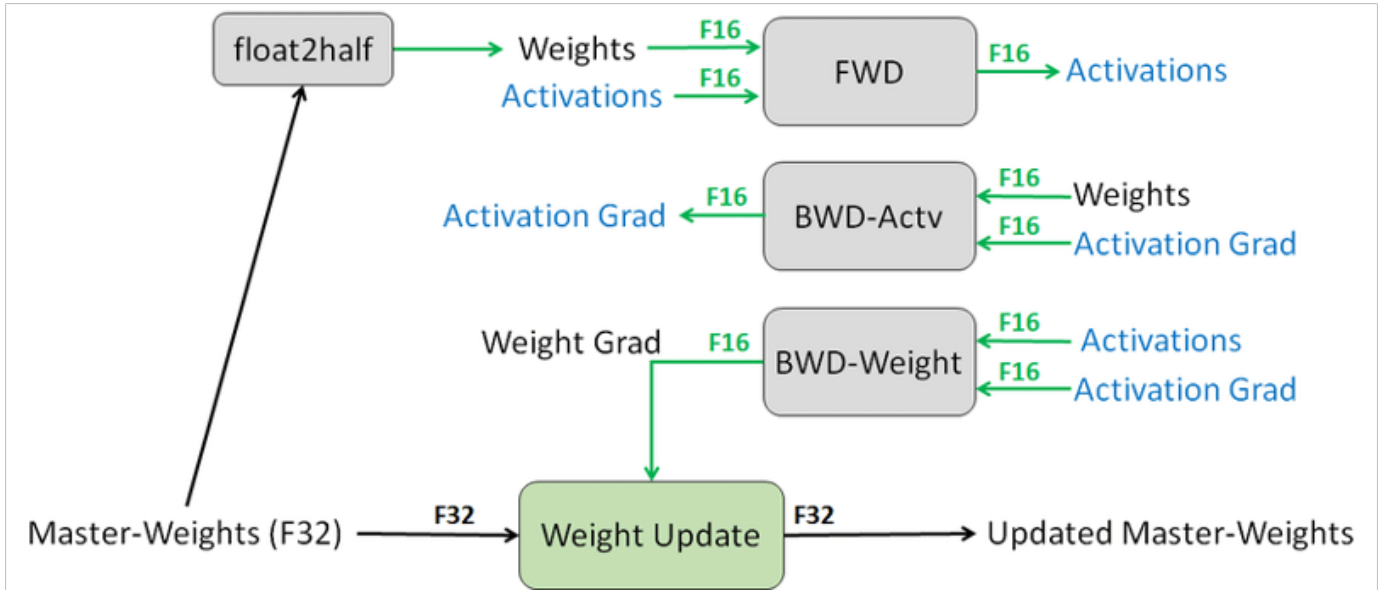
Mixed Precision

Множество операций требуют от нас большой точности: нормализации, residual connection, gradient accumulation...

Mixed Precision – используем несколько форматов записи числа

- 👉 Веса модели держим в 2 форматах
- 👉 Апкастим в определенных слоях

Быстрее работаем, но больше памяти тратим :(



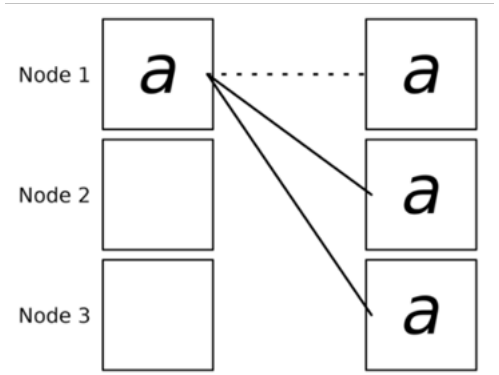
[1] — Mixed Precision Training, Narang & Micikevicius et al., ICLR'18

Training Parallelism

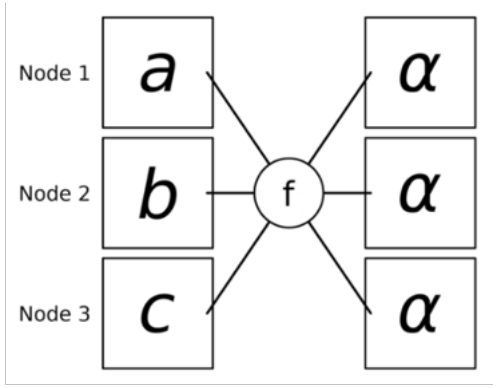
💡 Надо использовать несколько GPU

Collective Communications

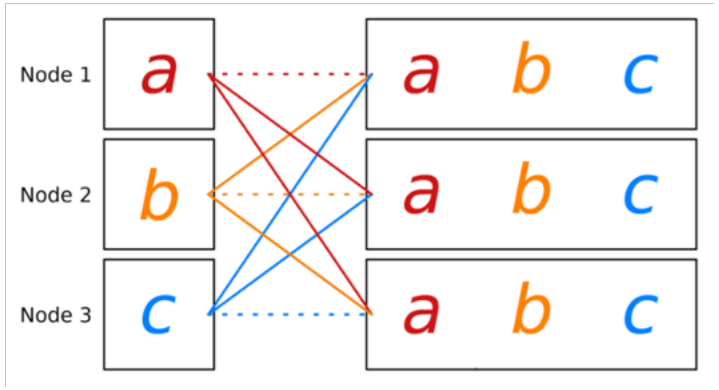
Несколько GPU в обучении ⇒ необходимо обмениваться информацией



`Broadcast`



`AllReduce`



`AllGather`

\+ стандартные примитивы синхронизации в виде барьеров

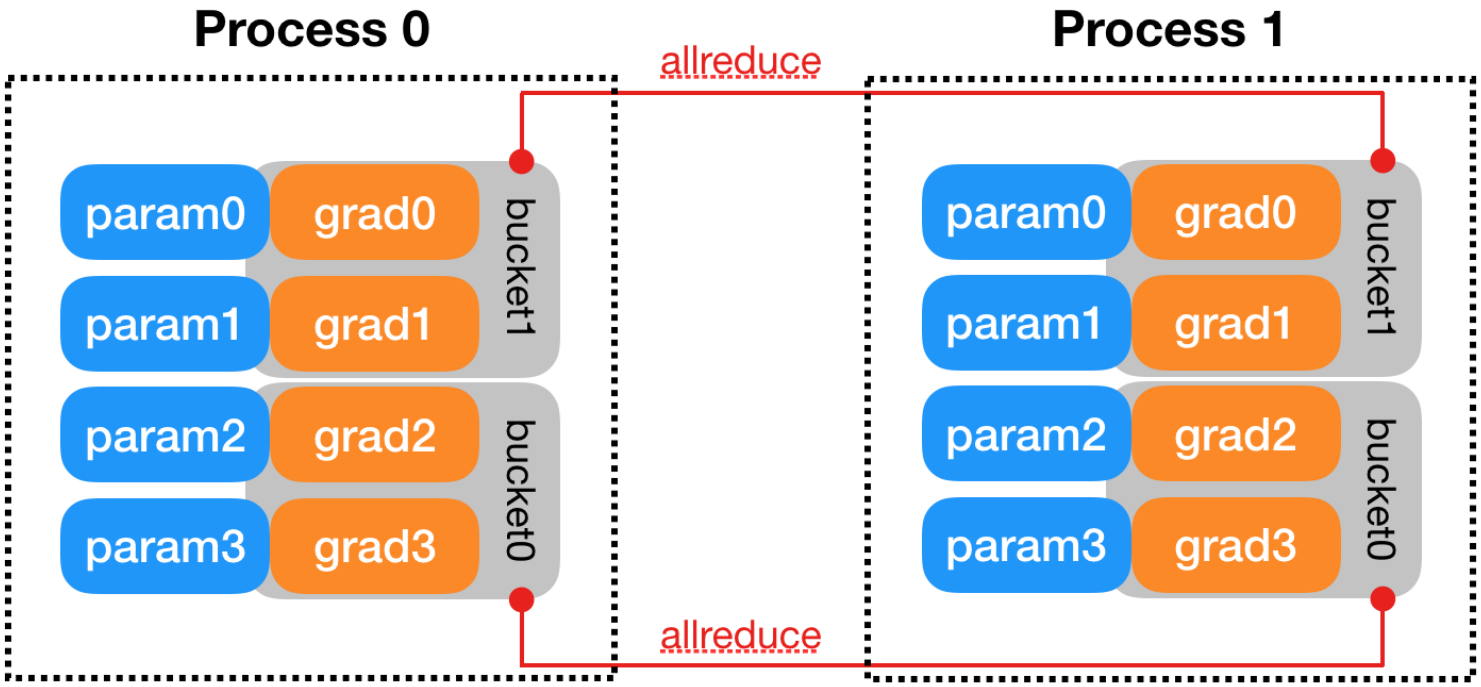
Несколько популярных реализаций: NCCL, GLO, MPI

Data Parallelism

Когда модель влазит на GPU, но необходимо увеличить размер батча

👉 Каждое устройство работает со своей частью данных

👉 После forward-backward синхронизируем градиенты и обновляем веса на каждом устройстве



Sharded DP

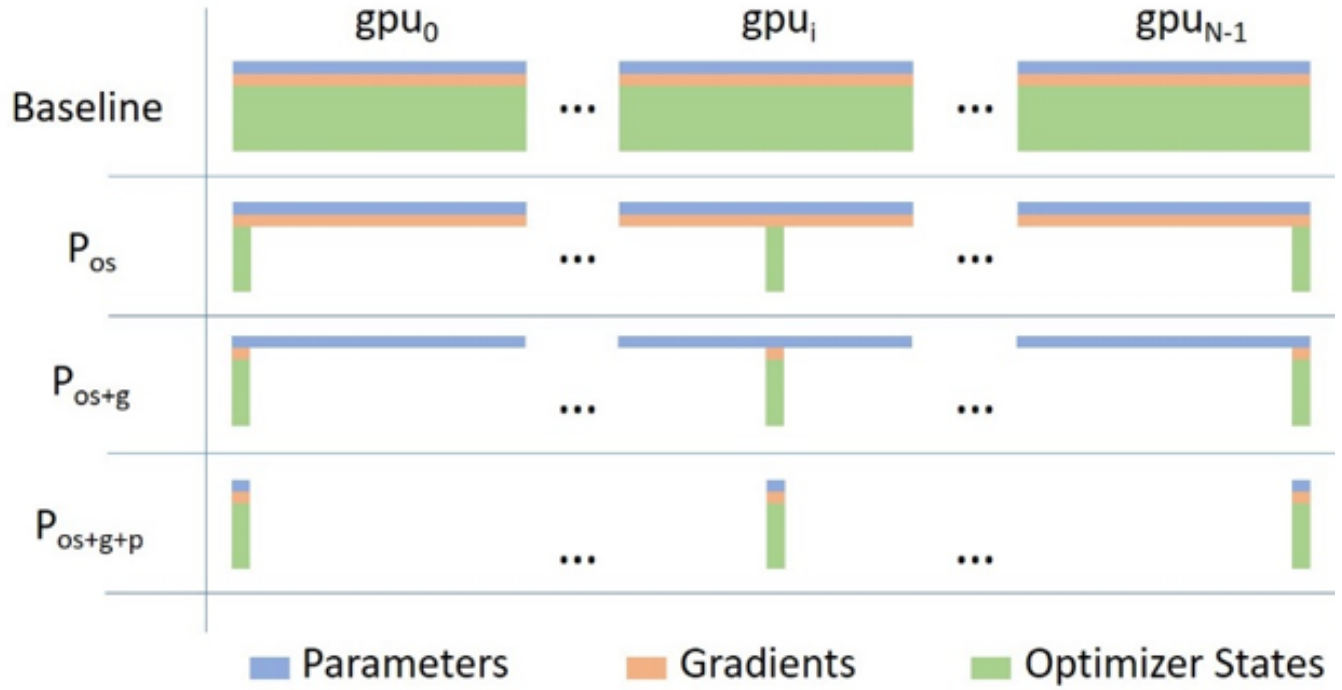
Если не хватает памяти на GPU

Шардируем:

Stage 1: состояние оптимизатора

Stage 2: + градиенты

Stage 3: + параметры модели



Горизонтальное масштабирование: во время forward-backward делаем множество `AllGather` и `AllReduce`

DeepSpeed, Torch FSDP

[2] — ZeRO: Memory Optimizations Toward Training Trillion Parameter Models, Samyam Rajbhandari, Jeff Rasley, SC'20
[3] — PyTorch FSDP: Experiences on Scaling Fully Sharded Data Parallel, Zhao et al., VLDB Endowment'23

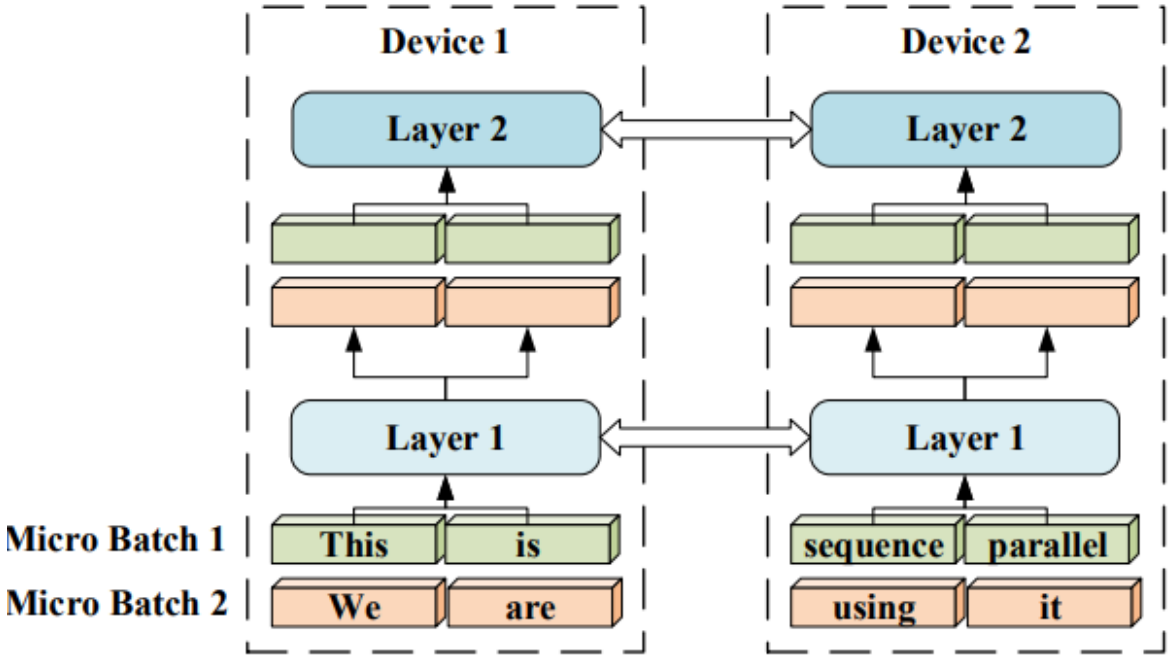
Huge Global Batch Size Issue

Больше кластер — сильнее дробление на шары ⇒ больше свободного места

Постоянно увеличивать размер батча нельзя, зато можно увеличить длину контекста!

Sequence parallelism — обрабатываем более длинные последовательности

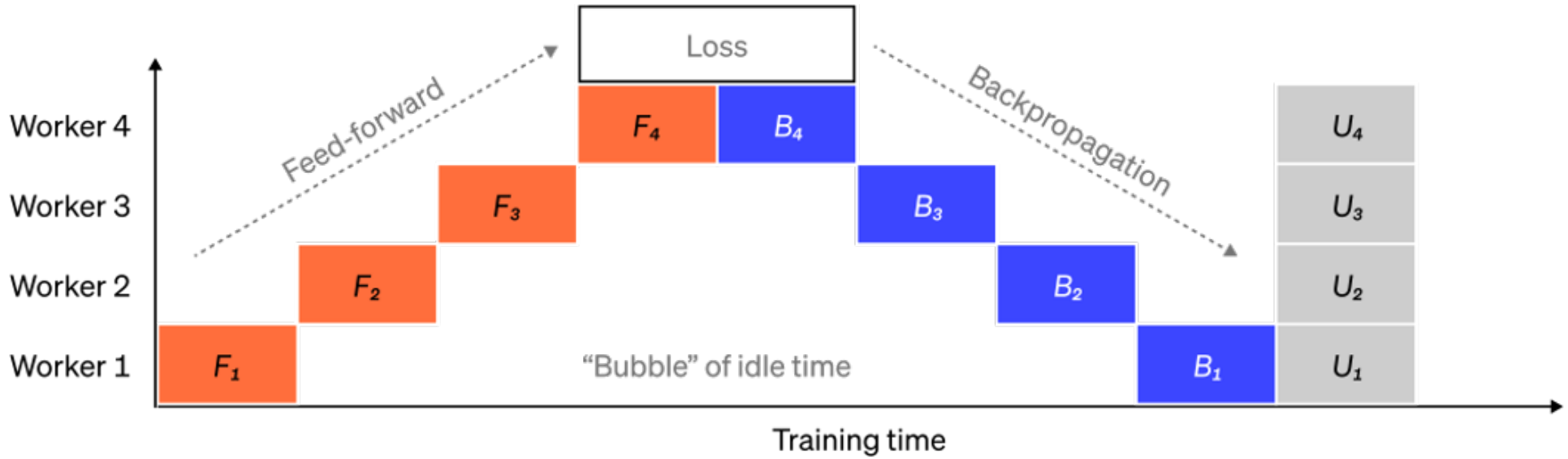
- 👉 Нативно можем параллеризовать Dropout, LayerNorm, ...
- 👉 Больше всего памяти уходит на Multi-Head Attention
- 👉 Colossal-AI, Megatron-LM, DeepSpeed-Ulysses



[4] — Sequence Parallelism: Long Sequence Training from System Perspective, Shenggui Li, and Fuzhao Xue et al., ACL'23
[5] — Reducing Activation Recomputation in Large Transformer Models, Korthikanti et al., MLSYS'23
[6] — DeepSpeed Ulysses: System Optimizations for Enabling Training of Extreme Long Sequence Transformer Models, Jacobs et al., PODC'24

Pipeline Parallelism

Модель разбивается на блоки, которые помещаются на отдельные девайсы



Вертикальное масштабирование: во время forward-backward делаем множество `Broadcast`

😥 В моменте используем только 1 GPU, остальные простаивают

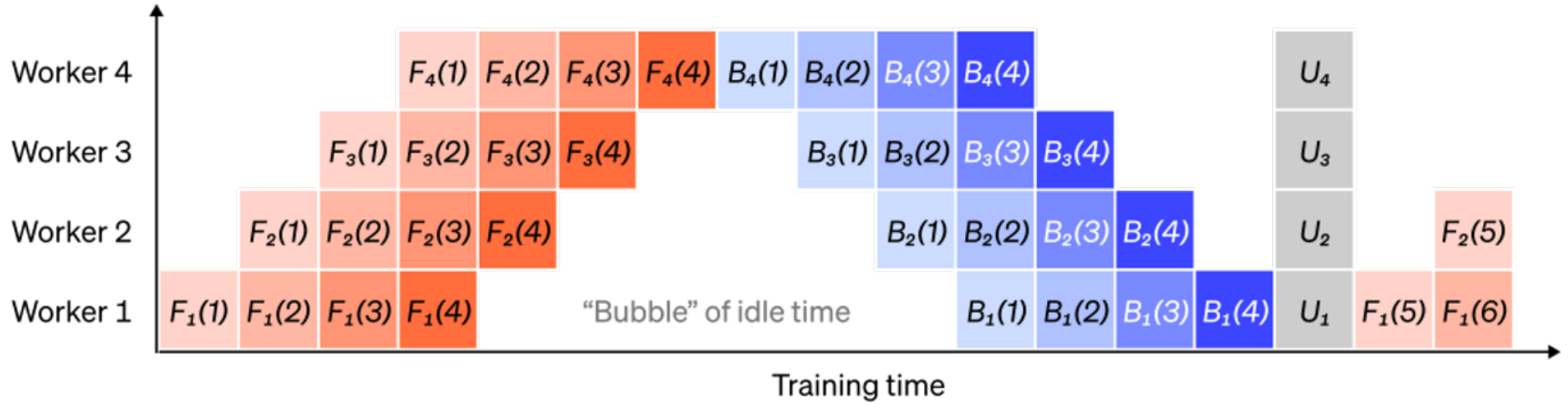
GPipe

💡 Пока ждем остальных считаем следующий микробатч (chunk) ~ аккумуляция градиентов

Теперь размер пузыря зависит от числа чанков

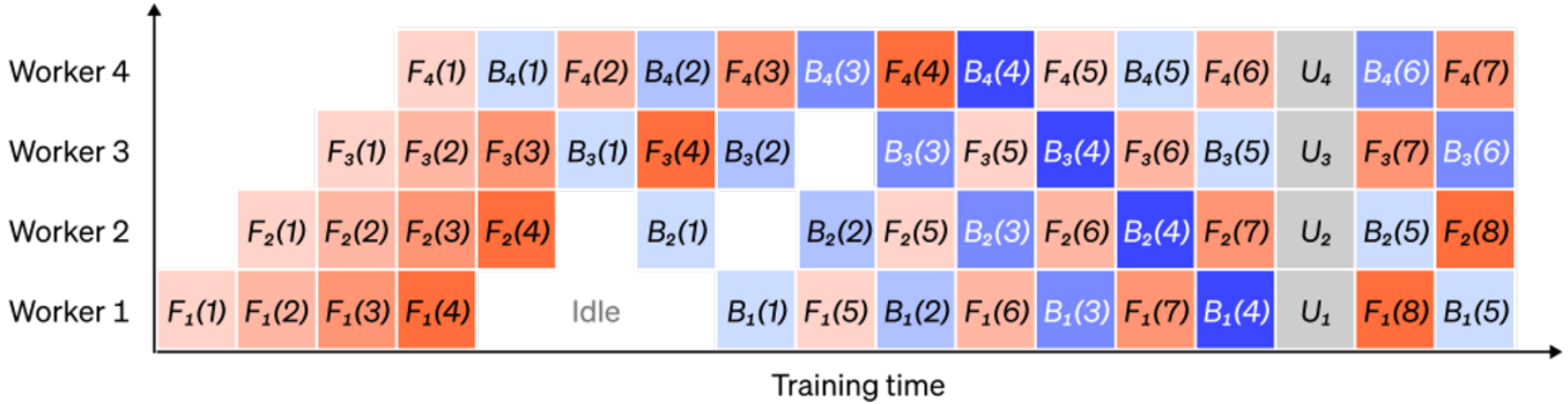
👉 Для 1: наивный PP

👉 Много: либо слишком маленькие микробатчи, либо слишком большой итоговой батч



[7] — GPipe: Efficient Training of Giant Neural Networks using Pipeline Parallelism, Huang et al., NIPS'19

PipeDream | порядок важен!



Нет единой версии модели, каждый воркер работает со своей копией

👉 Weight Stashing: храним несколько копией модели, для каждой пары forward-backward

👉 Vertical Sync: иногда синхронизируем воркеров

[8] — PipeDream: Fast and Efficient Pipeline Parallel DNN Training, Harlap et al., SOSP'19
[9] — Efficient Large-Scale Language Model Training on GPU Clusters Using Megatron-LM, Narayanan et al., SC'21
[10] — Breadth-First Pipeline Parallelism, Joel Lamy-Poirier et al., MLSys'23

Tensor Parallelism

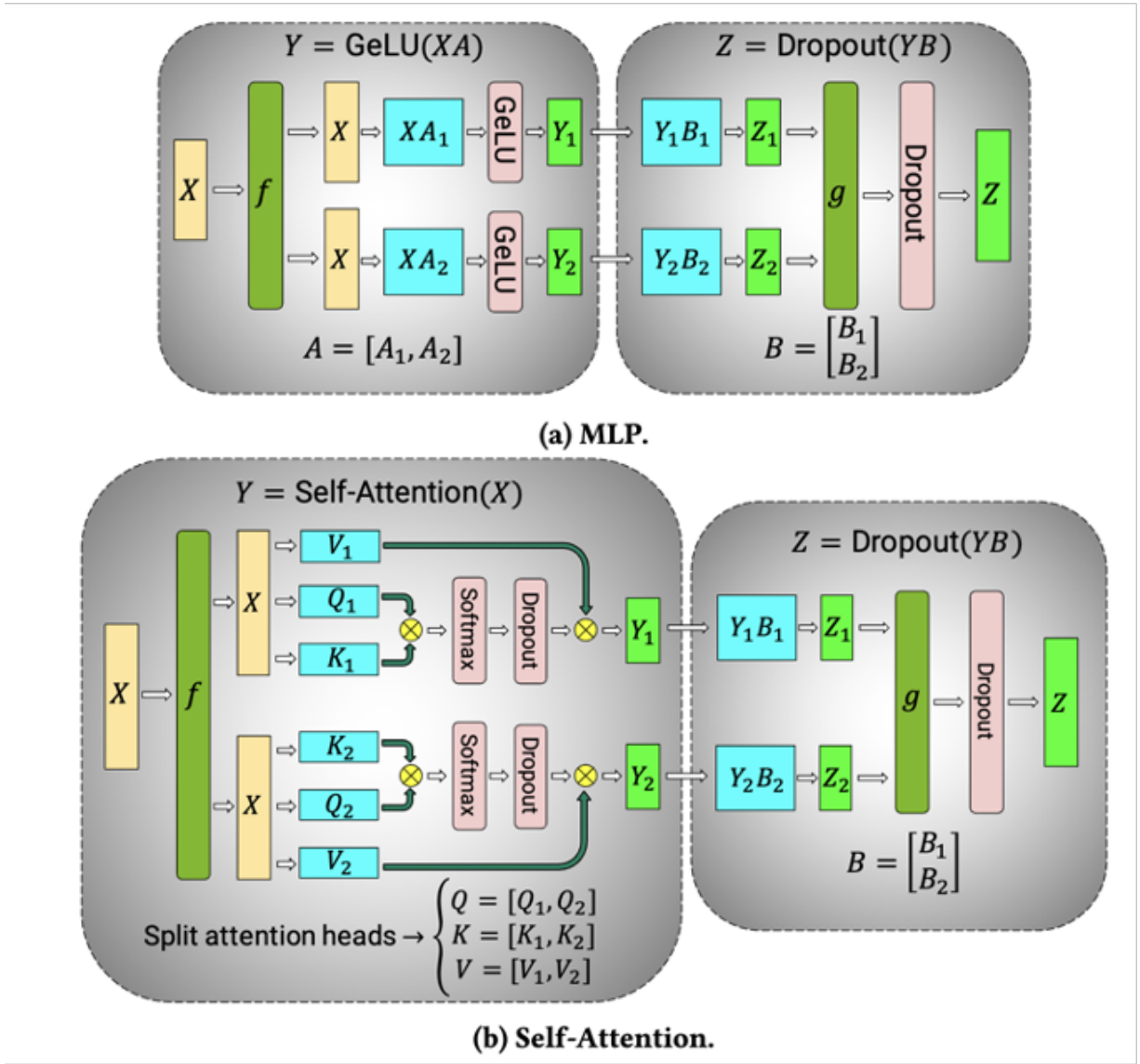
Большие матричные умножения можно разбить на несколько устройств

Горизонтальное масштабирование: во время forward-backward делаем множество `AllGather` и `AllReduce`

➔PyTorch 2.3+

➔DeepSpeed (tensor slicing)

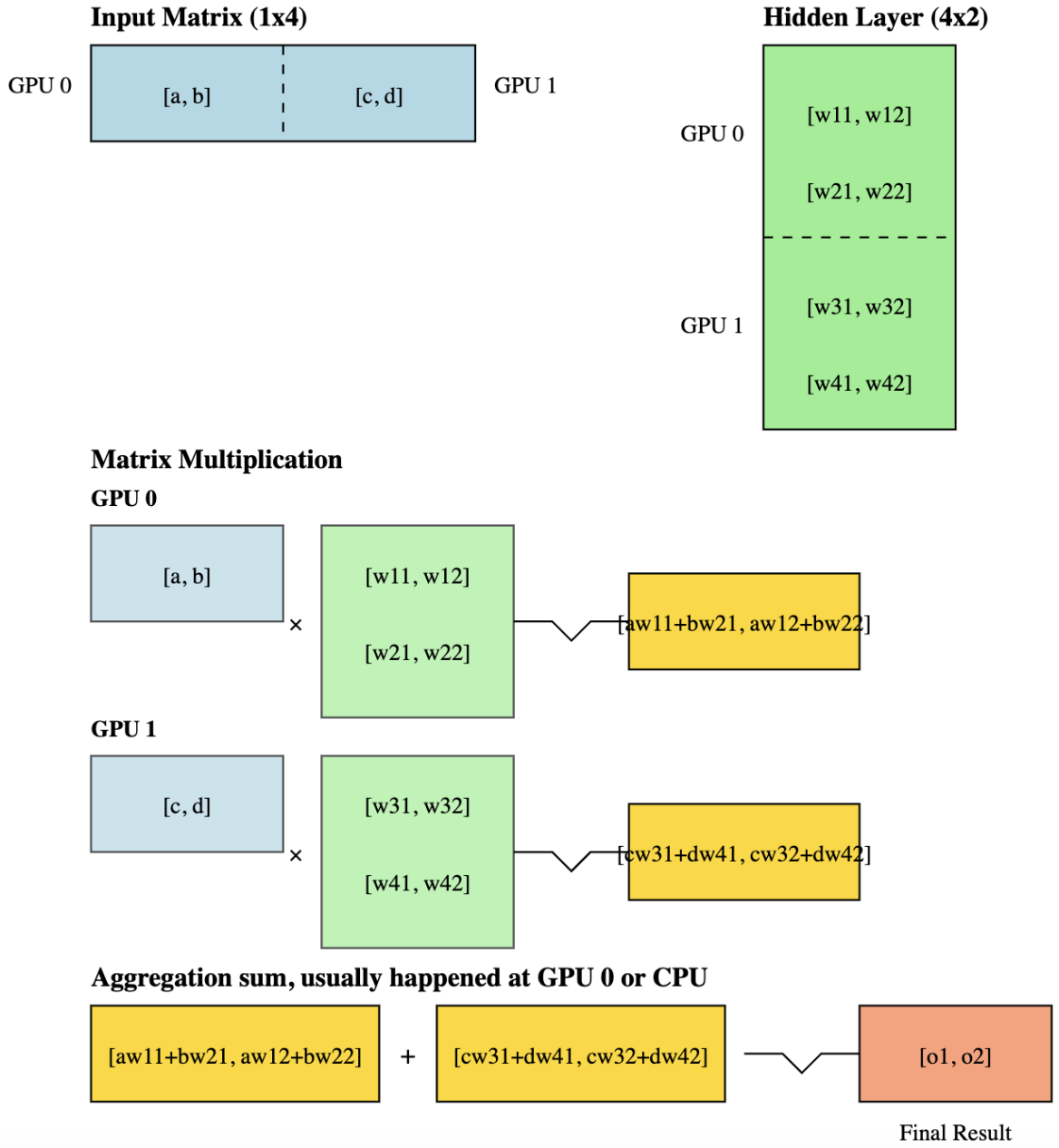
➔Megatron-LM



Row-Wise Parallel

Матрица весов разбивается **построчно** на n частей

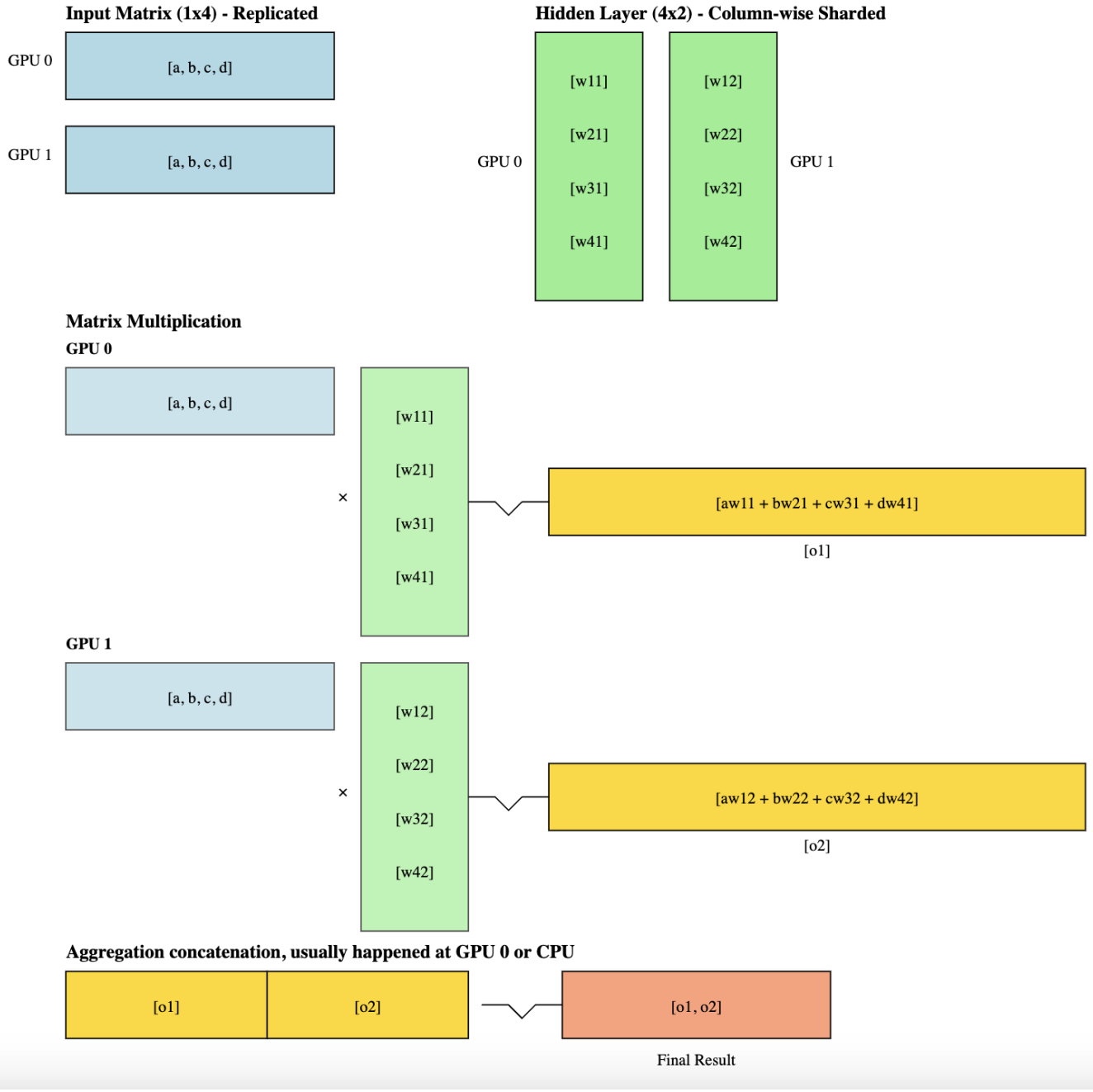
1. Input бьется на n частей, каждая часть отправляется на отдельный девайс
2. На девайсе перемножаем часть input и часть весов
3. Собираем все части на мастер устройстве
4. Агрегрируем — суммируем покомпонентно



Column-Wise Parallel

Матрица весов разбивается **по столбцам** на n частей

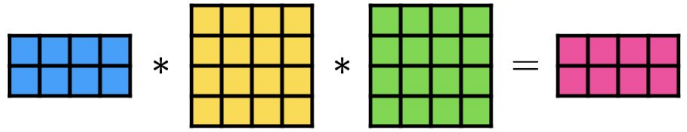
1. Input *реплицируется* n раз, каждая копия отправляется на отдельный девайс
2. На девайсе перемножаем копию input и часть весов
3. Собираем все части на мастер устройстве
4. Агрегрируем — *конкатенация*

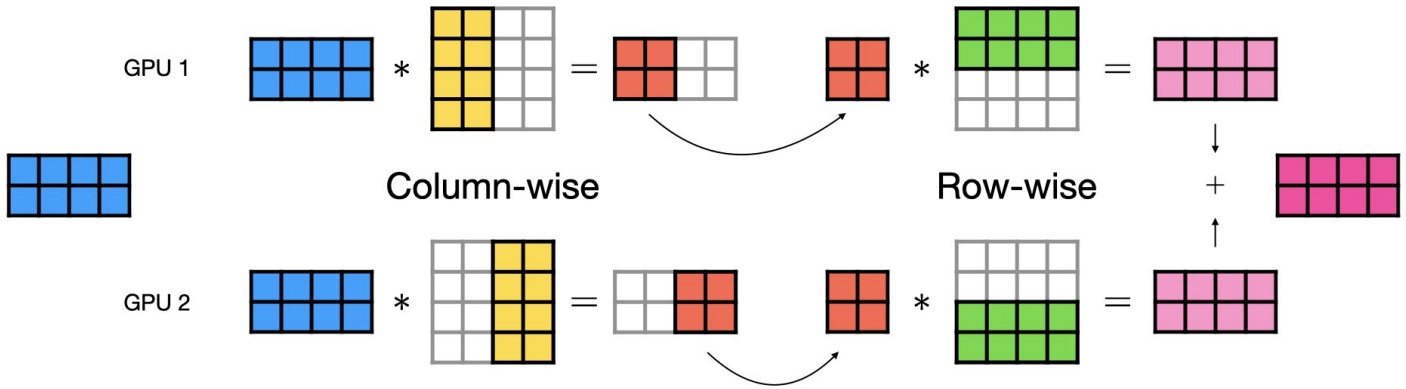


Row vs. Column

Выбор параллелизации зависит от типа слоя, его очередности и комбинации с остальными

- 👉 Embeddings, CrossEntropyLoss — row-wise параллелизация, разные токены словаря уходят на разные устройства
- 👉 MLP: ACT$(X \cdot W_1 + b_1) \cdot W_2 + b_2$
  - Если ReLU активация, то нельзя row-wise: $\max(0, X_1) + \max(0, X_2) \neq \max(0, X_1 + X_2)$
  - Оптимально комбинировать!



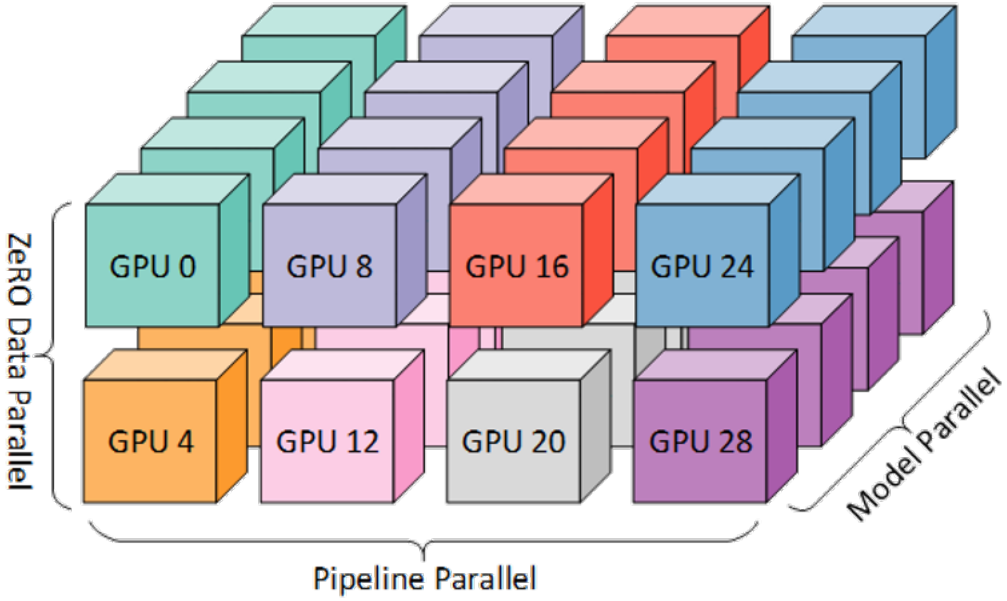


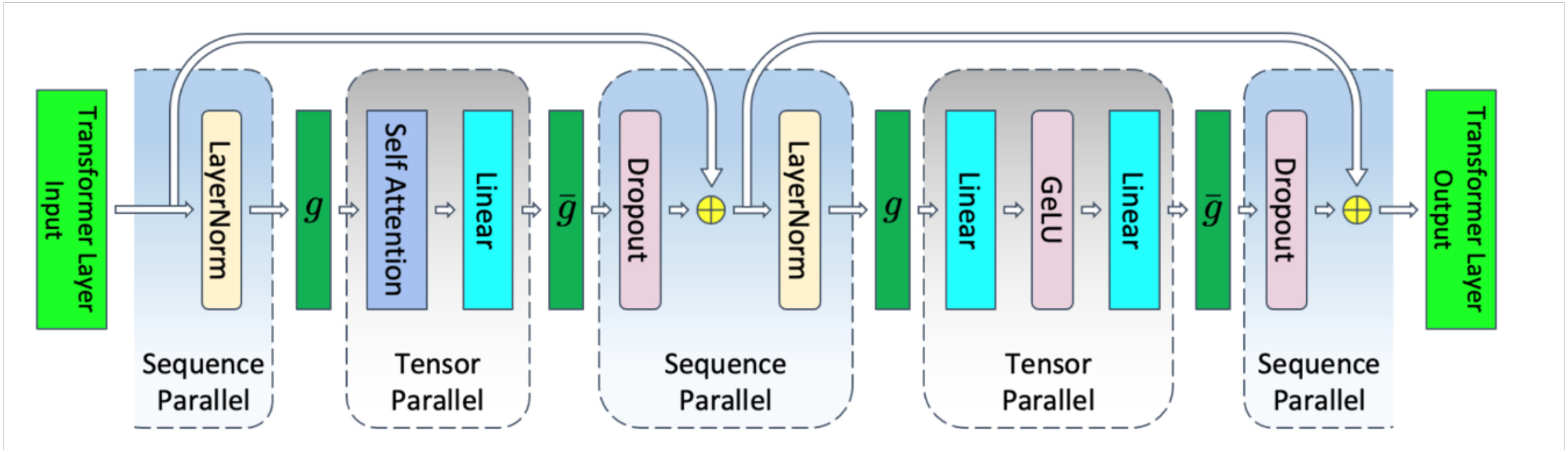
2D, 3D, 4D, ...

Почти все можно комбинировать друг с другом!

- 👉 8 GPU — DP на 8 девайсов
- 👉 8x8 GPU
  - Sharded DP на 64 девайса
  - DP на 8 девайсов + PP на 8 девайсов
  - DP на 8 девайсов + TP на 8 девайсов
- 👉 8x8xN GPU
  - Sharded DP на все девайсы
  - DP + PP + TP
  - Подключаем SP

Важно понимать нагрузку на сеть!





[11] — Megatron-LM: Training Multi-Billion Parameter Language Models Using Model Parallelism, Shoeybi et al., NVIDIA'20

Tl;Dr

Data Parallelism

+ Не требует модификации модели
+ AllReduce раз в шаг
- Никак не оптимизирует память

Sharded Data Parallelism

+ Не требует модификации модели
+ Существенно сокращает потребление памяти
- Forward и Backward делают множество синхронизаций между девайсами

Tensor Parallelism

+ Более “широкие” слои
+ Оптимизирует потребление памяти
- Требует вмешательство модели
- Scatter/Reduce раз в шаг

Pipeline Parallelism

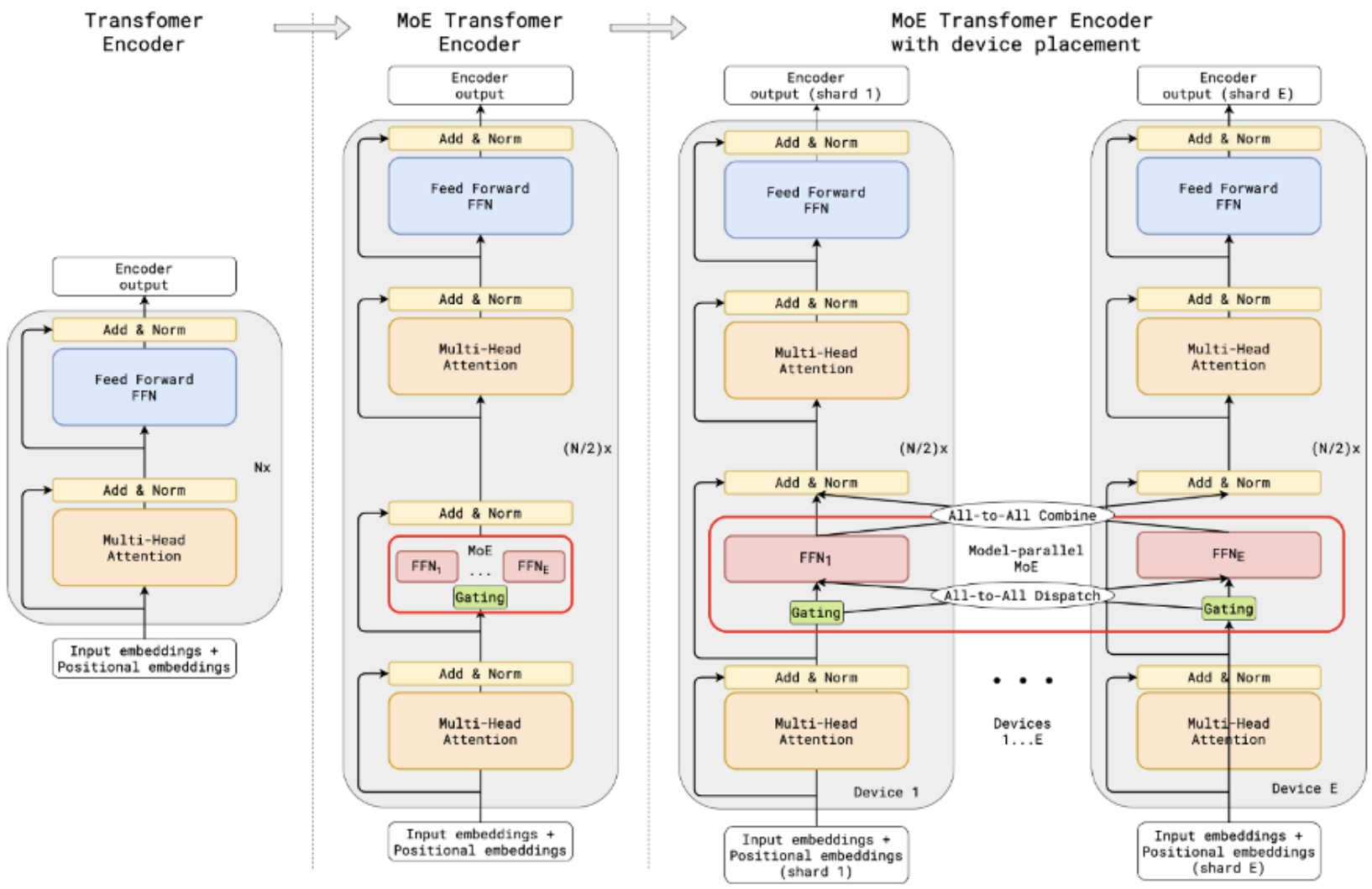
+ Оптимизирует потребление памяти
+ Scatter/Reduce раз в шаг для 1 девайса
- Существенные доработки в коде модели
- Работает не со всеми моделями

Все еще не хватает памяти

1. **CPU Offload**
   Давайте сгружать на CPU что-то ненужное
   1. Оптимизатор (DeepSpeedCPUAdam)
   2. Активации
2. **Activation Recomputation / Gradient Checkpointing**
   Давайте пересчитывать активации заново
   1. Определяем границы пересчета, например, слой трансформера или простая активация
   2. Поддерживается в моделях от 🤗
3. **Memory-efficient оптимизаторы**
   1. 4-byte оптимизаторы: Adafactor, LION
   2. 8-bit оптимизаторы: bitsandbytes.Adam8bit

[12] — Reducing Activation Recomputation in Large Transformer Models, Korthikanti et al., MLSys'24

MoE & Expert Parallelization



MoE — sparse-модель

- 👉 Значительно увеличиваем число параметров
- 👉 Expert Parallelism — каждый эксперт помещается на отдельный девайс
- 👉 Router — определяет для каждого токена эксперта
- 👉 Перед и после эксперта надо синхронизировать токены между девайсами

[13] — GShard: Scaling Giant Models with Conditional Computation and Automatic Sharding, Lepikhin et al., ICLR'21

На этом все🤝

Егор Спирин — vk.com/boss
VK Lab — vk.com/lab