Школа анализа данных

Арифметика глубокого обучения

Хрущев Михаил
Руководитель группы претрейна YandexGPT

Познакомимся

Я – Михаил Хрущев, руковожу группой претрейна YandexGPT.

Более 4 лет я и моя команда обучаем и разрабатываем инфраструктуру больших языковых моделей.

Основная часть LLM моделей Яндекса были обучены нами или на нашей инфраструктуре.

Наши open source:

- YaFSDP
- YaLM 100B
- Yandex GPT5 Lite

Мотивация

- Обучение одного инфраструктурного DL-специалиста требует работы с тысячами GPU в течение пары лет.
- Это дорого, мы хотим поделиться опытом, заинтересовать в нашем домене

Содержание

История 1

Перед итерацией обучения нам нужно вычитать и подготовить батч.
Обычно подготовка происходит на CPU:

История 1

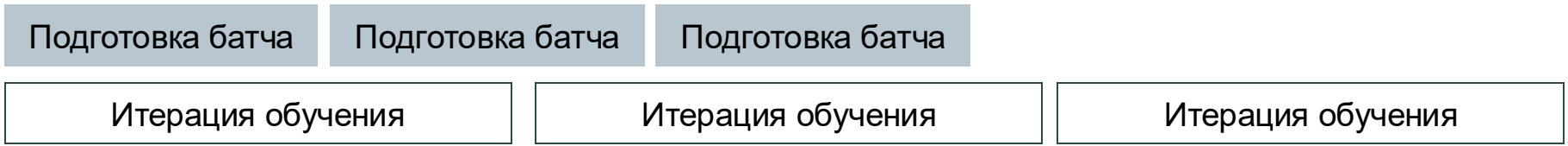
Перед итерацией обучения нам нужно вычитать и подготовить батч.
Обычно подготовка происходит на CPU:

Проблема: Из-за подготовки батча простаивает GPU – это очень дорогой ресурс.

История 1

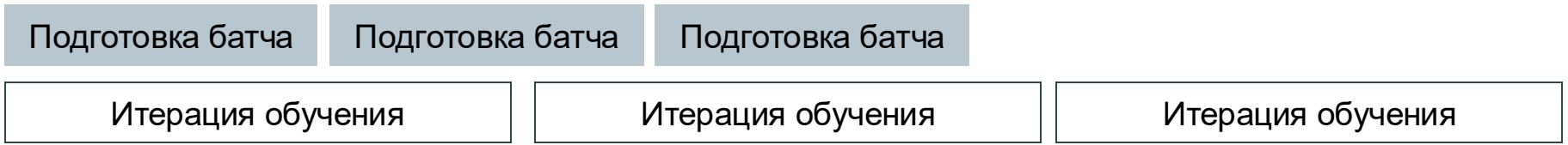
Перед итерацией обучения нам нужно вычитать и подготовить батч.
Обычно подготовка происходит на CPU:

Что можно сделать: Выносим чтение и подготовку батча в отдельный процесс



История 1. Выводы

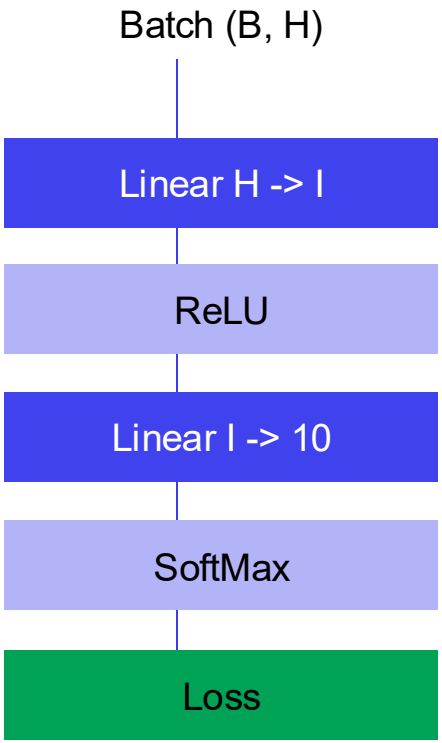
- Подготовка данных может привести к замедлению обучения и простою GPU.
- GPU – самый дорогой ресурс, его простой нежелателен.
- Чтобы избежать простоя, можно сделать загрузку данных **асинхронной**.



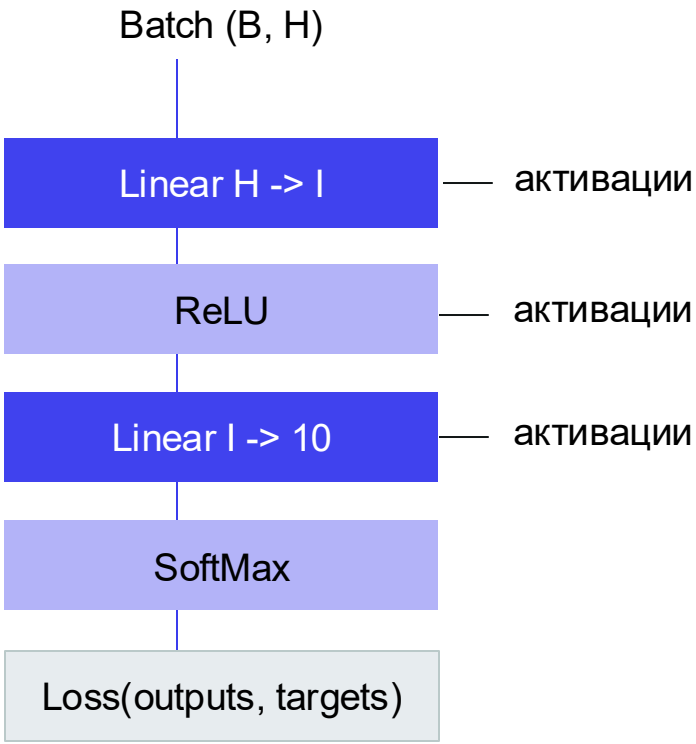
История 2. Внутри итерации

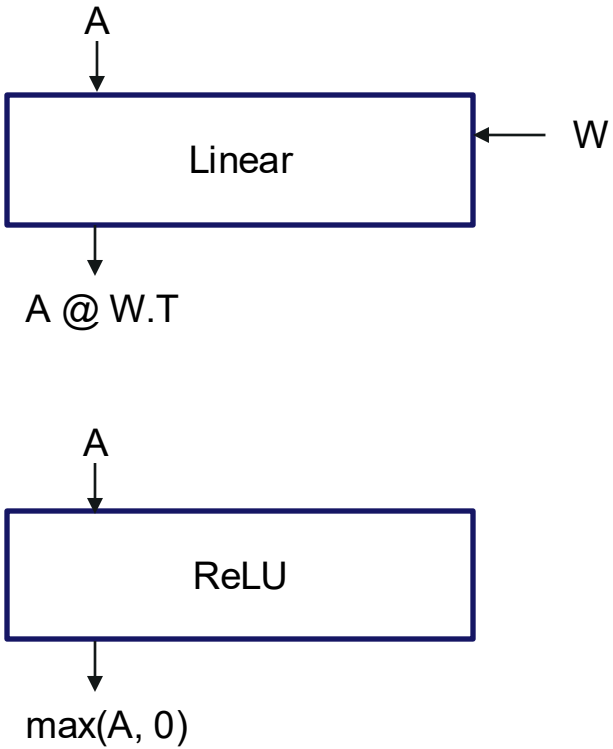
Итерация обучения

История 2. Небольшой экскурс в DL

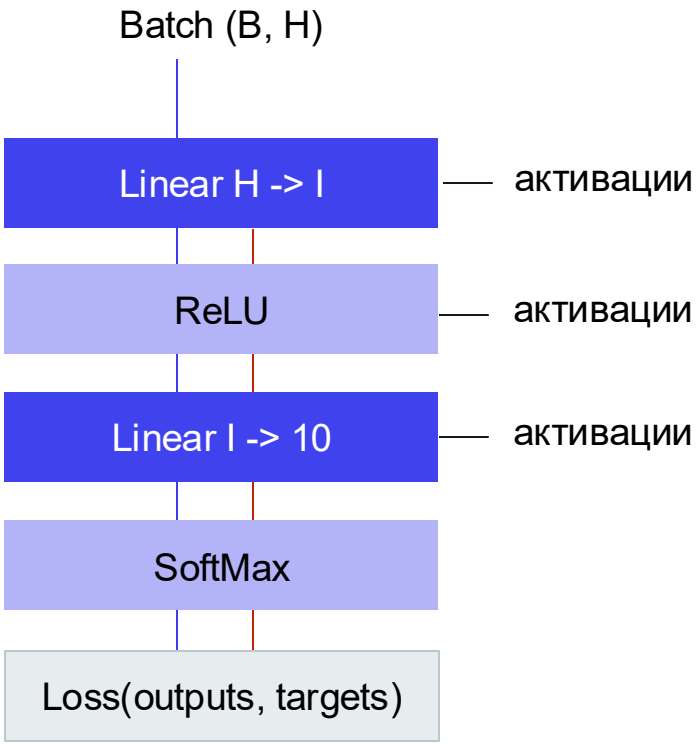


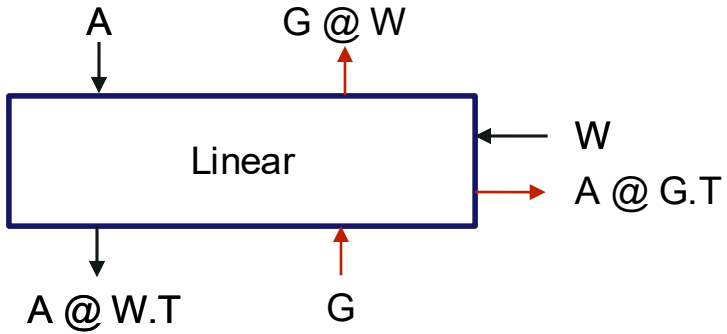
История 2. Небольшой экскурс в DL

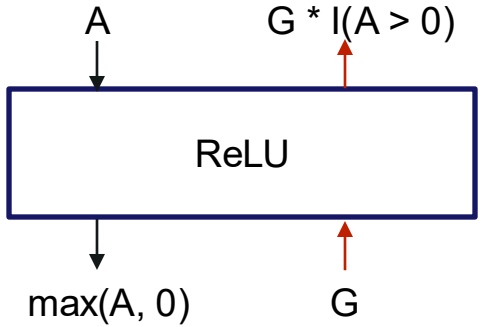




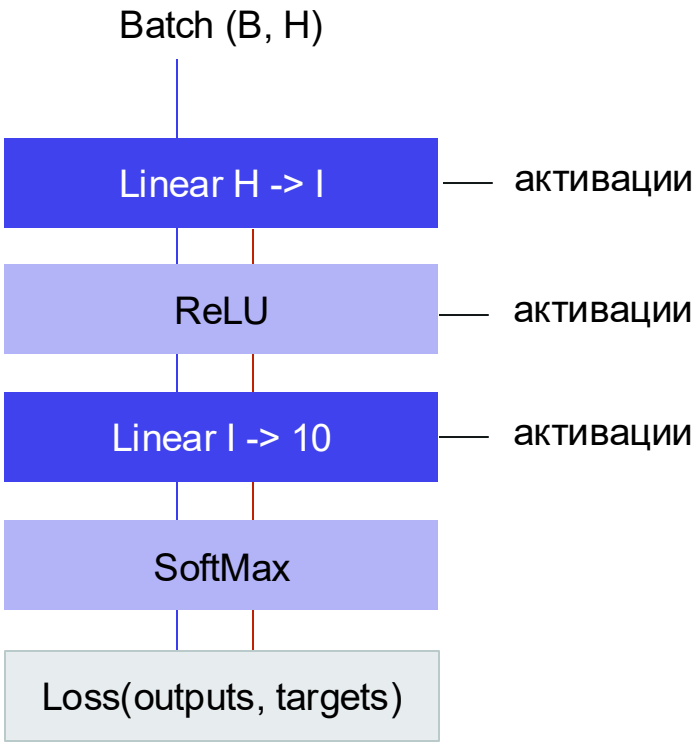
История 2. Небольшой экскурс в DL

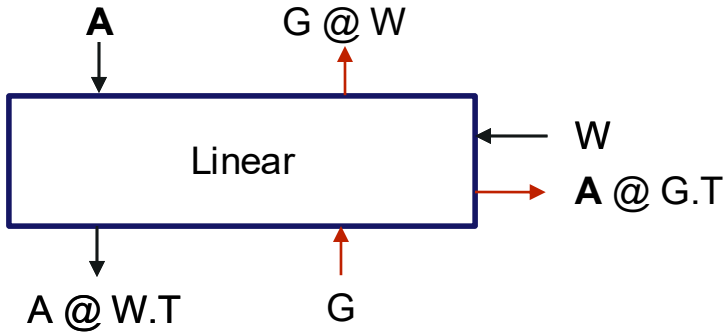


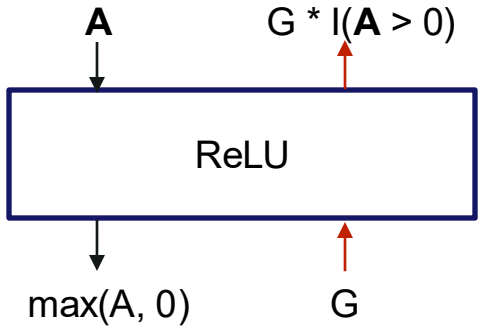




История 2. Небольшой экскурс в DL

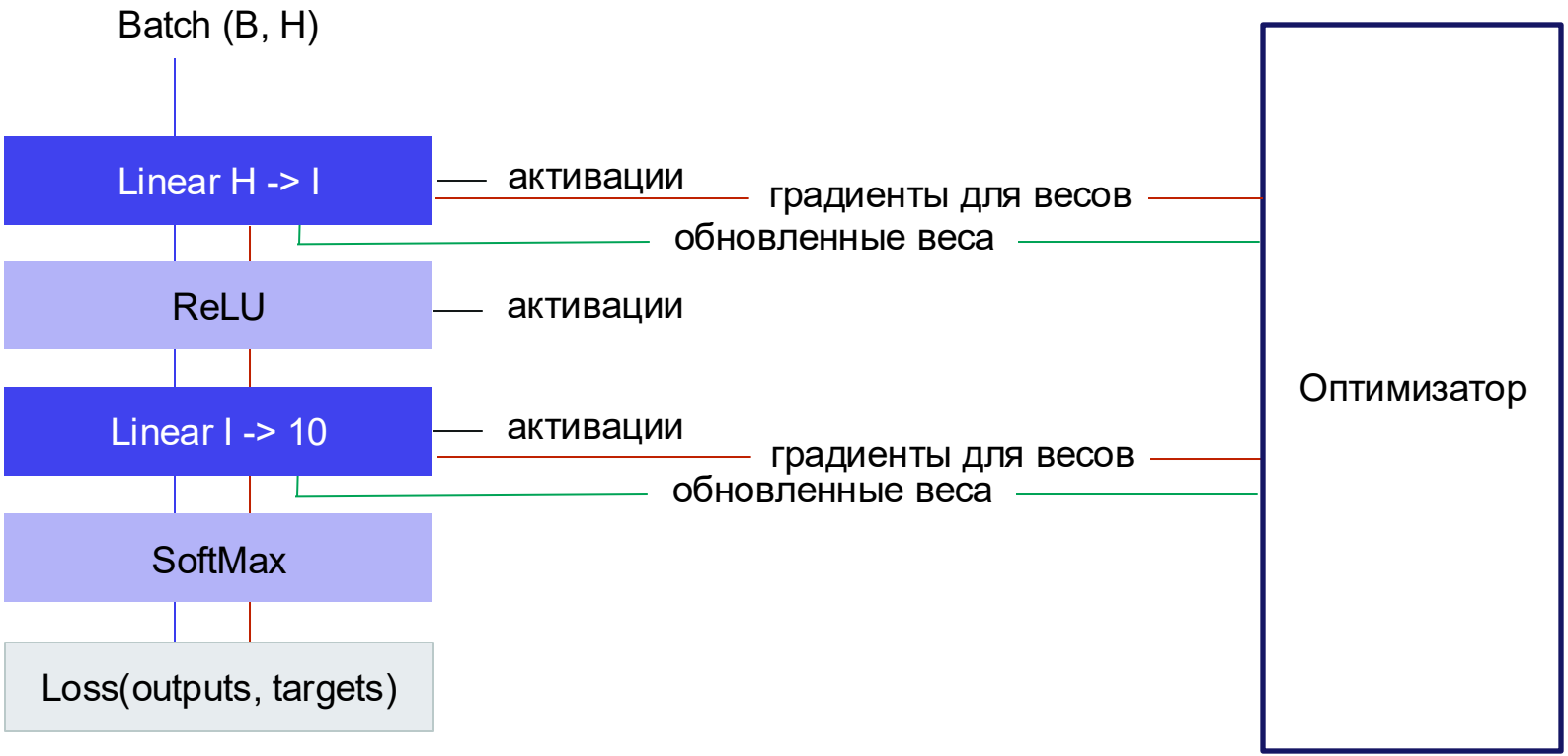




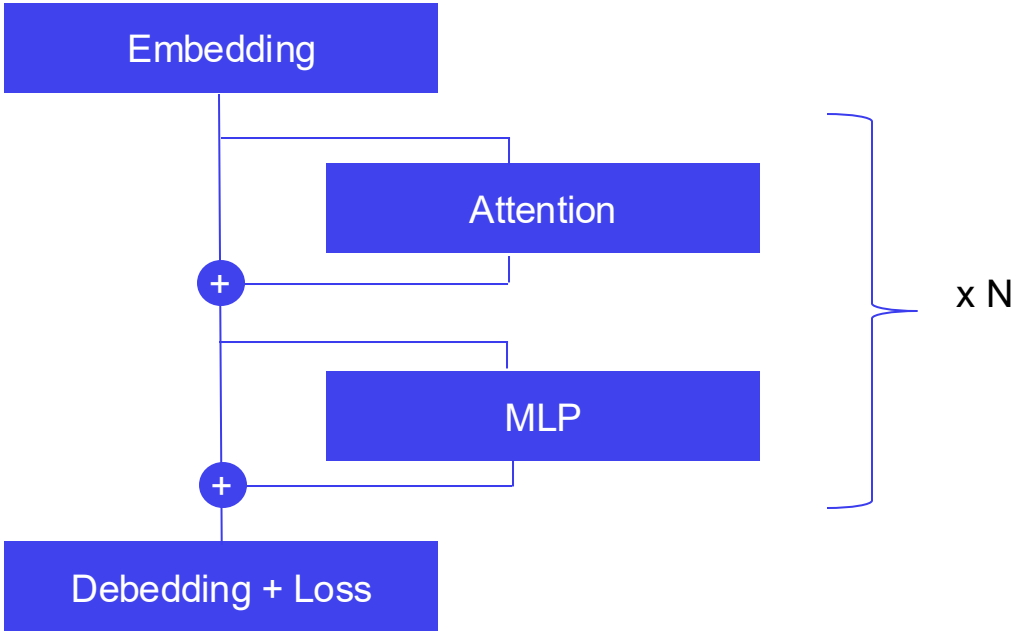


История 2. Небольшой экскурс в DL

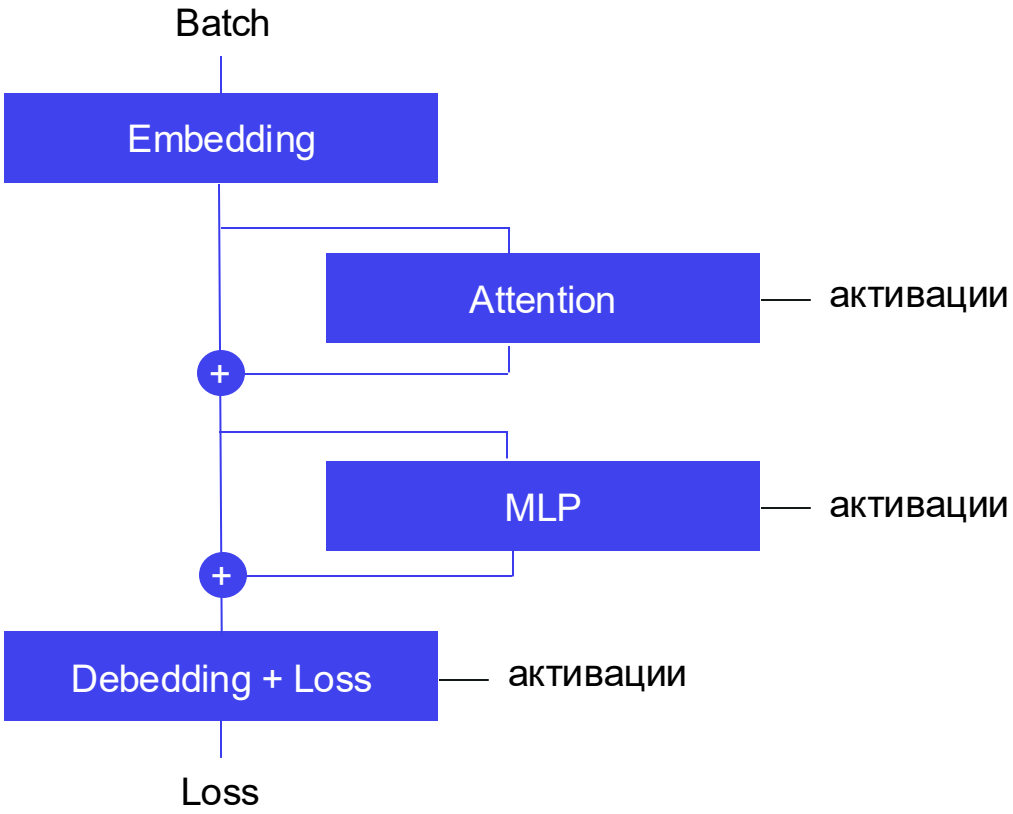
История 2. Небольшой экскурс в DL



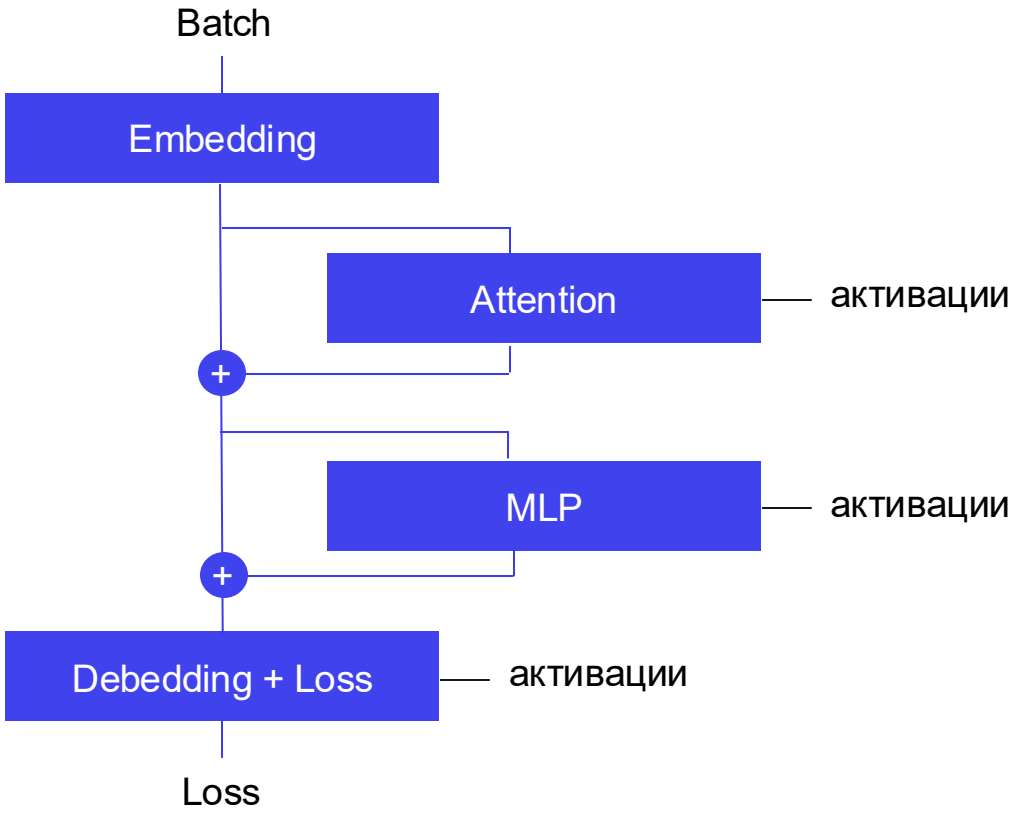
История 2

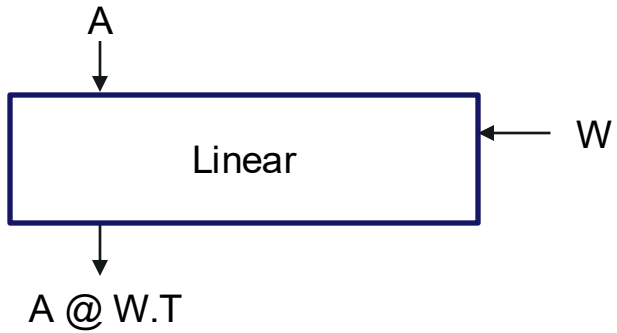


История 2

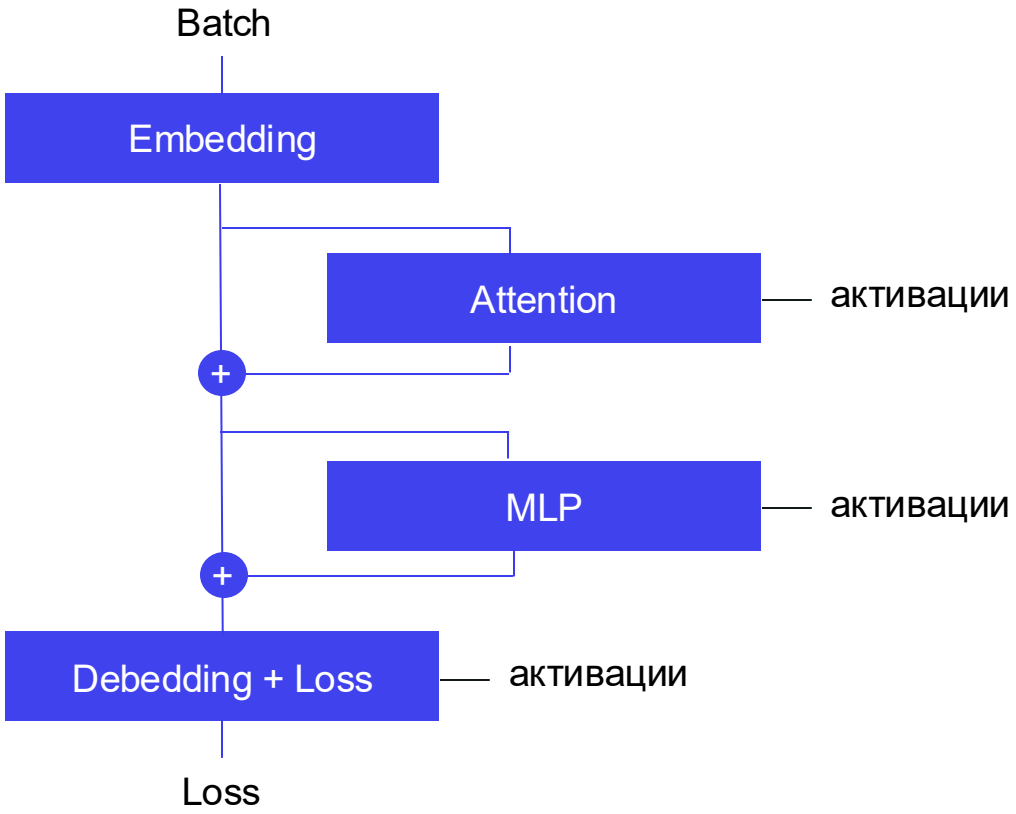

Batch
Embedding
Attention
активации
+
MLP
активации
+
Debedding + Loss
активации
Loss

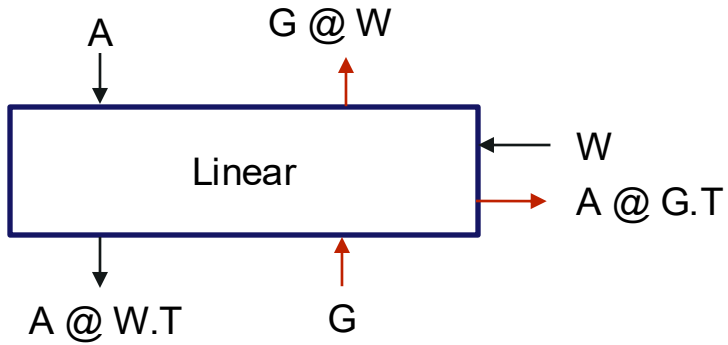
История 2


Batch
Embedding
Attention
активации
+
MLP
активации
+
Debedding + Loss
активации
Loss

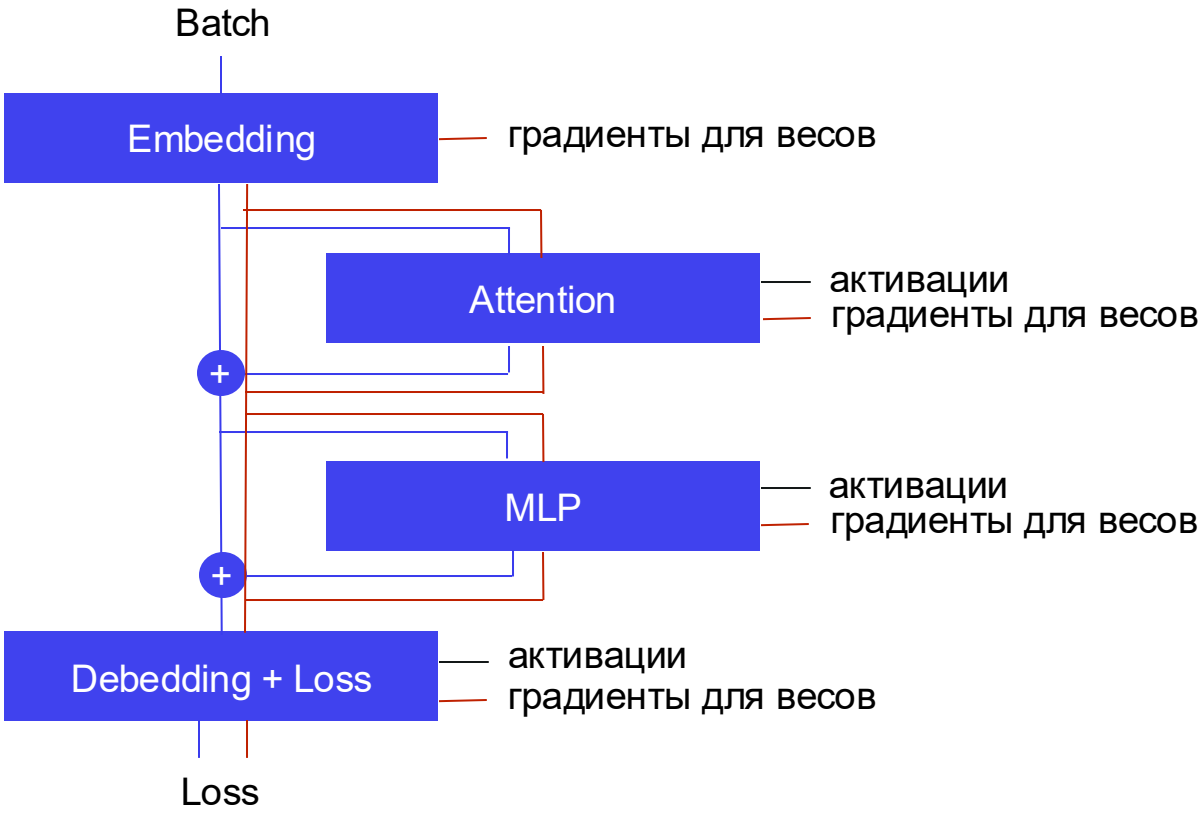

A
Linear
W
A @ W.T

История 2

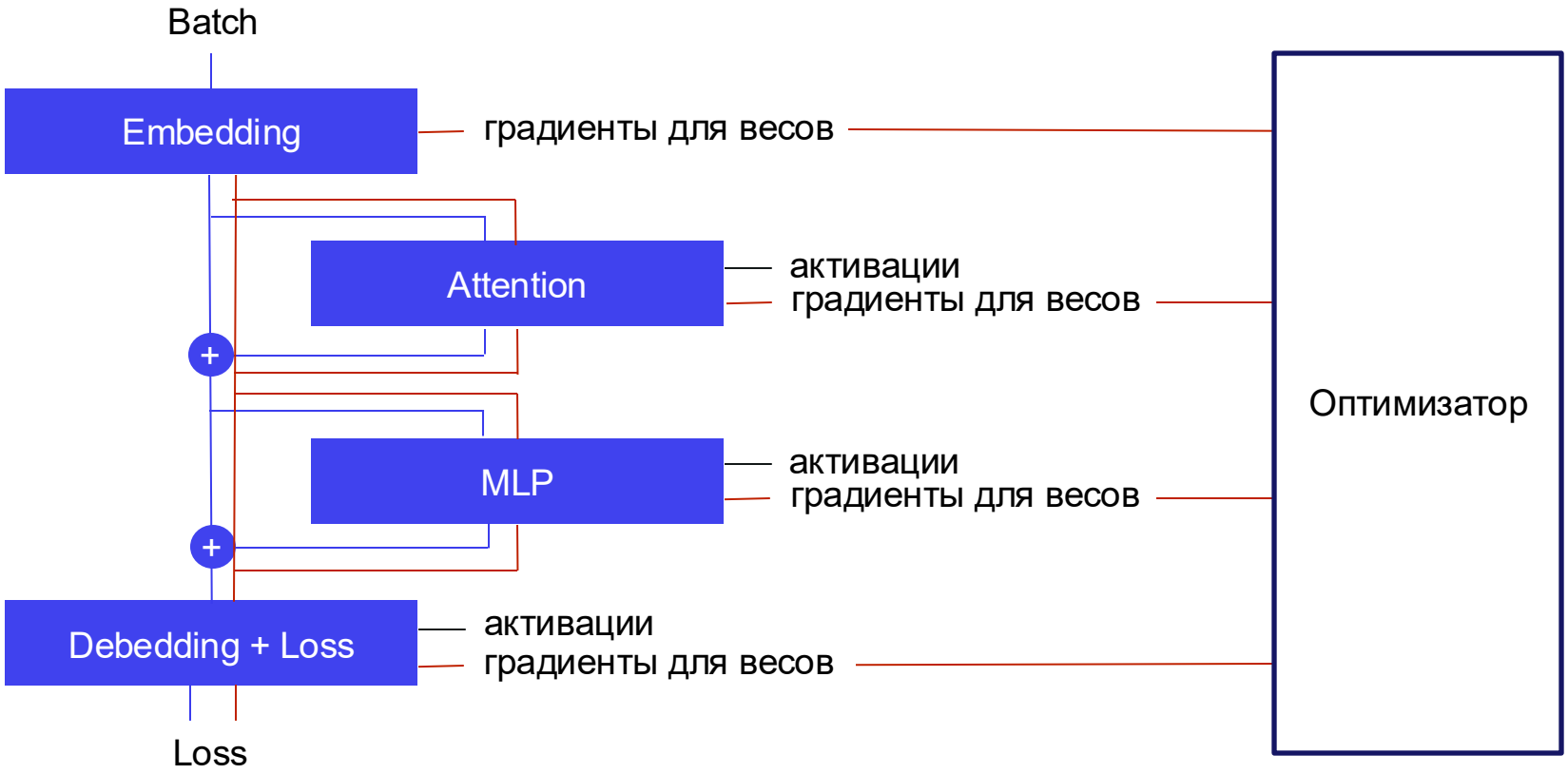

Batch
Embedding
Attention
активации
+
MLP
активации
+
Debedding + Loss
активации
Loss


A
G @ W
Linear
W
A @ G.T
A @ W.T
G

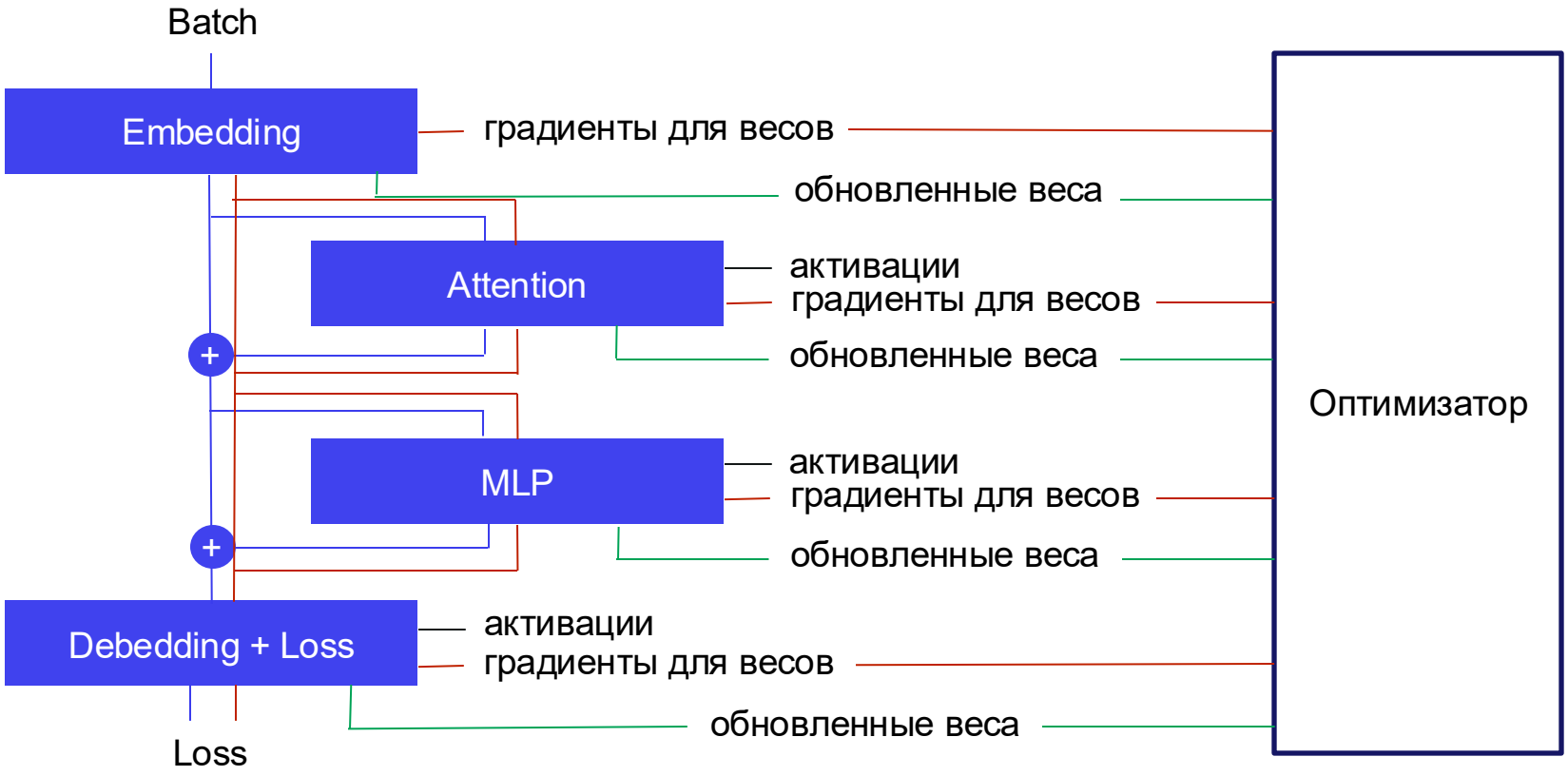
История 2


Batch
Embedding
градиенты для весов
Attention
активации
градиенты для весов
+
MLP
активации
градиенты для весов
+
Debedding + Loss
активации
градиенты для весов
Loss

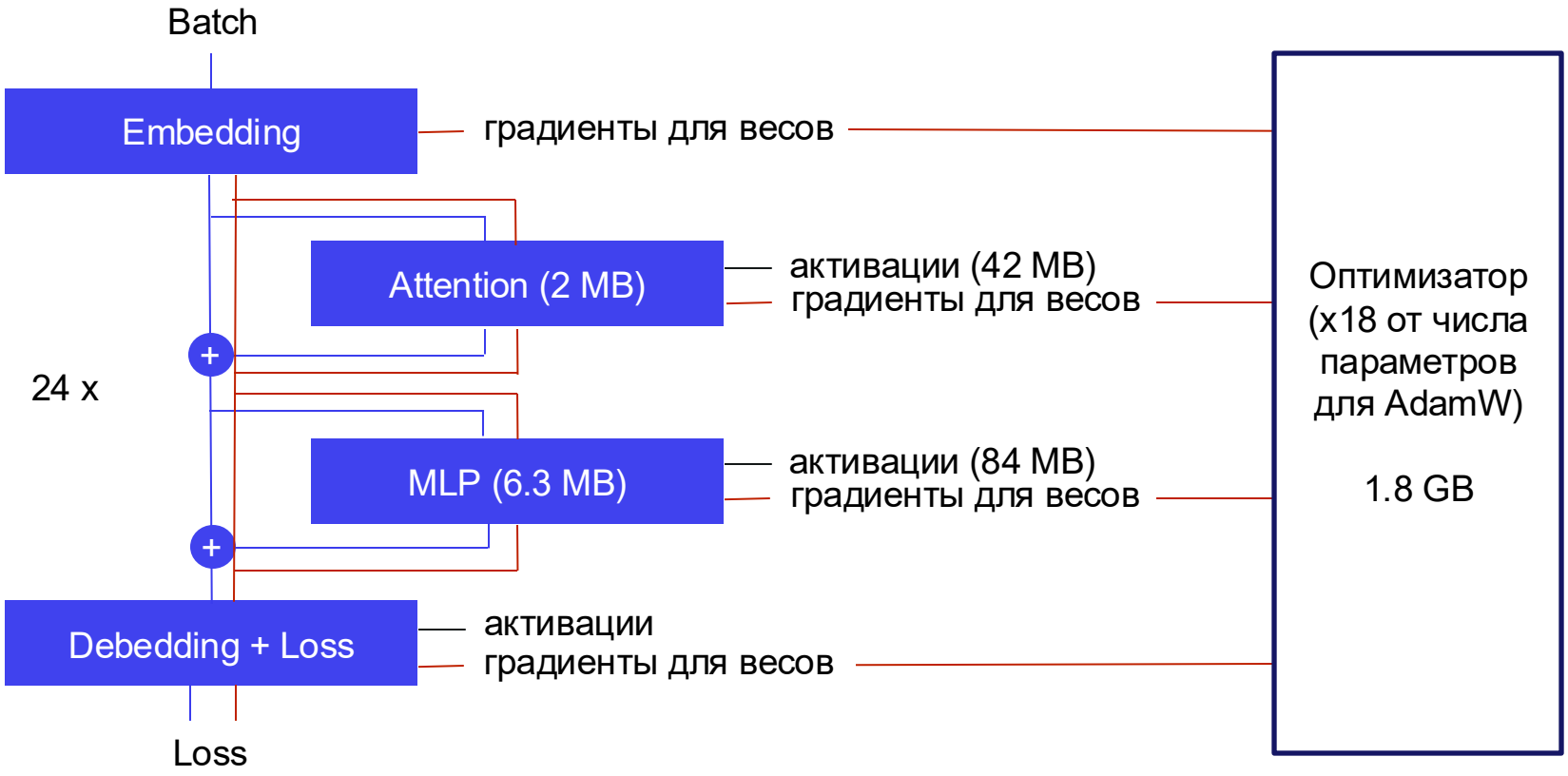
История 2


Batch
Embedding
градиенты для весов
Attention
активации
градиенты для весов
+
MLP
активации
градиенты для весов
+
Debedding + Loss
активации
градиенты для весов
Loss
Оптимизатор

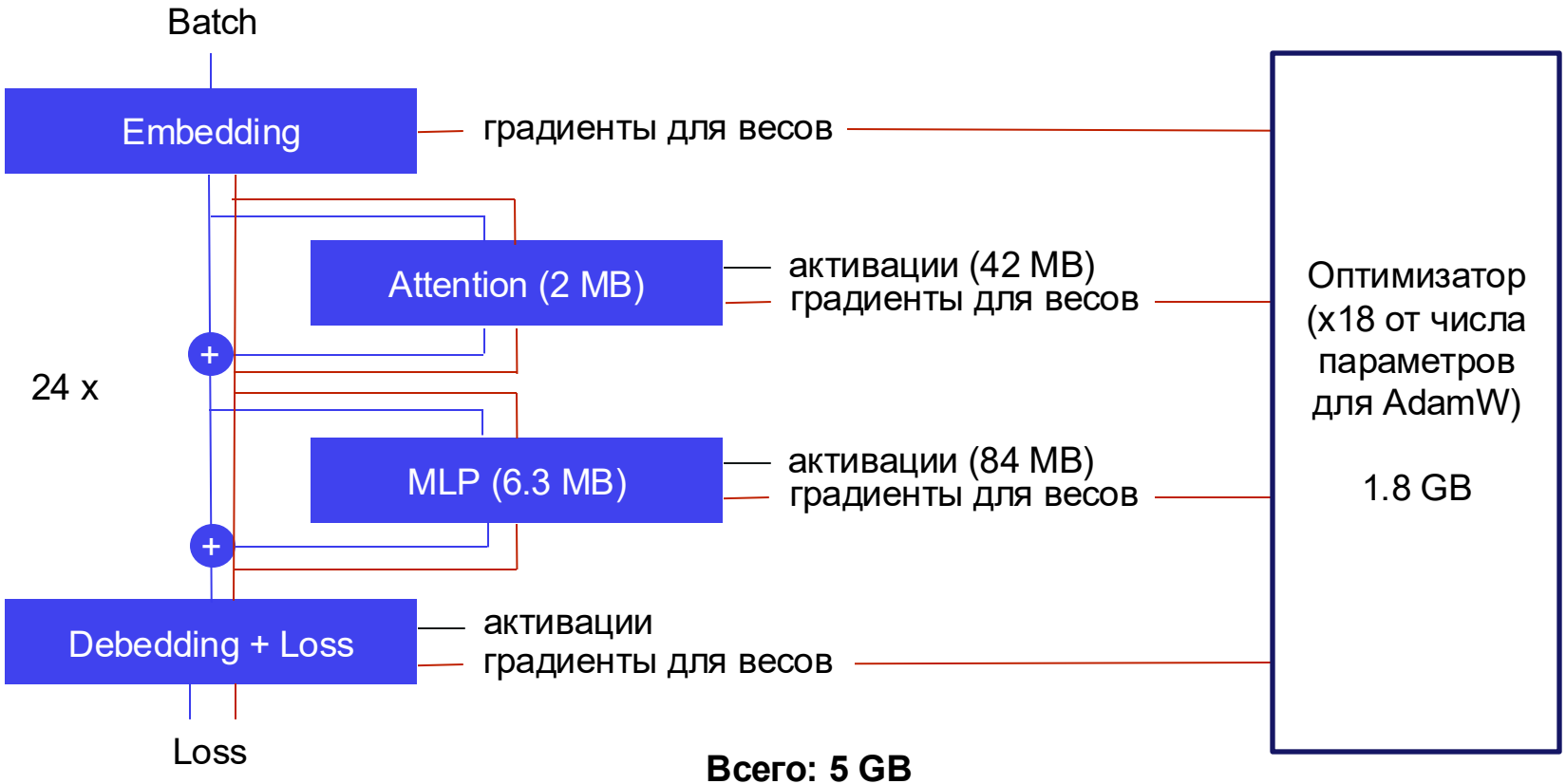
История 2


Batch
Embedding
градиенты для весов
обновленные веса
Attention
активации
градиенты для весов
обновленные веса
+
MLP
активации
градиенты для весов
обновленные веса
+
Debedding + Loss
активации
градиенты для весов
обновленные веса
Loss
Оптимизатор

100M модель

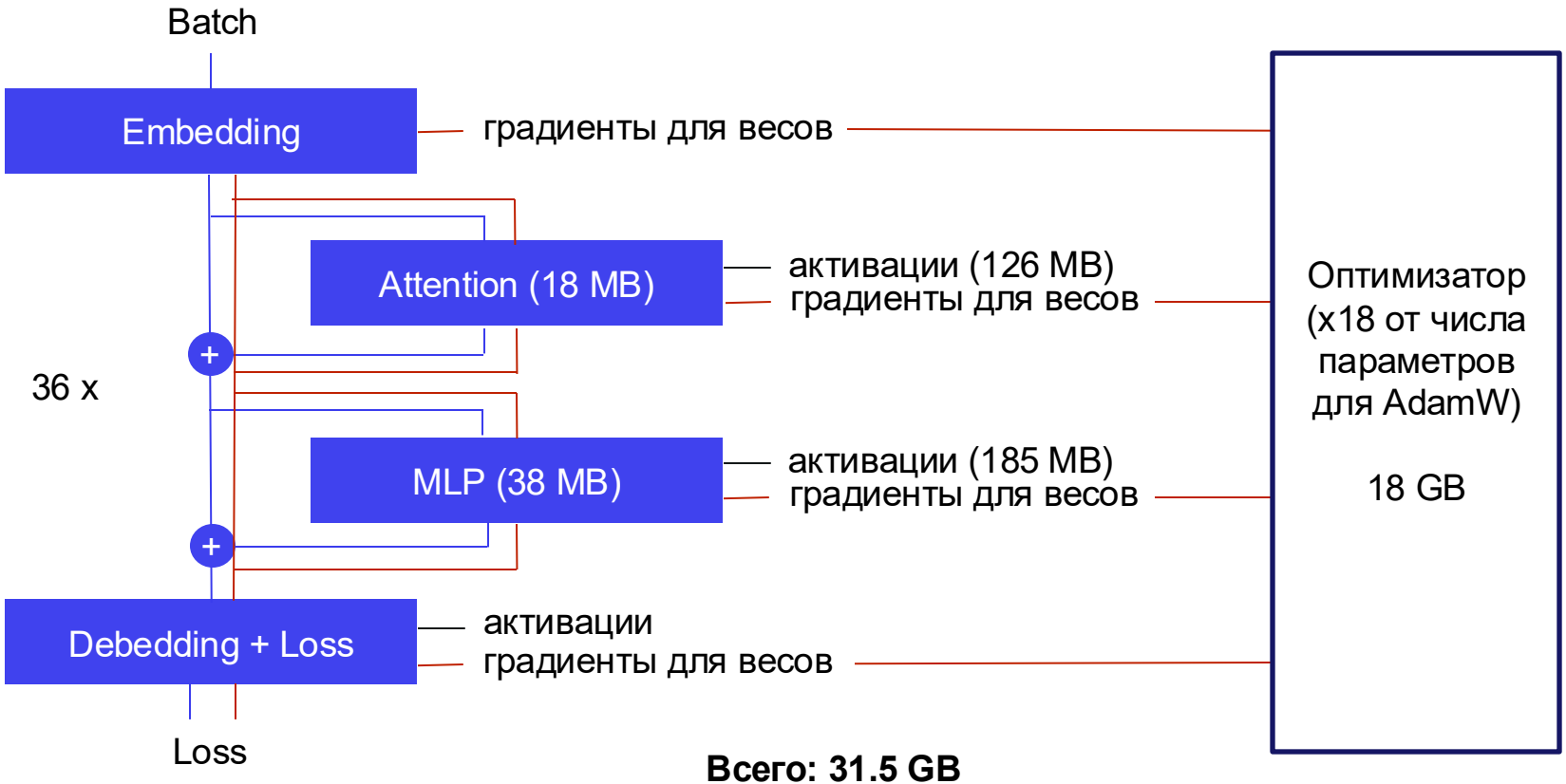

Batch
Embedding
градиенты для весов
Attention (2 MB)
активации (42 MB)
градиенты для весов
24 x
MLP (6.3 MB)
активации (84 MB)
градиенты для весов
Debedding + Loss
активации
градиенты для весов
Loss
Оптимизатор (x18 от числа параметров для AdamW)
1.8 GB

100M модель



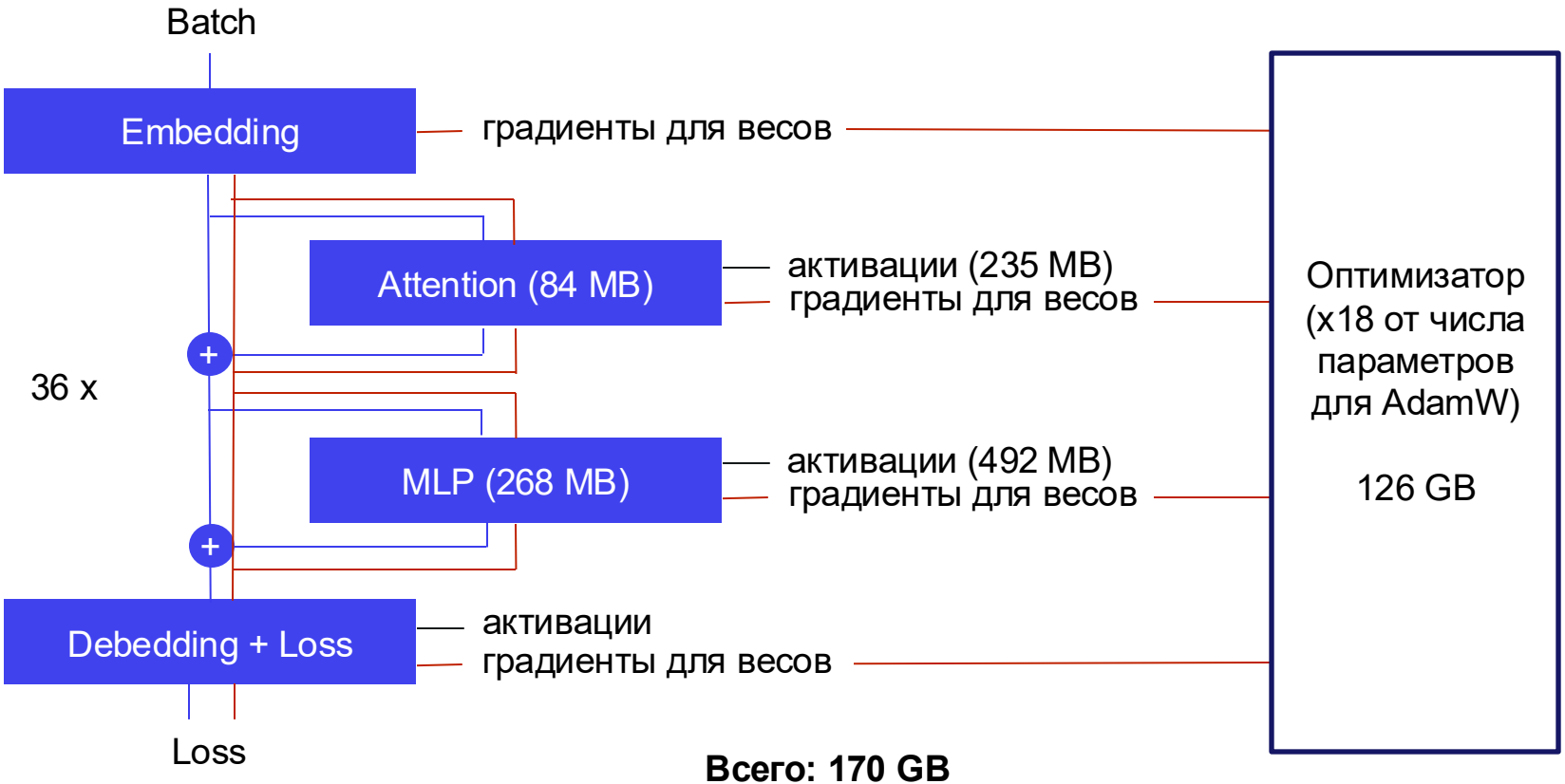
Всего: 5 GB

1B модель



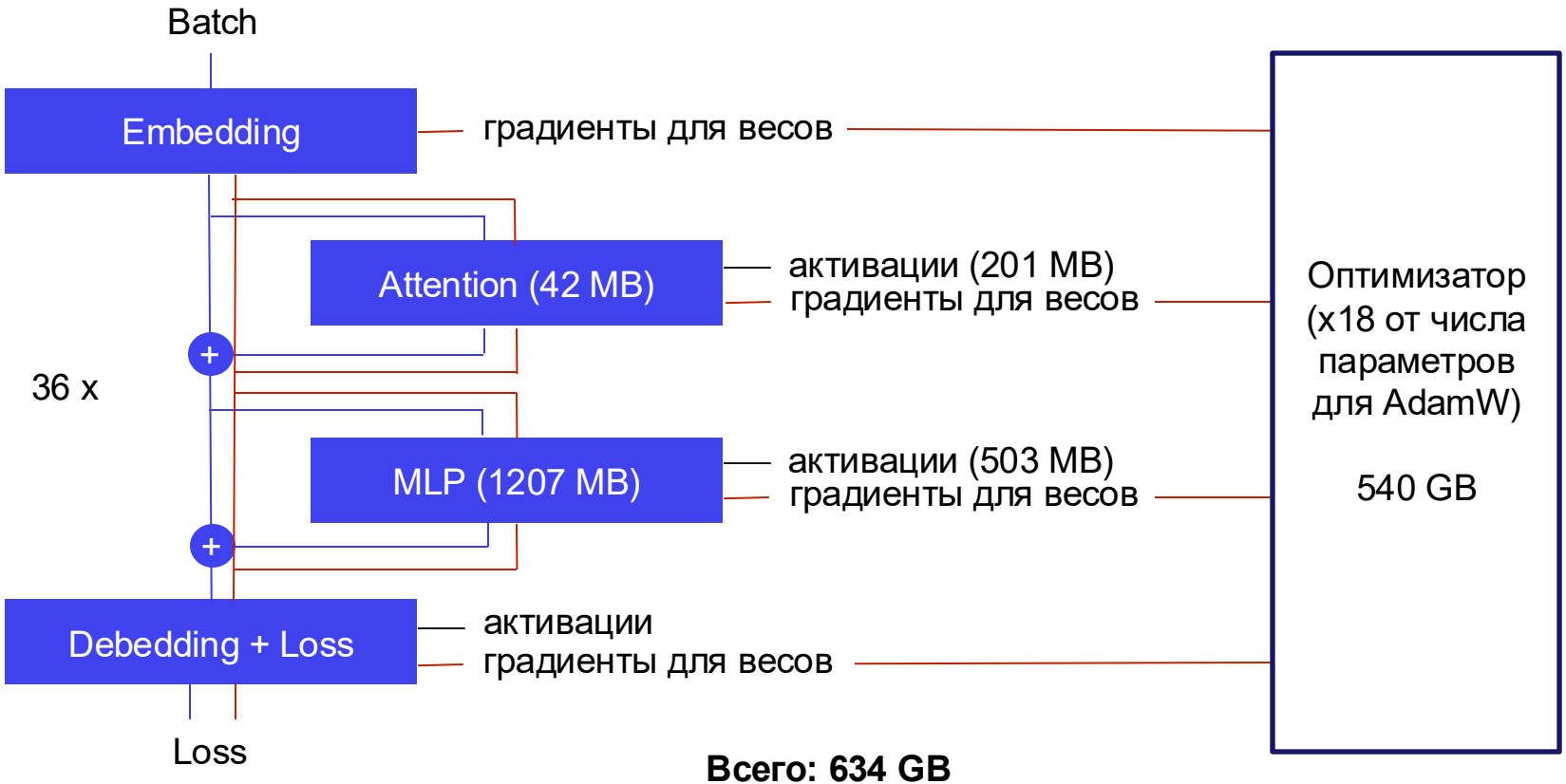
Всего: 31.5 GB

7B модель



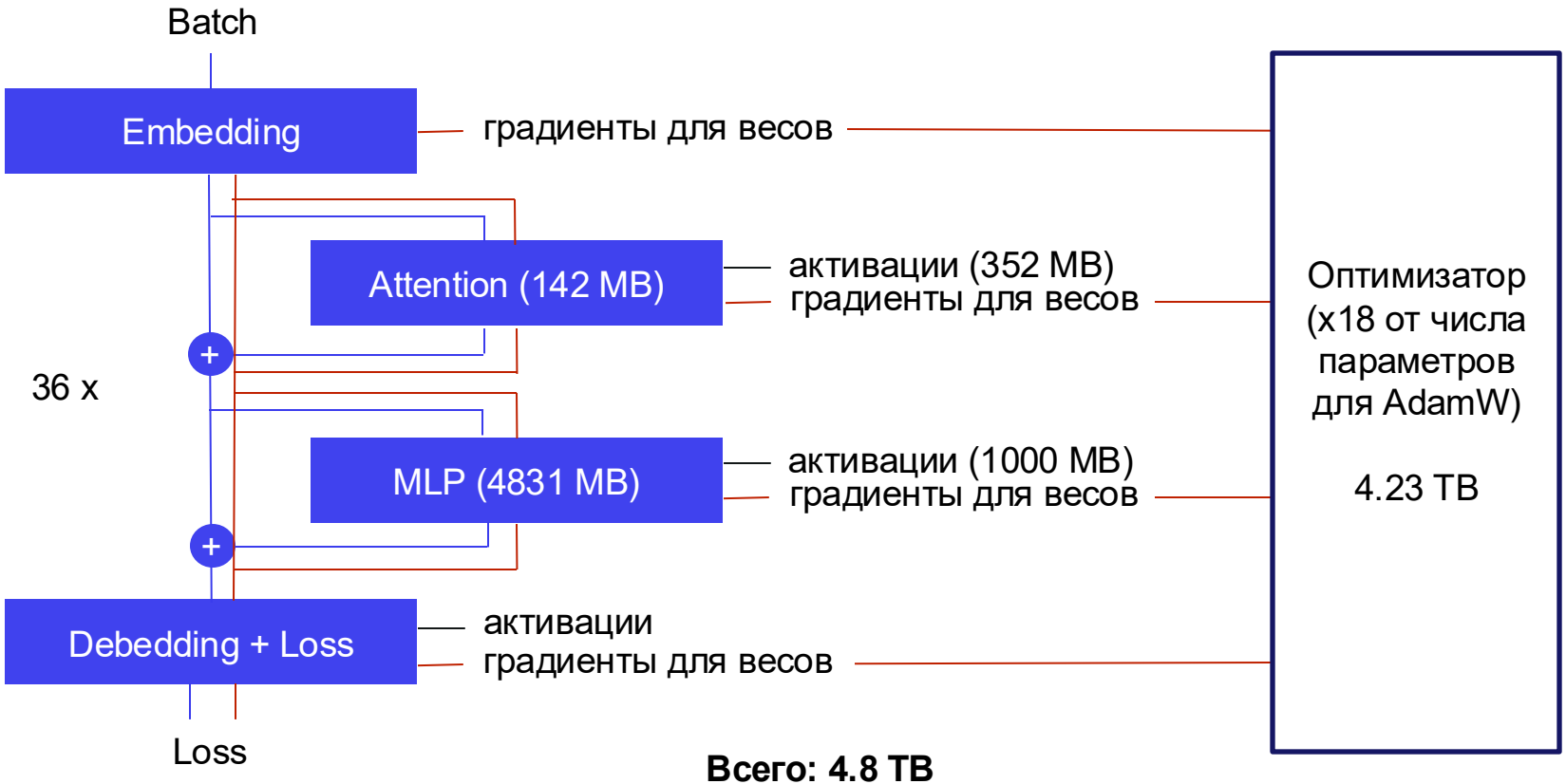
Всего: 170 GB

Qwen 30B-A3B



Всего: 634 GB

Qwen 235B-A32B



Всего: 4.8 TB

История 2. Проблема

- Для обучения модели нужны терабайты памяти.
- Но на GPU не так много: 40GB-190GB для серверных GPU, единицы GB для игровых.

Что делать?

История 2. Проблема

- Для обучения модели нужны терабайты памяти.
- Но на GPU не так много: 40GB-190GB для серверных GPU, единицы GB для игровых.

Что делать?
Хранить где-то еще!

Как использовать веса/активации/состояния оптимизатора, если они лежат где-то?

История 2. Проблема

- Для обучения модели нужны терабайты памяти.
- Но на GPU не так много: 40GB-190GB для серверных GPU, единицы GB для игровых.

Что делать?
Хранить где-то еще!

Как использовать веса/активации/состояния оптимизатора, если они лежат где-то?
Организовать пересылку

Но GPU будут простаивать, если пересылать данные?

История 2. Проблема

- Для обучения модели нужны терабайты памяти.
- Но на GPU не так много: 40GB-190GB для серверных GPU, единицы GB для игровых.

Что делать?
Хранить где-то еще!

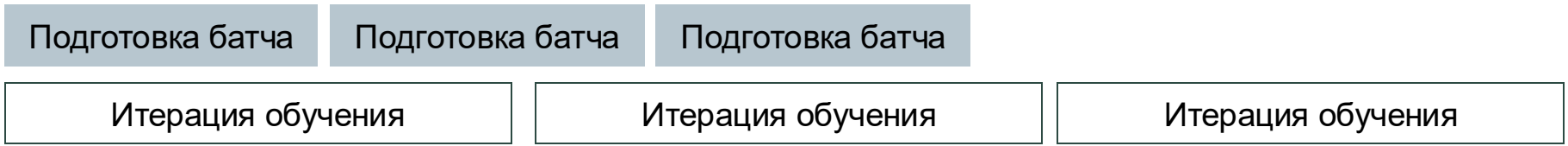
Как использовать веса/активации/состояния оптимизатора, если они лежат где-то?
Организовать пересылку

Но GPU будут простаивать, если пересылать данные?
Нет, если пересылка асинхронная

История 1

Перед итерацией обучения нам нужно вычитать и подготовить батч.
Обычно подготовка происходит на CPU:

Что можно сделать: Выносим чтение и подготовку батча в отдельный процесс



Эффективное обучение – логистическая проблема

- Нужно разнести состояния оптимизатора, веса и активации на разные ресурсы: другие GPU, RAM, SSD.
- Нужно обеспечить сборку этих данных в момент использования на каждой GPU.
- Нужно организовать логистику так, чтобы GPU простаивала минимально

Эффективное обучение – логистическая проблема

- Нужно разнести состояния оптимизатора, веса и активации на разные ресурсы: другие GPU, RAM, SSD.
- Нужно обеспечить сборку этих данных в момент использования на каждой GPU.
- Нужно организовать логистику так, чтобы GPU простаивала минимально

Этой задачей занимаются многие исследователи DL уже больше 7 лет.

Содержание

01

Определяемся с логистикой

Логистика

Что нужно для правильного построения логистики

- Нужна карта маршрутов.
- Нужно понимание того, сколько стоит пройти тот или иной маршрут.

Что нужно для правильного построения логистики

- Нужна карта маршрутов.
- Нужно понимание того, сколько стоит пройти тот или иной маршрут.

Как получить такие данные?

- Спросить у экспертов
- Почитать спецификацию
- Замерить самим

Смотрим в спецификацию

	H100 SXM	H100 NVL
TF32 Tensor Core*	989 teraFLOPS	835 teraFLOPs
BFLOAT16 Tensor Core*	1,979 teraFLOPS	1,671 teraFLOPS
FP8 Tensor Core*	3,958 teraFLOPS	3,341 teraFLOPS
GPU Memory	80GB	94GB
GPU Memory Bandwidth	3.35TB/s	3.9TB/s
Interconnect	NVIDIA NVLink™: 900GB/s PCIe Gen5: 128GB/s	NVIDIA NVLink: 600GB/s PCIe Gen5: 128GB/s

* With sparsity

Смотрим в спецификацию

	H100 SXM	H100 NVL
TF32 Tensor Core*	989 teraFLOPS	835 teraFLOPs
BFLOAT16 Tensor Core*	1,979 teraFLOPS	1,671 teraFLOPS
FP8 Tensor Core*	3,958 teraFLOPS	3,341 teraFLOPS
GPU Memory	80GB	94GB
GPU Memory Bandwidth	3.35TB/s	3.9TB/s
Interconnect	NVIDIA NVLink™: 900GB/s PCIe Gen5: 128GB/s	NVIDIA NVLink: 600GB/s PCIe Gen5: 128GB/s

* With sparsity

Смотрим в спецификацию

	H100 SXM	H100 NVL
TF32 Tensor Core*	~~989 teraFLOPS~~ 400 TFLOPS	835 teraFLOPs
BFLOAT16 Tensor Core*	~~1,979 teraFLOPS~~ 800 TFLOPS	1,671 teraFLOPS
FP8 Tensor Core*	~~3,958 teraFLOPS~~ 1600 TFLOPS	3,341 teraFLOPS
GPU Memory	80GB	94GB
GPU Memory Bandwidth	~~3.35TB/s~~ 2.4 TB/sec	3.9TB/s
Interconnect	NVIDIA NVLink™: ~~900GB/s~~ 400 + 400 GB/sec PCIe Gen5: 128GB/s	NVIDIA NVLink: 600GB/s PCIe Gen5: 128GB/s

* With sparsity

Что нужно для правильного построения логистики

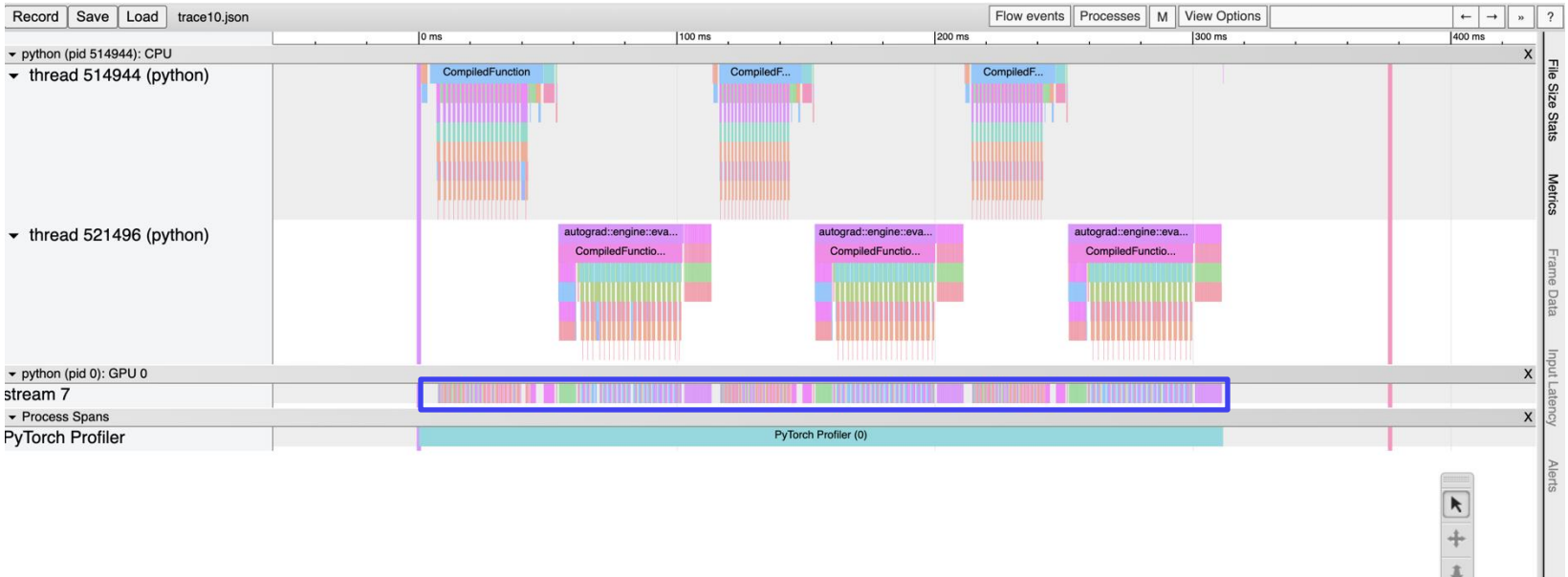
- Нужна карта маршрутов.
- Нужно понимание того, сколько стоит пройти тот или иной маршрут.

Как получить такие данные?

- Спросить у экспертов
- Почитать спецификацию
- Замерить самим

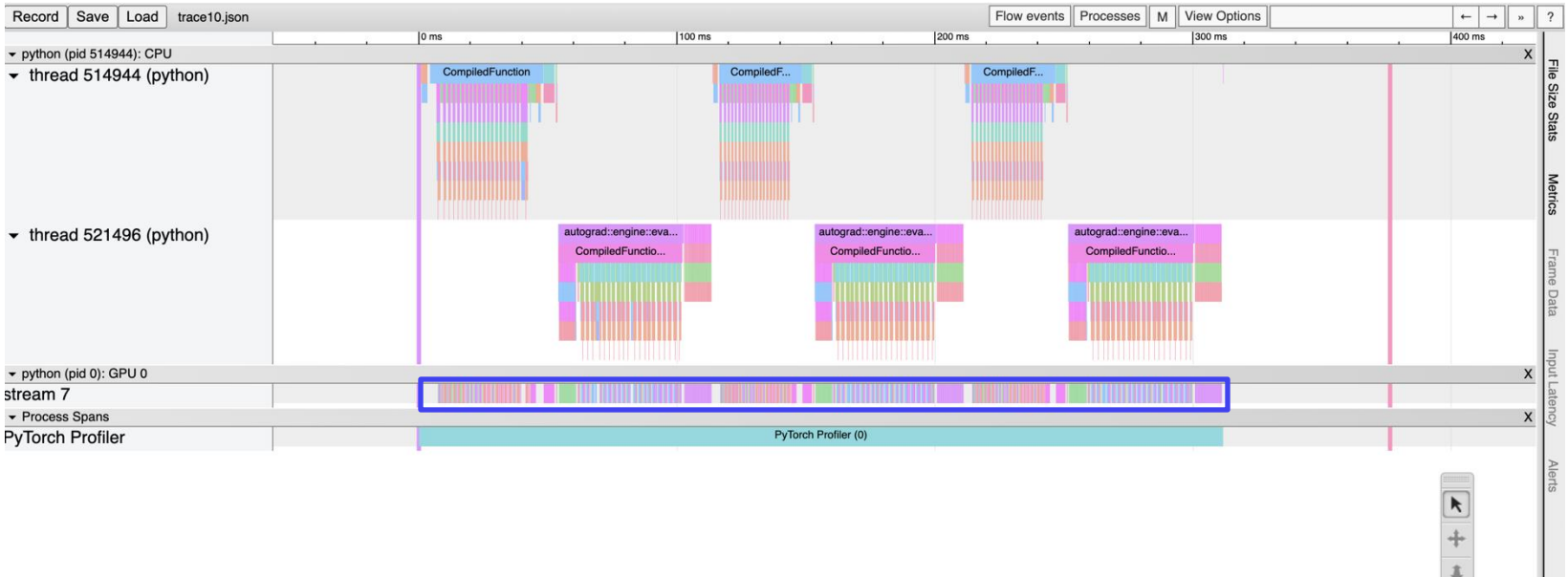
Как замерить время и память?

- Использовать torch.profiler или nsys



Как замерить время и память?

- Использовать torch.profiler или nsys. **Важно:** время CPU искажается.

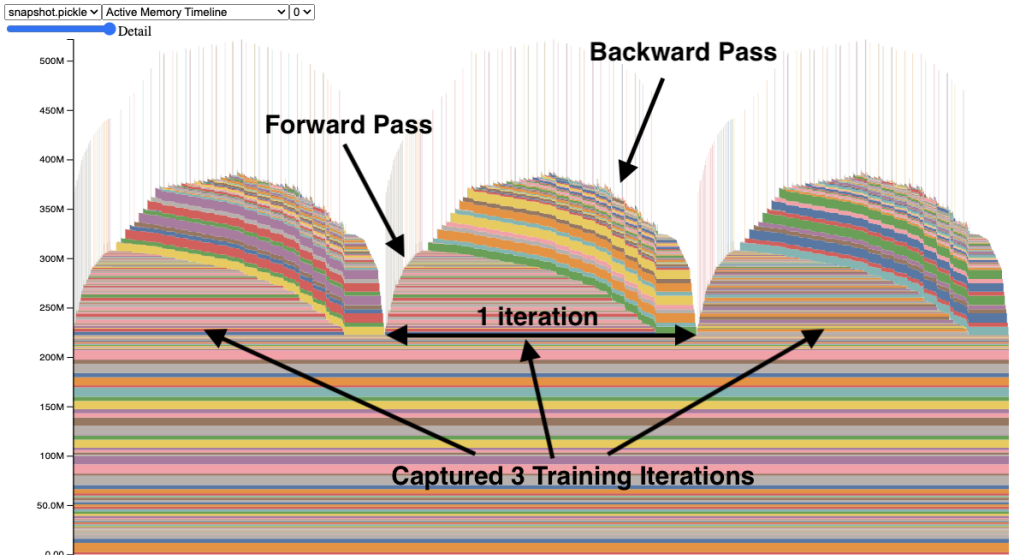


Как замерить время и память?

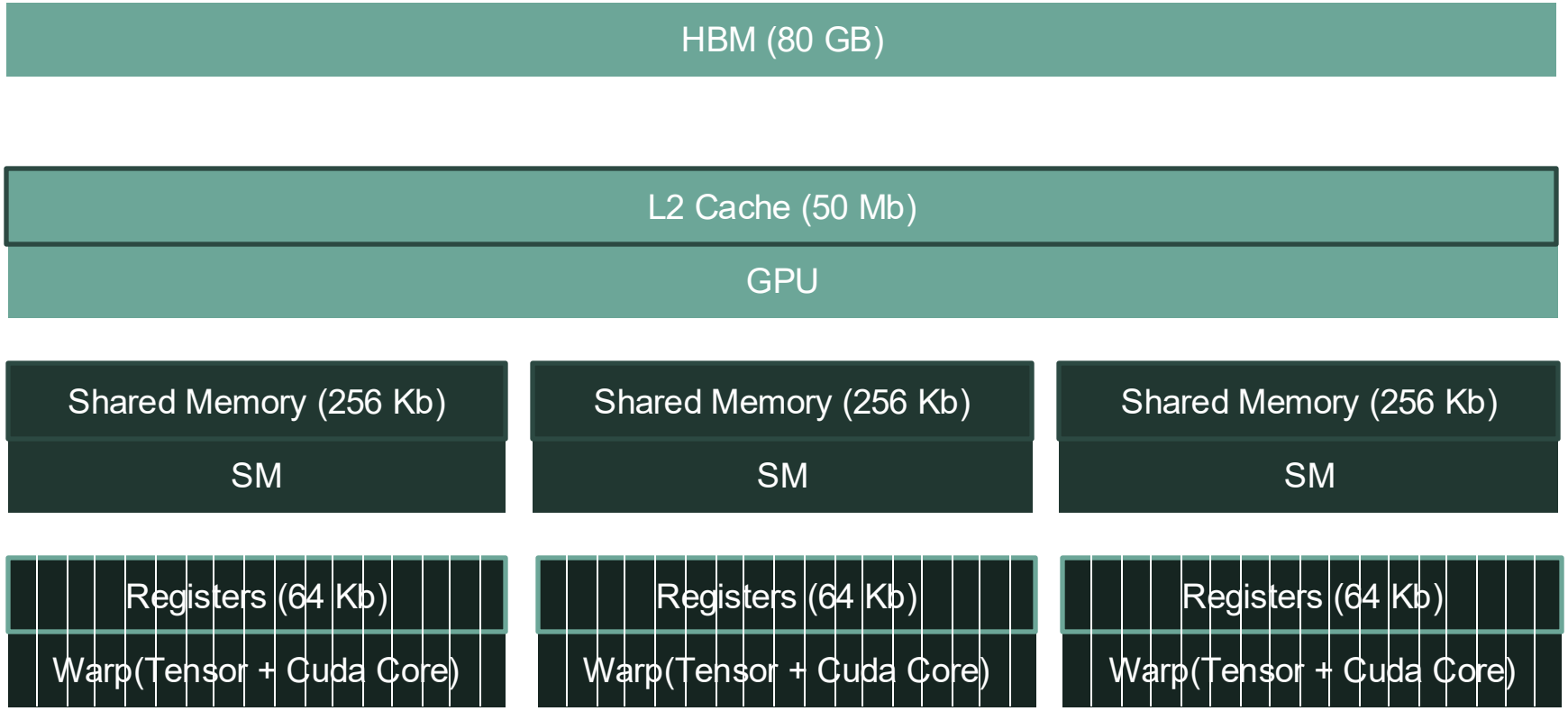
- Использовать torch.profiler или nsys. **Важно:** время CPU искажается.
- Снять профиль памяти

```
torch.cuda.empty_cache()
torch.cuda.memory._record_memory_history()

# ...

torch.cuda.memory._dump_snapshot(snapshot_path)
```

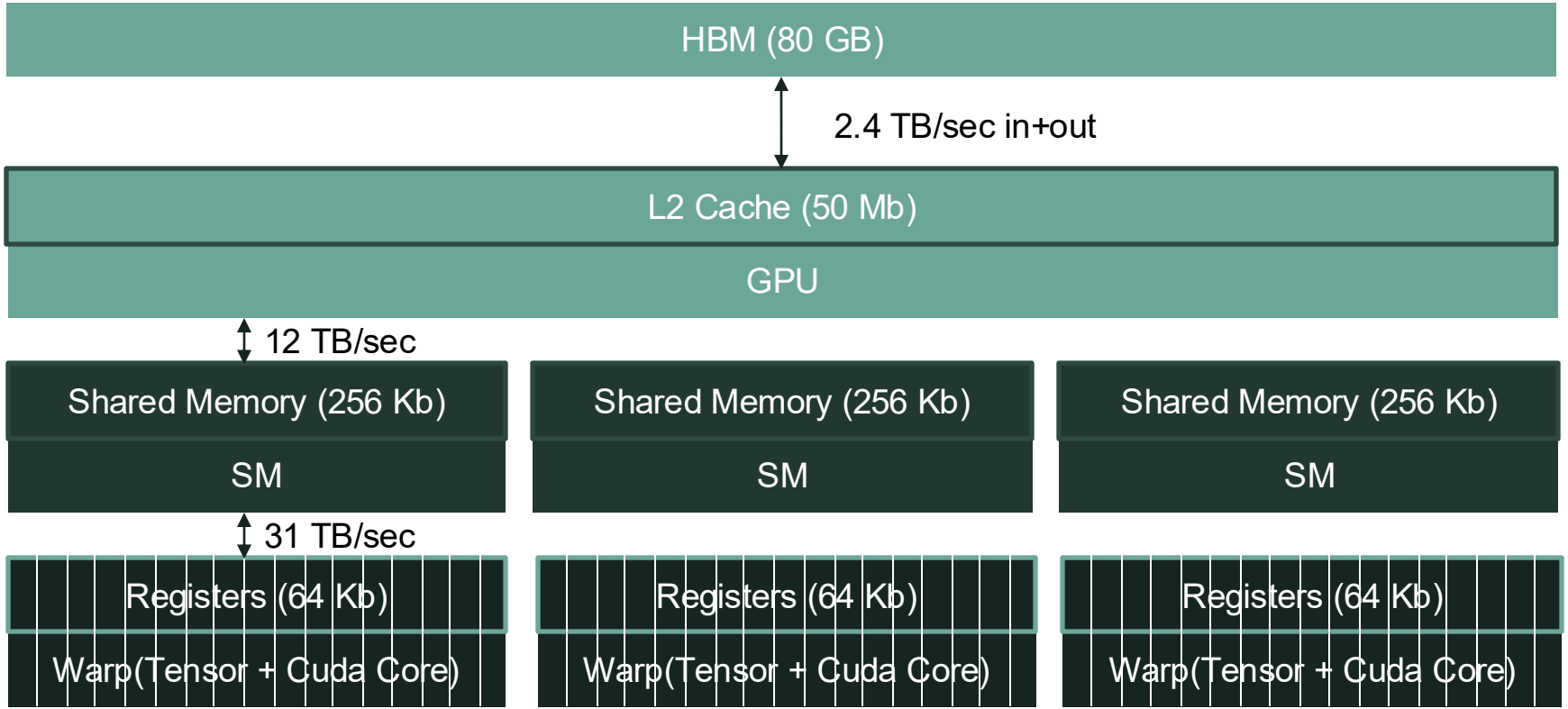
https://pytorch.org/memory_viz



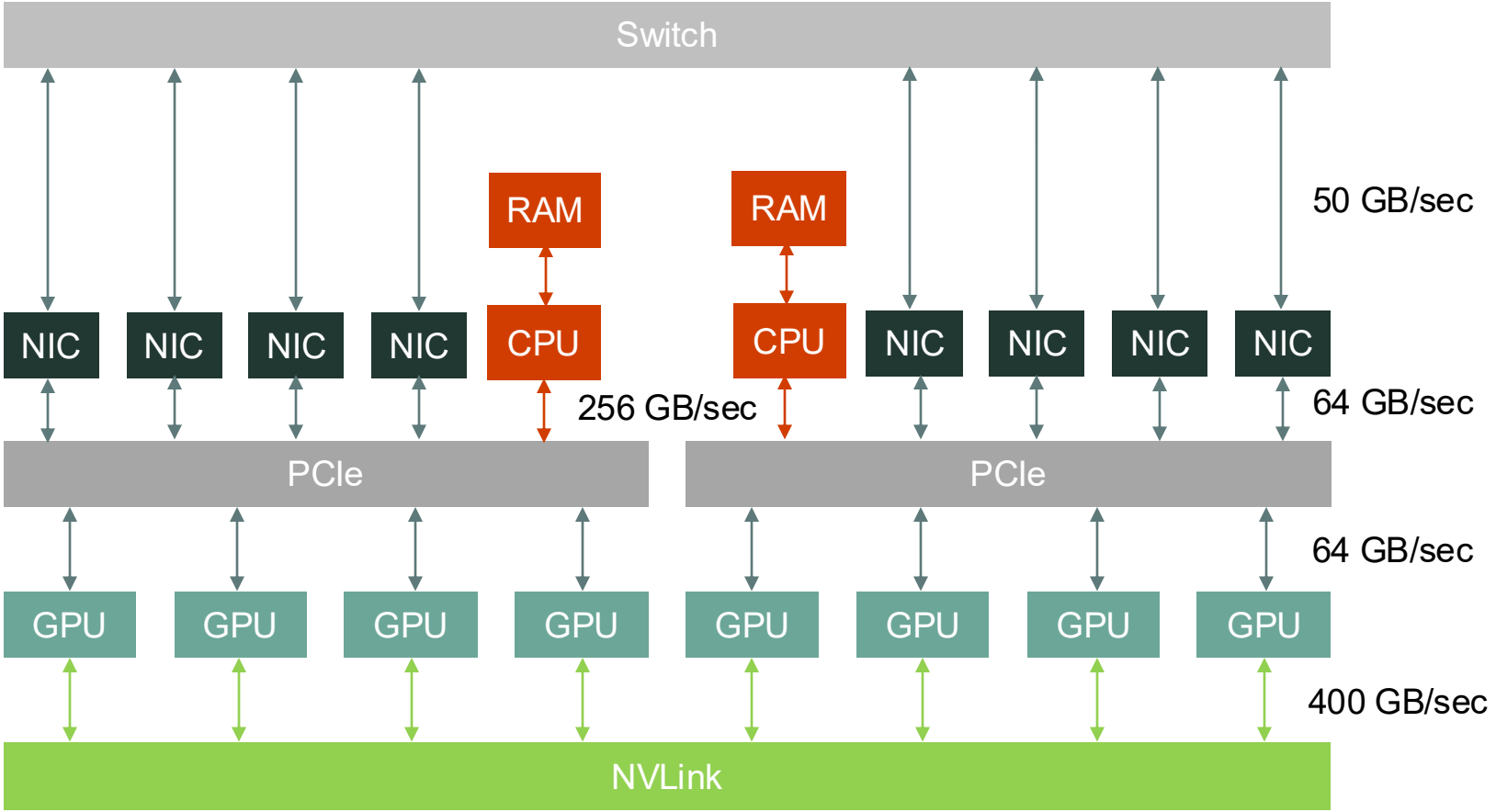
Карта GPU (H100)



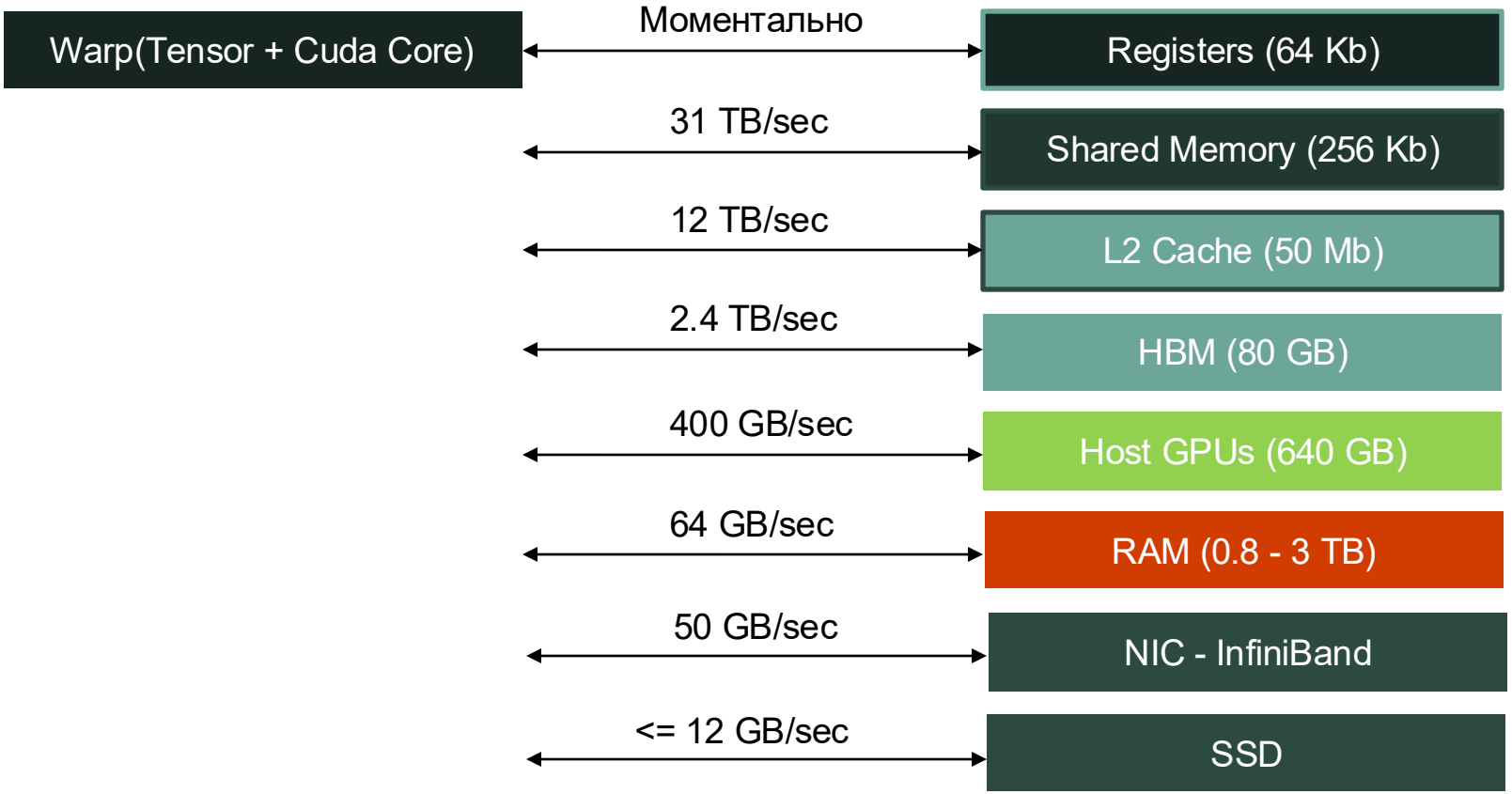
Карта GPU (H100)



Карта Хоста (H100 SXM)



Доступность памяти (H100)



Достаточно ли вводных?

С логистикой разобрались, осталось понять, сколько времени занимают вычисления на GPU:

- Вычисления на Tensor Cores (H100): 800 TFLOPS в bf16
- Вычисления на CUDA Cores (H100): <25 TFLOPS в bf16

Достаточно ли вводных?

С логистикой разобрались, осталось понять, сколько времени занимают вычисления на GPU:

- Вычисления на Tensor Cores (H100): 800 TFLOPS в bf16
- Вычисления на CUDA Cores (H100): <25 TFLOPS в bf16

Мы построили карту логистики данных. Пора делать что-то конкретное!

02

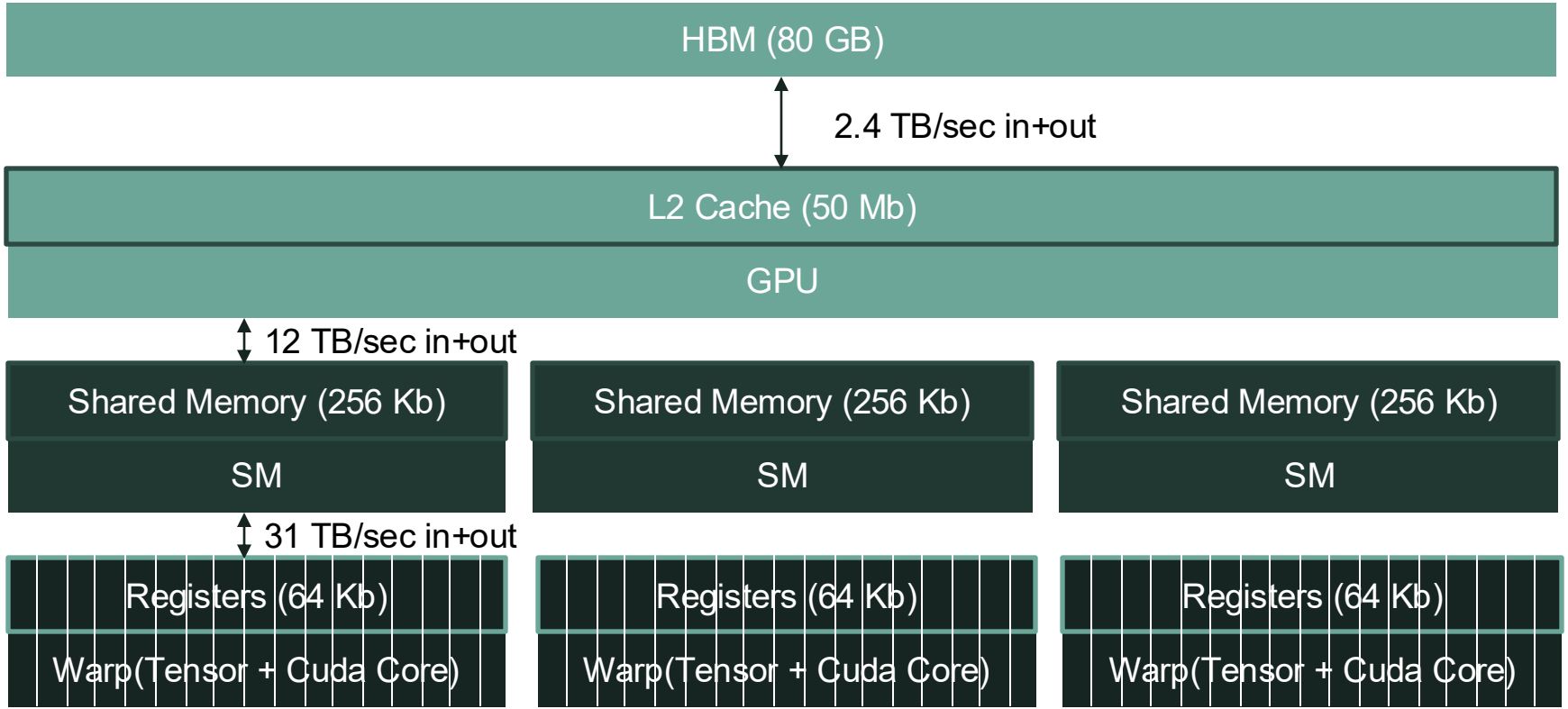
Локальная логистика на GPU

Упражнение 1

Пусть скорость HBM-памяти - 2.4 TB/sec, а скорость GPU в BF16 – 800TFLOPS. Сколько по времени займет такая операция в BF16 (размерности тензоров – 8192x8192):

$$C = A + B$$

Карта GPU (H100)



Упражнение 1

Пусть скорость HBM-памяти - 2.4 TB/sec, а скорость GPU в BF16 – 800TFLOPS. Сколько по времени займет такая операция в BF16 (размерности тензоров – 8192x8192):

C = A + B

- Время на загрузку и выгрузку: 8192*8192*2*(2+1) / 2.4e12 = 168 мкрсек

Достаточно ли вводных?

С логистикой разобрались, осталось понять, сколько времени занимают вычисления на GPU:

- Вычисления на Tensor Cores (H100): 800 TFLOPS в bf16
- Вычисления на CUDA Cores (H100): <25 TFLOPS в bf16

Упражнение 1

Пусть скорость HBM-памяти - 2.4 TB/sec, а скорость GPU в BF16 – 800TFLOPS. Сколько по времени займет такая операция в BF16 (размерности тензоров – 8192x8192):

C = A + B

- Время на загрузку и выгрузку: 8192*8192*2*(2+1) / 2.4e12 = 168 мкрсек
- Время на вычисление: 8192*8192 / 25e12 = 2.6 мкрсек

Упражнение 1

Пусть скорость HBM-памяти - 2.4 TB/sec, а скорость GPU в BF16 – 800TFLOPS. Сколько по времени займет такая операция в BF16 (размерности тензоров – 8192x8192):

$$C = A + B$$

- Время на загрузку и выгрузку: 8192*8192*2*(2+1) / 2.4e12 = 168 мкрсек
- Время на вычисление: 8192*8192 / 25e12 = 2.6 мкрсек
- Итоговое время выполнения: 168мкрсек – операция memory bound

Упражнение 2

Пусть скорость HBM-памяти - 2.4 TB/sec, а скорость GPU в BF16 – 800TFLOPS. Сколько по времени займет такая операция в BF16 (размерности тензоров – 1024x1024):

C = A @ B

- Время на загрузку и выгрузку: 1024*1024*2*(2+1) / 2.4e12 = 2.6 мкрсек
- Время на вычисление: 1024*1024*1024*2 / 800e12 = 2.6 мкрсек
- Ожидаемое время выполнения: 2.6 мкрсек

Упражнение 3

Пусть скорость HBM-памяти - 2.4 TB/sec, а скорость GPU в BF16 – 800TFLOPS. Сколько по времени займет такая операция в BF16 (размерности тензоров – A - 4x8192, B – 8192x8192):

C = A @ B

- Время на загрузку и выгрузку: (8192*8192*2 + 8192*4*2*2) / 2.4e12 = 56 мкрсек
- Время на вычисление: 4*8192*8192*2 / 800e12 = 0.6 мкрсек
- Ожидаемое время выполнения: 56 мкрсек

Упражнение 3

Пусть скорость HBM-памяти - 2.4 TB/sec, а скорость GPU в BF16 – 800TFLOPS. Сколько по времени займет такая операция в BF16 (размерности тензоров – A - 4x8192, B – 8192x8192):

C = A @ B

- Время на загрузку и выгрузку: (8192*8192*2 + 8192*4*2*2) / 2.4e12 = 56 мкрсек
- Время на вычисление: 4*8192*8192*2 / 800e12 = 0.6 мкрсек
- Ожидаемое время выполнения: 56 мкрсек

При большом дизбалансе матричные умножения могут быть Memory Bound

Пример Attention

$$Attention(Q, K, V) = softmax(\frac{QK^T}{\sqrt{d_k}})V$$

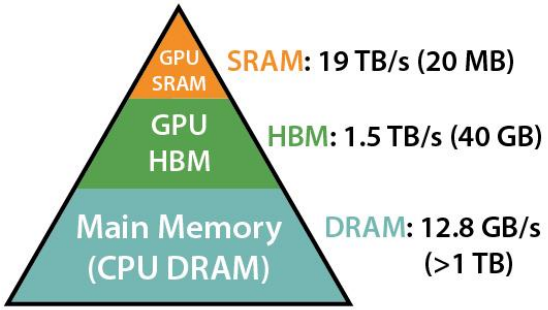
Упражнение 4. Attention

$$Attention(Q, K, V) = softmax(\frac{QK^T}{\sqrt{d_k}})V$$

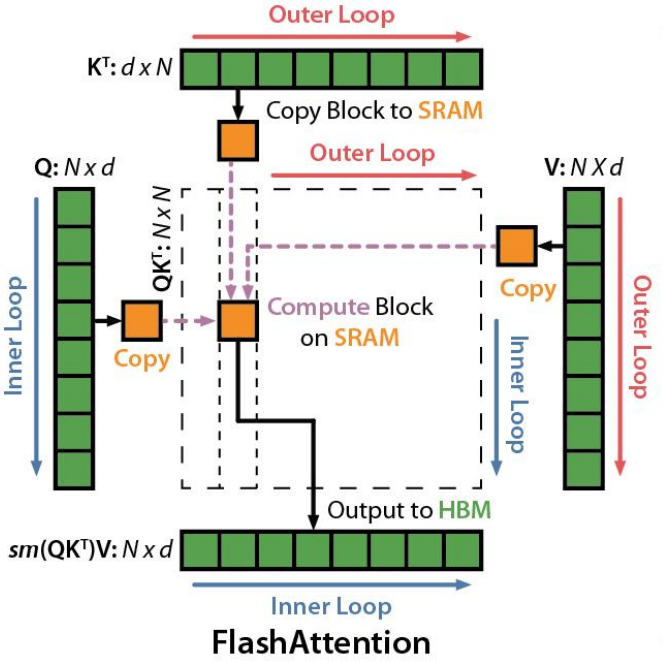
S = 8192, H = 4096, nh = 64

- Линейные слои: 4*S*H*H*2 / 800e12 = 1.4 ms
- Матричные умножения в attention: 2*H*S*S*(2+1) / 800e12 = 2 ms
- Scaling, softmax: 2*nh*S*S*2 / 2.4e12 = 7.1 ms

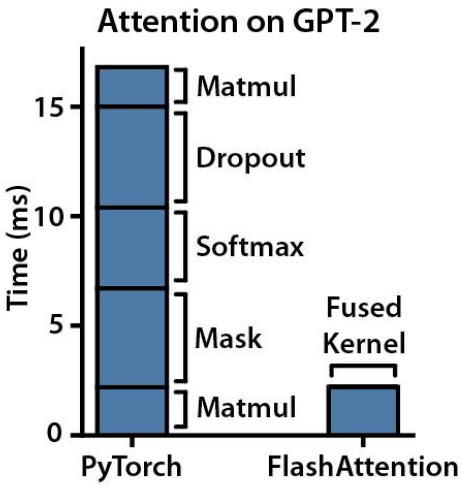
Flash Attention



Memory Hierarchy with Bandwidth & Memory Size



FlashAttention



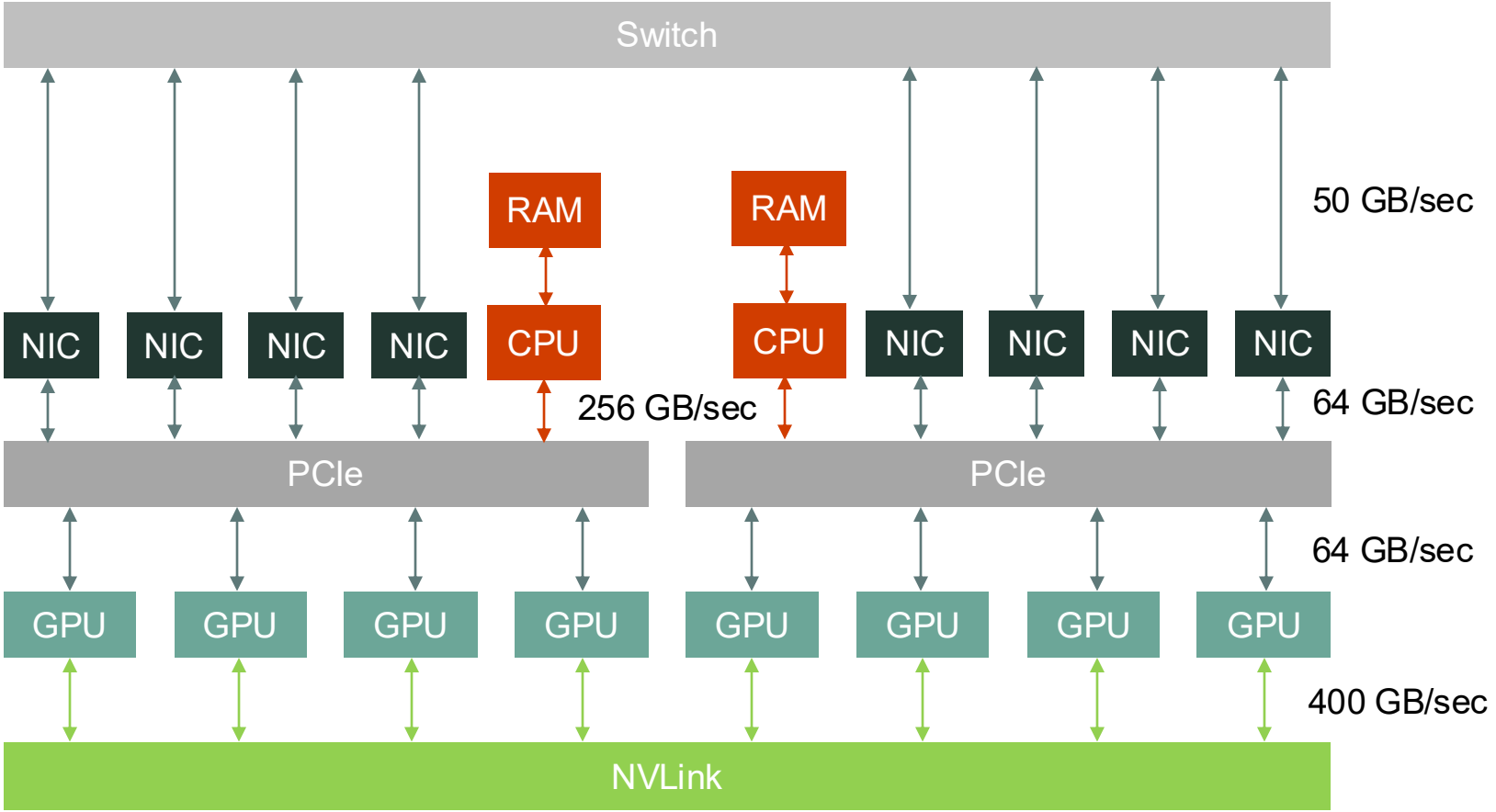
Выводы

- Даже на GPU важно просчитывать логистику;
- Memory bound операции могут занимать существенное время, не загружая GPU;
- Матричные умножения могут стать неэффективными на маленьких размерностях или при дизбалансе размеров матриц

03

Коммуникации между GPU

Карта Хоста (H100 SXM)

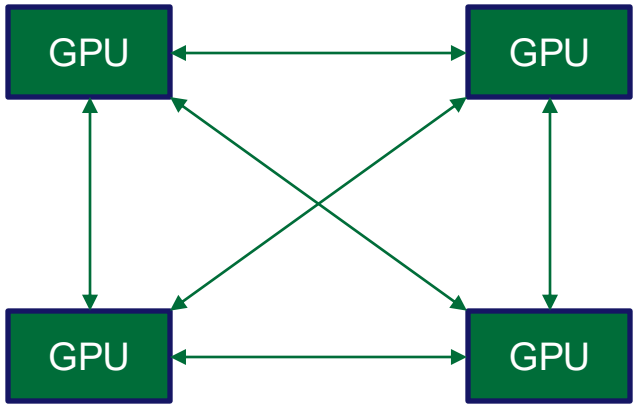


Типы коммуникаций

- Симметричные
- Ассиметричные

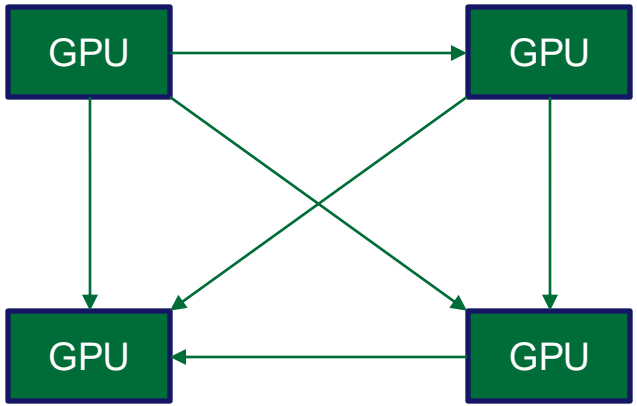
Типы коммуникаций

- **Симметричные**
- Ассиметричные



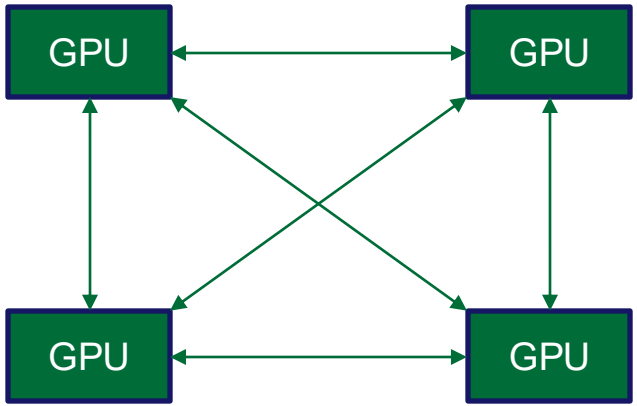
Типы коммуникаций

- Симметричные
- **Ассиметричные**

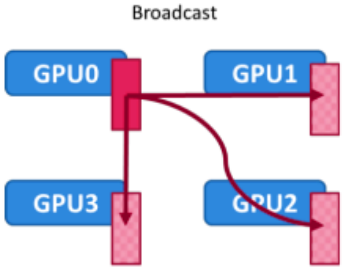


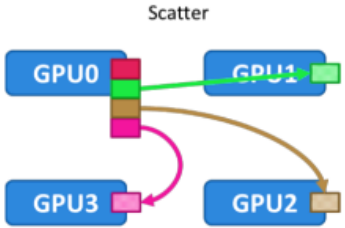
Типы коммуникаций

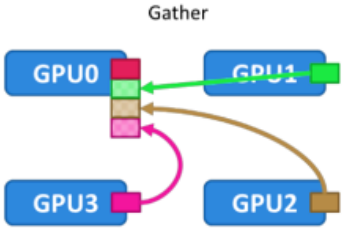
- **Симметричные**
- Ассиметричные

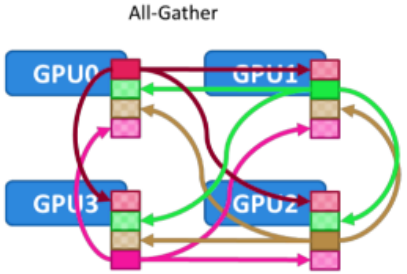


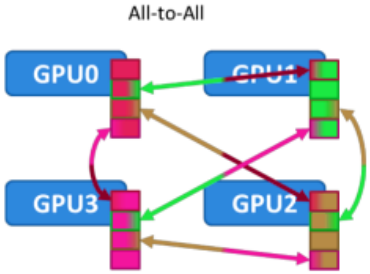
NCCL

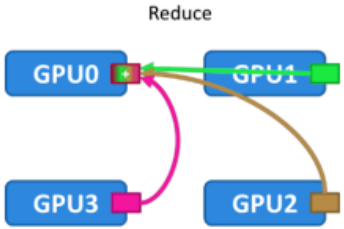

Broadcast
GPU0
GPU1
GPU3
GPU2

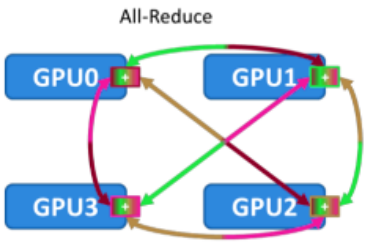

Scatter
GPU0
GPU1
GPU3
GPU2


Gather
GPU0
GPU1
GPU3
GPU2

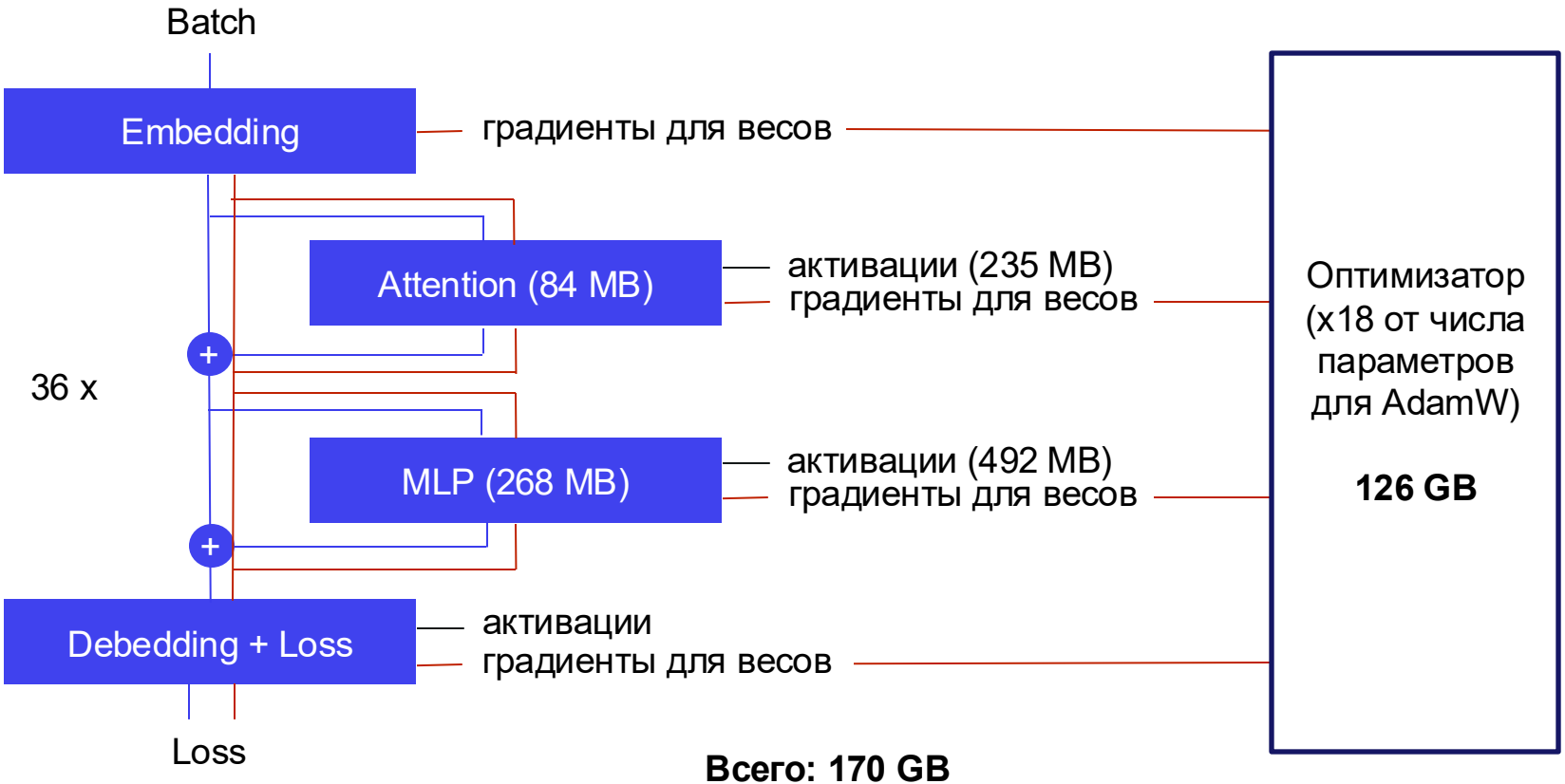

All-Gather
GPU0
GPU1
GPU3
GPU2


All-to-All
GPU0
GPU1
GPU3
GPU2


Reduce
GPU0
GPU1
GPU3
GPU2


All-Reduce
GPU0
GPU1
GPU3
GPU2

7B модель

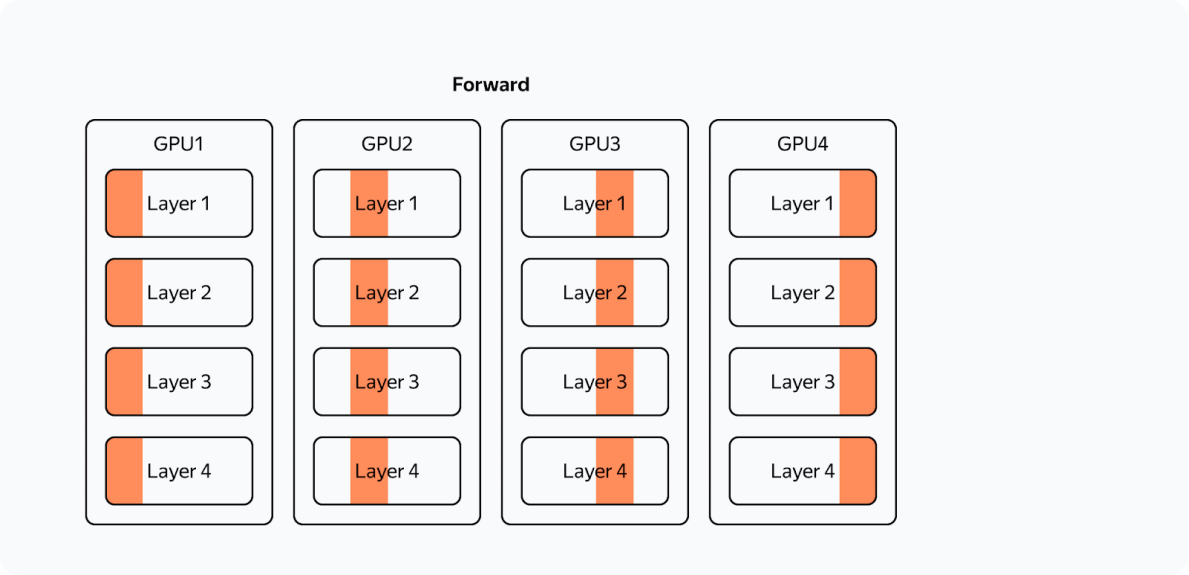

Batch
Embedding
градиенты для весов
Attention (84 MB)
активации (235 MB)
градиенты для весов
36 x
MLP (268 MB)
активации (492 MB)
градиенты для весов
Debedding + Loss
активации
градиенты для весов
Loss
Оптимизатор (x18 от числа параметров для AdamW)
126 GB
Всего: 170 GB

Бьем состояния оптимизатора на разные GPU

- А заодно веса
- Собираем веса перед каждым forward
- Усредняем градиенты после каждого backward

FSDP

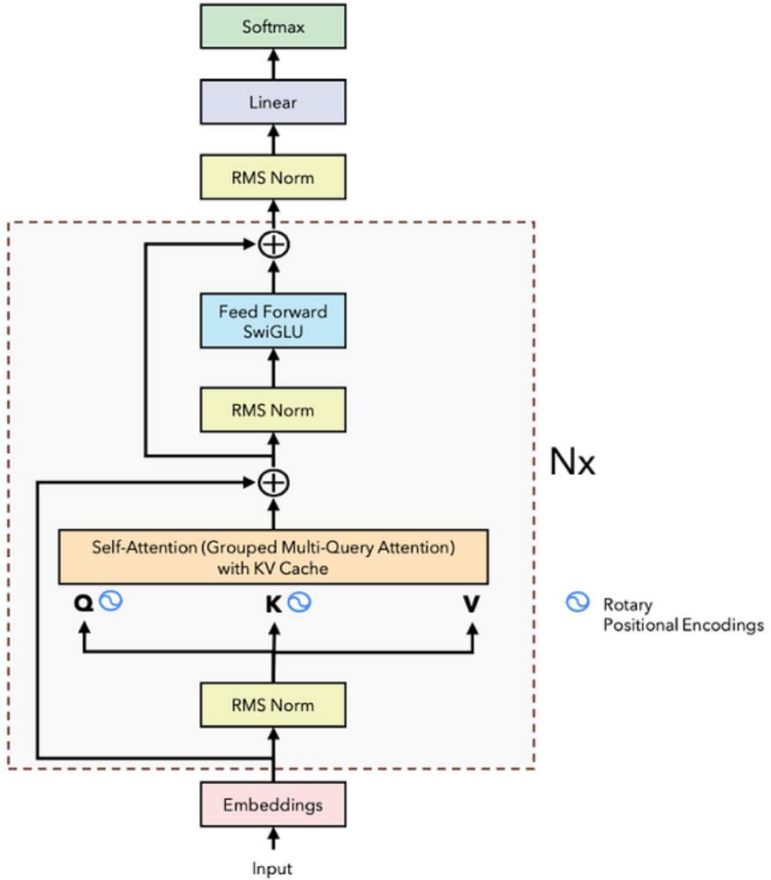
- Разделим веса и состояния оптимизатора между процессами
- Сделаем коммуникации асинхронными



FSDP

- Разделим веса и состояния оптимизатора между процессами
- Сделаем коммуникации асинхронными
- Есть версия YaFSDP, оптимизированная под compute bound сценарии

Математика (Llama 7B)



Время forward одного слоя 7B

- S = 8192 токенов, H = 4096, I = 11008, nh*dh = 4096
- Суммарное время матричных умножений:

$$S*H*(nh*dh*4*2 + I*3*2 + S*(2+1)) / 800e12 = 5.1 \text{ ms}$$

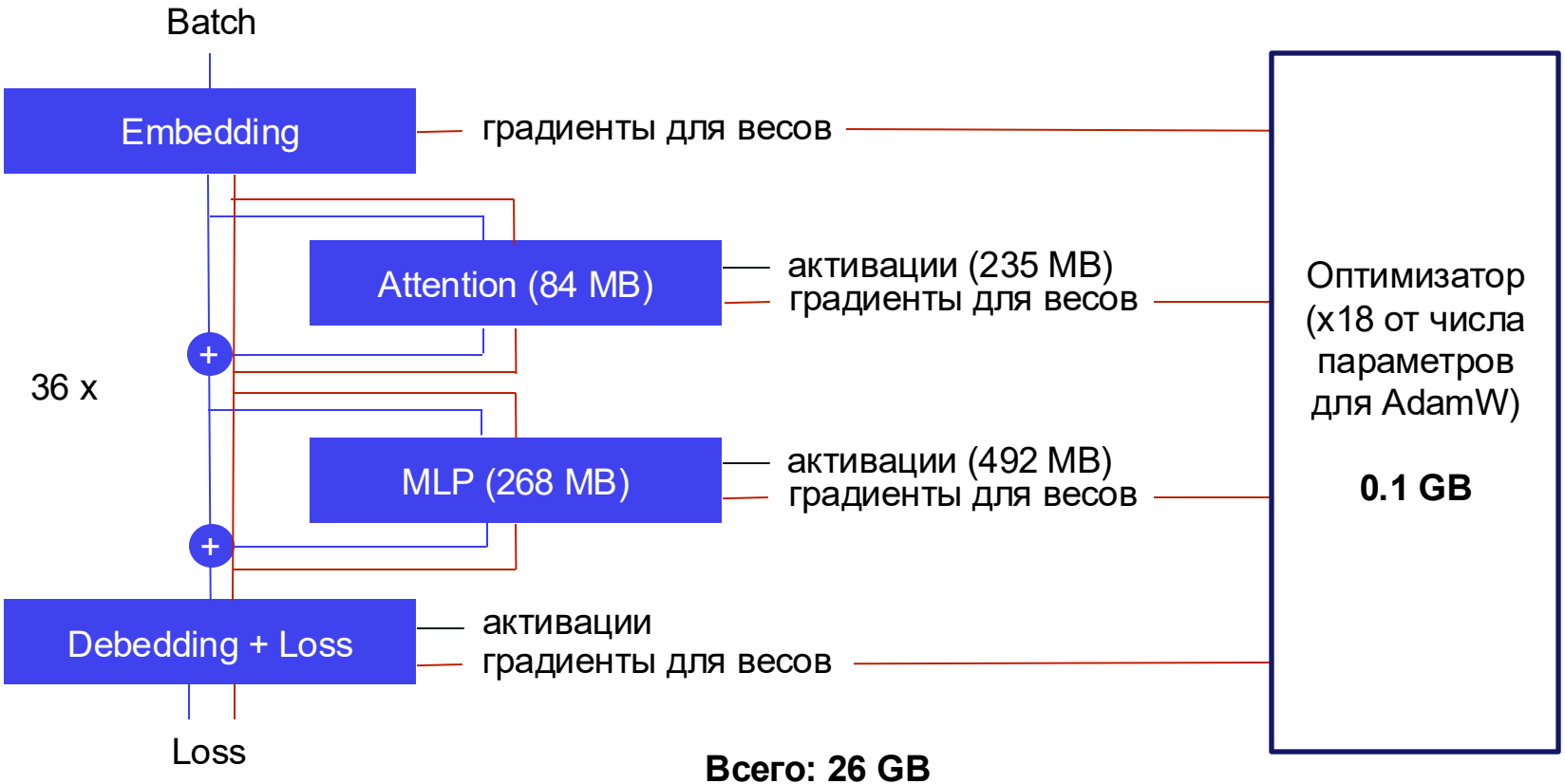
- Время memory bound операций (2 RMSNorm, 2 Add, 1 SwiGLU):

$$(S*H*2*(2*2 + 2*3) + S*I*2*3) / 2.4e12 = 0.5 \text{ ms}$$

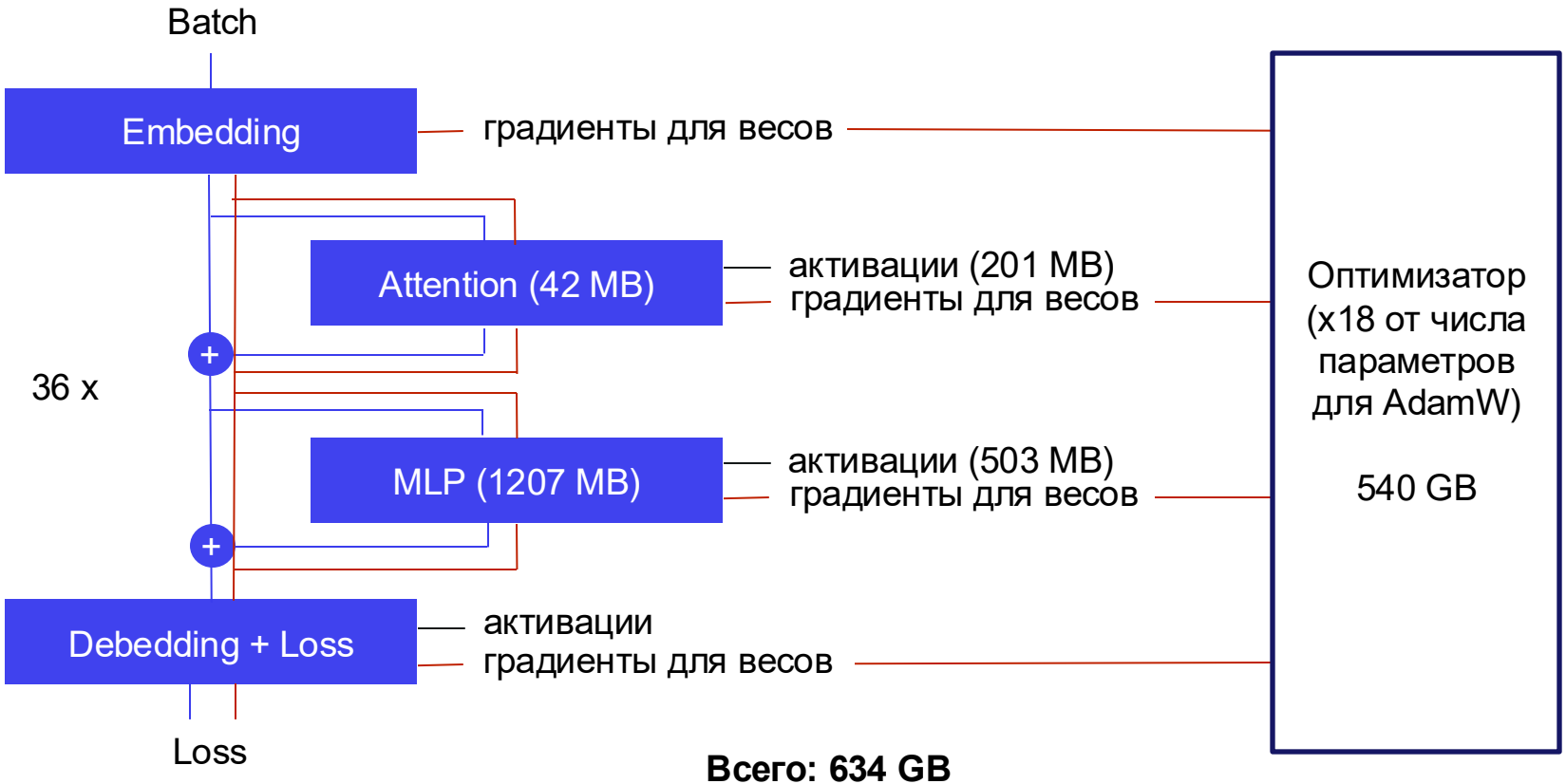
- Время на AllGather FSDP (в одном слое 540MB)

$$(H*nh*dh*4 + H*I*3)*2 / 400e9 = 1.34 \text{ ms}$$

7B модель на 1024 GPU

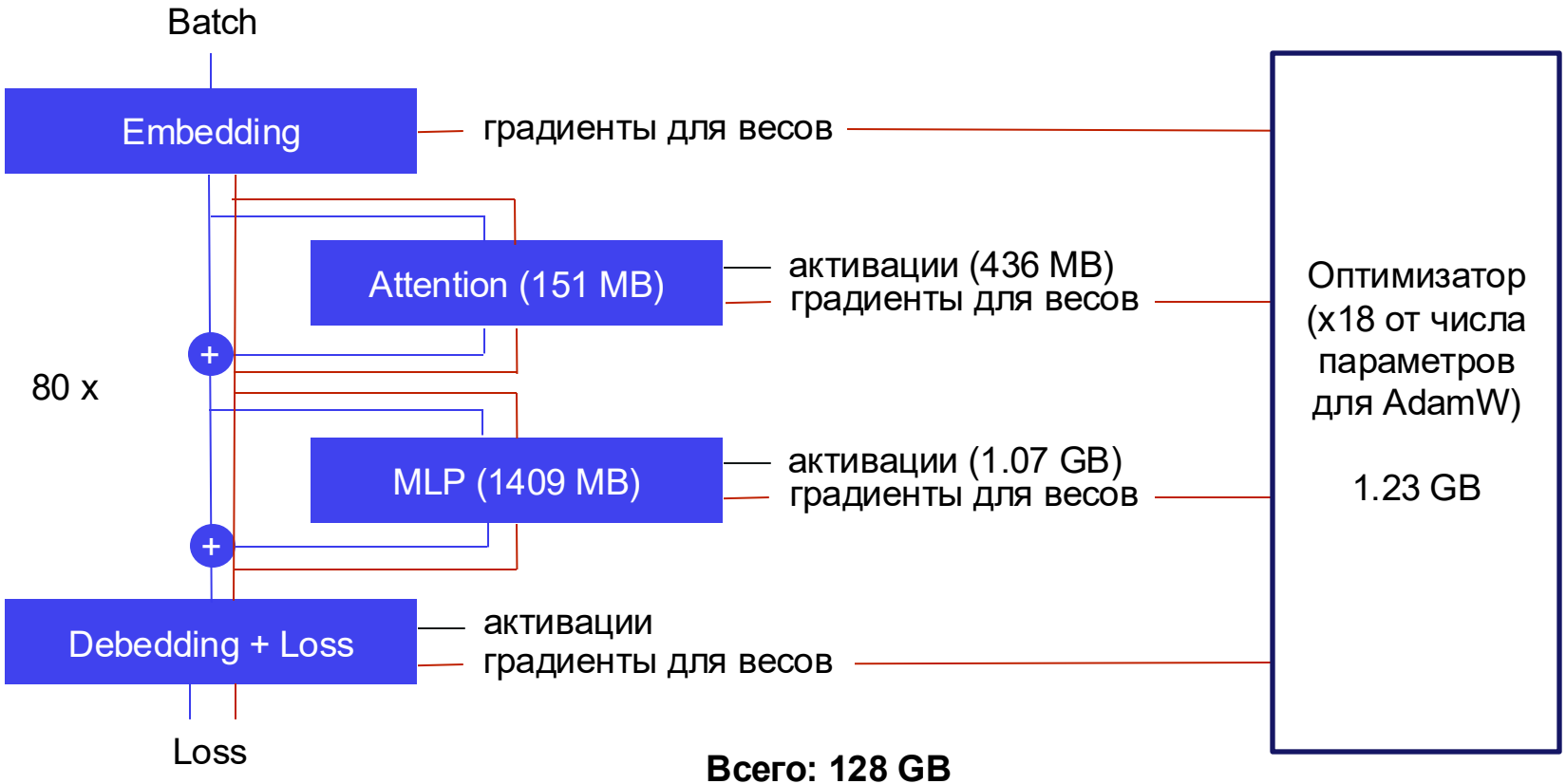


Qwen 30B-A3B



Всего: 634 GB

Llama 70B на 1024 GPU

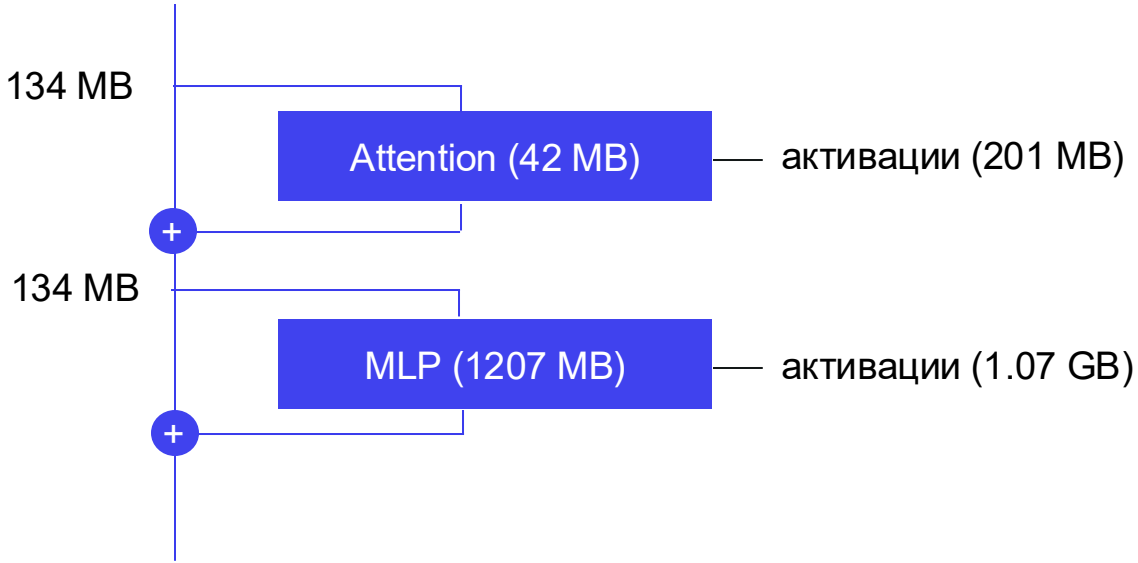


Всего: 128 GB
Активации: 120.8 GB

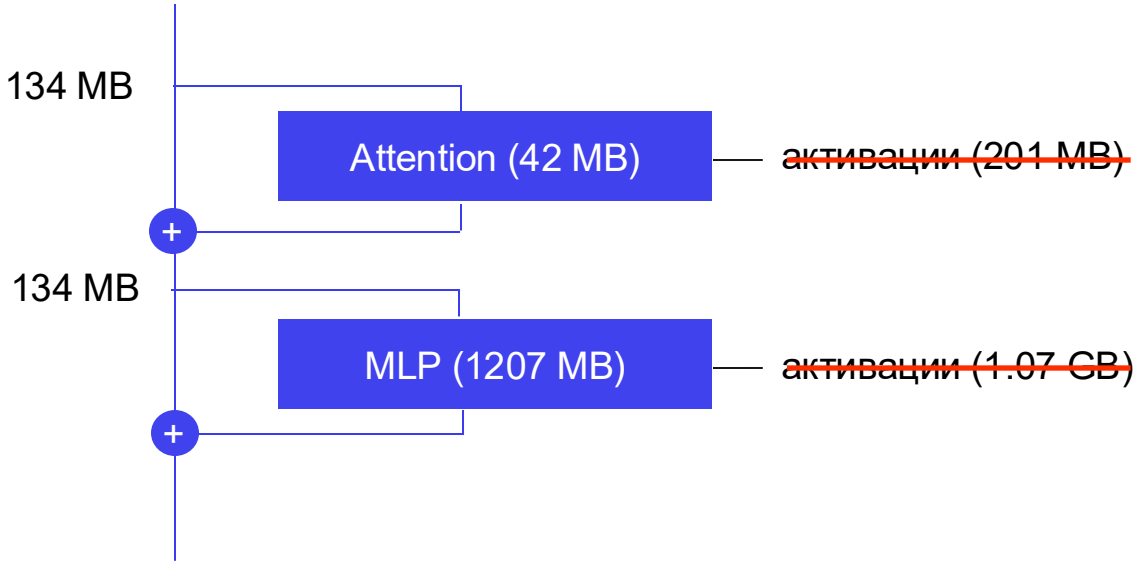
2 варианта логистики активаций

- Через перевычисление (чекпоинт активаций)
- Через пересылку

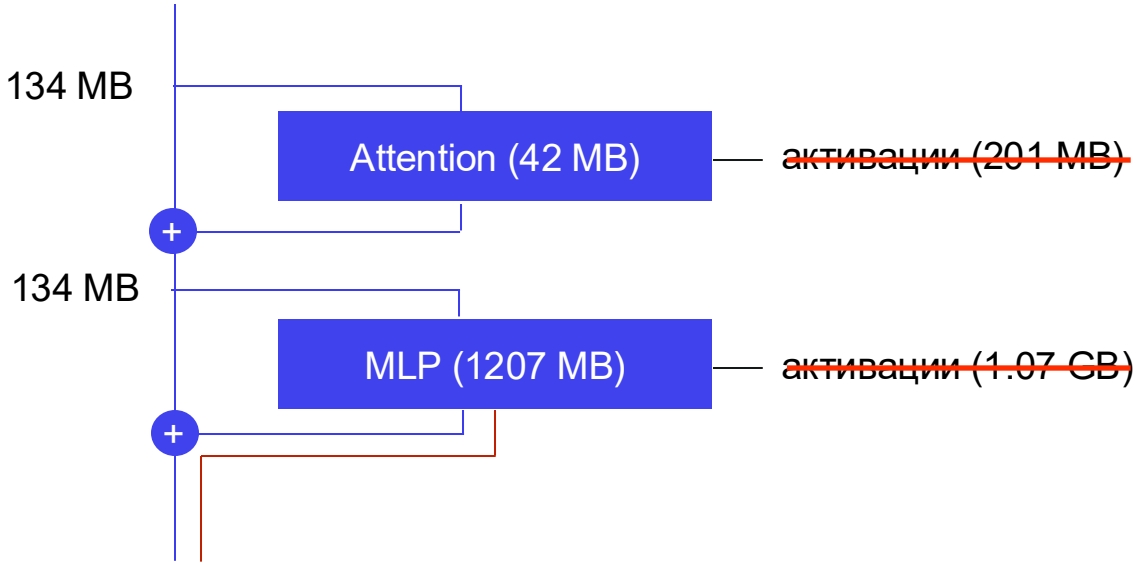
Чекпоинт активаций



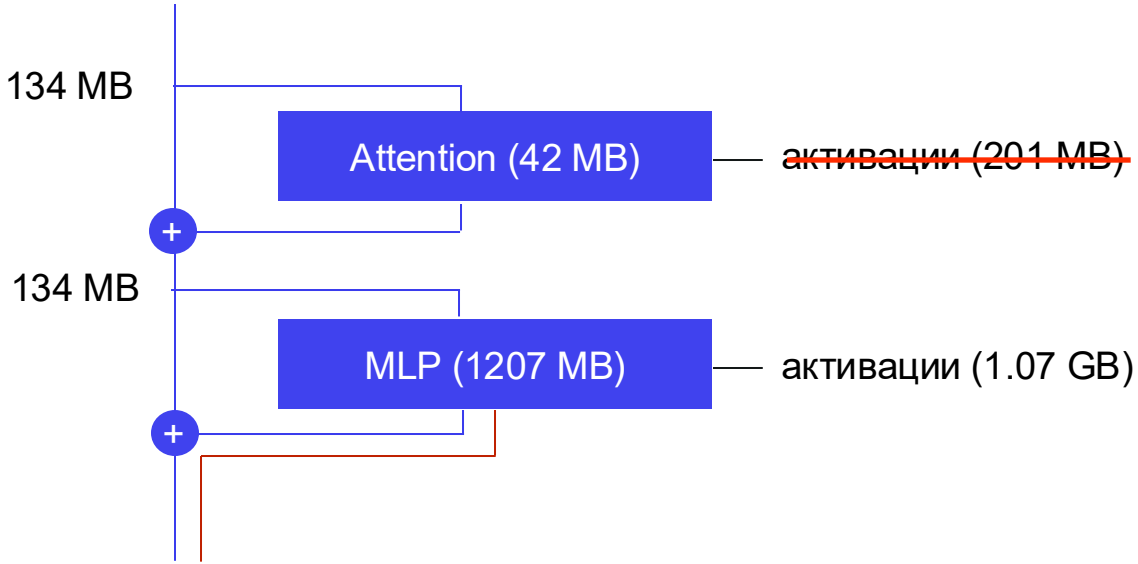
Чекпоинт активаций



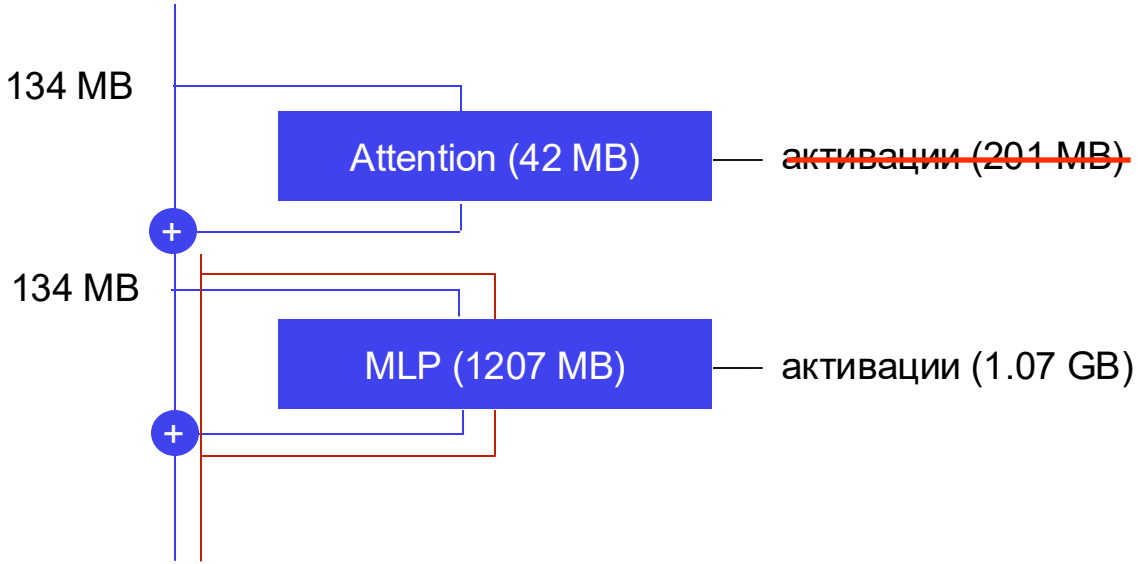
Чекпоинт активаций



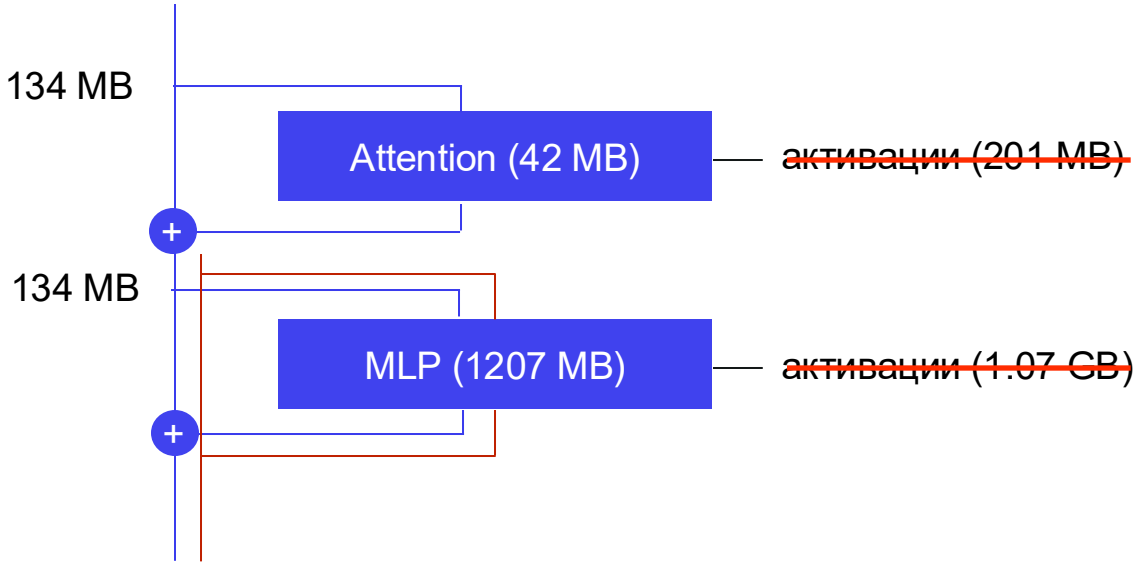
Чекпоинт активаций



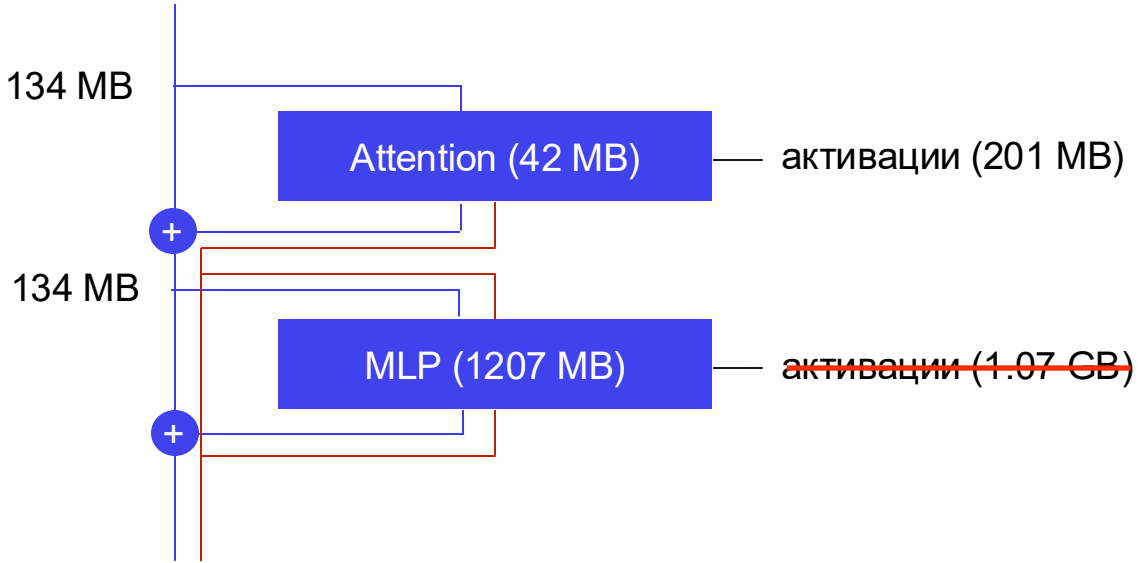
Чекпоинт активаций



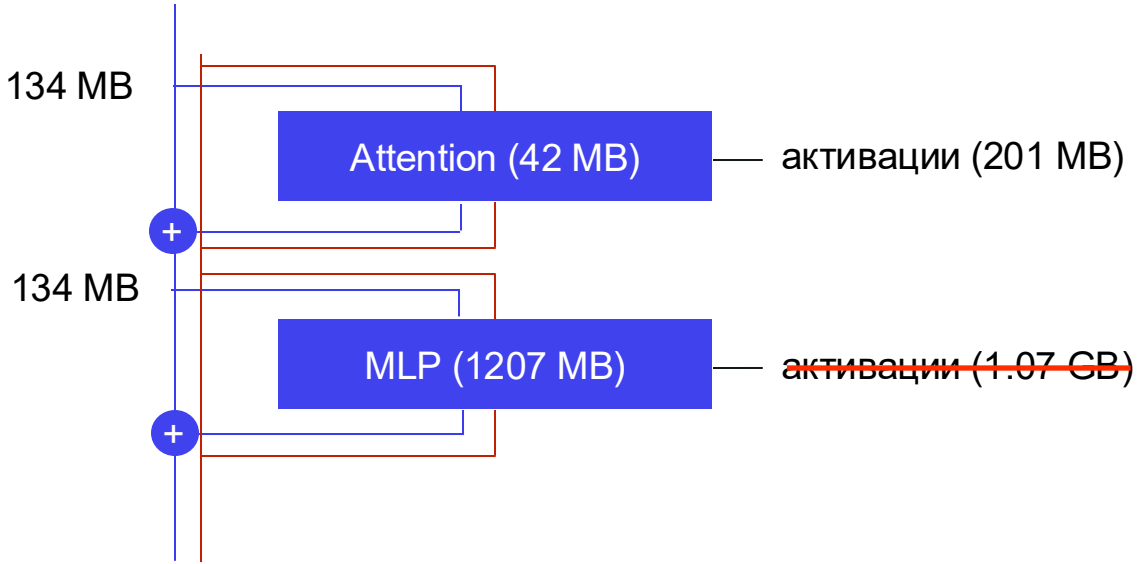
Чекпоинт активаций



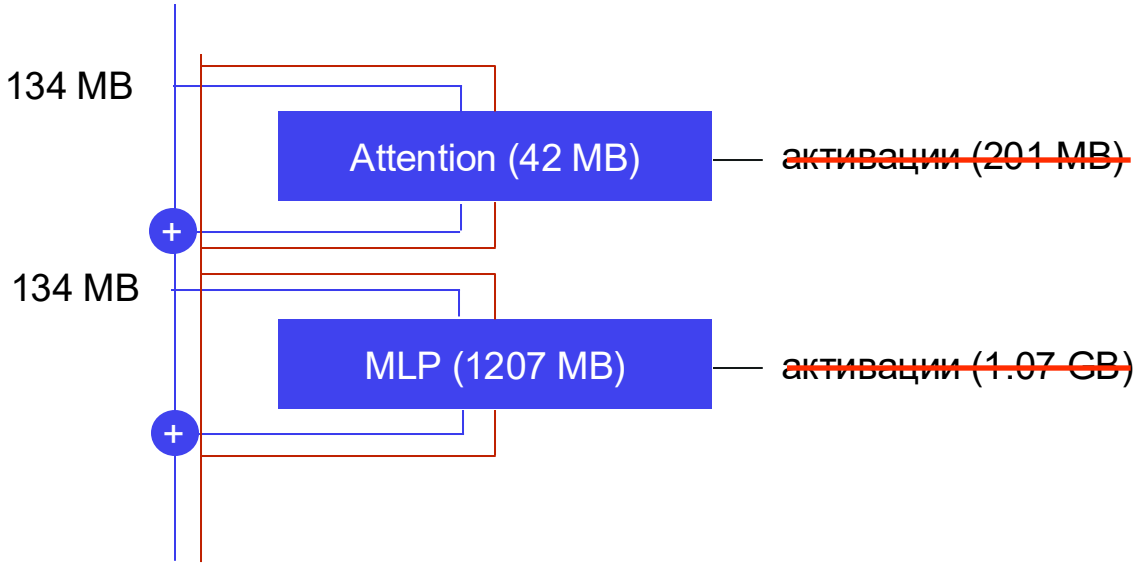
Чекпоинт активаций



Чекпоинт активаций

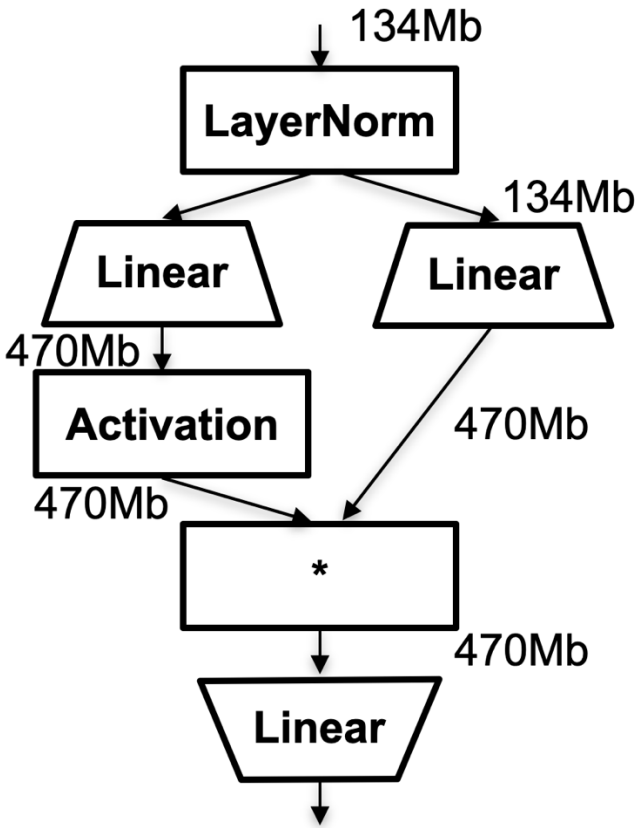


Чекпоинт активаций



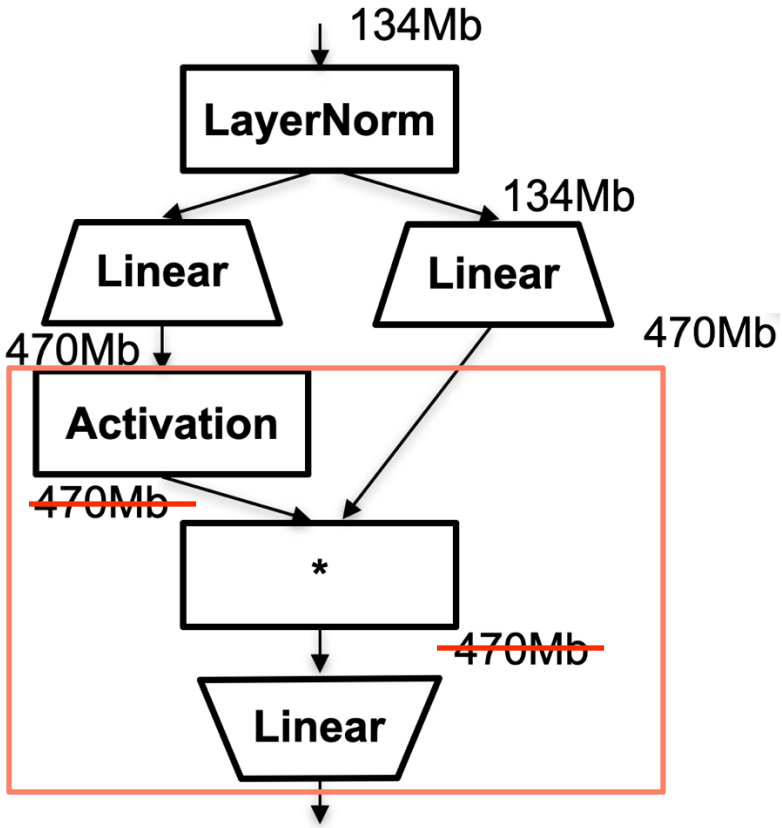
Частичный рекомпьют

- Не все активации одинаково дорого вычислять



Частичный рекомпьют

- Не все активации одинаково дорого вычислять



2 варианта логистики активаций

- Через перевычисление (чекпоинт активаций)
- Через пересылку (тензорный параллелизм)

Тензорный параллелизм

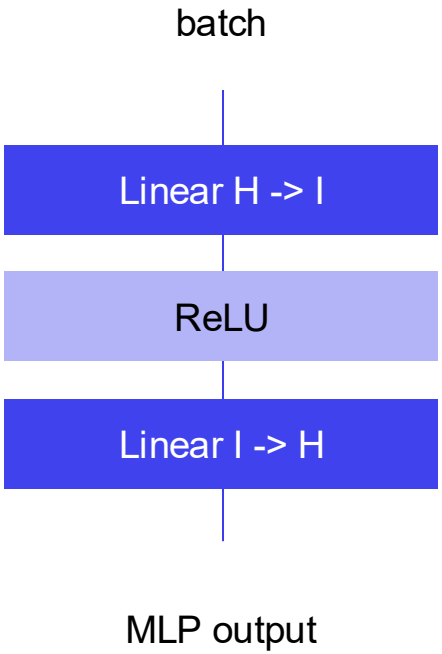
Как было раньше:

- Шардируем только веса, собираем их по необходимости

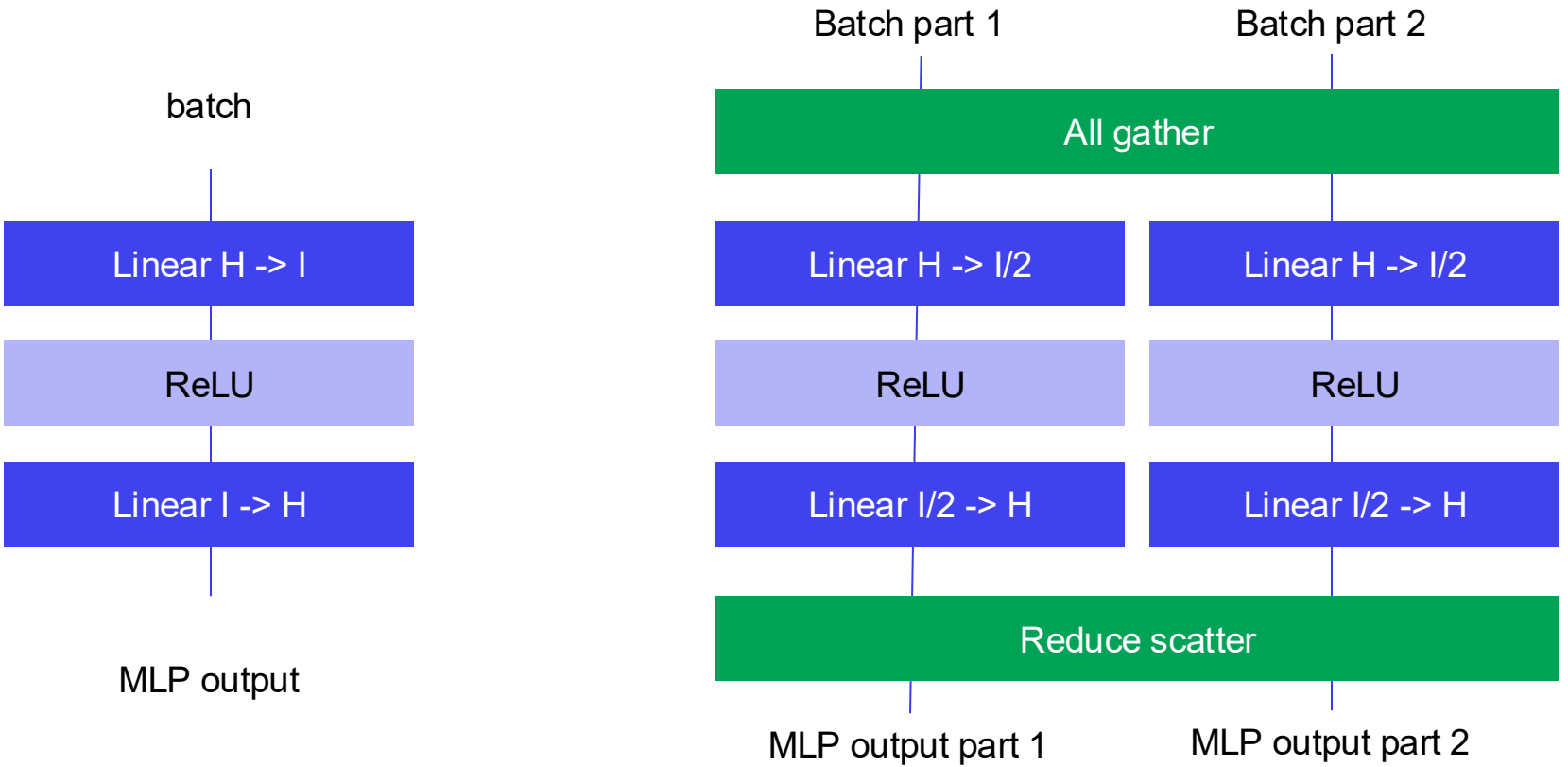
Как хотим:

- Шардируем веса и активации, собираем их

Тензорный параллелизм



Тензорный параллелизм



Время forward одного слоя 70B (TP=2)

- S = 4096 токенов, H = 8192, I = 28672, nh*dh = 8192, kvh = 4
- Суммарное время матричных умножений:

$$S*H*((nh + kvh)*dh*2 + I*3 + S*(2+1))*2 / 800e12 = 10ms$$

- Время memory bound операций (2 RMSNorm, 2 Add, 1 SwiGLU):

$$(S*H*2*(2*2 + 2*3) + S*I*2*3) / 2.4e12 = 0.55ms$$

- Время на AllGather FSDP (в одном слое 1560 MB)

$$(H*(nh + kvh)*dh*2 + H*I*3)*2 / 400e9 = 4ms$$

- Время TP коммуникаций (NVLink)

$$S*H*TP*2*2 / 400e9 = 1.3ms$$

Время forward одного слоя 70B (TP=2)

- S = 4096 токенов, H = 8192, I = 28672, nh*dh = 8192, kvh = 4
- Суммарное время матричных умножений:

$$S*H*((nh + kvh)*dh*2 + I*3 + S*(2+1))*2 / 800e12 = 10ms$$

- Время memory bound операций (2 RMSNorm, 2 Add, 1 SwiGLU):

$$(S*H*2*(2*2 + 2*3) + S*I*2*3) / 2.4e12 = 0.55ms$$

- Время на AllGather FSDP (в одном слое 1560 MB)

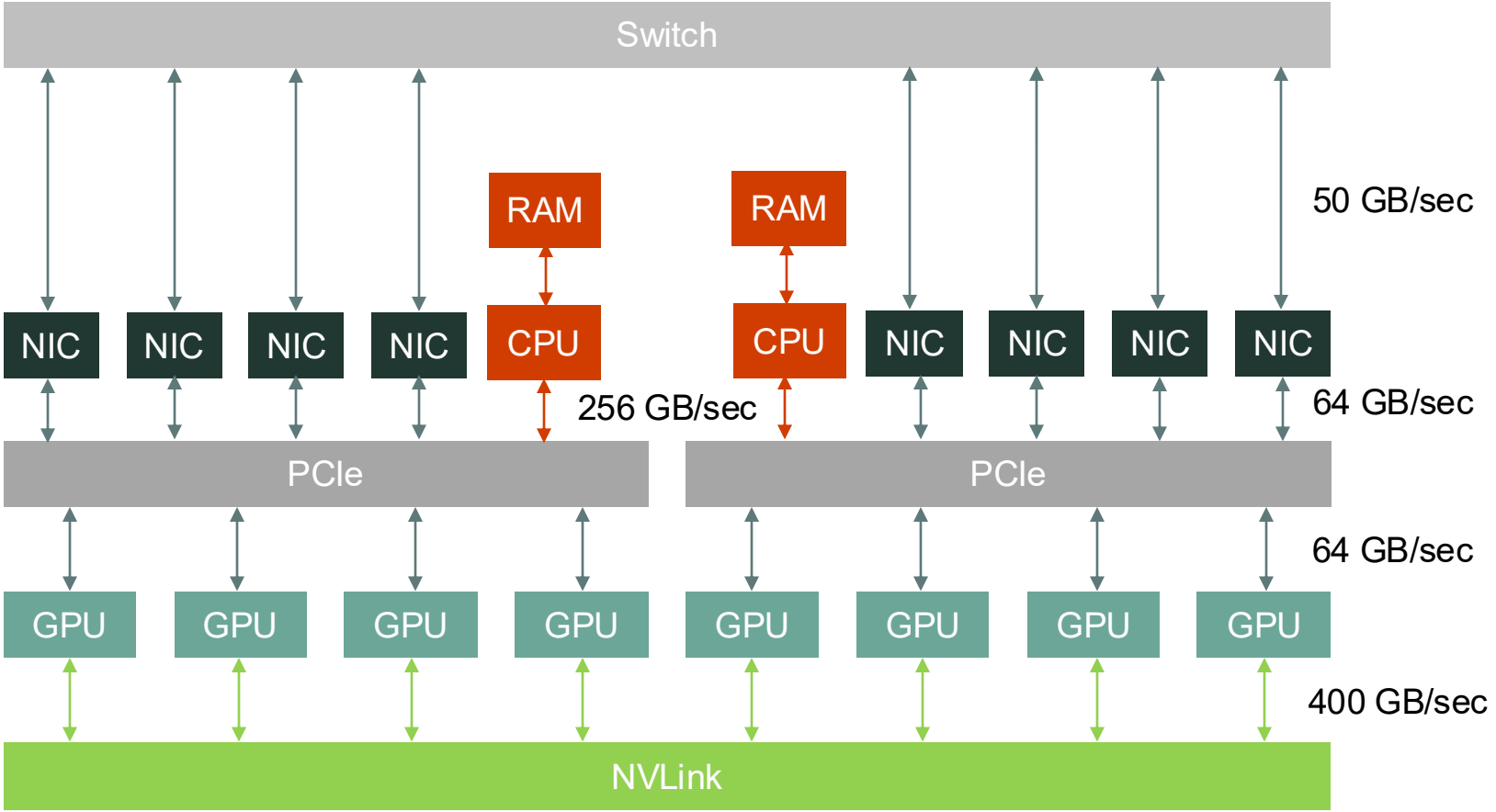
$$(H*(nh + kvh)*dh*2 + H*I*3)*2 / 400e9 = 4ms$$

- Время TP коммуникаций (NVLink)

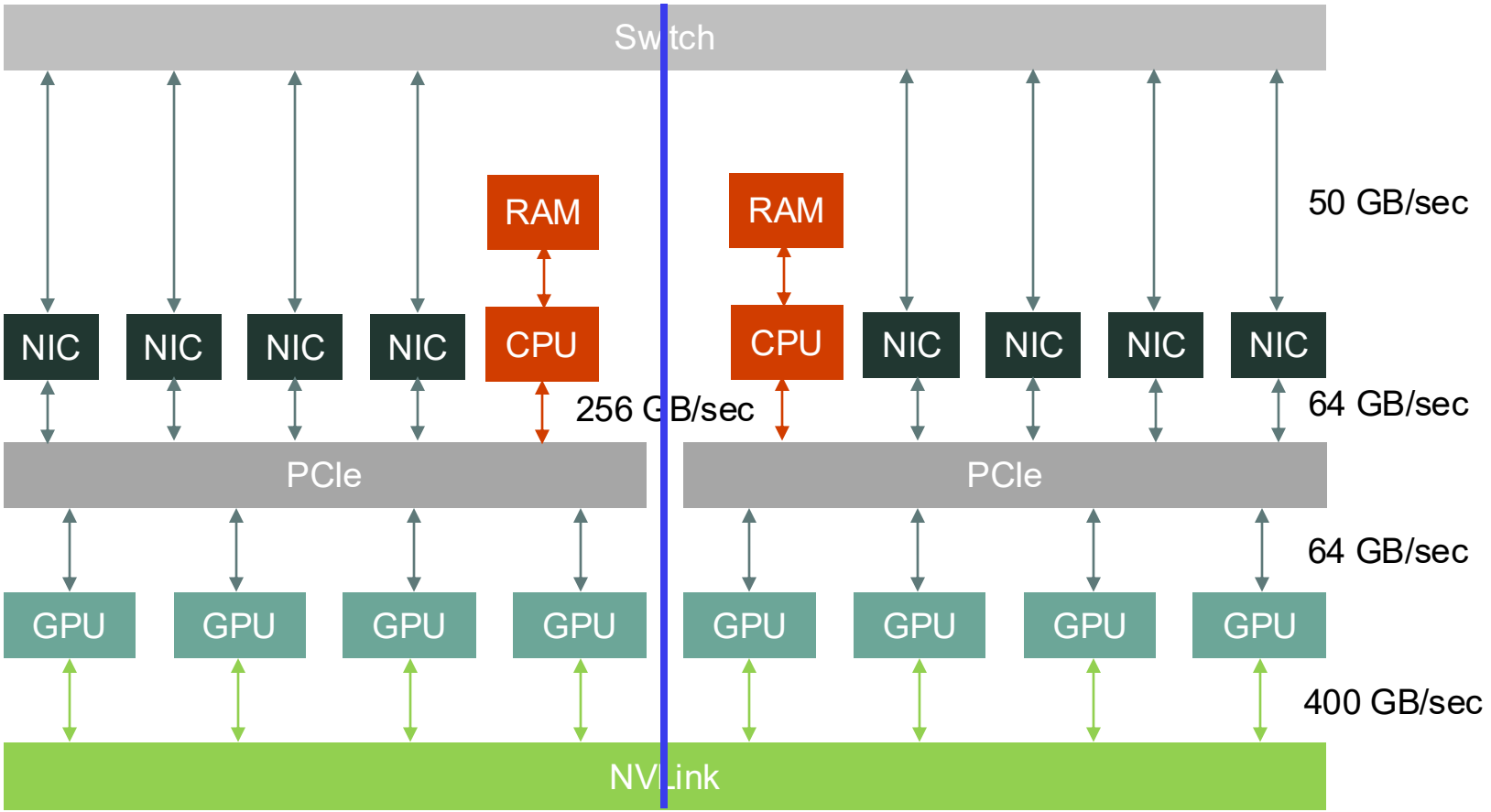
Почему слой модели не стал меньше в 2 раза?

$$S*H*TP*2*2 / 400e9 = 1.3ms$$

Карта Хоста (H100 SXM)



Карта Хоста (H100 SXM)



Время forward одного слоя 70B (TP=2)

- S = 4096 токенов, H = 8192, I = 28672, nh*dh = 8192, kvh = 4
- Суммарное время матричных умножений:

$$S*H*((nh + kvh)*dh*2 + I*3 + S*(2+1))*2 / 800e12 = 10ms$$

- Время memory bound операций (2 RMSNorm, 2 Add, 1 SwiGLU):

$$(S*H*2*(2*2 + 2*3) + S*I*2*3) / 2.4e12 = 0.55ms$$

- Время на AllGather FSDP (в одном слое 1560 MB)

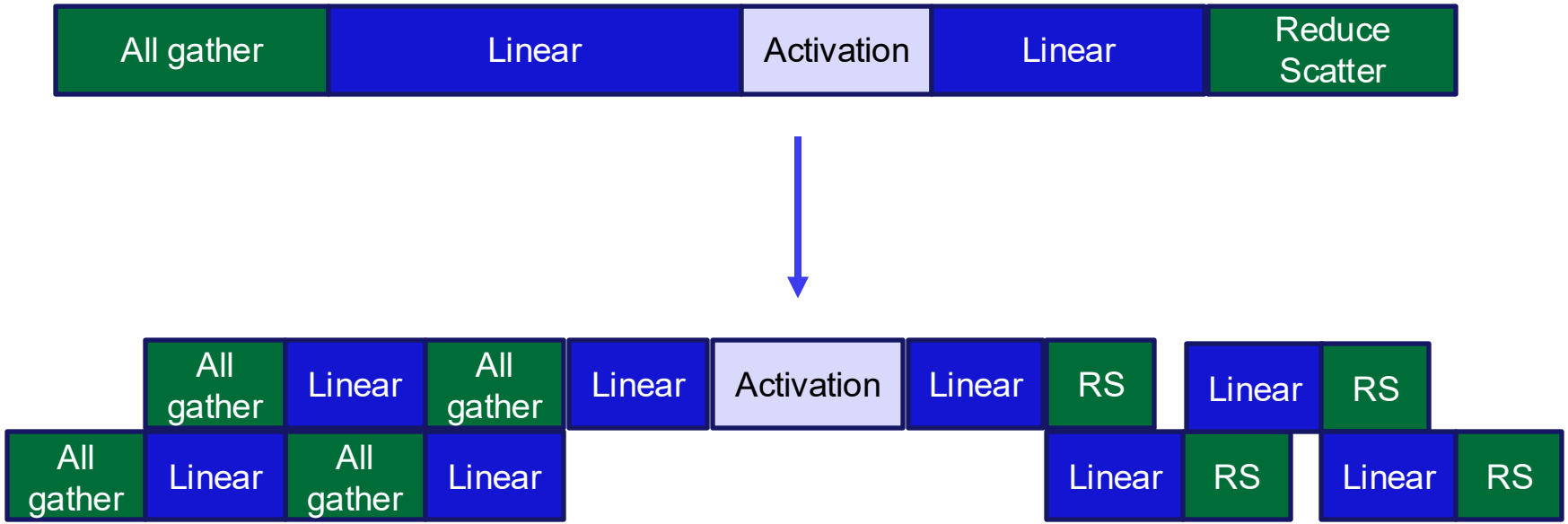
$$(H*(nh + kvh)*dh*2 + H*I*3)*2 / 400e9 = 4ms$$

- Время TP коммуникаций (NVLink)

$$S*H*TP*2*2 / 400e9 = 1.3ms$$

Эти коммуникации тормозят обучение?

Асинхронный TP



Асинхронный TP. Риски

- Матричные умножения могут оказаться memory bound – это приведет к замедлению

Выводы

- Симметричные коммуникации: FSDP и TP позволяют производить обучение LLM без потери скорости.
- Рекомпьют активаций – альтернатива доставке.
- Увеличение TP внутри хоста не приводит к уменьшению FSDP коммуникаций.
- Это основная причина, по которой подход FSDP+TP не является универсальным рецептом масштабирования обучения.
- Альтернативы: Context parallelism, Expert parallelism для MoE, Pipeline parallelism.

04

Offload / Upload

Что делать с одной GPU?

- Большие модели хочется обучать и на небольших конфигурациях.
- Как это сделать?

Что делать с одной GPU?

- Большие модели хочется обучать и на небольших конфигурациях.
- Как это сделать? Через Offload.

Как организовать Offload?

- Память, в которую вы хотите делать Offload стоит выделять с pin_memory=True.
- Выгрузку и загрузку можно делать в отдельном CUDA-stream.

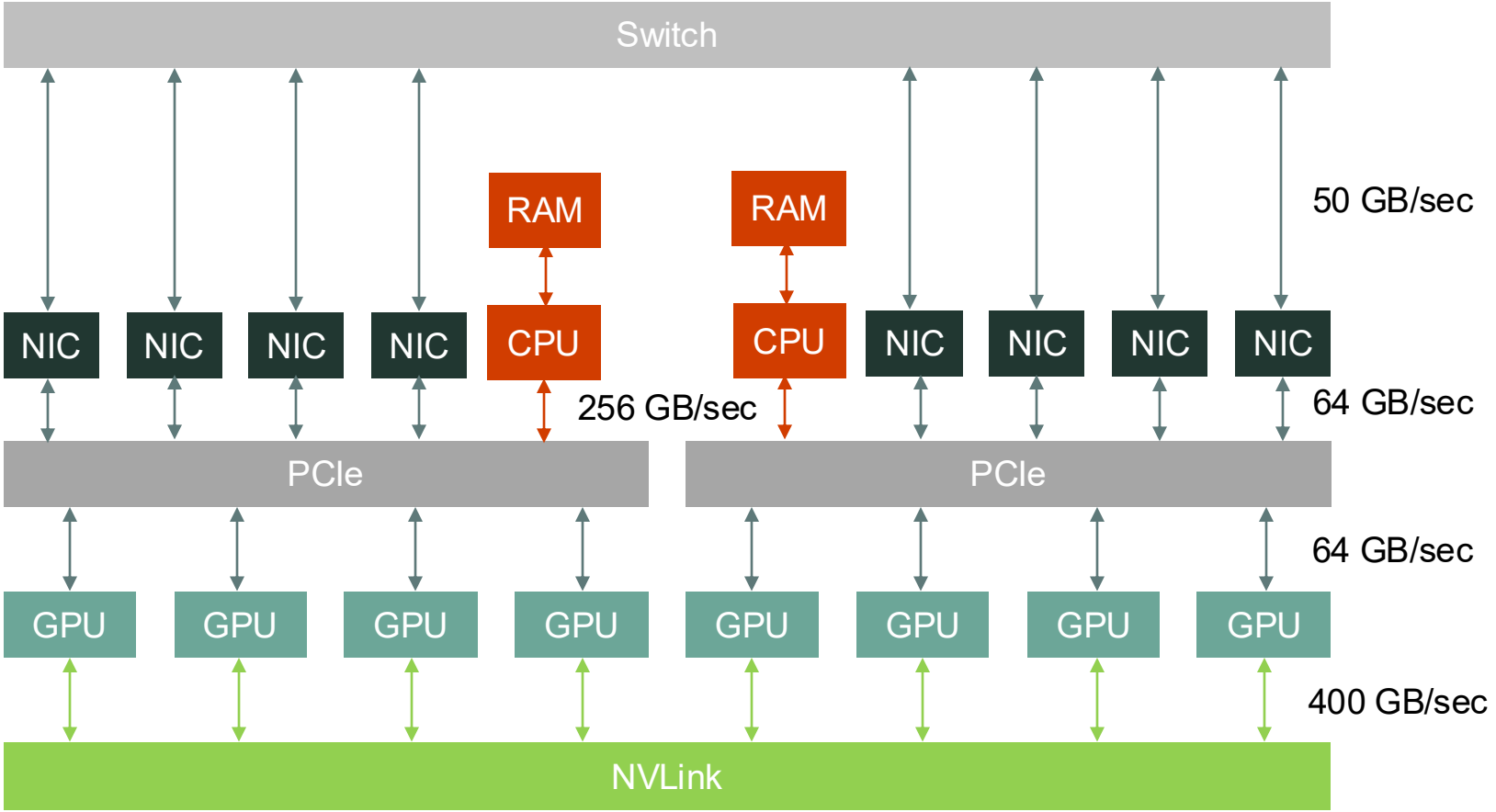
Как организовать Offload?

- Память, в которую вы хотите делать Offload стоит выделять с pin_memory=True.
- Выгрузку и загрузку можно делать в отдельном CUDA-stream.

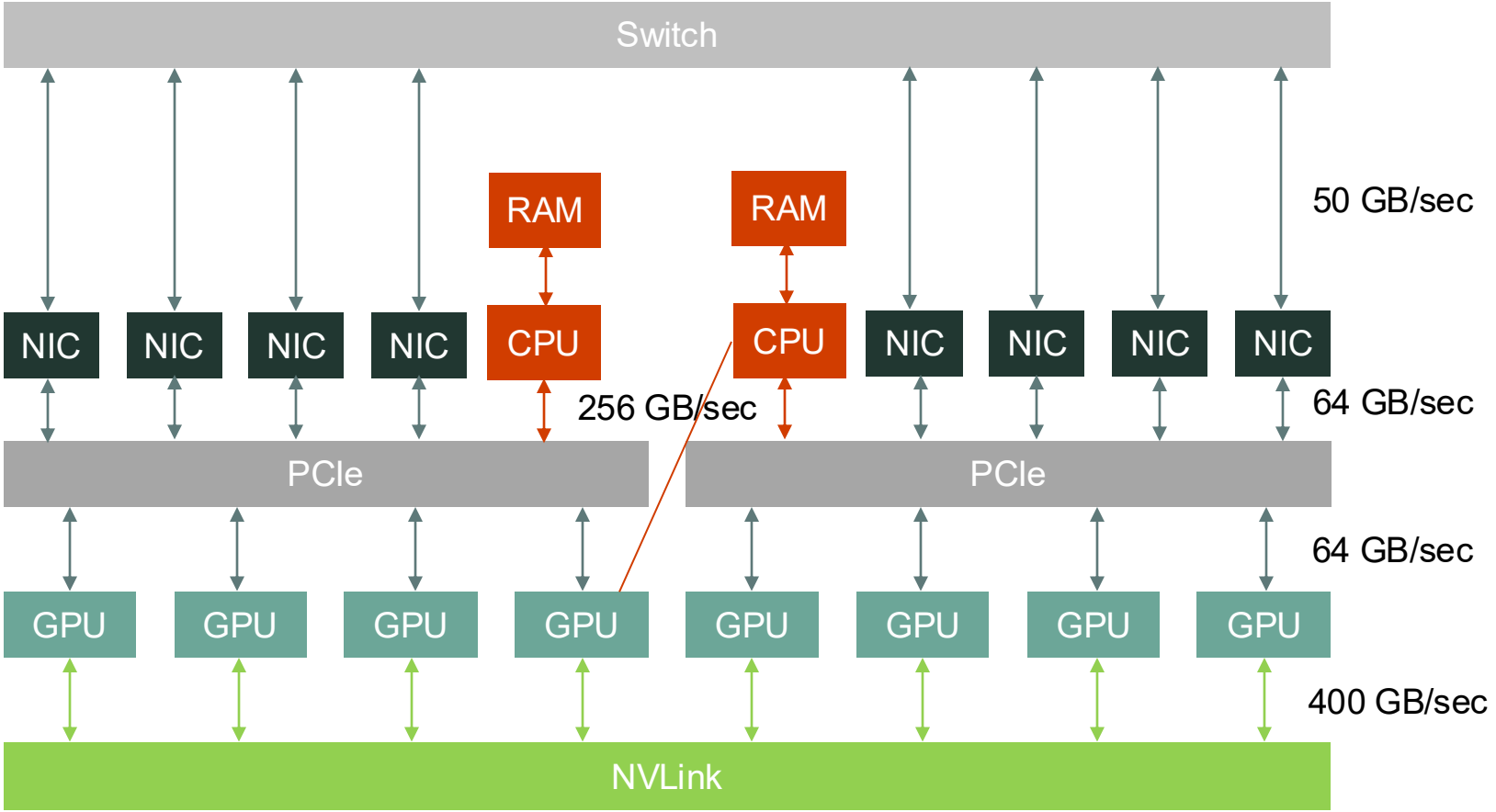
Но нужно быть осторожным:

- На кластере с несколькими NUMA-нодами надо явно указывать cpu_affinity при запуске, чтобы CPU и GPU были связаны быстрой шиной.

Карта Хоста (H100 SXM)



Карта Хоста (H100 SXM)



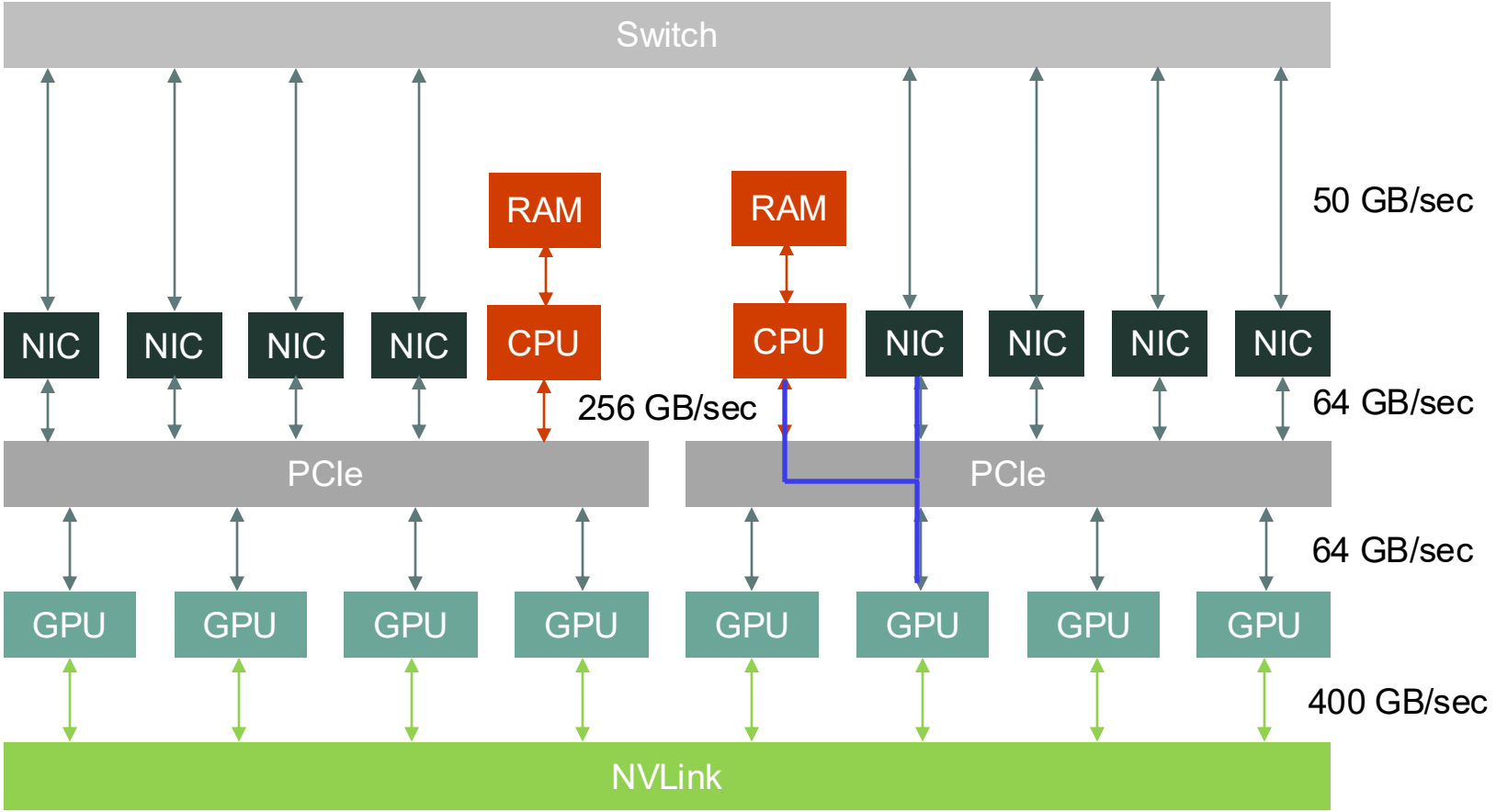
Как организовать Offload?

- Память, в которую вы хотите делать Offload стоит выделять с pin_memory=True.
- Выгрузку и загрузку можно делать в отдельном CUDA-stream.

Но нужно быть осторожным:

- На кластере с несколькими NUMA-нодами надо явно указывать cpu_affinity при запуске, чтобы CPU и GPU были связаны быстрой шиной.
- В пропускную способность шины можно упереться.

Карта Хоста (H100 SXM)



Пример асинхронного Offload

```
offload_stream = torch.cuda.Stream()
upload_stream = torch.cuda.Stream()

cuda_tensor = torch.randn([1024, 1024], device='cuda')
cpu_tensor = torch.empty([1024, 1024], pin_memory=True)

offload_stream.wait_stream(torch.cuda.default_stream())

with offload_stream:
    cpu_tensor.copy_(cuda_tensor)
    cpu_offload_event = torch.cuda.Event(enable_timing=True)
    cpu_offload_event.record()

upload_stream.wait_stream(torch.cuda.default_stream())

with upload_stream:
    cuda_tensor.copy_(cpu_tensor)
    cpu_upload_event = torch.cuda.Event(enable_timing=True)
    cpu_upload_event.record()
```

Как это работает?

- В рамках одного кернела в CUDA Stream достигается большая степень параллелизма, но кернелы выполняются строго последовательно.
- Некоторые кернелы используют разные ресурсы: compute/PCIe шину/NVLink. Мы хотим их ставить в параллель. Для этого можно использовать другие стримы.
- Для управления порядком используются event.

Как это работает?

- В рамках одного кернела в CUDA Stream достигается большая степень параллелизма, но кернелы выполняются строго последовательно.
- Некоторые кернелы используют разные ресурсы: compute/PCIe шину/NVLink. Мы хотим их ставить в параллель. Для этого можно использовать другие стримы.
- Для управления порядком используются event.

Как это работает?

- В рамках одного кернела в CUDA Stream достигается большая степень параллелизма, но кернелы выполняются строго последовательно.
- Некоторые кернелы используют разные ресурсы: compute/PCIe шину/NVLink. Мы хотим их ставить в параллель. Для этого можно использовать другие стримы.
- Для управления порядком используются event.

Пример асинхронного Offload

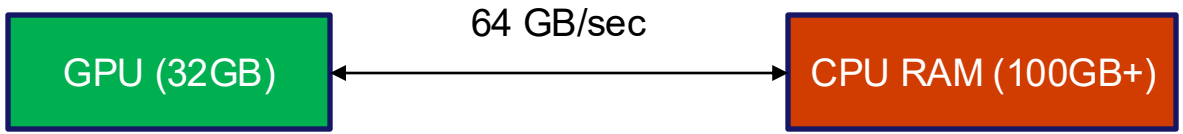
```
offload_stream = torch.cuda.Stream()
upload_stream = torch.cuda.Stream()

cuda_tensor = torch.randn([1024, 1024], device='cuda')
cpu_tensor = torch.empty([1024, 1024], pin_memory=True)

offload_stream.wait_stream(torch.cuda.default_stream())

with offload_stream:
    cpu_tensor.copy_(cuda_tensor)
    cpu_offload_event = torch.cuda.Event(enable_timing=True)
    cpu_offload_event.record()

upload_stream.wait_stream(torch.cuda.default_stream())

with upload_stream:
    cuda_tensor.copy_(cpu_tensor)
    cpu_upload_event = torch.cuda.Event(enable_timing=True)
    cpu_upload_event.record()
```

Пример асинхронного Offload

```
offload_stream = torch.cuda.Stream()
upload_stream = torch.cuda.Stream()

cuda_tensor = torch.randn([1024, 1024], device='cuda')
cpu_tensor = torch.empty([1024, 1024], pin_memory=True)

offload_stream.wait_stream(torch.cuda.default_stream())

with offload_stream:
    cpu_tensor.copy_(cuda_tensor)
    cpu_offload_event = torch.cuda.Event(enable_timing=True)
    cpu_offload_event.record()

upload_stream.wait_stream(torch.cuda.default_stream())

with upload_stream:
    cuda_tensor.copy_(cpu_tensor)
    cpu_upload_event = torch.cuda.Event(enable_timing=True)
    cpu_upload_event.record()
```

Математика

- Допустим, мы хотим дообучить Qwen 30B-A3B на RTX Nvidia 5090

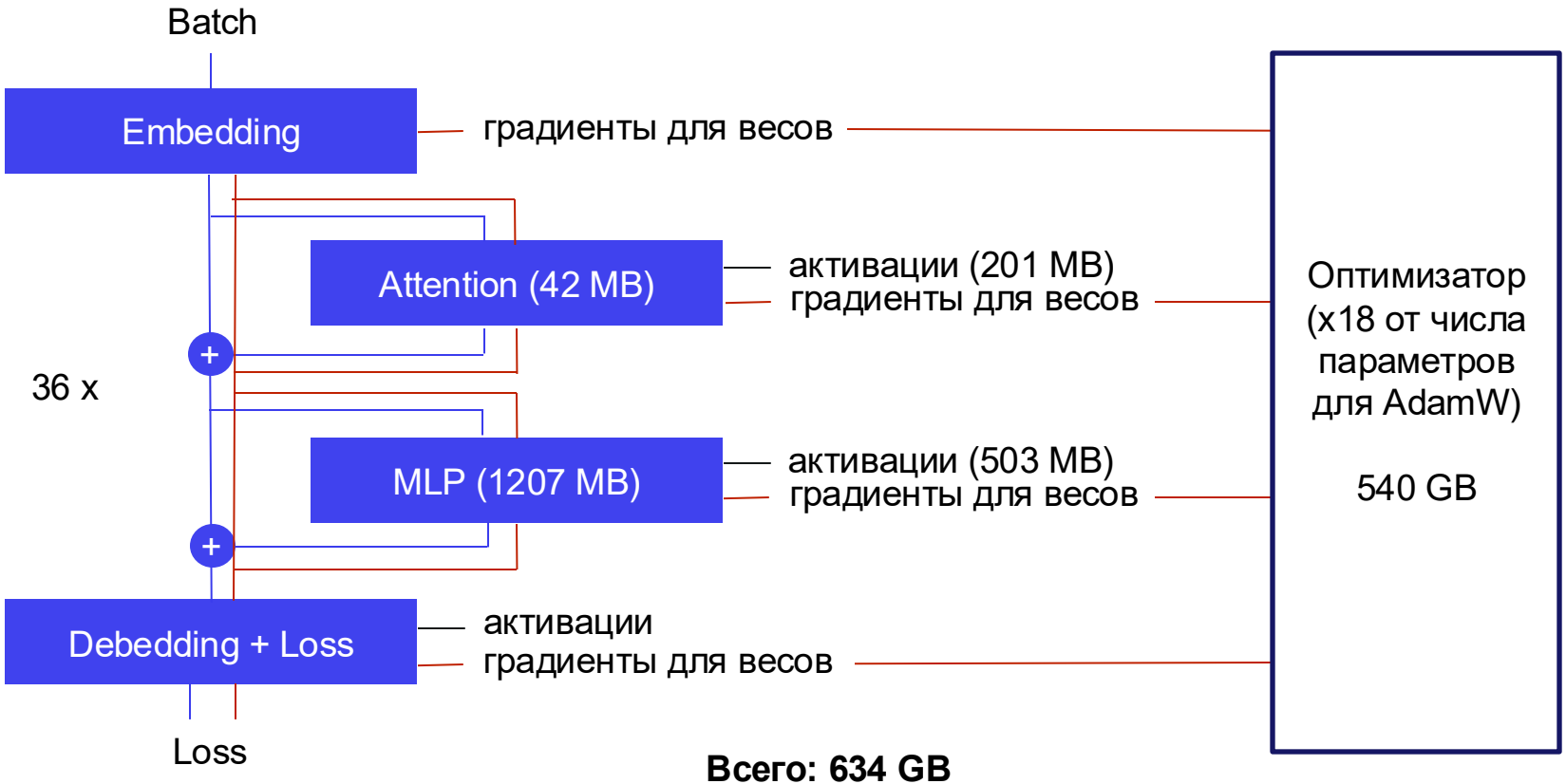


Математика

- Допустим, мы хотим дообучить Qwen 30B-A3B на RTX Nvidia 5090

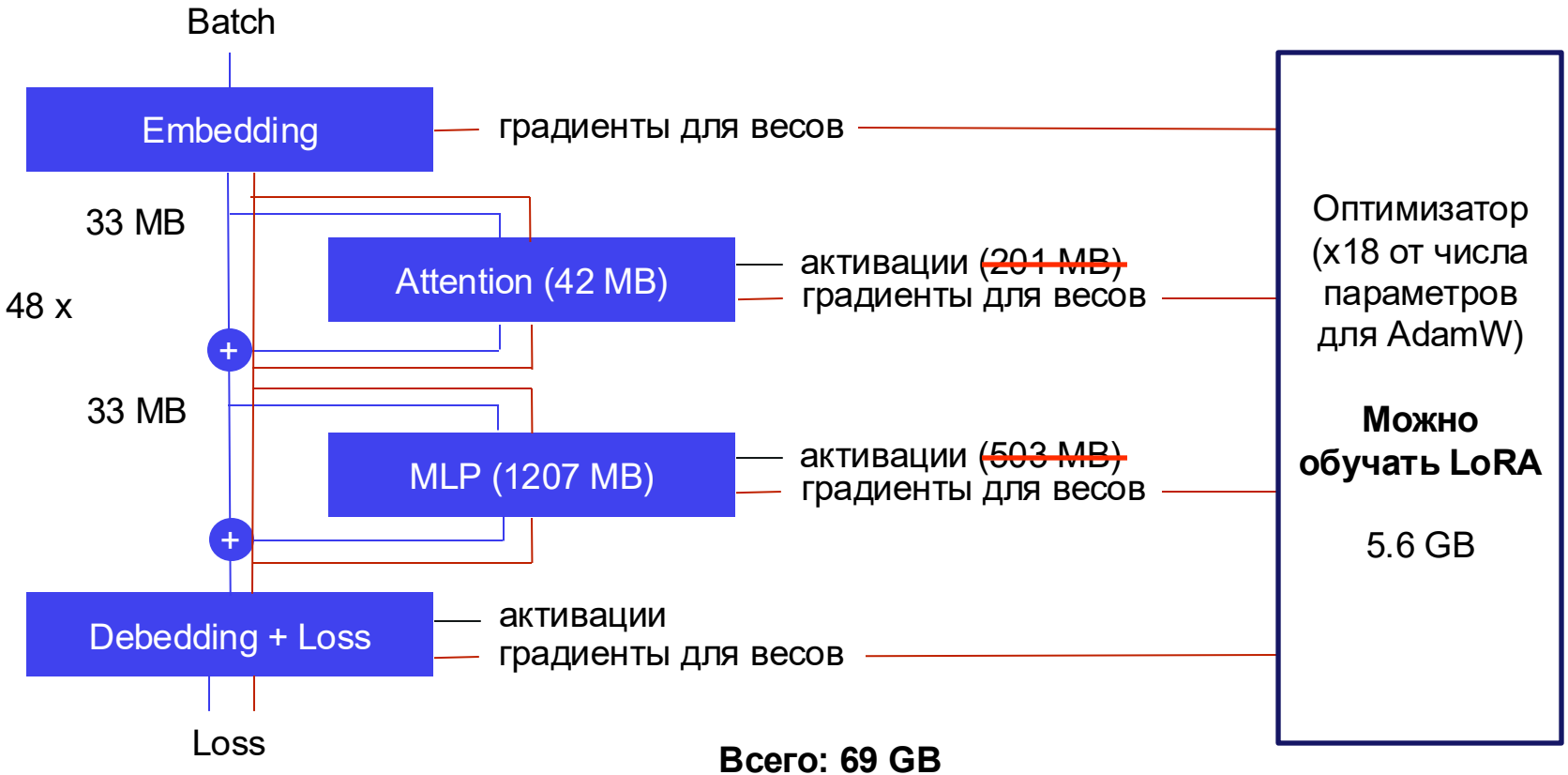
- Скорость: ~250 TFLOPS
- Скорость памяти: ~1.4 TB/sec
- Скорость доступа к CPU RAM: 64 GB/sec

Qwen 30B-A3B



Всего: 634 GB

Qwen 30B-A3B. LoRA и рекомпьют



Как организовать Offload?

- Через FSDP

```
class torch.distributed.fsdp.FullyShardedDataParallel(module, process_group=None,
sharding_strategy=None, cpu_offload=None, auto_wrap_policy=None,
backward_prefetch=BackwardPrefetch.BACKWARD_PRE, mixed_precision=None,
ignored_modules=None, param_init_fn=None, device_id=None, sync_module_states=False,
forward_prefetch=False, limit_all_gathers=True, use_orig_params=False,
ignored_states=None, device_mesh=None) #
```

Математика

- Допустим, мы хотим дообучить Qwen 30B-A3B на RTX Nvidia 5090

- Скорость: ~250 TFLOPS
- Скорость памяти: ~1.4 TB/sec
- Скорость доступа к CPU RAM: 64 GB/sec

- Компьют:

$$S*H*(top\_k*I*3*2 + (nh*hd*2 + kvh*hd*2)*2 + S* (2+1)) / 250e12 = 5.3 \text{ ms}$$

- Offload:

$$(H*(nh*hd*2 + kvh*hd*2)*2 + E*H*I*3*2) / 64e9 = 19.4 \text{ ms}$$

Математика

- Допустим, мы хотим дообучить Qwen 30B-A3B на RTX Nvidia 5090

- Скорость: ~250 TFLOPS
- Скорость памяти: ~1.4 TB/sec
- Скорость доступа к CPU RAM: 64 GB/sec

- Компьют:

$$S*H*(top\_k*I*3*2 + (nh*hd*2 + kvh*hd*2)*2 + S* (2+1)) / 250e12 = 5.3 \text{ ms}$$

- Offload:

$$(H*(nh*hd*2 + kvh*hd*2)*2 + E*H*I*3*2) / 64e9 = 19.4 \text{ ms}$$

– много, но это делает дообучение возможным

Что еще можно сделать?

- Сжать веса в fp8 или даже int4 и использовать QLoRA.
- Выбрать более дешевый оптимизатор и обучать больше весов.

05

Заключение

Заключение

- Эффективное обучение LLM связано в первую очередь с правильно выстроенной логистикой данных.
- Можно пересылать веса и их градиенты через FSDP, в том числе на CPU RAM.
- Можно пересылать активации и их градиенты через TP и другие параллелизмы.
- Можно делать их рекомпьют.

Школа анализа данных

Спасибо за внимание!

Хрущев Михаил
khr2@yandex-team.ru