Распределенное обучение нейросетей

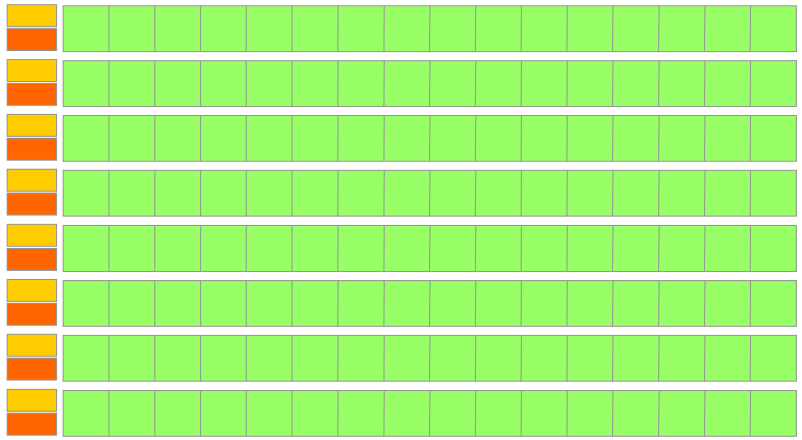
Барышников Антон

GPU vs CPU

- На GPU много ядер (~700-5000), на CPU мало (~4-8).
- Ядра CPU работают с произвольными сложными программами.
- GPU работает в модели SIMT, очень много небольших идентичных параллельных вычислений для больших объемов данных.
- CPU может очень быстро загрузить небольшой объем данных.
- GPU может загрузить гораздо больше данных за раз.
- GPU используют двухуровневую параллелизацию.

GPU vs CPU

CPU



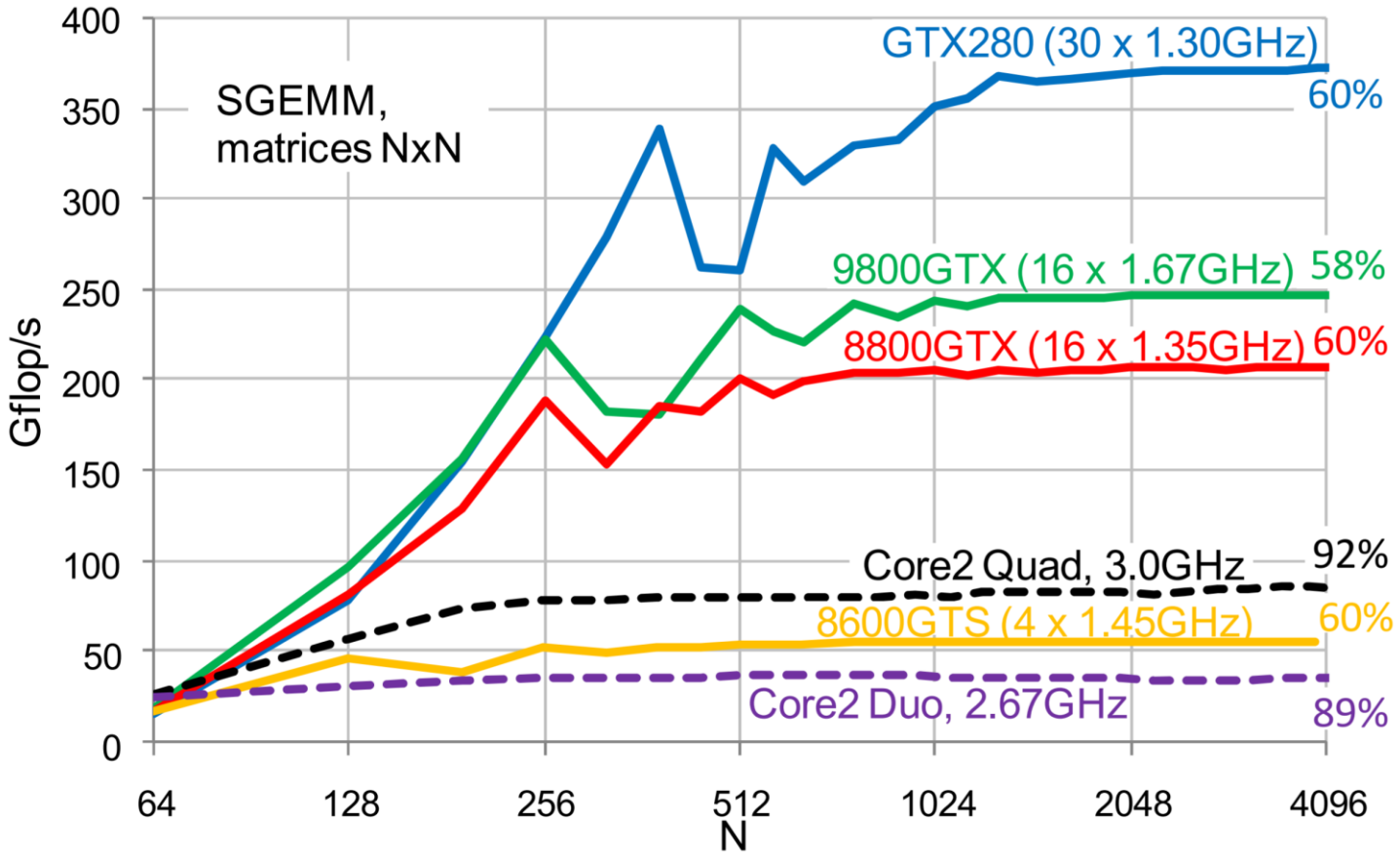
GPU

Умножение матриц на GPU

- R = AB.
- Single-pass, результат за один проход рендеринга.
- Один шейдер считает один элемент R.
- Multi-pass, несколько проходов рендеринга.
- Один шейдер считает часть одного элемента R.
- Слишком много разных вариаций.

[1] Understanding the Efficiency of GPU Algorithms for Matrix-Matrix Multiplication

Умножение матриц на GPU



[2] Benchmarking GPUs to Tune Dense Linear Algebra

Почему GPU так важны в DL?

- 2009 год, статья про использование GPU в DL от Стэнфорда.
- Умножение двух матриц 1000x1000 занимает 20ms.
- Вычисления занимают 0.5%, все остальное время — подгрузка данных.
- Решение — батчи.

[3] Large-scale Deep Unsupervised Learning using Graphics Processors

Почему GPU так важны в DL?

Package	Architecture	576x1024	1024x4096	2304x16000	4096x11008
Goto BLAS	Single CPU	563s	3638s	172803s	223741s
Goto BLAS	Dual-core CPU	497s	2987s	93586s	125381s
GPU		38.6s	184s	1376s	1726s
GPU Speedup		**12.9x**	**16.2x**	**68.0x**	**72.6x**

[3] Large-scale Deep Unsupervised Learning using Graphics Processors

Почему GPU так важны в DL?

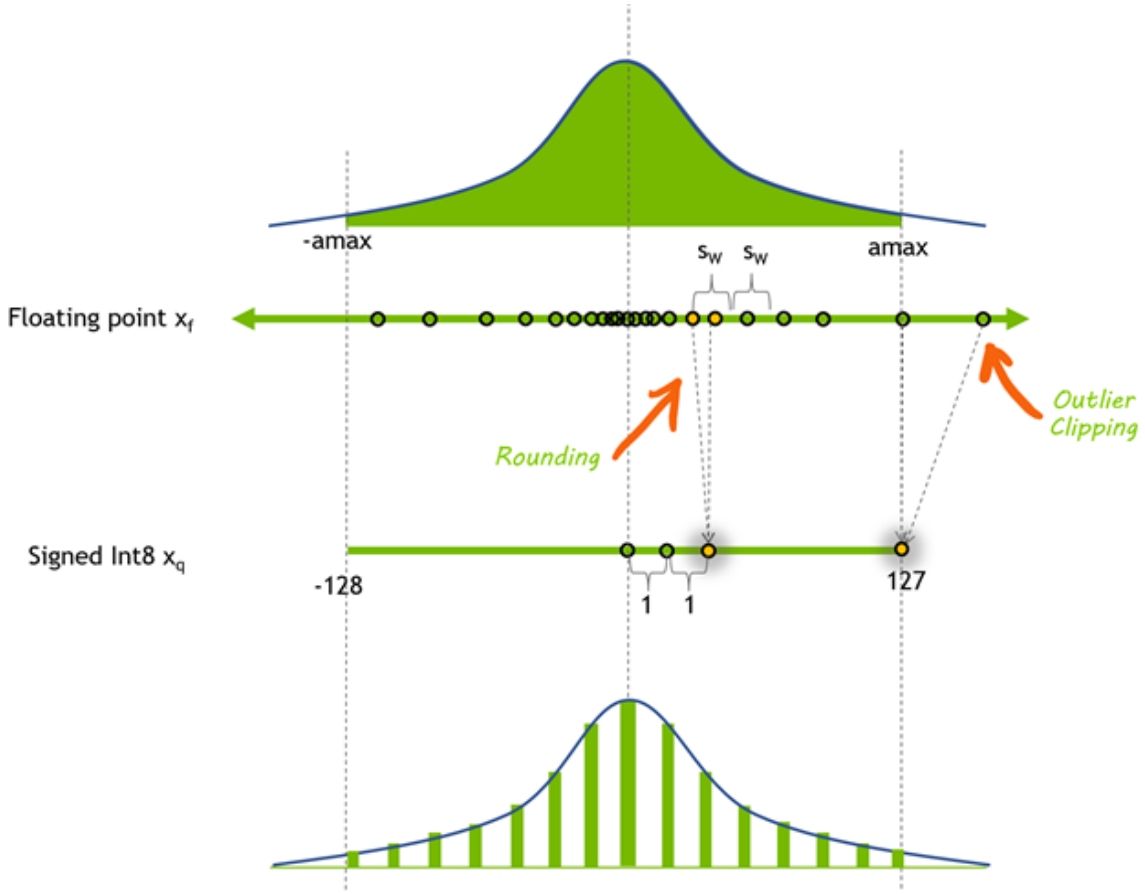
- The rest is history.
- Время для вопросов.

TPU v1

- Копроцессор, не может ничего делать сам.
- Сделаны для инференса.
- Более энерго-эффективны, быстрее работают.
- Работают только с int8, встроенная поддержка квантизации.
- Специальные блоки для перемножения матриц.
- Вшитые активации.

[4] An in-depth look at Google’s first Tensor Processing Unit (TPU)

Квантизация?

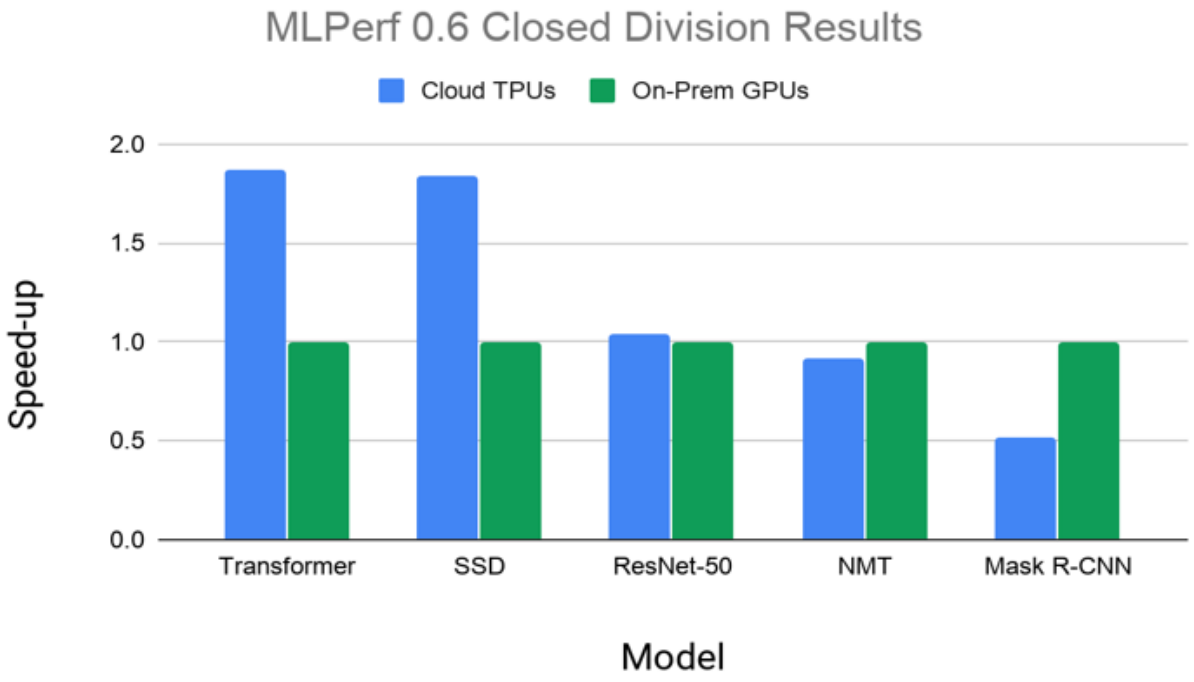


[5] Achieving FP32 Accuracy for INT8 Inference Using Quantization Aware Training with NVIDIA TensorRT

TPU v2

- Больше не копроцессор, можно обучать модели.
- Появилась поддержка float.
- Матричное умножение: операции в FP16, накопление в FP32.
- Еще больше вшитых операций: перестановки, транспонирования.

Результаты TPU



Google Cloud TPU v3 Pod speed-ups over the largest-scale on-premise NVIDIA DGX-2h clusters entered in the MLPerf 0.6 Closed Division. The Cloud TPU Pod submissions use 1024, 1024, 1024, 512, and 128 chips respectively; the NVIDIA DGX-2h clusters use 480, 240, 1536, 256, and 192 chips respectively. [1,2]

Время для вопросов

- А то сейчас будет посложнее.

CUDA

- Архитектура параллельных вычислений от NVIDIA.
- Есть SDK!

Пользовательские ядра

- CUDA-ядро (CUDA kernel) — небольшой код, который параллельно выполняется на GPU.
- Когда нужно?
- Пишем новую статью, нужен быстрый код.
- Хотим оптимизировать существующий слой.

Пишем новую статью, нужен быстрый код

- Пример от PyTorch — новый тип RNN с другой схемой ячейки.
- Нельзя использовать существующий код.
- Вариант 1 — написать модуль на питоне.
- Вариант 2 — на C++/CUDA.

Пишем новую статью, нужен быстрый код

Имплементация	Forward	Backward
Python	187.719 **us**	410.815 **us**
C++	149.802 **us**	393.458 **us**
C++/CUDA	129.431 **us**	304.641 **us**

Хотим оптимизировать существующий слой

- Пример — LightSeq2 от ByteDance.
- У формул градиентов есть несколько разных форм, которые по-разному хорошо параллелизуются (LayerNorm).
- Можно руками тьюнить сколько данных и как фетчится, какие кэши, и прочее.

[7] LightSeq2: Accelerated Training for Transformer-based Models on GPUs

Хотим оптимизировать существующий слой

$$\sigma(\mathrm{x}) = \sqrt{\mu(\mathrm{x}^2) - \mu(\mathrm{x})^2}.$$

This inspires us to rearrange the equation for backward calculation, which parallels synchronizations as well:

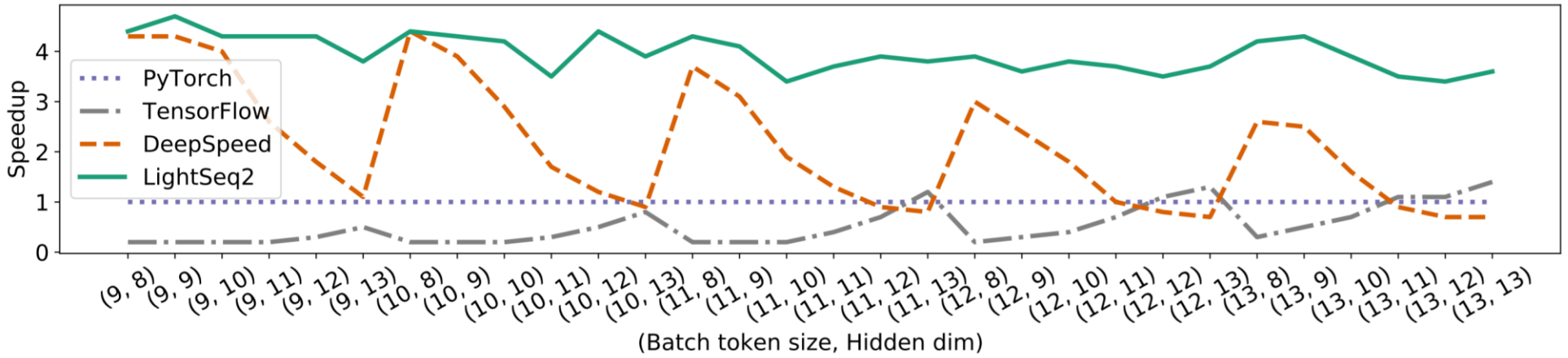
$$\nabla \mathrm{x}_i = \frac{\mathrm{w}_i \nabla \mathrm{y}_i}{\sigma(\mathrm{x})} + \alpha \cdot \sum_j \mathrm{w}_j \nabla \mathrm{y}_j + \beta \cdot \sum_j \mathrm{w}_j \nabla \mathrm{y}_j \mathrm{x}_j,$$

where α and β are coefficients that can be solved in parallel:

$$\alpha = \frac{[\mathrm{x}_i - \mu(\mathrm{x})]\mu(\mathrm{x}) - \sigma(\mathrm{x})}{m\sigma(\mathrm{x})^3} \qquad \beta = \frac{\mu(\mathrm{x}) - \mathrm{x}_i}{m\sigma(\mathrm{x})^3} \quad .$$

[7] LightSeq2: Accelerated Training for Transformer-based Models on GPUs

Хотим оптимизировать существующий слой



[7] LightSeq2: Accelerated Training for Transformer-based Models on GPUs

Layer Fusion

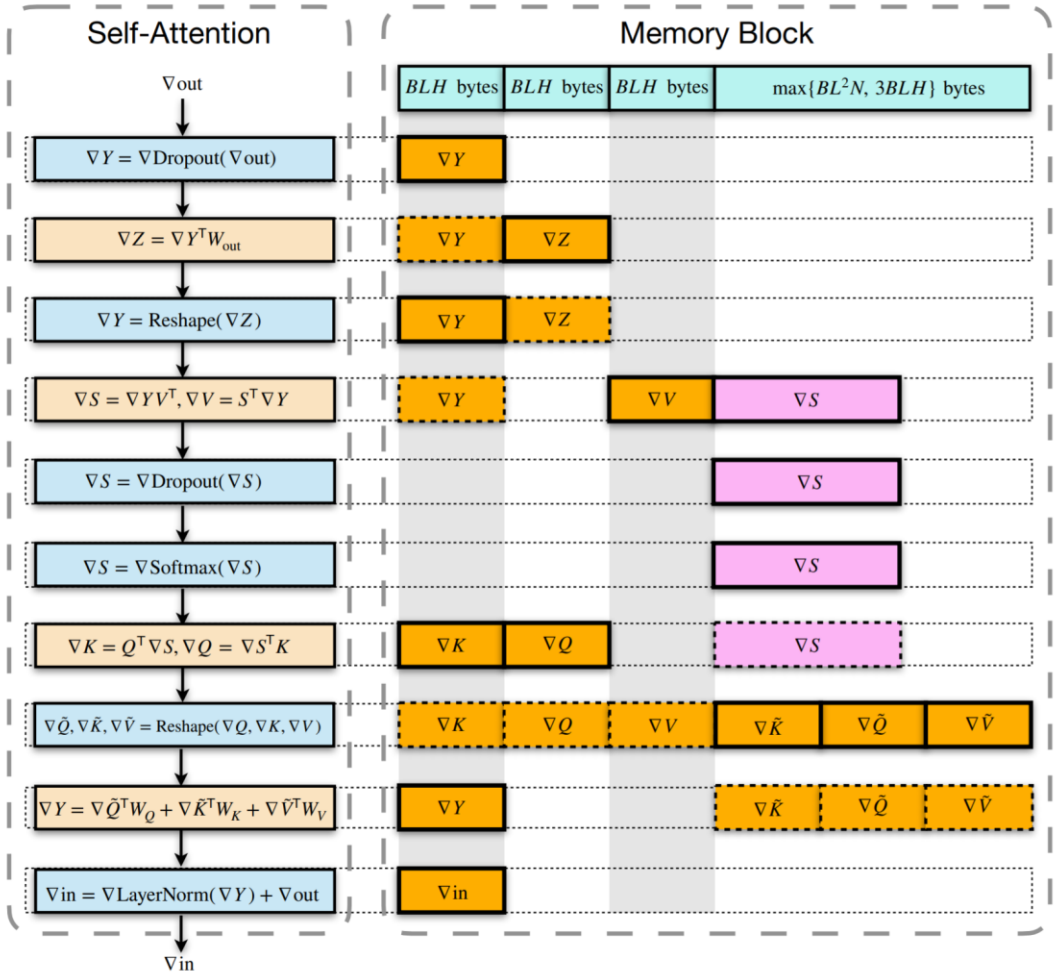
- Во время инференса можно сливать два соседних слоя, чтобы модель работала быстрее.
- Пример — Linear + BN, обе операции линейные.
- Во время обучения можно делать что-то похожее на gradient checkpoints.
- Объявляем новый слой ConvBn.
- Считаем промежуточную активацию отдельно во время backward.
- Итог — 1.56GB vs 2.68GB peak memory.

Memory Management

- Можно написать свое ядро для FP16 обучения и уменьшить число операций с памятью.
- Поделим память GPU на постоянную и временную.
- Постоянная будет отведена под параметры модели и их градиенты.
- Временная будет отведена под промежуточные вычисления.

[7] LightSeq2: Accelerated Training for Transformer-based Models on GPUs

Memory Management



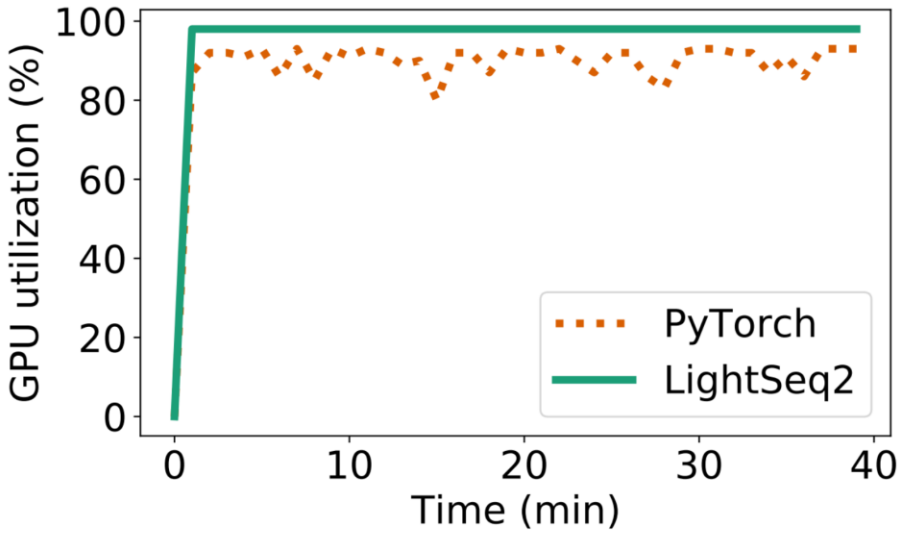
[7] LightSeq2: Accelerated Training for Transformer-based Models on GPUs

Итог

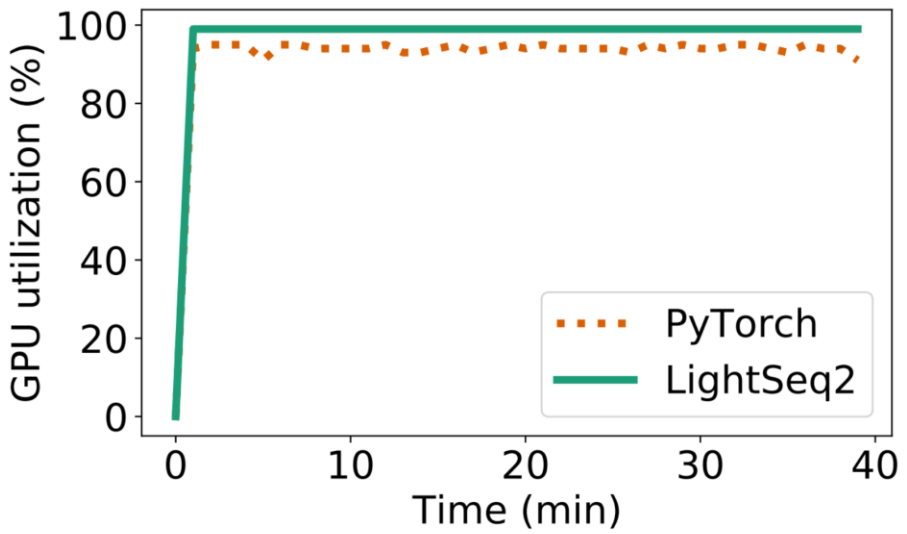
Models	nGPUs	Libraries	FP32	FP16
BERT-Base	1	PyTorch	124	224
		DeepSpeed	136	276
		LightSeq2	**148**	**380**
	8	PyTorch	928	1582
		DeepSpeed	1019	1804
		LightSeq2	**1097**	**2601**
BERT-Large	1	PyTorch	41	95
		DeepSpeed	44	114
		LightSeq2	**45**	**136**
	8	PyTorch	322	680
		DeepSpeed	344	838
		LightSeq2	**354**	**1074**

[7] LightSeq2: Accelerated Training for Transformer-based Models on GPUs

Итог



(a) Transformer-base.



(b) Transformer-big.

[7] LightSeq2: Accelerated Training for Transformer-based Models on GPUs

Спасибо!

Ссылки

- [1] Understanding the Efficiency of GPU Algorithms for Matrix-Matrix Multiplication
- [2] Benchmarking GPUs to Tune Dense Linear Algebra
- [3] Large-scale Deep Unsupervised Learning using Graphics Processors
- [4] An in-depth look at Google’s first Tensor Processing Unit (TPU)
- [5] Achieving FP32 Accuracy for INT8 Inference Using Quantization Aware Training with NVIDIA TensorRT
- [6] Custom C++ and CUDA extensions — PyTorch Tutorials
- [7] LightSeq2: Accelerated Training for Transformer-based Models on GPUs