

Nvidia CUDA

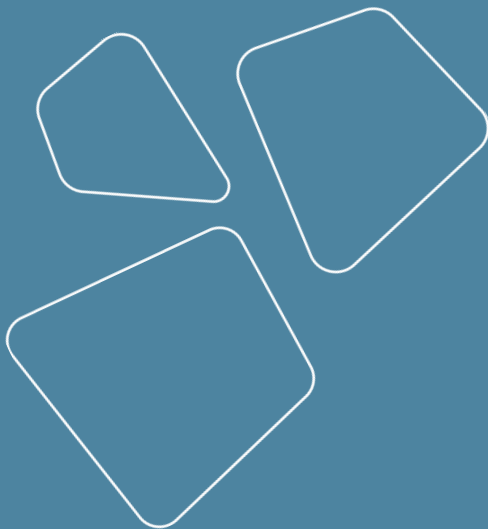
Astafurov Eugene

MIPT, MSU, fall 2023



Timeline

- Модель вычислений PTX Machine
- Nvidia GPU
- DL Performance Guide P.1



Часть 1: PTX Machine

Parallel Thread Machine

girafe
ai

01

PTX

- PTX - низкоуровневая виртуальная машина параллельного выполнения потоков
- PTX и ISA (Instruction Set Architecture) представляет собой GPU как устройство параллельных вычислений с данными

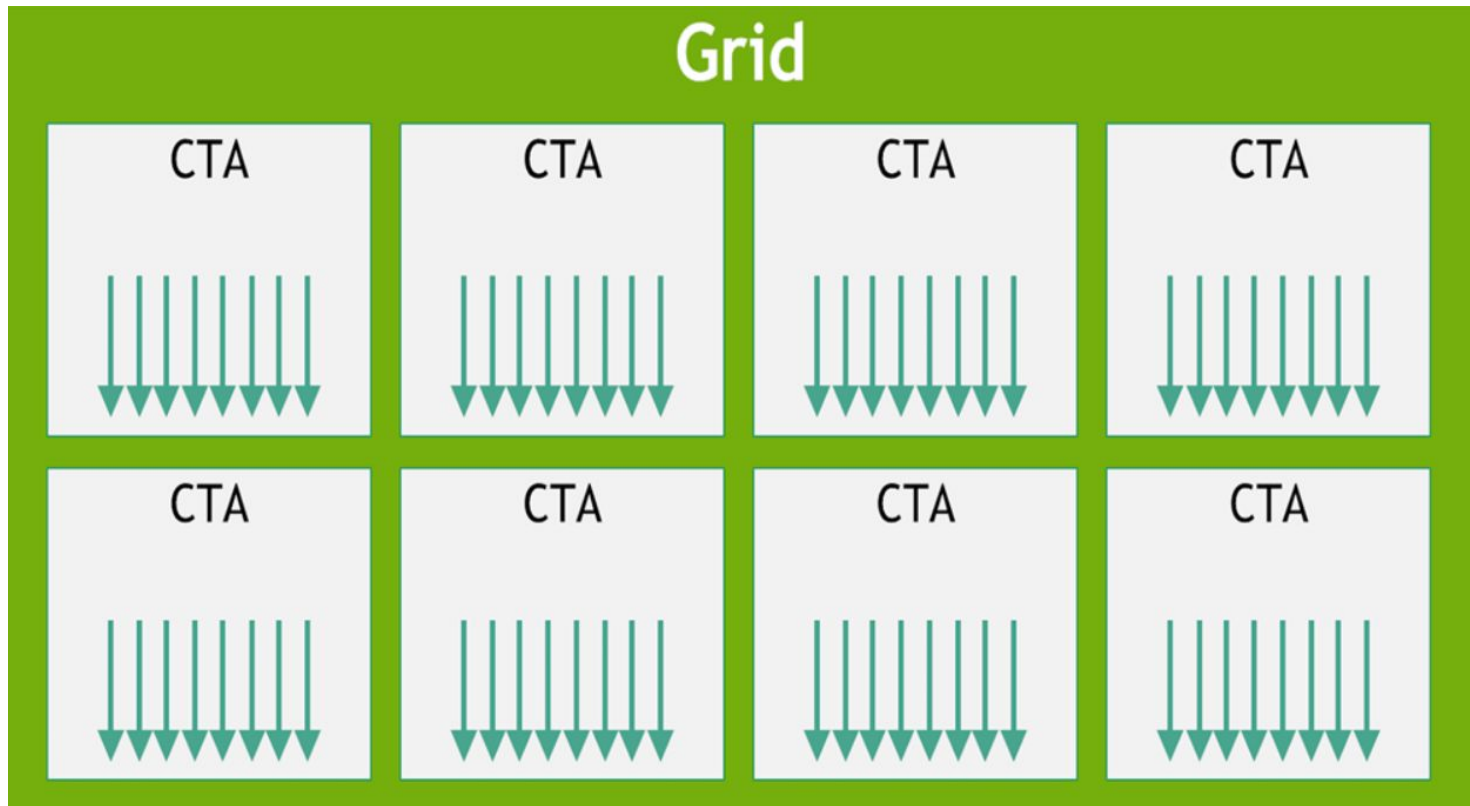
Масштабируемое вычисление

- параллельные вычисления с данными
- высокая арифметическая интенсивность
- соотношение арифметических операций к операциям с памятью

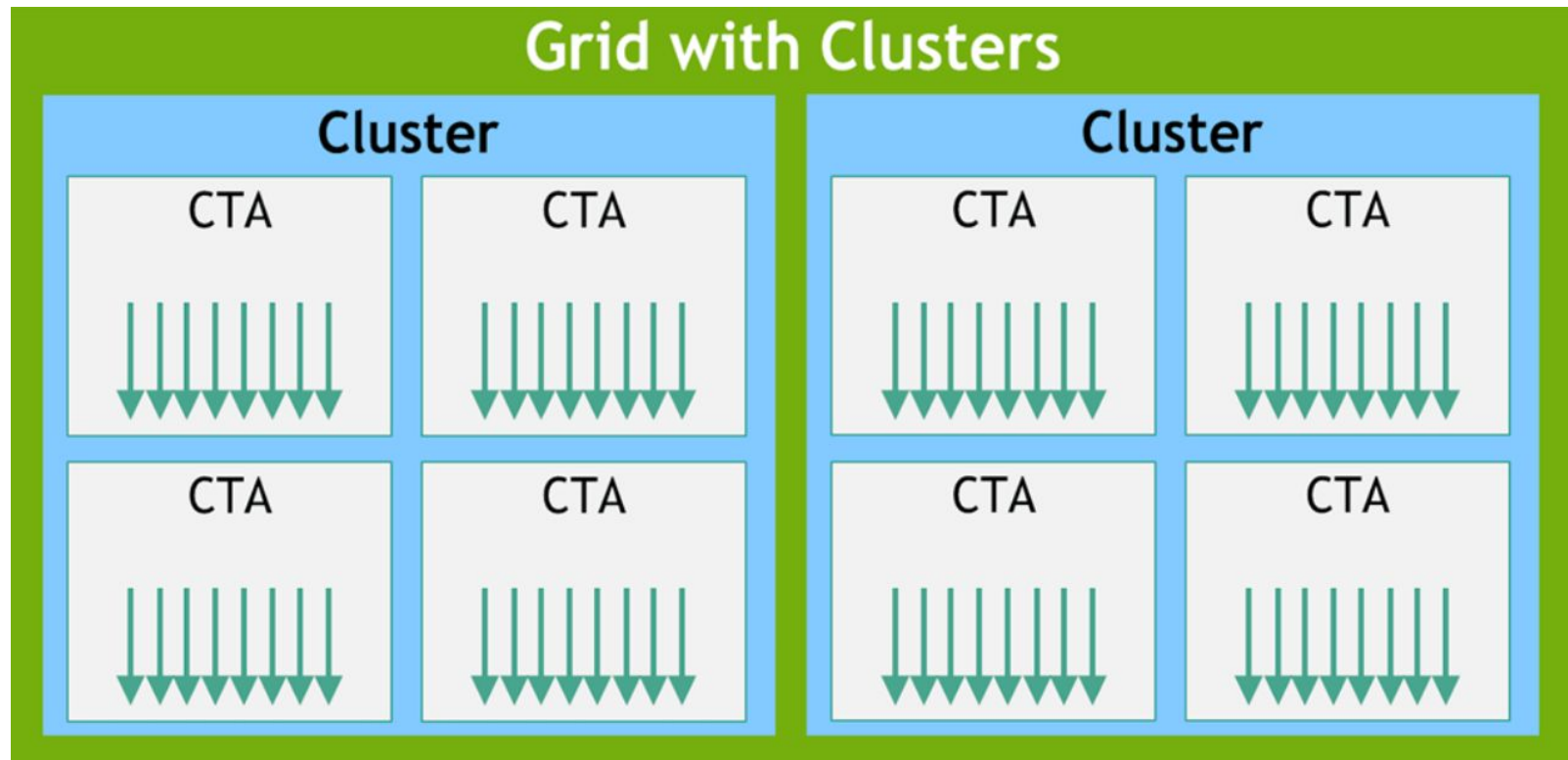
Со-процессор

1. Компиляция функции ядра в набор инструкций PTX
2. Перевод в целевой набор инструкций GPU

Иерархия потоков



Иерархия потоков

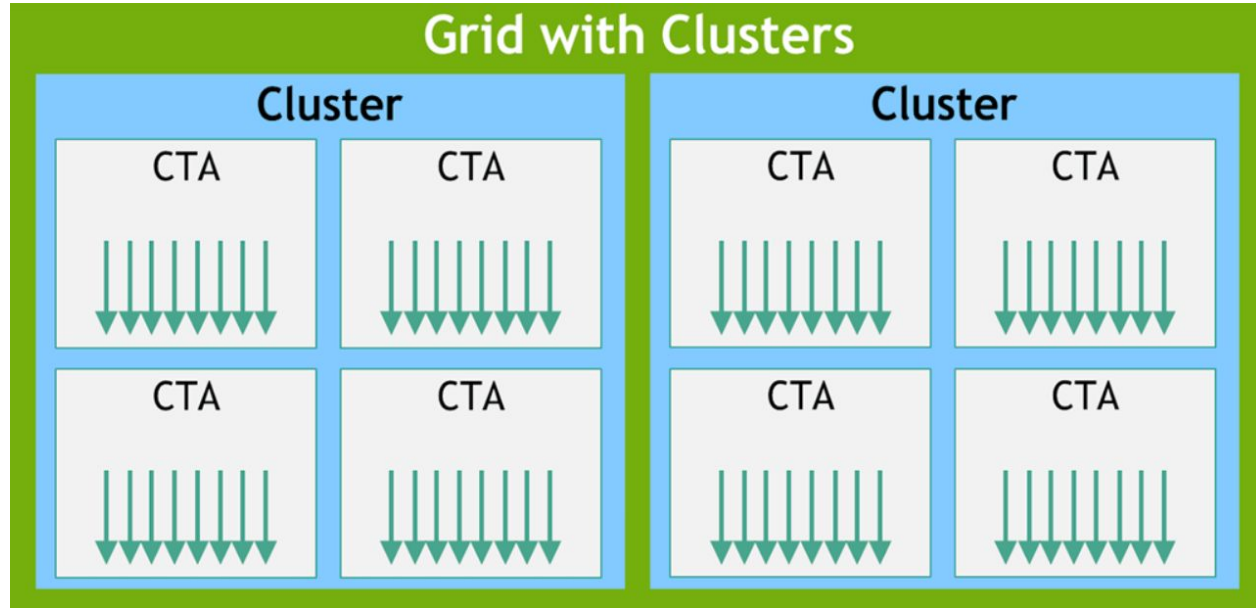


Согласованные массивы потоков

- “Ядро” - специальная функция, написанная для выполнения на GPU
- “CTA - Cooperative Thread Array” - массив поток, выполняющих ядро одновременно или параллельно
- “SIMT” - Single instruction Multiple Threads
- “Варп” - максимальное подмножество потоков из CTA, выполняющие одни и те же инструкции одновременно (выполняющиеся в режиме SIMT)

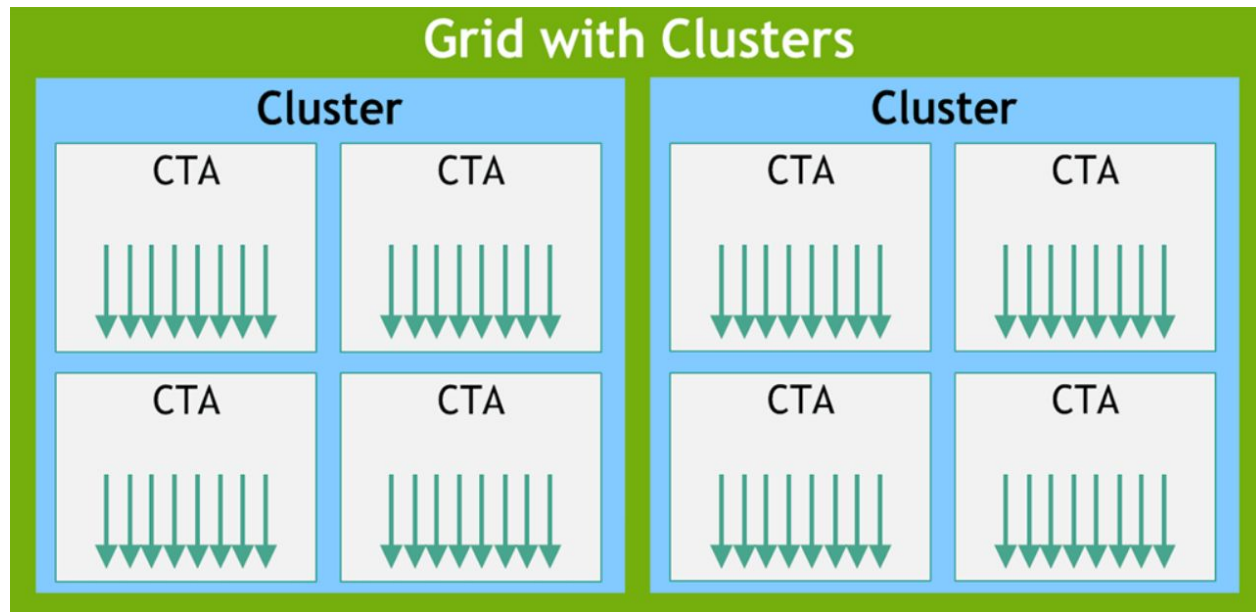
Кластер согласованных потоков

- “Кластер” - это группа СТА, которые выполняются одновременно или параллельно и могут синхронизироваться и взаимодействовать друг с другом через Shared Memory

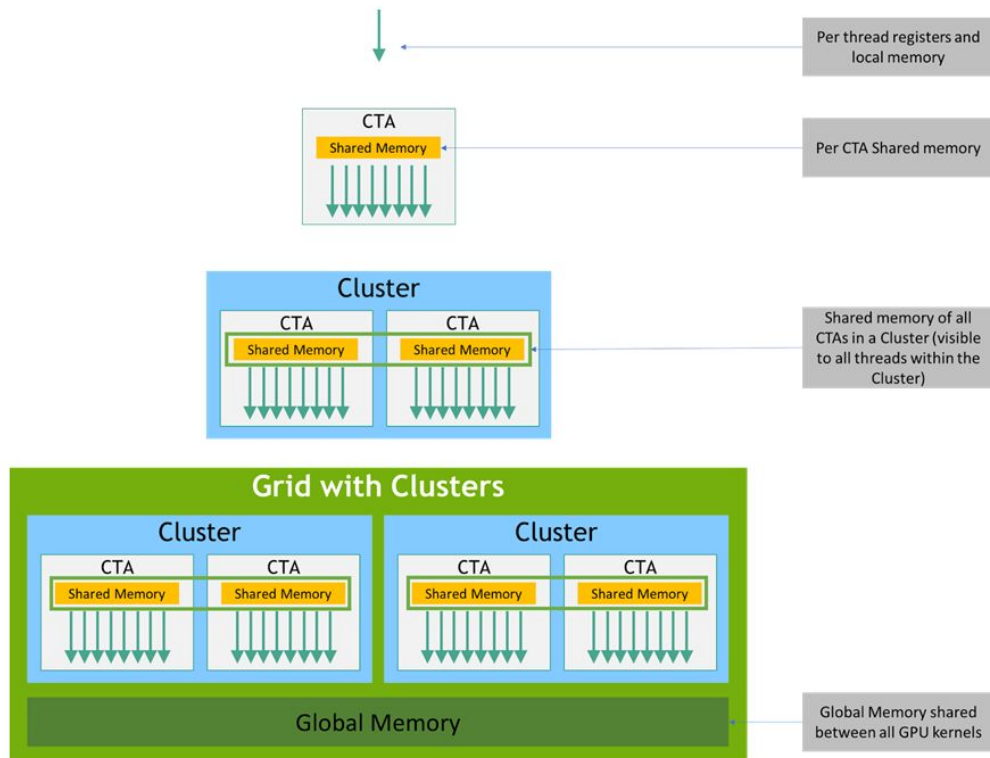


Сетка кластеров

- Кластеры СТА, выполняющие одно и то же ядро, могут быть сгруппированы в сетку кластеров так, что общее количество потоков становится очень велико



Иерархия памяти



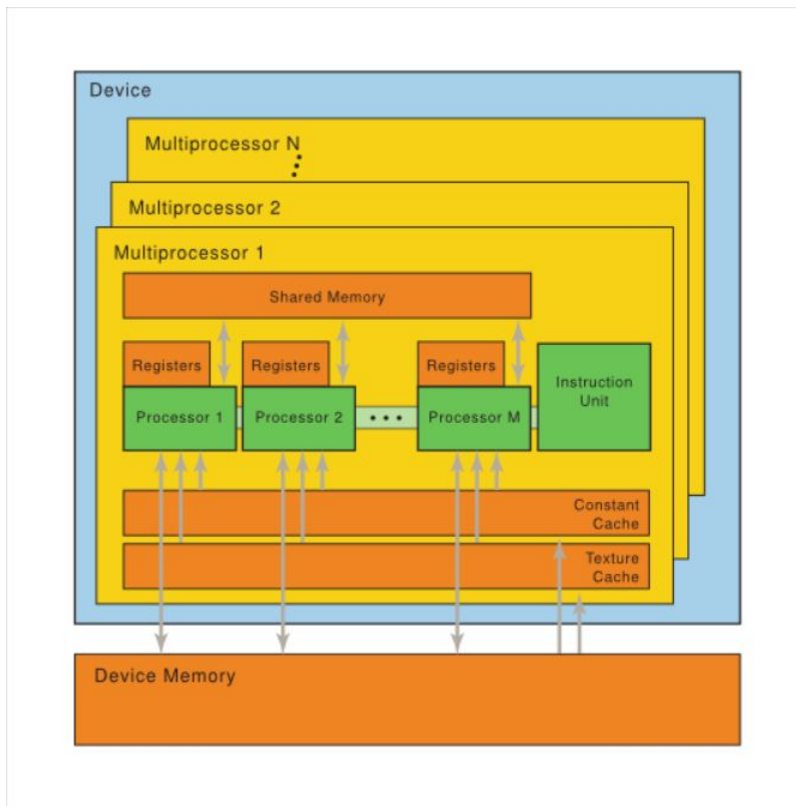
Набор мультипроцессоров SIMT

- Масштабируемая архитектура
- Блоки сетки распределяются по мультипроцессорам для параллельного выполнения
- *“Конвейерное выполнение”* - новые блоки запускаются на освободившихся ресурсах
- Мультипроцессор содержит ядра Scalar Processor (CUDA ядра) и встроенную shared memory
- Нулевой оверхед шедулера
- Барьерная синхронизация в SP
- Мелкозернистый параллелизм

Модель выполнения SIMT

- На каждом этапе выбирается варп
- Выдается следующая инструкция активным потокам варпа

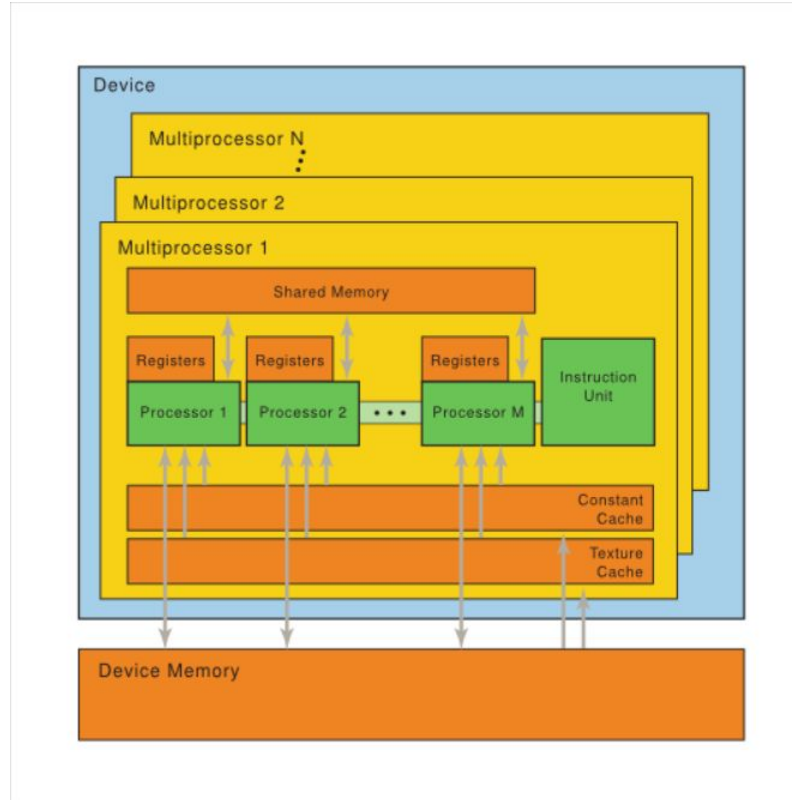
Различия между SIMT и SIMD



Независимое планирование потоков

- До Volta варпы использовали общий программный счетчик для всех потоков в варпе
- Начиная с Volta независимое планирование потоков независимо от варпа

On-Chip Shared memory



ОСНОВНЫЕ ТИПЫ

Basic Type	Fundamental Type Specifiers
Signed integer	<code>.s8</code> , <code>.s16</code> , <code>.s32</code> , <code>.s64</code>
Unsigned integer	<code>.u8</code> , <code>.u16</code> , <code>.u32</code> , <code>.u64</code>
Floating-point	<code>.f16</code> , <code>.f16x2</code> , <code>.f32</code> , <code>.f64</code>
Bits (untyped)	<code>.b8</code> , <code>.b16</code> , <code>.b32</code> , <code>.b64</code> , <code>.b128</code>
Predicate	<code>.pred</code>

ОСНОВНЫЕ ТИПЫ

- **Формат данных bf16:**

Этот формат данных представляет собой 16-битный формат с плавающей точкой с 8 битами для экспоненты и 7 битами для мантиссы.

- **Формат данных e4m3:**

Этот формат данных представляет собой 8-битный формат с плавающей точкой с 4 битами для экспоненты и 3 битами для мантиссы. Кодирование e4m3 не поддерживает бесконечность

- **Формат данных e5m2:**

Этот формат данных представляет собой 8-битный формат с плавающей точкой с 5 битами для экспоненты и 2 битами для мантиссы.

- **Формат данных tf32:**

Этот формат данных представляет собой специальный 32-битный формат с плавающей точкой, поддерживаемый инструкциями умножения и аккумуляции матриц, с тем же диапазоном, что и .f32, и уменьшенной точностью (≥ 10 бит).

Упакованные типы данных

С плавающей точкой:

- Тип **.f16x2**, содержащий два значения с плавающей точкой .f16.
- Тип **.bf16x2**, содержащий два значения альтернативного формата с плавающей точкой .bf16.
- Тип **.e4m3x2**, содержащий два значения альтернативного формата с плавающей точкой .e4m3.
- Тип **.e5m2x2**, содержащий два значения альтернативного формата с плавающей точкой .e5m2.
- Тип .f16x2 поддерживается как основной тип. Но типы .bf16x2, .e4m3x2 и .e5m2x2 не могут использоваться как фундаментальные типы - они поддерживаются как типы для определенных инструкций. Например если мы кладем в переменную регистра тип .bf16x2, то она должна быть объявлена с типом .b32. Переменная регистра, содержащая данные .e4m3x2 или .e5m2x2, должна быть объявлена с типом .b16.

Целочисленные:

PTX поддерживает два варианта типов упакованных целочисленных данных: .u16x2 и .s16x2. Упакованный тип данных состоит из двух значений .u16 или .s16. Переменная регистра, содержащая данные .u16x2 или .s16x2, должна быть объявлена с типом .b32. Типы упакованных целочисленных данных не могут использоваться как фундаментальные типы. Они поддерживаются только для определенных инструкций.

Тензоры

Свойства:

- Размерность
- Размеры в каждом измерении
- Типы отдельных эоментов
- Tenor stride

Инструкции RTX для тензоров так же включают:

- Копирование данных
- Уменьшение данных

Тезоры

Размерность, размер и формат тензора:

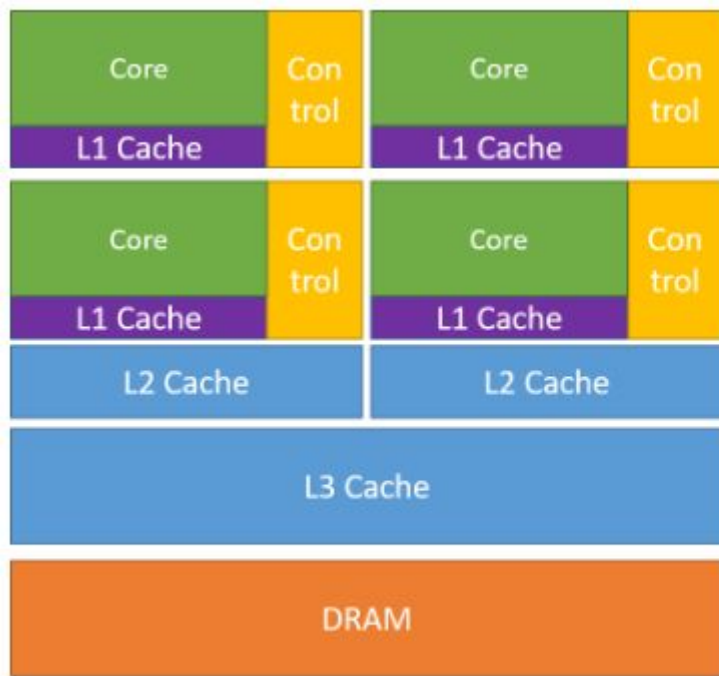
- Битовый тип: *.b32, .b64*
- Целое число: *.u8, .u16, .u32, .s32, .u64, .s64*
- Числа с плавающей точкой и альтернативные: *.f16, .bf16, .tf32, .f32, .f64*

Часть 2: Nvidia GPU

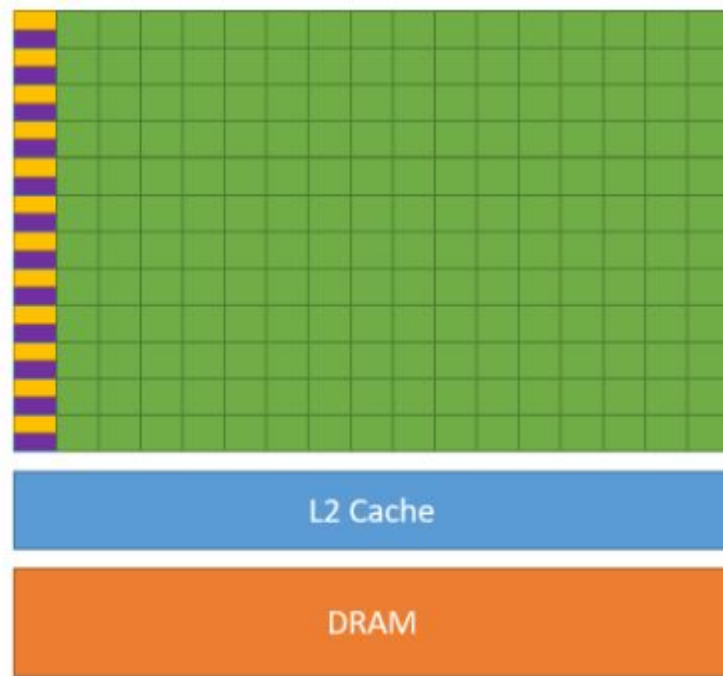
girafe
ai

02

Преимущества использования GPU



CPU



GPU

CUDA


GPU Computing Applications

Libraries and Middleware

cuDNN TensorRT	cuFFT cuBLAS cuRAND cuSPARSE	CULA MAGMA	Thrust NPP	VSIPL SVM OpenCurrent	PhysX OptiX iRay	MATLAB Mathematica
-------------------	---------------------------------------	---------------	---------------	-----------------------------	------------------------	-----------------------





Programming Languages

C	C++	Fortran	Java Python Wrappers	DirectCompute	Directives (e.g. OpenACC)
---	-----	---------	----------------------------	---------------	------------------------------

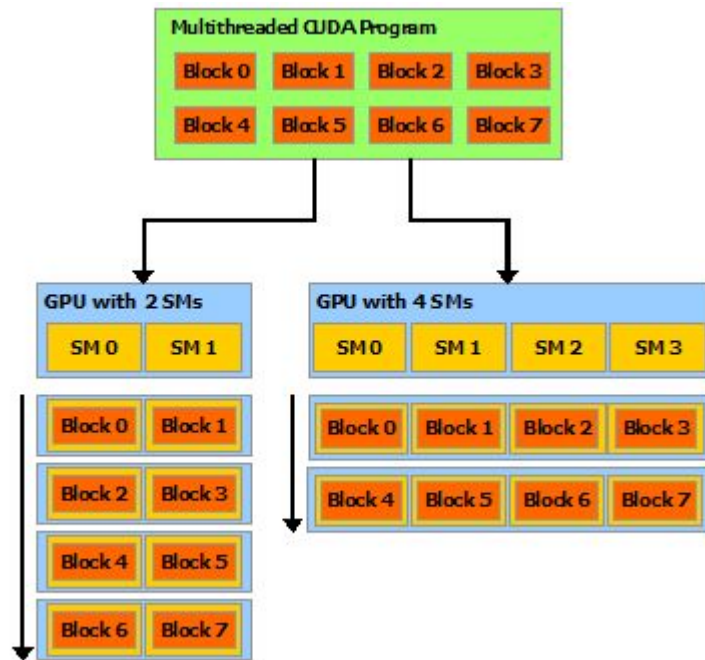


CUDA-Enabled NVIDIA GPUs

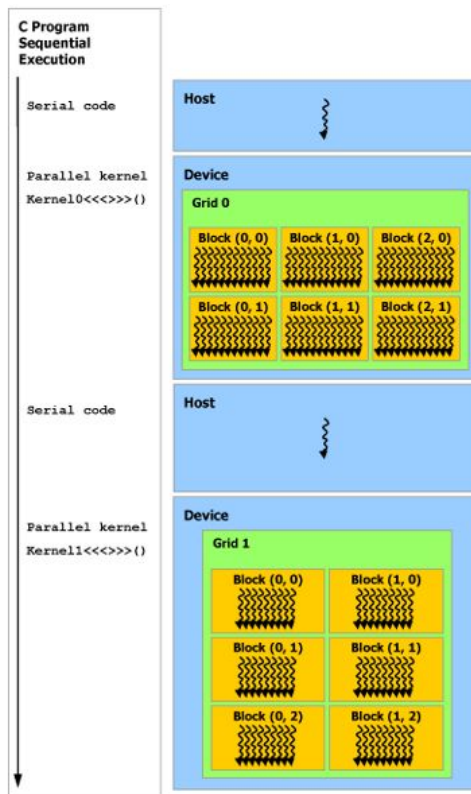
NVIDIA Ampere Architecture (compute capabilities 8.x)				Tesla A Series
NVIDIA Turing Architecture (compute capabilities 7.x)		GeForce 2000 Series	Quadro RTX Series	Tesla T Series
NVIDIA Volta Architecture (compute capabilities 7.x)	DRIVE/JETSON AGX Xavier		Quadro GV Series	Tesla V Series
NVIDIA Pascal Architecture (compute capabilities 6.x)	Tegra X2	GeForce 1000 Series	Quadro P Series	Tesla P Series

 Embedded	 Consumer Desktop/Laptop	 Professional Workstation	 Data Center
------------------------------------------------------------------------------------------------	-------------------------------------------------------------------------------------------------------------------	---------------------------------------------------------------------------------------------------------------------	-----------------------------------------------------------------------------------------------------

CUDA



Гетерогенное исполнение



Performance Guidelines

- **Максимизация параллельного выполнения** для достижения максимальной утилизации;
- **Оптимизация использования памяти** для достижения максимальной пропускной способности памяти;
- **Оптимизация использования инструкций** для достижения максимальной пропускной способности инструкций;

Максимизация утилизации

- Полная утилизация достигается, когда планировщики варпов всегда имеют какую-то инструкцию для выдачи для какого-то варпа в каждом тактовом цикле в течение периода latency
- Если какой-либо входной операнд находится в памяти вне чипа, latency намного выше: обычно сотни тактовых циклов.

Максимизация throughput memory

- Передача данных между хостом и GPU
 - pinned memory, page locked memory, mapped memory
- Доступы к памяти устройства
 - global memory
- Практическое применение в pytorch

Максимизация пропускной способности инструкций

- Арифметические инструкции с низкой пропускной способностью
 - половинная точность
 - обнуление денормализованных чисел

Compute Capability	5.0, 5.2	5.3	6.0	6.1	6.2	7.x	8.0	8.6	8.9	9.0
16-bit floating-point add, multiply, multiply-add	N/A	256	128	2	256	128	256 ³		128	256
32-bit floating-point add, multiply, multiply-add	128		64	128		64		128		
64-bit floating-point add, multiply, multiply-add	4		32	4		32 ⁵	32	2		64
32-bit floating-point reciprocal, reciprocal square root, base-2 logarithm (<code>__log2f</code>), base 2 exponential (<code>exp2f</code>), sine (<code>__sinf</code>), cosine (<code>__cosf</code>)	32		16	32		16				
32-bit integer add, extended-precision add, subtract, extended-precision subtract	128		64	128		64				
32-bit integer multiply, multiply-add, extended-precision multiply-add	Multiple instruct.					64 ⁶				

Recap

- Мультипроцессор
 - базовая вычислительная единица GPU
- Streaming Multiprocessor (SM)
 - другое название мультипроцессора в контексте CUDA
 - SM содержит набор функциональных единиц, таких как SP, блоки памяти и прочее
- Scalar Processor (SP)
 - Так же известен как CUDA Core / Tensor Core / RT Core
 - Базовая вычислительная единица внутри SM

Часть 3: Deep Learning Performance

girafe
ai

03

Цели:

- Научиться определять где бутлнек: в памяти или в арифметике
- Как избавиться от бутлнека
- Как подогнать обучение и инференс под конкретную архитектуру
- И прочее...

Арифметическая интенсивность

Большие слои имеют больше арифметических вычислений по отношению к количеству транзакций доступов к памяти. Это соотношение называется *арифметическая интенсивность*.

Используем тензорные ядра с уважением

Ключевые параметры операции

- Кратны 4 при использовании TF32
- Кратные 8 при использовании FP16
- Кратны 16 при использовании INT8

(то есть, когда ключевые размеры операции выровнены по кратным 16 байтам в память)

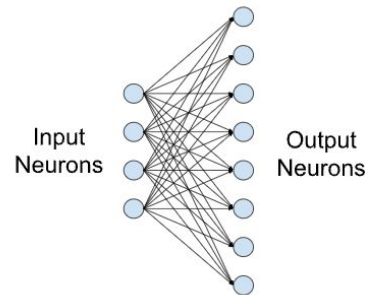
Что является ключевыми параметрами?

- Для полносвязных слоев ключевыми параметрами являются размер батча и количество входов и выходов;
- для сверточных слоев — количество входных и выходных каналов;
- для рекуррентных слоев — размер батча и скрытая размерность.

Размер батча (в общем случае)

- кратен большим степеням двойки (как минимум 65)
- Использование кратности более 512 особого прироста не дает
- Делимость на степень двойки наиболее важна для малых параметров:
 - выбор 512 вместо 520 дает больший прирост чем выбор 5120 вместо 5218

Линейные слои



TLDR

- Размер батча и количество входов и выходов выбирать так, чтобы они делились нацело на
 - 4 (TF32)
 - 8 (FP16)
 - 16 (INT8)

для эффективной работы на Tensor Cores. В случае A100 параметры должны делиться нацело на 32 (TF32) / 64 (FP16) / 128 (INT8);

- В случае если один или несколько гиперпараметров малы: размер батча должен делиться нацело как минимум на 64

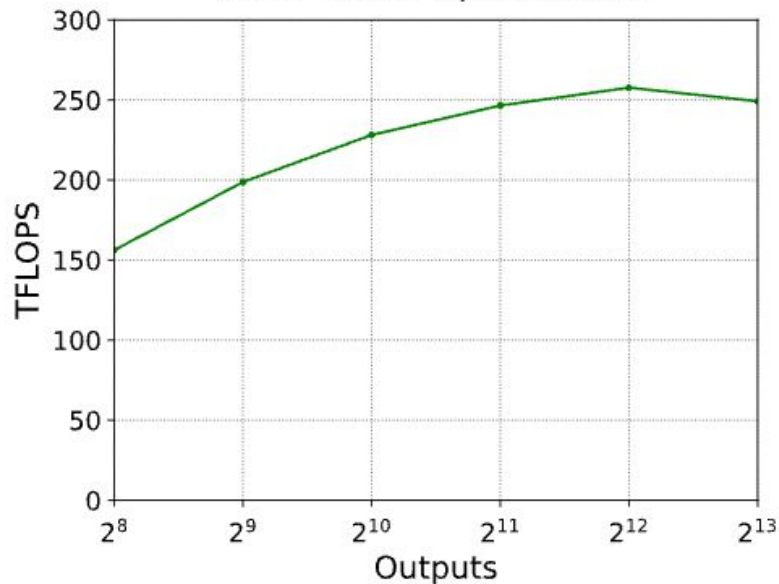
Перформанс и GEMM

Table 1. Mapping of inputs, outputs, and batch size to GEMM parameters M, N, K.

Computation Phase	M	N	K
Forward Propagation	Number of outputs	Batch size	Number of inputs
Activation Gradient	Number of inputs	Batch size	Number of outputs
Weight Gradient	Number of inputs	Number of outputs	Batch size

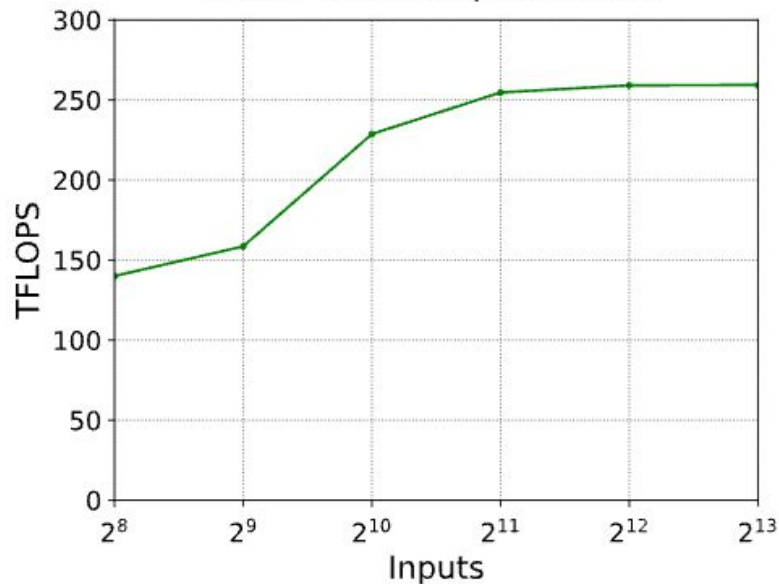
Перформанс и GEMM

Performance of Activation Gradient of
Fully-Connected Layer with Batch
Size = 4096, Inputs = 4096



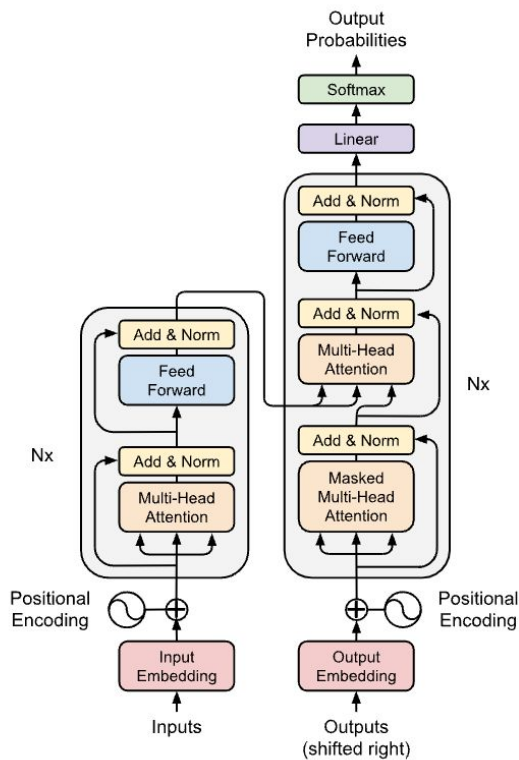
(a)

Performance of Weight Gradient of
Fully-Connected Layer with Batch
Size = 4096, Outputs = 4096



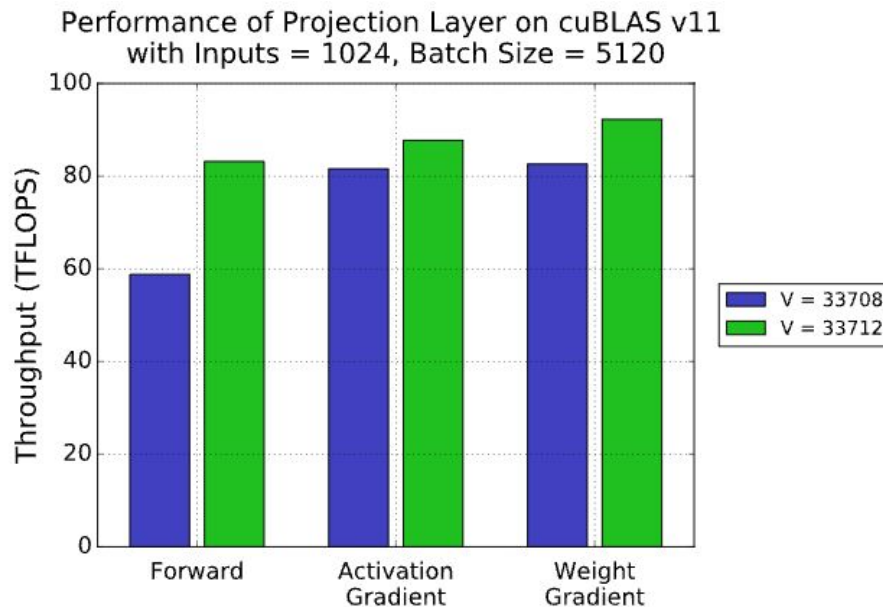
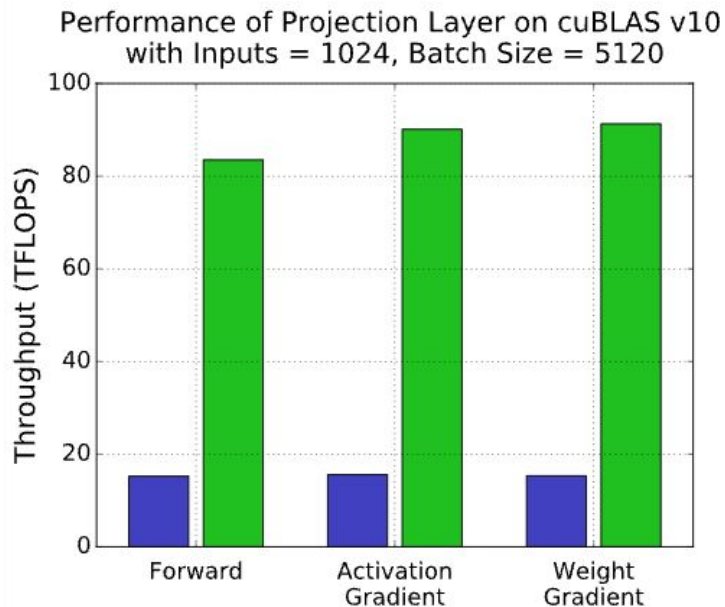
(b)

Полносвязные слои в трансформерах



Полносвязные слои в трансформерах

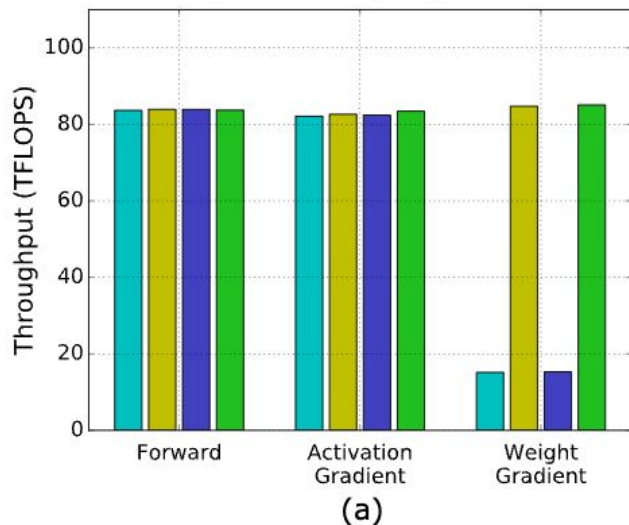
Шаг 1: Выравнивание размера словаря



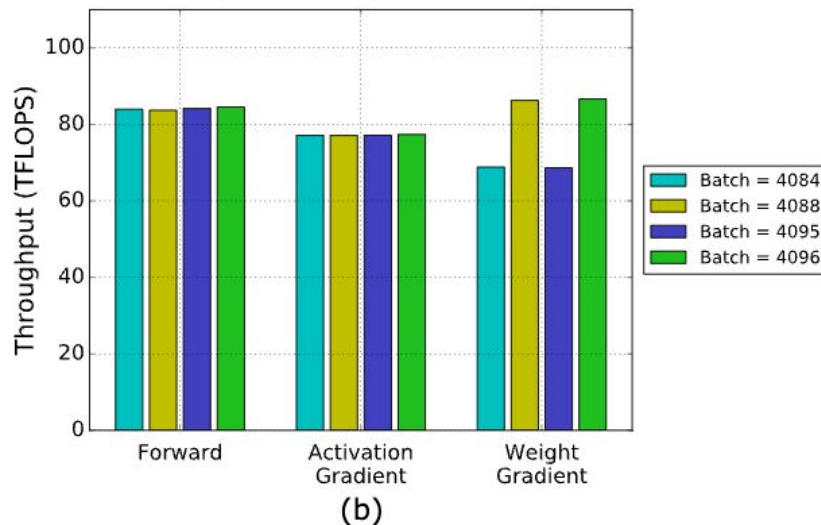
Полносвязные слои в трансформерах

Шаг 2: размер батча кратен 8

Performance of Feed-Forward Layer on cuBLAS v10
with Inputs = 1024, Outputs = 4096



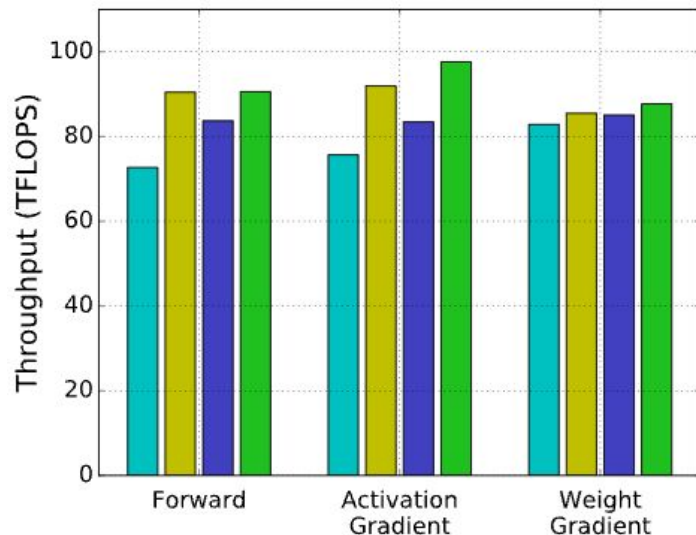
Performance of Feed-Forward Layer on cuBLAS v11
with Inputs = 1024, Outputs = 4096



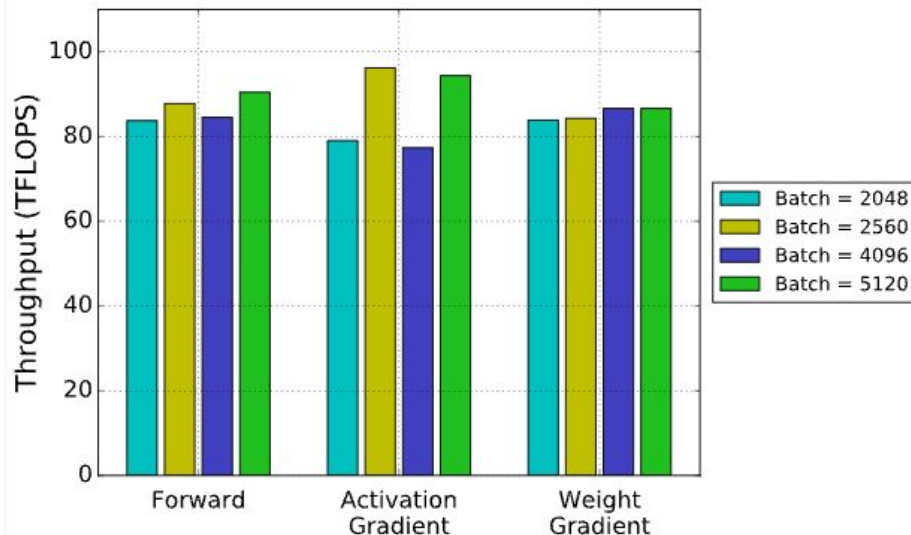
Полносвязанные слои в трансформерах

Шаг 3: Избегание Квантования Волн

Performance of Feed-Forward Layer on cuBLAS v10
with Inputs = 1024, Outputs = 4096



Performance of Feed-Forward Layer on cuBLAS v11
with Inputs = 1024, Outputs = 4096



Сверточные слои

TLDR

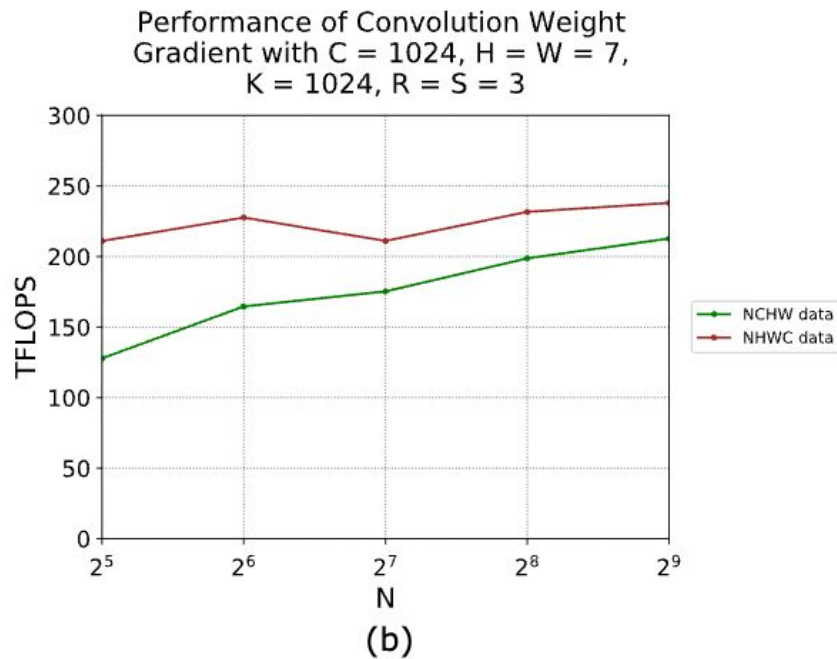
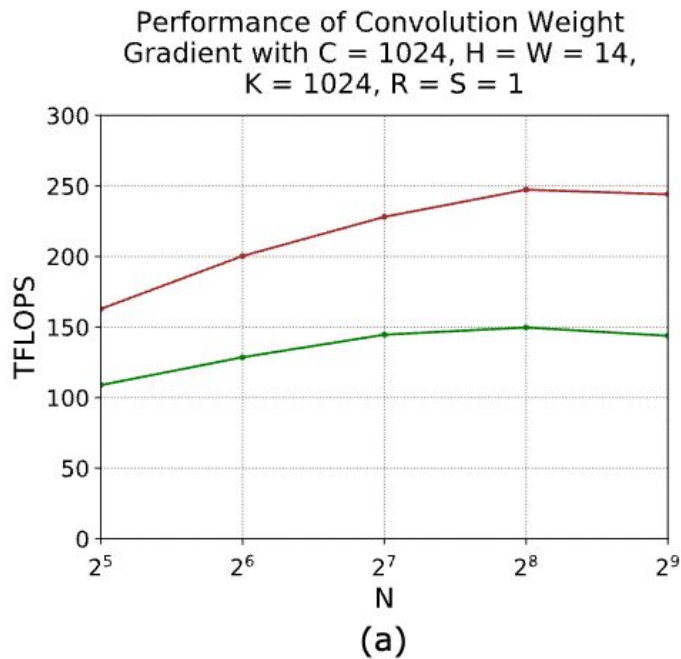
- Количество входных и выходных каналов, делящихся на 8 (для FP16) или 4 (для TF32), чтобы эффективно работать на Tensor Cores.
- Размер батча, количество входных и выходных каналов, делящиеся хотя бы на 64 и идеально на 256, чтобы уменьшить эффект квантования волн;
- Библиотеки NVIDIA предлагают набор различных алгоритмов свертки с различными характеристиками производительности, зависящими от параметров свертки. Когда размер входных данных, обрабатываемых сетью, одинаков в каждой итерации, автотюнинг является эффективным методом для обеспечения выбора идеального алгоритма для каждой свертки в сети. Для PyTorch включите автотюнинг, добавив `torch.backends.cudnn.benchmark = True`.
- Выбирайте расположение тензоров в памяти, чтобы избежать транспонирования входных и выходных данных. Существует две основные конвенции, каждая из которых названа в соответствии с порядком размерностей: NHWC и NCHW. Рекомендуется использовать формат NHWC, где это возможно.

Сверточные слои: тензорные ядра

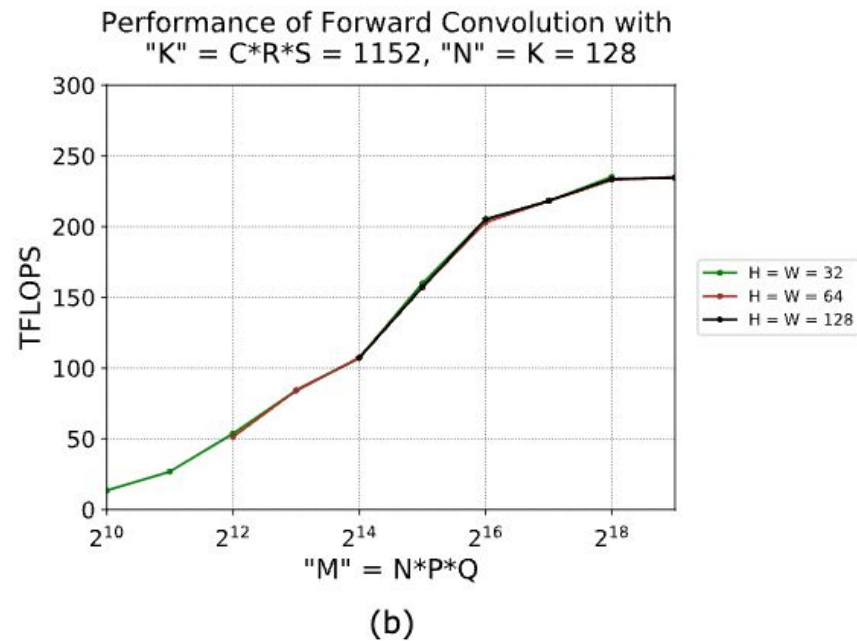
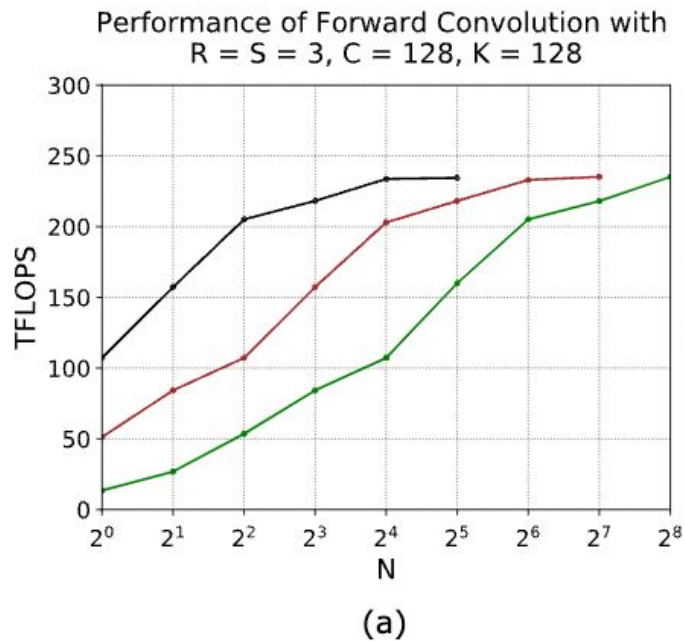
Table 2. Translation of convolution parameters to corresponding GEMM parameters

Computation Phase	"M"	"N"	"K"
Forward Propagation	$N \times P \times Q$	K	$C \times R \times S$
Activation Gradient	$N \times H \times W$	C	$K \times R \times S$
Weight Gradient	$C \times R \times S$	K	$N \times P \times Q$

Сверточные слои: тензорные ядра



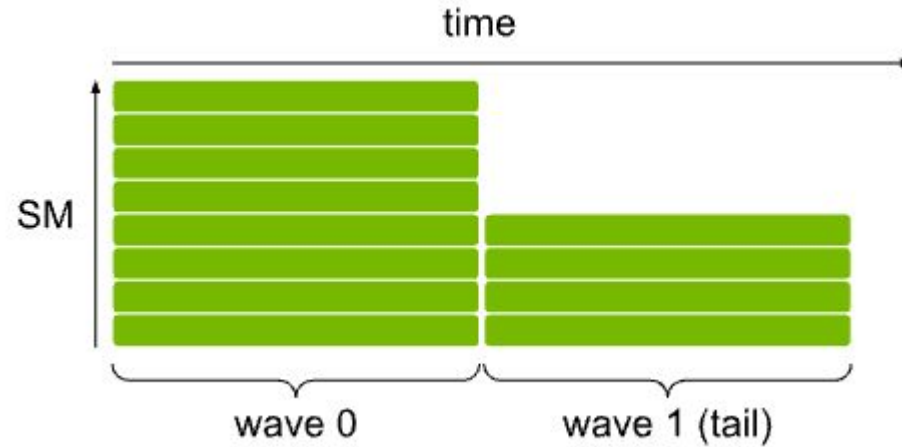
Размер батча, высота и ширина



Performance Background

	CUDA Cores				Tensor Cores					
NVIDIA Architecture	FP64	FP32	FP16	INT8	FP64	TF32	FP16	INT8	INT4	INT1
Volta	32	64	128	256			512			
Turing	2	64	128	256			512	1024	2048	8192
Ampere (A100)	32	64	256	256	64	512	1024	2048	4096	16384
Ampere, sparse						1024	2048	4096	8192	

Execution Model Background



Understanding Performance

Table 1. Examples of neural network operations with their arithmetic intensities. Limiters assume FP16 data and an NVIDIA V100 GPU.

Operation	Arithmetic Intensity	Usually limited by...
Linear layer (4096 outputs, 1024 inputs, batch size 512)	315 FLOPS/B	arithmetic
Linear layer (4096 outputs, 1024 inputs, batch size 1)	1 FLOPS/B	memory
Max pooling with 3x3 window and unit stride	2.25 FLOPS/B	memory
ReLU activation	0.25 FLOPS/B	memory
Layer normalization	< 10 FLOPS/B	memory

Спасибо за внимание!

Жду вопросов и обсуждений

