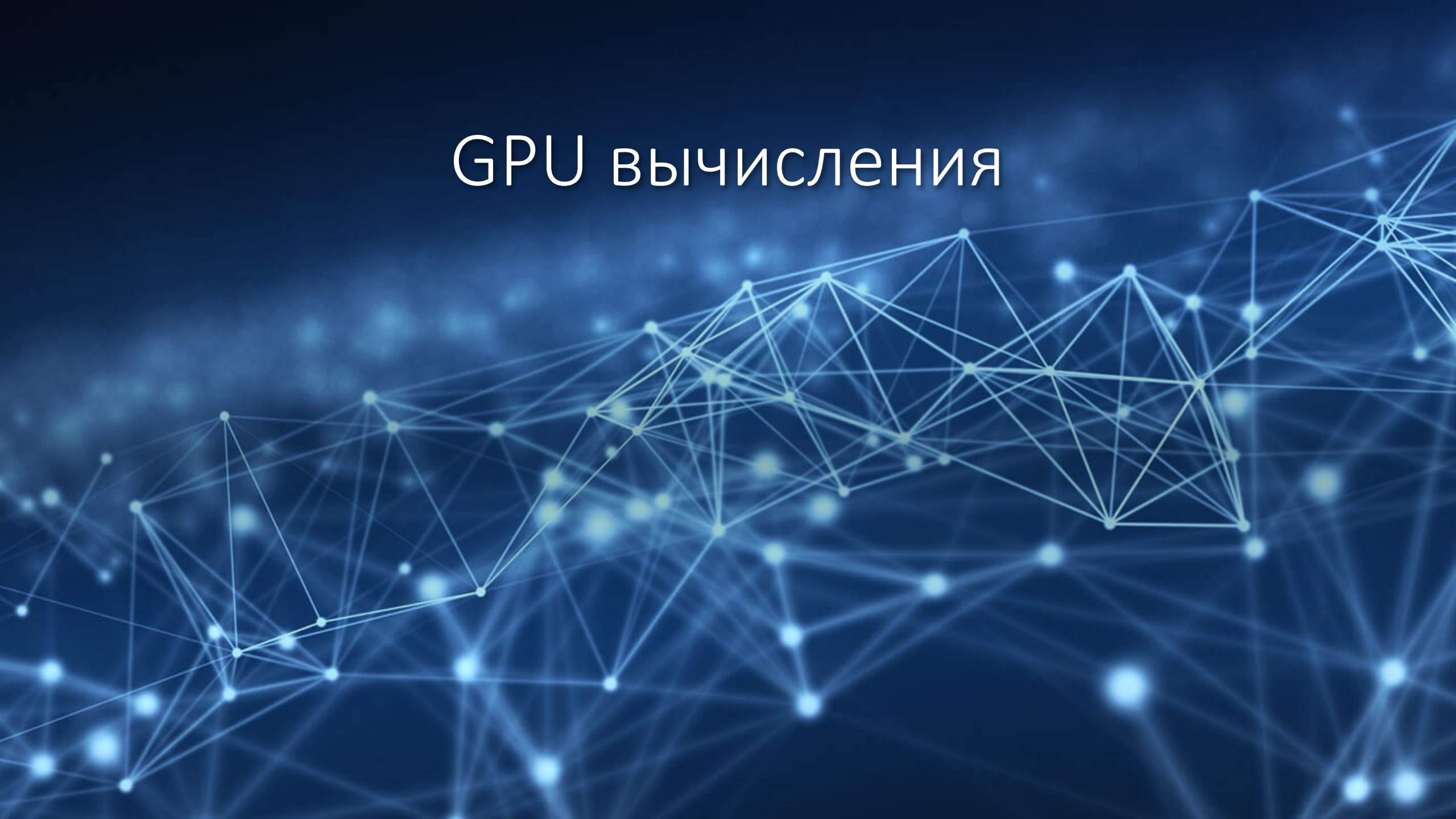
GPU вычисления



О чем будет курс

1. Вводная информация про GPU вычисления
2. Оптимизация кода при помощи Numba (CPU и GPU)
3. CuPy и Rapids (GPU и multi-GPU)
4. ML на GPU (GPU и multi-GPU)
5. Анализ графов (GPU и multi-GPU)
6. Оптимизация инференса нейронных сетей

Что такое CPU

CPU – central processing unit

CPU является универсальным и способным выполнять огромное число разнообразных задач за счет заложенных разработчиками инструкций внутри.

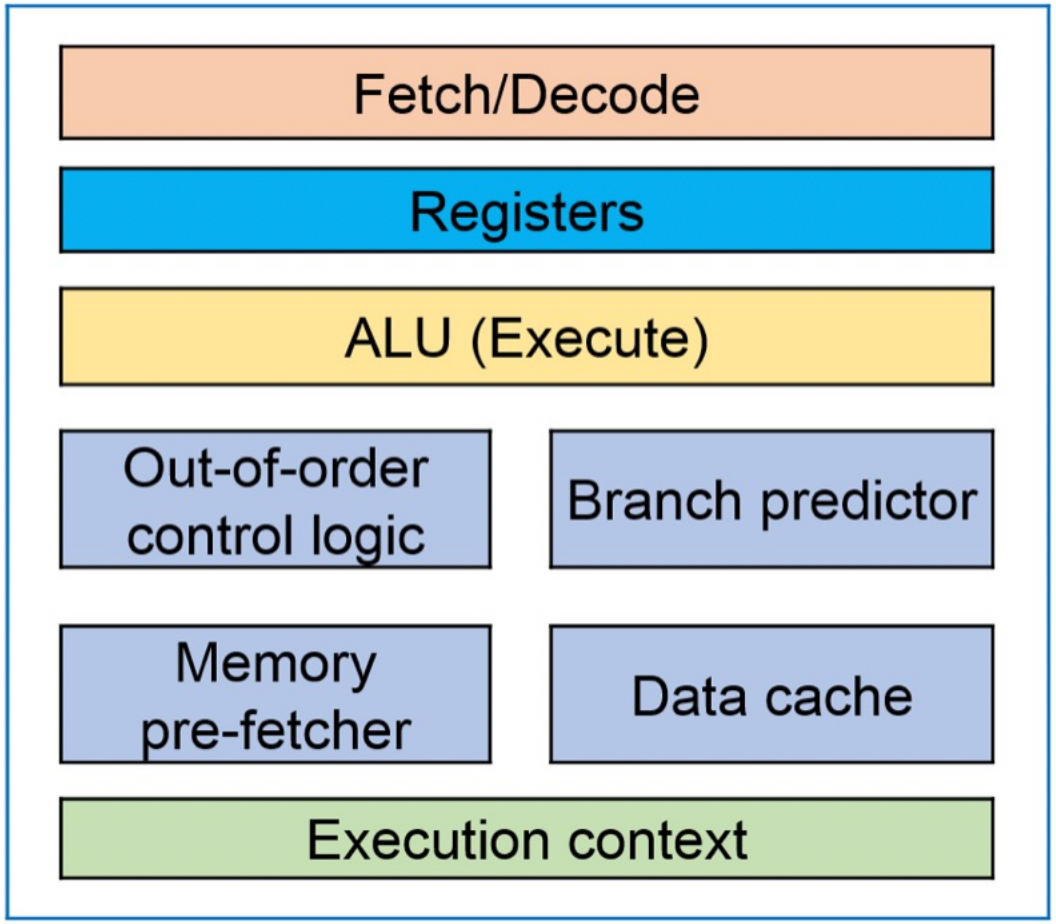
Создан для большого спектра задач, особенно тех, где важно время задержки и производительность одного ядра.

Архитектура CPU

ALU – arithmetic logic unit. Блок процессора, который служит для выполнения арифметических и логических преобразований.

Характерные особенности:

- Небольшое число ядер
- Высокая тактовая частота (количество операций, выполняемых за единицы времени, обычно это 1 секунда)
- Способен выполнять большой спектр задач



А что у GPU

GPU – graphics processing unit.

Специализированный тип процессора, который первоначально был создан для решения задачи обработки видео и оптимизирован под данную задачу.

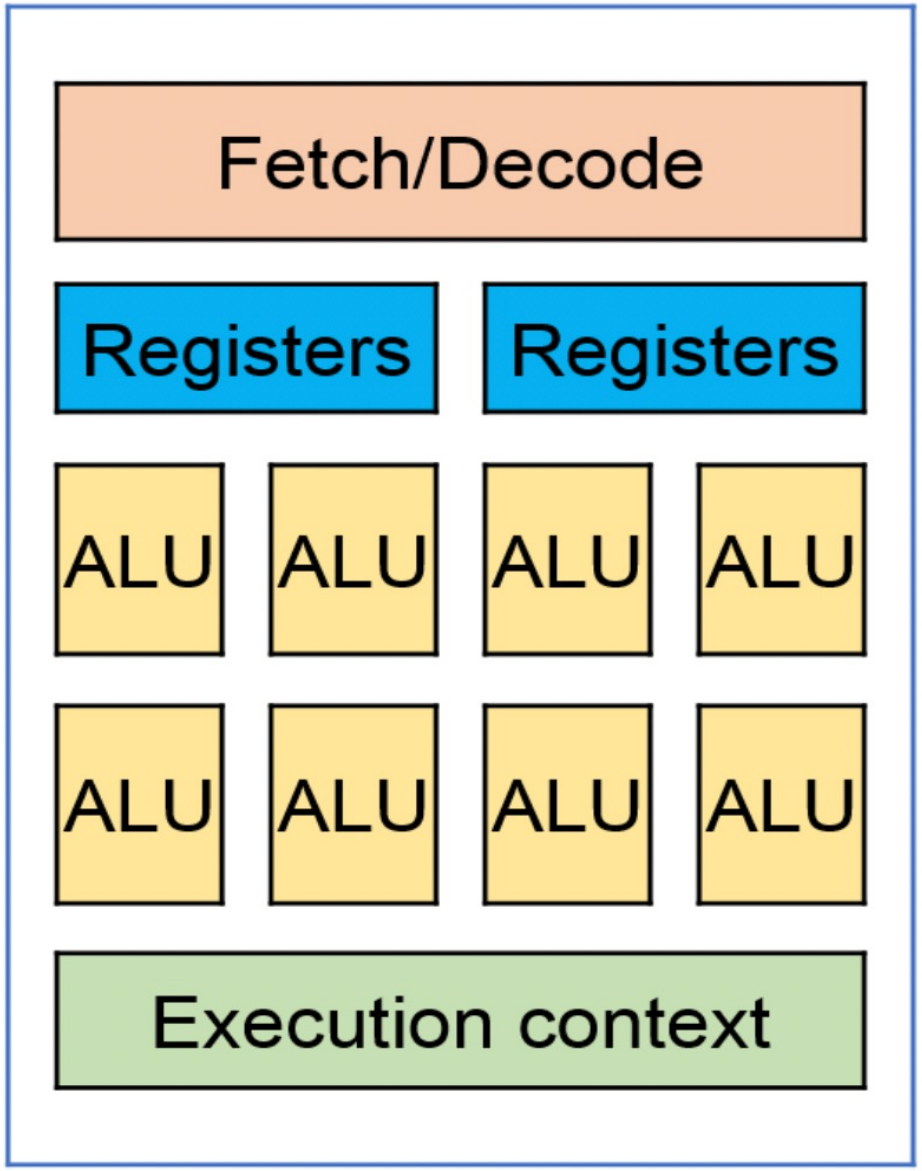
У видеокарты множество вычислительных ядер, обычно несколько тысяч, но они объединены в блоки, для видеокарт NVIDIA обычно по 32, и имеют общие элементы, в т.ч. и регистры. Архитектура ядра GPU и логических элементов существенно проще, чем на CPU, а именно, нет префетчеров, бранч-предикторов и много чего еще.

Архитектура GPU

Архитектура упрощена по сравнению с CPU.

Характерные особенности:

- Огромное число ядер
- Низкая частота
- Выполняет только простые специализированные задачи



Должно быть что-то не так

Что дает?

Дает быстрые вычисления для простых операций, которые можно параллелить!

Что берет взамен?

- Так как процессоры находятся в блоках, то мы не можем выделить только 1 ядро, выделяется сразу весь блок
- Ядра выполняют одну и туже инструкцию, но с разными данными
- GPU не любит ветвлений в логике расчетов и сложных операций

Сходства CPU и GPU

- Вычислительные движки
- Кремневые микропроцессоры
- Умеют обрабатывать данные

А где тут про нейронки и прочее?

Появился термин GPGPU – general-purpose graphics processing unit. Идея заключается в использовании GPU не только для графичеких вычислений, но и для общих вычислений.

Это стало возможным благодаря добавлению программируемых шейдерных блоков (программа, предназначенная для выполнения задач графическими процессорами) и более высокой арифметической точности растровых конвейеров, что позволяет разработчикам ПО использовать потоковые процессоры для не-графических данных.

Основные технологии для работы с видеокартами

- CUDA
- OpenCL
- AMD FireStream ранее, сейчас AMD FirePro
- OpenACC
- C++ AMP

CUDA

CUDA - *Compute Unified Device Architecture.*

Архитекутра параллельных вычислений от компании Nvidia.

- Для разработки используются языки CUDA C/C++, Fortran и другие
- Поддержка только на видеокартах Nvidia
- Очень популярна (много библиотек, большое сообщество, документация, инструменты отладки)

Основная идея заключается в следующем:

Host – CPU, на котором выполняется основная программа.
Device – GPU, на котором будут запускаться параллельные вычисления.

OpenCL

OpenCL – *Open Computing Language*

Является open source проектом для написания программ с параллельным вычислением на графических процессорах. Стандарт является полностью свободным для использования.

Разработка может вестись на языках, базирующихся на стандарте C99.

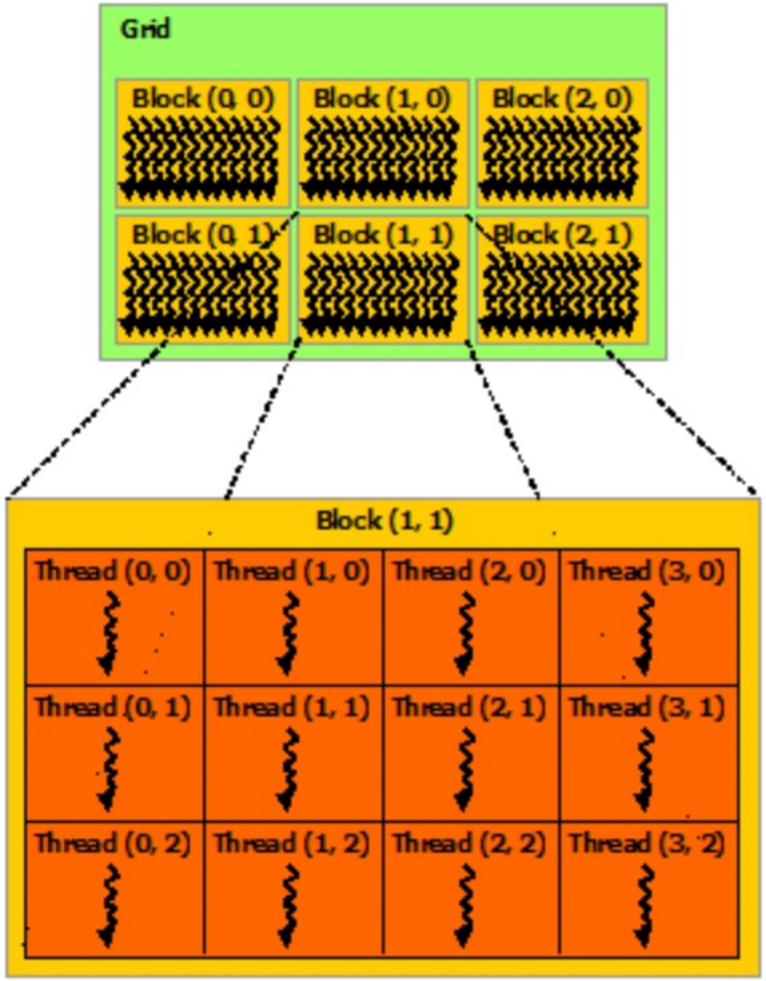
Поддерживается большинством языком. Можно использовать с видеокартами различных производителей.

Как это работает в CUDA?

Данные разбиваются на блоки для обработки.

Для каждого блока выделяются потоки, которые будут обрабатывать данные и имеют свой уникальный идентификатор.

Grid(сетка) – совокупность всех потоков, запущенных в рамках одной задачи.



https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html

Как подбирать количество блоков?

Начнем с основ…

Видеокарты состоят из потоковых мультипроцессоров (SM – Streaming Multiprocessor), внутри которых в том числе и живут наши волшебные ядра cuda.

Чем больше процессоров и ядер, тем дороже и производительнее видеокарта. (ну не забываем про частоту тоже)

Так вот, один блок вычислений целиком выполняется на потоковом процессоре, где 1 поток занимает 1 вычислительное ядро.

Рулит всем планировщик внутри SM.

Как подбирать количество блоков?

1 потоковый процессор содержит какое-то количество cuda-ядер (например, 32 ядра).

Так как поток занимает целое ядро, то если мы зададим размер блока 1 на 1, то 1 ядро будет занято, а 31 простаивает.

Плохо, ведь по умолчанию куда разбивает наш блок на 32 подряд идущих потока (warp) для выполнения задачи.
Нет смысла брать блоки размером не кратным размеру warp.

Если есть условные операторы только для части потоков, то если из 32 30 без условий, они будут ждать остальные 2 потока и простаивать.

Как подбирать количество блоков?

- Исходим из задачи
- Ставим эксперименты
- Читаем документацию по видеокарте и изучаем (тут размер warp, количество и мощность ядер в зависимости от точности чисел и т.д.)

Что с Tesla V100?

Table 1. Comparison of NVIDIA Tesla GPUs

Tesla Product	Tesla K40	Tesla M40	Tesla P100	Tesla V100
GPU	GK180 (Kepler)	GM200 (Maxwell)	GP100 (Pascal)	GV100 (Volta)
SMs	15	24	56	80
TPCs	15	24	28	40
FP32 Cores / SM	192	128	64	64
FP32 Cores / GPU	2880	3072	3584	5120
FP64 Cores / SM	64	4	32	32
FP64 Cores / GPU	960	96	1792	2560
Tensor Cores / SM	NA	NA	NA	8
Tensor Cores / GPU	NA	NA	NA	640
GPU Boost Clock	810/875 MHz	1114 MHz	1480 MHz	1530 MHz
Peak FP32 TFLOPS[1]	5	6.8	10.6	15.7
Peak FP64 TFLOPS[1]	1.7	.21	5.3	7.8
Peak Tensor TFLOPS[1]	NA	NA	NA	125
Texture Units	240	192	224	320
Memory Interface	384-bit GDDR5	384-bit GDDR5	4096-bit HBM2	4096-bit HBM2
Memory Size	Up to 12 GB	Up to 24 GB	16 GB	16 GB
L2 Cache Size	1536 KB	3072 KB	4096 KB	6144 KB
Shared Memory Size / SM	16 KB/32 KB/48 KB	96 KB	64 KB	Configurable up to 96 KB
Register File Size / SM	256 KB	256 KB	256 KB	256KB

Tensor Cores – ядра, которые имеют специальные оптимизированные инструкции для работы с разреженными матрицами (sparse matrix). Особенно классно они перемножают матрицы.

https://images.nvidia.com/content/volta-architecture/pdf/volta-architecture-whitepaper.pdf