ФЕДЕРАЛЬНОЕ АГЕНТСТВО СВЯЗИ

Федеральное государственное образовательное учреждение высшего профессионального образования ордена Трудового Красного Знамени

«МОСКОВСКИЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ СВЯЗИ И ИНФОРМАТИКИ»

МКиИТ

**ЛАБОРАТОРНАЯ РАБОТА №1**

По дисциплине «Распределенные и параллельные вычисления»

На тему **«Обращение к устройству»**

Выполнил студент 2 курса группы МБД2032:

Трояновский Семён Владиславович

Москва

2021

**Задача лабораторной работы:**

Получить представление об аппаратных ресурсах CUDA наряду с их возможностями.

Вывести аппаратные характеристики:

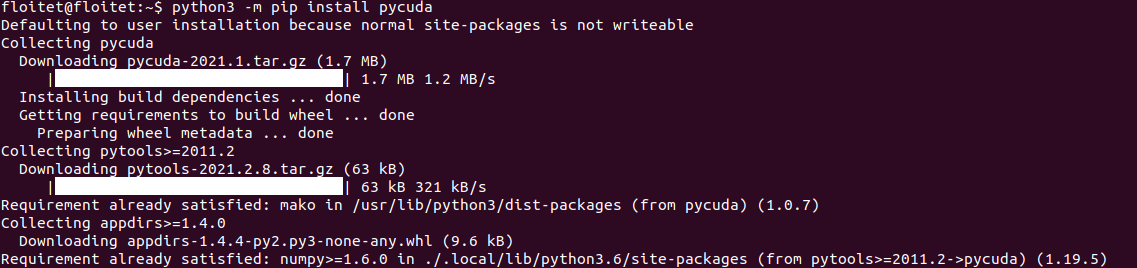
* название карты GPU
* GPU computation capability
* максимальная размерность блока
* максимальная размерность сетки
* размер видеопамяти
* размер постоянной и разделяемой памяти
* размер варпа

**Ход выполнения работы:**

1. **Установка библиотеки.**

Для выполнения задачи я буду использовать официальную Python библиотеку NVIDIA PyCUDA. Данная библиотека предоставляет удобное API для разработки с использованием ресурсов GPU. Установка библиотеки производится стандартно:

python3 -m pip install pycuda



1. **Получение и вывод данных.**

С помощью PyCUDA можно получить данные о GPU выполнив следующий код:

import json

from dataclasses import dataclass

import pycuda.autoinit

import pycuda.driver as cuda

import pycuda.tools

import pprint

@dataclass

class GpuProperties:

ComputeCapabilityMajor = pycuda.driver.device\_attribute.COMPUTE\_CAPABILITY\_MAJOR

ComputeCapabilityMinor = pycuda.driver.device\_attribute.COMPUTE\_CAPABILITY\_MINOR

MaxConstantMemory = pycuda.driver.device\_attribute.TOTAL\_CONSTANT\_MEMORY

SharedMemPerBlock = pycuda.driver.device\_attribute.MAX\_SHARED\_MEMORY\_PER\_BLOCK

MaxBlockDimX = pycuda.driver.device\_attribute.MAX\_BLOCK\_DIM\_X

MaxBlockDimY = pycuda.driver.device\_attribute.MAX\_BLOCK\_DIM\_Y

MaxBlockDimZ = pycuda.driver.device\_attribute.MAX\_BLOCK\_DIM\_Z

MaxGridDimX = pycuda.driver.device\_attribute.MAX\_GRID\_DIM\_X

MaxGridDimY = pycuda.driver.device\_attribute.MAX\_GRID\_DIM\_Y

MaxGridDimZ = pycuda.driver.device\_attribute.MAX\_GRID\_DIM\_Z

WarpSize = pycuda.driver.device\_attribute.WARP\_SIZE

@dataclass

class GpuCapability:

def \_\_init\_\_(self, major: int, minor: int):

self.minor = minor

self.major = major

def serializable(self):

return self.\_\_dict\_\_

@dataclass

class Dimensions:

def \_\_init\_\_(self, x: int, y: int, z: int):

self.z = z

self.y = y

self.x = x

def serializable(self):

return self.\_\_dict\_\_

@dataclass

class GpuInfo:

def \_\_init\_\_(self, name: str, gpu\_capability: GpuCapability, max\_constant\_mem: int, total\_mem: int,

shared\_mem\_per\_block: int, max\_block\_dim: Dimensions, max\_grid\_dim: Dimensions, warp\_size: int):

self.total\_mem = total\_mem

self.warp\_size = warp\_size

self.max\_grid\_dim = max\_grid\_dim

self.max\_block\_dim = max\_block\_dim

self.shared\_mem\_block = shared\_mem\_per\_block

self.max\_constant\_mem = max\_constant\_mem

self.gpu\_capability = gpu\_capability

self.name = name

def serializable(self):

return self.\_\_dict\_\_

def NestedSerializer(Obj):

if hasattr(Obj, 'serializable'):

return Obj.serializable()

else:

raise TypeError('Object of type %s with value of %s is not JSON serializable' % (type(Obj), repr(Obj)))

# Set device and collect all info

gpu\_device = cuda.Context.get\_device()

dev\_all\_props = cuda.Device.get\_attributes(gpu\_device)

# Start retrieve required values

dev\_name = pycuda.tools.cuda.Device.name(gpu\_device)

dev\_total\_memory = cuda.mem\_get\_info()[1]

dev\_capability = GpuCapability(major=dev\_all\_props[GpuProperties.ComputeCapabilityMajor],

minor=dev\_all\_props[GpuProperties.ComputeCapabilityMinor])

dev\_constant\_mem = dev\_all\_props[GpuProperties.MaxConstantMemory]

dev\_shared\_mem\_per\_block = dev\_all\_props[GpuProperties.SharedMemPerBlock]

dev\_block\_max\_dim = Dimensions(x=dev\_all\_props[GpuProperties.MaxBlockDimX],

y=dev\_all\_props[GpuProperties.MaxBlockDimY],

z=dev\_all\_props[GpuProperties.MaxBlockDimZ])

dev\_grid\_max\_dim = Dimensions(x=dev\_all\_props[GpuProperties.MaxGridDimX],

y=dev\_all\_props[GpuProperties.MaxGridDimY],

z=dev\_all\_props[GpuProperties.MaxGridDimZ])

dev\_warp\_size = dev\_all\_props[GpuProperties.WarpSize]

# Create dto

full\_info = GpuInfo(name=dev\_name,

gpu\_capability=dev\_capability,

max\_constant\_mem=dev\_constant\_mem,

total\_mem=dev\_total\_memory,

max\_grid\_dim=dev\_grid\_max\_dim,

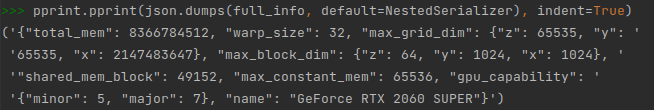
max\_block\_dim=dev\_block\_max\_dim,

shared\_mem\_per\_block=dev\_shared\_mem\_per\_block,

warp\_size=dev\_warp\_size)

# Print

pprint.pprint(json.dumps(full\_info, default=NestedSerializer))

****

1. **Сохранение в файл**

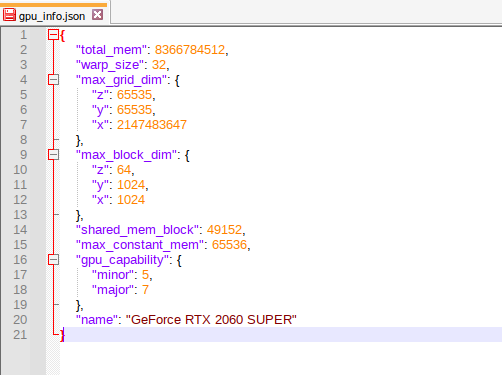
Также результаты можно сохранить в файл:

# Store

with open('gpu\_info.json', 'w') as f:

json.dump(full\_info, f, default=NestedSerializer)

В итоге мы полученная информация будет сохранена в удобно для дальнейшего использования формате:



1. **Ответы на вопросы**
2. Какая версия compute capability у архитектуры NVIDIA Fermi?

*2.0, 2.1*

1. Какая максимальная размерность блока у GPU с compute capability 3.0?

Maximum x-dimension of a grid of thread blocks 2^31-1

Maximum y- or z-dimension of a grid of thread blocks 65535

1. Предположим, вы используете одномерную сетку и блок. Если максимальная размерность сетки на устройстве 65535, а максимальная размерность блока 512, какое максимальное количество нитей может быть запущено на GPU?

*65535 \* 512 = 33.553.920*

1. При каких условиях программист предпочтет не запускать максимальное количество нитей?

*В целом, мы хотим, чтобы размер блоков / сетки соответствовал данным и одновременно максимизировал занятость, то есть количество потоков, активных одновременно. Основными факторами, влияющими на занятость, являются использование разделяемой памяти, использование регистров и размер блока потока.*

1. Что может помешать программе запустить максимальное количество нитей?

*Проблемы с памятью при тяжелых вычислениях.*

1. Что такое разделяемая память?

*Общая память - это мощная функция для написания хорошо оптимизированного кода CUDA. Доступ к общей памяти намного быстрее, чем доступ к глобальной памяти, поскольку она расположена на чипе. Поскольку общая память совместно используется потоками в блоке потока, она обеспечивает механизм взаимодействия потоков.*

1. Что такое глобальная память?

*Глобальная память - это память с высокой задержкой (самая медленная на рисунке). Чтобы увеличить арифметическую нагрузку нашего ядра, мы хотим уменьшить как можно больше обращений к глобальной памяти. Следует отметить, что в отношении глобальной памяти нет ограничений на то, какие потоки могут к ней обращаться. Все потоки любого блока могут получить к нему доступ. Нет никаких ограничений, как в случае с разделяемой памятью или регистрами.*

1. Что такое константная память?

*Постоянная память: графические процессоры NVIDIA предоставляют постоянную память, которая обрабатывается иначе, чем стандартная глобальная память. В некоторых ситуациях использование постоянной памяти вместо глобальной может уменьшить пропускную способность памяти (что выгодно для ядер). Постоянная память также наиболее эффективна, когда все потоки одновременно обращаются к одному и тому же значению (т. Е. Индекс массива не является функцией позиции).*

1. Что характеризует размер варпа в GPU?

Размер варп - это количество потоков в варп, которая представляет собой подразделение, используемое в аппаратной реализации для объединения доступа к памяти и отправки команд.

1. **Поддерживаются ли числа двойной точности в GPU версии 1.3?**

*Да*

***Вывод:***

В ходе выполнения данной работы я получил представление об аппаратных ресурсах CUDA наряду с их возможностями, а также использовал Python библиотеку для получения информации о GPU.