Open in Colab

Средства интеграции языка **Python** с инструментами параллельного программирования на платформе **NVIDIA CUDA®.**

Практическое занятие по дисциплине "Интеграция компонентов ИС"

*Кафедра КБ-**3** "Управление и моделирование систем" Института Комплексной Безопасности и Специального Приборостроения, МИРЭА, **2020**г.*

Дата проведения занятия: 19.03.2020.

Время: 18:20-21:20.

Место/Форма проведения занятия: Дистанционное обучение/Конференция **Skype.** Ссылка на видео-конференцию будет предоставлена непосредственно перед занятием по почте**.**

Цель: Изучение платформы **NVIDIA CUDA®,** закрепление навыков использования библиотек **Numba® (**http://numba.pydata.org/**)** и **PyCUDA (**https://documen.tician.de/pycuda/**, Andreas Klöckner, Nicolas Pinto, Yunsup Lee, Bryan Catanzaro, Paul Ivanov, Ahmed Fasih, PyCUDA and PyOpenCL: A scripting-based approach to GPU run-time code generation, Parallel Computing, Volume 38, Issue 3, March 2012, Pages 157-174).** Реализация алгоритма двумерной фильтрации **(**применение Лапласиана к изображению**)** на платформе **CUDA®.**

Технология параллельных с использованием NVIDIA GPU

CUDA® - это аппаратная платформа для параллельных вычислений на базе **GPU** и программная среда для реализации высокопроизводительных параллельных вычислений разработанная компанией **NVIDIA (NVIDIA Corporation,** https://www.nvidia.com/en-us/about-nvidia/legal-info/**).**

Подробно **-->** https://developer.nvidia.com/cuda-zone

Компания **NVIDIA** представила технологию **CUDA®** в Ноябре **2006** года**,** охарактеризовав ее как аппаратную платформу для параллельных вычислений общего назначения и программную модель**,** реализующую концепцию параллельных вычислений на базе графических ускорителей **NVIDIA GPU.**

Основные отличия между архитектурой **CPU** и **GPU** представлены на схеме**:** https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html#from-graphics-processing-to-general-purpose-parallel-computing__gpu-devotes-more-transistors-to-data-processing

Значительная часть ресурсов **CPU** используется для кэширования данных и управления исполнением **(**организация ветвлений и циклов**).** Архитектура **GPU** направлена на параллельную обработку векторизованных данных без ветвлений**.**

Отличия в производительности вычислений между **CPU** и **GPU** представлены на графиках по ссылке**:**

https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html#from-graphics-processing-to-general-purpose-parallel-computing__floating-point-operations-per-second-for-cpu-and-gpu

https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html#from-graphics-processing-to-general-purpose-parallel-computing__memory-bandwidth-for-cpu-and-gpu

CUDA® включает программные средства которые позволяют использовать язык **C++** для разработки высокопроизводительных параллельных алгоритмов**.** Также могут быть использованы другие языки высокого уровня**.**

см**.** https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html#cuda-general-purpose-parallel-computing-architecture__cuda-is-designed-to-support-various-languages-and-application-programming-interfaces

Базовым програмным инструментом для разработки приложений с использованием **CUDA® Runtime** является библиотека **CUDA® Toolkit**

библиотека **CUDA® Toolkit.**

Подробно **-->** **https://docs.nvidia.com/cuda/index.html.**

Основными сущностями программной модели **CUDA®** являются**:** ***Kernels*** **(**ядра**)** и ***Threads*** **(**потоки**).**

Kernel (ядро**) -** это функция**,** написанная на языке **C++,** которая вызывается и исполняется парралельно в контексте N различных потоков **(Threads).**

Пример ядра**:** **https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html#kernels**

Для использования **CUDA® Toolkit** в языке **Python** в подавляющем большинстве приложений используются библиотеки **Numba® (http://numba.pydata.org/)** и **PyCUDA (https://documen.tician.de/pycuda/).**

Библиотека **PyCUDA** использует вставки на языке **C++** для описания **kernel-**функций**.**

Библиотека **Numba®** использует **JIT (Just-In-Time)** компилятор и позволяет использовать подмножество языка **Python** и **Numpy** для создания **CUDA®** приложений **(**без использования кода на **C++).**

Для установки **pyCuda** в среде **Google Colaboration** следует использовать следующие команды**:**

In [0]:

```
pip install pycuda
```

```
Collecting pycuda
  Downloading https://files.pythonhosted.org/packages/5e/3f/5658c38579b41866ba21ee1b5020b
8225cec86fe717e4b1c5c972de0a33c/pycuda-2019.1.2.tar.gz (1.6MB)
     |████████████████████████████████| 1.6MB 35.8MB/s
Collecting pytools>=2011.2
  Downloading https://files.pythonhosted.org/packages/66/c7/88a4f8b6f0f78d0115ec3320861a0
cc1f6daa3b67e97c3c2842c33f9c089/pytools-2020.1.tar.gz (60kB)
     |████████████████████████████████| 61kB 10.1MB/s
Requirement already satisfied: decorator>=3.2.0 in /usr/local/lib/python3.6/dist-packages
(from pycuda) (4.4.2)
Collecting appdirs>=1.4.0
  Downloading https://files.pythonhosted.org/packages/56/eb/810e700ed1349edde4cbdc1b2a21e
28cdf115f9faf263f6bbf8447c1abf3/appdirs-1.4.3-py2.py3-none-any.whl
Collecting mako
  Downloading https://files.pythonhosted.org/packages/50/78/f6ade1e18aebda570eed33b7c5343
78d9659351cadce2fcbc7b31be5f615/Mako-1.1.2-py2.py3-none-any.whl (75kB)
     |████████████████████████████████| 81kB 13.6MB/s
Requirement already satisfied: six>=1.8.0 in /usr/local/lib/python3.6/dist-packages (from
pytools>=2011.2->pycuda) (1.12.0)
Requirement already satisfied: numpy>=1.6.0 in /usr/local/lib/python3.6/dist-packages (fr
om pytools>=2011.2->pycuda) (1.18.1)
Requirement already satisfied: MarkupSafe>=0.9.2 in /usr/local/lib/python3.6/dist-package
s (from mako->pycuda) (1.1.1)
Building wheels for collected packages: pycuda, pytools
  Building wheel for pycuda (setup.py) ... done
  Created wheel for pycuda: filename=pycuda-2019.1.2-cp36-cp36m-linux_x86_64.whl size=453
7838 sha256=cc68f5f5b05e969a62d2f61515eb837896b78750041f0775bd3edb4221ec98d3
  Stored in directory: /root/.cache/pip/wheels/a6/60/f0/b1c430c73d281ac3e46070480db50f790
7364eb6f6d3188396
  Building wheel for pytools (setup.py) ... done
  Created wheel for pytools: filename=pytools-2020.1-py2.py3-none-any.whl size=59602 sha2
56=0b402ddb0335538b3bc53dd915c67c45a2cd032af12ddc4c34cfcf1dcecbd2eb
  Stored in directory: /root/.cache/pip/wheels/6f/da/1b/946775a88291378182ed92c9800d6d0eb
c2a554cb89829cc24
Successfully built pycuda pytools
Installing collected packages: appdirs, pytools, mako, pycuda
Successfully installed appdirs-1.4.3 mako-1.1.2 pycuda-2019.1.2 pytools-2020.1
```

In [0]:

```
!curl https://colab.chainer.org/install | sh -
```

Пример кода**,** реализующий вычисления с помощью **PyCUDA:**

In [0]:

```python
import pycuda.autoinit
import pycuda.driver as drv
import numpy

from pycuda.compiler import SourceModule
mod = SourceModule("""
__global__ void multiply_them(float *dest, float *a, float *b)
{
  const int i = threadIdx.x;

  dest[i] = a[i] * b[i];
}
""")

multiply_them = mod.get_function("multiply_them")

a = numpy.random.randn(400).astype(numpy.float32)
b = numpy.random.randn(400).astype(numpy.float32)

dest = numpy.zeros_like(a)
multiply_them(
        drv.Out(dest), drv.In(a), drv.In(b),
        block=(width,height,1), grid=(1,1))

print( dest-a*b )
```

```
[ 0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
  0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
  0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
  0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
  0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
  0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
  0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
  0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
  0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
  0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
  0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
  0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
  0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
  0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
  0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
  0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
  0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
  0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
  0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
  0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
  0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
  0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
  0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
  0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
  0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
  0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
  0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
  0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
  0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
  0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
  0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
  0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
  0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
  0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
  0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
  0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
  0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
  0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
  0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
  0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
  0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
  0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
  0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
  0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
  0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
  0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
```

```
  0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
  0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
  0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
  0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
  0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
  0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
  0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
  0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
  0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
  0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
  0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
  0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
  0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
  0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
  0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
  0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
  0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
  0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
  0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
  0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
  0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
  0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
  0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
  0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
  0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
  0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
  0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
  0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
  0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
  0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
  0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
  0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
  0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
  0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
  4.41706879e-03 -5.93006253e-01  6.97807893e-02  4.62284386e-01
 -2.42393523e-01 -1.45455524e-01 -1.36915008e-02  3.57694566e-01
  1.90945074e-01  2.17070162e-01  3.15717965e-01 -1.61096081e-01
 -2.88717955e-01 -4.58446681e-01 -1.31431118e-01  2.95538902e-01
  1.05959279e-02  1.13890588e+00 -7.29831338e-01 -1.32285655e-01
 -1.19194545e-01 -1.73787385e-01  4.48431581e-01  9.70144197e-03
  2.31546521e+00 -2.38885954e-01 -1.53242159e+00 -1.40406060e+00
  9.90763724e-01 -1.87731400e-01  6.58007920e-01  2.55334750e-03
 -3.02286053e+00 -2.13761139e+00  1.51428676e+00  7.92610943e-01
  3.29436481e-01 -9.55612004e-01  5.84591106e-02  2.37663910e-02
  8.68091173e-03  8.39686021e-02 -4.39529061e-01 -5.73802665e-02
 -5.07507682e-01  9.87449810e-02 -1.66000292e-01  6.61307752e-01
 -6.10365927e-01  2.60176897e-01  1.43816757e+00  7.66645730e-01
 -4.64147050e-03 -6.70524478e-01 -9.20255333e-02 -2.39190217e-02
  6.44118607e-01  1.32254675e-01  5.08459866e-01  1.58455759e-01
 -2.11163187e+00  2.30418861e-01  6.05224907e-01  4.23264593e-01
  2.55549908e-01  7.95720160e-01 -1.84087276e-01  5.35931408e-01
  1.36089766e+00  2.75502014e+00 -2.34805748e-01  5.77658474e-01
  1.82272005e+00 -6.06343113e-02 -4.27649841e-02 -6.70750672e-03
 -3.16865474e-01 -2.77975768e-01 -5.01955524e-02 -2.44606882e-01]
```

Библиотека **Numba** предустановлена по умолчанию в среде **Google Colaboration.** Пример программы с использованием **Numba** приведен ниже**.**

In [0]:

```
import numpy as np
from numba import vectorize

@vectorize(['float32(float32, float32)'], target='cuda')
def Add(a, b):
  return a + b

# Initialize arrays
N = 100000
A = np.ones(N, dtype=np.float32)
B = np.ones(A.shape, dtype=A.dtype)
```

```
C = np.empty_like(A, dtype=A.dtype)

# Add arrays on GPU
C = Add(A, B)
```

Задание

Реализовать алгоритм фильтрации изображения с использованием библиотек **Numba** и **PyCuda.** Сравнить эффективность реализации**.** Базовый алгоритм реализации фильтра Лапласа без использования **GPU** приведен ниже**:**

In [1]:

```
from PIL import Image
from PIL import ImageFilter
import requests
import matplotlib.pyplot as plt

img_url = 'https://www.humanesociety.org/sites/default/files/styles/768x326/public/2018/08/kitten-440379.jpg'

img = Image.open(requests.get(img_url, stream=True).raw)

plt.figure()
plt.imshow(img)
plt.axis('off')

#Y = .2126 * R^gamma + .7152 * G^gamma + .0722 * B^gamma
img_gs = img.convert("L")

img_px = img_gs.load()

Laplacian = ImageFilter.Kernel((3,3), (0,-1,0,-1,4,-1,0,-1,0), scale=0.1, offset=1)
img_la = img_gs.filter(Laplacian)
plt.figure()
plt.imshow(img_la)
plt.axis('off')
```

Out[1]:

```
(-0.5, 767.5, 325.5, -0.5)
```