```
!nvidia-smi
```

```
Mon May  6 02:53:38 2024
+---------------------------------------------------------------------------------------+
| NVIDIA-SMI 535.104.05             Driver Version: 535.104.05   CUDA Versi
|-----------------------------------------+----------------------+----------
| GPU  Name                 Persistence-M | Bus-Id        Disp.A | Volatile
| Fan  Temp   Perf          Pwr:Usage/Cap |         Memory-Usage | GPU-Util
|                                         |                      |
|=========================================+======================+=========
|   0  Tesla T4                       Off | 00000000:00:04.0 Off |
| N/A   38C    P8              9W /  70W |      3MiB / 15360MiB |      0%
|                                         |                      |
+-----------------------------------------+----------------------+----------

+---------------------------------------------------------------------------------------+
| Processes:
|  GPU   GI   CI        PID   Type   Process name
|        ID   ID
|=======================================================================================
|  No running processes found
+---------------------------------------------------------------------------------------+
```

```
!nvcc --version
```

```
nvcc: NVIDIA (R) Cuda compiler driver
Copyright (c) 2005-2023 NVIDIA Corporation
Built on Tue_Aug_15_22:02:13_PDT_2023
Cuda compilation tools, release 12.2, V12.2.140
Build cuda_12.2.r12.2/compiler.33191640_0
```

```
!pip install nvcc4jupyter
```

```
Collecting nvcc4jupyter
  Downloading nvcc4jupyter-1.2.1-py3-none-any.whl (10 kB)
Installing collected packages: nvcc4jupyter
Successfully installed nvcc4jupyter-1.2.1
```

```
%load_ext nvcc4jupyter
```

```
Detected platform "Colab". Running its setup...
Source files will be saved in "/tmp/tmpyqclptyg".
```

```
%%cuda
#include <stdio.h>
__global__ void HelloKernel() {
   printf("Hello world!");
}
int main() {
   HelloKernel<<<1,1>>>();
   cudaDeviceSynchronize();
   return 0;
}
```

```
 Hello world!
```

```
%%cuda
#include <stdio.h>
#include <stdlib.h>

__global__ void CalcPiKernel(const int n, double *result) {
    int idx = blockIdx.x * blockDim.x + threadIdx.x;
    // printf("Block %d, Thread %d\n", blockIdx.x, threadIdx.x); // Вывод номер
    if (idx < n) {
        const double xi = (idx + 0.5) * (1.0 / n);
        result[idx] = 4.0 / (1.0 + xi * xi);
    }
}
double CalcPi(const int n) {
    double *result;
    double *d_result;
    double pi = 0;
    const double coef = 1.0 / n;

    // Выделение памяти на хосте и устройстве
    result = (double*)malloc(n * sizeof(double));
    cudaMalloc((void**)&d_result, n * sizeof(double));

    // Запуск ядра
    int blockSize = 32; // 256;
    int numBlocks = (n + blockSize - 1) / blockSize; // количесто блоков зависи
    CalcPiKernel<<<numBlocks, blockSize>>>(n, d_result);

    // Копирование результатов обратно на хост
    cudaMemcpy(result, d_result, n * sizeof(double), cudaMemcpyDeviceToHost);

    // Суммирование результатов
    for (int i = 0; i < n; ++i) {
        pi += result[i];
    }
```

```
    // Освобождение памяти на устройстве и хосте
    cudaFree(d_result);
    free(result);

    return pi * coef;
}

int main() {
    int n = 100; // Пример количества итераций для расчета Pi
    double pi = CalcPi(n);
    printf("Approximated value of Pi: %.40lf\n", pi);
    return 0;
}
```

```
 Approximated value of Pi: 3.1416009869231253937016390409553423523903
```

```
%%cuda
#include <stdio.h>

__global__ void matrixMultiplication(double *A, double *B, double *C, int n) {
    int row = blockIdx.x * blockDim.x + threadIdx.x;
    int col = blockIdx.y * blockDim.y + threadIdx.y;
    if (row < n && col < n) {
        double sum = 0.0;
        for (int i = 0; i < n; ++i) {
            sum += A[row * n + i] * B[i * n + col];
        }
        C[row * n + col] = sum;
    }
}

int main() {
    int n = 2; // Размер матриц

    int total_elements = n * n;
    double *A_host = (double *)malloc(total_elements * sizeof(double));
    double *B_host = (double *)malloc(total_elements * sizeof(double));
    double *C_host = (double *)malloc(total_elements * sizeof(double));

    double A[] = {1, 3, 4, 8};
    double B[] = {5, 4, 3, 0};

    cudaMemcpy(A_host, A, total_elements * sizeof(double), cudaMemcpyHostToHost
    cudaMemcpy(B_host, B, total_elements * sizeof(double), cudaMemcpyHostToHost

    double *d_A, *d_B, *d_C;
```

```
    cudaMalloc(&d_A, total_elements * sizeof(double));
    cudaMalloc(&d_B, total_elements * sizeof(double));
    cudaMalloc(&d_C, total_elements * sizeof(double));

    cudaMemcpy(d_A, A_host, total_elements * sizeof(double), cudaMemcpyHostToDe
    cudaMemcpy(d_B, B_host, total_elements * sizeof(double), cudaMemcpyHostToDe

    dim3 blockSize(2, 2); // Количество потоков в блоке
    dim3 numBlocks((n + blockSize.x - 1) / blockSize.x, (n + blockSize.y - 1) /

    matrixMultiplication<<<numBlocks, blockSize>>>(d_A, d_B, d_C, n);

    cudaMemcpy(C_host, d_C, total_elements * sizeof(double), cudaMemcpyDeviceTo

    printf("Result:\n");
    for (int i = 0; i < total_elements; ++i) {
        printf("%.2f ", C_host[i]);
        if ((i + 1) % n == 0) {
            printf("\n");
        }
    }

    free(A_host);
    free(B_host);
    free(C_host);
    cudaFree(d_A);
    cudaFree(d_B);
    cudaFree(d_C);

    return 0;
}
```

```
Result:
14.00 4.00
44.00 16.00
```

```
%%cuda
#include <iostream>
#include <openacc.h>

int main() {
    const int n = 1000000;
    double x, pi, sum = 0.0;

    pi = 0.0;

    #pragma acc parallel loop reduction(+:sum)
    for (int i = 0; i < n; ++i) {
        x = (i + 0.5) / n;
        sum += 4.0 / (1.0 + x * x);
    }

    pi = sum / n;
    printf("Approximated value of Pi: %.40lf\n", pi);

    return 0;
}
```

```
 Approximated value of Pi: 3.1415926535897642501993232144741341471672
```