```
!nvcc --version
```

```
nvcc: NVIDIA (R) Cuda compiler driver
Copyright (c) 2005-2022 NVIDIA Corporation
Built on Wed_Sep_21_10:33:58_PDT_2022
Cuda compilation tools, release 11.8, V11.8.89
Build cuda_11.8.r11.8/compiler.31833905_0
```

```
!pip install git+https://github.com/andreinechaev/nvcc4jupyter.git
%load_ext nvcc_plugin
```

```
Collecting git+https://github.com/andreinechaev/nvcc4jupyter.git
  Cloning https://github.com/andreinechaev/nvcc4jupyter.git to /tmp/pip-req-build-xfsbi6z1
  Running command git clone --filter=blob:none --quiet https://github.com/andreinechaev/nvcc4jupyter.git /tmp/pip-req-build-xfsbi6z1
  Resolved https://github.com/andreinechaev/nvcc4jupyter.git to commit 0a71d56e5dce3ff1f0dd2c47c29367629262f527
  Preparing metadata (setup.py) ... done
Building wheels for collected packages: NVCCPlugin
  Building wheel for NVCCPlugin (setup.py) ... done
  Created wheel for NVCCPlugin: filename=NVCCPlugin-0.0.2-py3-none-any.whl size=4293 sha256=1713f5a051d439cf53d1d4a125c31d3f4ec4db56
  Stored in directory: /tmp/pip-ephem-wheel-cache-9aei5w7p/wheels/a8/b9/18/23f8ef71ceb0f63297dd1903aedd067e6243a68ea756d6feea
Successfully built NVCCPlugin
Installing collected packages: NVCCPlugin
Successfully installed NVCCPlugin-0.0.2
created output directory at /content/src
Out bin /content/result.out
```

```
%%cuda --name testGoogleColab.cu
```

```
'File written in /content/src/testGoogleColab.cu'
```

```
!pip install pycuda
```

```
Collecting pycuda
  Downloading pycuda-2023.1.tar.gz (1.7 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.7/1.7 MB 10.0 MB/s eta 0:00:00
  Installing build dependencies ... done
  Getting requirements to build wheel ... done
  Preparing metadata (pyproject.toml) ... done
Collecting pytools>=2011.2 (from pycuda)
  Downloading pytools-2023.1.1-py2.py3-none-any.whl (70 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 70.6/70.6 kB 8.8 MB/s eta 0:00:00
Requirement already satisfied: appdirs>=1.4.0 in /usr/local/lib/python3.10/dist-packages (from pycuda) (1.4.4)
Collecting mako (from pycuda)
  Downloading Mako-1.3.0-py3-none-any.whl (78 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 78.6/78.6 kB 11.9 MB/s eta 0:00:00
Requirement already satisfied: platformdirs>=2.2.0 in /usr/local/lib/python3.10/dist-packages (from pytools>=2011.2->pycuda) (4.0.0
Requirement already satisfied: typing-extensions>=4.0 in /usr/local/lib/python3.10/dist-packages (from pytools>=2011.2->pycuda) (4.5
Requirement already satisfied: MarkupSafe>=0.9.2 in /usr/local/lib/python3.10/dist-packages (from mako->pycuda) (2.1.3)
Building wheels for collected packages: pycuda
  Building wheel for pycuda (pyproject.toml) ... done
  Created wheel for pycuda: filename=pycuda-2023.1-cp310-cp310-linux_x86_64.whl size=661263 sha256=6bd266202699b6f72e620079c068e721a
  Stored in directory: /root/.cache/pip/wheels/46/65/06/b997165edd2fd9690c3497ca54ea4485b571d7bd959c21c6c4
Successfully built pycuda
Installing collected packages: pytools, mako, pycuda
Successfully installed mako-1.3.0 pycuda-2023.1 pytools-2023.1.1
```

```
%%cu
#include <stdio.h>
#include <stdlib.h>
#include <sys/time.h>
#define vertices 2500
#define THREADS_PER_BLOCK 8
__global__ void floydWarshallGPU(int *dist,int k){
    int i = (blockIdx.x*blockDim.x)+threadIdx.x;
    int j = (blockIdx.y*blockDim.y)+threadIdx.y;
    if(i<vertices&&j<vertices) {
        int idx=i*vertices+j;
        int idx1=k*vertices+j;
        int idx2=i*vertices+k;
        __syncthreads();
        if(dist[idx]>dist[idx1]+dist[idx2]) {
            dist[idx]=dist[idx1]+dist[idx2];
        }
        __syncthreads();
    }
}
void floydWarshall(int adj_matrix[vertices][vertices]){
    int *distances;
    int *device_distances;
    int *ans;
    size_t bytes=vertices*vertices*sizeof(int);
    distances=(int *)malloc(bytes);
    ans=(int*)malloc(bytes);
    cudaMalloc(&device_distances,bytes);
    for(int i=0;i<vertices;i++){
        for(int j=0;j<vertices;j++){
            distances[i*vertices+j]=adj_matrix[i][j];
        }
    }
    cudaMemcpy(device_distances, distances,bytes,cudaMemcpyHostToDevice);
    dim3 block((vertices/THREADS_PER_BLOCK)+1,(vertices/THREADS_PER_BLOCK)+1,1);
    dim3 threadsPerBlock(THREADS_PER_BLOCK,THREADS_PER_BLOCK,1);
    for(int k=0;k<vertices;k++){
        floydWarshallGPU<<<block,threadsPerBlock>>>(device_distances,k);
    }
    cudaDeviceSynchronize();
    cudaMemcpy(ans,device_distances,bytes,cudaMemcpyDeviceToHost);
}
int main(int argc,char** argv){
    struct timeval TimeValue_Start;
    struct timezone TimeZone_Start;
    struct timeval TimeValue_Final;
    struct timezone TimeZone_Final;
    long time_start,time_end;
    double time_overhead;
    static int adj_matrix[vertices][vertices];
    for(int i=0;i<vertices;i++){
        for(int j=0;j<vertices;j++){
            if(i==j)
                adj_matrix[i][j]=0;
            else
                adj_matrix[i][j]=rand()%10000;
        }
    }
    gettimeofday(&TimeValue_Start,&TimeZone_Start);
    floydWarshall(adj_matrix);
    gettimeofday(&TimeValue_Final,&TimeZone_Final);
    time_start=TimeValue_Start.tv_sec*1000000+TimeValue_Start.tv_usec;
    time_end=TimeValue_Final.tv_sec*1000000+TimeValue_Final.tv_usec;
    time_overhead=(time_end-time_start)/1000000.0;
    printf("\n\n\t\t Time in Seconds (T) :%lf\n",time_overhead);
    return 0;
}
```

```
                    Time in Seconds (T) :0.021309
```

```
%%cu
#include <stdio.h>
#include <stdlib.h>
#include <sys/time.h>
#define vertices 1250
#define THREADS_PER_BLOCK 8
__global__ void floydWarshallGPU(int *dist,int k){
    int i = (blockIdx.x*blockDim.x)+threadIdx.x;
    int j = (blockIdx.y*blockDim.y)+threadIdx.y;
    if(i<vertices&&j<vertices) {
```

```
    if(i<vertices&&j<vertices) {
        int idx=i*vertices+j;
        int idx1=k*vertices+j;
        int idx2=i*vertices+k;
        __syncthreads();
        if(dist[idx]>dist[idx1]+dist[idx2]) {
            dist[idx]=dist[idx1]+dist[idx2];
        }
        __syncthreads();
    }
}
void floydWarshall(int adj_matrix[vertices][vertices]){
    int *distances;
    int *device_distances;
    int *ans;
    size_t bytes=vertices*vertices*sizeof(int);
    distances=(int *)malloc(bytes);
    ans=(int*)malloc(bytes);
    cudaMalloc(&device_distances,bytes);
    for(int i=0;i<vertices;i++){
        for(int j=0;j<vertices;j++){
            distances[i*vertices+j]=adj_matrix[i][j];
        }
    }
    cudaMemcpy(device_distances, distances,bytes,cudaMemcpyHostToDevice);
    dim3 block((vertices/THREADS_PER_BLOCK)+1,(vertices/THREADS_PER_BLOCK)+1,1);
    dim3 threadsPerBlock(THREADS_PER_BLOCK,THREADS_PER_BLOCK,1);
    for(int k=0;k<vertices;k++){
        floydWarshallGPU<<<block,threadsPerBlock>>>(device_distances,k);
    }
    cudaDeviceSynchronize();
    cudaMemcpy(ans,device_distances,bytes,cudaMemcpyDeviceToHost);
}
int main(int argc,char** argv){
    struct timeval TimeValue_Start;
    struct timezone TimeZone_Start;
    struct timeval TimeValue_Final;
    struct timezone TimeZone_Final;
    long time_start,time_end;
    double time_overhead;
    static int adj_matrix[vertices][vertices];
    for(int i=0;i<vertices;i++){
        for(int j=0;j<vertices;j++){
            if(i==j)
                adj_matrix[i][j]=0;
            else
                adj_matrix[i][j]=rand()%10000;
        }
    }P
    gettimeofday(&TimeValue_Start,&TimeZone_Start);
    floydWarshall(adj_matrix);
    gettimeofday(&TimeValue_Final,&TimeZone_Final);
    time_start=TimeValue_Start.tv_sec*1000000+TimeValue_Start.tv_usec;
    time_end=TimeValue_Final.tv_sec*1000000+TimeValue_Final.tv_usec;
    time_overhead=(time_end-time_start)/1000000.0;
    printf("\n\n\t\t Time in Seconds (T) :%lf\n",time_overhead);
    return 0;
}
```

```
                        Time in Seconds (T) :0.004328
```