

Name :Abdulrahman Almyman

ID:441170135

Rank Sort is an algorithm for sorting a vector based on the rank of each element in the vector.

// Rank Sort Idea: <https://mobylab.docs.crescdi.pub.ro/en/docs/parallelAndDistributed/laboratory9/rankSort/>

//How To Run CUDA C or C++ on Google Colab or Azure Notebook : <https://harshityadav95.medium.com/how-to-run-cuda-c-or-c-on-google-colab-or-azure-notebook-ea75a23a5962>

// LINK Colab : https://colab.research.google.com/drive/1hev7e8QseO9B__LONuAAbRduiMefUN32?usp=sharing

```
In [ ]: !apt-get --purge remove cuda nvidia* libnvidia-*
!dpkg -l | grep cuda- | awk '{print $2}' | xargs -n1 dpkg --purge
!apt-get remove cuda-*
!apt autoremove
!apt-get update
```

```
In [ ]: !wget https://developer.nvidia.com/compute/cuda/9.2/Prod/local_installers/cuda-repo-ubuntu1604-9-2-local_9.2.88-1_amd64 -O cuda-repo-ubuntu1604-9-2-local_9.2.88-1_amd64.deb
!dpkg -i cuda-repo-ubuntu1604-9-2-local_9.2.88-1_amd64.deb
!apt-key add /var/cuda-repo-9-2-local/7fa2af80.pub
!apt-get update
```

```
In [ ]: !apt-get install cuda-9.2
```

```
In [ ]: !apt-get install -y nvidia-cuda-toolkit
```

```
In [39]: !nvcc --version

nvcc: NVIDIA (R) Cuda compiler driver
Copyright (c) 2005-2021 NVIDIA Corporation
Built on Thu_Nov_18_09:45:30_PST_2021
Cuda compilation tools, release 11.5, V11.5.119
Build cuda_11.5.r11.5/compiler.30672275_0
```

```
In [ ]: !pip install nvcc4jupyter==1.0.0
```

```
In [ ]: !pip install nvcc4jupyter
```

```
In [42]: %load_ext nvcc4jupyter
```

The nvcc4jupyter extension is already loaded. To reload it, use:
%reload_ext nvcc4jupyter

```
In [48]: %%cuda

#include "cuda_runtime.h"
#include "device_launch_parameters.h"
#include <stdio.h>
#include <stdio.h>
#include <cuda.h>
#include <stdlib.h>

#define N 15 // Array size for any number
int step = 0;
__global__ void rankSort(int* d_input, int* d_output, int n) {
    //find the corresponding array index of the thread
    int index = blockIdx.x * blockDim.x + threadIdx.x;
    //check if thread is inside the bounds of array
    if (index >= n)
        return;
    //initialize rank to 0
    int rank = 0;
    //store the value of array cell
    int cell = d_input[index];
    //increment rank for each element less than or equal to d_input[index], and with index i < index
    for (int i = 0; i < index; i++) {
        if (d_input[i] <= cell)
            rank++;
    }
    //increment rank for each element less than d_input[index], and with index i > index
    for (int i = index + 1; i < n; i++) {
        if (d_input[i] < cell)
            rank++;
    }
    //store d_input[index] in the correct index in the output array
    d_output[rank] = cell;
}

void printNumbers(int* array, int n) {
    for (int i = 0; i < n; i++) {
        printf("%d ", array[i]);
    }
}

void generateRandom(int* array, int n, int range) {
    for (int i = 0; i < n; i++) {
        array[i] = rand() % range;
    }
}

int main() {
    int h_input[N];
    int h_output[N];
    int* d_input, * d_output;

    generateRandom(h_input, N, N);
    printf("The Original Sort : ");
    printNumbers(h_input, N);
    printf("\n");

    cudaMalloc((void*)&d_input, N * sizeof(int));
    cudaMalloc((void*)&d_output, N * sizeof(int));

    cudaMemcpy(d_input, h_input, N * sizeof(int), cudaMemcpyHostToDevice);
    //
    int n_Blocks = (N + 1023) / 1024;
    //
    int n_Threads = 1024;

    rankSort << <n_Blocks, n_Threads >> > (d_input, d_output, N);
    //Because calling the non blocking function, each process must wait for the d_output result to be correct
    cudaDeviceSynchronize();
    //
    cudaMemcpy(h_output, d_output, N * sizeof(int), cudaMemcpyDeviceToHost);
    printf("the sort after using rankSort: ");
    printNumbers(h_output, N);

    printf("\nDone!!\n");
    cudaFree(d_input);
    cudaFree(d_output);
    //printf("%d",rand());

    return 0;
}
```

The Original Sort : 13 1 12 10 8 10 1 12 9 1 2 7 5 4 8
the sort after using rankSort: 1 1 1 2 4 5 7 8 8 9 10 10 12 12 13
Done!!