## Refreshing the Cloud Instance of CUDA On Server

```
!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
```

## Installing CUDA Version 9

```
!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
!apt-get install cuda-9.2
```

```
!nvcc --version
```

```
!pip install git+git://github.com/andreinechaev/nvcc4jupyter.git
```

```
%load_ext nvcc_plugin
```

### Q1. Write and execute a program in CUDA to add two vectors of length N to meet the following requirements using 3 different kernels

a) block size as N

b) N threads within a block

c) Keep the number of threads per block as 256 (constant) and vary the number of blocks to handle N elements.

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

__global__ void vecAddKernel(float *dev_arr1, float *dev_arr2, float *dev_arr3, int N)
{
    int threadId = threadIdx.x + blockDim.x * blockIdx.x;

    if (threadId < N)
        dev_arr3[threadId] = dev_arr1[threadId] + dev_arr2[threadId];
}

int main()
{
    float *dev_arr1, *dev_arr2, *dev_arr3;
    float host_arr1[1024], host_arr2[1024], host_arr3[1024];

    int N = 1024;
    int arr_size = N * sizeof(float);

    // Initializing the arrays with 1024 random integers
    for (int f = 0; f < 1024; f++) host_arr1[f] = (rand() % 20) + 50;
    for (int f = 0; f < 1024; f++) host_arr2[f] = (rand() % 20) + 50;

    printf("\n Initial Array 1 of 1024 elements is: \n");
    for (int f = 0; f < 1024; f++) printf("%.0f ", host_arr1[f]);
    printf("\n Initial Array 2 of 1024 elements is: \n");
    for (int f = 0; f < 1024; f++) printf("%.0f ", host_arr2[f]);

    // Allocate device memory for A,B and C
    cudaMalloc((void **)&dev_arr1, arr_size);
    cudaMalloc((void **)&dev_arr2, arr_size);
    cudaMalloc((void **)&dev_arr3, arr_size);

    // Copy A and B to device memory
    cudaMemcpy(dev_arr1, host_arr1, arr_size, cudaMemcpyHostToDevice);
    cudaMemcpy(dev_arr2, host_arr2, arr_size, cudaMemcpyHostToDevice);

    // Kernel launch code – to have the device
    // to perform the actual vector addition
    vecAddKernel<<<ceil(N/1.0), 1>>>(dev_arr1, dev_arr2, dev_arr3, N);

    // Copy C from the device memory
    // Free the device vectorsz
    cudaMemcpy(host_arr3, dev_arr3, arr_size, cudaMemcpyDeviceToHost);
    cudaFree(dev_arr1);
    cudaFree(dev_arr2);
    cudaFree(dev_arr3);


    printf("\n Result Array 3 of 1024 elements after addition is: \n");
    for (int f = 0; f < 1024; f++)
        printf("%.0f ", host_arr3[f]);
    return 0;
}
```

```
 Initial Array 1 of 1024 elements is:
53 56 67 65 63 65 56 62 59 51 52 57 60 69 53 56 50 56 62 66 61 58 57 59 52 60 52 53 57 65 59 52 52 68 59 57 63 66 61 52 59 63 51 69 54 67 68 54 65 60 63 56 61 50 66 63 52 (
 Initial Array 2 of 1024 elements is:
61 54 60 53 61 62 67 67 69 58 61 50 57 66 60 51 60 62 55 61 59 67 51 53 61 62 50 56 68 57 54 59 62 64 54 65 69 51 62 68 59 66 60 66 54 50 59 64 55 64 56 56 53 69 59 65 62 !
 Result Array 3 of 1024 elements after addition is:
114 110 127 118 124 127 123 129 128 109 113 107 117 135 113 107 110 118 117 127 120 125 108 112 113 122 102 109 125 122 113 111 114 132 113 122 132 117 123 120 118 129 111
```

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

```

__global__ void vecAddKernel(float *dev_arr1, float *dev_arr2, float *dev_arr3, int N)
{
    int threadId = threadIdx.x + blockDim.x * blockIdx.x;

    if (threadId < N)
        dev_arr3[threadId] = dev_arr1[threadId] + dev_arr2[threadId];
}

int main()
{
    float *dev_arr1, *dev_arr2, *dev_arr3;
    float host_arr1[1024], host_arr2[1024], host_arr3[1024];

    int N = 1024;
    int arr_size = N * sizeof(float);

    // Initializing the arrays with 1024 random integers
    for (int f = 0; f < 1024; f++) host_arr1[f] = (rand() % 20) + 50;
    for (int f = 0; f < 1024; f++) host_arr2[f] = (rand() % 20) + 50;

    printf("\n Initial Array 1 of 1024 elements is: \n");
    for (int f = 0; f < 1024; f++) printf("%.0f ", host_arr1[f]);
    printf("\n Initial Array 2 of 1024 elements is: \n");
    for (int f = 0; f < 1024; f++) printf("%.0f ", host_arr2[f]);

    // Allocate device memory for A,B and C
    cudaMalloc((void **)&dev_arr1, arr_size);
    cudaMalloc((void **)&dev_arr2, arr_size);
    cudaMalloc((void **)&dev_arr3, arr_size);

    // Copy A and B to device memory
    cudaMemcpy(dev_arr1, host_arr1, arr_size, cudaMemcpyHostToDevice);
    cudaMemcpy(dev_arr2, host_arr2, arr_size, cudaMemcpyHostToDevice);

    // Kernel launch code – to have the device
    // to perform the actual vector addition
    vecAddKernel<<<1,ceil(N/1.0)>>>(dev_arr1, dev_arr2, dev_arr3, N);

    // Copy C from the device memory
    // Free the device vectorsz
    cudaMemcpy(host_arr3, dev_arr3, arr_size, cudaMemcpyDeviceToHost);
    cudaFree(dev_arr1);
    cudaFree(dev_arr2);
    cudaFree(dev_arr3);


    printf("\n Result Array 3 of 1024 elements after addition is: \n");
    for (int f = 0; f < 1024; f++)
        printf("%.0f ", host_arr3[f]);
    return 0;
}
```

```
 Initial Array 1 of 1024 elements is:
53 56 67 65 63 65 56 62 59 51 52 57 60 69 53 56 50 56 62 66 61 58 57 59 52 60 52 53 57 65 59 52 52 68 59 57 63 66 61 52 59 63 51 69 54 67 68 54 65 60 63 56 61 50 66 63 52 (
 Initial Array 2 of 1024 elements is:
61 54 60 53 61 62 67 67 69 58 61 50 57 66 60 51 60 62 55 61 59 67 51 53 61 62 50 56 68 57 54 59 62 64 54 65 69 51 62 68 59 66 60 66 54 50 59 64 55 64 56 56 53 69 59 65 62 !
 Result Array 3 of 1024 elements after addition is:
114 110 127 118 124 127 123 129 128 109 113 107 117 135 113 107 110 118 117 127 120 125 108 112 113 122 102 109 125 122 113 111 114 132 113 122 132 117 123 120 118 129 111
```

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

__global__ void vecAddKernel(float *dev_arr1, float *dev_arr2, float *dev_arr3, int N)
{
    int threadId = threadIdx.x + blockDim.x * blockIdx.x;

    if (threadId < N)
        dev_arr3[threadId] = dev_arr1[threadId] + dev_arr2[threadId];
}

int main()
{
    float *dev_arr1, *dev_arr2, *dev_arr3;
    float host_arr1[1024], host_arr2[1024], host_arr3[1024];

    int N = 1024;
    int arr_size = N * sizeof(float);

    // Initializing the arrays with 1024 random integers
    for (int f = 0; f < 1024; f++) host_arr1[f] = (rand() % 20) + 50;
    for (int f = 0; f < 1024; f++) host_arr2[f] = (rand() % 20) + 50;

    printf("\n Initial Array 1 of 1024 elements is: \n");
    for (int f = 0; f < 1024; f++) printf("%.0f ", host_arr1[f]);
    printf("\n Initial Array 2 of 1024 elements is: \n");
    for (int f = 0; f < 1024; f++) printf("%.0f ", host_arr2[f]);

    // Allocate device memory for A,B and C
    cudaMalloc((void **)&dev_arr1, arr_size);
    cudaMalloc((void **)&dev_arr2, arr_size);
    cudaMalloc((void **)&dev_arr3, arr_size);

    // Copy A and B to device memory
    cudaMemcpy(dev_arr1, host_arr1, arr_size, cudaMemcpyHostToDevice);
    cudaMemcpy(dev_arr2, host_arr2, arr_size, cudaMemcpyHostToDevice);

    // Kernel launch code – to have the device
    // to perform the actual vector addition
    vecAddKernel<<<ceil(N/256.0), 256>>>(dev_arr1, dev_arr2, dev_arr3, N);

    // Copy C from the device memory
    // Free the device vectorsz
    cudaMemcpy(host_arr3, dev_arr3, arr_size, cudaMemcpyDeviceToHost);
    cudaFree(dev_arr1);
    cudaFree(dev_arr2);
    cudaFree(dev_arr3);


    printf("\n Result Array 3 of 1024 elements after addition is: \n");
    for (int f = 0; f < 1024; f++)
```

```
        printf("%.0f ", host_arr3[f]);
    return 0;
}
```

```
 Initial Array 1 of 1024 elements is:
53 56 67 65 63 65 56 62 59 51 52 57 60 69 53 56 50 56 62 66 61 58 57 59 52 60 52 53 57 65 59 52 52 68 59 57 63 66 61 52 59 63 51 69 54 67 68 54 65 60 63 56 61 50 66 63 52 (
 Initial Array 2 of 1024 elements is:
61 54 60 53 61 62 67 67 69 58 61 50 57 66 60 51 60 62 55 61 59 67 51 53 61 62 50 56 68 57 54 59 62 64 54 65 69 51 62 68 59 66 60 66 54 50 59 64 55 64 56 56 53 69 59 65 62 !
 Result Array 3 of 1024 elements after addition is:
114 110 127 118 124 127 123 129 128 109 113 107 117 135 113 107 110 118 117 127 120 125 108 112 113 122 102 109 125 122 113 111 114 132 113 122 132 117 123 120 118 129 111
```

## Q2. Write and execute a CUDA program to read an array of N integer values. Sort the array in parallel using parallel selection sort and store the result in another array.

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

__global__ void vecSelectionSortKernel(float *dev_arr1, float *dev_arr2, int N)
{
    int threadId = threadIdx.x + blockDim.x * blockIdx.x;

    int data = dev_arr1[threadId];
    int pos = 0;
    for (int i = 0; i < N; i++)
    {
        if ((dev_arr1[i] < data) || (dev_arr1[i] == data && i < threadId))
            pos++;
    }

    dev_arr2[pos] = data;
}

int main()
{
    float *dev_arr1, *dev_arr2;
    float host_arr1[1024], host_arr2[1024];

    int N = 1024;
    int arr_size = N * sizeof(float);

    // Initializing the arrays with 1024 random integers
    for (int f = 0; f < 1024; f++) host_arr1[f] = (rand() % 49) + 50;

    printf("\n Array 0f 1024 elements before sorting is: \n");
    for (int f = 0; f < 1024; f++) printf("%.0f ", host_arr1[f]);

    // Allocate device memory for A,B and C
    cudaMalloc((void **)&dev_arr1, arr_size);
    cudaMalloc((void **)&dev_arr2, arr_size);

    // Copy A and B to device memory
    cudaMemcpy(dev_arr1, host_arr1, arr_size, cudaMemcpyHostToDevice);

    // Kernel launch code – to have the device
    // to perform the actual vector addition
    vecSelectionSortKernel<<<ceil(N / 1), 1>>>(dev_arr1, dev_arr2, N);

    // Copy C from the device memory
    // Free the device vectorsz
    cudaMemcpy(host_arr2, dev_arr2, arr_size, cudaMemcpyDeviceToHost);
    cudaFree(dev_arr1);
    cudaFree(dev_arr2);

    printf("\n Array 0f 1024 elements after sorting is: \n");
    for (int f = 0; f < 1024; f++) printf("%.0f ", host_arr2[f]);
    return 0;
}
```

```
 Array 0f 1024 elements before sorting is:
65 89 87 55 79 60 74 73 58 67 59 97 56 68 70 71 67 72 73 73 53 71 55 87 80 97 83 98 79 55 55 50 95 97 60 80 58 85 59 66 58 68 69 64 91 95 91 59 73 65 88 76 92 93 65 73 96 !
 Array 0f 1024 elements after sorting is:
50 50 50 50 50 50 50 50 50 50 50 50 50 50 50 50 50 50 50 51 51 51 51 51 51 51 51 51 51 51 51 51 51 51 52 52 52 52 52 52 52 52 52 52 52 52 52 52 52 52 52 52 !
```

## 3 Write a execute a CUDA program to read an integer array of size N. Sort this array using odd-even transposition sorting. Use 2 kernels.

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

__global__ void vecTranspositionSortingOddEvenKernel(float *dev_arr, int N)
{
    int threadId = threadIdx.x + blockDim.x * blockIdx.x;

    if((threadId % 2) != 0 && threadId + 1 <= N-1)
    {
        if(dev_arr[threadId] > dev_arr[threadId + 1])
        {
            int temp = dev_arr[threadId];
            dev_arr[threadId] = dev_arr[threadId + 1];
            dev_arr[threadId + 1] = temp;
        }
    }

}

__global__ void vecTranspositionSortingEvenOddKernel(float *dev_arr, int N)
{
    int threadId = threadIdx.x + blockDim.x * blockIdx.x;

        if((threadId % 2) == 0 && threadId + 1 <= N-1)
    {
        if(dev_arr[threadId] > dev_arr[threadId + 1])
```

```
        {
            int temp = dev_arr[threadId];
            dev_arr[threadId] = dev_arr[threadId + 1];
            dev_arr[threadId + 1] = temp;
        }
    }

}

int main()
{
    float *dev_arr;
    float host_arr[1024];

    int N = 1024;
    int arr_size = N * sizeof(float);

    // Initializing the arrays with 1024 random integers
    for (int f = 0; f < 1024; f++) host_arr[f] = (rand() % 49) + 50;

    printf("\n Array of 1024 elements before sorting is: \n");
    for (int f = 0; f < 1024; f++) printf("%.0f ", host_arr[f]);

    // Allocate device memory for A,B and C
    cudaMalloc((void **)&dev_arr, arr_size);

    // Copy A and B to device memory
    cudaMemcpy(dev_arr, host_arr, arr_size, cudaMemcpyHostToDevice);

    // Kernel launch code – to have the device
    // to perform the actual vector addition

    for(int i = 0 ; i < N / 2 ; i++)
    {
        vecTranspositionSortingOddEvenKernel<<<ceil(N / 256.0), 256>>>(dev_arr, N);
        vecTranspositionSortingEvenOddKernel<<<ceil(N / 256.0), 256>>>(dev_arr, N);
    }

    // Copy C from the device memory
    // Free the device vectorsz
    cudaMemcpy(host_arr, dev_arr, arr_size, cudaMemcpyDeviceToHost);
    cudaFree(dev_arr);

    printf("\n Array of 1024 elements after sorting is: \n");
    for (int f = 0; f < 1024; f++) printf("%.0f ", host_arr[f]);
    return 0;
}
```

```
 Array of 1024 elements before sorting is:
65 89 87 55 79 60 74 73 58 67 59 97 56 68 70 71 67 72 73 73 53 71 55 87 80 97 83 98 79 55 55 50 95 97 60 80 58 85 59 66 58 68 69 64 91 95 91 59 73 65 88 76 92 93 65 73 96 !
 Array of 1024 elements after sorting is:
50 50 50 50 50 50 50 50 50 50 50 50 50 50 50 50 50 50 50 50 51 51 51 51 51 51 51 51 51 51 51 51 51 51 51 51 52 52 52 52 52 52 52 52 52 52 52 52 52 52 52 52 52 52 !
```

```
1
```

✓ 0s completed at 12:52