▼ Refreshing the Cloud Instance of CUDA On Server

```
1 !apt-get --purge remove cuda nvidia* libnvidia-*
2 !dpkg -l | grep cuda- | awk '{print $2}' | xargs -n1 dpkg --purge
3 !apt-get remove cuda-*
4 !apt autoremove
5 !apt-get update
```

▼ Installing CUDA Version 9

```
1 !wget https://developer.nvidia.com/compute/cuda/9.2/Prod/local_installers/cuda-repo-ubuntu1604-9-2-local_9.2.88-1_amd64 -0 cuda-repo-ubuntu1604-9-2-local_9.2.88-1_amd64.deb
2 !dpkg -i cuda-repo-ubuntu1604-9-2-local_9.2.88-1_amd64.deb
3 !apt-key add /var/cuda-repo-9-2-local/7fa2af80.pub
4 !apt-get update
5 !apt-get install cuda-9.2

1 !nvcc --version

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

- 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.

```
1 %%cu
2 #include <stdio.h>
3 #include <stdlib.h>
    _global__ void vecAddKernel(float *dev_arr1, float *dev_arr2, float *dev_arr3, int N)
 6 {
      int threadId = threadIdx.x + blockDim.x * blockIdx.x;
      if (threadId < N)</pre>
          dev_arr3[threadId] = dev_arr1[threadId] + dev_arr2[threadId];
11 }
12
13 int main()
14 {
      float *dev_arr1, *dev_arr2, *dev_arr3;
       float host_arr1[1024], host_arr2[1024], host_arr3[1024];
17
      int N = 1024;
      int arr size = N * sizeof(float);
20
21
      // Initializing the arrays with 1024 random integers
       for (int f = 0; f < 1024; f++) host_arr1[f] = (rand() % 20) + 50;
22
       for (int f = 0; f < 1024; f++) host arr2[f] = (rand() % 20) + 50;
23
24
25
      printf("\n Initial Array 1 of 1024 elements is: \n");
       for (int f = 0; f < 1024; f++) printf("%.0f ", host_arr1[f]);</pre>
26
       printf("\n Initial Array 2 of 1024 elements is: \n");
27
       for (int f = 0; f < 1024; f++) printf("%.0f", host arr2[f]);
28
29
30
      // Allocate device memory for A,B and C
31
       cudaMalloc((void **)&dev_arr1, arr_size);
32
      cudaMalloc((void **)&dev_arr2, arr_size);
33
      cudaMalloc((void **)&dev_arr3, arr_size);
34
35
      // Copy A and B to device memory
      cudaMemcpy(dev arr1, host arr1, arr size, cudaMemcpyHostToDevice);
      cudaMemcpy(dev arr2, host arr2, arr size, cudaMemcpyHostToDevice);
37
38
39
      // Kernel launch code - to have the device
40
      // to perform the actual vector addition
41
      vecAddKernel<<<ceil(N/1.0), 1>>>(dev_arr1, dev_arr2, dev_arr3, N);
42
43
      // Copy C from the device memory
      // Free the device vectorsz
      cudaMemcpy(host_arr3, dev_arr3, arr_size, cudaMemcpyDeviceToHost);
      cudaFree(dev arr1);
47
      cudaFree(dev arr2);
      cudaFree(dev_arr3);
50
51
      printf("\n Result Array 3 of 1024 elements after addition is: \n");
52
       for (int f = 0; f < 1024; f++)
53
           printf("%.0f ", host_arr3[f]);
54
       return 0;
55 }
```

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 6

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

```
1 %cu
2 #include <stdio.h>
3 #include <stdlib.h>
```

```
_global__ void vecAddKernel(float *dev_arr1, float *dev_arr2, float *dev_arr3, int N)
6 {
      int threadId = threadIdx.x + blockDim.x * blockIdx.x;
      if (threadId < N)</pre>
10
           dev arr3[threadId] = dev arr1[threadId] + dev arr2[threadId];
11 }
12
13 int main()
14 {
       float *dev_arr1, *dev_arr2, *dev_arr3;
       float host_arr1[1024], host_arr2[1024], host_arr3[1024];
16
17
18
      int N = 1024;
19
      int arr_size = N * sizeof(float);
20
21
      // Initializing the arrays with 1024 random integers
       for (int f = 0; f < 1024; f++) host_arr1[f] = (rand() % 20) + 50;
22
23
       for (int f = 0; f < 1024; f++) host_arr2[f] = (rand() % 20) + 50;
24
25
      printf("\n Initial Array 1 of 1024 elements is: \n");
       for (int f = 0; f < 1024; f++) printf("%.0f", host arr1[f]);
26
       printf("\n Initial Array 2 of 1024 elements is: \n");
27
       for (int f = 0; f < 1024; f++) printf("%.0f ", host_arr2[f]);</pre>
28
29
30
      // Allocate device memory for A,B and C
31
      cudaMalloc((void **)&dev_arr1, arr_size);
32
      cudaMalloc((void **)&dev_arr2, arr_size);
      cudaMalloc((void **)&dev arr3, arr size);
33
34
35
      // Copy A and B to device memory
36
      cudaMemcpy(dev_arr1, host_arr1, arr_size, cudaMemcpyHostToDevice);
37
      cudaMemcpy(dev arr2, host arr2, arr size, cudaMemcpyHostToDevice);
38
39
      // Kernel launch code — to have the device
40
      // to perform the actual vector addition
      vecAddKernel<<<1,ceil(N/1.0)>>>(dev_arr1, dev_arr2, dev_arr3, N);
41
42
43
      // Copy C from the device memory
44
      // Free the device vectorsz
45
      cudaMemcpy(host_arr3, dev_arr3, arr_size, cudaMemcpyDeviceToHost);
46
      cudaFree(dev_arr1);
47
      cudaFree(dev_arr2);
      cudaFree(dev arr3);
50
51
      printf("\n Result Array 3 of 1024 elements after addition is: \n");
52
       for (int f = 0; f < 1024; f++)
53
           printf("%.0f ", host_arr3[f]);
54
      return 0;
55 }
     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 5 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

```
1 %%cu
2 #include <stdio.h>
3 #include <stdlib.h>
5 __global__ void vecAddKernel(float *dev_arr1, float *dev_arr2, float *dev_arr3, int N)
6 {
      int threadId = threadIdx.x + blockDim.x * blockIdx.x;
      if (threadId < N)</pre>
          dev_arr3[threadId] = dev_arr1[threadId] + dev_arr2[threadId];
11 }
12
13 int main()
14 {
15
       float *dev_arr1, *dev_arr2, *dev_arr3;
       float host_arr1[1024], host_arr2[1024], host_arr3[1024];
17
      int N = 1024;
19
      int arr_size = N * sizeof(float);
20
21
      // Initializing the arrays with 1024 random integers
22
       for (int f = 0; f < 1024; f++) host arr1[f] = (rand() % 20) + 50;
23
       for (int f = 0; f < 1024; f++) host_arr2[f] = (rand() % 20) + 50;
24
25
       printf("\n Initial Array 1 of 1024 elements is: \n");
26
       for (int f = 0; f < 1024; f++) printf("%.0f ", host_arr1[f]);
27
       printf("\n Initial Array 2 of 1024 elements is: \n");
28
       for (int f = 0; f < 1024; f++) printf("%.0f ", host_arr2[f]);
29
30
      // Allocate device memory for A,B and C
31
      cudaMalloc((void **)&dev_arr1, arr_size);
32
       cudaMalloc((void **)&dev_arr2, arr_size);
33
       cudaMalloc((void **)&dev_arr3, arr_size);
34
      // Copy A and B to device memory
       cudaMemcpy(dev arr1, host arr1, arr size, cudaMemcpyHostToDevice);
36
37
      cudaMemcpy(dev_arr2, host_arr2, arr_size, cudaMemcpyHostToDevice);
38
      // Kernel launch code — to have the device
      // to perform the actual vector addition
41
      vecAddKernel<<<ceil(N/256.0), 256>>>(dev arr1, dev arr2, dev arr3, N);
42
43
      // Copy C from the device memory
      // Free the device vectorsz
      cudaMemcpy(host arr3, dev arr3, arr size, cudaMemcpyDeviceToHost);
45
      cudaFree(dev arr1);
47
       cudaFree(dev arr2);
       cudaFree(dev_arr3);
50
51
       printf("\n Result Array 3 of 1024 elements after addition is: \n");
52
       for (int f = 0; f < 1024; f++)
```

```
05/05/2021
                                                                                  Kaustav-PCAP-LAB05-Report.ipynb - Colaboratory
              printf("%.0f ", host_arr3[f]);
   54
          return 0;
   55 }
        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 5
        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.

```
1 %%cu
2 #include <stdio.h>
3 #include <stdlib.h>
   __global__ void vecSelectionSortKernel(float *dev_arr1, float *dev_arr2, int N)
6 {
      int threadId = threadIdx.x + blockDim.x * blockIdx.x;
      int data = dev_arr1[threadId];
      int pos = 0;
      for (int i = 0; i < N; i++)
11
12
13
           if ((dev_arr1[i] < data) || (dev_arr1[i] == data && i < threadId))</pre>
14
               pos++;
15
16
      dev arr2[pos] = data;
17
18 }
19
20 int main()
       float *dev_arr1, *dev arr2;
23
       float host_arr1[1024], host_arr2[1024];
24
25
      int N = 1024;
      int arr size = N * sizeof(float);
26
27
28
      // Initializing the arrays with 1024 random integers
29
       for (int f = 0; f < 1024; f++) host_arr1[f] = (rand() % 49) + 50;
30
31
      printf("\n Array Of 1024 elements before sorting is: \n");
       for (int f = 0; f < 1024; f++) printf("%.0f ", host_arr1[f]);</pre>
33
34
      // Allocate device memory for A,B and C
      cudaMalloc((void **)&dev_arr1, arr_size);
36
      cudaMalloc((void **)&dev_arr2, arr_size);
37
38
      // Copy A and B to device memory
39
      cudaMemcpy(dev_arr1, host_arr1, arr_size, cudaMemcpyHostToDevice);
41
      // Kernel launch code — to have the device
42
       // to perform the actual vector addition
       vecSelectionSortKernel<<<ceil(N / 1), 1>>>(dev_arr1, dev_arr2, N);
44
      // Copy C from the device memory
      // Free the device vectorsz
      cudaMemcpy(host_arr2, dev_arr2, arr_size, cudaMemcpyDeviceToHost);
       cudaFree(dev_arr1);
       cudaFree(dev_arr2);
50
      printf("\n Array Of 1024 elements after sorting is: \n");
       for (int f = 0; f < 1024; f++) printf("%.0f ", host_arr2[f]);
53
54 }
```

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:

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.

```
1 %%cu
2 #include <stdio.h>
3 #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)
11
           if(dev_arr[threadId] > dev_arr[threadId + 1])
12
13
               int temp = dev_arr[threadId];
14
              dev_arr[threadId] = dev_arr[threadId + 1];
15
              dev_arr[threadId + 1] = temp;
17
19 }
20
    _global__ void vecTranspositionSortingEvenOddKernel(float *dev_arr, int N)
22 {
23
      int threadId = threadIdx.x + blockDim.x * blockIdx.x;
24
25
           if((threadId % 2) == 0 \&\& threadId + 1 <= N-1)
26
           if(dev arr[threadId] > dev arr[threadId + 1])
```

```
05/05/2021
                                                                         Kaustav-PCAP-LAB05-Report.ipynb - Colaboratory
  28
                int temp = dev_arr[threadId];
                dev_arr[threadId] = dev_arr[threadId + 1];
  30
                dev_arr[threadId + 1] = temp;
  31
  32
  33
  34
  35 }
  36
  37 int main()
  38 {
  39
         float *dev_arr;
         float host_arr[1024];
  40
  41
  42
        int N = 1024;
  43
        int arr_size = N * sizeof(float);
  44
  45
        // Initializing the arrays with 1024 random integers
         for (int f = 0; f < 1024; f++) host_arr[f] = (rand() % 49) + 50;
  46
  47
        printf("\n Array of 1024 elements before sorting is: \n");
         for (int f = 0; f < 1024; f++) printf("%.0f ", host_arr[f]);</pre>
  50
  51
        // Allocate device memory for A,B and C
  52
        cudaMalloc((void **)&dev_arr, arr_size);
  54
        // Copy A and B to device memory
  55
        cudaMemcpy(dev_arr, host_arr, arr_size, cudaMemcpyHostToDevice);
  56
        // Kernel launch code - to have the device
  58
        // to perform the actual vector addition
  59
        for(int i = 0; i < N / 2; i++)
  60
  61
        {
  62
            vecTranspositionSortingOddEvenKernel<<<ceil(N / 256.0), 256>>>(dev_arr, N);
            vecTranspositionSortingEvenOddKernel<<<ceil(N / 256.0), 256>>>(dev_arr, N);
  63
  64
  65
  66
        // Copy C from the device memory
        // Free the device vectorsz
  67
  68
        cudaMemcpy(host_arr, dev_arr, arr_size, cudaMemcpyDeviceToHost);
  69
        cudaFree(dev_arr);
  70
  71
        printf("\n Array of 1024 elements after sorting is: \n");
         for (int f = 0; f < 1024; f++) printf("%.0f ", host_arr[f]);</pre>
  72
  73
         return 0;
  74 }
       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:
```

✓ 0s completed at 12:52