Saurav Rai
17558

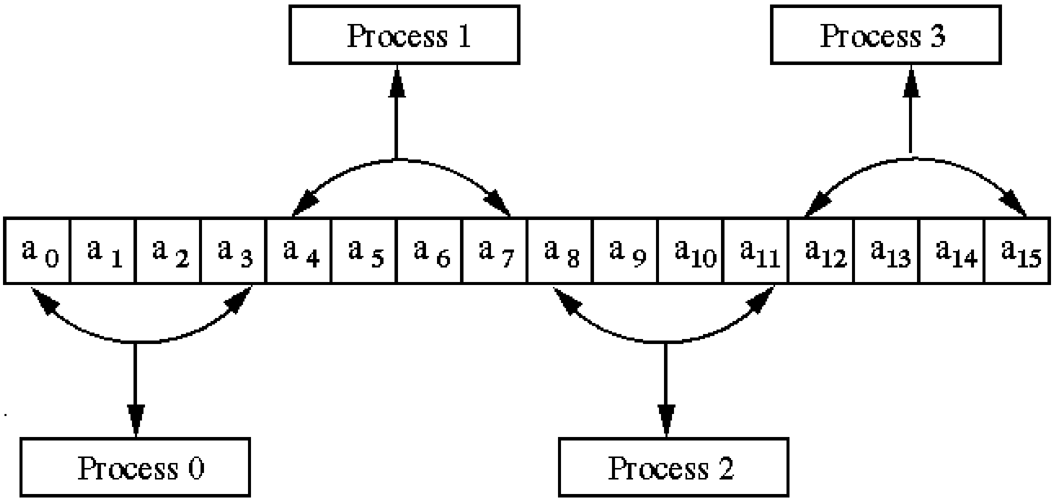
VECTOR ADDITION USING MULTIPLE GPUS

Objective of the Lab:

- To write a MPI - CUDA program, to compute the vector-vector addition on Multi-Core processor system with multi-GPU. Assume that each process computes the partial Vector Vector Addition using CUDA kernel.

Description:

- The partitioning is called block-striped if each process is assigned contiguous elements. The process P0 gets the first n/p elements, P1 gets the next n/p elements and so on.
- The distribution of 16 elements of vector A on 4 processes is shown in the following below.


Process 1
Process 3
a_0 a_1 a_2 a_3 a_4 a_5 a_6 a_7 a_8 a_9 a_{10} a_{11} a_{12} a_{13} a_{14} a_{15}
Process 0
Process 2

- Initially process with rank 0 distributes the input vectors using *MPI_Scatter* on p processes. Each process will call a CUDA kernel which performs local addition of the vectors and stores the partial addition. Now the process with rank 0 performs global reduction using *MPI_Reduce* to get the final addition product of two vectors.

Implementation of Vector Vector Addition :

Step 1 :

Four vectors are required for computation. Two arrays for vector A and Vector B and the other two arrays for temporary storage data for two vector at the each node.

Step2 :

Root process initializes the two vectors i.e Vector A and Vector B. The two vectors are constructed by assigning to each element one more than its index value.

Step 3 :

Vector Size is broadcasted to all processes from the root process.

Step 4 :

Memory is assigned for MyVectorA and MyVectorB on all nodes.

Step 5 :

Process with rank 0 distributes the input vectors using MPI_Scatter on to p processes.

Step 6 :

Similar arrays are allocated on the device. The values of the arrays in the host machine are c opied on to the arrays allocated on the device.

Step 7:

Each node computes the partial addition value by calling VectorVectorAddition Cuda kernel.

Step 8 :

Process with rank 0 performs global reduction using MPI_Reduce to get the final addtion of two vectors.

Step 9 : Process with rank 0 prints the addition value.

CUDA API used :

To Allocate memory on device-GPU :
cudaMalloc(void** array, int size)

To Free memory allocated on device-GPU:
cudaFree(void* array)

To transfer from host-CPU to device-GPU:
cudaMemcpy((void*)device_array, (void*)host_array, size, cudaMemcpyHostToDevice)

To transfer from device-GPU to host-GPU:
cudaMemcpy((void*)host_array, (void*)device_array, size, cudaMemcpyDeviceToHost)

INPUT:

The input to the problem is given as arguments in the command line. It should be given in the following format ; Suppose that the size of the vector is n and the number of nodes is m, then the program must be run as,

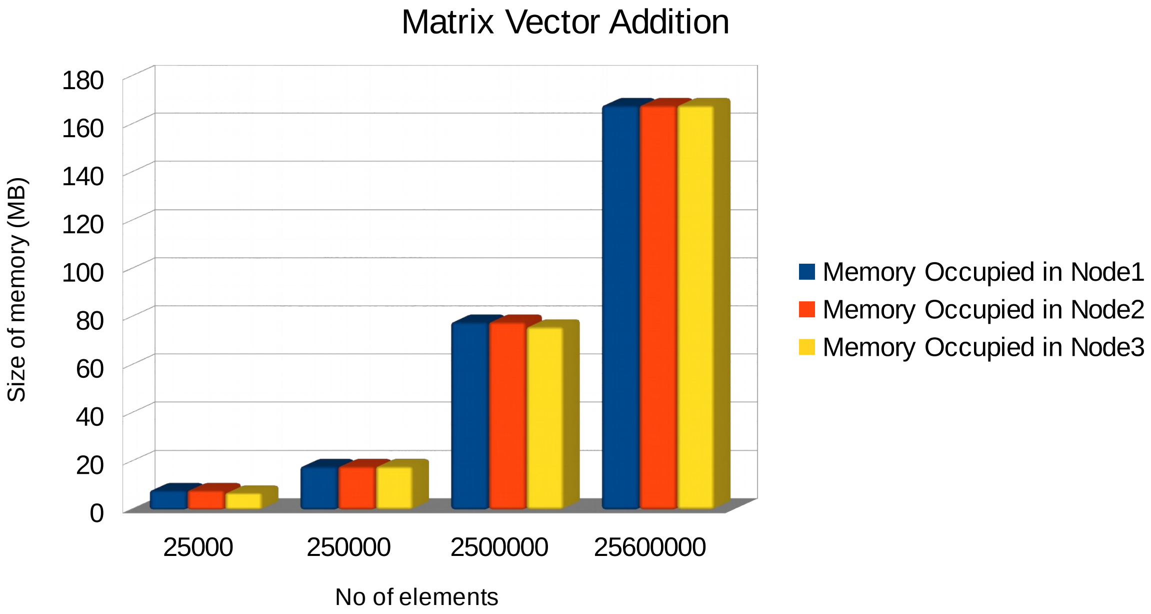
```
mpirun -n m ./program_name n
```

Process 0 generates the two vectors i.e Vector A and Vector B

OUTPUT:

Process 0 prints the final addition value of two vectors.The correctness of the output can be verified using the below formula The sum of the squares of the first n numbers is (n(n+1)(2n+1))/6.

Input Size	Memory Occupied (Node 1)	Memory Occupied (Node 2)	Memory Occupied (Node 3)
25000	8	8	8
250000	20	20	18
2500000	78	78	76
25600000	168	168	168



POINT TO NOTE:

Each node got an equal no of the elements and hence does an equal number of computation.