Name: Ayush Arikar **Roll no: 47**
Department: IT **Subject: Parallel Computing Lab**
Registration No: 18010838

Practical 8

Aim: Write a CUDA program to find the Row Sums of given matrix.
Input: Dimension of square matrix.
Output: Row Sums of matrix with speed up.

Theory:

What is CUDA?

CUDA is a parallel computing platform and programming model developed by NVidia for general computing on its own GPUs (graphics processing units). CUDA enables developers to speed up compute-intensive applications by harnessing the power of GPUs for the parallelizable part of the computation.

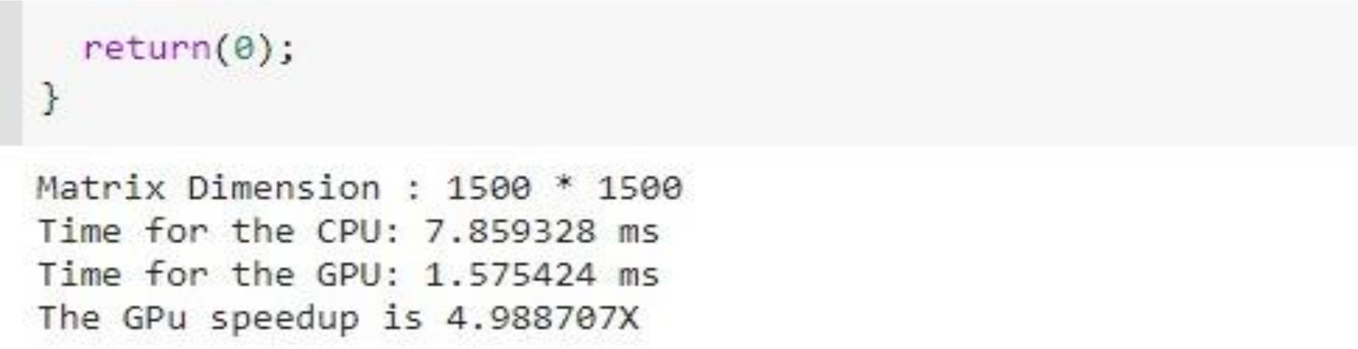
CUDA performance boost: CUDA has improved and broadened its scope over the years, more or less in lockstep with improved NVidia GPUs. As of CUDA version 9.2, using multiple P100 server GPUs, you can realize up to 50x performance improvements over CPUs. The V100 (not shown in this figure) is another 3x faster for some loads. The previous generation of server GPUs, the K80, offered 5x to 12x performance improvements over CPUs.

Code:

```
#include <stdio.h>
#include <stdlib.h>
#include <cuda.h>
//#define NUMTHREADS 512 // you can specify maximum upto 512
/*Kernel code to compute the row sums of matrix */
__global__ void RowSum(int *InputMatrix, int *rowSum, int n)
{
 int tId = blockIdx.x; // this thread will handle rowSum
 int sum = 0;
 for (int i = 0; i < n; i++)
 sum += InputMatrix[tId * n + i];
 rowSum[tId] = sum;
}
int main()
{
 int n;
 int Rsum;
 cudaEvent_t start, stop;
 float CPUtime, GPUtime;
 cudaEventCreate(&start);
 cudaEventCreate(&stop);
 int *Hmatrix, *Dmatrix, *Hrowsum, *Drowsum, *cpuRowSum;
 n = 1500; // matrix dimension
printf("Matrix Dimension : %d * %d",n,n);
 int MatrixSize = n * n * sizeof(int);

 Hmatrix = (int *) malloc(MatrixSize);
 Hrowsum = (int *) malloc(n * sizeof(int));
 cpuRowSum = (int *) malloc(n * sizeof(int));

 /* Initialize the matrix*/
 for(int i = 0; i < n; i++)
 {
 for(int j = 0; j < n; j++)
 {
 Hmatrix[i * n + j] = rand() % 10;
 }
 }

 /*Display the matrix*/
/* for(int i = 0; i < n; i++)
 {
3
 for(int j = 0; j < n; j++)
 {
 printf("%d ",Hmatrix[i * n + j]);
 }
 printf("\n");
 }
*/
 cudaEventRecord(start, 0);
 /*Row sum on CPU */
 for(int i = 0; i < n; i++)
 {
 Rsum = 0;
 for(int j = 0; j < n; j++)
 {
 Rsum += Hmatrix[i * n + j];
 }
 cpuRowSum[i] = Rsum;
 }
 cudaEventRecord(stop, 0);
 cudaEventSynchronize(stop);
 cudaEventElapsedTime(&CPUtime, start, stop);
 printf ("Time for the CPU: %f ms\n", CPUtime);

/* printf("The CPU row sum\n");
 for(int i = 0; i < n; i++)
 printf("%d\n", cpuRowSum[i]);
*/

 /* Memory allocation on device */
 cudaMalloc((void **)&Dmatrix, MatrixSize);
 cudaMalloc((void **)&Drowsum, (n * sizeof(int)));

 /* Copy host matrix to device matrix */
 cudaMemcpy(Dmatrix, Hmatrix, MatrixSize, cudaMemcpyHostToDevice);

 cudaEventRecord(start, 0);
 /* Kernel call */
 RowSum <<< n, 1 >>> (Dmatrix, Drowsum, n);

 /* wait for all threads to finish the work */
 cudaThreadSynchronize();

 cudaEventRecord(stop, 0);
 cudaEventSynchronize(stop);
 cudaEventElapsedTime(&GPUtime, start, stop);
 printf ("Time for the GPU: %f ms\n", GPUtime);

 /*copy row sum from device to host */
 cudaMemcpy(Hrowsum, Drowsum, (n * sizeof(int)),
cudaMemcpyDeviceToHost);

4
 /*check row sum */
 for(int i = 0; i < n; i++)
 {
 if(Hrowsum[i] == cpuRowSum[i])
 continue;
 else
 {
 printf("The row sum is incorrect\n");
 break;
 }
 }
 printf("The GPu speedup is %fX\n", CPUtime/GPUtime);
 /*Display row sum of matrix*/
/* printf("The GPU row sum\n");
 for(int i = 0; i < n; i++)
 printf("%d\n", Hrowsum[i]);
*/
 /* Free the host and device memory*/
 free(Hmatrix);
 free(Hrowsum);
 free(cpuRowSum);
 cudaFree(Dmatrix);
 cudaFree(Drowsum);

 return(0);
}
```

Output :

Dimensions are 1500*1500



```
  return(0);
}
```

```
Matrix Dimension : 1500 * 1500
Time for the CPU: 7.859328 ms
Time for the GPU: 1.575424 ms
The GPu speedup is 4.988707X
```

Conclusion: We have successfully implemented a CUDA Program to perform Row Sums of given matrix