Name : Harshal Kodgire
PRN : 2019BTECS00029
Batch : B1
Sub : HPC Lab

# Assignment - 10

**1. ImplementMatrix-matrix Multiplication using global memory in CUDA C. Analyze and tune the program for getting maximum speed up. Do Profiling and state what part of the code takes the huge amount of time to execute.**

**Code :**

```
#include <stdio.h>

void initWith(float num, float *a, int SIZE)
{
  for(int i = 0; i < SIZE; ++i)
  {
    a[i] = num;
  }
}

__global__
void matrixMultiply(float *result, float *a, float *b, int N, int SIZE)
{
  int start = blockIdx.x * blockDim.x + threadIdx.x;
  int stride = gridDim.x * blockDim.x;

  for(int i = start; i < SIZE; i += stride)
  {
    int row = i / N;

    float sum = 0;

    for (int j = 0; j < N; j++)
    {
      sum += a[row * N + j] * b[N * j + row];
    }

    result[i] = sum;
```

```
  }
}

void checkElementsAre(float target, float *array, int SIZE)
{
  for(int i = 0; i < SIZE; i++)
  {
    if(array[i] != target)
    {
      printf("FAIL: array[%d] - %0.0f does not equal %0.0f\n", i,
array[i], target);
      exit(1);
    }
  }
  printf("SUCCESS! All values multiplied correctly.\n");
}
int main()
{
  const int N = 1024;
  const int SIZE = N * N; // sqaure matrix
  size_t size = SIZE * sizeof(float);

  float *a;
  float *b;
  float *c;

  cudaMallocManaged(&a, size);
  cudaMallocManaged(&b, size);
  cudaMallocManaged(&c, size);

  initWith(3, a, SIZE);
  initWith(4, b, SIZE);
  initWith(0, c, SIZE);

  matrixMultiply<<<100, 1024>>>(c, a, b, N, SIZE);
  cudaDeviceSynchronize();

  checkElementsAre(12288, c, SIZE);

  cudaFree(a);
  cudaFree(b);
  cudaFree(c);
}
```

## For <<<1,1>>>

```
CUDA API Statistics:

 Time(%)  Total Time (ns)  Num Calls     Average       Minimum      Maximum           Name
 -------  ---------------  ---------  -------------  -----------  -----------  --------------------
    98.5      17517828790          1  17517828790.0  17517828790  17517828790  cudaDeviceSynchronize
     1.5        257972371          3     85990790.3        11934    257920335  cudaMallocManaged
     0.0           988305          3       329435.0       238272       453146  cudaFree
     0.0           171998          1       171998.0       171998       171998  cudaLaunchKernel


CUDA Kernel Statistics:

 Time(%)  Total Time (ns)  Instances     Average       Minimum      Maximum                          Name
 -------  ---------------  ---------  -------------  -----------  -----------  ----------------------------------------------
--
   100.0      17517909720          1  17517909720.0  17517909720  17517909720  matrixMultiply(float*, float*, float*, int, in
t)
```

## For <<<2,64>>>

```
CUDA API Statistics:

 Time(%)  Total Time (ns)  Num Calls    Average     Minimum    Maximum          Name
 -------  ---------------  ---------  -----------  ---------  ---------  --------------------
    51.1        259782965          1  259782965.0  259782965  259782965  cudaDeviceSynchronize
    48.7        247815716          3   82605238.7      11843  247763813  cudaMallocManaged
     0.2          1011007          3     337002.3     249803     451445  cudaFree
     0.0            54573          1      54573.0      54573      54573  cudaLaunchKernel


CUDA Kernel Statistics:

 Time(%)  Total Time (ns)  Instances    Average     Minimum    Maximum                       Name
 -------  ---------------  ---------  -----------  ---------  ---------  ------------------------------------------------
   100.0        259773469          1  259773469.0  259773469  259773469  matrixMultiply(float*, float*, float*, int, int)
```

## For <<<8,1024>>>

```
CUDA API Statistics:

 Time(%)  Total Time (ns)  Num Calls   Average    Minimum    Maximum          Name
 -------  ---------------  ---------  ----------  --------  ---------  --------------------
    86.6        251401904          3  83800634.7     13364  251343474  cudaMallocManaged
    13.0         37627422          1  37627422.0  37627422   37627422  cudaDeviceSynchronize
     0.4          1301650          3    433883.3    342498     493750  cudaFree
     0.0           112198          1    112198.0    112198     112198  cudaLaunchKernel


CUDA Kernel Statistics:

 Time(%)  Total Time (ns)  Instances   Average    Minimum   Maximum                       Name
 -------  ---------------  ---------  ----------  --------  --------  ------------------------------------------------
   100.0         37685507          1  37685507.0  37685507  37685507  matrixMultiply(float*, float*, float*, int, int)
```

**2. Implement Matrix-Matrix Multiplication using shared memory in CUDA C.Analyze and tune the program for getting maximum speed up. Do Profiling and state what part of the code takes a huge amount of time to execute.**

**Code :**

```
#include <stdio.h>

void initWith(float num, float *a, int SIZE)
{
  for(int i = 0; i < SIZE; ++i)
  {
    a[i] = num;
  }
}

__global__
void matrixMultiply(float *result, float *a, float *b, int N, int SIZE)
{
  __shared__ int stride;
  if (threadIdx.x == 0)
    stride = gridDim.x * blockDim.x;

  __syncthreads();

  int start = blockIdx.x * blockDim.x + threadIdx.x;

  for(int i = start; i < SIZE; i += stride)
  {
    int row = i / N;

    float sum = 0;

    for (int j = 0; j < N; j++)
    {
      sum += a[row * N + j] * b[N * j + row];
    }

    result[i] = sum;
  }
}
```

```
void checkElementsAre(float target, float *array, int SIZE)
{
  for(int i = 0; i < SIZE; i++)
  {
    if(array[i] != target)
    {
      printf("FAIL: array[%d] - %0.0f does not equal %0.0f\n", i,
array[i], target);
      exit(1);
    }
  }
  printf("SUCCESS! All values multiplied correctly.\n");
}

int main()
{
  const int N = 1024;
  const int SIZE = N * N; // sqaure matrix
  size_t size = SIZE * sizeof(float);

  float *a;
  float *b;
  float *c;

  cudaMallocManaged(&a, size);
  cudaMallocManaged(&b, size);
  cudaMallocManaged(&c, size);

  initWith(3, a, SIZE);
  initWith(4, b, SIZE);
  initWith(0, c, SIZE);

  matrixMultiply<<<100, 1024>>>(c, a, b, N, SIZE);
  cudaDeviceSynchronize();

  checkElementsAre(12288, c, SIZE);

  cudaFree(a);
  cudaFree(b);
  cudaFree(c);
}
```

## For <<<1,1>>>

```
CUDA API Statistics:

 Time(%)  Total Time (ns)  Num Calls     Average       Minimum      Maximum           Name
 -------  ---------------  ---------  -------------  -----------  -----------  --------------------
    98.5      17369051221          1  17369051221.0  17369051221  17369051221  cudaDeviceSynchronize
     1.5        258258307          3     86086102.3        12171    258206765  cudaMallocManaged
     0.0          1041476          3       347158.7       247639       478768  cudaFree
     0.0            44540          1        44540.0        44540        44540  cudaLaunchKernel


CUDA Kernel Statistics:

 Time(%)  Total Time (ns)  Instances     Average       Minimum      Maximum                          Name
 -------  ---------------  ---------  -------------  -----------  -----------  ----------------------------------------------
--
   100.0      17369040074          1  17369040074.0  17369040074  17369040074  matrixMultiply(float*, float*, float*, int, in
t)
```

## For<<<2,32>>>

```
CUDA API Statistics:

 Time(%)  Total Time (ns)  Num Calls    Average     Minimum    Maximum          Name
 -------  ---------------  ---------  -----------  ---------  ---------  --------------------
    62.1        409089518          1  409089518.0  409089518  409089518  cudaDeviceSynchronize
    37.7        248680286          3   82893428.7      11056  248636404  cudaMallocManaged
     0.2          1111405          3     370468.3     308327     412287  cudaFree
     0.0            44526          1      44526.0      44526      44526  cudaLaunchKernel


CUDA Kernel Statistics:

 Time(%)  Total Time (ns)  Instances    Average     Minimum    Maximum                        Name
 -------  ---------------  ---------  -----------  ---------  ---------  ------------------------------------------------
   100.0        409076954          1  409076954.0  409076954  409076954  matrixMultiply(float*, float*, float*, int, int)
```

## For <<<8,1024>>>

```
CUDA API Statistics:

 Time(%)  Total Time (ns)  Num Calls   Average    Minimum    Maximum          Name
 -------  ---------------  ---------  ----------  --------  ---------  --------------------
    86.7        245422323          3  81807441.0     11958  245372065  cudaMallocManaged
    12.9         36514318          1  36514318.0  36514318   36514318  cudaDeviceSynchronize
     0.3           964639          3    321546.3    290844     379769  cudaFree
     0.0            36969          1     36969.0     36969      36969  cudaLaunchKernel


CUDA Kernel Statistics:

 Time(%)  Total Time (ns)  Instances   Average    Minimum   Maximum                        Name
 -------  ---------------  ---------  ----------  --------  --------  ------------------------------------------------
   100.0         36506815          1  36506815.0  36506815  36506815  matrixMultiply(float*, float*, float*, int, int)
```

**3. ImplementPrefix sum using CUDA C. Analyze and tune the program for getting maximum speed up. Do Profiling and state what part of the code takes the huge amount of time to execute.**

**Code :**

```
#include <stdio.h>

void initWith(float val, float *arr, int N)
{
  for (int i = 0; i < N; i++)
  {
    arr[i] = val;
  }
}

__global__
void prefixSum(float *arr, float *res, float *ptemp, float* ttemp, int
N)
{
  int threadId = blockIdx.x * blockDim.x + threadIdx.x;
  int totalThreads = gridDim.x * blockDim.x;
  int elementsPerThread = ceil(1.0 * N / totalThreads);

  int start = threadId * elementsPerThread;
  int count = 0;
  float *sums = new float[elementsPerThread];
  float sum = 0;

  for (int i = start; i < N && count < elementsPerThread; i++, count++)
{
    sum += arr[i];
    sums[count] = sum;
  }

  float localSum;
  if (count)
    localSum = sums[count - 1];
  else
    localSum = 0;
  ptemp[threadId] = localSum;
  ttemp[threadId] = localSum;
```

```
  __syncthreads();

  if (totalThreads == 1) {
    for (int i = 0; i < N; i++)
      res[i] = sums[i];
  } else {
    int d = 0; // log2(totalThreads)
    int x = totalThreads;
    while (x > 1) {
      d++;
      x = x >> 1;
    }

    x = 1;
    for (int i = 0; i < 2*d; i++) {
      int tsum = ttemp[threadId];

      __syncthreads();

      int newId = threadId / x;
      if (newId % 2 == 0) {
        int nextId = threadId + x;
        ptemp[nextId] += tsum;
        ttemp[nextId] += tsum;
      } else {
        int nextId = threadId - x;
        ttemp[nextId] += tsum;
      }

      x = x << 1;
    }

    __syncthreads();

    float diff = ptemp[threadId] - localSum;
    for (int i = start, j = 0; i < N && j < count; i++, j++) {
      res[i] = sums[j] + diff;
    }
  }
}

void checkRes(float *arr, float *res, int N, float *ptemp, float*
ttemp)
```

```
{
  float sum = 0;
  for (int i = 0; i < N; i++)
  {
    sum += arr[i];
    if (sum != res[i])
    {
      printf("FAIL: res[%d] - %0.0f does not equal %0.0f\n", i, res[i],
sum);
      exit(1);
    }
  }
  printf("SUCCESS! All prefix sums added correctly.\n");
}

int main()
{
  const int N = 1000000;
  size_t size = N * sizeof(float);

  float *arr;
  float *res;

  cudaMallocManaged(&arr, size);
  cudaMallocManaged(&res, size);

  initWith(2, arr, N);
  initWith(0, res, N);

  int blocks = 1;
  int threadsPerBlock = 32;
  int totalThreads = blocks * threadsPerBlock;

  float *ptemp;
  float *ttemp;
  cudaMallocManaged(&ptemp, totalThreads * sizeof(float));
  cudaMallocManaged(&ttemp, totalThreads * sizeof(float));

  prefixSum<<<blocks, threadsPerBlock>>>(arr, res, ptemp, ttemp, N);
  cudaDeviceSynchronize();

  checkRes(arr, res, N, ptemp, ttemp);
```

```
  cudaFree(arr);
  cudaFree(res);
  cudaFree(ttemp);
  cudaFree(ptemp);
}
```

## For <<<1,1>>>

```
CUDA API Statistics:

 Time(%)  Total Time (ns)  Num Calls    Average     Minimum    Maximum          Name
 -------  ---------------  ---------  -----------  ---------  ---------  ---------------------
    55.1        258619904          4   64654976.0      12965  258418165  cudaMallocManaged
    44.6        209494653          1  209494653.0  209494653  209494653  cudaDeviceSynchronize
     0.2           877024          4     219256.0      18382     457012  cudaFree
     0.1           643360          1     643360.0     643360     643360  cudaLaunchKernel


CUDA Kernel Statistics:

 Time(%)  Total Time (ns)  Instances    Average     Minimum    Maximum                      Name
 -------  ---------------  ---------  -----------  ---------  ---------  ---------------------------------------------
   100.0        209485950          1  209485950.0  209485950  209485950  prefixSum(float*, float*, float*, float*, int)
```

## For <<<2,32>>>

```
CUDA API Statistics:

 Time(%)  Total Time (ns)  Num Calls    Average    Minimum    Maximum          Name
 -------  ---------------  ---------  -----------  --------  ---------  ---------------------
    94.4        273722276          4  68430569.0      5906  273598108  cudaMallocManaged
     5.5         15910297          1  15910297.0  15910297   15910297  cudaDeviceSynchronize
     0.1           346921          1    346921.0    346921     346921  cudaLaunchKernel


CUDA Kernel Statistics:

 Time(%)  Total Time (ns)  Instances    Average    Minimum   Maximum                      Name
 -------  ---------------  ---------  -----------  --------  --------  ---------------------------------------------
   100.0         15899585          1  15899585.0  15899585  15899585  prefixSum(float*, float*, float*, float*, int)
```

## For <<<8,1024>>>

```
CUDA API Statistics:

 Time(%)  Total Time (ns)  Num Calls    Average     Minimum    Maximum          Name
 -------  ---------------  ---------  -----------  ---------  ---------  ---------------------
    55.5        240526657          4   60131664.3       4048  240435145  cudaMallocManaged
    44.4        192252427          1  192252427.0  192252427  192252427  cudaDeviceSynchronize
     0.1           314547          1     314547.0     314547     314547  cudaLaunchKernel
```

**4. Implement 2D Convolution using shared memoryusing CUDA C. Analyze and tune the program for getting maximum speed up. Do Profiling and state what part of the code takes the huge amount of time to execute.**

**Code :**

```
#include <stdio.h>

#define MASK_DIM 7

#define MASK_OFFSET (MASK_DIM / 2)

__constant__ int mask[7 * 7];

__global__ void convolution_2d(int *matrix, int *result, int N)
{
    // Calculate the global thread positions
    int row = blockIdx.y * blockDim.y + threadIdx.y;
    int col = blockIdx.x * blockDim.x + threadIdx.x;

    // Starting index for calculation
    int start_r = row - MASK_OFFSET;
    int start_c = col - MASK_OFFSET;

    // Temp value for accumulating the result
    int temp = 0;

    // Iterate over all the rows
    for (int i = 0; i < MASK_DIM; i++)
    {
        // Go over each column
        for (int j = 0; j < MASK_DIM; j++)
        {
            // Range check for rows
            if ((start_r + i) >= 0 && (start_r + i) < N)
            {
                // Range check for columns
                if ((start_c + j) >= 0 && (start_c + j) < N)
                {
                    // Accumulate result
                    temp += matrix[(start_r + i) * N + (start_c + j)] *
mask[i * MASK_DIM + j];
```

```
                }
            }
        }
    }

    // Write back the result
    result[row * N + col] = temp;
}

void init_matrix(int *m, int n)
{
    for (int i = 0; i < n; i++)
    {
        for (int j = 0; j < n; j++)
        {
            m[n * i + j] = rand() % 100;
        }
    }
}

void verify_result(int *m, int *mask, int *result, int N)
{

    int temp;

    int offset_r;
    int offset_c;

    // Go over each row
    for (int i = 0; i < N; i++)
    {
        // Go over each column
        for (int j = 0; j < N; j++)
        {
            // Reset the temp variable
            temp = 0;

            // Go over each mask row
            for (int k = 0; k < MASK_DIM; k++)
            {
                // Update offset value for row
                offset_r = i - MASK_OFFSET + k;
```

```
                // Go over each mask column
                for (int l = 0; l < MASK_DIM; l++)
                {
                    // Update offset value for column
                    offset_c = j - MASK_OFFSET + l;

                    // Range checks if we are hanging off the matrix
                    if (offset_r >= 0 && offset_r < N)
                    {
                        if (offset_c >= 0 && offset_c < N)
                        {
                            // Accumulate partial results
                            temp += m[offset_r * N + offset_c] * mask[k
* MASK_DIM + l];
                        }
                    }
                }
            }
            // Fail if the results don't match
            if (result[i * N + j] != temp)
            {
                printf("Check failed");
                return;
            }
        }
    }
}

int main()
{

    int N = 1 << 10; // 2^10

    size_t bytes_n = N * N * sizeof(int);
    size_t bytes_m = MASK_DIM * MASK_DIM * sizeof(int);

    int *matrix;
    int *result;
    int *h_mask;

    cudaMallocManaged(&matrix, bytes_n);
    cudaMallocManaged(&result, bytes_n);
    cudaMallocManaged(&h_mask, bytes_m);
```

```
    init_matrix(matrix, N);
    init_matrix(mask, MASK_DIM);

    cudaMemcpyToSymbol(mask, h_mask, bytes_m);

    // Calculate grid dimensions
    int THREADS = 32;
    int BLOCKS = (N + THREADS - 1) / THREADS;

    // Dimension launch arguments
    dim3 block_dim(THREADS, THREADS);
    dim3 grid_dim(BLOCKS, BLOCKS);

    convolution_2d<<<grid_dim, block_dim>>>(matrix, result, N);

    verify_result(matrix, h_mask, result, N);

    printf("COMPLETED SUCCESSFULLY!");

    cudaFree(matrix);
    cudaFree(result);
    cudaFree(h_mask);

    return 0;
}
```

## Thread : 1

```
CUDA API Statistics:

 Time(%)  Total Time (ns)  Num Calls   Average    Minimum   Maximum          Name
 -------  ---------------  ---------  ----------  -------  ---------  ------------------
    99.6        262995034          3  87665011.3    19260  262917410  cudaMallocManaged
     0.3           828399          3    276133.0    42626     467855  cudaFree
     0.1           145724          1    145724.0   145724     145724  cudaMemcpyToSymbol
     0.0            42681          1     42681.0    42681      42681  cudaLaunchKernel


CUDA Kernel Statistics:

 Time(%)  Total Time (ns)  Instances   Average    Minimum   Maximum                Name
 -------  ---------------  ---------  ----------  --------  --------  -------------------------------
   100.0         13188981          1  13188981.0  13188981  13188981  convolution_2d(int*, int*, int)
```

## Thread : 32

CUDA API Statistics:

| Time(%) | Total Time (ns) | Num Calls | Average | Minimum | Maximum | Name |
|---|---|---|---|---|---|---|
| 99.7 | 294157376 | 3 | 98052458.7 | 16277 | 294105578 | cudaMallocManaged |
| 0.3 | 835546 | 3 | 278515.3 | 51361 | 476559 | cudaFree |
| 0.0 | 119871 | 1 | 119871.0 | 119871 | 119871 | cudaMemcpyToSymbol |
| 0.0 | 32208 | 1 | 32208.0 | 32208 | 32208 | cudaLaunchKernel |

CUDA Kernel Statistics:

| Time(%) | Total Time (ns) | Instances | Average | Minimum | Maximum | Name |
|---|---|---|---|---|---|---|
| 100.0 | 4863826 | 1 | 4863826.0 | 4863826 | 4863826 | convolution_2d(int*, int*, int) |

## Thread : 256

CUDA API Statistics:

| Time(%) | Total Time (ns) | Num Calls | Average | Minimum | Maximum | Name |
|---|---|---|---|---|---|---|
| 99.8 | 403220402 | 3 | 134406800.7 | 35548 | 403134048 | cudaMallocManaged |
| 0.1 | 566824 | 3 | 188941.3 | 39183 | 357136 | cudaFree |
| 0.0 | 129007 | 1 | 129007.0 | 129007 | 129007 | cudaMemcpyToSymbol |
| 0.0 | 927 | 1 | 927.0 | 927 | 927 | cudaLaunchKernel |

CUDA Memory Operation Statistics (by time):

| Time(%) | Total Time (ns) | Operations | Average | Minimum | Maximum | Operation |
|---|---|---|---|---|---|---|
| 100.0 | 2368 | 1 | 2368.0 | 2368 | 2368 | [CUDA memcpy DtoD] |