**DEPARTMENT OF MATHEMATICS AND COMPUTING**

**V-M.Tech. (M&C)**

**Monsoon Semester 2022-2023**

**GPU Computing Lab**

**(MCC302)**

**LAB-7**

**Sum Reduction Algorithm**

NAME: **ROMEO SARKAR**

ADMISSION NO.: **20JE0814**

DATE OF EXPERIMENT: **12-10-2022**

**Experiment 2.1:** Sum reduction

**Objectives:** Write a CUDA program for sum reduction.

**CUDA Sample Program:**

#include <cuda\_runtime.h>

#include <stdio.h>

#define N 100

#define BD 256

#define CHECK(call) \

{\

    const cudaError\_t error **=** call;\

    if (error **!=** cudaSuccess)\

    {\

        fprintf (stderr, "error: %s: %d,", \_\_FILE\_\_, \_\_LINE\_\_);\

        fprintf (stderr, "code:%d, reason:%s\n", error, cudaGetErrorString (error));\

        exit (1);\

    }\

}

\_\_global\_\_ void sumReduce (float \*dev\_a, float \*dev\_b)

{

    \_\_shared\_\_ float partialSum[BD];

    partialSum[threadIdx.x] **=** dev\_a[blockIdx.x **\*** blockDim.x **+** threadIdx.x];

    unsigned int t **=** threadIdx.x;

    for (unsigned int stride **=** 1; stride **<** blockDim.x; stride **\*=** 2)

    {

        \_\_syncthreads ();

        if ((t **%** (2 **\*** stride)) **==** 0)

        {

            partialSum[t] **+=** partialSum[t **+** stride];

        }

    }

    if (0 **==** threadIdx.x)

    {

        dev\_b[blockIdx.x] **=** partialSum[0];

    }

    return;

}

int main (int argc, char \*\*argv)

{

    float a[N], b[N];

    float **\***dev\_a, **\***dev\_b;

    int bdimx **=** BD;

    float elapsedTime;

    dim3 block (bdimx);

    dim3 grid ((N **+** block.x **-** 1) **/** block.x, 1, 1);

    cudaEvent\_t start, stop;

    CHECK (cudaEventCreate (**&**start));

    CHECK (cudaEventCreate (**&**stop));

    printf ("Array Size is = %d\n", N);

    //allocate the memory on device

    CHECK (cudaMalloc ((void **\*\***) **&**dev\_a, N **\*** **sizeof** (float)));

    CHECK (cudaMalloc ((void **\*\***) **&**dev\_b, N **\*** **sizeof** (float)));

    for (int i **=** 0; i **<** N; i**++**)

    {

        a[i] **=** 1;

        // a[i] = i + 1;

        // a[i] = ((float) (rand ())) / (float) (RAND\_MAX);

    }

    //Cuda events for time measure

    CHECK (cudaEventRecord (start, 0));

    cudaMemcpy (dev\_a, a, N **\*** **sizeof** (float), cudaMemcpyHostToDevice);

    CHECK (cudaEventRecord (stop, 0));

    CHECK (cudaEventSynchronize (stop));

    cudaEventElapsedTime (**&**elapsedTime, start, stop);

    printf ("Time to do memory transfer of array a from host to device is %3.6f ms\n", elapsedTime);

    //kernel launch

    CHECK (cudaEventRecord (start, 0));

    sumReduce **<<<**grid, block**>>>** (dev\_a, dev\_b);

   //Copy result from device to host

    CHECK (cudaMemcpy (b, dev\_b, N **\*** **sizeof** (float), cudaMemcpyDeviceToHost));

    CHECK (cudaEventRecord (stop, 0));

    CHECK (cudaEventSynchronize (stop));

    cudaEventElapsedTime (**&**elapsedTime, start, stop);

    printf ("Time to do sum reduction is %3.6f ms\n", elapsedTime);

    printf ("Sum = %f\n", b[0]);

    cudaDeviceSynchronize ();

    cudaEventDestroy (start);

    cudaEventDestroy (stop);

    cudaFree (dev\_a);

    cudaFree (dev\_b);

    return 0;

}

SumReduction.cu

**Output:**

Array Size is **=** 100

Time to **do** memory transfer of array a from host to device is 0.024416 ms

Time to **do** sum reduction is 0.115840 ms

Sum **=** 100.000000

**Lab Exercise 2.2:** Write a CUDA program to demonstrate the followings:

1. Write a header file for declaring Error function
2. Write device functions to do the sum reduction with less warp divergence
3. Print the execution time of the kernel and compare with classical sum reduction as given in 2.1
4. Print the result

**CODE:**

***#ifndef*** ERROR\_CUH

***#define*** ERROR\_CUH

***#define*** chkError(param) **\**

{ **\**

    cudaError\_t err **=** (param)**;** **\**

**if** (err **!=** cudaSuccess) **\**

    { **\**

        printf (**"%s**(**\033**[1;32m**%d\033**[m): **\033**[1;4;31merror**\033**[m: **\033**[1;33m**%s\033**[m i.e. **%s\n",** \_\_FILE\_\_**,** \_\_LINE\_\_**,** cudaGetErrorName (err)**,** cudaGetErrorString (err))**;** **\**

        exit (err)**;** **\**

    } **\**

}

***#define*** getLastError() **\**

{ **\**

    cudaError\_t err **=** cudaGetLastError ()**;** **\**

**if** (err **!=** cudaSuccess) **\**

    { **\**

        printf (**"%s**(**\033**[1;32m**%d\033**[m): **\033**[1;4;31merror**\033**[m: **\033**[1;33m**%s\033**[m i.e. **%s\n",** \_\_FILE\_\_**,** \_\_LINE\_\_**,** cudaGetErrorName (err)**,** cudaGetErrorString (err))**;** **\**

        exit (err)**;** **\**

    } **\**

}

***#endif***

Error.cuh

***#include*** <stdio.h>

***#include*** <time.h>

// macros

***#define*** \_SHARED\_ARR\_LEN\_ **347U**

***#define*** ceil\_div(a, b) (((a) **+** (b) **-** **1**) **/** (b))

***#define*** floor\_div(a, b) ((a) **/** (b))

***#include*** "Error.cuh"

\_\_global\_\_ void reduced\_sum1 (double \*arr, double \*sum, size\_t size)

{

    \_\_shared\_\_ double s\_arr[\_SHARED\_ARR\_LEN\_];

    // unsigned int s = ceil\_div (\_SHARED\_ARR\_LEN\_, 2);

    // unsigned int globalIdx = 2 \* threadIdx.x + blockIdx.x \* \_SHARED\_ARR\_LEN\_;

    // if (2 \* threadIdx.x < )

    unsigned int s;

**if** (floor\_div (size, 2 \* \_SHARED\_ARR\_LEN\_) == blockIdx.x)

    {

        s **=** ceil\_div (size % (2 \* \_SHARED\_ARR\_LEN\_), 2);

    }

**else**

    {

        s **=** \_SHARED\_ARR\_LEN\_;

    }

**if** ((threadIdx.x < s) **&&** ((2 \* (threadIdx.x + blockIdx.x \* \_SHARED\_ARR\_LEN\_)) < size))

    {

        s\_arr[threadIdx.x] **=** arr[2 \* (threadIdx.x + blockIdx.x \* \_SHARED\_ARR\_LEN\_)];

    }

    // else

    // {

    //     goto finish\_line;

    // }

    \_\_syncthreads ();

**if** ((threadIdx.x < s) **&&** ((2 \* (threadIdx.x + blockIdx.x \* \_SHARED\_ARR\_LEN\_) + 1) < size))

    {

        s\_arr[threadIdx.x] **+=** arr[2 \* (threadIdx.x + blockIdx.x \* \_SHARED\_ARR\_LEN\_) + 1];

    }

    // if (0 == blockIdx.x)

    // {

    //     // for (unsigned int i = 0; i < \_SHARED\_ARR\_LEN\_; i++)

    //     // {

    //     printf ("%.0lf ", s\_arr[threadIdx.x]);

    //     // }

    //     \_\_syncthreads ();

    //     if (0 == threadIdx.x)

    //         printf ("\n");

    // }

    \_\_syncthreads ();

    // unsigned int

    // now, find the sum of the entire block:

**for** (unsigned int stride **=** 1; stride < s; stride **<<=** 1)

    {

**if** ((threadIdx.x % (stride << 1)) == 0)

        {

**if** ((threadIdx.x + stride) < s)

            {

                s\_arr[threadIdx.x] **+=** s\_arr[threadIdx.x + stride];

            }

        }

        // else

        // {

        //     goto finish\_line;

        // }

        \_\_syncthreads ();

    }

    sum[blockIdx.x] **=** s\_arr[0];

    // finish\_line:

    // printf ("<<%u;%u>>\n", blockIdx.x, threadIdx.x);

    // blockIdx.x

**return**;

}

\_\_global\_\_ void reduced\_sum2 (double \*arr, double \*sum, unsigned int size)

{

    \_\_shared\_\_ double s\_arr[\_SHARED\_ARR\_LEN\_];

    // #define s\_arr arr

    unsigned int globalIdx **=** threadIdx.x + blockIdx.x \* \_SHARED\_ARR\_LEN\_;

**if** (globalIdx < size)

    {

        s\_arr[threadIdx.x] **=** arr[globalIdx];

        // if (0 == blockIdx.x)

        // printf ("%.0lf ", arr[globalIdx]);

    }

    \_\_syncthreads ();

    // \_\_syncthreads ();

    // if (threadIdx.x == 0 && blockIdx.x == 0)

    //     printf ("\n\n\n\n");

    // adding the entire block

    unsigned int trailing\_stride, stride;

**if** (floor\_div (size, \_SHARED\_ARR\_LEN\_) == blockIdx.x)

    {

        // printf ("\033[90mH\033[m");

        trailing\_stride **=** size % \_SHARED\_ARR\_LEN\_;

    }

**else**

    {

        // printf ("\033[90mX\033[m");

        trailing\_stride **=** \_SHARED\_ARR\_LEN\_;

    }

    stride **=** ceil\_div (trailing\_stride, 2);

**for** (; trailing\_stride > 1; trailing\_stride **=** stride, stride **=** ceil\_div (stride, 2))

    {

**if** (threadIdx.x < stride)

        {

**if** ((threadIdx.x + stride) < trailing\_stride)

            {

                s\_arr[threadIdx.x] **+=** s\_arr[threadIdx.x + stride];

                // if (blockIdx.x == 0)

                // {

                //     printf ("\033[32m%d->%.0lf\033[m ", threadIdx.x, s\_arr[threadIdx.x]);

                // }

            }

            // \_\_syncthreads ();

        }

**else**

        {

**goto** finish\_line;

        }

        // \_\_syncthreads ();

        // if (blockIdx.x == 0 && threadIdx.x == 0)

        // {

        //     printf ("\n\n\n\n");

        // }

        \_\_syncthreads ();

    }

    // if (0 == threadIdx.x)

    // {

    //     printf ("\033[32m%.1lf\033[m ", s\_arr[0]);

    // }

    // if (0 == blockIdx.x && threadIdx.x == 0)

    // {

    //     printf ("\033[31m%lf\033[m\n", s\_arr[0]);

    // }

**if** (0 == threadIdx.x)

    {

        sum[blockIdx.x] **=** s\_arr[0];

    }

    // sum[blockIdx.x] = arr[blockIdx.x \* \_SHARED\_ARR\_LEN\_];

    finish\_line:

**return**;

}

double calculate\_sum\_cpu (double \*arr, size\_t size)

{

    double s **=** 0;

**for** (size\_t i **=** 0; i < size; i++)

    {

        s **+=** arr[i];

    }

**return** s;

}

double calculate\_sum\_cpu (double \*arr, size\_t startIdx, size\_t endIdx)

{

    double s **=** 0;

**for** (int i **=** startIdx; i < endIdx; i++)

    {

        s **+=** arr[i];

    }

**return** s;

}

void initialize\_array (double \*arr, size\_t size)

{

    struct timespec start, stop;

    timespec\_get (&start, TIME\_UTC);

**for** (size\_t i **=** 0; i < size; i++)

    {

        // arr[i] = ((double) rand ()) \* ((double) (rand ()));

        arr[i] **=** (double) rand ();

        // arr[i] = i + 1;

        // arr[i] = 1;

    }

    timespec\_get (&stop, TIME\_UTC);

    printf ("time taken to initialize the array: **%.9lf** secs.**\n**", ((double) (stop.tv\_nsec - start.tv\_nsec) \* 1e-9 + ((double) (stop.tv\_sec - start.tv\_sec))));

**return**;

}

int cmp (const void \*a, const void \*b)

{

    const double \*x **=** (const double \*) (a), \*y **=** (const double \*) (b);

**if** (x < y)

    {

**return** 0; // i.e. don't swap

    }

**else**

    {

**return** 1; // i.e. swap

    }

}

void sort\_array (double \*arr, size\_t size)

{

    struct timespec start, stop;

    timespec\_get (&start, TIME\_UTC);

    qsort (arr, size, sizeof (double), cmp);

    timespec\_get (&stop, TIME\_UTC);

    printf ("time taken to sort the array: **%.9lf** secs.**\n**", ((double) (stop.tv\_nsec - start.tv\_nsec)) \* 1e-9 + ((double) (stop.tv\_sec - start.tv\_sec)));

**return**;

}

void print\_array (double \*arr, size\_t size)

{

**for** (size\_t i **=** 0; i < size; i++)

    {

        printf ("**%.0f** ", arr[i]);

    }

    printf ("**\n**");

**return**;

}

double calculate\_sum\_gpu1 (double \*arr, size\_t size)

{

    // double \*dev\_arr;

    // cudaMalloc (&dev\_arr, size);

    // cudaMemcpy (dev\_arr, arr, size \* sizeof (double), cudaMemcpyHostToDevice);

    // array will be divided into smaller array of size \_SHARED\_ARR\_LEN\_

    double sum;

    size\_t temp\_arr\_size **=** size, temp\_sum\_arr\_size **=** ceil\_div (temp\_arr\_size, 2 \* \_SHARED\_ARR\_LEN\_);

    double \*dev\_temp\_arr **=** NULL, \*dev\_temp\_sum\_arr **=** NULL;

    cudaMalloc (&dev\_temp\_arr, sizeof (double) \* temp\_arr\_size);

    cudaMemcpy (dev\_temp\_arr, arr, temp\_arr\_size \* sizeof (double), cudaMemcpyHostToDevice);

**for** (; temp\_arr\_size > 1; temp\_arr\_size **=** temp\_sum\_arr\_size, temp\_sum\_arr\_size **=** ceil\_div (temp\_arr\_size, 2 \* \_SHARED\_ARR\_LEN\_))

    {

        // temp\_size = ceildiv (temp\_size, \_SHARED\_ARR\_LEN\_);

        // temp\_arr = (double \*) (malloc (sizeof (double) \* temp\_arr\_size));

        // printf ("launch param: <<< %zu, %u >>>\n", temp\_sum\_arr\_size, \_SHARED\_ARR\_LEN\_);

        // printf ("launch param1: %u\n", ceil\_div (12, 5));

        chkError (cudaMalloc (&dev\_temp\_sum\_arr, sizeof (double) \* temp\_sum\_arr\_size))

        reduced\_sum1 <<< temp\_sum\_arr\_size, \_SHARED\_ARR\_LEN\_ >>> (dev\_temp\_arr, dev\_temp\_sum\_arr, temp\_arr\_size);

        getLastError ();

        cudaDeviceSynchronize ();

        // printf ("launch param2: %zu, %zu\n", temp\_arr\_size, temp\_sum\_arr\_size);

        // printf ("\n");

        // comment:

        // double \*p = (double \*) malloc (temp\_sum\_arr\_size \* sizeof (double));

        // cudaMemcpy (p, dev\_temp\_sum\_arr, sizeof (double) \* temp\_sum\_arr\_size, cudaMemcpyDeviceToHost);

        // double t;

        // for (int i = 0; i < temp\_sum\_arr\_size; i++)

        // {

        //     printf ("\033[31m%.1lf\033[m ", p[i]);

        // }

        // printf ("\n");

        // for (int i = 0; i < temp\_sum\_arr\_size; i++)

        // {

        //     if (p[i] != (t = calculate\_sum\_cpu (arr, i \* 2 \* \_SHARED\_ARR\_LEN\_, (i + 1) \* 2 \* \_SHARED\_ARR\_LEN\_)))

        //     {

        //         printf ("error: %d; %.1lf instead of %.1lf\n", i, p[i], t);

        //         exit (0);

        //     }

        // }

        // exit (0);

        // comment:

        cudaFree (dev\_temp\_arr);

        dev\_temp\_arr **=** dev\_temp\_sum\_arr;

        dev\_temp\_sum\_arr **=** NULL;

        // return 0;

    }

    cudaMemcpy (&sum, dev\_temp\_arr, sizeof (double), cudaMemcpyDeviceToHost);

    cudaFree (dev\_temp\_arr);

**return** sum;

}

double calculate\_sum\_gpu2 (double \*arr, size\_t size)

{

    // double \*dev\_arr;

    // cudaMalloc (&dev\_arr, size);

    // cudaMemcpy (dev\_arr, arr, size \* sizeof (double), cudaMemcpyHostToDevice);

    // array will be divided into smaller array of size \_SHARED\_ARR\_LEN\_

    double sum;

    size\_t temp\_arr\_size **=** size, temp\_sum\_arr\_size **=** ceil\_div (size, \_SHARED\_ARR\_LEN\_);

    double \*dev\_temp\_arr **=** NULL, \*dev\_temp\_sum\_arr **=** NULL;

    cudaMalloc (&dev\_temp\_arr, sizeof (double) \* temp\_arr\_size);

    cudaMemcpy (dev\_temp\_arr, arr, temp\_arr\_size \* sizeof (double), cudaMemcpyHostToDevice);

**for** (; temp\_arr\_size > 1; temp\_arr\_size **=** temp\_sum\_arr\_size, temp\_sum\_arr\_size **=** ceil\_div (temp\_sum\_arr\_size, \_SHARED\_ARR\_LEN\_))

    {

        // temp\_size = ceildiv (temp\_size, \_SHARED\_ARR\_LEN\_);

        // temp\_arr = (double \*) (malloc (sizeof (double) \* temp\_arr\_size));

        cudaMalloc (&dev\_temp\_sum\_arr, sizeof (double) \* temp\_sum\_arr\_size);

        reduced\_sum2 <<< temp\_sum\_arr\_size, \_SHARED\_ARR\_LEN\_ >>> (dev\_temp\_arr, dev\_temp\_sum\_arr, temp\_arr\_size);

        getLastError ();

        cudaDeviceSynchronize ();

        // printf ("\n");

        // comment:

        // double \*p = (double \*) malloc (temp\_sum\_arr\_size \* sizeof (double));

        // cudaMemcpy (p, dev\_temp\_sum\_arr, sizeof (double) \* temp\_sum\_arr\_size, cudaMemcpyDeviceToHost);

        // double t;

        // for (int i = 0; i < temp\_sum\_arr\_size && i < 1; i++)

        // {

        //     if (p[i] != (t = calculate\_sum\_cpu (arr, i \* \_SHARED\_ARR\_LEN\_, (i + 1) \* \_SHARED\_ARR\_LEN\_)))

        //     {

        //         printf ("error: %d; %.1lf instead of %.1lf\n", i, p[i], t);

        //     }

        // }

        // comment:

        /\*

        f

        sf

        sd

        \*/

        cudaFree (dev\_temp\_arr);

        dev\_temp\_arr **=** dev\_temp\_sum\_arr;

        dev\_temp\_sum\_arr **=** NULL;

        // return 0;

    }

    cudaMemcpy (&sum, dev\_temp\_arr, sizeof (double), cudaMemcpyDeviceToHost);

    cudaFree (dev\_temp\_arr);

**return** sum;

}

int sum (double \*arr, size\_t size)

{

    struct timespec start, stop;

    // timespec\_get (&start, TIME\_UTC);

    // clock\_t st = clock ();

    sort\_array (arr, size);

    // printf ("time: %.3lf secs.\n", ((double) (clock () - st)) / CLOCKS\_PER\_SEC);

    // timespec\_get (&stop, TIME\_UTC);

    // printf ("time taken to sort the array: %.9lf secs.\n", ((double) (stop.tv\_nsec - start.tv\_nsec)) \* 1e-9 + ((double) (stop.tv\_sec - start.tv\_sec)));

    /\* = = = = = = = = = = = = = \*/

    timespec\_get (&start, TIME\_UTC);

    double sum\_cpu **=** calculate\_sum\_cpu (arr, size);

    timespec\_get (&stop, TIME\_UTC);

    printf ("sum\_cpu time: **%.9lf** secs.**\n**", ((double) (stop.tv\_nsec - start.tv\_nsec) \* 1e-9 + ((double) (stop.tv\_sec - start.tv\_sec))));

    /\*= = = = = = = = = = = = = \*/

    timespec\_get (&start, TIME\_UTC);

    double sum\_gpu1 **=** calculate\_sum\_gpu1 (arr, size);

    timespec\_get (&stop, TIME\_UTC);

    printf ("sum\_gpu1 time: **%.9lf** secs.**\n**", ((double) (stop.tv\_nsec - start.tv\_nsec) \* 1e-9 + ((double) (stop.tv\_sec - start.tv\_sec))));

    /\*= = = = = = = = = = = = = \*/

    timespec\_get (&start, TIME\_UTC);

    double sum\_gpu2 **=** calculate\_sum\_gpu2 (arr, size);

    timespec\_get (&stop, TIME\_UTC);

    printf ("sum\_gpu2 time: **%.9lf** secs. **\033**[90m(less warp divergence)**\033**[m**\n**", ((double) (stop.tv\_nsec - start.tv\_nsec) \* 1e-9 + ((double) (stop.tv\_sec - start.tv\_sec))));

    /\*= = = = = = = = = = = = = \*/

    printf ("{sum\_cpu, sum\_gpu1, sum\_gpu2} = {**%.0lf**, **%.0lf**, **%.0lf**}**\n**", sum\_cpu, sum\_gpu1, sum\_gpu2);

**if** (sum\_cpu != sum\_gpu1)

    {

        printf ("**\033**[1;31merror**\033**[m: (sum\_cpu != sum\_gpu1)**\n**");

**return** 1;

    }

**else**

    {

**if** (sum\_cpu != sum\_gpu2)

        {

            printf ("**\033**[1;31merror**\033**[m: (sum\_cpu != sum\_gpu2)**\n**");

**return** 1;

        }

**else**

        {

**return** 0;

        }

    }

    // const int i = 0;

    // if (sum\_cpu != sum\_gpu2)

    // {

    //     printf ("\033[1;31merror\033[m: (sum\_cpu != sum\_gpu)\n");

    //     return 1;

    // }

    // else

    // {

    //     return 0;

    // }

**return** 0;

}

double \*allocate\_array (size\_t size)

{

    printf ("size of array: **%zd** Bytes (**%.6lf** GB)**\n**", sizeof (double) \* size, ((double) (sizeof (double) \* size)) / (1024.0 \* 1024.0 \* 1024.0));

    double \*arr **=** (double \*) (malloc (sizeof (double) \* size));

**return** arr;

}

int main ()

{

    srand (time (NULL));

    size\_t size **=** 565786565; // array size;

    double \*arr **=** allocate\_array (size);

    initialize\_array (arr, size);

    sum (arr, size);

**return** 0;

}

SumReduction.cu

**Output:**

size of array: 4526292520 Bytes (4.215438 GB)

time taken to initialize the array: 6.944266100 secs.

time taken to sort the array: 3.187436400 secs.

sum\_cpu time: 1.145248200 secs.

sum\_gpu1 time: 0.987274100 secs.

sum\_gpu2 time: 0.801436400 secs. (less warp divergence)

{sum\_cpu, sum\_gpu1, sum\_gpu2} **=** {9269397027886, 9269397027886, 9269397027886}