Final Year B. Tech, Sem VII 2022-23
PRN – 2020BTECS00211
Name – Aashita Narendra Gupta
High Performance Computing
Lab Batch: B4
SLIP No - 42
HPC LAB ESE

Github Link for Code - https://github.com/Aashita06/HPC_Practicals

Q.1) Write a program to demonstrate distributed sum of an array using MPI.
→

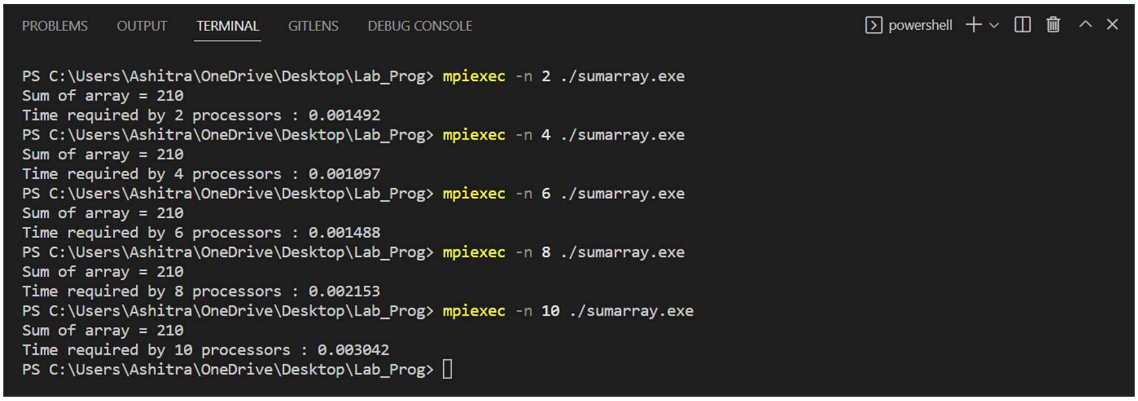
Code:

```
#include "mpi.h"
#include <stdio.h>

#define localSize 1000

// store the subarray data comming from process 0;
int local[1000];

int main(int argc, char **argv)
{
    int rank;
    int num;

    int n = 20;
    int arr[20] = {1, 2, 3, 4, 5, 6, 7, 8, 9,
10,11,12,13,14,15,16,17,18,19,20};

    int per_process, elements_received;

    MPI_Init(&argc, &argv);

    MPI_Comm_size(MPI_COMM_WORLD, &num);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);

    MPI_Status status;
```

```
    // process with rank 0 will divide data among all processes and add 
partial sums to get final sum
    if (rank == 0)
    {
        int index, i;

        per_process = n / num;

        double start = MPI_Wtime();

        // if more than 1 processes available
        if (num > 1)
        {
            //divide array data among processes
            for (i = 1; i < num - 1; i++)
            {
                //calculating first index of subarray that need to be send to 
ith process
                index = i * per_process;

                //send no of elements and subarray of that lenght to each 
process
                MPI_Send(&per_process, 1, MPI_INT, i, 0, MPI_COMM_WORLD);
                MPI_Send(&arr[index], per_process, MPI_INT, i, 0, 
MPI_COMM_WORLD);
            }

            // for last process send all remaining elements
            index = i * per_process;
            int ele_left = n - index;

            MPI_Send(&ele_left, 1, MPI_INT, i, 0, MPI_COMM_WORLD);
            MPI_Send(&arr[index], ele_left, MPI_INT, i, 0, MPI_COMM_WORLD);
        }

        // add numbers on process with rank 0
        int sum = 0;
        for (int i = 0; i < per_process; i++)
        {
            sum += arr[i];
        }

        // add all partial sums from all processes
        int tmp;
        for (int i = 1; i < num; i++)
        {
            MPI_Recv(&tmp, 1, MPI_INT, MPI_ANY_SOURCE, 0, MPI_COMM_WORLD, 
&status);
```

```
            int sender = status.MPI_SOURCE;

            sum += tmp;
        }

        printf("\nSum of array = %d\n", sum);

        double end = MPI_Wtime();
        printf("Time required by %d processors : %f",num,end-start);
    }
    else // if rank of process is not 0, then receive elements and calculate
partial sums
    {
        // receive no of elements and elements form process 0 and store them
on local array
        MPI_Recv(&elements_received, 1, MPI_INT, 0, 0, MPI_COMM_WORLD,
&status);

        MPI_Recv(&local, elements_received, MPI_INT, 0, 0, MPI_COMM_WORLD,
&status);

        // calculate partial local sum
        int partial_sum = 0;
        for (int i = 0; i < elements_received; i++)
        {
            partial_sum += local[i];
           // printf("\nPartial Sum of rank %d is %d ",rank,partial_sum);
        }

        //send calculated partial sum to process with rank 0
        MPI_Send(&partial_sum, 1, MPI_INT, 0, 0, MPI_COMM_WORLD);
    }

    MPI_Finalize();
    return 0;
}
```

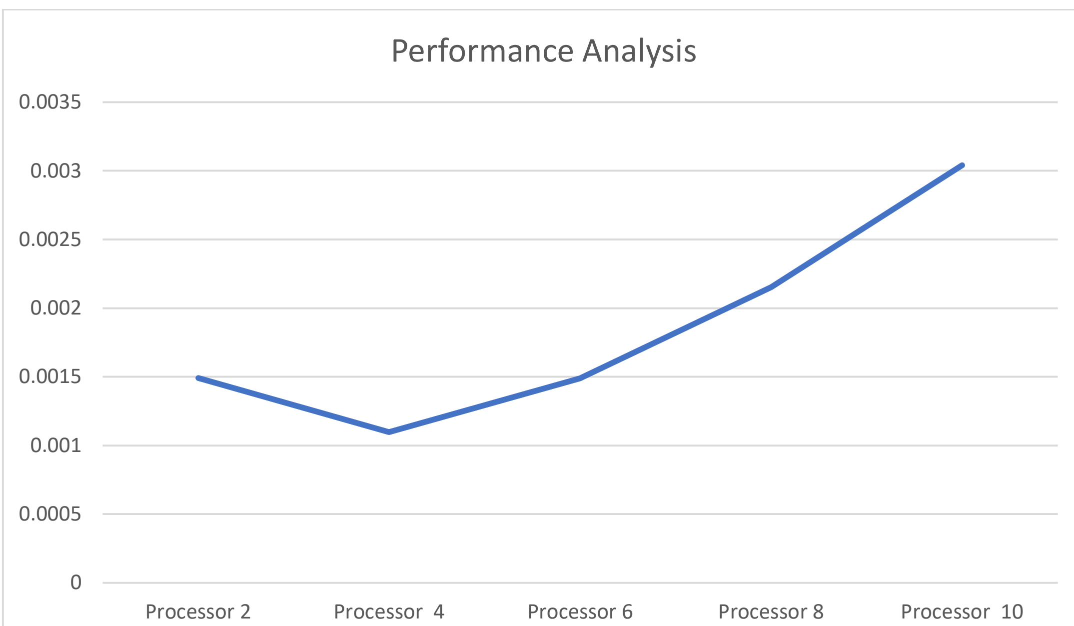
Output:



```
PROBLEMS   OUTPUT   TERMINAL   GITLENS   DEBUG CONSOLE                                  powershell
PS C:\Users\Ashitra\OneDrive\Desktop\Lab_Prog> mpiexec -n 2 ./sumarray.exe
Sum of array = 210
Time required by 2 processors : 0.001492
PS C:\Users\Ashitra\OneDrive\Desktop\Lab_Prog> mpiexec -n 4 ./sumarray.exe
Sum of array = 210
Time required by 4 processors : 0.001097
PS C:\Users\Ashitra\OneDrive\Desktop\Lab_Prog> mpiexec -n 6 ./sumarray.exe
Sum of array = 210
Time required by 6 processors : 0.001488
PS C:\Users\Ashitra\OneDrive\Desktop\Lab_Prog> mpiexec -n 8 ./sumarray.exe
Sum of array = 210
Time required by 8 processors : 0.002153
PS C:\Users\Ashitra\OneDrive\Desktop\Lab_Prog> mpiexec -n 10 ./sumarray.exe
Sum of array = 210
Time required by 10 processors : 0.003042
PS C:\Users\Ashitra\OneDrive\Desktop\Lab_Prog>
```

We are analysis the execution time taken by different number of processors to calculate the distributed sum of array. Here, due to communication overhead, time is increasing with increasing processors.

Processors	Time
2	0.001492
4	0.001097
6	0.001488
8	0.002153
10	0.003042



Q.2) Implement matrix matrix multiplication using CUDA (shared memory). (Conduct performance and speedup analysis of all programs)

→

Code:

```
%%writefile matrix_multi_share.cu
#include <stdio.h>
#define row1 20
#define col1 30
#define row2 30
#define col2 20

__global__ void matproductsharedmemory(int *l, int *m, int *n)
{
    int x = blockIdx.x;
    int y = blockIdx.y;
    __shared__ int p[col1];

    int i;
    int k = threadIdx.x;

    n[col2 * y + x] = 0;

    p[k] = l[col1 * y + k] * m[col2 * k + x];

    __syncthreads();

    for (i = 0; i < col1; i++)
        n[col2 * y + x] = n[col2 * y + x] + p[i];
}

int main()
{
    int a[row1][col1];
    int b[row2][col2];
```

```
    int c[row1][col2];
    int *d, *e, *f;
    int i, j;

    for (i = 0; i < row1; i++)
    {
        for (j = 0; j < col1; j++)
        {
            a[i][j] = 2;
        }
    }

    for (i = 0; i < row2; i++)
    {
        for (j = 0; j < col2; j++)
        {
            b[i][j] = 3;
        }
    }

    cudaMalloc((void **)&d, row1 * col1 * sizeof(int));
    cudaMalloc((void **)&e, row2 * col2 * sizeof(int));
    cudaMalloc((void **)&f, row1 * col2 * sizeof(int));

    cudaMemcpy(d, a, row1 * col1 * sizeof(int), cudaMemcpyHostToDevice);
    cudaMemcpy(e, b, row2 * col2 * sizeof(int), cudaMemcpyHostToDevice);

    dim3 grid(col2, row1);

    matproductsharedmemory<<<grid, col1>>>(d, e, f);
    cudaDeviceSynchronize();
```

```
    cudaMemcpy(c, f, row1 * col2 * sizeof(int), cudaMemcpyDeviceToHost);

    for (i = 0; i < row1; i++)
    {
        for (j = 0; j < col2; j++)
        {
            if (c[i][j] != 180)
            {
                printf("False\n");
                return -1;
            }
        }
    }

    cudaFree(d);
    cudaFree(e);
    cudaFree(f);

    printf("True\n");
    return 0;
}
```

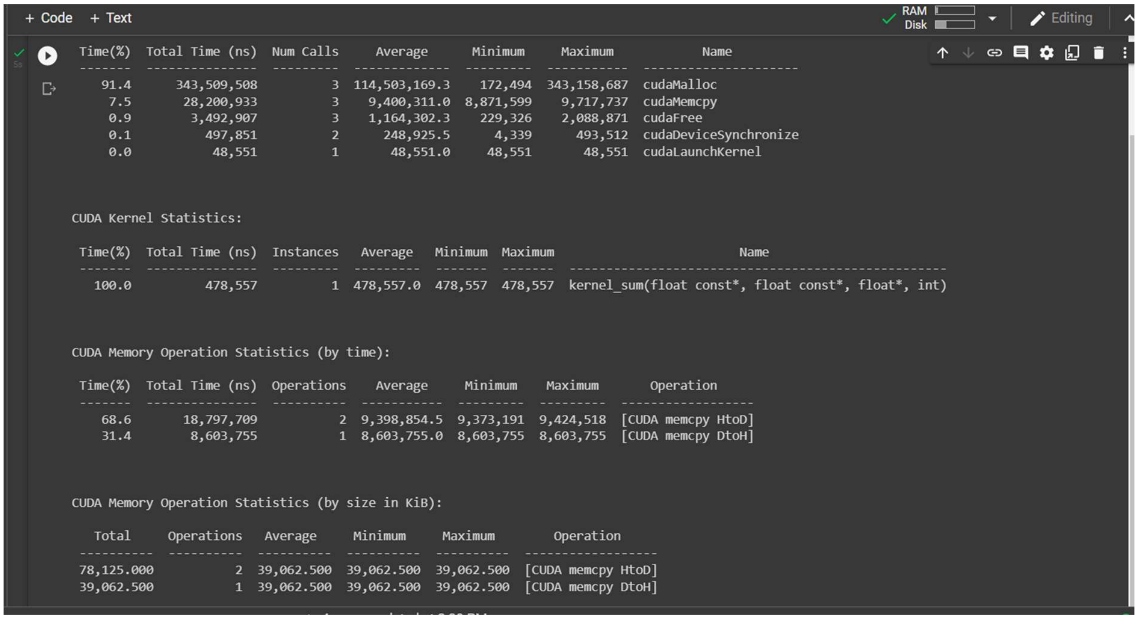
Output:

```
Writing matrix_multi_share.cu

!nvcc -o matrix_multi_share matrix_multi_share.cu
!./matrix_multi_share

True
```

Profiling of matrix matrix multiplication program in CUDA (Shared Memory).

```
True

!nsys profile --stats=true matrixshare

Warning: LBR backtrace method is not supported on this platform. DWARF backtrace method will be used.
Collecting data...
True
The target application terminated with signal 11 (SIGSEGV)
Processing events...
Capturing symbol files...
Saving temporary "/tmp/nsys-report-b443-e0d0-593d-c942.qdstrm" file to disk...
Creating final output files...

Processing [==============================================================100%]
Saved report file to "/tmp/nsys-report-b443-e0d0-593d-c942.qdrep"
Exporting 11741 events: [=============================================100%]

Exported successfully to
/tmp/nsys-report-b443-e0d0-593d-c942.sqlite


CUDA API Statistics:

 Time(%)  Total Time (ns)  Num Calls    Average     Minimum    Maximum           Name
 -------  ---------------  ---------  ------------  ---------  -----------  ---------------------
    91.4      343,509,508          3  114,503,169.3    172,494  343,158,687  cudaMalloc
     7.5       28,200,933          3    9,400,311.0  8,871,599    9,717,737  cudaMemcpy
     0.9        3,492,907          3    1,164,302.3    229,326    2,088,871  cudaFree
     0.1          497,851          2      248,925.5      4,339      493,512  cudaDeviceSynchronize
     0.0           48,551          1       48,551.0     48,551       48,551  cudaLaunchKernel


CUDA Kernel Statistics:
```

```
+ Code  + Text                                                    RAM Disk    Editing

CUDA Kernel Statistics:

 Time(%)  Total Time (ns)  Instances   Average   Minimum  Maximum                      Name
 -------  ---------------  ---------  ---------  -------  -------  ----------------------------------------------------
   100.0          478,557          1  478,557.0  478,557  478,557  kernel_sum(float const*, float const*, float*, int)



CUDA Memory Operation Statistics (by time):

 Time(%)  Total Time (ns)  Operations    Average     Minimum    Maximum       Operation
 -------  ---------------  ----------  -----------  ---------  ---------  ------------------
    68.6       18,797,709           2  9,398,854.5  9,373,191  9,424,518  [CUDA memcpy HtoD]
    31.4        8,603,755           1  8,603,755.0  8,603,755  8,603,755  [CUDA memcpy DtoH]



CUDA Memory Operation Statistics (by size in KiB):

   Total     Operations   Average     Minimum     Maximum       Operation
 ----------  ----------  ----------  ----------  ----------  ------------------
 78,125.000           2  39,062.500  39,062.500  39,062.500  [CUDA memcpy HtoD]
 39,062.500           1  39,062.500  39,062.500  39,062.500  [CUDA memcpy DtoH]
```



Time(%)	Total Time (ns)	Num Calls	Average	Minimum	Maximum	Name
91.4	343,509,508	3	114,503,169.3	172,494	343,158,687	cudaMalloc
7.5	28,200,933	3	9,400,311.0	8,871,599	9,717,737	cudaMemcpy
0.9	3,492,907	3	1,164,302.3	229,326	2,088,871	cudaFree
0.1	497,851	2	248,925.5	4,339	493,512	cudaDeviceSynchronize
0.0	48,551	1	48,551.0	48,551	48,551	cudaLaunchKernel

CUDA Kernel Statistics:

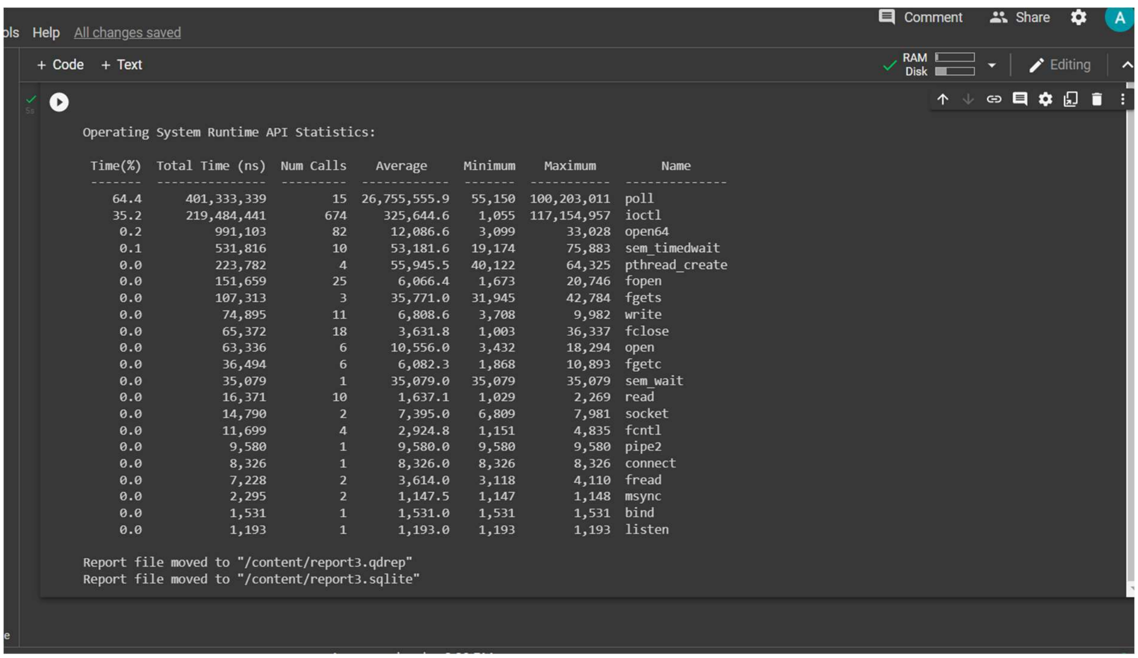
Time(%)	Total Time (ns)	Instances	Average	Minimum	Maximum	Name
100.0	478,557	1	478,557.0	478,557	478,557	kernel_sum(float const*, float const*, float*, int)

CUDA Memory Operation Statistics (by time):

Time(%)	Total Time (ns)	Operations	Average	Minimum	Maximum	Operation
68.6	18,797,709	2	9,398,854.5	9,373,191	9,424,518	[CUDA memcpy HtoD]
31.4	8,603,755	1	8,603,755.0	8,603,755	8,603,755	[CUDA memcpy DtoH]

CUDA Memory Operation Statistics (by size in KiB):

Total	Operations	Average	Minimum	Maximum	Operation
78,125.000	2	39,062.500	39,062.500	39,062.500	[CUDA memcpy HtoD]
39,062.500	1	39,062.500	39,062.500	39,062.500	[CUDA memcpy DtoH]



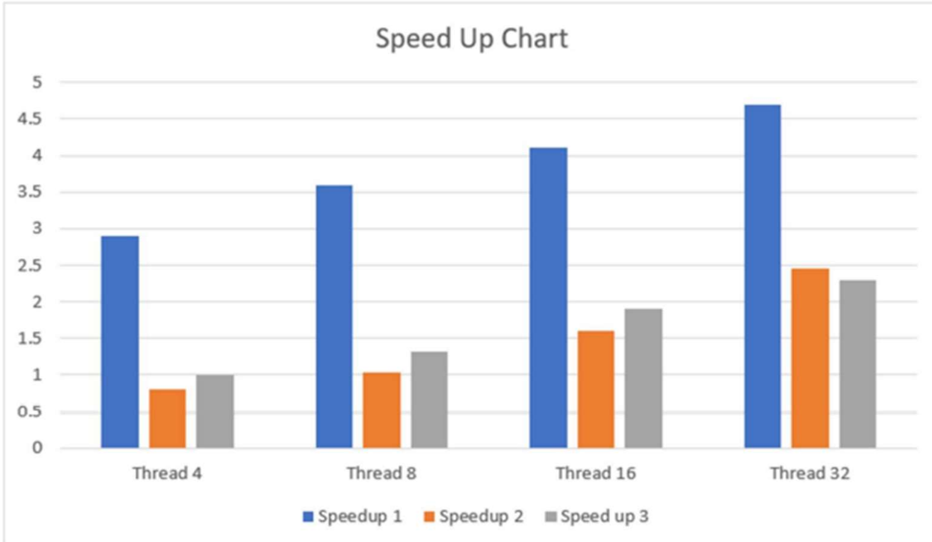
Operating System Runtime API Statistics:

Time(%)	Total Time (ns)	Num Calls	Average	Minimum	Maximum	Name
64.4	401,333,339	15	26,755,555.9	55,150	100,203,011	poll
35.2	219,484,441	674	325,644.6	1,055	117,154,957	ioctl
0.2	991,103	82	12,086.6	3,099	33,028	open64
0.1	531,816	10	53,181.6	19,174	75,883	sem_timedwait
0.0	223,782	4	55,945.5	40,122	64,325	pthread_create
0.0	151,659	25	6,066.4	1,673	20,746	fopen
0.0	107,313	3	35,771.0	31,945	42,784	fgets
0.0	74,895	11	6,808.6	3,708	9,982	write
0.0	65,372	18	3,631.8	1,003	36,337	fclose
0.0	63,336	6	10,556.0	3,432	18,294	open
0.0	36,494	6	6,082.3	1,868	10,893	fgetc
0.0	35,079	1	35,079.0	35,079	35,079	sem_wait
0.0	16,371	10	1,637.1	1,029	2,269	read
0.0	14,790	2	7,395.0	6,809	7,981	socket
0.0	11,699	4	2,924.8	1,151	4,835	fcntl
0.0	9,580	1	9,580.0	9,580	9,580	pipe2
0.0	8,326	1	8,326.0	8,326	8,326	connect
0.0	7,228	2	3,614.0	3,118	4,110	fread
0.0	2,295	2	1,147.5	1,147	1,148	msync
0.0	1,531	1	1,531.0	1,531	1,531	bind
0.0	1,193	1	1,193.0	1,193	1,193	listen

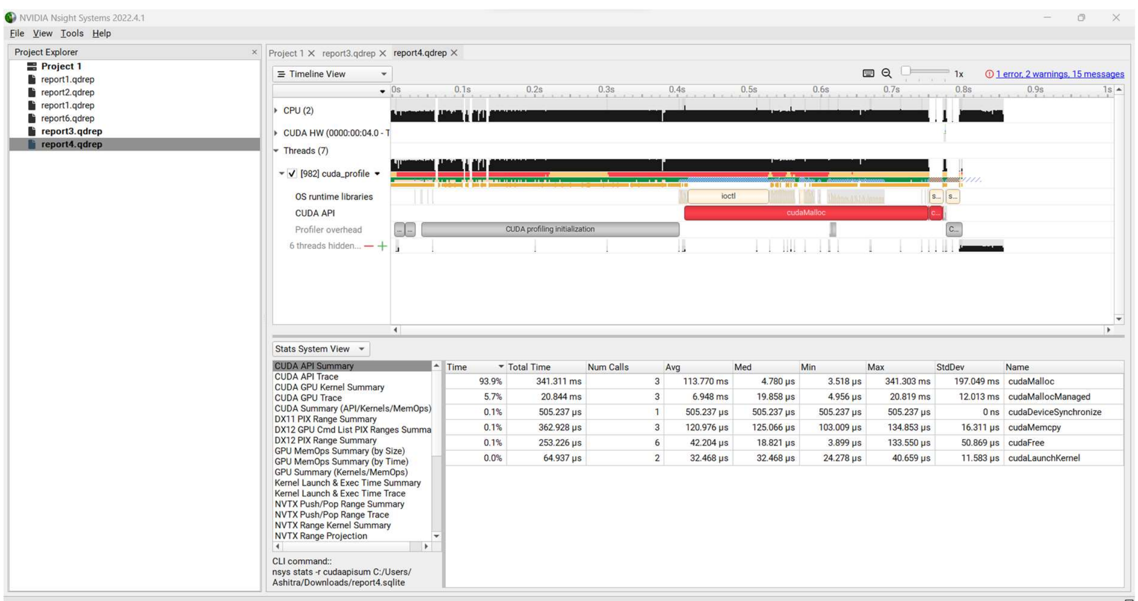
Report file moved to "/content/report3.qdrep"
Report file moved to "/content/report3.sqlite"

Keeping No. of blocks constants and increasing no. of threads, we are calculating speedup of the following pmatrix multiplication program.

Speedup	No.of threads
3.6, 1.04, 1.0,1.31	8
3.7, 1.32, 1.2, 1.51	16
4.23, 2.5, 2.1, 2.7	32



Speed Up Chart
5
4.5
4
3.5
3
2.5
2
1.5
1
0.5
0
Thread 4
Thread 8
Thread 16
Thread 32
Speedup 1
Speedup 2
Speed up 3



NVIDIA Nsight Systems 2022.4.1
File View Tools Help
Project Explorer
Project 1
report1.qdrep
report2.qdrep
report1.qdrep
report6.qdrep
report3.qdrep
report4.qdrep
Project 1
report3.qdrep
report4.qdrep
Timeline View
1x
1 error, 2 warnings, 15 messages
CPU (2)
Threads (7)
[982] cuda_profile
OS runtime libraries
CUDA API
Profiler overhead
6 threads hidden...
ioctl
cudaMalloc
CUDA profiling initialization
Stats System View
CUDA API Summary
CUDA API Trace
CUDA GPU Kernel Summary
CUDA GPU Trace
CUDA Summary (API/Kernels/MemOps)
DX11 PIX Range Summary
DX12 GPU Cmd List PIX Ranges Summa
DX12 PIX Range Summary
GPU MemOps Summary (by Size)
GPU MemOps Summary (by Time)
GPU Summary (Kernels/MemOps)
Kernel Launch & Exec Time Summary
Kernel Launch & Exec Time Trace
NVTX Push/Pop Range Summary
NVTX Push/Pop Range Trace
NVTX Range Kernel Summary
NVTX Range Projection
Time Total Time Num Calls Avg Med Min Max StdDev Name
93.9% 341.311 ms 3 113.770 ms 4.780 μs 3.518 μs 341.303 ms 197.049 ms cudaMalloc
5.7% 20.844 ms 3 6.948 ms 19.858 μs 4.956 μs 20.819 ms 12.013 ms cudaMallocManaged
0.1% 505.237 μs 1 505.237 μs 505.237 μs 505.237 μs 505.237 μs 0 ns cudaDeviceSynchronize
0.1% 362.928 μs 3 120.976 μs 125.066 μs 103.009 μs 134.853 μs 16.311 μs cudaMemcpy
0.1% 253.226 μs 6 42.204 μs 18.821 μs 3.899 μs 133.550 μs 50.869 μs cudaFree
0.0% 64.937 μs 2 32.468 μs 32.468 μs 24.278 μs 40.659 μs 11.583 μs cudaLaunchKernel
CLI command::
nsys stats -r cudaapisum C:/Users/Ashitra/Downloads/report4.sqlite