Assignment 4

Name: Ranjan Baro ID: 2023PCP5274

Program 1: Hello GPU

```
%%writefile hello.cu
#include <stdio.h>
__global__ void hello() {
  printf("Hello, I am GPU.\n");
}
int main() {
  printf("Running Kernel...\n");
  hello<<<1,1>>>();
  cudaDeviceSynchronize();
  printf("Hello, I am CPU\n");
  return 0;
}
```

```
Overwriting hello.cu
```

```
[8] !nvcc hello.cu -o hello
```

```
[9] !./hello
```

```
Running Kernel...
Hello, I am GPU.
Hello, I am CPU
```

Program 2: Vector Addition

```
%%writefile vector_add.cu
#include<iostream>
#define N (4*4)
#define THREADS_PER_BLOCK 4
using namespace std;
__global__ void addition(int *a, int *b, int *c, int *n) {
    int index = threadIdx.x + blockIdx.x * blockDim.x;
    printf("(%d, %d)", index, *n);
    if(index < *n) {
      c[index] = a[index] + b[index];
    }
}
int main() {
    int *a, *b, *c, *d_a, *d_b, *d_c, *d_n;
    int size_int = sizeof(int); int size = N * sizeof(int); int n = N;
    cudaMalloc((void **)&d_a, size); cudaMalloc((void **)&d_b, size); cudaMalloc((void **)&d_c, size); cudaMalloc((void **)&d_n, size_int);
    a = (int *)malloc(size); b = (int *)malloc(size); c = (int *)malloc(size);
    for(int i = 0; i < N; i++) {
      a[i] = rand() % N; b[i] = rand() % N;
    }
    cudaMemcpy(d_a, a, size, cudaMemcpyHostToDevice); cudaMemcpy(d_b, b, size, cudaMemcpyHostToDevice);
    cudaMemcpy(d_n, &n, size_int, cudaMemcpyHostToDevice);
    addition<<<N/THREADS_PER_BLOCK, THREADS_PER_BLOCK>>>(d_a, d_b, d_c, d_n);
    cudaDeviceSynchronize(); cudaMemcpy(c, d_c, size, cudaMemcpyDeviceToHost); printf("Additon of first 10 elements : \n");
    for(int i = 0; i < 10; i++) {
      printf("(%d + %d = %d), ", a[i], b[i], (c[i]));
    }
    free(a); free(b); free(c); cudaFree(a); cudaFree(b); cudaFree(c);
    return 0;
}
```

Output:

```
Overwriting vector_add.cu
```

```
!nvcc vector_add.cu -o add
!nvprof ./add
```

```
==22556== NVPROF is profiling process 22556, command: ./add
(8, 16)(9, 16)(10, 16)(11, 16)(0, 16)(1, 16)(2, 16)(3, 16)(4, 16)(5, 16)(6, 16)(7, 16)(12, 16)(13, 16)(14, 16)(15, 16)Additon of first 10 elements :
==22556== Profiling application: ./add
==22556== Profiling result:
            Type  Time(%)      Time     Calls       Avg       Min       Max  Name
 GPU activities:   91.29%  72.735us         1  72.735us  72.735us  72.735us  addition(int*, int*, int*, int*)
                    5.94%  4.7360us         3  1.5780us  1.3760us  1.9840us  [CUDA memcpy HtoD]
                    2.77%  2.2080us         1  2.2080us  2.2080us  2.2080us  [CUDA memcpy DtoH]
      API calls:   99.69%  132.97ms         4  33.243ms  5.0810us  132.95ms  cudaMalloc
                    0.09%  114.23us       101  1.1310us     130ns  47.118us  cuDeviceGetAttribute
                    0.08%  113.02us         1  113.02us  113.02us  113.02us  cudaDeviceSynchronize
                    0.06%  75.311us         4  18.827us  9.1340us  29.084us  cudaMemcpy
                    0.05%  63.629us         1  63.629us  63.629us  63.629us  cudaLaunchKernel
                    0.02%  28.384us         1  28.384us  28.384us  28.384us  cuDeviceGetName
                    0.00%  5.8050us         1  5.8050us  5.8050us  5.8050us  cuDeviceGetPCIBusId
                    0.00%  4.6300us         3  1.5430us     880ns  2.7270us  cudaFree
                    0.00%  1.6130us         3     537ns     221ns  1.0870us  cuDeviceGetCount
                    0.00%  1.1000us         2     550ns     312ns     788ns  cuDeviceGet
                    0.00%     437ns         1     437ns     437ns     437ns  cuDeviceTotalMem
                    0.00%     432ns         1     432ns     432ns     432ns  cuModuleGetLoadingMode
                    0.00%     238ns         1     238ns     238ns     238ns  cuDeviceGetUuid
(7 + 6 = 13), (9 + 3 = 12), (1 + 15 = 16), (10 + 12 = 22), (9 + 13 = 22), (10 + 11 = 21), (2 + 11 = 13), (3 + 6 = 9), (12 + 2 = 14), (4 + 8 = 12),
```

Program 3: Matrix Multiplication

```
%%writefile matrix_multiplication.cu
#include <stdio.h>

__global__ void matrixMultKernel(int *a, int *b, int *c, int w) {
  int row = threadIdx.y + blockIdx.y * blockDim.y;
  int col = threadIdx.x + blockIdx.x * blockDim.y;
  int sum = 0;
  for(int k = 0; k < w; k++) {
    sum += a[row * w + k] * b[k * w + col];
  }
  c[row * w + col] = sum;
}

void matrixMult(int *a, int *b, int *c, int w) {
  int *d_a, *d_b, *d_c;
  int size = w * w * sizeof(int);
  cudaMalloc((void **) &d_a, size);
  cudaMalloc((void **) &d_b, size);
  cudaMalloc((void **) &d_c, size);
  cudaMemcpy(d_a, a, size, cudaMemcpyHostToDevice);
  cudaMemcpy(d_b, b, size, cudaMemcpyHostToDevice);
  dim3 blockDim(16, 16);
  dim3 gridDim((w + blockDim.x - 1) / blockDim.x, (w + blockDim.y - 1) / blockDim.y);
  matrixMultKernel<<<gridDim,blockDim>>>(d_a, d_b, d_c, w);
  cudaDeviceSynchronize();
  cudaMemcpy(c, d_c, size, cudaMemcpyDeviceToHost);
  cudaFree(d_a); cudaFree(d_b); cudaFree(d_c);
}
```

```
int main() {
  const int w = 4;
  int a[w][w], b[w][w], c[w][w];
  for(int i = 0; i < w; i++) {
    for(int j = 0; j < w; j++) {
      a[i][j] = rand() % 50;
      b[i][j] = rand() % 50;
    }
  }
  matrixMult((int *)a, (int *)b, (int *)c, w);
  printf("Martix A :\n");
  for(int i = 0; i < w; i++) {
    for(int j = 0; j < w; j++) {
      printf("%d\t", a[i][j]);
    }
    printf("\n");
  }
  printf("\nMatrix B :\n");
  for(int i = 0; i < w; i++) {
    for(int j = 0; j < w; j++) {
      printf("%d\t", b[i][j]);
    }
    printf("\n");
  }
  printf("Matrix C (A * B) :\n");
  for(int i = 0; i < w; i++) {
    for(int j = 0; j < w; j++) {
      printf("%d\t", c[i][j]);
    }
    printf("\n");
  }
  return 0;
}
```

Output:

matrix_mult.ipynb

File Edit View Insert Runtime Tools Help All changes saved

Comment Share

+ Code + Text

```
!nvprof ./matrix
```

[52]

```
==16011== NVPROF is profiling process 16011, command: ./matrix
Martix A :
33      27      43      36
49      12      40      13
40      22      11      17
32      12      17      29

Matrix B :
36      15      35      42
21      27      9       26
26      36      18      29
30      23      35      2
Matrix C (A * B) :
3953    3600    3432    3407
2685    2852    2288    1727
2698    1981    2391    2605
1742    2125    1141    1700
==16011== Profiling application: ./matrix
==16011== Profiling result:
            Type  Time(%)      Time     Calls       Avg       Min       Max  Name
 GPU activities:   51.67%  5.4400us         1  5.4400us  5.4400us  5.4400us  matrixMultKernel(int*, int*, int*, int)
                   28.57%  3.0080us         2  1.5040us  1.3440us  1.6640us  [CUDA memcpy HtoD]
                   19.76%  2.0800us         1  2.0800us  2.0800us  2.0800us  [CUDA memcpy DtoH]
      API calls:   99.79%  152.61ms         3  50.870ms  3.3920us  152.60ms  cudaMalloc
                    0.07%  111.27us       101  1.1010us     136ns  46.017us  cuDeviceGetAttribute
                    0.06%  89.319us         3  29.773us  3.3860us  79.581us  cudaFree
                    0.03%  46.282us         3  15.427us  7.5240us  19.669us  cudaMemcpy
                    0.02%  27.000us         1  27.000us  27.000us  27.000us  cudaLaunchKernel
                    0.01%  22.659us         1  22.659us  22.659us  22.659us  cuDeviceGetName
                    0.01%  19.633us         1  19.633us  19.633us  19.633us  cuDeviceGetPCIBusId
                    0.01%  8.6320us         1  8.6320us  8.6320us  8.6320us  cudaDeviceSynchronize
                    0.00%  1.8400us         3     613ns     224ns  1.3910us  cuDeviceGetCount
                    0.00%     934ns         2     467ns     152ns     782ns  cuDeviceGet
                    0.00%     555ns         1     555ns     555ns     555ns  cuModuleGetLoadingMode
                    0.00%     425ns         1     425ns     425ns     425ns  cuDeviceTotalMem
```

1s completed at 7:30 PM

Summary Report On CUDA:

CUDA: CUDA, which stands for Compute Unified Device Architecture, is a parallel computing platform and programming model that makes using a GPU for general purpose computing simple.

- **Keywords:**
  - **Host** – CPU
  - **Device** – GPU
  - **Host Memory** – System memory (DRAM) associated with host
  - **Device Memory** – GPU memory
  - **Kernel** – Function executed on GPU by single thread

Key components of CUDA:

a) **GPU Utilization:** CUDA leverages the parallel processing capabilities of NVIDIA GPUs, allowing developers to perform general-purpose computations in parallel, leading to significant speedups.

b) **CUDA Toolkit:** NVIDIA provides a software development kit called the CUDA Toolkit, which includes the necessary tools, libraries, and programming environments for GPU development.

c) **Programming Languages:** CUDA supports programming in CUDA C and CUDA C++, extensions of the C and C++ languages, enabling developers to write parallel programs explicitly controlling GPU execution.

d) **Runtime API:** The CUDA Runtime API provides a set of functions that C/C++ programs can call to manage and control the execution of GPU kernels.

e) **Parallel Architecture:** NVIDIA GPUs are designed with multiple cores, allowing the simultaneous execution of numerous parallel threads. CUDA takes advantage of this architecture to accelerate a wide range of applications, including scientific simulations, deep learning, and image processing.

CUDA Memory

a) **Global Memory:**
   - Global memory is the largest and slowest memory in the CUDA hierarchy.
   - It is accessible by all threads on the GPU and is used to store global variables.

b) **Shared Memory:**
   - Shared by all threads in a thread block
   - Faster and small
   - used for communication and data sharing between threads within the same block.

c) **Local Memory:**
   - slower than registers and is typically used when a thread's register usage is high.
   - Per thread private memory

d) **Texture Memory:**
   - a read-only cache designed for texture fetching.
   - optimized for 2D spatial locality.
   - can be beneficial for certain types of memory access patterns.