BIRLA INSTITUTE OF TECHNOLOGY & SCIENCE, PILANI (RAJASTHAN)

CS F422 – Parallel Computing

Lab#4

Note: Please use programs under *Code_lab4* directory supplied with this sheet. Do not copy from this sheet.

The lab has the following objectives:
Giving practice programs for CUDA.

Query device features

```
1. #include "./book.h"
2.
3. int main(void)
4. {
5.     cudaDeviceProp prop;
6.     int count;
7.     HANDLE_ERROR(cudaGetDeviceCount(&count));
8.     for (int i = 0; i < count; i++)
9.     {
10.             HANDLE_ERROR(cudaGetDeviceProperties(&prop, i));
11.             printf(" --- General Information for device %d ---\n", i);
12.             printf("Name: %s\n", prop.name);
13.             printf("Compute capability: %d.%d\n", prop.major,
   prop.minor);
14.             printf("Clock rate: %d\n", prop.clockRate);
15.             printf("Device copy overlap: ");
16.             if (prop.deviceOverlap)
17.                 printf("Enabled\n");
18.             else
19.                 printf("Disabled\n");
20.             printf("Kernel execition timeout : ");
21.             if (prop.kernelExecTimeoutEnabled)
22.                 printf("Enabled\n");
23.             else
24.                 printf("Disabled\n");
25.             printf(" --- Memory Information for device %d ---\n", i);
26.             printf("Total global mem: %ld\n", prop.totalGlobalMem);
27.             printf("Total constant Mem: %ld\n", prop.totalConstMem);
28.             printf("Max mem pitch: %ld\n", prop.memPitch);
29.             printf("Texture Alignment: %ld\n", prop.textureAlignment);
30.             printf(" --- MP Information for device %d ---\n", i);
31.             printf("Multiprocessor count: %d\n",
32.                    prop.multiProcessorCount);
33.             printf("Shared mem per mp: %ld\n", prop.sharedMemPerBlock);
34.             printf("Registers per mp: %d\n", prop.regsPerBlock);
35.             printf("Threads in warp: %d\n", prop.warpSize);
```

```
36.             printf("Max threads per block: %d\n",
37.                    prop.maxThreadsPerBlock);
38.             printf("Max thread dimensions: (%d, %d, %d)\n",
39.                    prop.maxThreadsDim[0], prop.maxThreadsDim[1],
40.                    prop.maxThreadsDim[2]);
41.             printf("Max grid dimensions: (%d, %d, %d)\n",
42.                    prop.maxGridSize[0], prop.maxGridSize[1],
43.                    prop.maxGridSize[2]);
44.             printf("\n");
45.         }
46.     }
```

Q?

1. This is a program to query the device features in your GPU. See if you can understand most of what they mean, and if they match your device configuration.

Using parameters to kernel

```
1. #include <iostream>
2. #include "book.h"
3.
4. __global__ void add(int a, int b, int *c)
5. {
6.     *c = a + b;
7. }
8.
9. int main(void)
10.     {
11.         int c;
12.         int *dev_c;
13.         HANDLE_ERROR(cudaMalloc((void **)&dev_c, sizeof(int)));
14.         add<<<1, 1>>>(2, 7, dev_c);
15.         HANDLE_ERROR(cudaMemcpy(&c,
16.                                 dev_c,
17.                                 sizeof(int),
18.                                 cudaMemcpyDeviceToHost));
19.         printf("2 + 7 = %d\n", c);
20.         cudaFree(dev_c);
21.         return 0;
22.     }
```

Q?

1. This is a program to teach passing of parameters to the kernel.

GPU Vector sum using parallel blocks

```
1. #include "./book.h"
2. #define N 10
3.
4. int main(void)
5. {
6.     int a[N], b[N], c[N];
7.     int *dev_a, *dev_b, *dev_c;
8.     // allocate the memory on the GPU
9.     HANDLE_ERROR(cudaMalloc((void **)&dev_a, N * sizeof(int)));
10.          HANDLE_ERROR(cudaMalloc((void **)&dev_b, N * sizeof(int)));
11.          HANDLE_ERROR(cudaMalloc((void **)&dev_c, N * sizeof(int)));
12.          // fill the arrays 'a' and 'b' on the CPU
13.          for (int i = 0; i < N; i++)
14.          {
15.              a[i] = -i;
16.              b[i] = i * i;
17.          }
18.
19.          // copy the arrays 'a' and 'b' to the GPU
20.          HANDLE_ERROR(cudaMemcpy(dev_a, a, N * sizeof(int),
   cudaMemcpyHostToDevice));
21.          HANDLE_ERROR(cudaMemcpy(dev_a, a, N * sizeof(int),
   cudaMemcpyHostToDevice));
22.
23.          add<<<N,1>>>(dev_a, dev_b, dev_c);
24.
25.          // copy array 'c' back from the GPU to the CPU
26.          HANDLE_ERROR(cudaMemcpy(c, dev_c, N * sizeof(int),
   cudaMemcpyHostToDevice));
27.
28.          // display the results
29.          for(int i=0; i<N; ++i){
30.              printf("%d + %d = %d\n", a[i], b[i], c[i]);
31.          }
32.
33.          // free the memory allocated on the GPU
34.          cudaFree(dev_a);
35.          cudaFree(dev_b);
36.          cudaFree(dev_c);
37.
38.          return 0;
39.      }
```

Q?

1. This is a program which uses GPU to compute the vector sum using a parallel block implementation.

GPU Vector sum using threads

```
1. #include "../common/book.h"
2. #define N 10
3. __global__ void add(int *a, int *b, int *c)
4. {
5.     int tid = threadIdx.x;
6.     if (tid < N)
7.         c[tid] = a[tid] + b[tid];
8. }
9. int main(void)
10.      {
11.          int a[N], b[N], c[N];
12.          int *dev_a, *dev_b, *dev_c;
13.          // allocate the memory on the GPU
14.          HANDLE_ERROR(cudaMalloc((void **)&dev_a, N * sizeof(int)));
15.          HANDLE_ERROR(cudaMalloc((void **)&dev_b, N * sizeof(int)));
16.          HANDLE_ERROR(cudaMalloc((void **)&dev_c, N * sizeof(int)));
17.          // fill the arrays "a" and "b" on the CPU
18.          for (int i = 0; i < N; i++)
19.          {
20.              a[i] = i;
21.              b[i] = i * i;
22.          }
23.          // copy the arrays "a" and "b" to the GPU
24.          HANDLE_ERROR(cudaMemcpy(dev_a,
25.                                  a,
26.                                  N * sizeof(int),
27.                                  cudaMemcpyHostToDevice));
28.          HANDLE_ERROR(cudaMemcpy(dev_b,
29.                                  b,
30.                                  N * sizeof(int),
31.                                  cudaMemcpyHostToDevice));
32.          add<<<1, N>>>(dev_a, dev_b, dev_c);
33.          // copy the array "c" back from the GPU to the CPU
34.          HANDLE_ERROR(cudaMemcpy(c,
35.                                  dev_c,
36.                                  N * sizeof(int),
37.                                  cudaMemcpyDeviceToHost));
38.          // display the results
39.          for (int i = 0; i < N; i++)
40.          {
41.              printf( "% d + % d = % d\n", a[i], b[i], c[i]);
42.          }
43.          // free the memory allocated on the GPU
44.          cudaFree(dev_a);
45.          cudaFree(dev_b);
46.          cudaFree(dev_c);
47.          return 0;
48.      }
```

Q?

1. This is a program which uses parallel threads instead of parallel block to compute vector sum using GPU. Can you understand the difference b/w the two programs?

End of lab4