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

```
#include "./book.h"

int main(void)
{
    cudaDeviceProp prop;
    int count;
    HANDLE_ERROR(cudaGetDeviceCount(&count));
    for (int i = 0; i < count; i++)
    {
              HANDLE_ERROR(cudaGetDeviceProperties(&prop, i));
              printf(" --- General Information for device %d ---\n", i);
              printf("Name: %s\n", prop.name);
              printf("Compute capability: %d.%d\n", prop.major,
prop.minor);
              printf("Clock rate: %d\n", prop.clockRate);
              printf("Device copy overlap: ");
              if (prop.deviceOverlap)
                  printf("Enabled\n");
              else
                  printf("Disabled\n");
              printf("Kernel execition timeout : ");
              if (prop.kernelExecTimeoutEnabled)
                  printf("Enabled\n");
              else
                  printf("Disabled\n");
              printf(" --- Memory Information for device %d ---\n", i);
              printf("Total global mem: %ld\n", prop.totalGlobalMem);
              printf("Total constant Mem: %ld\n", prop.totalConstMem);
              printf("Max mem pitch: %ld\n", prop.memPitch);
              printf("Texture Alignment: %ld\n", prop.textureAlignment);
              printf(" --- MP Information for device %d ---\n", i);
              printf("Multiprocessor count: %d\n",
                     prop.multiProcessorCount);
              printf("Shared mem per mp: %ld\n", prop.sharedMemPerBlock);
              printf("Registers per mp: %d\n", prop.regsPerBlock);
              printf("Threads in warp: %d\n", prop.warpSize);
```

```
      printf("Max threads per block: %d\n",
          prop.maxThreadsPerBlock);
      printf("Max thread dimensions: (%d, %d, %d)\n",
          prop.maxThreadsDim[0], prop.maxThreadsDim[1],
          prop.maxThreadsDim[2]);
      printf("Max grid dimensions: (%d, %d, %d)\n",
          prop.maxGridSize[0], prop.maxGridSize[1],
          prop.maxGridSize[2]);
      printf("\n");
    }
  }
```

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

```
#include <iostream>
#include "book.h"

__global__ void add(int a, int b, int *c)
{
  *c = a + b;
}

int main(void)
  {
    int c;
    int *dev_c;
    HANDLE_ERROR(cudaMalloc((void **)&dev_c, sizeof(int)));
    add<<<1, 1>>>(2, 7, dev_c);
    HANDLE_ERROR(cudaMemcpy(&c,
                            dev_c,
                            sizeof(int),
                            cudaMemcpyDeviceToHost));
    printf("2 + 7 = %d\n", c);
    cudaFree(dev_c);
    return 0;
  }
```

Q?

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

GPU Vector sum using parallel blocks

```
#include "./book.h"
#define N 10

int main(void)
{
    int a[N], b[N], c[N];
    int *dev_a, *dev_b, *dev_c;
    // allocate the memory on the GPU
    HANDLE_ERROR(cudaMalloc((void **)&dev_a, N * sizeof(int)));
          HANDLE_ERROR(cudaMalloc((void **)&dev_b, N * sizeof(int)));
          HANDLE_ERROR(cudaMalloc((void **)&dev_c, N * sizeof(int)));
          // fill the arrays 'a' and 'b' on the CPU
          for (int i = 0; i < N; i++)
          {
              a[i] = -i;
              b[i] = i * i;
          }

          // copy the arrays 'a' and 'b' to the GPU
          HANDLE_ERROR(cudaMemcpy(dev_a, a, N * sizeof(int), cudaMemcpyHostToDevice));
          HANDLE_ERROR(cudaMemcpy(dev_a, a, N * sizeof(int), cudaMemcpyHostToDevice));

          add<<<N,1>>>(dev_a, dev_b, dev_c);

          // copy array 'c' back from the GPU to the CPU
          HANDLE_ERROR(cudaMemcpy(c, dev_c, N * sizeof(int), cudaMemcpyHostToDevice));

          // display the results
          for(int i=0; i<N; ++i){
              printf("%d + %d = %d\n", a[i], b[i], c[i]);
          }

          // free the memory allocated on the GPU
          cudaFree(dev_a);
          cudaFree(dev_b);
          cudaFree(dev_c);

          return 0;
      }
```

Q?

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

GPU Vector sum using threads

```
#include "../common/book.h"
#define N 10
__global__ void add(int *a, int *b, int *c)
{
    int tid = threadIdx.x;
    if (tid < N)
        c[tid] = a[tid] + b[tid];
}
int main(void)
      {
          int a[N], b[N], c[N];
          int *dev_a, *dev_b, *dev_c;
          // allocate the memory on the GPU
          HANDLE_ERROR(cudaMalloc((void **)&dev_a, N * sizeof(int)));
          HANDLE_ERROR(cudaMalloc((void **)&dev_b, N * sizeof(int)));
          HANDLE_ERROR(cudaMalloc((void **)&dev_c, N * sizeof(int)));
          // fill the arrays "a" and "b" on the CPU
          for (int i = 0; i < N; i++)
          {
              a[i] = i;
              b[i] = i * i;
          }
          // copy the arrays "a" and "b" to the GPU
          HANDLE_ERROR(cudaMemcpy(dev_a,
                                  a,
                                  N * sizeof(int),
                                  cudaMemcpyHostToDevice));
          HANDLE_ERROR(cudaMemcpy(dev_b,
                                  b,
                                  N * sizeof(int),
                                  cudaMemcpyHostToDevice));
          add<<<1, N>>>(dev_a, dev_b, dev_c);
          // copy the array "c" back from the GPU to the CPU
          HANDLE_ERROR(cudaMemcpy(c,
                                  dev_c,
                                  N * sizeof(int),
                                  cudaMemcpyDeviceToHost));
          // display the results
          for (int i = 0; i < N; i++)
          {
              printf( "% d + % d = % d\n", a[i], b[i], c[i]);
          }
          // free the memory allocated on the GPU
          cudaFree(dev_a);
          cudaFree(dev_b);
          cudaFree(dev_c);
          return 0;
      }
```

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