BIRLA INSTITUTE OF TECHNOLOGY & SCIENCE, PILANI (RAJASTHAN)

CS F422 – Parallel Computing

Lab#9

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.

New Thread id to avoid stalls

The following code inside a kernel makes half of the threads stalled due to a conditional branching.

```
if(threadID % 2 ==0)
 {
   doSmt(IDprime);
 }
 else
 {
   doSmtElse(IDprime);
 }
```

Solution to this problem is to generate a new id that will assign either even or odd id to all the threads in a warp.

Q?

1. Compile the program warpFix.cu using the following command. Add the following code to measure time. Find the difference between time taken with threadId and warpId.

```
nvcc warpFix.cu -o out
./out
```

```
cudaEvent_t start, stop;
cudaEventCreate(&start);
cudaEventCreate(&stop);
cudaEventRecord(start);
vadd <<< grid, BLOCK_SIZE >>> (da, db, dc, N);
cudaEventRecord(stop);
cudaEventSynchronize(stop);
float milliseconds = 0;
cudaEventElapsedTime(&milliseconds, start, stop);
printf("Timetaken %f\n", milliseconds);
```

2. The program warpFix.cu provides solution only for two-way branching. If there is a multi-way branching, one more parameter is added to compute warpId. Please check in warpFixMultiway.cu

CUDA Streams

Consider the following code. In this code, data is transferred from host to GPU and vice-versa in synchronous mode. This will make all cores in GPU idle due to lack of data.

```
CUDA_CHECK_RETURN (cudaMemcpy (da, ha, sizeof (int) * N,
cudaMemcpyHostToDevice));
  CUDA_CHECK_RETURN (cudaMemcpy (db, hb, sizeof (int) * N,
cudaMemcpyHostToDevice));

  int grid = ceil (N * 1.0 / BLOCK_SIZE);
  vadd <<< grid, BLOCK_SIZE >>> (da, db, dc, N);

  CUDA_CHECK_RETURN (cudaThreadSynchronize ());
  // Wait for the GPU launched work to complete
  CUDA_CHECK_RETURN (cudaGetLastError ());
  CUDA_CHECK_RETURN (cudaMemcpy (hc, dc, sizeof (int) * N,
cudaMemcpyDeviceToHost));
```

CUDA streams makes it possible to transfer data in asynchronous mode.

Q?

1. Compile the program streamTest.cu. Add the code to measure time. Find the difference between time taken for using streams and not using streams.
2. Look at StreamTest2.cu. What difference does it make when compared to StreamTest.cu?
3. Consider the program vectorAdd.cu. Modify this program to use streams.

CUDA Dynamic Parallelism

An extension to the CUDA programming model which allows a thread to launch another grid of threads executing another kernel. Consider the program given below. Kernel in this program calls the same kernel again.

```
__global__ void QSort (int *data, int N, int depth)
{
  if(depth == MAXRECURSIONDEPTH)
  {
    insertionSort(data, N);
```

```
      return ;
    }

    if (N <= 1)
      return;

    // break the data into a left and right part
    int pivotPos = partition (data, N);

    cudaStream_t s0, s1;
    // sort the left part if it exists
    if (pivotPos > 0)
      {
        cudaStreamCreateWithFlags (&s0, cudaStreamNonBlocking);
        QSort <<< 1, 1, 0, s0 >>> (data, pivotPos, depth+1);
        cudaStreamDestroy (s0);
      }

    // sort the right part if it exists
    if (pivotPos < N - 1)
      {
        cudaStreamCreateWithFlags (&s1, cudaStreamNonBlocking);
        QSort <<< 1, 1, 0, s1 >>> (&(data[pivotPos + 1]), N - pivotPos - 1, depth+1);
        cudaStreamDestroy (s1);
      }
  }
```

Q?

1. Dynamic parallelism is supported in GPUs with compute capability at least 3.5. Check compute capability of your device. [If you do not know how to do this, check previous lab].
2. Compile and run the code in quicksort_dynamic.cu. Add code to measure time. Measure time for N=10, 100, 1000, 10000 etc.
3. Change the kernel by removing streams. Observe the time taken for different values of N.
4. When is the recursion depth of 24 breached?

Multi-GPU Programming with CUDA

If there are multiple GPUS attached to the system, how to use them affectively. Consider the following code. cudaSetDevice() is used to specify the GPU to use.

```
#include <stdio.h>
#include <cuda.h>
```

```

int main ()
{
  int deviceCount = 0;
  cudaGetDeviceCount (&deviceCount);
  if (deviceCount == 0)
    printf ("No CUDA compatible GPU.\n");
  else
    {
      for (int i = 0; i < deviceCount; i++)
        {
                    cudaSetDevice(i);
            cudaStreamCreate(&(str[i]));
                    h_data[i] = (int *)malloc(sizeof(int) * DATASIZE);
                    cudaMalloc((void ** )&(d_data[i]), sizeof(int) * DATASIZE);

                    // inititalize h_data[i]....

                    cudaMemcpyAsync(d_data[i], h_data[i], sizeof(int) * DATASIZE,
cudaMemcpyHostToDevice, str[i]);

                    doSmt <<< 10, 256, 0, str[i] >>> (d_data[i]);

                    cudaMemcpyAsync(h_data[i], d_data[i], sizeof(int) * DATASIZE,
cudaMemcpyDeviceToHost, str[i]);
        }
    }
  return 1;
}
```

Q?

1. Find out how many GPU devices are there on your system. [If you do not know how to do this, check previous lab].
2. Compile multigpu.cu and run it. Can you measure time if using one device vs multiple devices.
3. By checking slides, can you check if peer-to-peer data transfer is allowed?

End of lab9