CS546 Parallel and distributed Processing
Programming Assignment
Design

In this assignment, I have designed a CUDA-C/C++ version code of a Matrix normalization Algorithm, the sequential code is provided. By using power of GPUs to perform this task as fast as possible.

Let us discuss about design of CUDA-C/C++ version code of a Matrix normalization Algorithm:

Initially, we should accept 4 command line parameters.

1. Read Matrix size,

2. Read Number of Blocks

3. Read the Number of threads in each Block

4. Random Seed

And initialize inputs of A and B. We know that The CUDA event API includes calls to create and destroy events, record events, and compute the elapsed time in milliseconds between two recorded events. Before starting the computation, I've created CUDA event of type cudaEvent_t and are created and destroyed with cudaEventCreate () and cudaEventDestroy (). And cudaEventRecord() places the start and stop events into the default stream 0. And allocated memory for both A and B using cudaMalloc. And Matrix A is transferred to device from host using cudaMemcpy () since data transfers happened between the host and device using cudaMemcpy () are synchronous transfers.

Now, we will call the kernel normcal() by number of blocks and number of threads in each block. And I globally declared the function normcal(). Since Global memory is built from a bank of SDRAM chips connected to the GPU chip. Any thread in any MP can read or write to any location in the global memory. And declared the shared memory variables row, mean and standard deviation (Shared memory is a small memory within each MP that can be read/written by any thread in a block assigned to that MP). And calculate the column index by using block id and thread id. We know that column normalization is composed of three steps per column:

1. Calculating the mean of the column

2. Calculating the standard deviation

3. Finally, calculating the normalized value by performing the following calculation

B[row][col] = (A[row][col] – mean) / standard deviation

Here, I considered splitting the reductions into two:

1. inside the values in each block

2. Reducing the totals for every block.

And synchronized all the threads after calculation of each block. First, I calculated mean in each block and Synchronized. Then, calculated standard deviation in each block and synchronized.

Once the mean and standard deviation are calculated, then calculated the normalized value by performing the following calculation:

```
d_B[row*n+col] = (d_A[col*n+row] - mean) / standard deviation;
```

And we know that, when kernel launches, control returns immediately to the CPU and does not wait for the kernel to complete. Then, Matrix B is transferred to host from device using cudaMemcpy() The function cudaEventSynchronize() blocks CPU execution until the specified event is recorded. The cudaEventElapsedTime() function returns the number of milliseconds time elapsed between the recording of start and stop. Finally ,Deallocate memory of A and B.