Advanced Topic in CUDA

1) Create and run the `matmul2.cu` program (has been done in the previous lab) which multiplies two square matrices. *Width* is strictly a multiple of *TILE_WIDTH*.

```
#include <stdio.h>

#define Width 32     // size of Width x Width matrix
#define TILE_WIDTH 16

__global__ void MatrixMulKernel(float* Md, float* Nd, float* Pd, int ncols) {

    int row = blockIdx.y*blockDim.y + threadIdx.y;
    int col = blockIdx.x*blockDim.x + threadIdx.x;

    // Pvalue is used to store the element of the output matrix
    // that is computed by the thread

    float Pvalue = 0;
    for (int k = 0; k < ncols; ++k) {
       float Melement = Md[row*ncols+k];
       float Nelement = Nd[k*ncols+col];
       Pvalue += Melement * Nelement;
    }

    Pd[row*ncols+col] = Pvalue;
}


int main (int argc, char *argv[] ) {

    int i,j;
    int size = Width * Width * sizeof(float);
    float M[Width][Width],N[Width][Width],P[Width][Width];
    float* Md, *Nd, *Pd;

    for (i=0; i < Width; i++) {
        for (j=0; j < Width; j++) {
           M[i][j] = 1; N[i][j] = 2;
        }
    }
    cudaMalloc( (void**)&Md, size);
    cudaMalloc( (void**)&Nd, size);
    cudaMalloc( (void**)&Pd, size);

    cudaMemcpy( Md, M, size, cudaMemcpyHostToDevice);
    cudaMemcpy( Nd, N, size, cudaMemcpyHostToDevice);

    // Setup the execution configuration
    dim3 dimBlock(TILE_WIDTH, TILE_WIDTH);
    dim3 dimGrid(Width/TILE_WIDTH,Width/TILE_WIDTH);
```

```
    // Launch the device computation threads!
    MatrixMulKernel<<<dimGrid, dimBlock>>>(Md, Nd, Pd, Width);

    // Read P from the device
    cudaMemcpy(P, Pd, size, cudaMemcpyDeviceToHost);

    // Free device matrices
    cudaFree(Md); cudaFree(Nd); cudaFree (Pd);

    for (i=0; i < Width; i++) {
       for (j=0; j < Width; j++) {
           printf("%.2f ",P[i][j]);
       }
       printf("\n");
    }

}
```

2) Modify the program `matmul2.cu` into `matmul_shared.cu` to use tiling with shared memory (as given in the lecture)

3) Get the compiler version. Run the following command

nvcc –version

4) Compile and run the following code (`hist.cu`). Please recall that to compile this code in emulation mode, you must enable the atomic operation support.

```
#include <stdio.h>

#define n 1024
#define NUMTHREADS 256

__global__ void histogram_kernel( unsigned int *data, unsigned int *bin) {
   int i = blockIdx.x * blockDim.x + threadIdx.x;
   if (i < n) {
      atomicAdd( &(bin[data[i]]), 1 );
   }
}


int main (int argc, char *argv[] ) {

   int i;
   int size = n *sizeof(int);
   unsigned int a[n];
   unsigned int bin[10];
   unsigned int *dA, *dBin;

   for (i=0; i < n; i++) {
      a[i] = i % 10;
   }

   cudaMalloc( (void**)&dA, size);
   cudaMalloc( (void**)&dBin, 10*sizeof(int));

   cudaMemcpy( dA, a, size, cudaMemcpyHostToDevice);
   cudaMemset( dBin,0, 10*sizeof(int));
```

```
    int nblocks = (n+NUMTHREADS-1)/NUMTHREADS;

    histogram_kernel<<<nblocks, NUMTHREADS>>>(dA,dBin);

    cudaMemcpy(bin, dBin, 10*sizeof(int), cudaMemcpyDeviceToHost);

    cudaFree( dA);
    cudaFree( dBin);

    int count = 0;
    for (i=0; i < 10; i++) {
      printf("Freq %d = %d\n",i,bin[i]);
      count = count + bin[i];
    }
    printf("#elements = %d\n",count);

}
```