Assignment #3: OpenCL Matrix Multiplication

Create an OpenCL program that takes as inputs two square matrices A and B with dimension (40 x 40) and performs the multiplication of the two matrices to create matrix C = A x B.

Requirements:

1. For the purpose of easy grading and error checking, please define N and BLOCK_SIZE to 40 and 1 respectively, and initialize your input matrices A and B as below ("inputMatrix1" and "inputMatrix2" stand for A and B, "results" stands for C):

   ```
   #define N 40
   #define BLOCK_SIZE 1
   ```

   …

   ```
   cl_float *inputMatrix1;
   cl_float *inputMatrix2;
   cl_float *results;
   cl_uint width = N;

   int x,y;
   int data = 0;
   inputMatrix1 = (cl_float *) malloc(sizeof(cl_float) * width * width);
   inputMatrix2 = (cl_float *) malloc(sizeof(cl_float) * width * width);
   results = (cl_float *) malloc(sizeof(cl_float) * width * width);

   for(y = 0; y < width; y++) {
     for(x = 0; x < width; x++) {
        inputMatrix1[y * width + x]= data;
        inputMatrix2[y * width + x]= data;
        results[y * width + x]=0;
        data ++;
     }
   }
   ```

2. Write your kernel function separately, i.e., create a.cl file for your kernel function other than writing it as a string inside the main program. You can load your .cl kernel file by calling loadProgSource(…) function. (Refer to the Sample Code in 14_OpenCL_ProgramFlow "vecSquare_2.cpp").
3. Retrieve the latest compilation results embedded in the program object by clGetBuildProgramInfo(). (Refer to page 14 in Slides_14_Programming Details).
4. In your kernel function, decompose the multiplication into small work-groups working in parallel, i.e., you need to specify both the

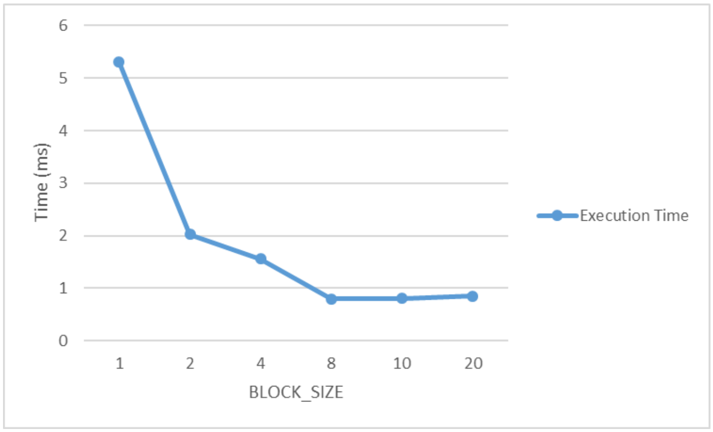
total number of work-items (global dimensions) and the number of work-items per work-group (local dimensions) and pass them to clEnqueueNDRangeKernel(…), e.g.,

```
global[0] = width;
global[1] = width;

local[0] = BLOCK_SIZE;
local[1] = BLOCK_SIZE;

clEnqueueNDRangeKernel(..., global, local, ...);
```

5. Use the event to profile the kernel execution time. (Refer to Page 60 on Slides_15_Kernels, Memories, Synchronization and Events, but use CL_PROFILING_COMMAND_START and CL_PROFILING_COMMAND_END in clGetEventProfilingInfo(…) calls instead.)
6. Gradually increase the BLOCK_SIZE from 1 to 2, 4, 8, 10, and 20, respectively, rerun your code, record your kernel execution time from event profiling each time, and finally draw a time vs. BLOCK_SIZE chart to show the trend, e.g.,


Time (ms)
BLOCK_SIZE
Execution Time