Report Lab 1

Andres Calderon

October 29, 2015

1 Code

The following code was used to complete the report:

1.1 kernel.cu

```
/******************************************************************************
 *cr
 *cr            (C) Copyright 2010 The Board of Trustees of the
 *cr                        University of Illinois
 *cr                         All Rights Reserved
 *cr
 ******************************************************************************/

#include <stdio.h>

#define TILE_SIZE 16

__global__ void mysgemm(int m, int n, int k, const float *A, const float *B, float *C) {

    /********************************************************************
     *
     * Compute C = A x B
     *   where A is a (m x k) matrix
     *   where B is a (k x n) matrix
     *   where C is a (m x n) matrix
     *
     * Use shared memory for tiling
     *
     ********************************************************************/

    // INSERT KERNEL CODE HERE
    // Declaring the variables in shared memory...
    __shared__ float A_s[TILE_SIZE][TILE_SIZE];
  __shared__ float B_s[TILE_SIZE][TILE_SIZE];

  // Finding the coordinates for the current thread...
  int tx = threadIdx.x;
  int ty = threadIdx.y;
  int col = blockIdx.x * blockDim.x + tx;
  int row = blockIdx.y * blockDim.y + ty;

  float sum = 0.0f;

  for(int i = 0; i < ((k - 1) / TILE_SIZE) + 1; ++i){
    // Validation in the case the thread tries to write in share
    // memory of the dimensions of matrix A...
    if(row < m && (i * TILE_SIZE + tx) < k){
      A_s[ty][tx] = A[(row * k) + (i * TILE_SIZE + tx)];
    } else {
      // In that case, just write a 0 which will no affect
      // the computation...
      A_s[ty][tx] = 0.0f;
    }
    // Similar validation for B...
    if((i * TILE_SIZE + ty) < k && col < n){
      B_s[ty][tx] = B[((i * TILE_SIZE + ty) * n) + col];
    } else {
      B_s[ty][tx] = 0.0f;
    }
    // Wait for all the threads to write in share memory
    __syncthreads();

    // Compute the multiplication on the tile...
    for(int j = 0; j < TILE_SIZE; ++j){
      sum += A_s[ty][j] * B_s[j][tx];
    }
    // Wait to finish before to go ahead with the next phase...
    __syncthreads();
  }
  // Write the final result in C just if it is inside of the valid
  // dimensions...
  if(row < m && col < n){
    C[row * n + col] = sum;
  }

}

void basicSgemm(char transa, char transb, int m, int n, int k, float alpha, const float *A, int lda, const float *B, int ldb, float beta, float *C, int ldc)
{
    if ((transa != 'N') && (transa != 'n')) {
    printf("unsupported value of 'transa'\n");
      return;
    }

    if ((transb != 'N') && (transb != 'n')) {
    printf("unsupported value of 'transb'\n");
    return;
    }

    if ((alpha - 1.0f > 1e-10) || (alpha - 1.0f < -1e-10)) {
    printf("unsupported value of alpha\n");
    return;
    }

    if ((beta - 0.0f > 1e-10) || (beta - 0.0f < -1e-10)) {
    printf("unsupported value of beta\n");
    return;
    }
    const unsigned int BLOCK_SIZE = TILE_SIZE;

    // Initialize thread block and kernel grid dimensions
    const dim3 dim_block(BLOCK_SIZE, BLOCK_SIZE, 1);
  const dim3 dim_grid(((n - 1) / BLOCK_SIZE) + 1, ((m - 1) / BLOCK_SIZE) + 1, 1);

    // Calling the kernel with the above-mentioned setting...
  mysgemm<<<dim_grid, dim_block>>>(m, n, k, A, B, C);
}
```

2 Answer to Questions

1. How many times is each element of the input matrices loaded during the execution of the kernel?

References

[1] Nvidia Corporation. *CUDA C Programing Guide*. PG-02829-001_v7.5, 2015.

[2] David Kirk and Wen-Mei Hwu. *Programming Massively Parallel Processors: A Hands-On Approach*. Morgan Kaufmann, 2012.

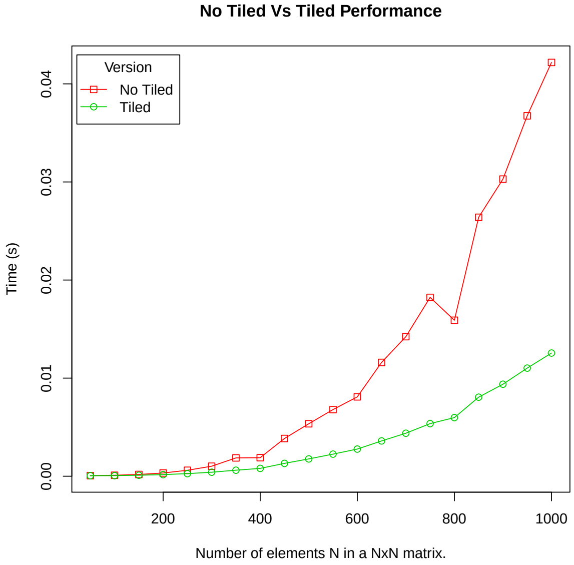

Figure 1: First performance comparisson between tiling and no tiling versions.

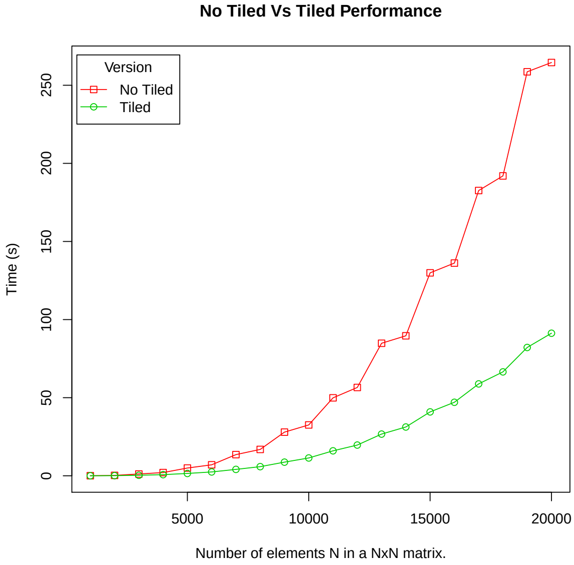

Figure 2: Second performance comparisson between tiling and no tiling versions.

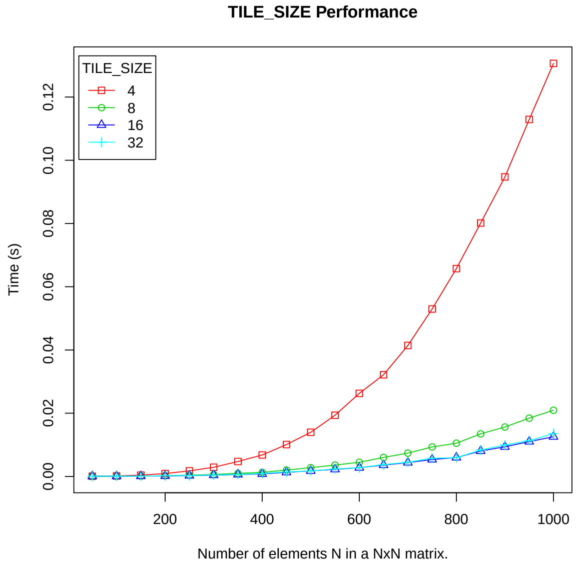

Figure 3: Performance using different values of TILE_SIZE.

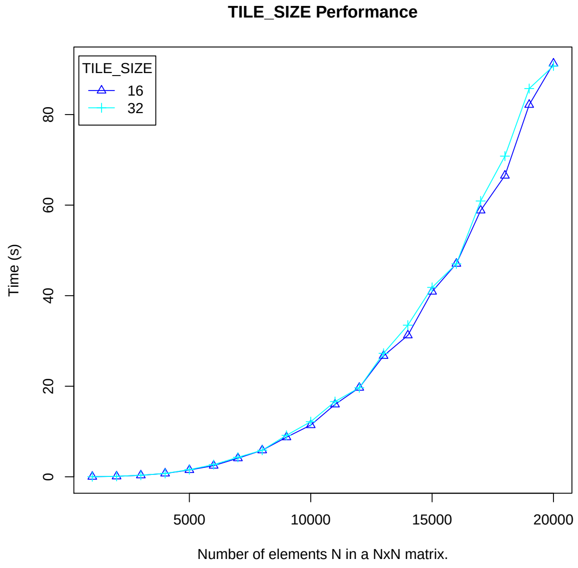

Figure 4: Performance of TILE_SIZE 16 y 32 with more data.

[3] David Luebke, John Owens, Mike Roberts and Cheng-Han Lee. *Coalesce Memory Access - Intro to Parallel Programming*. Udacity Course, 2015. `https://www.udacity.com/course/intro-to-parallel-programming--cs344`.