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;
```

```
48      }
49      // Similar validation for B...
50      if((i * TILE_SIZE + ty) < k && col < n){
51        B_s[ty][tx] = B[((i * TILE_SIZE + ty) * n) + col];
52      } else {
53        B_s[ty][tx] = 0.0f;
54      }
55      // Wait for all the threads to write in share memory
56      __syncthreads();
57
58      // Compute the multiplication on the tile...
59      for(int j = 0; j < TILE_SIZE; ++j){
60        sum += A_s[ty][j] * B_s[j][tx];
61      }
62      // Wait to finish before to go ahead with the next phase...
63      __syncthreads();
64    }
65    // Write the final result in C just if it is inside of the valid
66    // dimensions...
67    if(row < m && col < n){
68      C[row * n + col] = sum;
69    }
70
71  }
72
73  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)
74  {
75      if ((transa != 'N') && (transa != 'n')) {
76      printf("unsupported value of 'transa'\n");
77        return;
78      }
79
80      if ((transb != 'N') && (transb != 'n')) {
81      printf("unsupported value of 'transb'\n");
82      return;
83      }
84
85      if ((alpha - 1.0f > 1e-10) || (alpha - 1.0f < -1e-10)) {
86      printf("unsupported value of alpha\n");
87      return;
88      }
89
90      if ((beta - 0.0f > 1e-10) || (beta - 0.0f < -1e-10)) {
91      printf("unsupported value of beta\n");
92      return;
93      }
94      const unsigned int BLOCK_SIZE = TILE_SIZE;
95
96      // Initialize thread block and kernel grid dimensions
97      const dim3 dim_block(BLOCK_SIZE, BLOCK_SIZE, 1);
98    const dim3 dim_grid(((n - 1) / BLOCK_SIZE) + 1, ((m - 1) / BLOCK_SIZE) + 1, 1);
99
100     // Calling the kernel with the above-mentioned setting...
101   mysgemm<<<dim_grid, dim_block>>>(m, n, k, A, B, C);
102 }
```

2 Answer to Questions

1. In your kernel implementation, how many threads can be simultaneously executing? Assume a GeForce GTX 280 GPU which has 30 streaming multiprocessors.

2. Use nvcc –ptxas-options="-v" to report the resource usage of your implementation your implementation. Note that the compilation will fail but you will still get a report of the relevant information. Experiment with the Nvidia visual profiler, which is part of the CUDA toolkit, and use it to further understand the resource usage. In particular, report your branch divergence behavior and whether your memory accesses are coalesced.

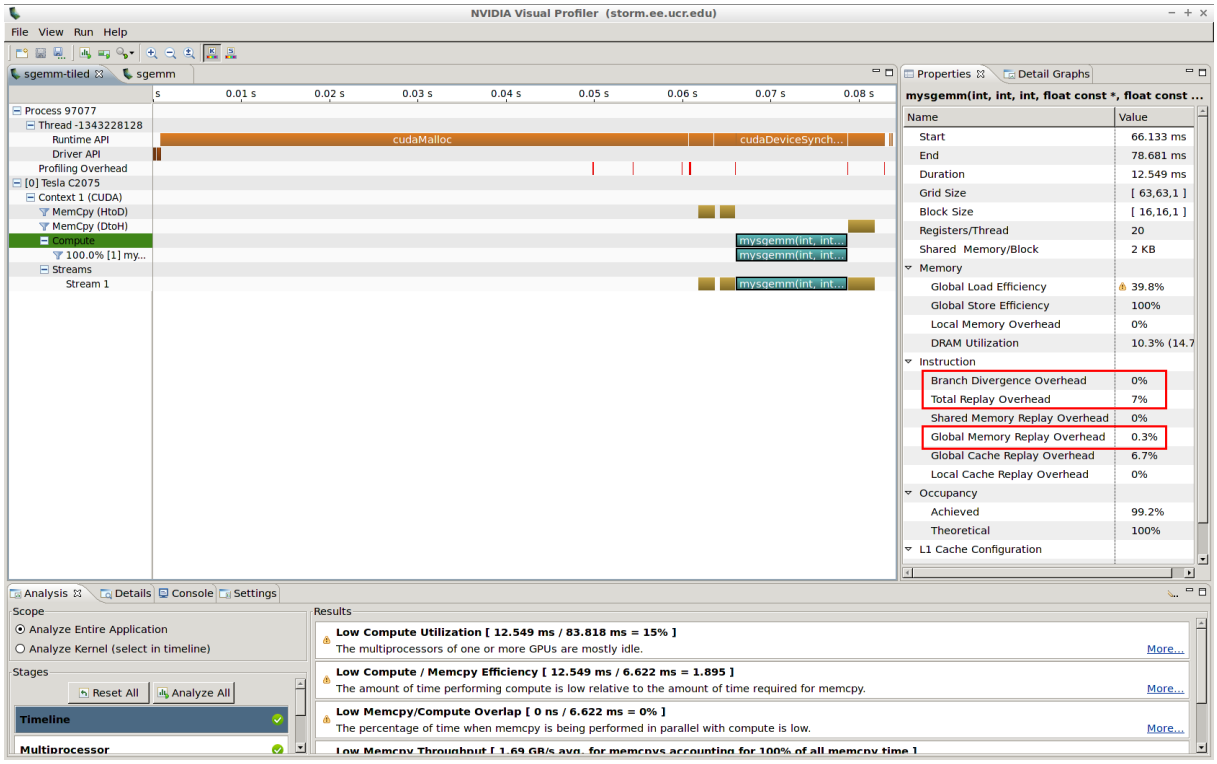

Figure 1: NVVP performance analysis for *sgemm-tiled*.

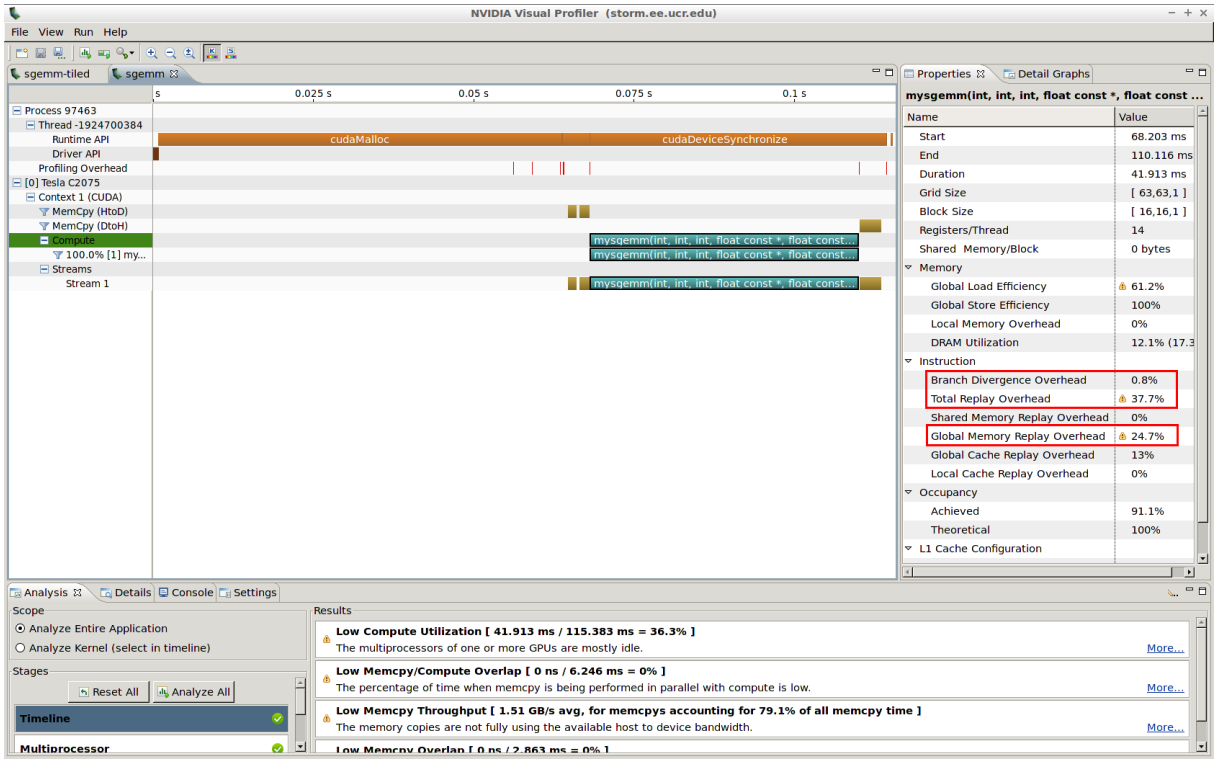

Figure 2: NVVP performance analysis for *sgemm*.

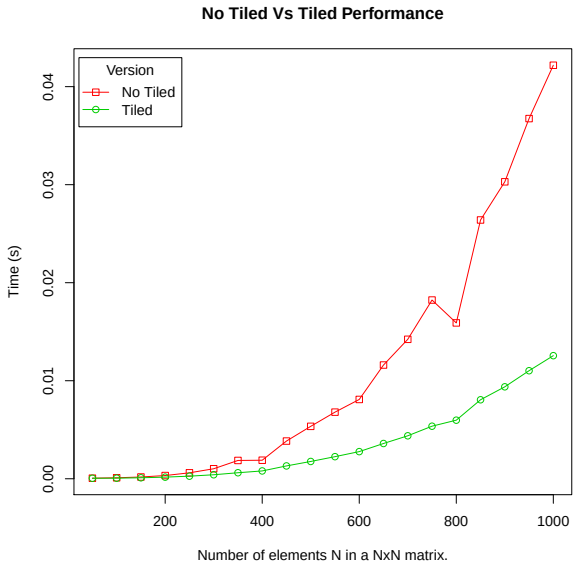

Figure 3: First performance comparisson between tiling and no tiling versions.

3. Compare the performance of the The Tiled Matrix multiplication to the simple matrix multiplication as you increase the size of the matrices and for different tile sizes. Explain any trends that you see.

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.

[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`.

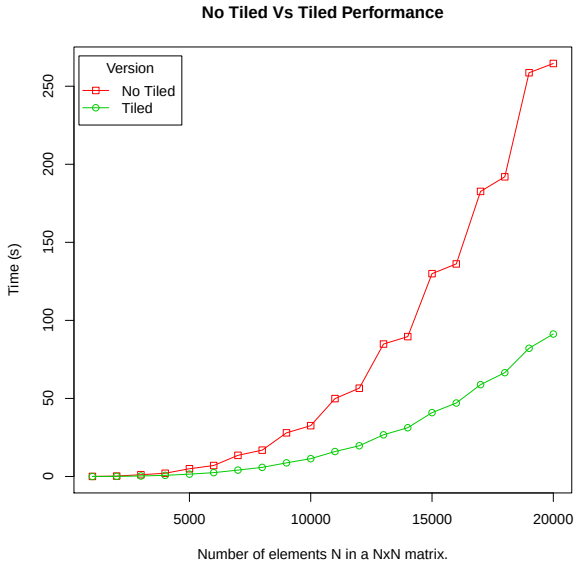

Figure 4: Second performance comparisson between tiling and no tiling versions.

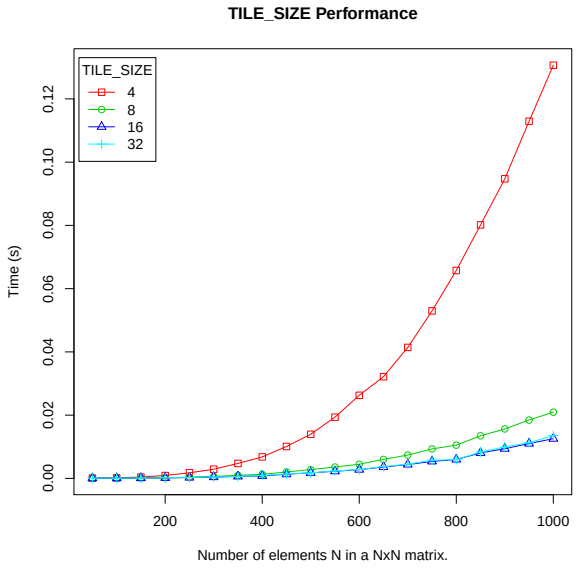

Figure 5: Performance using different values of TILE_SIZE.

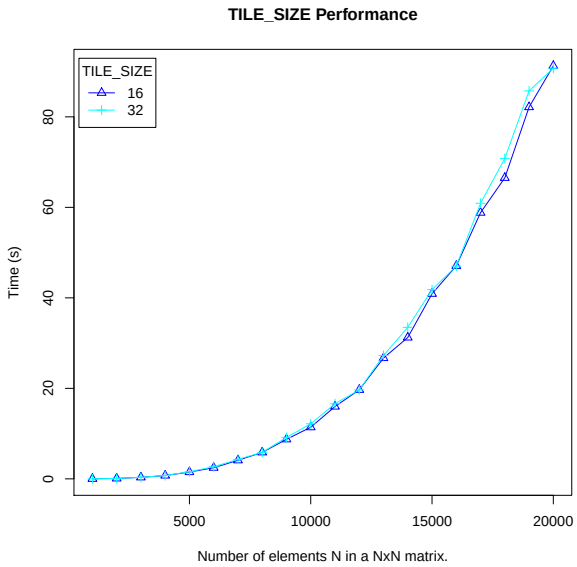

Figure 6: Performance of TILE_SIZE 16 y 32 with more data.