

Homework 5
Due: 7/2/09

CIS 665

Problem 1: Analyzing Performance of CUDA (20 points)

Utilizing the NVIDIA CUDA Visual Profiler, execute a CUDA Matrix Multiplication with profiling enabled and view the profiler output as a table. (You can download the Visual Profiler from the CUDA download section on the NVIDIA website if you've not done so already.)

Building on the Matrix Multiplication exercise from the last homework; Compare profiler output for multiple program runs of matrixMul. Each program run is referred to as a session. For the largest matrix you tested in your CUDA program last homework that produced interesting results (the size seemed to be different for many of you) vary the BLOCK_SIZE parameter 1,4,8,16, etc

Compare and contrast the various times, memcopy, and Occupancy. Find the 'sweet spot' and report why that BLOCK_SIZE produces the most optimal results.

Problem 2: Advanced Performance Experiments of CUDA (30 points)

Select **one** of the following questions below. Write a unique CUDA program that illustrates the "optimization benefit" (OB) or "performance cliff" (PC) in the example. Also provide a brief (a few sentences) description of what is happening as a comment inside the README document.

- a) [PC] Show an example code where you fill up the register file due to too many threads. You should have two versions of the code, one where the number of threads is within the range of registers, and one where the register capacity is exceeded.
- b) [OB] Show the performance impact of unrolling an innermost loop in a nest. See how far you can push it before you run into the problems of a. above.
- c) [OB/PC] Explore when the compiler decides to put array variables that are local to the device function in registers. What access patterns lead to the compiler using a register vs. using local memory?
- d) [OB/PC] Show the performance advantage of constant memory when the data is cached, and what happens to performance when the data exceeds the cache capacity and locality is not realized.
- e) Demonstrate the performance impact of parallel memory access (no bank conflicts) in shared memory. For example, implement 3-4 of reduction improvement computations like in Lecture in shared memory, with one version demonstrating bank conflicts and the other without.

Problem 3: Advanced CUDA Programming (50 points)

Select **one** of the following questions below. Write a unique CUDA program that demonstrates the solution. (Please document which problem you are solving in this and the previous question.) Email instructor for approval once you choose your own adventure here, so I can confirm it's a nontrivial amount of work.

A) Consider the following program fragment, which is a Jacobi relaxation algorithm

```
#define n 64
float a[n][n][n], b[n][n][n];
for (i=1; i<n-1; i++)
  for (j=1; j<n-1; j++)
    for (k=1; k<n-1; k++) {
      a[i][j][k]=0.8*(b[i-1][j][k]+b[i+1][j][k]+b[i][j-1][k] +
                    b[i][j+1][k]+b[i][j][k-1]+b[i][j][k+1]);
    }
```

This type of computation accesses the “nearest neighbors” of a data point, applying a function to the neighbors to compute the value at the current point. Such computations are sometimes referred to as stencils. The access pattern has reuse on array B in 3 dimensions. Your assignment is to write a CUDA version of Jacobi that applies tiling to exploit locality in shared memory for the 3 dimensions of B. You will be provided with a sequential CPU implementation of the code if you choose this, which extends the above loop nest to include initialization of B and final output. Please use the timing and comparison to the CPU results found in the matrix multiply example code.

B) A nontrivial addition to the smoke simulator developed in CIS 563: Physically Based Animation. For example, porting the Conjugate Gradient algorithm and other calculations to solve the linear system of equations onto the GPU to improve performance or adapting a volume renderer on the GPU for the final rendering display.

C) Matrix convolution is primarily used in image processing for tasks such as image enhancing, blurring, etc. A standard image convolution formula for a 5x5 convolution kernel A with matrix B is

$$C(i,j) = \text{sum}(m = 0 \text{ to } 4) \{ \text{sum}(n = 0 \text{ to } 4) \{ A[m][n] * B[i+m-2][j+n-2] \} \}$$

where $0 \leq i < B.\text{height}$ and $0 \leq j < B.\text{width}$

Elements that are "outside" the matrix B, for this exercise, are treated as if they had value zero.

- 1) Unzip the startup code files into <Proj_Dir>\NVIDIA CUDA SDK\projects
- 2) Edit the source files 2Dconvolution.cu and 2Dconvolution_kernel.cu to complete the functionality of the matrix convolution on the device. Compile using the provided solution file with Visual Studio
- 3) The modes of operation for the application are described here.

No arguments: The application will create a randomized kernel and image. A CPU implementation of the convolution algorithm will be used to generate a correct solution which will be compared with your programs output. If it matches (within a certain tolerance), it will print out "Test PASSED" to the screen before exiting.

One argument: The application will use the random initialization to create the input matrices, and write the device-computed output to the file specified by the argument.

Three arguments: The application will read input matrices from provided files. The first argument should be a file containing two integers. The first and second integers will be used as N.height and N.width, respectively. The second and third function arguments will be expected to be files which have exactly enough entries to fill matrices M and N respectively. No output is written to file.

Four arguments: The application will read its inputs using the files provided by the first three arguments, and write its output to the file provided in the fourth.

Note that if you wish to use the output of one run of the application as an input, you must delete the first line in the output file, which displays the accuracy of the values within the file. The value is not relevant for this application.

- 3) Construct a detail performance report.