EECS 368/468 – Lab2
Programming Massively Parallel Processors with CUDA

Tiled Matrix Multiplication in CUDA

In this assignment, you will work with your teammates to write a tiled matrix multiplication code in CUDA.

Assignment Task

In this lab you will implement a tiled matrix multiplication. You will need to break the input and output matrices into tiles. Each thread block should compute one output tile at a time. To do this, first bring two source tiles into shared memory, and use them to compute the partial results of one tile of the output matrix. Then, bring in the next two source tiles for the same output tile and update the partial results. Repeat until the output tile is fully computed. Other blocks may be working on other tiles of the output matrix at the same time. A picture of this process is shown in the handout of Lecture 5 (page 15).

Your goal is to write a code that:

1. Correctly computes the result matrix
2. Issues coalesced accesses to global memory
3. Avoids shared memory bank conflicts.

The techniques to achieve this are similar to the transpose example of Lecture 6.

Install lab2 at your Wilkinson Lab account

1. Download the lab2 tarball from canvas into the `EECS468-CUDA-Labs` directory in your Wilkinson Lab account. NOTE: the remaining labs will use the same code scaffold used in lab1. Each new tarball we will provide from now on will have only the new sources needed for the new lab.

2. **Remember**: every time you login to the Wilkinson lab to program in CUDA you need to setup the CUDA environment. If you use csh or tcsh:
   ```
   source /usr/local/cuda-5.0/cuda-env.csh
   ```
   or if you use the bash shell:
   ```
   . /usr/local/cuda-5.0/cuda-env.sh
   ```

3. Install the lab2 tarball and compile the lab sources:
   ```
   cd EECS468-CUDA-Labs
   tar xvf EECS468-CUDA-Lab2.tgz
   make
   ```

Complete the assignment

4. Edit the following two functions
   `MatrixMulOnDevice(...)` in the source file `matrixmul.cu`
   `MatrixMulKernel(...)` in the source file `matrixmul_kernel.cu`
   to implement tiled matrix multiplication on the GPU device as explained above. **Do not change any other piece of the source code**. The two input matrices could be of **any size**, but one CUDA grid is guaranteed to cover the entire output matrix.

The source code you will be working on is at :
`EECS468-CUDA-Labs/labs/src/lab2`

Compiling your code will produce the executable :
`EECS468-CUDA-Labs/labs/bin/linux/release/lab2`

5. There are several modes of operation for the application. Note that the file interface has been updated to allow the size of the input matrices to be read.

   a. **No arguments:** the application will create two randomly sized and initialized matrices such that the matrix operation M * N is valid, and P is properly sized to hold the result. After the device multiplication is invoked, it will compute the correct solution matrix using the CPU, and compare that solution with the device-computed solution. If it matches (within a certain error tolerance), it will print out "`Test PASSED`" to the screen before exiting.

   b. **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.

   c. **Three arguments:** the application will read the input matrices from provided files. The first argument should be a file containing three integers. The first, second and third integers will be used as `M.height`, `M.width`, and `N.width` respectively. The second and third 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.

   d. **Four arguments:** the application will read its inputs from the files provided by the first three arguments as described above, 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.

6. **Hand in your solution:**

   a. Create a tarball of your solution, which includes the contents of the `lab2` source folder provided, with all the changes and additions you made to the source files. Please make sure you do a `make clobber` before submitting; don't include object code or executables, as these are typically large files:

      ```
      cd EECS468-CUDA-Labs/labs/src/lab2
      make clobber
      tar cvfz EECS468-lab2-YourNames.tgz *
      ```

      where you should replace `YourNames` with the **concatenated full names of the team members**.

   b. In addition to the code, add a text file named **`answer.txt`** in your lab2 source directory with the names and netIDs of all team members, and with your **answer to the following question**:

      In your kernel implementation of tiled matrix multiplication, how many threads can be simultaneously scheduled for execution on a GeForce GTX-680 GPU?

You can use the command:

```
nvcc   -gencode=arch=compute_30,code=\"sm_30,compute_30\"
--ptxas-options=-v matrixmul_kernel.cu
```

to see the resource usage of your kernel (although compilation may fail, it will only do so after compiling the kernel and displaying the relevant information.) The slides of the previous lectures have all the resource limitations of the GeForce GTX-680 GPU (compute capability=3.0). You can see the configuration of your GPU device by executing the `deviceQuery` facility in the SDK (step #4 in lab1.)

c. Submit your solution tarball via Canvas. You can submit as part of the same group that you used for the previous assignment, or join a new one. Then submit your solution on behalf of the entire group. This submission counts for every person in the group, and all group members will receive the same grade for it. If you submit multiple times, only the latest submission will be graded.

Grading

Your submission will be graded based on the following parameters.

a. Functionality/knowledge: 80%
   - The kernel produces correct results at the output. Remember, the input matrixes can be of any size. (20%)
   - The kernel uses shared memory to avoid global memory latencies. (30%)
   - The kernel minimizes shared memory bank conflicts. (30%)

b. Report: 20%
   - You correctly answer the provided question and explain your solution.

Good luck!