CPSC 418: Parallel Computing
Due 11:59 on April 7, 2017.
Early bird 11:59 on April 5, 2017.

Winter 2016–2017 Term 2

Homework #5: More Fun with CUDA

Please submit your solution using the handin program. Submit your solution as

cs418 hw5

Your submission should consist of three files:

- `hw5_conv.cu`: CUDA source code for your solution to the coding parts of the convolution problems.
- `hw5_blas.cu`: CUDA source code for your solution to the coding parts of the power iteration problem.
- `hw5.pdf`: PDF for the written response parts of your solution

If your code does not compile, we might give you zero points for the problem. Points will be deducted for compiler warnings.

An archive containing the necessary source code and separate archives for the data files for this assignment are available at

http://www.ugrad.cs.ubc.ca/~cs418/2016-2/hw/5/code.html.

Note that the data files are *very* large. If you are working on the `linXX` boxes, see the instructions at the end of this assignment for how to symbolically link to our copy of these files rather than making your own copy.

1. **1D Convolution – code (35 points):** In this question you will write and test simple 1D convolution kernels that make use of another specialized class of GPU memory called "constant memory." Before starting this question, be sure to
   - Read K&H sections 8.1–8.3 or 7.1–7.3 respectively.
   - Download and make the template convolution code. For instructions on how to do so, see the end of the assignment. All functions referred to below are in the file `hw5_conv.cu` and this is the only file you should modify.

For this question, you goal is to get working implementations of convolution. You don't need to worry about optimizing your code—we'll save that for Question 3.

(a) (15 points): In the file `hw5_conv.cu` the sample code `conv1h_basic()` loads the input image, copies it into global GPU memory, launches `conv1h_kernel()`, copies the result back to the CPU memory and saves it as an output image. The sample kernel code `conv1h_basic_kernel()` creates the photo-negative of the input image.

Modify `conv1h_basic()` and `conv1h_basic_kernel()` to implement a 1D convolution of the image in the horizontal direction by the convolution stencil specified in the global variable `hos_stencil_1dx`. To improve performance, you should store the stencil data in the GPU's constant memory.

You will find the code in K&H figure 8.8 or 7.8 useful for this question, but keep in mind that the input data is a two dimensional array. If you are successful, your output image should show a noticeable horizontal blurring.

(b) (10 points): Write the function `conv1v_basic()` and kernel `conv1v_basic_kernel()` which implements a 1D convolution of the image in the vertical direction by the convolution stencil specified in the global variable `hos_stencil_1dy`. Your code for this part should be similar but not identical to your code for the previous part. You should use the GPU's constant memory for the stencil data. Check that your output image shows a noticeable vertical blurring.

(c) (10 points): Write the function `conv1to2_basic()` which implements a 2D convolution of the image by performing a 1D horizontal convolution followed by a 1D vertical convolution. You should use the kernels you developed in the previous two parts. Check that your output image shows a noticeable blurring in both directions.

2. **1D Convolution – performance (20 points):** In this problem, you will perform some simple performance analysis of `conv1to2_basic` from Question 1. As noted in that question, the goal in the simple implementation is just to get working code. You don't need to worry getting good performance, you just need to understand the performance. You should be sure to read section 8.1 of Kirk's & Hwu's 2nd edition or section 7.1 of the 3rd edition.

(a) (5 points): How many floating point operations (i.e. the total number of multiplications plus the total number of additions) does your implementation of `conv1to2_basic` perform to compute the 2D convolution of an image with h rows and w columns using a stencil with s_x elements in the horizontal direction and a stencil with s_y elements in the vertical direction? Your answer should be stated in terms of h, w, s_x and/or s_y. You may assume that $s_y < h$ and $s_x < w$.

(b) (5 points): With the same assumptions as the previous part, how many global memory accesses does your implementation of `conv1to2_basic` perform? You may assume that the stencil data is successfully cached and so does not require any global memory access.

(c) (5 points): With the same assumptions as the previous parts, what is the CGMA for your implementation of `conv1to2_basic`?

(d) (5 points): What is the throughput of your implementation of `conv1to2_basic` in pixels/second when using `booby.ppm` as the source image?

Note: Even if you do not have a working solution to `conv1to2_basic` you may answer these questions based on your solution for `conv1h_basic` or `conv1v_basic`. To receive credit, you must clearly state that you are doing so in your solution.

3. **1D Convolution with Tiling (45 points):** In this question you will improve the throughput of your convolution kernels using tiling. Before starting this question, be sure to read section 8.4 or 7.4 respectively.

   For this question, you should try to get reasonably good performance. Given the time constraints of this assignment, we aren't asking you to push for the absolute highest performance possible.

   (a) (30 points) Create functions `conv1to2_tiled()`, `conv1h_tiled_kernel()` and `conv1v_tiled_kernel()` to implement a 2D convolution of the image using 1D tiles of size t in shared memory to reduce the number of global memory accesses. You should try to achieve a reasonably good throughput, for example, as measured in pixels/second.

   You will find the code in K&H figure 8.11 or 7.11 useful for this question, but keep in mind that the input data is a two dimensional array. If you are successful, your output image should look the same as the final output image from the previous question.

   (b) (5 points): Under the same assumptions as the previous question, approximately what ratio of global memory accesses to arithmetic operations does your code achieve as a function of h, w, s_x, s_y and t? You may assume that the convolution stencil data is successfully cached, and so does not require global memory access. You may also place constraints on the size of t (for example, $\max(s_x, s_y) < t < \max(h, w)$ are obvious bounds), but do not assume that t is constant.

   (c) (10 points): What throughput, in pixels/second, does your kernel achieve? Describe the performance trade-offs that you considered to achieve this performance.

4. **cuBLAS (40 points)**. The *power method* or *power iteration* is a simple algorithm for finding the dominant (largest) eigenvalue and corresponding eigenvector of a matrix. For a matrix A, let λ denote the largest eigenvalue and v the corresponding eigenvector.

   The power method starts with a vector b_0, which technically must satisfy $v^T b_0 \neq 0$ but for practical purposes is usually chosen randomly. The method then iterates the formula

$$b_{k+1} = \frac{Ab_k}{\|Ab_k\|}, \tag{1}$$

where $\|x\|$ denotes the two-norm of a vector x. Then b_k will converge to v and $\|Ab_k\|$ will converge to $|\lambda|$ if $|\lambda|$ is strictly larger than the magnitude of any other eigenvalue of A.

Note that the iteration (1) can be decomposed into the steps

$$\begin{aligned} b &\leftarrow Ab \\ \lambda &\leftarrow \|b\| \\ b &\leftarrow (1/\lambda)b \end{aligned} \tag{2}$$

which can be repeated as desired.

(a) (5 points): What are the BLAS subroutines for each of the three steps in (2)?

(b) (15 points): In the file `hw5_blas.cu` complete the function `power_method()` using calls to the cuBLAS library. You may assume that the input matrix and vector satisfy the appropriate mathematical requirements for convergence.

(c) (5 points): Test your function on the provided data using $k = 500$ iterations of the power method. How long does your code take to estimate the eigenvalue of a matrix that is $n \times n$ for $n = 2048$?

(d) (5 points): The `linXX` machines have nVidia GTX 550 Ti GPUs. These GPUs have a maximum memory bandwidth of 98.2 Gbytes per second. Given the time that you measured to run your code, what is the maximum number of single precision floating point values that could be loaded from global memory on these GPUs?

(e) (5 points): Assume that the cuBLAS routines perform the same number of floating point operations as the brute-force algorithm—the cuBLAS routines just arrange these operations to make good use of the GPU. How many floating point operations (i.e. the total number of multiplications plus the total number of additions) are needed to run k iterations of (1) on a matrix of size $n \times n$?

(f) (5 points): Use your answers to the previous two parts of this question to compute a lower bound on the CGMA that your algorithm based on cuBLAS achieves when estimating the maximum eigenvalue of a matrix with $n = 2048$ and $k = 500$ iterations.

The Convolution Template Code

To build the project, call `make clean; make` under the homework folder. If the code compiles, this should generate a binary `hw5_conv`.

The sample images for this question are large. If you are working on the `linXX` machines, we recommend that you create a link to our copy of these files rather than copying them

into your own directory. To create a (symbolic) link to our image directory in your current directory, use the command

```
ln -s ~cs418/public_html/2016-2/hw/5/images .
```

Note that there is a space between "`/images`" and the final "." in this command. You should then be able to access the `stippling.ppm` image file as `images/stippling.ppm`.

For the questions involving convolution, you will only be modifying a subset of the code in the file `hw5_conv.cu`; the sections which must be modified are flagged with "`TODO`". **Do not modify any of the other source files, the makefile, or any of the code in `hw5_conv.cu` below the "No change to code after this point" comment.** If your code in `hw5_conv.cu` fails to compile with the other provided source files and makefile, you may receive zero on the related questions.

The homework comes with several test cases in PPM image format: `checker20.ppm`, `checker40.ppm`, `booby.ppm`, `stippling.ppm` which can be accessed by symbolic link using the instructions above or downloaded. The two checker cases are small images (40x40). We recommend you debug your code on these small images before moving on to larger images.

The binary `hw5_conv` contains several options. Here are explanations with examples (for additional details, call `./hw5_conv` without any arguments):

- Basic execution of a convolution kernel:

  ```
  ./hw5_conv -f ./result.ppm -c 1 ./images/stippling.ppm
  ```

  This command would take `./images/stippling.ppm` as the input, run `conv1v_basic` (specified by `-c`) on it and write to `./result.ppm` (specified by `-f`). Here is a list of `-c` options: 0: `conv1h_basic`; 1: `conv1v_basic`; 2: `conv1to2_basic`; 3: `conv1h_tiled`; 4: `conv1v_tiled`; 5: `conv1to2_tiled`.

- Running a convolution with a specified sigma value:

  ```
  ./hw5_conv -f ./result.ppm -c 1 -s 5 ./images/stippling.ppm
  ```

  Here the sigma is 5 (specified by `-s`). It controls the convolution stencil and hence the degree of blurring. The larger the value, the blurrier the result. Your code will be tested only with the default value for sigma.

- Running a convolution for multiple iterations:

  ```
  ./hw5_conv -f ./result.ppm -c 1 -n 1000 ./images/stippling.ppm
  ```

  This command would repeat the operation 1000 times (specified by `-n`). This option is provided to improve timing estimates when running small images.

- Comparing results against the reference images:

  `./hw5_conv -f ./diff.ppm -t -c 1 ./images/stippling.ppm`

  This command would take `stippling.ppm` as the input and run `conv1v_basic` on it. It would check the result against the reference image (`stippling_1_ref.ppm` in this case). If your output matches the reference image, you will receive a textual success message. Otherwise, the system will list pixels (at most 10) which disagree: `<("x", "y"), "float pixel value">`, where `("x", "y")` is the position of the error pixel and `"float pixel value"` is the value of your output. The output file specified by `-f` (`./diff.ppm` in this case) would be the difference image between your output and the reference image. This option is provided to help with debugging.

Chenxi has provided a few other tips that may be useful for debugging:

- To view your results as an image on the `linXX` machines you can use the `display` command. In order to view the results remotely, you must have support for X Windows on your local machine. Linux and OSX machines naturally support X Windows; for Windows you can download the Xmanager software from `http://my.cs.ubc.ca`. Make sure you enable X forwarding through whatever `ssh` client you are using to remotely connect.
- To view your results on your own machine, you will need a local viewer capable of handling `.ppm` files. If the `display` command (part of the ImageMagick package) is not available, the open source GIMP package, recent versions of the emacs text editor, or the commerical Photoshop package can be used.
- If you are developing code on your own machine, you will need to adjust the library path (specified in `Makefile` with the variable `L_DIR`) so that the CUDA runtime library `cudart` can be found. But be sure to test your code on the `linXX` machines prior to submission, since we will be running your code on these machines for grading purposes.
- You can call `printf` from CUDA kernels, but be careful—printing from all threads may generate incomprehensible results and/or hang / crash the process.

The Power Iteration Template Code

This question involves only the file `hw5_blas.cu`. To build, use the command

```
nvcc -lcublas hw5_blas.cu -o hw5_blas
```

to generate the binary `hw5_blas`. This compilation will only work if your `LD_LIBRARY_PATH` environment variable in your shell is set correctly; for example, on the `linXX` machines it

should include /cs/local/lib/pkg/cudatoolkit/lib64). You can see the value LD_LIBRARY_PATH by typing echo $LD_LIBRARY_PATH at the prompt. If it does not exist or has no value, you can set it (again on the linXX machines assuming that your account is using the default bash shell) with the command:

```
export LD_LIBRARY_PATH=/cs/local/lib/pkg/cudatoolkit/lib64
```

If it has a value which does not include the CUDA path, you can add the CUDA path (again on the linXX machines assuming that your account is using the default bash shell) with the command:

```
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/cs/local/lib/pkg/cudatoolkit/lib64
```

You can either run the appropriate command manually every time you log in, or you can add it to your ~/.bashrc file so it gets run automatically every time you log in.

The sample data for this question are very large. If you are working on the linXX machines, we recommend that you create a link to our copy of these files rather than copying them into your own directory. To create a (symbolic) link to our image directory in your current directory, use the command

```
ln -s ~cs418/public_html/2016-2/hw/5/matrix .
```

Note that there is a space between "/matrix" and the final "." in this command. Once you have the sample data in a subdirectory matrix/, a typical call will be

```
./hw5_blas 1024 matrix/power_A_1.500_1024.data matrix/power_b0_1.500_1024.data 500
```

The data files' names specify either a matrix ("A") or initial vector ("b0"), the actual maximum eigenvalue (such as "1.500") and the size of the matrix or vector (such as "1024"). Note that you still need to tell hw5_blas the size of the matrix since it does not attempt to parse the data file names.

Why?

Convolution: The convolution operation is often encountered in numerical computing, although not always under that name; for example, the finite difference stencils widely used to approximate derivatives when solving partial differential equations are a form of convolution. Convolution is used in computer vision, signal processing, and many, *many* other problem domains.

For the purposes of this parallel computing course, convolution is a great example of a data-parallel programming pattern. The problem gives you some experience with using tiling on a GPU—this is critical for achieving high throughput in most GPU kernels. By getting some experience with convolution with this problem, you'll be more likely to recognize it when it comes up in other contexts, and now you will know how to get started on an efficient, parallel implementation. Keep in mind, however, that both convolution and tiling in two dimensions or higher require much more

detailed index manipulation. For this question we applied convolution in a very simple context—image blurring—because it is easy to set up the problem and visualize the output, but the kernels we developed are applicable in the much broader context.

Power Iteration: From the perspective of the course, the main purpose of this question is to provide an opportunity to practice using the cuBLAS library. However, the power method is used industrially, typically to find a single eigenvalue / eigenvector (or sometimes a small number of them) for large sparse matrices because it requires nothing more than marix-vector products and so the matrix need not be stored explicitly. A "famous" example of the sparse-matrix version is Google's page-ranking algorithm. For this problem, we focus on dense matrices, because it makes the coding much simpler.

For dense matrices and large numbers of eigenvalues / eigenvectors more sophisticated methods are usually used in practice, but we wanted to keep the problem simple. This is, after all, a course on parallel computation. The number of iterations (matrix-vector products) needed in the power method is inversely proportional to the size of the gap between the largest and next largest eigenvalues, and for large matrices that gap can be very small. If a large number of iterations is needed, roundoff error can accumulate; consequently, the power method is rarely performed in single precision arithmetic. In order to keep things well behaved, we are carefully constructing the test matrices for this particular problem so that the gap is relatively large and only a few iterations are needed.