Computer Architecture Fall 2023
Assignment 4: Explicit Parallelism on the GPU

Deadline: Wednesday, December 20, 2023

Within this assignment you will write GPU implementations (exploiting Data-Level Parallelism) of image processing kernels using CUDA. Next to writing the required CUDA kernels, you need to add memory allocation and memory transfer function calls. On Brightspace you can find a concise introduction to CUDA programming.

Similar to the third assignment, this assignment is structured around a number of tasks. These all involve the implementation of performance improvements, followed by experiments to analyze and quantify the performance improvements. As deliverables we again expect a report and the modified materials that were used to conduct the experiments.

For tasks 1 and 2 skeleton codes are provided. The skeleton code contains functional CPU-versions of the image processing kernels to optimize. To load and manipulate images the same small support library is used as in assignment 3. The same `image_get_pixel` macro will work correctly in CUDA kernels for host as well as device pointers. However, instead of the type `rgba_t` the type `float4` is used. Both types do have the same `x, y, z, w` member fields. You may assume that the input images will always have dimensions that are a multiple of 64 (but are not necessarily square). This property will greatly simplify optimization, since many corner cases do not have to be checked and do not have to be dealt with.

Environment

A suitable version of CUDA is not installed on the LIACS workstations by default. We have prepared installations of this software in the `ca2023` environment. This environment can be enabled as follows:

```
source /vol/share/groups/liacs/scratch/ca2023/ca2023.bashrc
```

Note that `remotelx` (`huisuil`) is *not* equipped with a GPU. However, many LIACS lab computers are equipped with an NVIDIA GPU. Exceptions are rooms 307 and 309 (which are not LIACS lab rooms) and a small number of workstations with a "No NVIDIA GPU" sticker. You can work on your CUDA project remotely, by connecting to a computer in the LIACS lab rooms using `ssh`. In case you want to work on the assignment at home on your own hardware, you need a NVIDIA GPU with CUDA installed.

Task 1: Exploring Thread Block Configuration

For the first task you will work with the 'grayscale' kernel. The main goal is to become familiar with CUDA and to explore the influence of different thread block configurations on the performance. The file `task1.cu` contains the skeleton code to use. This skeleton also includes a single image copy CUDA kernel as an example.

To help test your implementation, we have included a test mode in the skeleton code. This command will apply your 'grayscale' kernel to the specified image file name and compare the output of your GPU kernel to that of the CPU kernel (reference implementation). This way, you can test your implementation without repeatedly having to transfer the output image over SFTP. Once the test mode indicates the images match, we recommend you to perform a visual inspection of the result as well. The test mode is enabled with the `-t` command line option, example:

```
./task1 -t /path/to/lab3data/color4096.png
```

Make sure to use a color image as input! By removing the `-t` option and adding `-r 5`, the program will measure the performance of the CUDA kernel and repeat this measurement 5 times.

The task is to implement a number of variants of this function to answer a set of research questions. Important: place the different variants in separate functions, use `#define`s, or perhaps place the different variants in separate files. The research questions to address are:

RQ1. Implement a CUDA kernel that processes 1 pixel per thread and uses a thread block size of 8×8. This is a simple one-to-one implementation of the provided CPU kernel. Measure the performance of the GPU kernel on images of increasing size and observe the scaling trend.

RQ2. Write a variant that processes more than one pixel per thread. Does the performance improve?
Important: ensure that each pixel is only processed once!

RQ3. Investigate the influence of different thread block configurations on the execution time of the kernel.

For the performance measurements it is important to repeat the experiments and to look at the processing time of individual frames/images. In this case 'compute time' only includes the time required to execute the CUDA kernels on the GPU. It does not include the time required to perform memory allocation and memory transfers, which is fine, because we will not consider it in this task.

Task 2: Shared Memory

We now investigate whether the use of the shared memory present in each Streaming Multiprocessor can help us to improve the performance. The GTX 1050 Ti present in the lab room computers has 49 KiB shared memory available per thread block. Within this task, we will use the 'tile composition' kernel, which this time places the tile on the entire background image, instead of just the diagonal as was the case in assignment 3.

The program for task 2 ('tile composition') accepts a directory of images as input argument rather than a single filename. This is because this program will be extended into a program that can process images in batches for tasks 3 and 4. We have provided a directory with frames of a video stored as separate PNG files: `/vol/share/groups/liacs/scratch/ca2023/lab3data/frames/`. The second argument specifies an image file containing the tile to use.

For task 2, use `task2.cu` as the skeleton program with the `-1` option to only process a single image (the batch size already defaults to 1). You can use the first of the provided 'frames' as the background image and a tile of 64×64 pixels. The command becomes:

```
./task2 -1 /path/to/lab3data/frames/ /path/to/lab3data/tux64.png
```

Furthermore, note that also the `task2` program has a `-t` test option, which will only process the first image found in this directory and always runs with a batch size of 1.

With this information provided, investigate the following research question:

RQ1. Investigate the effectiveness of placing the tile in GPU shared memory. Develop two variants of the CUDA kernel: one that does not use shared memory and one that does. Compare their performance.

⚠ When placing the tile in shared memory, do not already perform the pixel unpacking, so that we really investigate just the impact of using shared memory. Make sure to architect the kernel such that data loaded into shared memory is used multiple times – which should be feasible as the tile is placed onto the background multiple times.

Task 3: Batched Processing

Up till now we have designed the CUDA kernels such that they only work on a single image. What if we process multiple frames per kernel invocation? This reduces the number of kernel launches, which might be beneficial. More in particular, we are interested in finding out whether this increases the effectiveness of the shared memory optimization.

To work on this task, make a copy of `task2.cu` to `task3.cu`. Within the new files, created batched variants of both kernels from task 2.

To help you test the implementation, the skeleton code allows the number of images in a batch to be configured using the `-b` command line option. The following command processes a single batch of 8 images and repeats this 3 times:

```
./task3 -1 -b 8 -r 3 /path/to/lab3data/frames/ /path/to/lab3data/tux64.png
```

If you would like to reconstruct a video from the frames in the output directory, refer to Appendix A.

The research question to be investigated is:

RQ1. If we process multiple images in batches, does the effectiveness of the shared memory optimization increase? Measure the performance of the new batched kernels and compare the results. Do both show a similar improvement in performance, or does one improve more than the other?

Task 4: Overlapped Data Transfer / Compute

So far, we have only assessed compute time and not the time required for data transfer. However, the overall execution time comprises compute as well as data transfer time (for input and output!), with the time taken by data transfers often being significant. This leads to the following research question:

RQ1. Can the overall execution time be reduced by overlapping data transfers and compute when processing all frames of the video? Implement such a variant, and compare its performance to your program for task 3. When running the programs, remove the `-1` option such that all frames are processed.

This final task is to be based on your code for task 3. Make a copy of your program with the batched kernels from task 3, name it `task4.cu` and modify the code such that data transfer of the next batch of images happens at the same time as running the kernel for the current batch of images. Such functionality can be implemented using, for example, CUDA Streams. Note that the results also need to be transferred back to the host. The skeleton code measures load time, compute time and overall execution time. The data transfer time is not included in the load or compute time, but is included in the overall execution time. So, by properly implementing the overlapped data transfers, we expect the overall execution time to decrease.

Submission and Grading

Teams may be formed that consist of at most *two* persons. The **deadline** is Wednesday, December 20, 2023. Submit your assignments according to the instructions below.

The maximum grade that can be obtained is 10. The grade is the sum of the scores for the following components, maximum score between square brackets:

- [**2.0 out of 10**] Overall quality of the report.
- [**3.0 out of 10**] Task 1 (Exploring Thread Block Configuration).
- [**2.0 out of 10**] Task 2 (Shared Memory).
- [**2.0 out of 10**] Task 3 (Batched Processing).
- [**1.0 out of 10**] Task 4 (Overlapped Data Transfer / Compute).

The following needs to be submitted:

- Source code of the implemented optimizations. It *MUST* be based on the starting point (template) of this academic year, otherwise your submission will not be graded.

  Note again: we want to see the code for each task individually. Please run `make clean` before making the archive; do not include large files (such as executables or data files). The archive should be smaller than 250 KiB.

- A well-organized report (preferably written in LaTeX) in PDF-format. Refer to the guidelines as noted in the description of assignment 3. For this particular assignment it is also important to mention GPU model and CUDA version used for implementation and experiments.

Ensure that all files that are submitted (source code, report, etc.) contain your names and student IDs! The source code tarball and PDF-report are to be submitted as *separate* files. Please create a "gzipped tar" archive of your source code:

```
tar -czvf assignment4-sXXXXXXX-sYYYYYYY.tar.gz assignment4/
```

Substitute `XXXXXXX` and `YYYYYYY` with your student IDs.
Use the filename `report-sXXXXXXX-sYYYYYYY.pdf` for your report.
Submit the tar-archive and your report through the Brightspace submission When submitting the files in Brightspace, write your names and student IDs in the text box. *If you work in a team, ensure only one team member submits the assignment, such that there is a single submission per team!*

The use of text or code generated by ChatGPT or other AI tools is **not allowed**. You are required to implement the requested code transformations **yourself**, and to write your **own** text for the report. All submitted reports and source code will be subject to (automatic) **plagiarism checks** using Turnitin and/or MOSS. Suspicions of fraud and plagiarism will be reported to the Board of Examiners.

A Reconstructing a video from individual frames

The result of the program will be a directory containing the modified frames. If you want to see the result as a video, you can use the command (ffmpeg must be installed):

```
ffplay -framerate 30 -i frame%04d.png
```

You can also reassemble the modified frames into a new MP4 file. This can be achieved with the following command:

```
ffmpeg -r 30 -f image2 -i frame%04d.png -vcodec libx264 -crf 25 \
    -pix_fmt yuv420p -s 1920x1024 test.mp4
```

After completion of this command, open `test.mp4` with a video player.