Weekly Assignment 4
Parallel Functional Programming

Troels Henriksen and Cosmin Oancea
DIKU, University of Copenhagen

December 2021

Introduction

The handin deadline is the 23rd of December.

The handin is expected to consist of a report in PDF format of 4—5 pages, excluding any figures, along with an archive containing your source code. The report should contain instructions on how to run and benchmark your code.

Task 1: Matrix Inversion

An $n \times n$ square matrix A is said to be *invertible* if there exists a unique matrix A^{-1} such that

$$AA^{-1} = I_n$$

where I_n is the $n \times n$ identity matrix. We also call such an invertible matrix A *non-singular*, and is characterised by having a determinant $|A|$ different from 0. Computing A^{-1} is called *matrix inversion*.

For this exercise you will be implementing matrix inversion based on the Gauss-Jordan algorithm (without pivoting, for simplicity).

In the Gauss-Jordan algorithm, we augment A with I_n to the right, forming an $n \times 2n$ matrix typically written $[A|I_n]$. We then perform Gaussian elimination on $[A|I_n]$ to compute the reduced row echelon form, which produces a matrix $[B|A^{-1}]$, from which we can then extract the desired A^{-1}.

For this task, you are given a function `gaussian_elimination` for performing Gaussian elimination (without pivoting). It's a simple (bad) implementation that is not numerically stable, but it will do for our purposes. Your task is to

1. Implement matrix_inverse: augment the input with the identity matrix, call gaussian_elimination, then extract the inverted matrix. **Hint:** in order to satisfy the type checker, you will likely have to define a variable let n2 = n + n at the start of matrix_inverse and then use this `n2` with either concat_to or in a replicate followed by in-place updates.

2. Write a `main` function that maps matrix_inverse across an array of k square matrices. That is, the input to `main` must have type `[k][n][n]f32`, for some `k` and `n`.

3. Answer: how many levels of parallelism does this program have? Based on the incremental flattening rules, approximately how many versions will be generated?

4. Using a GPU backend (`cuda` or `opencl`), benchmark your program on a range of inputs. Don't worry about whether the input matrices are in fact invertible. Include cases with many small matrices ($n \leq 16$).

5. Use `futhark autotune` to optimise the threshold parameters. How does this affect performance for each of your datasets? Why do you think that is?

Task 2: Polyhedral Transformations

This task refers to polyhedral analysis, please see `L7-polyhedral.pdf`; the task is also summarized by the last slides in said document.

Please install the `islpy` library by running `pip install -user islpy`.

Your task is to encode in the polyhedral model three code transformations:

- loop interchange (a.k.a., permutation), in file `code-handout/poly-transf/permutation.py`;
- scaling, in file `code-handout/poly-transf/scaling.py`;
- reindexing, in file `code-handout/poly-transf/reindexing.py`.

Please follow the hints and instructions in said files. Your task is to fill in the blanks—in each file—the implementation of:

- the iteration domain,
- the original (sequential) schedule,

- the read/write access relations, and
- most importantly **the transformed schedule**.

Please include in your report:

- your (full) implementation of the (i) iteration domain, (ii) original schedule, (iii) read/write access relations and (iv) the transformed schedule (i.e., only those full lines; do not include the rest of the handed out code);
- a brief explanation of the encoding of the transformed schedule (for the others, the code should be self explanatory);
- for the *permutation (interchange)* and *reindexing* transformation, can you devise a schedule that tests that the transformed loop is parallel? (If so, please report it as well.)