CS3014 Lab 1: Parallel Multichannel Multikernel Convolution

19th February 2019

Modern **deep neural networks (DNNs)** are one of the most successful types of artificial intelligence. There are many artificial intelligence competitions each years in areas such as image processing, and pretty much all the winning entries to these competitions are DNNs. An important type of DNN that is very successful at image classification and other image processing tasks is the **convolutional neural network (CNN).**

A CNN consists of **a directed acyclic graph of "layers"** which are typically selected from a small number of standard layers such as activation layers, pooling layers, fully-connected layers, and convolutional layers. CNNs require very large amounts of computation to process and input image, and most of this time is spent in convolution layers.

The convolution layers are similar to, but nonetheless different from, standard two-dimensional convolution. In CNNs the images have multiple "channels" which might be red, green and blue, or simply notional different pieces of information about a single pixel of the image. In addition, CNN convolution is always performed with multiple different convolution kernels, each of which has multiple channels. We therefore refer to it as multichannel multikernel convolution.

The purpose of this labs is to better understand vectorization, code optimization and parallelization by parallelizing and optimizing an efficient convolution routine.

Our target machine is stoker.scss.tcd.ie. This machine has four processors. Each processor has eight out-of-order pipelined, superscalar cores. And each core has two-way simultaneous multithreading.

To compile with SSE on stoker you should:
- #include<x86intrin.h> in your C/C++ code
- compile your program with: gcc -O3 -msse4 file.c

To comile with OpenMP:
- add the flag -fopenmp.

To write an efficient multichannel multikernel convolution routine you will need an efficient basic algorithm, but it is also important that you take account of issues such as locality of data access and multiple available processor cores.

Your routine should be written in C/C++, and work on the same data structures as are used in the sample code. You may consult books, papers and online sources, but **ALL SOURCES SHOULD BE ACKNOWLEDGED IN YOUR SUBMISSION**. All the submitted code should be your own.

The work for this lab should be carried out in pairs, and each pair of students will make a joint submission. The submission should take the form of:

(1) a working, well-written, commented piece of code
(2) a document of a couple of pages describing the efforts made to make the code efficient.
(3) a set of slides in PDF format that document your parallelization and optimization strategy, and list you execution times on the standard input sizes.

You will present these slides at the lecture on Thursday of Week 9 of the semester as part of your team. Everyone on the team must speak as part of the presentation, and everyone should be capable of answering questions about the code. Each team will have a speaking slot of around 120-150 seconds. This time includes setup and switching.

Please give the PDF file that contains your slides a filename based on the first names of those who are part of the team. Please use the first name that you are registered under in College rather

than another first name (with apologies to those who use a different name or a different form of their name in practice). Please put the team member with the **alphabetically** first first-name first. (Yes, this will cause a bias with larger teams tending, on average, to have an alphabetically earlier filename, but hey!).

I will put all the slides of PDFs on my laptop for presentation. Unfortunately we cannot deal with other file formats, nor with animations or anything else that is not simple PDF.

Your routine should be capable of operating efficiently on a range of matrix sizes and shapes. The range of sizes we will consider are:

image_width: 16..512
image height: 16..512
kernel order: 1, 3, 5, or 7
number of channels: 32..2048 (always powers of 2)
number of kernels: 32..2048 (always powers of 2)

You must use the following test harness to test and time your code:
https://www.scss.tcd.ie/David.Gregg/cs3014/labs/conv-harness.c

The marks you get for this lab will depend on two things. First, the mark will depend on the code itself --- its correctness, quality and the optimization ideas it contains. Secondly your mark will depend on the running time of the code. The faster your code compared to that of other teams, the higher the mark you will get.