ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Dr. P. Hadjidoukas, Prof. C. Papadimitriou
ETH Zentrum, CLT E 13
CH-8092 Zürich

Project guidelines

Issued: 15.05.2017

- In these scripts, we outline five computational engineering problems. Choose one and work either individually or in a group of two people. Communication is allowed to the extent that you do not copy the work of other groups.
- You are encouraged to contact one of the TAs in order to arrange a meeting to discuss your chosen project. These meetings are meant for clarifying and detailing the projects, give early feedback on your approach, *not* for correcting your code.
- In evaluating your work, we will consider your ability to analyse the problem and the hardware at your disposal, to appropriately apply the principles taught during the whole HPCSE class, and to report your reasoning and findings.
- The report (including text, code, figures) needs to be emailed before the day of the exam. If working in a group of two, each student has to write an *individual* report.

General remarks

The purpose of the projects is to give you an insight into solving a serious computational task, which due to its scope does not fit into the weekly assignments and/or lectures.

During both HPCSE classes, you have been exposed to several programming techniques, ways of analysing performance, as well as some specific numerical algorithms, which are either widely used in practice or interesting because of their properties.

Use the projects as means of improving your programming skills and high-performance optimization abilities. The knowledge that you've gained in class should be applicable to all the listed tasks, but there might be some algorithm-specific optimizations that you can only apply in your case. Try finding them.

1 Report structure

Your report should be reflective of the work you have done and should generally follow the structure of a case study of the given numerical algorithm and its implementation. As such, it should broadly cover the following areas:

1.1 Background

This section should include the motivation (i.e. why the method is useful) and the mathematical formulation. Some of the latter will be similar to the project description itself, but you might find it useful to recap it and include more analysis and derivations if you found them necessary for your own understanding. Note, however, that the background section should not include any results that you have learned of as you worked on the project. Think of the background section as one that answers the question of **how the world looked like before I started working on the project**.

Length: up to 10% of the whole report, excluding references.

1.2 Baseline implementation

This section should clearly and succinctly describe your initial (serial, scalar) implementation.

It should contain the **explanation of how your code-base is structured**, the outline of important routines/functions and where they are location. If you had to make a particular design decision that had a major impact on the development, briefly describe it and provide justifications.

Where applicable, you should report on the **verification** of your algorithm. If you know that it should converge with some rate, include a plot that confirms this, by either comparing with a trustworthy reference implementation or by comparing with an analytic solution.

This section should conclude with the **first performance analysis**. The *apriori* analysis should include the computational intensity and location on the roofline plot. If the result is ambiguous, you should clarify how you counted the operations and memory transfers by pointing to specific code snippets.

The *aposteriori* analysis should measure the runtime, observing how it corresponds to the asymptotic algorithmic complexity, and the performance, observing what percentage of peak performance is reached. Both of these quantities are best presented by the means plotting them against the problem size.

You may find the following remarks useful:

- Make sure that the peak-performance is calculated for the machine you are using, and that it is the actual peak, that is, take into account that we are in single core, scalar mode.
- You may either work with flops per cycle or flops per second.
- Pay attention to how you count the data transfers (8B for `double`, 4B for `single`).

We suggest that you write this section before actually attempting further optimizations.

Length: up to 30% of the whole report, excluding references.

1.3 Optimization results

Depending on your analysis of the algorithm you will apply different optimization strategies, as discussed in the later sections of this guide and throughout the course. You might discover that some techniques are independent of each other, or that they are conflicting. Some might bring an improvement in one area, but cause a bottleneck in another. It might even happen that an optimization technique causes a runtime speedup, but decreases the performance (because the number of flops changes).

What we are trying to illustrate is that there are numerous outcomes of your attempts and we encourage you to try and report on as many of them as possible. Even if a particular technique did not work, do report on it.

Every technique that you attempt should be clearly explained, followed by runtime speedup (speedown) and performance plots. If an optimization did not work, try to provide your reasoning. Make all the measurements **relative to the same baseline implementation**. In explaining the technique, visual aids can be of great assistance. Use diagrams and computational flow graphs. Animations can also be very useful.

Lastly, pay attention to report on these:

- What compiler, which version and what optimization flags were used?
- How many experiments did you run to measure your timings?
- Were your timings done in cold or warm cache scenarios? If in cold cache, briefly state how you achieved it, if in warm cache, justify why.

Length: up to 50% of the whole report, excluding references.

1.4 Conclusions

If the background answered the question how the world looked like before your work, the conclusions should answer how it looks like afterwards. Of all the optimization strategies, mention the ones that stand out, either in positive or negative way. Be evaluative of the algorithm itself, i.e. how well it lends itself to optimization strategies. Clearly state the contributions to the case study. A good report will also end with possible questions that you discovered, but whose answers did not fit this particular work.

Length: up to 10% of the whole report, excluding references.

2 Optimizations

We encourage you to go review the lecture material from both classes. In general, however, a rule-of-thumb approach should proceed in stages:

1. scalar optimizations (single scalar assignments, reordering and unrolling for pipelining, varying memory layouts, blocking),

2. vectorization,
3. shared-memory parallelization,
4. distributed-memory parallelization or offloading to a GPU.

Some algorithms consist of only one major computation, while some are multi stage. You have to figure out which stage is the bottleneck, otherwise you will be aimlessly optimizing everything. Do look into valgrind's tool callgrind in combination with KCachegrind/QCachegrind or Intel VTune. For vectorization, you might either do it by hand, but you can also look into Intel's SPMD compiler `icpc`[1]. For loop unrolling you can help yourself with scripting languages, such as Python in combination with some templating library (such as Jinja).

3 Report style, formatting and length

The report should be written in English in academic style. We do not prescribe any specific template. We suggest that you use LaTeX. A good template to use can be found here.[2]

If you want to make your report interactive, have a look at Scholarly Markdown. Among other things, it allows you to include videos, and the report is read in a web browser.

We ask you that you make your plots using vectorized graphics, and that you make the font sizes large enough. If you use `matplotlib`, you may look into its new feature (stylesheets[3]) which corrects a lot of the visual shortcomings of the default settings.

4 Academic integrity

Up to two people can work on the project together in a group, but each person has to hand in the report individually.

Communication between groups is allowed, but any sharing of the code is strictly prohibited. Cite all the references that you used. The citation style is up to you, but has to be consistent.

Every report has to be accompanied with a signed Declaration of originality [4]

[1] `https://ispc.github.io`

[2] `https://people.inf.ethz.ch/markusp/teaching/263-2300-ETH-spring15/project/report.zip`

[3] `http://matplotlib.org/examples/style_sheets/plot_ggplot.html`

[4] `https://www.ethz.ch/content/dam/ethz/main/education/rechtliches-abschluesse/leistungskontrollen/declaration-originality.pdf`