An Introduction to High Performance Computing

Crash-course on Writing Reports

Andrei Poenaru

17 October 2019

University of BRISTOL

Why?

- This is a report-based coursework, so the overall quality of your report is **very** important
  - You need to present what you did—just submitting the code isn't enough
- Not many occasions to get feedback on your writing so far
  - We see a number of common mistakes repeated every year
  - Start thinking early about how your reader will perceive your text
- This session will touch on a wide range of points
  - Not all may be applicable to each of you
  - Don't do things just because they're mentioned here—there isn't a *single* right way, and you need to combine good writing principles with *your* style

What you want to avoid

I. INTRODUCTION

This assignment involved taking the lattice Boltzmann code provided and applying serial optimisations, as well as the OpenMP API to parallelise the code, enabling it to run on all cores within both sockets of a BCp3 node.

Given that OpenMP is focused on shared memory paradigms, whilst OpenMPI (the standard used in the previous coursework assignment) is centred around a message passing paradigm, the strategy used within OpenMP is vastly different; there will be less focus on the sending and receiving of messages with the halo exchange, and more focus on running multiple blocks of code in parallel, with commands such as pragma.

This report will include detailed descriptions of the serial optimisations I chose to make, as well as my method of parallelism, and analysis into performance. Any runtimes or speed-ups referenced will be for the 128x128 image, unless stated otherwise.

II. SERIAL OPTIMISATIONS

My first step involved compiling and running the program – the average runtime was 58.8 seconds.

I then immediately optimised the compiler; I decided to implement both gcc and icc, and compare by using the equivalent optimisation flags on both compilers. Interestingly, even at the first stage, using icc was beneficial; gcc produced a speed-up of 1.8X to 32.5 seconds, whilst icc produced a speed-up of almost 2X to 30 seconds – I therefore opted for icc in this assignment, using standard compiler flags such as -Ofast for statement optimisation and xHost to tailor the program to the processor.

An initial traversal through the code led me to an obvious bottleneck – the repetition of the two for loops through jj and ii, operating on a large number of cells. It seemed logical to combine the operations within the functions rebound, accelerate_flow, collide and propagate into a single function via loop fusion, as they contained almost identical parameters, and operated on the same data. After some experimentation, I managed to simplify this into a single function, named calculate, which encompassed all of the behaviour of the above functions, meaning I could safely discard the timestep function, which acted as an intermediate step between initialisation and the functions themselves. This greatly reduced cell access (by almost a factor of 3), and incurred a speed-up of around 1.6X.

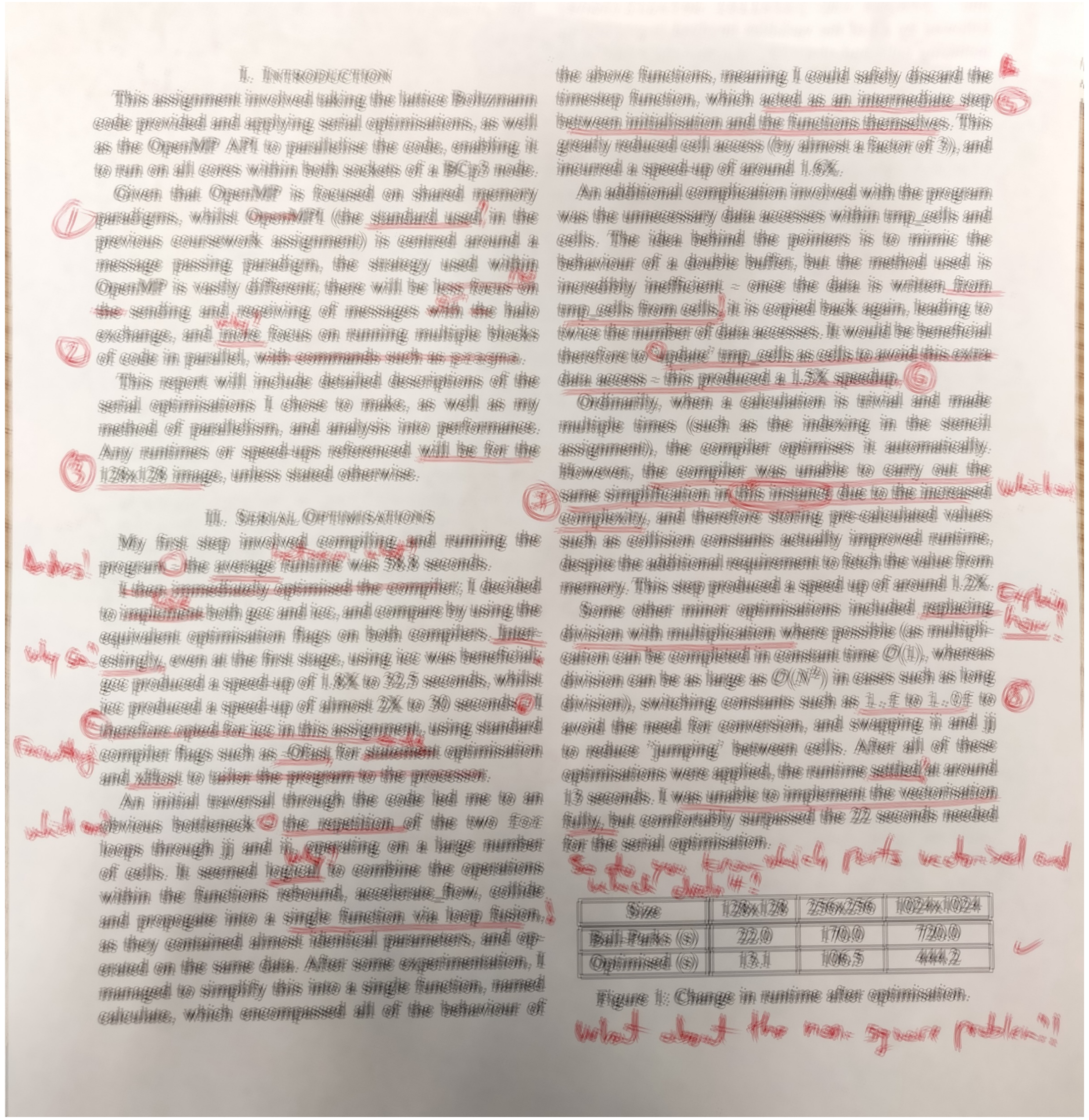
An additional complication involved with the program was the unnecessary data accesses within tmp_cells and cells. The idea behind the pointers is to mimic the behaviour of a double buffer, but the method used is incredibly inefficient – once the data is written from tmp_cells from cells, it is copied back again, leading to twice the number of data accesses. It would be beneficial therefore to 'update' tmp_cells as cells to avoid this extra data access – this produced a 1.5X speedup.

Ordinarily, when a calculation is trivial and made multiple times (such as the indexing in the stencil assignment), the compiler optimises it automatically. However, the compiler was unable to carry out the same simplification in this instance due to the increased complexity, and therefore storing pre-calculated values such as collision constants actually improved runtimes, despite the additional requirement to fetch the value from memory. This step produced a speed-up of around 1.2X.

Some other minor optimisations included replacing division with multiplication where possible (as multiplication can be completed in constant time $O(1)$, whereas division can be as large as $O(N^2)$ in cases such as long division), switching constants such as 1.f to 1.0f to avoid the need for conversion, and swapping ii and jj to reduce 'jumping' between cells. After all of these optimisations were applied, the runtime settled at around 13 seconds. I was unable to implement the vectorisation fully, but comfortably surpassed the 22 seconds needed for the serial optimisation.

Size	128x128	256x256	1024x1024
Ball-Parks (s)	22.0	170.0	720.0
Optimised (s)	13.1	106.5	444.2

Figure 1: Change in runtime after optimisation.



How?

- Formatting
- Structure and layout
- Content
  - Story
  - Language and style
  - Present findings
  - *Explain* findings
- Proofread

Conventions

- Black text contains general explanations and suggestions
- Dark red text shows examples of **bad** writing
  - Avoid similar issues in your own report
  - Emphasis is mine, highlighting the problems
- Blue text shows examples of **good** writing
  - Use similar strategies to improve your own writing

Formatting – why?

- You are not marked on the aesthetics of your report *per se*
  - But good formatting principles have been developed to increase clarity, so following them can only work in your favour

- When you write—in general—you are trying to convince your reader of your knowledge on the topic
  - They are more likely to take you seriously if they can see you've made an effort to write easy-to-read text
  - They are more likely to believe you if you demonstrate you have paid attention to *other* good pieces of writing, e.g. edited articles or research papers

Formatting (1)

- Avoid non-standard fonts
  - Comic Sans is **not** a sensible choice
  - Monospace fonts are **not** sensible choices
  - Times New Roman is not great, not terrible[3.5 R]
  - Arial is a good sans-serif choice
  - Georgia is a good serif choice

- Pick a sensible font size
  - Try to stay between 10 and 12 pt (but references can be smaller)
    - Any lower and text can become hard to read, any higher and it starts looking as if you're trying to cover up a lack of content
  - Stay below 1.5X line spacing
  - If unsure, print the report and make sure it's comfortable to read

Formatting (2)

- Use alternative font styles where there may be ambiguity
  - Use *italics* for emphasis
    - But don’t overuse it, because then it becomes distracting
    - **Bold** is a *different* kind of emphasis—only use it if you’re confident you understand the subtle difference
  - Use a `monospace` font for inline code
    - It makes it clear where the fragment of code starts and where it ends

- Hyphens and (the several types of) dashes are different: -, –, —
  All versions of the application—regardless of whether inter-procedural optimisation was enabled—ran in 325–330 seconds
  - If you don’t know which one is right, rephrase and avoid using them altogether

Formatting (3)

- If you use “typographic quotes”, make sure you use them correctly
  - It’s better to use "plain quotes" than to ”get it wrong”
    The profiler **’gprof’** was used

- But be careful when quoting terms: it may imply a different meaning
  - Please send me your **“notes”** might suggest I don’t think they are doing a very good job at writing notes…
  - If you’re simply introducing terms, use italics instead

Structure – why?

(1) There is a lot of content to convey

(2) Humans have very small short-term memories

(1) + (2) => Provide detailed explanations of all your points. Describe all new terms comprehensively, going into all relevant details. Go back to previous concepts if a refresher is necessary.

Structure – why?

(1) There is a lot of content to convey

(2) Humans have very small short-term memories

(1) + (2) => Provide detailed explanations of all your points. Describe all new terms comprehensively, going into all relevant details. Go back to previous concepts if a refresher is necessary.

(3) There is a page limit

(1) + (2) + (3) => You can’t afford to waste any space, and you don’t want to sacrifice clarity either. You need the reader to be able to follow your text and remember as they go. This can only happen if you divide your content into sensible and appropriately sized chunks.

Structure (1)

- There's no fixed structure for this report (or for many others)
  - This gives you some freedom, but you still need to respect general structure guidelines, e.g. don't skip the introduction or the conclusion
    - In this coursework, keep introductions and conclusions short

- Use a separate section for each core part of your content
  - The stopping points should give your reader a chance to take a breath and summarise to themselves what they've just read
    - If you end a section too early, they won't be able to draw an appropriate conclusion
    - If you end a section too late, they may not realise you've moved on to a *different* subject

Structure (2)

- Clearly delimit your sections, usually through a combination of spacing and bigger font for the heading
  - Keep the headings short, but use them to tell the reader what to expect to find in the upcoming section
  - For short reports, it's better to get straight to the point and avoid signposting

- 3–5 sections is a sensible default choice
  - At two pages, use subsections only if you're confident they add value

- Use plenty of paragraphs to allow your reader frequent breaks
  - If in doubt, (in academic writing) using more paragraphs is better than using fewer

Layout (1)

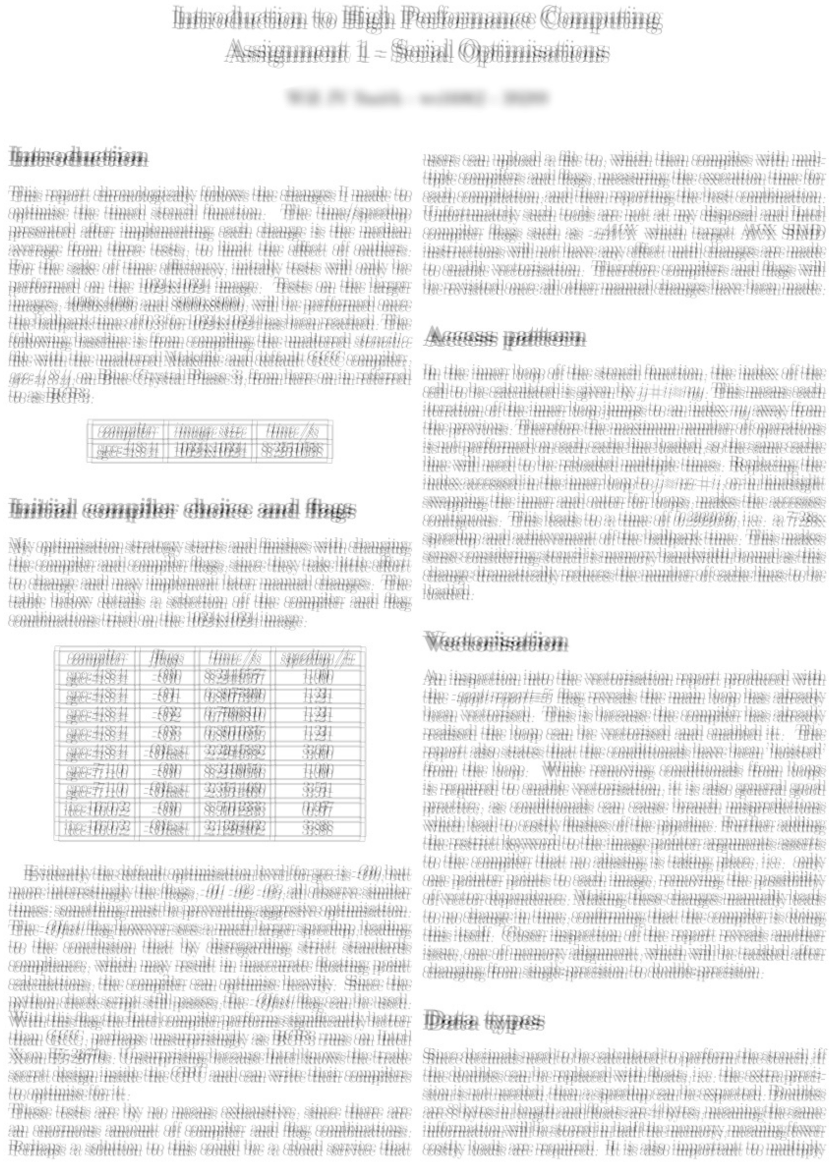
- At 10–12 pt, a single-column layout may make your text hard to read
  - As a rule of thumb, don’t exceed 20 words per line (but even 15 may be uncomfortably long)
  - This is why many academic papers are written in two columns
- If you decide to use two columns, be careful laying them out
  - Readers expect to go through the whole left column first, and only then move to the right column
  - Section breaks should be contained *within* a column

Layout (2)

- You have many choices for embedding figures and tables in your text
  - Pick the right layout based on their sizes
  - In two-column layouts, they can span both columns, but make sure you don’t alter the column flow
  - Don’t use more white space than necessary around your figures

- **Always** caption figures, tables, code snippets, and all other floating elements
  - Number each element, then reference it in the text
  - Figure captions go *below* the image; table captions go *above* the table

Layout: Examples


Introduction to High Performance Computing
Assignment 1 – Serial Optimisations

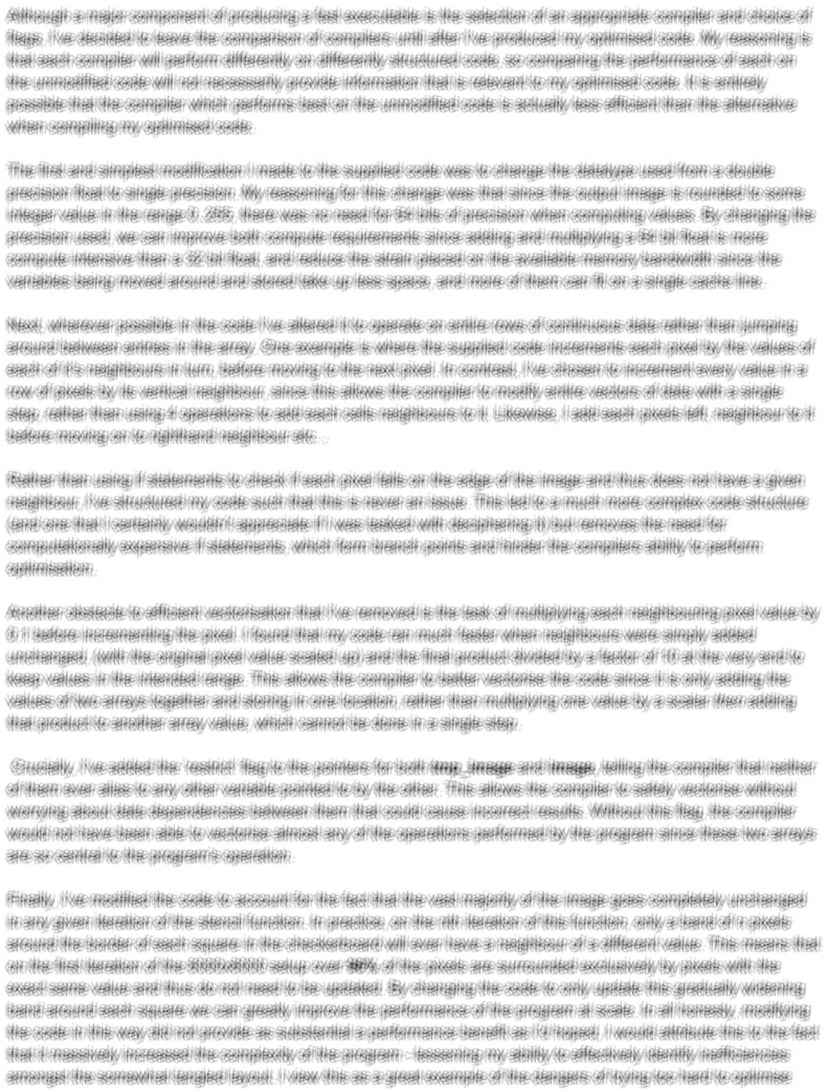
Introduction

Initial compiler choice and flags

Access pattern

Vectorisation

Data types



Content: Story

- Write your report such that text *flows* naturally from one section to the next
  - Some call it *telling a story*, because connections between sections/paragraphs should be self-evident
    - But don't include irrelevant details—these just waste space and don't earn you any marks
      I reverted my changes back to the last correct code commit
  - Often this means presenting your content in non-chronological order

- Many CS research papers are similar in style, so reading some is a good point to start
  - https://arxiv.org/archive/cs
  - But do use your own filter: a lot of papers also have mistakes and bad writing

Content: Style

- Remember this is *academic* writing
  - Don't use informal language
    I thought it would be worthwhile to run a vectorisation report with gcc to **get an idea** of what was limiting the vectorisation.
  - Avoid pompous words and prefer clear, concise, simple sentences
  - Use the appropriate technical terms and be aware of subtle differences:
    GCC is the name of a *compiler suite*, `gcc` is a terminal command

- It is common to want to chain sentences using commas
  - Don't. Stop as often as possible. Delimit statements clearly. Link them up with connectives that clearly point out coherence relations. Remember that your readers only have very limited scratch-space memory.
  - It is better to use a semicolon than to comma-splice, but abuse it and the pacing will upset your readers

Content: Present Findings

- The purpose of the report is to present what you've done
  - Without it, we won't know, even if you *have* done the work
  - Include every detail that you think you deserve marks for
  - Back up **all** claims with evidence
    The code ran twice as fast, **I imagine** due to a higher level of vectorisation
- The report *shouldn't* contain raw data
  - Make good use of tables and graphs to *interpret* the data in intuitive ways
  - No screenshots—extract the relevant content and explain it
- Don't repeat yourself
  - Don't show the same numbers in *both* a table and a graph

Content: Experiments (1)

- Take each of your experiments and present them individually
  Example: compiler experiments
  - Clearly describe your set-up:
    I’m comparing GCC versions __ and __ with Intel version __
  - Do not change more than one variable at a time, because you might introduce confounders:
    I’m comparing **GCC with `-O2`** against **Intel with `-O0`**
  - Quantify performance gains using speed-up
    The version compiled with Intel is 3.5X faster than the GNU one
  - Explain *why* you are seeing these results:
    The compiler optimisation reports show that the Intel Compiler is able to vectorise my loops, but GCC is not

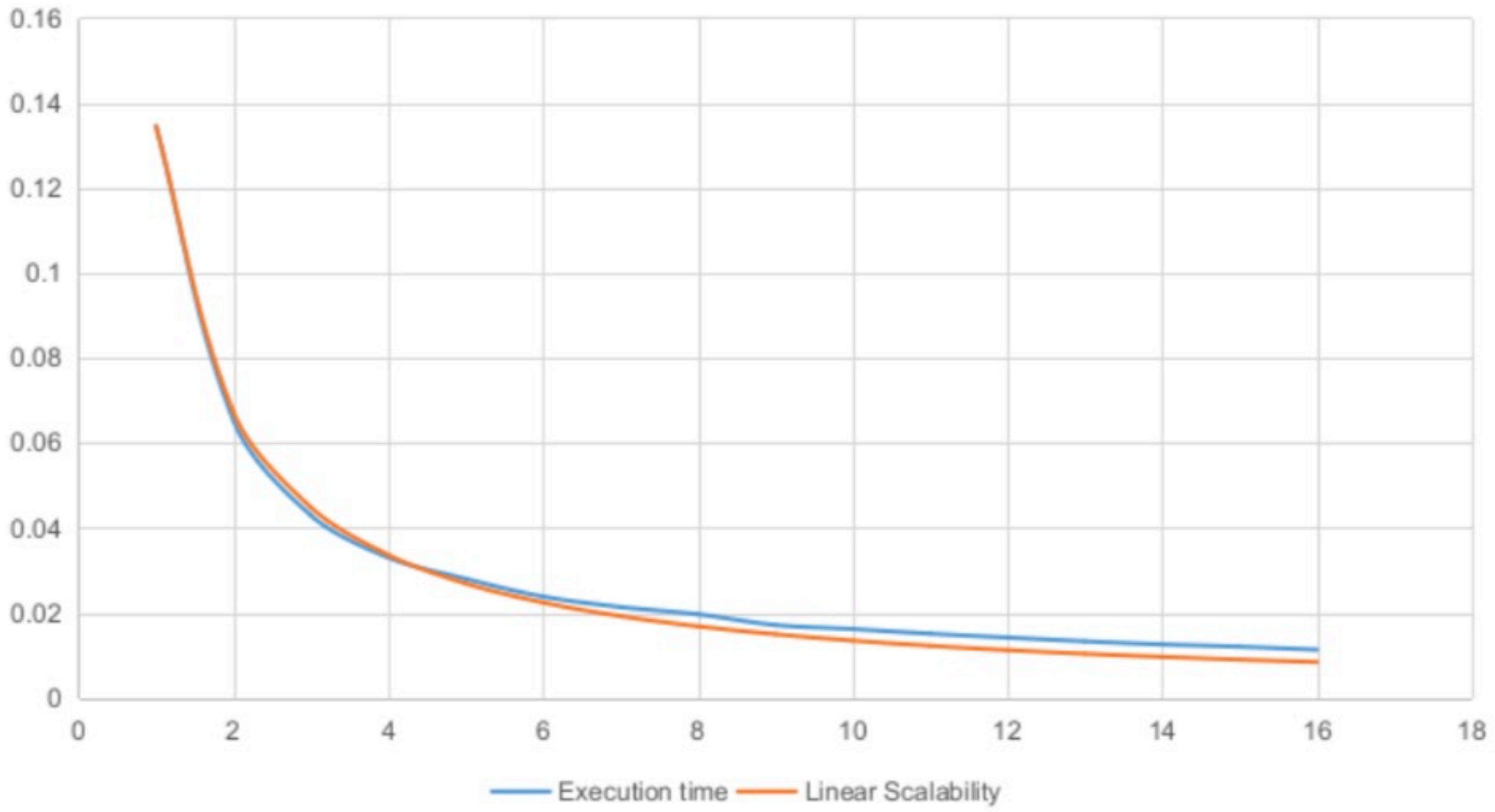
Content: Experiments (2)

- If you have several data sets, e.g. different inputs, evaluate your work on *all* of them
  - For the stencil program, each input image will highlight different performance characteristics
    All runtimes in this report will be based on an input of 1024x1024
- Don't talk about work you don't have to do in the first place
  I used a profiler to analyse the code and see where the biggest bottleneck was. **The result was that the `stencil` method was taking over 99% of the time of the program.**
- When presenting results, choose the right units and precision:
  The run time decreased to **00:00:00**
  My final run time was **0.85 s**
- Be careful of noise:
  My run time **improved from 3.23546 to 3.23518 s**

Content: Figures

- Pick the right type of graph for the data you are showing
- Always have a legend, always have labelled axes, always show the units you are working with, always caption figures
- For data points that are far apart, consider using a log scale
- Choose a sensible colour scheme, avoiding similar tones
  - Keep in mind some readers may be colour-blind
  - If you need to project your graphs, light colours are risky

Figures: Examples (1)



This graph has many issues:

- The data series is discrete, i.e. no real data was recorded for non-integer number of cores, so the data points need markers
- No axis labels
- No units
- x=1 is an important data point and it’s not clearly presented
- x axis doesn’t need to go up to 18

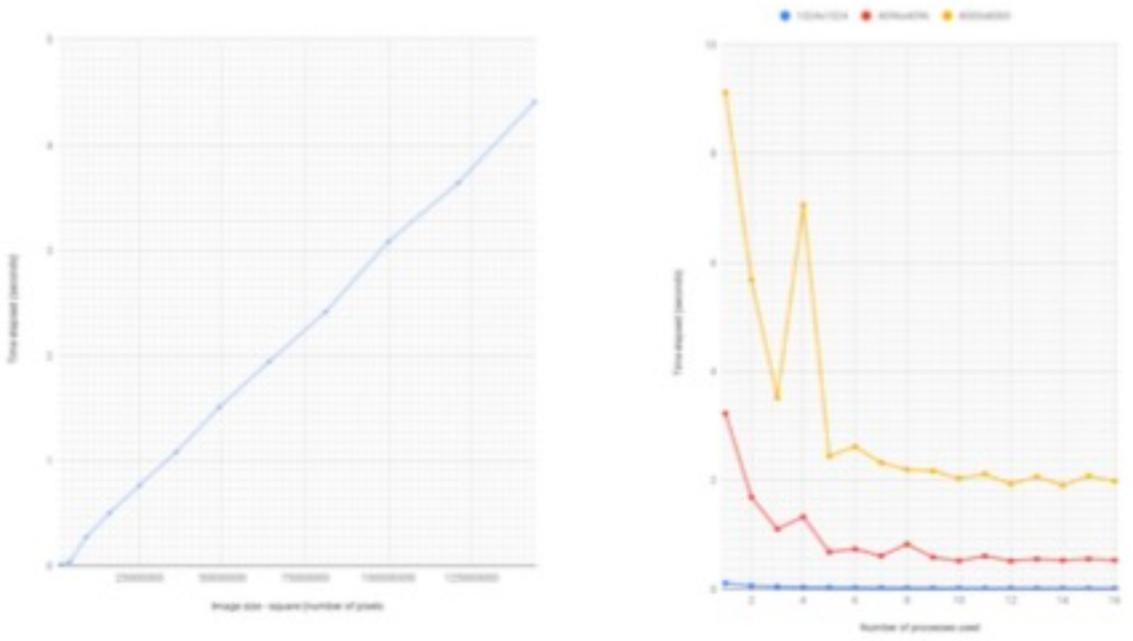
Figures: Examples (2)

ppn=1	ppn=2	ppn=3	ppn=4	ppn=8	ppn=16
runtime: 0.105461 s	runtime: 0.054993 s	runtime: 0.036725 s	runtime: 0.029283 s	runtime: 0.016261 s	runtime: 0.010369 s
	runtime: 0.055470 s	runtime: 0.037238 s	runtime: 0.029787 s	runtime: 0.016328 s	runtime: 0.009788 s
		runtime: 0.036957 s	runtime: 0.029389 s	runtime: 0.017388 s	runtime: 0.023223 s
			runtime: 0.029011 s	runtime: 0.016657 s	runtime: 0.010701 s
				runtime: 0.017182 s	runtime: 0.009990 s
				runtime: 0.016947 s	runtime: 0.009883 s
				runtime: 0.018317 s	runtime: 0.010898 s
				runtime: 0.016482 s	runtime: 0.009698 s
					runtime: 0.009672 s
					runtime: 0.010788 s
					runtime: 0.010604 s
					runtime: 0.010463 s
					runtime: 0.010285 s
					runtime: 0.010199 s
					runtime: 0.010106 s
					runtime: 0.009604 s

Figure 2: Run time for 1024x1024 when using 1, 2, 3, 4, 8 and 16 cores

- This is not a figure—it's a table
- In the original report, this is a *screenshot of a table*
- More than half the page space taken by this object is wasted
- There are too many decimals shown: the runtime is 0.01 s + noise
- Tables should be captioned at the top, not at the bottom

Figures: Examples (3)



(a) (b)

- This graph is taken full-resolution from a paper
- You can’t read the text even on a monitor (where you can zoom), let alone on paper

- Make sure the text on your graphs is readable—print out a copy if you’re unsure

Content: Explain Findings

- It is not enough to show *what* you've done
  - Remember that you are trying to demonstrate understanding
  - You need to explain *why* the effect your are observing occurs
    - This applies for both positive and negative results

- Be careful with concepts you don't have a clear grasp of…
  The conditional `if` statements were replaced by **`for`** loops, which are considerably **less expensive** and **avoid branching predictions**.
- … and with misusing technical terms …
  Restricting pointers might also help with vectorisation.
  Conditionals in a for loop **break the pipeline** in the processor.
- … they both work against you

Content: Language

- If your writing is riddled with language mistakes, it is natural for the reader to suspect the rest of your work may also be flawed
  - In writing—as in many other aspects of life—first impression *does* count
  - A proofreading pass or two go a long way to improving the perceived quality of your write-up

- Common mistakes (from past years) to be aware of:
  - *its* != *it's*
    - even though it may feel natural to apply the same transformation as in *Joe* → *Joe's*
  - Semicolons (*;*) **never** go before lists
    - that's what colons (*:*) are for
  - Commas **never** separate a sentence's subject from its predicate
    - even if it's natural to pause when speaking

Proofreading

- Read your report **many times**
  - Ask yourself whether *your reader* will understand (as opposed to whether *you* understand)
- Read your report out loud
  - If you find it hard to *say* the words, they will not be pleasant to read either
  - If you run out of breath, you need to split up your sentences more
- Don’t rely on your editor’s spell-checker only
  - You can try reading your sentences *backwards* when you check for typos

Conclusions

- Leave enough time for your report
  - This is what you are marked on!
- From now on, you'll only need to do *more* writing, so use this opportunity to practise
  - We'll give you individual feedback on your submissions
- Think about and avoid the common mistakes
- Write clearly, concisely, unambiguously

Questions