Programming for Performance

Saska Dönges

AGENDA

1. Goals
2. Introductions and probing questions.
3. Course practicalities.
4. Short break / Pre-assignment panic.
5. High level discussion about course contents / Intro to PfP.
6. Exit questionnaire.

Interrupt to comment or ask questions!

AFTER TODAYS LECTURE YOU SHOULD...

- know how the course is sructured,
- know what is required of you to pass the course
- and have a basic understanding of some of the higher level concepts explored on the course.

Who am I? Who are you?

- Where from (BSc, CS, Math, Physics)?
- What for?
- Course history?
- Relevant supporting knowledge?

THIS COURSE...

...is a replacement for the OLD course “Programming in C”. A self-guided, automatically assessed C Programming course.

...now has a fancyer name and a skew towad performance. (Beyond the addition of “++”.)

...aims to give you all programming practice in C++, and an awareness of some issues affecting performance and ways to improve performance.

DO THIS, DO THAT

There is lots of C++ on the web – by all means learn from it, but NO CHEATING!!!

Engage in difficult study in order to develop your ability to think so that your life becomes easier.

GRADING AND ASSIGNMENTS

Final course grade is based on:

- A series programming assignments,
- B series programming assignments,
- written assignments, and
- lecture / exercise session / case study activity.

To pass you will also need to have done 50% of A series programming tasks and written assignments, as well as either actively attend at least 3 course meetings or pass an oral examination.

GRADING

There are over 160 points available on the course. At least 40 points for each of the different activity types. Initital point limits are as follows:

0-39 points: grade 0
40-59 points: grade 1
60-79 points: grade 2
80-99 points: grade 3
100-119 points: grade 4
120+ points: grade 5

This is the second course implementation with this set of activieties, the initial grading criteria are still not guaranteed to be very good. The given point limits may lower, but will not under any circumstances be higher than this.

“A” SERIES PROGRAMMING TASKS

Intended to be doable for people not particularly familiar with C++ or systems programming.

Main challenges should be:

- working with C++,
- implementing software based on techincal descriptions and
- becoming familiar with concepts presented on the course.

If your implementations perform as required asymptotically, time and space limits should not be much of a concern.

I will be making spot checks to CSES submissions. No cheating!

“B” SERIES PROGRAMMING TASKS

Meant to be challenging.

I will make performance requirements as difficult as I can without making problems unfair.

Some tasks may still be very hard. Don’t get discouraged if you can’c complete some B series tasks.

I don’t expect many (any?) people to solve all of the B series tasks.

You don’t technically need to solve any to get grade 5 on the course.

WRITTEN TASKS

Written tasks are meant to push students in the direction of performance oriented thinking.

All written tasks are submitted to moodle.

For students not present at the exercise session following the assignment deadline, the submission will be manually graded by the TA (or me). (Feedback is not guaranteed)

For students present at the exercise session, self-reported points will be given by default, and TA will be making spot checks to verify that the self reported points and moodle submissions agree.

MEETING ACTIVITY

There will be 3 kinds of meetings during the course:

Lectures. Flipped classroom with pre-assignments.

Case studies. Discussion on specific problems with pre-assignments.

Exercise sessions. Discussing / presenting solutions to written assignments.

For each meeting (except the intro lecture), 4 points will be available for meeting activity.

Unfortunately, there are too many people enrolled for me to be able to give activity points without book-keeping.

A list will be circulated at the start of each meeting, and I will make note of activity on the attendence sheet

QUESTIONS AND COMMENTS?

Do you disagree with how this course is run?

Do you have questions about course practicalities?

MOODLE

Now is the time to switch to the browser.

Hope I remembered to set it up before.

PRE-ASSIGNMENT AND BREAK

Hopefully I remembered to show the pre-assignment on moodle.

If you didn't realize that there was a pre-assignment you will now have until the end of the break to try to get up to speed.

Why PfP

Why PfP

Computers are getting slower [citation needed] $\Rightarrow$

We need to be more efficient.

Also, it's fun.

WHAT IS SYSTEMS PROGRAMMING?

WHAT IS SYSTEMS PROGRAMMING?

According to Wikipedia

"[T]he activity of programming computer system software. The primary distinguishing characteristic of systems programming when compared to application programming is that application programming aims to produce software which provides services to the user directly, whereas systems programming aims to produce software and software platforms which provide services to other software, are performance constrained, or both."

How is systems programming relevant for this course?

How is systems programming relevant for this course?

While not completely accurate, systems programming is the best description for what we do on the course (that I can think of).

How is systems programming relevant for this course?

While not completely accurate, systems programming is the best description for what we do on the course (that I can think of).

Specifically, we don't really care about user interfaces (graphical or otherwise), as long as we can control execution enough to run tests.

If someone were to ever actually use the code produced on this course, it would be as a data structure library or to get inspiration from the implementation. Not to run the software as is.

So what do we actually do?

Beyond learning C++.

So what do we actually do?

Beyond learning C++.

Single thread, performance oriented data structures and supporting algorithms.

WHAT DON'T WE DO

WHAT DON'T WE DO

Competetive programming. “Algoritmit ongelmanratkaisussa” (Algorithmic problem solving), by Antti laaksonen.

Cluster computing. “Tools for high performance computing” at the physics department.

Parallel computation. “Programming parallel computers” by Antti Laaksonen, from Aalto.

WHAT ELSE WE DON'T DO

WHAT ELSE WE DON'T DO

Real time systems. Theory is covered on operating systems and computer architecture courses, and perhaps something practical on the Networks study track.

Process Cohabitation on a single core / thread / system. Theory is somewhat covered on other courses, but nothing practical (AFAIK).

Actual low level programming. Writing device drivers or programming in assembly. Perhaps something on the Networks study track?

WHY C++

WHY C++

The vast majority of systems / low level code has been and is still written in C, C++ or fortran (if you are physics inclined).

The vast majority of all research code is written in C++, Python, fortran, Matlab or JavaScript

Since C++ is a superset(ish) of C, and C++ is more convenient, we use C++ over C.

Since we aren't physicists (at least I'm not), we skip fortran.

WHY NOT ⟨LANGUAGE⟩?

- Python
- Matlab
- JavaScript
- ASM

- Rust
- Java
- C#
- ...

QUESTIONNAIRE FOR COURSE DEVELOPMENT AND RESEARCH

`https://elomake.helsinki.fi/lomakkeet/128673/lomake.html`

It would really help me if you take the time to respond.