Efficient C++

Petr Ročkai

October 8, 2020

Part A: Introduction

Organisation

- online discussion at a specific time
  - we will find a common slot
- collect 12 points to pass the subject
- most points come from assignments
- a few from competitions & peer review

Assignments

- one assignment every 2 weeks, 6 in total
- one assignment = 2 points

Bonuses (per assignment)

- add 1 point if you pass within 14 days
- else add 0.5 points if you pass by 30.1.

Assignments (cont'd)

- details about submission next week
- we will use a small subset of C++
- the code must be valid C++17
- sanity tests run every midnight
- verity tests run every Friday

Competitions

- we will hold 3 online competitions
- do your best in 40–60 minutes on a small problem
- the winner gets 2 points, 2nd & 3rd place get 1.5
- all other working programs get 1 point
- we'll dissect the winning program together

Peer Review

- you can write feedback for classmates
- up to 6 reviews, 0.5 points each
- you must pass the assignment first

Exercises

- there will be a few weekly exercises
- working them out is optional but recommended
- we can discuss them in the online chat

Summary of Points

- 12 points for assignments
- 6 points for early work
- 3–6 points for competitions
- 3 points for peer review

- you need 12 points by 27.2. to pass

Semester Plan (part 1)

		date
1.	computational complexity	8.10.
2.	microbenchmarking & stats	15.10.
3.	the memory hierarchy	22.10.
4.	using `callgrind`	29.10.
5.	tuning for the compiler	21.10.
6.	competition 1	5.11.

Semester Plan (part 2)

		date
7.	understanding the CPU	12.11.
8.	exploiting parallelism	19.11.
9.	competition 2	26.11.
10.	using `perf`	3.12.
11.	competition 3	10.12.

Assignment Schedule

	given	due
1. benchmarking tool	8.10.	23.10.
2. matrix multiplication	22.10.	6.11.
3. sets of integers	5.11.	20.11.
4. substring search	19.11.	4.12.
5. parallel computation	3.12.	18.12.
6. a hash table	31.12.	15.1.

Efficient Code

- computational complexity
- the memory hierarchy
- tuning for the compiler & optimiser
- understanding the CPU

- exploiting parallelism

Understanding Performance

- writing and evaluating benchmarks
- profiling with `callgrind`
- profiling with `perf`
- the law of diminishing returns

- premature optimisation is the root of all evil
- (but when is the right time?)

Tools

- on a POSIX operating system (preferably not in a VM)
- `perf` (Linux-only, sorry)
- `callgrind` (part of the `valgrind` suite)
- `kcachegrind` (for visualisation of `callgrind` logs)
- maybe `gnuplot` for plotting performance data

Compilers

- please stick to C++17 and C11 (or C99)
- the reference compiler will be gcc 9.3.0
- you can use other compilers locally
- but your code has to build with the above

Part 1: Computational Complexity

Complexity and Efficiency

- this class is not about asymptotic behaviour
- you need to understand complexity to write good code
- performance and security implications

- what is your expected input size?
- complexity vs constants vs memory use

Quiz

- what's the worst-case complexity of:
  - a bubble sort? (standard) quick sort?
  - inserting an element into a RB tree?
  - inserting an element into a hash table?
  - inserting an element into a sorted array?
  - appending an element to a dynamic array?
- what are the amortised complexities?
- how about expected (average)?

Hash Tables

- often the most efficient data structure available
- poor theoretical worst-case complexity
  - what if the hash function is really bad?
- needs a fast hash function for efficiency
  - rules out secure (cryptographic) hashes

Worst-Case Complexity Matters

- CVE-2011-4815, 4838, 4885, 2012-0880, …
- apps can become unusable with too many items
- use a better algorithm if you can (or must)
- but: simplicity of code is worth a lot, too
- also take memory complexity and constants into account

Constants Matter

- n ops if each takes 1 second
- $n \log n$ ops if each takes .1 second
- n^2 ops if each takes .01 second

Picking the Right Approach

- where are the crossover points?
- what is my typical input size?
- is it worth picking an approach dynamically?
- what happens in pathological cases?

Exercises

- log into `aisa`
- run `pb173a update`
- then `cd ~/pb173a/01`
- and `cat intro.txt`