

Programmieren in C++

SS 2010

Vorlesung 13, Mittwoch 21. Juli 2010
(Projekt, Performance + Profiling, Ende)

Prof. Dr. Hannah Bast
Lehrstuhl für Algorithmen und Datenstrukturen
Institut für Informatik
Universität Freiburg

Blick über die Vorlesung heute

■ Organisatorisches

- Erfahrungen mit dem 12. Übungsblatt
- Bitte alle Evaluationsbögen abgeben
- Noch dieses und jenes zum Projekt

■ Performance + Profiling

- Bisher haben wir uns gar nicht um Performance gekümmert
- Das gehört auch eher in Algorithmen und Datenstrukturen
- Trotzdem heute schon mal ein Ausblick
 - Compiler optimization flags, insbesondere -O
 - Profiling mit gprof
 - Diverse Tipps und Tricks

Erfahrungen mit dem 12. Übungsblatt

- Zusammenfassung / Auszüge
 - Header Dateien schreiben ist ganz schön schwer, wenn man noch gar nicht weiß, wie man alles implementiert

- Kurze Zusammenfassung (aus bisher 30 Bögen)
 - Den meisten hat es gut oder sehr gut gefallen, insbesondere:
 - Vorlesungsaufzeichnungen, Art der Übungsblätter, Praxisnähe, Motivation, Struktur und Stil der Vorlesung, Atmosphäre, Kommunikation auf Wiki / Forum / überhaupt ...
 - ca. 1/3 waren tendenziell überfordert, einige davon sehr
 - ca. 1/3 waren tendenziell unterfordert, einige davon sehr
 - ca. 1/3 fanden Tempo / Stoffmenge genau richtig
 - Unzuverlässigkeit mancher Tutoren wurde kritisiert
 - An der Stelle eine Bitte:
 - Melden Sie sich für die Vorlesung **Programmieren in C++ im SS 2011** bei mir und machen Sie es besser !

■ Oscars

- Kategorie "Das musste ja kommen"

Was könnte man besser machen: Forum → Wiki

- Kategorie "Moralischer Zeigefinger"

Kommentare wie "ich habe heute nur 3 Stunden geschlafen" gehören nicht in eine Vorlesung

- Kategorie "Frech aber ok"

Dozentin wirkt manchmal etwas verplant bzw. übermüdet

- Noch ein paar Infos dazu
 - Bei Fragen Mail ans Forum oder Termin mit Tutor oder Jens Hoffmann abmachen
 - Wer mit dem Projekt fertig ist, bitte Mail an den Tutor mit Cc an Jens Hoffmann
 - Wir würden uns freuen, wenn möglichst viele ihr Projekt möglichst früh fertig machen
 - Ansonsten, gnadenlose Deadline am **15. September**

■ Infos zur Notenvergabe

- Übungsblätter 1 – 11 + Evaluationsbogen : 120 Punkte
- Übungsblatt 12 + Projekt : 50 Punkte
- Insgesamt : 170 Punkte (mit Bonuspunkten auch mehr)
- Den Schein gibt's ab 100 Punkten, Notenzuordnung:
 - 100 – 107 : 4.0
 - 108 – 114 : 3.7, 115 – 121 : 3.3, 122 – 128 : 3.0
 - 129 – 135 : 2.7, 136 – 142 : 2.3, 143 – 149 : 2.0
 - 150 – 156 : 1.7, 157 – 163 : 1.3, ≥ 164 : 1.0
- Nochmal für alle **INFO** Studierenden:
 - Das zählt **nicht** für Ihre Endnote

- Automatentheorie (Info 3) ist oft langweilig
 - das soll jetzt anders werden
 - <http://www.summercamp-informatik.de>
 - auf Deutsch (größtenteils)
 - kostenlos
 - in Saarbrücken
 - findet statt vom 19. – 24. September 2010
 - Bewerbungsschluss ist schon der **31. Juli** !

■ Fragen dabei

- Läuft mein Programm so schnell wie es könnte?
- Und wenn nein, wie kann ich es schneller machen?

■ Es gibt im Wesentlichen drei Potenziale

- Algorithmische Verbesserung
 - Thema von Algorithmen und Datenstrukturen, z.B. Sortieren von n Zahlen in n^2 Zeit vs. $n \log n$ Zeit
- Compiler effizienteren Code erzeugen lassen
 - Da ist (mindestens) eine Vorlesung für sich
- Algorithm Engineering
 - Wissen darüber welche Konstrukte im Code aus welchen Gründen wie lange dauern

■ Werkzeuge / Tools

- Wissen über Rechnerarchitektur und Compiler
- Zeitmessung
- Ein sogenannter Profiler wie [gprof](#) (GNU profiler)

Wir machen das anhand eines Beispiels

- Vereinigung von sortierten Listen von Zahlen
 - Ein in vielerlei Hinsicht typisches Beispiel
 - Große Datenmengen
 - Werden in einer Schleife verarbeitet
 - Sehr viele Iterationen, von denen jede Einzelne etwas relativ Einfaches macht
- Effekte die eine Rolle spielen
 - Möglichst einfacher (Maschinen)code
 - Speicher(pre)allokation
 - Branch prediction
 - Loop unrolling

Vereinigung sortierter Listen

- So geht der Standard-Algorithmus dafür

■ Moderne Prozessoren

- Versuchen die nächste Anweisung auszuführen noch bevor die aktuelle fertig ausgeführt ist (**pipelining**)
- Das wird besonders bei einem **if** schwierig (branch), weil der Wert der Bedingung vielleicht in der aktuellen Anweisung erst berechnet wird
- Ein guter Compiler versucht das vorherzusagen
- In jedem Fall hilft es aber, die Anzahl der **if** Anweisungen in einer Schleife zu minimieren

■ Schleifenoptimierung

- Statt einer Schleife mit 8 Durchgängen
`for (int i = 0; i < 8; i++) { BODY; }`
- Kann man auch 8 mal hintereinander `BODY` schreiben
`BODY; BODY; BODY; BODY; BODY; BODY; BODY; BODY`
- Code dafür ist schneller, weil die Auswertung der Schleifenbedingung entfällt
- Bei loop unrolling, macht der Compiler einen Mix, etwa
`for (int i = 0; i < 4; i++) { BODY; BODY; }`
(Statt 8 Durchläufe nur 4 Durchläufe, wobei in jedem Durchlauf 2 mal `BODY` ausgeführt wird)

- gprof

- <http://www.cs.utah.edu/dept/old/texinfo/as/gprof.html>

- g++ optimization levels

- http://www.network-theory.co.uk/docs/gccintro/gccintro_49.html

- <http://linuxmanpages.com/man1/g++.1.php>

- oder einfach `man g++`

- Branch Prediction

- http://en.wikipedia.org/wiki/Branch_prediction

- Loop unrolling

- http://en.wikipedia.org/wiki/Loop_unwinding

