Programmieren in C++

SS 2011

Vorlesung 13, Mittwoch 3. August 2011

(Evaluation, Profiling, Aktuelles vom Lehrstuhl)

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

UNI FREIBURG

Blick über die Vorlesung heute

UNI FREIBURG

- Organisatorisches
  - Erfahrungen mit dem 12. Übungsblatt (Projekt .cpp)
  - Zuordnung Punkte → Note
  - Ihre Evaluation dieser Veranstaltung: Zusammenfassung
- Profiling
  - Ist mein Programm so schnell wie es sein könnte?
  - Wenn nicht, wie kann ich es schneller machen?
  - Dazu werfen wir einen Blick auf
    - Profiling (mit gprof)
    - Compileroptimierung (für den g++)
    - Assemblercode
- Am Ende kurz: aktuelle Projekte am Lehrstuhl

Erfahrungen mit dem Ü12 (Projekt .cpp)

UNI FREIBURG

- Zusammenfassung / Auszüge Stand 3.8 14:29
  - Habe den Evaluationsbogen abgegeben
  - Evaluationsbogen wurde ausgefüllt
  - Den Evaluationsbogen habe ich hochgeladen
  - Evaluation: done
  - Habe den komischen Evaluationsbogen ausgefüllt
  - ...

  - Einige sind schon fertig, fast fertig oder ziemlich weit
  - Ging soweit bei den meisten ganz gut vorwärts

Gesamtnote zur Veranstaltung

UNI FREIBURG

- Wird nach folgendem Schema festgesetzt
  - Insgesamt 160 Punkte, davon 100 Punkte für Ü1 – U10, 10 Punkte für den Evaluationsbogen und 50 Punkte für das Projekt
  - Den Schein gibt's ab 80 Punkten, davon ≥ 50 Punkte in Ü1 – Ü10 und ≥ 25 Punkte im Projekt; Notenzuordnung:
    - 080 – 087 : 4.0
    - 088 – 095 : 3.7, 096 – 103 : 3.3, 104 – 111 : 3.0
    - 112 – 119 : 2.7, 120 – 127 : 2.3, 128 – 135 : 2.0
    - 136 – 143 : 1.7, 144 – 151 : 1.3, ≥ 152 : 1.0
  - Für **INFO** Studierende ist das "nur" eine Studienleistung, d.h. die Note erscheint zwar (irgendwo) auf dem Zeugnis, zählt aber **nicht** für die Gesamtnote

Evaluationsergebnisse 1/7

UNI FREIBURG

- Stil der Vorlesung
  - Locker, angenehm, unaufgeregt, gechillt, unterhaltsam, motivierend, verständlich, amüsant, kompetent, ... (viele)
  - Beste Vorlesung bisher (einige)
  - Die Dozentin tut sehr gelangweilt vom Stoff (einer)
  - Die Dozentin legt viel Wert auf stud. Feedback (einige)
  - Live-Programmierung sehr lehr- und hilfreich (viele)

    ... allerdings nicht bei zuviel Code (einer)
  - Einige Vorlesungen etwas hektisch am Ende (einige)
  - Auf den Folien nur das Wesentliche ist gut (einer)
  - Mehr Infos auf den Folien (einer)
  - Folien wirken wie um 2:00 Uhr nachts erstellt (einer)

Evaluationsergebnisse 2/7

UNI FREIBURG

- Inhalt der Vorlesung
  - Guter Praxisbezug (viele)
  - Gute Mischung aus "learning" und "doing" (einer)
  - Linux-spezifisch, was ist mit Windows (einige)
  - Mehr Inhalt und mehr in die Tiefe (einige)
  - Grafische Benutzeroberflächen machen (einer)
  - Threads machen (einer)
  - Valgrind früher vorstellen (einer)
  - Mehr zu Effizienz / Software Engineering sagen (einer)
  - Endlich mal strukturiert Programmieren gelernt (einer)

Evaluationsergebnisse 3/7

UNI FREIBURG

- Online-Angebot
  - Vorlesungsaufzeichnung super und nützlich (sehr viele)
  - Fast immer sehr schnelle Hilfe im Forum (viele)
  - Alle Materialien online (viele)
  - Videomitschnitt ist prima (einige)
  - Videomitschnitt ist überflüssig (einige)
  - Wiki, Daphne, Jenkins, SVN alles sehr gut (viele)
  - Abgabe über SVN nicht ganz ausgereift (einer)
  - Das Vorlesungs-PDF ist zur Lösung der Übungsblätter quasi immer nutzlos, es sollte mehr erklärt werden (einer)

Evaluationsergebnisse 4/7

UNI FREIBURG

- Übungsblätter
  - Schöne und lehrreiche Aufgaben (viele)
  - Aufgaben sind realitätsfern (einer)
  - Schön, dass stark mit Vorlesung abgestimmt (viele)
  - Aber verführt auch zum einfach nur Abschreiben (einer)
  - Aufgaben mal einfach, mal schwierig, das ist gut (einige)
  - Aufgaben mal einfach, mal schwierig, das ist doof (einige)
  - Aufgaben manchmal nicht präzise gestellt (einige)
  - Teilweise viel zu viel für Anfänger (einige)
  - Zuviel für 4 ECTS Punkte (einer)
  - Mehr Stoff & schwierigiere Aufgaben bitte (einige)
  - Früher Aufgaben wo die .h Datei nicht gegeben ist (einer)

Evaluationsergebnisse 5/7

- Tutoren
  - Hilfreicher, freundlicher Tutor (einige)
  - Feedback nicht erst 1 Tag vor Deadline vom nächsten Übungsblatt wäre nützlich gewesen (einige)
  - Zu wenig Feedback vom Tutor (einige)
  - Treffen mit dem Tutor besser organisieren (einer)
  - Was gefällt Ihne am besten an der Veranstaltung?

    Mein Tutor! (einer)

Evaluationsergebnisse 6/7

UNI FREIBURG

- Linter und Tests
  - Linter zu streng (einige)
  - Froh über Linter, weil guter Stil wichtig (einer)
  - Testfälle nerven ... aber wichtig (einer)
  - Weniger Tests (einer)

Evaluationsergebnisse 7/7

UNI FREIBURG

- Sonstige Anregungen
  - Uhrzeit zu spät, 14 – 16 Uhr wäre besser (einige)
  - Mehr ECTS Punkte + Note sollte zählen (einige)
  - Vorkurs besser ankündigen (zwei)
  - Mehr Feedback zu den Übungsblättern wäre schön (einer)
  - Nächstes Mal C lehren und nicht C++ (zwei)
  - Jenkins und Daphne auf https setzen (einer)
  - Git statt SVN verwenden (einer)
  - Anfänglichen Code in der Vorlesung nicht überschreiben, sondern erhalten. z.B. durch Auskommentieren (einer)
  - Projektthemen früher bekannt geben (einer)

Performance / Profiling

UNI FREIBURG

- Fragen dabei
  - Läuft mein Programm so schnell wie es könnte?
  - Und wenn nein, wie kann ich es schneller machen?
- Es gibt im Wesentlich drei Potenziale
  - Algorithmische Verbesserung
    - Thema von Algorithmen und Datenstrukturen, z.B. Sortieren von n Zahlen in n^2 Zeit vs. $n \log n$ Zeit
  - Algorithm Engineering
    - Wissen darüber welche Konstrukte im Code aus welchen Gründen wie lange dauern
  - Compileroptimierung
    - Wissen darüber wofür der Compiler unter welchen Umständen welchen Maschinencode erzeugt

Beispielprogramm

UNI FREIBURG

- Vereinigung von sortierten Listen von Zahlen
  - Ein in vielerlei Hinsicht typisches Beispiel
  - Große Datenmengen
  - Werden in einer Schleife verarbeitet
  - Die einzelnen Iteration machen etwas sehr Einfaches
- Effekte die im Folgenden eine Rolle spielen
  - Kosten für Speicherallokation
  - Kosten für Funktionsaufrufe
  - Kosten für Konditionale (if ...) / Branch Prediction
  - Möglichst einfacher (Maschinen)code

Vereinigung von sortierten Listen

- Hier nochmal kurz der Standard-Algorithmus

Kosten für Speicherallokation

UNI FREIBURG

- Speicherallokation hat Kosten
  - ... und zwar fixe Kosten pro Allokation + Kosten linear in der Größe des allozierten Speichersegmentes
  - Deswegen sollte man vermeiden
    - unnötig viele kleine Speicherallokationen
    - unnötige große Speicherallokationen
  - In unserem Beispiel können wir den Speicher für das Ausgabefeld **vorher** allozieren, weil wir vorher schon wissen wie groß das Feld wird
    - so sparen wir uns die vielen Reallokationen beim dynamischen Vergrößern des Feldes

Kosten für Funktionsaufrufe 1/2

UNI FREIBURG

- Funktionsaufrufe haben Kosten
  - Im Maschinencode muss der lokale Kontext und die aktuelle Adresse im Programmcode gesichert werden
  - Dann müssen die Argumente kopiert werden
  - Dann muss zu der Adresse der Funktion gesprungen werden
  - Am Ende der Funktion muss wieder zurückgesprungen und der lokale Kontext wieder hergestellt werden
  - Bei Funktionen, in denen verhältmismäßig viel passiert sind diese Kosten vernachlässigbar
  - Aber bei Funktionen, die nur sehr einfache Dinge tun, können diese Kosten größer sein als die des eigentlichen Codes der Funktion

Kosten für Funktionsaufrufe 2/2

UNI FREIBURG

- Vermeiden von Funktionsaufrufen
  - Wenn man eine Funktion in der .h Datei implementiert und sie nicht zu groß ist, wird der Compiler den Funktionsaufruf einsparen
  - ... indem er einfach den Code der aufgerufenen Funktion an die Stelle des Aufrufes setzt
  - Insbesondere passiert das bei den ganzen "kleinen" Funktionen aus der STL wie std::vector::size, std::vector::push_back, usw.
  - Allerdings nur mit der Compileroption -O, siehe gleich

Profiling

- Wo verbraucht mein Programm wieviel Zeit?
  - Das erfährt man mit einem sogenannten **profiler**
  - Zum Beispiel gprof, das ist der GNU Profiler
  - Einfach das Programm mit der Option –pg übersetzen

    CXX = g++ -pg ...
  - Dann das Programm normal bis zum Ende laufen lassen, das erzeugt eine Datei gmon.out
  - Die erhält Informationen darüber wie oft sich das Programm in welchem Teil des Codes aufgehalten hat
  - Man schaut sich aber nicht die rohe Datei gmon.out an, sondern ruft gprof mit dem Namen der ausf. Datei auf

    gprof ./MergeListsMain

Branch Prediction

UNI FREIBURG

- 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
  - Es gibt zahlreiche Techniken mit dem Ziel, das herauszufinden, schon zur Compilezeit oder zur Laufzeit
    - siehe Referenzen am Ende
  - In jedem Fall hilft es aber, die Anzahl der if Anweisungen in einer Schleife zu minimieren oder gar zu eliminieren

Maschinencode 1/2

UNI FREIBURG

- Den erzeugten Maschinecode
  - ... kan man sich mit g++ einfach wie folgt anschauen

    g++ -S MergeTwoLists.cpp
  - Das erzeugt dann eine Datei

    MergeTwoLists.s
  - Das ist quasi der Maschinencode der .o Datei in menschenlesbarer Form (Assemblersprache)
  - Die (triviale) Übersetzung in Maschinencode geht dann mit

    g++ -c MergeTwoLists.s

Maschinencode 2/2

UNI FREIBURG

- Optimierter Code
  - Ohne Optimierung erzeugt g++ Maschinencode der eins zu eins dem C/C++ Code entspricht
    - siehe einfaches Codebeispiel
  - Mit Optimierung wird versucht Code zu erzeugen, der schneller läuft, dabei gibt es verschiedene Stufen

    g++ -O1 ...

    g++ -O2 ...

    g++ -O3 ...
  - Als Faustregel gilt, je höher die Optimierungsstufe, desto größer der Code und desto schneller läuft er ... -O1 bringt aber schon das meiste, für Details siehe Referenzen

Aktuelle Forschung

UNI FREIBURG

- Forschung an unserem Lehrstuhl
  - Wir machen Algorithmen und Datenstrukturen
    - 1/3 Theorie (neue Algorithmen, Laufzeitanalyse, etc.)
    - 1/3 Algorithm Engineering (gute Implementierungen)
    - 1/3 Software Engineering (gute Software)
  - Aktuelle Projekte
    - Routenplanung, insbesondere die auf Google Maps
    - Suchmaschinen, insbesondere CompleteSearch & Broccoli
  - Aktuelle Arbeiten dazu
    - http://ad.informatik.uni-freiburg.de/papers

Literatur / Links

- 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