Algorithmen und Datenstrukturen (für ESE) WS 2010 / 2011

Vorlesung 9, Montag, 20. Dezember 2010
(Cache-Effizienz, IO-Effizienz)

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

UNI FREIBURG

Blick über die Vorlesung heute

UNI FREIBURG

- Organisatorisches
  - Diese Woche weder Übungsgruppe noch Fragetermin
    - alle etwaigen Fragen bitte ans Forum oder Ihren Tutor
  - Ihre Erfahrungen mit dem Ü8 (Balancierte Suchbäume)
- Cache- bzw. IO-Effizienz
  - Bisher haben wir zur Abschätzung der Laufzeit immer die Anzahl der Operationen gezählt
  - Heute sehen wir, dass das nicht immer ein gutes Maß ist

Ihre Erfahrungen mit dem 8. Ü-Blatt

UNI FREIBURG

- Zusammenfassung von Ihrem Feedback
  - Manche fanden das Blatt sehr einfach
  - Aufgabe 2 und 3 fanden fast alle einfach
  - Manche fanden Aufgabe 1 verwirrend
  - Manche unsicher ob Aufgabe richtig verstanden
  - Programmieren macht mehr Spaß, aber auch mehr Arbeit
  - Die Kekse passen leider nicht ins SVN
  - Bis heute morgen hatten gerade mal 3 Leute abgegeben

Cache-Effizienz

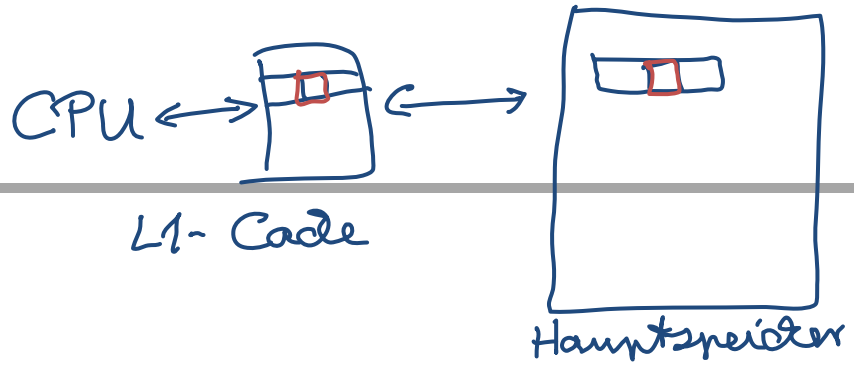
UNI FREIBURG

- Hintergrund
  - Bisher haben wir immer die Anzahl der Operationen gezählt
  - In der Annahme, dass das ein gutes Maß für die Laufzeit eines Algorithmus / Programms ist
  - Heute sehen wir Beispiele, wo das kein gutes Maß ist
- Beispiel 

  - Wir addieren die Elemente eines Feldes der Größe n auf
    - einmal in der natürlichen Reihenfolge 1, ..., n 1+2+3+4+5
    - einmal in zufälliger Reihenfolge 2+1+4+5+3
  - Die Anzahl der Operationen ist in beiden Fällen identisch
  - Aber die Laufzeiten unterscheiden sich sehr, warum?

Faktor bis zu 40

CPU Cache



- Prinzip / Aufbau
  - Zugriff auf ein Byte im Hauptspeicher kostet ca. 100ns
  - Zugriff auf ein Byte im (L1-)Cache kostet ca. 1ns
  - Bei Zugriff auf ein oder mehrere Bytes im Hauptspeicher holt man gerade einen ganzen Block von ~ 100 Bytes in den Cache
  - Solange dieser Block im Cache ist, braucht man für Bytes aus diesem Block nicht auf den Hauptspeicher zuzugreifen
  - Der Cache hat Platz für viele solcher Blöcke (die cache lines)
    - ein typischer L1-Cache ist ~ 100 Kilobytes groß
  - Ist der Cache voll, wird einer der Blöcke entfernt
    - z.B. der least recently used (LRU) Block
    - das soll aber heute nicht das Thema sein

Disk Cache

UNI FREIBURG

- Prinzip / Aufbau
  - Den Lesekopf einer Festplatte an eine bestimmte Stelle zu bewegen kostet ~ 5ms (seek time)
  - Ist man an einer Stelle kann man mit ~ 100 MB / Sekunde Daten lesen (transfer rate)
  - Deshalb geht das Betriebssystem wie folgt vor
    - Wird ein Byte von der Platte gelesen, wird gleich ein ganzer Block eingelesen (z.B. 64 Kilobytes auf einmal)
    - Solange dieser Block im Speicher ist, braucht man für Bytes aus diesem Block nicht mehr auf die Platte zuzugreifen und spart sich die seek time
    - Ist der Speicher voll, muss man sich wieder überlegen, welche Blöcke man drin behält

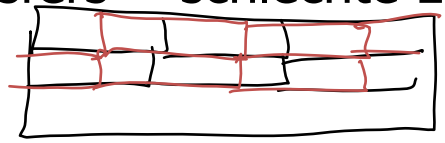
Anzahl Blocktransfers — Definition 1/3

UNI FREIBURG

- Terminologie
  - Wir haben einen langsamen und einen schnellen Speicher
  - Der langsame Speicher ist in Blöcke der Größe B unterteilt
  - Der schnelle Speicher ist M groß (Platz für M/B Blöcke)
  - Stehen die Daten nicht im schnellen Speicher wird der entsprechende Block in den schnellen Speicher geladen
  - Das Programm kann sich aussuchen, welche Blöcke im schnellen Speicher gehalten werden
  - Wir zählen nur die **Anzahl der Blocktransfers (BTs)**
    - In der Praxis hat man auch noch die Kosten für das Verwalten der Blöcke im schnellen Speicher, insbesondere welcher Block entfernt wird wenn der Speicher voll ist

Anzahl Blocktransfers — Definition 2/3

- Für B Operationen hat man also
  - im besten Fall nur 1 Blocktransfer "gute Lokalität"
  - im schlechtesten Fall B Blocktransfers "schlechte Lokalität"



- Variationen
  - Die folgenden Variationen machen nur einen konstanten Faktor in der Anzahl der Blocktransfers aus:
  - Die genaue Aufteilung des langsamen Speichers in Blöcke
  - Ob die Speichereinheit 1 Byte oder 4 Bytes oder 8 Bytes ist
- Man beachte:
  - Das Ganze wird erst interessant, wenn die Eingabe größer als M ist, sonst kann man einfach erstmal die gesamte Eingabe in den schnellen Speicher laden

Anzahl Blocktransfers — Definition 3/3

- Typische Werte (für einen Server)
  - CPU Cache: B = 128 Bytes, M = 128 KB (L1), 8 MB (L2)
  - Disk Cache: B = 64 Kilobytes, M = 16 GB
    - Die meisten Betriebssysteme benutzen alles was vom Hauptspeicher gerade nicht genutzt wird als Disk Cache
  - Sinnvollerweise wählt man B proportional zum Quotienten der Zugriffsgeschwindigkeiten der beiden Medien (Hauptspeicher / Cache oder Festplatte / Hauptspeicher)
- Noch etwas Terminologie
  - Die Blocktransfers beim CPU Cache nennt man cache misses
  - Die Blocktransfers beim Disk Cache nennt man oft IOs
    - IO oder I/O = Input/Output Operation
  - Man spricht auch von Cache-Effizienz und IO-Effizienz

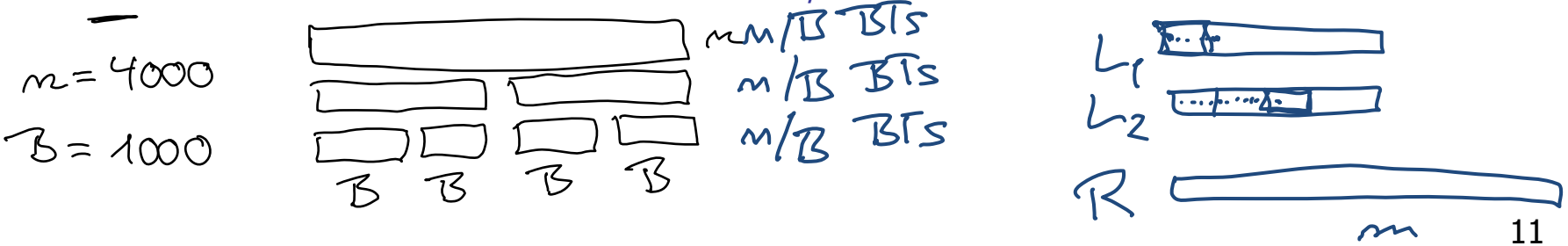
Anzahl Blocktransfers — Beispiel 1/2

UNI FREIBURG

- Nehmen wir unser ArraySum Programm
  - Wenn wir über die Elemente in der Reihenfolge 1, 2, 3, ... iterieren brauchen wir ceil(n/B) = Θ(n/B) Blocktransfers
  - Wenn wir über die Elemente in einer zufälligen Reihenfolge iterieren brauchen wir im schlechtesten Fall n = Θ(n) Blocktransfers
  - Das ist ein Faktor von B Unterschied und das ist der Hauptgrund für den beobachteten Laufzeitunterschied
    - man beachte, dass wir auch bei der zufälligen Reihenfolge pro Element immerhin auf 4 benachbarte Bytes (ein int) auf einmal zugegriffen haben
    - außerdem wird, wenn nicht n >> M, das nächste Element manchmal zufällig schon im schnellen Speicher stehen

Anzahl Blocktransfers — Beispiel 2/2

- Schauen wir uns MergeSort an
  - Nehmen wir an n/B ist eine Zweierpotenz
  - Dann sind wir nach $k = \log_2(n/B)$ Rekursionen bei n/B Stücken der Größe B
  - Von denen können wir jedes mit 1 Blocktransfer sortieren
  - Zwei Folgen der Gesamtlänge m kann man mit $\Theta(m/B)$ Blocktransfers mischen
  - Macht $\Theta(n/B)$ Blocktransfers pro Rekursionsstufe
  - Macht insgesamt $\Theta(n/B \cdot k) = \Theta(n/B \cdot \log_2(n/B))$ BTs
  - Es geht auch mit $\Theta(n/B \cdot \log_{M/B}(n/B))$ Blocktransfers



Literatur / Links

- Cache-Effizienz / IO-Effizienz
  - In Mehlhorn/Sanders:

    2 Introduction 2.2.1 External Memory
  - In Cormen/Leiserson/Rivest

    Nothing! [zero matches for the word "cache"]
  - In Wikipedia

    http://en.wikipedia.org/wiki/Cache

    http://de.wikipedia.org/wiki/Cache

    (da wird das Prinzip eines Caches beschrieben, es gibt keinen separaten Artikel zur Cache-Effizienz bei Algorithmen, nicht mal auf Tschechisch)