Algorithmen und Datenstrukturen (ESE)
Entwurf, Analyse und Umsetzung von Algorithmen (IEMS)
WS 2012 / 2013

Vorlesung 8, Dienstag 11. Dezember 2012
(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
  - Ihre Erfahrungen mit dem Ü7 (dynamische Felder)
- Cache-Effizienz 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
  - Insbesondere: Wie funktioniert ein Cache, Blockoperationen
  - Übungsblatt 8: Sortieren vs. Hashing
    - Vergleich #Operationen vs. #Block-Operationen
    - Vergleich der tatsächlichen Laufzeiten

Erfahrungen mit dem Ü7 (dyn. Felder)

UNI FREIBURG

- Zusammenfassung / Auszüge Stand 11. Dezember 15:56
  - Zur Erinnerung: VL wurde von Björn Buchhold gehalten
  - Vorlesung + Aufgabe waren gut !
  - Aufwand für die meisten 4 Stunden oder weniger
  - Vorprogrammieren wie immer toll
  - Beweise haben sich in die Länge gezogen
  - Video nur mit Bild und Ton braucht nicht 500 MB groß sein

    korrekt, aber das ist nur die Roh-Version direkt nach der Vorlesung

Cache-Effizienz

UNI FREIBURG

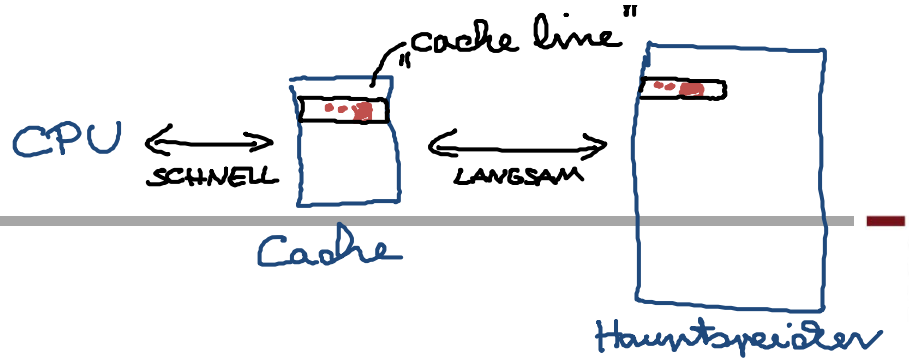
- Hintergrund
  - Bisher haben wir immer die Anzahl der Operationen gezählt
  - In der Annahme, dass das ein gutes Maß für die Laufzeit eines Algorithmus / Programms ist
  - Heute sehen wir Beispiele, wo das kein gutes Maß ist
- Beispiel
  - Wir addieren die Elemente eines Feldes der Größe n auf

    ... in der natürlichen Reihenfolge: 1 + 2 + 3 + 4 + 5

    ... in einer zufälligen Reihenfolge: 2 + 5 + 1 + 3 + 4
  - Die Anzahl der Operationen ist für beide **identisch**
  - Aber die Laufzeiten unterscheiden sich sehr, warum?

bis zu Faktor 10 warum?

CPU Cache



UNI FREIBURG

- 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 mehr 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

d.h. 10ms pro Byte

- 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. 128 KB 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

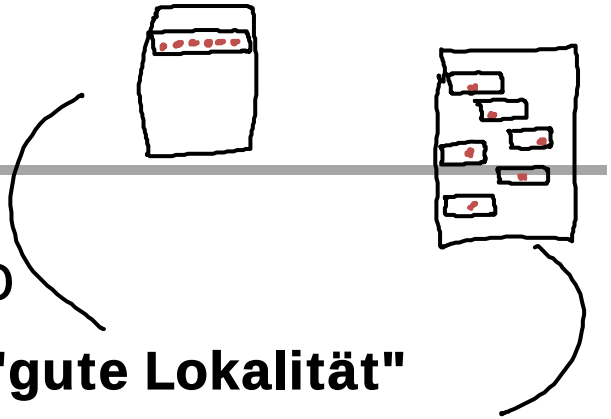
Block-Operationen 1/7

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 Block-Operationen**

    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 ... das ignorieren wir hier

Block-Operationen 2/7



- Für B Operationen hat man also
  - im besten Fall nur 1 Block-Op. **"gute Lokalität"**
  - im schlechtesten Fall B Block-Op. **"schlechte Lokalität"**
- Die folgenden Variationen ...
  - ... machen nur einen (kleinen) konstanten Faktor in der Anzahl der Block-Operationen 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

Block-Operationen 3/7

- Typische Werte (für einen Server)
  - CPU Cache: B = 128 Bytes, M = 6 x 32 KB (L1), 6 x 256 KB (L2)
  - Disk Cache: B = 64 Kilobytes, M = 64 GB
    - Die meisten Betriebssysteme benutzen alles was vom Hauptspeicher gerade nicht genutzt wird als Disk Cache
  - Sinnvollerweise wählt man B so, dass die transfer time für einen Block ein Bruchteil der seek time ist
- Noch etwas Terminologie
  - Die Block-Operationen beim CPU Cache nennt man cache misses
  - Die Block-Operationen beim Disk Cache nennt man oft IOs
    - IO oder I/O = Input/Output
  - Man spricht auch von Cache-Effizienz und IO-Effizienz

Block-Operationen 4/7

n >> M

UNI FREIBURG

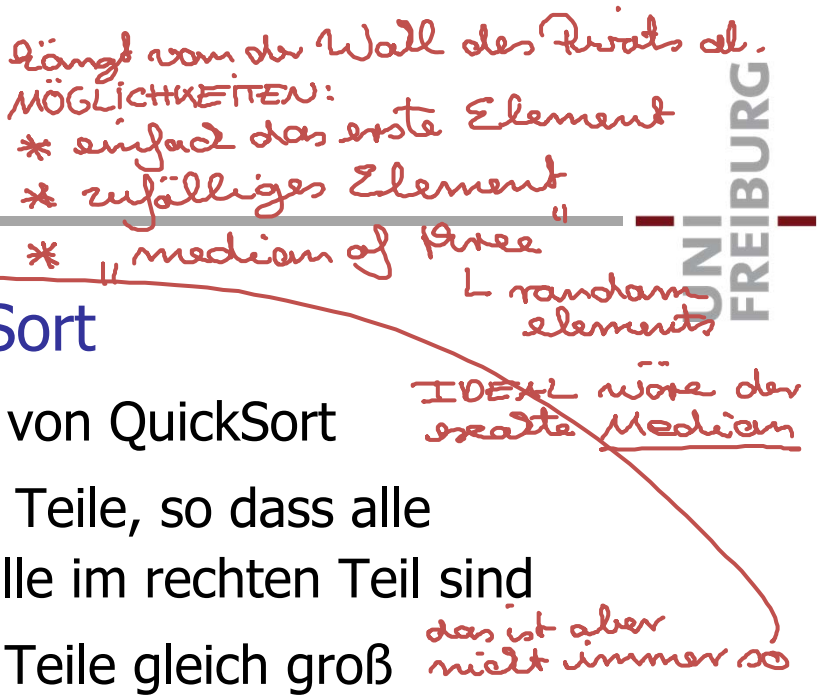
- Beispiel 1: unser ArraySumMain Programm
  - Wenn wir über die Elemente in der Reihenfolge 1, 2, 3, ... iterieren, ist die Anzahl Block-Operationen: $\lceil n/B \rceil$
  - Wenn wir über die Elemente in einer zufälligen Reihenfolge iterieren, ist die Anzahl Block-Op. im schlechtesten Fall: n
  - Das ist ein Faktor von B Unterschied und das ist der Hauptgrund für den beobachteten Laufzeitunterschied

    Man beachte, dass auch bei der zufälligen Reihenfolge pro Element auf 4 benachbarte Bytes (ein int) auf einmal zugegriffen wurde

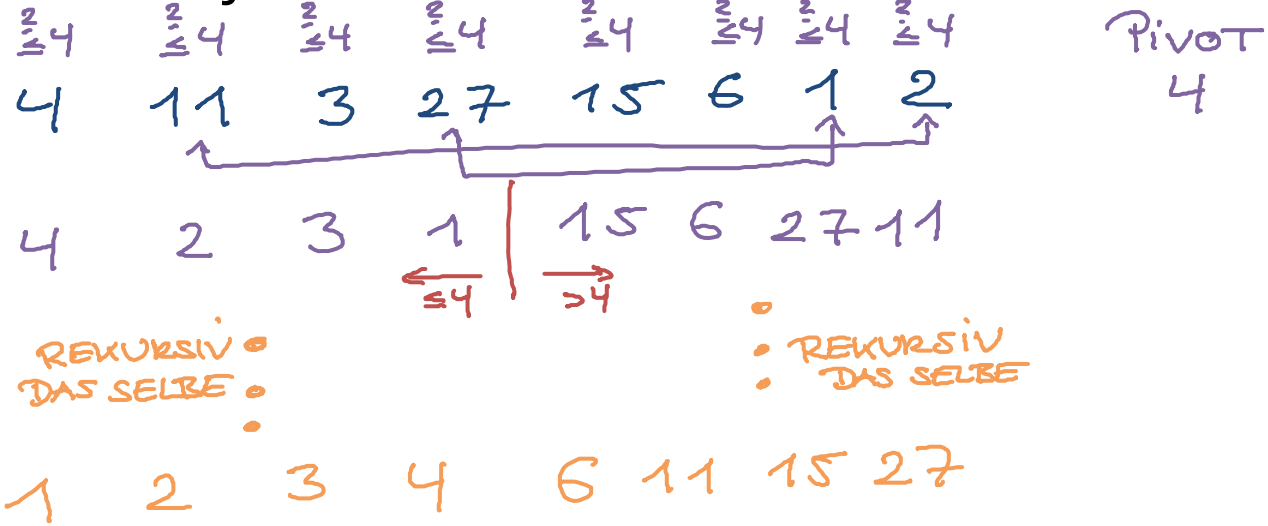
    Außerdem wird, wenn nicht n >> M, das nächste Element manchmal zufällig schon im schnellen Speicher stehen

    Deswegen ist der Unterschied in der Praxis deutlich < B

Block-Operationen 5/7



- Beispiel 2: Sortieren mit QuickSort
  - Vorweg kurz die Funktionsweise von QuickSort
    - Teile das Eingabefeld in zwei Teile, so dass alle Elemente im linken Teil $\leq$ alle im rechten Teil sind
    - Idealerweise sind die beiden Teile gleich groß
    - Sortiere dann jeden der beiden Teile rekursiv



Block-Operationen 6/7

- Beispiel 2: Sortieren mit QuickSort
  - Analyse der Anzahl gewöhnlicher Operationen

Sei $T(n)$ = Laufzeit für Eingabegröße n

Annahme: Felder werden immer perfekt in der Mitte geteilt + n ist eine Zweierpotenz

$$T(n) \leq \underbrace{A \cdot n}_{\text{zum Teilen in zwei Teile}} + \underbrace{2 \cdot T(n/2)}_{\text{für die rek. Aufrufe}}$$

$$\leq A \cdot n + 2 \cdot [A \cdot n/2 + 2 \cdot T(n/4)]$$

$$= 2 \cdot A \cdot n + 4 \cdot T(n/4)$$

$$\leq 3 \cdot A \cdot n + 8 \cdot T(n/8)$$

$$\leq \ldots$$

$$\leq k \cdot A \cdot n + \underbrace{2^k}_{=n} \cdot \underbrace{T(1)}_{\leq A} \qquad k = \log_2 n$$

$$= O(n \cdot \log n)$$

Block-Operationen 7/7

UNI FREIBURG

- Beispiel 2: Sortieren mit QuickSort
  - Analyse der Anzahl Block-Operationen $IO(n)$

Annahmen wie auf der Folie davor

$$IO(n) \leq A \cdot n/B + 2 \cdot IO(n/2)$$
$$\leq A \cdot n/B + 2 \cdot [A \cdot n/2B + 2 \cdot IO(n/4)]$$
$$= 2 \cdot A \cdot n/B + 4 \cdot IO(n/4)$$
$$\leq 3 \cdot A \cdot n/B + 8 \cdot IO(n/8)$$
$$\vdots$$
$$\leq \underbrace{k}_{\log_2 \frac{n}{B}} \cdot A \cdot n/B + \underbrace{2^k}_{n/B} \cdot \underbrace{IO(n/2^k)}_{IO(B)=1}$$

$k = \log_2 \frac{n}{B} = \log_2 n - \log_2 B$

$$= O(n/B \cdot \log_2 \tfrac{n}{B})$$

es gilt sogar $O(\frac{n}{B} \cdot \log_{M/B} \frac{n}{B})$

Literatur / Links

UNI FREIBURG

- 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 aber keinen separaten Artikel zur Cache-Effizienz bei Algorithmen)