Algorithmen und Datenstrukturen (für ESE) WS 2011 / 2012

Vorlesung 8, Dienstag, 20. Dezember 2011
(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- 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 Ü7 (DynArr)

UNI FREIBURG

- Zusammenfassung von Ihrem Feedback Stand 20.12 15:40
  - Implemententierungsteil war recht einfach
  - Auf dem ÜB stand nicht, dass man remove implementieren soll
  - Beweis war auch nicht wirklich schwer, hat aber gedauert
  - Zeitprobleme ... weihnachts- oder virusbedingt
  - Fehler in der Vorlesung hat es schwerer gemacht / Zeit gekostet
    - Auf der letzten Folie musste man das s_{il} separat addieren
    - Außerdem wurde im Programm die Größe verdoppelt und in der Analyse hieß es mal 3/2
    - Jetzt schon zum dritten Mal Beweis in der Vorlesung inkorrekt
  - Vorlesung hat diesmal nicht so gut auf das ÜB vorbereitet
  - Gab aber auch viele, für die das kein Problem war

Ihre Erfahrungen mit dem Ü7 (DynArr)

UNI FREIBURG

- Feedback Fortsetzung ...
  - Implementierung für Aufgabe 1 weniger effizient als möglich
    - aber dadurch Beweis für Aufgabe 2 einfacher ... komisch
  - Vergleich statische vs. dynamische Felder besser schon in erster VL bringen ... man braucht es ja von Anfang an
  - Probleme mit Math.floor
  - Durch Programmieren lernt man mehr als durch Beweise
  - Beweisen macht mehr Spaß als Programmieren

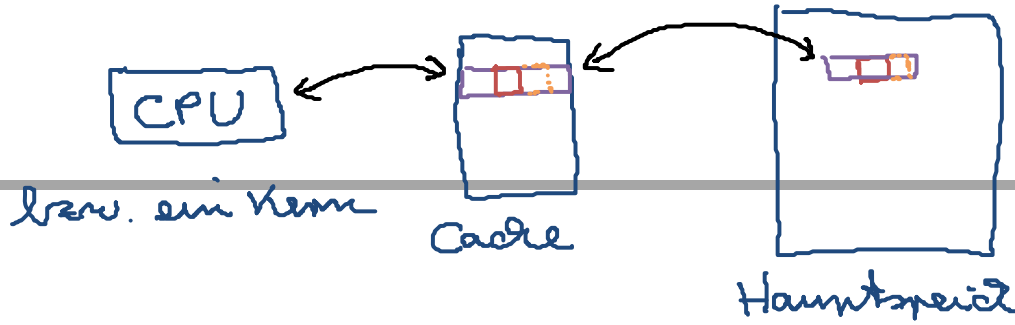
Cache-Effizienz

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: $1 + 2 + 3 + 4 + 5$
    - einmal in zufälliger Reihenfolge: $2 + 3 + 1 + 5 + 4$
  - Die Anzahl der Operationen ist im ersten Fall: $n - 1$
  - Die Anzahl der Operationen ist im zweiten Fall: $n - 1$
  - Aber die Laufzeiten unterscheiden sich sehr, warum?

Faktor 20 und mehr ...

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 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

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. 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

Anzahl Block-Op. — 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 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

Anzahl Block-Op. — Definition 2/3

- 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

Anzahl Block-Op. — Definition 3/3

- 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

Anzahl Block-Op. — Beispiel 1/2

Arraygröße n; Blockgröße B

- Nehmen wir unser ArraySum 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 BOs im schlechtesten Fall: n
  - 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

z.B. $M = n$

Anzahl Block-Op. — Beispiel 2/2

- Schauen wir uns MergeSort an
  - Nehmen wir an n/B ist eine Zweierpotenz
  - Dann sind wir nach $R = \log_2(n/B)$ Rekursionen bei n/B Stücken der Größe B
  - Von denen können wir jedes mit 1 Block-Op. sortieren
  - Zwei Folgen der Gesamtlänge m kann man mit $\Theta(m/B)$ Block-Operationen mischen

  - Macht $\Theta(n/B)$ Block-Operationen pro Rekursionsstufe
  - Macht insgesamt $\Theta(n/B \cdot R) = \Theta(n/B \cdot \log_2(n/B))$ BOs
  - Es geht auch mit $\Theta(n/B \cdot \log_{M/B}(n/B))$ Blockoperationen
    - In jedem Rekursionsschritt in M/B Stücke teilen
    - Beweisskizze auf der nächsten Folie

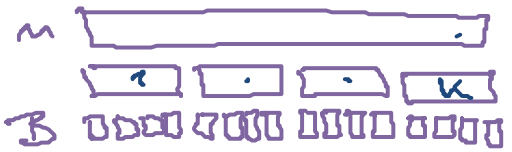
Anzahl Block-Op. — Beispiel 2/2

"cache-oblivious" = man muss B nicht kennen

UNI FREIBURG

Beweis MergeSort $\Theta(n/B \cdot \log_{M/B}(n/B))$

Idee: in jedem Schritt nicht nur 2 Teilprobleme aufteilen, sondern in $K = M/B$

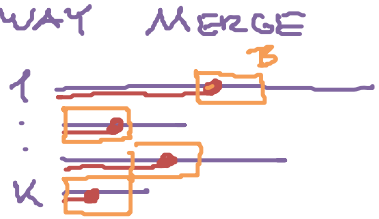


Analyse:

$IO(n) = K \cdot IO(n/K) + n/B$

#Block-Op.

K-way merge für $K \leq M/B$



$= K \cdot [K \cdot IO(n/K^2) + n/K/B] + n/B$

$= K^2 \cdot IO(n/K^2) + 2 \cdot n/B$

$\ldots = K^R \cdot IO(n/K^R) + R \cdot n/B$

$R = \log_K \frac{N}{B}$

$= n/B \cdot \underbrace{IO(B)}_{=1} + \log_K(\tfrac{N}{B}) \cdot n/B$

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 aber keinen separaten Artikel zur Cache-Effizienz bei Algorithmen)