universität freiburg

Algorithmen und Datenstrukturen SS 2025

Vorlesung 7: Cache-Effizienz, Blockoperationen

Dienstag, 3. Juni 2025

Prof. Dr. Hannah Bast
Professur für Algorithmen und Datenstrukturen
Institut für Informatik
Albert-Ludwigs-Universität Freiburg

Blick über die Vorlesung heute

- Organisatorisches
  - Erfahrungen Ü6 — Kilometerzähler, Potenzialfunktion
  - Zeitmanagement — Alles nicht so einfach
  - Vorgucker Ü7 — Sie haben **drei** Wochen Zeit
- Inhalt
  - Lokalität Speicherzugriffe — Motivation + Hintergründe
  - Anzahl Blockoperationen — Definition + viele Beispiele
  - ArraySum und MergeSort — Analyse Anzahl Blockoperationen

Nächte Vorlesung erst wieder am **24. Juni** … erholen Sie sich gut, aber fangen Sie nicht zu spät an mit dem 7. Übungsblatt!

Erfahrungen mit dem Ü6 1/4

- Auszüge aus Ihrem Feedback [halbwegs repräsentativ]

  "Schöne Mischung aus Programmier-Praxis und Theorie"

  "Das Blatt hat mir überraschend gut gefallen"

  "Interessantes Blatt und Vorlesung sehr hilfreich dafür"

  "VL anspruchsvoller, aber interessant, didaktisch super stark"

  "Folien sind perfekt zum Wiederholen des Stoffes für das ÜB"

  "Bei den Aufzeichnungen auf YouTube wird Werbung geschaltet"

  "Zweite Aufgabe war die schwierigste bisher"

  "Aufgabe 2 unvollständig, da Beweis in VL nicht verstanden"

  "In jeder VL wird gesagt, dass Studenten zu faul oder dumm sind"

  "Feedback immer nur positiv, das entspricht nicht der Realität"

Erfahrungen mit dem Ü6 2/4

- Diesmal mehr negatives Feedback als sonst, Gründe?
  - Vorweg: wir lesen immer das gesamte Feedback und nehmen es sehr ernst … aber auch nicht alles für bare Münze

    Manchmal (nicht immer) ist der Grund auch einfach Frust
  - Die Aufgabe 2 vom Ü6 war die bisher schwierigste

    Man konnte den Beweis aus der Vorlesung nicht einfach nachmachen + es war mehr Transferleistung gefordert

    Aber keine Sorge, schwieriger wird es jetzt nicht mehr
  - Die fünf Gruppen von Studierenden (V2F5) sind eine Realität

    Damit war und ist aber **keine** Wertung verbunden
  - Bitte bleiben Sie fair … und nutzen Sie die zwei Woche Pause

Erfahrungen mit dem Ü6 3/4

- Wie geht es aktuell im Semester / mit dem Studium?

  "Mir geht's prima, alles gechillt, viel Zeit für Hobbies"

  "Sehr gut, habe mich gut eingefunden, weiß mich zu organisieren"

  "Es geht mir richtig gut, auch wenn es sehr fordernd ist"

  "Größerer Wandel, arbeite bewusster und auf Verständnis"

  "C++ Vorlesung wenig hilfreich, AlgoDat ist super spannend"

  "Wird anspruchsvoller, viel zu tun, aber ich komme noch mit"

  "Insgesamt gut, bin gespannt wie es nach den Prüfungen sein wird"

  "Eigentlich gut, aber das Wetter bringt einen zur Zeit etwas runter"

  "Noch komme ich gut mit, hatte mir aber weniger Mathe erhofft"

  "Wird schwieriger aber noch aushaltbar, es lebe die Pfingstpause"

Erfahrungen mit dem Ü6 4/4

- Wie geht es aktuell im Semester / mit dem Studium?
  - "Frage mich manchmal, wie viel ich eigentlich gelernt habe"
  - "Ich finde, dass einen das Studium sehr isoliert"
  - "Dieses Semester deutlich besser, weil ich nette Menschen kennengelernt habe, mit denen sich das Leid leichter teilen lässt"
  - "Mein Semester ist sehr voll und ich freue mich wenn's vorbei ist"
  - "Ich werde an vielen Stellen gefaltet, aber ansonsten ganz gut"
  - "In Ordnung, dann eine Woche krank, seitdem hänge ich hinterher"
  - "Mir fehlt der Spaß, weil Lernen für mich persönlich nichts ist"
  - "Mache mir sehr viel Druck und habe große Angst zu versagen"
  - "Bin ziemlich gestresst und habe Angst nicht mitzukommen"

Zeitmanagement

Zum Ausgleich, eine ganze Vorlesung weniger und zwar am 17. Juni

- Manchmal fehlt uns am Ende etwas die Zeit

  Warum dann nicht einfach nächste Woche weitermachen?

  Ein Thema pro Vorlesung hat sich sehr sehr bewährt

  Filme und so einen Quatsch am Anfang weglassen?

  Fünf Minuten Sinnlosigkeit (oder Fehler) pro VL müssen sein

  Bei den letzten Folien geht es manchmal schneller

  1. Versuche ich zu vermeiden, ist aber schwieriger als man denkt, zumal die Vorlesung bewusst sehr interaktiv gestaltet ist
  2. Früher war die Vorlesung perfekt getimed, aber es blieb kaum Zeit zum Nachdenken und es wurden kaum Fehler bemerkt
  3. Das Wichtigste kommt **immer im ersten und zweiten Drittel**

Vorgucker auf das Ü7

- Einmal "Rechnen" und einmal "Laufzeitanalyse"

  A1 Denkaufgabe zum Zählen von Blockoperation

  Aufgabe zum Verständnis der Definition von Blockoperationen

  Wenn man die Aufgabe gemacht hat, hat man wirklich verstanden, wie man Blockoperationen richtig zählt

  Sowas in der Art kommt auch oft in der Klausur dran

  A2 Anzahl Blockoperationen von MergeSort

  Konkretes Anwendungsbeispiel

  Wir bereiten das heute in der Vorlesung vor, indem wir MergeSort etwas umschreiben (damit es möglichst wenige Blockoperationen braucht)

Lokalität Speicherzugriffe 1/5

- Einfach(st)es Beispiel
  - Wir addieren die n Elemente eines Feldes auf
    … in der natürlichen Reihenfolge: 1 + 2 + 3 + 4 + 5
    … in einer zufälligen Reihenfolge: 2 + 5 + 3 + 1 + 4
  - Das Ergebnis ist in beiden Fällen **identisch**
  - Die Anzahl der Operationen ist ebenfalls **identisch**
  - Laufzeit bei natürlicher Reihenfolge: 95ms
  - Laufzeit bei zufälliger Reihenfolge: 1187ms
  - Wir fragen uns:

mit PyPy

n = 100M

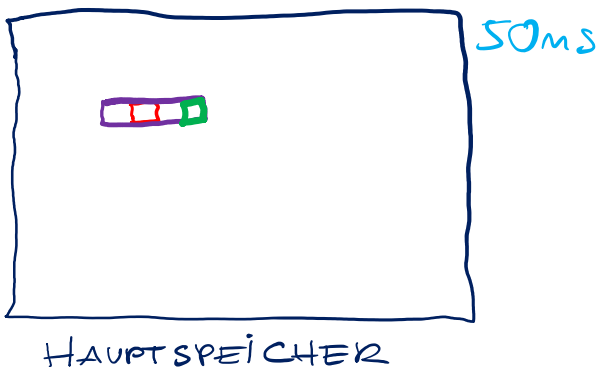
n = 100M

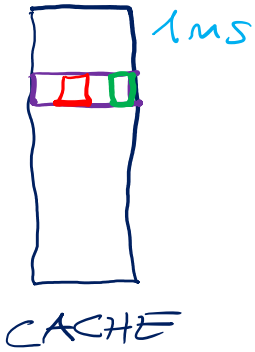
WARUM ?

Lokalität Speicherzugriffe 2/5

- CPU Cache, Prinzip
  - Zugriff auf ein Byte im Hauptspeicher: **~ 10 – 50 ns**
  - Zugriff auf ein Byte im Level-1 Cache: **~ 1 ns**

    Wir sehen gleich, was es mit dem "Level-1" auf sich hat
  - Bei Zugriff auf ein oder mehrere Bytes im Hauptspeicher holt man gleich einen ganzen Block ("cache line") in den Cache
  - Solange dieser Block im Cache ist, braucht man für Bytes aus diesem Block nicht mehr auf den Hauptspeicher zuzugreifen





Lokalität Speicherzugriffe 3/5

- Disk Cache, Prinzip
  - Latenz: **~ 5ms** (HDD), **~ 0.1ms** / **0.02ms** (SATA / NVMe SSD)

    Bei HDD: Lesekopf muss an die Stelle bewegt werden
  - Leserate:~ **100 MB/s** (HDD), **~ 600 MB/s** - **6 GB/s** (SSD)

    Bei HDD: Kopf bleibt stehen, Platte dreht sich schnell weiter
  - Deshalb geht das Betriebssystem in der Regel wie folgt vor

    Wird ein Byte von der Platte gelesen, wird gleich ein ganzer Block gelesen (der Standard ist: **4096 Bytes = 4 KiB**)

    Solange dieser Block im Speicher ist, braucht man für Bytes aus diesem Block nicht mehr auf die Platte zugreifen

    Also dasselbe Prinzip wie beim CPU Cache !

Lokalität Speicherzugriffe 4/5

- Speicherhierarchie
  - Auf realen Maschinen gibt es eine ganze **Hierarchie von Speichern**, jede mit einem anderen "Trade-Off" zwischen Größe, Geschwindigkeit und Preis pro Byte
  - Zum Beispiel auf unserem Vorlesungsrechner ("tura"):

    Level-1 Cache: 33 **K**B … sehr schnell, aber sehr teuer / Byte

    Level-2 Cache: 524 **K**B … etwas langsamer + etwas günstiger

    Level-3 Cache: 67 **M**B … noch langsamer + noch günstiger

    RAM: 128 **G**B … noch langsamer + noch günstiger

    Platte: 33 **T**B … siehe vorherige Folie

    Unter Linux: getconf -a | grep CACHE und free –h --si und df -h

Lokalität Speicherzugriffe 5/5

- Wenn der Cache voll ist
  - … muss einer der Blöcke entfernt werden, dafür gibt es zahlreiche Strategien ("eviction strategy"), zum Beispiel:

    LRU (Least Recently Used) = der Block, für den es am längsten her ist, dass darauf zugegriffen wurde

    LFU (Least Frequently Used) = der Block, auf den am wenigsten zugegriffen wurde, seit er im Cache ist
  - Im Folgenden spielt die "eviction strategy" oft keine Rolle

    Falls es doch eine Rolle spielt, sagen wir explizit, welche wir annehmen (i.d. Regel ist es dann LRU, die auch Standard ist)

Blockoperationen 1/8

- Abstraktion der bisherigen Beobachtungen
  - Es gibt einen **langsamen** und einen **schnellen** Speicher
  - Beide Speicher sind in Blöcke der Größe B unterteilt
  - Der schnelle Speicher ist M groß = Platz für M/B Blöcke

    Im Folgenden ist M immer ein Vielfaches von B
  - Stehen die Daten nicht im schnellen Speicher, wird der entsprechende Block in den schnellen Speicher geladen (und ein anderer Block herausgeschmissen, siehe Folie 13)
  - Wir zählen nur die **Anzahl der Blockoperationen**

    Also **+1** für jedes Mal, wenn ein Block in den schnellen Speicher geladen wird (alle anderen Operationen ignorieren wir, egal wie viele es sind)

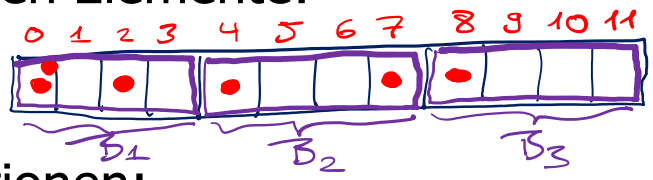
Blockoperationen 2/8

- Beispiel 1
  - Langsamer und schneller Speicher sind in Blöcke der Größe $B = 4$ unterteilt und der schnelle Speicher hat Größe $\mathbf{M = 4}$

    Feld A mit 12 Elementen, fängt an einer Blockgrenze an

    Hintereinander Zugriff auf die folgenden Elemente:

    A[2] A[0] A[8] A[4] A[7] A[0]



  - Wir zählen die Anzahl der Blockoperationen:

	Block im schnellen Speicher	Blockoperation
A[2]	B_1	+1
A[0]	B_1	nix
A[8]	B_3	+1
A[4]	B_2	+1
A[7]	B_2	nix
A[0]	B_1	+1

Insgesamt 4 Blockoperationen

Blockoperationen 3/8

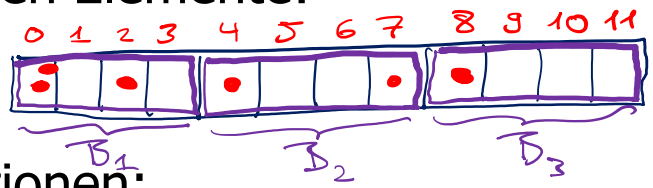
- Beispiel 2 (wie Beispiel 1, aber mit M = 8, LRU)

  – Langsamer und schneller Speicher sind in Blöcke der Größe B = 4 unterteilt und der schnelle Speicher hat Größe **M = 8**

  Feld A mit 12 Elementen, fängt an einer Blockgrenze an

  Hintereinander Zugriff auf die folgenden Elemente:

  A[2] A[0] A[8] A[4] A[7] A[0]



  – Wir zählen die Anzahl der Blockoperationen:

Reihenfolge irrelevant

A[2]	B_1, ×	+1
A[0]	B_1, ×	nix
A[8]	B_1, B_3	+1
A[4]	B_2, B_3	+1
A[7]	B_2, B_3	nix
A[0]	B_2, B_1	+1

Jetzt Platz für zwei Blöcke

Auch 4 Blockoperationen

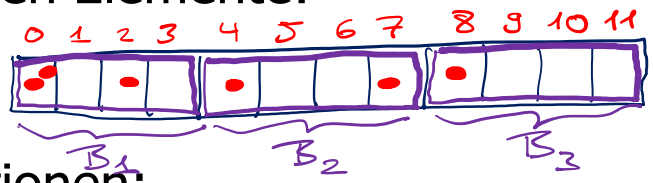
Blockoperationen 4/8

- **Beispiel 3** (wie Beispiel 1 und 2, aber mit M = 12)
  - Langsamer und schneller Speicher sind in Blöcke der Größe B = 4 unterteilt und der schnelle Speicher hat Größe **M = 12**

    Feld A mit 12 Elementen, fängt an einer Blockgrenze an

    Hintereinander Zugriff auf die folgenden Elemente:

    A[2] A[0] A[8] A[4] A[7] A[0]



  - Wir zählen die Anzahl der Blockoperationen:

A[2] — B_1, x, x — +1
A[0] — B_1, x, x — nix
A[8] — B_1, B_3, x — +1
A[4] — B_1, B_3, B_2 — +1
A[7] — B_1, B_3, B_2 — nix
A[0] — B_1, B_3, B_2 — nix

Num 3 Blockoperationen

Blockoperationen 5/8

- Zählen der Anzahl der Blockoperationen
  - Was man dabei vernachlässigt:

    Sämtliche Berechnung auf einem Block im schnellen Speicher

    Kosten für das Verwalten der Blöcke im schnellen Speicher
  - Was nur einen konstanten Faktor Unterschied macht

    Ob eine Operation 1, 2, 4 oder 8 Bytes liest

    Wo genau die Blockgrenzen liegen

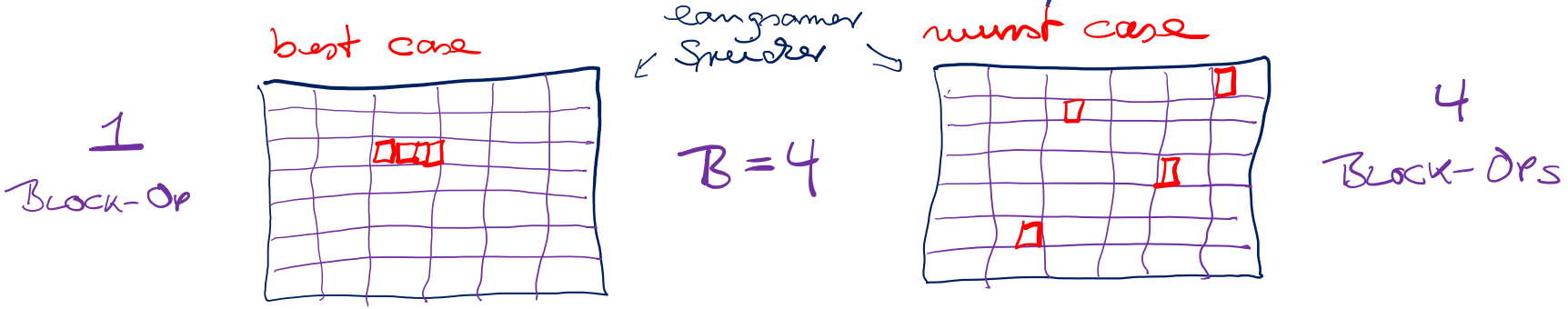
Blockoperationen 6/8

- Gute vs. Schlechte Lokalität
  - Für B Operationen hat man also:

    Im "best case" nur 1 Blockoperation **gute Lokalität**

    Im "worst case" B Blockoperationen **schlechte Lokalität**
  - In der Regel ist $n \gg M$, das heißt, die Eingabe ist viel größer als der schnelle Speicher

    Wenn die Eingabe komplett in den schnellen Speicher passt, kann sie der Algorithmus einmal mit n/B Blockoperationen dorthin laden und dann so viel rechnen, wie er will



Blockoperationen 7/8

- Werte für B und M auf unserem Vorlesungsrechner:
  - CPU L1-Cache: B = 64 Bytes, M = 32 KB — 512 Blöcke
    CPU L2-Cache: B = 64 Bytes, M = 524 KB — 16384 Blöcke
    CPU L3-Cache: B = 64 Bytes, M = 34 MB — 315392 Blöcke
  - Disk Cache: B = 4096 Bytes, M = 128 GB — 32 Mill. Blöcke

    Die meisten Betriebssysteme benutzen alles, was vom Hauptspeicher gerade nicht genutzt wird, als Disk Cache
  - In der Praxis wählt man das B so, dass die Zeit für das Übertragen eines Blocks ein Bruchteil der Zeit ist, die man für einen Zugriff auf ein einzelnes Elements braucht

    Wenn ein Zugriff schon teuer ist, kann man auch noch einen Bruchteil dieser Zeit darauf verwenden, mehr Elemente in den schnelleren Speicher zu holen

Blockoperationen 8/8

- Terminologie
  - Blockoperationen nennt man beim

    CPU Cache: in der Regel **cache misses**

    Weil sie dann nötig werden, wenn ein Stück vom (langsamen) Hauptspeicher nicht im (schnellen) Cache ist

    Disk Cache: oft einfach **IOs**

    IO oder I/O = Input/Output … eher historische Bezeichnung für Datentransfer von der oder auf die Platte
  - Wenn man die Anzahl Blockoperationen eines Algorithmus analysiert, spricht man deswegen oft von

    Cache-Effizienz oder **IO-Effizienz**

ArraySum und MergeSort 1/4

- IO-Effizienz von **Array Sum** 1/2
  - Sei B wie gehabt die Blockgröße
  - Wenn wir über die n Elemente in der Reihenfolge 1, 2, 3, ... iterieren, dann ist die Anzahl Blockoperationen

    $\lceil n/B \rceil$

    
  - Wenn wir über die n Elemente in einer zufälligen Reihenfolge iterieren, dann ist die Anzahl Blockoperationen

    im besten Fall: $\lceil n/B \rceil$

    im schlechtesten Fall: n

    $n \gg M$

    z.B.
    $B = 4$
    $M = 40$
    $n = 10{,}000{,}000$

ArraySum und MergeSort 2/4

- IO-Effizienz von **Array Sum** 2/2
  - Bei unserem Beispielprogramm am Anfang der Vorlesung war der Laufzeitunterschied kleiner als die Blockgröße des Level-1 Cache ($B = 64$)
  - Mögliche Gründe dafür:
    1. Wir haben nicht einzelne Bytes gelesen, sondern einzelne Elemente und ein Element besteht aus mehreren Bytes, die hintereinander im Speicher stehen
    2. Manchmal hat man Glück und das nächste Element in der zufälligen Reihenfolge steht in einem Block, der noch im Cache ist (wird aber kaum passieren, wenn $n \gg M$)
    3. Python (und seine diversen Overheads)

ArraySum und MergeSort 3/4

- IO-Effizienz von **MergeSort** … das ist A2 vom Ü7
  - Wir schauen uns nochmal an, wie das MergeSort aus Vorlesung 1 das Eingabefeld sortiert

    Wir modifizieren dazu das Programm aus der Musterlösung vom Ü1, so dass man sieht, welche Teilfelder in welcher Reihenfolge sortiert werden
  - Wir zählen die Anzahl Blockoperationen für $n = 16$ und $B = 4$

Runde 1 4
Runde 2 4
Runde 3 4
Runde 4 4

$\log_2 n$ Runden

× FAKTOR 2 weil Eingabe und Ausgabe separat

ArraySum und MergeSort 4/4

- IO-Effizienz von **MergeSort** … das ist A2 vom Ü7
  - Wir modifizieren jetzt das Programm so, dass erst der Bereich $[0 : B - 1]$ ganz sortiert wird, dann $[B : 2B - 1]$ usw.
  - Wenn auf diese Weise jeder Block der Größe B in sich sortiert ist, machen wir mit dem normalen MergeSort weiter

    Wichtig zum Verständnis: die Mischvorgänge sind identisch zu vorher, nur die **Reihenfolge** ist etwas anders
  - Wie ist jetzt die Anzahl Blockoperationen für $n = 16$ und $B = 4$

Runde 1+2 4
Runde 3 4
Runde 4 4

Literatur / Links

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

    2 Introduction 2.2.1 External Memory
  - In Wikipedia

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

    https://en.wikipedia.org/wiki/Cache-oblivious_algorithm#Idealized_cache_model