Algorithm Engineering

Alexander Kröller, Abteilung Algorithmik, IBR

#7

Terminchaos

Nächste Vorlesungen:

27. 5.	Vertretung durch Prof. Fekete
3. 6.	Exkursionswoche
10. 6.	Vertretung durch N.N.
17. 6.	back to normal...

‘

Experiment

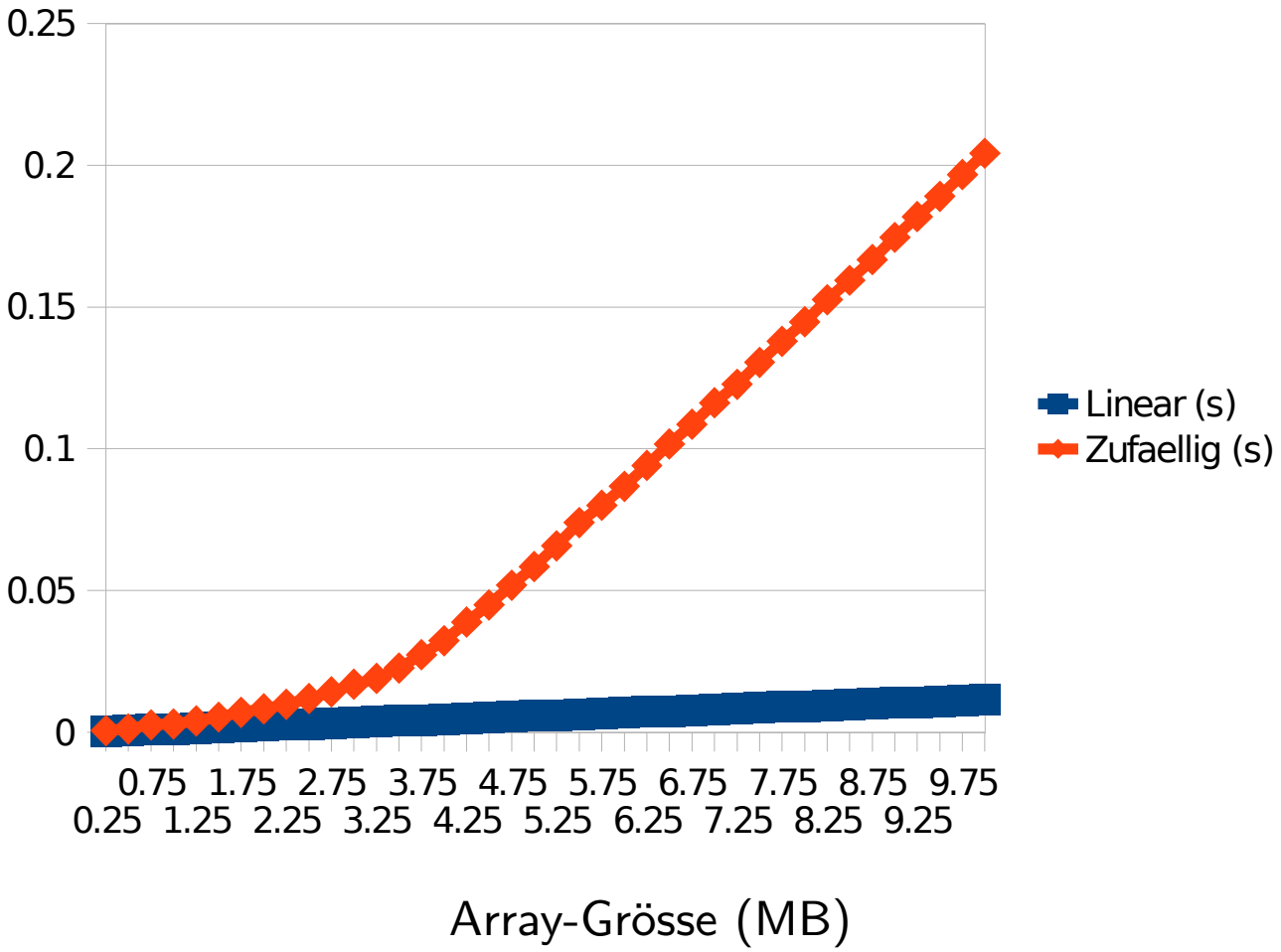
Durchlaufe zwei gleichgrosse Arrays:

- Sortierte Zahlen $1, \ldots, n$
- Zufällige Permutation von $0, \ldots, n-1$ (ein Zyklus)

```
void test( int* arr, int sz )
{
  int* pt=arr;
  for( int cnt=0; cnt<sz; ++cnt )
    {
      pt = arr + *pt;
    }
}
```

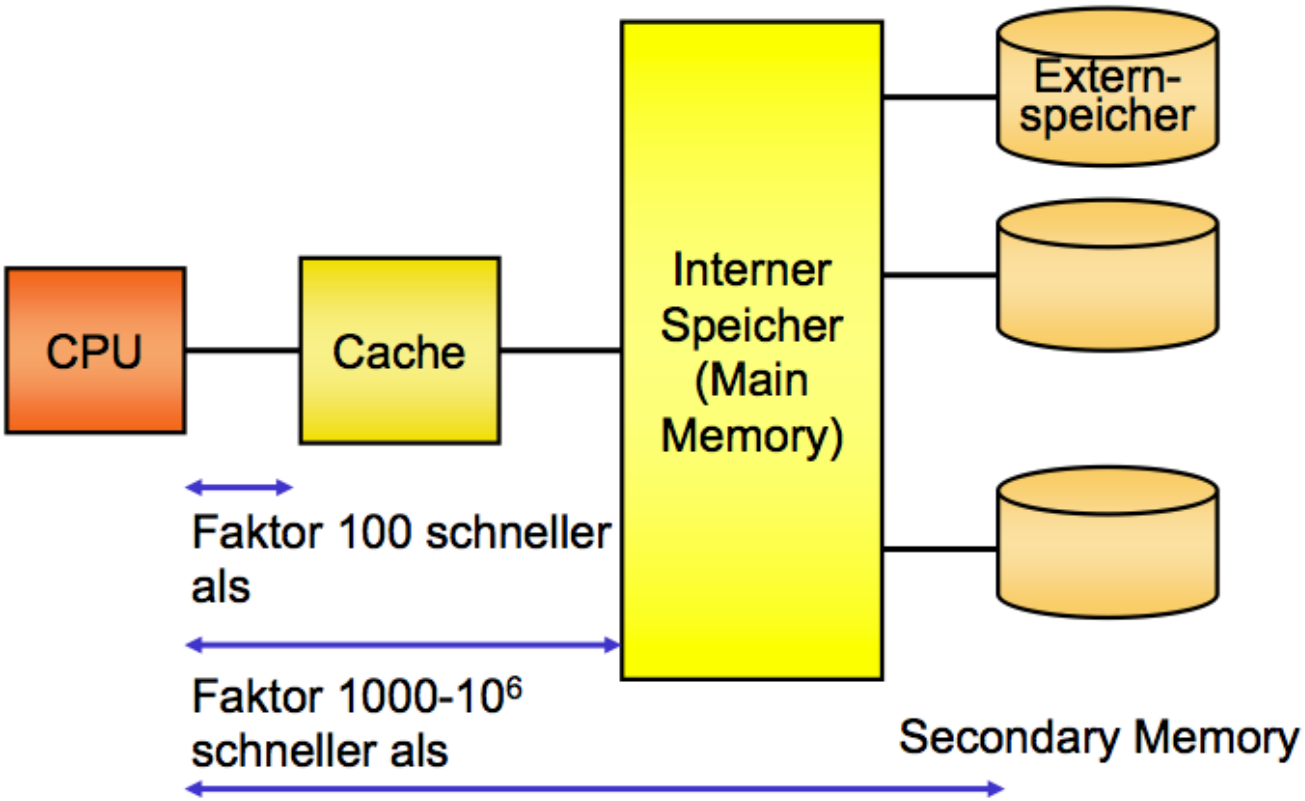
Im RAM-Modell: `test()` hat auf beiden Arrays dieselbe Laufzeit...

Laufzeiten


0.25
0.2
0.15
0.1
0.05
0
0.25 0.75 1.25 1.75 2.25 2.75 3.25 3.75 4.25 4.75 5.25 5.75 6.25 6.75 7.25 7.75 8.25 8.75 9.25 9.75
Linear (s)
Zufaellig (s)
Array-Grösse (MB)

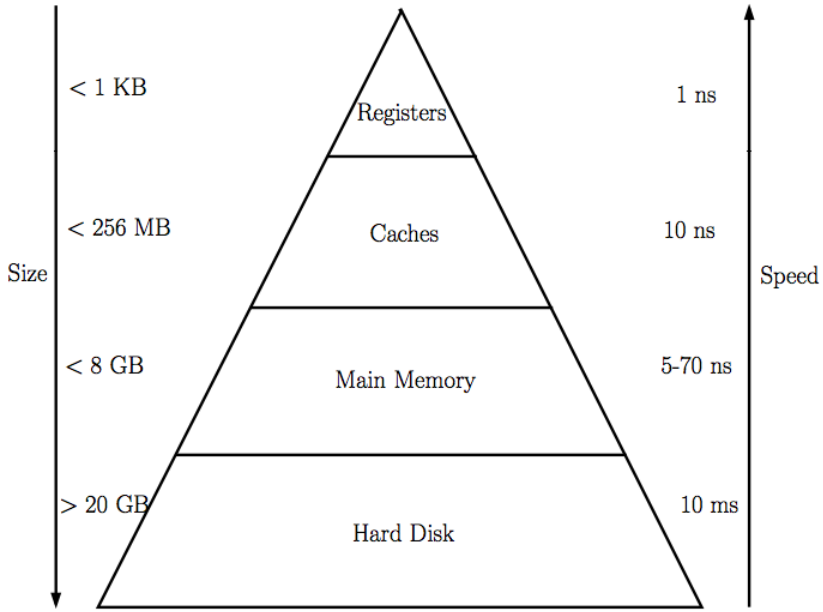
Speicherhierarchien

Cache



[Mutzel]

Speicherhierarchie



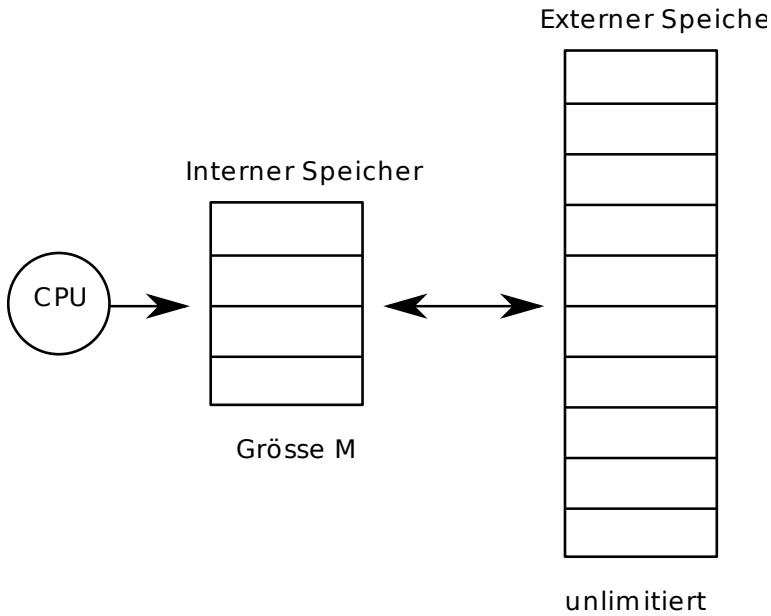
[Müller-Hannemann, Schirra]

Cache-Modell

Interner und externer Speicher haben gemeinsame Blockgrösse B. Der interne Speicher hat Grösse M. (Keine richtige Einheit: Manchmal Wort, manchmal Byte, oft Grösse der verwendeten Datenelemente)

CPU greift auf Daten zu, die nicht im Internen liegen: „Cache miss“. Eine interne Page (welche?) muss gegen eine externe (welche?) ausgetauscht werden. Zählt als „1 I/O“.

Daten müssen Lokalität haben, um I/Os zu vermeiden.



Ideal Cache

Ein „Ideal Cache“ ist ein theoretisches Modell:

- Zwei Ebenen.
- Fully associative: Jede Page des externen Speichers kann in jeder Page des internen abgelegt werden.
- Optimale Strategie: Wenn eine Page aus dem internen gelöscht werden muss, ist es diejenige, die am spätesten in der Zukunft wieder gebraucht wird.

Es gibt viele praktische Pagingalgorithmen, siehe VL „Online-Algorithmen“. Oft verwendet: Least Recently Used (LRU): Die Page, deren letzter Zugriff am ältesten ist, wird gelöscht.

Typen von Algorithmen

Gesucht sind Algorithmen, die eine vorhandene Speicherhierarchie ausnutzen und bei gleicher CPU-Laufzeit weniger I/O verursachen als „naive“ Algorithmen.

Externspeicher-Algorithmus Kann entweder frei kopieren, oder verwendet einen assoziativen Cache und bestimmt das Paging selbst. Kennt M und B.
Anwendung eher bei Problemen, bei denen die Daten nicht in den Hauptspeicher passen. (Also Ebenen Hauptspeicher und Festplatte)

Cache-Aware Kennt M und B und den Paging-Algorithmus.

Cache-Oblivious Kennt weder M, noch B. (Für die Analyse wird ein ideal Cache angenommen)

Rechtfertigung für Cache-Oblivious-Modell: Idealer Cache

Vergleiche Ideal Cache mit LRU:

Definiere $Q_{\text{ideal}}(n, M, B)$: Anzahl I/Os, die gegebener Algorithmus im Ideal Cache verursacht; n ist die Inputgrösse. Analog definiere $Q_{\text{LRU}}(n, M, B)$.

Theorem

$Q_{LRU}(n, M, B) \leqslant 2Q_{ideal}(n, M/2, B)$.

Definiere einen Algorithmus als „regulär“, wenn $Q_{\text{ideal}}(n, M, B) = O(Q_{\text{ideal}}(n, 2M, B))$ gilt.

Theorem

Für einen regulären Algorithmus gilt $Q_{LRU}(n, M, B) \leqslant O(Q_{ideal}(n, M, B))$.

Rechtfertigung für Cache-Oblivious-Modell: Nur zwei Level

Vergleiche 2-Level Ideal Cache mit k-Level LRU (mit inclusion property!)

Theorem

Der i-te Level enthält zu jedem Zeitpunkt dieselben Pages, die er enthielte, wenn er der einzige Level wäre.

Theorem

Ein I/O-optimaler cache-oblivious Algorithmus verursacht in einem k-Level LRU in jedem Level eine (asymptotisch) optimale Zahl von I/Os.

Externspeicher: Stacks

Strategie Lege Stack in sequenziellen Blöcken ab, halte den obersten im Speicher.

`push` Wenn Block voll, starte neuen: 1 I/O.

`pop` Wenn Block leer, hole vorherigen: 1 I/O.

Amortisiert: $1/B$ I/O-Kosten für `push` und `pop`. (Optimal: Mehr als B Elemente lassen sich nicht mit einem I/O transferieren)

Externspeicher: Queues

Strategie Lege Queue in sequenziellen Blöcken ab. Halte ersten und letzten im Speicher.

`push` Wenn letzter Block voll, starte neuen: 1 I/O.

`pop` Wenn erster Block leer, hole nächsten: 1 I/O. (Oder greife auf letzten zu, wenn nur einer belegt)

Amortisiert: $1/B$ I/O-Kosten für `push` und `pop`. (Optimal: Mehr als B Elemente lassen sich nicht mit einem I/O transferieren)

Externspeicher: Listen

Verwende Pages für Teilsequenzen der Liste. Innerhalb einer Page beliebige Ordnung.

Invariante: Zwei aufeinanderfolgende Blöcke sind zusammen mit mindestens $\frac{2}{3}B$ belegt.

`search` Durchsuche die Liste linear.

Worst-Case-Kosten (Listenlänge N):

Interner Algorithmus:	N
Optimaler Algorithmus:	N/B
Externspeicheralgorithmus:	$3N/B$

Listen cont'd

insert

1. Finde den Vorgänger (N/B I/Os)
2. Wenn Platz in dessen Block: füge ein. (0 I/Os)
   Wenn Platz in benachbartem Block: kopiere ein Element um, füge ein. (1 I/O)
   Sonst: Splitte Block in zwei, füge ein. (1 I/O)

Worst-Case-Kosten:

Interner Algorithmus:	$N + 2$
Externspeicheralgorithmus:	$N/B + 1$

Worst-Case-Kosten (wenn Pointer zu Vorgänger gegeben):

Interner Algorithmus:	3
Externspeicheralgorithmus:	3

Listen cont"d

delete

1. Finde den Vorgänger und das Element (N/B I/Os)
2. Lösche das Element (0 I/Os)
3. Wenn Invariante verletzt: Verschmelze zwei Blöcke (1 I/O)

Worst-Case-Kosten:

Interner Algorithmus:	$N+1$
Externspeicheralgorithmus:	$N/B+1$

Worst-Case-Kosten (wenn Pointer gegeben):

Interner Algorithmus:	3
Externspeicheralgorithmus:	3