

Cache-Optimale Algorithmen

VO Algorithm Engineering

Professor Dr. Petra Mutzel

Lehrstuhl für Algorithm Engineering, LS11

15. VO

13.12.2005

Literatur für diese VO

Frigo, Leiserson, Prokop, Ramachandran:
Cache-Oblivious Algorithms, 40th Symposium on
Foundations of Computer Science (FOCS), 285-
298, 1999

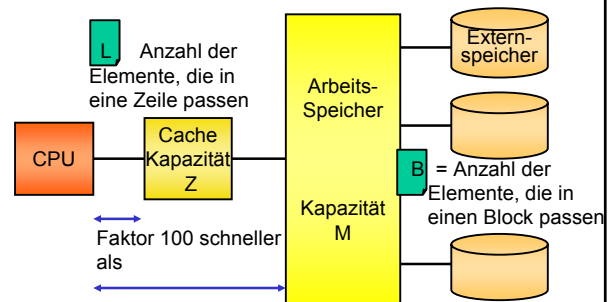
2

Überblick

- Einführung
- Matrixmultiplikation
 - Extern
 - Cache-oblivious

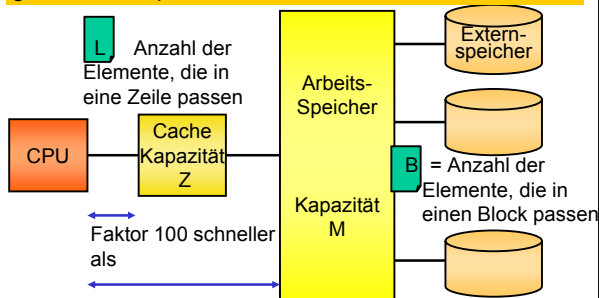
3

Hierarchisches Speichermodell moderner Computer



Auch zur Cache-Optimierung könnte EM-Speichermodell verwendet werden

Problem: Speicherhierarchie, alle mit anderer Blockgröße und Kapazität



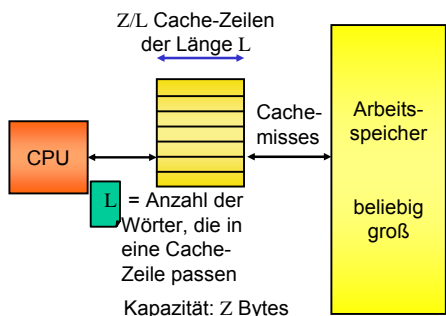
Auch zur Cache-Optimierung könnte EM-Speichermodell verwendet werden

Lösung: "Cache-oblivious" Speichermodell

- Blockgröße B und Kapazität M sind unbekannt
- Eine Algorithmen-Analyse muss für alle Blockgrößen und Kapazitäten gelten – also auch für alle Speicherhierarchie-Ebenen
- Dies erhöht die Portabilität von Programmen: eben nicht für festes B und M entwickelt
- Dies garantiert optimale Speicherzugriffe auf jeder Speicherhierarchie-Ebene.

6

Das "ideal-cache"-Modell von Frigo et al. 1999



2-Ebenen Speicherhierarchie

Annahme: $Z = \Omega(L^2)$: "Cache ist hoch"

7

Das "ideal-cache"-Modell von Frigo et al. 1999

- CPU kann nur Wörter bearbeiten, die sich im Cache befinden
- Wenn sich ein referenziertes Wort im Cache befindet: Cache-Hit
- Sonst: Cache miss: Zeile muss erst in den Cache gebracht werden. **Praxis: least recently used**
- Falls der Cache voll ist, muß eine Cache-Zeile vorher entfernt werden. Der ideale Cache entfernt diejenige off-line optimale Zeile, die zuletzt (zeitlich) wieder benötigt wird.

"Cache-oblivious" Analyse

- Cache-Komplexität $Q(n, Z, L)$: die Anzahl der Cache-Misses, abhängig von Z und L (n Eingabegröße der Instanz).
- Die Anzahl der ausgeführten CPU-Operationen im RAM Modell $T(n)$
- Ein Algorithmus heißt "Cache-Aware" (Cache-bewußt), wenn er Parameter enthält, mit denen die Cache-Komplexität für gegebenes Z und L optimiert werden kann.
- Andernfalls heißt der Algorithmus "Cache-oblivious" (Cache-ignorierend).
- Ein optimaler "Cache-oblivious" Algorithmus für 2 Ebenen, ist auch für mehrere Ebenen optimal.

Cache-Aware Algorithmus

Algorithmus BLOCK_MULT(A,B,C,n)
Für $i=1$ bis n/s
Für $j=1$ bis n/s
Für $k=1$ bis n/s
MULT(A_{ik}, B_{kj}, C_{ij}, s);

MULT(A,B,C,s) ist die Standard-Prozedur, die $C=C+AB$ auf $s \times s$ Untermatrizen in Zeit $O(s^3)$ berechnet.

- Annahme: s ist Teiler von n
- s sei der größte Wert, so dass die drei Untermatrizen zusammen in den Cache passen.
- Eine $s \times s$ Untermatrix benötigt $((s+s^2)/L)$ Cache Zeilen.
- Der Cache besitzt Z/L Cache Zeilen.
- Cache hat Größe Z: $3s^2 \leq Z$; $s = \Theta(\sqrt{Z})$.
- Jeder Aufruf von MULT() benötigt höchstens $Z/L = \Theta(s^2/L)$ Cache-Misses, um die drei Untermatrizen in den Cache zu bringen.
- Insgesamt: $\Theta(1 + n^2/L + (n/\sqrt{Z})^3(Z/L)) = \Theta(1 + n^2/L + n^3/L\sqrt{Z})$

Cache-Oblivious Algorithmus

Idee: Divide & Conquer Prinzip

A: $m \times n$ Matrix, B: $n \times p$ Matrix

- Fall 1: $m \geq \max\{n, p\}$: splüte A horizontal in A_1 und A_2 mit jeweils $m/2$ Zeilen; es folgen 2 Aufrufe der Form $A_1 B$ und $A_2 B$. Denn es gilt:
$$\begin{pmatrix} A_1 \\ A_2 \end{pmatrix} B = \begin{pmatrix} A_1 B \\ A_2 B \end{pmatrix}$$
- Fall 2: $n \geq \max\{m, p\}$: splüte A vertikal in A_1 und A_2 mit jeweils $n/2$ Spalten und B horizontal in B_1 und B_2 mit jeweils $n/2$ Zeilen. Denn es gilt

$$(A_1, A_2) \begin{pmatrix} B_1 \\ B_2 \end{pmatrix} = (A_1 B + A_2 B)$$

Cache-Oblivious Algorithmus

Idee: Divide & Conquer Prinzip

A: $m \times n$ Matrix, B: $n \times p$ Matrix

- Fall 3: $p \geq \max\{m, n\}$: splüte B vertikal in B_1 und B_2 mit jeweils $p/2$ Spalten. Denn es gilt: $A(B_1, B_2) = (A B_1, A B_2)$
- Fall 4: Falls $m=n=p=1$ gilt, dann werden die beiden Elemente multipliziert und zur resultierenden Matrix C hinzuaddiert.

12

Analyse des Cache-Oblivious Algorithmus

Theorem: Der Algorithmus benötigt $\Theta(mnp)$ Zeit im RAM Modell. Die Cache Komplexität beträgt $\Theta((m+n+p)(mn+np+mp)/L+mnp/L\sqrt{Z})$ Cache-Misses.

Theorem: Zur Multiplikation zweier $n \times n$ Matrizen benötigt der Algorithmus REC_MULT $\Theta(n^3)$ Zeit im RAM Modell und $\Theta((n+n^2/L+n^3/L\sqrt{Z}))$ Cache-Komplexität.

Gegenüberstellung BLOCK_MULT:

$$\Theta(1+n^2/L+(n\sqrt{Z})^3(Z/L)) = \Theta(1+n^2/L+n^3/L\sqrt{Z})$$

13

Analyse des Cache-Oblivious Algorithmus

Intuitiv benutzt der Algorithmus den Cache effektiv, denn: sobald ein Unterproblem ganz in den Cache passt, können dessen Unterprobleme ohne einen einzigen Cache-Miss gelöst werden.

Deswegen sind Divide&Conquer Verfahren grundsätzlich sehr gut für große Datenmengen geeignet.

Divide&Conquer Verfahren sind grundsätzlich gute Kandidaten für Cache-optimale Algorithmen.

14

Algorithmus mit besserer Laufzeit

Verfahren von Strassen (1986) zur Matrixmultiplikation

Auch ein Divide&Conquer Verfahren: jede Untermatrix wird in 4 Viertel der Höhe und Länge jeweils ungefähr $n/2$ zerlegt

RAM-Komplexität: $\Theta(N^{\log 7}) = \Theta(N^{2.81})$

Cache-Komplexität: $\Theta((n+n^2/L+n^{\log 7}/L\sqrt{Z}))$

In Praxis: sehr aufwändig zu implementieren

Für $n \leq 1.000.000$ noch ungefähr genauso langsam wie REC_MULT

Offenes Problem!

Eher theoretischer, aber überraschender Beitrag

ENDE