

Exercise 2:

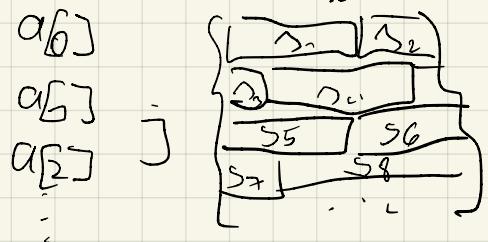
$$\text{elements per cacheline} = \lceil \frac{n}{\text{sizeof(int_32_t)}} \rceil$$

$n \times n \rightarrow S$
nxn Matrix
cacheline size

Variante 1:

Schleife iteriert zuerst über

$a[0]$; $a[m]$ Zeilen, dann über die Spalten



wenn auf ein Matrixelement zugriffen wird, werden "elements per cacheline" Elemente der Matrix in der Cacheline gepackt.

da $n \gg S$ ist, wird in einer Cacheline nie eine ganze Matrixzeile gepackt



Zuerst werden

$a[0][0]$ bis $a[0][\text{elements per cacheline} - 1]$ Elemente in den Cache geladen (eine Cacheline)

* (das selbe passiert für b)

(das passiert bis wir vollkommen durch iteriert sind)

\Rightarrow wir haben insgesamt $\frac{n \cdot n}{\text{sizeof(int 32)}} \text{ cache}$

read misses für a

* das selbe gilt für b



$2 \cdot n \cdot n \cdot \text{sizeof(int 32 - t)}$

cache-read
- misses

Zellen müssen nie erneut in den Cache geladen werden, da wir immer zeilenweise alles abarbeiten und die Zellen dann „verwischen“ werden können.

Variante 2:

Bei der zweiten Implementierung wird zuerst auf die Spalten zugegriffen. Daraus dass die Elemente der Zeilenweise in den Cache geladen werden ändert sich nichts.

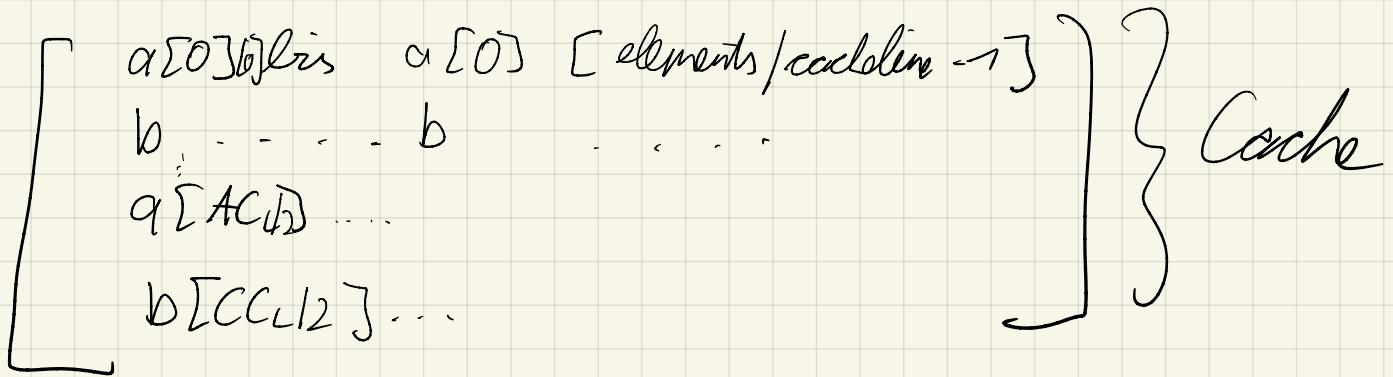
zuerst wird wiederum

$a[0][0]$ bis $a[0][\text{elements per cacheline} - 1]$
wir greifen aber immer auf $a[0, 0]$ $b[0, 0]$
 $a[0, 1]$ $b[0, 1]$
 $a[0, 2]$ $b[0, 2]$
= bis n zu

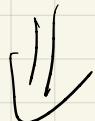
deshalb werden immer ganze cachelines reingeladen, aber immer nur auf ein Element zugegriffen

wenn der Cache nicht groß genug ist müssen Zeilen erneut geladen werden, dies hängt aber dann von der Größe des Caches sowie der Größe N ab.

Cachegröße \Rightarrow Anzahl CacheLines = AC,



davon kann man sehen, dass bei jedem Zugriff die CacheLine erneut in den Cache geladen werden muss



2. Non cache read misses