Cache models — cache-oblivious

I/O — cache-aware

parameters: B ... block size
M ... cache size

linear scan $O(N/B+1)$ reads

we generally assume $M \geq c \cdot B$
arbitrary constant

Sorting → Merge

run 1 — output — run 2

$T := |run1| + |run2|$

time $\Theta(T)$
#reads $O(T/B+1)$ if $M \geq 3B$

→ Mergesort

runs of size T

merge, merge, m., m., copy

runs of size $2T$

start with N runs of size 1 → after $\log N$ steps we have 1 run (sorted)

1 step: time $\Theta(N)$
#reads $O(N/B+1)$ → whole alg.: time $\Theta(N \cdot \log N)$
#reads $O(\frac{N}{B} \cdot \log N + \cancel{\log N}\,1)$

also upper bound for cache-aware model

→ K-way Merge K runs → 1 run

K runs — output — heap — in cache

time $\Theta(\cancel{T \cdot K})$ $\Theta(T \cdot \log K)$

#reads $O(T/B + K)$ $+1$ if runs are consecutive   if $M \geq (K+1)B + K + O(1)$ ... so $M \geq 2KB$ is sufficient
scans

we can set $K := \frac{M}{2B}$

→ K-way Mergesort

In every step, we merge K-tuples of runs
↳ #steps $\leq \log_K N = \frac{\log N}{\log K}$

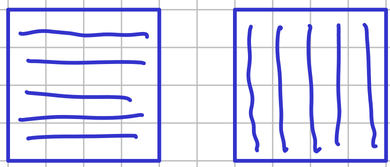
1 step: time $\Theta(N \cdot \log K)$
#reads $O(N/B + 1)$ → all steps: time $\Theta(N \cdot \cancel{\log K} \cdot \frac{\log N}{\cancel{\log K}})$
#reads $O(\frac{N}{B} \cdot \frac{\log N}{\log K} + 1)$

#reads is $O\left(\frac{N}{B} \cdot \frac{\log N}{\log M/B} + 1\right)$

1. this is known to be optimal in the I/O model (permutation bound)
2. also works in cache-aware model
3. the same time + I/O complexity is reached by Funnelsort in the cache-oblivious model.

Matrix Transposition of square matrix $N \times N$.

How is a matrix stored



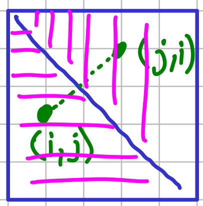
row-major order ✓

column-major order
Fortran & MATLAB

1 Trivial transpose

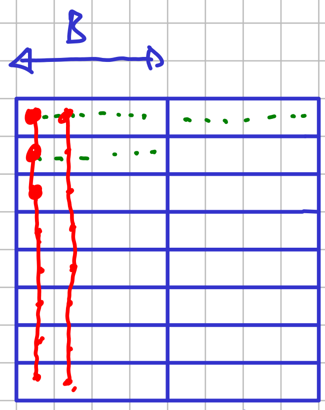
$\Theta(N^2)$ time

$\Theta(N^2)$ I/Os



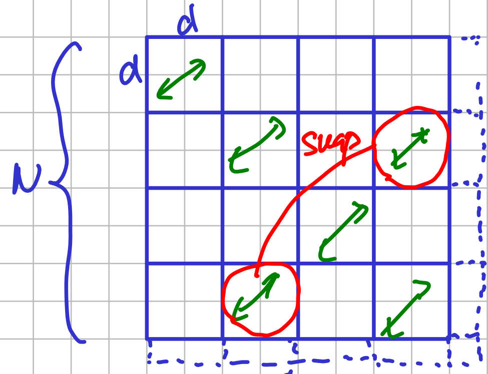
row scan
⇓
consecutive in memory
⇓
$\#\text{reads} \in O(N^2/B)$

Column scan: $\#\text{reads} \in \Theta(N^2)$
unless $M \geq N \cdot B$



row-major
assume $B \mid N$

2 Use tiles of size $d \times d$



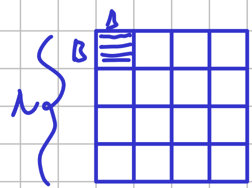
- transpose every tile
- swap tiles in symmetric pairs
- if 2 tiles fit in the cache, both tile transpose & tile swap done in the cache

in general: rectangular tiles at the border

we want: $\frac{d^2}{B} \cdot \left(\frac{N}{d}\right)^2 = \frac{N^2}{B}$

a) if N is a multiple of B:

we set $d := B$



each row of a tile is a block
⇓
tile spans B full blocks

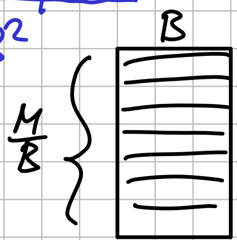
$\#\text{reads to load tile to cache} = B$

$\Theta(B)$ I/Os per tile
$\left(\frac{N}{B}\right)^2$ tiles
total I/O $O\left(\frac{N^2}{B}\right)$

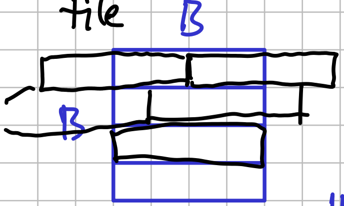
But we need $M \geq 2B^2$

tall cache assumption

$M \geq c \cdot B^2$



b) general N ... we still set $d := B$



each row spans max. 2 blocks
⇒ # reads to load a tile $\leq 2B \in \Theta(B)$

$$\#\text{tiles} = \lceil N/d \rceil^2 = \lceil N/B \rceil^2 \leq \left(N/B + 1\right)^2 \in O\left(\frac{N^2}{B^2} + 1\right)$$

total time: $\Theta(N^2)$

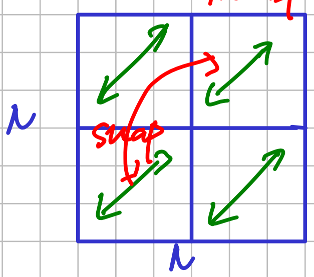
total I/Os: $O\left(B \cdot \left(\frac{N^2}{B^2} + 1\right)\right) = O\left(\frac{N^2}{B} + 1\right)$

$M \geq 4B^2$

optimal cache-aware algorithm

③ Cache-oblivious – Divide & Conquer alg.

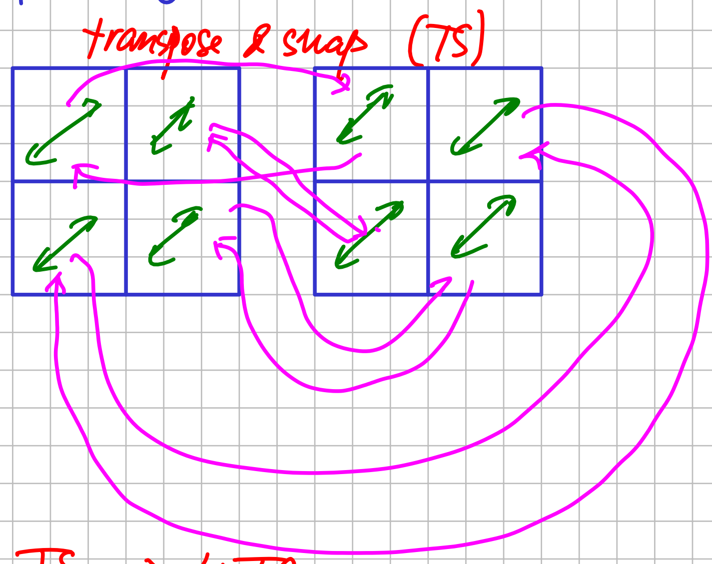
transpose (T)



4 quadrants of size $N/2$

recurse on them

transpose & swap (TS)

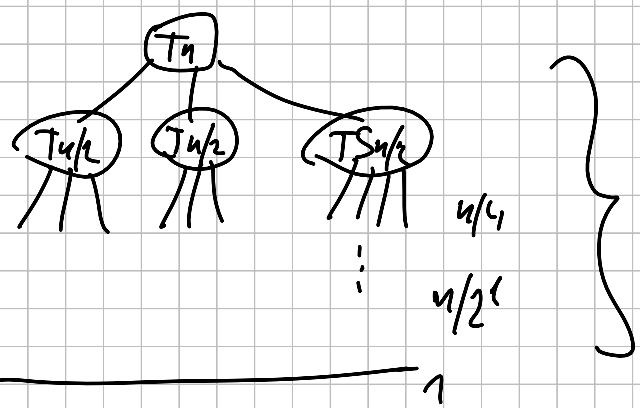


temporarily assume $N = 2^k$

$T_n \to 2T_{n/2} + TS_{n/2}$

$TS_n \to 4\,TS_{n/2}$

tree of recursion



$\log N$ levels

at most 4^l subproblems at level l of size $N/2^l$

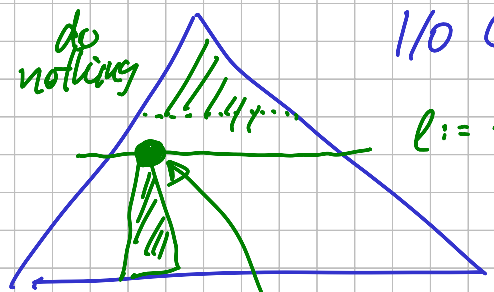
$\Rightarrow 4^{\log N}$ leaves , $O(1)$ time per leaf

$(2^{\log N})^2 = N^2$

int. nodes $\leq$ # leaves $\leq N^2$ $\quad$ $O(1)$ time per int. node

$O(N^2)$ time

I/O Complexity



l := topmost level at which problem size $\leq B$

$N/2^l \leq B$ $\quad$ but also $N/2^{l-1} > B$

$N/2^l > B/2$

load tile to the cache

... $O(B)$ I/Os

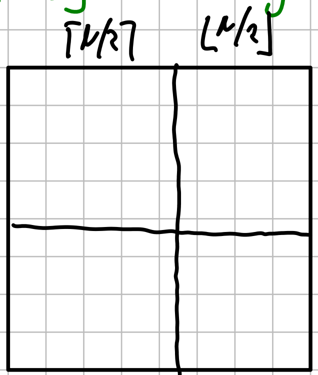
$B/2 < N/2^l \leq B$

$N/2^l \in \Theta(B)$

but: the subproblems at level l form a tiling of the whole matrix

$\hookrightarrow$ by reasoning from ② $\quad$ # I/Os $\in O(N^2/B + 1)$

upper bound in the c/o model



for general N

prove that all subproblems are almost square : sides differ by at most 1.

Proof: Exercise.