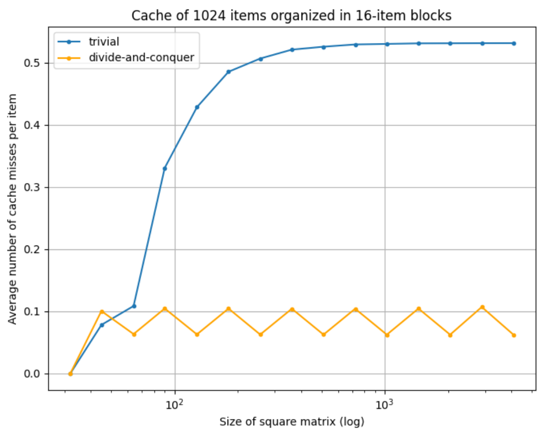
Jan Maděra
janmadera97@gmail.com

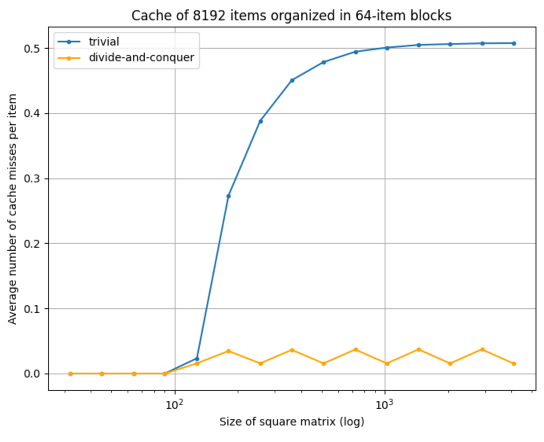
Matrix transposition experiment

2021/04/29

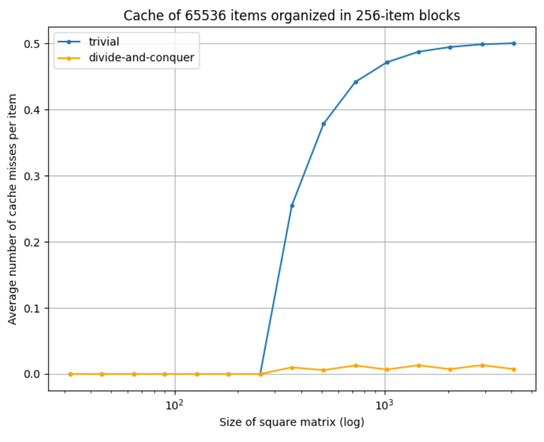
My random seed: 35



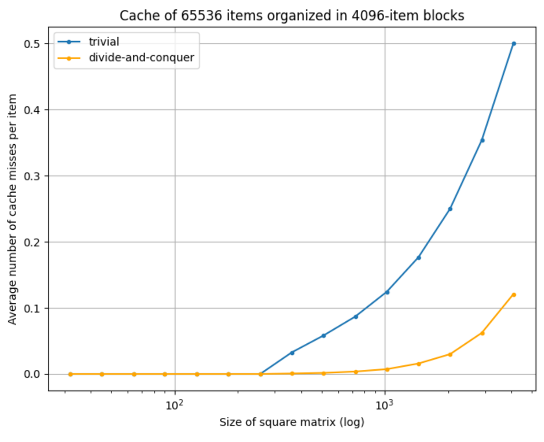
(a)



(b)



(c)



(d)

Figure 1: Average number of cache misses per item with dependence on size of square matrix (length of row, column).

1 Small matrix fit to cache

We can see that there are no cache misses for small enough matrices that probably fit to the cache whole. The max size N of $N \times N$ square matrix that fit to cache is in our cases (rounded toward zero):

Cache size	N
1024	32
8192	90
65536	256

Table 1: Maximal size of square matrix "side" N for which the whole matrix fit in the cache with given size.

2 Naive implementation cache misses increase rapidly

We can see in graphs that when the matrix does not fit to the cache, the average number misses of trivial implementation grow fast and probably converges close to 50% of average cache misses per item. In graphs of smaller caches this number seems to be higher than 50%.
The average number of cache misses of *divide-and-conquer* implementation seems to grow only slightly and then stays in constant range, which is getting smaller with growing cache size. This will be described in more detail in the section 4.

3 The cache misses do not grow rapidly right after the matrix does not fit to the cache

We can also see for example in figure 1a that the cache misses grow similarly in both cases of trivial and DaC implementations between values of N equal to 32 and 64 (first three values). This is caused by the fact that if the rows are larger than the size of blocks B (16 in case of figure 1a), each item in the column that is being read is located in a different block. When we read next column, we usually need the same N blocks. In cache of size 1024 with blocks of size 16, we have exactly 64 blocks. Therefore, when the matrix side is smaller or equal to 64, the column being read can fit to the cache. When we read the next column, it should be also in the cache as long as it holds that the size of cache $M \in \Omega(NB)$.

4 Comparison of cache with size 65536 with 256-item blocks and 4096-i. b.

If we compare the two graphs, the cache with 256-item block looks like it has similar number of average cache misses with arbitrary size of square matrix. On the other hand, the average number of cache misses in cache with 16 x 4096-item blocks looks like it grows with increasing size of square matrix. From lectures we know that the cache-oblivious model (which divide-and-conquer method transpose method is) has I/O complexity $\mathcal{O}(N^2/B+1)$ which matches complexity the cache-aware algorithm up to a constant. We know that the divide and conquer method approximate the tile size by recursive subdivisions and works with cache similarly to the cache-aware method which uses tiles. For the tile method to work, the cache must be able to hold two tiles at once. In our case this means that cache size should be at least $2B^2 = 2 \cdot 4096^2$. As this does not hold, the two tiles do not fit to cache and the transposition will be I/O inefficient and the number of cache misses will grow faster than in cases of caches in figures 1a and 1b.
But cache with 256-item blocks in figure 1c similarly to the one with 4096-i. b. has too big blocks ($2 \cdot 256^2 > 65536$), so why does it look like the average number of the cache misses per item stays in constant range?
This is caused by the fact that in two blocks of size 256×256 there are only two times more elements than is the size of cache (65536). On the other hand in case of cache with blocks of size 4096×4096, the two blocks have 512 times more elements than is the size of cache (65536). If we look at the values of number of cache misses in the case of cache with size 65536 with 256-item blocks (rounded to four decimal places), we can clearly see the increasing trend:

N	avg. cache misses
362	0.0100
512	0.0059
724	0.0129
1024	0.0068
1448	0.0132
2048	0.0073
2896	0.0134
4096	0.0076

This increasing trend is not seen in cache of size 65536 with 128-item blocks which fits at least two blocks of size 128×128:

N	avg. cache misses	$N/128$
362	0.0125	2.828
512	0.0078	4
724	0.0132	5.65
1024	0.0078	8
1448	0.0133	11.312
2048	0.0078	16
2896	0.0133	22.625
4096	0.0078	32

Differences in average misses are mostly caused by the fact that the size of matrix N is not divisible by size of tile 128 which increases cache misses as there are on average less elements loaded to the cache when cache misses.