COMPUTER ORGANIZATION
2023 SPRING
FINAL PROJECT PART 2

0811128 林洋宏

310511050 張祐誠

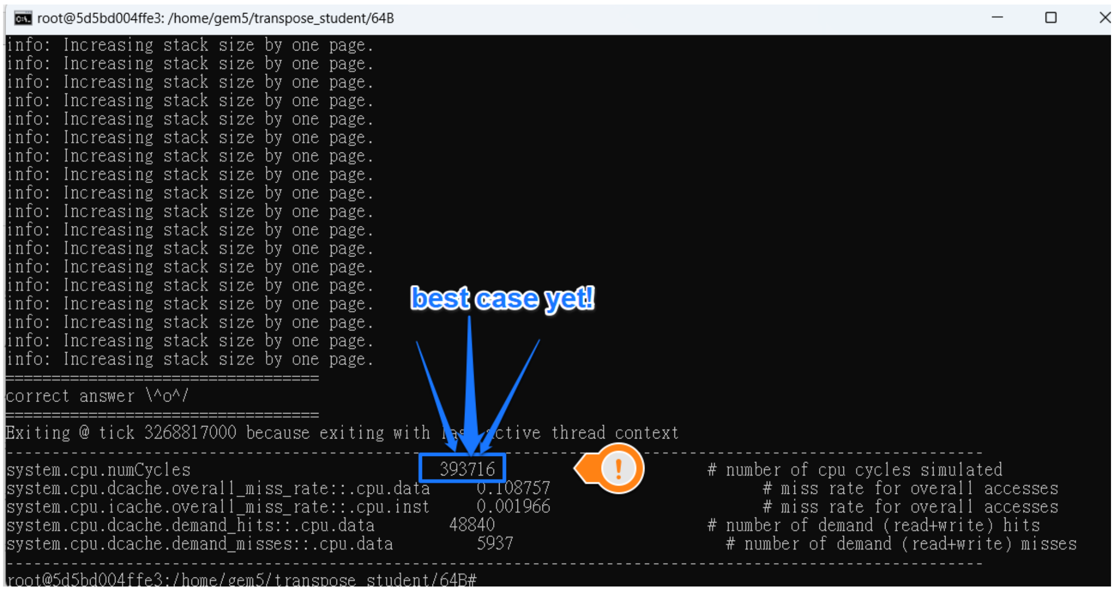
109612029 郭子維

Function Correctness&Best Performance (80%)

Cachline size = 64B, cache size = 2kB (in transpose/64B/), matrix size = 136*136

Baseline=404,972±10,000

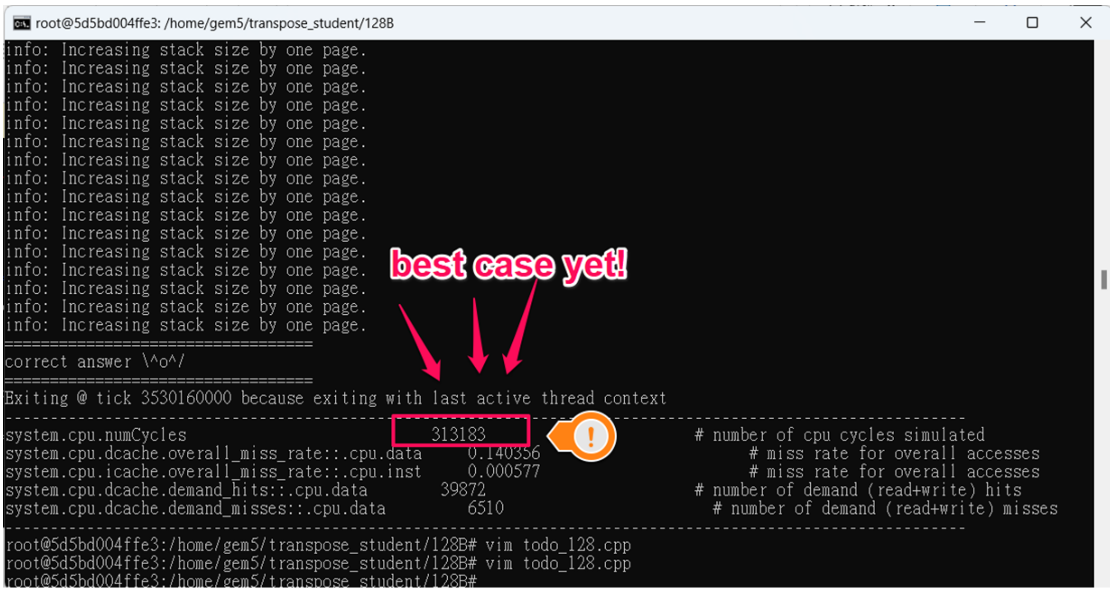
Our performance=393,716



Cachline size = 128B, cache size = 4kB (in transpose/128B/), matrix size = 144*144

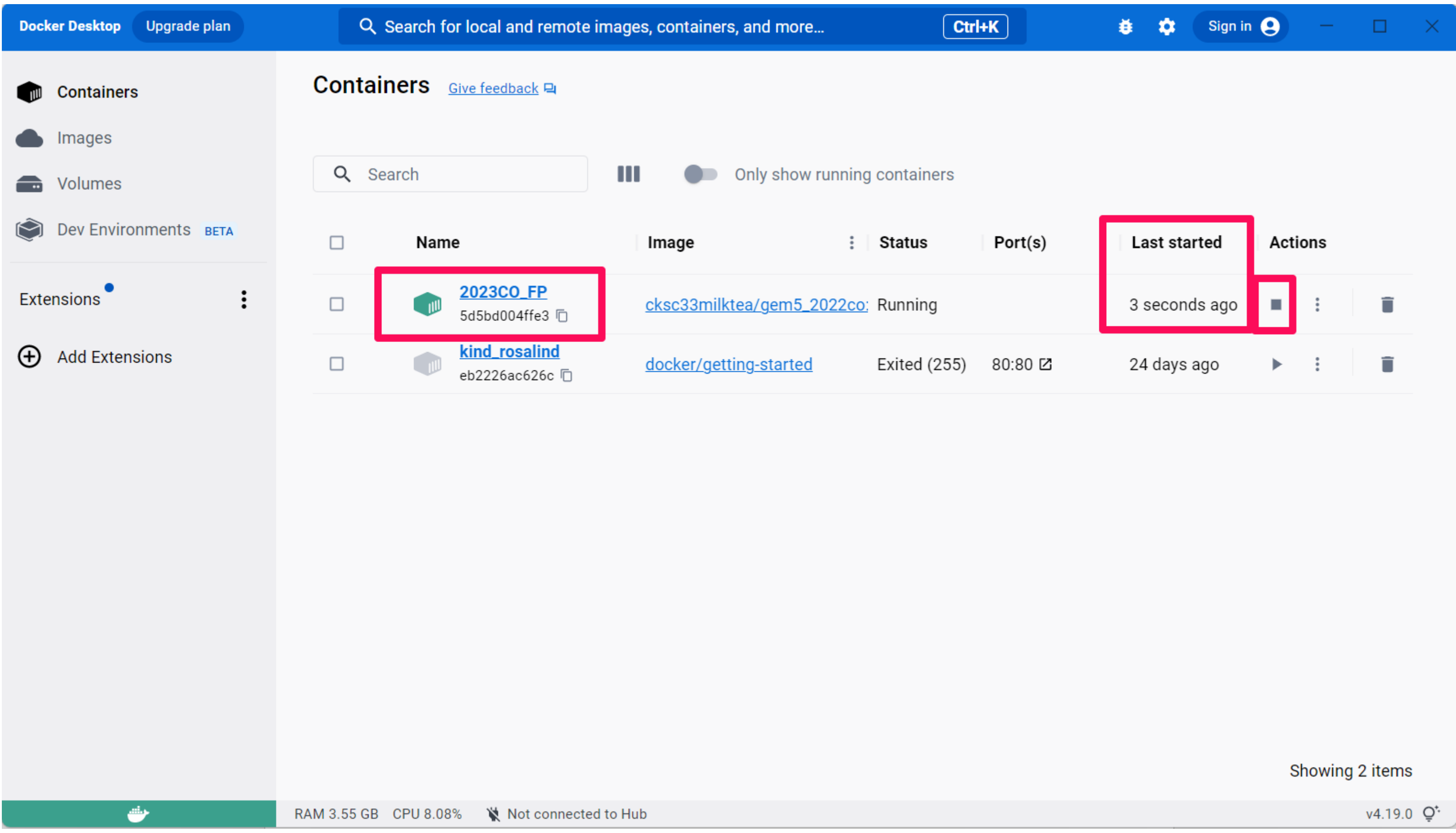
Baseline=409,695±10,000

Our performance=313,183



Report (20%)

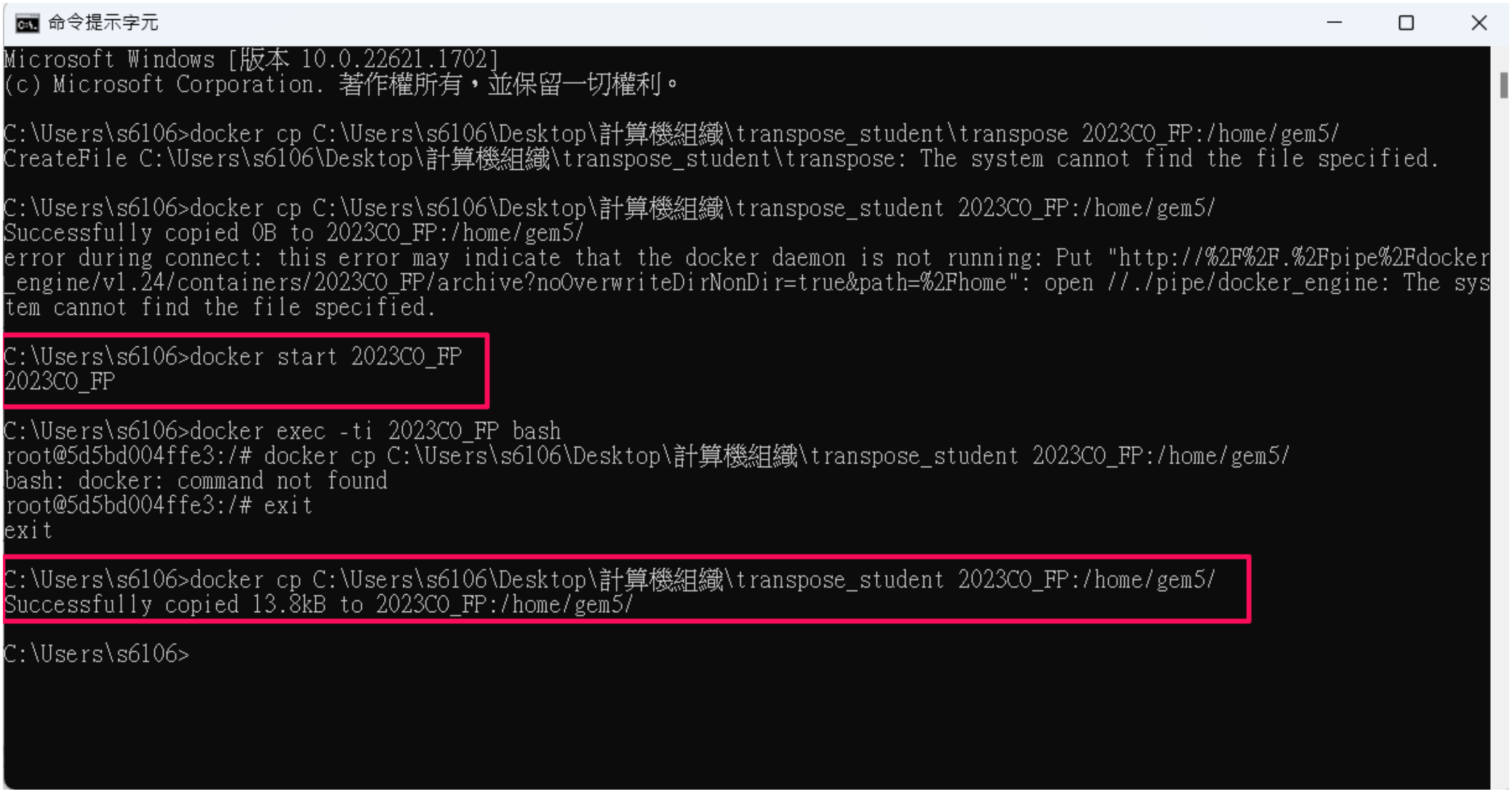
○ Screenshot of your results



1. Start the container and enter
2. its command line (Open Docker first):

docker start 2023CO_FP

docker cp YOUR_DOWNLOAD_PATH\transpose 2023CO_FP:/home/gem5/



3. Start docker image:

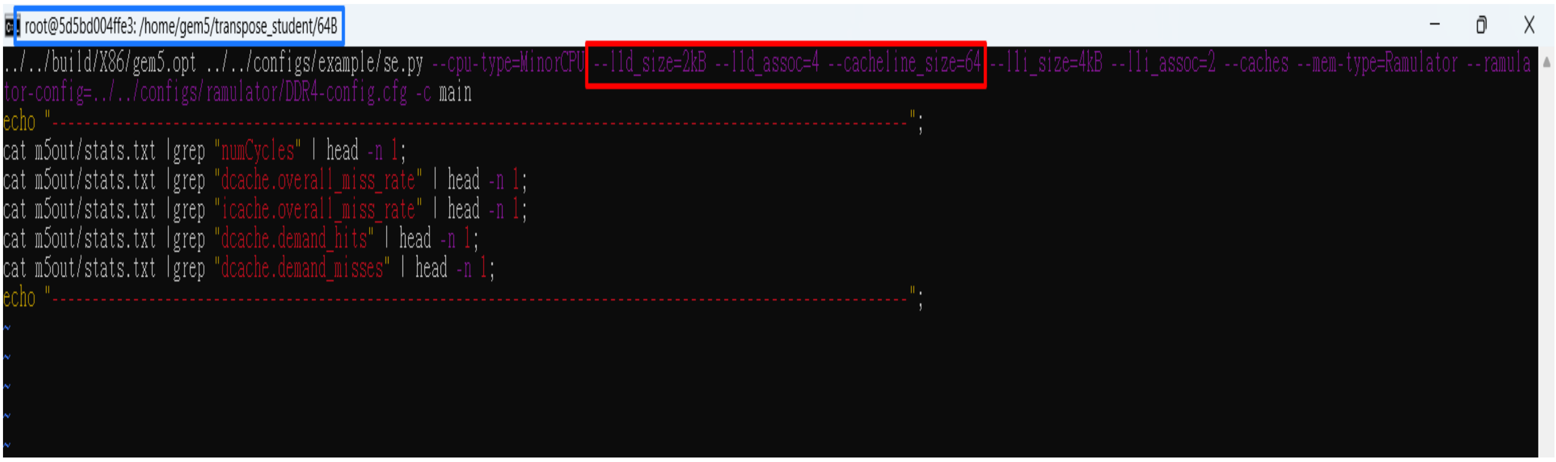
docker start 2023CO_FP

docker exec -ti 2023CO_FP bash

4. Revise todo_128.cpp and todo_64.cpp:

cd home/gem5/transpose_student/

cd 128B/ or cd 64B/



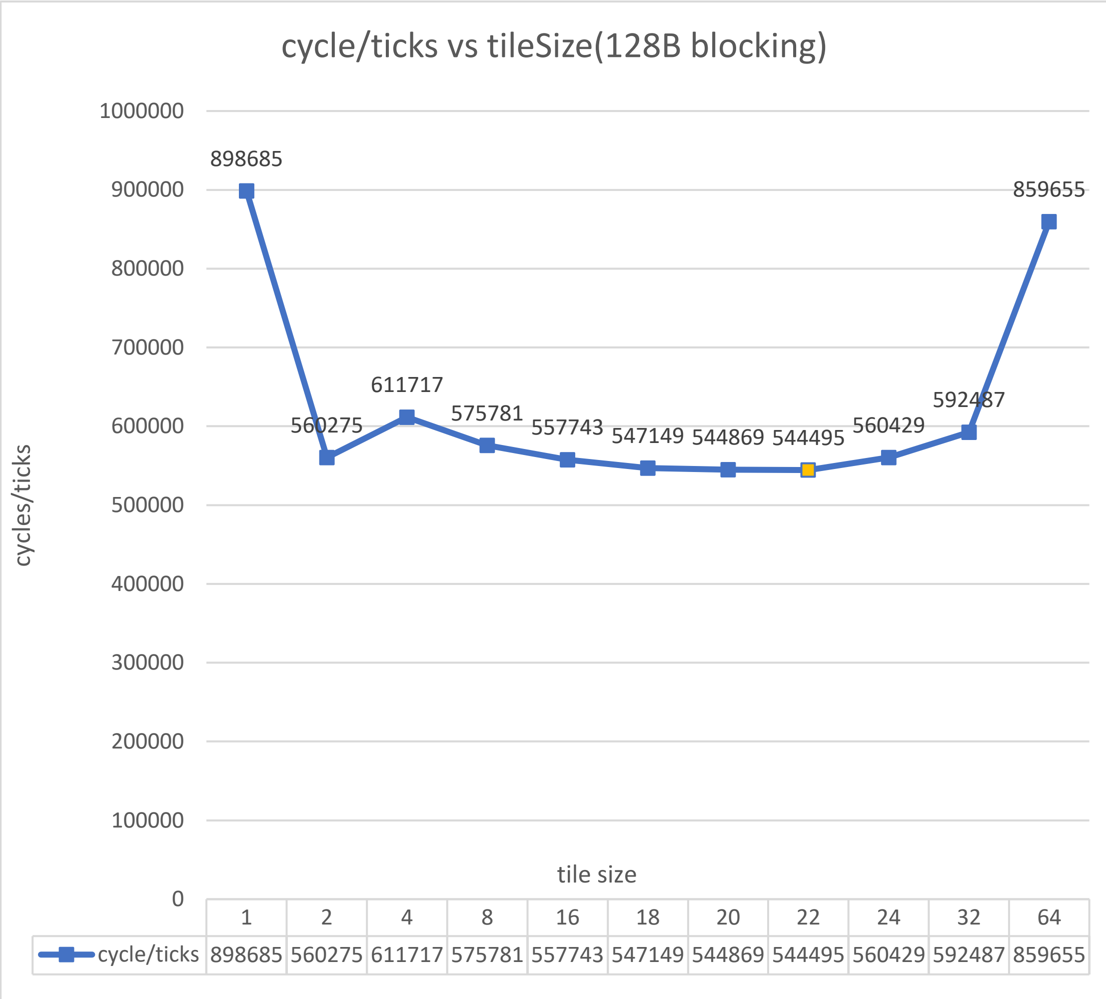
Run run_sim.sh:

make

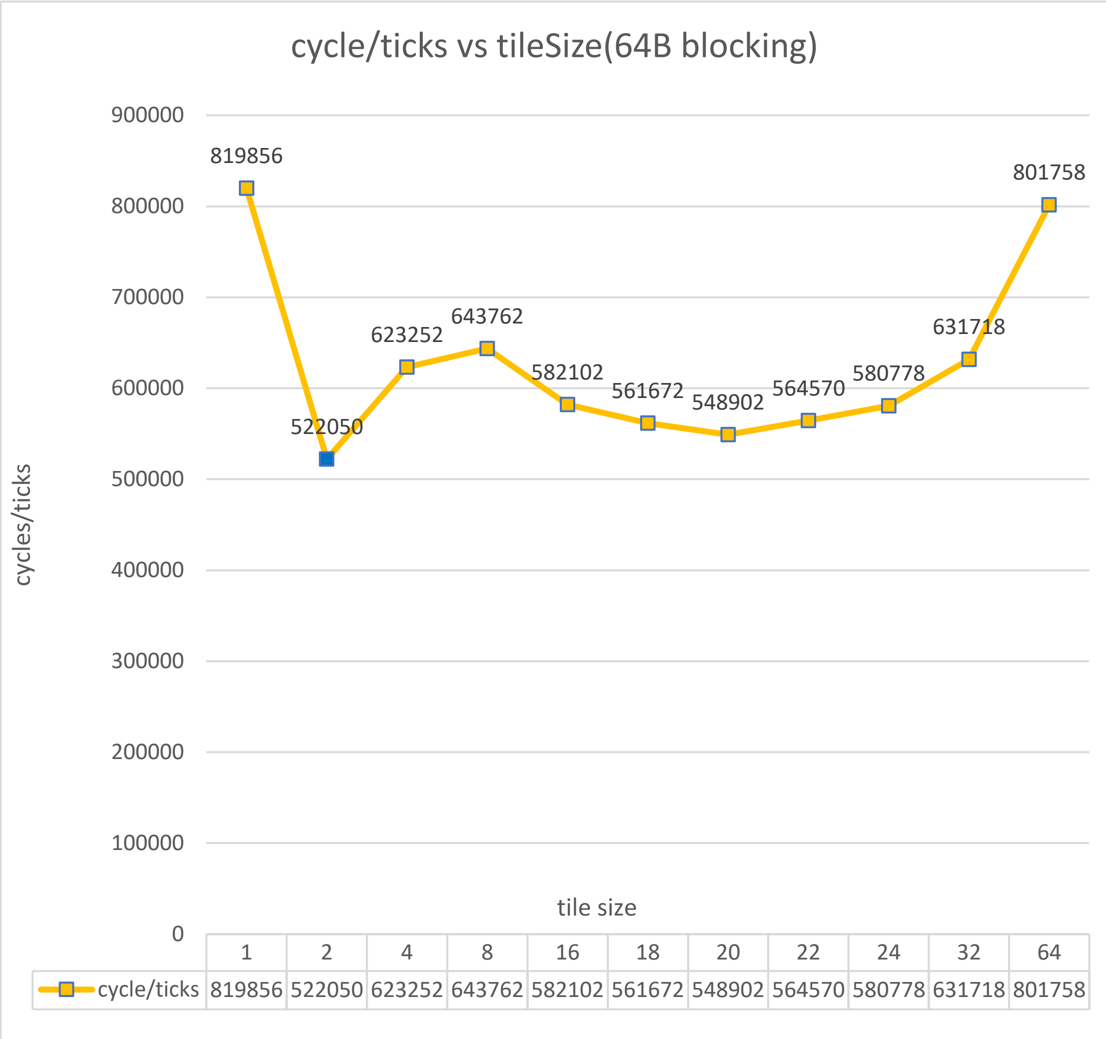
./run_sim.sh

Blocking method:

128B		64B	
tileSize	cycles/ticks	tileSize	cycles/ticks
1	898685	1	819856
2	560275	2	522050
4	611717	4	623252
8	575781	8	643762
16	557743	16	582102
18	547149	18	561672
20	544869	20	548902
22	544495	22	564570
24	560429	24	580778
32	592487	32	631718
64	859655	64	801758



The best case for 128B cahceline with blocking is 544495 cycles.



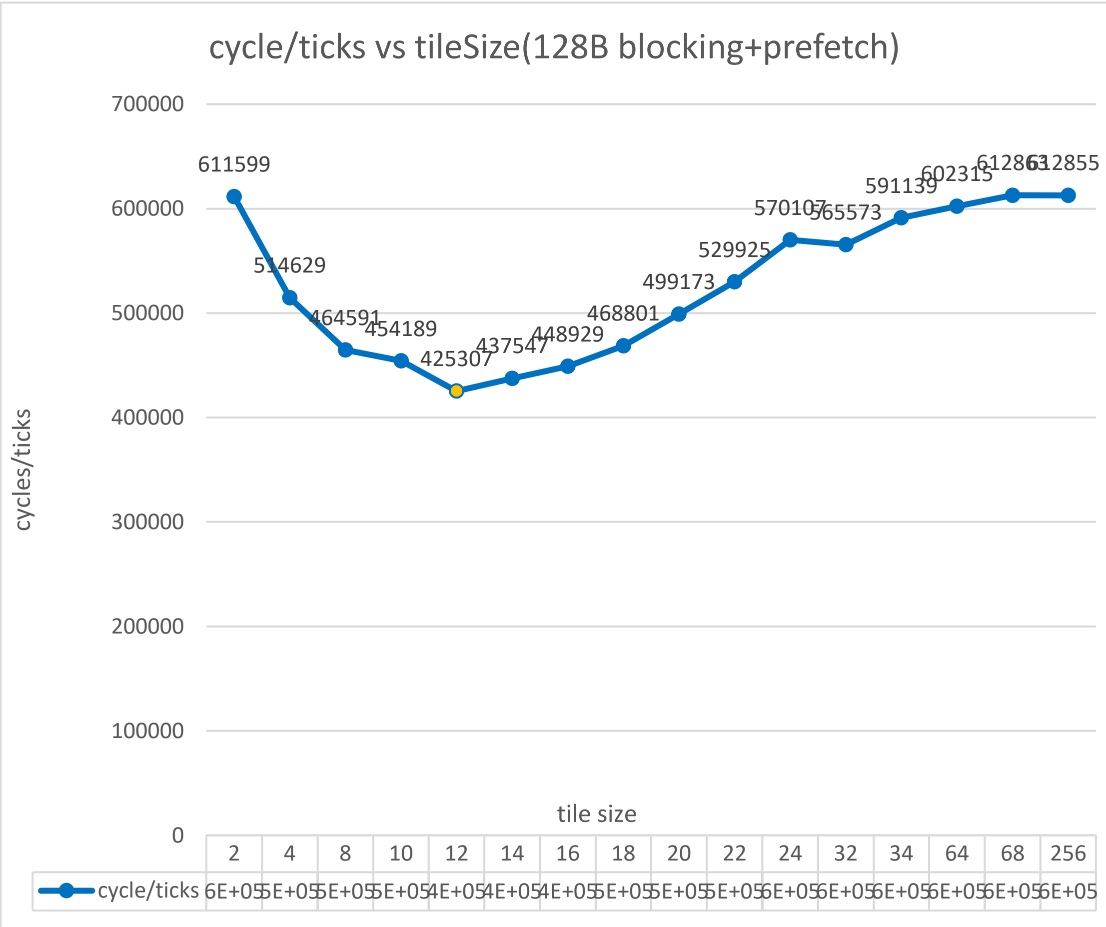
The best case for 64B cahceline with blocking is 522050 cycles.

Conclusion: optimizing using only blocking method is not enough.

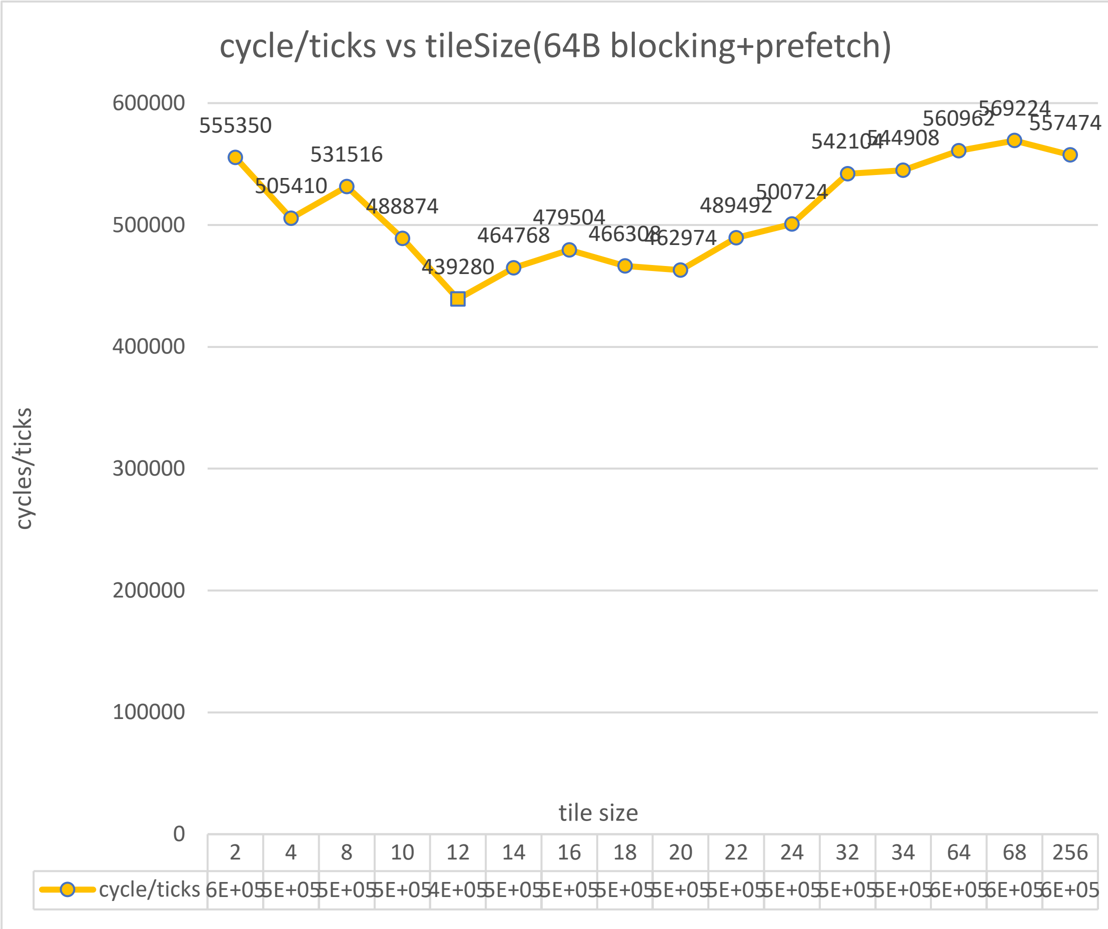
Blocking + prefetch:

128B		64B	
tileSize	cycles/ticks	tileSize	cycles/ticks
2	611599	2	555350
4	514629	4	505410
8	464591	8	531516
10	454189	10	488874
12	425307	12	439280

14	437547	14	464768
16	448929	16	479504
18	468801	18	466308
20	499173	20	462974
22	529925	22	489492
24	570107	24	500724
32	565573	32	542104
34	591139	34	544908
64	602315	64	560962
68	612863	68	569224
256	612855	256	557474



The best case for blocking + prefetch for 128B cachline is 425307 cycles.



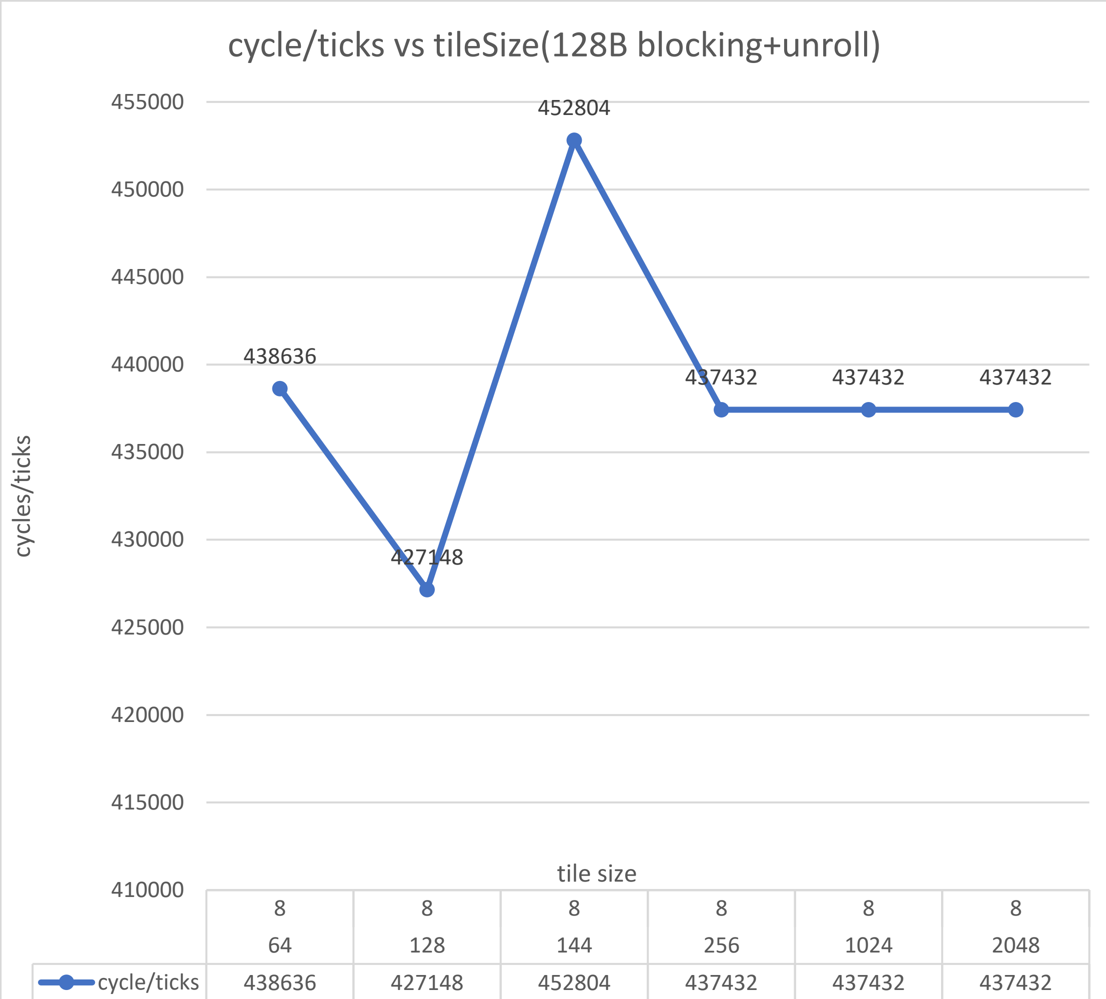
The best case for blocking + prefetch for 64B cachline is 439280 cycles.

Conclusion: we are on the right track!

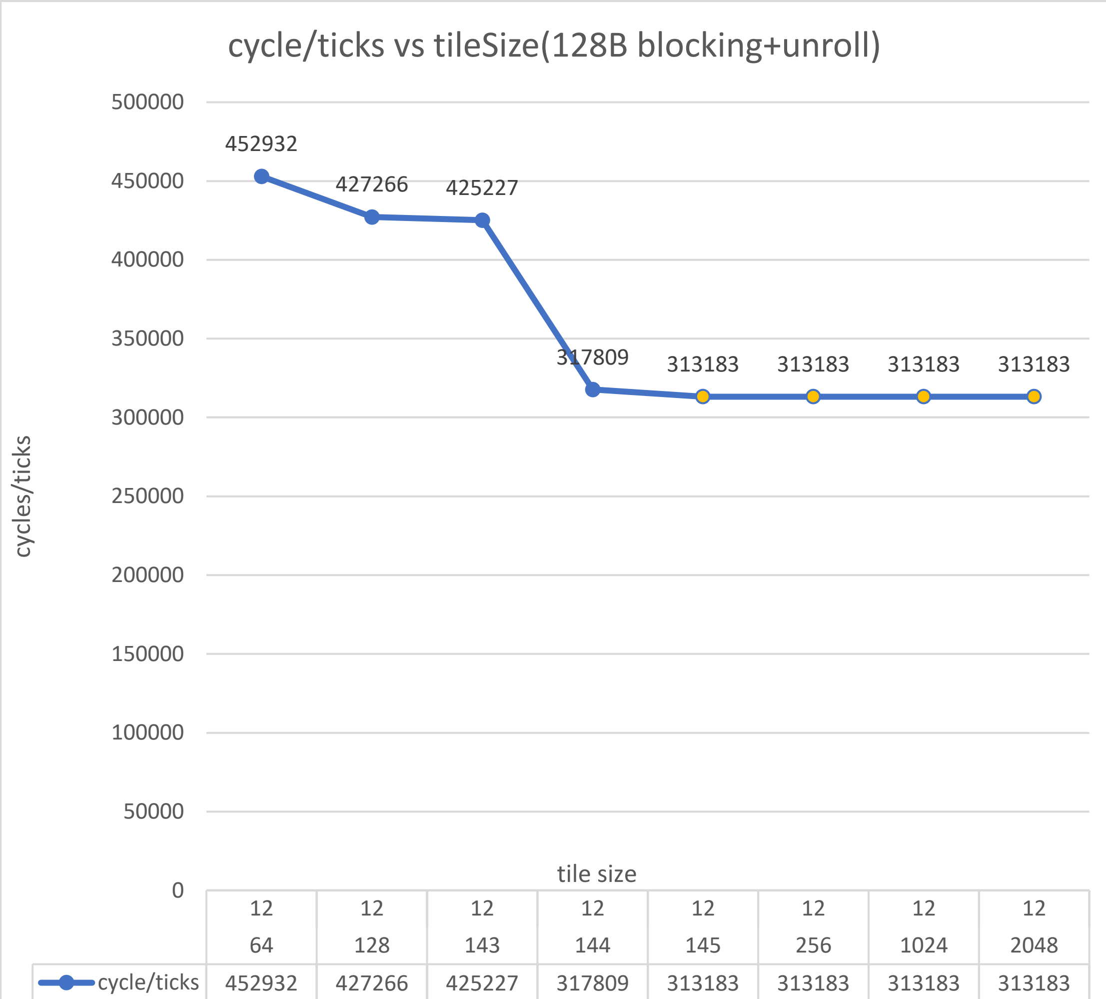
Blocking + unroll:

128B			64B		
tileSize	unrollFactor	cycle	tileSize	unrollFactor	cycle
64	8	438636	64	8	484718
128	8	427148	128	8	475938
144	8	452804	136	8	489164
256	8	437432	256	8	474514
1024	8	437432	1024	8	474514
2048	8	437432	2048	8	474514
64	12	452932	64	12	535552
128	12	427266	128	12	498440
143	12	425227	136	12	423650
144	12	317809	137	12	***414534***

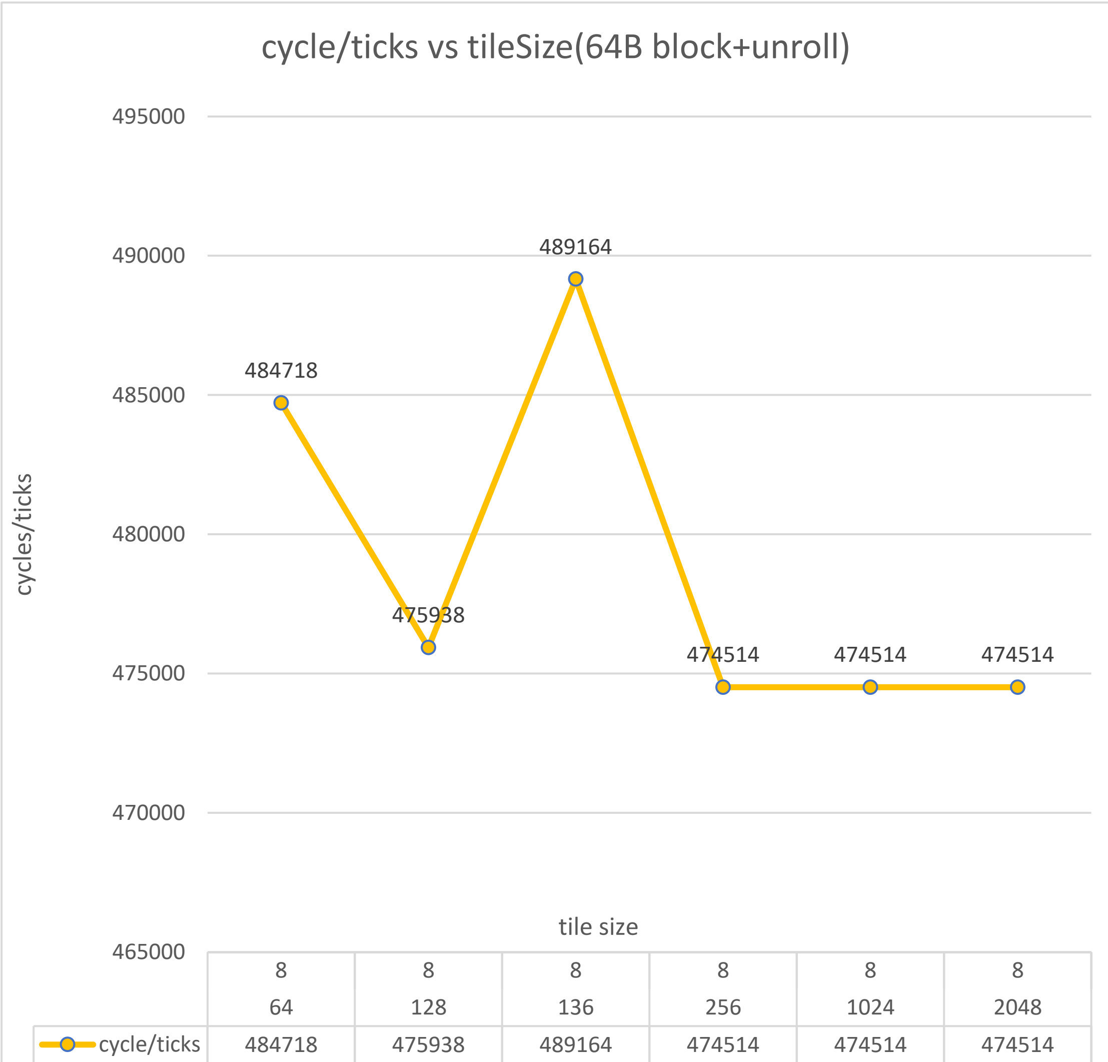
145	12	313183	256	12	***414534***
256	12	313183	1024	12	***414534***
1024	12	313183	2048	12	***414534***
2048	12	313183			



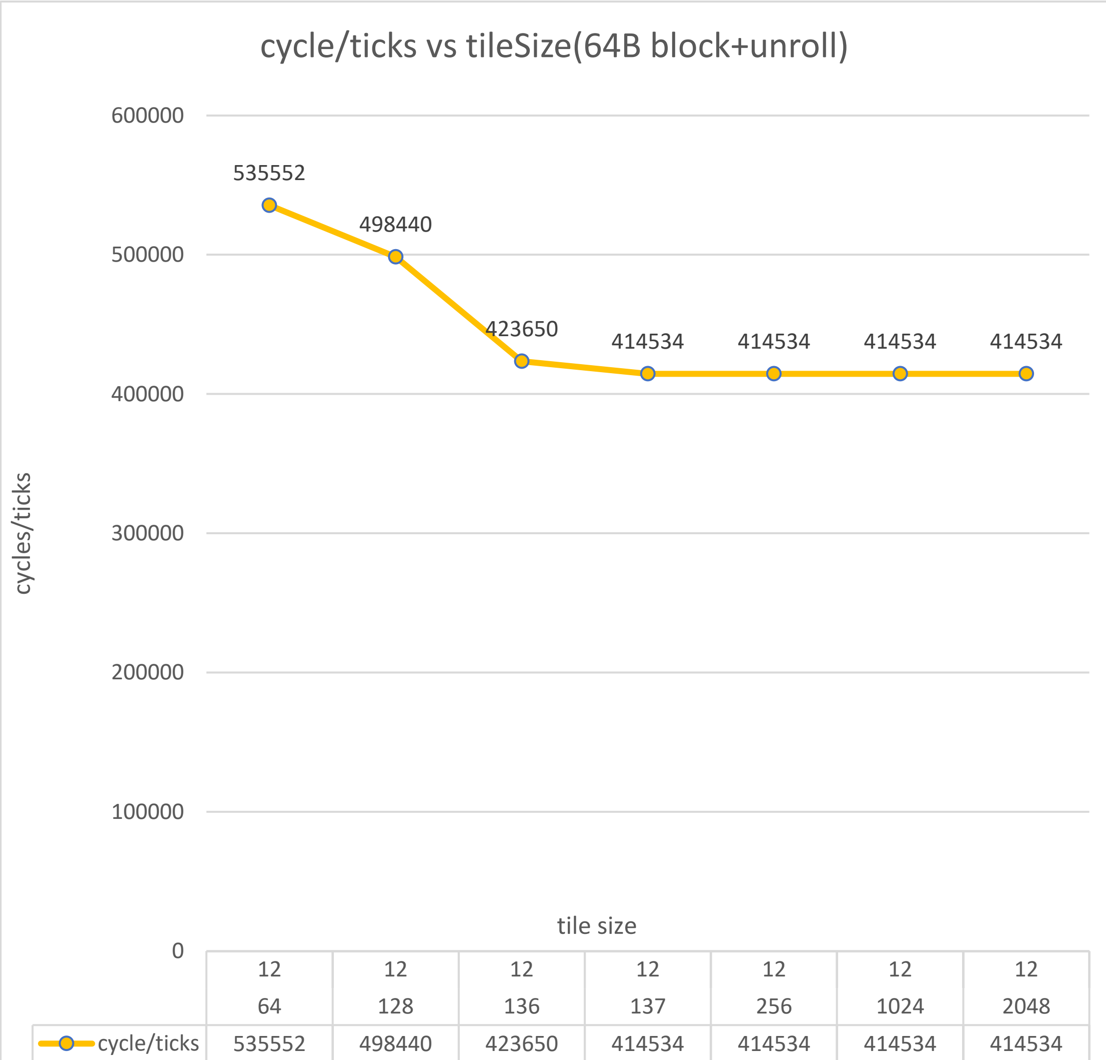
Let unrollfactor(upper) be set to 8, and tileSize(lower) be set to 64~2048, we can see it converges to 437432 cycles.



Let unrollfactor(upper) be set to 12, and tileSize(lower) be set to 64~2048, we can see it converges to 313183 cycles.(great improvement)



Let unrollfactor(upper) be set to 8, and tileSize(lower) be set to 64~2048, we can see it converges to 474514 cycles.



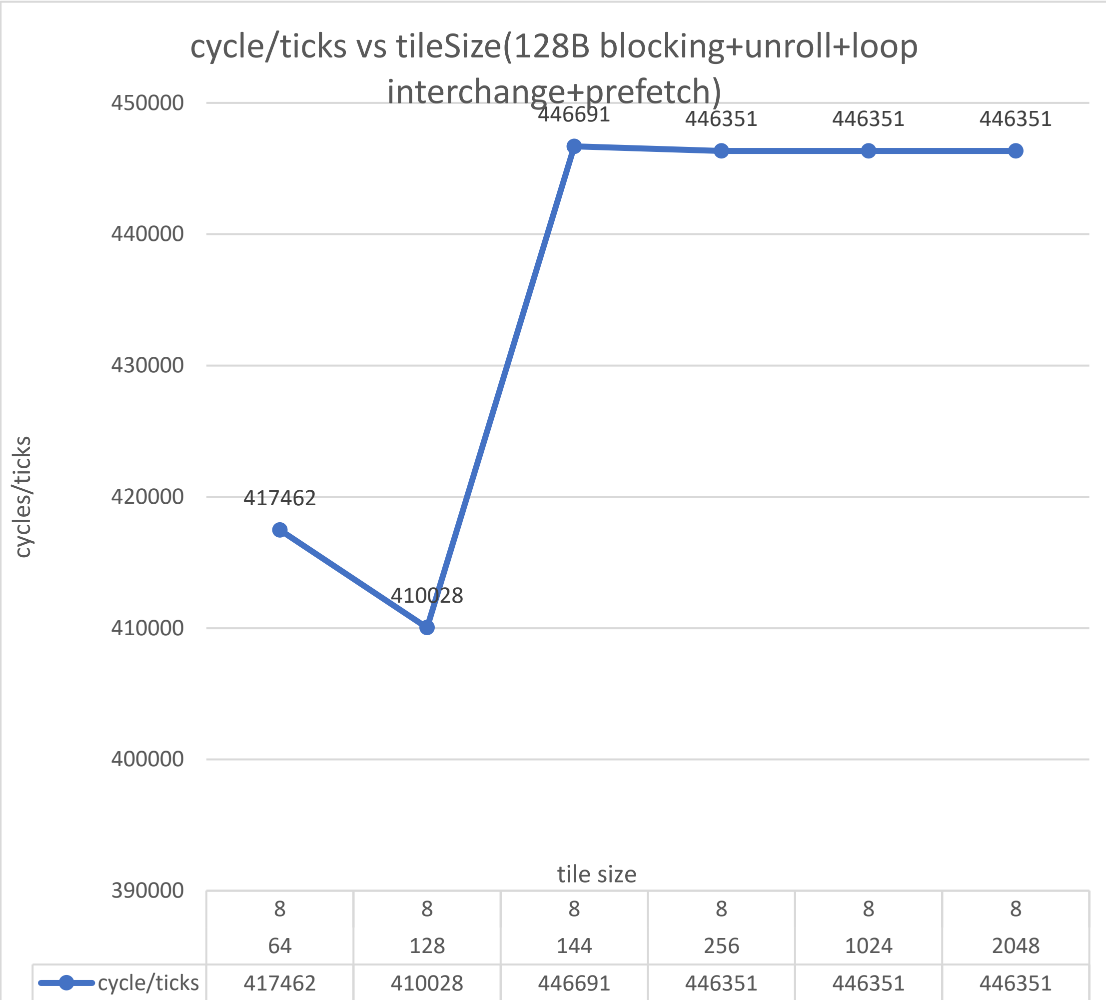
Let unrollfactor(upper) be set to 12, and tileSize(lower) be set to 64~2048, we can see it converges to <u>414534 cycles.</u>

Conclusion, we have reached the goal! But there are still possibilities of optimizing even further on 64B cache line.

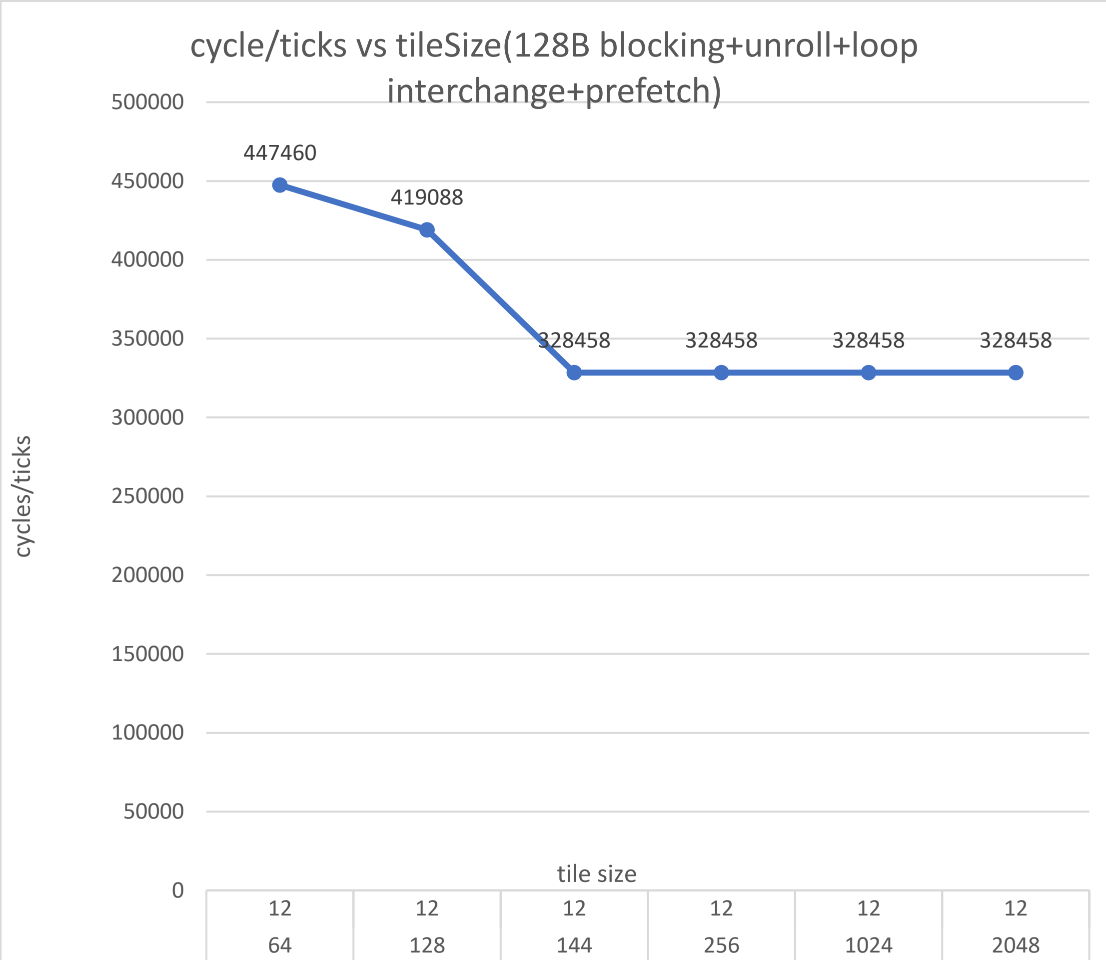
Blocking + Unrolling + loop interchange+ prefetch:

128B			64B		
tileSize	unrollFactor	cycle	tileSize	unrollFactor	cycle
64	8	417462	64	8	474054
128	8	410028	128	8	464948
144	8	446691	136	8	463442
256	8	446351	256	8	463944
1024	8	446351	1024	8	463944

2048	8	446351	2048	8	463944
64	12	447460	64	12	482296
128	12	419088	128	12	431100
144	12	328458	136	12	393716
256	12	328458	256	12	393716
1024	12	328458	1024	12	393716
2048	12	328458	2048	12	393716

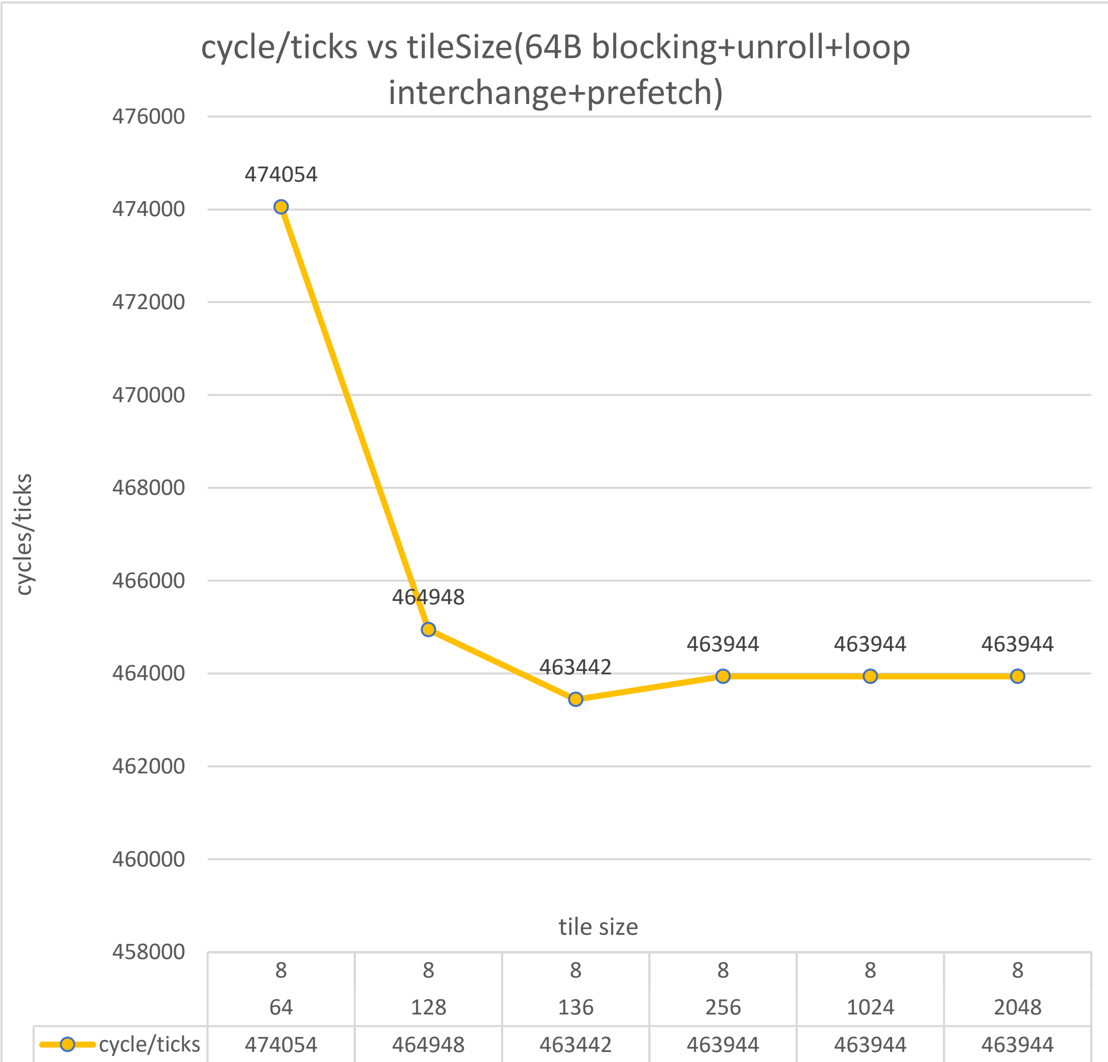


Let unrollfactor(upper) be set to 8, and tileSize(lower) be set to 64~2048, we can see it converges to 446351 cycles.

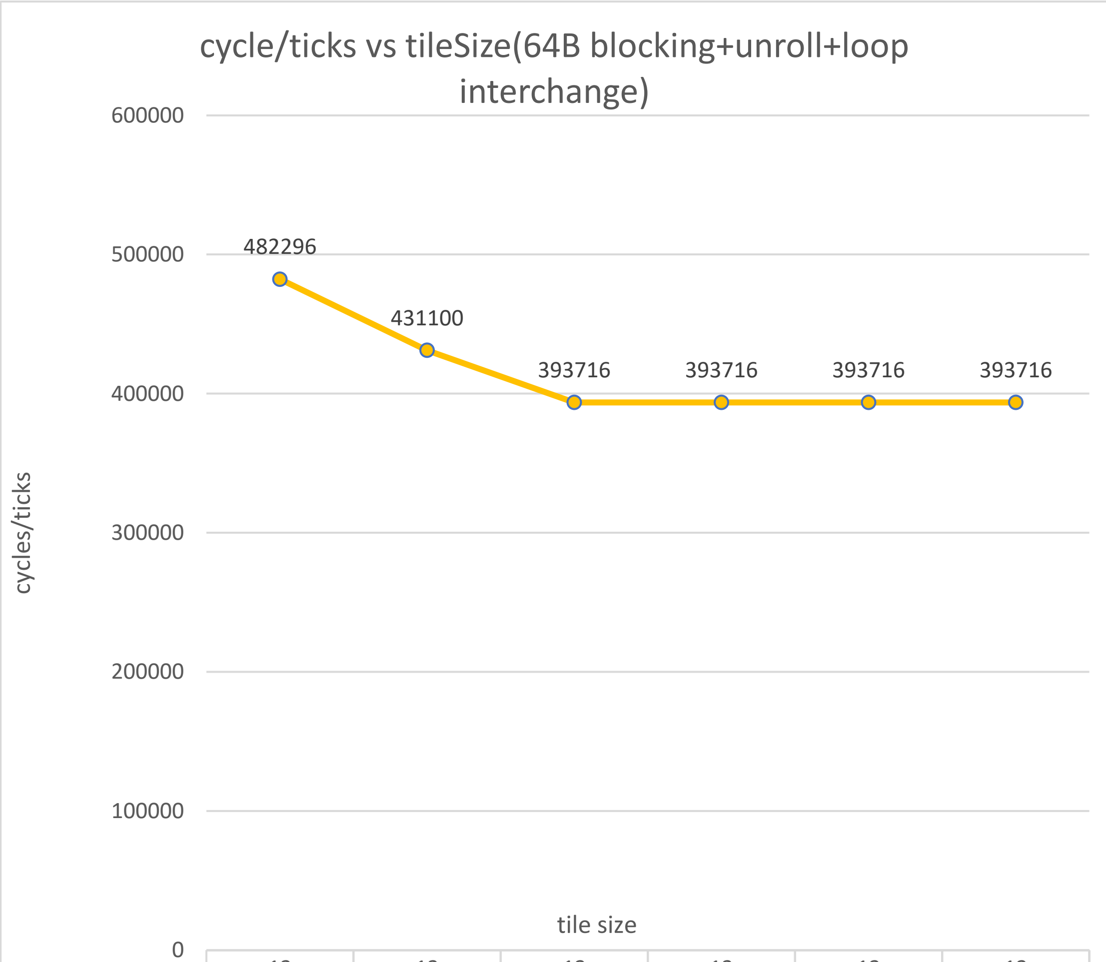


	12 64	12 128	12 144	12 256	12 1024	12 2048
cycle/ticks	447460	419088	328458	328458	328458	328458

Let unrollfactor(upper) be set to 12, and tileSize(lower) be set to 64~2048, we can see it converges to 328458 cycles.



Let unrollfactor(upper) be set to 8, and tileSize(lower) be set to 64~2048, we can see it converges to 463944 cycles.



	12	12	12	12	12	12
	64	128	136	256	1024	2048
cycle/ticks	482296	431100	393716	393716	393716	393716

Let unrollfactor(upper) be set to 8, and tileSize(lower) be set to 64~2048, we can see it converges to 393716 cycles.

Conclusion, 64B cacheline has been further optimized.

- Explain how did you optimize your code according to two different caches

128B: Best optimization→ Blocking + unrolling

```
void transpose(const uint64_t a[M][M], uint64_t b[M][M]){
const int tileSize = 1024; // Size of the tiles
```

```
const int unrollFactor = 8; // Number of loop iterations to process per
iteration
for (int ii = 0; ii < M; ii += tileSize) {
   for (int jj = 0; jj < M; jj += tileSize) {
      for (int i = ii; i < min(ii + tileSize, M); i += unrollFactor) {
         for (int j = jj; j < min(jj + tileSize, M); j++){
            for (int k = 0; k < unrollFactor; k++) {
               if (i + k < M) {
                 b[j][i + k] = a[i + k][j];
               }
            }
        }
      }
    }
  }
}
```

In this code, the matrix transposition is divided into smaller tiles of size 'tileSize', and the outer two loops iterate over these tiles. However, there is a difference in the inner loops. The loop iteration is unrolled for 'unrollFactor' times in the 'i' loop, and the 'j' loop is not unrolled. The 'k' loop iterates over the unrolled iterations.

Here's how these optimizations are associated with the cache characteristics:

1. Cache Line Size: The cache line size is 128 bytes. By transposing the matrix in tiles of size tileSize (1024), the code ensures that each tile fits within a few cache lines (1024 bytes / 128 bytes = 8 cache lines). This alignment with the cache line size improves cache utilization and reduces cache misses.
2. Cache Size: The cache size is 4kB, which is larger than the tile size (1024). This allows for efficient utilization of the cache. Each tile can fit within the cache, ensuring that a significant portion of the accessed data resides in cache memory. This reduces cache misses and improves overall performance.
3. LRU Policy: The LRU policy determines which cache lines to evict when the cache is full. By using loop tiling, the code enhances spatial locality and increases the chances of reusing cache lines within each tile. This aligns with

the LRU policy by maximizing cache hits within each set and reducing cache evictions.

4. Associativity: The cache has an associativity of 4, meaning each cache set can hold up to 4 cache lines. With loop tiling, the code accesses data within each tile in a localized manner, increasing the chances of cache hits within each set. This reduces conflicts and improves cache utilization, aligning with the given associativity.

In summary, the revised code optimizes cache utilization by employing loop tiling and unrolling techniques. The tiles fit well within the cache lines, ensuring efficient utilization of the cache. The cache size of 4kB provides large enough space for the tiles, reducing cache misses. The LRU policy and associativity of 4 align well with the code's focus on spatial locality, improving cache hit rates and overall performance. Therefore, this method can be optimized for a cache with a cache line size of 128B, cache size of 4kB, LRU policy, and associativity of 4 by effectively utilizing the cache through loop tiling and unrolling, reducing cache misses, and maximizing cache hit rates.

64B: Best optimization→ Blocking + unrolling +loop interchange + prefetch

```
#include <immintrin.h> // Required for prefetching

void transpose(const uint64_t a[M][M], uint64_t b[M][M]) {
   const int tileSize = 1024; // Size of the tiles
   const int unrollFactor = 8; // Number of loop iterations to process
per iteration

   for (int ii = 0; ii < M; ii += tileSize) {
      for (int jj = 0; jj < M; jj += tileSize) {
         for (int j = jj; j < min(jj + tileSize, M); j += unrollFactor)
{
            // Prefetch the next tile of matrix a
            _mm_prefetch((char*)&a[ii + tileSize][j], _MM_HINT_T0);

            for (int i = ii; i < min(ii + tileSize, M); i++) {
```

```
                for (int k = 0; k < unrollFactor; k++) {
                    if (j + k < M) {
                        b[j + k][i] = a[i][j + k];
                    }
                }
            }

            // Prefetch the next set of elements from matrix a
            _mm_prefetch((char*)&a[ii][j + unrollFactor], _MM_HINT_T0);
        }
    }
  }
}
```

In this code, the matrix transposition is divided into smaller tiles of size 'tileSize'. The outer two loops iterate over these tiles, and the inner loops perform the actual transposition within each tile. By processing smaller tiles at a time, the code takes advantage of spatial locality and reduces the number of cache misses.

The prefetching instructions (_mm_prefetch) are used to explicitly request the cache system to bring data into the cache ahead of time. This helps to minimize the impact of cache misses by fetching data from memory in advance, anticipating future usage.

The Loop Interchange instructions: in the innermost loop, we were accessing elements of array **a** in a column-major manner (**a[i + k][j]**). By interchanging the loop order, you can access the elements in a row-major manner (**a[j][i + k]**), which can improve cache locality and potentially lead to better performance.

Here's how these optimizations are associated with the cache characteristics:

1. Cache Line Size: The cache line size is 64 bytes. By transposing the matrix in tiles of size tileSize, which is 1024, the code ensures that each tile fits within a few cache lines (1024 bytes / 64 bytes = 16 cache lines). This improves cache utilization as the data accessed within each tile is likely to reside in the same cache lines, reducing cache misses.
2. Cache Size: The cache size is 2kB. The tile size of 1024 allows for efficient utilization of the cache. Each tile is smaller than the cache size, ensuring that a significant portion of the accessed data resides within the cache. This

reduces the number of cache misses and improves overall performance.

3. LRU Policy: The LRU policy determines which cache lines to evict when the cache is full. By using loop tiling, the code enhances spatial locality and increases the chances of reusing cache lines within each tile. This reduces cache evictions and improves cache hit rates, thereby benefiting from the LRU policy.
4. Associativity: The cache has an associativity of 4, meaning each cache set can hold up to 4 cache lines. With loop tiling, the code accesses data within each tile in a localized manner, increasing the chances of cache hits within each set. This reduces conflicts and improves cache utilization.
5.

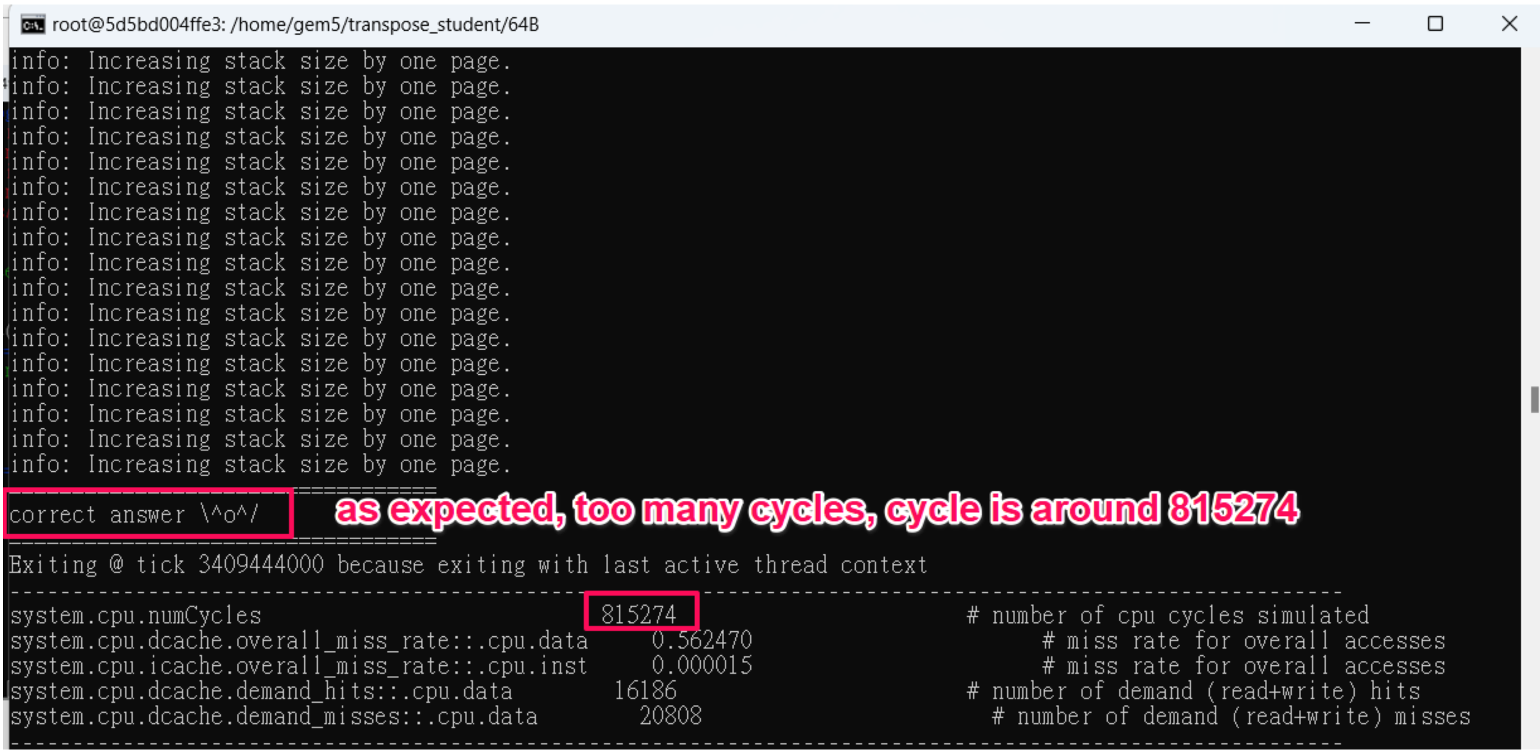
In summary, by employing loop tiling and prefetching techniques, the code optimizes cache utilization by exploiting spatial locality. It ensures that the accessed data within each tile fits within cache lines, minimizes cache misses, and maximizes cache hit rates. This leads to improved performance, especially when the cache has a cache line size of 64B, cache size of 2kB, LRU policy, and associativity of 4.

- Compare the result (cycles, miss rate of data cache) between the optimized and no-optimized codes

Simplest implementation in todo_64.cpp:

root@5d5bd004ffe3: /home/gem5/transpose_student/64B

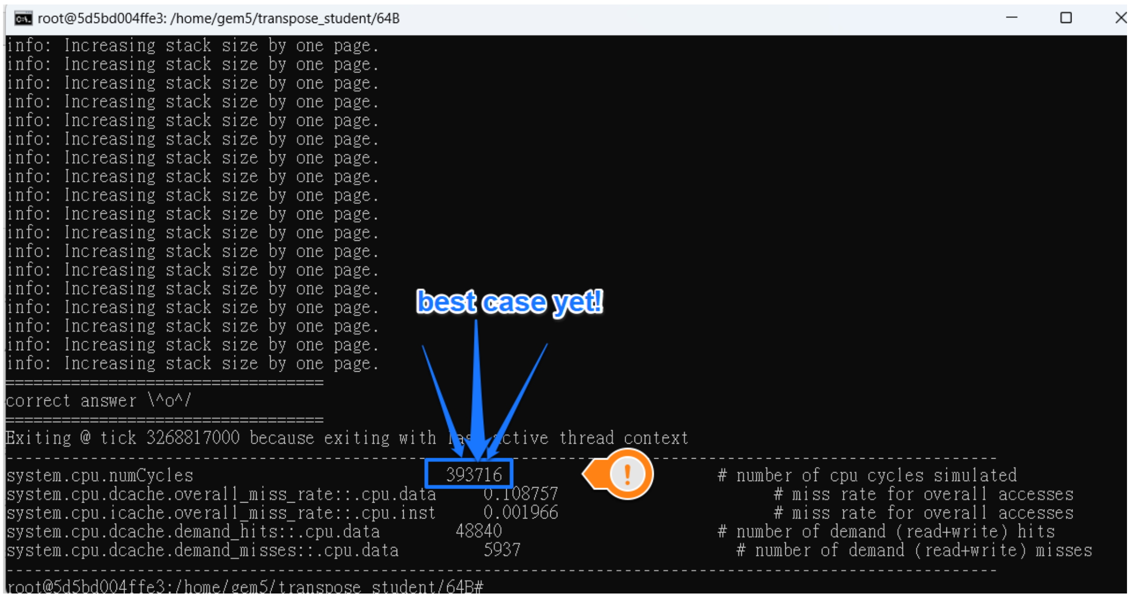
```
// Example program
#include <iostream>
#include <string>
#include <stdlib.h>
#include <stdint.h>
#include <bits/stdc++.h>
#define M 136

using namespace std;


void transpose(const uint64_t a[M][M], uint64_t b[M][M]){
//===================only modify in this region==================================
        for (int i = 0; i < M; i++) {
                for (int j = 0; j < M; j++) {
                        b[j][i] = a[i][j];
                }
        }
//==============================================================================
}
~
~
~
~
~
~
~
~
~
"todo_64.cpp" [dos] 20L, 501C                                    16,3-24       All
```



root@5d5bd004ffe3: /home/gem5/transpose_student/64B

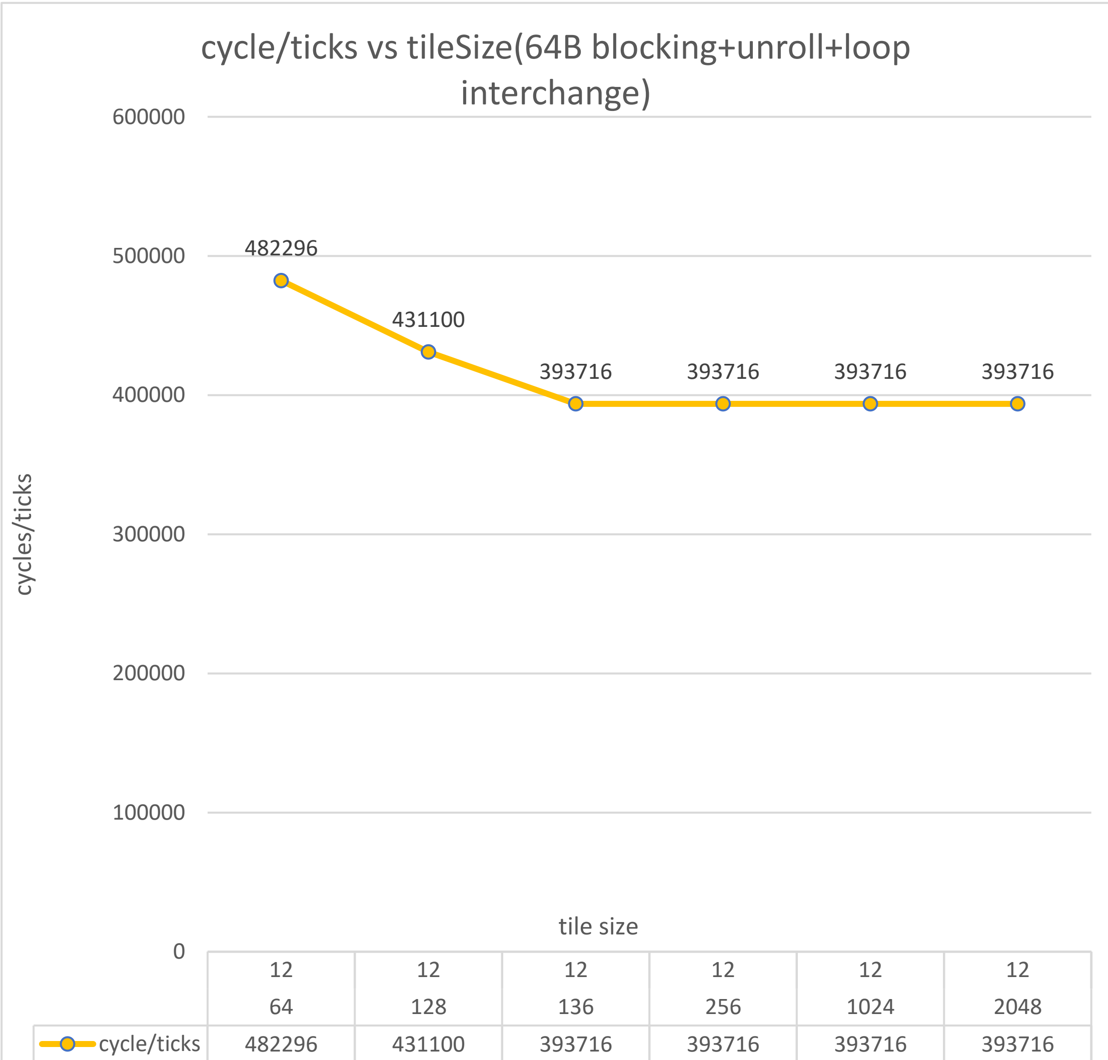
```
info: Increasing stack size by one page.
info: Increasing stack size by one page.
info: Increasing stack size by one page.
info: Increasing stack size by one page.
info: Increasing stack size by one page.
info: Increasing stack size by one page.
info: Increasing stack size by one page.
info: Increasing stack size by one page.
info: Increasing stack size by one page.
info: Increasing stack size by one page.
info: Increasing stack size by one page.
info: Increasing stack size by one page.
info: Increasing stack size by one page.
info: Increasing stack size by one page.
info: Increasing stack size by one page.
info: Increasing stack size by one page.
info: Increasing stack size by one page.
==================================
correct answer \^o^/
==================================
Exiting @ tick 3409444000 because exiting with last active thread context
-------------------------------------------------------------------------------------------------------
system.cpu.numCycles                             815274                    # number of cpu cycles simulated
system.cpu.dcache.overall_miss_rate::.cpu.data     0.562470                      # miss rate for overall accesses
system.cpu.icache.overall_miss_rate::.cpu.inst     0.000015                      # miss rate for overall accesses
system.cpu.dcache.demand_hits::.cpu.data        16186                      # number of demand (read+write) hits
system.cpu.dcache.demand_misses::.cpu.data        20808                      # number of demand (read+write) misses
-------------------------------------------------------------------------------------------------------
```

Blocking + Unrolling + loop interchange+ prefetch:



root@5d5bd004ffe3: /home/gem5/transpose_student/64B

```
info: Increasing stack size by one page.
info: Increasing stack size by one page.
info: Increasing stack size by one page.
info: Increasing stack size by one page.
info: Increasing stack size by one page.
info: Increasing stack size by one page.
info: Increasing stack size by one page.
info: Increasing stack size by one page.
info: Increasing stack size by one page.
info: Increasing stack size by one page.
info: Increasing stack size by one page.
info: Increasing stack size by one page.
info: Increasing stack size by one page.
info: Increasing stack size by one page.
info: Increasing stack size by one page.
info: Increasing stack size by one page.
info: Increasing stack size by one page.
info: Increasing stack size by one page.
==================================
correct answer \^o^/
==================================
Exiting @ tick 3268817000 because exiting with last active thread context
---------------------------------------------------------------------------------------------------------------
system.cpu.numCycles                          393716          # number of cpu cycles simulated
system.cpu.dcache.overall_miss_rate::.cpu.data   0.108757     # miss rate for overall accesses
system.cpu.icache.overall_miss_rate::.cpu.inst   0.001966     # miss rate for overall accesses
system.cpu.dcache.demand_hits::.cpu.data       48840          # number of demand (read+write) hits
system.cpu.dcache.demand_misses::.cpu.data      5937          # number of demand (read+write) misses
---------------------------------------------------------------------------------------------------------------
root@5d5bd004ffe3:/home/gem5/transpose_student/64B#
```

	Non-optimized	optimized
Cycles	815274	393716
dcache.overall_miss_rate::.cpu.data	0.562470	0.108757
icache.overall_miss_rate::.cpu.inst	0.000015	0.001966
dcache.demand_hits::.cpu.data	16186	48840
dcache.demand_misses::.cpu.data	20808	5937

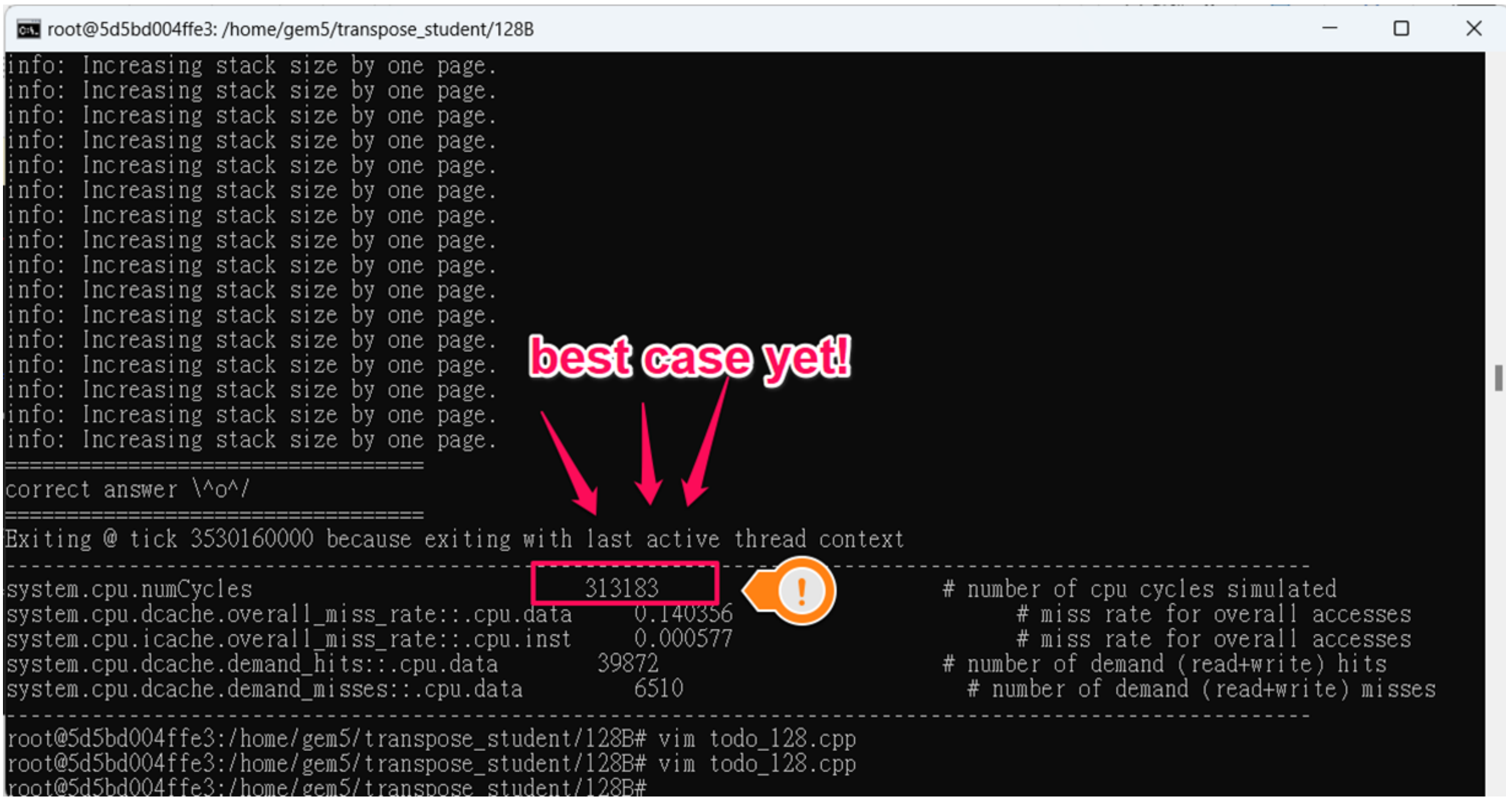
Simplest implementation in todo_128.cpp:

```
// Example program
#include <iostream>
#include <string>
#include <stdlib.h>
#include <stdint.h>
#include <bits/stdc++.h>
#define M 144

using namespace std;


void transpose(const uint64_t a[M][M], uint64_t b[M][M]){
//==================only modify in this region===============================
    for (int i = 0; i < M; i++) {
         for (int j = 0; j < M; j++) {
                b[j][i] = a[i][j];
         }
    }
//==========================================================================
}
~
~
~
~
~
~
~
~
"todo_128.cpp" [dos] 20L, 500C                                    16,21-35      All
```

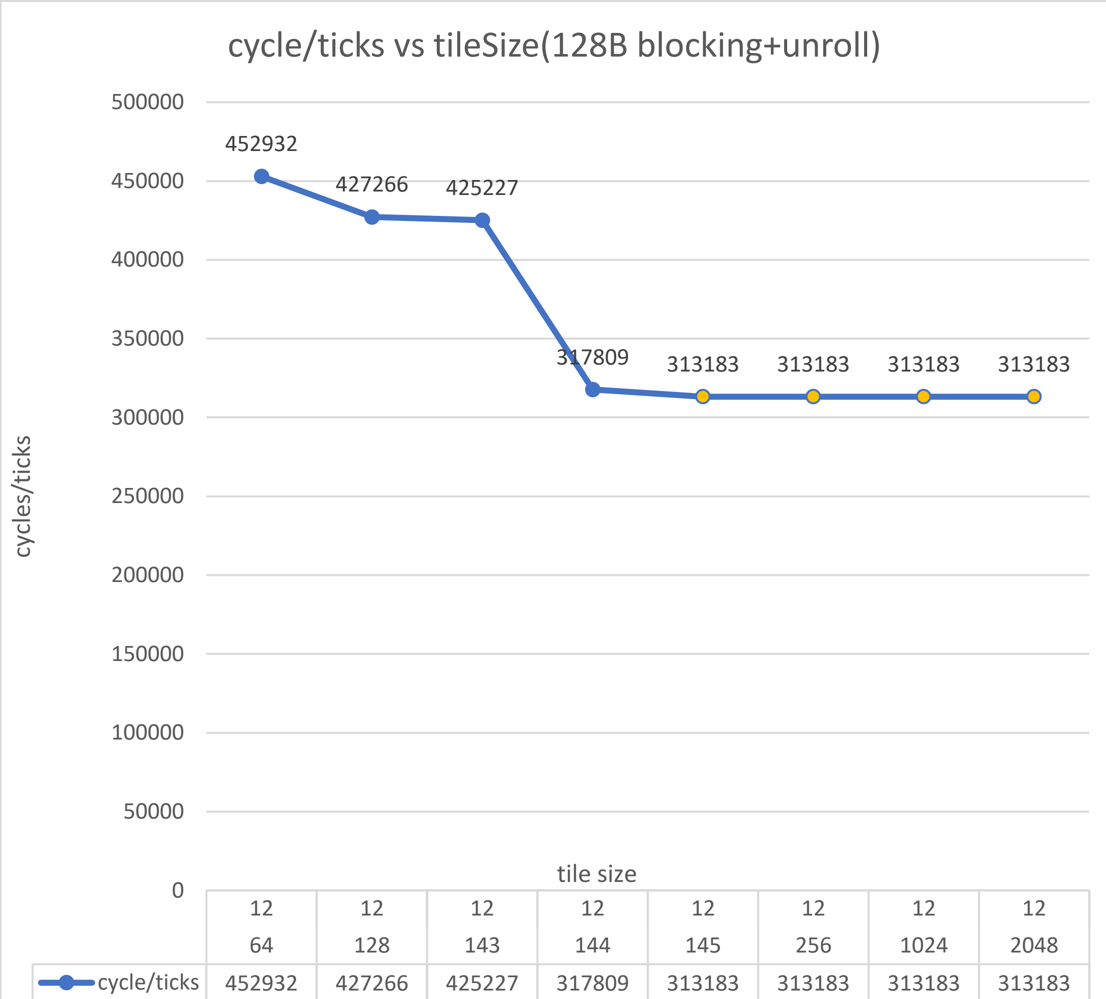
```
root@5d5bd004ffe3: /home/gem5/transpose_student/128B
info: Increasing stack size by one page.
info: Increasing stack size by one page.
info: Increasing stack size by one page.
info: Increasing stack size by one page.
info: Increasing stack size by one page.
info: Increasing stack size by one page.
info: Increasing stack size by one page.
info: Increasing stack size by one page.
info: Increasing stack size by one page.
info: Increasing stack size by one page.
info: Increasing stack size by one page.
info: Increasing stack size by one page.
info: Increasing stack size by one page.
info: Increasing stack size by one page.
info: Increasing stack size by one page.
info: Increasing stack size by one page.
info: Increasing stack size by one page.
======================================
correct answer \^o^/
======================================
Exiting @ tick 3726640000 because exiting with last active thread context
---------------------------------------------------------------------------------------
system.cpu.numCycles                              897502                  # number of cpu cycles simulated
system.cpu.dcache.overall_miss_rate::.cpu.data     0.531237                    # miss rate for overall accesses
system.cpu.icache.overall_miss_rate::.cpu.inst     0.000048                    # miss rate for overall accesses
system.cpu.dcache.demand_hits::.cpu.data          19441                   # number of demand (read+write) hits
system.cpu.dcache.demand_misses::.cpu.data          22032                   # number of demand (read+write) misses
---------------------------------------------------------------------------------------
```

as expected, too many cycles, cycle is around 897502

Blocking + unrolling



```
root@5d5bd004ffe3: /home/gem5/transpose_student/128B
info: Increasing stack size by one page.
info: Increasing stack size by one page.
info: Increasing stack size by one page.
info: Increasing stack size by one page.
info: Increasing stack size by one page.
info: Increasing stack size by one page.
info: Increasing stack size by one page.
info: Increasing stack size by one page.
info: Increasing stack size by one page.
info: Increasing stack size by one page.
info: Increasing stack size by one page.
info: Increasing stack size by one page.
info: Increasing stack size by one page.
info: Increasing stack size by one page.
info: Increasing stack size by one page.
info: Increasing stack size by one page.
==================================
correct answer \^o^/
==================================
Exiting @ tick 3530160000 because exiting with last active thread context
---------------------------------------------------------------------------------------------------------
system.cpu.numCycles                          313183                    # number of cpu cycles simulated
system.cpu.dcache.overall_miss_rate::.cpu.data     0.140356                   # miss rate for overall accesses
system.cpu.icache.overall_miss_rate::.cpu.inst     0.000577                   # miss rate for overall accesses
system.cpu.dcache.demand_hits::.cpu.data        39872                   # number of demand (read+write) hits
system.cpu.dcache.demand_misses::.cpu.data         6510                   # number of demand (read+write) misses
---------------------------------------------------------------------------------------------------------
root@5d5bd004ffe3:/home/gem5/transpose_student/128B# vim todo_128.cpp
root@5d5bd004ffe3:/home/gem5/transpose_student/128B# vim todo_128.cpp
root@5d5bd004ffe3:/home/gem5/transpose_student/128B#
```



	Non-optimized	optimized
Cycles	897502	313183
dcache.overall_miss_rate::.cpu.data	0.531237	0.140336
icache.overall_miss_rate::.cpu.inst	0.000048	0.000577
dcache.demand_hits::.cpu.data	19441	39872
dcache.demand_misses::.cpu.data	22032	6510