University of Vienna

Faculty of computer science

Computer science master's degree

Focus en data science

Course : parallel architectures and programming models

Report of the first assignment

By :

Mohamed Ait Amara

Student ID : 12204436

Winter semester: 2022 / 2023

Chapter I : Introduction

1. Presenting the problem

1.1 Explanation of the naive solution

The naive approach is literally scanning the rows of the matrix sequentially and transposing them. This approach triggers two problems:

Spatial locality: because the algorithm scans the data elements within relatively close storage locations. We can say that we are dealing with sequential locality with each row because the elements of each row are arranged and accessed linearly, similarly to traversing elements of a one-dimensional array.

Read / Write misses: Because the matrix size is much bigger than the cache size and each time the elements of the matrix are transferred to the cache, read and write misses occur. Also, the loops are executed n*n times and no special care is made to use the cache efficiently. This problem is shown is the following simulation with 4x4 matrix:

Matrix A:

A[0][0]	A[0][1]	A[0][2]	A[0][3]
A[1][0]	A[1][1]	A[1][2]	A[1][3]
A[2][0]	A[2][1]	A[2][2]	A[2][3]
A[3][0]	A[3][1]	A[3][1]	A[3][2]

A fully-associative cache, with 4 lines, each line is capable of storing 2 integers, and the replacement policy is the LRU:
4 misses:

Tag &A[0][0]	A[0][0]	A[0][1]
Tag &A[0][2]	A[0][2]	A[0][3]
Tag &A[1][0]	A[1][0]	A[1][1]
Tag &A[1][2]	A[1][2]	A[1][3]

We will get as many misses as the number of lines.

1.2 Runtime on Alma

Runtime = 25.440754 seconds.

Chapter II : Presentation of solutions

1. Cache-Aware Out-of-Place Matrix Transposition

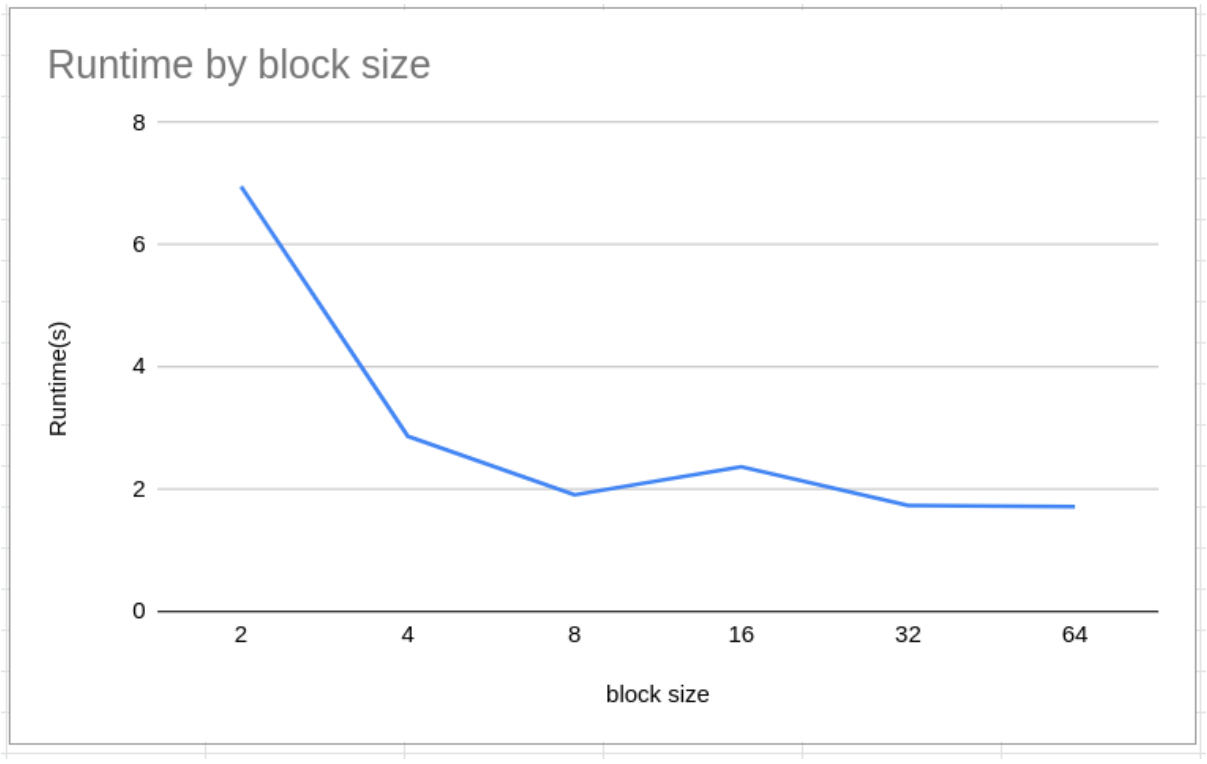
1.1 Explanation of the algorithm

In the cache aware transposition algorithm the matrix is effectively divided into a checkerboard of small blocks. Two blocks that are symmetrically distributed are identified and their data is copied into cache. The cache is then copied in the transposed matrix.

The block size must perfectly divide the matrix dimension n. In contrast to the naive algorithm, each element of the matrix is now loaded into the cache registers twice; once when copying the data from the source matrix, and once when copying each element into the destination matrix. “See source code”.

1.2 Runtime on Alma

This graph shows the runtime with respect to the block size:



Best speedup compared to the naive algorithm is : 14.81 times faster.

2 Cache-Oblivious Out-of-Place Matrix Transposition

1.1 Explanation of the algorithm

In this oblivious approach I used the z-order curve to map the two-dimensional matrix to one dimension while preserving locality of the data points. The advantage of laying out data in this way is that it improves data locality (and hence cache use) without having to tune a block size or similar parameter.

1.2 Runtime on Alma

Runtime = 7.139924 seconds.
Speedup compared to the naive algorithm is : more than 3 times faster.

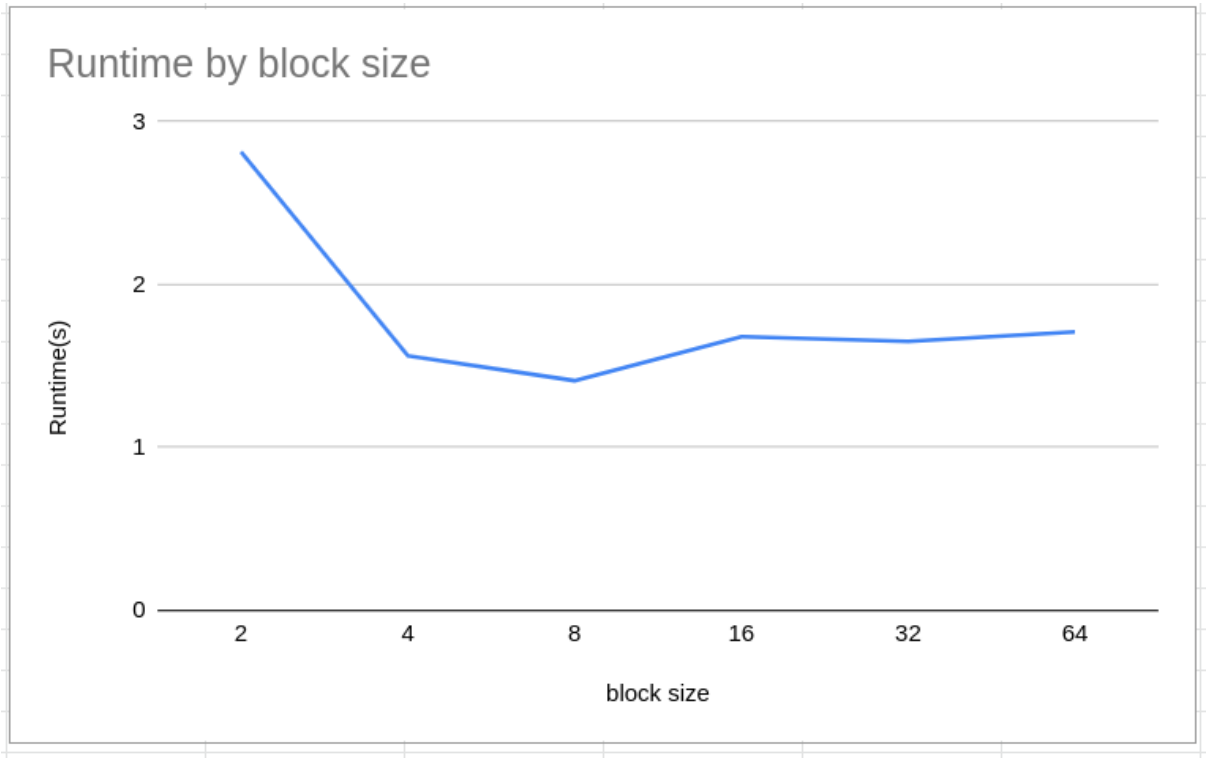
3 Optimized Out-of-Place Matrix Transposition

1.1 Explanation of the algorithm

In this approach I simply combined the two previous solutions in a way that I pass through blocks following a z-curve path. This approach benefits from the z-order and the transfer of elements to cache by blocks. Instead of scanning the data sequentially, I used one loop to follow the z-curve by a step representing the number of elements in each block and I transpose after that all the elements of the block.

1.2 Runtime

This graph shows the runtime with respect to the block size:



Best speedup compared to the naive algorithm is : 18.03 times faster.

Chapter III : Conclusion

1. Ensuring that the solutions are NUMA-aware

All my solutions are NUMA aware because they ensure memory coherence. This is an informal answer however, because to verify practically if my solutions are NUMA aware, I have to check instead if my operating system(Alma) structure is NUMA aware using predefined C functions.