# Parallel and Distributed Computing: Homework 5

**Objectives:** The main goal of this exercise is to experimentally measure the performance of the parallelization of a data transfer process using different versions of fine grain and coarse grain muti-threading.

Consider the problem of in-place[1] transposition of a given square matrix. The pseudo-code to perform this in a single processor is shown in the listing below:

**Algorithm 1:** Matrix Transpose

```
   Input: A;                    // Input Matrix
1  Function (A):
2  |  for 0 ≤ i ≤ n − 1 do
3  |  |  for i + 1 ≤ j ≤ n − 1 do
4  |  |  |  temp ← A_ij;
5  |  |  |  A_ij ← A_ji;
6  |  |  |  A_ji ← temp;
7  |  |  end
8  |  end
9  return
```

Note that this code scans the matrix by rows and exchanges symmetric elements with respect to the main diagonal focusing on individual elements of the upper triangular section of the matrix A, excluding the main diagonal.

One way of parallelizing the transposition of A in a muti-threaded multi-core environment is to share A among different threads, so all of them can access it, and do part of the job *concurrently*. Elements of A would be read by the threads following a coordinated indexing scheme to guarantee each element of A is accessed once and only once by only one thread.

Threads may have different *levels of coarseness*. On the one hand, they could be light or **fine grain**, doing little work each. For example, we could define the work-unit as "one single exchange"; then, each thread repetitively requests more of these small work-units until the transposition is finished.

On the other hand, threads can be made heavy, or **coarse grain**, by giving a big amount of work to each thread. For example, we could define the work-unit as "element exchanges of several rows"; once again, each thread repetitively requests more work-units until the transposition is finished, however, this time it takes longer for each thread to issue a new request because each work-unit takes longer in being finished. An extreme example would be defining the work-unit as "exchange all elements in the matrix". That would result in a single thread doing all the work.

Note that the level of coarseness of the work-unit is completely independent of the level of concurrency of the algorithm. For example, you could define the work-unit as having from one to a thousand exchanges, and independent of that, execute the algorithm using from one to a thousand threads.

Given this *coarseness* parameter, several questions arise:

- Is any coarseness/granularity level *"the best"*, and under what circumstances?
- How can *"best"* be even defined?
- What are the benefits and trade-offs towards each fine or coarse grain extremes?

[1] This is, using $O(1)$ extra memory

## Assignment

For simplicity, given that we are dealing with square matrices, a matrix of "size n" means an $n \times n$ matrix. Consider a square matrix $A$ of size $n$ stored in row major form, where $2 \leq n \leq 10,000$. Your programming task is to implement the following sequential and multi-threaded transposer functions.

`mat_squaretransp_sequential(mat)`

This sequential transposer function performs an in-place transposition, exactly as shown in the listing at the beginning. It receives one parameter: the matrix to be transposed `mat`.

`mat_squaretransp_parallel(mat, grain, thr)`

This multi-threaded transposer function also performs an in-place transposition. It receives three parameters:

- **Matrix to transpose** `mat`**:** Data structure with the matrix to be transposed. It is shared across all the threads performing the transposition.
- **Coarseness level** `grain`**:** The grain size that defines the unit of work. This will normally be a number of exchange operations.
- **Concurrency level** `thr`**:** Number of threads that will work together to transpose the matrix.

As for debugging purposes, you may change the content of any source code. However, your implemented solution must **not** depend on those changes, as the only file to be submitted is `transpose.c`. Only use standard libraries. Your implementation must strictly use the function signatures provided in `transpose.h`. Inside of `transpose.c` you can create as many static (internal) functions and data structures as needed.

The file `transpose.c` is used to build the program `granularity.bin`. This program will call your function with different parameters. It already handles and checks the correctness of the command line arguments that it receives. You can check details on how to pass arguments to it by running `./granularity.bin help`. Or run the demo code with something like `./granularity.bin 6 m 4 2`.

## Report

Please write a report discussing and giving answers to the following 5 questionss. It is advised to use a different page for each answer. Please be concise but complete. **The use of Chat GPT is strongly discouraged**.

1. A characterization of the sequential transposer execution time as the size of the matrix increases. Add plots and figures as necessary to explain your results.
2. A characterization of the multi threaded transposer execution time for a **fine-grain** configuration. Keep the number of exchanges per thread constant, `grain=1`. Keep the size of the matrix constant, `n=7,000`. Vary the number of threads in $\{1, 2, 3, 4, 5, 6, 8, 16, 24, 32, 64, 128\}$. Add plots and figures as necessary to explain your results.
3. A characterization of the multi threaded transposer execution time for a **coarse-grain** configuration. Experiment with different values for the parameter `grain`, particularly `grain=n` and `grain=(n×n)/threads`.

Keep the size of the matrix constant, `n=7,000`. Vary the number of threads in $\{1, 2, 3, 4, 5, 6, 8, 16, 24, 32, 64, 128\}$. Add plots and figures as necessary to explain your results.

4. Fixing `n=7,000`, `grain=1` for fine-grain, and `grain=n` for coarse-grain; graph the speedup obtained by using the fine and coarse grain multi-thread version with respect to the sequential one, as the number of threads used in the multi-threaded version varies in $\{1, 2, 3, 4, 6, 8, 16, 32, 64, 128\}$. Explain your results.

5. Consider matrices of size n from 10 to 10,000. Graph the speedup obtained by the fastest coarse-grain transposer with respect to the sequential version. Explain your results.

## Submission

Submit **one** zip file named `hw5.zip`, containing **exactly 3 files**. Make sure the zip file does **not** include sub-directories (as Mac's default compression tool does) or extra files:

1. `transpose.c`: The edited C source template with your well commented implementation of the single and multi threaded transposer. **Do not submit any other source code file**.

2. `report.pdf`: The digitally produced report of **at most** 5 pages.

3. `team.txt`: Text file with the UCINetID and name of each student in your team. Each line will have this format:
   `<UCINetID>, <firstName>, <lastName>`

Remember to sign-up with your team in a Canvas-Group. Only one student in the team needs to submit the work. Late submissions will not be accepted.