Assignment 2: Report

Name: **Lukas Hinterleitner**

ID: **51912219**

Based on your solution and understanding of the assignment and lecture slides, fill out the questions below:

1. Identify opportunities for parallelism in the sequential version of the code by stating which loops can be parallelized using OpenMP. Consider all loops of the sequential code and explain why these loops can be parallelized, and how did you proceed to parallelize them with OpenMP.

The main opportunities for parallelism in the sequential code are the loops that iterate over the matrix *U* und *W*. These loops are independent, meaning that each iteration of the loop does not depend on the results of the other iterations. This is especially true for the initialization loops. For the calculation loops, this is also true since data is only written to the W matrix in the first for-loop. Hence, the *U* matrix where data will be read from different locations is not touched. The second for-loop that can be parallelized only transfers data from *W* to *U* and is independent from the previous loop and hence, does not interfere with it. Summarized, I parallelized all outer-for-loops and OpenMP. For the loop where the *diffnorm* is calculated, I used the term *reduction(+:diffnorm)* to synchronize the variable *diffnorm* with the threads since it is the stopping criterion for the do-while-loop.

1. Was there any loop in your OpenMP code that you could not parallelize? Explain why you could not parallelize it via data dependency analysis.

Yes, the do-while-loop. This loop was not parallelizeable because then the U and W operations would not be independent of each other anymore which leads to data races etc.

1. In your OpenMP code, you were supposed to implicitly define the data-scope of all variables that are used within a parallel region. Justify your data-scoping choices for each variable within the extent of a parallel region.

Basically, the variables, *int I,j;* are defined in the outmost scope of the main method. When we leave the code like that, we have to tell OpenMP to use this variables in private for each thread, e.g. append to the for-loop pragmas the clause *private(i, j).* But since we need to implicitly define the data scope of the variables, I moved the variable declaration of *int I,j;* into their corresponding for-loops. The *diffnorm* variable has to be defined outside of their corresponding loops since it is our stopping criterion that needs to be synchronized.

1. Was there any variable in your parallel code that you had to protect with OpenMP synchronization constructs? If yes, identify the variable, and explain how you protected the accesses to this variable.

Yes, the *diffnorm* variable. I appended the OpenMP *reduction(+:diffnorm)* to the corresponding for-loop pragma.

1. In your code you most likely used OpenMP for construct to parallelize the different loops of a2.cpp. In the context of this problem, which scheduling strategy would you use to distribute the iterations of the different loops? Justify your scheduling choices. You should resort to the plots of 7. to verify your hypothesis.

1. Consider the two loops inside the while-do loop of a2.cpp. Assuming you parallelized both of these for-loops with OpenMP, discuss whether the nowait clause could be applied to the first for-loop.
2. Include a speedup plot of your OpenMP solution, with static and dynamic (default chunksize) iteration distribution on ALMA (for 1, 2, 4, 8, 16 and 32 threads). Remember: Use the horizontal axis to represent the number of cores, while the vertical is used to represent the speedup (not execution time).
3. Different matrix sizes result in different performance in both OpenMP and MPI versions. Try to explain this behavior. Consider memory hierarchy and communication/synchronization within and across (MPI) the nodes.
4. Explain differences between OpenMP and MPI code. Which code is suitable for which type of architecture? How about the code complexity? What are the common problems that you need to deal with in these approaches?
5. Measure the overall execution time of your MPI program, and measure the time required to collect data on rank 0 for verification (discuss differences for different input arguments). Explain how you measured execution time in the MPI code, and which values you used to print the elapsed time on rank 0. Why these values correctly represent the execution time of your program? Is the time required to collect data different when the program is executed on a single node?
6. How is the data distributed among the MPI processes? How big is each matrix on each process with respect to the M and N. How did you handle the case where (M%numprocs != 0)?
7. What data needs to be communicated between MPI processes and at which points in your code? Which MPI routines have you used to accomplish this?
8. Are there any points in the code where you need to be careful not to introduce a deadlock? If so, where?
9. Briefly explain how you collected data to rank 0 at the end of your code for verification on rank 0. Which routines have you used and how? What data was relevant, and on which ranks?
10. Include a table with execution times and speedup of the OpenMP and MPI codes for different configurations given in the slides. Speedup needs to be measured with respect to the sequential version (around 50, 112, and 5 seconds for different versions on ALMA nodes – see slides). If you have these in an Excel sheet, you can just attach it with your submission. Plot OpenMP and MPI single node configurations on a separate speedup graph for comparison and discuss the performance differences.

Below, you can add any additional content and analysis that does not fit into the above categories: