Chair for Computational Analysis of Technical Systems | RWTH AACHEN UNIVERSITY

Parallel Computing for Computational Mechanics

Summer semester 2022

Final Project

Parallelization of a FE code using OpenMP and MPI

The final project described in this document will cover the main concepts discussed during the course *Parallel Computing For Computational Mechanics*. For this project the FE solver from the homework tasks will be used to solve the temperature distribution on a two-dimensional disk. The solver will be compiled and executed serially as well as in its parallelized version using OpenMP and MPI. Afterwards the performance of the individual implementations will be documented and analyzed.
The temperature problem solved on the circular domain is presented in Section 1. The procedure to compile the provided skeleton code is discussed in Section 2. The specific simulations and corresponding data to be documented is given in Section 3. Finally, a short report needs to be written in which your results are presented and discussed. More information on the requirements of the final report is given in Section 4.

1 Problem Description

The temperature is obtained by solving the heat equation on a disk Ω with boundary Γ, as shown in Figure 1. The heat equation for this project is defined as follows:

$$\frac{\partial T}{\partial t} - \kappa \boldsymbol{\nabla} T = f \qquad \text{on } \Omega \quad \forall\, t \in (0, t_f), \tag{1a}$$

$$T(\boldsymbol{x}, t) = T_D \qquad \text{on } \Gamma \quad \forall\, t \in (0, t_f), \tag{1b}$$

$$T(\boldsymbol{x}, 0) = T_0 \qquad \text{on } \Omega. \tag{1c}$$

where T is the temperature, κ the thermal diffusivity, and f the thermal heat source. At the domain boundary Γ a Dirichlet condition is imposed with temperature T_0. The problem is solved for time $t \in (0, t_f)$ using initial condition T_i and final solution time t_f. The heat source changes based on the location of the node:

$$f(r) = \begin{cases} \frac{Q}{\pi R_1^2} & \text{if } r < R_1, \\ 0 & \text{if } r \geq R_1. \end{cases} \tag{2}$$

$Q = P * d_z$ is the volumetric heat source. $d_z = 0.1\mathrm{m}$ is the out-of-plane thickness. The given problem comes with an analytic solution for $t \to \infty$ given by:

$$T(r) = \begin{cases} T_0 - \frac{Q}{2\pi\alpha}\left(\frac{1}{2}\left(\frac{r^2}{R_1^2} - 1\right) + \ln\left(\frac{R_1}{R_2}\right)\right) & \text{if } r < R_1, \\ T_0 - \frac{Q}{2\pi\alpha}\ln\left(\frac{r}{R_2}\right) & \text{if } r \geq R_1. \end{cases} \tag{3}$$

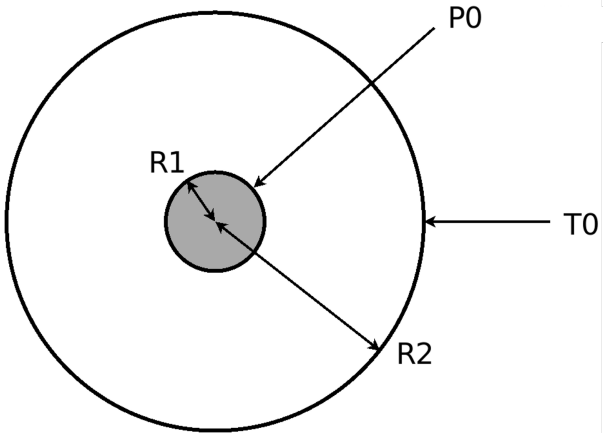

Figure 1: Problem domain.

where, α and β are user specified parameters. α and β, as well as the other parameters used in this project are depicted in Table 1.

Table 1: Problem parameters for the heated disk problem.

Parameter	Variable	Value	Unit
Inner circle radius	R_1	0.01	[m]
Outer circle radius	R_2	0.1	[m]
Heat source on the area	Q	100	[W]
Dirichlet BC temperature	$T_0(x, y)$	500	[K]
Initial temperature	$T_s(x, y, 0)$	0	[K]
Thermal diffusivity	κ	1.0	[m^2/s]

2 Compiling the FE solver

The source code of the FE solver is located in the `skeleton/` directory. The FE solver can be compiled for execution in serial and parallel using either OpenMP or MPI. The compilation process is done by executing the following commands within the `skeleton` directory:

```
mkdir build                              # Create new directory to compile code
cd build                                 # navigate into build directory
cmake .. -DBUILD_SELECTOR=<build>        # Execute CMake to generate the make setup
make                                     # compile the actual code using make
```

By replacing the placeholder `<build>`, by one of the options below the different builds can be compiled. After the make process is completed, an executable of the solver with the name `2D_Unsteady_<build>` is placed inside the build folder.

```
cmake .. -DBUILD_SELECTOR=Serial          # Compile in Serial
cmake .. -DBUILD_SELECTOR=OpenMP_Task_A # Compile in parallel with OpenMP for task A
cmake .. -DBUILD_SELECTOR=OpenMP_Task_B # Compile in parallel with OpenMP for task B
```

```
cmake .. -DBUILD_SELECTOR=MPI          # Compile in parallel with MPI
```

Hint 1. It is recommended to compile the different build in separate directories (`skeleton/build-serial/`, `skeleton/build-mpi/`, etc.), this way potential compilation errors are avoided.

Hint 2. In addition to the information given above, a `README.md` document with additional hints is located in the `skeleton` directory.

Hint 3. Before compiling, make sure you have the correct compiler module loaded in your shell. For this project you will need to use Intel 19.1 compiler. In your shell, execute the following command to switch to the correct module: `module switch Intel Intel/19.1`. To check which module is currently loaded you can use `module list`.

3 Project Tasks

This project can be divided into three main tasks which use the serial, OpenMP, and MPI solver respectively. Each task is described in more detail in the following section.

3.1 Optimization at Compile Time

Before running the finite element solver in parallel, it is important that the solver's serial performance is as efficient as possible. A low hanging fruit in making code run faster is to exploit the compiler's ability to improve code efficiency. In this task you will have to determine which compiler options result in the fastest serial solver. These options will then be applied in the remaining tasks.

1 a Compile the serial solver by using the `Serial` build selector and solve the model problem from Section 1 using the medium mesh. By default, the skeleton code compiled with compiler optimization deactivated (`-O0`). Document the timing of this non-optimized setup for later comparison.

1 b Investigate how various compiler options affect the solver's efficiency. For this, compile the serial solver using various compiler options and document the corresponding run times of the solver for the medium mesh. The compiler options can be set in the main `CMakeLists.txt` file located, in the `skeleton/` directory by modifying the `add_compile_options()` clause. For more information on CMake please refer to the documentation available online and the results of the class participation session on compiler optimization.

1 c Provide a table in your report which highlights the different compiler flags that you have tested and their impact on the run time.

Make sure to set the best combination of compiler options in the main `CMakeLists.txt` file for the remaining tasks. These options will then be applied automatically when compiling the parallel solvers.

3.2 Shared-Memory Parallelization using OpenMP

In this next task, you will be investigating the efficiency improvement of the solver that is parallelized using the OpenMP shared-memory interface. Within the skeleton directory, two solvers are provided. Both solvers make use of OpenMP, however, both are using a different approach to parallelize the computational intensive code sections. The different OpenMP implementations are located in the `skeleton/solver-openmp-a/` and `skeleton/solver-openmp-b/`. Using these implementations, you need to do the following steps:

2 a Compile both OpenMP solvers by using the `OpenMP_Task_A` or `OpenMP_Task_B` build selectors with CMake. Using both of the resulting executables, solve the model problem from Section 1 on the coarsest mesh and document the timings. By investigating the source code of both solvers, can you explain the difference in run time?

2 b For this next task, we will be working with solver `OpenMP_Task_B`. Within this solver there are several scheduling approaches already implemented but commented out. Uncomment the various scheduling methods and document the solver timings when solving the finest mesh problem. For some scheduling methods, the chunk size can be specified. Run the different scheduling methods for various chunk sizes and document the different timings.

2 c Using the most effective scheduling approach, run the solver on 1, 2, 4, 6, 8, and 12 threads, document the timings and report on the scaling and efficiency of the shared memory solver for the coarse, medium and fine mesh.

Hint 4. Make sure to use the compiler options obtained from the optimization task in Section 3.1.

3.3 Distributed Memory Parallelization using MPI

Complete the following tasks:

3 a In the directory `mpi-solver` an MPI parallelized version of the FEM heat equation solver is included. It uses the same MPI parallelization as was presented in Homework 4. Use **cmake** to compile the solver with the compiler flag(s) you have found within the tasks in subsection 3.1.

3 b Run the `mpi-solver` for the provided coarse, medium, and fine grids based on the provided settings files. For each of the grids, run the solver for 1,2,4,6,8,12 and 16 cores by adjusting the provided `run.j` script. Collect the timings provided from the log data.

3 c Use the timings collected and plot the runtime over the different numbers of cores as well as the speed-up and efficiency for the 3 different grids.

3 d Look at the different `.vtu` files that are created for each core. Plot the partitions of the medium mesh with 8 cores using Paraview. Load all 8 numbered `.vtu` files into Paraview, for each one select Solid Color and make them distinguishable by applying different colors to each partition.

4 Project Report

4.1 Report Requirements

Write a precise, coherent report. Refer to the report instructions provided in the exercise. You can keep the *Theory and Methods*, and *Implementation and Validation* sections short. The report should include all the tasks specified in this assignment. In the following subsections, we will give a wrap-up and more details about the tasks described above and what you are supposed to put in to the report in the *Results* and *Discussion* sections. Do not forget to include a *Conclusion*.

4.2 Serial Solver

Include the answers to the following tasks in your report:

R 1 a Provide a plot of the simulation result on the coarse mesh in comparison to the analytical solution. Provide a table with timings for the various compiler flags you have tried. Describe what each flag does. State the best combination of flags that you have found.

R 1 b Can you think of a case where you would prefer the non-optimized code over the optimized one? What are potential drawbacks of using optimization flags?

4.3 Parallel OpenMP Solver

Include the answers to the following tasks in your report:

R 2 a Document the timings of the solvers with the `OpenMP_Task_A` and `OpenMP_Task_B` options on the coarsest mesh for 1,2 and 4 threads. Did you expect those results and why do they differ?

R 2 b Document the timings when using the solver with the `OpenMP_Task_B` option with different types of scheduling and chunk sizes. Use the finest mesh and a constant number of threads of 4. How do the different scheduling types compare? Which chunk size works best for which scheduling?

R 2 c Document the timings when using the best performing solver from above with respect to scheduling and chunk size. Document the timings for 1,2,4,6,8,10 and 12 threads for the coarse, medium and fine mesh. From the timings obtained, calculate the speed-up and parallel efficiency. Provide the timings in a table and the speed-up and efficiencies in plots. Discuss the results.

R 2 d What are the default settings for scheduling in OpenMP? What are the default chunk sizes for `static`, `dynamic` and `guided`?

4.4 Parallel MPI Solver

Based on your findings from Section 3.3, complete the following tasks and include the results and discussion in your report:

R3 a Provide the timing, speed-up and parallel efficiency plots for all 3 meshes. For the efficiency plot, include the optimum efficiency that a parallelized code could achieve. Describe the plots in the text.

Include a discussion of the results. For the discussion, think about what you expected from a parallel code. What are expected values for speed-up and efficiency? What are the differences between the meshes? What are maximum/minimum values of the speed-up and efficiency? Give your opinion on which number of cores to use for which mesh and why.

R3 b Provide a image of the partitions on the medium mesh, when using 8 cores. Comment on whether this is a good partitioning or not. What effect does this have on the code?

R3 c Discuss the performance of the code. Do you see room for improvement? If so, how would you improve the code? Make suggestions to the developers.

R3 d Answer the following MPI related questions in written form: I) Why do we use `MPI_Accumulate` in the code instead of `MPI_Put`? II) What are advantages of using one-sided MPI communication over two-sided MPI communication? III) What advantage does MPI offer over OpenMP, what is a main obstacle of MPI?

The hard deadline for the submission of this report is **Sunday, 24.07.2022 at 23:59**. We will not accept late submissions. If you have questions, use the Moodle Discussion Forum and contact us during the office hours and the Q&A session in the exercise sessions.

Contact:
Maximilian Schuster · schuster@cats.rwth-aachen.de
Michel Make · make@cats.rwth-aachen.de