Technical University of Munich
Chair of Aerodynamics and Fluid Mechanics
Chair of Scientific Computing in Computer Science
M. Wille, M.Sc.
S. Narvaez Rivas, M.Sc.
A. Bußmann, M.Sc.
F. Fritz, M.Sc.
A. Toshev, M.Sc.

WS 2022/23
Worksheet 2
Nov 22 2022

Turbulent Flow Simulation on HPC-Systems

Worksheet 2: Parallelization and Turbulence
Deadline: Dec 14, 2022 (11:59pm)

1 Getting Started as a Team

You will work on worksheet 2 as a team. This means that you will also have to work in one single repository. Please decide as a team which one of the already existing repositories from worksheet 1 you want to continue using as a team. Make this decision early, quick and live it! Don't overthink it. It is very important to work on a single version as a team, since working on different version will just waste a lot of precious development time.

As soon as you have decided which repository to continue working with: Invite all team members into this repository as developers. Please make sure that your fork is a *private* repository and invite **Mario and Santiago**[1] as *reporters to the repository*[2]. Don't forget to also invite all team members as developers to the common respository. Every team member should make sure that the correct repository is cloned [3].

If the merge request from worksheet 1 is not merged yet, please do it now. This will make your `main` branch your new reference implementation. You can also discuss the merge request with your team and try to fix minor (!) issues directly on the `main` branch.

Now the workflow is the same as for worksheet 1: Create a branch `ws2` and start working on it. You may also create additional branches, if you want. But make sure to not merge into `main`, but into `ws2`. You can also use merge requests, if you want. But: Merging `ws2` into `main` will, again, be done via a merge request (see section 8).

2 Parallelization

In order to approach large-scale simulations, the program NS-EOF is enhanced by distributed memory parallelization using the *Message Passing Interface* (MPI). You may assume that we have a domain decomposition into $P := P_x \times P_y \times P_z$ processes along the different coordinate axes. The PETSc-based pressure solver is already parallelized internally. Still, velocity and

[1] Usernames: mariowille, santiagonar1
[2] `https://docs.gitlab.com/ee/development/permissions.html`
[3] Check this with `git remote -v` from inside your repository. There you should see the URL of the common repository.

pressure values need to be updated at the boundaries between neighboring processes once per time step – this represents the task of this section. Besides, the maximum time-step size which is allowed during the simulation needs to be determined over all processes. The latter is already implemented.

2.1 Communication of Flow Quantities

In this exercise, the exchange of pressure and velocity values is established between the processes, again making use of the stencil-iterator concept.

1. Check out the implementation of the iterator `ParallelBoundaryIterator`. Read the implementation and infer its functional meaning: which cells does the iterator operate on? Which functions does a respective stencil object need to provide? Can you reuse existing stencil formats, or are new stencil types required?
2. Implement a boundary stencil `PressureBufferFillStencil` which reads the pressure values in each of the six (3D) boundary faces of a sub-domain (i.e. the domain of one process) and stores them consecutively in one-dimensional buffer arrays (one array for each of the six faces). For traversal of the respectivesub-domain cells, cf. 1.
3. Implement a boundary stencil `PressureBufferReadStencil` which reads data from one-dimensional arrays (one array for each of the six faces) and writes them into the correct cells of the boundary.
4. Integrate the read- and write operations into a class `PetscParallelManager`. The `PetscParallelManager` should provide a method `communicatePressure()` which
   a) fills pressure values from the domain into buffers using the `PressureBufferFillStencil`,
   b) communicates the pressure values between neighbored processes. You may use the function `MPI_Sendrecv(...)` for this purpose,
   c) reads pressure values from the buffers of the preceding communication step and writes them to the grid cells using the `PressureBufferReadStencil`.
5. Implement the exchange of velocities analogously to the steps 1-4. Implement write- and read-stencils `VelocityBufferFillStencil` and `VelocityBufferReadStencil` and integrate them into the `PetscParallelManager`. The respective method `communicateVelocities()` should further handle the exchange of the velocity buffers between the processes.
6. Integrate the `PetscParallelManager` and respective calls to its communication methods into the class `Simulation`. Test the exchange of flow quantities in cavity, channel and backward-facing step flows. It is sufficient to only consider the first time step (the choice of a maximum time-step size should not have severe impact on the testing).

2.2 Global Synchronization of the Time-step Size

Check out the implementation of `setTimeStep()` in the class `Simulation`. How is the maximum time-step size evaluated in the parallel simulation?

2.3 Validation

Validate your parallel implementation for different domain decompositions, that is different choices $P_x \times P_y \times P_z$, and domain sizes. Use cavity, channel and backward-facing step simulations for this purpose. Compare the solutions to your sequential program. What do you observe?

3 Scaling and Efficiency

In this exercise, the quality of the parallel implementation NS-EOF is investigated. The platform is given by the Linux-cluster[4].

3.1 Theory: Towards Scaling Experiments

An important property of parallel software is scaling behavior:

- How much faster is my code for a given problem (i.e. fixed domain resolution) when using more processes (strong scaling)?
  A perfect strong scaling implies halving the time-to-solution when using twice as many processes.
- How long does it take to solve a problem with $N_x \times N_y \times N_z$ cells on each process compared to the sequential solution of a problem with $N_x \times N_y \times N_z$ cells (weak scaling)?
  A perfect weak scaling means that the execution time of the program stays the same when going from P to $2P$ processes.

How can weak and strong scaling be measured for NS-EOF? Which problems do you expect?

3.2 NS-EOF on the Linux-cluster

In the following, the steps to execute and compile the program on the cluster are briefly described. For a more detailed description of the cluster, available libraries/modules etc., check out the webpages given in the footnotes.

3.2.1 Environment and Login

The Linux-cluster is a heterogeneous environment in the sense that it comprises various types of processors on different systems. Specifically we will work on the CoolMUC-2 system. You can login using ssh on the login nodes 1, 2, 3 or 4. The full ssh command is

```
$ ssh <YOUR LOGIN>@lxlogin<#>.lrz.de
```

using your login at the LRZ and your password to login. Replace `<YOUR LOGIN>` with your login and `<#>` with the number of the login node. For example,

```
$ ssh xy42abc@lxlogin4.lrz.de
```

would login user `xy42abc` via login node 4.

[4] http://www.lrz.de/services/compute/linux-cluster/intro/

3.2.2 Compiling

PETSc and CMake installations are available on the cluster. In order to use them you need to load the respective modules [5]. Hence, before compiling the code, please execute the following commands:

```
module load cmake
module load petsc
```

Now, you can just proceed to compile the code as you did in worksheet 1.

3.2.3 Execution

There are basically two methods to run your parallel program on the Linux-cluster: interactively or via batch jobs.

Interactive jobs: you can use the command `salloc` to allocate resources for your parallel execution:

```
salloc --ntasks=SOME_NUMBER --time=HH:MM:SS
```

Example:

```
salloc --ntasks=8 --time=01:00:00
```

allocates eight processors for one hour.
Using `salloc` opens a new shell and gives you exclusive access to the respective resources and you can immediately run your program in the usual way using `mpirun -np ...`. Make sure to free the resources immediately at the end of your session so that other cluster users obtain access to them (just close your session typing Ctrl+D)! The maximal runtime for `salloc` is two hours.

Important: in the interactive job you will need to execute

```
module load cmake
module load petsc
module load gcc/10 # Additional requirement
```

Batch jobs: write a batch job file and submit it to the queuing system of the Linux-cluster. Detailed information on the respective job files is given on the login screen after logging in to the cluster. An exemplary job (let's call it `myjob.job`) file is given in the following:

```
#!/bin/bash
#SBATCH -o /home/hpc/pr63so/MYLOGIN/MYJOB.out #job file
#SBATCH -D /home/hpc/pr63so/MYLOGIN/ #starting directory
#SBATCH -J MYJOB #name shown in squeue
#SBATCH --get-user-env #load user environment settings
#SBATCH --clusters=mpp2
#SBATCH --ntasks=512
#SBATCH --mail-type=end
```

[5] Please see the official documentation of the cluster to learn more about environment modules. For instance, you can use `module list` to see all currently loaded modules.

```
#SBATCH --mail-user=MYEMAIL@MYEMAIL.COM
#SBATCH --export=NONE
#SBATCH --time=02:00:00 #runtime

module load cmake
module load petsc
module load gcc/10

./executable input1 input2 ...
```

From the directory which contains the job file, submit this job by typing

sbatch myjob.job.

You can check the status of your job by typing `squeue`. This will show you the current queue of the cluster.

3.3 Scaling Measurements

Perform weak and strong scaling measurements for your program NS-EOF.

1. Choose **suitable** simulation scenarios for the scaling measurements with respect to domain size, domain decomposition and the flow problem under consideration.
2. Run your parallel simulations for various numbers of processors. (Optional: Run on different architectures.) Plot your speedup and parallel efficiency in respective graphs (processes vs. speedup, e.g.).
   Hint: Deactivate VTK output in the scaling experiments since our output is not optimized for parallel execution yet.
3. How can you improve the performance of your program?
4. Optimize your software. You may work on both sequential and parallel improvements as well as on algorithmic optimizations (such as optimization of the pressure Poisson solver).

4 Turbulence Modeling

4.1 Information

Up until now the focus of our course has been put on obtaining a code that has the necessary capabilities of performing simulations of incompressible flows by means of direct numerical simulation. The RANS simulation of turbulent flows involves a large range of scales that need to be resolved. Furthermore, a modeling approach to close the RANS equations is required that has been not considered so far.

To close the unknown terms in the RANS equations, i.e. the Reynolds stress tensor, one modelling approach is to introduce a turbulent eddy viscosity ν_t that mimics the exchange of momentum exchange (see lecture notes):

$$\langle u_i' u_j' \rangle - \frac{2}{3} K \delta_{ij} = -\nu_T 2 S_{ij} \tag{1}$$

Equation (1) accounts for the fact that only the anisotropic part of the RST leads to turbulent kinetic energy (TKE) production. The isotropic part $\frac{2}{3}K\delta_{ij}$ of the RST is accounted for by the Poisson solver; it does not need to be considered any further.

The momentum equations, including an eddy viscosity, may be written as

$$\frac{\partial \langle u_i \rangle}{\partial t} + \langle u_j \rangle \frac{\partial \langle u_i \rangle}{\partial x_j} = -\frac{\partial P}{\partial x_i} + \frac{\partial}{\partial x_j} \left(2(\nu + \nu_T) S_{ij} \right) \tag{2}$$

with $P = \langle p \rangle + \frac{2}{3}K$.

4.2 Algebraic turbulence model

The most simple choice to model the turbulent viscosity has been proposed by Ludwig Prandtl, the mixing length model. It states that the turbulent viscosity is proportional to a mixing length scale l_m and the shear rate tensor S_{ij}

$$\nu_T = l_m^2 \sqrt{(2 S_{ij} S_{ij})} \, . \tag{3}$$

Prandtl proposed to model the mixing length l_m as

$$l_m = min \left(\kappa h; 0.09 \delta \right) \tag{4}$$

with δ denoting the local boundary layer thickness and $\kappa = 0.41$. In equation (4), h is the local distance of the current cell to the nearest wall.

To compute the boundary layer δ different options exist:

- non-existing boundary layers:
$$\delta = 0, \tag{5}$$
- laminar flat plate (Blasius theory):
$$\delta \approx 4.91 x / \sqrt{\mathrm{Re}_x}, \tag{6}$$
- turbulent flat plate:
$$\delta \approx 0.382 x / \mathrm{Re}_x^{1/5}. \tag{7}$$

5 Implementation (30%)

5.1 Input

1. Extend the configuration file of the simulation such that all parameters required by the turbulence model implementation can also be read from the XML-files.

5.2 Fields

1. Both, the distance to the (nearest) wall h and the turbulent viscosity ν_t require an additional scalar field. Both values are stored at the **cell center**, i.e. at the same location as the pressure.
2. Consider the basic interplay of stencils, iterators and data structures in NS-EOF. How can you apply the concept of inheritance so that you can naturally extend your data structures and support both old and potentially new data structures in case of turbulence-model-based simulations?
3. Implement a VTK visualization for the turbulent viscosity. You may base your implementations on the existing plotter (VTKStencil) for the turbulent case and create a new stencil for this purpose which operates on the new data structure.

5.3 Momentum equations

Due to turbulence modeling, viscosity is not homogeneous anymore, but varies in space and time. Besides, as viscosity now depends on the velocity field, we cannot formulate the diffusive part of the Navier-Stokes equations by a single Laplacian ($\rightarrow$ term $\frac{1}{\mathrm{Re}}\Delta u$) anymore.

1. Replace the discretization of the Laplacian in the evaluation of F, G and H by discrete expressions for

$$F: \quad 2\frac{\partial}{\partial x}\left((\nu+\nu_T)\frac{\partial u}{\partial x}\right)+\frac{\partial}{\partial y}\left((\nu+\nu_T)\left(\frac{\partial u}{\partial y}+\frac{\partial v}{\partial x}\right)\right)+\frac{\partial}{\partial z}\left((\nu+\nu_T)\left(\frac{\partial u}{\partial z}+\frac{\partial w}{\partial x}\right)\right) \tag{8}$$

$$G: \quad \frac{\partial}{\partial x}\left((\nu+\nu_T)\left(\frac{\partial v}{\partial x}+\frac{\partial u}{\partial y}\right)\right)+2\frac{\partial}{\partial y}\left((\nu+\nu_T)\frac{\partial v}{\partial y}\right)+\frac{\partial}{\partial z}\left((\nu+\nu_T)\left(\frac{\partial v}{\partial z}+\frac{\partial w}{\partial y}\right)\right) \tag{9}$$

$$H: \quad \frac{\partial}{\partial x}\left((\nu+\nu_T)\left(\frac{\partial w}{\partial x}+\frac{\partial u}{\partial z}\right)\right)+\frac{\partial}{\partial y}\left((\nu+\nu_T)\left(\frac{\partial w}{\partial y}+\frac{\partial v}{\partial z}\right)\right)+2\frac{\partial}{\partial z}\left((\nu+\nu_T)\frac{\partial w}{\partial z}\right). \tag{10}$$

The discretization is similar to the discretization process of the convective transport terms (cf. previous lectures); a more extensive description can be found in the book *Numerical simulation in fluid dynamics* by Griebel et al.; two exemplary discretizations are given in the following, *assuming a uniform meshsize*:

$$\begin{aligned}\left[\frac{\partial}{\partial x}\left(\nu^*\frac{\partial u}{\partial x}\right)\right]_{i,j,k} &= \frac{1}{\delta x^2}\left(\nu^*_{i+1,j,k}(u_{i+1,j,k}-u_{i,j,k})-\nu^*_{i,j,k}(u_{i,j,k}-u_{i-1,j,k})\right)\\ \left[\frac{\partial}{\partial y}\left(\nu^*\left(\frac{\partial u}{\partial y}+\frac{\partial v}{\partial x}\right)\right)\right]_{i,j,k} &= \frac{1}{\delta y}\left(\nu^*_{i+\frac{1}{2},j+\frac{1}{2},k}\left(\frac{u_{i,j+1,k}-u_{i,j,k}}{\delta y}+\frac{v_{i+1,j,k}-v_{i,j,k}}{\delta x}\right)\right.\\ &\quad \left.-\nu^*_{i+\frac{1}{2},j-\frac{1}{2},k}\left(\frac{u_{i,j,k}-u_{i,j-1,k}}{\delta y}+\frac{v_{i+1,j-1,k}-v_{i,j-1,k}}{\delta x}\right)\right)\end{aligned} \tag{11}$$

where $\nu^* := \nu + \nu_T$. All other expressions are evaluated analogously. Implement the new diffusive terms and make use of the stencil-iterator concept of NS-EOF as well as the new data structure.

2. Improve your implementation by extending it to stretched meshes. How do you need to adapt the formulas for F, G, H from above?

5.4 Parallelization

1. Extend interprocess communication by communication routines for the viscosity field. The communication of turbulent viscosity is carried out analogously to the communication of the pressure. How can you use the concept of inheritance to extend and reuse the existing communication routines from the `PetscParallelManager`?
2. Extend the configuration by broadcasting the new parameters to all processes. You may use the methods `MPI_Bcast(...)` and broadcastString(...), provided by MPI and the class `Configuration`.

5.5 Plugging things together

1. Use the concept of inheritance to create a simulation class `TurbulentSimulation` which possesses all properties of `Simulation` and incorporates the features of the algebraic turbulence modeling.
2. Adapt the time-step size restriction in `TurbulentSimulation`. Due to your inhomogeneous viscosity, the original expression for diffusive time-step size limitations

$$\delta t < \frac{\mathrm{Re}}{2}\left(\frac{1}{\delta x_{min}^2} + \frac{1}{\delta y_{min}^2} + \frac{1}{\delta z_{min}^2}\right)^{-1} \tag{12}$$

needs to be adapted where dx_{min}, dy_{min}, dz_{min} correspond to the minimum meshsizes used in the simulation. Since both meshsize and turbulent viscosity are local quantities and may differ from grid cell to grid cell, find a new expression f for the time-step size limitation,

$$\delta t < \min_{c \,\in\, cells} f(dx_c, dy_c, dz_c, \nu, \nu_t). \tag{13}$$

Implement the evaluation of the new time-step size criterion using the stencil-iterator concept.

6 Testing (5%)

1. Test that your implementations recover the laminar reference case for vanishing turbulent viscosities. (Note: Choose an appropriate way to quantify the error)
2. Test that your implementation to compute the minimal distance is done correctly. How can you verify that?
3. How can you verify that your finite difference expressions are evaluated correctly?

7 Flow Physics (15%)

1. Scenario **channel**:

- Study velocity and pressure profiles of the channel flow for Reynolds numbers 500, 3000, and 10000.
- Compare the results of the turbulent channel flow with the laminar reference case ($\nu_t = 0$). What do you observe, and how does it correspond to theory? (Note: focus on the velocity profile)
- How do the different choices of the boundary layer influence the results?
- How does the eddy viscosity behave in the field for the different Reynolds numbers? (Note: focus on the laminar boundary layer)

2. Scenario **backward-facing step**:
   - Study the velocity field and vortices for the backward facing step for Reynolds numbers 500, 3000, and 10000.
   - How do the different choices of the boundary layer influence the results?
   - How does the eddy viscosity behave in the field for the different Reynolds numbers? (Note: focus on the laminar boundary layer)
   - Discuss the use of the minimum distance.

8 Sbumission

Code

Please create a merge request, merging your branch `ws2` into `main`. You should be able to see all your changes in the merge request. We will use this merge request for reviewing your work and giving feedback.

Report

Please submit a report, where you summarize and visualize (e.g., figures, tables) your findings in a pdf file. Use the share-LaTeX template you can find on Moodle for the report. **Only the report will be used to evaluate the worksheet**. Additional figures, links, and information in the code repository will not be counted for the analysis of the flow physics and profiling.

Additionally, include some words about your teamwork. How did you organize yourself as a group? Which tools you did you use? What issues did you have?

A Appendix

Shear strain tensor:

$$S_{ij} = \begin{bmatrix} S_{11} & S_{12} & S_{13} \\ S_{21} & S_{22} & S_{23} \\ S_{31} & S_{32} & S_{33} \end{bmatrix} = \frac{1}{2} \begin{bmatrix} 2\frac{\partial \langle u_1 \rangle}{\partial x_1} & \frac{\partial \langle u_1 \rangle}{\partial x_2} + \frac{\partial \langle u_2 \rangle}{\partial x_1} & \frac{\partial \langle u_1 \rangle}{\partial x_3} + \frac{\partial \langle u_3 \rangle}{\partial x_1} \\ \frac{\partial \langle u_1 \rangle}{\partial x_2} + \frac{\partial \langle u_2 \rangle}{\partial x_1} & 2\frac{\partial \langle u_2 \rangle}{\partial x_2} & \frac{\partial \langle u_2 \rangle}{\partial x_3} + \frac{\partial \langle u_3 \rangle}{\partial x_2} \\ \frac{\partial \langle u_1 \rangle}{\partial x_3} + \frac{\partial \langle u_3 \rangle}{\partial x_1} & \frac{\partial \langle u_2 \rangle}{\partial x_3} + \frac{\partial \langle u_3 \rangle}{\partial x_2} & 2\frac{\partial \langle u_3 \rangle}{\partial x_3} \end{bmatrix} \tag{14}$$

Einstein summation for $S_{ij}S_{ij}$:

$$S_{ij}S_{ij} = \left(S_{11}^2 + S_{22}^2 + S_{33}^2\right) + 2\left(S_{12}^2 + S_{13}^2 + S_{23}^2\right) \tag{15}$$

Downstream Reynolds number:

$$Re_x = \frac{\langle u_0 \rangle x}{\nu} \tag{16}$$