Steps we need to take:

---

1. Familiarize Yourself with the Codebase

- **Read the README.md in detail to understand the project's goals, structure, and usage.**
- **Explore the directory structure and identify key files (source code, tests, scripts, resources).**

---

2. Set Up the Development Environment

- **Ensure you have the required compilers installed (g++ or icpx).**
- **If performance measurement is important, install LIKWID for benchmarking support.**
- **Make sure you have make and any other dependencies (check for requirements in the README or source code).**

---

3. Build the Project

- **Run the standard compilation command to make sure the project builds:**

bash

- **CXX=<compiler> make**
-
- **If you encounter errors, resolve missing dependencies or configuration issues.**

---

4. Run the Tests

- **Execute the provided tests to verify that the build is successful and the code works as expected:**

bash

```
./test
```

- **Review the output for any failed tests or warnings.**

---

5. Explore Debug and Performance Modes

- **Compile and run in debug mode for deeper insights:**

bash

```
CXX=<compiler> make EXTRA_FLAGS=-DDEBUG
./test
```

- **Use performance mode if LIKWID is available:**

bash

```
LIKWID=on CXX=<compiler> make
./perf <grid size y> <grid size x>
```

---

6. Generate and Review Plots

- **If you need to visualize results:**

bash

```
./plot.sh
```

-

- Compare generated plots with reference images in the resources folder.

---

7. Review and Update Documentation

- If you make changes or improvements, update the README and/or add developer notes for clarity.
- Document any new dependencies, scripts, or usage patterns.

---

8. Plan and Assign Tasks

- Identify improvements, bugs, or features you want to tackle.
- Create GitHub issues for each task and assign them to yourself or teammates.
- Example tasks: code refactoring, performance tuning, additional tests, documentation.

---

9. Set Milestones and Deadlines

- Use GitHub Projects or Milestones to organize work into phases (e.g., Setup, Testing, Optimization).
- Assign deadlines based on your schedule and course requirements.

---

10. Establish a Regular Workflow

- **Commit changes frequently and use descriptive messages.**
- **Use branches for new features or bug fixes.**
- **Review each other’s code if you’re working as a team via pull requests.**

---

11. Monitor Progress and Adjust Plan

- **Regularly review open issues and milestones.**
- **Adjust timelines or priorities as needed.**

ADDITIONAL INFO:

System & Compiler Settings:

- Use Ice Lake nodes (Fritz) and fix CPU frequency to 2.2 GHz.
- Compile with -O3 -march=icelake-server -fopenmp -funroll-loops.
- Prefer Intel's icpx compiler for better AVX-512 support.

Code Optimization:

- Apply OpenMP parallelization to performance-critical loops, especially in stencil application and CG/PCG solvers.
- Use loop fusion to reduce memory access and improve cache performance.
- Avoid if statements inside loops that run per grid point; restructure conditionals to run outside hot loops.
- Stick to the matrix-free stencil approach for computing matrix-vector products.
- Optimize the preconditioner implementation carefully; Gauss-Seidel is difficult to parallelize.

Benchmarking Best Practices:

- Use the ./perf executable to measure runtime.
- Use scripts for development-time profiling only; remove all logging or measurement overhead during the final leaderboard run.
- Use LIKWID=on for performance profiling during optimization.

- Request dedicated nodes, fix CPU frequency, and use thread pinning to ensure stable results.

Tools and Workflow Notes:

- Open the SSH config file in VS Code with: code ~/.ssh/config.
- Ensure there is only one .ssh directory, typically at C:\Users\<YourUsername>\.ssh.
- Generate assembly output using icpx -S or g++ -S to inspect low-level performance.