HPC Project Manual

Summer 2025

Ioannis Vardas, TU Wien, vardas@par.tuwien.ac.at

1 Building and Configuring the project

Before trying to configure and build the project ensure that the following are installed on your system.

1.1 Requirements

1. C/C++ compiler that supports C++17 standard (gcc or clang).
2. CMake version > 3.10.
3. MPI Library: Open MPI (preferred) or MPICH.

1.2 Quick Overview

1. Configure first in the project root (the directory named `sources`):

   ```
   cmake -B build -S . -DCMAKE_BUILD_TYPE=Release
   ```

2. Build (produces executable in `./build/allgather_merge`).

   ```
   cmake --build build --parallel
   ```

3. Running/Testing

   - **Local First**: Test thoroughly on your machine before using the cluster.
     - Use Standard Mode: `./allgather_merge [options] -c`.
     - Use Measurements Mode: `./allgather_merge -c`.
   - **Hydra Cluster (for Report)**:
     - See the instructions on how to use Hydra.
     - Use Measurements Mode (`./allgather_merge` without `-m` and without `-c`) for final results.
     - Run experiments using the provided `job-run.sh` script (check paths inside it first).

1.3 Configure

Configuration prepares the project for compilation. We use CMake build system to build the project. There are two configurations available: `Debug` and `Release`. The `Debug` configuration is optional, it is meant to help you debug your code. The main components of the project are the source files .cpp, .c and .h and the `CMakeLists.txt`, which is used by CMake. When you configure your project you specify a build folder.

- Run the configuration commands in the top folder of the project, which is where `CMakeLists.txt` is located.
- **For the first configuration use the Release configuration**.

1.3.1 Release Configuration

```
cmake -B build -S . -DCMAKE_BUILD_TYPE=Release
```

- The above commands will create a folder named "build" where the project will be build next.
- You can use the Release configuration for testing the correctness of your code locally.
- Use the Release configuration to perform the experiments and get measurements on hydra when your code is correct.

1.3.2 Debug Configuration (Optional)

```
cmake -B build_debug -S . -DCMAKE_BUILD_TYPE=Debug
```

- This activates the DEBUG directives, allowing you to print debug information in specific parts of your code.
- You can add DEBUG directives in specific parts of your code using `#ifdef DEBUG`, see examples of in the **main.cpp** file.

1.4 Build

The build phase compiles the code into an executable. After a successful Release configuration you can build the project:

```
cmake --build build --parallel
```

- A successful build will produce the executable: `./build/allgather_merge`
- Run this command to rebuild your project after you change the source code. You do not need to reconfigure the project.

1.4.1 Debug Build (Optional)

If you configured the project with Debug:

```
cmake --build build_debug --parallel
```

This produces a debug executable: `./build_debug/allgather_merge`

2 Development

- The `algorithms.h` file contains the function signatures of the three algorithms you will need to implement for the project. **DO NOT edit this file**.
- For simplicity you can assume that the `recvbuf` and `sendbuf` data types are `tuwtype_t`, which is defined in `algorithm.h` as `MPI_INT`:

```
#define TUW_TYPE MPI_INT
typedef int tuwtype_t;
```

  This simplifies declaring the types of internal buffers that you may use.

- Write your implementations in the following files:
  - `baseline.cpp`: The Baseline algorithm.
  - `algorithm1.cpp`: Distributed merging with Bruck's algorithm.
  - `algorithm2.cpp`: Distributed merging with a circulant algorithm.
- The algorithms are called by the main function in `main.cpp`. You can add code for debugging (DEBUG directives) or correctness checks, and you can also adjust the `WARMUP` and `REPEAT` parameters but **you should not change the main structure** `main.cpp`.

2.1 Adding your own source files (Optional)

- If you want to add other source files in the project, you can append them in line 18 of `CMakeLists.txt`.
- The source files can be either C or C++. Use `.c` suffix for C files and `.cpp` for C++.
- **The suffix is important** because CMake will invoke the C compiler if the file ends in `.c` and the C++ compiler if the file ends in `.cpp`.
- You can also change the compiler options or flags such as `-Wall` in `CMakeLists.txt`.

3 Run/Test

- To run on your local machine use the `mpirun` provided by the MPI installation. For example, `mpirun -np <number of processes> ./executable args`.
- There are two major modes for running, the Standard and Measurements mode.

3.1 Standard Mode

- **Use this mode to test, debug and check the correctness of your solutions**.
- To run in Standard Mode, you must specify the message size (`-m, --msgsize`) as input argument (use `./allgather_merge -h` to print the list of the available arguments):

```
Usage: ./allgather_merge [-m <value>] [-a <value>] [-t <value>] [-c <value>]
  -m, --msgsize: Message size (default: 10)
  -a, --algo: Algorithm (0-baseline, 1-Bruck, 2-Circulant) (default: 0)
  -t, --type: Input type (0-2) (default: 0)
  -c, --check: Verify results (default: false)
```

- You can also run debug builds (`-DCMAKE_BUILD_TYPE=Debug`) in this mode.
- Run in Standard Mode and make sure your solutions do not produce errors and they are correct before running in Measurements Mode.

3.2 Measurements Mode

- **Use this mode to produce the results for your report**.
- Measurements Mode is activated **if you do not** specify any message size (`-m, --msgsize`).
- This runs all three algorithms, each with seven message sizes (see `main.cpp`), and three types of input.
- Run in Measurements Mode only after performing some basic correctness checks in Standard Mode and ensure that the project is configured with `-DCMAKE_BUILD_TYPE=Release`.
- You can use the `-c` to verify your results for multiple message sizes in Measurements Mode. However, because this can take too long (with many processes), so you should do it **only on your local machine**.

4 General Workflow

We strongly suggest that you run the code locally before running on the hydra cluster otherwise the system will be overloaded. If the system is overloaded, all student experiments will take longer to complete, because hydra is shared by all students.

4.1 Local Machine

- Configure the project.
- Build the project and do not forget to rebuild it anytime you modify the source code.
- Implement your solutions.
- Run and Test on your **local machine** and check the following:
  1. The code does not produce runtime errors, e.g., segmentation faults.
  2. The code is correct, meaning that the output is as expected.
- You are advised to run both in Standard and Measurements modes on your local machine.

4.2 Hydra Cluster

If your code is correct and does not produce runtime errors run on hydra cluster.

1. Ensure that you can connect to hydra: Instructions for connecting to hydra. The link works only from the internal TU Wien network.
2. Copy **only your sources** on hydra: Instructions for transferring data on hydra. Do not copy binaries, you need only the CMakeLists.txt and the source files that are `sources` folder.
3. SSH to hydra to get a remote shell.

4. Load the environment: `spack load openmpi/4.1.5`.

5. Configure and build the project.

6. Before running the experiments read the following: Slurm commands and Running MPI jobs on Hydra.

7. We use the `sbatch` utility to run jobs on Hydra, we do not use `mpirun`.

8. Use the provided `job-run.sh` in the `sources` folder SBATCH script to run your experiments in Measurements Mode on hydra: `sbatch job-run.sh`

    - Before executing the `sbatch` command, examine the `job-run.sh` and ensure that `OUTFOLDER` and `EXEC_PATH` are correct.
    - In case your code is very slow, modify the `for` loop in `job-run.sh` to run with only one `ntasks`, e.g., `for ntasks in 32; do`.

9. If successful, the script creates a folder with three files in it, each will contain the results from the respective Nodes x Processes Per Node (PPN) configuration.

    - The form of the file names is `NodesxPPN_JOBID.out`. The `.out` files contain the results, which you can copy into your report.
    - The script will also produce a slurm log (`.log`) and error files (`.err`) for each configuration in the `OUTFOLDER`.
    - Check the error files for potential errors.