Contents

1 Parallel Programming Task Report Template

This is the root report template for course tasks. The complete template is split across this report tree. Fill in only the branch that matches your semester or assignment:

- use `threads/report.md` for thread-based tasks;
- use `processes/report.md` for MPI/process tasks;
- use implementation-level reports for the concrete technologies that you implemented.

Students normally complete either the thread-based semester or the process/MPI semester. Do not describe both groups unless the assignment explicitly requires both.

1.1 General Information

Fill in this table for the whole task report.

Field	Value
Student full name	`<Student full name>`
Group	`<Group>`
Task name	`<Task name>`
Task path in repository	`<Path to task>`
Implementation variants	`<Sequential / MPI / OpenMP / oneTBB / STL / Combined>`
Used technologies	`<Used technologies>`
Repository branch / commit	`<Branch or commit hash>`

1.2 How to Use the Report Tree

Use the existing reports below instead of creating new Markdown files.

For a single PDF report, assemble `report.md` first, then exactly one semester branch unless the assignment explicitly requires both branches. Include only the implementation-level reports that match the completed work. A full course template PDF may include both branches for reference, but a student submission should normally keep only the branch assigned for the semester.

Recommended PDF order for a thread-based semester:

1. `report.md`
2. `threads/report.md`
3. `threads/seq/report.md`
4. `threads/omp/report.md`
5. `threads/tbb/report.md`
6. `threads/stl/report.md`
7. `threads/all/report.md`

Recommended PDF order for an MPI/process semester:

1. `report.md`
2. `processes/report.md`
3. `processes/t1/report.md`
4. `processes/t1/seq/report.md`
5. `processes/t1/mpi/report.md`
6. `processes/t2/report.md`
7. `processes/t2/seq/report.md`
8. `processes/t2/mpi/report.md`
9. `processes/t3/report.md`
10. `processes/t3/seq/report.md`
11. `processes/t3/mpi/report.md`

Report file	What to fill in
`threads/report.md`	Thread-semester task overview and thread results
`threads/seq/report.md`	Sequential baseline for the thread task
`threads/omp/report.md`	OpenMP implementation details
`threads/tbb/report.md`	oneTBB implementation details
`threads/stl/report.md`	STL or `std::thread` implementation details
`threads/all/report.md`	Combined implementation details
`processes/report.md`	Process-semester overview and process results
`processes/t1/report.md`	Process task T1 statement, verification, and result
`processes/t1/seq/report.md`	Sequential baseline for process task T1
`processes/t1/mpi/report.md`	MPI implementation for process task T1
`processes/t2/report.md`	Process task T2 statement, verification, and result
`processes/t2/seq/report.md`	Sequential baseline for process task T2
`processes/t2/mpi/report.md`	MPI implementation for process task T2
`processes/t3/report.md`	Process task T3 statement, verification, and result
`processes/t3/seq/report.md`	Sequential baseline for process task T3
`processes/t3/mpi/report.md`	MPI implementation for process task T3

1.3 Shared Files to Mention When Relevant

Mention these shared files when they affect your task, tests, or measurements.

Repository path	What to mention in the report
`common/include/common.hpp`	Defines shared `InType`, `OutType`, `TestType`, `BaseTask`
`data/pic.ppm`	Shared image fixture used by functional tests
`settings.json`	Enables or disables example implementations
`info.json`	Stores student metadata for automation
`threads/tests/functional`	Functional tests for thread implementations
`threads/tests/performance`	Performance tests for thread implementations
`processes/t*/tests/functional`	Functional tests for each process task
`processes/t*/tests/performance`	Performance tests for each process task

1.4 Where to Describe Implementations

Write implementation details in the file that matches the implementation you completed.

- Sequential implementation: `threads/seq/report.md` or `processes/t*/seq/report.md`
- MPI / processes implementation: `processes/t*/mpi/report.md`
- OpenMP implementation: `threads/omp/report.md`
- oneTBB implementation: `threads/tbb/report.md`
- STL / `std::thread` implementation: `threads/stl/report.md`
- Combined implementation: `threads/all/report.md`

1.5 Performance Formulas

Use these formulas in the relevant semester report:

```
Speedup = Sequential time / Parallel time
Efficiency = Speedup / Number of workers
```

Workers can mean the number of threads, the number of processes, or a combined process/thread configuration depending on the implementation.

1.6 Appendix

Use this root appendix for shared material that applies to the whole task.

- Additional logs: `<Additional logs>`
- Raw benchmark output: `<Raw benchmark output>`
- Links to source files: `<Links to source files>`
- Screenshots, if needed: `<Screenshots>`

1.7 How to Fill in This Report

- Do not write only “it became faster” without measurements.
- Always compare parallel implementations with the sequential implementation.
- Always specify input data and launch parameters.
- Explain bad, unexpected, or unstable results.
- The report must be reproducible.
- Do not invent performance numbers.
- Use the same input data for fair comparison.
- Use Release builds for performance measurements unless explicitly stated otherwise.
- Fill in only the implementations required by your semester or assignment.
- Mark non-applicable sections as `N/A` if your instructor requires the section to remain visible.

2 Thread-Based Task Report

Fill in this report for a thread-based semester or assignment. It aggregates the sequential, OpenMP, oneTBB, STL, and combined thread-level reports. Skip implementations that are not required for your assignment.

Child reports:

- `seq/report.md`
- `omp/report.md`
- `tbb/report.md`
- `stl/report.md`
- `all/report.md`

2.1 Where to Write Thread-Semester Details

Repository path	What to describe
`seq/include`, `seq/src`	Sequential baseline implementation
`omp/include`, `omp/src`	OpenMP implementation
`tbb/include`, `tbb/src`	oneTBB implementation
`stl/include`, `stl/src`	STL or `std::thread` implementation
`all/include`, `all/src`	Combined implementation
`tests/functional/main.cpp`	Functional tests using `data/pic.ppm`
`tests/performance/main.cpp`	Performance tests for all thread variants

2.2 Problem Statement

Describe the thread-based task.

- What must be implemented: `<Description of the required computation>`
- Input data: `<Input data format, source, size, and meaning>`
- Output data: `<Output data format and meaning>`
- Constraints: `<Input limits, memory limits, time limits, or domain-specific restrictions>`
- Correctness criteria: `<Rules used to decide that the output is correct>`
- Expected behavior: `<Expected behavior for normal, edge, and invalid inputs>`

2.3 Algorithm Description

Explain the algorithm used by the thread-based implementations.

- General idea of the algorithm: `<Algorithm overview>`
- Sequential algorithm: `<Step-by-step sequential baseline>`
- Parallel decomposition: `<How work is split between threads>`
- Data partitioning: `<How input data is divided between threads>`
- Synchronization strategy: `<Barriers, locks, reductions, atomics, or none>`
- Result aggregation: `<How partial thread results are combined>`
- Possible data races and how they are avoided: `<Race risks and prevention strategy>`
- Complexity, if applicable: `<Time and memory complexity>`

2.4 Build and Run Instructions

Provide the commands used for the thread-based task.

```
<Configure command>
<Build command>
<Test command>
<Run command>
```

- Command-line arguments: `<Command-line arguments>`

2.5 Correctness Verification

Explain how thread implementations were checked against the sequential baseline.

- Implemented tests: `<Thread functional tests and performance tests>`

- Normal cases: `<Normal cases>`
- Edge cases: `<Edge cases>`
- Invalid input cases, if applicable: `<Invalid input cases>`
- Comparison with sequential implementation: `<How thread outputs were compared with sequential output>`
- Floating-point tolerance, if applicable: `<Tolerance value and reason>`
- Correctness explanation: `<How correctness was checked>`
- Test input data: `data/pic.ppm` is loaded in functional tests through `stb_image`
- Functional test class to mention: `NesterovARunFuncTestsThreads`
- Performance test class to mention: `ExampleRunPerfTestThreads`
- Implementation classes to describe: `NesterovATestTaskSEQ`, `NesterovATestTaskOMP`, `NesterovATestTaskTBB`, `NesterovATestTaskSTL`, `NesterovATestTaskALL`

Test case	Input size / parameters	Expected result	Actual result	Status
`<Test name>`	`<Input>`	`<Expected>`	`<Actual>`	`<Passed/Failed>`

2.6 Experiment Setup

Describe the environment used for thread performance measurements.

Parameter	Value
CPU	`<CPU model>`
RAM	`<RAM size>`
OS	`<Operating system>`
Compiler	`<Compiler and version>`
Build type	`<Debug/Release>`
CMake options	`<CMake options>`
Number of runs	`<N>`
Input data sizes	`<Input data sizes>`
Number of threads	`<Thread counts>`
Time measurement method	`<Method>`

2.7 Performance Results

Use the same input data for every implementation.

2.7.1 Execution Time

Implementation	Input size	Processes	Threads	Time, ms	Notes
Sequential	`<N>`	1	1	`<Time>`	`<Notes>`
OpenMP	`<N>`	1	`<T>`	`<Time>`	`<Notes>`
oneTBB	`<N>`	1	`<T>`	`<Time>`	`<Notes>`
STL	`<N>`	1	`<T>`	`<Time>`	`<Notes>`
Combined	`<N>`	`<P>`	`<T>`	`<Time>`	`<Notes>`

2.7.2 Speedup

Implementation	Input size	Threads	Sequential time, ms	Parallel time, ms	Speedup
`<Implementation>`	`<N>`	`<T>`	`<Seq time>`	`<Par time>`	`<Speedup>`

2.7.3 Efficiency

Implementation	Input size	Threads	Speedup	Efficiency
`<Implementation>`	`<N>`	`<T>`	`<Speedup>`	`<Efficiency>`

2.7.4 Best Result Summary

Implementation	Best configuration	Best time, ms	Speedup	Comment
`<Implementation>`	`<Configuration>`	`<Time>`	`<Speedup>`	`<Comment>`

```
Speedup = Sequential time / Parallel time
Efficiency = Speedup / Number of workers
```

For thread-based implementations, workers usually mean the number of threads.

2.8 Results Analysis

- Which thread implementation showed the best performance?
- Why did this implementation perform better?
- Where was speedup not achieved?
- What overheads affected the result?
- How did the number of threads affect performance?
- Were there any anomalous results?
- How can the implementation be improved?

2.9 Conclusion

- Short summary: `<Summary of completed work>`
- Best implementation: `<Best implementation>`
- Explanation of the best result: `<Why this result was best>`
- Limitations: `<Current limitations>`
- Possible future improvements: `<Future improvements>`

2.10 Thread Task Sequential Implementation

Fill in this report with the sequential baseline for the thread-based task. Use it as the baseline for all thread-level comparisons.

- Source files: `<Sequential source files>`
- Source files to describe: `include/ops_seq.hpp`, `src/ops_seq.cpp`
- Short description: `<What the sequential implementation does>`
- Main functions/classes: `<Main functions or classes>`
- Task class to describe: `NesterovATestTaskSEQ`
- Important implementation details: `<Important implementation details>`
- Methods to describe: `ValidationImpl`, `PreProcessingImpl`, `RunImpl`, `PostProcessingImpl`
- Input preparation: `<How input data is prepared for the baseline>`
- Output validation: `<How baseline output is validated>`
- Limitations: `<Known limitations>`

2.11 Thread Task OpenMP Implementation

Fill in this report with the OpenMP implementation for the thread-based task. Compare it with `../seq/report.md`.

- Source files: `<OpenMP source files>`
- Source files to describe: `include/ops_omp.hpp`, `src/ops_omp.cpp`
- Task class to describe: `NesterovATestTaskOMP`
- Number of threads: `<Number of threads>`
- Parallel regions: `<Parallel regions used>`
- Parallel construct to explain: `#pragma omp parallel`
- Scheduling strategy: `<Static / dynamic / guided / default scheduling>`
- Reductions / critical sections / atomics, if used: `<Synchronization primitives>`
- Synchronization primitive to explain: `std::atomic<int>`
- Result aggregation: `<How partial thread results are combined>`
- Methods to describe: `ValidationImpl`, `PreProcessingImpl`, `RunImpl`, `PostProcessingImpl`
- Limitations: `<Known limitations>`

2.12 Thread Task oneTBB Implementation

Fill in this report with the oneTBB implementation for the thread-based task. Compare it with `../seq/report.md`.

- Source files: `<oneTBB source files>`
- Source files to describe: `include/ops_tbb.hpp`, `src/ops_tbb.cpp`
- Task class to describe: `NesterovATestTaskTBB`
- Used oneTBB algorithms: `<parallel_for / parallel_reduce / flow graph / other algorithms>`
- oneTBB algorithm to explain: `tbb::parallel_for`
- Blocked ranges / partitioning: `<Range and partitioning strategy>`
- Reductions, if used: `<Reduction strategy>`
- Task-based decomposition: `<Task decomposition strategy>`
- Result aggregation: `<How partial task results are combined>`
- Synchronization primitive to explain: `std::atomic<int>`
- Methods to describe: `ValidationImpl`, `PreProcessingImpl`, `RunImpl`, `PostProcessingImpl`
- Limitations: `<Known limitations>`

2.13 Thread Task STL / std::thread Implementation

Fill in this report with the STL or `std::thread` implementation for the thread-based task. Compare it with `../seq/report.md`.

- Source files: `<STL or std::thread source files>`
- Source files to describe: `include/ops_stl.hpp`, `src/ops_stl.cpp`
- Task class to describe: `NesterovATestTaskSTL`
- Number of threads: `<Number of threads>`
- Manual thread partitioning: `<How work is split manually>`
- Thread primitive to explain: `std::thread`
- Synchronization primitives: `<mutex / atomic / condition_variable / other primitives>`
- Synchronization primitive to explain: `std::atomic<int>`
- Result aggregation: `<How thread results are combined>`
- Methods to describe: `ValidationImpl`, `PreProcessingImpl`, `RunImpl`, `PostProcessingImpl`
- Limitations: `<Known limitations>`

2.14 Thread Task Combined Implementation

Fill in this report with the combined or `all` implementation. Use it only when the assignment requires combined process/thread or multi-technology parallelism.

- Source files: `<Combined implementation source files>`
- Source files to describe: `include/ops_all.hpp`, `src/ops_all.cpp`
- Task class to describe: `NesterovATestTaskALL`
- Processes and threads configuration: `<Processes and threads per process>`
- Interaction between MPI and thread-level parallelism: `<How process-level and thread-level parallelism interact>`
- MPI calls to explain: `MPI_Comm_rank`, `MPI_Barrier`
- Data exchange: `<How data is exchanged between processes and threads>`
- Thread-level technology: `<OpenMP / oneTBB / STL / other technology>`
- Thread-level constructs to explain: `#pragma omp parallel`, `std::thread`, `tbb::parallel_for`
- Result aggregation: `<How final results are combined>`
- Synchronization primitive to explain: `std::atomic<int>`
- Methods to describe: `ValidationImpl`, `PreProcessingImpl`, `RunImpl`, `PostProcessingImpl`
- Limitations: `<Known limitations>`

3 Process-Based Task Report

Fill in this report for an MPI/process semester or assignment. It aggregates process tasks T1, T2, and T3. Fill in only the tasks required for your assignment. Each task must compare its MPI implementation only with its own sequential baseline.

Child reports:

- `t1/report.md`
- `t2/report.md`
- `t3/report.md`

3.1 Where to Write Process-Semester Details

Repository path	What to describe
`t1/seq/include`, `t1/seq/src`	Sequential baseline for process task T1
`t1/mpi/include`, `t1/mpi/src`	MPI implementation for process task T1
`t1/tests/functional/main.cpp`	Functional tests for process task T1
`t1/tests/performance/main.cpp`	Performance tests for process task T1
`t2/seq/include`, `t2/seq/src`	Sequential baseline for process task T2
`t2/mpi/include`, `t2/mpi/src`	MPI implementation for process task T2
`t2/tests/functional/main.cpp`	Functional tests for process task T2
`t2/tests/performance/main.cpp`	Performance tests for process task T2
`t3/seq/include`, `t3/seq/src`	Sequential baseline for process task T3
`t3/mpi/include`, `t3/mpi/src`	MPI implementation for process task T3
`t3/tests/functional/main.cpp`	Functional tests for process task T3
`t3/tests/performance/main.cpp`	Performance tests for process task T3

Each `t1`, `t2`, and `t3` report owns its own correctness and performance baseline. Do not compare an MPI task with a sequential implementation from another process task or from the thread branch.

3.2 Problem Statement

Describe the process-based assignment scope.

- Required process tasks: `<T1 / T2 / T3>`
- Input data: `<Input data format, source, size, and meaning>`
- Output data: `<Output data format and meaning>`
- Constraints: `<Input limits, memory limits, process limits, or memory limits>`
- Correctness criteria: `<Rules used to decide that the output is correct>`
- Expected behavior: `<Expected behavior for normal, edge, and invalid inputs>`

3.3 Build and Run Instructions

Provide the commands used for process-based tasks.

```
<Configure command>
<Build command>
<Test command>
<Run command>
```

- Command-line arguments: `<Command-line arguments>`

3.4 Experiment Setup

Describe the environment used for MPI/process performance measurements.

Parameter	Value
CPU	`<CPU model>`
RAM	`<RAM size>`
OS	`<Operating system>`
Compiler	`<Compiler and version>`

Parameter	Value
Build type	<Debug/Release>
CMake options	<CMake options>
Number of runs	<N>
Input data sizes	<Input data sizes>
Number of processes	<Process counts>
Time measurement method	<Method>

3.5 Performance Results

Use a separate sequential baseline for each process task.

3.5.1 Execution Time

Task	Implementation	Input size	Processes	Threads	Time, ms	Notes
<T1>	Sequential	<N>	1	1	<Time>	<Notes>
<T1>	MPI	<N>	<P>	1	<Time>	<Notes>
<T2>	Sequential	<N>	1	1	<Time>	<Notes>
<T2>	MPI	<N>	<P>	1	<Time>	<Notes>
<T3>	Sequential	<N>	1	1	<Time>	<Notes>
<T3>	MPI	<N>	<P>	1	<Time>	<Notes>

3.5.2 Speedup

Task	Input size	Processes	Sequential time, ms	Parallel time, ms	Speedup
<Task>	<N>	<P>	<Seq time>	<Par time>	<Speedup>

3.5.3 Efficiency

Task	Input size	Processes	Speedup	Efficiency
<Task>	<N>	<P>	<Speedup>	<Efficiency>

3.5.4 Best Result Summary

Task	Best configuration	Best time, ms	Speedup	Comment
<Task>	<Configuration>	<Time>	<Speedup>	<Comment>

```
Speedup = Sequential time / Parallel time
Efficiency = Speedup / Number of workers
```

For process-based implementations, workers usually mean the number of MPI processes.

3.6 Results Analysis

- Which process task showed the best MPI speedup?
- Why did this task perform better?
- Where was speedup not achieved?
- What MPI communication overheads affected the result?
- How did the number of processes affect performance?
- Were there any anomalous results?
- How can the implementation be improved?

3.7 Conclusion

- Short summary: `<Summary of completed process tasks>`
- Best implementation: `<Best task and configuration>`
- Explanation of the best result: `<Why this result was best>`
- Limitations: `<Current limitations>`
- Possible future improvements: `<Future improvements>`

3.8 Process Task T1 Report

Fill in this report for the first process-based task. Compare `mpi/report.md` only with `seq/report.md` from this same `t1` directory.

Child reports:

- `seq/report.md`
- `mpi/report.md`

3.8.1 Problem Statement

- What must be implemented: `<T1 required computation>`
- Input data: `<T1 input data>`
- Output data: `<T1 output data>`
- Constraints: `<T1 constraints>`
- Correctness criteria: `<T1 correctness criteria>`
- Expected behavior: `<T1 expected behavior>`

3.8.2 Algorithm Description

- General idea of the algorithm: `<T1 algorithm overview>`
- Sequential algorithm: `<T1 sequential baseline>`
- Parallel decomposition: `<How T1 work is split between processes>`
- Data partitioning: `<How T1 data is divided>`
- Synchronization strategy: `<MPI synchronization strategy>`
- Result aggregation: `<How T1 partial results are combined>`
- Possible data races and how they are avoided: `N/A for separate MPI process memory unless threads are used`
- Complexity, if applicable: `<T1 time and memory complexity>`

3.8.3 Correctness Verification

- Implemented tests: `<T1 tests>`
- Normal cases: `<T1 normal cases>`
- Edge cases: `<T1 edge cases>`
- Invalid input cases, if applicable: `<T1 invalid input cases>`
- Comparison with sequential implementation: `<How T1 MPI output is compared with T1 sequential output>`
- Floating-point tolerance, if applicable: `<Tolerance value and reason>`
- Correctness explanation: `<How T1 correctness was checked>`
- Test input data: `data/pic.ppm` is loaded in functional tests through `stb_image`
- Functional test class to mention: `NesterovARunFuncTestsProcesses`
- Performance test class to mention: `ExampleRunPerfTestProcesses`
- Implementation classes to describe: `NesterovATestTaskSEQ`, `NesterovATestTaskMPI`

Test case	Input size / parameters	Expected result	Actual result	Status
`<Test name>`	`<Input>`	`<Expected>`	`<Actual>`	`<Passed/Failed>`

3.8.4 Performance Results

Implementation	Input size	Processes	Threads	Time, ms	Notes
Sequential	`<N>`	1	1	`<Time>`	`<Notes>`
MPI	`<N>`	`<P>`	1	`<Time>`	`<Notes>`

3.8.5 Process Task T1 Sequential Implementation

Fill in this report with the sequential baseline for process task T1. Use it as the only baseline for `../mpi/report.md`.

- Source files: `<T1 sequential source files>`
- Source files to describe: `include/ops_seq.hpp`, `src/ops_seq.cpp`
- Short description: `<T1 sequential implementation description>`
- Main functions/classes: `<T1 sequential functions or classes>`
- Task class to describe: `NesterovATestTaskSEQ`
- Important implementation details: `<T1 sequential implementation details>`
- Methods to describe: `ValidationImpl`, `PreProcessingImpl`, `RunImpl`, `PostProcessingImpl`
- Input preparation: `<How input data is prepared for the baseline>`
- Output validation: `<How baseline output is validated>`
- Limitations: `<Known limitations>`

3.8.6 Process Task T1 MPI Implementation

Fill in this report with the MPI implementation for process task T1. Compare it only with `../seq/report.md`.

- Source files: `<T1 MPI source files>`
- Source files to describe: `include/ops_mpi.hpp`, `src/ops_mpi.cpp`
- Task class to describe: `NesterovATestTaskMPI`
- Number of processes: `<Number of processes>`
- Data distribution: `<How T1 data is distributed between processes>`
- Communication pattern: `<Point-to-point or collective communication pattern>`
- Collective operations, if used: `<MPI_Bcast / MPI_Reduce / MPI_Gather / other collectives>`
- Result gathering: `<How final T1 results are collected>`
- Synchronization points: `<MPI synchronization points>`
- MPI calls to explain: `MPI_Comm_rank`, `MPI_Barrier`
- Methods to describe: `ValidationImpl`, `PreProcessingImpl`, `RunImpl`, `PostProcessingImpl`
- Limitations: `<Known limitations>`

3.9 Process Task T2 Report

Fill in this report for the second process-based task. Compare `mpi/report.md` only with `seq/report.md` from this same `t2` directory.

Child reports:

- `seq/report.md`
- `mpi/report.md`

3.9.1 Problem Statement

- What must be implemented: `<T2 required computation>`
- Input data: `<T2 input data>`
- Output data: `<T2 output data>`
- Constraints: `<T2 constraints>`
- Correctness criteria: `<T2 correctness criteria>`
- Expected behavior: `<T2 expected behavior>`

3.9.2 Algorithm Description

- General idea of the algorithm: `<T2 algorithm overview>`
- Sequential algorithm: `<T2 sequential baseline>`
- Parallel decomposition: `<How T2 work is split between processes>`
- Data partitioning: `<How T2 data is divided>`
- Synchronization strategy: `<MPI synchronization strategy>`
- Result aggregation: `<How T2 partial results are combined>`
- Possible data races and how they are avoided: `N/A for separate MPI process memory unless threads are used`
- Complexity, if applicable: `<T2 time and memory complexity>`

3.9.3 Correctness Verification

- Implemented tests: `<T2 tests>`
- Normal cases: `<T2 normal cases>`
- Edge cases: `<T2 edge cases>`
- Invalid input cases, if applicable: `<T2 invalid input cases>`
- Comparison with sequential implementation: `<How T2 MPI output is compared with T2 sequential output>`
- Floating-point tolerance, if applicable: `<Tolerance value and reason>`
- Correctness explanation: `<How T2 correctness was checked>`
- Test input data: `data/pic.ppm` is loaded in functional tests through `stb_image`
- Functional test class to mention: `NesterovARunFuncTestsProcesses2`
- Performance test class to mention: `ExampleRunPerfTestProcesses2`
- Implementation classes to describe: `NesterovATestTaskSEQ`, `NesterovATestTaskMPI`

Test case	Input size / parameters	Expected result	Actual result	Status
`<Test name>`	`<Input>`	`<Expected>`	`<Actual>`	`<Passed/Failed>`

3.9.4 Performance Results

Implementation	Input size	Processes	Threads	Time, ms	Notes
Sequential	`<N>`	1	1	`<Time>`	`<Notes>`
MPI	`<N>`	`<P>`	1	`<Time>`	`<Notes>`

3.9.5 Process Task T2 Sequential Implementation

Fill in this report with the sequential baseline for process task T2. Use it as the only baseline for `../mpi/report.md`.

- Source files: `<T2 sequential source files>`
- Source files to describe: `include/ops_seq.hpp`, `src/ops_seq.cpp`
- Short description: `<T2 sequential implementation description>`
- Main functions/classes: `<T2 sequential functions or classes>`
- Task class to describe: `NesterovATestTaskSEQ`
- Important implementation details: `<T2 sequential implementation details>`
- Methods to describe: `ValidationImpl`, `PreProcessingImpl`, `RunImpl`, `PostProcessingImpl`
- Input preparation: `<How input data is prepared for the baseline>`
- Output validation: `<How baseline output is validated>`
- Limitations: `<Known limitations>`

3.9.6 Process Task T2 MPI Implementation

Fill in this report with the MPI implementation for process task T2. Compare it only with `../seq/report.md`.

- Source files: `<T2 MPI source files>`
- Source files to describe: `include/ops_mpi.hpp`, `src/ops_mpi.cpp`
- Task class to describe: `NesterovATestTaskMPI`
- Number of processes: `<Number of processes>`
- Data distribution: `<How T2 data is distributed between processes>`
- Communication pattern: `<Point-to-point or collective communication pattern>`
- Collective operations, if used: `<MPI_Bcast / MPI_Reduce / MPI_Gather / other collectives>`
- Result gathering: `<How final T2 results are collected>`
- Synchronization points: `<MPI synchronization points>`
- MPI calls to explain: `MPI_Comm_rank`, `MPI_Barrier`
- Methods to describe: `ValidationImpl`, `PreProcessingImpl`, `RunImpl`, `PostProcessingImpl`
- Limitations: `<Known limitations>`

3.10 Process Task T3 Report

Fill in this report for the third process-based task. Compare `mpi/report.md` only with `seq/report.md` from this same `t3` directory.

Child reports:

- `seq/report.md`
- `mpi/report.md`

3.10.1 Problem Statement

- What must be implemented: `<T3 required computation>`
- Input data: `<T3 input data>`
- Output data: `<T3 output data>`
- Constraints: `<T3 constraints>`
- Correctness criteria: `<T3 correctness criteria>`
- Expected behavior: `<T3 expected behavior>`

3.10.2 Algorithm Description

- General idea of the algorithm: `<T3 algorithm overview>`
- Sequential algorithm: `<T3 sequential baseline>`
- Parallel decomposition: `<How T3 work is split between processes>`
- Data partitioning: `<How T3 data is divided>`
- Synchronization strategy: `<MPI synchronization strategy>`
- Result aggregation: `<How T3 partial results are combined>`
- Possible data races and how they are avoided: `N/A for separate MPI process memory unless threads are used`
- Complexity, if applicable: `<T3 time and memory complexity>`

3.10.3 Correctness Verification

- Implemented tests: `<T3 tests>`
- Normal cases: `<T3 normal cases>`
- Edge cases: `<T3 edge cases>`
- Invalid input cases, if applicable: `<T3 invalid input cases>`
- Comparison with sequential implementation: `<How T3 MPI output is compared with T3 sequential output>`
- Floating-point tolerance, if applicable: `<Tolerance value and reason>`
- Correctness explanation: `<How T3 correctness was checked>`
- Test input data: `data/pic.ppm` is loaded in functional tests through `stb_image`
- Functional test class to mention: `NesterovARunFuncTestsProcesses3`
- Performance test class to mention: `ExampleRunPerfTestProcesses3`
- Implementation classes to describe: `NesterovATestTaskSEQ`, `NesterovATestTaskMPI`

Test case	Input size / parameters	Expected result	Actual result	Status
`<Test name>`	`<Input>`	`<Expected>`	`<Actual>`	`<Passed/Failed>`

3.10.4 Performance Results

Implementation	Input size	Processes	Threads	Time, ms	Notes
Sequential	`<N>`	1	1	`<Time>`	`<Notes>`
MPI	`<N>`	`<P>`	1	`<Time>`	`<Notes>`

3.10.5 Process Task T3 Sequential Implementation

Fill in this report with the sequential baseline for process task T3. Use it as the only baseline for `../mpi/report.md`.

- Source files: `<T3 sequential source files>`
- Source files to describe: `include/ops_seq.hpp`, `src/ops_seq.cpp`
- Short description: `<T3 sequential implementation description>`
- Main functions/classes: `<T3 sequential functions or classes>`
- Task class to describe: `NesterovATestTaskSEQ`
- Important implementation details: `<T3 sequential implementation details>`
- Methods to describe: `ValidationImpl`, `PreProcessingImpl`, `RunImpl`, `PostProcessingImpl`
- Input preparation: `<How input data is prepared for the baseline>`
- Output validation: `<How baseline output is validated>`
- Limitations: `<Known limitations>`

3.10.6 Process Task T3 MPI Implementation

Fill in this report with the MPI implementation for process task T3. Compare it only with `../seq/report.md`.

- Source files: `<T3 MPI source files>`
- Source files to describe: `include/ops_mpi.hpp`, `src/ops_mpi.cpp`
- Task class to describe: `NesterovATestTaskMPI`
- Number of processes: `<Number of processes>`
- Data distribution: `<How T3 data is distributed between processes>`
- Communication pattern: `<Point-to-point or collective communication pattern>`
- Collective operations, if used: `<MPI_Bcast / MPI_Reduce / MPI_Gather / other collectives>`
- Result gathering: `<How final T3 results are collected>`
- Synchronization points: `<MPI synchronization points>`
- MPI calls to explain: `MPI_Comm_rank`, `MPI_Barrier`
- Methods to describe: `ValidationImpl`, `PreProcessingImpl`, `RunImpl`, `PostProcessingImpl`
- Limitations: `<Known limitations>`