Department of Electrical and Computer Engineering
Chair for Data Processing
Technical University of Munich

High Performance Computing

Christoph, Alexander | Stümke, Daniel | Kiechle, Johannes
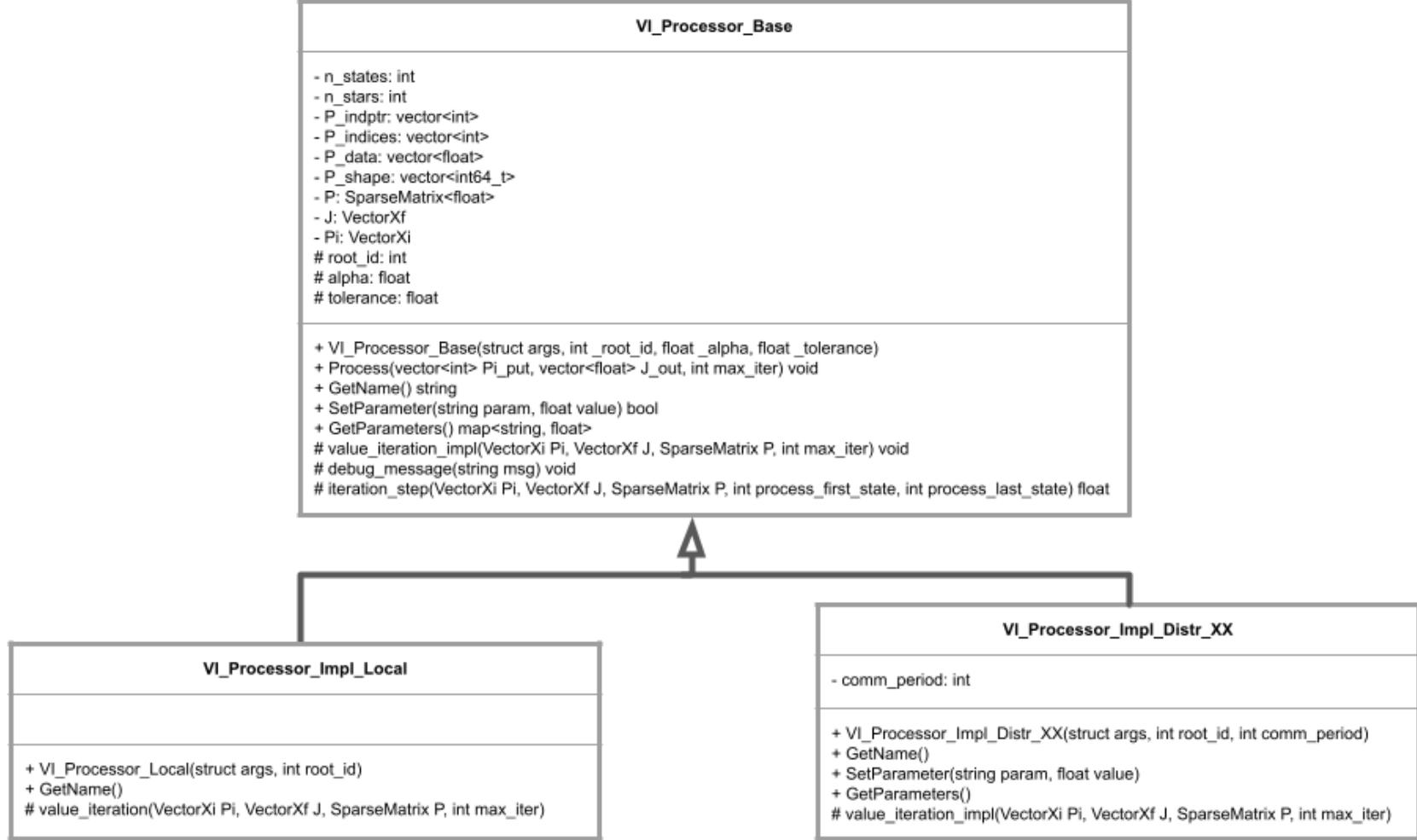
Department of Electrical and Computer Engineering, Technical University of Munich (TUM)

09.02.2021

structure of project

VI_Processor_Base

- n_states: int
- n_stars: int
- P_indptr: vector<int>
- P_indices: vector<int>
- P_data: vector<float>
- P_shape: vector<int64_t>
- P: SparseMatrix<float>
- J: VectorXf
- Pi: VectorXi
root_id: int
alpha: float
tolerance: float

+ VI_Processor_Base(struct args, int _root_id, float _alpha, float _tolerance)
+ Process(vector<int> Pi_put, vector<float> J_out, int max_iter) void
+ GetName() string
+ SetParameter(string param, float value) bool
+ GetParameters() map<string, float>
# value_iteration_impl(VectorXi Pi, VectorXf J, SparseMatrix P, int max_iter) void
# debug_message(string msg) void
# iteration_step(VectorXi Pi, VectorXf J, SparseMatrix P, int process_first_state, int process_last_state) float

VI_Processor_Impl_Local

+ VI_Processor_Local(struct args, int root_id)
+ GetName()
# value_iteration(VectorXi Pi, VectorXf J, SparseMatrix P, int max_iter)

VI_Processor_Impl_Distr_XX

- comm_period: int

+ VI_Processor_Impl_Distr_XX(struct args, int root_id, int comm_period)
+ GetName()
+ SetParameter(string param, float value)
+ GetParameters()
# value_iteration_impl(VectorXi Pi, VectorXf J, SparseMatrix P, int max_iter)

Pipeline

Makefile

- Compiling
- Starting the scripts
    - run_debug
    - run_small
    - run_normal
- Looping
  (N = 2, 4)

main.cpp

Executing the Value
Iteration for all
implemented
communication schemes

benchmark_comm_period.py

Visualization:
Mean execution time over
communication period

(Comparison of all implementations)

benchmark_np.py

Visualization:
Mean execution time over the
different datasets with variable
number of processors
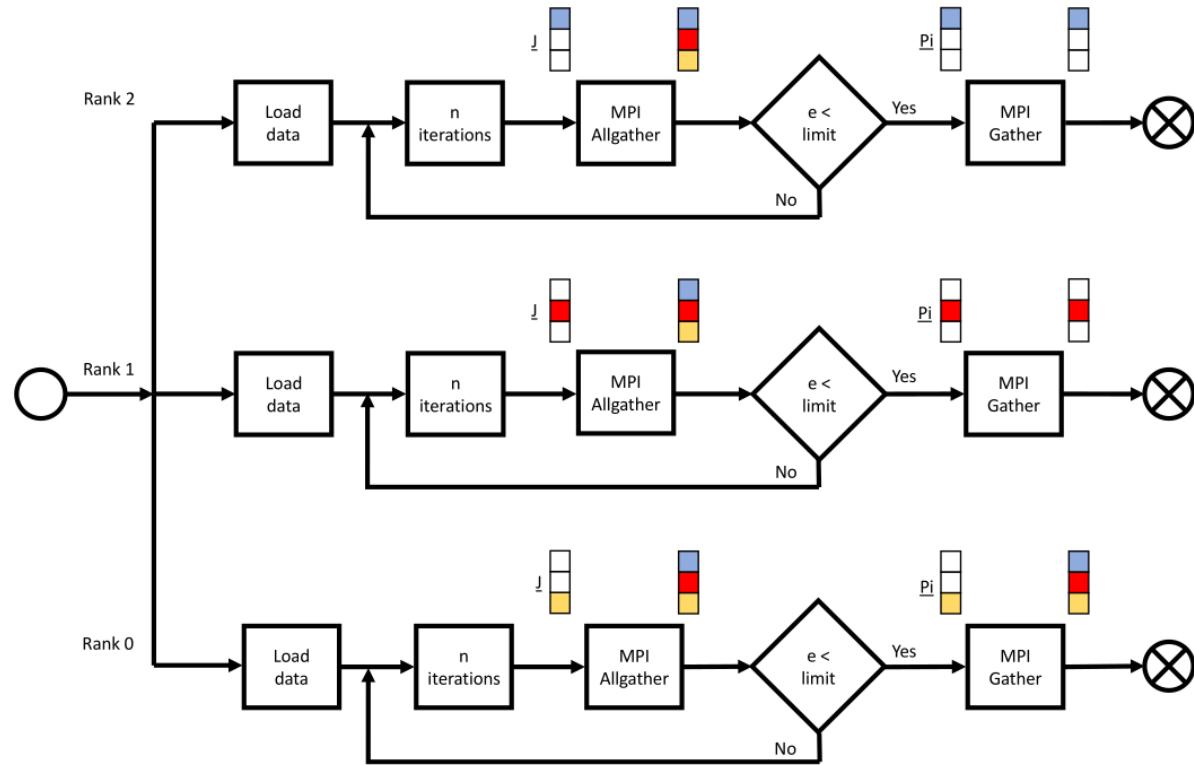
OpenMPI communication schemes

Used OpenMPI API's

- Allgatherv + Allreduce + Gatherv

- Send + Recv + Bcast

- Sendrecv + Gatherv

- Isend + Irecv + Ibcast + Igatherv

- Igatherv + Bcast + Gatherv
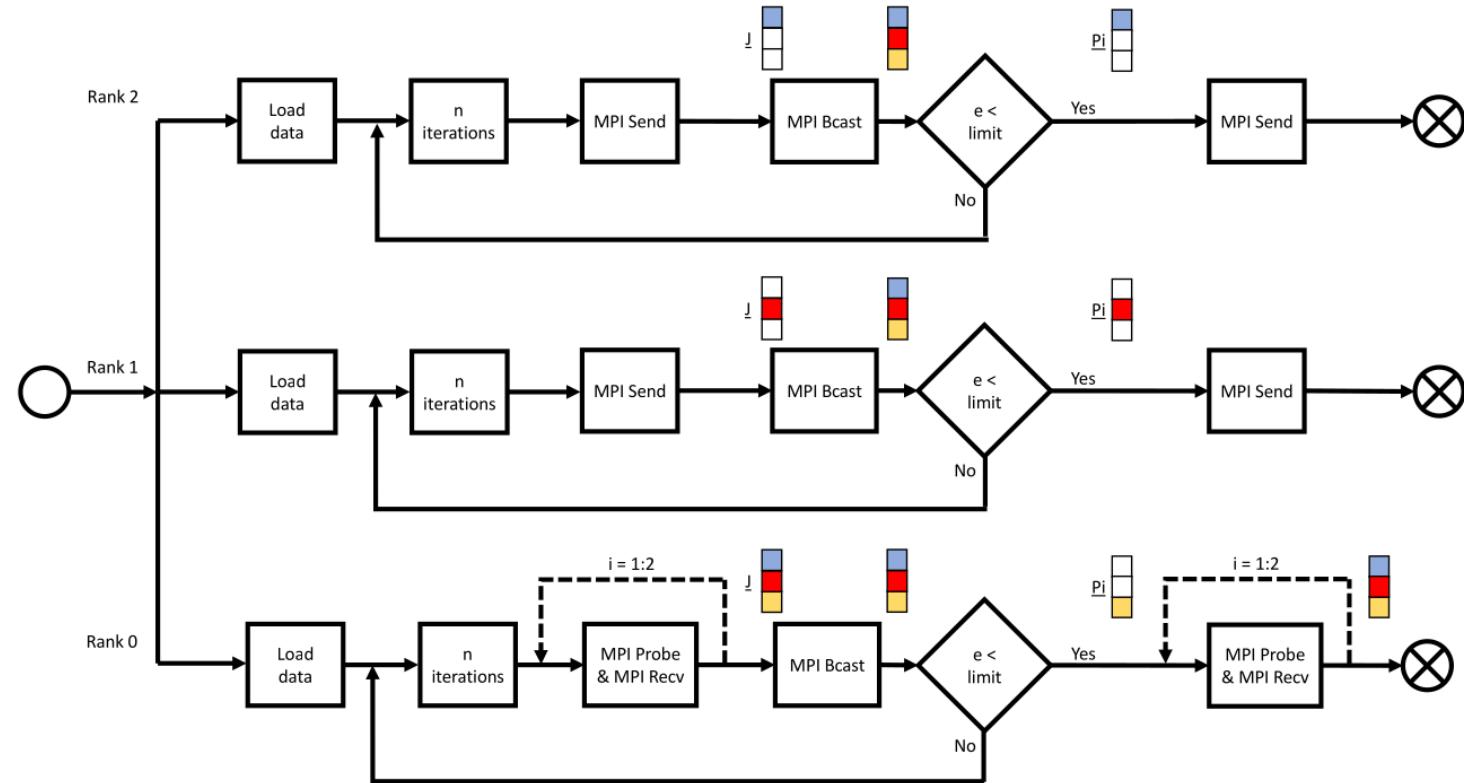
Scheme 1

- Allgatherv

- Allreduce
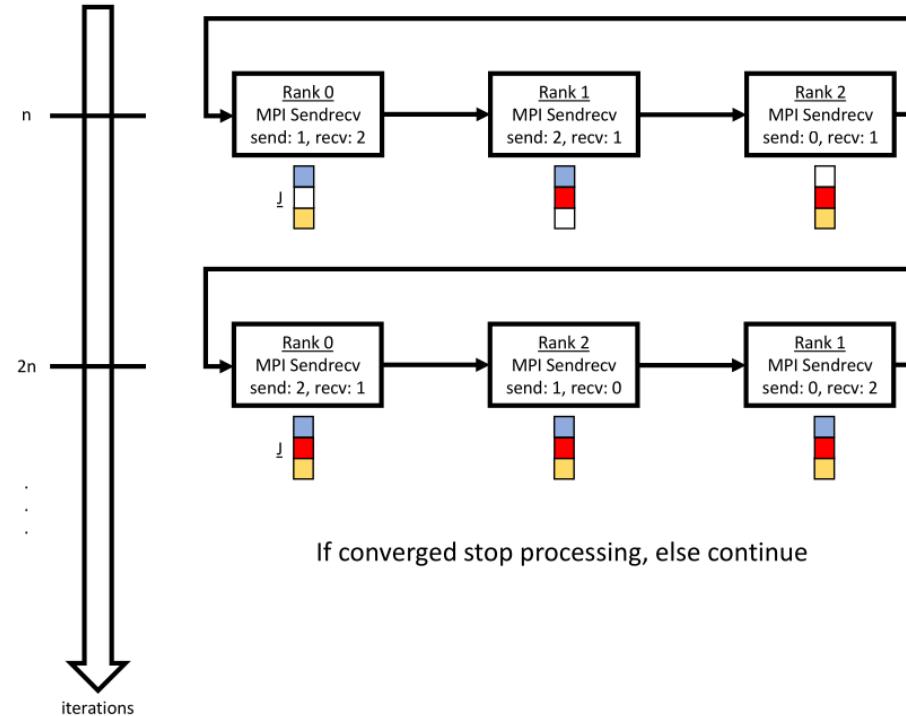
- Gatherv

Scheme 2

- Send

- Recv

- Bcast
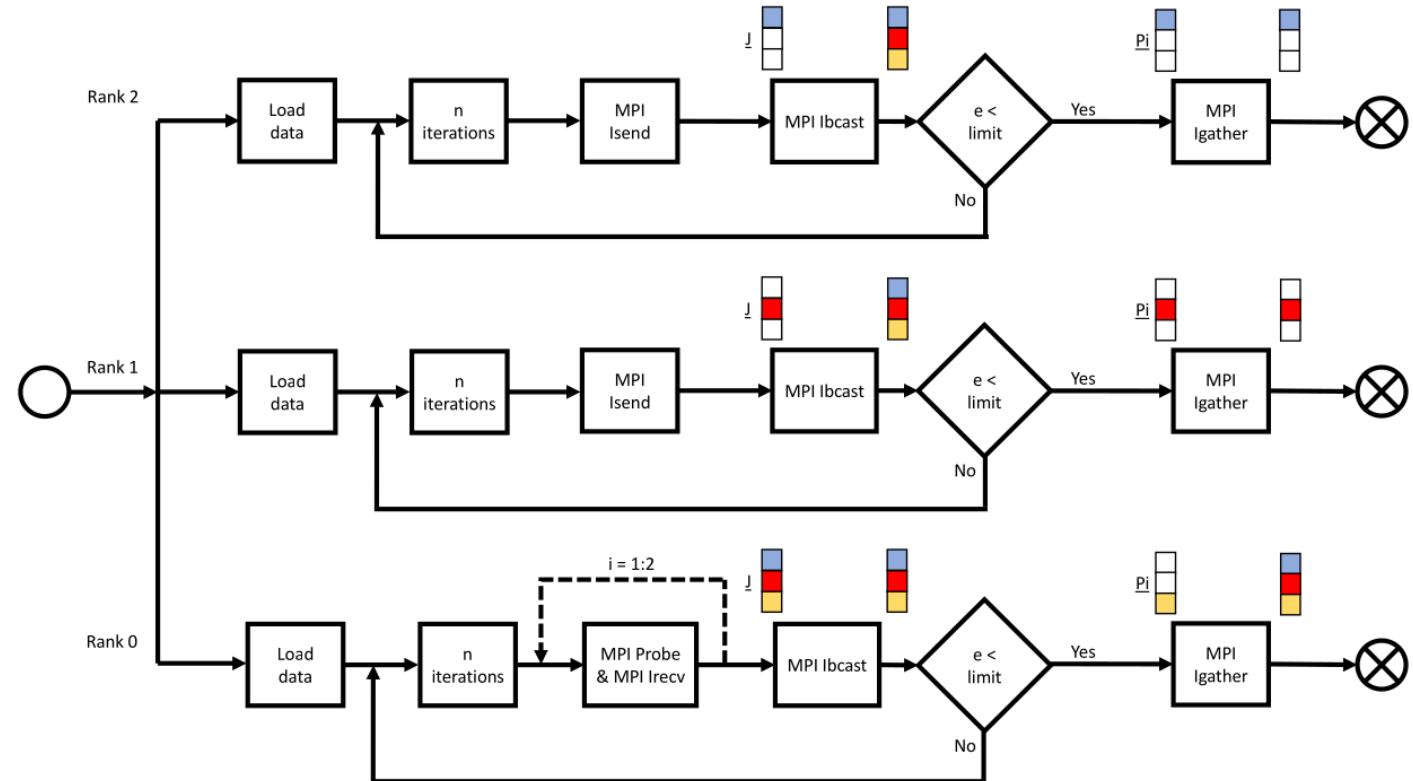
Scheme 3

- Sendrecv

- Gatherv
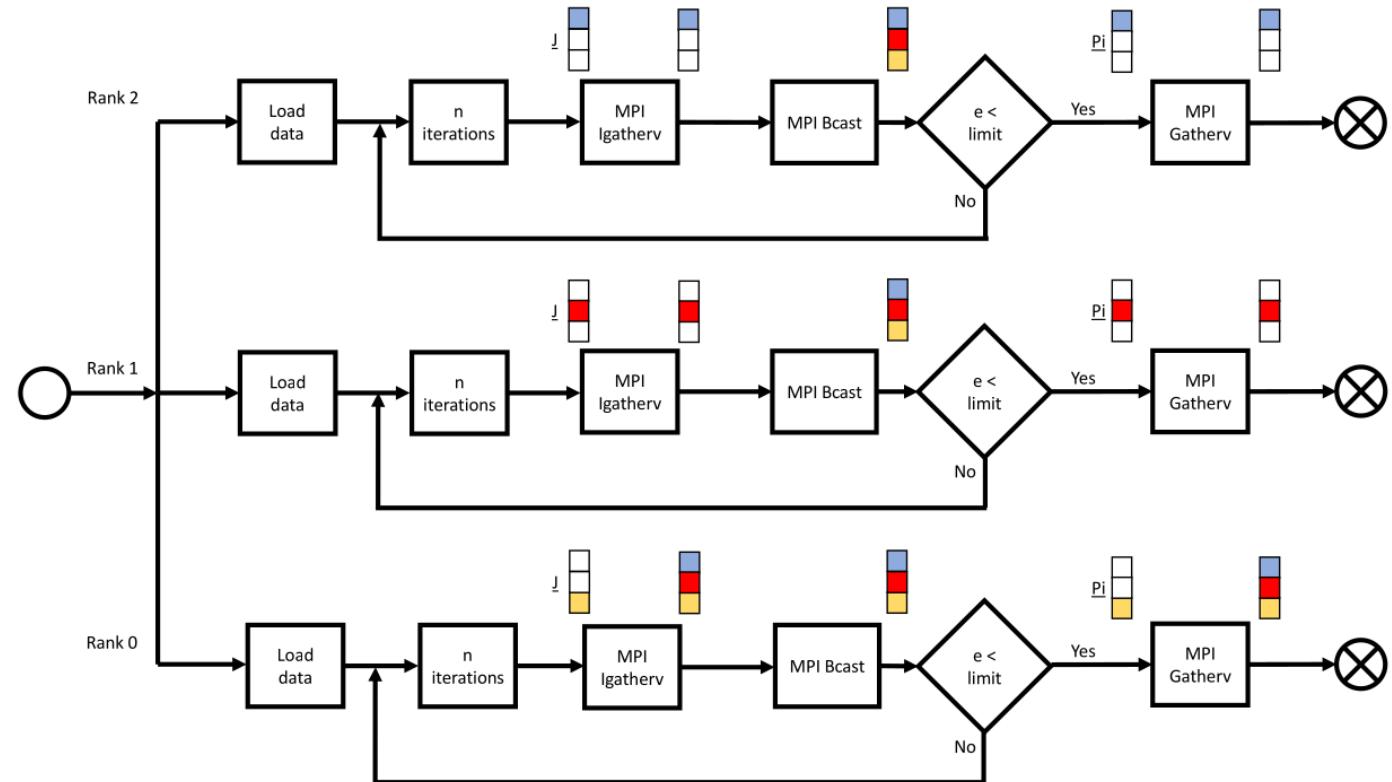
Scheme 4

- Isend

- Irecv

- Ibcast
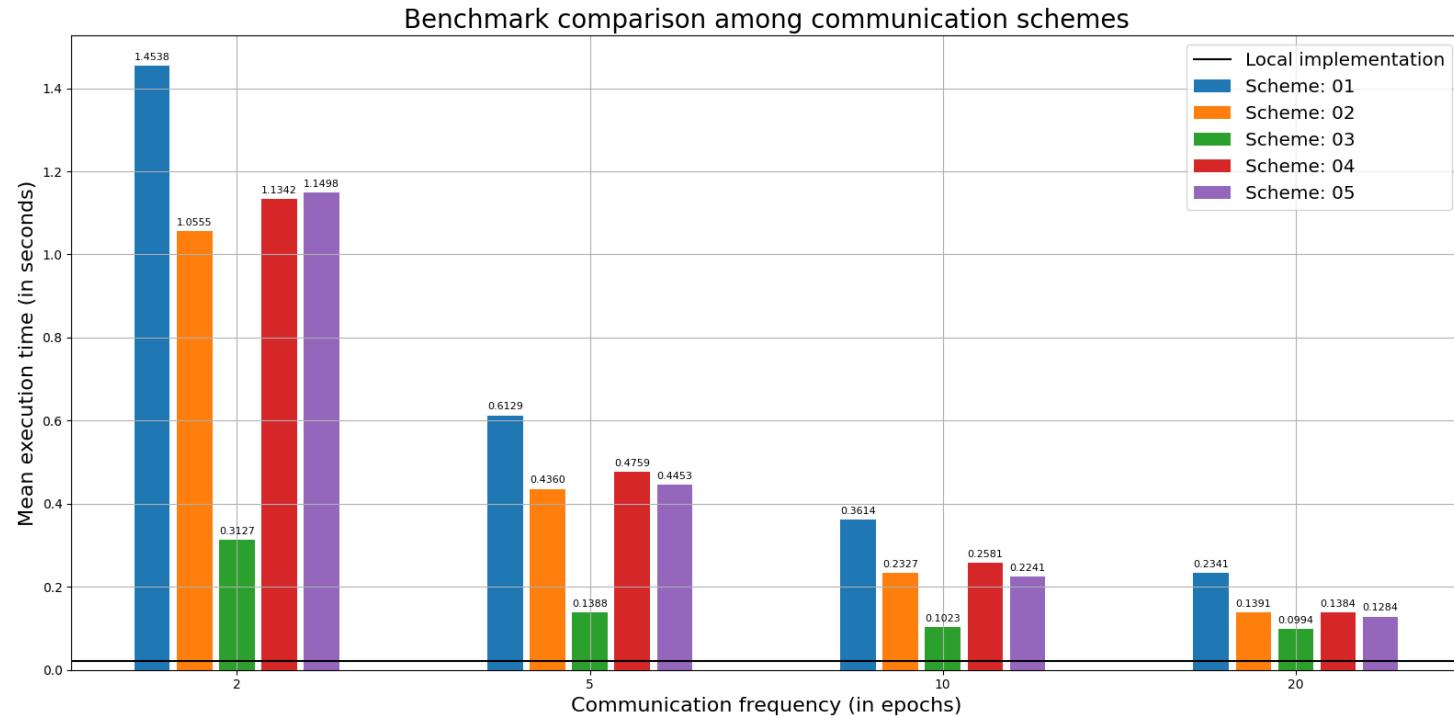
- Igatherv

Scheme 5

- Igatherv

- Bcast

- Gatherv
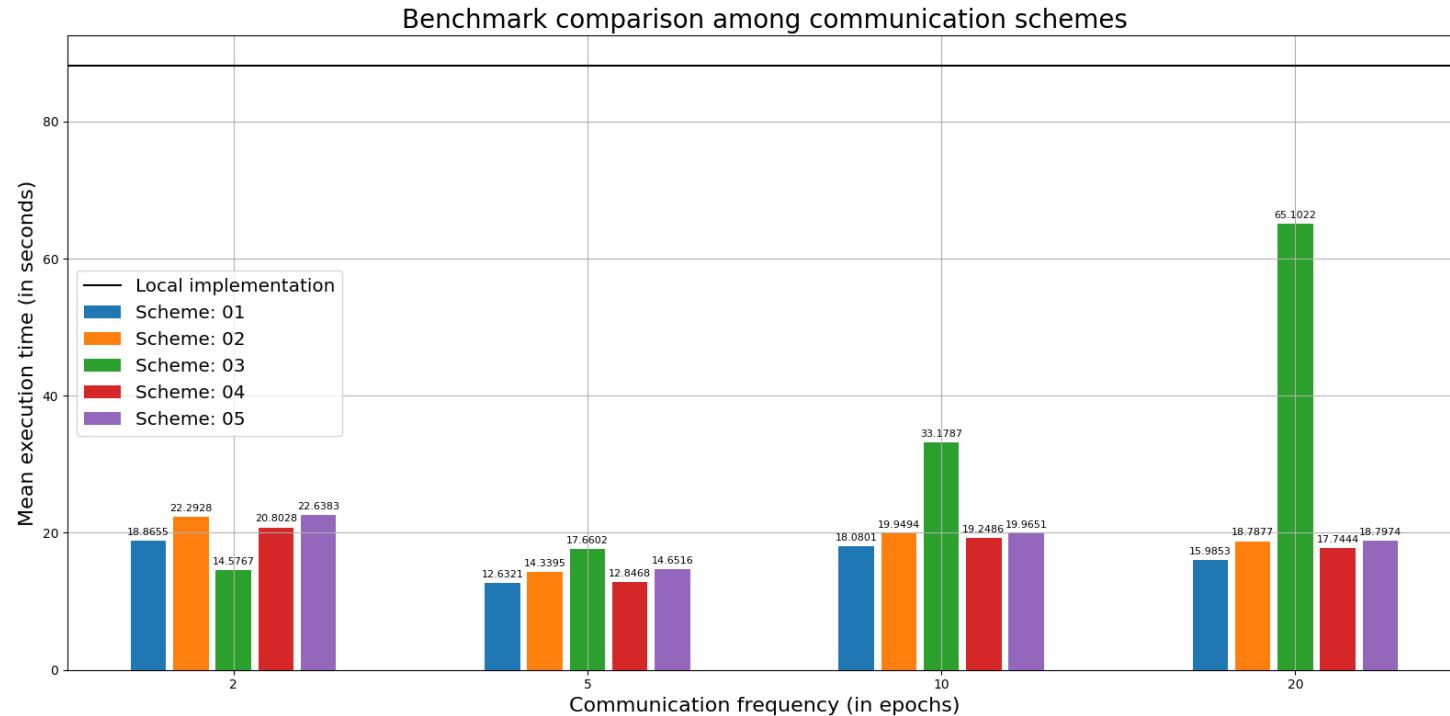
Results
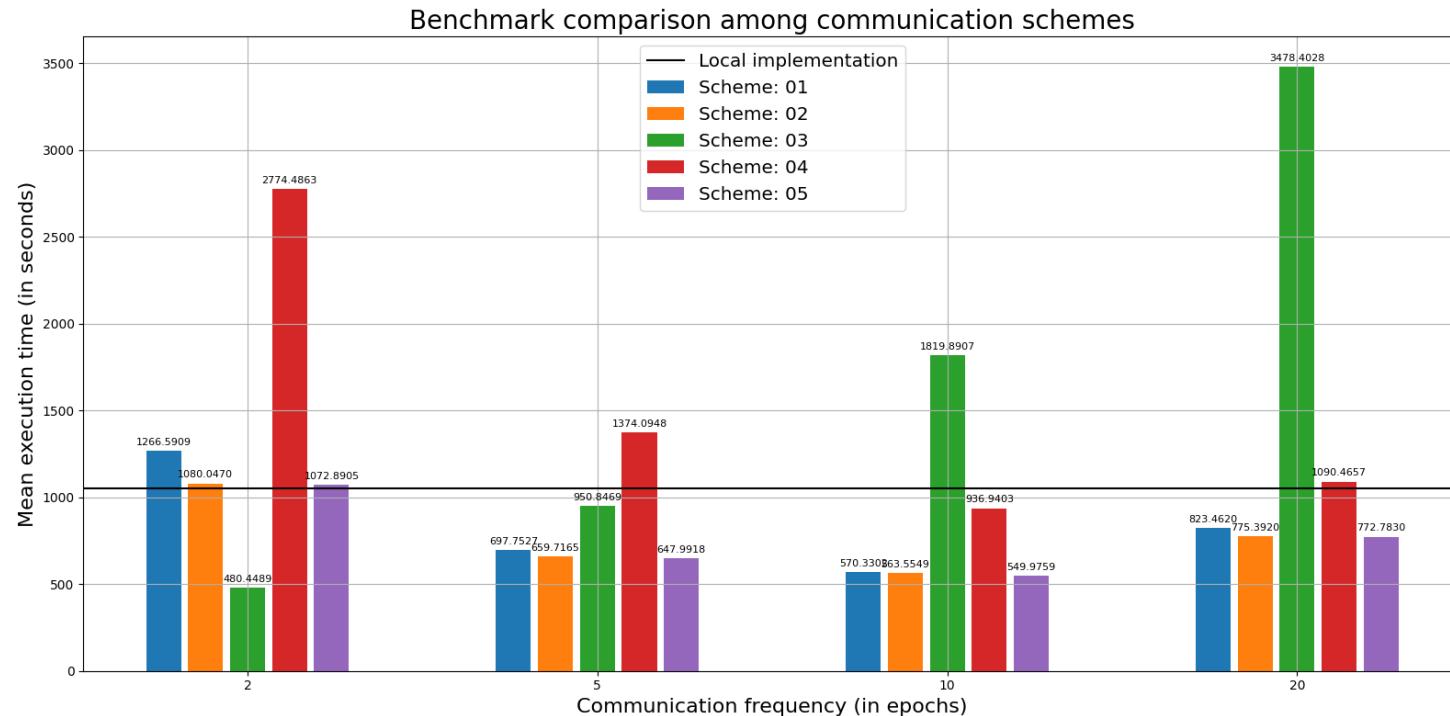
# Execution times (small dataset) - 8 processors, 2 slots

Benchmark comparison among communication schemes
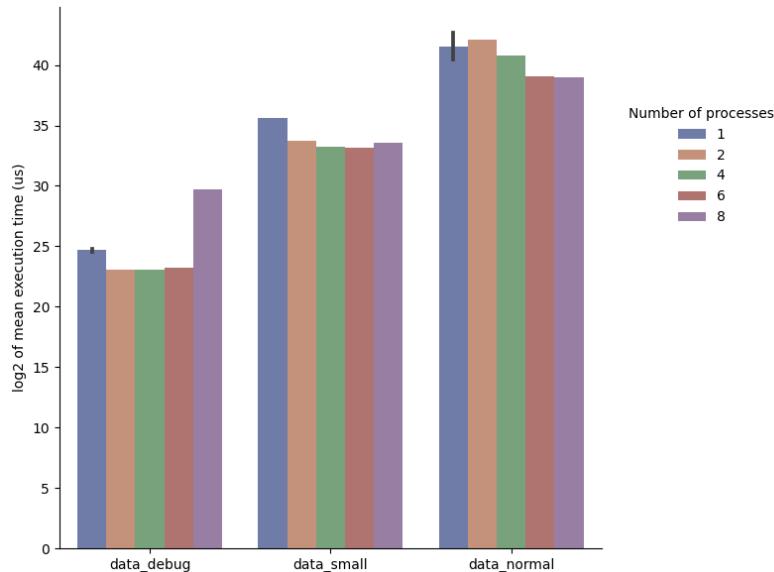
Execution times (medium dataset) - 8 processors, 2 slots

Benchmark comparison among communication schemes
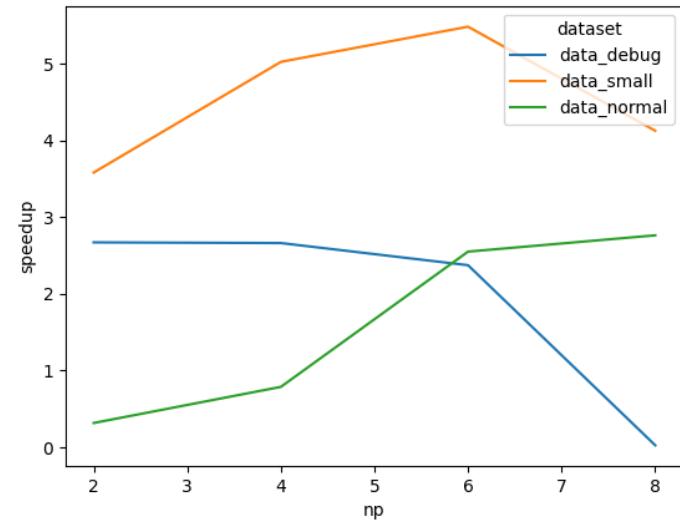
Execution times (big dataset) - 8 processors, 2 slots

Benchmark comparison among communication schemes

Overview results

Execution times for different datasets with varying number of processors (slots = 2)

relative improvement of execution time in dependency of numbers of processors (slots = 2)

Conclusion

Conclusion

- Use of OpenMPI beneficial especially for bigger datasets

- Synchronizing is not trivial for distributed algorithms

- No further speedup while using OpenMP in our attempts

- No usable C++ libraries for reading in .pickle files

Dataset size	# of processors	comm period
↓ *low*	↓ *low*	↗ *medium/high*
↓ *low*	↑ *high*	↑ *high*
↑ *high*	↓ *low*	↓ *low*
↑ *high*	↑ *high*	↘ *medium/low*

Table: recommended choice of parameters for different communication periods