Fakultät für Elektro- und Informationstechnik
Lehrstuhl für Datenverarbeitung
Technical University of Munich

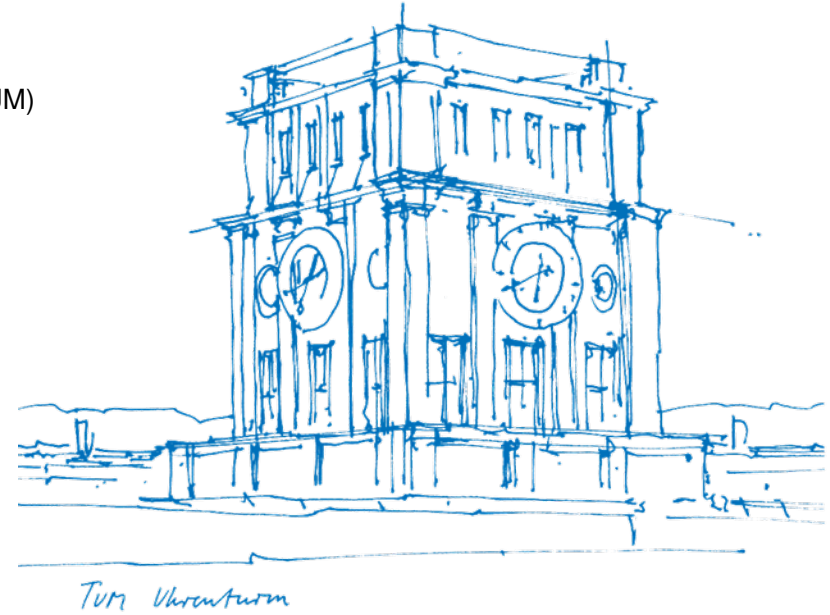
High Performance Computing für Maschinelle Intelligenz

Abschlussvortrag | Gruppe 1

Christoph, Alexander | Stümke, Daniel | Kiechle, Johannes

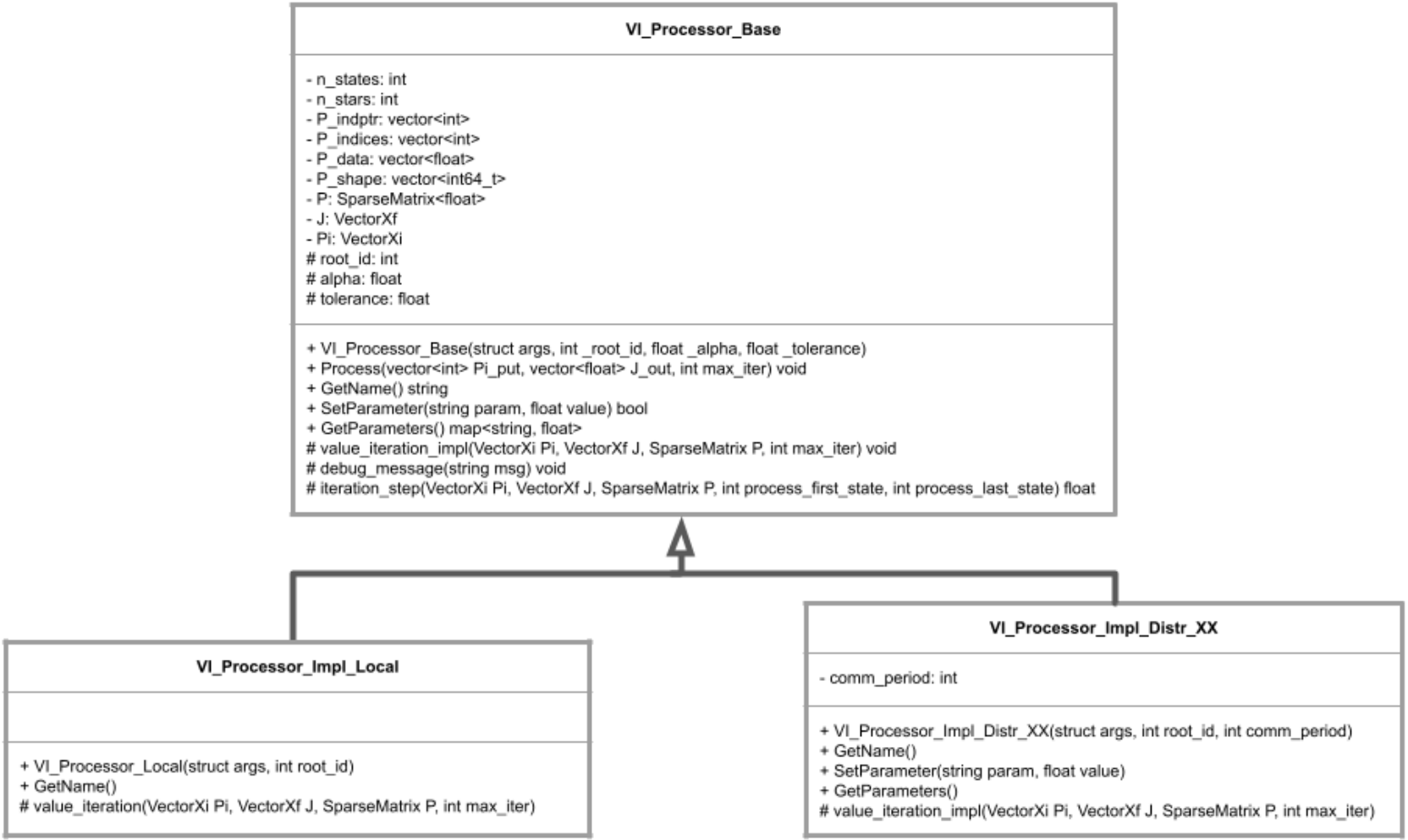
Fakultät für Elektro- und Informationstechnik, Technische Universität München (TUM)

09.02.2021



Inhaltsverzeichnis

Projekt Struktur


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

- Kompilieren
- Starten der Skripts
  - run_debug
  - run_small
  - run_normal
- Schleifendurchlauf
  (N = 2, 4, 6, 8)

main.cpp

Durchführung Value Iteration für alle Kommunikations-schemata

benchmark_comm_period.py

Visualisierung:
Mittlere Ausführungszeit über Kommunikationsfrequenz / Updatehäufigkeit

(Vergleich aller Implementationen)

benchmark_np.py

Visualisierung:
Mittlere Ausführungszeit über die verschiedenen Datensätze bei variabler Anzahl der Prozessoren

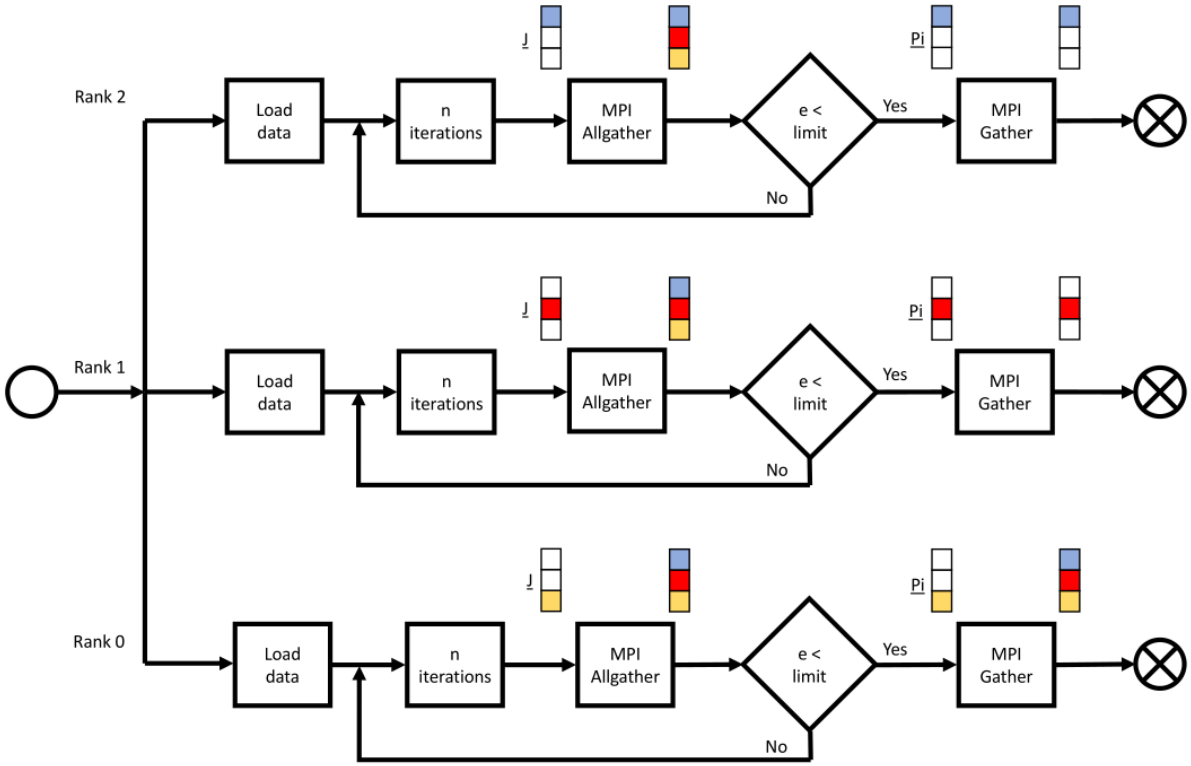
OpenMPI Kommunikationsschemata

Übersicht verwendeter OpenMPI API's

- Allgatherv + Allreduce + Gatherv
- Send + Recv + Bcast
- Sendrecv + Gatherv
- Isend + Irecv + Ibcast + Igatherv
- Igatherv + Bcast + Gatherv

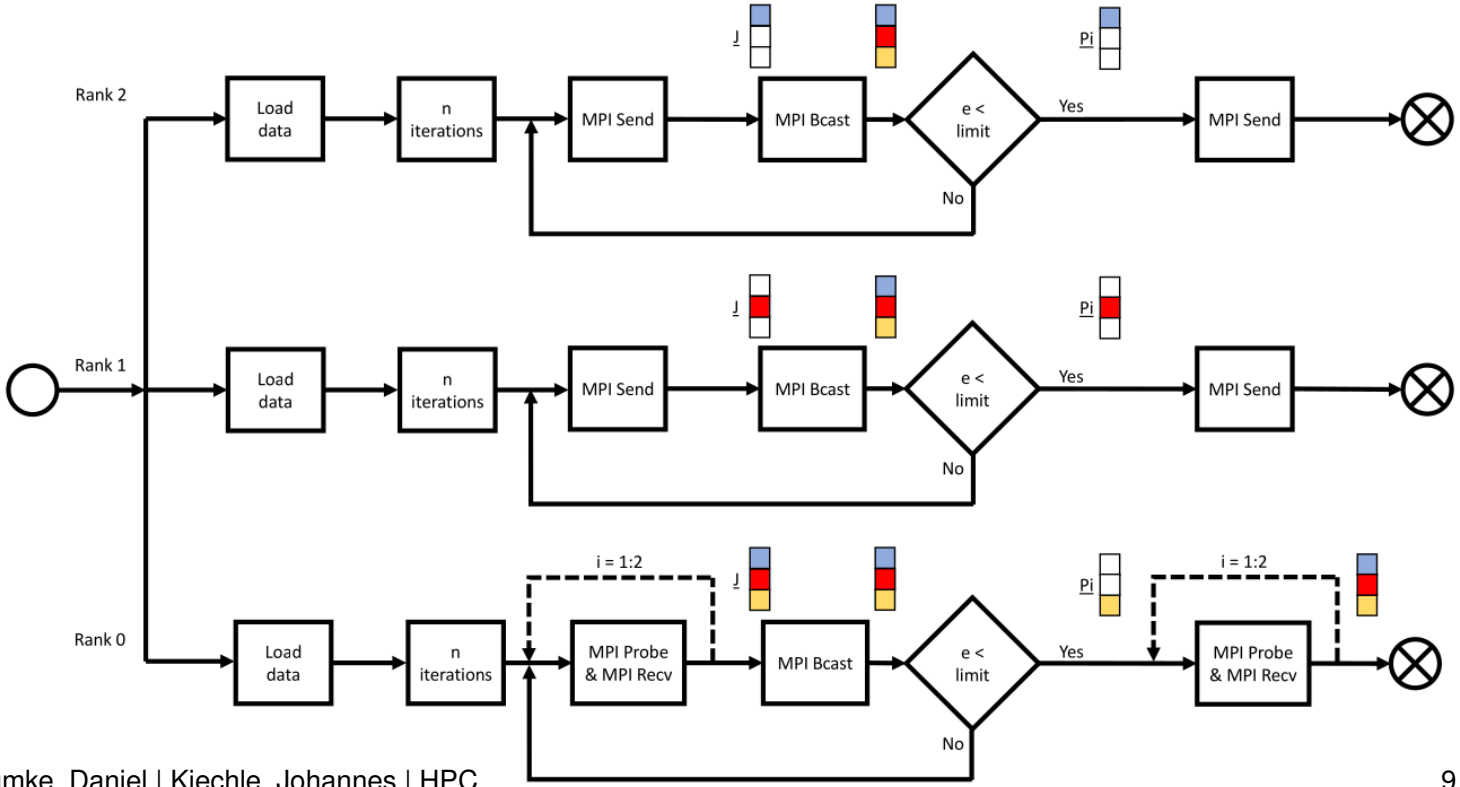
Schema 1

- Allgatherv
- Allreduce
- Gatherv



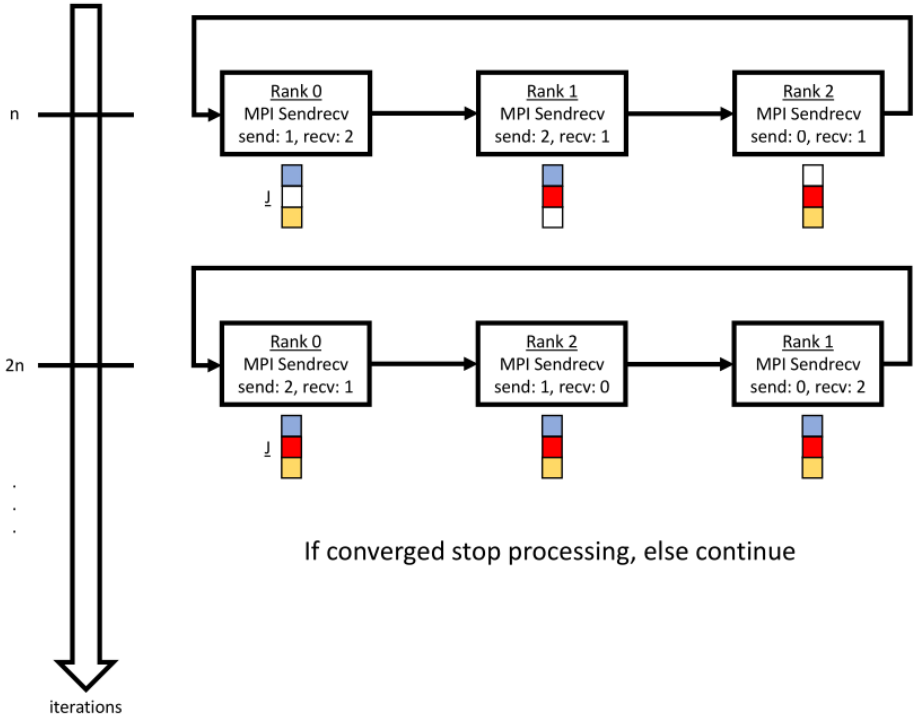
Schema 2

- Send
- Recv
- Bcast



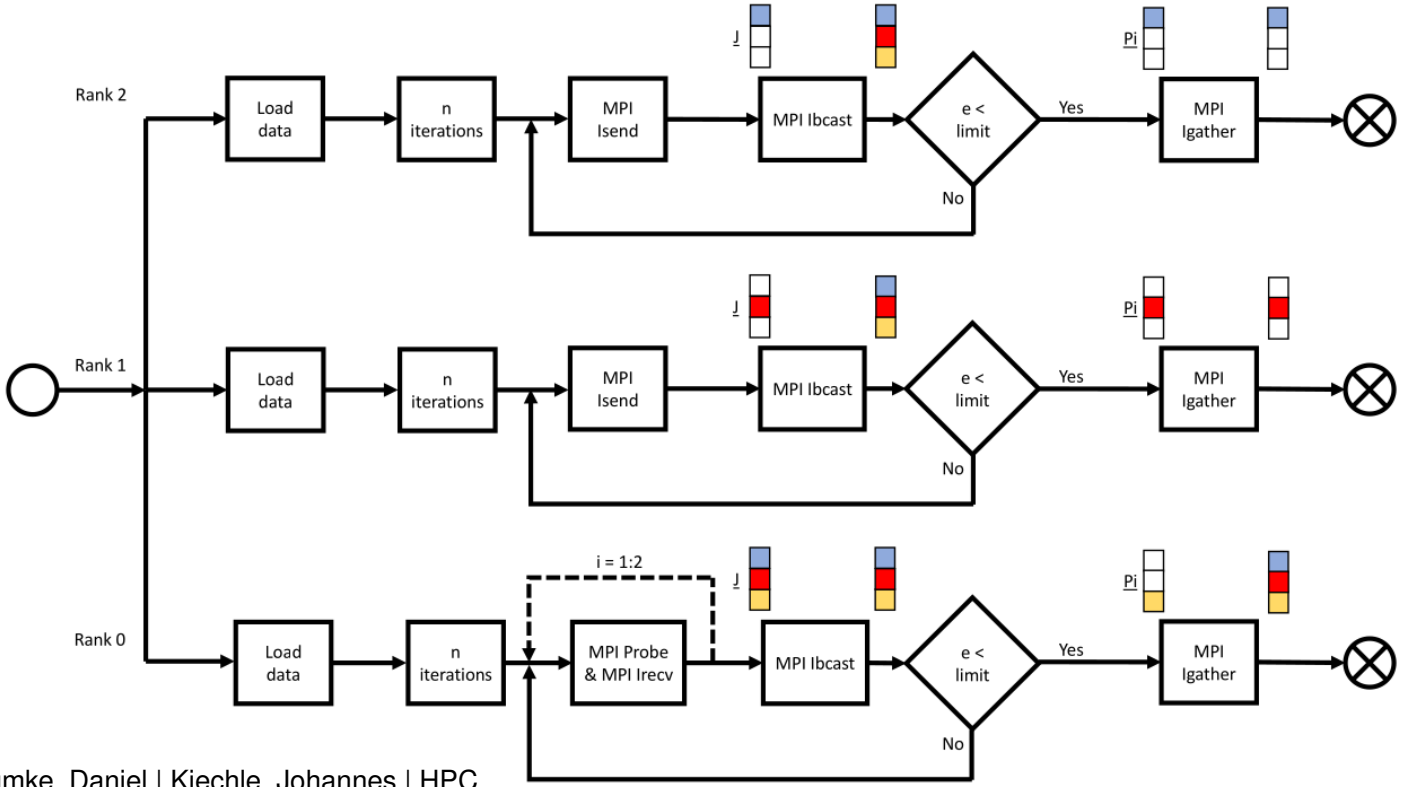
Schema 3

- Sendrecv
- Gatherv



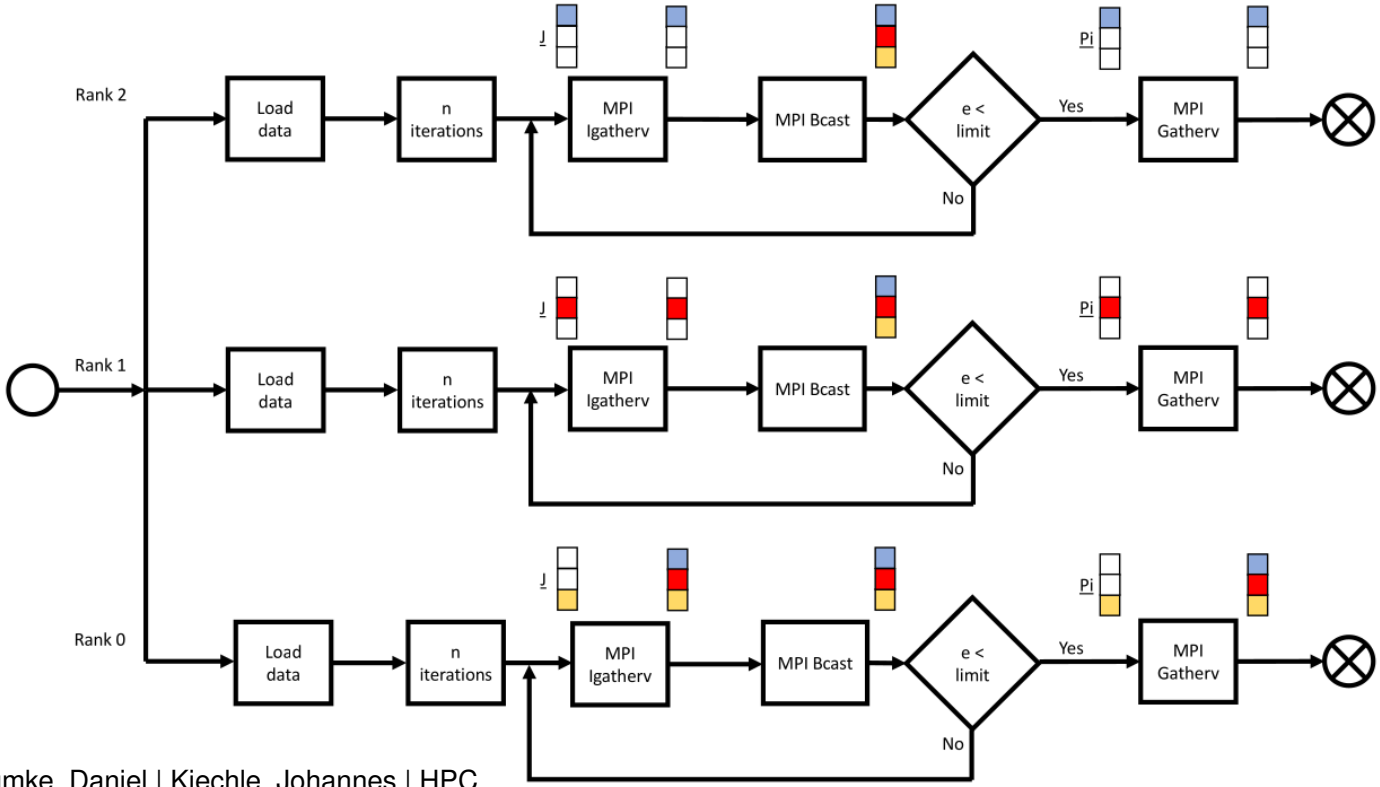
Schema 4

- Isend
- Irecv
- Ibcast
- Igatherv



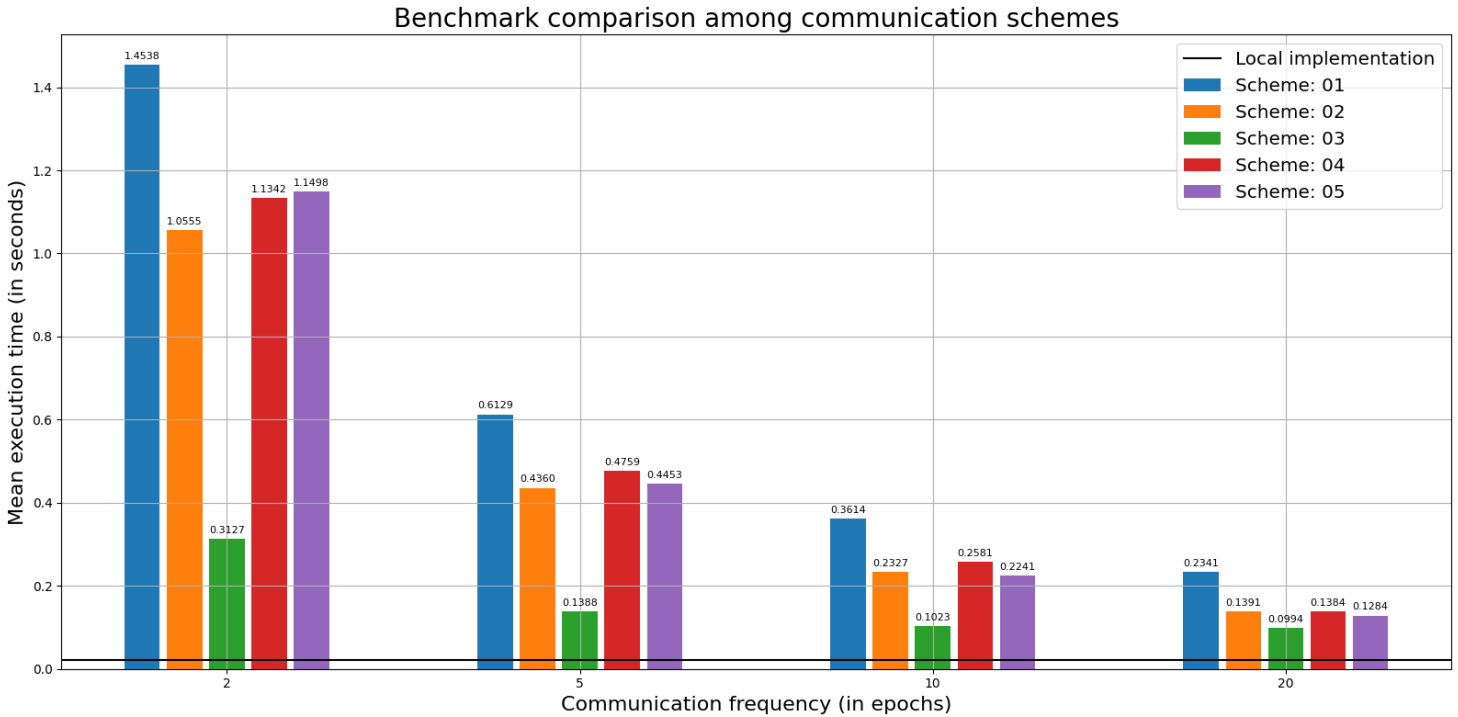
Schema 5

- Igatherv
- Bcast
- Gatherv

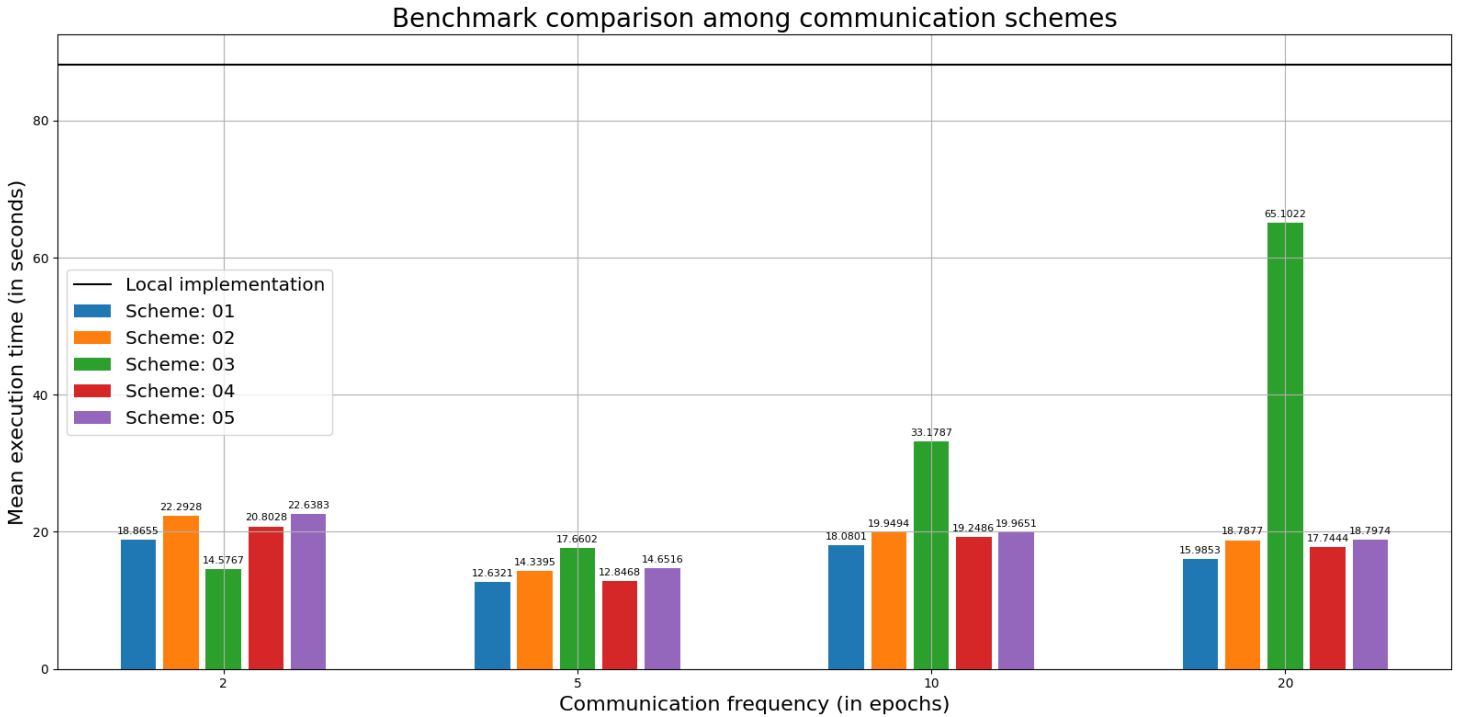


Ergebnisse

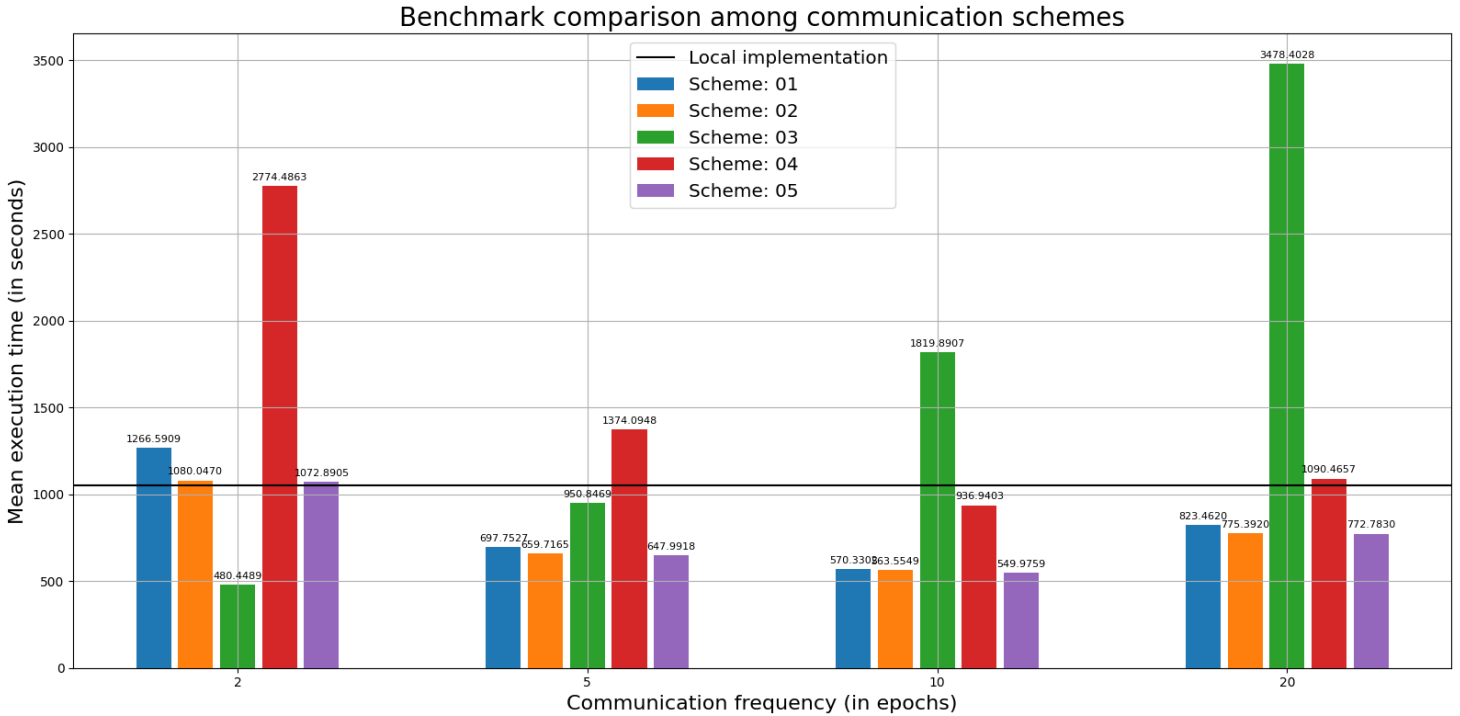
Ausführungszeit (kleiner Datensatz) - 8 Prozessoren, 2 Slots



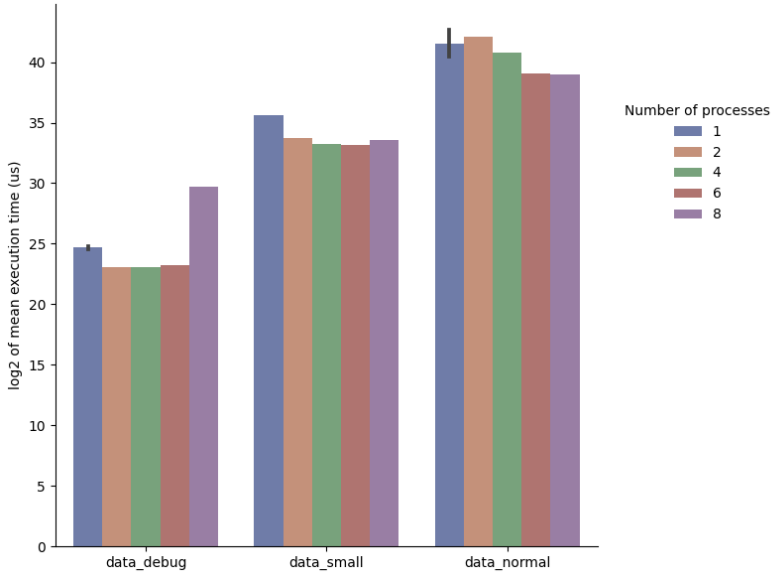
Ausführungszeit (mittlerer Datensatz) - 8 Prozessoren, 2 Slots



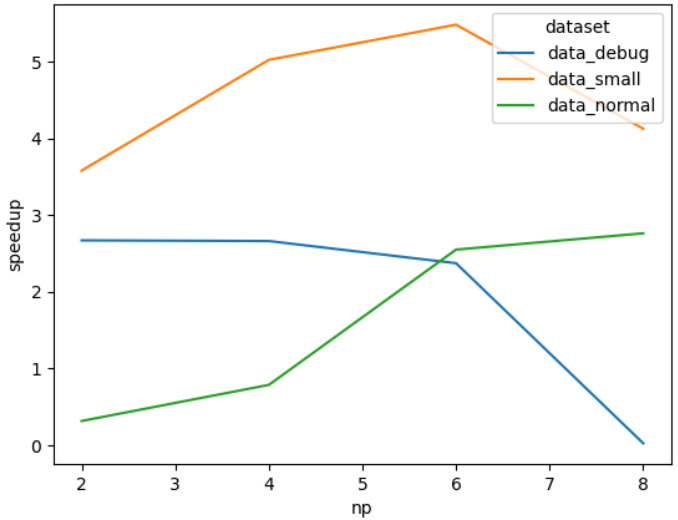
Ausführungszeit (großer Datensatz) - 8 Prozessoren, 2 Slots



Gesamtübersicht der Ergebnisse



Ausführungszeiten bei verschiedenen Datensätzen und variierender Anzahl von Prozessoren (Slots = 2)



Relative Verbesserung der Ausführungszeit in Abhängigkeit der Anzahl von Prozessoren (Slots = 2)

Fazit

Fazit

- Einsatz von OpenMPI bei größeren Datensätzen vorteilhaft
- Synchronisation ist beim verteilten Rechnen keine triviale Aufgabe (Stichpunkt: Konvergenzerkennung des VI-Algorithmus)
- Zusätzliche Verwendung von OpenMP brachte bei unseren Versuchen keine Geschwindigkeitsvorteile
- Schwankungen festgestellt, wenn andere Nutzer auf Eikon Rechnern Berechnungen durchführen
- Für das einlesen von Python Pickle-Dateien existieren momentan keine guten C++ Bibliotheken

Dataset size	# of processors	comm period
$\downarrow$ *low*	$\downarrow$ *low*	$\nearrow$ *medium/high*
$\downarrow$ *low*	$\uparrow$ *high*	$\uparrow$ *high*
$\uparrow$ *high*	$\downarrow$ *low*	$\downarrow$ *low*
$\uparrow$ *high*	$\uparrow$ *high*	$\searrow$ *medium/low*

Table: Unsere Parameterempfehlung für die Kommunikationsperioden

Vielen Dank für eure Aufmerksamkeit!