SIFEL

IT4Innovations

Lukas Maly (lukas.maly@vsb.cz)

EU H2020 Centre of Excellence (CoE)

1 December 2018 – 30 November 2021

Grant Agreement No 824080

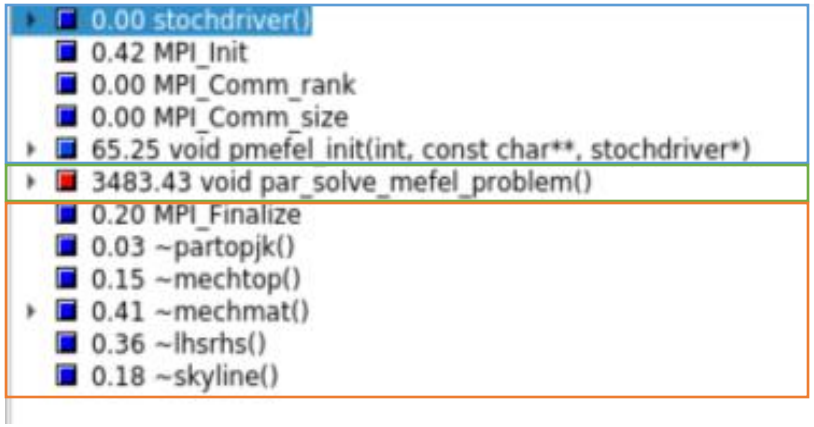
Background

- **Application Name:** SImple Finite Elements (SIFEL) (http://mech.fsv.cvut.cz/~sifel/index.html)
- **Developer:** Department of Mechanics, FCU at CTU Prague
- **Programming Language:** C/C++
- **Programming Models:** MPI
- **Applicant/User:** Jaroslav Kruis
- **Testcase:** Mechanical problems solved by the Schur complement approach, production runs up to 60 MPI ranks. (FEM)
- **Target:** Initial audit of the entire application, weak scaling, find hotspots,
- **Machine:** Barbora - 2x Intel Cascade Lake 6240, 18 cores, 2.6 GHz (up to 3.9 GHz); 2x Intel Skylake Gold 6126, 12 cores, 2.6 GHz (up to 3.7 GHz)
- **Tools:** Extrae 3.8.8 (Intel MPI 2018), Paraver 4.8.2, Clustering, Tracking

Application structure

- 3 phases: initialization, computation, finalization

Call tree

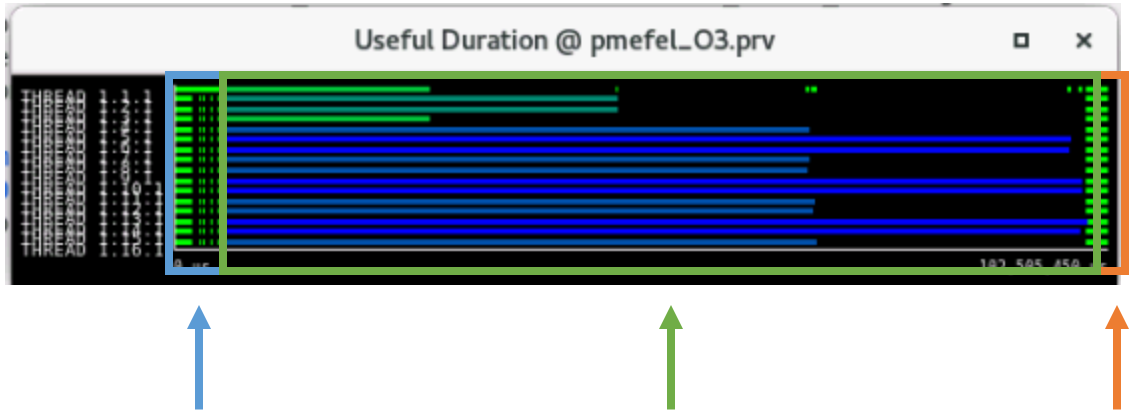


<u>Reference run</u> – <u>runtime ~100 seconds</u>

- 16 MPI ranks = 16 processes on one node
- No OpenMP parallelization
- Computation consists of
  - system matrix factorization (≥ 90%)
  - iterative solver – 319 iterations

<u>Production run</u>

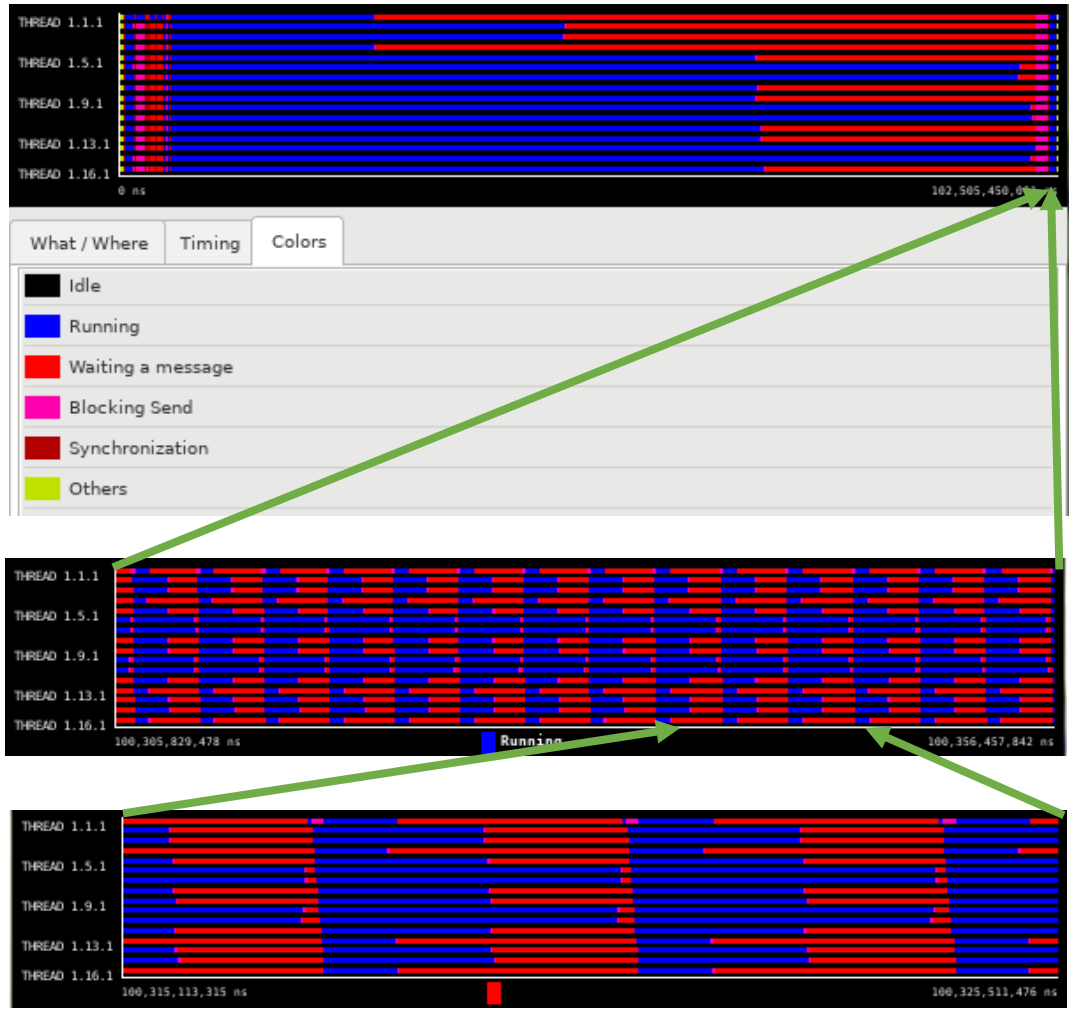
- Similar ratio of the factorization and iterations is expected (factor>70%)



- Focus of Analysis = computation phase

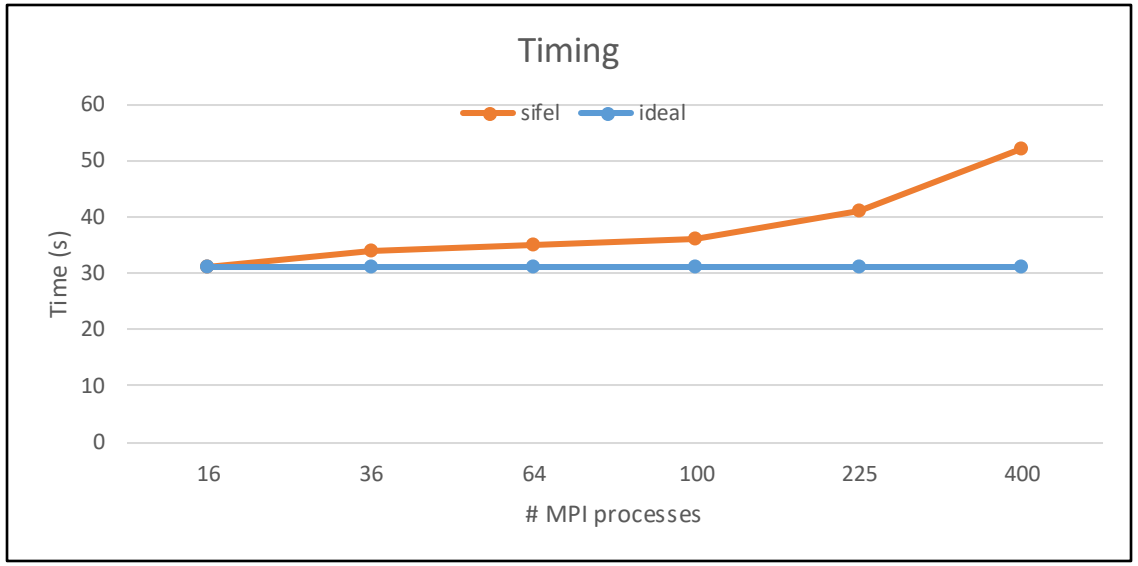
Application structure

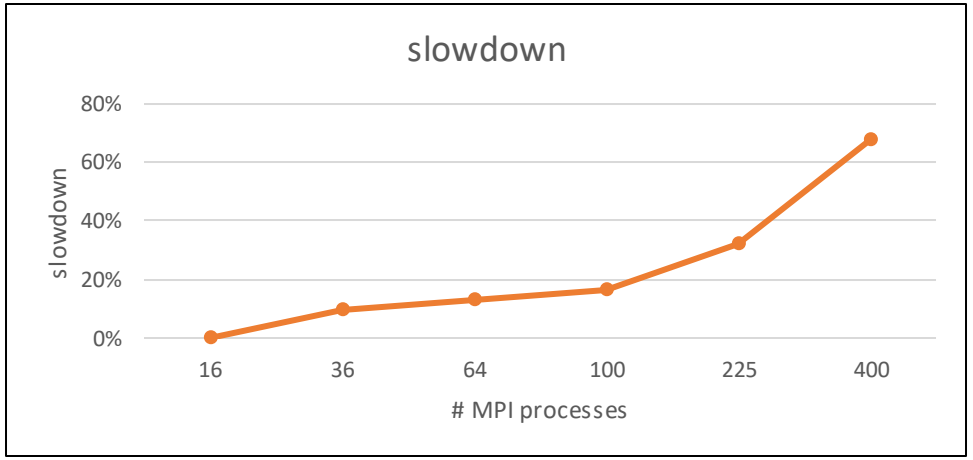
- Factorization = computation of the local Schur complements on each subdomain (MPI process)
  - Basic mathematical implementation
- ~300 iterations is only a tiny part of the runtime compared to the factorization
- Zoom into the iterative part
- Zoom into three iterations
  - They have similar unbalance pattern as the factorization step



Weak scalability

- Runtime of the FoA
  - Ideal situation is to have a constant time
- Slowdown of the FoA
- Detailed view for both components of the FoA in the next slide

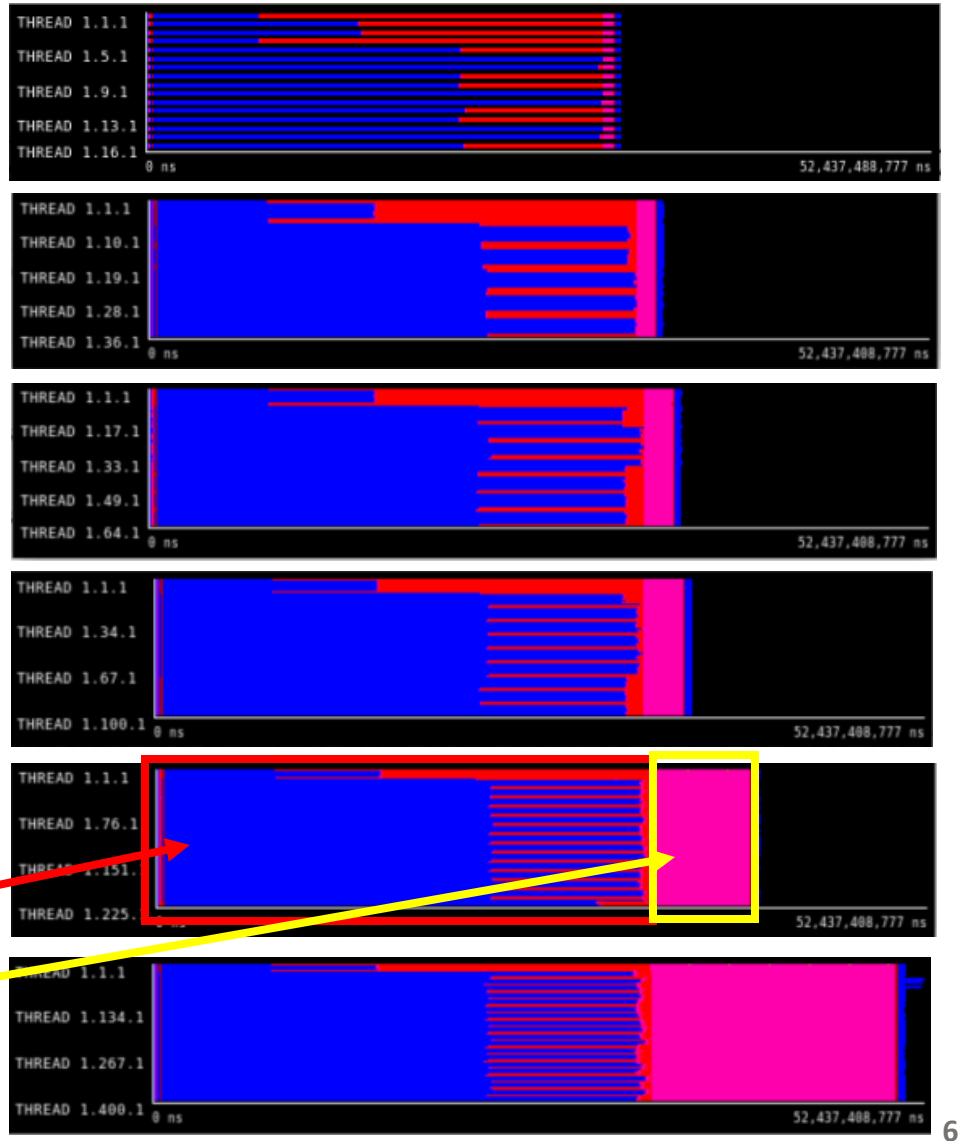




Weak Scalability

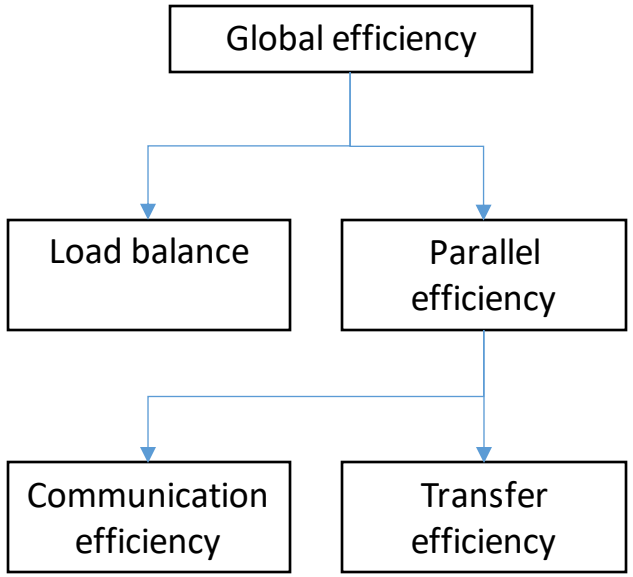
- Runtimes for different scales

- Factorization scales well
- Iterations cause delay and break the scalability
- Their number grows with scale
  - *341, 509, 738, 935, 1433, 1930*



POP efficiency metrics

- **Global efficiency** = parallel efficiency x computational scaling
- **Parallel efficiency** = average useful computation / runtime
  - Note that if a code is perfectly parallelised then runtime = average useful comp
  - Useful computation excludes time within MPI
- **Load balance** measures the absolute cost of the load imbalance
- **Communication efficiency** = maximum useful computation / runtime
  - If time overhead of MPI is zero, then runtime would equal maximum useful computation
- **Transfer efficiency** = ideal runtime / runtime
  - Ideal runtime is calculated assuming an infinite bandwidth and zero latency network
- **Serialisation efficiency** measures remaining cost of MPI
  - Due to process waiting idle for communication to start / end
  - It excludes any contribution to inefficiency captured in 'load balance' (see above)
- **Computational Scaling**
  - Sum of useful computation over all processes relative to a single node reference value
- **Scaling** of useful **instructions, IPC** (instructions per cycle) and **frequency**
- Note: a value of 100 represents ideal performance for all metrics
  - The values reduce as inefficiency increases



Multiplicative model

Weak Scalability (FoA)

- Three issues in the POP efficiency metrics
- We will try to explain their causes on the next slides

1. Load balance – starts low and gets better increasing the MPI processes, but the progress stops with the highest scale
2. Significant drops of the Comm. eff. from 100 to 400 MPI
   a) Serialization
   b) Transfer (significant drop from 225 to 400 MPI)

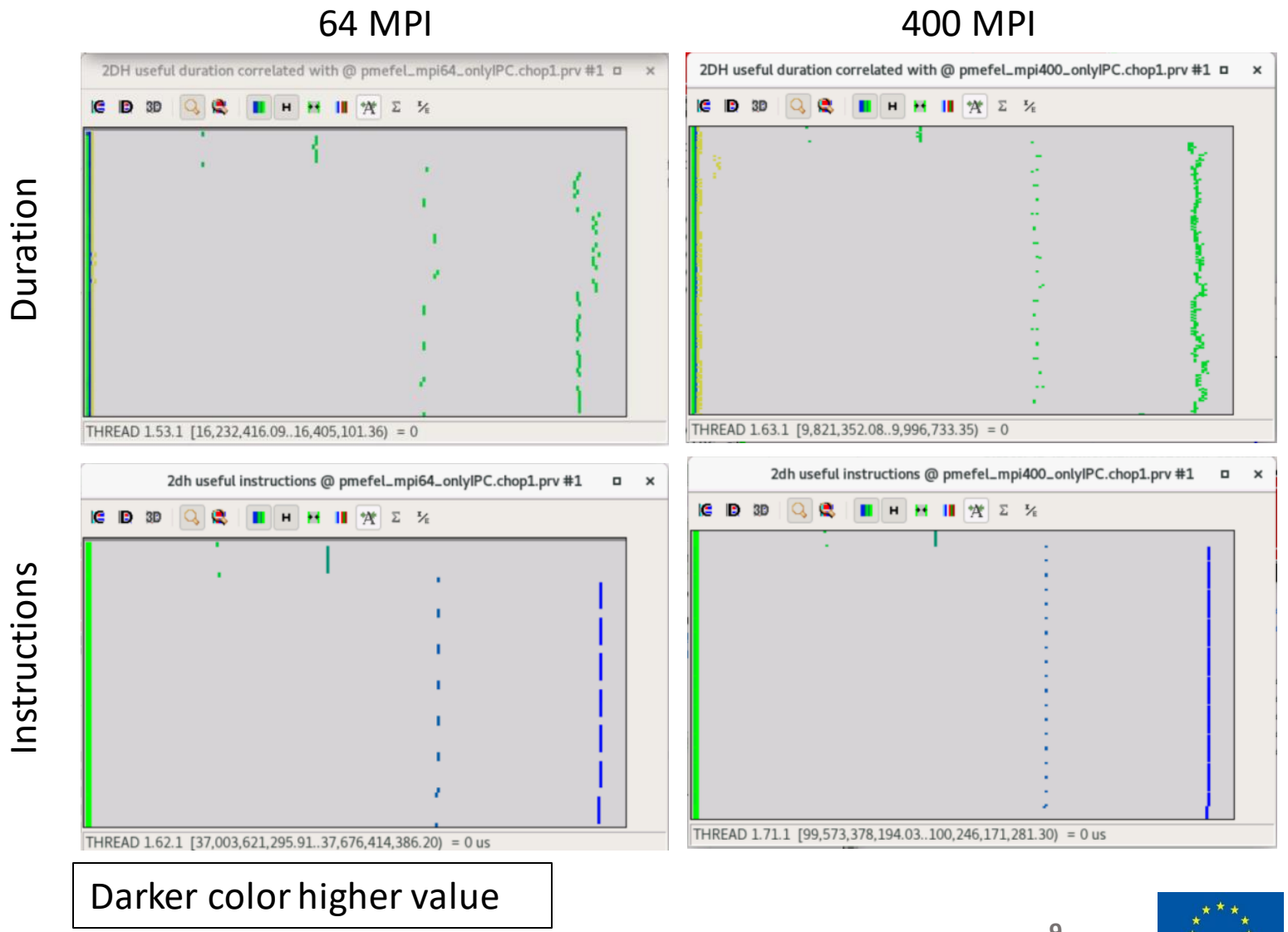
- Instruction scalability
  - Big drop for 16 to 36, then continues decreasing

MPI processes	16	36	64	100	225	400
- Parallel efficiency	72	80	82	84	81	67
-- Load balance	72	81	83	87	91	90
-- Communication eff.	100	99	99	97	90	75
-- Serialization eff.	100	100	99	98	93	84
-- Transfer eff.	100	100	100	99	96	89
- Computation scalab.	100	83	78	74	68	65
Global efficiency	72	66	64	62	57	44

MPI processes	16	36	64	100	225	400
IPC scalability	100	94	94	93	92	91
Instruction scalability	100	88	83	79	74	71
Frequency scalability	100	100	100	100	100	100

Load Balance

- Correlation of useful duration and number of instructions
- Some processes have less work to do -> imbalance
- LB efficiency is increasing since the avg value increases with more MPI processes
- Maximum value of instructions stays constant
- With more ranks the contribution of the lower ranks to the global unbalance is mitigated

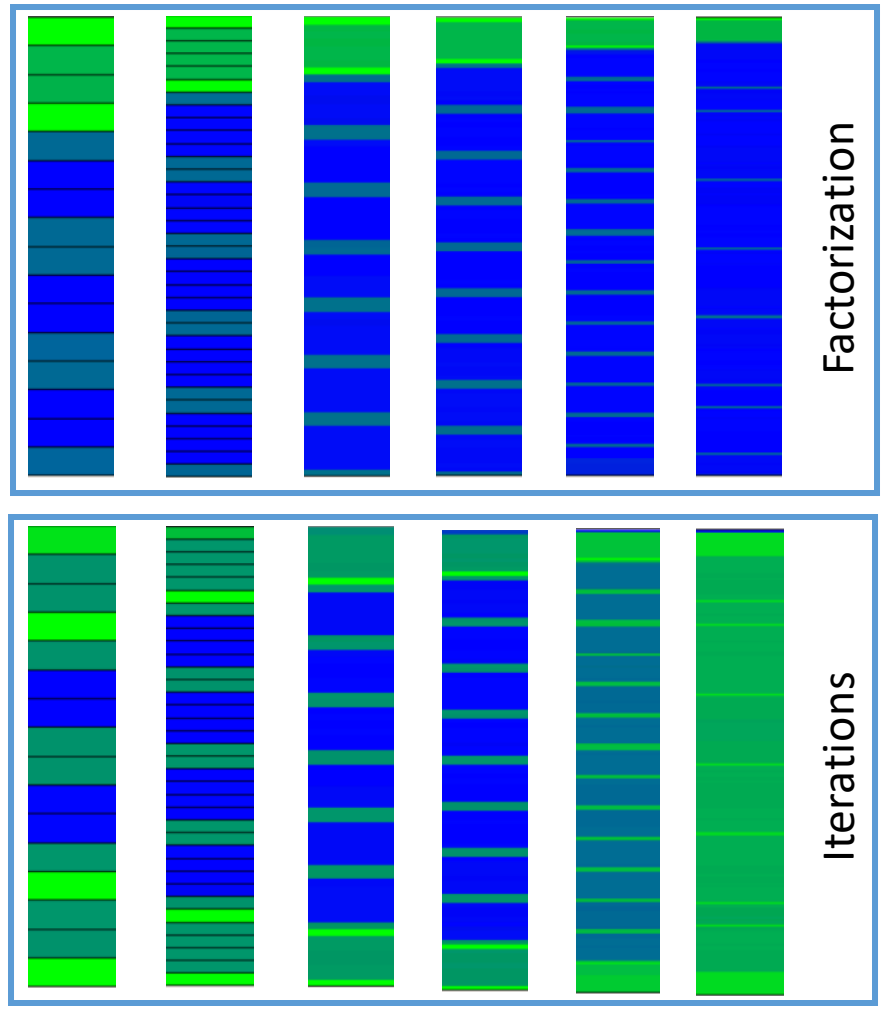


Darker color higher value

Load Balance

MPI processes	16	36	64	100	225	400
Factorization – LB	72	81	83	87	91	92
Iterations – LB	59	72	77	82	56	35

- Factorization
  - LB increases when the average value of the computational time over processes grows
- Iterations
  - For the beginning, the LB follows the same behavior as for the Factorization phase
  - With more processes, the master thread has more work to control them and thus the master spends much more time computing than the other processes



Darker color higher value

Communication

- MPI p2p communication
  - Master-slave approach
  - Msg sizes stay constant for weak scaling
- Factorization
  - Synchronization only at the end, perfect comm. efficiency
- Iteration phase
  - Serialization efficiency degrades
    - Dependency of slave processes on the master
  - Transfer eff. decreases
    - Higher percentage of time is spent in communication with more processes
    - More data need to be exchanged with master process while the computation is not sufficiently long

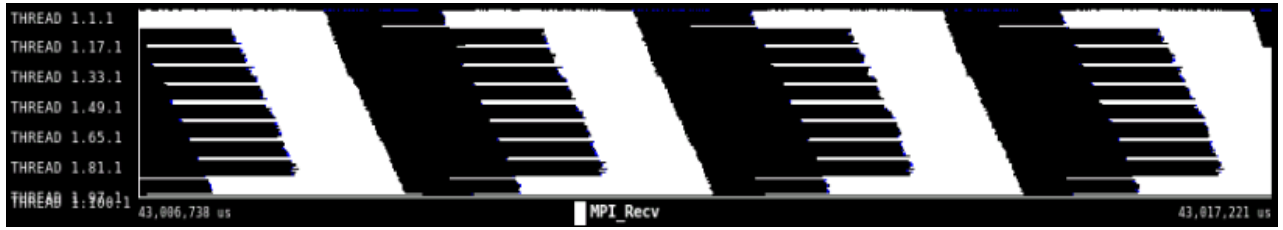
MPI processes	16	36	64	100	225	400
Factor – Comm. eff.	100	100	100	100	100	100
Iters – Comm. eff.	95	88	80	68	64	61
- Serialization eff.	97	91	85	76	83	97
- Transfer eff.	99	96	93	89	78	63

Strange improvement?
(? may be a bad trace-cut ?)

Iteration phase – avg time spent in MPI

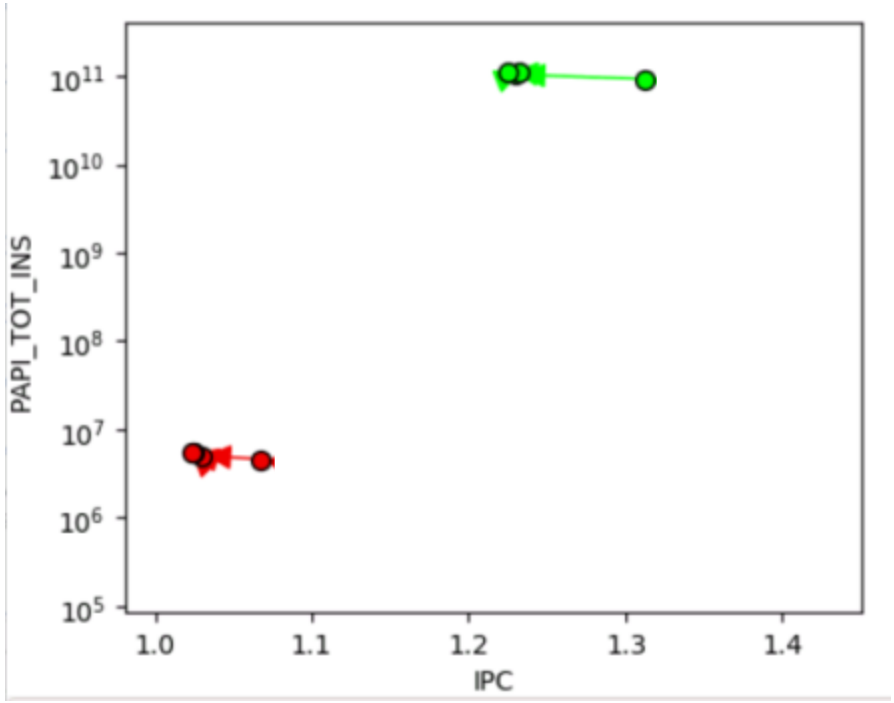
MPI processes	16	36	64	100	225	400
% Time in MPI calls	45	37	38	45	64	78

Iteration phase – serialized communication pattern



Serial Performance

- Cluster identification based on the IPC and instructions
- Tracking for 16,36,64,100 MPI processes
  - Green = factorization (higher IPC, more instructions)
  - Red = iterations (lower IPC, less instructions)
- IPC drops from 16 to 36 MPIs
  - Caused by node utilization (system has 36 cores/node)
  - MPI (nodes): 36 (1), 64 (2), 100 (3)
    - > 30 MPI per node: factorization IPC ~ 1.22
  - MPI (nodes): 16 (1), 64 (4)
    - 16 MPI per node: factorization IPC ~ 1.30
- IPC is stable from 36 to 400 MPIs



Instruction scaling

- Large drop from 16 to 36 processes, then the degradation is moreless constant
- Instruction scalability is computed from average number of instructions
  - This is affected by the load balance issue as the average value of instructions slowly grows for more processes due to increasing processes with higher number of instructions
- Factorization phase
  - Maximum number of instructions is constant
  - Instructions are perfectly scaled affected only by the load balance issue
- Iteration phase
  - Affected by the change of the number of iterations and by instruction scaling per iteration
    - Number of iterations grows
  - Instruction scaling per iteration
    - A significant drop from 16 to 36 processes, then continuing degradation

MPI processes	16	36	64	100	225	400
Instruction scalability	100	88	83	79	74	71
- factorization:	100	94	94	94	93	92
- iterations:	100	55	35	26	16	12
(# iterations)	*341*	*509*	*738*	*935*	*1433*	*1930*
- per iteration:	100	82	76	72	67	65

Summary and Recommendations

- SIFEL is most frequently run with less than 100 MPI processes
  - The most time-consuming part of the code is the factorization phase
  - Iterative phase perfomrs approximately 400 - 1000 iterations
- Running the application with more ranks does not pay off due to the increase of the number iterations
  - idea: the number of iterations may be possibly lowered by applying a preconditioner
- Each process works with different problem size – load balance issue
  - Fixing this issue will result in more balanced and faster factorization and individual iterations
- For the tested size of local problems, the most optimal runs with respect to the load balance in the iterative solver are those with 64 and 100 MPIs
- Transfer Efficiency – decreasing with increasing number of processes
  - More messages to and from the master process with a constant message size
  - consider using MPI collective operations such as Gather and Scatter instead of sequential Send and Receive calls in Round Robin fashion
- Serial performance seems to be stable with scale (except the lowest run)
- As the most time-consuming part is the factorization phase, we would advise the user to focus on its optimization.
  - One possible option may be to vectorize the code
  - Another option is to substitute this part of the code using some already optimized math library