Parallelization of the Structural Mechanics Solver b2000++pro: Assessment, Status and Future Strategy

Harald Klimach <harald.klimach@dlr.de>, Neda Ebrahimi Pour, Sabine Roller

DLR – SP

36th Workshop on Sustained Simulation Performance 2023, Sendai

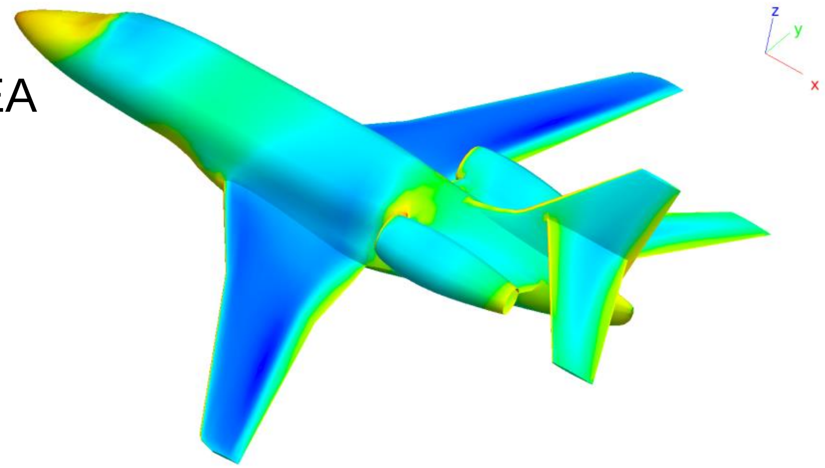
FEM Solver for Structural Analysis

General purpose structural solver b2000++ pro

- Solving various FE problems, with special focus on shell and composites in lightweight construction and buckling and post-buckling
- It is similar to:
  - Linear static and dynamic solvers of Nastran
  - Nonlinear static and dynamic solvers of ANSYS and Abaqus FEA

Application areas

- Linear and nonlinear structural mechanic problems
- Eigenvalue analysis
- Buckling analysis and vibration
- Damage analysis on laminates



High fidelity problems need to be solved → High demand for high performance computing (HPC)

FEM Solver for Structural Analysis

- Modern modular code design in C++ utilizing templates

- User manual available for new users with various examples
  - https://www.smr.ch/newdoc/b2000pp/b2user/html/index.html

- Plugin infrastructure with exchangable parts for user written code
  - User defined elements
  - User defined materials
  - User defined ‚solvers' for different problems

→ Large flexibility, enabling wide application

Existing Parallelism

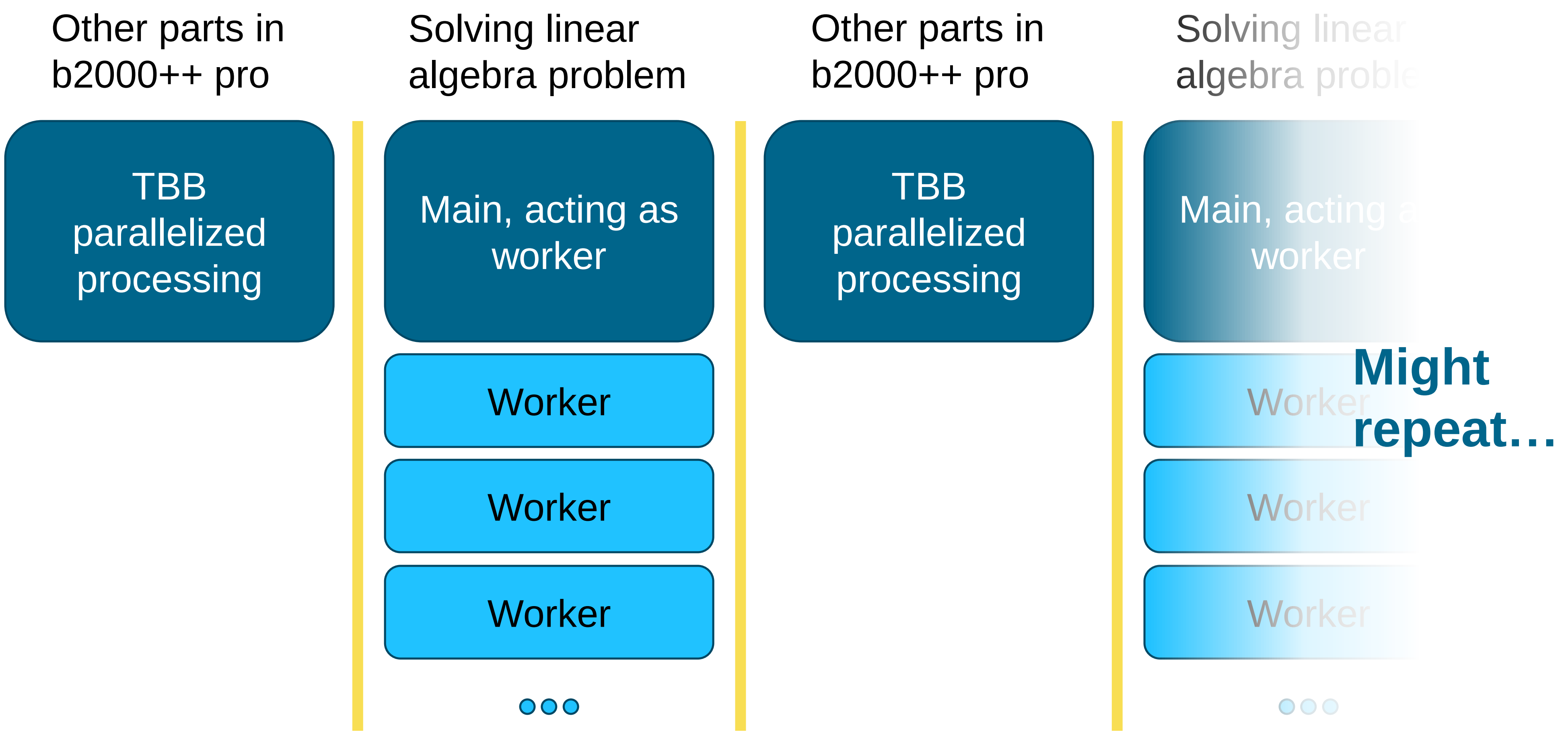
- Predominantly shared-memory parallelized with the help of Intels Threading Building Blocks (TBB)
- Distributed memory parallelism is only employed via the used linear algebra package (which itself can be hybrid parallel)

- Results in a Main / Worker concept with a single main process holding the overall problem, and workers for the solution of the linear algebra problems

- Consecutive, non-overlapping for both parts
  - in one only the main process works
  - during solution of linear algebra all processes (including the main one) are involved

Existing Parallelism

- Consecutive, non-overlapping time periods for both parts



Linear Algebra Packages

- Various Linear Algebra packages can be used, like
  - PastiX
  - Spliss
  - SuperLU

- However, the main tool is **MUMPS** (**MU**ltifrontal **M**assively **P**arallel sparse direct **S**olver)
  - Presented existing concept works with MUMPS as linear algebra solver
  - https://mumps-solver.org/index.php
  - Implemented in Fortran
  - Hybrid parallelism with OpenMP

Parallel Performance Assessment

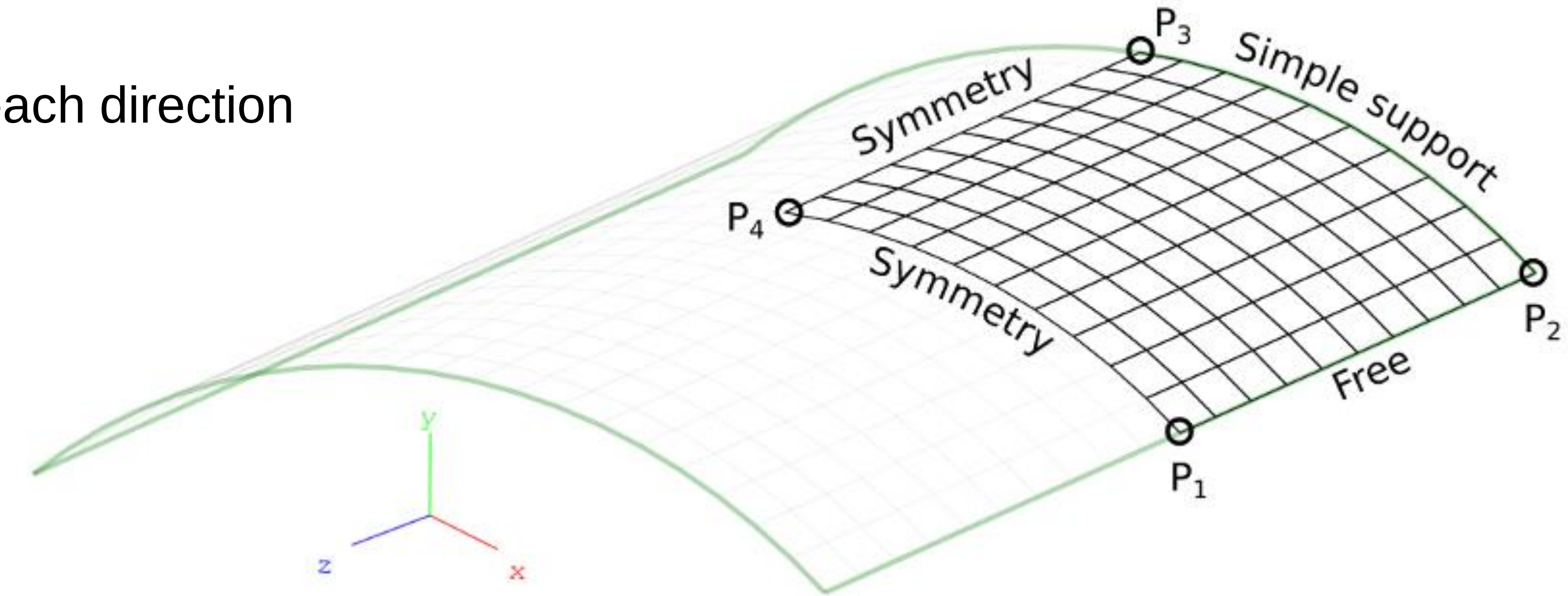
- CARO
  - AMD EPYC 7702 with 64 cores (8 cores share a L3 cache)
  - 2 Processors per node (total of 128 physical cores)
  - 256 GB RAM
  - 1276 nodes, max. aggregated network bandwidth: 557 TB/s

- b2000++ pro in version 4.5.2
  - MUMPS in version 5.5.1
  - OpenBLAS in version 0.3.21
  - Intel TBB in version 2020.3

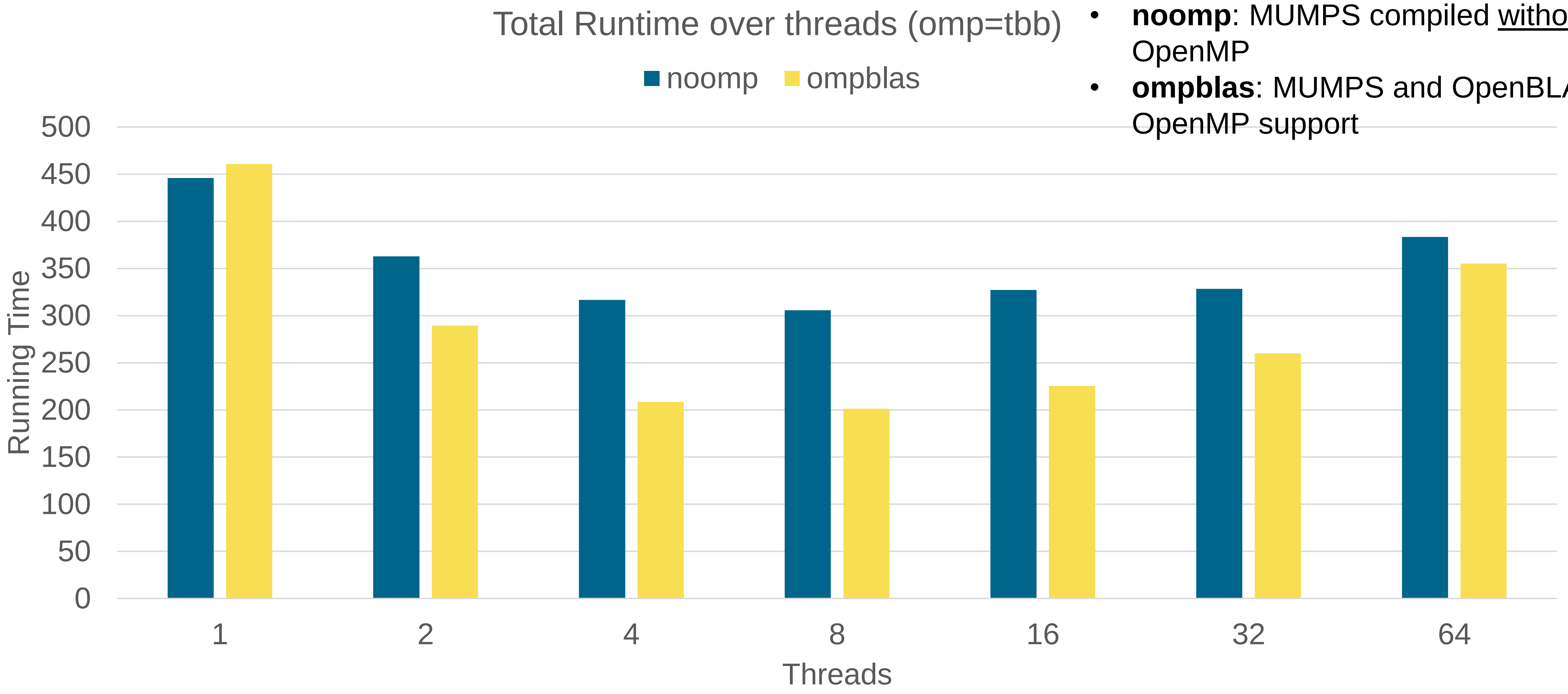
- Processes pinned to as many cores as threads used

Setup for Assessment

- Scordelis-Lo Roof:
  - https://www.smr.ch/doc/b2000pp/b2examples/html/static.html#scordelis-lo-roof-linear-analysis
  - Shell elements
  - Standard test case
  - 920 Elements in each direction



Single MPI Process: with / without OpenMP

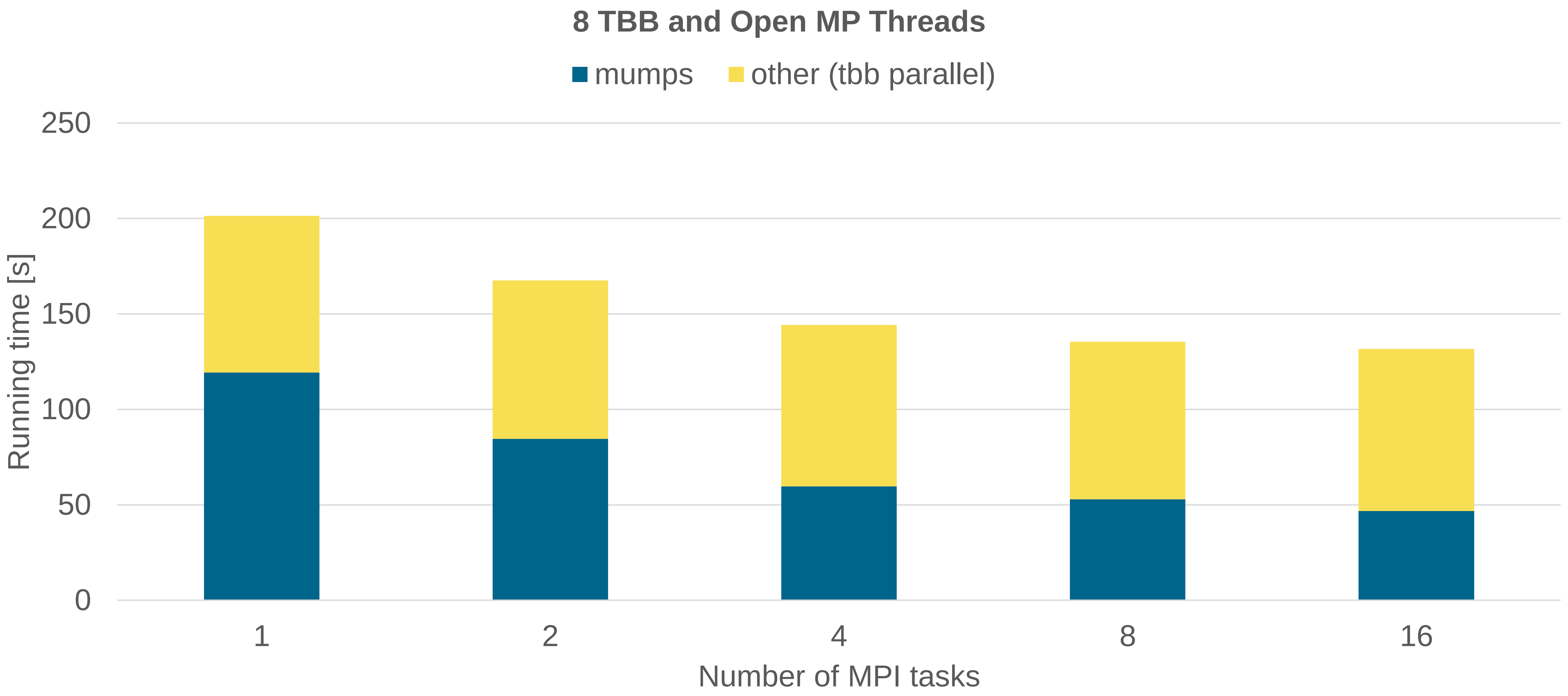


- **noomp**: MUMPS compiled without OpenMP
- **ompblas**: MUMPS and OpenBLAS with OpenMP support

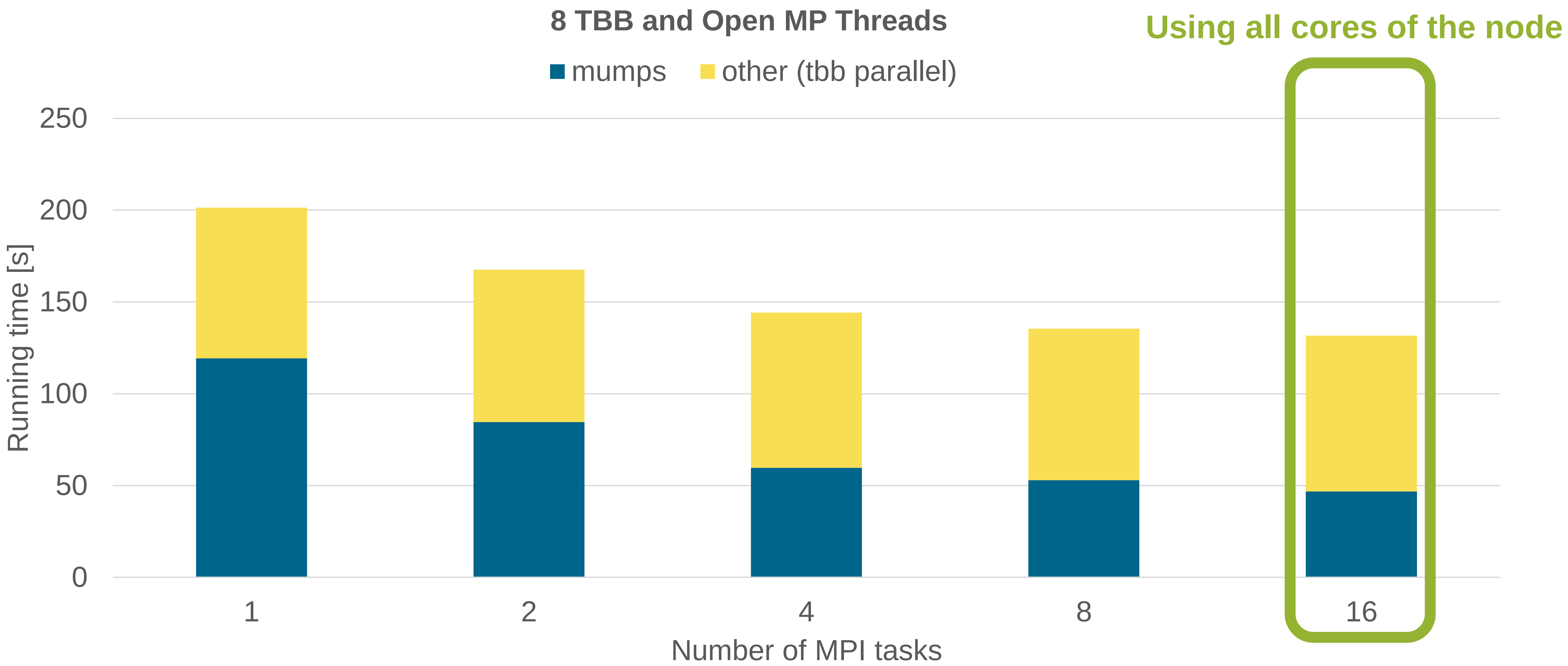
Single MPI Process: Resulting Configuration

- Fastest time to solution achieved with 8 threads and utilization of OpenMP
- Note: in serial execution on a single core the MUMPS part of the computation makes up around 60% of the overall running time
- In the (optimal) configuration with 8 threads the MUMPS part makes up 75% of the overall running time
- Nearly no difference between 4 and 8 threads

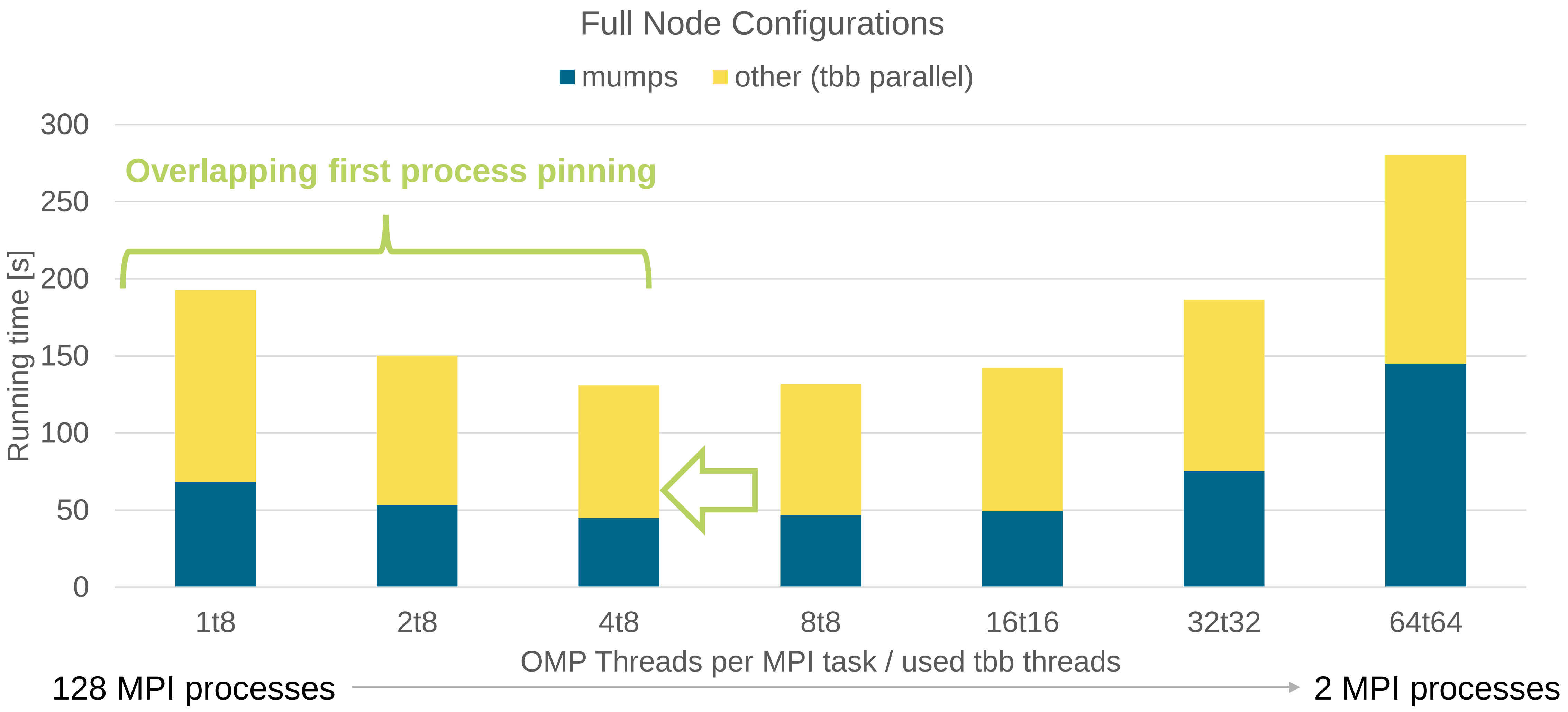
Using MPI to Accelerate MUMPS part



Using MPI to Accelerate MUMPS Part



Optimal Full-Node Configuration



Summary Current Parallelism

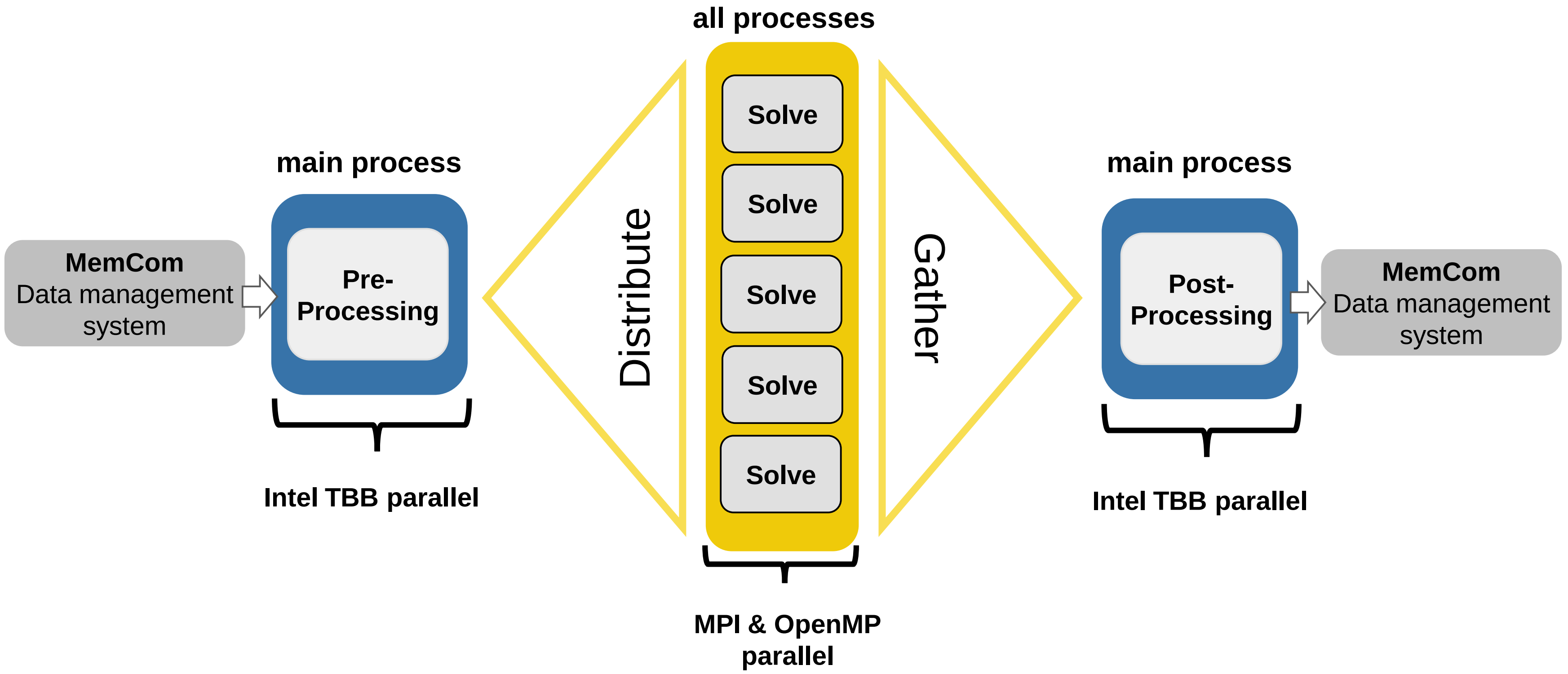
- Shared memory scaling limited around size of logical NUMA domain

- Optimal configuration using all cores has:
  - 32 MPI tasks
  - with 4 OpenMP threads each
  - but the main (first) MPI process is pinned to 8 cores, overlapping the pinning of the second process, to allow it to
  - utilize 8 cores for TBB

- In this optimal configuration the MUMPS part makes up 34% of the overall running time

Need for Revising Parallelization

- Current implementation severly limits scalability of the TBB-only part
- Complete problem has to fit into main process and thus, a single node

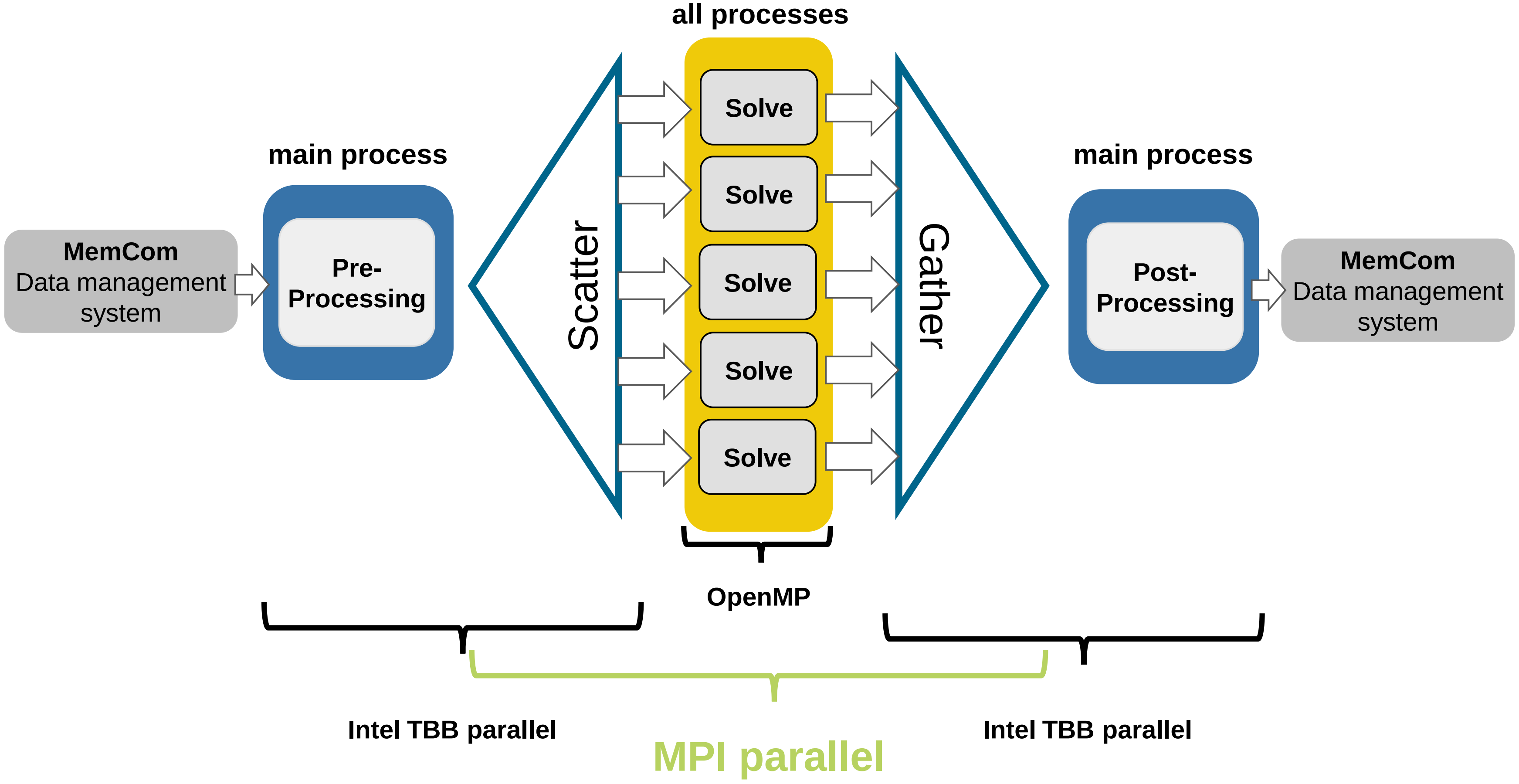


Partitioned, Distributed Computation

- Transforming the code from two ends:
  - Backwards from the linear algebra towards exchanging data with memcom
  - Borwards from the memcom data source towards the linear algebra

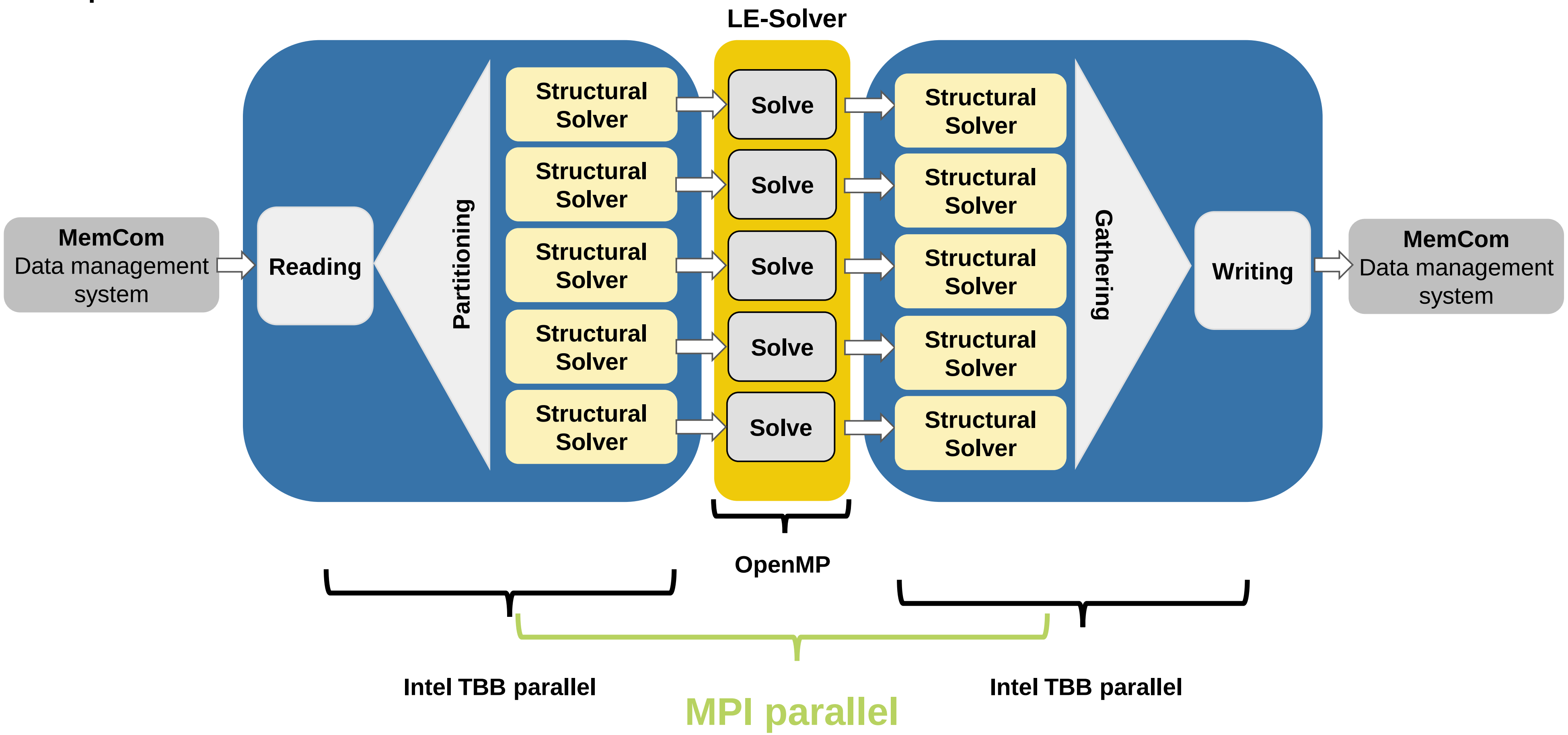
- First backward step, non breaking drop-in replacement:
  - Perform distribution of Matrix for the linear algebra in b2000++pro itself, rather than let the linear algebra package take care of that

Scattering and Gathering by the Structural Solver Instead of MUMPS



Forward-Step: Partitioning the Mesh After Reading

- Breaking the b2000++pro application:
  - Complete execution needs to deal with distributed mesh



Partitioning

- Main process obtains mesh from database and scatters it to all processes
- For now all processes get all nodes, but only a subset of elements to work on
- Partitioning is complicated by:
  - Different kinds of elements
  - Different kinds of nodes
- Requires communication to identify respective kinds

Distributing the Structural Solver Itself

- Both ends of structural solver part are distributed
- Distributed handling of the structural solver itself:
  - Need to deal with distributed global matrix
  - Need to allow for reduced matrices
  - Adapt internal dof representation to allow for distribution
  - Changed interfaces
- Introducing tpetra from Trilinos to handle distributed matrix operations in the solver
  - https://docs.trilinos.org/dev/packages/tpetra/doc/html/index.html
- Ongoing work with multiple redesigns of the class interfaces

Future Plans

- Further down the road it would also be desirable to distribute the reading of mesh data from the data base
- Use a container like HDF-5 for parallel IO in combination with the memcom database
- Probably more work in memcom than in b2000++

Summary

- Assessment of b2000++pro on CARO revealed optimal parallel configuration for single node computations
- Limited scalability with current split approach
  - Main process can only exploit limited number of cores (8 of 128) in thread parallelism
  - All data has to fit on main process
- Distribution of more parts is ongoing effort
  - Current stage requires large change of everything at once before it works again
  - „Easy“ parts have been transformed
  - Possibly, utilizing tpetra for distributed matrix handling

THANK YOU!