ParaSkel++: Parallel implementation of high-order Skeletal methods in C++

Thoma Zoto

Inria, Lille

thoma.zoto@inria.fr

March 8th, 2024

Introduction to ParaSkel++ v1

- Numerical approximation of PDEs on general polytopal meshes.
- Motivation: Big problems. Long-time non-linear problems.
- Skeletal Galerkin methods: FEM, VEM, HHO, etc.
  - Skeleton DOFs shared by adjacent cells and possibly bulk DOF.
  - Global discrete problem writes as sum of local bilinear contributions.
- Implemented in a unified way for both 2D and 3D polytopal meshes.
- Programmed in C++ and running in serial.
- Open source: https://gitlab.inria.fr/simlemai/paraskel
- Goal: evolve to a parallel high-performance code.
- Other contributors: Laurence Beaude and Simon Lemaire.
- Other codes: HArDCore2D, HArDCore3D, DiSk++.

Introduction to ParaSkel++ v1

- **Initialization**: parse execution options, read and build the mesh.
- **Local Driver**: manage local cell-by-cell memory and computations.
- **Assembly**: use a local-to-global map to assemble the global problem.
- **Solver**: solve a linear system.
- **Post-processing**: reconstruct solution, visualization, compute norms.

ParaSkel++ v1: Initialization

- Parse execution options: meshfile, skeletal degree, bulk degree etc.
- Read the mesh: gmsh, fvca5 and fvca6 formats.
- Build the mesh: fill the vector of vertices and the vectors that specify the mesh connectivity.

ParaSkel++ v1: Local Driver

- Virtual base class FiniteElement holds all the local information for each cell.
- FiniteElement class acts as an interface to any of the methods implemented (i.e. VEM, HHO etc).
- Many of the functions are *virtual* and only defined in the respective derived classes.

```
template<size_t DIM>
class FiniteElement
{
public:
  void init_dof_local_index();
  void init_mass_matrices(...);
  void init_stiffness_matrices(...);
  virtual void add_stiffness_contribution(...) = 0;
  virtual void add_mass_contribution(...) = 0;

  MyMesh::Cell<DIM>* get_cell();
};
```

ParaSkel++ v1: Local Driver

- Driver class DiscreteSpace instantiates the FiniteElement objects and loops over all to perform the local computations.
- Templates allow to fix the dimension and the continuous spaces where the variables live in.

```
ds.init<MyDiscreteSpace::HHOdiscontinuousPk,
     MyDiscreteSpace::H1d, MyDiscreteSpace::L2_0>(
     mesh.get(), bulk_k, skeletal_k);
ds.add_stiffness_contribution(var_u, mu);
```

- Creation of the FiniteElement objects is done by using FET and CONTINUOUS_SPACES passed as template args:

```
for (auto& c : mesh->get_cells()) {
  FiniteElement<DIM>* fe(new FET<DIM,
        CONTINUOUS_SPACES...>(c, kb, ks, m_var_sizes));
  fe->init_dof_local_index();
  m_finite_elements[i++] = fe;
}
```

ParaSkel++ v1: Local Driver

- When looping through FE objects to perform local computations, we call member functions from derived classes:

```
for (auto& fe : m_finite_elements) {
  fe->add_stiffness_contribution(var_index, mult_cst);
}

for (auto& fe : m_finite_elements) {
  fe->add_divergence_contribution(var_index1,
                                  var_index2, measure, mult_cst);
}
```

- DiscreteSpace class also performs static condensation by means of a purely algebraic Schur complement routine:

```
for (auto& fe : m_finite_elements) {
  fe->static_condensation();
}
```

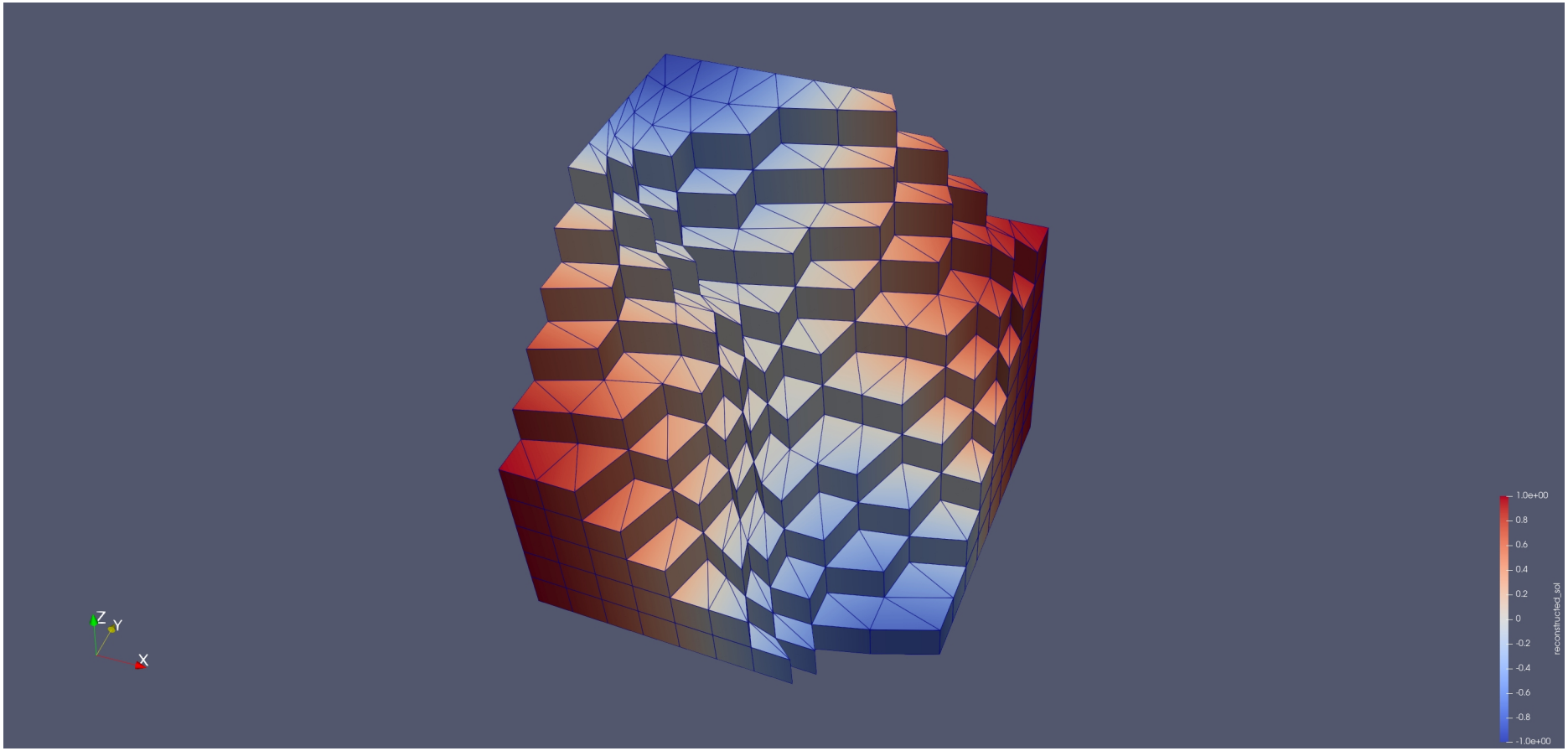
ParaSkel++ v1: Assembly

- The assembly functionality is implemented within the MatrixPattern class in a unified way regardless of the particular skeletal method used.
- No dependence on the type of DOF (vertex, edge, face or cell).
- Creates the local-to-global map that maps the local indices of the degrees of freedom to their global indices.
- Prompts the DiscreteSpace object to perform static condensation if enabled by the user.
- Assembles the local contributions to form the global matrix and RHS.

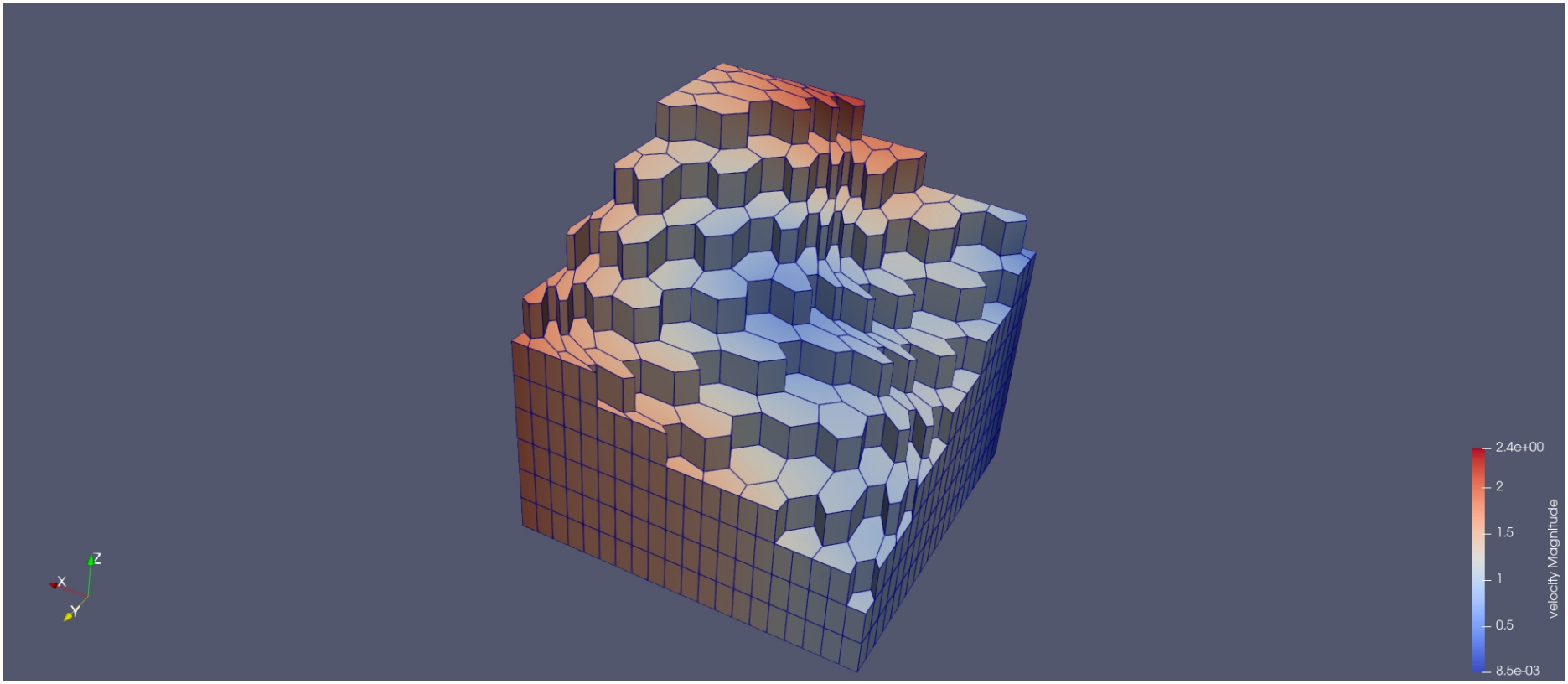
ParaSkel++ v1: Solver and Post-processing

- Linear solvers from the Eigen Library (direct or iterative).
- Reconstruction of the numerical solution of the PDE from the computed DOFs is done in the DiscreteSpace object that has been created before the local computations.
- VtkVisu class handles the visualization output in a format that is to be read by ParaView.
- Free functions within the MyDiscreteSpace namespace compute the relative l2 and h1 norms.
- The user can choose at execution time to output simulation characteristics and timers.

ParaSkel++ v1: Poisson problem with $HHO(3,2)$



ParaSkel++ v1: Stokes problem with $v \in HHO(1,0)^3$ and $p \in disP0$



Plans for ParaSkel++ v2

- Extend the code to run in parallel (with minimal code disturbance).
- Implement quadrature computations on a general polytope (without using mesh sub-tessellation).

Introduction to ParaSkel++ v2

- **Initialization**: parse execution options, read and build the mesh.
- **Local Driver**: manage local cell-by-cell memory and computations.
- **Assembly**: use a local-to-global map to assemble the global problem.
- **Solver**: solve a linear system.
- **Post-processing**: reconstruct solution, visualization, compute norms.

ParaSkel++ v2: Parallelization

- Two main phases: local computations and solution of linear system.
- "In house" parallelization in the local computations phase.
- Third library parallelization in the solver phase.
- Three parallelization modes: openMP, MPI, MPI + openMP.
- Difficulty: minimal disturbance of the current code design.
- Solution: implement as a sub-library, encapsulated in a *namespace*.
- Result: logic behind the work distribution identical for all modes.

ParaSkel++ v2: Parallelism of Local Driver

- "In house" parallelism in ParaSkel++ is built around the possibility of passing lambdas to C++ functions:

```
#if defined PARALLEL_HYBRID

// Same function as below,
// but treating the for loop differently!

#else

template<size_t DIM, class FEM>
inline void parallel_if(bool condition,
                        std::vector<FEM *> const &list_of_elems,
                        std::function<void(FEM*)> code) {
  for(size_t i=0; i<list_of_elems.size(); i++) {
    code(list_of_elems[i]);
  }
}

#endif
```

ParaSkel++ v2: Parallelism of Local Driver

- Before:

```
for (auto& fe : m_finite_elements) {
  fe->add_stiffness_contribution(var_index, mult_cst);
}
```

- After:

```
ParallelDependent::parallel_if<DIM,
        MyDiscreteSpace::FiniteElement<DIM>>(true,
        m_finite_elements, [=](FiniteElement<DIM>* fe){
  fe->add_stiffness_contribution(var_index, mult_cst);
});
```

ParaSkel++ v2: Parallelism of Local Driver

- Extra step required for distributed and hybrid mode: gathering all the data from each node to form the linear system in the main node.
- Currently achieved by:
  - collecting all the *Eigen Triplets* on each MPI node,
  - gathering them to the main MPI node,
  - creating the Eigen Sparse matrix from Triplets.
- Extra delicacies: Input/Output wrappers, custom timers, MPI initialize/finalize functions etc.
- Very little disturbance (all the magic happens in parallel.hpp).

ParaSkel++ v2: Heisenbugs!

```
thoma@Asgard:~/Documents/RAPSODI/paraskel/build$ tmpi 2 gdb ./bin/hho_poisson
```

```
GNU gdb (Ubuntu 12.1-0ubuntu1~22.04) 12.1
Copyright (C) 2022 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<https://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
    <http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from ./bin/hho_poisson...
(gdb) set disable-randomization off
(gdb) run -m ../Meshes/gmsh_meshes/3Dcube_1_737tetra.msh
```

```
GNU gdb (Ubuntu 12.1-0ubuntu1~22.04) 12.1
Copyright (C) 2022 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<https://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
    <http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from ./bin/hho_poisson...
(gdb) set disable-randomization off
(gdb) run -m ../Meshes/gmsh_meshes/3Dcube_1_737tetra.msh
```

```
[0] 0:[tmux]- 1:gdb*                                  "Asgard" 17:44 06-mars-24
```

ParaSkel++ v2: Parallelism of Solver and Post-processing

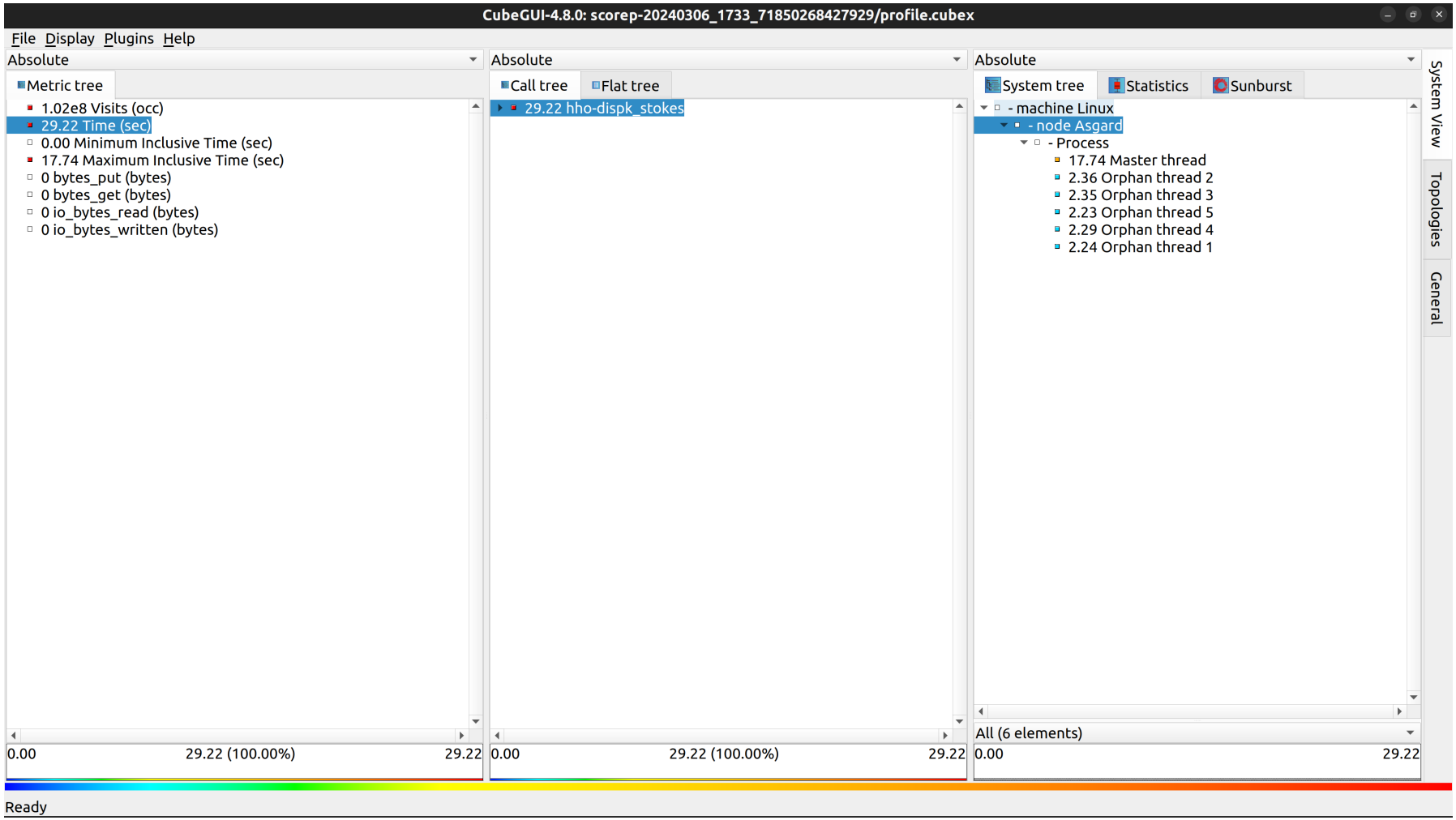
- Experimentation with parallel solvers in ParaSkel++.
- Eigen's own solvers can be used in parallel, but only in shared memory.
- Succesful usage of Intel MKL Pardiso solver.
- Due to Intel MKL being difficult to install, usage of Pardiso support is optional at compile time.
- Parallel reconstruction of the numerical solution (part of the "in house" sub-library).
- In distributed memory, the local cell-by-cell reconstructions are gathered in the main node.

ParaSkel++ v2: Parallelism of Solver and Post-processing

- Visualization class not disturbed at all (even on distributed memory mode).
- "In house" parallel norm computations (also implemented using lambdas).

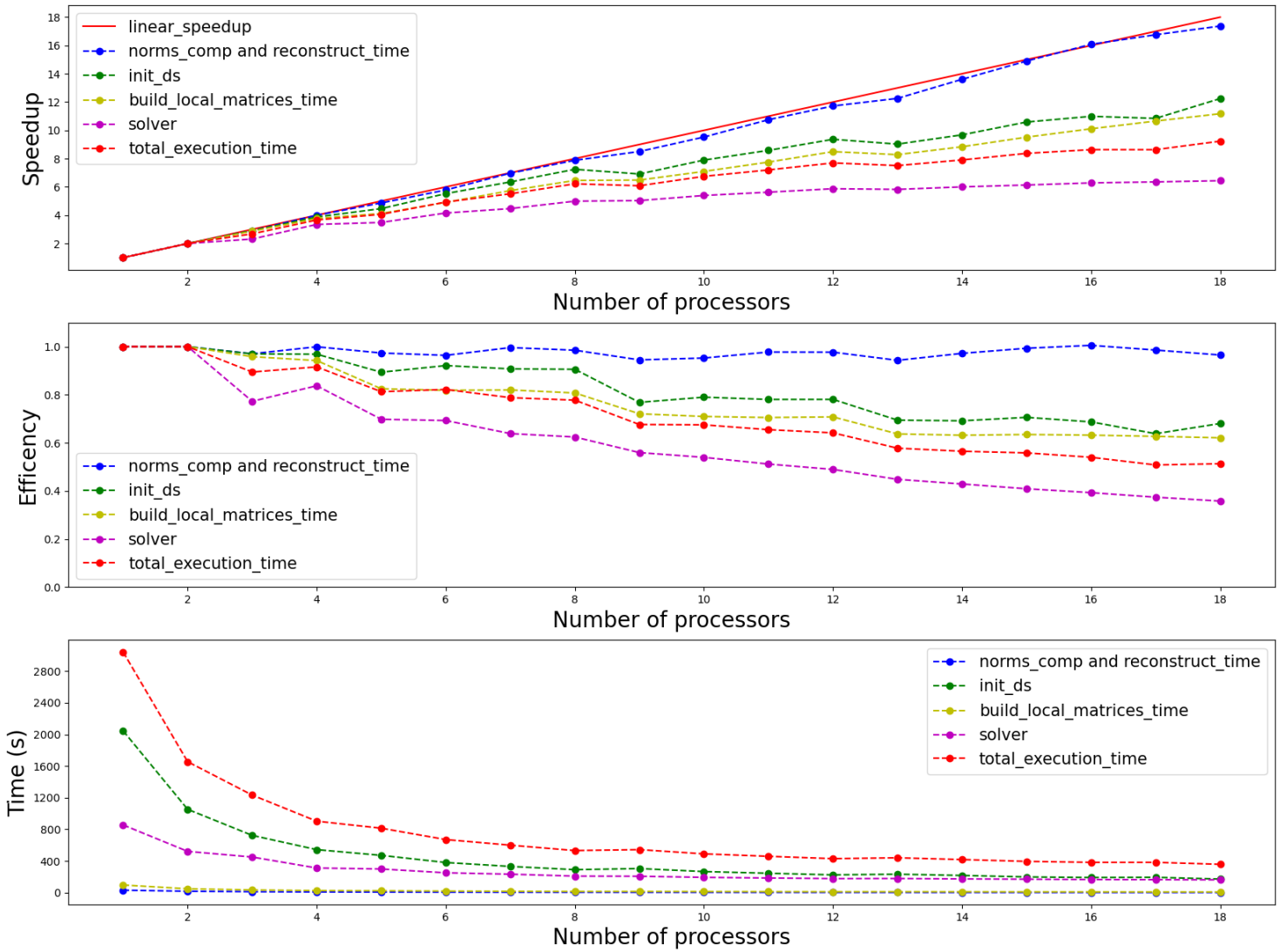
```
template<size_t DIM, typename RET>
inline double relative_l2error(DiscreteSpace<DIM>* ds,
        RET func (Eigen::ArrayXd), size_t quadra_order,
        size_t var_index) {
  double diff_norm(0); double exact_norm(0);
  ParallelDependent::parallel_add<DIM,
        MyDiscreteSpace::FiniteElement<DIM>>(true,
        ds->get_finite_elements(), diff_norm,
        exact_norm, [=](FiniteElement<DIM>* fe,
        double& proc_diff_norm, double& proc_exact_norm){
    ...
    });
    ...
}
```

ParaSkel++ v2: Profiling with Score-P for load balancing



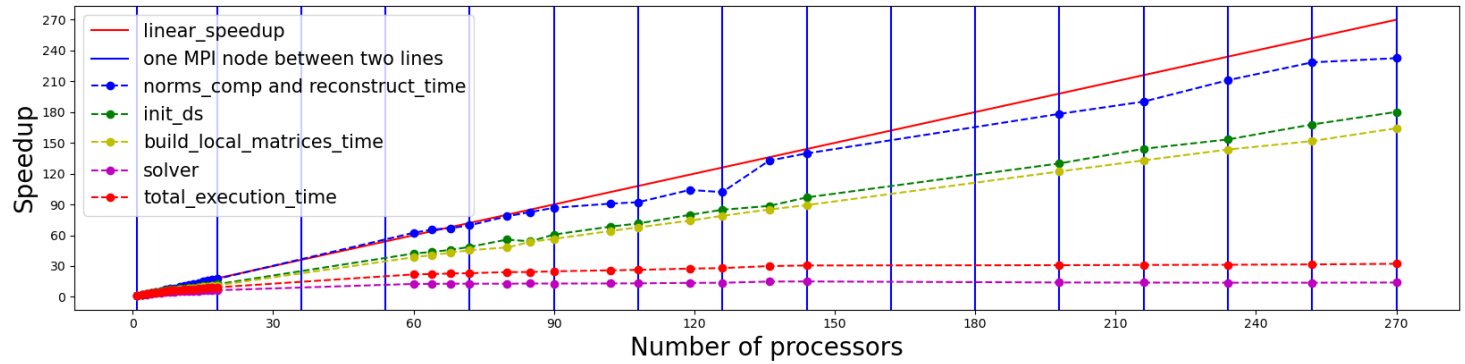
ParaSkel++ v2: Parallel scaling in Grid5000

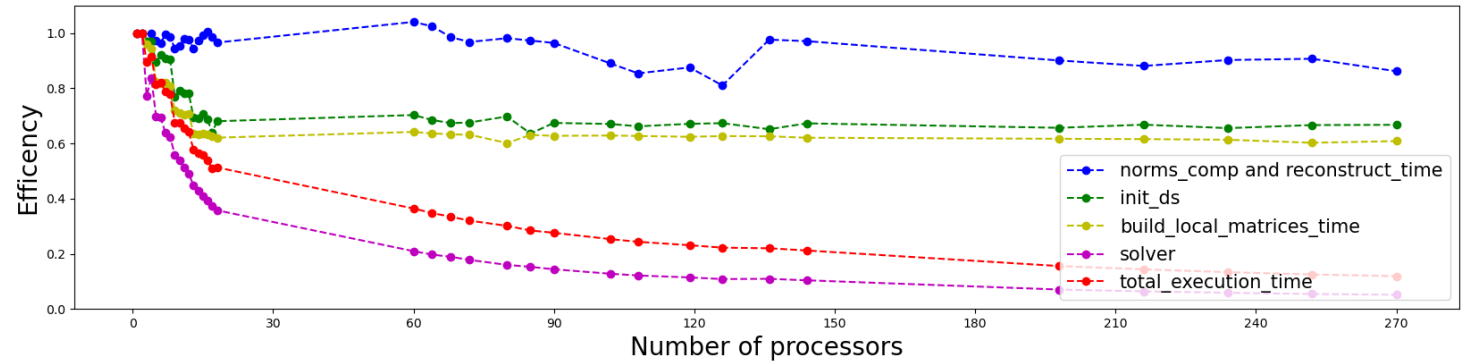
Stokes Problem with HHO (bulk order 4 and skeletal order 3) for velocity and discontinuous P3 for pressure (OpenMP)

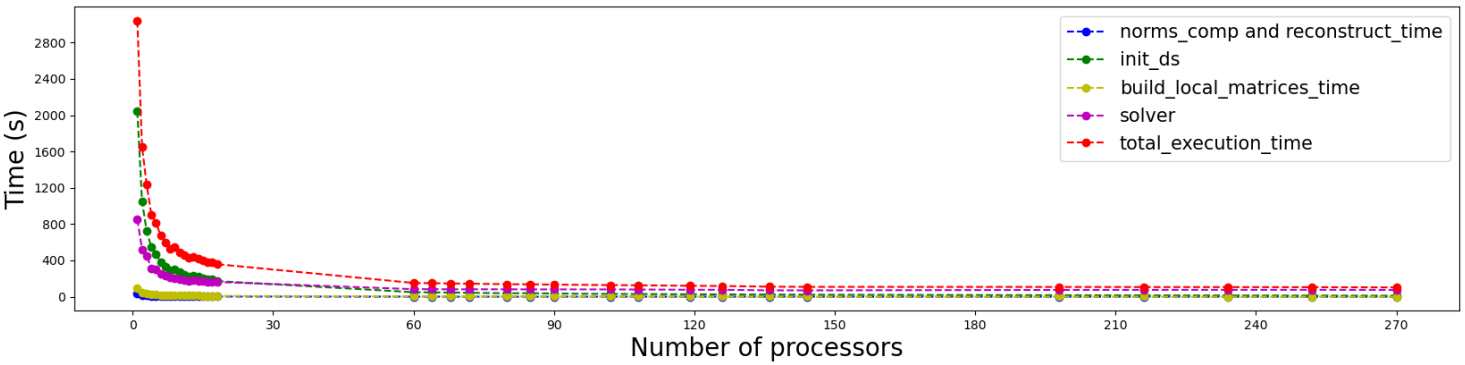


ParaSkel++ v2: Parallel scaling in Grid5000

Stokes Problem with HHO (bulk order 4 and skeletal order 3) for velocity and discontinuous P3 for pressure (MPI + OpenMP)



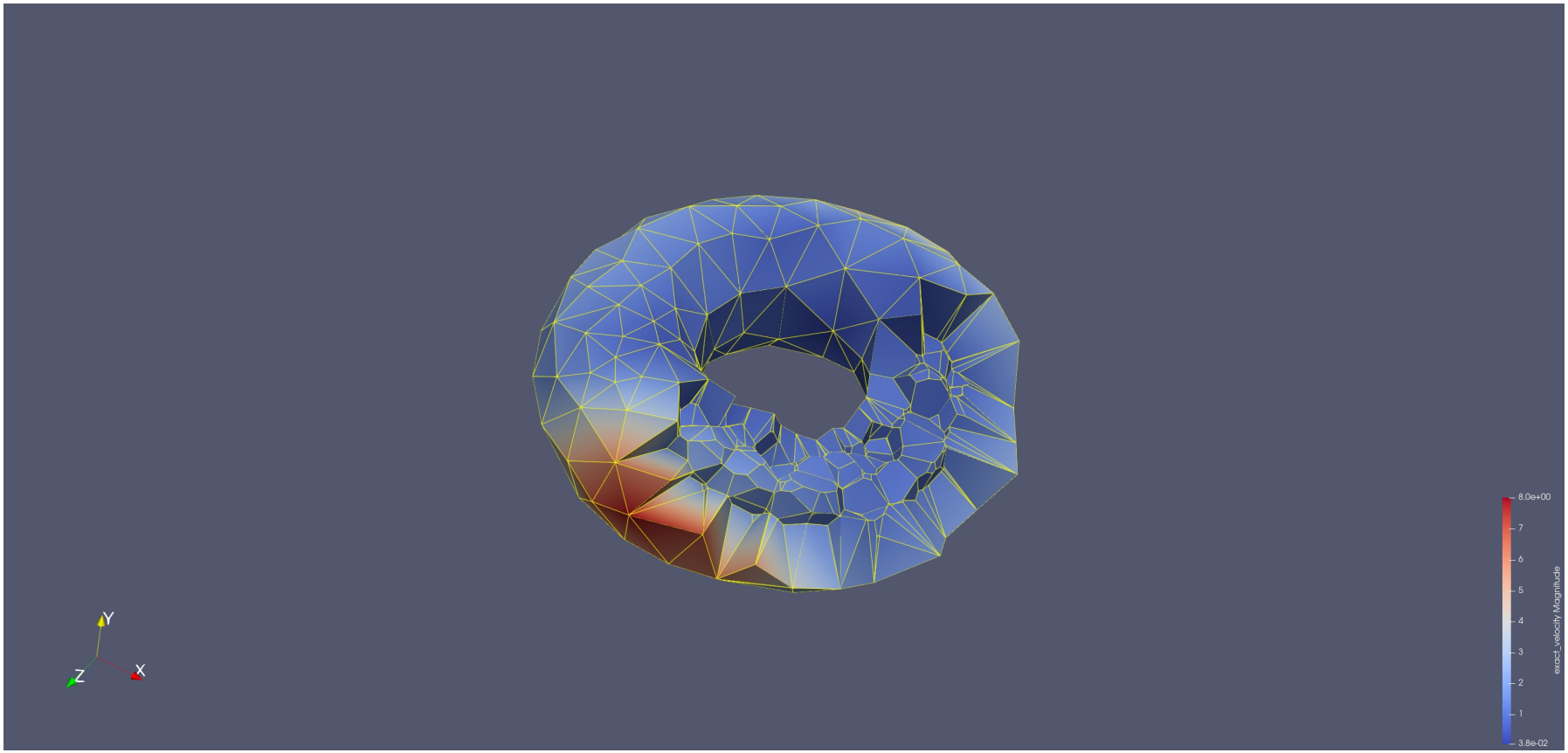




ParaSkel++ v2: Polytopal quadratures

- Monomials (homogeneous functions in general) satisfy Euler's homogeneous function theorem.
- Generalized Stokes theorem.
- Integration of monomials over a polytope can be simplified to integration over the edges.
- Further simplifications lead to a cubature rule for a monomial, where the integration points are the vertices of the polytope.
- Idea from Chin, Lasserre, Sukumar [2015].
- Improvement by Yemm [2023].

ParaSkel++ v2: Interfacing VoroCrust



Developed at Sandia National Labs:
https://github.com/sandialabs/vorocrust-meshing

Moving Forward

- Finalizing the interface to VoroCrust.
- Further testing and optimisation of polytopal quadratures with no tessellation by simplices.
- Optimising the data gathering after the local computations are done in parallel.
- Interfacing another linear solver library (MUMPS or PETSc).
- Further scalability testing with bigger problems using more MPI nodes.

Thank you!

Figure: Cap Blanc-Nez, Côte d'Opale