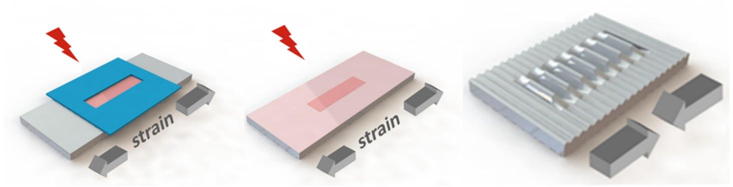
Lisa Julia Nebel, Ansgar Burchardt and Oliver Sander

Container use on Taurus: FEM simulations with the DUNE framework in C++

Container im HPC - Admin Workshop, 13.12.2021

FEM simulations for experiments with a bi-layer system

- Project of Institute for Numerical Mathematics, TU Dresden, and Leibniz-Institut für Polymerforschung Dresden
- Goal: create wrinkles of a **specific wavelength and control the splitting behavior**
- Experimental setup:
  - Stretching of an elastic polymer layer
  - Different gas treatments of specific areas
  - Relaxation of the elastic polymer layer

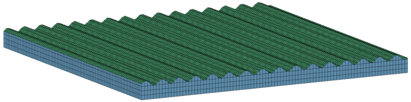


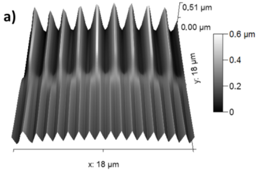
Simulation of the bi-layer system: Setup

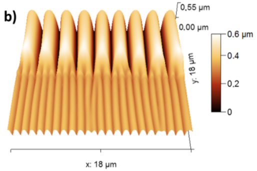
- Combination of a model from 3D-elasticity and a 2D-Cosserat shell model
- Minimization problem on the set A of all admissible deformations:

$$\text{find a local minimum } x^* \in A : \quad J(x^*) \leq J(x) \quad \forall x \text{ close to } x^*$$

where J is a possibly non-linear and non-convex energy function







Simulation of the bi-layer system: Optimiziation

- Non-linear and non-convex minimization problem:

$$\text{find a local minimum } x^* \in A: \quad J(x^*) \leq J(x) \quad \forall x \text{ close to } x^*$$

- solve **iteratively** using a Trust-Region algorithm (similar to Newton's method):

1. Quadratic approximation of J at the current iterate x_k:

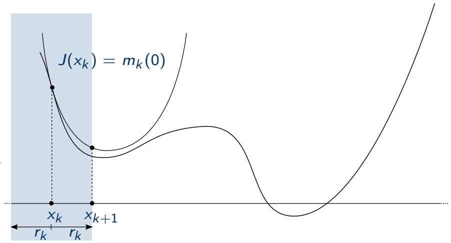
$$m_k(c) = x_k + \boldsymbol{\nabla J}(x_k)c + \frac{1}{2}c^T\boldsymbol{\nabla}^2\boldsymbol{J}(x_k)c$$

2. Minimize $m_k(c)$ **inside Trust-Region**:

$$\text{minimize} \quad \nabla^2 J(x_k)\cdot c + \nabla J(x_k), \quad \|\boldsymbol{c}\| \leq \boldsymbol{r_k}$$

3. Add correction: $x_{k+1} := x_k + c$

4. Update Trust-Region radius



Simulation of the bi-layer system: High effort

- Combination of a model from 3D-elasticity and a 2D-Cosserat shell model

  $\rightarrow$ Non-linear, non-convex minimization problem
  $\rightarrow$ Iterative solver needed
  $\rightarrow$ In each iteration: Assembly of $\nabla J(x_k)$ and $\nabla^2 J(x_k)$ at $x_k \rightarrow$ expensive
  $\rightarrow$ In each iteration: Quadratic obstacle problem $\rightarrow$ expensive

- FEM-functions of second order needed to avoid “locking“-effects

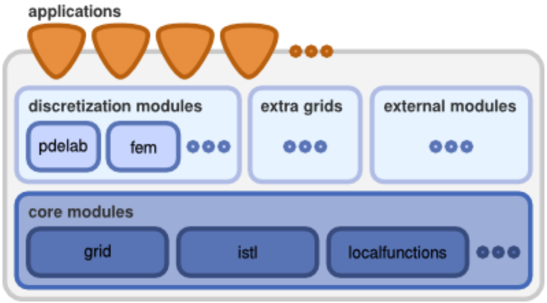
  $\rightarrow$ High coupling of the FEM-functions

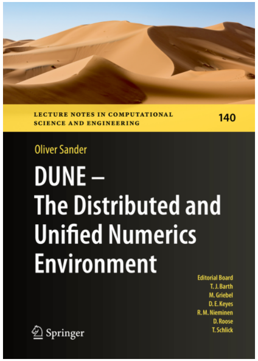
- Interest in fine structures

  $\rightarrow$ Fine grid in the vicinity of the shell needed

Implementation using DUNE: `www.dune-project.org`

- **DUNE**: toolbox for solving partial differential equations in C++
- Free software licensed under the GNU General Public License




LECTURE NOTES IN COMPUTATIONAL SCIENCE AND ENGINEERING
140
Oliver Sander
DUNE – The Distributed and Unified Numerics Environment
Editorial Board
T. J. Barth
M. Griebel
D. E. Keyes
R. M. Nieminen
D. Roose
T. Schlick
Springer

Implementation using DUNE: `www.dune-project.org`

- **DUNE**: toolbox for solving partial differential equations in C++
- Free software licensed under the GNU General Public License

- Implementation of the iterative Trust-Region-Solver
- In each iteration:
  1. Setup of the quadratic obstacle problem:
     - **Assembly of $\nabla J(x_k)$ and $\nabla^2 J(x_k)$ with ADOL-C**
     - Parallelized with MPI: grid is distributed on different processes
  2. Solve the quadratic obstacle problem:
     - **Multigrid method (iterative)**
     - Not parallelized yet

Running the simulation on Taurus

Development and simulations initially done locally, thus:

- Run simulation directly on Taurus with the module system
  - "Break" in the whole setup
  - Ongoing local development: **Bring Taurus' setup to local machine**, thus work with a module system locally as well ×
  - Many dependencies, some not included in module system: ×
    - ADOL-C to calculate the derivatives ∇J and $\nabla^2 J$
    - Parmetis to partition the grid
    - Linear algebra packages (IPOPT, SuperLU, CHOLMOD ...)
    - ...
- Run simulation inside a container: **Bring local setup to Taurus** ✓
  - Ongoing local development easy ✓
  - All dependencies needed are in debian, add them to the container definition ✓
  - Option to build simulation inside the container
  - Option to give the container to others

Running the simulation on Taurus inside a container

Current workflow:

1. Build the container locally
2. Push the container to a repository (Docker)
3. Load the container on Taurus using singularity
4. Build the simulation inside the container
5. Start the simulation inside the container using a batch file

Running the simulation on Taurus inside a container

Current workflow:

1. Build the container locally
2. Push the container to a repository (Docker)
3. Load the container on Taurus using singularity
   - Singularity needs `TMP` directory, standard set to `/tmp`
   - Needed to change this to `TMP` directory I can access
4. Build the simulation inside the container
5. Start the simulation inside the container using a batch file
   - Problems with MPI support on several nodes

Running the simulation on Taurus inside a container

5. Start the program inside the Docker container **on one node**:

```
 #!  /bin/bash
...
#SBATCH --nodes=1
#SBATCH --ntasks=26
#SBATCH --mem=0
...
/.../container.sif mpirun ./simulation parameterset
```

- Starts container on one node and uses maximal number of tasks for MPI
- Works fine ✓

Running the simulation on Taurus inside a container

5. Start the program inside the container **on several nodes**:
   $\rightarrow$ OpenMPI in the container needs to use InfiniBand hardware of Taurus

```
 #!  /bin/bash
...
#SBATCH --ntasks=96
#SBATCH --mem=0
...
export OMPI_MCA_btl_openib_allow_ib=true; srun --mpi=pmix -n 96
singularity exec /.../container.sif ./simulation parameterset
```

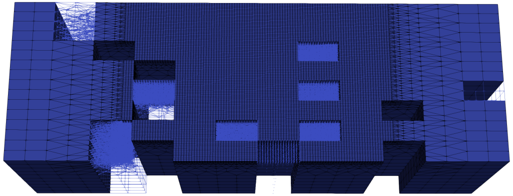
   - `OMPI_MCA_btl_openib_allow_ib=true`: use InfiniBand ports for communication
   - `srun --mpi=pmix`: slurm directly launches the application, performs initialization of communication through PMIx

Running the simulation on Taurus inside a container

5. Start the program inside the container **on several nodes**:

```
export OMPI_MCA_btl_openib_allow_ib=true; srun --mpi=pmix -n 96
singularity exec /.../container.sif ./simulation parameterset
```

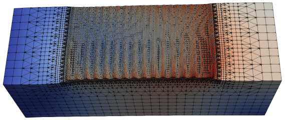
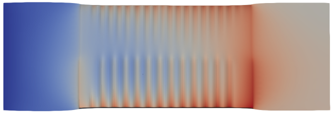
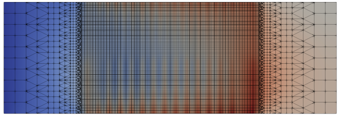
- Starts and distributes the grid as intended ✓
- Communication error, probably bug in our own code ☹



Simulation of the bi-layer system

- Quadratic elements in the shell: $40 \times 64 = 2560$
- DOFs: 477.468 (FEs of second order)
- 61 Trust-Region steps

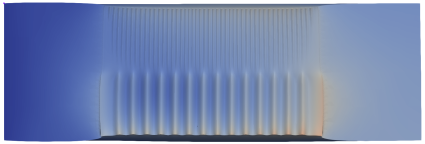
Total time for the problem setup (parallel on 27 tasks)	5,8 h	$\approx$ 6 min per step,	all tasks: 162 min
Total time to solve	9,98 h	$\approx$ 10 min per step	
Total time	16,27 h	$\approx$ 16 min per step	

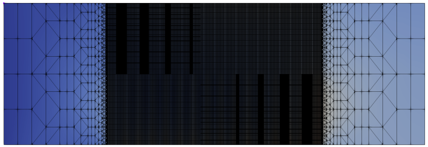




Simulation of the bi-layer system with a much finer grid

- DOFs: 2.663.508 (FEs of second order)
- 264 Trust-Region steps

Total time for the problem setup (parallel on 4 tasks)	206,43 h	$\approx$ 47 min per step,	all tasks: 188 min
Total time to solve	189,14 h	$\approx$ 43 min per step	
Total time	402,03 h	$\approx$ 90 min per step	





Simulation of a tri-layer system

- Quadratic elements in the shell: 48 $\times$ 48 = 2304
- 607 Trust-Region steps
- DOFs: 389.154 (FEs of second order)

Total time for the problem setup (parallel on 24 tasks)	35,97 h	$\approx$ 3,5 min per step	all tasks: 84 min
Total time to solve	73,76 h	$\approx$ 7,3 min per step	
Total time	115,89 h	$\approx$ 11,5 min per step