Modular implementation of local meshless numerical method

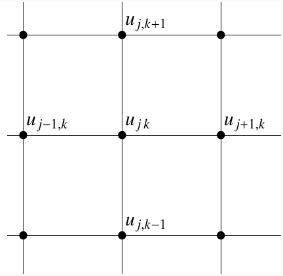
Gregor Kosec, **Jure Slak**

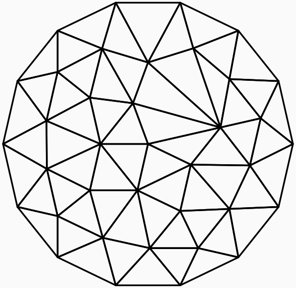
“Jožef Stefan” Institute, Parallel and Distributed Systems Laboratory

29. 10. 2019, ParNum 2019

1. Strong form meshless methods
2. Typical method description
3. Parallelization opportunities
4. Examples

- Classical approaches:
  Finite Difference Method, Finite Element Method





- Problems: inflexible geometry, mesh generation
- Response: mesh-free methods (EFG, MLPG, FPM)

Typical method description

JSI

Typical model problem: solve an elliptic boundary value problem:

$$
\begin{aligned}
-\mathcal{L}u &= f && \text{in } \Omega \\
u &= u_0 && \text{on } \partial\Omega
\end{aligned}
$$

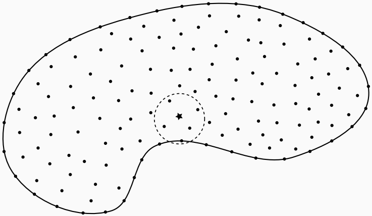
Typical solution procedure:

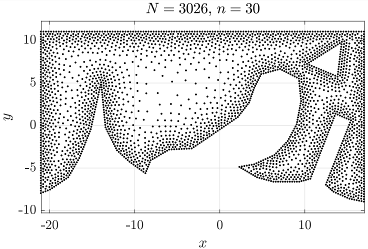
1. Domain discretization
2. Differential operator discretization
3. PDE discretization

Domain discretization

Computational nodes instead of a mesh:

- Points x_i on the boundary (work in progress) and in the interior[1]
- Complexity: $O(N \log N)$ for N nodes
- Point neighborhoods $N(x_i)$ of n nodes
- Complexity: $O(nN \log N)$





[1] Slak, J. and Kosec, G. *On Generation of Node Distributions for Meshless PDE Discretizations.* SIAM J. Sci. Comput., 41(**5**), A3202–A3229.

Classical Finite Differences:

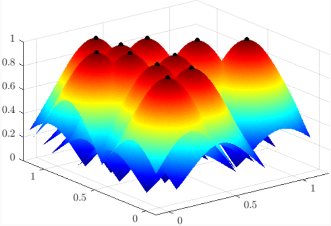
$$u''(x_i) \approx \frac{1}{h^2}u(x_{i-1}) - \frac{2}{h^2}u(x_i) + \frac{1}{h^2}u(x_{i+1})$$

General strong form methods:

$$(\mathcal{L}u)(x_i) \approx \sum_{x_j \in N(x_i)} w_j^i u(x_j)$$

Many different methods to obtain w_j^i (FPM, GFDM, LRBFCM, MLSM, RBF-FD, DAM, DLSM). Brief description of RBF-FD follows.

Weights are obtained by imposing exactness for RBFs:



- Given nodes $X = \{x_1, \ldots, x_n\}$ and a radial function $\varphi = \varphi(r)$
- Generate $\{\varphi_i := \varphi(\|\cdot - x_i\|), x_i \in X\}$

Imposing exactness of

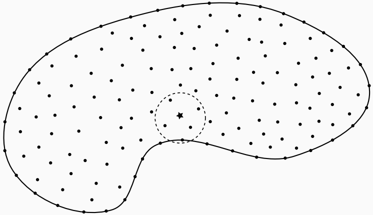
$$(\mathcal{L}u)(x_i) \approx \sum_{x_j \in N(x_i)} w_j^i u(x_j)$$

for each φ_j for $x_j \in N(x_i)$, we get

$$\begin{bmatrix} \varphi(\|x_{j_1} - x_{j_1}\|) & \cdots & \varphi(\|x_{j_{n_i}} - x_{j_1}\|) \\ \vdots & \ddots & \vdots \\ \varphi(\|x_{j_1} - x_{j_{n_i}}\|) & \cdots & \varphi(\|x_{j_{n_i}} - x_{j_{n_i}}\|) \end{bmatrix} \begin{bmatrix} w_{j_1}^i \\ \vdots \\ w_{j_{n_i}}^i \end{bmatrix} = \begin{bmatrix} (\mathcal{L}\varphi_{j_1})(x_i) \\ \vdots \\ (\mathcal{L}\varphi_{j_{n_i}})(x_i) \end{bmatrix}$$

Problem:

$$\mathcal{L}u = f \quad \text{on } \Omega,$$
$$u = u_0 \quad \text{on } \partial\Omega,$$



1. Discretize domain Ω
2. Find neighborhoods $N(x_i)$
3. Compute weights $\boldsymbol{w}^i$ for approximation of $\mathcal{L}$ over $N(x_i)$
4. Assemble weights in a sparse system $Wu = f$
5. Solve the sparse system $Wu = f$
6. Approximate/interpolate the solution

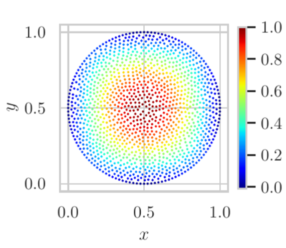
Try it out: `http://e6.ijs.si/medusa`

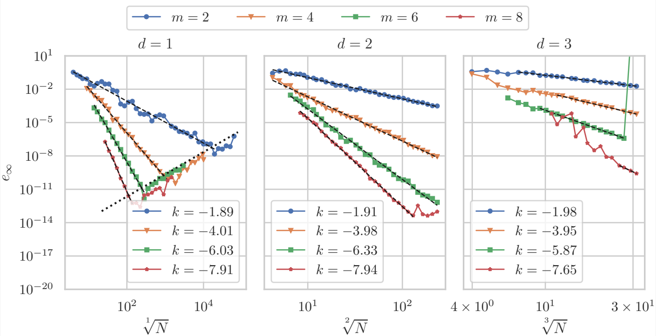
Sample Poisson problem

$$-\triangle u = -d\pi^2 \prod_i \sin(\pi x_i) \qquad \text{in } \Omega,$$

$$u = \prod_i \sin(\pi x_i) \qquad \text{on } \Gamma_1,$$

$$\frac{\partial u}{\partial \vec{n}} = \vec{n} \cdot \nabla \prod_i \sin(\pi x_i) \qquad \text{on } \Gamma_2.$$





C++ implementation with Eigen as linear algebra library

```
BallShape<Vec2d> ball(0.5, 0.5);
DomainDiscretization<Vec2d> domain =
    ball.discretizeWithStep(0.01);
domain.findSupport(FindClosest(9));
RBFFD<Gaussian<double>, Vec2d> approx(
    1.0, Monomials<Vec2d>(2));

auto storage = domain.computeShapes
    <sh::lap|sh::d1>(approx);
Eigen::SparseMatrix<double, Eigen::RowMajor> M(N, N);
M.reserve(storage.supportSizes());
Eigen::VectorXd rhs(N);
rhs.setZero();
```

Continued...

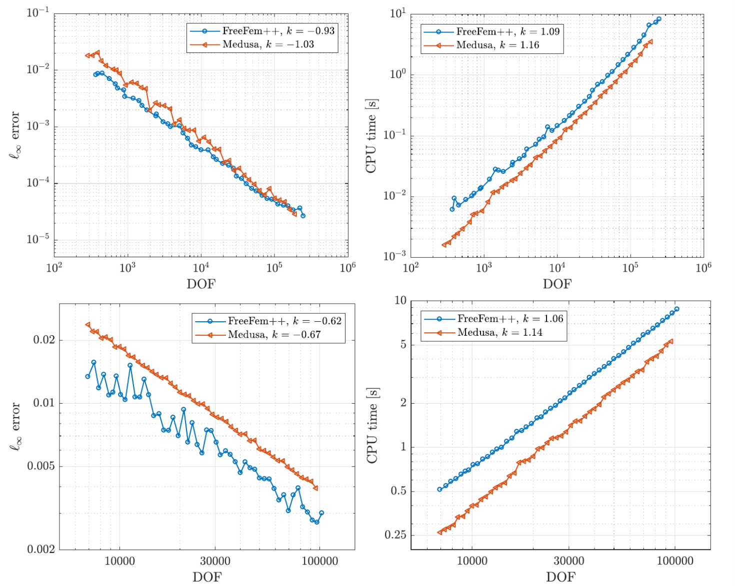
```
auto op = storage.implicitOperators(M, rhs);
for (int i : domain.interior()) {
    -op.lap(i) = 0.0;
}
for (int i : domain.boundary()) {
    op.value(i) = 1.0;
}

PardisoLU<decltype(M)> solver;
solver.compute(M);
Eigen::VectorXd u = solver.solve(rhs);
```

Modularity, readability, speed, dimension independence, negligible cost of abstractions

Execution time and accuracy – comparison

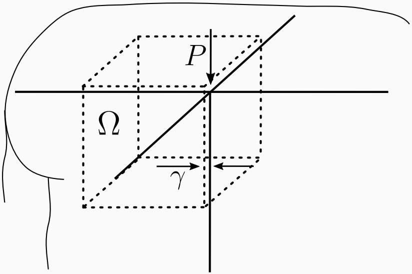
Comparison with FreeFem++



2D

3D

Test case - linear elasticity

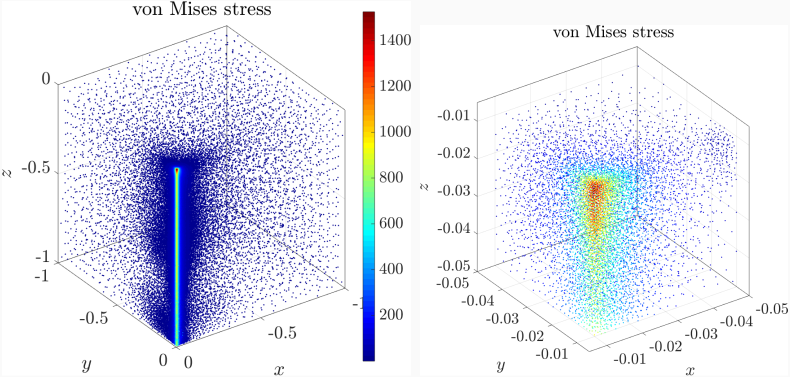


Cauchy-Navier equation

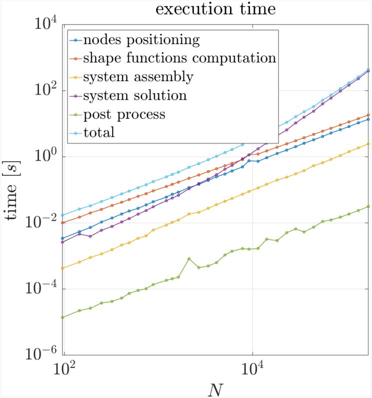
$$(\lambda + \mu)\nabla(\nabla \cdot \vec{u}) + \mu \nabla^2 \vec{u} = \vec{f}$$

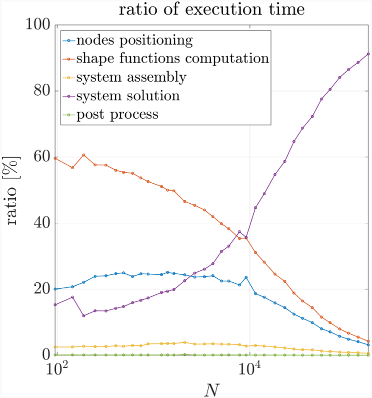
in domain $\Omega = [-1, -\gamma]^3$ with Dirichlet boundary conditions.

Solution for $\gamma = 0.01$.



Execution time





Parallelization opportunities

JSI

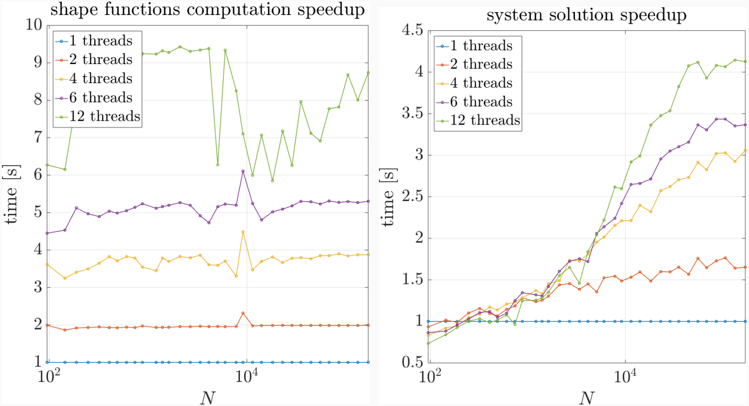
Recall the general procedure:

1. Discretize domain Ω – **development of parallel algorithm in progress**
2. Find neighborhoods $N(x_i)$ – **well explored**
3. Compute weights $\boldsymbol{w}^i$ for approximation of $\mathcal{L}$ over $N(x_i)$ – **trivial parallelization**
4. Assemble weights in a sparse system $Wu = f$ and solve it – offload the work to a suitable solver, currently **Pardiso**

or

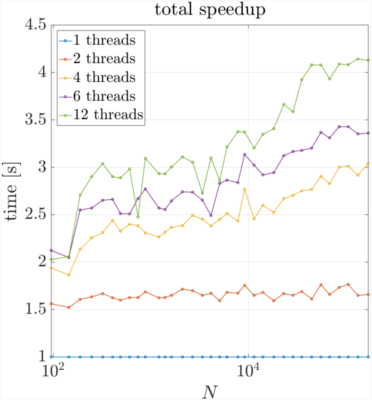
4. Use weights for explicit time iteration – **parallelization of the spatial loop**

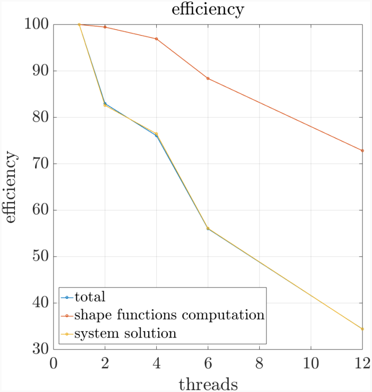
Speedup of shape function computation (left) and speedup of system solution (right)



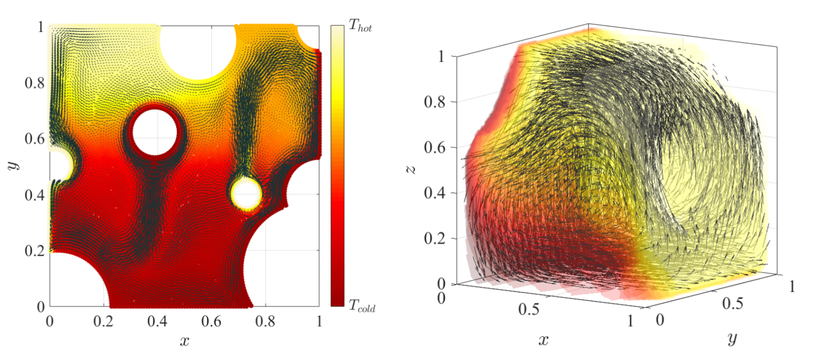
JSI

Total speedup (left) and efficiency (right)



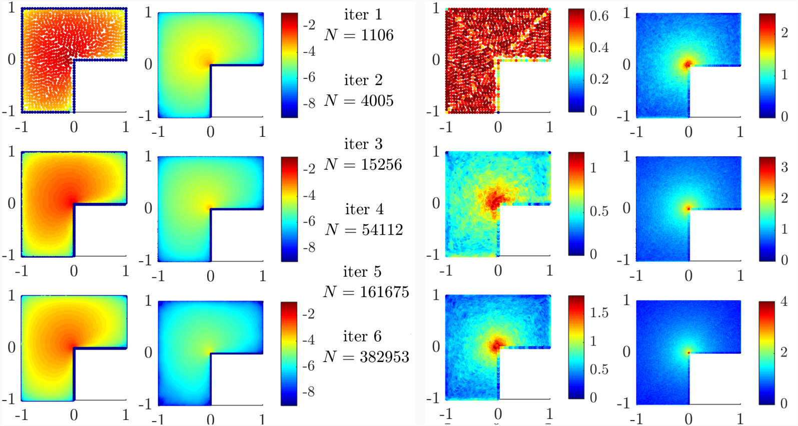


JSI



1
0.8
0.6
0.4
0.2
0
0
0.2
0.4
0.6
0.8
1
x
y
T_{hot}
T_{cold}
1
0.8
0.6
0.4
0.2
0
0
0.5
1
x
z
0
0.5
1
y

Further examples – adaptivity



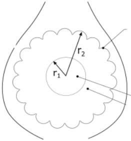
Further examples – overhead power line cooling

$$\rho^a \frac{\partial \mathbf{v}}{\partial t} + \rho^a \nabla \cdot (\mathbf{v}\mathbf{v}) = -\nabla P + \nabla \cdot (\mu^a \nabla \mathbf{v}) + \mathbf{b}$$

$$\nabla \cdot \mathbf{v} = 0$$

$$\rho^a c_p^a \frac{\partial T^a}{\partial t} + \rho^a c_p^a \nabla \cdot (T^a \mathbf{v}) = \nabla \cdot (\lambda^a \nabla T^a)$$

$$\mathbf{b} = \rho_{ref}\left[1 - \beta_T (T^a - T_{ref})\right]\mathbf{g}$$



Radiation Boundary condition

$$q_R = \sigma_B \varepsilon \left(T_{al}^4(r_2) - T_a^4\right) \left[\frac{\mathrm{W}}{\mathrm{m}^2}\right]$$

$$q_S = \frac{\alpha_s I_T}{\pi} \left[\frac{\mathrm{W}}{\mathrm{m}^2}\right],$$

Steel part – heat transport

$$T^{al}(r_1) = T^{st}(r_1)$$

$$c_p^{st} \rho^{st} \frac{\partial T^{st}}{\partial t} = \lambda^{st} \nabla^2 T^{st}$$

Aluminium part – heat transport and heat generation

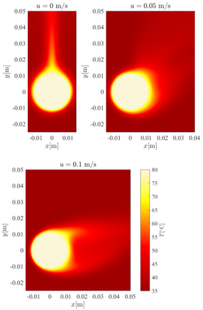
$$c_p^{al} \rho^{al} \frac{\partial T^{al}}{\partial t} = \lambda^{al} \nabla^2 T^{al} + q_J$$

$$q_J = \frac{I^2 R(T^{al})}{S^{al}} \left[\frac{\mathrm{W}}{\mathrm{m}^3}\right],$$

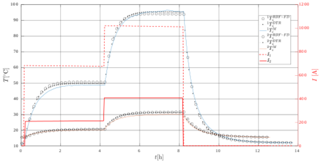
$$\lambda^{al} \left.\frac{\partial T^{al}}{\partial \mathbf{n}}\right|_{r_1} = \lambda^{st} \left.\frac{\partial T^{st}}{\partial \mathbf{n}}\right|_{r_1}$$

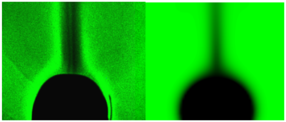
$$\lambda^{al} \left.\frac{\partial T^{al}}{\partial \mathbf{n}}\right|_{r_2} - \lambda^{a} \left.\frac{\partial T^{a}}{\partial \mathbf{n}}\right|_{r_2} = q_s + q_r$$

$$T^{al}(r_2) = T^a(r_2)$$



RBF-FD – numerical simulation, M - measurement





Result of Schlieren photography | Simulated temperature

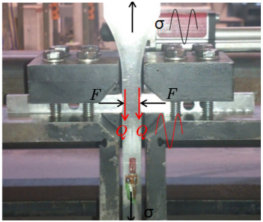
Experimental setup for measurements of conductor temperature

The project is funded by:

- ELES: transmitting energy, maintaining balance.

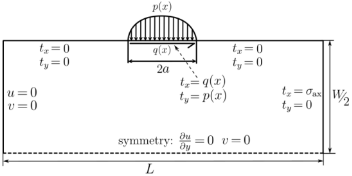
ELES

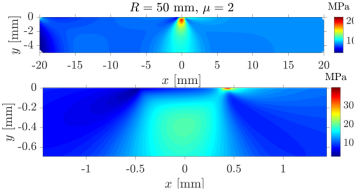
Further examples – fretting fatigue simulation

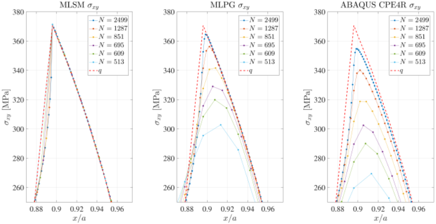


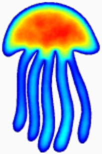
The project is funded by:

- Research Foundation – Flanders – (FWO)
- The Luxembourg National Research Fund (FNR)
- Slovenian Research Agency (ARRS)









Medusa

Coordinate Free Meshless Method implementation

http://e6.ijs.si/medusa/

http://e6.ijs.si/medusa/wiki/

Future work: (parallel) node generation on manifolds, adaptivity, better geometry support, domain decomposition

Slides available at http://e6.ijs.si/~jslak/talks/.

Thank you for your attention!

Acknowledgments: ARRS research core funding No. P2-0095 and Young Researcher program PR-08346.