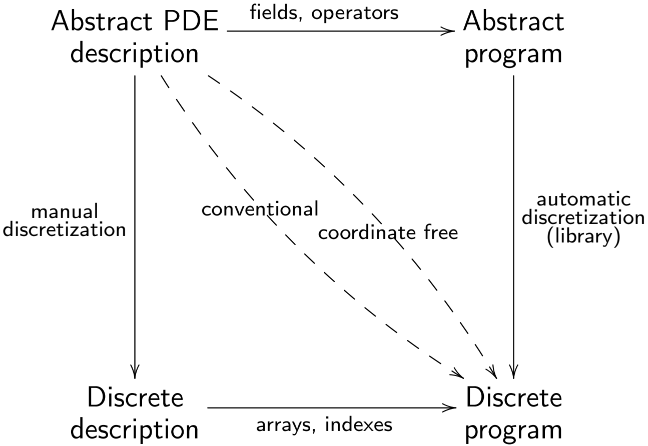
Parallel Coordinate Free Implementation of Local Meshless Method

Jure Slak, Gregor Kosec, Matjaž Depolli

"Jožef Stefan" Institute

21. 5. 2018



- Abstract programs are easier to understand and modify

Solving a boundary value problem:

$$u''(x) = \sin(x), \quad x \in [0, 1]$$
$$u(0) = 0, u'(1) = 0$$

Classical:

```
vector<Triplet<scalar_t>> ts;
for (int i = 1; i < N-1; ++i) {
    ts.emplace_back(i, i, -2/h/h);
    ts.emplace_back(i, i-1, 1/h/h);
    ts.emplace_back(i, i+1, 1/h/h);
    rhs(i) = std::sin(i*h);
}
rhs(0) = 0;
ts.emplace_back(0, 0, 1);
ts.emplace_back(N-1, N-3, -1./2/h);
ts.emplace_back(N-1, N-2, 2/h);
ts.emplace_back(N-1, N-1, -3./2/h);
rhs(N-1) = 0;
M.setFromTriplets(ts.begin(), ts.end());
VectorXd solution = M.lu().solve(rhs);
```

Coordinate free:

```
auto op = make_operators(sh, M, rhs);
for (int i : domain.internal()) {
    double x = domain.pos(i)[0];
    op.lap(i) = std::sin(x);
}
op.value(0) = 0;
op.neumann(N-1, {1}) = 0;
VectorXd solution = M.lu().solve(rhs);
```

General boundary value problem:

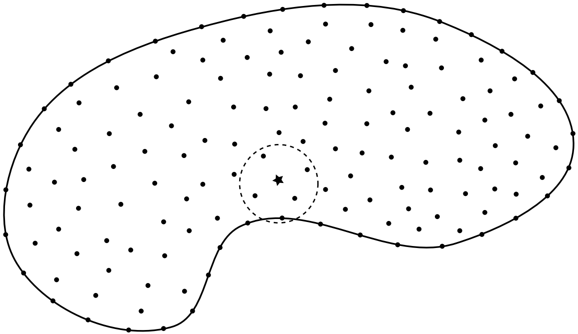
$$\mathcal{L}u = f, \text{ in } \Omega \tag{1}$$

$$\mathcal{R}u = g, \text{ on } \partial\Omega, \tag{2}$$

where $\Omega \subseteq \mathbb{R}^d$ is a domain u is an unknown scalar or vector field, $\mathcal{L}$ and $\mathcal{R}$ are linear partial differential operators and f and g are known functions.

- Discretize by choosing N points in the domain

Approximation of spatial partial differential operators:

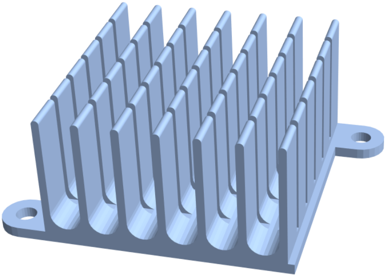


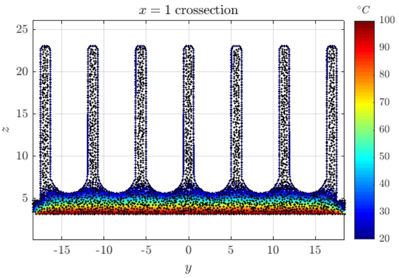
- Approximate $\mathcal{L}$ at a point p with function values on the neighborhood of p
- Example: $u''(x_i) = \frac{1}{h^2}u(x_{i-1}) - \frac{2}{h^2}u(x_i) + \frac{1}{h^2}u(x_{i+1})$
- $(\mathcal{L}u)p = \sum_{i=1}^{n} \chi_i u_i = \boldsymbol{\chi} \cdot \boldsymbol{u}$

Heat equation

$$-\alpha\nabla^2 u = q, \tag{3}$$

where q is the volumetric heat source and α is thermal diffusivity. Dirichlet boundary conditions $u|_{\partial\Omega} = u_0$ are considered below.
Heatsink:





Heat equation, $q = 0$, bottom set to $100^\circ C$, remainder to $20^\circ C$.

Linear elasticity

Lamé-Navier equation

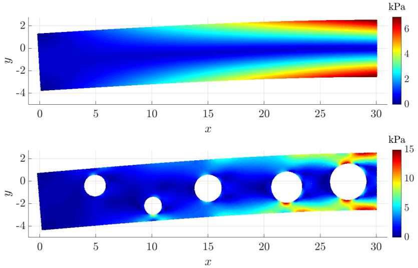
$$(\lambda + \mu)\nabla(\nabla \cdot \vec{u}) + \mu \nabla^2 \vec{u} = \vec{f} \tag{4}$$

describing displacements $\vec{u} = (u, v)$ and stresses σ, computed from $\vec{u}$ as

$$\sigma = \lambda(\operatorname{tr}\varepsilon)I + 2\mu\varepsilon, \quad \varepsilon = \frac{\nabla\vec{u} + (\nabla\vec{u})^{\mathsf{T}}}{2}.$$

Cantilever beam:

Beam bent at left end with a force P, along with "drilled" variant



Cantilever beam

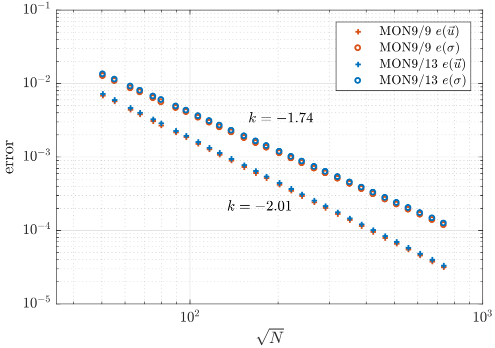
$$u = \frac{Py\left(3D^2(\nu+1)-4\left(3L^2+(\nu+2)y^2-3x^2\right)\right)}{24EI},$$
$$v = -\frac{P\left(3D^2(\nu+1)(L-x)+4(L-x)^2(2L+x)+12\nu xy^2\right)}{24EI},$$

```
for (int i : domain.internal()) {
    (lam+mu)*op.graddiv(i) + mu*op.lap(i) = 0;
}
for (int i : domain.bottom()) {
    op.traction(i lam, mu, {0, -1}) = 0;
}
for (int i : domain.top()) {
    op.traction(i, lam, mu, {0, 1}) = 0;
}
for (int i : domain.left()) {
    double y = domain.pos(i)[1];
    op.traction(i, lam, mu, {-1, 0}) = {0, -P*(D*D - 4*y*y)/(8*I))};
}
for (int i : domain.right()) {
    double y = domain.pos(i)[1];
    op.value(i) = {(P*y*(3*D*D*(1+v)-4*(2+v)*y*y))/ 24.*E*I),
                   -(L*v*P*y*y) / (2.*E*I))};
}
VectorXd solution = M.lu().solve(rhs);
```

Convergence

Errors are measured using analogues of the ℓ_∞ norms

$$e(\vec{u}) = \frac{\max_{x\in X}\{\max\{|u(x)-\hat{u}(x)|,|v(x)-\hat{v}(x)|\}\}}{\max_{x\in X}\{\max\{|u(x)|,|v(x)|\}\}},$$
$$e(\sigma) = \frac{\max_{x\in X}\{\max|\sigma(x)-\hat{\sigma}(x)|}{\max_{x\in X}\{\max|\sigma(x)|\}}$$



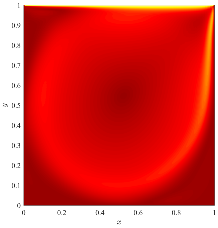
Fluid flow

The Navier-Stokes equations for incompressible flow are

$$\frac{\partial \vec{u}}{\partial t} + (\vec{u} \cdot \nabla)\vec{u} = -\nabla p + \frac{1}{\mathrm{Re}} \nabla^2 \vec{u}$$
$$\nabla \cdot \vec{u} = 0$$

where $\vec{u} = (u, v)$ is the flow velocity, Re is the Reynolds number and p is the pressure.

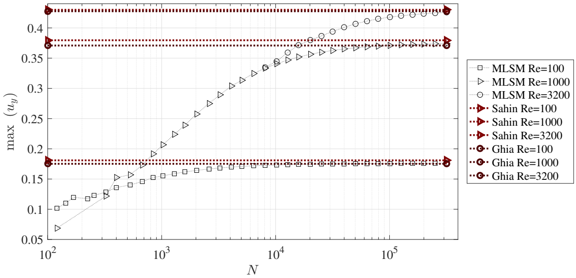
Lid driven cavity: Upper boundary has $\vec{v} = (0, 1)$, other boundaries fixed.



Lid driven cavity

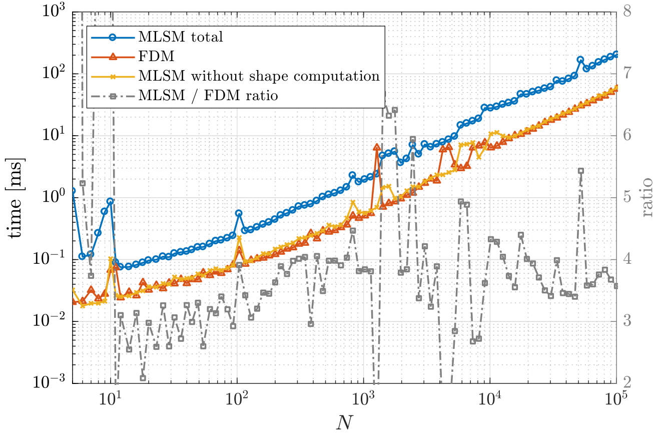
```
for (int i : domain.internal()) {
    1/dt * op.value(i) + (-mu/rho) * op.lap(i) + op.grad(i, v1[i]) =
        -1/rho * op.grad(p, i) + v1[i] / dt;
}
for (int i : domain.boundary()) {
    op.value(i) = {0, 0};
}
rhs[domain.top()] = {1, 0};
v_iter = solver.solveWithGuess(rhs, v_1);
```

Comparison against reference data:



Execution time

Comparison of Mesless Local Strong Form method and Finite Difference Method



- Time difference due to shape precomputation
- Shape computation is trivially parallelizable

Thank you for your attention!