Abstract

This paper explores a numerical solution of the three dimensional Boussinesq's problem. To obtain a solution, the Cauchy-Navier equation is numerically solved on a variable density nodal distribution with a meshless method. The method is implemented in C++ programming language with modularity in mind, enabling construction of many different meshless strong form approaches. It is demonstrated that such code can be efficiently employed to solve a complex problem in parallel, using a popular Radial Basis Function-generated Finite Differences meshless method. Results are presented in terms of von Mises stress field, convergence behaviour and execution analysis on dual CPU machine with total of 12 cores.

Keywords: Navier equation, RBF-FD, OpenMP, sparse system,

1 Introduction

This paper discusses a numerical solution of a contact problem from the field of linear elasticity, where the goal is to compute displacements within an isotropic half-space subjected to the concentrated normal traction [1]. Mathematically, the considered problem is described by a vector partial differential equation (PDE), namely the Navier-Cauchy equation, which often cannot be solved in a closed form and only numerical solution can be obtained. Traditionally, the Finite Element Method (FEM) [2] is used to do so; however, alternative approaches, referred to as meshless methods, emerged as a response to the problematic meshing of realistic 3D domains in FEM analysis [3, 4, 5].

The development of meshless methods began with weak form methods [6, 7], soon followed by a strong form variants [8, 3, 9]. The most important feature of the meshless methods is, as suggested by the name, that they do not require a mesh to oper-

ate. In a meshless setting, one has to deal with node positioning problem, which is considered a much easier task than meshing, but still not trivial [10, 11, 12]. In the most basic variant, meshless methods define all the relations between the nodes solely through relative inter nodal positions, however, more complex stencil selections can be used [13] to mitigate problems such as ill conditioning in non-smooth nodal configurations [14]. A popular variant of strong form meshless methods is the Radial Basis Function-generated Finite Differences (RBF-FD) method [15]. The RBF-FD can be seen, like many of the strong form meshless methods, as a generalization of the Finite Differences Method. The RBF-FD method is being actively researched [16, 17] and has been recently used also to solve linear elasticity problems [3].

In this paper we present a modular abstract C++ implementation of a strong form meshless method [18]. The presented implementation is a part of an open source project *Medusa* [19] and can be used to set up many different strong form methods or solve various problems in one, two, three or more dimensions. We use it here to solve the Boussinesq's problem in 3-D with RBF-FD on a machine with two Intel® Xeon® E5-2620 v3 6 core processors.

The rest of the paper is organized as follows: in section 2 the governing problem is presented, in section 3 RBF-FD methodology and its modular implementation is explained, in section 4 the results are discussed, and in section 5 the paper offers some conclusions and directions for future work.

2 The Boussinesq's problem

A deformation of elastic homogeneous isotropic half-space under a point load is considered. The problem is governed by Cauchy-Navier equation

$$(\lambda + \mu)\nabla(\nabla \cdot \vec{u}) + \mu\nabla^2\vec{u} = \vec{f}, \tag{1}$$

where $\vec{u}$ are unknown displacements, $\vec{f}$ is the loading force, and λ and μ are Lamé parameters, often expressed in terms of Young's modulus E and Poisson's ratio ν. Another important quantities are the stress tensor σ and strain tensor ε, related via Hooke's law as

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

where λ and μ are the same Lamé parameters as above and I is the identity tensor. The Boussinesq's problem is thoroughly described in [1], where also a closed form solution in cylindrical coordinates (r, θ and z) is given by

$$\begin{aligned}
u_r &= \frac{Pr}{4\pi\mu}\left(\frac{z}{R^3} - \frac{1-2\nu}{R(z+R)}\right), \qquad u_\theta = 0, \qquad u_z = \frac{P}{4\pi\mu}\left(\frac{2(1-\nu)}{R} + \frac{z^2}{R^3}\right), \\
\sigma_{rr} &= \frac{P}{2\pi}\left(\frac{1-2\nu}{R(z+R)} - \frac{3r^2 z}{R^5}\right), \qquad \sigma_{\theta\theta} = \frac{P(1-2\nu)}{2\pi}\left(\frac{z}{R^3} - \frac{1}{R(z+R)}\right), \\
\sigma_{zz} &= -\frac{3Pz^3}{2\pi R^5}, \qquad \sigma_{rz} = -\frac{3Prz^2}{2\pi R^5}, \qquad \sigma_{r\theta} = 0, \qquad \sigma_{\theta z} = 0,
\end{aligned} \tag{3}$$

where P is the magnitude of the point force and $R = \sqrt{r^2 + z^2}$ is the distance of given point from origin. This solution has a singularity at the origin. Numerically, we examine (2) with essential boundary conditions given by (3) on a finite domain

$$\Omega = [-1, -0.01] \times [-1, -0.01] \times [-1, -0.01] \tag{4}$$

Problem is schematically presented in Figure 1.

The stress field is computed in a post-processing by explicitly applying appropriate differential operators on the displacement field.

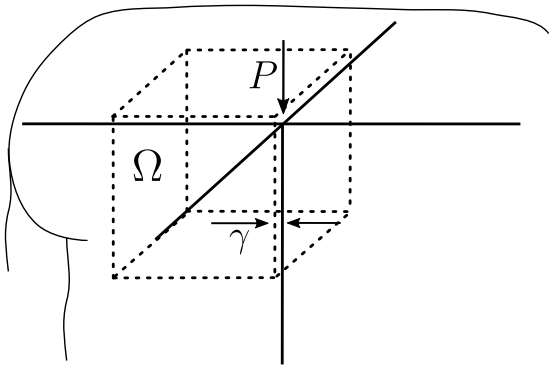

Figure 1: Boussinesq's problem and the computational domain Ω.

3 Solution procedure

The core of the strong form meshless method is the approximation of the considered field and its derivatives on overlapping *support domains*. To put it simply, for each discretization node, an approximation over a local support domain is constructed. This approximation is then used to create so-called shape functions that are used to compute approximations of differential operators from the field values. Shape functions can be computed with different approaches, e.g. least squares [20], collocation [21], augmented collocation [16], etc., nonetheless, in all cases a following form of approximation function is used

$$u(p) = \sum_{i=1}^{m} a_i B_i(p), \tag{5}$$

where a_i stand for unknown coefficients and B_i for *basis functions*. Now, consider a boundary value problem of form

$$\mathcal{L}u = f, \text{ in } \Omega \tag{6}$$

$$\mathcal{R}u = g, \text{ on } \partial\Omega, \tag{7}$$

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

of PDE is constructed in N points, also referred to as nodes, that are placed in the domain Ω, where N_i are in the interior and N_b are on the boundary. *Positioning of the nodes* is done in present work with a Poisson disk sampling based algorithm [12]. For all N_i internal points p, the operator $\mathcal{L}$ at point p is approximated over support domain of p as

$$\mathcal{L}|_p \approx \boldsymbol{\chi}_{\mathcal{L}}(p) \tag{8}$$

where $\boldsymbol{\chi}_{\mathcal{L}}(p)$ is a shape function for operator $\mathcal{L}$ at point p and only depends on the local geometry of the domain. The details on computation of $\chi_{\mathcal{L}}$ can be found in [3, 20]. After computing the shape functions the problem (6) is approximated by a linear equation

$$\boldsymbol{\chi}_{\mathcal{L}}(p) \cdot \boldsymbol{u} = f(p), \tag{9}$$

in all internal nodes, where $\boldsymbol{u}$ is the vector of unknown function values in support domain of point p and $\cdot$ denotes the dot product. Similar reasoning holds for the boundary nodes.

In a *system assembly* phase, operators are approximated in all nodes, which results in a $N \times N$ sparse system

$$\begin{bmatrix} \boldsymbol{\chi}_{\mathcal{L}}(p_1) \\ \vdots \\ \boldsymbol{\chi}_{\mathcal{L}}(p_{N_i}) \\ \boldsymbol{\chi}_{\mathcal{R}}(p_{N_i+1}) \\ \vdots \\ \boldsymbol{\chi}_{\mathcal{R}}(p_{N_i+N_b}) \end{bmatrix} \begin{bmatrix} u_1 \\ \vdots \\ u_N \end{bmatrix} = \begin{bmatrix} f(p_1) \\ \vdots \\ f(p_N) \end{bmatrix}. \tag{10}$$

In a final step, to obtain an approximation for function u at points p_i, above system is solved.

To ease the implementation of problem solution an additional abstraction *operators* is created, acting as an interface between the shape functions and the PDEs. The *operators* enable user to explicitly transform governing equations into the C++ code, as presented in for the Navier equation (1) in Listing 1.

The main solution procedure is typically followed by a post process that computes stresses from displacement field, i.e. explicitly applies first derivatives on the displacement field [3].

The strongest advantage of the presented method is that all building blocks, namely *nodes positioning*, *finding support domain*, *shape functions computation* , *basis functions*, *operators*, *system assembly*, and *system solution* are independent and can be therefore elegantly coded as abstract modules, not knowing about each other in the core of their implementation. Such approach offers great flexibility in the implementation of several different methods. For example, to construct a RBF-FD operators one combines RBF basis class with a collocation class, computes shapes and uses the computed shapes to construct operators, as presented in Listing 1. Note, that vector and scalar fields are implemented as plain arrays using a well developed linear algebra library [22] that also implements or otherwise supports various direct and iterative

linear solvers. In the present study we use *Pardiso* direct solver from the Inter® Math Kernel Library, which is capable of shared memory parallel execution. Please refer to our open source *Medusa* library [19] for more examples and features.

```
// RBF Gaussians basis
Gaussian<double> g(1.0);
// create RBF-FD approximation
RBFFD<Gaussian<double>, Vec3d> appr(g, 2);
// compute shapes
auto shapes = domain.computeShapes(RBFFD);
// construct implicit operators
auto op = shapes.implicitVectorOperators(M, rhs);

// assemble system
SparseMatrix<double, RowMajor> M(3*N, 3*N);
VectorXd rhs(3*N);
M.reserve(shapes.supportSizesVec());
for (int i : domain.interior()) {
    (lam+mu)*op.graddiv(i) + mu*op.lap(i) = 0.0;
}
for (int i : domain.boundary()) {
    op.value(i) = analytical(domain.pos(i));
}

solver.compute(M);
Vec3d u = solver.solve(rhs);
```

Listing 1: Implementation of Navier equation with use of operators.

4 Results

The presented implementation is demonstrated in a solution of The Boussinesq's problem with a RBF-FD setup [3], i.e. collocation with support size of 15 neighbouring nodes and a basis of 15 Gaussian RBFs with shape parameter 1.0. Values of $P = -1$, $E = 1$ and $\nu = 0.33$ were taken for physical parameters of the problem. First, the von Mises stress σ_v of the solution is presented in Fig. 2, computed as

$$\sigma_v = \sqrt{\frac{1}{2}[(\sigma_{11} - \sigma_{22})^2 + (\sigma_{22} - \sigma_{33})^2 + (\sigma_{33} - \sigma_{11})^2] + 3(\sigma_{12}^2 + \sigma_{23}^2 + \sigma_{31}^2)}. \quad (11)$$

To verify the solution the approximated values are compared against closed form solution. Comparison is presented in terms of relative difference between approximated and analytical values in L^∞ and L^1 norms for displacement field, denoted as $e_\infty(\vec{u})$ and $e_1(\vec{u})$, and in L^∞, L^1 norms for stress field, denoted as $e_\infty(\sigma)$, $e_1(\sigma)$ [3].

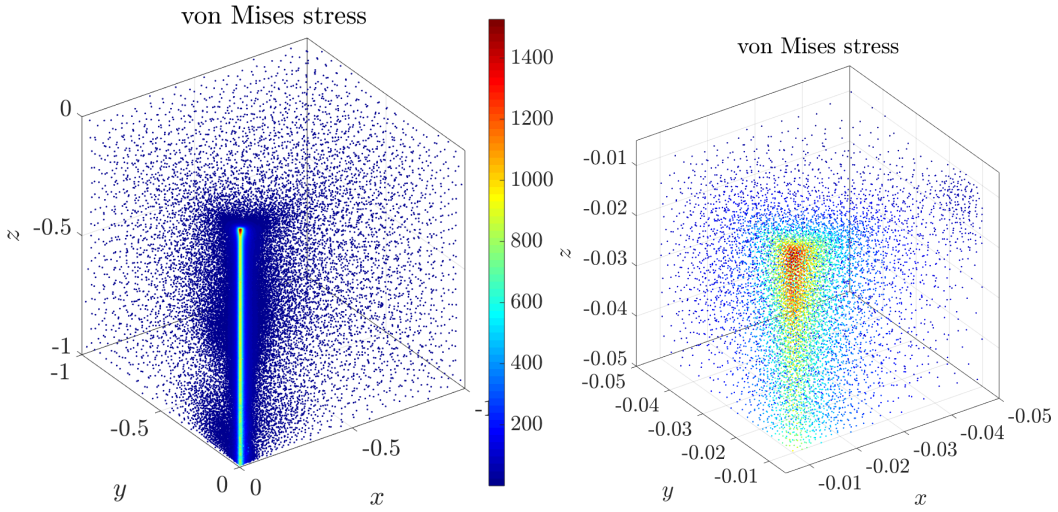

Figure 2: The numerically computed solution with an enlarged portion around the contact area. Solutions are presented in computation nodes coloured proportionally to the values of von Mises stress.

The error with respect to the number of computational nodes is presented in Figure 3. The solution exhibits expected convergence rates.

Next, we demonstrate the execution performance of presented solution procedure on the machine with two Intel® Xeon® E5-2620 v3 6 core processors. First, timings of different modules in sequential mode are analysed. In the left plot of Figure 4 an absolute time needed to complete certain modules with respect to the problem size is demonstrated, while on the right plot ratios of computational time for individual modules are shown. First, expected observation is that with increasing the problem size, the system solution dominates the execution time. In this study at around $N = 10^4$ nodes system solution becomes the most computationally expensive. However, using different sparse solver, for example *Pardiso 6.0* [23] would shift this breaking point towards higher N.

Next, a shared memory parallel performance is measured in terms of execution time, speedup $S_n = t_1/t_n$, where n is number of utilized cores, and efficiency $e_f = S_n/n$. In Figure 5, a total speedup (left) and efficiency (right) are presented, again with respect to the number of nodes. As expected, the total efficiency is primarily governed by efficiency of a system solution, since most of the time is spent there. The overall speedup is decent, if we take into account that only naive parallelization is implemented, i.e. the independent loops were executed in parallel using OpenMP `parallel for` with static scheduling, with no special care taken for thread affinity, which can have negative effect on the execution performance due to decreased cache hit ratios [21]. In Figure 6 speedups for two most expensive modules are presented, in the left plot a speedup of shape computation and in the right plot speedup of the system solution. The speedup of shape computation is good, especially when a low number of threads is utilized. With increasing the number of cores the speedup reduces, mainly as a result increasing cache invalidation [21]. The speedup of sys-

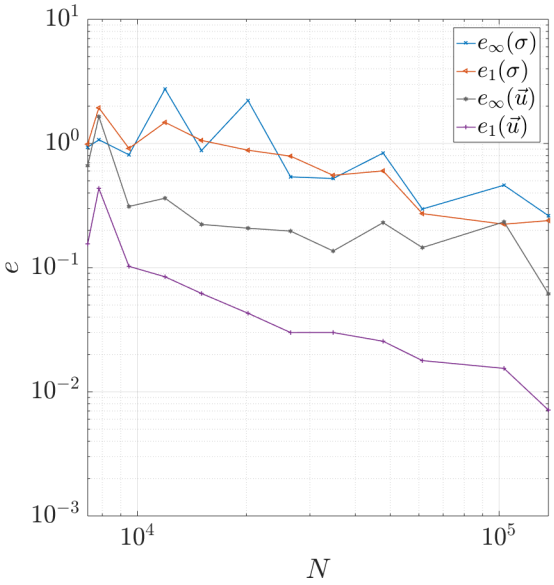

Figure 3: Convergence analysis of presented solution in L^∞, and L^1 norms for displacement and stress fields.

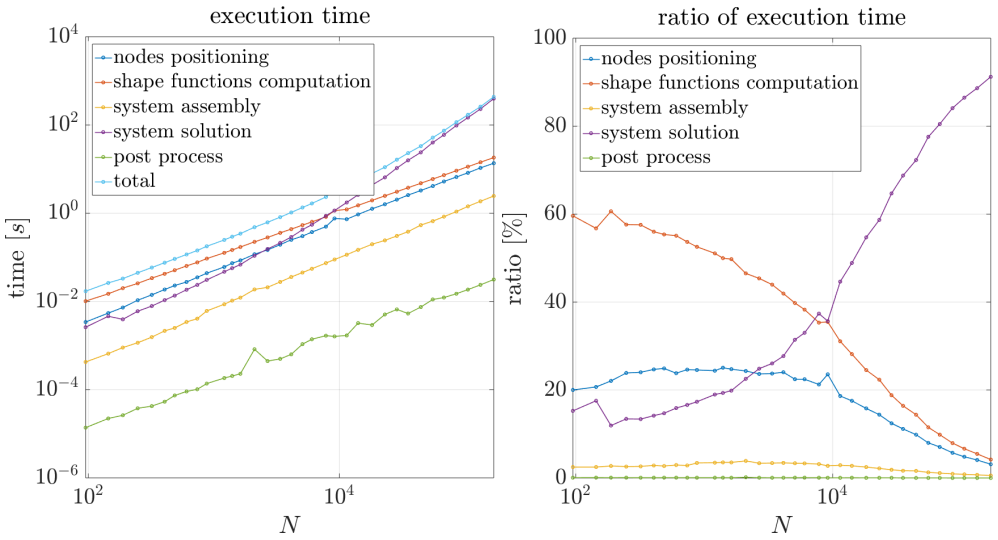

Figure 4: Execution times (left) and relative computational time (right) of individual modules.

tem solution is less regular and the parallelisation is only helpful wiht large N. The speedups might be improved with a different solver choice, however, of the tested solvers this one performed best.

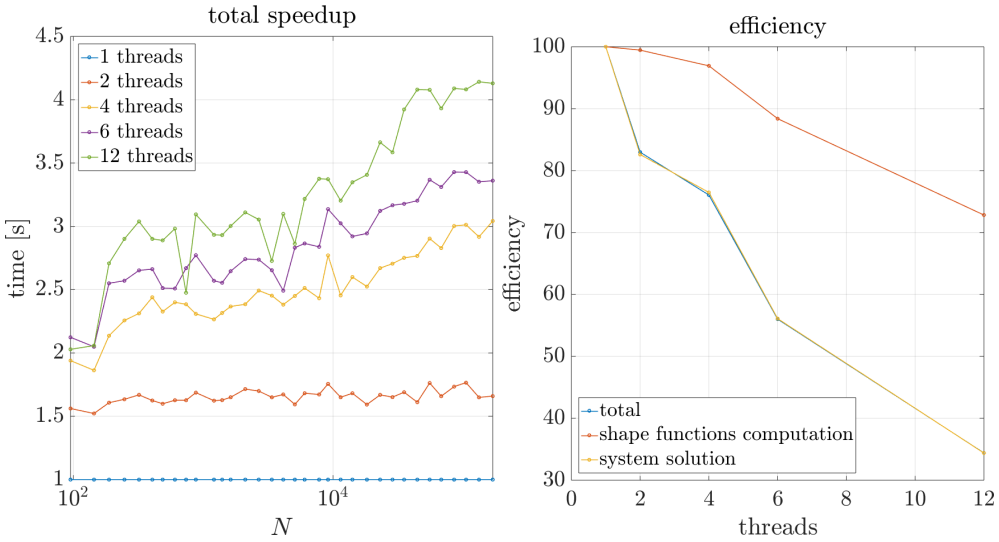

Figure 5: Total speedup (left) and efficiency (right)

5 Conclusions

In this paper we explore parallel performance of the RBF-FD solution procedure on the Boussinesq's problem. First, the modular solution procedure and its implementation is introduced. Although the presented implementation enables constructing of several different strong form meshless methods and it is not limited to a specific problem, a popular RBF-FD method and a 3-D linear elasticity problem is chosen for a benchmarking. The results of the RBF-FD method are evaluated by comparing them against the known closed form solution and exhibit satisfactory convergence. Timing of sequential code reveals that for linear stationary problems, the final system solution dominates the total execution time. However, this can rapidly change for non-linear problems or time transient simulations, where cumbersome part of the system solution is required only in the first iteration, while in consequent iterations precomputed decomposition is used. In case of explicit stepping this becomes even more pronounced, since the solution procedure does not use an implicit system. Nevertheless, in this paper we demonstrated that even in the worst case scenario, i.e. a linear stationary problem, the execution of a meshless solution can be accelerated with naive parallelization that can be trivially implemented in a proposed numerical framework. In future work we plan to enrich our numerical library with the domain decomposition module that will enable absolute freedom in parallel execution.

Acknowledgement

The authors would like to acknowledge the financial support of the Research Foundation Flanders (FWO), The Luxembourg National Research Fund (FNR) and Slovenian Research Agency (ARRS) in the framework of the FWO Lead Agency project:

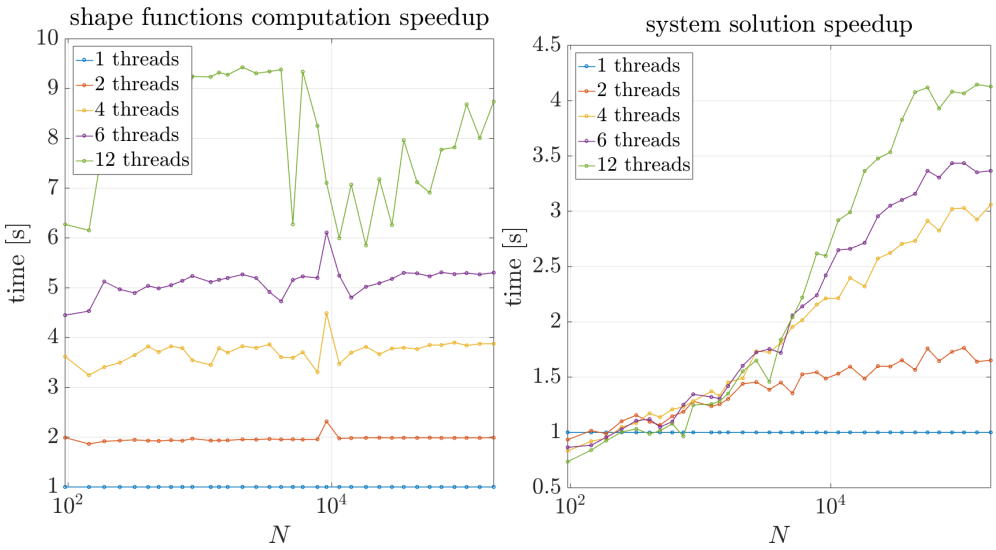

Figure 6: Speedup of shape function computation (left) and speedup of system solution (right)

G018916N Multi-analysis of fretting fatigue using physical and virtual experiments, and the ARRS research core funding No. P2-0095.

References

[1] W.S. Slaughter, *The Linearized Theory of Elasticity*, Birkhäuser Boston, 2002, ISBN 978-1-4612-6608-2, Pages 351–352.

[2] O.C. Zienkiewicz, R.L. Taylor, *The Finite Element Method: Solid Mechanics*, Butterworth-Heinemann, 2000.

[3] J. Slak, G. Kosec, "Refined Meshless Local Strong Form solution of Cauchy–Navier equation on an irregular domain", *Engineering Analysis with Boundary Elements*, 2018.

[4] Y. Chen, J.D. Lee, A. Eskandarian, *Meshless methods in solid mechanics*, Springer, New York, NY, 2006, page 200.

[5] G.R. Liu, Y.T. Gu, *An Introduction to Meshfree Methods and Their Programming*, Springer, Dordrecht, 2005.

[6] T. Belytschko, Y.Y. Lu, L. Gu, "Element-free Galerkin methods", *International Journal for Numerical Methods in Engineering*, 37(2): 229–256, 1994.

[7] S.N. Atluri, T. Zhu, "A new meshless local Petrov-Galerkin (MLPG) approach in computational mechanics", *Computational Mechanics*, 22(2): 117–127, 1998.

[8] E. Oñate, F. Perazzo, J. Miquel, "A finite point method for elasticity problems", *Computers & Structures*, 79(22–25): 2151–2163, 2001.

[9] B. Mavrič, B. Šarler, "Local radial basis function collocation method for linear thermoelasticity in two dimensions", *International Journal of Numerical Methods for Heat and Fluid Flow*, 25: 1488–1510, 2015.

[10] G. Kosec, "Stability analysis of a meshless method in irregular nodal distributions for flow problems", *International journal of computational methods and experimental measurements*, 5: 329–336, 2017.

[11] B. Fornberg, N. Flyer, "Fast generation of 2-D node distributions for mesh-free PDE discretizations", *Computers & Mathematics with Applications*, 69(7): 531–544, Apr 2015, ISSN 08981221.

[12] J. Slak, G. Kosec, "Fast generation of variable density node distributions for mesh-free methods", in A. Cheng, C.A. Brebbia (Editors), *Boundary elements and other mesh reduction methods XXXXI, 41st International Conference on Boundary Elements and other Mesh Reduction Methods, September 11–13, 2018, New Forest, UK*, Volume 122 of *WIT transactions on engineering sciences*, page ?? Wessex institute, WIT press, 2018.

[13] O. Davydov, D.T. Oanh, "Adaptive meshless centres and RBF stencils for Poisson equation", *Journal of Computational Physics*, 230(2): 287–304, 2011.

[14] T.A. Driscoll, A.R.H. Heryudono, "Adaptive residual subsampling methods for radial basis function interpolation and collocation problems", *Computers & Mathematics with Applications*, 53(6): 927–939, 2007.

[15] A.I. Tolstykh, D.A. Shirobokov, "On using radial basis functions in a "finite difference mode" with applications to elasticity problems", *Computational Mechanics*, 33(1): 68–79, 2003.

[16] B. Fornberg, N. Flyer, "Solving PDEs with radial basis functions", *Acta Numerica*, 24: 215–258, May 2015, ISSN 0962-4929, 1474-0508.

[17] V. Bayona, N. Flyer, B. Fornberg, G.A. Barnett, "On the role of polynomials in RBF-FD approximations: II. Numerical solution of elliptic PDEs", *Journal of Computational Physics*, 332: 257–273, 2017.

[18] J. Slak, G. Kosec, "Parallel coordinate free implementation of local meshless method", in K. Skala (Editor), *MIPRO 2018: 41st International Convention on Information and Communication Technology, Electronics and Microelectronics, May 21–25, 2018, Opatija, Croatia*, MIPRO proceedings, pages 194–200. IEEE, Croatian Society for Information and Communication Technology, Electronics and Microelectronics, 2018.

[19] "Medusa: coordinate free implementation of meshless methods", http://e6.ijs.si/medusa/.

[20] G. Kosec, "A local numerical solution of a fluid-flow problem on an irregular domain", *Advances in Engineering Software*, 120: 36–44, 2018.

[21] G. Kosec, M. Depolli, A. Rashkovska, R. Trobec, "Super linear speedup in a local parallel meshless solution of thermo-fluid problems", *Computers & Structures*, 133: 30–38, 2014.

[22] G. Guennebaud, B. Jacob, et al., "Eigen v3", 2010, URL: http://eigen.tuxfamily.org, accessed 2018-01-12.

[23] P.O. Schenk, et al., "Pardiso 6.0", 2018, URL: https://www.pardiso-project.org/, accessed 2019-02-02.