Radial basis function for mesh-to-mesh interpolation in parallel solving fluid-structure interaction problem

Sergey Kopysov, Igor Kuzmin, Alexander Novikov, Nikita Nedozhogin, and Leonid Tonkov

Institute of Mechanics, Ural Branch of the Russian Academy of Sciences

Abstract. In strongly coupled fluid-structure interaction simulations, the fluid dynamics and solid dynamics problems are solved independently on their own meshes. Therefore, it becomes necessary to interpolate physical properties (pressure, displacement) across two meshes. In this paper, we propose the acceleration of the interpolation process by using the matrix-free approach of the interpolation problem on GPU based on radial basis function method. Also, we reduce the number of equations of the system by using the adaptive algorithm of the choice of the interpolation points.

Keywords: parallel computing, hybrid HPC platforms, fluid-structure interaction, radial basis functions, layer-by-layer partitioning

1 Introduction

There are several approaches used to simplify algorithms for the numerical solving of the Fluid-Structure Interaction (FSI) problem. These include the reduction of the problem dimension, the algorithms that accelerate the computations, the analysis methods and the possibilities of using the properties of subtasks. Consider the reduction computational costs when interpolating data between non-matching meshes: reducing the data of interpolation; parallelization of algorithms, taking into account the geometry of extended boundaries in axisymmetric bodies.

Main interpolation methods on non-matching meshes for FSI simulations are overviewed in [1,5]. We consider the one based on radial basis functions (RBF) [2]. In this method, the coefficients of the interpolant are found from the system of equations whose matrix is formed using a radial basis function. The choice of the function determines the conditioning and density of the matrix, and as a result, the computational complexity of solving the system of equations.

The advantages of the RBF interpolation are the following:

- It does not require mesh-connectivity information.
- It requires solving a sparse system of equations, especially with the compact basis functions.

– It can be efficiently parallelized.

This paper is structured as follows. In Section 2, we briefly describe the RBF interpolation scheme for the FSI problem. In Section 3, we present new approach of reducing the problem size, based on layer-by-layer mesh partitioning. In Section 4, we describe a matrix-free solution of the interpolation problem on GPU.

2 RBF interpolation for FSI problems

Let P^d_{Q-1} is d dimension space of polynomial no more than $Q-1$ and $p_1, \ldots, p_q$ be a basis in this space. The main idea of the RBF method is to find the required interpolation function as a linear combination of the following functions:

$$w(x_i) = \sum_{j=1}^{n} \alpha_j \phi(||x_i - x_j||) + \sum_{l=1}^{q} \beta_l p_l(x_i), \quad 1 \le i \le n \ , \tag{1}$$

where q are additional degrees of freedom and the coefficients α_i and the polynomials $p_l(x_i)$ satisfy the following conditions:

$$\sum_{j=1}^{n} \alpha_j p_l(x_j) = 0, \quad 1 \le l \le q \ . \tag{2}$$

The solution of the system exists and is unique if

$$p(x_j) = 0, \text{ for all } 1 \le j \le n \text{ and } p \in P^d_{Q-1} \text{ valid } p = 0 \ . \tag{3}$$

The system of equations (1) - (3) is always solvable if ϕ is a positive-definite radial basis function.

Let Ω be the domain with the given pressure p_Ω. .The domain with the required pressure is denoted by Φ. The pressure interpolation between the mesh can be expressed in matrix form as follows:

$$\begin{bmatrix} W_{\Omega\Omega} & P_\Omega \\ P_\Omega^T & O \end{bmatrix} \begin{bmatrix} \alpha \\ \beta \end{bmatrix} = \begin{bmatrix} p_\Omega \\ 0 \end{bmatrix} \quad \text{or} \quad A\,\gamma = b \ , \tag{4}$$

here $W_{\Omega\Omega}$ is $n_\Omega \times n_\Omega$ matrix, consisting of the elements $\phi(\,||\mathrm{x}^i_\Omega - \mathrm{x}^j_\Omega||\,)$, $1 \le i, j \le n_\Omega$, P_Ω is column matrix consisting of the elements $\left[1\ \mathrm{x}^i_\Omega\right]$, α, β are the coefficients of the interpolant; n_Ω is the number of interpolation points of the domain. The target pressure vector p_Φ is obtained by matrix-vector product:

$$p_\Phi = [W_{\Phi\Omega} P_\Phi] \begin{bmatrix} \alpha \\ \beta \end{bmatrix} \ , \tag{5}$$

where $W_{\Phi\Omega}$ is $n_\Phi \times n_\Omega$ matrix, consisting of the elements $\phi(\,||\mathrm{x}^i_\Phi - \mathrm{x}^j_\Omega||\,)$, $1 \le i \le n_\Phi$, $1 \le j \le n_\Omega$, n_Φ is the number of interpolation points on the domain Φ,

P_Φ is column matrix $\left[1\ \mathrm{x}^i_\Phi\right]$. The dimension of the matrices P_Ω and P_Φ depends on the type of basis functions. For example, the dimension of the matrices for global radial basis function Thin-Plate Spline is $3 \times n_\Omega$ and $3 \times n_\Phi$, respectively.

Solving the system of the equations (4) is the most computationally expensive part of the interpolation. In [3], it was shown that the choice of basis functions affects both the quality of the interpolation and the solution time. The functions that provide more accurate interpolation may lead to large solution time. The computational cost can be optimized by (i) reducing the system and (ii) parallelizing the steps of preconditioning and solving sparse/dense systems of equations.

3 Reducing the size of the system of equations

In this section, we demonstrate the reducing the size of the system of equations for the fluid-structure interaction of supersonic flow with a nozzle wall that has a high geometric expansion ratio [11]. The boundary along which the computational data are interpolated is quite long and pressure is irregularly spaced along the boundary Ω (nozzle wall). We address these issues in the layer-by-layer mesh partitioning method proposed in our previous work [8]. The method provides a conflict-free data access during parallel summation of the components of finite element vectors in the shared memory of multi-core computing systems.

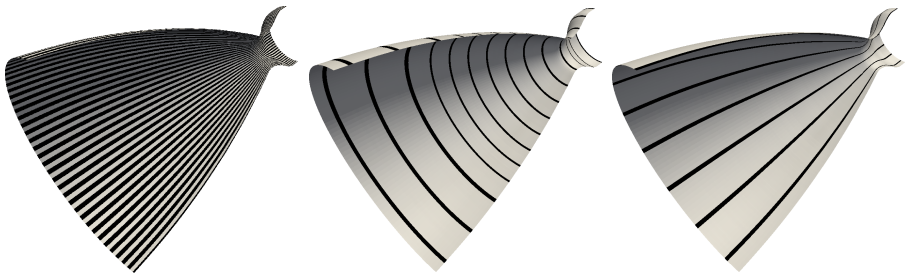
We divide the interface part $\Gamma_{\Omega^h} = \partial\Omega^h$ of the mesh Ω^h formed by the cell faces that belong to the surface Γ_Ω. To partition Γ_{Ω^h} into layers, we use the neighborhood criterion, where any two mesh cells are considered adjacent if they have at least one common node.

The considered physical area and the computational mesh are symmetrical. Therefore, the choice of the initial set of interpolation points is carried out in accordance with the distribution of the layers. Here, there are two possibilities for selecting layers: along the generatrix and along the directrix.

The mesh Γ_{Ω^h} is the discrete description of the rotation surface Γ_Ω with the closed directrix. To form layers in parallel to the directrix Γ_Ω (see Fig. 1 (b)) or along the surface generatrix (see Fig. 1 (c)), we use the algorithm proposed in [8].

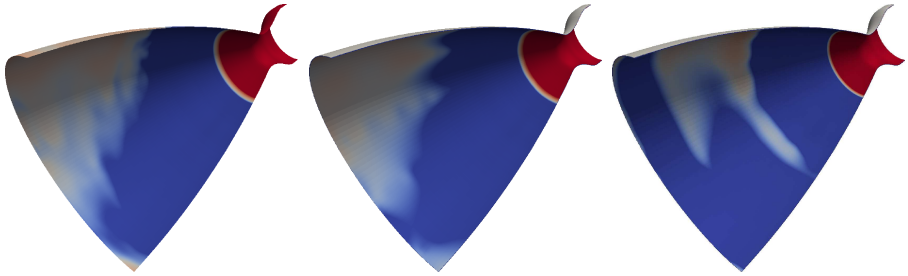
The layer-by-layer partitioning is used to reduce the number of interpolation points. To construct the interpolant, we choose those layers of the interface surface which most accurately represent the distribution of interpolated data (pressure). Fig. 1 shows the partitioning of the surface mesh into 150 layers. The dark layers correspond to 15 layers involved in the pressure interpolation (Fig. 1(b), (c)).

The quality of interpolation is compared for the local $\phi(||\mathrm{x}||) = 1 - ||\mathrm{x}||$, $\phi(||\mathrm{x}||) = (1 - ||\mathrm{x}||)^2$ and the global $\phi(||\mathrm{x}||) = e^{-||\mathrm{x}||^2}$, $\phi(||\mathrm{x}||) = ||\mathrm{x}||^2 \log ||\mathrm{x}||$ basis functions of Inverse Distance Weighting (IDW) [10], using different partitions and the numbers of layers. The quality of pressure interpolation can be estimated as the relative error computed by the ratio of the norms of the resultant forces of pressure on the interface boundary.



(a) (b) (c)

Fig. 1. (a) The interface boundary partitioned into 150 layers; (b) the partition into 15 layers parallel to the directrix; (c) the partition into 15 layers by the surface generatrix.



(a) (b) (c)

Fig. 2. The given pressure distribution at the boundary of the domain Ω (a); the resulting pressure distribution (TPS) using 15 layers from 150 by radial partition (b) and surface generatrix (c).

Table 1 shows the results for the pressure interpolation in parallel to the directrix (Radial partitioning) and along the surface generators (Longitudinal partitioning). In the second column ("all points"), we estimate the interpolation for all possible interpolation points from Γ_{Ω^h}.

The quality of the interpolation with the data reduction depends not only on the number of interpolation points but also on the choice of the points (Fig. 2). When compact basis functions in form $(1-||\mathrm{x}||)^2$ are used, the orientation of the pressure distribution after interpolation depends on the partitioning. The best interpolation is achieved for radial partitioning of the domain. The method of radial basis functions with the global basis function $\phi(\mathrm{x}) = ||\mathrm{x}||^2 \log ||\mathrm{x}||$ gives the best results. It allowed to reduce the number of equations (4) in 15 times, with the acceptable quality of the interpolation. Inverse distances weighting (IDW) interpolation gives the greatest error, even in the case of the full data.

The obtained matrix A is ill-conditioned. When using global basis functions, the condition number depends on the number and location of the interpolation points, as well as on the type of basis functions. The distance between interpolation points effect on condition number (Table 2). As the number of interpolation

Table 1. Relative error of the pressure interpolation, %

	all points	Longitudinal / Radial distribution			
n_{Γ_Ω}	28800	9600	5760	2880	960
$1 - \|\|x\|\|$	0.24	3.79/0.34	10.4/0.92	15.2/10.8	349.6/214.3
$(1 - \|\|x\|\|)^2$	0.78	7.79/1.58	69.3/9.34	74.3/15.9	365.4/36.5
$\mathrm{IDW}_{p=3}$	14.2	50.6/53.9	105/111	189/205	407.0/534.2
$e^{-\|\|x\|\|^2}$	0.54	1.18/1.07	2.68/3.99	8.04/5.41	147.6/101.6
$\|\|x\|\|^2 \log \|\|x\|\|$	0.01	0.23/0.01	1.07/0.21	5.95/1.13	28.61/21.6

Table 2. Condition number and relative error of the pressure interpolation, %

	Radial / Adaptive	Adaptive distribution			
n_{Γ_Ω}	condition, $\times 10^5$	9213	5595	2946	909
$1 - \|\|x\|\|$	9 / 10	1.23	1.14	0.93	4.89
$(1 - \|\|x\|\|)^2$	5 / 8	0.67	1.75	2.41	1360
$\mathrm{IDW}_{p=3}$	—	26.1	27.3	30.8	85.1
$e^{-\|\|x\|\|^2}$	4 / 5	0.94	1.12	0.82	9.61
$\|\|x\|\|^2 \log \|\|x\|\|$	10^5 / 10^6	0.37	1.25	1.31	3.65

points increases, the distance between them decreases, and the condition number increases. In this work, we consider the adaptive algorithm for choosing interpolation points, which affects both their number and distribution.

The interpolation error is not the only reason for using an adaptive algorithm. In the limit, as De Marchi showed [4], the optimal distribution of interpolation points is their homogeneous distribution. Note that in the case of asymptotically uniformly distributed interpolation points for some functions, theoretically, one can obtain an arbitrarily high error, but for others, this is not attainable. Even the use of greedy algorithms [9] does not provide the optimal data for RBF with different functions.

The aim of adaptation algorithm is achieved sufficiently small interpolation errors for n_Ω points while taking only n_a points, where $n_a \ll n_\Omega$. The adaptive algorithm starts on a very small number of points and then refines the data set by adding new points of interpolation where observed the interpolation error is largest. This algorithm returns the reduced points set, together with their associated coefficients that reproduce the interpolant function within a certain tolerance. Furthermore, limits on the number of iterations and on the total number of interpolate points (n_t) can be imposed.

Adaptive algorithm for selecting interpolation points

1. Select initial set of layers and solve for n_a init points

2. Form and solve the linear system of the interpolation coefficients on initial layers $A\gamma = b$
3. Evaluate the interpolant at all n_Ω data points $p_\Phi = A\gamma$
4. Compute the residual vector or find errors $E(\mathrm{x}) = (p(\mathrm{x}) - p_\Omega(\mathrm{x}))$
5. Check the stopping criteria, and if they are not satisfied, increase the iteration count and add new points with the largest $E(\mathrm{x})$
6. Adaptation successful done if the residual is smaller than a given tolerance

For the fluid dynamic problem about the supersonic nozzle flow with overexpansion, the line $\ell = \{(\arg\min_x p(x,\, \varphi_*), \varphi_*) \in \Gamma : 0 \le \varphi_* < 2\pi\}$ divide the set of interpolation points Ω into Γ_1, where $p = p(x)$ is the region of axial symmetry and Γ_2, where $p = p(x, \varphi)$ (Fig. 3, 4, 5 (a)).

Let apply the above adaptive algorithm to the pressure interpolation for the consider fluid-structure interaction problem. At the first step, we apply it to the layers to which the set of interpolation points is divided, that is, we shorten the set of interpolation points along the coordinate x. Thus, we leave from all layers the minimum quantity that satisfies the error of interpolation. At the second step, we apply this algorithm in each remaining layer to select interpolation points with the largest error in the coordinate φ. Thus, we reduce the number of interpolation points in both coordinates, preserving the interpolation error.

In addition to the pressure $p = p(x,\, \varphi)$ determined on Ω, the gradient $\operatorname{grad} p$ is also known. It is used as an indicator of adding additional interpolation points. Layers and interpolation points are added to the minimal set of points in the case of their location in the regions of the largest gradient.

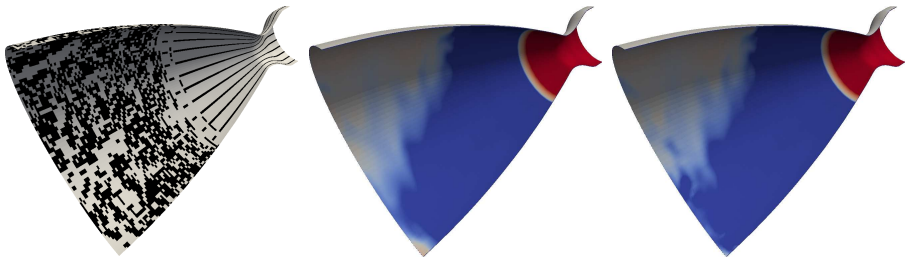
We also considered an adaptive algorithm without information about the gradient. In this case, the indicator of the interpolation error is obtained by local basis functions or by inverse distance method.

Tables 1, 2 shows the error of the pressure interpolation at the radial and adaptive distribution of interpolation points. We note that the adaptive choice of interpolation points is increased the condition number of the matrix of the system of equations (4), so that the iterative solution process converges slowly.

The adaptive distribution of the interpolation points using with local basis function $1 - ||x||$ has substantially reduced the error for any number of interpolation points. But, another local basis function $(1 - ||x||)^2$ has a large error for a small number of points. The Gaussian global function (Fig. 5 (b)) shows good results for a small number of adaptively chosen interpolate points $n_\Omega = 909$.

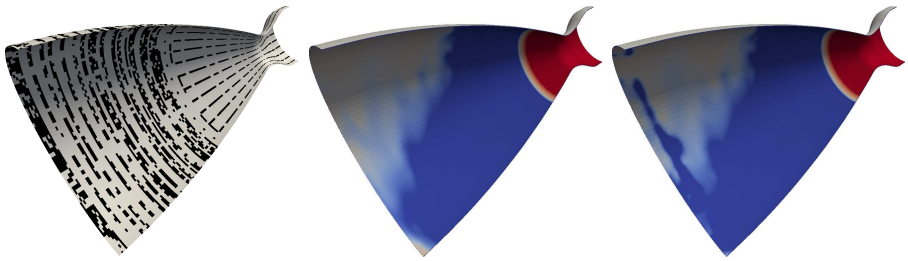
4 Matrix-free solution of the interpolation system on GPU

The one feature of the system (4) is a dense matrix A. This imposes restrictions on the possibility of using GPUs, which are linked with a small capacity of available GPU memory size. The resolution of this problem can be reached in several ways: (i) using several GPUs, thereby increasing the total memory available for the system solution; (ii) solving the system of equations without formations of a



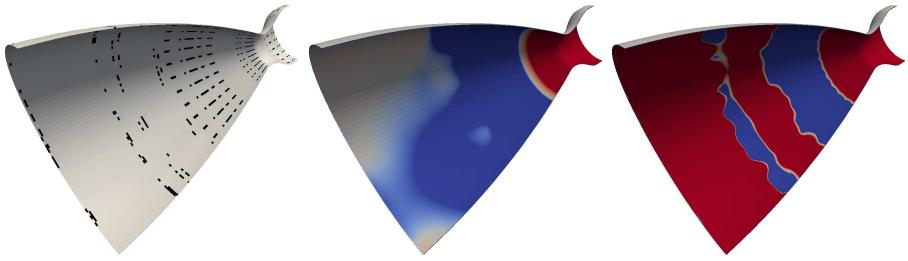
(a) (b) (c)

Fig. 3. Pressure distribution: (a) adaptive distribution with $n_\Omega = 9213$; (b) $e^{-||x||^2}$; (c) $1 - ||x||$.



(a) (b) (c)

Fig. 4. Pressure distribution: (a) adaptive distribution with $n_\Omega = 5595$; (b) $e^{-||x||^2}$; (c) $1 - ||x||$.



(a) (b) (c)

Fig. 5. Pressure distribution: (a) adaptive distribution with $n_\Omega = 909$; (b) $e^{-||x||^2}$; (c) $1 - ||x||$.

matrix (Free Matrix Algorithm). In this case, the matrix elements are computed as required their uses in an algorithm of the system solution. The solution of the system in the method of radial basis functions is possible without the formation of a matrix. It possible because the elements of the matrix are computed by the chosen basis function. This improves the data locality and arithmetic intensity for matrices and vectors. The memory requirements and CPU-GPU communica-

tions are reduced. The efficiency of the algorithm can be improved if multi-GPUs are used, similar to [6].

Consider in more detail computing expenses of FMA. Table 3 shows the time of sequential and parallel formation of a matrix A of the system (4). In the FMA, the formation time is excluded. For comparison, the time of the solution of the system with assembled matrix is given. Also, we presented the time of copying matrix to the GPU memory. The time of solving systems is presented for algorithms with a assembled matrix and a free matrix form.

CPU parallelization is carried out with OpenMP. The solution of the system of equations on several GPUs is carried out by CUDA in conjunction with OpenMP. The system of equations is solved by the conjugate gradient method with the diagonal preconditioner [6]. The precision is equal to 10^{-6}. In computations is used double-precision arithmetic. The analysis and performance estimations are performed on a computing node consisting of $2 \times \text{quad} - \text{core}$ Intel Xeon processor E5-2609, $2 \times$ GeForce GTX 980 with 4Gb GDDR.

The solution of system of the equations with the assembled matrix consists of several steps. In the case of using the CPU, the step of forming the matrix is added. While using the GPU, adds a step of copying data to the GPU memory. In the solution of the system using FMA, these steps are excluded. The total time of each approach, with necessary steps is given in the last line of the table 3.

The numerical computations show that the use of eight CPU threads within a one computing node reduces the solution time up to 7 times. Using of one GPU the speedup of solving the system is up to 250 times in respect to one CPU thread and up to 50 times in respect to $8 \times$ CPU. The GPU efficiency increases with the increase of the system dimension. The using of two GPUs reduced the time to 1.5 times in respect to the use of one GPU and 350 times in respect to the CPU. In this case, we can speak about strong scalability, which, with an increase in the number of GPUs, can be provided only with preserving of the sizes of the submatrices on each GPU.

The solution of the system in the matrix-free form using $8 \times$ CPU shortens the solution time to 2.5 times. However, the solution with the assembled matrix is twice as faster as in the matrix-free form. As for the GPU, in the case of using one GPU the operating time of the FMA is more than the solution of the system with the assembled matrix in 5 times. In the case of using two GPUs, this difference is reduced to 3 times. It is interesting to note that in the case of FMA the increase in the number of GPUs in twice has halved the time of the solving system. Using one GPU with FMA is obtained speedup about 55 times that CPU and 110 times in the case of two GPUs. The memory reduction reaches up to n_Ω times. Note that the use of local basis functions with the introduced radius of influence increases the efficiency of FMA.

Let us estimate the maximum size of the system, which can be solved using FMA on a one GPU. To forming the matrix A, the coordinates of the interpolation points are used. Then for interpolation in three-dimensional space, it is necessary to allocate memory to the vector of coordinates of length equal to

Table 3. The execution time of the interpolation for $e^{-||x||^2}$, s

		Number of equation			
		1202	8512	17984	28459
Forming of matrix A	$1 \times$ CPU	0.098	5.655	33.91	60.85
	$8 \times$ CPU	0.018	0.907	6.166	12.16
	FMA	0.0	0.0	0.0	0.0
Copy of matrix A to GPU	$1 \times$ GPU	0.009	0.312	0.842	—
	$2 \times$ GPU	0.006	0.125	0.433	1.393
	FMA	0.0	0.0	0.0	0.0
System solution $A\gamma = b$	$1 \times$ CPU	3.343	592.8	4732	10359
	$8 \times$ CPU	0.271	86.82	946.6	2273
	$8 \times$ CPU (FMA)	1.712	249.5	1489	4492
	$1 \times$ GPU	0.288	2.282	12.11	—
	$2 \times$ GPU	1.262	2.354	8.643	16.19
	$1 \times$ GPU (FMA)	0.471	14.01	91.59	191.1
	$2 \times$ GPU (FMA)	1.226	8.664	47.03	95.58
Total time	$1 \times$ CPU	3.438	598.5	4765	10419
	$8 \times$ CPU	0.288	86.82	952.7	2285
	$8 \times$ CPU (FMA)	1.712	249.6	1489	4492
	$1 \times$ GPU	0.317	3.53	19.12	—
	$2 \times$ GPU	1.283	3.38	15.24	29.74
	$1 \times$ GPU (FMA)	0.472	14.01	91.59	191.1
	$2 \times$ GPU (FMA)	1.226	8.664	47.03	95.58

$n_\Omega \times 3$. In the case of solving the system with the assembled matrix, the size of necessary memory is $n_\Omega \times n_\Omega$. The remaining vectors participating in the conjugate gradient method coincide for both algorithms. Thus, the size of memory to interpolate mesh data in three-dimensional space is reduced by $n_\Omega/3$ times. And the maximum system size solved by the FMA increases by this number of times. The algorithm of the conjugate gradient method with a diagonal preconditioner involves the use of memory to store a matrix of size $n_\Omega \times 3$ (FMA algorithm) and six vectors $n_\Omega \times 1$. Thus, using double precision arithmetic, to solve the system using the FMA algorithm, $n_\Omega \times (3 + 6) \times 8$ byte is necessary. Consequently, the maximum size of system for GPU with a 4Gb GDDR is about 6×10^8 equations. Using two graphics cards, the possible size of the system is increased to 1.2×10^9 equations. Thus, for dense matrices obtaining on the basis of global basis functions is constructed a parallel method of conjugate gradients. The computations are distributed among several GPUs. The using of the

matrix-free approach makes it possible to remove the limitations on the amount of memory.

5 Conclusion

We consider various approaches to reduce computing for mesh-to-mesh interpolation by the example interaction of supersonic flow with a nozzle wall. The using of adaptive data reduction based on the layer-by-layer partition of the mesh make it possible to reduce the number of interpolation points. At the same time, the quality of interpolation for the irregularly-spaced data is preserved, both for global and for local basis functions.

The solution of systems with dense matrices by the FMA algorithm on the CPU does not lead to significant time reductions. Since the time of matrix formation is less than 1% of the solution time, the use of FMA in conjunction with the CPU is inefficient.

A matrix-free approach is most effective when using a GPU, especially when it is not possible to achieve a large reduction of points without loss of quality of the interpolation. Using a GPU with a large number of system equations allows minimizing the cost of additional computations associated with the formation of matrix elements. At the same time, the reduction in the required memory is essentially and multiple of the dimensionality of the system.Further increase in the efficiency of FMA is associated with a decrease in the number of iterations of the algorithm of solving the system of equation by constructing effective parallel preconditioners. And also by using local basis functions with the radius of influence.

Acknowledgments. This work is supported by the Russian Foundation for Basic Research (projects: 16-37-00060-mol_a, 16-01-00129-a, 17-01-00402-a).

References

1. Berndt, M., Breil, J., Galera, S., Kucharik, M., Maire, P.H., Shashkov, M.: Two-step hybrid conservative remapping for multimaterial arbitrary Lagrangian–Eulerian methods. Journal of Computational Physics 230(17), 6664–6687 (2011)
2. De Boer, A., der Schoot, M.V., Bijl, H.: Mesh deformation based on radial basis function interpolation. Computer and Structures 85, 784–795 (2007)
3. De Boer, A., Van der Schoot, M.S., Bijl, H.: New method for mesh moving based on radial basis function interpolation. In: ECCOMAS CFD 2006: Proceedings of the European Conference on Computational Fluid Dynamics, Egmond aan Zee, The Netherlands, September 5-8, 2006. Delft University of Technology; European Community on Computational Methods in Applied Sciences (ECCOMAS) (2006)
4. De Marchi, S., Schaback, R., Wendland, H.: Near-optimal data-independent point locations for radial basis function interpolation. Advances in Computational Mathematics 23(3), 317–330 (2005)

5. Farrell, P., Piggott, M., Pain, C., Gorman, G., Wilson, C.: Conservative interpolation between unstructured meshes via supermesh construction. CMAME 198(33-36), 2632–2642 (2009)
6. Kopysov, S.P., Kuzmin, I.M., Nedozhogin, N., Novikov, A.K., Sagdeeva, Y.A.: Scalable hybrid implementation of the Schur complement method for multi-GPU systems. Journal of Supercomputing 69, 81–88 (2014)
7. Kopysov, S.P., Novikov, A.K., Nedozhogin, N., Rychkov, V.: Accelerating assembly operation in element-by-element FEM on multi-core platforms. Communications in Computer and Information Science 687 (2016)
8. Novikov, A., Piminova, N., Kopysov, S., Sagdeeva, Y.: Layer-by-Layer Partitioning of Finite Element Meshes for Multicore Architectures. Communications in Computer and Information Science 687, 106–117 (2016)
9. Rendall, T.C.S., Allen, C.B.: Efficient mesh motion using radial basis functions with data reduction algorithms. J. Comput. Phys. 228(17), 6231–6249 (2009)
10. Shepard, D.: A Two-dimensional Interpolation Function for Irregularly-spaced Data. In: Proceedings of the 1968 23rd ACM National Conference. p. 517–524. ACM '68, ACM (1968)
11. Wang, T.S., Zhao, X., Zhang, S.: Aeroelastic Modeling of a Nozzle Startup Transient. In: Journal of Propulsion and Power. vol. 30 (2013)