Efficient Parallel Solvers for the FireStar3D Wildfire Numerical Simulation Model

Oleg Bessonov[1(✉)] and Sofiane Meradji[2]

[1] Ishlinsky Institute for Problems in Mechanics RAS, 101, Vernadsky ave., 119526 Moscow, Russia
bess@ipmnet.ru

[2] IMATH, EA 2134, University of Toulon, Avenue de l'Université, 83957 La Garde, France
sofiane.meradji@univ-tln.fr

Abstract. This paper presents efficient parallel methods for solving ill-conditioned linear systems arising in fluid dynamics problems. The first method is based on the Modified LU decomposition, applied as a preconditioner to the Conjugate gradient algorithm. Parallelization of this method is based on the use of nested twisted factorization. Another method is based on a highly parallel Algebraic multigrid algorithm with a new smoother developed for anisotropic grids. Performance comparisons demonstrate superiority of new methods over commonly used variants of the Conjugate gradient method.

Keywords: Ill-conditioned linear systems · Conjugate gradient · Preconditioners · Multigrid · Smoothers · Parallelization

1 Introduction

The multi-physical FireStar3D numerical simulation model was developed in order to predict the behavior of wildfires at local scales (up to 500 m) [1,2]. This model consists of solving the conservation equations of a coupled system composed of vegetation and the surrounding gaseous medium. The model is able to account explicitly for all mechanisms of degradation of vegetation and various interactions between the gas mixture and the vegetation cover such as drag force, heat transfer by convection and radiation, and mass transfer.

Solving a three-dimensional nonstationary multi-physical problem requires significant computational resources. An appreciable part of the computational time is spent on solving large sparse linear systems arising from the discretization of partial differential equations in the above model [3].

The most popular iterative methods used to solve large linear systems are the Conjugate Gradient for symmetric matrices and its non-symmetric variants (BiCGStab, GMRES etc.) [4]. To accelerate convergence, these methods require

V. Malyshkin (Ed.): PaCT 2019, LNCS 11657, pp. 140–150, 2019.
https://doi.org/10.1007/978-3-030-25636-4_11

preconditioning [5]. There exists also a family of multigrid methods which possess very good convergence and parallelization properties [6,7].

The applicability of solvers depends on the nature of the underlying physical processes and on the speed of propagation of physical information. In particular, incompressible viscous fluid flows can be driven by three basic mechanisms with different propagation speeds:

- convection: slow propagation, Courant condition can be applied (one or few grid distances per time-step); using an iterative solver with few iterations;
- diffusion: faster propagation (tens grid distances per time-step), well-conditioned linear system; using an iterative solver with more iterations;
- pressure: instant propagation, ill-conditioned linear system; using an iterative solver with a robust preconditioner or a multigrid or a direct solver.

The choice of the solution method is determined by the above property. In the FireStar3D code, robust and efficient methods are used to solve the most time-consuming Poisson equation for pressure – the preconditioned Conjugate Gradient and the Algebraic multigrid. To solve the coupled system of convection-diffusion equations, for which a robust solver is not required, the BiCGStab method is applied.

In the previous papers [5,7,8], we analyzed various properties of iterative methods from the point of view of mathematics, convergence, efficiency and parallelization. In this paper we will consider the application of these methods for wildfire modeling, taking into account specific properties and requirements of the corresponding numerical simulation model.

The remaining part of the paper is organized as follows. Section 2 briefly presents the mathematical and geometric formulation of the FireStar3D model. Section 3 discusses the preconditioned Conjugate gradient method and describes the parallelization approach for the implicit MILU preconditioner. Section 4 introduces the multigrid method and describes a new smoother for anisotropic grids. Section 5 presents and analyzes the performance comparison results.

2 Mathematical Model

The mathematical model is based on a multiphase formulation [1]. It consists of two parts, that are solved on two distinct grids. The first part is described by the equations of the reacting turbulent flow in the gaseous phase, consisting of a mixture of fresh air with gaseous products resulting from the degradation of the solid phase and homogeneous combustion in the flaming zone. The second part consists of the equations governing the state and composition of the solid phase subjected to an intense heat flux coming from the flaming zone.

Solving the gaseous phase model consists in the resolution of conservation equations of mass, momentum, energy (in enthalpy formulation), and chemical species filtered using an unsteady RANS approach. Degradation of the vegetation is governed by three temperature-dependent mechanisms: drying, pyrolysis, and charcoal combustion.

The balance equations in the gaseous phase are solved numerically using the fully implicit finite volume method in a segregated formulation [9,10]. The Finite Volume discretization is applied to the non-uniform Cartesian staggered grid. The transport equations are solved by a fully implicit segregated method based on the PISO algorithm [11].

Figure 1 shows the computational domain of the wildfire numerical simulation model with two distinct grids [2].

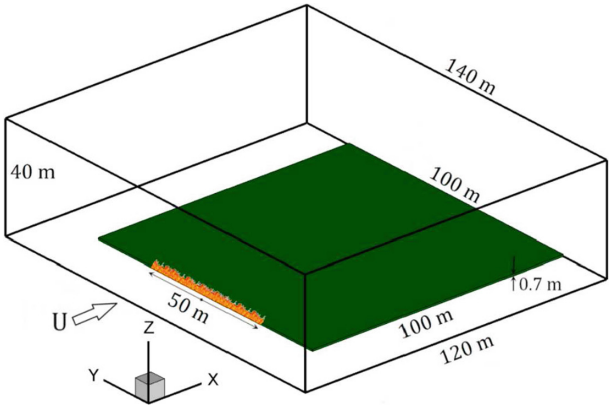

Fig. 1. Perspective view showing the computational domain and vegetation cover. The ignition line is shown on the left side of the vegetation cover

3 Preconditioned Conjugate Gradient Method

3.1 Explicit and Implicit Preconditioners

The original non-preconditioned Conjugate Gradient method (CG) [4] for solving a linear system $A\boldsymbol{x} = \boldsymbol{b}$ is simple to implement and can be easily parallelized. However, due to the explicit nature, it has a low rate of convergence and requires about $O(N)$ iterations, where N is the dimension of the problem in one spatial direction.

Because of this, the CG method is usually applied to the preconditioned linear system $(M^{-1}A)\boldsymbol{x} = M^{-1}\boldsymbol{b}$ where M is a symmetric positive-definite matrix that is "close" to the main matrix A (also symmetric and positive-definite). In practice, the system to be solved looks like $(L^{-1}AL^{-\mathrm{T}})\boldsymbol{x}^* = L^{-1}\boldsymbol{b}$ where $LL^{-1} = M$ (Incomplete LU decomposition), but in the preconditioned CG algorithm, only computations of the form $\boldsymbol{x} = M^{-1}\boldsymbol{z}$ or $M\boldsymbol{x} = \boldsymbol{z}$ are required [4].

Preconditioning works well if the condition number of the matrix $L^{-1}AL^{-\mathrm{T}}$ is much less than that of the original matrix A. The easiest way to reduce this condition number and speed up the convergence is to apply an "explicit" preconditioner ($B = M^{-1}$) than does not require the inversion of M (i.e. $\boldsymbol{x} = B\boldsymbol{z}$ is to be computed).

A good example of this kind is the polynomial Jacobi preconditioner [8], based on the truncated approximation series $1/(1-a) = 1 + a + a^2 + \ldots$

$$B = M^{-1} = \sum_{k=0}^{n} (H^k) P^{-1} \text{ where } P = \text{diag}(A),\ H = P^{-1}(P - A) = I - P^{-1}A$$

For $n = 0$, this expression degenerates into a diagonal preconditioner $B = P^{-1}$, which, due to its simplicity, is usually not considered as a true preconditioner. For $n = 1$, the Jacobi preconditioner looks like $B = (I + (I - P^{-1}A))P^{-1}$ and improves the acceleration rate twice (with some increase in computational complexity). This exactly corresponds to the expansion of the computational stencil in one iteration of the algorithm. Therefore, it can be easily applied and parallelized.

Unfortunately, neither kind of the simple explicit preconditioner can drastically improve convergence. The reason is that the explicit preconditioner acts locally using a stencil of limited size and propagates information through the domain with low speed. On the other hand, the implicit preconditioner, based on solving auxiliary linear systems, operates globally and propagates information almost instantly. Due to this, the implicit preconditioner works much faster and has a better than linear dependence of convergence on the geometric size of the problem. For this reason, to solve an ill-conditioned linear system, it is necessary to apply a preconditioner of the implicit kind.

In the FireStar3D code, the explicit Jacobi preconditioner is used to solve well-conditioned linear systems resulting from the discretization of a coupled system of convection-diffusion equations. Due to the non-symmetric nature of these linear system, the BiCGStab method is used.

3.2 Parallelization of the Implicit Preconditioner

The parallel properties of preconditioners are strongly dependent on how information is propagated in the algorithm. For this reason, it can be difficult to parallelize an implicit preconditioner, and a lot of effort is required to find the geometric and algebraic approach to parallelization. In particular, this applies to Incomplete LU-decomposition (ILU).

For the Cartesian computational domain, the geometric potential of parallelization can be revealed. The initial idea of the method is taken from the twisted parallelization of the tridiagonal linear system, when Gauss elimination is performed from both sides simultaneously. This idea can be naturally generalized to three dimensions. The resulting method is called "nested twisted factorization" [8,12].

In this method, the rectangular parallelepipedic domain is divided into 8 octants by separator planes (Fig. 2). In each octant, Gauss elimination is performed from the corner in the direction inwards independently in different threads (Fig. 2, left).

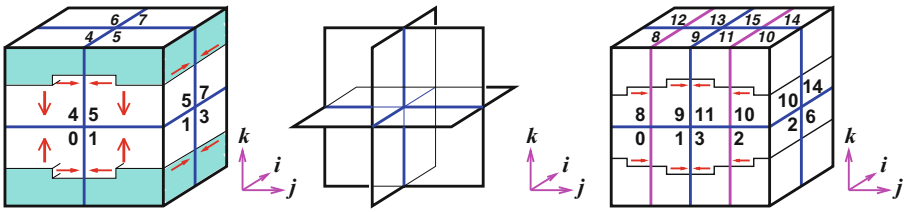

Fig. 2. Parallelization of the nested twisted factorization: illustration of the method (left); separator planes (center). Parallelization for 16 threads, staircase method (right)

After doing eliminations at internal octant points, they are performed in quadrants of separator planes in the same way (Fig. 2, center). Then, the intersection lines of the separator planes are processed and, finally, the solution is calculated at the central point. The following backsubstitution is performed in the reverse order, from the central point outwards.

Parallelization for 16 threads can be achieved by applying the staircase method shown on Fig. 2 (right). Here, each octant is divided into two halves in the direction `j` (see bottom left octant, divided between threads 0 and 1). Computations in the plane `(i,j)` for a certain `k` cannot be performed by thread 1 until they are completed by thread 0. However, they can be performed by a pipelined fashion: thread 1 computes the layer for some `k` at the same time when thread 0 computes the next layer for `k+1` (this looks like a step on the stairs). At the backsubstitution stage of the algorithm, the computations are performed in the reverse order.

Additional parallelization of the method for more threads seems to be impractical due to synchronization overhead. Nevertheless, this method can be used on a computer with more cores, since the performance of the algorithm is mainly limited by the memory bandwidth (i.e. the method belongs to the memory-bound class). Because of this, it is possible to implement a procedure for any reasonable number of threads, and not just for 8 or 16. To achieve this, it is necessary to distribute the active threads of the method (8 or 16) among all cores of the computing system, thus ensuring load balance. As a result of this modification, the method works well on up to 32 cores of a bi-processor computer.

The convergence of the ILU preconditioner depends on how the decomposition is calculated. The most accurate variant of the method, Modified ILU (MILU), requires about $O(N^{\frac{1}{2}})$ iterations, where N is the dimension of the problem in one spatial direction [8,13]. As a result, this algorithm becomes 5 to 6 times faster than the Conjugate gradient method with explicit Jacobi polynomial preconditioner.

3.3 Modified ILU Preconditioner for Periodic Boundary Conditions

The Modified ILU preconditioner can be mathematically strictly implemented and parallelized only for a rectangular parallelepipedic domain with non-periodic boundary conditions. In the case of periodic conditions, the algorithm becomes

not strict, and its convergence properties deteriorate. In particular, while the convergence estimate for a strict MILU is $O(N^{\frac{1}{2}})$ iterations, the loss of these properties leads to an estimate of $O(N)$ iterations.

However, in the problem under consideration, the flow properties in the periodic transverse direction are almost uniform with some fluctuations. For this reason, it becomes possible to use the original MILU preconditioner, which does not care on the periodic boundaries. This preconditioner is applied on the top of the Conjugate gradient algorithm with accurate treatment of the periodicity. This algorithm smooths the solution around periodic boundaries and maintains relatively fast convergence.

This new algorithm was implemented and tested. Its convergence with an accuracy 10^{-10} for a problem size $100 \times 200 \times 224$ is 68 iterations, compared with about 50 iterations of the original algorithms applied to a non-periodical problem of a similar size. This is much less than 350 iteration of the Conjugate gradient method with explicit Jacobi polynomial preconditioner. In term of the computational time, the new algorithm is about 4 times faster.

4 Algebraic Multigrid

The multigrid method is potentially the most efficient one for solving ill-conditioned linear systems because of its ability to suppress error components of all scales. Also, it can be parallelized to a large number of threads. This method solves differential equations using a hierarchy of discretizations.

In one multigrid cycle (V-cycle, Fig. 3), both short-range and long-range components of the error are smoothed out, so information is instantaneously transmitted throughout the domain. As a result, this method becomes very efficient for elliptic problems that spread physical information infinitely fast.

1	Pre-smooth	$x_1 = S_1(x_0, b)$
2	Residual	$b_1 = b - Ax_1$
3	Restriction	$\tilde{b}_1 = R\, b_1$
4	Next level	$\tilde{A}\tilde{x}_2 \approx \tilde{b}_1$
5	Prolongation	$x_2 = P\tilde{x}_2$
6	Correction	$x_3 = x_1 + x_2$
7	Post-smooth	$x_0 = S_2(x_3, b)$

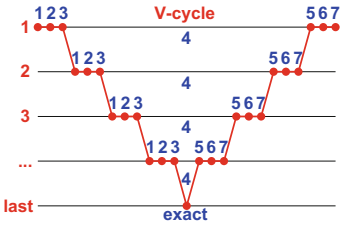

Fig. 3. Scheme of the multigrid algorithm (left); illustration of the V-cycle (right)

In the FireStar3D code, the Algebraic multigrid (AMG) approach [6,7] is applied. This method is based on matrix coefficients rather than on geometric parameters of the domain. The main computational operations in the multigrid cycle are smoothing (usually an iteration of the Gauss-Seidel or SOR method) and, to a lesser extent, restriction (fine-to-coarse grid conversion by averaging) and prolongation (coarse-to-fine conversion by interpolation).

4.1 Smoothers for Anisotropic Grids

The multigrid is a very efficient method, its convergence does not depend on the problem size. However, it does not perfectly work on anisotropic grids (with cells that have a high aspect ratio). The reason is that the typically used smoothing procedure (Gauss-Seidel or SOR) effectively suppresses error components only along the shortest cell dimension. In the considered problem, the cell aspect ratio reaches 15:1. Because of this, the traditional approach leads to extremely slow convergence (up to 300 iterations against a typical value of the order of 10).

There are several approaches to resolve this problem. The most straightforward method is semi-coarsening [6]. However, after applying this procedure, the grid becomes non-structured, and the overall method becomes very complex and numerically less efficient. Another method is based on the use of incomplete matrix factorization as a smoother [14], which improves the performance and convergence of the multigrid. Other approaches originate on building a more robust smoother that is not sensitive to grid anisotropy [15]. They are based on the replacement of point relaxation methods with plane relaxation ones.

If the grid cells are compressed in a single spatial direction, it becomes possible to apply the line Gauss-Seidel (line GS) or the line SOR smoothing procedure in this direction. The idea is to solve the GS or SOR equation for the full line of grid points, rather than separately for each grid point. As a result, the smoothing of error components along the longest cell dimension is not suppressed. The new procedure requires solving a tridiagonal linear system along a compressed direction and, therefore, is slightly more expensive than the standard one.

It was found that the line smoother successfully solves the above problem, but it is not efficient enough to smooth the error components in the remaining part of the domain. To improve the convergence, this procedure was supplemented by standard (point) Gauss-Seidel or SOR smoother, which costs less. The above approach was applied for all levels of the multigrid algorithm.

To achieve good convergence, it is necessary to determine the optimal over-relaxation parameters for SOR procedures. These values depend on the size and configuration of the grid. For the first (finest) grid level, the optimal values are about 1.3–1.4 for line smoothers and about 1.6–1.65 for point smoothers. For the upper (coarser) levels, a plain GS is used as a line smoother, while the optimal values for point smoothers are about 1.6–1.9.

The application of over-relaxation reduces the number of iterations from 40–50 to 10–11 (for grid sizes up to $100 \times 200 \times 504$ and relative accuracy 10^{-10}).

4.2 Parallelization of Smoothers

An iteration of the Gauss-Seidel or SOR method looks like an implicit procedure: $(D + L)\boldsymbol{x}_{k+1} = b - U\boldsymbol{x}_k$ (here D, L and U are diagonal, lower and upper parts of the matrix A in the equation $A\boldsymbol{x} = \boldsymbol{b}$). To avoid dependences that prevent parallelization, a multicolor grid partitioning is required. For the first level of the grid with 7-point stencils, a two-color (red-black) scheme can be used. In this scheme, the procedure is divided into two explicit steps: $D^{(1)}\boldsymbol{x}^{(1)}_{k+1} = \boldsymbol{b}^{(1)} - U\boldsymbol{x}^{(2)}_k$

and $D^{(2)}\boldsymbol{x}_{k+1}^{(2)} = \boldsymbol{b}^{(2)} - L\boldsymbol{x}_{k+1}^{(1)}$ (superscripts $^{(1)}$ and $^{(2)}$ refer to red and black grid points, respectively). After that, elements with the same color can be processed independently, and, as a consequence, parallel splitting can be applied.

For line smoothers, red-black partitioning is applied to whole lines.

For the upper levels of the grid with 27-point stencils, a 4-color scheme is used, also applied to whole lines.

Multicolor processing of the computational domain can be performed in several passes according to the number of colors. However, each pass needs access to all the elements of the data arrays. Since the performance of the algorithm depends primarily on the memory access rate, this proportionally increases the computational time.

To reduce the number of passes, it is necessary to somehow combine the processing of different colors, while retaining the property of a multicolor scheme. The idea of the combination technique is illustrated in Fig. 4. Shown here are the cross-sections of the computational domain perpendicular to the compressed direction (i.e. the direction where the line GS or SOR is applied). The proposed idea is expressed in terms of rows and columns assuming that, in lexicographic order, rows are processed first.

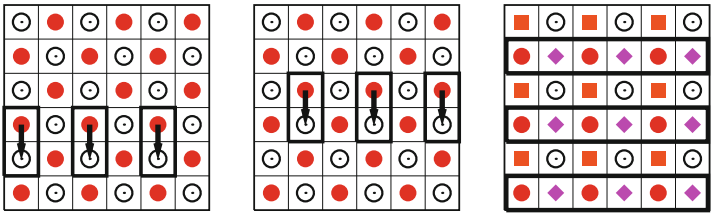

Fig. 4. Illustration of a multicolor smoothing procedure: alternating iterations of the red-black pass (left, center); single pass of the 4-color case (right) (Color figure online)

For the red-black case, processing is performed in a single pass with alternating iterations. At even iterations of the pass (Fig. 4, left), adjacent pairs of red and black elements of even columns are calculated (first red, then black). At odd iterations (Fig. 4, center), similar pairs of elements of odd columns with a row number increased by one are processed (in the same order).

For the 4-color case, two passes are required (Fig. 4, right) – one pass for even rows and another pass for odd ones. Within each pass, two sub-passed are performed – one for each color in a row. The second sub-pass does not require costly memory accesses, since most of the data is cached after the first sub-pass.

Multicolor partitioning allows to implement in the shown cross-section any splitting of the computational domain required for parallelization.

The above technique ensures regular and efficient memory accesses as an important requirement of computational efficiency. It is supplemented by the vectorization of arithmetic operations (in frame of the AVX vector extension) and by other optimizations.

5 Performance Comparison

The convergence and performance of the new solvers were evaluated using matrices and data taken from the typical runs of the FireStar3D code (Table 1). The first matrix corresponds to a larger problem with the periodic boundary conditions for the second spatial dimension. The second matrix was taken from a smaller problem with non-periodic boundary conditions (see Fig. 1 for the geometric illustration of this problem).

The tests were conducted on a cluster node built on two 16-core Xeon Gold 6142 processors at the Mesocentre computer center (Marseille, France). In addition to the new solvers described in the paper (Algebraic multigrid and MILU-preconditioned Conjugate gradient), two variants of the Conjugate gradient method were tested – one with explicit Jacobi preconditioner and another one with simple diagonal scaling. Results are presented for parallel runs on 32 cores of a cluster node with the relative accuracy 10^{-10}, time for solving a linear system is shown in seconds.

Table 1. Comparison of convergence and performance of different solvers

Matrix size	AMG		CG MILU		CG Jacobi		CG diag	
	Iter.	Time	Iter.	Time	Iter.	Time	Iter.	Time
100 × 200 × 504	10	0.440	48	0.788	267	3.15	541	5.02
100 × 248 × 224	11	0.293	46	0.398	317	2.06	638	3.22

It can be seen that the multigrid solver is about 11 times faster than the plain Conjugate gradient (CG diag). Using the explicit Jacobi preconditioner makes the CG method 1.5 times faster due to more optimal structure of the algorithm, but does not change its convergence properties, so the number of iterations still depends linearly on the largest dimension of the discretized problem.

Compared to the Conjugate gradient method with the MILU-preconditioner, the multigrid solver is 36% faster for the problem with non-periodic boundary conditions and 79% faster for the problem with a periodicity. The latter can be explained by two properties of the CG MILU method – sensitivity to the size of the problem (as opposed to the multigrid) and some decrease in convergence due to explicit treatment of periodic boundary conditions.

Another advantage of the multigrid method is better scalability. In particular, the speedup for this method for 32 threads ranges from 15 to 17 (depending on the size of the matrix), while for CG MILU it is at the level of 10–11. For both methods, the speedup is limited by the memory bandwidth, but the second method is more memory-bound than the first one. In addition, CG MILU parallelization is limited to 16 threads.

For these reasons, the multigrid method is more preferable for using in the FireStar3D code. On the other hand, variants of the CG MILU method do not require adjustment of the parameters of over-relaxation, as is necessary for the multigrid. Therefore, it may happen to be more robust for some problems. Thus, this method can still be applicable, at least, for running on computer systems with a smaller number of processor cores.

There is another promising approach, Algebraic multigrid as a preconditioner for the Conjugate gradient method. At the moment, it does not benefit when solving linear systems in FireStar3D runs, unlike happens in other fluid dynamics problems [7]. However, in the future this approach will be examined more carefully in order to achieve faster convergence and lower computational costs.

6 Conclusion

In this paper, we presented two parallel methods for solving ill-conditioned linear systems arising from the discretization of partial differential equations as applied to the FireStar3D wildfire numerical simulation model.

The first of the presented methods is based on a parallel MILU-preconditioned Conjugate gradient algorithm. This method has been extended to run on any number of processor cores and to support periodic boundary conditions. The second method is based on an Algebraic multigrid. It uses a new smoothing algorithm that can work with highly compressed grids. This smoother is optimally parallelized using multicolor grid partitioning and a special processing scheme.

New methods were used to build efficient parallel solvers for the FireStar3D code. Both solvers were evaluated using data taken from the production runs of the code. They demonstrate robustness and superiority over the widely used variants of the Conjugate gradient method. In particular, the multigrid solver is more than ten times faster than the diagonally scaled Conjugate gradient solver.

Of these two methods, the multigrid algorithm is faster and more scalable for large number threads. On the user hand, it requires some tuning to achieve faster convergence. For this reason, MILU-based methods remain attractive because of their robustness and therefore can be used for running with fewer threads.

Acknowledgements. This work was supported by the Russian State Assignment under contract No. AAAA-A17-117021310375-7. The work was granted access to the HPC resources of Aix-Marseille Université financed by the project Equip@Meso (ANR-10-EQPX-29-01) of the program Investissements d'Avenir supervised by the Agence Nationale pour la Recherche (France).

References

1. Morvan, D., Accary, G., Meradji, S., Frangieh, N., Bessonov, O.: A 3D physical model to study the behavior of vegetation fires at laboratory scale. Fire Saf. J. **101**, 39–53 (2018). https://doi.org/10.1016/j.firesaf.2018.08.011

2. Frangieh, N., Morvan, D., Meradji, S., Accary, G., Bessonov, O.: Numerical simulation of grassland fires behavior using an implicit physical multiphase model. Fire Saf. J. **102**, 37–47 (2018). https://doi.org/10.1016/j.firesaf.2018.06.004
3. Saad, Y.: Iterative Methods for Sparse Linear Systems. PWS Publishing, Boston (2000)
4. Shewchuk, J.R.: An Introduction to the Conjugate Gradient Method Without the Agonizing Pain. School of Computer Science, Carnegie Mellon University, Pittsburgh (1994)
5. Bessonov, O.: Parallelization properties of preconditioners for the conjugate gradient methods. In: Malyshkin, V. (ed.) PaCT 2013. LNCS, vol. 7979, pp. 26–36. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-39958-9_3
6. Stüben, K.: A review of algebraic multigrid. J. Comput. Appl. Math. **128**, 281–309 (2001). https://doi.org/10.1016/S0377-0427(00)00516-1
7. Bessonov, O.: Highly parallel multigrid solvers for multicore and manycore processors. In: Malyshkin, V. (ed.) PaCT 2015. LNCS, vol. 9251, pp. 10–20. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-21909-7_2
8. Accary, G., Bessonov, O., Fougère, D., Gavrilov, K., Meradji, S., Morvan, D.: Efficient Parallelization of the preconditioned conjugate gradient method. In: Malyshkin, V. (ed.) PaCT 2009. LNCS, vol. 5698, pp. 60–72. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-03275-2_7
9. Patankar, S.V.: Numerical Heat Transfer and Fluid Flow. Hemisphere Publishing, New York (1980)
10. Versteeg, H., Malalasekera, W.: An Introduction to Computational Fluid Dynamics: The Finite Volume Method. Prentice Hall, Harlow (2007)
11. Moukalled, F., Darwish, M.: A unified formulation of the segregated class of algorithms for fluid flow at all speed. Numer. Heat Transf. Part B **37**, 103–139 (2000). https://doi.org/10.1080/104077900275576
12. van der Vorst, H.A.: Large tridiagonal and block tridiagonal linear systems on vector and parallel computers. Parallel Comput. **5**, 45–54 (1987). https://doi.org/10.1016/0167-8191(87)90005-6
13. Gustafsson, I.: A class of first order factorization methods. BIT **18**, 142–156 (1978). https://doi.org/10.1007/BF01931691
14. Axelsson, O.: Analysis of incomplete matrix factorizations as multigrid smoothers for vector and parallel computers. Appl. Math. Comput. **19**, 3–22 (1986). https://doi.org/10.1016/0096-3003(86)90094-9
15. Llorente, I.M., Melson, N.D.: Robust multigrid smoothers for three dimensional elliptic equations with strong anisotropies. Technical report 98-37, ICASE (1998)