



DAMIAN GOIK
KRZYSZTOF BANAŚ 
JAN BIELAŃSKI 
KAZIMIERZ CHŁOŃ

EFFICIENT SIMULATIONS OF LARGE-SCALE CONVECTIVE HEAT TRANSFER PROBLEMS

Abstract

We describe an approach for efficient solution of large-scale convective heat transfer problems that are formulated as coupled unsteady heat conduction and incompressible fluid-flow equations. The original problem is discretized over time using classical implicit methods, while stabilized finite elements are used for space discretization. The algorithm employed for the discretization of the fluid-flow problem uses Picard's iterations to solve the arising nonlinear equations. Both problems (the heat transfer and Navier–Stokes equations) give rise to large sparse systems of linear equations. The systems are solved by using an iterative GMRES solver with suitable preconditioning. For the incompressible flow equations, we employ a special preconditioner that is based on an algebraic multigrid (AMG) technique.

This paper presents algorithmic and implementation details of the solution procedure, which is suitably tuned – especially for ill-conditioned systems that arise from discretizations of incompressible Navier–Stokes equations. We describe a parallel implementation of the solver using MPI and elements from the PETSC library. The scalability of the solver is favorably compared with other methods, such as direct solvers and the standard GMRES method with ILU preconditioning.

Keywords

convective heat transfer, finite element method, sparse linear equations, algebraic multigrid, Navier–Stokes equations, GMRES, block preconditioning, SUPG stabilization, MPI, PETSC, scalability

Citation

Computer Science 22(4) 2021: 517–538

Copyright

© 2021 Author(s). This is an open access publication, which can be used, distributed and reproduced in any medium according to the Creative Commons CC-BY 4.0 License.

1. Introduction

An efficient numerical solution for convective heat transfer problems has remained a challenge for many decades [22, 29]. Because of the parabolic nature of heat transfer equations, large time steps can be used in time discretization. This leads to attempts to employ implicit methods for the Navier–Stokes equations of an incompressible fluid flow. The non-linear nature of the equations requires an iterative process for a solution. Finally, large sparse systems of linear equations are produced by the time and space discretizations of the coupled problem.

The most typical approach to tackling the problem numerically is to use some form of iterative process. The first step of introducing iterations is to decouple the heat and fluid-flow equations. For incompressible fluids (contrary to the case of compressible flows [5]), the coupling is weak; the temperature only affects the coefficients of the Navier–Stokes equations and buoyancy terms due to the Boussinesq approximation [4].

After applying an iterative solution to the non-linear flow problem, the next step is to use iterative solvers for the systems of linear equations that are produced by both decoupled problems, since the direct solvers (especially for simulations in 3D space) suffer from excessive requirements for storage and computation time [9, 21]. The smaller system for the heat equation can usually be efficiently solved by standard methods, while the system for incompressible fluid flow is ill-conditioned due to the assumed infinite speed of the propagation of pressure changes. When using Krylov subspace methods like GMRES [25], for example, efficient preconditioning is necessary for obtaining feasible solution times for large-scale problems.

Preconditioners for the Navier–Stokes problem is another place where the iterative process can be used. Following the approach of decoupling the velocity and pressure parts of the discretized fluid-flow equations [22], a similar decoupling for linear systems (the so-called block decomposition) can be introduced [13]. This leads to the application of a preconditioner in several steps where smaller decoupled linear systems are employed. For these systems, iterative procedures can be applied as well to ensure fast and accurate approximations.

The speed of the convergence of the whole procedure depends on many factors, with the size of the time step (and its related CFL number) as well as the Reynolds number of the considered flow being most important [11]. Moreover, this also depends on the details of the discrete problem formulation – especially the method that is used to deal with the numerical instabilities that appear in the standard formulations of the convection dominated equations.

For the finite element method, one of the classic ways of obtaining stable solutions is to employ a stabilization that is based on second-order terms, such as the streamline-upwind/Petrov–Galerkin (SUPG) formulation [7, 14]. Compared to the alternative solution of using one of the mixed formulations that uses different orders of approximation for the pressure and velocity unknowns, SUPG stabilization is easier to

implement; however, it creates linear systems with negative and positive eigenvalues that slows down the convergence of the iterative solvers [23].

In this paper, we present a solution procedure for the convective heat transfer problems that employ the preconditioner introduced in [16] as well as the results of its application to the well-known heat-driven (buoyancy) cavity-flow simulations. Implemented on the basis of the PETSC library, the preconditioner uses the block decomposition of the system of linear equations and Schur complement techniques to produce a three-step iterative procedure with separate linear subsystems. The subsystem that is related to the velocity tends to be diagonally dominant; on the other hand, the subsystem that is related to the pressure has a worse condition number and small terms on the diagonal (especially for fine meshes). The most efficient solution technique for the latter system is multigrid [6, 19], with pressure changes that are transferred throughout the whole computational domain sufficiently quickly due to the use of a sequence of coarser grids. Our preconditioner employs the algebraic multigrid method [28], which has an important advantage of constructing coarse problems independently of any information that is related to the discretization of the computational domain.

The procedure is designed for large-scale problems; hence, parallel implementation is applied. We use MPI and distributed memory machines in order to guarantee the scalability of the computations. The parallelization is based on domain decomposition; for the solvers of linear equations, this translates to some form of system matrix decomposition [27]. The subdomains (subsystems) at all levels are balanced in terms of their size and preferably have as small of an interface as possible so that the required communication can be reduced [3, 10]. For standard decomposition methods [18, 26], the performance of the solver may deteriorate when compared to the sequential version. In such a case, some modifications for parallel versions of the partitioning algorithm are employed [17].

In the next section of the paper, we briefly describe the space and time discretizations of the coupled heat transfer and incompressible flow equations that we employ. In Section 3, the resulting systems of linear equations are discussed (taking their structure and possible decomposition into account). We present a preconditioner algorithm for the GMRES solver that exploits the block structure of the system matrix in the fluid-flow problem and uses the multigrid strategy for its pressure part. In Section 4, the details of the parallel implementation of the developed solver algorithm are presented (using the ModFEM framework and PETSC library). Section 5 contains the description of a test problem and the results of the simulations, with a special stress on the performance of the different solvers of the linear equations as well as the scalability of the proposed algorithm and its implementation. In Section 6, we present our conclusions from the numerical experiments.

2. Convective heat transfer problem and its discretization

We consider the formulated convective heat transfer problem as coupled unsteady Fourier (heat conduction) and Navier–Stokes (incompressible fluid-flow) equations.

Contrary to compressible flows (where the coupling between the momentum and energy balance is provided by some material model like the ideal gas law or the van der Waals equation, for example), the assumption of constant density ρ (fundamental to the incompressible model) leads to several simplifications. First, the mass balance reduces to the divergence-free condition for velocity field \mathbf{u} , $u_{i,i} = 0$. Pressure p becomes related to only the velocity field, and the stress tensor can be represented as the sum of pressure p and the viscous stresses $\boldsymbol{\tau}$ (which we assume in a form that is typical for Newtonian fluids with a dynamic viscosity μ) and the Stokes hypothesis ($\tau_{i,i} = 0$) applied: $\tau_{ji} \approx \mu(u_{i,j} + u_{j,i} - \frac{2}{3}\delta_{ji}u_{k,k})$ (this form is further simplified using the divergence free condition for the velocity field).

Furthermore, the energy balance can be expressed exclusively for internal energy e_I (neglecting the potential energy and using the momentum balance to eliminate the mechanical energy from the general energy balance):

$$(\rho e_I)_{,t} + (\rho e_I u_i)_{,i} + p u_{i,i} - \tau_{ij} u_{i,j} + q_{i,i} = 0$$

Apart from explicit heat flux q (which has already been taken into account), the heat that is produced by the viscosity and the work of the pressure, the possible influence of the body forces, and the internal heat sources can be introduced in the equation above using additional terms. The equation for the internal energy is further simplified by using the following thermodynamic assumptions concerning specific heats at constant pressure c_p , constant volume c_V , and temperature T : $c_p = c_V = c$, $e_I = cT$.

Finally, the temperature is related to the heat flux using Fourier's law: $q_i = -\lambda T_{,i}$; this leads to the final form of the energy balance that we accept in our model (after neglecting the heat that is produced by the fluid viscosity and the introduction of internal heat sources s):

$$\rho c \left(\frac{\partial T}{\partial t} + \mathbf{u} \cdot \nabla T \right) - \nabla \cdot (\lambda \nabla T) = s \quad (1)$$

In our simulations, we assume that the material parameters (assumed to be constant for the purpose of the equation derivation) such as density ρ , specific heat c , and heat conductivity λ can still be treated as possibly purely experimental functions of the temperature.

The above equation of conductive heat transfer for unknown temperature field $T(\mathbf{x}, t)$ posed in the computational domain Ω is accompanied by the set of boundary conditions that include classical Dirichlet conditions for the temperature ($T = T_0$) on the part of $\partial\Omega$ denoted by Γ_T as well as the conditions for the heat flux ($-\lambda \frac{\partial T}{\partial \mathbf{n}} = \mathbf{q}$) on the Γ_q part of $\partial\Omega$ (with \mathbf{n} denoting, in the standard way, the unit outward vector that is normal to the boundary). Heat flux \mathbf{q} may be specified as constant or as a function (linear or nonlinear) of the temperature on the boundary and some other parameters (like the ambient temperature, heat transfer coefficient, or parameters of the radiation condition [4]).

Vector of conductive velocity \mathbf{u} is supplied to the heat transfer equation by the coupled system of the Navier–Stokes equations. The coupling in this direction is strong, which influences the solution procedure for the whole system.

The particular form of the Navier–Stokes equations in our model is derived from the mass and momentum-balance equations assuming the introduced form of the viscous stresses. Formulated for the unknown fluid velocity $\mathbf{u}(\mathbf{x}, t)$ and pressure $p(\mathbf{x}, t)$ they are considered in the following form:

$$\begin{aligned} \rho \left(\frac{\partial \mathbf{u}}{\partial t} + (\mathbf{u} \cdot \nabla) \mathbf{u} - \nu \nabla^2 \mathbf{u} \right) + \nabla p &= \mathbf{f} \\ \nabla \cdot \mathbf{u} &= 0 \end{aligned} \quad (2)$$

with boundary conditions

$$\begin{aligned} \mathbf{u} &= \hat{\mathbf{u}}_0 \quad \text{on } \Gamma_D \\ (\nu \nabla \mathbf{u}) \mathbf{n} - p \mathbf{n} &= \mathbf{g} \quad \text{on } \Gamma_N \end{aligned}$$

In Equation (2), ν denotes the kinematic viscosity of fluid ($\nu = \mu/\rho$), and \mathbf{f} is a source term that includes gravity forces (the system is considered in the dimensional form). Vector fields $\hat{\mathbf{u}}_0$ and \mathbf{g} are given on disjoint parts of the boundary of computational domain Ω (Γ_D for velocities and Γ_N for stresses, respectively).

The influence of the temperature field on the velocity and pressure fields is relatively weak. Similar as for the heat transfer equation, we first assume that all material parameters such as density ρ and viscosity ν can be functions of the temperature. Moreover, using the terms for forces \mathbf{f} , we take the temperature-induced buoyancy forces that result from the varying density of the fluid that is subject to the gravity field into account.

The coupled system of Equations (1) and (2) is transformed by using the standard finite element space discretization procedures of multiplying by test functions and integrating over the computational domain. System (2) is transformed into a single weak statement (with test functions \mathbf{w} for the momentum balance and r for the divergence condition), while (1) is treated as a separate system. Both problems are coupled by the solution procedure described below, which provides field \mathbf{u} to (1) and field T to (2).

For both systems, the same triangulation of domain Ω into elements Ω_e is introduced (in the practical examples, we use 3D prismatic elements), and the same approximation based on the linear shape functions is assumed. For the heat equation, the spaces of the continuous piece-wise linear polynomials (V_T^h for the temperature, and V_v^h for the test functions) are introduced, with the functions in V_T^h satisfying the Dirichlet boundary conditions for the temperature on Γ_T and the functions in V_v^h being equal to zero on Γ_T .

For the Navier–Stokes equations, the similar spaces $V_{\mathbf{u}}^h$ and V_p^h of the continuous piece-wise linear polynomials that satisfy the Dirichlet boundary conditions for

velocities and pressure (vector valued for velocities) are defined, together with corresponding function spaces $V_{\mathbf{w}}^h$ and V_r^h for the test functions (with zero values on the Dirichlet parts of the boundary).

Both the (1) and (2) systems are unstable in their standard Galerkin forms, so both are stabilized by using the SUPG method [14]. Using the index notation and the summation convention for repeated indices together with “ $_{,i}$ ” denoting the space differentiation with respect to the i -th space coordinate, the final weak formulation for the heat equation takes the following form:

Find approximate function $T^h \in V_T^h$ such that statement

$$\int_{\Omega} \rho c \frac{\partial T^h}{\partial t} v^h d\Omega + \int_{\Omega} \rho c u_i T_{,i}^h v^h d\Omega + \int_{\Omega} \lambda T_{,i}^h v_{,i}^h d\Omega + \sum_e \int_{\Omega_e} R^F(T^h) \sigma R^F(v^h) d\Omega + \int_{\Gamma_q} q v^h d\Gamma = \int_{\Omega} s v^h d\Omega \tag{3}$$

holds for each test function $v^h \in V_v^h$. Above, $R^F(T^h)$ and $R^F(v^h)$ denote the residuals of heat equation (1) that were computed for the respective arguments, while σ is the coefficient of the SUPG stabilization.

The system of the Navier–Stokes equations is transformed into the following weak formulation:

Find approximate functions $\mathbf{u}^h \in V_{\mathbf{u}}^h$ and $p^h \in V_p^h$ such that statement

$$\int_{\Omega} \rho \frac{\partial u_j^h}{\partial t} w_j^h d\Omega + \int_{\Omega} \rho u_{j,i}^h u_i^h w_j^h d\Omega + \int_{\Omega} \rho \nu u_{j,i}^h w_{j,i}^h d\Omega - \int_{\Omega} p^h w_{j,j}^h d\Omega - \int_{\Omega} u_{j,j}^h r^h d\Omega + \sum_e \int_{\Omega_e} u_{j,l}^h \gamma w_{j,l}^h d\Omega + \sum_e \int_{\Omega_e} R_j^{NS}(\mathbf{u}^h, p^h) \omega \delta_{jl} R_l^{NS}(\mathbf{w}^h, r^h) d\Omega = \int_{\Omega} f_j w_j^h d\Omega - \int_{\Gamma_N} g_j w_j^h d\Gamma \tag{4}$$

holds for each test function $\mathbf{w}^h \in V_{\mathbf{w}}^h$ and $r^h \in V_r^h$.

Above, $R_j^{NS}(\mathbf{u}^h, p^h)$ and $R_l^{NS}(\mathbf{w}^h, r^h)$ denote the residuals of the momentum balance equations that were computed for the respective arguments. The symbol δ_{jl} denotes the usual Kronecker’s delta, while ω and γ are the coefficients of the SUPG stabilization (we refer to papers [14] and [15], fundamental for the SUPG stabilization, for further details concerning the existence, uniqueness, stability, and convergence of the solutions).

The resulting weak statements are non-linear and include time derivatives. For the space-time discretization, we use the method of lines (with finite differences for discretization over time) and the presented finite element discretization in space. For each unknown field, we represent the values at point \mathbf{x} and time instant t as the product of finite element basis functions $\psi_L(\mathbf{x})$ (depending only on \mathbf{x}) and the time-dependent

coefficients of the linear combination of basis functions (the values of the unknowns at finite element nodes). For the temperature field, this would read as follows:

$$T(\mathbf{x}, t) = \sum_{L=1}^N T_L(t) \psi_L(\mathbf{x})$$

with similar expressions for other fields. As a consequence, the time derivative concerns only those values at the nodes, and the functions at a particular time instant t^n can be expressed with the formulae (using temperature again as an example):

$$T^h(\mathbf{x}, t^n) = \sum_{L=1}^N T_L(t^n) \psi_L(\mathbf{x})$$

$$\left. \frac{\partial T^h(\mathbf{x}, t)}{\partial t} \right|_{t^n} = \sum_{L=1}^N \left. \frac{dT_L(t)}{dt} \right|_{t^n} \psi_L(\mathbf{x})$$

The standard implicit Euler time-integration method is applied for both of the coupled problems. The whole formulation is considered at time t^{n+1} , with the time derivative at t^{n+1} calculated as the backwards-in-time finite difference (which reads as follows for the example temperature field):

$$\left. \frac{\partial T^h(\mathbf{x}, t)}{\partial t} \right|_{t^{n+1}} \approx \frac{T^h(\mathbf{x}, t^{n+1}) - T^h(\mathbf{x}, t^n)}{\Delta t}$$

where $\Delta t = t^{n+1} - t^n$. Since we deal with stationary problems in the current paper, the problem of accuracy in time is irrelevant only if the procedure converges to the steady state. We exploit the unconditional stability of the implicit backward Euler scheme, which is also valid for non-linear fluid-flow problems ([5]). For transient problems (which are not considered in the current paper) we use the second-order Crank-Nicolson method, having conditional stability.

After the time discretization, the weak statements consist of the terms without time derivatives but possibly with non-linear coefficients (including velocity field \mathbf{u} , which can be treated as a coefficient for both the heat transfer and Navier–Stokes equations).

The procedure of the simple (Picard) iteration (which solves for unknown values in the next iteration by assuming the values of the coefficients that are computed based on the unknowns in the previous iteration) is used for solving the system of non-linear equations. Denoting the value of each function using the superscripts for the time instant and the subscripts for the subsequent Picard iterations (dropping superscript h for brevity), we arrive at the final weak statement of the coupled problem.

The weak formulation for the heat equation aims at finding solution T_{k+1}^{n+1} in the next $(k+1)$ non-linear (Picard) iteration and time step $n+1$ given solution T_k^{n+1} from

the previous iteration (used for calculating nonlinear coefficients) and solution T^n in the previous time step:

$$\int_{\Omega} \rho c \frac{T_{k+1}^{n+1}}{\Delta t} v d\Omega + \int_{\Omega} \rho c (u_{k+1}^{n+1})_i (T_{k+1}^{n+1})_{,i} v d\Omega + \int_{\Omega} \lambda (T_{k+1}^{n+1})_{,i} v_{,i} d\Omega \tag{5}$$

$$+ \sum_e \int_{\Omega_e} R^F(T_{k+1}^{n+1}) \sigma R^F(v) d\Omega + \int_{\Gamma_q} q v^h d\Gamma = \int_{\Omega} \rho c \frac{T^n}{\Delta t} v d\Omega + \int_{\Omega} s v d\Omega$$

A similar formulation is obtained for the Navier–Stokes equations, where the velocity in the convection term is treated as a coefficient and supplied from the previous iteration:

$$\int_{\Omega} \rho \frac{(u_{k+1}^{n+1})_j}{\Delta t} w_j d\Omega + \int_{\Omega} \rho (u_{k+1}^{n+1})_{j,i} (u_k^{n+1})_l w_j d\Omega + \int_{\Omega} \rho \nu (u_{k+1}^{n+1})_{j,i} l w_{j,i} d\Omega$$

$$- \int_{\Omega} p_{k+1}^{n+1} w_{j,j} d\Omega - \int_{\Omega} (u_{k+1}^{n+1})_{j,j} r d\Omega + \sum_e \int_{\Omega_e} R_j^{NS}(\mathbf{u}, p) \omega_{j,l} R_l^{NS}(\mathbf{w}, r) d\Omega \tag{6}$$

$$+ \sum_e \int_{\Omega_e} (u_{k+1}^{n+1})_{j,i} \gamma w_{j,i} d\Omega = \int_{\Omega} \rho \frac{(u^n)_j}{\Delta t} w_j d\Omega + \int_{\Omega} f_j w_j d\Omega - \int_{\Gamma_N} g_j w_j d\Gamma$$

with residual $R_j^{NS}(\mathbf{u}, p)$ calculated as a function of \mathbf{u}_{k+1}^{n+1} , \mathbf{u}_k^{n+1} , and p_{k+1}^{n+1} .

Statements 5 and 6 reveal the details of the solution procedure for the coupled system. In each time step and each iteration, we first solve the linear system that is related to the Navier–Stokes equations due to its weak coupling with the heat transfer equation. The indicated coefficients that introduce the non-linearity (including velocity and the other parameters that depend on the temperature) are calculated for the solutions from the previous iteration (for the first iteration, the solutions from the previous time step are taken). Then, we solve the linear system for the heat equation using the conductive velocity that was just calculated by the linear system that is related to the Navier–Stokes equations in the same nonlinear iteration. The procedure is repeated for the next (and subsequent) non-linear Picard iterations. First, the linear system that is related to the Navier–Stokes equations is solved, then the linear system for the heat transfer equation (strongly coupled by the velocity field). For each linear system, the coefficients (and, as a consequence, the system matrices) are recalculated for the most recent values of the velocity and temperature fields. We repeat the iterations until the criterion of convergence for the coupled non-linear problem is met or the maximal number of iterations is reached.

As a result, we get the converged solution fields for the whole coupled system at the end of non-linear iterations for each time step (despite the fact that we decoupled the linear systems).

The calculations are performed for a sequence of time steps. For the transient problems, the time-step length is selected based on the accuracy requirements, and the simulation continues for the specified period of time. For steady-state problems, we use time integration until a stationary limit is found (with the time-step length

adapted to the stability constraints). The particular form of the adaptation algorithm depends on the problem and the degree of non-linearity. Usually, the simulations start with a time-step length that is close to the value that corresponds to the classical CFL condition and is enlarged when the convergence of the non-linear iterations is sufficiently fast.

3. Systems of linear equations

For the described procedure, two systems of linear equations are solved in each non-linear iteration at each time step. The system for the (scalar) heat equation has a typical form for the finite element approximations of the time-dependent problems. It is sparse, with the diagonal dominance depending on the value of time step Δt .

The system of linear equations for the Navier–Stokes problem has a more complicated structure. For the purpose of the solution procedure, the equations of the system can be arranged in two different ways. Due to the use of stabilized formulation and equal-order interpolation, the unknowns – three velocity components and pressure – are defined at each finite element node (with the nodes that correspond to the vertices of the elements in our linear approximation). If all four unknowns at each node are arranged in the solution vector as subsequent components, the global system matrix can be split into 4×4 blocks. The solution procedure can take advantage of this structure by using the BCRS format, for example, for storing the matrix and suitably adapting the algorithms. We employ this approach in our reference-incomplete LU preconditioner for the GMRES solver.

Another possible solution is to split the unknown vector into two large parts: the first with all of the velocity unknowns (that we will denote as \mathbf{u}_v), and the second with the pressure degrees of freedom (denoted by \mathbf{u}_p). In this approach, the system of equations splits into the following parts:

$$\begin{pmatrix} \mathbf{D}_{vv} & \mathbf{D}_{vp} \\ \mathbf{D}_{pv} & \mathbf{D}_{pp} \end{pmatrix} \cdot \begin{pmatrix} \mathbf{u}_v \\ \mathbf{u}_p \end{pmatrix} = \begin{pmatrix} \bar{\mathbf{b}}_w \\ \mathbf{b}_q \end{pmatrix} \quad (7)$$

In the absence of stabilization terms, part \mathbf{D}_{vp} is just the transpose of \mathbf{D}_{pv} , while part \mathbf{D}_{pp} vanishes. For the stabilized formulation, additional terms appear in \mathbf{D}_{vp} , \mathbf{D}_{pv} , and \mathbf{D}_{pp} , while \mathbf{D}_{vv} keeps its diagonally dominant form due to the discretized time-derivative term. Additionally, matrix \mathbf{D}_{vv} depends on the solution in the previous Picard iteration, while the right-hand side $\bar{\mathbf{b}}_w$ depends on the solution in the previous time step.

We solve system (7) using the restarted GMRES method with left preconditioning (where the preconditioning in the standard way corresponds to the multiplication of both sides of the system by a matrix that tries to approximate the inverse of the system matrix [24]). In the GMRES procedure, this would manifest by the multiplication of the preconditioner matrix with the vector that resulted from the product of the system matrix with the suitable vector (related to the residual of the linear system). In our

implementation, we use a typical approach where the preconditioner matrix is not constructed explicitly and the result of matrix-vector product is achieved by several steps of a specially designed algorithm (where only linear operations are applied to the input vector in each step).

In order to develop an algorithm that reflects the action of the preconditioner, the following identity is employed:

$$\begin{pmatrix} \mathbf{D}_{\mathbf{v}\mathbf{v}} & \mathbf{D}_{\mathbf{v}\mathbf{p}} \\ \mathbf{D}_{\mathbf{p}\mathbf{v}} & \mathbf{D}_{\mathbf{p}\mathbf{p}} \end{pmatrix} = \begin{pmatrix} \mathbf{I} & 0 \\ \mathbf{D}_{\mathbf{p}\mathbf{v}}\mathbf{D}_{\mathbf{v}\mathbf{v}}^{-1} & \mathbf{I} \end{pmatrix} \cdot \begin{pmatrix} \mathbf{D}_{\mathbf{v}\mathbf{v}} & 0 \\ 0 & \mathbf{S} \end{pmatrix} \cdot \begin{pmatrix} \mathbf{I} & \mathbf{D}_{\mathbf{v}\mathbf{v}}^{-1}\mathbf{D}_{\mathbf{v}\mathbf{p}} \\ 0 & \mathbf{I} \end{pmatrix}$$

where \mathbf{S} is the Schur complement for $\mathbf{D}_{\mathbf{p}\mathbf{p}}$

$$\mathbf{S} = \mathbf{D}_{\mathbf{p}\mathbf{p}} - \mathbf{D}_{\mathbf{p}\mathbf{v}}\mathbf{D}_{\mathbf{v}\mathbf{v}}^{-1}\mathbf{D}_{\mathbf{v}\mathbf{p}}$$

Hence, the inverse of the system matrix is given by the following:

$$\begin{pmatrix} \mathbf{D}_{\mathbf{v}\mathbf{v}} & \mathbf{D}_{\mathbf{v}\mathbf{p}} \\ \mathbf{D}_{\mathbf{p}\mathbf{v}} & \mathbf{D}_{\mathbf{p}\mathbf{p}} \end{pmatrix}^{-1} = \begin{pmatrix} \mathbf{I} & -\mathbf{D}_{\mathbf{v}\mathbf{v}}^{-1}\mathbf{D}_{\mathbf{v}\mathbf{p}} \\ 0 & \mathbf{I} \end{pmatrix} \cdot \begin{pmatrix} \mathbf{D}_{\mathbf{v}\mathbf{v}}^{-1} & 0 \\ 0 & \mathbf{S}^{-1} \end{pmatrix} \cdot \begin{pmatrix} \mathbf{I} & 0 \\ -\mathbf{D}_{\mathbf{p}\mathbf{v}}\mathbf{D}_{\mathbf{v}\mathbf{v}}^{-1} & \mathbf{I} \end{pmatrix}$$

The action of the inverse of the matrix on a vector (with the parts that are related to the velocities and pressure denoted by $\mathbf{z}_{\mathbf{v}}$ and $\mathbf{z}_{\mathbf{p}}$, respectively) can be written as follows:

$$\begin{pmatrix} \mathbf{D}_{\mathbf{v}\mathbf{v}} & \mathbf{D}_{\mathbf{v}\mathbf{p}} \\ \mathbf{D}_{\mathbf{p}\mathbf{v}} & \mathbf{D}_{\mathbf{p}\mathbf{p}} \end{pmatrix}^{-1} \begin{pmatrix} \mathbf{z}_{\mathbf{v}} \\ \mathbf{z}_{\mathbf{p}} \end{pmatrix} = \begin{pmatrix} \mathbf{D}_{\mathbf{v}\mathbf{v}}^{-1}(\mathbf{z}_{\mathbf{v}} - \mathbf{D}_{\mathbf{v}\mathbf{p}}\tilde{\mathbf{z}}_{\mathbf{p}}) \\ \mathbf{S}^{-1}(\mathbf{z}_{\mathbf{p}} - \mathbf{D}_{\mathbf{p}\mathbf{v}}\mathbf{D}_{\mathbf{v}\mathbf{v}}^{-1}\mathbf{z}_{\mathbf{v}}) \end{pmatrix}$$

The formulae presented above give rise to the family of SIMPLE (semi-implicit pressure-linked equation) solvers and preconditioners [22]. In the version of the SIMPLE preconditioning strategy that we use in our solver [13], the following algorithm is employed to approximate the action of the inverse of the system matrix on any vector with parts $\mathbf{z}_{\mathbf{v}}$ and $\mathbf{z}_{\mathbf{p}}$ and storing the result in $\hat{\mathbf{z}}_{\mathbf{v}}$ and $\hat{\mathbf{z}}_{\mathbf{p}}$:

1. solve approximately: $\mathbf{D}_{\mathbf{v}\mathbf{v}}\tilde{\mathbf{z}}_{\mathbf{v}} = \mathbf{z}_{\mathbf{v}}$;
2. solve approximately: $\tilde{\mathbf{S}}\hat{\mathbf{z}}_{\mathbf{p}} = \mathbf{z}_{\mathbf{p}} - \mathbf{D}_{\mathbf{p}\mathbf{v}}\tilde{\mathbf{z}}_{\mathbf{v}}$;
3. substitute: $\hat{\mathbf{z}}_{\mathbf{v}} = \tilde{\mathbf{z}}_{\mathbf{v}} - \tilde{\mathbf{D}}_{\mathbf{v}\mathbf{v}}^{-1}\mathbf{D}_{\mathbf{v}\mathbf{p}}\hat{\mathbf{z}}_{\mathbf{p}}$.

In Step 2 of the algorithm, an approximation $\tilde{\mathbf{S}}$ to the original Schur complement matrix \mathbf{S} is used. The approximation changes the original block $\mathbf{D}_{\mathbf{v}\mathbf{v}}^{-1}$ to some approximation. The approximation to $\mathbf{D}_{\mathbf{v}\mathbf{v}}^{-1}$ is also used in Step 3 of the algorithm (denoted their by $\tilde{\mathbf{D}}_{\mathbf{v}\mathbf{v}}^{-1}$), although these two approximations to $\mathbf{D}_{\mathbf{v}\mathbf{v}}^{-1}$ can be different.

The quality of the approximations to $\mathbf{D}_{\mathbf{v}\mathbf{v}}^{-1}$ (both in the Schur complement and the standalone) constitutes the most important factor that influences the quality of

the preconditioner and, as a consequence, the convergence rate of the solver. Even with the same approximation being employed for all of the subsystems in the algorithm, the selected solver and the accuracy of the solution (the number of iterations) can be different for each subsystem. The exact computation of the Schur complement is infeasible, as this is more difficult than solving the whole system. In a general case, finding a good approximation for the Schur complement is difficult, and the solution of the subsystem that is related to the pressure (which is included in the Schur complement) usually limits the convergence rate of the whole solver.

We use the following techniques in each step of the solution procedure [16]. In Step 1, we approximately solve the system by simply employing the Gauss-Seidel iterations to $\mathbf{D}_{\mathbf{v}\mathbf{v}}$. This simple approach is sufficient for assuring that the efficiency of the preconditioner and the convergence of the whole solver solely depends on the quality of solving Step 2 of the procedure. We develop special heuristics to determine the exact ratio of the number of iterations used in Step 1 to the number of iterations in Step 2.

There are several possible techniques to approximate $\mathbf{D}_{\mathbf{v}\mathbf{v}}^{-1}$ in the Schur complement matrix. One of the simplest and most efficient (the one that we use in our solution procedure) is to change the original block $\mathbf{D}_{\mathbf{v}\mathbf{v}}^{-1}$ to the diagonal matrix, with each row containing the inverse of the sum of the absolute values of the entries in the corresponding row of $\mathbf{D}_{\mathbf{v}\mathbf{v}}$.

Step 2 constitutes the most important phase of the application of the preconditioner – inducing its overall efficiency that is reflected by the convergence rate of the solver. Apart from using some form of approximation to $\mathbf{D}_{\mathbf{v}\mathbf{v}}^{-1}$, this requires the approximate solution of the system with matrix $\mathbf{D}_{\mathbf{p}\mathbf{p}}$ as its dominant part. Being related to the pressure unknowns, this matrix has no time-derivative terms and corresponds to the changes that immediately propagate across the computational domain; therefore, it has an elliptic character and is ill-conditioned. To solve this system, we use a special variant of the algebraic multigrid method.

We use a single V-cycle of the multigrid. First, we perform the smoothing on the finest level using the Gauss-Seidel algorithm for matrix $\tilde{\mathbf{S}}$. Then, we create the coarsened system. Using the standard algebraic multigrid criteria (based on measuring the influence of the values of the solution at certain nodes on the values of the solution in the neighboring nodes [17]), the nodes that are meant to remain at the coarser level are selected (with the rest of the nodes omitted). The lines and columns of the system matrix at the fine level that correspond to the omitted nodes are also neglected, and the new system matrix for the coarse level is created using the Galerkin projection [28]. The solution from the fine level is projected (restricted) to the coarse level, and the coarse level system is used to smooth the coarse-level solution.

The procedure is repeated for several coarser levels up to the coarsest system, which is solved exactly and its solution added as the coarsest level correction to the solution. Then, the solution is projected back (prolongated) to the next-finer level and smoothed again at that level. The final smoothing at the finest level ends the

V-cycle procedure. We use standard algebraic multigrid techniques for the restriction and interpolation [28], with Gauss-Seidel again used for the smoothing.

In Step 3 of the preconditioner algorithm, we simply perform two matrix vector products using the previously created $\mathbf{D}_{\mathbf{v}\mathbf{v}}^{-1}$ approximation and one vector subtraction. Step 3 is computationally less demanding, so the performance-tuning is done for Steps 1 and 2 only.

4. Parallel implementation

The parallelization of the whole solution procedure follows the steps that are typically employed in such cases. The subsequent stages of the computations (which include not only the consecutive non-linear iterations and time steps but also the creation and solution of the two linear systems of equations) are performed in sequential order.

The most interesting, from the theoretical point of view, is the stage of solving a single linear system that corresponds to the Navier–Stokes equations. The history of the tasks that are related to the most time-consuming operations after the system’s creation can be analyzed by using the Mazurkiewicz trace model [12]. For this purpose, we introduce the notation for the velocity part of the system:

- a_{ij} – operations that are related to i -th local iteration of Gauss-Seidel method for j -th subdomain;
- A_i – beginning of i -th global Gauss-Seidel iteration;

and for the pressure part:

- b_{ij}^k – operations that are related to i -th local iteration of Gauss-Seidel method for j -th subdomain and level k ;
- B_i^k – beginning of i -th global Gauss-Seidel iteration at level k .

The dependency relationships appear as follows:

- $\forall_{i,j} \{a_{ij}, A_i\}^2 \cup \{A_i, a_{i+1j}\}^2$;
- $\forall_{i,j,k} \{b_{ij}^k, B_i^k\}^2 \cup \{B_i^k, b_{i+1j}^k\}^2 \cup \{B_i^k, b_{ij}^{k+1}\}^2$ – for part of V-cycle going up;
- $\forall_{i,j,k} \{b_{ij}^k, B_i^k\}^2 \cup \{B_i^k, b_{i+1j}^k\}^2 \cup \{B_i^k, b_{ij}^{k-1}\}^2$ – for part of V-cycle going down.

Using the developed framework, example histories can be derived:

- for the velocity, solve

$$a_{11}a_{12}\dots a_{1J}A_1a_{21}a_{22}\dots a_{2J}A_2\dots A_I;$$

- for the pressure, solve

$$b_{11}^1b_{12}^1\dots b_{1J}^1B_1^1b_{21}^1b_{22}^1\dots b_{2J}^1B_2^1\dots B_I^1b_{11}^2b_{12}^2\dots b_{1J}^2B_1^2b_{21}^2b_{22}^2\dots b_{2J}^2B_2^2\dots B_I^2\dots B_I^K.$$

Analyzing the trace, one can conclude that the full concurrency concerns only the local iterations. This fact is fully exploited in the practical implementation of the solver. The proportion of easily parallelized local iterations increases until the sufficient reduction of the residuum is achieved.

The implementation of the solution procedure for the coupled problem is done by using the finite element framework ModFEM [20], with the PETSC library [1] being employed for the linear algebra operations during the solution of the system that is related to the Navier–Stokes equations.

ModFEM is a general-purpose finite-element software framework with a modular structure [2] that uses special problem-dependent modules to create codes for different application domains [4]. In our setting, the generic ModFEM modules are used to manage the computational grids (particularly to perform the domain decomposition for the parallel execution) for both coupled problems. ModFEM also handles the creation of the local element matrices and vectors according to the weak statements of both problems and the solution of the linear system that is related to the heat equation. For the Navier–Stokes equations, ModFEM passes the element system matrices and load vectors to a special module that designed to solve the system that is related to the Navier–Stokes equations.

The special module for the Navier–Stokes equations implements the preconditioner that was described in the previous section. It is built around the matrix and vector data structures provided by the PETSC library along with the basic matrix and vector operations (including the sparse matrix-matrix product). The module assembles the local element matrices and vectors to global matrices $\mathbf{D}_{\mathbf{v}\mathbf{v}}$, $\mathbf{D}_{\mathbf{v}\mathbf{p}}$, $\mathbf{D}_{\mathbf{p}\mathbf{v}}$, and $\mathbf{D}_{\mathbf{p}\mathbf{p}}$ as well as vectors $\bar{\mathbf{b}}_{\mathbf{w}}$ and $\mathbf{b}_{\mathbf{q}}$. During the solution procedure, the module realizes the steps of the solver algorithm (with the parallel successive over-relaxation, adapted to serve as the Gauss–Seidel smoother, being the only PETSC algorithm utilized).

The main interface of the module allows for two main operations: the AMG-levels creation (set-up phase), and the execution of a single V-cycle on the created levels' structure. The exact form of each operation can be controlled by multiple parameters, including the number of levels, the number of pre- and post-smoothing steps, and the number of local and global iterations [16]. When tuning the solver, it is necessary to maintain good proportions among the level's set-up time, the number of iterations for the convergence, and the single iteration time. For the algebraic multigrid, better set-up will create denser matrices, thus the iteration time will increase but the convergence, hopefully, will improve.

The subsequent steps of the whole parallel solution procedure are as follows. First, the ModFEM code reads those files with mesh and (possibly) initial field data and performs a decomposition of the computational domain into the overlapping subdomains [27]. Then, the simulations follow the scheme of advancing in time, performing non-linear iterations, and solving the two coupled systems of the linear equations at each iteration. For each system and each subdomain, a part of the system is created. Each subdomain that corresponds to a set of rows of the linear systems is assigned to a single process that performs calculations for the subsequent steps of the solution procedure (in a sequential or multi-threaded manner).

The internal ModFEM solver (which is employed to solve the system that is related to the heat equation) uses GMRES with the hybrid block Gauss–Seidel/Jacobi

algorithm as a preconditioner [8]. The algorithm performs the in-parallel Gauss-Seidel iterations within the subdomains, followed by the exchange of the values of the degrees of freedom within the overlap between the subdomains (for the “ghost” nodes). Such a hybrid algorithm (which is frequently used in parallel iterative solvers) results in lower convergence rates than the Gauss-Seidel method at the global level; however, it has a much lower cost and provides good scalability.

When solving the system that is related to the Navier–Stokes equations in parallel, the special solver module assembles the global system matrices so that each process has a continuous range of rows for the subdomain’s internal degrees of freedom. The module performs the AMG level’s structure construction in parallel and creates auxiliary matrices and vectors that are used in the developed preconditioner algorithm.

After creating the data structures for the solution of the system of equations, the control is handed back to the ModFEM module that runs the GMRES algorithm. For each GMRES iteration, the preconditioned residual is computed for the solved system, with the special Navier–Stokes module being used for performing the action of the preconditioner. The same hybrid block Gauss-Seidel/Jacobi algorithm as seen in the heat equation is used for the smoothing in the V-cycle, and the same pattern of exchanging the values of the degrees of freedom for the “ghost” nodes is employed.

During the solution procedures for both systems of linear equations, the only communication steps are required for the global vector operations (norm, scalar product) and the exchange of data during the Gauss-Seidel/Jacobi iterations.

5. Numerical examples

5.1. Test problem

As the problem for testing the performance of the developed procedure, we chose a well-known problem – the classical heat-driven cavity flow in its 3D version. The computational domain consists of the unit cube that is discretized into prismatic finite elements. For the heat problem, the left and right sides have different constant temperatures ($T_L = 1$ and $T_R = 2$, respectively) that drive the fluid motion, while the remaining sides are insulated (with zero heat flux). For the flow problem, all of the sides have no flow (zero velocity) boundary conditions. The parameters are chosen in such a way that the Rayleigh number of the problem is equal to 10^6 (belonging to the laminar flow regime). Figures 1 and 2 present the velocity field of the final stationary flow, while Figure 3 shows the temperature field. Figure 4 combines both fields by using a contour plot of the temperature and arrows that correspond to the velocity vectors.

In order to investigate the performance characteristics of the solution procedure, we select a typical time step (one of internal steps during the convergence to the steady state) and a single non-linear iteration.

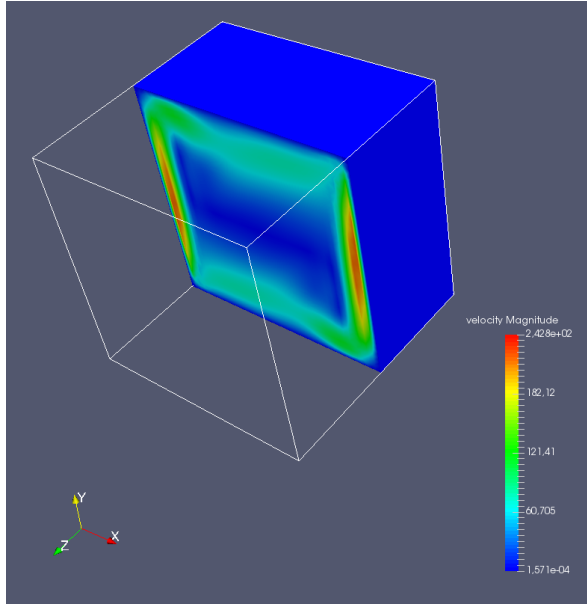


Figure 1. Heat-driven cavity example – vertical cross-section with velocity magnitude field

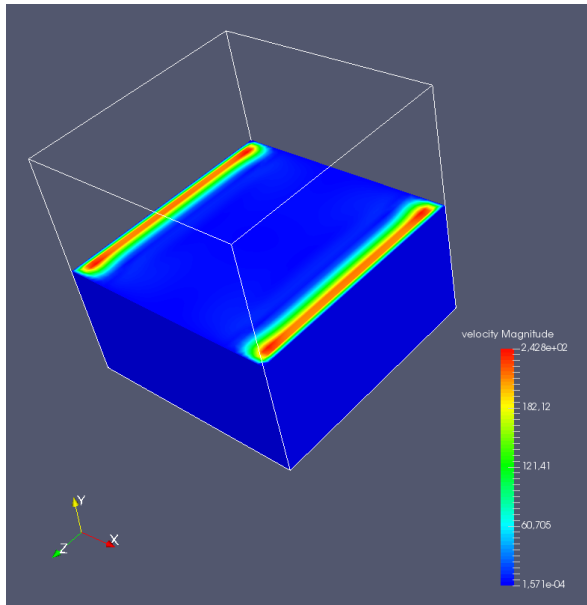


Figure 2. Heat-driven cavity example – horizontal cross-section with velocity magnitude field

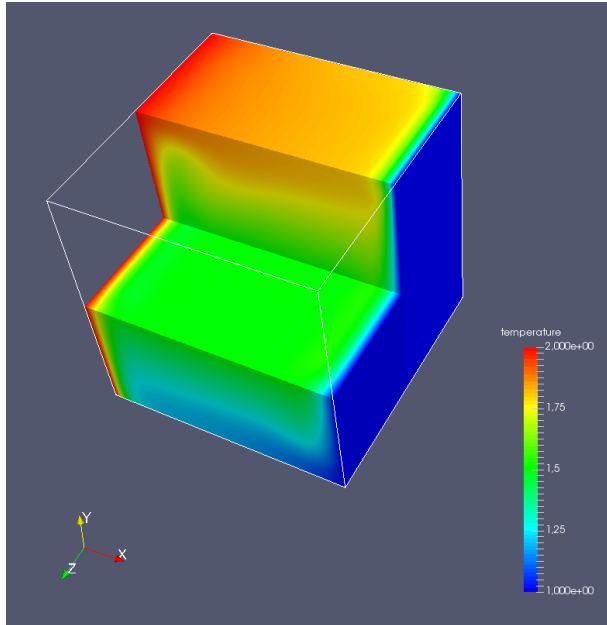


Figure 3. Heat-driven cavity example – cross-section with temperature field

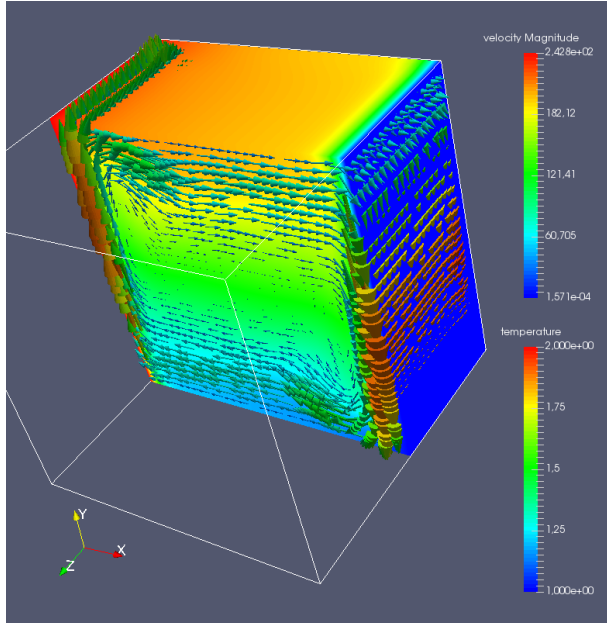


Figure 4. Heat-driven cavity example – cross-section showing contour plot of temperature and arrows corresponding to velocity vectors

5.2. Performance comparison for single computational node

The first test was performed for a single computational node – a system with two 12-core Intel Xeon E5-2650v4 CPUs (2.2 GHz) and 256 GB DRAM running under Centos7 Linux with the 4.4 kernel version.

We compare the performance of the developed solver with a direct solver in the form of a highly tuned PARDISO algorithm from the Intel MKL library and the internal ModFEM solver that uses the same GMRES procedure as with the developed solver but with standard ILU preconditioners.

For a selected time step and non-linear iteration, we report the characteristics for the solution of a single linear system for the Navier–Stokes equations, which always takes more than 80% of the execution time (the rest being the creation of both systems and the heat system solution – the same for all of the compared solvers). The developed solver runs in MPI alone mode, with the other solvers running in OpenMP only modes.

Table 1 presents the execution times that were obtained on a mesh with 409,101 nodes (1,636,404 degrees of freedom). The calculations were performed while using up to 16 cores (threads/processes). The results for the ILU preconditioning of the GMRES are reported only for the ILU(2) variant, since ILU(0) and ILU(1) had too-slow convergence rates due to the strong ill-conditioning of the system.

Table 1

Heat-driven cavity problem – execution time (in seconds) for solving system of linear equations (1,636,404 DOFs) on server using three solvers: direct PARDISO solver from Intel MKL library, standard ModFEM GMRES solver with ILU(2) preconditioner, and GMRES solver with developed block preconditioner based on AMG

Solver	Number of cores				
	1	2	4	8	16
PARDISO	38,241	19,142	10,428	6744	3881
GMRES+ILU(2)	2063	1713	1664	1726	1816
GMRES+AMG	530	272	226	254	414

It can be seen that the developed solver was much faster than both of the other solvers. The direct solver had very long execution times (the 3D grid employed led to high fill-in in the direct decomposition of the system matrix), but it scaled well with increasing numbers of cores/threads. Both of the iterative solvers scaled poorly above two cores. For the case of using the ILU(2) preconditioning, the reason is algorithmic (and the situation cannot easily be improved). For the developed solver, the algorithmic considerations indicated that a much better scalability can be reached; hence, the reason for the low speed-up may lie in the hardware set-up (especially the low memory throughput).

5.3. Scalability study

In order to show the true scalability of the developed solver (not limited by the shared memory bandwidth), the tests were done in a cluster setting with a scalable memory throughput. For this purpose, we employed a specified number of nodes from the Prometheus system at the Cyfronet AGH Computing Center. Each node has two 12-core Intel Xeon E5-2680v3 CPUs (2.5 GHz) and 128 GB DRAM and also runs under the Centos7 Linux version.

Figure 5 shows several performance metrics for the two problem sizes (the number of DOFs equal to 1,636,404 and 12,328,132) and parallel runs on different numbers of processors in the Prometheus cluster. The metrics include execution time, standard parallel speed-up, and efficiency. Figure 6 presents the results for the weak scalability study, where the execution times are compared for different runs with the same number of DOFs per single subdomain. It can be seen that the solver scales well up to 128 processes, with the efficiency of the parallelization above 60% for the larger case with the 12,328,132 unknowns.

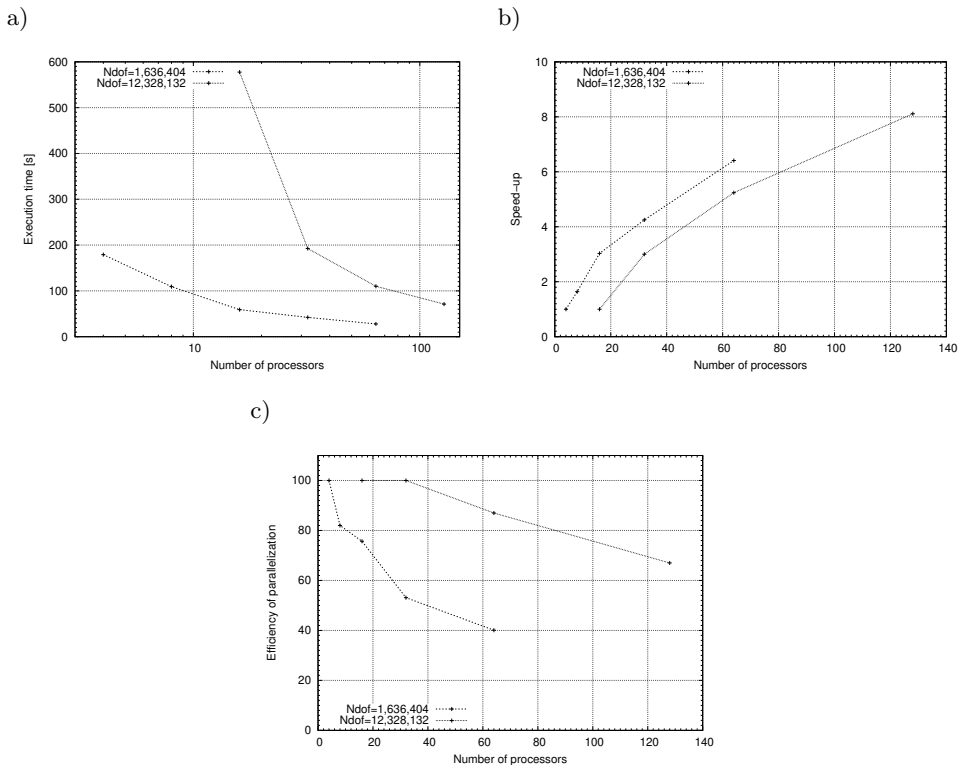


Figure 5. Heat-driven cavity problem – parallel performance metrics for two problem sizes (N_{dof} equal to 1,636,404 and 12,328,132): execution time, speed-up, and parallel efficiency for growing number of cluster processors: a) execution time; b) speed-up; c) efficiency

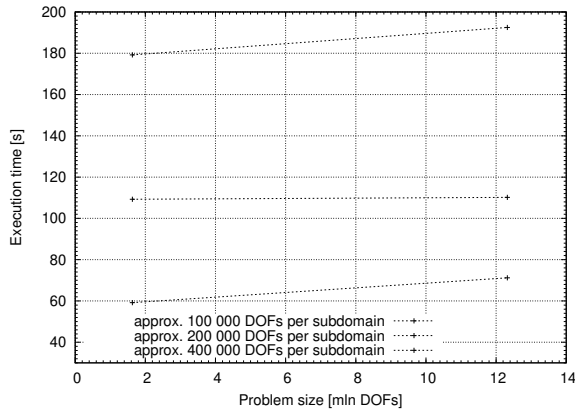


Figure 6. Heat-driven cavity problem – weak scalability results: execution time for three different numbers of DOFs per single subdomain

6. Conclusions

We have presented an efficient solution procedure for simulating large-scale convective heat transfer problems. The efficiency is achieved thanks to the proper selection of algorithms and their parallel implementation, that both guarantee the scalability of the computations. The key ingredient of the procedure is a special preconditioner for linear systems that arise from the discretization of the Navier–Stokes equations. The GMRES solver with the developed preconditioner achieves much shorter solution times than the high-performance PARDISO solver from the Intel MKL library and iterative solvers with ILU preconditioning.

In our future developments, we plan to introduce further algorithmic improvements to the solver and to extend its implementation to hybrid CPU/GPU environments.

Acknowledgements

The work was realized as a part of the fundamental research that is financed by the Ministry of Science and Higher Education, Grant No. 16.16.110.663.

References

- [1] Balay S., Gropp W.D., McInnes L.C., Smith B.F.: Efficient Management of Parallelism in Object-Oriented Numerical Software Libraries. In: E. Arge, A.M. Bruset, H.P. Langtangen (eds.), *Modern Software Tools in Scientific Computing*, pp. 163–202, Birkhäuser Press, 1997.

- [2] Banaś K.: A Modular Design for Parallel Adaptive Finite Element Computational Kernels. In: M. Bubak, G. van Albada, P. Sloot, J. Dongarra (eds.), *Computational Science – ICCS 2004, 4th International Conference, Proceedings, Part II, Lecture Notes in Computer Science*, vol. 3037, pp. 155–162, Springer, 2004.
- [3] Banaś K.: Scalability Analysis for a Multigrid Linear Equations Solver. In: R. Wyrzykowski, J. Dongarra, K. Karczewski, J. Wasniewski, (eds.), *Parallel Processing and Applied Mathematics, Proceedings of VIIth International Conference, PPAM 2007, Gdansk, Poland, 2007, Lecture Notes in Computer Science*, vol. 4967, pp. 1265–1274. Springer, 2008.
- [4] Banaś K., Chłoń K., Cybulka P., Michalik K., Płaszewski P., Siwek A.: Adaptive Finite Element Modelling of Welding Processes. In: M. Bubak, J. Kitowski, K. Wiatr (eds.), *eScience on Distributed Computing Infrastructure – Achievements of PLGrid Plus Domain-Specific Services and Tools, Lecture Notes in Computer Science*, vol. 8500, pp. 391–406. Springer, 2014. doi: 10.1007/978-3-319-10894-0_28.
- [5] Banaś K., Demkowicz L.: Entropy Controlled Adaptive Finite Element Simulations for Compressible Gas Flow, *Journal of Computational Physics*, vol. 126, pp. 181–201, 1996.
- [6] Brandt A.: Multi-Level Adaptive Solutions to Boundary-Value Problems, *Mathematics of Computation*, vol. 31(138), pp. 333–390, 1977. doi: 10.2307/2006422.
- [7] Brooks A.N., Hughes T.J.R.: Streamline upwind/Petrov-Galerkin formulations for convection dominated flows with the particular emphasis on the incompressible Navier–Stokes equations, *Computer Methods in Applied Mechanics and Engineering*, vol. 32(1–3), pp. 199–259, 1982.
- [8] Cai X., Sarkis M.: A Restricted Additive Schwarz Preconditioner for General Sparse Linear Systems, *SIAM Journal on Scientific Computing*, vol. 21, pp. 792–797, 1999.
- [9] Calo V.M., Collier N.O., Pardo D., Paszynski M.R.: Computational complexity and memory usage for multi-frontal direct solvers used in p finite element analysis, *Procedia Computer Science*, vol. 4, pp. 1854–1861, 2011.
- [10] Chow E., Falgout R.D., Hu J.J., Tuminaro R.S., Yang U.M.: A Survey of Parallelization Techniques for Multigrid Solvers. In: M.A. Heroux, P. Raghavan and H.D. Simon (eds.), *Parallel Processing for Scientific Computing* chap. 10, pp. 179–201, SIAM, 2006. doi: 10.1137/1.9780898718133.ch10.
- [11] Cyr E.C., Shadid J.N., Tuminaro R.S.: Stabilization and scalable block preconditioning for the Navier–Stokes equations, *Journal of Computational Physics*, vol. 231(2), pp. 345–363, 2012. doi: 10.1016/j.jcp.2011.09.001.
- [12] Diekert V., Rozenberg G. (eds.): *The Book of Traces*, World Scientific, 1995. doi: 10.1142/2563.

- [13] Elman H., Howle V.E., Shadid J., Shuttleworth R., Tuminaro R.: A taxonomy and comparison of parallel block multi-level preconditioners for the incompressible Navier–Stokes equations, *Journal of Computational Physics*, vol. 227(3), pp. 1790–1808, 2008. doi: 10.1016/j.jcp.2007.09.026.
- [14] Franca L.P., Frey S.L.: Stabilized finite element methods: II. The incompressible Navier–Stokes equations, *Computer Methods in Applied Mechanics and Engineering*, vol. 99(2–3), pp. 209–233, 1992. doi: 10.1016/0045-7825(92)90041-H.
- [15] Franca L.P., Frey S.L., Hughes T.J.R.: Stabilized finite element methods: I. Application to the advective-diffusive model, *Computer Methods in Applied Mechanics and Engineering*, vol. 95(2), pp. 253–276, 1992. doi: 10.1016/0045-7825(92)90143-8
- [16] Goik D., Banaś K.: A Block Preconditioner for Scalable Large Scale Finite Element Incompressible Flow Simulations. In: V.V. Krzhizhanovskaya, G. Závodszky, M.H. Lees, J.J. Dongarra, P.M.A. Sloot, S. Brissos, J. Teixeira (eds.), *Computational Science – ICCS 2020. ICCS 2020, Lecture Notes in Computer Science*, vol. 12139, pp. 199–211. Springer, Cham, 2020. doi: : 10.1007/978-3-030-50420-5_15.
- [17] Henson van E., Yang U.M.: *BoomerAMG*: A parallel algebraic multigrid solver and preconditioner, *Applied Numerical Mathematics*, vol. 41(1), pp. 155–177, 2002. doi: 10.1016/S0168-9274(01)00115-5.
- [18] Jurczyk T., Glut B., Kitowski J.: An Empirical Comparison of Decomposition Algorithms for Complex Finite Element Meshes. In: *PPAM '01: Proceedings of the 4th International Conference on Parallel Processing and Applied Mathematics – Revised Papers*, pp. 493–501, Springer-Verlag, Berlin, Heidelberg, 2001.
- [19] McCormick S.F. (ed.): *Multigrid Methods. Frontiers in Applied Mathematics*, Society for Industrial and Applied Mathematics, Philadelphia, PA, 1987.
- [20] Michalik K., Banaś K., Płaszewski P., Cybułka P.: Modular FEM framework “ModFEM” for generic scientific parallel simulations, *Computer Science*, vol. 14(3), pp. 513–528, 2013. doi: 10.7494/csci.2013.14.3.513.
- [21] Paszyński M., Pardo D., Paszyńska A., Demkowicz L.F.: Out-of-core multi-frontal solver for multi-physics *hp* adaptive problems, *Procedia Computer Science*, vol. 4, pp. 1788–1797, 2011.
- [22] Patankar S.V.: *Numerical Heat Transfer and Fluid Flow. Series on Computational Methods in Mechanics and Thermal Science*, Hemisphere Publishing Corporation, 1980.
- [23] Rehman ur M., Vuik C., Segal G.: A comparison of preconditioners for incompressible Navier–Stokes solvers, *International Journal for Numerical Methods in Fluids*, vol. 57(12), pp. 1731–1751, 2008. doi: 10.1002/flid.1684.
- [24] Saad Y.: *Iterative methods for sparse linear systems*, PWS Publishing, Boston, 1996.

- [25] Saad Y., Schultz M.H.: GMRES: A Generalized Minimal Residual Algorithm for Solving Nonsymmetric Linear Systems, *SIAM Journal on Scientific and Statistical Computing*, vol. 7(3), pp. 856–869, 1986. doi: 10.1137/0907058.
- [26] Schloegel K., Karypis G., Kumar V.: Graph partitioning for high performance scientific simulations. In: J. Dongarra, I. Foster, G. Fox, W. Gropp, K. Kennedy, L. Torczon, A. White, (eds.), *Sourcebook of Parallel Computing*, Morgan Kaufmann Publishers Inc., San Francisco, 2002.
- [27] Smith B., Bjorstad P., Gropp W.: *Domain Decomposition. Parallel Multilevel Methods for Elliptic Partial Differential Equation*, Cambridge University Press, Cambridge, 1996.
- [28] Stüben K.: A review of algebraic multigrid, *Journal of Computational and Applied Mathematics*, vol. 128(1–2), pp. 281–309, 2001. doi: 10.1016/S0377-0427(00)00516-1.
- [29] Volker J.: *Finite Element Methods for Incompressible Flow Problems*, Springer Series in Computational Mathematics, vol. 51, Springer, Cham, 2016.

Affiliations

Damian Goik

AGH University of Science and Technology, al. A. Mickiewicza 30, 30-059 Krakow, Poland

Krzysztof Banaś

AGH University of Science and Technology, al. A. Mickiewicza 30, 30-059 Krakow, Poland, pobanas@cyf-kr.edu.pl, ORCID ID: <https://orcid.org/0000-0002-4045-1530>

Jan Bielański

AGH University of Science and Technology, al. A. Mickiewicza 30, 30-059 Krakow, Poland, jbielan@agh.edu.pl, ORCID ID: <https://orcid.org/0000-0001-5605-8875>

Kazimierz Chłoń

AGH University of Science and Technology, al. A. Mickiewicza 30, 30-059 Krakow, Poland

Received: 11.03.2021

Revised: 10.07.2021

Accepted: 06.09.2021