Advances in Engineering Software 000 (2016) 1–12

Contents lists available at ScienceDirect

Advances in Engineering Software

journal homepage: www.elsevier.com/locate/advengsoft

A parallel implementation of an implicit discontinuous Galerkin finite element scheme for fluid flow problems

Jan Vimmr*, Ondřej Bublík, Aleš Pecka

European Centre of Excellence NTIS – New Technologies for the Information Society, Faculty of Applied Sciences, University of West Bohemia, Univerzitní 8, CZ-306 14, Pilsen, Czech Republic

ARTICLE INFO

Article history:
Received 29 February 2016
Revised 10 November 2016
Accepted 30 November 2016
Available online xxx

MSC:
76N10
65N30
65Y05
35Q35

Keywords:
Discontinuous Galerkin finite element method
Implicit scheme
Compressible Navier–Stokes equations
Parallel computing
Overlapping Schwarz method
Shock capturing
Java RMI

ABSTRACT

The discontinuous Galerkin (DG) method is frequently used in computational fluid dynamics for its stability and high order of accuracy. A disadvantage of the DG method is its high computational demands. The aim of this paper is to weaken this drawback by means of parallelization of the DG algorithm. The computation is performed on a network of computers with distributed memory using the Java Remote Method Invocation, which is included in the Java programming language. The partition of the boundary value problem into n subproblems, which is then solved by n computers separately, is based on the overlapping Schwarz method. On basis of physical nature of the problem, the present paper proposes minimal size of the overlap that allows for only one Schwarz iteration thereby increasing efficiency of parallelization. The scalability and efficiency of the presented parallelization approach is demonstrated on several test problems. In order to stabilize the DG method in presence of shocks, a recently developed technique by Huerta et al. (Int. J. Numer. Meth. Fluids 69(10), 2012, 1614–1632), which introduces discontinuities in basis functions in regions with a shock, is adopted here. A modification of this approach, which lowers the computational and implementational demands, is presented here.

1. Introduction

The discontinuous Galerkin (DG) finite element method [1–6] is currently the most rapidly developing method in the field of computational fluid dynamics. The growing popularity is mainly due to its stability, robustness, low artificial damping and the ability to achieve high-order spatial accuracy. The DG discretization produces a large number of degrees of freedom in comparison with the finite element method or even more so to the finite volume method for the same computational mesh. In other words the DG method has higher computational demands. Papers [7,8] marginally deal with this drawback and compare different time integration methods, namely implicit [9], explicit [4] and explicit local time stepping methods [7,8,10]. The outcome of studies [7,8] is that the computational efficiency of the implicit and explicit local time stepping schemes are comparable, whereas classical explicit schemes are considerably less efficient. In this paper, we employ implicit methods. Namely the backward Euler method is sufficient alternative for problems of finding the steady state. In order to find the time dependent solution we choose a second-order implicit method. We solve the resulting system of linear equations by GMRES [11] with the Jacobi preconditioner.

The main target of this study is to overcome high computational demands of the DG method using parallel computing. The aim is not only to be able to perform the parallel computation on a supercomputer, but also on PCs which might be found in an average office or in a computer laboratory in a college. Such computers are typically connected by slow Ethernet or Wi-Fi and have various hardware setups and various operating systems installed. The computation may run in the background while the computers are being used, since the hardware is rarely fully utilized during ordinary office work. For these purposes the Java programming language seems to be an appropriate choice, since software developed in Java is not dependent on the type or version of the operating system. It is still a common misconception that code written in Java, which is a dynamically compiled language, is considerably slower in comparison with statically compiled languages such as C, C++ or Fortran. While there is still a noticeable difference between

* Corresponding author. Tel: +420 377 632 304.
E-mail address: jvimmr@kme.zcu.cz (J. Vimmr).

http://dx.doi.org/10.1016/j.advengsoft.2016.11.007
0965-9978/

the execution times in favour of statically compiled languages, the gap significantly shrank after the introduction of the JIT compiler in 1998 and after the release of extensive performance updates to the JIT compiler in the following years. The fact that Java is catching up with C and Fortran was already reported in 2001 by Bull et al. [12] who concluded that 'the performance gap between Java and more traditional scientific programming languages is no longer a wide gulf'. We believe that although the performance of Java does not match the performance of C, C++ or Fortran, the ease of coding and platform independence compensates for this inconvenience. Furthermore, if necessary, the critical sections of the code can be written in other languages and then included in the Java application using the Java Native Interface (JNI).

The communication among computers is usually realized by the Message Passing Interface (MPI). In the present work, we apply the Java Remote Method Invocation (Java RMI) included in the Java programming language. The Java RMI technology enables to call a Java method on a remote virtual machine and, as opposed to MPI, is easily applicable to a heterogeneous computer network. Studies [13,14] compare Java RMI with MPI.

For parallelization of algorithms for solving boundary value problems on systems with distributed memory, the domain decomposition method is commonly used. Examples of domain decomposition methods are the Schwarz method [15–19] or the Schur complement Method [20]. In this work, there are two levels at which the implicit DG algorithm is parallelized - among different nodes (computers) in a computer network and within each one. At the level of network, we employ the Schwarz method [15–17], where the computational domain is divided into several overlapping sub-domains. The computation is then performed on each sub-domain by a different node. At the level of individual nodes, we subject the GMRES solver to parallelization. More specifically, the individual vector operations involved in GMRES (e.g. matrix-vector multiplications) are parallelized within each node using Java threads.

This paper also addresses stabilization of the DG method. The dissipation due to the numerical fluxes, which arises from the jump terms, is not sufficient to stabilize the solution in presence of shocks when high order of approximation is used. One option is to damp the solution near shocks, which in combination with mesh refinement is an effective approach. A very popular form of damping is adding artificial viscosity into the shock regions [9,21,22]. Another possibility is to subject the solution in the critical region to a limiting process [23,24], which unfortunately decreases approximation order. We adopt a novel technique developed by Huerta et al. [25], which stabilizes the approximate solution near a shock with the aid of numerical fluxes, by introducing basis functions, which have discontinuities inside elements, in regions with a shock. We also propose a simplification of this stabilization approach, which decreases the implementational and computational demands and is still fairly effective.

We choose three test problems, namely the subsonic viscous flow in a 2D convergent channel, supersonic inviscid flow in the 2D Mach 3 wind tunnel with a step and subsonic inviscid flow in the 2D GAMM channel. For each test problem we perform several computations on various numbers of nodes and threads and calculate the efficiency and speedup of the parallelization.

The outline of this paper is as follows. In Section 2 we define the mathematical model based on the compressible Navier–Stokes equations in two dimensions and briefly review the DG discretization along with the implicit time integration. The stabilization approach is presented in Section 2.3. Section 3 contains the main topic of the present paper, which is parallelization. The numerical test are performed in Section 4 and their results are discussed and conclusion is drawn in Section 5.

2. Mathematical model and implicit DG scheme

Before moving onto the parallelization itself, which is the main topic of this paper, we first describe the mathematical model and the implicit DG scheme employed in the upcoming simulations. The developed parallelization approach is not dependent on the discretization, hence other schemes can be parallelized in the same manner.

2.1. Governing equations

The non-linear system of Navier–Stokes equations, which describes the compressible viscous flow, written in a conservative vector form for the two dimensional case is

$$\frac{\partial \boldsymbol{w}}{\partial t} + \sum_{s=1}^{2} \frac{\partial}{\partial x_s} \boldsymbol{f}_s(\boldsymbol{w}) = \sum_{s=1}^{2} \frac{\partial}{\partial x_s} \boldsymbol{f}_s^v(\boldsymbol{w}, \nabla \boldsymbol{w}), \tag{1}$$

where $\boldsymbol{x} = [x_1, x_2]^T \in \Omega \subset \mathbb{R}^2$ is the vector of Cartesian coordinates, $t \in [0, \mathcal{T}]$ is the time variable, $\boldsymbol{w}(\boldsymbol{x}, t) = [\varrho, \varrho u_1, \varrho u_2, E]^T$ is the vector of conservative variables with density ϱ, velocity vector $\boldsymbol{u} = [u_1, u_2]^T$ and total energy per unit volume E. Furthermore, the inviscid and viscous fluxes are respectively defined by

$$\begin{aligned} \boldsymbol{f}_s(\boldsymbol{w}) &= [\varrho u_s, \varrho u_s u_1 + p\delta_{s1}, \varrho u_s u_2 + p\delta_{s2}, (E+p)u_s]^T, \\ \boldsymbol{f}_s^v(\boldsymbol{w}) &= [0, \tau_{1s}, \tau_{2s}, u_1\tau_{1s} + u_2\tau_{2s} + k\partial T/\partial x_s]^T, \end{aligned} \qquad s = 1, 2 \tag{2}$$

with pressure p, temperature T and thermal conductivity k. The stress tensor τ_{ij} is given by

$$\tau_{ij} = \mu \left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i} - \frac{2}{3} \frac{\partial u_k}{\partial x_k} \delta_{ij} \right), \quad i, j = 1, 2, \tag{3}$$

where μ is the dynamic viscosity and δ_{ij} is the Kronecker delta. The system of governing Eqs. (1) is completed by the constitutive relation for ideal gas

$$p = (\kappa - 1) \left[E - \frac{1}{2} \varrho \sum_{s=1}^{2} u_s^2 \right], \tag{4}$$

with the adiabatic index κ. The term $k\partial T/\partial x_s$ is often rewritten as follows

$$k \frac{\partial T}{\partial x_s} = \frac{\kappa}{\kappa - 1} \frac{\mu}{\mathrm{Pr}} \frac{\partial}{\partial x_s} \left(\frac{p}{\varrho} \right), \tag{5}$$

where Pr is the Prandtl number. Constants κ and Pr take values $\kappa = 1.4$ and $\mathrm{Pr} = 0.72$ for air. The system of Eqs. (1) is equipped with the initial condition

$$\boldsymbol{w}(\boldsymbol{x}, 0) = \boldsymbol{w}_0(\boldsymbol{x}), \quad \boldsymbol{x} \in \Omega. \tag{6}$$

The boundary condition prescribed on the boundary $\partial\Omega$ of the computational domain $\Omega \in \mathbb{R}^2$ are as follows:

- At the subsonic inlet, stagnation pressure p_0^{in}, stagnation density ϱ_0^{in}, angle of attack α^{in} are prescribed.
- At the supersonic inlet, density ϱ^{in}, pressure p^{in}, velocities u_1^{in} and u_2^{in} are prescribed.
- At the subsonic outlet, the static pressure p^{out} is prescribed.
- At the supersonic outlet, no boundary condition is prescribed.
- In case the flow is viscous, zero traction $\sum_{s=1}^{2} \tau_{ks} n_s = 0,\ k = 1, 2$ and zero normal derivative $\frac{\partial T}{\partial \boldsymbol{n}} = 0$ are also prescribed at the inlet and outlet.
- On the wall, zero velocity components $u_s = 0,\ s = 1, 2$ and zero normal derivative $\frac{\partial T}{\partial \boldsymbol{n}} = 0$ are prescribed in case of viscous flow

Please cite this article as: J. Vimmr et al., A parallel implementation of an implicit discontinuous Galerkin finite element scheme for fluid flow problems, Advances in Engineering Software (2016), http://dx.doi.org/10.1016/j.advengsoft.2016.11.007