Contents lists available at ScienceDirect

International Journal of Heat and Mass Transfer

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

A superposition-based parallel discrete operator splitting method for incompressible flows

K.K.Q. Zhang [a], K. Sengupta [a], K. Xia [b], W.J. Minkowycz [a], F. Mashayek [a,*]

[a] Department of Mechanical and Industrial Engineering, University of Illinois at Chicago, 842 West Taylor Street, Chicago, IL 60607, USA
[b] Caterpillar Inc., Technical Center, P.O. Box 1875, Peoria, IL 61656, USA

ARTICLE INFO

Article history:
Received 5 January 2009
Received in revised form 19 February 2009
Accepted 19 February 2009
Available online 26 March 2009

Keywords:
Parallel computing
Superposition
Discrete operator splitting
Finite element method
Incompressible flow

ABSTRACT

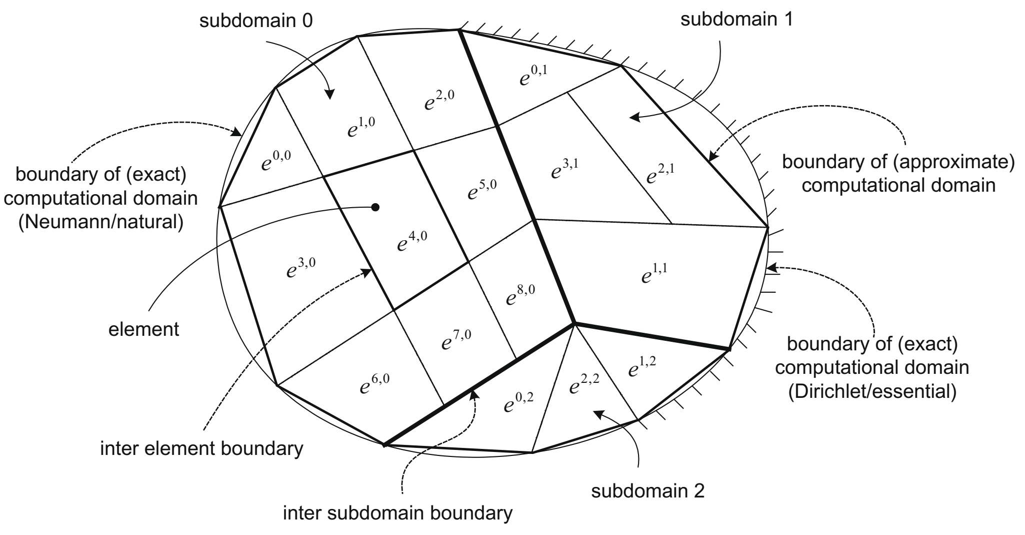
Juxtaposition-based domain decomposition requires complicated pre-processing and communications of pre-consolidated data, and is restricted to field problems. In this paper, we propose a superposition-based domain decomposition parallelization, which employs element-by-element construction, processor-level assembling, and condensed random data structure. Superposition-based parallelization shows great flexibility in partitioning the computational domains, communicates more consolidated data, and can be applied beyond field problems. Moreover, superposition-based parallelization can, as an option, follow the same numerical process as its serial counterpart to produce digit-by-digit identically the same result, which makes code development and debugging much easier. Solving large scale indefinite systems continues to pose as a challenging issue for incompressible flows. In this paper, we propose the discrete operator splitting (DOS) technique to break the original ill-natured large indefinite system into two smaller well-natured definite systems coupled through source terms. The underpinning idea of the technique is to seamlessly combine the splitting and iterations together. Equipped with the parallelization and DOS, we present in details the superposition-based parallel discrete operator splitting finite element method and apply it to incompressible Navier–Stokes flows. Backward-facing step flow, cavity flow, and pipe flow are simulated to demonstrate the success of the method.

1. Introduction

The past two decades witnessed a monotonic growth in high performance computing on distributed computers for tackling large scale problems such as those arising from fluid flows. Domain decomposition [9,16,28,26,34,24,23,25,35,3] stands out as the primary parallelization method for solving a field problem. In a typical domain decomposition, the overall domain is decomposed into several subdomains consisted of many elements, as shown in Fig. 1(a). Computation of each subdomain is typically conducted on each processor, and the inter-subdomain information is communicated among processors. In Fig. 1(a), the numbering of elements are relative to local processor, as indicated by the superscripts of $e^{0,1}$ for example. Alternatively, the whole domain is discretized into elements, then these elements are grouped into different subdomains. In this way, elements are naturally numbered in a global sense. In present paper, this approach (with its two variants) is called *juxtaposition-based parallelization* (or *domain decomposition* as commonly called in scientific computing), simply because the overall domain is a juxtaposition of all subdomains. Exactly due to the same nature, juxtaposition-based domain decomposition is restricted to field problems, where an actual physical domain must be clearly identified. Also, the method involves complicated pre-processing for the processing-stage data communications. For simulation of moving boundary problems, these pre-processing level communications need to be moved into time marching loop, and this makes computer programming more error-prone. Moreover, it is very common for the juxtaposition-based domain-decomposition to communicate excessive number of quantities (in continuous sense). A good example is the Poisson solver illustrated in [16], where a significant drop in parallel scale up is demonstrated as the number of processors increases. In juxtaposition-based domain decomposition, data to be communicated is determined in the early stage of system formation. If this is determined in the system solving stage, the number of messages can be considerably reduced. Finally, the domain decomposition parallelization does not follow the same numeric process of its corresponding serial algorithm. In the serial case, the whole domain is handled by a single processor. In the parallel case, each processor handles a subdomain and all other processors act like boundary conditions providers. The parallel one follows a different numerical process from the serial one. Therefore, although the parallel algorithm can be trivially reduced to its serial counterpart, their results slightly differ from each other. In other words, domain decomposition is not just a matter of logistics, and this makes code debugging

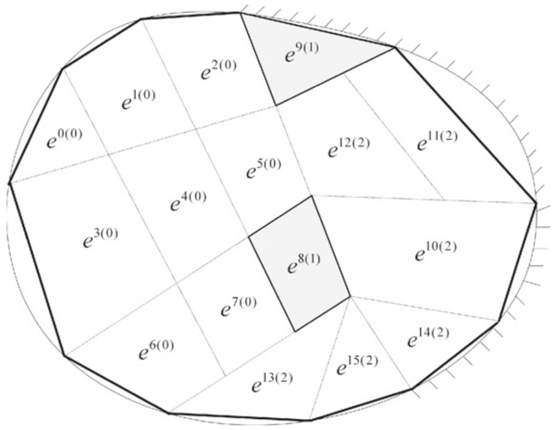
* Corresponding author.
E-mail address: mashayek@uic.edu (F. Mashayek).

0017-9310/$ - see front matter © 2009 Elsevier Ltd. All rights reserved.
doi:10.1016/j.ijheatmasstransfer.2009.02.019



subdomain 0
subdomain 1
subdomain 2
boundary of (exact) computational domain (Neumann/natural)
boundary of (approximate) computational domain
boundary of (exact) computational domain (Dirichlet/essential)
element
inter element boundary
inter subdomain boundary
$e^{0,0}$ $e^{1,0}$ $e^{2,0}$ $e^{3,0}$ $e^{4,0}$ $e^{5,0}$ $e^{6,0}$ $e^{7,0}$ $e^{8,0}$
$e^{0,1}$ $e^{1,1}$ $e^{2,1}$ $e^{3,1}$
$e^{0,2}$ $e^{1,2}$ $e^{2,2}$

(a)



$e^{0(0)}$ $e^{1(0)}$ $e^{2(0)}$ $e^{3(0)}$ $e^{4(0)}$ $e^{5(0)}$ $e^{6(0)}$ $e^{7(0)}$
$e^{8(1)}$ $e^{9(1)}$
$e^{10(2)}$ $e^{11(2)}$ $e^{12(2)}$ $e^{13(2)}$ $e^{14(2)}$ $e^{15(2)}$

(b)

Fig. 1. (a) Juxtaposition-based domain decomposition, where each subdomain is typically computed by one processor. As an example, the second superscript of $e^{4,0}$ is used to index the processor while the first superscript is the processor-level local index. (b) Geometric representation of superposition-based partition with global numbering of elements. As an example, the second superscript of $e^{8(1)}$ is used to index the processor while the first superscript is the global index.

more difficult. Debugging a parallel code is more effective if its serial counterpart should produce identically the same result.

In the superposition-based parallelization, no domain decomposition is needed. Instead, the overall domain is directly discretized into elements and each processor handles a subset of all elements, grouped by numbering of elements as an example. Each processor stores local matrices, local vectors (hereafter 'vectors' excludes 'solution vectors'), and global solution vectors. The majority of expensive operations, including matrix–vector-product, are then parallelized based on their additive property, and results of operations by individual processors are communicated. The elements operated by a particular processor may scatter in the physical space, a great flexibility over domain decomposition. Fig. 1(b) shows two shaded elements belonging to the same processor. It should be pointed out that the figure is merely for illustration purpose and not needed in actual parallel implementation. Superposition-based parallelization is relatively simple and can be applied beyond field problems. In addition, numerically it can be chosen to follow its corresponding serial algorithm exactly and a parallel code can produce digit-by-digit identically the same values as its serial counterpart, which makes coding and debugging of parallel algorithms very easy. For this reason, a more specific name for the current parallelization is *superposition-based non-numeric parallelization*. Note that in juxtaposition-based domain decomposition, a parallel code running on several processors will not produce digit-by-digit the same result as its serial counterpart. More explanation on this "digit-by-digit" matter can be found in Section 6.3. Although the system formation is completely in a local sense in superposition-based parallelization, the overall structure of system solving is in a global sense and solution vectors are updated globally (that is, all processors update the same global solution vectors). It has to be pointed out that under many situations the global-sense structure does not hamper efficiency of the algorithm because the expensive calculations of vectors, which are used to update solution vectors, are conducted locally by individual processors and communicated among them. To make communications more efficient, all data is compressed and stored randomly so that only condensed local vectors are communicated among processors. In short, the superposition-based parallelization achieves both the simplicity and the reasonable efficiency for a broad class of problems. Furthermore, the technique can be applied to non-field problems and is easy to debug.

In vector form, the non-dimensional governing equations for incompressible flows read

$$\nabla \cdot \vec{u} = 0, \tag{1a}$$

$$\frac{\partial \vec{u}}{\partial t} + \vec{u} \cdot \nabla \vec{u} = -\nabla p + \frac{1}{Re}[\nabla \cdot \nabla \vec{u} + \nabla(\nabla \cdot \vec{u})] + \vec{f}, \tag{1b}$$

where $\vec{u}$ and p stand for velocity and pressure, respectively, $\vec{f}$ stands for body force, and Re stands for Reynolds number. For decades, a numerical simulation of this incompressible flow system has remained as one of the most intriguing topics in computational fluid dynamics. The pressure is neither fully decoupled from the velocity nor fully engaged in the system. And due to the lack of a time derivative, the pressure is not an evolutionary quantity. However, the actual challenge is revealed only when one attempts to solve the system employing the mixed formulation used for compressible flows, where all unknown variables are solved simultaneously. The indefinite system due to the mixed formulation, in which velocities and pressure are solved simultaneously without any manipulations, is ill-conditioned [7]. In such a system, a relatively small change in some entry of the matrix results in a relatively large change in the solution. Hence, accumulated computer round-off errors or some inherent perturbations of iterative processes make the convergence very hard to achieve. When the size of the system increases or when the physical solutions tend to be more rugged as a consequence of higher Reynolds numbers or discontinuities, the conditioning of the discrete system will further deteriorate and consequently the convergence will become even more difficult. In some situations, eventually the discrete system becomes singular and no solution can be found.

Splitting methods exploit the weak coupling between the pressure and the velocity. Splitting methods can be divided into three categories, continuous splitting, semi-discrete splitting, and discrete splitting. Both continuous splitting and the semi-discrete splitting require *numerical boundary condition* (which is in contrast to physical boundary condition) for the pressure, and the semi-discrete splitting require additional boundary condition for some intermediate variable. Discrete splitting often does not require numerical boundary condition for the pressure [36]. The first two papers on the subject introduced Harlow and Welch's marker-and-cell (MAC) method [18] and Chorin's projection method [4]. MAC takes the discrete approach while the projection method follows the semi-discrete path, and both require some numerical boundary condition for the pressure. An example of continuous splitting can be found in [15].

The issue of the boundary conditions for the Poisson equation, as well as boundary conditions for intermediate velocities (whenever introduced), has in the past sparked a considerable debate [18,4,5,27,30,6,22,33,15,1,13,14,20,8,29,32,31,2,17] (in chronic order). As a matter of fact, the numerical boundary condition for the pressure Poisson equation is already implied in the system and is actually not required in practice, as shown in the pressure-correction type of approximate factorization technique by Dukowicz and Dvinsky [8] and in the pressure-update type of approximate factorization technique by Perot [29]. However, our experience shows that only appropriate elements (or spatial scheme) can faithfully reflect the inherent pressure boundary condition. In semi-discrete splitting and continuous splitting, extra boundary conditions for pressure are introduced, but they must be consistent with the inherent boundary conditions. In this paper, we take the discrete splitting approach which necessarily and sufficiently converts the original large indefinite system with mixed formulation into smaller subsystems. Thus, whether or not there is a need to resort to numerical boundary conditions is up to the original mixed formulation.

The elusive issue of splitting error has also drawn substantial attention, in that many of those papers on the issue of numerical boundary conditions also concern the issue of time accuracy. Approximate factorization techniques remove the splitting error through an approximate inversion of some matrix. Quarteroni et al. [31] presented a framework for splitting methods and approximate factorization techniques, including Perot's approach. The factorization technique [36] takes a different path in terms of restoring time accuracy. However, the discrete operator splitting technique to be introduced in this paper is free from the worry of any splitting errors.

The remainder of this paper is organized as follows: generic differential equations and their weak forms of integral equations are derived in Section 2. In Section 3, the superposition-based parallelization, one of two key contributions of this paper, is elucidated. In Section 4, the discrete operator splitting, the other key contribution of this paper, is derived rigorously. In Section 5 the above two techniques are combined and a procedure for the system solving is stipulated. Numerical examples are given in Section 6. Finally, some conclusions are drawn in Section 7.

2. System formulation

We would like to obtain integral governing equations in weak form for both Cartesian and axisymmetric cylindrical coordinates. The generic coordinates are introduced for this purpose and its orientation is shown in Fig. 2. The generic differential governing equations are

$$\frac{\partial u_j}{\partial x_j} + c_a \frac{u_2}{x_2} = 0, \tag{2a}$$

$$\frac{\partial u_i}{\partial t} + u_j \frac{\partial u_i}{\partial x_j} - f_i = -\frac{\partial p}{\partial x_i} + \frac{1}{Re}\frac{\partial}{\partial x_j}\left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i}\right) + \frac{c_a}{Re}\frac{1}{x_2}\left(\frac{\partial u_1}{\partial x_2} + \frac{\partial u_2}{\partial x_1}\right)\delta_{1i} + \frac{2c_a}{Re}\frac{1}{x_2}\left(\frac{\partial u_j}{\partial x_j} + \frac{\partial u_2}{\partial x_2}\right)\delta_{2i}, \tag{2b}$$

where $c_a = 0$ for Cartesian coordinates and $c_a = 1$ for axisymmetric coordinates. In this section the governing equations are transformed into the format for numerical discretization. The variational form for Eqs. (2a) and (2b) reads

$$-(2\pi)^{c_a}\int_\Omega q\left(\frac{\partial u_j}{\partial x_j} + c_a\frac{u_2}{x_2}\right)x_2^{c_a}\,d\Omega + (2\pi)^{c_a}\int_\Omega w_i\left(\frac{\partial u_i}{\partial t} + u_j\frac{\partial u_i}{\partial x_j} - f_i\right)x_2^{c_a}\,d\Omega + (2\pi)^{c_a}\int_\Omega w_i\frac{\partial p}{\partial x_i}x_2^{c_a}\,d\Omega - \frac{(2\pi)^{c_a}}{Re}\int_\Omega w_i\frac{\partial}{\partial x_j}\left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i}\right)x_2^{c_a}\,d\Omega - c_a\frac{(2\pi)^{c_a}}{Re} \times \int_\Omega w_1\left(\frac{\partial u_1}{\partial x_2} + \frac{\partial u_2}{\partial x_1}\right)d\Omega - 2c_a\frac{(2\pi)^{c_a}}{Re}\int_\Omega w_2\left(\frac{\partial u_j}{\partial x_j} + \frac{\partial u_2}{\partial x_2}\right)d\Omega = 0,$$

where $d\Omega = dx_1\,dx_2$ for 2D and axisymmetric cases, and $d\Omega = dx_1\,dx_2\,dx_3$ for 3D cases. For convenience of boundary condition imposition and reduction of derivatives, integration by parts for two terms are carried out,

$$\int_\Omega w_i\frac{\partial p}{\partial x_i}x_2^{c_a}\,d\Omega = \int_\Omega\left[\frac{\partial}{\partial x_i}\left(w_i p x_2^{c_a}\right) - \frac{\partial w_i}{\partial x_i}p x_2^{c_a} - w_i p c_a \delta_{2i}\right]d\Omega = \oint_\Gamma w_i n_i p x_2^{c_a}\,d\Gamma - \int_\Omega\frac{\partial w_i}{\partial x_i}p x_2^{c_a}\,d\Omega - c_a\int_\Omega w_2 p\,d\Omega,$$

and

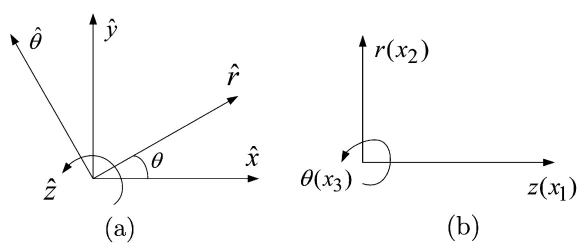


Fig. 2. (a) Relations between cylindrical and Cartesian coordinates, (b) generic coordinates.

$$\begin{aligned}\int_\Omega w_i \frac{\partial}{\partial x_j}\left(\frac{\partial u_i}{\partial x_j}+\frac{\partial u_j}{\partial x_i}\right)x_2^{c_a}d\Omega &= \int_\Omega\left\{\frac{\partial}{\partial x_j}\left[w_i\left(\frac{\partial u_i}{\partial x_j}+\frac{\partial u_j}{\partial x_i}\right)x_2^{c_a}\right]\right.\\ &\quad\left.-\frac{\partial w_i}{\partial x_j}\left(\frac{\partial u_i}{\partial x_j}+\frac{\partial u_j}{\partial x_i}\right)x_2^{c_a}-w_i\left(\frac{\partial u_i}{\partial x_j}+\frac{\partial u_j}{\partial x_i}\right)c_a\delta_{2j}\right\}d\Omega\\ &= \oint_\Gamma w_i\left(\frac{\partial u_i}{\partial x_j}+\frac{\partial u_j}{\partial x_i}\right)x_2^{c_a}n_j d\Gamma-\int_\Omega\frac{\partial w_i}{\partial x_j}\left(\frac{\partial u_i}{\partial x_j}+\frac{\partial u_j}{\partial x_i}\right)x_2^{c_a}d\Omega-c_a\\ &\quad\times\int_\Omega w_i\left(\frac{\partial u_i}{\partial x_2}+\frac{\partial u_2}{\partial x_i}\right)d\Omega.\end{aligned}$$

Hence the variational form becomes

$$\begin{aligned}&-\int_\Omega q\left(\frac{\partial u_j}{\partial x_j}+c_a\frac{u_2}{x_2}\right)x_2^{c_a}d\Omega+\int_\Omega w_i\left(\frac{\partial u_i}{\partial t}+u_j\frac{\partial u_i}{\partial x_j}\right)x_2^{c_a}d\Omega\\ &-\int_\Omega\frac{\partial w_i}{\partial x_i}px_2^{c_a}d\Omega-c_a\int_\Omega w_2pd\Omega+\frac{1}{Re}\int_\Omega\frac{\partial w_i}{\partial x_j}\left(\frac{\partial u_i}{\partial x_j}+\frac{\partial u_j}{\partial x_i}\right)x_2^{c_a}d\Omega\\ &+\frac{c_a}{Re}\left[\int_\Omega w_i\left(\frac{\partial u_i}{\partial x_2}+\frac{\partial u_2}{\partial x_i}\right)d\Omega-\int_\Omega w_1\left(\frac{\partial u_1}{\partial x_2}+\frac{\partial u_2}{\partial x_1}\right)d\Omega-2\right.\\ &\left.\times\int_\Omega w_2\left(\frac{\partial u_j}{\partial x_j}+\frac{\partial u_2}{\partial x_2}\right)d\Omega\right]=\oint_\Gamma w_in_j\left[-p\delta_{ij}+\frac{1}{Re}\left(\frac{\partial u_i}{\partial x_j}+\frac{\partial u_j}{\partial x_i}\right)\right]\\ &\times x_2^{c_a}d\Gamma+\int_\Omega w_if_ix_2^{c_a}d\Omega.\end{aligned}$$

After consolidation of axisymmetric diffusion terms, the variational weight-function form of integral equations reduces to

$$\begin{aligned}&-\int_\Omega q\left(\frac{\partial u_j}{\partial x_j}+c_a\frac{u_2}{x_2}\right)x_2^{c_a}d\Omega+\int_\Omega w_i\left(\frac{\partial u_i}{\partial t}+u_j\frac{\partial u_i}{\partial x_j}\right)x_2^{c_a}d\Omega\\ &-\int_\Omega\partial w_i\partial x_ipx_2^{c_a}d\Omega-c_a\int_\Omega w_2pd\Omega+\frac{1}{Re}\int_\Omega\frac{\partial w_i}{\partial x_j}\left(\frac{\partial u_i}{\partial x_j}+\frac{\partial u_j}{\partial x_i}\right)x_2^{c_a}d\Omega\\ &-\frac{2c_a}{Re}\int_\Omega w_2\frac{\partial u_j}{\partial x_j}d\Omega=\oint_\Gamma w_in_j\left[-p\delta_{ij}+\frac{1}{Re}\left(\frac{\partial u_i}{\partial x_j}+\frac{\partial u_j}{\partial x_i}\right)\right]x_2^{c_a}d\Gamma\\ &+\int_\Omega w_if_ix_2^{c_a}d\Omega=\oint_\Gamma w_in_j\sigma_{ij}x_2^{c_a}d\Gamma+\int_\Omega w_if_ix_2^{c_a}d\Omega=\oint_\Gamma w_iT_ix_2^{c_a}d\Gamma\\ &+\int_\Omega w_if_ix_2^{c_a}d\Omega,\end{aligned}\tag{3}$$

where T_i stands for the surface traction. Let all quantities be expressed in terms of interpolation functions

$$\begin{aligned}&u_i=u_i^l\psi_l,\quad w_i=w_i^m\psi_m,\quad u_j=u_j^r\psi_r,\\ &p=p^l\psi_l^p,\quad q=q^m\psi_m^p,\\ &f_i=f_i^l\psi_l,\quad T_i=T_i^l\psi_l,\end{aligned}$$

then

$$\begin{aligned}&-q^m\left(\int_\Omega\psi_m^p\frac{\partial\psi_l}{\partial x_j}x_2^{c_a}d\Omega\right)u_j^l-c_aq^m\left(\int_\Omega\psi_m^p\psi_ld\Omega\right)u_2^l\\ &+w_i^m\left(\int_\Omega\psi_m\psi_lx_2^{c_a}d\Omega\right)\frac{\partial u_i^l}{\partial t}+w_i^m\left[\left(\int_\Omega\psi_m\psi_r\frac{\partial\psi_l}{\partial x_j}x_2^{c_a}d\Omega\right)u_j^r\right]u_i^l\\ &-w_i^m\left(\int_\Omega\frac{\partial\psi_m}{\partial x_i}\psi_l^px_2^{c_a}d\Omega\right)p^l-c_aw_2^m\left(\int_\Omega\psi_m\psi_l^pd\Omega\right)p^l\\ &+w_i^m\left(\frac{1}{Re}\int_\Omega\frac{\partial\psi_m}{\partial x_j}\frac{\partial\psi_l}{\partial x_j}x_2^{c_a}d\Omega\right)u_i^l+w_i^m\left(\frac{1}{Re}\int_\Omega\frac{\partial\psi_m}{\partial x_j}\frac{\partial\psi_l}{\partial x_i}x_2^{c_a}d\Omega\right)u_j^l\\ &-w_2^m\left(\frac{2c_a}{Re}\int_\Omega\psi_m\frac{\partial\psi_l}{\partial x_j}d\Omega\right)u_j^l\\ &=w_i^m\left(\oint_\Gamma\psi_m\psi_lx_2^{c_a}d\Gamma\right)T_i^l+w_i^m\left(\int_\Omega\psi_m\psi_lx_2^{c_a}d\Omega\right)f_i^l,\end{aligned}\tag{4}$$

which is the *semi-continuous* (continuous in time and discrete in space) weak form of the governing equations. The term *semi-discrete* is reserved for equations discrete in time and continuous in space. Note that Eq. (4) is in strong form if all admissible weight functions are exhausted, but in weak form if weight functions are the subset of admissible functions. We should emphasize that the integral form and the integration by part are not approximate operations and are not the reasons for the terminology the *weak form*. Also in Eq. (4), the trial functions for pressure ψ^p and for velocity ψ may be any suitable interpolation functions, such as spectral functions and Lagrangian polynomials. In other words, Eq. (4) is not restricted to finite element methods.

With second-order semi-implicit time scheme, Eq. (4) is discretized further to

$$\begin{aligned}&q^m\left(-\int_\Omega\psi_m^p\frac{\partial\psi_l}{\partial x_j}x_2^{c_a}d\Omega\right)u_j^{l,(n+1)}+c_aq^m\left(-\int_\Omega\psi_m^p\psi_ld\Omega\right)u_2^{l,(n+1)}\\ &+w_i^m\left(\frac{1}{\Delta t}\int_\Omega\psi_m\psi_lx_2^{c_a}d\Omega\right)u_i^{l,(n+1)}+w_i^m\left(-\int_\Omega\frac{\partial\psi_m}{\partial x_i}\psi_l^px_2^{c_a}d\Omega\right)p^{l,(n+\frac{1}{2})}\\ &+c_aw_2^m\left(-\int_\Omega\psi_m\psi_l^pd\Omega\right)p^{l,(n+\frac{1}{2})}-\frac{1}{2}w_i^m\left(-\frac{1}{Re}\int_\Omega\frac{\partial\psi_m}{\partial x_j}\frac{\partial\psi_l}{\partial x_j}x_2^{c_a}d\Omega\right)u_i^{l,(n+1)}\\ &-\frac{1}{2}w_i^m\left(-\frac{1}{Re}\int_\Omega\frac{\partial\psi_m}{\partial x_j}\frac{\partial\psi_l}{\partial x_i}x_2^{c_a}d\Omega\right)u_j^{l,(n+1)}-\frac{1}{2}c_aw_2^m\left(\frac{2}{Re}\int_\Omega\psi_m\frac{\partial\psi_l}{\partial x_j}d\Omega\right)u_j^{l,(n+1)}\\ &=w_i^m\left(\frac{1}{\Delta t}\int_\Omega\psi_m\psi_lx_2^{c_a}d\Omega\right)u_i^{l,(n)}-\frac{3}{2}w_i^m\left[\left(\int_\Omega\psi_m\psi_r\frac{\partial\psi_l}{\partial x_j}x_2^{c_a}d\Omega\right)u_j^{r,(n)}\right]u_i^{l,(n)}\\ &\quad+\frac{1}{2}w_i^m\left[\left(\int_\Omega\psi_m\psi_r\frac{\partial\psi_l}{\partial x_j}x_2^{c_a}d\Omega\right)u_j^{r,(n-1)}\right]u_i^{l,(n-1)}+\frac{1}{2}w_i^m\left(-\frac{1}{Re}\int_\Omega\frac{\partial\psi_m}{\partial x_j}\frac{\partial\psi_l}{\partial x_j}x_2^{c_a}d\Omega\right)u_i^{l,(n)}\\ &\quad+\frac{1}{2}w_i^m\left(-\frac{1}{Re}\int_\Omega\frac{\partial\psi_m}{\partial x_j}\frac{\partial\psi_l}{\partial x_i}x_2^{c_a}d\Omega\right)u_j^{l,(n)}+\frac{1}{2}c_aw_2^m\left(\frac{2}{Re}\int_\Omega\psi_m\frac{\partial\psi_l}{\partial x_j}d\Omega\right)u_j^{l,(n)}\\ &\quad+w_i^m\left(\oint_\Gamma\psi_m\psi_lx_2^{c_a}d\Gamma\right)T_i^{l,(n+\frac{1}{2})}+w_i^m\left(\int_\Omega\psi_m\psi_lx_2^{c_a}d\Omega\right)f_i^{l,(n+\frac{1}{2})},\end{aligned}\tag{5}$$

where superscripts (n), $(n+\frac{1}{2})$, and $(n+1)$ are used to indicate time levels. The variational nature in Eq. (5) demands the coefficients of each weight function vanish so that a series of discrete equations can be generated.

Eq. (5) applies to general interpolation polynomials for the pressure (and the velocity). Next, the pressure is restricted to discontinuous constant distribution. Due to the discontinuous constant distribution, the pressure and its corresponding weight function can be pulled out of the integrals, which can be simplified to

$$\int_\Omega\left(\frac{\partial u_j}{\partial x_j}+c_a\frac{u_2}{x_2}\right)x_2^{c_a}d\Omega=\int_\Omega\frac{\partial}{\partial x_j}(u_jx_2^{c_a})d\Omega=\oint_\Gamma n_ju_jx_2^{c_a}d\Gamma,$$

and similarly

$$\int_\Omega\frac{\partial w_i}{\partial x_i}x_2^{c_a}d\Omega+c_a\int_\Omega w_2d\Omega=\oint_\Gamma n_iw_ix_2^{c_a}d\Gamma.$$

Finally, the fully discrete system for the generic semi-discontinuous weak form becomes

$$\begin{aligned}&q\left(-\oint_\Gamma n_j\psi_lx_2^{c_a}d\Gamma\right)u_j^{l,(n+1)}+w_i^m\left(\frac{1}{\Delta t}\int_\Omega\psi_m\psi_lx_2^{c_a}d\Omega\right)u_i^{l,(n+1)}\\ &+w_i^m\left(-\oint_\Gamma n_i\psi_mx_2^{c_a}d\Gamma\right)p^{(n+\frac{1}{2})}-\frac{1}{2}w_i^m\left(-\frac{1}{Re}\int_\Omega\frac{\partial\psi_m}{\partial x_j}\frac{\partial\psi_l}{\partial x_j}x_2^{c_a}d\Omega\right)u_i^{l,(n+1)}\\ &-\frac{1}{2}w_i^m\left(-\frac{1}{Re}\int_\Omega\frac{\partial\psi_m}{\partial x_j}\frac{\partial\psi_l}{\partial x_i}x_2^{c_a}d\Omega\right)u_j^{l,(n+1)}-\frac{1}{2}c_aw_2^m\left(\frac{2}{Re}\int_\Omega\psi_m\frac{\partial\psi_l}{\partial x_j}d\Omega\right)u_j^{l,(n+1)}\\ &=w_i^m\left(\frac{1}{\Delta t}\int_\Omega\psi_m\psi_lx_2^{c_a}d\Omega\right)u_i^{l,(n)}-\frac{3}{2}w_i^m\left[\left(\int_\Omega\psi_m\psi_r\frac{\partial\psi_l}{\partial x_j}x_2^{c_a}d\Omega\right)u_j^{r,(n)}\right]u_i^{l,(n)}\\ &\quad+\frac{1}{2}w_i^m\left[\left(\int_\Omega\psi_m\psi_r\frac{\partial\psi_l}{\partial x_j}x_2^{c_a}d\Omega\right)u_j^{r,(n-1)}\right]u_i^{l,(n-1)}+\frac{1}{2}w_i^m\left(-\frac{1}{Re}\int_\Omega\frac{\partial\psi_m}{\partial x_j}\frac{\partial\psi_l}{\partial x_j}x_2^{c_a}d\Omega\right)u_i^{l,(n)}\\ &\quad+\frac{1}{2}w_i^m\left(-\frac{1}{Re}\int_\Omega\frac{\partial\psi_m}{\partial x_j}\frac{\partial\psi_l}{\partial x_i}x_2^{c_a}d\Omega\right)u_j^{l,(n)}+\frac{1}{2}c_aw_2^m\left(\frac{2}{Re}\int_\Omega\psi_m\frac{\partial\psi_l}{\partial x_j}d\Omega\right)u_j^{l,(n)}\\ &\quad+w_i^m\left(\oint_\Gamma\psi_m\psi_lx_2^{c_a}d\Gamma\right)T_i^{l,(n+\frac{1}{2})}+w_i^m\left(\int_\Omega\psi_m\psi_lx_2^{c_a}d\Omega\right)f_i^{l,(n+\frac{1}{2})},\end{aligned}\tag{6}$$

Eq. (6) involves several domain integrals and one boundary integral. These integrals are intended on the whole computational domain in the physical space, as indicated by Ω and Γ. However, due to the additive property of integration, the integral on the overall domain is the sum of integrals on all individual elements. Also, trial functions and test functions, which are defined on the whole space and identified by nodes, are selected such that any function completely vanishes on an element where the corresponding node is not presented. As a consequence, the integration becomes localized. Furthermore, the integration on an element in the physical space can be mapped into the integration on the master element in the transform space.

3. Superposition-based parallelization

3.1. Element-by-element construction

In a traditional implementation, the element level matrices and vectors are calculated first, without application of boundary conditions, then assembled to form global matrices and vectors. Next, boundary conditions are applied to global matrices and vectors which are directly used to solve discrete unknowns. This approach is inconvenient for parallel computation. In element-by-element (EBE) approach, element level matrices and vectors are calculated and imposed by boundary conditions, and the resulting system is directly solved with element level matrices and vectors, through the inter-element iteration. The global matrices and vectors are never formed, resulting in a significant reduction of memory storage. Compared with the traditional approach, EBE requires almost negligible memory. However, in most scientific computing, the constraint is the speed rather than the memory. EBE leaves the memory capacity of a computer untapped but greatly sacrifices the speed. In every linear iteration of EBE the system must be re-formed; in contrast, the traditional approach requires system re-formation only for nonlinear iteration. It can be easily noticed that a system formation in finite element method roughly costs two dozens times as one sweep of linear iteration. Hence, for a transient problem the EBE approach is about two dozens times as slow as the traditional approach.

To overcome the drawbacks of both approaches, we first follow the EBE approach, that is, boundary conditions are applied to the element matrices and element vectors. Then, contrary to EBE, element matrices and vectors are assembled to form processor-level local matrices and vectors (corresponding to global matrices and vectors in the serial counterpart), which will be used in system solving. This is called *element-by-element construction*. Natural boundary conditions are automatically imposed on matrices and vectors by being incorporated into the formulation, hence, they never cause problems in this approach. What should be concerned are the essential boundary conditions (EBC).

A common practice for applying EBC on a specific discrete unknown is to set the diagonal entry to unity and set all off-diagonal entries to zero for the row indexed by this discrete unknown. Then the corresponding entry on the right-hand-side vector is set equal to the value as specified by EBC. It is important that this practice of applying EBC can actually swap the order with the assembling. Suppose a discrete unknown, imposed by an EBC, appears in two

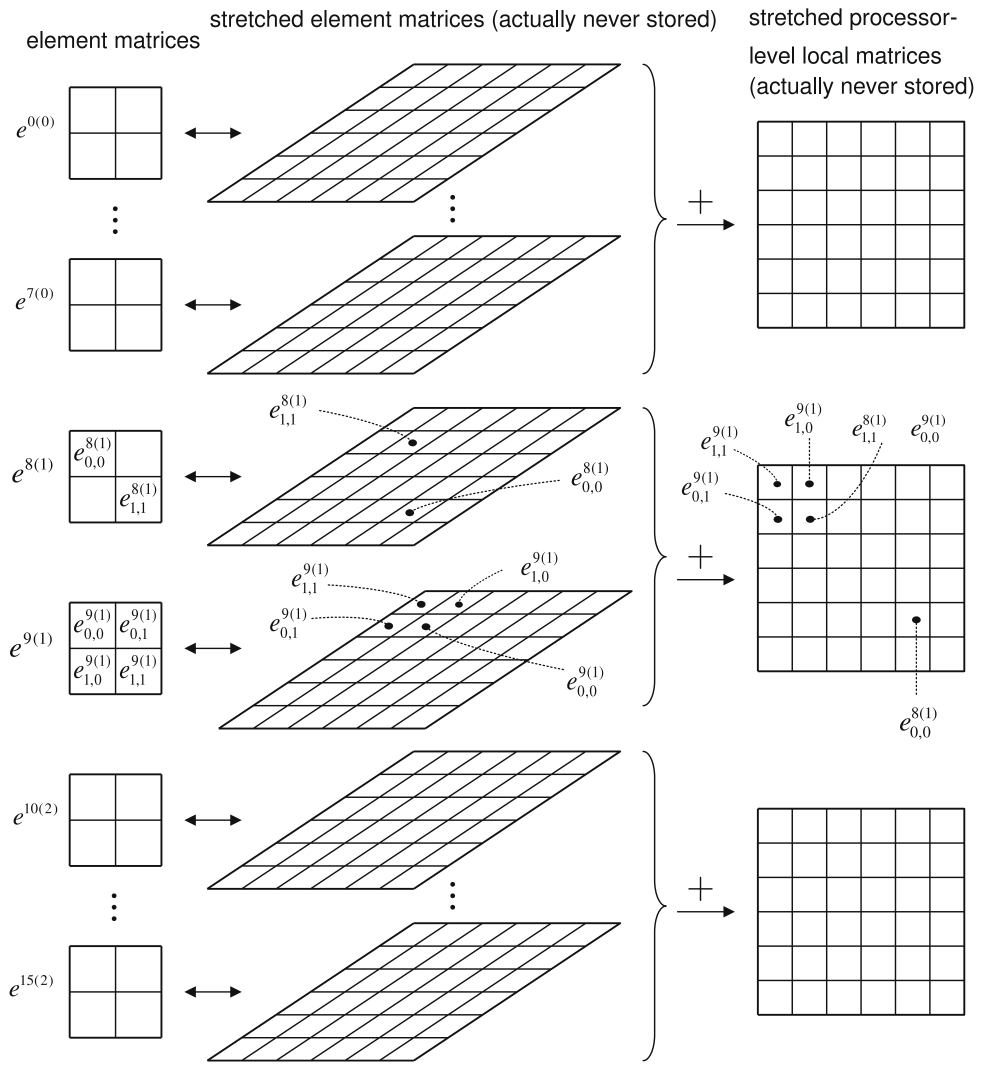


Fig. 3. An illustration of partition and superposition on three processors. The sign "+" stands for superposition.

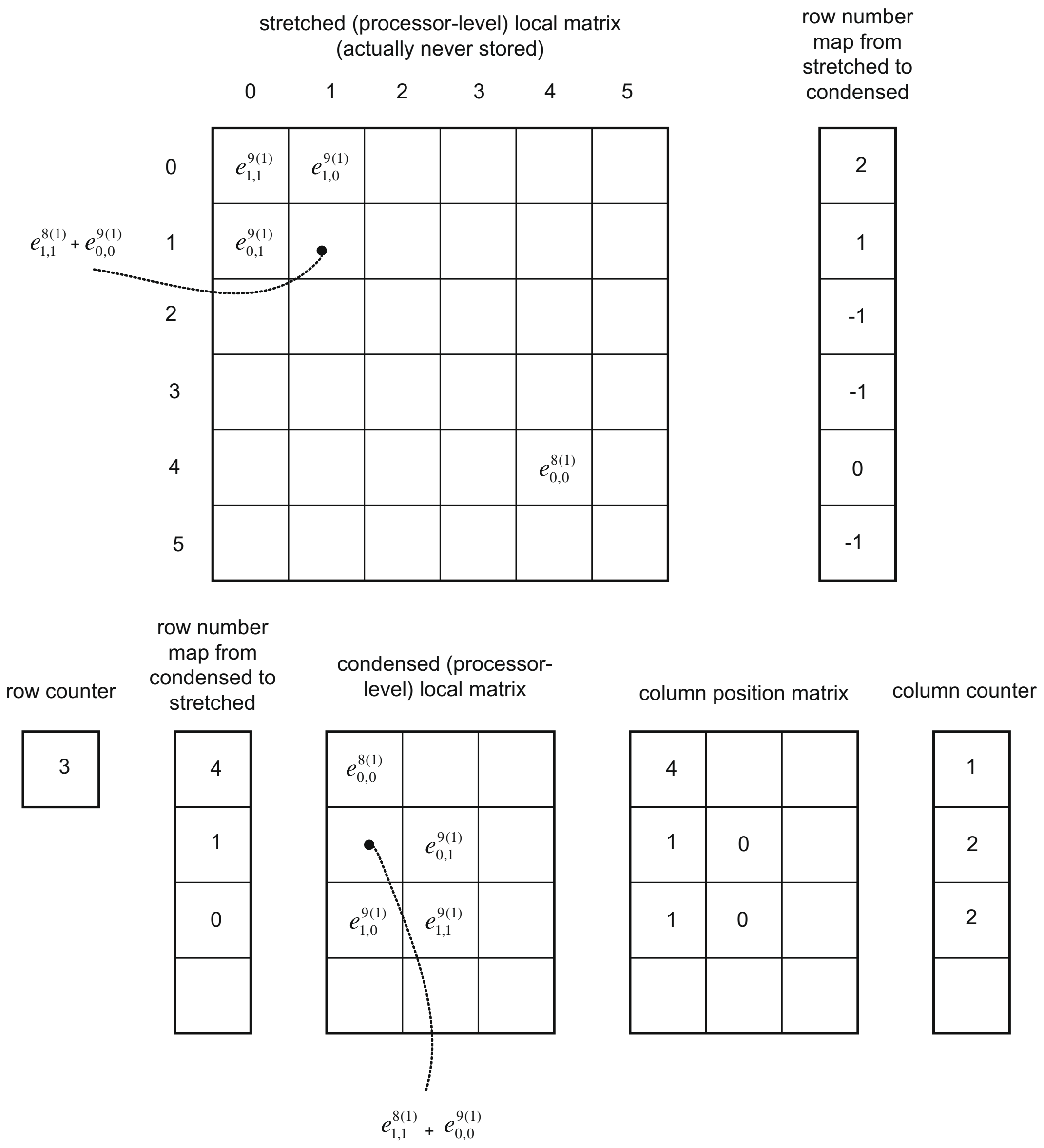
different elements. After the common practice of EBC application (just described) on both elements and superposition of the resulting element matrices and vectors onto the global ones, the global matrix has 2 in the diagonal entry and zeros in all off-diagonal entries, and the corresponding entry on right-hand-side has twice of the value specified by EBC. Therefore, the EBE construction does not cause any problem for EBC.

3.2. Grouping and superposition

In this subsection, we illustrate through an example how to partition the domain and what the superposition means. The original full domain shown in Fig. 1(b) is first discretized into many elements as the usual, and for each element an element matrix is constructed, shown in Fig. 3 in the first column. Then an arbitrary method of grouping, such as according to numbering of all these elements, is adopted to allocate elements to processors. The second superscript of $e^{8(1)}$, for instance, indicates that this element belongs to processor 1 (the second processor). Each element matrix has a stretched version as shown in Fig. 3 in the second column. As usual, numbering inside element matrices follows element-level nodes and numbering inside stretched element matrices follows global nodes. The stretched ones, all in global sense, are actually never stored and are for illustration only. As shown in Fig. 3 in the third column, these stretched element matrices are assembled into stretched process-level local matrices, which again are never stored and are for illustration only. For example, element matrices $e^{8(1)}$ and $e^{9(1)}$ belong to processor 1 and are assembled to obtain a processor-level local matrix. Quantity $e^{8(1)}_{1,1}$ and quantity $e^{9(1)}_{0,0}$ are superposed onto the same entry in the stretched local matrix.

3.3. Condensed random data structure

As shown in the previous subsection, element matrices and stretched element matrices can be converted into each other one-to-one, and the latter are assembled to obtain stretched processor-level local matrices. All stretched matrices are large and sparse, and should be avoided. This subsection illustrates


stretched (processor-level) local matrix (actually never stored)
row number map from stretched to condensed
row counter
row number map from condensed to stretched
condensed (processor-level) local matrix
column position matrix
column counter

	0	1	2	3	4	5
0	$e^{9(1)}_{1,1}$	$e^{9(1)}_{1,0}$				
1	$e^{9(1)}_{0,1}$	$e^{8(1)}_{1,1}+e^{9(1)}_{0,0}$				
2						
3						
4					$e^{8(1)}_{0,0}$	
5						

row number map from stretched to condensed
2
1
-1
-1
0
-1

row counter
3

row number map from condensed to stretched
4
1
0

condensed (processor-level) local matrix		
$e^{8(1)}_{0,0}$		
$e^{8(1)}_{1,1}+e^{9(1)}_{0,0}$	$e^{9(1)}_{0,1}$	
$e^{9(1)}_{1,0}$	$e^{9(1)}_{1,1}$	

column position matrix		
4		
1	0	
1	0	

column counter
1
2
2

Fig. 4. Condensed random data structure on a specific processor. The stretched processor-level local matrix and the condensed processor-level local matrix can be mutually converted into each other.

how to convert stretched process-level local matrix into condensed processor-level local matrix, again one-to-one. In the serial counterpart, the assembled (global) matrix is sparse on each row, but does not contain any row which is totally blank. Illustrated in Fig. 3, in the parallel version, all stretched process-level local matrices are not only sparse on each row but also contain many blank rows. Thus, the stretched local matrices can be compressed in both column-wise (horizontal) and row-wise (vertical) directions. The actual bulk of data is stored in condensed processor-level local matrices (shown in Fig. 4), whose numbers of rows and columns are pre-estimated. Using chain or cross chain data structures, the lengths of condensed local matrices can be more flexible and data storage can be reduced to some extent, however, data access speed will be significantly slowed down. Now, we illustrate through the same example how the data is actually stored.

As shown in Fig. 3, processor 1 processes element 8 and obtains two quantities, $e^{8(1)}_{0,0}$ and $e^{8(1)}_{1,1}$. As shown in Fig. 4, quantity $e^{8(1)}_{0,0}$ (supposedly positioned at the fifth row and fifth column in stretched local matrix) is actually filled into the first row and first column of condensed process-level local matrix. Accordingly, the row number (4) and column number (again 4) of $e^{8(1)}_{0,0}$ in stretched local matrix are recorded in "row number map from condensed to stretched" and "column position matrix", respectively; row number of $e^{8(1)}_{0,0}$ in condensed local matrix is recorded in "row number map from stretched to condensed"; and both row counter and the column counter increase by 1. Quantities $e^{8(1)}_{1,1}$ is filled into the second row and first column of the condensed local matrix. Accordingly, its positions in both matrices must be recorded mutually and counters must be updated. Other quantities can be placed in the condensed local matrix similarly.

In terms of row numbers, in the stretched local matrix $e^{8(1)}_{1,1}$ is ahead of $e^{8(1)}_{0,0}$, while in the condensed local matrix $e^{8(1)}_{1,1}$ is behind $e^{8(1)}_{0,0}$. In the stretched case, the row number is determined by global numbering, while in the condensed case it is determined by the order being processed. Hence, the data structure for the condensed local matrix is random. However, the randomness should be limited to matrix construction stage only, and later access to the condensed local matrix must be deterministic. The "row number map from stretched to condensed" serves for this purpose. In Fig. 4 any "counter" must be initialized to zero and "row number map from stretched to condensed" must be initialized to some number such as −1 for a proper tracking of data. Now we can replace expensive stretched local matrix by condensed local matrix because of their one-to-one mutual conversion.

To this point, element matrices belonging to a specific processor can be assembled to obtain condensed processor-level local matrices. Further details of the superposition-based parallelization have to be postponed to the system solving section, which is after the next section.

4. Discrete operator splitting

4.1. Forward derivation

Discretization of Eqs. (2a) and (2b) or Eq. (6) eventually produces the following discrete system

$$Au + Gp = S_u, \tag{7a}$$

$$Du = S_p, \tag{7b}$$

where A in the momentum equation is the coefficient matrix for the velocity, G is the coefficient matrix for the pressure, D in the continuity equation is the coefficient matrix for the velocity, and S_u and S_p are the right-hand-side known vectors for the momentum and the continuity equations, respectively. The momentum Eq. (7a), involved by both u and p, reflects the relation between the velocity and the pressure, while the continuity Eq. (7b), involved by u only, imposes a constraint on the velocity. It is well known that such a system is ill-conditioned, which makes it difficult to iteratively solve the whole large system simultaneously. Alternatively, we may consider to iterate between the two equations back and forth (and each equation can be temporarily solved directly or iteratively). This is named source-term iteration (or subsystem iteration). However, the primitive coupled system, Eqs. (7a) and (7b), is not fit for source-term iteration. Instead, we need an equation involved by both the velocity and the pressure and with invertible coefficient matrix for the velocity, which is exactly the primitive momentum Eq. (7a). And we need an equation involved by both the velocity and the pressure and with invertible coefficient matrix for the pressure, which is still absent. Hence, we would like to replace the continuity equation by a combination of the momentum equation and the continuity equation. In momentum Eq. (7a), we split matrix A into the diagonal part A^d and the off-diagonal part $(A - A^d)$, consequently

$$\begin{aligned}
&(A - A^d + A^d)u + Gp = S_u \\
&\quad \Longleftrightarrow A^d u + Gp = S_u - (A - A^d)u \\
&\quad \Longleftrightarrow u + A^{-d}Gp = A^{-d}[S_u - (A - A^d)u] \\
&\quad \Rightarrow Du + DA^{-d}Gp = DA^{-d}[S_u - (A - A^d)u],
\end{aligned} \tag{8}$$

where A^{-d} stands for the inverse of A^d. Let the continuity Eq. (7b) be incorporated into the above momentum equation and let the original momentum Eq. (7a) retained, we obtain the following system

$$\begin{aligned}
&DA^{-d}Gp = -S_p + DA^{-d}[S_u - (A - A^d)u], \\
&Au = S_u - Gp.
\end{aligned}$$

For convenience, we introduce

$$D^* \equiv DA^{-d},$$

so that the discrete forms of the momentum equation and the continuity equation become

$$\begin{aligned}
&D^*Gp = -(S_p - D^*S_u) - D^*(Au - A^d u), \\
&Au = S_u - Gp.
\end{aligned}$$

Define

$$\begin{aligned}
&L \equiv D^*G, \\
&S_p^* \equiv S_p - D^*S_u,
\end{aligned}$$

finally we have two well-posed subsystems

$$Lp = -S_p^* - D^*(Au - A^d u) \equiv b_p(u), \tag{9a}$$

$$Au = S_u - Gp \equiv b_u(p). \tag{9b}$$

4.2. Backward derivation

So far we have proved that the original indefinite system (Eqs. (7a) and (7b)) implies the two definite subsystems (Eqs. (9a) and (9b)). This is not adequate yet. We must show these two systems are equivalent to each other. All moves in the above forward derivation actually can be reversed *directly*, except the move to deduce Eq. (8). Hence, we would like to take a slightly different path to prove the reverse, that the two definite subsystems (Eqs. (9a) and (9b)) also implies the original indefinite system (Eqs. (7a) and (7b)).

Eq. (9b) directly recovers momentum Eq. (7a). With incorporations of definitions of D^*, L, and S_p^*, Eq. (9a) can be recast as

$$DA^{-d}Gp = -S_p + DA^{-d}S_u - DA^{-d}(Au - A^d u) = -S_p + Du + DA^{-d}(S_u - Au).$$

With Eq. (9b) taken into account, the above equation reduces to continuity Eq. (7b). Therefore, Eqs. (9a) and (9b) are necessary and sufficient conditions of Eqs. (7a) and (7b).

4.3. Discussions

Now the original single ill-conditioned large system (Eqs. (7a) and (7b)) is replaced by two smaller well-conditioned subsystems (Eqs. (9a) and (9b)). With predicted source terms, Eqs. (9a) and (9b) can be iterated alternatingly. Within each source-term iteration, an iterative technique can be used to solve two linear subsystems. The overall technique introduced in this paper to tackle indefinite systems is named discrete operator splitting (DOS).

The idea behind the present method is to combine the effort of decoupling of the pressure from the velocity with the iterative process of solving a linear system (which in fact is also a decoupling process). The key move is to split matrices into diagonal and off-diagonal parts. The technique neither introduces intermediate velocity, nor requires additional numerical boundary conditions for the pressure, and nor incurs any splitting errors. Although the idea of discrete splitting has been decades old, the DOS is presented here for the first time. It differs from the algorithm introduced in MAC method [18] and algorithms used in some commercial software, which all require numerical boundary conditions. It differs from the segregated technique [19], where the key step of the present method was never introduced. It differs from factorization techniques (such as [29]), in which the pressure and the velocity are completely decoupled. The process of system solving is the process of decoupling of all discrete unknowns. However, in general we should not be so lucky to decouple a group of discrete unknowns (such as those for the pressure) from the remaining discrete unknowns (such as those for the velocity). The pressure and the velocity in continuous splitting, semi-discrete splitting, and a part of discrete splitting are coupled through boundary conditions. In segregated methods and DOS, they are coupled through source terms. Note that in continuous splitting the pressure and the velocity also initially appear to be coupled through source terms. However, after incompressibility imposed, the source terms only involve quantities in previous time steps. The derivation of DOS includes both a forward derivation and a backward derivation, so that the original ill-conditioned system is converted into a well-conditioned system sufficiently and necessarily. It should also be mentioned that the DOS technique can be applied to various meshes or methods, such as a high-order implicit-in-space compact finite difference method [38] and a fully conservative finite volume method on lattice grid [37].

5. System solving

5.1. Source-term iteration and linear iteration

We assign superscripts $(n), (l)$, and (k) to identify time level, source-term iteration, and linear iteration, respectively. Superscript (m) is reserved for nonlinear iteration, but it has been circumvented in this paper because of the semi-implicit time scheme. We assign indices to Eqs. (9a) and (9b) to obtain

$$Lp^{(n+1,l+1)} = b_p(u^{**}), \tag{10a}$$

$$Au^{(n+1,l+1)} = b_u(p^{**}), \tag{10b}$$

where predicted solutions for source-term iterations are defined as

$$u^{**} \equiv \beta u^{(n+1,l)} + (1-\beta)u^{(n+1,l-1)},$$

$$p^{**} \equiv \beta p^{(n+1,l+1)} + (1-\beta)p^{(n+1,l)}.$$

In the above definitions the optimal value for β varies from case to case but convergence is always achieved if $\beta = 1.0$.

Using the vector-by-vector Jacobi iteration, the p-subsystem (10a) and the u-subsystem (10b) turn out to be

$$p^{(n+1,l+1,k+1)} = p^* + \omega L^{-d}[b_p(u^{**}) - Lp^*], \tag{11}$$

$$u^{(n+1,l+1,k+1)} = u^* + \omega A^{-d}[b_u(p^{**}) - Au^*], \tag{12}$$

where predicted solutions for linear iterations are defined as

$$p^* \equiv \alpha p^{(n+1,l+1,k)} + (1-\alpha)p^{(n+1,l+1,k-1)},$$

$$u^* \equiv \alpha u^{(n+1,l+1,k)} + (1-\alpha)u^{(n+1,l+1,k-1)}.$$

In the above definitions $\alpha \geqslant 1.0$, typically $\alpha = 1.6$. In Eqs. (11) and (12), $0 < \omega < 1.0$, and typically $\omega = 0.3$. Eqs. (11) and (12) are the primary equations used to update the pressure and the velocity.

5.2. Key steps of the algorithm for system solving

Although the system formation is completely in a local sense in superposition-based non-numeric parallelization, the overall structure of system solving is in a global sense and solution vectors are updated globally (that is, all processors update the same global solution vectors). It has to be pointed out that under many situations the global-sense structure does not hamper efficiency of the algorithm because the expensive calculations of vectors, which are used to update solution vectors, are conducted locally by individual processors and communicated among them. To make communications more efficient, all data is compressed and stored randomly so that only condensed local vectors are communicated among processors. In short, the superposition-based parallelization achieves both the simplicity and the reasonable efficiency for a broad class of problems. Furthermore, the technique can be applied to non-field problems and is easy to debug.

So far the algorithm for system solving has been presented in a narrative way, not organized and tailored for computer programming. To make the actual implementation easier, we summarize the key steps of the algorithm. Some standard operations such as recording of solution vectors are not included. As throughout this paper, "local" in the following indicates the processor level (not the element level) and "condensed" refers to a randomly compressed data structure. Again, we emphasize that any "solution vector" is stored globally and does not take part in communications; typically a "matrix" is stored locally and condensed; and typically a "vector", which excludes "solution vector", is stored locally and condensed. Apart from global solution vectors, there are several global vectors for intermediate storage. Inside the time marching process, these ordered steps are as follows:

- Calculate coefficient matrices $\widehat{A}, \widehat{G}$, and $\widehat{D}$, and right-hand-side vectors $\widehat{S}_u$ and $\widehat{S}_p$, all in condensed local sense (here, the hat indicates that the matrix is condensed and local).
- Find condensed local diagonal matrix $\widehat{A}^d$, the diagonal part of condensed local matrix $\widehat{A}$; *communicate* all $\widehat{A}^d$ to obtain global diagonal matrix A^d.
- Calculate condensed local matrix $\widehat{D}^* = \widehat{D}A^{-d}$, where $\widehat{D}$ is a condensed local divergence matrix and A^{-d} is the inverse of global diagonal matrix A^d.
- Calculate condensed local Laplace matrix $\widehat{L} = \widehat{D}^*G$, where $\widehat{D}^*$ is a condensed local matrix and G is a global matrix. This involves *communication* of condensed local gradient matrix $\widehat{G}$ to obtain G, however, G is never stored.
- Find condensed local diagonal matrix $\widehat{L}^d$, the diagonal part of condensed local matrix $\widehat{L}$; *communicate* all $\widehat{L}^d$ to obtain global diagonal matrix L^d.
- *Communicate* all condensed local vector $\widehat{S}_u$ to obtain global vector S_u.
- Calculate condensed local vector $\widehat{S}^*_p = \widehat{S}_p - \widehat{D}^*S_u$, the solution-independent part of the source term of the p-subsystem (this begins the source term iteration).

- Calculate intermediate condensed local vector $\widehat{S}_q = \widehat{A}u^{**} - \widehat{A}^d u^{**}$, where the predicted solution u^{**} for source-term iteration is global; *communicate* all $\widehat{S}_q$ to obtain intermediate global vector S_q.
- Calculate the source term, condensed local vector $\hat{b}_p = -\widehat{S}^*_p - \widehat{D}^* S_q$, for the p-subsystem (this begins the p-subsystem linear iteration).
- Calculate intermediate condensed local vector $\Delta\hat{p} = \omega L^{-d}(\hat{b}_p - \widehat{L}p^*)$, where predicted solution p^* for linear iteration is global; *communicate* all $\Delta\hat{p}$ to obtain intermediate global vector Δp.
- Update global solution vector $p = p^* + \Delta p$ (make a decision to end the p-subsystem linear iteration, for instance, by the number of iterations).
- Calculate the source term, condensed local vector $\hat{b}_u = \widehat{S}_u - \widehat{G}p^{**}$, where predicted solution p^{**} for source-term iteration is global (this begins the u-subsystem linear iteration).
- Calculate intermediate condensed local vector $\Delta\hat{u} = \omega A^{-d}(\hat{b}_u - \widehat{A}u^*)$, where the predicted solution u^* for linear iteration is global; *communicate* all $\Delta\hat{u}$ to obtain intermediate global vector Δu.
- Update global solution vector $u = u^* + \Delta u$ (make a decision to end the u-subsystem linear iteration, for instance, by the number of iterations. Finally, make a decision to end the source-term iteration, for instance, by the relative changes of solutions u and p over their previous values, both as the consequence of the source-term iteration).

In the above algorithm, we have three inevitable communications of condensed local vectors and one communication of condensed local matrix G (which can be avoided in both semi-discrete splitting and continuous splitting methods) outside the source-term iteration. There is one communication inside the source-term iteration but outside both linear iterations. And there is one communication for each linear iteration. In a time dependent problem, the number of source-term iterations is large. Hence, the number of communications of condensed local vectors inside the source-term iteration determines the efficiency of the parallel algorithm. For a steady problem solved by a time marching process on a fixed mesh, quantities $\widehat{A}, \widehat{G}, \widehat{D}, \widehat{A}^d, A^d, \widehat{D}^*, \widehat{L}, \hat{L}^d$, and $\hat{L}$ in the above algorithm are invariant. Therefore, several relevant steps right after formation of the system and before iteration of the system solving can be avoided.

6. Numerical results

To demonstrate the success of the method we select the backward-facing step flow, lid-driven cavity flow, and axisymmetric pipe flow as benchmark numerical examples. All these three laminar and steady flows are solved by a time marching process. Constant time steps vary from case to case, but are small enough to ensure stability and time accuracy. A representative time step size is 0.001.

6.1. 2D backward-facing step flow

The backward-facing step flow is short-handed as the *backstep* flow (Fig. 5), which serves as a popular benchmark problem. In this flow, the geometry is simple enough so that various numerical methods can be quickly coded and tested. The flow is rich in structure. At low *Re*, one eddy is formed in the corner between the step and the bottom wall. As *Re* increases, the second eddy attaches the top wall. When *Re* reaches some critical value, the flow becomes unsteady and eddies are formed alternatingly near the bottom and top walls, which is somewhat similar to the von Karman vortex street. The reattachment length of the primary eddy is sensitive to numerical methods. Numerical methods with artificial diffusion or other smoothing techniques often find it hard to predict an accurate reattachment length. The flow involves the issue of open boundary conditions and the singularity around the step could create some difficulties for numerical methods [11]. The location of the exit must be far downstream to minimize the influence of exit conditions on upstream flows, hence the flow is expensive to simulate. Because of all these reasons, the backstep flow is an ideal benchmark problem to demonstrate the capability of DOS technique. The results shown in Fig. 6(a) and (b) are based on a 256×16 mesh. Fig. 6(a) shows that the exit location set at $x = 32.0$ is sufficiently good, whereas at a location $x = 25.0$ the flow is not fully developed. Fig. 6(b) shows good agreement at $x = 7.0$ between the present result and that from [11].

6.2. 2D cavity flow

The lid-driven cavity flow is a well-known benchmark problem with well documented numerical results. The configuration is shown in Fig. 7. The sudden movement of the top lid introduces a singularity, which makes the flow riveting. Some elements, such as linear element fail to simulate this flow very well, unless some artificial smoothing is used. The 4–1 element, linear polynomial for velocity and constant (and discontinuous) for pressure, was tried to solve the cavity flow. Unfortunately, except for very low *Re* the velocity profiles show a lot of wiggles. The 8–1 element is immune from this issue and can be used to solve this flow reliably. Fig. 8 shows the comparison between the numerical result, which is based on a mesh of 256×256 elements, and that from [12].

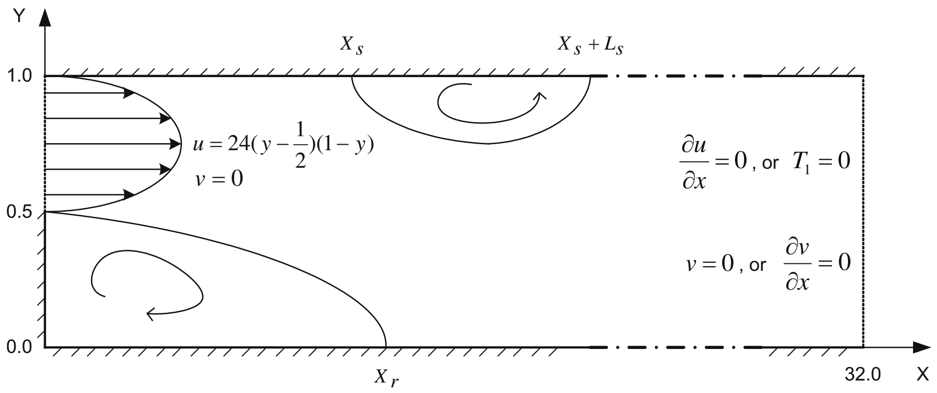

Y
1.0
0.5
0.0
X_S
$X_S + L_S$
$u = 24(y - \frac{1}{2})(1 - y)$
$v = 0$
$\frac{\partial u}{\partial x} = 0$, or $T_1 = 0$
$v = 0$, or $\frac{\partial v}{\partial x} = 0$
X_r
32.0
X

Fig. 5. Configuration for the backstep flow. Both velocity components vanish on top and bottom walls and the step. The pressure is free from boundary conditions, or vanished as implied in free traction boundary condition (if used) at exit.

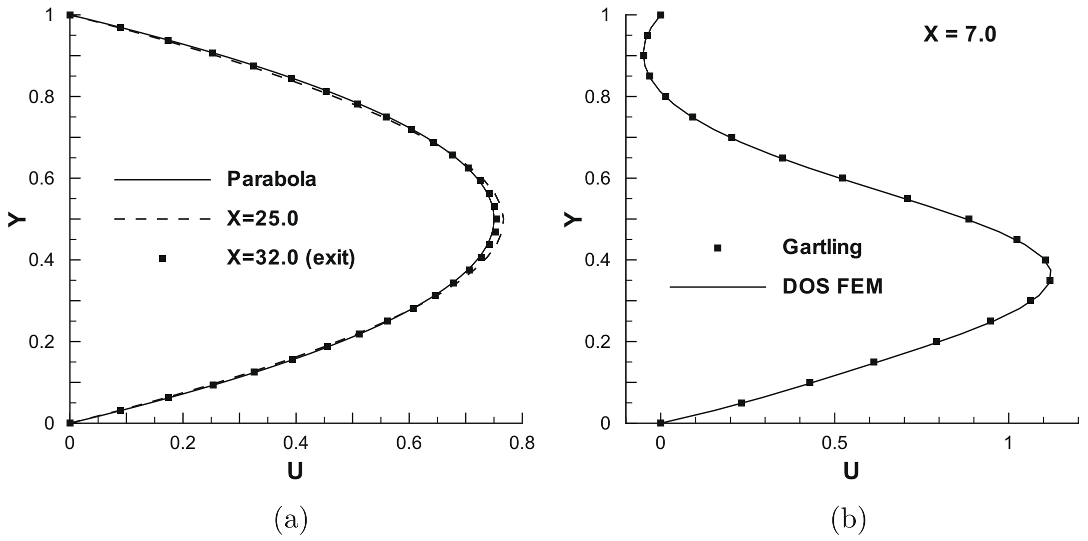


Fig. 6. Verifications of streamwise velocity profiles for the backstep flow at $Re = 800$ at two locations: (a) at the exit, (b) at $x = 7.0$. Both results are based on a 256×16 mesh of 8–4 (quadratic-linear) elements. Gartling's reference result can be found in [11].

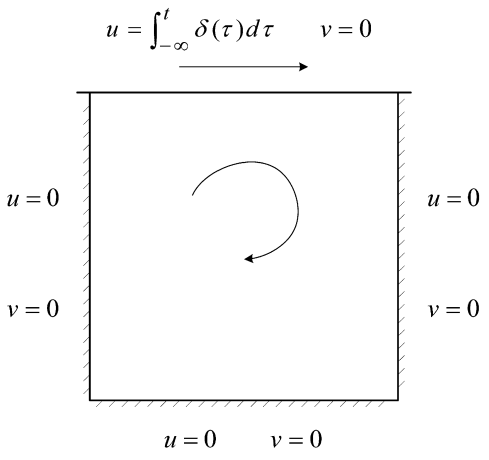


Fig. 7. Configuration of the cavity flow. The pressure is free from boundary conditions.

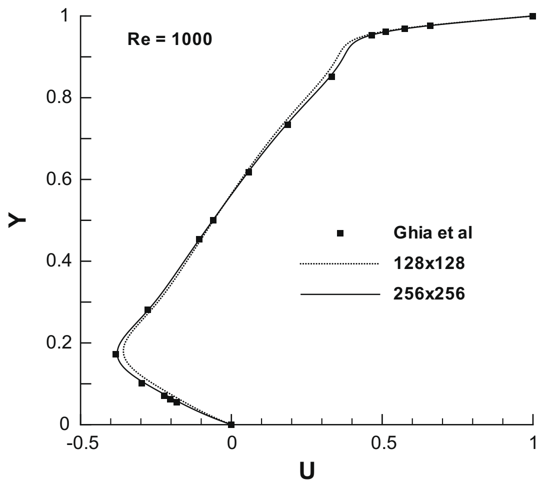


Fig. 8. Comparison between the present numerical result and the standard reference result [12] for the horizontal velocity profile on the vertical mid plane of the lid-driven cavity flow.

Good agreement is observed. A mixed formulation, where velocity and pressure are solved simultaneously, was attempted to solve the cavity flow. When the mesh is really fine, even with 8–1 element and with very low *Re*, the numerical results start to distort. It is commonly believed by many that the finer meshes produce the better results. However, this is true only for a well-conditioned system. As the size of the system increases, the conditioning number, the ratio of the largest eigenvalue and the smallest eigenvalue, continues to grow and the nature of the system continues to deteriorate. Therefore, the splitting technique such as the DOS proposed in this paper is crucial to efficiently and reliably solve incompressible flows.

6.3. Axisymmetric pipe flow

The configuration for the pipe flow is shown in Fig. 9, where the radius of the pipe flow is set as unity. The pipe flow possesses several features ideal for numerical demonstration. Both the geometry and the flow are relatively simple and the exit velocity profile is known exactly, so that it is suitable for numerical comparison. The uniform flow at the entrance develops into a parallel flow at the exit, hence the flow is axisymmetric three-dimensional and ideal for testing the generic formulation. Further, the pipe flow involves non-essential boundary conditions, along the symmetric axis and at the open exit, and the open boundary conditions at the exit. Various boundary conditions can be used for a specific numerical method. As a standard practice, traction boundary conditions are derived from velocity boundary conditions for the present DOS FEM. However, velocity boundary conditions at the exit, as commonly adopted in finite volume and finite difference methods, are also implemented with DOS FEM. This is fulfilled by avoiding integration by part for the diffusion term. Unfortunately this approach does not show any advantage over the standard one, except for some cases with the non-free traction exit boundary condition. For example, if the flow is exerted by a conservative body force which will be absorbed into the pressure numerically. In this case, the traction at the exit no longer vanishes. In contrast, the velocity boundary conditions remain the same. Finally, once the benchmark results based on an axisymmetric formulation are validated, the pipe flow can be used to test a three-dimensional code by solving the same flow. All these above reasons make the pipe flow a useful benchmark problem.

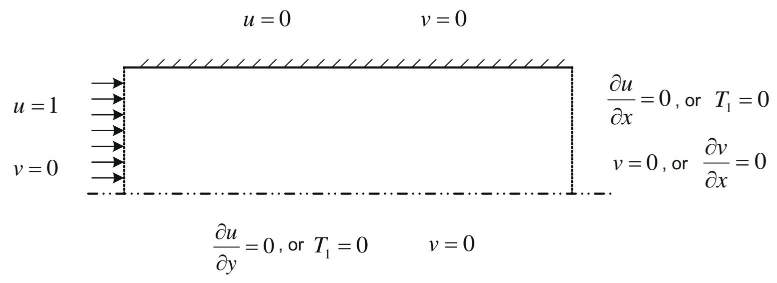

$u=0$
$v=0$
$u=1$
$v=0$
$\frac{\partial u}{\partial x}=0$, or $T_1=0$
$v=0$, or $\frac{\partial v}{\partial x}=0$
$\frac{\partial u}{\partial y}=0$, or $T_1=0$
$v=0$

Fig. 9. Configuration for the axisymmetric pipe flow. The pressure is free from boundary conditions, or vanished as implied in free traction boundary condition (if used) at exit.

Fig. 10(a) shows the comparison of exit (set at $x = 32.0$) velocity profile between the numerical result and exact solution. The numerical result is based on a mesh with 8 quadratic elements in y direction and 128 quadratic elements in x direction. We have to emphasize, the results based on our parallel code with 8 processors under a Linux operating system with MPI C++ compiler produces digit-by-digit identically the same numbers as those based on the counterpart serial code under Windows operating system with Microsoft Visual C++ compiler.

Fig. 10(b) shows comparisons of two exit velocity profiles against the fully developed parabolic distribution for $Re = 800$. In one case the computational domain is truncated at $x = 128.0$, which is twice as long as that in the backstep flow. However, the figure shows that the corresponding result is still very discrepant

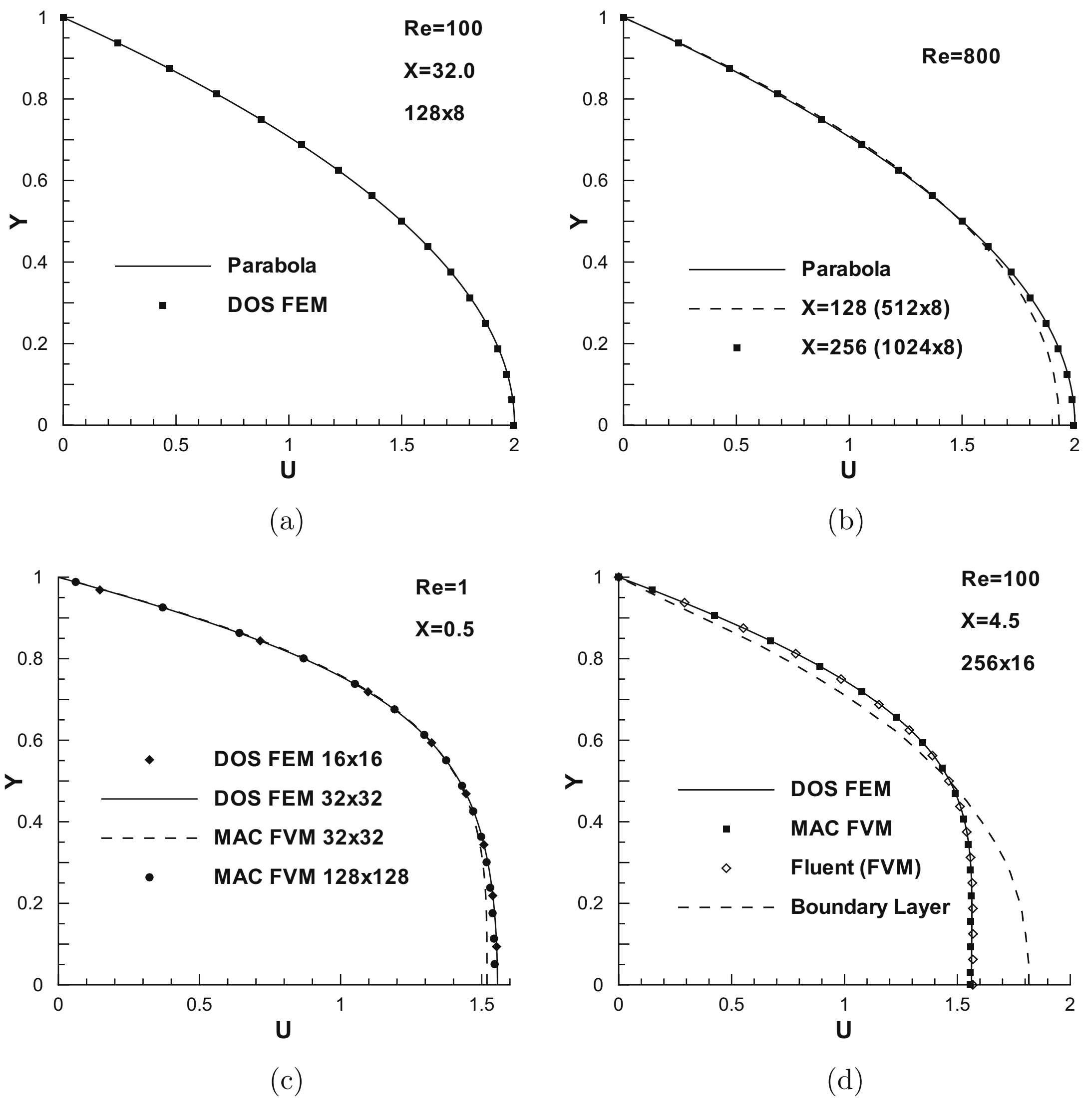


Fig. 10. Comparisons of streamwise velocity profiles for the axisymmetric pipe flow. (a) Exit ($x = 32.0$) profile for $Re = 100$. (b) Exit profiles for $Re = 800$ based on two different truncation lengths, $x = 128.0$ and $x = 256.0$. (c) Comparison for $Re = 1$ at $x = 0.5$ with truncation length of $x = 2.0$. (d) Comparison for $Re = 100$ at $x = 4.5$ with truncation length of $x = 32.0$.

from the exact profile. In the other case, the flow is truncated at $x = 256.0$, and with the same mesh resolution we observe an excellent agreement between the present numerical result and the exact solution.

Next, the numerical results generated by the parallel DOS finite element method are compared with those generated by the MAC FVM code [36], where the MAC staggered grid is employed [18]. Fig. 10(c) shows the results for a pipe flow at $Re = 1.0$ truncated at $x = 2.0$. While the mesh resolution is resolved easily for quadratic finite element method, it takes much more effort for the finite volume method on MAC staggered grid to achieve the same convergence. This is because in second-order finite volume method, all domain and boundary integrals are evaluated at centers. This is equivalent to one-point quadrature, and it is efficient and accurate in general. However, its performance on axisymmetric and diffusive problems is poor.

Fig. 10(d) shows the comparison of several results for a pipe flow at $Re = 100$ and at $x = 4.5$. In this case, the DOS FEM, the MAC FVM, and the commercial software Fluent (a finite volume method on non-staggered grid) agree with each other very well. However, the numerical result based on the boundary layer equations [21] shows an obvious inaccuracy. We have to point out, all these four methods yield excellent agreement with the exit parabola profile. That indicates that a mere examination at the exit is not sufficient for computer code testing.

6.4. Scale-up study

The 2D cavity flow is used to measure the efficiency of the parallel algorithm. Table 1 shows the comparison of the cpu time among cases with different number of processors. The experiment in the table is based on cavity flow at $Re = 1000$ on a 256×256 mesh with a time marching process terminated at the convergence (defined as relative change of solution less than 10^{-7}). This mesh is fine enough and the time is long enough to exclude the communication overhead. The experiment shows that an increase from 2 processors to 8 processors speeds up the computations by 3.68 times. An ideal parallel algorithm speeds up nearly linearly for large systems, which implies in the same situation it is expected to be 4 times faster. Hence the current algorithm achieves 92% efficiency.

6.5. DOS versus mixed formulation

To further investigate the performance of the DOS technique, a mixed formulation is also implemented for comparison. The construction of the equation system in the mixed formulation is exactly the same as that in the DOS approach. Consequently, both approaches lead to a linear algebraic equation system in the form

$$\widetilde{A}x = \tilde{b}, \tag{13}$$

where the structure of matrix $\widetilde{A}$ is shown in Eqs. (7a) and (7b). The indefinite nature of matrix $\widetilde{A}$ makes Eq. (13) very difficult to solve. For example, the Gauss–Seidel method fails to converge for Eq. (13) and Krylov-based BiCGStab [10] frequently fails for Eq. (13). This observation alone is sufficient to demonstrate the success of DOS, which works for all these linear solvers. To make the mixed formulation work, one may use artificial compressibility or pressure stabilization to modify matrix $\widetilde{A}$, then an efficiency comparison versus DOS is possible. Here we take a different approach. Let $\widetilde{A}^T$ be the transpose of the non-singular matrix $\widetilde{A}$, then Eq. (13) is equivalent to

$$\widetilde{A}^T\widetilde{A}x = \widetilde{A}^T\tilde{b}. \tag{14}$$

The detailed structure of matrix $\widetilde{A}^T\widetilde{A}$ is

$$\begin{bmatrix} A^TA + D^TD & A^TG \\ G^TA & G^TG \end{bmatrix},$$

which is symmetric and no longer indefinite. Now, Eq. (14) can be solved with the well-known Conjugate Gradient method. The cavity flow at $Re = 100$ on a 16×16 mesh is solved with a time marching procedure ($\Delta t = 0.002$). For mixed formulation the consumed computational time is 15,986 s while for DOS it is only 900 s. This shows that the DOS procedure is approximately 17 times as efficient as the mixed approach.

Table 1
Scale up study of the parallel finite element implementation.

Number of processors	1	2	4	8
CPU time (h)	78.3	51.1	26.7	13.9

7. Conclusions

In this paper, the weak form of integral equations for incompressible flows under the generic coordinates is rigorously derived and some element-level implementation is discussed. To parallelize the computer code, the system is formed in an element-by-element manner, which incorporates boundary conditions at element stage. To solve the system efficiently a processor level assembling is carried out, and the data of the system is locally stored with the condensed random structure. The whole system is solved globally in terms of the overall structure. However, the actual time consuming operations are executed locally and the resulting data are communicated among processors. This is the key idea of the superposition-based parallelization. Superposition-based parallelization is simple and fairly efficient.

To replace the original large indefinite system by two well natured subsystems, the DOS (discrete operator splitting) technique is introduced. On one hand, a system with mutual dependency of the unknowns (in the continuous sense) on the left-hand-side is coupled, while a system with mutual dependency of the unknowns on the right-hand-side is decoupled. On the other hand, the iterative process of system solving is a process of moving a portion of quantities to the right-hand-side. Hence, the decoupling process and the iterative process have the same goal and can be combined together. The key move to tackle both issues is surprisingly simple: split the matrix into the diagonal part and the off diagonal part. It has to be emphasize that the transformed equation system must be equivalent to the original system.

The DOS and the parallel scheme are implemented with a finite element method. The verification, effectiveness, and accuracy of the method are demonstrated by three numerical examples, a backstep flow, a cavity flow, and a pipe flow. Applications to transient and three-dimensional problems will be shown in a future paper. In a continuous sense, an incompressible flow is a special case of a compressible flow. However, in the discrete sense the compressible flow can be regarded as a special case of the incompressible flow, because after splitting, the resulting definite system is in the same format as in a compressible flow. While solving compressible and incompressible flows in a unified approach is desirable, currently no successful method, to the authors' knowledge, are seen in published literature. The DOS might spark some ideas toward this direction.

Acknowledgement

This work was in part supported through Grant N00014-08-1-0612 from the US Office of Naval Research.

References

[1] J.B. Bell, P. Colella, H.M. Glaz, A second-order projection method for the incompressible Navier–Stokes equations, Journal of Computational Physics 85 (1989) 257–283.
[2] D.L. Brown, R. Cortez, M.L. Minion, Accurate projection methods for the incompressible Navier–Stokes equations, Journal of Computational Physics 168 (2001) 464–499.
[3] A.J. Chandy, D.J. Glaze, S.H. Frankel, Parallelizing the discrete ordinates method (DOM) for three-dimensional radiative heat transfer calculations using a priority queuing technique, Numerical Heat Transfer Part B-Fundamentals 52 (2007) 33–49.
[4] A.J. Chorin, Numerical solution of the Navier–Stokes equations, Mathematics of Computation 22 (1968) 745–762.
[5] A.J. Chorin, On the convergence of discrete approximations to the Navier–Stokes equations, Mathematics of Computation 23 (1969) 341–353.
[6] M. Deville, L. Kleiser, F. Montigny-Rannou, Pressure and time treatment for chebyshev spectral solution of a stokes problem, International Journal for Numerical Methods in Fluids 4 (1984) 1149–1163.
[7] M.O. Deville, P.F. Fischer, E.H. Mund, High-Order Methods for Incompressible Fluid Flow, Cambridge University Press, 2002.
[8] J.K. Dukowicz, A.S. Dvinsky, Approximate factorization as a high order splitting for the implicit incompressible flow equations, Journal of Computational Physics 102 (1992) 336–347.
[9] P.F. Fischer, Spectral Element Solution of Navier–Stokes Equations on High Performance Distributed-Memory Parallel Processors. PhD thesis, Massachusetts Institute of Technology, 1989.
[10] S. Fujino, GPBiCG(m,l): a hybrid of BiCGStab and GPBiCG methods with efficiency and robustness, Applied Numerical Mathematics 41 (2002) 107–117.
[11] D.K. Gartling, A test problem for outflow boundary conditions – flow over a backward-facing step, International Journal for Numerical Methods in Fluids 11 (1990) 953–967.
[12] U. Ghia, K.N. Ghia, C.T. Shin, High-*Re* solutions for incompressible flow using the Navier–Stokes equations and a multigrid method, Journal of Computational Physics 48 (1982) 387–411.
[13] P.M. Gresho, On the theory of semi-implicit projection methods for viscous incompressible flow and its implementation via a finite element method that also introduces a nearly consistent mass matrix part 1: theory, International Journal for Numerical Methods in Fluids 11 (1990) 587–620.
[14] P.M. Gresho, Incompressible fluid dynamics: some fundamental formulation issues, Annual Review of Fluid Mechanics 23 (1991) 413–453.
[15] P.M. Gresho, R.L. Sani, On pressure boundary conditions for the incompressible Navier–Stokes equations, International Journal for Numerical Methods in Fluids 7 (1987) 1111–1145.
[16] W. Gropp, E. Lusk, A. Skjellum, Using MPI, 2nd ed., The MIT Press, Berlin, 1999.
[17] J.L. Guermond, J. Shen, A new class of truely consistent splitting schemes for incompressible flows, Journal of Computational Physics 192 (2003) 262–276.
[18] F.H. Harlow, J.F. Welch, Numerical calculation of time-dependent viscous incompressible flow of fluid with free surface, Physics of Fluids 8 (1965) 2182–2189.
[19] V. Haroutunian, M.S. Engelman, I. Hasbani, Segregated finite element algorithms for the numerical solution of large-scale incompressible flow problems, International Journal for Numerical Methods in Fluids 17 (1993) 323–348.
[20] G.E.M. Karniadakis, M. Israeli, S.A. Orszag, High-order splitting methods for the incompressible Navier–Stokes equations, Journal of Computational Physics 97 (1991) 414–443.
[21] W.M. Kays, M.E. Crawford, B. Weigand, Convective Heat and Mass Transfer, 4th ed., McGraw-Hill Professional, 2004. Chapter 7.
[22] J. Kim, P. Moin, Application of a fractional-step method to incompressible Navier–Stokes equations, Journal of Computational Physics 59 (1985) 308–323.
[23] G. Krishnamoorthy, R. Rawat, P.J. Smith, Parallel computations of nongary radiative heat transfer, Numerical Heat Transfer Part B-Fundamentals 48 (2005) 191–211.
[24] G. Krishnamoorthy, R. Rawat, P.J. Smith, Parallel computations of radiative heat transfer using discrete ordinates method, Numerical Heat Transfer Part B-Fundamentals 46 (2005) 19–38.
[25] G. Krishnamoorthy, R. Rawat, P.J. Smith, Parallelization of the p-1 radiation model, Numerical Heat Transfer Part B-Fundamentals 49 (2006) 1–17.
[26] J. Liu, Y.S. Chen, Simulation of rapid thermal processing in a distributed computing environment, Numerical Heat Transfer Part A-Applications 38 (2000) 129–152.
[27] P. Moin, J. Kim, On the numerical solution of time-dependent viscous incompressible fluid flows involving solid boundaries, Journal of Computational Physics 35 (1980) 381–392.
[28] P.J. Novo, P.J. Coelho, M.G. Carvalho, Parallelization of the discrete transfer method, Numerical Heat Transfer Part B-Fundamentals 35 (1999) 137–161.
[29] J.B. Perot, An analysis of the fractional step method, Journal of Computational Physics 108 (1993) 51–58.
[30] R. Peyret, D. Taylor, Computational Methods for Fluid Flow, Springer-Verlag, Berlin, 1983.
[31] A. Quarteroni, F. Saleri, A. Veneziani, Factorization methods for the numerical approximation of Navier–Stokes equations, Computer Methods in Applied Mechanics and Engineering 188 (2000) 505–526.
[32] J.C. Strikwerda, Y.S. Lee, The accuracy of the fractional step method, SIAM Journal on Numerical Analysis 37 (1999) 37–47.
[33] J. van Kan, A second-order accurate pressure-correction scheme for viscous incompressible flow, SIAM Journal on Scientific and Statistical Computing 7 (1986) 870–891.
[34] Z.H. Yan, A numerical study of effect of initial condition on large eddy simulation of thermal plume, Numerical Heat Transfer Part B-Fundamentals 43 (2003) 167–178.
[35] O. Yildiz, H. Bedir, A parallel solution to the radiative transport in three-dimensional participating media, Numerical Heat Transfer Part B-Fundamentals 50 (2006) 79–95.
[36] K.K.Q. Zhang, W.J. Minkowycz, F. Mashayek, Exact factorization technique for numerical simulations of incompressible Navier–Stokes flows, International Journal of Heat and Mass Transfer 49 (2006) 535–545.
[37] K.K.Q. Zhang, B. Rovagnati, Z. Gao, W.J. Minkowycz, F. Mashayek, An introduction to lattice grid, Numerical Heat Transfer Part B-Fundamentals 51 (2007) 415–431.
[38] K.K.Q. Zhang, B. Shotorban, W.J. Minkowycz, F. Mashayek, A compact finite difference method on staggered grid for Navier–Stokes flows, International Journal for Numerical Methods in Fluids 52 (2006) 867–881.