Available online at www.sciencedirect.com

SciVerse ScienceDirect

Procedia Computer Science 18 (2013) 439 – 448

Procedia Computer Science

International Conference on Computational Science, ICCS 2013

# A parallel solver for incompressible fluid flows

Yushan Wang[a,*], Marc Baboulin[a,b], Jack Dongarra[c], Joël Falcou[a], Yann Fraigneau[a,d], Olivier Le Maître[a,d]

[a]*Université Paris-Sud, 91400, Orsay, France*
[b]*INRIA, 1 rue Honoré d'Estienne d'Orves, 91120, Palaiseau, France*
[c]*University of Tennessee, 37996, Knoxville, USA*
[d]*LIMSI, Rue John Von Neumann, 91400, Orsay, France*

**Abstract**

The Navier-Stokes equations describe a large class of fluid flows but are difficult to solve analytically because of their nonlinearity. We present in this paper a parallel solver for the 3-D Navier-Stokes equations of incompressible unsteady flows with constant coefficients, discretized by the finite difference method. We apply the prediction-projection method which transforms the Navier-Stokes equations into three Helmholtz equations and one Poisson equation. For each Helmholtz system, we apply the Alternating Direction Implicit (ADI) method resulting in three tridiagonal systems. The Poisson equation is solved using partial diagonalization which transforms the Laplacian operator into a tridiagonal one. We describe an implementation based on MPI where the computations are performed on each subdomain and information is exchanged on the interfaces, and where the tridiagonal system solutions are accelerated using vectorization techniques. We present performance results on a current multicore system.

*Keywords:* Navier-Stokes equations; prediction-projection; ADI; partial diagonalization; SIMD; parallel computing.

## 1. Introduction

The incompressible Navier-Stokes (NS) equations express the Newton's second law applied to fluid motion, with the assumptions that the fluid density is constant and the fluid stress is the sum of a viscous term (proportional to the gradient of the velocity) and an isotropic pressure term. These equations are widely used to model fluid flows in numerous scientific and industrial applications (e.g. in aeronautics, aerospace, meteorology, thermo-hydraulics, and climatology). Efficient numerical methods for the NS equations have then received considerable attention over the last decades, as their simulation remains a challenging problem especially for flows exhibiting complex transient, turbulent dynamics and features at many different scales. Alternative numerical methods for the NS equations include *finite difference* (FD) [1, 2, 3, 4], *finite element* [5, 6], *finite volume* [1, 2], *spectral methods* [7, 8] and mesh-free methods such as the *vortex method* [9] or *smoothed particle hydrodynamics* [10, 11, 12]. These methods have pros and cons, depending on the problem considered. In addition, the evolution of computer architectures and the emergence of massively parallel computers call for the development of new algorithms to fully exploit hardware advances.

*Corresponding author. Tel.: +33-(0)174854212 .
*E-mail address:* yushan.wang@u-psud.fr.

doi:10.1016/j.procs.2013.05.207

In this paper, we focus on FD discretizations (of second order in time and space) of the three-dimensional NS equations which are solved using the classical prediction-projection method [13, 14, 15, 16]. This method relies on an explicit treatment of the non-linear terms and an implicit treatment of the viscous and pressure terms to ensure stability. As detailed in Section 2, it leads to the solution of 3-D diffusion (velocity) and Poisson (pressure) equations that constitute the main computational workload of the simulation. Further, we restrict ourself to spatial discretizations on structured cartesian grids resulting from the tensorization of one-dimensional grid. Although not general, the structured character of these grids greatly facilitates the design and efficient implementation of the algorithms on modern computer architectures, and encompasses many situations of interest. Even for these simple meshes, 3-D parallel solvers are needed since complex flow simulations and targeted applications commonly require grid resolution corresponding to $10^7 - 10^9$ unknowns to properly capture the flow dynamics. These large simulations require the use of parallel computers. Another important advantage brought by the prediction-projection approach on tensored grids is that it leads to Helmholtz-like or Poisson equations that can be consistently reduced to one-dimensional operators using respectively operator splitting and partial diagonalization [17]. The second order discretization results in tridiagonal systems that can solved efficiently by taking advantage of current parallel architectures. The parallel strategy is based on the idea of dividing the computational work between several processors which perform their calculations concurrently. We use a domain decomposition method in which the computational domain is divided into subdomains, each associated to one processor. The Poisson and Helmholtz-like equations are then solved over each subdomain in parallel using the Schur complement method [18] to ensure the continuity of the solution at the subdomains interfaces. This involves data communication between neighboring subdomains for the interface solution. Communications are handled by the *Message-Passing-Interface* (MPI) [19]. We also rely on the BLAS [20], LAPACK [21] and ScaLAPACK [22] kernels from the MKL [23] library to perform the computations. In addition to the combination of the numerical methods mentioned above, an important contribution of this work is the implementation of a vectorized version of Thomas algorithm for a fast solution of the tridiagonal systems.

This paper is organized as follows. Section 2 summarizes the solution of the Navier-Stokes equations for incompressible fluid flows and the prediction-projection method. Then, the FD discretization in space and time is introduced together with the directional splitting and partial diagonalization to obtain the tridiagonal systems to be solved. Section 3 discusses the parallel implementation and emphasizes on vectorization techniques to improve the Thomas algorithm for the solution of the tridiagonal systems. In Section 4, several performance measurements of our Navier-Stokes solver are provided using a current parallel system. Concluding remarks are presented in Section 5.

## 2. Navier-Stokes Problem

We consider the unsteady flow of a incompressible Newtonian fluid governed by the Navier–Stokes (NS) equations that can be expressed as

$$\frac{\partial \boldsymbol{V}}{\partial t} + \boldsymbol{\nabla} \cdot (\boldsymbol{V}^T \otimes \boldsymbol{V}) = -\boldsymbol{\nabla} P + \frac{1}{\mathrm{Re}} \Delta \boldsymbol{V}, \tag{1}$$

$$\boldsymbol{\nabla} \cdot \boldsymbol{V} = 0, \tag{2}$$

where $\boldsymbol{V} = (u(x, y, z, t), v(x, y, z, t), w(x, y, z, t))^T$ is the velocity vector and $P = P(x, y, z, t)$ is the pressure. Eqs. (1) and (2) state the conservation of momentum and mass respectively. We have denoted $\boldsymbol{\nabla}$ the nabla and $\Delta$ the Laplacian operators, while $\otimes$ is the Kronecker product [24]. In the following, we use the shorthand notation $\mathcal{CT} := \boldsymbol{\nabla} \cdot (\boldsymbol{V}^T \otimes \boldsymbol{V})$ for the convective term. In Eq. (1), $\mathrm{Re} := \rho U L/\mu$ is the Reynolds number based on a reference length $L$, a reference velocity $U$, the fluid density $\rho$, and the dynamic viscosity $\mu$. The Reynolds number is a good indicator of the fluid state (laminar, transient, turbulent). We see from Eq. (1) that the larger Re is, the more dominant the convection is, with respect to viscous effects. This leads to the onset of flow instabilities and a turbulent state of the flow for high Reynolds numbers. In turbulent flows, the order of magnitude for the ratio between the largest and the smallest length scale of eddy structures is $\mathrm{Re}^{\frac{3}{4}}$. So, a finer discretization is required when the Reynolds number increases and the resulting order of magnitude for the 3D mesh size will be $\mathrm{Re}^{\frac{9}{4}}$. Solving the NS equations also involves defining the initial and boundary conditions. The details about the implementations of the boundary condition will not be discussed in this paper. We mention that our code can deal with mixed Dirichlet and Neumann (homogeneous or not), and periodic/symmetric boundary conditions. The initial conditions for velocity and pressure will depend on the physical problem.

## 2.1. Prediction-Projection method

In this section, we present the prediction-projection method applied on the NS equations. The spatial discretization uses second-order finite differences approximations of the differential operator. The domain is discretized using $\mathrm{N_x} \times \mathrm{N_y} \times \mathrm{N_z}$ cells, where $\mathrm{N_i}$ the number of mesh along the $i$–directions ($i = x, y, z$). The discrete velocity and pressure unknowns are defined along a staggered arrangement, in which the pressure is defined on the center of each cell, the velocity components are defined on the center of the corresponding edge, see Fig. 1, to prevent spurious oscillations [3].

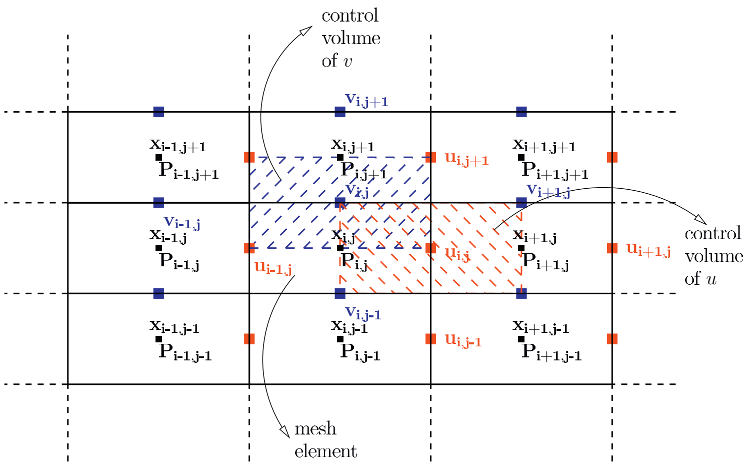


Fig. 1. Staggered mesh structure.

We denote the time level with upper-indices and $\Delta t$ the time-step size. We rely on a second order approximation of the time derivative (BDF2) [25, 26], an implicit treatment of the viscous and pressure term, and a second order explicit extrapolation of the convective term. Then the discrete NS equations can be expressed as

$$\frac{3\boldsymbol{V}^{n+1} - 4\boldsymbol{V}^{n} + \boldsymbol{V}^{n-1}}{2\Delta t} + \widetilde{\mathcal{CT}}^{n+1} = -\boldsymbol{\nabla} P^{n+1} + \frac{1}{\mathrm{Re}}\Delta \boldsymbol{V}^{n+1}, \tag{3}$$

$$\boldsymbol{\nabla} \cdot \boldsymbol{V}^{n+1} = 0, \tag{4}$$

where $\widetilde{\mathcal{CT}}^{n+1} = 2\mathcal{CT}^{n} - \mathcal{CT}^{n-1}$.

The integration for one time-step consists of two steps: a prediction followed by a projection. The latter step is also called the correction step.

## 2.2. Prediction step

In the prediction step, an approximate solution for the momentum equation is sought without accounting for the incompressibility condition in Eq. (4). More specifically, we compute an auxiliary velocity $\boldsymbol{V}^{*}$

$$\frac{3\boldsymbol{V}^{*} - 4\boldsymbol{V}^{n} + \boldsymbol{V}^{n-1}}{2\Delta t} + \widetilde{\mathcal{CT}}^{n+1} = -\boldsymbol{\nabla} P^{n} + \frac{1}{\mathrm{Re}}\Delta \boldsymbol{V}^{*}. \tag{5}$$

Note that in Eq. (5), the pressure $P^{n+1}$ has been substituted by $P^{n}$. Defining $\sigma := 2\Delta t/3$, the Eq. (5) for $\boldsymbol{V}^{*}$ can be recast as

$$\left(I - \frac{\sigma}{\mathrm{Re}}\Delta\right)\boldsymbol{V}^{*} = S, \quad S := \frac{4\boldsymbol{V}^{n} - \boldsymbol{V}^{n-1}}{3} - \sigma(\widetilde{\mathcal{CT}}^{n+1} + \boldsymbol{\nabla} P^{n}). \tag{6}$$

To solve Eq. (6), we apply the Alternating Direction Implicit (ADI) method which approximates the 3-D operator $(I - \frac{\sigma}{\mathrm{Re}}\Delta)$ as a product of three 1-D operators:

$$\left(I - \frac{\sigma}{\mathrm{Re}}\Delta\right) = \left(I - \frac{\sigma}{\mathrm{Re}}\Delta_x\right)\left(I - \frac{\sigma}{\mathrm{Re}}\Delta_y\right)\left(I - \frac{\sigma}{\mathrm{Re}}\Delta_z\right) + O(\Delta t^2), \tag{7}$$

where $\Delta_i$ is the second order derivative along the $i$–direction. With this product, we can transform the 3-D Helmholtz-like equation (Eq. (6)) into a system of three 1-D Helmholtz-like equations

$$\left(I - \frac{\sigma}{\mathrm{Re}}\Delta_x\right)T_1 = S, \quad \left(I - \frac{\sigma}{\mathrm{Re}}\Delta_y\right)T_2 = T_1, \quad \left(I - \frac{\sigma}{\mathrm{Re}}\Delta_z\right)\boldsymbol{V}^* = T_2. \tag{8}$$

In the remainder of this paper, Eq. (8) will be simply referred to as Helmholtz equation.

## 2.3. Projection step

In general, $\boldsymbol{V}^*$ is not divergence-free so it is corrected in the second step. The correction is sought as a potential velocity field as follows. Taking the divergence of the difference of Eqs. (3) and (5), it comes:

$$\boldsymbol{\nabla}\cdot(\boldsymbol{V}^{n+1} - \boldsymbol{V}^*) = -\sigma\Delta(P^{n+1} - P^n) + \frac{\sigma}{\mathrm{Re}}\boldsymbol{\nabla}\cdot\Delta(\boldsymbol{V}^{n+1} - \boldsymbol{V}^*). \tag{9}$$

Since $\boldsymbol{\nabla}\cdot\boldsymbol{V}^{n+1} = 0$ and since, for any vector $\boldsymbol{U}$, we have $\boldsymbol{\nabla}\cdot\Delta\boldsymbol{U} = \Delta\boldsymbol{\nabla}\cdot\boldsymbol{U}$, then Eq. (9) becomes

$$\boldsymbol{\nabla}\cdot\boldsymbol{V}^* = \sigma\Delta\left(P^{n+1} - P^n - \frac{1}{\mathrm{Re}}\boldsymbol{\nabla}\cdot\boldsymbol{V}^*\right). \tag{10}$$

Denoting $\phi := P^{n+1} - P^n + \frac{1}{\mathrm{Re}}\nabla\cdot\boldsymbol{V}^*$, then $\phi$ satisfies the Poisson equation

$$\Delta\phi = \frac{1}{\sigma}\boldsymbol{\nabla}\cdot\boldsymbol{V}^*, \tag{11}$$

which is complemented with appropriate boundary conditions. After computing $\phi$, we update the velocity and pressure fields using

$$\boldsymbol{V}^{n+1} = \boldsymbol{V}^* - \sigma\phi, \quad P^{n+1} = P^n + \phi - \frac{1}{\mathrm{Re}}\boldsymbol{\nabla}\cdot\boldsymbol{V}^*. \tag{12}$$

There exist several methods to solve the Poisson equation given in Eq. (11). For example, the fast Fourier transformation (FFT), fast sine/cosine transformations, and multigrid methods [27]. In this paper, we consider a direct method based on partial diagonalization.

Let us denote $\Delta\phi = \frac{1}{\sigma}\boldsymbol{\nabla}\cdot\boldsymbol{V}^* = S$, and rewrite $\Delta = \Delta_x + \Delta_y + \Delta_z$. Then the second order central schemes for these 1-D operators $\Delta_x$, $\Delta_y$ and $\Delta_z$ give three tridiagonal matrices for the considered direction. To further simplify the computation, we transform tridiagonal matrices into diagonal ones by using their eigendecomposition [18] .

Given a real square $(N \times N)$ matrix $A$ with $N$ linearly independent eigenvectors $q_i$ $(i = 1, ..., N)$, then $A$ can be factorized as $A = Q\Lambda Q^{-1}$ where $\Lambda$ is the diagonal matrix whose diagonal elements are the corresponding eigenvalues, and $Q$ is the square $(N \times N)$ matrix whose $i$th column is the eigenvector $q_i$ of $A$.

For our problem, we decompose only $\Delta_x$, $\Delta_y$ and keep $\Delta_z$, then we have

$$(Q_x\Lambda_x Q_x^{-1} + Q_y\Lambda_y Q_y^{-1} + \Delta_z)\phi = S. \tag{13}$$

Then we multiply Eq. (13) with $Q_x^{-1}Q_y^{-1}$, and observe that

$$Q_x^{-1}Q_y^{-1} = Q_y^{-1}Q_x^{-1}, \quad Q_x^{-1}\Lambda_y^{-1} = \Lambda_y^{-1}Q_x^{-1}, \quad Q_y^{-1}\Lambda_x^{-1} = \Lambda_x^{-1}Q_y^{-1}, \tag{14}$$

so we have

$$(\Lambda_x + \Lambda_y + \Delta_z)\tilde{\phi} = \tilde{S}, \text{ where } \tilde{\phi} = Q_y^{-1}Q_x^{-1}\phi \text{ and } \tilde{S} = Q_y^{-1}Q_x^{-1}S. \tag{15}$$

So the original Poisson equation (Eq. 11) is now transformed into the form of Eq. (15) which is a 1-D tridiagonal equation for the $z$-direction.

## 3. Parallel implementation for solving Navier-Stokes equations

### 3.1. Domain decomposition and Schur complement method

In our parallelization strategy we divide the domain equally into subdomains and we assign each subdomain to one MPI process in such a way that each process has the same amount of computational workload. Once the subdomains and the boundary conditions have been defined, the computations on each subdomain can be performed in parallel. The boundaries here include the real domain boundaries and the interfaces between the subdomains. The classical Schur complement method is used to ensure continuity of the solution at the interface between subdomains, and the information at the interfaces is exchanged via MPI routines.

Fig. 2 shows an example of a 2-D domain decomposition with boundary cells. For instance in subdomain IV, the top and right boundaries are real domain boundaries while the left and lower boundaries are actually interfaces with neighboring subdomains.

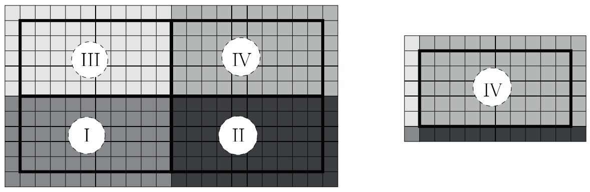


Fig. 2. Domain decomposition and boundary cells.

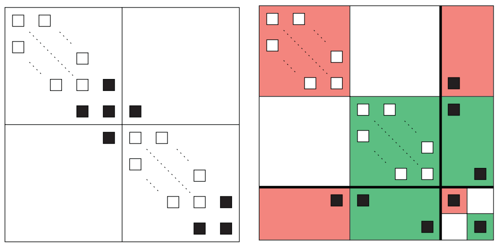


Fig. 3. Illustration of variable reordering for Schur Complement method for the case of 2 subdomains. □: interior variables. ■: interface variables. (a) original tridiagonal matrix; (b) matrix after reordering.

We suppose now that we have obtained the tridiagonal system for one direction (corresponding for instance to the x-direction, as depicted in Fig. 2. The matrix structure is shown by Fig. 3.(a). This tridiagonal system is solved by the Schur complement method. By reordering the variables so that the interface variables are listed last, we obtain a new system where the matrix has a structure as depicted in Fig. 3.(b).

We recall below how the solution is computed via the Schur complement method. Given a block system

$$\begin{pmatrix} A_k & A_{ki} \\ A_{ik} & A_i \end{pmatrix} \begin{pmatrix} X_k \\ X_i \end{pmatrix} = \begin{pmatrix} S_k \\ S_i \end{pmatrix}, \tag{16}$$

where indices $k$ and $i$ refer respectively to interior and interface variables.

From the first block equation in Eq. (16) the unknown vector $X_k$ can be expressed as

$$X_k = A_k^{-1}(S_k - A_{ki}X_i), \tag{17}$$

and we substitute this into the second equation to obtain

$$(A_i - A_{ik}A_k^{-1}A_{ki})X_i = S_i - A_{ik}A_k^{-1}S_k. \tag{18}$$

If the Schur complement matrix $A_i - A_{ik}A_k^{-1}A_{ki}$ can be formed and the linear system (18) can be solved, all the interface variables $X_i$ are then available and the remaining unknowns can be computed via Eq. (17). Due to the particular block structure of $A_k$ and $A_i$, we have $r$ separate systems that can be solved in parallel, where $r$ is the number of subdomains along the direction in which we solve the original tridiagonal system.

### 3.2. Solving tridiagonal systems

We describe in this section how the block-tridiagonal systems resulting from the discretizations of Eq. (8) (using ADI method) and Eq. (11) (using partial diagonalization) can be solved efficiently. There have been several efforts in recent years to develop efficient tridiagonal solvers [28, 29, 30]. A well-known procedure is based on the Thomas algorithm [31] which basically corresponds to a Gaussian elimination without pivoting.

Given a tridiagonal system $Ax = s$, where $A \in \mathbb{R}^{n\times n}$ is diagonally dominant (nonsingular), and $x, s \in \mathbb{R}^n$, the Thomas algorithm consists of two steps. The first step is a forward elimination where we eliminate the lower diagonal coefficients and transform the original matrix into an upper triangular one. The second step is a backward substitution that solves the upper triangular system. This requires $O(n)$ operations.

To better understand the structure of the linear system, let us take a simple 2-D example as depicted in Fig. 4 which uses a $4 \times 3$ 2D mesh and let us consider the operator $\Delta_x$ in Eq. (8). We order the unknowns first by row and then by column as shown in Fig. 4. Then the matrix corresponding to the operator $\Delta_x$ can be represented as the matrix given in Fig. 5 (with $m = 3$ tridiagonal blocks). Similarly, if we consider the operator $\Delta_y$ in Eq. (8), the unknowns will be ordered first by column then by row and the resulting matrix will contain 4 tridiagonal blocks.

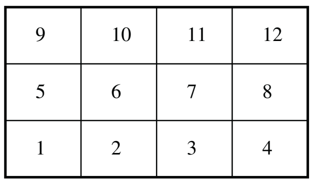


Fig. 4. 2D $n \times m$ mesh with $n = 4$, $m = 3$.

$$\begin{array}{c|cccccccccccc}
 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 \\
\hline
1 & \blacksquare & \blacksquare & & & & & & & & & & \\
2 & \blacksquare & \blacksquare & \blacksquare & & & & & & & & & \\
3 & & \blacksquare & \blacksquare & \blacksquare & & & & & & & & \\
4 & & & \blacksquare & \blacksquare & & & & & & & & \\
5 & & & & & \blacksquare & \blacksquare & & & & & & \\
6 & & & & & \blacksquare & \blacksquare & \blacksquare & & & & & \\
7 & & & & & & \blacksquare & \blacksquare & \blacksquare & & & & \\
8 & & & & & & & \blacksquare & \blacksquare & & & & \\
9 & & & & & & & & & \blacksquare & \blacksquare & & \\
10 & & & & & & & & & \blacksquare & \blacksquare & \blacksquare & \\
11 & & & & & & & & & & \blacksquare & \blacksquare & \blacksquare \\
12 & & & & & & & & & & & \blacksquare & \blacksquare
\end{array}$$

Fig. 5. An example of block tridiagonal matrix for operator $\Delta_x$.

We apply the Thomas algorithm within each tridiagonal block. If the tridiagonal blocks are identical, then we consider a smaller system with multiple right-hand sides (RHS) that can be expressed as $AX = S$ with $A \in \mathbb{R}^{n\times n}$ and $X, S \in \mathbb{R}^{n\times m}$. In the following we explain how the solution of this system can be accelerated using vectorization techniques.

Since the late 90's, processor manufacturers provide specialized processing units called multimedia extensions or Single Instruction Multiple Data (SIMD) extensions. This feature allows processors to exploit the latent data parallelism available in applications by executing a given instruction simultaneously on multiple data stored in a single special register. However, taking advantage of SIMD extensions in applications remains a complex task. We can find for instance in [32] a description of Boost.SIMD, a high-level C++ library to program SIMD architectures. Boost.SIMD provides both expressiveness and performance by using generic programming to handle vectorization in a portable way.

A first requirement to achieve performance using SIMD is to ensure memory alignment. In our case, this means that the Fortran arrays have to be allocated by a dedicated C++ function that uses the *ISO_C_BINDING* interface.

The SIMD opportunity in our algorithm stands from the fact that solving the tridiagonal systems with multiple RHS can be performed in parallel. This means that once we have an SIMD vector containing the tridiagonal values of a set of rows, the backward substitution of these rows can be issued as a single SIMD instruction. The triangularization followed by the backward substitution will be accelerated by the fact that data handled at each step of the algorithm will be contiguous in memory using a shuffling operation. Shuffling is a typical idiom of SIMD programming that replaces some class of complex memory access patterns by computation on simpler patterns. Depending on the SIMD extension, these shuffling operations can either be limited (like in SSE2 where shuffling can only occur piecewise inside a given register) or be arbitrary (e.g. in Altivec where shuffling corresponds to permutations of bytes). Shuffling also enables deinterleaving of scattered data that can be fetched from main memory and bring back into a single, contiguous SIMD register. In our case, we use shuffling to aggregate values from the sparse representation of the system into a set of contiguous values located at the proper place so that the elimination and backward substitution phases can take place inside an SIMD vector. Expected benefit should be a faster code due to less memory access in different part of the memory (thus limiting cache misses) and augmented SIMD speedup due to the layout of the data after the shuffling and better use of pipelining.

We illustrate this method in Fig. 6 by considering 2 RHS (case of SSE where a double precision vector holds 2 values) where we would perform the following operations:

- load 3 registers units with coefficients $(a_i, a_{i+1})$, $(b_i, b_{i+1})$ and $(c_i, c_{i+1})$ coming from respectively the lower, main and upper diagonals,
- shuffle the data to have a $2 \times 3$ local matrix $\begin{pmatrix} a_i & b_i & c_i \\ 0 & a_{i+1} & b_{i+1} \end{pmatrix}$,
- perform two Gaussian eliminations to eliminate the coefficients $(a_i, a_{i+1})$ in the lower diagonal,
- after elimination of the $a_i$'s, we perform a backward substitution to solve the upper triangular system with the RHS coefficients loaded in one register (e.g. 2 RHS for SSE or 4 RHS for AVX SIMD extension).

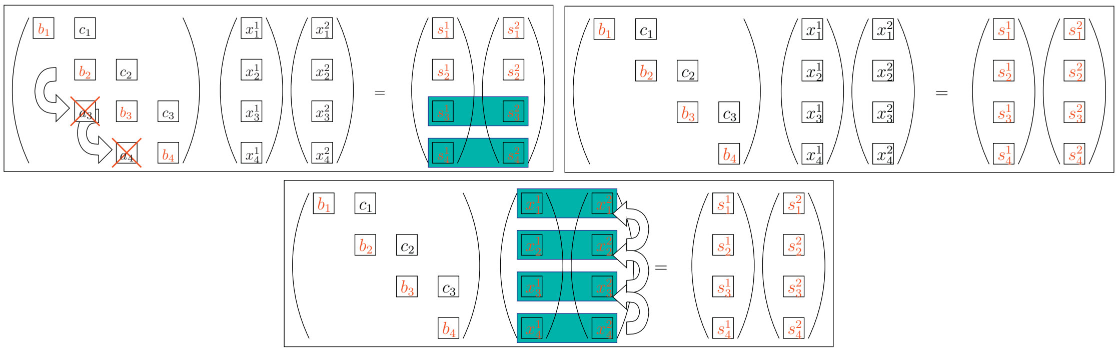


Fig. 6. Illustations of Thomas algorithm with vectorization. (a) current iteration; (b) after eliminations; (c) backward substitution.

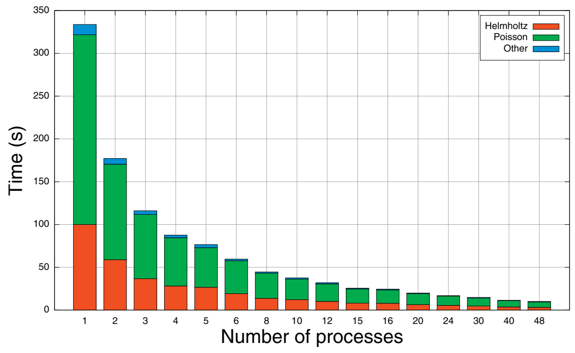
Our implementation uses this method to handle multiple RHS. Note that the memory alignment mentioned previously enables here the "load" process to be performed efficiently. We point out that this vectorization approach can be also used in other parts of our solver, for instance to compute the source term faster.

## 4. Performance results

The following experiments were carried out using a MagnyCours-48 system from University of Tennessee. This machine has a NUMA architecture and is composed of four AMD Opteron 6172 (with a SSE-4a instruction set) running at 2.1GHz with twelve cores each (48 cores total) and 128GB of memory. Our solver is linked with the LAPACK and ScaLAPACK routines from the 10.3.6 version of the Intel MKL [23] library and communications are performed using OpenMPI 1.4.3. In our experiments, we use one MPI process per core and each MPI process is a single thread.

In Fig. 4.(a) we consider a 3-D vortex problem with mesh size $240^3$, e.g. about $1.4 \times 10^7$ unknowns. We represent the performance (in seconds) for one iteration of the NS solver including the Helmholtz and Poisson equations, and miscellaneous tasks (mainly I/O and velocity/pressure updates). The number of threads varies from 1 to 48. We observe in Fig. 4.(a) that the CPU time decreases significantly with the number of threads, showing then a good scalability of the solver. We observe that the Helmholtz equation represents about 30% of the global computational time and this percentage remains the same when the number of threads increases.

Fig. 4.(b) shows the parallel speedup of the solver. We obtain a speedup up of 33 using 48 cores. We also observe in Fig. 4.(b) that the solution for the Poisson equation scales slightly better than for the Helmholtz equation. This is because in the Poisson equation we have only one tridiagonal system to solve (vs three systems in Helmholtz equation) and thus there is less information exchanged (as mentionned in Section 3, when we solve



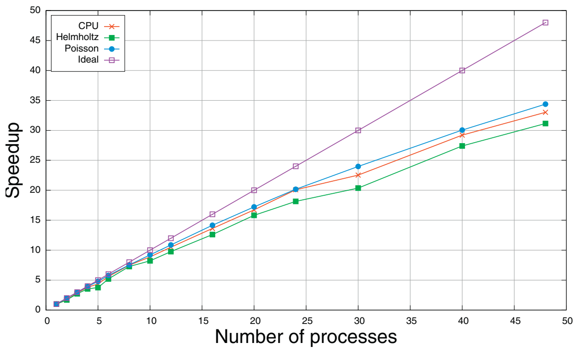


Fig. 7. (a) time breakdown for one iteration of the NS solver; (b) scalability of NS solver.

a tridiagonal system via the Schur complement method, we have a reordering of the variables that results in additionnal communication).

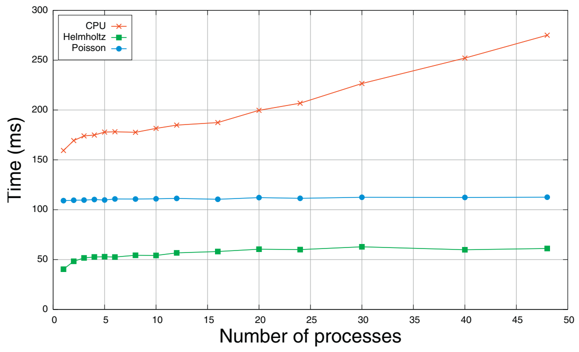


Fig. 8. Scalability of problems with variable size.

In a second experiment, we consider a 3-D vortex problem where the size increases with the number of threads and with a fixed mesh size per subdomain (weak scalability). The local size is set to $240 \times 240 \times 10$ and 10 time iterations are performed. We observe in Fig. 8 that the time spent in solving Helmholtz and Poisson equations does not vary (about 60 and 110 milliseconds respectively) because the number of unknowns computed is always $240^2 \times 10 = 5.76 \times 10^5$. However the global CPU time increases linearly with the number of processes due to a larger amount of I/O after each iteration. Indeed, the numerical solution is stored after each time iteration in order to generate a detailed animation that visualizes the fluid movement. In this numerical test, we only divide the domain along one direction ($z-$direction here) to keep the shape of the interfaces. Thus, the amount of information to be exchanged (the number of interface elements) is $240^2 \times$ (number of MPI processes $- 1$).

Let us now study more specifically the performance of the tridiagonal solver described in Section 3.2. We compare the performance of a vectorized Thomas algorithm (as illustrated by Fig. 6) with the LAPACK 3.2 routine DGTSV from Netlib, linked with the MKL BLAS library, which solves a general tridiagonal system using Gaussian elimination with partial pivoting (note that, since our matrices are diagonally dominant, the routine DGTSV does not pivot and only the search for pivot is performed).

In Fig. 9.(a), we plot the number of cycles per value, e.g. the amount of CPU cycles required to compute one element of the result (we operate here on tridiagonal blocks of size 100 in order to exhibit the cache effects). This metric allows us to do a fine grain analysis of both the impact of vectorization and the impact of memory access. For the curve related to DGTSV, we observe a cycle/value amount that jumps when we hit the cache size (L2 and L3). This is due to the fact that every computation in the elimination and backward substitution processes requires more memory access to be completed. However, when we consider the vectorized Thomas algorithm, we have a constant amount of CPU cycle/value up to the largest sizes, due to the shuffling that replaces memory access by computation. Fig. 9.(b) shows the time ratio between DGTSV and the vectorized Thomas algorithm. For large problem sizes (e.g. more than $10^6$ unknowns), we observe that the vectorized version of Thomas algorithm outperforms DGTSV with a factor 3.

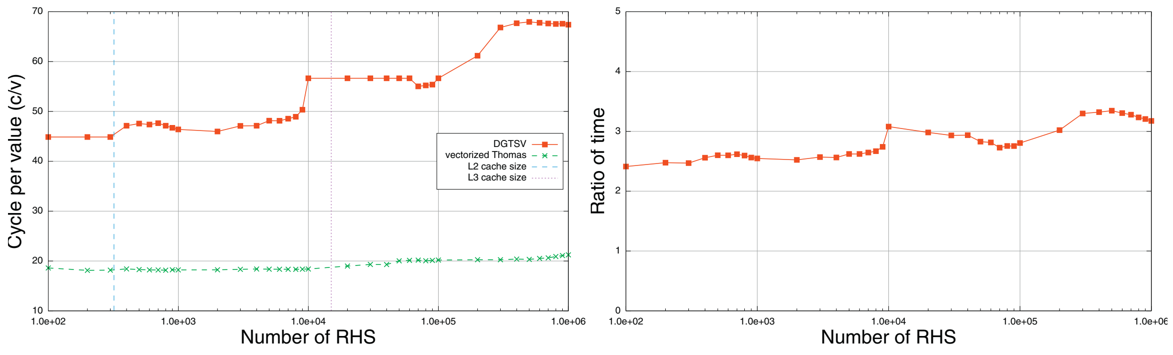


Fig. 9. (a) cycles per value for Thomas algorithm; (b) time ratio between DGTSV and vectorized Thomas.

## 5. Conclusion

In this paper, we presented a numerical method and a parallel implementation for the solution of the 3-D Navier-Stokes equations. The resulting solver is scalable on problems of realistic size and the solution of the tridiagonal systems can be significantly enhanced by using vectorization. We point out that this solver can also take into account other characteristics of the fluid or of the domain that are not discussed in details in the paper. For example, several boundary conditions are implemented and the domain can be treated either as continuous or discontinuous. Note that other physical factors like temperature and density are also considered in the solver.

## Acknowledgments

This work was supported by Région Île-de-France and Digitéo (*http://www.digiteo.fr*), CALIFHA project, contract No 2011-038D. We also thank Pierre Esterie (University Paris-Sud) for his support in using Boost SIMD.

## References

[1] J. Anderson, Computational Fluid Dynamics: The Basics with Applications, McGraw-Hill, 1995.
[2] J. H. Ferziger, M. Perić, Computational Methods for Fluid Dynamics, 3rd Edition, Springer, 1996.
[3] F. Harlow, J. E. Welch, Numerical calculation of time-deendent viscous incompressible flow of fluid with free surface, The Physics of Fluids 8 (12) (1965) 2182–2189.
[4] J. P. D. Angeli, A. M. P. Valli, N. C. Reis Jr., A. F. D. Souza, Finite difference simulations of the Navier-Stokes equations using parallel distributed computing, 15th Symposium on Computer Architecture and High Performance Computing, 2003.
[5] O. Pironneau, The Finite Element Method for Fluids, 1st Edition, John Wiley & Sons, 1990.
[6] R. Rannacher, Finite element methods for the incompressible Navier-Stokes equations, Fundamental Directions in Mathematical Fluid Mechanics (2000) 191–293.
[7] A. Q. C. Canuto, M. Meneguzzi, T. Zang, Spectral Methods: Fundamentals in Single Domains, Springer, 2006.
[8] A. Vincent, M. Meneguzzi, The spatial structure and statistical properties of homogeneous turbulence, Journal of Fluid Mechanics 225 (1991) 1–20.
[9] G. Cottet, P. Koumoutsakos, Vortex Methods Theory and Practice, Cambridge University Press, 2000.
[10] J. J. Monaghan, Smoothed particle hydrodynamics, Annual Review of Astronomy and Astrophysics 30 (1992) 543–574.
[11] S. Li, W. K. Liu, Meshfree and particle methods and their applications, Applied Mechanics Reviews 55 (2002) 1–34.
[12] D. Martin, P. Colella, N. Keen, A. Deane, G. Brenner, A. Ecer, *et al*, An incompressible Navier-Stokes with particles algorithm and parallel implementation, in: Proceedings of the 2005 International Conference on Parallel Computational Fluid Dynamics, College Park, MD, USA, 2006, pp. 461–468.
[13] A. J. Chorin, Numerical solution of the Navier-Stokes equations, Mathematics of Computation 22 (104) (1968) 745–762.

[14] J. Kim, P. Moin, Application of a fractional-step method to incompressible Naver-Stokes equations, Journal of Computational Physics 59 (1985) 308–323.

[15] J. L. Guermond, P. D. Minev, J. Shen, An overview of projection methods for incompressible flows, CMAME 195 (2006) 6011–6045.

[16] D. L. Brown, R. Cortez, M. L. Minion, Accurate projection methods for the incompressible Navier-Stokes equations, Journal of Computational Physics (2001) 464–499.

[17] H. Abdi, The eigen-decomposition: Eigenvalues and eigenvectors, in: N. J. Salkind (Ed.), Encyclopedia of Measurement and Statistics, Sage Publications, 2007, pp. 304–308.

[18] Y. Saad, Iterative Methods for Sparse Linear Systems, 2nd Edition, SIAM, 2003.

[19] M. P. I. Forum, MPI : A Message-Passing Interface Standard, Int. J. Supercomputer Applications and High Performance Computing, 1994.

[20] Basic Linear Algebra Subprograms Technical Forum Standard, Int. J. of High Performance Computing Applications 16 (1).

[21] E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Dongarra, *et al.*, LAPACK Users' Guide, SIAM, Philadelphia, 1999, third edition.

[22] L. Blackford, J. Choi, A. Cleary, E. D'Azevedo, J. Demmel, I. Dhillon, *et al.*, ScaLAPACK Users' Guide, SIAM, Philadelphia, 1997.

[23] Intel, Math Kernel Library (MKL), `http://www.intel.com/software/products/mkl/`.

[24] A. Graham, Kronecker Products and Matrix Calculus with Application, Wiley, New York, 1981.

[25] U. M. Ascher, L. R. Petzold, Computational Methods for Ordinary Differential Equations and Differential-Algebraic Equations, SIAM, 1998.

[26] W. Hundsdorfer, Partially implicit BDF2 blends for convection dominated flows, Journal of Numerical Analysis 38 (6) (2001) 1763–1783.

[27] S. Noury, S. Boivin, O. L. Maître, A fast Poisson solver for OpenCL using multigrid methods, GPU Pro2: Advanced Rendering Techniques, 2011.

[28] D. A. Bini, B. Meini, The cyclic reduction algorithm: From Poisson equation to stochastic processes and beyond, Journal of Numerical Algorithm 51 (1) (2009) 23–60.

[29] G. E. Karniadakis, R. M. Kirby II, Parallel Scientific Computing in C++ and MPI, Cambridge University Press, 2003.

[30] E. Polizzi, A. H. Sameh, A parallel hybrid banded system solver: the SPIKE algorithm, Journal of Parallel Computing 32 (2006) 177–194.

[31] K. Atkinson, W. Han, Elementary Numerical Analysis, John Wiley & Sons, 2003.

[32] P. Esterie, M. Gaunard, J. Falcou, J.-T. Lapresté, B. Rozoy, Boost.simd: generic programming for portable simdization, in: Proceedings of the 21st international conference on Parallel architectures and compilation techniques, New York, NY, USA, 2012, pp. 431–432.