Available online at www.sciencedirect.com

SciVerse ScienceDirect

Procedia Computer Science 00 (2013) 000–000

Procedia Computer Science

www.elsevier.com/locate/procedia

International Conference on Computational Science, ICCS 2013

A parallel solver for incompressible fluid flows

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

[a] *LRI, Université Paris-Sud, France*
[b] *Inria Saclay Île-de-France, France*
[c] *University of Tennessee, USA*
[d] *Oak Ridge National Laboratory, USA*
[e] *University of Manchester, United Kingdom*
[f] *LIMSI-CNRS, 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 a prediction-projection method that 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 present an implementation based on MPI where the computations are performed on each subdomain and information is exchanged at the interfaces between subdomains. We describe in particular how the solution of tridiagonal systems can be accelerated using vectorization techniques.

Keywords: Navier-Stokes equations ; prediction-projection method; ADI method; partial diagonalization; SIMD ; parallel computing, tridiagonal systems.

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, 5], *finite element* [6, 7], *finite volume* [1, 3], *spectral methods* [8, 9] and mesh-free methods such as the *vortex method* [10] or *smoothed particle hydrodynamics* [11,

*Corresponding author.
E-mail address: yushan.wang@lri.fr.

12, 13]. 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.

In this paper, we focus on FD discretizations of the three-dimensional NS equations which are solved using the classical prediction-projection method [2, 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 resolution 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 development 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 $10^5 - 10^9$ unknowns to properly capture the flow dynamics. A standard sequential algorithm cannot meet this demand and hence the use of parallel computers is mandatory. Another important advantage brought by the prediction-correction approach on tensored grids is that it yields systems of Poisson / Helmholtz equations which can be consistently reduced to one-dimensional operators using operator splitting and partial diagonalization [17], resulting in discrete tridiagonal systems for which the direct solution can be accelerated by taking advantage of the hardware. 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 diffusion equations are then solved over each subdomain in parallel using the Schur complement method 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) [18]. We also rely on the BLAS [19], LAPACK [20] and ScaLAPACK [21] kernels from the MKL library to further speedup the computations. 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 for architectures with up to 48 cores (or MPI processes). 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 Δ the Laplacian operators, while $\otimes$ is the Kronecker product [22]. In the following, we use the shorthand notation $\mathcal{CT} := \boldsymbol{\nabla} \cdot (\boldsymbol{V}^T \otimes \boldsymbol{V})$ for the convective term. In the NS equations $\mathrm{Re} := \rho U L / \nu$ is the Reynolds number based on the reference length L, velocity U, fluid density ρ and viscosity ν. The Reynolds number compares the characteristic inertial and viscous forces. Flows with same Re number exhibit the same structures. We see from the NS equations that the larger Re is, the more dominant the convection is. The diffusion effect makes the flow smoother, more regular, and spreads fluid structures and eddies. So as Re increases, the fluid structures get smaller because they are not dissipated effectively which demands a finer discretization for the simulation. The NS equations also

involve 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 condition is chosen be to Dirichlet for both velocity and pressure.

2.1. Prediction-Projection method

In this section, we present the prediction-projection method applied on the NS equations. We denote the time level with upper-indices and Δt the time-step size. We rely on a second order approximation of the time derivative (BDF2) [23, 24], 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 $\widetilde{\mathcal{CT}}^{n+1} \approx 2\mathcal{CT}^{n} - \mathcal{CT}^{n-1}$.

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.

2.1.1. 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. (7), 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:

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

where Δ_i is the second order derivative along the $i-$direction. With this product, we can transform the 3-D Helmholtz-like equation (Eq. (7)) into a system of three 1-D Helmholtz-like equations

$$\left(I - \frac{\sigma}{\mathrm{Re}}\Delta_x\right)T_1 = S, \quad \left(I - \frac{\sigma}{\mathrm{Re}}\Delta_y\right)T_2 = T_1, \quad \left(I - \frac{\sigma}{\mathrm{Re}}\Delta_z\right)\boldsymbol{V}^{*} = T_2. \tag{7}$$

2.1.2. 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:

$$\boldsymbol{\nabla} \cdot (\boldsymbol{V}^{n+1} - \boldsymbol{V}^{*}) = -\sigma\Delta(P^{n+1} - P^{n}) + \frac{\sigma}{\mathrm{Re}}\boldsymbol{\nabla} \cdot \Delta(\boldsymbol{V}^{n+1} - \boldsymbol{V}^{*}). \tag{8}$$

Owing to $\boldsymbol{\nabla} \cdot \boldsymbol{V}^{n+1} = 0$ and to $\boldsymbol{\nabla} \cdot \Delta = \Delta\boldsymbol{\nabla}\cdot$, Eq. (8) is recast as

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

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

$$\Delta\phi = \frac{1}{\sigma}\boldsymbol{\nabla} \cdot \boldsymbol{V}^*, \tag{10}$$

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

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

There exist several methods to solve the Poisson equation given in Eq. (10). For example, the fast Fourier transformation (FFT), fast sine/cosine transformations, and multigrid methods [25]. 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 Δ_x, Δ_y and Δ_z gives three tridiagonal matrices for the considered direction. To further simplify the computation, we transform tridiagonal matrices into diagonal ones by using their eigendecomposition [26] .

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 Λ is the diagonal matrix whose diagonal elements are the corresponding eigenvalues, and Q is the square $(N \times N)$ matrix whose ith column is the eigenvector q_i of A.

For our problem, we decompose only Δ_x, Δ_y and keep Δ_z, then we have

$$(Q_x\Lambda_x Q_x^{-1} + Q_y\Lambda_y Q_y^{-1} + \Delta_z)\phi = S. \tag{11}$$

Then we multiply Eq. (11) with $Q_x^{-1}Q_y^{-1}$, and observe that

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

so we have

$$(\Lambda_x + \Lambda_y + \Delta_z)\tilde{\phi} = \tilde{S}, \text{ where } \tilde{\phi} = Q_y^{-1}Q_x^{-1}\phi \text{ and } \tilde{S} = Q_y^{-1}Q_x^{-1}S. \tag{12}$$

So the original Poisson equation (Eq. 10) is now transformed into the form of Eq. (12) which is a 1-D tridiagonal equation.

2.2. Spatial Discretization

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, in the x, y and z directions. 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 [27]. The discretization of the convective term $C\mathcal{T}$ requires a mixture of central differences and donor-cell discretization [28] for strongly convective problems (Re >> 1) to ensure stability.

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 [26] is used to ensure continuity of the solution at the interface between subdomains, and the information at the interfaces is exchanged via MPI routines.

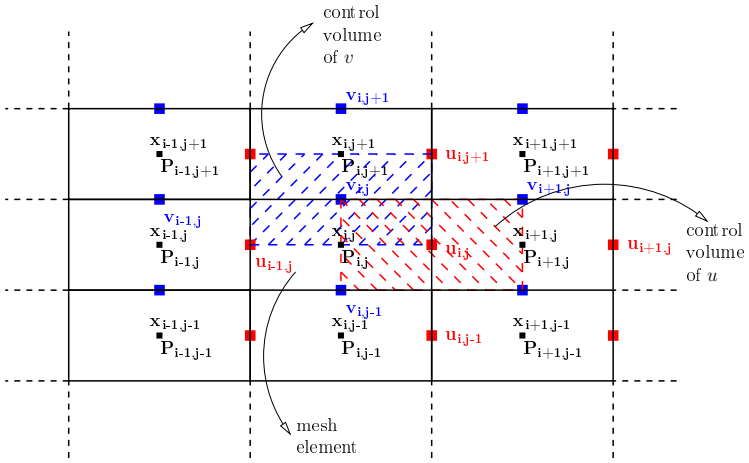


Fig. 1. Staggered mesh structure.

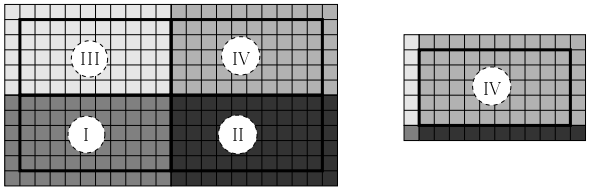
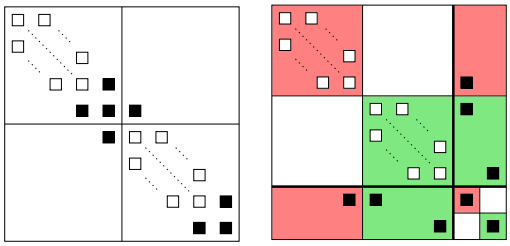


Fig. 2. Domain decomposition and boundary cells.



(a) Original tridiagonal matrix. (b) Matrix after reordering.

Fig. 3. Illustration of variable reordering for Schur Complement method for the case of 2 subdomains. (□: interior variables. ■: interface variables.)

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.

We suppose now that we have obtained the tridiagonal system for one direction (for example the x-direction in Fig. 2). The matrix structure is shown by Fig. 3(a). This tridiagonal system is solved by the Schur complement method. We reorder the variables in such ways that the interface variables are ordered after the interior variables. In this way, we obtain the new system in which the matrix is shown by Fig. 3(b).

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

$$\begin{pmatrix} A_k & A_{ki} \\ A_{ik} & A_i \end{pmatrix} \begin{pmatrix} X_k \\ X_i \end{pmatrix} = \begin{pmatrix} S_k \\ S_i \end{pmatrix}, \tag{13}$$

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

From the first block equation in Eq. (13) the unknown vector X_k can be expressed as

$$X_k = A_k^{-1}(S_k - A_{ki}X_i), \tag{14}$$

and we substitute this into the second equation to obtain

$$(A_i - A_{ik}A_k^{-1}A_{ki})X_i = S_i - A_{ik}A_k^{-1}S_k. \tag{15}$$

If the Schur complement matrix $A_i - A_{ik}A_k^{-1}A_{ki}$ can be formed and the linear system (15) can be solved, all the interface variables X_i are then available and the remaining unknowns can be computed via Eq. (14). 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. (7) (using ADI method) and Eq. (10) (using partial diagonalization) can be solved efficiently. There have been several efforts in recent years to develop efficient tridiagonal solvers [29, 30, 31]. A well-known procedure is based on the Thomas algorithm [32] which basically corresponds to a Gaussian elimination without pivoting.

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 4×3 2D mesh and let us consider the operator Δ_x in Eq. (7). We order the unknowns first by row and then by column as shown in Fig. 4. Then the matrix corresponding to the operator Δ_x can be represented as the matrix given in Fig. 5 (with $m = 3$ tridiagonal blocks). Similarly, if we consider the operator Δ_y in Eq. (7), the unknowns will be ordered first by column then by row and the resulting matrix will contain 3 tridiagonal blocks.

9	10	11	12
5	6	7	8
1	2	3	4

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

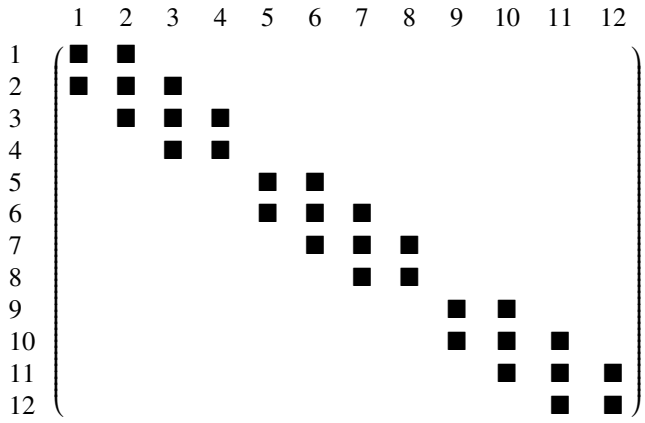


Fig. 5. An example of block tridiagonal matrix for operator Δ_x.

We apply the Thomas algorithm within each tridiagonal block (for instance $m = 3$ times for the matrix in Fig. 5). 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 [33] 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.

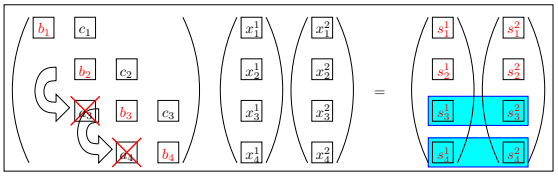
A first requirement to achieve performance using SIMD is to ensure memory alignment. In our case, this means that the Fortran arrays have to be allocated by a dedicated C++ function that uses the *ISO_C_BINDING* interface.

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

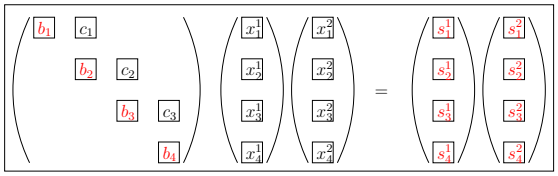
simpler patterns. Depending on the SIMD extension, these shuffling operations can either be limited (like in SSE2 where shuffling can only occur piecewise inside a given register) or be arbitrary (*e.g.* in Altivec where shuffling corresponds to permutations of bytes). Shuffling also enables deinterleaving 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 backtracking and solving 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:

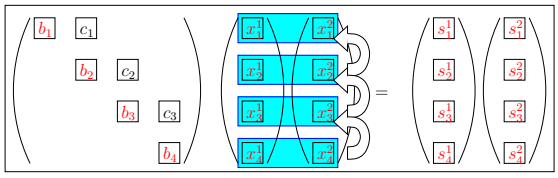
- load 3 registers units with coefficients (a_i, a_{i+1}), (b_i, b_{i+1}) and (c_i, c_{i+1}) coming from respectively the lower, main and upper diagonals,
- shuffle the data to have a 2×2 local matrix $\begin{pmatrix} b_i & c_i \\ a_{i+1} & b_{i+1} \end{pmatrix}$,
- perform two Gaussian eliminations 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).



(a) Current iteration.



(b) After elimination.



(c) Backward substitution.

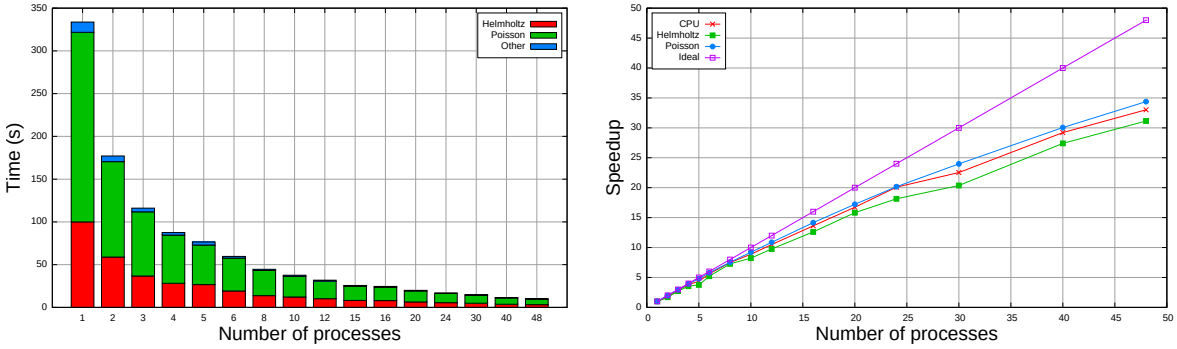
Fig. 6. Thomas algorithm with vectorization.

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. We consider a 3-D vortex problem with mesh size 240^3, *i.e.* about 1.4×10^7 unknowns. In our experiments, we use one MPI process per core and each MPI process is a single thread.

We first represent in Fig. 7(a) 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. 7(a) that the CPU time decreases significantly with the



(a) Time breakdown for one iteration of the NS solver.

(b) Scalability of NS solver.

Fig. 7. Performance of the Navier-Stokes solver.

number of threads, showing then a good scalability of the solver. Fig. 7(a) also shows the percentage of time spent in solving respectively the Helmholtz equation and the Poisson equation (one iteration). The Helmholtz equation represents about 30% of the global computational time and this percentage remains the same when the number of threads increases.

Fig. 7(b) shows the parallel speedup of the solver. We obtain a speedup up of 33 using 48 cores. We also observe in Fig. 7(b) that the solution for the Poisson equation scales 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. Note also that when we solve a tridiagonal system in the Schur complement method, we proceed the interface value arrangement, resulting in additionnal communication.

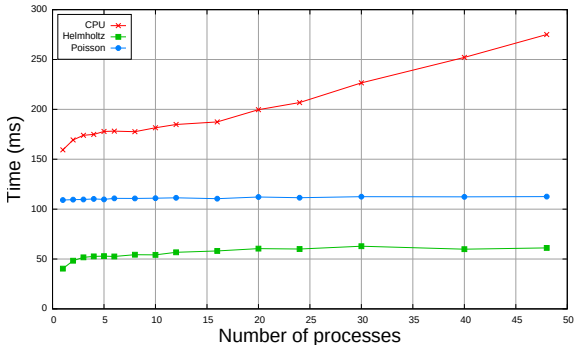


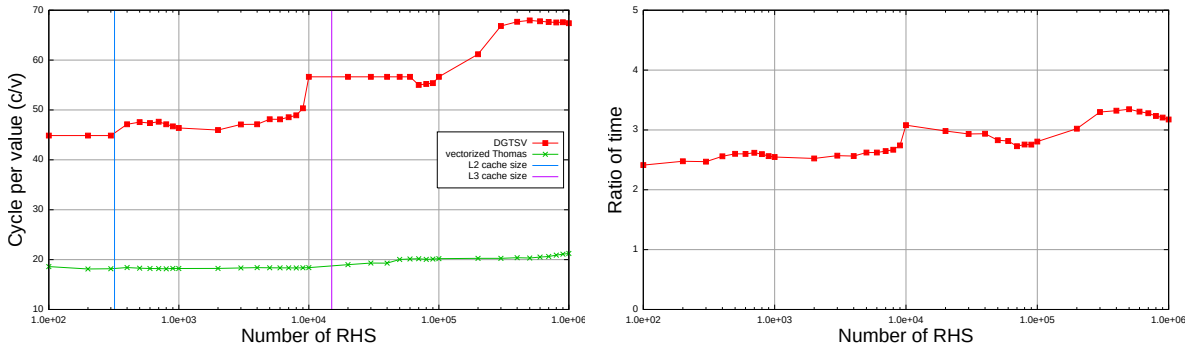
Fig. 8. Scalability of problems with variable size.

In a second experiment, we consider problems 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 temporal 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. 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 [34] BLAS multi-threaded library, which solves a general

tridiagonal system using Gaussian elimination with partial pivoting (note that, since our matrices are diagonally dominant, DGTSV does not pivot and only the search for pivot is performed).

In Fig. 9(a), we plot the number of cycles per value, *i.e.* 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. We observe, for this curve, a cycle/value amount that jumps when we hit the cache size (L2 and L3). This is due to the fact that every computation requires the full amount of memory access to be completed. However, as shown in the second curve of Fig. 9(a), by using vectorized shuffle which replaces memory access by computation, the vectorized Thomas algorithm has a constant amount of CPU cycle/value up to the largest size. Fig. 9(b) shows the time ratio between DGTSV and the vectorized Thomas algorithm. For large problem sizes, for example more than 10^6 unknowns, the vectorized version of Thomas algorithm can go three times faster than the LAPACK routine DGTSV linked with MKL BLAS library for solving a diagonally dominant tridiagonal system.



(a) Cycles per value for Thomas algorithm.

(b) Time ratio between DGTSV and vectorized Thomas.

Fig. 9. Performance of vectorized Thomas algorithm.

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 technical support in using Boost SIMD.

References

[1] J. Anderson, Computational Fluid Dynamics: The Basics with Applications, McGraw-Hill, 1995.
[2] A. J. Chorin, Numerical solution of the Navier-Stokes equations, Mathematics of Computation 22 (104) (1968) 745–762.
[3] J. H. Ferziger, M. Perić, Computational Methods for Fluid Dynamics, 3rd Edition, Springer, 1996.
[4] 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.

[5] 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.
[6] O. Pironneau, The Finite Element Method for Fluids, 1st Edition, John Wiley & Sons, 1990.
[7] R. Rannacher, Finite element methods for the incompressible Navier-Stokes equations (1999).
[8] A. Q. C. Canuto, M. Meneguzzi, T. Zang, Spectral Methods: Fundamentals in Single Domains, Springer, 2006.
[9] A. Vincent, M. Meneguzzi, The spatial structure and statistical properties of homogeneous turbulence, Journal of Fluid Mechanics 225 (1991) 1–20.
[10] G. Cottet, P. Koumoutsakos, Vortex Methods Theory and Practice, Cambridge University Press, 2000.
[11] J. J. Monaghan, Smoothed particle hydrodynamics, Annual Review of Astronomy and Astrophysics 30 (1992) 543–574.
[12] S. Li, W. K. Liu, Meshfree and particle methods and their applications.
[13] D. Martina, P. Colellaa, N. Keen, An incompressible Navier-Stokes with particles algorithm and parallel implementation.
[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, Remarques sur les méthodes de projection pour l'approximation des équations de Navier-Stokes, Numerische Mathematik 67 (1994) 465–473.
[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 (2007).
[18] M. P. I. Forum, MPI : A Message-Passing Interface Standard, Int. J. Supercomputer Applications and High Performance Computing, 1994.
[19] Basic Linear Algebra Subprograms Technical Forum Standard 16 (1).
[20] E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Dongarra, *et al.*, LAPACK Users' Guide, SIAM, Philadelphia, 1999, third edition.
[21] L. Blackford, J. Choi, A. Cleary, E. D'Azevedo, J. Demmel, I. Dhillon, *et al.*, ScaLAPACK Users' Guide, SIAM, Philadelphia, 1997.
[22] A. Graham, Kronecker products and matrix calculus with application, Wiley, New York, 1981.
[23] U. M. Ascher, L. R. Petzold, Computational Methods for Ordinary Differential Equations and Differential-Algebraic Equations, SIAM, 1998.
[24] W. Hundsdorfer, Partially implicit BDF2 blends for convection dominated flows, Journal of Numerical Analysis 38 (6) (2001) 1763–1783.
[25] S. Noury, S. Boivin, O. L. Maître, A fast Poisson solver for OpenCL using multigrid methods, GPU Pro2: Advanced Rendering Techniques, 2011.
[26] Y. Saad, Iterative Methods for Sparse Linear Systems, 2nd Edition, SIAM, 2003.
[27] R. J. Leveque, Numerical Methods for Conservation Laws, 2nd Edition, Lectures in Mathematics. Eth Zurich, 1999.
[28] T. L. Wilson, B. D. Nichols, C. W. Hirt, L. R. Stein, SOLA-DM: A numerical solution algorithm for transient three-dimensional flows (1988).
[29] 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.
[30] G. E. Karniadakis, R. M. Kirby II, Parallel Scientific Computing in C++ and MPI, Cambridge University Press, 2003.
[31] E. Polizzi, A. H. Sameh, A parallel hybrid banded system solver: the SPIKE algorithm, Journal of Parallel Computing 32 (2006) 177–194.
[32] K. Atkinson, W. Han, Elementary Numerical Analysis, John Wiley & Sons, 2003.
[33] P. Esterie, M. Gaunard, J. Falcou, J.-T. Lapresté, B. Rozoy, Boost.simd: generic programming for portable simdization, in: PACT, 2012, pp. 431–432.
[34] Intel, Math Kernel Library (MKL), `http://www.intel.com/software/products/mkl/`.