Proceedings of the Federated Conference on
Computer Science and Information Systems pp. 443–450

ISBN 978-83-60810-22-4

Parallel alternating directions algorithm for 3D Stokes equation

Ivan Lirkov
Institute of Information
and Communication Technologies
Bulgarian Academy of Sciences
Acad G. Bonchev, bl. 25A
1113 Sofia, Bulgaria
ivan@parallel.bas.bg
http://parallel.bas.bg/~ivan/

Marcin Paprzycki Maria Ganzha
Systems Research Institute
Polish Academy of Sciences
ul. Newelska 6
01-447 Warsaw, Poland
paprzyck@ibspan.waw.pl
maria.ganzha@ibspan.waw.pl
http://www.ibspan.waw.pl/~paprzyck/
http://inf.ug.edu.pl/~mganzha/

Paweł Gepner
Intel Corporation
Pipers Way
Swindon Wiltshire SN3 1RJ
United Kingdom
pawel.gepner@intel.com

***Abstract*—We consider the 3D time dependent Stokes equation on a finite time interval and on a uniform rectangular mesh, written in terms of velocity and pressure.**

For this problem, a parallel algorithm, based on a recently proposed direction splitting approach, is applied. Here, the pressure equation is derived from a perturbed form of the continuity equation, where the incompressibility constraint is penalized in a negative norm induced by the direction splitting. The scheme used in the algorithm is composed of: (a) pressure prediction, (b) velocity update, (c) penalty step, and (d) pressure correction. In order to achieve good parallel performance, the solution of the Poison problem for the pressure correction is replaced by a solution to a sequence of one-dimensional second order elliptic boundary value problems (in each spatial direction). The efficiency and scalability of the proposed approach are tested on two distinct parallel computers and the experimental results are analyzed.

I. Introduction

THE OBJECTIVE of this note is to analyze the parallel performance of a novel fractional time stepping technique, based on a direction splitting strategy, developed to solve the incompressible Navier-Stokes equations.

Computational fluid dynamics (CFD) has undergone tremendous development as a discipline. This has been made possible by progresses in many fronts, including numerical algorithms for the Navier-Stokes equations, grid generation and adaptation, turbulence modeling, flow visualization, as well as the dramatic increase of computer CPU and network speeds.

Finding an approximate solution of the Navier-Stokes equations can be done by a large range of numerical methods. Among these, finite element methods are used mostly by mathematicians, while spectral methods and finite volume methods are favored by engineers and physicists. One reason for this difference in computational practices is that an advantage of finite volume methods over finite element ones lies primarily in ease of their physical interpretation and in simpler implementation. Currently, nearly all production-class flow solvers are based on second-order numerical methods, either finite volume [9], [10], [23], [25], finite difference [33], or finite element [4], [11], [12], [18], [20]. They are capable of delivering, within a few hours, design-quality Reynolds Averaged Navier-Stokes results with several million cells (degrees of freedom) on various Beowulf-style cluster computers.

The efficient solution of the discretized Navier-Stokes equations necessitates rapidly convergent iterative methods. The two main approaches available here are: (i) preconditioned Krylov subspace methods [30], and (ii) multigrid methods [35], [36], [38]. These two approaches can be combined by using one or more multigrid cycles as preconditioners for the Krylov-type methods. Most of recent papers on the iterative solution of the discretized Navier-Stokes equations are devoted to block preconditioners [3], [7], [21], [31]. Here, more recent contributions include the preconditioning based on the augmented Lagrangian approach [3], and the least-squares commutator preconditioner generalized to the stabilized finite element discretizations of the Oseen problem [8]. Other relevant work includes the development of ILU-type preconditioners for saddle-point problems [28], and SIMPLE-type block preconditioners [29].

Alternatively, one could start with the “physics-based” iterative solution methods for the Navier-Stokes equations [26], [27] and develop preconditioners based on these techniques as described in [22]. In this case, the system is transformed by the factorization into component systems that are essentially convection-diffusion and Poisson type operators. The result is a system to which multi-level methods and algebraic multi-level methods (AMG) can be successfully applied. In recent years there has been tremendous interest in the mathematical development and practical implementation of discontinuous Galerkin finite element methods (DGFEMs) for the discretization of compressible fluid flow problems, [2], [6], [19]. The key advantages of these schemes are that the DGFEMs provide robust and high-order accurate approximations, particularly in transport-dominated regimes, and that they are locally conservative. Moreover, they provide considerable flexibility in the choice of the mesh design. Indeed, the DGFEMs can easily handle non-matching grids and non-uniform, even anisotropic, polynomial approximation degrees.

978-83-60810-22-4/$25.00 © 2011 IEEE

Projection schemes were first introduced in [5], [34] and they have been used in CFD for about forty years. During these years, such techniques went through some evolution, but the main paradigm, consisting of decomposing vector fields into a divergence-free part and a gradient, has been preserved (see [14] for a review of projection methods). In terms of computational efficiency, projection algorithms are far superior to the methods that solve the coupled velocity-pressure system. This feature makes them the most popular techniques in the CFD community for solving the unsteady Navier-Stokes equations. The computational complexity of each time step of the projection methods is that of solving one vector-valued advection-diffusion equation, plus one scalar-valued Poisson equation with Neumann boundary conditions. Note that, for large scale problems, and large Reynolds numbers, the cost of solving the Poisson equation becomes dominant.

The alternating directions algorithm, initially proposed in [13], reduces the computational complexity of the action of the incompressibility constraint. The key idea is to modify the projection paradigm, in which the vector fields are decomposed into a divergence-free part plus a gradient part. Departure from the standard projection methods has been proved to be very efficient for solving variable density flows (see, for instance, [15], [16]). In the new method, the pressure equation is derived from a perturbed form of the continuity equation, in which the incompressibility constraint is penalized in a negative norm, induced by the direction splitting. The standard Poisson problem for the pressure correction is replaced by series of one-dimensional second-order boundary value problems. This technique was proved to be stable and convergent; for details see [13]. Furthermore, a very sketchy assessment indicated that it has good potential for parallelization.

In this note we follow the proposal introduced in [13] and study its performance characteristics on two different computers. One of them is an Intel-Xeon-processor-based cluster, while the other is an IBM Blue Gene supercomputer. Experimental results reported in Section IV confirm the preliminary assessment provided in [13].

II. Stokes Equation

Let us first define the problem to be solved. We consider the time-dependent Navier-Stokes equations on a finite time interval $[0,T]$, and in a rectangular domain Ω. Since the nonlinear term in the Navier-Stokes equations does not interfere with the incompressibility constraint, we focus our attention on the time-dependent Stokes equations written in terms of velocity $\mathbf{u}$ and pressure p:

$$\begin{cases} \mathbf{u}_t - \nu\Delta\mathbf{u} + \nabla p = \mathbf{f} & \text{in} \quad \Omega\times(0,T) \\ \nabla\cdot\mathbf{u} = 0 & \text{in} \quad \Omega\times(0,T) \\ \mathbf{u}|_{\partial\Omega} = 0, \quad \partial_n p|_{\partial\Omega} = 0 & \text{in} \quad (0,T) \\ \mathbf{u}|_{t=0} = \mathbf{u}_0, \quad p|_{t=0} = p_0 & \text{in} \quad \Omega \end{cases}, \tag{1}$$

where $\mathbf{f}$ is a smooth source term, ν is the kinematic viscosity, and $\mathbf{u}_0$ is a solenoidal initial velocity field with a zero normal trace. In our work, we consider homogeneous Dirichlet boundary conditions on the velocity.

To solve thus described problem, we discretize the time interval $[0,T]$ using a uniform mesh. Furthermore, let τ be the time step used in the algorithm.

III. Parallel Alternating Directions Algorithm

For thus introduced problem, let us describe the proposed parallel solution method. In [13], Guermond and Minev introduced a novel fractional time stepping technique for solving the incompressible Navier-Stokes equations. Their approach is based on a direction splitting strategy. They used a singular perturbation of the Stokes equation with a perturbation parameter τ. The standard Poisson problem for the pressure correction was replaced by series of one-dimensional second-order boundary value problems. The focus of their work was to show numerical properties of the proposed approach (e.g. its stability and convergence). However, they also very briefly indicated its potential for efficient parallelization. Therefore, to describe the parallel solution approach, let us start from the overview of the alternating directions method.

A. Formulation of the Scheme

The scheme used in the Guermond-Minev algorithm is composed of the following parts: (i) pressure prediction, (ii) velocity update, (iii) penalty step, and (iv) pressure correction. Let us now describe an algorithm that uses the direction splitting operator

$$A := \left(1 - \frac{\partial^2}{\partial x^2}\right)\left(1 - \frac{\partial^2}{\partial y^2}\right)\left(1 - \frac{\partial^2}{\partial z^2}\right).$$

- *Pressure predictor*
  Denoting by p_0 the pressure field at $t = 0$, the algorithm is initialized by setting $p^{-\frac{1}{2}} = p^{-\frac{3}{2}} = p_0$. Next, for all $n \geq 0$, a pressure predictor is computed as follows
  $$p^{*,n+\frac{1}{2}} = 2p^{n-\frac{1}{2}} - p^{n-\frac{3}{2}}. \tag{2}$$
- *Velocity update*
  In the *velocity update* step, the velocity field is initialized by setting $\mathbf{u}^0 = \mathbf{u}_0$, and for all $n \geq 0$ the velocity update is computed by solving the following series of one-dimensional problems
  $$\frac{\boldsymbol{\xi}^{n+1} - \mathbf{u}^n}{\tau} - \nu\Delta\mathbf{u}^n + \nabla p^{*,n+\frac{1}{2}} = \mathbf{f}|_{t=(n+\frac{1}{2})\tau},$$
  $$\frac{\boldsymbol{\eta}^{n+1} - \boldsymbol{\xi}^{n+1}}{\tau} - \frac{\nu}{2}\frac{\partial^2(\boldsymbol{\eta}^{n+1} - \mathbf{u}^n)}{\partial x^2} = 0, \tag{3}$$
  $$\frac{\boldsymbol{\zeta}^{n+1} - \boldsymbol{\eta}^{n+1}}{\tau} - \frac{\nu}{2}\frac{\partial^2(\boldsymbol{\zeta}^{n+1} - \mathbf{u}^n)}{\partial y^2} = 0, \tag{4}$$
  $$\frac{\mathbf{u}^{n+1} - \boldsymbol{\zeta}^{n+1}}{\tau} - \frac{\nu}{2}\frac{\partial^2(\mathbf{u}^{n+1} - \mathbf{u}^n)}{\partial z^2} = 0, \tag{5}$$
  where $\boldsymbol{\xi}^{n+1}|_{\partial\Omega} = \boldsymbol{\eta}^{n+1}|_{\partial\Omega} = \boldsymbol{\zeta}^{n+1}|_{\partial\Omega} = \mathbf{u}^{n+1}|_{\partial\Omega} = 0$.
- *Penalty step*
  in the *Penalty step*, the intermediate parameter ϕ is approximated by solving $A\phi = -\frac{1}{\tau}\nabla\cdot\mathbf{u}^{n+1}$. Owing to the definition of the direction splitting operator A, this is

done by solving the following series of one-dimensional problems:

$$\begin{aligned} \theta - \theta_{xx} &= -\tfrac{1}{\tau}\nabla\cdot\mathbf{u}^{n+1}, & \theta_x|_{\partial\Omega} = 0, \\ \psi - \psi_{yy} &= \theta, & \psi_y|_{\partial\Omega} = 0, \\ \phi - \phi_{zz} &= \psi, & \phi_z|_{\partial\Omega} = 0, \end{aligned} \tag{6}$$

- *Pressure update*
  The last sub-step of the algorithm consists of *updating the pressure*:

$$p^{n+\frac{1}{2}} = p^{n-\frac{1}{2}} + \phi - \chi\nu\nabla\cdot\frac{\mathbf{u}^{n+1}+\mathbf{u}^{n}}{2} \tag{7}$$

The algorithm is in a standard incremental form when the parameter $\chi = 0$; while the algorithm is in a rotational incremental form when $\chi \in (0, \frac{1}{2}]$.

B. Parallel Algorithm

The proposed algorithm uses a rectangular uniform mesh combined with a central difference scheme for the second derivatives for solving equations (3–5) and (6). Thus the algorithm requires only the solution of tridiagonal linear systems.

The parallelization is based on a decomposition of the domain into rectangular sub-domains. Let us associate with each such sub-domain a set of integer coordinates (i_x, i_y, i_z), and identify it with a given processor. The linear systems, generated by the one-dimensional problems that need to be solved in each direction, are divided into systems for each set of unknowns corresponding to the internal nodes, for each block that can be solved independently by a direct method. The corresponding Schur complement for the interface unknowns between the blocks that have an equal coordinate i_x, i_y, or i_z is also tridiagonal and can therefore be easily directly inverted. The overall algorithm requires only exchange of the interface data (between sub-domains), which allows for a very efficient parallelization with an expected efficiency comparable to that of explicit schemes.

IV. Experimental Results

As stated above, the main goal of our current work is to evaluate the performance of the proposed approach; to experimentally confirm the initial positive assessment found in [13]. Therefore, to assess the performance of the proposed approach, we have solved the problem (1) in $\Omega = (0,1)^3$, for $t \in [0,2]$, with Dirichlet boundary conditions. The discretization in time was done with the time step 10^{-2}, while the parameter in the pressure update sub-step was $\chi = \frac{1}{2}$, and the kinematic viscosity was $\nu = 10^{-3}$. The discretization in space used mesh sizes $h_x = \frac{1}{n_x-1}$, $h_y = \frac{1}{n_y-1}$, and $h_z = \frac{1}{n_z-1}$. Thus, the equation (3) resulted in linear systems of size n_x, while equation (4) resulted in linear systems of size n_y, and equation (5) in linear systems of size n_z. The total number of unknowns in the discrete problem was $800\, n_x\, n_y\, n_z$.

To solve the problem, a portable parallel code was designed and implemented in C, while the parallelization has been facilitated using the MPI library [32], [37]. In the code, we used the LAPACK subroutines DPTTRF and DPTTS2 (see [1]) for solving tridiagonal systems of equations, resulting from equations (3), (4), (5), and (6), for the unknowns corresponding to the internal nodes of each sub-domain. The same subroutines were used to solve the tridiagonal systems with the Schur complement.

The parallel code has been tested on an Intel processor-based cluster computer system (Sooner), located in the Oklahoma Supercomputing Center (OSCER), and the IBM Blue Gene/P machine at the Bulgarian Supercomputing Center. In our experiments, times have been collected using the MPI provided timer and we report the best results from multiple runs. In what follows, we report the elapsed time T_c in seconds using c cores, the parallel speed-up $S_c = T_1/T_c$, and the parallel efficiency $E_c = S_c/c$.

Table I represents the results collected on the Sooner, which is a Dell Intel Xeon E5405 ("Harpertown") quad core-based Linux cluster. It has 486 Dell PowerEdge 1950 III nodes, and two quad core processors per node. Each processor runs at 2 GHz. Processors within each node share 16 GB of memory, while nodes are interconnected through a high-speed InfiniBand network (for additional details concerning the machine, see http://www.oscer.ou.edu/resources.php). We have used an Intel C compiler, and compiled the code with the following options: "-O3 -march=core2 -mtune=core2." Note that, even though such approach would be possible, we have not attempted at a two-level parallelization, where the OpenMP would be used within multi-core processors (or possibly within each computational node, where 8 cores reside), while the MPI would be used for the "between-nodes" parallelization. We have decided that such approach would not be warranted for the initial performance evaluation. However, due to the promising nature of our results, we plan to pursue such two-level parallelization in the near future, especially in view of increasing number of computational cores that are to appear within both Intel and AMD families of processors. Such approach may also be applicable for multi-core GPU-type processors (e.g. based on the Fermi or the Cypress architectures). We plan to explore usability of GPU processors, for the problem at hand, in the future.

It has to be noted that our code needs 11 GB of memory for solving the problem for $n_x = n_y = n_z = 400$. Since the memory on one node of the Sooner is 16 GB, it is the largest size of the discrete problem that can be solved on a single node. Therefore, results presented in Table I represent the largest problems we were able to solve.

The sets of results in each "column-box" of Table I were obtained for an equal number of unknowns per core. For large discrete problems, the execution time in one and the same "column-box" is much larger on two processors (8 cores) than on one processor, but on more processors the time is approximately constant. The obtained execution times confirm that the communication time between processors is larger than the communication time between cores within one processor. Also, the execution time for solving one and the same discrete problem decreases with increasing the number of cores, which shows that the communication in our parallel algorithm is mainly local.

TABLE I
EXECUTION TIME ON SOONER.

c	n_x	n_y	n_z	T_c	n_x	n_y	n_z	T_c	n_x	n_y	n_z	T_c	n_x	n_y	n_z	T_c
1	50	50	50	18.96	50	50	100	41.46	50	100	100	101.01	100	100	100	205.96
2	50	50	100	20.11	50	100	100	49.09	100	100	100	107.91	100	100	200	236.44
4	50	100	100	22.16	100	100	100	50.53	100	100	200	145.75	100	200	200	344.56
8	100	100	100	37.16	100	100	200	113.61	100	200	200	280.45	200	200	200	571.77
16	100	100	200	48.22	100	200	200	129.06	200	200	200	283.17	200	200	400	625.50
32	100	200	200	48.80	200	200	200	116.61	200	200	400	283.29	200	400	400	629.75
64	200	200	200	39.95	200	200	400	117.94	200	400	400	286.85	400	400	400	581.27
128	200	200	400	51.20	200	400	400	134.07	400	400	400	291.26	400	400	800	644.10
256	200	400	400	55.14	400	400	400	126.39	400	400	800	315.44	400	800	800	669.79
512	400	400	400	47.30	400	400	800	129.97	400	800	800	308.08	800	800	800	624.17
1024	400	400	800	59.18	400	800	800	212.37	800	800	800	437.97	800	800	1600	995.06
1	100	100	200	437.87	100	200	200	989.29	200	200	200	2122.42	200	200	400	4280.93
2	100	200	200	513.81	200	200	200	1078.89	200	200	400	2238.08	200	400	400	4579.28
4	200	200	200	661.82	200	200	400	1461.90	200	400	400	3251.23	400	400	400	6808.54
8	200	200	400	1273.53	200	400	400	2754.40	400	400	400	5792.10				
16	200	400	400	1374.05	400	400	400	2775.11	400	400	800	5615.87				
32	400	400	400	1294.36	400	400	800	2687.45	400	800	800	5642.81				
64	400	400	800	1296.88	400	800	800	2803.95	800	800	800	5882.27				
128	400	800	800	1409.56	800	800	800	2840.15	800	800	1600	5740.12				
256	800	800	800	1373.87	800	800	1600	2853.72	800	1600	1600	5854.39				
512	800	800	1600	1391.43	800	1600	1600	2941.25	1600	1600	1600	6153.34				
1024	800	1600	1600	1574.67	1600	1600	1600	3171.37								

TABLE II
SPEED-UP ON SOONER.

n_x	n_y	n_z	c									
			2	4	8	16	32	64	128	256	512	1024
100	100	100	1.91	4.08	5.54	13.05	29.26	68.78	126.52	167.47	187.22	230.99
100	100	200	1.85	3.00	3.85	9.08	25.86	60.01	123.94	157.16	318.42	335.50
100	200	200	1.93	2.87	3.53	7.67	20.27	59.35	114.35	196.78	393.56	463.59
200	200	200	1.97	3.21	3.71	7.50	18.20	53.13	118.27	215.01	459.83	672.63
200	200	400	1.91	2.93	3.36	6.84	15.11	36.30	83.62	201.98	411.38	683.22
200	400	400	1.92	2.71	3.20	6.41	14.00	30.73	65.74	159.85	380.78	677.28
400	400	400	1.93	2.72	3.19	6.67	14.29	31.83	63.52	146.37	391.15	814.68

TABLE III
PARALLEL EFFICIENCY ON SOONER.

n_x	n_y	n_z	c									
			2	4	8	16	32	64	128	256	512	1024
100	100	100	0.954	1.019	0.693	0.816	0.914	1.075	0.988	0.654	0.366	0.226
100	100	200	0.926	0.751	0.482	0.568	0.808	0.938	0.968	0.614	0.622	0.328
100	200	200	0.963	0.718	0.441	0.479	0.633	0.927	0.893	0.769	0.769	0.453
200	200	200	0.984	0.802	0.464	0.468	0.569	0.830	0.924	0.840	0.898	0.657
200	200	400	0.956	0.732	0.420	0.428	0.472	0.567	0.653	0.789	0.803	0.667
200	400	400	0.962	0.678	0.400	0.401	0.437	0.480	0.514	0.624	0.744	0.661
400	400	400	0.963	0.679	0.399	0.417	0.447	0.497	0.496	0.572	0.764	0.796

The somehow slower performance on Sooner using 8 cores is clearly visible. The same effect was observed during our previous work (see [24]). There are some factors which could play role for the slower performance using both processors and all available cores within each node. Generally they are a consequence of the limitations of the memory subsystems and their hierarchical organization in modern computers. One such factor might be the limited bandwidth of the main memory bus. This causes the processors literally to "starve" for data, thus, decreasing the overall performance. Since the L2 cache memory is shared among each pair of cores within the processors, this boost the performance of programs utilizing only single core within such pair (it can use the whole cache to itself). Conversely, this leads for somehow decreased speedups when all cores are used. For memory intensive programs, these factors play crucial role for the codes' performance. At this stage we have run into some technical problems attempting at running the code with specific number of cores per processor within each node. We will try to establish a way to explicitly evaluate this effect in the future.

To provide an analytical view on performance, the speed-up obtained on Sooner is reported in Table II and the parallel efficiency is shown in Table III. Here, let us recall that the discrete problem with $n_x = n_y = n_z = 400$ requires 11 GB of memory and that is why we report the speed-up and the parallel efficiency on Sooner only for problems with $100 \leq$

$n_x, n_y, n_z \leq 400$. Specifically, for larger problems we could not run the code on a single computational unit (within a node with only 16 GB of memory) and thus neither speed-up nor efficiency could be calculated.

Increasing the number of cores, the parallel efficiency decreases on 8 cores, and after that it increases. This effect is particularly visible in the case of smaller problems. Specifically, a super-linear speed-up (and thus efficiency of more than 100%) is observed for $n_x = n_y = n_z = 100$ on 4 and 64 cores. The main reasons for this fact can be related to the well-known fact that splitting a large problem into smaller sub-problems helps memory management. In particular, it allows for better usage of cache memories of individual parallel processors. Interestingly, the effect of performance dip on 8 cores is visible even for the largest reported problems (for $n_x = n_y = n_z = 400$), where the efficiency increases all the way to $c = 512$ cores. Overall, it can be stated that the performance of the code on the Sooner is more than promising for solving large problems using the proposed method.

Table IV represents execution times collected on the IBM Blue Gene/P machine at the Bulgarian Supercomputing Center. It consists of 2048 compute nodes with quad core PowerPC 450 processors (running at 850 MHz). Note that, here, a single node has only 4 cores (not 8 cores in 2 processors, as in the case of the Sooner). Each node has 2 GB of RAM (amount much smaller than the 16GB of RAM on the Sooner). For the point-to-point communications a 3.4 Gb 3D mesh network is used. Reduction operations are performed on a 6.8 Gb tree network (for more details, see http://www.scc.acad.bg/); thus the networking within the Blue Gene/P has much larger throughput than on the Sooner. We have used the IBM XL C compiler and compiled the code with the following options: "-O5 -qstrict -qarch=450d -qtune=450". Again, no attempt at the two-level parallelization was made (in this case it would be even less worthy the effort, with only 4 cores per node). Due to the limits of memory available per node (2 GB), we did run into a more severe restrictions on the problem size than in the case of the Sooner. Therefore, the largest system that we were able to solve on a single node was for $n_x = n_y = n_z = 200$. However, in the case of the Blue Gene we were able to run jobs with up to 1024 nodes, which allowed us to solve large problems (up to size $n_x = n_y = 1600, n_z = 3200$).

We observed that using 2 or 4 cores per processor leads to slower execution, e.g. the execution time for $n_x = n_y = n_z = 800, c = 512$ is 982 seconds using 512 nodes, 1079.22 seconds using 256 nodes, and 1212.62 seconds using 128 nodes. This shows that (as expected) the communication between processors is faster than the communication between cores of one processor using the MPI communication functions.

In order to get better parallel performance we plan to align the decomposition of the computational domain into sub-domains, with the topology of the compute nodes in the Blue Gene connectivity network. In such way we will minimize the communication time in the parallel algorithm.

To complete the analysis of the performance of the IBM Blue Gene/P, Table V shows the obtained speed-up, while the parallel efficiency is presented in Table VI. Recall, that due to memory limitations, the largest problem solvable on a single node was $n_x = n_y = n_z = 200$, thus limiting available speed-up and efficiency data. Observe that a super-linear speed-up is observed on up to 128 cores of the supercomputer. There are at least two causes for the higher speed-up: individual processors of the supercomputer are slower than these of the Sooner, while the communication is faster (due to the, above mentioned, special networking used in the Blue Gene). As a result, the single-processor data can be seen as relatively "slow" while in the case of multiple nodes the performance gain from using multiple processing units is boosted by the speed of the interconnect (combined with the decrease in sub-problem sizes). It is also worthy observing that as the problem size increases, the parallel efficiency increases as well (e.g. on 4096 cores, it raises from 21% to 44%). This again shows the overall parallel robustness of the approach under investigation.

Finally, we have decided to compare head-to-head both computers. To this effect, computing times obtained on both parallel systems are shown in Fig. 1, while the obtained speed-up are shown in Fig. 2.

Execution times on the Blue Gene/P are substantially larger than that on the Sooner (for the same number of cores); e.g. for the $n_x = n_y = n_z = 200$ discrete problem on 64 cores the solution time on the Sooner is ~ 40 seconds, whereas on the Blue Gene it is ~ 100 seconds (and, recall that on the Blue Gene we observe a super-linear speed-up for up to 128 cores). This difference decreases, in relative terms, as the problems size and the number of cores increase; e.g. for the discrete problem of size $n_x = 800, n_y = n_z = 1600$ the solution time on 256 cores of the Sooner is ~ 5854 seconds, while on the Blue Gene it is ~ 8177 seconds. In other words, the parallel efficiency obtained on the supercomputer is better. For instance, the execution time on single core on Sooner is 3.6 times faster than on the Blue Gene/P, in comparison with 1.4 times faster performance on 256 cores. This indicates, among others that the networking infrastructure of the Blue Gene supercomputer is superior to that of the Sooner cluster. Therefore, as the total number of nodes increases, the initial advantage of Sooner decreases. Interestingly, we have run some initial experiments on the IBM Blue Gene/P in the West University of Timisoara (for details, see http://hpc.uvt.ro/infrastructure/bluegenep/). The main hardware difference between the two machines is the 4GB per node memory of the Timisoara machine, which should result in it being more powerful. However, times obtained on that machine were substantially worse. When checking the reason we have found out that while the Sofia Center machine runs optimized LAPACK 3.2, the Timisoara Canter runs unoptimized LAPACK 3.3.1. Since we have run the same code using the same compiler options, this was the only difference we could spot. Obviously, we plan to investigate this issue further and will not report any obtained results. However, we mention this here as one of the "lessons learned." Library software that is not fully optimized may degrade performance of a code and may not be immediately noticeable

TABLE IV
EXECUTION TIME ON IBM BLUE GENE/P.

c	n_x	n_y	n_z	T_c	n_x	n_y	n_z	T_c	n_x	n_y	n_z	T_c	n_x	n_y	n_z	T_c
1	50	50	50	93.95	50	50	100	205.13	50	100	100	411.26	100	100	100	886.70
2	50	50	100	96.67	50	100	100	194.68	100	100	100	417.83	100	100	200	890.07
4	50	100	100	98.53	100	100	100	212.01	100	100	200	434.32	100	200	200	901.87
8	100	100	100	97.86	100	100	200	212.58	100	200	200	424.77	200	200	200	911.64
16	100	100	200	100.94	100	200	200	203.09	200	200	200	430.86	200	200	400	914.95
32	100	200	200	102.94	200	200	200	219.07	200	200	400	447.72	200	400	400	925.29
64	200	200	200	101.81	200	200	400	220.03	200	400	400	437.99	400	400	400	933.07
128	200	200	400	110.91	200	400	400	221.31	400	400	400	456.42	400	400	800	963.69
256	200	400	400	114.54	400	400	400	236.37	400	400	800	481.54	400	800	800	980.73
512	400	400	400	112.40	400	400	800	238.32	400	800	800	468.68	800	800	800	982.00
1024	400	400	800	126.38	400	800	800	249.90	800	800	800	494.18	800	800	1600	1048.29
2048	400	800	800	136.22	800	800	800	275.03	800	800	1600	593.02	800	1600	1600	1154.40
4096	800	800	800	170.64	800	800	1600	357.42	800	1600	1600	677.97	1600	1600	1600	1374.78
1	100	100	200	1822.27	100	200	200	3797.75	200	200	200	7715.32				
2	100	200	200	1813.75	200	200	200	3836.06	200	200	400	7660.58				
4	200	200	200	1865.64	200	200	400	3839.99	200	400	400	7749.17				
8	200	200	400	1867.87	200	400	400	3884.58	400	400	400	7870.17				
16	200	400	400	1862.39	400	400	400	3919.05	400	400	800	7810.51				
32	400	400	400	1904.62	400	400	800	3918.49	400	800	800	7874.35				
64	400	400	800	1907.69	400	800	800	3957.46	800	800	800	8006.56				
128	400	800	800	1961.68	800	800	800	4062.99	800	800	1600	8088.98				
256	800	800	800	1988.93	800	800	1600	4096.10	800	1600	1600	8177.80				
512	800	800	1600	1997.28	800	1600	1600	4119.41	1600	1600	1600	8269.49				
1024	800	1600	1600	2122.42	1600	1600	1600	4242.13	1600	1600	3200	8422.26				
2048	1600	1600	1600	2266.55	1600	1600	3200	4645.32								
4096	1600	1600	3200	2663.84												

TABLE V
SPEED-UP ON IBM BLUE GENE/P.

n_x	n_y	n_z	c											
			2	4	8	16	32	64	128	256	512	1024	2048	4096
100	100	100	2.12	4.18	9.06	17.30	33.61	64.88	117.22	206.55	357.78	454.72	635.06	872.79
100	100	200	2.05	4.20	8.57	18.05	34.65	68.43	123.19	213.54	408.99	535.78	716.23	1156.77
100	200	200	2.09	4.21	8.94	18.70	36.89	72.23	130.94	240.37	427.99	620.19	966.68	1535.38
200	200	200	2.01	4.14	8.46	17.91	35.22	75.78	136.65	254.69	451.79	793.12	1263.89	1800.66

TABLE VI
PARALLEL EFFICIENCY ON IBM BLUE GENE/P.

n_x	n_y	n_z	c											
			2	4	8	16	32	64	128	256	512	1024	2048	4096
100	100	100	1.061	1.046	1.133	1.081	1.050	1.014	0.916	0.807	0.699	0.444	0.310	0.213
100	100	200	1.024	1.049	1.072	1.128	1.083	1.069	0.962	0.834	0.799	0.523	0.350	0.282
100	200	200	1.047	1.053	1.118	1.169	1.153	1.129	1.023	0.939	0.836	0.606	0.472	0.375
200	200	200	1.006	1.034	1.058	1.119	1.101	1.184	1.068	0.995	0.882	0.775	0.617	0.440

by inexperienced user, who does not have multiple machines to run tests of her/his code on.

V. CONCLUSIONS AND FUTURE WORK

We have studied parallel performance of the recently developed parallel algorithm based on a new direction splitting approach for solving of the 3D time dependent Stokes equation on a finite time interval and on a uniform rectangular mesh. The performance was evaluated on two different parallel architectures. Satisfactory parallel efficiency was obtained on both parallel systems, on up to 1024 processors. Out of the two machines, the faster CPUs on the Sooner lead to shorter run-time, on the same number of processors.

In the near future, it is our intention to consider and compare the performance of this algorithm to other efficient methods for solving of the time dependent Stokes equation. In order to get better parallel performance using four cores per processor on the IBM Blue Gene/P (and future multi-core computers) we plan to develop mixed MPI/OpenMP code. Furthermore, we plan to synchronize the decomposition of the computational domain into sub-domains with the topology of the compute nodes in the Blue Gene connectivity network. In such way we will minimize the communication time in the parallel algorithm.

ACKNOWLEDGMENTS

Computer time grants from the Oklahoma Supercomputing Center (OSCER) and the Bulgarian Supercomputing Center (BGSC) are kindly acknowledged. This research was partially supported by grants DO02-147 and DPRP7RP-02/13 from

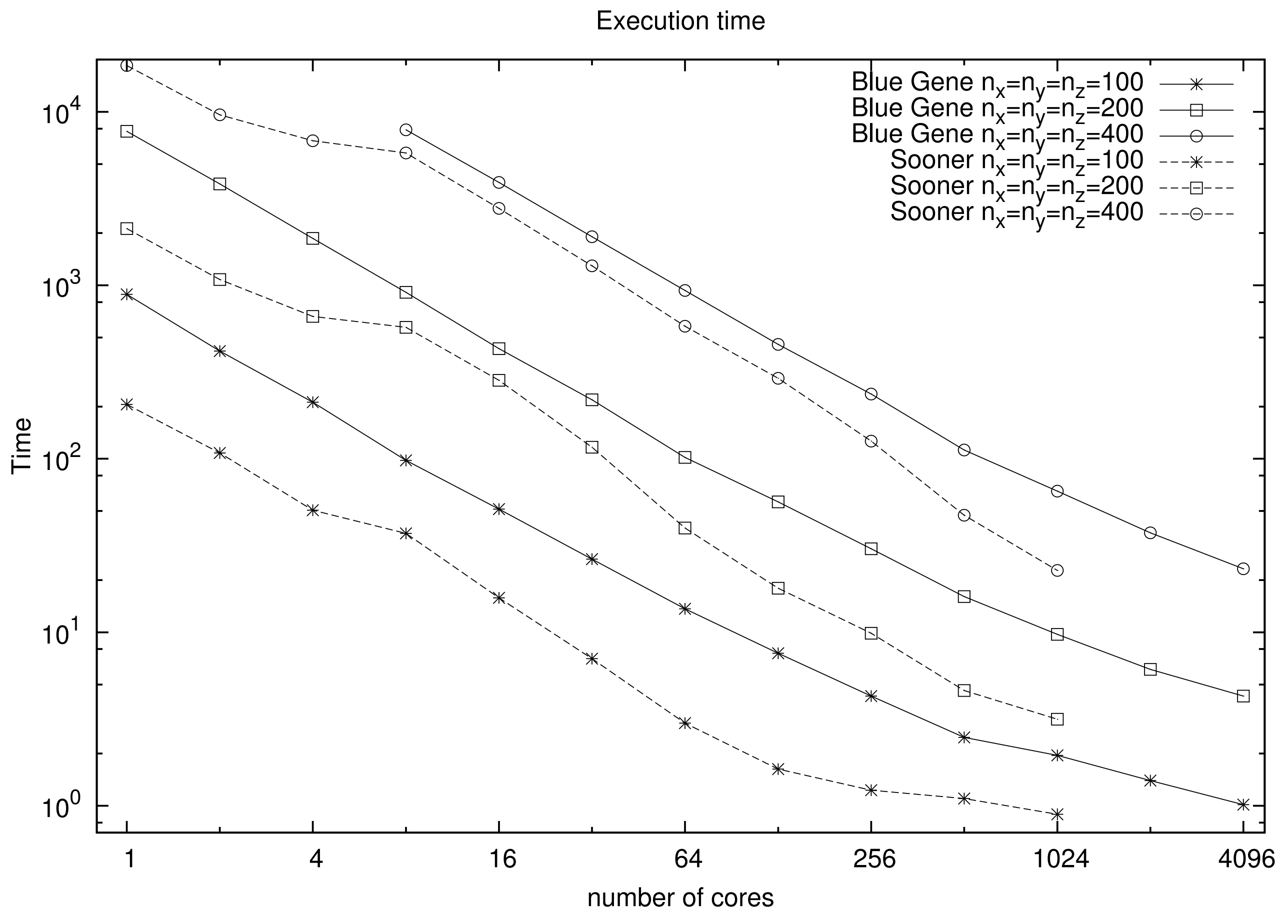

Fig. 1. Execution times for $n_x = n_y = n_z = 100, 200, 400$; both computers

the Bulgarian NSF. Work presented here is a part of the Poland-Bulgaria collaborative grant "Parallel and distributed computing practices".

REFERENCES

[1] E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, D. Sorensen, LAPACK Users' Guide, Third Edition, SIAM, 1999.

[2] C. E. Baumann, J. T. Oden, A discontinuous hp finite element method for the solution of the Euler and Navier-Stokes equations, *Int. J. Numer. Meth. Fluids*, **31**, 1999, 79–95.

[3] M. Benzi, M. A. Olshanskii, An augmented Lagrangian-based approach to the Oseen problem, *SIAM Journal on Scientific Computing*, **28** (6), 2006, 2095–2113.

[4] R. Biswas, K. D. Devine, J. E. Flaherty, Parallel, adaptive finite element methods for conservation laws, *Appl. Numer. Math.*, **14** (1–3), 1994, 255–283.

[5] A. J. Chorin, Numerical solution of the Navier-Stokes equations, *Math. Comp.*, **22**, 1968, 745–762.

[6] V. Dolejší, M. Feistauer, Ch. Schwab, On discontinuous Galerkin methods for nonlinear convection-diffusion problems and compressible flow, *Mathematica Bohemica*, **127** (2), 2002, 163–179.

[7] H. C. Elman, D. Silvester, A. J. Wathen, *Finite Elements and Fast Iterative Solvers with Applications in Incompressible Fluids Dynamics*, Oxford University Press, Oxford, 2005.

[8] H. Elman, V. E. Howle, J. Shadid, D. Silvester, R. Tuminaro, Least squares preconditioners for stabilized discretizations of the Navier-Stokes equations, *SIAM Journal on Scientific Computing*, **30**, 2007, 290–311.

[9] R. Eymard, R. Herbin, J. C. Latche, On a stabilized colocated Finite Volume scheme for the Stokes problem, *ESAIM: Math. Model. Numer. Anal.*, **40** (3), 2006, 501–527.

[10] N. T. Frink, Recent progress toward a three dimensional unstructured Navier-Stokes flow solver. AIAA Paper No. 94–0061, 1994.

[11] V. Girault, P.-A. Raviart, *Finite element methods for the Navier-Stokes equations: Theory and algorithms*, Springer, 1986.

[12] R. Glowinski, *Numerical Methods for fluids (Part 3), Handbook of Numerical Analysis*, Vol. IX, P.G. Ciarlet, J.L. Lions eds., North Holland, 2003.

[13] J.-L. Guermond, P. Minev, A new class of fractional step techniques for the incompressible Navier-Stokes equations using direction splitting, *Comptes Rendus Mathematique*, **348** (9–10), 2010, 581–585.

[14] J.-L. Guermond, P. Minev, J. Shen, An overview of projection methods for incompressible flows, *Comput. Methods Appl. Mech. Engrg.*, **195**, 2006, 6011–6054.

[15] J.-L. Guermond, A. Salgado, A splitting method for incompressible flows with variable density based on a pressure Poisson equation, *Journal of Computational Physics*, **228** (8), 2009, 2834–2846.

[16] J.-L. Guermond, A. Salgado, A fractional step method based on a pressure Poisson equation for incompressible flows with variable density, *Comptes Rendus Mathematique*, **346** (15–16), 2008, 913–918.

[17] J.-L. Guermond, J. Shen, On the error estimates for the rotational pressure-correction projection methods, *Math. Comp.*, **73** (248), 2004, 1719–1737.

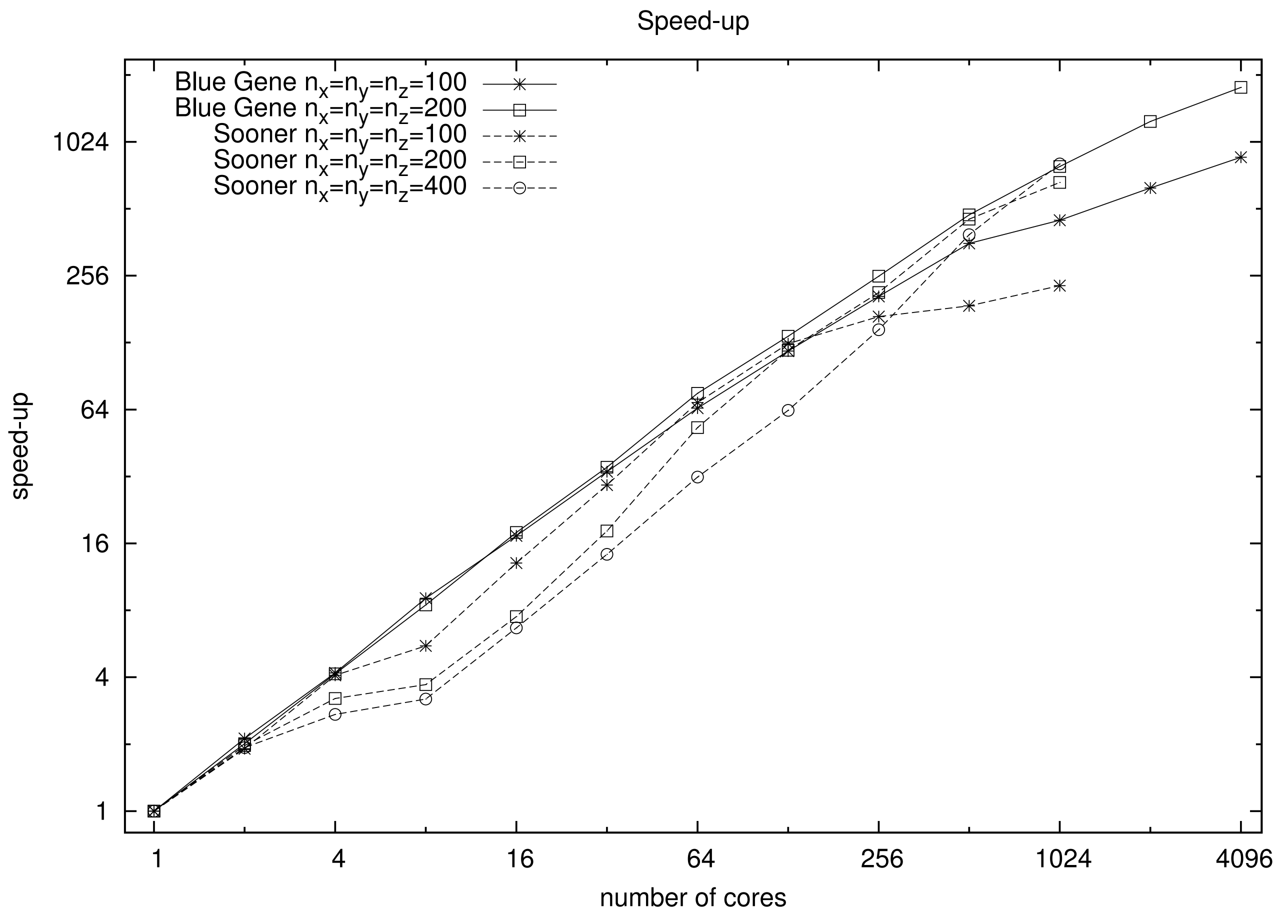

Fig. 2. Speed-up for $n_x = n_y = n_z = 100, 200, 400$; both computers

[18] M. D. Gunzburger, Finite element methods for viscous incompressible flows — A guide to theory, practice, and algorithms, Computer Science and Scientific Computing (Academic Press), 1989.

[19] R. Hartmann, P. Houston, Symmetric Interior Penalty DG Methods for the Compressible Navier-Stokes Equations I: Method formulation, *Int. J. Num. Anal. Model.*, **3** (1), 2006, 1–20.

[20] O. Hassan, K. Morgan, J. Peraire, An implicit finite element method for high speed flows. AIAA Paper No. 90–0402, 1990.

[21] D. Kay, D. Loghin, A. Wathen, A preconditioner for the steady-state Navier-Stokes equations, *SIAM Journal on Scientific Computing*, **24** (1), 2002, 237–256.

[22] D. A. Knoll, D. E. Keyes, Jacobian-free Newton-Krylov methods: a survey of approaches and applications, *Journal of Computational Physics*, **193**, 2004, 357–397.

[23] R. J. LeVeque, Finite volume methods for hyperbolic problems. Cambridge University Press, 2002.

[24] I. Lirkov, Y. Vutov, M. Paprzycki, M. Ganzha, Parallel Performance Evaluation of MIC(0) Preconditioning Algorithm for Voxel μFE Simulation, *Parallel processing and applied mathematics*, Part II, R. Wyrzykowski, J. Dongarra, K. Karczewski, J. Waśniewski ed., *Lecture notes in computer science*, **6068**, Springer, 2010, 135–144.

[25] S. V. Patankar, Numerical Heat Transfer and Fluid Flow, *Series in Computational Methods in Mechanics and Thermal Sciences*, Mc Graw Hill, 1980.

[26] S. V. Patankar, Numerical Heat Transfer and Fluid Flow, Hemisphere Publishing Corporation, New York, 1980.

[27] S. V. Patankar, D. A. Spalding, A calculation procedure for heat, mass and momentum transfer in three dimensional parabolic flows, *International Journal on Heat and Mass Transfer*, **15**, 1972, 1787–1806.

[28] M. ur Rehman, C. Vuik, G. Segal, Preconditioners for the steady incompressible Navier-Stokes problem, *International Journal of Applied Mathematics*, **38**, 2008, 223–232.

[29] M. ur Rehman, C. Vuik, G. Segal, SIMPLE-type preconditioners for the Oseen problem, *International Journal for Numerical Methods in Fluids*, **61**(4), 2009, 432–452.

[30] Y. Saad, *Iterative Methods for Sparse Linear Systems* (2nd edn), SIAM, Philadelphia, PA, 2003.

[31] Y. Saad, M. H. Schultz, GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems, *SIAM Journal on Scientific and Statistical Computing*, **7**, 1986, 856–869.

[32] M. Snir, St. Otto, St. Huss-Lederman, D. Walker, J. Dongarra, *MPI: the complete reference*, Scientific and engineering computation series. The MIT Press, Cambridge, Massachusetts, 1997, Second printing.

[33] J. L. Steger, R. F. Warming, Flux vector splitting of the inviscid gas dynamics equations with application to finite difference methods. *J. Comput. Phys.*, **40** (2), 1981, 263–293.

[34] R. Temam, Sur l'approximation de la solution des équations de Navier-Stokes par la méthode des pas fractionnaires, *Arch. Rat. Mech. Anal.*, **33**, 1969, 377–385.

[35] U. Trottenberg, C. Oosterlee, A. Schuller, *Multigrid*, Academic Press, San Diego, 2001.

[36] S. Turek, *Efficient Solvers for Incompressible Flow Problems*, Springer, Berlin, 1999.

[37] D. Walker, J. Dongarra, MPI: a standard Message Passing Interface, *Supercomputer*, **63**, 1996, 56–68.

[38] P. Wesseling, *Principles of Computational Fluid Dynamics*, Springer Series in Computational Mathematics, **29**, Springer, New York, 2001.