Performance Analysis of Parallel Alternating Directions Algorithm for Time Dependent Problems

Ivan Lirkov[1], Marcin Paprzycki[2], and Maria Ganzha[2]

[1] 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/`
[2] Systems Research Institute, Polish Academy of Sciences,
ul. Newelska 6, 01-447 Warsaw, Poland
`{paprzyck,maria.ganzha}@ibspan.waw.pl`
`http://www.ibspan.waw.pl/~paprzyck/`
`http://inf.ug.edu.pl/~mganzha/`

Abstract. We consider the time dependent Stokes equation on a finite time interval and on a uniform rectangular mesh, written in terms of velocity and pressure. In a parallel algorithm, based on a new direction splitting approach, 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 scheme used in the algorithm is composed of: pressure prediction, velocity update, penalty step, and pressure correction. In order to achieve good parallel performance, the solution of the Poison problem for the pressure correction is replaced by solving a sequence of one-dimensional second order elliptic boundary value problems in each spatial direction. The parallel code was developed using MPI and tested on modern computer systems. The performed numerical tests illustrate the parallel efficiency, and the scalability, of the direction-splitting based algorithm.

1 Introduction

The objective of this paper 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.

Projection schemes were introduced in [2,9] and they have been used in Computational Fluid Dynamics (CFD) since. 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 [4] for a review. In terms of computational efficiency, projection algorithms are far superior to the methods that solve the coupled velocity-pressure system, making them the most popular techniques for solving unsteady Navier-Stokes equations.

The alternating directions algorithm proposed in [3] reduces the computational complexity of the enforcement of the incompressibility constraint. The key idea consists of abandoning the projection paradigm in which vector fields are decomposed into a divergence-free part plus a gradient part. Departure from the projection paradigm has been proved to be very efficient for solving variable density flows [5,6]. 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 the series of one-dimensional second-order boundary value problems. This technique is proved to be stable and convergent (see [3]). Furthermore, a very brief initial assessment, found in [3], indicates that the new approach should be efficiently parallelizable. The aim of this paper is to experimentally investigate this claim on three distinct parallel systems, for two dimensional problems.

2 Stokes Equation

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 henceforth mainly focus our attention on the time-dependent Stokes equations written in terms of velocity with components (u,v) and pressure p:

$$\begin{cases} u_t - \nu\,(u_{xx}+u_{yy}) + p_x = f & \\ v_t - \nu\,(v_{xx}+v_{yy}) + p_y = g & \text{in } \Omega\times(0,T) \\ u_x + v_y = 0 & \\ u|_{\partial\Omega} = v|_{\partial\Omega} = 0, \quad \partial_n p|_{\partial\Omega} = 0 & \text{in } (0,T) \\ u|_{t=0} = u_0, \quad v|_{t=0} = v_0, \quad p|_{t=0} = p_0 & \text{in } \Omega \end{cases}, \qquad (1)$$

where a smooth source term has components (f,g), ν is the kinematic viscosity, and (u_0,v_0) is a solenoidal initial velocity field with a zero normal trace. The time interval $[0,T]$ was discretized on a uniform mesh and τ was the time step.

3 Parallel Alternating Directions Algorithm

Guermond and Minev introduced (in [3]) a novel fractional time stepping technique for solving the incompressible Navier-Stokes equations, based on a direction splitting strategy. They used a singular perturbation of Stokes equation with a perturbation parameter τ. The standard Poisson problem was replaced by series of one-dimensional second-order boundary value problems.

3.1 Formulation of the Scheme

The scheme used in the algorithm is composed of the following parts: pressure prediction, velocity update, penalty step, and pressure correction. We now describe an algorithm that uses the direction splitting operator

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

– *Pressure predictor.*
Denoting p_0 the pressure field at $t = 0$, the algorithm is initialized by setting $p^{-\frac{1}{2}} = p^{-\frac{3}{2}} = p_0$. Then for all $n \geq 0$ a pressure predictor is computed:

$$p^{*,n+\frac{1}{2}} = 2p^{n-\frac{1}{2}} - p^{n-\frac{3}{2}}. \tag{2}$$

– *Velocity update.*
The velocity field is initialized by setting $\mathbf{u}^0 = \begin{pmatrix} u_0 \\ v_0 \end{pmatrix}$, and for all $n \geq 0$ the velocity update is computed by solving the following series of one-dimensional problems

$$\frac{\boldsymbol{\xi}^{n+1} - \mathbf{u}^n}{\tau} - \nu\Delta\mathbf{u}^n + \nabla p^{*,n+\frac{1}{2}} = \mathbf{f}^{n+\frac{1}{2}},\ \ \boldsymbol{\xi}^{n+1}|_{\partial\Omega} = 0, \tag{3}$$

$$\frac{\boldsymbol{\eta}^{n+1} - \boldsymbol{\xi}^{n+1}}{\tau} - \frac{\nu}{2}\frac{\partial^2(\boldsymbol{\eta}^{n+1} - \mathbf{u}^n)}{\partial x^2} = 0,\ \ \boldsymbol{\eta}^{n+1}|_{\partial\Omega} = 0, \tag{4}$$

$$\frac{\mathbf{u}^{n+1} - \boldsymbol{\eta}^{n+1}}{\tau} - \frac{\nu}{2}\frac{\partial^2(\mathbf{u}^{n+1} - \mathbf{u}^n)}{\partial y^2} = 0,\ \ \mathbf{u}^{n+1}|_{\partial\Omega} = 0, \tag{5}$$

where $\mathbf{f}^{n+\frac{1}{2}} = \begin{pmatrix} f|_{t=\left(n+\frac{1}{2}\right)\tau} \\ g|_{t=\left(n+\frac{1}{2}\right)\tau} \end{pmatrix}$.

– 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:

$$\begin{array}{ll} \psi - \psi_{xx} = -\frac{1}{\tau}\nabla\cdot\mathbf{u}^{n+1}, & \psi_x|_{\partial\Omega} = 0, \\ \phi - \phi_{yy} = \psi, & \phi_y|_{\partial\Omega} = 0, \end{array} \tag{6}$$

– *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 $\chi \in (0, \frac{1}{2}]$.

3.2 Parallel Algorithm

We use a rectangular uniform mesh combined with a central difference scheme for the second derivatives for solving equations (4–5), 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 coordinates (i_x, i_y), and identify it with a given processor. The linear systems, generated by one-dimensional problems that need to be solved in each direction, are divided into systems for sets 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 or i_y is also tridiagonal and can be inverted directly. The overall algorithm requires only exchange of the interface data, which allows for a very efficient parallelization with an efficiency comparable to that of explicit schemes.

4 Experimental Results

The problem (1) is solved in $\Omega = (0,1)^2$, for $t \in [0,2]$ with Dirichlet boundary conditions. The discretization in time is done with time step 10^{-2}, the parameter $\chi = \frac{1}{2}$, the kinematic viscosity $\nu = 10^{-3}$. The discretization in space uses mesh sizes $h_x = \frac{1}{n_x-1}$ and $h_y = \frac{1}{n_y-1}$. Thus, (4) leads to linear systems of size n_x and (5) leads to linear systems of size n_y. The total number of unknowns in the discrete problem is $600\, n_x\, n_y$.

To solve the problem, a portable parallel code was designed and implemented in C, while the parallelization has been facilitated using the MPI library [8,10]. We use the LAPACK subroutines DPTTRF and DPTTS2 (see [1]) for solving tridiagonal systems in equations (4), (5), and (6) for the unknowns corresponding to the internal nodes of each sub-domain. The same subroutines are used to solve the tridiagonal systems with the Schur complement.

The parallel code has been tested on three computer systems: Galera, located in the Centrum Informatyczne TASK; 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. 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 1 shows the results collected on the Galera. It is a Linux cluster with 336 nodes, and two Intel Xeon quad core processors per node. Each processor runs at 2.33 GHz. Processors within each node share 8, 16, or 32 GB of memory, while nodes are interconnected with a high-speed InfiniBand network (see also `http://www.task.gda.pl/kdm/sprzet/Galera`). Here, we used an Intel C compiler, and compiled the code with the option "-O3".

Table 2 shows the results collected on the Sooner, a quad core Linux cluster (see `http://www.oscer.ou.edu/resources.php`). It has 486 Dell PowerEdge 1950 III nodes, and two quad core processors (Dell Pentium4 Xeon E5405; running at 2 GHz, sharing 16 GB of memory) per node. Nodes are interconnected with a high-speed InfiniBand network. We have used an Intel C compiler, and compiled the code with the following options: "-O3 -march=core2 -mtune=core2".

The results in each column of Tables 1 and 2 are obtained for an equal number of unknowns per core. For large discrete problems the execution time is much

Table 1. Execution time on Galera

c	$n_x n_y/c$										
	10^4	$2\ 10^4$	$4\ 10^4$	$8\ 10^4$	$1.6\ 10^5$	$3.2\ 10^5$	$6.4\ 10^5$	$1.28\ 10^6$	$2.56\ 10^6$	$5.12\ 10^6$	$1.024\ 10^7$
1	0.38	0.67	1.55	3.59	9.16	24.06	53.32	109.73	228.37	508.85	1204.13
2	0.38	0.67	1.51	3.97	10.73	26.67	57.74	120.03	245.60	572.72	1399.29
4	0.35	0.73	1.77	5.17	14.22	34.93	77.91	160.31	331.14	772.65	2171.64
8	0.36	0.84	3.29	9.56	24.05	54.78	113.65	232.26	517.02	1416.69	3606.02
16	0.40	0.93	3.44	9.61	24.18	54.13	114.15	237.16	526.47	1408.33	3680.95
32	0.48	1.03	3.48	9.90	24.48	55.41	114.57	233.91	530.87	1452.24	3664.30
64	0.74	1.22	3.91	10.19	25.26	55.88	117.21	243.30	550.54	1454.70	3826.07

Table 2. Execution time on Sooner

c	$n_x n_y/c$										
	10^4	$2\ 10^4$	$4\ 10^4$	$8\ 10^4$	$1.6\ 10^5$	$3.2\ 10^5$	$6.4\ 10^5$	$1.28\ 10^6$	$2.56\ 10^6$	$5.12\ 10^6$	$1.024\ 10^7$
1	0.58	1.38	2.78	5.72	12.52	27.56	59.60	120.33	248.72	511.40	1196.21
2	0.59	1.37	2.76	5.83	12.55	28.73	61.20	126.34	255.50	543.37	1268.61
4	0.58	1.38	2.77	5.88	13.57	32.42	72.43	147.28	301.04	628.10	1636.03
8	0.61	1.42	3.32	9.02	23.02	53.37	109.13	220.54	455.72	1226.75	3308.30
16	0.63	1.44	3.35	9.02	23.01	52.26	109.56	219.52	456.44	1213.96	3352.22
32	0.67	1.51	3.45	9.21	23.21	54.49	110.22	222.13	457.92	1235.84	3454.01
64	0.73	1.57	3.61	9.34	23.55	53.28	111.74	222.56	463.14	1256.47	3499.66
128	0.85	1.85	3.96	10.21	24.52	56.61	114.53	235.95	471.35	1283.83	3507.78
256	0.98	1.88	4.13	10.01	25.07	55.00	116.02	227.57	476.61	1288.46	3580.68
512	1.36	2.39	5.15	12.99	27.61	63.71	126.36	250.11			

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 of one processor. Furthermore, the execution time for solving one and the same discrete problem decrease when increasing the number of cores, which shows that the communication in our parallel algorithm is mainly local.

The somehow slower performance using 8 cores is clearly visible. The same effect was observed during our previous work, see [7]. There are some factors which could play role for the slower performance using all processors of a single node. Generally, they are a consequence of limitations of 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 a single core within such pair (this core can monopolize use of the L2 cache). Conversely, this leads to a somehow decreased speedups when all cores are used. For the memory intensive programs, these factors can play a crucial role for the performance.

Comparing the performance of the Galera and the Sooner we can observe that times on Sooner are shorter across the board. This is somewhat surprising, as

Table 3. Execution time on IBM Blue Gene/P

c	$n_x n_y / c$										
	10^4	$2\ 10^4$	$4\ 10^4$	$8\ 10^4$	$1.6\ 10^5$	$3.2\ 10^5$	$6.4\ 10^5$	$1.28\ 10^6$	$2.56\ 10^6$	$5.12\ 10^6$	$1.024\ 10^7$
1	5.79	12.33	24.51	49.02	103.58	210.81	431.43	877.01	1764.68	3586.12	7416.03
2	5.96	11.84	24.93	49.89	105.23	214.55	437.71	880.14	1793.94	3604.56	7526.88
4	6.16	13.01	25.68	51.34	108.00	219.95	450.15	913.40	1839.21	3742.63	7706.22
8	6.34	12.48	26.21	52.59	109.85	223.77	455.95	917.29	1865.41	3764.16	7813.90
16	6.61	13.80	27.31	54.38	113.83	230.67	471.25	959.19	1930.85	3908.36	8049.69
32	6.71	13.24	27.45	54.92	114.41	232.50	473.96	952.89	1931.31	3882.43	8059.08
64	6.84	14.19	27.71	55.15	115.56	233.56	476.84	964.14	1935.43	3925.14	8070.35
128	7.04	13.72	28.15	56.20	116.68	236.12	478.39	962.12	1944.61	3915.91	8117.98
256	7.17	14.69	28.34	56.44	117.87	237.57	482.76	978.74	1959.50	3980.72	8183.79
512	7.55	14.59	29.10	58.08	119.14	241.37	486.01	972.81	1971.07	3958.97	8205.10
1024	7.91	15.70	29.78	58.66	120.68	242.69	488.99	987.21	1980.31	4003.01	8216.88
2048	8.81	16.71	31.35	62.83	124.67	251.24	501.47	1027.31	2028.63	4200.09	
4096	9.86	18.31	33.81	65.22	130.91	268.78	559.89	1163.18	2443.63		

Galera is much newer and has more powerful processors. We plan to investigate this peculiarity in the near future.

Table 3 presents execution time on the IBM Blue Gene/P machine at the Bulgarian Supercomputing Center (see also `http://www.scc.acad.bg/`). It consists of 2048 compute nodes with quad core PowerPC 450 processors (running at 850 MHz). Each node has 2 GB of RAM. 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. We have used the IBM XL C compiler and compiled the code with the following options: "-O5 -qstrict -qarch=450d -qtune=450".

We observed that using 2 or 4 cores per processor leads to slower execution time, e.g. the execution time for $n_x = n_y = 6400$, $c = 512$ is 58.08 seconds using 512 nodes, 58.83 seconds using 256 nodes, and 60.34 seconds using 128 nodes. This fact shows that using the MPI communication functions, the communication between processors is faster than the communication between cores of one processor. In order to get better parallel performance we plan to develop a mixed MPI/OpenMP code and to use the nodes of the Blue Gene supercomputer in the SMP mode with 4 OpenMP processes per node. This code will also allow us to run efficiently on the upcoming machines with 16-core AMD processors (and future computers with ever increasing number of cores per processor). Note that, for the time being, in our work we omit all issues concerning GPU-based parallelization.

To round up the performance analysis, the speed-up obtained on Galera is reported in Table 4 and the parallel efficiency is shown in Table 5, while the speed-up on Sooner — in Table 6 and the parallel efficiency — in Table 7. Finally, the speed-up on the IBM Blue Gene/P — in Table 8, and the parallel efficiency — in Table 9. In each case, when increasing the number of cores of the two clusters, the parallel efficiency decreases on 8 cores and after that it increases to 100%. Moreover, a super-linear speed-up is observed in multiple cases. The main reasons

Table 4. Speed-up on Galera

n_x	n_y	c					
		2	4	8	16	32	64
800	800	2.00	3.75	5.58	15.49	51.64	71.86
800	1600	1.90	3.14	4.56	11.42	31.49	90.07
1600	1600	1.90	2.93	4.17	9.45	23.08	58.40
1600	3200	2.07	3.17	4.48	9.40	20.78	49.95
3200	3200	2.10	3.64	5.18	10.55	21.73	47.66
3200	6400	2.05	3.72	5.56	12.12	25.09	51.44
6400	6400	1.90	3.84	5.88	15.82	35.61	71.06
6400	12800	1.60	2.49	3.76	9.63	25.55	55.74
12800	12800	2.12	2.90	3.98	10.08	25.55	67.39

Table 5. Parallel efficiency on Galera

n_x	n_y	c					
		2	4	8	16	32	64
800	800	1.000	0.938	0.697	0.968	1.614	1.123
800	1600	0.950	0.785	0.570	0.714	0.984	1.407
1600	1600	0.951	0.733	0.521	0.590	0.721	0.913
1600	3200	1.036	0.794	0.560	0.588	0.649	0.780
3200	3200	1.051	0.909	0.648	0.659	0.679	0.745
3200	6400	1.027	0.930	0.695	0.758	0.784	0.804
6400	6400	0.949	0.959	0.735	0.989	1.113	1.110
6400	12800	0.800	0.622	0.470	0.602	0.798	0.871
12800	12800	1.060	0.726	0.498	0.630	0.798	1.053

for this fact can be related to splitting the entire problem into subproblems which helps the memory management. In particular, it allows for better usage of cache memories of individual parallel processors. As expected, the parallel efficiency on the IBM Blue Gene/P improves with the size of the discrete problems. The efficiency on 1024 cores increases from 57% for the smallest problems to 94% for the largest problems.

Execution time on the Blue Gene/P is substantially larger than that on the Sooner and the Galera, but in some cases the parallel efficiency obtained on the supercomputer is better. For example, the execution time on single core on Sooner is seven times faster than on the Blue Gene/P, in comparison with four times faster performance on 256 cores.

The decomposition of the computational domain in sub-domains is important for the parallel performance of the studied algorithm. Table 10 shows the execution time for the problem with $n_x = n_y = 3200$ on 128 cores using different number of sub-domains in each space direction.

Finally, computing time on both parallel systems is shown in Fig. 1 and the obtained speed-up is shown in Fig. 2.

Table 6. Speed-up on Sooner

n_x	n_y	c								
		2	4	8	16	32	64	128	256	512
800	800	2.07	4.39	6.61	17.79	39.57	81.17	110.21	188.19	134.30
800	1600	1.97	3.71	5.23	13.34	34.93	76.64	142.35	228.36	237.10
1600	1600	1.97	3.43	4.66	10.81	27.00	68.89	134.56	254.61	328.64
1600	3200	2.00	3.47	4.69	9.79	22.04	54.74	129.16	272.71	375.11
3200	3200	2.20	3.97	5.42	10.92	21.95	50.79	117.21	289.73	500.82
3200	6400	2.15	4.35	6.00	12.45	24.80	51.31	111.49	273.13	530.73
6400	6400	1.98	4.01	5.35	14.38	29.55	58.93	115.96	261.83	505.35
6400	12800	1.88	3.21	4.30	11.71	31.03	63.84	124.07	258.34	514.72

Table 7. Parallel efficiency on Sooner

n_x	n_y	c								
		2	4	8	16	32	64	128	256	512
800	800	1.037	1.098	0.826	1.112	1.237	1.268	0.861	0.735	0.262
800	1600	0.983	0.928	0.653	0.834	1.091	1.198	1.112	0.892	0.463
1600	1600	0.984	0.858	0.583	0.676	0.844	1.076	1.051	0.995	0.642
1600	3200	1.001	0.868	0.586	0.612	0.689	0.855	1.009	1.065	0.733
3200	3200	1.101	0.993	0.678	0.682	0.686	0.794	0.916	1.132	0.978
3200	6400	1.077	1.088	0.750	0.778	0.775	0.802	0.871	1.067	1.037
6400	6400	0.988	1.003	0.669	0.899	0.924	0.921	0.906	1.023	0.987
6400	12800	0.938	0.802	0.537	0.732	0.970	0.998	0.969	1.009	1.005

Table 8. Speed-up on IBM Blue Gene/P

n_x	n_y	c											
		2	4	8	16	32	64	128	256	512	1024	2048	4096
800	800	2.01	3.99	8.20	15.80	32.59	63.08	114.74	216.58	330.83	585.7	631.5	782.5
800	1600	2.00	3.99	7.98	16.13	31.95	61.81	124.52	226.79	401.70	655.0	944.8	1177.6
1600	1600	2.00	3.92	7.89	15.50	32.13	63.68	128.66	246.03	424.30	745.3	1002.3	1541.7
1600	3200	2.00	3.93	7.87	15.55	31.34	65.02	127.38	244.20	474.86	812.0	1290.0	1902.5
3200	3200	2.06	4.03	8.08	15.74	31.90	64.17	131.95	261.66	508.46	937.1	1411.7	2274.2

Table 9. Parallel efficiency on IBM Blue Gene/P

n_x	n_y	c											
		2	4	8	16	32	64	128	256	512	1024	2048	4096
800	800	1.005	0.999	1.025	0.987	1.018	0.986	0.896	0.846	0.646	0.572	0.308	0.191
800	1600	1.002	0.997	0.998	1.008	0.998	0.966	0.973	0.886	0.785	0.640	0.461	0.287
1600	1600	1.002	0.980	0.986	0.969	1.004	0.995	1.005	0.961	0.829	0.728	0.489	0.376
1600	3200	1.000	0.982	0.983	0.972	0.979	1.016	0.995	0.954	0.927	0.793	0.630	0.464
3200	3200	1.029	1.008	1.011	0.984	0.997	1.003	1.031	1.022	0.993	0.915	0.689	0.555

Table 10. Execution time on 128 cores

machine	sub-domains			
	8×16	4×32	2×64	1×128
Sooner	10.30	13.14	16.80	84.90
IBM Blue Gene/P	56.20	60.17	74.12	170.46

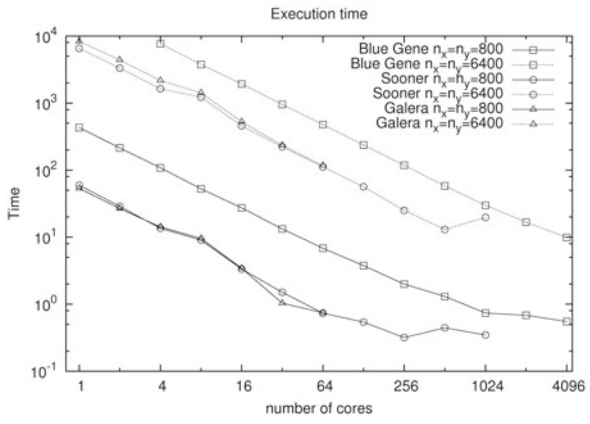

Fig. 1. Execution time for $n_x = n_y = 800, 6400$

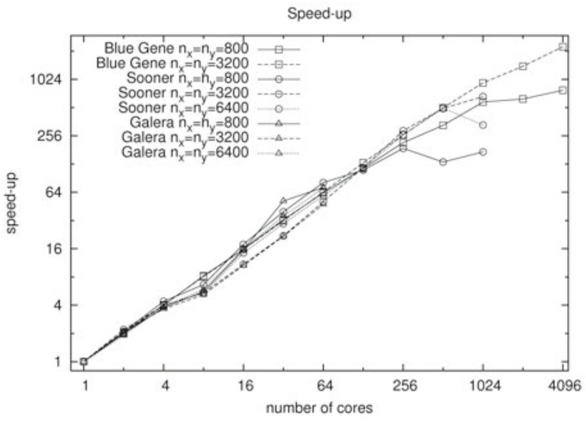

Fig. 2. Speed-up for $n_x = n_y = 800, 3200, 6400$

5 Conclusions and Future Work

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

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.

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 DCVP 02/1 and DO02-147 from the Bulgarian NSF. Work presented here is a part of the Poland-Bulgaria collaborative grant "Parallel and distributed computing practices".

References

1. Anderson, E., Bai, Z., Bischof, C., Blackford, S., Demmel, J., Dongarra, J., Croz, J.D., Greenbaum, A., Hammarling, S., McKenney, A., Sorensen, D.: LAPACK Users' Guide, 3rd edn. SIAM, Philadelphia (1999)
2. Chorin, A.J.: Numerical solution of the Navier-Stokes equations. Math. Comp. 22, 745–762 (1968)
3. Guermond, J.L., Minev, P.: A new class of fractional step techniques for the incompressible Navier-Stokes equations using direction splitting. Comptes Rendus Mathematique 348(9-10), 581–585 (2010)
4. Guermond, J.L., Minev, P., Shen, J.: An overview of projection methods for incompressible flows. Comput. Methods Appl. Mech. Engrg. 195, 6011–6054 (2006)
5. Guermond, J.L., Salgado, A.: A fractional step method based on a pressure poisson equation for incompressible flows with variable density. Comptes Rendus Mathematique 346(15-16), 913–918 (2008)
6. Guermond, J.L., Salgado, A.: A splitting method for incompressible flows with variable density based on a pressure Poisson equation. Journal of Computational Physics 228(8), 2834–2846 (2009)
7. Lirkov, I., Vutov, Y., Paprzycki, M., Ganzha, M.: Parallel Performance Evaluation of MIC(0) Preconditioning Algorithm for Voxel μFE Simulation. In: Wyrzykowski, R., Dongarra, J., Karczewski, K., Waśniewski, J. (eds.) PPAM 2009, Part II. LNCS, vol. 6068, pp. 135–144. Springer, Heidelberg (2010)
8. Snir, M., Otto, S., Huss-Lederman, S., Walker, D., Dongarra, J.: MPI: The Complete Reference. Scientific and engineering computation series. The MIT Press, Cambridge (1997); second printing
9. Temam, R.: Sur l'approximation de la solution des équations de Navier-Stokes par la méthode des pas fractionnaires. Arch. Rat. Mech. Anal. 33, 377–385 (1969)
10. Walker, D., Dongarra, J.: MPI: a standard Message Passing Interface. Supercomputer 63, 56–68 (1996)