Preprints of Position Papers of the Federated Conference on
Computer Science and Information Systems pp. 27–32

Performance Evaluation of MPI/OpenMP Algorithm for 3D Time Dependent Problems

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, approached in terms of velocity and pressure. In a parallel algorithm, based on a novel 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. 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 OpenMP and tested on modern computer systems. The performed tests illustrate the parallel efficiency, and the scalability, of the direction-splitting based algorithm.**

I. 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 [4], [12] and they have been used in Computational Fluid Dynamics (CFD) for the last 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 [7] for an overview. In terms of computational efficiency, projection algorithms are far superior than 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 [5], [6] 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 and a gradient part. Departure from the projection paradigm has been proved to be very efficient for solving variable density flows [8], [9]. 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 has been proved to be stable and convergent (see [5], [6]). The parallel performance of the direction splitting algorithm was evaluated (in [10]) on three different parallel architectures when solving 2D problems . The aim of this paper is to study the parallel performance of the algorithm for solving of three dimensional problems.

II. Stokes Equation

Let us start from the mathematical formulation of the problem. 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 $\mathbf{u}$ 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. Let us also assume that the time interval $[0,T]$ was discretized on a uniform mesh and τ was the time step.

III. Parallel Alternating Directions Algorithm

As what concerns the solution method, Guermond and Minev (in [5], [6]) introduced a novel fractional time stepping technique for solving the incompressible Navier-Stokes equations. This technique is based on a direction splitting strategy. They used a singular perturbation of the Stokes equation with a perturbation parameter τ. In this way, the standard Poisson problem was replaced by series of one-dimensional second-order boundary value problems.

A. 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

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

- *Pressure predictor*. Denoting by p_0 the pressure field at $t = 0$, the algorithm is initialized by setting $p^{-\frac{1}{2}} = p^{-\frac{3}{2}} = p_0$. 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 $\mathbf{u}^0 = \mathbf{u}_0$, 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* 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:

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

- *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 $\chi \in (0, \frac{1}{2}]$. The convergence tests reported in [5], [6] confirm that the rotational form of the incremental version of the method is second-order in time for the L^2-norm of the velocity field.

IV. Experimental Results

Lest us recall, that our aim is to evaluate the performance of just described method for 3D problems. Therefore, we have solved the problem (1) in the domain $\Omega = (0, 1)^3$, for $t \in [0, 2]$, with Dirichlet boundary conditions. The discretization in time was done with time step 10^{-2}. The parameter in the pressure update sub-step was $\chi = \frac{1}{2}$, and the kinematic viscosity was $\nu = 10^{-3}$. The second order central differences were used for the discretization in space on a rectangular mesh with 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, the equation (4) resulted in linear systems of size n_y, and the 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. The parallelization was based on the MPI and OpenMP standards [2], [3], [11], [13]. 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 the following three systems: (1) a cluster computer *Galera*, located in the Polish Centrum Informatyczne TASK, (2) on a cluster computer system *HPCG*, located in the Institute of Information and Communication Technologies, and (3) on the IBM Blue Gene/P machine, at the Bulgarian Supercomputing Center. Table I summarizes the information about compilers and libraries used on the three computers. In our experiments, times have been collected using the MPI provided timer, and we report the best results from multiple runs. However, in all our experiments, recorded times did not differ by more than 5%. In the following tables, we report the elapsed (wall-clock) time T_p in seconds using m MPI processes and k OpenMP processes, where $p = m \times k$, and the parallel speed-up $S_p = T_1/T_p$.

Tables II and III show 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 -openmp". For solving the tridiagonal systems of equations using LAPACK subroutines we linked our code to multi-threaded layer Intel Math Kernel Library (see http://software.intel.com/en-us/articles/intel-mkl/).

Note that, the discrete problem with $n_x = n_y = 400$, $n_z = 800$ requires 22 GB of memory. That is why, for larger problems, we could not run the code for small number of nodes. As a matter of fact, the largest considered problem required a minimum of 64 nodes.

Table IV contains the speed-up obtained on Galera. For the reasons described above, it was impossible to calculate the standard speed-up (time on a single core vs. time on P cores). That is why we report the speed-up on Galera only for problems with $n_x, n_y = 100, 200, 400$, $n_z = 100, 200, 400, 800$. However it is easy to calculate the "normalized" speed-up even for the largest problem. For instance, when comparing execution time on 128 and 256 cores with that on 64 cores

TABLE I
COMPILERS AND LIBRARIES ON THE THREE COMPUTER SYSTEMS

	Galera	HPCG	IBM Blue Gene/P
Compiler	Intel C Compiler 12.1.0	Intel C Compiler 12.1.0	IBM XL C Compiler 9.0
MPI	OpenMPI 1.4.3	Intel MPI 4.0.3.008	MPICH2
LAPACK	Intel Math Kernel Library 10.0	Intel Math Kernel Library 10.0	Engineering and Scientific Subroutine Library 5.1

TABLE II
EXECUTION TIME FOR SOLVING OF 3D PROBLEM ON ONE NODE OF GALERA.

n_x	n_y	n_z	k			
			1	2	4	8
100	100	100	179.13	108.92	82.91	78.31
100	100	200	386.53	233.35	176.05	160.64
100	200	200	848.59	506.58	377.94	339.81
200	200	200	1802.57	1057.07	788.10	685.73
200	200	400	3695.46	2157.89	1593.42	1374.58
200	400	400	7618.41	4440.56	3272.34	2765.14
400	400	400	15839.00	9120.79	6545.47	5473.50
400	400	800	32763.20	18849.10	13582.80	11524.80

TABLE III
EXECUTION TIME FOR SOLVING OF 3D PROBLEM ON MANY NODES OF GALERA.

n_x	n_y	n_z	nodes							
			2	4	8	16	32	64	128	256
			k=8							
100	100	100	42.3	21.1	10.8	5.8	3.2	2.6	2.2	1.8
100	100	200	83.2	42.2	21.0	11.1	6.1	3.3	2.4	1.7
100	200	200	174.5	86.0	43.3	22.8	11.6	6.2	4.6	2.6
200	200	200	343.2	166.3	81.0	44.3	22.3	11.5	7.1	4.1
200	200	400	692.3	337.8	165.3	89.1	45.5	22.8	13.5	7.8
200	400	400	1410.2	706.4	347.7	187.0	90.7	46.3	29.2	14.2
400	400	400	2795.8	1429.7	703.8	357.8	174.2	85.5	51.6	25.9
400	400	800	5470.0	2758.6	1415.7	723.1	359.2	177.1	108.0	56.5
400	800	800	11613.7	5594.8	2797.6	1466.9	741.7	372.7	225.5	113.4
800	800	800		11926.1	5669.0	2802.2	1485.4	753.8	404.5	207.9
800	800	1600			12028.3	5660.6	2874.8	1515.1	817.3	428.5
800	1600	1600				12006.2	5719.0	2868.4	1653.2	853.2
1600	1600	1600					12151.7	5757.7	3022.9	1646.3
1600	1600	3200						12231.2	6076.1	3306.1

one can say that such speed-up is 2.015 and 3.699 respectively. Here, the slight superlinear speed-up can be attributed to the well-known effect caused by halving the problem size and improving memory management. These results indicate also that the communication in our parallel algorithm is mainly local. Specifically, if halving the problem leads to superlinear speed-up, it means that communication between nodes is not as important as memory contention within a node.

In our previous work [10] we observed slower performance of the MPI code for solving 2D problem on a single node, while using all available cores. Therefore, we have used OpenMP for solving of 3D problem on a single node. However, the slower performance using 8 OpenMP processes is clearly visible. It can be stipulated that this effect is a result of limitations of the memory subsystems, and their hierarchical organization. One of them might be the limited bandwidth of the main memory bus. This causes processors to "starve" for data, thus, decreasing the overall performance. This stipulation is consistent with the above observation based on superlinearity of speed-up for largest problems.

Tables V and VI show the results collected on the HPCG cluster. HP Cluster Platform Express 7000 consists of 36 blades BL 280c, dual Intel Xeon X5560 processors (total of 576 cores). Each processor runs at 2.8 GHz. Processors within each blade share 24 GB RAM, while nodes are interconnected with non-blocking DDR Interconnection via a Voltaire Grid director 2004 with latency 2.5 μs and bandwidth 20 Gbps (see also http://www.grid.bas.bg/hpcg/). Again, we used an Intel C compiler, and compiled the code with the option "-O3 -openmp". For solving the tridiagonal systems of equations using LAPACK subroutines, we linked our code to the multi-threaded layer Intel Math Kernel Library. The execution time presented in Table VI shows that for solving small systems using the multi-threaded library the best results are obtained when using 8 OpenMP processes per blade. However, hyper-threading (using 2 OpenMP processes per physical core) helps only when solving discrete problems with more than 16×10^6 grid points.

To provide an analytical view on performance, the speed-up obtained on the HPCG is reported in Table VII. Here, let us

TABLE IV
SPEED-UP ON GALERA.

n_x	n_y	n_z	$m \times k$										
			2	4	8	16	32	64	128	256	512	1024	2048
100	100	100	1.64	2.16	2.29	4.23	8.50	16.64	30.66	55.35	69.74	83.10	99.59
100	100	200	1.66	2.20	2.41	4.64	9.17	18.43	34.65	63.66	116.14	158.43	233.33
100	200	200	1.68	2.25	2.50	4.86	9.87	19.61	37.22	72.81	136.44	185.16	327.56
200	200	200	1.71	2.29	2.63	5.25	10.84	22.24	40.72	80.80	156.55	254.74	439.52
200	200	400	1.71	2.32	2.69	5.34	10.94	22.35	41.45	81.29	162.28	274.49	473.96
200	400	400	1.72	2.33	2.76	5.40	10.78	21.91	40.73	84.03	164.69	260.76	537.08
400	400	400	1.74	2.42	2.89	5.67	11.08	22.51	44.27	90.91	185.23	306.84	611.41
400	400	800	1.74	2.41	2.84	5.99	11.88	23.14	45.31	91.21	184.96	303.23	579.91

TABLE V
EXECUTION TIME FOR SOLVING OF 3D PROBLEM ON ONE NODE OF HPCG.

n_x	n_y	n_z	k				
			1	2	4	8	16
100	100	100	86.14	48.47	30.23	26.26	27.42
100	100	200	190.45	98.81	63.44	52.07	54.51
100	200	200	397.90	224.73	137.70	110.45	109.30
200	200	200	899.24	456.19	279.53	207.31	219.29
200	200	400	1891.51	945.52	598.95	447.60	449.36
200	400	400	3806.96	1987.19	1225.93	889.76	843.52
400	400	400	6519.28	4162.41	2507.75	1900.73	1722.60
400	400	800	14343.40	8174.26	5048.87	3840.93	3648.71

recall that (similarly to the case of Galera) we were not able to solve largest problems for small number of nodes. Specifically, the largest discrete problem that we could solve on a single blade had 128×10^6 grid points. Furthermore, problems larger than these reported above did not fit into the available memory at all. However, we can see that the normalized speed-up for the largest solvable problem was 1.94.

Tables VIII and IX present times collected on the IBM Blue Gene/P, at the Bulgarian Supercomputing Center. 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 (for more details, see http://www.scc.acad.bg/). We have used the IBM XL C compiler and compiled the code with the following options: "-O5 -qstrict -qarch=450d -qtune=450 -qsmp=omp". For solving the tridiagonal systems using LAPACK subroutines, we linked our code to the multi-threaded Engineering and Scientific Subroutine Library (ESSL) (see http://www-03.ibm.com/systems/software/essl/index.html). Note that, the memory of a node of the IBM supercomputer is substantially smaller than that on the clusters and was not sufficient for solving 3D problem larger than $n_x = n_y = n_z = 200$. We solved these problems on two and more nodes. Table X shows the speed-up obtained on the Blue Gene. Because of smaller memory per node we calculated the actual speed-up only for $n_x, n_y, n_z = 100, 200$. Interestingly, a super-linear speed-up was observed using up to 128 processes. Here, note that individual processors on supercomputer are slower than these on clusters while the communication is faster (due to special networking used in the Blue Gene). Furthermore, due to the memory scarcity the above mentioned problem halving effect is particularly pronounced. Separately, note that the normalized speed-up for the largest problem we were able to solve was 1.98.

Finally, we represent (in Figure 1) performance of the three computer systems. Here, we represent results for discrete problem sizes that were obtainable across all machines. Here, it can be seen that, for up to 128 processes, the HPCG cluster is the most efficient for our problem. For 256 processes (using 16 OpenMP processes), for the smallest problems, we observe a performance drop of the HPCG. Here, the Galera becomes the performance leader. We observed a performance drop on both clusters when we use 8 OpenMP processes on one node. This happens regardless of the fact that we have used a hybrid OpenMP+MPI approach, which should have resulted in optimal performance of computers with mixed shared-distributed memory.

V. CONCLUDING REMARKS

In this paper we have investigated performance of a novel approach to the solution of 3D Navier-Stokes equations on three parallel computers. We have found out that the mixed OpenMP+MPI approach, used to implement the proposed algorithm, works well on shared-distributed memory computers. Furthermore, the Blue Gene/P machine is not well balanced, with memory that is relatively too small, networking that is relatively too fast, while processors being relatively too slow. The HPCG and Galera machines exhibits unusual behavior when all of its processors and cores on one node are used. The Galera seems to be best balanced (even though not the fastest of the three). It was also the Galera that allowed us to solve the largest problems.

ACKNOWLEDGMENTS

Computer time grants from the Bulgarian Supercomputing Center and the TASK computing center in Gdansk, Poland are

TABLE VI
EXECUTION TIME FOR SOLVING OF 3D PROBLEM ON MANY NODES OF HPCG.

			nodes			
			2	4	8	16
n_x	n_y	n_z	k=8			
100	100	100	15.10	7.40	4.38	2.56
100	100	200	28.61	17.06	8.14	5.05
100	200	200	61.72	35.38	17.58	9.45
200	200	200	118.89	68.54	35.96	18.63
200	200	400	246.84	120.25	70.53	37.90
200	400	400	500.00	259.89	152.23	83.25
400	400	400	914.44	505.90	248.07	146.35
400	400	800	2109.01	985.69	529.95	321.91
400	800	800	4271.35	2074.35	1042.33	626.71
800	800	800		4463.87	2204.67	1167.19
800	800	1600			4495.82	2378.40
n_x	n_y	n_z	k=16			
100	100	100	15.45	9.08	5.73	3.92
100	100	200	28.30	18.56	10.47	7.09
100	200	200	57.77	34.18	19.10	12.49
200	200	200	115.16	61.55	35.88	21.00
200	200	400	229.05	122.69	72.39	39.07
200	400	400	439.35	237.16	132.74	71.45
400	400	400	894.70	462.40	242.39	137.17
400	400	800	1744.18	886.36	485.19	261.39
400	800	800	3733.36	1928.69	1014.47	524.18
800	800	800		4138.85	2209.53	1000.43
800	800	1600			4169.04	2138.83

TABLE VII
SPEED-UP ON HPCG.

n_x	n_y	n_z	$m \times k$							
			2	4	8	16	32	64	128	256
100	100	100	1.78	2.85	3.28	5.71	11.64	19.69	33.63	21.96
100	100	200	1.93	3.00	3.66	6.66	11.16	23.40	37.71	26.85
100	200	200	1.77	2.89	3.60	6.45	11.25	22.64	42.10	31.86
200	200	200	1.97	3.22	4.34	7.56	13.12	25.01	48.27	42.83
200	200	400	2.00	3.16	4.23	7.66	15.73	26.82	49.90	48.41
200	400	400	1.92	3.11	4.28	7.61	14.65	25.01	45.73	53.28
400	400	400	1.57	2.60	3.43	7.13	12.89	26.28	44.54	47.53
400	400	800	1.75	2.84	3.73	6.80	14.55	27.07	44.56	54.87

TABLE VIII
EXECUTION TIME FOR SOLVING OF 3D PROBLEM ON ONE NODE OF IBM BLUE GENE/P.

n_x	n_y	n_z	k		
			1	2	4
100	100	100	882.72	516.00	306.88
100	100	200	1815.72	1040.66	632.95
100	200	200	3782.91	2153.78	1291.85
200	200	200	7685.65	4343.88	2710.08

TABLE IX
EXECUTION TIME FOR SOLVING OF 3D PROBLEM ON MANY NODES OF IBM BLUE GENE/P.

n_x	n_y	n_z	nodes								
			2	4	8	16	32	64	128	256	512
			k=4								
100	100	100	146.8	74.9	38.3	20.9	10.9	6.2	4.5	3.0	1.9
100	100	200	309.0	157.3	77.0	41.0	21.1	11.2	7.1	4.9	3.1
100	200	200	626.8	317.8	159.9	79.3	39.8	21.3	13.2	7.9	5.1
200	200	200	1315.4	645.1	323.4	164.2	79.1	39.5	24.9	14.7	8.5
200	200	400	2645.4	1325.9	647.5	337.9	165.2	80.2	42.8	26.0	15.0
200	400	400		2656.2	1341.7	657.7	330.0	159.4	87.5	46.2	27.4
400	400	400			2719.6	1383.3	677.9	334.4	176.2	96.3	48.7
400	400	800				2807.4	1387.8	681.4	348.8	184.2	95.7
400	800	800					2753.6	1379.3	692.2	362.7	184.6
800	800	800						2853.6	1457.6	748.5	374.5
800	800	1600							2914.0	1524.5	749.6
800	1600	1600								2980.3	1498.6

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

n_x	n_y	n_z	$m \times k$										
			2	4	8	16	32	64	128	256	512	1024	2048
100	100	100	2.11	4.43	8.78	17.02	33.73	64.93	111.54	191.55	351.96	421.32	461.68
100	100	200	2.04	4.45	8.96	17.61	34.78	68.73	125.50	218.75	407.52	517.90	593.41
100	200	200	2.08	4.54	9.16	18.38	36.92	72.02	134.22	246.66	452.54	616.02	746.84
200	200	200	2.00	4.18	8.41	17.23	35.15	75.19	138.63	259.63	479.53	770.25	1006.01

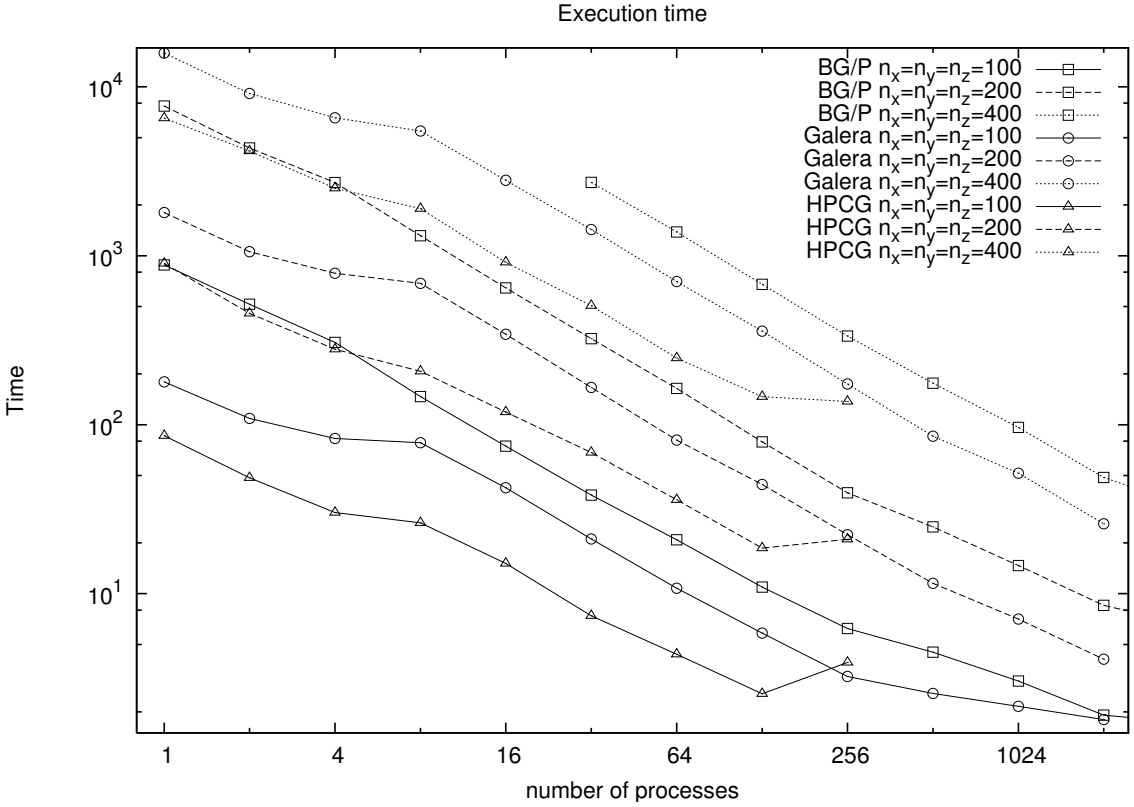

Fig. 1. Execution time for $n_x = n_y = n_z = 100, 200, 400$.

kindly acknowledged. This research was partially supported by grants DCVP 02/1 and I01/5 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. SIAM, Philadelphia, third edn. (1999)

[2] Chandra, R., Menon, R., Dagum, L., Kohr, D., Maydan, D., McDonald, J.: Parallel programming in OpenMP. Morgan Kaufmann (2000)

[3] Chapman, B., Jost, G., Van Der Pas, R.: Using OpenMP: portable shared memory parallel programming, vol. 10. MIT press (2008)

[4] Chorin, A.J.: Numerical solution of the Navier-Stokes equations. Math. Comp. 22, 745–762 (1968)

[5] 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)

[6] Guermond, J.L., Minev, P.: A new class of massively parallel direction splitting for the incompressible navier–stokes equations. Computer Methods in Applied Mechanics and Engineering 200(23), 2083–2093 (2011)

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

[8] 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)

[9] 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)

[10] Lirkov, I., Paprzycki, M., Ganzha, M.: Performance analysis of parallel alternating directions algorithm for time dependent problems. In: Wyrzykowski, R., Dongarra, J., Karczewski, K., Waśniewski, J. (eds.) 9th international conference on Parallel Processing and Applied Mathematics, PPAM 2011, Part I. Lecture notes in computer science, vol. 7203, pp. 173–182. Springer (2012)

[11] Snir, M., Otto, S., Huss-Lederman, S., Walker, D., Dongarra, J.: MPI: The Complete Reference. Scientific and engineering computation series, The MIT Press, Cambridge, Massachusetts (1997), second printing

[12] 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)

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