Comparative analysis of high performance solvers for solving Stokes equation

Maria Ganzha*, Ivan Lirkov† and Marcin Paprzycki*

*Systems Research Institute, Polish Academy of Science, ul. Newelska 6, 01-447 Warsaw, Poland
†Institute of Information and Communication Technologies, Bulgarian Academy of Sciences
Acad. G. Bonchev, bl. 25A, 1113 Sofia, Bulgaria

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. A parallel algorithm based on a direction splitting approach is implemented. We are targeting the massively parallel computer as well as clusters of many-core nodes. The implementation is tested on the IBM Blue Gene/P supercomputer and two Linux clusters. We compared the results from the direction splitting algorithm with the results from Finite Element software package for solving of Stokes equation.

Keywords: Navier-Stokes, time splitting, ADI, incompressible flows, pressure Poisson equation, parallel algorithm
PACS: 02.60.Cb, 02.60.Lj, 02.70.Bf, 07.05.Tp, 47.10.ad, 47.11.Bc

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 [1, 2] and they have been used in Computational Fluid Dynamics 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 [3] 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 [4, 5] 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 [6, 7]. 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 [4, 5]). The aim of this paper is to experimentally investigate the parallel properties of the algorithm on three distinct parallel systems, for two dimensional problems.

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:

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

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.

PARALLEL ALTERNATING DIRECTIONS ALGORITHM

Guermond and Minev introduced (in [4]) a 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.

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
  $$\frac{\xi^{n+1} - \mathbf{u}^n}{\tau} - \nu\Delta\mathbf{u}^n + \nabla p^{*,n+\frac{1}{2}} = \mathbf{f}^{n+\frac{1}{2}}, \qquad \xi^{n+1}|_{\partial\Omega} = 0, \tag{3}$$
  $$\frac{\eta^{n+1} - \xi^{n+1}}{\tau} - \frac{\nu}{2}\frac{\partial^2(\eta^{n+1} - \mathbf{u}^n)}{\partial x^2} = 0, \qquad \eta^{n+1}|_{\partial\Omega} = 0, \tag{4}$$
  $$\frac{\mathbf{u}^{n+1} - \eta^{n+1}}{\tau} - \frac{\nu}{2}\frac{\partial^2(\mathbf{u}^{n+1} - \mathbf{u}^n)}{\partial y^2} = 0, \qquad \mathbf{u}^{n+1}|_{\partial\Omega} = 0, \tag{5}$$
  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:
  $$\begin{array}{lll} \psi - \psi_{xx} = & -\frac{1}{\tau}\nabla\cdot\mathbf{u}^{n+1}, & \psi_x|_{\partial\Omega} = 0, \\ \phi - \phi_{yy} = & \psi, & \phi_y|_{\partial\Omega} = 0, \end{array} \tag{6}$$
- *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 $\chi \in (0, \frac{1}{2}]$. The convergence tests reported in [4, 5] 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.

TABLE 1. 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

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.

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 second order central differences were used for the discretization in space on a rectangular mesh with mesh sizes $h_x = \frac{1}{n_x-1}$ 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 and OpenMP standards [8, 9, 10, 11]. We use the LAPACK subroutines DPTTRF and DPTTS2 (see [12]) 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, on a cluster computer system HPCG located in the Institute of Information and Communication Technologies, and on the IBM Blue Gene/P machine at the Bulgarian Supercomputing Center. Table 1 summarizes the information about used compilers and libraries on the three computer systems. In our experiments, times have been collected using the MPI provided timer and we report the best results from multiple runs. In the following tables, we report the elapsed time T_p in seconds using m MPI processes and k OpenMP processes, where $p = m \times k$, and the parallel speed-up $S_p = T_1/T_p$.

Table 2 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 -openmp”. For solving the tridiagonal systems of equations using LAPACK subroutines we linked our code to multi-threaded layer Intel Math Kernel Library (MKL, see `http://software.intel.com/en-us/articles/intel-mkl/`). The results obtained with an MPI implementation of the alternating directions algorithm were reported in [13]. We observed slower performance using 8 cores on one node of Galera using MPI code. Now we used OpenMP and multi-threaded layer Intel MKL for execution of the code on one node. We were unpleasantly surprised because the new code has slower performance on 2, 4, and 8 cores, e.g. for $n_x = n_y = 3200$ the execution time of the MPI code on 8 cores is 232 seconds while the execution time of the OpenMP code on 8 cores is 550 seconds.

For solving the problem with $n_x = n_y = 12800$ 18 GB memory is needed. The physical memory on a single node of Galera is not large enough for solving of twice larger discrete problems and we used two and more nodes of the cluster for such problems.

TABLE 2. Execution time for solving of 2D problem on Galera.

$\mathbf{n_x}$	$\mathbf{n_y}$	**processes**											
		1	2	4	8	16	32	64	128	256	512	1024	2048
800	800	47.4	29.1	22.4	19.6	9.2	4.6	2.4	1.3	0.8	0.6	0.5	1.5
800	1600	96.7	58.3	44.5	39.8	19.3	9.4	4.7	2.5	1.4	0.9	0.8	0.6
1600	1600	201.3	119.7	91.7	82.9	38.6	19.7	9.6	4.8	2.6	1.5	1.1	0.9
1600	3200	437.2	263.4	212.9	200.0	79.3	39.5	19.8	9.8	5.0	2.7	1.8	1.2
3200	3200	1070.0	672.6	682.8	550.1	174.0	79.8	39.7	20.3	10.1	5.1	3.0	1.9
3200	6400	2525.6	1754.7	1920.2	1477.6	432.7	178.3	80.5	40.8	20.7	10.2	5.7	3.2
6400	6400	7418.9	4750.9	4007.3	3137.8	993.8	442.9	177.0	85.4	41.5	20.9	10.8	5.9
6400	12800	12650.0	8146.2	5823.4	4956.7	2188.2	1087.3	461.5	205.5	85.9	41.9	21.8	11.3
12800	12800	34804.7	21416.3	14318.1	11272.0	4985.2	2806.5	1111.4	556.6	209.2	86.4	42.7	22.0
12800	25600					10317.4	5035.7	2387.2	1547.4	543.7	211.0	92.3	43.2
25600	25600						11465.1	5126.7	3195.8	1582.4	567.8	214.1	88.6
25600	51200							10507.2	5292.6	3406.1	1596.8	559.3	214.7
51200	51200								11678.2	5220.0	3381.2	1620.5	582.8

TABLE 3. Execution time for solving of 2D problem on HPCG.

$\mathbf{n_x}$	$\mathbf{n_y}$	**processes**							
		1	2	4	8	16	32	64	128
800	800	21.85	12.09	7.72	6.47	3.41	1.60	1.10	0.65
800	1600	46.79	25.27	15.98	13.29	6.94	3.67	2.03	1.13
1600	1600	95.89	50.63	31.53	25.43	13.87	6.65	4.16	2.20
1600	3200	194.19	100.66	63.58	50.56	29.60	13.52	7.96	4.23
3200	3200	400.59	206.77	129.85	106.50	51.77	28.20	14.97	8.25
3200	6400	901.91	470.02	299.95	240.32	116.79	53.55	29.97	17.34
6400	6400	1882.27	1108.73	696.65	562.38	253.67	113.27	71.79	35.34
6400	12800	4277.73	2323.90	1463.49	1113.83	562.59	337.00	153.85	72.42
12800	12800	8068.33	4748.90	3119.29	2761.58	1217.21	593.05	335.91	161.65

Table 3 shows the results collected on the HPCG cluster. HP Cluster Platform Express 7000 enclosures with 36 blades BL 280c, dual Intel Xeon X5560 processors (total 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 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 multi-threaded layer Intel MKL.

Again, the somehow slower performance using 8 cores is clearly visible. 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.

Table 4 presents execution time 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). 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 multi threaded Engineering and Scientific Subroutine Library (ESSL, see `http://www-03.ibm.com/systems/software/essl/index.html`).

Again, the new code has slower performance than the MPI code on 2 and 4 cores. The memory of one node of IBM supercomputer is substantially smaller than on clusters (2 GB vs. 24 or 32 GB) and the largest discrete problem in our experiments which can be solved on one node have $n_x = n_y = 3200$.

The execution time on the three parallel systems is shown in Fig. 1. Because of the slower processors, the execution time obtained on the Blue Gene/P is substantially larger than that on the clusters. At the same time, the parallel efficiency obtained on a large number of nodes on the supercomputer is better. The main reason of this can be related

TABLE 4. Execution time for solving of 2D problem on IBM Blue Gene/P.

n_x	n_y	processes												
		1	2	4	8	16	32	64	128	256	512	1024	2048	4096
800	800	432.4	237.0	148.6	69.5	34.4	17.3	8.8	4.8	2.6	1.7	1.1	0.9	0.6
800	1600	879.2	505.1	305.8	144.0	69.8	35.5	17.3	9.5	4.9	3.2	1.8	1.6	1.0
1600	1600	1772.9	979.6	595.1	312.0	152.8	71.6	35.6	17.7	9.2	5.3	3.0	2.2	1.5
1600	3200	3600.3	2082.4	1295.4	633.9	313.1	148.4	72.0	36.5	18.0	10.3	5.5	3.9	2.4
3200	3200	7439.4	4281.1	2720.9	1324.2	608.7	320.8	157.3	73.0	36.5	18.7	10.0	6.1	3.7
3200	6400				2795.3	1327.1	651.0	321.7	151.1	73.3	38.0	19.1	11.7	6.5
6400	6400					2777.5	1354.1	625.4	323.7	160.1	74.9	37.9	20.3	11.2
6400	12800						2853.8	1357.3	656.4	325.0	154.2	75.7	41.0	21.0
12800	12800							2844.2	1362.2	628.7	329.2	163.7	78.4	40.5
12800	25600								2867.5	1365.6	666.7	330.9	160.8	79.5
25600	25600									2858.7	1376.7	639.1	335.9	168.9
25600	51200										2897.0	1381.0	679.0	338.5
51200	51200											2884.5	1390.8	649.2

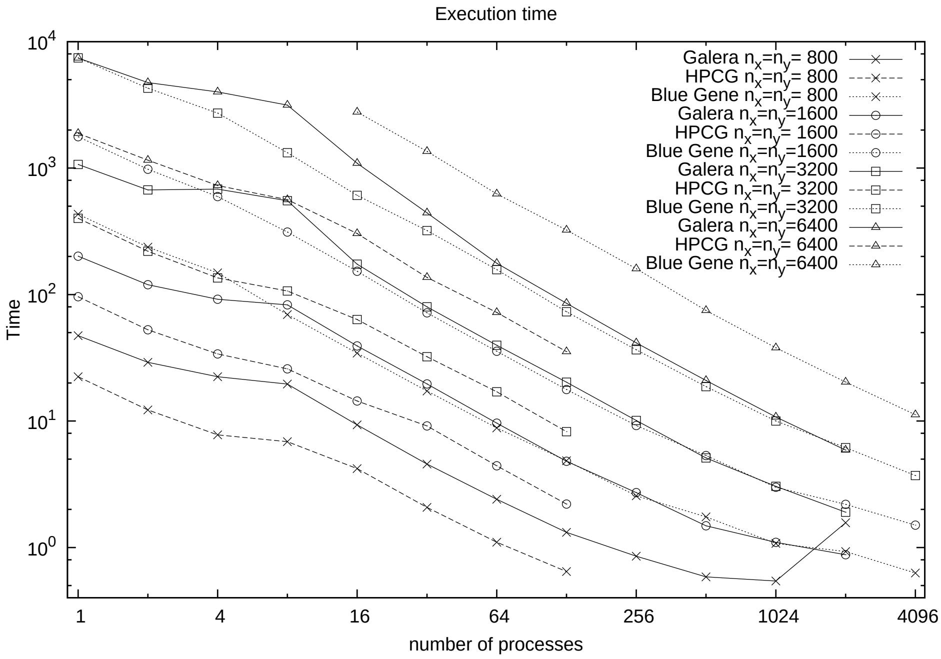

FIGURE 1. Execution time for 2D problem with $n_x = n_y = 800, 1600, 3200, 6400$

to the superior performance of the networking infrastructure of the Blue Gene.

To round up the performance analysis of the alternating directions algorithm, the speed-up obtained on Galera is reported in Table 5, while the speed-up on HPCG — in Table 6, the speed-up on the IBM Blue Gene/P — in Table 7, and the parallel efficiency — in Table 8.

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. 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 39% for the smallest problems to 73% for the largest problems.

For solving of the same 2D Stokes problem Elmer [14] Open Source Finite Element Software for Multiphysical Problems (see `http://www.csc.fi/english/pages/elmer`) was used with the following keywords in the solver input file:

TABLE 5. Speed-up on Galera.

n_x	n_y	processes										
		2	4	8	16	32	64	128	256	512	1024	2048
800	800	1.63	2.12	2.41	5.08	10.38	19.71	35.94	55.46	81.01	87.16	25.33
800	1600	1.66	2.17	2.43	4.97	10.27	20.51	38.80	67.11	106.83	120.14	127.52
1600	1600	1.68	2.20	2.43	5.12	10.24	20.90	41.90	74.07	135.77	183.15	229.49
1600	3200	1.66	2.05	2.19	5.51	11.06	22.08	44.61	87.61	160.21	236.53	361.04
3200	3200	1.59	1.57	1.94	6.15	13.39	26.94	52.74	105.98	209.44	351.23	562.98
3200	6400	1.44	1.32	1.71	5.67	14.18	31.44	61.93	122.34	246.74	443.73	779.57
6400	6400	1.56	1.85	2.36	6.80	16.75	41.92	86.86	178.57	354.76	688.90	1259.92
6400	12800	1.55	2.17	2.55	5.78	11.63	27.40	61.53	147.18	301.45	580.05	1118.72
12800	12800	1.63	2.43	3.09	6.98	12.40	31.32	62.53	166.39	402.91	815.48	1584.34

TABLE 6. Speed-up on HPCG.

n_x	n_y	processes						
		2	4	8	16	32	64	128
800	800	1.81	2.83	3.38	6.40	13.65	20.52	33.82
800	1600	1.85	2.93	3.52	6.74	12.76	23.04	41.47
1600	1600	1.89	3.04	3.77	6.92	14.43	23.05	43.55
1600	3200	1.93	3.05	3.84	6.56	14.37	24.39	45.87
3200	3200	1.94	3.08	3.76	7.74	14.20	26.76	48.58
3200	6400	1.92	3.01	3.75	7.72	16.84	30.09	52.01
6400	6400	1.70	2.70	3.35	7.42	16.62	26.22	53.26
6400	12800	1.84	2.92	3.84	7.60	12.69	27.80	59.07
12800	12800	1.70	2.59	2.92	6.63	13.60	24.02	49.91

```
Simulation
Coordinate System = "Cartesian 2D"
Simulation Type = Transient
Timestep intervals = 200
Timestep Sizes = 0.01
End

Solver 1
Equation = Navier-Stokes
Procedure = "FlowSolve" "FlowSolver"
Variable = Flow Solution[Velocity:2 Pressure:1]
Flow Model = Stokes
Stabilize = True
Optimize Bandwidth = True
Stabilization Method = Stabilized
Linear System Solver = Iterative
Linear System Iterative Method = BiCGStab
Linear System Max Iterations = 500
Linear System Convergence Tolerance = 1.0e-6
Linear System Preconditioning = ILU2
Linear System ILUT Tolerance = 1.0e-3
End
```

Fig. 2 shows the measured CPU time for solving of the 2D Stokes problem using Elmer software on the Blue Gene/P machine for discrete problems with $n_x = n_y = 100, 200, 400$.

TABLE 7. Speed-up on IBM Blue Gene/P.

n_x	n_y	processes											
		2	4	8	16	32	64	128	256	512	1024	2048	4096
800	800	1.82	2.91	6.22	12.57	25.05	48.99	89.10	168.67	247.90	401.65	465.48	687.32
800	1600	1.74	2.87	6.10	12.60	24.75	50.69	92.20	177.88	276.51	486.02	552.26	897.63
1600	1600	1.81	2.98	5.68	11.60	24.76	49.81	99.87	192.28	332.75	592.66	808.05	1180.51
1600	3200	1.73	2.78	5.68	11.50	24.26	49.97	98.57	200.07	349.59	651.68	923.50	1522.27
3200	3200	1.74	2.73	5.62	12.22	23.19	47.30	101.85	203.58	397.74	744.26	1209.29	2009.58

TABLE 8. Parallel efficiency.

n_x	n_y	processes											
		2	4	8	16	32	64	128	256	512	1024	2048	4096
							Galera						
800	800	0.813	0.529	0.302	0.321	0.324	0.308	0.281	0.217	0.158	0.085	0.015	
800	1600	0.829	0.543	0.304	0.314	0.321	0.320	0.303	0.262	0.209	0.117	0.076	
1600	1600	0.841	0.549	0.304	0.326	0.320	0.328	0.327	0.299	0.265	0.179	0.114	
1600	3200	0.830	0.513	0.273	0.345	0.346	0.345	0.348	0.342	0.313	0.231	0.176	
3200	3200	0.795	0.392	0.243	0.391	0.419	0.421	0.412	0.414	0.409	0.343	0.275	
3200	6400	0.720	0.329	0.214	0.365	0.443	0.490	0.483	0.477	0.481	0.433	0.381	
6400	6400	0.781	0.463	0.296	0.467	0.523	0.655	0.679	0.698	0.693	0.673	0.615	
6400	12800	0.776	0.543	0.319	0.361	0.363	0.428	0.481	0.575	0.589	0.566	0.546	
12800	12800	0.813	0.608	0.386	0.436	0.388	0.489	0.489	0.650	0.787	0.796	0.774	
							HPCG						
800	800	0.904	0.708	0.422	0.400	0.426	0.321	0.264					
800	1600	0.926	0.732	0.440	0.421	0.399	0.360	0.324					
1600	1600	0.947	0.760	0.471	0.432	0.451	0.360	0.340					
1600	3200	0.965	0.764	0.480	0.410	0.449	0.381	0.358					
3200	3200	0.969	0.771	0.470	0.484	0.444	0.418	0.380					
3200	6400	0.959	0.752	0.469	0.483	0.526	0.470	0.406					
6400	6400	0.849	0.675	0.418	0.464	0.519	0.410	0.416					
6400	12800	0.920	0.731	0.480	0.475	0.397	0.434	0.461					
12800	12800	0.849	0.647	0.365	0.414	0.425	0.375	0.390					
							IBM Blue Gene/P						
800	800	0.912	0.727	0.777	0.786	0.783	0.765	0.696	0.659	0.484	0.392	0.227	0.168
800	1600	0.870	0.719	0.763	0.787	0.773	0.792	0.720	0.695	0.540	0.475	0.270	0.219
1600	1600	0.905	0.745	0.710	0.725	0.774	0.778	0.780	0.751	0.650	0.579	0.395	0.288
1600	3200	0.864	0.695	0.710	0.719	0.758	0.781	0.770	0.782	0.683	0.636	0.451	0.372
3200	3200	0.869	0.684	0.702	0.764	0.725	0.739	0.796	0.795	0.777	0.727	0.590	0.491

ACKNOWLEDGMENTS

Computer time grants from the Bulgarian Supercomputing Center (BGSC) and the TASK are kindly acknowledged. This research was partially supported by grants DCVP 02/1 and I01/5 of the Bulgarian NSF. Work presented here is a part of the Poland-Bulgaria collaborative grant “Parallel and distributed computing practices”.

REFERENCES

1. A. J. Chorin, *Math. Comp.* **22**, 745–762 (1968).
2. R. Temam, *Arch. Rat. Mech. Anal.* **33**, 377–385 (1969).
3. J.-L. Guermond, P. Minev, and J. Shen, *Comput. Methods Appl. Mech. Engrg.* **195**, 6011–6054 (2006).
4. J.-L. Guermond, and P. Minev, *Comptes Rendus Mathematique* **348**, 581–585 (2010).

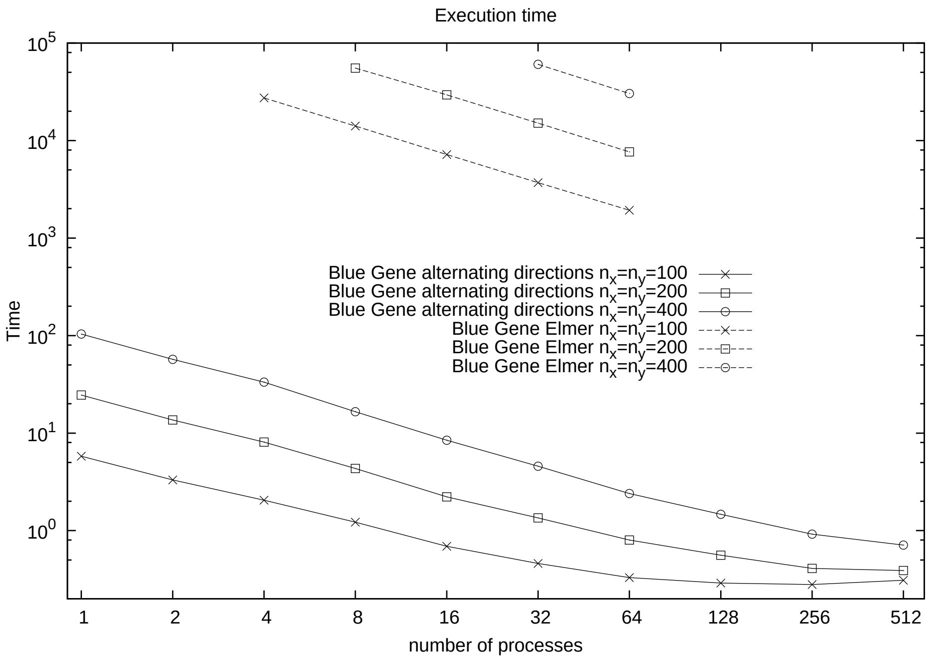

FIGURE 2. Execution time for 2D problem with $n_x = n_y = 100, 200, 400$

5. J.-L. Guermond, and P. Minev, *Computer Methods in Applied Mechanics and Engineering* **200**, 2083–2093 (2011).
6. J.-L. Guermond, and A. Salgado, *Comptes Rendus Mathematique* **346**, 913–918 (2008).
7. J.-L. Guermond, and A. Salgado, *Journal of Computational Physics* **228**, 2834–2846 (2009).
8. M. Snir, S. Otto, S. Huss-Lederman, D. Walker, and J. Dongarra, *MPI: The Complete Reference*, Scientific and engineering computation series, The MIT Press, Cambridge, Massachusetts, 1997, second printing.
9. D. Walker, and J. Dongarra, *Supercomputer* **63**, 56–68 (1996).
10. R. Chandra, R. Menon, L. Dagum, D. Kohr, D. Maydan, and J. McDonald, *Parallel programming in OpenMP*, Morgan Kaufmann, 2000.
11. B. Chapman, G. Jost, and R. Van Der Pas, *Using OpenMP: portable shared memory parallel programming*, vol. 10, MIT press, 2008.
12. E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Dongarra, J. D. Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorensen, *LAPACK Users' Guide*, SIAM, Philadelphia, 1999, third edn.
13. I. Lirkov, M. Paprzycki, and M. Ganzha, "Performance Analysis of Parallel Alternating Directions Algorithm for Time Dependent Problems," in *9th international conference on Parallel Processing and Applied Mathematics, PPAM 2011, Part I*, edited by R. Wyrzykowski, J. Dongarra, K. Karczewski, and J. Waśniewski, Springer, 2012, vol. 7203 of *Lecture notes in computer science*, pp. 173–182.
14. M. Lyly, J. Ruokolainen, and E. Järvinen, *CSC-report on scientific computing* **2000**, 156–159 (1999).