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@ibspan.waw.pl maria.ganzha@ibspan.waw.pl
http://www.ibspan.waw.pl/~paprzyck/ http://www.ganzha.euh-e.edu.pl

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 new direction splitting approach is developed. Here, 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 direction splitting. The scheme used in the algorithm is composed by: pressure prediction, velocity update, penalty step, and pressure correction. In order to achieve a 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 is developed using the standard MPI functions and tested on modern parallel computer systems. The performed numerical tests demonstrate the level of parallel efficiency and 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 for solving the incompressible Navier-Stokes equations based on a direction splitting strategy.

Projection schemes were introduced in [2, 11] and they have been used in Computational Fluid Dynamics (CFD) for about forty years. These techniques went through some evolution during the years, but the main paradigm consisting of decomposing vector fields into a divergence-free part and a gradient has been preserved; see [5] for a review. 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 at each time step of projection methods is that of solving one vector-valued advection-diffusion equation plus one scalar-valued Poisson equation with Neumann boundary conditions. For large size problems and large Reynolds numbers, the cost of solving the Poisson equation becomes dominant.

The alternating directions algorithm proposed in [4] reduce 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 direction splitting. The standard Poisson problem for the pressure correction is replaced by series of one-dimensional second-order boundary value problems. This technique is proved to be stable and convergent, for details see [4].

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:

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

where a smooth source term has components (f, g), (u_0, v_0) is a solenoidal initial velocity field with zero normal trace, and ν is the kinematic viscosity.

We discretized the time interval $[0, T]$ using uniform mesh. Let τ be the time step in the algorithm. Then we will denote by $t^n = n\tau$.

3 Parallel Alternating Directions Algorithm

Guermond and Minev introduced in [4] 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 perturbation parameter τ. The standard Poisson problem for the pressure correction is 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 by 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 $n > 0$ 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

The velocity field is initialized by setting $\mathbf{u}^0 = \begin{pmatrix} u_0 \\ v_0 \end{pmatrix}$, and for all $n > 0$ 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 $\mathbf{f}^{n+\frac{1}{2}} = \begin{pmatrix} f|_{t=(n+\frac{1}{2})\tau} \\ g|_{t=(n+\frac{1}{2})\tau} \end{pmatrix}$.

– 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 as follows:

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

The algorithm is in standard incremental form when the parameter $\chi = 0$ and the algorithm is in 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 of (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 on rectangular sub-domains. Let us associate with each such sub-domain a set of integer coordinates (i_x, i_y) 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 internal nodes for each block, that can be solved independently by a direct method, and the corresponding Schur complement for the interface unknowns between the blocks that have an equal coordinate i_x or i_y. The Schur complement is also tridiagonal and can therefore easily 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 an 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 with size n_x and (5) leads to linear systems with 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 [10, 12]. We use the LAPACK subroutines DPTTRF and DPTTS2 (see [1]) for solving tridiagonal systems in (4), (5), and (6) for the unknowns corresponding to internal nodes for each sub-domain. The same subroutines are used to solve tridiagonal systems with the Schur complement.

The parallel code has been tested on cluster computer system 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 Sooner. It is a Dell Pentium4 Xeon E5405 ("Harpertown") quad core Linux cluster located in the Oklahoma Supercomputing Center (see http://www.oscer.ou.edu/resources.php). 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 with a high-speed InfiniBand network. We have used Intel compiler and compiled the code with the following options: "-O3 -march=core2 -mtune=core2".

Table 1. Execution time on Sooner.

c	n_x	n_y	Time	n_x	n_y	Time	n_x	n_y	Time	n_x	n_y	Time
1	100	100	0.58	100	200	1.38	200	200	2.78	200	400	5.72
2	100	200	0.59	200	200	1.37	200	400	2.76	400	400	5.83
4	200	200	0.60	200	400	1.38	400	400	2.77	400	800	5.88
8	200	400	0.61	400	400	1.42	400	800	3.32	800	800	9.02
16	400	400	0.63	400	800	1.44	800	800	3.35	800	1600	9.02
32	400	800	0.67	800	800	1.51	800	1600	3.45	1600	1600	9.21
64	800	800	0.74	800	1600	1.58	1600	1600	3.61	1600	3200	9.34
128	800	1600	0.86	1600	1600	1.88	1600	3200	4.03	3200	3200	10.45
256	1600	1600	1.04	1600	3200	1.91	3200	3200	4.29	3200	6400	10.70
1	400	400	12.52	400	800	27.56	800	800	59.60	800	1600	120.33
2	400	800	12.55	800	800	28.73	800	1600	61.20	1600	1600	126.34
4	800	800	13.57	800	1600	32.42	1600	1600	72.43	1600	3200	147.28
8	800	1600	23.02	1600	1600	53.37	1600	3200	109.13	3200	3200	220.54
16	1600	1600	23.01	1600	3200	52.26	3200	3200	109.56	3200	6400	219.52
32	1600	3200	23.46	3200	3200	54.57	3200	6400	110.22	6400	6400	222.13
64	3200	3200	23.55	3200	6400	53.43	6400	6400	111.74	6400	12800	223.31
128	3200	6400	25.25	6400	6400	57.57	6400	12800	116.16	12800	12800	238.16
256	6400	6400	26.47	6400	12800	57.41	12800	12800	120.03	12800	25600	234.56

The results in one box of Table 1 are obtained for equal number of unknowns per core. For large discrete problems the execution time 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 show that the communication time between processors is larger than the communication time between cores of one processor. Also, the execution time for solving one and the same discrete problem decrease with increasing number of cores which shows that the communications in our parallel algorithm are mainly local.

The somehow slower performance on Sooner using 8 cores is clearly visible. The same effect was observed during our previous work, see [9]. There are some factors which could play role for the slower performance using all processors of one node. Generally they are a consequence of the 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. The L2 cache memory is shared among each pair of cores within the processors of Sooner. This boost the performance of programs utilizing only single core within such pair. Conversely, this leads for somehow decreased speedups when all cores are used. For memory intensive programs, these factors play a crucial role for the performance.

The speed-up obtained on Sooner is reported in Table 2 and the parallel efficiency is shown in Table 3. Increasing the number of cores, the parallel efficiency decreases on 8 cores and after that it increases to 100%. Moreover, a super-linear speed-up is observed. The main reasons for this fact can be related to splitting the entire problem into subproblems which helps memory management,

Table 2. Speed-up on Sooner.

n_x	n_y	c							
		2	4	8	16	32	64	128	256
800	800	2.07	4.39	6.61	17.79	39.57	80.71	93.72	122.91
800	1600	1.97	3.71	5.23	13.34	34.93	75.96	139.80	173.48
1600	1600	1.97	3.43	4.66	10.81	27.00	68.89	132.45	238.39
1600	3200	2.00	3.47	4.69	9.79	22.04	54.74	126.88	267.99
3200	3200	2.20	3.97	5.42	10.92	21.92	50.79	114.43	278.95
3200	6400	2.15	4.35	6.00	12.45	24.80	51.17	108.28	255.55
6400	6400	1.98	4.01	5.35	14.38	29.55	58.75	114.04	248.00

Table 3. Parallel efficiency on Sooner.

n_x	n_y	c							
		2	4	8	16	32	64	128	256
800	800	1.037	1.098	0.826	1.112	1.237	1.261	0.732	0.480
800	1600	0.983	0.928	0.653	0.834	1.091	1.187	1.092	0.678
1600	1600	0.984	0.858	0.583	0.676	0.844	1.076	1.035	0.931
1600	3200	1.001	0.868	0.586	0.612	0.689	0.855	0.991	1.047
3200	3200	1.101	0.993	0.678	0.682	0.685	0.794	0.894	1.090
3200	6400	1.077	1.087	0.750	0.778	0.775	0.799	0.846	0.998
6400	6400	0.988	1.003	0.669	0.899	0.924	0.918	0.891	0.969

in particular allows for better usage of cache memories of individual parallel processors.

Table 4 presents execution time on IBM Blue Gene/P machine at the Bulgarian Supercomputing Center (see 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 IBM XL 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, 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 the communication between processors is faster than the communication between cores of one processor using MPI communication functions. In order to get better parallel performance we plan to develop mixed MPI/OpenMP code and to use the nodes of the supercomputer in SMP mode with 4 OpenMP processes per node.

Table 5 shows the speed-up on IBM Blue Gene/P and the parallel efficiency is shown in Table 6. As expected, the parallel efficiency 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 in this set of experiments.

Execution time on Blue Gene/P is substantially larger than that on Sooner, but in some cases the parallel efficiency obtained on the supercomputer is better.

Table 4. Execution time on IBM Blue Gene/P.

c	n_x	n_y	Time	n_x	n_y	Time	n_x	n_y	Time	n_x	n_y	Time
1	100	100	5.79	100	200	12.33	200	200	24.51	200	400	49.02
2	100	200	5.96	200	200	11.84	200	400	24.93	400	400	49.89
4	200	200	6.17	200	400	13.02	400	400	25.68	400	800	51.34
8	200	400	6.34	400	400	12.48	400	800	26.21	800	800	52.59
16	400	400	6.63	400	800	13.83	800	800	27.33	800	1600	54.43
32	400	800	6.71	800	800	13.27	800	1600	27.53	1600	1600	54.94
64	800	800	6.85	800	1600	14.19	1600	1600	27.71	1600	3200	55.15
128	800	1600	7.04	1600	1600	13.72	1600	3200	28.15	3200	3200	56.20
256	1600	1600	7.18	1600	3200	14.69	3200	3200	28.34	3200	6400	56.44
512	1600	3200	7.56	3200	3200	14.59	3200	6400	29.13	6400	6400	58.08
1024	3200	3200	7.92	3200	6400	15.70	6400	6400	29.84	6400	12800	58.68
2048	3200	6400	8.81	6400	6400	16.74	6400	12800	31.38	12800	12800	62.83
4096	6400	6400	9.89	6400	12800	18.31	12800	12800	33.81	12800	25600	65.28
1	400	400	103.58	400	800	210.81	800	800	431.43	800	1600	877.0
2	400	800	105.23	800	800	214.55	800	1600	437.71	1600	1600	880.1
4	800	800	108.00	800	1600	219.95	1600	1600	450.15	1600	3200	913.4
8	800	1600	109.85	1600	1600	223.77	1600	3200	455.95	3200	3200	917.3
16	1600	1600	113.93	1600	3200	230.91	3200	3200	471.25	3200	6400	959.3
32	1600	3200	114.64	3200	3200	232.71	3200	6400	474.14	6400	6400	954.4
64	3200	3200	115.56	3200	6400	233.56	6400	6400	476.84	6400	12800	964.1
128	3200	6400	116.68	6400	6400	236.13	6400	12800	478.39	12800	12800	962.8
256	6400	6400	117.87	6400	12800	237.57	12800	12800	482.76	12800	25600	978.8
512	6400	12800	119.14	12800	12800	241.38	12800	25600	486.01	25600	25600	973.8
1024	12800	12800	120.69	12800	25600	242.71	25600	25600	489.00	25600	51200	987.2
2048	12800	25600	124.67	25600	25600	251.24	25600	51200	501.47	51200	51200	1027.3
4096	25600	25600	130.91	25600	51200	268.93	51200	51200	560.06	51200	102400	1163.2

Table 5. 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	15.79	32.52	62.94	114.61	215.88	330.37	585.6	631.5	780.8
800	1600	2.00	3.99	7.98	16.11	31.85	61.80	124.52	226.34	401.66	655.0	944.8	1177.6
1600	1600	2.00	3.92	7.89	15.49	32.12	63.68	128.66	245.91	422.72	745.0	999.9	1541.7
1600	3200	2.00	3.93	7.87	15.53	31.28	65.02	127.38	244.20	474.66	811.9	1290.0	1898.9
3200	3200	2.06	4.03	8.08	15.74	31.87	64.17	131.95	261.66	508.43	936.8	1411.7	2272.1

Table 6. 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	1.016	0.983	0.895	0.843	0.645	0.572	0.308	0.191
800	1600	1.002	0.997	0.998	1.007	0.995	0.966	0.973	0.884	0.784	0.640	0.461	0.287
1600	1600	1.002	0.980	0.986	0.968	1.004	0.995	1.005	0.961	0.826	0.728	0.488	0.376
1600	3200	1.000	0.982	0.983	0.971	0.978	1.016	0.995	0.954	0.927	0.793	0.630	0.464
3200	3200	1.029	1.008	1.011	0.984	0.996	1.003	1.031	1.022	0.993	0.915	0.689	0.555

For instance, 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 7 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.

Table 7. 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

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

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 two different parallel architectures. Satisfying parallel efficiency is obtained on both 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 we plan to develop mixed MPI/OpenMP code. We will continue our research developing a parallel algorithm for solving of 3D Stokes equation.

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

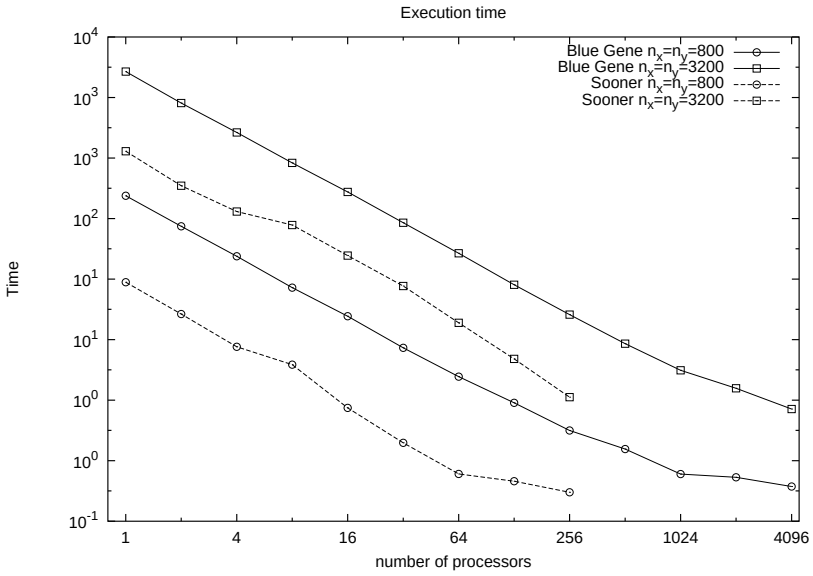

Fig. 1. Execution time for $n_x = n_y = 800, 3200$

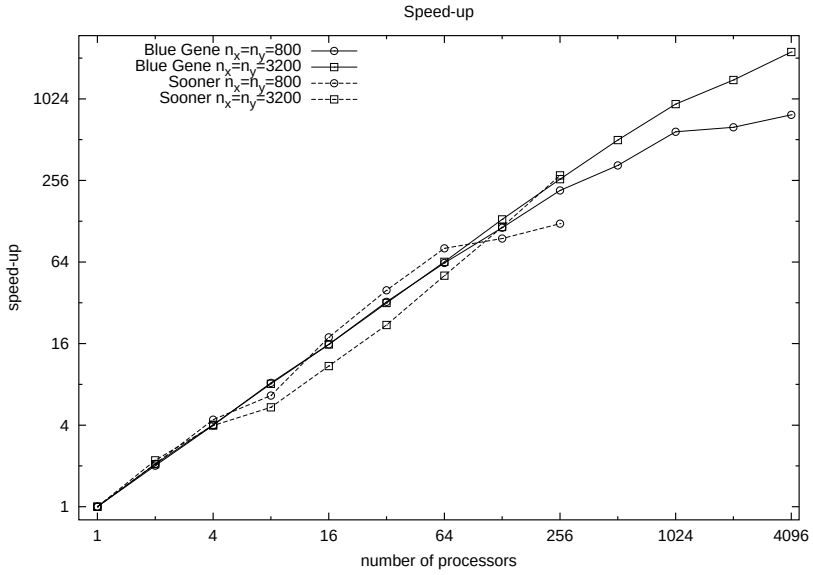

Fig. 2. Speed-up for $n_x = n_y = 800, 3200$

2. A. J. Chorin, Numerical solution of the Navier-Stokes equations, *Math. Comp.*, **22**, 1968, 745–762.
3. G. H. Golub, C. F. Van Loan, *Matrix computations*, Johns Hopkins Univ. Press, Baltimore, 2nd edition, 1989.
4. 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.
5. J.-L. Guermond, P. Minev, J. Shen, An overview of projection methods for incompressible flows, *Comput. Methods Appl. Mech. Engrg.*, **195**, 2006, 6011–6054.
6. 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.
7. 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.
8. J.-L. Guermond, J. Shen, On the error estimates for the rotational pressure-correction projection methods, *Math. Comp.*, **73** (248), 2004, 1719–1737.
9. 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.
10. M. Snir, St. Otto, St. Huss-Lederman, D. Walker, and J. Dongarra, *MPI: the complete reference*, Scientific and engineering computation series. The MIT Press, Cambridge, Massachusetts, 1997, Second printing.
11. 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.
12. D. Walker and J. Dongarra, MPI: a standard Message Passing Interface, *Supercomputer*, **63**, 1996, 56–68.