Implementation of a three dimensional Three-Phase Fluid Flow ("Oil-Water-Gas") Numerical Model in LuNA Fragmented Programming System

Darkhan Akhmed-Zaki[1], Danil Lebedev [1], Vladislav Perepelkin [2]

[1] al-Farabi Kazakh National University, Almaty, Republic of Kazakhstan
Darhan.Ahmed-Zaki@kaznu.kz, danil.lebedev.0881@gmail.com
[2]Institute of Computational Mathematics and Mathematical Geophysics SB RAS, Novosibirsk, Russia
perepelkin@ssd.sscc.ru

Abstract. To overcome the difficulties of efficient scalable numerical algorithms implementation on multicomputers with a large number of computing nodes, LuNA system is being developed. LuNA automatically generates a code, related to communications, resources management and dynamic workload balancing, thus simplifying construction of parallel programs. To examine the efficiency of numerical model implementation, created with LuNA, a real application of a three-phase (oil, water and gas) fluid filtration simulation is studied. The application algorithm is parallelized and implemented using LuNA and conventional MPI-based approach. A comparative performance testing of the two implementations is done. The results show that LuNA implementation is easier to develop, but its efficiency is significantly lower than the efficiency of MPI program, but manual tuning of the program execution makes its efficiency comparable to that of MPI program.

Keywords: Fragmented programming, LuNA system, parallel algorithm study, Thomas algorithm.

Acknowledgments

This work was supported by grant funding of scientific and technical programs and projects by the Committee of Science, Ministry of Education and Science of RK, grant No. 80/GF4; and Russian Foundation for Basic Research (grants No 14-07-00381-a and 14-01-31328-mol_a). The tests were conducted on the 'MVS' supercomputer of Joint Supercomputing Center of Russian Academy of Sciences.

1 Introduction

Implementation of large-scale numerical models on supercomputers is challenging, especially for non-experts in system parallel programming because of the necessity of having program tune and scale to available resources, to dynamically balance work-

load and to organize a huge number of computing nodes in parallel computations. Hence the effort put into study and development of parallel programming systems and tools which simplify parallel programs construction (see: PaRSEC [9], libgeodecomp [10], Charm++ [11], KeLP [12], LuNA [7–8, 13]). Although the systems automate construction of parallel programs, the efficiency of the programs is hardly as high as the one of conventional hand-made MPI-based programs. Thus, it is important to examine the efficiency of such programs against MPI-based programs in case of real applications. The objective of this paper is to compare the efficiency of implementation of the solution of a 3D boundary value problem for three-phase (oil, water and gas) fluid filtration simulation with LuNA system against the efficiency of conventional MPI-based implementation of the same algorithm.

The paper is organized as follows. The next section describes the application problem and the parallel algorithm of its solution. Section 3 contains a brief description of LuNA system. The paper is ended with performance tests and conclusion.

2 The Problem of Filtration and its Parallel Solution

2.1 The Problem and the Sequential Algorithm

A three-phase fluid filtration process is described by a system of three differential equations which define the change of pressure P_l and saturation S_l, $l \in \{o,w,g\}$ for each fluid phase (oil, water and gas) in space-time coordinates for dimensionless variables [1-4]. The ratio in relation to pressure and saturation and initial and boundary conditions are added to the system.

$$
\begin{cases}
\frac{\partial}{\partial x}\left[K_w\left(\frac{\partial P_w}{\partial x}-\gamma_w\frac{L}{P_H}\frac{\partial z}{\partial x}\right)\right]+\frac{\partial}{\partial y}\left[K_w\left(\frac{\partial P_w}{\partial y}-\gamma_w\frac{L}{P_H}\frac{\partial z}{\partial y}\right)\right]+\frac{\partial}{\partial z}\left[K_w\left(\frac{\partial P_w}{\partial z}-\gamma_w\frac{L}{P_H}\right)\right]=\frac{\mu_w}{\mu_o}\frac{\partial S_w}{\partial \tau}+\frac{\mu_w L^2}{K\rho_w P_H}q_w \\
\frac{\partial}{\partial x}\left[K_o\left(1+C_f P_H(P_o-1)\right)\left(\frac{\partial P_o}{\partial x}-\gamma_o\frac{L}{P_H}\frac{\partial z}{\partial x}\right)\right]+\frac{\partial}{\partial y}\left[K_o\left(1+C_f P_H(P_o-1)\right)\left(\frac{\partial P_o}{\partial y}-\gamma_o\frac{L}{P_H}\frac{\partial z}{\partial y}\right)\right]+ \\
+\frac{\partial}{\partial z}\left[K_o\left(1+C_f P_H(P_o-1)\right)\left(\frac{\partial P_o}{\partial z}-\gamma_o\frac{L}{P_H}\right)\right]=\frac{\partial}{\partial \tau}\left[S_o\left(1+C_f P_H(P_o-1)\right)\right]+\frac{\mu_o L^2}{K\rho_H P_H}q_o \\
\frac{\partial}{\partial x}\left[R_s K_o\left(1+C_f P_H(P_o-1)\right)\left(\frac{\partial P_g}{\partial x}-\frac{\partial P_{cog}}{\partial x}-\gamma_o\frac{L}{P_H}\frac{\partial z}{\partial x}\right)\right]+ \\
+\frac{\partial}{\partial y}\left[R_s K_o\left(1+C_f P_H(P_o-1)\right)\left(\frac{\partial P_g}{\partial y}-\frac{\partial P_{cog}}{\partial y}-\gamma_o\frac{L}{P_H}\frac{\partial z}{\partial y}\right)\right]+ \\
+\frac{\partial}{\partial z}\left[R_s K_o\left(1+C_f P_H(P_o-1)\right)\left(\frac{\partial P_g}{\partial z}-\frac{\partial P_{cog}}{\partial z}-\gamma_o\frac{L}{P_H}\right)\right]+ \\
\frac{\mu_o P_H}{\mu_g\rho_H RTZ}\frac{\partial}{\partial x}\left[K_g P_g\left(\frac{\partial P_g}{\partial x}-\gamma_g\frac{L}{P_H}\frac{\partial z}{\partial x}\right)\right]+\frac{\mu_o P_H}{\mu_g\rho_H RTZ}\frac{\partial}{\partial y}\left[K_g P_g\left(\frac{\partial P_g}{\partial y}-\gamma_g\frac{L}{P_H}\frac{\partial z}{\partial y}\right)\right]+ \\
+\frac{\mu_o P_H}{\mu_g\rho_H RTZ}\frac{\partial}{\partial z}\left[K_g P_g\left(\frac{\partial P_g}{\partial z}-\gamma_g\frac{L}{P_H}\right)\right] \\
=\left(1-C_f P_H\right)\frac{\partial}{\partial \tau}(R_s S_o)+C_f P_H\frac{\partial}{\partial \tau}(S_o)+\frac{P_H}{\rho_H RTZ}\frac{\partial}{\partial \tau}\left(P_g S_g\right)+\frac{\mu_o L^2}{K\rho_H P_H}\left(R_s q_o+q_g\right) \\
P_o-P_w=P_{cow} \\
P_g-P_o=P_{cog} \\
S_w+S_o+S_g=1
\end{cases}
\tag{1}
$$

Initial and boundary conditions

$$
\begin{gathered}
P_0(x,y,z,0)=P_o^H(x,y,z), P_w(x,y,z,0)=P_w^H(x,y,z), P_g(x,y,z,0)=P_g^H(x,y,z) \\
S_0(x,y,z,0)=S_o^H(x,y,z), S_w(x,y,z,0)=S_w^H(x,y,z), S_g(x,y,z,0)=S_g^H(x,y,z)
\end{gathered}
\tag{2}
$$

$$
\left.\frac{\partial P_o}{\partial n}\right|_\Gamma=0, \left.\frac{\partial P_w}{\partial n}\right|_\Gamma=0, \left.\frac{\partial P_g}{\partial n}\right|_\Gamma=0 \tag{3}
$$

To solve the system (1) an iterative method with implicit pressure and explicit saturation is used [2]. For this, the first three equations of the system are summed and, using the fourth and fifth ratios of the system (1) the obtained equation is solved in relation to the gas pressure. The resulting equation is reduced to the implicit difference scheme, then solved by the scheme of Douglas and Rachford [5].

Each time step comprises three intermediate steps, each being an iterative process. At each iteration the following is computed:

1. The gas pressure is determined by the tridiagonal matrix algorithm (TMA) [6];
2. Oil and water pressures are computed from the ratios $P_g - P_o = P_{cog}$ and $P_o - P_w = P_{cow}$;
3. Oil and water saturations are found using the first and second differential equations, while the gas saturation is found from the ratio $S_w + S_o + S_g = 1$.

The iterative process of each intermediate step ends when the convergence condition is satisfied. At each iteration step, the value of pressure and saturation from the previous iteration step, previous intermediate step and previous time step are used.

2.2 Parallel Algorithm

We propose the following parallel algorithm. The 3D mesh of the size Nx×Ny×Nz is partitioned in one direction (1D decomposition), one subdomain for each process. Each subdomain is extended with shadow edges of size 1. At the beginning, the decomposition is made in z direction with the subdomain of the size Nx×Ny×(Nz/size+2).

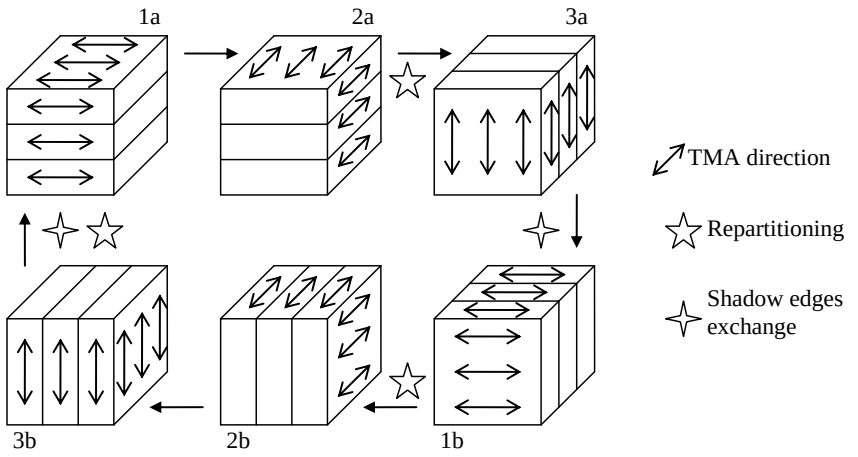

Fig.1. Scheme of computations: six intermediate steps repeating each two time steps (a and b); stars denote communications

Such decomposition allows computing two intermediate steps without any communications. After each two intermediate steps, mesh redistribution is required with

the direction of partitioning changing from z to y, then from y to x, then from x to z, and so on. After each third intermediate step, shadow edges exchange is performed. In such a way, the full cycle takes two time steps or six intermediate steps (see Fig. 1).

2.3 Algorithm analysis

The main advantage of the proposed algorithm is the possibility to employ sequential implementation of the TMA. Other parallel methods such as Yanenko method [15], pipelined Thomas method[16] or parallel cyclic reduction method [17] contain communications within each iteration step.

This advantage comes with the price of global collective all-to-all communications, necessary for repartitioning. Also, it is known that 1D decomposition is not too much scalable, since the whole subdomain with the minimal width of 1 must fit into the memory of a computing node, as well as the number of computing nodes cannot exceed the mesh size in any direction. For the listed methods there is no such limitation.

3 LuNA Language and System of Fragmented Programming

LuNA (Language for Numerical Algorithms) is a language and a system of parallel programs construction, being developed in ICMMG SB RAS. Its objective is automation of construction of parallel programs which implement large-scale numerical models on distributed memory multicomputers with big number of computing nodes. Its theoretical basis can be found in [14].

To effectively automate parallel program construction, LuNA employs a distinctive model of computations to represent an application algorithm, which is called Fragmented Algorithm (FA). Unlike MPI-based programs, FA consists of lightweight processes, called Computational Fragments (CF) which are basically calls of conventional pure functions. Data of an application are represented by immutable data blocks called Data Fragments (DF). values of DFs are computed by CFs from other DFs called output and input DFs, respectively. FA is executed is accordance with the data-flow model by LuNA run-time system.

Using FA to represent an application, the algorithm has a number of advantages for the benefit of the user:

- FA is independent of multicomputer configuration and thus portable and tunable to any resources available. Neither multicomputer size nor its heterogeneity or network topology has to be considered by the user.
- Due to the absence of side effects CFs and DFs can be transparently moved by run-time system from one computing node to another to implement dynamic workload balancing or automate communications.
- Creation of FA requires no parallel programming as such, the user's role is cut down to sequential programming of pure functions with a conventional programming language and description of the sets of DFs and CFs in basically functional LuNA language. In particular, no communications, synchronizations or resources management are to be coded.

Currently, LuNA system is implemented as a prototype. FA described in LuNA language is interpreted by the run-time system which invokes the user`s sequential procedures encapsulated in a traditional dynamic load library according to informational dependencies described in FA. Thus, implementation of a coarse-grained parallel algorithm mainly consists of native code execution and minor system overhead. On the contrary, fine-grained FA is likely to have poor performance due to run-time system overhead. The LuNA run-time system is designed to scale to multicomputers of any size, because it only employs scalable distributed system algorithms with neighbor-only communications (in sense of network topology).

Similar to other dataflow models, FA can be executed in a number of scenarios, different in terms of resources distribution, CFs execution order, etc. The choice of the best (or at least a good) scenario is a computationally hard optimization problem, which is a serious obstacle to achieving high efficiency of computations. To partially overcome it, LuNA employs the recommendations concept, which allows the user to manually tune FA execution by specifying preferred resources distribution scheme, CF execution partial order, etc. in terms of event-driven computations. Despite the necessity to concern efficiency issues the user still only has to describe the desired FA behavior, not to program it.

4 Performance Tests

Performance testing was conducted on ‘MVS’ supercomputer of Joint Supercomputing Center of Russian Academy of Sciences, which comprises nodes with dual Xeon E5-2690 processors and 64GB RAM per node. The MPICH3 implementation of MPI and GCC was employed.

Performance tests were conducted on a series of experiments, which determine weak scalability of an implementation. That is, for different problem sizes, the growth of the problem size increases proportionally to the number of computing nodes. While, ideally, the time measured would be the same for all experiments, since the amount of computations per node is approximately the same, in real life, time increases due to the growing length of communications and other impeding factors.

That is what we see in Fig. 2 (the X axis is the number of cores and the corresponding mesh size) for the MPI implementation, which we consider the best, since it does not possess the overhead a run-time system implies. The impeding factor here is the above-mentioned global all-to-all communications. It also can be seen that LuNA implementation is approximately 200 times slower, than MPI, while LuNA with manual tuning (LuNA-fw) is approximately 10 times slower.

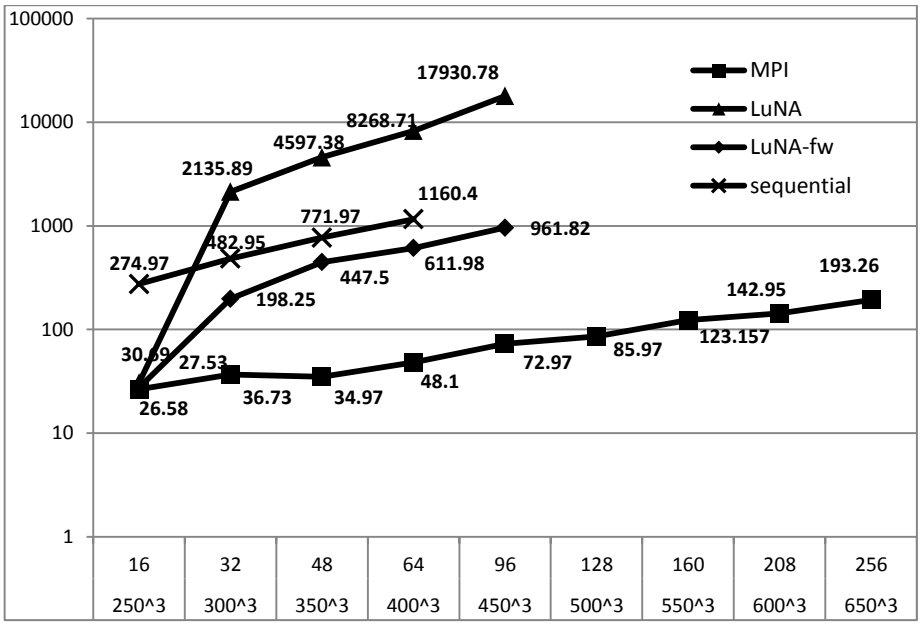

Fig. 2 Weak scalability testing: dependency of computation time on problem and multicomputer sizes

Despite the lower efficiency of FA execution, FA development and debugging is easier since no communications, synchronization or resources allocation are to be coded.

5 Conclusion

A comparative efficiency analysis of a three-phase fluid flow ("oil-water-gas") numerical model implementation with LuNA system and MPI is performed. The study shows that LuNA system simplifies algorithm implementation, but the efficiency is poor, unless LuNA-program execution is manually tuned using corresponding LuNA system means.

References

1. Aziz, K., Settari, A.: Petroleum reservoir simulation. Applied Science Publishers Ltd. London, U.K., 407p. (1979)
2. Crichlow, H.B.: Modern Reservoir Engineering – a simulation Approach. Prentice-Hall, Inc., Englewood Cliffs, NJ 303 p.. (1977)
3. Konovalov, A.N.: The problem of filtration multiphase incompressible fluid. Novosibirsk: science. 166p. (1988)

4. Akhmed-Zaki, D. Zh., Danaev, N. T., Mukhambetzhanov, S. T., Imankulov, T.: Analysis and Evaluation of Heat and Mass Transfer Processes in Porous Media Based on Darcy - Stefan's Model, ECMOR XIII 2012, http://dx.doi.org/10.3997/2214-4609.20143274
5. Jim Douglas, jr. and H.H. Rachford, jr. On the numerical solution of heat conduction problems in two and three space variables. «Transactions of the Amer. Math. Soc.», 1956, vol. 82, № 2,)
6. L.H. Thomas, "Elliptic Problems in Linear Difference Equations over a Network," Watson Sci. Comput. Lab. Rept., Columbia University, New York, 1949.
7. Malyshkin, V., Perepelkin, V.: Optimization methods of parallel execution of numerical programs in the LuNA fragmented programming system. J. Supercomput., Springer, Vol. 61, N. 1, pp. 235-248 (2012) DOI: 10.1007/s11227-011-0649-6.
8. Kireev, S, Malyshkin, V., Fujita, H.: The LuNA Library of Parallel Numerical Fragmented Subroutines. LNCS Vol. 6873, p. 290 –301 (2011)
9. PaRSEC: Parallel Runtime Scheduling and Execution Controller. http://icl.cs.utk.edu/parsec/index.html - accessed 15 jan 2015
10. Schaefer, A., Fey, D.: LibGeoDecomp: A Grid-Enabled Library for Geometric Decomposition Codes. Proc. of the 15th European PVM/MPI Users' Group Meeting on Recent Advances in Parallel Virtual Machine and Message Passing Interface. Springer-Verlag, Berlin, Heidelberg, pp 285-294 (2008) DOI: 10.1007/978-3-540-87475-1_39
11. Kale, L.V., Krishnan, S.: CHARM++: a portable concurrent object oriented system based on C++. OOPSLA '93 Proc. of the eighth annual conference on Object-oriented programming systems, languages, and applications. ACM New York, NY, USA pp 91-108 (1993) DOI: 10.1145/165854.165874
12. Kohn, S., Weare, J., Ong, M.E. Baden, S.B.: Software Abstractions and Computational Issues in Parallel Structured Adaptive Mesh Methods for Electronic Structure Calculations. IMA Volumes in Mathematics and its Applications, Volume 117, Structured Adaptive Mesh Refinement (SAMR) Grid Methods. S. B. Baden, N. Chrisochoides, M. Norman, and D. Gannon, Eds., Springer-Verlag, pp. 75-95 (1999)
13. Malyshkin, V.E., Perepelkin, V.A. LuNA Fragmented Programming System, Main Functions and Peculiarities of Run-Time Subsystem. In: Proc. of the 11th Conference on Parallel Computing Technologis, LNCS 6873. p. 53-61, Springer (2011)
14. Valkovsky, V.A., Malyshkin, V.E.: Synthesis of parallel programs and system on the basis of computational models. Nauka, Novosibirsk, 1988, 128 pp. (In Russian) (1988)
15. Yanenko N.N., Konovalov A.N., Bugrov A.N., Shustov G.V On organization of parallel computations and parallelization of the tridiagonal matrix algorithm // Numerical Methods of Continuum Mechanics. 1978.
16. Vol. 9. №7. P. 139-146Sapronov I.S., Bykov A.N.. Pipelined Thomas algorithm // Atom. 2009. № 44. P. 24-25
17. Hockney R. W., Jesshope C. R. Parallel computers: architecture, programming and algorithm. Hilger. Bristol. 1986. P. 274-280