Implementation of a 3D Model Heat Equation Using Fragmented Programming Technology

Darkhan Akhmed-Zaki[1,2], Danil Lebedev[1,2] and Vladislav Perepelkin[3,4]

[1] Al Farabi Kazakh National University, Almaty, Kazakhstan
[2] University of International Business, Almaty, Kazakhstan
[3] Institute of Computational Mathematics and Mathematical Geophysics SB RAS, Novosibirsk, Russia
[4] Novosibirsk State University, Novosibirsk, Russia
danil.lebedev.0881@gmail.com, perepelkin@ssd.sscc.ru

Abstract. Development of efficient numerical programs for large distributed parallel computers is a challenging problem. Many programming languages, systems and libraries exist and evolve to help with it, yet the problem is far from being solved. Of interest are particular application implementations' studies, which reveal actual capabilities of a system in the real computation. In this paper the implementation of an indicative 3D model heat equation parallel solver using Fragmented Programming Technology and LuNA system is investigated. A comparative testing with conventional MPI implementation is presented. The pros and cons of the approach are analyzed for corresponding applications class.

Keywords: Scalable numerical algorithms, pipelined Thomas algorithm, fragmented programming system LuNA, parallel programming automation.

1 Introduction

Nowadays increasingly-complex high performance computing (HPC) hardware imposes increasingly strong demands on application software. HPC clusters often comprise computing nodes of different performance or memory capacity, non-uniform network topology is also common. Some of computing nodes may contain co-processors, such as GPUs or FPGAs. Programming for such an environment is complex and error-prone, it requires high skills in system parallel programming. Such skills are different from the ones an application programmer normally possesses. In particular, one has to cope with network communications, processes' and threads' synchronization, load balancing, interaction with co-processors, etc. Moreover, the program has to tune to certain hardware configuration, which is generally unknown until program start.

Under such constraints an important role is played by programming systems, capable of dealing with the low-level problems. The more complex problems are the more intelligent programming systems have to be. In particular, a system not only has to "know" the hardware it works with, but also to "understand" application code (and data) peculiarities. Such knowledge allows the system to reconfigure and adapt application code to given hardware in order to satisfy performance or other non-functional requirements. Also, systems hide low-level programming details from the user.

Currently there are a lot of systems and projects, aimed to ease the programming. Worth mentioning are PaRSEC [2,3], Charm++ [4,5], Chapel [10], X10 [12], UCX [13] and many others. Despite the big effort involved with their development, the problem of parallel programming automation is still challenging and far from being solved. Existence of many research projects indicates the necessity to further investigate the problem. Of interest are particular studies, which reveal capabilities of certain programming systems and approaches in dealing with real-life computations.

This paper is devoted to such a study. A model 3D equation solver is chosen as an indicative iterative application on a 3D mesh. Computations mostly consist of solving tridiagonal equations. The application is implemented with Fragmented Programming Technology [1], which is being developed in Institute of Computational Mathematics and Mathematical Geophysics, SB RAS. The technology is aimed at automation of implementation of numerical applications for multicomputers. The experimentation is conducted on the computing cluster of Joint Supercomputing Centre of RAS [11].

2 LuNA System

System LuNA (Language for Numerical Algorithms) is an academic project of ICMMG SB RAS to develop a programming system, capable of parallel execution of numerical algorithms, represented in a resources-independent coarse-grained explicitly-parallel form, called Fragmented Algorithm (FA). As compared to conventional parallel programming, for example, with use of MPI, LuNA automates data and computations distribution, network data transfer, scheduling of computations and a number of other tasks, essential to MPI programs development.

FA is basically a potentially infinite bipartite directed acyclic graph of computational fragments (CF) and data fragments (DF) with arcs denoting informational dependencies. Input, output or intermediate data of the algorithm are represented by a set of DFs, each being an immutable aggregated variable (e.g. a subdomain of a mesh at given time step or iteration). The set of DFs represent data decomposition, each DF being a mesh domain, an array slice, etc. Computations of the algorithm are represented by a set of CFs. Each CF is associated with an operation to compute and a finite number of input and output DFs. Once values of input DFs of a CF are available (computed), the CF may be executed to produce values of its output DFs. Execution of a FA consists of execution of all its CFs. A FA is defined in LuNA (Language for Numerical Algorithms) programming language, which is a functional language. The operations CFs perform are sequential side-effect free (“pure”) subroutines in C++ (or other conventional linking-compatible language). FA is interpreted and executed by LuNA system on a multicomputer, automating CFs and DFs distribution to computing nodes, transfer of input DFs to CFs, that consume them, scheduling and execution of CFs in an order, which does not contradict their informational dependencies. LuNA run-time system also performs dynamic load balancing and a number of other system functions. “Under the hood” LuNA system only uses scalable distributed algorithms, which only employ communications between neighboring computing nodes (in sense of network topology), which makes it scalable to potentially unlimited number of computing nodes, provided the application FA itself is scalable enough.

One of the possibilities Fragmented Programming Technology suggests is the ability to construct alternative programs which implement the same FA, but possess different non-functional properties (different behaviors, i.e. fragments distribution to nodes over time and CFs execution order). LuNA run-time system, for instance, employs dynamic interpretation of FA in order to implement it, with dynamic resources distribution and CFs scheduling. In some cases resources distribution and CFs scheduling can be performed statically and defined using an alternate run-time subsystem called LuNA-framework. Such approach eliminates significant amount of run-time overhead with the price of manual FA behavior specification and reduction of program’s dynamism.

3 Problem Formulation and Fragmented Algorithm

The following model 3D heat equation in the unit cube [6] is considered:

$$u_t = u_{xx} + u_{yy} + u_{zz} \quad (1)$$

Initial conditions:

$$u(x,y,z,0) = u_0(x,y,z) \quad (2)$$

Boundary conditions:

$$\begin{matrix} u(0,y,z,t) = u(x,0,z,t) = u(x,y,0,t) = 1 \\ u(1,y,z,t) = u(x,1,z,t) = u(x,y,1,t) = 1 \end{matrix} \quad (3)$$

To solve equation (1) with initial (2) and boundary (3) conditions the scheme of Douglas and Rachford [7] is applicable. The basic idea is to define intermediate steps for a time step. The first step gives a total approximation of the heat equation, next steps are correction steps, which improve the numerical stability. Each intermediate step is solved with Thomas algorithm. The difference scheme is:

$$\begin{aligned} \frac{u^{n+1/3}-u^n}{\tau} &= \Lambda_1 u^{n+1/3} + \Lambda_2 u^n + \Lambda_3 u^n \\ \frac{u^{n+2/3}-u^{n+1/3}}{\tau} &= \Lambda_2\left(u^{n+2/3} - u^n\right) \qquad (4) \\ \frac{u^{n+1}-u^{n+2/3}}{\tau} &= \Lambda_3(u^{n+1} - u^n) \end{aligned}$$

For parallel implementation of the tridiagonal equation solution the parallel pipelined algorithm [8] was employed. It uses spatial domain decomposition for the set of tridiagonal equations. At first, a subset of equations is selected. Once a domain of each equation is processed, the domain boundary values are transferred to the node, containing the next domain, and the next subset is processed. The same scheme is employed for both forward and backward computational steps of the tridiagonal equation solution.

Douglas and Rachford scheme was used in combination with the pipelined Thomas algorithm because of the following reasons. Firstly, since all tridiagonal equations for given dimension are independent, they can be computed using the pipelined Thomas algorithm in parallel. Secondly, no edges' exchanges (and therefore communications) are required between intermediate steps. Thirdly, in [9] authors show applicability with high efficiency of the pipelined Thomas algorithm for one intermediate step of the scheme of Douglas and Rachford for a part of the whole domain.

The mesh u^n is decomposed into a 3D grid of domains each being a submesh DF. Each domain is normally assigned to a computing node, which will store and process the domain (best for a 3D mesh of computing nodes of corresponding size, but also good for typical fat tree cluster topologies). Each intermediate step comprises multiple solutions of the pipelined Thomas algorithm. The total number of tridiagonal equations to solve is equal to multiplication of domain sizes in two dimensions, perpendicular to the dimension of the equation. The pipelined Thomas algorithm possesses a parameter, the portion size. Based on the experimental research, conducted in [9], we choose size of subsets equal to *y* domain size, which makes *z* domain size a number of pipeline cycles. After all three intermediate steps are finished u^{n+1} is obtained. Before passing to the next time step border mesh values of *u* are exchanged between neighboring domains. Each intermediate or final value of each domain corresponds to a DF, is computed by a CF from other DFs according to informational dependencies.

4 Testing

Performance testing was conducted on MVS10P supercomputer of the Joint Supercomputer Centre of Russian Academy of Sciences [11]. It comprises 2×Xeon E5-2690 CPU-based computing nodes with 64 GB RAM each. A sequential, MPI, LuNA and LuNA framework programs were tested. The following parameters, representative for such applications, were chosen. Mesh size: from 100^3 to 700^3 with step 100 (in every dimension). Number of processors: from 8 (2^3) to 216 (6^3) with step 1 (in each dimension).

Test results are shown in Table 1. It can be seen, that hand-coded MPI program has the best performance. The fragmented algorithm execution with LuNA framework has lower performance, yet comparable with MPI program's performance. Basic LuNA performance is significantly lower. The performance decrease in the implementations correlates with the increase of ease of programming. LuNA program development and debugging is the simplest due to absence of necessity to deal with communications and resources distribution. LuNA-framework program required behavior (control and resources distribution) to be defined without the need to program it, while MPI implementation required both definition and hand-coding of control and resources distribution. Test results also allow seeing system overhead and scalability of different implementations of the algorithm.

Table 1. Execution time (in seconds)

NP	**Size**					
	200^3	**300^3**	**400^3**	**500^3**	**600^3**	**700^3**

Sequential						
1^3	1 341,9	5 443,0	19 406,0	29 195,5	50 728,4	82 882,3
MPI						
2^3	250,3	950,4	2 192,4	4 263,9	7 273,9	13 072,0
3^3	192,8	568,4	1 264,4	2 262,1	3 185,2	5 297,0
4^3	196,6	522,6	1 027,0	1 920,4	2 971,7	4 437,8
5^3	169,5	363,8	764,8	1 273,6	2 040,6	3 130,9
6^3	130,1	346,3	593,1	1 016,9	1 412,4	2 046,1
LuNA						
2^3	2 963,08	4 557,84	7 216,67	11 671,46	18 163,35	28 392,50
3^3	4 935,30	7 258,55	10 153,5	13 802,12	18 421,85	24 280,50
4^3	8 222,12	11 776,55	15 928,4	20 830,20	26 620,70	33 428,60
5^3	2 560,43	19 429,94	26 107,8	32 214,55	40 891,55	72 236,53
LuNA-fw						
2^3	349,18	1 004,40	2 632,80	6 791,05	9 373,10	16 960,20
3^3	202,28	587,67	1 202,88	2 462,91	4 880,40	7 537,10
4^3	1 920,21	1 654,13	2 899,45	3 412,50	4 133,35	5 576,15
5^3	1 922,18	2 554,05	7 425,75	2 806,90	3 354,75	3 956,15
6^3	1 536,18	1 425,01	2 578,61	1 894,11	2 465,57	2 935,53

It can be stated, that currently fragmented programming technology and LuNA system provide means for development and debugging of parallel programs in high level of abstraction (LuNA language), but in order to achieve good performance, additional fragmented programming execution tuning has to be performed (behavior specification with LuNA-Framework).

5 Conclusion

An implementation of a model 3D heat equation solver using Douglas and Rachford scheme with the Fragmented Programming Technology is considered. Comparative performance tests of different implementations of the solver are conducted. The comparative testing of MPI and LuNA implementations shows advantage of MPI implementation over LuNA in sense of execution time, although LuNA program is easier to develop due to programming automation, provided by LuNA system. The semi-automated implementation with LuNA-framework is in the middle in both senses and provides competitive performance for processing of large problems.

The work was supported by Funding Science committee of Ministry of Education and Science of Kazakhstan, grants no. AP05134651 and BR05236340.

References

1. Malyshkin, V.E., Perepelkin, V.A. LuNA Fragmented Programming System, Main Functions and Peculiarities of Run-Time Subsystem. Proceedings of the 11th International Conference on Parallel Computing Technologies, LNCS 6873. – pp. 53-61, Springer (2011)
2. Bosilca, G., Bouteiller, A., Danalis, A., Faverge, M., Herault, T., Dongarra, J. PaRSEC: exploiting heterogeneity to enhance scalability. IEEE Comput Sci Eng 15(6):36-45. DOI: 10.1007/978-3-642-23178-0_5 (2013)
3. Bosilca, G., Bouteiller, A., Danalis, A., Faverge, M., Haidar, A., Herault, T., Kurzak, J., Langou, J., Lemarinier, P., Ltaeif, H., Luszczek, P., YarKhan, A., Dongarra, J. Flexible Development of Dense Linear Algebra Algorithms on Massively Parallel Architectures with DPLASMA. Proceedings of the Workshops of the 25th IEEE International Symposium on Parallel and Distributed Processing (IPDPS 2011 Workshops), IEEE, Anchorage, Alaska, USA, 1432-1441, 16-20 May (2011)
4. Charm++. http://charm.cs.uiuc.edu (accessed 2018-05-01)
5. NAMD: Scalable molecular dynamics library. http://www.ks.uiuc.edu/Research/namd/ (accessed 2018-05-01)
6. Tikhonov, A.N., Samarsky, A.A. Equations of Mathematical Physics (in Russian). M. Nauka, 735 pp. (1977)
7. Yanenko, N.N. Method of Fractional Step for Solution of Multi-Dimensional Promlems of Mathematical Physics (in Russian). Novosibirsk, Nauka, 197 pp. (1967)
8. Sapronov, I.S., Bykov, A.N., Parallel Pipelined Algorithm (in Russian). Atom 2009, No. 44. pp. 24–25 (2009)
9. Akhmed-Zaki, D.Zh., Lebedev, D.V., Perepelkin V.A. Comparisson of efficiency of Parallel Implementation of the Tridiagonal SLAE Solver: Parallel Pipelined Method, Parallel Solver (in Russian). Vestnik KazNU, Mathematics, Mechanics, Informatics, No 3(91), Almaty, p. 75–85 (2016)
10. Chapel. https://chapel-lang.org/ (accessed 2018-05-01)
11. Joint Supercomputing Center of Russian Academy of Sciences. http://www.jscc.ru/resources/hpc/ (accessed 2018-05-01)
12. X10 programming language. http://x10-lang.org/ (accessed 2018-05-01)
13. UCX. http://www.openucx.org/ (accessed 2018-05-01)