Fragmentation of IADE Method Using LuNA System *

Norma Alias[1], Sergey Kireev[2,3]

[1] Ibnu Sina Institute, Faculty of Science, Universiti Teknologi Malaysia, Malaysia
[2] ICMMG SB RAS, Novosibirsk, Russia
[3] Novosibirsk National Research University, Russia
norma@ibnusina.utm.my, kireev@ssd.sscc.ru

Abstract. The fragmented programming system LuNA is based on the Fragmented Programming Technology. LuNA is a platform for building automatically tunable portable libraries of parallel numerical subroutines. This paper focuses on the parallel implementation of the IADE method for solving 1D partial differential equation (PDE) of parabolic type using LuNA programming system. A fragmented numerical algorithm of IADE method is designed in terms of the data-flow graph. A performance comparison of different algorithm's implementations including LuNA and Message Passing Interface are given.

Keywords: Fragmented Programming Technology, LuNA system, algorithm fragmentation, IADE method.

1 Introduction

Today's rapid development of parallel computing systems makes accumulation of a portable numerical library of parallel subroutines an important and difficult issue. A variety of parallel computer architectures including clusters, multicore and many-core CPUs, accelerators, grids and hybrid systems makes it difficult to develop a portable parallel algorithm and implement it efficiently for given hardware. The Fragmented Programming Technology (FPT) [1] is an approach to parallel programming that is aimed to solve the problem of parallel numerical library accumulation [2, 3]. It suggests to write a numerical algorithm in an architecture-independent form of a data-flow graph and to tune it to a given computer system in an automated way.

The purpose of the paper is to demonstrate the current implementation of FPT on a solution of a certain problem and to evaluate the obtained performance. The paper considers a model problem of 1D parabolic equation solution using modification of IADE method [4] as an example of FPT application. A fragmented algorithm for the considered problem was developed and implemented using fragmented programming system LuNA [5]. A performance comparison of different algorithm implementations including LuNA and MPI was made.

* The work has been supported by research grant FRGS/1/2015/TK10/UTM/02/7 and RAS Presidium Programs II.2Π/I.3-1 and II.2Π/I.4-1.

2 Fragmented Programming Technology

The FPT defines a representation of an algorithm and a process of its optimization to a certain parallel architecture. In FPT the representation of an algorithm (called "fragmented algorithm") have the following peculiarities:

– *portability* – the fragmented algorithm does not depend on a certain parallel architecture of a multicomputer,
– *automated tunability* – the fragmented algorithm is able to be semi-automatically tuned to a supercomputer of a certain class.

These properties allow to accumulate a library of parallel numerical subroutines with high portability among present and future parallel architectures. In order to satisfy these properties in FTP the following decisions were made:

– The fragmented algorithm is defined as a bipartite data-flow graph with nodes being single assignment variables (called "data fragments") and single execution operations (called "fragments of computation"). Each fragment of computation is a designation of execution of some pure function called "code fragment". Arcs in the graph correspond to data dependencies originated from numerical algorithm. So, the fragmented algorithm does not have any implementation specific elements, and is thus portable. Declarative concurrency of the algorithm representation allows to execute fragments of execution in any order that does not contradict to data dependencies.
– Data fragments of a fragmented algorithm are actually aggregates of atomic variables. The sizes of the data fragments are parameters of the algorithm. Consequently, the fragments of computation are aggregates of atomic variables and operations. On execution of a given fragmented algorithm the sizes of fragments should be tuned to characteristics of a specific parallel computer, for example, to fit a cache memory size.
– The problem of automated tuning of a fragmented algorithm to a certain parallel computer is supposed to be solved in FPT by a special execution system. In order to transfer all existing algorithms to a new supercomputer architecture a new execution system should be implemented.

LuNA programming system [5–7] is a realization of FPT for a class of multicomputers with multicore computing nodes. It comprises LuNA language and LuNA execution system. LuNA language is based on the structure of fragmented algorithm with addition of means for working with enumerated sets of fragments, for organization of structured code fragments that are similar to subroutines in common programming languages, and provides an interface with code fragments written in C++. LuNA execution system consists of a compiler, a generator and a runtime system [5]. LuNA compiler receives a fragmented algorithm written in LuNA language as input and makes common static optimizations. LuNA generator in turn makes architecture specific static optimizations. Then, LuNA runtime system executes the tuned algorithm on a certain multicomputer in semi-interpreted mode providing necessary dynamic properties such as workload balancing.

3 IADE-RB-CG Method

Iterative Alternating Decomposition Explicit (IADE) method has been used for solution of multidimensional parabolic type problems since 90-s [4, 8, 9]. Being the second-order accurate in time and fourth-order accurate in space, the IADE method is proved to be more accurate, more efficient, and has better rate of convergence than the classical fourth-order iterative methods [8]. The method is fully explicit, and this feature can be fully utilized for parallelization.

To approximate the solution of diffusion equation, the IADE scheme employs the fractional splitting resulting in a two-stage iterative process. On the first half-step of the i-th iteration the approximation solution $x^{i+\frac{1}{2}}$ is computed using values x^i, and on the second half-step the new values x^{i+1} are computed using $x^{i+\frac{1}{2}}$. In basic sequential algorithm for 1D problem solution the value $x_j^{i+\frac{1}{2}}$ depends on $x_{j-1}^{i+\frac{1}{2}}$, whereas the value x_j^{i+1} depends on x_{j+1}^{i+1} (j is a spatial index) [4]. To avoid dependency situation, several parallelization strategies are developed to construct non-overlapping subdomains [9]. In this paper, a Red-Black ordering approach was used for 1D problem, resulting in two half-sweeps for Red (even) and Black (odd) elements. In addition, a Conjugate Gradient (CG) acceleration was employed to improve convergence [9].

4 Fragmentation of IADE-RB-CG Method

A solution of 1D parabolic problem [4] is considered as an example of IADE-RB-CG method application. The process of solution is a sequence of time steps; on each step a system of linear equations with the same matrix and a new right-hand side is solved by an iterative process.

The problem of creating a fragmented algorithm consists in decomposing the algorithm into data fragments and fragments of computations with their sizes being parameters of the algorithm and in defining all the necessary dependencies between them. The sizes of fragments should be approximately equal to ease the load balancing. Therefore, the global domain is divided into a number of subdomains of equal sizes. Each subdomain contains Red and Black elements, grouped in separate data fragments to reduce the number of data dependencies. For example, the solution vector x of size M divided into m subdomains is represented in fragmented algorithm as a set of data fragments $xR_0, xR_1, \ldots, xR_{m-1}$ (for Red), and $xB_0, xB_1, \ldots, xB_{m-1}$ (for Black). The size of each data fragment (except the last ones) is $S \approx M/(2m)$.

The scheme of the fragmented algorithm of IADE-RB-CG method is shown in fig. 1a as a sequence of time steps $n = 1, \ldots, N$. Circles denote sets of data fragments, rectangles are code fragments. Black circles represent input data fragments, white ones are output and gray ones are intermediate data fragments for the time step. Code fragment "init_f" calculates right-hand side vector values in data fragments fR_n, fB_n. Code fragment "solve" calculates the solution for the next time step. Data fragments on the left side of fig. 1a hold coefficients used for calculation. Data fragments yR, yB hold the half-step solution $x^{i+\frac{1}{2}}$.

Hereinafter, the initialization phase in the algorithm representation is omitted. In fig. 1b an implementation of a “solve” code fragment is shown. It contains an iterative process continuing until convergence. Data fragments rR, rB and zR, zB hold vectors used for CG acceleration.

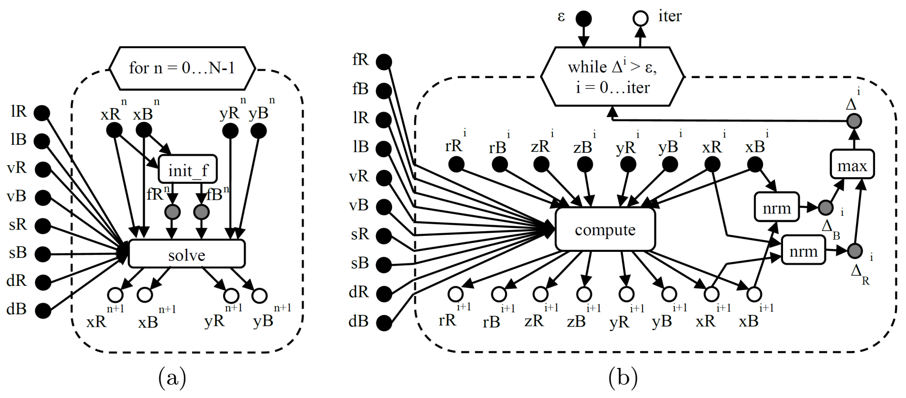

for n = 0…N-1
lR
lB
vR
vB
sR
sB
dR
dB
xR^n xB^n yR^n yB^n
init_f
fR^n fB^n
solve
xR^n+1 xB^n+1 yR^n+1 yB^n+1
(a)
ε
iter
while Δ^i > ε, i = 0…iter
fR
fB
rR^i rB^i zR^i zB^i yR^i yB^i xR^i xB^i
compute
nrm
nrm
max
Δ^i
Δ_B^i
Δ_R^i
rR^i+1 rB^i+1 zR^i+1 zB^i+1 yR^i+1 yB^i+1 xR^i+1 xB^i+1
(b)

Fig. 1. Fragmented algorithm of IADE-RB-CG method (a) and fragmented algorithm of “solve” code fragment (b)

Code fragment “compute” performs one iteration of IADE-RB-CG method (fig. 2). Notice that circles here are sets of m data fragments corresponding to m subdomains. Rectangles consequently correspond to sets of fragments of computation. Thus rectangle labeled “iadeR” denotes a set of m fragments of computation, implemented by the same code fragment called “iadeR”. The j-th fragment of computation gets j-th data fragments from the sets of input data fragments and produce j-th data fragments from the sets of output data fragments without interaction between subdomains. The same goes for the “iadeB”, “cgR”, and “cgB”. This property allows parallel execution of these fragments of computation in the case of necessary resources availability.

The only interaction between subdomains occurs in the code fragments “left” and “right” where boundaries exchange is performed (fig. 3). Since all subdomains (except the last one) has an even number of elements, only the Red elements on exchange go to the right subdomain (fig. 3a), and only the Black elements go to the left subdomain (fig. 3b).

5 Fragmented Algorithm Implementation Using LuNA Language

The fragmented algorithm was implemented with LuNA fragmented programming system using LuNA language. LuNA stands for Language for Numerical Algorithms. LuNA program is just a textual representation of a bipartite dataflow graph of an algorithm. Here, an example of LuNA subroutine corresponding to the algorithms on fig. 3a is presented.

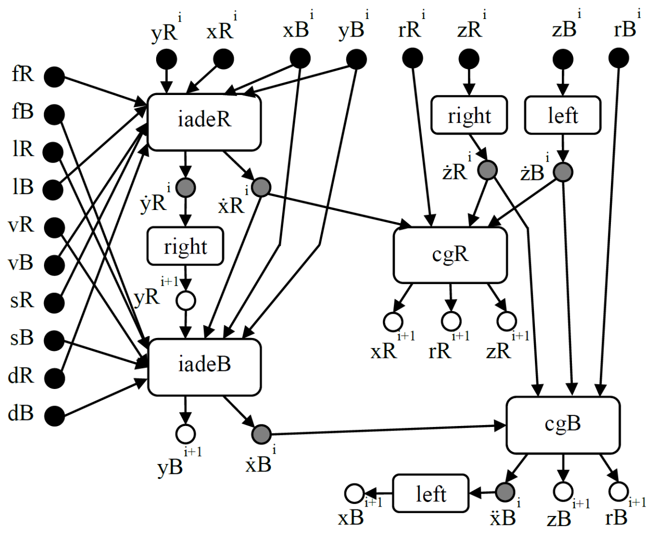


Fig. 2. Fragmented algorithm for i-th iteration of IADE-RB-CG method

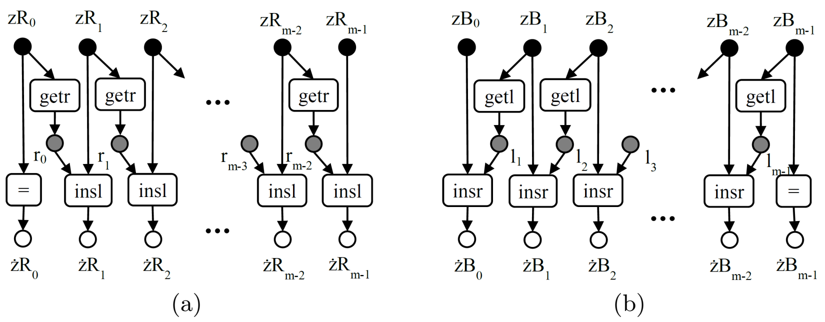


Fig. 3. Fragmented algorithms of boundaries exchange: right (a) and left (b)

```
sub send_right(#in int m, name sizeR, name zR_in, #out name zR_out)
{ df r;
  for j=0..m-2
    get_R_right_border(#in sizeR[j],zR_in[j], #out r[j]);
  copy(#in zR_in[0], #out zR_out[0]);
  for j=1..m-1
    set_R_left_border(#in sizeR[j],zR_in[j],r[j-1], #out zR_out[j]);
}
```

`get_R_right_border`, `copy` and `set_R_left_border` denote fragments of computation with input and output data fragments shown in parenthesis. The keyword `for` defines a set (unordered) of fragments of computation for a given range of values of index variable (`j`).

6 Performance Evaluation

Having such a fragmented algorithm, as presented in section 4, resource allocation and execution order control can be done automatically during execution and dynamically adjusted to available resources. Dataflow-based parallel programming systems, such as LuNA, often lack efficiency due to a high degree of non-determinism of a parallel program execution and execution overhead it causes. It is a price that is paid for the ability to avoid writing a parallel program manually and for obtaining dynamical properties of parallel execution automatically. To evaluate the performance of LuNA system a series of tests was made.

Each test run is an execution of 20 time steps of IADE-RB-CG algorithm (more than 100 executions of "compute" fragment). Average execution time of one "compute" fragment was taken as a result. Task parameters are: M - solution vector size, m - number of subdomains. Cluster MVS-10P [10] was used for the tests. Each cluster node contains 2 x 8-core Intel Xeon E5-2690 2.9 GHz (16 cores per node), 64 GB RAM, and 2 x MIC accelerators (not used in tests).

The first test evaluates the performance characteristics of LuNA runtime system depending on problem size and number of processor cores used. Currently, the LuNA runtime system is implemented as a set of MPI processes each running one or more working threads. Two variants of execution were compared:

- "Processes" - number of MPI processes is equal to the total number of processor cores used, each process running only one working thread,
- "Threads" - number of MPI processes is equal to the number of cluster nodes, each process running the same number of working threads as the processor cores used per node.

The number of subdomains here is equal to the total number of cores used. The results (fig. 4) show that the variant "Threads" runs faster, so it will be used in the following experiments. One can also see that with the current implementation of LuNA runtime system the execution time grows rapidly with increasing number of cluster nodes due to the execution overhead. The overhead is expected to be reduced in future LuNA system releases.

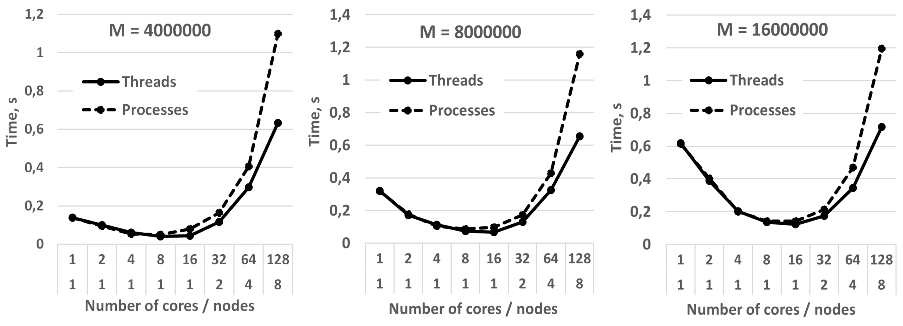


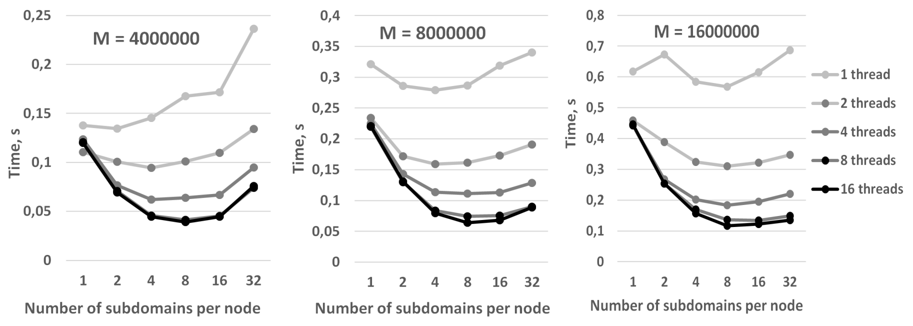
Fig. 4. LuNA runtime performance characteristics

The performance of a parallel program depends largely on the availability of work for the available computational resources, which in turn depends on the degree of algorithm fragmentation. The second test demonstrates how the execution time depends on the number of subdomains. It is known that the resulting graph should be a U-shaped with an optimal fragmentation degree giving minimal execution time [2]. Results in fig. 5 correspond to that and show optimal fragmentation degrees for different problem sizes and different computing resources.

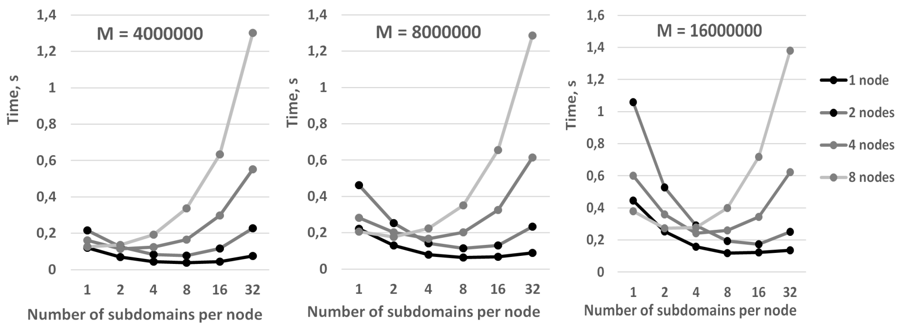
The last test compares three different implementation of the IADE-RB-CG fragmented algorithm with different degrees of the automation of parallel execution.

- "LuNA" is implementation of the algorithm in LuNA system using execution parameters from previous tests that give the best execution time for given resources. It is the most automated implementation since the LuNA runtime system makes most decisions on fragmented algorithm execution dynamically.
- "LuNA-fw" is a semi-automated implementation of the fragmented algorithm using a simple event-driven MPI-based runtime system with manually written control program. By means of the control program the programmer specifies resource allocation and data fragments' management.
- "MPI" is a manual MPI implementation of the fragmented algorithm with the least execution automation.

Results on fig. 6 show that the "LuNA" implementation has a considerable execution overhead when using several cluster nodes, and shows perfomance close to "LuNA-fw" implementation within one node. "LuNA" and "LuNA-fw" implementations both have a noticable overhead compared to the "MPI" implementation due to necessity to maintain single-assignment data fragments, which leads to redundant memory usage and hence worse cache usage.



(a)



(b)

Fig. 5. The dependence of the execution time of the algorithm on the degree of fragmentation: using one cluster node and different number of threads (a), using different number of nodes with 16 threads each (b)

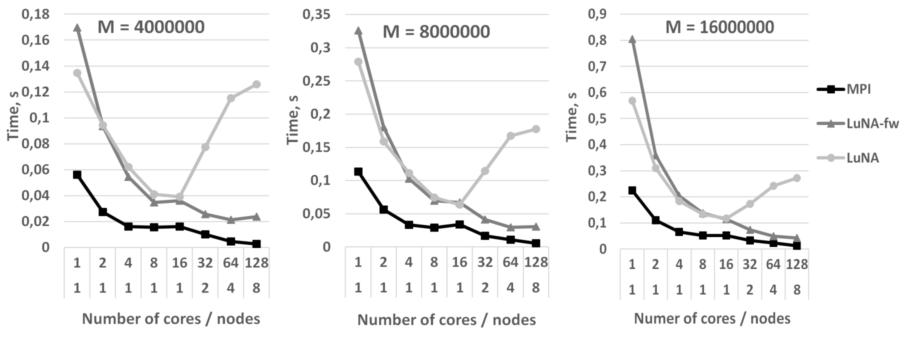


Fig. 6. Comparison of different implementations of the IADE-RB-CG fragmented algorithm

7 Conclusion

Fragmented Programming Technology (FPT) is a promising approach leading to accumulation of numerical algorithms in a portable and tunable form. A solution of a 1D problem by IADE-RB-CG method using LuNA fragmented programming system was presented as an example of FPT application. A fragmented algorithm for the considered problem is proposed.

The performance comparison of different fragmented algorithm implementations shows that LuNA system has a considerable execution overhead when using several cluster nodes, while the LuNA-fw implementation based on event-driven control program has much lower overhead. Thus, a promising direction for the future development and optimization of the LuNA system is an automated generation of a LuNA-fw-like control program.

References

1. Kraeva, M.A., Malyshkin, V.E.: Assembly Technology for Parallel Realization of Numerical Models on MIMD-Multicomputers. In the Int. Journal on Future Generation Computer Systems, Elsevier Science, NH. Vol. 17, No. 6, 755–765 (2001)
2. Kireev, S., Malyshkin, V.: Fragmentation of Numerical Algorithms for Parallel Subroutines Library. The Journal of Supercomputing, Vol. 57, Issue 2, 161–171 (2011)
3. Kireev, S., Malyshkin, V., Fujita H.: The LuNA Library of Parallel Numerical Fragmented Subroutines. PaCT-2011 proceedings, Springer, LNCS 6873, 290–301 (2011)
4. Sahimi, M.S., Ahmad, A., Bakar, A.A.: The Iterative Alternating Decomposition Explicit (IADE) method to solve the heat conduction equation. International Journal of Computer Mathematics, Vol. 47, 219-229 (1993)
5. Malyshkin, V., Perepelkin, V.: LuNA Fragmented Programming System, Main Functions and Peculiarities of Run-Time Subsystem. PaCT-2011 proceedings, Springer, LNCS 6873, 53–61 (2011)
6. Malyshkin, V., Perepelkin, V.: Optimization methods of parallel execution of numerical programs in the LuNA fragmented programming system. The Journal of Supercomputing, Vol. 61, Issue 1, 235–248 (2012)
7. Malyshkin, V., Perepelkin, V.: The PIC implementation in LuNA system of fragmented programming. The Journal of Supercomputing, Vol. 69, Issue 1, 89–97 (2014)
8. Mansor, N.A., Zulkifle, A.K., Alias, N., Hasan, M.K., Boyce, M.J.N.: The Higher Accuracy Fourth-Order IADE Algorithm. Journal of Applied Mathematics, Vol. 2013, 1–13 (2013)
9. Alias, N., Sahimi, M.S., Abdullah, A.R.: Parallel strategies for the iterative alternating decomposition explicit interpolation-conjugate gradient method in solving heat conductor equation on a distributed parallel computer systems. Proceedings of The 3rd International Conference On Numerical Analysis in Engineering, Vol. 3, 31–38 (2003)
10. Joint Supercomputer Center of the Russian Academy of Sciences. http://www.jscc.ru/eng/index.shtml (access date 12.05.2017)