Parallel Numerical Methods for Ordinary Differential Equations: a Survey

Svyatoslav I. Solodushkin[1,2] and Irina F. Iumanova[1]

[1] Ural Federal University, Yekaterinburg, Russia
[2] Krasovskii Institute of Mathematics and Mechanics, Yekaterinburg, Russia
solodushkin_s@mail.ru

Abstract. Tremendous efforts have been made in the field of parallel numerical methods for ODE. Special volumes are dedicated to the theme we are going to discuss [1]. The brief survey of some classical results and recent developments is the aim of this paper. We particulary focus on small scale parallelism across the method, but not large scale parallelism across the system.

Keywords: parallel numerical methods · parallelism across the method · small scale parallelism

1 Introduction

Modern science poses a lot of computationally intensive problems, among them weather forecast or heart modeling. It is widely believed that the only appropriate means to solve these problems is to use parallel computers. This fact dictates the necessity to elaborate special classes of numerical algorithms which are inherently parallel and designed for use on parallel computers.

In this paper we consider parallel numerical methods for initial value problem (IVP) for ordinary differential equations

$$\begin{aligned} y'(x) &= f(x, y(x)), \\ y(x_0) &= y_0, \end{aligned} \tag{1}$$

where $x \in [x_0; X]$, $y \in R^m$.

According to Gear classification [2] the means of achieving parallelism in numerical methods for IVP could belong to two main classes:

1. parallelism across the system or, that is the same, parallelism across the space;
2. parallelism across the method or, that is the same, parallelism across the time.

Methods fallen into the first group are usually based on more or less obvious techniques such as exploiting parallelism of evaluation of function f or solving linear equations which arise at each step of an otherwise standard IVP solver. For

traditional forward-step methods, e. g. explicit Runge–Kutta methods, this approach implies simple dividing the original system into a number of subsystems which are processed concurrently by separated computing nodes with interprocessor communications at the end of each step of IVP solver. It is clear that this approach is effective for high dimensional systems only. In practice such systems appear in molecular dynamics modelling, where thousands of particles should be taken into account, or after the discretization of time dependent PDE.

The second group includes numerical algorithms, which allow several simultaneous function evaluations within each step of IVP solver. These methods are suitable for multi-core processors or computers with a few processors with fast interprocessor communication which use shared memory.

The second group also includes methods which search solution for many step simultaneously, they are also known as time parallel methods and can be classified into four different groups: methods based on multiple shooting, methods based on domain decomposition and waveform relaxation, space-time multigrid methods and direct time parallel methods [3].

A dedicated survey should be devoted to parallel across the space numerical methods, and this is the reason why they are beyond the scope of present article which is focused on parallel across the time numerical methods.

2 Predictor-Corrector, Runge–Kutta and Block Type Methods

Miranker and Linger [5] were probably the first who proposed small-scale parallel method of predictor-corrector type. They mentioned that at first sight the sequential nature of the numerical methods like (2) does not permit parallel computation on all of the processors to be performed, in other words the front of computation is too narrow to take advantage of more than one processor.

$$\begin{aligned} y^p_{n+1} &= y^c_n + \tfrac{h}{2}\Big(3f(x_n, y^c_n) - f(x_{n-1}, y^c_{n-1})\Big), \\ y^c_{n+1} &= y^c_n + \tfrac{h}{2}\Big(f(x_n, y^c_n) + f(x_{n+1}, y^p_{n+1})\Big), \end{aligned} \tag{2}$$

Indeed, y^c_n must be computed before y^p_{n+1}, which in turn must be computed before y^c_{n+1} etc.

However, if the predictor is taken in another form, as represented in (3), then y^c_n and y^p_{n+1} can be evaluated in parallel by two processors.

$$\begin{aligned} y^p_{n+1} &= y^c_{n-1} + 2hf(x_n, y^p_n), \\ y^c_n &= y^c_{n-1} + \tfrac{h}{2}\Big(f(x_{n-1}, y^c_{n-1}) + f(x_n, y^p_n)\Big), \end{aligned} \tag{3}$$

Method (2) is based on couple of Adams–Bashfort and Adams–Moulton methods of second order, where the first one is used to estimate y_{n+1} which is substituted to the second one to avoid the implicity and, consequently, the necessity of solving a nonlinear equation. In (3) the predictor outruns the corrector and stands one step ahead; this made the parallel implementation possible.

As a development of this idea Miranker and Liniger showed how to systematically derive a general class of numerical integration methods which can be executed on multiple processors in parallel, and present a stability and convergence analysis for those methods. More precisely, they considered predictor-corrector methods in a PECE mode (one predicted derivative evaluation and one corrected derivative evaluation) where calculation advances s steps at a time. So, if $2s$ processors are available it is possible to arrange tasks in such a way that each processor performs either a predictor or a corrector calculation.

Continuing the research started by Mirankel Katz, Franklin and Sen [6] studied the stability properties of predictor-corrector parallel methods. Specifically, they considered the case where there are two processors, i.e. $s = 1$. They showed that for a fixed Adams–Moulton corrector the optimally stable parallel predictor (in the sense that the parallel scheme has a maximum stability interval on the negative real axis) is the Adams–Bashforth predictor shifted to the right by one integration step. Methods constructed in [6] are not A-stable.

Another type of numerical methods which possess an internal parallelism are block methods. Block method which finds solution in r points at one step is called r-dots block method. By analogy with the classification into one-step and multistep for classical methods, block methods could also be divided into these two groups. Shampine and Watts studied a family of one-step block methods, and gave one specific example (4):

$$\begin{bmatrix} y_{n+1} \\ y_{n+2} \end{bmatrix} = h \begin{bmatrix} 2/3 & -1/12 \\ 4/3 & 1/3 \end{bmatrix} \begin{bmatrix} f_{n+1} \\ f_{n+2} \end{bmatrix} + \begin{bmatrix} y_n \\ y_n \end{bmatrix} + h \begin{bmatrix} 5/12 \\ 1/3 \end{bmatrix} \begin{bmatrix} f_n \\ f_n \end{bmatrix}. \tag{4}$$

The peculiar feature of this method is the form of discretization error:

$$\varepsilon_n = \left(\frac{h^4}{24} u^{(4)}(\zeta_{n+1}),\ -\frac{h^5}{90} u^{(5)}(\zeta_{n+2}) \right)$$

Shampine and Watts took advantages of this formula which is more accurate at the end of the block than in the middle and proved (4) has global convergence like $O(h^4)$ instead of $O(h^3)$ one might expect. They also proved the A-stability of (4), so, in a sense they overcame the Dahlquist barrier which states that of the linear multistep methods the trapezoidal rule is the most accurate A-stable method.

To avoid the necessity of solving nonlinear equation which arise at each step Shampine and Watts supplied a predictor-corrector form of one-step r-dots block methods. Good prediction allow them to compute the corrector two times only rather than iterate until it is satisfied, furthermore suitable predictor-corrector scheme asymptotically gives the same result as iterating to completion. Unfortunately block implicit methods when used as a predictor-corrector combination are even not A(α)-stable.

To expand the stability region Lee and Song [8] proposed a family of explicit and implicit multistep block methods as well as block predictor corrector schemes based on couples of these methods where explicit one is used as predictor.

Feldman used collocation approach to derive implicit multistep block methods of general form [9]. He presented formulas for A-stable one-step 4-dots block implicit method and mentioned that collocation approach allow to construct A-stable one-step block methods of higher order as well. At the same time multistep methods constructed in such a way are not A-stable.

Deep analysis of predictor-corrector methods suggests a simple general paradigm for achieving parallelism across the time in traditional forward-step methods: group the stages of a the method into blocks for which all evaluations associated with each block can be performed simultaneously. This idea could be illustrated by parallel explicit and diagonally implicit Runge–Kutta (ERK and DIRK, respectively) methods in a very natural way.

An s-stage RK method has the following form

$$\begin{aligned} k_{n,i} &= f\Big(x_n + c_i h, y_n + h\sum_{j=1}^{s} a_{ij}k_{n,j}\Big), \quad i = 1, ..., s \\ y_{n+1} &= y_n + h\sum_{i=1}^{s} b_i k_{n,i}, \end{aligned} \tag{5}$$

where $b_i,\ c_i,\ a_{i,j}$ are coefficients; $a_{i,j} = 0,\ i \le j$ for explicit RK formulas and $a_{i,j} = 0,\ i < j$ and at least one $a_{ii} \neq 0$ for diagonally implicit RK formulas. Let us denote by $A = \{a)_{ij}\}$ the $s \times s$ RK matrix.

Suppose, for example, that the coefficients of explicit RK method have the zero-pattern presented in Table 1.

Table 1. Coefficients of explicit RK method

0				
×	×			
×	×	0		
×	×	×	×	
	×	×	×	×

Each arrow in the corresponding "production graph" depicted in Fig. 1, pointing from vertex i to vertex j, stands for a non-zero a_{ji}. Here the vertices 2 and 3 are independent and can be processed in parallel. The number of vertices in the longest chain of successive arrows (in this case 3) is called the number of *effective sages* [13] or *number of sequential function evaluations* [12].

In general, let us consider RK matrix A which could be partitioned (possibly after a permutation of the stages) as

$$A = \begin{bmatrix} \mathbf{D}_1 & & & & \\ A_{21} & \mathbf{D}_2 & & & \\ A_{31} & A_{32} & \mathbf{D}_3 & & \\ \vdots & \vdots & \ddots & & \\ A_{\sigma 1} & A_{\sigma 2} & \dots & A_{\sigma 2} & \mathbf{D}_\sigma \end{bmatrix},$$

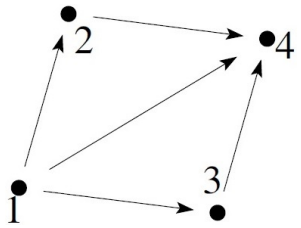

Fig. 1. Production graph

where, for ERK methods, $\mathbf{D}_l,\ l = 1, \ldots, \sigma$, are zero matrixes and, for DIRK methods, $\mathbf{D}_l,\ l = 1, \ldots, \sigma$, are (possibly different) diagonal matrixes. For any $l = 1, \ldots, \sigma$ all $k_{n,i}$ in l-th block of ERK method could be computed concurrently as soon as all $k_{n,j}$ in all previous blocks are available. Similarly, for any $l = 1, \ldots, \sigma$ each $k_{n,i}$ in l-th block of DIRK method depends on itself and the previously competed $k_{n,j}$ in blocks $1, \ldots, l-1$. Thus, all $k_{n,i}$ in l-th block could be computed in parallel by solving the system of uncoupled equations.

Unfortunately, these ERK schemes do not have a high potential of parallelism. The following theorem is a severe restriction on parallel methods [12]

Theorem 1. *The order p of the ERK method with σ sequential stages satisfies the inequality $p \leq \sigma$ for any number of available processors.*

Let us remark that stability function of these ERK methods is a polinimial and therefore they can not be A-stable.

On the other hand DIRK methods have better stability properties and allow one to achive higher order of convergency. For example, method (6) is L-stable (but not B-stable) and has a 4-th order.

1/2	1/2	0	0	0
2/3	0	2/3	0	0
1/2	-5/2	5/2	1/2	0
1/3	-5/3	4/3	0	2/3
	-1	3/2	-1	3/2

(6)

This method consists of two uncoupled blocks, i. e. $\sigma = 2$, of two equations in each, hence $k_{n,1}$ and $k_{n,2}$ could be found in parallel, after that $k_{n,3}$ and $k_{n,4}$ could be found in parallel too.

The theorem 1 motivated to pose the question whether it is always possible to find ERK method of order p using not more than p effective stages, assuming that suffucient number of processors are available, these methods are called P-optimal. For sequential ERK of order $p \leq 4$ it is possible to choose coefficient of the method in such a way that the number of stages s is equal to p, so these sequential ERK are P-optimal. At the same time even for $p \geq 5$ the number of stages is greater than p and increase rapidly, see table 2. Also the number of stages s_{min}, the number of stages S for which these RK

methods have actually been constructed, the number of effective stages S_{eff} are reported. Special techniques allow one to construct methods which are P-optimal, because some stages are performed in parallel (Table 2). So, these methods allow one to decrease the computation time with the help of parallelism.

Table 2. Stages and orders

		≤ 4	5	6	7	8	9	10
Sequential ERK	s_{min}	p	6	7	9	11	≥ 12	≥ 13
	S	p	5	6	7	8	9	10
Optimal RK	S_{eff}	p	6	7	9	11	≥ 12	≥ 13
	Num. of proc	-	3	3	4	4	5	5

One possibility of constructing P-optimal methods is by fixed point iteration. Method (5) can be interpreted as an ERK method with scheme

$$\begin{array}{c|cccccc} 0 & 0 & & & & & \\ c & A & 0 & & & & \\ c & 0 & A & 0 & & & \\ \vdots & \vdots & & \ddots & & & \\ c & 0 & 0 & 0 & A & 0 & \\ \hline & 0 & \dots & 0 & 0 & b^T & \end{array} \tag{7}$$

where σ sequential stages are performed and s processors are available is assumed. The following theorem discover the connection between the order and the number of iterations.

Theorem 2. *The parallel iterated Runge–Kutta method* (5) *in form* (7) *is of order* $p = \min(p_0, \sigma)$, *where* p_0 *denotes the order of the basic method.*

This theorem shows that the choice $\sigma = p_0$ yields P-optimal ERK methods.

The next question is the least number of processors needed to implement an optimal ERK method. Houwen and Sommejeir [13] took as basic method the s-stage Gaussian–Legendre type RK method, which has the smallest number of stages with respect to their order. This allowed them to construct the method of order $p = 2s$ which is P-optimal on s processors.

In this scheme one may use linear multistep (LM) predictors reducing the number of effective stages. First results based on LM predictors are reported by Lie [14], using a 4th-order, 2-stage Gauss–Legendre corrector and a 3rd-order Hermite extrapolation predictor. Future investigation of LM correctors in this scheme was made in [13].

In conclusion it should be said that predictor-corrector and block as well as Runge–Kutta methods are ideally suited to be used on the few cores of a multicore processor, but they do not have the potential for large scale parallelism.

3 Extrapolation Methods

Many authors note that extrapolation methods posses a high degree of parallelism and offer an extremely simple technique to obtain for generating high-order methods [15–18].

Let us consider a *basic* method of order p for integrating (1) from x_0 until $x_1 = x_0 + h$. As usual simple method of low order, e. g. explicit Euler method or Crank–Nicolson method, is taken as a basic. Denote the numerical approximation to the exact solution value $y(x_1)$ by $y(x_1;\, h)$. Let the error of approximation allows the series expansion in powers of h^q, where $q = 2$ for symmetric basic method and $q = 1$ otherwise. Let us consider a sequence of integration steps $h_i = h/i,\ i = 1, \ldots, r$, then the corresponding r-point extrapolation method is defined as follow

$$\begin{gathered} y_1 = \sum_{i=1}^{r} \gamma_i\, y(x_0 + h,\, h_i), \\ \sum_{i=1}^{r} \gamma_i = 1, \qquad \sum_{i=1}^{r} \frac{\gamma_i}{i^j} = 0,\ j = p, p+q, \ldots, p+(r-2)q. \end{gathered} \tag{8}$$

Theorem 3. *Let the basic method providing the values $y(x_1;\, h_i)$ be of order p, then the extrapolation method* (8) *has order $p + q(r-1)$.*

As soon as y_1 is computed one can perform the second step using the y_1 as a new initial value at t_1, etc. Obviously the different terms $y(x_1,\, h_i)$ in (8) could be computed in parallel. It is clear that the time needed to compute $y(x_1;, h_i)$ is proportional to i, hence to balance the load it is recommended to compute $y(x_1;\, h_1)$ and $y(x_1;\, h_r)$ on the first processor, $y(x_1;\, h_2)$ and $y(x_1;\, h_{r-1})$ on the second processor and so on. In this way $\lfloor (r+1)/2 \rfloor$ processors are used.

Based on Richardson extrapolation technique Houwen in [15] constructed method which requires less processors to be optimal that the Gauss–Legendre-based parallel iterated RK method. For example, an optimal RK method of order ten requires only three processors when using Richardson extrapolation and five processors when using predictor-corrector iteration of the tenth-order Gauss–Legendre method.

Korch et al. considered in [17] the parallel explicit extrapolation methods for high-dimensional ODEs, up to 10^8. They analyze and compare the scalability of several implementations for distributed-memory architectures which use different load balancing strategies and different loop structures. By exploiting the special structure of a large class of ODE systems, the communications costs can be reduced significantly. More then, by processing the micro-steps using a pipeline-like structure, the locality of memory references could be increased and better utilization of the cache memory can be achieved.

4 Multiple Shooting and Parareal Algorithm

According to Gander [3] time parallel time integration methods have received renewed interest over the last decade because of the advent of massively parallel

computers, which is mainly due to the clock speed limit reached on today's processors. When solving time dependent partial differential equations, the time direction is usually not used for parallelization. But when parallelization in space saturates, the time direction offers itself as a further direction for parallelization. The time direction is however special, and for evolution problems there is a causality principle: the solution later in time is affected (it is even determined) by the solution earlier in time, but not the other way round. Algorithms trying to use the time direction for parallelization must therefore be special, and take this very different property of the time dimension into account.

Below we overview only two parallel in time methods.

The basic idea of multiple shooting method proposed in [19] is to apply a shooting method which was originally developed for boundary value problems to an initial value problem (1). To do this one splits the time interval into subintervals $[x_0, x_1]$, $[x_1, x_2], \ldots,$ $[x_{n-1}, x_n]$, and then solves on each subinterval the underlying initial value problem

$$\begin{array}{lll} y_0'(x) = f(x, y_0(x)), & y_1'(x) = f(x, y_1(x)), \ldots & y_{n-1}'(x) = f(x, y_{n-1}(x)), \\ y_0(0) = y_0 & y_1(x_1) = U_1 & y_{n-1}(x_{n-1}) = U_{n-1} \end{array} \tag{9}$$

together with the matching conditions $U_1 = y_0(x_1), \ldots, U_{n-1} = y_{n-2}(x_{n-1})$, where U_i, $i = 1, \ldots, n-1$ are so called shooting parameters which are unknown. This leads to the system of non-linear equations

$$\begin{cases} U_1 - y_0(x_1) = 0 \\ U_2 - y_1(x_2) = 0 \\ \ldots \\ U_{n-1} - y_{n-1}(x_{n-1}) = 0 \end{cases}. \tag{10}$$

To determine the shooting parameters the Newton's method could be applied to this system. Fortunately the Jacobian matrix has a very special band form and could be inverted easily. The corresponding iteration process has a following form

$$\begin{aligned} U_0^{k+1} &= y_0 \\ U_{i+1}^{k+1} &= y_i(x_{i+1}, U_i^k) + \frac{\partial y_i(x_{i+1}, U_i^k)}{\partial U_i}(U_i^{k+1} - U_i^k), \quad i = 0, 1, 2, \ldots, n-2 \end{aligned} \tag{11}$$

and allow parallel implementation. Chartier and Philippe prove that (11) converges locally quadratically. Note, that (11) has a form of Bellen and Zennaro method [20] who used multiple shooting technique to parallelize discrete recurrent relations $y_{n+1} = F_{n+1}(y_n)$ and reported that the speedup of 53 for a problem with 400 time steps.

Following Lions, Maday and Turinici [21] let us explain the parareal algorithms on the simple scalar model

$$y'(x) = -ay(x), \qquad x \in [0, X], \qquad y(0) = y_0. \tag{12}$$

The solution is first approximated using Backward Euler on the time grid x_i with coarse time step Δ,

$$Y_{i+1}^1 - Y_i^1 = -a\Delta Y_{i+1}^1, \qquad Y_0^1 = y_0.$$

The approximate solution values Y_i^1 are then used to compute on each time interval $[x_i, x_{i+1}]$ exactly and in parallel the solution of

$$y_i'^1(x) = -ay_i^1(x), \qquad x \in [x_i, x_{i+1}], \qquad y_i^1(x_i) = Y_i^1.$$

After that corrections are performed that lead as to the iteration process

1. Compute the jumps $S_i^k = y_{i-1}^k(x_i) - Y_i^k$.
2. Propogate the jumps $\delta_{i+1}^k - \delta_i^k + a\Delta\delta_{i+1}^k = S_i^k, \quad \delta_0^k = 0.$
3. Set $Y_i^{k+1} = y_{i-1}^k(x_i) + \delta_i^k$ and solve in parallel

$$y_i'^{k+1}(x) = -ay_i^{k+1}(x), \qquad x \in [x_i, x_{i+1}], \qquad y_i^{k+1}(x_i) = Y_i^{k+1}.$$

Theorem 4. *[21] The parareal scheme is of order k, i. e. there exists C_k such that*

$$\|Y_i^k - y(x_i)\| + \max_{x\in[x_i,x_{i+1}]} \|y_i^k(x) - y(x)\| \le C_k\Delta^k.$$

In particular, this result means that with each iteration of the parareal algorithm, one obtains a numerical time stepping scheme which has a truncation error that is one order higher than before. So for a fixed iteration number k, one can obtain high order time integration methods that are naturally parallel. The same proposition also holds for Forward Euler. Unfortunately in both discretization schemes the stability of the higher order methods obtained with the parareal correction scheme degrades with iterations.

Lions et al. gave two numerical examples: one for a heat equation where they obtain a simulated speedup of a factor 8 with 500 processors, and one for a semi-linear advection diffusion problem where the obtained speedup was 18.

Acknowledgements This research is supported by RFBR 17-01-00033 and 17-01-00392, Russian Science Foundation (RSF) 14-35-00005. We acknowledge the support by the program 02.A03.21.0006 on 27.08.2013.

References

1. Advances in Computational Mathematics Volume 7, Issue 1–2, 1997.
2. Gear, C.W.: The parallel methods for ordinary differential equations, Tech. Rep. UIUCDCS R–87–1369, Comp. Sci. Dept., Univ. of Illinois, Urbana, IL, 1987.
3. Gander, M.J.: 50 Years of Time Parallel Time Integration. In: in Multiple Shooting and Time Domain Decomposition, Springer, 2015.
4. Jackson, K.R.: A Survey of Parallel Numerical Methods for Initial Value Problems for Ordinary Differential Equations. In: IEEE Transactions on Magnetics, 1991. Vol. 27. No. 5, P. 3792–3797.

5. Miranker, W.L., Linger, W.: Parallel methods for the numerical integration of ordinary differential equations. In: Math. Comp., 1967. Vol. 21. P. 303–320.
6. Katz, I.N., Franklin, M.A., Sen, A.: Optimally stable parallel predictors for Adams-Moulton correctors. In: Camp. and Moth. with Appl., 1977. Vol 3. pp 217–233.
7. Shampine, L.F., Watts, H.A.: Block Implicit One-Step Methods. In: Math. Comp., 1969. V. 23. P. 731–740.
8. Lee, M.G., Song, R.W.: Computation of Stability Region of some Block Methods and Their Application to Numerical Solutions of ODEs. In: Proceedings of XIV Baikal International Conference, Baikal, Siberia, Russia, July 2008.
9. Feldman, L.P., Nazarova, I.A.: Parallel algorithms for the numerical decision of Caushy's problem for ordinary differential equations systems. In: Mathematical modelling, 2006. Vol. 18. No. 9. P. 17–31.
10. Iserles, A., Nørsett, S.P.: On the theory of parallel Runge–Kutta methods. IMA J. Numer. Anal., 1990. Vol.10. P. 463–488.
11. Jackson, K.R., Nørsett, S.P.: The potential of parallelism in Runge–Kutta methods. Part 1: RK formulas in standard form. Report 239/90, 1992.
12. Hairer, E., Wanner, G., Nørsett, S.P.: Solving Ordinary Differential Equations I, 1993.
13. van Der Houwen, P.J., Sommeijer, B.P.: Parallel iteration of high-order Runge-Kutta methods with stepsize control. In: Journal of Computational and Applied Mathematics, 1990. Volume 29. Issue 1. P. 111–127.
14. Lie, I.: Some aspects of parallel Runge-Kutta methods, Report No. 3/87, University of Trondheim, Division Numerical Mathematics, 1988.
15. van Der Houwen, P.J.: Parallel step-by-step methods. In: Applied Numerical Mathematics, 1993. Volume 29. Issue 1. P. 69–81.
16. Rauber, T., Runger, G.: Load balancing schemes for extrapolation methods. In: Concurrency: Practice and Experience, 1997. Volume 9. Issue 3. P. 181–202.
17. Korch, M., Rauber, T., Scholtes, C.: Scalability and locality of extrapolation methods on large parallel systems. In: Euro-Par 2010, Part II, LNCS 6272, 2010. P. 65–76.
18. Ketcheson, D., bin Waheed, U.: A comparison of high-order explicit Runge–Kutta, extrapolation, and deferred correction methods in serial and parallel. In: Communications in Applied Mathematics and Computational Science, 2014, Volume 9. Issue 2. P. 175–200.
19. Chartier, P., Philippe, B.A.: parallel shooting technique for solving dissipative ODEs. In: Computing, 1993. Volume 51. Issue 3. P. 209–236.
20. Bellen, A., Zennaro, M.: Parallel algorithms for initial-value problems for difference and differential equations. In: J. Comput. Appl. Math., 1989. Volume 25. Issue 3. P. 341-350.
21. Lions, J.-L., Maday, Y., Turinici, G.: A parareal in time discretization of PDEs. In: C.R. Acad. Sci. Paris, Serie I, 2001332, 2001. P. 661–668.