Electronic Preprint for Journal of Information Processing Vol.27

Regular Paper

Time Segment Correction Method for Parallel Time Integration

Akihiro Fujii[1,a)] Shigeo Kaneko[2,b)] Teruo Tanaka[1,c)] Takeshi Iwashita[3,d)]

Received: April 3, 2019, Accepted: July 25, 2019

Abstract: Parallel time integration methods provide the time integral simulation with parallelism in time direction and enhance the performance of the applications on parallel machines. Many parallel time integration methods apply re-discretization with enlarged time step width to propagate prior time step information to later time steps with low calculation cost. However, this re-discretization tends to create an unstable time integral problem and causes the performance degradation depending on the length of the time step width. Therefore, we paid attention to the correction scheme involved in the (parallel) TP-EEC method that uses Jacobian matrices of original simulation problem to propagate the prior time step information to later time steps very quickly. This paper showed our parallel time integration approach using a correction method similar to the TP-EEC method, and proposed the efficient implementation methods with pipeplined coarse level correction. The performance results for simple non-linear time step simulation show our method performed well and was faster than other representative parallel time integration scheme.

Keywords: parallel time integration, non-linear problem

1. Introduction

Our target problem is non-linear time evolution simulations that appear in many kinds of scientific problems. We assume implicit time integration methods and discretization with grids such as finite difference or finite element method. For these kinds of problems, supercomputers usually help to accelerate them. Performance levels of supercomputers are known to have been growing dramatically and exponentially for more than 2 decades. But the performance growth of supercomputers has been gained by increasing parallelism especially in recent years. There are many supercomputers registered in TOP500 list [7] that have more than 1 million cores. Therefore, simulation codes must have much parallelism to provide work loads for many cores.

The time evolution simulations usually solve the unknown variables of the current time step from the relationships between previous time step and the current time step. Then it moves the current time step forward to the next time step. Since the time step is forwarded sequentially, the calculation cost will increase in proportion to the number of time steps. There are many applications that have less than 1 million unknown variables at each time step but that have to proceed more time steps than 1,000 steps. These simulations cannot gain enough parallelism for supercomputers with millions of cores by the usual step-by-step time integration. In order to exploit more parallelism, studies of parallel time integration attract many researchers' attention.

Many parallel time integration algorithms such as parareal [1], [6] and MGRIT [2] were proposed. They usually cut dependencies in the time stepping direction to introduce parallelism. They update variables over all time steps iteratively until the values of variables satisfy the governing relationship equations between neighboring timesteps. The norm of the residuals over all time steps can be used as an indicator for the correctness of the solution. A convergence criterion is set as this norm of the residuals over all time steps. In order to accelerate the convergence, parallel time integration algorithms usually create the "coarse grid" simulation with enlarged time step width to propagate prior time step information to later time steps at a fast pace but with low accuracy. However, problems with enlarged time step width tend to cause instability, which leads to difficulty with the time integration.

On the other hand, the parallel time integration method, parallel TP-EEC method (Time-Periodic Explicit Error Correction) [10], that does not introduce re-discretization with enlarged time step width was proposed for time-periodic nonlinear magnetic field problems. It uses the Jacobian matrix information to calculate rough solution correction. The algorithm has been already used for real application in the industry. This paper extends the correction scheme of the TP-EEC method [9] to an ordinary nonlinear time integral simulation problem, and we call it TSC (time segment correction) method. We studied its implementation method and checked its effectiveness in comparison with the MGRIT method.

2. Related Works

This section introduces the parareal method, MGRIT method

[1] Kogakuin University, Shinjuku, Tokyo 163–8677, Japan
[2] Ark Infomation Systems, Chiyoda, Tokyo 102–0076, Japan
[3] Hokkaido University, Sapporo, Hokkaido 060–0811, Japan
[a)] fujii@cc.kogakuin.ac.jp
[b)] kaneko.shigeo@ark-info-sys.co.jp
[c)] teru@cc.kogakuin.ac.jp
[d)] iwashita@iic.hokudai.ac.jp

and (parallel) TP-EEC method initially. Then we explain the features of the TSC method.

The parareal method [1], [6] uses a time integral function $F(t_{k+1}, t_k, u_k)$ that calculates the integral from t_k time step variables u_k to t_{k+1} time step variables accurately, and uses a function $G(t_{k+1}, t_k, u_k)$ that calculates it approximately with low calculation cost. Although time integral is ordinarily calculated by calling the function F at each time step in the order of time sequence, Parareal applies the function F in parallel by ignoring the dependency in the time sequence and corrects the error using the function G. Here, u_k^{old}, $k = 0, \ldots, N_t$ is assumed to be the approximate solution of time step k. Superscript "old" means it is before the parareal iteration. Updated values of parareal iteration are stored in variables with superscript "new". N_t is the last time step number. u_k^{old}, $k = 0, \ldots, N_t$ will be updated by the following equation.

$$u_{k+1}^{new} = F(t_{k+1}, t_k, u_k^{old}) + G(t_{k+1}, t_k, u_k^{new}) - G(t_{k+1}, t_k, u_k^{old})$$

In the parareal iteration, F function can be called in parallel, because the old superscript variable u_k^{old} is used as an initial point of the time integration. Then it calls the functions G to correct the initial point difference between u_k^{old} and u_k^{new}. The function call of G with u_k^{new} has time step dependency, and it must be called in the order of time steps.

This parareal iteration repeatedly updates the solution, and makes it reach convergence. Although dominant part of the computation becomes function F, it can be calculated in parallel. Since the time integral functions F and G are not specified on how to construct those functions, many methods of parallel time integration can be regarded as one of the parareal methods.

The MGRIT method [2], [3], [12] is a multi-grid algorithm that can be seen as one of parareal methods. Multigrid algorithm has a fine grid problem, a coarse grid problem, and a smoother. Here, we introduce them in MGRIT method.

- A fine grid problem corresponds to the given time integral simulation problem.
- Coarse grid problems are constructed by re-discretization with coarse time step points. Thus the time step width is enlarged from that of the fine level.
- Smoothers make solution reaching convergence in iterative process. They cut the dependency in every fixed number of time steps and calculate time integration in parallel. Repeating this parallel updates makes the initial time step information propagate to the end time step, and finally, the solution reaches convergence.

One cycle of MGRIT is the same as ordinary multigrid methods. At first, a smoother is applied at the fine level. Then error is corrected by solving the coarse level problem. At the end of the cycle, a smoother is applied again at the fine level. Although this explanation is about 2-level methods, the number of levels can be increased by applying the process recursively. If the user code has routines of time integration for arbitrary time step width, MGRIT can be incorporated to the user application without changing the user codes, which is called non-intrusiveness in MGRIT papers [2]. The algorithm is introduced in the next section.

The parallel TP-EEC [10], [11] method is proposed for parallel time integration of time-periodic nonlinear magnetic field problems. It uses the property of time-periodicity and accelerates the convergence to the steady state solution. The method divides the time steps among MPI processes. Each process calculates the time evolution in parallel. The error between processes is corrected by a linear equation calculated from Jacobian matrices of all time steps. This method does not have a re-discretization process with enlarged time step width for coarse level correction.

This paper considers how to extend the parallel TP-EEC method to the non-periodic time evolution problem. We call the method Time segment correction (TSC) method. The TSC method does not have a re-discretization process as with the parallel TP-EEC method. The TSC method corrects the error using Jacobian matrices of all time steps. The concept of introducing small sized linear equation after linearlization with Jacobian matrices is the same as is used in the parallel TP-EEC method. However, as far as authors know, there are no research papers that apply this concept to solve the non-periodic time evolusion problems. In addition, the parallel TP-EEC method divides unknowns over one period of a time-periodic problem by the number of processes, and approximates them with one time step unknowns at the coarse level. Therefore, we extend the parallel TP-EEC method as the TSC method in this paper.

Since the TSC method is applied to non-periodic problems unlike the parallel TP-EEC method, Jacobian matrices of all time steps become a block lower triangular matrix problem, which is used for solution correction. This lower triangular structure allows the pipeline execution of corrections that are introduced in Section 5. The TSC method has also non-intrusiveness as with MGRIT. It uses only Jacobian matrices at the fine level for solution correction. Therefore the method does not require changing the user code if it can output Jacobian matrices.

3. MGRIT

This section explains problem setting and step-by-step time integration with the Newton-Raphson method to begin with. Then, the MGRIT method that parallelizes the time integration process is described.

The solution vector and the right hand side vector at the i-th time step are denoted as u_i and g_i, respectively. This paper assumes that the relationship of the i-th and $(i-1)$-th time step can be expressed as Eq. (1). A is non-linear operator with 2 arguments of i-th and $(i-1)$-th time step variables, u_i and u_{i-1}. u_{i-1} in Eq. (1) is given value, and u_i is unknown variables to be fixed by the equation. N_t is the end time step number. In the Newton-Raphson method it solves linear problem of Eq. (2) at first, where u_i^0 means initial value of u_i, and Jacobian matrix is $J_i = \frac{\partial A}{\partial u_i}(u_i^0, u_{i-1})$. Then, it assigns u_i to u_i^0, and it solves Eq. (2) repeatedly until it converges.

$$A(u_i, u_{i-1}) = g_i, \quad i = 1, \ldots, N_t \tag{1}$$

$$u_i = u_i^0 - J_i^{-1}(g_i - A(u_i^0, u_{i-1})) \tag{2}$$

In step-by-step time integration, this process solves one-time step integration, and this is repeatedly applied until it reaches the end

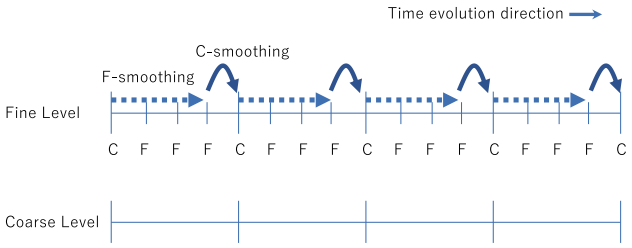

Fig. 1 C points, F points, and Smoothers.

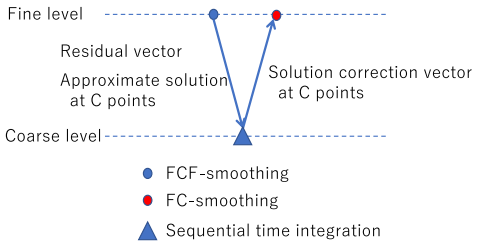

Fig. 2 Cycle shape: MGRIT.

time step.

The MGRIT method exploits parallelism by cutting the dependency in time evolution direction, and calculates time integration from the appropriate initial value. Here, F-smoothing and C-smoothing are described. All time steps are labeled as F or C as shown in the Fine level of **Fig. 1**. C points are the time steps that are left on the coarser level. Thus, in the case of Fig. 1, the time step width at the coarser level will be 4 times larger than that of the original problem setting. This ratio is called coarse to fine ratio in this paper. F-smoothing calculates time integration from each C time steps. It updates the variables at F time steps. In Fig. 1, 4 groups of F time steps are updated group by group in parallel, which is shown as 4 dotted arrows. Next, C smoothing updates variables at C time steps from F point that is previous to the C point, which is shown as 4 solid arrows between F and C time steps.

When F-smoothing and C-smoothing (FC-smoothing) is applied, the information of the initial condition propagates m time steps, where m is the coarse to fine ratio. Thus, FC-smoothing can make the solution convergent by repeating itself N_t/m times. N_t is the number of time steps of the problem. Since FC-smoothing has the same degree of parallelism of N_t/m, it is not possible to speed up the step-by-step time integration by FC-smoothing, even if it is fully parallelized.

In order to speed up the propagation to future time steps, MGRIT generates the coarse problem by enlarging the time step width m times. At the coarse problem, only the time steps labeled with C at Fine level are calculated. By solving the coarse problem with step-by-step time integration, coarse approximate solution propagates to the future time steps fast. The 2-level MGRIT method is calculated in the following procedure.

step 1 F-smoothing, C-smoothing, and F-smoothing is applied to the problem in order, which is called FCF-smoothing.

step 2 The solution is copied to u_i^0 as in Eq. (3). It calculates the residual each C point as in Eq. (4).

$$u_i^0 = u_i \tag{3}$$

$$r_i = g_i - A(u_i^0, u_{i-1}) \tag{4}$$

step 3 It copies the solution values at C points to coarse level solution vector $u^{\Delta,0}$. Then, the right hand side vector g_I^Δ is calculated by Eq. (6) . The superscript Δ means that it is a coarse level variable. Coarse level index I corresponds to the fine level index i in the following equations. A^Δ operator is obtained by re-discretization with enlarged time step width.

$$u_I^{\Delta,0} = u_i,\ i \in Cpoints \tag{5}$$

$$g_I^\Delta = r_i + A^\Delta(u_I^{\Delta,0}, u_{I-1}^{\Delta,0}) \tag{6}$$

step 4 It solves coarse level problem Eq. (7) by step-by-step time integration from the initial time step.

$$A^\Delta(u_I^\Delta, u_{I-1}^\Delta) = g_I^\Delta \tag{7}$$

step 5 The C points' solution vector is updated using coarse level solution of u_I^Δ as in Eq. (8).

$$u_i = u_i^0 + (u_I^\Delta - u_I^{\Delta,0}), i \in Cpoints \tag{8}$$

step 6 Since step 5 updates only C points, F smoothing must be applied for all variables to be updated. In this paper, FC smoothing is applied in order to connect the pre-smoothing of step 1. It goes to step 1 until it converges.

Step 1 and Step 6 list typically used smoother settings. The optimal smoother setting depends on the problem and computing environment that specifies the upper bound of parallelism. You can change the type of smoothers accordingly. As for the multigrid scheme for nonlinear problems, it uses a full approximation scheme that determines the coarse level correction method as in Eq. (6), Eq. (7) and Eq. (8). The shape of the MGRIT cycle in this paper is shown in **Fig. 2**. The dashed line shows the fine and coarse levels. Calculation at the fine level uses the original time step width for time integration, and calculation at the coarse level uses the enlarged time step width. Circles and a triangle correspond to time integration operations, and arrows show data movements and the order of calculation.

4. Time Segment Correction Method

In time evolution or time periodic problems, the method that uses projection matrix (interpolation matrix) described in the next Section 4.1 is called Time Segment Correction (TSC) method in this paper. The TSC method calculates a linear coarse level problem using both Jacobian matrices of all time steps and projection matrices. Therefore, its coarse level correction is different from that of MGRIT which uses the re-discretized coarse level nonlinear problem. This section explains the coarse level correction mechanism and then describes an iteration procedure of the TSC method.

4.1 Coarse Level Correction

Governing equation Eq. (1) can be written as the following non-linear simultaneous equations.

$$\begin{cases} u_0 = g_0 \\ A(u_1, u_0) = g_1 \\ A(u_2, u_1) = g_2 \\ \quad \vdots \\ A(u_{N_t}, u_{N_t-1}) = g_{N_t} \end{cases} \tag{9}$$

The TSC method considers a linear equation with variables of all time steps by Newton-Raphson method as follows.

$$L\mathbf{\Delta u} = \mathbf{r} \tag{10}$$

The bold font is used to show a vector of variables over all time steps in this paper. The matrix L can be regarded as the Jacobian matrix over all the time steps. The solution vector $\mathbf{\Delta u}$ in Eq. (10) is used to update the solution $\mathbf{u} \leftarrow \mathbf{u^0} + \mathbf{\Delta u}$ in Newton Raphthon method. $\mathbf{\Delta u}$ is the difference from the initial aproximate values $\mathbf{u^0}$.

For example, the linear equation with variables of $\Delta u_0..\Delta u_5$ over 6 time steps is described as in Eq. (11), when the relation of variables of i-th and $(i-1)$-th time steps are expressed as Eq. (1), where J_i and K_i is defined as $\frac{\partial A}{\partial u_i}(u_i^0, u_{i-1}^0)$ and $\frac{\partial A}{\partial u_{i-1}}(u_i^0, u_{i-1}^0)$, respectively.

$$\begin{pmatrix} J_0 & & & & & \\ K_1 & J_1 & & & & \\ & K_2 & J_2 & & & \\ & & K_3 & J_3 & & \\ & & & K_4 & J_4 & \\ & & & & K_5 & J_5 \end{pmatrix} \begin{pmatrix} \Delta u_0 \\ \Delta u_1 \\ \Delta u_2 \\ \Delta u_3 \\ \Delta u_4 \\ \Delta u_5 \end{pmatrix} = \begin{pmatrix} r_0 \\ r_1 \\ r_2 \\ r_3 \\ r_4 \\ r_5 \end{pmatrix} \tag{11}$$

Equation (11) linearly approximates the non-linear relationship of variables of all time steps.

The coarse level matrix L_c is generated as 3 matrix product $B^T LB$, where the matrix B is the prolongation matrix that projects coarse level variables to fine level variables. The matrix B^T corresponds to restriction operation. Thus the coarse level equation becomes $L_c\mathbf{u_c} = \mathbf{r_c}$, where L_c, $\mathbf{u_c}$, and $\mathbf{r_c}$ are coarse level matrix, coarse level correction vector, and coarse level right hand vector. $\mathbf{r_c}$ is calculated by $B^T\mathbf{r}$, where $\mathbf{r}$ is the residual vector at fine level. Therefore, the solution vector $\mathbf{u}$ at the fine level is corrected by the following operation, similarly as an ordinary multigrid method for linear problems.

$$\mathbf{u} \leftarrow \mathbf{u} + B(L_c^{-1}\mathbf{r_c}) \tag{12}$$

The prolongation matrix B can be constructed in many ways. This paper proposes to use the prolongation matrice as in Eq. (13). It shows examples of prolongation matrix B for 6 time step analysis as in Eq. (11).

$$\text{1st-order } B_1 = \begin{pmatrix} I \\ I \\ I \\ I \\ I \\ I \end{pmatrix}, \quad \text{2nd-order } B_2 = \begin{pmatrix} I & \\ I & \\ I & \\ & I \\ & I \\ & I \end{pmatrix} \tag{13}$$

I in Eq. (13) is the identity matrix with N_s rows, where N_s represents the number of unknown variables at each time step. By aligning identity matrices at the same column of the matrix B, coarse level matrix $B^T LB$ can approximate the constant or low frequency mode of the solution in the time direction. When we use first-order B_1 as prolongation matrix B, the coarse level problem size becomes the size of one time step variables, N_s. It corresponds to the correction of constant mode for all time steps. If we use second-order B_2 as prolongation matrix B, then the coarse level problem size becomes $2N_s$. It corresponds to the constant mode correction for the first half and the second half time steps. As for the sparsity of the coarse level matrix $B^T LB$, it has the same sparsity patterns of the matrix L in many cases. It can be calculated with low calculation cost just by adding elements of the matrix L like additive correction multigrid method [5]. For example, $B_2^T LB_2$ becomes as follows, when the matrix L is assumed to be the matrix in the Eq. (11).

$$B_2^T LB_2 = \begin{pmatrix} J_0 + J_1 + J_2 + K_1 + K_2 & \\ K_3 & J_3 + J_4 + J_5 + K_4 + K_5 \end{pmatrix}$$

The n-th order matrix B_n has n columns of identity matrices. It means that all time step variables are represented with n time step sized variables at the coarse level. Since identity matrices are aligned contiguously at each column, it divides all time steps into n blocks of contiguous time steps. Each block variables are reduced to one time step variables at the coarse level. In this paper, we allocated each block a process so that each blocked time steps can be calculated in parallel. Although it is possible to allocate multiple blocks to one process, we consider it may lead to more dependency cuts between contiguous time steps than is needed by the degree of parallelism. The optimized relationship between the order n of B_n and parallelism will be investigated in the future.

It is possible to consider a multi-level correction scheme. The additive Schwarz type multi-level correction scheme is given by Eq. (14), where $\mathbf{r_i}$ is calculated as restricted residual $B_i^T\mathbf{r}$.

$$\mathbf{u} \leftarrow \mathbf{u} + \sum_{i=1,..l} B_i((B_i^T LB_i)^{-1}\mathbf{r_i}) \tag{14}$$

This scheme uses all levels at the same time and calculates the average correction vector. $(B_i^T LB_i)^{-1}$ can be calculated by some smoothing method instead of direct method for lower calculation cost. As a first step, this paper evaluates the TSC method as two level method.

4.2 TSC iteration

The previous subsection describes how to generate a coarse level equation using an interpolation matrix B. This subsection explains two-level TSC iteration procedure.

step 1 TSC method applies smoothers in time direction with initial approximate solution. Here, we used the same type smoother as MGRIT. Smoother setting is described in Section 5.

step 2 It calcuates Jacobian matrix L that was introduced in the previous subsection. It generates coarse level problem matrix equation

$$L_c\mathbf{u_c} = B^T\mathbf{r},$$

where L_c is $B^T LB$, and $\mathbf{r}$ is the residual vector for all time steps, that is, a vector of r_i expressed in Eq. (4). Since the matrix L is based on Jacobian matrix depending on the approximate solution, coarse level matrix L_c must be updated every iteration.

step 3 It gets the coarse level correction vector $\mathbf{u_c}$ by solving coarse level equation.

step 4 It applies correction vector to the solution vector as in $\mathbf{u} \leftarrow \mathbf{u} + B\mathbf{u_c}$.

step 5 It applies the smoother again. Then it goes to step 1 until it reaches the convergence. Here, the smoother applied after coarse level correction is called post smoother.

Coarse level matrix L_c has the similar sparsity pattern as the matrix L as in Eq. (11), and it becomes block lower triangular form. Therefore the coarse level equation can be easily solved by blocked forward substitution.

5. Efficient Implementation of TSC Method

This section describes the efficient implementation of the TSC method in comparison with MGRIT. The smoother and the coarse level correction is mainly described. For explanation, C-points in TSC method are supposed to be set on the initial time step of blocked time steps. A C-point and the following F-points constitute the blocked time steps. Fig. 1 also shows the situation of the TSC method with four-th order prolongation matrix. It has 4 C-F time step blocks. The TSC method uses this C-F labeling for smoothers. At the coarse level of the TSC method, a C-F time step block is reduced to one time step variables.

The coarse level correction of MGRIT updates only C labeled points at the fine level. Therefore, F smoothing must be applied after the coarse grid correction. On the other hand, the TSC method corrects all time step variables by coarse level correction. Therefore, the TSC method has more options on smoothers than MGRIT. The TSC method in this paper applies F smoothing before it goes to the coarse level, and applies CFC smoothing after coarse level correction. Although we can select some other smoother setting, this F and CFC smoother is expected to be one of the lightest cost smoother setting. It requires only FCFC smoothing for one cycle.

Next coarse level correction of the TSC method is described. At first, data distribution with 4 processes in parallel in time is described in **Fig. 3**. Unknown variables over all time steps can be shown as a rectangular shape with axes of space and time directions. For simplicity, space direction parallelism is not used in the figure. In time direction, unknowns are divided in block distribution. Time step blocks are assumed to be allocated to the processes in the order of the rank number. Rank 0 process has the initial time step block variables, and the last process, rank 3 process has the last time step block.

The coarse level problem of the TSC method is a linear problem, and its matrix is block lower triangular form, if our proposed prolongation matrix B is used. It is because contiguous time step variables are aggregated at the coarse level, and because time step information does not depend on the future time step variables. Thus, forward substitution is applied here at step 3 described in Section 4.2, and the solution of the coarse level equation is calculated in the order of time steps, which is the same order of the rank number.

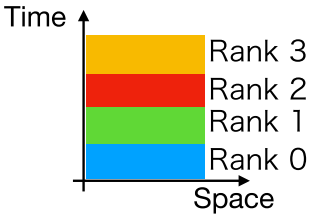

Fig. 3 Data distribution in parallel in time with 4 processes.

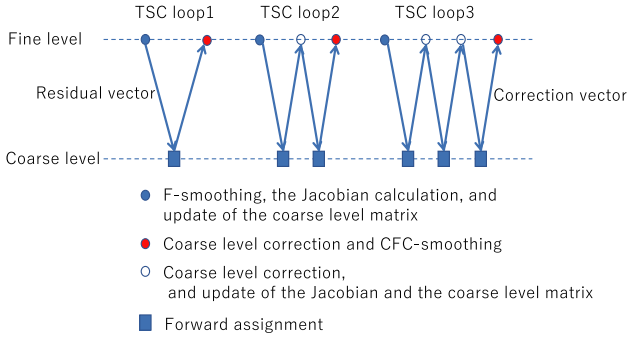

Fig. 4 Cycle shapes: TSC loop1, 2 and 3.

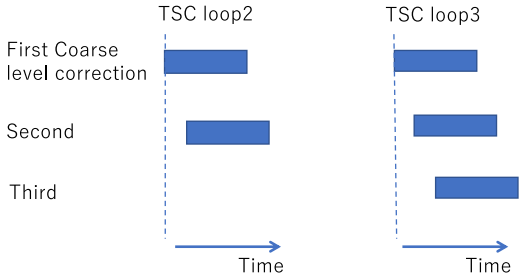

Fig. 5 Simplified image of coarse level correction: TSC loop2 and 3.

To improve the convergence, repeated coarse level correction would be considered in one TSC iteration. In that case, step 2 to step 4 will be repeated. Rank 0 process will get the correction vector at first in step 3, and it can proceed to step 4 without waiting for other processes to calculate step 3. Since all calculation of step 2 to step 4 does not depend on the future time step variables, the steps can be repeated in pipeline fashion.

We call the TSC iteration that applies coarse level correction once, which is described in Section 4.2, TSC loop1. The TSC iterations that apply coarse level correction twice and three times before post smoothing are called TSC loop2 and 3 respectively in this paper. **Figure 4** describes the cycles of TSC loop1, 2 and 3. Blue circles corresponds to step 1 and 2 in the TSC iteration. White circles correspond to step 4 and step 2. Blue squares are step 3. Red circles are step 4 and step 5. **Figure 5** shows the simplified image of coarse level calculation time of TSC loop2 and 3. The execution time of coarse level correction of TSC loop1 corresponds to First coarse level correction in the figure.

6. Numerical Tests and Discussion

6.1 Numerical Tests

This section investigates the effectiveness of the TSC method for the non-linear heat diffusion problem. The TSC methods were compared with the MGRIT method and step-by-step time inte-

gration. Although the heat diffusion problem is simple, it is used for performance evaluation of parallel time integration in many cases. For example, Gahvari et al. [4] used similar heat diffusion problem for evaluation of time space parallel solvers.

Problem setting is listed as follows.

- 2 Dimensional non-linear heat diffusion problem

$$\frac{\partial T}{\partial t} = \frac{\partial}{\partial x}\left(k(T)\frac{\partial T}{\partial x}\right) + \frac{\partial}{\partial y}\left(k(T)\frac{\partial T}{\partial y}\right)$$

  - Heat diffusion coefficient $k(T)$ is set as non-linear function depending on the temperature T. This experiment uses diffusion coefficients of steel as in **Fig. 6**. We used degree 2 polynomial least squares fitting over several diffusion coefficients data. The problem is discretized in the following equation using centered difference in right hand side terms and forward difference in the left side term.

$$\frac{\partial T}{\partial t} = k(T)\frac{\partial^2 T}{\partial x^2} + \frac{\partial k(T)}{\partial T}\cdot\left(\frac{\partial T}{\partial x}\right)^2 + k(T)\frac{\partial^2 T}{\partial y^2} + \frac{\partial k(T)}{\partial T}\cdot\left(\frac{\partial T}{\partial y}\right)^2$$

    The problem domain is assumed to be square of 50 cm by 50 cm, and time step width is 0.1 second. As an initial condition, center point of the object was set at 1000 k and the dirichlet boundary condition was set at 0 k as shown in **Fig. 7**.
  - Newton-Raphson method is used for solving nonlinear equation in the backward Euler method. The Jacobian matrix linear problem is solved by direct method.
- Parameter setting and Computing environment
  - Parallelization in time integration is done by Flat MPI with 64 processes. Reedbush-U Supercomputer [8] at Tokyo University was used.
  - 2 dimensional space size is $50 \times 50 = 2{,}500$ points.
  - The number of time steps N_t is between 256 and 16384. 7 different sized problems in time direction were measured.
  - Step-by-step time integration program has the convergence criterion that is used for termination of Newton iteration at each time step. It is specified by 2 norm of residual vector of one time step variables. In **Table 1**, this criterion is written in "one time step" row. As for parallel time integration methods like MGRIT or TSC, all time step variables are updated repleatedly. Therefore they need convergence criterion for all time step variables in addition to one time step convergence criterion. This criterion is written as "all time steps" row in Table 1. The convergence criterion is set so that solution would have almost the same quality between step-by-step time integration method and parallel time integration methods.
  - Coarse to Fine Ratio C_r setting
    - Since the TSC method generates smaller sized problem as coarse level by aggregating multiple time step information, the ratio between coarse level problem size and original problem size N_t can be specified. Here, we set the coarse level problem size as small as 64 time step variables so that each time step variables can be dealt with by each process. Thus the coarse to Fine ratio C_r is set at $N_t/64$, which is just from the number of processes.
    - MGRIT re-discretizes the problem with different time step width at the coarse level. Since the performance strongly depends on the time step width at the coarse level, the highest performance is chosen among several cases with different coarse to fine ratio C_r settings.

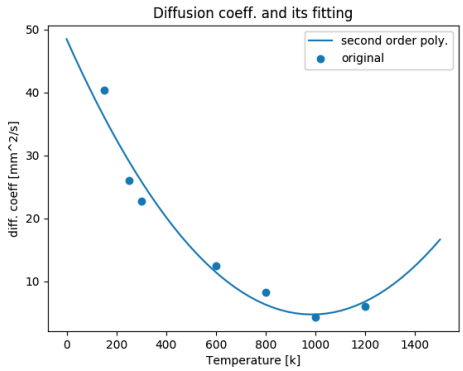

Fig. 6 Diffusion coefficient.

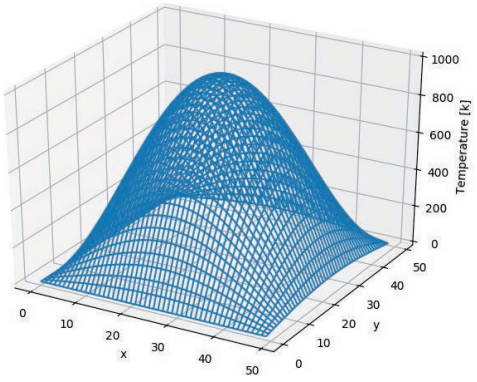

Fig. 7 Temperature distribution at initial time step: T=1000$sin(\pi x/50) \cdot sin(\pi y/50)$.

Table 1 Convergence criterion.

2 norm of residual	step-by-step integration	MGRIT or TSC
one time step	10^{-7}	10^{-8}
all time steps		$10^{-7}\sqrt{N_t}$

The results are shown in **Fig. 8**, and Tables 2 and 3. Figure 8 shows the execution time of each method when the problem size corresponding to the number of time steps is enlarged. From the figure, step-by-step time integration takes the execution time in proportional to the number of time steps from 256 time steps to 2,048 time steps. Since we could not allocate computing resources to one process execution more than 30 minutes, the execution time of more then 2,048 time steps is extrapolated with dashed line. It shows that MGRIT shortens the execution time in comparison with the step-by-step method for problems with larger than 8,192 time steps in this numerical test setting.

The TSC method that generates a linear equation at the coarse level reached convergence faster than the step-by-step method for all problems including small sized problems. In comparison with the MGRIT method, TSC methods are faster than the MGRIT method for almost all problems. The TSC loop3 was the fastest.

Table 2 has the execution results of TSC loop1, TSC loop2,

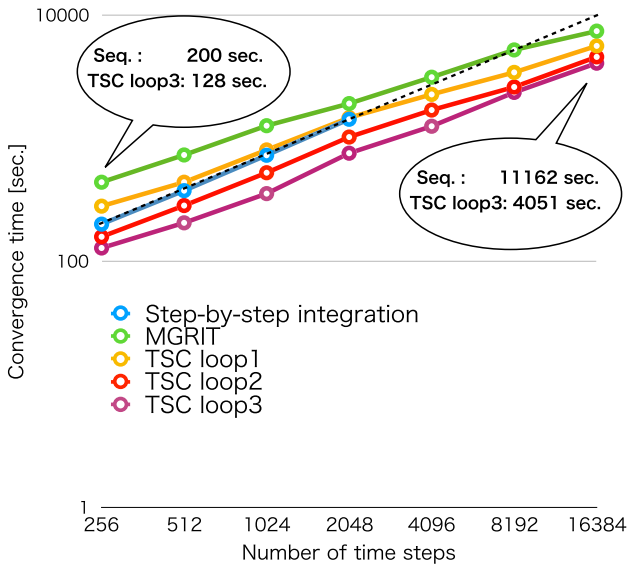

Fig. 8 Execution time with different number of time steps.

Table 2 Execution time and iteration number of TSC. First, second and third rows corresponds to TSC loop1, TSC loop2 and TSC loop3 respectively.

# of time steps	256	512	1024	2048	4096	8192	16384
Total time [s]	280	438	801	1463	2261	3413	5601
C_lev. time	189	235	309	452	724	1062	2173
# of iter.	9	11	14	16	15	12	10
Total time [s]	158	283	523	1019	1688	2596	4559
C_lev. time	110	158	214	344	570	867	1836
# of iter.	5	7	9	11	11	9	8
Total time [s]	128	205	354	753	1247	2346	4051
C_lev. time	91	119	153	266	457	797	1686
# of iter.	4	5	6	8	8	8	7

and TSC loop3. While TSC loop1's iteration number increases depending on the number of time steps, repeating coarse level correction methods such as TSC loop2 and TSC loop3 reduced the number of iterations for convergence. The error including nonlinearity of the problem is considered to be corrected more by repeating the coarse level correction. The table has also "C_lev. time" that shows the execution time for Jacobian matrix calculation and coarse level correction. Although TSC loop2 and TSC loop3 repeats the coarse level correction twice or three times in one TSC iteration, "C_lev. time" per one TSC iteration is almost the same as that of TSC loop1. Because the coarse level corrections of TSC loop2 and loop3 are executed in pipeline manner as described in Section 5. Since all time steps were distributed to 64 processes, "C_lev. time" of TSC loop2 is considered to be longer than that of TSC loop1 by the rate of 1/64. It has turned out that repeating the coarse level correction in pipeline manner enhances the stability of the TSC method with low additional cost.

In **Table 3** with the results of MGRIT, the first column shows the coarse to fine ratios, and first row shows the number of time steps of the problem. It records the execution time when coarse to fine ratio C_r is changed to several values. Bold values means the fastest time among several cases with different coarse to fine ratios. From these tables, TSC loop3 was faster than MGRIT by 80% even with the largest problem, and it was three times faster at most.

Table 3 Execution time [sec] of MGRIT with different Coarse to Fine ratio.

Time steps	256	512	1024	2048	4096	8192	16384
C_r =4	575	1431					
8	**438**	859	1845	3658			
16	519	**729**	**1260**	2532	4993		
32		879	1324	**1905**	3788	6938	
64				2581	**3119**	5830	
128					4790	**5194**	9750
256						7126	**7384**
512							11912

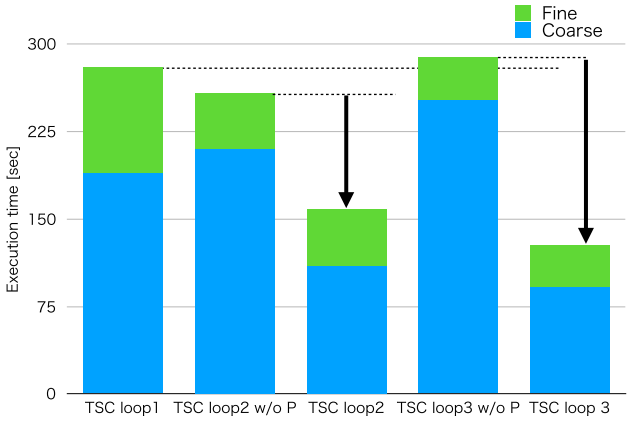

Fig. 9 Pipeline execution effect on total time with 256 time step problem: "w/o P" means without pipelining. The execution time of "w/o P" is estimated using TSC loop1's one iteration coarse level time.

6.2 Discussions

This subsection considers two points of the TSC method. First is the analysis on performance improvement by pipeline execution. Second is the TSC method for larger space sized problems.

The execution time of TSC loop2 and 3 without pipeline execution is considered first. The iteration number for convergence does not change, because pipeline execution does not change the arithmetic operactions. The non-pipelined version calculates Jacobian matrices of all time steps, the coarse level matrix, and solution update synchronously with all processes. The TSC loop1 calculates only one time coarse level correction in a cycle, and it can not be executed in pipeline fashion. Thus the "C_lev. time" of TSC loop1 shows the syncronous execution time of the coarse level correction. One iteration of TSC loop2 without pipeline execution contains twice of TSC loop1's one iteration "C_lev. time", which is calculated as "C_lev. time" divided by the iteration number.

When the number of time steps is 256, "C_lev. time" of TSC loop2 and 3 without pipeline exection can be estimated as 210 and 252 seconds from TSC loop1 "C_lev. time". Since pre and post smoothers are calculated syncronously with all processes, the increase of "C_lev. time" is reflected directly to total execution time of non-pipeline version. Thus, the total execution time of TSC loop2 and 3 without pipelining is estimated as 258 and 289 seconds. **Figure 9** shows the situation. Although just repeating the coarse level correction twice without pipeling will decrease the execution time by about 10%, pipeline execution enhances the performance more like the black arrow of TSC loop2 in Fig. 9. When coarse level correction is repeated 3 times, then the total execution time is estimated to be degraded from TSC loop1, because of the overhead of coarse level calculation three times.

However, pipeline execution decreases the overhead, and TSC loop3 becomes the fastest method among TSC loop1, 2, and 3.

In numerical tests, we checked different sized problems in time direction. Here we consider the TSC method for a large space sized problem. When the size of the space direction is enlarged, each time step equation becomes large, and it should be solved by an iterative method. The coarse level matrix shape is block lower triangular matrix, and the block size corresponds to the space size. Thus, each block row of coarse level matrix should be solved by an iterative method, and the coarse level matrix can be solved by forward substitution. Therefore, the pipeline execution can be applied to coarse level correction even if an iterative solver is used for each time step problem. In the case that repeated coarse level correction enhances the convergence, pipeline execution method is considered to be one of a good implementation candidate.

7. Conclusion

This paper extends the parallel TP-EEC method that was proposed for periodic non-linear electro-magnetic field problem to an ordinary non-linear time integral problem. We call this method the Time segment correction (TSC) method. We proposed its efficient implementation method and evaluated the TSC method in comparison with MGRIT method for simple nonlinear time integration problem. It is important to note that MGRIT has many parameters such as number of levels, smoother types, and coarse to fine ratios. It is difficult to optimize the parameters according to each problem. Therefore the performance of our MGRIT implementation is used as a performance baseline.

Our method uses pipelined coarse grid correction for the TSC method. In our numerical test, TSC methods were up to three times faster than MGRIT method. The TSC method repeating coarse level correction in pipeline manner was faster than the step-by-step method for all problems including a small sized problem of 256 time steps.

Although a non-linear problem is assumed for this paper, the TSC method can be applied to the linear problem. In that case, the governing linear equation can be directly used to generate the coarse level problem with the same prolongation matrix B described in Section 4.1. For linear problems, the coarse level equation does not depend on the approximate solution vector. Thus, repeating the coarse level correction is not effective. However, TSC loop1 is applicable to the problem and the effectiveness for linear problems must be evaluated in the future.

As for applicability to existing application codes, the TSC method uses only Jacobian matrices for calculation of the coarse level problem. Therefore, the TSC method becomes "non-intrusive" method for applications that can output Jacobian matrices. We will increase the evaluation cases in various simulation fields.

Acknowledgments We would thank Prof. Yasuhito Takahashi (Doshisha University) and Prof. Kengo Nakajima (The University of Tokyo) for their useful comments for the research. This paper is partially supported by "Joint Usage/Research Center for Interdisciplinary Large-scale Information Infrastructures" and "High Performance Computing Infrastructure" in Japan.

References

[1] Gander, M.J.: 50 years of time parallel time integration, *Multiple Shooting and Time Domain Decomposition Methods*, pp.69–113, Springer (2015).
[2] Falgout, R.D., Katz, A., Kolev, T.V., Schroder, J.B., Wissink, A.M. and Yang, U.M.: Parallel time integration with multigrid reduction for a compressible fluid dynamics application, Lawrence Livermore National Laboratory Technical Report (2015).
[3] Falgout, R.D., Friedhoff, S., Kolev, T.V., MacLachlan, S.P. and Schroder, J.B.: Parallel Time Integration with Multigrid., *SIAM J. Scientific Computing*, Vol.36, No.6, pp.C635–C661 (2014).
[4] Gahvari, H., Dobrev, V.A., Falgout, R.D., Kolev, T.V., Schroder, J.B., Schulz, M. and Yang, U.M.: A performance model for allocating the parallelism in a multigrid-in-time solver (2016).
[5] Hutchinson, B.R. and Raithby, G.D.: A Multigrid Method based on the Additive Correction Strategy, *Numerical Heat Transfer*, Vol.9, No.5, pp.511–537 (1986).
[6] Lions, J.-L., Maday, Y. and Turinici, G.: Résolution d'EDP par un schéma en temps ≪pararéel≫, *Comptes Rendus de l'Académie des Sciences - Series I - Mathematics*, Vol.332, No.7, pp.661–668 (online), DOI: https://doi.org/10.1016/S0764-4442(00)01793-6 (2001).
[7] Meuer, H.W., Strohmaier, E., Dongarra, J. and Simon, H.D.: *The TOP500: History, Trends, and Future Directions in High Performance Computing*, Chapman & Hall/CRC, 1st edition (2014).
[8] Reedbush-u: Information Technology Center, The University of Tokyo, available from ⟨http://www.cc.u-tokyo.ac.jp⟩.
[9] Takahashi, Y., Tokumasu, T., Kameari, A., Kaimori, H., Fujita, M., Iwashita, T. and Wakao, S.: Convergence Acceleration of Time-Periodic Electromagnetic Field Analysis by the Singularity Decomposition-Explicit Error Correction Method, *IEEE Trans. Magnetics*, Vol.46, No.8, pp.2947–2950 (online), DOI: 10.1109/TMAG.2010.2043721 (2010).
[10] Takahashi, Y., Iwashita, T., Nakashima, H., Tokumasu, T., Fujita, M., Wakao, S., Fujiwara, K. and Ishihara, Y.: Parallel Time-Periodic Finite-Element Method for Steady-State Analysis of Rotating Machines, *IEEE Trans. Magnetics*, Vol.48, No.2, pp.1019–1022 (2012).
[11] Takahashi, Y., Tokumasu, T., Fujiwara, K., Iwashita, T. and Nakashima, H.: Parallel TP-EEC Method Based on Phase Conversion for Time-Periodic Nonlinear Magnetic Field Problems, *IEEE Trans. Magnetics*, Vol.51, No.3, pp.1–5 (2015).
[12] XBraid: Parallel multigrid in time, available from ⟨https://computation.llnl.gov/projects/parallel-time-integration-multigrid⟩.

Akihiro Fujii was born in 1975. He received his B.S., M.S., and Ph.D. of Information Science and Technology from Tokyo University in 1999, 2001, and 2004, respectively. In 2004-2006, he worked as a lecturer professor at CPD center in Kogakuin University. In 2006, he moved to the faculty of informatics, and currently works as an associate professor in Kogakuin University. His reserch interest includes high performance computing, numerical linear algebra and hierarchical algorithms. He is a member of IPSJ, IEEE-CS and IEICE.

Shigeo Kaneko was born in 1992. He received his B.Sc. and M.Sc. from Kogakuin University in 2016 and 2018, respectively. He studied parallel time integration methods for a master thesis. From 2018, he works in Ark Infomation Systems.

Teruo Tanaka was born in 1958. He received his B.E., M.E., and Ph.D. from The University of Electro-communications in 1981, 1983, 2007, respectively. In 1983–1997, he worked at Central Research Laboratory, Hitachi Ltd. He currently works as a professor in the Department of Computer Science, Faculty of Informatics, Kogakuin University from 2011. His research interests include High Performance Computing, Software Auto-tuning, High precision Calculation. He is a member of IPSJ, IEEE CS and JSIAM.

Takeshi Iwashita was born in 1971. He received his B.E., M.E., and Ph.D. from Kyoto University in 1992, 1995, and 1998, respectively. In 1998–1999, he worked as a post-doctoral fellow of the JSPS project in the Graduate School of Engineering, Kyoto University. He moved to the Data Processing Center of the same university in 2000. In 2003–2014, he worked as an associate professor in the Academic Center for Computing and Media Studies, Kyoto University. He currently works as a professor in the Information Initiative Center, Hokkaido University. His research interests include high performance computing, linear iterative solver, and electromagnetic field analysis. He is a member of IEEE, SIAM, IPSJ, IEEJ, JSIAM, JSCES, and JSAEM.