NORGES TEKNISK-NATURVITENSKAPELIGE
UNIVERSITET

Parallelization in time for thermo-viscoplastic problems in extrusion of aluminium

by

E. Celledoni, T. Kvamsdal

PREPRINT
NUMERICS NO. 7/2007

NORWEGIAN UNIVERSITY OF
SCIENCE AND TECHNOLOGY
TRONDHEIM, NORWAY

Parallelization in time for thermo-viscoplastic problems in extrusion of aluminium

E. Celledoni, T. Kvamsdal

October 2, 2007

The ParaReal algorithm, [12], is a parallel approach for solving numerically systems of Ordinary Differential Equations by exploiting parallelism across the steps of the numerical integrator. The method performs well for dissipative problems and problems of fluid structure interaction [7]. We consider here a convergence analysis for the method and we report the performance achieved from the parallelization of a Stokes code via the ParaReal algorithm.

1 Introduction

The ParaReal algorithm, [12], [1], parallelizes in the time direction the computational effort required to solve evolution equations. This method is an iterative procedure based on the use of coarse sequential solvers and fine solvers to be computed in parallel.

A survey on earlier contributions in the field of parallelization of ODE solvers in the time direction can be found in [4] and references therein. The main idea behind these methods are very similar to the ParaReal algorithm, which however appears to be easier to implement, especially in the PDE context.

Recent articles and studies, [13], [7], give evidence that the ParaReal technique performs well in the case of dissipative problems, especially if particular care is taken in the choice of the numerical integrators used in the implementation, [13]. In [7], in particular, the method is used successfully also in problems of fluid structure interaction.

Herein we investigate the performance of the method in the parallelization of a finite element code for the numerical solution of time dependent Stokes and non-Newtonian Navier-Stokes problems for creeping flow. Our motivation for focusing on this type of equations comes from our interest into the challenging industrial problem of extrusion of aluminium modeled as a coupled thermo-viscoplastic problem, [10]. The movement of the stem, i.e. the contraction of the domain occupied by the aluminium in the extrusion tool, is taken into account to capture the transient character of the extrusion process. Even if in this paper we report experiments on somewhat simple test problems, the actual simulations require a very large number of degrees of freedom and the number of time steps required is also quite large. In these models the presence of high viscosity implies that the dynamics is dominated by the diffusion operator (Laplace operator), which is typically self-adjoint and has real eigenvalues. We will restrict our analysis to the case of symmetric negative definite spatial discretizations of such operator. Under these circumstances ParaReal can be applied succesfully provided the numerical integrator employed in the implementation of the algorithm is implicit and satisfies certain conditions discussed in section 2.2.

In section 2 we insert the ParaReal method in the context of other strategies of parallelism across time, we investigate the convergence of the iterative process, and we discuss classes of problems and time integration methods amenable to make the ParaReal strategy successful in terms of speed-up. In the final section of this paper we describe briefly the parallel implementation of the ParaReal method and we report some numerical experiments on the performance of our parallel implementation of ParaReal on three different PDE test problems.

2 Parallelism across time and the ParaReal algorithm

In this section we describe some known strategies of parallelism across time and relate them to the ParaReal algorithm. Suppose $[0, T_{\text{fin}}]$ is the interval of integartion of the ODE problem

$$\begin{array}{rcl} y' & = & f(y) \\ y(0) & = & b_0, \end{array} \tag{1}$$

and consider a grid of the type $t_0 = 0 < t_1 < \cdots < t_N = T_{\text{fin}}$, (the nodes will be here assumed equidistant). Our goal is to compute at the given times the sequence of numerical approximation of the ODE solution produced by a given numerical solver F_n, i.e.

$$y_0 = y_0, y_1 = F_1(y_0), y_2 = F_2(y_1), \ldots, y_N = F_N(y_{N-1}), \tag{2}$$

where the approximations $y_n \approx y(t_n)$, for $n = 0, \ldots, N$, have an adequate accuracy established a priori by the choice of F_n.

By introducing the notation $Y = [y_0, \ldots, y_N]^T$, problem (2) can be rewritten in the compact form

$$Y = \phi(Y), \qquad \phi(Y) = [y_0, F_1(y_0), \ldots, F_N(y_{N-1})]^T. \tag{3}$$

The fixed point iteration $Y^{k+1} = \phi(Y^k)$ can be easily parallelized computing each of the $N+1$ block components of Y^{k+1} on a different processor, the convergence is however in general very poor and the method does not lead to a computational gain. Acceleration of the convergence of this iterative process can be achieved by rewriting the fixed point equation (3) in a suitable manner.

In earlier papers, [3], [2], [6], see also [4], the following approach was considered. Suppose $\Delta(Y)$ is a matrix valued function such that

$$\Delta(Y) \cdot Y \approx \phi(Y),$$

then we can rewrite (3) as

$$Y - \Delta(Y) \cdot Y = \phi(Y) - \Delta(Y) \cdot Y, \tag{4}$$

and assuming $I - \Delta(Y)$ to be invertible we obtain a new fixed point equation

$$Y = (I - \Delta(Y))^{-1}(\phi(Y) - \Delta(Y) \cdot Y). \tag{5}$$

A natural choice is to take $\Delta(Y)$ to be the Jacobian of $\phi(Y)$, and then the fixed point iteration stemming from (4) corresponds to the Newton iteration for solving the nonlinear equation $Y - \phi(Y) = 0$. In this case $\Delta(Y)$ is a strictly lower triangular block matrix, $I -$

$\Delta(Y)$ is invertible, and $(I-\Delta(Y))^{-1}V$ can be easily computed by a backward substitution. This means that in the new iteration,

$$Y^{k+1} = (I - \Delta(Y^k))^{-1}(\phi(Y^k) - \Delta(Y^k) \cdot Y^k), \tag{6}$$

while $V^k := \phi(Y^k) - \Delta(Y^k) \cdot Y^k$ can be computed in parallel, the solution of the triangular linear system,

$$(I - \Delta(Y^k))^{-1}V^k,$$

is by its nature a sequential task. Since the computation of the Jacobian of $\phi(Y)$ can be difficult to achieve in practice, it might be useful to consider suitable surrogates, leading for example to a Steffensen iteration [2]. In this approach $\Delta(Y^k)$ has the same lower triangular block structure of the Jacobian of ϕ, and then

$$\Delta(Y^k)V = [v_1, \Delta_1(Y^k)v_2, \ldots, \Delta_N(Y^k)v_{N+1}]^T.$$

Each of the matrix blocks Δ_n is of dimension $m \times m$ and it is defined as follows. We consider

$$\tilde{Y}^{k+1} = \phi(Y^k),$$

and compute

$$Y^{k,j} := Y^k - \sum_{n=0}^{N} (e_n \otimes e_j)(e_n \otimes e_j)^T (Y^k - \tilde{Y}^{k+1}),$$

where e_n is the n-th canonical vector in $\mathbb{R}^{N+1}$, e_j is the j-th canonical vector in $\mathbb{R}^m$ and $\otimes$ is the Kroneker product (see for example [9] p. 243). Furthermore we consider

$$\tilde{Y}^{k+1,j} = \phi(Y^{k,j}), \;\; j = 1, \ldots m$$

and

$$\Delta_n(Y^k)_{i,j} = \frac{(e_{n+1} \otimes e_i)^T(\tilde{Y}^{k+1,j} - \tilde{Y}^{k+1})}{(e_n \otimes e_j)^T(\tilde{Y}^{k+1} - Y^k)},$$

for $i, j = 1, \ldots, m$. The parallelism of the overall approach is lying in the use of the mapping ϕ, which can be applied concurrently on each of the block components of the argument. However, in the case m is large this method can be quite demanding as it requires the computation and storage of each of the $m \times m$ matrix blocks $\Delta_n(Y^k)$ per each iteration. If F_n represents the implementation of an implicit solver for the numerical discretization of a PDE problem, the blocks Δ_n have the same size and sparsity of a typical linear system whose solution is required in the implementation of F_n.

The parallel algorithm is intertwined with the linear algebra of the solver F_n as the solution of linear systems with Δ_n become building blocks in the parallel scheme.

The advantage the ParaReal method is that no approximations of the Jacobian of ϕ are needed and that the parallelism in time in principle does not require to interfere with the linear algebra of the solver F_n. The idea behind the ParaReal algorithm can be described as follows. Suppose now we are given an approximation ψ of ϕ, and we write (4) in the following way

$$Y - \psi(Y) = \phi(Y) - \psi(Y), \tag{7}$$

this immediately leads to the fixed point iteration

$$Y^{k+1} - \psi(Y^{k+1}) = W^k, \quad\; W^k = \phi(Y^k) - \psi(Y^k), \tag{8}$$

and, in this case, a nonlinear equation involving ψ instead of ϕ must be solved at each iteration step. The approximation ψ to ϕ can be obtained introducing the operators $G_n \approx F_n$ and then taking

$$\psi(Y) = [y_0, G_1(y_0), \dots, G_N(y_{N-1})]^T.$$

The operators G_n should be both a good and inexpensive approximation of F_n. Typically F_n and G_n are based on the same numerical algorithm, F_n propagates the numerical solution of the evolution equation with a certain accuracy and a small time-step, and G_n with a lower accuracy and a bigger time-step. We will in fact here assume that F_n and G_n are obtained by applying a Runge–Kutta method. The solution of the nonlinear equation,

$$Y^{k+1} - \psi(Y^{k+1}) = W^k,$$

is done sequentially, but for suitable choices of G_n, is less expensive to compute compared to the original nonlinear equation. The computation of W^k is the predominant task of the computation, and it can be executed in parallel.

Iteration $k+1$ of the ParaReal procedure can then be defined by means of the iteration k by the following recursion formula,

$$\begin{aligned} y_{n+1}^0 &= G_n(y_n^0), \quad y_0^0 = y_0, && n = 0, \dots, N-1, \\ y_{n+1}^{k+1} &= F_n(y_n^k) + G_n(y_n^{k+1}) - G_n(y_n^k), && n = 0, \dots, N-1. \end{aligned} \tag{9}$$

The first iterate $[{y_0^0}^T, {y_1^0}^T, \dots, {y_N^0}^T]^T$, is obtained by computing the numerical approximation using the coarse operator.

2.1 Computational complexity and speed-up

Since y_{n+1}^{k+1} for $n = 0, \dots, N-1$ depends only on $F(y_n^k)$ for $n = 0, \dots, N-1$, assuming we have N processors at our disposal, we can compute one of the $F_n(y_n^k)$ on each processor. Once $F_n(y_n^k)$ for $n = 0, \dots, N-1$ are available as a result of a parallel computation, y_{n+1}^{k+1} is computed form (9) by sequentially applying G_n. If now we assume $\mathbf{u}$ is the cost of applying G_n, i.e. the cost of performing one step of a chosen one-step integrator (with step-size $\Delta T = T_{\text{fin}}/N$ and $[0, T_{\text{fin}}]$ is the interval of integration), then the overall cost of the ParaReal method per iteration is given by the cost of applying F_n once simultaneously on each processor, $l \cdot \mathbf{u}$, and N times the cost of applying G_n sequentially, in total

$$l \cdot \mathbf{u} + N \cdot \mathbf{u}.$$

Here l is the number of micro-steps of size $\Delta T/l$ required to implement F_n. In our experiments we will always consider $l = N$.

The cost of one iteration of ParaReal should be compared to the cost of applying F_n, N times sequentially, i.e.

$$N \cdot l \cdot \mathbf{u}.$$

We obtain a speed-up of

$$\frac{N \cdot l \cdot \mathbf{u}}{m(l \cdot \mathbf{u} + N \cdot \mathbf{u})},$$

where m is the total number of iterations performed. If $m = N$ there is no speed-up, and therefore it is important to apply the method to problems where convergence is obtained for $m << N$. If we want to maximize the speed-up, we must aim at applying ParaReal in cases when we can terminate the iteration after just few steps, while being certain to have reduced the norm of the initial error of a sufficient amount.

2.2 Convergence of the ParaReal iteration in the linear case

In this section we perform an analysis of the ParaReal algorithm. The thermo-viscoplastic model problems considered in this paper, all have a dominating linear part, and for this reason we will restrict our analysis to the linear case. In the case F is linear, i.e. $F(y) = F \cdot y$ where F is a constant if y is a scalar and F is a matrix if y is a vector, we can rewrite (2) in a matrix format as follows

$$\begin{bmatrix} I & O & \cdots & O \\ -F & I & \ddots & \vdots \\ \vdots & \ddots & \ddots & O \\ O & \cdots & -F & I \end{bmatrix} Y = b, \tag{10}$$

where $b = [y_0^T, 0^T, \ldots, 0^T]^T$. We denote the operator of the system of equations to be solved $I - S$, I is the identity and S is minus the lower triangular part. This operator $I - S$ is a bidiagonal block Toeplitz matrix.

Consistently with the adopted notation, in the linear case we can rewrite the iterative process as follows

$$\underbrace{\begin{bmatrix} I & O & \cdots & O \\ -G & I & \ddots & \vdots \\ \vdots & \ddots & \ddots & O \\ O & \cdots & -G & I \end{bmatrix}}_{(I-P)} \Lambda^{k+1} = \underbrace{\begin{bmatrix} O & O & \cdots & O \\ F & O & \ddots & \vdots \\ \vdots & \ddots & \ddots & O \\ O & \cdots & F & O \end{bmatrix}}_{S} \Lambda^k - \underbrace{\begin{bmatrix} O & O & \cdots & O \\ G & O & \ddots & \vdots \\ \vdots & \ddots & \ddots & O \\ O & \cdots & G & O \end{bmatrix}}_{P} \Lambda^k + b$$

where $\Lambda^k = [\lambda_0^k, \ldots, \lambda_N^k]^T$. Here $(I-P)$, and $(S-P)$ are bidiagonal block Toeplitz matrices.

In short we obtain a format of the type

$$\Lambda^{k+1} = (I-P)^{-1}(S-P)\Lambda^k + (I-P)^{-1}b. \tag{11}$$

This iteration stems from the linear block fix point equation

$$Y = (I-P)^{-1}(S-P)Y + (I-P)^{-1}b, \tag{12}$$

equivalent to the linear system

$$(I-S)Y = b. \tag{13}$$

Subtracting (11) from (12) we obtain the recursion formula for the error,

$$Y - \Lambda^k = (I-P)^{-1}(S-P)(Y - \Lambda^{k-1}). \tag{14}$$

We will use in the sequel the following expression for the inverse of $(I-P)^{-1}$ in terms of G,

$$(I-P)^{-1} = \begin{bmatrix} I & O & \cdots & \cdots & O \\ G & I & O & \cdots & \vdots \\ G^2 & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ G^N & \cdots & G^2 & G & I \end{bmatrix}.$$

The matrix $(I-S)^{-1}$ has a similar structure with G substituted by F. It is easy to veryfy by straightforward calculation that

$$(I-P)^{-1}(S-P) = \begin{bmatrix} O & O & \cdots & \cdots & O \\ F-G & O & O & \cdots & \vdots \\ G(F-G) & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & & \ddots & \vdots \\ G^{N-1}(F-G) & \cdots & G(F-G) & F-G & O \end{bmatrix}. \tag{15}$$

We are interested in the case when the differential equation to be solved is defined by a symmetric negative definite spatial discretization of the Laplace operator. The problem is therefore diagonalizable via an orthogonal similarity transformation, and in the analysis we can look separately at a scalar problem for each of the eigenvalues. We therefore restrict now our analysis to the linear scalar case constituting a good indication of what would happen in more general cases. We first assume F is the exact solution of the linear scalar differential equation $y' = \lambda y,\ y(0) = y_0$, i.e. $F = e^{\lambda \Delta T}$, and we assume $G = R(\lambda \Delta T)$, where R is a polynomial or a Pade' approximant of the exponential function. Then we have $F - G = e^{\lambda\Delta T} - R(\lambda\Delta T)$. We are interested in sufficient conditions guaranteeing that the error of the ParaReal iteration is contracted at each step.

Consider the function defined by

$$R_N(z) := \max_{k=1,\dots,N-1} \left| \sum_{l=1}^{k+1} R(z)^l (e^z/R(z) - 1) \right|.$$

By using (14) and (15) one shows that

$$\|Y - \Lambda^k\|_\infty \leq R_N(h\lambda)\|Y - \Lambda^{k-1}\|_\infty.$$

Consider also the subset of the complex plane here defined,

$$S_N := \{z \in \mathbf{C} \quad | \quad R_N(z) < 1\}.$$

If $\lambda\Delta T \in S_N$ then the error of the ParaReal iteration with N processors is contracted at each step in the max norm, i.e. we have

$$\|Y - \Lambda^k\|_\infty < \|Y - \Lambda^{k-1}\|_\infty, \qquad k = 1, 2, \dots, N.$$

In figures 1, 2 (left) and 3 (left) we report the level curves of $R_N(z)$ in a portion of the complex plane. We are in particular interested in the shape of the set S_N for different choices of $R(z)$. In the case $R(z) = 1 + z$, corresponding to using the explicit Euler method in the implementation of the coarse operator, S_N is quite small and it is included in an ellipse of the negative complex plane tangent in the origin to the imaginary axsis. This implies severe restrictions on the step-sze ΔT if we require to contract the error at each iteration.

For implicit methods S_N is including a bigger area of the negative complex plane even if $\mathbf{C}^-$ is not entirely included in S_N. In particular we observe that the implicit Euler method, (figure 3), has the desired property of contracting the error at each iteration, for a big range of values of z in the negative complex plane, apart from a thin strip along imaginary axis.

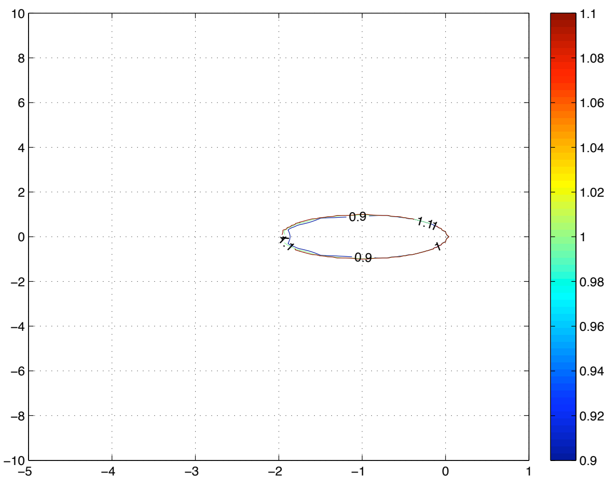

Figure 1: Level curves of the function $R_N(z)$ in the negative complex plane. The level curve corresponding to the value 1 delimits the set S_N. In this experiment $R(z) = (1 + z)$ the stability function of the explicit Euler integration method. In this experiment $N = 80$. Level curves at the values 0.9, 1 and 1.1 of $R_N(z)$.

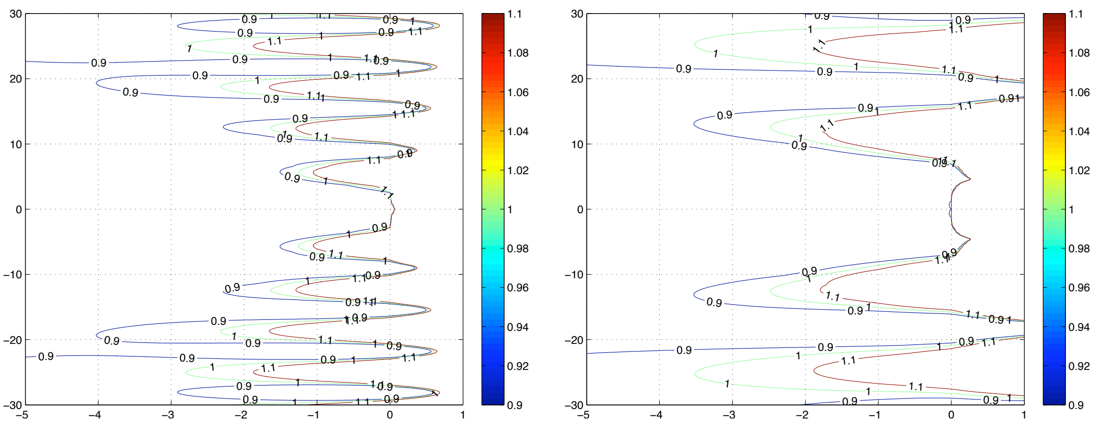

Figure 2: Level curves of the function $R_N(z)$ (left) and $\tilde{R}_N(z)$ (right) in a region of the complex plane. The level curve corresponding to the value 1 delimits the set S_N and $\tilde{S}_N$ respectively. In this experiment $R(z) = (1 + z/2)/(1 - z/2)$ the stability function of the midpoint integration method. In this experiment $N = 80$. Level curves at the values 0.9, 1 and 1.1 of $R_N(z)$ and $\tilde{R}_N(z)$ respectively.

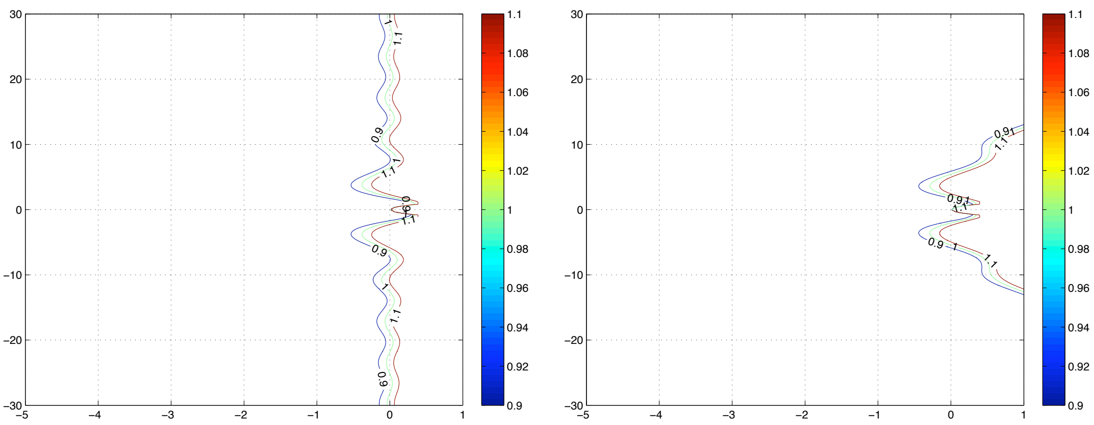

Figure 3: Level curves of the function $R_N(z)$ (left) and $\tilde{R}_N(z)$ (right) in a region of the complex plane. The level curve corresponding to the value 1 delimits the set S_N and $\tilde{S}_N$ respectively. In this experiment $R(z) = 1/(1-z)$ the stability function of the implicit Euler integration method. In this experiment $N = 80$. Level curves at the values 0.9, 1 and 1.1 of $R_N(z)$ and $\tilde{R}_N(z)$ respectively.

The implicit midpoint (figure 2), meets the requirement at least in a sector of the negative complex plane.

As in general the fine operator does not compute the exact solution, but just a more accurate approximation compared to the coarse operator, we now consider the function

$$\tilde{R}_N(z) := \max_{k=1,\dots,N-1} \left| \sum_{l=1}^{k+1} R(z)^l (R(z/N)^N / R(z) - 1) \right|,$$

where e^z has been substituted with $R(z/N)^N$. This corresponds to the choice we have done in the implementation of ParaReal in our codes, i.e. the fine operator is obtained by subdividing the macrostep ΔT in exactly N microsteps of size $\delta t = \Delta T/N$. Correspondingly we can define the set

$$\tilde{S}_N := \left\{ z \in \mathbf{C} \quad | \quad \tilde{R}_N(z) < 1 \right\},$$

such that if $\lambda \Delta T \in \tilde{S}_N$ then the error of the ParaReal iteration with N processors is contracted at each step in the max norm. The sets $\tilde{S}_N$ for the case of the implicit midpoint and the implicit Euler method are reported in figures 2 (right) and 3 (right). The case of the explicit Euler method is not reported as there is virtually no difference compared to figure 1.

Now the advantage of using the implicit Euler method is even more evident, in fact $\tilde{S}_N$ is significantly bigger in this case, (figure 3), compared to S_N. For the midpoint, in figure 2, $\tilde{S}_N$ is not improved compared to S_N. In conclusion the implicit Euler method seams to be a better choice for the implementation of ParaReal compared to the implicit midpoint.

2.3 Other estimates

It is also easy to realize that

$$\Lambda^k = \sum_{l=0}^{k} \left((I-P)^{-1}(S-P) \right)^l \Lambda^0, \quad \Lambda^0 = (I-P)^{-1} b. \tag{16}$$

In order to understand the behaviour of the iteration we want to find the powers

$$\big((I-P)^{-1}(S-P)\big)^j, \quad j=1,2,\ldots.$$

Since $(I-P)^{-1}(S-P)$ is nihilpotent the Neuman series for $I-(I-P)^{-1}(S-P)$ has a finite number of terms, and we have

$$\big(I-(I-P)^{-1}(S-P)\big)^{-1}=\sum_{j=0}^{N}\big((I-P)^{-1}(S-P)\big)^j, \tag{17}$$

which proves (in the linear case) that the algorithm gets to the exact solution in $m\le N$ steps. We are interested in characterizing the non trivial powers of $(I-P)^{-1}(S-P)$.

Proposition 2.1. *The j-th power of $(I-P)^{-1}(S-P)$ is a lower triangular block Toeplitz matrix. The first column of $\big((I-P)^{-1}(S-P)\big)^j$ has $N+1$ block components equal to zero for $k=1,\ldots,j$, and equal to*

$$\alpha_k^j G^{k-j-1}(F-G)^j, \ \text{with}\ \alpha_k^j := \left(\begin{array}{c} k-2 \\ j-1 \end{array}\right), \tag{18}$$

for $k=j+1,\ldots,N+1$.

Proof We proceed by induction. For $j=1$ we verify that formula (18) is returning the expression founded explicitely in (15). According to (18), for $j=1$, we have that $\alpha_k^1=1$, for $k=2,\ldots,N+1$, the first block component is zero, and the remaining ones are

$$G^{k-2}(F-G), \quad k=2,\ldots,N+1,$$

which verifies (18) for $j=1$.

Assume now the proposition holds true for the $j-1$-th power. It is well known that the lower triangular block Toeplitz matrices form a commutative algebra. Therefore multiplying $(I-P)^{-1}(S-P)$ and $((I-P)^{-1}(S-P))^{j-1}$ we obtain a Toeplitz lower triangular block matrix. It is then enough to multiply $(I-P)^{-1}(S-P)$ from formula (15), with the first column of $((I-P)^{-1}(S-P))^{j-1}$ in order to find the first column of $((I-P)^{-1}(S-P))^j$. Due to the Toeplitz structure, from the first column we are able to reconstruct the whole matrix.

By the induction hypothesis the first column of $((I-P)^{-1}(S-P))^{j-1}$ has $N+1$ block components equal to zero for $s=1,\ldots,j-1$, and equal to

$$\alpha_s^{j-1}G^{s-j}(F-G)^{j-1}, \ \text{with}\ \alpha_s^{j-1} := \left(\begin{array}{c} s-2 \\ j-2 \end{array}\right),$$

for $s=j,\ldots,N+1$. Multiplying the k-th row of $I-(I-P)^{-1}(S-P)$ with the first column of $((I-P)^{-1}(S-P))^{j-1}$ we obtain

$$\sum_{s=j+1}^{k-1}\alpha_s^{j-1}G^{k-1-s}(F-G)G^{s-j}(F-G)^{j-1},$$

for $k=j+1,\ldots,N+1$. Since G and F commute, the given expression is equal to

$$\left(\sum_{s=j+1}^{k-1}\alpha_s^{j-1}\right)G^{k-1-j}(F-G)^j. \tag{19}$$

Substituting in the sum the expressions for α_s we obtain

$$\sum_{s=j+1}^{k-1} \alpha_s^{j-1} = \begin{pmatrix} j-1 \\ j-2 \end{pmatrix} + \cdots + \begin{pmatrix} k-1-2 \\ j-2 \end{pmatrix}.$$

It can be easily proven by induction that

$$\sum_{s=j+1}^{k-1} \alpha_s = \begin{pmatrix} k-2 \\ j-1 \end{pmatrix} = \alpha_{k-1}^{j}, \qquad k = j+1, \ldots, N+1.$$

This together with (19) proves the proposition. *q.e.d.*

We will use now the result of the proposition for characterizing the error at the k-th iteration of the ParaReal method.

Using the previously deduced expression for the inverse of $(I - (I-P)^{-1}(S-P))$ and since Y satifies (12), we can write the following espression for the solution of the problem

$$Y = \sum_{j=0}^{N} ((I-P)^{-1}(S-P))^j (I-P)^{-1} b.$$

We can then obtain the following expressions for the error in the iterative process

$$Y - \Lambda^k = \sum_{j=k+1}^{N} ((I-P)^{-1}(S-P))^j (I-P)^{-1} b.$$

By straightforward calculation we have that

$$(I-P)^{-1} b = \begin{bmatrix} b_0 \\ G b_0 \\ \vdots \\ G^N b_0 \end{bmatrix}.$$

In order to find an explicit expression for the error $Y - \Lambda^k$, we compute

$$((I-P)^{-1}(S-P))^j (I-P)^{-1} b.$$

We use the characterization of $((I-P)^{-1}(S-P))^j$ given in Proposition 2.1, and with a similar procedure as the proof of Proposition 2.1, we find that $((I-P)^{-1}(S-P))^j (I-P)^{-1} b$ has components equal to zero for $s = 1, \ldots, j$ and equal to

$$\alpha_{s+1}^{j+1} (F-G)^j G^{s-j-1} b_0,$$

for $s = j+1, \ldots, N+1$. As an immediate consequence we obtain the following characterization of the error after k iterations of the ParaReal iteration in the linear case.

Proposition 2.2. *The error in the ParaReal iteration is given by*

$$Y - \Lambda^k = \sum_{j=k+1}^{N} ((I-P)^{-1}(S-P))^j (I-P)^{-1} b,$$

and it is a vector divided in $N+1$ component blocks. The first $k+1$ component blocks are zero, while the remaining $N-k-1$ ones are of the type

$$\sum_{l=k+2}^{s} \alpha_{s+1}^{l} (F-G)^{l-1} G^{s-l} b_0, \qquad s = k+2, \ldots, N+1. \tag{20}$$

N	4	8	16	32
speedup perc.	45.79	37.59	28.90	29.00
speedup	1.8319	3.00	4.62	9.56
error	.3118D-01	.9570D-02	.3201D-02	.1238D-02

Table 1: Heat equation discretized in space with central finite differences and in time with a implicit Euler method and the ParaReal iteration. Parallel implementation, N number of processors and of time steps used to integrate over the interval $[0, 1]$. Displayed: speed-up percentage, speed-up and 2-norm of the error. Results after one iteration.

3 Parallel implementation and performance

We have implemented the ParaReal method in a FORTRAN/MPI code. We have performed tests with a simple one-dimensional heat equation and by coupling our FORTRAN/MPI implementation of ParaReal to an existing C++ code for extrusion of aluminium developed within the PDE solving environment DiffPack, [11]. The code can be found at the website `http://www.math.ntnu.no/~elenac/` for the case of the heat equation.

We compute the speed-up for the method as the quotient between the CPU time employed by the sequential integrator and the CPU time employed by the ParaReal implementation with N processors. The number of processors employed is always equal in the experiments to the number of time steps and therefore the accuracy of the integration increases with the number of processors used in the calculation. The speed-up is reported on the second row of each table below, while the corresponding speed-up percent is reported in the first row. The 2-norm of the error between the numerical solution obtained by sequential and the parallel solver is reported in the third row. We perform experiments with only one or two iterations of the ParaReal approach. The operators G (coarse) and F (fine) are implemented by integrating in time with an implicit Euler' s method with stepsize $\Delta T = T_{\text{fin}}/N$ and $\delta t := \Delta T/N$ respectively, where T_{fin} is the amplitude of the integration interval. The experiments have been performed on a SGI origin 3800 of type SMP with a total of 384 nodes. We could not run the experiments with a dedicated machine.

3.1 Parallel experiments: Heat equation

We consider the one-dimensional heat equation

$$T_t = T_{xx}, \quad x \in [0, 1], \quad t \in [0, 1],$$

$$T(x, 0) = 1, \quad u(0, t) = u(1, t) = 0$$

where T is the temperature field. When we discretize this in space with central finite differences, we obtain a system of linear ordinary differential equations which we here solve with ParaReal. The number of degrees of freedom in our numerical example is $N_{\text{dof}} = 10$. The results are reported in Tables 1 and 2. The corresponding finite difference code is reported at the website `http://www.math.ntnu.no/~elenac/`.

N	4	8	16	32
speedup perc.	31.42	20.60	15.83	14.94
speedup	1.25	1.64	2.53	4.78
error	.2721D-01	.1019D-01	.2537D-02	.5743D-03

Table 2: Heat equation discretized in time with central finite differences and in time with a implicit Euler method and the ParaReal iteration. Parallel implementation, N number of processors and of time steps used to integrate over the interval $[0, 1]$. Displayed: speed-up percentage, speed-up and 2-norm of the error. Results after two iterations.

N	4	8	16	32
speedup perc.	50.71	49.71	48.63	48.07
speedup	2.0285	3.97	7.78	15.38
error	.4123D-03	.1192D-05	.3228D-09	.1780D-11

Table 3: Stokes channel flow problem discretized in space with finite elements and in time with an implicit Euler method and the ParaReal iteration. Parallel implementation, N number of processors and of time steps used to integrate over the interval $[0, 1]$. Displayed: speed-up percentage, speed-up and 2-norm of the error. Results after one iteration.

3.2 Parallel experiments: Stokes channel flow with constant inflow profile

In the following two experiments the ParaReal approach has been combined with an existing object oriented code for solving Stokes and Navier-Stokes equations arising in problems of extrusion of aluminium. This code named *Extrud* was developed by Kvamsdal *et al.* [10]. To begin with we consider the following incompressible Stokes problem in two space dimensions:

$$\begin{aligned} u_t &= -\nabla p + \mu \Delta u, \\ \nabla \cdot u &= 0, \end{aligned}$$

with $(x, y) \in [0, 1]x[0, 1]$, $t \in [0, 1]$, $u(x, y, t)$ the flow velocity with $u(x, y, 0) = 0$, $u(x, y, t) = 1$ on the boundary with a given initial velocity profile, p the pressure field, μ the dynamic viscosity. The equations are discretized in space with finite elements where we use a mixed interpolation of the aluminium flow variables, with quadratic test- and trial functions for the velocities and linear functions for the pressure. The number of degrees of freedom used in the numerical example is $N_{\text{dof}} = 187$, $\mu = 1$. The results are reported in table 3.

3.3 Parallel experiments: Navier-Stokes extrusion problem

In this last experiment we solve the following coupled thermo-viscoplastic extrusion flow problem:

$$\begin{aligned} \rho(u_t + (u - u_g) \cdot \nabla u) &= -\nabla p + (2\mu(u, T)\epsilon(u)), \\ \nabla \cdot u &= 0, \\ \rho(T_t + (u - u_g) \cdot \nabla T) &= \nabla \cdot (k \nabla T) + (2\mu(u, T)\epsilon(u) : \epsilon(u)), \end{aligned}$$

N	4	8	16	32
speedup perc.	51.77	52.48	50.57	50.33
speedup	2.07	4.19	8.09	16.10
error	.4125D-09	.3954D-15	.3738D-15	.5143D-15

Table 4: Navier-Stokes extrusion problem discretized in space with finite elements and in time with an implicit Euler method and the ParaReal iteration. Parallel implementation, N number of processors and of time steps used to integrate over the interval $[0, 1]$. Displayed: speed-up percentage, speed-up and 2-norm of the error. Results after one iteration.

Here, the extrusion code uses an ALE formulation (Arbitrary Lagrangian- Eulerian formulation) to take into account the movement of the stem in the extrusion process, i.e. we have to handle a moving boundary problem where u_g is the grid (mesh) velocity. The thermo-viscoplastic extrusion problem is a non-Newtonian Navier-Stokes problem where the dynamic viscosity μ depends on the flow velocity u and the temperature field T. The strains are given as $\epsilon(u) = 1/2(\nabla u + (\nabla u)^T)$, whereas the conductivity k is assumed constant. The discretization of the aluminium flow variables u and p is the same as for the Stokes problem above, whereas for the temperature problem we use quadratic interpolation. The number of degrees of freedom was here $N_{\text{dof}} = 854$, corresponding to 375 nodes, and 84 elements.

3.4 Conclusions

We applied succesfully the ParaReal algorithm to a problem of fluid structure interaction in the context of extrusion of alluminium and we discussed fast convergence criteria for the method. Our parallel implementation shows that the ParaReal approach brings significant improvements in terms of the CPU time spent in the calculation, with savings up to 50%. However due to the intrinsic sequential nature of time integration problems the speed-up can only be suboptimal. A similar, but different analysis of the convergence of the ParaReal algorithm was recently derived independently in [8]. A preliminary version of our work was presented in [5].

Aknowledgmens

The authors wish to thank Kunt Morten Okstad (FEDEM Technology ASA) and Runar Holdahl (SINTEF ICT) for their crucial help in coupling the ParaReal FORTRAN/MPI code to the *Extrud* code.

References

[1] G. Bal and Y. Maday. A "parareal" time discretization for non-linear PDE's with application to the pricing of an American put. in l.f. Pavarino and a. Toselli, editors. recent developments in domain decomposition methods. *Lect. Notes Comput. Sci. Eng.*, 23:189–202, 2002.

[2] A. Bellen, R. Vermiglio, and M. Zennaro. Parallel ODE-solvers with stepsize control. *J. CAM*, 31:277–293, 1990.

[3] A. Bellen and M. Zennaro. Parallel algorithms for initial-value problems for difference and differential equations. *J. CAM*, 25:341–350, 1989.

[4] K. Burrage. *Parrallel and sequential methods for Ordinary Differential Equations.* Oxford University Press, 1995.

[5] E. Celledoni and T. Kvamsdal. Parallel time solver for the transient stokes problem. In *NSCM-17Proceedings of the 17th Nordic seminar on computational mechanics October 15-16, 2004, Stockholm, KTH.*, page 200 pp., 2004.

[6] P. Chartier and B. Philippe. A parallel shooting technique for solving dissipative odes. *Computing*, 51:209–236, 1993.

[7] C. Farhat and M. Chandesris. Time-decomposed parallel time-integrators: theory and feasibility studies for fluid, structure, and fluid-structure applications. *Int. J. Numer. Meth. Engng*, 58, 2003. 38 pp, to appear.

[8] M. J. Gander and S. Vandewalle. Analysis of the parareal time-parallel time-integration method. *SIAM J. Sci. Comput.*, 29(2):556–578 (electronic), 2007.

[9] R.A. Horn and C.R. Johnson. *Topics in matrix analysis.* Cambridge University Press, 1991.

[10] T. Kvamsdal, K. M. Okstad, and S. Abtahi. EXTRUD: A coupled heat and flow solver for 3d simulation of aluminium extrusion. *International Journal for Applied Mechanics and Engineering*, 7:293–310, 2002.

[11] H. P. Langtangen. *Computational partial differential equations*, volume 1 of *Texts in Computational Science and Engineering*. Springer-Verlag, Berlin, second edition, 2003. Numerical methods and Diffpack programming.

[12] J. L. Lions, Y. Maday, and G. Turinici. Resolution d'EDP par un schema en temps parareel. *C.R. Acad. Sci. Paris*, pages 1–6, 2001. Analyse numerique.

[13] G. Staff. Convergence and stability of the parareal algorithm: A numerical and theoretical investigation. Master's thesis, NTNU, 2003.