Journal of Computational Science 5 (2014) 76–84

Contents lists available at ScienceDirect

Journal of Computational Science

journal homepage: www.elsevier.com/locate/jocs

An adjoint-based scalable algorithm for time-parallel integration

Vishwas Rao*, Adrian Sandu

Computational Science Laboratory, Department of Computer Science, Virginia Polytechnic Institute and State University, Blacksburg, VA 24061, United States

ARTICLE INFO

Article history:
Received 9 October 2012
Received in revised form 19 February 2013
Accepted 17 March 2013
Available online 28 March 2013

Keywords:
Parareal
Adjoints
Sensitivity analysis

ABSTRACT

As parallel architectures evolve the number of available cores continues to increase. Applications need to display a high degree of concurrency in order to effectively utilize the available resources. Large scale partial differential equations mainly rely on a spatial domain decomposition approach, where the number of parallel tasks is limited by the size of the spatial domain. Time parallelism offers a promising approach to increase the degree of concurrency. 'Parareal' is an iterative parallel in time algorithm that uses both low and high accuracy numerical solvers. Though the high accuracy solvers are computed in parallel, the low accuracy ones are in serial.

This paper revisits the parallel in time algorithm [11] using a nonlinear optimization approach. Like in the traditional 'Parareal' method, the time interval is partitioned into subintervals, and local time integrations are carried out in parallel. The objective cost function quantifies the mismatch of local solutions between adjacent subintervals. The optimization problem is solved iteratively using gradient-based methods. All the computational steps – forward solutions, gradients, and Hessian-vector products – involve only ideally parallel computations and therefore are highly scalable.

The feasibility of the proposed algorithm is studied on three different model problems, namely, heat equation, Arenstorf's orbit, and the Lorenz model.

© 2013 Elsevier B.V. All rights reserved.

1. Introduction

Computational sciences use numerical models to simulate time-evolving systems, where the governing physical laws are formulated as partial differential equations (PDEs). The computations associated with initial value problems are typically sequential, and advance the solution one time step after another. This mimics the physical behavior of the system where the current state determines its future evolution.

Considerable effort is spent in developing large-scale numerical PDE models that can run efficiently on high-performance computing architectures. The ubiquitous strategy is to parallelize the workload in space by mapping different parts of the simulation grid onto individual compute nodes. In contrast, parallelizing in time is not common practice. In principle the simulation time interval can also be split into subintervals: this decomposes the initial value problem into a sequence of initial value subproblems. The challenge for solving these subproblems in parallel is that each of them requires a different initial condition.

The 'Parareal' algorithm proposed by Lions et al. [1] computes the intermediate initial conditions for each subproblem by a scalar, coarse numerical integration. The coarse (low cost, low accuracy) and fine (high cost, high accuracy) solvers are applied in succession, and each iteration of the algorithm reduces the global error of the solution. This algorithm was extensively applied for solving problems in fluids and structure [2], Navier–Stokes equations [3], for reservoir simulation [4]. Staff and Rønquist discuss the stability of the 'Parareal' algorithm in [5]. Convergence of 'Parareal' applied to PDEs is discussed in [6]. Gander and Vanderwalle [7] cast 'Parareal' as a multiple shooting method and a multigrid method. There have been attempts to parallelize different portions of the solution procedure in related problems in the areas of optimal control [8], parameter estimation [9], CFD applications [10]. The traditional 'Parareal' method obtains the fine solutions in parallel, but requires a *serial* coarse solution to be computed in every iteration.

In a quest to increase scalability we revisit the adjoint-based parallel in time algorithm of Maday and Turinici [11] formulated in the framework of nonlinear optimization. Gradient and the Hessian-vector product computations, needed in optimization, are carried out using adjoint sensitivity analysis. The main computational steps are ideally parallel, which makes the entire algorithm highly scalable. The proposed approach has the potential to become useful for parallelizing large scale problems like climate modeling and weather forecasting [12].

The remaining part of the paper is organized as follows. In Section 2 we introduce the problem, the notation, and the

* Corresponding author. Tel.: +1 5402605414.
E-mail addresses: visrao@cs.vt.edu, vishwas1984@gmail.com (V. Rao), sandu@vt.edu (A. Sandu).

1877-7503/$ – see front matter © 2013 Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.jocs.2013.03.004

traditional 'Parareal' algorithm. Section 3 presents the proposed adjoint-based 'Parareal' algorithm, and discusses the computation of first and second order derivative information needed in optimization. Numerical experiments are performed in Section 4. Section 5 draws conclusions and points to future work.

2. The traditional 'Parareal' algorithm

Consider the following ODE,

$$\mathbf{y}' = f(t, \mathbf{y}), \quad \mathbf{y}(t_{\text{initial}}) = \mathbf{y}_0, \quad t_{\text{initial}} \le t \le t_{\text{final}}. \tag{1}$$

2.1. Notation

In this sub-section we introduce notation which will be useful in understanding the parallel solution procedure. A numerical integration method (e.g., a Runge Kutta method) uses N time steps to solve the differential equation (1)

$$t_i = t_{i-1} + h_i, \quad i = 1, \ldots, N, \quad t_0 = t_{\text{initial}}, t_N = t_{\text{final}},$$

and computes the numerical solutions $\mathbf{y}_i \approx \mathbf{y}(t_{i-1})$ for $i = 0, \ldots, N$. We formally denote the numerical solution process that computes the solution at t_i from the solution at t_i by:

$$\mathbf{y}_i = \mathcal{M}_{i-1,i}(\mathbf{y}_{i-1}), \quad i = 1, \ldots, N. \tag{2}$$

Similarly, the set of consecutive numerical steps that evolve the solution from t_k to t_ℓ, $\ell > k$, will be denoted by

$$\mathbf{y}_\ell = \mathcal{M}_{k,\ell}(\mathbf{y}_k), \quad \forall \ \ \ell > k.$$

For example,

$$\mathbf{y}_{k+3} = \mathcal{M}_{k,k+3}(\mathbf{y}_k) = \mathcal{M}_{k+2,k+3}(\mathcal{M}_{k+1,k+2}(\mathcal{M}_{k,k+1}(\mathbf{y}_k))).$$

The serial solution procedure consists in applying N consecutive steps of the method, starting from the initial condition, to obtain the solution at the final time:

$$\mathbf{y}(t_{\text{final}}) \approx \mathbf{y}_N = \mathcal{M}_{0,N}(\mathbf{y}_0).$$

In the context of the 'Parareal' algorithm, we regard $\mathcal{M}$ to be a "fine" (high compute time and high accuracy) solver. Similarly, we consider a "coarse" (low accuracy and high efficiency) numerical process, and denote it by:

$$\mathbf{y}_i = \mathcal{G}_{i-1,i}(\mathbf{y}_{i-1}), \quad i = 1, \ldots, N. \tag{3}$$

2.2. Partitioning the simulation time interval

For a parallel-in-time solution we partition the simulation time interval $[t_{\text{initial}}, t_{\text{final}}]$ into M subintervals, with boundaries as shown below:

$$t_{\text{initial}} = T_0 < T_1 < \cdots < T_{M-1} < T_M = t_{\text{final}}. \tag{4}$$

We choose the subinterval boundaries such that they correspond to integer steps of the "fine" numerical method,

$$T_i = t_{\ell_i}, \quad i = 0, \ldots, M,$$

where ℓ_i are integers. This implies that $\ell_0 = 0$ and $\ell_M = N$. The first subinterval consists of the first ℓ_1 steps of the numerical method

$$[T_0, T_1] = [t_0, t_{\ell_1}].$$

The second subinterval consists of the next $\ell_2 - \ell_1$ steps of the numerical method

$$[T_1, T_2] = [t_{\ell_1}, t_{\ell_2}] = [t_{\ell_1}, t_{\ell_1+1}] \cup [t_{\ell_1+1}, t_{\ell_1+2}] \cup \cdots \cup [t_{\ell_2-1}, t_{\ell_2}],$$

and so on.

Consider the following approximations of the solution at the interval boundaries:

$$u_0 = \mathbf{y}_0; \quad u_i \approx \mathbf{y}(t_i^+), \quad i = 1, \ldots, M-1.$$

Then a numerical solution can be computed on each interval $[t_{i-1}, t_i]$ starting from the initial condition u_{i-1}:

$$\widetilde{u}_i = \mathcal{M}_{\ell_{i-1},\ell_i}(u_{i-1}), \quad \text{in parallel for } i = 1, \ldots, M. \tag{5}$$

Since each interval has its own known initial condition, the integrations (5) proceed independently and can be computed in parallel.

2.3. The traditional 'Parareal' iterations

Lions et al. [1] define the 'Parareal' algorithm using both the fine (2) and the coarse (3) numerical solution operators. Let $u_i^{(k)} \approx \mathbf{y}(T_i)$ denote the initial conditions for the interval $[T_i, T_{i+1}]$ at the kth iteration. The following iterative scheme is used to obtain improved numerical solutions

$$u_{i+1}^{(k+1)} = \mathcal{G}_{i,i+1}(u_i^{(k+1)}) + \mathcal{M}_{i,i+1}(u_i^{(k)}) - \mathcal{G}_{i,i+1}(u_i^{(k)}). \tag{6}$$

Note that the coarse integration is carried out *serially*, while the fine integration is carried out *in parallel* (5). The serial coarse solver represents a major bottleneck that prevents the scalability of traditional 'Parareal' (6) on a very large number of threads.

3. An optimization approach to time parallel discretizations

In this section we propose new parallel in time solution procedure derived in a nonlinear optimization framework.

3.1. Define a cost function to penalize jumps

The set of ODEs (5) lead to a solution that is piecewise continuous within each interval $[T_{i-1}, T_i]$. At each interval boundary T_i there is a jump between $\widetilde{u}_i \sim \mathbf{y}(T_i^-)$, the solution computed on $[T_{i-1}, T_i]$, and $u_i \sim \mathbf{y}(T_i^+)$, the initial condition for the next interval $[T_i, T_{i+1}]$. To obtain the solution of the original ODE we need to choose the initial conditions u_i such that the result of the integration on the current interval matches the initial conditions of the next interval,

$$u_i = \widetilde{u}_i, \quad i = 1, \ldots, M.$$

We define a cost function that penalizes the jumps across interval boundaries:

$$\mathcal{J}(u_1, \ldots, u_{M-1}) = \frac{1}{2} \sum_{i=1}^{N-1} (u_i - \mathcal{M}_{i-1,i}(u_{i-1}))^T \mathbf{R}_i^{-1} (u_i - \mathcal{M}_{i-1,i}(u_{i-1})). \tag{7}$$

Here $\mathbf{R}_i$'s are symmetric weight matrices and can be chosen as follows:

$$\mathbf{R}_i = \text{diag}_{1 \le j \le n} \ (Atol_j + Rtol_j |(u_i)_j|)^2,$$

where $Atol$ and $Rtol$ are the absolute and relative tolerances used in the ODE integration, respectively. Note that the cost function is greater than or equal to zero. The initial conditions for each subinterval determine the entire solution. In order to enforce continuity across boundaries, they are obtained by minimizing this cost function

$$(u_1^{\text{opt}}, \ldots, u_{M-1}^{\text{opt}}) = arg \min_{(u_1, \ldots, u_{M-1})} \mathcal{J}(u_1, \ldots, u_{M-1}). \tag{8}$$

Download English Version:

https://daneshyari.com/en/article/429379

Download Persian Version:

https://daneshyari.com/article/429379

Daneshyari.com