Applied Mathematics and Computation 267 (2015) 727–741

Contents lists available at ScienceDirect

Applied Mathematics and Computation

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

# A stencil-based implementation of Parareal in the C++ domain specific embedded language STELLA

Andrea Arteaga [a], Daniel Ruprecht [b,*], Rolf Krause [b]

[a] Center for Climate System and Modeling, ETH Zürich, Universitätstrasse 16, CH-8092 Zurich, Switzerland
[b] Institute of Computational Science, Università della Svizzera italiana, Via Giuseppe Buffi 13, CH-6900 Lugano, Switzerland

ARTICLE INFO

*Keywords:*
Parareal
STELLA
Parallel-in-time
Stencil computation
Speedup
Energy consumption

ABSTRACT

In view of the rapid rise of the number of cores in modern supercomputers, time-parallel methods that introduce concurrency along the temporal axis are becoming increasingly popular. For the solution of time-dependent partial differential equations, these methods can add another direction for concurrency on top of spatial parallelization. The paper presents an implementation of the time-parallel Parareal method in a C++ domain specific language for stencil computations (STELLA). STELLA provides both an OpenMP and a CUDA backend for a shared memory parallelization, using the CPU or GPU inside a node for the spatial stencils. Here, we intertwine this node-wise spatial parallelism with the time-parallel Parareal. This is done by adding an MPI-based implementation of Parareal, which allows us to parallelize in time across nodes. The performance of Parareal with both backends is analyzed in terms of speedup, parallel efficiency and energy-to-solution for an advection–diffusion problem with a time-dependent diffusion coefficient.



## 1. Introduction

For the numerical solution of initial value problems on massively parallel computers, concurrency is typically introduced by distributing the spatial degrees of freedom of the solution over multiple nodes, processors or cores. In the case of time-dependent partial differential equations, this decomposition is usually induced by a decomposition of the underlying spatial computational domain and therefore this approach is referred to as "spatial parallelism". The approximate solution is then computed by advancing forward in time from some given initial value by time marching methods, e.g. Runge–Kutta or multi-step schemes, which compute time-step after time-step in a serial fashion.

As the number of cores and processors in state-of-the-art supercomputers continues to rise quickly, the time dimension is more and more becoming a serial bottleneck in at least the following sense: If, for a given problem, the resolution is increased, the increase in computational cost *per time-step* can be compensated by using more cores for the parallelization of the spatial problem – at least, as long as the used method and its implementation show good weak scaling behavior. When the temporal resolution is increased in order to match, e.g. stability or accuracy constraints, however, obviously more time-steps have to be computed to reach the same final time. Here, the resulting increase in computational cost cannot be reduced by spatial parallelization. As discussed in [1], the parallel complexity of time-stepping is always at least $\Theta(N_t)$, with $N_t$ being the number of time-steps. Following [1], $\Theta(x)$ denotes "a positive quantity whose leading order term is proportional to $x$".

* Corresponding author.
*E-mail address:* daniel.ruprecht@usi.ch (D. Ruprecht).

http://dx.doi.org/10.1016/j.amc.2014.12.055
0096-3003/

Because of the eventual saturation of spatial strong scaling and the resulting minimum time required per time-step, the lower constant in the estimate is bounded away from zero as the number of processors increases and the minimum computing time per step is approached.

Hence, methods that introduce some degree of concurrency along the time axis are becoming increasingly popular. Very early ideas go back to the 1960's [2,3]. In the 1980's and 1990's, parabolic and space–time multigrid methods [4,5] and parallel multiple shooting for initial value problems [6] were studied. A more recent and today very widely studied method is Parareal [7], which is also the method this paper is concerned with. However, a number of other methods exist, e.g. the "parallel implicit time algorithm" PITA [8], "revisionist deferred corrections" or RIDC [9,10], the "parallel full approximation scheme in space and time" PFASST [11,12], space–time parallel solvers for periodic problems [13,14] and newer approaches to space–time multigrid [15,16]. A recent overview can be found in [17].

There is literature available on the performance of time-parallel methods for a wide range of benchmark problems from different applications, e.g. finance [18], quantum chemistry [19] or plasma-physics [20]. Also, particularly for Parareal, there are a number of papers concerned with analysis of the method [21–25] or the development of improvements or modifications [26–31].

Not much literature exists studying, let alone comparing, different implementation strategies for time-parallel methods: there are obviously many different possibilities to map space–time parallelism to a computer consisting of a large number of multi-core nodes, probably also equipped with accelerators. Small-scale benchmarks for an all-MPI implementation of Parareal with spatial parallelization are discussed in [32], but with somewhat inconclusive performance results. Good performance on close to half a million cores of an implementation of PFASST and parallel multigrid in space using only MPI has been demonstrated in [33]. An MPI-based implementation of PFASST combined with a hybrid tree-code on more than two hundred thousand cores is studied in [34]. Smaller scale benchmarks for a hybrid space–time parallel approach, combining OpenMP in time and MPI in space, are reported in [35] for RIDC and in [36] for Parareal. In [37], the original method by Nievergelt [2] is implemented on GPUs. As performance metric, all publications so far consider only speedup, other metrics like energy-to-solution, memory footprint etc. have not yet been analyzed for time parallelization. Some concerns about the memory requirements of time-parallel methods on possible Exascale computers have been voiced in [38].

In this paper, we introduce a stencil-based implementation of Parareal in the recently developed C++ domain specific embedded language STELLA (STEncil Loop LAnguage) [39]. STELLA provides functionality for the application of finite difference stencil operators on structured grids, including an OpenMP and CUDA backend to run the computations on a multi-core CPU or a GPU. Stencil computation on structured grids is one of the "seven dwarfs" of high-performance computing defined by Collela in 2004, see e.g. [40]. The backends parallelize the stencil computation in a node across either its cores or by using an attached GPU. The implementation of Parareal in STELLA is then employed to parallelize in time across nodes: Each node holds one "timeslice" in Parareal and performs the corresponding computations exploiting the spatial concurrency from the corresponding STELLA backend while MPI is used to communicate between nodes. In order to optimize data transfer between GPUs, the CUDA-aware MPI implementation MPICH by Cray was used, which supports *GPUDirect*. This technology avoids the need for copying the data from the GPU memory into a temporary buffer of the host memory and transfers instead the data directly to the network interface, thus reducing the latency and increasing the bandwidth. In summary, we employ a paradigm combining a MPI-based distributed memory parallelization in time with an OpenMP or CUDA based shared memory parallelization in space. Performance of Parareal based on both the CPU and the GPU backend is investigated in terms of speedup, parallel efficiency and energy-to-solution and compared to theoretically predicted values. To our knowledge, this is the first time that the performance of a parallel-in-time method in terms of energy-to-solution is analyzed.

## 2. Methods and implementation

Below, we comment briefly on serial time-stepping and then describe Parareal, including a theoretical model for expected speedup and energy overhead. A short introduction of the STELLA language and the corresponding stencil-based implementation of Parareal is given.

### 2.1. Time-marching

Consider an initial value problem of the form

$$u_t = f(u(t), t), \quad u(0) = u_0 \in \mathbb{R}^d, \quad 0 \leqslant t \leqslant T, \tag{1}$$

where, in the examples below, the right hand side $f$ stems from the spatial discretization of a partial differential equation. To fix notation for the introduction of Parareal, let the time-interval $[0, T]$ be decomposed into $N_p$ so-called "time-slices" $[t_n, t_{n+1}], n = 0, \ldots, N_p - 1$ and let $u_n$ be an approximation of the solution $u$ of (1) at $t_n$, that is $u_n \approx u(t_n)$. Note that for Parareal, $N_p$ is identical to the number of processors used *in time*. It should not be confused with the number of processors *in space* or the total number of processors.

We denote by $\mathcal{F}_{\delta t}$ a so-called "fine integrator", typically a higher order method with a fine time-step size $\delta t$. That is, for some given approximation $u_n$ at $t_n$, denote by

$$u_{n+1} = \mathcal{F}_{\delta t}(u_n, t_{n+1}, t_n), \tag{2}$$