Hybrid Space-Time Parallel Solution of Burgers' Equation

Rolf Krause[1] and Daniel Ruprecht[1,2]

Key words: Parareal, spatial parallelization, hybrid parallelization

1 Introduction

Many applications in high performance computing (HPC) involve the integration of time-dependent partial differential equations (PDEs). Parallelization in space by decomposing the computational domain is by now a standard technique to speed up computations. While this approach can provide good parallel scaling up to a large number of processors, it nevertheless saturates when the subdomains become too small and the time required for exchanging data starts dominating. Regarding the anticipated massive increase of available cores in future HPC systems, additional directions of parallelization are required to further reduce runtimes. This is especially important for time-critical applications like, for example, numerical weather prediction, where there exists a very strict constraint on the total time-to-solution for a forecast in order to be useful.

One possibility for providing such an additional direction of parallelization are parallel-in-time integration schemes. A popular scheme of this type is Parareal, introduced in [1, 7]. It has been applied successfully to a broad range of problems and also undergone thorough analytical investigation. A large number of corresponding references can be found, for example, in [6, 9].

While numerous works exist dealing with different aspects of Parareal in a purely time-parallel approach, there seem to be few studies that address the combination of Parareal with spatial parallelization, in particular with a focus on implementation. First results on combining Parareal with spatial domain decomposition are presented in [8]. While scaling of the algorithm is discussed, no runtimes are reported. In [12, 13], computing times for a pure MPI-based combination of Parareal with spatial domain decomposition for the two-dimensional Navier-Stokes equations are given, but with ambiguous results: Either a pure time-parallel or a pure space-parallel approach performed best, depending on the problem size. In [4], the capability of a purely MPI-based approach to speed up simulations for the 3D Navier-Stokes equations beyond the saturation of the spatial parallelization is shown. Extensive scaling tests for the "revisionist deferred correction" method (RIDC) for the

Institute of Computational Science, Università della Svizzera italiana, Via Giuseppe Buffi 13, 6900 Lugano, Switzerland, {rolf.krause,daniel.ruprecht}@usi.ch · Mathematisches Institut, Heinrich-Heine-Universität, Universitätsstrasse 1, 40225 Düsseldorf, Germany

linear heat equation, also in combination with domain decomposition, can be found in [3].

The present paper investigates the performance of a combination of a shared memory implementation of Parareal featuring explicit integrators with an MPI-based parallelization of a stencil-based spatial discretization into a hybrid (see [10]) space-time parallel method. The code is an extension of the purely time-parallel, OpenMP-based implementation used in [11]. Using shared memory for Parareal avoids communication of volume data by message passing and thus reduces the memory footprint of the code.

2 Algorithm and Implementation

The starting point for Parareal is an initial value problem

$$\frac{d\mathbf{q}}{dt} = \mathbf{f}(\mathbf{q}),\ \mathbf{q}(0) = \mathbf{q}_0 \in \mathbb{R}^d, \tag{1}$$

where in the present work, the right hand side $\mathbf{f}$ stems from the spatial finite difference discretization of some PDE on a rectangular domain $\Omega \subset \mathbb{R}^2$. The spatial parallelization uses a standard non-overlapping decomposition of the domain, allowing for a distributed computation of $\mathbf{f}(\mathbf{q})$, where every MPI-process handles the degrees-of-freedom of one subdomain and ghost-cell values are exchanged at the boundaries. The implementation described below can be used for all integrators that involve only straightforward evaluations of $\mathbf{f}$, that is explicit methods or implicit methods where the arising linear or nonlinear system is solved with e.g. a fixed point iteration. For more complex solvers, e.g. a multi-grid method, a hybrid strategy will be more involved, because other parts like restriction or interpolation would have to be included in the hybrid paradigm as well.

2.1 Parareal

Parareal allows one to parallelize the integration of (1) by combining a number of time-steps into one coarse time-slice and performing an iteration where multiple time-slices are treated concurrently. Let $\mathscr{F}_{\delta t}$ denote a numerical integration scheme of suitable accuracy, using a time-step δt. A second integration scheme is required, typically called $\mathscr{G}_{\Delta t}$, using a time-step $\Delta t \gg \delta t$, which has to be much cheaper in terms of computation time but can also be much less accurate. Denote by

$$\tilde{\mathbf{q}}_{\mathrm{g}} = \mathscr{G}_{\Delta t}(\mathbf{q}, \tilde{t}, t), \quad \tilde{\mathbf{q}}_{\mathrm{f}} = \mathscr{F}_{\delta t}(\mathbf{q}, \tilde{t}, t) \tag{2}$$

the result of integrating forward in time from an initial value $\mathbf{q}$ at time t to a time $\tilde{t} > t$ using $\mathscr{G}_{\Delta t}$ or $\mathscr{F}_{\delta t}$. Parareal uses $\mathscr{G}_{\Delta t}$ to produce approximate solutions at nodes

Algorithm 1 Parareal algorithm implemented with OpenMP using N_c threads

```
1: $\mathbf{q}_0^0 = \mathbf{q}_0$, $k := 0$
2: for $i = 0$ to $N_\mathrm{c} - 1$ do
3:     $\mathbf{q}_{i+1}^0 = \mathscr{G}_{\Delta t}(\mathbf{q}_i^0, t_{i+1}, t_i)$
4: end for
5: repeat
6:     omp parallel for
7:     for $i = 0$ to $N_\mathrm{c} - 1$ do
8:         $\tilde{\mathbf{q}}_{i+1}^k = \mathscr{F}_{\delta t}(\mathbf{q}_i^k, t_{i+1}, t_i)$
9:     end for
10:    omp end parallel for
11:    for $i = 0$ to $N_\mathrm{c} - 1$ do
12:        $\mathbf{q}_{i+1}^{k+1} = \mathscr{G}_{\Delta t}(\mathbf{q}_i^{k+1}, t_{i+1}, t_i) + \tilde{\mathbf{q}}_{i+1}^k - \mathscr{G}_{\Delta t}(\mathbf{q}_i^k, t_{i+1}, t_i)$
13:    end for
14:    $k := k + 1$
15: until $k = N_\mathrm{it}$
```

$(t_i)_{i=0,\dots,N_\mathrm{c}}$ of a coarse temporal mesh (lines 2 – 4 in Algorithm 1). These guesses are then used as initial values for running $\mathscr{F}_{\delta t}$ concurrently on all N_c time intervals $[t_i, t_{i+1}]$ (lines 6–10). A correction is then propagated sequentially by another sweep of $\mathscr{G}_{\Delta t}$ (lines 11 – 13). The procedure is iterated and converges towards the solution that would be obtained by running $\mathscr{F}_{\delta t}$ sequentially from t_0 to t_{N_c}. For a detailed explanation and properties of the algorithm we refer to [6] and references therein. Note that an MPI-based implementation of Parareal requires communication of full volume data in line 12, which is avoided by the shared memory parallelization in time used here.

For a given time interval $[t_0, t_{N_\mathrm{c}}]$, denote by N_f the number of fine steps required to integrate from $t = t_0$ to $t = t_{N_\mathrm{c}}$, by τ_c and τ_f the execution time of one single coarse or fine time-step and by N_it the number of performed iterations. Further, assume that $\mathscr{G}_{\Delta t}$ always performs one single step, so that N_c is also the number of coarse steps between t_0 and t_{N_c}. The speedup obtainable by Parareal for a given number of processors can be estimated by

$$s(N_p) \approx \frac{1}{(1 + N_\mathrm{it}) \frac{N_\mathrm{c}}{N_\mathrm{f}} \frac{\tau_\mathrm{c}}{\tau_\mathrm{f}} + \frac{N_\mathrm{it}}{N_\mathrm{p}}} \le \frac{N_\mathrm{p}}{N_\mathrm{it}}. \tag{3}$$

Note that the maximum parallel efficiency is bounded by $1/N_\mathrm{it}$. Because of this limit, Parareal is commonly considered on top of a saturated spatial parallelization for problems where minimizing time-to-solution is critically important. Recently, a new scheme named PFASST, based on a combination of Parareal with spectral deferred correction methods, has been introduced in [5].

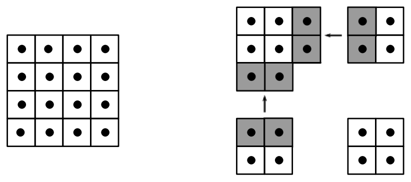

Fig. 1 Sketch of a decomposition of a 4x4 cells domain (left) into 4 sub-domains with 2x2 cells each (right). Cell-centers are marked as dots. The grey cells mark the halo values that have to be send to the processor handling the upper left sub-domain before each evaluation of **f** if a simple 5-point star is used. For stencils with wider support, the halos also need to be wider and communication between diagonally adjacent processors might be required. In the time-parallel OpenMp version, halo data has to be exchanged for each thread. In the implementation used here, the master thread handles all halo exchanges as sketched in Figure 2.

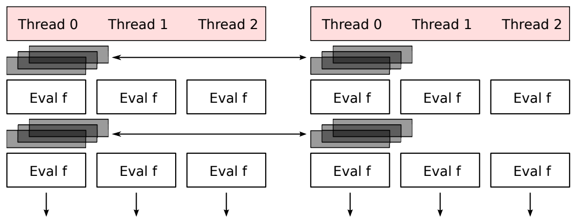

Fig. 2 Flow chart of halo exchange in funneled mode with 2 nodes, each running 3 threads: Before each evaluation of the right hand side **f**, the master thread (thread 0) exchanges up-to-date halo values (represented by three grey bars) for all threads with the other node. The other threads are idle during communication. After communication has finished, all threads continue with evaluating **f**. Synchronization is achieved by the OpenMp BARRIER directive while MPI calls are enclosed in MASTER directives to ensure they are only executed by the master thread.

2.2 Implementation

For the OpenMP-based parallelization sketched in Algorithm 1 to be efficient, the implementation of the fine propagator $\mathscr{F}_{\delta t}$ has to be suitably designed for multithreading. This involves a number of technical issues like taking care of "non-uniform memory access" (NUMA) inside compute nodes by ensuring that the data a core accesses while running a thread is located in the memory closest to this core. A detailed introduction into efficient OpenMp programming can be found in [2].

2.2.1 Ghost-Cell Exchange

To combine the OpenMp implementation of Parareal with parallelization in space, frequent exchange of boundary values between processors handling different sub-domains is necessary: Figure 1 sketches the decomposition of a 4×4 cell domain into 4 sub-domains. In order to evaluate e.g. a standard five-point stencil discrete Laplacian, every processor needs to receive a "halo" of up-to-date values before evaluating the stencil (halo cells for the upper left sub-domain are marked in grey in

Figure 1). Communication of these halo data is done here through message passing using MPI.

For using MPI in conjunction with OpenMp, different options exists for the initialization of the MPI library that govern how MPI routines can be called by different threads. Here, we use the option MPI_THREAD_FUNNELED which allows only the master thread to make calls to the MPI library. As the ghost-cell communication in $\mathscr{F}_{\delta t}$ takes place in the multithreaded part of the code, suitable OpenMP directives have to be used to synchronize threads and ensure compliance with the funneled option (OMP_BARRIER and OMP_MASTER). The coarse integrator is outside the parallel OpenMp region in Algorithm 1 so that no thread synchronization is required there. Organization of the ghost-cell exchange is sketched in Figure 2: Prior to every evaluation of the right hand side function $\mathbf{f}$, the master thread (thread 0) exchanges halo data for all threads on the node. While the master thread is busy communicating, the other threads are idle. This "idle threads problem" is one of the drawbacks of the funneled approach pointed out in [10]. Then, after the master thread has finished communicating, all threads continue with the computation of $\mathbf{f}$ and update the solution according to the integration method used for $\mathscr{F}_{\delta t}$. After every update (in case of a Runge-Kutta method for example, that means after every stage), the new halo values have to be exchanged again by the master thread before the next evaluation of $\mathbf{f}$ and so on.

3 Numerical Results

The performance of the hybrid space-time parallel approach is analyzed here for the two-dimensional, nonlinear, viscous Burgers equation

$$u_t + uu_x + uu_y = \nu \Delta u \tag{4}$$

on a domain $[-2,2] \times [-2,2]$ with initial value

$$u(x,y,0) = \sin(2\pi x)\sin(2\pi y), \tag{5}$$

a parameter $\nu = 0.02$, a mesh width $\Delta x = \Delta y = 1/40$ and periodic boundary conditions. A two-dimensional decomposition of the domain into square or rectangular subdomains, depending on the number of MPI-processes, is performed and a cartesian communicator for MPI is used. Parareal uses time-steps $\Delta t = 2 \times 10^{-3}$ and $\delta t = 2 \times 10^{-5}$. For $\mathscr{G}_{\Delta t}$, the spatial discretization uses 3rd-order upwind finite differences for the advection term and 2nd-order centered differences for the Laplacian, while $\mathscr{F}_{\delta t}$ uses a 5th-order upwind stencil for the advection and a 4th-order centered stencil for the Laplacian. Hence, a two-cell wide halo has to be exchanged in the coarse and a three-cell wide halo in the fine propagator. The simulations are run until $T = 0.5$ and $\mathscr{G}_{\Delta t}$ always performs one single step per coarse interval, so the number of restarts of Parareal depends on the number of threads assigned for the temporal parallelization. A forward Euler scheme is used for $\mathscr{G}_{\Delta t}$ and a Runge-Kutta-2

# MPI-P	time-serial	hybrid Parareal	speedup	# MPI-P	time-serial	hybrid Parareal	speedup
1	59.9 s	29.5 s	2.0	1	16.4 s	7.3 s	2.2
2	34.6 s	15.4 s	2.2	2	10.5 s	4.9 s	2.1
4	21.2 s	9.4 s	2.3	4	6.9 s	3.3 s	2.1
8	14.2 s	6.0 s	2.4	8	4.7 s	2.2 s	2.1
16	9.2 s	4.2 s	2.2	16	3.3 s	1.5 s	2.2
20	9.5 s	–	–	20	4.5 s	–	–

Table 1 Runtimes of the time-serial code and the hybrid Parareal code using 8 threads on each node for different numbers of spatial sub-domains, each corresponding to one MPI process. Shown are runtimes for a grid with 160×160 cells (left) and for a grid with 80×80 cells (right). Note that using more than 16 sub-domains no longer reduces runtime of the serial code in both cases.

scheme for $\mathscr{F}_{\delta t}$. To assess accuracy, a reference solution with $\delta t/10$ is computed sequentially. With a fixed number of $N_{\text{it}} = 3$ in Parareal, the relative $\|\cdot\|_\infty$-error of the time-parallel solution is $\varepsilon_{\text{para}} \approx 2.2 \times 10^{-8}$ and of the time-serial solution $\varepsilon_{\text{seq}} \approx 1.8 \times 10^{-8}$, so that both solutions are of comparable accuracy. The coarse integrator run alone results in $\varepsilon_{\text{coarse}} \approx 2.9 \times 10^{-2}$.

The used machine is a cluster consisting of 42 nodes, each containing 2 quad-core AMD Opteron CPUs with 2,700 MHz and 16 GB RAM per node. In the example below, the time parallelization always uses eight threads per node, in order to utilize one full node. The nodes are connected by an INFINIBAND network.

3.1 Runtimes and Scaling

Reported runtimes are measured with the MPI_WTIME routine provided by MPI and do not contain I/O operations.

3.1.1 Speedup from Parareal

With the used parameters, the speedup obtainable by Parareal using eight threads is bounded by $s \leq 2.57$ according to (3). The ratio $\tau_c/\tau_f = 0.35$ has been determined experimentally fby running $\mathscr{G}_{\Delta t}$ and $\mathscr{F}_{\delta t}$ serially on a single core. The value varies when using multiple processes, but the effects of the variation on the speedup estimate are small. Table 1 (left) shows the runtimes of the time-serial and the hybrid Parareal solution for different numbers of subdomains and corresponding MPI-processes. To further illustrate performance of the approach, runtimes for the 80×80 cell mesh are also shown (right). Runtimes obtained for a 40×40 mesh not shown here indicate similar speedups from Parareal using one, two and four MPI-processes as well as no further reduction of runtime of the time-serial code if more than four MPI-processes are used.

While the time-serial solution assigns each process to one core, the time-parallel solution assigns each process to one node and uses the eight cores inside the node

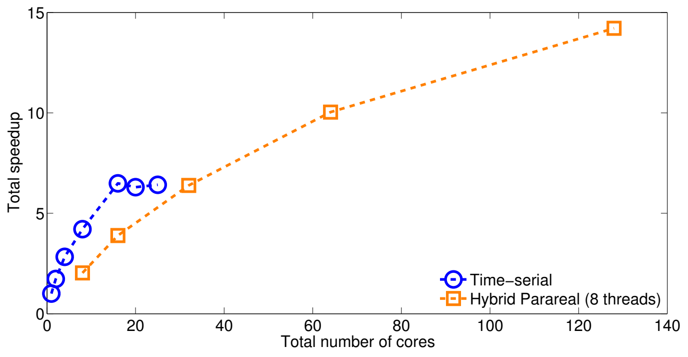

Fig. 3 Total speedup achieved by the space-parallel, time-serial (blue) and the hybrid space-time-parallel scheme (red) depending on the total number of used cores for the 160×160 cell mesh.

for the temporal parallelization. In both cases, the speedups from Parareal actually achieved by the hybrid implementation are between 78% and 93% of the theoretical maximum, despite the overhead caused by the funneled mode, supporting the efficiency of the hybrid space-time parallel approach.

3.1.2 Total scaling

As discussed above, one essential motivation for time-parallel schemes is to provide an additional direction of parallelization to achieve further reduction of time-to-solution after spatial parallelization saturates. Figure 3 shows the total speedup, that is compared against the time-serial solution on one core, for the time-serial and hybrid Parareal scheme. Because the considered problem is quite small and the underlying stencil-based discretization is comparably cheap to evaluate in terms of computation time, the pure spatial parallelization scales only to 16 cores (cf. Table 1). Beyond that point, using more cores does not further reduce runtime. Also, near perfect scaling is seen only up to two cores, after this the parallel efficiency is noticeable less than one. Note that the slow increase in speedup for the hybrid scheme is caused by the efficiency bound (3) of Parareal: For lower numbers of cores where the spatial parallelization is not yet saturated, the time-serial version performs better, because the efficiency of the parallelization in space, although no longer optimal, is still better than that of the time-parallel scheme. The advantage of the space-time-parallel scheme is that it can provide a significantly greater overall speedup. Hence, for a time-critical application where minimizing time-to-solution is of paramount importance and a purely spatial parallelization does not provide sufficient runtime reduction, a space-time parallel scheme can reduce runtime below some critical threshold if sufficient computational resources are available. The example clearly demonstrates the potential of the hybrid space-time parallelization to provide runtime reductions beyond the saturation of the space parallelization.

4 Conclusions

A shared memory implementation of the Parareal parallel-in-time integration scheme is combined with a standard distributed memory parallelization of a stencil-based spatial discretization. In the resulting hybrid space-time parallel scheme, each spatial subdomain is handled by one MPI-process which is assigned to one compute node. The time-slices from Parareal are assigned to different threads spawned by the process, with each thread running on one core of the node. The capability of the hybrid implementation to provide runtime reduction beyond the saturation of the spatial parallelization is documented.

Acknowledgements This work is funded by the Swiss "High Performance and High Productivity Computing" initiative HP2C.

References

1. Bal, G., Maday, Y.: A "parareal" time discretization for non-linear PDE's with application to the pricing of an american put. In: L. Pavarino, A. Toselli (eds.) Recent Developments in Domain Decomp. Meth., *LNCSE*, vol. 23, pp. 189–202. Springer Berlin (2002)
2. Chapman, B., Jost, G., van der Pas, R.: Using OpenMp: Portable shared memory parallel programming. Scientific and Engineering Computation Series. The MIT press, Cambridge, London (2008)
3. Christlieb, A.J., Haynes, R., Ong, B.W.: A parallel space-time algorithm. SIAM Journal on Scientific Computing **34**, C233–C248 (2012)
4. Croce, R., Ruprecht, D., Krause, R.: Parallel-in-space-and-time simulation of the three-dimensional, unsteady Navier-Stokes equations for incompressible flow. ICS-Preprint 2012-03 (2012)
5. Emmett, M., Minion, M.L.: Toward an efficient parallel in time method for partial differential equations. Comm. App. Math. and Comp. Sci. **7**, 105–132 (2012)
6. Gander, M.J., Vandewalle, S.: Analysis of the parareal time-parallel time-integration method. SIAM J. Sci. Comp. **29**(2), 556–578 (2007)
7. Lions, J.L., Maday, Y., Turinici, G.: A "parareal" in time discretization of PDE's. C. R. Acad. Sci. – Ser. I – Math. **332**, 661–668 (2001)
8. Maday, Y., Turinici, G.: The parareal in time iterative solver: A further direction to parallel implementation. In: R. Kornhuber, et al. (eds.) Domain Decomposition Methods in Science and Engineering, *LNCSE*, vol. 40, pp. 441–448. Springer, Berlin (2005)
9. Minion, M.L.: A hybrid parareal spectral deferred corrections method. Comm. App. Math. and Comp. Sci. **5**(2), 265–301 (2010)
10. Rabenseifner, R., Hager, G., Jost, G.: Hybrid MPI/OpenMP parallel programming on clusters of multi-core SMP nodes. In: 17th Euromicro International Conference on Parallel, Distributed and Network-based processing, pp. 427–436 (2009)
11. Ruprecht, D., Krause, R.: Explicit parallel-in-time integration of a linear acoustic-advection system. Computers & Fluids **59**, 72–83 (2012)
12. Trindade, J.M.F., Pereira, J.C.F.: Parallel-in-time simulation of the unsteady Navier-Stokes equations for incompressible flow. Int J. Numer. Meth. Fluids **45**, 1123–1136 (2004)
13. Trindade, J.M.F., Pereira, J.C.F.: Parallel-in-time simulation of two-dimensional, unsteady, incompressible laminar flows. Num. Heat Trans., Part B **50**, 25–40 (2006)