MPI–OpenMP algorithms for the parallel space–time solution of Time Dependent PDEs

Ronald D. Haynes[1] and Benjamin Ong[2]

1 Introduction

Modern high performance computers offer hundreds of thousands of processors that can be leveraged, in parallel, to compute numerical solutions to time dependent partial differential equations (PDEs). For grid-based solutions to these PDEs, domain decomposition (DD) is often employed to add spatial parallelism [18].

Parallelism in the time variable is more difficult to exploit due to the inherent causality. Recently, researchers have explored this issue as a means to improve the scalability of existing parallel spatial solvers applied to time dependent problems. There are two general approaches to combining temporal parallelism with spatial parallelism. The parareal method [10, 15, 14], a so-called "parallel across the problem" approach, decomposes a time domain into smaller temporal subdomains. Parareal alternates between applying a coarse (relatively fast) sequential solver to compute an approximate (not very accurate) solution, and applying a fine (expensive) solver on each temporal subdomain in parallel. Alternatively, one can consider "parallel across the step" methods. Examples of such approaches include the computation of intermediate Runge–Kutta stage values in parallel [16], and Revisionist Integral Deferred Correction (RIDC) methods, which are the family of parallel time integrators considered in this paper.

This paper discusses the implementation details and profiling results of the parallel space–time RIDC-DD algorithm described in [17]. Two hybrid OpenMP – MPI frameworks are discussed: (i) a more traditional fork-join approach of combining threads before doing MPI communications, and (ii) a threaded MPI communications framework. The latter framework is highly desirable because existing (spatially parallel) legacy software can be easily integrated with the parallel time integrator. Numerical experiments measure the communication overhead of both frameworks, and demonstrate that the fork-join approach scales well in space and time. Our results indicate that one should strongly consider temporal parallelization for the solution of time dependent PDEs.

2 Review

The authors are interested in the general class of PDEs

Memorial University of Newfoundland, St. John's, Newfoundland, Canada rhaynes@mun.ca · Michigan State University, Institute for Cyber-Enabled Research, e-mail: ongbw@msu.edu

$$u_t = f(t,u) + \Delta u, \quad x \in \Omega \subset \mathbb{R}^3, \quad u(x,0) = u_0(x), \quad u(z,t) \text{ specified}, \quad z \in \partial\Omega.$$

A common semi–implicit time discretization involves treating the diffusion term implicitly and the nonlinearity $f(t,u)$ explicitly. Therefore, without loss of generality, we describe the application of our method to the linear heat equation in one spatial dimension $x \in [0,1]$ and $t \in [0,T]$,

$$u_t = u_{xx},\, u(t,0) = g_0(t),\, u(t,1) = g_1(t),\, u(0,x) = u_0(x). \tag{1}$$

The actual numerical results in §4 are presented for the 2D heat equation.

2.1 RIDC

RIDC methods [5, 6] are a family of parallel time integrators that can be broadly classified as predictor corrector algorithms [9, 2]. The basic idea is to simultaneously compute solutions to the PDE of interest and associated error PDEs using a low order time integrator. We first review the derivation of the error equation.

Suppose $v(t,x)$ is an approximate solution to (1), and $u(t,x)$ is the (unknown) exact solution. The error in the approximate solution is $e(t,x) = u(t,x) - v(t,x)$. We define the residual as $\varepsilon(t,x) = v_t(t,x) - v_{xx}(t,x)$. Then the time derivative of the error satisfies $e_t = u_t - v_t = u_{xx} - (v_{xx} + \varepsilon)$. The integral form of the error equation,

$$\left[e + \int_0^t \varepsilon(\tau,x)\, d\tau\right]_t = (v+e)_{xx} - v_{xx}, \tag{2}$$

can then be solved for $e(t,x)$ using the initial condition $e(0,x) = 0$. The correction $e(t,x)$ is combined with the approximate solution $v(t,x)$ to form an improved solution. This improved solution can be fed back in to the error equation (2) and the process repeated until a sufficiently accurate solution is obtained. It has been shown that each application of the error equation improves the order of the overall method, provided the integral is approximated with sufficient accuracy using quadrature [7].

We introduce some notation to identify the sequence of corrected approximations. We use $v^{[0]}(t,x)$ to denote the initial approximate solution obtained by solving the physical PDE (1) using a low order integrator. The approximate solution $v^{[0]}(t,x)$ has an error $e^{[0]}(t,x)$ that is approximated by solving equation (2). The first corrected solution, $v^{[1]}(t,x)$, satisfies $v^{[1]}(t,x) = v^{[0]}(t,x) + e^{[0]}(t,x)$. In general, the error from the pth correction equation is used to construct the $(p+1)st$ approximation, $v^{[p+1]}(t,x) = v^{[p]}(t,x) + e^{[p]}(t,x)$. Hence, equation (2) can be expressed as

$$\left[v^{[p+1]} - \int_0^t v_{xx}^{[p]}(\tau,x)\, d\tau\right]_t = v_{xx}^{[p+1]} - v_{xx}^{[p]}. \tag{3}$$

We compute a low order prediction, $v^{[0],n+1}$, for the solution of (1) at time t_{n+1} using a first order backward Euler discretization (in time):

$$v^{[0],n+1} - \Delta t\, v_{xx}^{[0],n+1} = v^{[0],n},\ v^{[0],n+1}(a) = g_0(t^{n+1}),\ v^{[0],n+1}(b) = g_1(t^{n+1}), \quad (4)$$

with $v^{[0],0}(x) = u_0(x)$. With some algebra, a first order backward Euler discretization of equation (3) gives the update, $v^{[p+1],n+1}$, as

$$v^{[p+1],n+1} - \Delta t\, v_{xx}^{[p+1],n+1} = v^{[p+1],n} - \Delta t\, v_{xx}^{[p],n+1} + \int_{t^n}^{t_{n+1}} v_{xx}^{[p]}(\tau,x)\, d\tau, \quad (5)$$

with $v^{[p+1],n+1}(a) = g_0(t^{n+1})$ and $v^{[p+1],n+1}(b) = g_1(t^{n+1})$. The integral in equation (5) is approximated using a sufficiently high order quadrature rule [7].

Parallelism in time is possible because the PDE of interest (4) and the error PDEs (5) can be solved simultaneously, after initial startup costs. The idea is to fill out the memory footprint, Figure 1, before marching solutions to (4) and (5) in a pipe–line fashion, see [5] for more details.

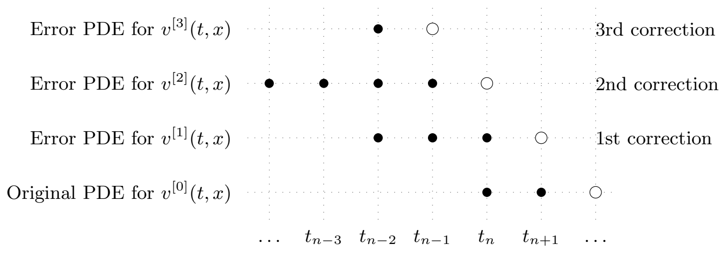

Error PDE for $v^{[3]}(t,x)$
Error PDE for $v^{[2]}(t,x)$
Error PDE for $v^{[1]}(t,x)$
Original PDE for $v^{[0]}(t,x)$
3rd correction
2nd correction
1st correction
$\ldots$ t_{n-3} t_{n-2} t_{n-1} t_n t_{n+1} $\ldots$

Fig. 1 The black dots represent the memory footprint that must be stored before the white dots can be computed in a pipe. In this figure, $v^{[0],n+2}(x)$, $v^{[1],n+1}(x)$, $v^{[2],n}(x)$ and $v^{[3],n-1}(x)$ are computed simultaneously.

2.2 RIDC–DD

The RIDC–DD algorithm solves the predictor (4) and corrections (5) using DD algorithms in space. The key observation is that (4) and (5) are **both** elliptic PDEs of the form $(1 - \Delta t\, \partial_{xx})z = f(x)$. The function $f(x)$ is known from the solution at the previous time step and previously computed lower order approximations. DD algorithms for solving elliptic PDEs are well known [3, 4]. The general idea is to replace the PDE by a coupled system of PDEs over some partitioning of the spatial domain using overlapping or non–overlapping subdomains. The coupling is provided by necessary transmission conditions at the subdomain boundaries. These transmission conditions are chosen to ensure the DD algorithm converges and to optimize the convergence rate. In [17], as a proof of principle, (4-5) are solved using a classical parallel Schwarz algorithm, with overlapping subdomains and Dirichlet transmission conditions. Optimized RIDC–DD variants are possible using an optimized Schwarz DD method [12, 11, 8], to solve (4-5). The solution from the previous time step can be used as initial subdomain solutions at the interfaces. We will use RIDCp–DD to refer to a pth order solution obtained using $p-1$ RIDC corrections in time and DD in space.

3 Implementation Details

We view the parallel time integrator reviewed in §2.1 as a simple yet powerful tool to add further scalability to a legacy MPI or modern MPI–CUDA code, while improving the accuracy of numerical solution. The RIDC integrators benefit from access to shared memory because solving the correction PDE (5) requires both the solution from the previous time step and previously computed lower order subdomain solution. Consequently, we propose two MPI-OpenMP hybrid implementations which map well to multi-core, multi-node compute resources. In the upcoming MPI 3.0 standard [1], shared memory access within the MPI library will provide alternative implementations.

Implementation #1: The RIDC-DD algorithm can be implemented using a traditional fork join approach, as illustrated in Program 1. After boundary information is exchanged, each MPI task spawns OpenMP threads to perform the linear solve. The threads are merged back together before MPI communication is used to check for convergence. The drawback to this fork-join implementation, is that the parallel space-time algorithm becomes tightly integrated, making it difficult to leverage an existing spatially parallel DD implementation.

```
MPI Initialization
...
  for each time step
     for each Schwarz iteration
        MPI Comm (exchange boundary info)
           OMP Parallel for each prediction/correction
              linear solve
           end parallel
        MPI Comm (check for convergence)
     end
  end
...
MPI Finalize
```

Program 1: RIDC-DD implementation using a fork-join approach.

Implementation #2: To leverage an existing spatially parallel DD implementation, a non-traditional hybrid approach must be considered. By changing the order of the loops, the Schwarz iterations for the prediction and the correction loops can be evaluated independently of each other. This is realized by spawning individual OpenMP threads to solve the prediction and correction loops on each subdomain; the Schwarz iterations for the prediction/correction step run independently of each other until convergence. This implementation (Program 2) has several consequences: (i) a thread safe version of MPI supporting `MPI_THREAD_MULTIPLE` is required. (ii) In addition, we required a thread-safe, thread-independent version of `MPI_BARRIER`, `MPI_BROADCAST` and `MPI_GATHER`. To achieve this, we wrote our own wrapper library using the thread safe `MPI_SEND`, `MPI_RECV` and `MPI_SENDRECV` provided by (i).

```
MPI Initialization
...
  for each time step
     OMP Parallel for each prediction/correction level
        for each Schwarz iteration
           MPI Comm (exchange boundary info)
           linear solve
           MPI Comm (check for convergence)
        end
     end parallel
  end
...
MPI Finalize
```

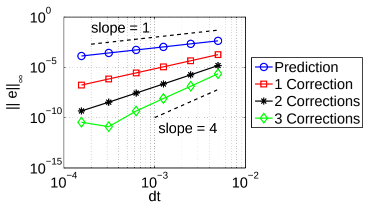
Program 2: RIDC-DD implementation using a non-traditional hybrid approach.

4 Numerical Experiments

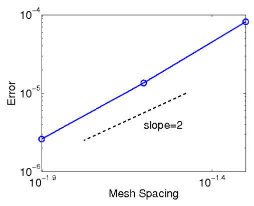
We show first that RIDC-DD methods converge with the designed orders in space and time. Then, we profile communication costs using TAU [13]. Finally, we show strong scaling studies for the RIDC-DD algorithm. We compute solutions to the heat equation in $\mathbb{R}^2$, where centered finite differences are used to approximate the second derivative operator. Errors are computed using the known analytic solution. The computations are performed on a Linux cluster at Michigan State University, where nodes (consisting of two quad core Intel Westmere processors) are interconnected using infiniband and a high speed Lustre file system.

4.1 Convergence Studies and Profile Analysis

In Figure 2, the convergence plots show that our classical Schwarz RIDC-DD algorithm converges as expected in space and time. The Schwarz iterations are iterated until a tolerance of 10^{-12} is reached for the predictors and correctors (which explains why the error in the fourth order approximation levels out as the time step becomes small).



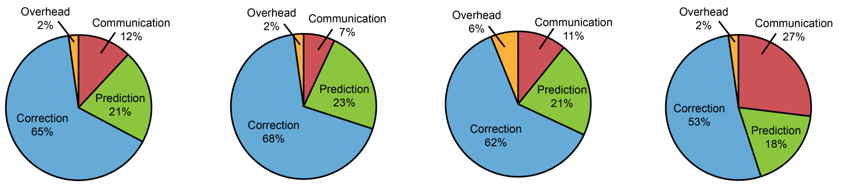
(a) Time Convergence



(b) Space Convergence

Fig. 2 (a) Classical Schwarz RIDCp-DD algorithms, $p = 1,2,3,4$, converge to the reference solution with the designed orders of accuracy. Here Δx is fixed while Δt is varied. (b) Second order convergence in space is demonstrated for the fourth order RIDC-DD algorithm. Here, Δt is fixed while Δx is varied.

The communication costs for our two implementations of RIDC4-DD are profiled using TAU [13]. We see in Figure 3, communication costs are minimal for implementation #1, and scales nicely as the number of nodes is increased, but the communication cost is significant for implementation #2. In Figure 3(a,c), the domain is discretized into 180×180 grid nodes, which are split into a 3×3 configuration of subdomains. In Figure 3(b,d), the domain is discretized into 360×360 grid nodes, which are split into a 6×6 configuration of subdomains. This keeps the number of grid points per subdomain constant so that the computation time for the matrix factorization and linear solve are the same.



(a) Implementation #1 3×3 domain (b) Implementation #1 6×6 domain (c) Implementation #2 3×3 domain (d) Implementation #2 6×6 domain

Fig. 3 Profile of the RIDC4-DD algorithm using both implementations. Overhead and communication costs are reasonable for implementation #1, but are high for implementation #2.

4.2 *Characterizing Parallel Performance*

A strong scaling study for the classical Schwarz RIDC4-DD algorithm is performed. Due to the better communication profile, framework #1 was used. We fix $\Delta x = \frac{1}{180}$, $\Delta y = \frac{1}{180}$, $\Delta t = \frac{1}{1000}$, and TOL=$10^{-12}$ (the Schwarz iteration tolerance). The overlap region is varied to keep the total number of Schwarz iterations constant. Table 1 summarizes the timing runs and observed speedup to a baseline solution. The column label, $N_x \times N_y \cdot N_t$, indicates parallelization using a $N_x \times N_y$ configuration of subdomains and N_t threads to integrate in time. For example, $6\times 6\cdot 4$ refers to four threads to compute a fourth order approximation in time, where the spatial domain is divided into a 6×6 configuration of sub-domains. A total of $6\cdot 6\cdot 4 = 144$ cores are used. The number of Schwarz iterations is reported, and the speedup is computed by the ratio of the runtimes (as compared with the $2\times 2\cdot 1$ simulation). Speedup is evident as the temporal or spatial parallelization is improved. The efficiency is computed by taking the ratio between the speedup and the additional number of cores used.

	$2\times 2\cdot 1$	$2\times 2\cdot 2$	$2\times 2\cdot 4$	$4\times 4\cdot 1$	$4\times 4\cdot 2$	$4\times 4\cdot 4$	$6\times 6\cdot 1$	$6\times 6\cdot 2$	$6\times 6\cdot 4$
# cores	4	8	16	16	32	64	36	72	144
walltime	1827	943	537	224	125	87	74	48	34
speedup	1.0	1.9	3.4	8.2	14.6	21.0	24.7	38.1	53.7
efficiency	1.00	0.97	0.85	2.04	1.83	1.31	2.74	2.11	1.49
# Schwarz	6295	6295	6295	6295	6295	6295	6295	6295	6295

Table 1 Strong scaling study for a fourth order (in time) RIDC-DD algorithm. The ratio of the overlap region to the size of the sub domain is held fixed.

The speedup from the spatial parallelization is unusually high. Further reflection reveals that one should not expect a linear decrease in computation time as the number of subdomains is increased. This is because the number of unknowns (grid points), m in each subdomain, decreases linearly with the number of subdomains, but, the cost for the linear solve scales as $\mathcal{O}(m^2)$ in our implementation.

5 Conclusions

This paper has presented the implementation details and first reported profiling results for a newly proposed space–time parallel algorithm for time dependent PDEs. The RIDC–DD method combines traditional domain decomposition in space with a new family of deferred correction methods designed to allow parallelism in time. Two possible implementations are described and profiled. The first, a traditional hybrid OpenMP–MPI implementation, requires potentially difficult modifications of an existing parallel spatial solver. Numerical experiments verify that the algorithm achieves its designed order of accuracy and scales well. The second strategy allows a relatively easy reuse of an existing parallel spatial solver by using OpenMP to spawn threads for the simultaneous prediction and correction steps. This non–traditional hybrid use of OpenMP and MPI currently requires writing of custom thread–safe and thread–independent MPI routines. Profile analysis shows that our non-traditional use of OpenMP–MPI suffers from higher communication costs than the standard use of OpenMP-MPI. An inspection of the prediction and correction equations indicates that optimized Schwarz variants of the algorithm are possible and will enjoy nice load balancing. This work is ongoing.

Acknowledgements This work was supported by the Institute for Cyber-Enabled Research (iCER) at MSU, NSERC Discovery Grant 311796, and AFOSR Grant FA9550-12-1-0455.

References

1. Mpi 3.0 standardization effort. `http://meetings.mpi-forum.org/MPI_3.0_main_page.php`. Accessed 10/25/2012
2. Böhmer, K., Stetter, H.: Defect correction methods. theory and applications (1984)
3. Cai, X.C.: Additive Schwarz algorithms for parabolic convection-diffusion equations. Numer. Math. **60**(1), 41–61 (1991)
4. Cai, X.C.: Multiplicative Schwarz methods for parabolic problems. SIAM J. Sci. Comput. **15**(3), 587–603 (1994)
5. Christlieb, A., Macdonald, C., Ong, B.: Parallel high-order integrators. SIAM J. Sci. Comput. **32**(2), 818–835 (2010)
6. Christlieb, A., Ong, B.: Implicit parallel time integrators. J. Sci. Comput. **49**(2), 167–179 (2011)
7. Christlieb, A., Ong, B., Qiu, J.M.: Comments on high order integrators embedded within integral deferred correction methods. Comm. Appl. Math. Comput. Sci. **4**(1), 27–56 (2009)
8. Dubois, O., Gander, M., Loisel, S., St-Cyr, A., Szyld, D.: The optimized Schwarz method with a coarse grid correction. SIAM J. Sci. Comput. **34**(1), A421–A458 (2012)
9. Dutt, A., Greengard, L., Rokhlin, V.: Spectral deferred correction methods for ordinary differential equations. BIT **40**(2), 241–266 (2000)
10. Gander, M., Vandewalle, S.: On the superlinear and linear convergence of the parareal algorithm. Lecture Notes in Computational Science and Engineering **55**, 291 (2007)
11. Gander, M.J.: Optimized Schwarz methods. SIAM J. Numer. Anal. **44**(2), 699–731 (2006)
12. Gander, M.J., Halpern, L.: Optimized Schwarz waveform relaxation methods for advection reaction diffusion problems. SIAM J. Numer. Anal. **45**(2), 666–697 (2007)
13. Koehler, S., Curreri, J., George, A.: Performance analysis challenges and framework for high-performance reconfigurable computing. Parallel Computing **34**(4), 217–230 (2008)
14. Lions, J., Maday, Y., Turinici, G.: A "parareal" in time discretization of PDEs. Comptes Rendus de l'Academie des Sciences Series I Mathematics **332**(7), 661–668 (2001)
15. Minion, M., Williams, S.: Parareal and spectral deferred corrections. In: NUMERICAL ANALYSIS AND APPLIED MATHEMATICS: International Conference on Numerical Analysis and Applied Mathematics 2008. AIP Conference Proceedings, vol. 1048, pp. 388–391 (2008)
16. Nievergelt, J.: Parallel methods for integrating ordinary differential equations. Communications of the ACM **7**(12), 731–733 (1964)
17. Ong, B., Haynes, R., Christlieb, A.: A parallel space–time algorithm. SIAM J. Sci. Comput. (2012). In Press
18. Toselli, A., Widlund, O.: Domain decomposition methods—algorithms and theory, *Springer Series in Computational Mathematics*, vol. 34. Springer-Verlag, Berlin (2005)