Implicit Parallel Time Integrators

Andrew Christlieb   Benjamin Ong [*]

October 25, 2010

Abstract

In this work, we discuss a family of parallel implicit time integrators for multi-core and potentially multi-node or multi-gpgpu systems. The method is an extension of Revisionist Integral Deferred Correction (RIDC) by Christlieb, Macdonald and Ong (SISC-2010) which constructed parallel explicit time integrators. The key idea is to re-write the defect correction framework so that, after initial startup costs, each correction loop can be lagged behind the previous correction loop in a manner that facilitates running the predictor and correctors in parallel.

In this paper, we show that RIDC provides a framework to use p cores to generate a pth-order implicit solution to an initial value problem (IVP) in approximately the same wall clock time as a single core, backward Euler implementation ($p \leq 12$). The construction, convergence and stability of the schemes are presented, along with supporting numerical evidence.

Keywords: Initial value problems, integral deferred correction, parallel computation, multi-core computing.

[*]Department of Math, Michigan State University, East Lansing, MI 48823

1 Introduction

In this paper, we construct and analyze a class of parallel implicit time integrators for initial value problems (IVP), known as Revisionist Integral Deferred Correction methods (RIDC), which can be efficiently implemented with multi-core and/or multi gpgpu architectures. The "revisionist" terminology was adopted to highlight that (1) this is a *revision* of the standard integral defect correction (IDC) formulation, and (2) successive corrections, running in parallel but lagging in time, *revise* and improve the approximation to the solution.

There are several avenues where many cores can be utilized for implicit time integration. A transparent area where a parallel implementation can be employed is in the Newton solve, where the Jacobian approximation and the linear solve can both be performed in parallel. There are several libraries such as PETSC [2], which utilizes the standard message passing interface, and ViennaCL [25], which performs computations on a gpgpu card, that can be utilized to speedup a code. Using a parallel framework for the Newton solve typically produces a significant speedup in the computation, however, hard scaling limits may arise for a fixed problem size. Another area where parallel algorithms are sometimes employed is domain decomposition. Consider the initial value problem

$$\begin{cases} y'(t) = f(t,y), \quad t \in [a,b], \\ y(a) = \alpha. \end{cases} \tag{1}$$

where $y \in \mathbb{R}^n$, $f : \mathbb{R} \times \mathbb{R}^n \to \mathbb{R}^n$. Domain decomposition splits the IVP into a series of N sub problems, $z_i'(t) = g_i(t,y)$, where $z_i \in \mathbb{R}^{n/N}$ and $g : \mathbb{R} \times \mathbb{R}^n \to \mathbb{R}^{n/N}$, and each core works on a separate subproblem [3, 14, 27]. In general, domain decomposition suffers from communication overhead, as well as scaling issues for a fixed problem size. A third area where a parallel framework can be employed is in the evaluation of $f(t,y)$, which may contain integral solutions that can be evaluated efficiently in parallel using fast multipole or treecode algorithms [11, 28].

A fourth area of parallelization, which is the focus of this paper, is the actual parallelization in the temporal direction. Parallelization in this area is particularly interesting because it can be combined with the previously discussed parallelization ideas. For example, the present authors are exploring (i) the use of p computing nodes, each containing a general purpose graphics processing unit (gpgpu) to generate a pth-order solution in approximately the same wall clock time as a single computing node with a single gpgpu card, and (ii) space-time parallelization of PDEs.

A natural idea to parallelize in the temporal direction is to design high order Runge–Kutta schemes whose stages can be evaluated in parallel [15]. Designing such schemes however involves tedious order conditions. The approach we take involves re-writing the defect correction framework so that, after initial startup costs, each correction loop can be lagged behind the previous correction loop in a manner that facilitates running the predictor and correctors in parallel. This idea for explicit parallel time integrators was previously published by the present authors in [4], and falls in the category of *parallel across the steps*. Implicit RIDC-BE integrators (RIDC algorithms constructed using backward Euler integrators) are particularly promising because of their stability and efficiency. In each step of a pth order RIDC-BE implementation with p computing nodes, each computing node performs a Newton solve on a system of N equations. This contrasts with a serial s-stage implicit RK integrator where a system of sN equations needs to be solved. In practice, p has to be ≤ 12 due to the Runge phenomenon [26].

For completeness, we mention that another approach to parallelization in the temporal direction is *parallel across the method*, e.g. the recent parareal algorithm [8, 9, 21, 22, 10]. In such methods, the time domain is split into sub-problems that can be computed in parallel, and an iterative procedure for coupling the sub-problems is applied, so that the overall method

converges to the solution of the full problem. Recently, the parareal algorithm has been coupled with the defect correction framework [23] to form a hybrid parareal-defect correction time integrator.

This paper is divided into five main sections. In Section 2, RIDC methods and their construction are presented. Convergence theorems and the stability of RIDC methods are discussed Section 3. Then, numerical benchmarks comparing RIDC and some implicit Runge–Kutta methods are given in Section 4, followed by concluding remarks in Section 5.

2 RIDC Methods

RIDC methods are a class of time integrators based on integral deferred correction [1, 20, 19, 16, 17, 7, 6, 24, 18, 12]. RIDC methods first compute a prediction to the solution ("level 0") using low order schemes (e.g. backward Euler) followed by one or more corrections to compute subsequent solution levels. Each correction *revises* the solution and increases the formal order of accuracy by p, the order of the integrator used to solve the error equation [4].

Parallel speedup is obtained by simultaneously computing provisional and corrected solutions. However, this can only be achieved if the solution to each correction level is delayed from the previous level, as illustrated in Figure 1 for a backward Euler predictor and corrector, and in Figure 2 for the RK2 Trapezoid predictor and corrector. For now, just observe that the white circles, denoting solution values that are simultaneously computed, are staggered. The staggering is necessary because one requires a provisional solution to update. The stencil (black filled circles) needed for the computation will be discussed in Section 2.3.

In Section 2.1, we first derive the error equation. Then, Section 2.2 and Section 2.3 give numerical schemes for solving the IVP and the error equation. In Section 2.4, details for start-

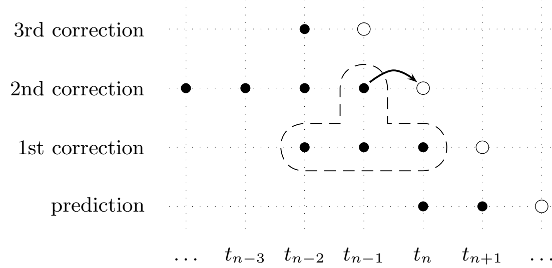


Figure 1: (RIDC4-BE) This plot shows the staggering required for a fourth order RIDC scheme, constructed using backward Euler predictors and correctors. The time axis runs horizontally, and the correction levels run vertically. The white circles denote solution values that are simultaneously computed, e.g., core 0 is computing the prediction solution at time t_{n+2}, core 1 is computing the 1st correction solution at time t_{n+1}, etc.

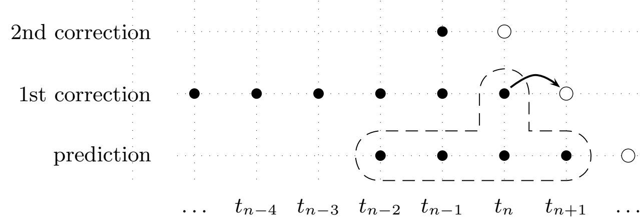


Figure 2: (RIDC6-RK2) This plot shows the staggering required for a sixth order RIDC scheme, constructed using second order trapezoid predictors and correctors. The time axis runs horizontally, and the correction levels run vertically. The white circles denote solution values that are simultaneously computed, e.g., core 0 is computing the prediction solution at time t_{n+2}, core 1 is computing the 1st correction solution at time t_{n+1}, etc.

ing and stopping the RIDC algorithm, as well as the theoretical speedup is discussed. We comment on the notion of resets and other observations in Section 2.5.

2.1 Error Equation

Suppose an approximate solution $\eta(t)$ to the exact solution $y(t)$ is computed. Then, the error of the approximate solution is

$$e(t) = y(t) - \eta(t). \tag{2}$$

If we define the residual as $\epsilon(t) = \eta'(t) - f(t, \eta(t))$, then the derivative of the error (2) satisfies

$$e'(t) = y'(t) - \eta'(t) = f(t, y(t)) - f(t, \eta(t)) - \epsilon(t).$$

The integral form of the error equation

$$\left[e(t) + \int_a^t \epsilon(\tau)\, d\tau \right]' = f(t, \eta(t) + e(t)) - f(t, \eta(t)), \tag{3}$$

can then be solved using the initial condition $e(a) = 0$.

2.2 The predictor

The predictor is a low order integrator that is applied to solve IVP (1). For example, a backward Euler integrator advances the solution from t_n to t_{n+1} using

$$\eta_{n+1}^{[0]} = \eta_n^{[0]} + \Delta t_n f(t_{n+1}, \eta_{n+1}^{[0]}), \tag{4}$$

where the superscript $^{[0]}$ indicates this is the solution at level 0, the prediction level. This non-linear equation can be solved using Newton's method.

For a more general s stage RK method with stage weights a_{ij}, node weights b_j and node points c_j, the solution is advanced from t_n to t_{n+1} using

$$\eta_{n+1}^{[0]} = \eta_n^{[0]} + \sum_{j=0}^{s} b_j K_j,$$

where the stages K_i are found by solving the nonlinear equations generated by

$$K_i = \Delta t_n f\left(t_n + c_i \Delta t_n, \eta_n^{[0]} + \sum_{j=0}^{s} a_{ij} K_j\right), \quad i = 1, \ldots, s.$$

2.3 The corrector

The correctors are also low order integrators, but are used to solve the error equation (3) for the error $e(t)$ to an approximate solution $\eta(t)$. Since the error equation is solved iteratively to improve a solution from the previous level, each correction level computes an error $e^{[j-1]}(t)$ to the solution at the previous level $\eta^{[j-1]}(t)$ to obtain a revised solution $\eta^{[j]}(t) = \eta^{[j-1]}(t) + e^{[j-1]}(t)$.

A backward Euler discretization of equation (3) (after some algebra) gives

$$\eta_{n+1}^{[j]} = \begin{cases} \eta_n^{[j]} + \Delta t_n \left(f_{n+1}^{[j]} - f_{n+1}^{[j-1]}\right) + \sum_{k=0}^{j} \alpha_k f_{n+1-k}^{[j-1]}, & \text{if } n \geq j-1 \\ \eta_n^{[j]} + \Delta t_n \left(f_{n+1}^{[j]} - f_{n+1}^{[j-1]}\right) + \sum_{k=0}^{j} \alpha_k f_{k}^{[j-1]}, & \text{if } n < j-1 \end{cases} \tag{5}$$

where $f_n^{[1]}$ denotes $f(t_n, \eta_n^{[1]})$ in the above nonlinear equations. α_k are quadrature weights so that the sum approximates the integral $\int_{t_n}^{t_{n+1}} f(\tau, \eta^{[j-1]}(\tau))\, d\tau$. For example, if $j = 2$ and $n > j-1$, the function values $f_{n+1}^{[1]}, f_n^{[1]}$ and $f_{n-1}^{[1]}$ are used for the quadrature and

$$\alpha_k = \int_{t_n}^{t_{n+1}} \prod_{i=0, i\neq k}^{j=2} \frac{(t - t_{n+1-i})}{(t_{n+1-k} - t_{n+1-i})}\, dt, \quad k = 0, 1, 2$$

Note that the number of terms in the sum increases with level j because the order of method after the jth corrector is $j+1$ and thus the integral must be approximated with increasing accuracy. For example, we show the stencil required to compute

a sufficiently accurate quadrature approximation for the second correction loop for RIDC4-BE in Figure 1 using dashed lines. Three values are needed from the previous value to approximate the quadrature of the residual. Note that the choice of stencils picked for the quadrature is not unique; in practice, the different quadrature stencils do not seem to significantly affect the accuracy of the solution.

If a more general pth order s-stage RK method with stage weights a_{ij}, node weights b_j and node points c_j, is used to compute $\eta^{[j]}_{n+1}$, we first note from [5] that the order increase is only guaranteed if the quadrature nodes are spaced equally with spacing Δt. Then,

$$\begin{aligned}
\eta^{[j]}_{n+1} &= \eta^{[j]}_n + \sum_{j=0}^{s} b_j K_j, + \int_{t_n}^{t_{n+1}} f(\tau, \eta^{[j-1]}(\tau))\, d\tau \\
&= \begin{cases} \eta^{[j]}_n + \sum_{j=0}^{s} b_j K_j, + \sum_{k=0}^{r_j} \alpha_k f^{[j-1]}_{n+1-k}, & \text{if } n \geq r_j - 1 \\ \eta^{[j]}_n + \sum_{j=0}^{s} b_j K_j, + \sum_{k=0}^{r_j} \alpha_k f^{[j-1]}_{k}, & \text{if } n < r_j - 1 \end{cases}
\end{aligned}$$

where r_j denotes the anticipated order of the method after the jth correction. The stages K_i are found by solving the nonlinear equations generated by

$$\begin{aligned}
K_i = \Delta t f \left(\gamma_i, \eta^{[j-1]}_n - \eta^{[j-1]}(\gamma_i) + \sum_{j=0}^{s} a_{ij} K_j + \int_{t_n}^{\gamma_i} f^{[j-1]}(\tau, \eta^{[j-1]}(\tau))\, d\tau \right) \\
- \Delta t f \left(\gamma_i, \eta^{[j-1]}(\gamma_i) \right)
\end{aligned}$$

where $\gamma_i = t_n + c_i \Delta t$. The integral is once again approximated by quadrature with the appropriate stencil. This same stencil is used to approximate the solution value $\eta^{[j-1]}(t_n + c_i \Delta t)$ by interpolating the computed function values at the previous level.

2.4 Startup, Shutdown and Speedup

During most of a RIDC calculation, multiple solution levels are computed in parallel using multiple computing nodes. Here, a computing node refers generically to (i) a single computing core in a multi-core system, (ii) a single computing core coupled with a gpgpu card or (iii) a cluster of computing nodes if domain decomposition is being used.

However, the computing nodes in the RIDC algorithm cannot start simultaneously: each must wait for the previous level to compute sufficient η values before marching all of them in a pipeline fashion. The order in which computations can be performed using a four node system during startup is illustrated in Figure 3 for a fourth order RIDC constructed with backward Euler integrators. The jth processor (running the jth correction) must initially wait for $j(j+1)/2$ steps. For example, in order for computing node 1 to compute $\eta_1^{[1]}$, it requires $\eta_0^{[0]}$ and $\eta_1^{[0]}$ to compute a sufficiently accurate quadrature approximation. Consequently, it cannot start in "step 1" (as denoted in the figure) since it has to wait for computing node 0 to complete computing $\eta_1^{[0]}$. Similarly, the starting algorithm for RIDC6-RK2 is illustrated in Figure 4. For RIDC6-RK2, the jth processor (running the jth correction) must initially wait for $(j+1)^2 - 1$ steps.

In terms of shutdown, the calculation ends when the highest level (most accurate) computation reaches the final time $t_N = b$. Note that the predictor and lower level correctors will reach $t_N = b$ earlier and as a consequence some computing threads will sit idle.

Estimating the potential speedup and parallel efficiency for our parallel implicit RIDC algorithms cannot be easily quantified, because advancing the solution from t_n to t_{n+1} on different levels might involve a different number of Newton iterations. Additionally, not all processors are made equal and latency, shared memory overhead, and interprocess communi-

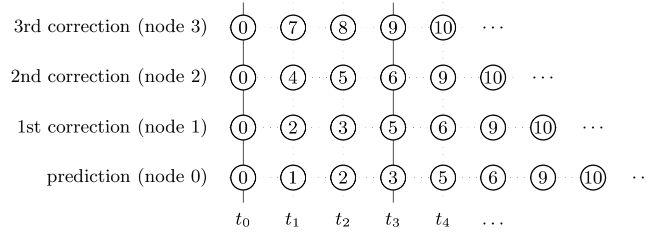


Figure 3: Starting the RIDC4-BE algorithm. Each number indicates the step in which the corresponding value of $f(\eta)$ is computed. In the starting algorithm, special care is taken to ensure that minimum memory is used by not letting the computing nodes run ahead until they can be marched in a pipeline; in this example, when node 3 starts computing t_3.

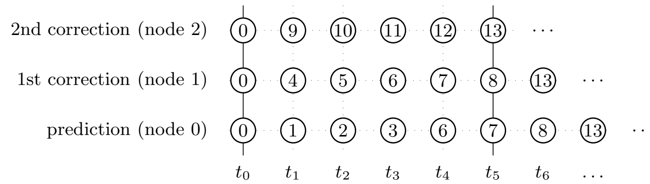


Figure 4: Starting the RIDC6-RK2 algorithm. Here, the computing nodes can be marched in a pipeline when node 2 starts computing t_5.

cation issues will decrease the speedup in practice. Under the assumption that advancing the solution from t_n to t_{n+1} involves the same number of Newton iterations, a rough estimate for speedup and efficiency for RIDC algorithms constructed with backward Euler is,

$$\text{speedup} = \frac{pN}{N + p(p+1)/2}, \quad \text{efficiency} = \frac{N}{N + p(p+1)/2},$$

where p is the order of the RIDC algorithm, and the number of computing nodes available. As $N \to \infty$, the speedup $\to p$, and the efficiency $\to 1$. For RIDC algorithms constructed with an implicit RK2 scheme,

$$\text{speedup} = \frac{pN}{N + (p+1)^2 - 1}, \quad \text{efficiency} = \frac{N}{N + (p+1)^2 - 1},$$

where the order of the RIDC algorithm is $2p$, and p computing nodes are available.

2.5 Resets and Other Comments

A RIDC computation can be periodically "reset". That is, instead of computing all the way to the final time, we compute on some smaller time interval to time $t_\star$, and use the most accurate solution to reset the RIDC computation at $t = t_\star$. In practice, this should lower the error constant of the overall method, but at the cost of decreasing the speedup due to the additional cost of starting the RIDC algorithm multiple times.

Finally, we comment that we cannot increase the order of RIDC indefinitely as (i) it is not practical and (ii) the Runge phenomenon [26], which arises from using equi-spaced interpolation points, will eventually cause the scheme to become unstable. In practice, $12th$ order RIDC methods have been constructed without any observable instability.

3 Convergence and Stability

We first discuss convergence theorems for the implicit RIDC methods, followed by the linear stability regions of various implicit RIDC methods. The effect of resets on the stability regions are explored.

3.1 Convergence of RIDC methods

The analysis in [7, 5], proving convergence under mild conditions for IDC methods, extends simply to these RIDC methods.

Theorem 3.1. Let $f(t, y)$ and $y(t)$ in IVP (1) be sufficiently smooth. Then, the local truncation error for a RIDC method constructed using uniform time steps and backward Euler integrators as the predictor and $(p - 1)$ Euler correction loops is $\mathcal{O}(h^{p+1})$.

Theorem 3.2. Let $f(t, y)$ and $y(t)$ in IVP (1) be sufficiently smooth. Then, the local truncation error for an RIDC method constructed using uniform time steps, a ${p_0}^{\text{th}}$-order Runge–Kutta method in the prediction loop, and $(p_1, p_2, \cdots, p_j)^{\text{th}}$-order RK methods in the correction loops, is $\mathcal{O}(h^{p+1})$, where $p = \sum_{i=0}^{j} p_i$.

The proof for both these theorems extend simply from the analysis in [5, 7].

3.2 Regions of Absolute Stability

We stick to the normal definition of absolute stability, as order stars are difficult to generate for RIDC schemes for $N \gg 1$. We instead use the following simpler definitions.

Definition 3.1. The amplification factor for a numerical method, $\phi(\lambda\Delta t)$, can be interpreted as the numerical solution to Dalhquist's test equation,

$$y'(t) = \lambda y(t), \quad y(0) = 1, \tag{6}$$

at $t = 1$. If the numerical method uses a stepsize of Δt, then $\phi(\lambda\Delta t) = y(1)$, for $\lambda \in \mathbb{C}$,

Definition 3.2. The stability region, S, for a numerical method, is the subset of the complex plane $\mathbb{C}$, consisting of all $\lambda\Delta t$ such that $|\phi(\lambda\Delta t)| \leq 1$,

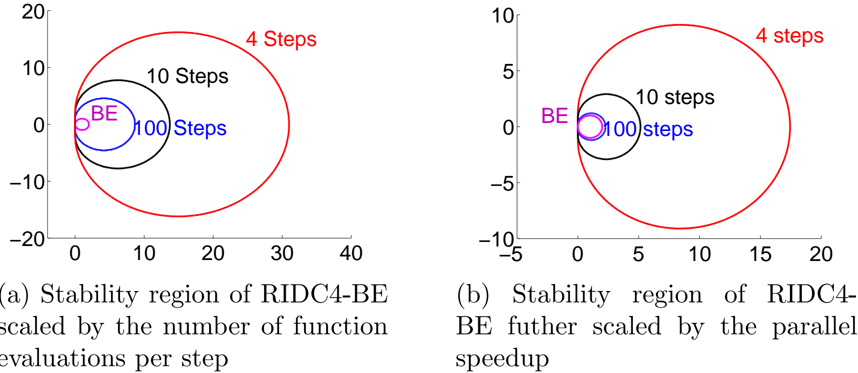
$$S = \{\lambda\Delta t : |\phi(\lambda\Delta t)| \leq 1\} .$$

We introduce the following definition for "scaled" stability regions for parallel methods, and note that the stability regions for RIDC methods in [4] should have been appropriately scaled.

Definition 3.3. The *scaled* stability region for a serial time integrator, S_s, is the stability region S defined above scaled by the number of function evaluations per Δt step. The scaled stability region for a *parallel time integrator*, S_p, is the stability region S defined above scaled by the number of function evaluations per Δt step, and further scaled by the parallel speedup.

In Figure 5, we plot the stability regions for RIDC4-BE, varying the number of steps taken before a reset. In Figure 5(a), we have scaled by the stability region by the number of function evaluations per time step, and in Figure 5(b), we have further scaled by the parallel speedup. (One can, and should interpret Figure 5(a) as the stability region for RIDC4-BE when only one computing node is used for the computation, and Figure 5(b) as the stability region of RIDC4-BE when four computing nodes are used.) Note that the axis scales are different in both plots for annotation purposes. The two main observations are (i) RIDC4-BE maintains A-stability regardless when the reset is performed, and (ii) the scaled stability region for RIDC4-BE when four nodes are used, approaches that of backward Euler as the number of steps before a reset increases.

Figure 6 shows the scaled stability regions of RIDC4-BE, RIDC4-RK2, RIDC8-RK2 after 100 steps. We observe that although the base trapezoidal RK2 scheme is A-stable, the RIDC methods constructed using that RK2 scheme loses its A-stability property.



(a) Stability region of RIDC4-BE scaled by the number of function evaluations per step

(b) Stability region of RIDC4-BE futher scaled by the parallel speedup

Figure 5: This figure shows the region of absolute stability for the backward Euler integrator (magenta), and the scaled regions of absolute stability for the RIDC4-BE algorithm after 4 steps (red), 10 steps (black), and 100 steps (blue). The two main observations are (i) RIDC4-BE maintains A-stability regardless when the reset is performed, and (ii) the stability region approaches that of backward Euler as the number of steps before a reset increases. The regions of stability are the regions outside of the enclosed curves.

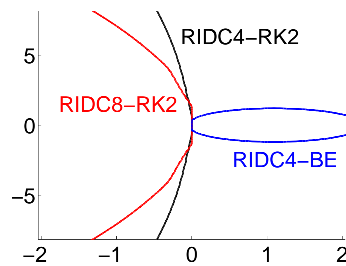


Figure 6: Scaled regions of absolute stability for various parallel RIDC schemes. For RIDC methods constructed using the trapezoidal RK2 integrator, the stability regions lie to the left of the curve. For RIDC4-BE, the region of absolute stability is the region outside the curve.

4 Numerical Examples

We present some numerical results validating the order of accuracy of the RIDC schemes, and the speedup obtained in this parallel framework. The computations were performed on a desktop containing two quad core Intel Xeon 3.0Ghz processors. The RIDC algorithms were implemented in C++ using the OpenMP protocol.

4.1 Advection-Diffusion

We apply various RIDC-BE algorithms to the advection-diffusion problem to show that RIDC methods are able to achieved designed orders of accuracy, and to study how the error changes with the number of resets. The advection-diffusion equation is

$$\begin{aligned} &u_t = u_x + D u_{xx}, \quad x \in [0,1], \quad t \in [0,1], \\ &u(x,0) = 2 + \sin(2\pi x), \end{aligned}$$

with periodic boundary conditions. The diffusion coefficient is set at $D = 10^{-2}$. The problem is discretized into 128 spatial intervals and solved in Fourier space; a system of odes are obtained for each discrete wave number.

$$(\hat{u}_k)_t = (ik)\hat{u}_k - k^2 D \hat{u}_k, \quad k = -\frac{64}{2\pi}, \ldots, \frac{63}{2\pi}.$$

The convergence studies are summarized in Table 1, which show that RIDC methods are able to achieve their designed orders of accuracy.

We also study how the number of resets affect the overall error for this advection-diffusion problem, as discussed in Section 2.5. A total of 400 steps are taken. In Figure 7, the error at the final time $t = 1$ is plotted for varying number of resets. For example, if three resets are taken, then the most accurate solution after every 100 steps is used to restart the RIDC algorithm. Note that 99 resets (i.e. a reset every four steps in this

–	RIDC2-BE		RIDC3-BE		RIDC4-BE	
N	error	order	error	order	error	order
80	1.75e-02	–	1.53e-03	–	1.27e-04	–
160	4.78e-03	1.88	2.14e-04	2.85	9.01e-06	3.82
240	2.19e-03	1.93	6.56e-05	2.91	1.85e-06	3.90
320	1.25e-03	1.95	2.82e-05	2.94	5.98e-07	3.93
400	8.06e-04	1.96	1.46e-05	2.95	2.48e-07	3.95

–	RIDC5-BE		RIDC6-BE		RIDC7-BE	
N	error	order	error	order	error	order
40	2.41e-04	–	3.34e-05	–	4.59e-06	–
80	9.88e-06	4.61	7.02e-07	5.57	4.95e-08	6.53
120	1.42e-06	4.78	6.79e-08	5.76	3.21e-09	6.75
160	3.52e-07	4.85	1.27e-08	5.83	4.55e-10	6.79
200	1.18e-07	4.89	3.45e-09	5.84	1.15e-10	6.17

–	RIDC4-RK2		RIDC6-RK2		RIDC8-RK2	
N	error	order	error	order	error	order
10	1.87e-02	–	1.05e-03	–	4.45e-04	–
20	1.20e-03	3.96	2.90e-05	5.17	3.00e-06	7.22
40	7.23e-05	4.05	5.23e-07	5.79	1.10e-08	8.09
80	4.38e-06	4.05	8.62e-09	5.92	3.07e-11	8.49

Table 1: Convergence study for various RIDC schemes applied to solve the advection-diffusion problem. In all cases, RIDC schemes achieve their designed orders of accuracy.

experiment) corresponds to the standard IDC algorithm without parallelism. It is observed that the error decreases as the number of restarts is increased, but not significantly.

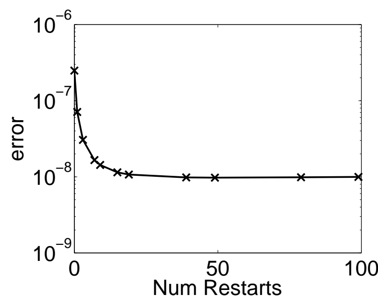


Figure 7: In the advection-diffusion example, we explore how the number of resets affect the error for the RIDC4-BE algorithm. As more resets are taken, the error decreases. Comparing with results in [4], the optimal number of restarts (balancing the efficiency with minimizing the error) appears to be problem dependent.

4.2 Brusselator

We consider an idealized autocatalytic reaction described by the Brusselator equations [13],

$$\frac{\partial u}{\partial t} = A + u^2 v - (B+1)u + \alpha \frac{\partial^2 u}{\partial x^2}$$
$$\frac{\partial v}{\partial t} = Bu - u^2 v + \alpha \frac{\partial^2 v}{\partial x^2},$$

where $A = 1$ and $B = 3$ are rate constants, and $\alpha = \frac{1}{50}$ is the diffusion constant. This system is nonlinear, and stiff due to the diffusion. A method of lines discretization is applied by using a centered differencing on the diffusion term. We set $x_i = i\Delta x, i = 0, 1, \ldots M$, where $\Delta x = \frac{1}{M} = \frac{1}{200}$, and obtain the

system of equations

$$\frac{\partial u_i}{\partial t} = A + u_i^2 v_i - (B+1)u_i + \alpha\left(\frac{u_{i+1} - 2u_i + u_{i-1}}{\Delta x^2}\right),$$
$$\frac{\partial v_i}{\partial t} = Bu_i - u_i^2 v_i + \alpha\left(\frac{v_{i+1} - 2v_i + v_{i-1}}{\Delta x^2}\right), \quad i = 1, \dots M-1$$

with boundary conditions

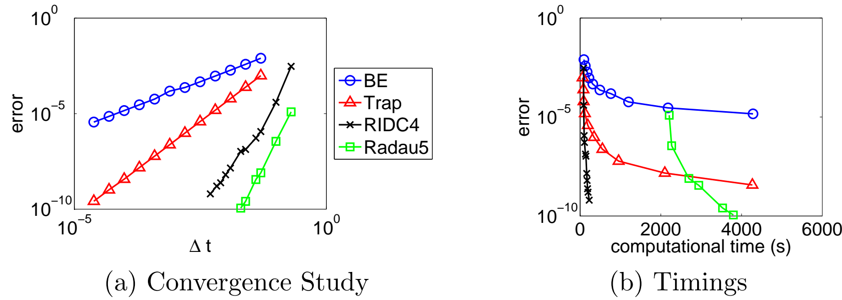
$$u(0,t) = u(1,t) = 1, \quad v(0,t) = v(1,t) = 3.$$

The following initial conditions were chosen:

$$u(x,0) = 1 + \sin(2\pi x), \quad v(x,0) = 3.$$

Newton's method is applied to solve the non-linear system of equations, and a GMRES algorithm is used to solve for each Newton update within Newton's method. In all the numerical runs presented below, a tolerance of 10^{-14} is used to find the Newton update using the GMRES algorithm, and a tolerance of 10^{-10} is used as a stopping criterion for Newton's method. For the non-linear solve, the Jacobian is approximated numerically using a first order finite difference formula.

In Figure 8(a), the error versus the stepsize for various schemes are plotted. All schemes achieve their designed rate of convergence. The error of the solution obtained using the three stage Radau5 scheme is smaller than that of the other schemes. One might speculate that the serial Radau5 scheme might be the most efficient scheme since each time update requires a Newton solve of a $3N$ system of equations (proportional to the number of stages), whereas a RIDC4-BE scheme requires four Newton solves of N system of equations for each update (after start up). We show in Figure 8(b) however that our parallel RIDC4-BE using four computing nodes is more efficient for this problem. In Figure 9, we show the speedup obtained by comparing timing runs for sequential RIDC integrators, and RIDC integrators run with multiple cores.



(a) Convergence Study (b) Timings

Figure 8: (a) shows a convergence study of various numerical schemes. All schemes achieve their designed rate of convergence. In (b), the errors are plotted as a function of computational time. RIDC4-BE run with four computing cares appear to be the most efficient.

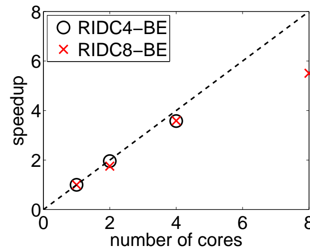


Figure 9: Speedup of RIDC4-BE (black circles) and RIDC8-BE methods are computed by comparing wall clock computation times of multi-core implementations with a single-core (sequential) implementation .

5 Conclusions

In this paper, we introduced implicit RIDC algorithms and implemented these algorithms on multi-core systems. We show that these algorithms attain their designed orders of accuracy for various problems, and show that significant speedup is obtained over serial algorithms. Stability and convergence of these methods were also discussed. In continuing work, the authors are applying RIDC algorithms to practical problems of interest, where time and spatial parallelization is desired, and are exploring the use of multi-core, multi-gpu nodes for use in RIDC algorithms.

Acknowledgments

This work was supported by AFRL and AFOSR under contract and grants FA9550-07-0092 and FA9550-07-0144, and the High Performance Computing Center (HPCC) at Michigan State University. The authors would also like to thank C.B. Macdonald and R.J. Spiteri for insightful comments and enlightening discussions related to this work.

References

[1] W. Auzinger, H. Hofstätter, W. Kreuzer, and E. Weinmüller. Modified defect correction algorithms for ODEs part I: General theory. *Numer. Algorithms*, 36:135–156, 2004.

[2] S. Balay, K. Buschelman, D. Gropp, W.D.and Kaushik, M.G. Knepley, L.C. McInnes, B.F. Smith, and Zhang H. PETSc Web page, 2009. http://www.mcs.anl.gov/petsc.

[3] E. Brakkee, A. Segal, and CGM Kassels. A parallel domain decomposition algorithm for the incompressible Navier-

Stokes equations. *Simulation Practice and Theory*, 3(4-5):185–205, 1995.

[4] Andrew Christlieb, Colin Macdonald, and Benjamin Ong. Parallel high-order integrators. *SIAM J. Sci. Comput.*, 32(2):818–835, 2010.

[5] Andrew Christlieb, Maureen Morton, Benjamin Ong, and Jing-Mei Qiu. Semi-implicit integral deferred correction constructed with high order additive Runge-Kutta integrators. submitted.

[6] Andrew Christlieb, Benjamin Ong, and Jing-Mei Qiu. Comments on high order integrators embedded within integral deferred correction methods. *Comm. Appl. Math. Comput. Sci.*, 4(1):27–56, 2009.

[7] Andrew Christlieb, Benjamin Ong, and Jing-Mei Qiu. Integral deferred correction methods constructed with high order Runge-Kutta integrators. *Math. Comput.*, 79:761–783, 2010.

[8] M.J. Gander and E. Hairer. Nonlinear convergence analysis for the parareal algorithm. *Lecture Notes in Computational Science and Engineering*, 60:45, 2008.

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

[10] M.J. Gander and S. Vandewalle. On the superlinear and linear convergence of the parareal algorithm. *Lecture Notes in Computational Science and Engineering*, 55:291, 2007.

[11] L. Greengard and W. D. Gropp. A parallel version of the fast multipole method. *Computers and Mathematics with Applications*, 20(7):63 – 71, 1990.

[12] Thomas Hagstrom and Ruhai Zhou. On the spectral deferred correction of splitting methods for initial value problems. *Commun. Appl. Math. Comput. Sci.*, 1:169–205, 2006.

[13] E. Hairer and G. Wanner. *Solving ordinary differential equations. II*, volume 14 of *Springer Series in Computational Mathematics*. Springer-Verlag, Berlin, second edition, 1996. Stiff and differential-algebraic problems.

[14] G. Horton and S. Vandewalle. A space-time multigrid method for parabolic PDEs. *Siam J. Sci. Comput*, 16(4):848–864, 1995.

[15] P. J. Van Der Houwen, B. P. Sommeijer, and W. Couzy. Embedded diagonally implicit runge-kutta algorithms on parallel computers. *Mathematics of Computation*, 58(197):135–159, 1992.

[16] Jingfang Huang, Jun Jia, and Michael Minion. Accelerating the convergence of spectral deferred correction methods. *J. Comput. Phys.*, 214(2):633–656, 2006.

[17] Jingfang Huang, Jun Jia, and Michael Minion. Arbitrary order Krylov deferred correction methods for differential algebraic equations. *J. Comput. Phys.*, 221(2):739–760, 2007.

[18] Anita T. Layton and Michael L. Minion. Implications of the choice of quadrature nodes for Picard integral deferred corrections methods for ordinary differential equations. *BIT*, 45(2):341–373, 2005.

[19] A.T. Layton. On the choice of correctors for semi-implicit Picard deferred correction methods. *Applied Numerical Mathematics*, 58(6):845–858, 2008.

[20] A.T. Layton and M.L. Minion. Implications of the choice of predictors for semi-implicit Picard integral deferred correc-

tions methods. *Comm. Appl. Math. Comput. Sci.*, 1(2):1–34, 2007.

[21] J.L. Lions, Y. Maday, and G. Turinici. A "parareal" in time discretization of PDEs. *Comptes Rendus de l'Academie des Sciences Series I Mathematics*, 332(7):661–668, 2001.

[22] Y. Maday and G. Turinici. A parareal in time procedure for the control of partial differential equations. *Comptes rendus-Mathématique*, 335(4):387–392, 2002.

[23] M. Minion. A hybrid parareal spectral deferred corrections method.

[24] Michael L. Minion. Semi-implicit spectral deferred correction methods for ordinary differential equations. *Commun. Math. Sci.*, 1(3):471–500, 2003.

[25] F. Rudolf, K. Rupp, and Weinbub J. ViennaCL Web page, 2010. http://viennacl.sourceforge.net.

[26] Carl Runge. Über empirische Funktionen und die Interpolation zwischen äquidistanten Ordinaten. *Zeit. für Math. und Phys.*, 46:224–243, 1901.

[27] B.F. Smith, P. Bjorstad, and W. Gropp. *Domain decomposition: parallel multilevel methods for elliptic partial differential equations.* Cambridge Univ Pr, 2004.

[28] M.S. Warren and J.K. Salmon. A portable parallel particle program. *Computer Physics Communications*, 87(1–2):266 – 290, 1995. Particle Simulation Methods.