

# Parallel Semi-Implicit Time Integrators

B. Ong<sup>a,\*</sup>, A. Melfi<sup>a</sup>, A. Christlieb<sup>b</sup>

<sup>a</sup>*Institute for Cyber Enabled Research, Michigan State University, East Lansing, MI 48823.*

<sup>b</sup>*Department of Math, Michigan State University, East Lansing, MI 48823.*

---

## Abstract

In this paper, we further develop a family of parallel time integrators known as Revisionist Integral Deferred Correction methods (RIDC) to allow for the semi-implicit solution of time dependent PDEs. Additionally, we show that our semi-implicit RIDC algorithm can harness the computational potential of *multiple* general purpose graphical processing units (GPGPUs) by utilizing existing CUBLAS libraries for matrix linear algebra routines in our implementation. In the numerical experiments, we show that our implementation computes a fourth order solution using four GPGPUs and four CPUs in approximately the same wall clock time as a first order solution computed using a single GPGPU and a single CPU.

*Keywords:* Advection–Diffusion, Reaction–Diffusion, integral deferred correction, parallel integrators, graphics processing units

---

## 1. Introduction

RIDC methods are parallel-in-step time integrators [3, 5]. The “revisionist” terminology was first adopted in [3] to highlight that (i) this is a *revision* of the standard integral defect correction (IDC) methods, and (ii) successive corrections, running in parallel but lagging in time, *revise* and im-

---

<sup>\*</sup>Corresponding author  
Prepared for submission to *Journal of Computational Physics* November 2, 2011  
Email addresses: [ongbw@msu.edu](mailto:ongbw@msu.edu) (B. Ong), [melfiand@msu.edu](mailto:melfiand@msu.edu) (A. Melfi),  
[andrewch@msu.edu](mailto:andrewch@msu.edu) (A. Christlieb)  
URL: <http://mathgeek.us> (B. Ong)

24 prove the approximation to the solution. This notion of time parallelization  
25 is particularly exciting because it can be potentially layered upon existing  
26 spatial parallelization techniques [2, 15], including algorithms that utilize a  
27 single GPGPU card to solve time dependent PDEs [9, 14, 1], to add further  
28 parallel scalability.

29 The main idea behind RIDC methods is to re-write the defect correc-  
30 tion framework so that, after initial start-up costs, each correction loop can  
31 be lagged behind the previous correction loop in a manner that facilitates  
32 running the predictor and correctors in parallel. This idea for parallel time  
33 integrators was previously published by the present authors in [3, 5]. As  
34 before, this is still small scale parallelism in the sense that the time paral-  
35 lelization is limited by the order one wants to achieve.

36 To harness the computational potential of *multiple* graphical processing  
37 units (GPGPUs), the CUBLAS library [12] (which is a collection of linear  
38 algebra subroutines coded in CUDA) are utilized to demonstrate that by  
39 threading the RIDC loops, multiple GPGPUs can be utilized for our semi-  
40 implicit RIDC algorithm. We present numerical experiments in Section 4  
41 to show that our algorithm and implementation computes a fourth order  
42 semi-implicit solution using four GPGPUs and four CPUs in approximately  
43 the same wall clock time as a first order forward-backward Euler solution  
44 computed using a single GPGPU and a single CPU.

45 This paper is organized as follows. In Section 2, we review additive  
46 Runge–Kutta methods, which are a family of high order semi-implicit in-  
47 tegrators. In Section 3, semi-implicit RIDC methods and their properties  
48 are presented. Then, numerical benchmarks comparing RIDC and additive

49 Runge–Kutta methods are given in Section 4, followed by concluding remarks  
50 in Section 5.

## 51 2. ARK Methods

52 We are interested in solutions to initial value problems of the form,

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

53 where  $y, \alpha \in \mathbb{R}^n$ ,  $f^N : \mathbb{R} \times \mathbb{R}^n \rightarrow \mathbb{R}^n$  and  $f^S : \mathbb{R} \times \mathbb{R}^n \rightarrow \mathbb{R}^n$ . The func-  
54 tion  $f^S(t, y)$  contains stiff terms that need to be handled implicitly, and  
55  $f^N(t, y)$  consists of non-stiff terms that can be handled explicitly. A first  
56 order implicit-explicit (IMEX) discretization of the IVP (1) can be written  
57 as

$$\frac{y_{n+1} - y_n}{\Delta t} = f^S(t_{n+1}, y_{n+1}) + f^N(t_n, y_n), \quad \text{with } y_0 = \alpha.$$

58 The above IMEX discretization is particularly useful if  $f^S(t, y)$  is linear in  $y$ ,  
59 i.e.  $f^S(t, y) = Dy$ , which is often the case in a method of lines discretization  
60 of PDEs containing relaxation terms. In such cases, the IMEX discretization  
61 reduces to a linear system solve the solution at each time level,

$$(I - D\Delta t)y_{n+1} = y_n + \Delta t f^N(t_n, y_n), \quad \text{with } y_0 = \alpha. \quad (2)$$

62 Higher order IMEX discretizations are also possible. A popular choice are  
63 the so-called additive Runge–Kutta (ARK) methods, which couples  $s$ -stage

64 diagonally implicit RK (DIRK) and explicit  $s$ -stage explicit RK method

$$\begin{array}{c|cccc|cccc}
 0 & 0 & & & & 0 & & & & \\
 c_2 & a_{21}^S & a_{22}^S & 0 & \dots & a_{21}^N & 0 & & & \\
 \vdots & \vdots & \vdots & \ddots & & \vdots & & & \ddots & \\
 c_s & a_{s1}^S & a_{s2}^S & \dots & a_{ss}^S & a_{s1}^N & a_{s2}^N & \dots & a_{s,s-1}^N & 0 \\
 \hline
 & b_1^S & b_2^S & \dots & b_s^S & b_1^N & b_2^N & \dots & b_{s-1}^N & b_s^N
 \end{array}$$

65 to generate a high order semi-implicit integrator. The discretization of  
 66 IVP (1) using an ARK method can be written as

$$y_{n+1} = y_n + \Delta t \sum_{i=1}^s (b_i^S K_{ni}^S + b_i^N K_{ni}^N), \quad \text{with } y_0 = \alpha,$$

67 where the stages satisfy

$$\begin{aligned}
 K_{ni}^S &= f^S \left( t_n + c_i \Delta t, y_n + \Delta t \sum_{j=1}^i a_{ij}^S K_{nj}^S + \Delta t \sum_{j=1}^{i-1} a_{ij}^N K_{nj}^N \right) \\
 K_{ni}^N &= f^N \left( t_n + c_i \Delta t, y_n + \Delta t \sum_{j=1}^i a_{ij}^S K_{nj}^S + \Delta t \sum_{j=1}^{i-1} a_{ij}^N K_{nj}^N \right).
 \end{aligned}$$

68 The third and fourth order ARK methods from [10] are used to bench-  
 69 mark against our fourth order RIDC-FBE (RIDC constructed using forward  
 70 and backward Euler integrators). These ARK methods have the following

71 Butcher tableaux:

0	0					0				
$\frac{1}{2}$	0	$\frac{1}{2}$				$\frac{1}{2}$				
$\frac{1}{2}$	$\frac{1}{4}$	$-\frac{5}{12}$	$\frac{2}{3}$			$\frac{1}{4}$	$\frac{1}{4}$			
1	2	$-\frac{7}{2}$	$\frac{1}{2}$	2			0	1	0	
1	$\frac{1}{6}$	0	$\frac{2}{3}$	$-\frac{5}{6}$	1	$\frac{1}{6}$	0	$\frac{2}{3}$	$\frac{1}{6}$	
	$\frac{1}{6}$	0	$\frac{2}{3}$	$-\frac{5}{6}$	1	$\frac{1}{6}$	0	$\frac{2}{3}$	$\frac{1}{6}$	

72

0	0							0						
$\frac{1}{3}$	$-\frac{1}{6}$	$\frac{1}{2}$					$\frac{1}{3}$							
$\frac{1}{3}$	$\frac{1}{6}$	$-\frac{1}{3}$	$\frac{1}{2}$				$\frac{1}{6}$	$\frac{1}{6}$						
$\frac{1}{2}$	$\frac{3}{8}$	$-\frac{3}{8}$	0	$\frac{1}{2}$			$\frac{1}{8}$	0	$\frac{3}{8}$					
$\frac{1}{2}$	$\frac{1}{8}$	0	$\frac{3}{8}$	$-\frac{1}{2}$	$\frac{1}{2}$			$\frac{1}{8}$	0	$\frac{3}{8}$	0			
1	$-\frac{1}{2}$	0	3	$-3$	1	$\frac{1}{2}$	$\frac{1}{2}$	0	$-\frac{3}{2}$	0	2			
1	$\frac{1}{6}$	0	0	0	$\frac{2}{3}$	$-\frac{1}{2}$	$\frac{2}{3}$	$\frac{1}{6}$	0	0	0	$\frac{2}{3}$	$\frac{1}{6}$	
	$\frac{1}{6}$	0	0	0	$\frac{2}{3}$	$-\frac{1}{2}$	$\frac{2}{3}$	$\frac{1}{6}$	0	0	0	$\frac{2}{3}$	$\frac{1}{6}$	0

73 (Note, a second order ARK scheme was also tested, but not presented because  
74 of its poor stability constraints). Similar to before, if  $f^S(t, y)$  is linear in  $y$ ,  
75 then each stage computation reduces to a linear solve since a DIRK method  
76 was paired with an explicit integrator in the discussed ARK methods.

77 **3. RIDC Methods**

78 RIDC methods are a class of time integrators based on integral de-  
 79 ferred correction [6]. RIDC methods first compute a prediction to the so-  
 80 lution (“level 0”) using low order schemes (e.g. a first order implicit-explicit  
 81 method) followed by one or more corrections to compute subsequent solution  
 82 levels. Each correction *revises* the solution and increases the formal order  
 83 of accuracy by 1, if a first order implicit-explicit integrator is used to solve  
 84 the error equation. Each correction level is delayed from the previous level  
 85 as illustrated in Figure 1 – the open circles denote solution values that are  
 86 simultaneously computed. This staggering in time means that the predictor  
 87 and each corrector can all be executed simultaneously, in parallel, while each  
 processes a different time-step.

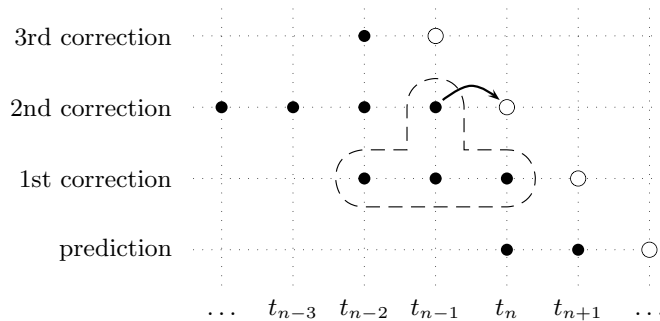


Figure 1: (RIDC4-FBE) This plot shows the staggering required for a fourth order RIDC scheme, constructed using a first order implicit–explicit predictors and correctors. The time axis runs horizontally, and the correction levels run vertically. The white circles denote solution values that are simultaneously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

89 In Section 3.1, we first derive the error equation. Then, Section 3.2 and  
 90 Section 3.3 give numerical schemes for solving the IVP and the error equation.  
 91 In Section 3.4, we review theorems related to the formal order of accuracy  
 92 that follow trivially from [4], and in Section 3.5, we summarize starting and  
 93 stopping details for the RIDC algorithm as well as the notion of resets.

### 94 3.1. Error Equation

95 Suppose an approximate solution  $\eta(t)$  to IVP (1) is computed. Denote  
 96 the exact solution as  $y(t)$ . Then, the error of the approximate solution is

$$e(t) = y(t) - \eta(t). \quad (3)$$

97 If we define the residual as  $\epsilon(t) = \eta'(t) - f^S(t, \eta(t)) - f^N(t, \eta(t))$ , then the  
 98 derivative of the error (3) satisfies

$$\begin{aligned} e'(t) &= y'(t) - \eta'(t) \\ &= f^S(t, y(t)) + f^N(t, y(t)) - f^S(t, \eta(t)) - f^N(t, \eta(t)) - \epsilon(t). \end{aligned}$$

99 The integral form of the error equation

$$\begin{aligned} \left[ e(t) + \int_a^t \epsilon(\tau) d\tau \right]' &= f^S(t, \eta(t) + e(t)) + f^N(t, \eta(t) + e(t)) \\ &\quad - f^S(t, \eta(t)) - f^N(t, \eta(t)) \end{aligned} \quad (4)$$

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

### 101 3.2. The predictor

102 To generate a provisional solution that can be corrected, a low order  
 103 integrator is applied to solve IVP (1); this process is typically known as  
 104 the prediction loop. The first-order IMEX scheme reviewed in Section 2

105 will be used to generate our RIDC-FBE (RIDC forward and backward Euler  
 106 method) though in theory, any ARK methods reviewed in Section 2 can be  
 107 used. We adopt the following notation:

$$\eta_{n+1}^{[0]} = \eta_n^{[0]} + \Delta t_n f^S(t_{n+1}, \eta_{n+1}^{[0]}) + \Delta t_n f^N(t_{n+1}, \eta_{n+1}^{[0]}), \quad (5)$$

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

### 110 3.3. The corrector

111 The correctors are also low order integrators, but are used to solve the  
 112 error equation (4) for the error  $e(t)$  to an approximate solution  $\eta(t)$ . Since the  
 113 error equation is solved iteratively to improve a solution from the previous  
 114 level, each correction level computes an error  $e^{[j-1]}(t)$  to the solution at the  
 115 previous level  $\eta^{[j-1]}(t)$  to obtain a revised solution  $\eta^{[j]}(t) = \eta^{[j-1]}(t) + e^{[j-1]}(t)$ .

116 A first order IMEX discretization of the error equation (4) (after some  
 117 algebra) gives

$$\begin{aligned} \eta_{n+1}^{[j]} = & \eta_n^{[j]} + \Delta t \left[ f^S(t_{n+1}, \eta_{n+1}^{[j]}) + f^N(t_n, \eta_n^{[j]}) \right] - \dots \\ & \left[ \Delta t f^S(t_{n+1}, \eta_{n+1}^{[j-1]}) + f^N(t_n, \eta_n^{[j-1]}) \right] + \int_{t_n}^{t_{n+1}} f(\tau, \eta^{[j-1]}(\tau)) d\tau. \end{aligned} \quad (6)$$

118 The integral  $\int_{t_n}^{t_{n+1}} f(\tau, \eta^{[j-1]}(\tau)) d\tau$  is approximated using quadrature. For  
 119 the  $j^{\text{th}}$  correction loop,  $(j + 1)$  nodes are needed in the stencil to accurately  
 120 approximate the integral. There are various choices for the stencil, but in  
 121 practice, the stencil should include the nodes  $t_n$  and  $t_{n+1}$ . We make the

122 following choice for selecting our quadrature nodes:

$$\begin{aligned}
 & \int_{t_n}^{t_{n+1}} f(\tau, \eta^{[j-1]}(\tau)) d\tau \approx & (7) \\
 & \begin{cases} \sum_{k=0}^j \alpha_{nk} \left( f^N(t_{n+1-k}, \eta_{n+1-k}^{[j-1]}) + f^S(t_{n+1-k}, \eta_{n+1-k}^{[j-1]}) \right), & \text{if } (n \geq j-1) \\ \sum_{k=0}^j \alpha_{nk} \left( f^N(t_k, \eta_k^{[j-1]}) + f^S(t_k, \eta_k^{[j-1]}) \right), & \text{if } (n < j-1) \end{cases}
 \end{aligned}$$

123 where the quadrature weights are given by

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

124 for  $n \geq j-1$ , and

$$\alpha_{nk} = \int_{t_n}^{t_{n+1}} \prod_{i=0, i \neq k}^j \frac{(t - t_i)}{(t_k - t_i)} dt, \quad k = 0, 1, \dots, j-1$$

125 for  $n < j-1$ . Since uniform time steps are used in the computation, then  
 126 only one set of quadrature weights needs to be computed, stored, then used  
 127 as necessary.

### 128 3.4. Formal order of accuracy

129 The analysis in [4], proving convergence under mild conditions for IDC-  
 130 ARK methods, extends simply to these RIDC-ARK methods.

131 **Theorem 3.1.** Let  $f(t, y)$  and  $y(t)$  in IVP (1) be sufficiently smooth. Then,  
 132 the local truncation error for a RIDC method constructed using a first order  
 133 IMEX integrators for the predictor and  $(p-1)$  correction loops is  $\mathcal{O}(\Delta t^{p+1})$ .

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

138 *3.5. Further Comments*

139 During most of a RIDC calculation, multiple solution levels are marched  
140 in a pipe using multiple computing CPUs/GPGPUs. However, the comput-  
141 ing nodes in the RIDC algorithm cannot start simultaneously: each must  
142 wait for the previous level to compute sufficient  $\eta$  values before they can  
143 be marched in a pipeline fashion. By carefully controlling the start-up of a  
144 RIDC method, one can minimize the amount of memory that is required to  
145 march the nodes in a pipeline fashion. In our implementation, the order in  
146 which computations are performed during start-up is illustrated in Figure 2  
147 for a fourth order RIDC constructed with first order IMEX predictors and  
148 correctors. The  $j^{\text{th}}$  processor (running the  $j^{\text{th}}$  correction) must initially wait  
149 for  $j(j+1)/2$  steps, e.g., node 2 has to wait 3 steps before starting. There  
150 are also idle computing threads at the end of the computation, since the  
151 predictor and lower level correctors will reach  $t_N = b$  earlier than the last  
152 corrector.

153 An important notion to consider are “resets”, that is, instead of comput-  
154 ing all the way to the final time, we compute on some smaller time interval to  
155 time  $t_*$ , and use the most accurate solution to reset the RIDC computation  
156 at  $t = t_*$ . In practice, resetting improves the stability of the semi implicit  
157 RIDC scheme, and could lower the error constant of the overall method, but  
158 at the cost of decreasing the speedup due to the additional cost of starting  
159 the RIDC algorithm multiple times.

160 Additionally, one cannot increase the order of RIDC indefinitely as (i) it is  
161 not practical (when would one ever want a 16th order method?) and (ii) the  
162 Runge phenomenon [13], which arises from using equi-spaced interpolation

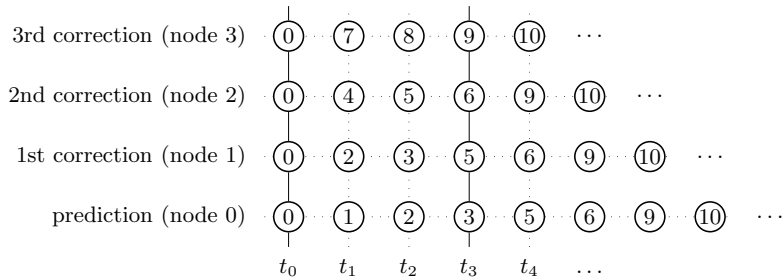


Figure 2: This figure is a graphical representation of how the RIDC4-BE algorithm is started. The time axis runs horizontally, the correction levels run vertically. All nodes are initially populated with the initial data at  $t_0$ . This is represented by computing step 0 (enclosed in a circle). At computing step 1, node 0 computes the predicted solution at  $t_1$ . The remaining nodes remain idle. At computing step 2, node 0 computes the predicted solution at time  $t_2$ , node 1 computes the 1st corrected solution at time  $t_1$ , the remaining two nodes remain idle. Note that in this starting algorithm, special care is taken to ensure that minimum memory is used by not letting the computing cores run ahead until they can be marched in a pipeline; in this example, when node 3 starts computing  $t_3$ .

163 points, will eventually cause the scheme to become unstable. In practice, 8th  
164 and 12th order RIDC methods using double precision do not suffer from the  
165 Runge phenomenon.

166 Lastly, there is another family of parallel time integrators, known as  
167 parareal methods [11], that is actively being researched [7, 8]. These meth-  
168 ods fall into the class of “parallel across the method” algorithms, where the  
169 entire time domain is split across multiple nodes, a coarse operator is run in  
170 serial, followed by a parallel correction update. We encourage users to more  
171 carefully consider parareal if the small scale parallelism offered by RIDC is  
172 not sufficient.

#### 173 4. Numerical Examples

174 Advection–reaction–diffusion equations have been widely used to model  
175 chemical processes across many disciplines. Here, we present two numeri-  
176 cal examples: an advection–diffusion and a reaction–diffusion equation, to  
177 validate the order of accuracy of the RIDC4-FBE scheme, and the speedup  
178 obtained in the parallel OpenMP framework, and the OpenMP–CUDA hy-  
179 brid framework. In each example, the stiff term is chosen as the diffusion  
180 operator,  $f^S(t, y) = y_{xx}$ . Applying a centered finite difference operator to  
181 approximate  $\partial_{xx}$  reduces each RIDC/ARK step to a series of decoupled lin-  
182 ear system solves. The matrices are pre-factored into their QR components  
183 so that each linear solve is reduced to a matrix–vector multiplication and a  
184 back solve operation.

185 The computations presented were performed on a stand alone server con-  
186 taining a quad core AMD Phenom X4 9950 2.6Ghz processor with four Nvidia

187 Tesla GPU C1060 GPGPU cards. The ARK schemes are coded using plain  
 188 C++ with (i) a homegrown linear algebra library and (ii) the CUBLAS 4.0  
 189 library [12]. The RIDC4-FBE is coded in C++ with (i) OpenMP and the  
 190 homegrown linear algebra library, and (ii) OpenMP and the CUBLAS 4.0  
 191 library. Some important subtleties for creating a hybrid OpenMP – CUDA  
 192 RIDC code are: (i) we can control which GPGPU card is used for a linear  
 193 solve by calling the `cudaSetDevice()` function, and (ii) in using “`#pragma`  
 194 `for`” loop to spawn individual threads for each prediction/correction loop,  
 195 we have to utilize static scheduling.

#### 196 *4.1. Advection-Diffusion*

197 We first consider the canonical advection-diffusion problem to show that  
 198 we can achieve designed orders of accuracy for our RIDC-FBE algorithm.  
 199 The constant coefficient advection-diffusion equation

$$u_t = cu_x + du_{xx}, \quad x \in [0, 1], \quad t \in [0, 40],$$

$$u(x, 0) = 2 + \sin(2\pi x),$$

200 with periodic boundary conditions, is discretized using the method of lines  
 201 methodology. Specifically, the advection term is discretized using upwind  
 202 first order differences, and the diffusion term is discretized using central dif-  
 203 ferences. The following system is then recovered,

$$u_t = Au + Du, \quad u(0) = \alpha,$$

204 where the matrix  $A$  approximates the advection operator, and the matrix  
 205  $D$  approximates the diffusion operator. We choose the obvious splitting,  
 206  $f^N(t, u) = Au$ , and  $f^S(t, u) = Du$ . We take  $c = 0.1, d = 10^{-3}, \Delta x = \frac{1}{1000}$ .

207 First, we show in Figure 3a that ARK and RIDC4 achieve their designed  
 208 orders of accuracy. Observe that the error coefficient for ARK4 is several  
 209 orders of magnitude smaller than that of RIDC4. In Figure 3b, we instead  
 210 plot the results from the same numerical run, this time plotting error as  
 211 a function of the wall clock time. Several observations can be made: (i)  
 212 for all the schemes, our GPU implementation is approximately an order of  
 213 magnitude faster than the CPU implementation, (ii) RIDC4 (both the CPU  
 214 and GPU implementations) compute a fourth order solution in the same  
 215 wall clock as the FBE solution, (iii) for a fixed wall clock time, RIDC4 (with  
 216 4 GPGPUs) computes a solution that is several orders of magnitude more  
 217 accurate than the solution computed using ARK4 (with 1 GPGPU).

218 We also show in Figure 4 the error of RIDC4 as a function of restarts.  
 219 As expected, the error decreases as the number of restarts is increased. The  
 220 penalty for each restart is having to fill the memory footprint at each restart  
 221 before marching the cores/GPGPUs in a pipe.

222 Finally, in Figure 5, we show the speedup that is obtained when RIDC4  
 223 is computed using one, two and four CPUs, and when RIDC4 is computed  
 224 using one, two and for GPGPUs. We appear to obtain almost linear speedup,  
 225 even with 10 restarts.

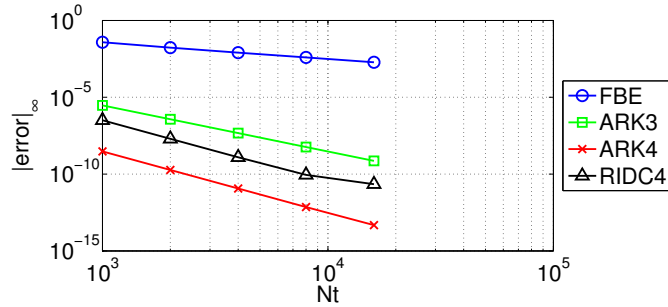
#### 226 4.2. Viscous Burgers' Equation

227 We also consider the solution to viscous Burgers' equation,

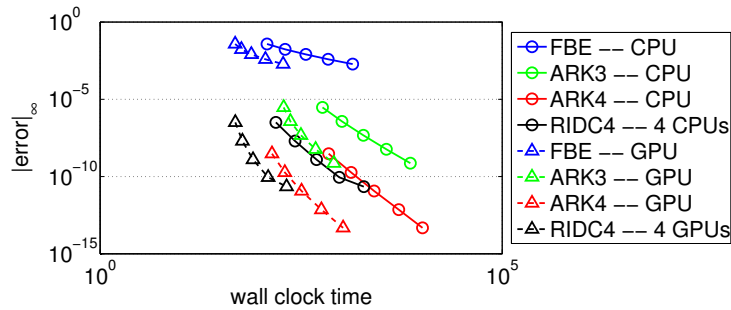
$$u_t + \frac{1}{2} (u^2)_x = \epsilon u_{xx}, \quad (x, t) \in [0, 1] \times [0, 1],$$

228 with initial and boundary conditions

$$u(0, t) = u(1, t) = 0, \quad u(x, 0) = \sin(2\pi x) + \frac{1}{2} \sin(\pi x).$$



(a) Convergence study: error versus number of time steps



(b) error versus wall clock time

Figure 3: (a) This standard “error versus step size” convergence study for RIDC4-FBE (with 10 restarts) and the various ARK methods presented in section 2. All schemes achieve their designed orders of accuracy. Observe that the RIDC4-FBE error coefficient is much larger than that of the ARK4 scheme. This is a small price to pay for the parallel speedup that can be obtained, as shown in (b). Two observations should be made: (i) for all the schemes, our GPU implementation is approximately an order of magnitude faster than the CPU implementation, (ii) RIDC4 (both the CPU and GPU implementations) compute a fourth order solution in the same wall clock as the FBE solution.

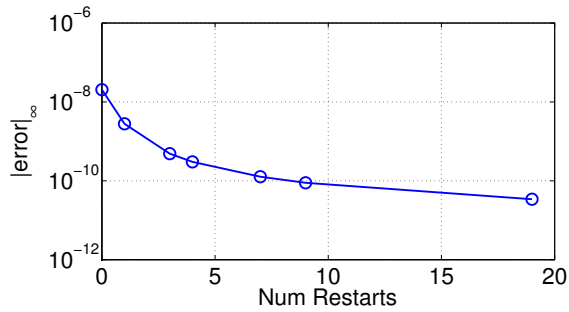


Figure 4: The error of RIDC4 schemes at the final time  $T = 40$  decreases as the number of restarts is increased (for a fixed number of time steps, in this case, 4000 time steps). Each restart requires that the memory footprint be refilled before the cores/GPGPUs can be marched in a pipe.

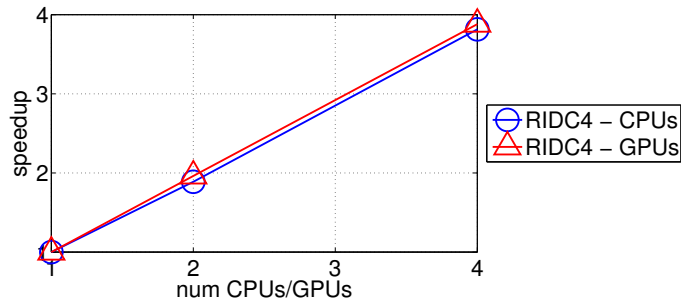


Figure 5: Our RIDC4 implementation (both the CPU and GPU implementation) attains almost linear speedup.

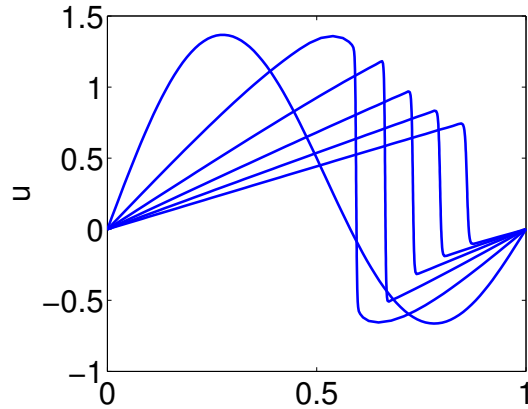


Figure 6: Solution to Burgers' equation, with  $\epsilon = 10^{-3}$  and  $\Delta x = \frac{1}{1000}$ . Time snapshots at  $t = 0, 0.2, 0.4, 0.6, 0.8$  and  $1$  are shown.

229 The solution develops a layer that propagates to the right, as shown in Fig-  
 230 ure 6. The diffusion term is again discretized using centered finite differences.

231 A numerical flux is used to approximate the advection operator,

$$\frac{1}{2}((u_i^n)^2)_x = \frac{1}{2} \frac{f_{i+1/2}^n - f_{i-1/2}^n}{\Delta x},$$

232 where

$$f_{i+1/2}^n = \frac{1}{2} ((u_{i+1}^n)^2 + (u_i^n)^2).$$

233 Hence, the following system of equations is obtained,

$$u_t = \mathcal{L}(u) + Du,$$

234 where the operator  $\mathcal{L}(u)$  approximates the hyperbolic term using the numer-  
 235 ical flux, and the matrix  $D$  approximates the diffusion operator. We choose  
 236 the splitting  $f^N(t, u) = \mathcal{L}(u)$  and  $f^S(t, u) = Du$ , and take  $\epsilon = 10^{-3}$  and  
 237  $\Delta x = \frac{1}{1000}$ . No restarts are used for this simulation.

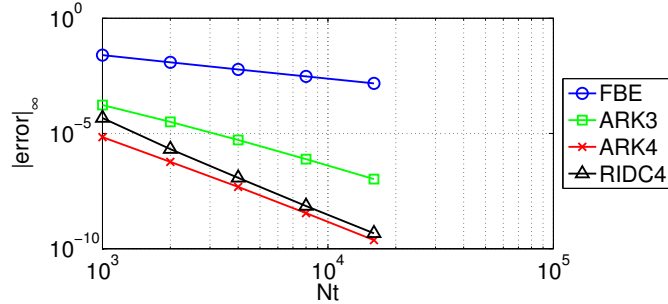
238 The same numerical results as the previous advection–diffusion example  
239 are observed in Figure 7. In plot (a), the RIDC scheme achieves it’s designed  
240 order of accuracy. In plot (b), we show that our RIDC implementations  
241 (both the CPU and GPGPU versions) obtain a fourth order solution in the  
242 same wall clock time as a first order semi-implicit FBE solution. The RIDC  
243 implementations with multiple CPU/GPU resources also achieve comparable  
244 errors to a fourth order ARK scheme in approximately one tenth the time.  
245 In plot (c), we show that our RIDC implementation obtains linear speedup.

## 246 5. Conclusions

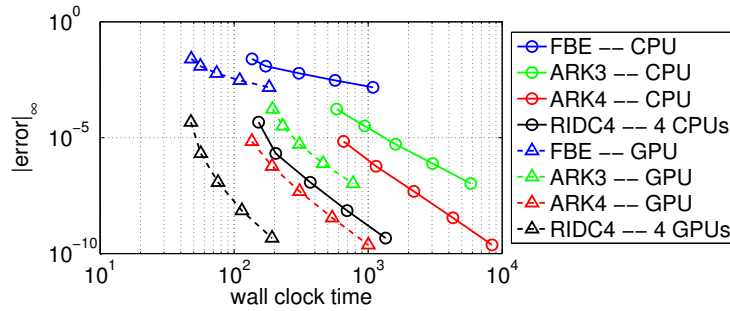
247 In this paper, we further developed RIDC algorithms to generate a fam-  
248 ily of high order semi-implicit parallel integrators. The analysis related to  
249 convergence is a simple extension from previous work, and the numerical ex-  
250 periments demonstrate that the fourth order RIDC-FBE algorithm achieves  
251 its designed order of accuracy. Additionally, we showed that our semi-implicit  
252 RIDC algorithm harnessed the computational potential of four GPGPUs by  
253 utilizing OpenMP coupled with with the CUBLAS library. This semi-implicit  
254 RIDC algorithm can potentially be coupled with existing legacy parallel spa-  
255 tial codes. Work is on-going to explore a hybrid MPI–OpenMP–CUDA al-  
256 gorithm for more heterogeneous architectures.

## 257 Acknowledgments

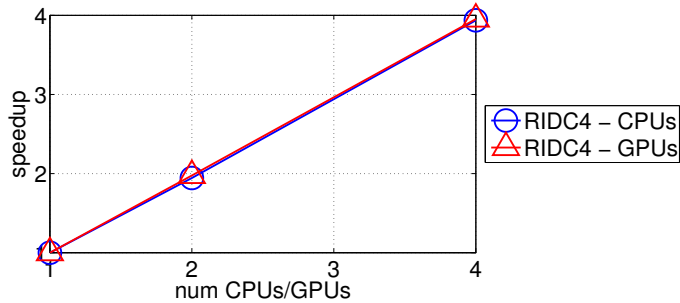
258 This work was supported by AFRL and AFOSR under contract and  
259 grants FA9550-07-0092 and FA9550-07-0144 and NSF grant number DMS-  
260 0934568. We also wish to acknowledge the support of the Michigan State



(a) Convergence study: error versus number of time steps



(b) error versus wall clock time



(c) Speedup of RIDC4

Figure 7: In (a), we show that the ARK schemes and our RIDC4-FBE scheme achieve the designed orders of accuracy. The plot in (b) shows the error as a function of wall clock time. Two observations should be made: (i) for all the schemes, our GPU implementation is approximately an order of magnitude faster than the CPU implementation. In plot (c), we show that our RIDC4 implementation attains nearly perfect speedup.

261 University High Performance Computing Center and the Institute for Cyber  
262 Enabled Research.

## 263 **References**

- 264 [1] Joshua A. Anderson, Chris D. Lorenz, and A. Travasset. General pur-  
265 pose molecular dynamics simulations fully implemented on graphics pro-  
266 cessing units. *Journal of Computational Physics*, 227(10):5342 – 5359,  
267 2008.
- 268 [2] Andrew Christlieb, Ron Haynes, and Benjamin Ong. A parallel space-  
269 time algorithm. *submitted*.
- 270 [3] Andrew Christlieb, Colin Macdonald, and Benjamin Ong. Parallel high-  
271 order integrators. *SIAM J. Sci. Comput.*, 32(2):818–835, 2010.
- 272 [4] Andrew Christlieb, Maureen Morton, Benjamin Ong, and Jing-Mei Qiu.  
273 Semi-implicit integral deferred correction constructed with high order  
274 additive Runge-Kutta integrators. *Comm. Math. Sci*, 9(3):879–902,  
275 2011.
- 276 [5] Andrew Christlieb and Benjamin Ong. Implicit parallel time integrators.  
277 *J. Sci. Comput.*, 49(2):167–179, 2011.
- 278 [6] Alok Dutt, Leslie Greengard, and Vladimir Rokhlin. Spectral deferred  
279 correction methods for ordinary differential equations. *BIT*, 40(2):241–  
280 266, 2000.

- 281 [7] W. Elwasif, S. Foley, D. Bernholdt, L. Berry, D. Samaddar, Newman D.,  
282 and R Sanchez. A dependency-driven formulation of parareal: Parallel-  
283 in-time solution of pdes as a many-task application. *4th IEEE Workshop*  
284 *on Many-Task Computing on Grids and Supercomputers*, submitted.
- 285 [8] M. Emmett and M. Minion. Toward an efficient parallel in time method  
286 for partial differential equations. *Journal of Computational Physics*,  
287 submitted.
- 288 [9] Christian Janßen and Manfred Krafczyk. Free surface flow simulations  
289 on gpgpus using the lbm. *Computers and Mathematics with Applica-*  
290 *tions*, 61(12):3549 – 3563, 2011.
- 291 [10] Hongyu Liu and Jun Zou. Some new additive Runge–Kutta methods  
292 and their applications. *J. Comput. Appl. Math.*, 190(1-2):74–98, 2006.
- 293 [11] Y. Maday and G. Turinici. A parareal in time procedure for the con-  
294 trol of partial differential equations. *Comptes rendus-Mathématique*,  
295 335(4):387–392, 2002.
- 296 [12] Nvidia. NVIDIA CUDA C programming guide, 2011.  
297 <http://developer.nvidia.com>.
- 298 [13] Carl Runge. Über empirische Funktionen und die Interpolation zwischen  
299 äquidistanten Ordinaten. *Zeit. für Math. und Phys.*, 46:224–243, 1901.
- 300 [14] Daisuke Sato, Yuanfang Xie, James Weiss, Zhilin Qu, Alan Garfinkel,  
301 and Allen Sanderson. Acceleration of cardiac tissue simulation with

302 graphic processing units. *Medical and Biological Engineering and Com-*  
303 *puting*, 47:1011–1015, 2009. 10.1007/s11517-009-0514-4.

304 [15] Andrea Toselli and Olof Widlund. *Domain decomposition methods—*  
305 *algorithms and theory*, volume 34 of *Springer Series in Computational*  
306 *Mathematics*. Springer-Verlag, Berlin, 2005.