A theoretical comparison of high order explicit Runge-Kutta, extrapolation, and deferred correction methods

David I. Ketcheson* Umair bin Waheed†
May 28, 2013

Abstract

We analyze computational efficiency of serial and parallel high order numerical methods for ODEs. We focus on extrapolation, deferred correction, and Runge–Kutta embedded pairs of orders four through twelve. We cast extrapolation and deferred correction methods as fixed-order Runge–Kutta methods, providing a natural framework for the comparison. In serial, only midpoint extrapolation is competitive with more standard Runge–Kutta methods. But high order extrapolation and deferred correction methods are potentially more efficient than standard high-order Runge–Kutta methods when concurrency is taken into account. Computational tests confirm the theoretical predictions.

1 Introduction

The construction of very high order integrators for initial value ordinary differential equations (ODEs) is challenging: very high order Runge–Kutta (RK) methods are subject to vast numbers of order conditions, while very high order linear multistep methods tend to have poor stability properties. Both extrapolation [9, 16] and deferred correction [7, 10] can be used to construct initial value ODE integrators of arbitrarily high order in a straightforward way.

Both are usually viewed as iterative methods, since they build up a high order solution based on lower order approximations. However, when the order is fixed, methods in both classes can be viewed as Runge–Kutta methods with a number of stages that grows quadratically with the desired order of accuracy.

It is natural to ask how these methods compare with standard Runge–Kutta methods. Previous studies have compared the relative (serial) efficiency of explicit extrapolation and Runge–Kutta (RK) methods [18, 33, 17], finding that extrapolation methods have no advantage over moderate to high order Runge–Kutta methods, and may well be inferior to them [33, 17]. Consequently, extrapolation has received little attention in the last two decades. It seems that no work has thoroughly compared the efficiency of deferred correction methods with that of their extrapolation and RK counterparts.

In this paper we compare the efficiency of explicit Runge–Kutta, extrapolation, and deferred correction (DC) methods based on their accuracy and stability properties. The methods we study are introduced in Section 2 and range in order from four to twelve though much of our analysis applies to methods of arbitrary order. In order to make our study broadly applicable and avoid bias due to implementation differences, we focus on theoretical metrics, described in Section 3. Our study is similar in spirit and in methodology to that of Hosea & Shampine [17]. In Section 4 we validate the theoretical predictions using simple numerical tests. In the numerical

*King Abdullah University of Science & Technology (KAUST), Thuwal, 23955-6900 Saudi Arabia, ketcheson@kaust.edu.sa
†King Abdullah University of Science & Technology (KAUST), Thuwal, 23955-6900 Saudi Arabia
tests, we measure the number of function evaluations rather than the wall clock time in order to avoid implementation- or machine-dependent effects. These tests indicate, in agreement with our theoretical analysis and with previous studies, that high order extrapolation and deferred correction methods do not have a significant advantage over high order Runge–Kutta methods, and may in fact be significantly less efficient.

In Section 5 we analyze the potential of parallel implementations of extrapolation and deferred correction methods. It has long been recognized that extrapolation methods offer excellent opportunities for parallel implementation [9]. Nevertheless, to our knowledge no parallel implementation has appeared, and comparisons of extrapolation methods have not taken parallel computation into account, even from a theoretical perspective. Recently, a “revisionist” integral deferred correction method has been implemented in parallel and seems promising [5]. We focus on a simpler modification of deferred correction that also allows for significant parallelism. For both extrapolation and (appropriately chosen) deferred correction methods, the number of stages that must be computed sequentially grows only linearly with the desired order of accuracy. Based on simple algorithmic analysis, we extend our theoretical analysis to parallel implementations of extrapolation and deferred correction.

No study of numerical methods can claim to yield conclusions that are valid for all possible problems. Our intent is to give some broadly useful comparisons and draw general conclusions that can serve as a guide to further studies. The analysis presented here was performed using the NodePy (Numerical ODEs in Python) package, which is freely available from [http://github.com/ketch/nodepy](http://github.com/ketch/nodepy).

2 High order one-step embedded pairs

...for high order RK formulas the construction of an embedding RK formula may be beyond human possibilities...

(P. Deuflhard, 1985)

We are concerned with one-step methods for the solution of the initial value ODE

\[ y'(t) = f(y) \quad y(t_0) = y_0, \]

(1)

where \( y \in \mathbb{R}^m, f : \mathbb{R}^m \rightarrow \mathbb{R}^m \). For simplicity of notation, we assume the problem has been written in autonomous form. An explicit Runge–Kutta pair computes approximations \( y_n, \hat{y}_n \approx y(t_n) \) as follows:

\[
\begin{align*}
Y_i &= y_n + h \sum_{j=1}^{s} a_{ij} f(Y_j) \\
y_{n+1} &= y_n + h \sum_{j=1}^{s} b_j f(Y_j) \\
\hat{y}_{n+1} &= y_n + h \sum_{j=1}^{s} \hat{b}_j f(Y_j).
\end{align*}
\]

Here \( h \) is the step size, the stages \( Y_i \) are intermediate approximations, and one evaluation of \( f \) is required for each stage. The coefficients \( A, b, \hat{b} \) determine the accuracy and stability of the method. The coefficients are typically chosen so that \( y_{n+1} \) has local error \( \tau = \mathcal{O}(h^p) \), and \( \hat{y}_{n+1} \) has local error \( \hat{\tau} = \mathcal{O}(h^{\hat{p}}) \) for some \( 1 < \hat{p} < p \). Here \( p \) is referred to as the order of the method, and sometimes such a method is referred to as a \( p(\hat{p}) \) pair. The value \( \| y_{n+1} - \hat{y}_{n+1} \| \) is used to estimate the error and determine an appropriate size for the next step.

The theory of Runge–Kutta order conditions gives necessary and sufficient conditions for a Runge-Kutta method to be consistent to a given order [16] [3]. For order \( p \), these conditions
involve polynomials of degree up to $p$ in the coefficients $A, b$. The number of order conditions increases dramatically with $p$: only eight conditions are required for order four, but order ten requires 1,205 conditions and order fourteen requires 53,263 conditions. Although the order conditions possess a great deal of structure and certain simplifying assumptions can be used to facilitate their solution, the design of efficient Runge–Kutta pairs of higher than eighth order by direct solution of the order conditions remains a challenging area. Some methods of order as high as 14 have been constructed [12].

2.1 Extrapolation methods

Extrapolation methods provide a straightforward approach to the construction of high order one-step methods; they can be viewed as Runge–Kutta methods, which is the approach taken here. For the mathematical foundations of extrapolation methods we refer the reader to [16, Section II.9]. The algorithmic structure of extrapolation methods has been considered in detail in previous works, including [32, 37]; we review the main results here. Various sequences of step numbers have been proposed, but we consider the harmonic sequence as it is usually the most efficient [8, 17]. We do not consider the use of smoothing, as previous studies have shown that it reduces efficiency [17].

2.1.1 Euler extrapolation (Ex-Euler)

Extrapolation is most easily understood by considering the explicit Euler method

$$y_{n+1} = y_n + hf(y_n) \tag{2}$$

as a building block. The order $p$ Ex-Euler algorithm computes $p$ approximations to $y(t_{n+1})$ by using the explicit Euler method, first breaking the interval into one step, then two steps, and so forth. The approximations to $y(t_{n+1})$ computed in this manner are all first order accurate and are labeled $T_{11}, T_{21}, \ldots, T_{p1}$. These values are combined using the Aitken-Neville interpolation algorithm to obtain a higher order approximation to $y(t_{n+1})$. The algorithm is depicted in Figure 1. For error estimation, we use the approximation $T_{p-1,p-1}$ whose accuracy is one order less.

| Algorithm 1 Explicit Euler extrapolation (Ex-Euler) |
|----------------------------------------------------|
| for $k = 1 \rightarrow p$ do \>
| \> \> Compute first order approximations \>
| $Y_{k0} = y_n$ \>
| for $j = 1 \rightarrow k$ do \>
| \> \> $Y_{kj} = Y_{k,j-1} + \frac{h}{k} f(Y_{k,j-1})$ \>
| end for \>
| $T_{k1} = Y_{kk}$ \>
| end for \>
| for $k = 2 \rightarrow p$ do \>
| \> \> Extrapolate to get higher order \>
| for $j = k \rightarrow p$ do \>
| \> \> $T_{jk} = T_{j,k-1} + \frac{T_{j,k-1} - T_{j-1,k-1}}{j-k+1}$ \>
| \> \> Aitken-Neville formula for extrapolation to order $k$ \>
| end for \>
| $y_{n+1} = T_{pp}$ \>
| $\hat{y}_{n+1} = T_{p-1,p-1}$ \>

\> New solution value
\> Embedded method solution value

Simply counting the number of evaluations of $f$ in Algorithm 1 shows that this is an $s$-stage
Runge-Kutta method, where

\[ s = \frac{p^2 - p + 2}{2}. \]

The quadratic growth of \( s \) as the order \( p \) is increased leads to relative inefficiency of very high order extrapolation methods when compared to directly constructed Runge–Kutta methods, as we will see in later sections.

2.1.2 Midpoint extrapolation (Ex-Midpoint)

It is common to perform extrapolation based on an integration method whose error function contains only even terms, such as the midpoint method \([10, 57]\). In this case, each extrapolation step raises the order of accuracy by two. We refer to this approach as Ex-Midpoint and give the algorithm below. Using midpoint extrapolation to obtain order \( p \) requires about half as many stages, compared to Ex-Euler:

\[ s = \frac{p^2 + 4}{4}. \]

Again, the number of stages grows quadratically with the order.

2.2 Deferred correction (DC-Euler)

Like extrapolation, deferred correction has a long history; its application to initial value problems goes back to \([7]\). Recently it has been revived as an area of research, see \([10, 15]\) and subsequent works. Here we focus on the class of methods introduced in \([10]\), with a modification introduced in \([27]\). These DC methods are one-step methods and can be constructed for any order of accuracy.

DC methods start like extrapolation methods, by using a low-order method to step over subintervals of the time step; the subintervals can be equally sized or Chebyshev nodes can be used; we consider methods based on the explicit Euler method and Chebyshev nodes. Subsequently, high-order polynomial interpolation of the computed values is used to approximate the
Algorithm 2 Explicit Midpoint extrapolation (Ex-Midpoint)

\[
\begin{align*}
r &= p/2 \\
&\text{for } k = 1 \rightarrow r \text{ do} \\
&\quad Y_{k0} = y_n \\
&\quad Y_{k1} = Y_{k0} + \frac{h}{2k}f(Y_{k0}) \\
&\quad \text{for } j = 2 \rightarrow 2k \text{ do} \\
&\quad \quad Y_{kj} = Y_{kj-2} + \frac{h}{k}f(Y_{kj-1}) \\
&\quad \text{end for} \\
&\quad T_{k1} = Y_{k2} \\
&\text{end for} \\
&\text{for } k = 2 \rightarrow p \text{ do} \\
&\quad \text{for } j = k \rightarrow r \text{ do} \\
&\quad \quad T_{jk} = T_{j,k-1} + \frac{T_{j,k-1} - T_{j-1,k-1}}{j^2-1} \\
&\quad \text{end for} \\
&\quad \text{end for} \\
&y_{n+1} = T_{rr} \\
&\hat{y}_{n+1} = T_{r-1,r-1}
\end{align*}
\]

\[\text{Compute second-order approximations} \]
\[\text{Initial Euler step} \]
\[\text{Midpoint steps} \]
\[\text{Extrapolate to get higher order} \]
\[\text{Aitken-Neville formula for extrapolation to order } 2k \]
\[\text{New solution value} \]
\[\text{Embedded method solution value} \]

integral of the error, or defect. Then the method steps over the same nodes again, applying a correction. This procedure is repeated until the desired accuracy is achieved.

A modification of the DC method appears in [27], in which a parameter \(\theta\) is used to adjust the dependence of the correction steps on previous iterations. The original scheme corresponds to \(\theta = 1\); by taking \(\theta \in [0, 1]\) the stability of the method can be improved. Given a fixed order of accuracy and a predictor method, the resulting DC method can be written as a Runge–Kutta method [13]. The algorithm is defined below (the values \(c_j\) denote the locations of the Chebyshev nodes) and depicted in Figure 2. For error estimation, we use the solution from the next-to-last correction iteration, whose order is one less than that of the overall method.

Algorithm 3 Explicit Euler-based deferred correction (DC-Euler)

\[
\begin{align*}
Y_{10} &= y_n \\
&\text{for } k = 1 \rightarrow p - 1 \text{ do} \\
&\quad Y_{1k} = Y_{1,k-1} + (c_{k+1} - c_k)hf(Y_{1,k-1}) \\
&\text{end for} \\
&\text{for } k = 2 \rightarrow p \text{ do} \\
&\quad Y_{k0} = y_n \\
&\quad \text{for } j = 1 \rightarrow p - 1 \text{ do} \\
&\quad \quad Y_{kj} = Y_{kj-1} + h\theta(f(Y_{kj-1}) - f(Y_{k-1,j-1})) + I_{j-1}^{p-1}(f(Y_{k-1,j-1})) \\
&\quad \text{end for} \\
&\quad y_{n+1} = Y_{p,p-1} \\
&\hat{y}_{n+1} = Y_{p-1,p-1}
\end{align*}
\]

\[\text{Compute initial prediction} \]
\[\text{Compute successive corrections} \]
\[\text{New solution value} \]
\[\text{New solution value} \]

In Algorithm 3, \(I_{j-1}^{p-1}(f(Y_{k-1,j-1}))\) represents the integral of the degree \(p - 1\) polynomial that interpolates the points \(Y_{k-1,j}\) for \(j = 1, \ldots, p - 1\), over the interval \([t_n + c_jh, t_n + c_{j+1}h]\).

The number of stages per step is
\[s = p(p - 1)\]

unless \(\theta = 0\), in which case the stages \(Y_{p,j}\) (for \(j < p - 1\)) need not be computed at all since
Figure 2: Structure of a 4th-order DC step using 3 Euler substeps. Each numbered circle represents a function evaluation, and the numbers indicate the order in which they would be performed. The black arrows represent dependencies; the grey arrows are dependencies that vanish when \( \theta = 0 \). Note that node 1 is connected to all other nodes; some of those arrows have been omitted for clarity.

\[ Y_{p,p-1} \] depends only on the \( Y_{p-1,j} \). Then the number of stages per step reduces to \((p - 1)^2 + 1\).

### 2.3 Reference Runge–Kutta methods

In this work we use the following existing Runge-Kutta pairs as benchmarks for evaluating extrapolation and deferred correction methods:

- Fourth order: the embedded formula of Merson 4(3) [16, pg. 167]
- Sixth order: the 6(5) pair of Calvo et. al. [4], which was found to be the most efficient out of those considered by Hosea and Shampine [17]
- Eighth order: the well-known Prince-Dormand 8(7) pair [31]
- Tenth order: the 10(8) pair of Curtis [6]
- Twelfth order: The 12(9) pair of Ono [30]

### 3 Theoretical measures of efficiency

Here we describe the theoretical metrics we use to evaluate the methods. Our metrics are fairly standard; a useful and thorough reference is [22]. The overarching metric for comparing methods is efficiency: the number of function evaluations required to integrate a given problem over a specified time interval to a specified accuracy. We assume that function evaluations are relatively expensive so that other arithmetic operations and overhead for things like step size selection are not significant.

The number of function evaluations is the product of the number of stages of the method and the number of steps that must be taken. The number of steps to be taken depends on the step size \( h \), which is usually determined adaptively to satisfy accuracy and stability constraints, i.e.

\[ h = \min(h_{\text{stab}}, h_{\text{acc}}) \]
where $h_{\text{stab}}, h_{\text{acc}}$ are the maximum step sizes that ensure numerical stability and prescribed accuracy, respectively. Since the cost of a step is proportional to the number of stages of the method, $s$, then a fair measure of efficiency is $h/s$. A simple observation that partially explains results in this section is as follows: extrapolation and deferred correction are straightforward approaches to creating methods that satisfy the huge numbers of order conditions for very high order Runge–Kutta methods. However, this straightforward approach comes with a cost: they use many more than the minimum necessary number of stages to achieve a particular order, leading to relatively low efficiency.

3.1 Absolute stability

The stable step size $h_{\text{stab}}$ is typically the limiting factor when a very loose error tolerance is applied. A method’s region of absolute stability (in conjunction with the spectrum of $f'$) typically dictates $h_{\text{acc}}$ \cite{16}. Figure 3 shows some absolute stability regions for 8th order methods. The poor stability of the original ($\theta = 1$) DC methods is immediately apparent; the other three methods have very similar stability regions. In fact, the Ex-Euler and Ex-midpoint methods of order $p$ both have stability polynomial

$$
\sum_{k=0}^{p} z^p\frac{p!}{p!},
$$

i.e., the degree-$p$ Taylor polynomial of the exponential function.

In order to make broad comparisons, we measure the size of the and real-axis interval that is contained in the absolute stability region. Specifically, let $S \subset \mathbb{C}$ denote the region of absolute stability; then we measure

$$
I_{\text{real}} = \max\{r \geq 0 : [-r, 0] \subset S\}
$$

Figure 4 shows stability interval sizes for Ex-Euler, Ex-Midpoint, and DC-Euler methods of orders 4-12. The deferred correction methods are considered with three different values of $\theta$. For most classes of methods, the size of the stability region grows with increasing order, but for DC methods with $\theta > 0$, the opposite behavior can be observed. Again, the two classes of extrapolation methods have identical stability regions, but their scaled stability intervals differ since they have different numbers of stages.

A fair metric for efficiency is obtained by dividing these interval sizes by the number of stages in the method. The result is shown in the lower two plots of Figure 4. Higher-order methods have smaller relative stability regions, with the DC methods being the worst (even when we take $\theta = 0$). For orders $p \leq 10$, the reference RK methods have better real stability properties.

We also examined the length of the imaginary axis interval contained in the stability region. However, for high order methods the stability region boundary lies very close to the imaginary axis, so the practical significance of this measurement is unclear. Here for simplicity we have considered only the stability region of the principal method; in the design of embedded pairs, it is important that the embedded method have a similar stability region. All the pairs considered here seem to have fairly well matched stability regions.

3.2 Accuracy efficiency

Typically, the local error is controlled by requiring that $\|y_{n+1} - \hat{y}_{n+1}\| < \epsilon$ for some tolerance $\epsilon > 0$. When the maximum stable step size does not yield sufficient accuracy, the accuracy constraint determines the step size. This is typically the case when the error tolerance is reasonably small. In theoretical analyses, the principal error norm \cite{22}

$$
C_{p+1} = \left(\sum_{k} (r_k^{(p+1)})^2\right)^{\frac{1}{2}}.
$$

(3)
Figure 3: Absolute stability regions of some 8th order pairs. The region for the principal method is filled in red, while the region for the embedded method is outlined in black.

Figure 4: Comparison of stability regions for reference methods, Euler extrapolation, midpoint extrapolation and deferred correction.

is often used as a way to compare accuracy between two methods of the same order. Here the constants $\tau_k^{(p+1)}$ are the coefficients appearing in the leading order truncation error terms. Assuming that the one-step error is proportional to $C_{p+1} h^{p+1}$ leads to a fair comparison of accuracy efficiency given by the accuracy efficiency index, introduced in [17]:

$$\eta = \frac{1}{s} \left( \frac{1}{C_{p+1}} \right)^{1/p+1}.$$  \hspace{1cm} (4)

Figure 5 plots the accuracy efficiency index for the methods under consideration. Interestingly, a ranking of methods based on this metric gives the same ordering as that based on
$I_{\text{real}}/s$.

### 3.3 Predictions

The results in this section indicate that fixed-order extrapolation and deferred correction methods are less efficient than traditional Runge–Kutta methods, at least up to order eight. At higher orders, the disadvantage of extrapolation and DC are less pronounced, but they still offer no theoretical advantage.

### 4 Performance tests

In this section we perform numerical tests, solving some initial value problems with the methods under consideration, to validate the theoretical predictions of the last section.

#### 4.1 Step size control

For step size selection, we use a standard $P$-controller [22]:

$$h_{n+1}^* = \kappa h_n \left( \frac{\epsilon}{\|\delta_{n+1}\|_{\infty}} \right)^\alpha.$$  \hfill (5)

Here $\epsilon$ is the chosen integration tolerance and $\delta_{n+1} = y_{n+1} - \hat{y}_{n+1}$. We take $\kappa = 0.9$ and $\alpha = 0.7/p$, where $p$ is the order of the embedded method. The step size is not allowed to increase or decrease too suddenly; we use [16]:

$$h_{n+1} = \min \left( \kappa_{\text{max}} h_n, \max \left( \kappa_{\text{min}} h_n, h_{n+1}^* \right) \right)$$  \hfill (6)

with $\kappa_{\text{min}} = 0.2$ and $\kappa_{\text{max}} = 5$. A step is rejected if the error estimate exceeds the tolerance; i.e., if $\|\delta_n\|_{\infty} > \epsilon.$
4.2 Test problems and results

We consider the first three-body problem from [33]:

\[ SB1 : \]
\[ y_1' = y_3, \]
\[ y_2' = y_1, \]
\[ y_3' = y_1 + 2y_4 - \mu' \frac{y_1 + \mu}{((y_1 + \mu)^2 + y_2^2)^2} - \mu \frac{y_1 - \mu'}{((y_1 - \mu')^2 + y_2^2)^2}, \]
\[ y_4' = y_2 + 2y_3 - \mu' \frac{y_2}{((y_1 + \mu)^2 + y_2^2)^2} - \mu \frac{y_2}{((y_1 - \mu')^2 + y_2^2)^2}. \]  

(7)

Here \( \mu' = 1 - \mu \), the final time is \( T = 6.192169331319639 \), and the initial values are \( y_1(0) = 1.2, y_2(0) = 0, y_3(0) = 0, y_4(0) = -1.04935759830319 \) and \( \mu = 0.0121285627653123 \).

Figure 6 plots number of function evaluations (cost) against the absolute error for this problem. The absolute error is

\[ \text{Error} = |y_N - y(T)|, \]

(8)

where \( T \) is the final time and \( y_N \) is the numerical solution at that time, while \( y(T) \) is a reference solution computed using a fine grid and the method of Bogacki & Shampine [1]. The initial step size is 0.01. In every case, the method efficiencies follow the ordering predicted by the accuracy efficiency index, and are consistent with previous studies.

Next we consider a collection of problems known as the non-stiff DETEST suite [18]. To obtain a composite measure of error over all problems in the DETEST suite, we take the geometric mean over errors at final time \( T \):

\[ \text{Mean Error} = \left( \prod_{i=1}^{m} |y_N - y(T)| \right)^{1/m}, \]

(9)

where \( m \) denotes the total number of problems considered in the geometric mean. Figure 7 shows the results, which are again consistent with predictions.

Two anomalies were noted in the results for particular test problems. First, on most problems, Ex-Euler failed completely for very tight tolerances. Second, on just a few problems the DC-Euler methods gave errors much larger than the prescribed tolerance. These are discussed in the next two subsections.

Finally, we consider the integration of a high-order PDE semi-discretization from [23]. We solve the 1D elasticity equations

\[ \epsilon_t(x,t) - \epsilon_x(x,t) = 0 \]
\[ \rho(x)u_t(x,t) - \sigma(\epsilon(x,t), x)_x = 0. \]

with nonlinear stress-strain relation

\[ \sigma(\epsilon, x) = \exp(K(x)e) - 1, \]

and a simple periodic medium composed of alternating homogeneous layers:

\[ \rho(x) = K(x) = \begin{cases} 4 & \text{if } j < x < (j + 1/2) \text{ for some integer } j, \\ 1 & \text{otherwise}. \end{cases} \]

We consider the domain \( 0 \leq x \leq 300 \), an initial Gaussian perturbation to the stress, and final time \( T = 100 \). The solution consists of two trains of emerging solitary waves; one of them is depicted in Figure 8(a). The semi-discretization is based on the WENO wave-propagation method implemented in SharpClaw [25].

Efficiency results for 8th order methods are shown in Figure 8(b). For the most part, these are consistent with the results from the smaller problems above. However, the midpoint extrapolation method performs quite poorly on this problem. The reason is not clear, but this underscores the fact that performance on particular problems can be very different from the “average” performance of a method.
Figure 6: Efficiency tests on problem SB1.
Figure 7: Efficiency tests on the non-stiff DETEST suite.

Figure 8: Solution and efficiency for methods applied to the stegoton problem.
4.3 Internal Stability

Error analysis for Runge–Kutta methods typically assumes that the stage equations are solved exactly. In practice, errors including roundoff occur. In methods with very many stages, amplification of such errors can impact the overall accuracy of the method; this phenomenon is referred to as internal instability [38]. When a Runge–Kutta method is applied to the test equation $y'(t) = \lambda y$, the full expression for the error takes the form

$$
\epsilon_{n+1} = P(z)\epsilon_n + \sum_j Q_j(z)r_j,
$$

where $\epsilon_n = y_n - y(t_n)$ is the error, $z = h\lambda$, $P(z)$ is the stability function, $Q_j(z)$ is the internal stability functions corresponding to stage $j$, and $r_j$ is the roundoff error committed in stage $j$. For an $s$-stage explicit method, the functions $Q_j$ are polynomials of degree at most $s$. These polynomials govern the amplification of roundoff errors. Internal stability is most important when considering tight error tolerances, for which the step size $h$ is typically very small. Thus we consider the internal amplification factor

$$
\mathcal{M} = \max_j |Q_j(0)|.
$$

It is reasonable to expect that a method will be unable to achieve accuracy better than $\epsilon_{\text{mach}}\mathcal{M}$, where $\epsilon_{\text{mach}}$ is the typical size of roundoff error (approximately $10^{-15}$ for double precision).

Table 1 shows the internal amplification factors of the extrapolation and reference RK methods. The Ex-Euler methods of high order have large internal amplification factors – as large as $10^5$ for the method of order 12. This leads to failure when the tolerance is set below $\epsilon_{\text{mach}}\mathcal{M}$ – the step size controller reduces $h$ in an attempt to control error, but no reduction in the overall error occurs since it is dominated by roundoff. This can be observed by inspecting the Ex-Euler curves in Figure 7 and comparing them to their respective values from Table 1. For a more detailed discussion of internal stability of extrapolation methods, see [24].

4.4 Reliability of error estimator

For some of the test problems, it was observed that the high order DC methods can drastically underestimate the local error. For example, Figure 9 shows results for DC methods applied to problem SB1. Clearly, the actual error for the high order methods is orders of magnitude larger than the prescribed tolerance. This can happen when the embedded method of order $\hat{p}$ happens to satisfy many of the conditions for order $\hat{p} + 1$ [34, 36]. Then one expects that the method may give a solution whose error is much larger than the prescribed tolerance.

We define the defectiveness index, $0 \leq \zeta \leq 1$, as the ratio of the number of order conditions of order $p$ satisfied by the embedded method to the total number of order conditions of order $p$ (here, as usual, $p$ denotes the order of the principal method). Ideally, one would like to have $\zeta = 0$, but this is perhaps unrealistic for very high order methods. If $\zeta$ is very close to unity, the error estimator is defective and we may expect failure of the error estimator in some instances.
Table 2: Index of defectiveness for the methods studied.

| Order | Reference RK | Ex-Euler | Ex-Midpoint | DC    |
|-------|--------------|----------|-------------|-------|
| 4     | 0.25         | 0.30     | 0.25        | 0.75  |
| 6     | 0.60         | 0.10     | 0.30        | 0.95  |
| 8     | 0.86         | 0.04     | 0.38        | 0.99  |
| 10    | 0.91         | 0.04     | 0.45        | 0.999 |
| 12    | 0.89         | 0.04     | 0.51        | 0.9998|

Table 2 shows the index of defectiveness for each method. It is clear that the DC methods have $\zeta \approx 1$; in fact, the error estimator in each case satisfies all but 1 of the order $p$ conditions. Clearly, our natural but simplified error control approach is insufficient in general, and more extensive error control techniques, such as those proposed in [10], should be employed. It is also notable that, except for Ex-Euler, all of the higher order methods have $\zeta$ far from zero.

We also tested DC methods with uniformly spaced nodes (instead of Chebyshev nodes). Failure of the error estimator was observed even more frequently with those methods. On the other hand, we note that this failure did not have a substantial effect on the results shown in Figure 7 since for most problems the DC methods achieve an error similar to that of other methods.
5 Concurrency

In view of an implementation on parallel computers, extrapolation methods (as opposed to RKp methods or multistep methods) have an important distinguishing feature: the rows can be computed independently.

(P. Deuflhard, 1985)

If a Runge-Kutta method includes stages that are mutually independent, then those stages may be computed concurrently [20]. In this section we investigate theoretically achievable parallel speedup and efficiency of extrapolation and deferred correction methods. Our goal is to determine hardware- and problem-independent upper bounds based purely on algorithmic concerns. We do not attempt to account for machine-specific overhead or communication, although the simple parallel tests in Section 5.3 suggest that the bounds we give are realistically achievable for at least some classes of moderate-sized problems. Previous works that have considered concurrency in explicit extrapolation and deferred correction methods include [35, 37, 32, 2, 14, 11, 21, 28, 29].

5.1 Computational model and speedup

As in the serial case, our computational is based on the assumption that evaluation of \( f \) is sufficiently expensive so that all other operations (e.g., arithmetic, step size selection) are negligible by comparison.

Typically, stage \( y_j \) of an explicit Runge–Kutta method depends on all the previous stages \( y_1, y_2, \ldots, y_{j-1} \). However, if \( y_j \) does not depend on \( y_{j-1} \), then these two stages may be computed simultaneously on a parallel computer. More generally, by interpreting the incidence matrix of \( A \) as the adjacency matrix of a directed graph \( G(A) \), one can determine precisely which stages may be computed concurrently and how much speedup may be achieved. For extrapolation methods, the computation of each \( T_{1k1} \) may be performed independently in parallel [9], as depicted in Figure 10. Unlike some previous authors, we do not consider parallel implementation of the extrapolation process (i.e., the second loop in Algorithm 1) since it does not include any evaluations of \( f \) (so our computational model assumes its cost is negligible anyway).

For the deferred correction methods we consider, parallel computation is advantageous only if \( \theta = 0 \); the resulting parallel algorithm is depicted in Figure 11. A different approach to parallelism in DC methods is taken by the RIDC method [5]; see also [14]. Deferred correction has also been combined with the parareal algorithm to achieve parallel speedup [29, 11].

We define the minimum number of sequential stages \( s_{seq} \) as the minimum number of sequential function evaluations that must be made when parallelism is taken into account. To make this more precise, let us label each node in the graph \( G(A) \) by the index of the stage it corresponds to, with the node corresponding to \( y_{n+1} \) labeled \( s + 1 \). Then

\[ s_{seq} = \max_j \{ \text{path length from node } 1 \text{ to node } s + 1 \} . \]

The quantity \( s_{seq} \) represents the minimum time required to take one step with a given method on a parallel computer, in units of the cost of a single derivative evaluation. For instance, the maximum path length for the method shown in Figure 10 is equal to 4; for the method in Figure 11 it is 6. The maximum potential parallel speedup is

\[ S = s / s_{seq} . \]

The minimum number of processes required to achieve speedup \( S \) is denoted by \( P \) (equivalently, \( P \) is the maximum number of processes that can usefully be employed by the method). Finally,
Figure 10: Exploiting concurrency in an Euler extrapolation step using 2 processes. The blue circles with broken border would be computed by process 1 and the red circles with solid border would be computed by process 2. Observe that only $s_{seq} = 4$ sequential function evaluations are required for each process, as opposed to the $s = 7$ sequential evaluations required in serial.

Figure 11: Exploiting concurrency in a 4th-order DC step using 3 Euler substeps and 3 processes, with $\theta = 0$. The color and border of each circle indicate which process would compute it. Observe that only $s_{seq} = 6$ sequential function evaluations are required for each process, as opposed to the $s = 10$ sequential evaluations required in serial ($12$ in serial when $\theta \neq 0$). Note that node 1 is connected to all other nodes; some of those arrows have been omitted for clarity. Note that more synchronization is required than for an extrapolation step.
Table 3: Parallel implementation properties of extrapolation and deferred correction methods. \(s\): number of stages; \(s_{\text{seq}}\): number of sequentially dependent stages; \(S = s/s_{\text{seq}}\): optimal speedup; \(P\): number of processes required to achieve optimal speedup; \(E = S/P\): parallel efficiency.

| Method          | \(s\)   | \(s_{\text{seq}}\) | \(S\)   | \(P\)   | \(E\)   |
|-----------------|---------|---------------------|---------|---------|---------|
| Ex-Euler        | \(\frac{p^2 - p + 2}{2}\) | \(p\)                | \(\frac{p^2 - p + 2}{2p}\) | \(\lceil \frac{p}{2} \rceil\) | \(\frac{p^2 - p + 2}{2p(\frac{p}{2})^2}\) |
| Ex-Midpoint     | \(\frac{p^2 + 4}{4}\)          | \(p\)                | \(\frac{p^2 + 4}{4p}\)     | \(\lceil \frac{p + 2}{4} \rceil\) | \(\frac{p^2 + 4}{4p(\frac{p + 2}{4})^2}\) |
| DC-Euler, \(\theta = 0\) | \((p - 1)^2 + 1\)     | \(2(p - 1)\)          | \((p - 1)^2 + 1\) \(2(p - 1)\) | \(p - 1\) | \((p - 1)^2 + 1\) \(2(p - 1)\) |
| DC-Euler, \(\theta \neq 0\) | \(p(p - 1)\)            | \(p(p - 1)\)          | \(1\)                | \(1\)                | \(-\)                |

let \(E\) denote the theoretical parallel efficiency (here we use the term in the sense that is common in the parallel computing literature) that could be achieved by spreading the computation over \(P\) processes:

\[
E = \frac{s}{P s_{\text{seq}}} = \frac{S}{P}.
\]

Note that \(E\) is an upper bound on the achievable parallel efficiency; it accounts only for inefficiencies due to load imbalancing. It does not, of course, account for additional implementation-dependent losses in efficiency due to overhead or communication.

Table 3 shows the parallel algorithmic properties of fixed-order extrapolation and deferred correction methods. Note that for deferred correction methods with \(\theta \neq 0\), we have \(s_{\text{seq}} = s\), i.e., no parallel computation of stages is possible.

To our knowledge, no parallel implementation has been made of the deferred correction methods we consider here. For parallel implementation of a revisionist DC method, see [5].

5.2 Accuracy and stability metrics

In order to determine idealized accuracy and stability efficiency measures, we take the speedup factor \(s/s_{\text{seq}}\) into account. In other words, we consider

\[
\eta = \frac{s}{s_{\text{seq}}} = \left( \frac{1}{C_{p+1}} \right)^{1/p+1}.
\]

as a measure of accuracy efficiency. A similar scaling could be used to study stability efficiency of parallel implementations. We stress that in this context efficiency relates to the number of function evaluations required to advance to a given time, and is not related to the usual concept of parallel efficiency.

Figure 12 shows the accuracy efficiency, rescaled by the speedup factor, versus order accuracy. Comparing with Figure 5, we see a very different picture for methods of order 8 and above. Extrapolation methods are the most efficient, while the reference RK methods give the weakest showing – due to their more limited potential for parallelism.

5.3 Parallel speedup

Our theoretical analysis in Section 5.1 did not account for machine-dependent inefficiencies due to communication and overhead. Development and testing of a tuned parallel extrapolation or deferred correction code is beyond the scope of this paper, but in this section we run a simple example to demonstrate that it is possible in practice to achieve speedups like those listed in Table 3. We focus on speedup over efficiency with an eye to providing efficient black-box parallel ODE integrators for multicore machines, noting that the number of available cores is often more than can be advantageously used by the methods considered.

17
Previous studies have implemented explicit extrapolation methods in parallel and achieved efficiencies of up to about 80% [19, 21, 28]. As those studies were conducted about twenty years ago, it is not clear that their conclusions are relevant to current hardware.

In order to test the achievable parallel speedup, we took the code ODEX [16], downloaded from http://www.unige.ch/~hairer/software.html. With the objective of providing a testbed for our analysis, we modified the code as follows:

- Added an option to hard code the order of accuracy (disabling adaptive order selection)
- Inserted an OMP PARALLEL pragma with dynamic scheduling around the extrapolation loop
- Removed the smoothing step

We then ran the code for various values of $p$ and with various numbers of threads, solving the equations of motion for an $N$-body gravitational problem with 400 bodies. The tests were run on a workstation with two 2.66 Ghz quad-core Intel Xeon processors.

Figure 14(a) shows the achieved speedup for $p = 6, 10, 14, 18$; results for other orders are similar. The dotted lines show the theoretical maximum speedup $S = (p^2 + 4)/(4p)$ based on our earlier analysis. Two important observations are in order.

First, using $p/2$ threads, the measured speedup is very close to the theoretical maximum. This is very encouraging, as it validates the theoretical analysis and indicates that it is relevant for practical problems of interest. By using larger numbers of threads, no further speedup is observed.

Second, when using the theoretically optimal number of threads $P = \lceil p/2 \rceil$, the measured speedup is substantially less than the theoretical value. We suspected this to be an effect of inefficiencies in OpenMP’s automated dynamic load balancing. To overcome this, we modified the code so that both $T_{k,1}$ and $T_{r-k,1}$ are computed in a single loop iteration. Results computed with this modification are shown in Figure 14(b) and Table 4. A remarkably good agreement with the theory is observed: the speedup achieved is very close to $S$ when the number of
Figure 13: Efficiency for the whole IVP test suite based on sequential derivative evaluations.
Figure 14: Measured speedup of the midpoint extrapolation code ODEX on a 400-body gravitation problem by insertion of a single OMP parallel pragma in the code. The ratio of runtime with multiple threads to runtime using a single thread is plotted.

Table 4: Runtime, speedup and efficiency of manually load-balanced runs of the modified ODEX code with $P$ threads. The observed speedup (and efficiency) are close to the theoretically optimal values ($S$ and $E$).

| Order ($p$) | $P$ | Runtime 1 thread | $P$ threads | Max. speedup Theory ($S$) | Observed | Parallel Efficiency Theory ($E$) | Observed |
|------------|-----|------------------|-------------|--------------------------|----------|-------------------------------|----------|
| 6          | 2   | 13.140           | 7.977       | 1.67                     | 1.65     | 0.83                          | 0.82     |
| 10         | 3   | 17.370           | 6.770       | 2.60                     | 2.57     | 0.87                          | 0.86     |
| 14         | 4   | 19.508           | 5.573       | 3.57                     | 3.50     | 0.89                          | 0.88     |
| 18         | 5   | 25.876           | 5.827       | 4.56                     | 4.44     | 0.91                          | 0.89     |

threads is $P$. This load balancing scheme is optimal when using on the optimal number of threads $P$, but not necessarily for smaller numbers of threads; indeed, the automatic dynamic scheduler approach gives better speedup for some intermediate numbers of threads. Of course, with additional work a more complex optimal load-balancing for all possible numbers of threads could be implemented.

6 Discussion

The most significant new conclusions from the present study are:

1. **Extrapolation and deferred correction methods are not advantageous when compared to standard RK methods and concurrency is neglected.** Consistent with past studies [33, 17, 26], our results in Section 3 show that explicit extrapolation methods have no significant advantage when compared with high order Runge–Kutta methods, although midpoint extrapolation is at least competitive at high orders. Deferred correction methods are even less efficient.

2. **Parallel extrapolation and DC methods of high order may outperform traditional RK methods for problems with an expensive right hand side.** Our analysis in Section 5 based on the parallel accuracy efficiency measure (11), shows that when concurrent stage computation is taken into account, extrapolation is more efficient than traditional RK formulas at order eight and higher (even when extrapolation is imple-
mented as a fixed-order method). Surprisingly, extrapolation of the explicit Euler method is competitive with extrapolation of the midpoint rule in light of its greater potential concurrency.

3. **The serial and parallel model predictions are validated by numerical examples.** Experiments using the non-stiff DETEST suite in Section 4.2 confirm the predictions above. Of course one should not assume that similar results will hold for every class of problems, but the agreement of theory and empirical evidence seems to indicate a generally consistent behavior.

4. **Near-optimal speedup can be achieved in practice with simple modifications of an existing code.** As demonstrated in Section 5.3 speedup near the theoretically-predicted values can be obtained on a modern shared-memory workstation without very much effort.

5. **High order Euler extrapolation methods suffer from dramatic amplification of roundoff errors.** This leads to the loss of several digits of accuracy (and failure of the automatic error control) for very high order methods, and is observed in practice on most problems.

6. **Deferred correction methods with \( \theta = 0 \) can be efficiently parallelized.** Unlike other approaches to parallel deferred correction, the proposed method does not require staggered computation of \( f \) at different nodes. However, the efficiency of these methods is still inferior to that of extrapolation methods both in serial and in parallel.

This study is intended to provide a broadly useful characterization of the properties of extrapolation and deferred correction methods. Of course, no study like this can be exhaustive. Our approach handicaps extrapolation and deferred correction methods by fixing the order throughout each computation; practical implementations are order-adaptive and should achieve somewhat better efficiency. We have investigated only the most generic versions of each class of methods; other approaches (e.g., using higher order building blocks or exploiting concurrency in different ways) may give significantly different results. Such approaches could be evaluated using the same kind of analysis employed here. Finally, our parallel computational model is valid only when evaluation of \( f \) is relatively expensive – but that is when efficiency and concurrency are of most interest.

The theoretical and preliminary experimental results we have presented suggest that a carefully-designed parallel code based on midpoint extrapolation could be very efficient. Such a practical implementation is the subject of current efforts.

**References**

[1] P Bogacki and Lawrence F Shampine. An efficient Runge-Kutta \((4, 5)\) pair. *Computers & Mathematics with Applications*, 32(6):15–28, 1996.

[2] Kevin Burrage. *Parallel and sequential methods for ordinary differential equations*. Clarendon Press, 1995.

[3] J. C. Butcher. *Numerical Methods for Ordinary Differential Equations*. Wiley, 2 edition, 2008.

[4] M. Calvo, J.I. Montijano, and L. Randez. A new embedded pair of Runge-Kutta formulas of orders 5 and 6. *Computers & Mathematics with Applications*, 20(1):15–24, 1990.

[5] Andrew J. Christlieb, Colin B. Macdonald, and Benjamin W. Ong. Parallel high-order integrators. *SIAM Journal on Scientific Computing*, 32(2):818–835, Jan 2010.

[6] A. R. Curtis. High-order explicit Runge-Kutta formulae, their uses, and limitations. *IMA Journal of Applied Mathematics*, 16(1):3552, 1975.

[7] James W. Daniel, Victor Pereyra, and Larry L. Schumaker. Iterated deferred corrections for initial value problems. *Acta Cientifica Venezolana*, 19:128135, 1968.
[8] P. Deuflhard. Order and stepsize control in extrapolation methods. Numerische Mathematik, 41(3):399–422, 1983.

[9] P. Deuflhard. Recent progress in extrapolation methods for ordinary differential equations. SIAM Review, 27(4):505–535, December 1985.

[10] A. Dutt, L. Greengard, and Vladimir Rokhlin. Spectral deferred correction methods for ordinary differential equations. BIT Numerical Mathematics, 40:241–266, 2000.

[11] Matthew Emmett and Michael Minion. Toward an efficient parallel in time method for partial differential equations. Comm. App. Math. and Comp. Sci, 7(1):105–132, 2012.

[12] T Feagin. High-order explicit Runge-Kutta methods using M-symmetry. Neural Parallel and Scientific Computations, 20(3):437, 2012.

[13] Sigal Gottlieb, David I. Ketcheson, and Chi-Wang Shu. High Order Strong Stability Preserving Time Discretizations. Journal of Scientific Computing, 38(3):251–289, 2009.

[14] D Guibert and D Tromeur-Dervout. Cyclic distribution of pipelined parallel deferred correction method for ODE/DAE. In Parallel Computational Fluid Dynamics 2007, pages 171–178. Springer, 2009.

[15] Bertil Gustafsson and Wendy Kress. Deferred correction methods for initial value problems. BIT Numerical Mathematics, 41(5):986–995, 2001.

[16] Ernst Hairer, Syvert P. Nørsett, and G. Wanner. Solving ordinary differential equations I: Nonstiff Problems. Springer Series in Computational Mathematics. Springer, Berlin, second edition, 1993.

[17] M.E. Hosea and L.F. Shampine. Efficiency comparisons of methods for integrating ODEs. Computers & Mathematics with Applications, 28(6):45–55, September 1994.

[18] T. E. Hull, W. H. Enright, B. M. Fellen, and A. E. Sedgwick. Comparing Numerical Methods for Ordinary Differential Equations. SIAM Journal on Numerical Analysis, 9(4):603–637, December 1972.

[19] Takashi Ito and Toshio Fukushima. Parallelized extrapolation method and its application to the orbital dynamics. The Astronomical Journal, 114:1260, Sep 1997.

[20] K. R. Jackson and S. P. Nørsett. The potential for parallelism in Runge-Kutta methods . part 1 : Rk formulas in standard form. 32(1):49–82, 2012.

[21] M. Kappeller, M. Kiehl, M. Perzl, and M. Lenke. Optimized extrapolation methods for parallel solution of IVPs on different computer architectures. Applied Mathematics and Computation, 77(2-3):301–315, Jul 1996.

[22] C. A. Kennedy, M. H. Carpenter, and R Michael Lewis. Low-storage, explicit Runge-Kutta schemes for the compressible NavierStokes equations. Applied Numerical Mathematics, 35(3):177–219, November 2000.

[23] David I. Ketcheson and Randall J. LeVeque. Shock dynamics in layered periodic media. Communications in Mathematical Sciences, 10(3):859–874, 2012.

[24] David I Ketcheson, Lajos Lóczy, and Matteo Parsani. Internal error propagation in explicit Runge–Kutta discretization of PDEs. In preparation.

[25] DI Ketcheson, Matteo Parsani, and RJ LeVeque. High-order Wave Propagation Algorithms for Hyperbolic Systems. SIAM Journal on Scientific Computing, 35(1):A351–A377, 2013.

[26] Fred T. Krogh. Stepsize selection for ordinary differential equations. ACM Transactions on Mathematical Software, 37(2):1–11, April 2010.

[27] Y Liu, CW Shu, and M Zhang. Strong stability preserving property of the deferred correction time discretization. Journal of Computational Mathematics, 2007.

[28] L Lustman, B Neta, and W Gragg. Solution of ordinary differential initial value problems on an intel hypercube. Computers & Mathematics with Applications, 23(10):65–72, 1992.
[29] Michael Minion. A hybrid parareal spectral deferred corrections method. *Communications in Applied Mathematics and Computational Science*, 5:265–301, 2010.

[30] Hiroshi Ono. On the 25 stage 12th order explicit Runge-Kutta method. *Transactions-Japan Society for Industrial and Applied Mathematics*, 16(3):177, 2006.

[31] P.J. Prince and J.R. Dormand. High order embedded Runge-Kutta formulae. *Journal of Computational and Applied Mathematics*, 7(1):67–75, March 1981.

[32] Thomas Rauber and Gudula Rünger. Load balancing schemes for extrapolation methods. *Concurrency: Practice and Experience*, 9(3):181–202, 1997.

[33] L. F. Shampine and L. S. Baca. Fixed versus variable order Runge-Kutta. *ACM Transactions on Mathematical Software*, 12(1):1–23, May 1986.

[34] LF Shampire. Quadrature and Runge-Kutta formulas. *Applied Mathematics and Computation*, 2(2):161–171, 1976.

[35] HH Simonsen. *Extrapolation methods for ODE’s: continuous approximations, a parallel approach*. PhD thesis, University of Trondheim, Norway, 1990.

[36] Ch Tsitouras and SN Papakostas. Cheap error estimation for Runge–Kutta methods. *SIAM Journal on Scientific Computing*, 20(6):2067–2088, 1999.

[37] P. J. van der Houwen and B. P. Sommeijer. Parallel ODE solvers. *ACM SIGARCH Computer Architecture News*, 18(3):71–81, September 1990.

[38] J. G. Verwer, W. H. Hundsdorfer, and B. P. Sommeijer. Convergence properties of the Runge-Kutta-Chebyshev method. *Numerische Mathematik*, 57(1):157178, 1990.