Introduction to parallel in time algorithms

Jemma Shipton

January 2023

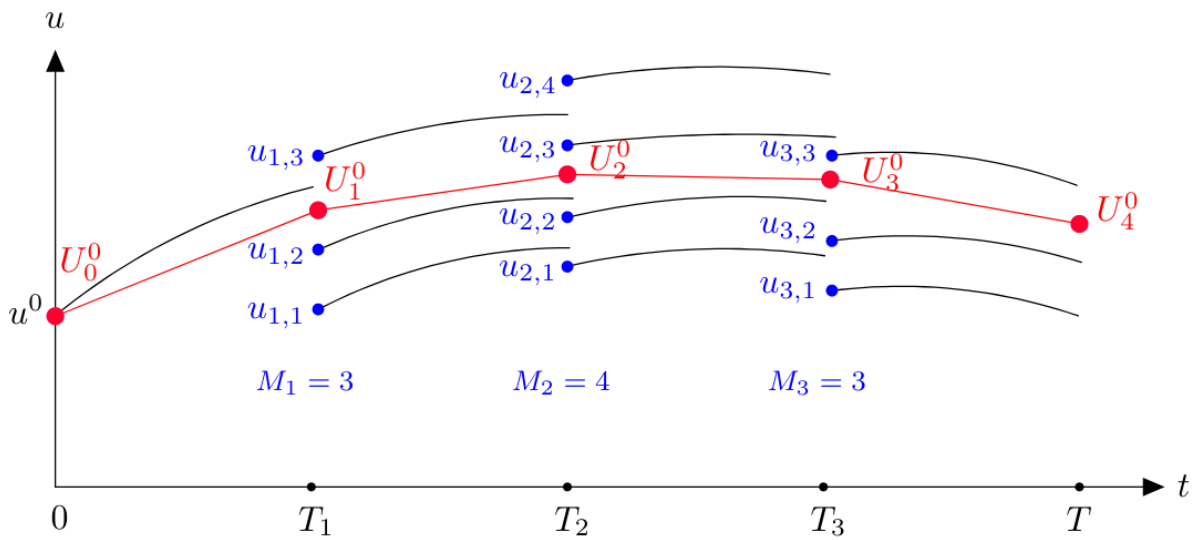
For the last 20 years, one has tried to speed up numerical computation mainly by providing ever faster computers. Today, as it appears that one is getting closer to the maximal speed of electronic components, emphasis is put on allowing operations to be performed in parallel. In the near future, much of numerical analysis will have to be recast in a more ‘parallel’ form.

Jörg Nievergelt

Parallel methods for integrating ordinary differential equations

1964

Nievergelt's Idea



- partition domain in to subintervals
- compute a rough prediction
- for a number M of starting points in the neighbourhood of the U_n, compute accurate trajectories in parallel
- compute an interpolated update

Nievergelt's Idea

'The integration methods introduced in this paper are to be regarded as tentative examples of a much wider class of numerical procedures in which parallelism is introduced at the expense of redundancy of computation. As such, their merits lie not so much in their usefulness as numerical algorithms as in their potential as prototypes of better methods based on the same principle. It is believed that more general and improved versions of these methods will be of great importance when computers capable of executing many computations in parallel become available.'

Parareal

To solve the initial value problem

$$\begin{aligned} u_t &= f(t,u), \quad t \in (0,T], \\ u(0) &= u^0 \end{aligned}$$

using the parareal algorithm, we require two integrators or propagators:

1. $G(t,u)$ is a coarse approximation
2. $F(t,u)$ is a more accurate, or fine, approximation

Parareal

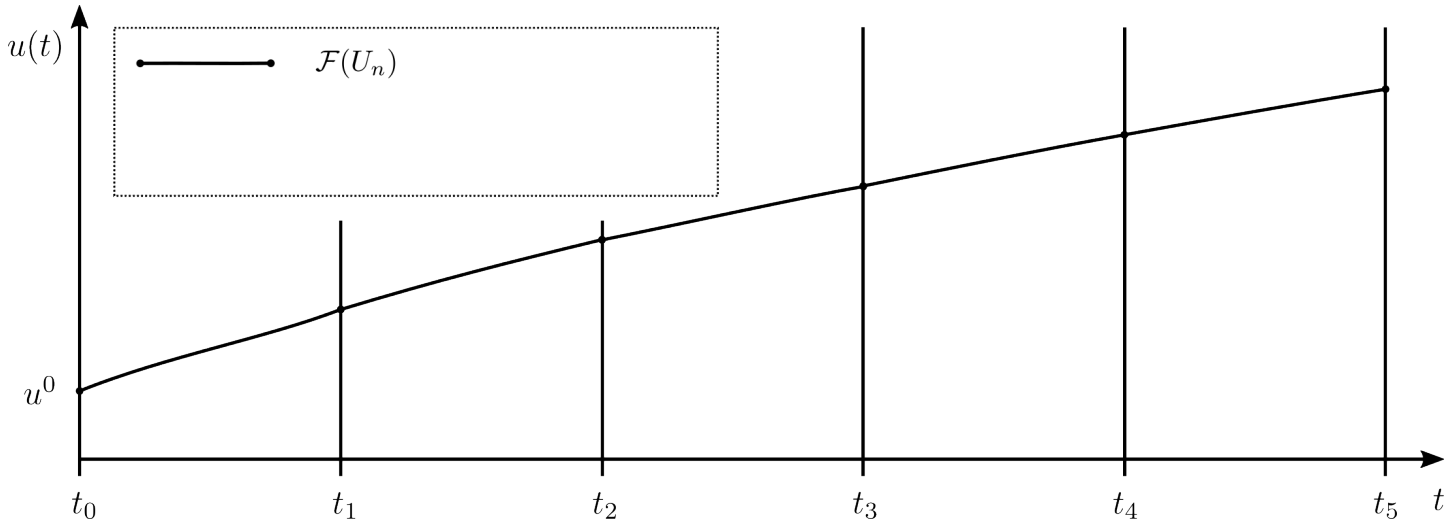
We partition the time domain into intervals (time-slices, or chunks) and use the coarse approximation G to give us an initial condition for F on each interval.

The more accurate (i.e. *expensive!*) approximation can now be computed in parallel for all slices at once, although obviously the later time slices do not have an accurate initial condition.

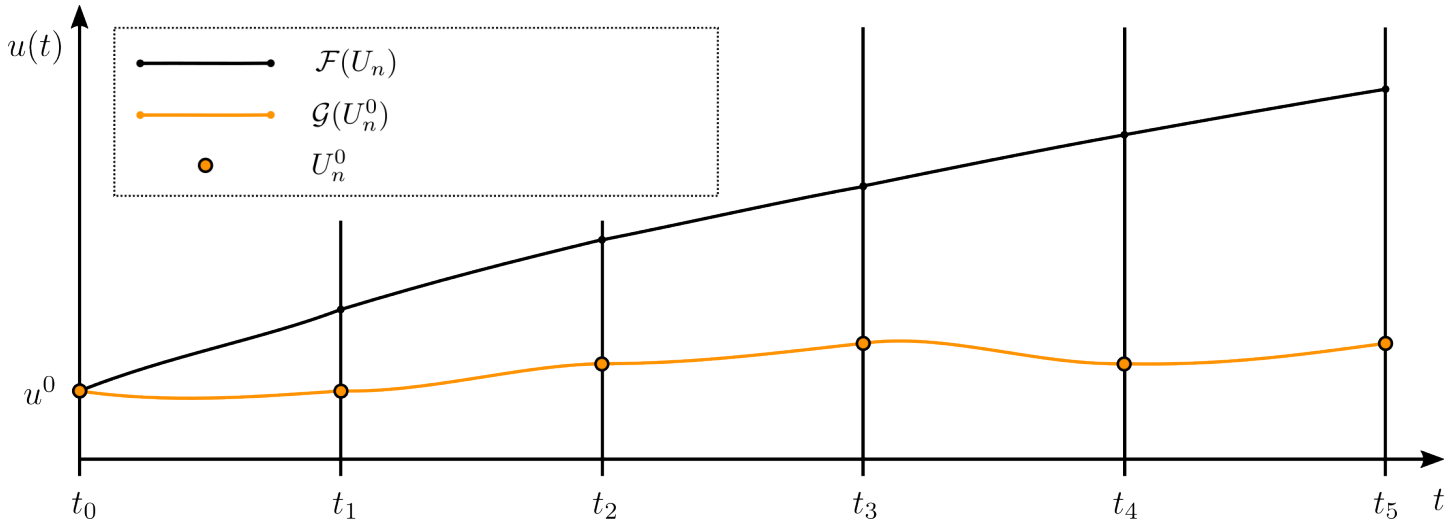
We iterate, correcting the initial condition on each interval:

$$u_0^{k+1} = u^0$$
$$u_{n+1}^{k+1} = F(u_n^k) + G(u_n^{k+1}) - G(u_n^k)$$

Changes to algorithms driven by parallelism

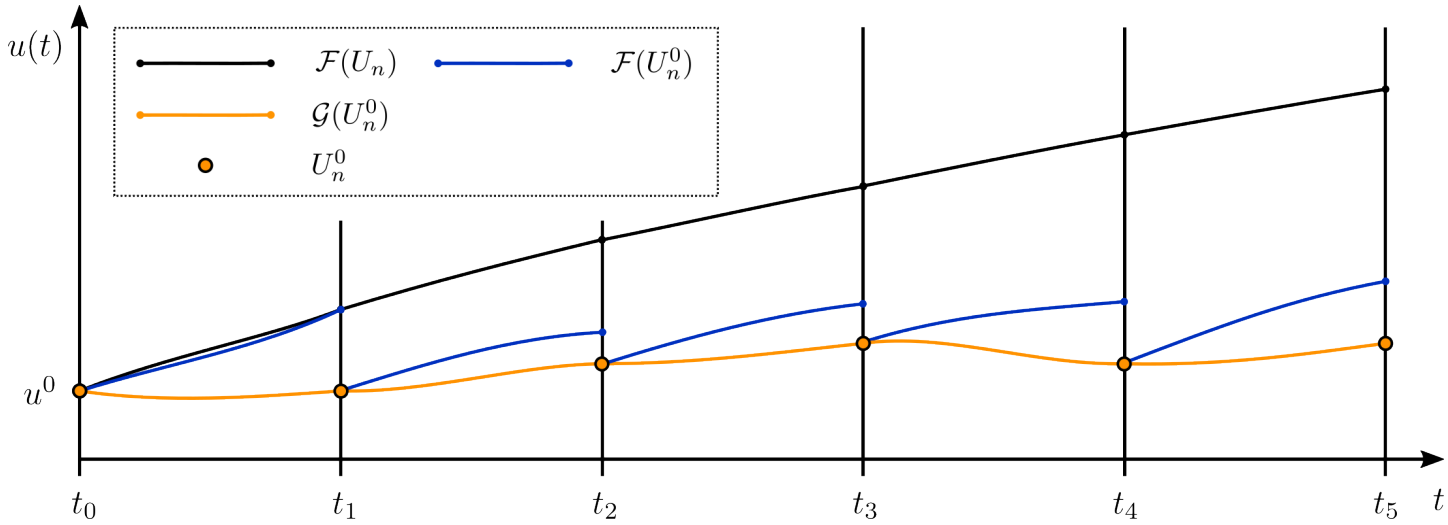


Changes to algorithms driven by parallelism



We partition the time domain into intervals and use G to give us an initial condition for F on each interval.

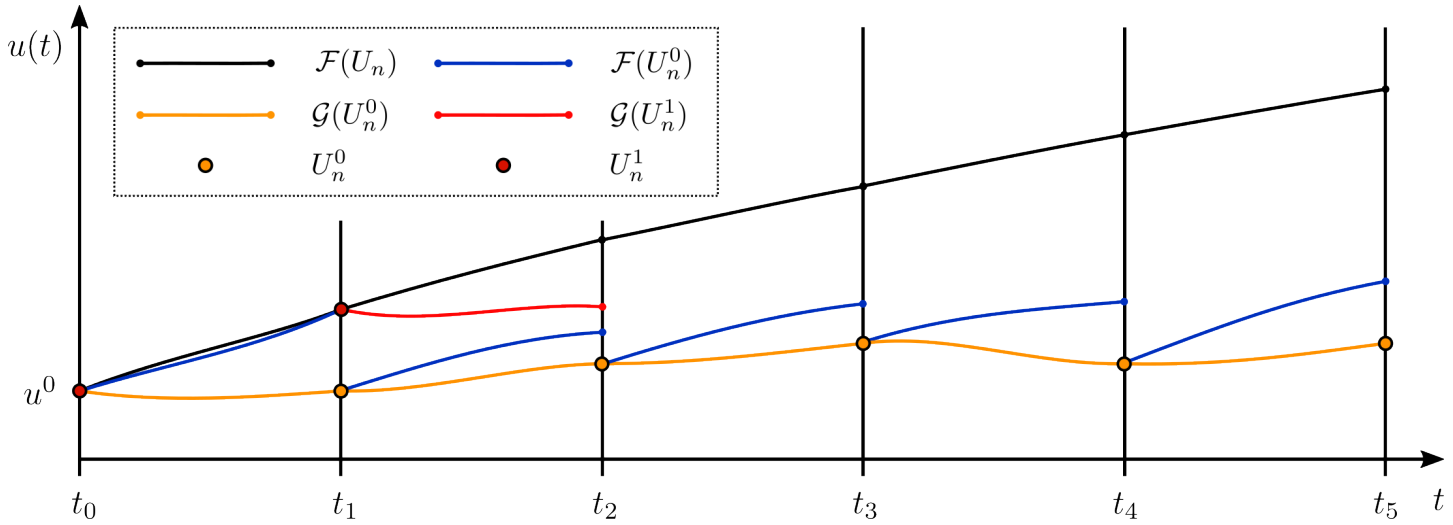
Changes to algorithms driven by parallelism



We partition the time domain into intervals and use G to give us an initial condition for F on each interval.

F can be computed in parallel, although the later time intervals do not have an accurate initial condition.

Changes to algorithms driven by parallelism



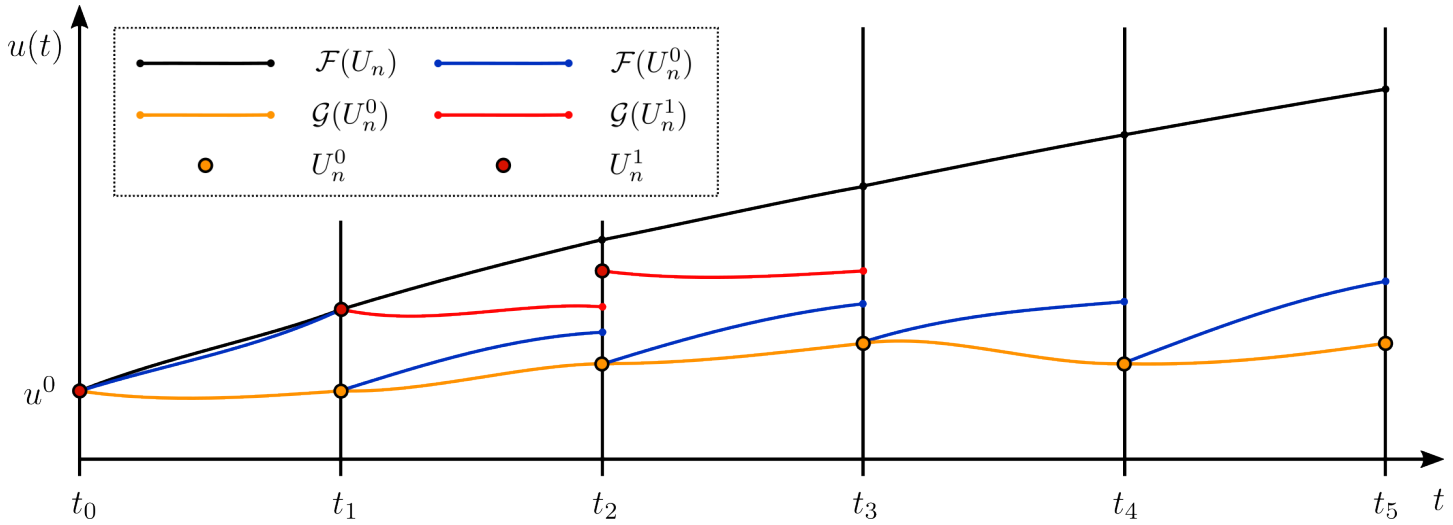
We partition the time domain into intervals and use G to give us an initial condition for F on each interval.

F can be computed in parallel, although the later time intervals do not have an accurate initial condition.

We iterate, correcting the initial condition on each interval:

$$u_0^{k+1} = u^0$$
$$u_{n+1}^{k+1} = F(u_n^k) + G(u_n^{k+1}) - G(u_n^k)$$

Changes to algorithms driven by parallelism



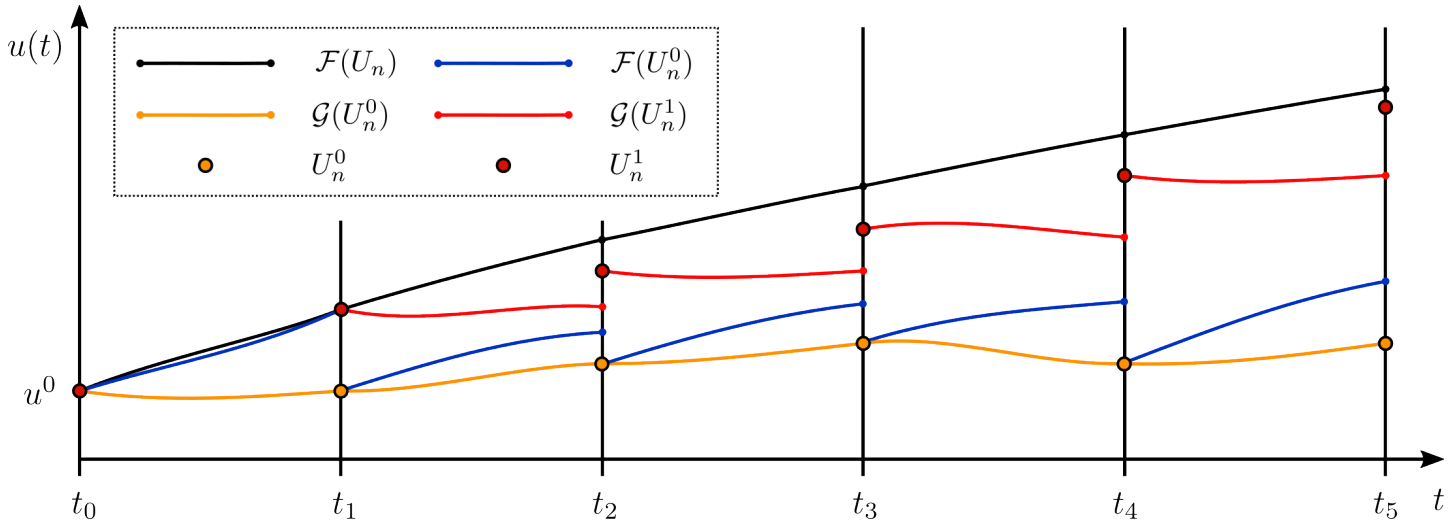
We partition the time domain into intervals and use G to give us an initial condition for F on each interval.

F can be computed in parallel, although the later time intervals do not have an accurate initial condition.

We iterate, correcting the initial condition on each interval:

$$u_0^{k+1} = u^0$$
$$u_{n+1}^{k+1} = F(u_n^k) + G(u_n^{k+1}) - G(u_n^k)$$

Changes to algorithms driven by parallelism



We partition the time domain into intervals and use G to give us an initial condition for F on each interval.

F can be computed in parallel, although the later time intervals do not have an accurate initial condition.

We iterate, correcting the initial condition on each interval:

$$u_0^{k+1} = u^0$$
$$u_{n+1}^{k+1} = F(u_n^k) + G(u_n^{k+1}) - G(u_n^k)$$

Parareal

Assume:

1. All time slices are the same length.
2. Both the coarse timestep dt_c and the fine timestep dt_f remain constant over the integration.
3. Each time slice has exactly N_c steps of the coarse integrator and N_f steps of the fine integrator.
4. The cost of a coarse timestep τ_c and the cost of a fine timestep τ_f remain constant.

We can model the cost of integrating over P time slices for K iterations as:

$$c_{\text{parareal}} = (K+1)PN_c\tau_c + KN_f\tau_f.$$

The speedup is then:

$$\frac{c_{\text{fine}}}{c_{\text{parareal}}} = \frac{1}{(K+1)\frac{N_c}{N_f}\frac{\tau_c}{\tau_f} + \frac{K}{P}} \leq \min\left(\frac{N_f\tau_f}{N_c\tau_c}, \frac{P}{K}\right).$$

Parareal

Speedup

$$\frac{c_{\text{fine}}}{c_{\text{parareal}}} = \frac{1}{(K+1)\frac{N_c}{N_f}\frac{\tau_c}{\tau_f} + \frac{K}{P}} \leq \min\left(\frac{N_f \tau_f}{N_c \tau_c}, \frac{P}{K}\right).$$

Trade off between:

- choosing the coarse method to be cheap or use a large timestep to keep the first bound large,
- requiring a low number of iterations to keep the second bound large.

How can we construct a coarse propagator?

Generally can't just take a bigger timestep due to stability.

- reduced order models?
- averaged equations?
- data driven methods?

Spectral Deferred Corrections

- Compute an approximate solution $\tilde{y}(t)$
- Solve the error equation to obtain δ
- Update approximate solution to $\tilde{y} + \delta$
- Gain an order of accuracy!

Revisionist Integral Deferred Correction (RIDC)

This method can be made parallel by amending quadrature term...

Spectral Deferred Corrections

Consider a scalar ODE defined on a time interval $t \in [t_0, t_N]$:

$$y'(t) = f(y, t)$$
$$y(t_0) = y_0$$

Recast into integral form:

$$y(t) = y_0 + \int_{t_0}^{t} f(y, \tau) d\tau$$

Denote the numerical solution $\tilde{y}(t)$ and define the **error** δ and the **residual** ϵ as:

$$\delta(t) = y(t) - \tilde{y}(t)$$
$$\epsilon(t) = y_0 + \int_{t_0}^{t} f(\tilde{y}, \tau) d\tau - \tilde{y}(t)$$

Spectral Deferred Corrections

Manipulating these equations gives an equation for δ:

$$\delta(t) = \epsilon(t) + \int_{t_0}^{t} f(y,\tau) - f(\tilde{y},\tau)d\tau$$

Applying the the forward Euler method to the Picard equation for δ gives

$$\delta^{n+1} = \delta^n + \Delta t\big(G(\tilde{y}^n,\delta^n,t^n) + \epsilon^{n+1} - \epsilon^n\big),$$

where

$$G(\tilde{y},\delta,t) = f(\tilde{y}+\delta,t) - f(\tilde{y},t),$$

and

$$\epsilon^{n+1} - \epsilon^n = \int_{t^n}^{t^{n+1}} f(\tilde{y},\tau)d\tau - \tilde{y}^{n+1} + \tilde{y}^n.$$

Spectral Deferred Corrections

$$\epsilon^{n+1} - \epsilon^n = \int_{t^n}^{t^{n+1}} f(\tilde{y}, \tau) d\tau - \tilde{y}^{n+1} + \tilde{y}^n.$$

We approximate the integral as a sum of terms comprised of weights multiplied by function evaluations, that is

$$\int_{t_n}^{t_{n+1}} f(\tilde{y}, \tau) d\tau \approx \sum_{m=1}^{M} w_{m,n} f(\tilde{y}_m, t_m),$$

where $t_m \in [t_n, t_{n+1}] \, \forall m \in [1, M]$. The weights $\{w_{m,n}\}$ are calculated using the Lagrange interpolating polynomial l_m and we choose the $\{t_m\}$ to be equispaced.

Spectral Deferred Corrections

Solving $y_t = f(y)$ with implicit Euler as the base method gives SDC iterations of the form:

$$y_m^{k+1} = y_{m-1}^{k+1} + \Delta\tau_m\big(f(y_m^{k+1}) - f(y_m^k)\big) + \sum_{j=1}^{M} s_{m,j} f(y_m^k), \quad m = 1, ..., M$$

Solving $y_t = f_s(y) + f_f(y)$ with IMEX Euler as the base method gives SDC iterations of the form:

$$y_m^{k+1} = y_{m-1}^{k+1} + \Delta\tau_m\big(f_f(y_m^{k+1}) - f_f(y_m^k) + f_s(y_{m-1}^{k+1}) - f_s(y_{m-1}^k)\big) + \sum_{j=1}^{M} s_{m,j} f(y_m^k), \quad m = 1, ..., M$$

Revisionist Integral Deferred Correction

This method relies on the idea of pipelining the corrections rather than waiting to compute the corrections in each group of nodes before continuing.

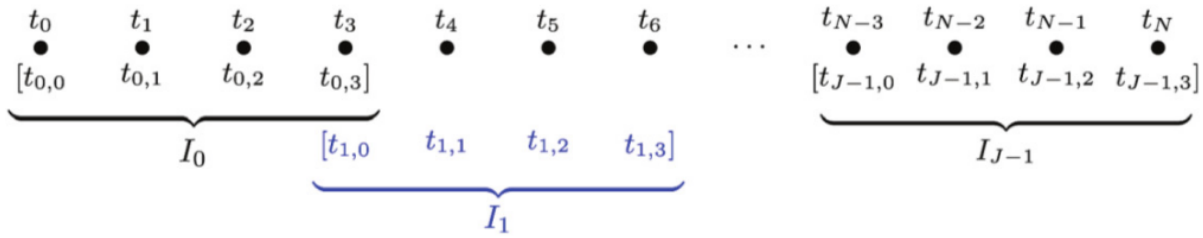


Figure 5.3: Classical application of integral deferred correction, picture taken from [17]

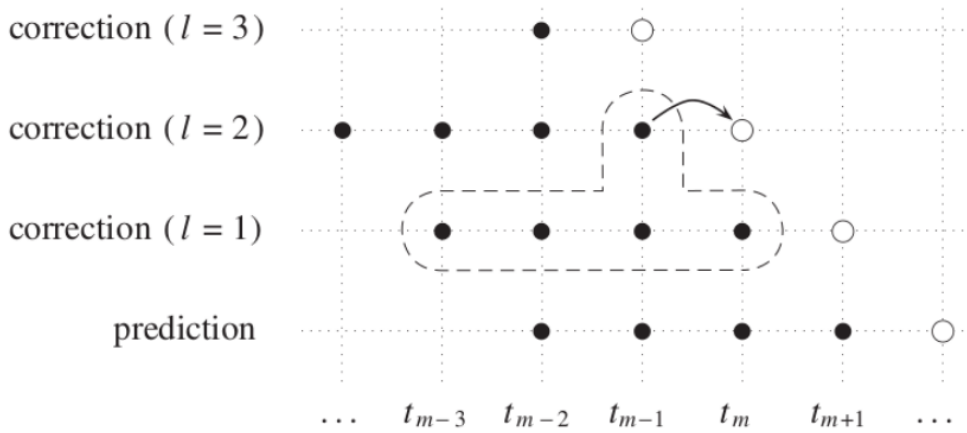


Figure 5.4: RIDC way to compute integral deferred correction type methods in a pipelined way, figure taken from [17]