Parallelization in Time

Mark Maienschein-Cline
Department of Chemistry
University of Chicago

and

L. Ridgway Scott
Departments of Computer Science and Mathematics
University of Chicago

Initial value problems

Finite difference methods for solving initial value problem for an ordinary differential equation exhibit limited natural parallelism.

However, for linear systems there are scalable parallel algorithms in which the domain decomposition is in the time domain [11].

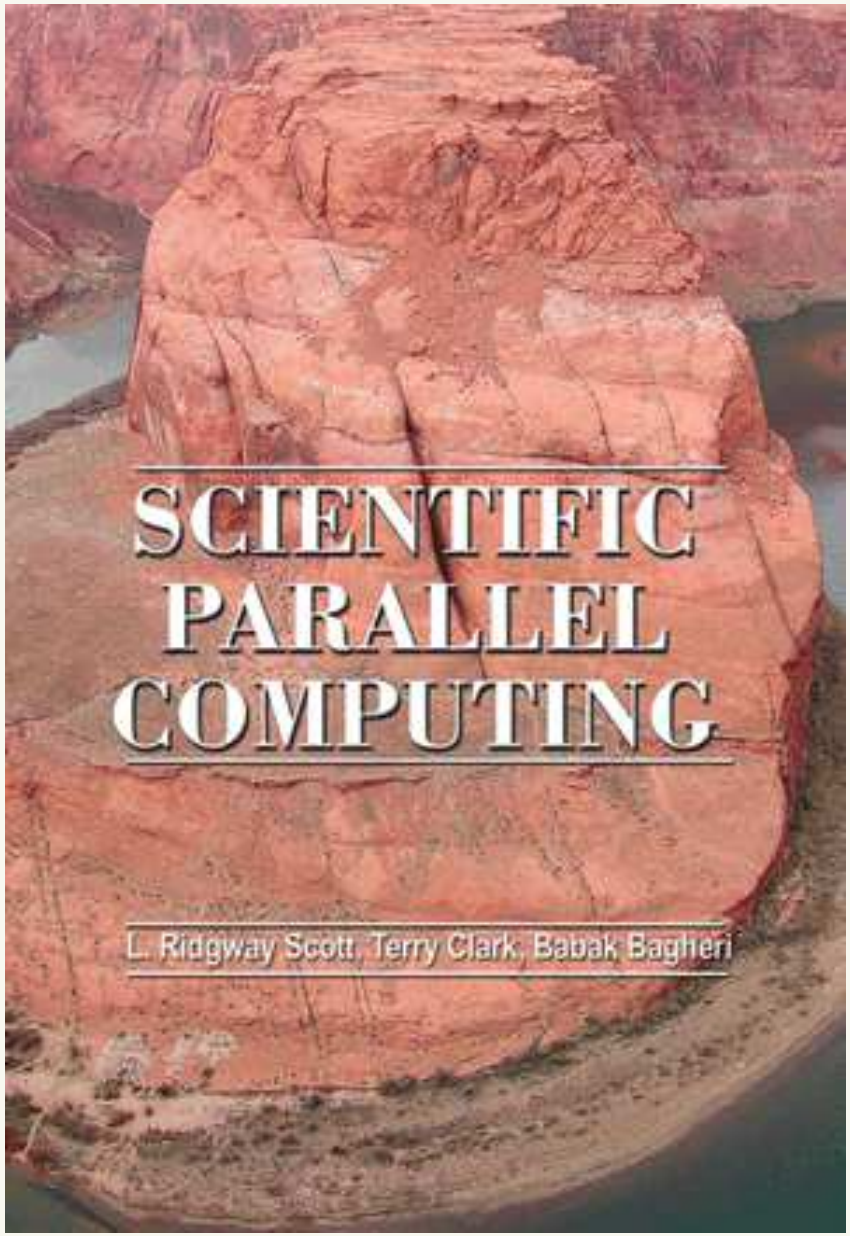
Such techniques are based on scalable methods for solving banded triangular linear systems of equations and have been known for some time (cf. [10, 6]).

What can provide the increasing data size needed for such scalability is a long time interval of integration [9].

Indeed, there are many simulations in which the primary interest is a very long time of integration.

For example, there is a celebrated simulation of the villin headpiece by molecular dynamics [3], which involved 500 million time steps.

Easy HPC



Parallel Performance

I will pay $100 to the first person to demonstrate a speedup of at least 200 on a general purpose, MIMD computer used for scientific computing. *E-mail challenge from Alan Karp, November 1985*

Computer Architecture

What else do you expect from the country that invented rock and roll? *Chevrolet Camero advertisement, May 1994*

Loop Tiling

The rumors of my demise are much exagerrated *Mark Twain*

Dependences

I wish I didn't know now what I didn't know then *from the song "Against the Wind" by Bob Seger*

Linear Systems

In scientific computing, performance is a constraint, not an objective *one of the authors*

Particle Dynamics

The Force will be with you, always *Obi-Wan Kenobi in "Star Wars"*

An example

We begin with a very simple example here motivated by a swinging pendulum.

To a first approximation, the position $u(t)$ of the pendulum satisfies a differential equation

$$\frac{d^2u}{dt^2} = f(u) \tag{1}$$

with initial conditions provided at $t = 0$:

$$u(0) = a_0,\ u'(0) = a_1 \tag{2}$$

for some given data values a and a_1.

The variable u can be taken to denote the angle that the pendulum makes compared with the direction of the force of gravity.

Then $f(u) = mg \sin u$ where g is the gravitational constant and m is the mass of the weight at the end of the pendulum.

(We are ignoring the mass of the rod that holds this weight.)

Discretization

We can approximate via a central difference method to get a recursion relation

$$u_{n+1} = 2u_n - u_{n-1} - \tau f(u_n) \tag{3}$$

where $\tau := \Delta t^2$.
If displacements of the pendulum position from vertical are small, then we can use the approximation $\sin u \approx u$ to get a linear equation

$$\frac{d^2u}{dt^2} = mgu. \tag{4}$$

In this case, the difference equations become a linear recursion relation of the form

$$u_{n+1} = (2 - \tau mg)\, u_n - u_{n-1}. \tag{5}$$

The initial conditions (2) provide starting values for the the recursion.

Dicretization as a linear system

initial values For example, we can take

$$u_0 = a \quad \text{and} \quad u_{-1} = a_0 - a_1 \Delta t. \tag{6}$$

This allows us to solve (5) for $n \geq 0$.

The recursion (5) corresponds to a banded, lower triangular system of equations of the form

$$Lu = b \tag{7}$$

where the bandwidth of L is $w = 2$, the diagonal and subsubdiagonal terms of L are all one, and the subdiagonal terms of L equal $\tau m g - 2$.

The right-hand side g is of the form

$$b_1 = (1 - \tau m g)\, a_0 + a_1 \Delta t \quad \text{and} \quad b_2 = -a_0 \tag{8}$$

and $b_i = 0$ for $i \geq 3$.

Solving triangular systems in parallel

Key to the scalability of Gaussian elimination (GE) is the fact that the work/memory ratio $\rho_{\text{WM}} = n$.

However, triangular solution has a work/memory ratio $\rho_{\text{WM}} = 1$.

The latter is (sequentially) trivial to solve, but there are loop-carried dependences in both the outer and inner loops, although the inner loop is a reduction.

These loops cannot easily be parallelized, but we will see that they can be decomposed in a suggestive way (see Figure 1).

Schematic of “toy duck”

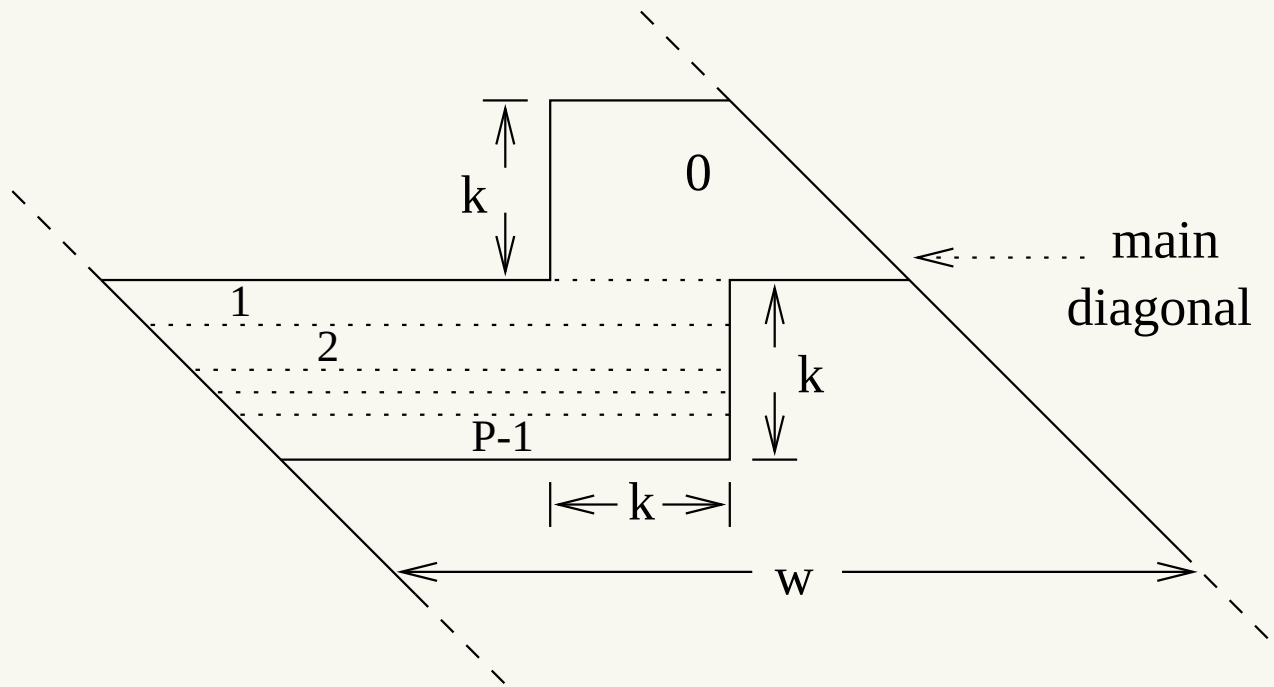


Figure 1: Schematic of “toy duck” parallelization of a banded, triangular matrix equation solution algorithm.

Processors 1 through $P-1$ compute

$$b_i^\ell := \sum_{j=\min(i-w,1)}^{k\ell} a_{i,j}x_j \quad \forall i = 1+(k+1)\ell,\ldots,(k+2)\ell. \qquad (9)$$

Typical step in toy duck

In the simplest case (as we now assume) we will have

$$k = \nu(P-1) \tag{10}$$

for some integer $\nu \geq 1$, so that each processor ≥ 1 computes ν different b_i's using previously computed x_j's.
Note that this requires access to the (previously computed) values x_j for $j \leq k\ell$.
Simulaneously, processor 0 computes $x_{k\ell+1}, \ldots, x_{(k+1)\ell}$ by the standard algorithm, namely,

$$x_i = a_{i,i}^{-1} \left(f_i - b_i^{\ell-1} - \sum_{j=(k-1)\ell+1}^{i-1} a_{i,j} x_j \right) \quad \forall k\ell < i \leq (k+1)\ell. \tag{11}$$

At end of this step, processor 0 sends $x_{k\ell+1}, \ldots, x_{(k+1)\ell}$ to other processors, and other processors send $b_{(k+1)\ell+1}, \ldots b_{(k+2)\ell}$ to processor 0.

Analysis of toy duck

This completes the ℓ-th step for $\ell > 1$.
These steps involve a total of $2k^2$ MAPs (**multiply-add pairs**).
Load balance in (9) can be achieved in a number of ways.
If $\nu = 2$, then perfect load balance is achieved by having processor 1 doing the first and last row, processor 2 doing the second and penultimate row, and so on.
The total number of operation to compute (9) is

$$k(w-k) - \tfrac{1}{2}k^2 = kw - \tfrac{3}{2}k^2 \tag{12}$$

MAPs, where MAPs stands for "multiply-add pairs."
Thus the time estimate for (9) is proportional to

$$\left(w - \tfrac{3}{2}k\right)\frac{k}{P-1} \tag{13}$$

time units, where the unit is taken to be the time required to do one multiply-add pair.

Continued analysis of toy duck

Processor zero does $\frac{3}{2}k^2$ MAPs, and thus the total time for one stage of the program is proportional to

$$\max\left\{\tfrac{3}{2}k^2, \left(w - \tfrac{3}{2}k\right)\frac{k}{P-1}\right\}. \tag{14}$$

These are *balanced* if

$$\tfrac{3}{2}k^2 = \left(w - \tfrac{3}{2}k\right)\frac{k}{P-1} \tag{15}$$

which reduces to having

$$P = \tfrac{2}{3}\frac{w}{k}. \tag{16}$$

Recalling our assumption (10), we find that

$$P(P-1) = \tfrac{2}{3}\frac{w}{\nu} \tag{17}$$

Scaling of toy duck

Optimal P depends only on the band width w and not on n.

Algorithm not scalable in usual sense if w remains fixed independently of n.

Total amount of data communicated at each stage is $2k$ words.

(15) implies that computational load is proportional to k^2, so this algorithm is scalable if $w \to \infty$ as $n \to \infty$.

The case of a full matrix corresponds to $w = n$.

The "toy duck" algorithm has substantial parallelism in this case.

For fixed ν, (17) implies that P is proportional to $\sqrt{w}$, and this in turn implies that k is proportional to $\sqrt{w}$.

The amount of memory per processor is directly proportional to the amount of work per processor, so this is proportional to k^2, and hence w, in the balanced case (15).

A block inverse algorithm

The following algorithm can be found in [10]. Let us write the lower triangular matrix L as a block matrix. Suppose that $n = ks$ for some integers k and $s > w$.

$$\begin{pmatrix} L_1 & 0 & 0 & 0 & 0 & 0 \\ R_1 & L_2 & 0 & 0 & 0 & 0 \\ 0 & R_2 & L_3 & 0 & 0 & 0 \\ 0 & 0 & \cdots & \cdots & 0 & 0 \\ 0 & 0 & 0 & R_{k-2} & L_{k-1} & 0 \\ 0 & 0 & 0 & 0 & R_{k-1} & L_k \end{pmatrix} \tag{18}$$

A triangular matrix is invertible if and only if its diagonal entries are not zero (just apply the forward solution algorithm). Thus any sub-blocks on the diagonal will be invertible as well if L is, as we now assume. That is, each L_i is invertible, no matter what choice of k we make.

Some details

Let D denote the block diagonal matrix with blocks $D_i := L_i^{-1}$. If we premultiply D times L, we get a simplified matrix:

$$DL = \begin{pmatrix} I_s & 0 & 0 & 0 & 0 & 0 \\ G_1 & I_s & 0 & 0 & 0 & 0 \\ 0 & G_2 & I_s & 0 & 0 & 0 \\ 0 & 0 & \cdots & \cdots & 0 & 0 \\ 0 & 0 & 0 & G_{k-2} & I_s & 0 \\ 0 & 0 & 0 & 0 & G_{k-1} & I_s \end{pmatrix} \tag{19}$$

where I_s denotes an $s \times s$ identity matrix, and the matrices G_i arise by solving

$$L_{i+1} G_i = R_i, \quad i = 1, \ldots, k-1. \tag{20}$$

The original system $Lx = f$ is changed to $(DL)x = Df$. Note that we can write Df in block form with blocks (or segments) b_i which solve

$$L_i b_i = f_i, \quad i = 1, \ldots, k. \tag{21}$$

Block inverse details

The blocks L_i in (21) are $s \times s$ lower-triangular matrices with bandwidth w, so the band forward solution algorithm is appropriate to solve (21).
Depending on the relationship between the block size s and the band width w, there may be a certain number of the first columns of the matrices R_i which are identically zero.
In particular, one can see (exercize) that the first $s - w$ columns are zero.
Due to the definition of G_i, the same must be true for them as well (exercize).
Let $\widehat{G}_i$ denote the right-most w columns of $G_i = (0 \quad \widehat{G}_i)$
let M_i denote the top $s - w$ rows of $\widehat{G}_i$ and
let H_i denote the bottom w rows of $\widehat{G}_i$.
Further, split b_i similarly, with
u_i denoting the top $s - w$ entries of b_i and
v_i denoting the bottom w entries of b_i.

Block inverse notation

We may then write the blocks (strips) x_i of the solution vector in two corresponding parts: y_i denoting the top $s-w$ entries of x_i and z_i denoting the bottom w entries of x_i.

The notation is summarized in

$$\widehat{G}_i = \begin{pmatrix} M_i \\ H_i \end{pmatrix} \qquad x_i = \begin{pmatrix} y_i \\ z_i \end{pmatrix} \qquad b_i = \begin{pmatrix} u_i \\ v_i \end{pmatrix} \qquad \begin{matrix} \}s-w \\ \}w \end{matrix} \tag{22}$$

and the dimensions of M_i and H_i are

$$\underbrace{\begin{matrix} M_i \\ \end{matrix}}_{w} \quad \begin{matrix} \} \, s-w \end{matrix} \qquad \underbrace{\begin{matrix} H_i \\ \end{matrix}}_{w} \quad \begin{matrix} \} \, w \end{matrix} \tag{23}$$

Block inverse reduction

All of these quantities have now simple relationships. First of all we have

$$y_1 = u_1, \quad z_1 = v_1 \tag{24}$$

we can inductively determine the z_i's by

$$z_{i+1} = v_{i+1} - H_i z_i \quad \forall i = 1, \ldots, k-1 \tag{25}$$

Then we can separately determine the y_i's by

$$y_{i+1} = u_{i+1} - M_i z_i \quad \forall i = 1, \ldots, k-1 \tag{26}$$

There are no dependences in (26), but (25) appears at first to be sequential.
However, if w is sufficiently large, there is an opportunity for parallelism in each iteration of (25).
Moreover, (25) can be written as a lower-triangular system itself, and we describe an appropriate parallel solution algorithm.

Two cases

There are two cases to distinguish: system is solved only once, and the systems (20) become a major part of the computation, and the system is solved many times, and the cost of solving the systems (20) can be amortized since they need be solved only once.
The primary amount of work in (21) is

$$k\left(ws - \tfrac{1}{2}w^2\right) = wn - \tfrac{1}{2}\frac{w^2 n}{s} \tag{27}$$

MAPs, whereas the primary amount of work in (25) is

$$w^2(k-1) = \frac{w^2(n-s)}{s} \tag{28}$$

MAPs, and the primary amount of work in (26) is (in MAPs)

$$w(s-w)(k-1) = \frac{w(s-w)(n-s)}{s} = wn - \frac{w^2 n}{s} - w(s-w). \tag{29}$$

Block inverse analysis

The sum of (27), (28) and (29) is nearly $2nw$, twice the amount of work in the sequential case.
The amount of parallelism in (25) is complex to assess [11], but once all the z_i's are computed (and appropriately distributed), (26) can be done (trivially) in parallel.
Moreover, (27) can also be computed trivially in parallel.
If (25) is computed sequentially, then

$$T_P \geq wn\left(\frac{w}{s} + \frac{1}{P}\left(1 - \frac{w}{s}\right)\right) = wn\left(\frac{wP}{n} + \frac{1}{P}\left(1 - \frac{wP}{n}\right)\right) \quad (30)$$

if $P = k$. Therefore

$$E_P^{-1} \geq \frac{wP^2}{n} + \left(1 - \frac{wP}{n}\right) \quad (31)$$

Taking $P \leq \sqrt{n/w}$ would be required for a scalable algorithm.

Decision process

We have two algorithms with complementary scalability regimes.

We can choose one or the other to match the data.

If $n \leq w^2$, choose Toy Duck and use $P \leq \sqrt{w}$.

If $n \geq w^2$, choose Block Inverse and use $P \leq \sqrt{n/w} \geq \sqrt{w}$.

In the latter case, note that the scalability limit on P is $\sqrt{n/w} \geq \sqrt{w}$.

Combining the two options, we get scalability for large n as desired:

$$P \leq \max\{\sqrt{n/w}, \sqrt{w}\}$$

Setting $k = n/w^2$ (so that $w = \sqrt{n/k}$) we have scalability for

$$P \leq n^{1/4} \max\{k^{1/4}, k^{-1/4}\}$$

Now we consider how these ideas can be applied to nonlinear equations.

Nonlinear systems

When f is not linear, such algorithms are not directly applicable. We can formulate a set of equations to define the entire vector of values u_i as an ensemble, but it is no longer a linear equation. We can write it formally as

$$F(U) = 0 \tag{32}$$

where F is defined by

$$F(U) = L^0 U + \tau m g \phi(U) - b \tag{33}$$

with L^0 the same as L above with $\tau = 0$, that is L^0 is a lower triangular matrix with bandwidth $w = 2$, the diagonal and subsubdiagonal terms of L^0 are all one, and the subdiagonal terms of L^0 equal -2. The function ϕ thus contains all of the nonlinearity and has the simple form ($\delta_{i,j}$ is the Kronecker delta)

$$\phi(U)_{ij} = \delta_{j,i-1} \sin u_{i-1}. \tag{34}$$

Newton's method

The Newton-Raphson method can be written in the form

$$F'(U^n)\left(U^{n+1} - U^n\right) = -F(U^n). \tag{35}$$

where F' is the Jacobian matrix of all partial derivatives of F with respect to the vector U.
To see how this works, let us return to the pendulum problem. In the case of the pendulum, we have F defined by (33). It takes a careful look at the definition, but it is not hard to see that the Jacobian matrix for a linear function of the form $U \to L^0U$ is the matrix L^0. Thus $J_F(U) = L^0 + \tau m g J_\phi(U)$. With ϕ of the form (34), it can be shown that

$$J_\phi(U)_{ij} = \delta_{j,i-1}\phi'(u_{i-1}) = \delta_{j,i-1}\cos u_{i-1}. \tag{36}$$

The Jacobian has the same form as the original matrix L. In fact, if F is linear, Newton's method is equivalent to just solving the system (7) and converges in one step.

Linearized initial value problem

There is another way to interpret the Newton algorithm for solving the initial value problem.

Suppose that we have an approximate solution u to (1) which satisfies the initial conditions (2), and define the residual $R(u)$ by

$$R(u) := \frac{d^2u}{dt^2} - f(u) \tag{37}$$

which is a function of t defined on whatever interval u is defined on.

We can apply Newton's method (in the appropriate infinite dimensional setting) to solve

$$R(u) = 0. \tag{38}$$

Let us derive the resulting equations by an elementary approach.

Newton step derivation

Suppose that we try to add a perturbation v to u to get it to satisfy (1) (more) exactly. We use a Taylor expansion to write

$$f(u+v) = f(u) + vf'(u) + \mathcal{O}(v^2) \tag{39}$$

Thus the sum $u+v$ satisfies an initial value problem of the form

$$\frac{d^2(u+v)}{dt^2} - f(u+v) = \boxed{\frac{d^2v}{dt^2} - vf'(u) + R(u)} + \mathcal{O}(v^2). \tag{40}$$

With (40) as motivation, we now *define* v by solving the initial value problem

$$\frac{d^2v}{dt^2} = vf'(u) - R(u) \tag{41}$$

with initial conditions provided at $t = 0$:

$$v(0) = 0,\ v'(0) = 0. \tag{42}$$

Newton versus discretization

Then the Newton step for solving (38) is the solution v of (41-42).
Now (35) can now be seen as just a discretization of the initial value problem (41).
This can be depicted in a diagram:

$$\begin{array}{ccc} \text{ODE } (1) & \xrightarrow{\text{Newton}} & \text{linearized ODE } (41-42) \\ \downarrow \delta & & \downarrow \delta \\ \text{diff. meth. } (5-6) & \xrightarrow{\text{Newton}} & \text{discrete Newton method } (35), \end{array} \tag{43}$$

where the symbol δ denotes time discretization.
Thus we expect that, in the limit of small time step, the number of Newton iterates as a function of time would approach a limit.
Note that Newton's method for ODEs always has a unique solution, at least on appropriate time intervals.

Atypical for Newton's method.

Getting Newton started

Newton's method (35) converges rapidly once you get close to a solution, but how do you get close in the first place?

Does not have a general answer, but we indicate one approach here.

Suppose that we had a simplified approximation $\tilde{f}$ to f and we solved

$$\frac{d^2u}{dt^2} = \tilde{f}(u) \tag{44}$$

exactly, together with the initial conditions (2).

For example, with $f(u) = \sin u$ we might have $\tilde{f}(u) = u$ as an approximation.

This makes (44) linear, and in this simple case we can even solve the equation in closed form (in terms of sines and cosines).

It is much faster to evaluate $\tilde{f}(u)$ in this case than it is $f(u)$, so the computation of the initial guess would be much less costly.

Newton start

If we use the solution u to (44) as the starting guess for Newton's method, then the initial residual $R(u)$ has a simple interpretation. We can express it simply as

$$R(u) = \frac{d^2u}{dt^2} - f(u) = \tilde{f}(u) - f(u). \tag{45}$$

The size of the residual is simply related to the error in approximation of f by $\tilde{f}$.
Since the first Newton step v satisfies (41), we can bound the size of v in terms of $R(u)$, and hence $f - \tilde{f}$. More precisely, (41) becomes

$$\frac{d^2v}{dt^2} = vf'(u) - \left(\tilde{f}(u) - f(u)\right). \tag{46}$$

Since v is zero at the start (see (2)), it will be small for at least some reasonable interval of time.
The size of v can be predicted as u is computed, and the process can be stopped if the prediction gets too large.

Using a constant solution to start

Thus there is a natural way to control the size of the Newton step in this case.
If we take $\tilde{f} \equiv 0$, this corresponds taking the initial guess to be constant in time.
We will explore this option in detail.

Using a coarse time-step to start It would also be possible to define an initial start for Newton by approximating on a coarser mesh.
That is, we could use a larger Δt in (3) (recall that $\tau = \Delta t^2$).
It would then be necessary to interpolate onto a finer mesh to define the residual appropriate for (41) (or equivalently in (35) in the discretized case).
In this way, a multi-grid approach could be developed in the time variable.
We will explore this option in detail.

Application to orbit simulation

We consider the simple two body problem of planetary motion, with one body (the "sun") fixed at the origin $(0,0)$. Thus the number of unknown particle positions is $N = 1$, and the number of dimensions $D = 2$. Let the state be $z = (x, y)$ (or (r, θ) in polar coordinates). The particle moves on a potential surface $U = G/r = \frac{G}{(x^2+y^2)^{1/2}}$, so the force field is

$$F(x,y) = (F_1(x,y), F_2(x,y)) = \left(\frac{-Gx}{(x^2+y^2)^{3/2}}, \frac{-Gy}{(x^2+y^2)^{3/2}} \right).$$

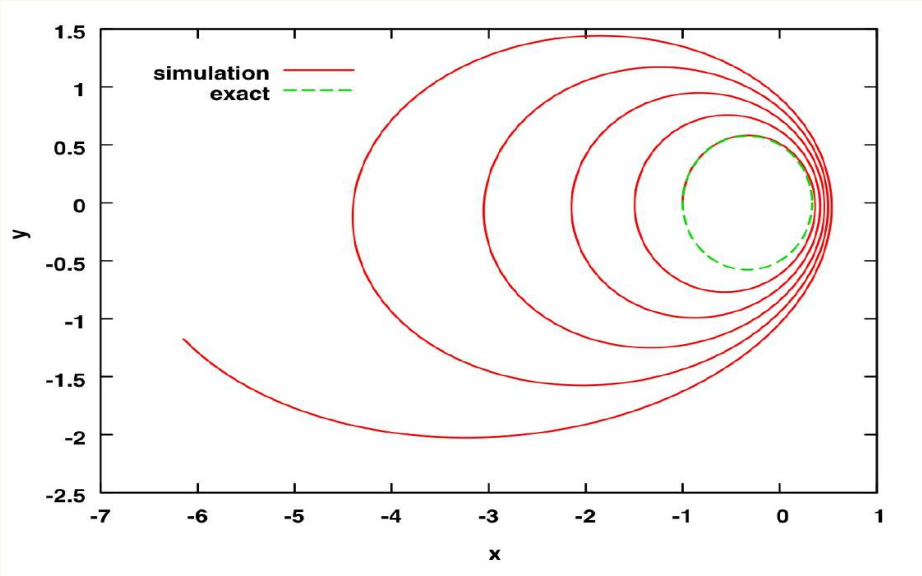
Note as well that

$$J_F(x,y) = \begin{pmatrix} \frac{G(2x^2-y^2)}{(x^2+y^2)^{5/2}} & \frac{3Gxy}{(x^2+y^2)^{5/2}} \\ \\ \frac{3Gxy}{(x^2+y^2)^{5/2}} & \frac{G(2y^2-x^2)}{(x^2+y^2)^{5/2}} \end{pmatrix} \tag{47}$$

Orbit system

Additionally, any given orbit can be mapped onto one with the gravitational source at $(0, 0)$ and initial conditions $(x_0, y_0) = (-1, 0)$, $(\dot{x}_0, \dot{y}_0) = (1, 0)$, and the parameter G variable. The previous values are used as initial conditions.

(a)



(b)

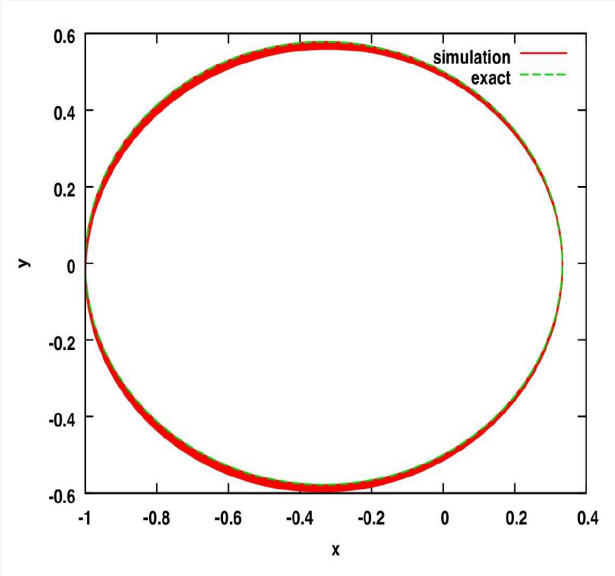


Figure 2: Integration of orbit system, with $G = 4$, gravity from origin, $\triangle t = 0.01$, run for 50 time units, initial condition $(x_0, y_0) = (-1, 0)$, $(\dot{x}_0, \dot{y}_0) = (0, 1)$. Exact solution in polar coordinates is $r = \frac{p}{1+e\cos\theta}$, where $p = (\vec{r} \times \vec{v})/G$, a conserved quantity ($p = 1/G$ here), and $e = 1 - \frac{2}{r_a/r_p+1}$, where r_a is the farthest the the orbit gets from the origin and r_p is the closest; $e = p - 1$ here. The period is $2\pi\sqrt{r_a^3/G}$. (a) Euler scheme, (b) Verlet.

Computational results for the orbit problem

Figure 3 depicts eight iterations of Newton's method for the orbit problem, each one offset artificially along the time axis.

The dashed (green) line indicates the exact orbit, and the solid (red) line indicates the computed Newton step.

The initial step has a constant state, as indicated by the straight line for the left most pair of curves.

Subsequent iterates follow the orbit more and more, but the first few eventually move away from the orbit.

The fifth iterate agrees with the exact orbit to graphical accuracy, and the remaining iterates home in to the orbit to a tolerance of 10^{-10}.

Whether we require only 5 iterations or insist on 8 iterations, the Newton strategy provides a substantial amount of parallelism for the orbit problem.

Orbit problem

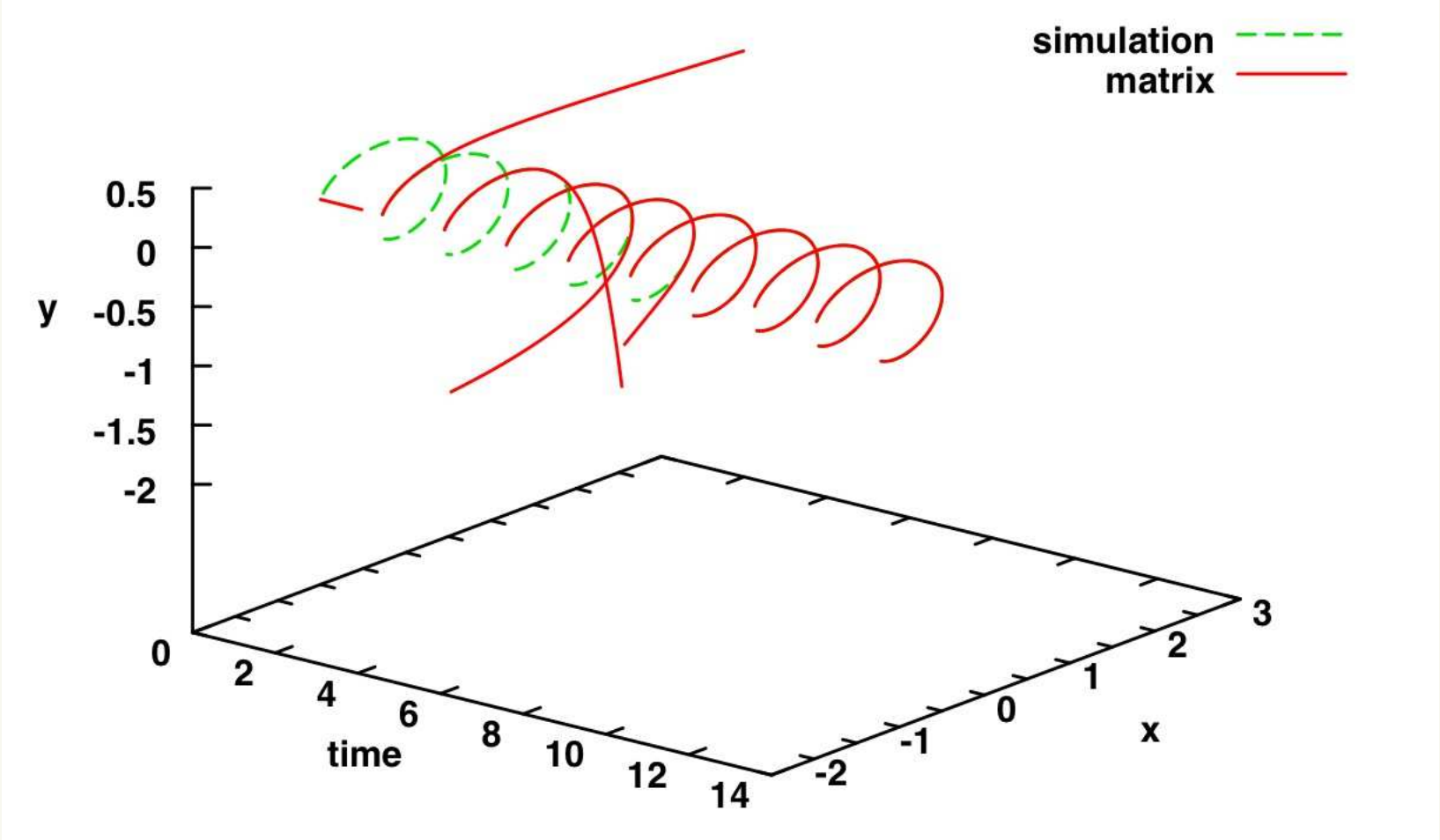


Figure 3: Orbit problem: period = 1.36. 1000 times steps with $\Delta t = 0.001$. Converged in 8 iterations.

Longer time integration

The computations in Figure 3 are carried out for a longer time integration in Figure 4.

It appears that for longer times, the number of iterations I of Newton's method required for convergence grows linearly with the number N of time steps.

Figure 4 suggests that

$$I \approx 0.004N = 4\Delta t\,\tau, \tag{48}$$

where τ is the time of integration.

Figure 5 confirms this for other time steps and supports the theoretical prediction in (43).

Summary of computations

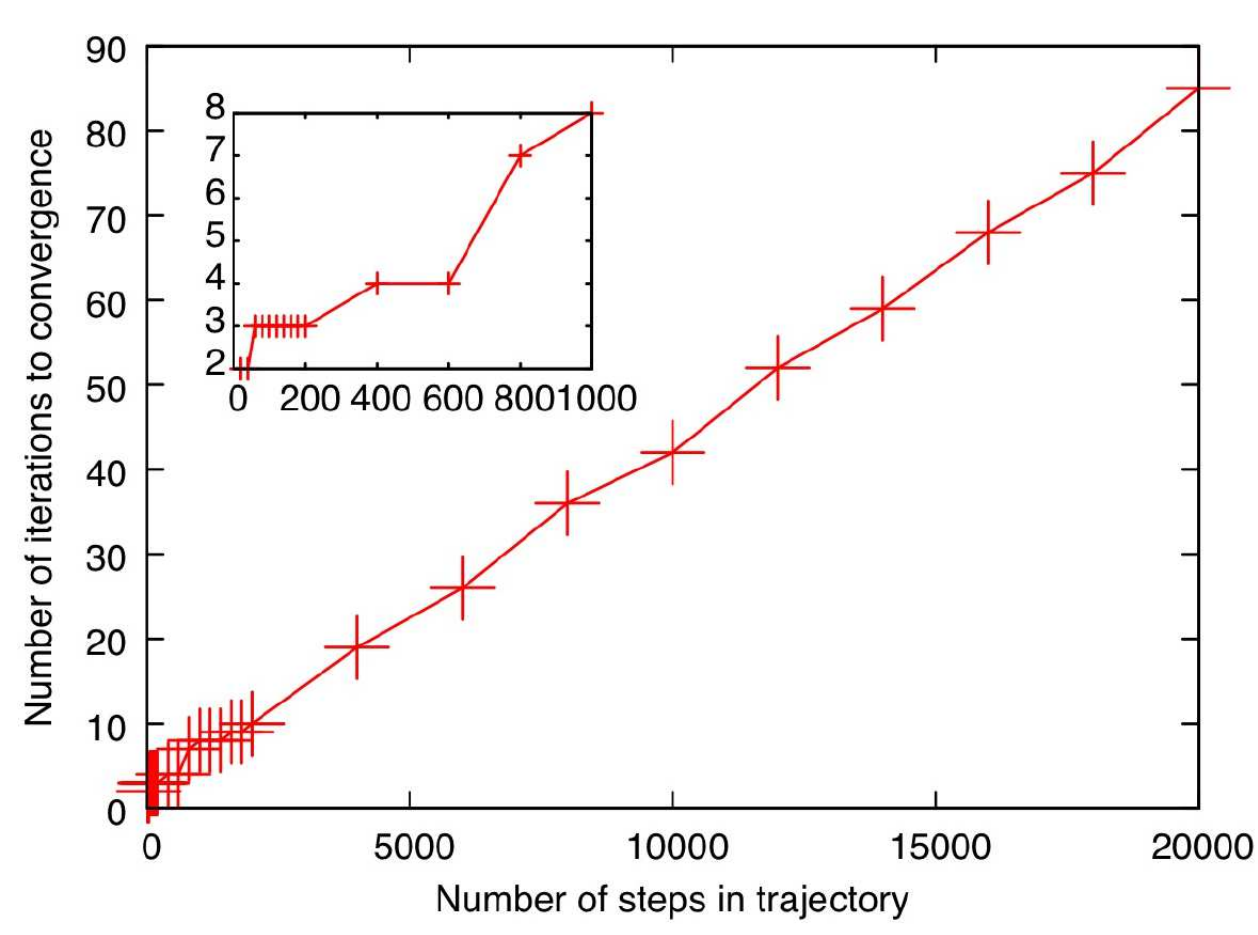


Figure 4: Summary of computations depicted in Figure 3 for different lengths of computation. Vertical axis is number of Newton iterations required to converge.

Orbit simulations

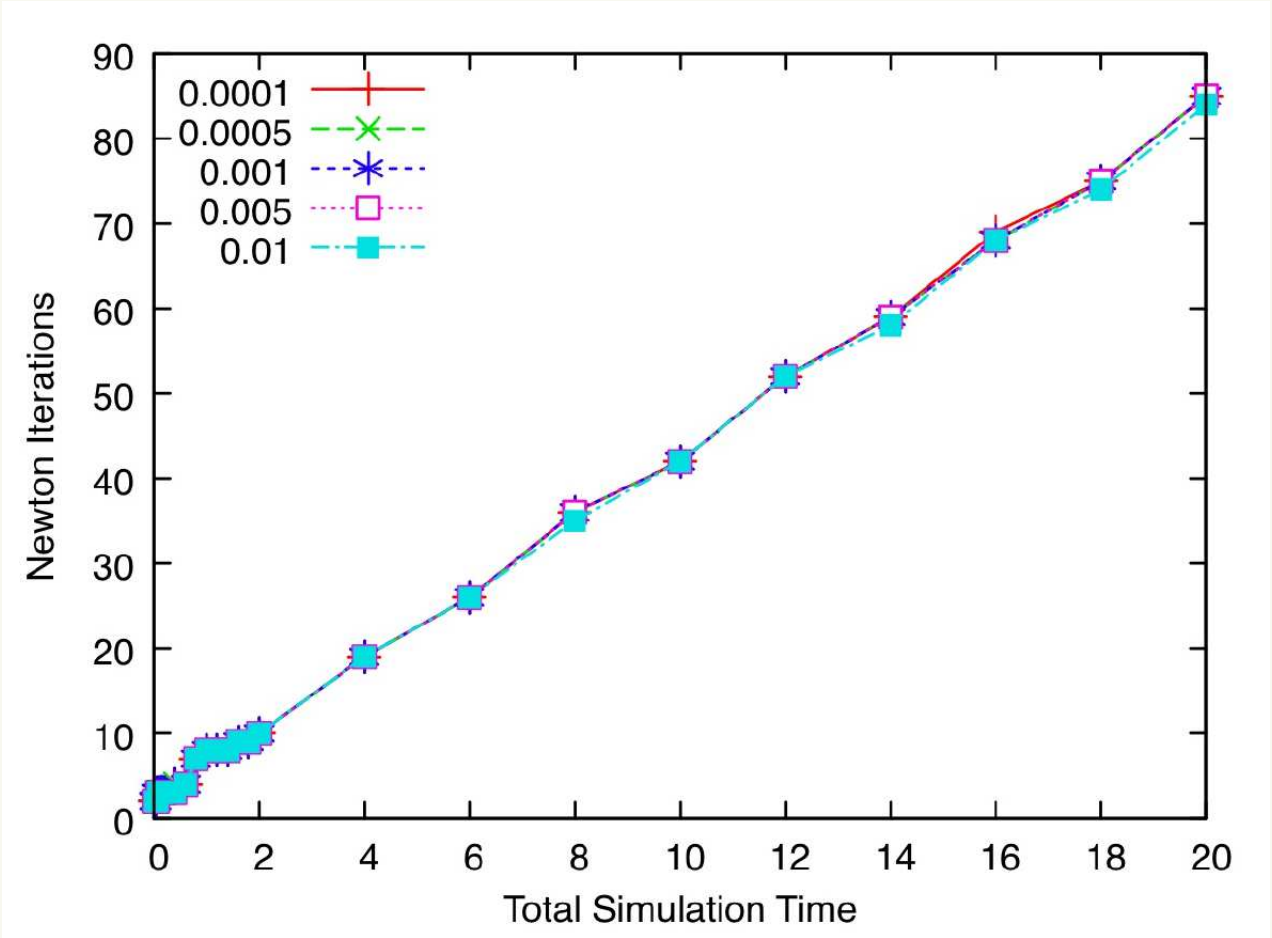


Figure 5: Orbit simulations for different time steps. Horizontal axis: simulation time. Vertical axis: number of Newton iterations. Constant initial guess.

Parallel computation of the orbit problem

The parallel tridiagonal linear solves can be performed efficiently with

$$P \log P = aN \tag{49}$$

for some constant a, with the parallel time $\widetilde{T}_{P,N} \approx cN/P$ for each Newton step for a constant c.
The constant a is ours to adjust; the larger we make it, the larger the granularity of the parallel solution algorithm, although the smaller is P.
Thus we can assume that c does not increase when a is increased.
Assume that the sequential problem takes about $T_{1,N} \approx cN$ time for simplicity.
The number of Newton iterations I required is $bN = \beta \Delta t \, \tau$, as suggested by (48), so the asymptotic total parallel execution time (for N large) is

$$T_{P,N} \approx \frac{IcN}{P} = \frac{bcN^2}{P} = \frac{b}{a} T_{1,N} \log P. \tag{50}$$

Speedup of the orbit problem

This says that the speedup would be estimated by the relation

$$S_{P,N} \approx \frac{a}{b \log P} = \frac{a}{\beta \Delta t \, \tau}. \tag{51}$$

In particular, this says that the speedup can be arbitrarily large for fixed τ as $\Delta t \to 0$.
On the other hand, (51) also says that the efficiency would be restricted by the relation

$$E_{P,N} \approx \frac{a}{bP \log P} = \frac{1}{bN} = \frac{1}{I}, \tag{52}$$

consistent with the observation that our algorithm simply duplicates the sequential algorithm I times.
Note that the adjustable parameters (a and P) drop out of the relation (52) for efficiency.

Smarter start

The data presented so far relate to an initial guess for the Newton iteration in which the initial solution is just constant in time.
It is remarkable that this works at all, but it would not be surprising that there are smarter ways to start.
The number of possible ways to do so is unlimited, so we experimented with just a simple approach: solving sequentially with a larger time step.
We chose a time step ten times larger to produce the initial guess.
Figure 6 shows that benefit of coarser time step improves when ultimate goal is to approximate with a smaller time step.
In view of (43), we can interpret these data as being equivalent to solving the continuous Newton problem with an initial guess corresponding to a discretization using increasingly finer time steps.
Thus it is not surprising that the curves in Figure 6 appear to tend to a constant as the time step is decreased.

Orbit simulations

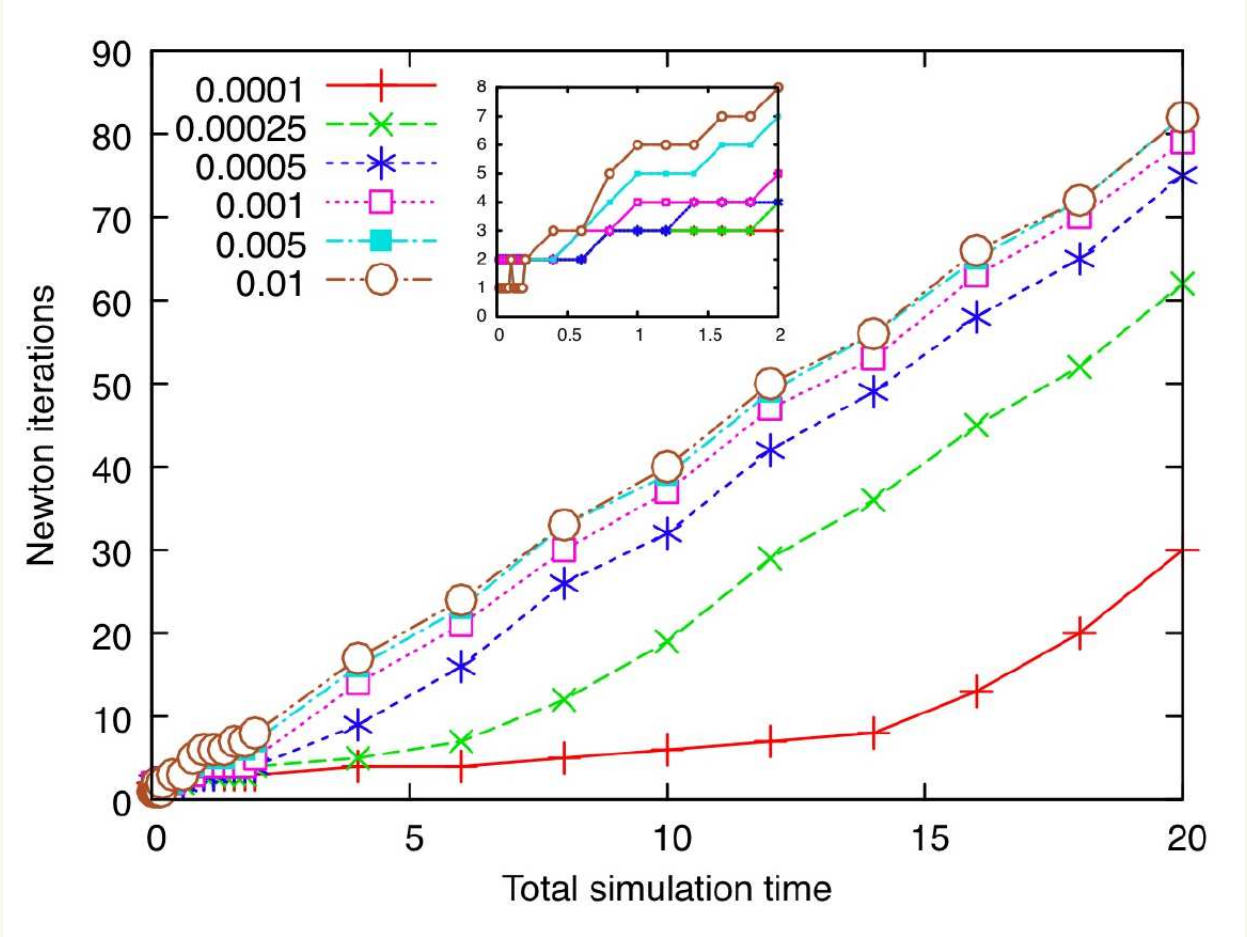


Figure 6: Orbit simulations for different time steps. Horizontal axis: simulation time. Vertical axis: number of Newton iterations. Initial guess is solution using a (ten-times) coarser time step.

Orbit simulations

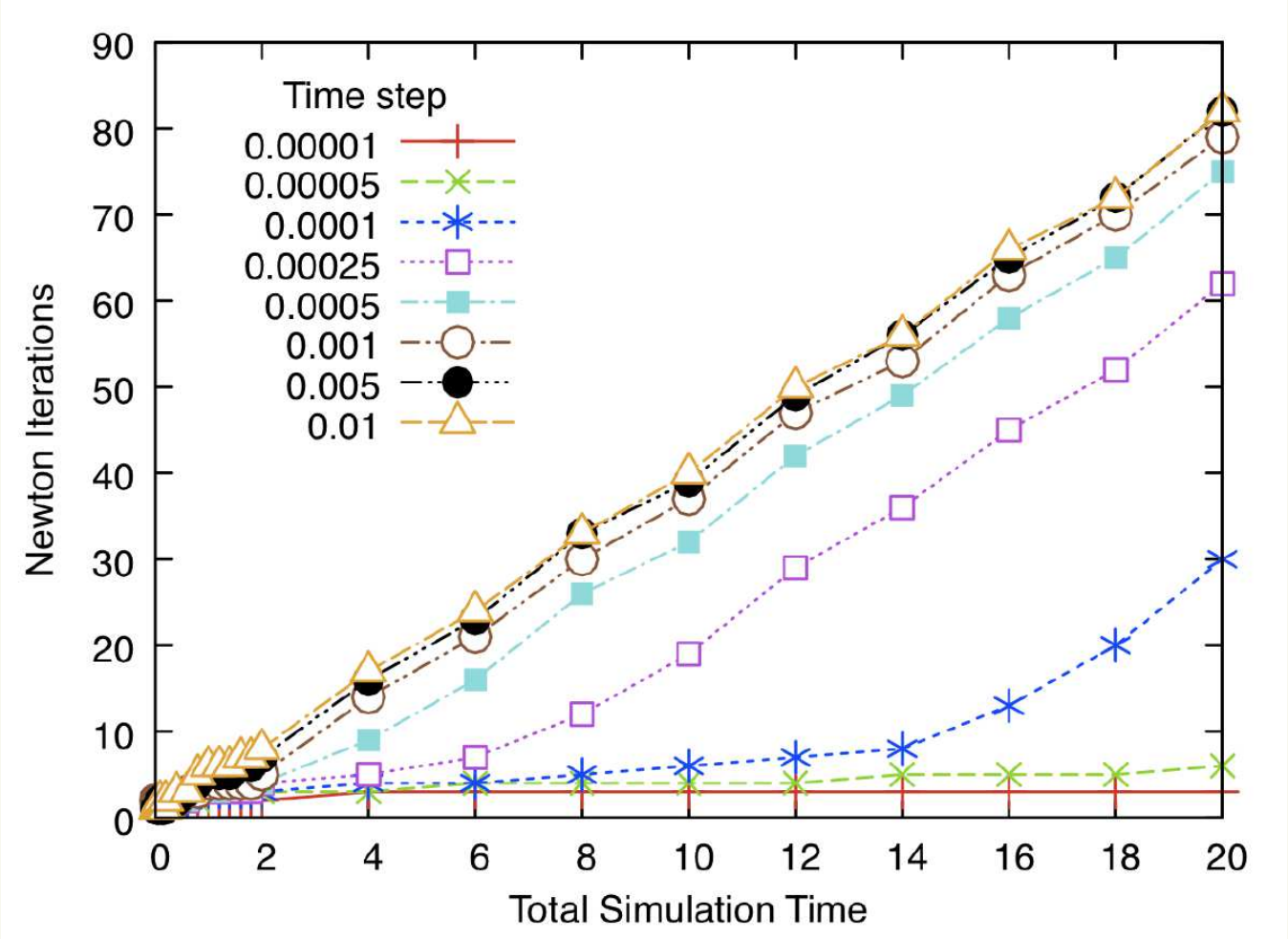


Figure 7: Orbit simulations for different time steps. Horizontal axis: simulation time. Vertical axis: number of Newton iterations. Initial guess is solution using a (ten-times) coarser time step.

Lorenz attractor

Another example was considered, the Lorenz system

$$\begin{aligned} \dot{x} &= \sigma(y - x) \\ \dot{y} &= rx - y - xz \\ \dot{z} &= xy - bz, \end{aligned} \tag{53}$$

where

$$\sigma = 10,\ b = 8/3, \quad \text{and } r = 28. \tag{54}$$

Typical behavior is shown in Figure 8 which depicts two solutions that start near each other but quickly diverge.

However, the solutions exhibit a type of near-periodic behavior, cycling around two attractors indicated by plus signs.

Lorenz solutions

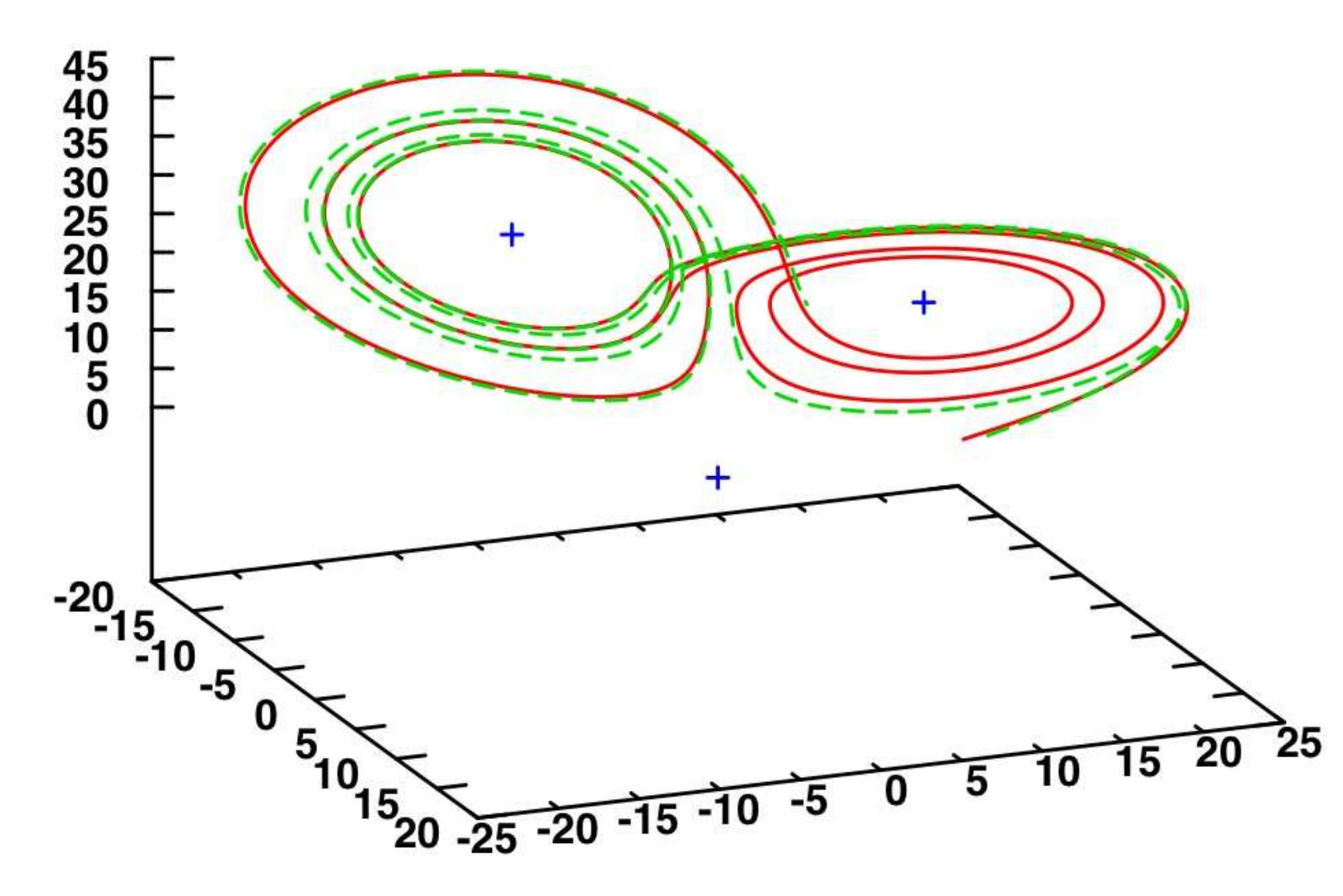


Figure 8: Solutions of the Lorenz system (52) with the coefficients (54).

Lorenz attractor Newton parallelization

Figure 9 indicates the number of Newton iterations required to solve the difference approximation to the Lorenz system (52) with the coefficients shown in (54) using a constant initial guess for various time steps.

We see confirmation of the prediction in (43).

Figure 10 indicates the decrease in the required number of Newton iterations to solve the difference approximation to the Lorenz system (52) with the coefficients shown in (54) using an initial guess based on a (ten times) coarser time step.

Number of Newton iterations

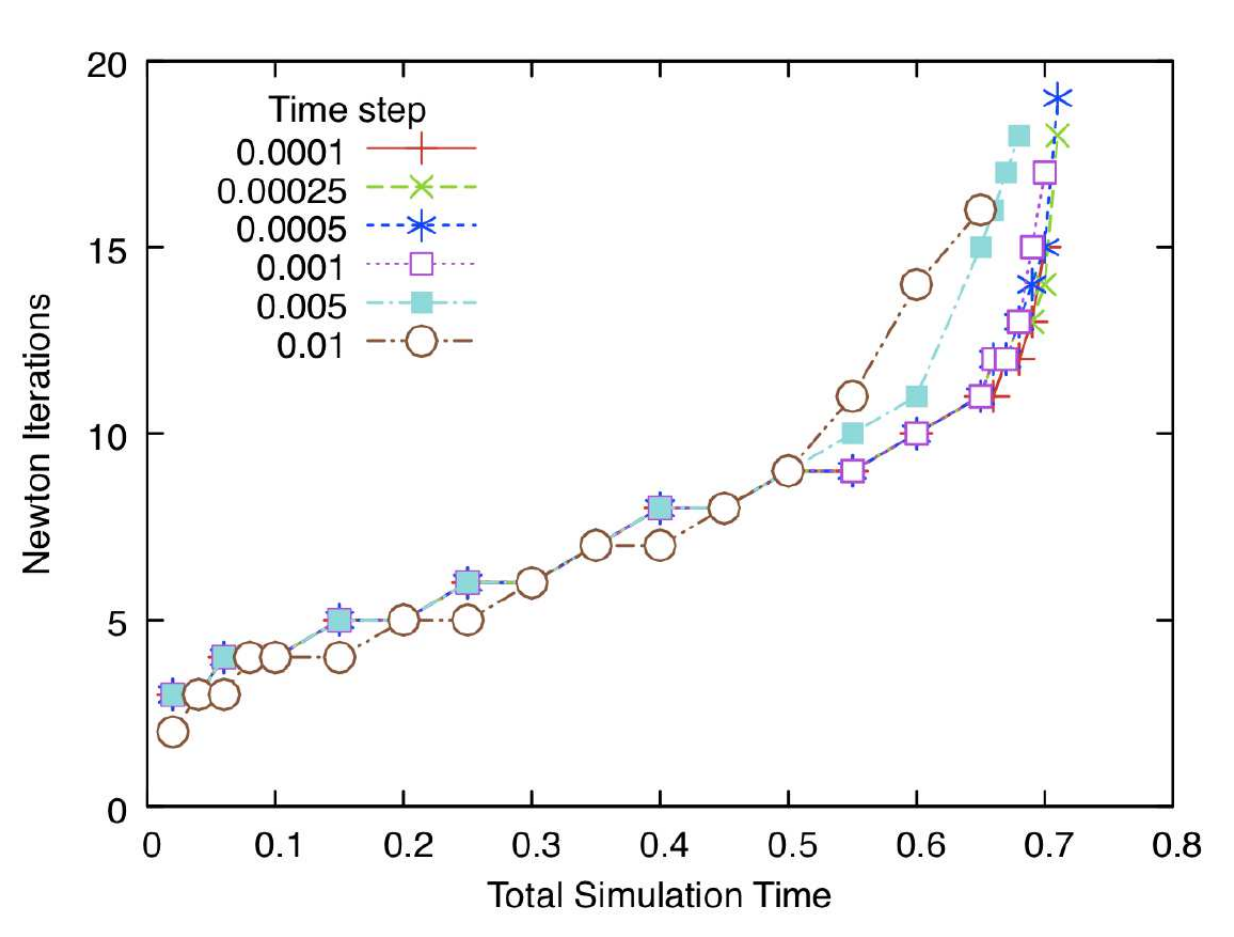


Figure 9: Number of Newton iterations required to solve the difference approximation to the Lorenz system (52) with the coefficients shown in (54) using various time steps. Horizontal axis is time. The initial guess for the Newton iteration is constant in time.

Number of Newton iterations

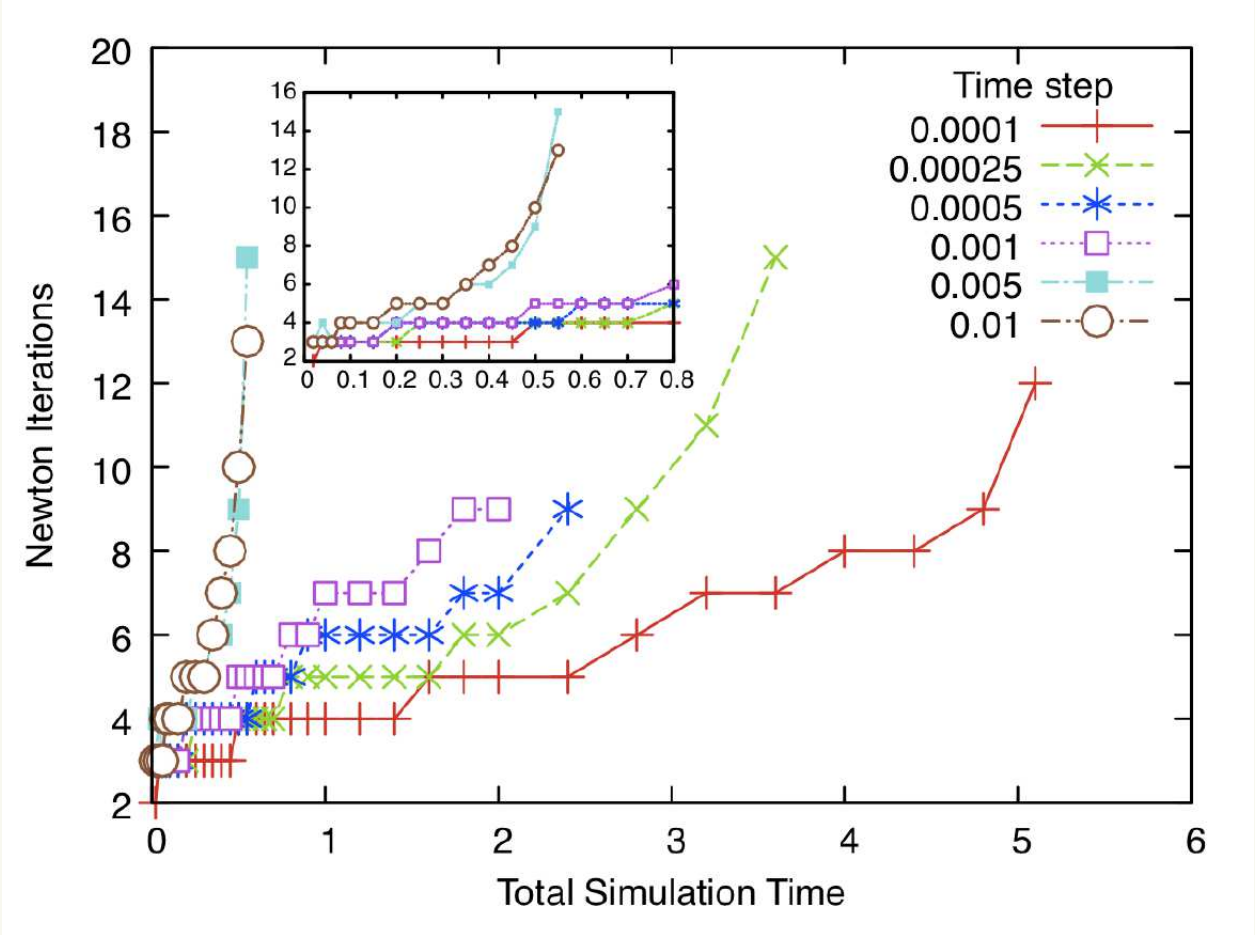


Figure 10: Number of Newton iterations required to solve the difference approximation to the Lorenz system (52) with the coefficients shown in (54) using various time steps. Initial guess based on a (ten times) coarser time step.

Parallel implementation

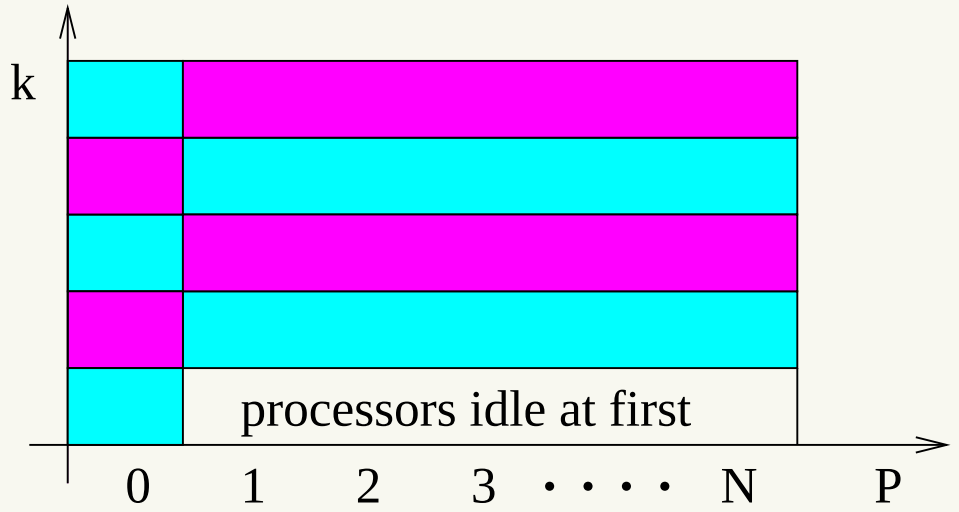


Figure 11: Parallel implmentation. At stage k, $\ell = k$ for processor 0 and $\ell = k - 1$ for the rest.

Cyan indicates processors computing solution for time values

$$2\ell\delta t \leq t \leq (2\ell + 1)\delta t \quad (\ell = 0, 1, 2, \dots)$$

and magenta indicates processors computing solution for

$$(2\ell + 1)\delta t \leq t \leq 2(\ell + 1)\delta t \quad (\ell = 0, 1, 2, \dots).$$

Other approaches

Many high-order methods have been proposed that exhibit parallelism in the high-order steps [2].

“Parareal” methods provide efficient parallelizations of non-linear, time-dependent problems via domain decomposition in the time domain [1, 7, 8, 4].

One limitation is the inclusion of a serial section which plays the role of a preconditioner.

Techniques presented here can be integrated with those of [1, 7, 8] to potentially improve scalability.

The techniques here may provide a scalable parallel algorithm for this or a similar preconditioner.

Conclusions (and Perspectives)

We have demonstrated that the Newton parallel method can provide significant speedup for solving ODE's.

The efficiency is limited by the number of required Newton iterations, but the speedup can be arbitrarily large as the time step is decreased.

The main reason that efficiency is limited by the number of required Newton iterations for the problems considered here is that we focused on explicit time-stepping schemes.

Thus the sequential algorithm gets replicated in each Newton iteration.

But for problems requiring implicit time-stepping schemes, a Newton (or similar) iteration might be required even in the sequential case.

In this scenario, the efficiency might be significantly better.

(Conclusions and) Perspectives

The Newton parallel method benefits greatly from a good initial guess.
What is the best way to use P processors to collectively approximate a solution to an ODE?

- Sounds like original question but now Newton parallel method computes refinement, so initial step need not be convergent.
- One idea: each processor solves slightly different problems and then we use this data to synthesize an approximation.
- Something like the data assimilation problem?

Applying these ideas to ODE's arising from the semi-discretization of PDE's will be a significant challenge.

- Molecular dynamics is typically done with explicit time-stepping methods (e.g., Verlet)
- flow problems (Navier-Stokes) often involve implicity methods (Euler or backward differentiaion) [5].

References

[1] L. Baffico, S. Bernard, Y. Maday, G. Turinici, and G. Zérah. Parallel-in-time molecular-dynamics simulations. *Phys. Rev. E*, 66:057701, 2002.

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

[3] Y. Duan and P. Kollman. Pathways to a protein folding intermediate observed in a 1-microsecond simulation in aqueous solution. *Science*, 282:740–744, 1998.

[4] M. Emmett and M. Minion. Toward and efficient parallel in time method for partial differential equations. *Communications in Applied Mathematics and Computational Science*, 7(1):105–132, 2012.

[5] Hector Juarez, L. R. Scott, R. Metcalfe, and B. Bagheri. Direct simulation of freely rotating cylinders in viscous flows by high–order finite element methods. *Computers & Fluids*, 29:547–582, 2000.

[6] F. Thomson Leighton. *Introduction to Parallel Algorithms and Architectures: Arrays, Trees, and Hypercubes*. Morgan Kaufmann, San Mateo, CA, 1992.

[7] Jacques-Louis Lions, Yvon Maday, and Gabriel Turinici. Résolution d'EDP par un schéma en temps "pararéel". *C. R. Acad. Sci. Paris*, 332(7):661–668, 2001.

[8] Yvon Maday and Gabriel Turinici. A parareal in time procedure for the control of partial differential equation. *C. R. Acad. Sci. Paris*, 335(4):387–392, 2002.

[9] Mark Maienschein-Cline and L. Ridgway Scott. Scalable solution of non-linear time-dependent systems. Research Report UC/CS TR-2011-1, Dept. Comp. Sci., Univ. Chicago, 2011.

[10] U. Schendel. *Introduction to numerical methods for parallel computers*. Chichester: Ellis Horwood Limited, 1984.

[11] L. R. Scott, T. W. Clark, and B. Bagheri. *Scientific Parlallel Computing*. Princeton University Press, 2005.