Numerical Solutions to the KdV Equation

Hannah Morgan

Abstract

Implicit difference schemes for nonlinear PDEs, such as the Korteweg-de Vries (KdV) equation, require large systems of equations to be solved at each timestep, leading to long computation times. A group of finite difference schemes for KdV are given here, and a parallel algorithm is presented and implemented for one such scheme. A computational model is compared to experimental results, and the performance and scalability of the algorithm are discussed.

1 Introduction

The Korteweg-de Vries (KdV) equation, given by (1), is a nonlinear PDE first introduced in [1] in 1895 to model low amplitude water waves in shallow, narrow channels like canals.

$$u_t + 6uu_x + u_{xxx} = 0 \tag{1}$$

It has since been applied to many other problems in physics and engineering including plasma physics. KdV has been extensively studied both theoretically and numerically. Many finite difference schemes have been derived for the equation including an explicit one by Zabusky and Kruskal in [2], who discovered the existence of solitons, and an implicit one from Goda in [3]. Spectral methods (e.g. [4] and [5]) and finite element methods (e.g. [6], [7], and [8]) have also been used to solve KdV. The accuracy of a given method is simple to measure since KdV has a smooth analytical solution given by (2).

$$u(t,x) = \frac{c}{2}\text{sech}^2\left[\frac{\sqrt{c}}{2}(x - ct - a)\right] \tag{2}$$

With the rise of parallel numerical analysis, some work has been done to effectively implement certain methods to solve KdV. For example, an intrinsically parallel finite difference scheme was developed in [9] and a pseudospectral method was implemented in parallel in [10]. In this paper, we discuss implementation techniques of a finite difference scheme for KdV and use the block inverse algorithm, a parallel numerical algorithm given in [11], in an attempt to improve the computation time needed to solve KdV.

2 A finite difference scheme

In this section, we will build a finite difference scheme to solve the KdV equation. To find $u(t,x)$, we will first find an approximation for the nonlinear advection equation $u_t + f(u)_x = 0$ with $f(u) = 3u^2$ and then add the dispersive term u_{xxx}. We will implement our scheme in Octave.

2.1 Formulation

To obtain a finite difference method, we first write the first order Taylor series where $x_{i+1} = x_i + h$ and $x_{i-1} = x_i - h$:

$$\begin{aligned} y(x_{i-1}) &= y(x_i) - hy'(x_i) \\ y(x_{i+1}) &= y(x_i) + hy'(x_i) \end{aligned}$$

Using the equations above, we can write a forward time and backward space scheme:

$$\begin{aligned} u_t &= \frac{u_{i+1,j} - u_{i,j}}{\Delta t} \\ f(u)_x &= \frac{f(u_{i,j}) - f(u_{i,j-1})}{\Delta x} \end{aligned}$$

Now we can write a finite difference method for the nonlinear advection equation
$u_t + f(u)_x = 0$ and, given some initial data $u(0, j) = g$ for $j = 0, \ldots, n$, solve it using:

$$u_{i+1,j} = u_{i,j} - \frac{\Delta t}{\Delta x}(f(u_{i,j}) - f(u_{i,j-1}))$$

Now we will write the third order Taylor series where $x_{i-2} = x_i - 2h$ and $x_{i-3} = x_i - 3h$:

$$y(x_{i-3}) = y(x_i) - 3hy'(x_i) + \frac{9h^2}{2}y''(x_i) - \frac{27h^3}{6}y'''(x_i) \tag{3}$$

$$y(x_{i-2}) = y(x_i) - 2hy'(x_i) + \frac{4h^2}{2}y''(x_i) - \frac{8h^3}{6}y'''(x_i) \tag{4}$$

$$y(x_{i-1}) = y(x_i) - hy'(x_i) + \frac{h^2}{2}y''(x_i) - \frac{h^3}{6}y'''(x_i) \tag{5}$$

$$y(x_i) = y(x_i) \tag{6}$$

Using the equations above, we want to take a linear combination $a \cdot (3) + b \cdot (4) + c \cdot (5) + d \cdot (6)$ to get an approximation for $y'''(x_i)$. Adding the equations above and combining like terms, we get:

$$\begin{aligned} ay(x_{i-3}) + by(x_{i-2}) + cy(x_{i-1}) + dy(x_i) &= (a + b + c + d)y(x_i) \\ &+ (-3a - 2b - c)hy'(x_i) + (9a + 4b + c)\frac{h^2}{2}y''(x_i) + (-27a - 8b - c)\frac{h^3}{6}y'''(x_i) \end{aligned}$$

Using the coefficients of the left hand side above, we get a system of four equations and four unknowns:

$$\begin{aligned} a + b + c + d &= 0 \\ -3a - 2b - c &= 0 \\ 9a + 4b + c &= 0 \\ -27a - 8b - c &= 6 \end{aligned}$$

Solving, we find $a = -1$, $b = 3$, $c = -3$, $d = 1$. Finally, we get:

$$y'''(x_i) = \frac{-y(x_{i-3}) + 3y(x_{i-2}) - 3y(x_{i-1}) + y(x_i)}{h^3}$$

Now we have a scheme for u_{xxx}:

$$u_{xxx} = \frac{u_{i,j} - 3u_{i,j-1} + 3u_{i,j-2} - u_{i,j-3}}{\Delta x^3}$$

Using Taylor series to rewrite each term, we can write a formulation for the KdV equation $u_t + f(u)_x + u_{xxx} = 0$ as:

$$u_{i+1,j} = u_{i,j} - \frac{\Delta t}{\Delta x}(f(u_{i,j}) - f(u_{i,j-1})) - \frac{\Delta t(u_{i,j} - 3u_{i,j-1} + 3u_{i,j-2} - u_{i,j-3})}{\Delta x^3}$$

2.2 Implementation

We will implement our method different ways in Octave. The first implementation uses the built-in function `filter` to compute finite differences. Then we will write a matrix formulation of the equation above and solve the problem using difference matrices in Octave. Finally, we will increase the efficiency our code by implementing sparse matrices.

2.2.1 Using filter

We will implement our method in Octave using the built-in function `filter` to compute the finite difference formula above. The command `y = filter(b, a, x)` returns the solution to the difference equation:

$$\sum_{k=0}^{N} a(k+1)y(n-k) = \sum_{k=0}^{M} b(k+1)x(n-k)$$

with N = length(a) - 1 and M = length(b) - 1. We write the backward difference method to compute $f(u)_x$ in Octave using the code:

```
a = [1];
b = [1 -1];
f_x = filter(b, a, f);
```

Checking with the summation expression above, we are setting $f'(n) = f(n) - f(n-1)$, as desired. Similarly, we write the difference method to compute u_{xxx} as:

```
c = [1];
d = [1 -3 3 -1];
u_xxx = filter(d, c, u);
```

Again checking that we are using `filter` correctly, we are computing:

$$u'''(n) = u(n) - 3u(n-1) + 3u(n-2) - u(n-3)$$

Finally, we can solve the KdV equation using the time-stepping scheme:

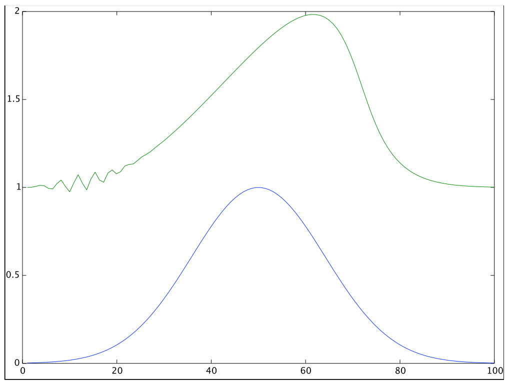


Figure 1: Resulting plot of KdV solution with Gaussian initial data using `filter` with $dt = 0.05$, $dx = 0.9$, and $nts = 40$.

```
u = u - (dt/dx)*filter(b, a, f) - (dt/(dx*dx*dx))*filter(d, c, u)
```

A complete Octave code to solve KdV using `filter` and resulting plot is given below.

```
% Finite difference time-stepping for 1-D KdV equation
% u_t + (3u^2)_x + u_xxx = 0

% Parameters
dx=.9;
dt=.05;
nts=40;
a=[1];
b=[1 -1];
c=[1];
d=[1 -3 3 -1];

r=1:dx:100;
% Gaussian initial data:
u0=1*exp(-(.05*(r-50)).^2);
u=u0;

% Computing finite differences:
for k=1:nts
   u=u-(dt/dx)*filter(b, a, 3*(u .* u))-(dt/(dx*dx*dx))*filter(d, c, u);
end
plot(r,u0,r,u+1)
```

2.2.2 Using difference matrices

Let us again consider a first order backward difference and write $y' = \frac{y_i - y_{i-1}}{h}$. If y is represented as a column vector, we can also write: $y' = \frac{1}{h}Dy$, where D is the difference matrix given by:

$$D = \begin{bmatrix} 1 & 0 & 0 & \cdots & 0 \\ -1 & 1 & 0 & \cdots & 0 \\ 0 & -1 & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \ddots & \vdots \\ 0 & 0 & \cdots & -1 & 1 \end{bmatrix}$$

Then to calculate $f(u)_x = \frac{f(u_{i,j})-f(u_{i,j-1})}{\Delta x}$, we can write:

$$f(u)_x = \frac{1}{\Delta x} \begin{bmatrix} 1 & 0 & 0 & \cdots & 0 \\ -1 & 1 & 0 & \cdots & 0 \\ 0 & -1 & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \ddots & \vdots \\ 0 & 0 & \cdots & -1 & 1 \end{bmatrix} \begin{bmatrix} f(u_{i,1}) \\ \vdots \\ f(u_{i,j-1}) \\ f(u_{i,j}) \\ \vdots \\ f(u_{i,n}) \end{bmatrix}$$

Similarly, we can use a matrix formulation to write $y''' = \frac{1}{h^3}D^3y$. Then for u_{xxx} we have:

$$u_{xxx} = \frac{1}{\Delta x^3} \begin{bmatrix} 1 & 0 & 0 & 0 & \cdots & 0 \\ -3 & 1 & 0 & 0 & \cdots & 0 \\ 3 & -3 & 1 & 0 & \cdots & 0 \\ -1 & 3 & -3 & 1 & \cdots & 0 \\ \vdots & \ddots & \ddots & \ddots & \ddots & \vdots \\ 0 & 0 & \cdots & 3 & -3 & 1 \end{bmatrix} \begin{bmatrix} u_{i,1} \\ \vdots \\ u_{i,j-3} \\ u_{i,j-2} \\ u_{i,j-1} \\ u_{i,j} \\ \vdots \\ u_{i,n} \end{bmatrix}$$

Altogether, we have a time-stepping scheme:

$$\begin{bmatrix} u_{i+1,1} \\ \vdots \\ u_{i+1,j} \\ \vdots \\ u_{i+1,n} \end{bmatrix} = \begin{bmatrix} u_{i,1} \\ \vdots \\ u_{i,j} \\ \vdots \\ u_{i,n} \end{bmatrix} - \frac{\Delta t}{\Delta x} D \begin{bmatrix} f(u_{i,1}) \\ \vdots \\ f(u_{i,j}) \\ \vdots \\ f(u_{i,n}) \end{bmatrix} - \frac{\Delta t}{\Delta x^3} D^3 \begin{bmatrix} u_{i,1} \\ \vdots \\ u_{i,j} \\ \vdots \\ u_{i,n} \end{bmatrix}$$

In our code, however, u is a row vector, so we take the transpose of the matrices defined above to compute finite differences. For the first difference, then, we have:

$$f(u)_x = \frac{1}{\Delta x} \begin{bmatrix} f(u_{i,1}) & \cdots & f(u_{i,n}) \end{bmatrix} \begin{bmatrix} 1 & -1 & 0 & \cdots & 0 \\ 0 & 1 & -1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 & -1 \\ 0 & 0 & \cdots & 0 & 1 \end{bmatrix}$$

In Octave, we populate the first order difference matrix `fod` and the third order `tod` and solve KdV using the time stepping scheme:

```
u = u - ((dt/dx)*(3*(u .* u))*fod) - ((dt/(dx*dx*dx))*u*tod)
```

A partial code is given below.

```
% Creating a first order difference matrix fod
fod=eye(nr);
for k=1:(nr-1)
        fod(k, k+1)=-1;
end

% Creating a third order difference matrix tod
tod=eye(nr);
for k=1:(nr-1)
        tod(k, k+1)=-3;
end
for k=1:(nr-2)
        tod(k, k+2)=3;
end
for k=1:(nr-3)
        tod(k, k+3)=-1;
end

% Computing finite differences:
for k=1:nts
        u=u-((dt/dx)*(3*(u .* u))*fod)-((dt/(dx*dx*dx))*u*tod);
end
```

2.2.3 Using sparse matrices

The implementation given above is not efficient. Instead, we will implement sparse matrices and compute finite differences using sparse linear algebra in Octave. We can partially solve the efficiency problem by converting the matrices defined above using the function `sparse` before computing finite differences:

```
fod = sparse(fod);
tod = sparse(tod);
```

However, we need not construct these full matrices `fod` and `tod`. Instead, we will initially create the sparse difference matrices we need to compute finite differences. The command `S = sparse(i, j, s)` uses the vectors `i`, `j` and `s` (all with same length) to generate a sparse matrix so that:

$$S(i(k), j(k)) = s(k)$$

where $k = 1, \ldots, \text{length}(i)$. A program for creating the sparse difference matrices `tod` and `fod` is given below with identical output as `filter` above.

```
% Creating a sparse first order difference matrix fod
```

```
kc=0;
for k=1:nr
        kc=kc+1;
        rindx(kc)=k;
        cindx(kc)=k;
        val(kc)=1;
end
for k=1:(nr-1)
        kc=kc+1;
        rindx(kc)=k;
        cindx(kc)=k+1;
        val(kc)=-1;
end
fod=sparse(rindx, cindx, val);

% Creating a sparse third order difference matrix tod
kc=0
for k=1:nr
        kc=kc+1;
        rindx2(kc)=k;
        cindx2(kc)=k;
        val2(kc)=1;
end
for k=1:(nr-1)
        kc=kc+1;
        rindx2(kc)=k;
        cindx2(kc)=k+1;
        val2(kc)=-3;
end
for k=1:(nr-2)
        kc=kc+1;
        rindx2(kc)=k;
        cindx2(kc)=k+2;
        val2(kc)=3;
end
for k=1:(nr-3)
        kc=kc+1;
        rindx2(kc)=k;
        cindx2(kc)=k+3;
        val2(kc)=-1;
end
tod=sparse(rindx2, cindx2, val2);
```

2.3 Results

We use the analytical solution (2) to KdV as initial data with varying parameters:

```
% Soliton initial data:
c=1;
u0=(c/2)*sech((sqrt(c)/2)*r).^2;
```

Because the number of time steps will vary depending on our choice of dt, we modify the parameters of our code:

```
% Parameters:
dx=.01;
dt=.01;
T=2; % total time
nts=T/dt;
```

We calculate the maximum relative error of our computed value for u using the code:

```
 max(abs(u-exact)./exact)
```

where `exact` is the analytical solution of KdV after a time T.
Because our direct formulation includes division by Δx^3, we cannot decrease this parameter by much. Table 1 shows the calculated maximum relative and absolute error and Figure 3 shows our calculated solution.

dx	relative error	absolute error
1	412.98	0.039905
0.9	1242.2	0.054803
0.8	6892.0	0.067737
0.7	1.1862e+05	0.16609
0.6	1.3426e+07	1.0894
0.5	Inf	Inf

Table 1: Maximum relative and absolute error for KdV with $c = 0.5$, $T = 1$, and $dt = 0.01$.

3 A parallel numerical solution

In this section, we will use a well known algorithm to implement a difference scheme for KdV in parallel. We will discuss the effectiveness of the algorithm for our chosen scheme after comparing a computational model with experimental results.

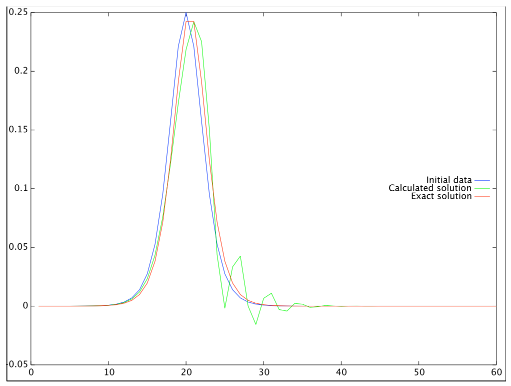


Figure 2: KdV after $T = 1$ with $dt = 0.01$ and $dx = 1$.

3.1 Overview of well studied methods

The authors of [12] compiled and tested a variety of numerical methods to solve KdV. We revisit the methods and certain results here. Note that the superscript j represents the time variable and the space variable is denoted by the subscript i so that $u_i^j = u(i\Delta x, j\Delta t)$.

1. Explicit scheme (Zabusky and Kruskal)

   Zabusky and Kruskal developed the explicit leapfrog finite difference scheme:

$$\begin{aligned} u_i^{j+1} = u_i^{j-1} - 2\frac{\Delta t}{\Delta x}\left(u_{i+1}^j + u_i^j + u_{i-1}^j\right)\left(u_{i+1}^j - u_{i-1}^j\right) \\ - \frac{\Delta t}{(\Delta x)^3}\left(u_{i+2}^j - 2u_{i+1}^j + 2u_{i-1}^j - u_{i-2}^j\right). \end{aligned}$$

2. Hopscotch Method

   The Hopscotch Method is a mixed implicit/explicit scheme given by:

$$\begin{aligned} u_i^{j+1} = u_i^j - 3\frac{\Delta t}{\Delta x}\left(f_{i+1}^j - f_{i-1}^j\right) \\ - \frac{\Delta t}{2(\Delta x)^3}\left(u_{i+2}^j - 2u_{i+1}^j + 2u_{i-1}^j - u_{i-2}^j\right) \end{aligned} \tag{7}$$

$$\begin{aligned} u_i^{j+1} = u_i^j - 3\frac{\Delta t}{\Delta x}\left(f_{i+1}^{j+1} - f_{i-1}^{j+1}\right) \\ - \frac{\Delta t}{2(\Delta x)^3}\left(u_{i+2}^{j+1} - 2u_{i+1}^{j+1} + 2u_{i-1}^{j+1} - u_{i-2}^{j+1}\right) \end{aligned} \tag{8}$$

   where $f = \frac{u^2}{2}$ and (7) is used to calculate $(i + j)$ even, (8) for $(i + j)$ odd.

3. Implicit scheme (Goda)

   An implicit finite difference scheme to solve KdV from Goda is given by:

$$\frac{1}{\Delta t}\left[u_i^{j+1} - u_{i+1}^{j}\right] + \frac{1}{\Delta x}\left[u_{i+1}^{j+1}\left(u_i^j + u_{i+1}^j\right) - u_{i-1}^{j+1}\left(u_i^j + u_{i-1}^j\right)\right]$$
$$+ \frac{1}{2(\Delta x)^3}\left[u_{i+2}^{j+1} - 2u_{i+1}^{j+1} + 2u_{i-1}^{j+1} - u_{i-2}^{j+1}\right] = 0$$

where the quasi-pentagonal system of equations shown below must be solved at each time step.

$$\begin{bmatrix}
x & x & x & & & & & x & x \\
x & x & x & x & & & & & x \\
x & x & x & x & x & & & & \\
 & x & x & x & x & x & & & \\
 & & & & & & & & \\
 & & & & x & x & x & x & x \\
x & & & & & x & x & x & x \\
x & x & & & & & x & x & x
\end{bmatrix}$$

4. The Proposed Scheme

The following scheme is based on the inverse scattering transform:

$$\begin{aligned}
\frac{1}{\Delta t}\left[u_i^{j+1} - u_i^j\right] =& \frac{1}{2(\Delta x)^3}\left[u_{i-1}^{j+1} - 3u_i^{j+1} + 3u_{i+1}^{j+1} - u_{i+2}^{j+1} + u_{i-2}^j - 3u_{i-1}^j + 3u_i^j - u_{i+1}^j\right] \\
&- \frac{3}{2\Delta x}\left[\left(u_i^j\right)^2 - \left(u_i^{j+1}\right)^2\right] \\
&- \frac{1}{2\Delta x}\left[u_{i+1}^{j+1}\left(u_i^{j+1} + u_{i+1}^{j+1} + u_{i+2}^{j+1}\right) - u_{i-1}^j\left(u_i^j + u_{i-1}^j + u_{i-2}^j\right)\right]
\end{aligned}$$

5. M. Kruskal

Kruskal suggested the following scheme for $u_t + u_{xxx} = 0$:

$$\begin{aligned}
\frac{u_i^{j+1} - u_i^j}{\Delta t} =& \frac{u_{i-1}^{j+1} - 3u_i^{j+1} + 3u_{i+1}^{j+1} - u_{i+2}^{j+1}}{2(\Delta x)^3} \\
&+ \frac{u_{i-2}^j - 3u_{i-1}^j + 3u_i^j - u_{i+1}^j}{2(\Delta x)^3}
\end{aligned}$$

and used the following to solve KdV:

$$\begin{aligned}\frac{u_i^{j+1}-u_i^j}{\Delta t}&+\frac{3\theta}{4\Delta x}\left[\left(u_{i+1}^{j+1}\right)^2-\left(u_{i-1}^{j+1}\right)^2+\left(u_{i+1}^{j}\right)^2-\left(u_{i-1}^{j}\right)^2\right]\\&+\frac{1-\theta}{2\Delta x}\left[u_i^{j+1}\left(u_{i+1}^{j+1}-u_{i-1}^{j+1}\right)-u_i^j\left(u_{i+1}^j-u_{i-1}^j\right)\right]\\&+\frac{u_{i-1}^{j+1}-3u_i^{j+1}+3u_{i+1}^{j+1}-u_{i+2}^{j+1}}{2(\Delta x)^3}\\&+\frac{u_{i-2}^{j}-3u_{i-1}^{j}+3u_{i}^{j}-u_{i+1}^{j}}{2(\Delta x)^3}=0.\end{aligned}$$

Experimentally, it was found that $\theta=\frac{2}{3}$ gave the best results.

6. Split Step Fourier Method by Tappert

Note that for this and the next method we will use the superscript m to represent the time variable and the subscript n for the space variable, since both use the imaginary number, i. A Fourier Method for KdV can be derived by making the change of variables $Y=(x+p)\frac{\pi}{p}$, where p is half the length of the interval of calculation, to normalize the spatial period to $[0,2\pi]$ and discretizing by N points so that $\Delta Y=\frac{2\pi}{N}$. KdV, then, becomes:

$$u_t+6\frac{\pi}{p}uu_Y+\frac{\pi^3}{p^3}u_{YYY}=0.$$

The transformation into discrete Fourier space for $k=-\frac{N}{2},\dots,\frac{N}{2}-1$ is given by:

$$\hat{u}(k,t)=Fu=\frac{1}{\sqrt{N}}\sum_{j=1}^{N-1}u(j\Delta Y,t)e^{-2\pi ijk/N}$$

and the inverse transform for $k=-\frac{N}{2},\dots,\frac{N}{2}-1$:

$$u(j\Delta Y,t)=F^{-1}\hat{u}=\frac{1}{\sqrt{N}}\sum_k\hat{u}(k,t)e^{2\pi ijk/N}.$$

The Slip Step Fourier Method uses a finite difference method to solve $u_t+6\frac{\pi}{p}uu_Y$ followed by the fast Fourier transform to advance the solution using the linear term u_{YYY}, described below.

$$\begin{aligned}\tilde{u}_n^{m+1}=u_n^m-\frac{\Delta t}{8\Delta Y}\frac{\pi}{p}\Bigg[&\left(u_{n+1}^{m+1}\right)^2-8\left(u_{n-1}^{m+1}\right)^2-\left(u_{n+2}^{m+1}\right)^2+\left(u_{n-2}^{m+1}\right)^2\\&+8\left(u_{n+1}^{m}\right)^2-8\left(u_{n-1}^{m}\right)^2-\left(u_{n+2}^{m}\right)^2+\left(u_{n-2}^{m}\right)^2\Bigg]\end{aligned}$$

followed by the fast Fourier transform:

$$u(Y_j, t + \Delta t) = F^{-1}\left(e^{ik^3\pi^3/p^3\Delta t} F\tilde{u}(Y_j, t)\right).$$

7. A Pseduospectral Method by Fornberg and Whitham

   This pseudospectral method uses the fast Fourier transform is used to evaluate $u_Y = F^{-1}(ikFu)$ and $u_{YYY} = F^{-1}(-ik^3Fu)$. Combined with a leap-frog scheme, the approximation becomes:

$$\begin{aligned} u(Y, t+\Delta t) - u(Y, t-\Delta t) \\ + 2i\frac{6\pi}{p}\Delta t u(Y,t)F^{-1}(kFu) - 2i\Delta t\frac{\pi^3}{p^3}F^{-1}(k^3Fu) = 0. \end{aligned}$$

   Fornberg and Whitham substituted $-2iF^{-1}\left(sin\left(\Delta t\frac{\pi^3k^3}{p^3}\right)Fu\right)$ for the last term above.

	Δx	Δt	L_∞	time
Zabusky	0.1739	0.002	0.00469	28
Hopscotch	0.2	0.003	0.00472	23
Goda	0.1	0.002	0.00492	244
Proposed	0.16	0.125	0.00173	7
Kruskal	0.08	0.04	0.00453	24
Tappert	0.3125	0.004	0.00494	63
Fornberg	0.625	0.0096	0.00113	12

Figure 3: Computation time in seconds for tolerance < 0.005 after $T = 1.0$ and wave amplitude 1

	Δx	Δt	L_∞	time
Zabusky	0.08	0.0019	0.00930	591
Hopscotch	0.1	0.0005	0.00994	272
Goda	0.04	0.00025	0.001282	3568
Proposed	0.1	0.1	0.00332	23
Kruskal	0.04	0.011	0.00952	163
Tappert	0.156	0.002	0.00943	271
Fornberg	0.3125	0.0042	0.00474	40

Figure 4: Computation time in seconds for tolerance < 0.01 after $T = 1.0$ and wave amplitude 2

Various tests were performed to analyze the goodness of the methods described above. A selection of them are included here in Figures 3, 4, and 5.

	Δx	Δt	L_∞	time
Zabusky	0.12	0.00066	0.00165	349
Hopscotch	0.13	0.001	0.00142	283
Goda	0.1	0.0005	0.00165	2764
Proposed	0.1	0.14	0.00148	19
Kruskal	0.08	0.015	0.00145	106
Tappert	0.15625	0.005	0.00186	322
Fornberg	0.625	0.0148	0.00107	24

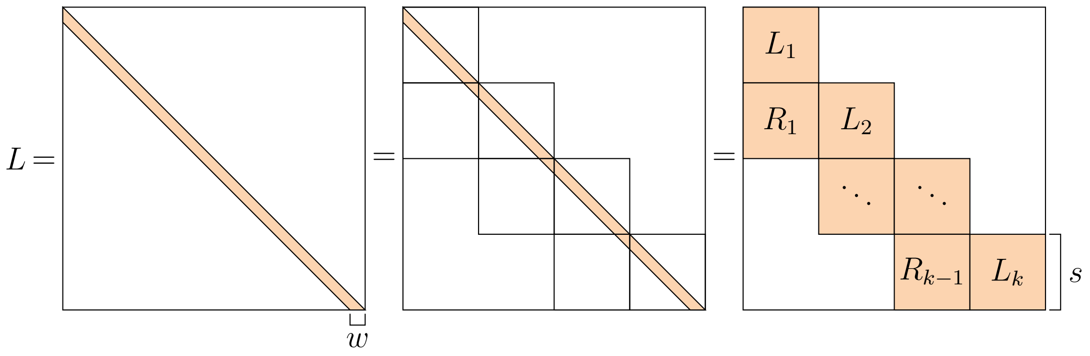
Figure 5: Computation time in seconds for tolerance < 0.002 after $T = 3.0$ with two interacting solitons as initial conditions with amplitudes 0.5 and 1

3.2 The block inverse algorithm

This section describes an algorithm that will allow us to solve the finite difference problem for the KdV equation in parallel.

Let A be an banded $n \times n$ matrix with bandwidth $2w - 1$. A can be factored and written as $A = LU$ where L is a lower triangular matrix and U is an upper triangular matrix, both with bandwidth w. We will solve the system $Ax = f$ in two steps by writing $LUx = f$. First we will solve $Ly = f$ for y, then $Ux = y$ for x, using the block inverse algorithm for both. The steps are described in detail below.

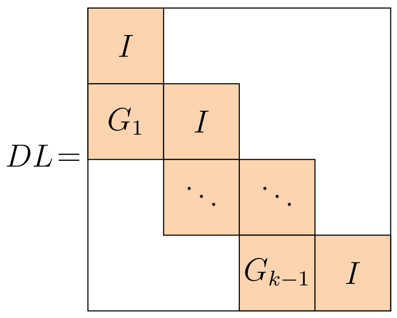
Let L be a lower triangular matrix with bandwidth w. Pick integers k and s so that $n = ks$ and $w \le s$. We can write L as a block matrix with k $s \times s$ blocks on the diagonal and $k - 1$ below the diagonal, shown below.

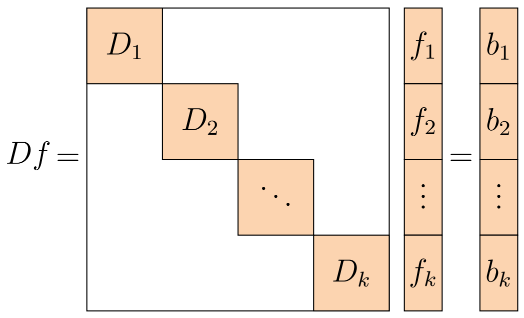


Assume that each L_i is invertible and let D be the $n \times n$ block diagonal matrix where

$$D_i = L_i^{-1}, \quad i = 1, \ldots, k.$$

Now, instead of the original system, we will solve $DLx = Df$. To do so, we multiply through by D and write f as a block vector, shown below.





Above, I is the $s \times s$ identity matrix and

$$L_{i+1}G_i = R_i, \quad i = 1, \dots, k-1 \tag{9}$$

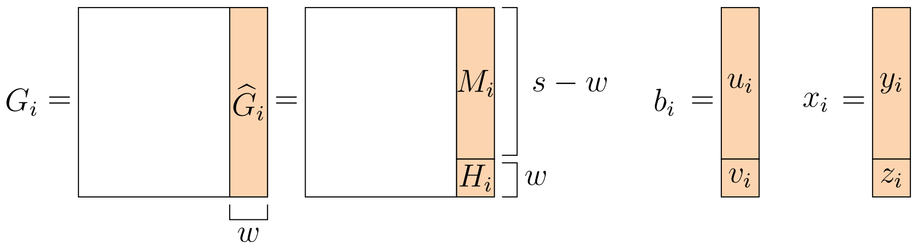
$$L_i b_i = f_i, \quad i = 1, \dots, k. \tag{10}$$

Since L_i is lower triangular for all i, solving (9) and (10) is a matter of forward substitution.

Note that the first $s - w$ columns of R_i are zero for all i and that the same must be true of G_i. We can simplify the computation of (9) by only solving for nonzero entries. Let $\widehat{G}_i$ be the rightmost w columns of G_i and $\widehat{R}_i$ be the rightmost w columns of R_i. Then (9) becomes:

$$L_{i+1}\widehat{G}_i = \widehat{R}_i, \quad i = 1, \dots, k-1 \tag{11}$$

We further decompose $\widehat{G}_i$ by letting M_i be the top $s - w$ rows of $\widehat{G}_i$, and H_i the bottom w rows of $\widehat{G}_i$, shown below. We can write b_i as u_i and v_i, and x_i as y_i and z_i in a similar fashion.



Now we have simple equations to the system $DLx = b$. The first block of x is straightforward:

$$y_1 = u_1, \quad z_1 = v_1$$

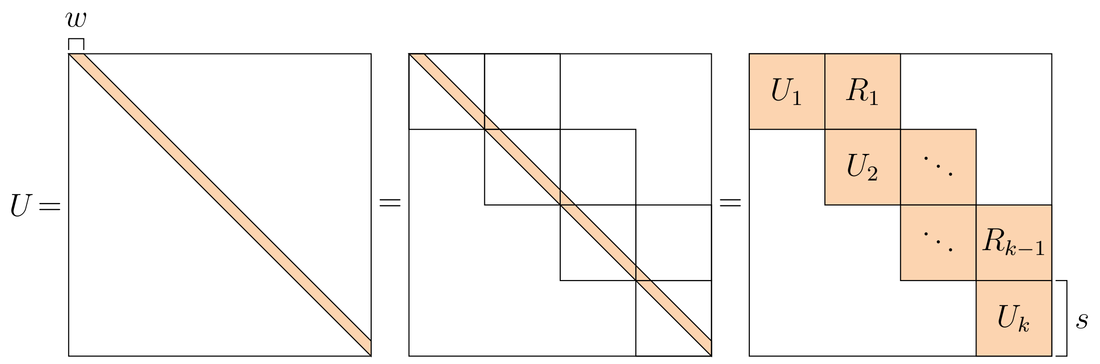
and the subsequent blocks are given by:

$$z_{i+1} = v_{i+1} - H_i z_i, \quad i = 1, \dots, k-1 \tag{12}$$

$$y_{i+1} = u_{i+1} - M_i z_i, \quad i = 1, \dots, k-1. \tag{13}$$

The z_i's in equation (12) above will be computed sequentially, but equation (13) gives us an opportunity to compute the y_i's in parallel. Note that there is a possibility for parallelism in (12) using the parallel prefix algorithm as suggested in [13] if w is large enough.

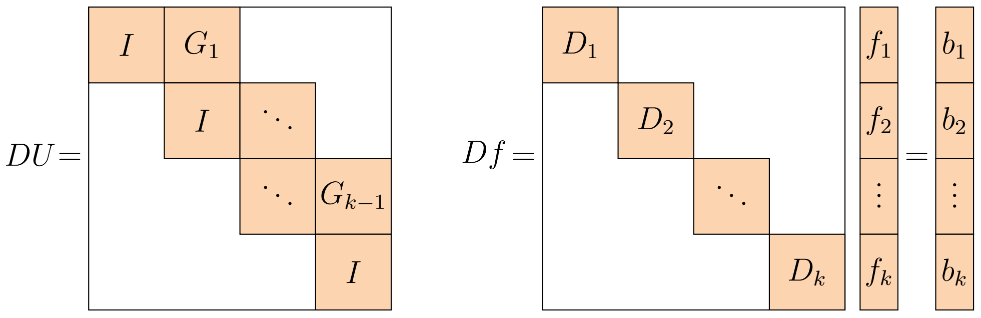
Conversely, let U be an upper triangular matrix with bandwidth w. Let k and s be as before, so that $n = ks$ and $w \leq s$ and write U as a block matrix shown below.



Assume each U_i is invertible and D is the block diagonal matrix where

$$D_i = U_i^{-1}, \quad i = 1, \ldots, k.$$

Now, instead of $Ux = f$, we will solve $DUx = Df$ following the same steps as before, shown below.

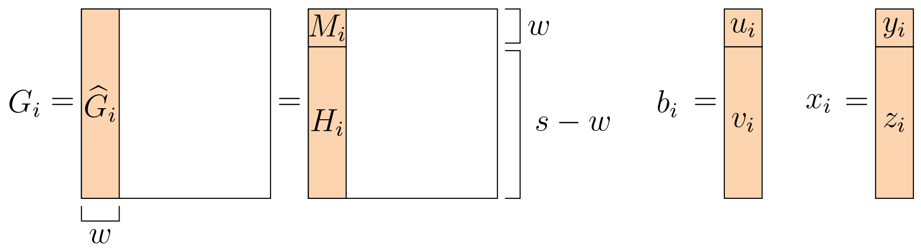


Again, I is the identity matrix and

$$U_i G_i = R_i, \quad i = 1, \ldots, k-1 \tag{14}$$
$$U_i b_i = f_i, \quad i = 1, \ldots, k. \tag{15}$$

Here, since U_i is upper triangular for all i, we will use backwards substitution to solve 14 and 15.

Note that now the last $s - w$ columns of R_i are zero for all i and that the same must be true of G_i. We will again decompose the problem and only solve for nonzero entries of G_i ($\widehat{G}_i$). Let M_i the top w rows of $\widehat{G}_i$, and H_i the bottom $s - w$ rows of $\widehat{G}_i$, shown below. We can write b_i as u_i and v_i, and x_i as y_i and z_i similarly, shown below.



Now we have simple equations to the system $DUx = b$ backwards. The last block of x is straightforward:

$$y_k = u_k, \quad z_k = v_k$$

and the subsequent blocks are given by:

$$y_i = u_i - M_i y_{i+1}, \quad i = 1, \ldots, k-1 \tag{16}$$

$$z_i = v_i - H_i y_{i+1}, \quad i = 1, \ldots, k-1. \tag{17}$$

This time, the y_i's in equation (16) above will be computed sequentially, but equation (17) lets us compute the z_i's in parallel.

3.3 Implementation

An implicit finite difference scheme to solve KdV from Goda proposed in [3] is given by:

$$\frac{1}{\Delta t}\left[u_i^{j+1} - u_{i+1}^{j}\right] + \frac{1}{\Delta x}\left[u_{i+1}^{j+1}\big(u_i^j + u_{i+1}^j\big) - u_{i-1}^{j+1}\big(u_i^j + u_{i-1}^j\big)\right]$$
$$+ \frac{1}{2(\Delta x)^3}\left[u_{i+2}^{j+1} - 2u_{i+1}^{j+1} + 2u_{i-1}^{j+1} - u_{i-2}^{j+1}\right] = 0$$

which can be written as the pentadiagonal system:

$$\begin{bmatrix} 1 & c & d & & & & \\ b & 1 & c & d & & & \\ a & b & 1 & c & \ddots & & \\ & a & b & 1 & \ddots & \ddots & \\ & & \ddots & \ddots & \ddots & \ddots & d \\ & & & \ddots & \ddots & \ddots & c \\ & & & & a & b & 1 \end{bmatrix} \begin{bmatrix} u_1^{j+1} \\ u_2^{j+1} \\ \vdots \\ \\ \\ \\ u_n^{j+1} \end{bmatrix} = \begin{bmatrix} u_1^{j} \\ u_2^{j} \\ \vdots \\ \\ \\ \\ u_n^{j} \end{bmatrix}$$

where

$$a = -\frac{\Delta t}{2(\Delta x)^3}$$
$$b = \frac{\Delta t}{(\Delta x)^3} - \frac{\Delta t}{\Delta x}\big(u_i^j + u_{i-1}^j\big)$$
$$c = -\frac{\Delta t}{(\Delta x)^3} + \frac{\Delta t}{\Delta x}\big(u_i^j + u_{i+1}^j\big)$$
$$d = \frac{\Delta t}{2(\Delta x)^3}.$$

The algorithm described above was implemented in C++ using MPI where A was set to be the finite difference matrix given by Goda described before. We chose Goda's scheme because [12] found it to have prohibitively large computation times in some cases but the scheme has truncation error $O(\Delta t) + O((\Delta x)^2)$ and is unconditionally stable.

Large matrices were implemented as $3 \times m$ arrays, where m is the number of elements in the matrix and each row in the array contains the row, column, and value of an element in the matrix. Parallel factorization of banded matrices has been studied in [14], [15], and [16], but here A was factored by one processor using a modified Dolittle's algorithm for a pentadiagonal matrix. The problem was then solved as described above using the block inverse algorithm with k processors. The details of the algorithm for $Ly = f$ are presented in Algorithm 1 below and solving $Ux = y$ follows similarly.

Algorithm 1 Block inverse algorithm to solve $Lx = f$

```
for $i = 0, \dots, k-1$ do
    $P_i$ initializes $f_i$
for $t = 1, \dots, T$ do
    $P_0$ initializes $A$
    $P_0$ factors $A$ into $L$ and $U$
    for $i = 1, \dots, k-1$ do
        $P_0$ sends $L_i, R_{i-1}$ to $P_i$
        $P_i$ solves for $G_{i-1}$ where $L_i G_{i-1} = R_{i-1}$
    for $i = 0, \dots, k-1$ do
        $P_i$ solves for $b_i$ where $L_i b_i = f_i$
    $P_0$ computes for $z_0$ where $z_0 = v_0$
    $P_0$ sends $z_0$ to $P_1$
    for $i = 1, \dots, k-2$ do
        $P_i$ receives $z_{i-1}$ from $P_{i-1}$
        $P_i$ computes $z_i$ where $z_i = v_i - H_i z_{i-1}$
        $P_i$ sends $z_i$ to $P_{i+1}$
    $P_{k-1}$ receives $z_{k-2}$ from $P_{k-2}$
    $P_{k-1}$ computes $z_{k-1}$
    for $i = 0, \dots, k-1$ do
        $P_i$ computes $y_i$ where $y_1 = u_i - M_i z_{i-1}$
        $P_i$ resets $f_i$ for next timestep
```

3.4 Performance Analysis

3.4.1 Computational model

We can count the amount of storage used and the number of operations done in our program to develop a computational model of the algorithm. Let n be the number of grid points used in the finite difference scheme so that A is an $n \times n$ matrix. We run our program on k processors so that the block matrix size is $s = n/k$. The block inverse algorithm uses $10k + 8$

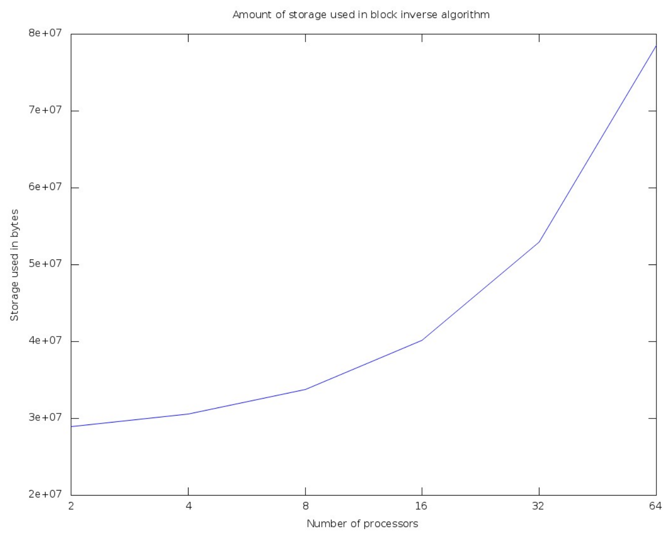


Figure 6: Total storage used for $n = 100,000$.

integers and $(29s + 3 + n)k + 33n - 36 + (k - 1)s$ doubles so that the total number of bytes used is:

$$4[10k + 8] + 8[(29s + 3 + n)k + 33n - 36 + (k - 1)s].$$

For a fixed sized problem, then, the amount of storage scales linearly with k, shown in Figure 6. Let T_1, T_2, T_3, T_4, T_5, T_{send}, T_{rec} be the amount of time it takes to perform an assignment, multiplication, division, addition, subtraction, MPI send, and MPI receive, respectively. We break each timestep into a series of "tasks" that we time separately and the amount of computation for each is given below. P_0 and P_{k-1} are handled separately from the other processors since they behave differently.

P_0	P_i	P_{k-1}
$(46n + 18sk - 47)T_1$ $+[43n + 12s(k - 1) - 55]T_2$ $+(8n - 11)T_3$ $+(46n + 18sk - 47)T_4$ $+(46n + 18sk - 47)T_5$ $+(46n + 18sk - 47)T_{send}$	$(18s)T_{rec}$	$(18s - 9)T_{rec}$

	$(18s)T_1$ $+(6s)T_2$ $+(3s)T_3$ $+(3s)T_4$ $+(66s)T_5$	$(18s)T_1$ $+(6s)T_2$ $+(3s)T_3$ $+(3s)T_4$ $+(66s)T_5$
$(6s)T_1$ $+(2s)T_2$ $+(s)T_3$ $+(s)T_4$ $+(2s+6)T_5$	$(6s)T_1$ $+(2s)T_2$ $+(s)T_3$ $+(s)T_4$ $+(2s+6)T_5$	$(6s)T_1$ $+(2s)T_2$ $+(s)T_3$ $+(s)T_4$ $+(2s+6s)T_5$
$(3)T_1$ $+(3)T_{send}$	$(15i+3)T_1$ $+(9i)T_2$ $+(9i)T_4$ $+(3i)T_5$ $+(3i)T_{rec}$ $+[3(i+1)]T_{send}$	$[15(k-1)+3]T_1$ $+[9(k-1)]T_2$ $+[9(k-1)]T_4$ $+[3(k-1)]T_5$ $+[3(k-1)]T_{rec}$ $+[3(k-1)]T_{send}$
$(s-3)T_1$	$(5s-15)T_1$ $+(3s-9)T_2$ $+(3s-9)T_4$ $+(s-3)T_5$	$(5s-15)T_1$ $+(3s-9)T_2$ $+(3s-9)T_4$ $+(s-3)T_5$
$(2s)T_1$ $+(3)T_4$ $+(3)T_5$	$(2s)T_1$ $+(3)T_4$ $+(3)T_5$	$(2s)T_1$ $+(3)T_4$ $+(3)T_5$

$(15s-6)T_1$ $+(6s-9)T_2$ $+(3s)T_3$ $+(3)T_4$ $+(9s-18)T_5$	$(15s-6)T_1$ $+(6s-9)T_2$ $+(3s)T_3$ $+(3)T_4$ $+(9s-18)T_5$	
$(6s)T_1$ $+(2s)T_2$ $+(s)T_3$ $+(s)T_4$ $+(3s-3)T_5$	$(6s)T_1$ $+(2s)T_2$ $+(s)T_3$ $+(s)T_4$ $+(3s-3)T_5$	$(6s)T_1$ $+(2s)T_2$ $+(s)T_3$ $+(s)T_4$ $+(3s-3)T_5$
$[15(k-1)+3]T_1$ $+[9(k-1)]T_2$ $+[9(k-1)]T_4$ $+[3(k-1)]T_5$ $+[3(k-1)]T_{rec}$ $+[3(k-1)]T_{send}$	$[15(k-i-1)+3]T_1$ $+[9(k-i-1)]T_2$ $+[9(k-i-1)]T_4$ $+[3(k-i-1)]T_5$ $+[3(k-i-1)]T_{rec}$ $+[3(k-i)]T_{send}$	$(3)T_1$ $+(3)T_{send}$
$(5s-15)T_1$ $+(3s-9)T_2$ $+(3s-9)T_4$ $+(s-3)T_5$	$(5s-15)T_1$ $+(3s-9)T_2$ $+(3s-9)T_4$ $+(s-3)T_5$	$(s-3)T_1$
$(3s)T_1$ $+(k-1)T_2$ $+(s-3)T_4$ $+[s(k-1)]T_{rec}$	$(2s)T_1$ $+(s-3)T_4$ $+(s)T_{send}$	$(2s)T_1$ $+(s-3)T_4$ $+(s)T_{send}$

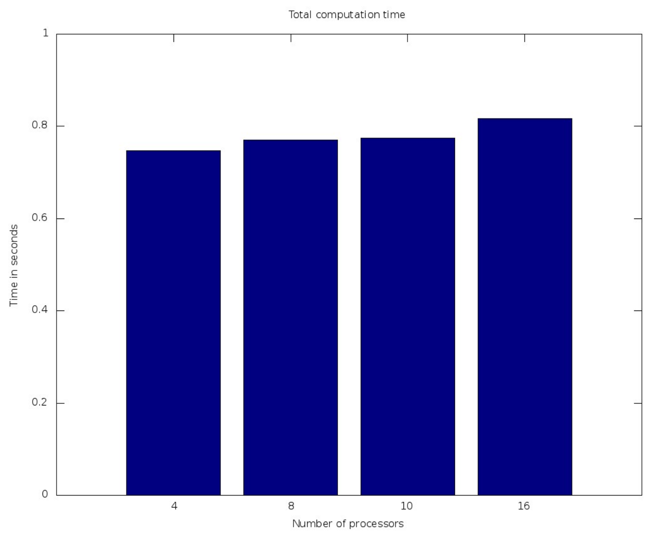


Figure 7: Total computation time on $k = 4, 8, 10, 16$ processors.

3.4.2 Discussion of results

The program was tested with the initial soliton $u(x, 0) = \frac{1}{2}\text{sech}^2\left(\frac{1}{2}x - 10\right)$ centered at $x = 20$ on the domain $[0, 40)$. The problem size was fixed with $n = 100,000$ grid points so that $\Delta x = 0.0004$ and was run for ten time steps with $\Delta t = 0.1$. It was tested on $k = 4, 8, 10,$ and 16 processors so that each processor solved an $s \times s$ system where $s = 25,000, 12,500, 10,000,$ and $6,250$, respectively. We can see in figure 7 that for this problem, increasing processor count and using the block inverse algorithm is not a particularly effective way to speed up the computation. We hypothesize why in the following discussion.

Figure 8 shows in detail the block inverse algorithm on four and eight processors during one timestep. The computation has been broken down into subtasks, each denoted by a

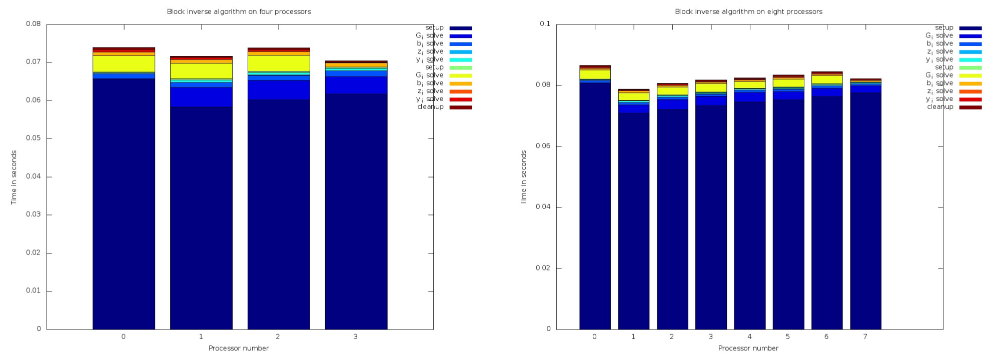


Figure 8: Computation time for one timestep on four processors (left) and eight (right).

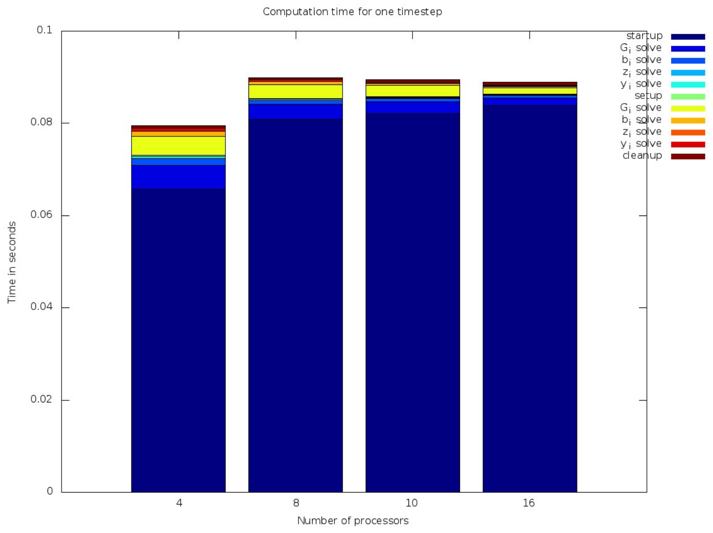


Figure 9: Total computation time on $k = 4, 8, 10, 16$ processors.

different color. For each task, the size of the bar is the length of computation. A description of the computation is given below.

1. During the "setup" of the problem (dark blue in Figure 8), P_0 factors A and collects and distributes blocks to $P_1, \ldots, P_{k-1}$.

2. All processors except for P_0 solve for G_i using equation (11). P_0 starts the next step.

3. All processors solve for b_i using equation (10).

4. Processor P_i waits for z_{i-1} from its neighbor P_{i-1}, computes z_i by (12) and sends it to P_{i+1}. P_0 immediately computes z_i and sends it to P_1.

5. All processors solve for y_i using (13).

6. The "setup" task resets the array used to store f_i so that code can be repeated.

7. $P_0, \ldots, P_{k-2}$ solve for G_i using (14). P_{k-1} starts the next step.

8. All processors solve for b_i using (15).

9. Processor P_i waits for z_{i+1} from P_{i+1}, computes z_i using (17) and sends it to P_{i-1}. P_{k-1} immediately computes z_i and sends it to P_{k-2}.

10. All processors solve for y_i by equation (16).

Figure 8 shows that most of the computation time is used factoring and distributing pieces of the matrix A. After each timestep, Goda's scheme updates the difference matrix A, so factoring at each timestep is unavoidable. This computation cannot be reduced by

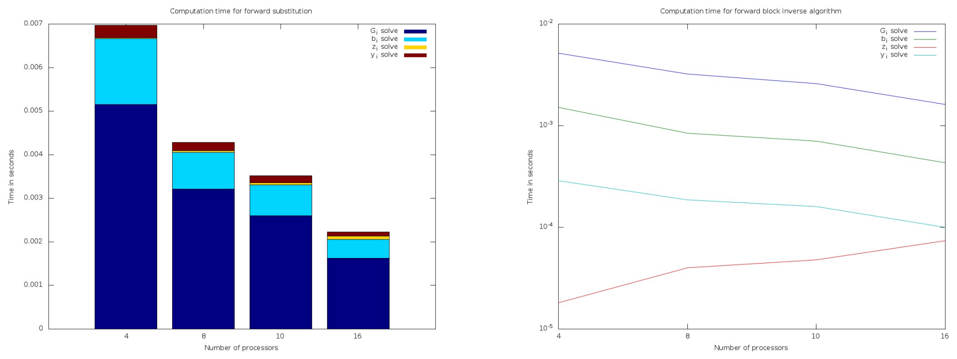


Figure 10: Forward block inverse algorithm on $k = 4, 8, 10, 16$ processors.

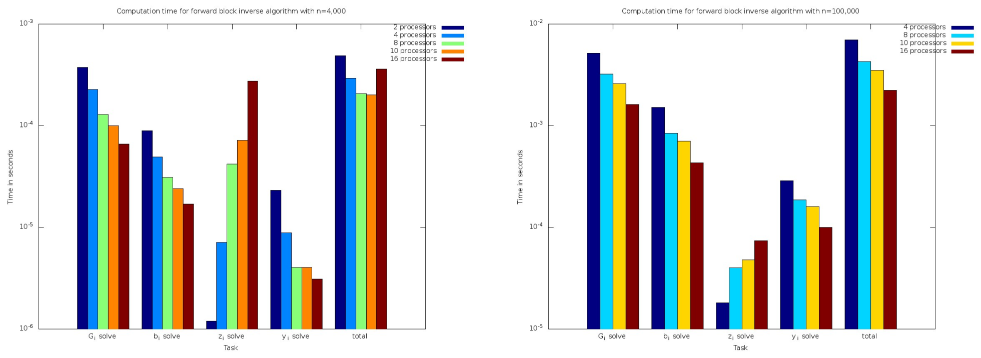


Figure 11: Computation time for problem size $n = 4,000$ (left) and $n = 100,000$ (right).

increasing the number of processors, but instead increases the length of this task by adding additional sends, shown in Figure 9.

If we restrict ourselves to solving $Ly = f$, the forward substitution part of our program and steps 2 through 5 above, we see that the block inverse algorithm works well in practice. Figure 10 shows clearly that speedup is gained by adding processors for a fixed problem size.

It is also worth mentioning that the size of the problem influences the effectiveness of the algorithm. We ran the program again on a smaller grid size $n = 4,000$ and found that the algorithm performs better on larger problems. Figure 11 shows the computation time for the forward part of the algorithm for both problem sizes. Each task can be done in less time using more processors (due to the fact that the block matrix size s is smaller), except for the ones involving communication. By increasing the problem size, large communication costs using more processors can be mitigated by a larger computation.

Finally, Figure 12 shows the computation time over all ten timesteps on P_1 and P_2 when $k = 4$. Here we can see that not much variance occurs between processors or over timesteps.

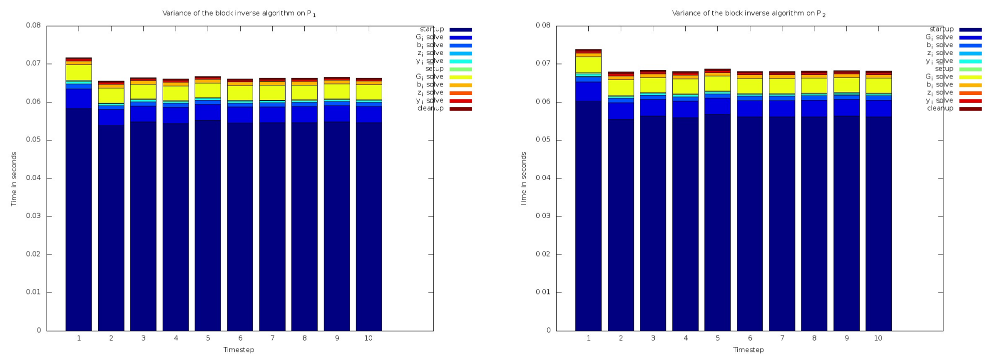


Figure 12: Computation time over 10 timesteps.

4 Conclusion

In this paper, an attempt was made at parallelizing a finite difference scheme for the KdV equation. The block inverse algorithm that was used does not appear to be a good candidate for this problem. While we have not accomplished our original goal, this study has exposed some of the difficulties with solving nonlinear PDEs in parallel that should be investigated further. We have shown, however, that the algorithm would work well with a linear PDE such as the linearlized KdV.

References

[1] Korteweg, Diederik Johannes, and Gustav De Vries. "Xli. on the change of form of long waves advancing in a rectangular canal, and on a new type of long stationary waves." *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science* 39.240 (1895): 422-443.

[2] Zabusky, Norman J., and Martin D. Kruskal. "Interaction of solitons in a collisionless plasma and the recurrence of initial states." *Phys. Rev. Lett* 15.6 (1965): 240-243.

[3] Goda, Katuhiko. "On stability of some finite difference schemes for the Korteweg-de Vries equation." *Journal of the Physical Society of Japan* 39.1 (1975): 229-236.

[4] Abe, Kanji, and Osamu Inoue. "Fourier expansion solution of the Korteweg-de Vries equation." *Journal of Computational Physics* 34.2 (1980): 202-210.

[5] Fornberg, Bengt, and G. B. Whitham. "A numerical and theoretical study of certain nonlinear wave phenomena." *Philosophical Transactions of the Royal Society of London A: Mathematical, Physical and Engineering Sciences* 289.1361 (1978): 373-404.

[6] Arnold, Douglas N., and Ragnar Winther. "A superconvergent finite element method for the Korteweg-de Vries equation." *Mathematics of Computation* 38.157 (1982): 23-36.

[7] Bona, Jerry L., Vassilios A. Dougalis, and Ohannes A. Karakashian. "Fully discrete Galerkin methods for the Korteweg-de Vries equation." *Computers & Mathematics with Applications* 12.7 (1986): 859-884.

[8] Winther, Ragnar. "A conservative finite element method for the Korteweg-de Vries equation." *Mathematics of Computation* (1980): 23-43.

[9] Qu, Fu-li, and Wen-qia Wang. "Alternating segment explicit-implicit scheme for nonlinear third-order KdV equation." *Applied Mathematics and Mechanics* 28 (2007): 973-980.

[10] Guo, Jinhua, and Thiab R. Taha. "Parallel implementation of the split-step and the pseudospectral methods for solving higher KdV equation." *Mathematics and Computers in Simulation* 62.1 (2003): 41-51.

[11] Schendel, Udo. *Introduction to numerical methods for parallel computers.* Prentice Hall PTR, 1984.

[12] Taha, Thiab R., and Mark J. Ablowitz. "Analytical and numerical aspects of certain nonelinear evolution equations III. Numerical, Korteweg-de Vries equation." *Journal of Computational Phycis* 55.2 (1984): 231-253.

[13] Blelloch, Guy E. "Prefix sums and their applications." (1990).

[14] Quintana-Ort, Gregorio, et al. "SuperMatrix for the Factorization of Band Matrices FLAME Working Note 27." (2007).

[15] Gupta, Anshul, et al. "The design, implementation, and evaluation of a symmetric banded linear solver for distributed-memory parallel computers." *ACM Transactions on Mathematical Software (TOMS)* 24.1 (1998): 74-101.

[16] Hajj, Ibrahim N., and Stig Skelboe. "A multilevel parallel solver for block tridiagonal and banded linear systems." *Parallel Computing* 15.1 (1990): 21-45.

[17] Hammack, Joseph L., and Harvey Segur. "The Korteweg-de Vries equation and water waves. Part 2. Comparison with experiments." *Journal of Fluid mechanics* 65.02 (1974): 289-314.

[18] Hammack, Joseph L., and Harvey Segur. "The Korteweg-de Vries equation and water waves. Part 3. Oscillatory waves." *Journal of Fluid Mechanics* 84.02 (1978): 337-358.

[19] Bona, J., et al. "Conservative, discontinuous Galerkin-methods for the generalized Kortewegde Vries equation." *Mathematics of Computation* 82.283 (2013): 1401-1432.

[20] Bona, J. L., et al. "Conservative, high-order numerical schemes for the generalized Korteweg-de Vries equation." *Philosophical Transactions of the Royal Society of London. Series A: Physical and Engineering Sciences* 351.1695 (1995): 107-164.

[21] Dutykh, Denys, Marx Chhay, and Francesco Fedele. "Geometric numerical schemes for the KdV equation." *Computational Mathematics and Mathematical Physics* 53.2 (2013): 221-236.

[22] Craig, Walter, and Catherine Sulem. "Numerical simulation of gravity waves." *Journal of Computational Physics* 108.1 (1993): 73-83.

[23] Kolebaje, Olusola Tosin, and Oluwole Emmanuel Oyewande. "Numerical solution of the Korteweg-de Vries equation by finite difference and adomian decomposition method." *International Journal of Basic and Applied Sciences* 1.3 (2012): 321-335.

[24] Scott, L. Ridgway, Terry Clark, and Babak Bagheri. *Scientific parallel computing.* Vol. 146. Princeton: Princeton University Press, 2005.

[25] Scott, L. Ridgway. "Tsunami Simulation." (2013).