

WEB BASED INTERFACE FOR NUMERICAL SIMULATIONS OF NONLINEAR EVOLUTION EQUATIONS

by

RYAN NAPOLEON FOSTER

(Under the Direction of Dr. Thiab R. Taha)

ABSTRACT

In Computational Science and Parallel Computing research, model equations have been developed to assist in solving problems in science and engineering. Such equations have aided researchers in developing methods used in the study of weather prediction, optical fiber communication systems, water waves, etc... Often, it is the desire of many researchers to further develop numerical methods and make these model equations, their numerical simulations and plots accessible to users through the Internet. In this thesis, I present a web based graphical user numerical simulation interface for nonlinear evolution equations such as the nonlinear Schrödinger (NLS), coupled NLS (CNLS), and the complex modified Korteweg-de Vries (CMKdV) equations. In addition, the numerical implementation of a proposed numerical scheme for the modified CNLS equation is presented. Sequential and parallel algorithms for such equations were implemented on sequential and multiprocessor machines.

Index Words: Numerical Simulation, Nonlinear Schrödinger (NLS) equation, Coupled Nonlinear Schrödinger (CNLS) equation, Complex Modified Korteweg-de Vries (CMKdV) equation, Web based graphical user interface

WEB BASED INTERFACE FOR NUMERICAL SIMULATIONS OF NONLINEAR
EVOLUTION EQUATIONS

by

RYAN NAPOLEON FOSTER

B.S., Georgia Southern University, 2004

A Thesis Submitted to the Graduate Faculty of the University of Georgia in Partial
Fulfillment of the Requirements for the Degree

MASTER OF SCIENCE

ATHENS, GEORGIA

2007

© 2007

Ryan Napoleon Foster

All Rights Reserved

WEB BASED INTERFACE FOR NUMERICAL SIMULATIONS OF NONLINEAR
EVOLUTION EQUATIONS

By

RYAN NAPOLEON FOSTER

Major Professor: Dr. Thiab R. Taha

Committee: Dr. Leon Deligiannidis
Dr. Daniel Everett

Electronic Version Approved:

Maureen Grasso
Dean of the Graduate School
The University of Georgia
May 2007

ACKNOWLEDGEMENTS

I would like to thank everyone who has provided encouragement, support, and assistance in the enhancement of my knowledge, experience, and personal growth in computer technology. I specifically would like to express my appreciation and thanks towards my major advisor, Dr. Thiab R. Taha, for his advisement throughout my graduate career. It has been a great honor to work under his direction. I would also like to thank the other members of my advisory committee, Dr. Daniel M. Everett and Dr. Leon Deligiannidis, for their insight, influence, and motivation. Without the coupling of all these resources, the success and completion of this thesis would not be possible.

TABLE OF CONTENTS

	Page
ACKNOWLEDGEMENTS	iv
CHAPTER	
1 INTRODUCTION	1
2 NUMERICAL SIMULATION OF THE NONLINEAR SCHRÖDINGER (NLS) EQUATION	3
Introduction	3
Methods for Solving the NLS Equation	3
3 NUMERICAL SIMULATION OF THE COUPLED NONLINEAR SCHRÖDINGER (CNLS) EQUATION	21
Introduction	21
Methods for Solving the CNLS Equation	21
4 NUMERICAL SIMULATION OF THE COMPLEX MODIFIED KORTEWEG-DE VRIES (CMKDV) EQUATION	32
Introduction	32

Methods for Solving the CMKdV Equation	33
5 PARALLEL (FFTW-MPI) IMPLEMENTATION OF THE CMKDV EQUATION ...	41
Introduction	41
Parallel Methods for Solving the CMKdV Equation	41
6 WEB BASED INTERFACE FOR NUMERICAL SIMULATION	51
Introduction	51
Equation Server	51
7 VISUALIZATION AND PROFILING TOOLS	60
Introduction	60
Jumpshot-4	60
8 CONCLUSIONS	67
BIBLIOGRAPHY	68
APPENDICES	70
A EQUATION SERVER README.....	70

CHAPTER 1

INTRODUCTION

In Computational Science and Parallel Computing research, equations have been developed to assist in solving problems in science and engineering. Such equations have aided researchers in developing methods used in weather prediction, optical fiber communication systems, water wave and plasma physics modeling. In this thesis, I discuss the various methods for numerical simulation of nonlinear evolution equations. These equations include: the Nonlinear Schrödinger (NLS) equation, the Dispersed Nonlinear Schrödinger (Dispersed NLS) equation, the Coupled Nonlinear Schrödinger (CNLS) equation, the Modified Coupled Nonlinear Schrödinger (Modified CNLS) Equation, and the Complex Modified Korteweg-de Vries (CMKdV) equation.

The nonlinear Schrödinger (NLS) and the coupled nonlinear Schrödinger (CNLS) equations are of tremendous interest in both theory and applications [6]. Various regimes of pulse propagation in optical fibers are modeled by some form of the NLS type equation. The CNLS equation is the governing equation for the propagation of two orthogonally polarized pulses in monomode birefringent fibers [6]. The complex modified Korteweg-de Vries (CMKdV) equation has been used to model the traveling-wave solutions as well as the double homoclinic orbits.

In parallel computation, developers seek ways to improve their algorithms based on speedup and efficiency. Speedup is determined by the execution time of the fastest sequential

algorithm divided by the execution time on a p processor system. Efficiency is determined by speedup divided by the number of processors. Therefore, the main goal is to split the computation load evenly between the available processors to achieve a better speedup and higher efficiency. The parallel algorithms discussed in this thesis have been simulated and results were obtained with this goal in mind.

With the advancement and success of the Internet, researchers wish to make their numerical methods for solving a wide range of mathematical problems available online. In this thesis I will concentrate on the nonlinear evolution equations. The output results of such equations contain a very large amount of data, vary based on user input, and are often analyzed and interpreted in the form of a graph or plot [1]. In this thesis, I will introduce Equation Server, a web based graphical user interface for providing such equations, their results, and plots available to users on the web.

Visualization and profiling tools are very important in the analysis and interpretation of numerical methods and simulations. I present Jumpshot-4, a profiling tool used in the analysis of the processor communication within the parallel algorithm implementations presented in this thesis.

CHAPTER 2

NUMERICAL SIMULATION OF THE NONLINEAR SCHRÖDINGER (NLS) EQUATION

2.1 Introduction

In this chapter, I present various numerical methods for solving the nonlinear Schrödinger (NLS) type equation that are developed by Xu and Taha [6]. A wide class of physical phenomena (e. g. modulation of deep water waves, propagation of pulses in optical fibers, and self-trapping of a light beam in a color-dispersive system) is modeled by the NLS type equation [6]. In their work, the NLS equation is described as:

$$iu_t - u_{xx} + q|u|^2 u = 0, \quad (1)$$

where u is a complex-valued function and q is a real number.

2.2 Methods for Solving the NLS Equation:

The split-step Fourier (SSF) method proposed by R. H. Hardin and F. D. Tappert is one of the most popular numerical methods for solving the NLS equation [6]. Various versions of the split-step method have been developed to solve the NLS equation. G. M. Muslu and H. A. Erbay also introduced the three different split-step schemes (first-order, second-order, and fourth order approximations) for the numerical simulation of the complex modified Korteweg-de Vries (CMKdV) equation

$$w_t + w_{xxx} + \alpha |w|^2 w_x = 0, \quad (2)$$

where w is a complex-valued function of the spatial coordinates x and time t , and α is a real parameter. The numerical simulation of the CMKdV equation will be covered later in Chapter 4.

Consider a general evolution equation of the form

$$\begin{aligned} u_t &= (L + N)u, \\ u(x, 0) &= u_0(x), \end{aligned} \quad (3)$$

where L and N are linear and nonlinear operators, respectively. In general, the operators L and N do not commute with each other. For example, the NLS equation

$$u_t = -iu_{xx} + iq|u|^2 u,$$

with q a real number, can be rewritten as

$$u_t = Lu + Nu,$$

where

$$Lu = -iu_{xx}, \quad Nu = iq|u|^2 u.$$

The solution of equation (3) may be advanced from one time-level to the next by the following formula

$$u(x, t + \Delta t) \doteq \exp[\Delta t(L + N)]u(x, t), \quad (4)$$

where Δt denotes the time step. It has been shown that this scheme is first order accurate.

The time-splitting procedure now consists of replacing the right-hand side of (4) by an appropriate combination of products of the exponential operators $\exp(\Delta tL)$ and $\exp(\Delta tN)$ [6]. An answer can be found by considering the Baker-Campbell-Hausdorff (BCH) formula for two operators A and B given by

$$\exp(\lambda A)\exp(\lambda B) = \exp\left(\sum_{n=1}^{\infty} \lambda^n Z_n\right), \quad (5)$$

where

$$Z_1 = A + B,$$

and the remaining operators Z_n are commutators of A and B, commutators of commutators of A and B, etc. The expression for Z_n is actually rather complicated, e.g.

$$Z_2 = \frac{1}{2}[A, B],$$

where $[A, B] = AB - BA$ is the commutator of A and B, and

$$Z_3 = \frac{1}{12}([A, [A, B]] + [[A, B], B]),$$

From this result, one can get the first-order approximation of the exponential operator in (4) as follows

$$A_1(\Delta t) = \exp(\Delta t L) \exp(\Delta t N). \quad (6)$$

This expression is exact whenever L and N commute.

It is convenient to view the scheme (6) as first solving the nonlinear equation

$$u_t = Nu,$$

then advancing the solution by solving the linear equation

$$u_t = Lu,$$

employing the solution of the former as the initial condition of the latter. That is, the advancement in time is carried out in two steps, the so called split-step method.

The second-order approximation of the exponential operator in (4) is given by

$$A_2(\Delta t) = \exp\left(\frac{1}{2}\Delta t N\right) \exp(\Delta t L) \exp\left(\frac{1}{2}\Delta t N\right). \quad (7)$$

It is symmetric in the sense that $A_2(\Delta t)A_2(-\Delta t) = 1$.

The fourth-order approximation of the exponential operator in (4) which preserves the symmetry can also be constructed, e.g.

$$A_4(\Delta t) = A_2(w\Delta t)A_2[(1-2w)\Delta t]A_2(w\Delta t), \quad (8)$$

where

$$\omega = \frac{2 + \sqrt[3]{2} + \frac{1}{\sqrt[3]{2}}}{3}. \quad (9)$$

Note that the operators L and N in (6)–(8) may be interchanged without affecting the order of the method.

The implementation of the above methods will be carried out by using the Fourier transform.

The Fourier transform

The Fourier transform is used to decompose a signal into its constituent frequencies. It is a powerful tool in linear system analysis.

The discrete Fourier transform

If $\{f_j\}$ is a sequence of length N, obtained by taking samples of a continuous function f at equal intervals, then its discrete Fourier transform (DFT) is the sequence $\{F_k\}$ given by

$$F_k = \frac{1}{\sqrt{N}} \sum_{j=0}^{N-1} f_j \omega_N^{-jk}, 0 \leq k < N, \quad (15)$$

where $\omega_N = e^{i\frac{2\pi}{N}}$ is a primitive N-th root of unity.

The inverse DFT flips the sign of the exponent of ω_N , and it is defined as

$$f_j = \frac{1}{\sqrt{N}} \sum_{k=0}^{N-1} F_k \omega_N^{jk}, 0 \leq j < N, \quad (16)$$

It is the “inverse” of the forward DFT, in the sense that computing the inverse transform after the forward transform of a given sequence yields the original sequence.

After the required values of the complex exponential have been stored in a table, the number of arithmetic (multiplication or addition) operations required to implement DFT as in (15) is about $2N^2$, and hence it is of order N^2 . So is the inverse DFT.

The Fast Fourier Transform

As mentioned above, the DFT requires $O(N^2)$ operations to compute and makes the computation potentially burdensome. Fortunately, there exists an algorithm called fast Fourier transform (FFT) that reduces the required number of arithmetic operations to $O(N \log_2(N))$. This requires that N can be factored into a product of small integers. The most common case is $N = 2^q$ for an integer q .

Suppose N can be factored as $N = p_1 p_2$, and then the indices j and k in (15) can be represented as

$$j = j_1 p_2 + j_0; j_1 = 0, \dots, p_1 - 1, j_0 = 0, \dots, p_2 - 1,$$

and

$$k = k_1 p_2 + k_0; k_1 = 0, \dots, p_2 - 1, k_0 = 0, \dots, p_1 - 1.$$

Substitute into the expression (15), we obtain

$$\begin{aligned} F_k &= \frac{1}{\sqrt{N}} \sum_{j_0=0}^{p_2-1} \sum_{j_1=0}^{p_1-1} f_{j_1 p_2 + j_0} \omega_N^{-(j_1 p_2 + j_0)k} \\ &= \frac{1}{\sqrt{N}} \sum_{j_0=0}^{p_2-1} \left(\sum_{j_1=0}^{p_1-1} f_{j_1 p_2 + j_0} \omega_N^{-(j_1 k p_2)} \right) \omega_N^{-j_0 k}. \end{aligned}$$

Note that we have used the fact that $\omega_N^{-j_1 k p_2} = \omega_N^{-j_1 k_0 p_2}$, since $\omega_N^N = 1$. It follows that

$$F_k = \frac{1}{\sqrt{N}} \sum_{j_0=0}^{p_2-1} \tilde{F}_{j_0, k_0} \omega_N^{-j_0 k}, \quad (17)$$

where

$$\tilde{F}_{j_0, k_0} = \sum_{j_1=0}^{p_1-1} f_{j_1 p_2 + j_0} \omega_N^{-j_1 k_0 p_2} \quad (18)$$

Observe that the number of arithmetic operations has indeed been reduced by this procedure.

Each of the N elements in (18), \tilde{F}_{j_0, k_0} , requires $2p_1$ arithmetic operations, for a total of $2Np_1$ operations. Each F_k in (17) requires additional $2p_2$ operations. Thus the number of arithmetic operations to obtain all the F_k is $N(p_1 + p_2)$.

If p_1 and p_2 are factorable then the procedure can be repeated. In fact, if

$$N = p_1 p_2 p_3 \dots p_m,$$

then the entire process applied recursively in this manner requires

$$2N(p_1 + p_2 + \dots + p_m)$$

operations. For $p_1 = p_2 = \dots = p_m = p$,

$$2pN \log_p N$$

operations are needed. In particular, for $p = 2$, which is the most common case, a total of

$4N \log_2 N$ arithmetic operations are required to compute the DFT.

The Fastest Fourier Transform in the West

The Fast Fourier Transform in the West (FFTW) was developed by Matteo Frigo and Steven G. Johnson at MIT. FFTW is a C subroutine library for computing the discrete Fourier transform (DFT) in one or more dimensions, of arbitrary input size, and of both real and complex data (as well as of even/odd data, i.e. the discrete cosine/sine transforms or DCT/DST). FFTW features also include strided transforms and parallel transforms. The FFTW library is portable to any platform with a C compiler and contains documentation in HTML and other formats. It also contains both C and Fortran interfaces and its own data types and routines for the successful implementation of FFTW programs. Currently, the FFTW package and documentation are freely available on the Internet (www.fftw.org/download.html).

According to FFTW developers, FFTW's benchmark performance has shown to be typically superior to that of other publicly available FFT software, and is even competitive with vendor-tuned codes, however, FFTW's performance is portable; the same program will perform well on most architectures without modification [4].

FFTW proves to be a valuable enhancement over the basic Fast Fourier Transform (FFT) library. To accomplish this, FFTW automatically adapts the DFT algorithm to details of the underlying hardware (cache size, memory size, registers, etc.). The inner loop of FFTW is generated automatically by a special-purpose compiler. The FFTW begins by generating codelets. A codelet is a fragment of C code that computes a Fourier transform of a fixed small size (e.g. 16 or 19). A composition of codelets is called a plan which depends on the size of the input and the underlying hardware. At runtime, the FFTW's planner finds the optimal decomposition for transforms of a specified size on your machine and produces a plan that

contains this information. The resulting plan can be reused as many times as needed. Many transforms of the same size are computed in typical high performance applications. This makes the FFTW's relatively expensive initialization acceptable [6].

FFTW also includes a shared-memory implementation on top of POSIX threads, and a distributed-memory implementation based on MPI (Message Passing Interface). The FFTW's MPI routines are significantly different from the ordinary FFTW because the transformed data are distributed over multiple processes, so that each processes gets only a portion of the transform data [4]. The distributed-memory implementation based on MPI (FFTW-MPI) was incorporated into the parallel algorithms used for solving the various model equations presented throughout this thesis.

Message Passing Interface

Message Passing Interface (MPI) is a method by which data from one processor memory is copied to the memory of another processor. MPI developers have provided a standard library of functions, data types, and routines for writing message-passing programs, and it was proposed as a standard by a broadly based committee of vendors, implementers, and users. MPI offers portability, standardization, high performance, rich functionality, and many high quality implementations. It was designed for high performance on both massively parallel machines and on workstation clusters and is a paradigm used widely on certain classes of parallel machines, especially those with distributed memory. Processes running on such machines communicate through these messages.

Each MPI implementation must set up the MPI environment by incorporating MPI environment routines in the implementation. These environment routines include `MPI_Init`, `MPI_Comm_size`, and `MPI_Comm_rank`, and `MPI_Finalize`. Each MPI routine consists of a communicator argument, `MPI_COMM_WORLD`. A communicator consists of a group of processes participating in the parallel job. `MPI_Init` initializes the MPI execution environment and must be called only once. `MPI_Comm_size` determines the number of processors associated with a communicator. `MPI_Comm_rank` uniquely identifies the rank of the calling process within the communicator. Typically, the root processor is given the rank of 0. `MPI_Finalize` terminates the MPE execution environment.

Due to the sending and receiving of data between the processes, MPI has developed its own data types that are consistent with primitive data types used in programming languages such as C. MPI data types include `MPI_CHAR`, `MPI_SHORT`, `MPI_INT`, `MPI_LONG`, `MPI_FLOAT`, `MPI_DOUBLE`, and `MPI_LONG_DOUBLE`. It is also possible to pass arrays between processes using the provided MPI routines.

To communicate between processes, MPI provides a set of communicator routines. These communicator routines include `MPI_SEND`, `MPI_RECV`, `MPI_ISEND`, `MPI_Irecv`, `MPI_BCAST`, `MPI_Gather`, `MPI_Scatter`, and `MPI_Reduce`. MPI also provides reduction routines such as `SUM`, `MAX`, `MIN`, etc...

`MPI_SEND` and `MPI_ISEND` are blocking and non-blocking routines that send data to other processes. `MPI_RECV` and `MPI_Irecv` are blocking and non-blocking routines that receive data from other processes. `MPI_BCAST` broadcasts values from the root process to the other processes of the group. `MPI_Gather` gathers together values from processes in a group. Each process sends the contents of its send buffer to the root process, which then receives those

messages and stores them in its receive buffer according to the rank order of the sender.

MPI_Scatter sends data from one task to all other tasks in a group. It performs the reverse operation of the MPI_Gather routine. Each process receives a segment from the root and places it in its receive buffer. MPI_Reduce combines data from all processes in the communicator and returns it to one process.

MPI also includes routines that provide the capability of creating and managing subgroups, which are a subset of the MPI_COMM_WORLD communicator handle. Such routines include MPI_Comm_split, MPI_Comm_create, MPI_Group_incl, MPI_Group_excl, MPI_Group_range_incl, MPI_Group_range_excl, MPI_Group_union, and MPI_Group_intersection.

MPI_Group_incl and MPI_Group_excl include and exclude specific members of the group. MPI_Group_range_incl and MPI_Group_range_excl include and exclude a range of members of the group. MPI_Group_union and MPI_Group_intersection creates a new group from two existing groups.

As mentioned above, the MPI package provides a wide variety of subroutines to assist in the successful implementation of various parallel algorithms. The MPI subroutines successfully facilitated communication between processes for the parallel algorithms covered in this thesis. In chapter 7, I will present, Jumpshot-4, a profiling tool useful in the analysis and interpretation of the communication occurring between parallel algorithms.

Numerical Implementation of the NLS equation

The following example is based on a presentation given by Xu and Taha on nonlinear evolutionary equations [6].

Consider the following NLS equation

$$iu_t = u_{xx} + 2|u|^2 u, \quad (19)$$

where u is a complex-valued function. The exact one-soliton solution of (19) on the infinite interval is

$$u(x, t) = 2\eta \exp\{-i[2\xi x - 4(\xi^2 - \eta^2)t + \phi_0 + \frac{\pi}{2}]\} \operatorname{sech}(2\eta x - 8\xi\eta t - x_0), \quad (20)$$

where x_0, η, ξ, ϕ_0 are constants.

They studied the NLS equation (19) with the initial condition given by

$$u(x, 0) = 2\eta \exp\{-i[2x + \frac{\pi}{2}]\} \operatorname{sech}(2\pi x), \quad (21)$$

where $\eta = \text{constant}$. It was assumed that $u(x, t)$ satisfies a periodic boundary condition with period $[-p, p]$, p is half the length of the interval.

If the spatial period is normalized to $[0, 2\pi]$, then equation (19) becomes

$$iu_t = \frac{\pi^2}{p^2} u_{xx} + 2|u|^2 u, \quad (22)$$

and $X = \pi(x + p)/p$. They divide the interval $[0, 2\pi]$ into N equal subintervals with grid spacing $\Delta X = 2\pi/N$, and denote $X_j = j\Delta X, j = 0, 1, \dots, N$ as the spatial grid points.

The solution of NLS may be advanced from time t to the next time-level $t + \Delta t$ by the following two steps.

(1) Advance the solution using only the nonlinear part:

$$iu_t = 2|u|^2 u, \quad (23)$$

through

$$\tilde{u}(X_j, t + \Delta t) = \exp\{-2i|u(X_j, t)|^2 \Delta t\} u(X_j, t). \quad (24)$$

(2) Advance the solution according to the linear part:

$$iu_t = \frac{\pi^2}{p^2} u_{xx}, \quad (25)$$

by means of computing

$$\hat{u}(X_k, t + \Delta t) = F(\tilde{u}(X_j, t + \Delta t))_k, \quad (26)$$

followed by

$$\tilde{u}(X_k, t + \Delta t) = \exp\{ik^2 \Delta t \frac{\pi^2}{p^2}\} \hat{u}(X_k, t + \Delta t), \quad (27)$$

and

$$u(X_j, t + \Delta t) = F^{-1}(\tilde{u}(X_k, t + \Delta t))_j, \quad (28)$$

where Δt denotes the time step, and F and F^{-1} are the discrete Fourier transform and its inverse respectively. This is the split-step Fourier method corresponding to the first-order splitting approximation (6).

Similarly, the advancement in time from t to $t + \Delta t$ by the split-step Fourier method using the second-order splitting approximation (7) can be carried out by the following three steps:

(1') Advance the solution using the nonlinear part (23) through the following scheme

$$\tilde{u}(X_j, t + \frac{1}{2} \Delta t) = \exp\{-2i|u(X_j, t)|^2 \frac{1}{2} \Delta t\} u(X_j, t).$$

(2') Advance the solution according to the linear part

(25) by means of the discrete Fourier transforms

$$\tilde{u}(X_k, t + \frac{1}{2}\Delta t) = F^{-1}(\exp\{ik^2\Delta t \frac{\pi^2}{p^2}\}F(\hat{u}(X_j, t + \frac{1}{2}\Delta t))).$$

(3') Advance the solution using the nonlinear part (23) through the following scheme

$$u(X_j, t + \Delta t) = \exp\{-2i\left|u(X_j, t + \frac{1}{2}\Delta t)\right|^2 \frac{1}{2}\Delta t\}\tilde{u}(X_j, t + \frac{1}{2}\Delta t).$$

The split-step method based on the fourth-order splitting approximation scheme (8) is described as follows. First, they advanced in time from t to $t+\omega\Delta t$ by the second-order split-step Fourier method described above with

$$\omega = \frac{2 + \sqrt[3]{2} + \frac{1}{\sqrt[3]{2}}}{3}.$$

Then they advance in time from $t+\omega\Delta t$ to $t+(1-\omega)\Delta t$ by the second-order split-step Fourier method. Finally, they advance in time from $t+(1-\omega)\Delta t$ to $t+\Delta t$ by the second-order split-step Fourier method, and obtain approximations to $u(x, t+\Delta t)$.

Numerical experiments

In the numerical experiments, Xu and Taha calculated the L_∞ norm, L_2 norm at the terminating time $T = 1$. They also calculated the relative errors, i_1, i_2 , of the following two conserved quantities

$$I_1 = \int_{-\infty}^{+\infty} |u|^2 dx, \quad (29)$$

and

$$I_2 = \int_{-\infty}^{+\infty} (|u|^4 - |\frac{\partial u}{\partial x}|^2) dx, \quad (30)$$

respectively. The two conserved quantities are calculated by means of the Simpson's rule, and the derivatives in (30) are calculated using Fourier method.

They showed that the first order split-step Fourier method converges linearly in time. The convergence rates in time for the second-order and fourth-order split-step Fourier method are second-order and fourth-order, respectively. Moreover, they showed that the computational cost of the second-order scheme is 1.2 times of the first-order scheme, whereas the computational cost of the fourth-order scheme is about 3 times of the second-order scheme. They also showed that all of the three split-step Fourier methods converge exponentially in space.

Parallel Implementation

For first-order split-step Fourier method, each of the four computational steps arise in (24) and (26)–(28) are parallelized.

Let A , of size N , be the approximate solution to u at time t . Suppose there are p processors in a distributed memory parallel computer. Parallelizing (24) and (27) are straightforward. The array A is distributed among P processors. Processor n , $0 \leq n \leq P - 1$, contains array elements $A[nN/P]$ to $A[(n+1)N/P-1]$. Each of the P processor works on its own subarrays independently without communicating with others. The FFTW's MPI routines are employed to implement parallel discrete Fourier transforms to parallelize the computations in stages (26) and (28).

The parallel algorithms for the second-order and fourth-order split-step Fourier methods can be developed in a straightforward manner.

Parallel algorithms of the split-step Fourier methods are implemented on the IBM Sp2 machine and taha 4 processor machine. The results from Xu and Taha are shown on Tables (2.2.1 - 2.2.3). The speedup S_p is defined by

$$S_p = \frac{\text{Time spent to run the MPI code on one processor}}{\text{Time spent to run the MPI code on } p \text{ processors}}$$

Table 2.2.1 CPU time and speedup for the parallel first-order scheme using FFTW

p	N = 4096 NS = 2000	Speedup	N = 16384 NS = 500	Speedup	N = 65536 NS = 125	Speedup
1	11.4	1.0	11.4	1.0	12.9	1.0
2	9.7	1.2	9.5	1.2	9.5	1.4
4	7.6	1.5	5.6	2.1	6.2	2.1

Table 2.2.2 CPU time and speedup for the parallel second-order scheme using FFTW

p	N = 4096 NS = 2000	Speedup	N = 16384 NS = 500	Speedup	N = 65536 NS = 125	Speedup
1	12.3	1.0	12.5	1.0	13.8	1.0
2	10.2	1.2	9.6	1.3	10.2	1.4
4	7.3	1.7	6.1	2.1	6.4	2.2

Table 2.2.3 CPU time and speedup for the parallel fourth-order scheme using FFTW

p	N = 4096 NS = 2000	Speedup	N = 16384 NS = 500	Speedup	N = 65536 NS = 125	Speedup
1	45.5	1.0	45.0	1.0	48.1	1.0
2	34.1	1.3	34.0	1.3	35.3	1.4
4	24.5	1.9	20.2	2.2	21.8	2.2

From the results, it is clear that the speedup increases as the problem size N becomes larger for a fixed number of processors p . For small problem sizes the computation/communication ratio is small, thus speedup is small. For fixed p , we can also see that the fourth-order scheme has a better speedup than the second-order scheme, whereas the second-order scheme has a slightly better speedup than the first-order scheme. This is due to the fact that the fourth-order scheme is more computational intensive than the second-order scheme, whereas the second-order scheme is more computational intensive than the first-order scheme. For large N , the speedups achieved on the multiprocessor computer running the parallel codes are considerable.

Perturbed nonlinear Schrödinger equation

In this section, they examine the perturbed NLS equation of the form:

$$iw_t + \frac{1}{2} D(t) \frac{\partial^2 w}{\partial x^2} + g(t) |w|^2 w = 0, \quad (33)$$

where $D(t)$ represents dispersion, which is given by the following periodic function

$$D(t) = \begin{cases} D_1, & 0 \leq t < \theta t_m, \\ D_2, & \theta t_m \leq t < t_m, \end{cases} \quad (0 \leq \theta < 1)$$

and $g(t)$ relates to effective nonlinearity, which is given by the periodic function

$$g(t) = \left(\frac{2\Gamma}{1 - e^{-2\Gamma t_a}} \right) e^{-2\Gamma t}, \text{ for } 0 \leq t \leq t_a$$

In their numerical experiments, they have chosen $D_1 = 1$, $D_2 = -1$, $\theta = 0.8$, the map period $t_m = 0.1$, the damping coefficient $\Gamma = 4$, and the amplifier spacing $t_a = 0.1$.

They studied equation (33) with periodic boundary condition of period $[-20, 20]$, and initial condition of the form

$$w(t, x) = A \sec h[A(x - \Omega t - x_0)] \exp\{i[\Omega^2 - A^2]t + \varphi\}, \quad (34)$$

with $t = 0$, amplitude $A = 1$, velocity $\Omega = 2$, initial position $x_0 = 0$, and phase $\varphi = 0$.

Modified nonlinear Schrödinger equation

The pulse propagation in a dispersion exponentially decreasing fiber can be described by the modified NLS equation

$$iUt - \frac{1}{2}\beta_2(t)\frac{\partial^2 U}{\partial x^2} - i\frac{1}{6}\beta_3\frac{\partial^3 U}{\partial x^3} + \gamma|U|^2 U = -i\frac{1}{2}\alpha U, \quad (35)$$

where U is the normalized field envelope; $\beta_2(t)$ is the second-order dispersion; β_3 is the third-order dispersion; $\gamma = n_2\omega/cA_{eff}$, where n_2 is the Kerr coefficient, ω is the carrier frequency, c is the velocity of the light in vacuum, and A_{eff} is the effective fiber cross section; α is the fiber loss.

In their numerical experiments, they let $\beta_2(t)$ be a periodic function

$$\beta_2(t) = e^{-\alpha t} \beta_2(0), \text{ for } 0 \leq t < 0.01,$$

where $\beta_2(0) = -0.5$. Other parameters are taken to be $\beta_3 = 0.14$, $\gamma = 3.2/1.55$, and $\alpha = 0.2$.

They study equation (35) with periodic boundary condition of period $[-10, 10]$, and the initial condition

$$U(x,0) = \text{sech}(x)e^{ix}. \tag{36}$$

CHAPTER 3

NUMERICAL SIMULATION OF THE COUPLED NONLINEAR SCHRÖDINGER (CNLS) EQUATION

3.1. Introduction

In this chapter, we discuss the numerical simulations for the coupled nonlinear Schrödinger (CNLS) equation introduced by Xu, Ismail and Taha [8]. This equation was implemented using two methods: 1) Split-step Fourier Method and First Order (BCH) Approximation to the Exponential Operator and 2) Crank Nicolson Method. These methods were studied and simulated. In the next sections, we present these two methods.

3.2 Methods for solving the CNLS Equation

The CNLS equation is of tremendous interest in both theory and applications. The governing equation for the propagation of two orthogonally polarized pulses in monomode birefringent fibers is given by a CNLS equation.

Consider a CNLS equation of the form:

$$\begin{aligned} i\left(\frac{\partial \psi_1}{\partial t} + \delta \frac{\partial \psi_1}{\partial x}\right) + \frac{1}{2} \frac{\partial^2 \psi_1}{\partial x^2} + (|\psi_1|^2 + \mu |\psi_2|^2) \psi_1 &= 0, \\ i\left(\frac{\partial \psi_2}{\partial t} + \delta \frac{\partial \psi_2}{\partial x}\right) + \frac{1}{2} \frac{\partial^2 \psi_2}{\partial x^2} + (\mu |\psi_1|^2 + |\psi_2|^2) \psi_2 &= 0, \end{aligned} \tag{37}$$

where ψ_1 and ψ_2 are the two polarized waves, μ is a real parameter, $i = \sqrt{-1}$, and δ is the normalized strength of the linear birefringence.

In general, the CNLS equation with arbitrary coefficients is not integrable. For $\mu = 1$, equation (37) reduces to the Manakov equation which is integrable.

The explicit form of soliton solution of the equation (37) is given by

$$\begin{aligned}\psi_1 &= \sqrt{\frac{2\alpha}{1+\mu}} \operatorname{sech}[\sqrt{2\alpha}(x-vt)] \exp\{i(v-\delta)x - i[\frac{1}{2}(v^2 - \delta^2) - \alpha]t\}, \\ \psi_2 &= \pm \sqrt{\frac{2\alpha}{1+\mu}} \operatorname{sech}[\sqrt{2\alpha}(x-vt)] \exp\{i(v-\delta)x - i[\frac{1}{2}(v^2 - \delta^2) - \alpha]t\},\end{aligned}\quad (38)$$

The CNLS equation has the following two conserved quantities

$$E_1 = \int_{-\infty}^{+\infty} |\psi_1|^2 dx, \quad (39)$$

and

$$E_2 = \int_{-\infty}^{+\infty} |\psi_2|^2 dx, \quad (40)$$

that remain constant in time. Note that they represent the energy of the system. From the exact solution (38), it is easy to show that

$$E_1 = E_2 = \frac{2}{1+\mu} \sqrt{\alpha}. \quad (41)$$

Recently, Xu and Taha solved the above equation by using the split-step Fourier method as follows:

Although the CNLS equation (37) is defined over the real line for the numerical experiments considered, they assume that the solution of equation (37) is negligibly small outside the interval $[x_l, x_r]$. The boundaries are far apart enough so that they do not affect the propagation of solitary waves.

In the following, they studied the coupled nonlinear Schrödinger equation

$$\begin{aligned} i\left(\frac{\partial \psi_1}{\partial t} + \delta \frac{\partial \psi_1}{\partial x}\right) + \frac{1}{2} \frac{\partial^2 \psi_1}{\partial x^2} + (|\psi_1|^2 + \mu |\psi_2|^2) \psi_1 &= 0, \\ i\left(\frac{\partial \psi_2}{\partial t} + \delta \frac{\partial \psi_2}{\partial x}\right) + \frac{1}{2} \frac{\partial^2 \psi_2}{\partial x^2} + (\mu |\psi_1|^2 + |\psi_2|^2) \psi_2 &= 0, \end{aligned} \quad (42)$$

and assumed that ψ_1 and ψ_2 satisfy the initial conditions

$$\psi(x,0) = g_1(x), \psi_2(x,0) = g_2(x), x \in [x_l, x_r], \quad (43)$$

and periodic boundary conditions

$$\begin{aligned} \psi_1(x_l, t) &= \psi_1(x_r, t), t \in [0, T], \\ \psi_2(x_l, t) &= \psi_2(x_r, t), t \in [0, T]. \end{aligned} \quad (44)$$

The space discretization is accomplished by a Fourier method. For convenience, the finite interval $[x_l, x_r]$ is normalized to $[0, 2\pi]$ by the linear transform $X = (x - x_l)\pi / P$, where P is the half length of the interval, i.e. $P = (x_r - x_l) / 2$. Equations (42)–(44) may be rewritten as

$$\begin{aligned} i\left(\frac{\partial \psi_1}{\partial t} + \frac{\delta \pi}{P} \frac{\partial \psi_1}{\partial X}\right) + \frac{1}{2} \frac{\pi^2}{P^2} \frac{\partial^2 \psi_1}{\partial X^2} + (|\psi_1|^2 + \mu |\psi_2|^2) \psi_1 &= 0, \\ i\left(\frac{\partial \psi_2}{\partial t} - \frac{\delta \pi}{P} \frac{\partial \psi_2}{\partial X}\right) + \frac{1}{2} \frac{\pi^2}{P^2} \frac{\partial^2 \psi_2}{\partial X^2} + (\mu |\psi_1|^2 + |\psi_2|^2) \psi_2 &= 0, \end{aligned} \quad (45)$$

with initial conditions

$$\psi_1(X, 0) = \tilde{g}_1(x), \psi_2(X, 0) = \tilde{g}_2(x), x \in [0, 2\pi], \quad (46)$$

and periodic boundary conditions

$$\begin{aligned} \psi_1(0, t) &= \psi_1(2\pi, t), t \in [0, T], \\ \psi_2(0, t) &= \psi_2(2\pi, t), t \in [0, T]. \end{aligned} \quad (47)$$

The interval $[0, 2\pi]$ is divided into N equal subintervals with grid spacing $\Delta X = 2\pi/N$.

The spatial grid points are denoted by $X_j = j\Delta X$, $j = 0, 1, \dots, N$. Let $\psi_1^j(t)$ and $\psi_2^j(t)$ be the

numerical approximation to $\psi_1(X_j, t)$ and $\psi_2(X_j, t)$ at time t , respectively. The discrete Fourier transform for the sequences $\{\Psi_m^j\}$ is defined as

$$\hat{\Psi}_m^k = F(\Psi_m)_k = \frac{1}{\sqrt{N}} \sum_{j=0}^{N-1} \Psi_m^j e^{-ikX_j}, -\frac{N}{2} \leq k \leq \frac{N}{2}-1, m=1,2, \quad (48)$$

The inverse discrete Fourier transform is given by

$$\hat{\Psi}_m^j = F^{-1}(\hat{\Psi}_m)_j = \frac{1}{\sqrt{N}} \sum_{k=-\frac{N}{2}}^{\frac{N}{2}-1} \hat{\Psi}_m^k e^{ijX_k}, 0 \leq j \leq N-1, m=1,2. \quad (49)$$

These transforms can be implemented very efficiently by a fast Fourier transform algorithm, e.g. the Fastest Fourier Transform in the West (FFTW).

They used the split-step Fourier method for the coupled nonlinear Schrödinger equation (45). The basic idea is to split the exponential operator $\exp[\Delta t(L+N)]$ using the Baker-Campbell-Hausdorff formula as discussed in Chapter 1. For instance, the first-order version of the split-step method (6) is carried out as the following two steps for the advancement in time from t to $t+\Delta t$.

(1) Advance the solution using only the nonlinear part:

$$\begin{aligned} i \frac{\partial \psi_1}{\partial t} + (|\psi_1|^2 + \mu |\psi_2|^2) \psi_1 &= 0, \\ i \frac{\partial \psi_2}{\partial t} + (\mu |\psi_1|^2 + |\psi_2|^2) \psi_2 &= 0, \end{aligned} \quad (50)$$

through the following scheme

$$\begin{aligned} \tilde{\psi}_1(X_j, t + \Delta t) &= \exp\{i(|\psi_1(X_j, t)|^2 + \mu |\psi_2(X_j, t)|^2) \Delta t\} \psi_1(X_j, t), \\ \tilde{\psi}_2(X_j, t + \Delta t) &= \exp\{i(\mu |\psi_1(X_j, t)|^2 + |\psi_2(X_j, t)|^2) \Delta t\} \psi_2(X_j, t). \end{aligned} \quad (51)$$

(2) Advance the solution according to the linear part:

$$\begin{aligned} i\left(\frac{\partial \psi_1}{\partial t} + \frac{\delta \pi}{P} \frac{\partial \psi_1}{\partial X}\right) + \frac{1}{2} \frac{\pi^2}{P^2} \frac{\partial^2 \psi_1}{\partial X^2} &= 0, \\ i\left(\frac{\partial \psi_2}{\partial t} - \frac{\delta \pi}{P} \frac{\partial \psi_2}{\partial X}\right) + \frac{1}{2} \frac{\pi^2}{P^2} \frac{\partial^2 \psi_2}{\partial X^2} &= 0, \end{aligned} \quad (52)$$

by means of computing

$$\begin{aligned} \hat{\psi}_1(X_k, t + \Delta t) &= F(\hat{\psi}_1(X_j, t + \Delta t))_k, \\ \hat{\psi}_2(X_k, t + \Delta t) &= F(\hat{\psi}_2(X_j, t + \Delta t))_k, \end{aligned} \quad (53)$$

followed by

$$\begin{aligned} \tilde{\psi}_1(X_k, t + \Delta t) &= \exp\left\{i\left(-\frac{1}{2} \frac{\pi^2}{P^2} k^2 - \frac{\pi}{P} \delta k\right) \Delta t\right\} \hat{\psi}_1(X_k, t + \Delta t), \\ \tilde{\psi}_2(X_k, t + \Delta t) &= \exp\left\{i\left(-\frac{1}{2} \frac{\pi^2}{P^2} k^2 + \frac{\pi}{P} \delta k\right) \Delta t\right\} \hat{\psi}_2(X_k, t + \Delta t), \end{aligned} \quad (54)$$

and

$$\begin{aligned} \psi_1(X_j, t + \Delta t) &= F^{-1}(\tilde{\psi}_1(X_k, t + \Delta t))_j, \\ \psi_2(X_j, t + \Delta t) &= F^{-1}(\tilde{\psi}_2(X_k, t + \Delta t))_j, \end{aligned} \quad (55)$$

where the transform F and its inverse F^{-1} are given by (48) and (49), respectively.

Similarly, the advancement in time from t to $t + \Delta t$ by the split-step Fourier method using the second order splitting approximation (7) is described in the following three steps:

(1') Advance the solution using the nonlinear part (50) through the following scheme

$$\begin{aligned} \tilde{\psi}_1(X_j, t + \frac{1}{2} \Delta t) &= \exp\left\{i\left(\left|\psi_1(X_j, t)\right|^2 + \mu \left|\psi_2(X_j, t)\right|^2\right) \frac{1}{2} \Delta t\right\} \psi_1(X_j, t), \\ \tilde{\psi}_2(X_j, t + \frac{1}{2} \Delta t) &= \exp\left\{i\left(\mu \left|\psi_1(X_j, t)\right|^2 + \left|\psi_2(X_j, t)\right|^2\right) \frac{1}{2} \Delta t\right\} \psi_2(X_j, t). \end{aligned}$$

(2') Advance the solution according to the linear part (52) by means of the discrete Fourier transforms

$$\begin{aligned}\tilde{\psi}_1(X_j, t + \frac{1}{2}\Delta t) &= F^{-1}(\exp\{i(-\frac{1}{2}\frac{\pi^2}{P^2}k^2 - \frac{\pi}{P}\delta k)\Delta t\}F(\tilde{\psi}_1(X_j, t + \frac{1}{2}\Delta t))), \\ \tilde{\psi}_2(X_j, t + \frac{1}{2}\Delta t) &= F^{-1}(\exp\{i(-\frac{1}{2}\frac{\pi^2}{P^2}k^2 + \frac{\pi}{P}\delta k)\Delta t\}F(\tilde{\psi}_1(X_j, t + \frac{1}{2}\Delta t))),\end{aligned}$$

(3') Advance the solution using the nonlinear part (50) through the following scheme

$$\begin{aligned}\psi_1(X_j, t + \Delta t) &= \exp\{i(\left|\tilde{\psi}_1(X_j, t + \frac{1}{2}\Delta t)\right|^2 + \mu\left|\tilde{\psi}_2(X_j, t + \frac{1}{2}\Delta t)\right|^2)\frac{1}{2}\Delta t\}\tilde{\psi}_1(X_j, t + \frac{1}{2}\Delta t), \\ \psi_2(X_j, t + \Delta t) &= \exp\{i(\mu\left|\tilde{\psi}_1(X_j, t + \frac{1}{2}\Delta t)\right|^2 + \left|\tilde{\psi}_2(X_j, t + \frac{1}{2}\Delta t)\right|^2)\frac{1}{2}\Delta t\}\tilde{\psi}_2(X_j, t + \frac{1}{2}\Delta t).\end{aligned}$$

Numerical simulation of the CNLS using the Crank Nicolson Method

The CNLS equation to be considered is

$$\begin{aligned}i\partial U / \partial z + (\cos \theta \cdot \sigma_3 + \sin \theta \cdot \sigma_1)(\Delta \beta U + i\Delta \beta' \partial U / \partial t) - 1/2\beta'' \partial^2 U / \partial^2 t \\ + \gamma[U^2|U - 1/3(U^\perp \sigma_2 U)\sigma_2 U] = 0,\end{aligned}\tag{1}$$

where:

$$U = \begin{bmatrix} u_1 \\ u_2 \end{bmatrix}, \quad U^\perp = \begin{pmatrix} u_1^* & u_2^* \end{pmatrix},$$

$$\beta'' = d(z), U^\perp \sigma_2 U = -i(u_1^* u_2 - u_2^* u_1),$$

$$\gamma = g(z), \quad \sigma_3 = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}, \quad \sigma_1 = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}, \quad \sigma_2 = \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix},$$

$$g(z) = \left(2\Gamma / (1 - e^{2\Gamma z_a})\right) e^{-2\Gamma z}, n z_a \prec z \prec (n+1)z_a$$

$$\theta = \theta(z) = \theta_n, n z_p \prec z \prec (n+1)z_p, \quad z_p = z_a / M,$$

M is an integer, say M=10, and θ_n is a uniform random variable in $[0, 2\pi]$. In other words, polarization fluctuations happen over shorter scales. The system in (1) can be written as a coupled system in the following manner:

$$iu_{1z} + \Delta\beta(u_1 \cos(\theta) + u_2 \sin(\theta)) + i\Delta\beta'(\cos(\theta).u_{1t} + \sin(\theta).u_{2t}) \\ - 1/2\beta''u_{1tt} + \gamma[u_1(|u_1|^2 + |u_2|^2) - 1/3(u_1|u_2|^2 - u_1^*u_2^2)] = 0$$

$$iu_{2z} - \Delta\beta(u_2 \cos(\theta) - u_1 \sin(\theta)) - i\Delta\beta'(\cos(\theta).u_{2t} - \sin(\theta).u_{1t}) \\ - 1/2\beta''u_{2tt} + \gamma[u_2(|u_1|^2 + |u_2|^2) + 1/3(u_1^2u_2^* - |u_1|^2u_2)] = 0$$

Initial Conditions:

$$U(t,0) = \sum_{j=1}^n U_j(t),$$

$$U_j(t) = C_j A_j \text{Sech}(A_j(t - t_j)) e^{i\Omega_j t + \Phi_j},$$

$$C_j = \begin{pmatrix} \cos(\alpha_j) & e^{i\varphi_j} \\ \sin(\alpha_j) & e^{-i\varphi_j} \end{pmatrix},$$

Parameters:

n (the number of solutions), $A_1, \dots, A_n, \Omega_1, \dots, \Omega_n, \Phi_1, \dots, \Phi_n, t_1, \dots, t_n, \alpha_1, \dots, \alpha_n, \varphi_1, \dots, \varphi_n$

$$\Gamma = 4, z_a = 0.05 \rightarrow 0.2,$$

d_1, d_2, z_1, z_2 , (parameters for the dispersion map, with $z_1 + z_2 = z_a$)

$$z_p, \Delta\beta, \Delta\beta'$$

$$d(z) = \begin{cases} d_1, 0 \leq z < \theta.z_m, 0 \leq \theta < 1.0 \\ d_2, \theta.z_m \leq z \leq z_m, z_m = 0.01 \rightarrow 1.0 \end{cases}$$

As for values to use, initially, we take:

- d(z)=1 (constant dispersion),

- Gamma= 10, z_a= 0.1,
- z_p= 0.01,
- N=1, A=1, Omega=0, T=0, Phi=0.
- $\Delta\beta = 0.1$, $\Delta\beta' = 0.1$
- propagate up to z_max= 20.

In this thesis, we take:

- > $\beta''' = 0$ [no third-order dispersion]
- > $\beta'' = d(z)$ [dispersion management]
- > $g = 0$ and $\gamma = g(z)$
- > $\Delta\beta$ and $\Delta\beta'$ are free parameters.

Ismail and Taha (to be submitted) developed the following Crank Nicholson method for solving the CNLS equation:

$$u_{1t} + \beta_1(\cos(\theta)u_2 + \sin(\theta)u_4) + \beta_2(\cos(\theta)u_{1x} + \sin(\theta)u_{3x}) - \frac{1}{2}\beta_3u_{2xx} + \gamma_2z - \frac{2\gamma}{3}(u_2u_3^2 - u_1u_3u_4) = 0$$

$$u_{2t} - \beta_1(\cos(\theta)u_1 + \sin(\theta)u_3) + \beta_2(\cos(\theta)u_{2x} + \sin(\theta)u_{4x}) - \frac{1}{2}\beta_3u_{1xx} - \gamma_1z + \frac{2\gamma}{3}(u_1u_4^2 - u_2u_3u_4) = 0$$

$$u_{3t} - \beta_1(\cos(\theta)u_4 - \sin(\theta)u_2) - \beta_2(\cos(\theta)u_{3x} - \sin(\theta)u_{1x}) - \frac{1}{2}\beta_3u_{4xx} - \gamma_4z - \frac{2\gamma}{3}(u_4u_1^2 - u_1u_2u_3) = 0$$

$$u_{4t} + \beta_1(\cos(\theta)u_3 - \sin(\theta)u_1) - \beta_2(\cos(\theta)u_{4x} - \sin(\theta)u_{2x}) + \frac{1}{2}\beta_3u_{3xx} - \gamma_3z - \frac{2\gamma}{3}(u_3u_2^2 - u_1u_2u_4) = 0$$

where

$$z = u_1^2 + u_2^2 + u_3^2 + u_4^2$$

$$\beta_1 = \Delta\beta$$

$$\beta_2 = \Delta\beta'$$

$$\beta_3 = \Delta\beta''$$

with the Crank Nicolson scheme:

$$\begin{aligned} & \frac{2}{k} (U_{1,m}^{n+1} - U_{1,m}^n) + p_3 (cs U_{2,m}^{n+1} + sn U_{4,m}^{n+1}) + p_1 (cs (U_{2,m+1}^{n+1} - U_{1,m-1}^{n+1}) + sn (U_{3,m+1}^{n+1} - U_{3,m-1}^{n+1})) - \\ & p_2 (U_{2,m+1}^{n+1} - 2U_{2,m}^{n+1} + U_{2,m-1}^{n+1}) + \mathcal{H}_{2,m}^{n+1} Z_m^{n+1} - \frac{2\gamma}{3} [U_2 U_3^2 - U_1 U_3 U_4]_m^{n+1} \\ & + p_3 (cs U_{2,m}^n + sn U_{4,m}^n) + p_1 (cs (U_{1,m+1}^n - U_{1,m-1}^n) + sn (U_{3,m+1}^n - U_{3,m-1}^n)) - \\ & p_2 (U_{2,m+1}^n - 2U_{2,m}^n + U_{2,m-1}^n) + \mathcal{H}_{2,m}^n Z_m^n - \frac{2\gamma}{3} [U_2 U_3^2 - U_1 U_3 U_4]_m^n = 0 \end{aligned}$$

$$\begin{aligned} & \frac{2}{k} (U_{2,m}^{n+1} - U_{2,m}^n) - p_3 (cs U_{1,m}^{n+1} + sn U_{3,m}^{n+1}) + p_1 (cs (U_{2,m+1}^{n+1} - U_{2,m-1}^{n+1}) + sn (U_{4,m+1}^{n+1} - U_{4,m-1}^{n+1})) + \\ & p_2 (U_{1,m+1}^{n+1} - 2U_{1,m}^{n+1} + U_{1,m-1}^{n+1}) - \mathcal{H}_{1,m}^{n+1} Z_m^{n+1} + \frac{2\gamma}{3} [U_1 U_4^2 - U_2 U_3 U_4]_m^{n+1} \\ & - p_3 (cs U_{1,m}^n + sn U_{3,m}^n) + p_1 (cs (U_{2,m+1}^n - U_{2,m-1}^n) + sn (U_{4,m+1}^n - U_{4,m-1}^n)) + \\ & p_2 (U_{1,m+1}^n - 2U_{1,m}^n + U_{1,m-1}^n) - \mathcal{H}_{1,m}^n Z_m^n + \frac{2\gamma}{3} [U_1 U_4^2 - U_2 U_3 U_4]_m^n = 0 \end{aligned}$$

$$\begin{aligned} & \frac{2}{k} (U_{3,m}^{n+1} - U_{3,m}^n) - p_3 (cs U_{4,m}^{n+1} - sn U_{2,m}^{n+1}) - p_1 (cs (U_{3,m+1}^{n+1} - U_{3,m-1}^{n+1}) - sn (U_{1,m+1}^{n+1} - U_{1,m-1}^{n+1})) - \\ & p_2 (U_{4,m+1}^{n+1} - 2U_{4,m}^{n+1} + U_{4,m-1}^{n+1}) + \mathcal{H}_{4,m}^{n+1} Z_m^{n+1} - \frac{2\gamma}{3} [U_4 U_1^2 - U_1 U_2 U_3]_m^{n+1} \\ & - p_3 (cs U_{4,m}^n - sn U_{2,m}^n) - p_1 (cs (U_{3,m+1}^n - U_{3,m-1}^n) - sn (U_{1,m+1}^n - U_{1,m-1}^n)) - \\ & p_2 (U_{4,m+1}^n - 2U_{4,m}^n + U_{4,m-1}^n) - \mathcal{H}_{4,m}^n Z_m^n - \frac{2\gamma}{3} [U_4 U_1^2 - U_1 U_2 U_3]_m^n = 0 \end{aligned}$$

$$\begin{aligned} & \frac{2}{k} (U_{4,m}^{n+1} - U_{4,m}^n) + p_3 (cs U_{3,m}^{n+1} - sn U_{1,m}^{n+1}) - p_1 (cs (U_{4,m+1}^{n+1} - U_{4,m-1}^{n+1}) - sn (U_{2,m+1}^{n+1} - U_{2,m-1}^{n+1})) + \\ & p_2 (U_{3,m+1}^{n+1} - 2U_{3,m}^{n+1} + U_{3,m-1}^{n+1}) - \mathcal{H}_{3,m}^{n+1} Z_m^{n+1} + \frac{2\gamma}{3} [U_3 U_2^2 - U_1 U_2 U_4]_m^{n+1} \end{aligned}$$

$$+ p_3 (cs U_{3,m}^n - sn U_{1,m}^n) - p_1 (cs (U_{4,m+1}^n - U_{4,m-1}^n) - sn (U_{2,m+1}^n - U_{2,m-1}^n)) + \\ p_2 (U_{3,m+1}^n - 2U_{3,m}^n + U_{3,m-1}^n) - \gamma U_{3,m}^n Z_m^n + \frac{2\gamma}{3} [U_3 U_2^2 - U_1 U_2 U_4]_m^n = 0$$

where

$$cs = \cos(\theta)$$

$$sn = \sin(\theta)$$

$$Z = U_1^2 + U_2^2 + U_3^2 + U_4^2$$

$$p_1 = \frac{\beta_2}{2h}$$

$$p_2 = \frac{\beta_3}{2h^2}$$

$$p_3 = \beta_1$$

and they use the boundary conditions

$$\frac{U_{m+1}^{n+s} - U_{m-1}^{n+s}}{2h} = 0 \text{ where } s=0, 1 \text{ at the boundaries i.e. at } (m=1, N).$$

The CNLS equation implementation was originally implemented in FORTRAN. The Fortran code was initially converted to C++ by Shanshan Ding. However, this version had only a few decimal digits of accuracy and the results did not match with the Fortran code. After further updates and revision by Taha and Foster, a final C++ version was developed and the results from the C++ and Fortran implementation versions match.

The process of matching the results of the C++ and Fortran implementations occurred in six changes:

1. The Fortran implementation was edited to declare each variable used by a subroutine.

If a subroutine has n arguments in its input parameter, then all n arguments were declared according to the argument's specific data type after the specification of the subroutine. The elimination of this change yielded erroneous results.

2. The Fortran constant values needed to be converted from floating point to double precision data types. For example, every occurrence of a hard-coded decimal value such as 50.0 was converted to the decimal precision format 50.D0 for hard-coded values. This change increased the decimal digit precision of the results considerably.
3. Later, a logic error was noticed in the C++ code based on the Fortran implementation. Upon comparing each algorithm between the implementations, the logic error was noticed and corrected.
4. Upon further testing, it was discovered that the C++ double variable name, CS, used within the implementation did not contain the value expected. Upon changing all occurrences of this variable name, the variable then produced the expected value.
5. To compare the Fortran and C++ in precision based on the output results, both the Fortran and C++ implementations were edited to display the output with a specific number of decimal digits. For our purposes, the 14 decimal digits of were chosen to be displayed.
6. To further match the C++ and Fortran results, every output statement, including comments, was compared and changed accordingly. After producing results from each implementation to compare, the UNIX diff command was used to display the differences in the output results. Later, these differences, if any, were eliminated.

CHAPTER 4

NUMERICAL SIMULATION OF THE COMPLEX MODIFIED KORTEWEG-DE VRIES (CMKDV) EQUATION

4.1 Introduction

In this chapter, the numerical methods for solving the complex modified Korteweg-de Vries (CMKdV) equation are presented.

For the CMKdV equation, five numerical schemes by Taha and Liu [2] were simulated. These schemes include First Order, Second Order, Fourth Order, IST, and Finite Difference. The methods for solving each one will be presented.

For this chapter, the following CMKdV equation was considered

$$q_t + 6|q|^2 q_x + q_{xxx} = 0, x \in [a, b], 0 \leq t \leq T$$

where $q = q(x, t)$ is a complex valued function. When this equation is normalized to $[0, 2\pi]$, it becomes:

$$q_t + \bar{\alpha}|q|^2 q_x + \bar{\beta}q_{xxx} = 0, x \in [0, 2\pi]$$

where $\bar{\alpha} = \frac{12\pi}{b-a}$, $\bar{\beta} = \frac{(2\pi)^3}{(b-a)^3}$.

As presented in Chapter 2, the split-step Fourier scheme is then applied, which splits the equation into linear and nonlinear sub-equations.

The nonlinear sub-equation is given by

$$q_t + \bar{\alpha}|q|^2 q_x = 0$$

and the linear sub-equation is given by

$$q_t + \bar{\beta}q_{xxx} = 0 .$$

The linear sub-equation can be solved by the discrete Fourier transform (DFT) discussed in Chapter 2. However, the nonlinear sub-equation can be solved by different split-step numerical schemes, developed by Muslu and Erbay, which were also introduced in Chapter 2. The numerical schemes that were studied and simulated include: the first-order, second-order, fourth-order, inverse scattering transform, finite difference, and Taha local scheme. In the next section, I present these numerical schemes.

4.2 Methods for Solving the CMKdV Equation

CMKdV First Order Scheme

To solve the first-order scheme for the CMKdV equation, the following steps were performed:

1. Set the time step to Δt and solve the nonlinear equation.
2. Use the discrete Fourier transform to solve the linear equation.

CMKdV Second Order Scheme

To solve the second-order scheme for the CMKdV equation, the following steps were performed:

1. Set the time step to $\frac{1}{2}\Delta t$ and solve the nonlinear equation.
2. Use the discrete Fourier transform to solve the linear equation.
3. Set the time step to $\frac{1}{2}\Delta t$ and solve the nonlinear equation.

CMKdV Fourth Order Scheme

To solve the fourth-order scheme for the CMKdV equation, the following steps were performed:

1. Replace Δt with $\lambda\Delta t$ and use the second order scheme to perform the computation.
2. Replace Δt with $(1 - 2\lambda)\Delta t$ and use the second order scheme to perform the computation.
3. Once again, replace Δt with $\lambda\Delta t$ and use the second order scheme to perform the computation.

CMKdV Inverse Scattering Transform (IST) Scheme

To assist in solving the IST scheme, Taha [5] developed a discretization formula for the CMKdV nonlinear sub-equation:

$$\begin{aligned}
q_n^{m+1} = & q_n^m - \frac{\bar{\alpha}}{12} \frac{\Delta t}{\Delta X} \left(q_{n+2}^{m+1} \left(|q_{n+1}^m|^2 + |q_n^m|^2 \right) - q_{n-2}^m \left(|q_n^{m+1}|^2 + |q_{n-1}^{m+1}|^2 \right) \right. \\
& + \frac{q_{n+1}^{m+1}}{2} \left((q^*)_n^m q_{n+1}^m + (q^*)_n^{m+1} q_{n+1}^{m+1} + 2q_n^m (q^*)_{n-1}^m \right) \\
& - \frac{q_{n-1}^m}{2} \left(q_{n-1}^{m+1} (q^*)_n^{m+1} + q_{n-1}^m (q^*)_n^m + 2q_n^{m+1} (q^*)_{n+1}^{m+1} \right) \\
& + \frac{q_n^m}{2} \left((q^*)_n^{m+1} q_{n-1}^{m+1} + (q^*)_n^m q_{n-1}^m \right) \\
& - \frac{q_n^{m+1}}{2} \left((q^*)_n^{m+1} q_{n-1}^{m+1} + (q^*)_n^m q_{n-1}^m \right) \\
& \left. + 3 \left(|q_n^m|^2 q_{n+1}^{m+1} - |q_n^{m+1}|^2 q_{n-1}^m \right) \right)
\end{aligned}$$

To solve the inverse scattering transform (IST) scheme for the CMKdV equation, the following steps were performed:

1. Use the local IST approximation discussed by Taha [5] to solve the nonlinear equation.
2. Use the Fourier transform to solve the linear equation.

CMKdV Finite Difference (FD) Scheme

To assist in solving the FD scheme, Taha and Ablowitz [7] developed a finite difference approximation for the CMKdV nonlinear sub-equation. Later, this equation was modified by Taha and Liu [2] and is represented by the following:

$$\begin{aligned}
q_n^{m+1} = & q_n^m - \frac{\bar{\alpha}}{48} \frac{\Delta t}{\Delta X} \left\{ |q_n^{m+1}|^2 (8q_{n+1}^{m+1} - 8q_{n-1}^{m+1} - q_{n+2}^{m+1} + q_{n-2}^{m+1}) \right. \\
& \left. + |q_n^m|^2 (8q_{n+1}^m - 8q_{n-1}^m - q_{n+2}^m + q_{n-2}^m) \right\}
\end{aligned}$$

To solve the finite difference (FD) scheme for the CMKdV equation, the following steps were performed:

1. Use the local IST approximation developed by Taha and Ablowitz [7] to solve the nonlinear equation.
2. Use the Fourier transform to solve the linear equation.

The CMKdV equation has been used for simulation with initial conditions related to the traveling wave solution and the double homoclinic orbit [5].

For the traveling-wave solution, the exact solution of the CMKdV equation

$$q_t + 6|q|^2 q_x + q_{xxx} = 0, x \in [a, b], 0 \leq t \leq T$$

is

$$q(x, t) = a \exp[i(kx - \omega t)]$$

where ω satisfies the dispersion relation, $\omega = 6|a|^2 k - k^3$ and a is the complex amplitude. The initial conditions $t=0$, $a = 0.5$, and $k = 1$ were used along with periodic boundary conditions on the interval $[0, 4\pi]$. The five numerical schemes discussed above were implemented with conditions of the traveling-wave solution. The double homoclinic orbit can be represented as the exact solution of the CMKdV equation

$$q_t + 6|q|^2 q_x + q_{xxx} = 0, x \in [a, b], 0 \leq t \leq T$$

with initial condition

$$q(x, t) = a \exp(ikx)(1 + \varepsilon_0 i \cos \mu_n x),$$

where $a = 0.5$, $\varepsilon_0 = 0.1$, $k = \mu_n = 2\pi / L$, and $L = 4\sqrt{2}\pi$ with periodic boundary conditions on the interval $[0, L]$. The taha local scheme mentioned next was implemented with conditions of the double homoclinic orbit.

Taha developed a numerical scheme, Taha local scheme, for solving the CMKdV equation, subject to conditions of the double homoclinic orbit. This scheme was originally implemented in the Fortran programming language. However, it was the desire of Taha to obtain C and C++ versions of this implementation. After successfully converting the Fortran code to C++, I was assigned the task of producing also a C version. After converting the C++ implementation back to C, it was determined that the C++ version of the complex library used in the C++ implementation was not compatible with the C implementation. To accommodate this incompatibility, a C version of the complex library was implemented to produce a compilation. The conversion of the Fortran code into C and C++ versions was successful. The output and data results from each simulation matched in syntax and decimal digit accuracy. The final C++ version's simulation output was later converted and formatted for the successful simulation of the Taha local scheme of the CMKdV equation on the Equation Server, which will be discussed later in Chapter 6.

Numerical Simulation

All the schemes presented in this chapter were simulated on the Equation Server, which will be discussed in Chapter 6. The following figures contain a gnuplot plot of each scheme generated by the Equation Server.

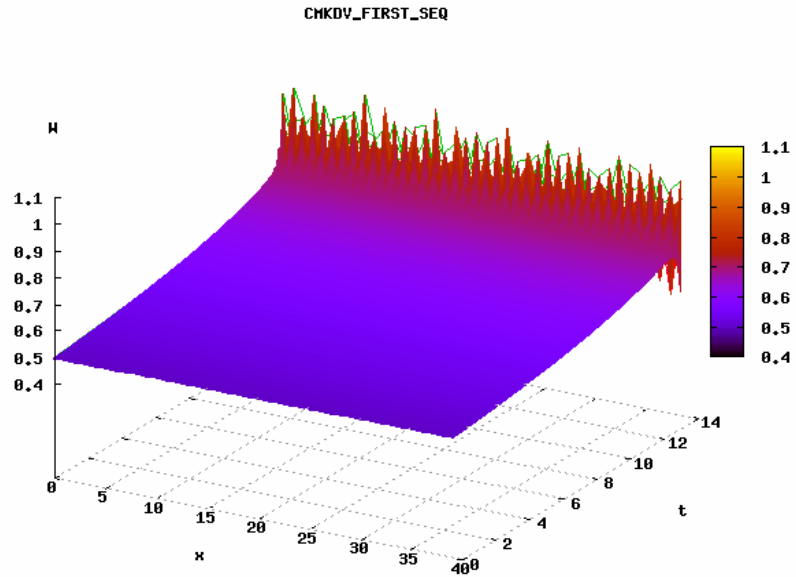


Figure 4.2.1 Equation Server gnuplot of CMKdV First Order Scheme ($N = 40$ and $0 \leq t < 13.5875$)

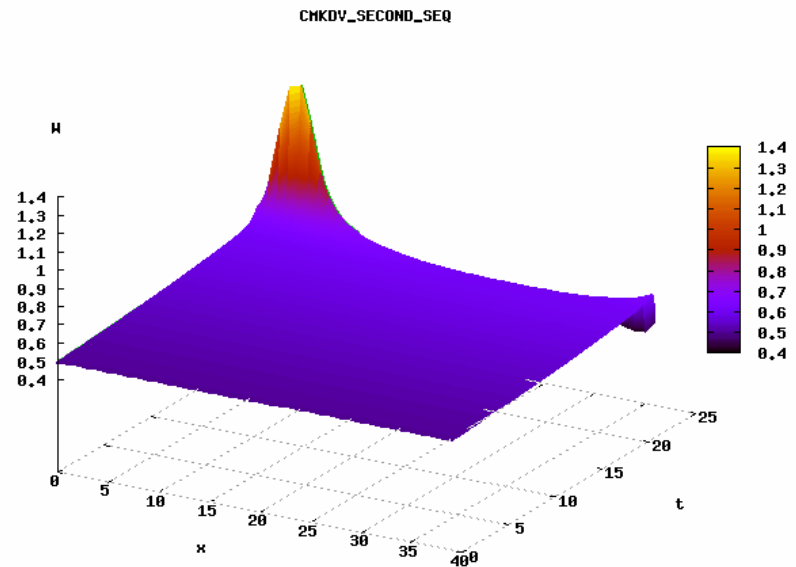


Figure 4.2.2 Equation Server gnuplot of CMKdV Second Order Scheme ($N = 40$ and $0 \leq t < 21.775$)

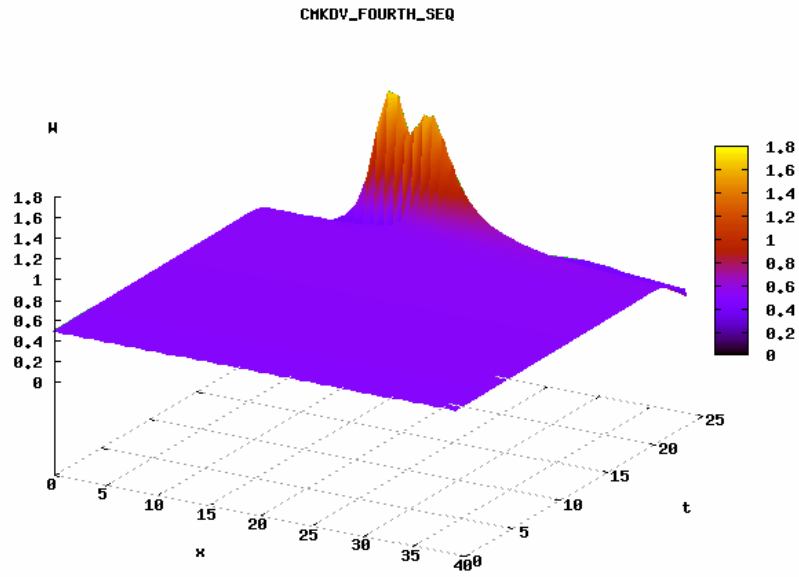


Figure 4.2.3 Equation Server gnuplot of CMKdV Fourth Order Scheme ($N = 40$ and $0 \leq t < 24.3875$)

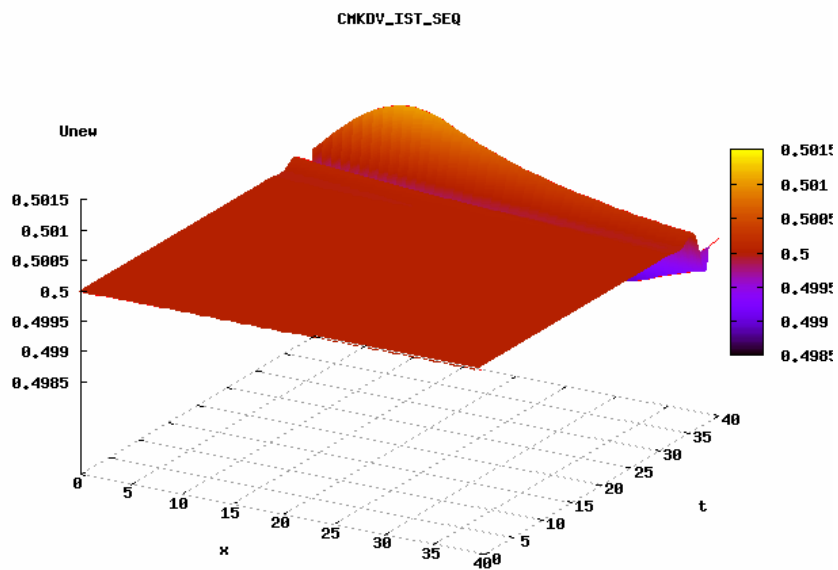


Figure 4.2.4 Equation Server gnuplot of CMKdV IST Scheme ($N = 40$ and $0 \leq t < 40$)

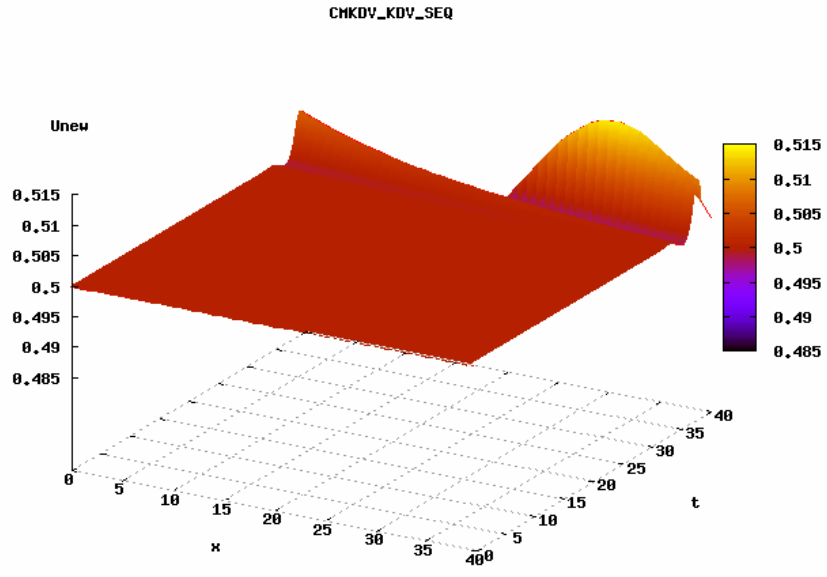


Figure 4.2.5 Equation Server gnuplot of CMKdV FD Scheme ($N = 40$ and $0 \leq t < 40$)

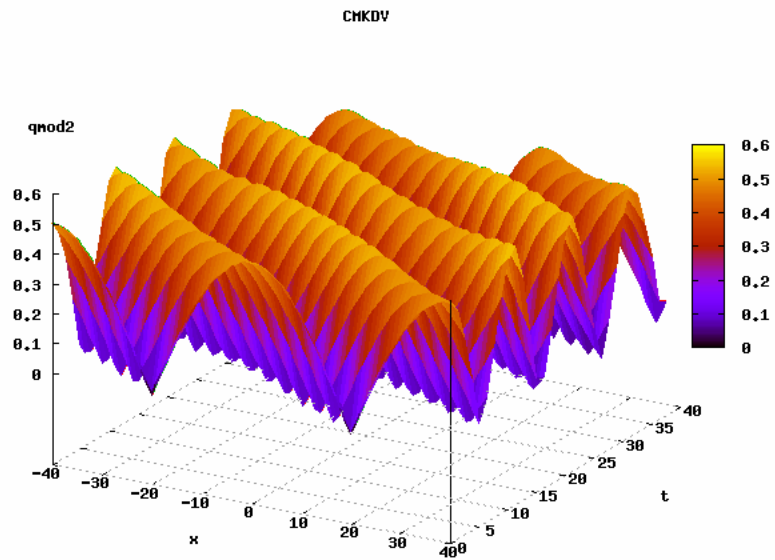


Figure 4.2.6 Equation Server gnuplot of CMKdV Taha Local Scheme ($N = 40$ and $0 \leq t < 40$)

CHAPTER 5

PARALLEL (FFTW-MPI) IMPLEMENTATION OF THE CMKdV EQUATION

5.1 Introduction

In this chapter, we discuss parallel (FFTW-MPI) implementations of the CMKdV equation developed by Taha and Liu [2]. These implementations include the parallel algorithms for the first-order, second-order, fourth-order, finite difference, and inverse scattering transform methods for solving the CMKdV equation. The Jumpshot-4 profiling tool, which I present in chapter 7, was used to analyze the communication between processors that occurred during the numerical simulations.

5.2 Parallel Methods for Solving CMKdV Equation

Taha and Lui developed five parallel schemes for the solving the CMKdV equation. These schemes include First Order, Second Order, Fourth Order, IST, and Finite Difference. These schemes were implemented and utilized FFTW-MPI capability. Each of these schemes were compiled on the taha 4 processor machine and simulated. Next, I will cover each one.

To assist in solving these schemes, Taha and Liu developed a parallel version of the discrete Fourier transform, Runge-Kutta method, and the finite difference approximation [2].

Parallel CMKdV First Order Scheme

Recall that to solve the first-order scheme for the CMKdV equation, the following steps were performed:

1. Set the time step to Δt and solve the nonlinear equation.
2. Use the discrete Fourier transform to solve the linear equation.

To solve the parallel first-order scheme for the CMKdV equation, the following steps were performed:

The master processor distributes the initial values $q_j^0, 0 \leq j < N$ among the p processors.

For $m = 0$ to $T_n - 1$, Do

1. Use the parallel DFT to compute the forward Fourier transform, $F_k(q_j^m)$.
2. The master collects the results of $F_k(q_j^m)$ and computes the product, $ikF_k(q_j^m)$.
3. Use the parallel DFT to compute the inverse transform, $F_k^{-1}(ikF_k(q_j^m))$.
4. The master collects the results of $F_k^{-1}(ikF_k(q_j^m))$.
5. Use the parallel Runge-Kutta method for the time integration.
6. The master collects the solution of the nonlinear sub-equation.
7. Use the parallel DFT to compute the forward Fourier transform, $F_k(q_j^m)$.
8. The master collects the results $F_k(q_j^m)$ and computes $\exp(i\bar{\beta}k^3\Delta t)F_k(q_j^m)$.
9. Use the parallel DFT to compute the inverse transform.
10. The master collects the solution of the linear sub-equation, which give new values

$$q_j^{m+1}.$$

Steps 1 through 5 solve the nonlinear sub-equation and steps 7 through 9 solve the linear sub-equation.

Parallel CMKdV Second Order Scheme

Recall that to solve the second-order scheme for the CMKdV equation, the following steps were performed:

1. Set the time step to $\frac{1}{2}\Delta t$ and solve the nonlinear equation.
2. Use the discrete Fourier transform to solve the linear equation.
3. Set the time step to $\frac{1}{2}\Delta t$ and solve the nonlinear equation.

To solve the parallel first-order scheme for the CMKdV equation, the following steps were performed:

The master processor distributes the initial values $q_j^0, 0 \leq j < N$ among the p processors.

For $m = 0$ to $T_n - 1$, Do

1. Use the parallel DFT to compute the forward Fourier transform, $F_k(q_j^m)$.
2. The master collects the results of $F_k(q_j^m)$ and computes the product, $ikF_k(q_j^m)$.
3. Use the parallel DFT to compute the inverse transform, $F_k^{-1}(ikF_k(q_j^m))$.
4. The master collects the results of $F_k^{-1}(ikF_k(q_j^m))$.
5. Use the parallel Runge-Kutta method for the time integration where the time step

Δt is replaced by $\frac{1}{2}\Delta t$.

6. The master collects the solution of the nonlinear sub-equation.
7. Use the parallel DFT to compute the forward Fourier transform, $F_k(q_j^m)$.
8. The master collects the results $F_k(q_j^m)$ and computes $\exp(i\bar{\beta}k^3\Delta t)F_k(q_j^m)$.
9. Use the parallel DFT to compute the inverse transform.
10. The master collects the solution of the linear sub-equation.
11. Repeat steps one through 5.
12. The master collects the results, which give the new values q_j^{m+1} .

Steps 1 through 5 solve the nonlinear sub-equation and steps 7 through 9 and 11 solve the linear sub-equation.

Parallel CMKdV Fourth Order Scheme

Recall that to solve the fourth-order scheme for the CMKdV equation, the following steps were performed:

1. Replace Δt with $\lambda\Delta t$ and use the second order scheme to perform the computation.
2. Replace Δt with $(1-2\lambda)\Delta t$ and use the second order scheme to perform the computation.
3. Once again, replace Δt with $\lambda\Delta t$ and use the second order scheme to perform the computation.

To solve the parallel fourth-order scheme for the CMKdV equation, the following steps were performed:

The master processor distributes the initial values $q_j^0, 0 \leq j < N$ among the p processors.

For $m = 0$ To $T_n - 1$, Do

1. Call the main body of second-order scheme (steps 1 through 12) with Δt replace by $\lambda \Delta t$.
2. Call the second-order scheme with Δt replace by $(1 - 2\lambda)\Delta t$.
3. Repeat step 1. The results give the new values q_j^{m+1} .

Parallel CMKdV Inverse Scattering Transform (IST) Scheme

Recall that to assist in solving the IST scheme, Taha [5] developed a discretization formula for the CMKdV nonlinear sub-equation:

$$\begin{aligned}
 q_n^{m+1} = & q_n^m - \frac{\bar{\alpha}}{12} \frac{\Delta t}{\Delta X} \left(q_{n+2}^{m+1} \left(|q_{n+1}^m|^2 + |q_n^m|^2 \right) - q_{n-2}^m \left(|q_n^{m+1}|^2 + |q_{n-1}^{m+1}|^2 \right) \right. \\
 & + \frac{q_{n+1}^{m+1}}{2} \left((q^*)_n^m q_{n+1}^m + (q^*)_n^{m+1} q_{n+1}^{m+1} + 2q_n^m (q^*)_{n-1}^m \right) \\
 & - \frac{q_{n-1}^m}{2} \left(q_{n-1}^{m+1} (q^*)_n^{m+1} + q_{n-1}^m (q^*)_n^m + 2q_n^{m+1} (q^*)_{n+1}^{m+1} \right) \\
 & + \frac{q_n^m}{2} \left((q^*)_n^{m+1} q_{n-1}^{m+1} + (q^*)_n^m q_{n-1}^m \right) \\
 & - \frac{q_n^{m+1}}{2} \left((q^*)_n^{m+1} q_{n-1}^{m+1} + (q^*)_n^m q_{n-1}^m \right) \\
 & \left. + 3 \left(|q_n^m|^2 q_{n+1}^{m+1} - |q_n^{m+1}|^2 q_{n-1}^m \right) \right)
 \end{aligned}$$

Recall that to solve the inverse scattering transform (IST) scheme for the CMKdV equation, the following steps were performed:

1. Using the local IST approximation discussed by Taha [5] to solve the nonlinear equation.
2. Use the Fourier transform to solve the linear equation.

To solve the parallel inverse scattering transform (IST) scheme for the CMKdV equation, the following steps were performed:

The master processor distributes the initial values $q_j^0, 0 \leq j < N$ among the p processors.

For $m = 0$ to $T_n - 1$, Do

1. Use the modified parallel finite difference (FD) approximation to solve the nonlinear sub-equation.
2. The master collects the solution of the nonlinear sub-equation.
3. Use the parallel Fourier transform to compute the forward Fourier transform $F_k(q_j^m)$.
4. The master collects the results $F_k(q_j^m)$, compute $\exp(i\bar{\beta}k^3\Delta t)F_k(q_j^m)$.
5. Use the parallel Fourier transform to compute the inverse transform.
6. The master collects the solution of the linear sub-equation, which give the new values q_j^{m+1} .

Parallel CMKdV Finite Difference (FD) Scheme

Recall that to assist in solving the FD scheme, Taha and Ablowitz [7] developed a finite difference approximation for the CMKdV nonlinear sub-equation. Later, this equation was modified by Taha and Liu [2] and is represented by the following:

$$q_n^{m+1} = q_n^m - \frac{\bar{\alpha}}{48} \frac{\Delta t}{\Delta X} \{ |q_n^{m+1}|^2 (8q_{n+1}^{m+1} - 8q_{n-1}^{m+1} - q_{n+2}^{m+1} + q_{n-2}^{m+1}) + |q_n^m|^2 (8q_{n+1}^m - 8q_{n-1}^m - q_{n+2}^m + q_{n-2}^m) \}$$

Recall that to solve the finite difference (FD) scheme for the CMKdV equation, the following steps were performed:

1. Using the FD approximation developed by Taha and Alblowitz [7] to solve the nonlinear equation.
2. Use the Fourier transform to solve the linear equation.

To solve the Parallel CMKdV Finite Difference (FD) scheme, the following steps were performed:

The master processor distributes the initial values $q_j^0, 0 \leq j < N$ among the p processors.

For $m = 0$ to $T_n - 1$, Do

1. Call the main body of fourth-order scheme to solve the nonlinear sub-equation.
2. The master collects the solution of the nonlinear sub-equation.
3. Use the parallel finite difference (FD) approximation to compute the forward Fourier transform $F_k(q_j^m)$.
4. The master collects the results $F_k(q_j^m)$, compute $\exp(i\bar{\beta}k^3\Delta t)F_k(q_j^m)$.
5. Use the parallel DFT to compute the inverse transform.
6. The master collects the solution of the linear sub-equation, which give the new values q_j^{m+1} .

Numerical Simulation

All the parallel schemes presented in this chapter were simulated through the Equation Server, which will be discussed in Chapter 6. The following tables contain performance results

from each scheme simulated on the taha 4 processor machine. Recall that speedup S_p is defined by

$$S_p = \frac{\text{Time spent to run the MPI code on one processor}}{\text{Time spent to run the MPI code on } p \text{ processors}}.$$

Efficiency E_p is defined by

$$E_p = \frac{\text{Speedup } S_p}{p \text{ processors}}$$

The goal of parallel implementation is to achieve efficiency closest to 1. From my simulations of the parallel algorithms of the CMKdV equation, as the value of N increased, efficiency increased. Also, as the number of processors, p, increased, there was an increase in speedup.

Table 5.1 CPU time and speedup for the parallel first-order scheme using FFTW

p	N=2048, Tn=1000	Speedup	Efficiency	N=131072, Tn=1000	Speedup	Efficiency
1	2.262113	1.000000	1.000000	337.890518	1.000000	1.000000
2	1.732446	1.305734	0.652867	200.162615	1.688080	0.844040
4	1.419344	1.593774	0.398443	117.609827	2.872979	0.718245

Table 5.2 CPU time and speedup for the parallel second-order scheme using FFTW

p	N=2048, Tn=1000	Speedup	Efficiency	N=131072, Tn=000	Speedup	Efficiency
1	3.143984	1.000000	1.000000	453.475492	1.000000	1.000000
2	2.484547	1.265415	0.632708	270.967933	1.673539	0.836770
4	2.003215	1.569469	0.392367	164.930097	2.749501	0.687375

Table 5.3 CPU time and speedup for the parallel fourth-order scheme using FFTW

p	N=2048, Tn=1000	Speedup	Efficiency	N=131072, Tn=1000	Speedup	Efficiency
1	9.028220	1.000000	1.000000	1195.996349	1.000000	1.000000
2	7.318890	1.233550	0.616775	736.085661	1.624806	0.812403
4	6.124466	1.474124	0.368531	388.465015	3.078775	0.769694

Table 5.4 CPU time and speedup for the parallel FD scheme using FFTW

p	N=2048, Tn=1000	Speedup	Efficiency	N=131072, Tn=1000	Speedup	Efficiency
1	2.176560	1.000000	1.000000	467.093942	1.000000	1.000000
2	1.435398	1.516346	0.748173	241.795503	1.931773	0.965886
4	1.293557	1.682616	0.420654	141.797573	3.294090	0.823522

Table 5.5 CPU time and speedup for the parallel IST scheme using FFTW

P	N=2048, Tn=1000	Speedup	Efficiency	N=131072, Tn=1000	Speedup	Efficiency
1	2.710533	1.000000	1.000000	499.627979	1.000000	1.000000
2	1.435398	1.516346	0.758173	249.984730	1.998634	0.999317
4	1.293557	1.682616	0.420654	148.675250	3.360532	0.840133

CHAPTER 6

WEB BASED INTERFACE FOR NUMERICAL SIMULATION

6.1 Introduction

In this chapter, we introduce Equation Server, a web based graphical user interface for the numerical simulation of nonlinear evolution equations. The concept of an equation server was suggested by Taha. After implementing various versions and prototypes, an acceptable version of the Equation Server was produced. In the next section, I will present Equation Server's purpose, architecture and capability.

6.2 Equation Server

It is the desire of many researchers to further develop numerical methods and make these equations, their results and plots accessible to users through the internet. The output results of such equations contain a very large amount of numbers, vary based on user input, and are often analyzed and interpreted in the form of a graph or plot [1]. Equation Server is a web based graphical user interface for solving such equations.

Equation Server was developed so that it could provide the developer an easier and more flexible way of implementing simple, complex, and parallel algorithms on the web than other existing web servers and technologies. The Equation Server consists of two main components:

a) the client machine, which is the web browser, and b) the server machine(s) where the Equation Solver is installed (Figure 6.2.3).

A user of the client machine would interact with the Equation Solver through a web browser. The web browser allows the user to select an available equation and provide input for that equation. The user's input is sent through a TCP/IP connection to the Equation Solver for further processing. Because the Equation Solver obtains input from the user through a web browser, the developer has the ability to integrate the Equation Solver into any webpage which serves as the graphical user interface (GUI). This also gives the developer the ability to manipulate how the Equation Solver results are displayed to the user.

The Solver includes: a) the equation input solver, b) the numerical method solver, and c) the equation result solver. As shown in Figure 6.2.3: The equation input solver initializes the Equation Solver by sending to the user a web page which asks for an equation selection (Figure 6.2.1).

After the user selects an equation, the equation input solver sends the chosen equation's numerical method page back to the user in order to specify the equation's numerical methods available to solve the method (Figure 6.2.2). After the user selects a numeral method for the selected equation, the equation input solver sends the chosen numerical method's input page back to the user in order to specify the equation's input values (Figure 6.2.4). After the user specifies the input values, they are sent to the numerical method which solves the selected equation.

Since most equation results are displayed in plot form using visualization tools such as Matlab, Maple, Mathematica, gnuplot, etc..., gnuplot, an interactive data and function plotting utility, was integrated into the Equation Server. Gnuplot was chosen due to its plotting

capability through the use of a command line interface. The plot produced assists in the interpretation and visualization of the data results.

Gnuplot provides the fantastic ability of plotting data on a variety of data sets. Traditionally, two-dimensional plots and three-dimensional plots, along with title and labeling capability, can be generated through gnuplot (Figure 6.2.6, Figure 6.2.9). Gnuplot also provides a variety of other tools and features to assist in a plotting task. Currently, gnuplot is freely available on the Internet (www.gnuplot.info).

After solving the equation, the results (Figure 6.2.5) are passed to gnuplot which generates a plot of the data (Figure 6.2.6). Because the generated results are passed to gnuplot, the equation method's output must be produced in a gnuplot readable format. The implementations of the model equations presented in this thesis were edited to produce results in a gnuplot readable format. These equations were successfully simulated on sequential and parallel processor machines (atlas, taha, altix) using the Equation Server.

The results are then sent back to the user by the equation result solver as a web page (Figure 6.2.7).

The output from the equation and gnuplot are stored in a directory for temporary storage. The contents of the directory are deleted based on the developer's specifications. Because the results are sent back to the user through the Internet, the user has the option to save the results to his or her machine. All web pages sent to the user can be edited according to the developer's specifications. This allows the developer to incorporate the Equation Solver into any page. If an equation provided by the developer needs to be updated or recompiled, it can be done without having to restart the Equation Solver. The Equation Solver also simplifies the complex task of

configuring existing web servers by reducing all configurations into one simple configuration file (Figure 6.2.8.)

The configuration file allows the developer to specify: 1) which ports the Equation Solver is running on, 2) the URL of the equation solvers, 3) the number of seconds to keep equation output results on the server, and 4) available user equations and input requirements.

Parallel computation is a powerful way to decrease computation time for solving a problem. The general idea is to break the problem under consideration into a number of sub problems. All of these sub problems are solved simultaneously, each on a different processor [2]. MPI and OpenMP are two programming libraries which allow developers to implement parallel equations on parallel machines. Systems that provide parallel computing capabilities, may require specific input from users upon each use. The following demonstrates an MPI example to run a parallel program, `FD_FFTW`, using 4 processors through command line.

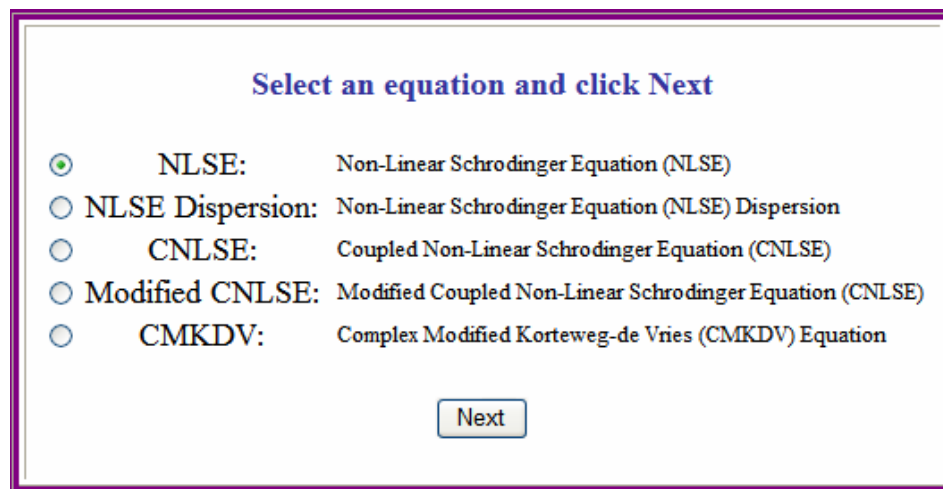
```
mpirun -np 4 ./FD_FFTW 40 163.84 0.0125
```

This required input (Ex. `mpirun -np, ./FD_FFTW`) can be redundant and often varies between systems. The Equation Solver remedies this problem by providing a developer the ability to specify only the required inputs (4, 40, 163.84, 0.0125) needed from the user. The developer can then provide the user an equation input page to obtain the input values.

Throughout the evolution and development of the Equation Server, it has been through a variety of transformations. First, it was implemented as a Java applet. Due to security restrictions of the Java applet interacting with a server and the user having to perform extra operations to ensure the full functionality of the Equation Server, it was then implemented as a web server using Tomcat and PHP. This technology could be used to implement the Equation Server as it is today. However, because of more security issues and the discretion of some

researchers not having the desire to install and configure extra servers on their machines, the Equation Server had to undergo another transformation. Finally, the Equation Server once again has the unique feature of being implemented as a web server, except without the use of existing web server technology such as Tomcat, Axis, ASP, PHP, etc... The current version of Equation Server is implemented in C++ and basic HTML, and has been tested on single and multiple processor machines. However, any machine installed with a programming language with socket and system call capabilities can implement the Equation Server.

Appendix A of this thesis contains a README file for the installation, configuration and deployment of the Equation Server.



The screenshot shows a web-based selection interface for the Equation Server. It features a title "Select an equation and click Next" in blue. Below the title, there are five radio button options, each with a label and a description. The first option, "NLSE: Non-Linear Schrodinger Equation (NLSE)", is selected. The other options are "NLSE Dispersion: Non-Linear Schrodinger Equation (NLSE) Dispersion", "CNLSE: Coupled Non-Linear Schrodinger Equation (CNLSE)", "Modified CNLSE: Modified Coupled Non-Linear Schrodinger Equation (CNLSE)", and "CMKDV: Complex Modified Korteweg-de Vries (CMKDV) Equation". At the bottom center, there is a "Next" button.

Equation	Description
<input checked="" type="radio"/> NLSE:	Non-Linear Schrodinger Equation (NLSE)
<input type="radio"/> NLSE Dispersion:	Non-Linear Schrodinger Equation (NLSE) Dispersion
<input type="radio"/> CNLSE:	Coupled Non-Linear Schrodinger Equation (CNLSE)
<input type="radio"/> Modified CNLSE:	Modified Coupled Non-Linear Schrodinger Equation (CNLSE)
<input type="radio"/> CMKDV:	Complex Modified Korteweg-de Vries (CMKDV) Equation

Next

Figure 6.2.1: Equation Server equation selection [1]

Consider the NLS equation

$$iu_t = u_{xx} + 2|u|^2 u,$$

where u is a complex-valued function.
The initial condition is

$$u(x,0) = 2\eta \exp\left\{-i\left[2x + \frac{\pi}{2}\right]\right\} \sec h(2\pi x),$$

where η = a constant.
We assume that $u(x,t)$ satisfies periodic boundary condition with period $[-P, P]$.

Select an available numerical method and click Next

- ☒ SSF1: First order split-step Fourier method
- ☐ SSF2: Second order split-step Fourier method
- ☐ SSF4: Fourth order split-step Fourier method
- ☐ PSSF1: Parallel first order split-step Fourier method
- ☐ PSSF2: Parallel second order split-step Fourier method
- ☐ PSSF4: Parallel fourth order split-step Fourier method

Figure 6.2.2: Equation Server numerical method selection [1]

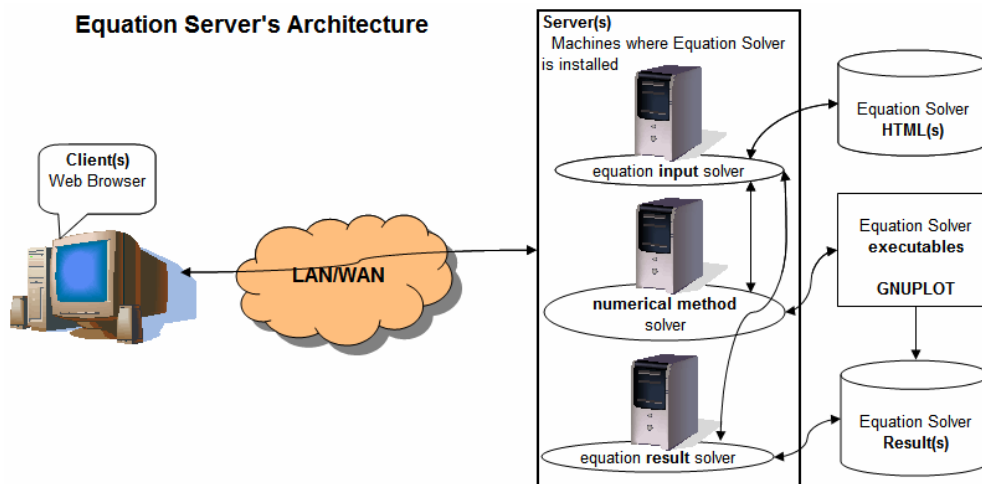


Figure 6.2.3: Equation Server architecture

Specify input values and click Next

m:	<input type="text" value="9"/>
numsteps:	<input type="text" value="8000"/>
P:	<input type="text" value="10.0"/>
z0:	<input type="text" value="0.0"/>
psi:	<input type="text" value="0.0"/>
xi:	<input type="text" value="1.0"/>
eta:	<input type="text" value="3.0"/>

Figure 6.2.4: Equation Server input specification [1]

```

I      T       $\sqrt{|k|^2}$ 
504 1.0000 0.000620
505 1.0000 0.000570
506 1.0000 0.000695
507 1.0000 0.000999
508 1.0000 0.001371
509 1.0000 0.001707
510 1.0000 0.001951
511 1.0000 0.002057

=====
#The values of the conserved quantities:
#The first one =1.200000E+01
# relative error = 6.461638E-09
#The second one =9.599497E+01
# relative error = 5.237005E-05
#*****
#The infinity norm =2.159379E-03
#The Eucleadian norm =3.089937E-03
#
#CPU time = 2.46 seconds
#

```

Figure 6.2.5: Equation Server equation result data [1]

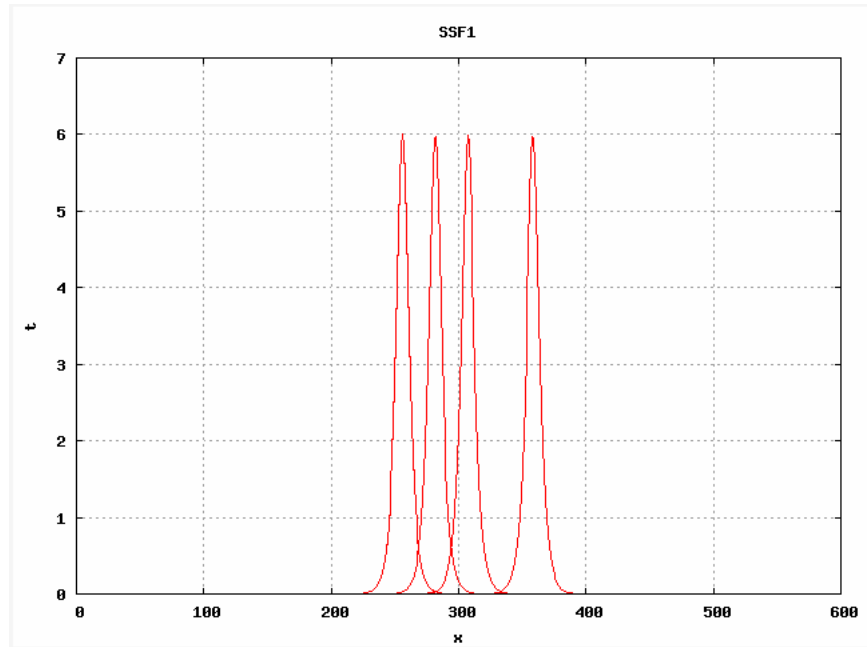


Figure 6.2.6: Equation Server gnuplot 2D plot [1]

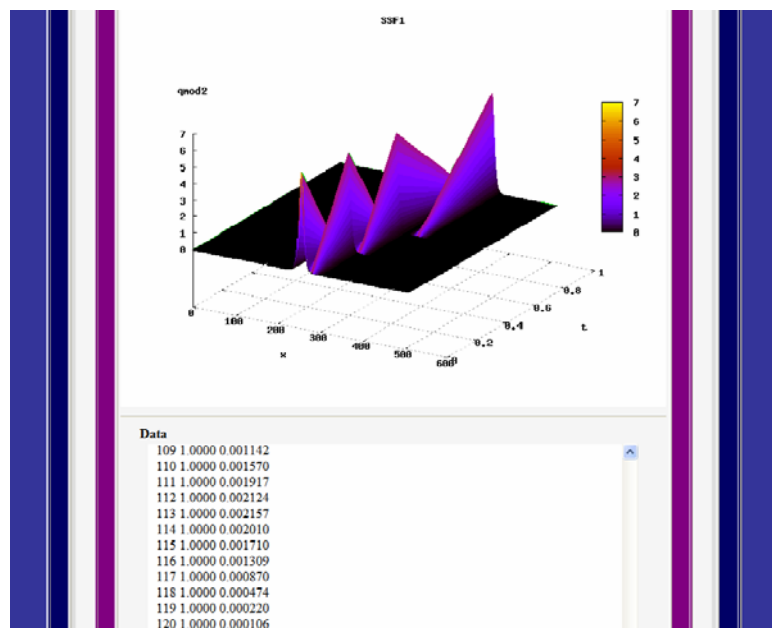


Figure 6.2.7: Equation Server web page display [1]

```

Equation Server 3.0: configuration file
!!! Note: maintain format specified !!!

[address and port] of equation input, ou
taha.cs.uga.edu 7334
taha.cs.uga.edu 7335
taha.cs.uga.edu 7336

[url] of equation input, output, and res
http://taha.cs.uga.edu:7334
http://taha.cs.uga.edu:7335
http://taha.cs.uga.edu:7336

[seconds] to remove results
30

[space] [equation name] [equation parame

SSF1
7
m integer
numsteps integer
P real
z0 real
psi real
xi real
eta real

```

Figure 6.2.8: configuration file

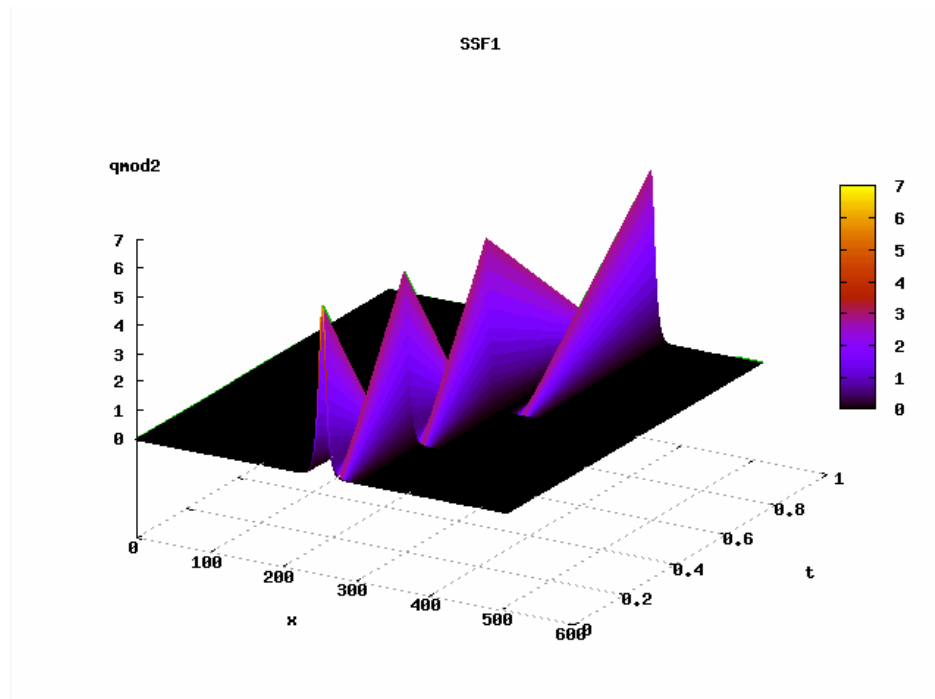


Figure 6.2.9: Equation Server gnuplot 3D plot

CHAPTER 7

VISUALIZATION AND PROFILING TOOLS

7.1 Introduction

Numerical visualization and profiling tools have been developed to assist in the analysis and interpretation of numerical data generated during the computational process. A variety of numerical visualization and profiling tools exist today such as Matlab, Maple, gnuplot, Mathematica, Jumpshot-4, Paragraph, etc.... Each tool has its advantages, disadvantages, and performs a certain function. Jumpshot-4 was chosen as the profiling tool of choice. Next, I present its purpose, capability, and how it assisted in the interpretation of parallel algorithms presented in this thesis.

7.2 Jumpshot-4

Parallel computation is a powerful way to decrease the computation time for solving a problem. The general idea is to break the problem under consideration into a number of sub problems. All of these sub problems are solved simultaneously, each on a different processor [2]. Jumpshot-4 is a Java-based visualization tool for doing postmortem performance analysis [3]. For the parallel algorithm implementations studied and simulated, the parallel FFTW-Message Passing Interface (MPI) was used. Jumpshot-4 assisted in profiling and performance analysis of these parallel algorithm implementations.

Some of Jumpshot-4's predecessors include Jumpshot-1 through 3, Nupshot and Upshot. However, Jumpshot-4 improves the portability, maintainability, and functionalities of the tools [3].

Jumpshot-4 provides users with a GUI (Figure 7.1.1) with the capability to analyze the duration of the profiled program, the various MPI operations and messages used during program execution (Figure 7.1.2), the duration of the program on each processor, and the duration of every MPI operation used on each processor (Figure 7.1.3) . It also provides zoom, timeline, histogram, and search/scan modules for ease of use (Figure 7.1.4). These capabilities provide great assistance in the optimization and analysis of the parallel implementation.

Jumpshot-4 requires an input log file (.clog or .slog2) to begin the profiling process. These files are produced after the execution of the profiled program. To produce the required *.clog files for Jumpshot-4, the FFTW-MPI equations simulated and studied were updated with the required Jumpshot-4 routines provided by the MPI Parallel Environment (MPE) library. The Jumpshot-4 routines include MPE_Start_log(), MPI_Com_rank(), MPI_Com_size(), etc... After the execution of the parallel MPI code, a *.clog file is produced. The MPE package also provides another Java utility, clog2slog2, which converts the .clog file produced from our simulations to a .slog2 file. Jumpshot-4 takes the *.slog2 file as an input file and produces a GUI for MPI analysis. If *.clog files are provided, Jumpshot-4 will prompt the user to convert them to *.slog files and proceed with the logfile converter window (Figure 7.1.5).

Each parallel algorithm for solving the CMKdV equations presented in Chapter 5 was profiled using Jumpshot-4. To produce the Jumpshot input file, a Jumpshot version of each of the parallel implementations were produced and simulated. Each numerical method was

simulated using one, two, and 4 processors on the taha 4 processor machine. Next, one parallel algorithm's implementation will be discussed.

Consider the parallel first-order scheme for solving the CMKdV equation. Jumpshot-4 was simulated for this parallel algorithm on the taha 4 processor machine using 1, 2 and 4 processors (Figures: 7.1.6, 7.1.7, 7.1.8 respectively). Immediately, from the interface, one can determine the number of processors used in the simulation base on the number of rows specified and displayed. Typically, the root processor is designated as having the processor id 0 (SLOG-2 0). From the figures presented, 7.1.7 and 7.1.8, the current method used in communication can also be determined based on its Jumpshot specified color (Green: MPI_Send, Blue: MPI_Receive). Such visualization assists in analysis of the communication between the processors. The arrows in the figures represent the messages passed back and forth between the connected processors. Also from the simulation, one can see that the duration of time needed to send messages to and from the root processor has been reduced. Note that the figures presented, do not represent complete simulation.

The Jumpshot-4 profiling results, for the parallel algorithms covered in this thesis, were generated on the taha 4 processor machine. Jumpshot-4 is currently distributed with the MPE software package. The MPE package provides users with: 1) a set of profiling libraries to collect information about the behavior of MPI programs, 2) various viewers for the logfiles, 3) a shared-display parallel X graphics library, 4) a profiling wrapper generator for MPI interface, 5) routines for sequentializing a section of code being executed in parallel, and 6) debugger setup routines [3]. Currently, the MPE software package and documentation are freely available on the Internet (www-unix.mcs.anl.gov/perfvis).



Figure 7.1.1 Jumpshot-4 GUI [3]

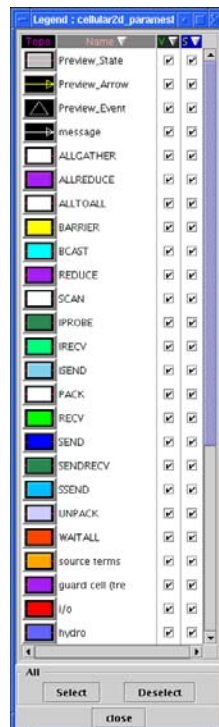


Figure 7.1.2 Jumpshot-4 Legend window [3]

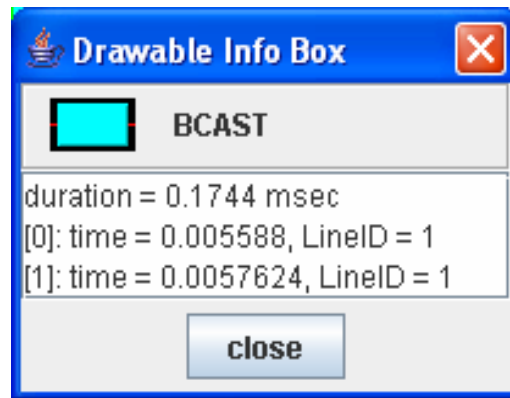


Figure 7.1.3 Jumpshot-4 MPI operation duration window [3]

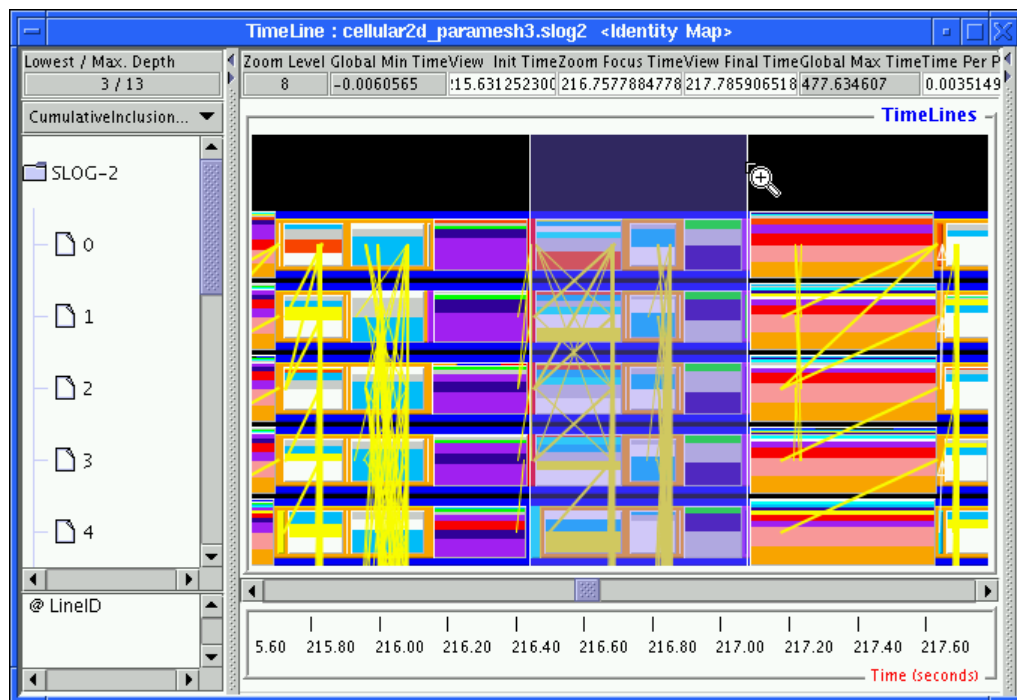


Figure 7.1.4 Jumpshot-4 Timeline window [3]

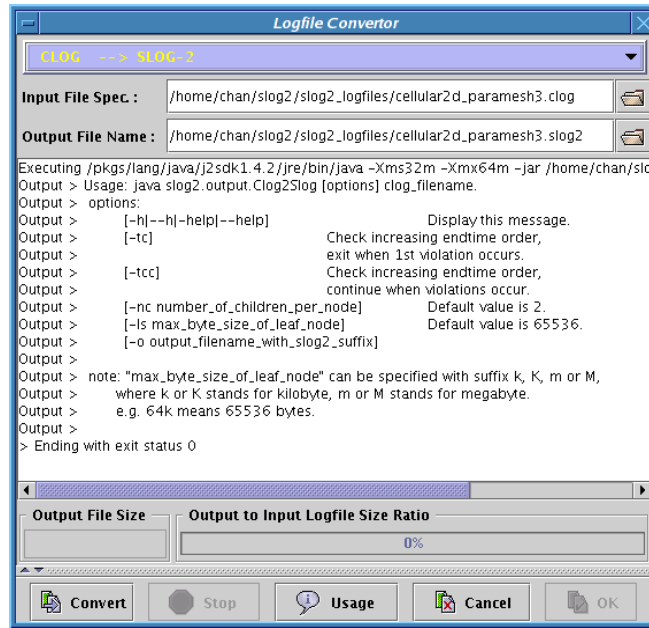


Figure 7.1.5 Jumpshot-4 Logfile Converter window [3]

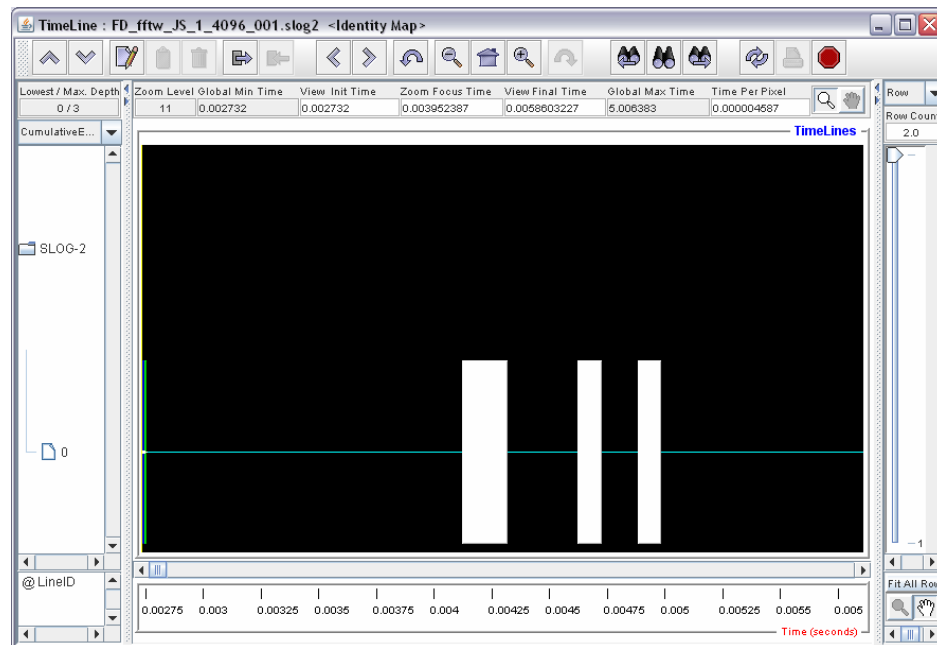


Figure 7.1.6 Jumpshot-4 parallel finite difference (FD) scheme for CMKdV equation on 1 processor (N=4096)

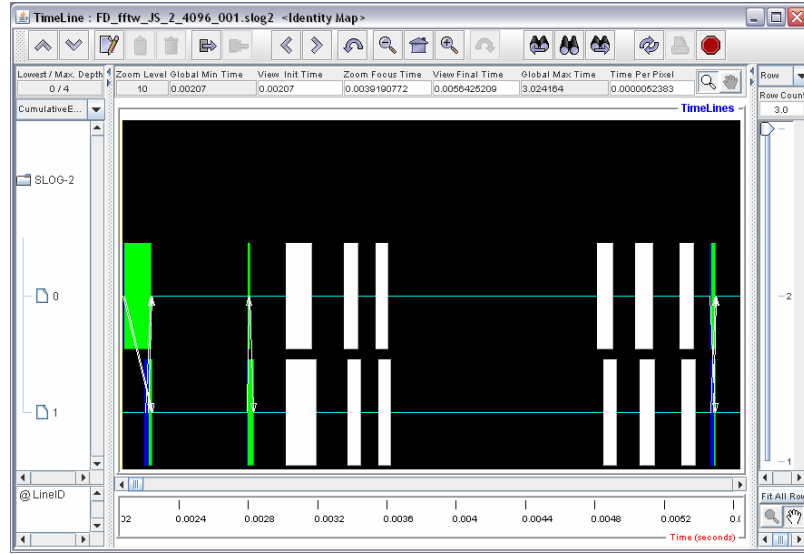


Figure 7.1.7 Jumpshot-4 parallel finite difference (FD) scheme for CMKdV equation on 2 processors (N=4096)

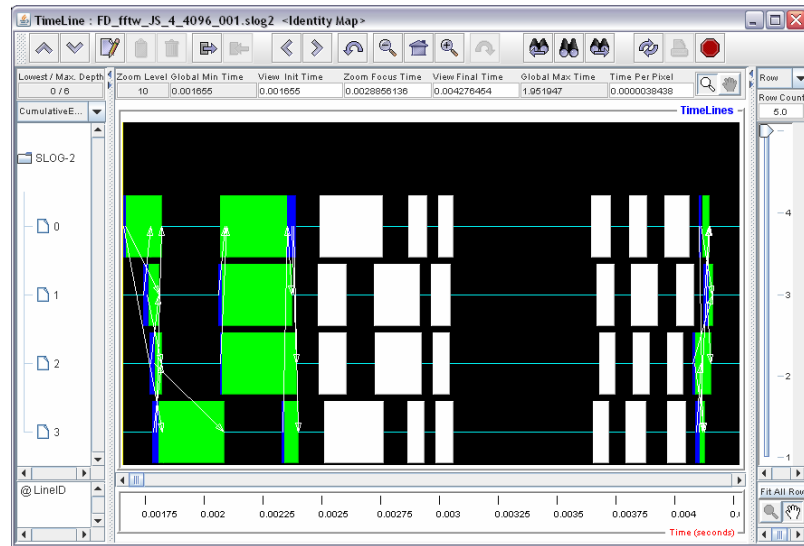


Figure 7.1.8 Jumpshot-4 parallel finite difference (FD) scheme for CMKdV equation on 4 processors (N = 4096)

CHAPTER 8

CONCLUSIONS

In this thesis, we presented various methods used for the numerical simulation of nonlinear evolutionary equations. These equations include: the Nonlinear Schrödinger (NLS) equation, the Dispersed Nonlinear Schrödinger (Dispersed NLS) equation, the Coupled Nonlinear Schrödinger (CNLS) equation, the Modified Coupled Nonlinear Schrödinger (Modified CNLS) equation, and the Complex Modified Korteweg-de Vries (CMKdV) equation. The various numerical methods used to solve them include the sequential and parallel first-order, second-order, fourth-order split-step Fourier method, local inverse scattering transform approximation, and finite difference approximation.

Also, I introduced and presented Equation Server, a web based graphical user interface for the simulation of these nonlinear evolutionary equations. A gnuplot plot for the numerical methods used for solving the CMKdV equation were produced and displayed. Performance results were also generated, analyzed and presented.

In addition, the Jumpshot-4 visualization and profiling tool was presented which assisted in the interpretation and analysis of processor communication during simulation of the parallel algorithms used to solve the CMKdV equation.

BIBLIOGRAPHY

- [1] Foster, R. N., 2007. Equation Server, Core Competency Certification (MS), The University of Georgia.
- [2] Liu, R., 2001. Numerical and Parallel Algorithms for the CMKDV Equation, Thesis (MS), The University of Georgia.
- [3] Chan, A., Gropp, B., Lusk, R., Performance Visualization for Parallel Programs [online]
Available from: <http://www-unix.mcs.anl.gov/perfvis/>, [Accessed 26 March 2007]
- [4] FFTW - FFTW Home Page, [online] Available from: <http://www.fftw.org>, [Accessed 29 March 2007]
- [5] Taha, T. R., Numerical simulations of the CMKdV equation, *Mathematics and Computers in Simulation*, 37 (1994), 461-467.
- [6] Xu, X., Taha, T. R., Parallel Split-Step Fourier Methods for Nonlinear Schrödinger-Type Equations, *Journal of Mathematical Modeling and Algorithms*, 2: 185-201, 2003.

- [7] Thiab R. Taha, "Numerical Simulations of the Complex Modified Korteweg-de Vries Equation", Special issue, "Solitons, Nonlinear Wave Equations and Computation" Mathematics and Computers in Simulation, 37 (1994) 461-467.
- [8] Taha, T. R., Xu, X., Parallel Split-Step Fourier Methods for the Coupled Nonlinear Schrödinger Type Equations, *The Journal of Supercomputing*, 32. 5-23, 2005.

APPENDIX A

EQUATION SERVER README

This appendix contains the README file for the installation, configuration and deployment of the Equation Server. Deviations from the README file should be avoided.

README

- I. Contents
- II. Directions
 - A. Setup
 - 1. The Source
 - 2. The Configuration
 - 3. The HTML
 - 4. The Equations
 - 5. The Makefile
 - 6. The Browser
 - B. Compilation
 - C. Execution
 - D. Termination

E. Install Checklist

III. Requirements

I. Contents

A. src	// directory
1. equationinputsolver.cpp	// source code
2. equationoutputsolver.cpp	// source code
3. equationresultsolver.cpp	// source code
4. MyConnection.cpp	// source code
5. MyConnection.h	// source code
6. ESequation.cpp	// source code
7. ESequation.h	// source code
8. ESgnuplot.cpp	// source code
9. ESgnuplot.h	// source code
B. equations	// directory
C. html	// directory
1. index.html	// html file
2. not_found.html	// html file
3. bad_request.html	// html file
4. http_version_not_supported.html	// html file
D. results	// directory
E. README.txt	// README file
F. Makefile	// Makefile

- G. ESequation.config // configuration file
- H. ESresult.html // result html file
- I. install // install checklist file

II. Directions

A. Setup

1. The Source - ./src/*

Equation Server 2.0+ no longer requires the developer to edit the source code. The configuration file (./ESequation.config) was created to specify what is needed by the source code.

2. The Configuration - ./ESequation.config

In Equation Server 2.0+, the configuration file allows the Equation Solver to be configured easily without editing the source code. In the configuration file, the following is specified:

- The address and port of equation input, output, and result solvers:

<address port>

- The url of the equation input, output, and result solvers:

<url>

- The number of seconds to remove the results:

<seconds>

- The equation name, number of variables, variable names, and variable type.

<space>

<equation name>

```

<N# of variables>

<variable1 name and type>

<variable2 name and type>

...

<variableN name and type>

```

Note: Follow the format specified in the configuration file. Variable types include: real and integer.

3. The HTML - ./html/index.html

The html file (./html/index.html), where the user selects an equation, should specify the equations available to the user and forward the user's equation selection to the equation input solver's address and port:

```

<form method="get" action="http://equation input solver address:port">

<select name="Equation">

<option value="EQUATION1" />EQUATION1

<option value="EQUATION2" />EQUATION2

</select>

```

4. The Equations

Executable equations (EQUATION.exe), which are to be made available to users, should be placed in the equation's directory and compiled. The equation's specific html file (EQUATION.html) and gnuplot compatible file (EQUATION.deguignu) should also be placed in the equation's directory.

Follow the directory and file format specified:

```

./equations/EQUATION/EQUATION.exe

```



```
./equations/EQUATION/EQUATION.html
```

```
./equations/EQUATION/EQUATION.deguignu
```

The html file will be displayed when the user selects the equation. The input from the user should be forwarded to the equation output solver's address and port in the same order as specified in the configuration file.

```
<form method="get" action="http://equation output solver address:port">
```

```
<input type="hidden" name="Equation" value="EQUATION" />
```

```
<input type="text" name="ARG1" size="10" />
```

```
<input type="text" name="ARG2" size="10" />
```

```
...
```

```
<input type="text" name="ARGN" size="10" />
```

The executable equation should be able to print its results to the screen based on the command line input arguments.

Equation Server 3.0 allows the developer to edit the gnuplot file's (EQUATION.deguignu) contents without editing the Equation Server's source code. Because the executable equation's results are dynamic, the developer must follow the example format explained below:

Example gnuplot file (EQUATION.deguignu):

```
set nokey
```

```
set title 'Equation'
```

```
set ylabel 'y Axis'
```

```
set xlabel 'x Axis'
```

```
set terminal png
```

```
set output '[gnuplot image file]'  
  
plot \  
  
'[gnuplot index data file]' using 1:3 with linespoints  
  
quit
```

Note: The Equation Solver will dynamically fill in what is needed for the [gnuplot image file] and [gnuplot index data file] sections. Without these two labels, a gnuplot plot of the executable equation's results would not be generated; They should not be edited by the developer.

The gnuplot file will be used when the equation is executed.

The printed results should be readable by gnuplot, which produces a plot of the equation's results.

All output from the executable equations and gnuplot are placed in the results directory. The output is deleted after a period of time, which is specified in the configuration file (II.A.2).

If an executable equation needs to be recompiled, it can be done so without recompiling the Equation Server.

5. The Makefile - ./Makefile

After the Equation Solver has been configured, the Makefile is used to easily compile (II.B) and execute (II.C) the Equation Server.

The Makefile may need to be changed to specify the C++ compiler and libraries on your system.

6. The Browser

After executing the Equation Server source code (II.C), with a web browser, browse to the html file (II.A.3) where the user can select which equations to use:

`http://<input solver address:port>`

Note: The Equation Server was designed so that it could be also incorporated into existing web pages. Doing so may require extra web design knowledge and/or experience.

The html file mentioned above can be bypassed if the equation is known. The equation's html file (`./equations/EQUATION/EQUATION.html`) can be accessed by pointing your web browser to:

`http://<equation input solver address:port>/?Equation=<EQUATION>`

The equation's html file mentioned above can also be bypassed if the equation, its arguments, and its argument values are known. The result html can be accessed by pointing your web browser to:

`http://<equation output solver
address:port>/?Equation=<EQUATION>&<ARG1>=<VAL1>&<ARG2>
=<VAL2>...&<ARGN>=<VALN>`

Equation Server 3.0 allows the developer to edit the result html file's (`ESresult.html`) contents without editing the Equation Server's source code. This gives the developer the ability to edit the result html file like a webpage. Because the executable equation's results are dynamic, therefore making the result html page dynamic, developers must follow the example format explained below:

Throughout the original result html file (ESresult.html), the following sections are specified:

1. [equation server home url] - url of the equation input solver, which is specified in the configuration file (II.A.2).
2. [equation server equation] - name of the equation, which is specified in the configuration file (II.A.2).
3. [equation server input] - input obtained from the user for the executable equation (II.A.4).
4. [equation server result image] - plot of the result data generate by gnuplot (II.A.4).
5. [equation server result data] - data generated by the executable equation (II.A.4).

The Equation Solver will dynamically fill in what is needed for these sections. The use and location of these sections can be changed by the developer.

Note: Without the [equation server input], [equation server result image], and [equation server result data] sections, the user input, gnuplot plot of the executable equation's results and the result data will not be visible to the user; They should not be edited by the developer.

The result html file is the final webpage displayed to the user through a web browser.

B. The Compilation

The Equation Server can be compiled with the Makefile using:

```
% make
```

Executables can be removed by using:

```
% make clean
```

If compilation is unsuccessful, the Makefile will need to be edited. Make sure the correct c++ compiler and libraries are specified for your system.

C. Execution

The Equation Server can be executed with the Makefile using:

```
% make run
```

When the Equation Server is executed, three solvers are run: the equationinputsolver, equationoutputsolver, and equationresultsolver. These solvers bind to three different ports. If any of these three ports are occupied, a new port will need to be specified in the configuration file (II.A.2). Perform steps II.D., II.A., II.B., and II.C.

D. Termination

To terminate the Equation Server, you will need to end each process created by the execution of the three solvers (C):

1. equationinputsolver
2. equationoutputsolver
3. equationresultsolver

You can view your processes by using:

```
%ps -ef | grep <developer user name>
```

-or-

```
%ps -u <developer user name>
```

Once you find the process ids of the processes you wish to terminate, you can stop them using:

```
%kill -9 <process id ... process id>
```

If you wish to restart the Equation Server after terminating it, wait a few seconds for the solvers to release the ports previously used.

E. Install Checklist

To assist in installing the Equation Server, an install checklist file (install) was created. The install checklist can be started by making it executable and using:

```
%install
```

The README file should be consulted for further assistance.

III. Requirements

1. C++ compiler with a socket library.
2. gnuplot 4.0