UCLA
COMPUTATIONAL AND APPLIED MATHEMATICS

Comparison of Finite Difference and the Pseudo-Spectral Approximations for Hyperbolic Equations and Implementation Analysis on Parallel Computer CM-2

Yu-Chung Chang

January 1992

CAM Report 92-02

Department of Mathematics
University of California, Los Angeles
Los Angeles, CA. 90024-1555

UNIVERSITY OF CALIFORNIA

Los Angeles

Comparison of Finite Difference and the Pseudo-Spectral Approximations for Hyperbolic Equations and Implementation Analysis on Parallel Computer CM-2

A dissertation submitted in partial satisfaction of the requirements for the degree Doctor of Philosophy in Mathematics

by

Yu-Chung Chang

1991

TABLE OF CONTENTS

ACKNOWLEDGEMENTS

My deepest gratitude goes to Prof. H.-O. Kreiss for his excellent guidance, his encouragement and his support, without which this work would not have been possible. His invaluable insight and overview on sciences have always inspired my work and led me to a deeper appreciation of the beauty of mathematics.

I also would like to express my deep appreciation to Prof. Tony Chan for his guidance and and his support, especially for the work on parallel computations. His enthusiasm about research has been a constant energy for my work.

Tom Y. Hou deserves my special acknowledgement. His spiritual support has accompanied with me to the very last stage of graduate study. His insight into mathematics has made complicated problems become comprehensible.

Thanks also go to Profs. C. Anderdon, R. Caflisch, A. Chang, S.Y. Cheng, B. Engquist, J. Garnett, C. Lange, K. C. Li, and S. Osher for their helps and encouragement through the years of my graduate study at UCLA.

I wish to thank Drs. M. Bromley, S. Duggirala, A. Greenburg and K. Mathur from the Thinking Machine Corporation and F. Hedman at K.T.H. for many valuable discussions on the machine features and architectures of the connection machine CM-2.

It deserves to mention my fellow students: June Donato, Erding Luo, Tachun Wang, Tien-Lun Tsoong, and Lixin Wu, and Sunny Wu whose kind and happy characters have made my graduate study more enjoyable.

I especially thank my parents for their love, their understanding and financial support throughout the years of my graduate study at UCLA.

Through my graduate study, I was generously supported by some grants, which include the Office of Naval Research under contracts N-00014-83-K-0422 and N00014-90-J-1695, the Department of Energy under contract DE-FG03-87ER25037, the National Science Foundation under contracts DMS-8312264, ASC 9003002 and BBS-87-14206, the Army Research Office under contract DAAL03-91-G-0150.

VITA

October 26, 1958	Born, Taipei, Taiwan, R.O.C.
1981	B.A., Mathematics Fu-Jen University Taipei, Taiwan, R.O.C.
1986	M. S., Mathematics UCLA
1985-1988	Teaching Assistant UCLA
1988-1991	Research Assistant UCLA

ABSTRACT OF THE DISSERTATION

Comparison of Finite Difference and the Pseudo-Spectral

Approximations for Hyperbolic Equations and

Implementation Analysis on Parallel Computer CM-2

by

Yu-Chung Chang

Doctor of Philosophy in Mathematics

University of California, Los Angeles, 1991

Professor Heinz Kreiss and Tony Chan, Chair

A detailed comparison between symmetric high order finite difference approximations and the pseudo spectral approximation is given for both linear and nonlinear hyperbolic equations. The problem of interest is the question which method is more efficient in achieving a given error tolerance. The understanding of this question could help to improve the practical performance of these methods, especially for problems where the physical solution is nearly singular.

We divide the thesis into two parts. The first part is concerned with comparison between finite difference approximations and the pseudo spectral approximation for hyperbolic equations. Emphasis is put on the error analysis, direct comparson of performance of these methods for some model problems on sequential machines.

The second part is devoted to studying the performance of high order difference methods on a parallel connection machine, CM-2. A timing model is established to analyze the performance of high order difference methods by estimating the arithmetic operation time and the communication time on a SIMD kind of machine.

In practice, the success of a spectral computation depends on whether or not the high frequency components of the physical solution can be well represented by the numerical Fourier modes. This criteria becomes difficult to fulfil when the solution contains many small scale features. Examples of this kind are computations for turbulent flow and flow of strong shear. Typically, small scale features of the physical solution are generated as time evolves and it is very difficult to accurately resolve them on computational grid due to the limited computer power. Then the natural question is what is the minimal numerical resolution needed in order to achieve a given error tolerance. This is a question which the traditional numerical analysis has difficulty to answer. One of our major findings is that the fourth order centered difference method is superior to the pseudo spectral method in the sense that the difference method could achieve the same accuracy more efficiently than the spectral method in the case where the solution is nearly singular. Extensive numerical experiments and some partial analysis are given for a 1-D model problems to support this conclusion. A 2-D nonlinear viscous Burger system is also considered. Our theoretical and computational results indicate that the 4th or 6th difference method could achieve a given error tolerance more efficiently than spectral methods.

It is very important to compare the performances of these high order methods in a massively parallel super computer since most of the large scale simulations are carried out by super computers. It is also crucial to investigate how the machine architecture may affect the performance of an individual numerical method. For this purpose, we have performed many experiments on the connection machine. The question is still the same: are the high order difference schemes still more efficient than the spectral method on the connection machine? In our experiments, we have taken into account the existing architecture for FFT and the interprocessor communication in the connection machine. Unlike sequential machines in which the operation count dominates the computing time, connection machines spend a lot of time in communication among different processors. To obtain a better understanding in the communication process, we have designed a timing model to analyze the relationship between the inter-processor communication time and the order of the difference scheme. We then use the timing model to predict the performance of the difference methods by varying different machine performance parameters, such as the communication time, the floating point operation time and the machine overhead time. We observe the performance of these difference methods are qualitatively insensitive to the perturbation of these performance parameters. From our analysis and numerical implementations, we have observed that the fourth order finite difference method is still the best. In general the high order difference methods are more preferable than the spectral method in sense that they achieve the given error tolerance more efficiently.

Part I. Comparison of Finite Difference and the Pseudo Spectral Approximations for Hyperbolic Equations

CHAPTER 1

Introduction to Part I.

It is well known that the pseudo spectral method has spectral accuracy and it can be implemented efficiently by use of the Fast Fourier Transform (FFT). For this reason, it has been widely used in industrial and engineering applications (see references in Gottlieb and Orszag [5]). In practice, the success of a spectral computation depends on whether or not the high frequency components of the physical solution can be well represented by the numerical Fourier modes. Whenever we can afford to accurately resolve the small scales in the physical solution, the pseudo-spectral method gives a very accurate approximation (see, e.g., [10, 5, 3, 18]). This criteria becomes difficult to fulfil when the solution contains many small scale features. Examples of this kind are computations for turbulent flow and flow of strong shear. Typically, small scale features of the physical solution are generated as time evolves and it is very difficult to accurately resolve them on computational grid due to the limited computer power. Then the natural question is what is the minimal numerical resolution needed in order to achieve a given error tolerance. This is a question which the traditional numerical analysis has difficulty to answer. Also, the global coupling nature of the pseudo spectral method tends to introduce the numerical oscillations everywhere when the high frequency components are under

resolved. Finite difference methods, on the other hand, have a more local coupling of stencil. Thus it is reasonable to speculate that high order finite difference method may have a chance to out perform the spectral method since it has the advantages of being high order accurate and localized. Indeed, our study shows that the sixth order difference method produces less numerical oscillations and is more efficient than the spectral method in the case where the solution is nearly singular.

Another advantage of using finite difference methods is that the operation count is smaller than that of the spectral method. In the 1-D pseudo spectral method, it usually takes $(5NlogN - 4N)$ operations for the convection term calculation per time step by using FFT (N is the number of grid points). In comparison, a sixth order finite difference method takes only $9N$ operations for the convection term calculation per time step. The difference of these two operation counts becomes more noticeable for large N. In our study, we observe that when the solution becomes nearly singular the accuracy of a spectral calculation with N grid points is comparable to that of a sixth order difference method with $2N$ grid points. Therefore, it would be more efficient to use a sixth order finite difference method in this case. This is also confirmed by our direct comparisons of the CPU times needed for these calculations.

Two model problems are considered in this paper. One is a linear hyperbolic equation with variable coefficient. When the coefficient changes signs from a positive to a negative one, it is well known that the solution develops an internal

layer whose thickness decreases to zero exponentially in time. Thus it becomes more and more difficult to accurately resolve this internal layer for large times. To study convergence of the spectral method and finite difference methods, we refine our numerical computations successively until the internal layer is well represented by the computational grid. The interesting question is to see which method can achieve the given accuracy uniformly with the least computation effort. We observe that when the solution is smooth (true for short times or for one sign velocity coefficient), the spectral method out performs the second, fourth and sixth order finite difference methods. However, when the solution becomes more singular for large times, the spectral method tends to introduce more numerical oscillations which spread over the entire domain. The fourth or sixth order finite difference methods seem to do a better job. The numerical oscillations are confined to a small region containing the internal layer. As the number of grid points increases, the amplitude of the numerical oscillations for both the difference methods and the spectral method decay to zero. There will be eventually a time when we are no longer able to resolve the solution numerically. Beyond that time, all numerical methods considered here fail to converge.

It is interesting to note that there appears to be no noticeable numerical instability in our pseudo spectral computations for the variable coefficient which changes sign. It has been a long open question as for whether or not the pseudo spectral method could develop numerical instability [6, 4]. Our numerical study tends to indicate that the pseudo spectral method converges to the correct solution

as the mesh size goes to zero. The numerical oscillations are due to the lack of numerical resolution, as is the case for the finite methods.

The second problem we consider is the 2-D viscous Burger system. In the inviscid case, the Burger's equation will develop a shock discontinuity at later times even if we start with smooth initial conditions. In the presence of viscosity, the shock is smoothed out. But there is still a viscous shock profile whose thickness is proportional to the viscosity. We ask the same questions for this 2-D viscous Burger system as before. Our numerical result indicates that before the time when a shock forms the observations we made for the 1-D model problem still hold for the 2-D problem. The observations for the 1-D case continues to hold even after the time when a shock forms. In this case, we need to have $O(\frac{1}{\epsilon})$ grid points to resolve the shock profile. Since the thickness of the shock profile remains to be of order $O(\frac{1}{\epsilon})$ for all times, the number of grid points needed for achieving a given error tolerance does not increase with time. This is confirmed by our numerical experiments. A sixth order central difference calculation with $N = 256$ approximately achieves the given error tolerance 0.01 at $t = 1.5$ in the case when viscosity equals to 0.015. The same calculation still achieves the error tolerance 0.01 at $t = 2.0$ and $t = 2.5$. This observation is also early confirmed by the 1-D calculation with viscosity 0.015 and $N = 128$ in the case $\alpha = 1.2$ and $\beta = 0.1$. In future, I plan to perform a similar test for random initial data. The goal is to see how much of the observation we made here for the smooth data applies to the random data. This may shed some light into computations of high Mach number

flow. (see, e.g. [11, 12]).

It is difficult to calculate the numerical errors because there is no explicit solution available for these equations. In the 1-D calculations, we use a pseudo spectral calculation with 2048 grid points as our "accurate" solution. Then we compare the coarser grid solutions with this accurate solution. This provides us with an error table. When the solution is smooth, the error decays at the rate predicted by the order of the method. As time increases, the error deteriorates rapidly. As a consequence, the rate of convergence is lost very fast. It is not clear a priori if this is due to the loss of accuracy in our numerical "exact solution". An independent check is required to verify the numerical errors.

Two approaches are proposed to verify our numerical errors. In order to perform an asymptotic error analysis, we first derive a partial differential equation for the leading order error . Although this error equation can be derived in a formal manner, it is not obvious that this equation indeed characterizes the leading order error. We show that this is the case by using an argument due to Richardson. The idea is to compare the numerical solution with a smooth error expansion of the exact solution. The advantage of this argument is that we not only obtain a convergence proof for the numerical method, we also establish the precise error expansion for the numerical solution. Once the error equation is obtained, what is left to do is to estimate the growth rate of the error constant. Since the error equation contains high order derivatives of the exact solution as a forcing term, it is essential to obtain a priori estimate for the derivatives of the exact solution.

One way to achieve this is to replace the original equation by a simpler one which can be solved analytically. This is our first approach. This approach works well for moderate times. But for high order methods (like fourth or sixth order) better approximations for the velocity coefficient are required to obtain an accurate approximation for the high order derivatives. It is not difficult to imagine that this is a very difficult task in general since the high order derivatives become even more singular when the solution itself becomes nearly singular.

The second approach is to derive a first order system to approximate the high order derivatives. Then try to approximate these equations together with the error equation numerically. We use a fourth order finite difference method and a particle method respectively to approximate this system. The result agrees very well with our numerical errors. For large times, the finite difference approximation converges slowly due to the increasing singular nature of the solution. The particle method gives a better approximation in this case since the method is more self adaptive.

Part I of this thesis is organized as follows. In Chapter 2, we describe the 1-D linear hyperbolic model problem. The finite difference and the pseudo spectral approximations for this model are introduced. Convergence properties of these methods and their numerical properties are discussed. In Chapter 3, we present detailed numerical computations for the 1-D model using these two types of methods. Some observations on the efficiency of different methods are given. Chapter 4 gives error analysis for the numerical results presented in Chapter 3. This includes the asymptotic error analysis and the direct approximation for the error equations.

The 2-D viscous Berger system is studied in Chapter 5. Many of our observations for the 1-D model still apply here.

CHAPTER 2

The 1-D Linear Hyperbolic Equation.

In this chapter, we introduce the finite differences and the pseudo-spectral method for a one-dimensional hyperbolic equation. This model equation is chosen so that it mimics the singular shear layer structures in the two-dimensional incompressible flow.

We consider the following one dimensional linear hyperbolic equation using periodic boundary condition,

$$u_t + a(x)u_x = 0 \tag{2.1}$$

$$u(x,0) = sin(x). \tag{2.2}$$

When $a(x)$ changes sign from a positive one to a negative one, the solution of the above equation experiences a linear compression. As a consequence, the solution produces an internal layer whose thickness decays to zero exponentially in time. In our study, we choose $a(x) = 1 - \alpha \exp(\frac{-1}{\beta}(\sin^2(\frac{x-\pi}{2}))$, with parameters α and β to be determined later.

The case where $\alpha = 1.2$ and $\beta = 0.1$ is especially interesting, since it corresponds to the case the compression is strong and localized near the region $a(x) = 0$ and changes sign from positive to negative. The cases of $\alpha = 1$ and $\alpha = 0.8$ are

also considered. The case of $\alpha = 1$ corresponds to the case that $a(x)$ vanishes only at a single point. The case of $\alpha = 0.8$ corresponds to the case when $a(x) \geq 0.2 > 0$. The solution is smooth for large times.

We consider two classes of numerical methods for this equation. The first is the pseudo spectral method. The second is the high order centered finite difference approximation. We test the second order, the fourth order and the sixth order finite difference methods respectively. Then we compare the numerical results obtained by the difference approximations to those obtained by the spectral method.

In order to describe the basic ideas we consider discretization in space only, this approach is often called the *method of lines*. It yields a set of ordinary equations for the grid values which can be solved numerically by a standard ODE-solver.

The second order centered difference scheme is simply

$$(V_j)_t + a(x_j) D_2^h(V_j) = 0 \tag{2.3}$$

$$V_j(0) = \sin(x_j),$$

where $x_j = jh$ and D_2^h is a centered differencing operator

$$D_2^h(V_j) = (V_{j+1} - V_{j-1})/(2h).$$

Similarly, the fourth order or the sixth orde difference schemes are given respectively by

$$(V_j)_t + a(x_j) D_4^h(V_j) = 0 \tag{2.4}$$

$$V_j(0) = \sin(x_j),$$

and

$$(V_j)_t + a(x_j)D_6^h(V_j) = 0 \tag{2.5}$$

$$V_j(0) = \sin(x_j),$$

where D_4^h and D_6^h are defined by

$$\begin{aligned} D_4^h(V_j) &= 4/3D_2^h(V_j) - 1/3D_2^{2h}(V_j) \\ D_6^h(V_j) &= 16/15(4/3D_2^h(V_j) - 1/3D_2^{2h}(V_j)) \\ &- 1/15(4/3D_2^{2h}(V_j) - 1/3D_2^{4h}(V_j)). \end{aligned}$$

The pseudo spectral method is based on the discrete Fourier transform and the related discrete Fourier series.

For any integer $N > 0$, consider the set of points

$$x_j = \frac{2\pi j}{N} \qquad j = 0, ..., N-1$$

referred to as nodes or grid points or knots. The *discrete Fourier coefficient* of a complex-valued function u in $[2\pi]$ with respect to these points are

$$\tilde{u}_k = \frac{1}{N}\sum_{j=0}^{N-1} u(x_j)e^{-ikx_j} \qquad -N/2 \le k \le N/2 - 1. \tag{2.6}$$

By the orthogonality relation

$$\frac{1}{N}\sum_{j=0}^{N-1} e^{ipx_j} = \begin{cases} 1 & \text{if } p = Nm, m = 0, \pm 1, \pm 2, ... \\ 0 & \text{otherwise,} \end{cases}$$

we have the inversion formula

$$u(x_j) = \sum_{k=-N/2}^{N/2-1} \tilde{u}_k e^{ikx_j} \qquad j = 0, ..., N-1. \tag{2.7}$$

We can show that the polynomial

$$Int_N u(x) = \frac{1}{2\pi} \sum_{k=-N/2}^{N/2} \tilde{u}(k) e^{ikx},$$

is the unique trignometric interpolant of u, i.e.

$$u_j = Int_N u(x_j), \qquad j = 0, 1, 2, ..., N.$$

This polynomial is also known as the discrete Fourier series of u.

The process of representing the sequence $u(x_j)_0^{N-1}$ using the Fourier coefficients $\tilde{u}(k)_{-N/2}^{N/2-1}$, defined by (2.6) is called the *discrete Fourier transform* (DFT). The two conventional forms are given by (2.6) and (2.7), with the latter sometimes referred to as the inverse DFT . They show that the discrete Fourier transform is an orthogonal transformation in C^N . It can be accomplished by the Fast Fourier Transform algorithm (FFT) (see Cooley and Tukey [1]).

The simplest Fast Fourier Fourier Transform requires N to be a power of 2. If the data are fully complex it requires $5Nlog_2N - 6N$ real operations, where the addition and multiplication are counted as separate operations. In most applications, u is real and $\tilde{u}_{-k} = \bar{\tilde{u}}_k$. If the data are real, the operations will be halved.

To compute approximations of $\frac{du}{dx}$ at the grid points x_j, we first calculate the trigonometric interpolant and then evaluating the derivative of this interpolant

in the points x_j. Denote by $w(x)$ the derivative of the interpolation polynomial $Int_N u(x)$, i.e.

$$w(x) = \frac{d}{dx} Int_N(u(x_j)) = \frac{1}{\sqrt{2\pi}} \sum_{k=-N/2}^{N/2-1} ik\tilde{u}(k)e^{ikx}.$$

The pseudo spectral scheme is given by

$$(V_j)_t + a(x_j)S(V_j) = 0$$

where $S(u(x)) = \frac{d}{dx} Int_N(u(x_j))$.

2.1 Convergence properties of these schemes

It is not difficult to prove convergence of those centered difference methods due to their antisymmetriness of the discretization (see,e.g., [18]). But what we really want to know is a precise asymptotic error expansion instead of a crude upper bound for the error (like in most of the convergence proofs). We will use the techniques of Richardson's [16, 17] to derive partial differential equations for the leading error constants.

2.2 Leading error constants:

Here, we again consider discretization in space only.

Substituting the true solution into the finite difference schemes (2.3), (2.4) and (2.5) , we get the formulas for the local truncation errors of the second, the fourth and the sixth order schemes respectively.

$$(u_j)_t + a_j D_2^h(u_j) \quad = \quad (u_j)_t + a(x_j)u_x(x_j) + a(x_j)\frac{u_{xxx}(x_j)}{6}h^2 + O(h^4)$$

$$= a(x_j)\frac{u_{xxx}(x_j)}{6}h^2 + O(h^4) \tag{2.8}$$

$$(u_j)_t + a_j D_4^h(u_j) = (u_j)_t + a(x_j)u_x(x_j) - a(x_j)\frac{u_{xxxxx}(x_j)}{30}h^4 + O(h^6)$$

$$= -a(x_j)\frac{u_{xxxxx}(x_j)}{30}h^4 + O(h^6) \tag{2.9}$$

$$(u_j)_t + a_j D_6^h(u_j) = (u_j)_t + a(x_j)u_x(x_j) + a(x_j)\frac{4u_{xxxxxxx}(x_j)}{315}h^6 + O(h^8)$$

$$= a(x_j)\frac{4u_{xxxxxxx}(x_j)}{315}h^6 + O(h^8). \tag{2.10}$$

Using energy estimate we can show that $V_j - u_j = O(h^2)$. Let $\xi_{2j}h^2 = V_j - u_j$, where $u_j = u(t, x_j)$. Subtracting (2.8) from (2.3), we get

$$(\xi_{2j})_t h^2 + a(x_j)D_2^h\xi_{2j}h^2 - a(x_j)\frac{u_{xxx}(x_j)}{6}h^2 + O(h^4) = 0,$$

or,

$$(\xi_{2j})_t + a(x_j)D_2^h\xi_{2j} - a(x_j)\frac{u_{xxx}(x_j)}{6} + O(h^2) = 0,$$

which is a second order semidiscretized approximation for the following partial differential equation,

$$(E_{2j})_t + a(x_j)E_{2x}(x_j) - a(x_j)\frac{u_{xxx}(x_j)}{6} = 0. \tag{2.11}$$

Similarly, let $\xi_{4j}h^4 = V_j - u_j$, and $\xi_{6j}h^6 = V_j - u_j$, subtracting (2.9) from (2.4), and subtracting (2.10) from (2.5), we get

$$(\xi_{4j})_t + a(x_j)\xi_{4x}(x_j) + a(x_j)\frac{u_{xxxxx}(x_j)}{30} + O(h^2) = 0,$$

$$(\xi_{6j})_t + a(x_j)\xi_{6x}(x_j) - a(x_j)\frac{4u_{xxxxxxx}(x_j)}{315} + O(h^2) = 0,$$

which are second order semidiscretized approximations for the following P.D.E.'s respectively,

$$(E_4)_t + a(x)E_{4x}(x) + a(x)\frac{u_{xxxxx}(x)}{30} = 0, \tag{2.12}$$

$$(E_6)_t + a(x)E_{6x}(x) - a(x)\frac{4u_{xxxxxxx}(x)}{315} = 0. \tag{2.13}$$

Solve these error equations we can know the behavior of the errors between the discrete solutions and the exact solutions.

The pseudo spectral method converges with the rate $O(h^p)$ where p is the number of smooth derivatives of the solution (see ,e.g., [5]).

$$E_{sp} = Cu^{[p]}h^p,$$

where $u^{[p]}$ stands for the pth derivative of u, C is a constant.

By using the Richardson error expansion, can estimate the numerical error mose easily. The key idea is to compare the numerical solution to a perturbed smooth function which satisfies the discrete equation up to higher order accuracy.

Lemma. Let $\hat{u}$ be a function

$$\hat{u}(x,t) = u(x,t) + C_2(x,t)h^2 + C_4(x,t)h^4, \tag{2.14}$$

where C_2 and C_4 satisfy

$$C_{2t}(x_j,t) + a(x_j)\frac{u_{xxx}}{6} + a(x_j)C_{2x} = 0,$$

$$C_{4t}(x_j,t) + a(x_j)\frac{u_{xxxxx}}{30} + a(x_j)C_{2xxx} + a(x_j)C_{4x} = 0.$$

Then $\hat{u}$ satisfies

$$\hat{u}(x_j,t) + a(x_j)\frac{\hat{u}_{j+1} - \hat{u}_{j-1}}{2h} = O(h^6).$$

Moreover, we can bound the error $e_j(t) = \hat{u}(x_j, t) - V(x_j, t)$ by

$$\|e\|_\infty \sqrt{h} \leq K h^6$$

where K is a constant.

proof: First, we would like to construct a solution of the form

$$\hat{u}(x,t) = u(x,t) + C_2(x,t)h^2 + C_4(x,t)h^4, \tag{2.15}$$

so that when we substitute this function into the difference equation, it satisfies the difference equation up to high order accuracy. Then we will compare the numerical solution V_J with this constructed function instead of the original solution u. The advantage of this approach is that by by taking the extra correction terms in the error expansion (2.15), we can make perturbation to the difference scheme (2.3) arbitrarily small. This is needed when we have to switch from the L_2 to the maximum norm in estimating the error.

Plug $\hat{u}_j(t)$ (2.15) into the scheme (2.3), we get

$$\begin{aligned}
& \hat{u}(x_j,t) + a(x_j)\frac{\hat{u}_{j+1} - \hat{u}_{j-1}}{2h} \\
= \; & u_t(x_j,t) + C_{2t}(x_j,t)h^2 + C_{4t}(x_j,t)h^4 + \\
& a(x_j)(u(x_{j+1},t) + C_2(x_{j+1},t)h^2 + C_4(x_{j+1},t)h^4) - \\
& (u(x_{j-1},t) + C_2(x_{j-1},t)h^2 + C_4(x_{j-1},t)h^4)/2h \\
= \; & u_t(x_j,t) + C_{2t}(x_j,t)h^2 + C_{4t}(x_j,t)h^4 + \\
& a(x_j)(u_x + \frac{u_{xxx}}{6}h^2 + u_{xxxxx}30h^4 + O(h^6)) +
\end{aligned}$$

$$a(x_j)h^2(C_{2x} + \frac{C_{2xxx}}{6}h^2 + \frac{C_{2xxxxx}}{30}h^4 + O(h^6)) +$$
$$a(x_j)h^4(C_{4x} + \frac{C_{4xxx}}{6}h^2 + \frac{C_{4xxxxx}}{30}h^4 + O(h^6))$$

by the Taylor expansion . Consider the coefficients of different order of h ,

$$\text{order one} \quad : \quad u_t(x_j,t) + a(x_j)u_x = 0, \text{from the original equation}$$

If we set the coefficients of the term of order of h^2 and h^4 to be zero for all x, i.e.

$$\text{order of } h^2 \quad : \quad C_{2t}(x,t) + a(x)\frac{u_{xxx}}{6} + a(x)C_{2x} = 0, \tag{2.16}$$

$$\text{order of } h^4 \quad : \quad C_{4t}(x,t) + a(x)\frac{u_{xxxxx}}{30} + a(x)C_{2xxx} + a(x_j)C_{4x} = 0, \tag{2.17}$$

where $C_2(x,0) = 0, C_4(x,0) = 0$, then by the theory from ordinary equations, there exists smooth solutions C_2 and C_4 uniquely. Therefore we have constructed a smooth function $\hat{u}$

$$\hat{u}(t) = u(x,t) + C_2(x,t)h^2 + C_4(x,t)h^4,$$

where C_2 and C_4 satisfy (2.16) and (2.17), such that

$$\hat{u}(x_j,t) + a(x_j)\frac{\hat{u}_{j+1} - \hat{u}_{j-1}}{2h} = O(h^6).$$

Now we compare this function $\hat{u}_j$ (instead of u) with the finite difference approximation V_j . Let $e_j(t) = \hat{u}(x_j,t) - V(x_j,t)$, then we obtain

$$e_t(x_j,t) + a(x_j)\frac{e(x_{j+1},t) - e(x_{j-1},t)}{2h} = O(h^6). \tag{2.18}$$

To prove the approximated $\tilde{u}_j$ converge to $\hat{u}_j$ we do the energy estimate. Multiplying both sides of equation (2.18) by he_j, and then summing them up, we get

$$\frac{1}{2}(\| e \|_h^2)_t + \sum_{j=-N/2}^{N/2-1} h a_j e_j \frac{e_{j+1}(t) - e_{j-1}(t)}{2h} = \sum_{j=-N/2}^{N/2-1} h e_j O(h^6).$$

By periodicity of e and summation by parts, we get

$$\begin{aligned}
\frac{1}{2}(\| e \|_h^2)_t &= -h(\sum_{j=-N/2}^{N/2-1} a_j e_j e_{j+1} - \sum_{j=-N/2}^{N/2-1} a_j e_j e_{j-1})/(2h) + \sum_{j=-N/2}^{N/2-1} h e_j O(h^6) \\
&= -h(\sum_{j=-N/2}^{N/2-1} a_j e_j e_{j+1} - \sum_{j=-N/2}^{N/2-1} a_{j+1} e_j e_{j+1})/(2h) + \sum_{j=-N/2}^{N/2-1} h e_j O(h^6) \\
&= \sum_{j=-N/2}^{N/2-1} h e_j e_{j+1} \frac{a_{j+1}(t) - a_j(t)}{2h} + \sum_{j=-N/2}^{N/2-1} h e_j O(h^6) \\
&= \sum_{j=-N/2}^{N/2-1} h e_j e_{j+1} \frac{a_{j+1} - a_j}{2h} + \sum_{j=-N/2}^{N/2-1} h e_j O(h^6) \\
&\leq \sum_{j=-N/2}^{N/2-1} h |e_j e_{j+1}| \frac{|\, a_x \,|_\infty}{2} + \|e\|_h C h^6 \sqrt{2\pi} \\
&\leq \sum_{j=-N/2}^{N/2-1} h \frac{|\, e_j \,|^2 + |\, e_{j+1} \,|^2}{2} \frac{|\, a_x \,|_\infty}{2} + \|e\|_h C h^6 \sqrt{2\pi} \\
&\leq \sum_{j=-N/2}^{N/2-1} h |\, e_j \,|^2 \frac{|\, a_x \,|_\infty}{2} + \|e\|_h C h^6 \sqrt{2\pi} \\
&\leq \|e\|_h^2 \|a_x\|_\infty / 2 + \|e\|_h C h^6 \sqrt{2\pi}.
\end{aligned}$$

Therefore,

$$\|e\|_t \leq \|e\|_h \,|\, a_x \,|_\infty / 2 + C h^6 \sqrt{2\pi},$$

and by Gronwall's inequality, we get

$$\begin{aligned} \mid e \mid_\infty \sqrt{h} &\leq (\sum_{j=-N/2}^{N/2-1} e_j e_j h)^{1/2} \\ &= \parallel e \parallel_h \\ &\leq \exp^{\frac{1}{2}|a_x|_\infty t} \parallel e(0) \parallel + \int_0^t \exp^{\frac{1}{2}|a_x|_\infty (t-s)} C h^6 ds \\ &= K h^6, \end{aligned}$$

where K is a constant. So, V_j converge to $\hat{u}_j$, and

$$\begin{aligned} V_j(t) &= \hat{u}(x_j) - e_j \stackrel{def}{=} u(x_j,t) + C_2(x_j,t)h^2 + C_4(x_j,t)h^4 + K h^{5+\frac{1}{2}} \\ &= u(x_j,t) + C_2(x_j,t)h^2 + O(h^4). \end{aligned}$$

Therefore, if we can solve for the system of ordinary equations (2.16) and (2.17), which are called the error equations corresponding to the central difference scheme (2.3), then (2.3) gives a second order central difference scheme. Similarly, we can derive the error equations for the fourth and the sixth order central difference schemes as

$$C_{4t}(x,t) + a(x)C_{4x} - a(x)\frac{u_{xxxxx}}{30} = 0, \tag{2.19}$$

$$C_{6t}(x,t) + a(x)C_{6x} + a(x)\frac{4u_{xxxxxxx}}{315} = 0. \tag{2.20}$$

Remark: For the pseudo spectral method, when a(x) is positive or negative, the convergence proof has been obtained (see, e.g., [5, 18]). But there is no convergence result for general variable coefficient $a(x)$ when a(x) changes sign. Actually the method has some mild instability [6, 4]. So it is interesting to see whether

or not this instability indeed exists in a computation and how it affects the convergence. Our calculations seem to indicate that there is no noticeable numerical instability. The spectral method converges as long as we have enough grid points to resolve the large gradient of the solution.

CHAPTER 3

Numerical result for 1-D model problem

We present a series of direct numerical simulations using the difference methods and the pseudo-spectral method for the 1-D linear hyperbolic equations. We then try to determine which method is computationally more efficient in terms of achieving an given error tolerance. Different shapes of the velocity coefficient are considered which corresponds to different degree of smoothness of the physical solution. We also include the viscosity effect.

We use the solution from the pseudo spectral method with 2048 grid points as our "accurate" solution, then compare it with those from the second, fourth, and sixth order central difference methods at different times. We present three cases of implementation with $\alpha = 0.8, \alpha = 1.0$, and $\alpha = 1.2$ respectively, and $\beta = 0.1$. These three cases correspond to different degrees of smoothness of the solution. The case of $\alpha = 1.2$ is the most singular case where the coefficient changes signs at two points. The case of $\alpha = 0.8$ gives rise a smoother solution in all times since it has a definite sign. The case of $\alpha = 1$ is in between these two cases since it is equal to zero at exactly one point. See Figure 3.1 for the shapes of $a(x)$ in these three cases.

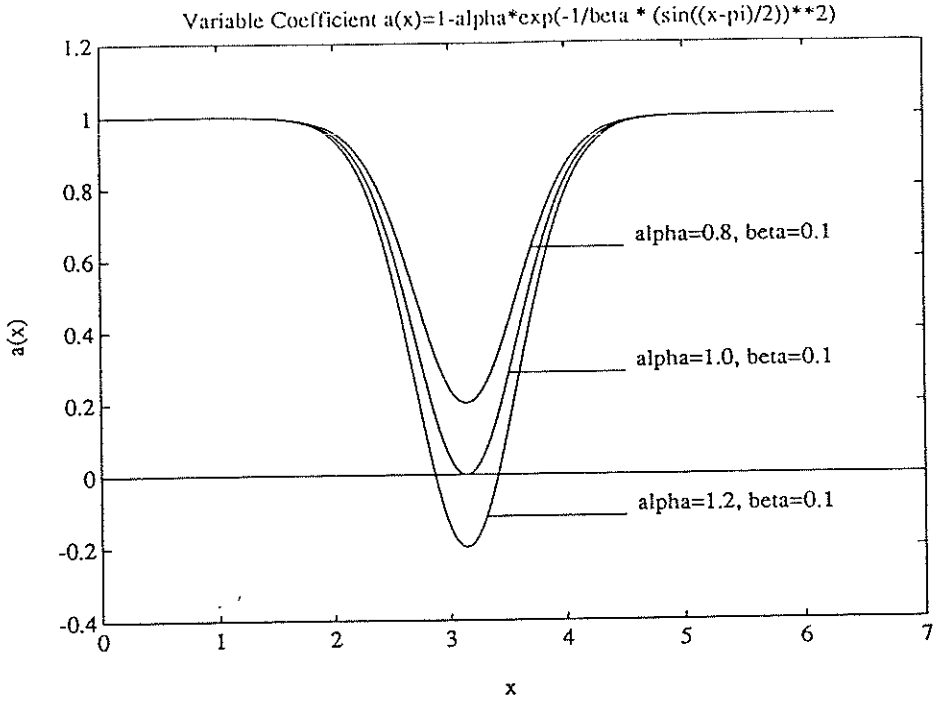


Figure 3.1

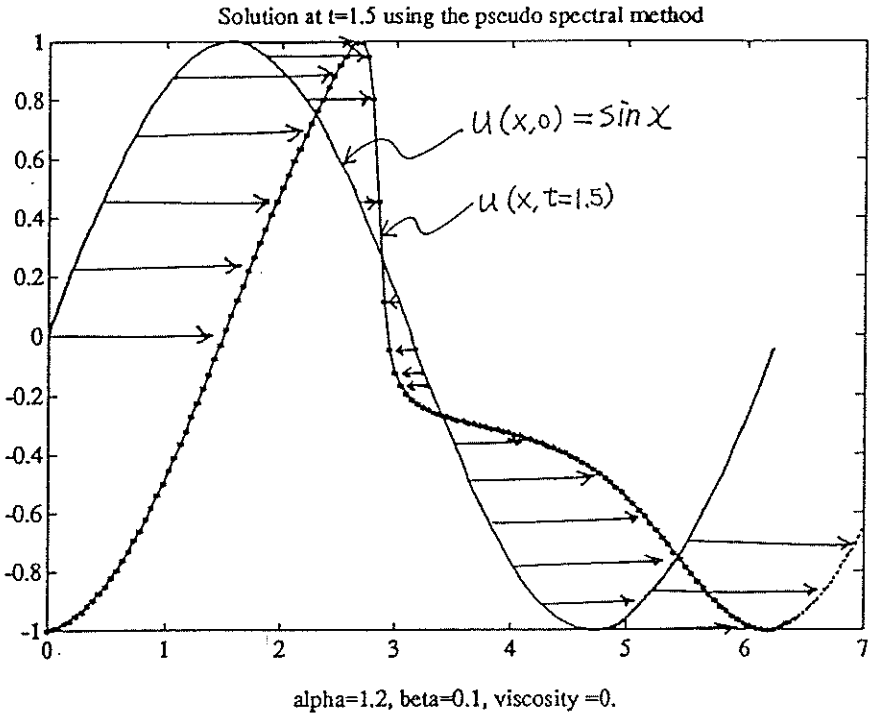


Figure 3.2

Before presenting any numerical result, I would like to give an estimate of the operation counts for these numerical methods.

N grid points	2nd order	4th order	6th order	pseudo spectral
	(3N)	(6N)	(9N)	$(5Nlog_2N - 4N)$
32	96	192	288	672
64	192	384	576	1664
128	384	768	1152	3968
256	768	1536	2304	9216
512	1536	3072	4608	20992

Table 3.1 The operation count for the convection term au_x

3.1 Inviscid cases.

We consider three cases: $\alpha = 1.2$, $\alpha = 1$ and $\alpha = 0.8$. In all these three cases, we take $\beta = 0.1$.

3.1.1 $\alpha = 1.2, \beta = 0.1, \epsilon = 0.0$

In the case of $\alpha = 1.2$, there are two places where the characteristic $\frac{dx}{dt} = a(x) = 0$. The solution will steepen up at the place where $a(x)$ changes sign from positive to negative when time increases. We can easily see this by locally approximating $a(x)$ by $-rx, (r > 0)$ at the point where $a(x)$ changes signs from positive to negative, and using $sin(x)$ as the initial data. We obtain an approximate solution near the point when $a(x) = 0$: $u(x,t) \sim sin(xe^{rt})$. This shows that

the slope of the solution tends to infinity exponentially as time increases. This phenomena is confirmed by our numerical calculation in Figure 3.2 which clearly indicates the formation of an internal layer as time increases.

In Table 3.2, we measure the errors of the second, the fourth, and the sixth order difference methods at times ranging from 0 to 3.0. The errors are obtained by comparing the numerical solutions with the "accurate" solution obtained by using the pseudo-spectral method with 2048 grid points, see Figure 3.3. We also give the errors for the pseudo-spectral method for a comparison. As we can see, the errors deteriorate very fast in times for all the methods being considered here. The larger the time is, the more spatial resolution is needed. In all these calculations, we have used the fourth order Runge-Kutta method to discretize the time integration. The time step size is taken to be so small that the discretization error in time is negligible.

N grid points	2nd order	4th order	6th order	pseudo spectral
32	0.00762	0.00475	0.00388	0.000928
64	0.00219	0.000366	0.000238	8.94e-06
128	0.000561	2.85e-05	5.8e-06	7.77e-10
256	0.00014	1.84e-06	1.19e-07	4.52e-12
512	3.51e-05	1.18e-07	2.02e-09	2.8e-13

Table 3.2-1 $\alpha = 1.2, \epsilon = 0, time = .5$

N grid points	2nd order	4th order	6th order	pseudo spectral
32	0.0667	0.0337	0.0259	0.0217
64	0.0203	0.00536	0.00345	0.0021
128	0.00449	0.000867	0.000529	2.36e-05
256	0.00113	6.15e-05	1.54e-05	5.02e-09
512	0.0003	3.85e-06	2.99e-07	5.7e-13

Table 3.2-2 $\alpha = 1.2, \epsilon = 0, time = 1$

N grid points	2nd order	4th order	6th order	pseudo spectral
32	.142	0.0418	0.0464	.134
64	0.0559	0.0552	0.0509	0.0238
128	0.0272	0.00685	0.00658	0.00217
256	0.00622	0.0012	0.000525	3.03e-05
512	0.0016	7.86e-05	1.8e-05	1.01e-08

Table 3.2-3 $\alpha = 1.2, \epsilon = 0, time = 1.5$

N grid points	2nd order	4th order	6th order	pseudo spectral
32	.206	.255	.26	.395
64	.235	.163	.142	0.0871
128	0.0866	0.0587	0.0489	0.0244
256	0.0228	0.0122	0.00885	0.0037
512	0.00683	0.00131	0.000661	3.5e-05

Table 3.2-4 $\alpha = 1.2, \epsilon = 0, time = 2$

N grid points	2nd order	4th order	6th order	pseudo spectral
32	0.577	0.592	0.59	0.769
64	0.403	0.231	0.2	0.271
128	0.222	0.102	0.0831	0.186
256	0.0908	0.0238	0.0201	0.0464
512	0.0256	0.012	0.00917	0.00349

Table 3.2-5 $\alpha = 1.2, \epsilon = 0, time = 2.5$

Ngrid points	2nd order	4th order	6th order	pseudo spectral
32	0.99	0.944	0.932	1.27
64	0.43	0.374	0.367	0.714
128	0.241	0.206	0.211	0.555
256	0.146	0.157	0.153	0.142
512	0.0933	0.0277	0.0196	0.0491

Table 3.2-6 $\alpha = 1.2, \epsilon = 0, time = 3.0$

We are interested to know how many grid points are needed to achieve a given error tolerance, say, 0.01, for these methods. Since the thickness of the internal layer decreases to zero exponentially in times, it requires more and more points to resolve the solution for large times.

The following table gives the least number of grid points for each method in achieving 1% error tolerance.

time	2nd order	4th order	6th order	pseudo spectral
0.5	32	32	32	32
1.0	128	64	64	64
1.5	256	128	128	128
2.0	512	256	256	128
2.5	512	512	256	256
3.0	more	more	512	512

Table 3.3 The least N to achieve the error tol. 0.01 ($\alpha = 1.2, \epsilon = 0$)

As we expected, all methods require more grid points to achieve the given error tolerance for large times when the solution steepens up. And up to $t = 3.0$, a fourth order and a sixth order calculation with $2N$ grid points are comparable with a pseudo spectral calculation with N grid points in the sense that they achieve the given error tolerance and using less operation counts. We also can see from Figure 3.4 that when the solution develops an internal layer at later times, a pseudo spectral calculation tends to introduce numerical oscillations which spread over the entire domain due to its global coupling nature, while finite difference calculation with local coupling stencil confine the numerical oscillations to a smaller region (see Figure 3.5).

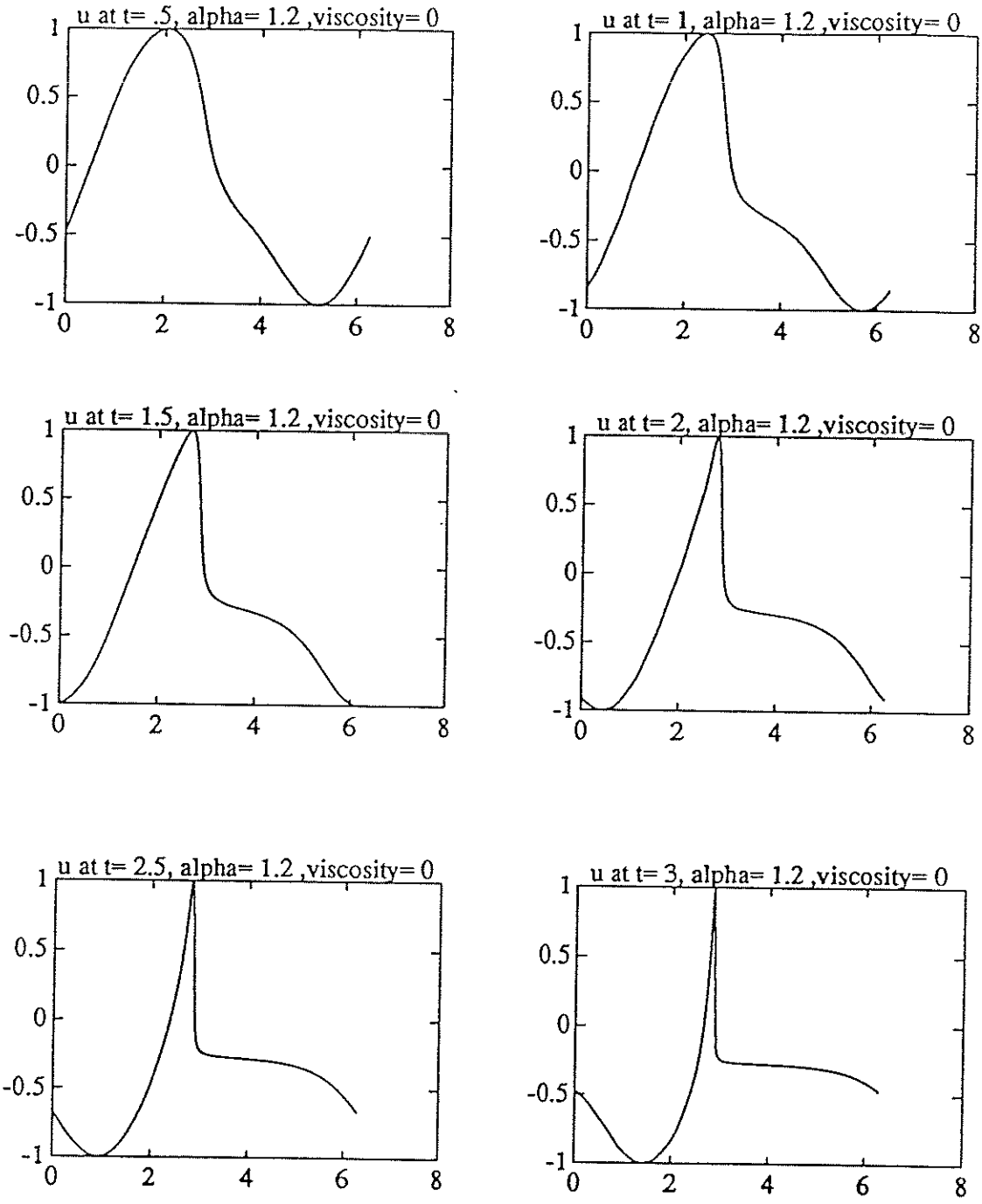


Figure 3.3 "Accurate" Sol. by Pseudo Spectral Method w/ N=2048

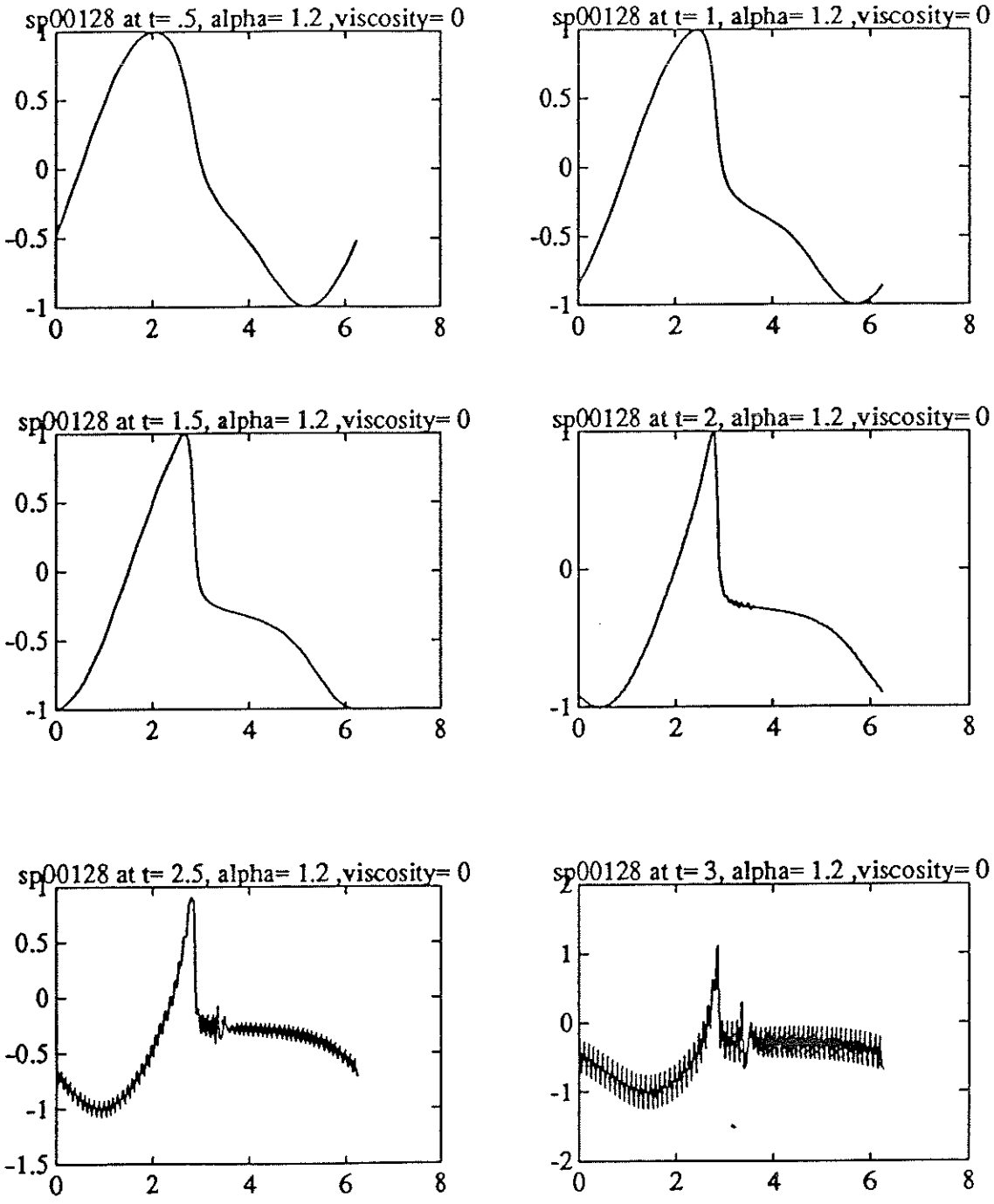


Figure 3.4 Sol. by Pseudo Spectral Method w/ N=128

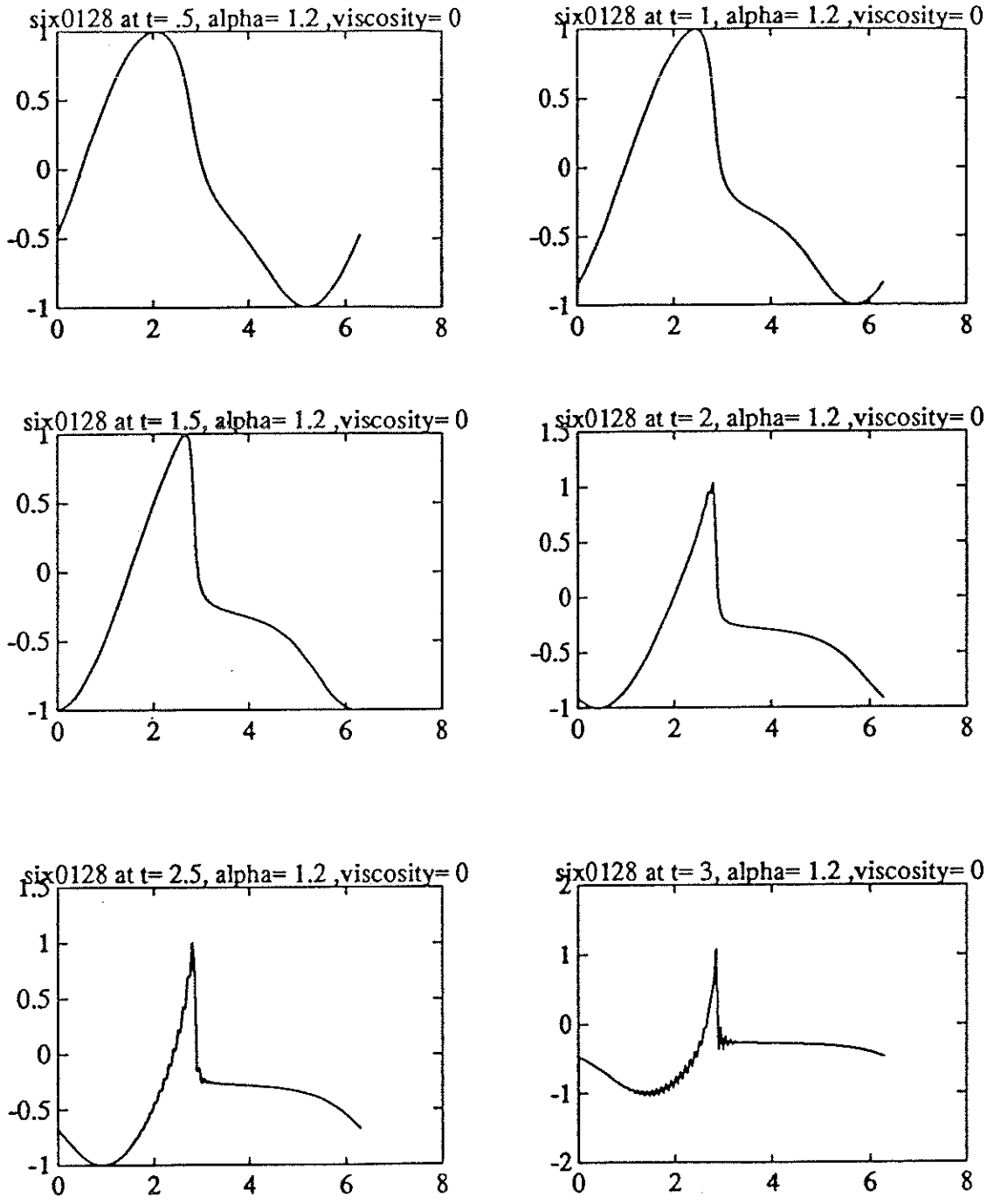


Figure 3.5 Sol. by 6th Order Difference Method w/ N=128

3.1.2 $\alpha = 1.0, \beta = 0.1, \epsilon = 0.0$

In the case of $\alpha = 1.0$, $x = \pi$ is only the point at which $a(x) = 0$. The solution will still steepen up in time and form an internal layer around $x = \pi$. But the thickness of the layer decreases to zero only algebraicly as time increases.

In Table 3.4, we list the errors of the second, the fourth, the sixth order difference methods, and the pseudo-spectral method at times ranging from 0 to 3.0. As we can see, the errors still deteriorate fast in times for all the methods being considered here, although it is not as bad as the case of $\alpha = 1.2$.

N grid points	2nd order	4th order	6th order	pseudo spectral
32	0.00794	0.00343	0.00245	0.000417
64	0.00204	0.000286	0.00013	2.07e-06
128	0.000589	1.91e-05	3.39e-06	1.13e-10
256	0.000148	1.25e-06	6.09e-08	4.08e-12
512	3.74e-05	7.98e-08	1.01e-09	2.6e-13

Table 3.4-1 $\alpha = 1, \epsilon = 0, time = .5$

N grid points	2nd order	4th order	6th order	pseudo spectral
32	0.0432	0.0113	0.0124	0.00907
64	0.013	0.00478	0.00323	0.000303
128	0.00373	0.000375	0.000152	1.07e-06
256	0.000988	2.81e-05	3.99e-06	2.24e-11
512	0.000254	1.83e-06	7.5e-08	5.1e-13

Table 3.4-2 $\alpha = 1, \epsilon = 0, time = 1$

N grid points	2nd order	4th order	6th order	pseudo spectral
32	0.0996	0.0976	0.0907	0.044
64	0.0476	0.0146	0.014	0.0049
128	0.0123	0.00303	0.0014	0.000104
256	0.00366	0.000258	7.51e-05	8.45e-08
512	0.000998	1.87e-05	1.91e-06	8.3e-13

Table 3.4-3 $\alpha = 1, \epsilon = 0, time = 1.5$

N grid points	2nd order	4th order	6th order	pseudo spectral
32	0.292	0.203	0.178	0.0971
64	0.11	0.071	0.058	0.0209
128	0.0307	0.0142	0.00945	0.00112
256	0.0111	0.00152	0.000686	1.19e-05
512	0.00302	0.000119	2.27e-05	1.34e-09

Table 3.4-4 $\alpha = 1, \epsilon = 0, time = 2$

N grid points	2nd order	4th order	6th order	pseudo spectral
32	0.431	0.253	0.218	0.204
64	0.219	0.094	0.0742	0.0446
128	0.0759	0.0296	0.0265	0.00716
256	0.0253	0.00555	0.00323	0.000211
512	0.0074	0.000532	0.000159	3e-07

Table 3.4-5 $\alpha = 1, \epsilon = 0, time = 2.5$

N grid points	2nd order	4th order	6th order	pseudo spectral
32	0.434	0.244	0.215	0.215
64	0.204	0.12	0.118	0.088
128	0.148	0.0734	0.0564	0.0193
256	0.0483	0.0125	0.00821	0.0012
512	0.0149	0.00196	0.000854	9.22e-06

Table 3.4-6 $\alpha = 1, \epsilon = 0, time = 3$

We again ask the same question: how many grid points are required for these methods to achieve the given error tolerance 0.01.

time	2nd order	4th order	6th order	pseudo spectral
0.5	32	32	32	32
1.0	128	64	64	32
1.5	256	128	128	64
2.0	512	256	256	128
2.5	512	256	256	128
3.0	/	512	256	256

Table 3.5 The least N to achieve the error tol. 0.01 ($\alpha = 1.0, \epsilon = 0$)

Although the solution for $\alpha = 1.0$ is not as singular as the one for $\alpha = 1.2$, the observation we made for the case $\alpha = 1.2$ still applies to this case. More grid points are needed to achieve the given error tolerance at later times. Again, we observe that up to $t = 3.0$, a fourth order and a sixth order calculation with $2N$ grid points are comparable with a pseudo spectral calculation with N grid points. Moreover, the numerical oscillations introduced by the pseudo-spectral method are more spread out than those introduced by the finite methods.

3.1.3 $\alpha = 0.8, \beta = 0.1, \epsilon = 0.0$

In the case of $\alpha = 0.8$, the variable coefficient $a(x)$ has only one sign. The solutions are relatively smooth, and do not form an internal layer. In Table 3.6, we list the errors of the second, the fourth, the sixth order difference methods and the pseudo-spectral method at times ranging from 0 to 3.0.

N grid points	2nd order	4th order	6th order	pseudo spectral
32	0.00635	0.00215	0.00127	0.00019
64	0.00225	0.000174	5.75e-05	2.59e-07
128	0.000569	1.49e-05	1.89e-06	5.73e-11
256	0.000146	9.64e-07	3.46e-08	3.61e-12
512	3.65e-05	6.08e-08	5.65e-10	2.3e-13

Table 3.6-1 $\alpha = .8, \epsilon = 0, time = .5$

N grid points	2nd order	4th order	6th order	pseudo spectral
32	0.0365	0.0165	0.0143	0.00334
64	0.0111	0.00185	0.0011	4.36e-05
128	0.00293	0.000172	4.22e-05	1.23e-08
256	0.000766	1.19e-05	9.53e-07	7.22e-12
512	0.000194	7.74e-07	1.72e-08	4.5e-13

Table 3.6-2 $\alpha = .8, \epsilon = 0, time = 1$

N grid points	2nd order	4th order	6th order	pseudo spectral
32	0.0919	0.0439	0.0348	0.0127
64	0.0292	0.00748	0.00468	0.000289
128	0.00829	0.000734	0.000255	5.73e-07
256	0.0022	5.9e-05	7.72e-06	1.4e-11
512	0.000553	3.82e-06	1.42e-07	6.7e-13

Table 3.6-3 $\alpha = .8, \epsilon = 0, time = 1.5$

N grid points	2nd order	4th order	6th order	pseudo spectral
32	0.132	0.0492	0.039	0.0113
64	0.0503	0.0159	0.00987	0.000899
128	0.0162	0.00192	0.000767	3.4e-06
256	0.00438	0.000152	2.46e-05	7.85e-11
512	0.0011	9.8e-06	4.65e-07	9.1e-13

Table 3.6-4 $\alpha = .8, \epsilon = 0, time = 2$

N grid points	2nd order	4th order	6th order	pseudo spectral
32	0.187	0.122	0.101	0.036
64	0.0664	0.0261	0.0179	0.00133
128	0.0235	0.00258	0.001	2.23e-06
256	0.00663	0.000231	3.48e-05	3.73e-11
512	0.0017	1.54e-05	7.16e-07	1.13e-12

Table 3.6-5 $\alpha = .8, \epsilon = 0, time = 2.5$

N prid points	2nd order	4th order	6th order	pseudo spectral
32	0.318	0.155	0.13	0.0484
64	0.114	0.0243	0.0158	0.00107
128	0.0308	0.00316	0.00111	2.13e-06
256	0.00853	0.000266	3.8e-05	3.76e-11
512	0.00215	1.69e-05	6.92e-07	1.35e-12

Table 3.6-6 $\alpha = .8, \epsilon = 0, time = 3$

As we can see, the errors increase more slowly in times compared with the case of $\alpha \geq 1$.

The pseudo spectral method clearly provides an more accurate approximation than any finite difference method considered here. However, if we ask the question: how many grid points are needed to achieve the 1% error tolerance, we still find that the fourth order or the sixth order difference method can achieve the same accuracy with twice as many grid points as required by the spectral method. Of course, in the case when the solution is very smooth, the gain of using a difference method instead of a spectral method is not as significant as the case when the solution is singular. But how to approximate a nearly singular solution efficiently is our primary concern in this study.

time	2nd order	4th order	6th order	pseudo spectral
0.5	32	32	32	32
1.0	128	64	64	32
1.5	128	64	64	64
2.0	256	128	64	64
2.5	256	128	128	64
3.0	256	128	128	64

Table 3.7 The least N in achieving the error tol. 0.01 ($\alpha = 0.8, \epsilon = 0$)

3.2 Viscous cases.

For the inviscid case, the larger the α is, the faster the solution steepens up. There will be eventually a time at which all the numerical methods fail to converge due to the lack of numerical resolution. In practice, however, there is always some physical viscosity present. This prevents the solution from getting more and more singular in time. In this case, the question is how many grid points are needed to resolve the viscous layer in order to achieve a given error tolerance. Since the thickness of the viscous layer is proportional to the viscosity, the number of grid points should be proportional to the inverse of the viscosity.

Consider the following viscous linear equation:

$$u_t(x,t) + a(x)u_x(x,t) = \epsilon u_{xx}. \tag{3.1}$$

In our implementation, we choose $\epsilon = 0.015$. To resolve the viscous layer, we require about $O(1/\epsilon)$ number of grid points. This is confirmed by our numerical experiments. The calculations with $N = 64$ provide us with fairly accurate approximations. The numerical errors for the pseudo spectral and the sixth order difference methods are below 0.01 using different values of α ranging from 0.8 to 1.2. The numerical calculations $N = 32$, on the other hand, can not accurately resolve the solution. Numerical oscillations are created in all these numerical methods.

We again compare the performance of the three finite difference methods with the pseudo-spectral method for $\alpha = 1.2, 1.0, 0.8$ respectively.

3.2.1 $\alpha = 1.2, \beta = 0.1, \epsilon = 0.015$

We list the numerical errors for the three finite difference methods and the pseudo-spectral method in Table 3.8. Although the corresponding inviscid problem is quite singular, the viscosity smooths out the internal layer to some extent and limits the smallest scale in time. As a result, the numerical methods give more accurate approximations.

N grid points	2nd order	4th order	6th order	pseudo spectral
32	0.0056	0.00322	0.00248	0.000431
64	0.00147	0.000247	0.000101	4.44e-07
128	0.000357	1.77e-05	2.52e-06	1.2e-13
256	8.86e-05	1.13e-06	4.44e-08	1e-14
512	2.22e-05	7.1e-08	7.16e-10	0

Table 3.8-1 $\alpha = 1.2, \epsilon = 0.015, time = .5$

N grid points	2nd order	4th order	6th order	pseudo spectral
32	0.0056	0.00322	0.00248	0.000431
64	0.00147	0.000247	0.000101	4.44e-07
128	0.000357	1.77e-05	2.52e-06	1.2e-13
256	8.86e-05	1.13e-06	4.44e-08	1e-14
512	2.22e-05	7.1e-08	7.16e-10	0

Table 3.8-2 $\alpha = 1.2, \epsilon = 0.015, time = 1$

N grid points	2nd order	4th order	6th order	pseudo spectral
32	0.0946	0.0281	0.0224	0.0204
64	0.0248	0.00401	0.00235	8.91e-05
128	0.00567	0.000378	8.11e-05	8.2e-11
256	0.00142	2.54e-05	1.68e-06	1e-14
512	0.000355	1.62e-06	2.85e-08	0

Table 3.8-3 $\alpha = 1.2, \epsilon = 0.015, time = 1.5$

N grid points	2nd order	4th order	6th order	pseudo spectral
32	0.102	0.0566	0.0475	0.0476
64	0.0341	0.00822	0.00464	0.000161
128	0.00932	0.00062	0.000145	2.34e-10
256	0.00228	4.11e-05	2.87e-06	1e-14
512	0.000573	2.61e-06	4.78e-08	0

Table 3.8-4 $\alpha = 1.2, \epsilon = 0.015, time = 2$

N grid points	2nd order	4th order	6th order	pseudo spectral
32	0.177	0.0948	0.0781	0.0774
64	0.0502	0.0106	0.00581	0.000182
128	0.0119	0.000757	0.000167	3.44e-10
256	0.00292	5.04e-05	3.63e-06	2e-14
512	0.000728	3.25e-06	6.08e-08	0

Table 3.8-5 $\alpha = 1.2, \epsilon = 0.015, time = 2.5$

N grid points	2nd order	4th order	6th order	pseudo spectral
32	0.243	0.116	0.0944	0.0988
64	0.0563	0.0103	0.0054	0.000157
128	0.0128	0.000782	0.000186	4.26e-10
256	0.00318	5.35e-05	3.75e-06	2e-14
512	0.000792	3.43e-06	6.47e-08	0

Table 3.8-6 $\alpha = 1.2, \epsilon = 0.015, time = 3$

For viscosity $\epsilon = 0.015$, we require about $O(1/\epsilon)$ many grid points to resolve the viscous layer. From our computational results, we clearly see that the fourth order or the sixth order achieves the error tolerance with $N = 64$. And the numerical errors tend to be unchanged at later times. This is to be expected since the thickness of the viscous layer remains fixed in times. Again we observe that with twice as many grid points, i.e. $N = 128$, the fourth order and the sixth order difference methods are comparable to the spectral method using $N = 64$ grid points. For N larger than 128, the physical solution is well resolved by all the methods considered. It is no surprise that higher order methods converge much faster. The spectral method is a clear winner in that case.

In the following table, we list the least number of grid points needed to achieve 1% error tolerance for the three difference methods and the pseudo-spectral method. Although the pseudo-spectral method can provide much more accurate approximations in the viscous case, the fourth and the sixth order difference methods can

still achieve the 1% error tolerance with at most twice as many grid points as needed for the pseudo-spectral method. Of course, if we set the error tolerance to be excessively small, like $10(-5)$ for example, the answer could be different.

time	2nd order	4th order	6th order	pseudo spectral
0.5	32	32	32	32
1.0	32	32	32	32
1.5	128	64	64	64
2.0	128	64	64	64
2.5	256	128	64	64
3.0	256	128	64	64

Table 3.9 The least N in achieving the error tol. 0.01 ($\alpha = 1.2, \epsilon = 0.015$)

3.2.2 $\alpha = 1.0, \beta = 0.1, \epsilon = 0.015$

In Table 3.10, we list the errors for the case of $\alpha = 1$, $\beta = 0.1$ and viscosity $\epsilon = 0.015$. Again, although the corresponding inviscid problem can develop an singular internal layer, the viscosity smooths out the internal layer to some extent and limits the smallest scale in time. As a result, the numerical methods give more accurate approximations.

2nd order	4th order	6th order	pseudo spectral
0.00582	0.00245	0.00168	0.000199
0.00157	0.000163	6.87e-05	1.29e-07
0.000406	1.19e-05	1.42e-06	9e-14
0.000104	7.7e-07	2.61e-08	1e-14
2.59e-05	4.84e-08	4.18e-10	0

Table 3.10-1 $\alpha = 1, \epsilon = 0.015, time = .5$

2nd order	4th order	6th order	pseudo spectral
0.0304	0.00856	0.00589	0.00196
0.00667	0.000983	0.000518	2.67e-06
0.00159	8.13e-05	1.39e-05	7.3e-13
0.000394	5.35e-06	2.56e-07	1e-14
9.84e-05	3.38e-07	4.18e-09	0

Table 3.10-2 $\alpha = 1, \epsilon = 0.015, time = 1$

2nd order	4th order	6th order	pseudo spectral
0.0495	0.0284	0.0231	0.00614
0.015	0.00293	0.00128	1.38e-05
0.00378	0.000189	3.62e-05	3.3e-12
0.000943	1.3e-05	6.84e-07	1e-14
0.000235	8.24e-07	1.12e-08	0

Table 3.10-3 $\alpha = 1, \epsilon = 0.015, time = 1.5$

2nd order	4th order	6th order	pseudo spectral
.112	0.0526	0.0403	0.00891
0.0265	0.0034	0.00169	2.4e-05
0.00595	0.000301	5.07e-05	6.65e-12
0.00151	1.97e-05	1.08e-06	1e-14
0.000375	1.26e-06	1.77e-08	0

Table 3.10-4 $\alpha = 1, \epsilon = 0.015, time = 2$

2nd order	4th order	6th order	pseudo spectral
.161	0.057	0.0496	0.0106
0.0309	0.0053	0.00252	3.64e-05
0.00781	0.000374	6.9e-05	1.34e-11
0.00196	2.44e-05	1.31e-06	1e-14
0.000487	1.55e-06	2.15e-08	0

Table 3.10-5 $\alpha = 1, \epsilon = 0.015, time = 2.5$

2nd order	4th order	6th order	pseudo spectral
.159	0.0588	0.0445	0.0188
0.0391	0.00551	0.00248	3.69e-05
0.00905	0.00038	6.81e-05	1.57e-11
0.00222	2.63e-05	1.39e-06	1e-14
0.000555	1.67e-06	2.29e-08	0

Table 3.10-6 $\alpha = 1, \epsilon = 0.015, time = 3$

If we ask how many points are needed to achieve 1% error tolerance, it turns out that the fourth order and the sixth order methods can achieve the error tolerance using the number of grid point as the pseudo-spectral method.

time	2nd order	4th order	6th order	pseudo spectral
0.5	32	32	32	32
1.0	64	32	32	32
1.5	128	64	64	32
2.0	128	64	64	32
2.5	128	64	64	64
3.0	128	64	64	64

The least N in achieving the error tol. 0.01 ($\alpha = 1.0, \epsilon = 0.015$)

Table 3.11

3.2.3 $\alpha = 0.8, \beta = 0.1, \epsilon = 0.015$

The solution for the case of $\alpha = 0.8$ is already very smooth without viscosity. With the inclusion of viscosity, the solution becomes even smoother. In Table 3.12, we list the numerical errors for the three difference methods and the pseudo-spectral method. The numerical errors for all the methods considered are extremely small.

2nd order	4th order	6th order	pseudo spectral
0.00589	0.00164	0.000963	7.61e-05
0.00176	0.000117	3.7e-05	1.54e-08
0.000454	8.71e-06	8.26e-07	8e-14
0.000114	5.57e-07	1.41e-08	0
2.86e-05	3.52e-08	2.25e-10	0

Table 3.12-1 $\alpha = .8, \epsilon = 0.015, time = .5$

2nd order	4th order	6th order	pseudo spectral
0.0231	0.00717	0.00571	0.000755
0.0059	0.000678	0.000239	4.1e-07
0.00155	4.36e-05	5.86e-06	1.3e-13
0.000388	2.79e-06	1.04e-07	1e-14
9.71e-05	1.77e-07	1.68e-09	0

Table 3.12-2 $\alpha = .8, \epsilon = 0.015, time = 1$

2nd order	4th order	6th order	pseudo spectral
0.0466	0.0191	0.0129	0.00151
0.0104	0.00136	0.000551	1.19e-06
0.00261	9.13e-05	1.25e-05	1.6e-13
0.000654	5.88e-06	2.24e-07	1e-14
0.000164	3.73e-07	3.71e-09	0

Table 3.12-3 $\alpha = .8, \epsilon = 0.015, time = 1.5$

2nd order	4th order	6th order	pseudo spectral
0.0778	0.0178	0.0136	0.00242
0.0164	0.00167	0.000515	1.67e-06
0.00376	0.000127	1.68e-05	2.2e-13
0.00092	8.14e-06	3e-07	1e-14
0.000229	5.13e-07	4.97e-09	0

Table 3.12-4 $\alpha = .8, \epsilon = 0.015, time = 2$

2nd order	4th order	6th order	pseudo spectral
0.0674	0.023	0.0155	0.00193
0.0194	0.00215	0.000786	1.8e-06
0.00485	0.000144	1.85e-05	2.3e-13
0.0012	9.25e-06	3.32e-07	1e-14
0.000299	5.83e-07	5.39e-09	0

Table 3.12-5 $\alpha = .8, \epsilon = 0.015, time = 2.5$

2nd order	4th order	6th order	pseudo spectral
.111	0.0306	0.0202	0.00273
0.0245	0.002	0.000679	2.26e-06
0.00564	0.00015	1.81e-05	2.1e-13
0.00142	9.61e-06	3.26e-07	1e-14
0.000353	6.09e-07	5.38e-09	0

Table 3.12-6 $\alpha = .8, \epsilon = 0.015, time = 3$

The solution is so smooth in this case that the pseudo-spectral method requires only 32 grid points to achieve 1% error tolerance. In comparison, the fourth order and the sixth order difference methods requires 64 grid points to achieve the same error tolerance. This is illustrated by the table below.

time	2nd order	4th order	6th order	pseudo spectral
0.5	32	32	32	32
1.0	64	32	32	32
1.5	128	64	64	32
2.0	128	64	64	32
2.5	128	64	64	32
3.0	128	64	64	32

Table 3.13 The least N in achieving the error tol. 0.01 ($\alpha = 0.8, \epsilon = 0.015$)

CHAPTER 4

Asymptotic Error Analysis

In this section, we would like to estimate the numerical errors for the finite difference methods by asymptotic error analysis. Once the error equation is obtained, what is left to do is to estimate the growth rate of the error constant. Since the error equation contains high order derivatives of the exact solution as a forcing term, it is essential to obtain a priori estimate for the derivatives of the exact solution. One way to achieve this is to replace the original equation by a simpler one which can be solved analytically. This is our first approach. We will begin with the simplest approximation to the velocity coefficient to estimate the second derivative of the exact solution. As we will see later, we need a better approximation for the velocity coefficient to obtain a useful approximation for the fourth and the sixth order derivatives. This is due to the fact that higher order derivatives grow very fast when the solution itself is nearly singular.

We approximate $a(x)$ by $-rx$ $(r \geq 0)$ near the point where $a(x)$ changes sign from positive to negative. Using the method of characteristics, we can integrate the approximated equation analytically. We obtain

$$u(x,t) \sim sin(xe^{rt}). \tag{4.1}$$

We regard this solution as a second order approximation to the exact solution since $-rx$ approximates $a(x)$ locally with second order accuracy. By direct differentiations of the above approximate solution, we obtain approximations for its high order derivatives as follows:

$$\begin{aligned}
u_x &= cos(xe^{rt})e^{rt}, \\
u_{xx} &= -sin(xe^{rt})e^{2rt}, \\
u_{xxx} &= -cos(xe^{rt})e^{3rt}, \\
u_{xxxx} &= sin(xe^{rt})e^{4rt}, \\
u_{xxxxx} &= cos(xe^{rt})e^{5rt}, \\
u_{xxxxxx} &= -sin(xe^{rt})e^{6rt}, \\
u_{xxxxxxx} &= -cos(xe^{rt})e^{7rt}.
\end{aligned}$$

Now we substitute these approximations into the error equations. By Duhamel's principle and the error equation (2.11) we get

$$E_2(x,t) = S(t,t_0)E_2(x,0) + \int_0^t S(t,\xi)\frac{u_{xxx}(x,\xi)}{6}d\xi \tag{4.2}$$

where S is the solution operator of the homogeneous problem

$$\tilde{E}_{2t}(x,t) + a(x)\tilde{E}_{2x}(x,t) = 0$$

introduced by (2.11). Thus we obtain

$$|E_2(x,t)| \leq |S(t,t_0)|_\infty |E_2(x,0)| + \int_0^t |S(t,\xi)|_\infty |\frac{u_{xxx}(x,\xi)}{6}|d\xi$$

$$
\begin{aligned}
&\leq \int_0^t |S(t,\xi)|_\infty |\frac{u_{xxx}(x,\xi)}{6}| d\xi \\
&= \int_0^t |\frac{-cos(xe^{r\xi})e^{3r\xi}}{6}| d\xi \\
&\leq \int_0^t |\frac{e^{3r\xi}}{6}| d\xi.
\end{aligned}
$$

And we can get the upper bound of the maximum of the leading coefficient of the error as following

$$
|E_2(.,t)|_\infty \leq \frac{1}{6}\frac{\exp^{3rt}-1}{3r}. \tag{4.3}
$$

Similarly, we can get the results for E_4 and E_6. The upper bound of the maximum errors are as follows,

$$
\begin{aligned}
|E_2(.,t)|_\infty h^2 &\leq \frac{1}{6}\frac{\exp^{3rt}-1}{3r}h^2 \\
|E_4(.,t)|_\infty h^4 &\leq \frac{1}{30}\frac{\exp^{5rt}-1}{5r}h^4 \\
|E_2(.,t)|_\infty h^6 &\leq \frac{1}{140}\frac{\exp^{7rt}-1}{7r}h^6
\end{aligned}
$$

Except for the second order difference method, the analytic error estimates are much smaller than the numerical errors (we have computed the asymptotic errors using these estimates for the derivatives of u, but we do not list them here because the predictions are very poor). This is due to the fact that our second order approximation of $a(x)$ is not accurate enough to provide a good bound for the higher order derivatives.

Next we expand $a(x)$ locally around the point where $a(x) = 0$ up to a higher

order term,

$$a(x) = -rx + sx^2, \qquad r = 1.3379, s = 1.13151.$$

Now solving the characteristics analytically, i.e.

$$dx/dt \sim -1.3379x + 1.13151x^2,$$

we get

$$(-1/x + 1.51385/(-1.3379 + 1.51385x))dx = 1.3379dt$$

$$ln\frac{1.51385x(t) - 1.3379}{x(t)} - ln\frac{1.51385x(0) - 1.3379}{x(0)} = 1.3379dt$$

$$x(0) = x\exp^{1.3379t}/[1 + \frac{1.51385}{1.3379}x(\exp^{1.3379t} - 1)].$$

Therefore, $u(x,t) = u(x(0))$ along the characteristic $\frac{dx}{dt} = -rx + sx^2$, i.e.

$$u(x,t) = sin(x\exp^{1.3379t}/[1 + \frac{1.51385}{1.3379}x(\exp^{1.3379t} - 1)]).$$

With one more term in the expansion for $a(x)$, we get a better approximation for u. This is essential to obtain a useful estimate for the higher order derivatives of the exact solution. Let

$$u(x,t) = sin(\frac{Ax}{1 + Bx}),$$

where $A = \exp^{1.3379t}$, and $B = 1.131512(\exp^{1.3379t} - 1)$, then we get the higher derivatives of $u(x,t)$ as follows:

$$u_x = cos(\frac{Ax}{1+Bx})\frac{A}{(1+Bx)^2}$$

$$u_{xxx} = -cos(\frac{Ax}{1+Bx})(\frac{A}{(1+Bx)^2})^3 + sin(\frac{Ax}{1+Bx})\frac{6A^2B}{(1+Bx)^5}$$

$$+cos(\frac{Ax}{1+Bx})\frac{6AB^2}{(1+Bx)^4}$$

$$u_{xxxxx} = cos(\frac{Ax}{1+Bx})(\frac{A}{(1+Bx)^2})^5 + sin(\frac{Ax}{1+Bx})\frac{-20A^4B}{(1+Bx)^9}$$

$$+cos(\frac{Ax}{1+Bx})\frac{-120A^3B^2}{(1+Bx)^8} + sin(\frac{Ax}{1+Bx})\frac{240A^2B^3}{(1+Bx)^7}$$

$$+cos(\frac{Ax}{1+Bx})\frac{120AB^4}{(1+Bx)^6}$$

$$u_{xxxxxxx} = -cos(\frac{Ax}{1+Bx})(\frac{A}{(1+Bx)^2})^7 - sin(\frac{Ax}{1+Bx})\frac{42A^6B}{(1+Bx)^{13}}$$

$$+cos(\frac{Ax}{1+Bx})\frac{630A^5B^2}{(1+Bx)^{12}} - sin(\frac{Ax}{1+Bx})\frac{4200A^4B^3}{(1+Bx)^{11}}$$

$$+cos(\frac{Ax}{1+Bx})\frac{-12600A^3B^4}{(1+Bx)^{10}} + sin(\frac{Ax}{1+Bx})\frac{15120A^2B^5}{(1+Bx)^9}$$

$$+cos(\frac{Ax}{1+Bx})\frac{5040AB^6}{(1+Bx)^8}.$$

It is reasonable to estimate $|u^{[p]}(.,t)|_\infty$ by $u^{[p]}(0,t)$ for p and t large, since the solution is most singular at the point where $a(x)=0$. Therefore, we obtain

$$\begin{aligned}
|u^{[3]}(.,t)|_\infty &= u^{[3]}(0,t) \\
&= -A^3+6AB^2 \\
&= 6.68e^{3rt}-15.36e^{2rt}-7.68e^{rt}, \\
|u^{[5]}(.,t)|_\infty &= u^{[5]}(0,t) \\
&= A^5-120A^3B^2+120AB^4 \\
&= 44.07e^{5rt}-479.54e^{4rt}+1026.60e^{3rt} \\
&\quad -786.82e^{2rt}+196.71e^{rt}, \\
|u^{[7]}(.,t)|_\infty &= u^{[7]}(0,t) \\
&= -A^7+630A^5B^2-12600A^3B^4+5040AB^6 \\
&= -9271e^{7rt}+17538e^{6rt}+35545e^{5rt} \\
&\quad -128930e^{4rt}+138010e^{3rt}-63465e^{2rt}+10578e^{rt}.
\end{aligned}$$

We can see that there are some large factors in front of the leading term which do not appear when we approximate $a(x)$ by $-rx$ previously. Now proceeding as before, we obtain the following upper bounds for the error constants:

$$\begin{aligned}
|E_2(.,t)|_\infty h^2 &\leq \frac{h^2}{6}\int_0^t |u^{[3]}(0,s)|_\infty ds \\
|E_4(.,t)|_\infty h^4 &\leq \frac{h^4}{30}\int_0^t |u^{[5]}(0,s)|_\infty ds
\end{aligned}$$

$$|E_2(.,t)|_\infty h^6 \leq \frac{4h^6}{315}\int_0^t |u^{[7]}(0,s)|_\infty ds \qquad \text{(stencil : j-4 .. j+4)}$$

$$|E_2(.,t)|_\infty h^6 \leq \frac{h^6}{140}\int_0^t |u^{[7]}(0,s)|_\infty ds \qquad \text{(stencil : j-3 .. j+3)}$$

In Table 4.1 below, we list the asymptotic errors for the 2nd, the 4th and the sixth order methods using the leading order term of $|u^{[j]}(.,t)|_\infty$ to represent $|u^{[j]}(.,t)|_\infty$. They are larger than the numerical errors listed in Table 3.2 at any time for any N, and give a good agreement with the numerical error up to the leading nonzero digit for $N = 512$ and $t \leq 2.0$.

For $N \leq 512$, the numerical solution is not well resolved for large times. The fourth or the sixth order accuracy is not strictly observed numerically. So it doesn't make sense to compare the numerical error with the analytical estimate directly in that case. Table 4.1 gives the error estimates in the case of $(a(x) \sim rx + sx^2), \alpha = 1.2, r = 1.3379, s = 1.13151$.

N grid points	t=0.5	t=1.0	t=1.5	t=2.0	t=2.5	t=3.0
32	0.0103	0.087	.658	4.9	36.5	271
64	0.00258	0.0218	.164	1.23	9.12	67.9
128	0.000644	0.00544	0.0411	.306	2.28	17
256	0.000161	0.00136	0.0103	0.0766	.57	4.24
512	4.03e-05	0.00034	0.00257	0.0192	.143	1.06

Table 4.1-1 2nd Order Error Bounds ($E_2 h^2$) from Asymptotic Error Analysis

N grid points	t=0.5	t=1.0	t=1.5	t=2.0	t=2.5	t=3.0
32	0.00893	.262	7.44	211	5.98e+03	1.7e+05
64	0.000558	0.0164	.465	13.2	374	1.06e+04
128	3.49e-05	0.00102	0.0291	.824	23.4	662
256	2.18e-06	6.4e-05	0.00182	0.0515	1.46	41.4
512	1.36e-07	4e-06	0.000114	0.00322	0.0913	2.59

Table 4.1-2 4th Order Error Bounds (E_4h^4) from Asymptotic Error Analysis

N grid points	t=0.5	t=1.0	t=1.5	t=2.0	t=2.5	t=3.0
32	0.0434	4.73	511	5.52e+04	5.97e+06	6.45e+08
64	0.000678	0.0739	7.99	863	9.33e+04	1.01e+07
128	1.06e-05	0.00115	.125	13.5	1.46e+03	1.57e+05
256	1.65e-07	1.8e-05	0.00195	.211	22.8	2.46e+03
512	2.59e-09	2.82e-07	3.05e-05	0.00329	.356	38.4

Table 4.1-3 6th Order Error Bounds E_6h^6 from Asymptotic Error Analysis

4.1 Method of characteristics

To confirm our numerical error we use the method of characteristics again.

4.1.1 Explicit Solution for Linear Equations

Lemma. The linear partial differential equation

$$u_t + a(x,t)u_x + b(x,t) = f(x,t),$$

$$u(x,0) = u_0(x)$$

where a, b, and f are smooth functions, has the solution

$$u(X(t,x),t) = \exp^{-\int_0^t b(X(s,x))ds} u_0(x) + \int_0^t \exp^{-\int_s^t b(X(\eta,x))d\eta} f(X(s,x),s)ds$$

along the characteristic line

$$\begin{aligned} dX(t,x)/dt &= a(X,t) \\ X(0,x) &= x. \end{aligned}$$

proof: Multiplying both sides of the equation by the integral factor $\exp^{\int_0^t b(X(s,x))ds}$, we get

$$\frac{d}{dt}[u(X(t,x),t)\exp^{\int_0^t b(X(s,x))ds}] = f(X(t,x),t)\exp^{\int_0^t b(X(s,x))}\,ds,$$

which implies

$$u(X(t,x),t)\exp^{\int_0^t b(X(s,x))ds} - u(x,0) = \int_0^t \exp^{\int_0^s b(X(\eta,x))d\eta} f(X(s,x),s)ds.$$

Hence, we have

$$u(X(t,x),t) = \exp^{-\int_0^t b(X(s,x))ds}(u_0(x) + \int_0^t \exp^{\int_0^s b(X(\eta,x))d\eta} f(X(s,x),s)ds.).$$

In the following, we will derive equations to determine the high order derivatives of u. We denote by u_j the jth derivative $u_x^{[j]}$. Differentiating the equation

$$u_t + a(x)u_x = 0$$

directly seven times, we get

$$
\begin{aligned}
(u_1)_t + a(u_1)_x + a_x u_1 &= 0, \\
(u_2)_t + a(u_2)_x + 2a_x u_2 + a_{xx} u_1 &= 0 \\
(u_3)_t + a(u_3)_x + 3a_x u_3 + 3a_{xx} u_2 + a_{xxx} u_1 &= 0, \\
(u_4)_t + a(u_4)_x + 4a_x u_4 + 6a_{xx} u_3 + 4a_{xxx} u_2 + a_{xxxx} u_1 &= 0, \\
(u_5)_t + a(u_5)_x + 5a_x u_5 + 10a_{xx} u_4 + 10a_{xxx} u_3 + 5a_{xxxx} u_2 + a_{xxxxx} u_3 &= 0, \\
(u_6)_t + a(u_6)_x + 6a_x u_6 + 15a_{xx} u_5 & \\
+20a_{xxx} u_4 + 15a_{xxxx} u_3 + 6a_{xxxxx} u_2 + a_{xxxxxx} u_1 &= 0, \\
(u_7)_t + a(u_7)_x + 7a_x u_7 + 21a_{xx} u_6 + 35a_{xxx} u_5 + & \\
35a_{xxxx} u_4 + 21a_{xxxxx} u_3 + 7a_{xxxxxx} u_2 + a_{xxxxxxx} u_1 &= 0.
\end{aligned}
$$

Equivalently, we can rewrite them as

$$
\begin{aligned}
(u_1)_t + a(u_1)_x + a_x u_1 &= g_1, \\
(u_2)_t + a(u_2)_x + 2a_x u_2 &= -(a_{xx} u_1) = g_2, \\
(u_3)_t + a(u_3)_x + 3a_x u_3 &= -(3a_{xx} u_2 + a_{xxx} u_1) = g_3, \\
(u_4)_t + a(u_4)_x + 4a_x u_4 &= -(6a_{xx} u_3 + 4a_{xxx} u_2 + a_{xxxx} u_1) = g_4, \\
(u_5)_t + a(u_5)_x + 5a_x u_5 &= -(10a_{xx} u_4 + 10a_{xxx} u_3 + \\
&\quad 5a_{xxxx} u_2 + a_{xxxxx} u_3) = g_5, \\
(u_6)_t + a(u_6)_x + 6a_x u_6 &= -(15a_{xx} u_5 + 20a_{xxx} u_4 + \\
&\quad 15a_{xxxx} u_3 + 6a_{xxxxx} u_2 + a_{xxxxxx} u_1) = g_6,
\end{aligned}
$$

$$(u_7)_t + a(u_7)_x + 7a_x u_7 = -(21a_{xx}u_6 + 35a_{xxx}u_5 + 35a_{xxxx}u_4 + 21a_{xxxxx}u_3 + 7a_{xxxxxx}u_2 + a_{xxxxxxx}u_1) = g_7.$$

By the above Lemma, we get

$$u_3(X(t,x),t) = \exp^{-\int_0^t 3a_x(X(s,x))ds} u_3(x,0) + \int_0^t \exp^{-\int_s^t 3a_x(X(s,x))ds} g_3(X(s,x),s)ds,$$

$$u_5(X(t,x),t) = \exp^{-\int_0^t 5a_x(X(s,x))ds} u_5(x,0) + \int_0^t \exp^{-\int_s^t 5a_x(X(s,x))ds} g_5(X(s,x),s)ds,$$

$$u_7(X(t,x),t) = \exp^{-\int_0^t 7a_x(X(s,x))ds} u_7(x,0) + \int_0^t \exp^{-\int_s^t 7a_x(X(s,x))ds} g_7(X(s,x),s)ds.$$

Therefore, we obtain approximately

$$|u_3|_\infty \leq 10.7\exp^{4.761t},$$

$$|E_2|_\infty \leq 0.3745(\exp^{4.761t} - 1).$$

But these estimates are not very sharp. The best way is to approximate these equations numerically. We will adopt two approaches. First, we use 4th order difference approximations for those derivatives of u, $u_1, u_2, ..., u_7$, and the 4th order Runge Kutta for the time discretization. Then the leading error coefficients, E_2, E_4, and E_6, converge and give good estimate for the errors for short times ($t \leq 2$). But the convergence is very slow for large times. More points are needed to obtain an accurate approximation.

The second approach uses the method of characteristics. We rewrite these equations in terms of the characteristics.

$$\frac{du_1}{dt} = (u_1)_t + a(u_1)_x = -a_x u_1 = f_1,$$

$$
\begin{aligned}
\frac{du_2}{dt} = (u_2)_t + a(u_2)_x &= -(2a_x u_2 + a_{xx} u_1) = f_2, \\
\frac{du_3}{dt} = (u_3)_t + a(u_3)_x &= -(3a_x u_3 + 3a_{xx} u_2 + a_{xxx} u_1) = f_3, \\
\frac{du_4}{dt} = (u_4)_t + a(u_4)_x &= -(4a_x u_4 + 6a_{xx} u_3 + 4a_{xxx} u_2 + a_{xxxx} u_1) = f_4, \\
\frac{du_5}{dt} = (u_5)_t + a(u_5)_x &= -(5a_x u_5 + 10a_{xx} u_4 + 10a_{xxx} u_3 + \\
&\quad 5a_{xxxx} u_2 + a_{xxxxx} u_3) = f_5, \\
\frac{du_6}{dt} = (u_6)_t + a(u_6)_x &= -(6a_x u_6 + 15a_{xx} u_5 + 20a_{xxx} u_4 + \\
&\quad 15a_{xxxx} u_3 + 6a_{xxxxx} u_2 + a_{xxxxxx} u_1) = f_6, \\
\frac{du_7}{dt} = (u_7)_t + a(u_7)_x &= -(7a_x u_7 + 21a_{xx} u_6 + 35a_{xxx} u_5 + \\
&\quad 35a_{xxxx} u_4 + 21a_{xxxxx} u_3 + 7a_{xxxxxx} u_2 + a_{xxxxxxx} u_1) = f_7.
\end{aligned}
$$

The advantage of this formulation is that the convection term is dropped and the computational grid points are self adaptive. In this formulation, we solve the following system of ordinary differential equations for the leading coefficient of the errors, E_2, $E_4$and E_6.

$$
\begin{aligned}
\frac{dX}{dt} &= a(x), \\
\frac{du_j(x,t)}{dt} &= f_j(x,t), \qquad j = 1..7, \\
\frac{E_2(x,t)}{dt} &= a u_{xxx}/6, \\
\frac{E_4(x,t)}{dt} &= -a u_{xxxxx}/30, \\
\frac{E_6(x,t)}{dt} &= a u_{xxxxxxx}/140, \qquad \text{(for j-3..j+3 stencil)}
\end{aligned}
$$

$$\frac{E_6(x,t)}{dt} = 4au_{xxxxxxx}/315. \qquad \text{(for j-4..j+4 stencil)}$$

Again, we use the 4th order Runge Kutta method to integrate the system of ordinary differential equations in time. It turns out that the particle method converges very fast for all these three error equations and for all times $t <= 3$. The error coefficients E_2, E_4 and E_6 obtained from the particle method approximation match very well with the numerical errors. It gives a better approximation than those of the asymptotic analysis and the finite difference approximation.

In Table 4.2, we list the error tables obtained by using the asymptotic error expansions and approximating the error equations by the method of characteristics. We also list the differences between the predicted errors and the measured numerical errors obtained by using the the pseudo-spectral method with 2048 grid points as an accurate solution. The approximations are extremely good for moderate times. Even for large times, the predicted errors are of the same order as the ones measured directly.

N grid points	t=0.5	t=1.0	t=1.5	t=2.0	t=2.5	t=3.0
32	0.00903	0.0769	.419	1.92	8.06	32.3
64	0.00226	0.0192	.105	.48	2.01	8.09
128	0.000564	0.00481	0.0262	.12	.504	2.02
256	0.000141	0.0012	0.00655	0.03	.126	.505
512	3.53e-05	0.000301	0.00164	0.0075	0.0315	.126

Table 4.2-1 2nd Order Error Bounds from the Particle Method

N grid points	t=0.5	t=1.0	t=1.5	t=2.0	t=2.5	t=3.0
32	0.00269	0.0203	.516	4.7	35.9	270
64	0.000392	0.00145	.109	.99	8.72	67.4
128	8.36e-05	0.000948	0.014	.22	2.06	16.7
256	2.08e-05	0.000231	0.00405	0.0539	.479	4.1
512	5.12e-06	3.99e-05	0.000972	0.0123	.117	.967

Difference between the predicted error and the measured error

Table 4.2-2, second order method, $\alpha = 1.2, \epsilon = 0$,

N grid points	t=0.5	t=1.0	t=1.5	t=2.0	t=2.5	t=3.0
32	0.00774	.258	5.44	98.8	1.61e+03	2.48e+04
64	0.000483	0.0161	.34	6.18	100	1.55e+03
128	3.02e-05	0.00101	0.0213	.386	6.28	96.7
256	1.89e-06	6.29e-05	0.00133	0.0241	.393	6.04
512	1.18e-07	3.93e-06	8.3e-05	0.00151	0.0245	.378

Table 4.3-1 4th Order Error Bounds from the Particle Method

N grid points	t=0.5	t=1.0	t=1.5	t=2.0	t=2.5	t=3.0
32	-0.00455	-0.0277	.127	4.53	135	3.85e+03
64	-0.000353	-0.00498	-0.0446	.136	8.25	240
128	-2.77e-05	-0.000844	-0.00619	-0.04	.428	14.8
256	-1.79e-06	-6.01e-05	-0.00116	-0.011	0.00937	.783
512	-1.15e-07	-3.76e-06	-7.6e-05	-0.00123	-0.0099	0.031

Difference between the predicted error and the measured error

Table 4.3-2 4th order method, $\alpha = 1.2, \epsilon = 0$,

N grid points	t=0.5	t=1.0	t=1.5	t=2.0	t=2.5	t=3.0
32	0.0346	5.47	475	3.37e+04	2.14e+06	1.28e+08
64	0.000541	0.0855	7.42	527	3.34e+04	2e+06
128	8.45e-06	0.00134	.116	8.23	522	3.12e+04
256	1.32e-07	2.09e-05	0.00181	.129	8.16	487
512	2.06e-09	3.26e-07	2.83e-05	0.00201	.128	7.61

Table 4.4-1 6th Order Error Bounds from the Particle Method

N grid points	t=0.5	t=1.0	t=1.5	t=2.0	t=2.5	t=3.0
32	-0.00388	-0.0254	0.00876	5.7	643	6.96e+04
64	-0.000238	-0.00344	-0.0501	-0.0488	9.86	1.09e+03
128	-5.8e-06	-0.000529	-0.00657	-0.0475	0.074	16.8
256	-1.19e-07	-1.54e-05	-0.000524	-0.00883	-0.0177	.112
512	-2.02e-09	-2.99e-07	-1.8e-05	-0.000661	-0.00914	-0.0154

Difference between the predicted error and the measured error

Table 4.4-2 6th order method, $\alpha = 1.2, \epsilon = 0$,

CHAPTER 5

Extension to 1-D and 2-D Burger's equations

It is of most interest to find out whether or not that those observations for the 1-D linear model problem are also true for the 2-D nonlinear problems. We consider the 2-D viscous Burger system. In the inviscid case, the Burger equation will develop a shock discontinuity at later times even if we start with smooth initial conditions. In the presence of viscosity, the shock is smoothed out. But there is still a viscous shock profile whose thickness is proportional to the viscosity. We ask the same questions for this 2-D viscous Burger system as before: Which method can achieve the given error tolerance with the most efficiency.

To perform an error analysis, we begin with a one-dimensional Burger's equation where we can afford to do well-resolved calculations.

$$u_t + uu_x = \epsilon u_{xx} \tag{5.1}$$

$$u(x,0) = sin(x). \tag{5.2}$$

This is a nonlinear equation. The solution is more compressed than the linear case, and more points are needed to resolve the shock profile. To determine the thickness of the shock profile, we introduce a change of variables:

$$y = x/\epsilon, \quad \tau = t/\epsilon$$

$$U_\epsilon(x,t) = U(x, x/\epsilon, t/\epsilon).$$

Then U satisfies

$$U_\tau + UU_y = U_{yy} \quad .$$

The transformed variable U satisfies the same viscous equation with unit viscosity. It is well known that the solution U is smooth for all times. Therefore u_ϵ has a s mallest scale proportional to $O(\epsilon)$. But unlike in the linear case, the nonlinear iteration of shock decreases the shock strength. The solution becomes smoother in time. This is also confirmed by our calculation.

The result indicates more points are required for achieving 1% error tolerance. Even for achieving 3% error tolerance, we need to have $N = 256$ for fourth order and sixth order difference methods when the solution is the most nearly singular . Basically, the observation agrees with the linear result. Forth order or sixth oreder difference methods can achieve a given error tolerance with only twice as many grid points as needed for the spectral method.

In our experiments, we use the pseudo-spectral method with 1024 grid points as our "accurate" solution. In Table 5.1, we list the errors obtained by comparing the numerical solutions of the three difference methods and the pseudo-spectral with this "accurate" solution.

N grid points	2nd order	4th order	6th order	pseudo spectral
32	0.0036	4.30e-4	1.41e-4	7.18e-6
64	0.0009	2.99e-5	3.51e-6	7.03e-10
128	0.0002	1.91e-6	6.03e-8	1.50e-13
256	5.51e-5	1.20e-7	9.85e-10	1.00e-14
512	1.38e-5	7.50e-9	1.56e-11	0

Table 5.1-1 1D Burger, $\epsilon = 0.15, time = .5$

N grid points	2nd order	4th order	6th order	pseudo spectral
32	0.1225	0.1095	0.0878	0.0363
64	0.0427	0.0317	0.0223	0.0056
128	0.0123	0.0039	0.0019	1.09e-4
256	0.0031	3.10e-4	5.45e-5	5.38e-8
512	7.79e-4	2.07e-5	1.19e-6	0

Table 5.1-2 1D Burger, $\epsilon = 0.15, time = 1.0$

N grid points	2nd order	4th order	6th order	pseudo spectral
32	1.1397	0.6038	0.4182	0.1423
64	1.0599	0.4613	0.3072	0.0881
128	0.5409	0.1889	0.1126	0.0183
256	0.1219	0.0287	0.0128	3.45e-4
512	0.0235	0.0019	4.36e-4	0

Table 5.1-3 1D Burger, $\epsilon = 0.15, time = 1.5$

N grid points	2nd order	4th order	6th order	pseudo spectral
32	1.7760	0.7128	0.4749	0.1410
64	1.5809	0.5455	0.3493	0.0865
128	0.6045	0.1916	0.1120	0.0167
256	0.1158	0.0265	0.0115	2.67e-4
512	0.0222	0.0017	3.73e-4	0

Table 5.1-4 1D Burger, $\epsilon = 0.15, time = 2.0$

N grid points	2nd order	4th order	6th order	pseudo spectral
32	2.0766	0.7221	0.4694	0.1249
64	1.5937	0.4771	0.2992	0.0688
128	0.4536	0.1387	0.0788	0.0100
256	0.0798	0.0166	0.0067	9.64e-5
512	0.0160	0.0011	1.86e-4	0

Table 5.1-5 1D Burger, $\epsilon = 0.15, time = 2.5$

N grid points	2nd order	4th order	6th order	pseudo spectral
32	2.2252	0.6810	0.4366	0.1083
64	1.8832	0.3843	0.2376	0.0519
128	0.3194	0.0948	0.0521	0.0053
256	0.0531	0.0099	0.0037	3.40e-5
512	0.0111	0.0007	8.67e-5	0

Table 5.1-6 1D Burger, $\epsilon = 0.15, time = 3.0$

We can see from the above tables that the results basically agree with the linear case. The numerical results indicate that we only need $O(\frac{1}{\epsilon})$ grid points (ϵ is the size of viscosity) to resolve the shock profile. Since the thickness of the shock profile remains to be of order $O(\frac{1}{\epsilon})$ for all times, the number of grid points needed for achieving a given error tolerance does not increase with time. A sixth order central difference calculation with $N = 256$ approximately achieves the given error tolerance 1.3% at $t = 1.5$ in the case when viscosity equals to 0.015. Beyond time $t = 1.5$, the errors begin to decrease because the strength of the shock profile is weakened as time increases due the smoothing effect from the viscosity terms. This observation is also confirmed by the 1-D calculation with viscosity 0.015 in the case $\alpha = 1.2$ and $\beta = 0.1$.

In the following two tables, we give the least number of grid points required to achieve the error tolerances 3% and 1% respectively. The results indicate that the fourth and the sixth order methods just need twice as many grid points as needed for the pseudo-spectral method in achieving the same error tolerances.

time	2nd order	4th order	6th order	pseudo spectral
hline 0.5	32	32	32	32
1.0	128	64	64	64
1.5	512	256	256	128
2.0	512	256	256	128
2.5	512	256	256	128
3.0	512	128	128	64

Table 5.2. 1-D Burger, the least N in achieving 3% error, $\epsilon = 0.015$

time	2nd order	4th order	6th order	pseudo spectral
hline 0.5	32	32	32	32
1.0	128	128	128	64
1.5	more	512	512	256
2.0	more	512	512	256
2.5	more	512	512	256
3.0	512	256	256	128

Table 5.3. 1-D Burger, the least N in achieving 1% error, $\epsilon = 0.015$

We now consider the 2-D viscous Burger system.

$$u_t + uu_x + vu_y = \epsilon(u_{xx} + u_{yy}) \tag{5.3}$$

$$v_t + uv_x + vv_y = \epsilon(v_{xx} + v_{yy}) \tag{5.4}$$

$$u(x, y, 0) = sin(x) * sin(y) \tag{5.5}$$

$$v(x,y,0) \;=\; cos(x) * cos(y). \tag{5.6}$$

We perform the same comparison between the finite difference methods and the pseudo-spectral method. In this case, we take the numerical solution obtained by using the sixth order difference method with 256×256 grid points as our "accurate" solution. The pseudo-spectral method is simply too expensive to compute for data sizes larger than 256×256 grid points on a Convex vector computer. It already takes about one day to compute the solution using the sixth order difference method with 256×256 grid points. The corresponding calculation using the pseudo-spectral method is estimated to take about 9 days.

In the following table, we give the least number of grid points needed in achieving 2.5% error tolerance. We can see that the observations we made for the 1-D model problem still hold for the 2-D problem.

time	4th order	6th order	pseudo spectral
1.5	256	256	128
2.0	256	256	128
2.5	256	256	128

Table 5.4. 2-D Burger system, the least N in achieving 2.5% error, $\epsilon = 0.015$

CHAPTER 6

2-D implementation on Parallel Computer CM-2:

As we can see in the calculations for the 2-D Burger system, it requires extremely high numerical resolutions to achieve a given error tolerance. As the viscosity ϵ tends to zero, the solution becomes more and more singular and it requires more and more grid points to resolve it. Eventually the required resolution cannot be fulfilled by a sequential computer. A more powerful supercomputer is required to achieve the desirable numerical resolution.

In this chapter, we implement our numerical experiments on a massive parallel machine, the connection machine CM-2. For practical purposes, we study the question of what is the optimal order of finite difference methods in the sense of achieving a given error tolerance with the least computational effort. For sequential computers, the computational time is mainly governed by the arithmetic operation count, and we found from the previous chapter that the fourth order difference method gives the best performance in terms of achieving a given error tolerance with the most computational efficiency. The computer architectures in a massively parallel super computer is fundamental different from that of a sequential machine. In a parallel machine, communication is the most time consuming part in general. We need to take into account the existing architecture for FFT

and the interprocessor communication factor. Naturally one may wonder if the comparison result obtained in a sequential machine still applies to a massively parallel machine.

In the following table, we give the total computational times for the 2nd, 4th and 6th order difference methods and the pseudo-spectral method.

N grid points	2nd order	4th order	6th order	pseudo spectral
32	0.0032	0.0055	0.0102	0.1653
64	0.0049	0.0101	0.0164	0.2989
128	0.0110	0.0221	0.0346	0.5680
256	0.0329	0.0611	0.0911	2.2244
512	0.1131	0.1981	0.2804	8.3098

Table 6.1 Execution times (sec) on the CM-2.

As we can see from the table above, the ratio of the computational times among these methods are roughly equal to $1.000 : 1.857 : 2.769 : 67.61$ in the case of 256×256 and $1.000 : 1.752 : 2.479 : 73.47$ in the case of 512×512.

Recall that the fourth order difference method can achieve the same order of accuracy as the sixth order method using the same number of grid points. And the fourth or sixth order method can achieve the same order of accuracy as the pseudo-spectral method using at most twice as many grid points. From the above table, we can see that using the fourth order method can save the computational time over the pseudo-spectral method up to a factor of 10 in achieving the 2.5% error

tolerance. And the sixth order method can save the computational time over the pseudo-spectral method up to a factor of 7. On the other hand, the second order method can achieve roughly the same order of accuracy (2.5 %) as the sixth order method using twice as many grid points. From the above table, we can see that the computational time for the second order method with twice as many grid points is comparable with that for the sixth order method. So the second order method is competitive with the sixth order method if the memory storage is not a concern. But taking all the factors into consideration, the fourth order method is still the most preferable method. This is because it takes roughly the same number of grid points to achieve the same accuracy as the sixth order method and is much faster than the second order method with twice as many points. Moreover it is about 10 times faster than the pseudo-spectral method to achieve the same error tolerance. In Part II, we will establish a timing model to analyze the performance of high order difference methods on CM-2. We find that the fourth order method remains to be the winner even under various possible improvements on the communication and floating point operations.

Part II. Performance Modeling on Parallel Computer CM-2

CHAPTER 7

Introduction to Part II.

There are a large class of physical problems whose solutions require large scale computations. The computation of turbulent flows is one example. Typically, these computations demand a extremely high numerical resolution in order to resolve the physically interesting small scale structures. Naturally one would like to perform such a calculation on a computer which can provide both extremely high speed arithmetic operation and a large memory storage. A massively parallel supercomputer has both of these advantages. By dividing the computational work into many individual processors, a parallel computer can in effect achieve a very high speed up over the conventional sequential machine. This provides a very powerful addition to the existing computational environments and makes many difficult computational problems become more tractable now.

One major difficulty in a massively parallel computer is that the inter-processor communication is very costly. In many cases, the communication time could be comparable to the time used for the arithmetic operation. Although many attempts have been made to reduce the communication time in a parallel machine, the communication time is still a very important factor that could affect the overall computational performance. For a given computational problem, one needs to

design the algorithm in such a way that it minimizes the communication time. Very often this is problem dependent. From a general user's point of view, it would be very desirable if one can provide some kind of timing model which can predict the overall performance by taking into account both the arithmetic operation count and the inter-processor communication time. This is the purpose of the second part of this thesis.

Finite difference methods are perhaps the most commonly used numerical methods in computational fluid dynamics. As a natural continuation to my previous work, I choose to study a timing model for high order finite difference approximations for the inviscid and viscous Navier-Stokes equations in periodic geometries. There are many different ways in designing a difference approximation depending on individual physical problem. Here we only study the simplest version of all: the symmetric finite difference approximation. The obvious advantage of this method is that it can be easily vectorized and parallelized.

As we know from the previous study, a high order difference method has the advantage of achieving a given error tolerance with a relatively fewer grid points. But it also suffers from being more expensive computationally. The interesting question is what is the optimal order of the scheme in the sense of achieving a given error tolerance with the most computational efficiency. The answer depends on several factors. First of all, it depends on how smooth the physical solution is and how much one can afford to completely resolve the smallest scale in the solution. Secondly it also depends on the arithmetic operation count of the method.

Lastly and most importantly it depends on the communication time needed for the numerical method. It is not difficult to see that the higher order the method is, the more expensive the communication time is in addition to the increase of the operation count. Thus there is a very subtle balance between the accuracy and the computational cost.

There are two execution models to compile a CM fortran program, the fieldwise (Paris) and slicewise models. The main difference is in the way the compiler maps CM array onto the underlying hardware. In the fieldwise model, the storage of a single precision number (a 32-bit word) is allocated in 32 consecutive bits of a physical bit-serial processor's memory. It requires a interface called "Sprint Chip" to transpose the data for the floating point operations. In the slicewise model, on the other hand, a 32-bit word data is stored in a 32-bit slice across the memory of 32 bit-serial processors. In this manner of storage, the floating point operations can be performed directly using the data from the common processor memory without additional transposition of data. This distinct feature of the floating point architecture significantly improves the CM's floating point computational power. For more detailed discussions regarding these two models, see chapter 1.1 and chapter 2.

Chapter 2 of the Part II is devoted to studying a timing model based on the slicewise model on CM-2 which takes into account both the communication time and the arithmetic operation time. In our timing model, we propose a method to estimate the purely internal communication time and the purely external com-

munication time. These two communication factors are then combined with the arithmetic operation estimate to provide an accurate prediction for the overall computational time needed for a high order difference method. Our preliminary calculations indicate that the relative error of our timing model is less than 10% in the three dimensional geometry on 16K CM-2, and less than 20% in two and three dimensional geometries on 8K CM-2. One new feature of our model is that the overhead time in the internal communication depends on the geometry of the data. Our model provides an explicit formula which account for this fact.

There are some existing timing models proposed for the fieldwise model on CM-2, see, e.g., Levit [13], and Pozo [15]. In the fieldwise model, each 32-bit word is stored in a bit-serial processor. And one can specify a geometry by using a special Paris function call. This defines the number of virtual processors simulated by a bit-serial processor in each dimension. Then one can use the specific description of each bit-serial processor to estimate the communication time for the fieldwise model. Appropriate parameters are obtained by using a least square fit. However, in the slicewise model, a 32-bit word data is stored in a 32-bit slice across the memory of 32 bit-serial processors. Thus it does not make sense to talk about the subgrid sizes of each bit-serial processor, because there is no single data completely stored in one bit-serial processor. Instead we should consider a floating point node (consist of 32 bit-serial processors sharing a floating unit) as the basic element of the slicewise model, and the VP ratio should be defined in terms of the number of virtual processors in this basic element. As a consequence, the timing model

developed for the fieldwise model does not directly apply to the case of the slicewise model, and a new timing model for the slicewise model is needed. To the best of our knowledge, there are no such models available for the slicewise model yet.

We use the timing model developed in chapter 2 of this part to perform a number of interesting studies. First of all, our timing model validates the observation we made in the previous chapter: the fourth order finite difference method is the most favorable method in terms of achieving 1% error tolerance with the least computational effort. This corresponds to the case when the solution develops a small scale structure. When the solution is very smooth (such as in the case of $\alpha = 0.8$ for the hyperbolic equation), a higher order method (6th order or higher) is more preferable depending on the size of the given error tolerance. We also ask the question of how sensitive this conclusion depends on the specific performance parameters of a parallel computer. To answer this question, we study the behavior of the our timing model by varying different parameters in our model, e.g. the communication startup overhead constant, and internal and external communication factors, as well as the floating point operation parameter which measures the arithmetic performance cost. To our surprise, the timing model is very stable under these perturbations. In the case of large data sizes 256×256, 512×512, and 1024×1024, the arithmetic time tends to dominate the communication time. Reducing the communication time by a factor of 10 only changes the overall time by roughly 35 percent. On the other hand, reducing the arithmetic operation time by a factor of 10 significantly reduces the overall computational time by roughly

55 percent. In the latter case, doubling the number of grid points in the second order method takes about as much time as the sixth order method. From our previous study on accuracy of these methods, the second order method with twice as many grid points performs equally well as the sixth order method from the point of view of accuracy. But doubling the total number of grid points costs more memory storages. So there is still some preference of using the sixth order method to the second order one. The fourth order is the true winner. This is because it takes roughly the same number of grid points to achieve the same accuracy as the sixth order method and is much faster than the second order method with twice as many points. This study seems to suggest that the fourth order difference method is still preferable even with an improved compiler of CM-2. We also have done the comparison by reducing various parameters by a factor of 100. Results are very similar to what we have discussed above.

The rest of the thesis is organized as follows. In the following I shall give a brief introduction to the architecture of CM-2, the parallel machine on which we have performed most of our calculations. In chapter 2, we derive and analyze our timing model, and estimate the relative error of the model. In chapter 3 we use the timing model developed in chapter 2 to study some possible improvements of performance by varying different parameters in the timing model. The main conclusions are summarized at the end of chapter 3.

7.1 Introduction to CM-2.

The Connection Machine Model CM2 is a massively parallel computing system, of the single instruction multiple data (SIMD) type, and with distributed memory hypercube structure compacted. The CM system consists of a parallel processing unit containing thousands of data processors, each with its own memory, all acting under the direction of a serial processor called the front end. There is also an I/O system that supports mass storage, graphic display devices and other peripherals. For details, see [20, 19].

A fully configured Connection Machine Model CM2 consists of 64K (65536) bit-serial processors which has a clock rate of 6.7 MHz and utilizes 2048 Weitek 3132 floating point units (FPU). When properly pipelined, the floating point unit performs a single precision 32-bit-floating point multiplication and a 32-bit-addition for each clock cycle. This gives us an aggregate peak floating point performance over 27 Gflops:

2048×2 floating point ops/cycle $\times 6.7 \times 10^6$ cycles/sec $\approx 27.4 \times 10^9$ floating point ops/sec.

To understand the performance on CM2, we have to first understand its fundamental structure. The central element of the CM system is the parallel processing unit, which contains

- thousands of data processors – (from 2K to 64K) each data processor acts very much like a serial computer (it can execute arithmetic and logic instruc-

tion, calculate memory address, and perform interprocessor communication or I/O) but there is a difference, which is the data processors do not get instructions from their respective memories. Instead, they are collectively under the control of a single micro-coded sequencer.

- a sequencer – to decode commands from the front end computer and broadcast them to the data processors, so that all the data processors execute the same instructions simultaneously.

- an interprocessor communication network – very few useful problems can be decomposed into completely independent subproblems, and most require the processors to interact. Processors must be able to pass information among themselves in the pattern best suited to the needs of the moment; the pattern must be able to adapt to any applications needed, and to change over time. CM2 supports this with 3 kinds of interprocessor communication:

    1. Nearest-neighbour (NEWS) communication – There is hardware support for the grid-based communication - nearest neighbour accesses, dimensional sifts, and cumulative computation along grid axis.

    2. General purpose (router) communication - each processor gets a value from any arbitrary processor.

    3. Global communication (scan)

- Zero or more I/O controllers and/or frame-buffer modules.

7.2 Two execution models: Paris (Fieldwise) model/ Slicewise model

There are two execution models the programmer can choose when he/she compile a CM Fortran program, the Paris (Fieldwise) or the Slicewise models. Here the notation "Paris" stands for *parallel arithmetic instruction sets.* The difference is in the way the compiler maps CM arrays onto the underlying hardware. The slicewise model only runs on a CM with the optional 64-bit floating-point accelerator, while the Paris model can execute on any CM configuration.

Here we introduce two terms: processing node and processing element (PE).

Processing node

- All CMs are organized into processing nodes which consists of 32 bit-serial processors and other associated hardware.
- If the CM system has a floating point accelerator, then it has one floating-point chip per node. (i.e. one floating-point chip per 32 bit-serial processor)

Processing element (PE) The basic unit of either model

	Paris (Fieldwise) Model	Slicewise Model
CM system required	It can be run on any CM hardware configuration	requires a CM w/ 64-bit floating point accelerator
processing element	Each bit-serial processor. e.g. a 64K CM-2 in Paris model has 64K PEs.	Each processing node. (i.e. 32 bit processors w/ FPU) e.g. A 64K CM-2 in slicewise model has 64K/32 = 2K PEs.
performance	slower	faster (due to special optimization)

7.3 Virtual Processing

Virtual processor facility is an important feature of CM system that enables each PE to simulate more than one virtual processor per PE equally dividing its memory and sequentially serving the virtual processors. The virtual processor facility associates one virtual processor (VP) with one data element of a data set.

The set of all virtual processors associated with a data set is called a virtual processor set (or a VP set) See Example 1. The ratio of the virtual processors required by the application to the number of physical PEs is called the VP ratio. The VP ratio indicates how many times each PE must perform a certain task

in order to simulate the appropriate number of virtual processors. In this way, a program can be executed unchanged on CM system with different number of physical processors. Furthermore, using high VP ratio allows one to run extremely large size problem and also improves the performance of the CM.

Example 1. Suppose we have a VP set of 64K (65536) virtual processors. In Paris model execution, if the program runs on a 64K CM, the VP ratio is 1 and if the program runs on a 32K CM, the VP ratio is 2. In the slicewise model execution, if the program runs on a 64K CM, the ratio is 32 while if the program runs on a 32K CM, the VP ratio is 64.

7.4 Principles of optimizing the performance of CM-2

To achieve the best performance on the CM system, an application program must maximize the processor use and streamline or reduce interprocessor communication, which can be summarized as:

- Use as many CM processors as possible in each operation because if a program leaves the processors inactive, it reduces the advantage of parallel processing.
- Avoid interprocessing node communication whenever possible because operations within processors are faster than those between processors.
- Whenever communication is necessary, use the most efficient communication mechanisms and paths.

Therefore the allocation of arrays on the CM gives different efficiency in performance. There is a canonical layout which the CM Fortran compiler uses to allocate arrays on the CM to achieve the above goals for many array uses. For some applications, one can get better efficiency by using a different layout. LAYOUT is one of the compiler directives in CM Fortran for this purpose. LAYOUT causes an array to be laid out in CM memory in a way that either reduces interprocessor communication or optimizes the speed of communication along specific dimensions.

7.5 VP geometries

The VP set is at least the size and the bank of the array. However, the VP set may be larger than the array because the exact size and shape (the geometry) of a VP must also meet some constraints set by the execution model.

- In the Paris model, the axes of the VP grid must be a power of 2. Hence the VP ratio is also a power of 2.
- In the slicewise model, the total size of the VP grid must be a multiple of 4 times the number of processing nodes. (i.e. Right now, the 64 bit floating point chips have a vector of 4. This constraint will be taken out in the future).

CHAPTER 8

Analysis of floating point operation and grid communication model

To study the performance of a program on CM-2 it is sufficient to study the performance of the operation within one processor and its inter-processor communication. If each grid point does not need to communicate with other grid points, then one need not worry about the data distribution. In that case, we just need to make sure that we use as many processors as possible to have a good load balance. However, in numerical approximations of physical problems, we need to approximate the physical differential equations by difference operators. These involve sending or fetching information from other grid points. In this case, communication between different grid points and/or processors plays a significant role in the overall performance. How to distribute the data point according to the specific computer architecture becomes very important if one wants to minimize the communication cost.

8.1 Fieldwise model versus slicewise model.

Our timing model is established based on the newest slicewise model. This model performs much faster than the standard fieldwise model. In the fieldwise model, the storage of a 32-bit word is allocated in 32 consecutive bits of a physical

bit-serial processor's memory. In each clock cycle, a bit-serial processor can send out only one bit of a single-precision (32-bit) floating point value. However, the (Weitek 3132) floating point unit can operate a 32-bit word in each clock cycle. Therefore it takes 32 cycles for a bit-serial processor to send out the whole value of a 32-bit floating point number to the Weitek 3132 floating point unit. To balance the bandwidth between these two processors (32 to 1), each two CM chips (total 32 bit processors) are designed to attach to one Weitek 3132 floating point unit. Thus these 32 bit-processors send out 32 single precision words in 32 clock cycles. which in average achieves one single precision word per clock cycle.

However, in this manner of storage, those 32 bits sent out by associated 32 bit-serial processors in each cycle do not represent any single precision number. Each bit just comes from a different bit-serial processor. Therefore, it requires an interface to transpose these 32 bits into a proper format for the Weitek 3132 floating point unit. This interface is called "Sprint Chip" between the common processor memory and the Weitek 3132 floating point unit.

In practice, it is not very efficient to store the data fieldwise and do transposition back and fourth between the common memory area and the Weitek 3132 floating point unit. Recently the CM architects have designed a new slicewise model to view the processors in a slicewise configuration. A 32-bit word is stored in a 32-bit slice across the memory of those associated 32 physical bit processors in the processing node (a processing node consists of 32 bit-serial processors sharing a FPU). In other words, each bit of a 32-bit number is stored in a different processor of that

processing node, 1-bit per processor. Therefore, with slicewise model, the Weitek 3132 floating point unit actually access data from their associated processors in each clock cycle. That is to say, in each cycle a 32-bit slice across processors is read into the floating point unit. Thus the slicewise model saves the transposition work between the main memory and the Weitek 3132 floating point unit. This distinctive feature of the floating point architecture significantly improves the CM's floating point computational power.

There are some existing timing models proposed for the fieldwise model on CM-2, see, e.g., Levit [13], and Pozo [15]. In the fieldwise model, each 32-bit word is stored in a bit-serial processor. And one can specify a geometry by using a special Paris function call. This defines the number of virtual processors simulated by a bit-serial processor in each dimension. Then one can use the specific description of each bit-serial processor to estimate the communication time for the fieldwise model. Appropriate parameters are obtained by using a least square fit. However, in the slicewise model, a 32-bit word data is stored in a 32-bit slice across the memory of 32 bit-serial processors. Thus it does not make sense to talk about the subgrid sizes of each bit-serial processor, because there is no single data completely stored in one bit-serial processor. Instead we should consider a floating point node (consist of 32 bit-serial processors sharing a floating unit) as the basic element of the slicewise model, and the VP ratio should be defined in terms of the number of virtual processors in this basic element. As a consequence, the timing model developed for the fieldwise model does not directly apply to the case of the slicewise

model, and a new timing model for the slicewise model is needed. To the best of our knowledge, there are no such models available for the slicewise model yet.

In Figure 8.0, we present the structure of such a floating point node which consists of 2 CM chips linked with a common memory area and its interface chip. The Sprint chip links between the processor common memory area and the Weitek 3132 floating point unit. In fieldwise model, the Sprint chip needs to transpose the arguments between the processor memory and the Weitek 3132 unit. In the slicewise model, on the other hand, it requires no transposition.

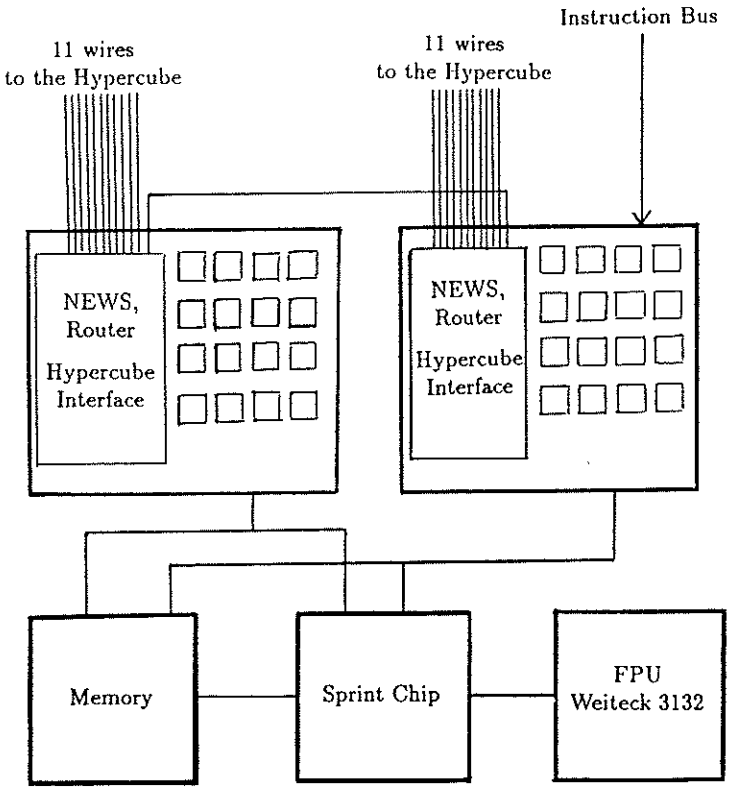


Figure 8.0 A Processing Node

So what is the difference between the fieldwise model and the slicewise model?

It seems that the only difference between them is the performance of floating point operations. Can we still use the same analytical model which has been developed for the fieldwise model? It is not clear. Recall that in a slicewise model, data are stored slicewisely across the memory of those associated 32 bit processors in each floating point node. There are no data completely stored in any single bit-serial processor. By using a special Paris function call, one can specify the number of virtual processors simulated by a bit-serial processor in each dimension. Then one can use the specific description of each bit-serial processor to estimate the communication time for the fieldwise model. Appropriate parameters are obtained by using a least square fit. In the slicewise model, on the other hand, a 32-bit word data is stored in a 32-bit slice across the memory of 32 bit-serial processors. Thus it is irrelevant to talk about the subgrid sizes of each bit-serial processor. Instead we should consider a floating point node as the basic element of the slicewise model, and the VP ratio should be defined in terms of the number of virtual processors in this basic element. Therefore the analytical model provided by Levit [13] and Pozo [15] in fieldwise model could not be used directly in the slicewise model. A new timing model is required to study the performance of the slicewise model.

8.2 Performance Analysis of Floating Point Operation for Slicewise CM-2

What is a reasonable model for the floating point operation on the slicewise model of CM-2 ? From the previous section, we understand the underlying archi-

tecture of each floating point node of CM-2, see Figure 8.0. We can decompose the floating point operation into three parts : (1) sending the data from common memory area through the Sprint chip without transposition to Weitek 3132 FPU, (2) performing computation in the Weitek 3132 FPU, (3) sending the result through the Sprint chip without transposition back to the associated bit processors. Therefore, it is reasonable to expect that the floating point operation performance model consists of a overhead time for filling the pipeline of the floating point vector co-processor unit and a linear growth rate when the pipeline is filled.

To verify our model for the floating point operations, we have performed a number of experiments. In our experiments, we compute each basic floating point operation one thousand times and average the measured time. Then we repeat the same computations for various data sizes. The measured time is plotted as a function of data sizes in Figure 8.1, which clearly indicates that the execution time grows linearly with respect to the VP ratio. Denote by n the VP ratio per processing node, by $T(n)$ the time for a floating point operations. c is the overhaed time to fill the pipe and α is the linear growth rate when the pipeline is filled. They satisfy $T(n) = c + \alpha * n$.

By using a least square fit, we obtain the corresponding overhead time, c (μsecs/vp), and pipelined rate, α (μsecs/vp) for each basic floating point operation. We list the values of these two parameters for different basic floating point operations in Table 8.1.

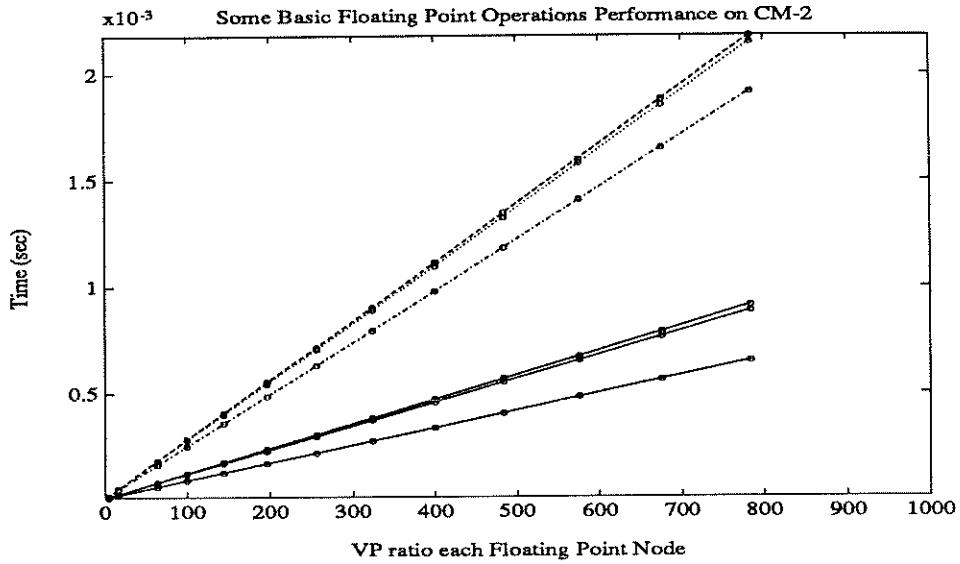


Figure 8.1: Linear relation between floating point performance and VP ratio

operation	c (ν sec)	α (ν sec)
x=a+b	9.9928	1.1555
a=a+s	9.5606	0.8233
x=x-b	10.1604	1.1557
a=a-b	10.3266	1.1212
a=a-s	9.5659	0.8233
x=x*b	10.1588	1.1557
a=a*b	10.3123	1.1212
a=a*s	9.6001	0.8233
x=a/b	9.4305	2.7644
a=a/s	9.3962	2.4302

Table 8.1: Performance Model for Some Basic Floating Point Operations,

It is important to note that the vector division or scalar division takes much more time (more than 2.5 times) in the floating point operation calculation stage than vector addition, vector subtraction or vector subtraction. Recall that a Gflop is defined as the ratio of number of floating point operations and the computational time, i.e. $N/(C + \alpha N) = 1/(\alpha + C/N)$. Moreover the higher the VP ratio, the more efficient a CM-2 is being used. Therefore higher VP ratio gives better Gflop performance of floating point operations. This is also confirmed by our experiments in Figure 8.2 where we plot the Gflop performance versus the VP ratio of each processor node. Different curves in Figure 8.2 correspond to different basic floating point operations listed in Table 8.1. Note the result reflected in Figure 8.2 is performed on a 8K CM-2. We can expect to 1.8 to 2.4 Gflop performances for a fully loaded 64K CM-2 because a 64K machine has 8 times as many processing nodes as for a 8K machine.

In fact , the CM-2 Weitek 3132 floating point unit has a pipe for multiplication and addition so that they can be performed in a unit clock cycle. In Table 8.2, we list the computational time of several triad vector operations. As we can see the computational times of the two floating point operations are not additive. Instead multiplication and addition are performed in a pipelined fashion, so that the performance of the resulting triad vector operations is almost twice as fast as that of the basic floating point operations described above. In Figure 8.3 we plot the triad floating point operation time versus VP ratios. We still observe the same

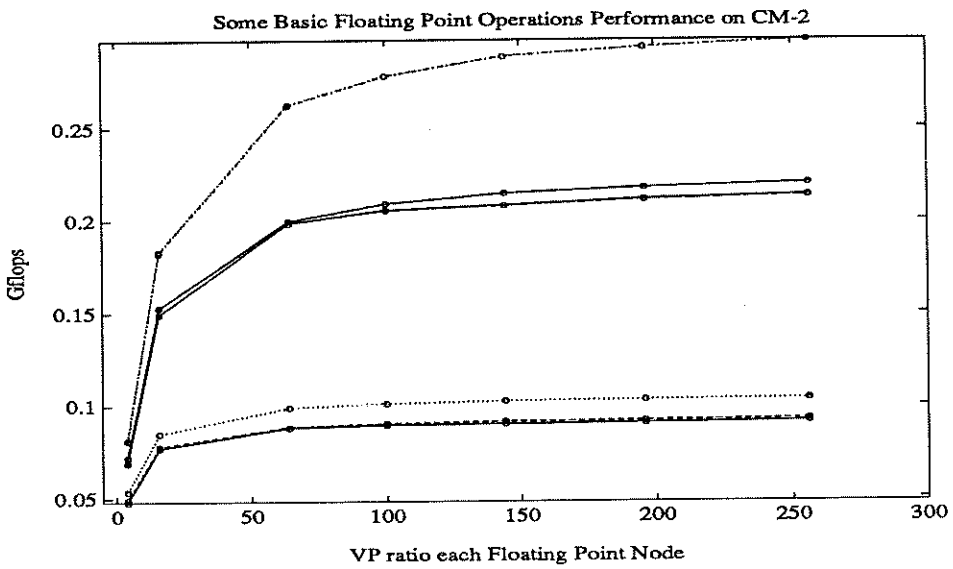


Figure 8.2:

linear relationship between the execution time and the corresponding VP ratio. We also plot their Gflop performance in Figure 8.4. They can achieve about 3.3 Gflops on a 64K CM-2.

operation	$c(\mu sec)$	$\alpha(\mu sec)$
a = s * (a+b)	10.4871	1.3475
x = s * (a+b)	10.6963	1.3819
c = c * (a+b)	8.8799	1.3885
x = c * (a+b)	9.4128	1.4210
x = c * a + b	9.2841	1.4996
c = c * a + b	8.7737	1.4616
x = a + s * b	10.6234	1.2338
a = a + s * b	10.2402	1.1962
a = t + s * b	13.0484	0.9692
b = t + s * b	11.7802	0.8979
b = t + a * b	10.2576	1.1962
a = t + a * b	10.6656	1.1977

Table 8.2: Performance Model for Some Triad Floating Point Operations,

We have tested the accuracy of our model for the floating point operations. In Figure 8.5 we plot the relative errors of the predicted time by the above models for the floating point operations versus VP ratios. We can see that the maximum relative error is less than 3 % . And for high VP ratios the model prediction is almost exact.

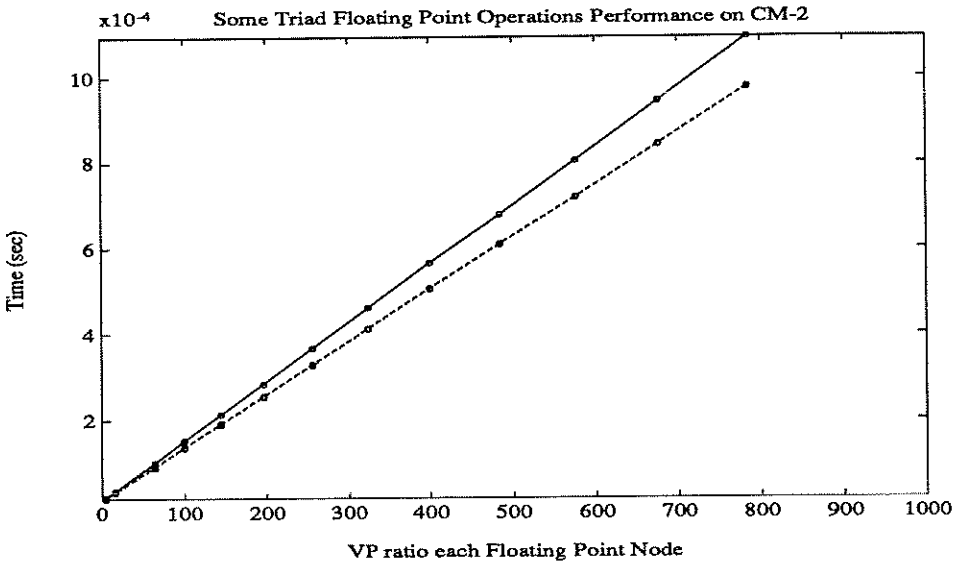


Figure 8.3: Linear relation between triad floating point performance and VP ratios

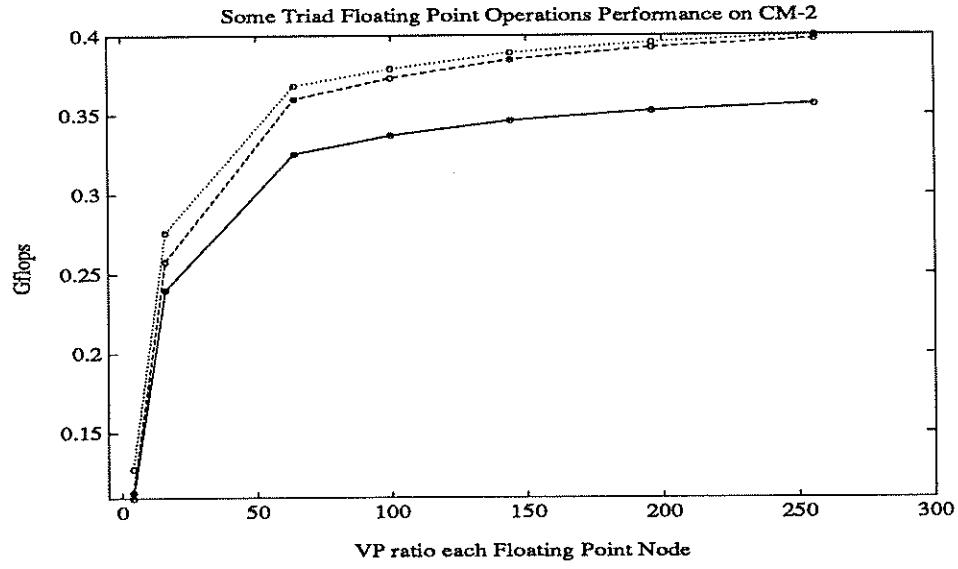


Figure 8.4: Gflop performance of triad floating point operations vs VP ratios

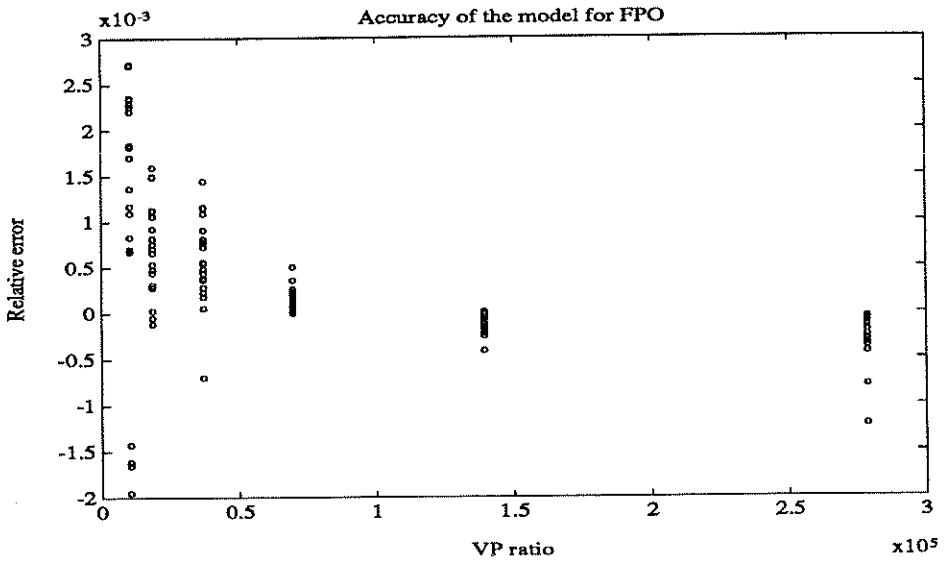


Figure 8.5: Accuracy of the model for floating point operations

In practice, the arithmetic operations are more complicated than those listed above. It could be a combination of many different basic and triad floating operations. In this case, the average Gflop performance for the floating point operations is much better than the basic and triad operations. In Figure 8.6, we give the performance (on an 8K CM) of many commonly used arithmetic expressions for finite difference schemes. For example, the curve at the bottom in Figure 8.6 corresponds to the simple arithmetic express $A + \epsilon(s(U - V) + tW)$, where A, U, V, W are vectors, s and t are scalars. The curve on the top corresponds to a more complicated arithmetic expression

$$A + \epsilon(s(U_1 - U_2) + t(V_1 - V_2) + r(W_1 - W_2) + gZ),$$

where the upper case letters stand for vectors, the lower case ones stand for scalars. The curves from the bottom to the top correspond to different arithmetic expres-

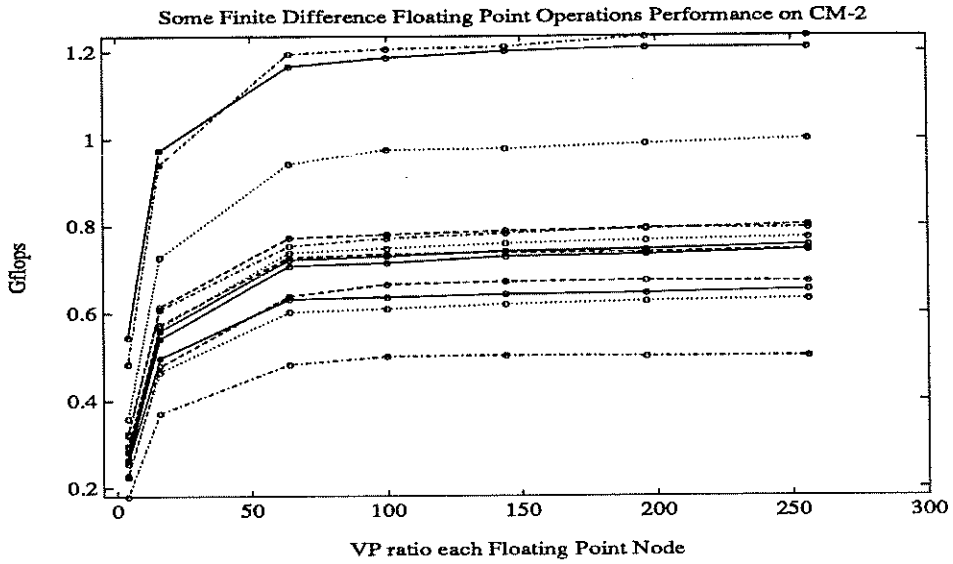


Figure 8.6: Gflop performance of finite difference operations and VP ratio

sions with increasing complexities. We can see from Figure 8.6 that the Gflop performance increases as the arithmetic expression becomes longer.

8.3 Grid Communication Analysis for Slicewise CM-2

The communication performance of the slicewise model is different from that of the fieldwise model. Therefore, we can not use the existing communication model for our timing model. With different data allocation strategy, slicewise model not only provides faster performance on the floating point operation, but also forces us to reconsider the data distribution on CM in a manner different from the fieldwise model. In the fieldwise model, each single precision data is allocated to some bit-serial processor so that it makes sense to study the VP ratio of a bit-

serial processor. However, in slicewise model, there will be no any single precision data which is stored in the memory of a single bit-serial processor. Each single precision data is stored across the memory of 32 bit processors of a floating point node. Therefore, it makes no sense to discuss how many data are in a bit-serial processor. Instead, we have to view a floating point node as the basic unit in the study of the slicewise model of CM-2. For this reason, we only consider the floating point node from now on. Thus a 64 K CM becomes 2048 floating point nodes, and 8K CM become 256 floating point nodes.

Unlike the fieldwise model, there are fewer layout directives in specifying a data structure in the slicewise model. In the fieldwise model, one can specify a data structure to determine how those dimensions of the original problem be mapped to the CM processors by calling the following Paris subroutine **CM-creat-detailed-geometry()** . However, we cannot do this in the slicewise model. But, as a CM fortran user, one does not need to specify or even worry about which mapping is optimized. CM fortran compiler uses a canonical layout, NEWS layout, to allocate arrays to achieve nearly optimized performance . We only need to use CMF-DESCRIBE-ARRAY() to figure out how the data being distributed into those floating point nodes.

Let us denote the data structure of a data set on CM-2 as $G = \{P, V\}$. The data structure $G = \{P, V\}$ is commonly called as the "geometry" of the data set. Here $P = \{p_1, p_2, ..., p_k\}$ denotes the physical processing node dimensions, and $V = \{v_1, v_2, ..., v_k\}$ denotes the virtual processing node dimension.

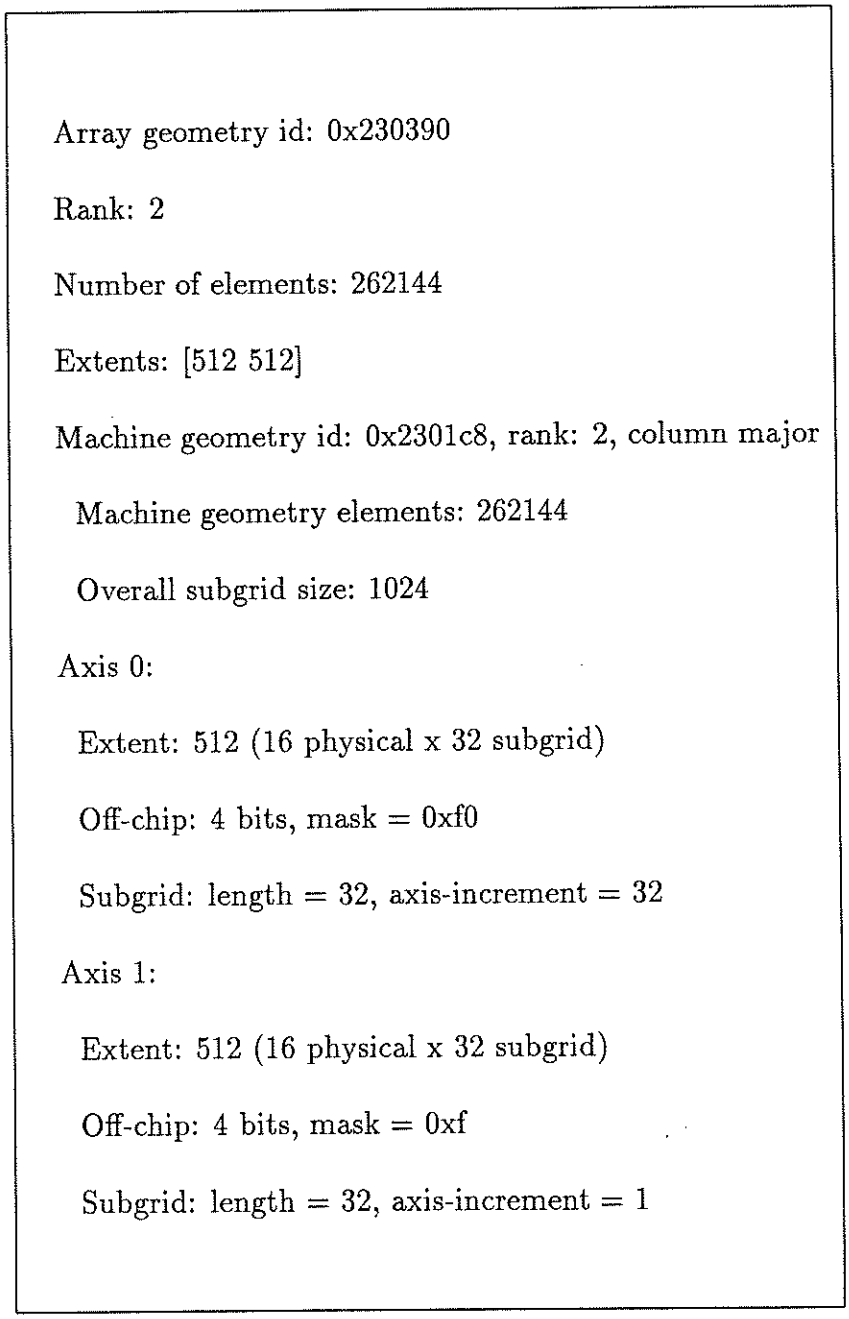
Although the slicewise compiler can automatically choose a layout to optimize the performance, it is difficult for us to know what is the geometry of the data structure without calling CMF-DESCRIBE-ARRAY() . In the following, we give a few examples to illustrate the underlying geometries associated with different problems. The first one is relatively easy to guess. The last two are less intuitive.

Example 1. The default layout is given by (:news,:news), i.e. each subgrid dimension is distributed by the nearest neighbor data distribution. We would like to find out how a 512×512 data is mapped to a 8K CM. In the slicewise model, a 8K CM becomes 256 floating point nodes. In this example, they are configured as a 2D dimension hypercube, with physical processing node dimension 16×16, and each virtual processing node gets a chunk of data with subgrid size 32×32. The geometry G consists of

$$P = \{16, 16\},$$

$$V = \{32, 32\}.$$

We can find out the exact geometry layout by calling **CMF-DESCRIBE-ARRAY()**. The results are given below.



Grid decomposition of a 512x512 grid onto a 8K CM-2

Example 2. We consider a three-dimensional default layout (:news,:news,:news). We would like to find out how a $128 \times 128 \times 128$ data is mapped to a 8K CM. In this case, those 256 floating point nodes are configured as a 3D dimension hypercube,

with physical processing node dimensions $4 \times 8 \times 8$, and each virtual processing node gets a chunk of data with subgrid sizes $32 \times 16 \times 16$. Its geometry G is described by

$$P = \{4, 8, 8\},$$

$$V = \{32, 16, 16\}.$$

We get the following information by calling CMF-DESCRIBE-ARRAY() .

```
Array geometry id: 0x230108
  Rank: 3
   Number of elements: 2097152
    Extents: [128 128 128]
    Machine geometry id: 0x22ff40, rank: 3, column major
    Machine geometry elements: 2097152
    Overall subgrid size: 8192
  Axis 0:
    Extent: 128 (4 physical x 32 subgrid)
    Off-chip: 2 bits, mask = 0xc0
    Subgrid: length = 32, axis-increment = 256
  Axis 1:
    Extent: 128 (8 physical x 16 subgrid)
    Off-chip: 3 bits, mask = 0x38
    Subgrid: length = 16, axis-increment = 16
  Axis 2:
    Extent: 128 (8 physical x 16 subgrid)
    Off-chip: 3 bits, mask = 0x7
    Subgrid: length = 16, axis-increment = 1
```

Grid decomposition of a 128x128x128 grid onto a 8K CM-2

Example 3. This time we consider the default layout (:news,:news,:news) with data size $32 \times 512 \times 512$. In this case, these 256 floating point nodes are configured as a 3D dimension hypercube, with physical processing node dimensions $2 \times 8 \times 16$, and each virtual processing node has a chunk of data with subgrid sizes $16 \times 64 \times 32$. In this example, the geometry G consists of

$$P = \{2, 8, 16\},$$

$$V = \{16, 64, 32\}.$$

We get the following information by calling CMF-DESCRIBE-ARRAY() .

```
Array geometry id: 0x22ba38
  Rank: 3
  Number of elements: 8388608
  Extents: [32 512 512]
  Machine geometry id: 0x22b870, rank: 3, column major
    Machine geometry elements: 8388608
    Overall subgrid size: 32768
  Axis 0:
    Extent: 32 (2 physical x 16 subgrid)
    Off-chip: 1 bits, mask = 0x80
    Subgrid: length = 16, axis-increment = 2048
  Axis 1:
    Extent: 512 (8 physical x 64 subgrid)
    Off-chip: 3 bits, mask = 0x70
    Subgrid: length = 64, axis-increment = 32
  Axis 2:
    Extent: 512 (16 physical x 32 subgrid)
    Off-chip: 4 bits, mask = 0xf
    Subgrid: length = 32, axis-increment = 1
```

Grid decomposition of a 32x512x512 grid onto a 8K CM-2

8.3.1 Data mappings on higher dimensional data structures

Suppose the total size of the original problem is k-dimensional with size $(n_1, n_2, ..., n_k)$ and we work on CM with 2^d processors. The geometry consists of $G = \{P, V\}$, where $P = \{p_1, p_2, ..., p_k\}$ denotes the physical processing node dimensions, and $V = \{v_1, v_2, ..., v_k\}$ denotes the virtual processing node dimension. It should satisfy the following two properties:

- The product of the physical processing node dimension, p_i, and the virtual processing node dimension, v_i, equals to the problem dimension size, for i-th dimension, where $i = 1...k$.

$$n_i = p_i * v_i,$$

- The product of the physical processing node dimension equals to the number of the processing nodes on which we work on. 2^d CM processors becomes 2^{d-5} processing nodes in the slicewise model. $\prod p_i = 2^{d-5}$

After we find out the description of the data mapping, we can easily study the mechanism of the grid communication of the slicewise model on CM-2.

8.3.2 Decomposition of NEWS grid communications

The NEWS grid communication can be decomposed into two major parts:

- On-Node communication : Communication between two virtual nodes which are on the same physical node. Since there is a common memory area for a node, it only costs memory moving (copying) within the same node. We denote the time required for transferring 32-bit word within the same node as t_M, M stands for internal Memory moving (copying).

- Off-Node communication : Communication between two different physical processing nodes. This requires moving (copying) data to a temporary buffer, and transfer data through Off-Node hypercube network. We denote the time required for transferring 32-bit word off-node as t_E, E stands for external communication.

Since we now view an assemble of 32 bit serial processors sharing with a FPU as a basic unit of the slicewise model of CM-2, the communication mechanism becomes much simpler than the fieldwise model. Consider Example 1 above. Suppose we would like to calculate the finite difference operator , ∂x, or ∂y on the 2D 512x512 data grid described in Example 1. A simple first order forward difference approximation for u_x with periodic boundary , at grid (i,j), is given by

```
do i = 1,n
        ux (i,j) =( u(i+1,j) - u(i,j) ) / h
end do
ux(1,j) = (u(1,j) - u(n,j)) / h
```

In the CM fortran command, it is translated to the following operation using the cshift function (circular shift):

(cshift (u,1,1) - u) / h

This requires each grid point to send a data to its nearest neighbor node (with distance one) along the first dimension. Why the nearest neighbor node? Because in the NEWS ordering, the data are mapped into the hypercube in a manner that all neighboring grids are mapped into the neighboring nodes of the hypercube.

One way to estimate the communication time of this difference operation is as follows. Assume every (virtual) node wants to communicate one floating point value (32-bit) to its west neighbor. We need to perform the NEWS communication with distance one along the first axis. Those virtual nodes located in the left boundary have to be sent out through the Off-Node (external) hypercube network to its neighbor. There are totally 32×1 external communications. Others are sent to their destinations within the same physical node, and it is simply a memory moving (copying) within a node, which is very fast. There are 32×31 internal memory movings (copying) . By a straightforward calculation, the overall communication time would be given by $32 \times t_M + 32 \times 31 \times t_{OUT}$. This is similar in spirit to the models used by Livet [13] and Pozo [15]. The only difference between the model proposed above and the models used in [13] and [15] is that we use this formulation for the processing node instead of for a bit-serial processor. The drawback of the communication model proposed above is that it ignores the overhead time in the internal memory moving. As we'll see in the next section, the overhead time in the internal communication can grow proportionally with respect to the product of the second and the third subgrid dimension of the data set in

the (:serial,:news,:news) layout.

8.4 Model for Internal Communication

In order to account for the dependence on the geometry of the data for the internal communication time, we propose a more sophisticated model to correct the discrepancy in the previous model. The idea is to separate the study of the internal communication from the external communication. For this purpose, we use a (:serial, :news, :news) layout with different data sizes as test cases. We perform $cshift$ operation of distance d along the first dimension. This corresponds to purely internal communication because of the special layout. For simplicity, we consider the case of $d = 1$. Other cases can be treated similarly.

One important observation in our study is that the overhead time in the internal communication depends on the geometry in a subtle way. See Figure 8.7. In this figure, each line corresponds to a set of data with similar geometry. For those data of sizes of $k \times 128 \times 128$, with (:serial, :news, :news) layout and k varying from 4 to few hundred, the performance of *cshift* with distance 1 along the first dimension is linear with respect to the size of total subgrid sizes of data. We observe the same behavior for data of sizes $k \times 256 \times 256$, $k \times 512 \times 512$, and $k \times 1024 \times 1024$ respectively. Again the performance of each set of data is linear with respect to the size of total subgrid sizes. However, the corresponding overhead times are different from each other.

A careful study shows that the overhead time of internal communication of

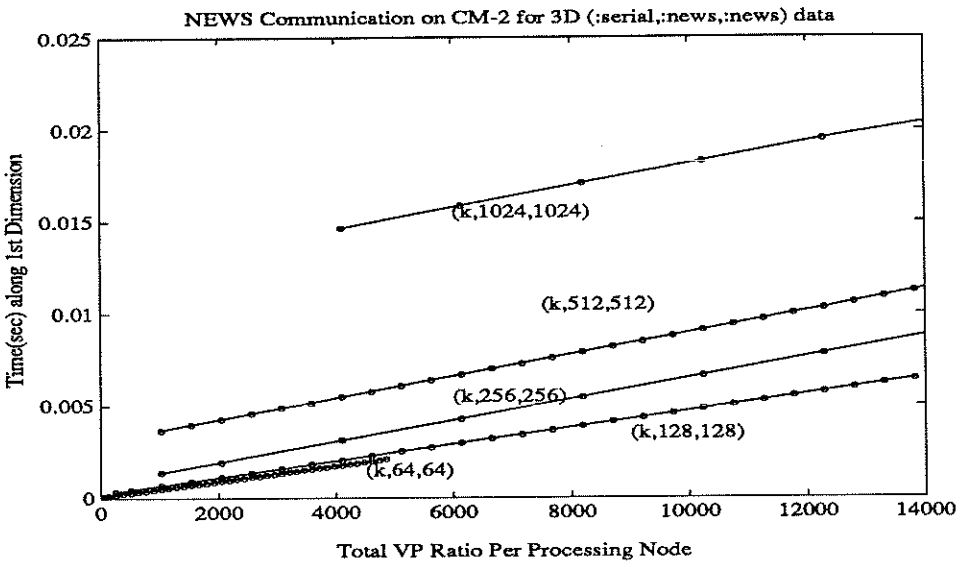


Figure 8.7:

each data set depends on its particular geometry. Consider a data set with a layout (:serial,:news,:news) and of subgrid size $v_1 \times v_2 \times v_3$. A serial array dimension has the distinctive property that it is always allocated entirely within (never across) processing nodes. Therefore the $cshift$ along the first dimension takes place within each processing node. This gives the natural way to describe the model of internal communication by performing a $cshift$ with distance one along the first dimension on such a data set.

Roughly speaking our internal communication model can be understood by considering the memory moving as a "DO LOOP" process. Consider a 3D data with (:serial, :news, :news) layout and the subgrid sizes v_1, v_2, and v_3. To make a $cshift$ on such a data set along the first dimension, we begin by performing the memory copying (moving) along that dimension. This consists of a basic overhead cost, C_1,

and a memory copying rate, t_M. But we have to repeat the same communication procedure v_2 times along the second dimension for each memory moving along the first dimension. Thus the accumulated time is $T = (C_2 + (C_1 + t_M \times v_1) \times v_2)$, where C_2 is the overhead time along the second dimension. Finally we have to repeat v_3 times along the third dimension the same memory moving procedure for the first two dimensions. Therefore the overall internal communication time is given by

$$T = (C_2 + (C_1 + t_M \times v_1) \times v_2) \times v_3,$$

where v_1, v_2, and v_3 are the subgrid sizes in each processing node, and t_M is the internal communication rate which may depend on the distance of communication. The overhead time for the internal communication is now given by $C_1 \times v_2 \times v_3 + C_2 \times v_3$. It is roughly proportional to the product of its subgrid sizes along the second and the third dimensions. Note that we have neglected the overhead time along the third dimension since it is very small compared with $C_1 \times v_2 \times v_3 + C_2 \times v_3$.

8.5 Model for External Communication

To find out the model for the external communication, we measure the time of some data sets with the layout (:serial, :news, :news). (We can also use (1000:news, :news, :news) layout) The idea is to construct some purely external communication cases. The best way to achieve this effect is to move the data along the second or the third dimension with distance equal to the subgrid size of that dimension. This causes the entire chunk of data in a node to move to its nearest node. For

example, if we consider a data of size $128 \times 128 \times 128$, with (:serial, :news, :news) layout, its subgrid sizes are $128 \times 8 \times 4$ on a 16K CM. To perform communication along the second dimension with distance 8, all the data within a processing node need to move out to its nearest neighboring node. This means there are $32 \times 8 \times 4$ external communications, and no internal communication. From a least square fitting, we easily get the cost for external communication in this case as

$$T = C_{OUT} + t_{OUT} \times v_1 \times v_3$$

where $C_{OUT} = 24.41 \times 10^{(-6)}$ sec and $t_{OUT} = 8.376 \times 10^{(-6)}$ sec. In this case, the overhead time does not depend on the geometry .

REMARKS: For distance less than the subgrid size of the axis performing communication, the internal memory moving rate is different. With a careful study, we can model the internal communication coefficient t_M as a linear function of the distance. For distances of power of two and larger than the subgrid size of the communication axis, we only need to take twice as much time as needed for communication with distance equal to the subgrid size of the communication axis. This is due to the binary-reflected Gray-Code ordering of the off-chip bits in the grid address [9].

8.6 Combine Internal and External Communication Together

In this section we combine the results on the internal communication with the external communication to form a general model for the overall communication

model. We still use the following example to illustrate our model. Consider a data set with (:news,:news,:news) layout and of subgrid sizes $v_1 \times v_2 \times v_3$. So a processing node simulates $v_1 \times v_2 \times v_3$ virtual processing nodes. A $cshift$ along the second dimension with distance d, where $d < v_2$ consists of both internal and external communication. This is because under the NEWS ordering the data are allocated across different processing nodes. Suppose all virtual processing nodes communicate a distance d to the right along the second axis. There are d layers of data to the rightmost side requiring off node communication, while the remaining $v_2 - d$ layers only need to do memory moving internally. Thus the total communication time T is the sum of both internal and external communication time. This is due to the fact that a SIMD machine can perform either internal or external communication instruction at any time. we thus have:

$$T = T_{internal} + T_{external}$$

$$T_{internal} = (C_2 + (C_1 + t_M \times (v_2 \text{ - d})) \times v_1) \times v_3$$

$$T_{external} = C_3 + t_{OUT} \times v_1 \times \text{ d } \times v_3$$

We can decompose the external communication t_{out} into two parts. One is memory moving to a temporary buffer t_M, the other is sending out through the external hypercube wire t_{EX}, i.e. $t_{OUT} = t_M + t_{EX}$. Then the formulas above can be rewritten as:

$$T_{Internal\_Memory\_Moving} = (C_2 + (C_1 + t_M \times v_2) \times v_1) \times v_3$$

$$T_{External\_Sending} = C_{EX} + t_{EX} \times v_1 \times d \times v_3$$

$$\text{T} = T_{Internal\_Memory\_Moving} + T_{External\_Sending}$$

where d is the distance of communication less than the subgrid size v_2.

Remark. So far, we have used three dimensional data sets as examples to explain the communication performance. In general, the communication for an m dimensional data set is performed in a manner similar to a three dimensional case [7]. For example, suppose we perform a $cshift$ operation along the jth axis along which the data are distributed in the NEWS ordering ($j <= m$). If the dimension $m > 3$, then the data of subgrids $V = v_1, v_2, ..., v_m$ is considered as $V = \prod_{k=1}^{j} v_k, v_j, \prod_{k=j+1}^{m}$, and if the dimension $m < 3$, then the the data will be padded to be a three dimensional one. Then the communication is performed as for the three dimensional data sets described above.

8.7 Accuracy of the NEWS Communication Model

To test the accuracy of our communication model, we use it to predict the time needed for performing one $cshift$ of distance d along the jth axis, where distance d is either an integer between 1 and v_j or a number of a power of two and greater than v_j if v_j is a power of two. For example, if $j = 2$, the predicted communication time is estimated by using the following formula

$$T = \begin{cases} (C_2 + (C_1 + t_M(v_2 - d))v_1)v_3 & \\ +C_{OUT} + t_{OUT}(d)v_1v_3, & \text{if } 1 \leq d < v_2; \\ C_{OUT} + t_{OUT}v_1v_2v_3, & \text{if } d = v_2; \\ 2(C_{OUT} + t_{OUT}v_1v_2v_3), & \text{if } d > v_2 \text{ and } d = 2^p \text{ for some integer } p. \end{cases}$$

We then compare the predicted communication time with with the measured time which is obtained by averaging over 100 direct implementation of $cshift$ operation.

In Figure 8.8, we plot the relative errors for performing the *cshift* with different distances along the first axis on the 2-D square data with (:news,:news) layout on the 8K CM-2. The relative errors are quite uniform with respect to the distance and are all less than 20%. Similarly, in Figure 8.9, we plot the same relative errors of the *cshift* operation along the *second* axis on the 2-D square data with (:news,:news) layout on the 8K CM-2. Again the maximum relative errors are less than 20%. But the relative errors are more spread out and centered around zero for different data sizes. In Figure 8.10, we plot the relative errors of the *cshift* operation along the first and second axis on the 3-D data with (:news,:news,:news) layout on the 8K CM-2. The relative errors are still less than 20% in this case. Lastly we do the same comparison on a 16K CM-2 for the 3-D data described above. The relative errors are less than 15% , slightly smaller than the corresponding calculation on a 8K machine.

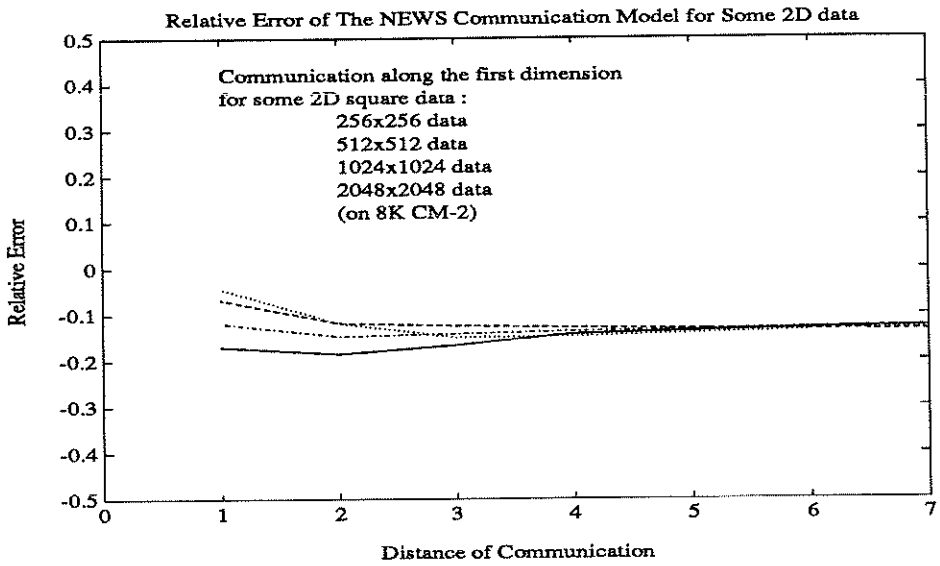


Figure 8.8:

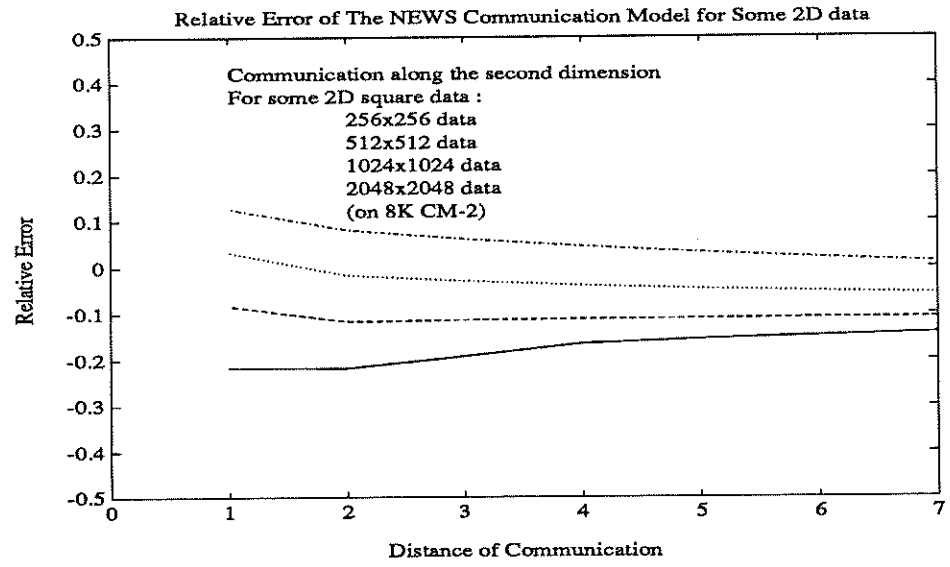


Figure 8.9:

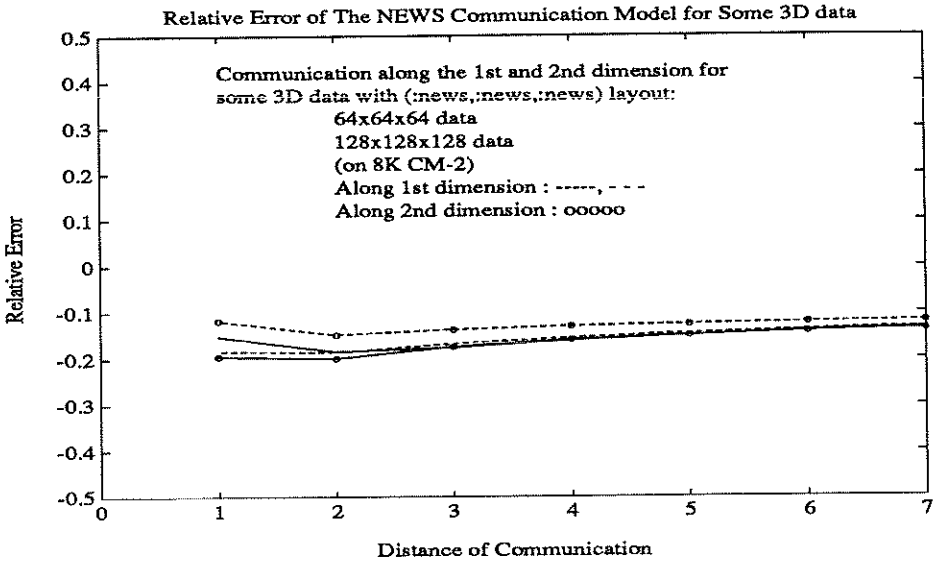


Figure 8.10:

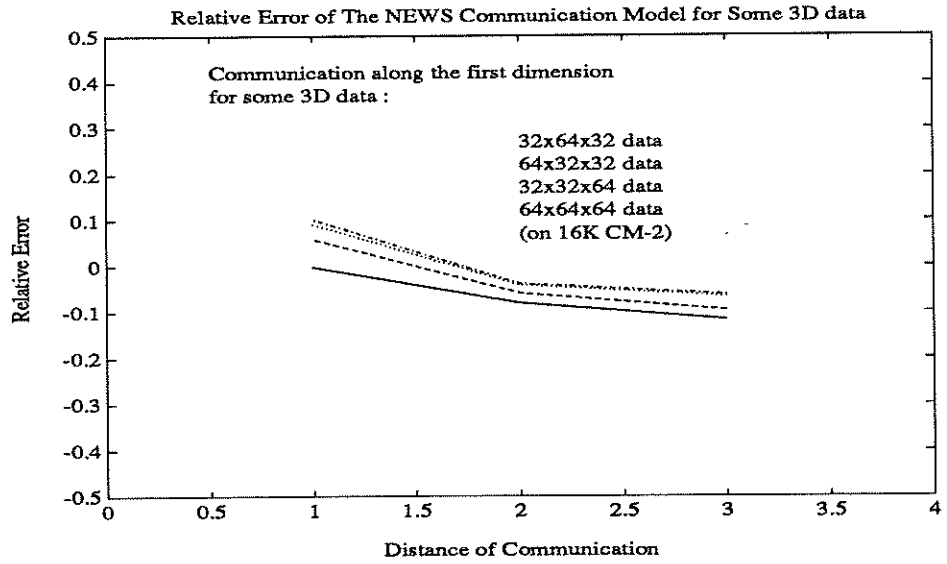


Figure 8.11:

CHAPTER 9

Applications

The model we proposed is designed for the slicewise model of CM-2 . There exist several different kinds of SIMD parallel machines which may have different performance in communication and in the floating point operation due to the difference in the hardware architecture and compilers. From the users' point of view, it would be desirable to have a simple timing model to help them decide which parallel computer is more suitable for his or her particular problem and which computational method is more suitable for certain machine. Such a model should reflect the performance of both arithmetic operation and the communication cost. Of course the user should have some rough knowledge about the arithmetic and communication performance of a SIMD type of machine. We would hope that our model study can provide some useful guideline for the user.

9.1 Predicted Performance of Difference Methods Using the Current CM-2

In this section, we apply our timing model developed in the previous chapter to predict the performance of high order finite difference methods for the 2-D viscous Burger systems using the slicewise model on the current CM-2 machine. In Table

9.1, we list the predicted computation times for several high order finite difference approximations , ranging from the second order difference method to the tenth order method.

Sizes	2nd Order	4th Order	6th Order	8th Order	10th Order
64x64	0.0039	0.0084	0.0139	0.0202	0.0281
128x128	0.0096	0.0192	0.0310	0.0440	0.0608
256x256	0.0286	0.0534	0.0824	0.1121	0.1530
512x512	0.0976	0.1715	0.2537	0.3306	0.4435
1024x1024	0.3593	0.6067	0.8709	1.0966	1.4498

Table 9.1: Performance for Original Timing Model

The ratio of the total times among the second order, fourth order and sixth order is 1:1.76:2.60 in the case of 512×512. This is basically consistent with the corresponding performance measured from the direct calculations, which is $1.000 : 1.752 : 2.479$. In fact the maximum relative error is less than 13% for the second, fourth, and the sixth order methods for various data sizes. This indicates that our timing model is reasonably accurate. Thus using our timing model, one can obtain a fairly reliable prediction on the total computational times of high order difference methods without measuring the computational times directly. Moreover, this works for any order of difference schemes. From this model study, one can obtain the general trend of their timing performance.

9.2 Performance Analysis for Possible Improved Machine Parameters

It is very hard to design one kind of parallel computer to fit general purpose problems with uniform excellent performance. Reasonable performance requires careful marriage of algorithm and architecture. If the architect wants to improve the speed of performance, which factor should he devote to? Should he work on the improvement of start up time, internal communication rate, or external communication rate? We hope through the model study we can reveal which factor is more dominant in affecting the overall performance of a massively parallel machine with a similar architecture to CM-2, in particular for the application in fluid dynamics calculations. In the following sections, we give some predictions under certain reasonable assumptions. We remark that although our model is specific to CM-2, but it should be extensible to other SIMD architecture with some minor modifications.

We are going to vary the parameters in the timing model developed in the previous chapter to see how much it will affect the overall performance. From our previous study in chapter 2 of Part II, we see that the communication time could be as much as the required arithmetic time, or even more in some cases. Thus there is a subtle balance between the arithmetic operation time and the communication time. This phenomena is very different from that of a sequential machine. So it is not clear a priori that a higher order difference method can be more efficient than a lower order method. And the answer may depend on how much one can

improve the speed of arithmetic operation and/or lower the communication cost in a parallel machine.

9.2.1 Predicted Performance of Improved Communication Startup Overhead Time on CM-2

Suppose we can improve the communication startup overhead time by a factor of 10 and keep the other parameters fixed. We obtain the following performance estimates.

Sizes	2nd Order	4th Order	6th Order	8th Order	10th Order
64x64	0.0032	0.0069	0.0117	0.0174	0.0245
128x128	0.0082	0.0165	0.0269	0.0385	0.0539
256x256	0.0259	0.0480	0.0742	0.1012	0.1394
512x512	0.0922	0.1607	0.2376	0.3090	0.4166
1024x1024	0.3486	0.5852	0.8387	1.0536	1.3962

Table 9.2: Overhead Parameters are 10 times faster.

As we can see from a comparison with Table 9.1, the total times do not decrease much, and the ratio of the total time among the second order, fourth order and sixth order for 512×512 case is $1 : 1.68 : 2.40$. The result is similar to the original estimates in Table 8.1. This seems to indicate that the overall performances of the difference methods are not sensitive to the overhead time. This is to be expected since the overhead time is proportional to $N_2 \times N_3$ which is much smaller than the

the total internal communication time ($O(N_1 \times N_2 \times N_3)$.

9.2.2 Predicted Performance of Improved External Communication Time

If we reduce the external communication parameter t_{OUT} by a factor of 10, we obtain the results in Table 9.3.

Sizes	2nd Order	4th Order	6th Order	8th Order	10th Order
64x64	0.0029	0.0054	0.0078	0.0102	0.0130
128x128	0.0076	0.0132	0.0189	0.0239	0.0306
256x256	0.0246	0.0413	0.0582	0.0719	0.0926
512x512	0.0896	0.1473	0.2054	0.2501	0.3227
1024x1024	0.3432	0.5584	0.7743	0.9356	1.2083

Table 9.3: External Communication Parameter is 10 times faster.

The total time does decrease, but not by much. We also see that the reduced time is of ratio 1:3:6 for the 2nd , 4th and the 6th order methods. This is because the external communication time is proportional to the amount of data moving outside the processing node, and does not depend on the geometry. For example, for the second order scheme, we need to do distance 1 communication. To accomplish U_x in CM fortran code, we do:

$$s_1 \times (cshift(u,1,1) - cshift(u,1,-1))$$

which requires one column of 2D data moving out a processing node. For fourth

order scheme, we need to do both distance 1 and 2 communication, in CM fortran code, we do:

$$s_1 \times (cshift(u,1,1) - cshift(u,1,-1)) + s_2 \times (cshift(u,1,2) - cshift(u,1,-2)),$$

which requires one column of 2D data moving out a processing node for distance 1 communication, and two columns of 2D data moving out for distance 2 communication. Totally, it requires 3 times the amount of time in the external communication of fourth order U_x compared with the second order U_x. Similarly, we can deduce the reduced time for sixth order U_x. Therefore the reduced time for external communication among second order, fourth order, and sixth oreder difference operator are roughly of ratio 1:3:6. The sixth order method saves more time when we reduce the external communication parameter, but the actual amount of saving is not substantial.

9.2.3 Predicted Performance of Improved Internal Communication Time

Now suppose we improve the internal communication parameter by a factor of 10. The results are given in Table 9.4.

Sizes	2nd Order	4th Order	6th Order	8th Order	10th Order
64x64	0.0037	0.0081	0.0135	0.0199	0.0278
128x128	0.0087	0.0177	0.0288	0.0414	0.0578
256x256	0.0250	0.0464	0.0723	0.0992	0.1374
512x512	0.0828	0.1422	0.2106	0.2740	0.3740
1024x1024	0.2989	0.4869	0.6926	0.8608	1.1575

Table 9.4: Internal Communication Parameter is 10 times faster.

The total time does decrease, not much but more than the case when we improve the external communication parameter correspondingly. The ratio of the total time among the second order, fourth order and sixth order for 512×512 case is 1:1.71:2.54, which is similar to the unchanged case in table 3.1, (1:1.76:2.60). We also see the saved time is about ratio 1:2:3. Because once the NEWS communication is required, data need to be moved in the internal memory of a processing node, or to be copied to some temporary buffer for external communication. Therefore, every data needs to be moved in memory. And the required internal communication time for second, fourth, and sixth order difference operators have the ratio about 1:2:3. The ratio of the total time among the second order, fourth order and sixth order for 512x512 case is 1:1.71:2.54, which is similar to the unchanged case in table 3.1, (1:1.76:2.60) . We remark that although the sixth order method has improved a lot over the unchanged case, the fourth order method is still more favorable. This is because even if the communication time can be negligible, the

operation time required by the fourth order method is still less than that required by the sixth order method. And we know from our previous study in Part I that the 4th order method can achieve roughly the same error tolerance as the sixth order method with the same number of grid points.

9.2.4 Predicted Performance of Improved Overall Communication Time

If we simultaneously improve the internal and external communication parameters by a factor of 10, we obtain the results in Table 9.5.

Sizes	2nd Order	4th Order	6th Order	8th Order	10th Order
64x64	0.0027	0.0051	0.0075	0.0098	0.0127
128x128	0.0067	0.0116	0.0168	0.0212	0.0277
256x256	0.0210	0.0344	0.0482	0.0589	0.0770
512x512	0.0747	0.1181	0.1623	0.1935	0.2533
1024x1024	0.2828	0.4386	0.5960	0.6998	0.9160

Table 9.5: Both Internal and External Communication Parameter are 10 times faster.

The total times decrease more due to the decrease of both internal and external communication parameters , but the ratio of the total time among the second order, fourth order and sixth order is 1:1.58:2.17 which is still not too far away from the original one , 1:1.76:2.60.

9.2.5 Predicted Performance of Improved Floating Point Operation Time

Next, we look at the case where we improve the floating point operation parameter by a factor of 10. The predicted performance of CM-2 is given in Table 9.6.

Sizes	2nd Order	4th Order	6th Order	8th Order	10th Order
64x64	0.0030	0.0070	0.0121	0.0182	0.0253
128x128	0.0060	0.0139	0.0239	0.0358	0.0501
256x256	0.0144	0.0321	0.0540	0.0797	0.1103
512x512	0.0407	0.0865	0.1406	0.2017	0.2740
1024x1024	0.1316	0.2670	0.4191	0.5824	0.7735

Table 9.6: Floating Point Operation Parameter is 10 times faster.

The total times in this case decrease much more than all the previous cases considered. The reduced times are proportional to the amount of the total amount of floating point operations, 1.85:2.85:3.81.

Why does the improvement of the floating point operation have more effect than the improvement of both internal and external communication parameters? This is because the floating point operation time tends to dominate the whole performance when the VP ratio is high. This can also be seen from the communication and arithmetic cost for the unchanged case of Table 9.1 given Tables 9.7 and 9.8 respectively.

Sizes	2nd Order	4th Order	6th Order	8th Order	10th Order
64x64	0.0029	0.0068	0.0118	0.0179	0.0250
128x128	0.0055	0.0131	0.0228	0.0346	0.0484
256x256	0.0123	0.0287	0.0494	0.0743	0.1034
512x512	0.0322	0.0729	0.1219	0.1794	0.2452
1024x1024	0.0977	0.2122	0.3436	0.4917	0.6567

Table 9.7: Communication Cost for Original Timing Model

Sizes	2nd Order	4th Order	6th Order	8th Order	10th Order
64x64	0.0010	0.0015	0.0021	0.0024	0.0031
128x128	0.0041	0.0062	0.0082	0.0095	0.0124
256x256	0.0163	0.0247	0.0330	0.0378	0.0496
512x512	0.0654	0.0986	0.1318	0.1512	0.1983
1024x1024	0.2616	0.3945	0.5273	0.6049	0.7931

Table 9.8: Arithmetic Cost for Original Timing Model

From Tables 9.7 and 9.8, we see that the arithmetic time is more than the communication time for larger data sizes. The communication time becomes dominant only when the data sizes are small. The crossover point varies depending on the order of the scheme. For example, the arithmetic operation time for the second order method begins to dominate the total computational time for data size larger than 128×128. In comparison, the arithmetic operation time for the sixth order

method begins to dominate the total computational time only for data size larger than 256×256. We illustrate this behaviour in Figure 9.1 for the sixth order difference method as an example. If we scale down the arithmetic time by 10, the total time becomes dominated by the communication for the case of large data sizes. This is why scaling down the floating point parameter gives more improvement than scaling down the parameters of communication.

One interesting observation is that when we scale down the arithmetic parameter by 10, the second order scheme with twice as many data takes about the same time as the sixth order method. Recall that in Part I, we found that the second order method with twice as many grid points can achieve the same error tolerance as sixth order method for the inviscid hyperbolic equations ($\alpha = 1.2$). Should we choose the simpler second order scheme in this case? Well, twice as many grid points require twice as much memory space. Therefore, we still prefer to use the fourth order scheme, which only needs roughly as many grid points as the sixth order scheme to achieve 2.5% error tolerance and is faster than second order scheme with twice as many data.

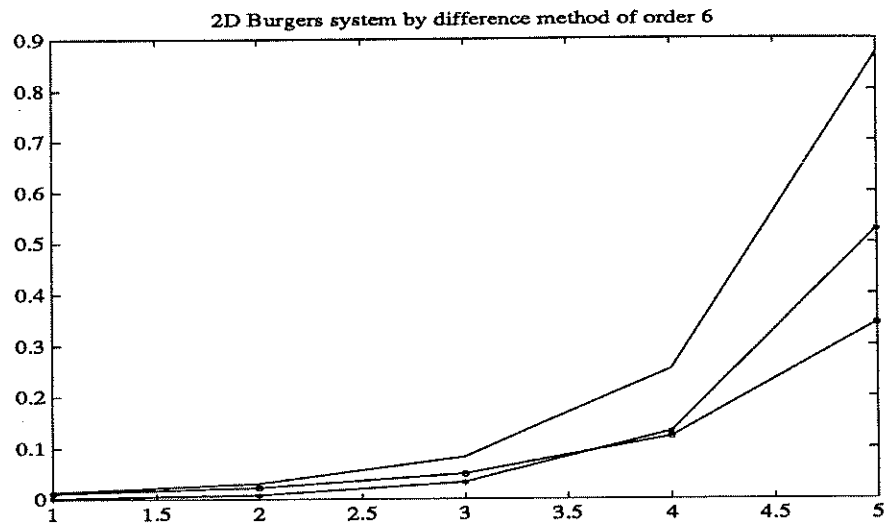


Figure 9.1: Total time: —, Communication: ooo , Arithmetic: ***

9.3 Asymptotic Behaviour for Various Improved Timing Parameters

It is of fundamental interests to understand the dependence of performance on various timing parameters. Here we test our model by systematically reducing those parameters by a factor of 5, 10, 30, 60 and 100. We plot the resulting performances in Figures 9.2, 9.3, and 9.4 for the second, the fourth, and the sixth order methods when reducing the total communication times. As we can see each curve reaches to an asymptote very quickly as we reduce the total communication parameters. In fact, there is little change after we reduce the communication parameter by a factor of 30 for various data sizes. In the extreme case when the total communication time is equal to zero, the total computational time is governed only by the arithmetic operation time. This is the same as in a sequential machine. But as we know from Table 9.8, the second order method with twice as many grid

points is more expensive than the sixth order method. So the fourth order method is preferable. The behaviour for reducing the arithmetic operation parameters is very similar. We plot the resulting performances in Figures 9.5, 9.6, and 9.7 for the second, the fourth, and the sixth order methods when reducing the arithmetic times. Again, each curve reaches to an asymptote as we reduce the arithmetic parameters. In the extreme case when we set the arithmetic time to zero, the total computational time is governed by communication. From Table 9.7, we know that the execution time required for the second order method with twice as many grid points is comparable to that of the sixth order method. So the simple second order method with twice as many grid points is competitive with the sixth order method in achieving the same error tolerance, as long as the memory storage is not a concern. As before, the fourth order method is still the true winner since it can achieve roughly the same error tolerance with the same number of grid points as the sixth order method, and it is faster than the second order method with twice as many grid points.

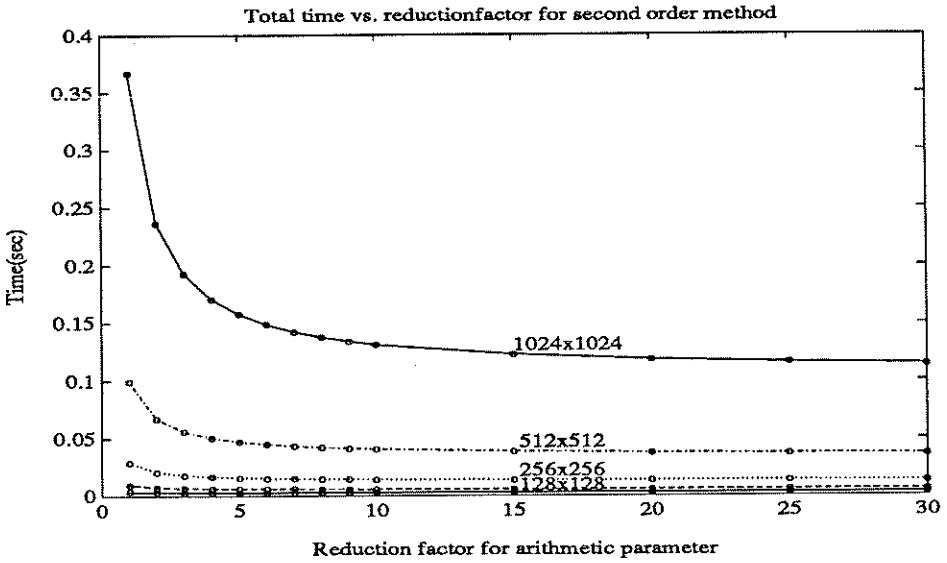


Figure 9.2:

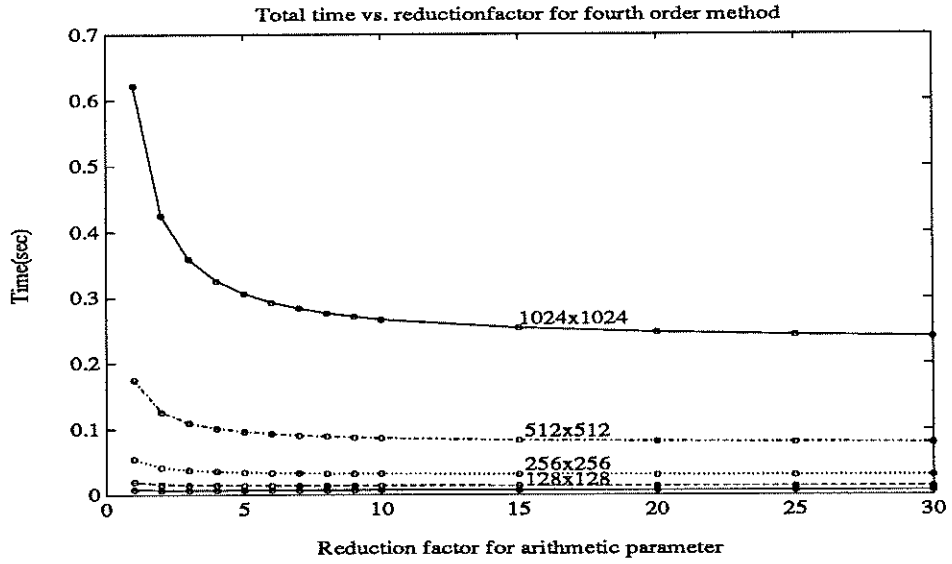


Figure 9.3:

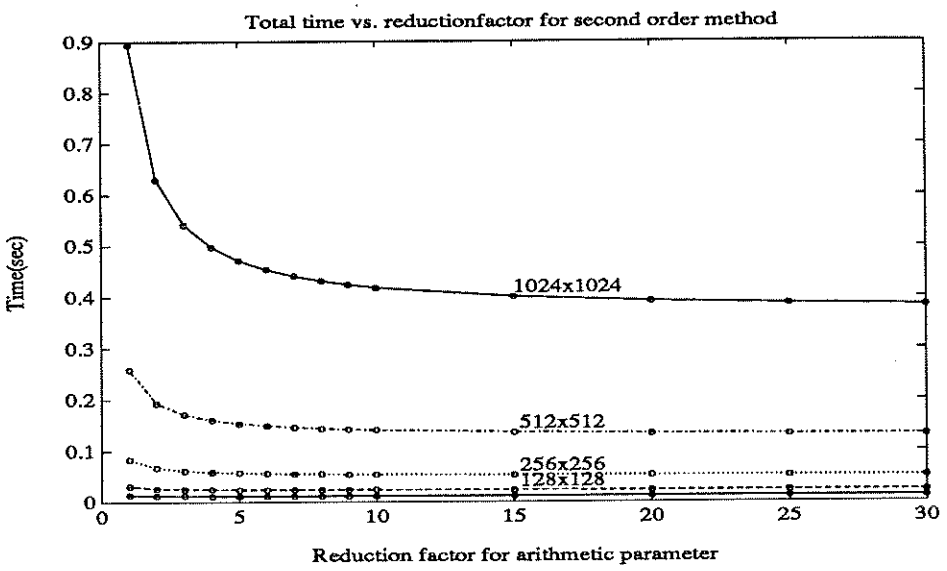


Figure 9.4:

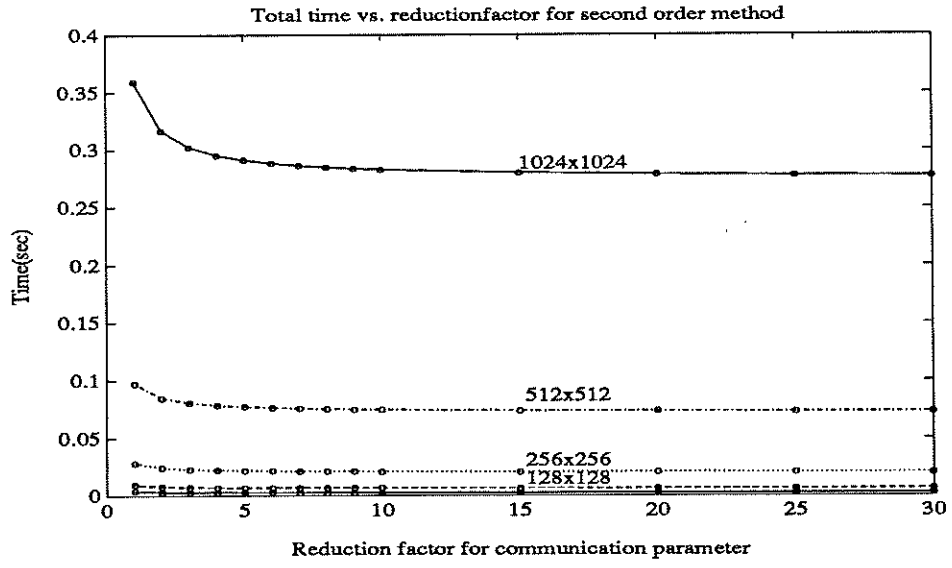


Figure 9.5:

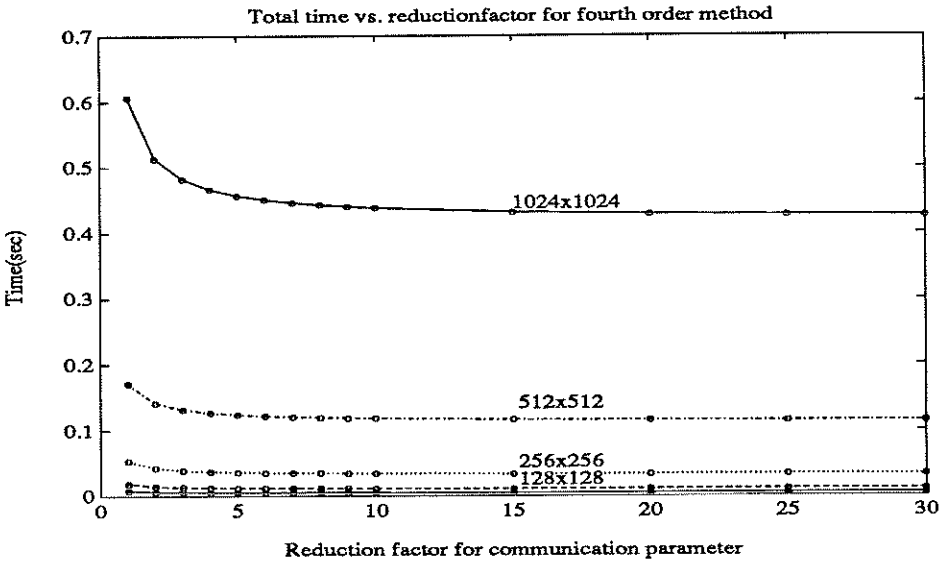


Figure 9.6:

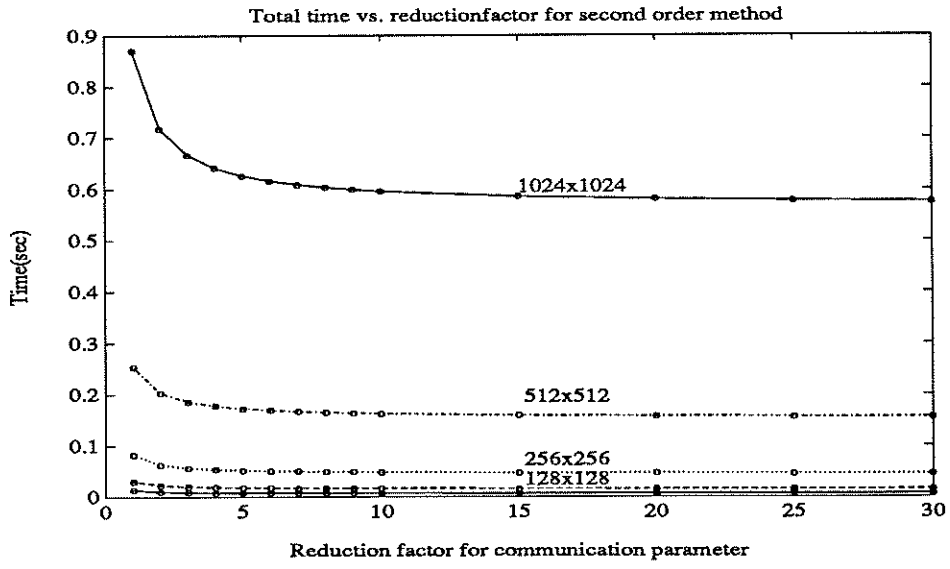


Figure 9.7:

9.4 Concluding Remarks

Our timing model is quite stable subject to various perturbations in the parameters. Our study indicates that the fourth order scheme gives the best overall performance on the current CM-2 for the 2D Burgers' system. This is based on the consideration of computational efficiency and the memory storage. This conclusion remains valid when we improve over the current CM-2 machine, such as reducing the communication cost by a factor of 10 or improving the arithmetic operation speed.

We also found that under the current CM-2 architecture the communication factor dominates the arithmetic operation when the data size is small. But for large data sizes the arithmetic operation tends to dominate the calculation. Only when improving the arithmetic speed substantially can we achieve a significant speed up in the overall computational performance. In other words, the communication is not as bad as one might think for these kinds of problems on the CM-2.

Can a lower order like the second order method becomes favorable over the higher order methods for a future generation of parallel computer? The answer is no if the future machine still keeps the similar architecture as the current CM-2 machine. Our study shows that if the arithmetic operation can be substantially improved, the second order method becomes very competitive with the higher order methods as far as the computational efficiency is concerned. However the price we pay is to use more grid points (e.g. twice as many as required for a sixth

order scheme). When memory storage is a concern, a higher order method is still more preferable. On the other hand, a lower order is easier to implement and more flexible to mesh refinement and treatment of boundary conditions. These considerations may lead to the choice of a lower order method. Moreover, it is likely that there will be more special purpose machine available in the future. Then special architecture could potentially lead to a substantial improvement in the overall performance for certain class of applications.

Bibliography

[1] J. W. Cooley and J. W. Tukey, "An algorithm for the machine computation of complex Fourier series," Math. Comp., Vol. 19, 1965, pp. 297-301.

[2] S. Duggirala, Thinking Machine Corp., private communication.

[3] B. Fornberg, "The Pseudo-Spectral Method: Comparisons with Finite Differences for the Elastic Wave Equation," GeoPhysics, Vol. 52, No. 4, 1987, pp. 483-501.

[4] J. Goodman, T. Y. Hou and E. Tadmor, "Weak Numerical Instabilities of Pseudo-Spectral Methods without Smoothing," in preparation.

[5] D. Gottlieb and S. A. Orszag, "Numerical Analysis of spectral Methods: Theory and Applications," CBMS-NSF Regional Conference Series in Applied Mathematics 26, Society for Industrial and Applied Mathematics, Philadelphia, 1977.

[6] D. Gottlieb, S. A. Orszag and E. Turkel, "Stability of Pseudospectral and Finite-Difference Methods for Variable Coefficient Problems," Math. Comp., Vol. 37, No.156, 1981, pp. 293-305.

[7] A. Greenburg, Thinking Machine Corp., private communication.

[8] R. W. Hockney and C. R. Jesshope, **Parallel Computers**, Adam Hilger Ltd, Bristol, U.K., 1981.

[9] L. S. Johnsson, "Communication Efficient Basic Linear Algebra Computations on Hypercube Architecture," J. Parallel and Distrib. Comput., Vol. 4, 1987, pp. 133-172.

[10] H.-O. Kreiss, "Comparison of Accurate Methods for the Integration of Hyperbolic Equations", Tellus XXIV (1972), Vol. 3, pp. 199-215.

[11] G. L. Browning and H.-O. Kreiss, "Comparison of Numerical Methods for the Calculation of Two-Dimensional Turbulence," Math. Comp., Vol. 52, 1989, pp. 369-388.

[12] W. D. Henshaw, H.-O. Kreiss and L. G. Reyna, "On the Smallest Scale for the Incompressible Navier-Stokes Equations," ICASE Report No. 88-8, 1988.

[13] C. Levit, "Grid Communication on the Connection Machine: Analysis, Performance, and Improvements," in Proceedings of the Conference on Scientific Applications of the Connection Machine, H. D. Simon, Editor, NASA Ames Research Center, California, World Scientific Publ., September, 1988, pp. 316-332.

[14] K. Mathur, Thinking Machine Corp., private communication.

[15] R. Pozo, "Performance Modeling of Parallel Architectures for Scientific Computing," Ph.D. thesis, Dept. of Computer Science, University of Colorado,

Boulder, 1991.

[16] L. F. Richardson, "The Deferred Approach to the Limit, I-Single Lattice," Trans. Roy. Soc., London, Vol. 226, 1927, pp. 299-349.

[17] G. Strang, "Accurate Partial Differential Methods II. Non-linear Problems," Numerische Mathematik, **6**, 37-46 (1964).

[18] E. Tadmor, "Stability Analysis of Finite Difference, Pseudo-spectral and Fourier-Galerkin Approximations for Time-dependent Problems," SIAM Review, Vol. 29, 1987, pp. 525-555.

[19] Thinking Machine Corp., **CM Fortran Optimization Notes: Slicewise Model, Version 1.0** Thinking Machine Corporation, Cambridge, Massachisetts, March, 1991.

[20] Thinking Machine Corp., **CM Fortran Reference Manual, Version 1.0,** Thinking Machine Corporation, Cambridge, Massachisetts, February, 1991.